**APP从抓包到分析总结**

[1、起源 1](#_Toc21187)

[2、环境部署与抓包 1](#_Toc17117)

[2.1 安卓下的抓包 1](#_Toc6768)

[2.2 ios下的抓包 4](#_Toc8600)

[3、 逆向浅析 5](#_Toc31682)

[3.1 加固与脱壳 5](#_Toc6768)

[3.2 关键字检索定位 9](#_Toc6768)

[3.3 Hook浅析与方法汇总 1](#_Toc6768)2

[4、 总结 1](#_Toc12756)7

**1、起源**

最近接到任务，需要分析一些APP的接口，之前没接触过这块，经过这次任务，学到不少东西，以下是一次技术性总结和概括。

**2、环境部署与抓包**

在安卓和ios下都有大量详细的教程，这里只做个人总结和观点，详细步骤过程可参考“详细步骤”链接以及自行百度。

安卓下的简易抓包方案：小黄鸟（HttpCanary）

ios下的简易抓包方案：Stream

2.1安卓下的抓包

## **安卓环境部署方案一：**

## 模拟器 + Charles（Burp、postern可选）

详细步骤：

参考：https://xz.aliyun.com/t/11817

大佬写的非常详细了，此处不再展开，只记录踩坑和个人观点

优点：

1、简单直接，win下直接傻瓜式操作即可

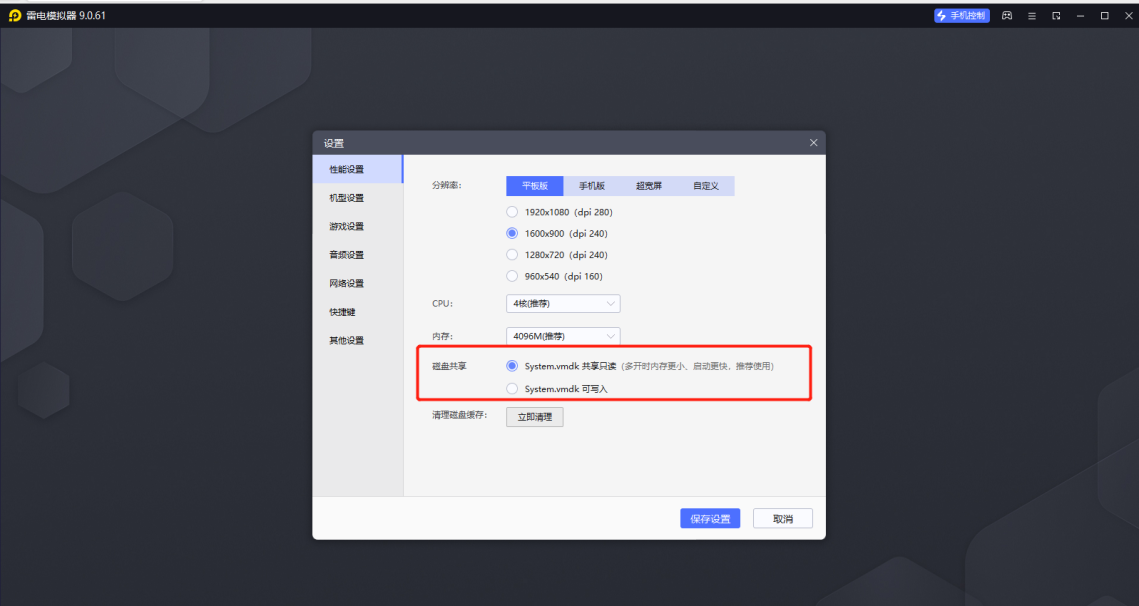
2、灵活，安卓版本、手机型号、地理位置等等都可自定义

踩坑：

1、不同的雷电版本，证书导入的方式有一定差异（**尤其高版本雷电，推荐使用雷电5.x的版本，最新版的坑太多了**）。

解决办法：

安装后最好检查一下，把“System.vmdk可写入”选上，不然会影响后续的证书导入（System目录不可写）



2、部分APP会检测模拟器（安装启动后直接闪退）

解决办法：换低版本APP、换安卓高版本、换真机、配置模拟器参数（手机号、运营商、型号等等）......

模拟器环境配置相关命令汇总：

安卓下常用adb命令：

adb devices #列出连接的设备状态

adb shell #切换至设备中（只有一台设备可直接切入，如果有多台，需指定adb -s emulator-5554 shell）

adb root #切换root用户，如果失败，检查雷电的软件设置-> 其他设置-> Root权限是否开启

adb push C:\temp\d3b09054.0 /sdcard/tmp/ #将d3b09054.0上传至指定目录

adb pull /sdcard/tmp/d3b09054.0 C:\temp\ #将d3b09054.0下载至指定目录

adb install C:\users\admin\Downloads\qq.apk #通过电脑安装指定apk

针对高版本安卓+最新版雷电模拟器的证书导入（临时解决办法，重启后失效）：

mkdir -m 700 /sdcard/tmp/htk-ca-copy

cp /system/etc/security/cacerts/\* /sdcard/tmp/htk-ca-copy/

mount -t tmpfs tmpfs /system/etc/security/cacerts

mv /sdcard/tmp/htk-ca-copy/\* /system/etc/security/cacerts/

cp /sdcard/tmp/d3b09054.0 /system/etc/security/cacerts/ #此处替换为Charles导出pem转换后的.0文件（提前push好）

