

XCTF easyhook

原创

YenKoc 于 2020-04-20 14:57:32 发布 367 收藏

分类专栏: [XCTF](#)

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/YenKoc/article/details/105631031>

版权



[XCTF 专栏收录该内容](#)

26 篇文章 2 订阅

订阅专栏

这题感觉主要是考一个动调, 学会要观察堆栈和寄存器, 以此来找到加密的函数
载入ida, 开始动调。

```
; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near
NumberOfBytesWritten= dword ptr -24h
Buffer= byte ptr -20h
argc= dword ptr 4
argv= dword ptr 8
envp= dword ptr 0Ch

sub    esp, 24h
push  edi
push  offset aPleaseInputFla ; "Please input flag:"
call  sub_401370
lea   eax, [esp+2Ch+Buffer]
push  eax
push  offset a31s ; "%31s"
call  _scanf
lea   edi, [esp+34h+Buffer]
or    ecx, 0FFFFFFFh
xor   eax, eax
add   esp, 0Ch
repne scasb
not   ecx
dec   ecx
pop   edi
cmp   ecx, 13h
jz    short loc_4012F1

push  offset aWrong ; "Wrong!\n"
call  sub_401370
push  offset aPause ; "pause"
call  _system

loc_4012F1:
call  sub_401220
push  0 ; hTemplateFile
```

先F8开始, 看看经过哪里, 字符串就被加密了,

```
push  eax ; hFile
call  ds:WriteFile
lea   eax, [esp+24h+NumberOfBytesWritten]
lea   ecx, [esp+24h+Buffer]
push  eax
push  ecx
```

F7单步进去看看。

```
sub_401080 proc near
hFile= dword ptr 4
lpBuffer= dword ptr 8
nNumberOfBytesToWrite= dword ptr 0Ch
lpNumberOfBytesWritten= dword ptr 10h
lpOverlapped= dword ptr 14h
```

```

push ebx
push ebp
push esi
mov esi, [esp+0Ch+nNumberOfBytesToWrite]
push edi
mov edi, [esp+10h+lpBuffer]
push esi
push edi
call sub_401000
add esp, 8
mov ebx, eax
call sub_401140
mov eax, [esp+10h+lpOverlapped]
mov ebp, [esp+10h+lpNumberOfBytesWritten]
mov ecx, [esp+10h+hFile]
push eax ; lpOverlapped
push ebp ; lpNumberOfBytesWritten
push esi ; nNumberOfBytesToWrite
push edi ; lpBuffer
push ecx ; hFile
call ds:WriteFile
test ebx, ebx
jz short loc_4010BF

```

那个标红的地方就是加密函数。

```

1 signed int __cdecl sub_401000(int a1, signed int a2)
2 {
3     char v2; // a1
4     char v3; // b1
5     char v4; // c1
6     int v5; // eax
7
8     v2 = 0;
9     if ( a2 > 0 )
10    {
11        do
12        {
13            if ( v2 == 18 )
14            {
15                *(_BYTE *)(a1 + 18) ^= 0x13u;
16            }
17            else
18            {
19                if ( v2 % 2 )
20                    v3 = *(_BYTE *)(v2 + a1) - v2;
21                else
22                    v3 = *(_BYTE *)(v2 + a1 + 2);
23                *(_BYTE *)(v2 + a1) = v2 ^ v3;
24            }
25            ++v2;
26        }
27        while ( v2 < a2 );
28    }
29    v4 = 0;
30    if ( a2 <= 0 )
31        return 1;
32    v5 = 0;
33    while ( byte_40A030[v5] == *(_BYTE *)(v5 + a1) )
34    {
35        v5 = ++v4;
36        if ( v4 >= a2 )

```

然后跳回去的话，有个地方很坑，后面判断长度为21。。。一开始就已经说明了是19的长度，又看到那个字符串是 this_is_not_flag，大概就猜到是个幌子了。。。无视它，直接写脚本跑

```

res=[
    0x61,0x6A, 0x79, 0x67, 0x6B, 0x46, 0x6D, 0x2E, 0x7F, 0x5F, 0x7E,0x2D, 0x53, 0x56, 0x7B, 0x38, 0x6D, 0x4C, 0x
6E, 0x00
]
#奇数时 (input[i]-i)^i
#偶数 (input[i+2])^i
flag=list("1234567891234567890")
for i in range(0,18):
    if i%2==1:
        flag[i]=chr((res[i]^i)+i)
    else:
        flag[i+2]=chr(res[i]^i)
print("".join(flag))
print(res[18]^0x13)

```