# 1 壳分类

## 第一代壳 Dex加密

1.Dex字符串加密

2.资源加密

3.对抗反编译

4.反调试

5.自定义DexClassLoader

## 第二代壳 Dex抽取与So加固

1.对抗第一代壳常见的脱壳法

2.Dex Method代码抽取到外部（通常企业版）

3.Dex动态加载

4.So加密

## 第三代壳 Dex动态解密与So混淆

1.Dex Method代码动态解密

2.So代码膨胀混淆

3.对抗之前出现的所有脱壳法

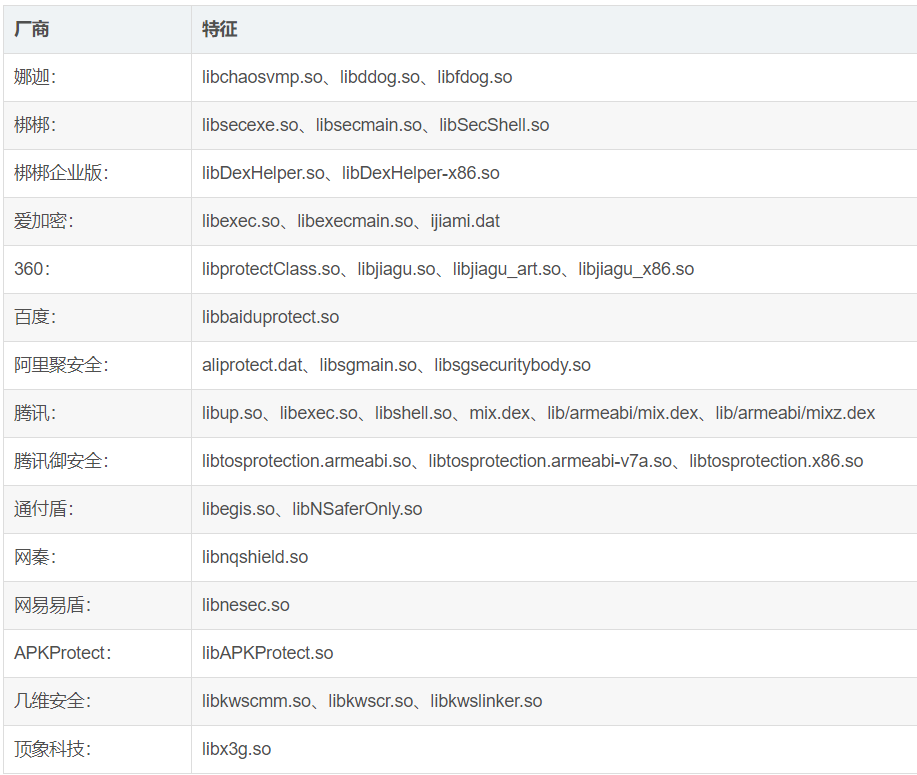
## 第四代壳 arm vmp（未来）

vmp

# 2 壳的识别

## 2.1 加固厂商特征

* 娜迦： libchaosvmp.so , libddog.solibfdog.so
* 爱加密：libexec.so, libexecmain.so
* 梆梆： libsecexe.so, libsecmain.so , libDexHelper.so
* 360：libprotectClass.so, libjiagu.so
* 通付盾：libegis.so
* 网秦：libnqshield.so
* 百度：libbaiduprotect.so



## 2.2 识别代码



# 3 第一代壳

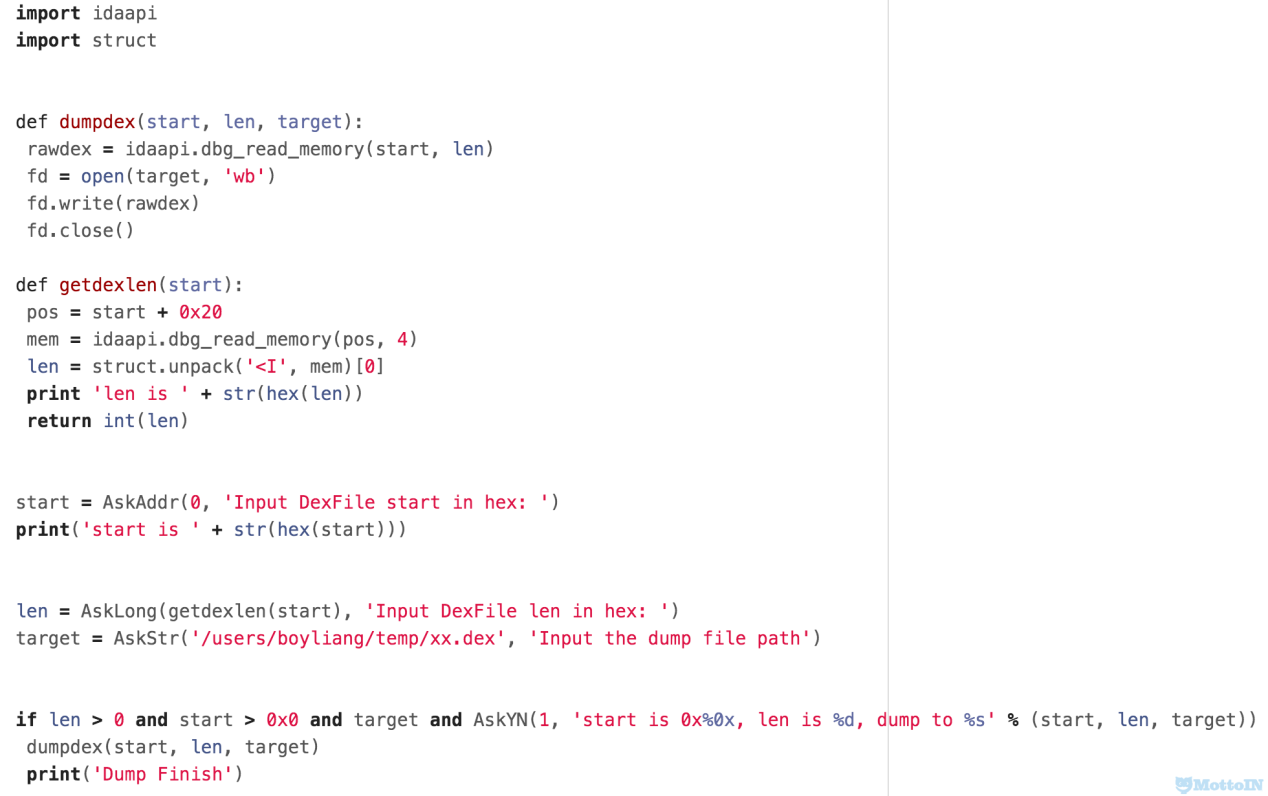
1. 内存Dump法
2. 文件监视法
3. Hook法
4. 定制系统
5. 动态调试法

