

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](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/m0_54933823/article/details/116616975

版权



[CTF日常练习OvO](#) 专栏收录该内容

1 篇文章 0 订阅

订阅专栏

2021.5.11 CTF Reverse 两题.....

ConsoleApplication4_3——Writeup

拿到exe文件, 打开, 研究一波:

```
-----/-----△
-----/-----○
-----/-----◇
-----/-----□
-----/-----☆
-----/-----▽
-----/----- (▽▽) /
-----/----- (;° Π° )
-----/-----

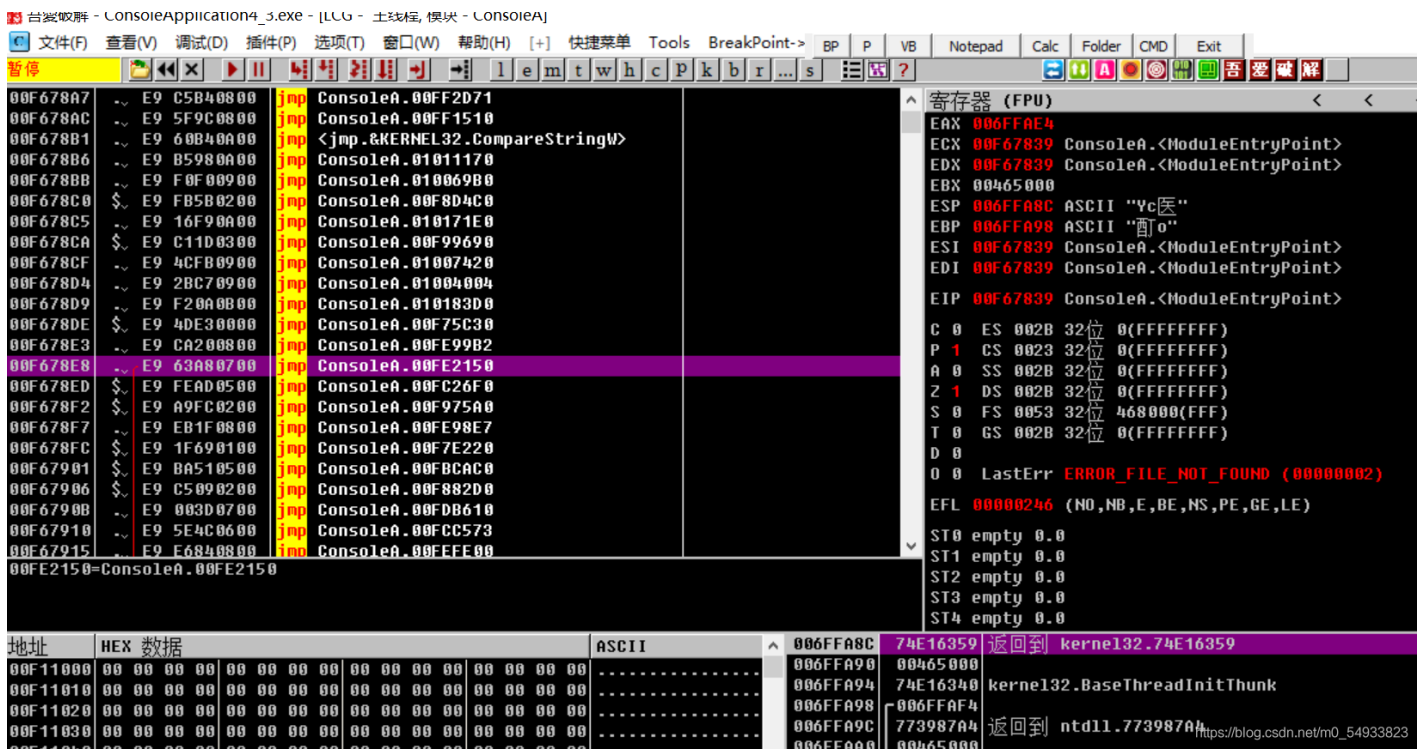
by 0x61

-----

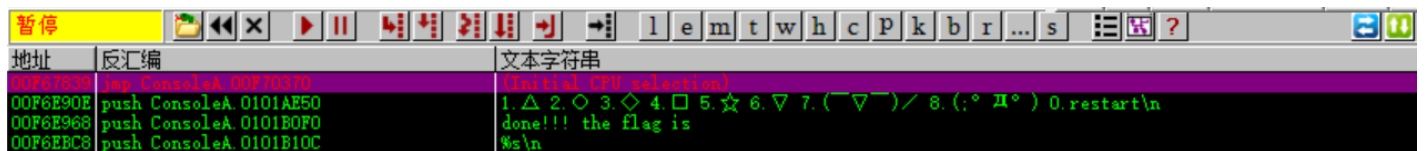
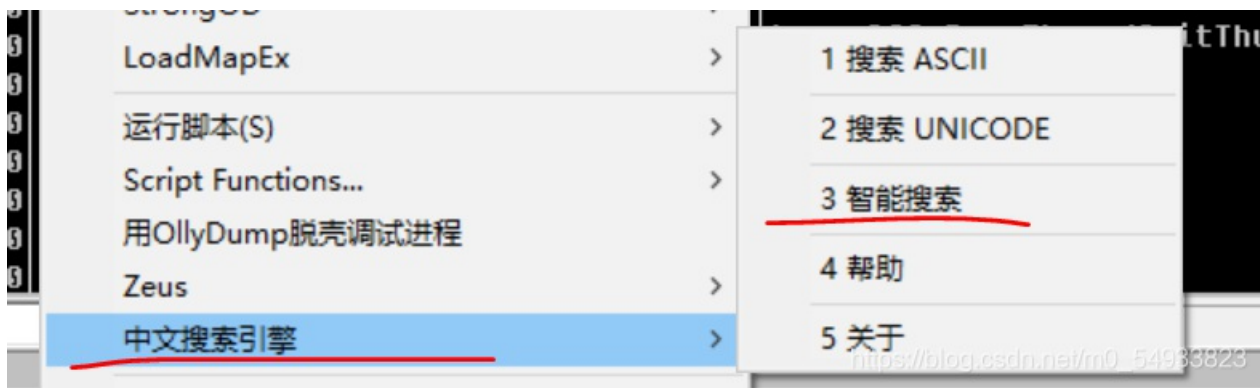
Play a game
The n is the serial number of the lamp, and m is the state of the lamp
If m of the Nth lamp is 1, it's on, if not it's off
At first all the lights were closed
Now you can input n to change its state
But you should pay attention to one thing, if you change the state of the Nth lamp, the state of (N-1)th and (N+1)th will
be changed too
When all lamps are on, flag will appear
Now, input n
input n, n(1-8)
1. △ 2. ○ 3. ◇ 4. □ 5. ☆ 6. ▽ 7. (▽▽) / 8. (;° Π° ) 0.restart
n=
```

