XCTF-Windows_Reverse1

原创

Lu1u~ 于 2021-06-23 22:52:48 发布 56 收藏 1

分类专栏： CTF-RE练习 文章标签： 信息安全

本文链接：https://blog.csdn.net/qq_52974719/article/details/118164150

XCTF-Windows_Reverse1

1、查壳

查出**upx**壳，老套路了，本想直接一波**esp**把壳子脱掉，结果出现内存访问失败的错误，呜呜，爬去**upx -d**

2、ida静态分析

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4; // [esp+4h] [ebp-804h] BYREF
  char v5[1023]; // [esp+5h] [ebp-803h] BYREF
  char v6; // [esp+404h] [ebp-404h] BYREF
  char v7[1023]; // [esp+405h] [ebp-403h] BYREF

  v6 = 0;
  memset(v7, 0, sizeof(v7));
  v4 = 0;
  memset(v5, 0, sizeof(v5));
  printf("please input code:");
  scanf("%s", &v6);
  sub_401000(&v6);                                  // 需要得到v6既为flag
  if ( !strcmp(&v4, "DDCTF{reverseME}") )
    printf("You've got it!!%s\n", &v4);
  else
    printf("Try again later.\n");
  return 0;
}
```


比较简洁，输入**v6**，调用函数**sub_401000**处理**v6**，最后判断**v4**的结果，芜湖，看似**v4**和**v6**毫无关系，其实细看调用**401000**函数前的汇编便能找到**v4**与**v6**间的联系。

```
:004010B7                lea     eax, [esp+82Ch+var_404] ; v6
:004010BE                push    eax
:004010BF                lea     ecx, [esp+830h+var_804] ; v4
:004010C3                call    sub_401000      ; 调用主要处理函数前 404和804和v6 v4的地址相关
```

注意到初始时**v4**和**v6**在栈内的位置便可以知道，**eax**中存放的是**v6**，**ecx**中存放的是**v4**，借助寄存器来实现参数传递，**ida**就分析不出来**v4**和**v6**的联系了。

```
unsigned int __cdecl sub_401000(const char *a1)
{
  _BYTE *v1; // ecx ecx其实就是v4数组 v4为DDCTF{reverseME}
  unsigned int v2; // edi
  unsigned int result; // eax
  const char *v4; // ebx

  v2 = 0;
  result = strlen(a1);
  if ( result )
  {
    v4 = (const char *)(a1 - v1);                   // 地址的差值
    do
    {
      *v1 = byte_402FF8[(char)v1[(_DWORD)v4]];      // 分析一下  原来v1[v4]就是顺序遍历v6 即我们的输入
      ++v2;
      ++v1;
      result = strlen(a1);                          // v4[i]=byte_402ff8[v6[i]] 需要借助OD将该字符dump出来
    }
    while ( v2 < result );
  }
  return result;
}
```


进入**401000**函数，有些小细节 **v4(临时变量)**，是地址**v6**和**v4**的差值 也就是**400** ,分析可知：

```
v1[v4] == *(v1+v4) == *(ebp-804+400) == *(ebp-404)
```

出现了，也就是这也是在遍历v6数组，可知最后v1*即v4的值为一个字符串[v6的ascii码]，所以只需拿到这个**byte_402FF8**字符串就ok了

3、提取索引字符串

1、提取的两种方式:一种在ida中巧妙分析

