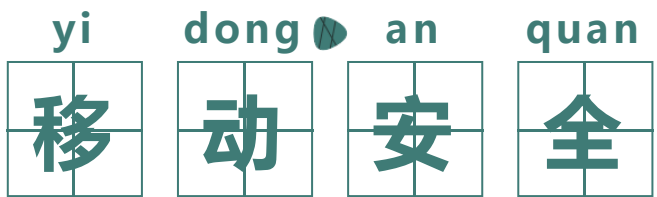


移动安全（十）|TengXun加固动态脱壳（下篇）

原创 辞令WHITECat WHITECat安全小组 前天

来自专辑

移动安全系列



0x00背景

本文是团队新加入的大佬**咸湿小和尚**在研究腾讯加固动态脱壳的一些总结经验，篇幅较长，希望各位大佬在细品之下有所收获～



0x01本文目录

▼ 腾讯加固壳之动态脱壳

▼ 1、分析 so 壳

▼ 1.1、使用 IDA 打开报错提示解决：

1.1.1 强行打开

1.1.2 使用 010editor 修改

1.1.3 使用 ELF 解析器，导入 bt 文件，然后启用

1.1.4 另存为并重新放入 ida

▼ 1.2、jni 函数被加密解决：

1.2.1 获取 so 文件中的 .init 或 .init_array

1.2.2 分析 .init_array

2、启动 IDA 动态调试

3、下断点

4、流程式查看

5、nop 掉反调试

6、找到关键的线程函数

7、查看分析关键函数

8、处理 dex 文件

0x02实验目的

通过动态分析调试对腾讯加固进行脱壳尝试

0x03实验开始

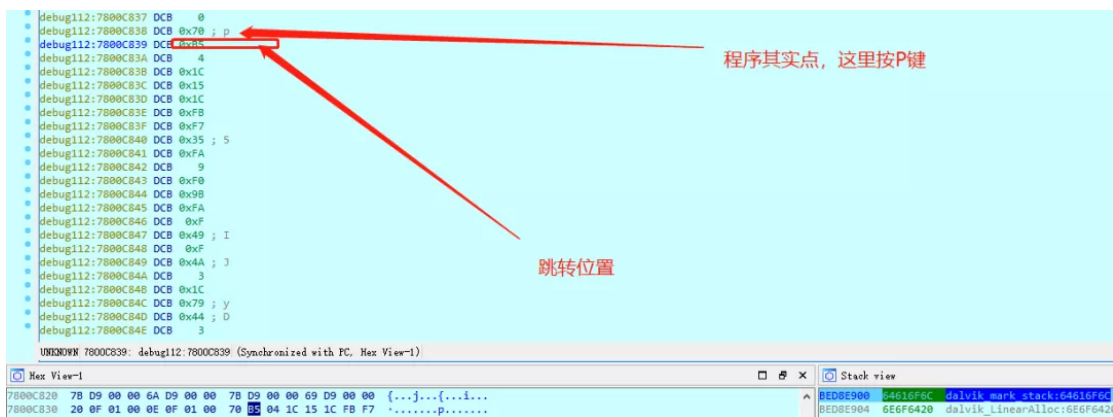
本篇为上一篇《移动安全（九）|TengXun加固动态脱壳（上篇）》的延续，如果上篇未读，请先点击上面标题移步上篇 >>>>

6、找到关键的线程函数

发现是关键(runCreate，而对应的参数是78017D96

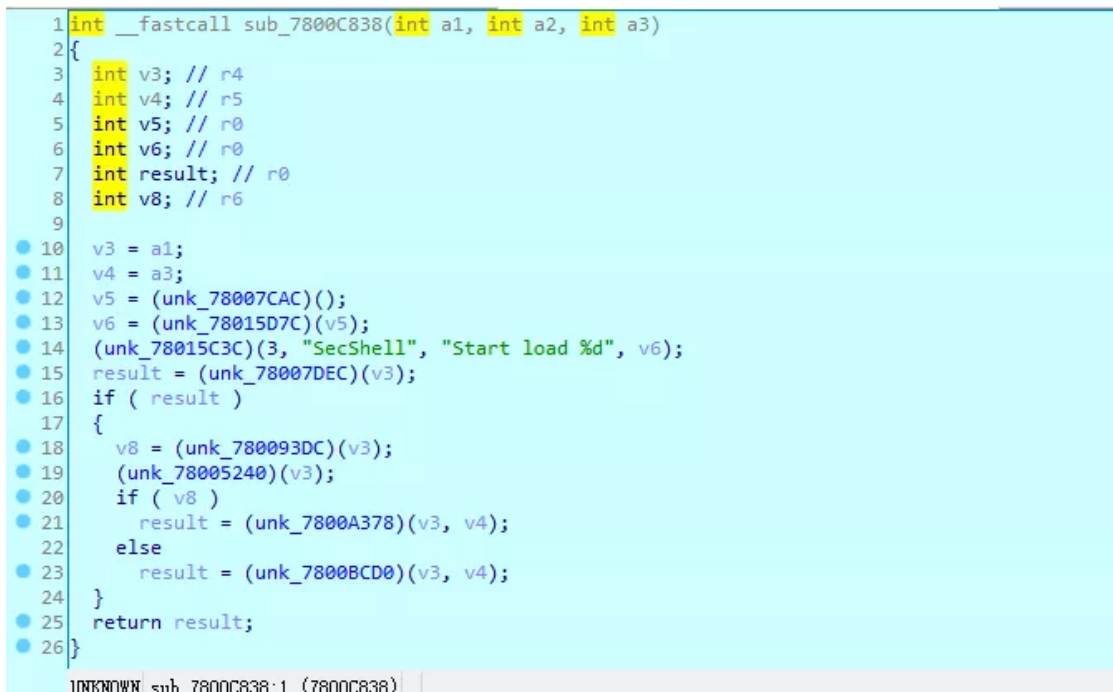
因此我们就要点击为上一个





进入到这里secshell的读取位置，这时候如果有经验的话，直接可以看到关键点在result，而result最后赋值hi在判断v8以后的

而相对较好就是每一个程序都跟踪一次



跟踪v8为判断java的vm类型，是否存在libart.so文件，这时候如果模块中没有libart.so文件就说明执行的是23行这个分支：

```

6| unsigned __int8 *v4; // r0
7| unsigned __int8 *v5; // r5
8|
9| v1 = a1;
10| v2 = 1;
11| if ( dword_7801D6E4 <= 19 )
12| {
13|     if ( dword_7801D6E4 == 19 && !(unk_78007D78)() )
14|     {
15|         (unk_78005CA4)(v1, "java.vm.version");
16|         v3 = (unk_780059C4)(v1, "java/lang/System", "getProperty", "(Ljava/lang/String;)Ljava/lang/String;");
17|         if ( v3 )
18|         {
19|             v4 = (unk_78005DB4)(v1, v3);
20|             v5 = v4;
21|             if ( v4 )
22|             {
23|                 if ( *v4 )
24|                 {
25|                     (unk_78015C3C)(3, "SecShell", "vm.version:%s", v4);
26|                     return ((*v5 - 48) & 0xFFu) > 1;
27|                 }
28|             }
29|         }
30|         if ( (unk_780093A4)(("/system/lib/libart.so") | (unk_780093A4)("/system/lib64/libart.so", 0) )
31|         {

