CTF Reverse

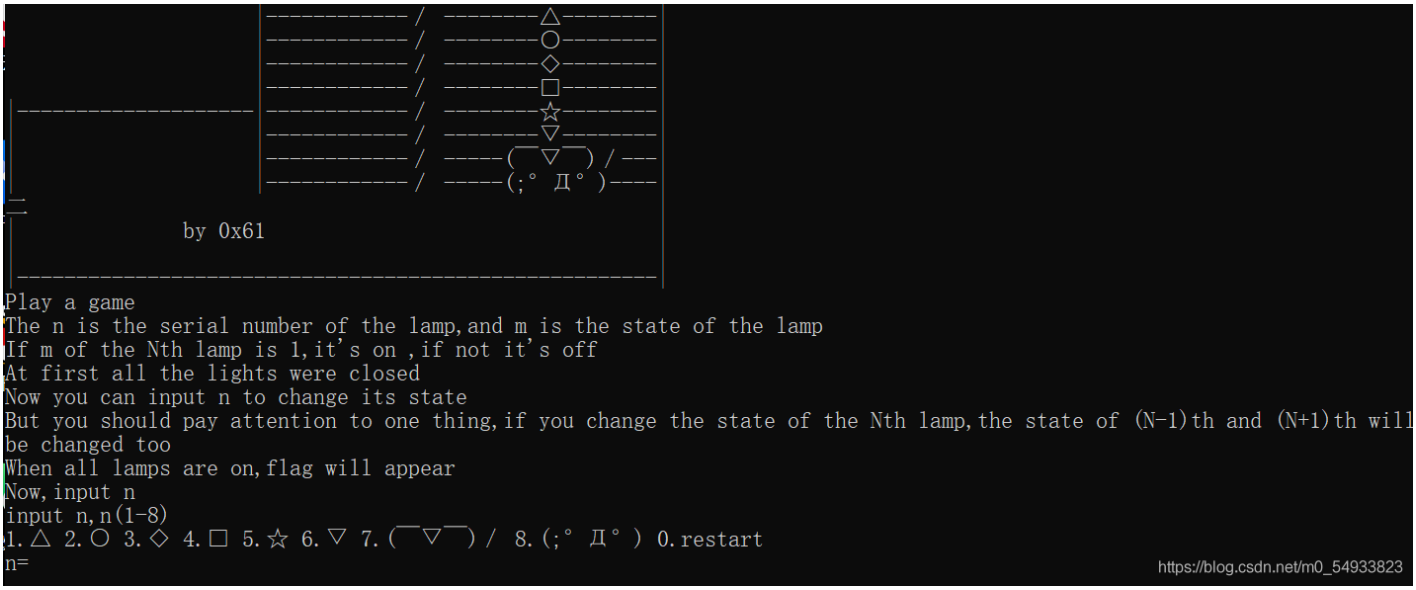
原创

Undefined-Liver 于 2021-05-11 00:23:10 发布 173 收藏
分类专栏： CTF日常练习OvO 文章标签： python c语言 经验分享

