[ACTF新生赛2020]Splendid_MineCraft

原创

Em0s_Er1t 于 2021-04-22 20:37:22 发布 282 收藏 2

分类专栏： CTF-RE 文章标签： 字符串 python c语言 信息安全 安全漏洞

本文链接：https://blog.csdn.net/m0_46296905/article/details/116007661

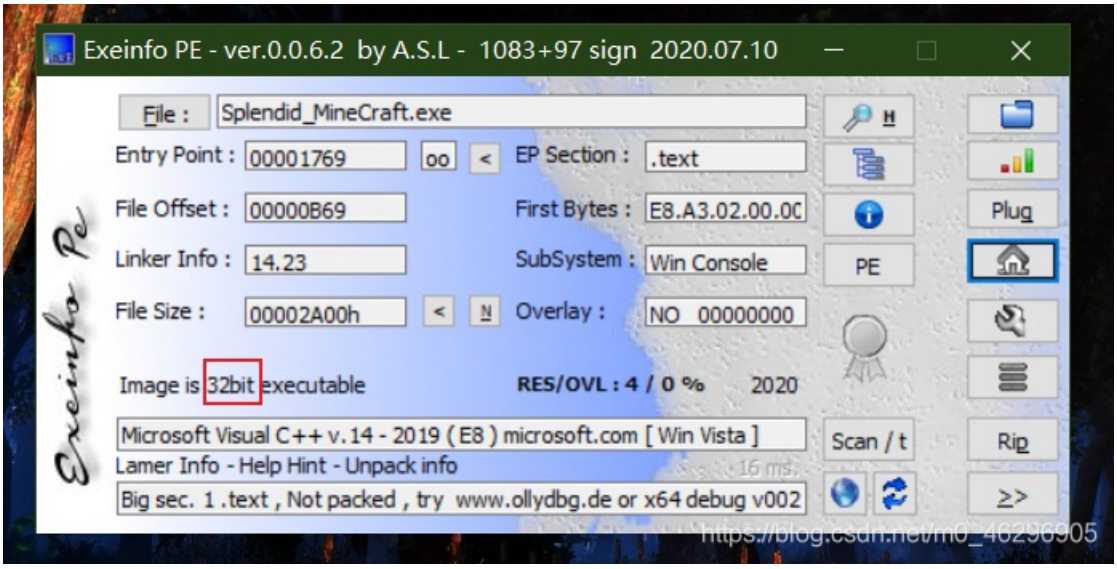
CTF-RE 专栏收录该内容

31 篇文章 2 订阅

订阅专栏

题目：[ACTF新生赛2020]Splendid_MineCraft

32位无壳程序



初步静态分析

老样子，ida32进去看主函数。

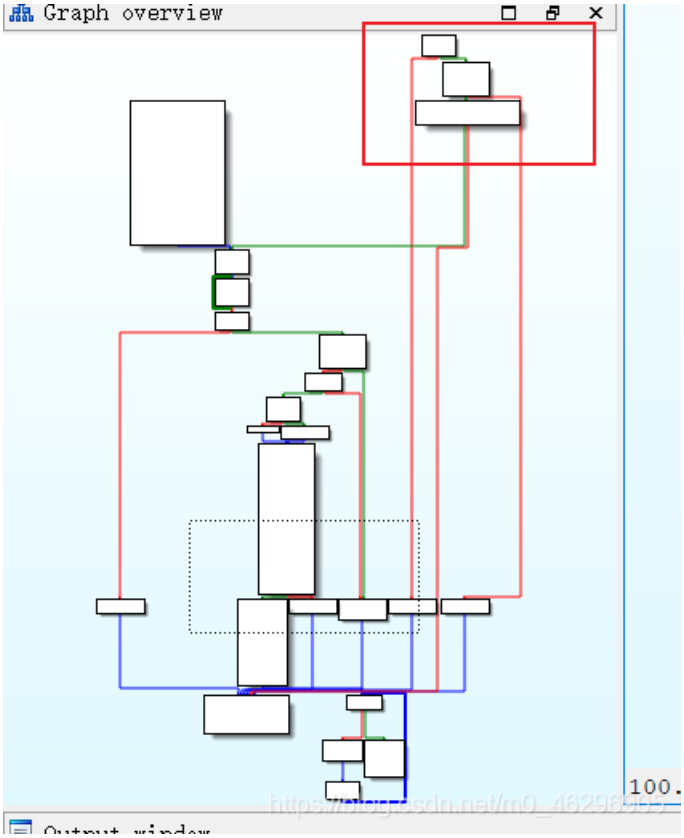
有提示输入错误的字串，但没有提示输入正确的字串。

```
__int16 v13; // [esp+48h] [ebp-20h]
char v14[4]; // [esp+4Ch] [ebp-1Ch]
__int16 v15; // [esp+50h] [ebp-18h]
int v16; // [esp+54h] [ebp-14h] BYREF
__int16 v17; // [esp+58h] [ebp-10h]
int v18; // [esp+5Ch] [ebp-Ch]
```

