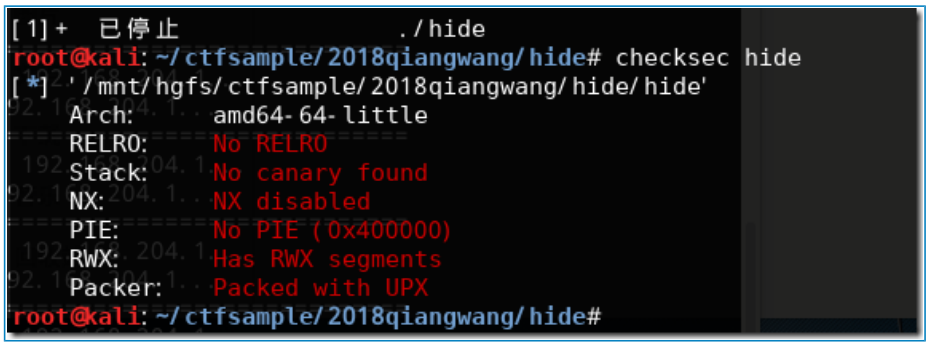
【强网杯2018】逆向hide

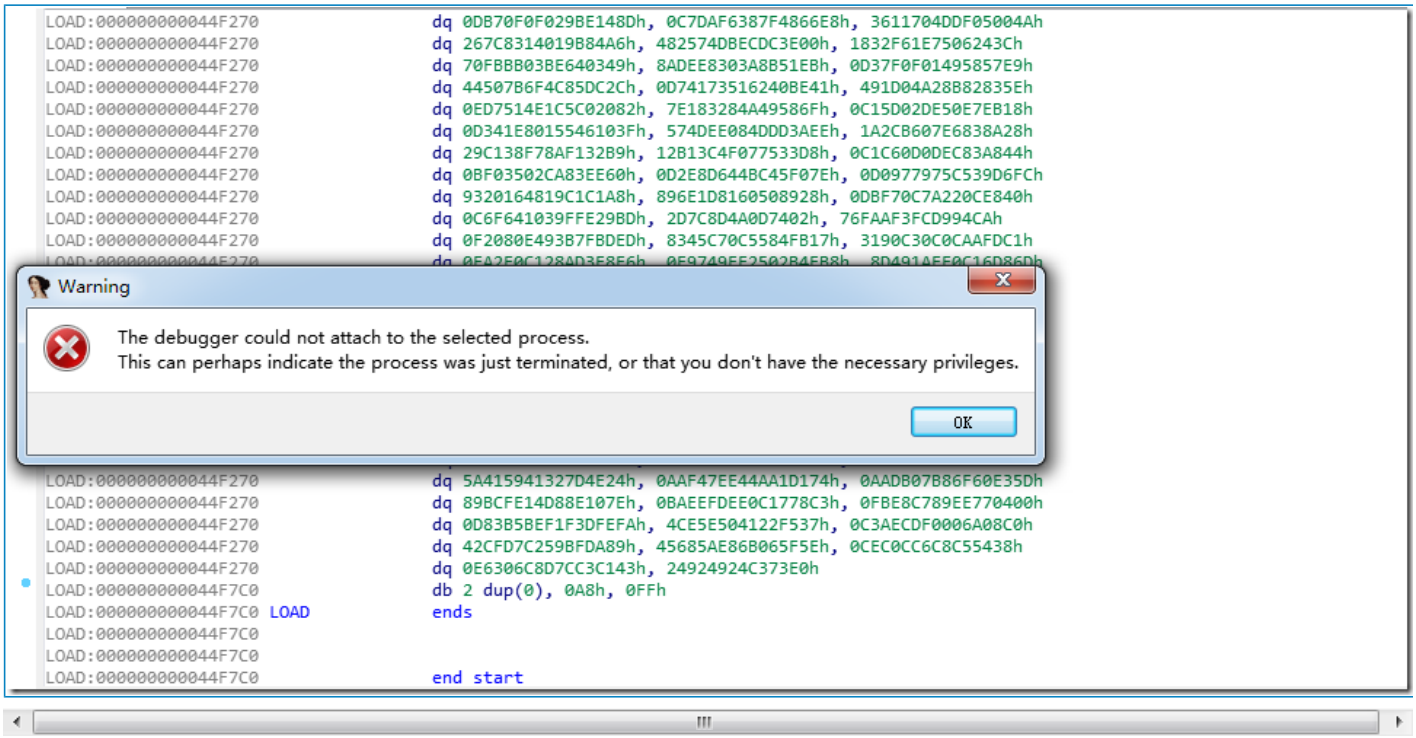
转载

weixin_30364147 于 2018-04-06 23:02:00 发布 57 收藏
文章标签： python
原文链接：http://www.cnblogs.com/anic/p/8729329.html

