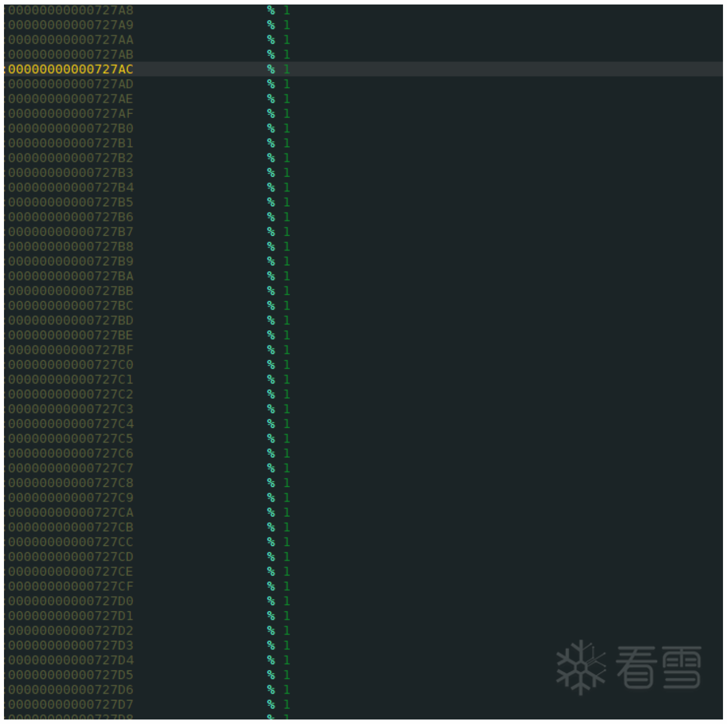
什么是脱壳的最高境界

3天前 3134

什么是脱壳的最高境界 莫过于非dump脱壳之后替换进apk中 依然正常运行 这对ELF功底无疑是一种挑战

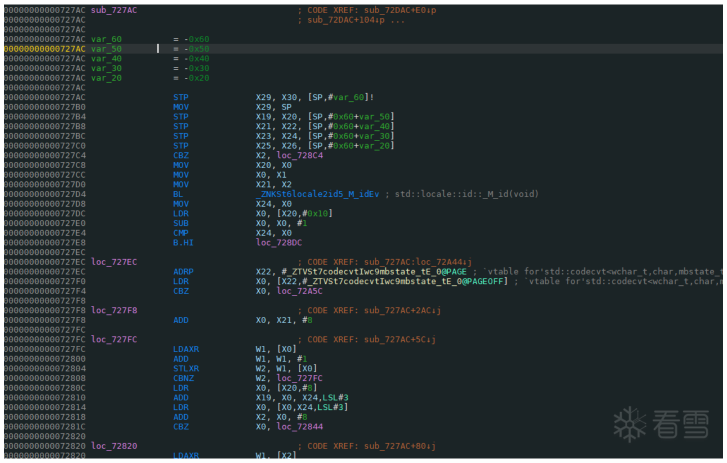
今天就献丑拿某分类第一的APK样本做一下静态脱壳分析。无需dump 自己去实现它解密的操作并且修改elf头实现脱壳并且完美运行

