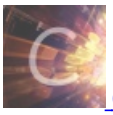
攻防世界逆向wp

原创

R4dish 于 2019-05-14 13:18:03 发布 2474 收藏 3

分类专栏： CTF-Writeup 文章标签： CTF-Writeup

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/qq_40934487/article/details/90204245
版权



CTF-Writeup 专栏收录该内容
4 篇文章 1 订阅
订阅专栏

babyre

file一下发现是64elf文件，拖到IDA中查看：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s; // [rsp+0h] [rbp-20h]
  int v5; // [rsp+18h] [rbp-8h]
  int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; i <= 181; ++i )
  {
    envp = (const char **)(*((unsigned __int8 *)judge + i) ^ 0xCu);
    *((_BYTE *)judge + i) ^= 0xCu;
  }
  printf("Please input flag:", argv, envp);
  __isoc99_scanf("%20s", &s);
  v5 = strlen(&s);
  if ( v5 == 14 && (unsigned int)judge(&s) )
    puts("Right!");
  else
    puts("Wrong!");
  return 0;
}
```

流程很简单，输入一个flag，然后长度限制为14，judge返回1，则是正确的flag。看到之前有一个for循环，作用是SMC，把代码还原出来，这里我用IDC脚本将代码直接在IDA中还原

```
auto start,i,a;
start = 0x600b00;
for(i=0;i<=181;i++)
{
    a=Byte(start+i)^0xc;
    PatchByte(start+i,a);
}
```

然后查看judge函数：

```
 55                                     push    rbp
.data:0000000000600B01 48 89 E5                                mov     rbp, rsp
```

```
.data:0000000000600B04 48 89 7D D8                          mov     [rbp-28h], rdi
.data:0000000000600B08 C6 45 E0 66                          mov     byte ptr [rbp-20h], 66h
.data:0000000000600B08                      judge           endp ; sp-analysis failed
.data:0000000000600B08
.data:0000000000600B0C
.data:0000000000600B0C                      loc_600B0C:
.data:0000000000600B0C C6 45 E1 6D                          mov     byte ptr [rbp-1Fh], 6Dh
.data:0000000000600B10 C6 45 E2 63                          mov     byte ptr [rbp-1Eh], 63h
.data:0000000000600B14 C6 45 E3 64                          mov     byte ptr [rbp-1Dh], 64h
.data:0000000000600B18 C6 45 E4 7F                          mov     byte ptr [rbp-1Ch], 7Fh
.data:0000000000600B1C C6 45 E5 6B                          mov     byte ptr [rbp-1Bh], 6Bh
.data:0000000000600B20 C6 45 E6 37                          mov     byte ptr [rbp-1Ah], 37h
.data:0000000000600B24 C6 45 E7 64                          mov     byte ptr [rbp-19h], 64h
.data:0000000000600B28 C6 45 E8 3B                          mov     byte ptr [rbp-18h], 3Bh
.data:0000000000600B2C C6 45 E9 56                          mov     byte ptr [rbp-17h], 56h
.data:0000000000600B30 C6 45 EA 60                          mov     byte ptr [rbp-16h], 60h
.data:0000000000600B34 C6 45 EB 3B                          mov     byte ptr [rbp-15h], 3Bh
.data:0000000000600B38 C6 45 EC 6E                          mov     byte ptr [rbp-14h], 6Eh
.data:0000000000600B3C C6 45 ED 70                          mov     byte ptr [rbp-13h], 70h
.data:0000000000600B40 C7 45 FC 00 00 00 00                 mov     dword ptr [rbp-4], 0
.data:0000000000600B47 EB 28                                jmp     short loc_600B71
.data:0000000000600B49                      ; ---------------------------------------------------------------
------------
.data:0000000000600B49
.data:0000000000600B49                      loc_600B49:                             ; CODE XREF: .data:000000
0000600B75↓j
.data:0000000000600B49 8B 45 FC                             mov     eax, [rbp-4]
.data:0000000000600B4C 48 63 D0                             movsxd  rdx, eax
.data:0000000000600B4F 48 8B 45 D8                          mov     rax, [rbp-28h]
.data:0000000000600B53 48 01 D0                             add     rax, rdx
.data:0000000000600B56 8B 55 FC                             mov     edx, [rbp-4]
.data:0000000000600B59 48 63 CA                             movsxd  rcx, edx
.data:0000000000600B5C 48 8B 55 D8                          mov     rdx, [rbp-28h]
.data:0000000000600B60 48 01 CA                             add     rdx, rcx
.data:0000000000600B63 0F B6 12                             movzx   edx, byte ptr [rdx]
.data:0000000000600B66 8B 4D FC                             mov     ecx, [rbp-4]
.data:0000000000600B69 31 CA                                xor     edx, ecx
.data:0000000000600B6B 88 10                                mov     [rax], dl
.data:0000000000600B6D 83 45 FC 01                          add     dword ptr [rbp-4], 1
.data:0000000000600B71
.data:0000000000600B71                      loc_600B71:                             ; CODE XREF: .data:000000
0000600B47↑j
.data:0000000000600B71 83 7D FC 0D                          cmp     dword ptr [rbp-4], 13
.data:0000000000600B75 7E D2                                jle     short loc_600B49
.data:0000000000600B77 C7 45 FC 00 00 00 00                 mov     dword ptr [rbp-4], 0
.data:0000000000600B7E EB 29                                jmp     short loc_600BA9
.data:0000000000600B80                      ; ---------------------------------------------------------------
------------
.data:0000000000600B80
.data:0000000000600B80                      loc_600B80:                             ; CODE XREF: .data:000000
0000600BAD↓j
.data:0000000000600B80 8B 45 FC                             mov     eax, [rbp-4]
.data:0000000000600B83 48 63 D0                             movsxd  rdx, eax
.data:0000000000600B86 48 8B 45 D8                          mov     rax, [rbp-28h]
.data:0000000000600B8A 48 01 D0                             add     rax, rdx
.data:0000000000600B8D 0F B6 10                             movzx   edx, byte ptr [rax]
.data:0000000000600B90 8B 45 FC                             mov     eax, [rbp-4]
.data:0000000000600B93 48 98                                cdqe
.data:0000000000600B95 0F B6 44 05 E0                       movzx   eax, byte ptr [rbp+rax-20h]
.data:0000000000600B9A 38 C2                                cmp     dl, al
```

