一个简单的e程序



点击按钮后显示消息框



现在测试远程call的调用，

首先附加od，下消息框断点：

INT3 断点项目 5

地址 = 76D2FD31

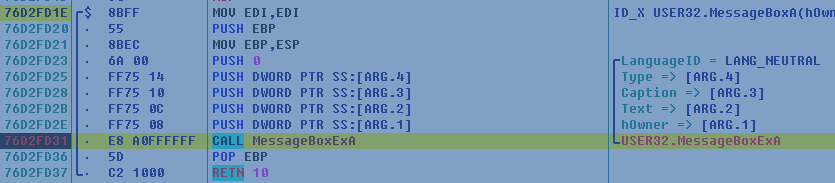
模块 = USER32

状态 = Active

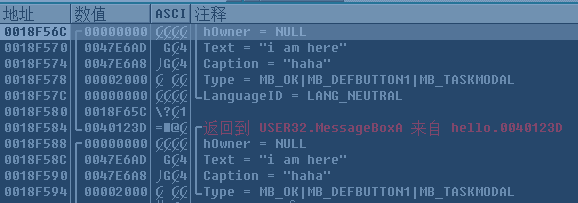
汇编 = CALL MessageBoxExA

注释 = LanguageID = LANG\_NEUTRAL

然后点击“click me ”按钮，od断下；



此时堆栈中内容如下：



显然这里就是要找的调用参数了

但是此时程序还在系统领空，并不在我们所要调试的程序领空内，按ctrl+f9并不能返回到程序领空；这时可以查看堆栈窗口，有如下内容：

CPU Stack

地址 数值 注释

0018F56C /00000000 ; |hOwner = NULL

0018F570 |0047E6AD ; |Text = "i am here"

0018F574 |0047E6A8 ; |Caption = "haha"

0018F578 |00002000 ; |Type = MB\_OK|MB\_DEFBUTTON1|MB\_TASKMODAL

0018F57C |00000000 ; \LanguageID = LANG\_NEUTRAL

0018F580 |0018F65C

0018F584 \0040123D ; /返回到 USER32.MessageBoxA 来自 hello.0040123D

0018F588 /00000000 ; |hOwner = NULL

0018F58C |0047E6AD ; |Text = "i am here"

0018F590 |0047E6A8 ; |Caption = "haha"

0018F594 |00002000 ; \Type = MB\_OK|MB\_DEFBUTTON1|MB\_TASKMODAL

0018F598 |01ECEA88 ; UNICODE "ƔH"

0018F59C |004AE698 ; UNICODE "ʐH"

0018F5A0 |00401160 ; 入口点

0018F5A4 |00000003

其中有如下的注释返回到 USER32.MessageBoxA 来自 hello.0040123D

堆栈中后面还有很多诸如像返回到 。。。。

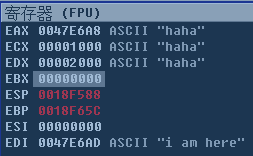
这样的句子，也可以一个个跟踪看看，

但是这里我们看注释就知道返回到 USER32.MessageBoxA 来自 hello.0040123D这一句就差不多是我们要找的call了；

右击反汇编窗口中跟随。



跟随到地址：0040123D，看上面这个call，可以下断点测试一下，确实是这个，此时记录下各个寄存器的值，如下：



此时还可以看到 call调用的

00401237 | CALL NEAR DWORD PTR DS:[<&USER32.MessageBoxA>]

是这样的并不是一个地址值，怎么办？

可以点击这一行，然后在下面的窗口中就可以看到具体地址了。



到这里就可以用工具测试一下了：

打开

并附加要调试程序的进程。

将反汇编指令：

地址 十六进制数据 指令 注释

00401233 |. 52 PUSH EDX /Type

00401234 |. 50 PUSH EAX |Caption

00401235 |. 57 PUSH EDI |Text

00401236 |. 53 PUSH EBX |hOwner

00401237 |. FF15 A0D34700 CALL NEAR DWORD PTR DS:[<&USER32.MessageBoxA>]