## 3.1 内存Dump法

### 内存中寻找dex.035或者dex.036

### 手动Dump

/proc/xxx/maps中查找后

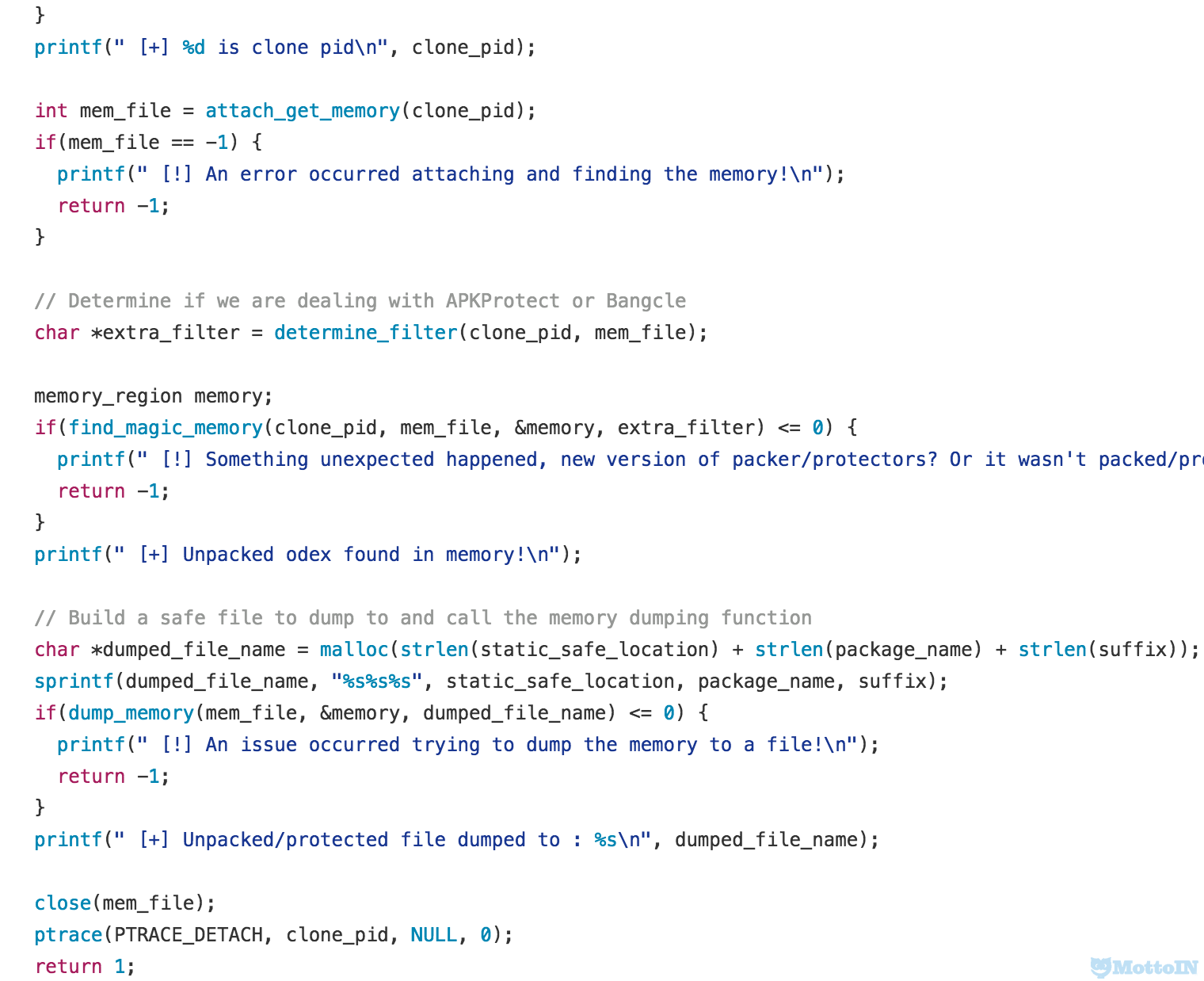


### android-unpacker工具

https://github.com/strazzere/android-unpacker

**（最后更新时间 2020年）**

该工具的核心是通过find\_magic\_memory()方法读取/proc/pid/maps内存映射表，找到DEX所在的内存起始位置，然后通过dump\_memory()方法将内存dump下来



