【2018年 网鼎杯CTF 第四场】 RE ： dalao

Ch4r1l3 / 2018-08-29 21:12:53 / 浏览数 3037 安全技术 CTF 顶(0) 踩(0)

---

这道题反编译之后，除了某些jmp和call以外，都是mov.....

这个很明显就是movfuscator 来加密的

这里就用一个神器，qira 来解这道题

这个工具记录整个运行的的过程每一个内存和寄存器的变化，我们可以随便看程序运行到某条指令时，某个内存或寄存器的内容

然后做0ctf的momo那道的时候找到了一个工具demovfuscator

这个工具可以将简化movfuscator后的程序，这样就能在ida里面反编译

虽然简化了之后一样还是看不太懂，但是还是能看出一些东西的

1. IDA静态查看逻辑

```
  alu_y = -2056;
  LOWORD(alu_s) = (_WORD)fp - 2056;
  HIWORD(alu_s) = HIWORD(fp) - 1 + (((unsigned int)(unsigned __int16)fp + 63480) >> 16);
  alu_c = HIWORD(fp) + 0xFFFF + (unsigned __int16)(((unsigned int)(unsigned __int16)fp + 63480)
  R3 = alu_s;
  stack_temp = alu_s;
  data_p = (int)&off_83FB140;
  *(_DWORD *)*(&sel_data + 1) = (char *)off_83FB140 - 4;
  data_p = (int)off_83FB140;
  *(_DWORD *)*(&sel_data + 1) = stack_temp;
  alu_x = -2012966900;
  alu_y = 2147483648;
  alu_s = 134516748;
  alu_c = 67588;
  stack_temp = 134516748;
  data_p = (int)&off_83FB140;
  *(_DWORD *)*(&sel_data + 1) = (char *)off_83FB140 - 4;
  data_p = (int)off_83FB140;
  *(_DWORD *)*(&sel_data + 1) = stack_temp;
  external = (int)memset;
  return dispatch();
}
```

这里调用了memset

```
14  data_p = (int)off_83FB140;
15  *(_DWORD *)*(&sel_data + 1) = stack_temp;
16  alu_x = (int)fp;
17  alu_y = -2056;
18  LOWORD(alu_s) = (_WORD)fp - 2056;
19  HIWORD(alu_s) = HIWORD(fp) - 1 + (((unsigned int)(unsigned __int16)fp + 63480) >> 16);
20  alu_c = HIWORD(fp) + 0xFFFF + (unsigned __int16)(((unsigned int)(unsigned __int16)fp + (
21  R3 = alu_s;
22  stack_temp = alu_s;
23  data_p = (int)&off_83FB140;
24  *(_DWORD *)*(&sel_data + 1) = (char *)off_83FB140 - 4;
25  data_p = (int)off_83FB140;
26  *(_DWORD *)*(&sel_data + 1) = stack_temp;
27  alu_x = -2012966195;
28  alu_y = 2147483648;
29  alu_s = 134517453;
30  alu_c = 67588;
31  stack_temp = 134517453;
32  data_p = (int)&off_83FB140;
33  *(_DWORD *)*(&sel_data + 1) = (char *)off_83FB140 - 4;
34  data_p = (int)off_83FB140;
35  *(_DWORD *)*(&sel_data + 1) = stack_temp;
36  external = (int)strcpy;
37  return dispatch();
38 }
```