这是事后才做出来的，网上没有找到现成的writeup，所以在这里记录一下

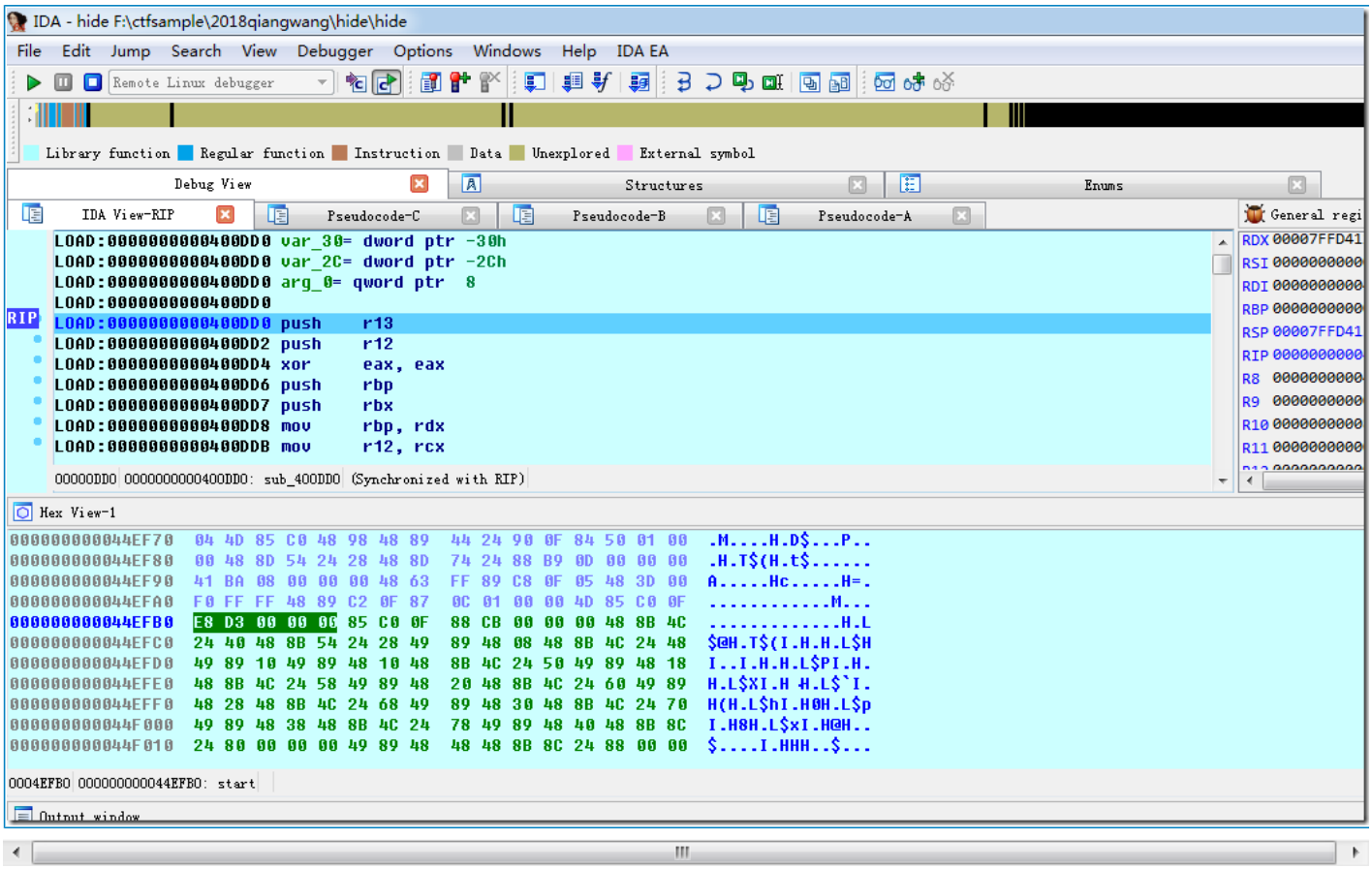
UPX加壳，而且linux下upx -d无法解，也无法gdb/ida attach

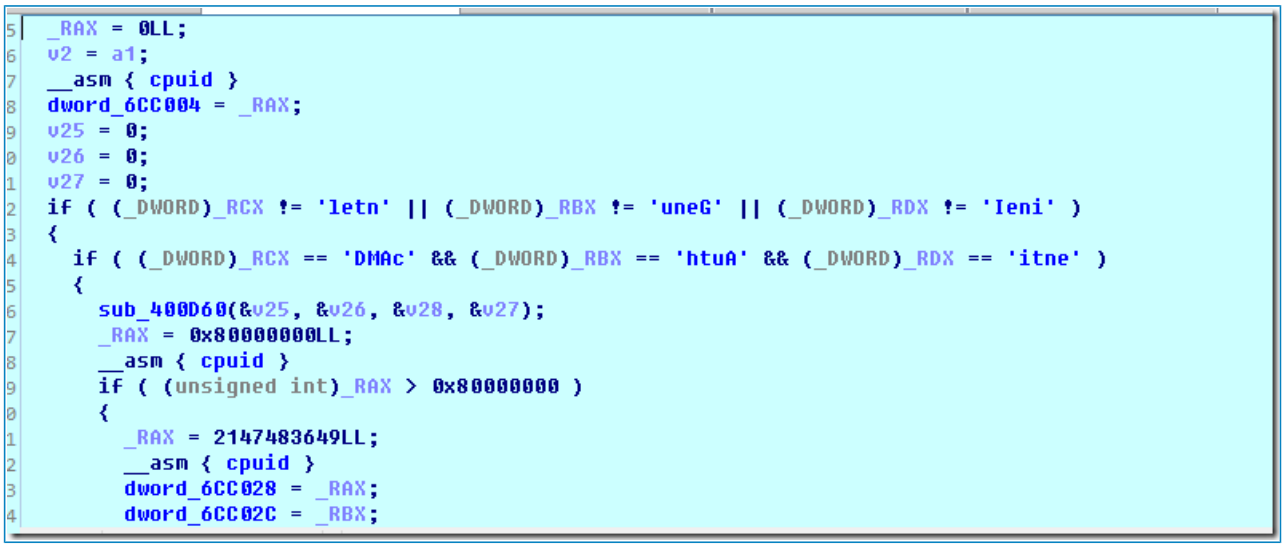




因为是64位，所以没有pushad，只能挨个函数进入，退出，看看程序是否恢复。

当运行到一0x400dd0，发现此时已经可以看见字符串了





```
5  _RAX = 0LL;
6  v2 = a1;
7  __asm { cpuid }
8  dword_6CC004 = _RAX;
9  v25 = 0;
0  v26 = 0;
1  v27 = 0;
2  if ( (_DWORD)_RCX != 'letn' || (_DWORD)_RBX != 'uneG' || (_DWORD)_RDX != 'Ieni' )
3  {
4    if ( (_DWORD)_RCX == 'DMAc' && (_DWORD)_RBX == 'htuA' && (_DWORD)_RDX == 'itne' )
5    {
6      sub_400D60(&v25, &v26, &v28, &v27);
7      _RAX = 0x80000000LL;
8      __asm { cpuid }
9      if ( (unsigned int)_RAX > 0x80000000 )
0      {
1        _RAX = 2147483649LL;
2        __asm { cpuid }
3        dword_6CC028 = _RAX;
4        dword_6CC02C = _RBX;
```