chown root:root /system/etc/security/cacerts/\*

chmod 644 /system/etc/security/cacerts/\*

chcon u:object\_r:system\_file:s0 /system/etc/security/cacerts/\*

rm -r /sdcard/tmp/htk-ca-copy

命令行直接设置代理（替代安装postern的方式-手动挡与自动挡的区别）：

先切进会话中，adb shell

关闭代理：

settings put global https\_proxy :0

settings put global http\_proxy :0

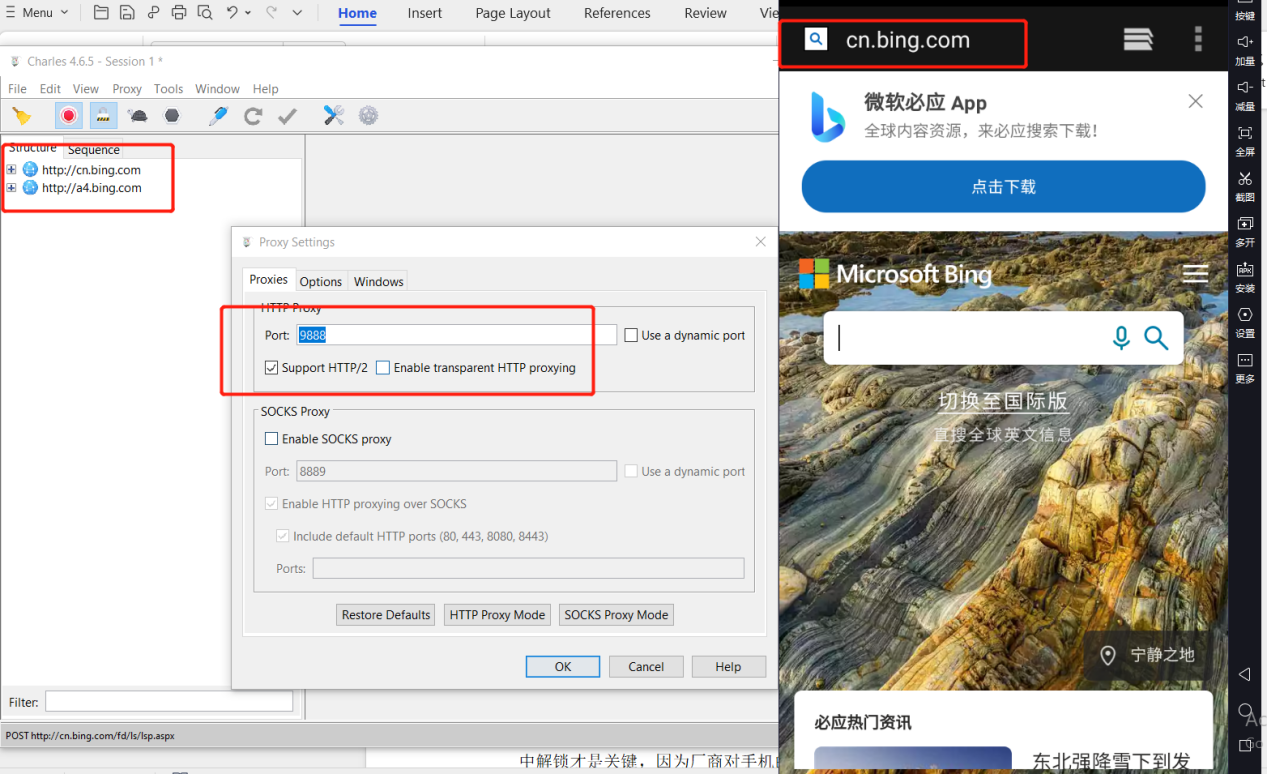
开启代理：

settings put global https\_proxy 192.168.0.103:9888

settings put global http\_proxy 192.168.0.103:9888

改为物理机的IP以及抓包工具的端口（此处使用的Charles的端口）

最终效果如图：



## **安卓环境部署方案二：**

## 真机 + Charles（Burp、postern可选）

详细步骤：

1、解锁刷机：根据手机型号不同，操作步骤也不同。

公司给了个华为nova 4e，以nova 4e来说，需要先解BL锁，比较麻烦，直接50块淘宝找人远程搞，解锁+root，其中解锁才是关键，因为厂商对手机的限制比较多，尤其华为这种麻烦玩意儿，一不小心就刷成砖了，最好是自己编译对应型号的内核，在编译时，就可以根据需要进行配置，例如：设置selinux（hook时会受影响）。

华为的刷机详细过程可参考（其他型号自行百度）：

https://github.com/Coconutat/android\_kernel\_huawei\_kirin970\_EMUI9.1.0\_KernelSU

2、root后，用数据线或配置WiFi adb按照模拟器安装步骤即可。

优点：

1. 稳定！绝大部分APP都能运行，不会出现闪退
2. 对于深入分析学习的，必备。（模拟器的实现与真机始终是不一样的，能做到模拟器做不到的一些操作）

缺陷：

1. 主要是刷机解锁有成砖风险，因此需要谨慎
2. 刷机过程比较繁琐
3. 导入证书时，需要system目录可读写，配置会有点麻烦（对于华为来说）

安卓环境配置总结：

模拟器下的环境安装配置相对真机来说比较简单，虽然过程繁琐，但只要仔细耐心的配置，都不会有什么问题，真机最大的坑在于限制比较多，但配置好以后，真的比模拟器舒服。

**2.1 ios下的抓包**

由于部分app在ios下的接口不同，并且部分app也只能运行在ios下，于是公司斥巨资600块买了部iPhone8，以下步骤均在iPhone8下操作。

ios下的抓包比安卓更简单（对iPhone8来说）

1、WiFi连接：与PC端抓包工具同一局域网

2、下载对应抓包工具的根证书

3、“设置”—“通用”—“关于本机”，往下滑找到“证书信任设置”并打开，找到对应下载的证书，选择完全信任即可（此处是导入的stream的证书）



**抓包总结：**

抓包的主要问题是抓包工具的证书信任，高版本安卓下只认system下的信任证书，用户层导入的证书不再信任（貌似是安卓7以上开始的），因此安卓下必须想方设法把抓包工具的证书导进system下即可。反观ios，其实相对没那么严格（也可能是我测试的iPhone版本不高）。当然，也有部分APP是不走system证书校验的（通过其他方式校验），遇到过，这块不太熟悉，毕竟没搞定。

**3、逆向浅析**

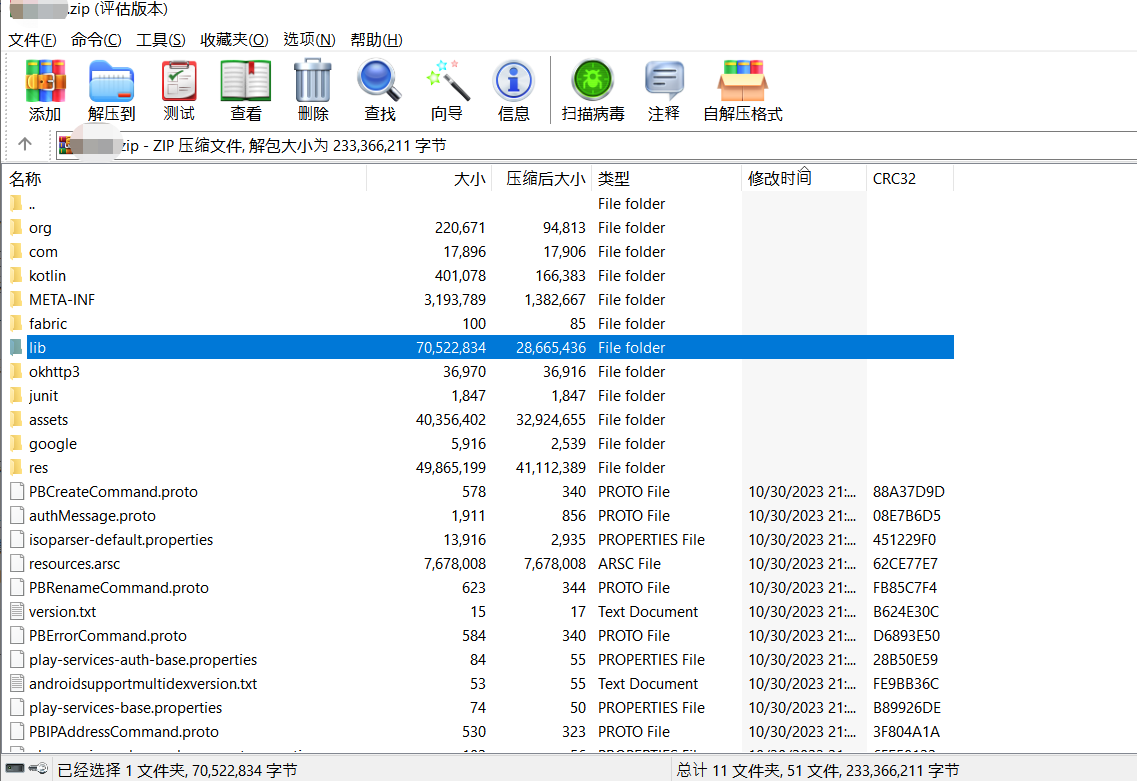
抓包问题解决后，经常会遇到各种加解密的情况，常见的有：AES、DES、3DES、RSA等等加密算法，当想要生成或解密数据时，则不得不对App进行逆向分析，大致可以分为查壳脱壳、定位关键位置、逆向分析。

