170930 逆向-Reversing.kr（Twist）

原创

奈沙夜影 于 2017-10-01 03:47:56 发布 533 收藏

分类专栏： CrackMe

本文链接：https://blog.csdn.net/whklhhhh/article/details/78146134

1625-5 王子昂 总结《2017年9月30日》【连续第365天总结】

A. Reversing.kr-Twist

B.

Twist

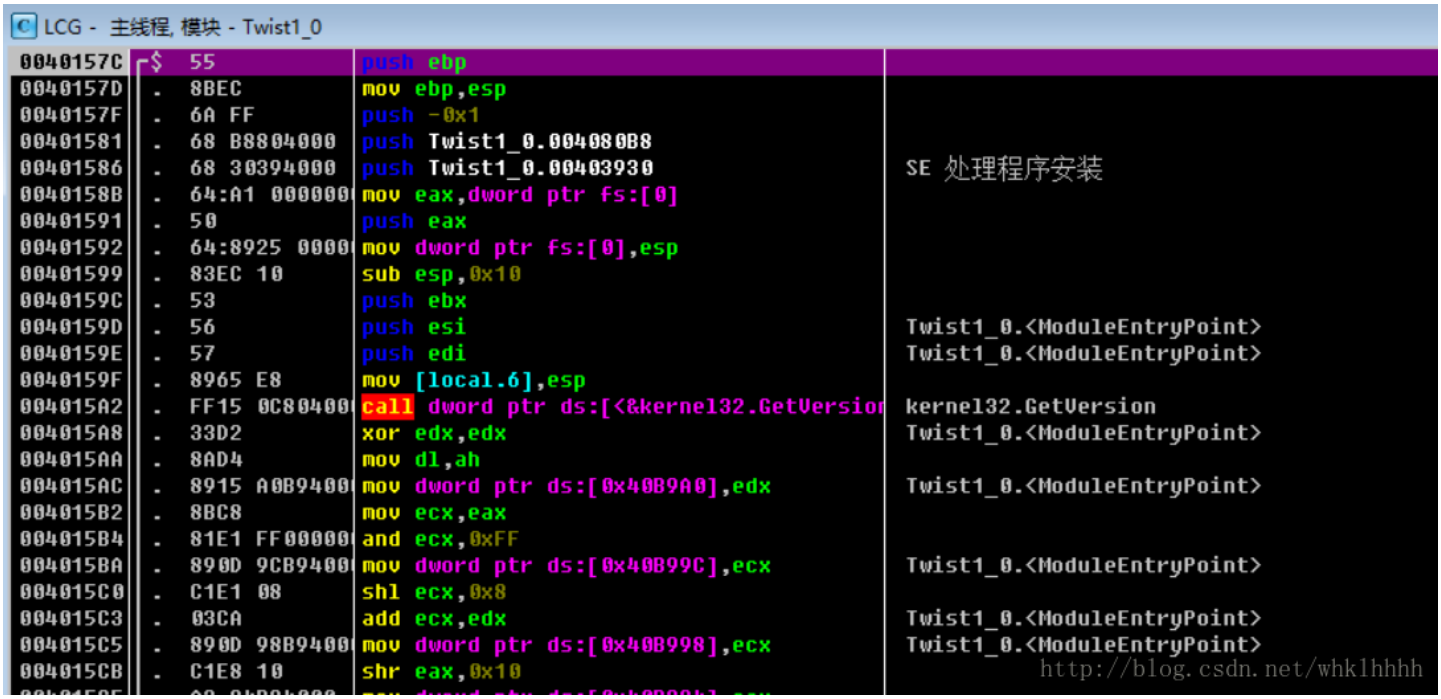
今天放假回家，在高速上耗了半天*(:з」∠)*再加上今天的题目恶心到爆炸，于是拖到现在

