广东省强网杯逆向题 SimpleGame writeup

原创

ptyin1604 于 2018-09-06 21:05:07 发布 899 收藏 2

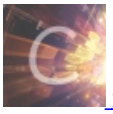
分类专栏： ctf 逆向 逆向工程 安全 信息安全 文章标签： ctf 逆向 逆向工程 信息安全 安全


ctf 同时被 3 个专栏收录

1 篇文章 0 订阅
订阅专栏


逆向

1 篇文章 0 订阅
订阅专栏

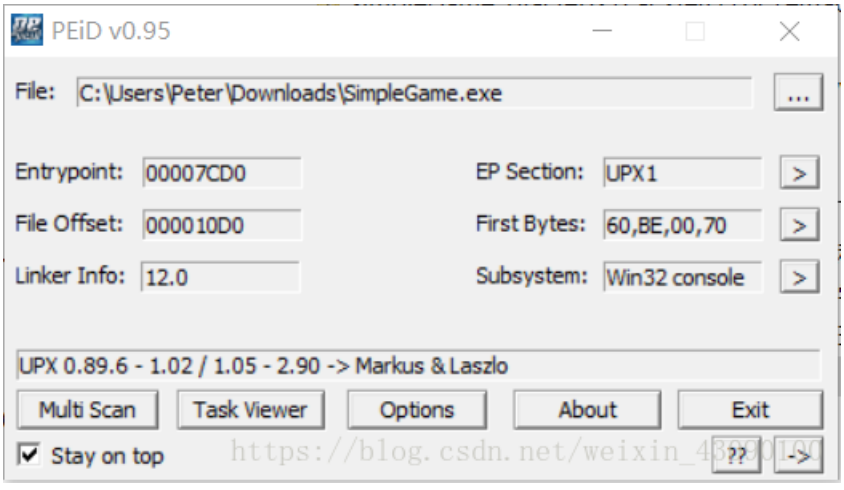

逆向工程

1 篇文章 0 订阅
订阅专栏

在i春秋平台上找到的这题，200分的题就是不一样哈：）

地址：reverse 题目SimpleGame

先运行一下，看起来还挺简单的（然而虐了我好长时间），输入flag，然后给出结果。加载到ida，一上来发现有壳。然后就查了下壳：upx



脱壳一开始用的esp定律，因为在程序开头发现一个pusha，然后alt+t搜索popa

```
UPX1:00407E4D                 popa
UPX1:00407E4E                 lea     eax, [esp+2Ch+var_AC]
UPX1:00407E52
UPX1:00407E52 loc_407E52:                             ; CODE XREF: start+186↓j
UPX1:00407E52                 push    0
UPX1:00407E54                 cmp     esp, eax
UPX1:00407E56                 jnz     short loc_407E52
```

```
UPX1:00407E58                   sub     esp, 0FFFFFF80h
UPX1:00407E5B                   jmp     near ptr byte_401779
UPX1:00407E5B start             endp ; sp-analysis failed
UPX1:00407E5B
```

