

社区 > Android安全

发新帖



05

[原创]X加密-反调试-DumpDex-修复指令-重打包 精

贝a塔 1 3

编辑



大牛



申请推荐此帖

举报

13

2019-2-27 03:45

20387



1.简介

如果有描述不准确的地方，还请各位大佬指正

某加密的最近的版本，网上资料比较少，是比较早的，没有混淆的，以及没有指令抽取的

这里分析的实例，是某加密直接把壳代码写到了原始dex里面去

1) 反调试 网上的大部分资料都是些基础的反调试，而且SO是没有经过ollvm混淆的，所以都是些比较直观的能看到代码
而现在大部分是ollvm混淆过，另外加UPX压缩壳，以及SO自解密等

- 2) dump dex，另一种找到dex.035的地址
- 3) 修复被抽取指令，找到codeitem的insns指针，取出指令，填充到被nop的dex里面去
- 4) 重打包APK 应该是做了对抗dex2smali，无法转回去，不过有办法

2. 调试

先看看Application

```

1 // 各家的壳都类似，attach和onCreate做初始化工作
2
3 protected void attachBaseContext(Context arg6) {
4     // some code
5
6     调用2个native方法初始化壳，以及hook相关还原函数
7     N.l((Application)this), "com.dmy.jiagushell");
8     N.r((Application)this), "android.app.Application");
9 }
10
11 public void onCreate() {
12     // some code
13     调用真正apk的Application
14     N.ra((Application)this), "android.app.Application");
15     if(S.n != null) {
16         S.n.onCreate();
17     }
18 }
```

这个样本解压开虽然直接能看到dex，但是还是通过调试dump了一次，这样可以了解一下反调试

SO被ollvm混淆了，看起来很费力，所以都是从一些关键点下断点去调试和绕过

1-反调试

关于反调试，有文章介绍了17中反调试，可以参考一下，现有资料博客能搜到的反调试，就这几种

时间检测，检测status，检测ida端口，以及java层的反调试

但是现在的壳已经不止这几种了，多而杂，现在常见的信号反调试，以及断点检测就比较麻烦

可以参考 [IDA技巧](#) 这位大神的方法，定位到initarray段的入口，和进入jni_onload的地址，

1.1 信号反调试，常识性的反调试，可以在bad_signal下断

信号函数通常是 signal(SIG_NO, antidebug_func)



TOP



05



跟进去看看 0xB388D391所在的函数内容，这个函数要在解密后才可以看到，就是获取pid，杀死

13

33

```

13  debug090:B388D390 ; 
  debug090:B388D390 PUSH {R7,LR}
  debug090:B388D392 BL sub_B38DF2E0 ; getpid
  debug090:B388D396 MOVS R1, #2
  debug090:B388D398 BL j_kill_0 ; kill自身进程
  debug090:B388D39C MOVS R0, #1
  debug090:B388D39E BL sub_B38DF570
  debug090:B388D3A2 POP {R7,PC}
  debug090:B388D3A2 ;

```

信号1, 2, 3的定义是

```

#ifndef _KERNEL_NODY_ 24
typedef unsigned long sigset_t;
#define SIGHUP 1
#define SIGINT 2
#define SIGQUIT 3
#define SIGTSTL 4

```

SIGHUP 终止进程 终端线路挂断

SIGINT 终止进程 中断进程

SIGQUIT 建立CORE文件终止进程，并且生成core文件

绕过方法就是，signal注册信号函数的时候，吧函数地址改为0，让他注册失败

1.2

进入到initarray段的时候，会有很多个函数，我的做法是通常跳过，如果某一次跳过崩了，就进去找反调试

```

linker:B6F45462 DCD 0x9E
linker:B6F45463 DCB 0xFB
linker:B6F45464 ;
linker:B6F45464 BLX R4
linker:B6F45466 LDR R0, =(_dl_g_ld_debug_verbose - 0xB6F4546C)
linker:B6F45468 ADD R0, PC ; _dl_g_ld_debug_verbose
linker:B6F4546A LDR R2, [R0]
linker:B6F4546C CMP R2, #1

```

入initarray段内的函数

fget断点后看到status检测，修改为0即可，后面还会遇到检测，应该是检测了多次，可以用hook过掉，或者IDA直接手动改掉端口号

```

debug090:B3872E98 LDR R0, =0x5B0 ; debug090:B3872F44+j ...
debug090:B3872E98 LDR R0, R0, R6
debug090:B3872E9A ADDS R0, #0x80
debug090:B3872E9C MOVS R1, #0x80
debug090:B3872E9E LDR R4, [R6,#0x38]
debug090:B3872EA0
debug090:B3872EA0 loc_B3872EA0 ; CODE XREF: debug090:B3872D7A+j
debug090:B3872EA0
debug090:B3872EA0 MOV R2, R4 ; debug090:loc_B3872E78+j ...
debug090:B3872EA0 BL unk_B38DF0E0 ; 跳转到fgets, 读取status
debug090:B3872EA6 CMP R0, #0
debug090:B3872EA8 BNE loc_B3872EAC
debug090:B3872EAA B loc_B3873036
debug090:B3872EAC
debug090:B3872EAC loc_B3872EAC ; CODE XREF: debug090:B3872E3A+j
debug090:B3872EAC MOV R0, SP
debug090:B3872EAE MOV R1, R0
debug090:B3872EB0 SUBS R1, #8
debug090:B3872EB2 MOV SP, R1
debug090:B3872EB4 MOVS R2, #7
debug090:B3872EB6 MVNS R4, R2
debug090:B3872EB8 MOVS R5, #0x3A
debug090:B3872EBA STRB R5, [R0,R4]
UNKNOWN B3872EA8: debug090:B3872EAB (Synchronized with PC)