```

因此，在23行跳进去函数后直接在其实头压栈的时候下好断点：

```

debug112:7800BCD0 var_A8= -0xA8
debug112:7800BCD0 var_A4= -0xA4
debug112:7800BCD0 var_90= -0x90
debug112:7800BCD0 var_BC= -0x8C
debug112:7800BCD0 var_78= -0x78
debug112:7800BCD0 var_1C= -0x1C
debug112:7800BCD0
PC debug112:7800BCD0 PUSH {R4-R7,LR}
debug112:7800BCD2 MOV R6, R0
debug112:7800BCD4 LDR R0, =(unk_7801CE70 - 0x7800BCDE)
debug112:7800BCD6 LDR R3, =0xFFFFFFFF34
debug112:7800BCD8 SUB SP, SP, #0x11C
debug112:7800BCDA ADD R0, PC ; unk_7801CE70
debug112:7800BCDC STR R0, [SP,#0x130+var_E8]
debug112:7800BCDE LDR R4, [R0,R3]
debug112:7800BCE0 LDR R2, =(aAndroidContent_2 - 0x7800BCEA)
debug112:7800BCE2 MOV R0, R6
debug112:7800BCE4 LDR R3, [R4]
debug112:7800BCE6 ADD R2, PC ; "android/content/Context"
debug112:7800BCE8 STR R3, [SP,#0x130+var_1C]
debug112:7800BCEA LDR R3, =(aLjavaLangClass_0 - 0x7800BCF0)
debug112:7800BCEC ADD R3, PC ; "()Ljava/lang/ClassLoader;"
debug112:7800BCEE STR R3, [SP,#0x130+var_130]
debug112:7800BCF0 LDR R3, =(aGetClassLoader - 0x7800BCF6)
debug112:7800BCF2 ADD R3, PC ; "getClassLoader"
debug112:7800BCF4 BL unk_78005BEC
UNKNOWN 7800BCF4: sub_7800BCD0+24 (Synchronized with PC, Hex View=1)

```

7、查看分析关键函数

然后查看程序：

```
debug112:7800BCD0 var_A8= -0xA8
debug112:7800BCD0 var_A4= -0xA4
debug112:7800BCD0 var_90= -0x90
debug112:7800BCD0 var_8C= -0x8C
debug112:7800BCD0 var_78= -0x78
debug112:7800BCD0 var_1C= -0x1C
debug112:7800BCD0
debug112:7800BCD0 PUSH {R4-R7,[R]}
debug112:7800BCD2 MOV5 R6, R0
debug112:7800BCD4 LDR R0, =(unk_7801CE70 - 0x7800BCDE)
debug112:7800BCD6 LDR R3, =0xFFFFFFFF34
debug112:7800BCD8 SUB SP, #0x11C
debug112:7800BCDA ADD R0, PC ; unk_7801CE70
debug112:7800BCDC STR R0, [SP,#0x130+var_E0]
debug112:7800BCDE LDR R4, [R0,R3]
debug112:7800BCE0 LDR R2, =(aAndroidContent_2 - 0x7800BCEA)
debug112:7800BCE2 MOV5 R0, R6
debug112:7800BCE4 LDR R3, [R4]
debug112:7800BCE6 ADD R2, PC ; "android/content/Context"
debug112:7800BCE8 STR R3, [SP,#0x130+var_1C]
debug112:7800BCEA LDR R3, =(aLjavaLangClass_0 - 0x7800BCF0)
debug112:7800BCEC ADD R3, PC ; "()Ljava/lang/ClassLoader;"
debug112:7800BCEE STR R3, [SP,#0x130+var_130]
debug112:7800BCF0 LDR R3, =(aGetClassLoader - 0x7800BCF6)
debug112:7800BCF2 ADD R3, PC ; "getClassLoader"
debug112:7800BCF4 BL unk_78005BEC
UNKNOWN 7800BCF4: sub_7800BCD0+24 (Synchronized with PC, Hex View-1)
```

得出：

```
233 v75 = 0;
234 }
235 else
236 {
237     v18 = (unk_780078A4)(v07) - 40;
238     v75 = 1;
239 }
240
241 v19 = (unk_780078CA)(v18 + 40);
242 (unk_78015C3C)(3, "SecShell", "orgDexOffset:%d", v19);
243 (unk_78015C9C)(0, 0, 112);
244 v09 = v18 + v19 + 40;
245 (unk_78015D1C)(0, v09, v18 + v19 + 40, 112);
246 (unk_78015C9C)(0, v09, 0, 16);
247 (unk_7800EE4C)(0, v09, 0, 112, 32);
248 v00 = v18 + v19 + 40;
249 v78 = v100;
250 v20 = (unk_78015C3C)(3, "SecShell", "fileSize:%d", v100);
251 if ( v75 )
252 {
253     v21 = v70;
254     if ( v70 & 0xFFF )
255         v21 = (v70 / 4096 + 1) << 12;
256     v20 = (unk_78015C7C)(v18, v20, 3);
257     if ( v20 )
258     {
259         v20 = v20 + 40;
260     }
261 }
```

文件偏移量计算

还有下文的filesize, 就说明是文件的大小

这个时候进行流程图查看：

```
238 v75 = 1;
239 }
240
241 v19 = (sub_780078CA)(v18 + 40);
242 (unk_78015C3C)(3, "SecShell", "orgDexOffset:%d", v19);
243 (unk_78015C9C)(0, 0, 112);
244 v09 = v18 + v19 + 40;
245 (unk_78015D1C)(0, v09, v18 + v19 + 40, 112);
246 (unk_78015C9C)(0, v09, 0, 16);
247 (unk_7800EE4C)(0, v09, 0, 112, 32);
248 v00 = v18 + v19 + 40;
249 v78 = v100;
250 v20 = (unk_78015C3C)(3, "SecShell", "fileSize:%d", v100);
251 if ( v75 )
252 {
253     v21 = v70;
254     if ( v70 & 0xFFF )
255         v21 = (v70 / 4096 + 1) << 12;
256     v20 = (unk_78015C7C)(v18, v20, 3);
257     if ( v20 )
258     {
259         v20 = v20 + 40;
260     }
261 }
```