寄存器的当前值，以及call地址填入其中，如下

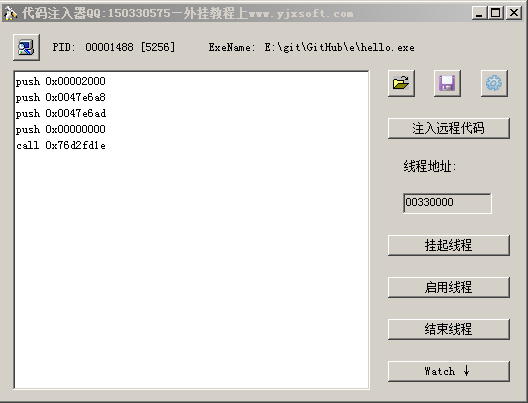
push 0x00002000

push 0x0047e6a8

push 0x0047e6ad

push 0x00000000

call 0x76d2fd1e

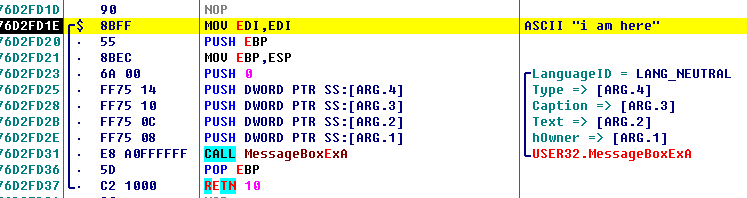


然后点击“注入远程代码”按钮就可以看到效果了，实现了，不需要点击“click me”按钮就可以显示信息框的功能了。

接着上面的分析，前面讲到的call是带了四个参数，而在堆栈窗口中我们只是取了前面第一个看上去很像的关键call，但是也不一定下面就没有同样可用的call了；

重新下消息断点MessageBoxA、

点击按钮断下：



堆栈窗口如下：

CPU Stack

地址 数值 注释

0018F584 [0040123D ; /返回到 USER32.MessageBoxA 来自 hello.0040123D -🡪这个地址我们前面跟踪了，就不看了

0018F588 /00000000 ; |hOwner = NULL

0018F58C |0047E6AD ; |Text = "i am here"

0018F590 |0047E6A8 ; |Caption = "haha"

0018F594 |00002000 ; \Type = MB\_OK|MB\_DEFBUTTON1|MB\_TASKMODAL

0018F598 |01D46D90 ; UNICODE "ƔH"

0018F59C |004AE698 ; UNICODE "ʐH"

0018F5A0 |00401160 ; 入口点

0018F5A4 |00000003

0018F5A8 |00000003

0018F5AC |0000004A

0018F5B0 |00000001

0018F5B4 |01100093

0018F5B8 |00000000

0018F5BC |00000003

0018F5C0 |0000001C

0018F5C4 |0000004A

0018F5C8 |00000001

0018F5CC |01100093

0018F5D0 |00000000

0018F5D4 |00000003

0018F5D8 |00000004

0018F5DC |00000001

0018F5E0 |00000018

0018F5E4 |01100093

0018F5E8 |00000000

0018F5EC |0000004C

0018F5F0 |00000004

0018F5F4 |00000001

0018F5F8 |00000018

0018F5FC |00000002

0018F600 |06010000

0018F604 |16010002

0018F608 \00416B91 ; 返回到 **hello.00416B91 –》我们看看那这个地址**

0018F60C /0018F618

0018F610 |00000004

0018F614 |0018F62C ; 到 PTR ASCII "i am here"

0018F618 |00000000

0018F61C |00000000

0018F620 |00000000

0018F624 \00401040 ; 返回到 hello.00416B70 来自 hello.00401040

0018F628 /00000004

0018F62C |0047E6AD ; ASCII "i am here"

0018F630 |00000000

0018F634 |80000004

0018F638 |00000000

0018F63C |00000000

0018F640 |80000301

0018F644 |0047E6A8 ; ASCII "haha"

0018F648 |00000000

0018F64C |80000004

0018F650 |00000000

0018F654 |00000000

0018F658 |00000000

0018F65C |0018F690

0018F660 \00416C20 ; 返回到 hello.00416C20

0018F664 /01D46D90 ; UNICODE "ƔH"

0018F668 |004AE698 ; UNICODE "ʐH"

0018F66C |01D46D90 ; UNICODE "ƔH"

0018F670 |004AE698 ; UNICODE "ʐH"

0018F674 |00000001

0018F678 |004AE698 ; UNICODE "ʐH"

0018F67C |FFFFFFFF ; ASCII "i"G"

0018F680 |01100073 ; ASCII "i"G"

0018F684 |00000002

0018F688 |00000002

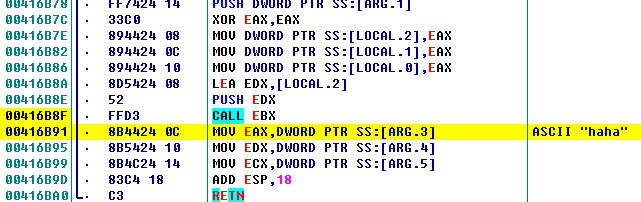
0018F68C |00401004 ; 入口点

0018F690 |000007D8

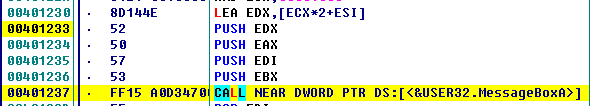
0018F694 \0040C599 ; /返回到 hello.00416BB0 来自 hello.0040C599