```
.data:0000000000600B9C 74 07                                jz      short loc_600BA5
.data:0000000000600B9E B8 00 00 00 00                       mov     eax, 0
.data:0000000000600BA3 EB 0F                                jmp     short loc_600BB4
.data:0000000000600BA5                      ; ---------------------------------------------------------------
------------
.data:0000000000600BA5
.data:0000000000600BA5                      loc_600BA5:                             ; CODE XREF: .data:000000
0000600B9C↑j
.data:0000000000600BA5 83 45 FC 01                          add     dword ptr [rbp-4], 1
.data:0000000000600BA9
.data:0000000000600BA9                      loc_600BA9:                             ; CODE XREF: .data:000000
0000600B7E↑j
.data:0000000000600BA9 83 7D FC 0D                          cmp     dword ptr [rbp-4], 0Dh
.data:0000000000600BAD 7E D1                                jle     short loc_600B80
.data:0000000000600BAF B8 01 00 00 00                       mov     eax, 1
.data:0000000000600BB4
.data:0000000000600BB4                      loc_600BB4:                             ; CODE XREF: .data:000000
0000600BA3↑j
.data:0000000000600BB4 5D                                   pop     rbp
.data:0000000000600BB5 C3                                   retn
.data:0000000000600BB5                      _data           ends
```

发现IDA在f5的时候抛出错误，所以我们只能读懂汇编代码，汇编代码也不多

```
for(i=0;i<14;i++)
{
 argv[i]=argv[i]^i;
}
for(i=0;i<14;i++)
{
 if(argv[i]!=current[i])
 {
  return 0;
 }
}
return 1;
```

生成flag：

```
current = [0x66,0x6D,0x63,0x64,0x7F,0x6B,0x37,0x64,0x3B,0x56,0x60,0x3B,0x6E,0x70]
for i in range(len(current)):
 print chr(current[i]^i),
```

flag: flag{n1c3_j0b}

easy_re

file一下发现是32位ELF文件，然后用IDA打开发现好像是加过壳的。用DIE.exe来查一下发现改程序是经过UPX加壳后的程序，可以手动脱壳，也可以用UPX脱壳

```
➜  upx-3.91-amd64_linux ./upx -d easy_re
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
      7540 <-      4352   57.72%    linux/386    easy_re

Unpacked 1 file.
➜  upx-3.91-amd64_linux
```

再用IDA打开就可以清晰的看出来代码流程了

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int pipedes[2]; // [esp+18h] [ebp-38h]
  __pid_t v5; // [esp+20h] [ebp-30h]
  int v6; // [esp+24h] [ebp-2Ch]
  char buf; // [esp+2Eh] [ebp-22h]
  unsigned int v8; // [esp+4Ch] [ebp-4h]

  v8 = __readgsdword(0x14u);
  pipe(pipedes);
  v5 = fork();
  if ( !v5 )
  {
    puts("\nOMG!!!! I forgot kid's id");
    write(pipedes[1], "6980087614356821435692875 3", 0x1Du);
    puts("Ready to exit     ");
    exit(0);
  }
  read(pipedes[0], &buf, 0x1Du);
  __isoc99_scanf("%d", &v6);
  if ( v6 == v5 )
  {
    if ( (*(_DWORD *)((_BYTE *)lol + 3) & 0xFF) == 204 )
    {
      puts(":D");
      exit(1);
    }
    printf("\nYou got the key\n ");
    lol(&buf);
  }
  wait(0);
  return 0;
}
```

pipe()函数

pipe函数是在linux编程上的一个函数，pipe是一种把两个进程之间的标准输入和标准输出连接起来的机制，从而提供一种让多个进程间通信的方法，当进程创建pipe时，每次都需要提供两个文件描述符来操作管道，其中一个进行写操作，另一个对管道进行读操作。对管道的读写与一般的IO系统函数一致，使用write()进行写入数据，使用read()读出数据.

```
#include<unistd.h>
int pipe(int filedes[2]);
```

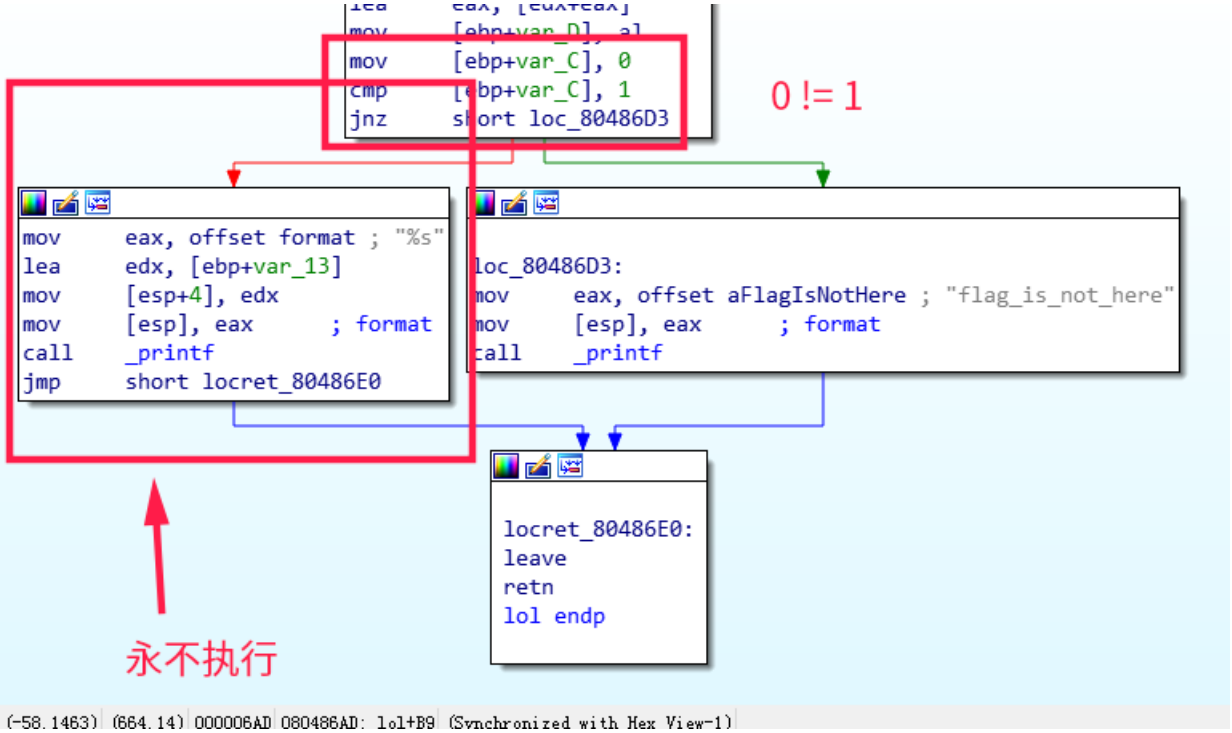
返回值：成功，返回0，否则返回-1。参数数组包含pipe使用的两个文件的描述符。fd[0]:读管道，fd[1]:写管道。