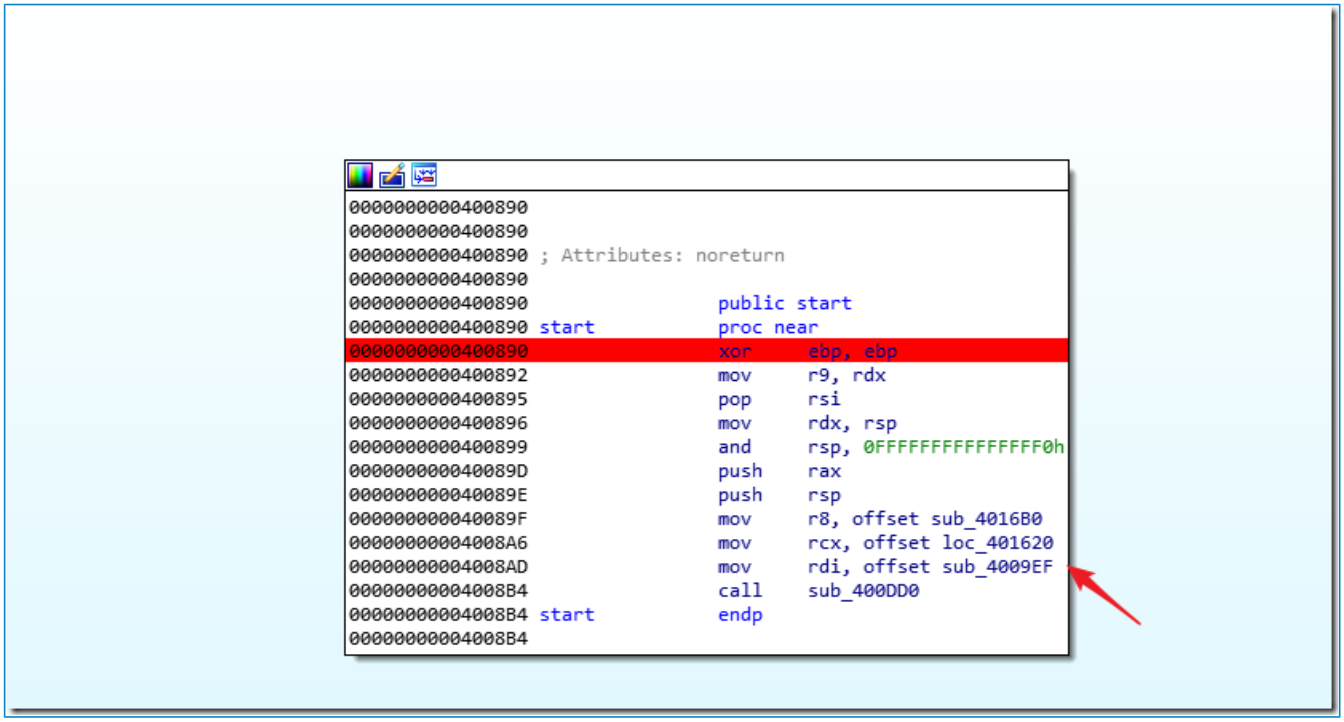
用dumphex的脚本来dump出内存，见hide_dump

```
static main(void)

{

auto fp, begin, end, dexbyte;

fp = fopen("C:\\dump.dex", "wb");

begin = 0x400000;

end = 0xADC000;

for ( dexbyte = begin; dexbyte < end; dexbyte ++ )

fputc(Byte(dexbyte), fp);
```

