

ctf php黑魔法,社团的CTF逆向题WriteUp

转载

唯真纯 于 2021-03-11 03:35:03 发布 61 收藏 1

文章标签: [ctf php黑魔法](#)

最近社团弄了CTF比赛,然后我就帮忙写了逆向的题目,这里写一下WriteUp,题目和源码在附件中给出

一个简单的逆向: one_jump_to_flag.exe

这题算是签到题,直接OD智能搜索就完事了, flag{Welcome_to_GeekFZCTF}

```
true
flag{Welcome_to_GeekFZCTF}
```

一个简单的对比: check.exe

这一道也算是送分题,可以用IDA直接分析可以看出来,这里就是读取输入的字符串,然后逐个进行对比,这里可以直接用IDA转化为char型就可以看出. flag{sAdf_fDfkl_Fdf}

```
if ( *flag != 102 )
    goto LABEL_22;
if ( flag[1] == 108
    && flag[2] == 97
    && flag[3] == 103
    && flag[4] == 123
    && flag[5] == 115
    && flag[6] == 65
    && flag[7] == 100
    && flag[8] == 102
    && flag[9] == 95
    && flag[10] == 102
    && flag[11] == 68
    && flag[12] == 102
    && flag[13] == 107
    && flag[14] == 108
    && flag[15] == 95
    && flag[16] == 70
    && flag[17] == 100
    && flag[18] == 102
    && flag[19] == 125 )
{
    _printf("yes,this is a flag");
}
```

```
if ( *flag != 'f' )
    goto LABEL_22;
if ( flag[1] == 'l'
    && flag[2] == 'a'
    && flag[3] == 'g'
    && flag[4] == '{'
    && flag[5] == 's'
    && flag[6] == 'A'
    && flag[7] == 'd'
    && flag[8] == 'f'
    && flag[9] == '_'
    && flag[10] == 'f'
    && flag[11] == 'D'
    && flag[12] == 'f'
    && flag[13] == 'k'
    && flag[14] == 'l'
    && flag[15] == '_'
    && flag[16] == 'F'
    && flag[17] == 'd'
    && flag[18] == 'f'
    && flag[19] == '}' )
{
    _printf("yes,this is a flag");
}
```

猜猜那个是flag: whichisflag.exe

首先加载进IDA, 先看一下基本逻辑, 输入的flag首先要小于25, 然后进行判断进行判断, 如果flag[5]=Y, 且flag[8] = flag[11], flag[16] = flag[18], 上面的都实现了就执行which_is_flag这个函数, 我们跟进去看一下。

```
6  _printf(aFlag);
7  _gets_s(flag, 0x104u);
8  if ( strlen(flag) < 25 )
9  {
10 if ( flag[5] != 'Y' || flag[8] != flag[11] || flag[16] != flag[18] )
11     _printf("flag is wrong");
12     else
13         which_is_flag(flag);
14     _getchar();
15     result = 0;
16 }
17 else
18 {
19     _printf("flag is too long");
20     result = 0;
21 }
22 return result;
```

可以看到这是通过flag[5]进行switch跳转, 然后我们之前有个判断条件flag[5]=Y, 所以这里的跳转就是到89处

```
6  v1 = flag[5];
7  _printf("yes, this is a flag:\n");
8  switch ( v1 )
9  {
10     case 7:
11         _printf("flag{ieog_dfet_ggfd_gfg}");
12         result = 0;
13         break;
14     case 8:
15         _printf("flag{wRrS3vSuDjYH0uUEkd}");
16         result = 0;
17         break;
18     case 9:
19         _printf("flag{dueYi3ZkvF5CX2g1qL}");
20         result = 0;
21     case 89:
22         _printf("flag{WWtFa19kamtmM19qRWpfZUUu}");
```

但是我们可以看到, 这个flag的长度是大于25的, 然后仔细观察一下flag里的内容, 可以看出是base64编码过的, 我们找个网站进行解码就可以, 这里的flag是用python随机出来的, 代码也在附件里。

flag{YkEj_djKf3_jEj_eUn}

明文:		BASE64:
<input type="text" value="YkEj_djKf3_jEj_eUn"/>	<input type="button" value="BASE64编码 >"/>	<input type="text" value="WWtFa19kamtmM19qRWpfZVVu"/>
	<input type="button" value="< BASE64解码"/>	

