

# 因一纸设计稿，我把竞品APP扒得裤衩不剩(中)

原创

[coder-pig](#) 于 2019-12-19 15:37:45 发布 3555 收藏 25

分类专栏: [Android实战](#) 文章标签: [逆向](#) [Android](#) [混淆](#) [反混淆](#) [脱壳](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: [https://blog.csdn.net/coder\\_pig/article/details/103615488](https://blog.csdn.net/coder_pig/article/details/103615488)

版权



[Android实战](#) 专栏收录该内容

20 篇文章 5 订阅

订阅专栏

严正声明:

- 1、相关破解技术仅限于技术研究使用, 不得用于非法目的, 否则后果自负。

- 2、笔者仅出于对技术的好奇, 无恶意破坏APP, 尊重原开发者的劳动成果, 未用于商业用途。

## 0x1、无形之刃, 最为致命 => 碎碎念

上一篇文章《因一纸设计稿, 我把竞品APP扒得裤衩不剩(上)》是一篇比较简单的:

- jsw => 技师文, 呸,
- jsw => 记述文, 呸呸,
- jsw => 技术文, 呸呸呸, 这什么垃圾输入法!

技术文, 但是这评论区的风气, 貌似有点不对???



冤枉啊, 小弟真没去过这种地方, 也没体验过这种“服务”, 只是道听途说, 可能:

我描述得「绘声绘色」, 加之各位看官「浮想联翩」, 才会觉得「煞有介事」。

em...那个, 可以扶下我起来么, 那个...跪久了...腿有点麻...



顺带恭喜下：FPX 3-0 G2，喜提S9总冠军，FPX牛逼！！！破音！！！！



亚索的快乐你不懂~



哈哈，回到本文：

- 发现 => 很多童鞋对APP逆向很感兴趣；
- 但是 => 网上关于APP逆向文章的比较零散；
- 不知 => 如何入手，毕竟逆向的水可深了；
- 笔者 => 也只是个小白玩家，兴趣使然玩玩而已；
- 分享 => 目前会的一些「**Android APP基础逆向姿势**」；
- 还望 => 各位真、逆向大佬轻喷；
- 如有 => 有更好的工具或者方法安利，欢迎评论区指出，谢谢~

顺带分享几个笔者常逛的逆向论坛：

- 看雪论坛：<https://bbs.pediy.com/>
- 吾爱破解：<https://www.52pojie.cn/>
- 蚁安网：<https://bbs.mayidui.net/>
- 逆向大佬姜维：<http://www.520monkey.com/>

贴心提醒：

此文内容较多，可能会有些枯燥，建议先点赞收藏，茶余饭后再慢慢品尝~

## 0x2、提莫队长，正在待命 => 硬件准备

在开始折腾Android APP逆向前，你需要：



1、一台「具有完整Root权限」的Android手机，注意是「完整Root」权限!!!

比如「魅族手机」在设置->安全->Root权限，中可以开启Root权限，但是却是「阉割的Root权限」，安装SuperSu重启后就一直卡气球。

2、怎么Root? 根据自己的手机机型百度和逛各种搞机论坛吧（不要问我!）一般的常见的流程:

解BL锁(BootLoader) -> 刷第三方Recovery(如TWRP) -> 卡刷Magisk 或 SuperSU(Android 8.0以前)

3、不要轻易尝试使用哪种「一键Root」的软件(大部分是毒瘤，如KingRoot)，当然不是就不能用，可以过河拆桥，比如我的魅蓝E2的Root流程:

- 安装KingRoot(v5.0)授予Root权限，修复Root权限异常，此时就有完整Root权限了;
- 接着用「移除KingRoot」删掉KingRoot，此时还有完整Root权限;
- 安装SuperSu(v2.8.2)，常规方式更新二进制文件，重启，Root完成。

4、推荐些能Root的手机?

Google Pixel亲儿子(真原生，香，就是性价比不高)，小米，一加 等。

5、没钱买Android机或者已经有不能Root的手机了，可以试试「Android模拟器」

AS自带的AVD模拟器Root可以参见《搞机：AS自带模拟器AVD Root 和 Xposed安装》  
也可以使用其他第三方的安卓模拟器，比如「夜游安卓模拟器、BlueStacks蓝叠」等。

## 0x3、一点寒芒先到，随后枪出如龙 => 概念与名词

在开始折腾APP逆向前，先来了解一些概念与名词~



### ① APK文件里都有什么?

获取APK的渠道：酷安、应用宝，豌豆荚等应用市场下载，有些还提供「应用历史版本」下载。

APK本质上是一个「压缩包」，把「.apk后缀」改为「.zip后缀」后解压，可以看到如下目录结构 (可能还有其他文件):

名称
▶ assets
▶ lib

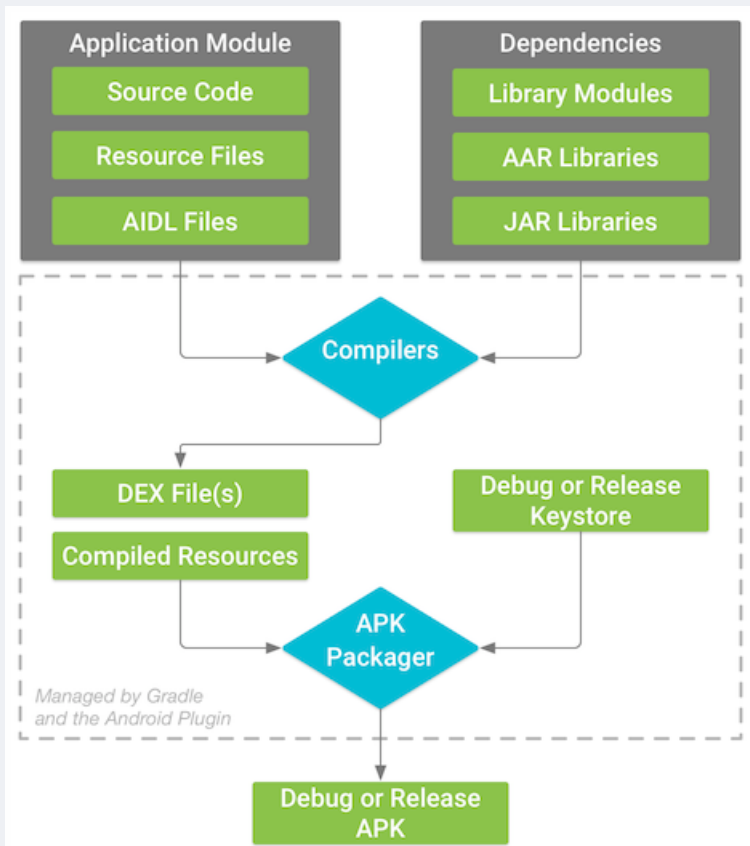


简单介绍下：



## ② 编译APK和反编译APK

所谓的「编译」，就是把「源码、资源文件等」按照一定的「规则」打包成APK，官网 提供了详细的编译构建过程图：



简述下大概流程：

### Step 1: 资源文件处理「AAPT」

- assets会原封不动地打包在APK中;
- res中每一个资源会赋予资源ID, 以常量形式定义在R.java中, 生成一个resource.arsc文件(资源索引表)。

### Step 2: aidl文件「aidl」

- 将aidl后缀的文件转换为可用于进程通信的C/S端Java代码。

### Step 3: Source Code「Java Compiler」

- 编译生成.class文件。

### Step 4: 代码混淆「ProGuard」(可选)

- 增加反编译难度, 命名缩短为1-2个字母的名字, 压缩(移除无效类、属性、方法等), 优化bytecode移除没用的结构。

### Step 5: 转换为dex「dx.bat」

- 把所有class文件转换为classes.dex文件, class -> Dalvik字节码, 生成常量池, 消除冗余数据等。(方法数超65535会生成多个dex文件)

