

**课 程 实 验 报 告**

**课程名称： 逆向工程分析技术**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2024年月日**

**网络空间安全学院**

1. **实验准备**

Redmi Note 4 (Android 6.0)

IDA Pro 7.7

jadx 1.4.7

MT管理器

Android Studio Iguana | 2023.2.1 Patch 2

adb 1.0.41

1. **实验内容**

参考文档“Android程序逆向分析示例.pdf”中描述的分析过程，下载附件，对其进行逆向分析，获取正确的flag。

1. **实验目标**

（1）熟悉Android程序逆向分析工具及使用方法；

（2）掌握Android程序静态分析和动态分析过程；

（3）理解程序逆向分析过程的复杂性。

1. **分析方法和过程**

使用jadx反编译CrackmeTest.apk，分析AndroidManifest.xml，获取程序启动入口Activity（如图1所示）。

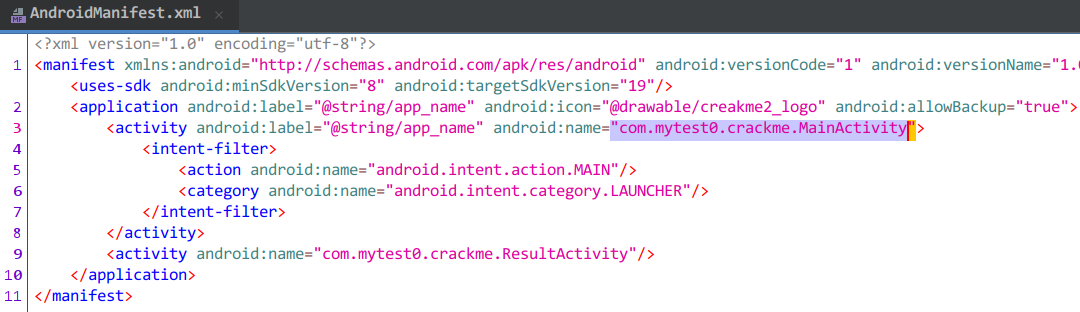


图1 入口Activity

分析com.mytest0.crackme.MainActivity反编译java代码（如图2所示）。



图2 MainActivity反编译代码

由反编译代码可以分析出，程序在启动时加载libcrackme.so，libcrackme.so中包含native方法securityCheck。程序的btn\_submit按钮绑定onClick，当按钮按下时，获取输入框输入字符串，调用native方法securityCheck进行校验，校验结果为false时显示”验证码校验失败”，校验结果为true时启动ResultActivity，ResultActivity创建后显示”密钥正确”（如图3所示）。

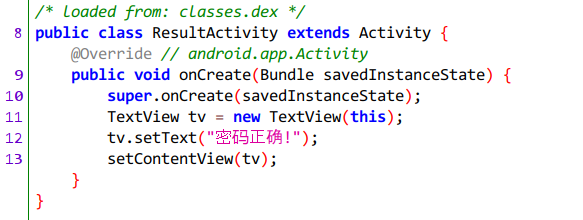


图3 ResultActivity

经过上述分析可知，程序的输入校验位于libcrackme.so中的securityCheck函数，使用IDA Pro分析libcrackme.so。

定位到Java\_com\_mytest0\_crackme\_MainActivity\_securityCheck函数，使用反编译查看分析函数逻辑，手动将第一个参数a1类型修正为JNIEnv \*。



图4 securityCheck反编译

由图4的反编译代码可知，第19行获取函数调用时的参数字符串v5，与字符串off\_628C进行逐字符校验，校验成功返回true,校验失败返回false。字符串off\_628C的内容为”wojiushidaan”，输入该字符串测试，程序显示检验失败。猜测flag字符串在运行时解密。

对securityCheck函数下断点，使用android\_server调试器附加进程进行调试。观察到调试器附加后程序立即退出，因此猜测程序包含反调试逻辑。

根据loadLibrary动态so文件的过程，so文件被加载至内存后，首先运行.init\_array中的函数，随后运行JNI\_OnLoad函数，猜测反调试部分在.init\_array或JNI\_OnLoad中，对.init\_array中的函数以及JNI\_OnLoad下断点进行调试。