在main函数中先是用pipe建立了一个管道，那么pipedes[1]就是写操作，pipedes[0]就是读操作。调用fork函数生成子进程，在子进程中将“69800876143568214356928753”送入管道中，然后退出子进程，进而在父进程中将管道中的字符串复制到buf变量中，然后入读一个数字，要求这个数字和子进程的PID一致才符合条件。lol函数是生成flag的函数。

```
int __cdecl lol(_BYTE *a1)
{
  char v2; // [esp+15h] [ebp-13h]
  char v3; // [esp+16h] [ebp-12h]
  char v4; // [esp+17h] [ebp-11h]
  char v5; // [esp+18h] [ebp-10h]
  char v6; // [esp+19h] [ebp-Fh]
  char v7; // [esp+1Ah] [ebp-Eh]
  char v8; // [esp+1Bh] [ebp-Dh]

  v2 = 2 * a1[1];
  v3 = a1[4] + a1[5];
  v4 = a1[8] + a1[9];
  v5 = 2 * a1[12];
  v6 = a1[18] + a1[17];
  v7 = a1[10] + a1[21];
  v8 = a1[9] + a1[25];
  return printf("flag_is_not_here");
}
```

这个函数的参数是buf变量，这个函数里面显示操作了这个字符串，最后输出了一句话，感觉不是太合乎情理，于是看汇编代码，发现隐藏了一部分代码：



可以看出来左边的代码是将处理过的字符串输出了出来：

```
current = [0x36,0x39,0x38,0x30,0x30,0x38,0x37,0x36,0x31,0x34,0x33,0x35,0x36,0x38,0x32,0x31,0x34,0x33,0x35,0x36,0x39,0x32,0x38,0x37,0x35,0x33]

print chr(0x39*2),
print chr(current[4]+current[5]),
print chr(current[8]+current[9]),
print chr(current[12]*2),
print chr(current[18]+current[17]),
print chr(current[10]+current[21]),
print chr(current[9]+current[25]),

out:
rhelheg
[Finished in 0.0s]
```

key

该程序是windows上的32位可执行文件，载入IDA中发现main函数中很复杂，通过查找字符串发现了几个相对醒目的字符串：

```
?W?h?a?t h?a?p?p?e?n?
C:\Users\CSAW2016\haha\flag_dir\flag.txt
Congrats You got it!
=W=r=o=n=g=K=e=y=
string too long
```

直接运行程序会返回：

```
➜  ./key.exe
?W?h?a?t h?a?p?p?e?n?
```

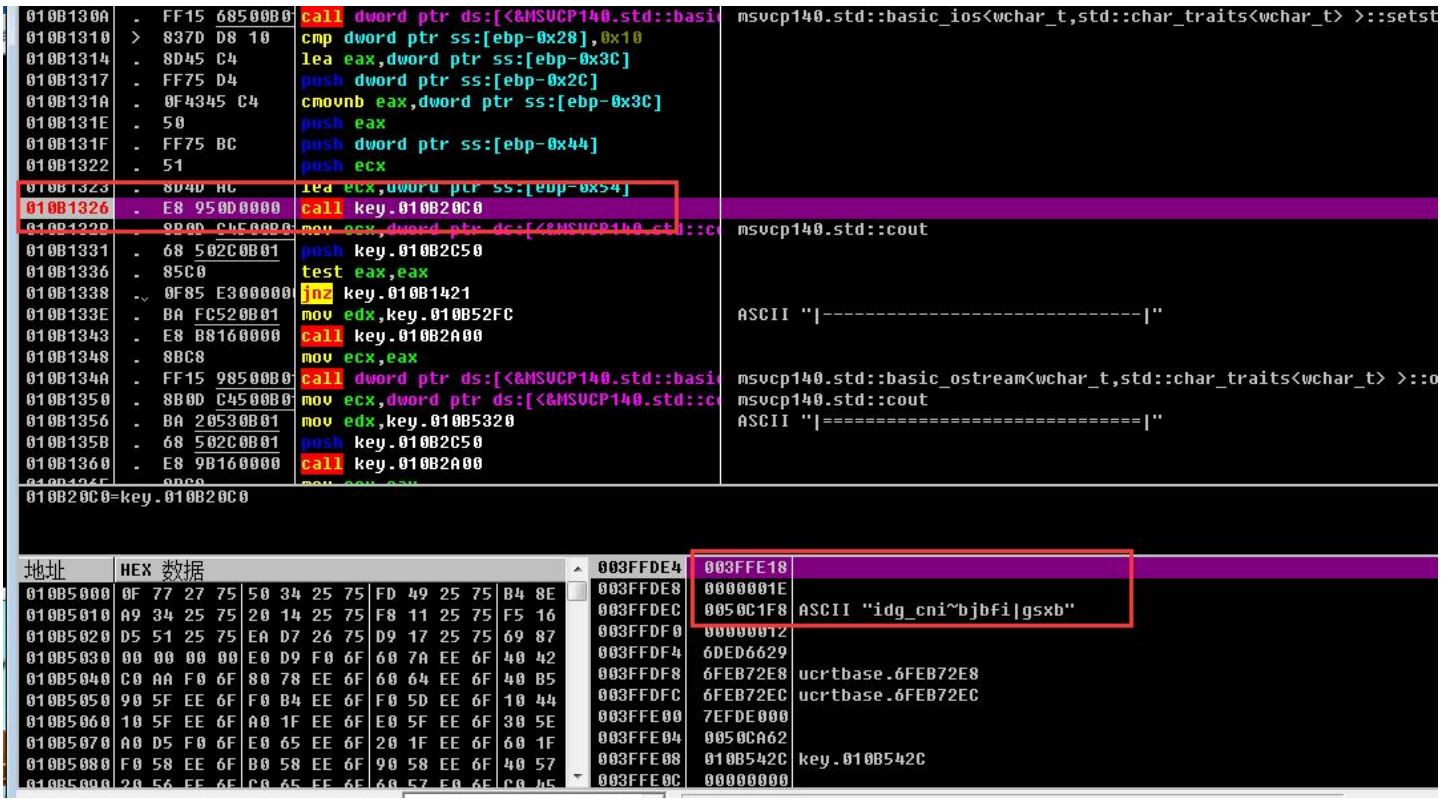
通过“C:\Users\CSAW2016\haha\flag_dir\flag.txt”找到引用这个字符串的地址我们可以发现该程序试图想从这个路径中读取数据，所以我们在本机上创建这个文件，然后再运行程序。

```
➜  ./key.exe
=W=r=o=n=g=K=e=y=
```