### Step 6: 打包「ApkBuilder」

- 把resources.arsc、classes.dex、其他的资源一块打包生成未签名apk。

### Step 7: 签名「Jarsigner」

- 对未签名apk进行debug或release签名。

### Step 8: 对齐优化「zipalign」

- 使apk中所有资源文件距离文件起始偏移为4字节的整数倍, 从而在通过内存映射访问apk文件时会更快。

如果想了解更多编译构建流程可移步至: [《10分钟了解Android项目构建流程》](#)

而「反编译」则是反过来了, 通过一些反编译工具, 提取出源码, 转换过程如下:

「APK =====> Dex =====> Jar(class文件)/Smali ==> Java源码」。

## ③ 加固和脱壳

APK可以说是每个Android开发仔的「心血结晶」, 把各种自己觉得「牛逼哄哄的奇淫巧技」封装其中。但总有些「心怀叵测」想去搞你的APP, 通过一些「反编译工具」获取你的源码, 然后为所欲为:

- 加广告: 你应用免费, 给你加点广告, 亦或者改成付费, 然后下载量比你的多, 气不气?
- 破解付费: 你应用收费, Hook掉你的检测方法, 发个破解包, 还到处传播, 气不气?
- 恶意攻击: 逆向得出请求接口规律, 批量短信验证注册, 耗光你的短信池等, 气不气?

这种「恶劣」的行径令人「气愤」像极了某类经典「动作电影」里的桥段:



- 男子上进, 努力工作, 妹子贤惠, 料理家务;
- 妹子每天做好饭菜, 等男子回来, 一起吃饭, 满怀憧憬, 畅谈以后的二人世界;
- 酒饱饭后, 温饱思XX, 不可描述一番, 却被「居心叵测」的邻居给盯上了;
- 和往常一样, 男子出门上班, 妹子在家做家务, 晾衣服;
- 邻居上线, 用「谎言」诱骗妹子开门, 接而挤门而入;
- 用「暴力和胁迫」, 无视妹子的やめて和反抗, 违背个人意愿;

- 粗暴地把衣服一件件褪去，只剩下那「万恶的马赛克」；
- 守护着最后的一处「绝对领域」；
- 在几番不可描述后，把妹子占为己有，然后像玩物般戏耍。

看着妹子「因情绪过激而身体抽搐」，哭得「梨花带雨」，不禁让人「心生怜惜」，像我这种感性的蓝孩子：



总会忍不住抽上几张抽纸，“静静抹泪”，擦拭完，顿觉索然无味，一片空明，然后开始反思：

为什么那个邻居不是我？呜呜呜...



除了「同情女主」和「斥责坏人」外，应该如何避免这种事情的发生呢？

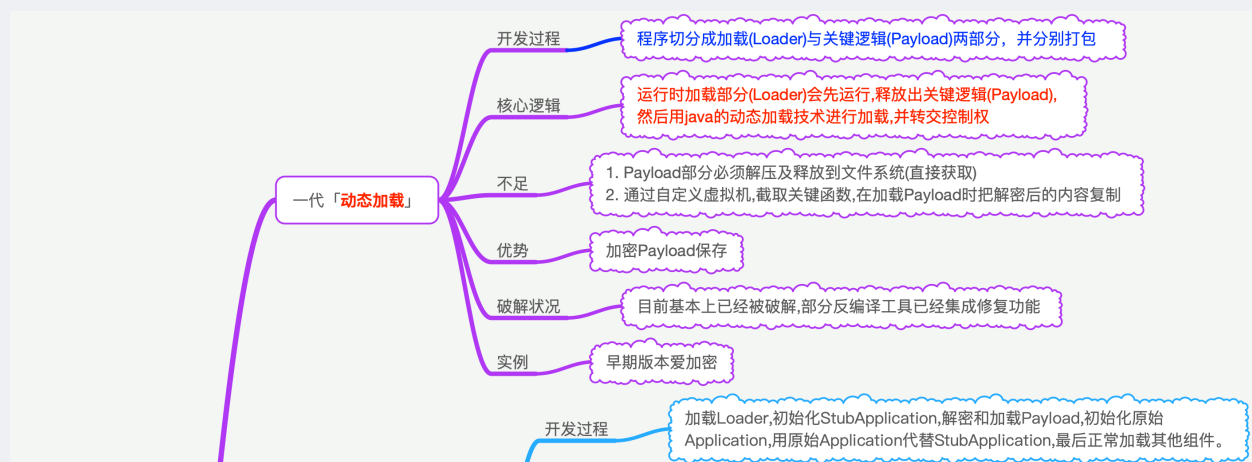
- 1、花点钱，请个「保镖」看门，坏人想进来要先过保镖这一关；
- 2、给妹子「加个锁」，让坏人无法不可描述，只能望而兴叹。

可以把例子中的「妹子」看做是我们编写的「APK」，而「请保镖」和「加锁的操作」则可以看做是「APK加固」，另外加固又称「加壳」，壳的定义：

一段专门负责「保护软件不被非法修改或反编译的程序」，一般先于程序运行，拿到控制权，然后完成它们保护软件的任务。

有「加壳」，自然也有「脱壳」，即去掉这层壳，拿到源码，也称为「砸壳」。

关于加固技术的发展，看雪上有篇：《一张表格看懂：市面上最为常见的 Android 安装包（APK）五代加固技术发展历程及优缺点比较》，不过图不怎么清晰，笔者重新排版了一下，有兴趣的读者可以看看：



## Android五代加固技术

### 二代「内存不落地加载」

- 核心逻辑**
  1. 拦截系统IO相关函数(read,write),在这些函数中提供透明加解密.
  2. 直接调用虚拟机提供的函数进行落地加载.
- 不足**
  1. 在启动过程中需要处理大量的解密操作,容易造成黑屏或假死
  2. Payload被加载后,在内存中是连续的,内存dump即可直接获取.(针对关键函数进行拦截即可把内存dump)
- 优势**

当前市面上最为常见,通常作为一项基础性的免费服务向用户提供.
- 破解状况**

已经出现专业人士自行研究的手工脱壳方法,但尚未出现自动脱壳工具,破解难度依然比较大.
- 实例**

市场上流行的大多数在线加固服务,如腾讯加固,360加固,百度加固,阿里聚安全,网易易盾等等

### 三代「指令抽取」

- 开发过程**

首先,将保护级别降到函数级别,然后将原始DEX内的函数内容(Code Item)清除.单独移除到一个文件中,运行阶段将函数内容重新恢复到对应的函数体.
- 核心逻辑**
  1. 加载之后恢复函数内容到DEX壳所在的内存区域
  2. 加载之后将函数内容恢复到虚拟机内部的结构体上,虚拟机读取DEX文件后内部对每一个函数有一个结构体,这个结构体有一个指针指向函数内容(CodeItem).可以通过修改这个指针修改对应的函数内容
  3. 拦截虚拟机内与查找执行代码相关的函数,返回函数内容
- 不足**
  1. 指令抽取方案跟虚拟机的JIT性能优化冲突,达不到性能最佳的性能
  2. 依然使用Java虚拟机进行函数内容执行,无法对抗自定义虚拟机.
  3. 指令抽取技术使用了大量的虚拟内部结构与未文档化的特性,再加上Android复杂的厂商定制,带来大量的兼容问题.
- 优势**