### drizzleDumper工具

https://github.com/DrizzleRisk/drizzleDumper

升级版的android-unpacker，read和lseek64代替pread，匹配dex代替匹配odex

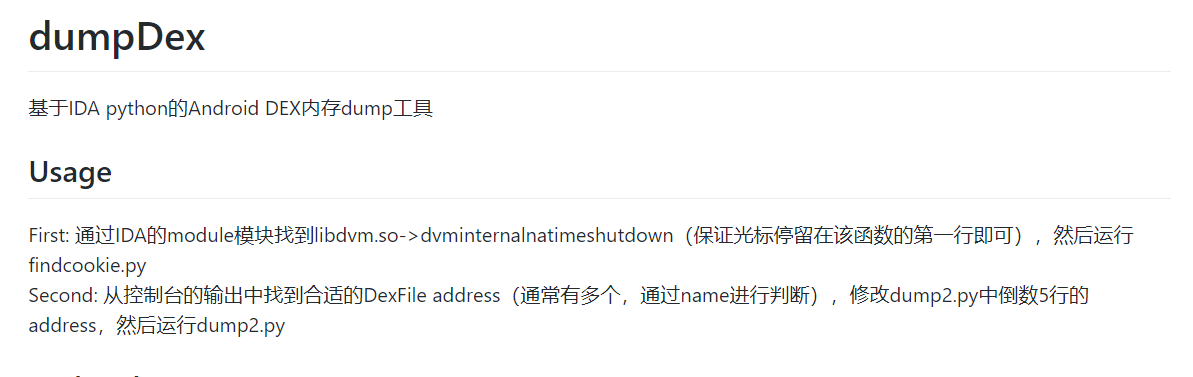
**（最后更新时间 2017年）**



### IDA Pro + dumpDEX

dumpDex工具 https://github.com/CvvT/dumpDex

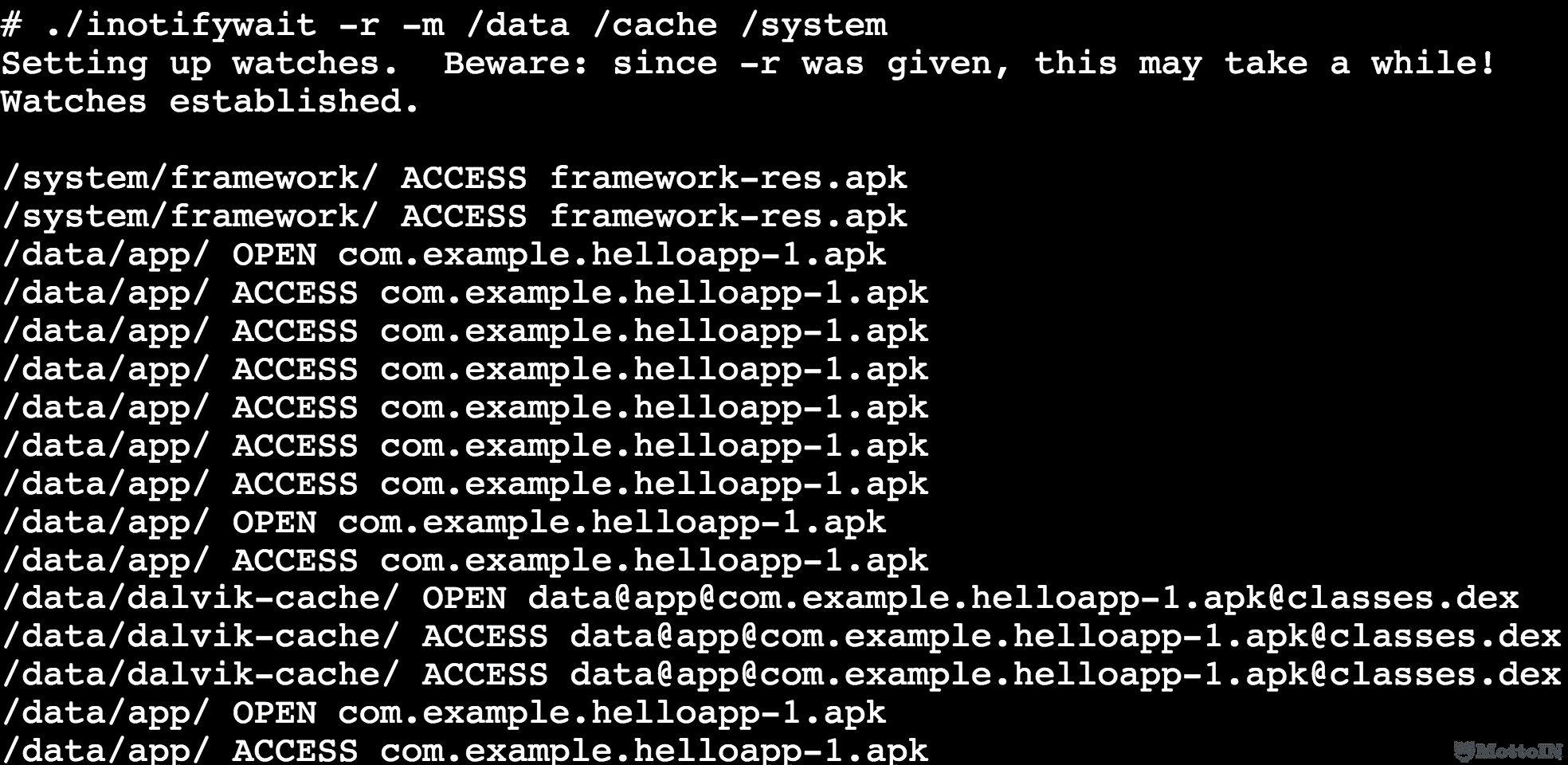
**（最后更新时间 2016年）**

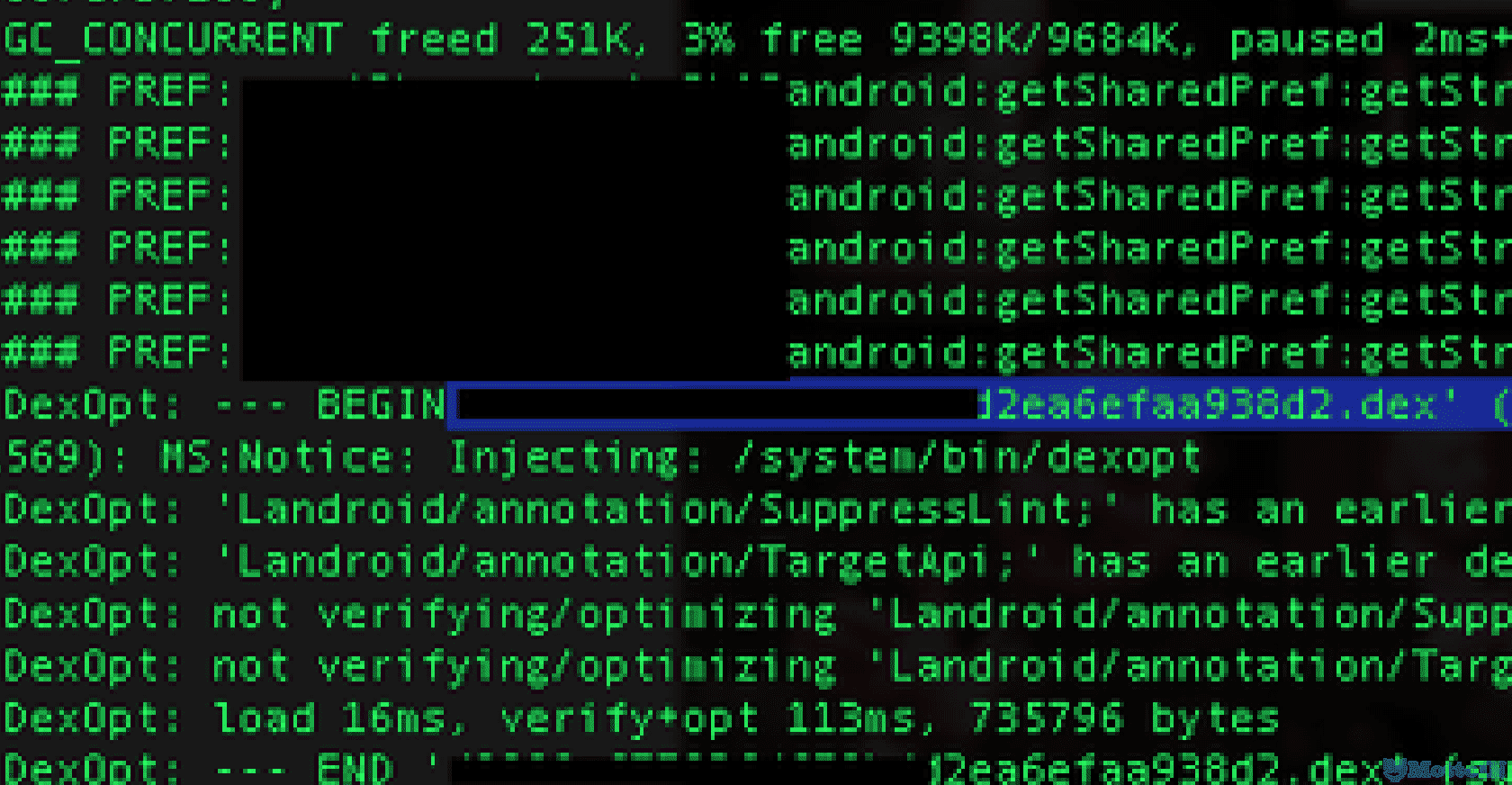