readme很简单，提示我运行在x86系统上
当时的我没多想，抄着WIN10 x64就运行下去了……嗨呀，前面好几个题目都说明题目一定要反复读透了才能做的

先查壳，EXEINFO说无壳，PEID提示是PASCAL写的无壳
事实证明高兴的太早了

拖入IDA，空荡荡的函数列表告诉我这里面还有猫腻……
F5提示函数内容被写过，那就只能乖乖用OD跟了

往下运行几步就看出来了，代码一边走一边被解密，还是有层保护壳
单步跟到最后一个大跳后，出现了熟悉的OEP：



这里我直接用OlleyDump脱壳出来以后，ImportRECR修复也无法运行，放到XP系统下可以运行，拖回物理机用IDA分析时F5提示某个call无法分析

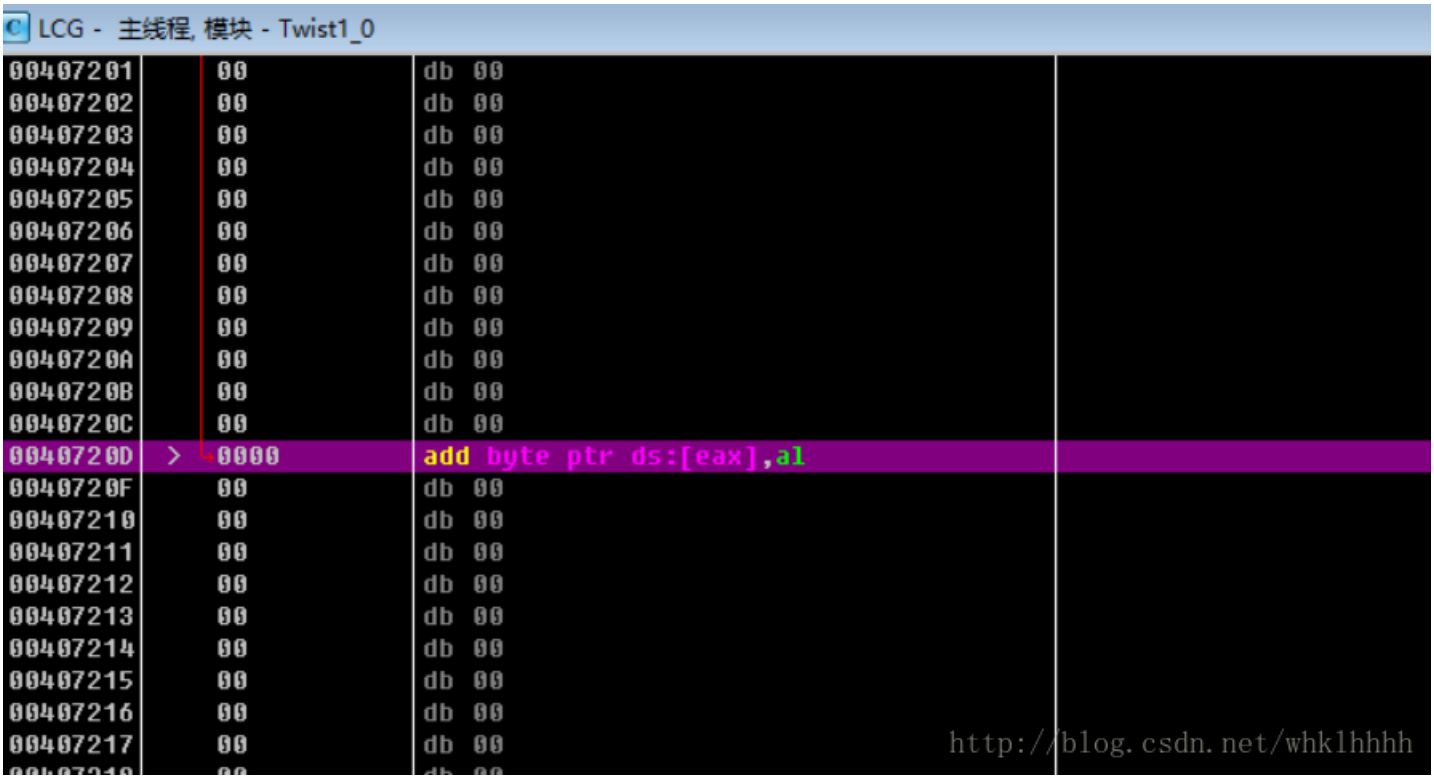
当时没多想，继续OD跟
在_main函数中先SetUnhandledExceptionFilter
这个函数绑定了异常捕获函数，是一种反调试机制：
当有调试器附加时，错误会先报给调试器，导致出错
无调试器时则会调用异常捕获函数