```

，就是这里的BLX R4,是循环进

```

R6 BEE21CC8 [stack]:BEE21CC8
R7 BEE22308 [stack]:BEE22308
R8 B6F6262C linker:_dl_g_ld_debug_verbose
R9 00000033
R10 B5A7C344 [anon:linker_alloc]:B5A7C344
R11 B6F5B2FC linker:_dl_aeabi_llsl+1200
R12 BEE2245C
SP BEE22350 [stack]:BEE22350
LR B388D363 debug090:B388D363
PC B6CD837E libo.so:bad_signal
PSR 200D0030

```

```

2260 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....frorPi
2270 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....frorPi
2280 64 3A 09 31 38 39 36 39 0A 00 0A 00 6C 0A 00 00 d1:18969.
2290 64 00 00 00 AD C3 82 B3 3B 65 00 00 CO 22 E2 BE d....A..je.....
22A0 A4 22 E2 BE A2 B4 74 6C 72 6F 2E 62 75 69 6C 64 ."...tiro.build
22B0 2F 76 65 22 23 69 6F 2F 73 64 6B 00 02 24 1F .var...rsc-a3xx.so

```

```

然后转换为数字
debug090:B387302C ADDS R0, R0, R6
debug090:B387302E BL atoi
debug090:B3873032 STR R0, [R6,#0x48]
debug090:B3873034 LDR R4, [R6,#0x38]
debug090:B3873036

```

，这个比较简单直接改掉字符串

就行。



1.3 时间反调试，通常就是检测代码运行时间，暴力一点，R0保存的传入的参数,tm所以直接写0

```
struct timeval tm;
gettimeofday(&tm, NULL);
```

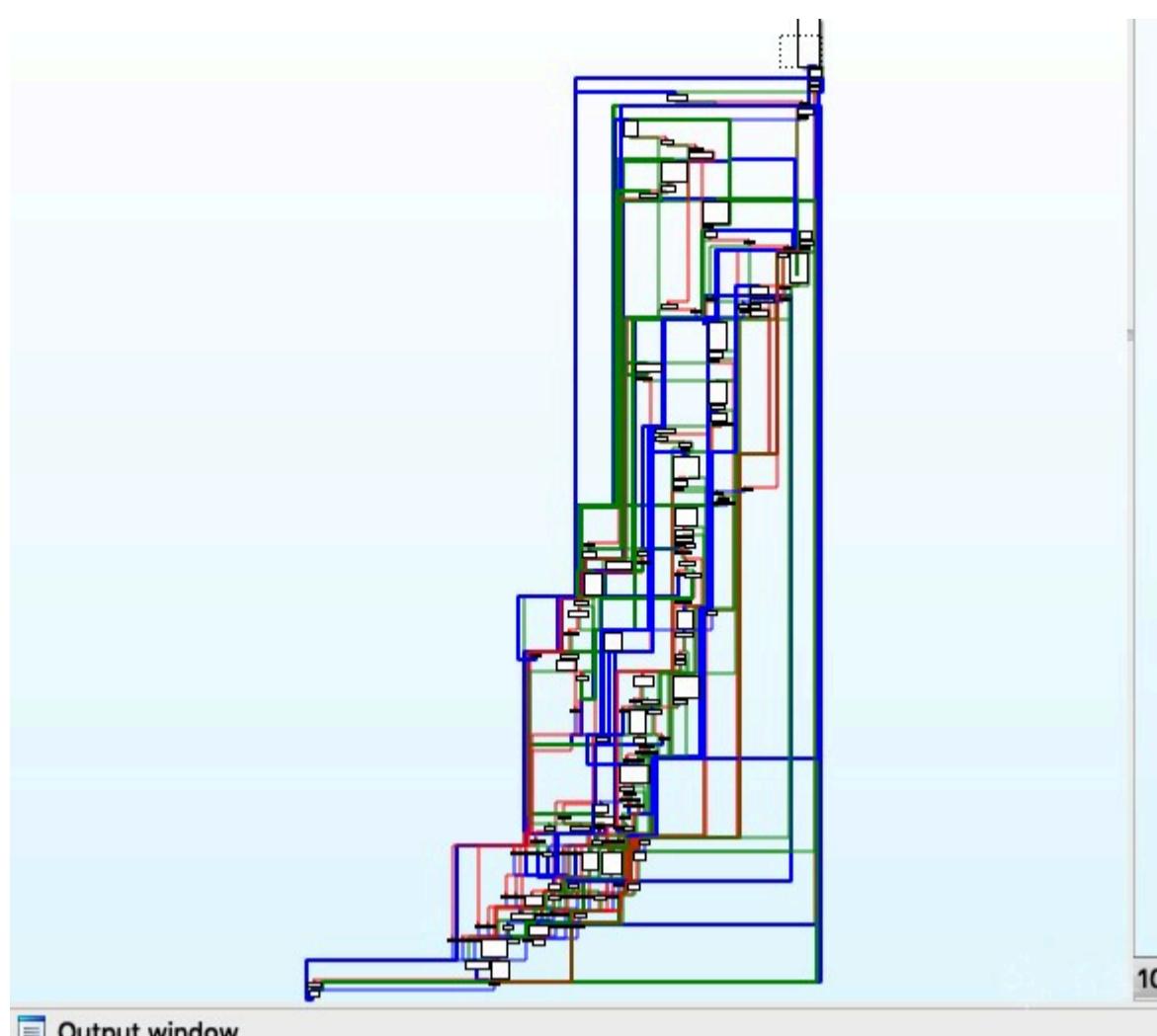
PC: libc.so:B6CD3328 07 C0 A0 E1 MOV R12, R7 ; 暴力干掉时间，直接R0写0