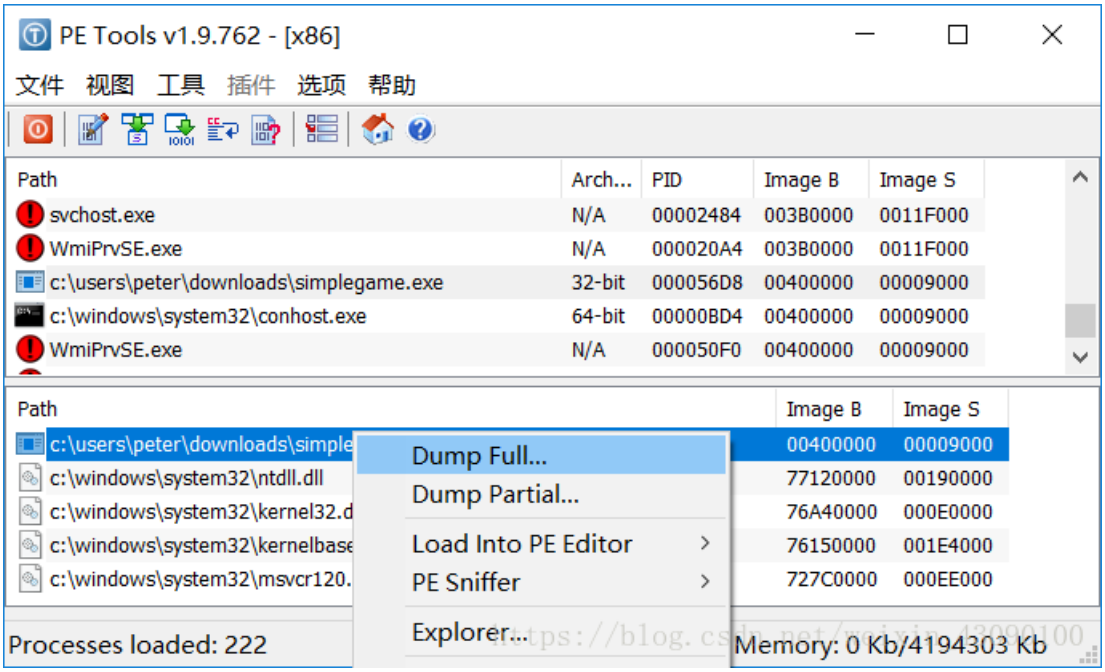

跟进到jmp目的地址发现应该还不是oep，继续跟进 直到一个call指令 f7单步步入到了401400

```
UPX0:00401400
UPX0:00401400 sub_401400 proc near                    ; CODE XREF: UPX0:0040170C↓p
UPX0:00401400
UPX0:00401400 var_2C= byte ptr -2Ch
UPX0:00401400 var_2B= xmmword ptr -2Bh
UPX0:00401400 var_1B= xmmword ptr -1Bh
UPX0:00401400 var_B= dword ptr -0Bh
UPX0:00401400 var_7= byte ptr -7
UPX0:00401400 var_4= dword ptr -4
UPX0:00401400
UPX0:00401400 push    ebp
UPX0:00401401 mov     ebp, esp
UPX0:00401403 sub     esp, 2Ch
UPX0:00401406 mov     eax, ___security_cookie
UPX0:0040140B xor     eax, ebp
UPX0:0040140D mov     [ebp+var_4], eax
UNKNOWN 00401400: sub_401400 (Synchronized with EIP)
```


应该是oep无疑了（ps：在分析的时候会发现这个程序用了security cookie的保护形式，是因为当读入flag的时候没有做字符数限制，可以触发栈溢出，当然这和这道题并无啥卵关系）

oep找到了，那就来脱壳吧

这里我用pe tools来dump的，importREConstructor来修复IAT，大家可以在52破解上找到资源



在401400往下拉一些就能找到一个printf函数的call

```
UPX0:00401446 push    esi
UPX0:00401447 call    sub_4013C0
UPX0:0040144C mov     esi, off_40209C
UPX0:00401452 push    offset aGameIsStart             ; "Game Is Start\n"
UPX0:00401457 call    esi ; msvcr120_printf
UPX0:00401459 push    offset aPleaseInputASt          ; "Please Input a String\n"
```

```
UPX0:0040145E call    esi ; msvcr120_printf
UPX0:00401460 call    off_4020A8
UPX0:00401466 push    eax
UPX0:00401467 lea     eax, [ebp+var_2C]
```



```
00401447 call    sub_4013C0
0040144C mov     esi, off_40209C
00401452 push    offset aGameIsStart             ; "Game I
00401457 call    esi ; msvcr120_printf
00401459 push    offset aPleaseInputASt          ; "Please
0040145E call    esi ; msvcr120_printf
```