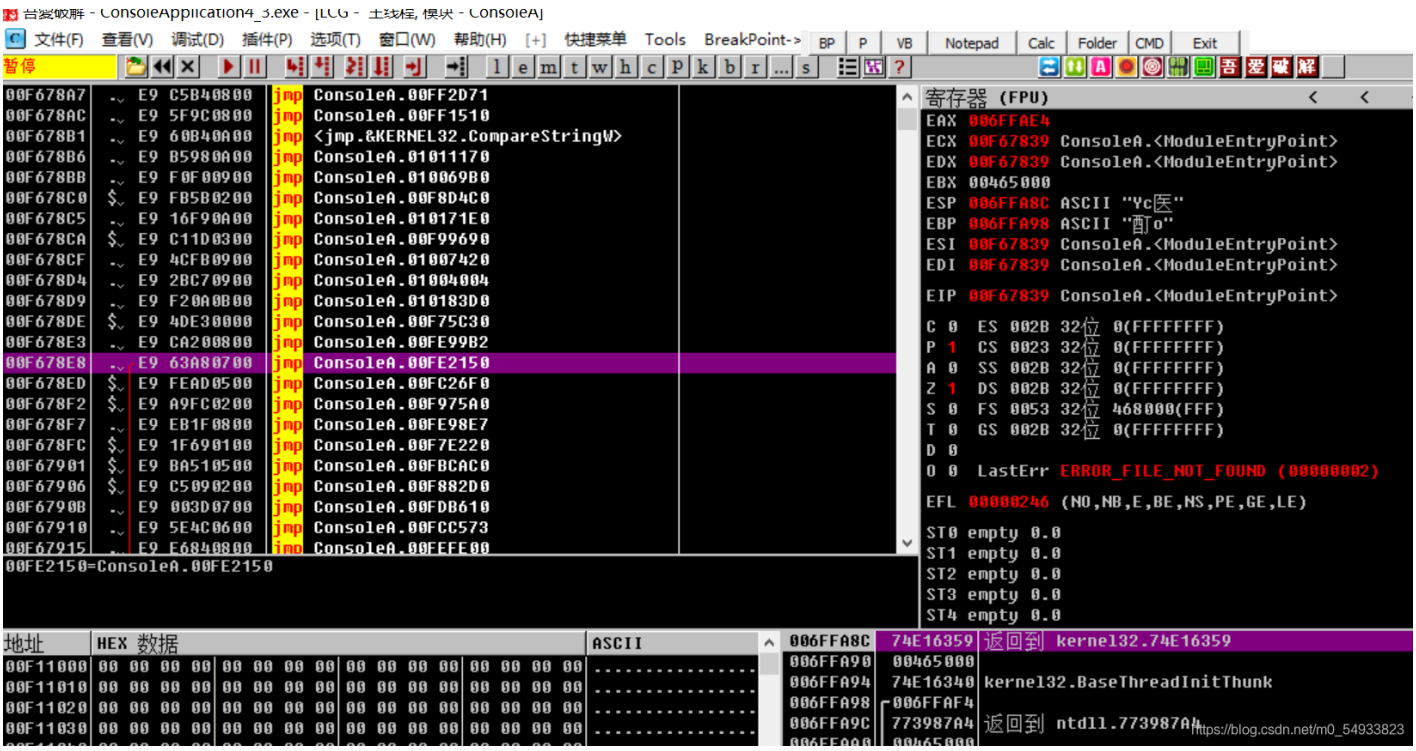
2021.5.11 CTF Reverse 两题……

ConsoleApplication4_3——Writeup

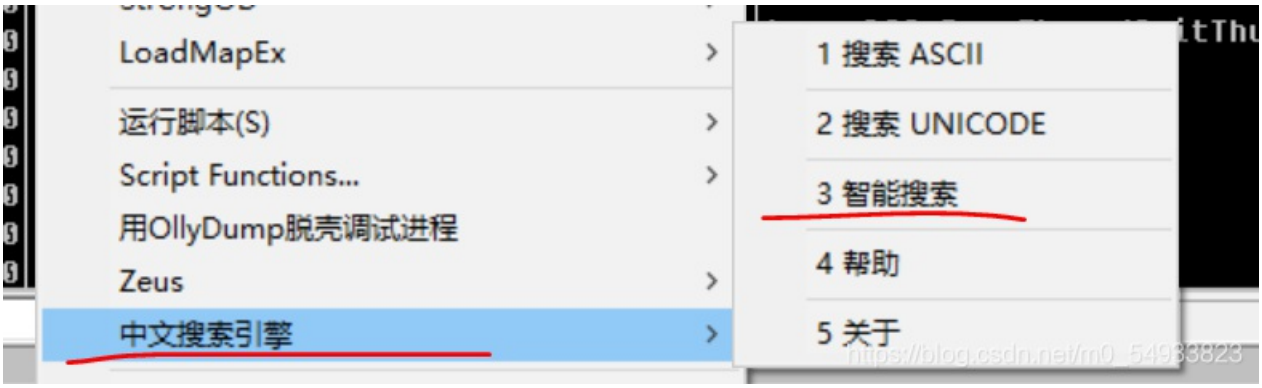
拿到exe文件，打开，研究一波：

