

【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>

版权

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

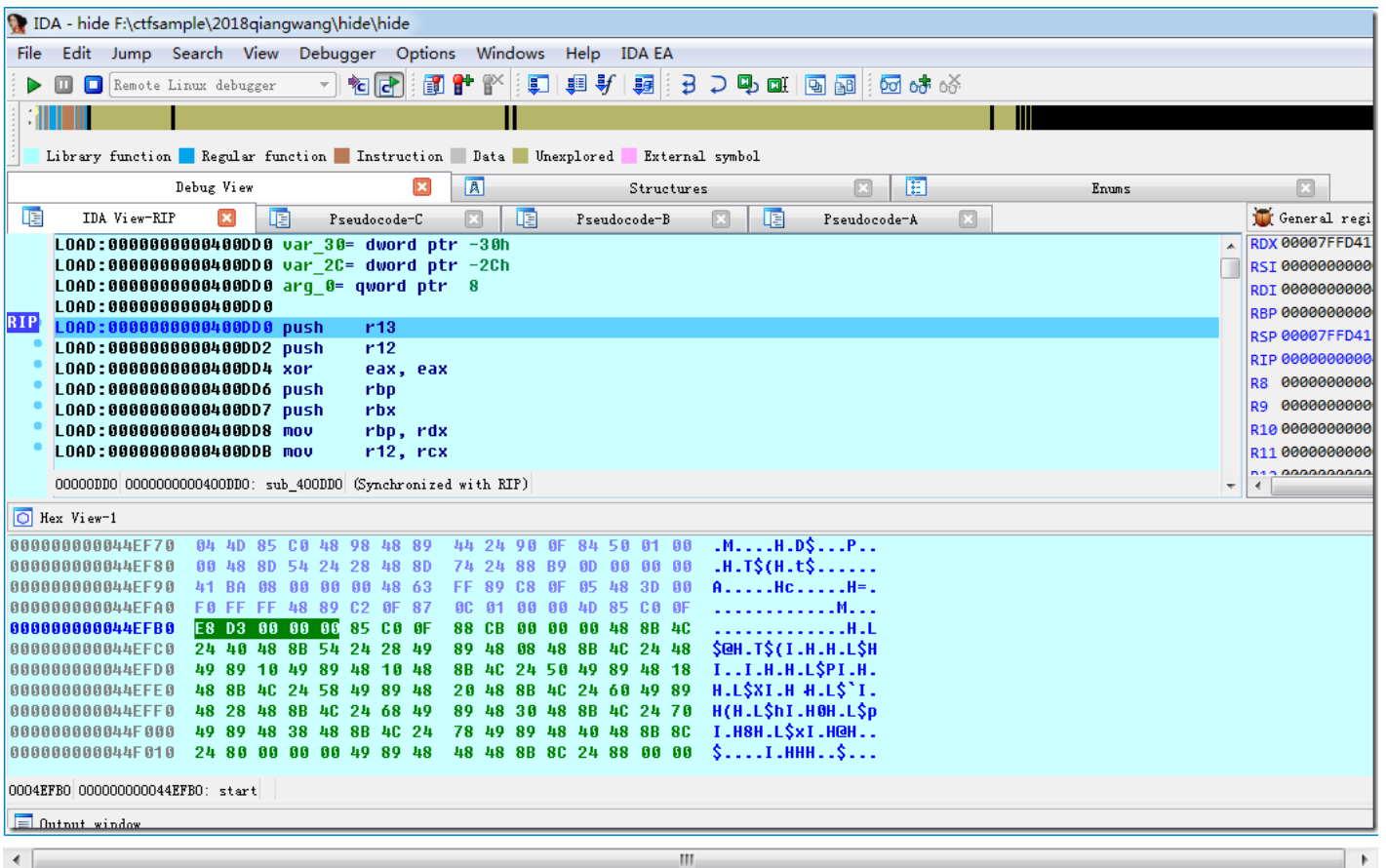
```
[1]+ 已停止 ./hide
root@kali: ~/ctfsample/2018qiangwang/hide# checksec hide
[*] '/mnt/hgfs/ctfsample/2018qiangwang/hide/hide'
Arch: amd64-64-little
RELRO: No RELRO
Stack: No canary found
NX: NX disabled
PIE: No PIE (0x400000)
RWX: Has RWX segments
Packer: Packed with UPX
root@kali: ~/ctfsample/2018qiangwang/hide#
```

