黑龙江省网络安全技能大赛 初赛 RE300 writeup ——Lilac 逆向组

原创

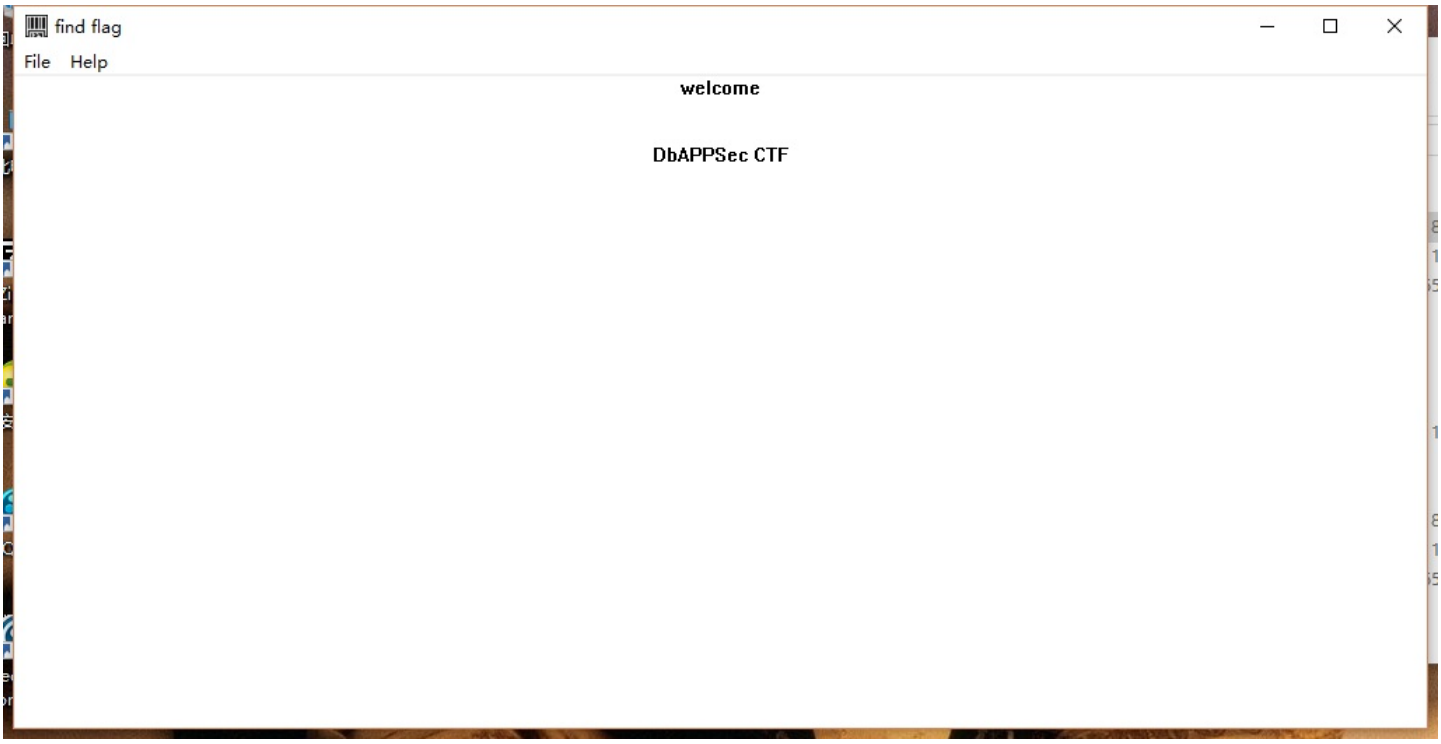
Vccxx 于 2016-10-03 18:58:45 发布 2153 收藏

分类专栏： CTF 文章标签： CTF-逆向

本文链接：https://blog.csdn.net/qq_29947311/article/details/52729041

一个exe文件，下载地址：https://yunpan.cn/cvgXYFXCpf39p 访问密码 edfd

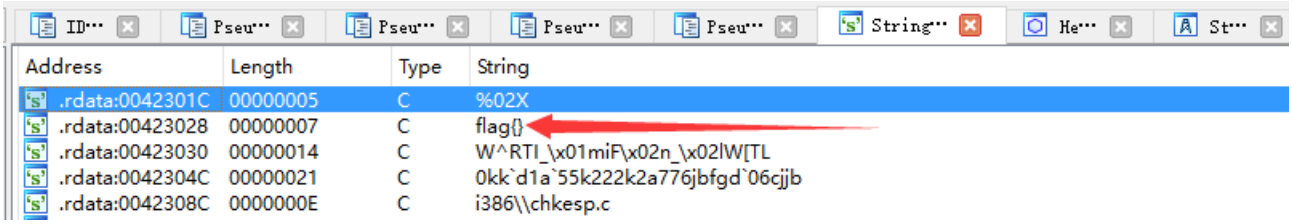
这个pe一打开是这样的:



与常规逆向不同的是，此题没有输入的地方，也没有提交flag的地方

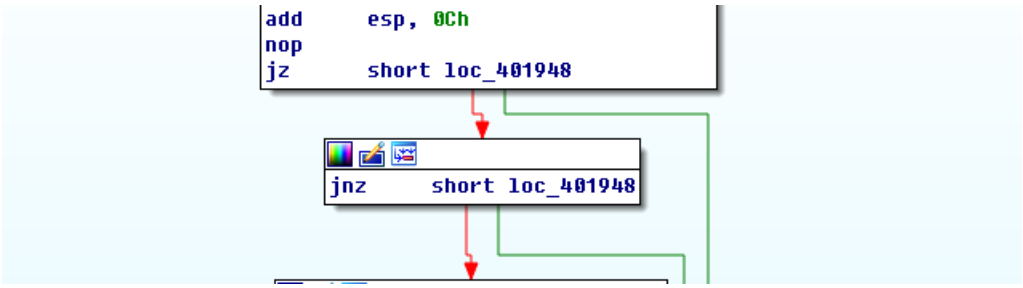
注意到上面有个菜单栏，然而里面并没有什么有用的东西

于是程序丢到ida里面，经验让我搜索了一下程序中的字符串，找到了有趣的东西：

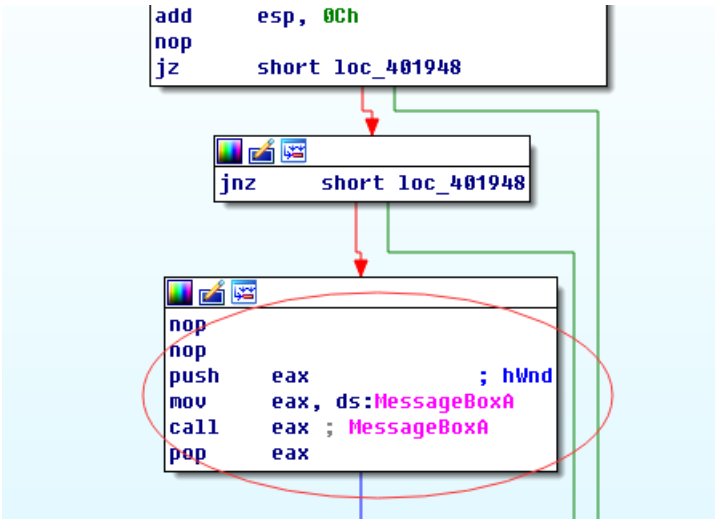


跟过去，到了一个函数里，这个函数是被混淆的，混淆点我比赛时已经处理了一下，但是处理得不够，所以ida还是不能很好的

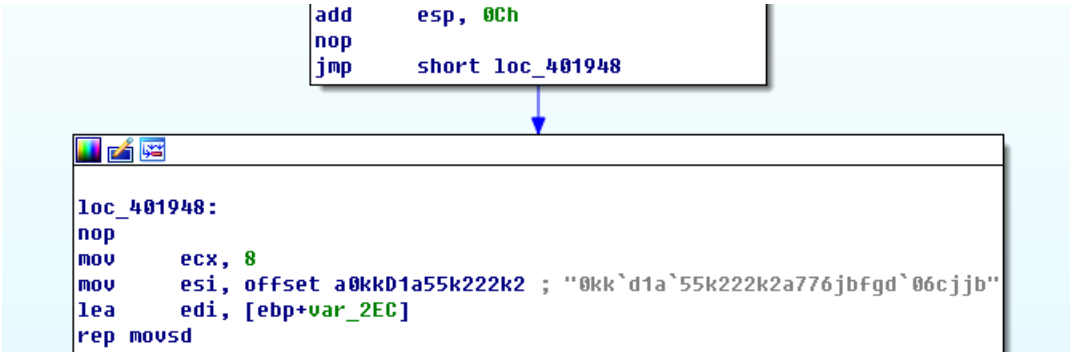
跟过去，到了一个函数里，这个函数是假混淆的，混淆点我比赛时已经处理了一下，但是处理得不够，所以ida还是不能很好的分析(原题我找不到了。。):
混淆点:



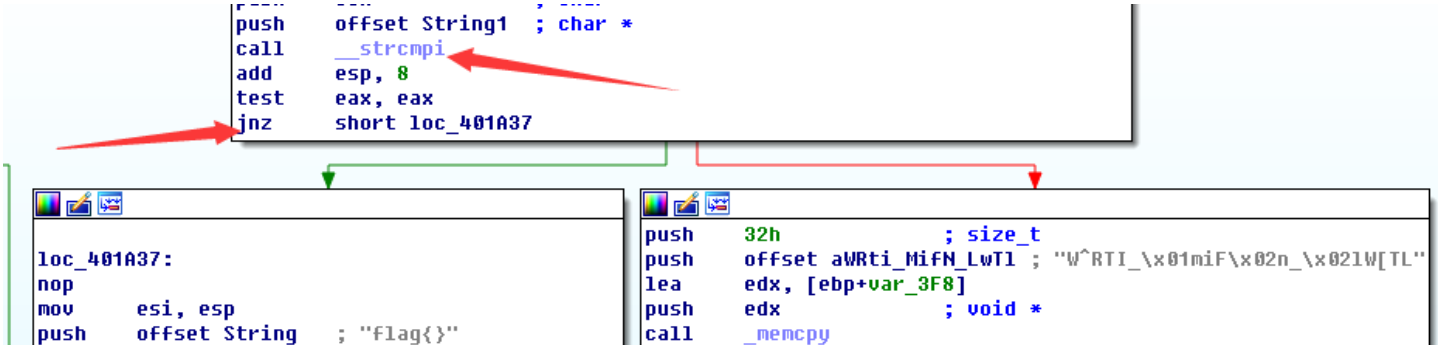
这两个跳转跳向同一个地址，相当于jump，而由于ida的分析原则是先分析使条件不成立的机器码，也就是下列这块代码是不可能执行的：



这就是混淆代码(代码块最前面的两个nop是我比赛时加的，当时没把这些都nop掉，导致ida还是无法正常分析)，把他们全都nop掉，jz和jnz合并成jmp，得到新的程序，再丢进ida，来到同一段代码:



这时按f5就可以完整分析了，来到一个关键的比较语句：



```
mov     edx, [ebp+hWndParent]
push    edx             ; hWnd
call    ds:SetWindowTextA
cmp     esi, esp
call    __chkesp
mov     esi, esp
push    0               ; uType
push    offset Caption  ; "^_^"
push    offset Text     ; "Are you kidding me?"
mov     eax, [ebp+hWndParent]
```

```
add     esp, 0Ch
lea     eax, [ebp+var_3F8]
push    eax             ; char *
call    _strlen
add     esp, 4
push    eax             ; int
lea     ecx, [ebp+var_3F8]
push    ecx             ; int
lea     edx, [ebp+var_1EC]
push    edx             ; lpString
```

显然，左边的代码块是失败的，右边的是成功的，让程序执行到右边我们就成功在望了，我们可以看到上面是一个strcmpi函数，这个函数比较的两个值如果相等，我们就能够跳到右边。
接下来看看这个函数在比较啥：

```
lea     ecx, [ebp+var_2EC]
push    ecx             ; char *
push    offset String1  ; char *
call    _strcmpi
```

比较的是string1和var_2ec，往上看看这两个值分别存的是什么,经过一番查找，终于弄清了：

```
mov     ecx, 8
mov     esi, offset a0kkD1a55k222k2 ; "0kk`d1a`55k222k2a776jbfgd`06cjjb"
lea     edi, [ebp+var_2EC]
rep movsd
movsb
```

上面这个代码块就是对var_2ec的赋值，可以看到这里的语句是用ecx配合rep movsd来将一个内置字串的值移到var_2ec所在的地址，至于string1的值，可以在F5窗口中看到:

```
38    {
39      if ( strlen((const char *)&String1) > 6 )
40        ExitProcess(0);
41      if ( strlen((const char *)&String1) )
42      {
43        memset(&v22, 0, 0x100u);
44        v6 = strlen((const char *)&String1);
45        memcpy(&v22, &String1, v6);
46        v7 = strlen((const char *)&String1);
47        sub_40101E(&String1, v7, (LPSTR)&String1);
48        qmemcpy(&v18, "0kk`d1a`55k222k2a776jbfgd`06cjjb", 0x21u);
49        memset(&v19, 0, 0xDCu);
50        v20 = 0;
51        v21 = 0;
52        strcpy(v14, "SS");
53        v15 = 0;
54        v16 = 0;
55        v17 = 0;
56        v8 = strlen(&v18);
57        sub_401005(v14, (int)&v18, v8);
58        if ( _strcmpi((const char *)&String1, &v18) )
```