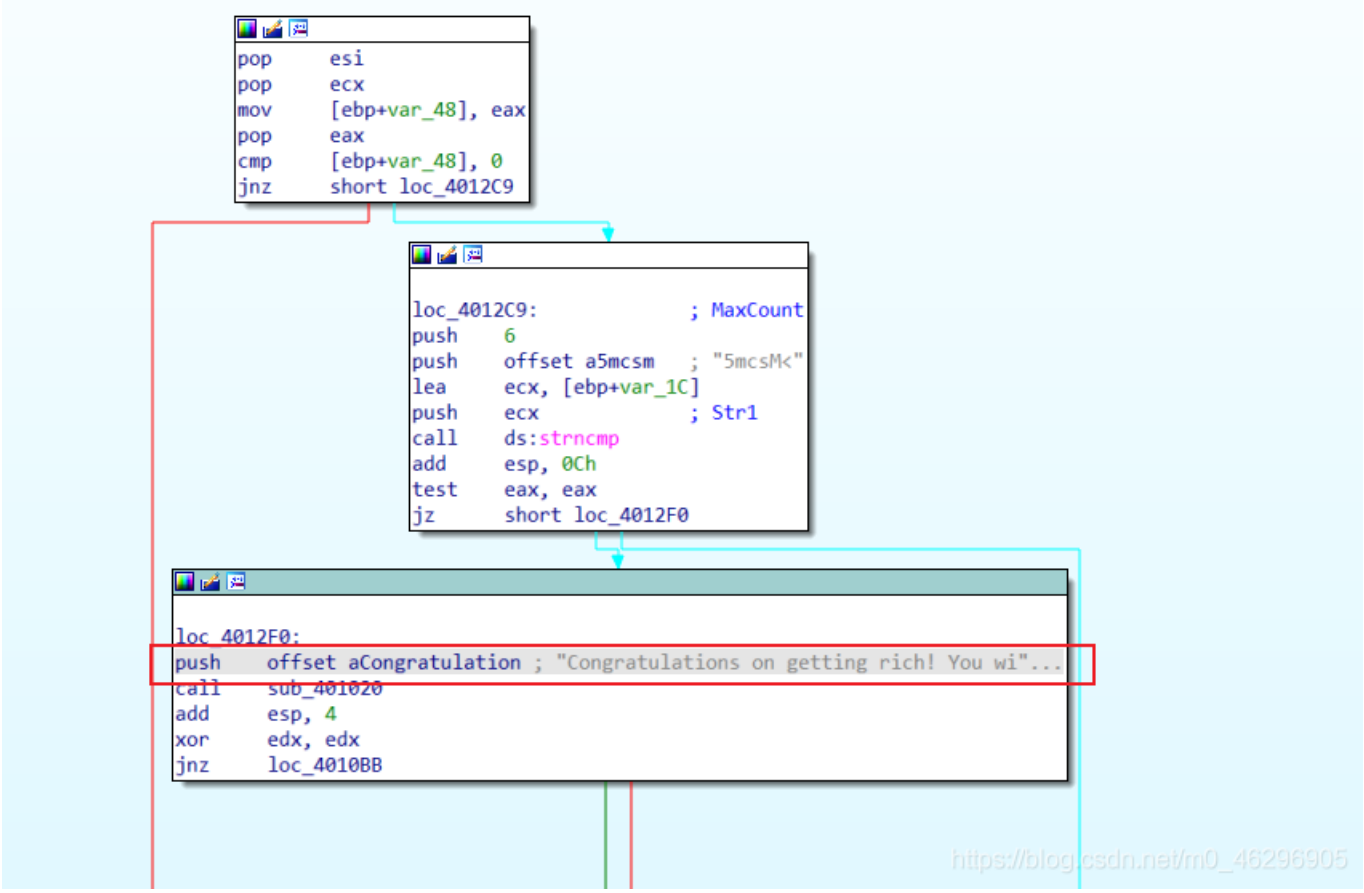
```
  __int16 v19; // [esp+60h] [ebp-8h]

  sub_401020("%s\n", aWelcomeToActfS);
  sub_401050("%s", Str1);
  if ( &Str1[strlen(Str1) + 1] - &Str1[1] == 26 && !strncmp(Str1, "ACTF{", 5u) && v11
  {
    v11 = 0;
    v3 = strtok(Str1, "_");
    v16 = *(v3 + 5);
    v17 = *(v3 + 9);
    v18 = *(v3 + 5);
    v19 = *(v3 + 9);
    v4 = strtok(0, "_");
    v12 = *v4;
    v13 = *(v4 + 2);
    v8 = strtok(0, "_");
    *v14 = *v8;
    v15 = *(v8 + 2);
    dword_403354 = &unk_4051D8;
    if ( (unk_4051D8)(&v16) )
    {
      v9 = BYTE2(v18) ^ HIBYTE(v19) ^ v18 ^ HIBYTE(v18) ^ BYTE1(v18) ^ v19;
      for ( i = 256; i < 496; ++i )
        byte_405018[i] ^= v9;
      __asm { jmp     eax }
    }
  }
  sub_401020("Wrong\n", v6);
  return 0;
}
```


瞄了一眼流程图（发现有个突出的独立分支），引起了我的怀疑。



红框部分的细节如下：



原来那个提示正确的字串在这个独立分支上，光标放在红框，再按tab看伪代码，发现显示在主函数的这个区域。

```
  dword_403354 = &unk_4051D8;
  if ( (unk_4051D8)(&v16) )
  {
    v9 = BYTE2(v18) ^ HIBYTE(v19) ^ v18 ^ HIBYTE(v18) ^ BYTE1(v18) ^ v19;
    for ( i = 256; i < 496; ++i )
      byte_405018[i] ^= v9;
    __asm { jmp     eax }
  }
}
```

在这个区域我们又找到意想不到的收获。

```
  v15 = *(v8 + 2);
  dword_403354 = &unk_4051D8;
  if ( (unk_4051D8)(&v16) )
  {
    v9 = BYTE2(v18) ^ HIBYTE(v19) ^ v18 ^ HIBYTE(v18) ^ BYTE1(v18) ^ v19;
    for ( i = 256; i < 496; ++i )
      byte_405018[i] ^= v9;
    __asm { jmp     eax }
  }
}
```

推测是smc没错了。

接下来具体分析主函数：

strtok函数：

- 函数原型：char *strtok(char *s, char *delim)
- 功能：作用于字符串s，以delim中的字符为分界符，将s切分成一个个子串；如果，s为空值NULL，则函数保存的指针SAVE_PTR在下一次调用中将作为起始位置。
- 内容：分解字符串，所谓分解，即没有生成新串，只是在s所指向的内容首次出现分界符的位置，将分界符修改成了'/0'
  第一次提取子串完毕之后，继续对源字符串s进行提取，应在其后（第二次，第三次。。。第n次）的调用中将strtok的第一个参数赋为空值NULL（表示函数继续从上 一次调用隐式保存的位置，继续分解字符串；对于前一次次调用来说，第一次调用结束前用一个this指针指向了分界符的下一位）
- 返回值：分隔符匹配到的第一个子串

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char *v3; // eax
  char *v4; // eax
  char v6; // [esp+0h] [ebp-68h]
  int i; // [esp+14h] [ebp-54h]
  char *v8; // [esp+18h] [ebp-50h]
  char v9; // [esp+20h] [ebp-48h]
  char Str1[25]; // [esp+24h] [ebp-44h] BYREF
  char v11; // [esp+3Dh] [ebp-2Bh]
  int v12; // [esp+44h] [ebp-24h]
  __int16 v13; // [esp+48h] [ebp-20h]
  char v14[4]; // [esp+4Ch] [ebp-1Ch]
  __int16 v15; // [esp+50h] [ebp-18h]
  int v16; // [esp+54h] [ebp-14h] BYREF
  __int16 v17; // [esp+58h] [ebp-10h]
  int v18; // [esp+5Ch] [ebp-Ch]
  __int16 v19; // [esp+60h] [ebp-8h]

  sub_401020("%s\n", aWelcomeToActfS);
  sub_401050("%s", Str1);
  if ( &Str1[strlen(Str1) + 1] - &Str1[1] == 26 && !strncmp(Str1, "ACTF{", 5u) && v11 == 125 )
  {
    v11 = 0;

    /*以‘_’为间隔划分flag*/
    v3 = strtok(Str1, "_");//第一次划分
    v16 = *(v3 + 5);
    v17 = *(v3 + 9);        //读取大括号里第一次划分出的内容（5个字符）
    v18 = *(v3 + 5);
    v19 = *(v3 + 9);        //读取大括号里第一次划分出的内容
    v4 = strtok(0, "_");    //第二次划分
    v12 = *v4;              //读取第二次划分出的内容
    v13 = *(v4 + 2);
    v8 = strtok(0, "_");    //第三次划分
    *v14 = *v8;
    v15 = *(v8 + 2);

    /*smc自修改*/
    dword_403354 = &unk_4051D8;
    if ( (unk_4051D8)(&v16) )
    {
     //取大括号内的前5个字符再加上‘/0’进行异或操作出v9
      v9 = BYTE2(v18) ^ HIBYTE(v19) ^ v18 ^ HIBYTE(v18) ^ BYTE1(v18) ^ v19;
      //用v9进行smc
      for ( i = 256; i < 496; ++i )
        byte_405018[i] ^= v9;
      __asm { jmp     eax }
    }
  }

  sub_401020("Wrong\n", v6);
  return 0;
}
```

推测出flag的格式

```
ACTF{??????_??????_??????}
```

然后开始重点分析smc部分。

我们可以发现这个程序的smc操作是跟flag的字符有关的，也就是说如果输入的大括号内的前5个字符正确，才会正常smc出正确代码，所以还是免不了动调。

动调解SMC

od打开，开始硬刚汇编。
先查找字符串，再结合ida的汇编代码定位程序进行smc的位置。下断点。
中间那个call指令调用应该就是smc代码了