The screenshot shows a debugger window with a memory dump on the left and a warning dialog box in the foreground. The memory dump consists of several lines of hex addresses and their corresponding disassembled instructions. The warning dialog box has a red 'X' icon and the text: "Warning: The debugger could not attach to the selected process. This can perhaps indicate the process was just terminated, or that you don't have the necessary privileges. OK".

```
LOAD:000000000044F270 dq 0DB70F0F029BE148Dh, 0C7DAF6387F4866E8h, 3611704DDF05004Ah
LOAD:000000000044F270 dq 267C8314019B84A6h, 482574DBECDC3E00h, 1832F61E7506243Ch
LOAD:000000000044F270 dq 70FB8B03BE640349h, 8ADEE8303A8851EBh, 0D37F0F01495857E9h
LOAD:000000000044F270 dq 44507B6F4C85DC2Ch, 0D74173516240BE41h, 491D04A28882835Eh
LOAD:000000000044F270 dq 0ED7514E1C5C02082h, 7E183284A49586Fh, 0C15D02DE50E7EB18h
LOAD:000000000044F270 dq 0D341E8015546103Fh, 574DEE0840DD3AEh, 1A2CB607E6838A28h
LOAD:000000000044F270 dq 29C138F78AF132B9h, 12B13C4F077533D08h, 0C1C600DEEC83A844h
LOAD:000000000044F270 dq 0BF03502CA83EE60h, 0D2E8D644BC45F07Eh, 0D0977975C539D6FCh
LOAD:000000000044F270 dq 9320164819C1C1A8h, 896E1D8160508928h, 0DBF70C7A220CE840h
LOAD:000000000044F270 dq 0C6F641039FFE29BDh, 2D7C8D4A0D7402h, 76FAAF3FCD994CAh
LOAD:000000000044F270 dq 0F2080E493B7FBDEDh, 8345C70C5584FB17h, 3190C30C0CAAFDC1h
LOAD:000000000044F270 dq 0EA2F0C128AD3E8E6h, 0E9749FE2502B4EB8h, 8D491AEF0C16D86Dh
LOAD:000000000044F270 dq 5A415941327D4E24h, 0AAF47EE44AA1D174h, 0AADB07B86F60E35Dh
LOAD:000000000044F270 dq 89BCFE14D88E107Eh, 0BAEEFDDE0C1778C3h, 0FBEC789EE770400h
LOAD:000000000044F270 dq 0D83B5BEF1F3DFEFAh, 4CE5E504122F537h, 0C3AECDF0006A08C0h
LOAD:000000000044F270 dq 42CFD7C259BFD89h, 45685AE86B065F5Eh, 0CEC0CC6C8C55438h
LOAD:000000000044F270 dq 0E6306C8D7CC3C143h, 24924924C373E0h
LOAD:000000000044F7C0 db 2 dup(0), 0A8h, 0FFh
LOAD:000000000044F7C0 LOAD ends
LOAD:000000000044F7C0 end start
```