libc.so:B6CD332C
libc.so:B6CD332C retaddr
libc.so:B6CD332C 4E 70 A0 E3 MOV R7, #0x4E
libc.so:B6CD3330 00 00 00 EF SVC 0
libc.so:B6CD3334 0C 70 A0 E1 MOV R7, R12
libc.so:B6CD3338 01 0A 70 E3 CMN R0, #0x1000
libc.so:B6CD333C 1E FF 2F 91 BXLS LR
libc.so:B6CD3340 00 00 00 F0 BFB

时间反调试是status检测是在同一函数内，这个结构看起来比较费力，而且比较多的是无效的if语句，还是找找汇编的关键点来的快



TOP



```

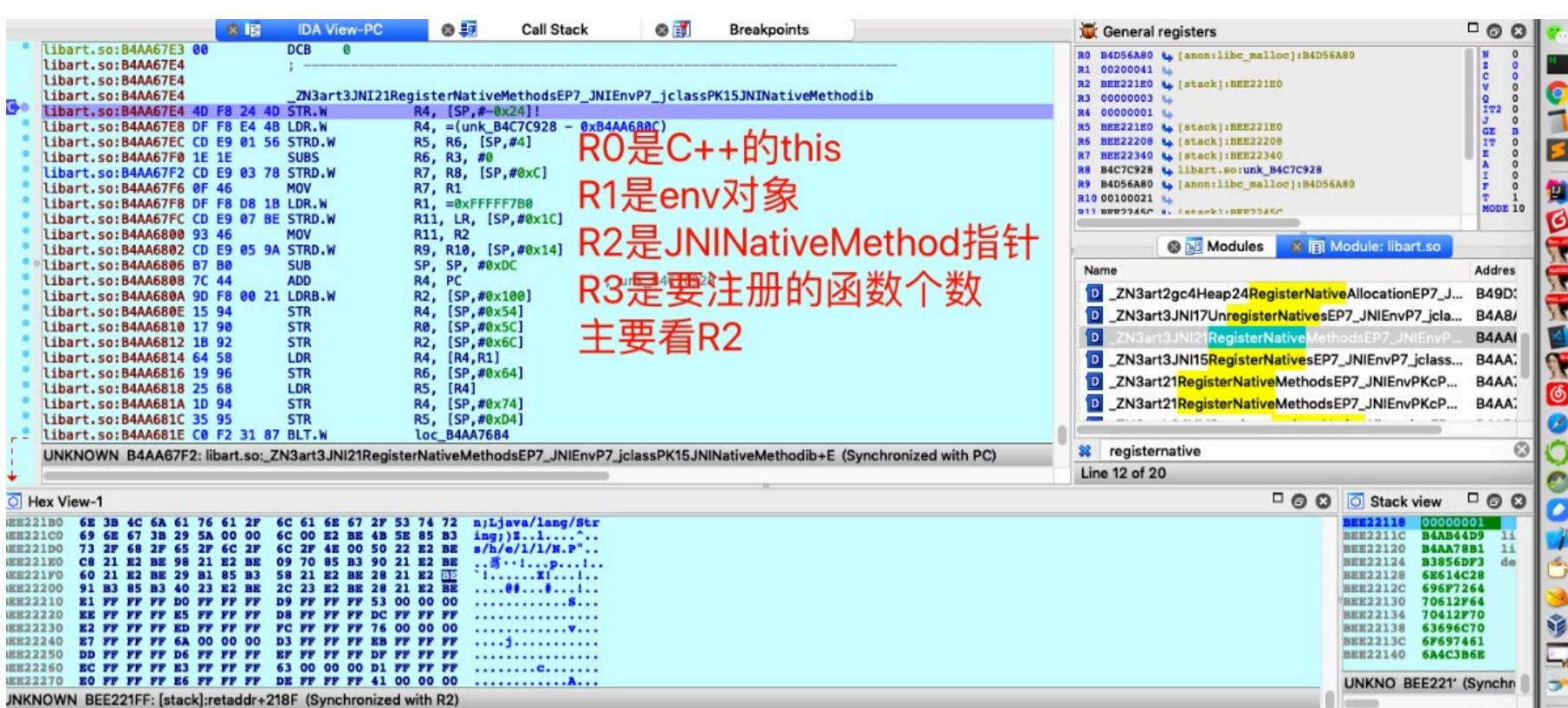
v4 = &v24 - 4;
v6 = &v24 == (int *)&word_10;
v11 = (signed int)(&v24 - 4) < 0;
if ( (unsigned int)&v24 > 0x10 )
    goto LABEL_93;
v23 = &v24 - 4;
if ( (unsigned int)&v24 <= 0x10 )
    goto LABEL_93;
v3 = (int)&v24;
if ( (unsigned int)&v24 <= 0x10 )
    goto LABEL_26;
if ( &v24 != (int *)&word_10 )
    break;
ABEL_73:
    if ( !v7 )
    {
ABEL_64:
    v42 = j_atoi((char *)&v24 + (_DWORD)&stru_46C +
    v3 = v38;
    goto LABEL_65;
}
    if ( v11 ^ v7 )
        goto LABEL_65;
}
if ( (unsigned int)&v24 > 0x10 )
    goto LABEL_35;
if ( (unsigned int)&v24 > 0x10 )
{
    if ( &v24 != (int *)&word_10 )
    {
        if ( (unsigned int)&v24 <= 0x10 )
            goto LABEL_51;
ABEL_63:
    if ( !j_strncmp(v8) )
        goto LABEL_64;
        goto LABEL_35;
}
    v7 = __OFSUB__(v4, v1);
    v5 = (unsigned int)v4 >= v1;
ABEL_93:
    v20 = v25;
    *((_BYTE *)&v24 - 14) = v25;
    *((_BYTE *)&v24 - 15) = v20;
    *((_BYTE *)&v24 - 11) = v33;
    *((_BYTE *)&v24 - 10) = v34;

