

第十四章：NAG 窗口（我不是在说你妈）^{注①}

一、简介

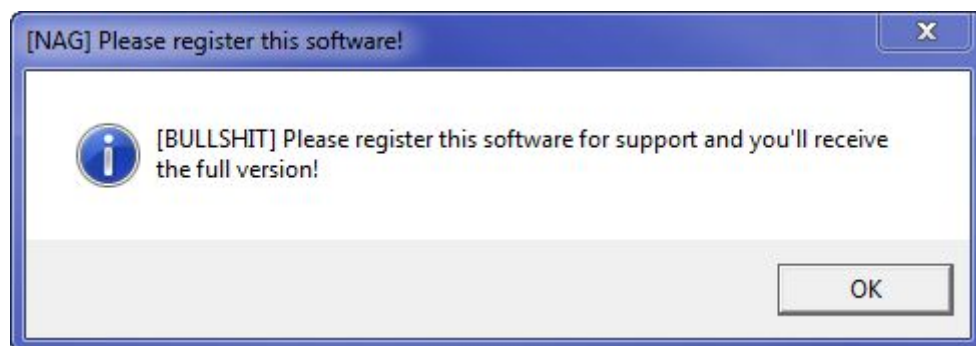
Nags，或者叫 Nag 窗口，是普通的消息对话框。它弹出来是提醒你你的试用结束了、你需要注册、关于访问网站的提醒...。基本上任何事它都要唠叨，而且还是不必要的（像大多数的 boss 一样😏）。许多免费软件之所以免费，是因为它们充满了 nag（广告、限时试用、重定向）。商业软件通常也有这些玩意儿，提醒你“你只剩下 18 天来使用此产品”等等。在逆向工程领域，除掉 nag 窗口是一个中心主题，有时候也提出了很多挑战。本章我们将会研究两个有 nag 的程序。我们将会绕过它们，之后它们就再也不会显示了，然后再打上补丁，这样它们就永远不会回来了。

我也会介绍一个新的 Olly 插件，叫 IDAFicator。它有许多特点和设置。你可以在[工具](#)页下载该插件。因为它有如此多的特性，我也在下载中包含了 IDAFicator 作者写的教程。我强烈推荐你看看教程，因为该插件有许多非常酷的特性。

你可以在[教程](#)页下载相关文件以及本文的 PDF 版。

二、第一个应用程序

我们将要研究的第一个二进制文件是 Nag1.exe。程序一运行就会弹出 nag：



它明摆着告诉你这就是一个 cracker 写的😏。不管怎么样，点了 OK 就可以看到主窗口：

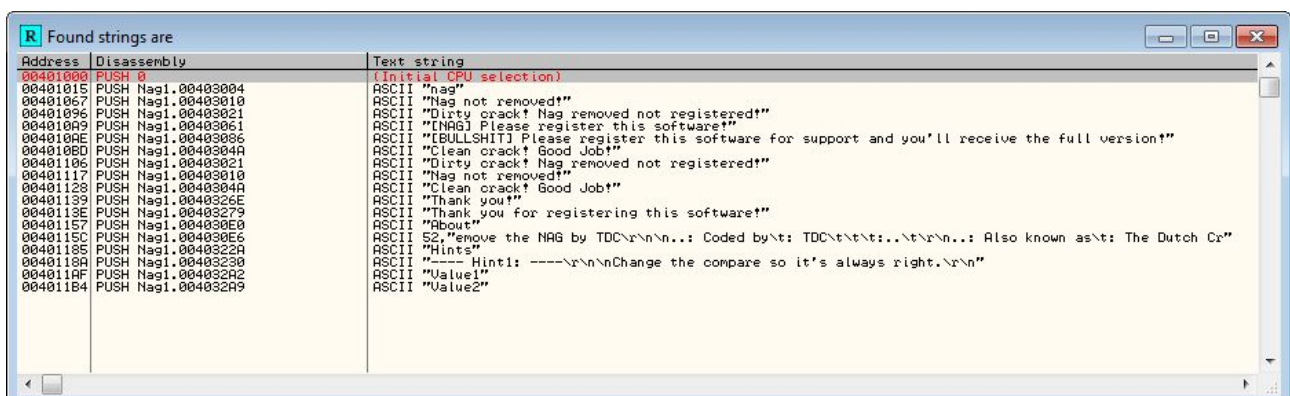
注①：标题中的“我不是在说你妈”可不是骂人的，因为 nag 有“唠叨”的意思，而英文的标题就是“Nags”，没有其他多余的字，所以你懂的。