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;
10 v26 = 0;
11 v27 = 0;
12 if ( (_DWORD)_RCX != 'letn' || (_DWORD)_RBX != 'uneG' || (_DWORD)_RDX != 'Ieni' )
13 {
14     if ( (_DWORD)_RCX == 'DMac' && (_DWORD)_RBX == 'htuA' && (_DWORD)_RDX == 'itne' )
15     {
16         sub_400D60(&v25, &v26, &v28, &v27);
17         _RAX = 0x80000000LL;
18         __asm { cpuid }
19         if ( (unsigned int)_RAX > 0x80000000 )
20         {
21             _RAX = 2147483649LL;
22             __asm { cpuid }
23             dword_6CC028 = _RAX;
24             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;
```

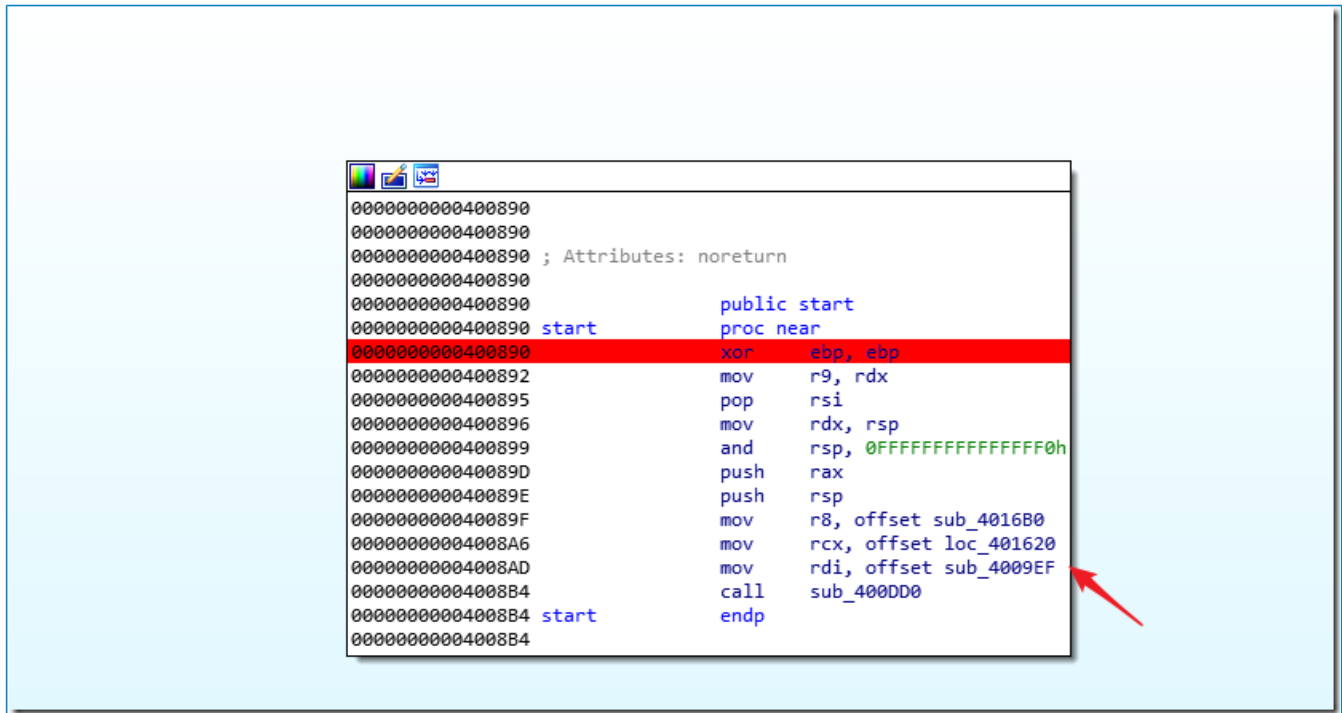
```
for ( dexbyte = begin; dexbyte < end; dexbyte ++ )
```