```

1.4 在application的几个native函数
找前面提到的java函数注册为native函数的地址



TOP



看R2指针数据 是结构体，也就是0xBEE221E0

```
1 | typedef struct {<br>    const char* name;<br>    const char* signature;<br>    void*      fnPtr;<br>} JNINativeMethod;
```

BEE221E0	C8 21 E2 BE 98 21 E2 BE 09 70 85 B3 90 21 E2 BE ..
BEE221F0	60 21 E2 BE 29 B1 85 B3 58 21 E2 BE 28 21 E2 BE ..
BEE22200	91 B3 85 B3 40 23 E2 BE 2C 23 E2 BE 28 21 E2 BE ..
BEE22210	E1 FF FF FF D0 FF FF FF D9 FF FF FF 53 00 00 00 ..

对应结构体，BEE221C8地址处就是函数名称，bee22198就是方法签名，可以看下面2张图

BEE221C0	69 6E 67 3B 29 5A 00 00 1E 00 E2 BE 4B 5E 85 B3 ing;)Z..1....
BEE221D0	73 2F 68 2F 65 2F 6C 2F 6C 2F 4E 00 50 22 E2 BE s/h/e/1/1/N.."
BEE221E0	C8 21 E2 BE 98 21 E2 BE 09 70 85 B3 90 21 E2 BE ..

也就是在java层的

attachBaseContext中的第一个

BEE22190	72 00 E2 BE 2F 00 00 00 2E 4C 61 6E 64 72 6F 69 r.....(Landro
BEE221A0	64 2F 61 70 70 2F 41 70 70 6C 69 63 61 74 69 6F d/app/Applicatio
BEE221B0	6E 3B 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 n;Ljava/lang/Str
BEE221C0	69 6E 67 3B 29 5A 00 00 6C 00 E2 BE 4B 5E 85 B3 ing;)Z..1....

则be867090就是壳的第一个执行java层native函数

到第二个函数，这个点不像其他的 status是检测端口，时间是检测运行是否过长，以及信号注册函数

猜测可能是一个全局变量，存在这里，然后有我还没有发现的检测点，然后后面在判断这个全局变量（不知道这个解释是否合理）
如下图是运行到N.r这个里面进去

```
this.l(arg6);
N.l((Application)this, "com.dmy.jiagushell");
N.r((Application)this, "android.app.Application");
}
```

就是这里的N.r这是个native函数，又一次检测，通过计算在调试的时候上一步的地方，也就是0xB38ABC38 看出是在比较这个地址的值，不明白是什么检测方式，反正到这里下一句被崩，所以改了下寄存器先绕过吧



B38ABB0	F4 CF 8A B3 F0 CF 8A B3 A0 B7 8A B3 70 B7 8A B3
B38ABC00	88 B7 8A B3 64 B7 8A B3 30 B7 8A B3 28 B8 8A B3
B38ABC10	68 B8 8A B3 38 B7 8A B3 C4 56 8A B3 CC 56 8A B3 h...8...
B38ABC20	D4 56 8A B3 39 94 CA B6 70 81 8A B3 98 6E 8A B3
B38ABC30	85 C3 89 B3 00 00 00 00 00 00 00 00 00 00 00 00
B38ABC40	B1 9C CD B6 9D 9D CD B6 9B 94 C0 00 AD FC CA B6

TOP

这里的反调试可能在检测一个全局变量

反调试可能还有检测断点的，偶尔会崩掉，这里请教下各位大佬，怎么快速定位检测断点的反调试

2.1 关于dumpdex，其实可以dump的点很多，我选择defineclass位于classlinker里面
是以为他的第三个参数是java类对应的签名就是哪个 env->FindClass的那个参数
由此可以判断此时加载的这个类是位于北加固的dex里面的，顺着这个地方可以找到DEX