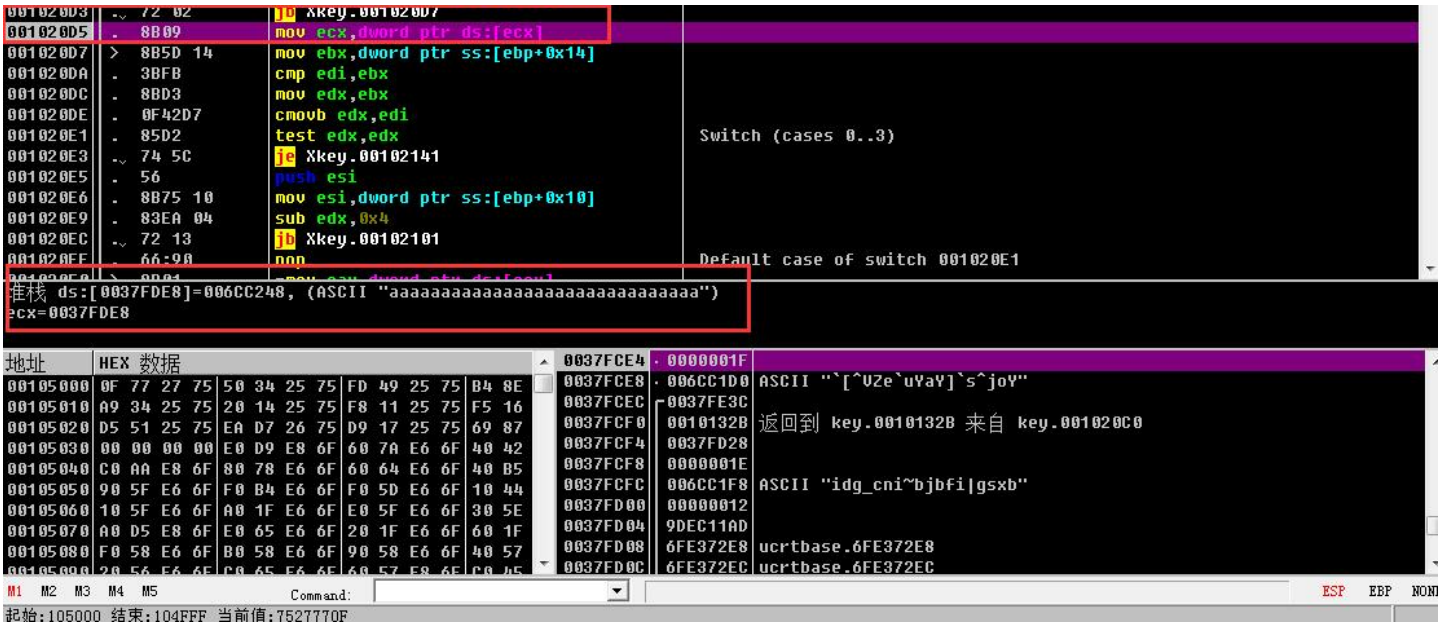
出现的和第一次不一样的输出，然后再看主函数：

```
v13 = sub_1D20C0(&v38, v11, v39, v12, v42);
 v14 = std::cout;
 if ( v13 )
 {
   v22 = "=W=r=o=n=g=K=e=y=";
 }
 else
 {
   v15 = print(std::cout, "|------------------------------|");
   std::basic_ostream<char,std::char_traits<char>>::operator<<(v15, sub_1D2C50);
   v16 = print(std::cout, "|==============================|");
   std::basic_ostream<char,std::char_traits<char>>::operator<<(v16, sub_1D2C50);
   v17 = print(std::cout, "|==============================|");
   std::basic_ostream<char,std::char_traits<char>>::operator<<(v17, sub_1D2C50);
   v18 = print(std::cout, "|==============================|");
   std::basic_ostream<char,std::char_traits<char>>::operator<<(v18, sub_1D2C50);
   v19 = print(std::cout, "\\  /\\  /\\  /\\  /\\==============|");
   std::basic_ostream<char,std::char_traits<char>>::operator<<(v19, sub_1D2C50);
   v20 = print(std::cout, " \\/  \\/  \\/  \\/  \\=============|");
   std::basic_ostream<char,std::char_traits<char>>::operator<<(v20, sub_1D2C50);
   v21 = print(std::cout, "                 |-------------|");
   std::basic_ostream<char,std::char_traits<char>>::operator<<(v21, sub_1D2C50);
   std::basic_ostream<char,std::char_traits<char>>::operator<<(std::cout, sub_1D2C50);
   v14 = std::cout;
   v22 = "Congrats You got it!";
 }
```

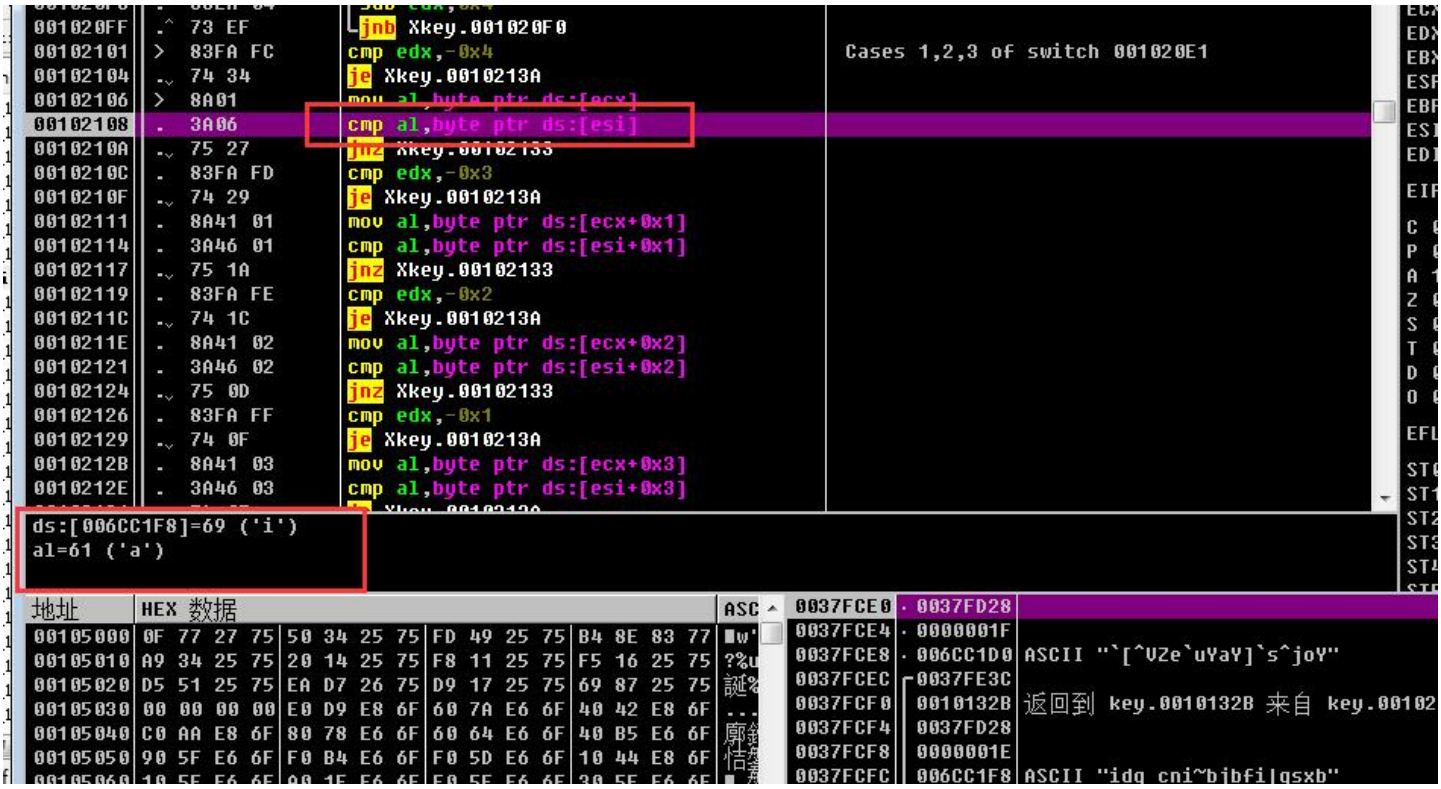
要想使条件成立，那么需要v13不为0，v13是函数sub_1D20C0()的返回值，所以猜测这个函数是检查flag的函数，动态调试一下：