```
push    offset Delimiter ; "_"
push    0                ; String
call    ds:strtok
add     esp, 8
mov     [ebp+var_50], eax
mov     edx, [ebp+var_50]
mov     eax, [edx]
mov     dword ptr [ebp+var_1C], eax
mov     cx, [edx+4]
mov     [ebp+var_18], cx
mov     ds:dword_403354, offset unk_4051D8
lea     edx, [ebp+var_14]
push    edx
call    ds:dword_403354
add     esp, 4
test    eax, eax
```



```
003111CA  .  8B55 B0      mov     edx, dword ptr [ebp-50]
003111CD  .  8B02         mov     eax, dword ptr [edx]
003111CF  .  8945 E4      mov     dword ptr [ebp-1C], eax
003111D2  .  66:8B4A 04   mov     cx, word ptr [edx+4]
003111D6  .  66:894D E8   mov     word ptr [ebp-18], cx
003111DA  .  C705 5433310 mov     dword ptr [313354], 003151D8
003111E4  .  8D55 EC      lea     edx, dword ptr [ebp-14]
003111E7  .  52           push    edx
003111E8  .  FF15 5433310 call    dword ptr [313354]
003111EE  .  83C4 04      add     esp, 4
003111F1  .  85C0         test    eax, eax
003111F3  .v 75 12        jnz     short 00311207
003111F5  .  68 20413100  push    00314120                wrong\n
003111FA  .  E8 21FEFFFF  call    00311020
003111FF  .  83C4 04      add     esp, 4
```


F9跑一下，为了程序能进入smc我们就输入符合格式的字符串

```
ACTF{111111_111111_111111}
```

F7单步步入那个刚刚说的call指令，发现一些奇怪的指令

```
003151D8   E8 00000000    call    003151DD
003151DD   5E             pop     esi
003151DE   57             push    edi
003151DF   33FF           xor     edi, edi
003151E1   81FF 51010000  cmp     edi, 151
003151E7   7F 12          jg      short 003151FB
003151E9   8A5C3E 1F      mov     bl, byte ptr [esi+edi+1F]
003151ED   80F3 72        xor     bl, 72
003151F0   885C3E 1F      mov     byte ptr [esi+edi+1F], bl
003151F4   47             inc     edi
003151F5   EB EA          jmp     short 003151E1
```

```
003151F7   48              dec     eax
003151F8   65:79 21        jns     short 0031521C
003151FB   5F              pop     edi
003151FC   27              daa
003151FD   F9              stc
003151FE   9E              sahf
003151FF   F1              int1
00315200   9E              sahf
00315201   5E              pop     esi
00315202   B5 37           mov     ch, 37
00315204   8A72 72         mov     dh, byte ptr [edx+72]
00315207 v 72 72           jb      short 0031527B
00315209   CA 7372         retf    7273
0031520C v 72 72           jb      short 00315280
0031520E   19BA 72B4367F   sbb     dword ptr [edx+7F36B472], edi
00315214   9A 41C87372 72  call    far 7272:7273C841
0031521B   B3 90           mov     bl, 90
0031521D ^ 72 B4           jb      short 003151D3
0031521F   36:67:9A 32CA7  call    far 7272:7273CA32
00315228   A3 92B43677     mov     dword ptr [7736B492], eax
0031522D   9A 43CB7372 72  call    far 7272:7273CB43
003151DD=003151DD
```


F7一步步运行就会发现下面的汇编代码变了，判断是正在smc。

```
3151DB   0000              add     byte ptr [eax], al
3151DD   5E                pop     esi
3151DE   57                push    edi
3151DF   33FF              xor     edi, edi
3151E1   81FF 51010000     cmp     edi, 151
3151E7 v 7F 12             jg      short 003151FB
3151E9   8A5C3E 1F         mov     bl, byte ptr [esi+edi+1F]
3151ED   80F3 72           xor     bl, 72
3151F0   885C3E 1F         mov     byte ptr [esi+edi+1F], bl
3151F4   47                inc     edi
3151F5 ^ EB EA             jmp     short 003151E1
3151F7   48                dec     eax
3151F8   65:79 21          jns     short 0031521C
3151FB   5F                pop     edi
3151FC   55                push    ebp
3151FD   8B9E F19E5EB5     mov     ebx, dword ptr [esi+B55E9EF1]
315203   37                aaa
315204   8A72 72           mov     dh, byte ptr [edx+72]
315207 v 72 72             jb      short 0031527B
315209   CA 7372           retf    7273
31520C v 72 72             jb      short 00315280
31520E   19BA 72B4367F     sbb     dword ptr [edx+7F36B472], edi
315214   9A 41C87372 72    call    far 7272:7273C841
31521B   B3 90             mov     bl, 90
31521D ^ 72 B4             jb      short 003151D3
31521F   36:67:9A 32CA7    call    far 7272:7273CA32
315228   A3 92B43677       mov     dword ptr [7736B492], eax
31522D   9A 43CB7372 72    call    far 7272:7273CB43
315234   19A3 71B43667     sbb     dword ptr [ebx+6736B471], esp
3151E1=003151E1
```

smc

在循环外下断点按F9走出循环，开始看汇编代码解密。

我们调试到这儿，发现一些字符串被存入内存中了

