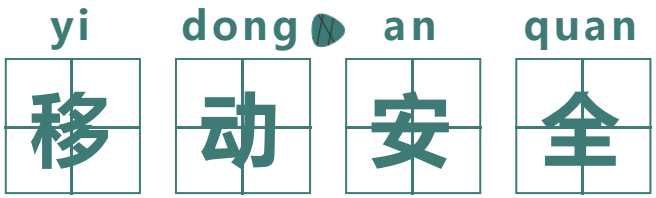
移动安全（十）|TengXun加固动态脱壳（下篇）

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EIF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

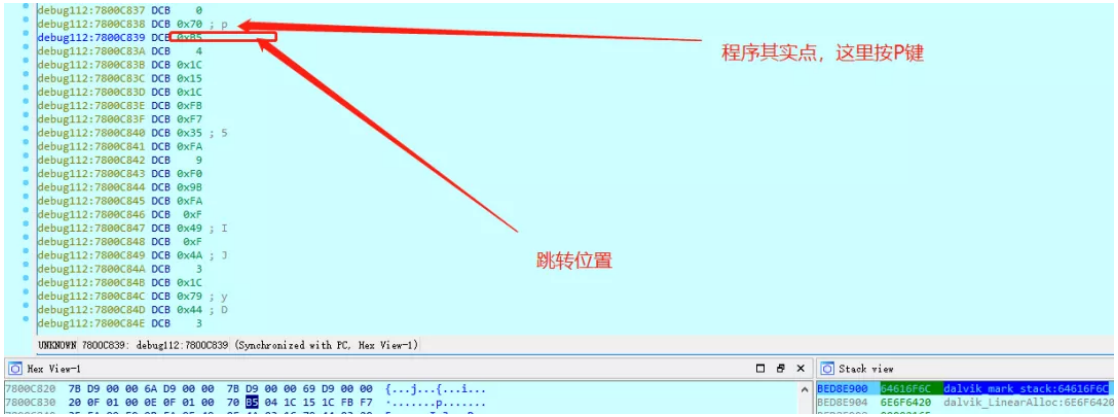
0x03实验开始

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

发现是关键的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

```
1  int __fastcall sub_7800C838(int a1, int a2, int a3)
2  {
3    int v3; // r4
4    int v4; // r5
5    int v5; // r0
6    int v6; // r0
7    int result; // r0
8    int v8; // r6
9
10   v3 = a1;
11   v4 = a3;
12   v5 = (unk_78007CAC)();
13   v6 = (unk_78015D7C)(v5);
14   (unk_78015C3C)(3, "SecShell", "Start load %d", v6);
15   result = (unk_78007DEC)(v3);
16   if ( result )
17   {
18     v8 = (unk_780093DC)(v3);
19     (unk_78005240)(v3);
20     if ( v8 )
21       result = (unk_7800A378)(v3, v4);
22     else
23       result = (unk_7800BCD0)(v3, v4);
24   }
25   return result;
26 }
```

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_8C= -0x8C
debug112:7800BCD0 var_78= -0x78
debug112:7800BCD0 var_1C= -0x1C
debug112:7800BCD0
debug112:7800BCD0 PUSH            {R4-R7,LR}
debug112:7800BCD2 MOVS            R6, R0
debug112:7800BCD4 LDR             R0, =(unk_7801CE70 - 0x7800BCDE)
debug112:7800BCD6 LDR             R3, =0xFFFFFF34
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 MOVS            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,LR}
debug112:7800BCD2 MOVS            R6, R0
debug112:7800BCD4 LDR             R0, =(unk_7801CE70 - 0x7800BCDE)
debug112:7800BCD6 LDR             R3, =0xFFFFFF34
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 MOVS            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：

```
    v75 = 0;
  }
  else
  {
    v18 = (unk_780087A4)(v67) - 40;
    v75 = 1;
  }
}
v19 = (unk_780078CA)(v18 + 40);
(unk_78015C3C)(3, "SecShell", "orgDexOffset:%d", v19);
(unk_78015C9C)(&v98, 0, 112);
v89 = v18 + v19 + 40;
(unk_78015D1C)(&v98, v18 + v19 + 40, 112);
(unk_78015C9C)(&v93, 0, 16);
(unk_7800EE4C)(&v93, &v98, 112, 32);
v80 = v18 + v19 + 40;
v70 = v100;
v20 = (unk_78015C3C)(3, "SecShell", "fileSize:%d", v100);
if ( v75 )
{
  v21 = v70;
  if ( v70 & 0xFFF )
    v21 = (v70 / 4096 + 1) << 12;
  v20 = (unk_78015C7C)(v18, v21, 3);
  if ( v20 )
  {
```

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：

```
    v75 = 1;
  }
}
v19 = (sub_780078CA)(v18 + 0x28);
(unk_78015C3C)(3, "SecShell", "orgDexOffset:%d", v19);
(unk_78015C9C)(&v98, 0, 112);
v89 = v18 + v19 + 40;
(unk_78015D1C)(&v98, v18 + v19 + 40, 112);
(unk_78015C9C)(&v93, 0, 16);
(unk_7800EE4C)(&v93, &v98, 112, 32);
v80 = v18 + v19 + 40;
v70 = v100;
v20 = (unk_78015C3C)(3, "SecShell", "fileSize:%d", v100);
if ( v75 )
{
  v21 = v70;
  if ( v70 & 0xFFF )
    v21 = (v70 / 4096 + 1) << 12;
UNKNOWN sub_7800BCD0:241 (7800C0BE)
```