发现在调用这个函数之前，讲一个字符串压入了栈中，其实也就是flag。继续往下调试，发现了我们在文件中写的字符串：



进而发现对比的是我们输出的第一个和刚刚压入栈中字符串的第一个值，所以这里就可以断定刚刚的字符串就是flag。



flag：idg_cni~bjbfi|gsxb

CRACKME

```
data = [0x66,0x31,0x4B,0x33,0x7B,0x63,0x35,0x3A,0x65,0x66,0x6C,0x32,0x31,0x74,0x34,0x3B,0x31,0x74,0x31,0x7A,0x61
,0x78,0x70,0x69,0x6D,0x39,0x7D,0x35,0x2B,0x3F,0x67,0x74,0x75,0x78,0x3B,0x3D,0x76,0x63,0x39,0x76,0x7B,0x76,0x37,0
x2B,0x62,0x75,0x68,0x55,0x7B,0x62,0x54,0x3D,0x2D,0x61,0x6D,0x32,0x71,0x7D,0x3D,0x66,0x68,0x5B,0x78,0x6B,0x7B,0x7
9,0x3F,0x78,0x72,0x71,0x65,0x7B,0x3F,0x7D,0x6C,0x35,0x2D,0x73,0x64,0x32,0x2D,0x4D,0x6F,0x2B,0x3A,0x6A,0x7B,0x39,
0x3D,0x73,0x59,0x5B,0x64,0x61,0x6C,0x76,0x70,0x78,0x3F,0x7A,0x33,0x7B,0x3F,0x6E,0x6F,0x7B,0x5B,0x6B,0x35,0x6C,0x
6C,0x7B,0x7A,0x6A,0x73,0x75,0x35,0x5B,0x6B,0x66,0x6C,0x61,0x2B,0x72,0x36,0x5A,0x67,0x37,0x32,0x6F,0x30,0x73,0x6B
,0x71,0x36,0x63,0x47,0x6C,0x35,0x63,0x77,0x5B,0x3D,0x64,0x3F,0x33,0x76,0x39,0x71,0x35,0x2D,0x76,0x6B,0x6A,0x53,0
x76,0x7B,0x34,0x73,0x71,0x74,0x67,0x3D,0x66,0x30,0x63,0x7A,0x7B,0x2B,0x6A,0x75,0x72,0x6A,0x66,0x6C,0x5B,0x74,0x6
2,0x5D,0x6C,0x72,0x66,0x46,0x31,0x3B,0x32,0x7D,0x75,0x64,0x68,0x62,0x3F,0x30,0x67,0x38,0x7B,0x6F,0x6D,0x3A,0x54,
0x34,0x64,0x68,0x3B,0x7A,0x3A,0x6F,0x7A,0x2D,0x44,0x6E,0x3D,0x6D,0x3D,0x75,0x78,0x3B,0x6F,0x5B,0x67,0x73,0x39,0x
7B,0x2B,0x7A,0x71,0x78,0x2B,0x73,0x71,0x2D,0x64,0x73,0x78,0x63,0x74,0x63,0x76,0x79,0x6B,0x55,0x73,0x32,0x6F,0x64
,0x64,0x72,0x74,0x34,0x33,0x70,0x77,0x76,0x3A,0x66,0x30,0x3B,0x6E,0x6A,0x6B,0x72,0x62,0x39,0x6C,0x6F,0x73,0x36,0
x67,0x30,0x7B,0x69,0x68,0x3F,0x72,0x71,0x61,0x6E,0x74,0x66,0x78,0x24,0x73,0x73,0x6C,0x71,0x64,0x3A,0x72,0x76,0x7
1,0x69,0x78,0x72,0x3B,0x6A,0x7B,0x3F,0x6F,0x3A,0x73,0x6E,0x2B,0x5B,0x69,0x5B,0x79,0x41,0x31,0x31,0x3B,0x67,0x73,
0x6D,0x72,0x38,0x6C,0x6D,0x30,0x3F,0x33,0x7D,0x3B,0x2B,0x69,0x76,0x2B,0x54,0x66,0x3A,0x34,0x47,0x74,0x76,0x32,0x
3A,0x2D,0x32,0x30,0x75,0x70,0x69,0x30,0x5D,0x37,0x3F,0x37,0x37,0x3D,0x3B,0x71,0x7A,0x78,0x7B,0x6D,0x2D,0x57,0x3B
,0x30,0x76,0x74,0x75,0x65,0x68,0x5D,0x6B,0x6F,0x38,0x64,0x3F,0x3D,0x77,0x3A,0x66,0x62,0x68,0x64,0x7B,0x45,0x3A,0
x3B,0x31,0x39,0x3F,0x70,0x3D,0x6B,0x3A,0x62,0x2B,0x7D,0x64,0x6F,0x68,0x74,0x36,0x77,0x70,0x45,0x71,0x2D,0x7A,0x5
D,0x32,0x71,0x62,0x56,0x31,0x7D,0x64,0x68,0x34,0x31,0x36,0x71,0x77,0x39,0x3A,0x78,0x6D,0x5B,0x3B,0x65,0x64,0x3B,
0x3A,0x65,0x63,0x62,0x2D,0x30,0x3A,0x6E,0x69,0x2D,0x73,0x34,0x75,0x32,0x6B,0x66,0x36,0x5D,0x32,0x77,0x6E,0x34,0x
35,0x61,0x6D,0x7A,0x6A,0x72,0x75,0x6E,0x3D,0x6F,0x66,0x6B,0x78,0x2D,0x3D,0x68,0x6D,0x67,0x6F,0x2D,0x6C,0x7A,0x3B
,0x6A,0x39,0x30,0x39,0x3D,0x72,0x6D,0x6F,0x37,0x78,0x63,0x6A,0x34,0x6C,0x65,0x30,0x68,0x78,0x73,0x5B,0x69,0x5D,0
x2D,0x76,0x6A,0x6C,0x5B,0x3F,0x6F,0x31,0x32,0x3A,0x73,0x76,0x34,0x75,0x70,0x69,0x6F,0x37,0x6D,0x61,0x31,0x68,0x5
2,0x79,0x37,0x35,0x35,0x36,0x2B,0x35,0x37,0x6B,0x72,0x65,0x76,0x3A,0x68,0x4C,0x51,0x2B,0x31,0x63,0x78,0x36,0x35,
0x7A,0x35,0x76,0x35,0x5D,0x3B,0x36,0x6E,0x3D,0x5B,0x70,0x38,0x33,0x3B,0x6E,0x3D,0x7B,0x7A,0x6D,0x7B,0x6B,0x32,0x
70]
v5 = 0
flag = ""
for x in range(33):
 flag+=chr(data[v5-1+1])
 v5 = v5+10
print flag
#flag{The-Y3ll0w-turb4ns-Upri$ing}
```

