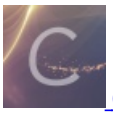
movfuscator混淆了解一下 CTF

原创

坚强的女程序员 于 2018-04-10 00:14:08 发布 1799 收藏 5

分类专栏： CTF 文章标签： reverse movfuscator

本文链接：https://blog.csdn.net/qq_33438733/article/details/79860304



参考文章

https://pediy.com/thread-208786.htm

前言

这次来了解一下movfuscator混淆。

例题一

https://secgroup.github.io/2017/02/06/alexctf2017-writeup-packed-movement/

这是一道AlexCTF 2017的re。首先拿到题目，查壳是upx，使用upx -d脱壳。

ida反汇编可以看到全是mov指令，并且在开头注册了两个信号。

```
                mov     sesp, esp
                mov     esp, off_840B140
                mov     esp, [esp+var_200068]
                mov     esp, [esp+var_200068]
                mov     esp, [esp+var_200068]
                mov     esp, [esp+var_200068]
                mov     dword ptr [esp+0], 0Bh ; sig
                mov     [esp+act], offset sa_dispatch ; act
                mov     [esp+oact], 0   ; oact
                call    _sigaction
                mov     esp, off_840B140
                mov     esp, [esp+var_200068]
                mov     esp, [esp+var_200068]
                mov     esp, [esp+var_200068]
                mov     dword ptr [esp+0], 4 ; sig
                mov     [esp+act], offset sa_loop ; act
                mov     [esp+oact], 0   ; oact
                call    _sigaction

                public master_loop
master_loop:                            ; DATA XREF: .data:sa_loop↓
```


不知道怎么下手，我们尝试运行一下程序，要求我们输入flag，这其中肯定存在cmp逻辑，就是不知道会不会对我们的输入进行一个加密处理。全局搜索字符串，发现全是乱七八糟的东西。

```
LOAD:080…  0000000A     C     sigaction
LOAD:080…  00000005     C     exit
LOAD:080…  00000007     C     strlen
```

```
LOAD:080…  00000006    C    write
LOAD:080…  00000005    C    read
LOAD:080…  0000000A    C    GLIBC_2.0
.data:08…  0000000C    C    Well Done!\n
.data:08…  0000000B    C    Well Done\n
.data:08…  0000000D    C    wrong flag!\n
.data:08…  0000000D    C    Wrong Flag!\n
.data:08…  0000000F    C    Guess a flag:
.data:08…  00000008    C    \a\a\t\t\v\v\r\r
.data:08…  0000005E    C    !!##%%'')) ++--//1133557799;;==??AACCEEGGIIKKMMOOQQSSUUWWYY[[]…
.data:08…  00000005    C    \a\n\v\n\v
.data:08…  0000005D    C    \"#\"#&'&'*++*././23236767:;:;>?>?BCBCFGFGJKJKNONORSRSVWVWZ[Z…
.data:08…  0000005B    C    $%&'$%&',-./,-./45674567<=>?<=>?DEFGDEFGLMNOLMNOTUVWTUVW\\]^_…
.data:08…  00000057    C    ()*+,-./()*+,-./89:;<=>?89:;<=>?HIJKLMNOHIJKLMNOXYZ[\\]^_XYZ[…
.data:08…  00000007    C    xyz{|}~
.data:08…  0000004F    C    0123456789:;<=>?0123456789:;<=>?PQRSTUVWXYZ[\\]^_PQRSTUVWXYZ[…
.data:08…  0000000F    C    pqrstuvwxyz{|}~
.data:08…  0000005F    C     !\"#$%&'()*+,-./0123456789:;<=>? !\"#$%&'()*+,-./0123456789:…
.data:08…  0000001F    C    `abcdefghijklmnopqrstuvwxyz{|}~
.data:08…  0000003F    C    @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{…
.data:08…  0000003F    C    @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{…
.data:08…  00000060    C      \"\"$$&&((**,,..0022446688::<<>>@@BBDDFFHHJJLLNNPPRRTTVVXXZ…
.data:08…  00000060    C     ! !$%$%()(),-,-0101454589891<=<=@A@ADEDEHIHILMLMPQPQTUTUXYXY\…
.data:08…  00000008    C    \b\t\n\v\b\t\n\v
.data:08…  00000061    C    \x1B !\"# !\"#()*+()*+0123012389:;89:;@ABC@ABCHIJKHIJKPQRSPQR…
.data:08…  00000060    C     !\"#$%&' !\"#$%&'01234567012345671@ABCDEFG@ABCDEFGPQRSTUVWPQR…
.data:08…  00000005    C    \a\b\t\n\v
.data:08…  00000005    C    \a\b\t\n\v
.data:08…  00000060    C     !\"#$%&'()*+,-./ !\"#$%&'()*+,-./@ABCDEFGHIJKLMNO@ABCDEFGHIJ…
.data:08…  00000005    C    \a\b\t\n\v
```