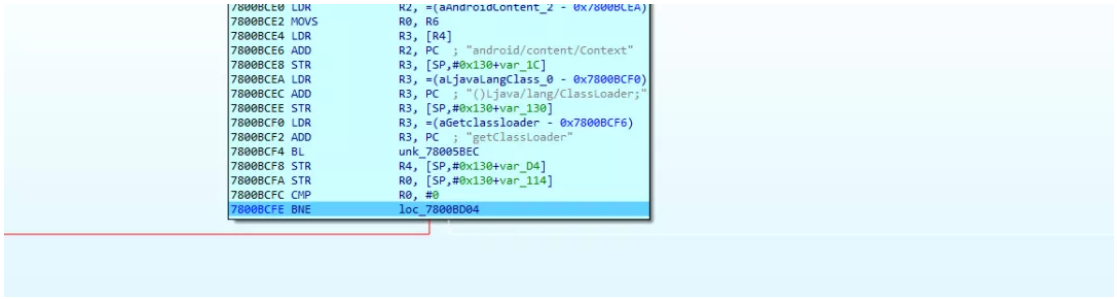
忽然看到，这里不是0x28固定了吗，里面计算偏移量，为v19

那么问题是下面的文件大小有何用呢?

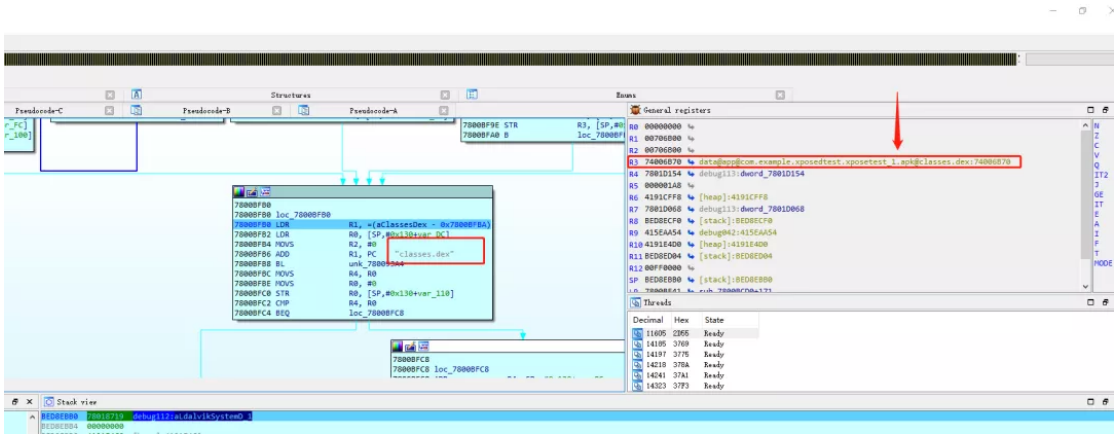
经过测试，经过查看，还有上面的两个classes.dex
推断出，第一次解密出来的只是程序的执行后部分，而第二次解密的是dex的文件头
需要两部分综合

```
Hex View-1
8008CC0  71 C0 00 00 4D E4 00 00  97 E4 00 00 34 04 00 00  q...M.......4...
8008CD0  F0 B5 06 1C D3 48 24 40  C7 B0 78 44 12 90 C4 58  .....H$@..xD...X
8008CE0  D2 4A 30 1C 23 6B 7A 44  45 93 D1 4B 7B 44 00 93  .J0.#kzDE..K{D..
8008CF0  D0 4B 7B 44 F9 F7 7A FF  17 94 07 90 00 28 01 D1  .K{D..z.....(..
8008D00  00 F0 5A FD CC 49 30 1C  CC 4D 79 44 F9 F7 B2 FD  ..Z..I0..MyD....
8008D10  CB 4A 04 1C 7D 44 2B 1C  21 1C 7A 44 30 1C FB F7  .J..}D+.!.zD0...
8008D20  A1 FD 21 1C 02 1C 30 1C  FB F7 A3 FD 0E 90 0E 99  ..!...0.........
8008D30  30 1C FA F7 3F F8 C3 4A  2B 1C 0A 90 21 1C 7A 44  0...?..J+...!.zD
8008D40  30 1C FB F7 8F FD 21 1C  02 1C 30 1C FB F7 91 FD  0.....!...0.....
8008D50  01 1C 30 1C FA F7 2E F8  B8 4B BC 4A 12 99 7B 44  ..0.....K.J..{D

Stack view
BED8ECE0  2D82E960  dalvik-LinearAlloc:6D82E960
BED8ECE4  4191E4C0  [heap]:4191E4C0
BED8ECE8  00000000
BED8ECEC  414FEBD0  libdvm.so:dvmPlatformInvoke+74
BED8ECF0  415EAA54  debug842:415EAA54
BED8ECF4  00000001
BED8ECF8  774DDF2E  data@app@com.example.helloworld_1.apk@classes.dex:774
BED8ECFC  42CEED70  dalvik_heap:42CEED70
BED8ED00  42CF5D5C  dalvik_heap:42CF5D5C
BED8ED04  4152F127  libdvm.so:_Z16dvmCallJNIMethodPKjP6JValuePK6MethodPvT
```