Sub_40101e使用了这个string1的地址，可能是做了某些修改，于是跟进这个函数：

```
memset(&v4, 0xCCu, 0x64u);
if ( CryptAcquireContextA(&phProv, 0, 0, 1u, 0xF0000000) )
{
  if ( CryptCreateHash(phProv, 0x8003u, 0, 0, &phHash) )
  {
    if ( CryptHashData(phHash, pbData, dwDataLen, 0) )
    {
      CryptGetHashParam(phHash, 2u, v7, &pdwDataLen, 0);
      *lpString1 = 0;
      for ( i = 0; i < pdwDataLen; ++i )
      {
        wsprintfA(&String2, "%02X", v7[i]);
        lstrcatA(lpString1, &String2);
      }
      CryptDestroyHash(phHash);
```

```
    CryptReleaseContext(phProv, 0);
    result = 1;
  }
  else
  {
    CryptDestroyHash(phHash);
    CryptReleaseContext(phProv, 0);
    result = 0;
```

发现里面一堆api，google之后发现这个是微软的hash算法：

Syntax

C++

```
BOOL WINAPI CryptCreateHash(
  _In_  HCRYPTPROV hProv,
  _In_  ALG_ID     Algid,
  _In_  HCRYPTKEY  hKey,
  _In_  DWORD      dwFlags,
  _Out_ HCRYPTHASH *phHash
);
```

Parameters

hProv [in]

A handle to a CSP created by a call to **CryptAcquireContext**.

Algid [in]

An **ALG_ID** value that identifies the hash algorithm to use.

Valid values for this parameter vary, depending on the CSP that is used. For a list of default algorithms, see Remarks.

hKey [in]

If the type of hash algorithm is a keyed hash, such as the *Hash-Based Message Authentication Code* (HMAC) or *Message Authentication Code* (MAC) algorithm, the key for the hash is passed in this parameter. For nonkeyed algorithms, this parameter must be set to zero.

For keyed algorithms, the key must be to a *block cipher* key, such as RC2, that has a *cipher mode* of *Cipher Block Chaining* (CBC).

可以看到这个函数的第二个参数是加密所选用的算法：

```
`
  if ( CryptCreateHash(phProv, 0x8003u, 0, 0, &phHash) )
  {
```

继续google这个参数的含义：

CALG_MD5	0x00008003	MD5 hashing algorithm. This algorithm is supported by the Microsoft Base Cryptographic Provider.

原来是个md5。

所以这个string1是经过一个md5加密的，加密之后的值应该还是存在了string1里。

现在已经知道strcmpi比较的是一个md5值和一个内置字符串，但是这个内置字符串怎么看都不像md5(MD5只有字母和数字)，再仔细看程序，发现程序对这个内存值进行了亦或加密:

```
call    _strlen
add     esp, 4
push    eax             ; int
lea     edx, [ebp+var_2EC]
push    edx             ; int
lea     eax, [ebp+var_2F8]
push    eax             ; lpString
call    sub_401005
add     esp, 0Ch
```

```
    if ( i >= a3 )
      break;
    *(_BYTE *)(i + a2) ^= v6[i % v7];
  }
```

通过亦或解密出真正的md5：

```
>>> str =  "0kk`d1a`55k222k2a776jbfgd`06cjjb"
>>> for i in range(len(str)):
...     print chr(ord(str[i]) ^ 0x53),
...
c 8 8 3 7 b 2 3 f f 8 a a a 8 a 2 d d e 9 1 5 4 7 3 c e 0 9 9 1
>>> str1 = "c 8 8 3 7 b 2 3 f f 8 a a a 8 a 2 d d e 9 1 5 4 7 3 c e 0 9 9 1"
>>> l = str1.split(" ")
>>> r  = ""
>>> r.join(l)
'c8837b23ff8aaa8a2dde915473ce0991'
>>>
```