```
00315209   B8 01000000    mov     eax, 1
0031520E   6BC8 00        imul    ecx, eax, 0
00315211   C64408 E8 33   mov     byte ptr [ebp+ecx-18], 33          '3'
00315216   BA 01000000    mov     edx, 1
0031521B   C1E2 00        shl     edx, 0
0031521E   C64415 E8 40   mov     byte ptr [ebp+edx-18], 40          '@'
00315223   B8 01000000    mov     eax, 1
00315228   D1E0           shl     eax, 1
0031522A   C64405 E8 31   mov     byte ptr [ebp+eax-18], 31          '1'
0031522F   B9 01000000    mov     ecx, 1
00315234   6BD1 03        imul    edx, ecx, 3
00315237   C64415 E8 62   mov     byte ptr [ebp+edx-18], 62          'b'
0031523C   B8 01000000    mov     eax, 1
00315241   C1E0 02        shl     eax, 2
00315244   C64405 E8 3B   mov     byte ptr [ebp+eax-18], 3B          ';'
00315249   B9 01000000    mov     ecx, 1
0031524E   6BD1 05        imul    edx, ecx, 5
00315251   C64415 E8 62   mov     byte ptr [ebp+edx-18], 62          'b'
00315256   B8 01000000    mov     eax, 1
0031525B   6BC8 00        imul    ecx, eax, 0
0031525E   C6440D D4 57   mov     byte ptr [ebp+ecx-2C], 57          'w'
00315263   BA 01000000    mov     edx, 1
00315268   C1E2 00        shl     edx, 0
0031526B   C64415 D4 65   mov     byte ptr [ebp+edx-2C], 65          'e'
00315270   B8 01000000    mov     eax, 1
00315275   D1E0           shl     eax, 1
00315277   C64405 D4 6C   mov     byte ptr [ebp+eax-2C], 6C          'l'
0031527C   B9 01000000    mov     ecx, 1
00315281   6BD1 03        imul    edx, ecx, 3
00315284   C64415 D4 63   mov     byte ptr [ebp+edx-2C], 63          'c'
00315289   B8 01000000    mov     eax, 1
0031528E   C1E0 02        shl     eax, 2
00315291   C64405 D4 6F   mov     byte ptr [ebp+eax-2C], 6F          'o'
00315296   B9 01000000    mov     ecx, 1
0031529B   6BD1 05        imul    edx, ecx, 5
```


接着向下看。但我们输入的字符还未被利用，所以遇到一些跳转语句不用管，就让他跳转就好了。

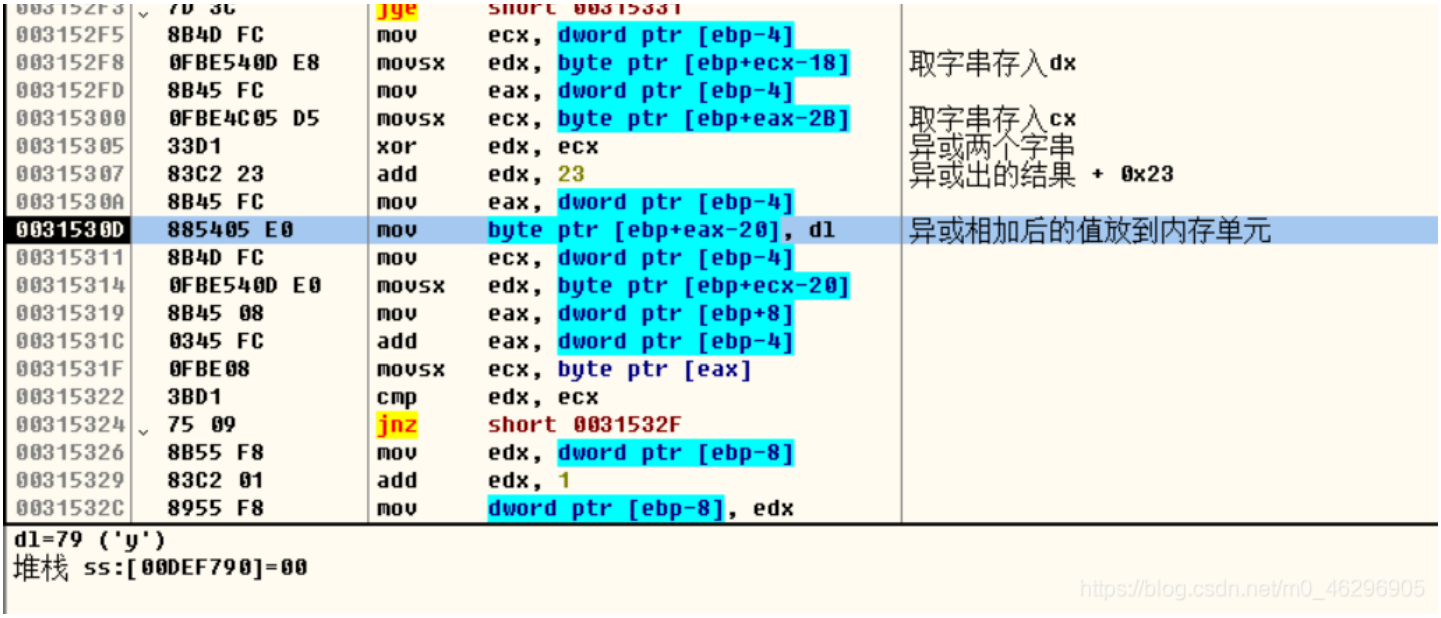
一直到这儿，把'3'放到了dx里面，感觉到程序应该要对刚刚载入内存的字符串做什么了

```
003152F5   8B4D FC        mov     ecx, dword ptr [ebp-4]
003152F8   0FBE540D E8    movsx   edx, byte ptr [ebp+ecx-18]
003152FD   8B45 FC        mov     eax, dword ptr [ebp-4]
00315300   0FBE4C05 D5    movsx   ecx, byte ptr [ebp+eax-2B]
00315305   33D1           xor     edx, ecx
00315307   83C2 23        add     edx, 23
0031530A   8B45 FC        mov     eax, dword ptr [ebp-4]
0031530D   885405 E0      mov     byte ptr [ebp+eax-20], dl
00315311   8B4D FC        mov     ecx, dword ptr [ebp-4]
00315314   0FBE540D E0    movsx   edx, byte ptr [ebp+ecx-20]
00315319   8B45 08        mov     eax, dword ptr [ebp+8]
0031531C   0345 FC        add     eax, dword ptr [ebp-4]
0031531F   0FBE08         movsx   ecx, byte ptr [eax]
00315322   3BD1           cmp     edx, ecx
00315324   75 09          jnz     short 0031532F
00315326   8B55 F8        mov     edx, dword ptr [ebp-8]
00315329   83C2 01        add     edx, 1
0031532C   8955 F8        mov     dword ptr [ebp-8], edx
堆栈 ss:[00DEF798]=33 ('3')
edx=00000008
```

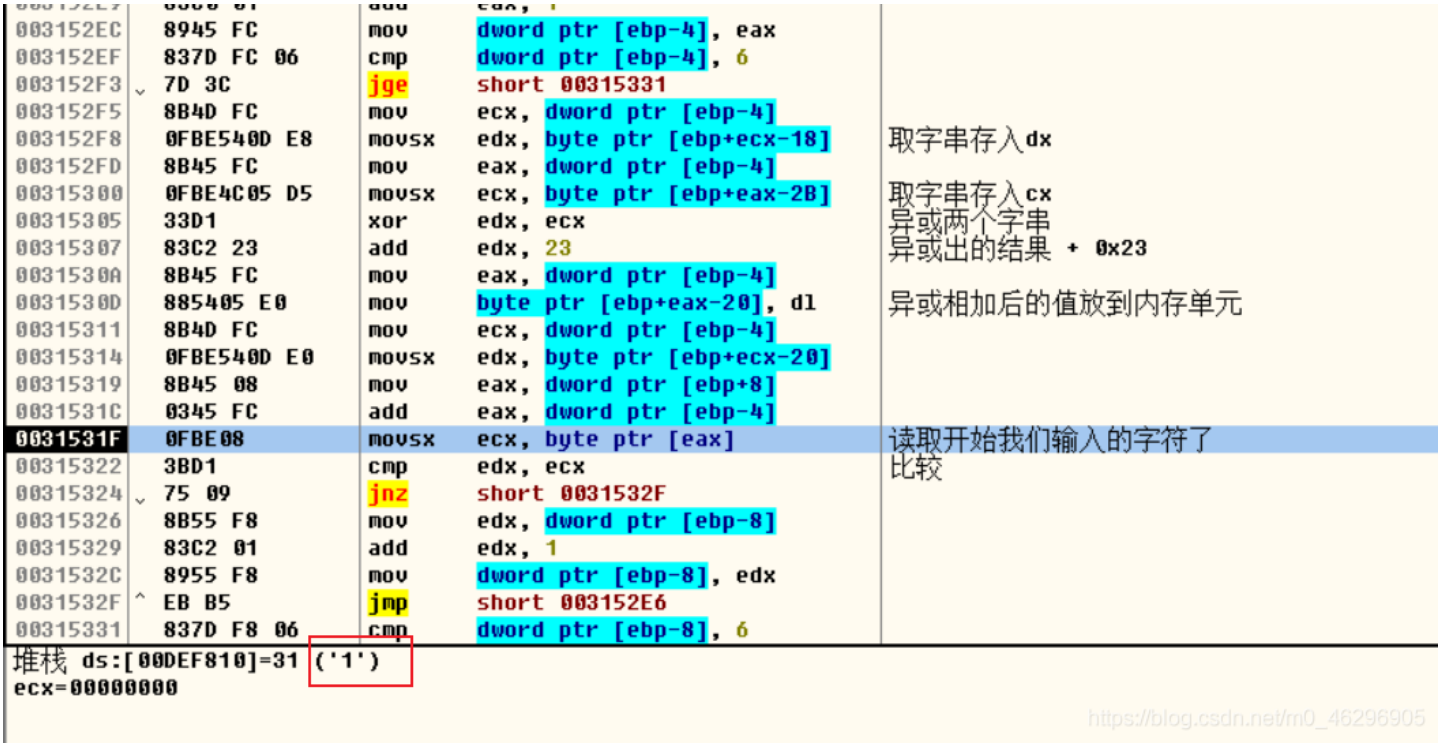

把'e'放到了cx里面。

看出来了，两个异或，结果再加0x23，提示也给出了结果'y'。