可以看到，这一行调用了ds:[edx]，而edx=0，出现了错误

直接NOP掉，或者修改EIP到下一行都可以跳过
再往下是很简单的输入、输出和判断结构

```
004012AA  |.  E8 21FFFFFF   call Twist1_0.004011D0                  ;  print说明
004012AF  |.  E8 4CFFFFFF   call Twist1_0.00401200                  ;  print "input:"
004012B4  |.  E8 67FFFFFF   call Twist1_0.00401220                  ;  scanf
……
NOP
……
004012C6  |.  E8 75FFFFFF   call Twist1_0.00401240                  ;  check
004012CB  |.  85C0          test eax,eax
004012CD  |.  74 10         je short Twist1_0.004012DF
; 关键跳
004012CF  |.  68 78904000   push Twist1_0.00409078                  ;  ASCII "Correct!\n"
004012D4  |.  E8 5B020000   call Twist1_0.00401534
004012D9  |.  83C4 04       add esp,0x4
004012DC  |.  33C0          xor eax,eax
004012DE  |.  C3            retn
004012DF  |>  68 70904000   push Twist1_0.00409070                  ;  ASCII "Wrong\n"
004012E4  |.  E8 4B020000   call Twist1_0.00401534
004012E9  |.  83C4 04       add esp,0x4
004012EC  |.  33C0          xor eax,eax
004012EE  \.  C3            retn
```

很明显，输入测试值后我们就应该跟入sub_401240了
在这个函数中将input送入堆栈中，然后jmp 40720D……



这啥都没有，执行个啥？难道说是直接调用异常处理函数？
本来是这么以为的，正巧异常处理函数是对某个地址的值+2，我想着如果能破坏掉Return Address就正好可以跳过关键跳，进入Correct分支中了
然而这意味着需要输入/x19/xff等不可见字符，flag很明显不应该是它们……

百思不得其解，无奈查找WriteUp
原来这是OlleyDump的锅，源程序中40720D这里好好的有一段代码，脱出来就被置0了……求解惑如何解决（于是我现在每次重来都得手动加载一次壳，又不能在OEP下断（因为软件断点的原理是将其指令暂时用INT 3来替换，当壳对此处代码解密的时候就会出岔子）。等等……突然想起来应该可以用硬件断点的……傻了傻了

拿回源程序，往下走：
这回sub_40720D有代码了，它提取ZwQueryInformationProcess函数的地址，然后拷贝16个字节到409150处
下面一段也是个坑

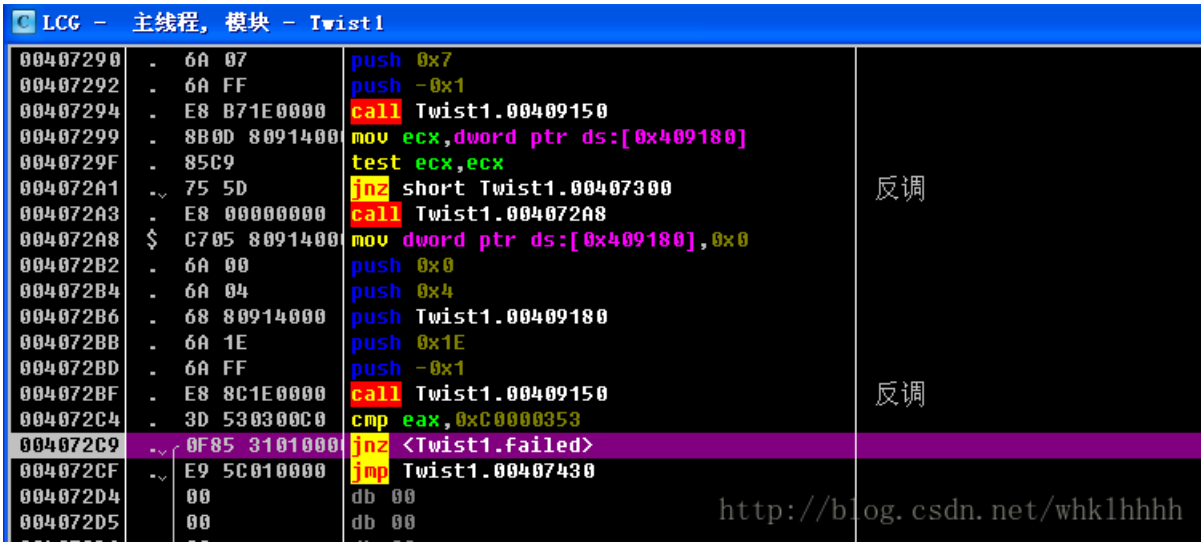
```
00407263   .  E8 05000000   call Twist1.0040726D                    ;  验证函数完整性
00407268   >  83F8 00       cmp eax,0x0
0040726B   .  74 10         je short Twist1.0040727D
0040726D   $  8039 B8       cmp byte ptr ds:[ecx],0xB8
00407270   .^ 75 F6         jnz short Twist1.00407268
00407272   .  83C1 05       add ecx,0x5
00407275   .  8039 BA       cmp byte ptr ds:[ecx],0xBA
00407278   .^ 75 EE         jnz short Twist1.00407268
0040727A   .  33C0          xor eax,eax
0040727C   .  C3            retn
```