献上对比图,因为没有对比就没有伤害



```
00000000000727A8                     % 1
00000000000727A9                     % 1
00000000000727AA                     % 1
00000000000727AB                     % 1
00000000000727AC                     % 1
00000000000727AD                     % 1
00000000000727AE                     % 1
00000000000727AF                     % 1
00000000000727B0                     % 1
00000000000727B1                     % 1
00000000000727B2                     % 1
00000000000727B3                     % 1
00000000000727B4                     % 1
00000000000727B5                     % 1
00000000000727B6                     % 1
00000000000727B7                     % 1
00000000000727B8                     % 1
00000000000727B9                     % 1
00000000000727BA                     % 1
00000000000727BB                     % 1
00000000000727BC                     % 1
00000000000727BD                     % 1
00000000000727BE                     % 1
00000000000727BF                     % 1
00000000000727C0                     % 1
00000000000727C1                     % 1
00000000000727C2                     % 1
00000000000727C3                     % 1
00000000000727C4                     % 1
00000000000727C5                     % 1
00000000000727C6                     % 1
00000000000727C7                     % 1
00000000000727C8                     % 1
00000000000727C9                     % 1
00000000000727CA                     % 1
00000000000727CB                     % 1
00000000000727CC                     % 1
00000000000727CD                     % 1
00000000000727CE                     % 1
00000000000727CF                     % 1
00000000000727D0                     % 1
00000000000727D1                     % 1
00000000000727D2                     % 1
00000000000727D3                     % 1
00000000000727D4                     % 1
00000000000727D5                     % 1
00000000000727D6                     % 1
00000000000727D7                     % 1
```

```
大家可以清晰的看到这里是同一块地址的脱壳前后的状态
```



```
00000000000727AC sub_727AC                               ; CODE XREF: sub_72DAC+E0↓p
00000000000727AC                                         ; sub_72DAC+104↓p ...
00000000000727AC
00000000000727AC var_60          = -0x60
00000000000727AC var_50          = -0x50
00000000000727AC var_40          = -0x40
00000000000727AC var_30          = -0x30
00000000000727AC var_20          = -0x20
00000000000727AC
00000000000727AC                 STP             X29, X30, [SP,#var_60]!
00000000000727B0                 MOV             X29, SP
00000000000727B4                 STP             X19, X20, [SP,#0x60+var_50]
00000000000727B8                 STP             X21, X22, [SP,#0x60+var_40]
00000000000727BC                 STP             X23, X24, [SP,#0x60+var_30]
00000000000727C0                 STP             X25, X26, [SP,#0x60+var_20]
00000000000727C4                 CBZ             X2, loc_728C4
00000000000727C8                 MOV             X20, X0
00000000000727CC                 MOV             X0, X1
00000000000727D0                 MOV             X21, X2
00000000000727D4                 BL              _ZNKSt6locale2id5_M_idEv ; std::locale::id::_M_id(void)
00000000000727D8                 MOV             X24, X0
00000000000727DC                 LDR             X0, [X20,#0x10]
00000000000727E0                 SUB             X0, X0, #1
00000000000727E4                 CMP             X24, X0
00000000000727E8                 B.HI            loc_728DC
00000000000727EC
00000000000727EC loc_727EC                               ; CODE XREF: sub_727AC:loc_72A44↓j
00000000000727EC                 ADRP            X22, #_ZTVSt7codecvtIwc9mbstate_tE_0@PAGE ; `vtable for'std::codecvt<wchar_t,char,mbstate_t
00000000000727F0                 LDR             X0, [X22,#_ZTVSt7codecvtIwc9mbstate_tE_0@PAGEOFF] ; `vtable for'std::codecvt<wchar_t,char,m
00000000000727F4                 CBZ             X0, loc_72A5C
00000000000727F8
00000000000727F8 loc_727F8                               ; CODE XREF: sub_727AC+2AC↓j
00000000000727F8                 ADD             X0, X21, #8
00000000000727FC
00000000000727FC loc_727FC                               ; CODE XREF: sub_727AC+5C↓j
00000000000727FC                 LDAXR           W1, [X0]
0000000000072800                 ADD             W1, W1, #1
0000000000072804                 STLXR           W2, W1, [X0]
0000000000072808                 CBNZ            W2, loc_727FC
000000000007280C                 LDR             X0, [X20,#8]
0000000000072810                 ADD             X19, X0, X24,LSL#3
0000000000072814                 LDR             X0, [X0,X24,LSL#3]
0000000000072818                 ADD             X2, X0, #8
000000000007281C                 CBZ             X0, loc_72844
0000000000072820
0000000000072820 loc_72820                               ; CODE XREF: sub_727AC+80↓j
0000000000072820                 LDAXR           W1, [X2]
```