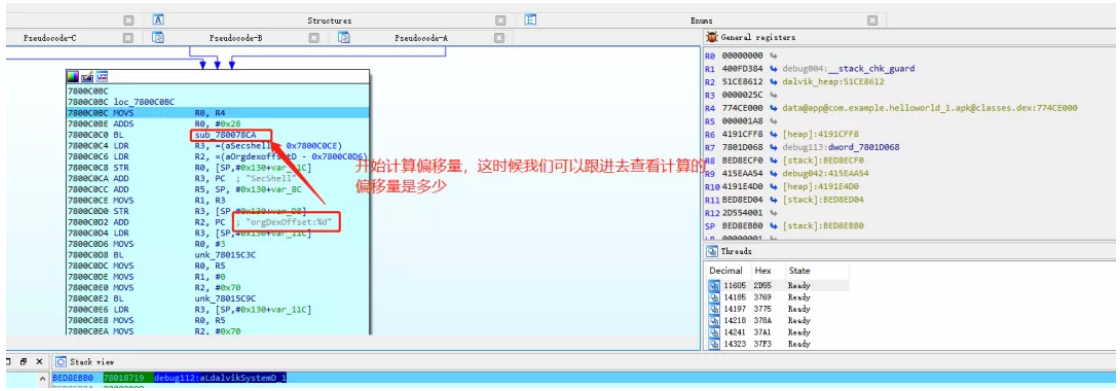
通过执行程序流程图，跟踪程序执行流程：



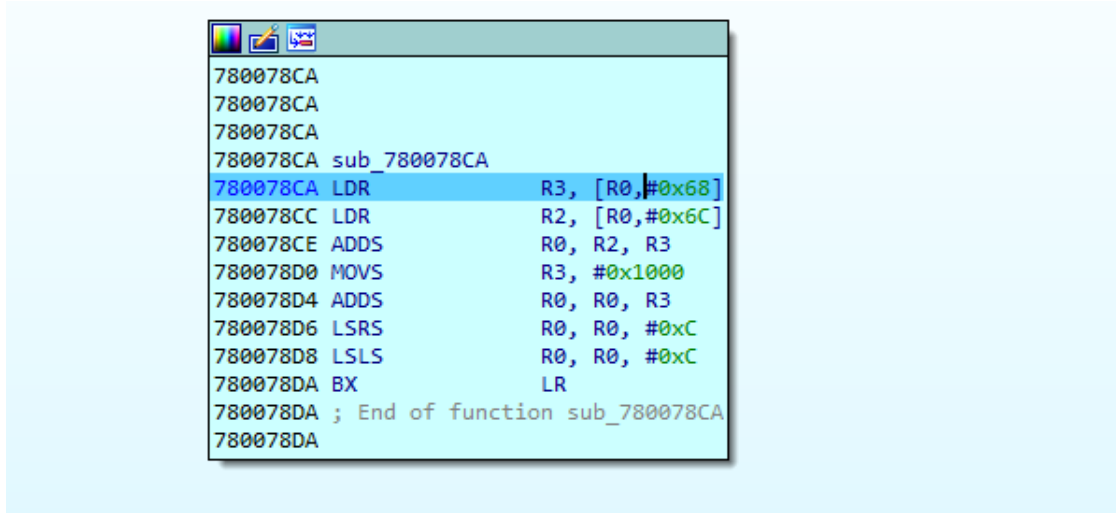
看到这里开始关注寄存器，堆栈，还有汇编的变化，记录每一步的变化：



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



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
```

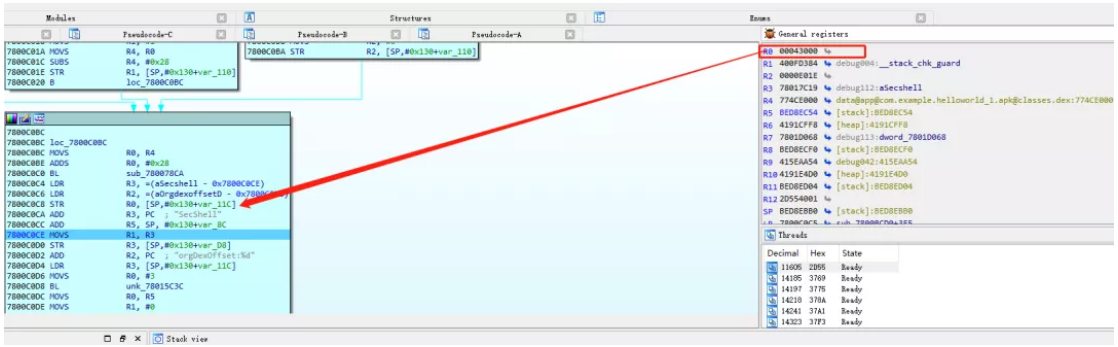
Python

AU: idle Down

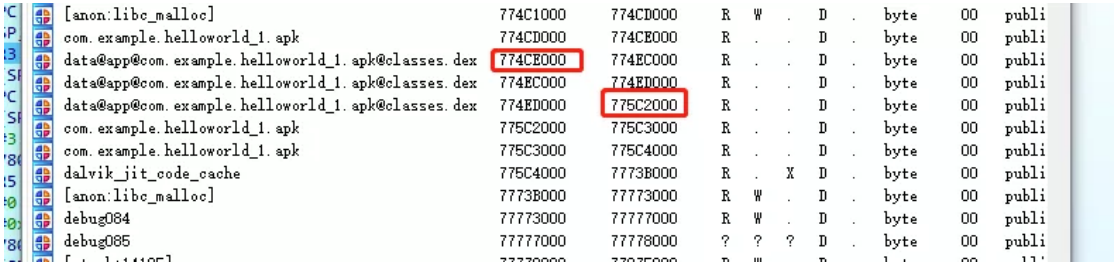
为0x43000

而数据头文件大小为0x43000+0x28=0x43028

跟踪R0