ReverseMe-120

```
import base64
data = "you_know_how_to_remove_junk_code"
# for x in data:
#  print chr(ord(x)^0x25),
print base64.b64encode("\JPzNKJRzMJRzQJzW@HJS@zOPKNzFJA@")
print base64.b64encode("\xD7\x6d\xf8\xe7\xae\xfc")
```

easyre-153

```
data="6980087614356821435692875 3"

flag = data[1]+data[1]+data[4]+data[5]+data[8]+data[9]+data[12]+data[12]+data[18]+data[17]+data[10]+data[21]+dat
a[9]+data[25]
print flag
print chr(ord("9")*2),
print chr(ord("0")+ord("8")),
print chr(ord("1")+ord("4")),
print chr(ord("6")+ord("6")),
print chr(ord("5")+ord("3")),
print chr(ord("3")+ord("2")),
print chr(ord("4")+ord("3")),
#rhelheg
```

notsequence

用IDA打开后发现有两个check函数，第一个check函数是限制表达式的值，第二个check函数是限制表达式的关系

通过C语言来打印出具体的下标值

```
#include<stdio.h>
int main()
{
 int i,v6,v4,v3;
 v6=0;
 for(i=1;i<20;i++)
 {
  v4=0;
  v3=i-1;
  while(19>v3)
  {

   printf("var_%d+", (v3*(v3+1))/2+v6);

   v3++;
  }
  printf("==var_%d\n",(v3*(v3+1))/2 +i );
  v6++;
 }
 return 0;
}

------------------------------------------------------------------------------

#include<stdio.h>
int main()
{
 int i,j,v3,v5;
 v5=0;
 for(i=0;i<=1024;i=(v5*(v5+1))/2)
 {
  v3=0;
  for(j=0;j<=v5;j++)
  {
   printf("%d ",i+j );
  }
  printf(" = %d\n",1<<v5 );
  v5++;
  if(v5==20)
  {
   return 0;
  }
 }
 return 0;
}
```

尝试用Z3来解，发现解不出来，emmm

```
from z3 import *
var_0 = Int('var_0')
……
var_209 = Int('var_209')

solver = Solver()
```

```
solver.add(var_0+var_1+var_3+var_6+var_10+var_15+var_21+var_28+var_36+var_45+var_55+var_66+var_78+var_91+var_105
+var_120+var_136+var_153+var_171==var_191)
solver.add(var_2+var_4+var_7+var_11+var_16+var_22+var_29+var_37+var_46+var_56+var_67+var_79+var_92+var_106+var_1
21+var_137+var_154+var_172==var_192)
solver.add(var_5+var_8+var_12+var_17+var_23+var_30+var_38+var_47+var_57+var_68+var_80+var_93+var_107+var_122+var
_138+var_155+var_173==var_193)
solver.add(var_9+var_13+var_18+var_24+var_31+var_39+var_48+var_58+var_69+var_81+var_94+var_108+var_123+var_139+v
ar_156+var_174==var_194)
solver.add(var_14+var_19+var_25+var_32+var_40+var_49+var_59+var_70+var_82+var_95+var_109+var_124+var_140+var_157
+var_175==var_195)
solver.add(var_20+var_26+var_33+var_41+var_50+var_60+var_71+var_83+var_96+var_110+var_125+var_141+var_158+var_17
6==var_196)
solver.add(var_27+var_34+var_42+var_51+var_61+var_72+var_84+var_97+var_111+var_126+var_142+var_159+var_177==var_
197)
solver.add(var_35+var_43+var_52+var_62+var_73+var_85+var_98+var_112+var_127+var_143+var_160+var_178==var_198)
solver.add(var_44+var_53+var_63+var_74+var_86+var_99+var_113+var_128+var_144+var_161+var_179==var_199)
solver.add(var_54+var_64+var_75+var_87+var_100+var_114+var_129+var_145+var_162+var_180==var_200)
solver.add(var_65+var_76+var_88+var_101+var_115+var_130+var_146+var_163+var_181==var_201)
solver.add(var_77+var_89+var_102+var_116+var_131+var_147+var_164+var_182==var_202)
solver.add(var_90+var_103+var_117+var_132+var_148+var_165+var_183==var_203)
solver.add(var_104+var_118+var_133+var_149+var_166+var_184==var_204)
solver.add(var_119+var_134+var_150+var_167+var_185==var_205)
solver.add(var_135+var_151+var_168+var_186==var_206)
solver.add(var_152+var_169+var_187==var_207)
solver.add(var_170+var_188==var_208)
solver.add(var_189==var_209)

solver.add(var_0==1)
solver.add(var_1+var_2==2)
solver.add(var_3+var_4+var_5==4)
solver.add(var_6+var_7+var_8+var_9==8)
solver.add(var_10+var_11+var_12+var_13+var_14==16)
solver.add(var_15+var_16+var_17+var_18+var_19+var_20==32)
solver.add(var_21+var_22+var_23+var_24+var_25+var_26+var_27==64)
solver.add(var_28+var_29+var_30+var_31+var_32+var_33+var_34+var_35==128)
solver.add(var_36+var_37+var_38+var_39+var_40+var_41+var_42+var_43+var_44==256)
solver.add(var_45+var_46+var_47+var_48+var_49+var_50+var_51+var_52+var_53+var_54==512)
solver.add(var_55+var_56+var_57+var_58+var_59+var_60+var_61+var_62+var_63+var_64+var_65==1024)
solver.add(var_66+var_67+var_68+var_69+var_70+var_71+var_72+var_73+var_74+var_75+var_76+var_77==2048)
solver.add(var_78+var_79+var_80+var_81+var_82+var_83+var_84+var_85+var_86+var_87+var_88+var_89+var_90==4096)
solver.add(var_91+var_92+var_93+var_94+var_95+var_96+var_97+var_98+var_99+var_100+var_101+var_102+var_103+var_10
4==8192)
solver.add(var_105+var_106+var_107+var_108+var_109+var_110+var_111+var_112+var_113+var_114+var_115+var_116+var_1
17+var_118+var_119==16384)
solver.add(var_120+var_121+var_122+var_123+var_124+var_125+var_126+var_127+var_128+var_129+var_130+var_131+var_1
32+var_133+var_134+var_135==32768)
solver.add(var_136+var_137+var_138+var_139+var_140+var_141+var_142+var_143+var_144+var_145+var_146+var_147+var_1
48+var_149+var_150+var_151+var_152==65536)
solver.add(var_153+var_154+var_155+var_156+var_157+var_158+var_159+var_160+var_161+var_162+var_163+var_164+var_1
65+var_166+var_167+var_168+var_169+var_170==131072)
solver.add(var_171+var_172+var_173+var_174+var_175+var_176+var_177+var_178+var_179+var_180+var_181+var_182+var_1
83+var_184+var_185+var_186+var_187+var_188+var_189==262144)
solver.add(var_190+var_191+var_192+var_193+var_194+var_195+var_196+var_197+var_198+var_199+var_200+var_201+var_2
02+var_203+var_204+var_205+var_206+var_207+var_208+var_209==524288)

print solver.check()
print solver.model()
```