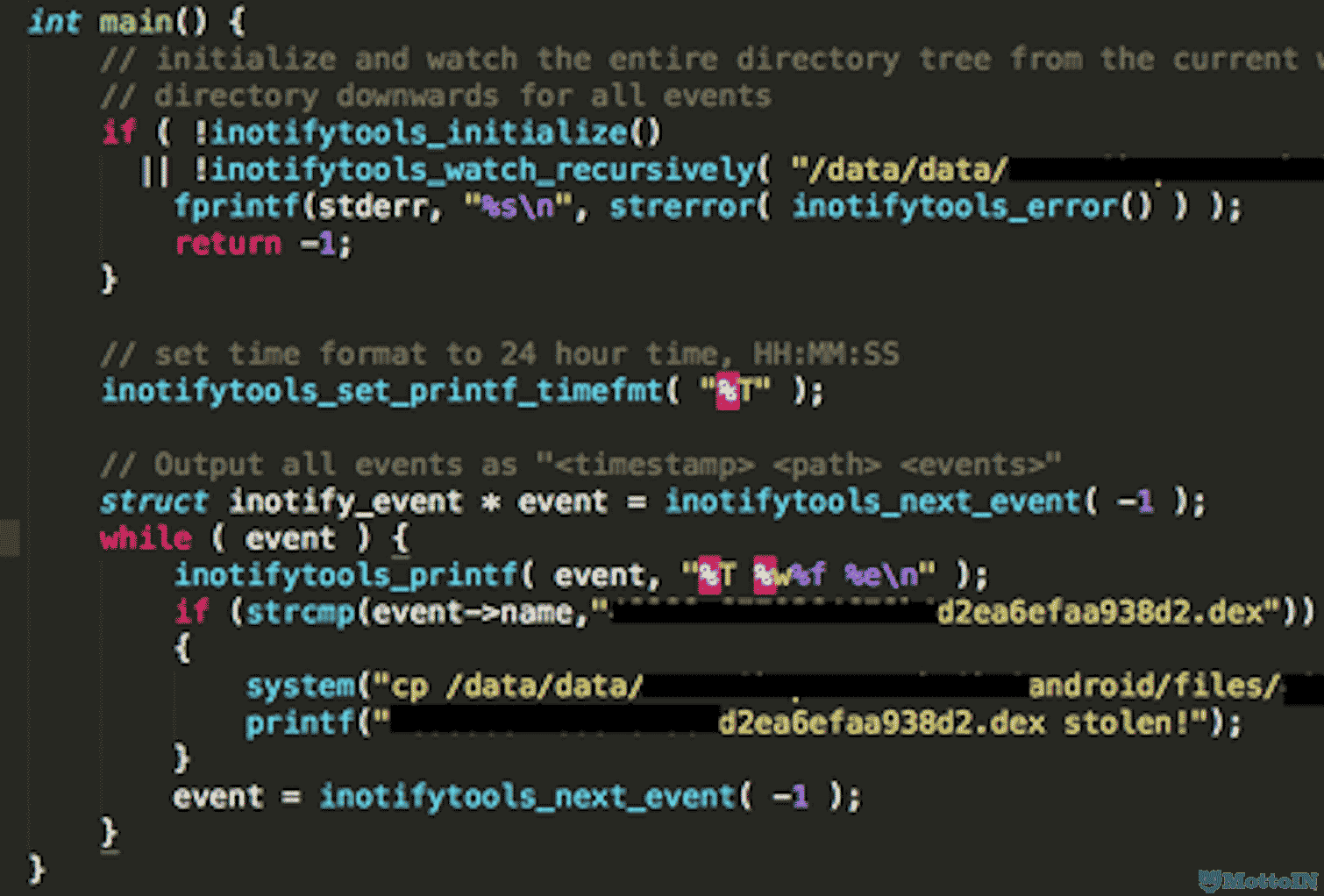
## 3.2 文件监视法

* inotifywait-for-Android https://github.com/mkttanabe/inotifywait-for-Android

Dex优化生成odex，监视文件变化







## 3.3 Hook法（存疑）

Hook dvmDexFileOpenPartial

http://androidxref.com/4.4\_r1/xref/dalvik/vm/DvmDex.cpp

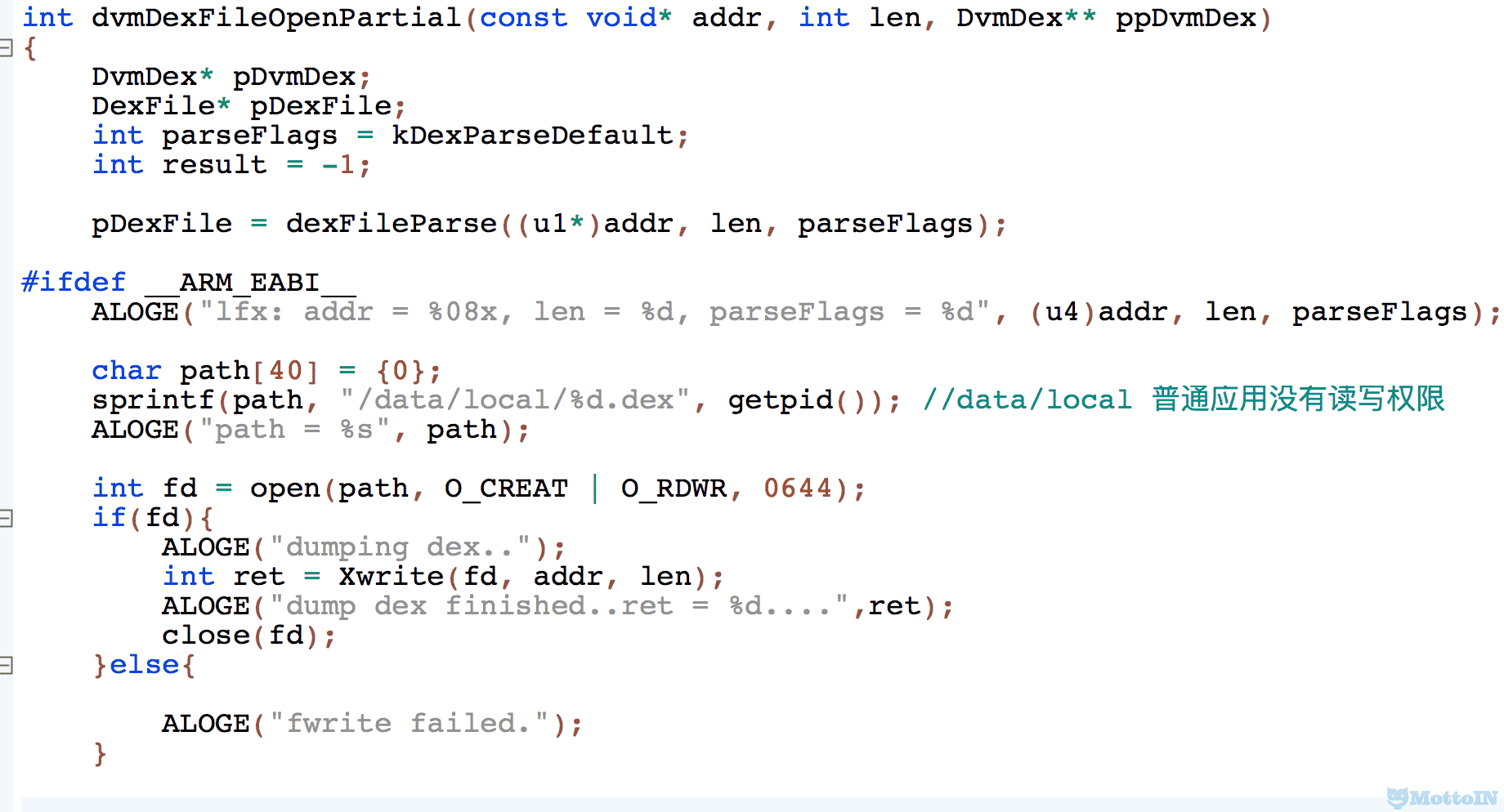
该方法使用HOOK框架，配合HOOK代码（需要找到合适的脱壳时机），实现工具的自动化脱壳。