```
003152EF   837D FC 06     cmp     dword ptr [ebp-4], 6
```



看看它接下来要干什么。
原来是读取输入的字符然后比较，有一个不对直接报错，如果正确则进入下一个异或再相加运算，然后cmp判断。



强制改汇编代码(把下面的jnz改成jmp，强制跳转)，我们可以得出前五个字串

```
yOu0y*
```

这个call指令到此就结束。所以这个调用的函数其实实现了smc和检查大括号内的前五个符串的功能。

一方面由于我们之前静态分析知道，后面的判断是跟这五个字符串密切相关的，另一方面为了更好的看清楚程序对输入的字符做了什么操作，我们还是把字符设置成不同的值。最后我们把输入更新成

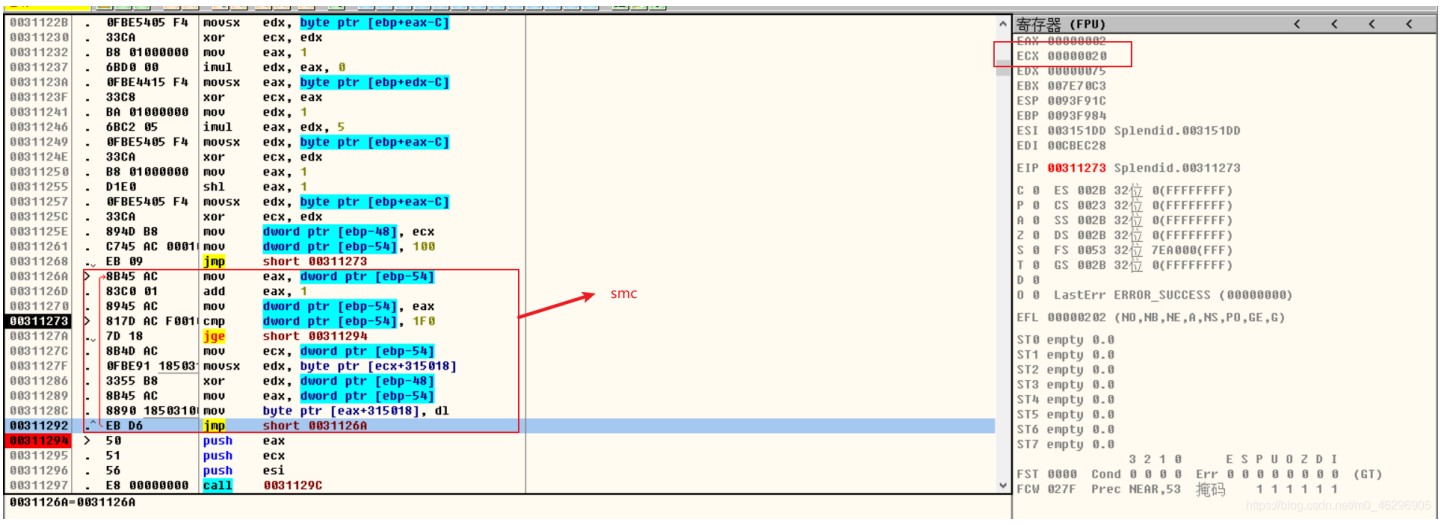
```
ACTF{yOu0y*_123456_123456}
```

重新动调进入中间五个字串的比较

```
003111F3  .  75 12          jnz     short 00311207
003111F5  .  68 20413100    push    00314120                 wrong\n
003111FA  .  E8 21FEFFFF    call    00311020
003111FF  .  83C4 04        add     esp, 4
00311202  .  E9 FE000000    jmp     00311305
00311207  >  B8 01000000    mov     eax, 1
0031120C  .  C1E0 02        shl     eax, 2
0031120F  .  0FBE4C05 F4    movsx   ecx, byte ptr [ebp+eax-C]
00311214  .  BA 01000000    mov     edx, 1
00311219  .  C1E2 00        shl     edx, 0
0031121C  .  0FBE4415 F4    movsx   eax, byte ptr [ebp+edx-C]
00311221  .  33C8           xor     ecx, eax
00311223  .  BA 01000000    mov     edx, 1
00311228  .  6BC2 03        imul    eax, edx, 3
0031122B  .  0FBE5405 F4    movsx   edx, byte ptr [ebp+eax-C]
00311230  .  33CA           xor     ecx, edx
00311232  .  B8 01000000    mov     eax, 1
00311237  .  6BD0 00        imul    edx, eax, 0
0031123A  .  0FBE4415 F4    movsx   eax, byte ptr [ebp+edx-C]
0031123F  .  33C8           xor     ecx, eax
00311241  .  BA 01000000    mov     edx, 1
00311246  .  6BC2 05        imul    eax, edx, 5
00311249  .  0FBE5405 F4    movsx   edx, byte ptr [ebp+eax-C]
0031124E  .  33CA           xor     ecx, edx
00311250  .  B8 01000000    mov     eax, 1
00311255  .  D1E0           shl     eax, 1
00311257  .  0FBE5405 F4    movsx   edx, byte ptr [ebp+eax-C]
0031125C  .  33CA           xor     ecx, edx
0031125E  .  894D B8        mov     dword ptr [ebp-48], ecx
跳转已实现
00311207=00311207
```


看下来这边的mov加异或应该对应之前用ida静态分析伪代码的时候，main函数那边求v9的那部分。

看寄存器窗口发现正确的v9就是0x20。我们也在紧跟的代码后面找到了smc循环。



走出循环后继续单步步入调用的子函数

```
00311292  .^ EB D6          jmp     short 0031126A
00311294  >  50             push    eax
00311295  .  51             push    ecx
00311296  .  56             push    esi
00311297  .  E8 00000000    call    0031129C
0031129C  $  59             pop     ecx
```

```
0031129D  .  8D05 1850310(  lea     eax, dword ptr [315018]
003112A3  .  8D7424 50      lea     esi, dword ptr [esp+50]
003112A7  .  05 00010000    add     eax, 100
003112AC  .  FFE0           jmp     eax
003112AE  .  5E             pop     esi
003112AF  .  59             pop     ecx
003112B0  .  8945 B8        mov     dword ptr [ebp-48], eax
003112B3  .  58             pop     eax
003112B4  .  837D B8 00     cmp     dword ptr [ebp-48], 0
003112B8  .v 75 0F          jnz     short 003112C9
003112BA  .  68 20413100    push    00314120                          wrong\n
003112BF  .  E8 5CFDFFFF    call    00311020
003112C4  .  83C4 04        add     esp, 4
003112C7  .v EB 3C          jmp     short 00311305
003112C9  >  6A 06          push    6                                 maxlen = 6
003112CB  .  68 34413100    push    00314134                          5mcsm<
003112D0  .  8D4D E4        lea     ecx, dword ptr [ebp-1C]
003112D3  .  51             push    ecx                               s1
003112D4  .  FF15 CC40310(  call    dword ptr [<&api-ms-win-crt-string-  strncmp
003112DA  .  83C4 0C        add     esp, 0C
003112DD  .  85C0           test    eax, eax
0031129C=0031129C
```

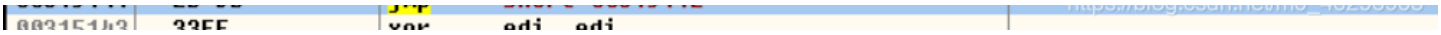

原来就在调用语句下面。这个函数里面还有个jmp eax指令