很容易看出来，它是对拷贝区域进行校验
要求在特定区域出现0xB8和0xBA字节，否则死循环

我刚运行到这里的时候以为是反调，找了一下也没看到什么有关的函数
虽然说ZwQueryInformationProcess是反调相关的函数，但它还没被执行呢啊
后来发现这个函数来自于ntdll.dll，而它与64位有关，估摸着拷贝来的函数既然不对，应该是这个dll有问题，放回xp系统运行试试

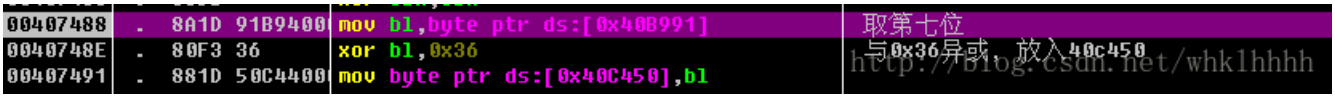
果然成功了（其实提示中早就说了，我压根没放在心上OTZ

再往下就是调用ZwQueryInformationProcess进行反调试了
不过估计是OD的插件在起作用，这里我没动什么也顺利通过反调了

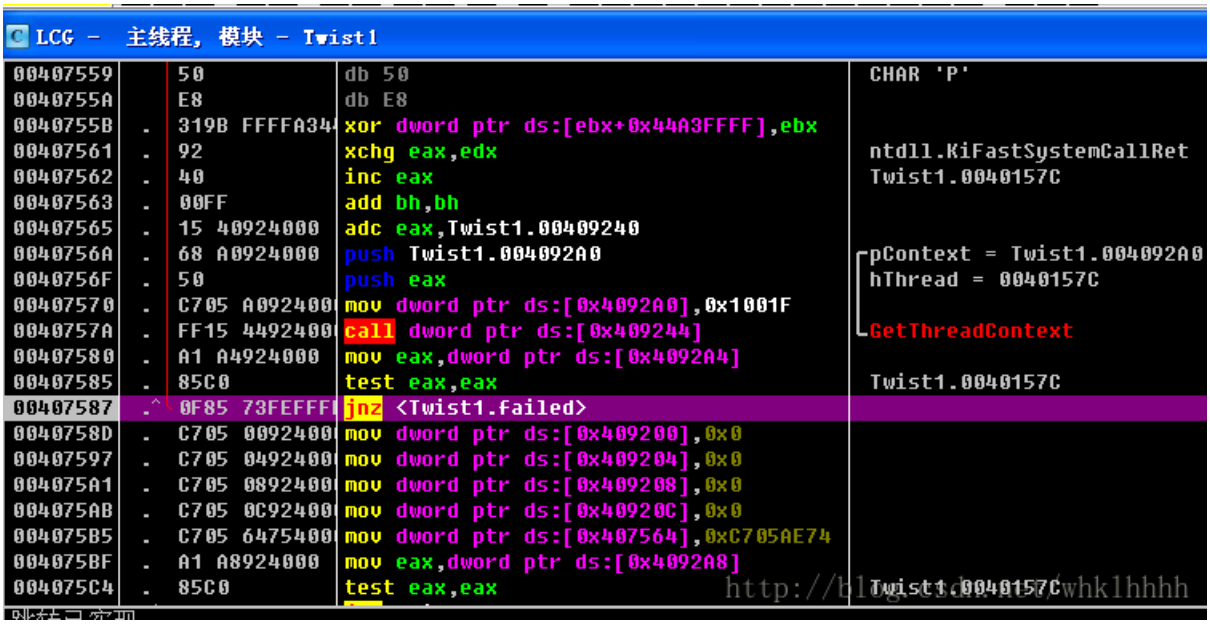


> 复制完后，409150函数就相当于ZwQueryInformationProcess函数，之后就是ZwQueryInformationProcess的反调试，分别是ProcessDebugPort(0x7)、ProcessDebugObjectHandle(0x1E)、ProcessDebugFlags(0x1F)，可以改成nop或改返回值进行绕过