简单的反调试: fts.exe

其实这道题目就只是用OD调试时会遇到一点反调试, 如果用IDA直接分析逻辑就可以了, 这里首先用IDA分析先。

```

_printf(asc_4021EC);
*( _DWORD *)flag_2 = 0x76707D77;
flag_2[4] = 0x6A;
v3 = 0;
do
{
    flag_2[v3] ^= 0x11u;
    _printf("%c", flag_2[v3++]);
}
while ( v3 < 5 );
v4 = FindWindowW(0, L"吾爱破解[LC6].exe");
v5 = FindWindowW(0, L"ida.exe");
if ( v4 && v5 )
{
    *( _DWORD *)flag_2 = 0x454C4D41;
    flag_2[4] = 0x50;
    v6 = 0;
    do
    {
        flag_2[v6] ^= 0x22u;
        _printf("%c", flag_2[v6++]);
    }
    while ( v6 < 5 );
    CheckDebug();
    fts_Rdtsc();
    the_end();
}

```

可以看到一个可疑的循环并printf出来字符，我们看到逻辑就是用flag_2这个数组的数据与0x11进行异或，我们手动计算一下得出字符串(这里注意的是0x76707D77是小端序，我们要从77开始异或): flag{

```

*( _DWORD *)flag_2 = 0x76707D77;
flag_2[4] = 0x6A;
v3 = 0;
do
{
    flag_2[v3] ^= 0x11u;
    _printf("%c", flag_2[v3++]);
}
while ( v3 < 5 );

```

然后我们可以继续往下面找关键循环，函数CheckDebug、fts_Rdtsc、the_end里都有循环输出，然后都解密出来就是: congratulations_for_you}

```

*( _DWORD *)flag_2 = 0x454C4D41;
flag_2[4] = 0x50;
v6 = 0;
do
{
    flag_2[v6] ^= 0x22u;
    _printf("%c", flag_2[v6++]);
}
while ( v6 < 5 );
CheckDebug();
fts_Rdtsc();
the_end();

```

```

*( _DWORD *)flag_3 = 0x5F464752;
flag_3[4] = 0x52;
v3 = 0;
do
{
    flag_3[v3] ^= 0x33u;
    result = _printf("%c", flag_3[v3++]);
}
while ( v3 < 5 );

```

```

*( _DWORD * )flag_4 = 707472688;
flag_4[4] = 55;
v3 = 0;
do
{
    flag_4[v3] ^= 0x44u;
    _printf("%c", flag_4[v3++]);
}
while ( v3 < 5 );

```

```

*( _DWORD * )flag_5 = 658125578;
*( _DWORD * )&flag_5[4] = 540683274;
flag_5[8] = 40;
v0 = 0;
do
{
    flag_5[v0] ^= 0x55u;
    _printf("%c", flag_5[v0++]); |
}
while ( v0 < 9 );

```

所以flag就是: flag{congratulations_for_you}

然后用OD动态调试一下,一开始就是先进行进程的检索,判断是否存在IDA或者OD,然后我们可以通过je直接跳过进行判断的地方。

0085 D0FDFFF	lea eax,dword ptr ss:[ebp-0x200]	
50	push eax	
57	push edi	fts.01390000
C785 D0FDFFF	mov dword ptr ss:[ebp-0x230],0x22C	
FF15 10203901	call dword ptr ds:[<&KERNEL32.Process32FirstW	kernel32.Process32FirstW
85C0	test eax,eax	
74 4E	je short fts.013910AA	
50	push ebx	
8B1D 14203901	mov ebx,dword ptr ds:[<&KERNEL32.Process32NextW	kernel32.Process32NextW
56	push esi	
8B35 B4203901	mov esi,dword ptr ds:[<&MSUCR110._wcsicm	msucr110._wcsicm
8D9B 00000000	lea ebx,dword ptr ds:[ebx]	
8D85 F4FDFFF	lea eax,dword ptr ss:[ebp-0x20C]	
68 40213901	push fts.01392140	吾爱破解[LCG].exe
50	push eax	
FFD6	call esi	
83C4 08	add esp,0x8	
85C0	test eax,eax	
74 4C	je short fts.013910D1	
8D85 F4FDFFF	lea eax,dword ptr ss:[ebp-0x20C]	
68 68213901	push fts.01392168	ida.exe
50	push eax	
FFD6	call esi	
83C4 08	add esp,0x8	

跳过第一个反调试的地方后就得看一下我们的main函数入口在哪里(为什么不先找main函数,因为你不跳过第一个反调试即使找到main函数入口也没用,尝试一下就发现跳过了第一个反调试就很容易去到main函数了),我们这里先智能搜索一下可以看到一些字符串,你可以每个都进去看一下,然后发现main函数就在走过八十一难哪里(用IDA可以直接找到)

```

吾爱破解[LCG].exe
ida.exe
face the IDA
face the od
ntdll.dll
NtQueryInformationProcess
%c
indebug\n
%c
\n
最后一步了,说一下感想!!!\n
%c
(Initial CPU selection)
走过八十一难,flag就在眼前\n
%c
吾爱破解[LCG].exe
ida.exe
%c
onFilt;\r