```
.rdata:00402FF8 ; char byte_402FF8[]
.rdata:00402FF8 byte_402FF8     db ?                    ; DATA XREF: sub_401000+24↑r
.rdata:00402FF9                 align 10h
.rdata:00402FF9 _rdata          ends
.rdata:00402FF9
.data:00403000 ; Section 3. (virtual address 00003000)
.data:00403000 ; Virtual size                  : 000003E4 (    996.)
.data:00403000 ; Section size in file          : 00000200 (    512.)
.data:00403000 ; Offset to raw data for section: 00001600
.data:00403000 ; Flags C0000040: Data Readable Writable
.data:00403000 ; Alignment     : default
.data:00403000 ; ===========================================================================
.data:00403000
.data:00403000 ; Segment type: Pure data
.data:00403000 ; Segment permissions: Read/Write
.data:00403000 _data            segment para public 'DATA' use32
.data:00403000                  assume cs:_data
.data:00403000                  ;org 403000h
.data:00403000 ; uintptr_t __security_cookie
.data:00403000 ___security_cookie dd 0BB40E64Eh         ; DATA XREF: _main+6↑r
.data:00403000                                          ; __security_check_cookie(x)↑r ...
.data:00403004 dword_403004     dd 44BF19B1h            ; DATA XREF: ___report_gsfailure+B0↑r
.data:00403004                                          ; ___security_init_cookie+2B↑w ...
.data:00403008                  db 0FFh
.data:00403009                  db 0FFh
.data:0040300A                  db 0FFh
.data:0040300B                  db 0FFh
.data:0040300C                  db 0FFh
.data:0040300D                  db 0FFh
.data:0040300E                  db 0FFh
