《电子证据分析》课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 实验名称 |  | 姓名 |  |
| 班级 |  | 学号 |  |
| 实验日期 |  | 指导老师 |  |
| 实验环境  (实验软硬件要求) | 安卓模拟器、ida、Ollyice、APKTool、dex2jar-2.0、jd-gui、signapk | | |
| 实验目的 | （给出本次实验所涉及并要求掌握的知识点）  *掌握安卓apk文件的文件签名，以及对安卓apk文件的内部文件具有一定了解*  *掌握修改安卓apk程序的基本思路和方法，并能够应用于实践*  *培养学生在遇到问题时能够自己思考并且解决问题的能力* | | |

|  |  |
| --- | --- |
| 实验内容 | 安装rev100软件，并且获得口令和flag，之后尝试修改口令和flag  安卓安装apk需要对应的签名，如果没有进行签名是会无法安装的。    但是安卓不会对签名进行验证，所以直接使用签名工具即可。    获得apk为signed.apk    但是假如java版本太老了，就会出现签名问题，安装不了咯~    重新安装java，这次使用了老师提供的java版本，就成功了。    没有报错信息。    用任意的模拟器安装这个软件，    打开后，得知这个软件就是要猜口令，猜对了口令就会提供给你flag    很显然，暴力硬猜不现实，需要对这个apk文件分析，直接将其丢进winhex，进行查看。  发现是50 4B，所以可以确定这个apk其实也是可以用zip压缩包来打开的。    在宿主机上对没有签过名的文件进行改后缀之后再解压出来一个文件夹    内容如下：    根据老师的讲解，里面文件夹里的class.dex就是所有编写的java代码，但是已经成了dex文件，所以要想看明白里面的内容还需要使用工具转换成对应的jar文件。    使用老师提供的工具即可。    下面的批处理命令就可以实现将dex文件转换为jar文件    生成的文件就是刚刚上面图片的那个jar文件    再使用对应的反汇编工具就可以看到内容        这里的MainActivity.class就相当于是应用的主要活动，可以看到这里有检测输入的内容是不是和“EYG3QMCS”一致，一致就会返回Flag      在手机中尝试输入EYG3QMCS：    获得了flag    这里我们已经得到了口令和flag，那么如果想要修改口令跟flag，将它们改成自己的名字，怎么办呢？  一个想法是直接在刚刚那个反汇编的工具上修改，但是很显然这个想法是没有任何用的，这工具不支持修改……  因此，这里就使用APKTool这个工具，首先将apk文件（我用的是没有签过名的apk，后面还需要签一下名）复制到对应的APKTool文件夹下。    然后启动cmd，输入 apktool.bat d rev100    稍等片刻，就能看到所有的反编译的文件了    就是上面的rev100.out文件  内容如下    smali文件就是smali汇编  可以看到这个文件的格式和之前的jd-gui显示的那个jar文件的内容是类似的      而ctf里面就是有一个crackme的文件夹    内部都是smali文件    经过和之前的gui的反汇编界面比较，发现也是具有一定的相似性。    因为之前是在MainActivity.class文件里，所以，很有可能，MainActivity.smali文件里面就有我们需要改的内容——EYG3QMCS  但是打开后，发现内容十分奇怪，看不明白，里面也没有要找的字符串    那么换一个地方找找，查找MainActivity$1.smali，在这里我直接找到了对应要修改的字符串，修改一下之后，保存    改成自己的名字的拼音了……    然后，直接保存退出就行  保存好之后，然后重新用APKTool.bat命令重新编译一下就行了    稍等片刻，内容就在这里了      然后重新签名，并且安装测试      输入shiyangyi，就发现有flag了（后面换了一个模拟器，之前那个模拟器忽然退掉了，所以就变了）      就说明修改成功了  接下来是修改flag。其实也是跟修改口令类似的操作。  在图形界面里发现了疑似存flag的文件，放的是flag的ascii码值，所以，就进入apktool文件里，找到了，就是这个，这个是用的十六进制来表示对应的ascii码值    同样的，计算我的姓名的ascii码值，然后改进去。  shiyangyi：    改好后重新编译一下，输入密码之后，flag也被修改掉了    这样就完成了这第一个实验了。  安装CTF-100这个软件，并且获取flag。    安装后，发现这个就是要求点击按键进行爬楼梯，爬到一定数量，就可以获得flag了。    因为模拟器支持模拟按键    所以只要不嫌麻烦，大可以这样……  但是这样也还是太慢，虽然一直按着空格要比不断点击快很多，但是还是很慢。  那就得要看看源代码，然后破解了。首先下把这个软件反汇编看看代码是怎么样的。  一样的，改后缀为zip，然后解压缩文件    然后将里面的 classes.dex 文件转成jar。      然后用图形化界面的 jd-gui 来查看    得到的内容如下：    在这里看到了这个apk应用的逻辑，这里面设置了两个参数，一个是 has\_gone\_int 、一个是 to\_reach\_int 。分别对应的是已经点了的次数和要点的次数。在初始化的时候，将那个显示flag的按钮设置为不可按，然后初始化已经走了的次数为0.随后随机生成一个数，这个数得先要是正数，然后得大于5并且之后还要模32，之后再乘以16384，这样的出来的数字就是要点的次数。  只有当点击的次数大于等于这个次数的时候，那个获取flag的按钮才会被设置为可以按下，这时候才会获得flag。  