在对自定义虚拟机记录函数执行时函数的内容,遍历触发所有的函数,从而获取全部抽离的内容,最终组装成完整的dex文件.科通过自动化完成整个过程.
- 破解状况**

部分被破解,已经出现专业人士自行研究手工破解方法,但是至今为止未出现自动脱壳工具
- 实例**
  1. 现在免费版的"爱加密"
  2. 棒棒安全免费版

### 四代「指令转换」

- 开发过程**

A. DEX文件内的函数被标记为native,内容被抽离并转换成一个符合JNI要求的动态库,动态库通过JNI和Android系统进行调用.  
B. DEX文件内的函数被标记为native,内容被抽离并转换成自定义的指令格式,该格式使用自定义接收器执行,和A一样需要使用JNI和Android系统进行交互.
- 核心逻辑**
- 不足**
  1. 不论使用指令转换/VMP加固的A方案或B方案,其必须通过虚拟机提供的JNI接口与虚拟机进行交互.攻击者可以直接将指令转换/VMP加固当作黑盒,通过自定义的JNI接口对象,对黑盒内部进行探测.记录和分析,进而得到完整DEX程序.
  2. 第四代VMP加固一般配合第三代加固技术使用,所以第三代的所有兼容问题,指令转换/VMP加固也存在
- 优势**
- 破解状况**

部分被破解,已经出现专业人士自行研究手工破解方法,但是至今为止未出现自动脱壳工具
- 实例**

大部分需要定制收费的加密服务(如爱加密,邦邦安全,中国移动加固,以及手机银行自研加固等).

### 五代「虚拟保护」

- 开发过程**

基于第四代方案的A方式(Java或Kotlin -> C/C++),基于LLVM编译工具链(同时支持C/C++,Swift,Object-C),通过对IR执行指令转换,生成自定义指令集(IR-VM).APP内部抽离出独立的执行环境,该核心代码运行程序在此独立的执行环境里.
- 核心逻辑**
- 不足**
  1. 无法摆脱对JNI的依赖,因此依然存在第四代加固技术的缺陷,存在被记录修复的可能性.
  2. 由Java转换成等价的C/C++,会导致体积展线性增大,性能有所下降.
- 优势**
  1. 由Java转C/C++后的代码,由于虚拟机的保护,逆向难度会上升一个数量级.
  2. 对于C/C++部分逻辑,智能投入时间去破译虚拟机的指令集含义.
- 破解状况**

大多数未被破解
- 实例**

极为少数,需要特殊定制的加固服务,通常用于银行金融机构等关乎国家安全的重点领域.

## ④ 混淆与反混淆

「混淆」可以类比为上面「万恶的马赛克」,阻碍人类进步的绊脚石.而混淆则是增加了反编译的难度,同理,「反混淆」则对应「去除马赛克」,试图还原它原来的样子.

## 0x4、发动机已启动，随时可以出发 => 获得APP源码



加固虽然能在一定程度上「防止反编译和二次打包」，但加固后的APP可能会带来一些问题：

体积增大，启动速度变慢，兼容问题等

网上「免费加固」方案有很多，脱壳教程也是烂大街，而且有些恶心的第三方加固还会给你加点料(360加固锁屏广告)，而使用「企业级的加固」，则需要支付不菲的费用，所以很多APP直接选择了「裸奔」。先来讲解一下未加固的怎么获取源码吧~

### ① 未加固(笔者使用的工具：apktool + jadx)

- 使用apktool: 获取「素材资源，AndroidManifest.xml以及smali代码」
- 使用jadx: 把「classes.dex」转换为「java」代码

使用Jadx的注意事项：

使用jadx-gui可直接打开apk查看源码，但如果APK比较大(classes.dex有好几个)，会直接卡死(比如微信)，笔者的做法是命令行一个个dex文件去反编译，最后再把反编译的文件夹整合到同一个目录下。

这样的操作繁琐且重复，最适合批处理了，遂写了个反编译的批处理脚本(按需)：

```
"""
自动解压apk，批量使用jadx进行反编译，结果代码汇总
"""
import os
import shutil
import zipfile
from datetime import datetime

apk_file_dict = {} # APK路径字典

# 遍历构造APK路径字典(构造文件路径列表，过滤apk，拼接)
def init_apk_dict(file_dir):
    apk_path_list = list(filter(lambda fp: fp.endswith(".apk"),
                               list(map(lambda x: os.path.join(file_dir, x), os.listdir(file_dir)
                               ))))
    index_list = [str(x) for x in range(1, len(apk_path_list) + 1)]
    return dict(zip(index_list, apk_path_list))

# 移动文件夹
def move_dir(origin_dir, finally_dir):
    shutil.move(origin_dir, finally_dir)

# 如果文件夹存在删除重建
def del_dir_if_existed(path):
```



```

def deal_dir_existed(path):
    if os.path.exists(path):
        print("检测到文件夹【%s】已存在, 执行删除..." % path)
        shutil.rmtree(path)
    os.makedirs(path)

# 判断目录是否存在, 不存在则创建
def is_dir_existed(path, mkdir=True):
    if mkdir:
        if not os.path.exists(path):
            os.makedirs(path)
    else:
        return os.path.exists(path)

# 获取目录下的所有文件路径
def fetch_all_file(file_dir):
    return list(map(lambda x: os.path.join(file_dir, x), os.listdir(file_dir)))

# 解压文件到特定路径中
def unzip_file(file_name, output_dir):
    print("开始解压文件...")
    f = zipfile.ZipFile(file_name, 'r')
    for file in f.namelist():
        f.extract(file, os.path.join(os.getcwd(), output_dir))
    print("文件解压完毕...")

if __name__ == '__main__':
    print("遍历当前目录下所有APK...")
    apk_file_dict = init_apk_dict(os.getcwd())
    print("遍历完毕...\n\n===== 当前目录下所有的APK =====\n")
    for (k, v) in apk_file_dict.items():
        print("%s.%s" % (k, v.split(os.sep)[-1]))
    print("\n%s" % ("=" * 45))
    choice_pos = input("%s" % "请输入需要反编译APK的数字编号: ")
    print("=" * 45, )
    choice_apk = apk_file_dict.get(choice_pos)
    apk_name = choice_apk.split(os.sep)[-1][:-4] # APK名字

    # 创建相关文件夹
    crack_dir = os.path.join(os.getcwd(), apk_name) # 工程根目录
    deal_dir_existed(crack_dir)
    crack_apktool_dir = os.path.join(crack_dir, "apktool" + os.sep) # APKTool反编译目录
    deal_dir_existed(crack_apktool_dir)
    crack_jadx_dir = os.path.join(crack_dir, "jadx" + os.sep) # JADX反编译目录
    deal_dir_existed(crack_jadx_dir)
    crack_temp_dir = os.path.join(crack_dir, "temp" + os.sep) # 解压后文件的临时存储路径
    deal_dir_existed(crack_temp_dir)

    # 利用APKTool提取资源文件
    begin = datetime.now() # 计时
    print("APKTool提取资源文件...")
    os.system("./apktool d %s -f -o %s" % (choice_apk, crack_apktool_dir))

    # 复制一份AndroidManifest.xml、res、assets文件到外部
    shutil.copy(os.path.join(crack_apktool_dir, "AndroidManifest.xml"), os.path.join(crack_dir, "
AndroidManifest.xml"))