主要的下断函数：**dvmDexFileOpenPartial()**

## 3.4 定制系统（存疑）

* 修改安卓源码并刷机

针对第一代壳在dvmDexFileOpenPartial()、dexFileParse()方法处设置断点来脱壳的特点，修改它们在源码中的实现，然后编译修改后的代码，刷机形成定制的系统并运行，以此实现脱壳。

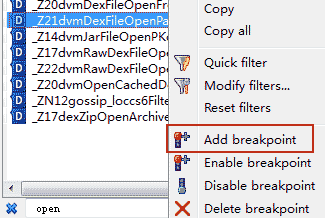


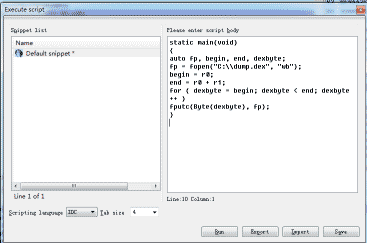
* DumpApk 工具

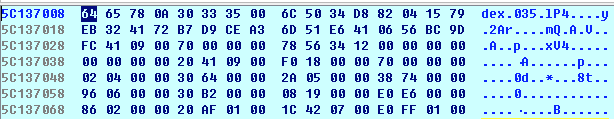
https://github.com/CvvT/DumpApk只针对部分壳

**（最后更新时间2015年）**

## 3.5 动态调试







本质还是内存dump脱壳法，需要通过调试器找到合适的dump时机，即DEX文件已经在内存中完全解密，且其中的代码还没有开始执行。寻找合适的Dump时机是动态调试脱壳法的重点。

主要在于分析AOSP源码文件：dalvik/dexopt/OptMain.cpp中

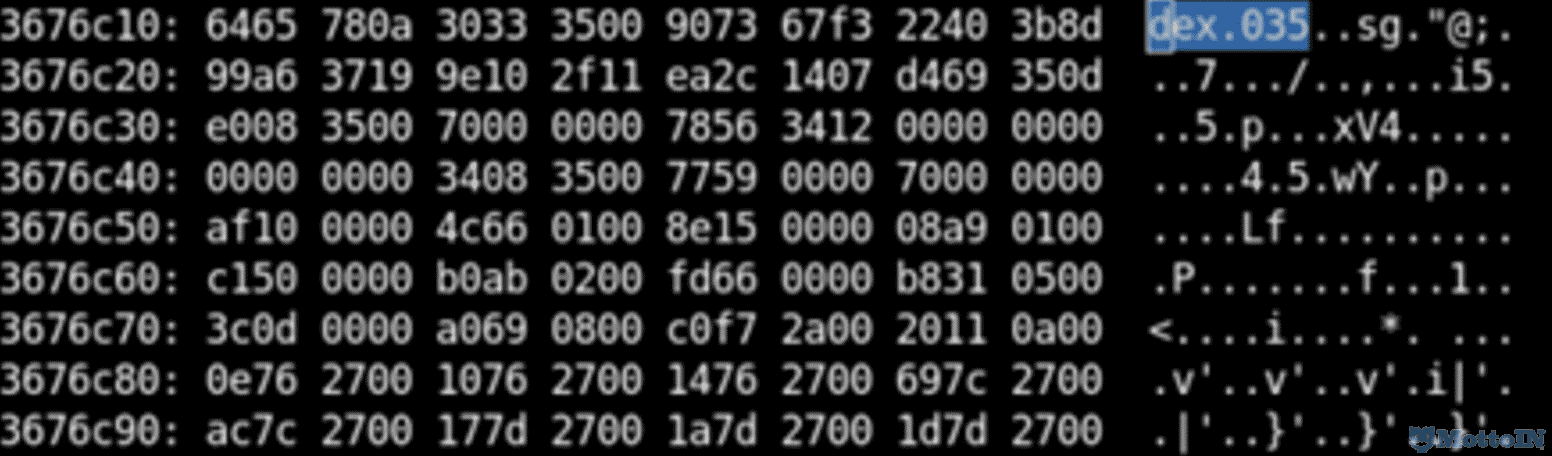
主流的设断点的方法有dvmDexFileOpenPartial()、dexFileParse()方法，在该方法处设置断点，当执行到该断点时，使用内存dump脚本将其dump下来，即可完成脱壳操作。

动态调试脱壳与DEX加载代码的处理方式有关。

## 3.6 gdb gcore法（存疑）

.gdbserver :1234 –attach pid .gdb (gdb) target remote :1234 (gdb) gcore

coredump文件中搜索“dex.035”



## 3.7 缓存脱壳法

动态加载型壳使用DexClassLoader这种方式将加密后的DEX文件在内存中解密后进行动态加载，但由于一些软件壳并没有处理DEX优化时缓存的路径，最终使得系统执行dexopt命令对加载的DEX文件进行优化时，将优化结果放到了默认的/data/dalvik-cache目录下。

解密时不需要做任何额外的工作，只需要将/data/dalvik-cache目录下的ODEX文件取出，进行一次deodex操作，即可完成脱壳工作。

# 4 第二代壳

1. 内存重组法
2. Hook法
3. 动态调试
4. 定制系统
5. 静态脱壳机

## 4.1 内存重组法

代码抽取壳特征：将DEX文件的DexCode提取后填0，将DEX文件的所有内容保存于APK文件中，当APK运行时，会在内存中进行动态解密，所有解密的方法内容指针都位于DEX文件结构体外部的内存中，从而有效避免了在只知道DEX文件的起始地址的情况下就可以快速进行dump的问题。

脱壳方法：解析内存中DEX文件的格式，将其重新组合成DEX文件，可以实现100%的DEX代码还原。

### Dex篇（ZjDroid工具）

对付一切内存中完整的dex，包括壳与动态加载的jar

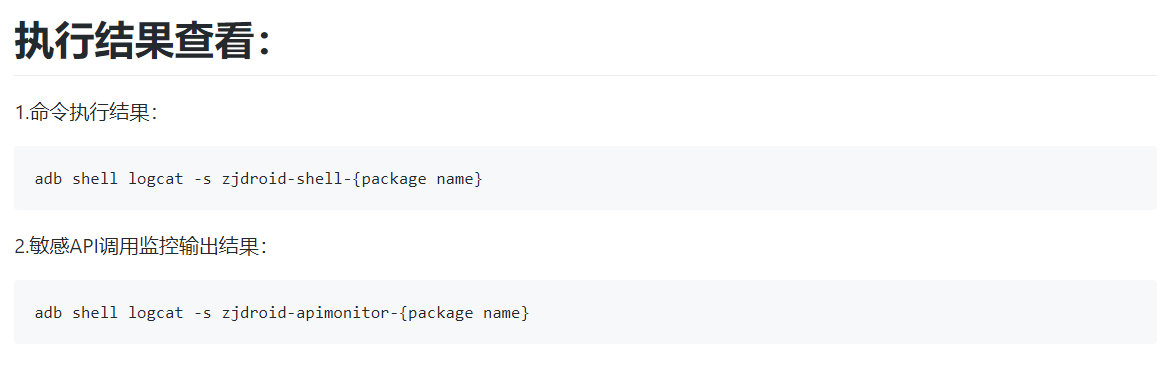
http://bbs.pediy.com/showthread.php?t=190494