不过还是发现了一些端倪，可以看到well done！

向上回溯，可以看到wrong flag ， strlen等字样，并且还注意到了"}"字符，

```
.text:0805F8AC                  mov     dl, [eax]
.text:0805F8AE                  mov     R0, edx
.text:0805F8B4                  mov     edx, 0
.text:0805F8B9                  mov     dl, byte ptr R0
.text:0805F8BF                  mov     edx, alu_sex8[edx*4]
.text:0805F8C6                  mov     R3, edx
.text:0805F8CC                  mov     R2, '}'
.text:0805F8D6                  mov     eax, R3
.text:0805F8DB                  mov     edx, R2
.text:0805F8E1                  mov     ecx, 8806001Ch
.text:0805F8E6                  mov     branch_temp, ecx
```


继续向上看，，发现类似的字符

```
.text:0805EF26                  mov     R0, edx
.text:0805EF2C                  mov     edx, 0
.text:0805EF31                  mov     dl, byte ptr R0
.text:0805EF37                  mov     edx, alu_sex8[edx*4]
.text:0805EF3E                  mov     R3, edx
.text:0805EF44                  mov     R2, 'c'
.text:0805EF4E                  mov     eax, R3
.text:0805EF53                  mov     edx, R2
```


因此我们便对mov R2 ， #立即数 这条指令产生了兴趣，全局搜索一下，便可得到flag。

Address	Function	Instruction
.text:080493DB		mov R2, 'A'
.text:08049DDE		mov R2, 'L'

```
.text:0804A7E1                                  mov     R2, 'E'
.text:0804B1E4                                  mov     R2, 'X'
.text:0804BB9C                                  mov     R2, 'C'
.text:0804C59F                                  mov     R2, 'T'
.text:0804CFA2                                  mov     R2, 'F'
.text:0804D9A5                                  mov     R2, '{'
.text:0804E357                                  mov     R2, 'M'
.text:0804ED5A                                  mov     R2, '0'
.text:0804F75D                                  mov     R2, 'V'
.text:08050160                                  mov     R2, 'f'
.text:08050B0C                                  mov     R2, 'u'
.text:0805150F                                  mov     R2, 's'
.text:08051F12                                  mov     R2, 'c'
.text:08052915                                  mov     R2, '4'
.text:080532BB                                  mov     R2, 't'
.text:08053CBE                                  mov     R2, '0'
.text:080546C1                                  mov     R2, 'r'
.text:080550C4                                  mov     R2, '_'
.text:08055A64                                  mov     R2, 'w'
.text:08056467                                  mov     R2, '0'
.text:08056E6A                                  mov     R2, 'r'
.text:0805786D                                  mov     R2, 'k'
.text:08058207                                  mov     R2, '5'
.text:08058C0A                                  mov     R2, '_'
.text:0805960D                                  mov     R2, 'l'
.text:0805A010                                  mov     R2, '1'
.text:0805A9A4                                  mov     R2, 'k'
.text:0805B3A7                                  mov     R2, 'e'
.text:0805BDAA                                  mov     R2, '_'
.text:0805C7AD                                  mov     R2, 'm'
.text:0805D13B                                  mov     R2, '4'
.text:0805DB3E                                  mov     R2, 'g'
.text:0805E541                                  mov     R2, '1'
.text:0805EF44                                  mov     R2, 'c'
.text:0805F8CC                                  mov     R2, '}'
```