```
00311292  .^ EB D6          jmp     short 0031126A
00311294  >  50             push    eax
00311295  .  51             push    ecx
00311296  .  56             push    esi
00311297  .  E8 00000000    call    0031129C
0031129C  $  59             pop     ecx
0031129D  .  8D05 1850310(  lea     eax, dword ptr [315018]
003112A3  .  8D7424 50      lea     esi, dword ptr [esp+50]
003112A7  .  05 00010000    add     eax, 100
003112AC  .- FFE0           jmp     eax                               Splendid.00315118
003112AE  .  5E             pop     esi
003112AF  .  59             pop     ecx
003112B0  .  8945 B8        mov     dword ptr [ebp-48], eax
003112B3  .  58             pop     eax
003112B4  .  837D B8 00     cmp     dword ptr [ebp-48], 0
003112B8  .v 75 0F          jnz     short 003112C9
003112BA  .  68 20413100    push    00314120                          wrong\n
003112BF  .  E8 5CFDFFFF    call    00311020
003112C4  .  83C4 04        add     esp, 4
003112C7  .v EB 3C          jmp     short 00311305
003112C9  >  6A 06          push    6                                 maxlen = 6
003112CB  .  68 34413100    push    00314134                          5mcsm<
003112D0  .  8D4D E4        lea     ecx, dword ptr [ebp-1C]
003112D3  .  51             push    ecx                               s1
003112D4  .  FF15 CC40310(  call    dword ptr [<&api-ms-win-crt-string-  strncmp
003112DA  .  83C4 0C        add     esp, 0C
003112DD  .  85C0           test    eax, eax
003112DF  .v 74 0F          je      short 003112F0
003112E1  .  68 20413100    push    00314120                          wrong\n
003112E6  .  E8 35FDFFFF    call    00311020
003112EB  .  83C4 04        add     esp, 4
eax=00315118 (Splendid.00315118)
```


jmp eax步入开始分析。

发现了一个循环，是读取字符串，把它们放到另一个内存空间

```
0031511D     53             push    ebx
0031511E     83FF 06        cmp     edi, 6
00315121  v  7D 20          jge     short 00315143
00315123     33C9           xor     ecx, ecx
00315125     8A0C3E         mov     cl, byte ptr [esi+edi]            开始读中间五个字串
00315128  v  EB 0C          jmp     short 00315136
0031512A     123456         adc     dh, byte ptr [esi+edx*2]
0031512D  ^  78 9A          js      short 003150C9
0031512F     05 31323334    add     eax, 34333231
00315134     54             push    esp
00315135     53             push    ebx
00315136     33DB           xor     ebx, ebx
00315138     8A5C38 12      mov     bl, byte ptr [eax+edi+12]
0031513C     884C38 18      mov     byte ptr [eax+edi+18], cl
00315140     47             inc     edi
00315141  ^  EB DB          jmp     short 0031511E
```

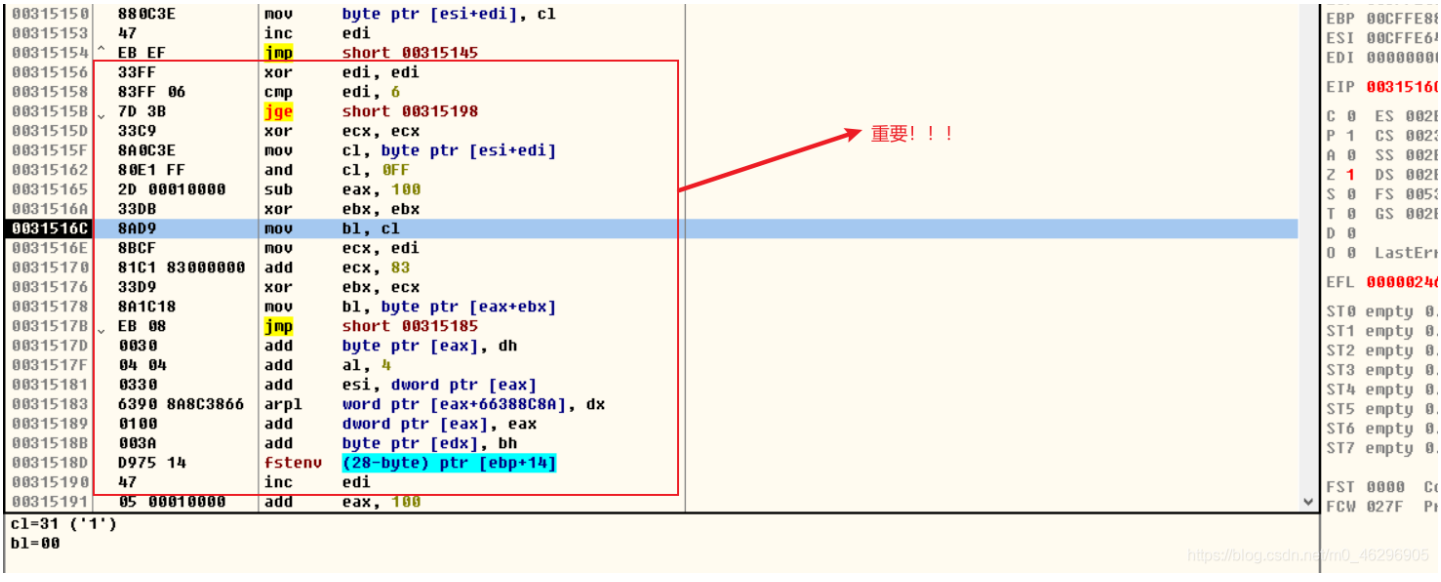


下面还是个同样作用的循环。。。

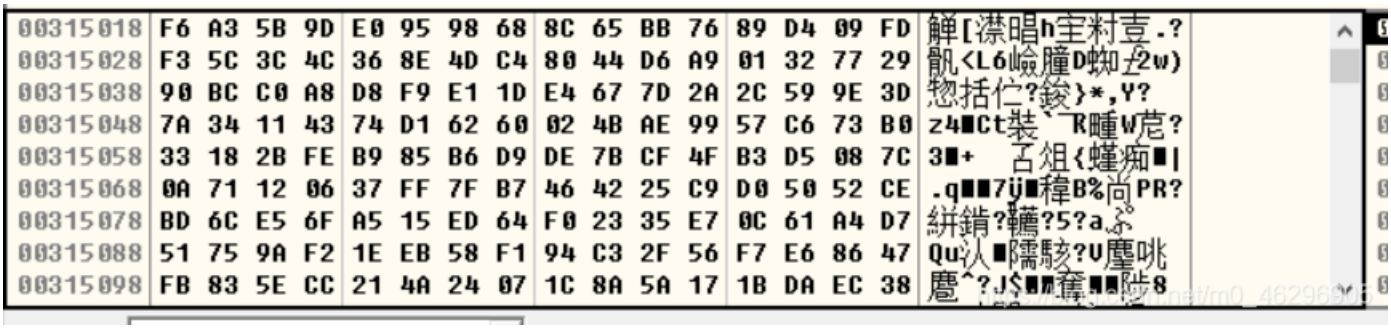
```
0031513A   3812          cmp     byte ptr [edx], dl
0031513C   884C38 18     mov     byte ptr [eax+edi+18], cl
00315140   47            inc     edi
00315141 ^ EB DB         jmp     short 0031511E
00315143   33FF          xor     edi, edi
00315145   83FF 06       cmp     edi, 6
00315148   7D 0C         jge     short 00315156
0031514A   33C9          xor     ecx, ecx
0031514C   8A4C38 18     mov     cl, byte ptr [eax+edi+18]
00315150   880C3E        mov     byte ptr [esi+edi], cl
00315153   47            inc     edi
00315154 ^ EB EF         jmp     short 00315145
00315156   33FF          xor     edi, edi
00315158   83FF 06       cmp     edi, 6
0031515B   7D 3B         jge     short 00315198
0031515D   33C9          xor     ecx, ecx
0031515F   8A0C3E        mov     cl, byte ptr [esi+edi]
00315162   80E1 FF       and     cl, 0FF
00315165   2D 00010000   sub     eax, 100
00315145=00315145
```


走出循环之后我们发现了。

程序将字符串的下表加上0x83，再跟字符本身异或，得到的结果存入bx寄存器。紧接着bx加上ax的值存入了bl，这里的al存放的是一个数组的首地址，所以这边是检索数组的元素。



直接选中ax寄存器里的值右键点击数据窗口中跟随，如下：

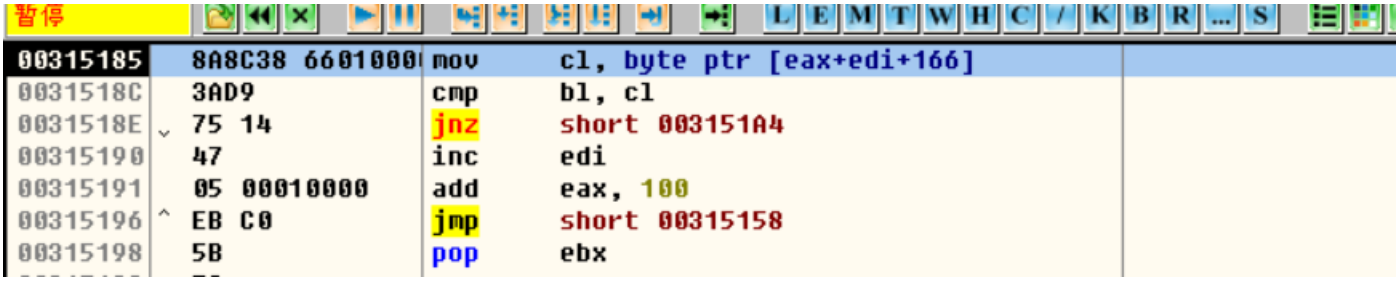


把这些数据复制下来便得到如下的检索表

把这些数据复制下来使得到如下的检索表：