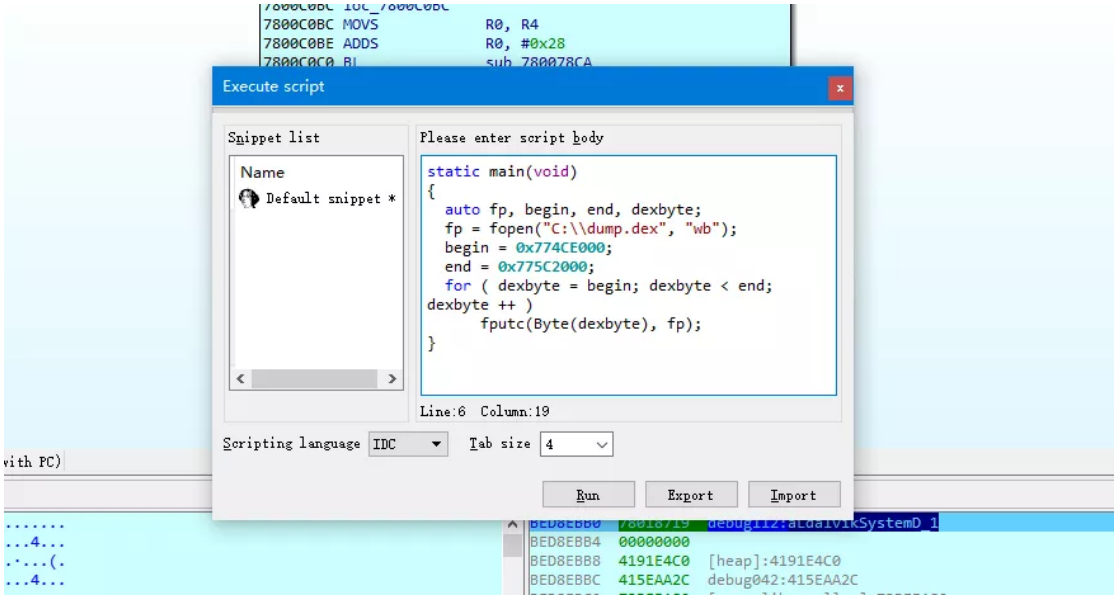


这时候为了方便计算



[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

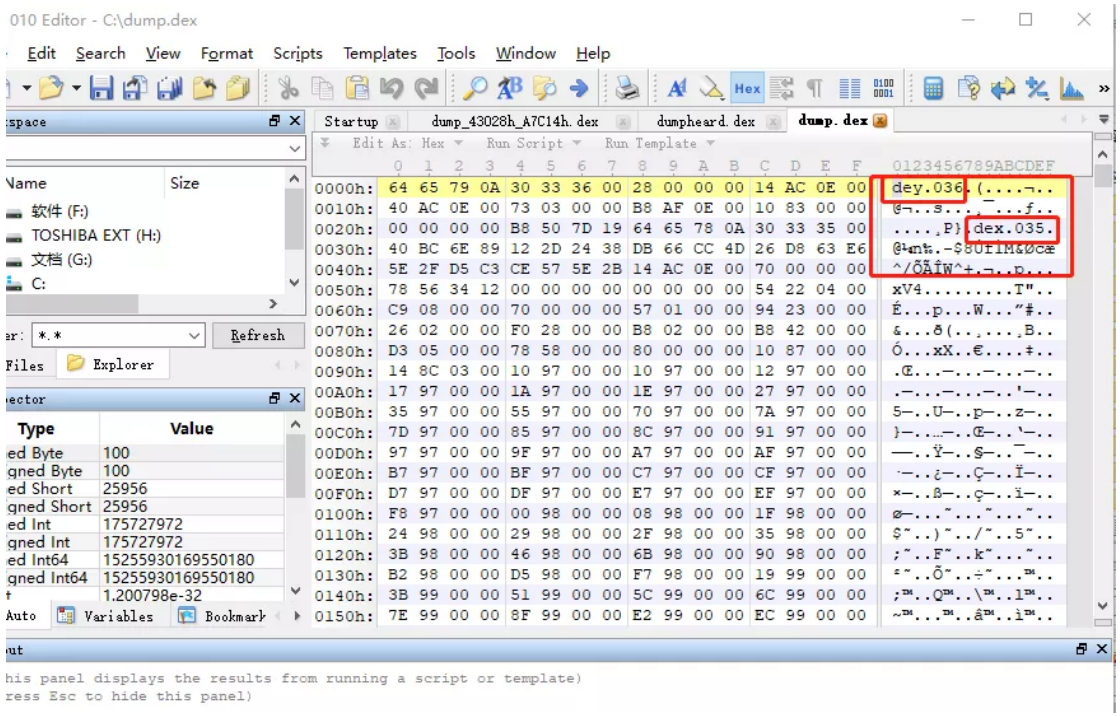
先dump下dex



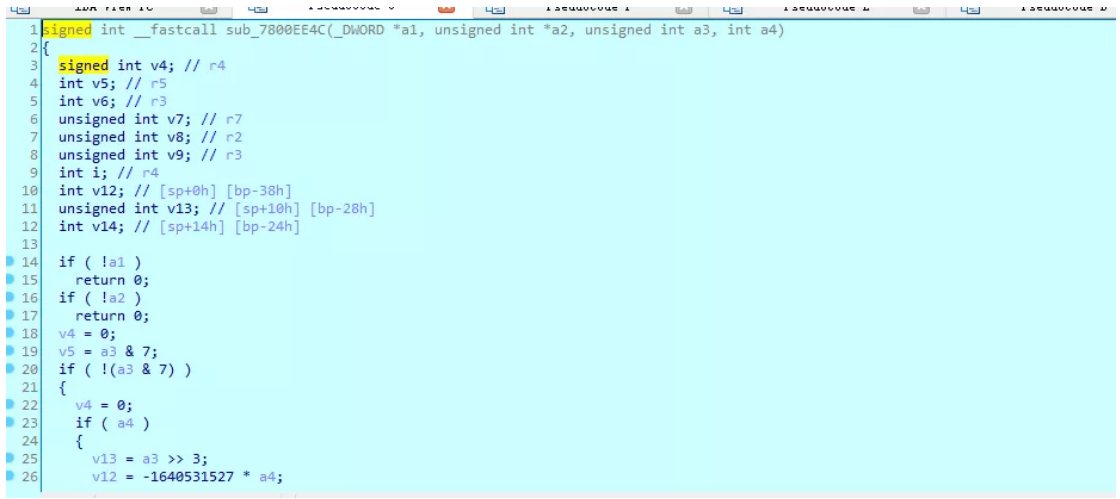
staticmain(void){ autofp,begin,end,dexbyte; fp=fopen("C:\\dump.dex","wb"); begin=0x774CE000; end=0x775C2000; for(dexbyte=begin;dexbyte<end;dexbyte++) fputc(Byte(dexbyte),fp);}

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：



