看雪CTF.TSRC 2018 团队赛-第六题 追凶者也--拼图游戏

原创

大帅锅1 于 2018-12-20 10:04:17 发布 1407 收藏
分类专栏： CTF 文章标签： 看雪1 ctf TSRC 追凶者也 2018

1.思路

先看下前面大佬答题的时间，

1. pizzatql	100分 2448s
2. OneName	100分 2473s
3. AceHub	100分 5203s
4. fade-vivi	100分 6320s
5. 萌新队	100分 6359s
6. tekkens	100分 6499s

有点变态的，不到一个小时就弄完了！说明可能有点希望 0.0！！！！！

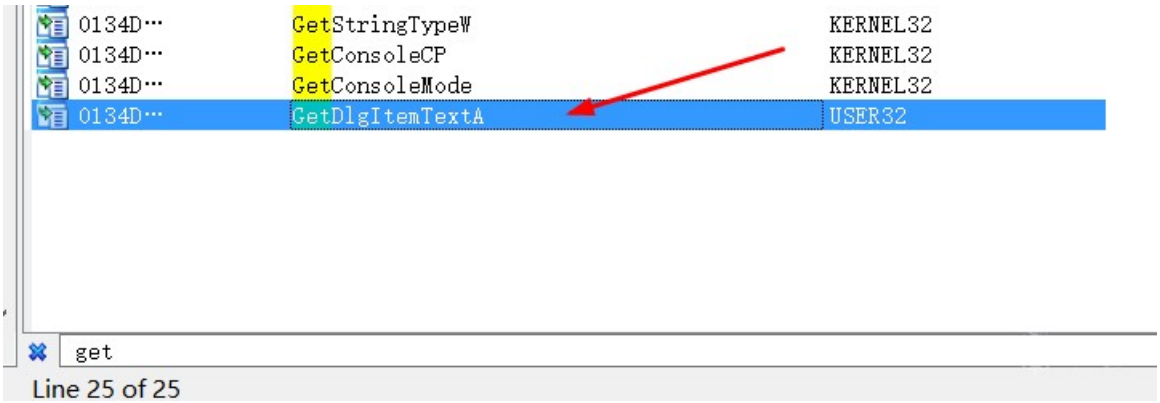
2.正文

刚开始那道题目的时候！

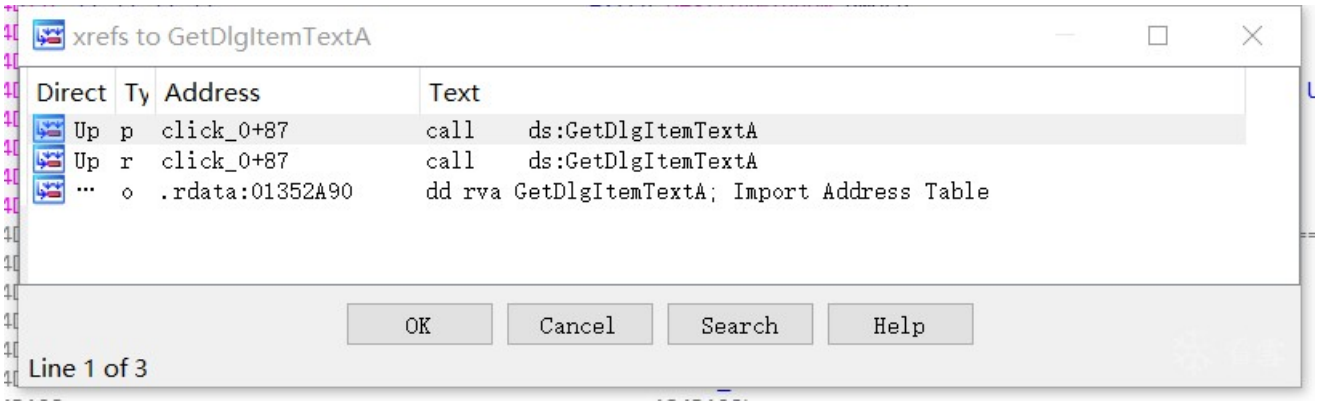
先运行下看下情况！

发现是个窗体程序

直接拖到IDA里面！



找到GetDlgItemTextA



然后由它的找到单击事件

```
int __cdecl click_0(HWND hDlg)
{
  void *v1; // ST08_4
  signed int i; // [esp+Ch] [ebp-24h]
  signed int v4; // [esp+10h] [ebp-20h]
  CHAR String[20]; // [esp+18h] [ebp-18h]

  strcpy(Text, "try again!");
  strcpy(Caption, "fail");
  String[0] = 0;
  *&String[1] = 0;
  *&String[5] = 0;
  *&String[9] = 0;
  *&String[13] = 0;
  *&String[17] = 0;
  String[19] = 0;
  GetDlgItemTextA(hDlg, 1001, String, 20);
  v4 = 0;
  for ( i = 0; i < 20; ++i )
    v4 += String[i];
  if ( v4 <= 0 || v4 >= 4132 )
    return MessageBoxA(0, Text, Caption, 0);
  v1 = malloc(v4);
  check1();
  j___free_base(v1);
  return MessageBoxA(0, Text, Caption, 0);
}
```

```