由于.init\_array与JNI\_OnLoad是在程序启动时调用，因此在程序运行后附加无法调试该部分内容，需要在程序启动后调用loadLibrary方法前完成IDA调试器的附加，因此需要对程序的java层代码进行调试，在调用loadLibrary前将进程断下并附加IDA调试器。

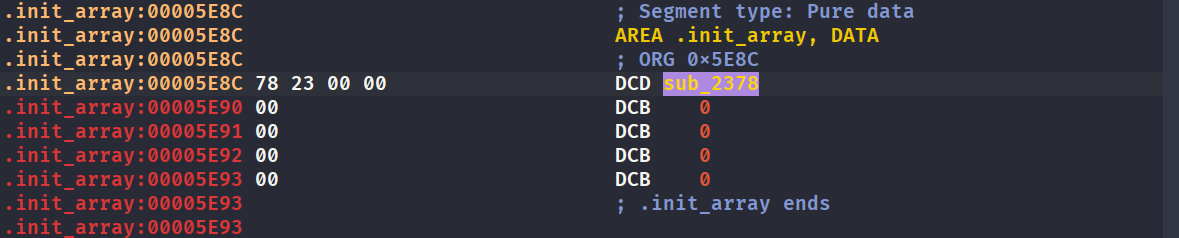


图5 .init\_array

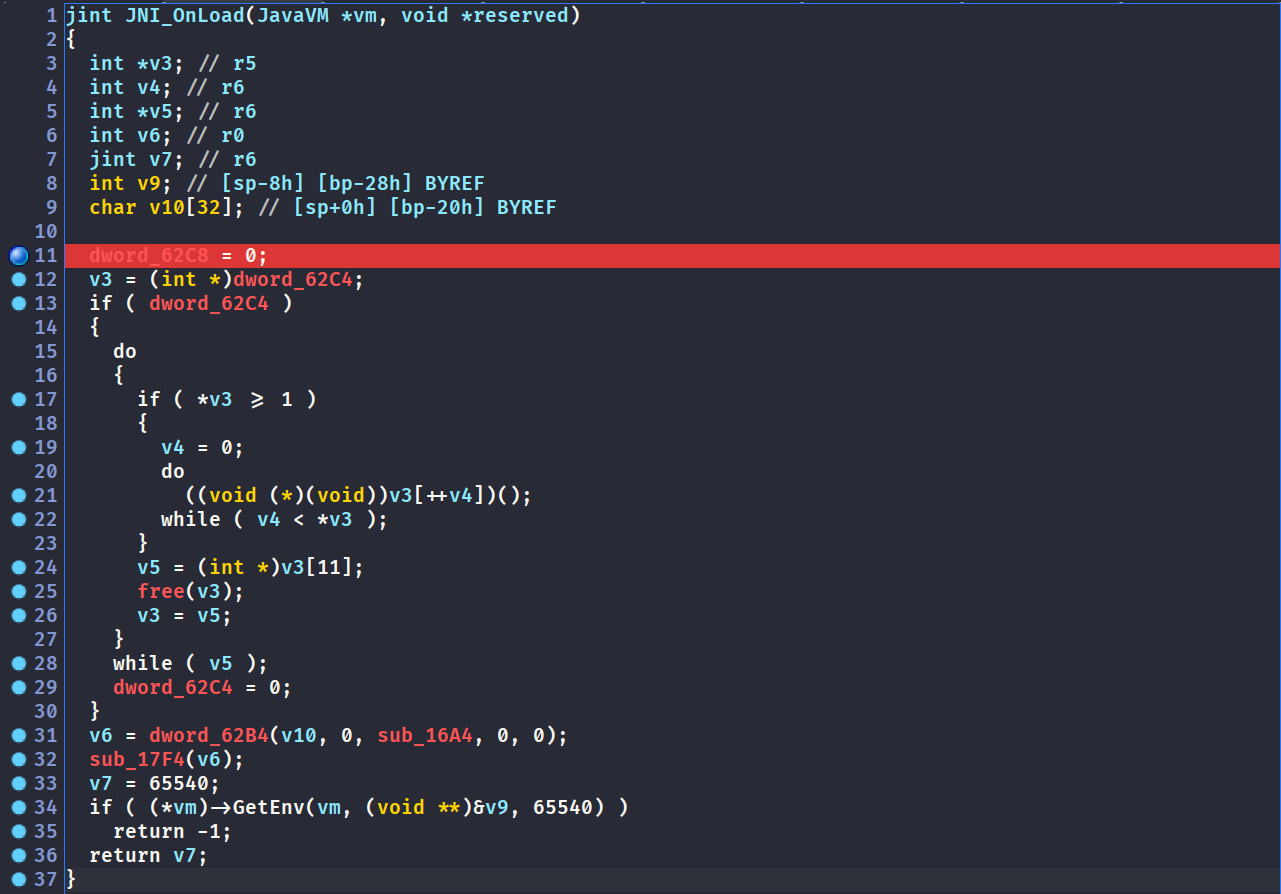


图6 JNI\_OnLoad

CrackmeTest.apk程序默认未开启调试，需要修改AndroidManifest.xml在application中添加android:debuggable=”true