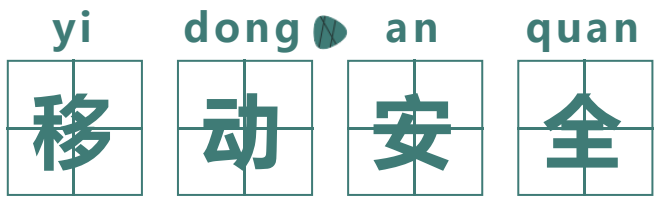


移动安全（九）|TengXun加固动态脱壳（上篇）

辞令WHITECat WHITECat安全小组 3天前



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

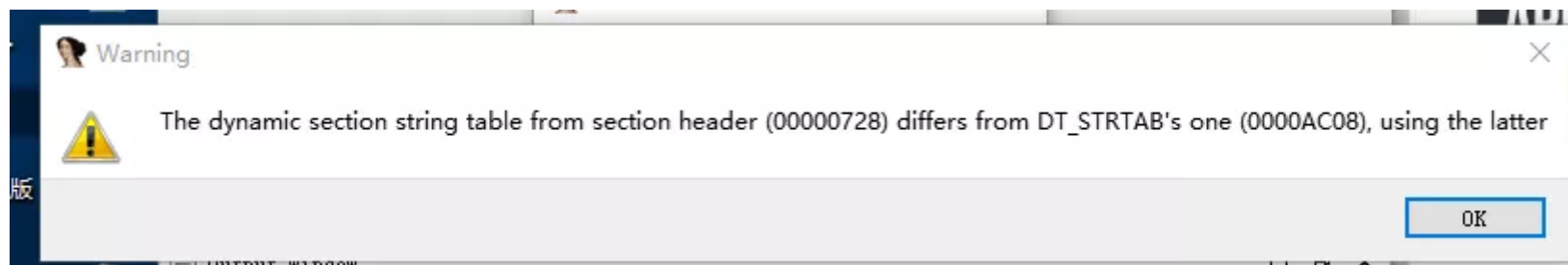
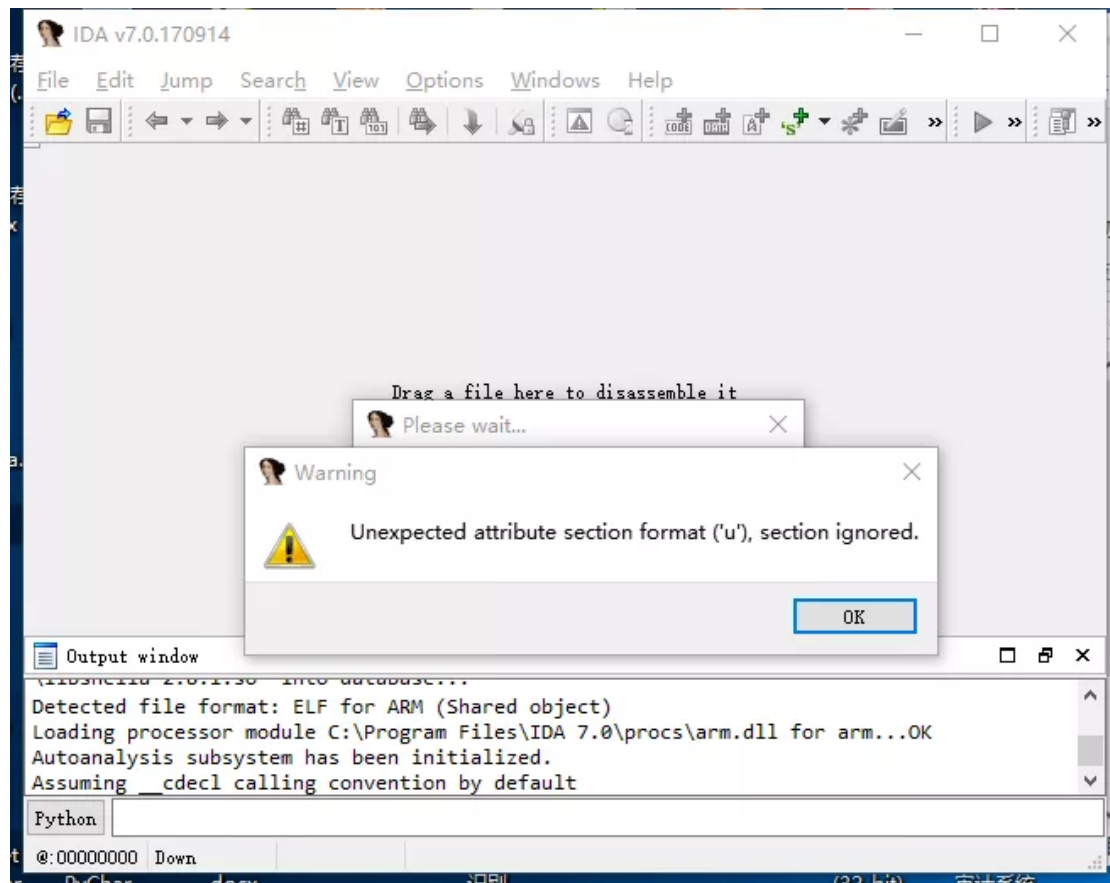
0x03实验开始

1、分析so壳

目标so文件：libshella-2.8.1.so

helloworld.apk\lib\armeabi - 解包大小为 1.3 MB				
名称	压缩前	压缩后	类型	修改时间
.. (上级目录)			文件夹	
libBugly.so	93.4 KB	48.6 KB	SO 文件	2020-08-11 14:00
libshella-2.8.1.so	115.3 KB	93.5 KB	SO 文件	2020-08-11 14:00
libshellx-2.8.1.so	129.0 KB	109.8 KB	SO 文件	2020-08-11 14:00
mix.dex	1 KB	1 KB	DEX 文件	2020-08-11 14:00
mixz.dex	1 KB	1 KB	DEX 文件	2020-08-11 14:00

1.1、使用IDA打开时候若会提示：



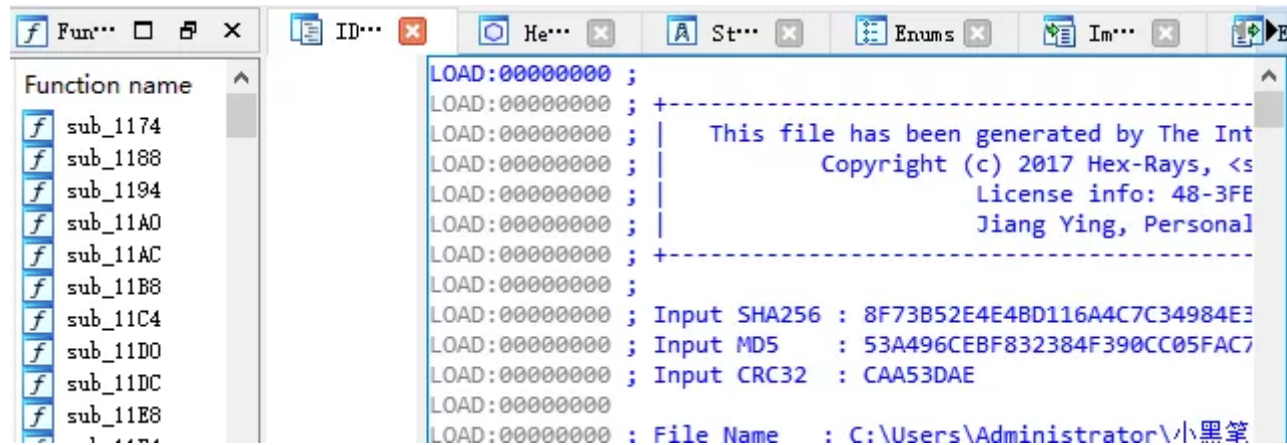
这时候注意关键在sectionignore

如果强行打开，则会出现混淆情况

因为elf文件可以没有section，所以我们就可以把section改为00

1.1.1强行打开

全程OK，强行读取，发现文件始终还是可以被反汇编

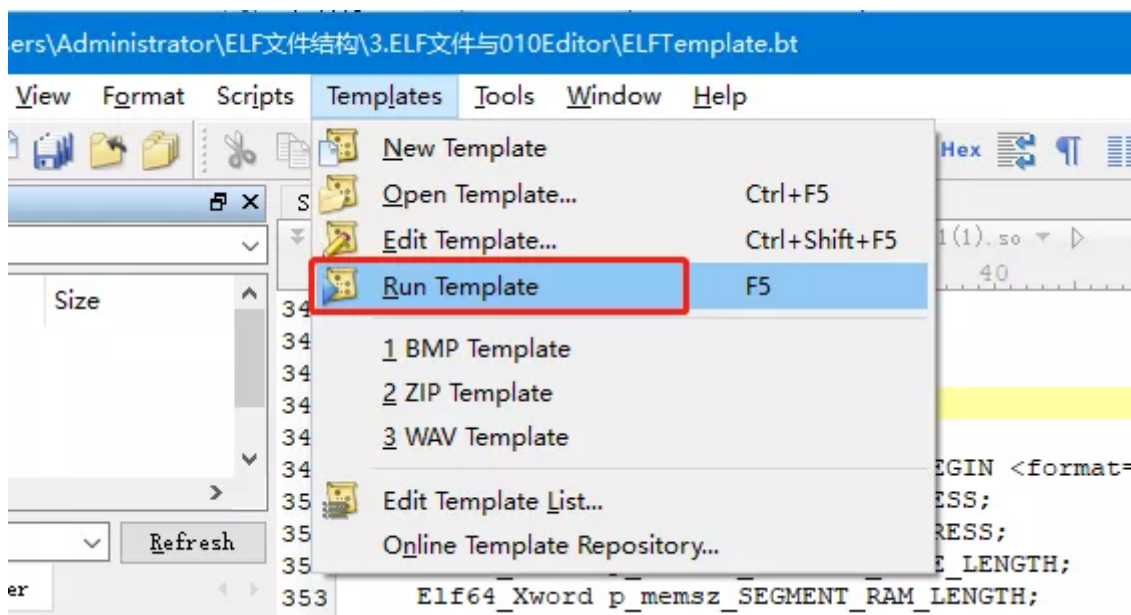
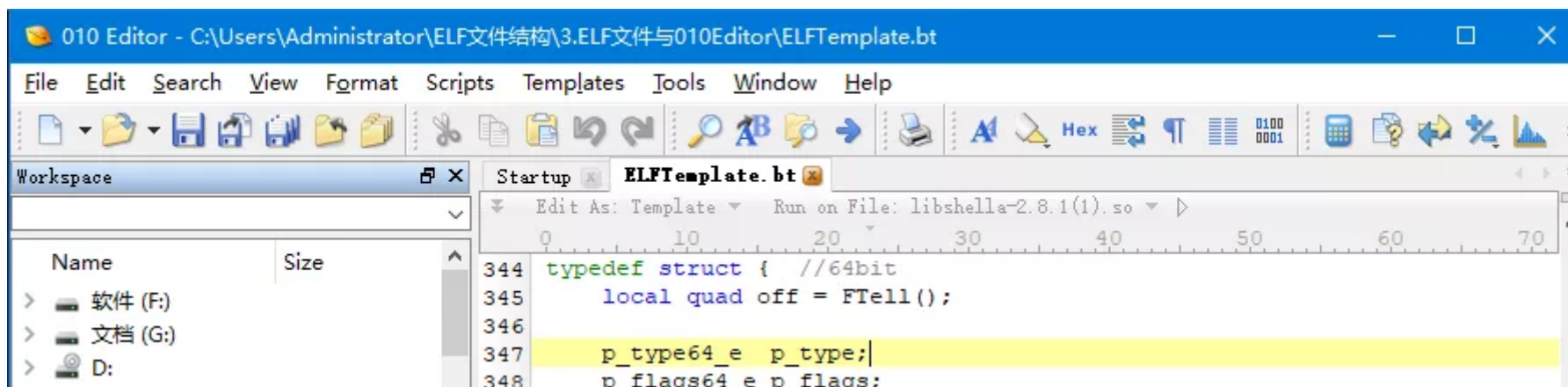