**3.1、加固与脱壳**

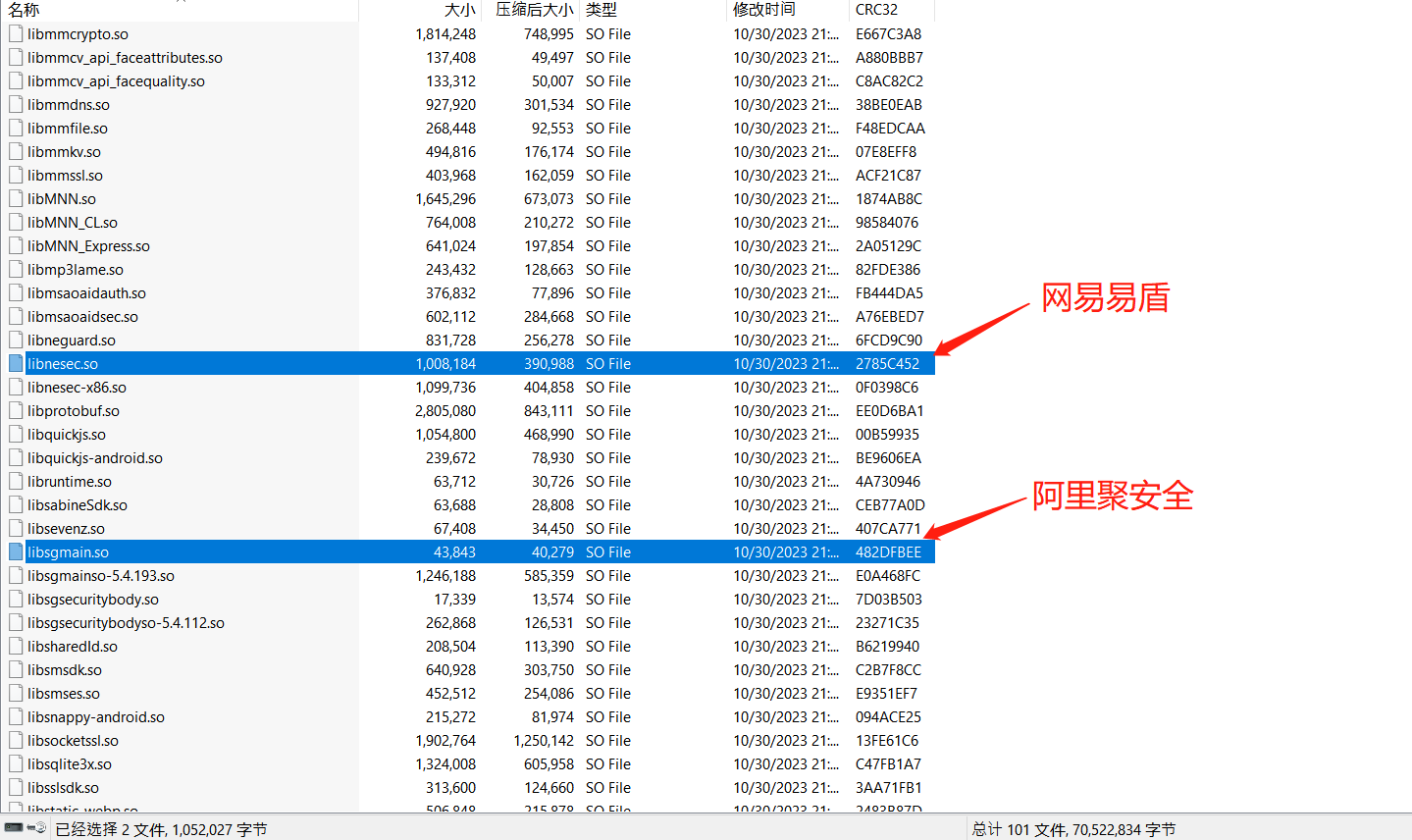
**查壳：**

开始分析之前，还是得从查壳（或者叫加固）开始，相关查壳工具、脚本蛮多的：

APKSCAN-PKID...实现原理大部分都是根据APK中lib目录下的so文件进行判断（关键字匹配），将APK改名为zip，直接打开即可查看加载的库：



不要依赖工具给出的结果，最好手动分析确认，否则遇到不同的情况，将非常懵逼，比如下面这种情况（工具识别出阿里加固，但实际情况不止是阿里）：



出现这种情况的原因是，当APK需要调用阿里支付、阿里认证等功能时，需要引用到相关加固的so文件。因此，通常是根据接口位置、功能点大致判断采用的什么加固，其次，也可通过apk中的一些功能点找到加固，例如：



Apk中的功能点



阿里聚安全的加固用途



网易易盾的加固

**脱壳：**

Java层一般是混淆、某60加固比较常见，当然也有部分app在Native层加固（主流，毕竟核心代码放在Native层增大逆向难度）。接触过的脱壳方式有：

frida-dexdump：从内存中导出dex，用法一搜一大把，此处就不再列举。

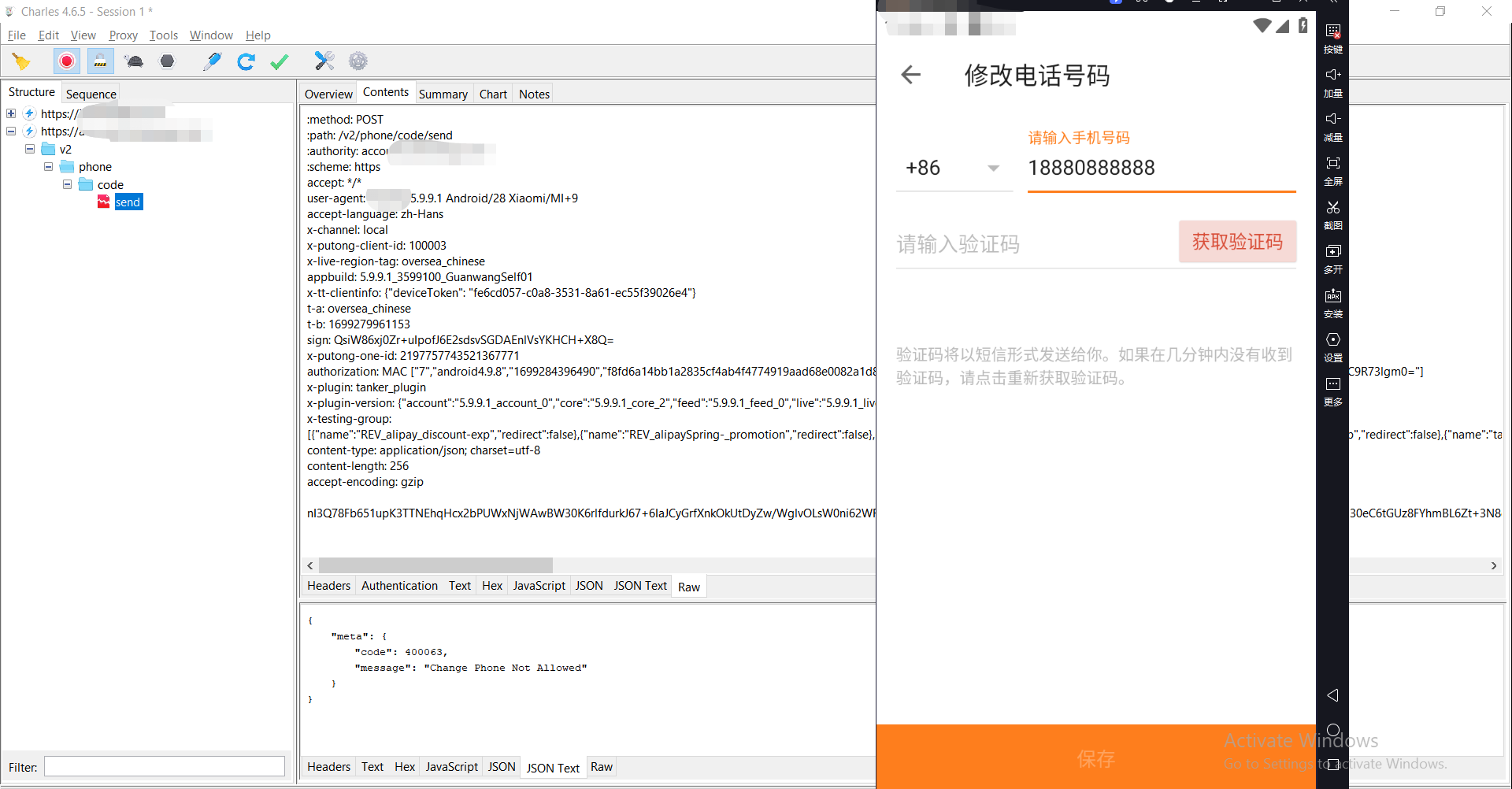
drizzleDumper：原理与上述工具大差不多，毕竟frida-server可能会受到反调试之类的影响，该工具则独立运行。

GG一键脱壳脚本：也是从内存导出，不过是安装apk，采用的lua脚本去从内存脱壳，并且该工具还可直接修改内存数据。

接触到的脱壳方式大部分都是从内存脱壳，但除了这种方式（毕竟不通用，存在很大局限性），但还有不少脱壳方式，像Fart脱壳等很强的方式，这块暂未做到深入性的研究（暂时解决项目问题，后续再学）。