这里跟到esi的地址，找到IAT

```
00402000  A0 4D A5 76 60 52 A5 76  60 85 A5 76 10 F5 AA 76  燻· ·`R ·`瑗·v.酹·v
00402010  D0 5A A5 76 90 56 A5 76  60 56 A5 76 10 E8 17 77  衄· ·恢· ·`V ·...w
00402020  50 DE 17 77 00 00 00 00  8D BB 83 72 5F E2 7E 72  P..w....岛·僨·_鈿·r
00402030  CE C7 7E 72 93 42 7F 72  B8 BB 83 72 04 41 7F 72  吻·~r揣·.r富·僨·.A.r
00402040  EB 55 86 72 E9 B9 83 72  86 CC 7D 72 50 CC 7D 72  險·哞·楣·僨·喬·}rP蘿·r
00402050  2C F6 89 72 40 F7 89 72  ED 4C 86 72 B8 6B 86 72  ,鰤·r@鲅·r鞮·哞·玱·哞·
00402060  0C 48 86 72 F7 47 86 72  2C DC 85 72 DB C7 7E 72  .H哞·鱔·哞·,魑·r矍·~r
00402070  D7 ED 7C 72 FC ED 7C 72  08 12 7D 72 46 CA 7D 72  醉·|r ·|r..}rF藐·r
00402080  6B BE 7E 72 9B 46 86 72  B5 C9 7E 72 AA 2A 7D 72  k緙·r汫·哞·瞪·~r..}r
00402090  9E 4F 7D 72 BC FD 7E 72  DE F9 7E 72 D9 2F 84 72  瀹·}r箭·~r搦·~r..则·
004020A0  C3 15 84 72 D9 1D 84 72  B5 42 7D 72 38 F6 89 72  ..则·..则·磔·}r8鰤·r
004020B0  33 15 7D 72 00 00 00 00  00 00 00 00 C9 15 40 00  3.}r........@.
```


起始地址，终止地址我都用粉色标了出来，然后就是importREConstructor

然后最终修复文件是dumped_.exe

接下来就是愉快的逆向分析

```
UPX0:0040142C                   call    sub_401200
UPX0:00401431                   cmp     eax, 0FFFFFFFFh
UPX0:00401434                   jmp     short loc_401446
```

注意一下粉色标出来的地方本来是 jnz，但是如果这样的话直接运行就中断了，所以在这里打了一个补丁，修改成了jmp。
下面开始真分析

```
if ( strlen(&input) == 36 )
{
  if ( a_to_z(&input) == -1 )
  {
    printf("Wrong Format!\n");
  }
```

长度是36，输入的字符必须是小写字母。

```
if ( sub_401000(&input) != -1 && sub_4012C0() != -1 && sub_401300() != -1 )
{
  printf("You Win!\n");
  return 0;
}
```

这里有三个检查函数，分别看一下
你会发现第一个函数代码量远远大于第二三个，所以就先从第一个函数开始

```
signed int __thiscall sub_401000(const char *this)
{
  signed int i; // ebx
  const char *input_1; // esi
  int char_1; // ecx
  unsigned int row; // esi
  int _2rd; // ecx
  unsigned int column; // ecx
  int _2; // esi
  int v8; // edi
  int *a; // ebx
  signed int k; // ecx
  int *a_1; // edx
  unsigned int i_1; // [esp+Ch] [ebp-Ch]
  signed int j; // [esp+10h] [ebp-8h]
  const char *input; // [esp+14h] [ebp-4h]

  input = this;
  i = 0;
  i_1 = 0;
  if ( !strlen(this) )
    return 1;
  input_1 = this;
  while ( 1 )
  {
    char_1 = input_1[i];
    if ( char_1 != dword_403170[i / 2] )        // i = 0,4,8,12,16,20,24,28,32
                                                // i/2 = 0,2,4,6,8,10,12,14,16
      return -1;
    switch ( char_1 )
    {
00001000 sub_401000:1 (401000)
```

```
_0 = input_1[i];
if ( _0 != dword_403170[i / 2] )
  return -1;
```

```
if ( _2 != dword_403170[(i + 2) / 2] )
  return -1;
```