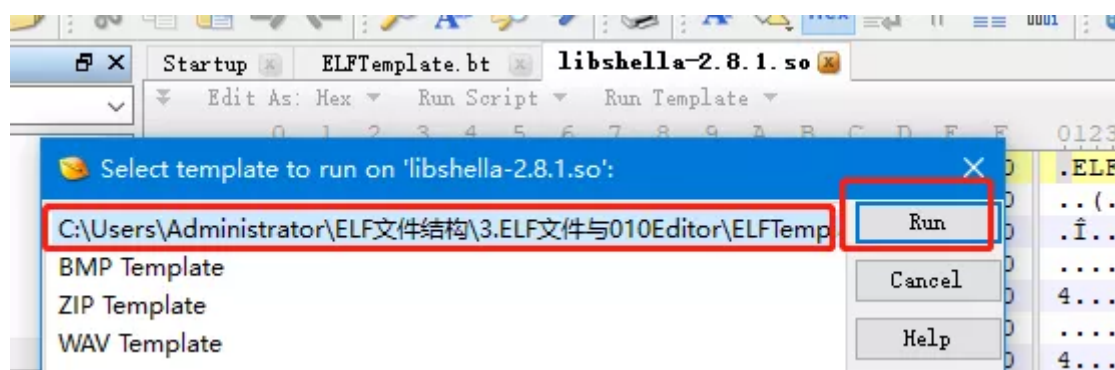
但是函数部分全部变成sub_xxx

1.1.2使用010editor修改

打开010editor，若未激活最好先激活

1.1.3使用ELF解析器，导入bt文件，然后启用

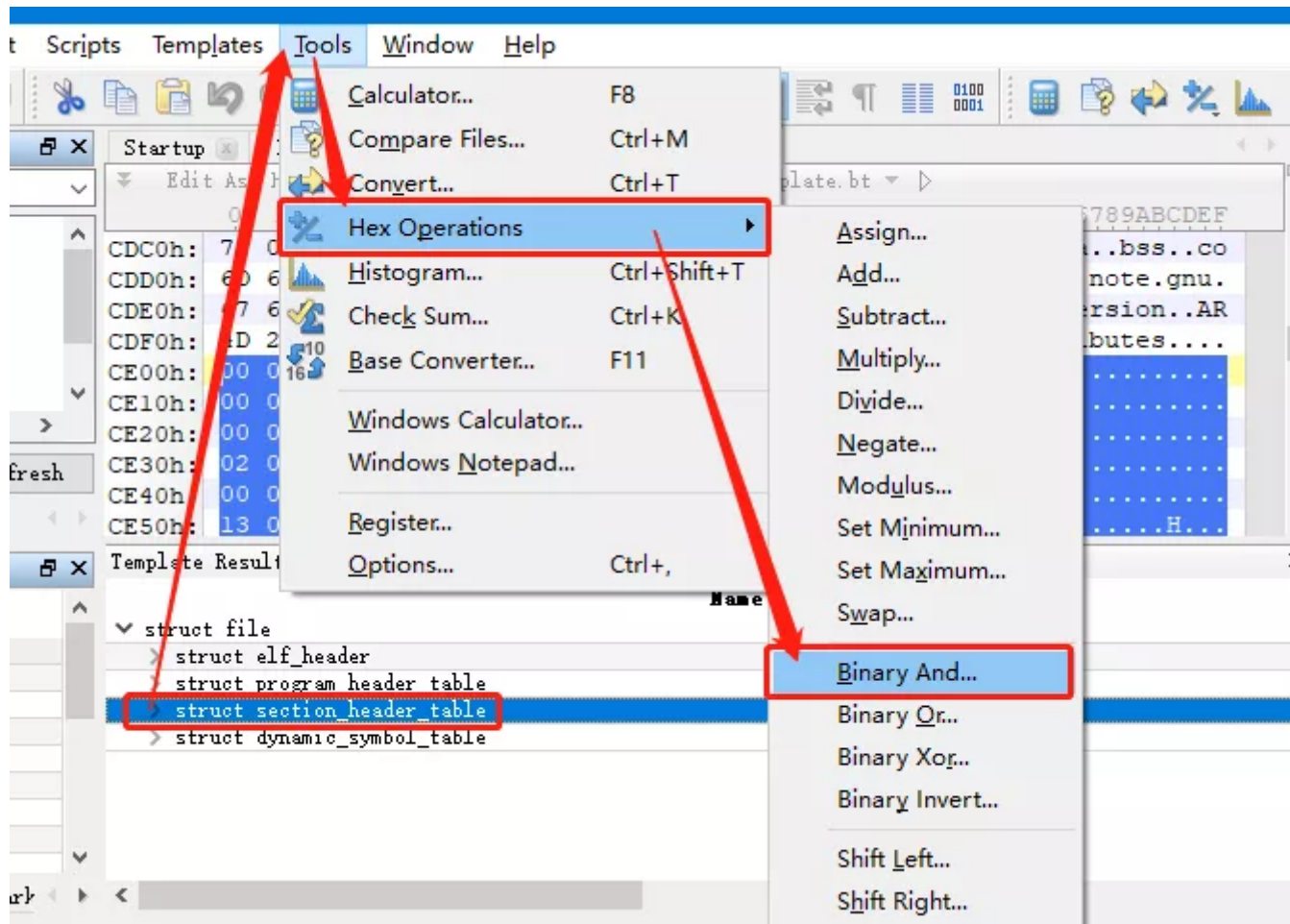


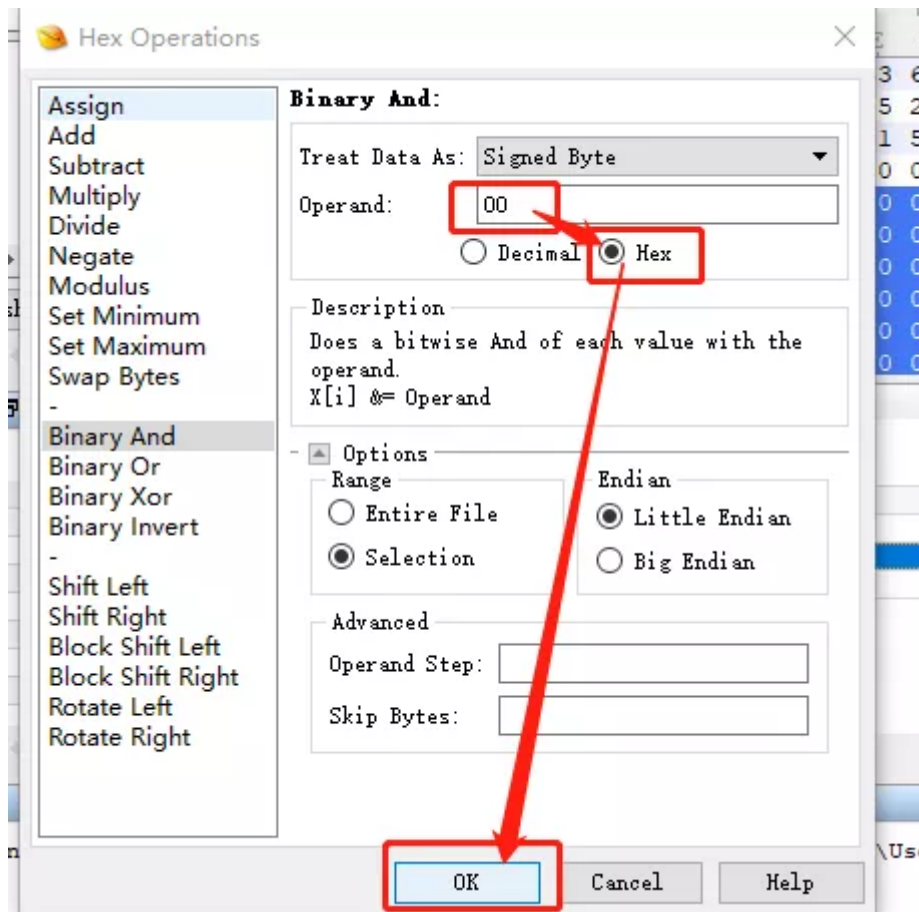


然后选中section_header_table

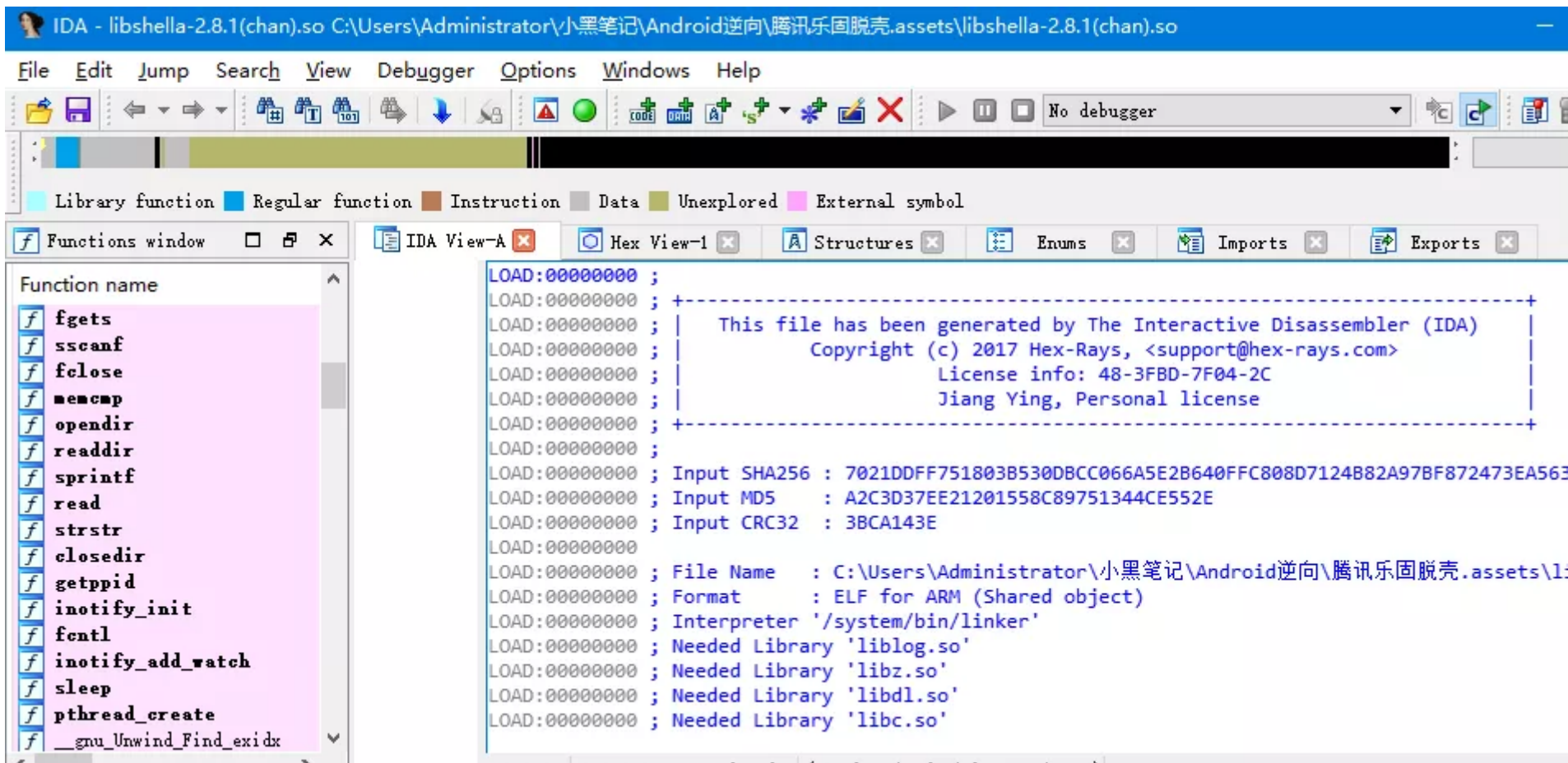
使用tool中的and操作改为hex的00







1.1.4另存为并重新放入ida



已经正常显示，这时候就可以静态分析了

1.2、jni函数被加密

Dynamic section at offset 0x8e2c contains 26 entries:

标记	类型	名称/值
0x00000003	(PLTGOT)	0x9f44
0x00000002	(PLTRELSZ)	352 (bytes)
0x00000017	(JMPREL)	0x1014
0x00000014	(PLTREL)	REL
0x00000011	(REL)	0xfbc
0x00000012	(RELSZ)	88 (bytes)
0x00000013	(RELENT)	8 (bytes)
0x6fffffff	(RELCOUNT)	8
0x00000006	(SYMTAB)	0xa394
0x0000000b	(SYMENT)	16 (bytes)
0x00000005	(STRTAB)	0xbc08
0x0000000a	(STRSZ)	5764 (bytes)
0x00000004	(HASH)	0xd290
0x00000001	(NEEDED)	共享库: [liblog.so]
0x00000001	(NEEDED)	共享库: [libz.so]
0x00000001	(NEEDED)	共享库: [libdl.so]
0x00000001	(NEEDED)	共享库: [libc.so]
0x0000000e	(SONAME)	Library soname: [wellsyang]
0x0000001a	(FINI_ARRAY)	0x9e18
0x0000001c	(FINI_ARRAYSZ)	8 (bytes)
0x00000019	(INIT_ARRAY)	0x9e20
0x0000001b	(INIT_ARRAYSZ)	12 (bytes)
0x00000010	(SYMBOLIC)	0x0
0x0000001e	(FLAGS)	SYMBOLIC BIND_NOW
0x6fffffff	(FLAGS_1)	标志: NOW
0x00000000	(NULL)	0x0

1.2.2分析.init_array

找到.init函数执行位置，使用IDA查看：

07

.dynamic

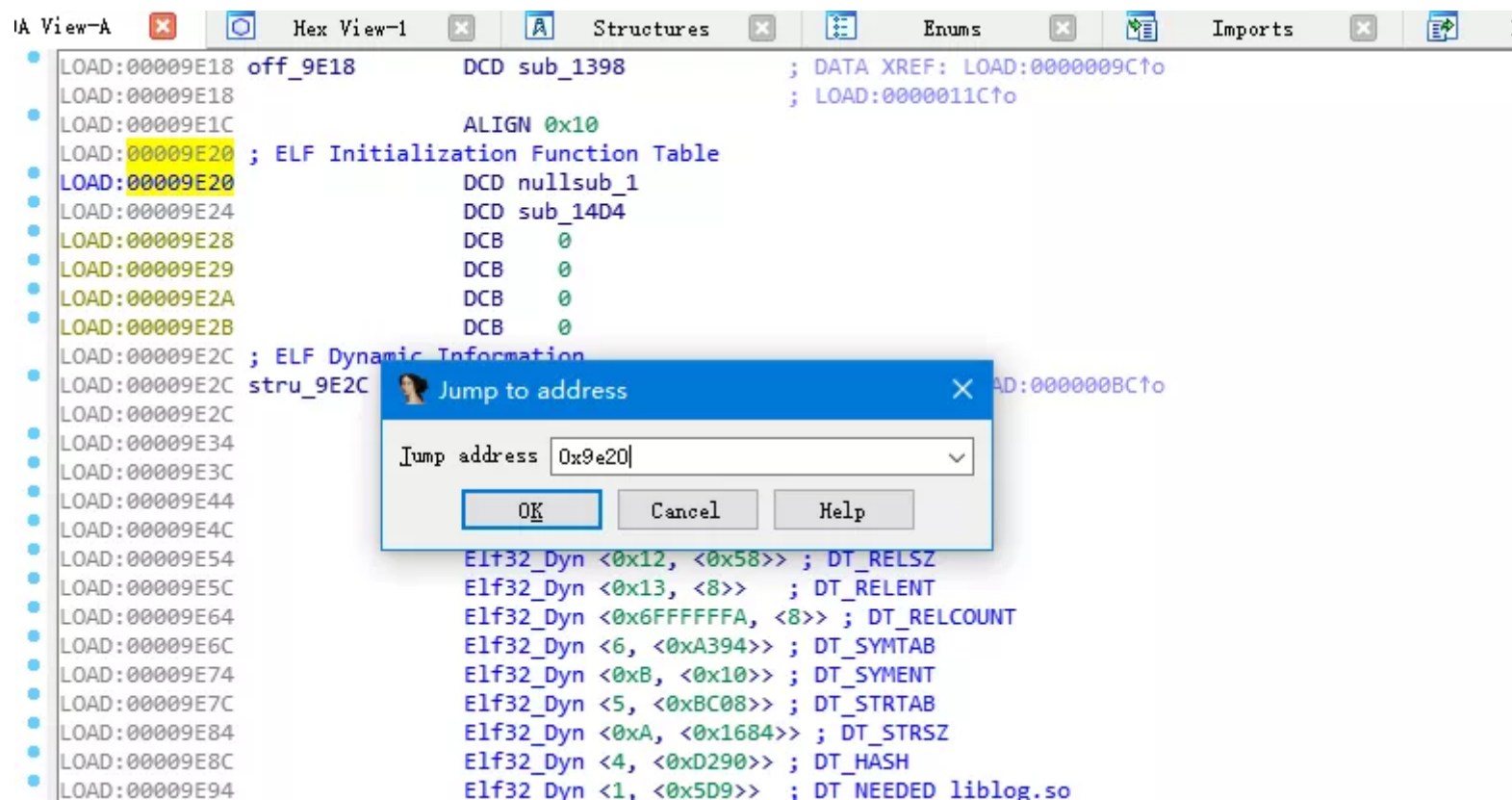
Dynamic section at offset 0x8e2c contains 26 entries:

标记	类型	名称/值
0x00000003	(PLTGOT)	0x9f44
0x00000002	(PLTRELSZ)	352 (bytes)
0x00000017	(JMPREL)	0x1014
0x00000014	(PLTREL)	REL
0x00000011	(REL)	0xfbc
0x00000012	(RELSZ)	88 (bytes)
0x00000013	(RELENT)	8 (bytes)
0x6fffffff	(RELCOUNT)	8
0x00000006	(SYMTAB)	0xa394
0x0000000b	(SYMENT)	16 (bytes)
0x00000005	(STRTAB)	0xbc08
0x0000000a	(STRSZ)	5764 (bytes)
0x00000004	(HASH)	0xd290
0x00000001	(NEEDED)	共享库 : [liblog.so]
0x00000001	(NEEDED)	共享库 : [libz.so]
0x00000001	(NEEDED)	共享库 : [libdl.so]
0x00000001	(NEEDED)	共享库 : [libc.so]
0x0000000e	(SONAME)	Library soname: [wellsyang]
0x0000001a	(FINI_ARRAY)	0x9e18
0x0000001c	(FINI_ARRAYSZ)	8 (bytes)
0x00000019	(INIT_ARRAY)	0x9e20
0x0000001b	(INIT_ARRAYSZ)	12 (bytes)
0x00000010	(SYMBOLIC)	0x0
0x0000001e	(FLAGS)	SYMBOLIC BIND_NOW
0x6fffffff	(FLAGS_1)	标志 : NOW
0x00000000	(NULL)	0x0

剩余空间 : 13.1 GB

0x9e20

(使用快捷键g) 直接搜索



接下来进入到sub_14D4

F5编译一下:

LOAD:000014D4		STMFD	SP!, {R4,R10,R11,LR}
LOAD:000014D4		ADD	R11, SP, #8
LOAD:000014D8		SUB	SP, SP, #0x50
LOAD:000014DC		LDR	R0, =(_GLOBAL_OFFSET_TABLE_ - 0x14EC)
LOAD:000014E0		ADD	R0, PC, R0 ; _GLOBAL_OFFSET_TABLE_
LOAD:000014E4		LDR	R1, =(sub_14D4 - 0x9F44)
LOAD:000014EC		ADD	R1, R1, R0 ; sub_14D4
LOAD:000014F0		LDR	R2, =0xFFFFF000
LOAD:000014F4		AND	R1, R1, R2
LOAD:000014F8		STR	R1, [R11,#var_C]
LOAD:000014FC		STR	R0, [SP,#0x58+var_48]
LOAD:00001500			
LOAD:00001500	loc_1500		; CODE XREF: sub_14D4+4C↓j
LOAD:00001500		LDR	R0, =0x464C457F
LOAD:00001504		LDR	R1, [R11,#var_C]
LOAD:00001508		LDR	R1, [R1]
LOAD:0000150C		CMP	R1, R0
LOAD:00001510		BEQ	loc_1524
LOAD:00001514		LDR	R0, [R11,#var_C]
LOAD:00001518		SUB	R0, R0, #0x1000
LOAD:0000151C		STR	R0, [R11,#var_C]
LOAD:00001520		B	loc_1500
LOAD:00001524 ;	-----		

```

1 void __noreturn sub_14D4()
2 {
3     unsigned int v0; // ST44_4
4     unsigned int v1; // ST40_4
5     unsigned __int64 v2; // kr00_8
6     char v3; // ST27_1
7     int j; // [sp+14h] [bp-44h]
8     int v5; // [sp+18h] [bp-40h]
9     int v6; // [sp+1Ch] [bp-3Ch]
10    char v7; // [sp+2Eh] [bp-2Ah]
11    char v8; // [sp+2Fh] [bp-29h]
12    char v9; // [sp+30h] [bp-28h]
13    char v10; // [sp+33h] [bp-25h]
14    unsigned int v11; // [sp+34h] [bp-24h]
15    _DWORD *v12; // [sp+38h] [bp-20h]
16    unsigned int v13; // [sp+48h] [bp-10h]
17    unsigned int i; // [sp+4Ch] [bp-Ch]
18
19    for ( i = (unsigned int)sub_14D4 & 0xFFFFF000; *(_DWORD *)i != 1179403647; i -= 4096 )
20    ;
21    v13 = 0;
22    v12 = (_DWORD *)(i + *(_DWORD *)(i + 28));
23    v11 = 0;
24    while ( 1 )
25    {
26        if ( v11 >= *(unsigned __int16 *)(i + 44) )
27            goto LABEL_13;
28        if ( *v12 != 1 || v12[6] != 5 )
29        {
30            if ( *v12 == 1 && v12[6] == 6 )