.data:0040300F                  db 0FFh
.data:00403010 dword_403010     dd 0FFFFFFFEh           ; DATA XREF: _pre_c_init+CC↑r
.data:00403014 dword_403014     dd 1                    ; DATA XREF: _pre_c_init+B2↑r
.data:00403018 aZyxwvutsrqponm db '~}|{zyxwvutsrqponmlkjihgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>'
.data:00403018                  db '=<;:9876543210/.-,+*)(',27h,'&%$#"! ',0
```

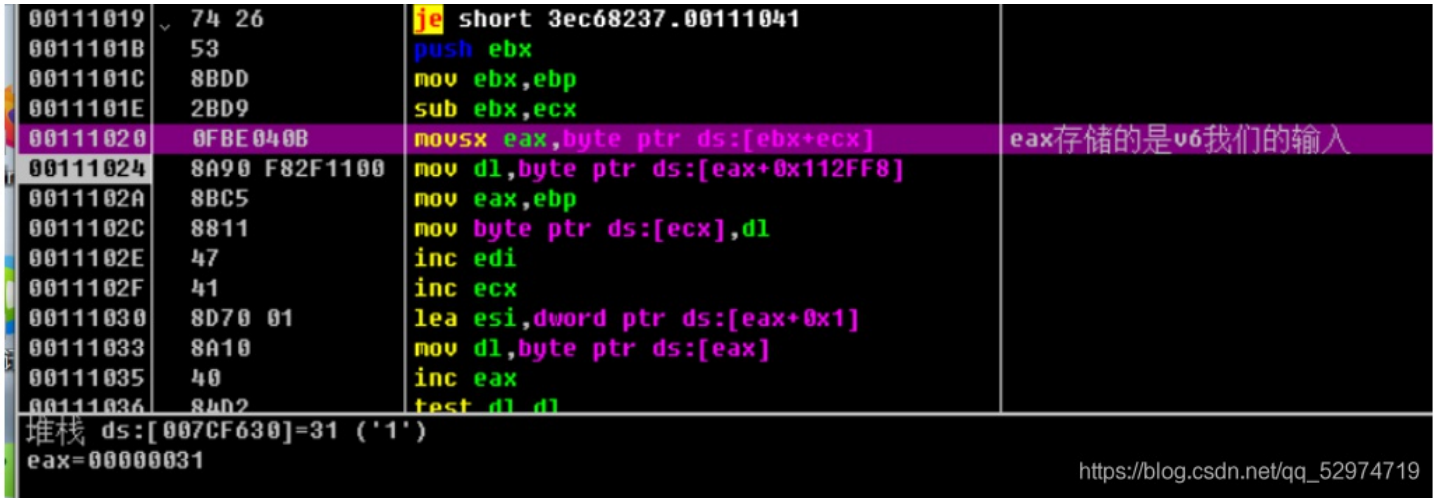

观察到改字符串从偏移量**0x2FF8**到**0x3018**到最后0，计算这两个地址的差值得到**32**,32我的天，一开是没想到，后来看师傅wp知道，前32的ASCII码对应的字符都是不可打印字符，而v6是我们的输入，所以根本不可能有前32位，而后面的字符串我们知道了，相当于字符串我们都知道了，写脚本就ok。

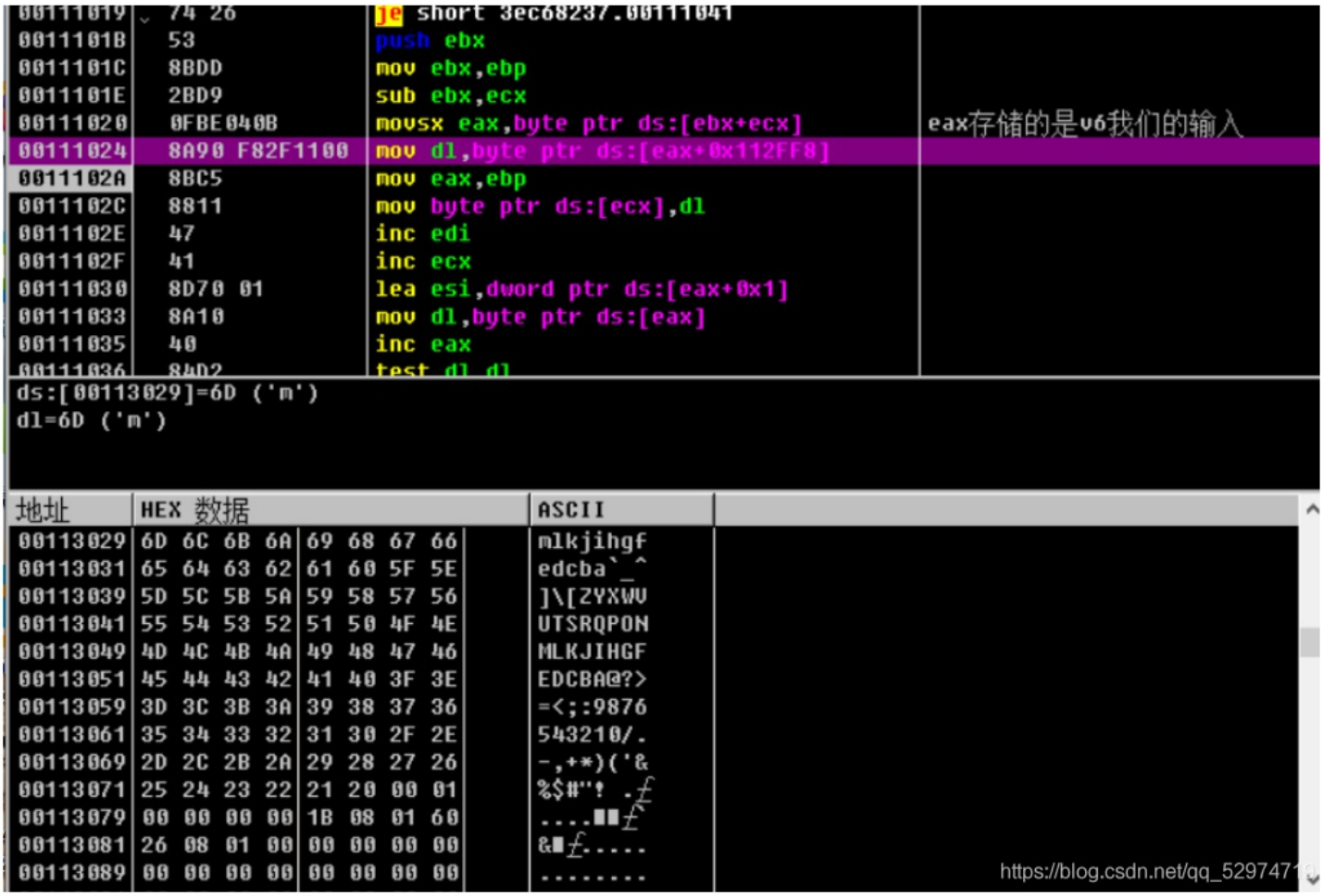
```
a='7E 7D 7C 7B 7A 79 78 77 76 75 74 73 72 71 70 6F 6E 6D 6C 6B 6A 69 68 67 66 65 64 63 62 61 60 5F 5E 5D 5C 5B 5
A 59 58 57 56 55 54 53 52 51 50 4F 4E 4D 4C 4B 4A 49 48 47 46 45 44 43 42 41 40 3F 3E 3D 3C 3B 3A 39 38 37 36 35
 34 33 32 31 30 2F 2E 2D 2C 2B 2A 29 28 27 26 25 24 23 22 21'
a=a.split(' ')
for i in range(len(a)):
    a[i]=int('0x'+a[i],16)
v4='DDCTF{reverseME}'
for i in v4:
    for j in range(len(a)):
        if a[j]==ord(i):
            print(chr(j+32),end='')#加32是因为前面32位虽然不可打印，但也在表中做下标
            break
```

2、OD动调找字符串，所有字符串都能提出来

虽然脱壳失败，但是我们用esp定律定位真正的程序还是挺简单的。

先单步，在**esp**数据窗口前四个字节下内存访问断点，之后运行，之后单步步入大跳，进入大跳内可以中文搜索引擎搜索一下，定位主要内容的位置。





通过第一张图的**eax=0x31**；**dl**串的地址为**eax+0x122FF8**，可知首地址为**0x122FF8**，而**0x6D**正好为第**50**个，即下标索引的**49**，其实**eax**为啥为**0x31**呢，因为在前面输入的时候输入**123**，而**1**的**ASCII**正好为**49**即**0x31**，这样也能看到是一个索引操作，索引后存到**dl**中之后**dl**赋值给**ecx**，即**v4**。我们已经知道最后变换过的**v4**了，而表也知道了，所以可以只需要逆向出**v6**就**ok**。