05



根据这个参数可以找到dex

13

R2存放的是java类对应的签名
地址是AF047E70

接下来，看源码可以知道，第5个参数是dex引用，多余的参数在栈里，计算好地址 + 4 进去就是dex引用地址，然后这里比较奇怪的是dex的这个地址要加上4就能找到dex.035字符串了，反正是加固前的DEX找到了，这里还要再琢磨下

源码里
第5个参数正好是dex的引用

SP的地址+4 = BEE22EC4
存放的是dex引用地址

这里看栈上的数据 aa73a0f0

AA73A0C0	76 00 69 00 65 00 77 00	2E 00 49
AA73A0D0	6E 00 64 00 6F 00 77 00	4D 00 61
AA73A0E0	67 00 65 00 72 00 00 00	78 00 66
AA73A0F0	E8 3E C7 B4 18 92 AA B3	98 3D 00
AA73A100	27 00 00 00 10 00	1B DA 59
AA73A110	18 92 AA B3 88 92 AA B3	D4 96 AA

b3aa9218,

按说这个就是dex的地址了，但是这里需要加上4个字节才是真实的地址 也就是去

TOP

B3AA9218是dex的地址，
比较明显

```

B3AA91D0 02 00 00 B0 3F 00 00 B4 3F 00 00 B8 3F 00 00 BC ....?...?...?
B3AA91E0 3F 00 00 C0 3F 00 00 C4 3F 00 00 C8 3F 00 00 CC ?.....
B3AA91F0 3F 00 00 D0 3F 00 00 D4 3F 00 00 D8 3F 00 00 DC ?.....
B3AA9200 3F 00 00 E0 3F 00 00 E4 3F 00 00 E8 3F 00 00 EC ?.....
B3AA9210 3F 00 00 14 40 00 00 00 65 78 0A 30 33 35 00 ?...@.dex.035.
B3AA9220 F9 72 B4 BD OC A3 40 F5 .....A 63 A4 9A .r...@...X.c.
B3AA9230 9E A3 3F OD 66 70 A3 53 98 3D 00 00 00 00 00 00 ...?fp.S.=..p.
B3AA9240 78 56 34 12 00 00 00 00 00 00 00 00 00 00 00 00 xV4....
B3AA9250 13 01 00 00 70 00 00 00 56 00 00 00 00 04 00 00 ....p..V.
B3AA9260 42 00 00 00 14 06 00 00 2A 00 00 00 2C 00 00 00 B.....*...
B3AA9270 A2 00 00 00 7C 0A 00 00 11 00 00 00 8C 0F .....|...
B3AA9280 EC 2B 00 00 AC 11 00 00 AC 11 00 00 AE 11 00 00 .....|...
B3AA9290 B1 11 00 00 BE 11 00 00 CF 11 00 00 E1 11 00 00 .....|...
B3AA92A0 E6 11 00 00 F0 11 00 00 F8 11 00 00 FD 11 00 00 .....|...
B3AA92B0 13 12 00 00 23 12 00 00 2B 12 00 00 37 12 00 00 .....$...7...
B3AA92C0 41 12 00 00 4F 12 00 00 58 12 00 00 62 12 00 00 A...O...X.B.
B3AA92D0 6B 12 00 00 79 12 00 00 7C 12 00 00 88 12 00 00 k...y...l...
B3AA92E0 9A 12 00 00 9D 12 00 00 B0 12 00 00 B4 12 00 00 .....|...
B3AA92F0 BC 0C 12 00 00 C3 12 00 00 CB 12 00 00 D0 12 00 00 .....|...
B3AA9300 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
B3AA9310 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
B3AA9320 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
B3AA9330 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
B3AA9340 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
B3AA9350 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
B3AA9360 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????

```

UNKNOWN B3AA9218: base.odex:oatdata+218 (Synchronized with R0)