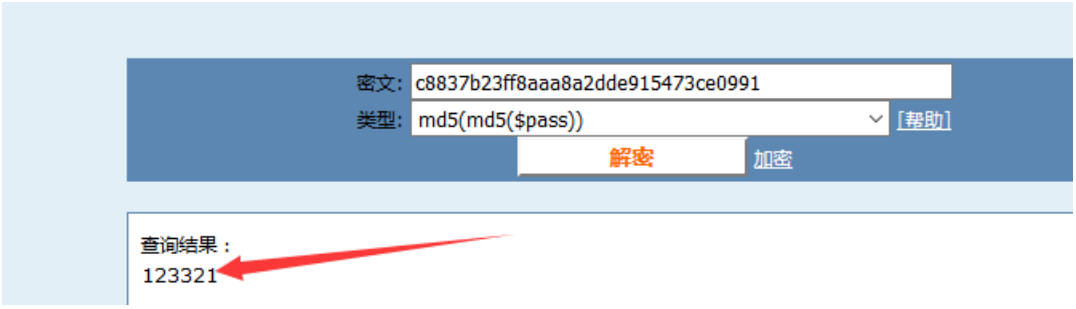
也就是说string1在md5加密后如果是这个值，程序就会跳到右边执行，现在我们看看右边的代码：

```
push    32h             ; size_t
push    offset aWRti_MifN_LwTl ; "W^RTI_\x01miF\x02n_\x02lW[TL"
lea     edx, [ebp+var_3F8]
push    edx             ; void *
call    _memcpy
add     esp, 0Ch
lea     eax, [ebp+var_3F8]
push    eax             ; char *
call    _strlen
add     esp, 4
push    eax             ; int
lea     ecx, [ebp+var_3F8]
push    ecx             ; int
lea     edx, [ebp+var_1EC]
push    edx             ; lpString
call    sub_401005
```

这里又做了一次和上面相同的亦或运算（高亮处），这次是把var_1ec和另一个内置字串亦或了，现在只要弄清楚var_1ec是什么就行了，在上方的代码块可以看到关于这个var_1ec变量的引用：

```
add     esp, 0Ch
push    offset String1  ; char *
call    _strlen
add     esp, 4
push    eax             ; size_t
push    offset String1  ; void *
lea     eax, [ebp+var_1EC]
push    eax             ; void *
call    _memcpy
add     esp, 0Ch
push    offset String1  ; lpString1
push    offset String1  ; char *
call    _strlen
add     esp, 4
push    eax             ; dwDataLen
push    offset String1  ; pbData
call    md5
add     esp, 0Ch
nop
jz      short loc_401948
```

原来这个变量就是加密以前的string1，既然这样，我们上网解密之前得到的md5值，就可以解出这个亦或的结果。Md5解码得到：



写脚本跑一下结果：

```
>>> str1 = "W^RTI_\x01miF\x02n_\x021W[TL"
>>> str2 = "123321"
>>> for i in range(len(str1)):
...     print chr(ord(str1[i])^ord(str2[i%len(str2)])),
...
f l a g { n 0 _ Z u 0 _ n 0 _ d i e }
```

Flag get

总结：

这一题在比赛中没有做出来，有如下原因：

1. 在反返混淆的时候没有把不可能执行的指令都nop掉，导致不能f5，方便分析

2. 一直纠结于怎么输入的问题，认为string1的值跟输入有关，于是捣鼓了几个小时的api还是没结果，甚至直接忽略了关键的md5加密，所以以后做题还是要搞清楚程序的具体流程再着手google才行。。

3. 我总觉得这一题是可以通过更改程序执行流程直接打印出flag的。。。但是动态调试发现这个程序完全没有执行这段代码，通过交叉引用可以发现我们分析的函数最开始其实是被绑定到了一个窗口上：

```
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+var_30.cbSize], 30h
mov     [ebp+var_30.style], 3
mov     [ebp+var_30.lpfnWndProc], offset sub_401014
mov     [ebp+var_30.cbClsExtra], 0
mov     [ebp+var_30.cbWndExtra], 0
mov     eax, [ebp+hInstance]
mov     [ebp+var_30.hInstance], eax
mov     esi, esp
push    6Bh             ; lpIconName
mov     ecx, [ebp+hInstance]
push    ecx             ; hInstance
call    ds:LoadIconA
cmp     esi, esp
```

消息循环绑定了我们的函数，然而我不知道怎么让他运行起来。。。。。有没有大神教教我。。。我的qq是574866810欢迎ctf爱好者来交流。