”并重新打包签名，使用MT管理器反编译修改AndroidManifest.xml，并重新打包签名（如图7所示）。

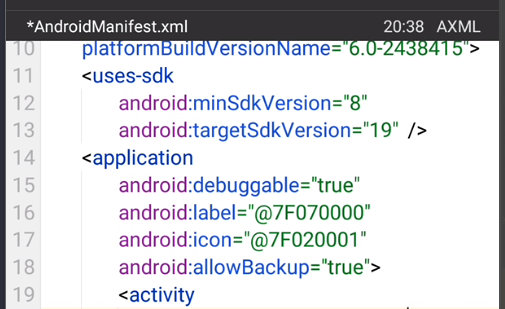


图7 修改AndroidManifest.xml

java层smali代码调试使用Android Studio，调试模式选择Java Only，对loadLibrary下断点。

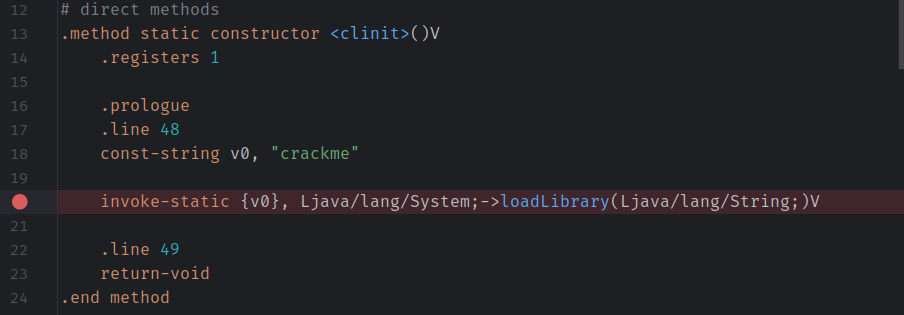


图8 Andorid Studio调试smali

程序启动在loadLibrary断下后，使用IDA附加到进程，Andorid Studio继续运行，程序加载libcrackme.so，IDA在.init\_array断下。