```
signed int __fastcall sub_7800EE4C(_DWORD *a1, unsigned int *a2, unsigned int a3, int a4)
{
  signed int v4; // r4
  int v5; // r5
  int v6; // r3
  unsigned int v7; // r7
  unsigned int v8; // r2
  unsigned int v9; // r3
  int i; // r4
  int v12; // [sp+0h] [bp-38h]
  unsigned int v13; // [sp+10h] [bp-28h]
  int v14; // [sp+14h] [bp-24h]

  if ( !a1 )
    return 0;
  if ( !a2 )
    return 0;
  v4 = 0;
  v5 = a3 & 7;
  if ( !(a3 & 7) )
  {
    v4 = 0;
    if ( a4 )
    {
      v13 = a3 >> 3;
      v12 = -1640531527 * a4;
```

这里对R5进行处理：

```
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：

```
[stack]:BED8EC51 DCB    1
[stack]:BED8EC52 DCB 0x59 ; Y
[stack]:BED8EC53 DCB 0x41 ; A
R5
[stack]:BED8EC54 DCB 0x64 ; d
[stack]:BED8EC55 DCB 0x65 ; e
[stack]:BED8EC56 DCB 0x78 ; x
[stack]:BED8EC57 DCB  0xA
[stack]:BED8EC58 DCB 0x30 ; 0
[stack]:BED8EC59 DCB 0x33 ; 3
[stack]:BED8EC5A DCB 0x35 ; 5
[stack]:BED8EC5B DCB    0
[stack]:BED8EC5C DCB 0xFC
[stack]:BED8EC5D DCB 0x29 ; )
[stack]:BED8EC5E DCB    8
[stack]:BED8EC5F DCB    5
[stack]:BED8EC60 DCB 0x85
[stack]:BED8EC61 DCB 0x3A ; :
[stack]:BED8EC62 DCB 0xE2
[stack]:BED8EC63 DCB 0x26 ; &
[stack]:BED8EC64 DCB    1
[stack]:BED8EC65 DCB 0xC9
[stack]:BED8EC66 DCB 0x74 ; t
[stack]:BED8EC67 DCB 0xE8
[stack]:BED8EC68 DCB 0x72 ; r
[stack]:BED8EC69 DCB 0xF1
```

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：

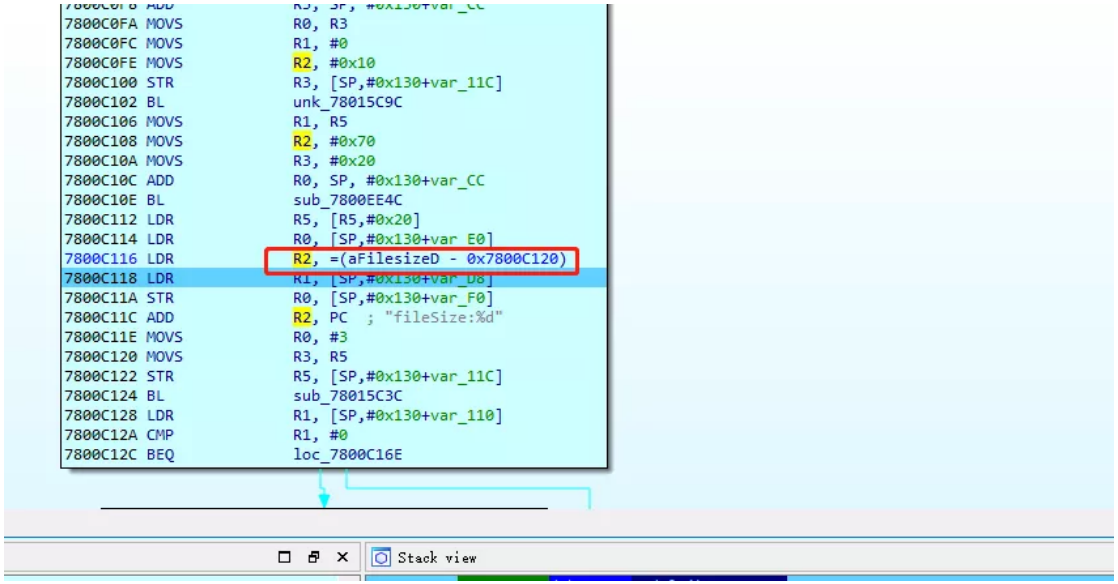
跟踪每一个可疑的数据：

```
7800C100 STR             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]
```