工具地址https://github.com/KB5201314/ZjDroid（经整合改进）





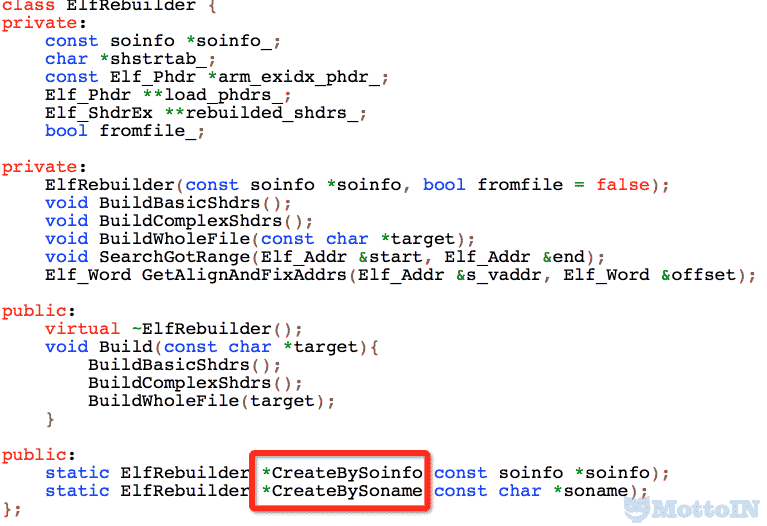


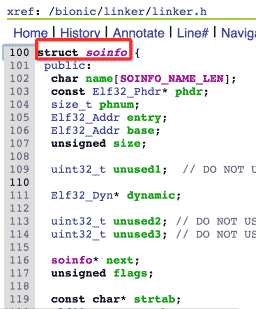


### so篇（ElfRebuild工具）

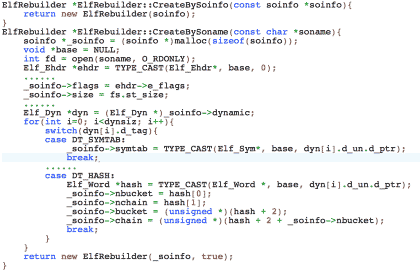
https://github.com/ylcangel/ElfRebuild

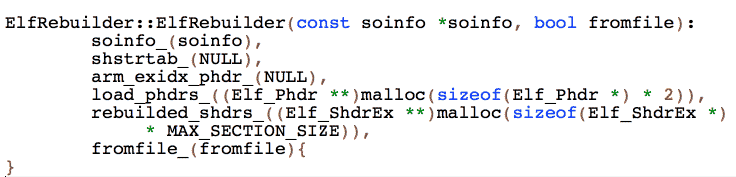






构造soinfo，然后对其进行重建

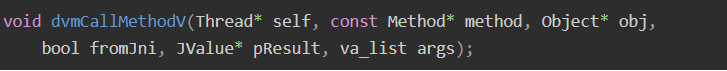




## 4.2 Hook法

合适的HOOK点是libdvm.so中的dvmCallMethodV()方法。APK启动时，会首先执行它的Application类的onCreate()方法。Dalvik虚拟机时通过dvmCallMethodV()方法来启动Java方法的，它的实现位于AOSP源代码的dalvik/vm/interp/Stack.cpp中。

函数如下：



对第二个Method类型的method参数，可以通过它的name字段来判断当前执行的方法名。当确定当前的方法是onCreate()时，可以进一步判断方法所在的类的名字，从而确定其是否为脱壳的目标。

获取ClassObject类型的类对象指针后，可以通过它的pDvmDex字段获取内存重组脱壳法所使用的DvmDex结构体的信息，接下来的DEX内存重组步骤就和内存重组脱壳法一样。

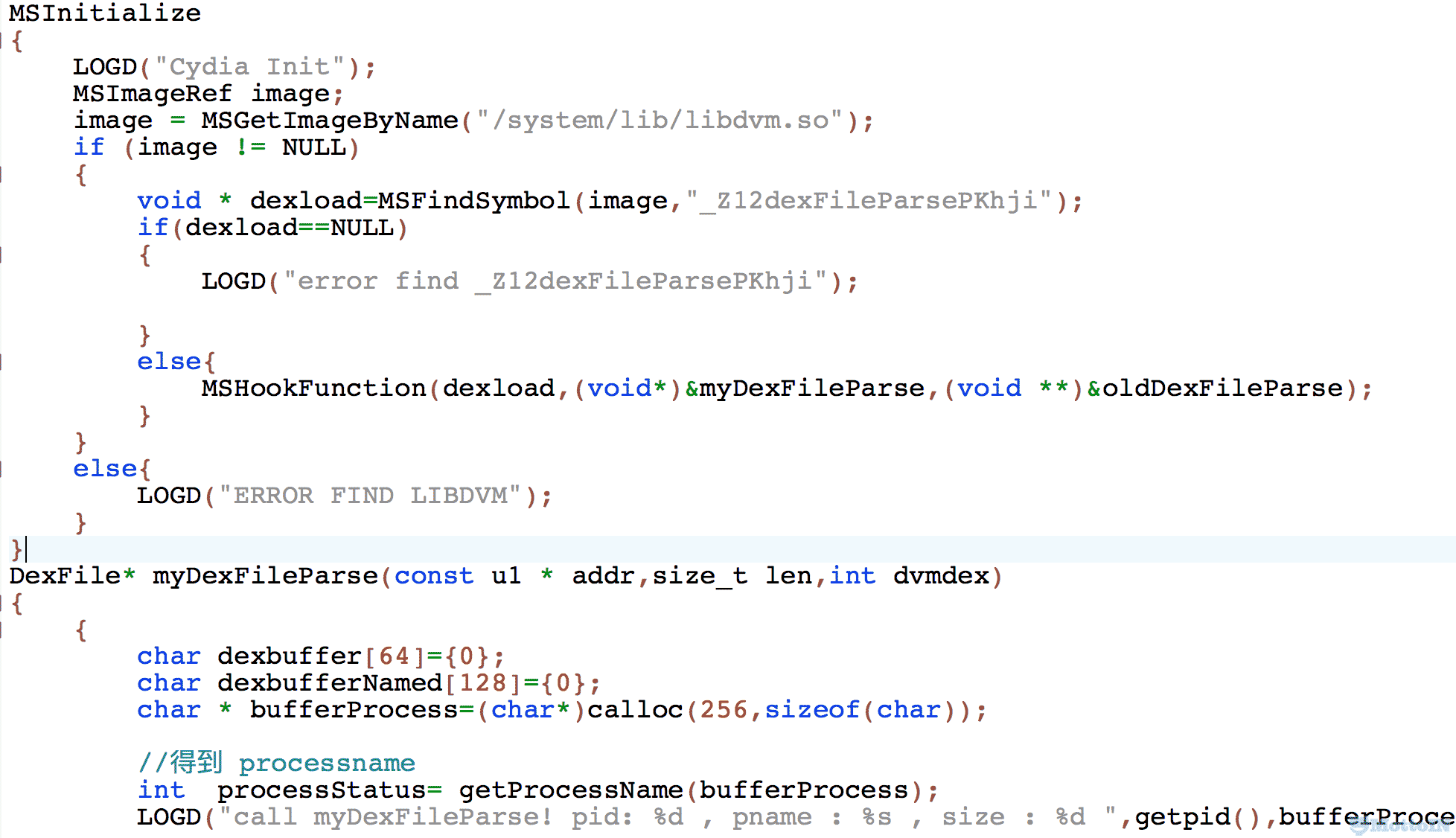
### DumpDex工具（基于Cydia框架）

工具地址：https://github.com/WooyunDota/DumpDex

文档地址：

https://github.com/WooyunDota/DroidDrops/blob/master/2015/Android.Hook%E6%A1%86%E6%9E%B6Cydia%E7%AF%87(%E8%84%B1%E5%A3%B3%E6%9C%BA%E5%88%B6%E4%BD%9C).md

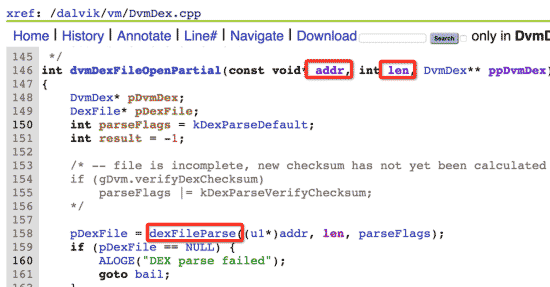




### 针对无代码抽取且Hook dvmDexFileOpenPartial失败

Hook dexFileParse

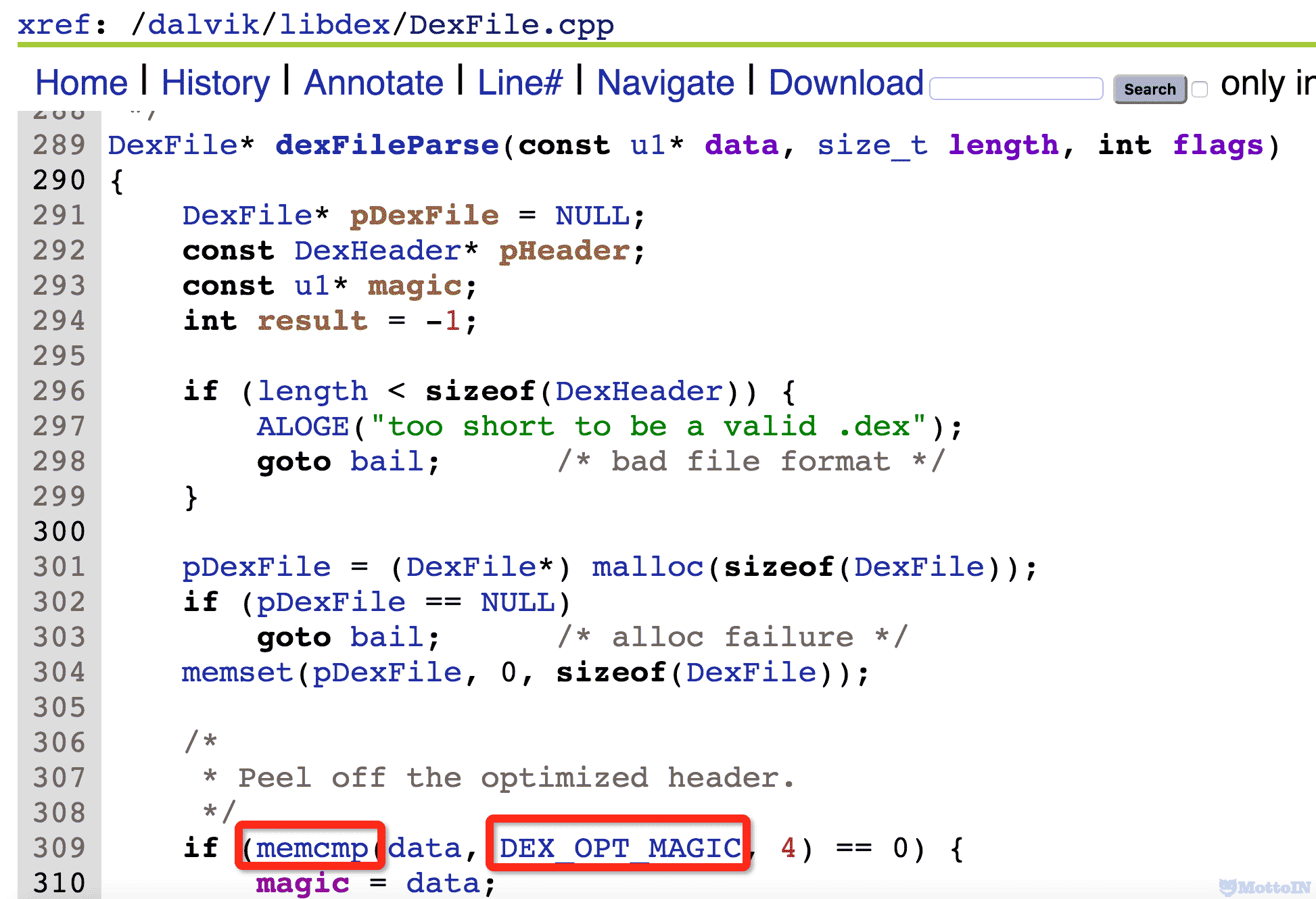
http://androidxref.com/4.4\_r1/xref/dalvik/vm/DvmDex.cpp