```
    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

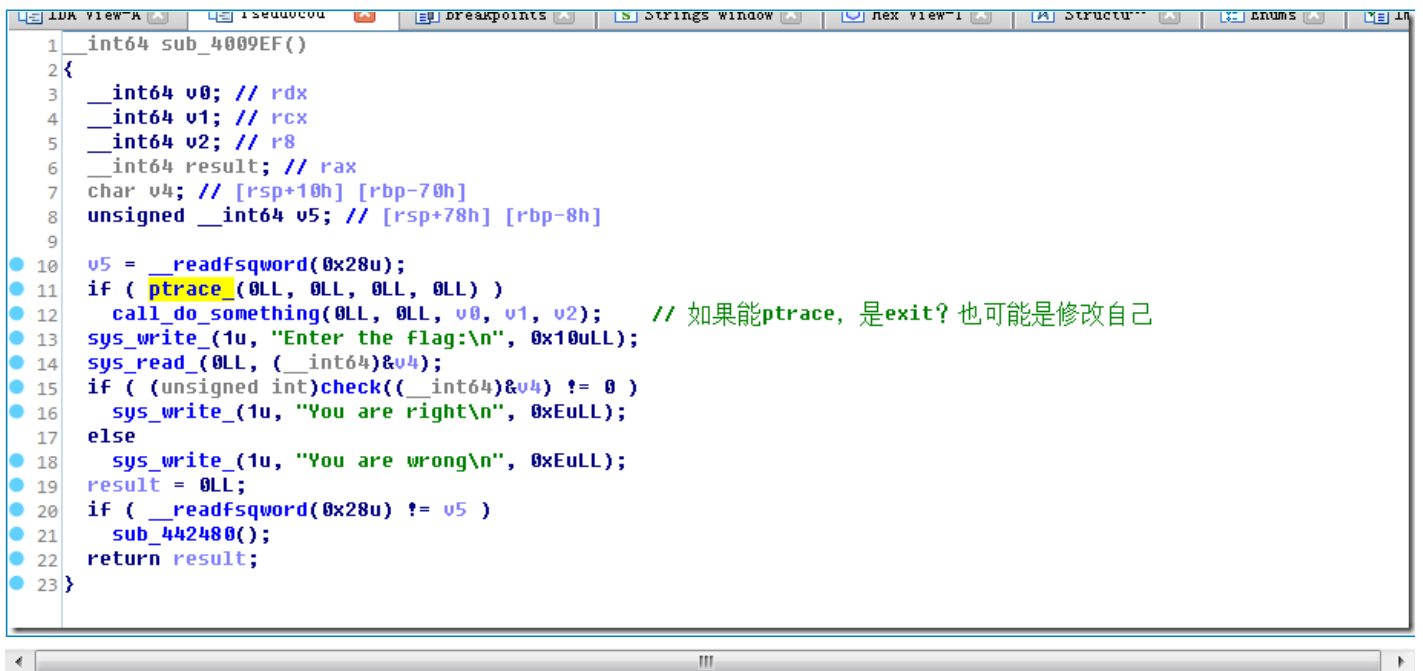


以后运行程序，在ida里面输入一下内容，即可直接运行到0x4009ef

```
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

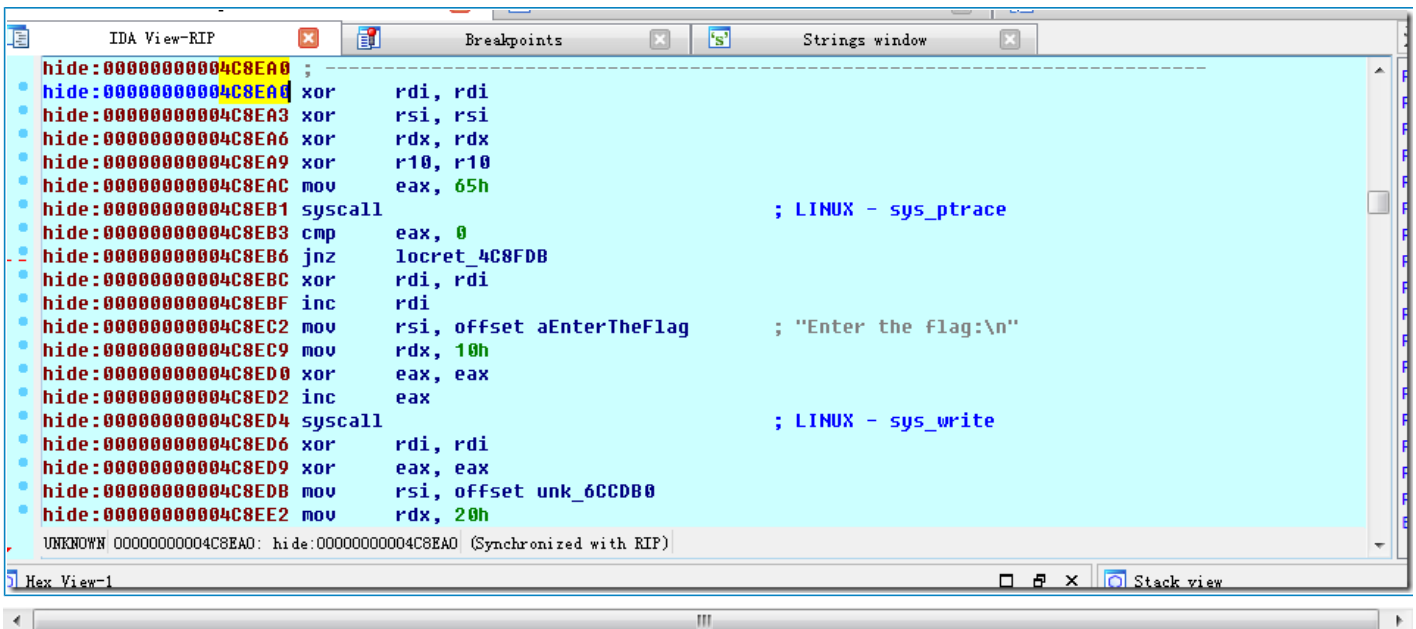
```
16  ***
17  *****/
18  *****/
19  #ifndef _UAPI_LINUX_PTRACE_H
20  #define _UAPI_LINUX_PTRACE_H
21  #include <linux/types.h>
22  #define PTRACE_TRACEME 0
23  #define PTRACE_PEEKTEXT 1
24  #define PTRACE_PEEKDATA 2
25  #define PTRACE_PEEKUSR 3
26  #define PTRACE_POKETEXT 4
27  #define PTRACE_POKEUSR 6
28  #define PTRACE_CONT 7
29  #define PTRACE_KILL 8
30  #define PTRACE_SINGLESTEP 9
31  #define PTRACE_ATTACH 16
32  #define PTRACE_DETACH 17
33  #define PTRACE_SYSCALL 24
34  #define PTRACE_SETOPTIONS 0x4200