}

此时dump出的内容已经有程序运行的字符串了，通过字符串反查，这里

0x400890才是真正的启动地址

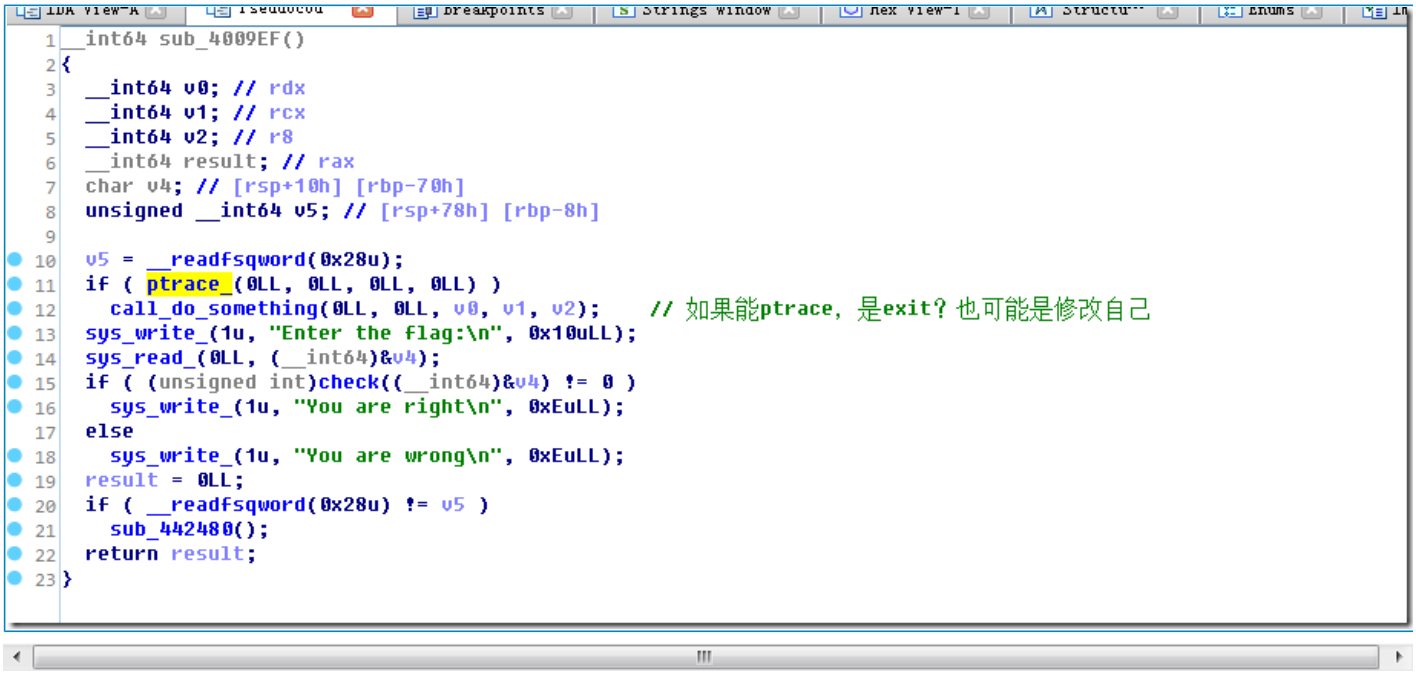


以后运行程序，在ida里面输入一下内容，即可直接运行到0x4009ef

from idaapi import *

from idc import *

run_to(0x4009ef)

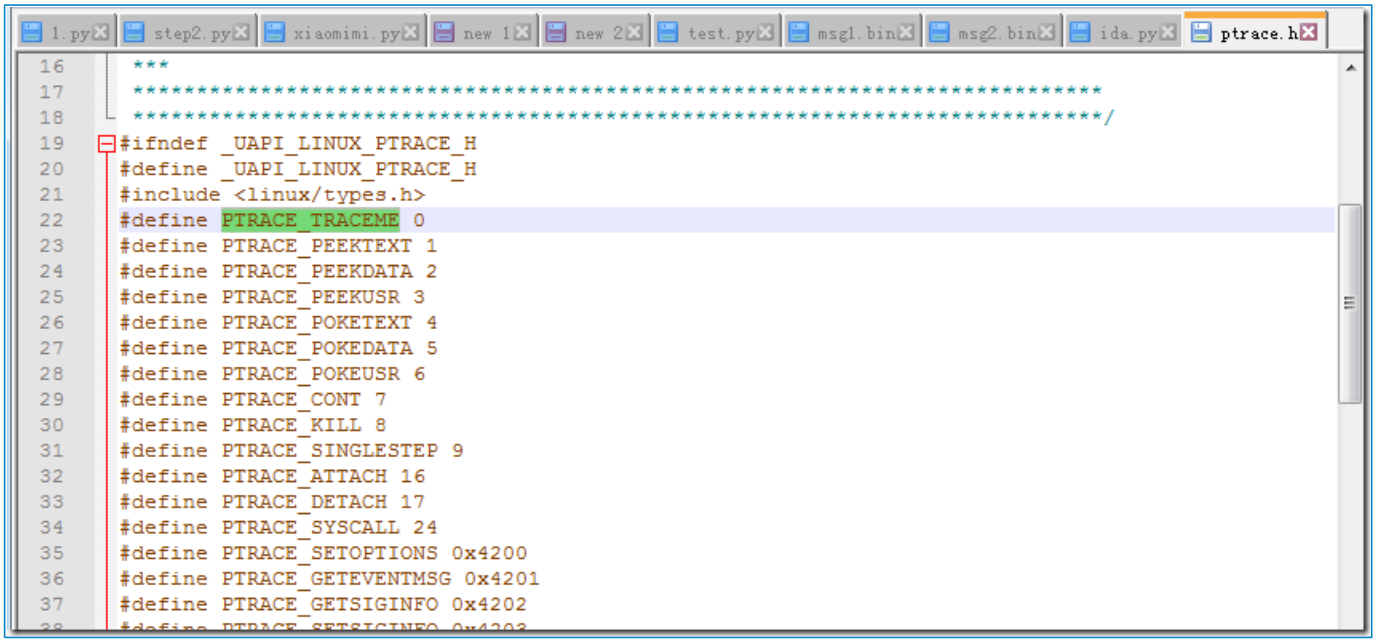


qwb{this_is_wrong_flag}

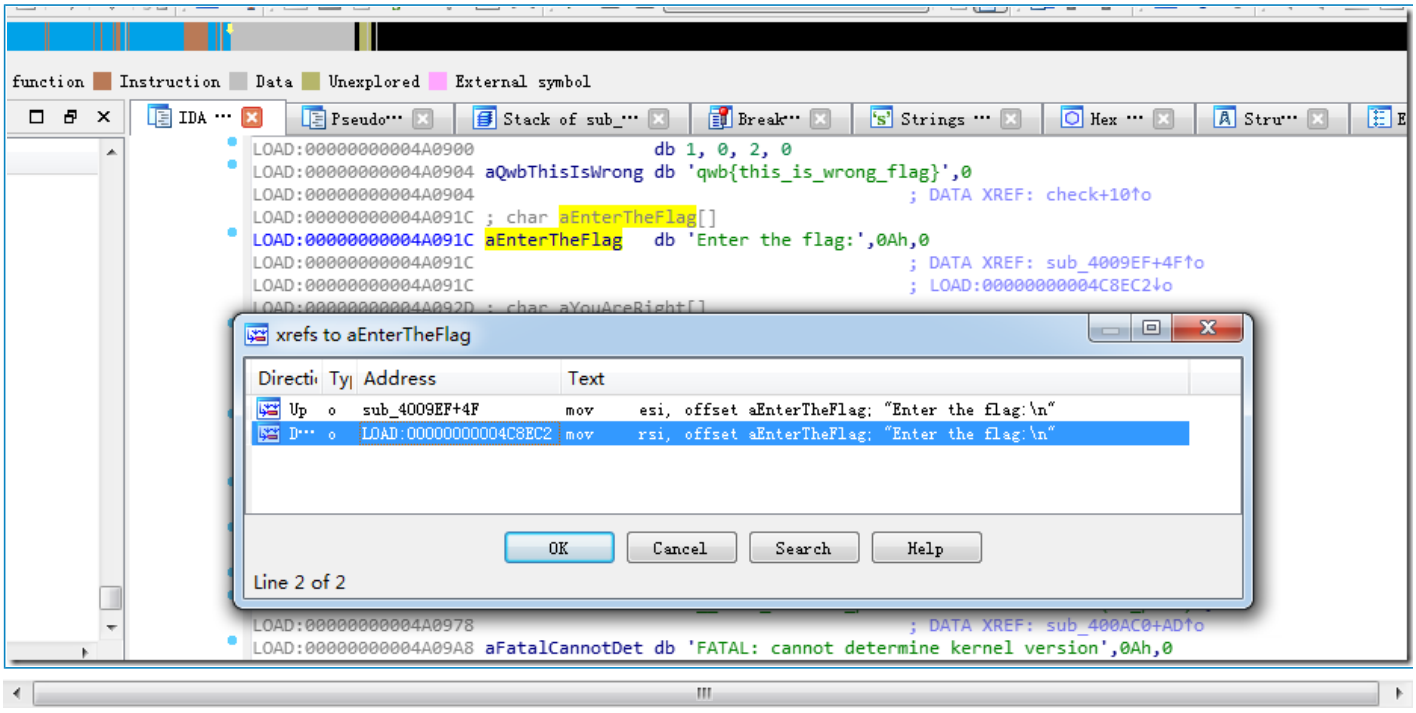
check到一个假flag，如果此时绕过ptrace且用ctrl+d作为结束，可以输出right。