注意，它说“Nag not removed!”。我情不自禁的就点了那个“Hints”按钮，然后给了一些非常详细的信息（译者注：我咱们没觉得很详细）：



Gee，谢谢。Oilly 载入应用，咱们试试老方法——搜索字符串：



运气还不错。你可以在 4010AE 处看到 nag 窗口中的文本。双击它，咱们就跳到了 nag 窗口被创建的地方：

```
0040107B  . 803D B0324000 03 CMP BYTE PTR DS:[4032B01],3
00401082  74 12 JE SHORT Nag1.00401096
00401084  . 803D B0324000 02 CMP BYTE PTR DS:[4032B01],2
0040108B  74 1A JE SHORT Nag1.004010A7
0040108D  . 803D B0324000 01 CMP BYTE PTR DS:[4032B01],1
00401094  74 27 JE SHORT Nag1.004010BD
00401096  > 68 21304000 PUSH Nag1.00403021
00401098  6A 73 PUSH 73
0040109D  FF75 08 PUSH [ARG.1]
004010A0  E8 6B010000 CALL <JMP.&user32.SetDlgItemTextA>
004010A5  EB 27 JMP SHORT Nag1.004010CE
004010A7  > 6A 40 PUSH 40
004010A9  68 61304000 PUSH Nag1.00403061
004010AE  68 86304000 PUSH Nag1.00403086
004010B3  FF75 08 PUSH [ARG.1]
004010B6  E8 49010000 CALL <JMP.&user32.MessageBoxA>
004010BB  EB 11 JMP SHORT Nag1.004010CE
004010BD  > 68 4A304000 PUSH Nag1.0040304A
004010C2  6A 73 PUSH 73
004010C4  FF75 08 PUSH [ARG.1]
004010C7  E8 44010000 CALL <JMP.&user32.SetDlgItemTextA>
004010CC  EB 00 JMP SHORT Nag1.004010CE
004010CE  > E9 D6000000 JMP Nag1.004011A9
004010D3  317D 0C 11010000 CMP [ARG.2],111
004010DA  0F85 B9000000 JNZ Nag1.00401199
004010E0  837D 10 6F CMP [ARG.3],6F
004010E4  75 69 JNZ SHORT Nag1.0040114F
004010E6  E8 C4000000 CALL Nag1.004011AF
004010EB  . 803D B0324000 03 CMP BYTE PTR DS:[4032B01],3
```

```
Text = "Dirty crack! Nag removed not registered!"
ControlID = 73 (115.)
hWnd = 7EFDE000
SetDlgItemTextA

Style = MB_OK!MB_ICONASTERISK!MB_APPLMODAL
Title = "[NAG] Please register this software!"
Text = "[BULLSHIT] Please register this software for support and y
hOwner = 7EFDE000
MessageBoxA

Text = "Clean crack! Good Job!"
ControlID = 73 (115.)
hWnd = 7EFDE000
SetDlgItemTextA
```

嗯，它上面有一个有趣的字符串，不过咱们现在先不管。咱们看看 4010A7 处，也就是调用 MessageBoxA 函数的第一行，看看哪里调用了它：

```
0040107B  . 803D B0324000 03 CMP BYTE PTR DS:[4032B01],3
00401082  74 12 JE SHORT Nag1.00401096
00401084  . 803D B0324000 02 CMP BYTE PTR DS:[4032B01],2
0040108B  74 1A JE SHORT Nag1.004010A7
0040108D  . 803D B0324000 01 CMP BYTE PTR DS:[4032B01],1
00401094  74 27 JE SHORT Nag1.004010BD
00401096  > 68 21304000 PUSH Nag1.00403021
00401098  6A 73 PUSH 73
0040109D  FF75 08 PUSH [ARG.1]
004010A0  E8 6B010000 CALL <JMP.&user32.SetDlgItemTextA>
004010A5  EB 27 JMP SHORT Nag1.004010CE
004010A7  > 6A 40 PUSH 40
004010A9  68 61304000 PUSH Nag1.00403061
004010AE  68 86304000 PUSH Nag1.00403086
004010B3  FF75 08 PUSH [ARG.1]
```

```
Text = "Dirty crack! Nag removed not registered!"
ControlID = 73 (115.)
hWnd = 7EFDE000
SetDlgItemTextA

Style = MB_OK!MB_ICONASTERISK!MB_APPLMODAL
Title = "[NAG] Please register this software!"
Text = "[BULLSHIT] Please register this software for support and y
hOwner = 7EFDE000
MessageBoxA
```

我们能看到 40108B 处的 JE 指令调用了它，而且刚好在一个比较指令的后面。好吧，这个场景我们已经很属性了😁。咱们在 JE 指令那设置一个 BP：

```
00401071  . E8 9A010000 CALL <JMP.&user32.SetDlgItemTextA>
00401076  . E8 34010000 CALL Nag1.004011AF
0040107B  . 803D B0324000 03 CMP BYTE PTR DS:[4032B01],3
00401082  74 12 JE SHORT Nag1.00401096
00401084  . 803D B0324000 02 CMP BYTE PTR DS:[4032B01],2
0040108B  74 1A JE SHORT Nag1.004010A7
0040108D  . 803D B0324000 01 CMP BYTE PTR DS:[4032B01],1
00401094  74 27 JE SHORT Nag1.004010BD
00401096  > 68 21304000 PUSH Nag1.00403021
00401098  6A 73 PUSH 73
0040109D  FF75 08 PUSH [ARG.1]
004010A0  E8 6B010000 CALL <JMP.&user32.SetDlgItemTextA>
004010A5  EB 27 JMP SHORT Nag1.004010CE
004010A7  > 6A 40 PUSH 40
004010A9  68 61304000 PUSH Nag1.00403061
004010AE  68 86304000 PUSH Nag1.00403086
004010B3  FF75 08 PUSH [ARG.1]
004010B6  E8 49010000 CALL <JMP.&user32.MessageBoxA>
004010BB  EB 11 JMP SHORT Nag1.004010CE
004010BD  > 68 4A304000 PUSH Nag1.0040304A
004010C2  6A 73 PUSH 73
004010C4  FF75 08 PUSH [ARG.1]
004010C7  E8 44010000 CALL <JMP.&user32.SetDlgItemTextA>
004010CC  EB 00 JMP SHORT Nag1.004010CE
```

```
Text = "Dirty crack! Nag removed not registered!"
ControlID = 73 (115.)
hWnd = 7EFDE000
SetDlgItemTextA

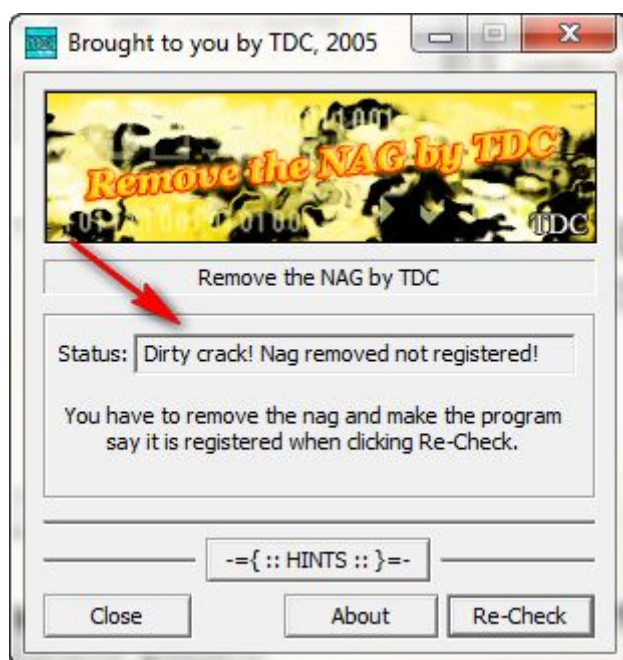
Style = MB_OK!MB_ICONASTERISK!MB_APPLMODAL
Title = "[NAG] Please register this software!"
Text = "[BULLSHIT] Please register this software for support and y
hOwner = 7EFDE000
MessageBoxA

Text = "Clean crack! Good Job!"
ControlID = 73 (115.)
hWnd = 7EFDE000
SetDlgItemTextA
```