```

找到main函数后下断点,然后F9跳过去就可以单步跟踪了,我们可以发现会自己打印出了第一个解密循环

```
走过八十一难, flag就在眼前
flag{
```

下面继续跟会发现有用到FindWindow函数去查找有没有IDA和OD, 然后我们只要不让下面两个je进行跳转就可以, 就会再打印出一段flag出来

```
push edi
push fts.01392140 吾爱破解[LCG].exe
push 0x0
call esi user32.FindWindowW
push fts.01392168 ida.exe
push 0x0
mov edi,eax
call esi user32.FindWindowW
test edi,edi
pop edi
je short fts.0139137A
test eax,eax
je short fts.0139137A
mov dword ptr ss:[ebp-0xC],0x454C4D41
mov byte ptr ss:[ebp-0x8],0x50
xor esi,esi
lea esp,dword ptr ss:[esp]
xor byte ptr ss:[ebp+esi-0xC],0x22
movsx eax,byte ptr ss:[ebp+esi-0xC]
```

```
走过八十一难, flag就在眼前
flag{congr
```

跟进去函数里面, 可以发现用了ZwQueryInformationProcess进行检测调试端口的, 我们不让它跳转就会打印出字符串来

```
E8 80FDFFFF call fts.CheckDebug_init_cookieFilterter
```

```
50 push eax
FFD6 call esi ntdll.ZwQueryInformationProcess
837D F0 00 cmp dword ptr ss:[ebp-0x10],0x0
75 40 jnz short fts.0139117B
8B3D B0203901 mov edi,dword ptr ds:[<&MSUCR110.printf>] msvcrt110.printf
C745 F4 524746 mov dword ptr ss:[ebp-0xC],0x5F464752
C645 F8 52 mov byte ptr ss:[ebp-0x8],0x52
33F6 xor esi,esi ntdll.ZwQueryInformationProcess
8BEE mov edi,edi
```

```
走过八十一难, flag就在眼前
flag{congratula
```

再进入下一个函数中, 在里面可以看到rdtsc命令, 这个用于把时间保存的指令, 这里是基于时间的反调试, 我们可以直接在下面的popad下断点然后F9直接跳过, 然后就找到了解密循环。

```
E8 2BF0FFFF call fts.fts_RdtscnitupBasexceptionFilter
```

```
60 pushad
0F31 rdtsc
52 push edx
```

```
走过八十一难, flag就在眼前
flag{congratulations
```

最后进入函数，我们可以看到有一个get函数并且下面的跳转是关键，可以先随机输入一些东西，然后可以看到下面的跳转跳过了我们的printf函数，所以我们让它不跳转，然后就跑出最后的flag出来。

```
E8 B6FEFFFF call fts.the_endtUnhandedExceptionFilter
```

```

50      push eax
FF15 AC203901 call dword ptr ds:[&MSUCR110.get_s_s]  msvcrl10.get_s_s
8D8D ECFEFFFF lea ecx,dword ptr ss:[ebp-0x114]
83C4 10      add esp,0x10
8D51 01      lea edx,dword ptr ds:[ecx+0x1]
8A01      mov al,byte ptr ds:[ecx]
41      inc ecx
84C0      test al,al
75 F9      jnz short fts.01391276
2BCA      sub ecx,edx
83F9 1E      cmp ecx,0x1E
76 38      jbe short fts.013912BC

```

```

83F9 1E      cmp ecx,0x1E
76 38      jbe short fts.013912BC
56      push esi
C745 F0 0A333A mov dword ptr ss:[ebp-0x10],0x273A330A
C745 F4 0A2C3A mov dword ptr ss:[ebp-0xC],0x203A2C0A
C645 F8 28      mov byte ptr ss:[ebp-0x8],0x28
33F6      xor esi,esi
8DA424 00000000 lea esp,dword ptr ss:[esp]
807435 F0 55      xor byte ptr ss:[ebp+esi-0x10],0x55
0FBF4435 F0      movsx eax,byte ptr ss:[ebp+esi-0x10]
50      push eax
68 B8213901      push fts.013921B8
FFD7      call edi
46      inc esi
83C4 08      add esp,0x8
83FE 09      cmp esi,0x9
7C E5      jl short fts.013912A0
5E      pop esi
8B4D FC      mov ecx,dword ptr ss:[ebp-0x4]
33CD      xor ecx,ebp