```

000014D4 sub_14D4:1 (14D4)

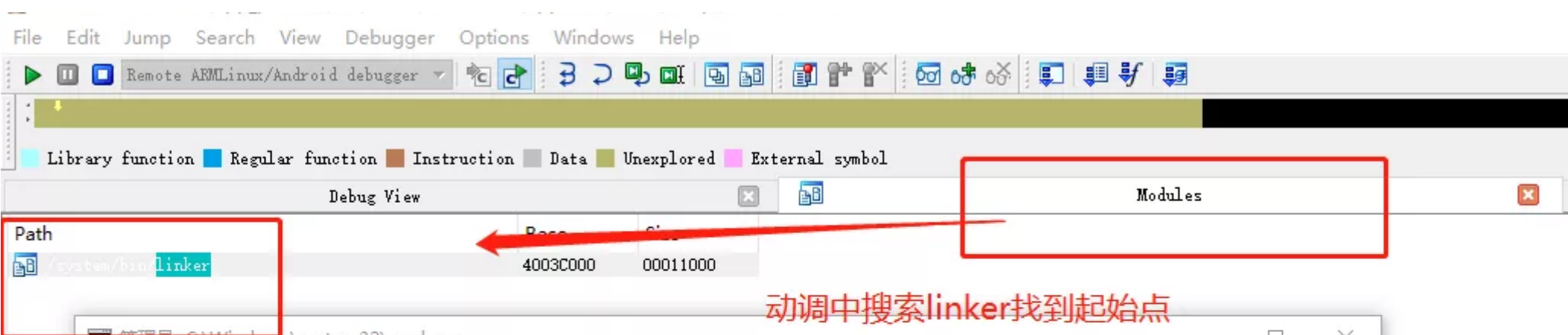
2、启动IDA动态调试

adbshell su /data/local/tmp/as -p31928adb forwardtcp:31928 tcp:31928adb shell am start -D -

ncom.qianyu.app/.LoginActivityadb forward tcp:8700jdpw:15127jdb -

connectcom.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8652如果出现错误查看serverSocket所监听的端口netstat-nao关键地址:基址+偏移地址





```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.16299.1565]
(c) 2017 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>adb forward tcp:31928 tcp:31928
```

```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.16299.1565]
(c) 2017 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>adb shell am start -D -n com.example.helloworld/.MainActivity
WARNING: linker: app_process has text relocations. This is wasting memory and is a security risk. Please fix.
WARNING: linker: app_process has text relocations. This is wasting memory and is a security risk. Please fix.
Starting: Intent { cmp=com.example.helloworld/.MainActivity }
```

```
管理员: C:\Windows\system32\cmd.exe - adb shell
Microsoft Windows [版本 10.0.16299.1565]
(c) 2017 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>adb install C:\Users\Administrator\Desktop\helloworld.apk
adb server is out of date. killing...
* daemon started successfully *
3322 KB/s (661130 bytes in 0.194s)
WARNING: linker: app_process has text relocations. This is wasting memory and is a security risk. Please fix.
WARNING: linker: app_process has text relocations. This is wasting memory and is a security risk. Please fix.
    pkg: /data/local/tmp/helloworld.apk
Success

C:\Users\Administrator>adb shell
shell@hammerhead:/ $ su
su
root@hammerhead:/ # ./data/local/tmp/as -p31928
./data/local/tmp/as -p31928
IDA Android 32-bit remote debug server(ST) v1.22. Hex-Rays (c) 2004-2017
Listening on 0.0.0.0:31928...
```



这时候将基址加上偏移地址

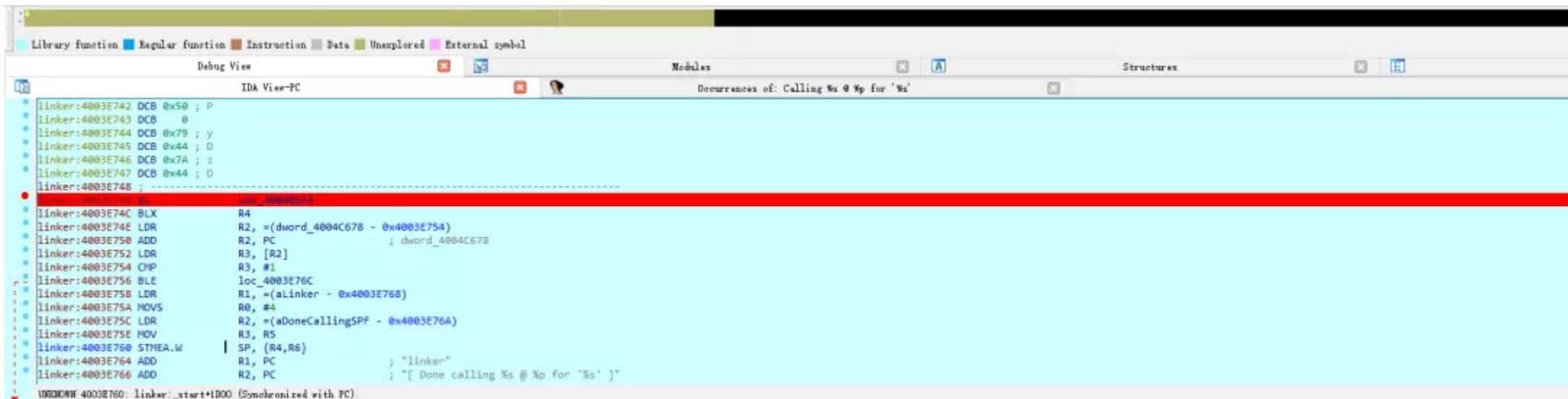
即base的值加上2748（查看文章偏移量计算）

$4003C000 + 2748 = 4003E748$

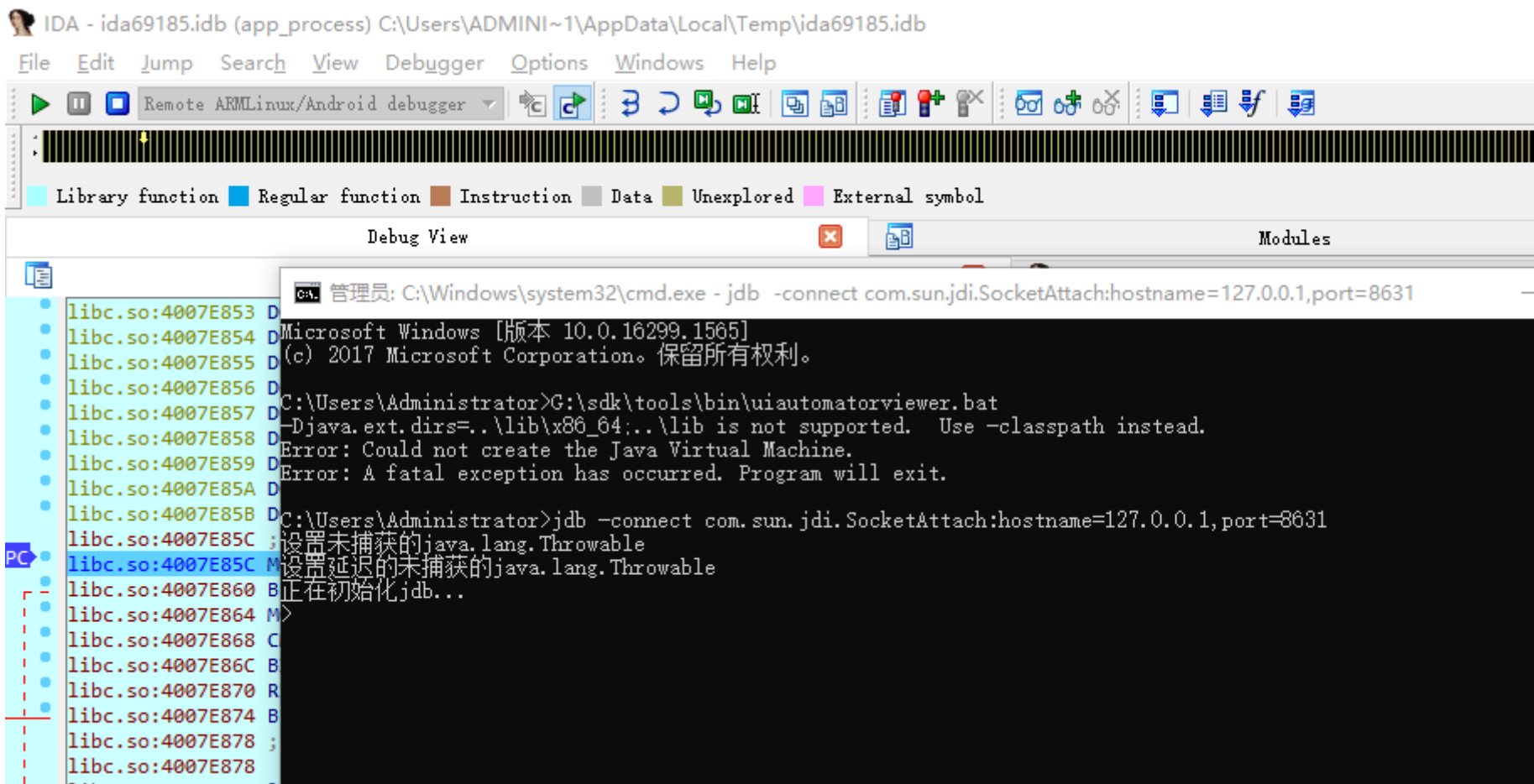
```
linker:4003E747 DCB 0x44 ; D
linker:4003E748 ; -----
linker:4003E748 BL      unk_400405F4
linker:4003E74C BLX     R4
linker:4003E74E LDR     R2, =(dword_4004C678 - 0x4003E754)
linker:4003E750 ADD     R2, PC      ; dword_4004C678
linker:4003E752 LDR     R3, [R2]
linker:4003E754 CMP     R3, #1
linker:4003E756 BLE     loc_4003E76C
linker:4003E758 LDR     R1, =(aLinker - 0x4003E768)
linker:4003E75A MOVS    R0, #4
linker:4003E75C LDR     R2, =(aDoneCallingSPF - 0x4003E76A)
linker:4003E75E MOV     R3, R5
linker:4003E760 STMEA.W SP, {R4,R6}
linker:4003E764 ADD     R1, PC      ; "linker"
linker:4003E766 ADD     R2, PC      ; "[ Done calling %s @ %p for '%s' ]"
linker:4003E768 BL      unk_400405F4
linker:4003E76C          ; CODE XREF: linker:_start+1CF6↑j
linker:4003E76C MOVS    R0, #3
```

找到位置即可用快捷键C

3、下断点



启动:



运行后在节点处停下进行下一步分析

4、流程式查看

F9到运行在BL unk_400405F4 使用F7进去看到第一个初始化函数；

这里需要注意，可以看到前面是libshella.so而不是其它的so文件，其中几次调试会出现libBugly.so的so文件

```
libBugly.so:77FA0660 DCB 0
libBugly.so:77FA0660 ; -----
libBugly.so:77FA0660 CODE16
libBugly.so:77FA0660 PUSH {R4,LR}
libBugly.so:77FA0662 LDR R4, =(unk_77FB30B4 - 0x77FA066A)
libBugly.so:77FA0664 MOVS R1, #0
libBugly.so:77FA0666 ADD R4, PC ; unk_77FB30B4
libBugly.so:77FA0668 MOVS R0, R4
libBugly.so:77FA066A BL unk_77FAC170
libBugly.so:77FA066E LDR R1, =(loc_77FA7920+1 - 0x77FA0678)
libBugly.so:77FA0670 LDR R2, =(unk_77FB3000 - 0x77FA067A)
libBugly.so:77FA0672 MOVS R0, R4
libBugly.so:77FA0674 ADD R1, PC ; loc_77FA7920
libBugly.so:77FA0676 ADD R2, PC ; unk_77FB3000
libBugly.so:77FA0678 BL unk_77FAC1E0
libBugly.so:77FA067C POP {R4,PC}
libBugly.so:77FA067C ; -----
libBugly.so:77FA067E DCB 0xC0
```

进入libshella.so前:

IDA - ida40934.idb (app_process) C:\Users\ADMINI~1\AppData\Local\Temp\ida40934.idb

File Edit Jump Search View Debugger Options Windows Help

Remote ARMLinux/Android debugger

Library function Regular function Instruction Data Unexplored External symbol

Debug View Modules

IDA View-PC

```
libshella_2.8.1.so:77BC28DD DCB 0x8A
libshella_2.8.1.so:77BC28DE DCB 0
libshella_2.8.1.so:77BC28DF DCB 0
libshella_2.8.1.so:77BC28E0 ; -----
libshella_2.8.1.so:77BC28E0 BX LR
libshella_2.8.1.so:77BC28E0 ; -----
libshella_2.8.1.so:77BC28E4 DCB 0
libshella_2.8.1.so:77BC28E5 DCB 0x30 ; 0
libshella_2.8.1.so:77BC28E6 DCB 0x90
libshella_2.8.1.so:77BC28E7 DCB 0xE5
libshella_2.8.1.so:77BC28E8 DCB 1
libshella_2.8.1.so:77BC28E9 DCB 1
```

由于没有逻辑处理，紧接着F8回来，再一次F9再一次F7进去：

这里点击P键恢复原函数

可以看到第二个初始化函数：

IDA - ida40934.idb (app_process) C:\Users\ADMINI~1\AppData\Local\Temp\ida40934.idb

File Edit Jump Search View Debugger Options Windows Help

Remote ARMLinux/Android debugger

Library function Regular function Instruction Data Unexplored External symbol

Debug View Modules

IDA View-PC

R2
R3
R4
PC

libshella_2.8.1.so:77BC24D1 DCB 0x88

libshella_2.8.1.so:77BC24D2 DCB 0

libshella_2.8.1.so:77BC24D3 DCB 0

libshella_2.8.1.so:77BC24D4 ; -----

libshella_2.8.1.so:77BC24D4 STMFD SP!, {R4,R10,R11,LR}

libshella_2.8.1.so:77BC24D8 ADD R11, SP, #8

libshella_2.8.1.so:77BC24DC SUB SP, SP, #0x50

libshella_2.8.1.so:77BC24E0 LDR R0, =(unk_77BCAF44 - 0x77BC24EC)

libshella_2.8.1.so:77BC24E4 ADD R0, PC, R0 ; unk_77BCAF44

libshella_2.8.1.so:77BC24E8 LDR R1, =0xFFFFF7590

libshella_2.8.1.so:77BC24EC ADD R1, R1, R0

libshella_2.8.1.so:77BC24F0 LDR R2, =0xFFFFF000

libshella_2.8.1.so:77BC24F4 AND R1, R1, R2

libshella_2.8.1.so:77BC24F8 STR R1, [R11,#-0xC]

libshella_2.8.1.so:77BC24FC STR R0, [SP,#0x10]

libshella_2.8.1.so:77BC2500

libshella_2.8.1.so:77BC2500 loc_77BC2500 ; CODE XREF: libshella_2.8.1.so:77BC2520↓j

libshella_2.8.1.so:77BC2500 LDR R0, =unk_464C457F

libshella_2.8.1.so:77BC2504 LDR R1, [R11,#-0xC]

UNKNOWN 77BC24D4: libshella_2.8.1.so:77BC24D4 (Synchronized with PC, Hex View-1)

Hex View-1

77BC2470 4A FF FF EB 44 10 9F E5 44 30 9F E5 44 20 9F E5 J.....

77BC2480 14 C0 1B E5 0C 20 92 E7 0C E0 1B E5 0C 40 1B E5

77BC2490 10 50 1B E5 05 40 84 E0 0C 00 8D E5 01 00 A0 E1 .P.....

77BC24A0 0E 10 A0 E1 08 20 8D E5 04 20 A0 E1 08 E0 9D E5

77BC24B0 3E FF 2F E1 04 00 8D E5 08 D0 4B E2 30 88 BD E8 >./.....K....

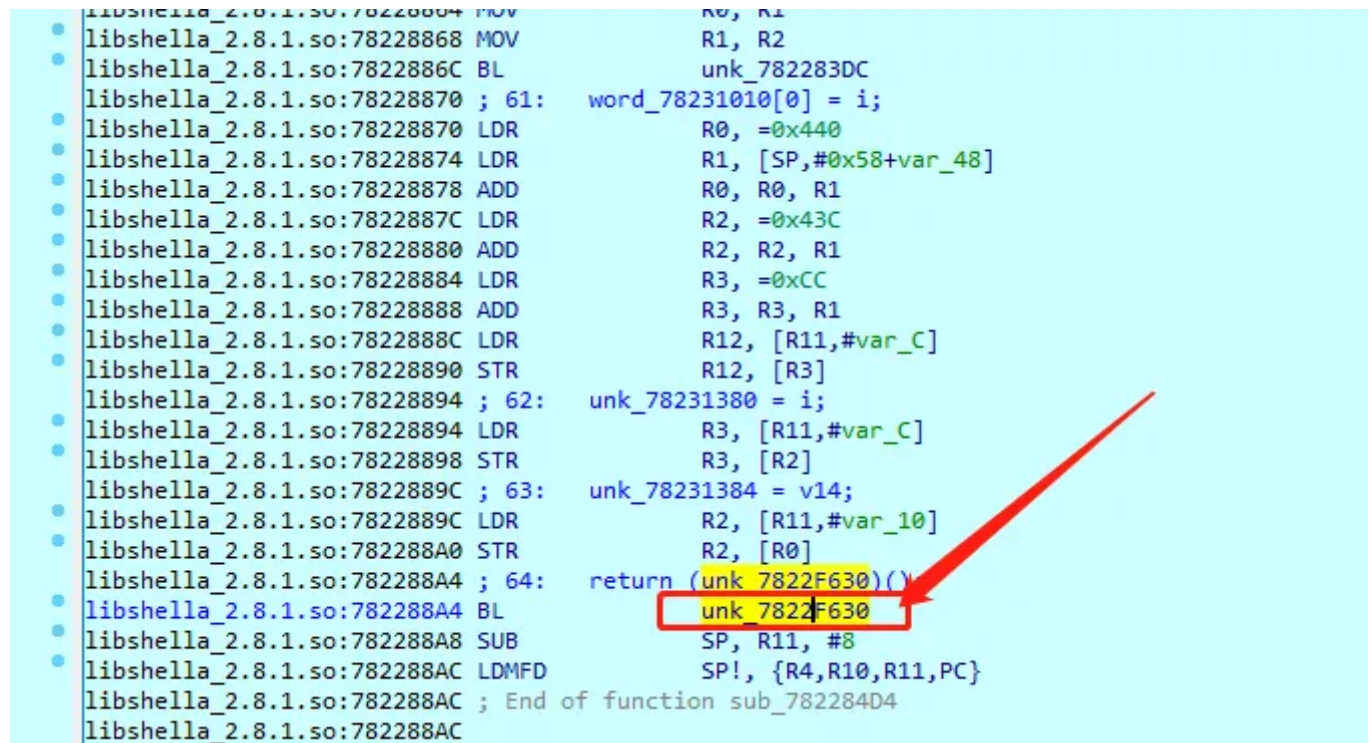
77BC24C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

同步SP寄存器查看就可以看到程序执行流程了

在这里可以跟随函数不断往下走，也可以试试脚本

```
static main(void){ do {      step_over();      wait_for_next_event(STEP,-1); }while(PC!=0xEEDAF630);//停在何处}
```

单步跟踪法查询到查询执行死亡的位置，然后定位



```
libshella_2.8.1.so:78228864 MOV     R0, R1
libshella_2.8.1.so:78228868 MOV     R1, R2
libshella_2.8.1.so:7822886C BL      unk_782283DC
libshella_2.8.1.so:78228870 ; 61:   word_78231010[0] = i;
libshella_2.8.1.so:78228870 LDR      R0, =0x440
libshella_2.8.1.so:78228874 LDR      R1, [SP,#0x58+var_48]
libshella_2.8.1.so:78228878 ADD     R0, R0, R1
libshella_2.8.1.so:7822887C LDR      R2, =0x43C
libshella_2.8.1.so:78228880 ADD     R2, R2, R1
libshella_2.8.1.so:78228884 LDR      R3, =0xCC
libshella_2.8.1.so:78228888 ADD     R3, R3, R1
libshella_2.8.1.so:7822888C LDR      R12, [R11,#var_C]
libshella_2.8.1.so:78228890 STR      R12, [R3]
libshella_2.8.1.so:78228894 ; 62:   unk_78231380 = i;
libshella_2.8.1.so:78228894 LDR      R3, [R11,#var_C]
libshella_2.8.1.so:78228898 STR      R3, [R2]
libshella_2.8.1.so:7822889C ; 63:   unk_78231384 = v14;
libshella_2.8.1.so:7822889C LDR      R2, [R11,#var_10]
libshella_2.8.1.so:782288A0 STR      R2, [R0]
libshella_2.8.1.so:782288A4 ; 64:   return (unk_7822F630)(Y
libshella_2.8.1.so:782288A4 BL      unk_7822F630
libshella_2.8.1.so:782288A8 SUB     SP, R11, #8
libshella_2.8.1.so:782288AC LDMFD   SP!, {R4,R10,R11,PC}
libshella_2.8.1.so:782288AC ; End of function sub_782284D4
libshella_2.8.1.so:782288AC
```

