180124 逆向-XMAN结营赛（前方高能）

原创

奈沙夜影 于 2018-01-25 08:47:43 发布 650 收藏

分类专栏： Android CTF

本文链接：https://blog.csdn.net/whklhhhh/article/details/79157436


Android 同时被 2 个专栏收录

54 篇文章 0 订阅

订阅专栏


CTF

163 篇文章 4 订阅

订阅专栏

1625-5 王子昂 总结《2018年1月24日》【连续第481天总结】

A. 结营赛前方高能WP

B.

前方高能

XMAN第七组WriteUp

说实话好像确实有点难，不过逻辑链条很完整，是中规中矩的逆向题。就是内容有点太多了2333有兴趣的可以再自己做做，都是我觉得挺有意思的题型

没人做到真正的部分让我们挺失落的qvq

qvqNDK的C不支持内联x86汇编，动态解密部分可是我辛辛苦苦用汇编Patch上去再不断调试了一个上午才完成的。不看我的面子也要看欣蕾姐被我们拖累到半夜四点还在上传题目的面子上啊（欣蕾姐真是对不起orz

作为出题人有点上帝视角，感觉这些东西都是轻轻松松就能被做出来的样子。比赛去结题的时候发现每一个步骤都有很多种可能，要逐个验证排除才能做出来~

第一次出题，转换视角的经验还是挺有意思的-0-

下面奉上WP

PS：正如分享时所说，模拟器中Android4.0\5.0是可以正常安装被伪加密的APK的。而高版本的安卓则不行，需要先去除伪加密。

伪加密

打开发现被加密，在模拟器中正常安装所以猜测伪加密

使用脚本清除所有”504b0102”后第8个字节的第一比特位

加密脚本如下

```
f = open(r"matex.apk", "rb")
res = f.read()
f.close()
flag = b"\x50\x4b\x01\x02"
for i in range(len(res)):
    if(res[i]==0x50):
        print(res[i:i+8])
        for j in range(3):
            if(res[i+j]!=flag[j]):
                break
        else:
            if(not res[i+8]&1):
                res = res[:i+8] + bytes(((res[i+8]) + 1 ,)) + res[i+9:]
f = open(r"matex_jiami.apk", "wb")
f.write(res)
f.close()
```

解密脚本如下：

```
f = open(r"matex.apk", "rb")
res = f.read()
f.close()
flag = b"\x50\x4b\x01\x02"
for i in range(len(res)):
    if(res[i]==0x50):
        print(res[i:i+8])
        for j in range(3):
            if(res[i+j]!=flag[j]):
                break
        else:
            if(res[i+8]&1):
                res = res[:i+8] + bytes(((res[i+8]) - 1 ,)) + res[i+9:]
f = open(r"matex_jiemi.apk", "wb")
f.write(res)
f.close()
```

整数溢出并Patch

反编译后发现输入字符串ginput在送入so校验之前，需要先执行zuxin类
count作为点击次数被送入zuxin类的构造方法
查看Encode类，发现encode方法初始化了值，最后decode输出3
其中的AES方法都没有被用到，负责写这里的小伙伴美滋滋地想骗过别人，最后被xkey那题多余的AES方法给骗到了（扶额

即要求3*count == 96728810

直接相除发现除不尽，猜想为整数溢出
写脚本爆破溢出高位：

```
p = 0
while(1):
    x = p * 0x100000000 + 96728810
    if(x%3==0):
        print("find: ", x//3)
        break
    else:
        p += 1
```

得到count：1463898702

静态将其Patch

```
iget v7, v6, Lcom/xman/matex/MainActivity;->count:I

const v7, 0x5741524e

iput v7, v6, Lcom/xman/matex/MainActivity;->count:I
```

PS：重打包的时候会报错，将/res/layout-v26下的xml删除即可。

XXTEA加密

通过这里以后发现调用了so的stringFromJNI函数
跟着so过去看

从GetStringUTFChars取得输入的字符串后进入了func函数
在其中调用了a函数
注意到参数有0x9E3779B9h，百度可知该常数为TEA类算法的特征