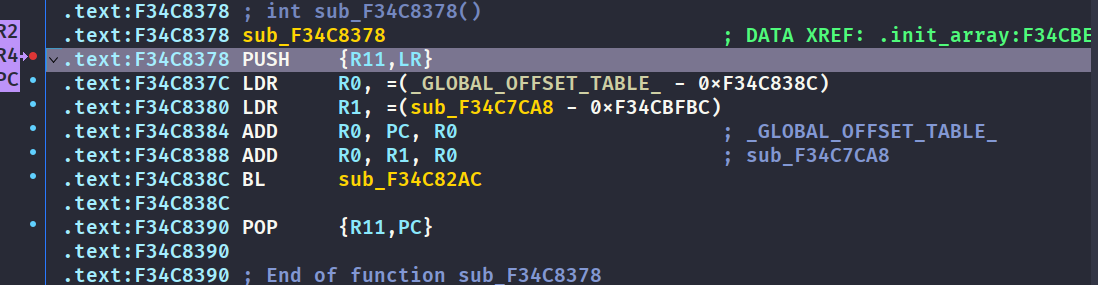


图9 .init\_array断下

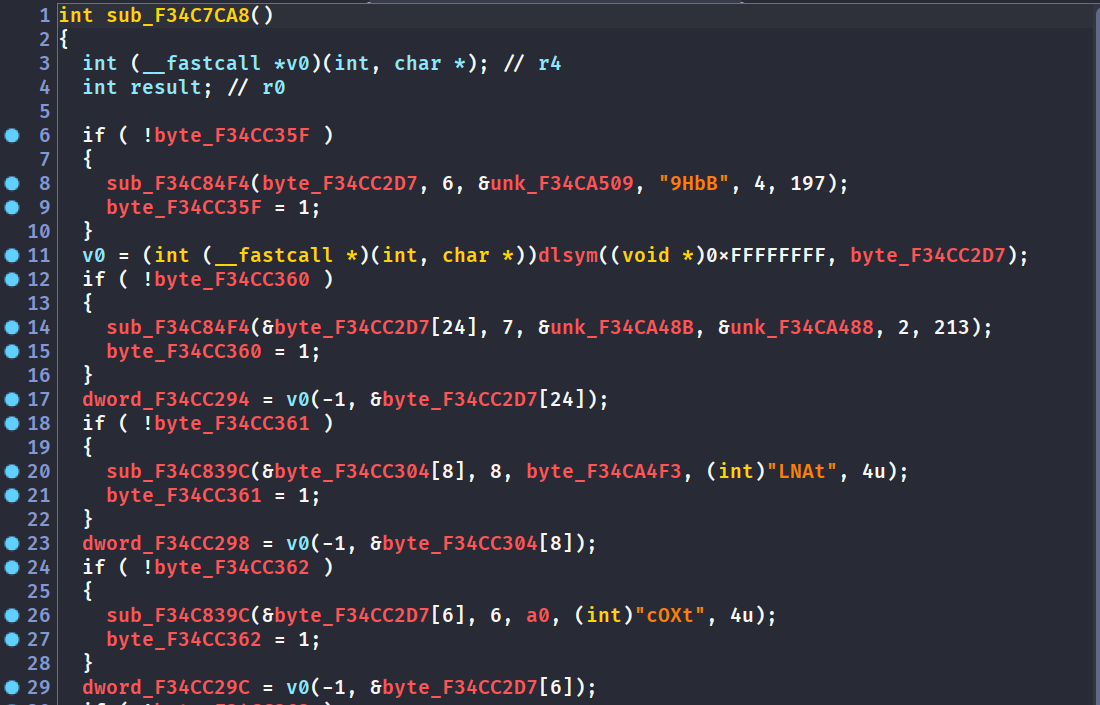
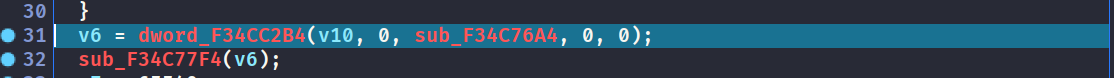


图10 字符串解密

跟踪分析后发现在.init\_array中进行了大量的字符串解密操作，解密得到的字符串均为libc中的导出函数名，因此可以猜测此处在进行函数导入操作，无反调试操作，反调试操作应该在JNI\_OnLoad中。