35  #define PTRACE_GETEVENTMSG 0x4201
36  #define PTRACE_GETSIGINFO 0x4202
37  #define PTRACE_SETSIGINFO 0x4203
```

突然发现“Enter the flag:”字符串有2处引用

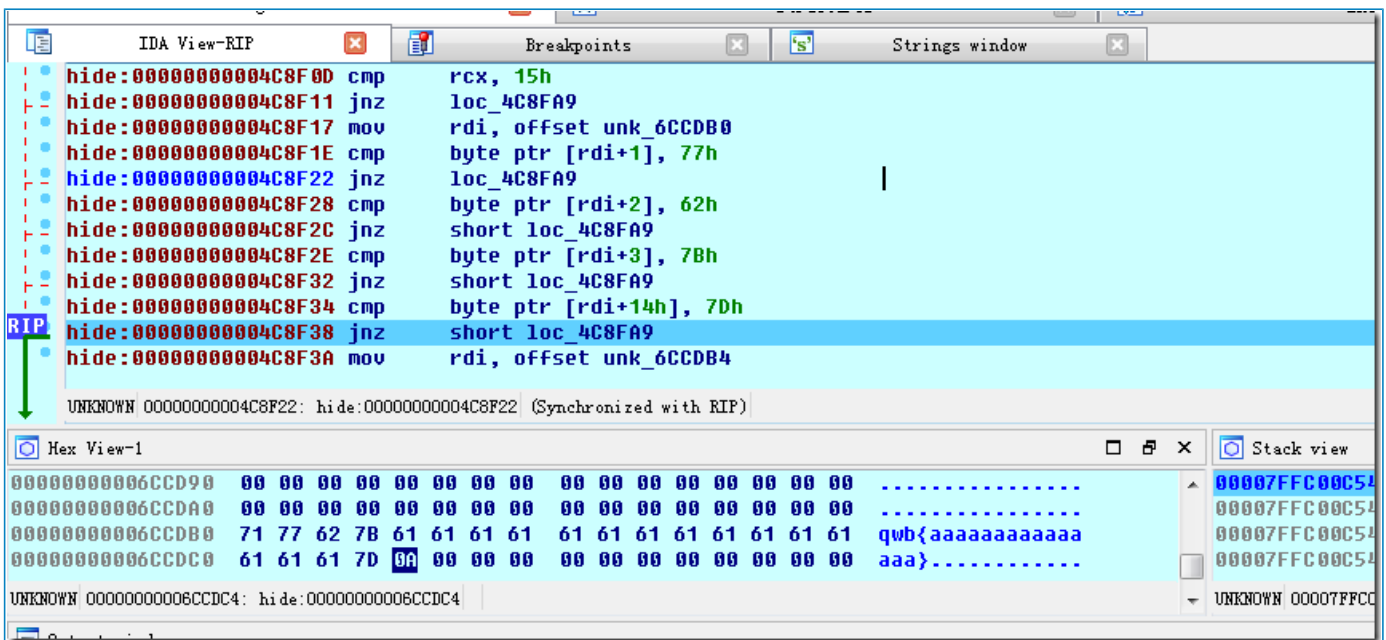
The screenshot shows the IDA Pro interface with assembly code loaded. A dialog box titled "xrefs to aEnterTheFlag" is open, displaying two entries:

Direction	Type	Address	Text
Up	o	sub_4009EF+4F	mov esi, offset aEnterTheFlag; "Enter the flag:\n"
D...	o	LOAD:0000000004C8EC2	mov rsi, offset aEnterTheFlag; "Enter the flag:\n"

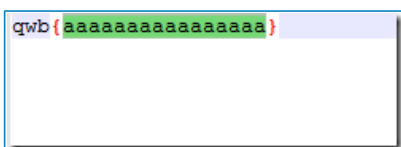
The dialog box also includes buttons for "OK", "Cancel", "Search", and "Help", and shows "Line 2 of 2" at the bottom.



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)