text:01341020                         check1          proc near               ; CODE XREF: click_0+E9↓p
text:01341020
text:01341020                         var_4           = dword ptr -4
text:01341020
text:01341020 55                                      push    ebp
text:01341021 8B EC                                   mov     ebp, esp
text:01341023 51                                      push    ecx
text:01341024 C7 45 FC CC CC CC CC                    mov     [ebp+var_4], 0CCCCCCCCh
text:0134102B C7 45 FC 80 D1 34 01                    mov     [ebp+var_4], offset sub_134D180 ; addr of size
text:01341032 8B E5                                   mov     esp, ebp
text:01341034 5D                                      pop     ebp
text:01341035 C3                                      retn
text:01341035                         check1          endp
text:01341035
```

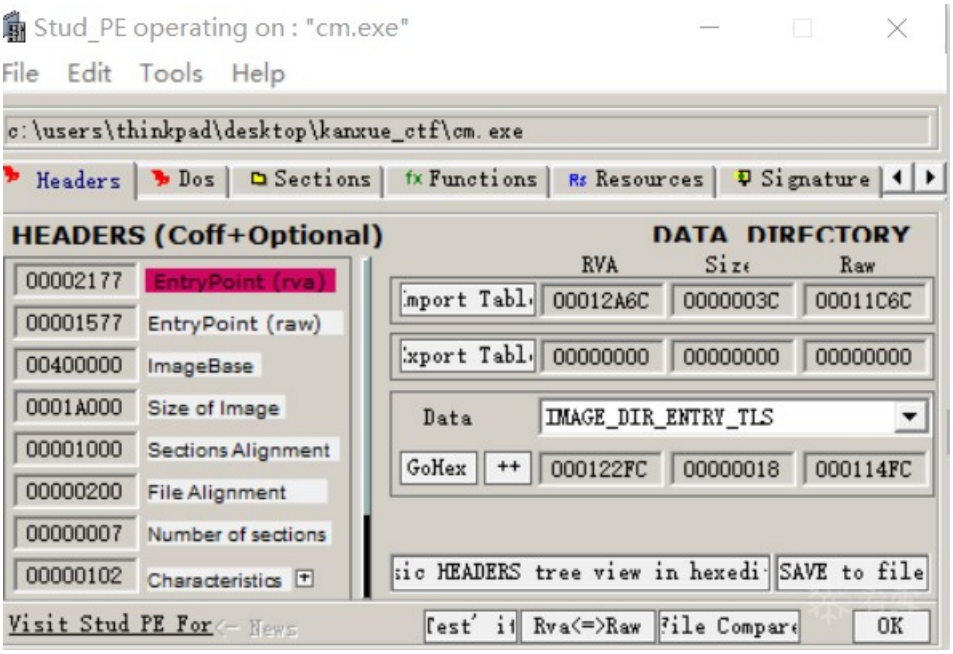
咋一看，懵了！怎么没有判断序列号成功的地方！

刚开始怀疑类似于第二题，初始化一个全局或者静态的类对象，将验证算法藏在析构函数里面！奈何太菜没有找到！

然后又怀疑是不是栈溢出，但是输入的时候给定了最大长度！而且string的缓冲区也符合最大。

然后一点点想法，一点点否定。

最后将程序拖入study_pe的时候，



发现这么一个东西。。。TLS表！

这东西注册回调函数的时候，一般先于入口点运行！

算了一下偏移值！

base+0x122FC

因为程序有重定向

所以在IDA中我把base值设成0x1340000(实际情况，由你来定)