```

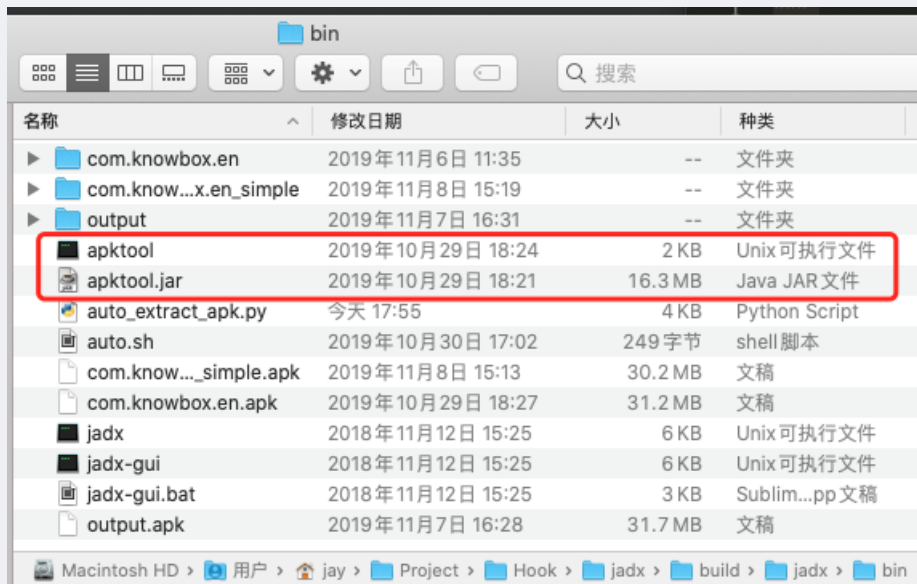
```

shutil.copytree(os.path.join(crack_apktool_dir, "res" + os.sep), os.path.join(crack_dir, "res"
+ os.sep))
shutil.copytree(os.path.join(crack_apktool_dir, "assets" + os.sep), os.path.join(crack_dir, "
assets" + os.sep))
print("资源文件提取完毕")

# 利用jadx反编译源码
print("JADX反编译提取源码...")
choice_apk_zip = shutil.copy(choice_apk, choice_apk.replace(".apk", ".zip"))
unzip_file(choice_apk_zip, crack_temp_dir)
print("开始批量反编译dex文件")
for dex in list(filter(lambda fp: fp.endswith(".dex"), fetch_all_file(crack_temp_dir))):
    os.system(
        "./jadx -d {0} {1}".format(os.path.join(crack_jadx_dir, dex.split(os.sep)[-1][: -4]),
dex))
print("所有dex文件反编译完毕")
# 将资源文件移入
shutil.move(os.path.join(crack_dir, "AndroidManifest.xml"), os.path.join(crack_jadx_dir, "And
roidManifest.xml"))
shutil.move(os.path.join(crack_dir, "res" + os.sep), os.path.join(crack_jadx_dir, "res" + os.
sep))
shutil.move(os.path.join(crack_dir, "assets" + os.sep), os.path.join(crack_jadx_dir, "assets"
+ os.sep))
# 删除临时文件夹，压缩文件
shutil.rmtree(crack_temp_dir)
os.unlink(choice_apk_zip)
end = datetime.now()
print("收尾操作~~~\n反编译完成，总耗时: %s秒" % (end - begin).seconds)

```

执行前，你需要把apktool相关的东西，丢到jadx/build/jadx/bin目录下，如图所示：



接着终端键入：`python3 auto_extract_apk.py`，回车后输入对应编号，回车开始编译：

```

+ bin git:(master) python3 auto_extract_apk.py
遍历当前目录下所有APK...
遍历完毕...

===== 当前目录下所有的APK =====

1.com.knowbox.en_simple.apk
2.weixin708android1540.apk
3.com.knowbox.en.apk

```

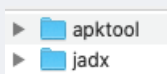
```
4.output.apk
=====
请输入需要反编译APK的数字编号： 4
```

静待片刻后：

```
ERROR - Method: com.mob.tools.util
ssage):boolean
WARN - 1911 warnings in 399 nodes
ERROR - finished with errors
所有dex文件反编译完毕
收尾操作~~~
反编译完成，总耗时：76秒
```

**Tips:** 这里没有把多个classes文件夹整合到一起，是因为有些APP会出现合并冲突。

打开反编译后的目录，有如下两个文件夹：



按照自己的需要用Android Studio打开其中一个就好了：

- **apktool**目录：apktool反编译后的内容，主要用于**smali**动态调试。
- **jadx**目录：反编译成Java的内容。

## ② 反混淆(simplify、Deguard)

代码是拿到了，但是打开代码，「一堆的abcd」，跟到眼花，可以试下「反混淆」，方案有两类，一种是通过「代码逆推」出名字，另一种是通过「统计逆推」出名字。

第一种方案的工具有很多（**Jeb2**，**simplify**等），前者付费需破解，Java版本有限制，Mac配置有点麻烦，故笔者用的是后者：「**Simplefy**」，**Github仓库**：<https://github.com/CalebFenton/simplify>，使用方法也很简单：

打开终端依次键入：

```
# 拉取仓库代码
git clone --recursive https://github.com/CalebFenton/simplify.git

# 来到目录下
cd simplify

# 编译
./gradlew fatjar
```

编译完后，执行下述指令即可反混淆APK：

```
# 反混淆APK（需要反混淆的APK，反混淆后的APK名）
./gradlew build && cp xxx.apk yyy.apk
```

静待反混淆完毕，接着反编译批处理脚本走一波，打开MapFragment比对下：



```
MapFragment.this.e.scrollToY(1, 12);
}
}

public void onScrollStateChanged(RecyclerView recyclerView, int i) {
    super.onScrollStateChanged(recyclerView, i);
    if (MapFragment.this.j.m) {
        LinearLayoutManager layoutManager = (LinearLayoutManager) recyclerView.getLayoutManager();
        if (i == 0) {
            int findFirstVisibleItemPosition = layoutManager.findFirstVisibleItemPosition();
            int findLastVisibleItemPosition = layoutManager.findLastVisibleItemPosition();
            if (findFirstVisibleItemPosition > MapFragment.this.l) {
                MapFragment.this.g.setVisibility(0);
            } else {
                MapFragment.this.g.setVisibility(8);
            }
            if (findLastVisibleItemPosition < MapFragment.this.l) {
                MapFragment.this.h.setVisibility(0);
            } else {
                MapFragment.this.h.setVisibility(8);
            }
        }
    }
}

};
@BindView(2131698465)
private RecyclerView b;
@BindView(2131698466)
private RecyclerView c;
@BindView(2131698467)
private RecyclerView d;
@BindView(2131698468)
private RecyclerView e;
@BindView(2131698469)
private RecyclerView f;
@BindView(2131698470)
private LottieAnimationView g;
@BindView(2131698471)
private LottieAnimationView h;
private LineAdapter i;
private OnMainCourseIndexAdapter k;
private int l = 1;
private OnTouchListener m = new OnTouchListener() {
    public boolean onTouch(View view, MotionEvent motionEvent) { return true; }
};
private OnVisibleChangeListener n = new OnVisibleChangeListener() {
    public void a(String str) {
        ...
    }
};
}
```

反混淆前

反混淆后

相比混淆前，多了一些变量名，当然也不是完全的，偶尔还是有abcd，但是可读性稍微提高了些，比如查找的时候不用在一个个abcd排除，但是，编译挺耗时的，而且我的电脑风扇呼呼地响。

第二种是通过统计的方法，利用统计推断出名字：DEGUARD: <http://apk-deguard.com/>，打开官网：

选择需要反混淆的APK后，Upload上传，接着等待处理完成，!!! 别关页面!!!