**3.2、关键字检索定位**

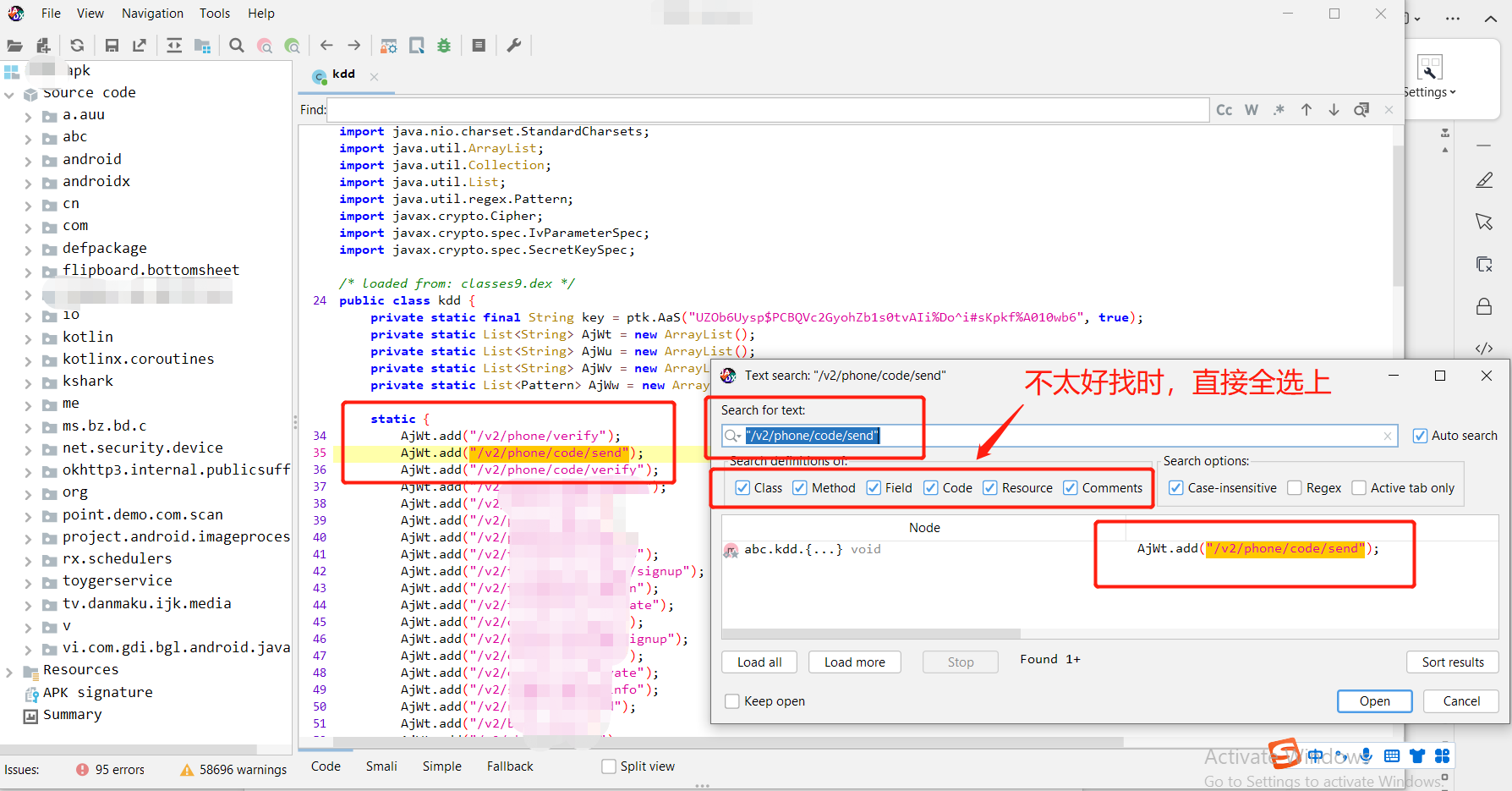
抓到包后，对需要测试的接口/功能点进行分析，举个例子：



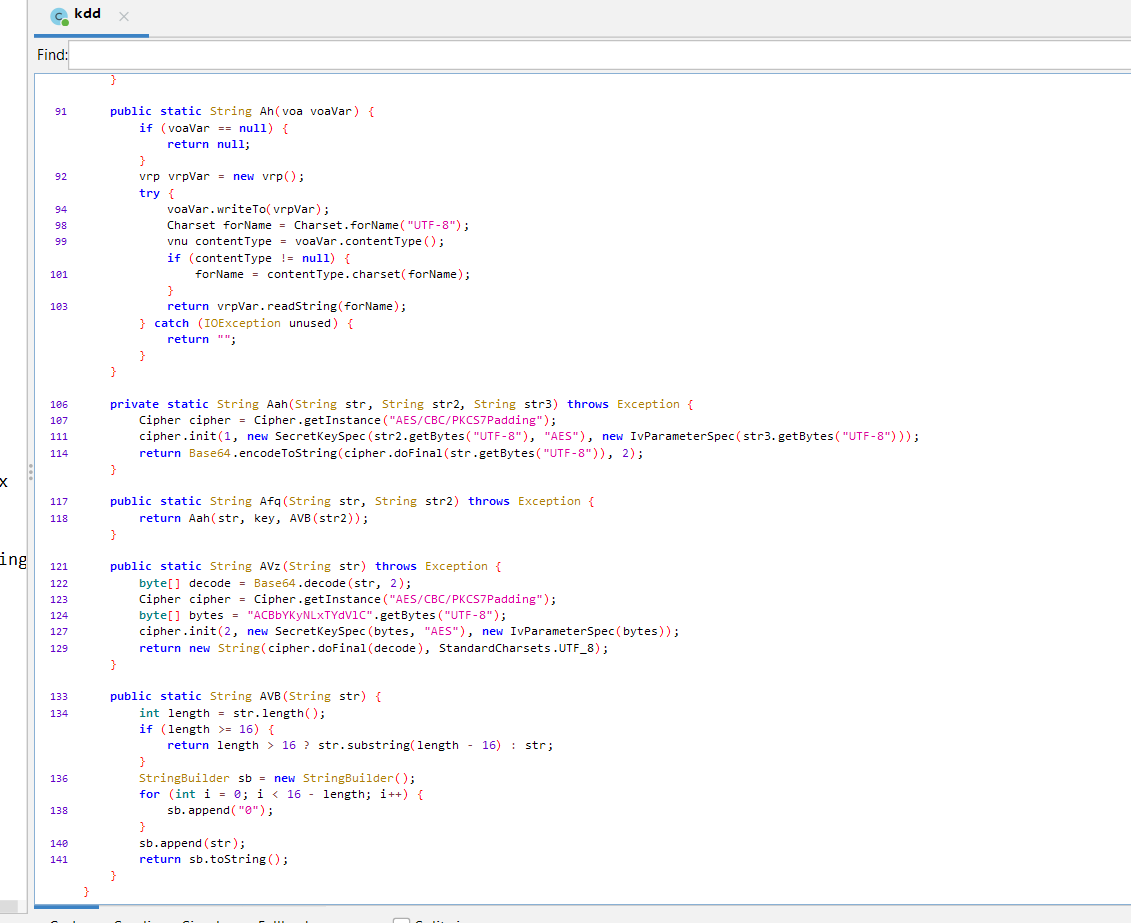
例如“修改电话号码”这个功能点，抓包后发现POST的数据全加密，但可以确定输入的手机号一定在其中，那么就需要根据一些特征去定位代码的实现位置，此处可以检索一些特征：