具体比较算法后确认是XXTEA

```
    z -= ((y<<4) + c) ^ (y + sum) ^ ((y>>5) + d);20
    y -= ((z<<4) + a) ^ (z + sum) ^ ((z>>5) + b);
```

key为{1, 2, 3, 4}

解出的值送给了c函数，要求c返回1

注：
刚才又动态调试验证了一下，貌似编译以后出现了什么奇奇怪怪的错误……
我拿源码的加密脚本都无法再还原过程，第一轮的第一个数相同，但从第二个数开始到八轮结束都完全不同了。因此也解密不回来。。。
之后有空再详细研究下。。。

动态解密

c函数中有23个结构体，在init中为每个结构体中保存了一个值（0x11*下标）和一个指针（指向申请的内存区域的不同位置）

rand_order中随机改变了除了第一个以外的结构体的顺序
然后遍历结构体，执行指针指向的函数块
从第一个开始，首先比较input对应的值，正确的话用后一个结构体的值异或解密该结构体中的代码块，错误的话就直接结束

在IDA中用IDC脚本可以直接解密，依次异或并提取每个代码块的中的第五个比特即可

```
.data:0002D004 ; ---------------------------------------------------------------------------
.data:0002D004
.data:0002D004                 public src
.data:0002D004 src:                                    ; DATA XREF: LOAD:00000310↑o
.data:0002D004                                         ; .got:src_ptr↑o
.data:0002D004                 push    esi
.data:0002D005                 push    edx
.data:0002D006                 mov     al, [edi]
.data:0002D008                 cmp     al, 5Eh ; '^'
.data:0002D00A                 jnz     loc_2D2EB
.data:0002D010                 jmp     loc_2D2CD
.data:0002D010 ; ---------------------------------------------------------------------------
.data:0002D015                 db  47h ; G
.data:0002D016                 db  43h ; C
.data:0002D017                 db  9Bh
.data:0002D018                 db  16h
.data:0002D019                 db  2Dh ; -
.data:0002D01A                 db  8Eh
.data:0002D01B                 db  1Eh
.data:0002D01C                 db  94h
.data:0002D01D                 db 0DBh
.data:0002D01E                 db  13h
.data:0002D01F                 db  11h
.data:0002D020                 db  11h
.data:0002D021                 db 0F8h
.data:0002D022                 db 0B6h
.data:0002D023                 db  13h
.data:0002D024                 db  11h
.data:0002D025                 db  11h
.data:0002D026                 db  74h ; t
.data:0002D027                 db  70h ; p
.data:0002D028                 db 0A8h
.data:0002D029                 db  25h ; %
.data:0002D02A                 db  1Eh
.data:0002D02B                 db 0C3h
```


IDC脚本如下：

```
auto i,j;
for(i=0;i<23;i++){
    for(j=0;j<17;j++){
        PatchByte(0x2d004+i*17+j, Byte(0x2d004+i*17+j)^(i*0x11));
        }
    }
```

Patch后的src如下

```
ata:0002D000                                    ; sub_4530+47↑o ...
ata:0002D004 ; ---------------------------------------------------------------------------
ata:0002D004
ata:0002D004                 public src
ata:0002D004 src:                               ; DATA XREF: LOAD:00000310↑o
ata:0002D004                                    ; .got:src_ptr↑o
ata:0002D004                 push    esi
ata:0002D005                 push    edx
ata:0002D006                 mov     al, [edi]
ata:0002D008                 cmp     al, 5Eh ; '^'
ata:0002D00A                 jnz     loc_2D2EB
ata:0002D010                 jmp     loc_2D2CD
ata:0002D015 ; ---------------------------------------------------------------------------
ata:0002D015                 push    esi
ata:0002D016                 push    edx
ata:0002D017                 mov     al, [edi]
ata:0002D019                 cmp     al, 9Fh
ata:0002D01B                 jnz     loc_2D2EB
ata:0002D021                 jmp     loc_2D2CD
ata:0002D026 ; ---------------------------------------------------------------------------
ata:0002D026                 push    esi
ata:0002D027                 push    edx
ata:0002D028                 mov     al, [edi]
ata:0002D02A                 cmp     al, 0E1h
ata:0002D02C                 jnz     loc_2D2EB
ata:0002D032                 jmp     loc_2D2CD
ata:0002D037 ; ---------------------------------------------------------------------------
ata:0002D037                 push    esi
ata:0002D038                 push    edx
ata:0002D039                 mov     al, [edi]
ata:0002D03B                 cmp     al, 89h
ata:0002D03D                 jnz     loc_2D2EB
ata:0002D043                 jmp     loc_2D2CD
ata:0002D048 ; ---------------------------------------------------------------------------
ata:0002D048                 push    esi
ata:0002D049                 push    edx
ata:0002D04A                 mov     al, [edi]
ata:0002D04C                 cmp     al, 0FAh
ata:0002D04E                 jnz     loc_2D2EB
ata:0002D054                 jmp     loc_2D2CD
```