运行程序。然后我们断在了那个 BP，能够看到我们将跳转到 nag 窗口那，所以不能让它跳：

```
C 0 ES 002
P 1 CS 002
A 0 SS 002
Z 0 DS 002
S 0 FS 005
T 0 GS 002
D 0
I 0
```

接着运行程序：



这就是“Dirty crack!”的出处，显然咱们的补丁打的还不够。重启应用，Oilly断在了BP那。再次将0标志位清零：

```

C 0  ES 0002
P 1  CS 0002
A 0  SS 0002
Z 0  DS 0002
S 0  FS 0005
T 0  GS 0002
D 0
R 0

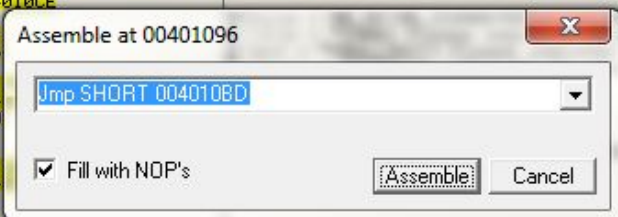
```

咱们单步执行两次到下一个跳转那。你可能已经猜出来了，这个跳转应该是跳到好消息那的，而不是到坏消息那：

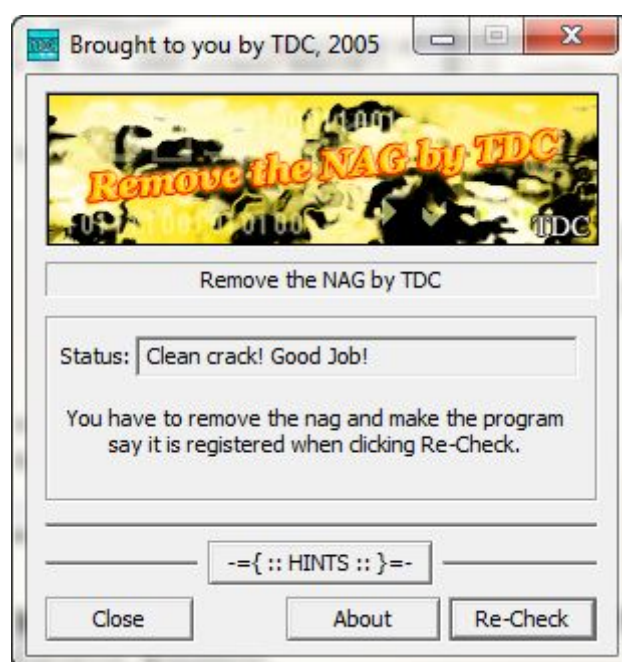
00401082	74 12	JE SHORT Nag1.00401096	
00401084	803D B0324000 02	CMP BYTE PTR DS:[4032B01,2]	
00401086	74 1A	JE SHORT Nag1.004010A7	
00401088	803D B0324000 01	CMP BYTE PTR DS:[4032B01,1]	
00401094	EB 27	JMP SHORT Nag1.004010BD	
00401096	68 21304000	PUSH Nag1.00403021	
00401098	6A 73	PUSH 73	
0040109D	FF75 08	PUSH [ARG.1]	
004010A0	E8 6B010000	CALL <JMP.&user32.SetDlgItemTextA>	Text = "Dirty crack! Nag removed not registered!" ControlID = 73 (115.) hWnd = 002B0668 ('Brought to you by TDC, 2005',class='#32770')
004010A5	EB 27	JMP SHORT Nag1.004010CE	SetDlgItemTextA
004010A7	6A 40	PUSH 40	
004010A9	68 61304000	PUSH Nag1.00403061	
004010AE	68 86304000	PUSH Nag1.00403086	
004010B3	FF75 08	PUSH [ARG.1]	
004010B6	E8 49010000	CALL <JMP.&user32.MessageBoxA>	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL Title = "[NAG] Please register this software!" Text = "[BULLSHIT] Please register this software for support and yo hWnd = 002B0668 ('Brought to you by TDC, 2005',class='#32770')
004010BB	EB 11	JMP SHORT Nag1.004010CE	MessageBoxA
004010BD	68 4A304000	PUSH Nag1.0040304A	
004010C2	6A 73	PUSH 73	
004010C4	FF75 08	PUSH [ARG.1]	
004010C7	E8 44010000	CALL <JMP.&user32.SetDlgItemTextA>	Text = "Clean crack! Good Job!" ControlID = 73 (115.) hWnd = 002B0668 ('Brought to you by TDC, 2005',class='#32770')
004010CC	EB 00	JMP SHORT Nag1.004010CE	SetDlgItemTextA
004010CE	E9 D6000000	JMP Nag1.004011A9	
004010D3	817D 0C 11010000	CMP [ARG.2],111	
004010DA	0F85 B9000000	JNZ Nag1.00401199	
004010E0	837D 10 6F	CMP [ARG.3],6F	
004010F4	7E 69	JNZ SHORT Nag1.0040114E	