忽然看到，这里不是0x28固定了吗，里面计算偏移量，为v19

那么问题是下面的文件大小有何用呢？

经过测试，经过查看，还有上面的两个classes.dex推断出，第一次解密出来的只是程序的执行后部分，而第二次解密的是dex的文件头需要两部分综合

Hex View-1

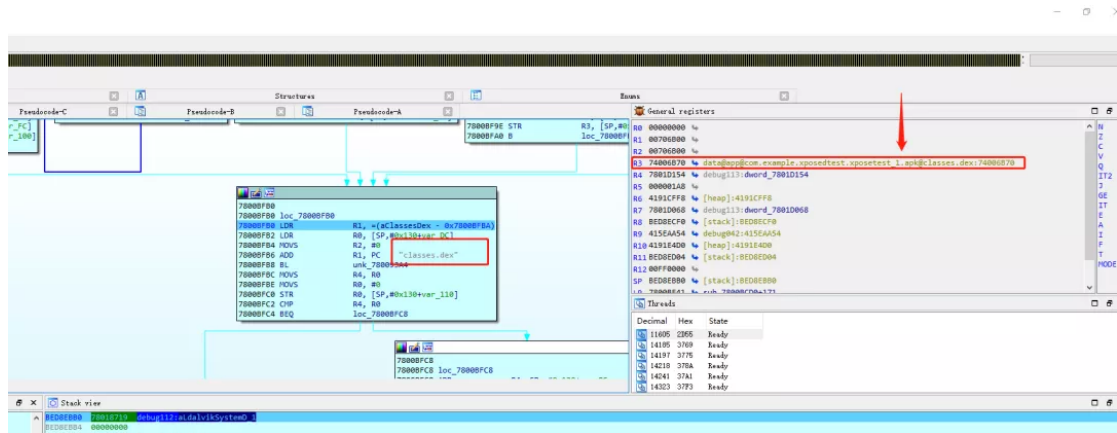
Stack view

0000C00 71 C0 00 00 4D E4 00 00 97 E4 00 00 34 04 00 00 q...N.....4...	BE0BCE0 00219068 dalvik_linearAlloc:00021906
0000C00 F0 05 06 1C D3 48 7A 00 C7 00 78 44 12 90 C4 58}x0...	BE0BCE4 4191E4C0 [heap]:4191E4C0
0000C00 D2 4A 30 1C 23 68 7A 44 45 93 D1 40 78 44 00 93 ...0..h0DE...{D...	BE0BCE8 00000000
0000CF0 D0 40 78 44 F9 F7 7A FF 17 94 07 90 00 28 01 D1D.....{D.....	BE0BCEC 414FE0D0 libdvm.so:dvmPlatformInvoke+74
0000D00 00 F0 5A F0 CC 49 30 1C CC 4D 79 44 F9 F7 82 F0D0...y0....	BE0BCE0 415EA054 debug042:415EA054
0000D10 CB 4A 04 1C 7D 44 2B 1C 21 1C 7A 44 30 1C F8 F7D+...z00...	BE0BCE4 00000001
0000D20 41 FD 21 1C 02 1C 30 1C F8 F7 A3 FD 0E 90 0E 99 ...l...0.....	BE0BCE8 7740D72E data@bapp@com.example.helloworld_1.apk:classes.dex:774
0000D30 30 1C FA F7 3F F8 C3 4A 2B 1C 0A 90 21 1C 7A 44 0.....3+...l..zD	BE0BCEC 42CEE070 dalvik_heap:42CEE070
0000D40 30 1C F8 F7 8F D0 21 1C 02 1C 30 1C F8 F7 91 FD 0.....0.....	BE0BED0 42CF5D5C dalvik_heap:42CF5D5C
0000D50 01 1C 30 1C FA F7 2E F8 8B 4B BC 4A 12 99 78 44 ..0.....K..J..D	BE0BED4 4152F127 libdvm.so:_Z16dvmCellJ1N1MethodPKJ1P6JValuePK6MethodP6T

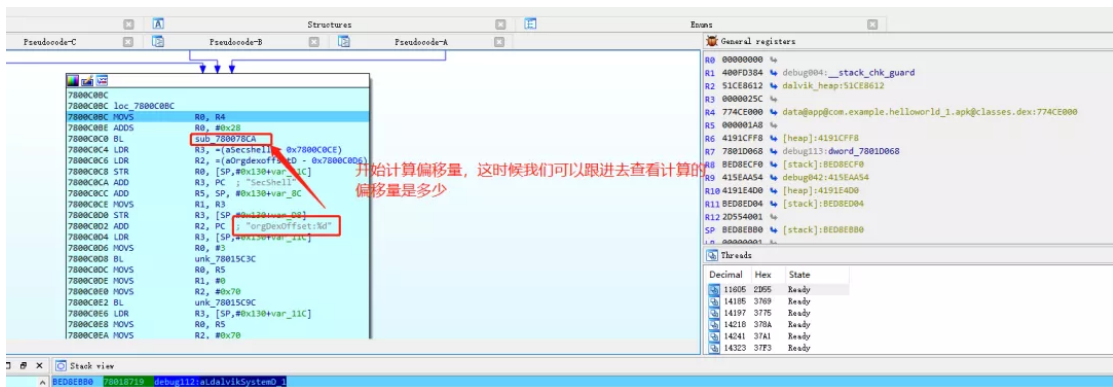
通过执行程序流程图，跟踪程序执行流程：

```
78008CE0 LDR      R2, =(aAndroidContent_2 - 0x78008CE0)
78008CE2 MOVS     R0, R6
78008CE4 LDR      R3, [R4]
78008CE6 ADD     R2, PC ; "android/content/Context"
78008CE8 STR      R3, [SP,#0x130+var_1C]
78008CEA LDR      R3, =(aJLjavaLangClass_0 - 0x78008CF0)
78008CEC ADD     R3, PC ; "()Ljava/lang/ClassLoader;"
78008CEE STR      R3, [SP,#0x130+var_130]
78008CF0 LDR      R3, =(aGetClassLoader - 0x78008CF6)
78008CF2 ADD     R3, PC ; "getClassLoader"
78008CF4 BL       unk_78005BEC
78008CF8 STR      R4, [SP,#0x130+var_D4]
78008CFA STR      R0, [SP,#0x130+var_114]
78008CFC CMP     R0, #0
78008CFE BNE     loc_78008D04
```

看到这里开始关注寄存器，堆栈，还有汇编的变化，记录每一步的变化：



关注点，跟踪解密特殊位置函数，已知数据大小为0x28即40：



跟踪计算偏移量, 进行解密

```

780078CA
780078CA
780078CA
780078CA sub_780078CA
780078CA LDR R3, [R0, #0x68]
780078CC LDR R2, [R0, #0x6C]
780078CE ADDS R0, R2, R3
780078D0 MOVS R3, #0x1000
780078D4 ADDS R0, R0, R3
780078D6 LSRs R0, R0, #0xC
780078D8 LSLS R0, R0, #0xC
780078DA BX LR
780078DA ; End of function sub_780078CA
780078DA

```

可知计算下来的执行下来的 $R3=0x38C14$, $R2=0x9710$, 然后 $R0=R3+R2$, 然后 $R3=0x1000$, 然后 $R0=R0+R3=(0x38C14+0x9710)+0x1000$, 然后逻辑右移 $0xC$, 再逻辑左移 $0xC$

因此写出python程序:

`hex((((0x38C14+0x9710)+0x1000)>>0xC)<<0xC)`

得到结果：

```
Python>hex((((0x38C14+0x9710)+0x1000)>>0xC)<<0xC)
0x43000
```

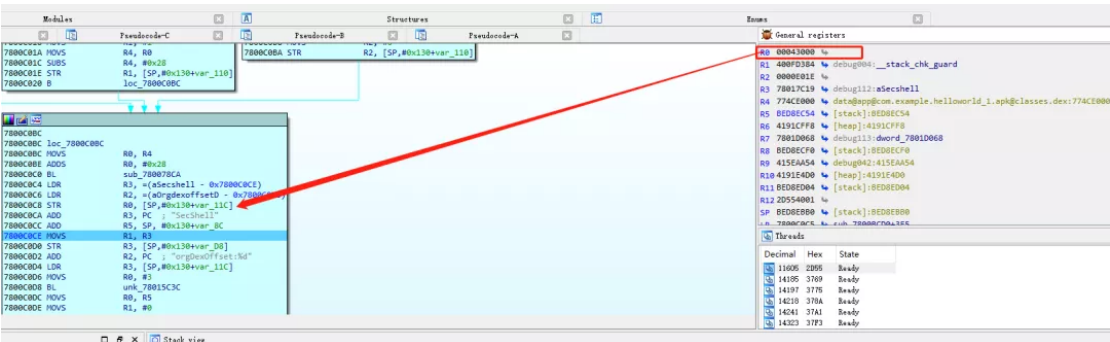
Python

AU: idle Down

为0x43000

而数据头文件大小为0x43000+0x28=0x43028

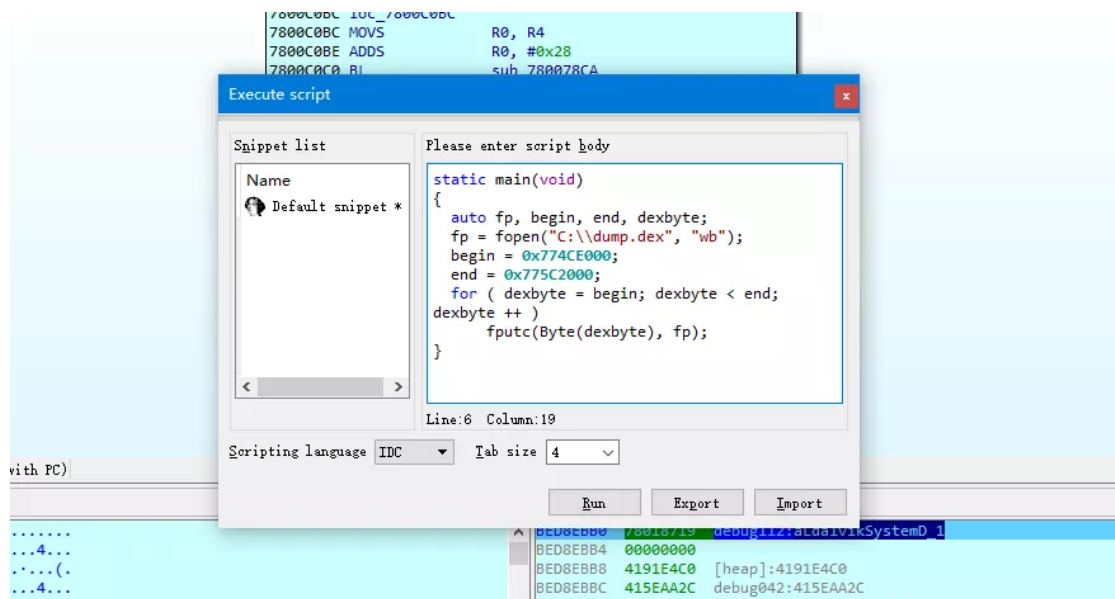
跟踪R0



这时候为了方便计算

[anon:libc_malloc]	774C1000	774CD000	R	W	.	D	.	byte	00	publi
com.example.helloworld_1.apk	774CD000	774CE000	R	.	.	D	.	byte	00	publi
data@app@com.example.helloworld_1.apk@classes.dex	774CE000	774EC000	R	.	.	D	.	byte	00	publi
data@app@com.example.helloworld_1.apk@classes.dex	774EC000	774ED000	R	.	.	D	.	byte	00	publi
data@app@com.example.helloworld_1.apk@classes.dex	774ED000	775C2000	R	.	.	D	.	byte	00	publi
com.example.helloworld_1.apk	775C2000	775C3000	R	.	.	D	.	byte	00	publi
com.example.helloworld_1.apk	775C3000	775C4000	R	.	.	D	.	byte	00	publi
dalvik_jit_code_cache	775C4000	7773B000	R	.	X	D	.	byte	00	publi
[anon:libc_malloc]	7773B000	77773000	R	W	.	D	.	byte	00	publi
debug084	77773000	77777000	R	W	.	D	.	byte	00	publi
debug085	77777000	77778000	?	?	?	D	.	byte	00	publi
[anon:libc_malloc]	77778000	77779000	R	W	.	D	.	byte	00	publi

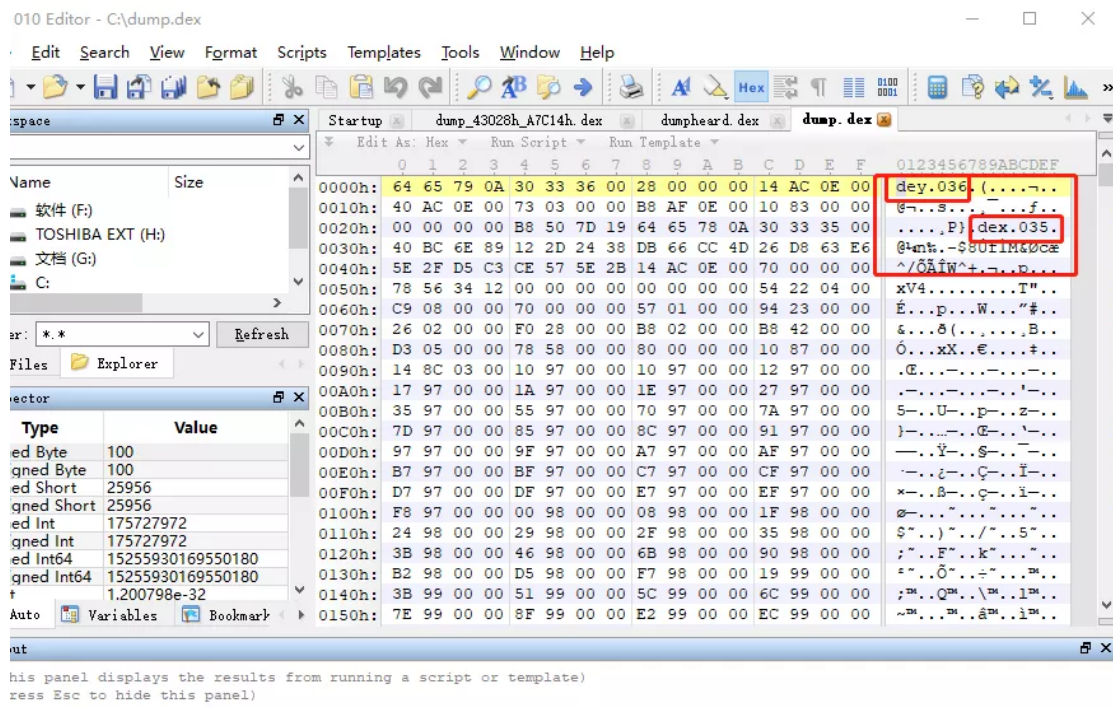
先dump下dex



```
staticmain(void){ autofp,begin,end,dexbyte; fp=fopen("C:\\dump.dex","wb"); begin=0x774CE000; end=0x775C2000; for(
dexbyte=begin;dexbyte<end;dexbyte++)    fputc(Byte(dexbyte),fp);}
```