0x1340000 +0x122FC=0x13522FC

```
rdata:013522F8 00 01 00 00                              dd 100h                 ; GuardFlags
rdata:013522FC 00 60 35 01              TlsDirectory    dd offset TlsStart
rdata:01352300 01 60 35 01              TlsEnd_ptr      dd offset TlsEnd
rdata:01352304 1C 48 35 01              TlsIndex_ptr    dd offset TlsIndex
rdata:01352308 58 D1 34 01              TlsCallbacks_ptr dd offset TlsCallbacks
rdata:0135230C 00 00 00 00              TlsSizeOfZeroFill dd 0
rdata:01352310 00 00 10 00              TlsCharacteristics dd 100000h
rdata:01352314 00 00 00 00 00 00 00 00+                 align 10h
```

由它找到回调函数

一路跟下来找到主要调用的地方，

```
BOOL make_addr()
{
  void *v0; // edx
  signed int j; // [esp+0h] [ebp-Ch]
  signed int i; // [esp+8h] [ebp-4h]

  sub_13418D0();
  lpAddress = v0;
  make_addr_exec(v0);
  for ( i = 0; i < 5; ++i )
    byte_13547DC[i] = *(lpAddress + i + 32);
  *&byte_1354028[1] = back_door_func - (lpAddress + 32) - 5;
  for ( j = 0; j < 5; ++j )
    *(lpAddress + j + 32) = byte_1354028[j];
  return make_addr_un(lpAddress);
}
```

实际上每次调用GetDlgItemTextA之后，紧接着就会 调用这个函数（ps:藏的可真深啊！）

实际情况这样

```
v8 = a1;
v3 = __readeflags();
v7 = v3;
dword_414800 = a3;
dword_4147EC = *(_DWORD *)(a2 + 20);
dword_4147F0 = malloc(dword_4147EC);
dword_4147F4 = a2 + 16;
memmove(dword_4147F0, *(const void **)(a2 + 16), dword_4147EC);
*(_DWORD *)(a2 - 16) = dword_4147F0;
*(_DWORD *)(a2 - 20) = *(_DWORD *)(a2 - 16) + 1;
do
  *(_BYTE *)(a2 - 21) = *(_BYTE *)(*(_DWORD *)(a2 - 16))++;
while ( *(_BYTE *)(a2 - 21) );
*(_DWORD *)(a2 - 28) = *(_DWORD *)(a2 - 16) - *(_DWORD *)(a2 - 20);
if ( (unsigned __int8)sub_401290(dword_4147F0, *(_DWORD *)(a2 - 28)) )
{
  v4 = sub_401870(dword_4147F0);
  if ( sub_4017B0(dword_4147F0, v4) == 1446302290 )
  {
    v5 = off_414030;
    *(_WORD *)off_414030 = 0;
    v5[2] = 0;
    for ( *(_WORD *)(a2 - 4) = 0; *(signed __int16 *)(a2 - 4) < 8; ++*(_WORD *)(a2 - 4) )
      *((char *)off_414030 - *(signed __int16 *)(a2 - 4)) = (39 - *(unsigned __int16 *)(a2 - 4)) ^ byte_41401C[7 - *(signed __int16 *)(a2 -
    *(_WORD *)off_414034 = 0;
    for ( *(_WORD *)(a2 - 8) = 0; *(signed __int16 *)(a2 - 8) < 3; ++*(_WORD *)(a2 - 8) )
      *((char *)off_414034 - *(signed __int16 *)(a2 - 8)) = (34 - *(unsigned __int16 *)(a2 - 8)) ^ byte_414024[2 - *(signed __int16 *)(a2 -
  }
}
j___free_base(dword_4147F0);
dword_4147F8 = (int (__thiscall *)(_DWORD))((char *)lpAddress + 32);
sub_4019B0(lpAddress);
for ( *(_DWORD *)(a2 - 12) = 0; *(_DWORD *)(a2 - 12) < 5; ++*(_DWORD *)(a2 - 12) )
  *((_BYTE *)lpAddress + *(_DWORD *)(a2 - 12) + 32) = byte_4147DC[*(_DWORD *)(a2 - 12)];
sub_4019E0(lpAddress);
```