★

05

👍

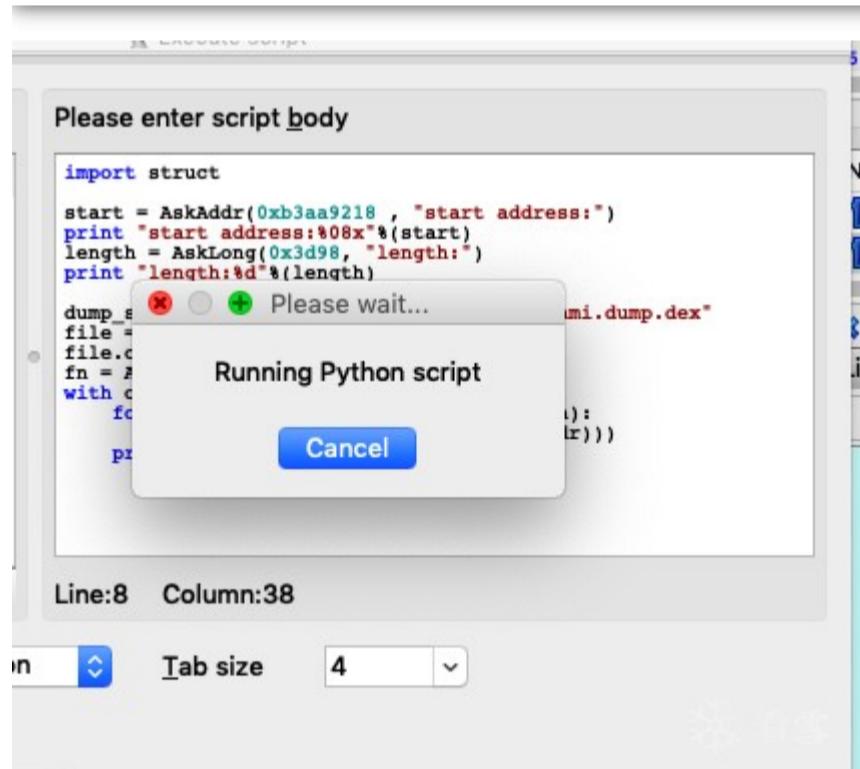
13

往后20个字节是DEX的长度，好了直接dump
上脚本

```

1 import struct
2
3 start = AskAddr(0xb3aa9218 , "start address:")
4 print "start address:%08x"%(start)
5 length = AskLong(0x3d98, "length:")
6 print "length:%d"%(length)
7
8 dump_so = "/Users/beita/Desktop//ijiami.dump.dex"
9 file = open(dump_so, 'w')
10 file.close()
11 fn = AskStr(dump_so , "save as:")
12 with open(fn,"wb+") as f:
13     for addr in range(start , start+length):
14         f.write(struct.pack("B" , Byte(addr)))
15     print "success to save as "

```



(绕过 部分反调试，因为这里的反调试我找到的不是所有的) 此时dump出来的是方法体是空的

```

package com.dmy.jiagushell;
import android.app.Activity;
import android.os.Bundle;

public class MainActivity
    extends Activity
{
    private void app() {}

    protected void onCreate(Bundle paramBundle) {}
}