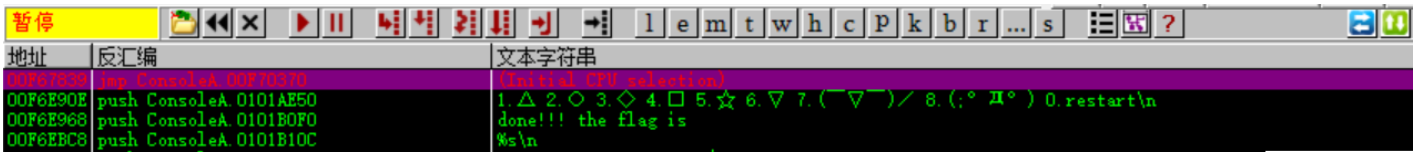


发现是让我们输入1-8，将所有开关闭合，就能获得flag。哈哈，那开始尝试吧。然鹅，得劲的人已经打开了od……



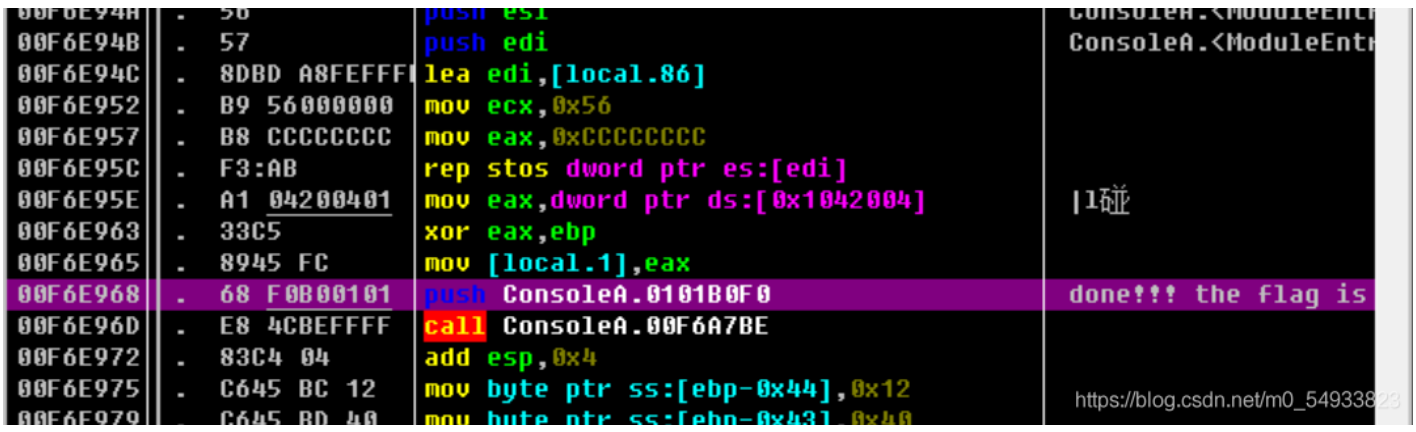
emmm。。翻看了一会儿，好复杂的样子。可以尝试用字符串搜索一些关键字符的位置。