看着很难看！

简单的定义了一个结构，让结果好看点

```
; ---------------------------------------------------------------

buf_            struc ; (sizeof=0x18, mappedto_69)
serial_len      db ?
                db ? ; undefined
                db ? ; undefined
i               db ?
tmp_calc        dd ?
serial_addr     dd ?                    ; seg
unkonw3         dd ?
unkonw2         db ?
                db ? ; undefined
                db ? ; undefined
field_13        db ?
len             db ?
field_15        db ?
field_16        db ?
field_17        db ?
buf_            ends
```

这里有一些地方没定义就是，是为了让ida识别的好看点，没有什么lowbyte的恶心人（ps:可能我是强迫症吧，嘻嘻）

```
unsigned int v7; // [esp-24h] [ebp-24h]
int v8; // [esp-18h] [ebp-18h]

v8 = len;
v3 = __readeflags();
v7 = v3;
zero = a3;
str_len = *&buf->len;
serial_addr_ = malloc(str_len);
serial_addr = &buf->unkonw2;
memmove(serial_addr_, *&buf->unkonw2, str_len);
buf[-1].serial_addr = serial_addr_;
buf[-1].tmp_calc = buf[-1].serial_addr + 1;
do
  buf[-1].i = *buf[-1].serial_addr++;
while ( buf[-1].i );
*&buf[-2].len = buf[-1].serial_addr - buf[-1].tmp_calc;
if ( check1(serial_addr_, *&buf[-2].len) )
{
  len_ = strlen_(serial_addr_);
  if ( check2(serial_addr_, len_) == 0x5634D252 )
  {
    v5 = off_1354030;
    *off_1354030 = 0;
    v5[2] = 0;
    for ( *&buf[-1].len = 0; *&buf[-1].len < 8; ++*&buf[-1].len )
      *(off_1354030 - *&buf[-1].len) = (39 - *&buf[-1].len) ^ byte_135401C[7 - *&buf[-1].len];
    *off_1354034 = 0;
    for ( *&buf[-1].unkonw2 = 0; *&buf[-1].unkonw2 < 3; ++*&buf[-1].unkonw2 )
      *(off_1354034 - *&buf[-1].unkonw2) = (34 - *&buf[-1].unkonw2) ^ byte_1354024[2 - *&buf[-1].u
  }
}
j___free_base(serial_addr_);
dword_13547F8 = (lpAddress + 32);
make_addr_exec(lpAddress);
for ( buf[-1].unkonw3 = 0; buf[-1].unkonw3 < 5; ++buf[-1].unkonw3 )
```

从上图中逻辑中可以看出主要有两个检测点！

check1

check2

实际情况中，我过了一个答案就出来了！

```
bool __cdecl check1(int s, int len)
{
  table1[0][0] = 4;
  table1[0][1] = 1;
  table1[0][2] = 3;
  table1[1][0] = 7;
  table1[1][1] = 2;
  table1[1][2] = 5;
  table1[2][0] = 8;
  table1[2][1] = 6;
  table1[2][2] = 0;
  return check_(s, len);
}
```

初始化一个3x3的表。

1 2 3	4 1 3 7 2 5 8 6 0

```
bool __cdecl check_(char *s, int len_)
{
  int i; // [esp+0h] [ebp-Ch]
  int v4; // [esp+8h] [ebp-4h]

  v4 = 0xCCCCCCCC;
  if ( len_ % 2 )                               // we
    return 0;
  for ( i = 0; i < len_; i += 2 )
  {
    if ( s[i] == 'w' )
      v4 = 0;
    if ( s[i] == 'd' )
      v4 = 1;
    if ( s[i] == 's' )
      v4 = 2;
    if ( s[i] == 'a' )
      v4 = 3;
    if ( !move(v4, s[i + 1] - 48) )
      return 0;
  }
  return table1[0][0] == 1
      && table1[0][1] == 2
      && table1[0][2] == 3
      && table1[1][0] == 4
      && table1[1][1] == 5
      && table1[1][2] == 6
      && table1[2][0] == 7
      && table1[2][1] == 8
      && !table1[2][2];
}
```