https://blog.csdn.net/m0_54933823

发现是要我们输入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.....

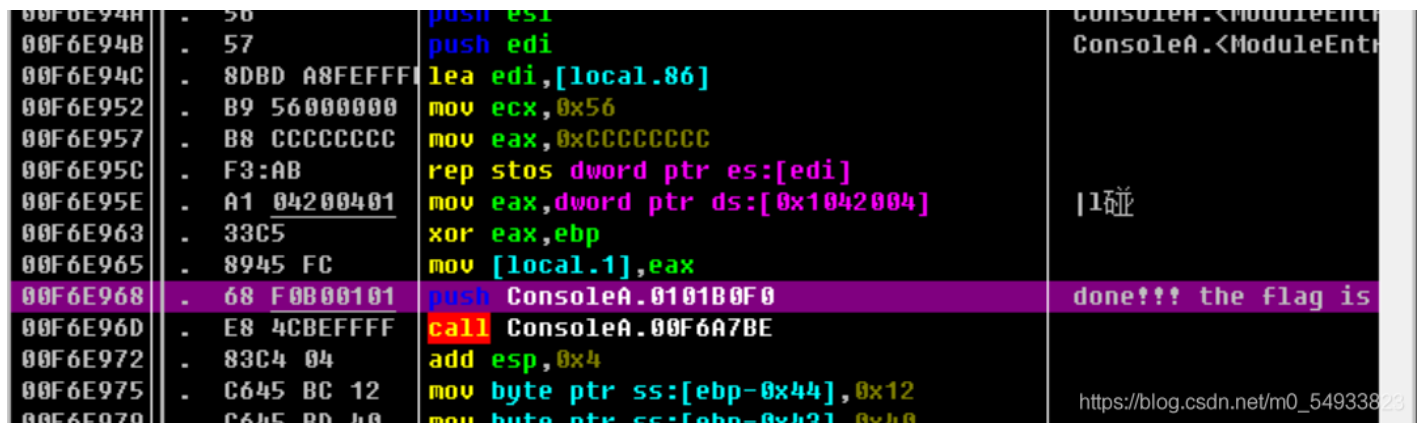


emmm。。翻看了一会儿，好复杂的样子。可以尝试用字符串搜索一些关键字的位置。



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:

```

00F67AAF ~ E9 2C680600 jmp ConsoleA.00FCE2E0
00F67AB4 $ E9 876E0000 jmp ConsoleA.00F6E940
00F67AB9 ~ E9 C26B0700 jmp ConsoleA.00FDE680
00F67ABE $ E9 DD490400 jmp ConsoleA.00FAC4A0
00F67AC3 ~ E9 F8170600 jmp ConsoleA.00FC92C0
00F67AC8 ~ E9 A2180800 jmp ConsoleA.00FE936F
00F67ACD $ E9 4EC70200 jmp ConsoleA.00F94220
00F67AD2 $ E9 19C10000 jmp ConsoleA.00F73BF0
00F67AD7 $ E9 14B50000 jmp ConsoleA.00F72FF0
00F67ADC $ E9 8F990300 jmp ConsoleA.00FA1470
00F67AE1 $ E9 1A160200 jmp ConsoleA.00F89100
00F67AE6 ~ E9 65260900 jmp ConsoleA.00FFA150
00F67AEB $ E9 80F40200 jmp ConsoleA.00F96F70
00F6E940=ConsoleA.00F6E940
本地调用来自 00F6F66C

```

https://blog.csdn.net/m0_54933823

发现这里依旧用jmp指令跳转, 所以继续回溯:

```

00F6F656 ~ 75 19 jnz short ConsoleA.00F6F671
00F6F658 . B8 01000000 mov eax,0x1
00F6F65D . 6BC8 07 imul ecx,eax,0x7
00F6F660 . 0FB691 282E04 movzx edx,byte ptr ds:[ecx+0x1042E28]
00F6F667 . 83FA 01 cmp edx,0x1
00F6F66A ~ 75 05 jnz short ConsoleA.00F6F671
00F6F66C . E8 4384FFFF call ConsoleA.00F67AB4
00F6F671 >^ E9 85FEFFFF jmp ConsoleA.00F6F4FB
00F6F676 > 33C0 xor eax,eax
00F6F678 . 52 push edx
00F6F679 . 8BCD mov ecx,ebp
00F6F67B . 50 push eax

```

ConsoleA.<Module...
https://blog.csdn.net/m0_54933823

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:

```

00F6F5AF . E8 2281FFFF call ConsoleA.00F676D6
00F6F5B4 . 83C4 04 add esp,0x4
00F6F5B7 > 68 BCB50101 push ConsoleA.0101B5BC
00F6F5BC . E8 F68BFFFF call ConsoleA.00F681B7
00F6F5C1 . 83C4 04 add esp,0x4
00F6F5C4 . E8 8B8AFFFF call ConsoleA.00F68054