```



TOP

★

05

👍

13

💬

我是改4.4的源码，因为小一点编译快一些

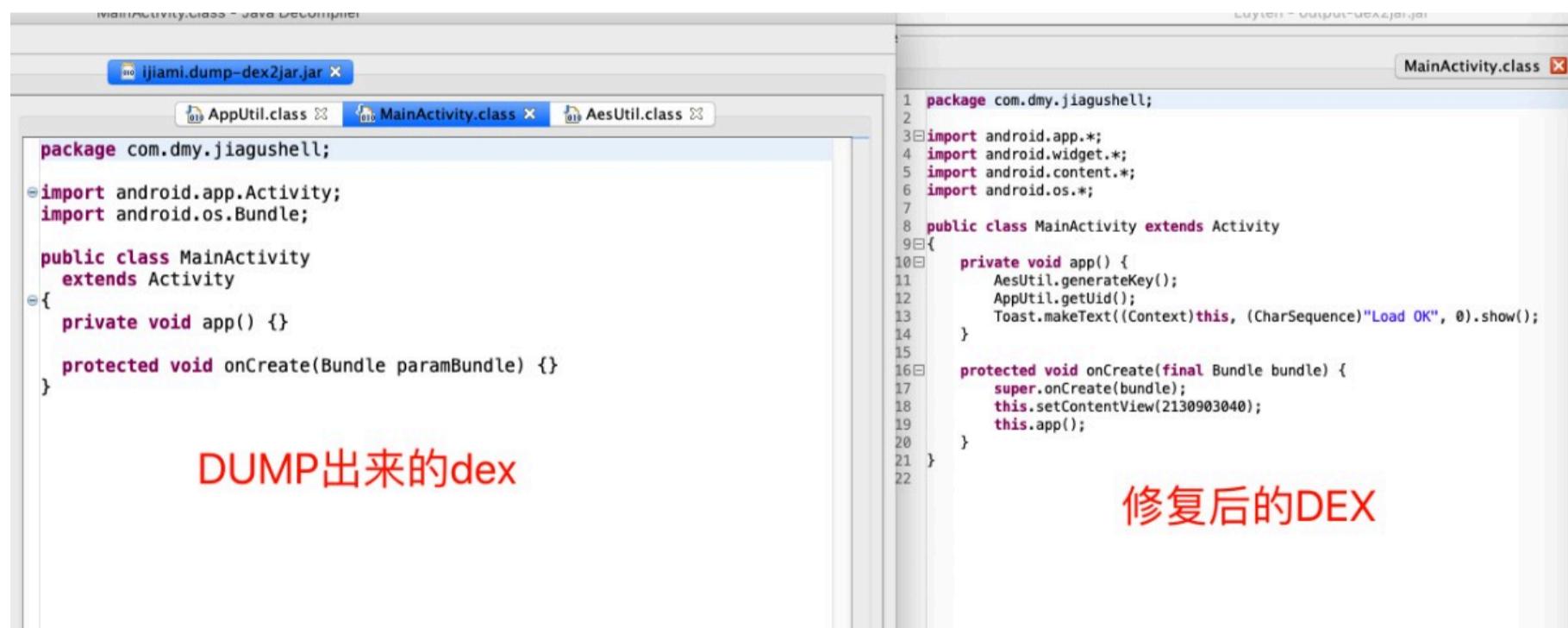
先看看被抽的dex，指令都是空的，

struct encoded_method_list direct_methods		methods
▼ struct encoded_method_list virtual_methods		1 methods
▼ struct encoded_method method		protected void com.dmy.jiagushell.MainActivity.onCreate(android.os.Bundle)
► struct uleb128 method_idx_diff		0x31
► struct uleb128 access_flags		(0x4) ACC_PROTECTED
► struct uleb128 code_off		0x2E44
▼ struct code_item code		2 registers, 2 in arguments, 2 out arguments, 0 tries, 12 instructions
ushort registers_size		2h
ushort ins_size		2h
ushort outs_size		2h
ushort tries_size		0h
uint debug_info_off		3FFFFFFEBh
uint insns_size		Ch
▼ ushort insns[12]		
ushort insns[0]		Eh
ushort insns[1]		0h
ushort insns[2]		0h
ushort insns[3]		0h
ushort insns[4]		0h
ushort insns[5]		0h
ushort insns[6]		0h
ushort insns[7]		0h
ushort insns[8]		0h
ushort insns[9]		0h
ushort insns[10]		0h
ushort insns[11]		0h

// 如下是代码片段

```

1 // 在dvmResolveClass中过滤下方法名称
2 DexClassData *classData = dexFindClassData(pDvmDex->pDexFile, clsObj->descriptor);
3 if(classData ==NULL) return;
4 // 里面的 dexFindClassData 直接调用就可以
5 for(u4 i = 0; i < classData->header.directMethodsSize; i++){
6
7     Method *method = (Method *) (clsObj->directMethods + i);
8     if(method == NULL || classData->directMethods ==NULL) continue;
9
10    DexMethod *dexMethod = classData->directMethods + i;
11    if(dexMethod == NULL || dexMethod->codeOff == 0) continue;
12    const DexCode *code = (const DexCode *) (pDvmDex->pDexFile->baseAddr + dexMethod->codeOff);
13    if(code == NULL) continue;
14
15    std::string insns_str;
16    for(u4 k=0; k < code->insnsSize; k++){
17        char tmp[32];
18        // method->insns就是要回复的指令
19        sprintf(tmp, "%04x ", method->insns[k]);
20        insns_str.append(tmp);
21    }
22    // 用字符串拼接起来
23 }
```



还原前后

TOP

首页

社区

课程

招聘

发现

★

05

ushort insns[12]	
ushort insns[0]	Eh
ushort insns[1]	0h
ushort insns[2]	0h
ushort insns[3]	0h
ushort insns[4]	0h
ushort insns[5]	0h
ushort insns[6]	0h
ushort insns[7]	0h
ushort insns[8]	0h
ushort insns[9]	0h
ushort insns[10]	0h
ushort insns[11]	0h
nt static values off	0h

!

13

uint insns_size Ch	
ushort insns[12]	206Fh
ushort insns[0]	1h
ushort insns[1]	10h
ushort insns[2]	115h
ushort insns[3]	7F03h
ushort insns[4]	206Eh
ushort insns[5]	32h
ushort insns[6]	10h
ushort insns[7]	1070h
ushort insns[8]	30h
ushort insns[9]	0h
ushort insns[10]	Eh
ushort insns[11]	0h
values off	

5.重打包

我发现修复后的dex不能直接转smali，因为里面有x加密加入的一些壳代码，用d2j-dex2smali.sh吧修复的转成smali，然后删除那部分可代码，发现dex2smali会出错

解决办法：直接把修复的dex放进去先APKTOOL打包一下，然后把打包好的APK在 apktool d demo.apk

这样就能转smali，方法比较简单，然后再打包就可以了

，经过测试运行是可以的，说明重打包方式有效，不过我觉得这里的原因可能是d2j-dex2smali的问题，apktool本身里面有转smali的工具，好在能解决这个问题

样本在后面的附件里

sodump出来的so进行分析比较简单，感谢ThomasKing的dump，快速下断点过反调试，我感觉我没有吧反调试都找出来，只是找出了dump DEX的那部分，先分析到这里

如下偏移处

0x717c 0xd380 0x10cac 0x10cac 都是结束的地方

有很多无效判断来干扰分析，比如这种代码，在汇编中会有很多看着蛋疼的地方

```
int x = 1;
```

```
if(x!= 1 && x==1) {
```

```
v5 = 0;
v2 = *(_DWORD *) (MEMORY[0x9F110030] + 4);
if ( (signed int)&off_BF9FC >= 0 && (signed int)&off_BF9FC < 0 )
{
```

还有花指令

参考：

<https://bbs.pediy.com/thread-194053.htm>

<https://bbs.pediy.com/thread-221876.htm>



TOP

5 是不是加固会检测如果一个DEX比如类比较少，或者体积非常小，就不进行加密DEX，(加固厂商判断这是个测试APK? ? ?)，加固后直接在原来的基础上加进去了ProxyApplication部分代码呢



05

[注意]看雪招聘，专注安全领域的专业人才平台！



13

上传的附件：

[样本.zip \(3.72MB, 326次下载\)](#)