8、处理dex文件

可以看到，混淆到我都不知道是odex还是dex文件了



然后继续看程序，执行解密heard部分数据：

```

1 signed int __fastcall sub_780EE4C(DWORD *a1, unsigned int *a2, unsigned int a3, int a4)
2 {
3     signed int v4; // r4
4     int v5; // r5
5     int v6; // r3
6     unsigned int v7; // r7
7     unsigned int v8; // r2
8     unsigned int v9; // r3
9     int i; // r4
10    int v12; // [sp+0h] [bp-38h]
11    unsigned int v13; // [sp+10h] [bp-28h]
12    int v14; // [sp+14h] [bp-24h]
13
14    if ( !a1 )
15        return 0;
16    if ( !a2 )
17        return 0;
18    v4 = 0;
19    v5 = a3 & 7;
20    if ( !(a3 & 7) )
21    {
22        v4 = 0;
23        if ( a4 )
24        {
25            v13 = a3 >> 3;
26            v12 = -1640531527 * a4;

```

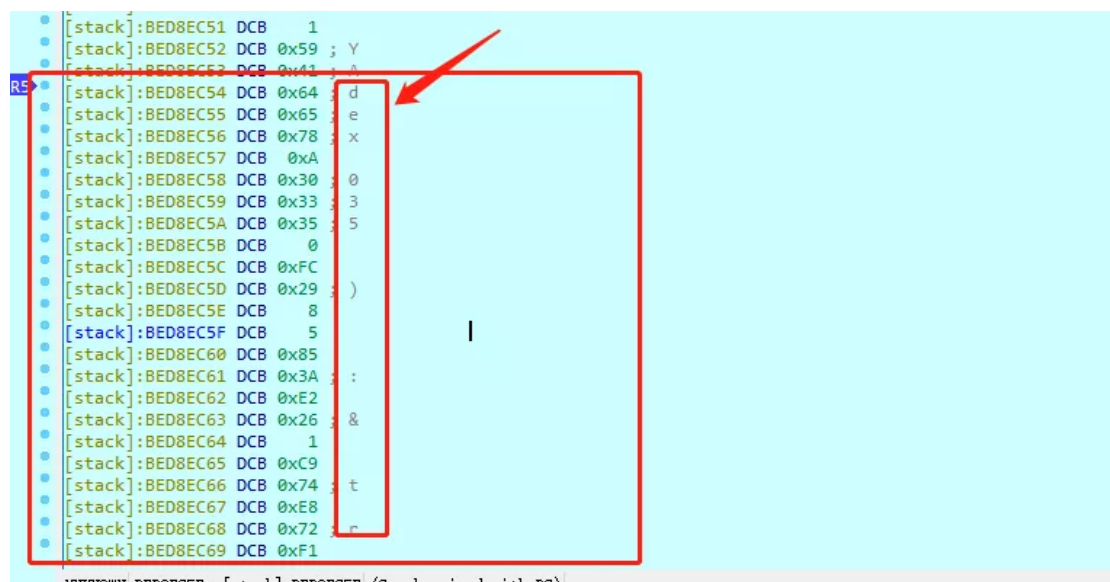
这里对R5进行处理：

```

7800C0E MOVS R2, #0x10
7800C100 STR R3, [SP,#0x130+var_11C]
7800C102 BL unk_78015C9C
7800C106 MOVS R1, R5
7800C108 MOVS R2, #0x70
7800C10A MOVS R3, #0x20
7800C10C ADD R0, SP, #0x130+var_CC
7800C10E BL sub_7800EE4C
7800C112 LDR R5, [R5,#0x20]
7800C114 LDR R0, [SP,#0x130+var_E0]
7800C116 LDR R2, =(aFileSizeD - 0x7800C120)
7800C118 LDR R1, [SP,#0x130+var_D8]
7800C11A STR R0, [SP,#0x130+var_F0]
7800C11C ADD R2, PC ; "fileSize:%d"
7800C11E MOVS R0, #3
7800C120 MOVS R3, R5
7800C122 STR R5, [SP,#0x130+var_11C]
7800C124 BL sub_78015C3C
7800C128 LDR R1, [SP,#0x130+var_110]
7800C12A CMP R1, #0
7800C12C BEQ loc_7800C16E

```

双击R5跟踪：



这里竟然出现了dex

说明这里就是起始点，那么还有就是结束点

结束点=起始点+文件大小，应该就在源程序中：

跟踪每一个可疑的数据：

```
7800C100 STR      R5, [SP,#0x130+var_11C]
7800C102 BL       unk_78015C9C
7800C106 MOVs     R1, R5
7800C108 MOVs     R2, #0x70
7800C10A MOVs     R3, #0x20
7800C10C ADD      R0, SP, #0x130+var_CC
7800C10E BL       sub_7800EE4C
7800C112 LDR      R5, [R5,#0x20]
```

看到R2是0x70就是文件大小，因为：程序执行四个参数分别为寄存器的r0,r1,r2,r3

```
1 signed int __fastcall sub_7800EE4C(_DWORD *a1, unsigned int *a2, unsigned int a3, int a4)
2 {
3     signed int v4; // r4
4     int v5; // r5
5     int v6; // r3
6     unsigned int v7; // r7
7     unsigned int v8; // r2
8     unsigned int v9; // r3
9     int i; // r4
10    int v12; // [sp+0h] [bp-38h]
11    unsigned int v13; // [sp+10h] [bp-28h]
12    int v14; // [sp+14h] [bp-24h]
13
14    if ( !a1 )
15        return 0;
16    if ( !a2 )
17        return 0;
18    v4 = 0;
19    v5 = a3 & 7;
20    if ( !(a3 & 7) )
21    {
22        v4 = 0;
23        if ( a4 )
24        {
25            v13 = a3 >> 3;
26            v12 = -1640531527 * a4;
```

而在输出的时候显示：

```

7800C0F8 MOV     R3, SP, #0x130+var_1C
7800C0FA MOV     R0, R3
7800C0FC MOV     R1, #0
7800C0FE MOV     R2, #0x10
7800C100 STR     R3, [SP, #0x130+var_11C]
7800C102 BL      unk_78015C9C
7800C106 MOV     R1, R5
7800C108 MOV     R2, #0x70
7800C10A MOV     R3, #0x20
7800C10C ADD     R0, SP, #0x130+var_CC
7800C10E BL      sub_7800EE4C
7800C112 LDR     R5, [R5, #0x20]
7800C114 LDR     R0, [SP, #0x130+var_E0]
7800C116 LDR     R2, =(aFileSizeD - 0x7800C120)
7800C118 LDR     R1, [SP, #0x130+var_D8]
7800C11A STR     R0, [SP, #0x130+var_F0]
7800C11C ADD     R2, PC, #0; "fileSize:%d"
7800C11E MOV     R0, #3
7800C120 MOV     R3, R5
7800C122 STR     R5, [SP, #0x130+var_11C]
7800C124 BL      sub_78015C3C
7800C128 LDR     R1, [SP, #0x130+var_110]
7800C12A CMP     R1, #0
7800C12C BEQ     loc_7800C16E