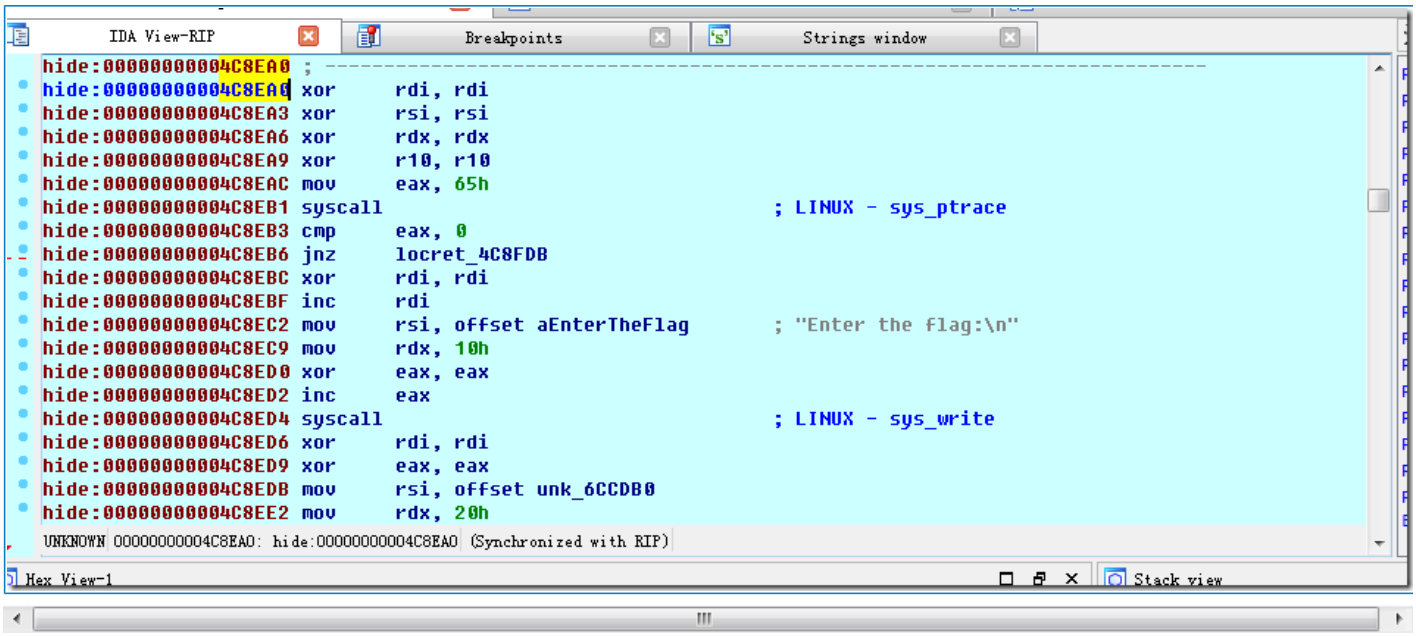
但是输入到正常程序是报wrong的，说明还有地方反调试以及修改了逻辑

根据ptrace.h，ptrace这里是PTRACE_TRACEME，自我调试

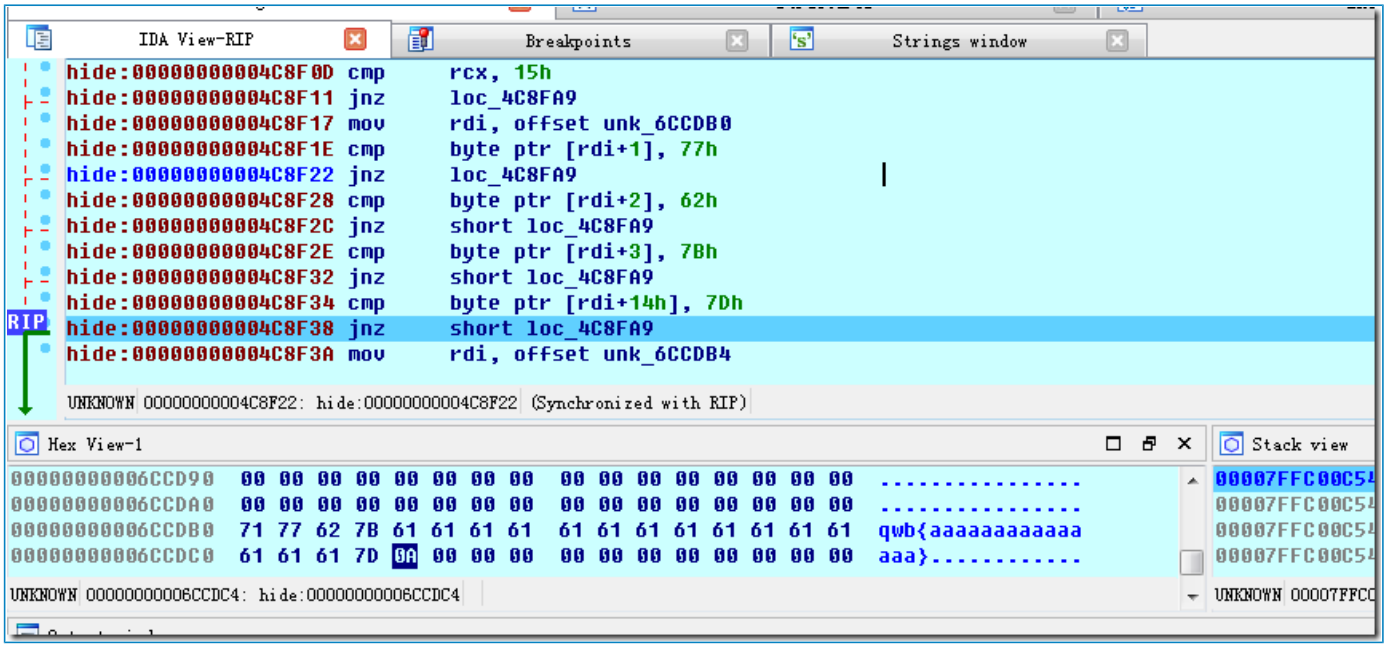


突然发现“Enter the flag:”字符串有2处引用

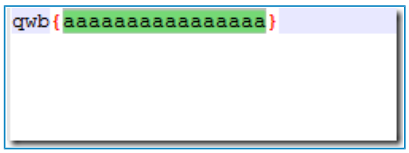




在4C8EA0也有类似的输出，怀疑正式运行时是这里。恢复函数失败，只能动态调试

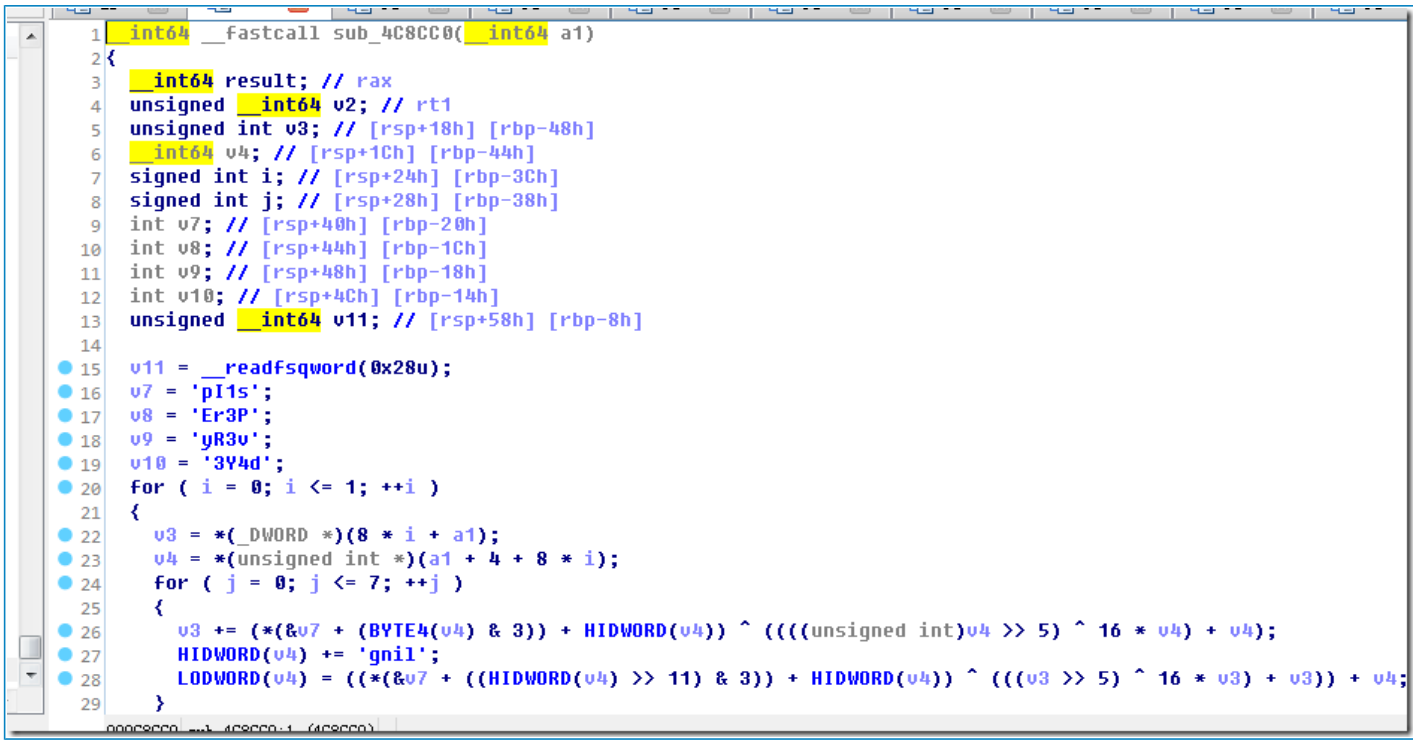


上面这里判断是不是qwb{}格式，构造payload

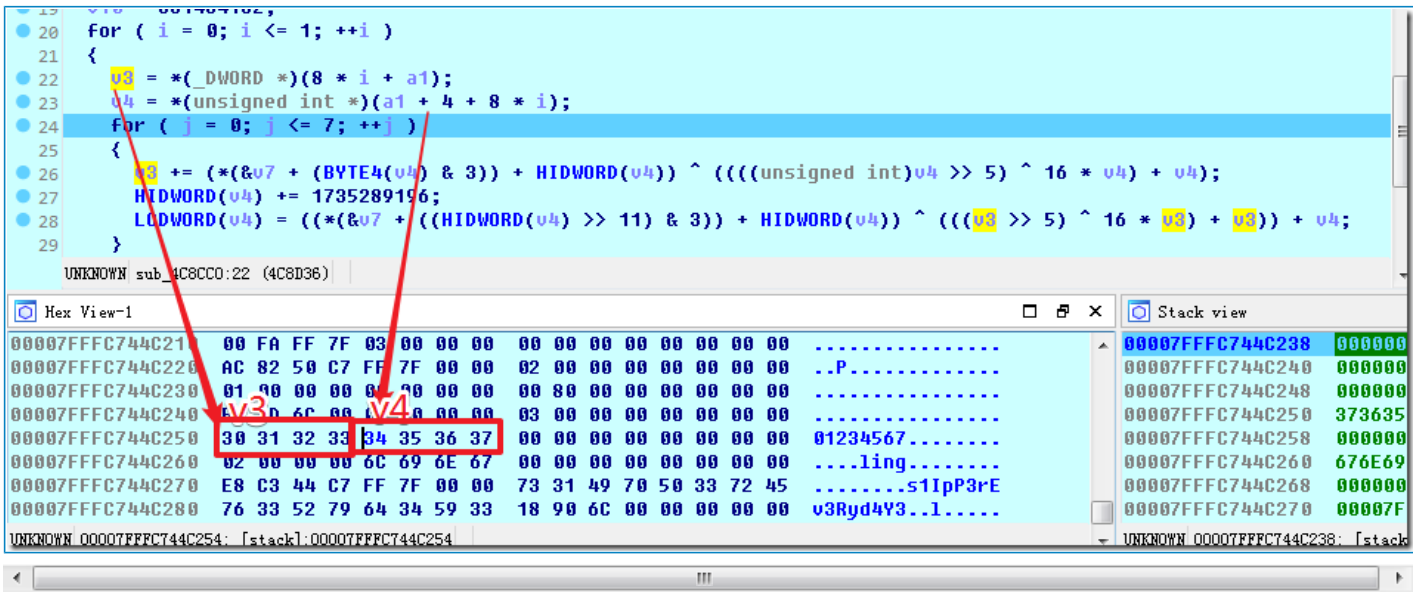


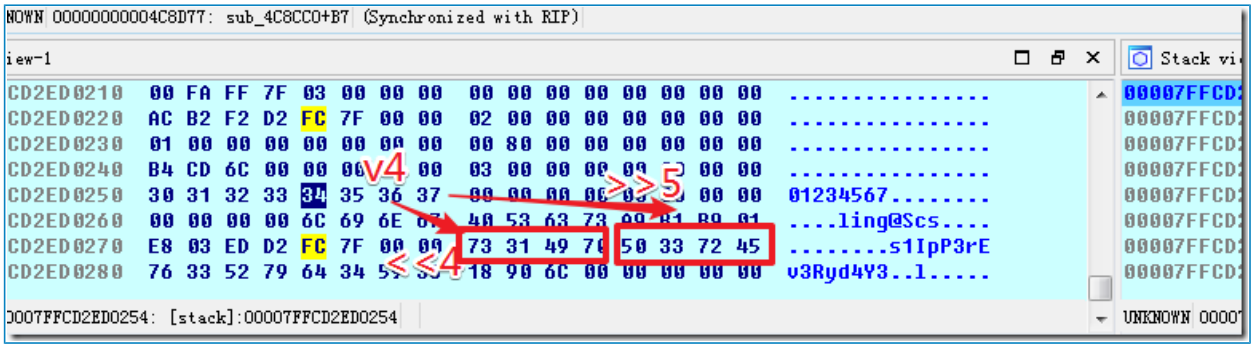
然后调用3次以下2个函数，输入内容为qwb{0123456789abcdef}中间部分的0123456789abcdef