当然还有替换进去的运行图,替换进去apk运行不成功那算什么去脱壳。

至尊小仙侠

极客

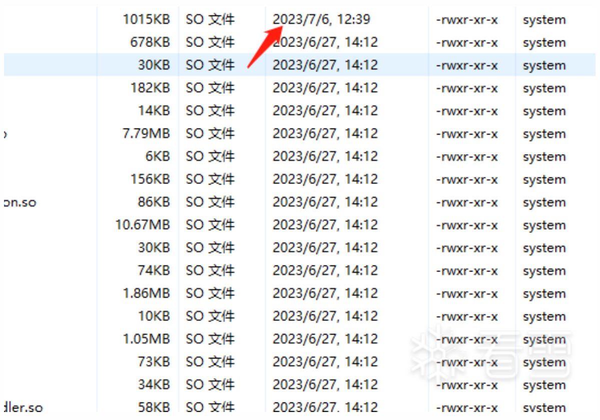
18 发帖 | 111 回帖 | 30 RANK

关注 私信

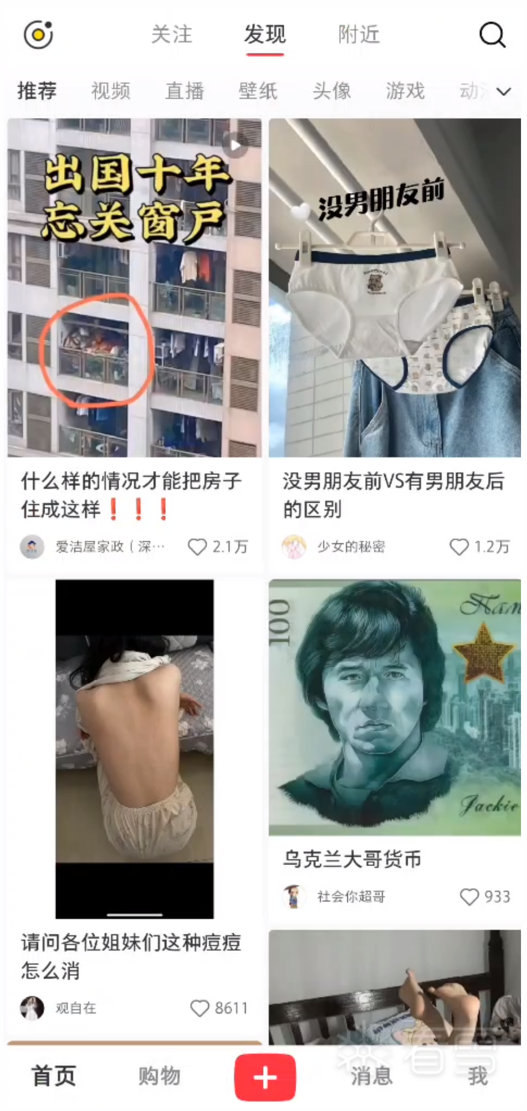
什么是脱壳的最高境界 莫过于…
今天就献丑拿某分类第一的…
简单分析
这样就可以了吗?

他的文章

- 什么是脱壳的最高境界 3135
- 什么是去花指令的最高境界 14518
- 逆向小记 bd-ticket-guard-client-cert | bd-ticket-guard-client-data 通过现象看本质 14182
- iOS Instgram 算法分析小记 7694
- [原创] WX Android 新版Rqt竟然有两套方案 12235



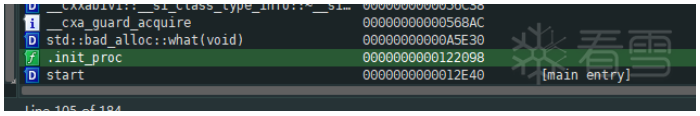
	1015KB	SO 文件	2023/7/6, 12:39	-rwxr-xr-x	system
	678KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	30KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	182KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	14KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
›	7.79MB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	6KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	156KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
on.so	86KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	10.67MB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	30KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	74KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	1.86MB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	10KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	1.05MB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	73KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
	34KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system
dler.so	58KB	SO 文件	2023/6/27, 14:12	-rwxr-xr-x	system



讲完效果就开干呗！

简单分析

- **.init_proc恢复**



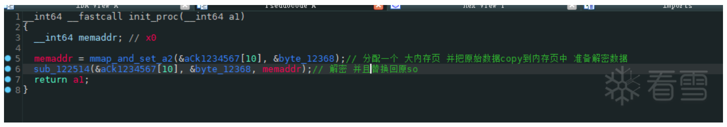
```
__cxa_guard_acquire             00000000000568AC
std::bad_alloc::what(void)      00000000000A5E30
.init_proc                      0000000000122098
start                           0000000000012E40    [main entry]
```