```

1  int64 __fastcall sub_4C8CC0(int64 a1)
2  {
3      int64 result; // rax
4      unsigned int v2; // rt1
5      unsigned int v3; // [rsp+18h] [rbp-48h]
6      int64 v4; // [rsp+1Ch] [rbp-44h]
7      signed int i; // [rsp+24h] [rbp-3Ch]
8      signed int j; // [rsp+28h] [rbp-38h]
9      int v7; // [rsp+40h] [rbp-20h]
10     int v8; // [rsp+44h] [rbp-1Ch]
11     int v9; // [rsp+48h] [rbp-18h]
12     int v10; // [rsp+4Ch] [rbp-14h]
13     unsigned int v11; // [rsp+58h] [rbp-8h]
14
15     v11 = __readfsqword(0x28u);
16     v7 = 'pI1s';
17     v8 = 'Er3P';
18     v9 = 'yR3u';
19     v10 = '3Y4d';
20     for ( i = 0; i <= 1; ++i )
21     {
22         v3 = *( _DWORD * )( 8 * i + a1 );
23         v4 = *( unsigned int * )( a1 + 4 + 8 * i );
24         for ( j = 0; j <= 7; ++j )
25         {
26             v3 += *( &v7 + ( BYTE4( v4 ) & 3 ) ) + HIDWORD( v4 ) ^ ( ( ( ( unsigned int ) v4 >> 5 ) ^ 16 * v4 ) + v4 );
27             HIDWORD( v4 ) += 'gnil';
28             LODWORD( v4 ) = ( *( &v7 + ( ( HIDWORD( v4 ) >> 11 ) & 3 ) ) + HIDWORD( v4 ) ^ ( ( ( v3 >> 5 ) ^ 16 * v3 ) + v3 ) ) + v4;
29         }
30     }
31 }

```

Code view showing the execution of the inner loop (lines 24-29). Red arrows point from the code to the corresponding values in the hex view below.