这里我分别用_0 、_1代表四个字符中第0个和第1个，以此类推。 403170地址上如果在动态调试之前是这样的

```
:00403170 dword_403170    dd '&', 2 dup('1'), '&', '5', '#', '1', '#', '5', '#'
:00403170                                     ; DATA XREF: sub_401000+3B↑r
:00403170                                     ; sub_401000+7E↑r ...
:00403170                 dd '5', 2 dup('#'), '1', '5', '&'
:004031B0 dword_4031B0    dd '&', '5'         ; DATA XREF: sub_4013C0+1F↑o
```

显然不对，动态调试到断点，在跟到403170地址上，dump下来的数组是这样的

```
d...s...s...d...
w...a...s...a...
w...a...w...a...
a...s...w...d...
d...w...........
```

所以可以知道 list_0 = ‘dswswwawd’，list_2=’sdaaaasdw’。
下面是个switch语句： _0 = ‘a’, _0=’d’情况如下

```
case 'a':
  column = 100 - input_1[i + 1];
  goto LABEL_14;
case 'd':
  column = input_1[i + 1] - 94;
LABEL_14:
  _2 = input_1[i + 2];
  if ( _2 != dword_403170[(i + 2) / 2] )
    return -1;
  if ( _2 == 's' )
  {
    row = input[i + 3] - 95;
  }
  else
  {
    if ( _2 != 'w' )
      return -1;
    row = 99 - input[i + 3];
  }
```

在这种情况下，_2只能是s或者w
再往下翻一些是_0 = ‘s’, _0=’w’情况，可以和上面相类比：

```
  case 's':
    row = input_1[i + 1] - 95;
    goto LABEL_6;
  case 'w':
    row = 99 - input_1[i + 1];
LABEL_6:
    _2rd = input[i + 2];
    if ( _2rd != dword_403170[(i + 2) / 2] )
      return -1;
    if ( _2rd == 'a' )
    {
      column = 100 - input[i + 3];
    }
    else
    {
```

```
    if ( _2ra != 'a' )
      return -1;
    column = input[i + 3] - 94;
  }
  goto LABEL_19;
```


这里我定义了两个变量，row和column，这么命名是有原因的，现在还不能看出来row和column的意义，但是知道row和column与_1和_3相关。

```
LABEL_19:
        if ( row > 5 || column > 5 )
          return -1;
        j = 0;
        v8 = row + column + 8 * row;            /
        a = &matrix_A[16 * (i / 4)];
        do
        {
          k = 0;
          a_1 = a;
          do
          {
            if ( *a_1 )
            {
              if ( matrix_B[v8 + k] )
                return -1;
              matrix_B[v8 + k] = *a_1;
            }
            ++k;
            ++a_1;
          }
          while ( k < 4 );
          a += 4;
          v8 += 9;
          ++j;
        }
        while ( j < 4 );
        input_1 = input;
        i_1 += 4;
        i = i_1;
        if ( i_1 >= strlen(input) )
```


粉色标出来的地方均可表明又是恶心的二维数组， 相对应的命名了两个二维矩阵。可以根据循环次数推出matrix_A其实是9个4*4的二维数组， 跟进到matrix_B地址可以发现其是一个9*9的二维数组。分别将A和B从hex-view里抠出来，这里就不放图了。在最底下的解题脚本里会给出生成矩阵的代码

A:

0090
0990
0090
0000

0600
0600
0600
0600

0000
0300
3300
3000

0080
0080
0880
0000
……
B:

```
fffff00ff
f0000f0ff
f0ffff00f
0000fff0f
0f00f0fff
f0ff00f0f
f00f0000f
ff0000f0f
fffffffff
```

这里的代码逻辑可以这么思考A和B是0的位置就相当于一个空格，非0就是有东西。将A中的第i个小矩阵嵌入B矩阵中，以（row，column）坐标为起点4*4的小矩阵中，如果有任何一个重合位置两个小矩阵都有东西，就return -1。也就是说所有重叠位置均只能有至多一个东西不能有两个东西。然后还要将A中第i个小矩阵嵌入到B矩阵中，下一次判断的B矩阵就是第i个小矩阵嵌入后的矩阵。

e.g.