然后发现表达式的关系是杨辉三角

```
def yhsj(max):
    n=0
    row = [1]
    while (n<max):
        n+=1
        yield(row)
        row = [1] + [row[k] + row[k + 1] for k in range(len(row) - 1)] + [1]
y=yhsj(20)
for i in y:
    print(i)


 out:
 [1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]
[1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]
[1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]
[1, 12, 66, 220, 495, 792, 924, 792, 495, 220, 66, 12, 1]
[1, 13, 78, 286, 715, 1287, 1716, 1716, 1287, 715, 286, 78, 13, 1]
[1, 14, 91, 364, 1001, 2002, 3003, 3432, 3003, 2002, 1001, 364, 91, 14, 1]
[1, 15, 105, 455, 1365, 3003, 5005, 6435, 6435, 5005, 3003, 1365, 455, 105, 15, 1]
[1, 16, 120, 560, 1820, 4368, 8008, 11440, 12870, 11440, 8008, 4368, 1820, 560, 120, 16, 1]
[1, 17, 136, 680, 2380, 6188, 12376, 19448, 24310, 24310, 19448, 12376, 6188, 2380, 680, 136, 17, 1]
[1, 18, 153, 816, 3060, 8568, 18564, 31824, 43758, 48620, 43758, 31824, 18564, 8568, 3060, 816, 153, 18, 1]
[1, 19, 171, 969, 3876, 11628, 27132, 50388, 75582, 92378, 92378, 75582, 50388, 27132, 11628, 3876, 969, 171, 19
, 1]
```

最后将md5加密即可

```
11112113311464115101051161520156117213535217118285670562881193684126126843691110451202102522101204510111155165330462462336016555111112662204957929247924952206612111378286715128717161716128771528678131114913641001200230033432300320021001364911411151054551365300350056435643550053003136545510515111612056018204368800811440128701144080084368182056012016111713668023806188123761944824310243101944812376618823806801361711181538163060856818564318244375848620437583182418564856830608161531811191719693876116282713250388755829237892378755825038827132116283876969171191
```

reverse-box

在平台上题目描述不全，少了一个重要的提示信息

原题题目描述

```
$ ./reverse_box ${FLAG}
95eeaf95ef94234999582f722f492f72b19a7aaf72e6e776b57aee722fe77ab5ad9aaeb156729676ae7a236d99b1df4a
```

也就是说，当输入争取的flag的时候，程序应该输出的是这一个字符串（如果少了这个信息，我请问怎么做，mmp）

主函数很简单：

```
int __cdecl main(int a1, char **a2)
{
  size_t i; // [esp+18h] [ebp-10Ch]
  int v4; // [esp+1Ch] [ebp-108h]
  unsigned int v5; // [esp+11Ch] [ebp-8h]

  v5 = __readgsdword(0x14u);
  if ( a1 <= 1 )
  {
    printf("usage: %s flag\n", *a2);
    exit(1);
  }
  make_boxs(&v4);                               // 生成box数据
  for ( i = 0; i < strlen(a2[1]); ++i )
    printf("%02x", *((unsigned __int8 *)&v4 + a2[1][i]));
  putchar(10);
  return 0;
}
```

会根据我们的输入当成索引在box里面输出对应的值

make_boxs函数：

```
int __cdecl sub_804858D(_BYTE *a1)
{
  unsigned int v1; // eax
  int v2; // edx
  char v3; // al
  char v4; // ST1B_1
  char v5; // al
  int result; // eax
  unsigned __int8 v7; // [esp+1Ah] [ebp-Eh]
  char v8; // [esp+1Bh] [ebp-Dh]
  char v9; // [esp+1Bh] [ebp-Dh]
  int v10; // [esp+1Ch] [ebp-Ch]

  v1 = time(0);
  srand(v1);
  do
    v10 = (unsigned __int8)rand();
  while ( !v10 );
  *a1 = v10;
  v7 = 1;
  v8 = 1;
  do
  {
    v2 = v7 ^ 2 * v7;
    if ( (v7 & 0x80u) == 0 )
      v3 = 0;
    else
      v3 = 27;
    v7 = v2 ^ v3;
    v4 = 4 * (2 * v8 ^ v8) ^ 2 * v8 ^ v8;
    v9 = 16 * v4 ^ v4;
    if ( v9 >= 0 )
      v5 = 0;
    else
      v5 = 9;
    v8 = v9 ^ v5;
    result = (unsigned __int8)__ROR1__(v8, 4) ^ (unsigned __int8)__ROR1__(v8, 5) ^ (unsigned __int8)__ROR1__(v8, 6) ^ (unsigned __int8)__ROR1__(v8, 7) ^ (unsigned __int8)(v8 ^ *a1);
    a1[v7] = result;
  }
  while ( v7 != 1 );
  return result;
}
```

根据时间中下srand的种子，然后生成一个随机数，然后把随机生成的这个数字带进去生成boxs的数据，可以看到，程序中将随机生成的种子转成了unsigned __int8类型，这个类型只有一个字节大小，所以范围是0～255,所以我们可以爆破一下，flag开头是“T”，对应的是0x95

第一处断点：0x80485b4

```
.text:080485B4                 cmp     [ebp+var_C], 0
.text:080485B8                 jz      short loc_80485A7
.text:080485BA                 mov     eax, [ebp+var_C]
```

断下之后将ebp-0xc的值变成0~255

第二处断点：0x8048704

```
.text:080486FF                 movzx   eax, byte ptr [esp+eax+1Ch]
.text:08048704                 movzx   eax, al
.text:08048707                 mov     [esp+4], eax
.text:0804870B                 mov     dword ptr [esp], offset a02x ; "%02x"
```