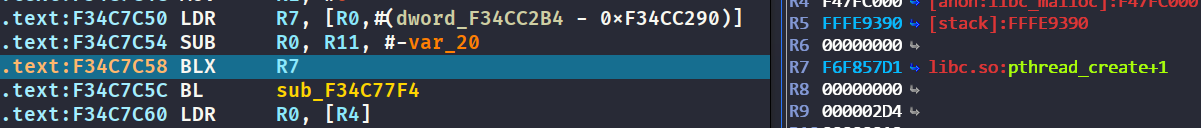
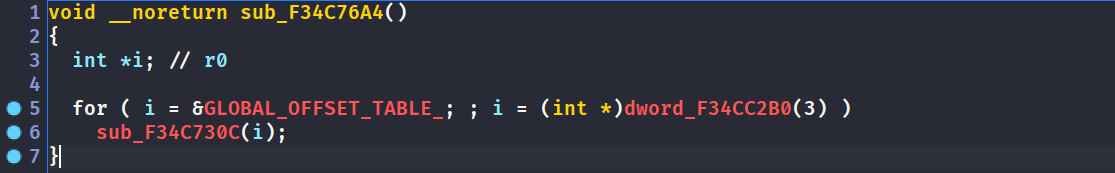
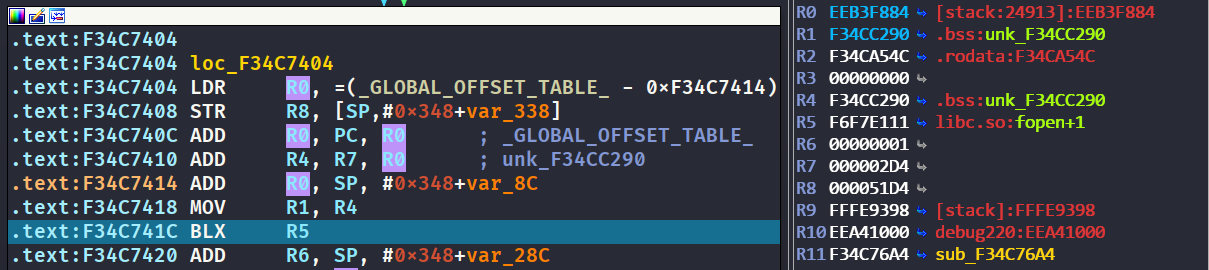


图11 JNI\_OnLoad调用pthread\_create

继续运行，程序在JNI\_OnLoad断下，分析函数执行过程，观察到调用pthread\_create创建线程，猜测反调试逻辑在线程过程函数sub\_F34C76A4中。





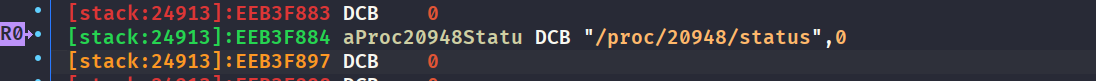
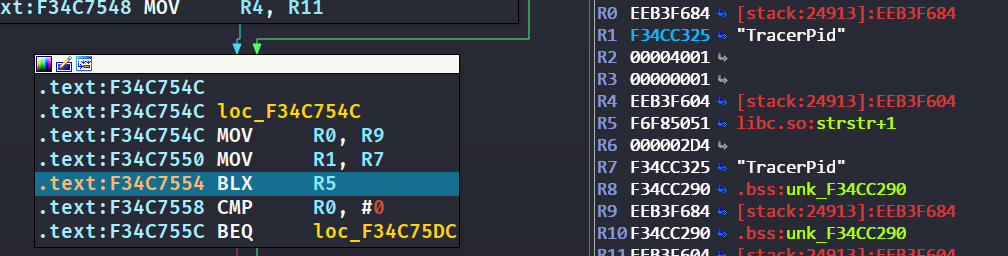