sub_4C8CC0(__int64 a1)



```
__int64 __fastcall sub_4C8CC0(__int64 a1)
{
  __int64 result; // rax
  unsigned __int64 v2; // rt1
  unsigned int v3; // [rsp+18h] [rbp-48h]
  __int64 v4; // [rsp+1Ch] [rbp-44h]
  signed int i; // [rsp+24h] [rbp-3Ch]
  signed int j; // [rsp+28h] [rbp-38h]
  int v7; // [rsp+40h] [rbp-20h]
  int v8; // [rsp+44h] [rbp-1Ch]
  int v9; // [rsp+48h] [rbp-18h]
  int v10; // [rsp+4Ch] [rbp-14h]
  unsigned __int64 v11; // [rsp+58h] [rbp-8h]

  v11 = __readfsqword(0x28u);
  v7 = 'pI1s';
  v8 = 'Er3P';
  v9 = 'yR3v';
  v10 = '3Y4d';
  for ( i = 0; i <= 1; ++i )
  {
    v3 = *(_DWORD *)(8 * i + a1);
    v4 = *(unsigned int *)(a1 + 4 + 8 * i);
    for ( j = 0; j <= 7; ++j )
    {
      v3 += (*(&v7 + (BYTE4(v4) & 3)) + HIDWORD(v4)) ^ ((((unsigned int)v4 >> 5) ^ 16 * v4) + v4);
      HIDWORD(v4) += 'gnil';
      LODWORD(v4) = ((*(&v7 + ((HIDWORD(v4) >> 11) & 3)) + HIDWORD(v4)) ^ (((v3 >> 5) ^ 16 * v3) + v3)) + v4;
    }
```