与输出的字符和0x95比较，若相等，则本次的box数据是正确的，将全部的数据输出出来

```
Breakpoint 2, 0x08048704 in ?? ()
$1 = 214
0xffffd04c: 0xd6 0xc9 0xc2 0xce 0x47 0xde 0xda 0x70
0xffffd054: 0x85 0xb4 0xd2 0x9e 0x4b 0x62 0x1e 0xc3
0xffffd05c: 0x7f 0x37 0x7c 0xc8 0x4f 0xec 0xf2 0x45
0xffffd064: 0x18 0x61 0x17 0x1a 0x29 0x11 0xc7 0x75
0xffffd06c: 0x02 0x48 0x26 0x93 0x83 0x8a 0x42 0x79
0xffffd074: 0x81 0x10 0x50 0x44 0xc4 0x6d 0x84 0xa0
0xffffd07c: 0xb1 0x72 0x96 0x76 0xad 0x23 0xb0 0x2f
0xffffd084: 0xb2 0xa7 0x35 0x57 0x5e 0x92 0x07 0xc0
0xffffd08c: 0xbc 0x36 0x99 0xaf 0xae 0xdb 0xef 0x15
0xffffd094: 0xe7 0x8e 0x63 0x06 0x9c 0x56 0x9a 0x31
0xffffd09c: 0xe6 0x64 0xb5 0x58 0x95 0x49 0x04 0xee
0xffffd0a4: 0xdf 0x7e 0x0b 0x8c 0xff 0xf9 0xed 0x7a
0xffffd0ac: 0x65 0x5a 0x1f 0x4e 0xf6 0xf8 0x86 0x30
0xffffd0b4: 0xf0 0x4c 0xb7 0xca 0xe5 0x89 0x2a 0x1d
0xffffd0bc: 0xe4 0x16 0xf5 0x3a 0x27 0x28 0x8d 0x40
0xffffd0c4: 0x09 0x03 0x6f 0x94 0xa5 0x4a 0x46 0x67
0xffffd0cc: 0x78 0xb9 0xa6 0x59 0xea 0x22 0xf1 0xa2
0xffffd0d4: 0x71 0x12 0xcb 0x88 0xd1 0xe8 0xac 0xc6
0xffffd0dc: 0xd5 0x34 0xfa 0x69 0x97 0x9f 0x25 0x3d
0xffffd0e4: 0xf3 0x5b 0x0d 0xa1 0x6b 0xeb 0xbe 0x6e
0xffffd0ec: 0x55 0x87 0x8f 0xbf 0xfc 0xb3 0x91 0xe9
0xffffd0f4: 0x77 0x66 0x19 0xd7 0x24 0x20 0x51 0xcc
0xffffd0fc: 0x52 0x7d 0x82 0xd8 0x38 0x60 0xfb 0x1c
0xffffd104: 0xd9 0xe3 0x41 0x5f 0xd0 0xcf 0x1b 0xbd
0xffffd10c: 0x0f 0xcd 0x90 0x9b 0xa9 0x13 0x01 0x73
0xffffd114: 0x5d 0x68 0xc1 0xaa 0xfe 0x08 0x3e 0x3f
0xffffd11c: 0xc5 0x8b 0x00 0xd3 0xfd 0xb6 0x43 0xbb
0xffffd124: 0xd4 0x80 0xe2 0x0c 0x33 0x74 0xa8 0x2b
0xffffd12c: 0x54 0x4d 0x2d 0xa4 0xdc 0x6c 0x3b 0x21
0xffffd134: 0x2e 0xab 0x32 0x5c 0x7b 0xe0 0x9d 0x6a
0xffffd13c: 0x39 0x14 0x3c 0xb8 0x0a 0x53 0xf7 0xdd
0xffffd144: 0xf4 0x2c 0x98 0xba 0x05 0xe1 0x0e 0xa3
```

最后将flag还原出来：

```
data = [0xd6,0xc9,0xc2,0xce,0x47,0xde,0xda,0x70,0x85,0xb4,0xd2,0x9e,0x4b,0x62,0x1e,0xc3,0x7f,0x37,0x7c,0xc8,0x4f
,0xec,0xf2,0x45,0x18,0x61,0x17,0x1a,0x29,0x11,0xc7,0x75,0x02,0x48,0x26,0x93,0x83,0x8a,0x42,0x79,0x81,0x10,0x50,0
x44,0xc4,0x6d,0x84,0xa0,0xb1,0x72,0x96,0x76,0xad,0x23,0xb0,0x2f,0xb2,0xa7,0x35,0x57,0x5e,0x92,0x07,0xc0,0xbc,0x3
6,0x99,0xaf,0xae,0xdb,0xef,0x15,0xe7,0x8e,0x63,0x06,0x9c,0x56,0x9a,0x31,0xe6,0x64,0xb5,0x58,0x95,0x49,0x04,0xee,
0xdf,0x7e,0x0b,0x8c,0xff,0xf9,0xed,0x7a,0x65,0x5a,0x1f,0x4e,0xf6,0xf8,0x86,0x30,0xf0,0x4c,0xb7,0xca,0xe5,0x89,0x
2a,0x1d,0xe4,0x16,0xf5,0x3a,0x27,0x28,0x8d,0x40,0x09,0x03,0x6f,0x94,0xa5,0x4a,0x46,0x67,0x78,0xb9,0xa6,0x59,0xea
,0x22,0xf1,0xa2,0x71,0x12,0xcb,0x88,0xd1,0xe8,0xac,0xc6,0xd5,0x34,0xfa,0x69,0x97,0x9f,0x25,0x3d,0xf3,0x5b,0x0d,0
xa1,0x6b,0xeb,0xbe,0x6e,0x55,0x87,0x8f,0xbf,0xfc,0xb3,0x91,0xe9,0x77,0x66,0x19,0xd7,0x24,0x20,0x51,0xcc,0x52,0x7
d,0x82,0xd8,0x38,0x60,0xfb,0x1c,0xd9,0xe3,0x41,0x5f,0xd0,0xcf,0x1b,0xbd,0x0f,0xcd,0x90,0x9b,0xa9,0x13,0x01,0x73,
0x5d,0x68,0xc1,0xaa,0xfe,0x08,0x3e,0x3f,0xc5,0x8b,0x00,0xd3,0xfd,0xb6,0x43,0xbb,0xd4,0x80,0xe2,0x0c,0x33,0x74,0x
a8,0x2b,0x54,0x4d,0x2d,0xa4,0xdc,0x6c,0x3b,0x21,0x2e,0xab,0x32,0x5c,0x7b,0xe0,0x9d,0x6a,0x39,0x14,0x3c,0xb8,0x0a
,0x53,0xf7,0xdd,0xf4,0x2c,0x98,0xba,0x05,0xe1,0x0e,0xa3]

index = "95eeaf95ef94234999582f722f492f72b19a7aaf72e6e776b57aee722fe77ab5ad9aaeb156729676ae7a236d99b1df4a"
#https://docs.microsoft.com/zh-cn/previous-versions/s3f49ktz(v=vs.120)
list1 = []
for x in range(0,len(index),2):
 list1.append(eval("0x"+index[x:x+2]))

flag = ""
for x in range(len(list1)):
 flag+=chr(data.index(list1[x]))
print flag
```