例题二

https://github.com/ctfs/write-ups-2015/tree/master/ekoparty-pre-ctf-2015/rev/mov

这题基本思路还是类似的，也是全局搜索关键字符串，然后向上回溯。

```
's' LOAD:080…  00000007  C  printf
's' LOAD:080…  00000007  C  strlen
's' LOAD:080…  00000007  C  memcmp
's' LOAD:080…  0000000A  C  GLIBC_2.0
's' .data:08…  00000008  C  Error\n\n
's' .data:08…  00000039  C  Congrats!\nThis is the flag you are looking for: EKO{%s}\n
's' .data:08…  0000000F  C  Processing...\n
's' .data:08…  00000037  C       \\/     \\/     |__|       \\/            \\/     \n\n
's' .data:08…  00000036  C   \\___  >__|_ \\____/|   __(____  /__|   |__|  / ____|\n
's' .data:08…  00000036  C   \\  ___/|    <  <_> )  |_> > __ \\|  | \\/|  |  \\___  |\n
's' .data:08…  00000036  C   _/ __ \\|  |/ /  _ \\\\____ \\__  \\\\_  __ \\   __<   |  |\n
's' .data:08…  00000036  C     ____ |  | ______ ___________ ________/  |_ ___.__.\n
's' .data:08…  00000036  C          __                                  __        \n
's' .data:08…  00000008  C  \a\a\t\t\v\v\r\r
's' .data:08…  0000005E  C  !!##%%''))++--//1133557799;;==??AACCEEGGIIKKMMOOQQSSUUWWYY[[]…
's' .data:08…  00000005  C  \a\n\v\n\v
's' .data:08…  0000005D  C  \"#\"#&'&'*+*+././23236767:;:;>?>?BCBCFGFGJKJKNONORSRSVWVWZ[Z…
's' .data:08…  0000005B  C  $%&'$%&',-./,-./45674567<=>?<=>?DEFGDEFGLMNOLMNOTUVWTUVW\\]^_…
```


因为我们知道flag格式即为EKO{}而刚好这个字符串中有一个EKO{%s}字样，可以猜测，这时在stack中应该能看到flag，但是我在用IDA调试的时候发现程序一运行就自动结束了。所以我就用gdb来attatch了。

通过ida可以知道我们在0804B41B 处下断点。

最终可以得到flag

EKO{All_You_Need_Is_m0v}

###例题三SusCTF 2018 MoVfuscationl

https://github.com/susers/Writeups/tree/master/2018/SUSCTF/Reverse/movfuscation1

这道题如果不采用逐位爆破的方式去解就会比较难了，我这里两种方法都实现一遍。

运行程序，测试发现它会将正确字符打印出来，因此可以采用逐位爆破的方法。

写个脚本如下：

```
import string
import subprocess
import sys
def main():
    bin="354e846facdf6f3d3205a1465d2fd811"
    flag=""
    length=0
    while 1:
        for i in string.printable:
            p = subprocess.Popen("./%s" % bin,stdin=subprocess.PIPE,stdout=subprocess.PIPE)
            p.stdin.write("1\n")
            p.stdin.write(flag+i+"\n")
            output = p.stdout.readlines()[-1]
            if len(output)-8 > length:
                length+=1
                flag+=i
                print flag
                break
if __name__=="__main__":
    main()
```

第二种方法：直接调试，哈哈我没调出来。GG

4.11更新。

这里我是通过自己写个demo然后用movfuscator混淆后对比来看代码的。

https://github.com/xoreaxeaxeax/movfuscator 照着做编译不会有什么问题的。我实在kali 64下编译通过的。

还有个链接可以参考一下 https://labs.mwrinfosecurity.com/blog/avrop/ 应该会有点价值

这题我问了一下出题的师傅，理解了大概的思路。

```
demo.c
#include<stdio.h>
int main(void){
    char* str = "CTF{";
    int cmp[]={67};
    int pos[]={2,0,3};
    if(str[0]==cmp[pos[1]]){
        printf("%c",str[0]);
    }
    return 0;
}
```