这里调用了strcpy

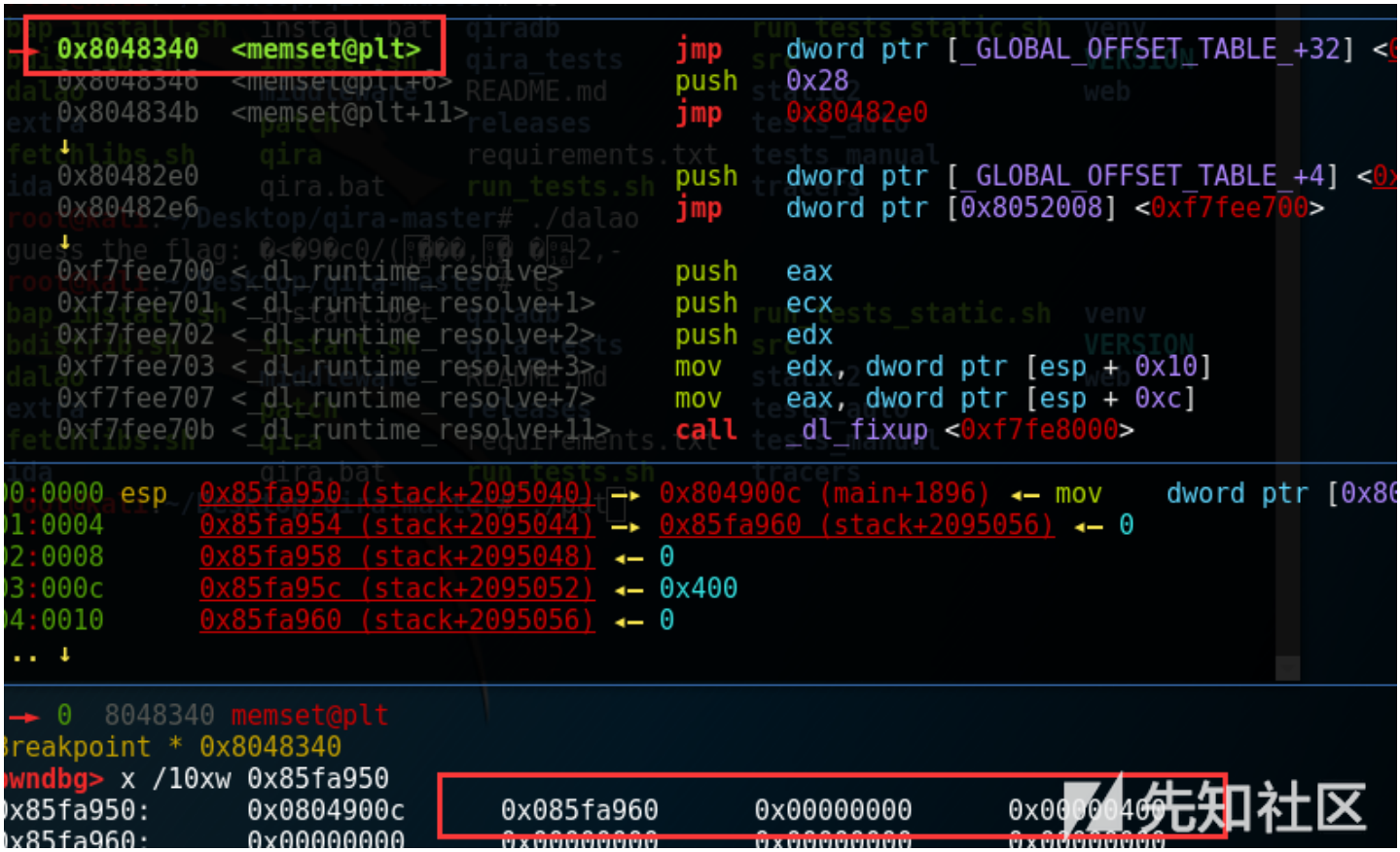
```
*(_DWORD *)*(&sel_data + 1) = (char *)off_83FB140 - 4;
data_p = (int)off_83FB140;
*(_DWORD *)*(&sel_data + 1) = stack_temp;
alu_x = -2012965586;
alu_y = 2147483648;
alu_s = 134518062;
alu_c = 67588;
stack_temp = 134518062;
data_p = (int)&off_83FB140;
*(_DWORD *)*(&sel_data + 1) = (char *)off_83FB140 - 4;
data_p = (int)off_83FB140;
*(_DWORD *)*(&sel_data + 1) = stack_temp;
external = (int)strlen;
return dispatch();
```

这里调用了strlen

可以看到调用了几个函数，但是除了调用以外的基本都看不太懂

2. gdb动态调试

因为库函数的位置是已知的，所以我们了解程序干了什么最简单的办法就是在各个函数的plt表处下断点




其他的就不放了，可以简单变为下面的c语言代码

```
memset(0x85fa960,0,0x400);
strcpy(0x85fa960,0x08052044);
int len=strlen(0x85fa960);
srand(0x00c0ffee);
for(int i=0;i<len;i++)
{
 int num=rand();
}
printf("guess the flag: %s",0x85fa960)
```