```

20  for ( i = 0; i <= 1; ++i )
21  {
22      v3 = *( _DWORD * )( 8 * i + a1 );
23      v4 = *( unsigned int * )( a1 + 4 + 8 * i );
24      for ( j = 0; j <= 7; ++j )
25      {
26          v3 += *( &v7 + ( BYTE4( v4 ) & 3 ) ) + HIDWORD( v4 ) ^ ( ( ( ( unsigned int ) v4 >> 5 ) ^ 16 * v4 ) + v4 );
27          HIDWORD( v4 ) += 1735289196;
28          LODWORD( v4 ) = ( *( &v7 + ( ( HIDWORD( v4 ) >> 11 ) & 3 ) ) + HIDWORD( v4 ) ^ ( ( ( v3 >> 5 ) ^ 16 * v3 ) + v3 ) ) + v4;
29      }
30  }

```

Hex View-1 (Address: 00007FFFC744C254):

00007FFFC744C210	00 FA FF 7F 03 00 00 00	00 00 00 00 00 00 00 00
00007FFFC744C220	AC 82 50 C7 FF 7F 00 00	02 00 00 00 00 00 00 00	..P.....
00007FFFC744C230	01 00 00 00 00 00 00 00	00 80 00 00 00 00 00 00
00007FFFC744C240	00 00 00 00 00 00 00 00	03 00 00 00 00 00 00 00
00007FFFC744C250	30 31 32 33 34 35 36 37	00 00 00 00 00 00 00 00	01234567.....
00007FFFC744C260	02 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00ling.....
00007FFFC744C270	E8 C3 44 C7 FF 7F 00 00	73 31 49 70 50 33 72 45s1IpP3rE
00007FFFC744C280	76 33 52 79 64 34 59 33	18 90 6C 00 00 00 00 00	v3Ryd4Y3...l.....

Stack view (Address: 00007FFFC744C238):

00007FFFC744C238	000000
00007FFFC744C240	000000
00007FFFC744C248	000000
00007FFFC744C250	373635
00007FFFC744C258	000000
00007FFFC744C260	676E69
00007FFFC744C268	000000
00007FFFC744C270	00007F

Hex View-1 (Address: CD2ED0254):

CD2ED0210	00 FA FF 7F 03 00 00 00	00 00 00 00 00 00 00 00
CD2ED0220	AC B2 F2 D2 FC 7F 00 00	02 00 00 00 00 00 00 00
CD2ED0230	01 00 00 00 00 00 00 00	00 80 00 00 00 00 00 00
CD2ED0240	B4 CD 6C 00 00 00 00 00	03 00 00 00 00 00 00 00
CD2ED0250	30 31 32 33 34 35 36 37	00 00 00 00 00 00 00 00	01234567.....
CD2ED0260	00 00 00 00 00 00 00 00	40 53 63 73 80 81 80 81ling@Scs....
CD2ED0270	E8 03 ED D2 FC 7F 00 00	73 31 49 70 50 33 72 45s1IpP3rE
CD2ED0280	76 33 52 79 64 34 59 33	18 90 6C 00 00 00 00 00	v3Ryd4Y3...l.....

Stack view (Address: 00007FFCD2ED0254):

00007FFCD2ED0254	000000
00007FFCD2ED0258	000000
00007FFCD2ED025C	000000
00007FFCD2ED0260	000000
00007FFCD2ED0264	000000
00007FFCD2ED0268	000000
00007FFCD2ED026C	000000
00007FFCD2ED0270	000000
00007FFCD2ED0274	000000
00007FFCD2ED0278	000000
00007FFCD2ED027C	000000
00007FFCD2ED0280	000000

这部分算法恢复见test2.py中的loop_j

```

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

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.....0.	00007FFCDC
00000000004C8C90	E3 02 00 B0 03 05 00 00 FF FF 01 0B C5 01 03 8A	00007FFCDC
00000000004C8CA0	03 00 D7 03 05 00 00 00 00 00 00 00 00 00 00	00007FFCDC
00000000004C8CB0	52 B8 13 7F 35 8C F2 1B F4 63 86 D2 73 4F 1E 31	R...5.....0.1	00007FFCDC
00000000004C8CC0	55 48 89 E5 48 83 EC 00 48 89 7D H8 04 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

UNKNOWN 00000000004C8CB0: hide:unk_4C8CB0

经过test2.py的逆向, 得到一个有意义的输入串f1Nd_TH3HldeC0dE

```
-----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
GetDebuggerEvent(WFNE_SUSP, -1)  函数
run_to(0x4C8EB3)
GetDebuggerEvent(WFNE_SUSP, -1)
SetRegValue(0,"RAX")  <-绕过ptrace检查
run_to(0x4C8CC0)
```