分析一下,没有InitArray节 但是so中存在**init_proc**函数 熟悉linker流程的小伙伴肯定知道，那么肯定是这个函数进行脱壳的操作的

- Tip：

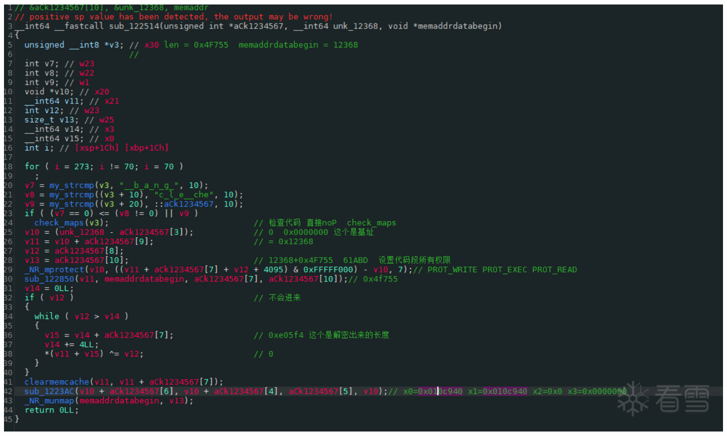
初始化（init）和终止（fini）过程是由动态链接器（linker）执行的。 动态链接器负责解析共享库的依赖关系，找出并加载所需要的库，然后解析和重定位符号。在完成这些任务之后，它会调用每个已加载模块的初始化函数。
由上文可知,函数在开头必然要分配函数所需的栈空间以及保存调用者保存寄存器因此，当你调用 dlopen() 时，实际上是动态链接器在背后完成的大部分工作。dlopen() 会通知动态链接器去加载指定的库，然后链接器会找到并执行这个库的 init 函数

- 上IDA分析



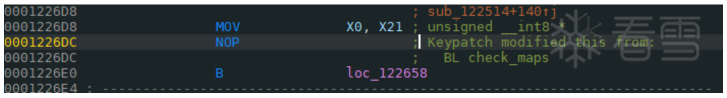
```
__int64 __fastcall init_proc(__int64 a1)
{
  __int64 memaddr; // x0

  memaddr = mmap_and_set_a2(&aCk1234567[10], &byte_12368);// 分配一个 大内存页 并把原始数据copy到内存页中 准备解密数据
  sub_122514(&aCk1234567[10], &byte_12368, memaddr);// 解密 并且替换回原so
  return a1;
}
```

很简单的两个函数，首先通过svc mmap 分配个足够大的内存空间并且把原始经过加密前的opcode复制进内存中，用于后面的解密释放



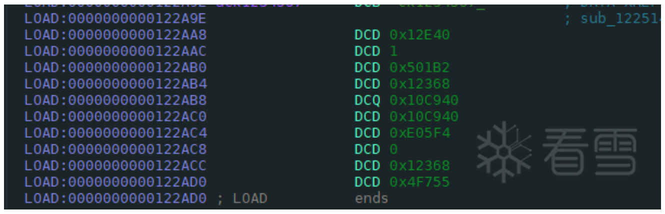
```
// &aCk1234567[10], &unk_12368, memaddr
// positive sp value has been detected, the output may be wrong!
__int64 __fastcall sub_122514(unsigned int *aCk1234567, __int64 unk_12368, void *memaddrdatabegin)
{
  unsigned __int8 *v3; // x30 len = 0x4F755  memaddrdatabegin = 12368
                       //
  int v7; // w23
  int v8; // w22
  int v9; // w1
  void *v10; // x20
  __int64 v11; // x21
  int v12; // w23
  size_t v13; // w25
  __int64 v14; // x3
  __int64 v15; // x0
  int i; // [xsp+1Ch] [xbp+1Ch]

  for ( i = 273; i != 70; i = 70 )
    ;
  v7 = my_strcmp(v3, "__b_a_n_g_", 10);
  v8 = my_strcmp((v3 + 10), "c_l_e__che", 10);
  v9 = my_strcmp((v3 + 20), ::aCk1234567, 10);
  if ( (v7 == 0) <= (v8 != 0) || v9 )
    check_maps(v3);                             // 检查代码 直接noP  check_maps
  v10 = (unk_12368 - aCk1234567[3]);            // 0  0x0000000 这个是基址
  v11 = v10 + aCk1234567[9];                    // = 0x12368
  v12 = aCk1234567[8];
  v13 = aCk1234567[10];                         // 12368+0x4F755  61ABD  设置代码段所有权限
  _NR_mprotect(v10, ((v11 + aCk1234567[7] + v12 + 4095) & 0xFFFFF000) - v10, 7);// PROT_WRITE PROT_EXEC PROT_READ
  sub_122850(v11, memaddrdatabegin, aCk1234567[7], aCk1234567[10]);// 0x4f755
  v14 = 0LL;
  if ( v12 )                                    // 不会进来
  {
    while ( v12 > v14 )
    {
      v15 = v14 + aCk1234567[7];                // 0xe05f4 这个是解密出来的长度
      v14 += 4LL;
      *(v11 + v15) ^= v12;                      // 0
    }
  }
  clearmemcache(v11, v11 + aCk1234567[7]);
  sub_1223AC(v10 + aCk1234567[6], v10 + aCk1234567[4], aCk1234567[5], v10);// x0=0x010c940 x1=0x010c940 x2=0x0 x3=0x0000000
  _NR_munmap(memaddrdatabegin, v13);
  return 0LL;
}
```