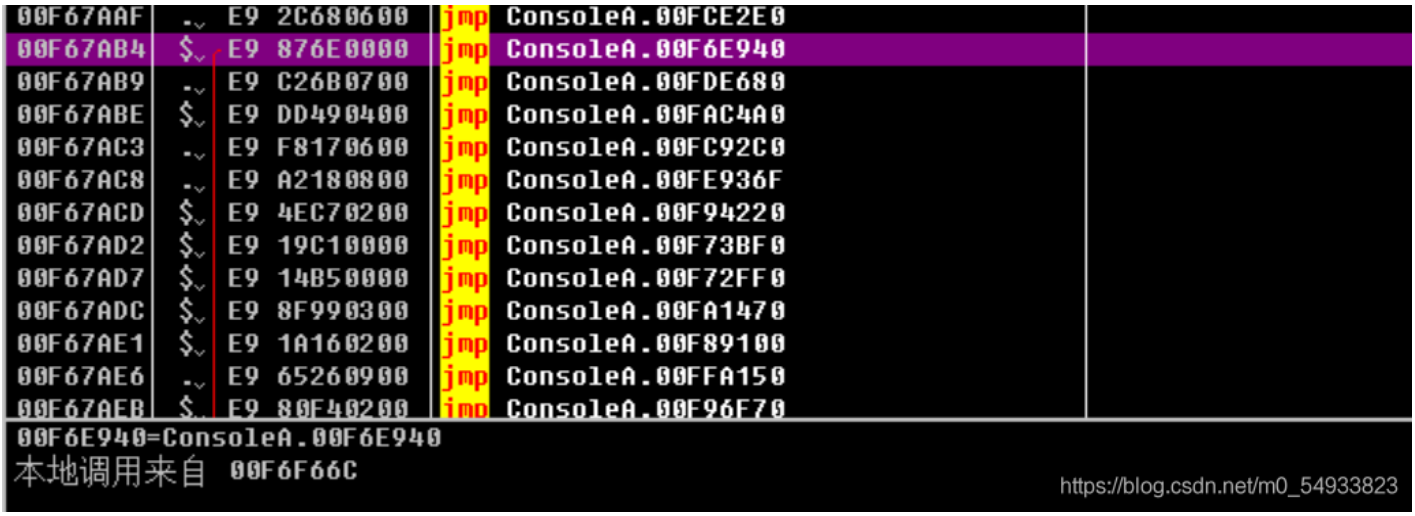


啊这，一开头就出现了“敏感”词汇？？

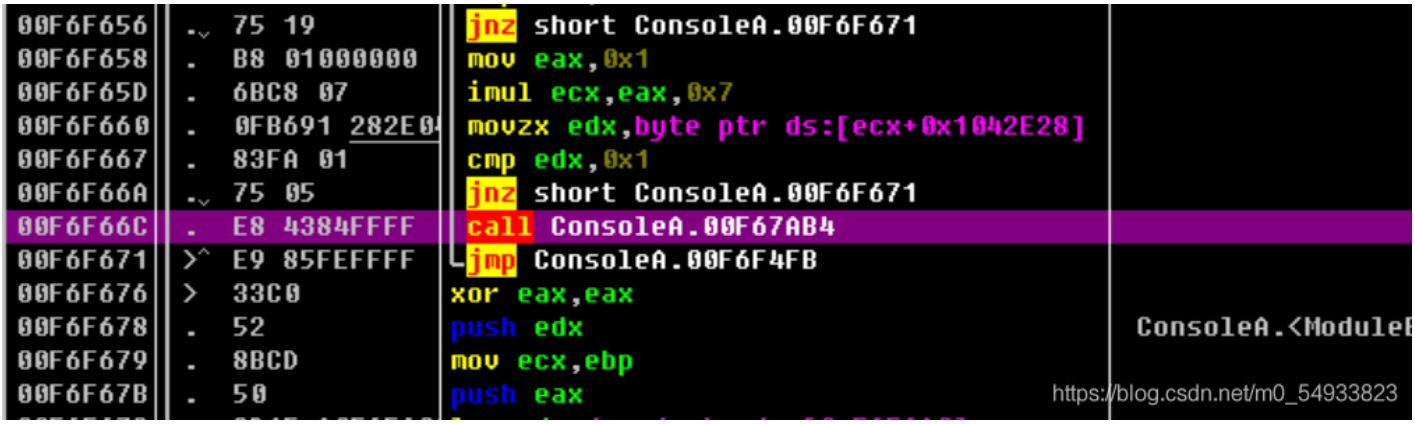
双击打开字符串的位置：



emmm，大概就是这里调用了一个函数，这个函数在下面一系列的mov的操作下自解密出了flag并输出了。那么我们找到调用这个函数的位置：点击函数头的push指令，跳转到调用位置：

