




# 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](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/whklhjh/article/details/78146134>

版权



[CrackMe](#) 专栏收录该内容

83 篇文章 2 订阅

订阅专栏

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(:3/ <)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.....

LCG - 主线程, 模块 - Twist1\_0

00407201	00	db 00
00407202	00	db 00
00407203	00	db 00
00407204	00	db 00
00407205	00	db 00
00407206	00	db 00
00407207	00	db 00
00407208	00	db 00
00407209	00	db 00
0040720A	00	db 00
0040720B	00	db 00
0040720C	00	db 00
0040720D	> 0000	add byte ptr ds:[eax],a1
0040720F	00	db 00
00407210	00	db 00
00407211	00	db 00
00407212	00	db 00
00407213	00	db 00
00407214	00	db 00
00407215	00	db 00
00407216	00	db 00
00407217	00	db 00
00407218	00	db 00

<http://blog.csdn.net/whklh1111>

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.....求解感如何解决（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

```
LCG - 主线程, 模块 - Twist1
00407290 . 6A 07      push 0x7
00407292 . 6A FF      push -0x1
00407294 . E8 B71E0000 call Twist1.00409150
00407299 . 8B0D 80914000 mov ecx,dword ptr ds:[0x409180]
0040729F . 85C9      test ecx,ecx
004072A1 ~ 75 5D     jnz short Twist1.00407300 反调
004072A3 . E8 00000000 call Twist1.004072A8
004072A8 $ C705 80914000 mov dword ptr ds:[0x409180],0x0
004072B2 . 6A 00      push 0x0
004072B4 . 6A 04      push 0x4
004072B6 . 68 80914000 push Twist1.00409180
004072BB . 6A 1E      push 0x1E
004072BD . 6A FF      push -0x1
004072BF . E8 8C1E0000 call Twist1.00409150 反调
004072C4 . 3D 530300C0 cmp eax,0xC0000353
004072C9 ~ 0F85 31010000 jnz <Twist1.failed>
004072CF ~ E9 5C010000 jmp Twist1.00407430
004072D4 . 00       db 00
004072D5 . 00       db 00
http://blog.csdn.net/whklhxxx
```

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

00407488	- 8A1D 91B9400	mov bl,byte ptr ds:[0x408991]	取第七位 与0x36异或，放入40c450 <a href="http://blog.csdn.net/whk1hxxx">http://blog.csdn.net/whk1hxxx</a>
0040748E	- 80F3 36	xor bl,0x36	
00407491	- 881D 50C4400	mov byte ptr ds:[0x40C450],bl	

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

```

LCG - 主线程, 模块 - Twist1
00407559 50 db 50 CHAR 'P'
0040755A E8 db E8
0040755B - 319B FFFFA34 xor dword ptr ds:[ebx+0x44A3FFFF],ebx
00407561 - 92 xchg eax,edx
00407562 - 40 inc eax
00407563 - 00FF add bh,bh
00407565 - 15 40924000 adc eax,Twist1.00409240
0040756A - 68 A0924000 push Twist1.004092A0
0040756F 50 push eax
00407570 - C705 A0924000 mov dword ptr ds:[0x4092A0],0x1001F
0040757A - FF15 44924000 call dword ptr ds:[0x409244]
00407580 - A1 A4924000 mov eax,dword ptr ds:[0x4092A4]
00407585 - 85C0 test eax,eax
00407587 - ^ 0F85 73FEFFF jnz <Twist1.failed>
0040758D - C705 00924000 mov dword ptr ds:[0x409200],0x0
00407597 - C705 04924000 mov dword ptr ds:[0x409204],0x0
004075A1 - C705 08924000 mov dword ptr ds:[0x409208],0x0
004075AB - C705 0C924000 mov dword ptr ds:[0x40920C],0x0
004075B5 - C705 64754000 mov dword ptr ds:[0x407564],0xC705AE74
004075BF - A1 A8924000 mov eax,dword ptr ds:[0x4092A8]
004075C4 - 85C0 test eax,eax
    
```

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'

```
00407783 . A0 E0CC4000 mov al,byte ptr ds:[0x40CCE0]
00407788 . 34 77 xor al,0x77
0040778A . EB 17 jmp short Twist1.004077A3
0040778C > 8A0D E4CC4000 mov cl,byte ptr ds:[0x40CCE4]
00407792 . 880D F4CF4000 mov byte ptr ds:[0x40CFF4],cl
00407798 . E8 17000000 call Twist1.004077B4
0040779D . 0000 add byte ptr ds:[eax],al
0040779F . 0000 add byte ptr ds:[eax],al
004077A1 . 0000 add byte ptr ds:[eax],al
004077A3 > 3C 35 cmp al,0x35
004077A5 . 75 F0 jnz short Twist1.004077C0
```

第三位与0x77异或==0x35

<http://blog.csdn.net/whklhxxx>

input[1]与0x20异或，检查是否与0x69相等，可知第二位为'I'

```
004077AC . 80F1 20 xor cl,0x20
004077AF . E8 11000000 call Twist1.004077C5
004077B4 $ E9 E5000000 jmp Twist1.0040789E
004077B9 . 90 nop
004077BA > 58 pop eax
004077BB . 58 pop eax
004077BC . 58 pop eax
004077BD . 59 pop ecx
004077BE . 59 pop ecx
004077BF . 59 pop ecx
004077C0 . 5A pop edx
004077C1 . 5A pop edx
004077C2 . EB 30 jmp short Twist1.004077F4
004077C4 . 00 db 00
004077C5 $ 80F9 69 cmp cl,0x69
004077C8 . 75 F0 jnz short Twist1.004077BA
004077CA . 00 db 00
```

第二位与0x20异或==0x69

Twist1.004077B4  
Twist1.004077B4  
Twist1.004077B4  
Twist1.004077B4  
Twist1.004077B4  
Twist1.004077B4  
Twist1.004077B4  
Twist1.004077B4

<http://blog.csdn.net/whklhxxx>

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

```
00407814 > 80F2 10 xor dl,0x10
00407817 . 80FA 43 cmp dl,0x43
0040781A . 75 08 jnz short Twist1.00407824
0040781C . E9 61FBFFFF jmp Twist1.00407382
00407821 . 00 db 00
00407822 . 00 db 00
00407823 . 00 db 00
00407824 > E8 0F000000 call Twist1.00407838
00407829 > 8015 00C40000 mov dl,byte ptr ds:[0x40C400]
```

第一位与0x10异或==0x43

<http://blog.csdn.net/whklhxxx>

input[3]与0x21异或，检查是否与0x64相等，可知第四位为'E'

PS: 这里跟上一部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

```
0040790E $ 8A15 01C40000 mov dl,byte ptr ds:[0x40C401]
00407914 . 80F2 21 xor dl,0x21
00407917 . C3 retn
00407918 $ 80FA 64 cmp dl,0x64
0040791B . 0F84 AD040000 je Twist1.00407DCE
00407921 . 00 db 00
```

第四位与0x21异或==0x64

<http://blog.csdn.net/whklhxxx>

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