其中还有个防止inlinehook的函数 **check_maps** 当检测到调试的时候直接触发 然后结束程序 当然老规矩 直接NOP掉



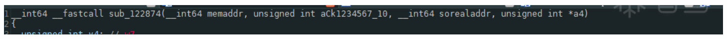
```
0001226D8                                ; sub_122514+140↑j
0001226D8           MOV          X0, X21 ; unsigned __int8 *
0001226DC           NOP                  ; Keypatch modified this from:
0001226DC                                ;   BL check_maps
0001226E0           B            loc_122658
0001226E4 ; ---------------------------------------------------------------
```

通过_NR_mprotect修改代码段权限为 **PROT_WRITE PROT_EXEC PROT_READ**



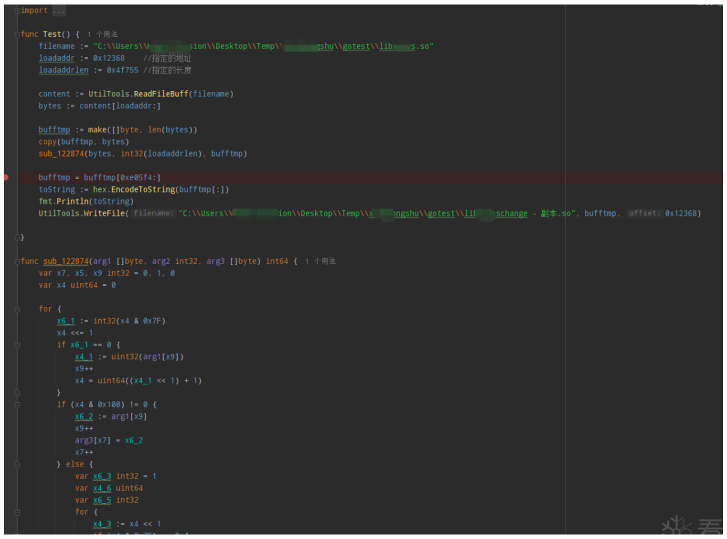
```
LOAD:0000000000122A9E                                    ; sub_12251
LOAD:0000000000122AA8                DCD 0x12E40
LOAD:0000000000122AAC                DCD 1
LOAD:0000000000122AB0                DCD 0x501B2
LOAD:0000000000122AB4                DCD 0x12368
LOAD:0000000000122AB8                DCQ 0x10C940
LOAD:0000000000122AC0                DCD 0x10C940
LOAD:0000000000122AC4                DCD 0xE05F4
LOAD:0000000000122AC8                DCD 0
LOAD:0000000000122ACC                DCD 0x12368
LOAD:0000000000122AD0                DCD 0x4F755
LOAD:0000000000122AD0 ; LOAD         ends
```

拿到三个值 基址 需要解密的头地址 以及数据长度 还有解密后应有的长度 这些值都是存在于一个数组中 当解密完成_NR_munmap
主要解密函数是



```
__int64 __fastcall sub_122874(__int64 memaddr, unsigned int aCk1234567_10, __int64 sorealaddr, unsigned int *a4)
{
```