一般需等待1-10分钟，处理完成后，点击output.apk，把APK下载到本地，同样执行批处理脚本反编译一波，和simplefy反混淆后的代码对比下：

<pre> public class MapFragment extends BaseUIFragment&lt;UIFragmentHelpers&gt; implements OnClickListener {     private int activity = 1;     private LineAdapter adapter;     @BindView(2131699471)     private LottieAnimationView emptyView;     private OnTouchListener mContainer = new OnTouchListener() {         public boolean onTouch(View view, MotionEvent motionEvent) { return true; }     };     @BindView(2131699466)     private RecyclerView mListview;     @BindView(2131699465)     private RecyclerView mRecyclerView;     @BindView(2131699467)     private RecyclerView mRootView;     @BindView(2131699470)     private LottieAnimationView mapView;     OnScrollListener mapViewPosition = new OnScrollListener() {         public void onScrollStateChanged(RecyclerView recyclerView, int i) {             super.onScrollStateChanged(recyclerView, i);             if (MapFragment.this.this\$0.min) {                 LinearLayoutManager recyclerLayout = (LinearLayoutManager) recyclerView.getLayoutManager();                 if (i == 0) {                     int \$i1 = \$r5.findFirstVisibleItemPosition();                     i = \$r5.findLastVisibleItemPosition();                     if (\$i1 &gt; MapFragment.this.activity) {                         MapFragment.this.mapView.setVisibility(0);                     } else {                         MapFragment.this.mapView.setVisibility(9);                     }                     if (i &lt; MapFragment.this.activity) {                         MapFragment.this.emptyView.setVisibility(0);                         return;                     }                     MapFragment.this.emptyView.setVisibility(8);                 }             }         }         public void onScrolled(RecyclerView recyclerView, int i, int i2) {             if (recyclerView.getScrollState() != 0) {                 MapFragment.this.mRecyclerView.scrollTo(((double) i) * 0.4d, i2);                 MapFragment.this.mRecyclerView.scrollTo(((double) i) * 0.6d, i2);                 MapFragment.this.mRootView.scrollTo(((double) i) * 0.8d, i2);                 MapFragment.this.view.scrollTo(i, i2);             }         }     }; } </pre>	<pre> 51 public class MapFragment extends BaseUIFragment&lt;UIFragmentHelpers&gt; implements OnClickListener { 52     private int activity = 1; 53     private LineAdapter adapter; 54     @BindView(2131699471) 55     private LottieAnimationView emptyView; 56     private OnTouchListener mContainer = new OnTouchListener() { 57         public boolean onTouch(View view, MotionEvent motionEvent) { return true; } 58     }; 59     @BindView(2131699466) 60     private RecyclerView mListview; 61     @BindView(2131699465) 62     private RecyclerView mRecyclerView; 63     @BindView(2131699467) 64     private RecyclerView mRootView; 65     @BindView(2131699470) 66     private LottieAnimationView mapView; 67     OnScrollListener mapViewPosition = new OnScrollListener() { 68         public void onScrollStateChanged(RecyclerView recyclerView, int i) { 69             super.onScrollStateChanged(recyclerView, i); 70             if (MapFragment.this.this\$0.length) { 71                 LinearLayoutManager recyclerLayout = (LinearLayoutManager) recyclerView.getLayoutManager(); 72                 if (i == 0) { 73                     int \$i1 = \$r5.findFirstVisibleItemPosition(); 74                     i = \$r5.findLastVisibleItemPosition(); 75                     if (\$i1 &gt; MapFragment.this.activity) { 76                         MapFragment.this.mapView.setVisibility(0); 77                     } else { 78                         MapFragment.this.mapView.setVisibility(9); 79                     } 80                     if (i &lt; MapFragment.this.activity) { 81                         MapFragment.this.emptyView.setVisibility(0); 82                         return; 83                     } 84                     MapFragment.this.emptyView.setVisibility(8); 85                 } 86             } 87         } 88         public void onScrolled(RecyclerView recyclerView, int i, int i2) { 89             if (recyclerView.getScrollState() != 0) { 90                 MapFragment.this.mRecyclerView.scrollTo(((double) i) * 0.4d, i2); 91                 MapFragment.this.mRecyclerView.scrollTo(((double) i) * 0.6d, i2); 92                 MapFragment.this.mRootView.scrollTo(((double) i) * 0.8d, i2); 93                 MapFragment.this.view.scrollTo(i, i2); 94             } 95         } 96     }; 97 } </pre>
--	--

大同小异，另外，反混淆并不能100%还原，而且还可能有些小错误，比如下面的代码：

<pre> this.r = new MainActivity\$WorkAdapter(getContext()); this.r.b(\$i0 == 1); this.r.b(new OnItemClickListener() {     public void onCellInfoChanged() {         if (MapFragment.this.webView != null) {             MapFragment.this.webView.b();         }     } }); </pre>	<pre> 340 341 342 343 344 345 346 347 </pre>	<pre> this.recyclerView.b(\$i0 == 1); this.recyclerView.b(\$i0 == 1); this.recyclerView.b(\$i0 == 1); this.recyclerView.b(\$i0 == 1); this.recyclerView.b(\$i0 == 1); this.recyclerView.b(\$i0 == 1); this.recyclerView.b(\$i0 == 1); this.recyclerView.b(\$i0 == 1); </pre>
--	--	--

<pre> public void b(OnItemClickListener onItemClickListener) {     this.G = onItemClickListener; } </pre>	<pre> 197 198 199 200 201 </pre>	<pre> public void solve(OnItemClickListener onItemClickListener) {     this.G = onItemClickListener; } </pre>
---	----------------------------------	---

虽说反编译后的可读性有所提高，但建议还是搭配着混淆的源码看。

### ③ 脱壳（FDex2，反射助手，dumpDex）

终于来到很多同学期待的脱壳环节，先说明下，笔者只是「工具党」水平，不会Native层的，so文件调试！如果本节的工具，你脱不出来，或者脱出来有问题，笔者也是爱莫能助。看雪有很多帮人脱壳的大佬，可以在上面发个帖子求助下~



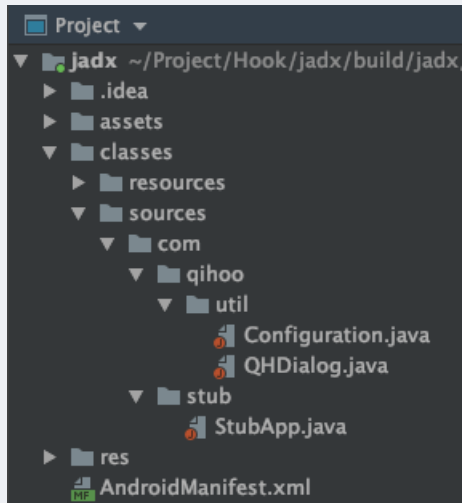
#### 1、判断是哪种加固

解压apk后在assets目录下看到so文件，比如360加固宝：libjagu.so和libjagu\_x86.so，百度搜下名字就知道是哪家的加固了，也可以直接用后面讲的「MT文件管理器2.0」直接查看。

## 2、FDex2脱壳（只适用于Android 7或以下版本，可以脱市面上大多数免费加固，成功率较高，推荐）

- 有ROOT：安装「XposedInstaller」和「FDex2」
- 没ROOT：安装「VirtualXposed」「FDex2」

比如：这里有个「360免费版加固的APK」，直接用jadx反编译后导入AS，但是反编译后的classes：

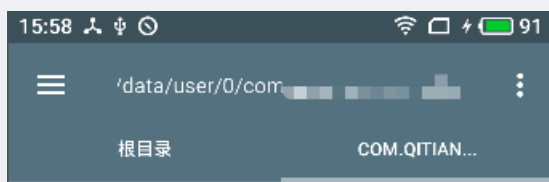


只有这么一丢丢点东西，把「待脱壳应用」安装到手机上，接着用FDex2来脱壳  
已Root玩家：XposedInstall启用FDex2插件重启后，按如下步骤脱：