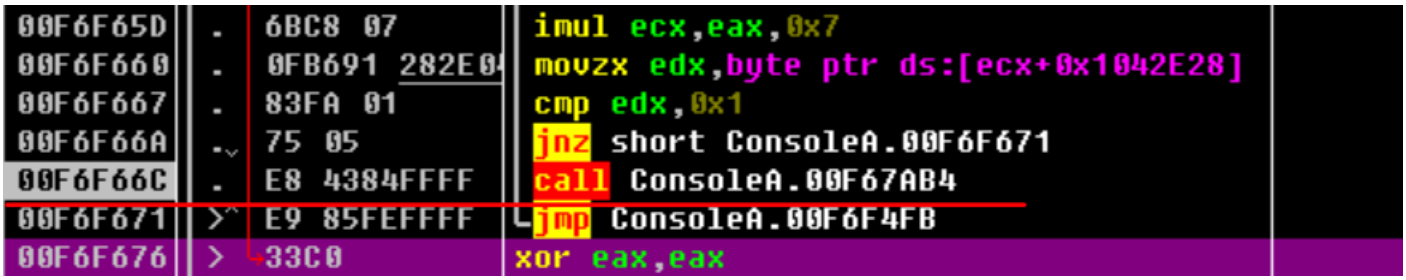


发现这里依旧用jmp指令跳转，所以继续回溯：

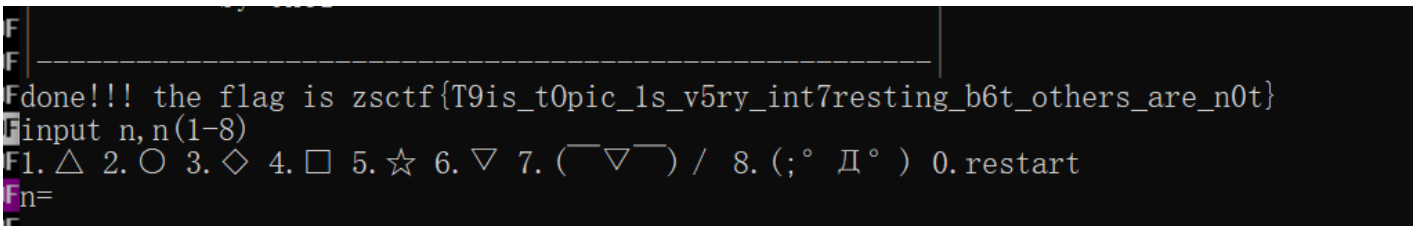


那么就找到了调用的位置，查看附近的指令，调用位置的上边是好多个jnz条件跳转指令，只有这些指令都不执行，才执行输出flag的函数，所以找到第一个jnz指令的位置设置一个断点，让程序运行至停到该位置：