ok 不纠结 照着还原一下



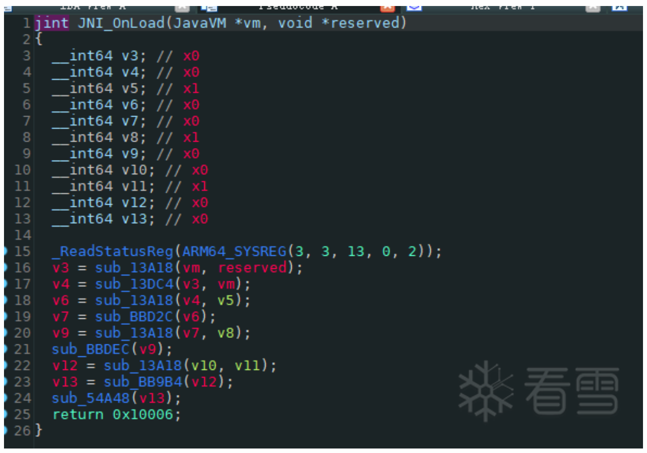
```
import ...

func Test() {  1 个用法
    filename := "C:\\Users\\[redacted]ion\\Desktop\\Temp\\[redacted]gshu\\gotest\\lib[redacted]s.so"
    loadaddr := 0x12368    //指定的地址
    loadaddrlen := 0x4f755 //指定的长度

    content := UtilTools.ReadFileBuff(filename)
    bytes := content[loadaddr:]

    bufftmp := make([]byte, len(bytes))
    copy(bufftmp, bytes)
    sub_122874(bytes, int32(loadaddrlen), bufftmp)

    bufftmp = bufftmp[0xe05f4:]
    toString := hex.EncodeToString(bufftmp[:])
    fmt.Println(toString)
    UtilTools.WriteFile( filename: "C:\\Users\\[redacted]ion\\Desktop\\Temp\\x[redacted]ongshu\\gotest\\lil[redacted]schange - 副本.so", bufftmp,  offset: 0x12368)

}

func sub_122874(arg1 []byte, arg2 int32, arg3 []byte) int64 {  1 个用法
    var x7, x5, x9 int32 = 0, 1, 0
    var x4 uint64 = 0

    for {
        x6_1 := int32(x4 & 0x7F)
        x4 <<= 1
        if x6_1 == 0 {
            x4_1 := uint32(arg1[x9])
            x9++
            x4 = uint64((x4_1 << 1) + 1)
        }
        if (x4 & 0x100) != 0 {
            x6_2 := arg1[x9]
            x9++
            arg3[x7] = x6_2
            x7++
        } else {
            var x6_3 int32 = 1
            var x4_6 uint64
            var x6_5 int32
            for {
                x4_3 := x4 << 1
```

一比一还原后，模拟so的操作从指定位置patch到内存中，当然 so是patch到内存，可是我们是文件啊,熟悉ELF结构的大佬应该知道，文件偏移地址和相对虚拟地址是需要转换的，涉及到解析ELF头的操作

当然这里有个简单的方案了，因为IDA加载so会自动的识别出虚拟地址所以我们直接在IDA中patch就好了

```python
import ida_bytes
def hex_to_bytes(filename):
    with open(filename, 'r') as file:
        hex_data = file.read().strip()  # Read the file and remove any
trailing spaces
    return bytes.fromhex(hex_data)  # Convert the hex data to bytes

filename = r'D:\Codes\Python\******\IDApy\*****\code.txt' #这里是我们解密出的
data
byte_data = hex_to_bytes(filename)
print(len(byte_data))

addr = 0x12368 #要patch的起始地址
ida_bytes.patch_bytes(addr, byte_data)
```

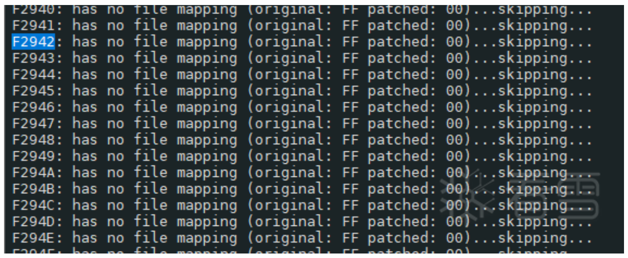
这样就可以了吗?



```
jint JNI_OnLoad(JavaVM *vm, void *reserved)
{
  __int64 v3; // x0
  __int64 v4; // x0
  __int64 v5; // x1
  __int64 v6; // x0
  __int64 v7; // x0
  __int64 v8; // x1
  __int64 v9; // x0
  __int64 v10; // x0
  __int64 v11; // x1
  __int64 v12; // x0
  __int64 v13; // x0

  _ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2));
  v3 = sub_13A18(vm, reserved);
  v4 = sub_13DC4(v3, vm);
  v6 = sub_13A18(v4, v5);
  v7 = sub_BBD2C(v6);
  v9 = sub_13A18(v7, v8);
  sub_BBDEC(v9);
  v12 = sub_13A18(v10, v11);
  v13 = sub_BB9B4(v12);
  sub_54A48(v13);
  return 0x10006;
}
```