APK丢进Jadx直接全局搜索：



发现“key”这种关键字以及比较像密钥的字符串后，观察上下代码：

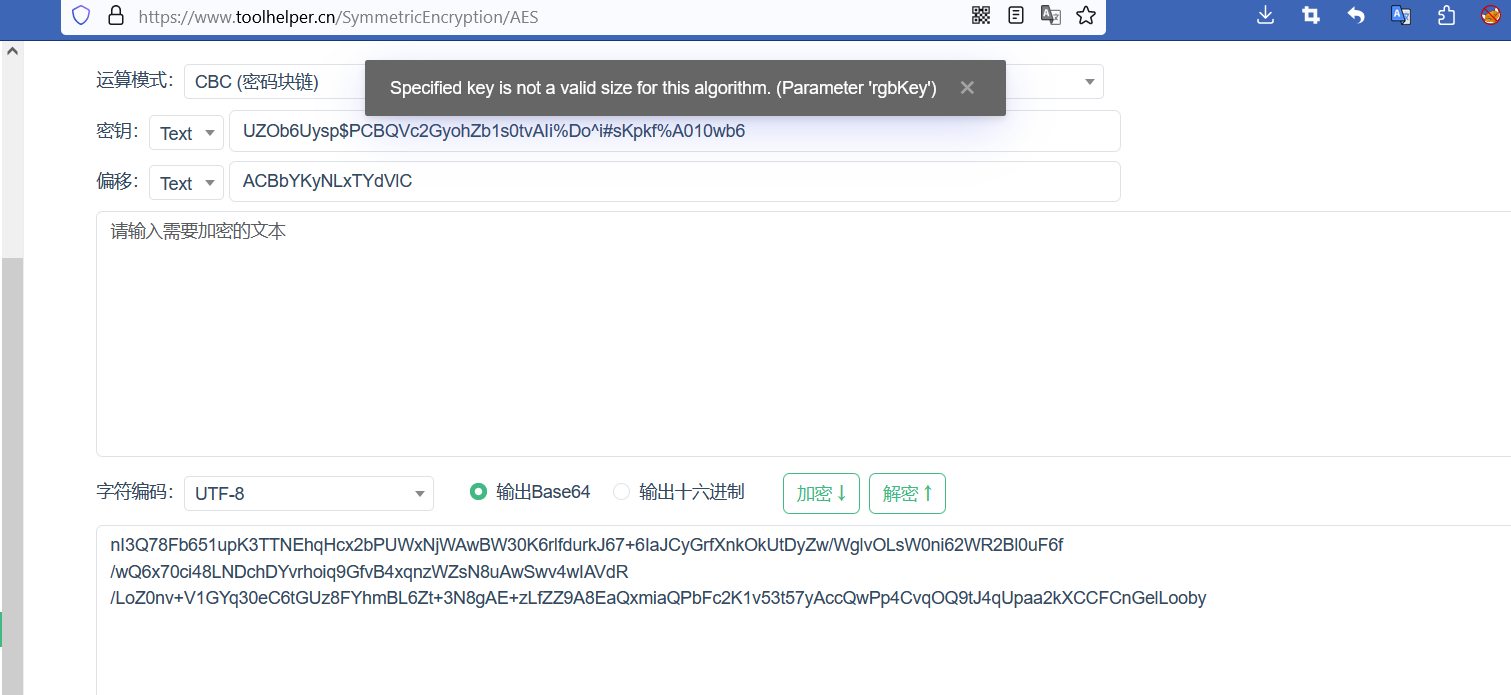


可以看到明显的AES特征，此时可先盲猜一手（不放心可以通过Frida、MT管理器之类工具查看该功能点调用的堆栈-参考后续方法）：

算法: AES/CBC/PKCS7Padding

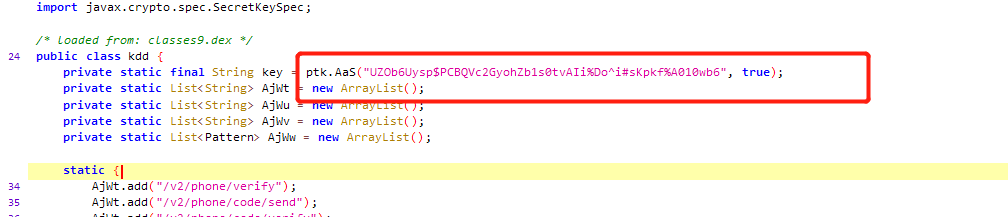
IV: ACBbYKyNLxTYdVlC

Key:UZOb6Uysp$PCBQVc2GyohZb1s0tvAIi%Do^i#sKpkf%A010wb6

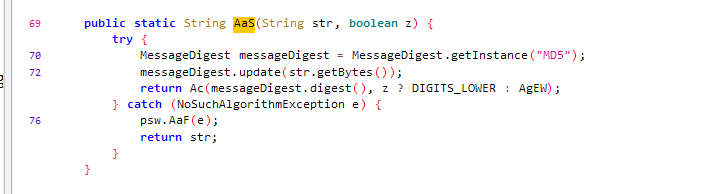


发现解不开（此时可尝试多换几个在线网站）

再次回到Java代码，发现key前面还有个方法调用：



直接跟进该方法，可以看到明显的md5加密：

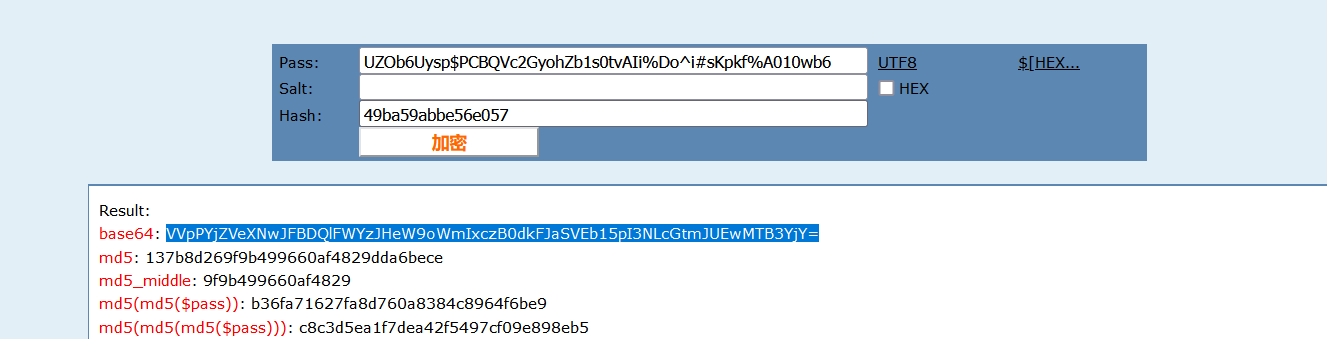


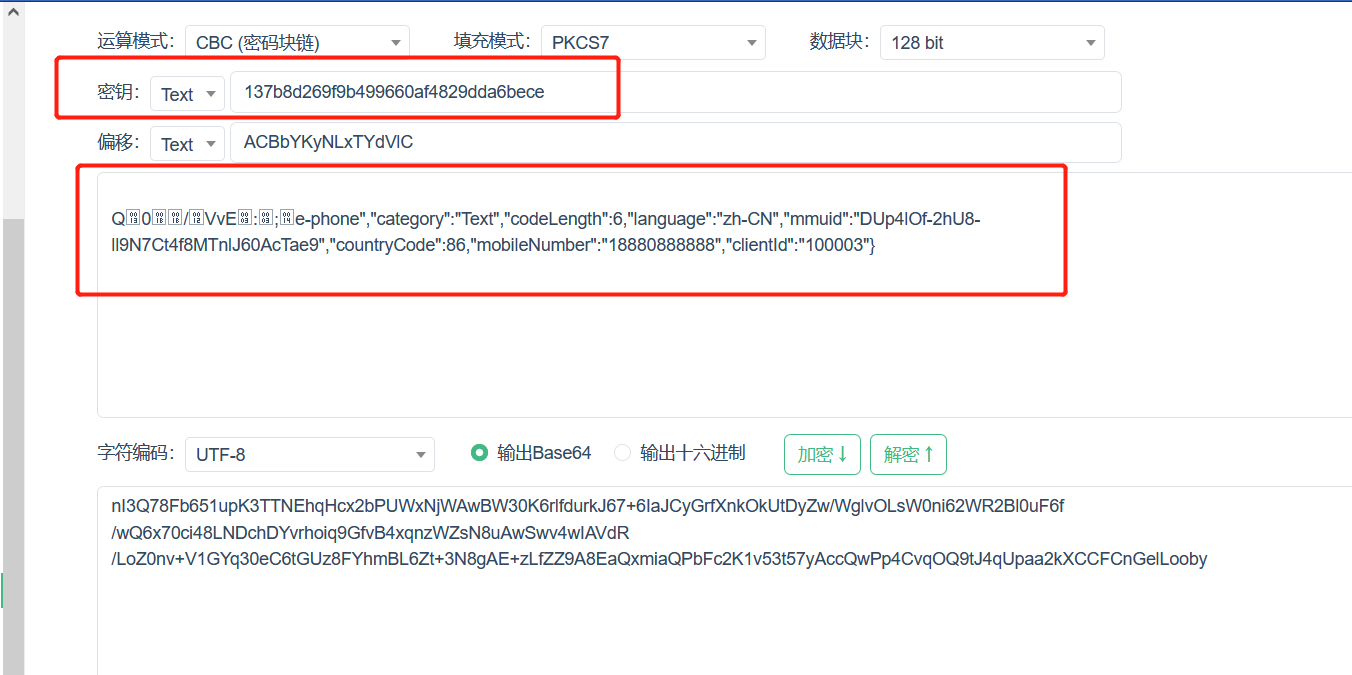
完善一下逻辑：

算法: AES/CBC/PKCS7Padding

IV: ACBbYKyNLxTYdVlC

Key:md5(UZOb6Uysp$PCBQVc2GyohZb1s0tvAIi%Do^i#sKpkf%A010wb6)