00F6F5C9 . B8 01000000 mov eax,0x1
00F6F5CE . 6BC8 00 imul ecx,eax,0x0
00F6F5D1 . 0FB691 282E04 movzx edx,byte ptr ds:[ecx+0x1042E28]
00F6F5D8 . 83FA 01 cmp edx,0x1
00F6F5DB ~ 0F85 90000000 jnz ConsoleA.00F6F671
00F6F5E1 . B8 01000000 mov eax,0x1

```

CLS
https://blog.csdn.net/m0_54933823

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.....

00F6F65D	. 6BC8 07	imul ecx,eax,0x7
00F6F660	. 0FB691 282E04	movzx edx,byte ptr ds:[ecx+0x1042E28]
00F6F667	. 83FA 01	cmp edx,0x1
00F6F66A	~ 75 05	jnz short ConsoleA.00F6F671
00F6F66C	. E8 4384FFFF	call ConsoleA.00F67AB4
00F6F671	> E9 85FEFFFF	jmp ConsoleA.00F6F4FB
00F6F676	> 33C0	xor eax,eax

芜湖，flag出来了?!（此时开关并未全部闭合，但flag照样出来了.....）

```

F
F
-----
Fdone!!! the flag is zscft{T9is_t0pic_1s_v5ry_int7resting_b6t_others_are_n0t}
Finput n,n(1-8)
F1.△ 2.○ 3.◇ 4.□ 5.☆ 6.▽ 7.(▽▽) / 8.(;° ∏°) 0.restart
Fn=
F

```

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 }

```

https://blog.csdn.net/m0_54933823

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: $s[i] \neq \text{check}[i]$, 之前s还进行了自身邻位的异或运算: $s[i] \wedge s[i+1]$; 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 $\text{check}[i]$ 。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
.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 _data db 0
.data:0000000000601076 ends

```

https://blog.csdn.net/m0_54933823

(tips: get()使得字符串最后一位会是0)

根据异或运算的规律: 若 $A \oplus B = C$, 则 $B \oplus C = A$ 。

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

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 $B \oplus C = 0x7D$ 。

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

```

https://blog.csdn.net/m0_54933823

运行得到flag.....

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



[创作打卡挑战赛](#) >
[赢取流量/现金/CSDN周边激励大奖](#)