```

Stack view

而这个时候还有一个问题就是第一个程序的文件大小，这样只有偏移量没有大小

这时候又得回到源程序

```

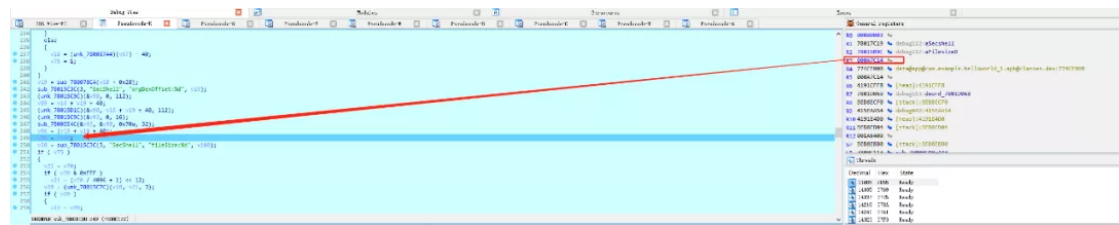
239     }
240     }
241     v19 = (sub_780078CA)(v18 + 0x28);
242     (unk_78015C3C)(3, "SecShell", "orgDexOffset:%d", v19);
243     (unk_78015C9C)(v98, 0, 112);
244     v89 = v18 + v19 + 40;
245     (unk_78015D1C)(v98, v18 + v19 + 40, 112);
246     (unk_78015C9C)(v93, 0, 16);
247     (unk_7800EE4C)(v93, v98, 112, 32);
248     v86 = v18 + v19 + 40;
249     v70 = v100;
250     v20 = (unk_78015C3C)(3, "SecShell", "fileSize:%d", v100);
251     if ( v75 )
252     {
253         v21 = v70;
254         if ( v70 & 0xFFFF )
255             v21 = (v70 / 4096 + 1) << 12;
256         v20 = (unk_78015C7C)(v18, v21, 3);
257         if ( v20 )
258         {
259             v22 = v70;
260             if ( v70 & 0xFFFF )
261                 v22 = (v70 / 4096 + 1) << 12;
262             v20 = (unk_78015C7C)(v18, v22, 5);
263         }

```

可以看到这里面引入了一个新的变量v70=v00,然后v70执行了一系列操作

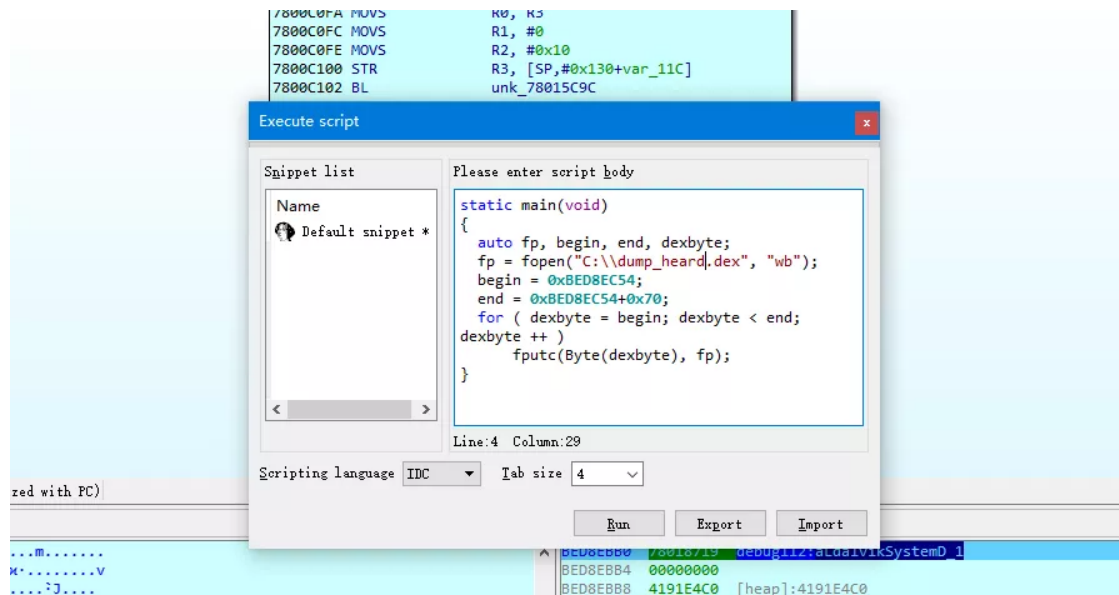
不管那么多，先执行，发现v100指向R3

R3为A7C14



若要分析v100是否为文件大小，向上逻辑虽然找不到，但是至少v70跟踪可以看出

先dump下dexheard文件



脚本：

```
staticmain(void)

{

autofp, begin, end, dexbyte;

fp= fopen("C:\\\\dump_heard.dex", "wb");

begin= 0xBED8EC54;

end= 0xBED8EC54+0x70;

for( dexbyte = begin; dexbyte < end; dexbyte ++ )

fputc(Byte(dexbyte),fp);

}
```

因此程序逻辑清晰了：

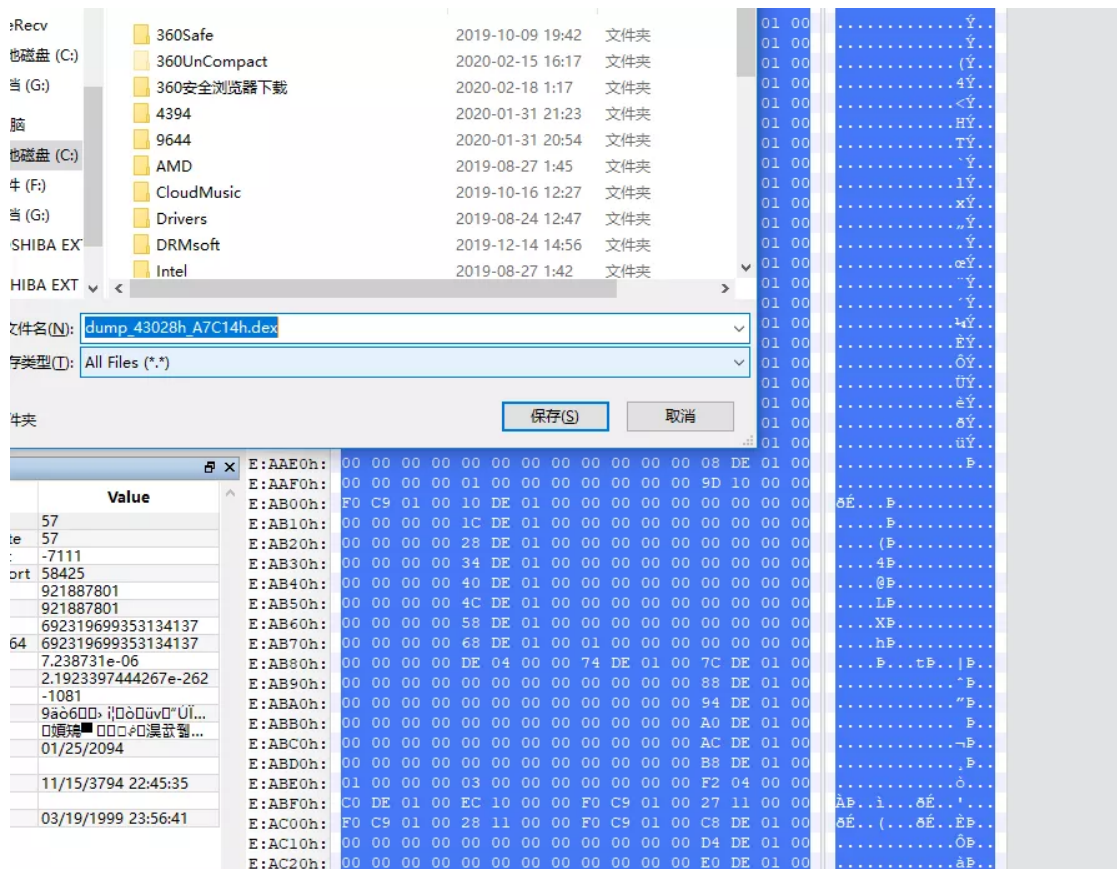
先获取到混淆的dex，然后根据偏移量，计算出起始点位置，然后再解密另一个dex文件头部和文件大小，然后拼接处再前面的dex的偏移量起始点处，这时候在赋上第一个文件大小A7C14