可以看到直接解开了（虽然前面还有部分数据乱码，多换几个解密网站即可）

**3.3、Hook浅析与方法汇总**

当我们调用某个功能点时，如果能直接打印出调用的堆栈信息，将有利于更直接、快速、准确的找到调用的位置。

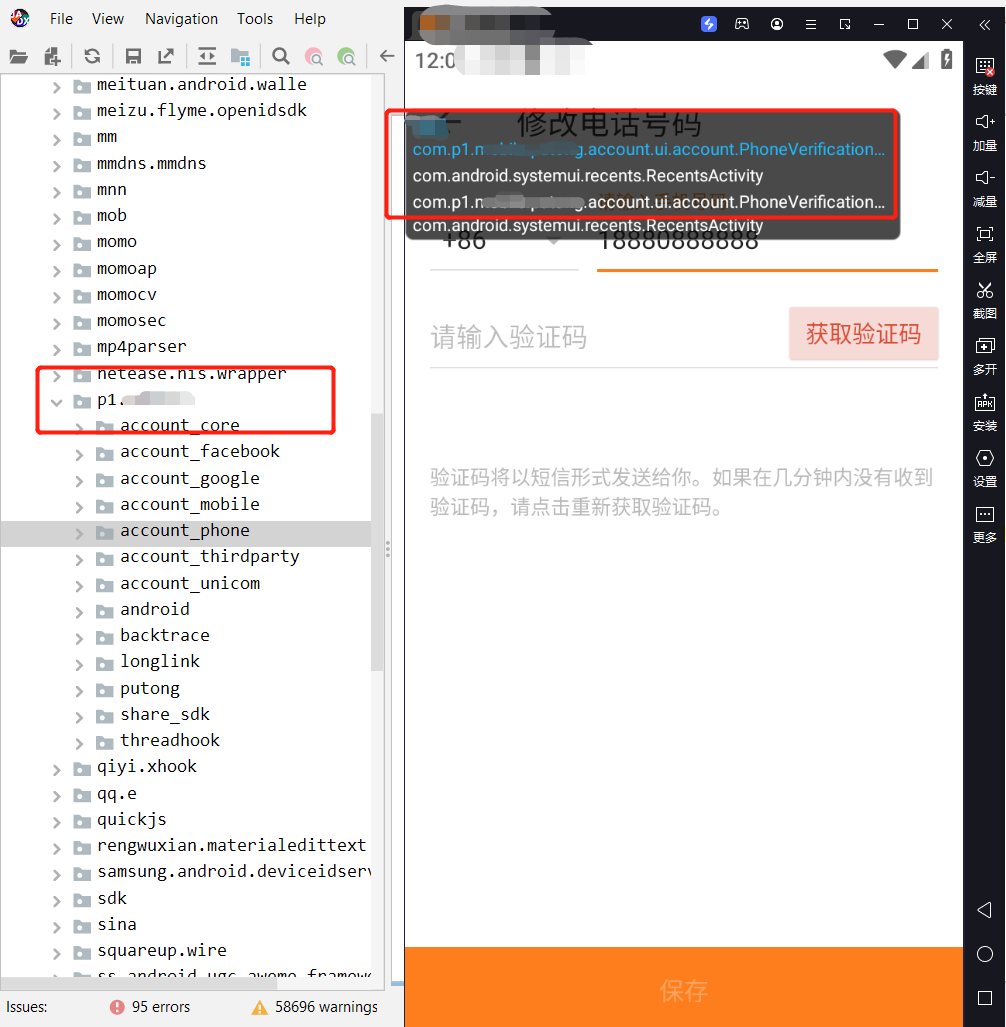
**方法一：**

MT管理器：简单直接，通过监视APP调用的Activity实现，缺陷是监视不完全。





寻找对应代码位置：



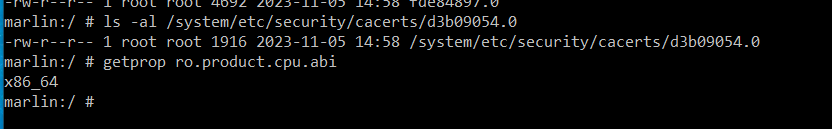
**方法二：**

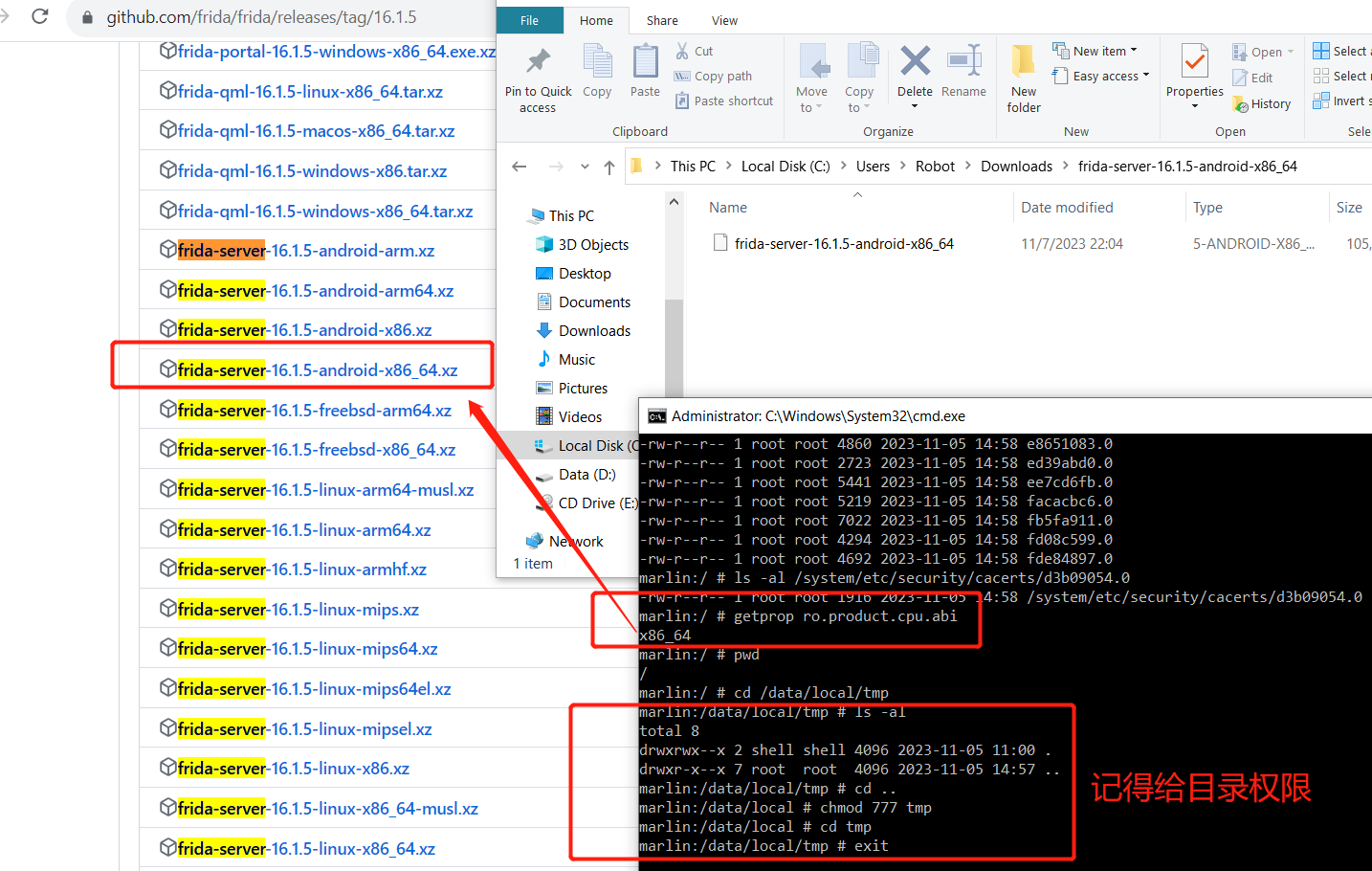
Frida：Hook相关代码位置，原理是将自定义的JS代码注入到APP中，因此有两种方式，一种附加、一种直接启动（有的APP在启动或运行时有反调试，可以切换一下这两种方式，也有概率绕过）。