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

```
signed int __fastcall sub_7800EE4C(_DWORD *a1, unsigned int *a2, unsigned int a3, int a4)
{
  signed int v4; // r4
  int v5; // r5
  int v6; // r3
  unsigned int v7; // r7
  unsigned int v8; // r2
  unsigned int v9; // r3
  int i; // r4
  int v12; // [sp+0h] [bp-38h]
  unsigned int v13; // [sp+10h] [bp-28h]
  int v14; // [sp+14h] [bp-24h]

  if ( !a1 )
    return 0;
  if ( !a2 )
    return 0;
  v4 = 0;
  v5 = a3 & 7;
  if ( !(a3 & 7) )
  {
    v4 = 0;
    if ( a4 )
    {
      v13 = a3 >> 3;
      v12 = -1640531527 * a4;
UNKNOWN sub_7800EE4C:1 (7800EE4C)
```

而在输出的时候显示：



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

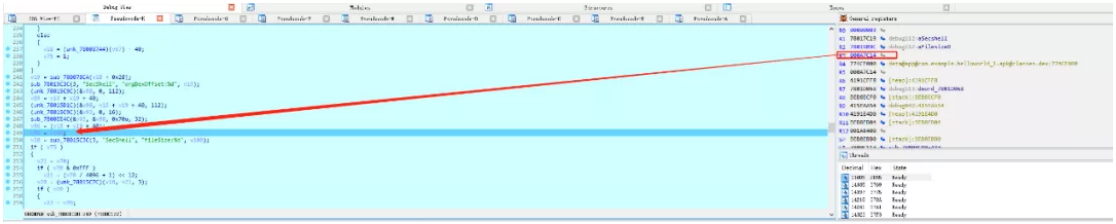
这时候又得回到源程序

```
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 & 0xFFF )
255      v21 = (v70 / 4096 + 1) << 12;
256    v20 = (unk_78015C7C)(v18, v21, 3);
257    if ( v20 )
258    {
259      v22 = v70;
260      if ( v70 & 0xFFF )
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

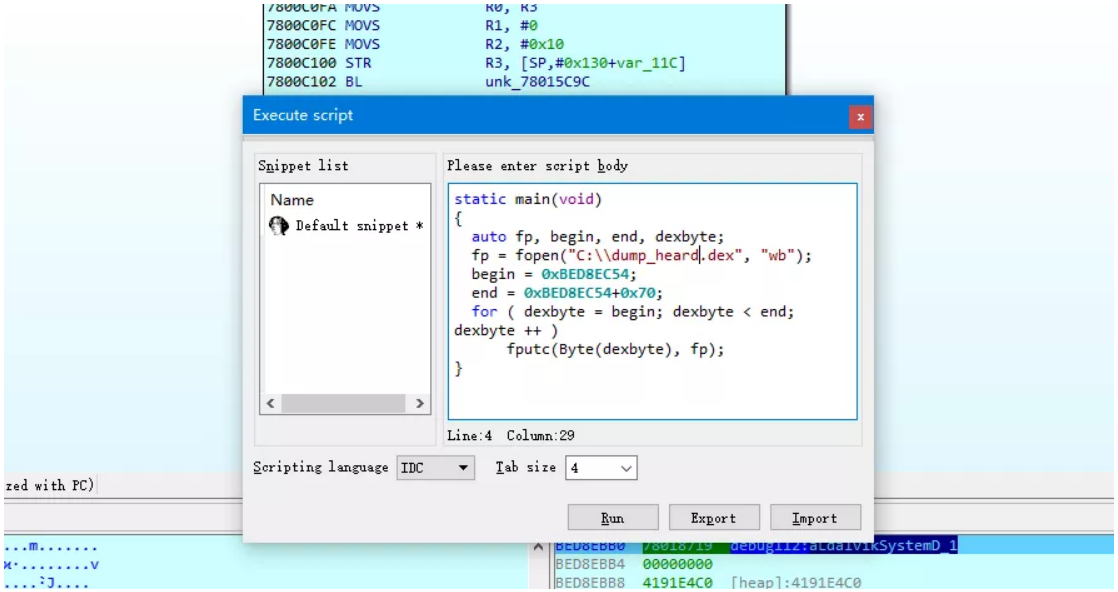
不管那么多，先执行，发现v100指向R3

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：

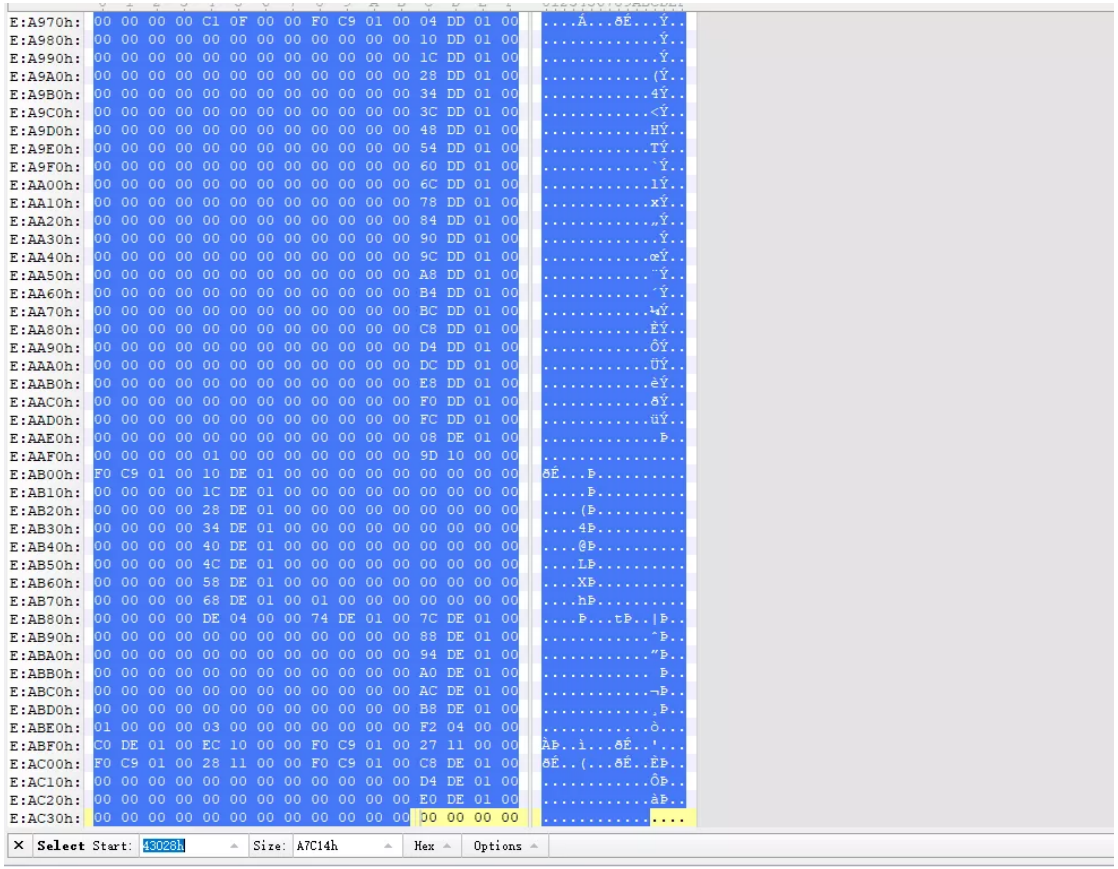
```
staticmain(void)