咱们打个补丁，让它直接跳：

00401078	803D B0324000 03	CMP BYTE PTR DS:[4032B01,3]	
00401082	74 12	JE SHORT Nag1.00401096	
00401084	803D B0324000 02	CMP BYTE PTR DS:[4032B01,2]	
00401086	74 1A	JE SHORT Nag1.004010A7	
00401088	803D B0324000 01	CMP BYTE PTR DS:[4032B01,1]	
00401094	EB 27	JMP SHORT Nag1.004010BD	
00401096	68 21304000	PUSH Nag1.00403021	
00401098	6A 73	PUSH 73	
0040109D	FF75 08	PUSH [ARG.1]	
004010A0	E8 6B010000	CALL <JMP.&user32.SetDlgItemTextA>	Text = "Dirty crack! Nag removed not register ControlID = 73 (115.) hWnd = 002B0668 ('Brought to you by TDC, 2005
004010A5	EB 27	JMP SHORT Nag1.004010CE	SetDlgItemTextA
004010A7	6A 40	PUSH 40	
004010A9	68 61304000	PUSH Nag1.00403061	
004010AE	68 86304000	PUSH Nag1.00403086	
004010B3	FF75 08	PUSH [ARG.1]	
004010B6	E8 49010000	CALL <JMP.&user32.MessageBoxA>	
004010BB	EB 11	JMP SHORT Nag1.004010CE	
004010BD	68 4A304000	PUSH Nag1.0040304A	
004010C2	6A 73	PUSH 73	
004010C4	FF75 08	PUSH [ARG.1]	
004010C7	E8 44010000	CALL <JMP.&user32.SetDlgItemTextA>	
004010CC	EB 00	JMP SHORT Nag1.004010CE	
004010CE	E9 D6000000	JMP Nag1.004011A9	
004010D3	817D 0C 11010000	CMP [ARG.2],111	
004010DA	0F85 B9000000	JNZ Nag1.00401199	
004010E0	837D 10 6F	CMP [ARG.3],6F	
004010F4	7E 69	JNZ SHORT Nag1.0040114E	



运行程序，可以看到咱们弄对了：



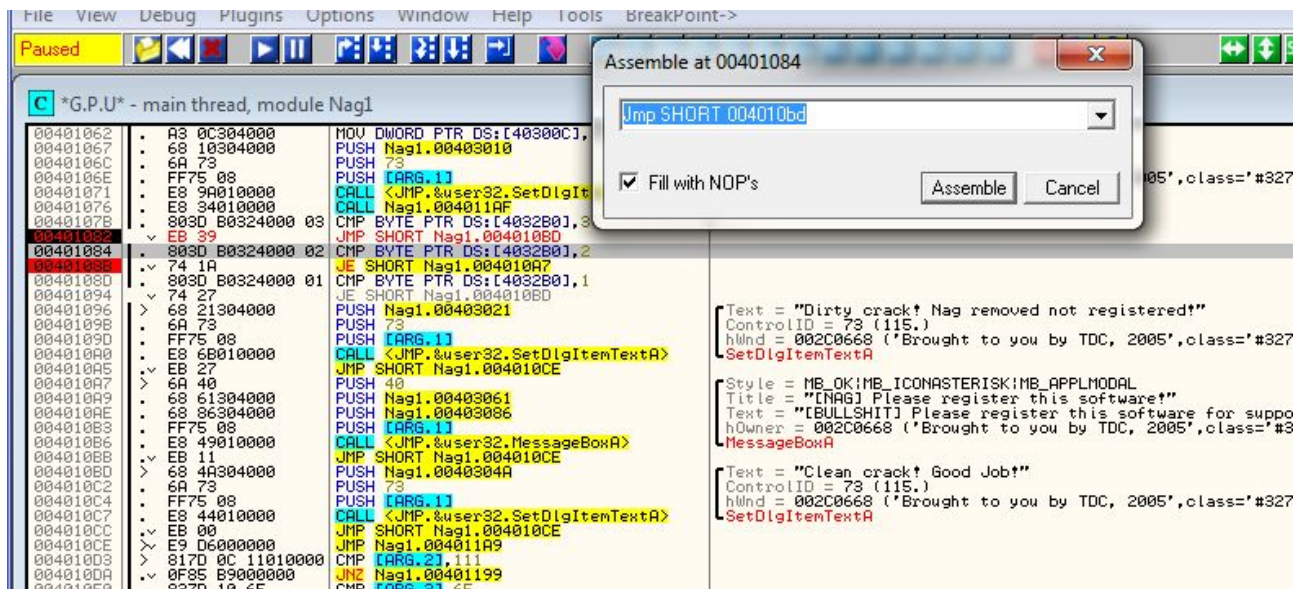
很明显，现在这个补丁就可以解决这个程序，咱们回到 40108B 那（咱们原来将 0 标志位清零的地方）给它打上补丁让它永远不会跳转。保存这两个补丁程序就会很好的运行。不过我也想向你展示（我以前提到过，如果我没有提到的话，那我应该提到的），通常总是有别的方法来给程序打补丁。重启应用，在我们的 BP 处断下来：