```
[ 0xF6, 0xA3, 0x5B, 0x9D, 0xE0, 0x95, 0x98, 0x68, 0x8C, 0x65,
  0xBB, 0x76, 0x89, 0xD4, 0x09, 0xFD, 0xF3, 0x5C, 0x3C, 0x4C,
  0x36, 0x8E, 0x4D, 0xC4, 0x80, 0x44, 0xD6, 0xA9, 0x01, 0x32,
  0x77, 0x29, 0x90, 0xBC, 0xC0, 0xA8, 0xD8, 0xF9, 0xE1, 0x1D,
  0xE4, 0x67, 0x7D, 0x2A, 0x2C, 0x59, 0x9E, 0x3D, 0x7A, 0x34,
  0x11, 0x43, 0x74, 0xD1, 0x62, 0x60, 0x02, 0x4B, 0xAE, 0x99,
  0x57, 0xC6, 0x73, 0xB0, 0x33, 0x18, 0x2B, 0xFE, 0xB9, 0x85,
  0xB6, 0xD9, 0xDE, 0x7B, 0xCF, 0x4F, 0xB3, 0xD5, 0x08, 0x7C,
  0x0A, 0x71, 0x12, 0x06, 0x37, 0xFF, 0x7F, 0xB7, 0x46, 0x42,
  0x25, 0xC9, 0xD0, 0x50, 0x52, 0xCE, 0xBD, 0x6C, 0xE5, 0x6F,
  0xA5, 0x15, 0xED, 0x64, 0xF0, 0x23, 0x35, 0xE7, 0x0C, 0x61,
  0xA4, 0xD7, 0x51, 0x75, 0x9A, 0xF2, 0x1E, 0xEB, 0x58, 0xF1,
  0x94, 0xC3, 0x2F, 0x56, 0xF7, 0xE6, 0x86, 0x47, 0xFB, 0x83,
  0x5E, 0xCC, 0x21, 0x4A, 0x24, 0x07, 0x1C, 0x8A, 0x5A, 0x17,
  0x1B, 0xDA, 0xEC, 0x38, 0x0E, 0x7E, 0xB4, 0x48, 0x88, 0xF4,
  0xB8, 0x27, 0x91, 0x00, 0x13, 0x97, 0xBE, 0x53, 0xC2, 0xE8,
  0xEA, 0x1A, 0xE9, 0x2D, 0x14, 0x0B, 0xBF, 0xB5, 0x40, 0x79,
  0xD2, 0x3E, 0x19, 0x5D, 0xF8, 0x69, 0x39, 0x5F, 0xDB, 0xFA,
  0xB2, 0x8B, 0x6E, 0xA2, 0xDF, 0x16, 0xE2, 0x63, 0xB1, 0x20,
  0xCB, 0xBA, 0xEE, 0x8D, 0xAA, 0xC8, 0xC7, 0xC5, 0x05, 0x66,
  0x6D, 0x3A, 0x45, 0x72, 0x0D, 0xCA, 0x84, 0x4E, 0xF5, 0x31,
  0x6B, 0x92, 0xDC, 0xDD, 0x9C, 0x3F, 0x55, 0x96, 0xA1, 0x9F,
  0xCD, 0x9B, 0xE3, 0xA0, 0xA7, 0xFC, 0xC1, 0x78, 0x10, 0x2E,
  0x82, 0x8F, 0x30, 0x54, 0x04, 0xAC, 0x41, 0x93, 0xD3, 0x3B,
  0xEF, 0x03, 0x81, 0x70, 0xA6, 0x1F, 0x22, 0x26, 0x28, 0x6A,
  0xAB, 0x87, 0xAD, 0x49, 0x0F, 0xAF]
```

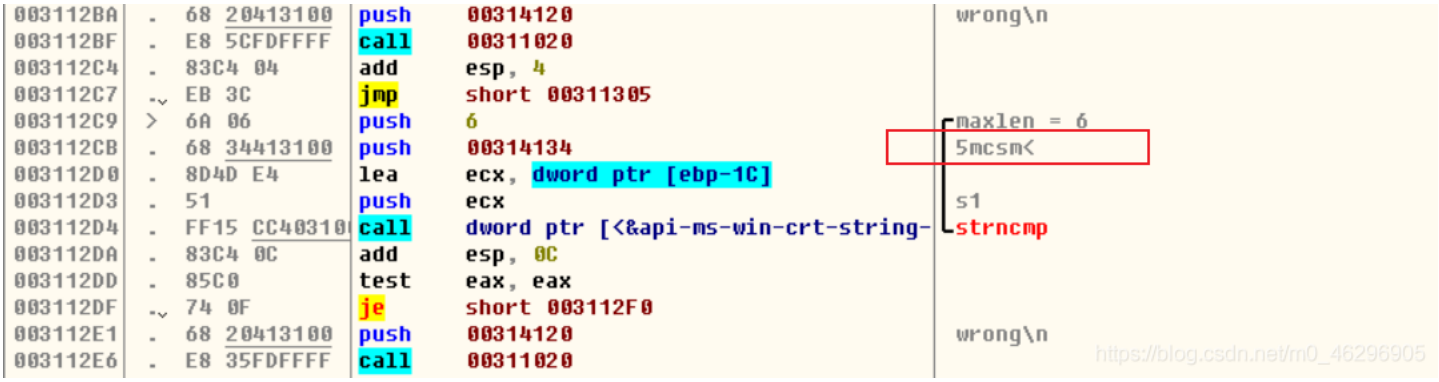
进入那个不明指向的跳转（这边不是很理解为什么这两个指令不触发跳转的时候显示不出来，触发跳转了才显示出来），不难看出cl里面存放了待比较的值



收集下来得出结论：我们加密后应该得到0x30,0x4,0x4,0x3,0x30,0x63这六个数据。
所以，第二个分析完毕

最后五个字串就简单了，直接告诉我们是

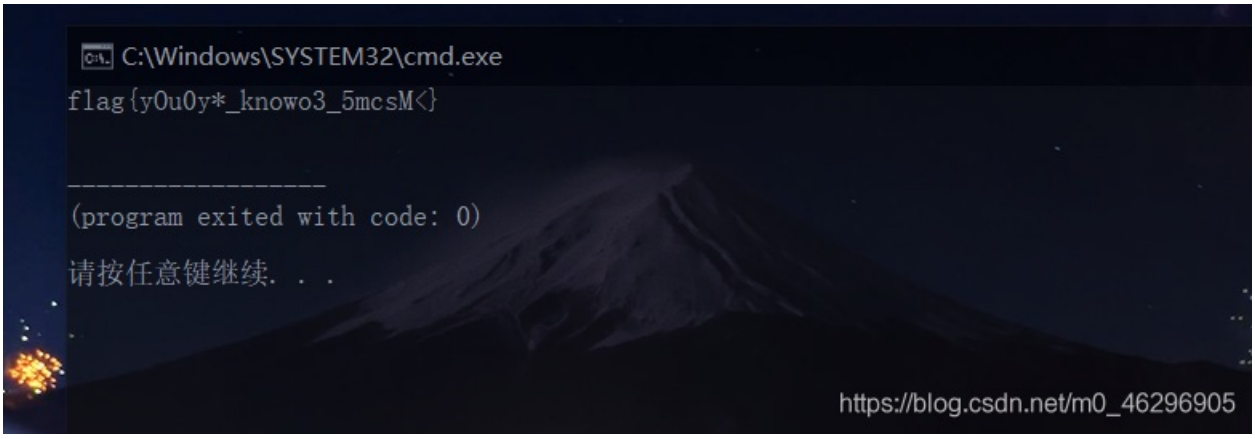
```
5mcsM<
```



附上exp：