{

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：

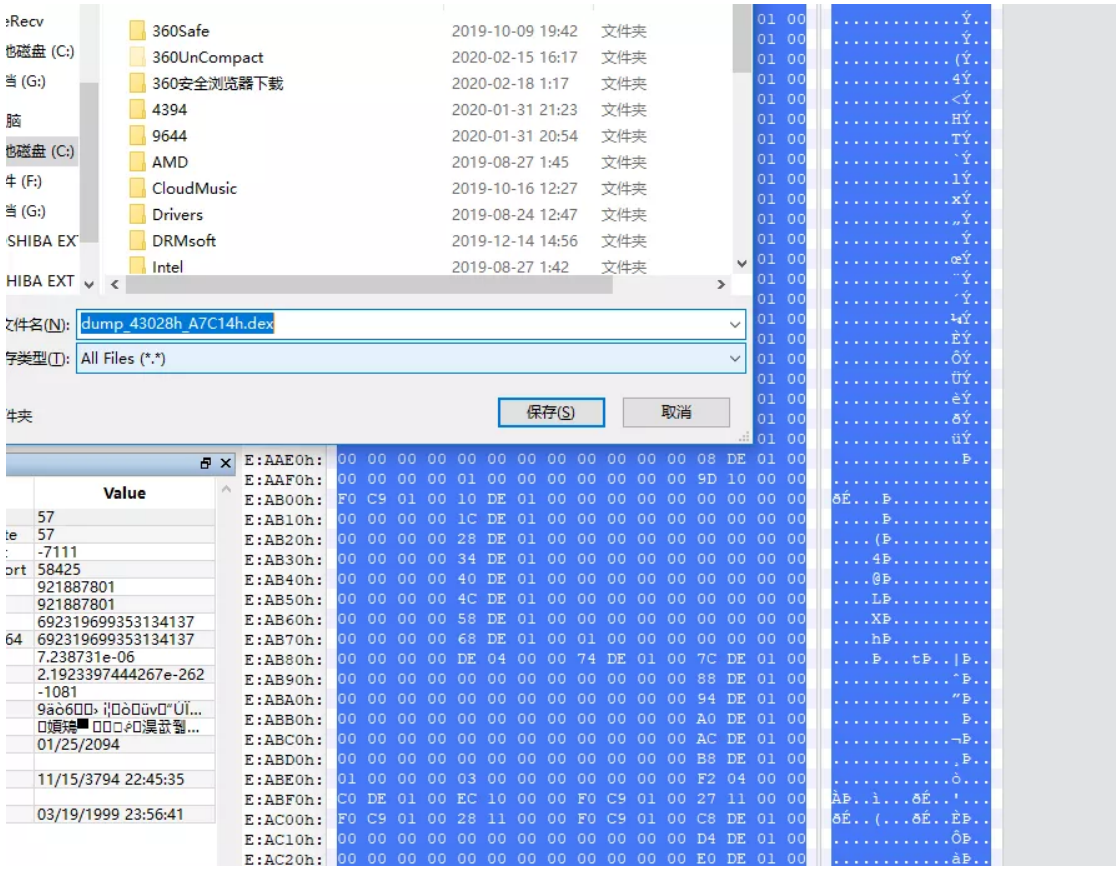
先获取到混淆的dex，然后根据偏移量，计算出起始点位置，然后再解密另一个dex文件头部和文件大小，然后拼接处再前面的dex的偏移量起始点处，这时候在赋上第一个文件大小A7C14