00401071 00401076 0040107B 00401082 00401084 00401085 0040108D 00401094 00401096 00401098 0040109D 004010A0 004010A5 004010A7 004010A9 004010AE 004010B3 004010B6 004010B8 004010BD 004010C2 004010C4 004010C7 004010CC 004010CE 004010D3	FF75 08 E8 9A010000 E8 34010000 803D B0324000 03 74 12 803D B0324000 02 74 1A 803D B0324000 01 EB 27 68 21304000 6A 73 FF75 08 E8 6B010000 EB 27 6A 40 68 61304000 68 86304000 FF75 08 E8 49010000 EB 11 68 4A304000 6A 73 FF75 08 E8 44010000 EB 00 E9 D6000000 817D 0C 11010000	PUSH EAX CALL <JMP.&user32.SetDlgItemTextA> CALL Nag1.004011AF CMP BYTE PTR DS:[4032B0],3 JE SHORT Nag1.00401096 CMP BYTE PTR DS:[4032B0],2 JE SHORT Nag1.004010A7 CMP BYTE PTR DS:[4032B0],1 JMP SHORT Nag1.004010BD PUSH Nag1.00403021 PUSH 73 PUSH [ARG.1] CALL <JMP.&user32.SetDlgItemTextA> JMP SHORT Nag1.004010CE PUSH 40 PUSH Nag1.00403061 PUSH Nag1.00403066 PUSH [ARG.1] CALL <JMP.&user32.MessageBoxA> JMP SHORT Nag1.004010CE PUSH Nag1.0040304A PUSH 73 PUSH [ARG.1] CALL <JMP.&user32.SetDlgItemTextA> JMP SHORT Nag1.004010CE JMP Nag1.004011A9 CMP [ARG.2],1	SetDlgItemTextA [Text = "Dirty crack! Nag removed not registered!" ControlID = 73 (115.) hWnd = 001C0008 SetDlgItemTextA [Style = MB_OK MB_ICONASTERISK MB_APPLMODAL Title = "[NAG] Please register this software!" Text = "[BULLSHIT] Please register this software for support hOwner = 001C0008 MessageBoxA [Text = "Clean crack! Good Job!" ControlID = 73 (115.) hWnd = 001C0008 SetDlgItemTextA
--	---	---	--

这些指令与下面这些（用高级语言表示）类似：

```
if (contents of 4032B0 == 3)
    jump "Dirty Crack"
else if ( contents of 4032B0 == 2)
    jump to "Show Nag Screen"
else if (contents of 4032B0 == 1)
    jump to Good Boy Msg
else
    Display "Dirty Crack"
```


我们知道 nag 窗口默认是要显示的，4032B0 处内存总是等于 2，因为跳转得实现才行。如果我们跳过整个 if/then 语句，直接跳到好消息怎么样？所以如果我们将最开始的第一个跳转替换成跳到好消息的跳转，那么我们就只需要一个补丁就行。试试看：

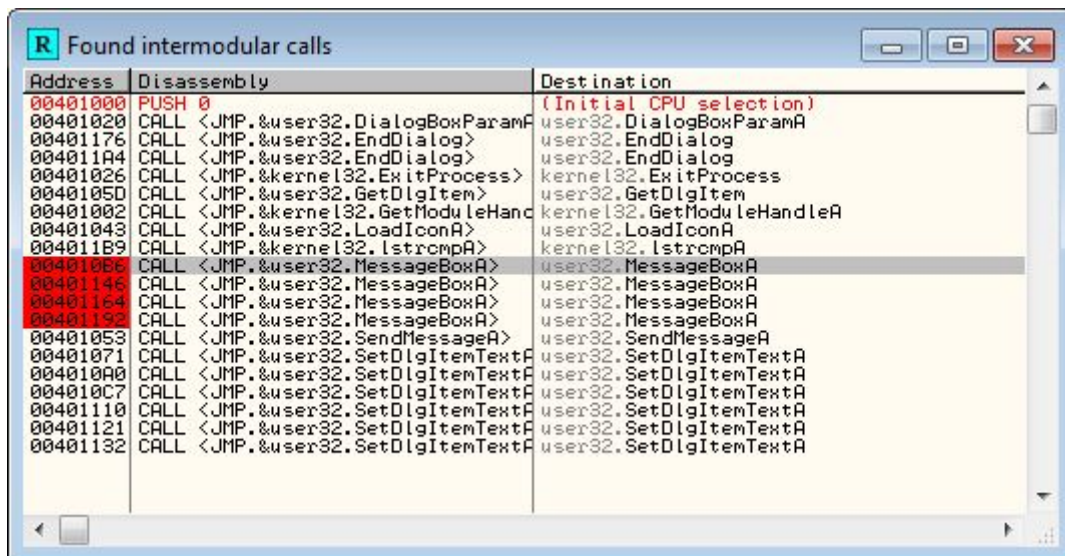


运行下程序：



可以看到结果是一样的。另外，可以思想下更加优雅的方法，“4032B0 中的内容总是等于 2，不过要显示好消息的话它就需要等于 1，那么为什么不在内存中就放一个 1 呢，那样的话就会一直显示好消息了呀？”你应该试试这个。重启应用，点一下数据窗口，转到 4032B0，用二进制编辑将它改成 1。起作用了没？