我自己写了个demo然后movcc编译。

```
.data:0804B020 unk_804B020     db  43h ; C          ; DATA XREF: .text:08048BF6↑o
.data:0804B021                 db    0
.data:0804B022                 db    0
.data:0804B023                 db    0
.data:0804B024 unk_804B024     db    2              ; DATA XREF: .text:08048C5C↑o
.data:0804B025                 db    0
.data:0804B026                 db    0
.data:0804B027                 db    0
.data:0804B028                 db    0
.data:0804B029                 db    0
.data:0804B02A                 db    0
.data:0804B02B                 db    0
.data:0804B02C                 db    3
.data:0804B02D                 db    0
.data:0804B02E                 db    0
.data:0804B02F                 db    0
.data:0804B030 unk_804B030     db  25h ; %          ; DATA XREF: .text:08049424↑o
.data:0804B031                 db  63h ; c
.data:0804B032                 db    0
.data:0804B033 aCtf            db 'CTF{',0          ; DATA XREF: .text:08048BB1↑o
.data:0804B038                 align 10h
.data:0804B040                 public R0
```


我在data段发现了这些字符。包括pos数组。虽然混淆后的程序很难懂，但是data段中的数据出卖了出题者。
我们可以来看一下题目中的data段的数据。

```
0804F020  0D 00 00 00 0B 00 00 00  07 00 00 00 0B 00 00 00  ................
0804F030  00 00 00 00 08 00 00 00  0C 00 00 00 00 00 00 00  ................
0804F040  00 00 00 00 0A 00 00 00  05 00 00 00 07 00 00 00  ................
0804F050  08 00 00 00 06 00 00 00  04 00 00 00 0E 00 00 00  ................
0804F060  0D 00 00 00 01 00 00 00  00 00 00 00 0E 00 00 00  ................
0804F070  0A 00 00 00 04 00 00 00  0C 00 00 00 08 00 00 00  ................
0804F080  0F 00 00 00 0E 00 00 00  0A 00 00 00 0E 00 00 00  ................
0804F090  00 00 00 00 00 00 00 00  05 00 00 00 08 00 00 00  ................
0804F0A0  43 6F 6E 67 72 61 7A 7E  0A 00 7D 0A 00 25 63 00  Congraz~..}..%c.
0804F0B0  53 55 53 43 54 46 7B 00  46 6C 61 67 20 69 73 20  SUSCTF{.Flag·is·
0804F0C0  00 25 73 00 59 6F 75 72  20 66 6C 61 67 3F 0A 00  .%s.Your·flag?..
0804F0D0  25 64 00 4C 65 6E 67 74  68 20 6F 66 20 66 6C 61  %d.Length·of·fla
0804F0E0  67 3A 20 00 31 32 33 34  35 36 37 38 39 30 61 62  g:·.1234567890ab
0804F0F0  63 64 65 66 67 68 69 6A  6B 6C 6D 6E 6F 70 71 72  cdefghijklmnopqr
0804F100  73 74 75 76 77 78 79 7A  41 42 43 44 45 46 47 48  stuvwxyzABCDEFGH
0804F110  49 4A 4B 4C 4D 4E 4F 50  51 52 53 54 55 56 57 58  IJKLMNOPQRSTUVWX
0804F120  59 5A 00 00 00 00 00 00  00 00 00 00 00 00 00 00  YZ..............
```


从这些数据，并且结合指令的逻辑，因为全是mov指令，它是一条线下来的，而且movcc是无法混淆库函数的。因此我们可以进行大胆的推测整个程序的逻辑。
显示比较SUSCTF{，之后通过pos[i]获取固定字符串的偏移量，循环的进行比较，那么这个pos[i]数组就应该是事先定义好的，可以看到0x0804f020开始的数据很想是pos[i]数组，注意偏移量0对应的字符应该是"1",所以我们可以根据偏移量求出{}之间的flag最终可以得到flag。
虽然最终还是没有用调试的方法硬干，但是这也不失为一种方法吧！刚完睡觉！晚安！ xinxin

总结

怎么说呢，本来先看下movfuscator这个项目的，但是环境一直搭建不起来，所以也就放弃了。
通过这几个题可以初步了解一下movfuscator混淆。