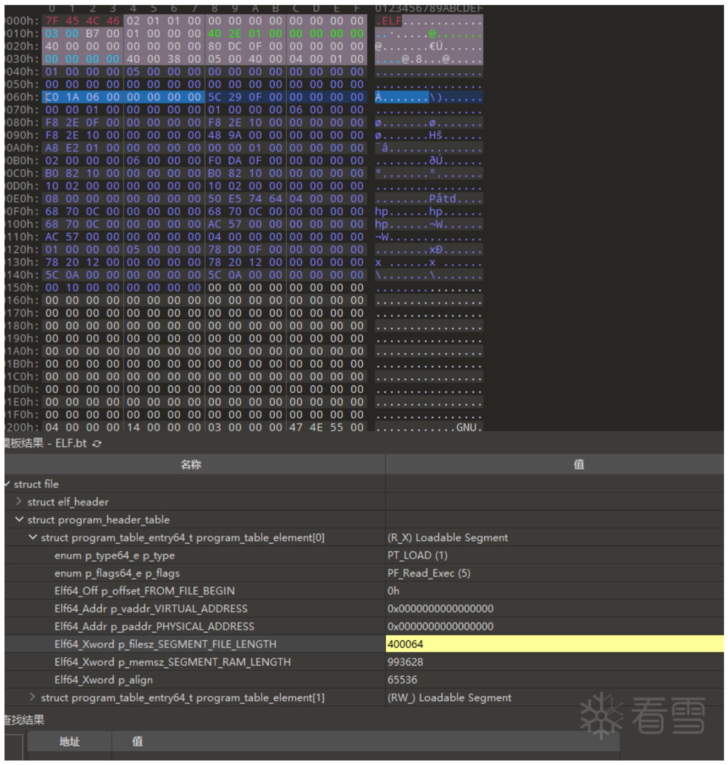
虽然IDA已经能解析成功了，其实IDApatch也只能静态看看，跟玩具一样，没啥大不了的，我们需要做到的可是能放到真机运行啊！！！！！

我们通过ida patch的方案如果需要保存会出现以下错误，错误的原因还是在ELF中



```
F2940: has no file mapping (original: FF patched: 00)...skipping...
F2941: has no file mapping (original: FF patched: 00)...skipping...
F2942: has no file mapping (original: FF patched: 00)...skipping...
F2943: has no file mapping (original: FF patched: 00)...skipping...
F2944: has no file mapping (original: FF patched: 00)...skipping...
F2945: has no file mapping (original: FF patched: 00)...skipping...
F2946: has no file mapping (original: FF patched: 00)...skipping...
F2947: has no file mapping (original: FF patched: 00)...skipping...
F2948: has no file mapping (original: FF patched: 00)...skipping...
F2949: has no file mapping (original: FF patched: 00)...skipping...
F294A: has no file mapping (original: FF patched: 00)...skipping...
F294B: has no file mapping (original: FF patched: 00)...skipping...
F294C: has no file mapping (original: FF patched: 00)...skipping...
F294D: has no file mapping (original: FF patched: 00)...skipping...
F294E: has no file mapping (original: FF patched: 00)...skipping...
```