因为程序没有输入，所以很难判断主要逻辑是什么

而且我们也不知道rand出来的数字干了什么

3. qira 跟踪

用qira跟踪了下

```
1437    0x804835b    jmp 0x80482e0
1438    0x80482e0    push dword ptr [0x8052004]
1439    0x80482e6    jmp dword ptr [0x8052008]
1440    0x8049b6c    mov dword ptr [0x8052060], eax
1441    0x8049b71    mov eax, dword ptr [0x83fb140]
1442    0x8049b76    mov edx, dword ptr [0x85fa95c]
1443    0x8049b78    mov dword ptr [0x81fb120], edx
1444    0x8049b7e    mov eax, 0x83fb140
1445    0x8049b83    mov edx, 1
1446    0x8049b88    nop
1447    0x8049b89    mov dword ptr [0x83fb184], eax
1448    0x8049b8e    mov eax, dword ptr [0x83fb184]
1449    0x8049b95    mov edx, dword ptr [0x83fb140]
1450    0x8049b9b    lea edx, dword ptr [0x85fa960]
1451    0x8049b9e    nop
1452    0x8049b9f    nop
1453    0x8049ba0    nop
1454    0x8049ba1    mov dword ptr [0x83fb140], edx
1455    0x8049ba3    mov edx, dword ptr [0x81fb120]
1456    0x8049ba9    mov eax, edx
1457    0x8049bab    mov eax, dword ptr [0x83fb140]
1458    0x8049bb0    mov dword ptr [0x81fb120], eax
1459    0x8049bb5    mov eax, 0x83fb140
1460    0x8049bba    mov edx, 1
1461    0x8049bbf    nop

EAX:  0x16718b06    ECX:  0x16718b06    EDX
```

第一个rand出来的是0x16718b06

然后mov 转移到0x8052060

```
1466    0x8049bd4    mov eax, dword ptr [0x8052060]
1467    0x8049bd9    mov dword ptr [0x805206c], eax
```

再从0x8052060转移到0x805206c

```
1584    0x8049d51    mov eax, dword ptr [0x805206c]
1585    0x8049d56    mov edx, dword ptr [0x8052064]
1586    0x8049d5c    mov dword ptr [0x81fb090], eax
```

然后0x805206c->0x81fb090

```
1604    0x8049d9d    mov ecx, dword ptr [0x81fb090]
1605    0x8049da3    mov dword ptr [0x81fb004], ecx
```

0x81fb090->0x81fb004

```
1611    0x8049dcd    mov cx, word ptr [0x81fb004]
1612    0x8049dd4    mov ecx, ecx
1613    0x8049dd6    not cx
1614    0x8049dd9    nop
```

然后这里就是对rand出来的东西进行操作

等价于

```
r=0x16718b06
r=r&0xffff
r=~r // r=0xffff-r
```

```
    0x8049de3    lea edx, dword ptr [0x74f9]
    0x8049de6    mov edx, dword ptr [0x8083324]
    0x8049ded    mov ecx, dword ptr [0x81fb010]
    0x8049df3    mov edx, dword ptr [0x810332c]
    0x8049df6    mov word ptr [0x81fb008], dx
```

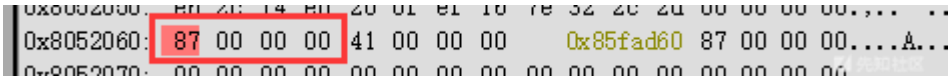
这里是

```
r=r+1
```

但是这样跟下去没完没了的

然后发现玄学只跟着0x8049B60这块内存的时候

会从四个字节的变为单个字节




把前四个字节记录下来，和bss段那些异或了下，发现是flag，然后就把24个字符全部dump出来，解密出来就是flag了

解密的脚本如下

```
b=[0x87,0x50,0x8d,0x5e,0x8d,0x53,0x48,0x4b,0x4d,0x7e,0x83,0x87,0x83,0x49,0x72,0x89,0x45,0x6e,0x8b,0x75,0x4e,0x56,0x49,0x50]
a=[0xE1, 0x3C, 0xEC, 0x39, 0xF6, 0x63, 0x30, 0x2F, 0x28, 0x1F,
 0xE7, 0xE5, 0xE6, 0x2C, 0x14, 0xED, 0x20, 0x0F, 0xEF, 0x16,
 0x7E, 0x32, 0x2C, 0x2D]
c=''
for i in range(24):
   c+=chr(a[i]^b[i])
print(c)
```

得到flag是

flag{0xdeadbeefdeadc0de}

总结

玄学re


上一篇：2018年 KCon 议题解读 |... 下一篇：【2018年 网鼎杯CTF 第四场...

1. 1 条回复

a1**** 2018-08-30 09:47:17

沙发！！

0 回复Ta

---

登录 后跟帖

先知社区

---

现在登录

热门节点

技术文章

社区小黑板

目录