最后于 2019-2-27 04:14 被贝a塔编辑，原因：

看雪论坛 - 安全技术与经验分享



收藏 · 105

· 13

支持

分享

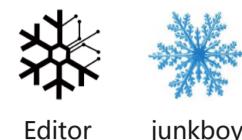
赞赏记录

参与人	雪币	留言	时间
PLEBFE		为你点赞~	2022-7-27 01:48
心游尘世外		为你点赞~	2022-7-26 23:43
飘零丶		为你点赞~	2022-7-17 03:23
EASONS		为你点赞~	2021-2-24 23:36
yezheyu		为你点赞~	2019-3-10 04:54
Exploring		为你点赞~	2019-3-8 21:40
贝a塔		为你点赞~	2019-3-3 21:00
glider菜鸟		为你点赞~	2019-3-2 21:19
aaaaaach		为你点赞~	2019-2-28 11:01
暴强		为你点赞~	2019-2-28 07:54
oecichial		为你点赞~	2019-2-27 10:58
bluth		为你点赞~	2019-2-27 10:53
Editor		为你点赞~	2019-2-27 10:21



打赏 + 10.00雪花

TOP



[Editor](#) [junkboy](#)



打赏次数 2

雪花 + 10.00

收起 ^

05



+5.00

2019/02/27

精品文章 ~

13



+5.00

2019/02/27

感谢分享 ~

34

最新回复 (34)

[1](#) [2](#) [▶](#)



[junkboy](#)

2 楼

支持

极客

2019-2-27 08:08

0 ...



[五天](#)

3 楼

辛苦，谢分享

极客

2019-2-27 08:55

0 ...



[openlight](#)

4 楼

感谢！

极客

2019-2-27 09:37

0 ...



[bluth](#)

5 楼

确实，包括基本上能搜到的脱壳的资料都很少提到信号反调试这块的

极客

2019-2-27 10:14

0 ...



[Editor](#)

6 楼

厉害！

版主

2019-2-27 10:21

0 ...



[黑洛](#)

7 楼

很强。随着手机性能过剩，以后移动端也不太好混了，代码变形会越来越厉害。

大侠

2019-2-27 12:05

0 ...

Q6329
55184

[邓dg](#)

8 楼



安卓编程有VC _asm{}功能吗 直接运行DEX里面的代码

临时

2019-2-27 12:55

0 ...



最新回复 (34)

1 2 ▶

9 楼



茄杉 7

mark 厉害

2019-2-27 17:46

0 ...

05



暴强 7

这个可是好东西，正好学习下！

极客

10 楼

2019-2-28 07:54

0 ...

13



bbsshop 7

mark

极客

11 楼

2019-2-28 10:07

0 ...



lishangdi 2

很棒感谢大佬分享

极客

12 楼

2019-2-28 11:17

0 ...



DegerYang 4

极客

13 楼

2019-2-28 12:16

0 ...



kofly 8 1

mark

大侠

14 楼

2019-2-28 12:52

0 ...



龙飞雪 10

支持

极客

15 楼

2019-2-28 18:02

0 ...



艺术大师 2

支持

极客

16 楼

2019-3-1 10:30

0 ...



DDIM 2

看分析是自定义rom，然后还原了类抽离加密，这是某密的企业版本么，我看他们的官网免费版就是这样子的。

极客

17 楼

2019-3-7 18:31

0 ...



TOP

首页

社区

课程

招聘

发现

最新回复 (34)

1 2 ▶

18 楼



没图你说个图 3 1

★

05

▲

13

找了一下 dvmResolveClass 这里修复指令的资料，说是发现被加固的 app 在这里还原代码，所以在这里添加代码就能修复；但是对于那些没在这里修复的，可能就要找其他位置；

我觉得，对于修复来说，怎么找这个还原的位置才是重点。

2019-3-8 14:16

▲ 1 ...

30

sunlulu 4 1

19 楼

mark

大侠

2019-3-27 19:57

▲ 0 ...

kinglinzi 6

20 楼

感谢大佬分享

极客

2019-6-15 22:18

▲ 0 ...

陈某人 9

21 楼

学习学习

极客

2019-6-17 14:22

▲ 0 ...

APP加固 临时

22 楼

需要加固的可以私信哟

2019-6-18 17:22

▲ 0 ...

THT-EX 2

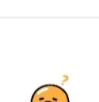
23 楼

学习学习

极客

2019-6-22 00:05

▲ 0 ...

奋斗der小鸟 7

24 楼

研究一下

极客

最后于 2019-8-19 10:05 被奋斗der小鸟编辑，原因：

2019-8-19 09:51

▲ 0 ...

cydian 9

25 楼

学习了

极客

2019-8-23 07:54

▲ 0 ...



贝a塔

3 年没回复的帖子已锁定，有问题请联系版主！



回帖

表情

高级回复

首页

社区

课程

招聘

发现

返回

©2000-2025 看雪 | Based on [Xiuno BBS](#)

域名: 加速乐 | SSL证书: 亚洲诚信 | 安全网易易盾

[看雪SRC](#) | [看雪APP](#) | 公众号: ikanxue | [关于我们](#) | [联系我们](#) | [企业服务](#)

Processed: 0.085s, SQL: 119 / 沪ICP备2022023406号 / 沪公网安备 31011502006611号

05



13



TOP