```
table=[0xF6, 0xA3, 0x5B, 0x9D, 0xE0, 0x95, 0x98, 0x68, 0x8C, 0x65,
  0xBB, 0x76, 0x89, 0xD4, 0x09, 0xFD, 0xF3, 0x5C, 0x3C, 0x4C,
  0x36, 0x8E, 0x4D, 0xC4, 0x80, 0x44, 0xD6, 0xA9, 0x01, 0x32,
  0x77, 0x29, 0x90, 0xBC, 0xC0, 0xA8, 0xD8, 0xF9, 0xE1, 0x1D,
  0xE4, 0x67, 0x7D, 0x2A, 0x2C, 0x59, 0x9E, 0x3D, 0x7A, 0x34,
  0x11, 0x43, 0x74, 0xD1, 0x62, 0x60, 0x02, 0x4B, 0xAE, 0x99,
  0x57, 0xC6, 0x73, 0xB0, 0x33, 0x18, 0x2B, 0xFE, 0xB9, 0x85,
  0xB6, 0xD9, 0xDE, 0x7B, 0xCF, 0x4F, 0xB3, 0xD5, 0x08, 0x7C,
  0x0A, 0x71, 0x12, 0x06, 0x37, 0xFF, 0x7F, 0xB7, 0x46, 0x42,
  0x25, 0xC9, 0xD0, 0x50, 0x52, 0xCE, 0xBD, 0x6C, 0xE5, 0x6F,
  0xA5, 0x15, 0xED, 0x64, 0xF0, 0x23, 0x35, 0xE7, 0x0C, 0x61,
  0xA4, 0xD7, 0x51, 0x75, 0x9A, 0xF2, 0x1E, 0xEB, 0x58, 0xF1,
  0x94, 0xC3, 0x2F, 0x56, 0xF7, 0xE6, 0x86, 0x47, 0xFB, 0x83,
  0x5E, 0xCC, 0x21, 0x4A, 0x24, 0x07, 0x1C, 0x8A, 0x5A, 0x17,
  0x1B, 0xDA, 0xEC, 0x38, 0x0E, 0x7E, 0xB4, 0x48, 0x88, 0xF4,
  0xB8, 0x27, 0x91, 0x00, 0x13, 0x97, 0xBE, 0x53, 0xC2, 0xE8,
  0xEA, 0x1A, 0xE9, 0x2D, 0x14, 0x0B, 0xBF, 0xB5, 0x40, 0x79,
  0xD2, 0x3E, 0x19, 0x5D, 0xF8, 0x69, 0x39, 0x5F, 0xDB, 0xFA,
  0xB2, 0x8B, 0x6E, 0xA2, 0xDF, 0x16, 0xE2, 0x63, 0xB1, 0x20,
  0xCB, 0xBA, 0xEE, 0x8D, 0xAA, 0xC8, 0xC7, 0xC5, 0x05, 0x66,
  0x6D, 0x3A, 0x45, 0x72, 0x0D, 0xCA, 0x84, 0x4E, 0xF5, 0x31,
  0x6B, 0x92, 0xDC, 0xDD, 0x9C, 0x3F, 0x55, 0x96, 0xA1, 0x9F,
  0xCD, 0x9B, 0xE3, 0xA0, 0xA7, 0xFC, 0xC1, 0x78, 0x10, 0x2E,
  0x82, 0x8F, 0x30, 0x54, 0x04, 0xAC, 0x41, 0x93, 0xD3, 0x3B,
  0xEF, 0x03, 0x81, 0x70, 0xA6, 0x1F, 0x22, 0x26, 0x28, 0x6A,
  0xAB, 0x87, 0xAD, 0x49, 0x0F, 0xAF]
m=[0x30,0x4,0x4,0x3,0x30,0x63]
flag=''
for i in range(len(m)):
 flag+=chr(table.index(m[i])^(i+0x83))
print('flag{'+'yOu0y*_'+flag+'_5mcsM<}')
```

get flag！！！




```
flag{yOu0y*_knowo3_5mcsM<}
```