需要记住的另一件事是，总是有别的方法可以找到我们正在寻找的代码块。比如，如果本例中我们不能用字符串，我们就可以用 搜索模块间调用（译者注：相关内容在第九章）：



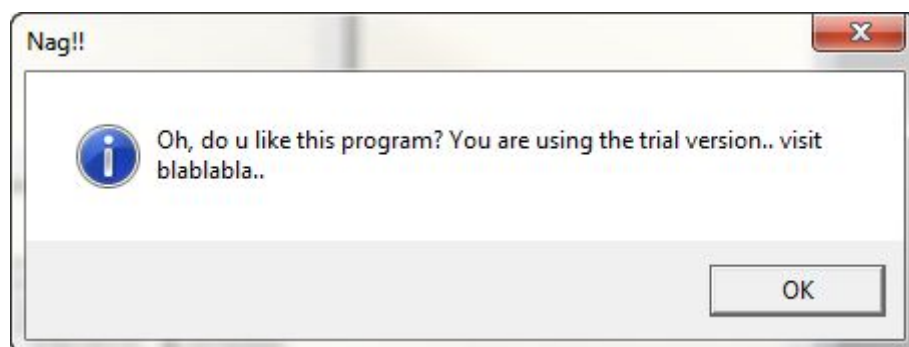
注意有四个对 MessageBoxA 的调用。右键其中一个，选择“Place a breakpoint on every call to MessageBoxA”。当你运行程序时，在显示任何东西之前，我们会停在下面这行代码：



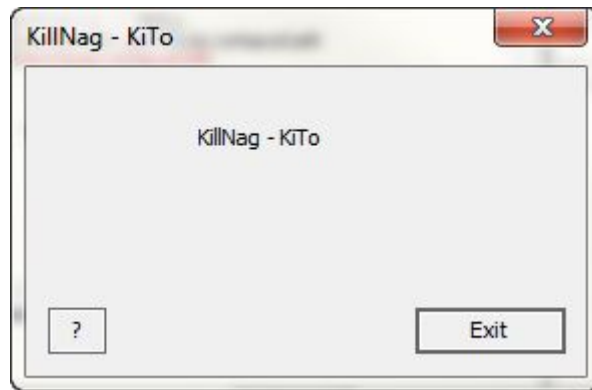
是不是很熟悉？它就是 nag 消息框!! 所以要记住总是有不只一种方法可以完成一些事情。不久我们将会学习其他的技术（像窗口消息处理），会给你更大包的技巧。

三、第二个应用程序

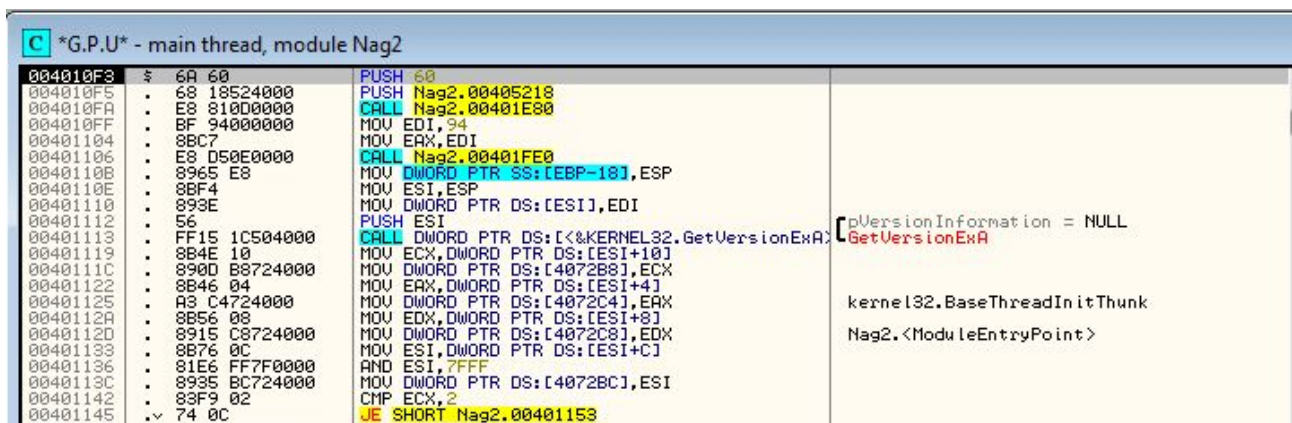
现在咱们来看看 Nag2.exe。看起来差不多，不过我们将用不同的方法来解决它。启动程序的时候，我们看到了意料中的 nag:



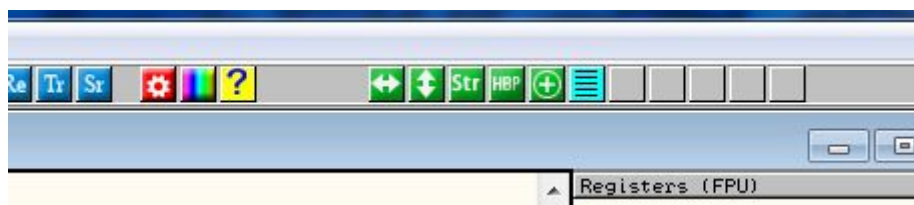
在点了 OK 后，我们看到了主窗口：



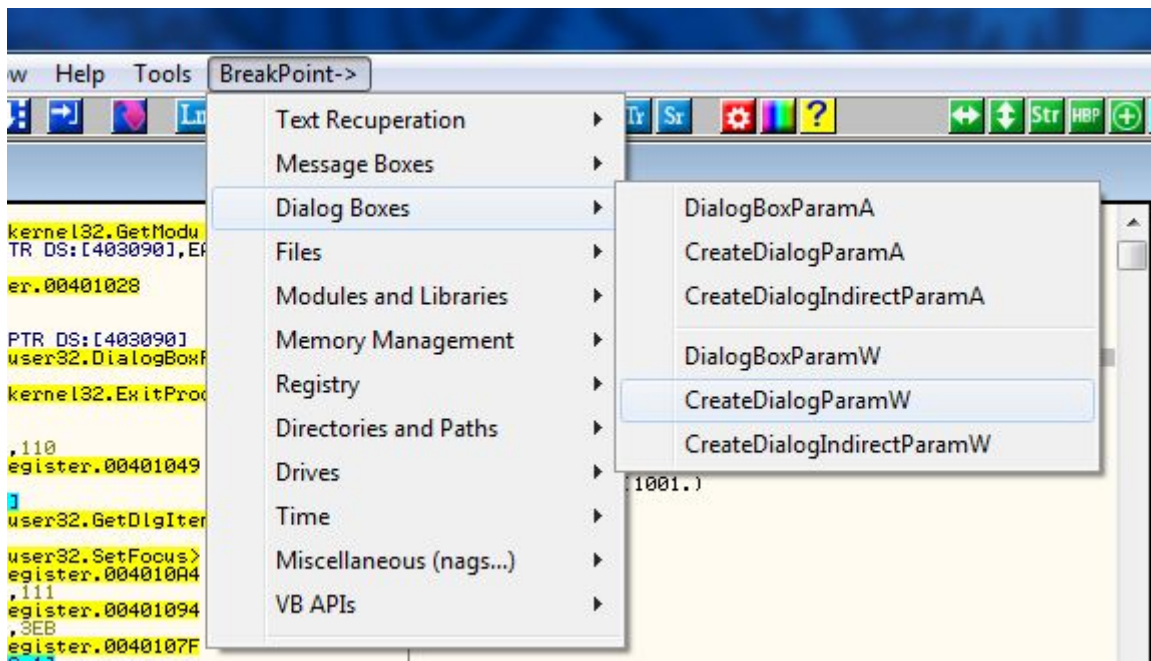
此时我关了程序，将其载入到 Olly 中：



首先，咱们来看看有没有字符串。这里我想提的一件事是 IDAFicator 插件。在众多的添加功能中，它在程序顶部提供了一组按钮，让搜索字符串变得更加的简单。当点击字符串按钮（Str）时，它会显示 ASCII 和 Unicode 两种字符串，并自动的将光标移到顶部，这样你就不用自己滚动到顶部了。下面是那些按钮的样子：

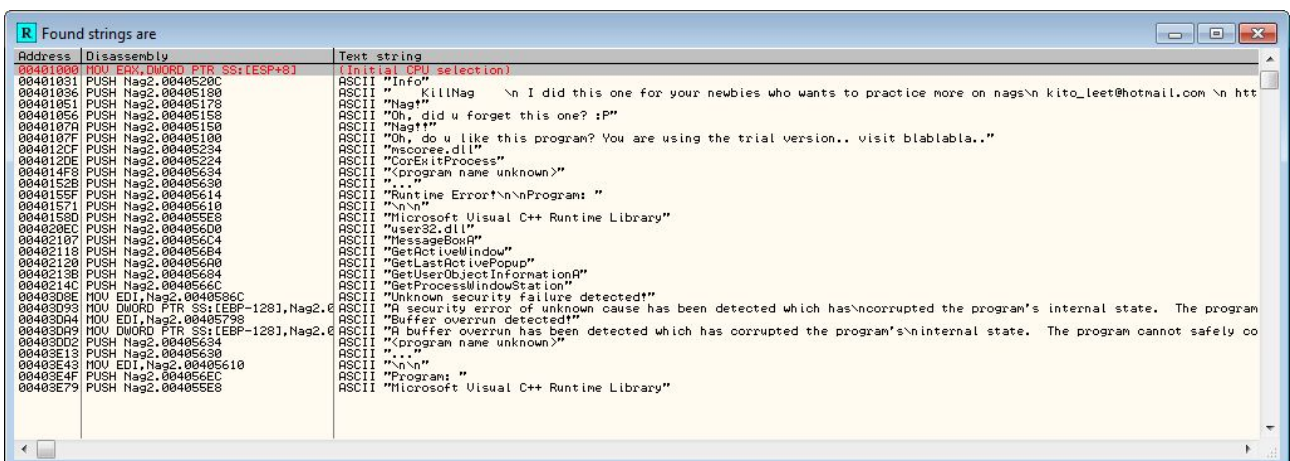


第一个按钮（有左右箭头的那个）是向前和往后。比如，你点击一个 CALL，然后按一下 enter 键转到该 CALL，点一下第一个图标你会回到 CALL 指令那。右击你就会前进一步。第二个按钮会尝试找出当前函数的开头，右击则会尝试找到结尾。下一个就是字符串按钮。再下一个是硬件断点按钮。它会弹出一个很漂亮的对话框来显示你设置的所有硬件断点。非常的便利。十字图标打开你的应用程序所在的文件夹，列表图标会弹出一个对话框以便于你输入多行汇编代码，如果你打算修改 exe 文件的大部分代码时可以用这个。你会注意到一个叫“Breakpoints”的新的菜单项，它会弹出一个下拉菜单，里面是许多用到的 API，你可以自动对它们设置断点：

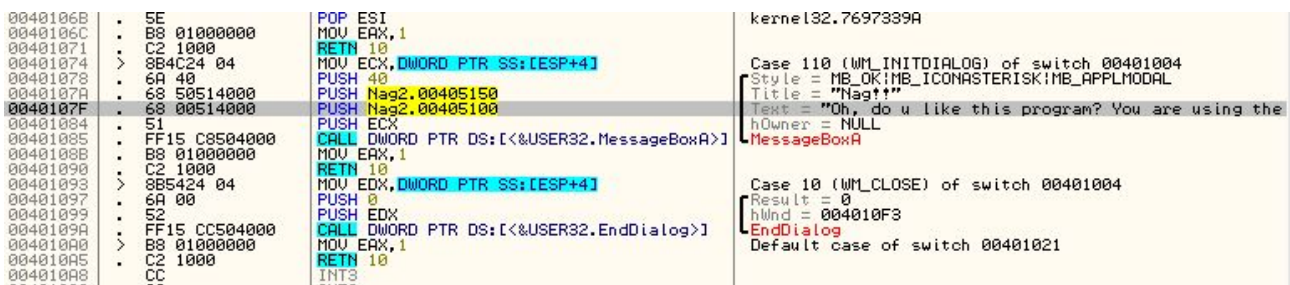


最后，有一个上下文菜单可以让你恢复隐藏的菜单，我们会在后面的章节中讨论。

咱们继续，点击新的按钮栏中的字符串（“Str”）按钮：



在第七行，我们看到了 nag 上面的文本，双击该行：



看到了 nag 的实现方法。是一个自包含的方法（在它的上面和下面各有一个 RETN 指令），所以我们知道它是在某个地方被调用的。点击 401074 那行，也就是该自包含方法的第一行，看看它是从哪被调用：