往后的call最好跟进去识别是在做什么，后面流程非常杂乱
首先sub_407405将input[0]和input[6]取出放入另一处内存
然后将input[6]与0x36异或，又放入另一处内存



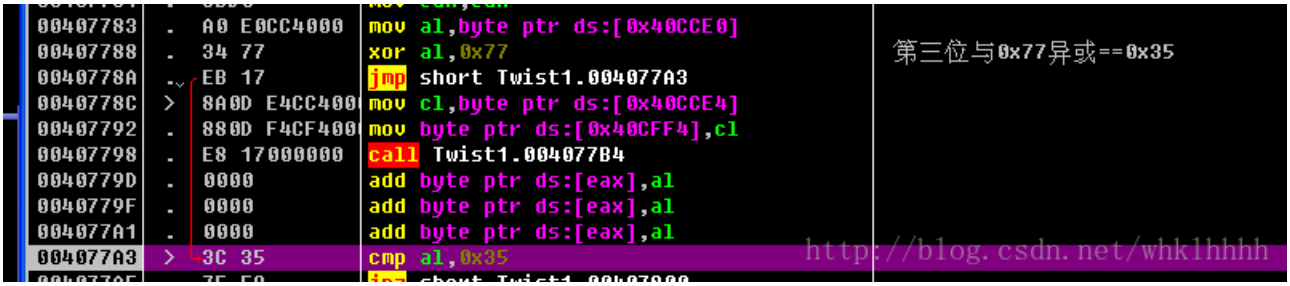
下面通过GetThreadContext进行反调
这个failed标签打上以后可以很容易看出失败跳转，从而避免重来



下面连续进行了若干个比较，实际上是对硬件断点的反调试，逐个对比寄存器

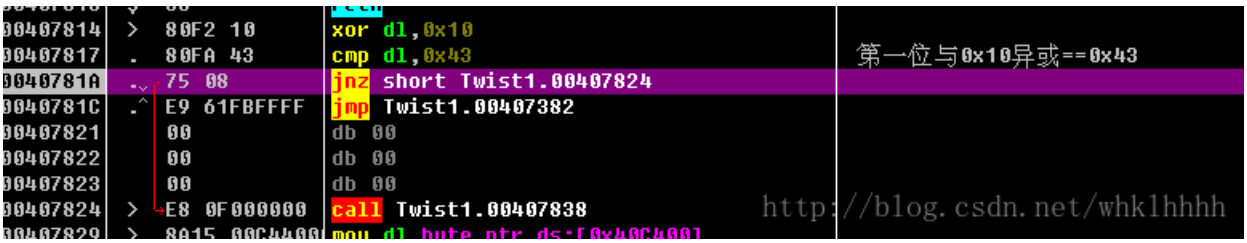
通过以后检查刚才input[6]与0x36异或后是否等于0x36，说明第7位应该为/0，即flag长度为6

然后将input[0]循环右移6位，检查是否与0x49相等，可知第一位为'R'
（0x49=b01001001，循环左移6位为01010010，即'R'）
input[2]与0x77异或，检查是否与0x35相等，可知第三位为'B'



input[1]与0x20异或，检查是否与0x69相等，可知第二位为'I'

下面是一个陷阱，重新拿input[0]与0x10异或，检查是否与0x43相等，相等就GG= =毕竟R是不满足该条件的，因此这个跳不需要爆破



input[3]与0x21异或，检查是否与0x64相等，可知第四位为'E'
PS：这里跟上一步之间经过了一个循环和许多无关指令，让人误以为走错了，可以灵活判断F4到出口

input[4]与0x46异或，检查是否与0x8相等，可知第五位为'N'
input[5]循环左移4位，检查是否与0x14相等，可知第六位为'A'
(0x14=b00010100，循环右移4位为b01000001，即'A'）

整理可知flag

整个流程坑非常多，要很细心的往下拖才行*(:3」∠)*

（反调部分参考http://www.mottoin.com/88447.html）

C. 明日计划

Reversing.kr