这里面有很多地方可以修改，在初始化的时候，可以直接将获取flag的按钮改成true的。  这样就可以直接按下获取flag了。    一样的，使用apktool转出文件之后进入对应的 MainActivity.smali 进行修改就行了。    因为语法的关系，要想直接找这个false会比较困难，要配合对应使用的函数名才可以找到。  因为这个false是在onCreate这个方法里面的，所以直接搜索 onCreate 先定位到方法  直接找到。    然后找参考之前我们能看懂的那个java代码，可以知道false是在调用 setClickable 函数时带的参数。所以一样的办法，在这个我们找到的办法里面查找哪里调用了这个函数就可以。    这里就看到了调用的函数就在这里，我们发现调用的函数里面有两个参数，一个 v0 、一个 v5 ，这里比较就能知道，肯定不是 v0 ，而 v5 的值则是 0 ，所以就是 v5 ，那么我们要改成 true ，也就是将 0 改成非零数即可了。我这里改成了 1 。    然后重新编译一下这整个文件就可以进行测试了。  另外，在编译的时候有一个关于文件的图片无法正常使用的问题（一共有两个图片都是这样的问题，这里就用一个图片了）    这个位置的照片出现了问题，不能找到这个照片，再打开对应的照片之后，发现这个照片命名就在这个文件夹里面。    在使用16进制的编辑器查看之后发现，这个图片其实是 jpg 文件，而不是 png 图片。这里需要对这个照片进行另存为 png 。在重新修改过后。就可以编译了。      对这个 apk 重新签名之后，再安装测试，现在就可以直接点击获取 flag 了。    除了修改这里，还有别的办法也能实现，因为比较的判定是在方法 Btn\_up\_onclick 里的。    可以看到，这里有一个 if 的判断。那么如果将这个 <= 修改成 >= 也就实现了目的。  一样的，进入 MainActivity.smali 文件后利用搜索，先找到对应的 Btn\_up\_onclick 方法，然后再找到这个比较判断在哪里。    在这里可以看到，程序直接将这两个赋值给了v3 和 v4 ，所以，一种改法就是直接把 v3 和 v4 改一下位置就可以。  这里就直接尝试一下这个办法。修改完成后如下：    安装完成后，如下：    我发现不能直接按爬到了，然后点击一次爬楼。  就发现 flag 按钮就能点了      成功了。  另外，我还发现了一个“偷鸡”的情况，这个 apk 有概率刷出需要爬出的楼层为0的情况。这个时候直接点击爬到了就可以了……直接获得flag 。这个情况出现的概率很低，比较看脸……如果运气好，第一次都不用逆向，直接就出flag了……（这个图真的不是我p的……是刷出来的）    运行 CaseA\_pwd.exe 并且获取其中的口令。  这是一个 windows 的可执行程序，所以也就用不着之前两个实验的虚拟机了……  先拖到 ida 里面看看这整个程序的工作的流程。    这里面可以知道整个程序的大致流程。大概就是要求输入一个字符串，而在程序内部有一个原始的字符串，之后程序会将这个字符串进行变化，然后将变化后的字符串和之前输入的字符串进行比较，如果一直就会显示是赢了，否则就会显示是错误的内容。  在利用 ida 的过程中，可以发现，在生成了变化后的字符串之后，程序会将生成的字符串放在 ecx 里面，那么这个其实是可以利用 ollyice 这些调试工具直接将程序在运行过程中，将寄存器里面的值显示出来，然后就知道了正确的答案了。（偷鸡行为）  （ eax 里面放的其实是我们输入的内容）  但是这里有个问题，就是虽然 ida 可以直接调出地址，并且我们也现在知道在哪里要设置断点再看寄存器就能知道答案，但是 ida 和 ollyice 运行的时候的内存地址是不一样的，是需要计算的。不过还好，程序虽然在不一样的地址，但是偏移量却是一样的，这样的话，那么就不难解决了。因为 ida 在反汇编这个程序的时候是从 0x401000 处开始的，而我们需要的设置断点的汇编指令是在 0x401072 处的。那么两者相减就是偏移量了，即 0x72 。那么接下来就是打开 ollyice ，进行调试这个软件了。这里可以看到程序就是从 0x0B11000 开始的。那么加上去就是要设置断点的地址，即 0x0B11072 。    移动到对应的位置，然后按下 F2 （设置断点）即可    接下来就是直接按下F9 （开始运行），然后需要输入内容，这里随便输入什么都可以。    回车键之后，需要按下F4（直接运行到目标位置），程序就会运行到断点处，然后就可以看到在寄存器里面就有我们需要的答案了    可以看到，密码就是 18YDcUGZ[`Y]P  然后重新用ollyice 运行一次试试。这次试用这个密码就行了。这次不断点运行的箭头就可以了。    成功了~ |
| 实验总结 | （对本次实验涉及到的知识点的重难点归纳、实验心得、思考与建议）  实验难点：这次实验有一些难度。因为汇编基础比较差，所以看汇编代码会很吃力。再加上上课的时候离屏幕远没办法看到内容，并且本人基本上没有使用过 ollyice ，因此最后的实验在一些细节上并没有研究透彻。 在整个第三个实验的汇编的执行流程，我还是没有理解最后部分为什么要执行了比较之后还要继续比较下去，是否是因为单单比较eax 里和 ecx 里的值还不够，里面的数值一个32位的寄存器放不下？  总结：第二个实验的修改方式有很多种，老师讲了很多种，我只用了两种，另外还有想到的几种种分别是对在初始化的时候的 has\_gone\_int 进行修改（改成很大很大就行），还有将那个 模32 改成每一次 模1，然后所有的值都只会是 0 ，修改每次按下的值，因为每一次增加的都只是 1 ，如果将这个按下的值也修改成比较大的值，应该也能让它更快地按到。 |
| 评分 |  |