```

```

走过八十一难，flag就在眼前
flag {congratulations
最后一步了，说一下感想！！
122
_for_you}

```

简单的算法：suafa.exe

这是一个算法，然后这里要道个歉是出题不够严谨，在跑flag的时候会出现多解的情况。正向的核心算法是输入的字符串减去key之后对4求余，然后在key1到key4之间选择字符，最后与我们给定的字符进行对比

```

1 char key1[65] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/-";
2 char key2[65] = "+/abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
3 char key3[65] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/-";
4 char key4[65] = "0123456789+/ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
5

```

```

6 char key[27] = "QASWZXDECVFRBNGTHYJUMKIOLP";
7 int len = strlen(flag);
8 for(int i=0; i= 33)
11 {
12 if(flag[i]-key[i] > 0)
13 {
14 int the_key = (flag[i]-key[i]) % 4;
15 switch(the_key)
16 {
17 case 0:
18 flag[i] = key1[flag[i]-key[i]];
19 continue;
20 case 1:
21 flag[i] = key2[flag[i]-key[i]];
22 continue;
23 case 2:
24 flag[i] = key3[flag[i]-key[i]];
25 continue;
26 case 3:
27 flag[i] = key4[flag[i]-key[i]];
28 continue;
29 default:
30 continue;
31 }
32 }
33 else

```

```

34 {
35 printf("flag is wrong");
36 ExitProcess(0);
37 }
38 }
39 }

```

然后在逆向的时候我们用IDA打开基本都能编译出来，然后有一小问题就是v5 = v4 - v1[key-flag]这里，反编译后与原来的代码int the_key = (flag[i]-key[i])有点差别，不过如果看汇编的代码可以更好的理解

```

v1 = flag;
strcpy(key1, "ABCDEFGHJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/");
strcpy(key2, "+/abcdefghijklmnopqrstuvwxy0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ");
strcpy(key3, "abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/");
strcpy(key4, "0123456789+/ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy");
*( _QWORD *)key = *( _QWORD *)aQaswzxde;
*( _QWORD *)&key[8] = qword_40224C;
*( _WORD *)&key[24] = 20556;
*( _QWORD *)&key[16] = qword_402254;
key[26] = 0;
v2 = strlen(flag);
v3 = 0;
if ( v2 > 0 )
{
do
{
v4 = *v1;
if ( *v1 <= 126 && v4 >= 33 )
{
v5 = v4 - v1[key - flag];
if ( v5 <= 0 )
{
_printf("flag is wrong");
ExitProcess(0);
}
}
}
}

```

其中sub eax, flag就是我们的v4 - v1[key-flag]，而v4=flag[i]，所以理论上flag = v1[key-flag]，而我们汇编中flag的值为byte ptr [ebx+edx]，这里的ebx可以从前面查看就是key，而edx就相当于索引。不懂得可以自己仔细看一下。

```

movsx  flag, byte ptr [ebx+edx]
movsx  eax, al
sub    eax, flag
test   eax, eax
jle    short loc_401217

```

然后分析完就可以写个脚本自己跑出来：flag{this_is_a_easy_suanfa}

```

1 #include "stdio.h"
2 #include "Windows.h"
3
4 intmain()5 {6 char key1[] =
"ABCDEFGHJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";7 char key2[] =
"+/abcdefghijklmnopqrstuvwxy0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";8 char key3[] =
"abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/";9 char key4[] =
"0123456789+/ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy";10
11 char key[] = "QASWZXDECVFRBNGTHYJUMKIOLP";12
13 char flag_1[] = "tfoQ5cckwhX51HYpxAjkMQYTAp5";14
15 int yushu = 0;16
17 for(int m=0; m<=26; m++)18 {19 for(int n=33; n<=126; n++)20 {21 yushu = n - int(key[m]);22 yushu %=
4;23 switch(yushu)24 {25 case 0:26 if(key1[n - int(key[m])] ==flag_1[m])27 {28 printf("%c", n);29 }30
continue;31 case 1:32 if(key2[n - int(key[m])] ==flag_1[m])33 {34 printf("%c", n);35 }36 continue;37 case 2:38
if(key3[n - int(key[m])] ==flag_1[m])39 {40 printf("%c", n);41 }42 continue;43 case 3:44 if(key4[n - int(key[m])]
==flag_1[m])45 {46 printf("%c", n);47 }48 continue;49 default:50 continue;51 }52 }53 printf("\n");54 }55 }

```

因为存在多解，所以跑出来得结果是样子的

```
f
l
a{
g
{
Zt
Nh
Oi
Ys
-
i
s
-
a{
_y
e
a
Ys
y
_y
Ys
[u
a{
n
Lf
a
!d)
```