- **tip： 在可执行文件中，某些部分（例如，动态生成或修改的数据）可能没有直接映射到原始的二进制文件。这些区域在运行时可能由程序动态分配或修改。这些区域在IDA中可以看到，但由于它们没有在原始的二进制文件中有相应的部分，因此无法被patch。**



```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
000h: 7F 45 4C 46 02 01 01 00 00 00 00 00 00 00 00 00  .ELF............
010h: 03 00 B7 00 01 00 00 00 40 2E 01 00 00 00 00 00  ..·.....@.......
020h: 40 00 00 00 00 00 00 00 80 DC 0F 00 00 00 00 00  @.......€Ü......
030h: 00 00 00 00 40 00 38 00 05 00 40 00 04 00 01 00  ....@.8...@.....
040h: 01 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00  ................
050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
060h: C0 1A 06 00 00 00 00 00 5C 29 0F 00 00 00 00 00  À.......\)......
070h: 00 00 01 00 00 00 00 00 01 00 00 00 06 00 00 00  ................
080h: F8 2E 0F 00 00 00 00 00 F8 2E 10 00 00 00 00 00  ø.......ø.......
090h: F8 2E 10 00 00 00 00 00 48 9A 00 00 00 00 00 00  ø.......Hš......
0A0h: A8 E2 01 00 00 00 00 00 00 00 01 00 00 00 00 00  ¨â..............
0B0h: 02 00 00 00 06 00 00 00 F0 DA 0F 00 00 00 00 00  ........ðÚ......
0C0h: B0 82 10 00 00 00 00 00 B0 82 10 00 00 00 00 00  °,......°,......
0D0h: 10 02 00 00 00 00 00 00 10 02 00 00 00 00 00 00  ................
0E0h: 08 00 00 00 00 00 00 00 50 E5 74 64 04 00 00 00  ........Påtd....
0F0h: 68 70 0C 00 00 00 00 00 68 70 0C 00 00 00 00 00  hp......hp......
100h: 68 70 0C 00 00 00 00 00 AC 57 00 00 00 00 00 00  hp......¬W......
110h: AC 57 00 00 00 00 00 00 04 00 00 00 00 00 00 00  ¬W..............
120h: 01 00 00 00 05 00 00 00 78 D0 0F 00 00 00 00 00  ........xÐ......
130h: 78 20 12 00 00 00 00 00 78 20 12 00 00 00 00 00  x ......x ......
140h: 5C 0A 00 00 00 00 00 00 5C 0A 00 00 00 00 00 00  \.......\.......
150h: 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
160h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
180h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
190h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1B0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1C0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1D0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
200h: 04 00 00 00 14 00 00 00 03 00 00 00 47 4E 55 00  ............GNU.
```

模板结果 - ELF.bt

名称	值
struct file	
struct elf_header	
struct program_header_table	
struct program_table_entry64_t program_table_element[0]	(R_X) Loadable Segment
enum p_type64_e p_type	PT_LOAD (1)
enum p_flags64_e p_flags	PF_Read_Exec (5)
Elf64_Off p_offset_FROM_FILE_BEGIN	0h
Elf64_Addr p_vaddr_VIRTUAL_ADDRESS	0x0000000000000000
Elf64_Addr p_paddr_PHYSICAL_ADDRESS	0x0000000000000000
Elf64_Xword p_filesz_SEGMENT_FILE_LENGTH	400064
Elf64_Xword p_memsz_SEGMENT_RAM_LENGTH	993628
Elf64_Xword p_align	65536
struct program_table_entry64_t program_table_element[1]	(RW_) Loadable Segment

查找结果

地址	值

Elf64_Xword p_filesz_SEGMENT_FILE_LENGTH 400064 60h 8h Fg:0xFF8080 Bg: Segment size in file
Elf64_Xword p_memsz_SEGMENT_RAM_LENGTH 993628 68h 8h Fg:0xFF8080 Bg: Segment size in ram
用010 ELF模版打开 找到报错的段 发现文件长度只到0x61AC0 但是内存中的长度确有0xF295C 所以我们如果patch大于0x61AC0地址的opcode自然会失败了
那我们手动把文件中的长度也改成993628就好了，当然这里涉及到文件是否足够容纳的问题，以后再跟大家探讨！
经过一系列的修改，当我们把所有段都修改好之后替换到真机上成功运行，并且执行了该so应该执行的功能！完活

物联网安全入门

最后于 7小时前 被至尊小仙侠编辑 ，原因：

#逆向分析 #基础理论

收藏 · 15　点赞 · 7　打赏　分享

最新回复 (8)

至尊小仙侠 10 3天前　2 楼 0
极客
更多好文章以及附件请到星球,哥们也要吃饭 谢谢大家了 https://t.zsxq.com/0f73KC4gN

最后于 2天前 被至尊小仙侠编辑 ，原因：

New对象处 7 3天前　3 楼 0
极客
66666

codeoooo 5 3天前 4 楼 0
极客

666

淡定小胖子 6 3天前 5 楼 0
极客

wx多少， 加一下微信群。大佬

至尊小仙侠 10 3天前 6 楼 0
极客

> 淡定小胖子 wx多少， 加一下微信群。大佬

最后于 3天前 被至尊小仙侠编辑 ，原因:

wx_阿伦_109 0 2天前 7 楼 0
临时

大佬 带我吃饭

lamper 2 11小时前 8 楼 0
极客

厉害了我的哥

mb_higyywgo 0 10小时前 9 楼 0
临时

厉害，牛

游客

登录 | 注册 方可回帖

回帖 表情 雪币赚取及消费 高级回复

返回