然后使用010editor：

edit->selectrange

E:A970h:	00 00 00 00 C1 0F 00 00 F0 C9 01 00 04 DD 01 00	A...øE...Y..	
E:A980h:	00 00 00 00 00 00 00 00 00 00 00 10 DD 01 00	Y..	
E:A990h:	00 00 00 00 00 00 00 00 00 00 00 1C DD 01 00	Y..	
E:A9A0h:	00 00 00 00 00 00 00 00 00 00 00 28 DD 01 00	(Y..	
E:A9B0h:	00 00 00 00 00 00 00 00 00 00 00 34 DD 01 00	4Y..	
E:A9C0h:	00 00 00 00 00 00 00 00 00 00 00 3C DD 01 00	<Y..	
E:A9D0h:	00 00 00 00 00 00 00 00 00 00 00 48 DD 01 00	HY..	
E:A9E0h:	00 00 00 00 00 00 00 00 00 00 00 54 DD 01 00	TY..	
E:A9F0h:	00 00 00 00 00 00 00 00 00 00 00 60 DD 01 00	Y..	
E:AA00h:	00 00 00 00 00 00 00 00 00 00 00 6C DD 01 00	1Y..	
E:AA10h:	00 00 00 00 00 00 00 00 00 00 00 78 DD 01 00	xY..	
E:AA20h:	00 00 00 00 00 00 00 00 00 00 00 84 DD 01 00	Y..	
E:AA30h:	00 00 00 00 00 00 00 00 00 00 00 90 DD 01 00	Y..	
E:AA40h:	00 00 00 00 00 00 00 00 00 00 00 9C DD 01 00	eY..	
E:AA50h:	00 00 00 00 00 00 00 00 00 00 00 A8 DD 01 00	Y..	
E:AA60h:	00 00 00 00 00 00 00 00 00 00 00 B4 DD 01 00	Y..	
E:AA70h:	00 00 00 00 00 00 00 00 00 00 00 BC DD 01 00	4Y..	
E:AA80h:	00 00 00 00 00 00 00 00 00 00 00 C8 DD 01 00	EY..	
E:AA90h:	00 00 00 00 00 00 00 00 00 00 00 D4 DD 01 00	ÖY..	
E:AAA0h:	00 00 00 00 00 00 00 00 00 00 00 DC DD 01 00	ÜY..	
E:AAB0h:	00 00 00 00 00 00 00 00 00 00 00 E8 DD 01 00	èY..	
E:AAC0h:	00 00 00 00 00 00 00 00 00 00 00 F0 DD 01 00	øY..	
E:AAD0h:	00 00 00 00 00 00 00 00 00 00 00 FC DD 01 00	üY..	
E:AAE0h:	00 00 00 00 00 00 00 00 00 00 00 08 DE 01 00	P..	
E:AAF0h:	00 00 00 00 01 00 00 00 00 00 00 0D 10 00 00	P..	
E:AB00h:	F0 C9 01 00 10 DE 01 00 00 00 00 00 00 00 00 00	øE...P.....	
E:AB10h:	00 00 00 00 1C DE 01 00 00 00 00 00 00 00 00 00	...P.....	
E:AB20h:	00 00 00 00 28 DE 01 00 00 00 00 00 00 00 00 00	...{P.....	
E:AB30h:	00 00 00 00 34 DE 01 00 00 00 00 00 00 00 00 00	...4P.....	
E:AB40h:	00 00 00 00 40 DE 01 00 00 00 00 00 00 00 00 00	...8P.....	
E:AB50h:	00 00 00 00 4C DE 01 00 00 00 00 00 00 00 00 00	...LP.....	
E:AB60h:	00 00 00 00 58 DE 01 00 00 00 00 00 00 00 00 00	...XP.....	
E:AB70h:	00 00 00 00 68 DE 01 00 01 00 00 00 00 00 00 00	...hP.....	
E:AB80h:	00 00 00 00 DE 04 00 00 74 DE 01 00 7C DE 01 00	...P...tP... P..	
E:AB90h:	00 00 00 00 00 00 00 00 00 00 00 88 DE 01 00	P..	
E:ABA0h:	00 00 00 00 00 00 00 00 00 00 00 94 DE 01 00	P..	
E:ABB0h:	00 00 00 00 00 00 00 00 00 00 00 A0 DE 01 00	P..	
E:ABC0h:	00 00 00 00 00 00 00 00 00 00 00 AC DE 01 00	P..	
E:ABD0h:	00 00 00 00 00 00 00 00 00 00 00 B8 DE 01 00	P..	
E:ABE0h:	01 00 00 00 03 00 00 00 00 00 00 F2 04 00 00	ö..	
E:ABF0h:	C0 DE 01 00 EC 10 00 00 F0 C9 01 00 27 11 00 00	Ap..i...øE...'	
E:AC00h:	F0 C9 01 00 28 11 00 00 F0 C9 01 00 C8 DE 01 00	øE...{...øE...EP..	
E:AC10h:	00 00 00 00 00 00 00 00 00 00 00 D4 DE 01 00	ÖP..	
E:AC20h:	00 00 00 00 00 00 00 00 00 00 00 E0 DE 01 00	äP..	
E:AC30h:	00 00 00 00 00 00 00 00 00 00 00 D0 00 00 00		
X	Select Start: 13028h	Size: A7C14h	Hex Options