- Step 1: FDex2选中待脱壳应用：



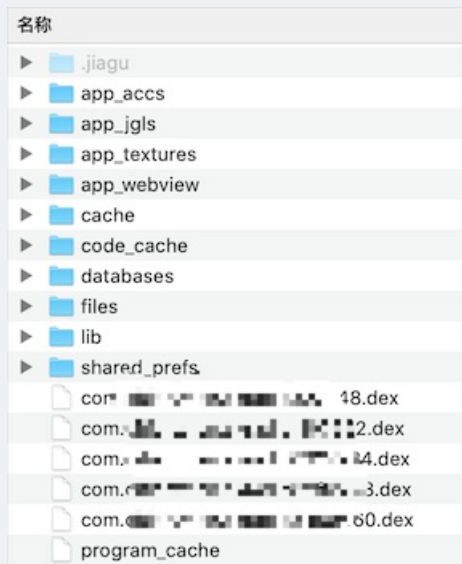
- Step 2: 打开待脱壳应用，接着来到上图的dex输出目录：



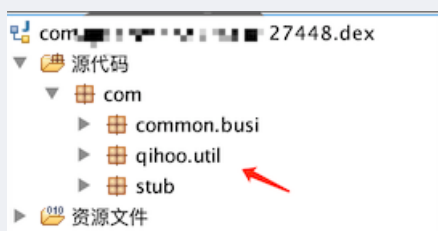


- Step 3: 把整个目录拉到电脑上，这里直接用adb命令拉取：

```
adb root
adb pull /data/user/0/包名 电脑文件夹
```



- Step 4: 「剔除加固相关的dex」，用jadx-gui依次打开，看到下面这种，直接把dex删掉

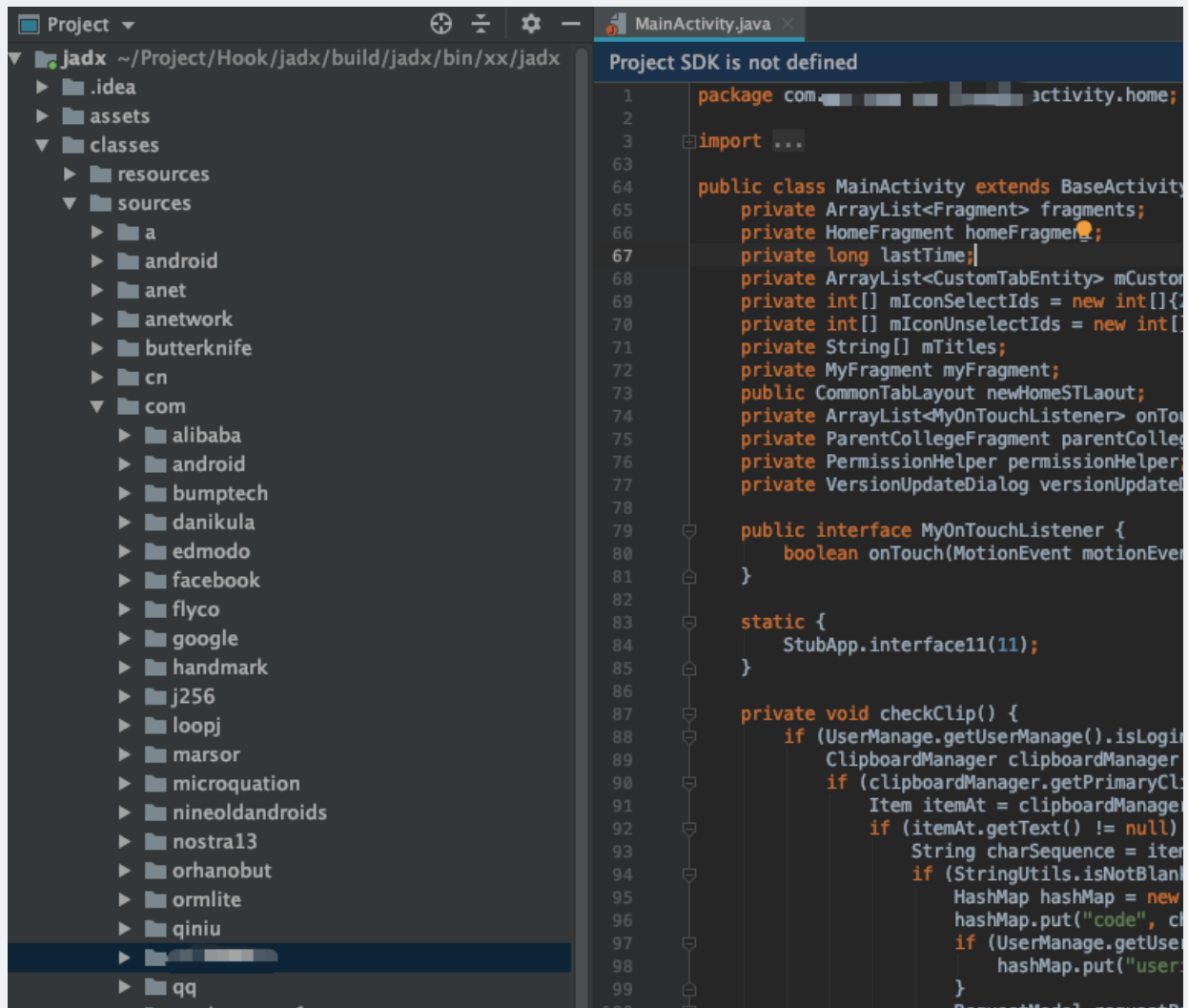


- Step 5: 使用jadx命令反编译dex，顺带改名，命名规则：按照文件大小降序，示例如下：

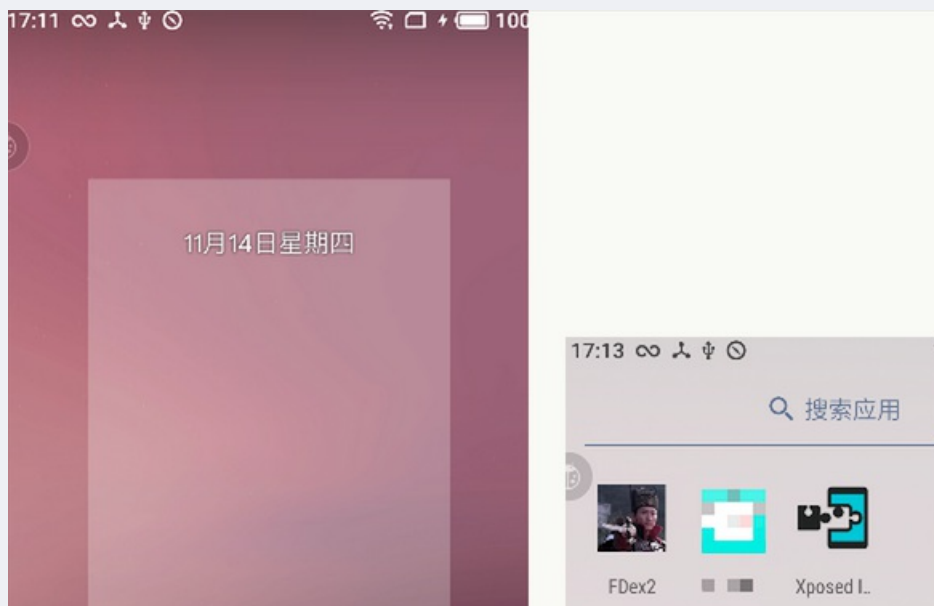
# 按照文件从大到小排序!!!

```
jadx aaa.dex -d classes  
jadx bbb.dex -d classes1  
jadx ccc.dex -d classes2
```