进入后，点击P分析代码

```
libshella_2.8.1.so:7822F62E DCB 0xFF
libshella_2.8.1.so:7822F62F DCB 0x9A
libshella_2.8.1.so:7822F630 unk_7822F630 DCB 0x71 ; q ; CODE XREF: sub_782284D4+3D0↑p
libshella_2.8.1.so:7822F631 DCB 0xF4
libshella_2.8.1.so:7822F632 DCB 0xF0
libshella_2.8.1.so:7822F633 DCB 0x8F
libshella_2.8.1.so:7822F634 DCB 0x9C
libshella_2.8.1.so:7822F635 DCB 0x9E
libshella_2.8.1.so:7822F636 DCB 0xFD
libshella_2.8.1.so:7822F637 DCB 3
libshella_2.8.1.so:7822F638 DCB 0x83
libshella_2.8.1.so:7822F639 DCB 0xF6
libshella_2.8.1.so:7822F63A DCB 0x16
libshella_2.8.1.so:7822F63B DCB 0x14
libshella_2.8.1.so:7822F63C DCB 5
libshella_2.8.1.so:7822F63D DCB 0x46 ; F
libshella_2.8.1.so:7822F63E DCB 4
libshella_2.8.1.so:7822F63F DCB 0x68 ; k
libshella_2.8.1.so:7822F640 DCB 0xD8
libshella_2.8.1.so:7822F641 DCB 0xE5
libshella_2.8.1.so:7822F642 DCB 0xBE
libshella_2.8.1.so:7822F643 DCB 0x8E
libshella_2.8.1.so:7822F644 DCB 0x9D
libshella_2.8.1.so:7822F645 DCB 0xE2
libshella_2.8.1.so:7822F646 DCB 0xDE
libshella_2.8.1.so:7822F647 DCB 0x4A ; j
libshella_2.8.1.so:7822F648 DCB 0xA1
```

```

libshella_2.8.1.so:7822F630 sub_7822F630 ; CODE XREF: sub_782284D4+3
libshella_2.8.1.so:7822F630 STMFD SP!, {R11,LR}
libshella_2.8.1.so:7822F634 MOV R11, SP
libshella_2.8.1.so:7822F638 SUB SP, SP, #0x10
libshella_2.8.1.so:7822F63C LDR R0, =(unk_78230F44 - 0x7822F648)
libshella_2.8.1.so:7822F640 ADD R0, PC, R0
libshella_2.8.1.so:7822F644 SUB R1, R11, #4
libshella_2.8.1.so:7822F648 LDR R2, =0
libshella_2.8.1.so:7822F64C LDR R3, =0xFFFFDC8C
libshella_2.8.1.so:7822F650 ADD R3, R3, R0
libshella_2.8.1.so:7822F654 LDR R12, =0x43C
libshella_2.8.1.so:7822F658 ADD R12, R12, R0
libshella_2.8.1.so:7822F65C LDR LR, =0x434
libshella_2.8.1.so:7822F660 ADD LR, LR, R0
libshella_2.8.1.so:7822F664 STR R2, [R11,#-4]
libshella_2.8.1.so:7822F668 STR R2, [LR]
libshella_2.8.1.so:7822F66C LDR R12, [R12]
libshella_2.8.1.so:7822F670 STR R0, [SP,#8]
libshella_2.8.1.so:7822F674 MOV R0, R1
libshella_2.8.1.so:7822F678 MOV R1, R2
libshella_2.8.1.so:7822F67C MOV R2, R3
libshella_2.8.1.so:7822F680 MOV R3, R12
libshella_2.8.1.so:7822F684 BL unk_7822835C
libshella_2.8.1.so:7822F688 LDR R1, =1
libshella_2.8.1.so:7822F688 ; End of function sub_7822F630
libshella_2.8.1.so:7822F688
libshella_2.8.1.so:7822F68C LDR R2, =0x438
libshella_2.8.1.so:7822F690 LDR R3, [SP,#8]
libshella_2.8.1.so:7822F694 ADD R2, R2, R3
libshella_2.8.1.so:7822F698 STR R1, [R2]
libshella_2.8.1.so:7822F69C STR R0, [SP,#4]
libshella_2.8.1.so:7822F6A0 MOV SP, R11
libshella_2.8.1.so:7822F6A4 LDMFD SP!, {R11,PC}
libshella_2.8.1.so:7822F6A4 ; -----

```

5、nop掉反调试

定位寻找函数create()相关的创建线程的反调试函数；

而定位后需要对其进行nop，即将16进制的hex数据改为**0000 A0 E1**

每一个create函数都会被隐写，因此都需要c键去分析一下，如下unk_7822835C

进去后，按下C键，从而确定为create函数，然后esc键返回。

```
libshella_2.8.1.so:7822F630 ADD R12, R12, R0
libshella_2.8.1.so:7822F65C LDR LR, =0x434
libshella_2.8.1.so:7822F660 ADD LR, LR, R0
libshella_2.8.1.so:7822F664 STR R2, [R11, #-4]
libshella_2.8.1.so:7822F668 STR R2, [LR]
libshella_2.8.1.so:7822F66C LDR R12, [R12]
libshella_2.8.1.so:7822F670 STR R0, [SP, #0x18+var_10]
libshella_2.8.1.so:7822F674 MOV R0, R1
libshella_2.8.1.so:7822F678 MOV R1, R2
libshella_2.8.1.so:7822F67C MOV R2, R3
libshella_2.8.1.so:7822F680 MOV R3, R12
libshella_2.8.1.so:7822F684 BL unk_7822835C
libshella_2.8.1.so:7822F688 LDR R1, =1
libshella_2.8.1.so:7822F688 ; End of function sub_7822F630
libshella_2.8.1.so:7822F688
libshella_2.8.1.so:7822F68C loc_7822F68C
libshella_2.8.1.so:7822F68C LDR R2, =0x438
libshella_2.8.1.so:7822F690 LDR R3, [SP, #8]
libshella_2.8.1.so:7822F694 ADD R2, R2, R3
libshella_2.8.1.so:7822F698 STR R1, [R2]
libshella_2.8.1.so:7822F69C STR R0, [SP, #4]
libshella_2.8.1.so:7822F6A0 MOV SP, R11
libshella_2.8.1.so:7822F6A4 LDMFD SP!, {R11, PC}
libshella_2.8.1.so:7822F6A4 ; -----
libshella_2.8.1.so:7822F6A8 dword_7822F6A8 DCD 1 ; DATA XREF: sub_7822F630+58↑r

libshella_2.8.1.so:78228359 DCB 0xFC
libshella_2.8.1.so:7822835A DCB 0xBC
libshella_2.8.1.so:7822835B DCB 0xE5
libshella_2.8.1.so:7822835C ; [0000000C BYTES: COLLAPSED FUNCTION create. PRESS CTRL-NUMPAD+ TO EXPAND]
libshella_2.8.1.so:78228368 DCB 0
libshella_2.8.1.so:78228369 DCB 0xC6
libshella_2.8.1.so:7822836A DCB 0x8F
libshella_2.8.1.so:7822836B DCB 0xE2
libshella_2.8.1.so:7822836C DCB 8
libshella_2.8.1.so:7822836D DCB 0xCA
```

```

libshella_2.8.1.so:7822F644 SUB      R1, R11, #4
libshella_2.8.1.so:7822F648 LDR      R2, =0
libshella_2.8.1.so:7822F64C LDR      R3, =0xFFFFDC8C
libshella_2.8.1.so:7822F650 ADD      R3, R3, R0
libshella_2.8.1.so:7822F654 LDR      R12, =0x43C
libshella_2.8.1.so:7822F658 ADD      R12, R12, R0
libshella_2.8.1.so:7822F65C LDR      LR, =0x434
libshella_2.8.1.so:7822F660 ADD      LR, LR, R0
libshella_2.8.1.so:7822F664 STR      R2, [R11,#-4]
libshella_2.8.1.so:7822F668 STR      R2, [LR]
libshella_2.8.1.so:7822F66C LDR      R12, [R12]
libshella_2.8.1.so:7822F670 STR      R0, [SP,#0x18+var_10]
libshella_2.8.1.so:7822F674 MOV      R0, R1
libshella_2.8.1.so:7822F678 MOV      R1, R2
libshella_2.8.1.so:7822F67C MOV      R2, R3
libshella_2.8.1.so:7822F680 MOV      R3, R12
libshella_2.8.1.so:7822F684 BL       create
libshella_2.8.1.so:7822F688 LDR      R1, =1
libshella_2.8.1.so:7822F688 ; End of function sub_7822F630
libshella_2.8.1.so:7822F688
libshella_2.8.1.so:7822F68C
libshella_2.8.1.so:7822F68C loc_7822F68C

```

然后定位create改为nop函数

```

libshella_2.8.1.so:7822F658 ADD      R12, R12, R0
libshella_2.8.1.so:7822F65C LDR      LR, =0x434
libshella_2.8.1.so:7822F660 ADD      LR, LR, R0
libshella_2.8.1.so:7822F664 STR      R2, [R11,#-4]
libshella_2.8.1.so:7822F668 STR      R2, [LR]
libshella_2.8.1.so:7822F66C LDR      R12, [R12]
libshella_2.8.1.so:7822F670 STR      R0, [SP,#0x18+var_10]
libshella_2.8.1.so:7822F674 MOV      R0, R1
libshella_2.8.1.so:7822F678 MOV      R1, R2
libshella_2.8.1.so:7822F67C MOV      R2, R3
libshella_2.8.1.so:7822F680 MOV      R3, R12
libshella_2.8.1.so:7822F684 BL       create
libshella_2.8.1.so:7822F688 LDR      R1, =1
libshella_2.8.1.so:7822F688 ; End of function sub_7822F630
libshella_2.8.1.so:7822F688
libshella_2.8.1.so:7822F68C
libshella_2.8.1.so:7822F68C loc_7822F68C

```

UNKNOWN 7822F688: sub_7822F630+58 (Synchronized with PC, Hex View-1)

Hex View-1

7822F680	0C 30 A0 E1	00 00 A0 E1	18 10 9F E5	18 20 9F E5	.0.....
7822F690	08 30 9D E5	03 20 82 E0	00 10 82 E5	04 00 8D E5	.0.....
7822F6A0	0B D0 A0 E1	00 88 BD E8	01 00 00 00	38 04 00 00	.P.....8..
7822F6B0	00 00 00 00	8C DC FF FF	3C 04 00 00	34 04 00 00	<...4..
7822F6C0	FC 18 00 00	18 8D FF 7F	01 00 00 00	14 92 FF 7F	
7822F6D0	B0 B0 B0 80	28 92 FF 7F	AF 07 B1 80	CC 92 FF 7F	(.....;
7822F6E0	B0 B0 B0 80	14 93 FF 7F	A9 07 B1 80	1C 94 FF 7F	
7822F6F0	B0 B0 A8 80	80 94 FF 7F	B0 B0 B0 80	98 94 FF 7F	
7822F700	B0 B0 AA 80	F4 94 FF 7F	AF 72 B2 80	10 96 FF 7F	r.....
7822F710	B0 B0 B0 80	10 96 FF 7F	AB 3F 38 80	AC 96 FF 7F	?8.....
7822F720	B0 B0 B0 80	C0 96 FF 7F	B0 B0 AA 80	2C 97 FF 7F	,
7822F730	B0 B0 B0 80	48 97 FF 7F	9C 00 00 00	60 97 FF 7F	H.....`...
7822F740	B0 B0 B0 80	B4 97 FF 7F	A8 0F B1 80	D4 97 FF 7F	p...