```
a='00 00 00 00 00 00 00 00 BC 7D 32 6D 43 82 CD 92 FF FF FF FF FF FF FF FF FE FF FF FF 01 00 00 00 7E 7D 7C 7B 7
A 79 78 77 76 75 74 73 72 71 70 6F 6E 6D 6C 6B 6A 69 68 67 66 65 64 63 62 61 60 5F 5E 5D 5C 5B 5A 59 58 57 56 55
 54 53 52 51 50 4F 4E 4D 4C 4B 4A 49 48 47 46 45 44 43 42 41 40 3F 3E 3D 3C 3B 3A 39 38 37 36 35 34 33 32 31 30
2F 2E 2D 2C 2B 2A 29 28 27 26 25 24 23 22 21 20 00 01 00 00 00 00 1B 08 01 60 26 08 01'
a=a.split(' ')
for i in range(len(a)):
    a[i]=int('0x'+a[i],16)
#print(a)
v4='DDCTF{reverseME}'
v=[]
for i in v4:
    for j in range(len(a)):
        if a[j]==ord(i) and j>32:
            print(chr(j),end='')  #这才是真正的flag
```

注意，如果是把所有表提出来，如过是在前32位内找到的不能要，因为v6是可打印的，必须ASCII码大于32！！！

对于脱壳后产生的问题，师傅们可以参考下这个师傅的链接：链接: 传送门

完结线:re弟弟的做题笔记，有时可能会带点稀奇古怪的思路，如有问题还望师傅们指正，专栏内容会随着比赛记录而不断更新哦。爬~