然后File->saveselection as file

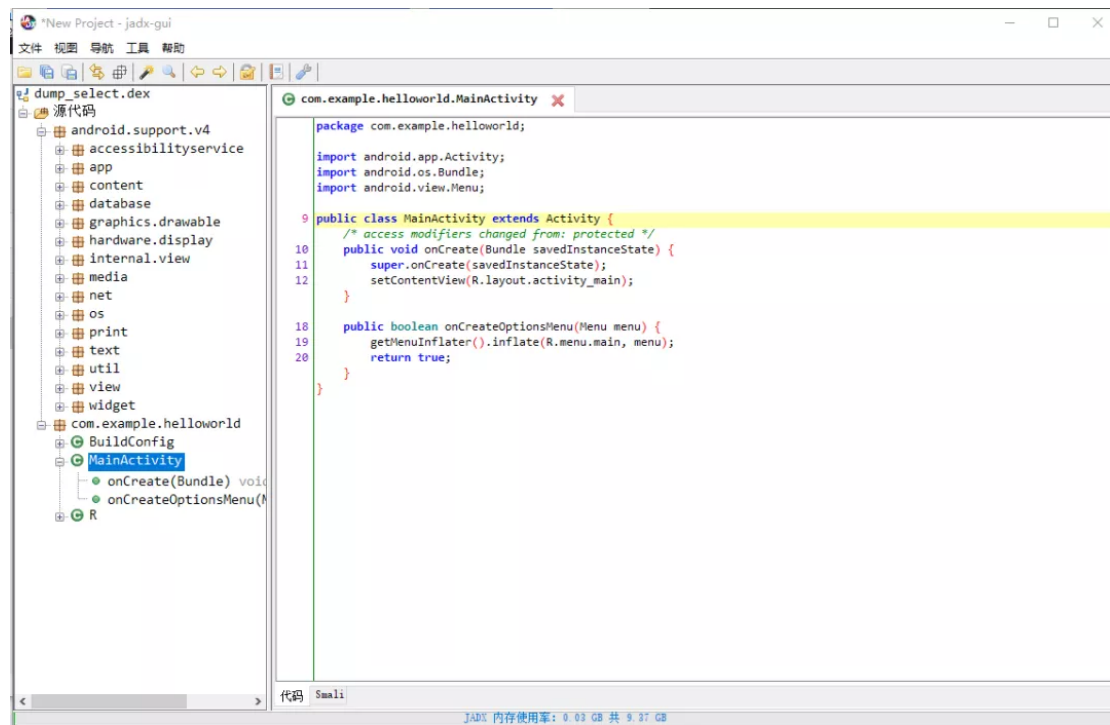


保存后，到dexheard.dex中全选，edit->copy as hex

然后到前面保存好的dex中pastefrom hex

Start-up																	dump_430208h_A7C14h.dex	dump.dex	dumpheard.dex	dump_select.dex
Edit As... Hex * Run Script * Run Template *																				
0 1 2 3 4 5 6 7 8 9 A B C D E F																				
0000h:	64	65	78	0A	30	33	35	00	FC	29	08	05	85	3A	E2	26	0123456789ABCDEF	dex.035.0)...:44		
0010h:	01	C9	74	E8	72	F1	C1	43	82	F7	1B	2C	7E	2E	9D	CD	...EterAAC,...:I			
0020h:	14	7C	0A	00	70	00	00	00	78	56	34	12	00	00	00	00	...p...xV4....			
0030h:	00	00	00	00	40	AE	01	00	65	17	00	00	70	00	00	00	...89...e...p...			
0040h:	4A	03	00	00	04	5E	00	00	44	04	00	00	2C	6B	00	00	...J...D...k...			
0050h:	39	05	00	00	5C	9E	00	00	9D	14	00	00	24	C8	00	00	9...E...E...			
0060h:	31	02	00	00	8C	4D	01	00	04	CD	0B	00	40	A3	01	00	...G...G...			
0070h:	BC	4C	06	00	DE	4C	06	00	E1	4C	06	00	E5	4C	06	00	UL...L...L...			
0080h:	E8	4C	06	00	F0	4C	06	00	07	4D	06	00	2F	4D	06	00	eL...d...M.../M...			
0090h:	3F	4D	06	00	42	4D	06	00	7C	4D	06	00	99	4D	06	00	?M...bM...IM...?M...			
00A0h:	BA	4D	06	00	D2	4D	06	00	F9	4D	06	00	21	4E	06	00	*M...dM...dM...IN...			
00B0h:	42	4E	06	00	4A	4E	06	00	67	4E	06	00	8A	4E	06	00	BN...JN...gN...SN...			
00C0h:	A7	4E	06	00	B5	4E	06	00	C4	4E	06	00	D2	4E	06	00	SN...pN...AN...ON...			
00D0h:	E0	4E	06	00	01	4F	06	00	0F	4F	06	00	22	4F	06	00	dN...O...O...O...			
00E0h:	31	4F	06	00	3F	4F	06	00	52	4F	06	00	6A	4F	06	00	1O...7O...RO...JO...			
00F0h:	85	4F	06	00	91	4F	06	00	A7	4F	06	00	AB	4F	06	00	_O...O...SO...eO...			
0100h:	AF	4F	06	00	F9	4F	06	00	03	50	06	00	08	50	06	00	"O...dO...P...P...			
0110h:	0E	50	06	00	1E	50	06	00	2B	50	06	00	43	50	06	00	.F...P...+P...CP...			
0120h:	4C	50	06	00	5D	50	06	00	69	50	06	00	7B	50	06	00	1F...jP...iP...iP...			
0130h:	87	50	06	00	8D	50	06	00	97	50	06	00	CB	50	06	00	+P...P...-P...EP...			
0140h:	F6	50	06	00	21	51	06	00	4D	51	06	00	7D	51	06	00	dP...lQ...MQ...lQ...			
0150h:	A8	51	06	00	D2	51	06	00	FD	51	06	00	27	52	06	00	"Q...dQ...yQ...R...			
0160h:	50	52	06	00	86	52	06	00	9F	52	06	00	D1	52	06	00	PR...rR...yR...NR...			
0170h:	11	53	06	00	2E	53	06	00	51	53	06	00	5D	53	06	00	.S...S...QS...JS...			
0180h:	6A	53	06	00	74	53	06	00	7D	53	06	00	93	53	06	00	jS...tS...jS...S...			
0190h:	BA	53	06	00	C0	53	06	00	C8	53	06	00	CE	53	06	00	*S...AS...ES...IS...			
01A0h:	E8	53	06	00	F9	53	06	00	2D	54	06	00	39	54	06	00	dS...dS...T...ST...			
01B0h:	4B	54	06	00	65	54	06	00	8F	54	06	00	9B	54	06	00	KI...eT...T...T...			
01C0h:	AD	54	06	00	CA	54	06	00	E2	54	06	00	F0	54	06	00	-T...ET...ET...ST...			
01D0h:	01	55	06	00	14	55	06	00	22	55	06	00	2F	55	06	00	.U...U...U...U...			
01E0h:	3D	55	06	00	4C	55	06	00	58	55	06	00	65	55	06	00	=U...LU...XU...eU...			
01F0h:	6F	55	06	00	88	55	06	00	95	55	06	00	AC	55	06	00	oU...U...U...U...			
0200h:	BC	55	06	00	CD	55	06	00	E2	55	06	00	F4	55	06	00	uU...IU...AU...OU...			
0210h:	01	56	06	00	15	56	06	00	1F	56	06	00	2B	56	06	00	.V...V...V...+V...			
0220h:	39	56	06	00	4E	56	06	00	5D	56	06	00	69	56	06	00	9V...NV...jV...iV...			
0230h:	71	56	06	00	87	56	06	00	9C	56	06	00	B0	56	06	00	qV...+V...eV...V...			
0240h:	B8	56	06	00	C3	56	06	00	DE	56	06	00	E5	56	06	00	.V...AV...BV...AV...			
0250h:	E8	56	06	00	F8	56	06	00	FF	56	06	00	06	57	06	00	eV...eV...yV...W...			
0260h:	16	57	06	00	22	57	06	00	40	57	06	00	4A	57	06	00	.W...W...W...W...			
0270h:	62	57	06	00	7E	57	06	00	84	57	06	00	96	57	06	00	bW...-W...W...-W...			
0280h:	B2	57	06	00	E8	57	06	00	F3	57	06	00	FB	57	06	00	*W...eW...dW...dW...			
0290h:	18	58	06	00	56	58	06	00	5F	58	06	00	63	58	06	00	.X...VX...X...cX...			

打开dex



完美脱壳!