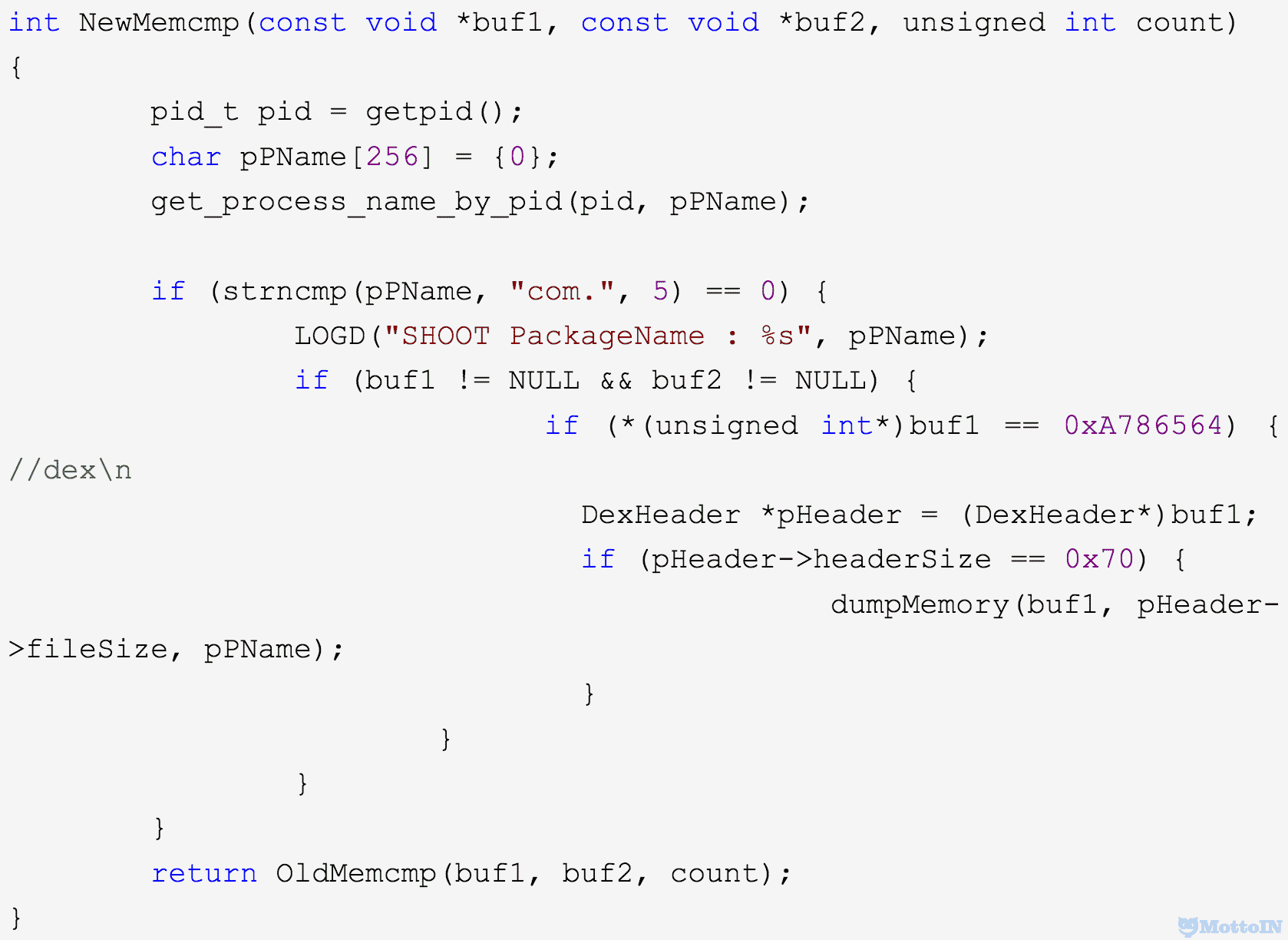


### 针对无代码抽取且Hook dexFileParse失败

Hook memcmp

http://androidxref.com/4.4\_r1/xref/dalvik/vm/DvmDex.cpp





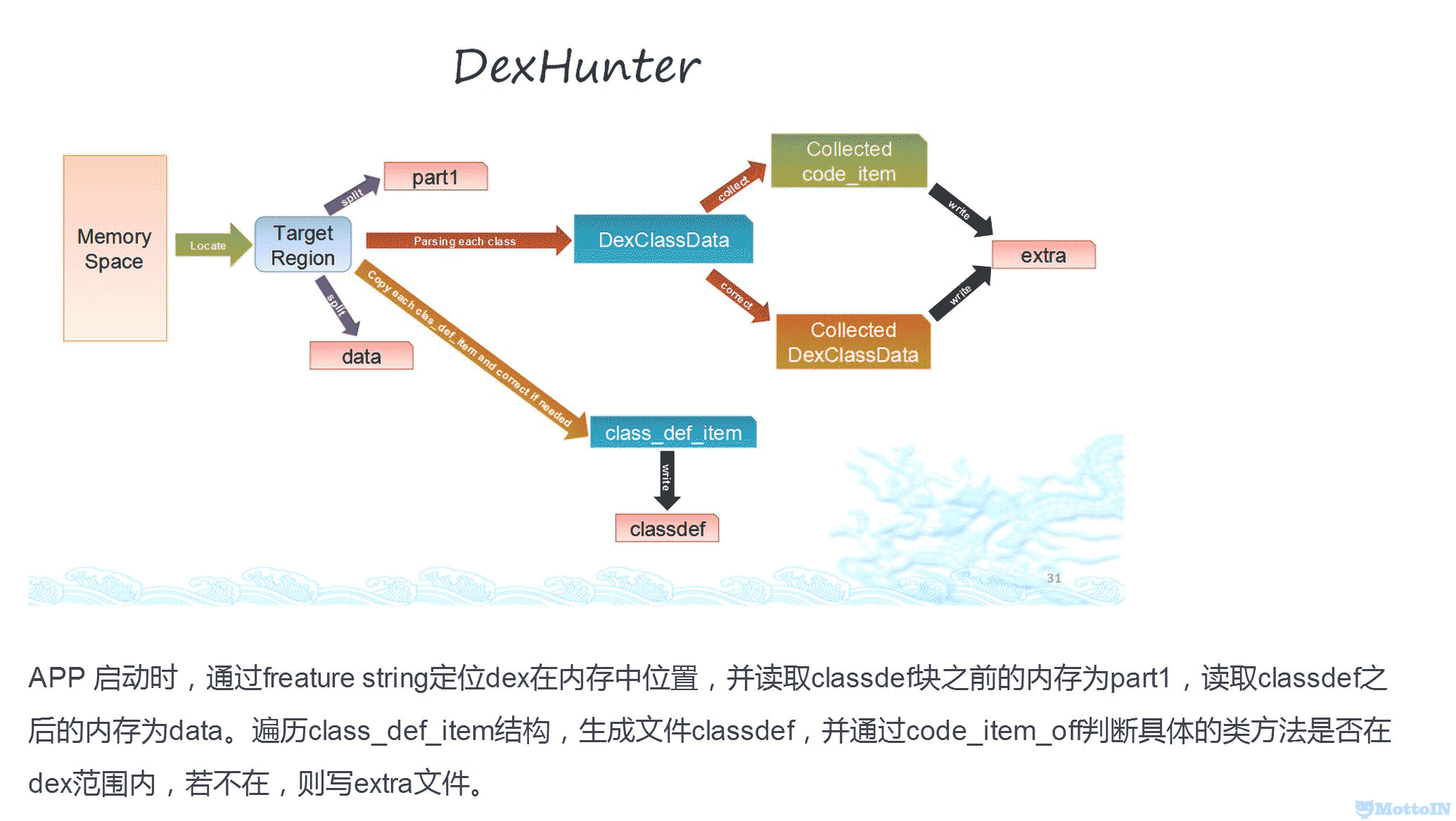
## 4.3 定制系统

### DexHunter工具（最强大的二代壳脱壳工具）

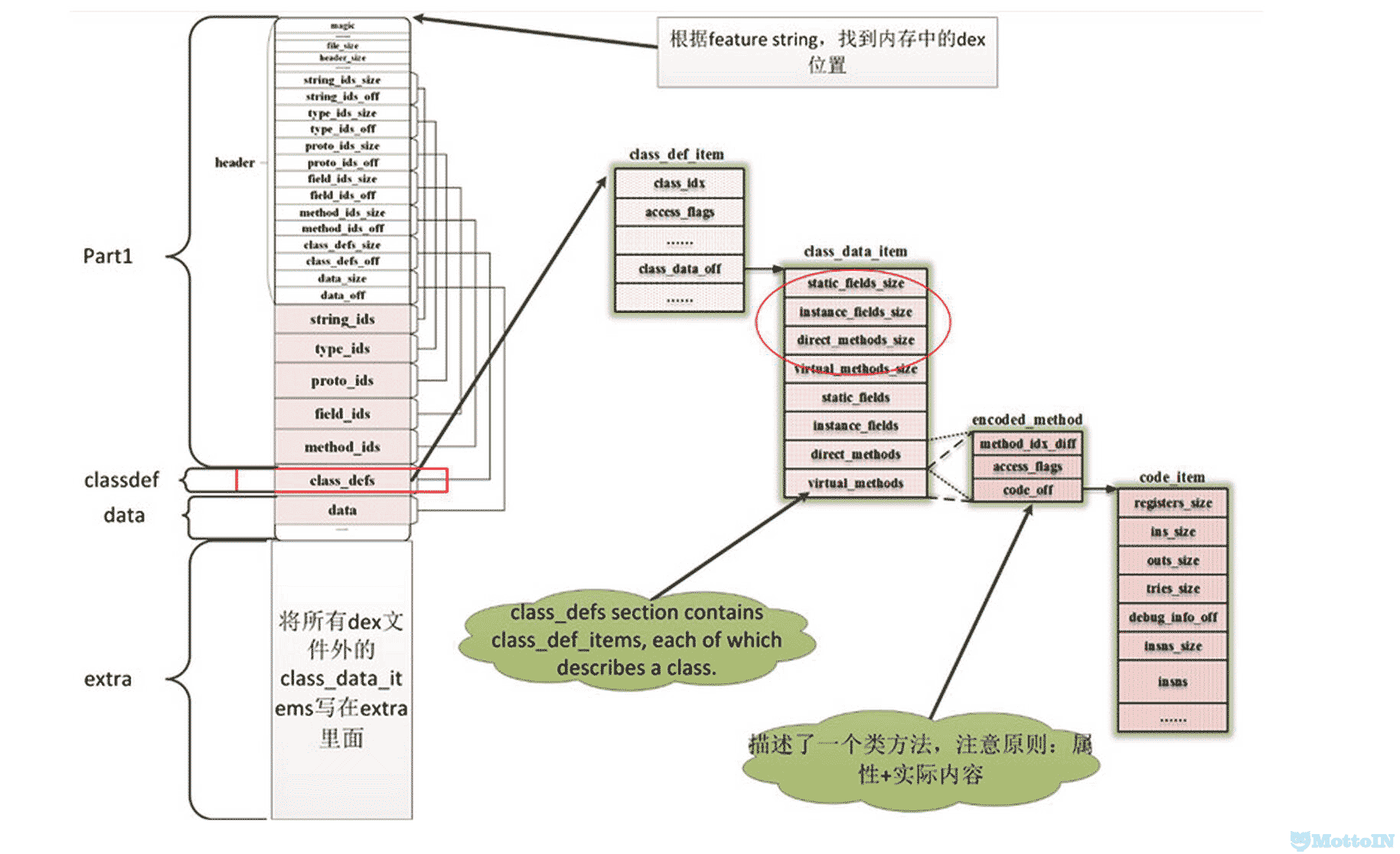
第二代壳通用脱壳方法，直接上工具：

工具地址及其用法：https://github.com/zyq8709/DexHunter

DexHunter的工作流程：



DexHunter的工作原理：



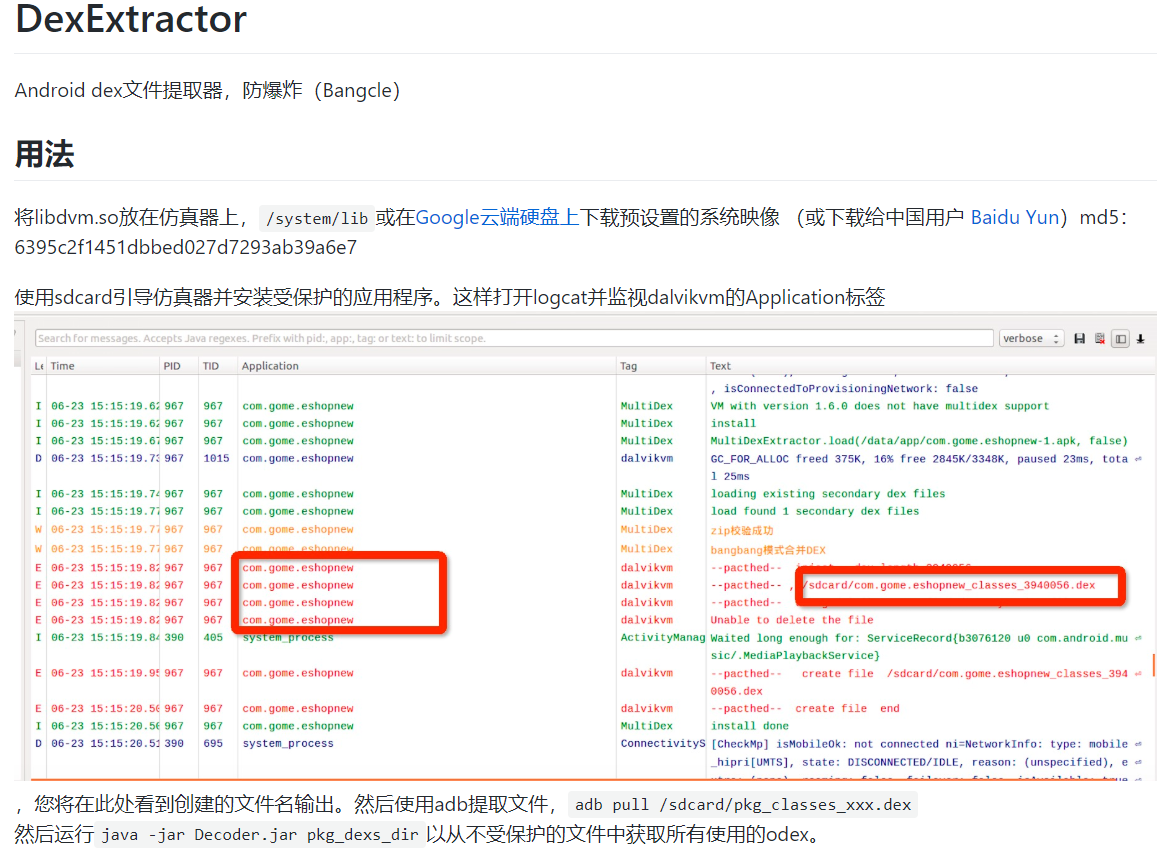
GitHub上有详细描述：

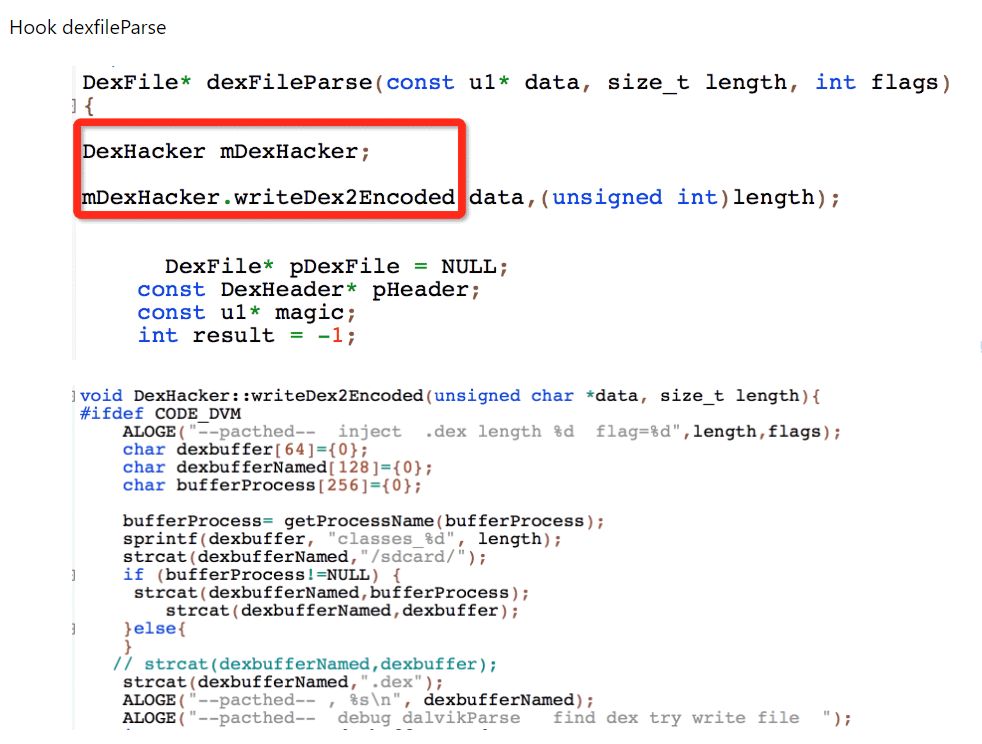


### 修改安卓源码并刷机（DexExtractor工具）

针对无抽取代码

https://github.com/lambdalang/DexExtractor





## 4.4 静态脱壳机

### 分析壳so逻辑并还原加密算法

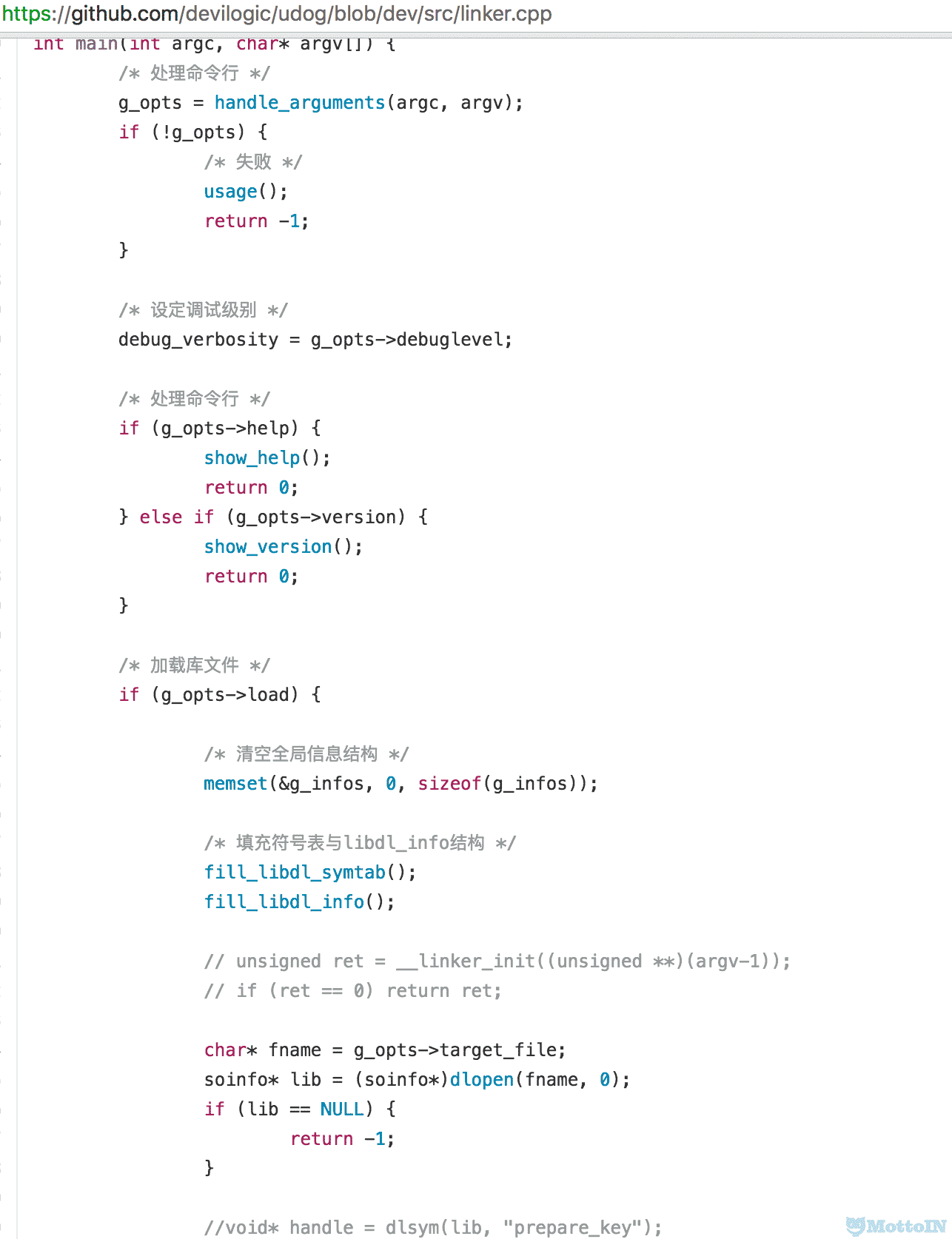
http://www.cnblogs.com/2014asm/p/4924342.html



### 自定义linker脱so壳

https://github.com/devilogic/udog