很明显是在逐个cmp，脚本依次提取即可

```
auto i;for(i=0;i<23;i++){Message("%d, ", Byte(0x2d004 + 5 + i*17));}
```

得到

> 94, 159, 225, 137, 250, 93, 3, 196, 81, 7, 187, 148, 245, 145, 87, 217, 225, 74, 130, 214, 250, 176, 155

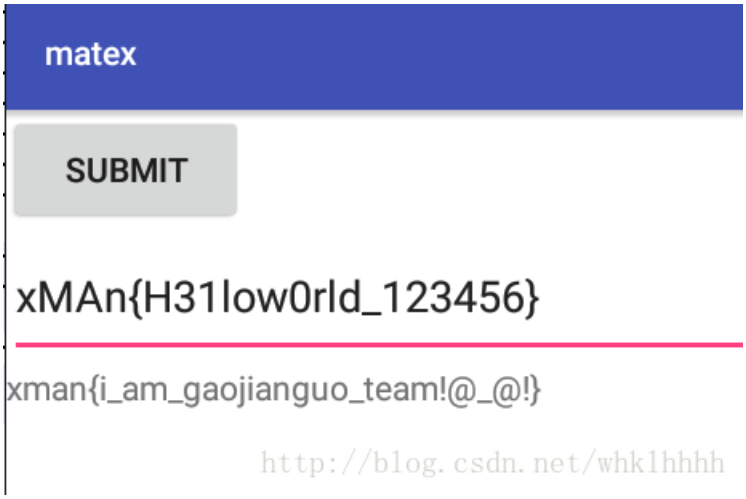
将这个值用XXTEA解密后即可得到输入

```
xMAn{H31low0rld_123456}
```

动态加载Dex

最后是dalao的动态加载Dex，但其实输入正确以后即可得到flag

如果在之前没有找到count的正确值将会在这里夭折
提取并解密Dex后发现它是个AES类
count作为IV向量，之前的输入作为key，解密一个字符串输出最终flag



附上作者提供的该模块解法：

1. 静态代码分析DEX，在MainActivity类中，发现LoadDex函数
   分析代码：
   public void loadDex()
   {
   this.d = new File(getDir(“DEX”, 0), “Abcd.dex”);
   this.o = getDir(“payload_assets”, 0);
   try
   {
   InputStream localInputStream = getAssets().open(“Abcd.jpg”);
   FileOutputStream localFileOutputStream = new java/io/FileOutputStream;
   localFileOutputStream.(this.d);
   int i = 0;
   for (;;)
   {
   int j = localInputStream.read();
   if (j == -1) {
   break;
   }
   if (i % 2 != 0)
   {
   i++;
   }
   else
   {
   localFileOutputStream.write(i / 2 ^ j);
   i++;
   }
   }
   localInputStream.close();
   localFileOutputStream.close();
   return;
   }
   catch (Exception localException)
   {

for (;;)
{
localException.printStackTrace();
}
}
}
在assets目录中有Abcd.jpg文件，被解密加载

2. 拿到Abcd.jpg，根据上面的算法，可以重构（解密)为Abcd.dex

   分析dex中的Abcd类的getAB函数，可以发现字符串转换算法
   public String getAB(String str) {

```
int[] ab= {3,2,7,5,4,2,1,5,3,6};

int a[]=new int[100];
char ch[]=new char[100];
ch=str.toCharArray();
for(int i=0;i<str.length();i++)
{


a[i]=(int)ch[i];

a[i]=a[i]^ab[i%ab.length];

ch[i]=(char)a[i];

}

String str1=new String(ch);
return str1;
```

   }

根据此算法，字符串中的每个字符进行异或处理

C. 明日计划
emmmmmm继续干内核驱动调试？