因为结果需返回1，所以可知最后的表结构

1 2 3	1 2 3 4 5 6 7 8 0

不用想也知道上图中move，改变了表。

而且对传入的序列号进行了判断输入w -> 0 , d -> 1 , s -> 2 a -> 3

经常 玩游戏的可能一下就反应过来了，这是方向键！

可能我太菜了，一下没有反应过来，哈哈！

进去move看一看。

```
char __cdecl move(int s1, int s2)
{
  char result; // al
  signed int i; // [esp+8h] [ebp-8h]
  signed int j; // [esp+Ch] [ebp-4h]

  if ( !s2 )
    return 0;
  j = 0;
LABEL_4:
  if ( j >= 3 )
    return 0;
  for ( i = 0; ; ++i )
  {
    if ( i >= 3 )
    {
      ++j;
      goto LABEL_4;
    }
    if ( table1[j][i] == s2 )
      break;
LABEL_6:
    ;
  }
  switch ( s1 )
  {
    case 0:
      if ( j )
      {
        if ( table1[j - 1][i] )
        {
          result = 0;
        }
        else
        {
          table1[j - 1][i] = table1[j][i];
```

看着很乱，但是用心分析很快也能分析出来

这是我粗略整理了一下

```
char __cdecl move(int s1, int s2)
{
    if(!s2)
        return 0;
    int j=0,i=0;
    for(j=0;j<3;j++){
        for(i=0;i<3;i++){
            if(table[j][i]==s2){
                switch ( s1 )
                {
                case 0:
                    if ( j )
                    {
                        if ( table1[j - 1][i] )
                        {
```

```
                result = 0;
            }
            else
            {
                table1[j - 1][i] = table1[j][i];
                table1[j][i] = 0;
                result = 1;
            }
        }
        else
        {
            result = 0;
        }
        return result;
    case 1:
        if ( i == 2 )
        {
            result = 0;
        }
        else if ( table1[j][i+1] )
        {
            result = 0;
        }
        else
        {
            table1[j][i+1] = table1[j][i];
            table1[j][i] = 0;
            result = 1;
        }
        return result;
        //break;
    case 2:
        if ( j == 2 )
        {
            result = 0;
        }
        else if ( table1[j + 1][i] )
        {
            result = 0;
        }
        else
        {
            table1[j + 1][i] = table1[j][i];
            table1[j][i] = 0;
            result = 1;
        }
        return result;
    case 3:
```

```
                    if ( i )
                    {
                        if ( table1[j][i-1] )
                        {
                            result = 0;
                        }
                        else
                        {
                            table1[j][i-1] = table1[j][i];
                            table1[j][i] = 0;
                            result = 1;
                        }
                    }
                    else
                    {
                        result = 0;
                    }
                    return result;
                default:
                    ;
                }
            }
        }

    }
}
```