这部分算法恢复见test2.py中的loop_j

```python
def loop_j(v3,v4):
    """
    v3 += (*(_DWORD *)&v7[4 * (BYTE4(v4) & 3)] + HIDWORD(v4)) ^ ((((unsigned int)v4 >> 5) ^ 16 * v4) + v4);
    HIDWORD(v4) += 'gnil';
    LODWORD(v4) = ((*(_DWORD *)&v7[4 * ((HIDWORD(v4) >> 11) & 3)] + HIDWORD(v4)) ^ (((v3 >> 5) ^ 16 * v3) + v3)) + v4;
    """
    HIDWORD_v4 = 0
    for j in range(8):
        i = HIDWORD_v4 & 3
        t = struct.unpack("I",v7ss[i*4:i*4+4])[0]

        v3 += (v4 + ((v4<<4 & 0xffffffff) ^ (v4 >> 5) )) ^ (t + HIDWORD_v4)
        v3 = v3 & 0xffffffff

        HIDWORD_v4 += 0x676E696C
        HIDWORD_v4 &= 0xffffffff

        i = HIDWORD_v4>>11 & 3
        t = struct.unpack("I",v7ss[i*4:i*4+4])[0]

        v4 += (v3 + ((v3 << 4 & 0xffffffff)^(v3 >> 5))) ^ (t + HIDWORD_v4)
        v4 = v4 & 0xffffffff

        """
        第一轮
        00007FFCB9B16030  9E 5A 8A 0D A2 FE EA 81
        """
    return v3,v4
```

sub_4C8E50——按位异或

```
1  _BYTE *__fastcall sub_4C8E50(__int64 a1)
2  {
3    _BYTE *result; // rax
4    signed int i; // [rsp+14h] [rbp-4h]
5
6    for ( i = 0; i <= 15; ++i )
7    {
8      result = (_BYTE *)(i + a1);
9      *result ^= i;
10   }
11   return result;
12 }
```

目标：rdi（qwb{}中间内容经过上面的多次变换后） == rsi（如下），

```
hide:00000000004C8F76 mov     rdi, offset unk_6CCDB4
hide:00000000004C8F7D call    near ptr unk_4C8E50
hide:00000000004C8F82 mov     rsi, offset unk_4C8CB0
hide:00000000004C8F89 mov     rdx, 0
hide:00000000004C8F90
hide:00000000004C8F90 loc_4C8F90:                             ; CODE XREF: hide:00000000004C8FA7↓j
hide:00000000004C8F90 cmp     rdx, 10h
hide:00000000004C8F94 jnb     short loc_4C8FB2
hide:00000000004C8F96 mov     al, [rdi]
hide:00000000004C8F98 mov     ah, [rsi]
hide:00000000004C8F9A cmp     al, ah
hide:00000000004C8F9C jnz     short loc_4C8FA9
hide:00000000004C8F9E inc     rdx
UNKNOWN 00000000004C8F90: hide:loc_4C8F90 (Synchronized with RIP)

Hex View-1                                                                    Stack view
00000000004C8C70  B8 03 C0 01 BA 05 00 87  06 05 00 00 FF FF 01 0A  ................  00007FFCDC
00000000004C8C80  74 05 D3 01 00 A0 02 05  00 00 FF FF 01 0A 6F 05  t.............o.  00007FFCDC
00000000004C8C90  E3 02 00 B0 03 05 00 00  FF FF 01 0B C5 01 03 8A  ................  00007FFCDC
00000000004C8CA0  03 A0 D7 03 05 A0 A0 A0  A0 A0 A0 A0 A0 A0 A0 A0  ................  00007FFCDC
00000000004C8CB0  52 B8 13 7F 35 8C F2 1B  F4 63 86 D2 73 4F 1E 31  R...5.....O.1     00007FFCDC
00000000004C8CC0  55 48 89 E5 48 83 EC 60  48 89 7D A8 64 48 8B 04  UH.....}.dH..     00007FFCDC
00000000004C8CD0  25 28 00 00 00 48 89 45  F8 31 C0 C7 45 E0 73 31  %(...H.E.1..E...  00007FFCDC
                                                                                  00007FFCDC
UNKNOWN 00000000004C8CB0: hide:unk_4C8CB0
```

经过test2.py的逆向，得到一个有意义的输入串f1Nd_TH3HIdeC0dE

```
----reverse----
52 b9 11 7c 31 89 f4 1c fc 6a 8c d9 7f 42 10 3e
e6 b4 a3 73 08 93 f7 48 68 30 b5 90 c2 8d 4f 53
e6 b5 a1 70 0c 96 f1 4f 60 39 bf 9b ce 80 41 5c
d8 ac 54 0d 7e 11 36 2d 17 d3 5a 0b b0 ad af ee
d8 ad 56 0e 7a 14 30 2a 1f da 50 00 bc a0 a1 e1
66 31 4e 64 5f 54 48 33 48 6c 64 65 43 30 64 45
f1Nd_TH3HldeC0dE
```

所以认为flag是qwb{f1Nd_TH3HldeC0dE}

实际运行，输入完qwb{f1Nd_TH3HldeC0dE}后，用ctrl+d可以看到成功（回车不行，因为用sys_read会连回车也认为是字符？）

```
root@kali:~/ctfsample/2018qiangwang/hide# ./hide
Enter the flag:
qwb{f1Nd_TH3HldeC0dE}You are right
```

但是实际为何会运行到hide脚本，就没有分析了，因为ptrace自己后发生什么事情，很难搞。

以下是通过IDA运行并跳过反调试的脚本

```
from idaapi import *

from idc import *

run_to(0x4009ef)

GetDebuggerEvent(WFNE_SUSP, -1)

SetRegValue(0x4C8EA0,"RIP")

GetDebuggerEvent(WFNE_SUSP, -1)

run_to(0x4C8EB3)

GetDebuggerEvent(WFNE_SUSP, -1)

SetRegValue(0,"RAX")

run_to(0x4C8CC0)
```

脚本含义

```
from idaapi import *
from idc import *
run_to(0x4009ef)          <-壳运行完成
GetDebuggerEvent(WFNE_SUSP, -1)
SetRegValue(0x4C8EA0,"RIP")   <-跳转至新函数
GetDebuggerEvent(WFNE_SUSP, -1)
run_to(0x4C8EB3)
GetDebuggerEvent(WFNE_SUSP, -1)
SetRegValue(0,"RAX")      <-绕过ptrace检查
run_to(0x4C8CC0)
```

转载于:https://www.cnblogs.com/anic/p/8729329.html