拿A中第一个小矩阵为例，将其嵌入到B矩阵

B:

```
fffff00ff
f0000f0ff
f0ffff00f
0000fff0f
0f00f0fff
f0ff00f0f
f00f0000f
ff0000f0f
fffffffff
```

A:

0090
0990
0090
0000

有三个坐标可以实现不重叠嵌入，分别是(4, 3), (5, 3), (5, 5)【ps:坐标形式是 (row，column)】

不妨试一试

但我们知道答案应该是唯一的，但第一步这就出现了三种情况怎么办，这时候需要解题脚本一个dfs函数，搜索出能使A的9个小矩阵都能嵌入到B中的情况。

前面写过每一个row和column都是由_1和_3生成的，生成规则很简单无非就是加减法，这里并不是难点。所以从row和column倒推出_1和_3也是很容易。

总结一下我们的解题脚本需要的功能：

1、由ida中dump出来的数据生成矩阵A和矩阵B

2、用深搜+回溯实现矩阵A嵌入到矩阵B，记录被嵌入的B中4*4小矩阵的起点坐标(row，column)

3、一直每个起点坐标，反向推出每个_1和_3

以下是我写的py脚本，代码量比较大，建议分片来看

```
    ans = []
```

```
ans = []

def dfs(step):
    if step == 9:
        print(ans)
    else:
        pos = []
        for row in range(6):
            for column in range(6):
                sum = 0
                for i in range(4):
                    for j in range(4):
                        if a[16 * step + 4 * i + j]:
                            if b[9 * (i + row) + j + column]:
                                break
                            sum += 1
                if sum == 16:
                    pos.append((row, column))
        for position in pos:
            for i in range(4):
                for j in range(4):
                    if a[16 * step + 4 * i + j]:
                        b[9 * (i + position[0]) + j + position[1]] = a[16 * step + 4 * i + j]
            ans.append(position)
            dfs(step+1)
            for i in range(4):
                for j in range(4):
                    if a[16 * step + 4 * i + j]:
                        b[9 * (i + position[0]) + j + position[1]] = 0
            del ans[step]


A = '''00 00 00 00 00 00 00 00  09 00 00 00 00 00 00 00
00 00 00 00 09 00 00 00  09 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  09 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 06 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 06 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 06 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 06 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 03 00 00 00  00 00 00 00 00 00 00 00
03 00 00 00 03 00 00 00  00 00 00 00 00 00 00 00
03 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  08 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  08 00 00 00 00 00 00 00
00 00 00 00 08 00 00 00  08 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 05 00 00 00  05 00 00 00 00 00 00 00
00 00 00 00 05 00 00 00  05 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
02 00 00 00 02 00 00 00  02 00 00 00 02 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 07 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 07 00 00 00  07 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  07 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