由此可以看出！对传入的参数在0 1 2 3中进行了判断，有w a s d 我们就就可以初步的推断是对表中的元素进行了移动。

结果正是如此，由上面化简的代码可知，实际上移动后的位置会被置为0，而后面判断的元素，每个都是初始化的内容！

所以可知移动的地方只能为0。

最后的表结构为

1 2 3	1 2 3 4 5 6 7 8 0

因此我们就可以大胆的猜测这是一个拼图游戏。

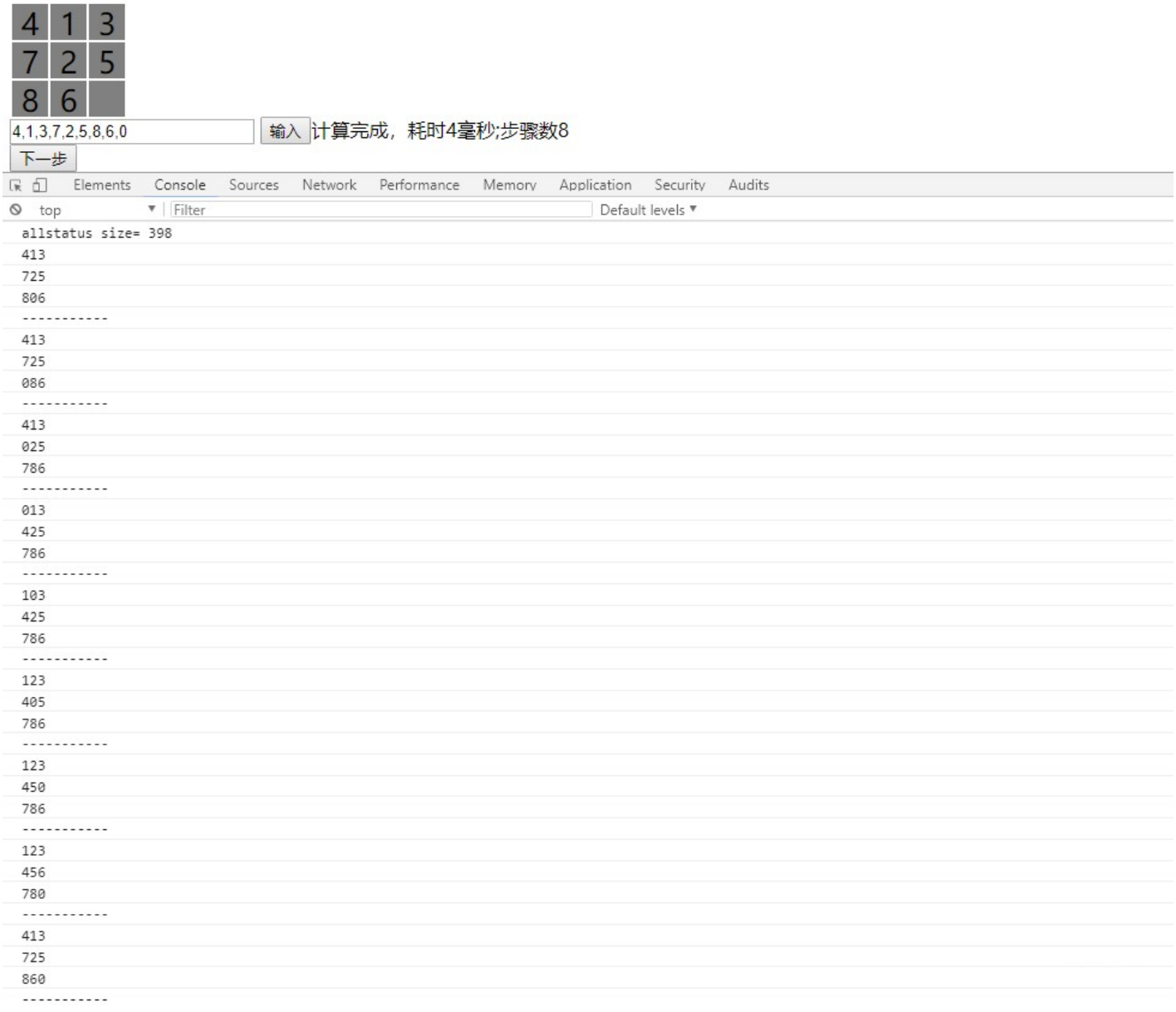
将

1 2 3	4 1 3 7 2 5 8 6 0	-- -> 1 2 3 4 5 6 7 8 0

下面就是写算法求解了

可能我太菜了，于是乎上网随便搜了一个。。。。。。。。。。

3.结果



1 2 3 4 5 6 7 8	6 d 8 d 7 s 4 s 1 a 2 w 5 a 6 w

最后的序列号就为：d6d8s7s4a1w2a5w6

4.总结

前面的大佬太强了，估计一下就看出算法了，学逆向一年左右了还是经验太少！还需学习啊，幸好有看雪这么好的论坛在（舔狗，哈哈 ^.^），看雪万岁！！

有空自己实现下拼图算法再发出来！先发网上的.....