我们这里跟踪一下这个地址：**00416B91**

**Ctrl+g 跳转到表达式：00416B91 或者直接反汇编窗口中跟随，cpu窗口如下：**

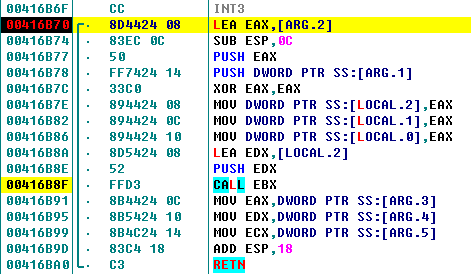


这里有个call ebx，下段点，查看ebx=00401160，回车进入，在这个call内可以看到如下：

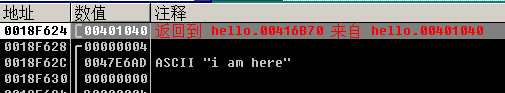


这个是前面我们找到的有参数的call，说明我们找的正确。

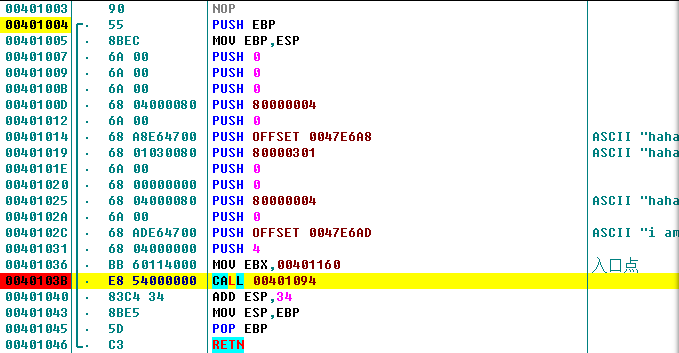
这里我们就在这个call所在子程序首部下断，地址是：00416b70



运行程序到这里，堆栈中：

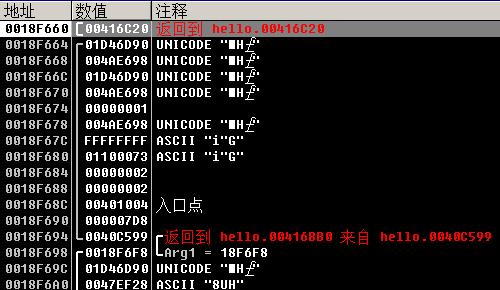


然反汇编跟随，

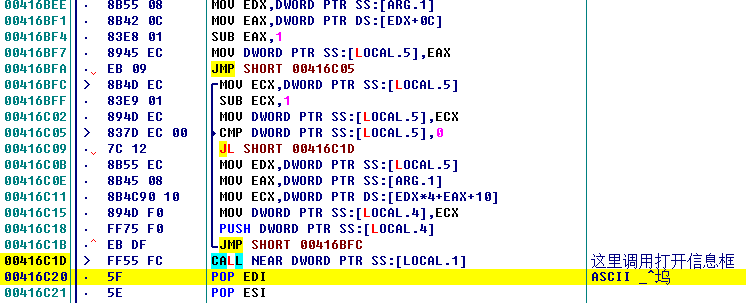


进入call 00401094,查看call没有跟错。也可以测试这个call，发现不行程序会关闭掉。

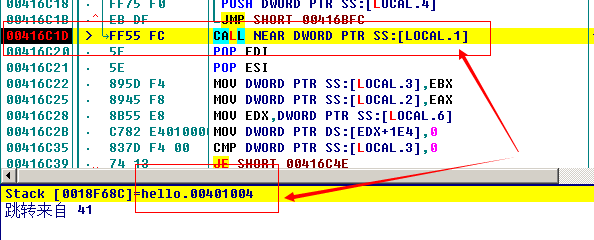
然后就在首部下断。堆栈窗口中



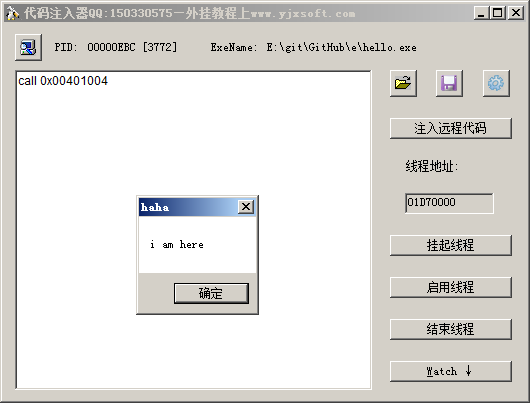
反汇编跟随：



这里是一个较大的子程序了，这个call的调用是有条件的，前面有个跳转，满足条件 才会调用到，实际上就是鼠标点击按钮动作，点击这个按钮，才会执行这个call，在这个call上下断，找出这个地址，也就是上一次的下断的地址：0x00401004



即，无参的调用带了call 00401004，然后测试一下，成功！



并且只要调用就可以无限次弹出信息框：

这时原本程序中不可能实现的。