但这时候只要程序继续往下，就不能确保它能执行到调用输出flag函数的位置，所以此时要在之前的位置“设置一个新的EIP”，接着运行……



芜湖，flag出来了？！（此时开关并未全部闭合，但flag照样出来了……）



end。。。

然后老师又讲了另一个方法……

回到之前很多jnz条件跳转的地方，将jnz指令都汇编成nop，即什么也不做。然后执行，就保证能运行到调用flag函数的指令，最后结果也和上图一样。

这两种方法我觉得都是绕过jnz指令，执行想要的指令，只是方式不一样罢了。

2.xor——writeup

拿到文件，发现是个未知的文件类型，这种情况，丢进ida瞧瞧呗。等待ida加载完成……

首先去main函数看看吧，将它反汇编一下：

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int i; // [rsp+1Ch] [rbp-44h]
4    char s[8]; // [rsp+20h] [rbp-40h]
5    __int64 v6; // [rsp+28h] [rbp-38h]
6    __int64 v7; // [rsp+30h] [rbp-30h]
7    __int64 v8; // [rsp+38h] [rbp-28h]
8    unsigned __int64 v9; // [rsp+48h] [rbp-18h]
9
10   v9 = __readfsqword(0x28u);
11   *(_QWORD *)s = 0LL;
12   v6 = 0LL;
13   v7 = 0LL;
14   v8 = 0LL;
15   gets(s, argv, envp);
16   for ( i = 0; i < strlen(s); ++i )
17   {
18     s[i] ^= s[i + 1];
19     if ( s[i] != check[i] )
20     {
21       puts("Wrong!");
22       exit(-1);
23     }
24   }
25   puts("Congratulations!");
26   return 0;
27 }
```


这里面发现“Wrong！”和“congratulations！”，明显要绕过W，然后执行C，很有可能就能得到flag了。

研究入口判断条件：s[i]!=check[i]，之前s还进行了自身邻位的异或运算：s[i]^=s[i+1]；那大概的意思就是要使s[i]与s[i+1]进行异或运算后，s[i]等于check[i]。那么我们可以从check数组开始研究，从后往前解出s数组。

查看check数组：

```
.data:0000000000601060 check           db 0Ah                  ; DATA X
.data:0000000000601061                 db  0Dh
.data:0000000000601062                 db    6
.data:0000000000601063                 db  1Ch
.data:0000000000601064                 db    3
.data:0000000000601065                 db  17h
.data:0000000000601066                 db  1Dh
.data:0000000000601067                 db  2Dh ; -
.data:0000000000601068                 db  36h ; 6
.data:0000000000601069                 db  1Ah
.data:000000000060106A                 db  2Ch ; ,
.data:000000000060106B                 db    9
.data:000000000060106C                 db  33h ; 3
.data:000000000060106D                 db  17h
.data:000000000060106E                 db  0Bh
.data:000000000060106F                 db  26h ; &
.data:0000000000601070                 db  3Ah ; :
.data:0000000000601071                 db    4
.data:0000000000601072                 db  54h ; T
.data:0000000000601073                 db  4Ch ; L
.data:0000000000601074                 db    4
.data:0000000000601075                 db  7Dh ; }
.data:0000000000601076                 db    0
.data:0000000000601076 _data           ends
.data:0000000000601076
```


（tips：get()使得字符串最后一位会是0）

根据异或运算的规律：若A^B=C，则B^C=A。

循环最后一步时，设s[i]=A，s[i+1]=B，则s[i]^s[i+1]=C=check[s.length]=0x00;——A^B=C=0x00；

令i是s.length-1，现在逆求s[i]：则check[i+1]=0x00=s[i+1]=C，s[i]=check[i]=0x7D，现在求A——即B^C=0x7D。

据此规律，写出解密脚本：

```
check = [0x0a,0x0d,0x06,0x1c,0x03,0x17,0x1d,0x2d,
         0x36,0x1a,0x2c,0x09,0x33,0x17,0x0b,0x26,
         0x3a,0x04,0x54,0x4c,0x04,0x7d,0x00]
i=len(check)-2
while i>=0:
    check[i]=check[i]^check[i+1]
    i-=1;

i=0;
while i<len(check)-1:
    print(chr(check[i]),end='')
    i+=1
```


运行得到flag……

```
xor
C:\Users\cxh-huo\Desktop\大学\pythonProject\venv\Scripts\python.exe
flag{xor_is_Very_ea5y}
Process finished with exit code 0
```