```
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  04 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  04 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 01 00 00 00  01 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00'''
B = '''0F 00 00 00 0F 00 00 00  0F 00 00 00 0F 00 00 00
0F 00 00 00 00 00 00 00  00 00 00 00 0F 00 00 00
0F 00 00 00 0F 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  0F 00 00 00 00 00 00 00
0F 00 00 00 0F 00 00 00  0F 00 00 00 00 00 00 00
0F 00 00 00 0F 00 00 00  0F 00 00 00 0F 00 00 00
00 00 00 00 00 00 00 00  0F 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 0F 00 00 00
0F 00 00 00 0F 00 00 00  00 00 00 00 0F 00 00 00
00 00 00 00 0F 00 00 00  00 00 00 00 00 00 00 00
0F 00 00 00 00 00 00 00  0F 00 00 00 0F 00 00 00
0F 00 00 00 0F 00 00 00  00 00 00 00 0F 00 00 00
0F 00 00 00 00 00 00 00  00 00 00 00 0F 00 00 00
00 00 00 00 0F 00 00 00  0F 00 00 00 00 00 00 00
00 00 00 00 0F 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  0F 00 00 00 0F 00 00 00
0F 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 0F 00 00 00  00 00 00 00 0F 00 00 00
0F 00 00 00 0F 00 00 00  0F 00 00 00 0F 00 00 00
0F 00 00 00 0F 00 00 00  0F 00 00 00 0F 00 00 00
0F 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00'''
#  ----------------------------------------init----------------------------------------
a = A.split('\n')
b = B.split('\n')
a_1 = []
b_1 = []
for i in range(len(a)):
    temp = a[i].split('  ')
    a_1 += temp[0].split(' ')+temp[1].split(' ')
for i in range(len(b)):
    temp = b[i].split('  ')
    b_1 += temp[0].split(' ') + temp[1].split(' ')
print(a_1, b_1)
a = []
b = []
for i in range(0, len(a_1), 4):
    a.append(int(a_1[i]))
for i in range(0, len(b_1), 4):
    b.append(eval('0x'+b_1[i]))
#  ------------------------------------------------------------------------------------
'''print('\n\n', a, b, sep='\n')'''
#  ----------------------------------generate_matrix----------------------------------
for i in range(9):
    for j in range(4):
        for k in range(4):
            print(a[16*i+4*j+k], end='')
        print('\n', end='')
    print('\n', end='')
for row in range(9):
    for column in range(9):
        if b[9*row+column] == 15:
            print('f', end='')
```

```
            else:
                print(b[9*row+column], end='')
        print('\n', end='')
#  ----------------------------------------------------------------------------
order = [9, 6, 3, 8, 5, 2, 7, 4, 1]
dfs(0)
ans = [(5, 5, 9), (4, 4, 6), (1, 0, 3), (5, 2, 8), (2, 1, 5), (0, 1, 2), (5, 0, 7), (1, 5, 4), (0, 4, 1
_0 = 'dswswwawd'
_1 = []
_2 = 'sdaaaasdw'
_3 = []
row = 0
column = 1
for i in range(len(_0)):
    input_1 = -1
    input_3 = -1
    if _0[i] == 'a':
        input_1 = chr(100-ans[i][column])
    if _0[i] == 'd':
        input_1 = chr(ans[i][column]+94)
    if _0[i] == 's':
        input_1 = chr(ans[i][row]+95)
    if _0[i] == 'w':
        input_1 = chr(99-ans[i][row])
    _1.append(input_1)
    if _0[i] == 'a' or _0[i] == 'd':
        if _2[i] == 's':
            input_3 = chr(ans[i][row]+95)
        if _2[i] == 'w':
            input_3 = chr(99-ans[i][row])
    else:
        if _2[i] == 'a':
            input_3 = chr(100-ans[i][column])
        if _2[i] == 'd':
            input_3 = chr(ans[i][column]+94)
    _3.append(input_3)
for i in range(len(_0)):
    print(_0[i]+_1[i]+_2[i]+_3[i], end='')
#  input='dcsdscdbwbadsdabwaacwcacadsdwbdcdbwc'
```

最后的flag别忘加上flag{}

flag：flag{dcsdscdbwbadsdabwaacwcacadsdwbdcdbwc}

在脱壳的时候我用的是ida脱的壳，因为在用ollydbg的时候pause在了奇怪的地方

call edx；edx的地址超出所有的段

但是ida却没有这个情况，是不是大家在做这个题的时候也遇到了这种情况，所以还想请教一下大神们这是什么原因，如果可以的话希望可以有评论