main() -> dump\_file()



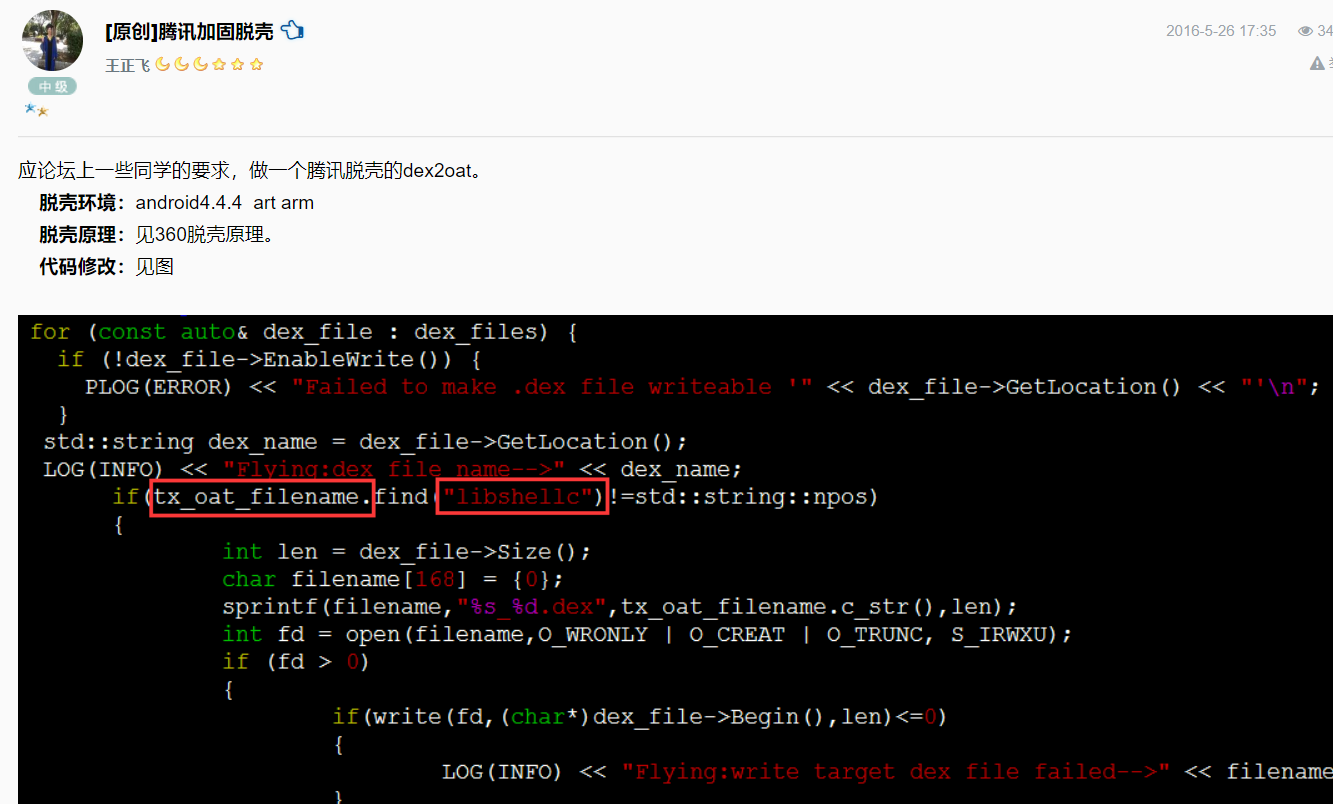
# 5 第三代壳

1. dex2oat法
2. 定制系统

## 5.1 dex2oat法

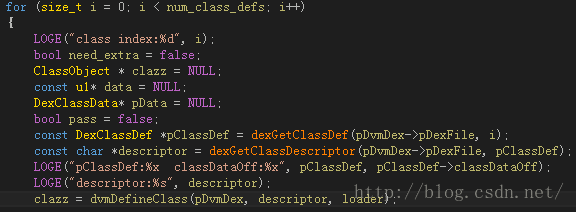
ART模式下，dex2oat生成oat时，内存中的dex是完整的，此时可以用修改后的dex2oat文件替换原系统的dex2oat文件

http://bbs.pediy.com/showthread.php?t=210532



## 5.2 定制系统

HOOK函数Dalvik\_dalvik\_system\_DexFile\_defineClassNative()，枚举所有的DexClassDef，对所有的class调用dvmDefineClass进行强制加载



# 6 第N代壳（代码混淆壳）

基于LLVM的虚拟机软件壳，即虚拟壳，还有基于白盒的代码混淆，第三代壳代表了Android平台上最高级别的软件加密技术。

so + vmp动态调试 ＋ 人肉还原

# 参考链接

<https://blog.csdn.net/wutianxu123/article/details/102865173>

该文档含以下内容，也需要参考



<http://www.45fan.com/article.php?aid=18012764813421009492230319>

<https://www.jianshu.com/p/6a504c7928da>