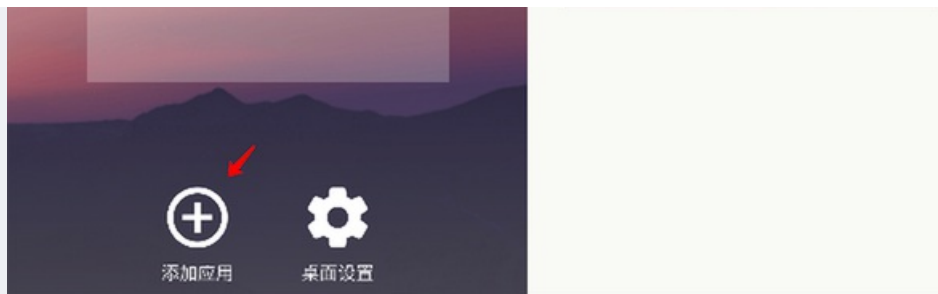
- Step 6: 删掉没脱壳前反编译项目里的classes, 把这几个复制到其中:



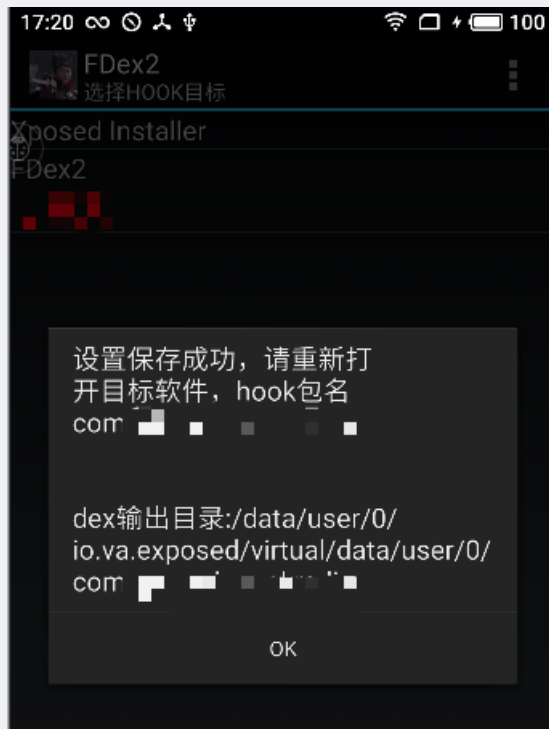
行吧, 脱壳成功, 这里其实还可以还原APK的(二次打包), 等下再讲~  
未root玩家, 安装打开VirtualXposed, 添加应用: Fdex2和待脱壳应用







如果炮制，只是dex的路径有些不一样。



### 3、反射大师（和FDex类似，下载地址：<https://www.lanzous.com/b04xdujg>）

注意，同样只支持Android 7.0及以下，adb安装后，xposed启用插件，重启手机，接着打开反射大师：

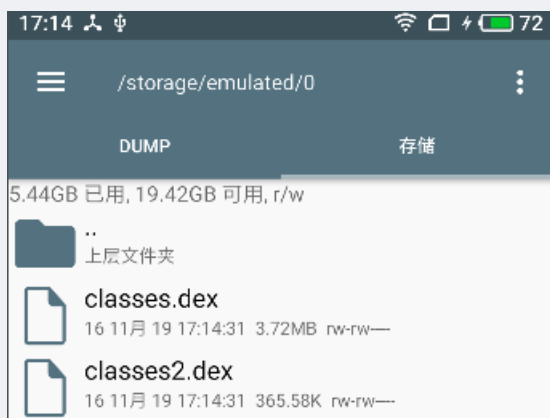
**Step 1:** 选中待脱壳APP，弹出对话框选择打开



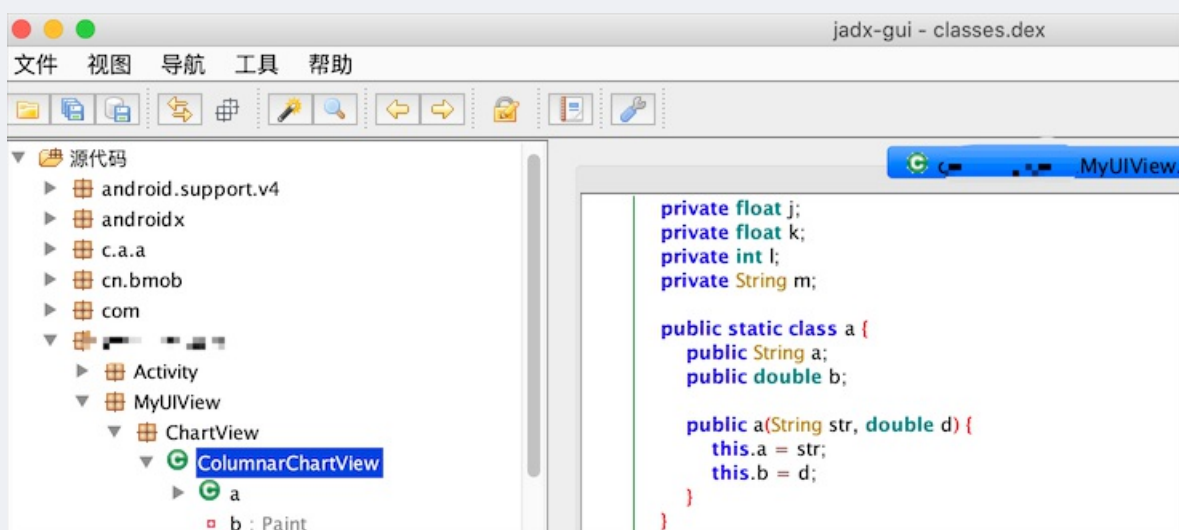
**Step 2:** 点击中间的六芒星，弹出如下对话框，长按「写出DEX」



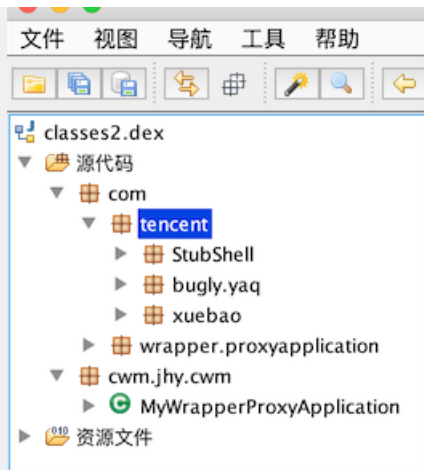
Step 3: 等待写出完毕，可以在/storage/emulated/0中找到导出的dex:



Step 4: pull到电脑上用jadx-gui打开看看:



行吧，脱壳成功，就是我们想要的dex了，另一个classes2.dex则是相关的~:



#### 4、dumpDex脱壳 (Github: <https://github.com/WrBug/dumpDex/releases>)

官方仓库的README.md中有一句:

**apk文件不会实时更新, 获取最新apk请自行编译源码**

可以的话建议自己编译, 流程也很简单:

# 1、拉取项目代码到本地

```
git clone https://github.com/WrBug/dumpDex.git
```

# 2、AS中Open项目, 等待编译完成

# 3、删掉build.gradle里签名相关的代码

# 4、点击顶部菜单栏Build -> Build APK, 或者直接在终端./gradlew clean build

# 5、adb命令直接把编译生成的apk安装到手机上

# 6、接着来到如下左图路径, 把对应的so, 通过adb push到目录下:

```
adb push lib/armeabi-v7a/libnativeDump.so /data/local/tmp
```

```
adb push lib/arm64-v8a/libnativeDump.so /data/local/tmp/libnativeDump64.so
```

# 修改权限

```
adb shell
```

```
su
```

```
chmod 777 /data/local/tmp/libnativeDump.so
```

```
chmod 777 /data/local/tmp/libnativeDump64.so
```

# 临时关闭SELinux(重启后会失效, 可调用getenforce查询)

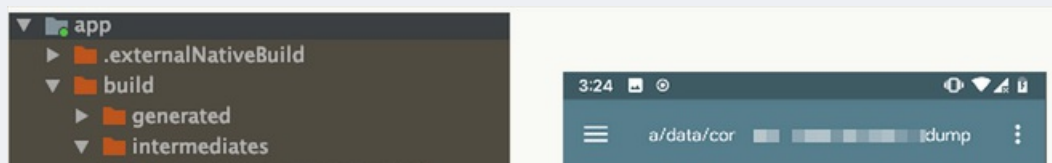
```
setenfore 0
```

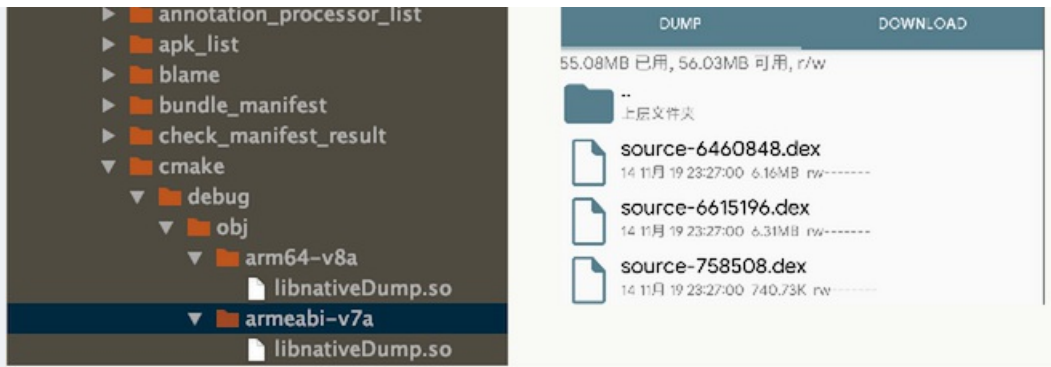
# 7、打开XposedInstaller看已经启用DumpDex插件, 是的话重启手机

# 8、开机后, 打开想脱壳的应用, 不用理闪退, 接着打开data/data/包名查看是否有Dump目录

# 9、进入如果出现下图所示的多个dex, 说明脱壳成功, 否则可能是脱壳失败

# (看是否有报错信息), 或者不支持(比如360加固免费版只支持新版, 不支持旧版)。

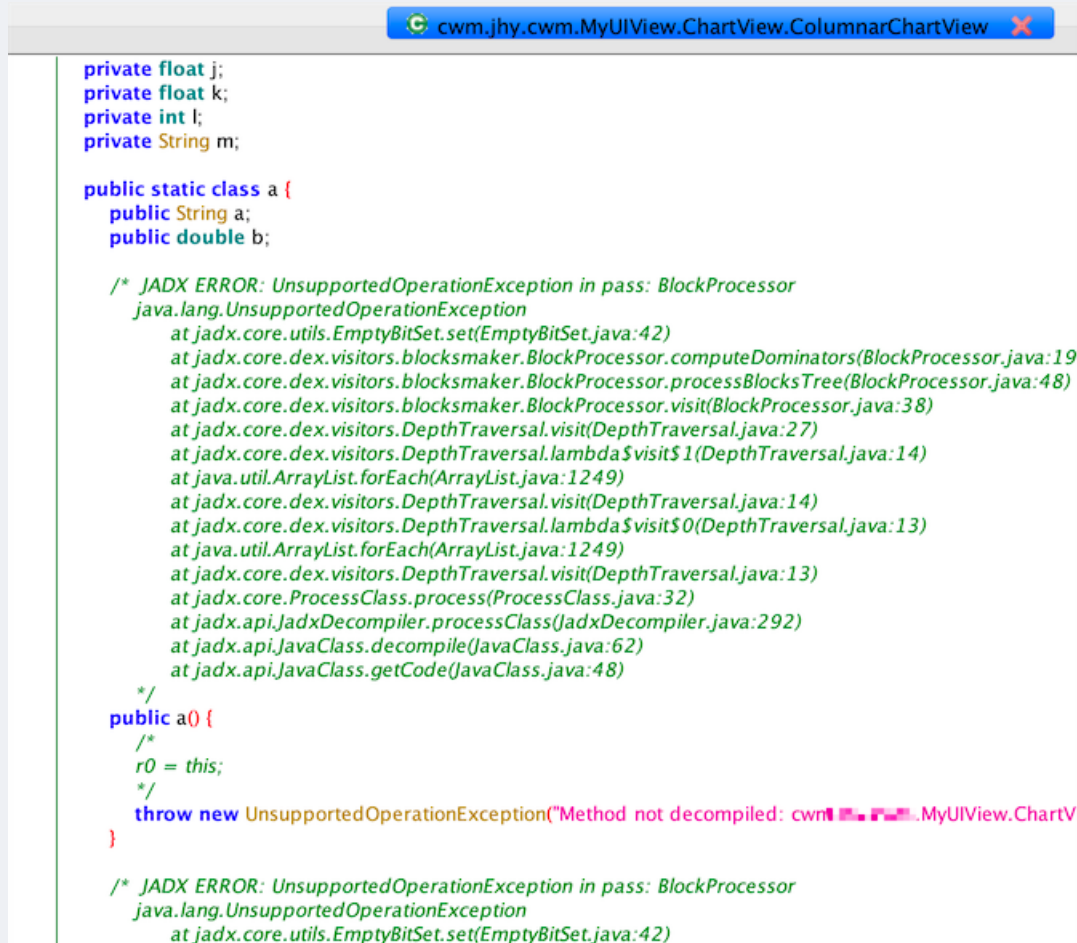




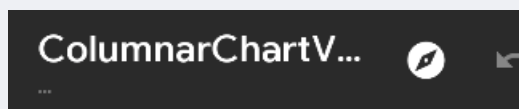
另外，脱出来的dex不一定就可用，比如某个用了「腾讯御安全」的应用：



用jadx-gui打开这的dex，一堆这样的错误：



出现这个的原因是「指令集被抽取」，打开smali文件你就知道了：



```
37 .field private i:J
38
39 .field private j:F
40
41 .field private k:F
42
43 .field private l:I
44
45 .field private m:Ljava/lang/String;
46
47
48 # direct methods
49 .method public constructor <init>(Landroid/conter
50     .registers 4
51
52     nop
53
54     nop
55
56     nop
57
58     nop
--
```

方法指令都被nop(零)替换了，工具党到这里就可以放弃了，要调试so文件。

---

**Tips:** 本节用到的东西，都有给出比较官方的下载链接!!! 你也可以到公众号「抠腕男孩」输入000，回复对应序号下载，谢谢~

---

参考文献:

- [Android打包流程](#)