然后使用010editor：

edit->selectrange

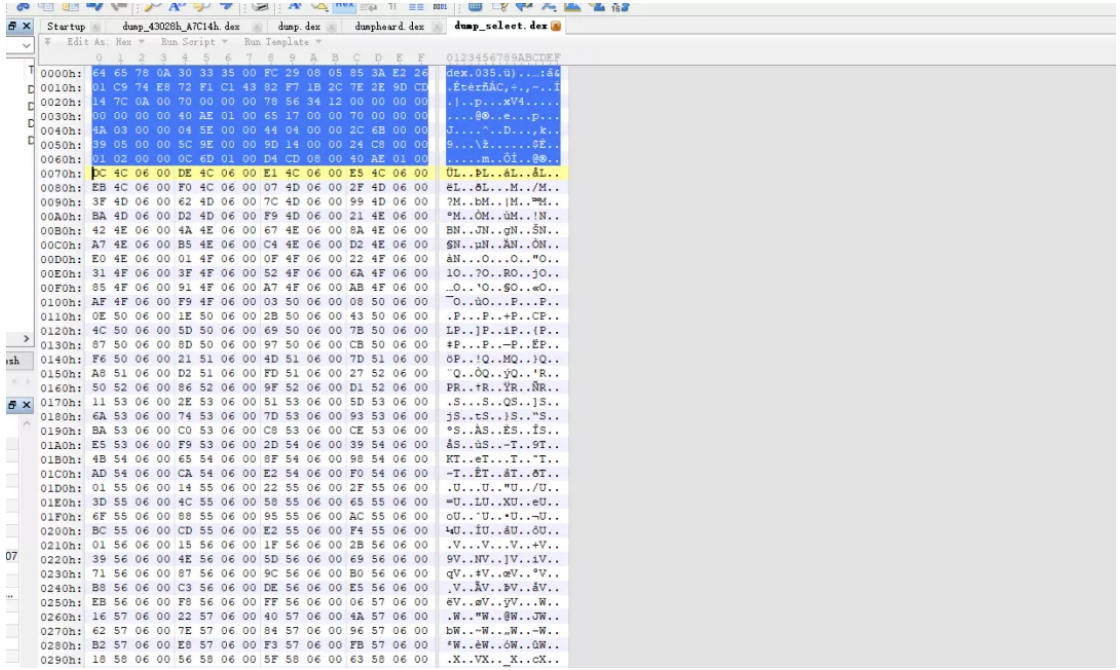


然后File->saveselection as file

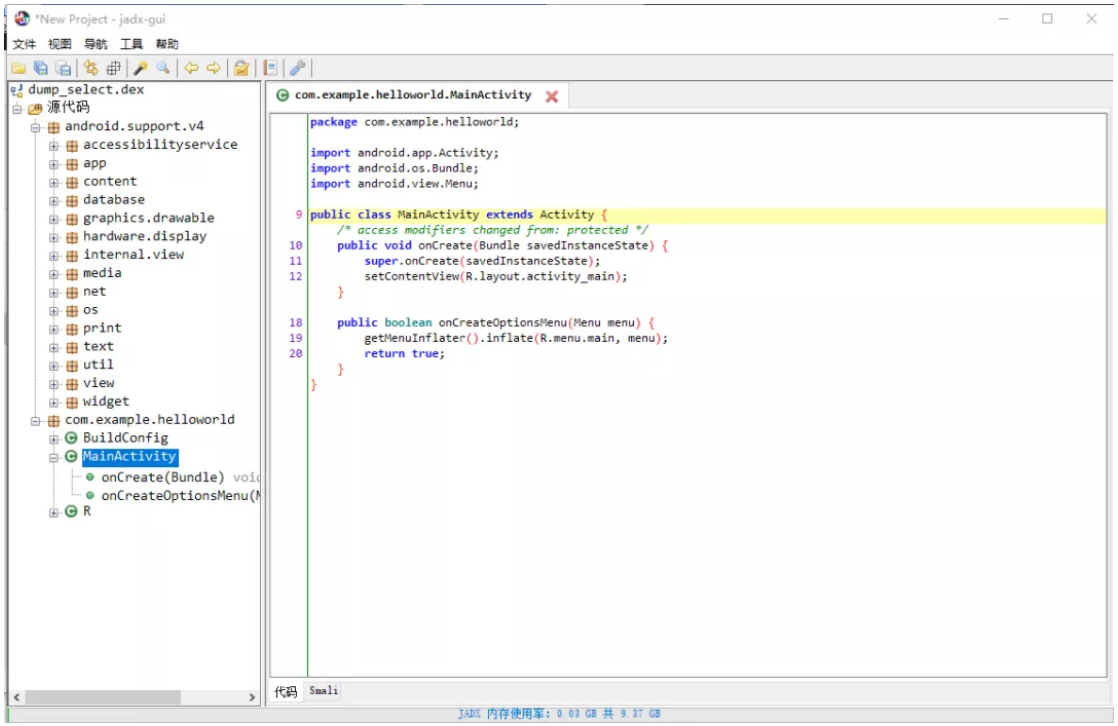


保存后，到dexheard.dex中全选，edit->copy as hex

然后到前面保存好的dex中pastefrom hex


Startup
dump_43028h_A7C14h.dex
dump.dex
dumpheard.dex
dump_select.dex
Edit As: Hex
Run Script
Run Template

打开dex


*New Project - jadx-gui
文件 视图 导航 工具 帮助
dump_select.dex
源代码
android.support.v4
accessibilityservice
app
content
database
graphics.drawable
hardware.display
internal.view
media
net
os
print
text
util
view
widget
com.example.helloworld
BuildConfig
MainActivity
onCreate(Bundle) voi
onCreateOptionsMenu(M
R
com.example.helloworld.MainActivity
package com.example.helloworld;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
public class MainActivity extends Activity {
/* access modifiers changed from: protected */
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
public boolean onCreateOptionsMenu(Menu menu) {
getMenuInflater().inflate(R.menu.main, menu);
return true;
}
}
代码
Smali
JADX 内存使用率: 0.03 GB 共 9.37 GB

完美脱壳!