图12 反调试线程fopen



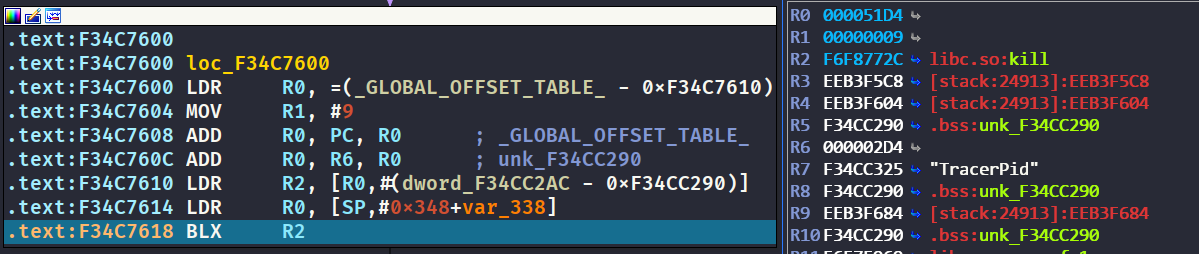


图13 TracerPid检测

如图12、13所示，在新创建的线程中循环去取进程status文件，检测TracerPid的值判断进程是否被调试，被调试后使用kill结束进程。将条件跳转指令BGE loc\_1600修改为NOP，跳过kill分支的执行即可完成反调试（如图14所示）。

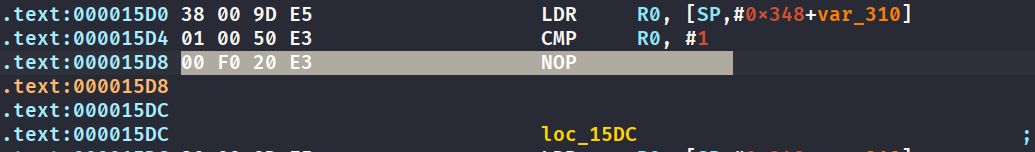


图14 NOP跳转指令

保存修改，将修改后的libcrackme.so重新打包到apk文件中并重新签名。

重新运行程序，使用IDA附加调试，程序未退出，成功去除程序的反调试。对secruityCheck下断点，点击按钮后程序在securityCheck断下。

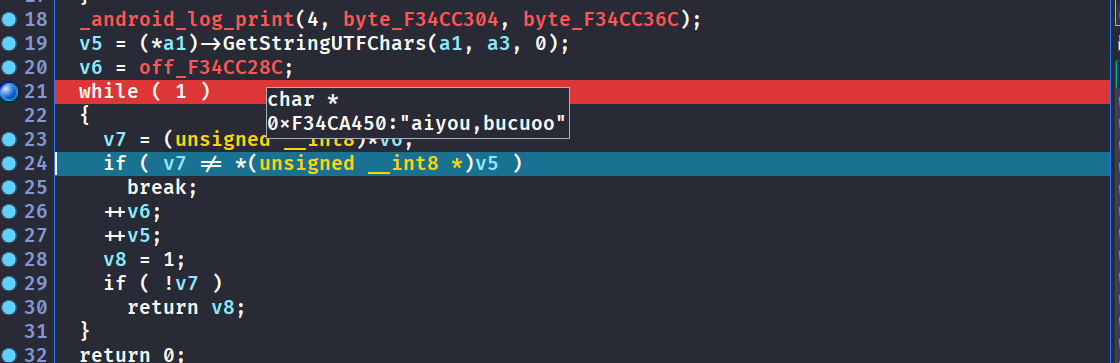


图15 securityCheck

如图15所示，很明显，flag为”aiyou,bucuoo”，输入flag校验，如图16所示，密码正确。

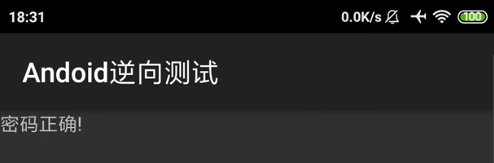


图15 密码正确

1. **实验结论**

实验中使用多种Android逆向程序分析工具对目标程序进行分析，通过静态分析程序逻辑以及对程序动态调试，去除程序的反调试逻辑，最终获取到正确的flag：”aiyou,bucuoo”。

1. **课程学习心得**

本次实验中，针对CrackmeTest.apk程序，学习并使用了jadx、Android Studio、IDA Pro等逆向分析工具，通过静态分析和动态调试，破解了程序的反调试逻辑，并通过动态调试获取到了正确的flag。

通过对实验样例程序的分析，学习和掌握了Android程序的逆向分析技术，学习了如何识别并绕过程序的反调试，了解了Android程序逆向的基本过程和技术复杂性，对软件逆向技术有了更深入的了解。