准备工作：

1、根据模拟器版本选择对应的Frida-Server（下载地址：https://github.com/frida/frida/releases）

可使用“getprop ro.product.cpu.abi”查看当前安卓架构，下载、解压、Push。



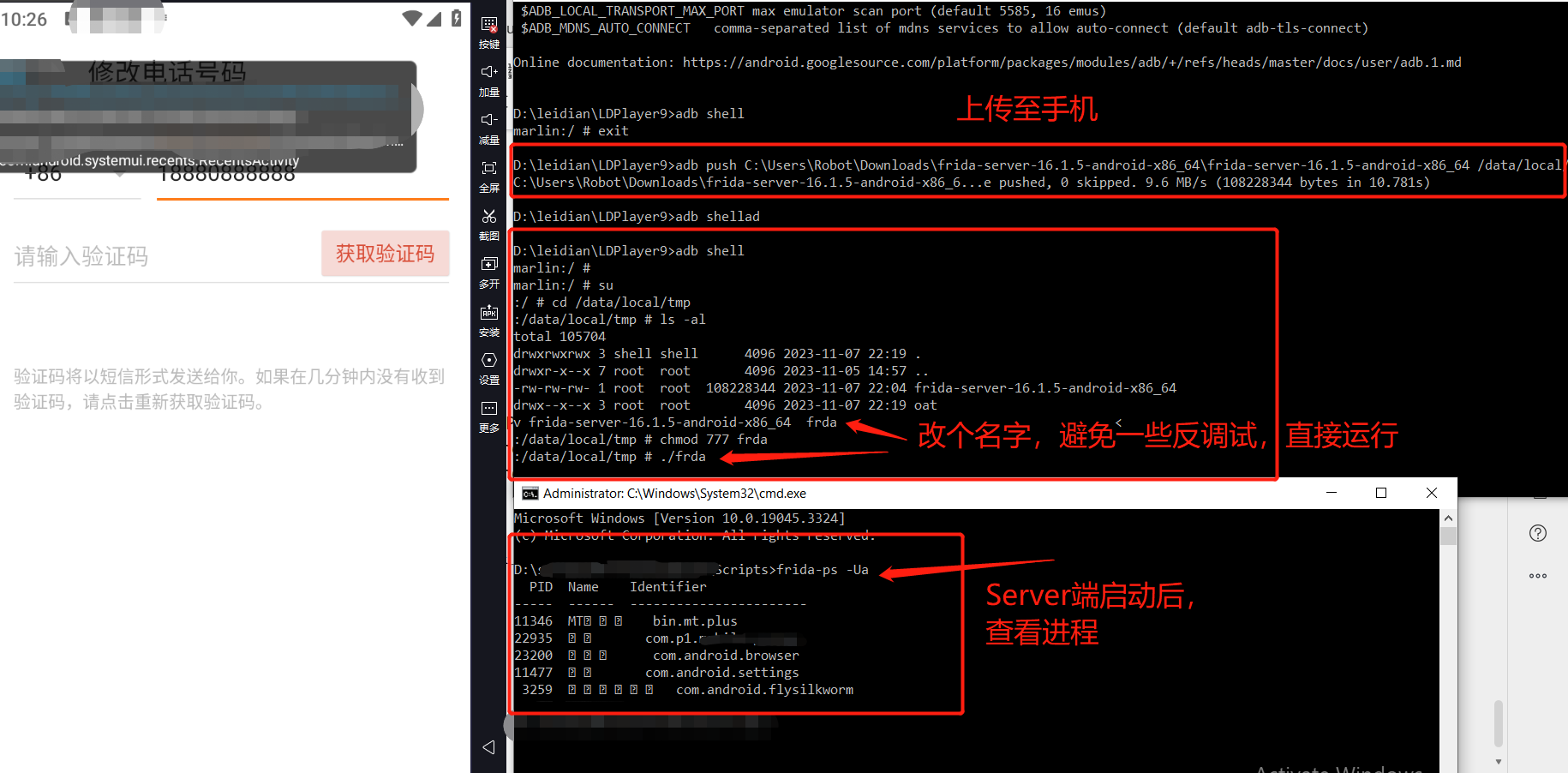


2、Python安装Frida：

python -m pip install frida

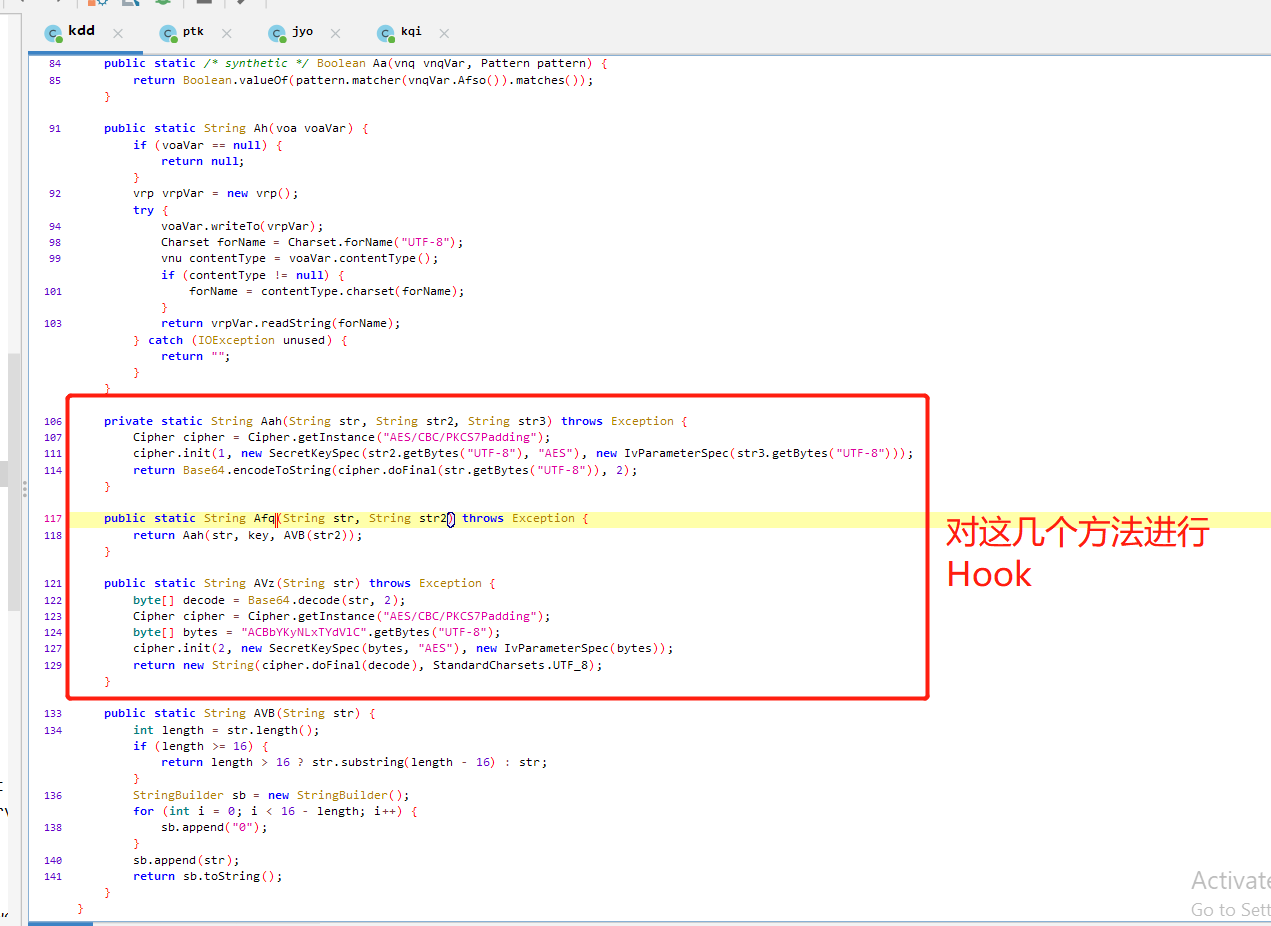
python -m pip install frida-tools

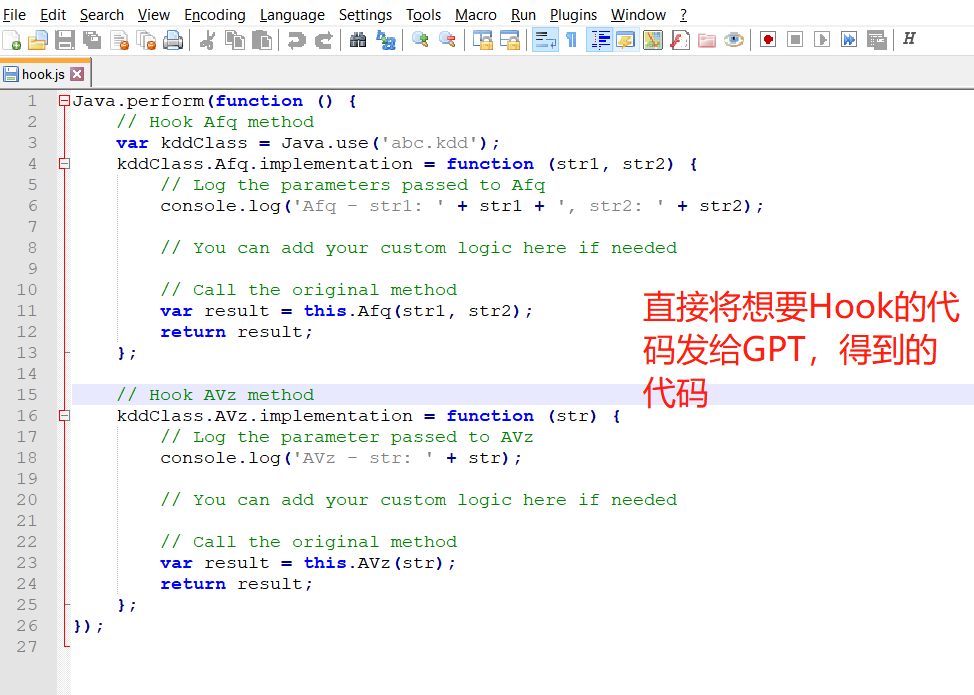
3、安卓运行Server端，物理机列出进程：



到此，准备工作结束。

编写注入代码：

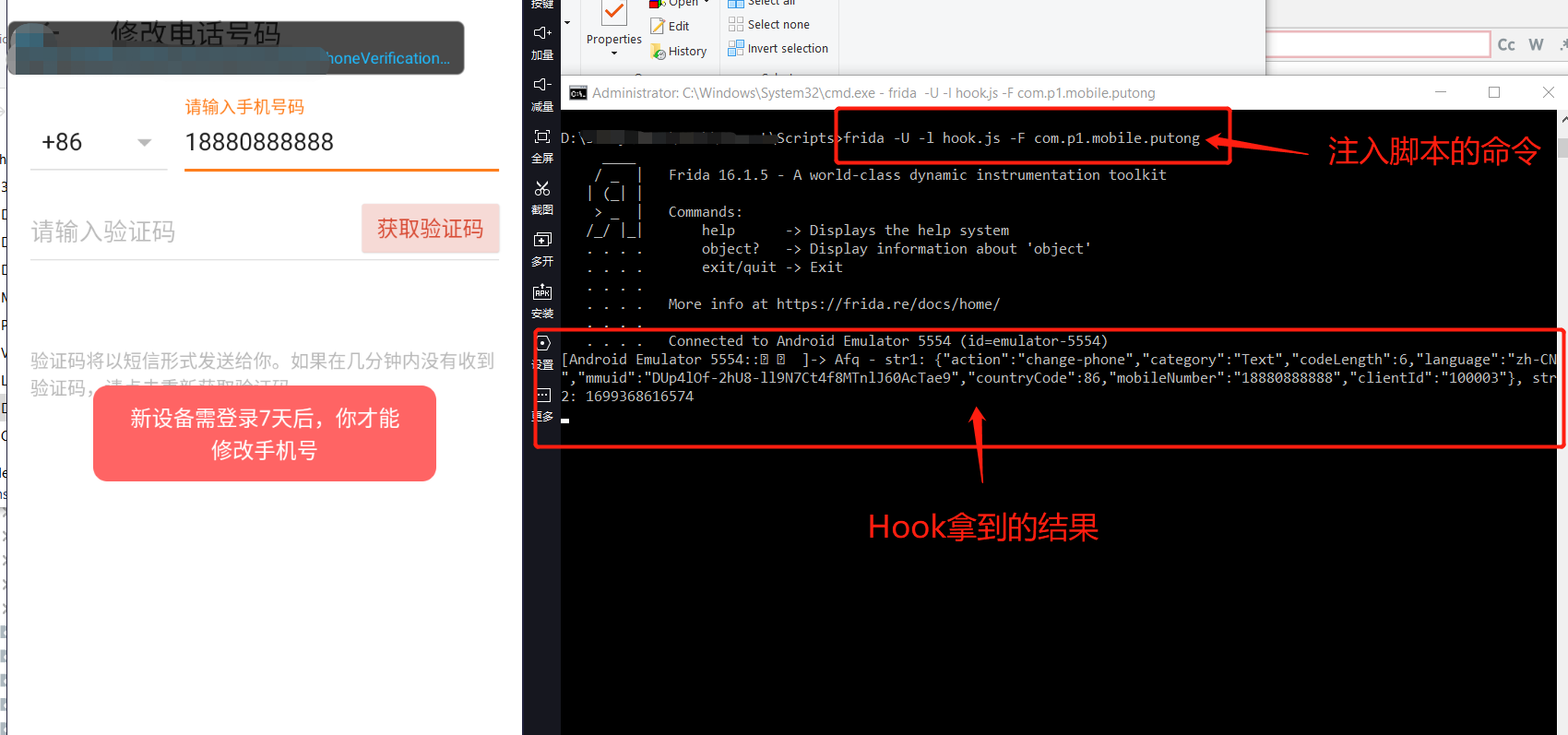




使用frida进行Hook：

APP启动时注入：frida -U -l hook.js -f 包名

APP运行后注入：frida -U -l hook.js -F 包名



**方法三：**

算法助手：对常见的一些加解密方法进行Hook，也支持自定义Hook、行为记录等等功能，使用时，基本秒杀大部分的Java层加密。但环境配置稍微繁琐一些（依赖Xposed框架）。

相关参考链接：

https://blog.csdn.net/u014645251/article/details/119030982

个人比较喜欢抓包+jadx先行分析一波，再上算法助手，配合frida快速定位堆栈，找到关键位置。

**4、总结**

总的来说，对APP这块可以说是刚入门吧，不论是脱壳、Native层逆向还是frida的Hook，都不深入，只能说勉强应付一些这项任务，所以得继续往深入的学（缺乏系统性的学习）。其次，太久没写技术文章了，个人感觉非常乱，不明白、不正确的地方望指正。