F2键进行修改后
F2键保存

```
libshella_2.8.1.so:7822F668 STR R2, [LR]
libshella_2.8.1.so:7822F66C LDR R12, [R12]
libshella_2.8.1.so:7822F670 STR R0, [SP,#0x18+var_10]
libshella_2.8.1.so:7822F674 MOV R0, R1
libshella_2.8.1.so:7822F678 MOV R1, R2
libshella_2.8.1.so:7822F67C MOV R2, R3
libshella_2.8.1.so:7822F680 MOV R3, R12
libshella_2.8.1.so:7822F684 NOP
libshella_2.8.1.so:7822F688 LDR R1, =1
libshella_2.8.1.so:7822F688 ; end of function sub_7822F630
libshella_2.8.1.so:7822F688
libshella_2.8.1.so:7822F68C
libshella_2.8.1.so:7822F68C loc_7822F68C

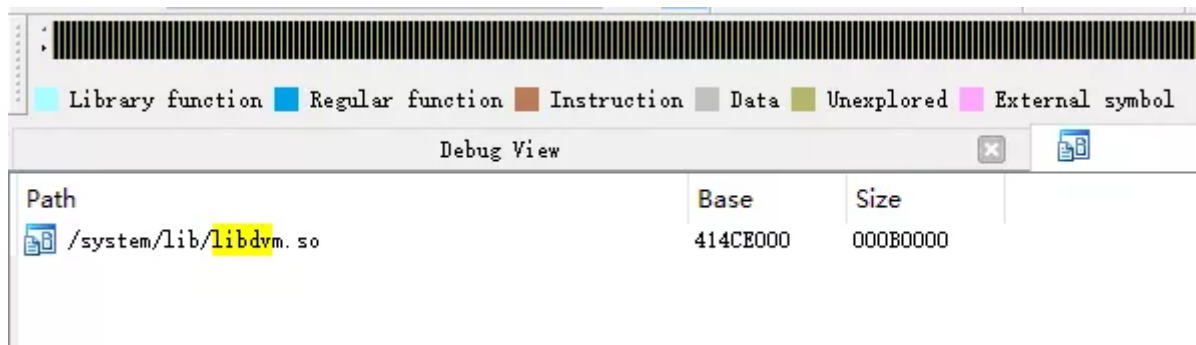
UNKNOWN 7822F688: sub_7822F630+58 (Synchronized with PC, Hex View-1)
```

Hex View-1

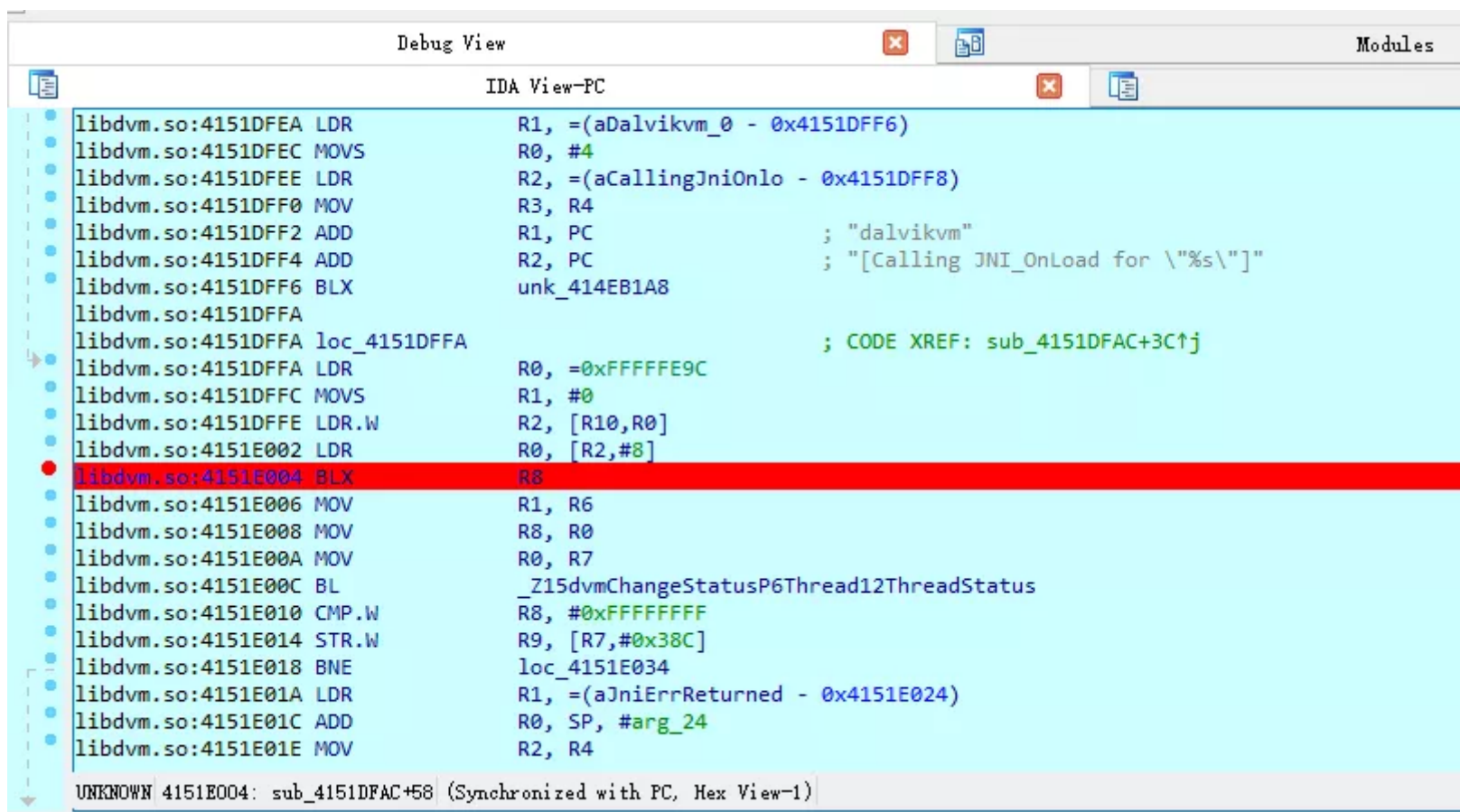
7822F680	0C 30 A0 E1	00 00 A0 E1	18 10 9F E5	18 20 9F E5	.0.....
7822F690	08 30 9D E5	03 20 82 E0	00 10 82 E5	04 00 8D E5	.0.....
7822F6A0	0B D0 A0 E1	00 88 BD E8	01 00 00 00	38 04 00 00	.P.....8..
7822F6B0	00 00 00 00	8C DC FF FF	3C 04 00 00	34 04 00 00	<...4..
7822F6C0	FC 18 00 00	18 8D FF 7F	01 00 00 00	14 92 FF 7F	
7822F6D0	B0 B0 B0 80	28 92 FF 7F	AF 07 B1 80	CC 92 FF 7F	(.....:

然后根据libdvm.so计算偏移量：

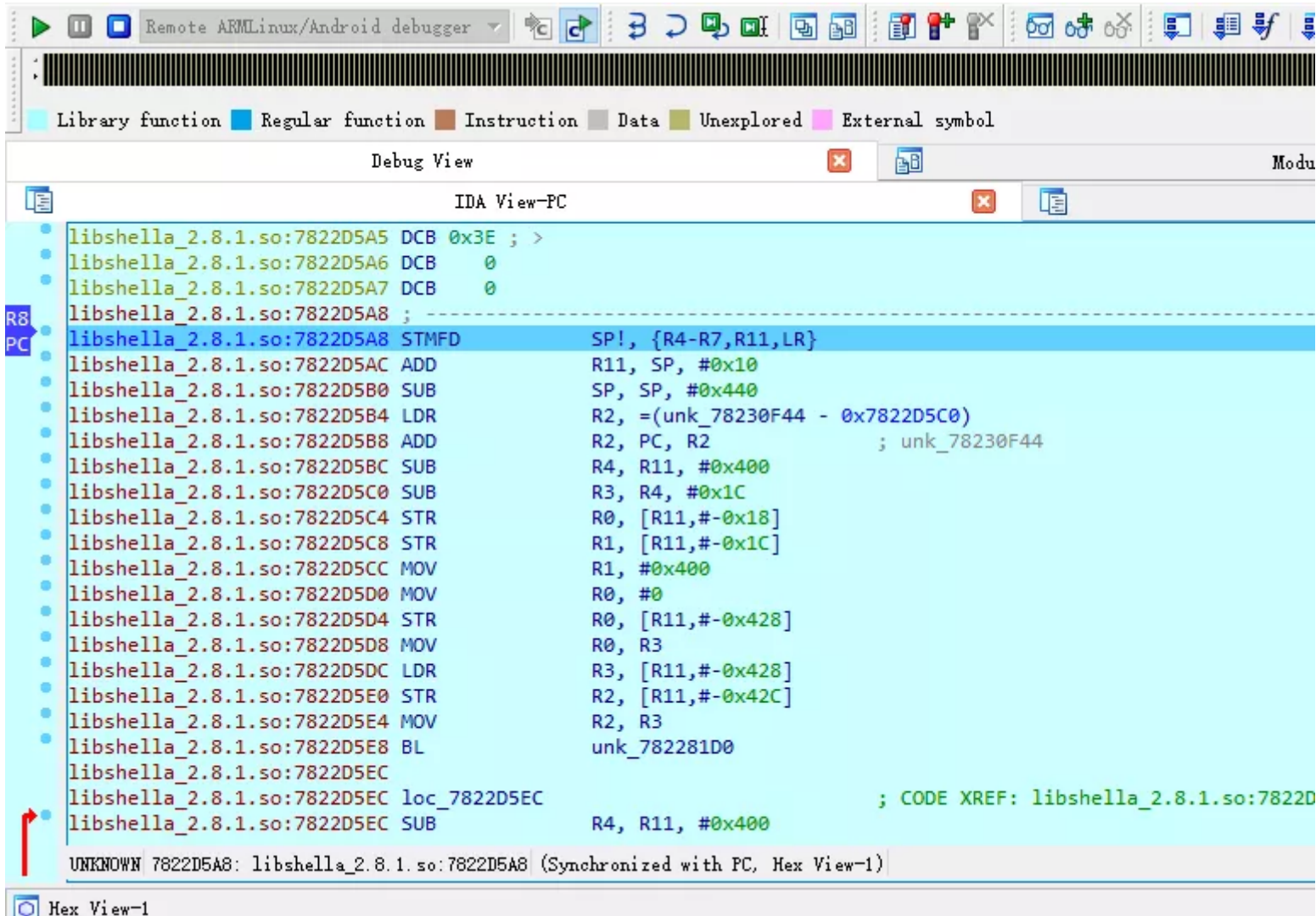
414CE000+手机本机偏移量计算得偏移量得值=4151DFAC（查阅文章arm手机偏移量）



定位jni_onload()的调用文件点：



运行到此处后F7进入:

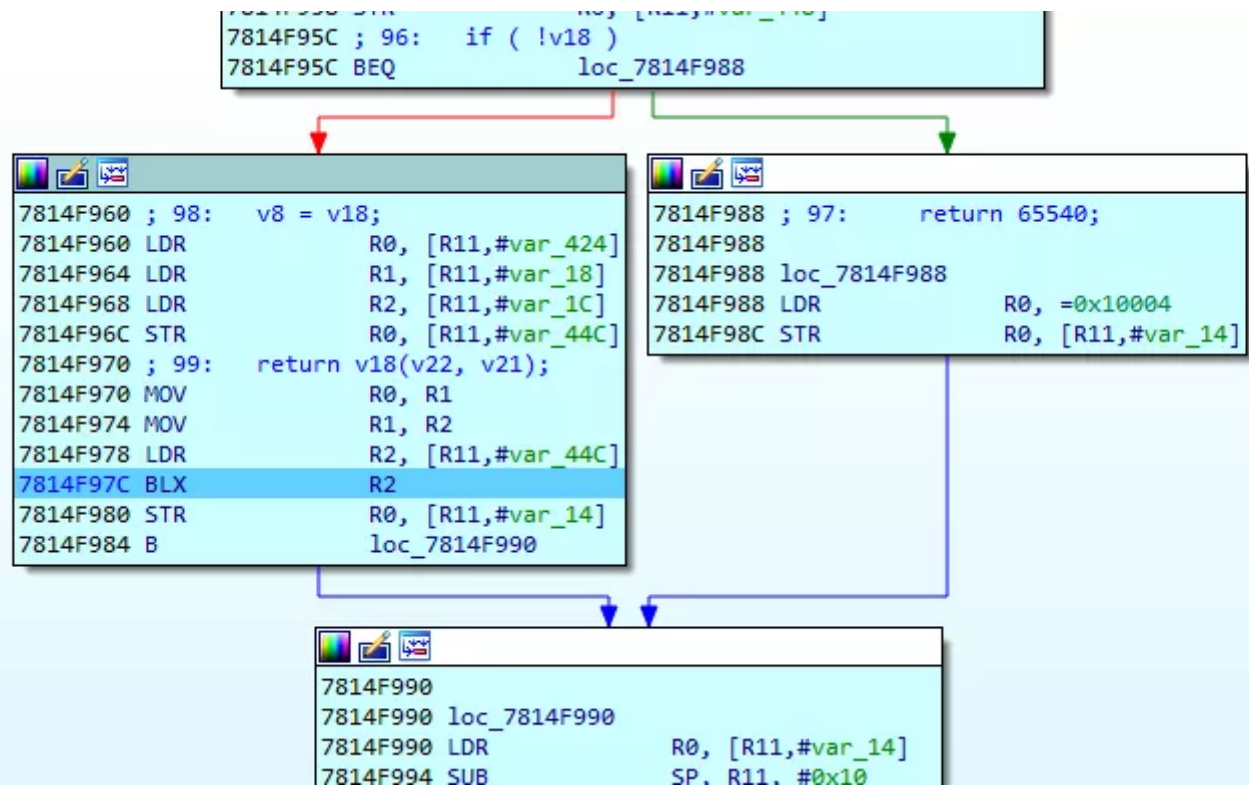


P一下后F5查看伪代码查询逻辑，发现调用dlopen打开动态链接库和dlsym返回符号地址

因此这个函数不仅仅是获取函数地址，还可以获取变量地址

在反汇编窗口，运行完程序后找到一个关键跳转点：

在这里return返回前进行了一个逻辑处理：



选择F7跟进处理查看，这里可见需要关注的大多为寄存器的存储值，除此之外，还需要每个函数跟进分析：

```
IDA VIEW 1C
1 signed int __fastcall sub_78005800(int a1)
2 {
3     signed int v1; // r4
4     int v2; // r6
5     int v4; // [sp+4h] [bp-14h]
6
7     v1 = 0;
8     v2 = a1;
9     v4 = 0;
10    if ( (unk_78005774)() )
11    {
12        if ( (unk_78005774)(v2, &v4, 65540) )
13        {
14            if ( (unk_78005774)(v2, &v4, 65538) )
15            {
16                if ( (unk_78005774)(v2, &v4, 65537) )
17                {
18                    return v1;
19                }
20                v1 = 65537;
21            }
22            else
23            {
24                v1 = 65538;
25            }
26        }
27    }
28    else
29    {
30    }
```

在判断以后执行最后一个:

```
17     return v1;
18     v1 = 65537;
19 }
20 else
21 {
22     v1 = 65538;
23 }
24 }
25 else
26 {
27     v1 = 65540;
28 }
29 }
30 else
31 {
32     v1 = 65542;
33 }
34 if ( v4 )
35 {
36     (unk_78005FB8)();
37     (unk_780057C4)(v4);
38 }
39 return v1;
40 }
```



而进去后为：

找到了Android的大多数信息：

```
1 unsigned int sub_78005FB8()
2 {
3     unsigned int result; // r0
4
5     dword_7801D068 = "java/lang/String";
6     dword_7801D06C = "getBytes";
7     dword_7801D070 = "() [B";
8     dword_7801D074 = "android/os/Build$VERSION";
9     dword_7801D078 = "SDK_INT";
10    dword_7801D07C = &unk_78018267;
11    dword_7801D08C = "android/content/ContextWrapper";
12    dword_7801D080 = "android/app/ActivityThread";
13    dword_7801D084 = "currentActivityThread";
14    dword_7801D090 = "getPackageName";
15    dword_7801D088 = "() Landroid/app/ActivityThread;";
16    dword_7801D0D0 = "Ljava/lang/ClassLoader;";
17    dword_7801D094 = "() Ljava/lang/String;";
18    dword_7801D0D8 = "java/lang/ClassLoader";
19    dword_7801D098 = "mPackages";
20    dword_7801D09C = "Ljava/util/HashMap;";
21    dword_7801D0A0 = "Landroid/util/ArrayMap;";
22    dword_7801D0A4 = "java/util/HashMap";
23    dword_7801D0A8 = "android/util/ArrayMap";
24    dword_7801D0AC = "(Ljava/lang/Object;) Ljava/lang/Object;";
25    dword_7801D0B0 = "() Ljava/lang/Object;";
26    dword_7801D0B4 = &unk_78018381;
```

却不是执行程序，然后就看第二个函数：

```

79 dword_7801D180 = "mInitialApplication";
80 dword_7801D194 = "remove";
81 dword_7801D1BC = "mInitialApplication";
82 dword_7801D198 = "(Ljava/lang/Object;)Z";
83 dword_7801D1C0 = "mProviderMap";
84 dword_7801D19C = "java/util/ArrayList";
85 dword_7801D1C4 = "values";
86 dword_7801D1A0 = "mApplicationInfo";
87 dword_7801D1C8 = "()Ljava/util/Collection;";
88 dword_7801D1A4 = "Landroid/content/pm/ApplicationInfo;";
89 dword_7801D1CC = "java/util/Collection";
90 dword_7801D1A8 = "className";
91 dword_7801D1D0 = "iterator";
92 dword_7801D1AC = "appInfo";
93 dword_7801D1D4 = "()Ljava/util/Iterator;";
94 dword_7801D1B0 = "mApplication";
95 dword_7801D1D8 = "java/util/Iterator";
96 dword_7801D1B4 = "makeApplication";
97 dword_7801D1DC = "hasNext";
98 dword_7801D1B8 = "(ZLandroid/app/Instrumentation;)Landroid/app/Application;";
99 dword_7801D1E0 = &unk_780189AE;
100 dword_7801D1E4 = "next";
101 dword_7801D1E8 = "android/app/ActivityThread$ProviderClientRecord";
102 dword_7801D1EC = "android/app/ActivityThread$ProviderRecord";
103 dword_7801D1F0 = "mLocalProvider";
104 dword_7801D1F4 = "Landroid/content/ContentProvider;";

```

UNKNOWN sub_78005FB8:79 (780061F4)

Hex View-1

```

78005FA0 D7 1C 01 00 88 22 01 00 B9 1C 01 00 97 22 01 00 ....."......"..
78005FB0 00 1C 01 00 88 22 01 00 B9 1C 01 00 97 22 01 00 ....."......"..

```

第二个函数传入v4，而v4为：

```
19     }
20     else
21     {
22         v1 = 65538;
23     }
24 }
25 else
26 {
27     v1 = 65540;
28 }
29 }
30 else
31 {
32     v1 = 65542;
33 }
34 if ( v4 )
35 {
36     (unk_78005FB8)(v4);
37     (unk_780057C4)(v4);
38 }
39 return v1;
```

[stack]:BED8E8FF DCB 0x78 ; y
[stack]:BED8E900 DCB 0x6C ; 1
[stack]:BED8E901 DCB 0x6F ; o
[stack]:BED8E902 DCB 0x61 ; a
[stack]:BED8E903 DCB 0x64 ; d
[stack]:BED8E904 DCB 0x20 ;
[stack]:BED8E905 DCB 0x64 ; d
[stack]:BED8E906 DCB 0x6F ; o
[stack]:BED8E907 DCB 0x6E ; n
[stack]:BED8E908 DCB 0x65 ; e
[stack]:BED8E909 DCB 0x21 ; !

loaddone! 为何load? 肯定是用了一些基址函数去读取, 继续跟随

```
1 signed int __fastcall sub_780057C4(int a1)
2 {
3     int v1; // r4
4
5     v1 = (unk_78005780)(a1, "com/tencent/StubShell/TxAppEntry", &unk_7801D004, 5);
6     if ( v1 )
7         return 1;
8     (unk_78015C3C)(3, "SecShell", "registerNatives Fail");
9     return v1;
10 }
```

可见, v4相当于a1传入, 如果是成功的, 执行了第五行程序, 而第五行程序为什么程序呢?

```

1 int __fastcall sub_78005780(int a1, int a2, int a3, int a4)
2 {
3     int v4; // r5
4     int v5; // r4
5     int v6; // r6
6     int v7; // r1
7     int v8; // r0
8
9     v4 = a4;
10    v5 = a1;
11    v6 = a3;
12    v7 = ((*a1 + 24))();
13    if ( v7 )
14    {
15        v8 = ((*v5 + 860))(v5, v7, v6, v4);
16        v7 = 1;
17        if ( v8 < 0 )
18        {
19            (unk_78015C3C)(3, "SecShell", "register natives error");
20            v7 = 0;
21        }
22    }
23    return v7;
24 }

```

跟踪查看：

可见，传入的4个值均为注册，而v1程序里，未知的那个值就是读取的dex文件

因此可以推理出，如果这个时候secshell加载成功，dex文件就会被隐藏

我们dump点只能在执行过程后的那一刻，而必须是在执行过程，因此，对v1程序里的那个未知参数进行跟踪：

```
debug113:7801D003 DCB 0
debug113:7801D004 unk_7801D004 DCB 0x74 ; t ; DATA XREF: sub_780057C4+8↑o
debug113:7801D004 ; debug112:off_780057F4↑o
debug113:7801D005 DCB 0x7D ; }
debug113:7801D006 DCB 1
debug113:7801D007 DCB 0x78 ; x
debug113:7801D008 DCB 0x79 ; y
debug113:7801D009 DCB 0x7D ; }
debug113:7801D00A DCB 1
debug113:7801D00B DCB 0x78 ; x
debug113:7801D00C DCB 0x39 ; 9
debug113:7801D00D DCB 0xC8
debug113:7801D00E DCB 0
debug113:7801D00F DCB 0x78 ; x
debug113:7801D010 DCB 0x96
debug113:7801D011 DCB 0x7D ; }
debug113:7801D012 DCB 1
debug113:7801D013 DCB 0x78 ; x
debug113:7801D014 DCB 0x79 ; y
debug113:7801D015 DCB 0x7D ; }
debug113:7801D016 DCB 1
debug113:7801D017 DCB 0x78 ; x
```

点D键分析或直接跟踪：

```
debug112:78017D5F ; debug112:off_780057FC↑o
debug112:78017D74 aLoad DCB "load",0
debug112:78017D79 aLandroidConten_3 DCB "(Landroid/content/Context;)V",0
debug112:78017D96 aRuncreate DCB "runCreate",0
debug112:78017DA0 aChangeenv DCB "changeEnv",0
debug112:78017DAA aReciver DCB "reciver",0
debug112:78017DB2 aLandroidConten_4 DCB "(Landroid/content/Intent;)V",0
debug112:78017DCE aTxentries DCB "txEntries",0
debug112:78017DD8 aLdalvikSystemD_3 DCB "(Ldalvik/system/DexFile;)Ljava/util/Enumeration;",0
debug112:78017E09 aCallstaticvoid DCB "CallStaticVoidFunc CallStaticObjectMethodV Exception:funcName",0
debug112:78017E47 aCallstaticvoid_0 DCB "CallStaticVoidFunc FindClass Exception:className",0
debug112:78017E78 DCB 0x43 ; C
```

由于篇幅过长，剩余内容敬请阅读下篇详解～