00401000	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	Switch (cases 10..111)
00401004	83E8 10	SUB EAX,10	
00401007	0F84 86000000	JE Nag2.00401093	
0040100D	2D 00010000	SUB EAX,100	
00401012	74 60	JE SHORT Nag2.00401074	
00401014	48	DEC EAX	kernel32.BaseThreadInitThunk
00401015	74 05	JE SHORT Nag2.0040101C	kernel32.BaseThreadInitThunk; Default case of switch
00401017	33C0	XOR EAX,EAX	
00401019	C2 1000	RETN 10	Case 111 (WM_COMMAND) of switch 00401004
0040101C	0FB74424 0C	MOVZX EAX,WORD PTR SS:[ESP+C]	Switch (cases 3E9..3EA)
00401021	2D E9030000	SUB EAX,3E9	kernel32.BaseThreadInitThunk
00401026	74 22	JE SHORT Nag2.0040104A	
00401028	48	DEC EAX	
00401029	75 75	JNZ SHORT Nag2.004010A0	
0040102B	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]	Case 3EA of switch 00401021
0040102F	6A 40	PUSH 40	Style = MB_OK;MB_ICONASTERISK;MB_APPLMODAL
00401031	68 0C524000	PUSH Nag2.0040520C	Title = "Info"
00401036	68 80514000	PUSH Nag2.00405180	Text = "KillNag \n I did this one for your new"
0040103B	50	PUSH EAX	hOwner = 76973388
0040103C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
00401042	B8 01000000	MOV EAX,1	
00401047	C2 1000	RETN 10	Case 3E9 of switch 00401021
0040104A	56	PUSH ESI	Style = MB_OK;MB_ICONHAND;MB_APPLMODAL
0040104B	8B7424 08	MOV ESI,DWORD PTR SS:[ESP+8]	Title = "Nag!"
0040104F	6A 10	PUSH 10	Text = "Oh, did u forget this one? :P"
00401051	68 78514000	PUSH Nag2.00405178	hOwner = NULL
00401056	68 58514000	PUSH Nag2.00405158	Result = 1
0040105B	56	PUSH ESI	hWnd = NULL
0040105C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	EndDialog
00401062	6A 01	PUSH 1	kernel32.7697339A
00401064	56	PUSH ESI	
00401065	FF15 CC504000	CALL DWORD PTR DS:[&USER32.EndDialog]	
00401068	5E	POP ESI	
0040106C	B8 01000000	MOV EAX,1	
00401071	C2 1000	RETN 10	
00401074	8B4C24 04	MOV ECX,DWORD PTR SS:[ESP+4]	Case 110 (WM_INITDIALOG) of switch 00401004
00401078	6A 40	PUSH 40	Style = MB_OK;MB_ICONASTERISK;MB_APPLMODAL
0040107A	68 50514000	PUSH Nag2.00405150	Title = "Nag!"
0040107F	68 00514000	PUSH Nag2.00405100	Text = "Oh, do u like this program? You are using the"
00401084	51	PUSH ECX	hOwner = NULL
00401085	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
00401088	B8 01000000	MOV EAX,1	
00401093	C2 1000	RETN 10	

可以看到是被 401012 的 JE 指令调用。咱们在这里设置一个断点，运行程序：

00401000	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	Switch (cases 10..111)
00401004	83E8 10	SUB EAX,10	
00401007	0F84 86000000	JE Nag2.00401093	
0040100D	2D 00010000	SUB EAX,100	
00401012	74 60	JE SHORT Nag2.00401074	
00401014	48	DEC EAX	kernel32.BaseThreadInitThunk
00401015	74 05	JE SHORT Nag2.0040101C	kernel32.BaseThreadInitThunk; Default case of switch
00401017	33C0	XOR EAX,EAX	
00401019	C2 1000	RETN 10	Case 111 (WM_COMMAND) of switch 00401004
0040101C	0FB74424 0C	MOVZX EAX,WORD PTR SS:[ESP+C]	Switch (cases 3E9..3EA)
00401021	2D E9030000	SUB EAX,3E9	kernel32.BaseThreadInitThunk
00401026	74 22	JE SHORT Nag2.0040104A	
00401028	48	DEC EAX	
00401029	75 75	JNZ SHORT Nag2.004010A0	
0040102B	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]	
0040102F	6A 40	PUSH 40	
00401031	68 0C524000	PUSH Nag2.0040520C	
00401036	68 80514000	PUSH Nag2.00405180	
0040103B	50	PUSH EAX	
0040103C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	
00401042	B8 01000000	MOV EAX,1	
00401047	C2 1000	RETN 10	
0040104A	56	PUSH ESI	

咱们断在了 JE 指令处。注意它没有调用我们的 nag 窗口代码。原来是我们刚好在 windows 的处理的中间。我们会在另一章深入讨论消息处理，不过目前我们只需要知道所有的 GUI Windows 程序都有一个消息处理程序，并且 Windows 通过它来发送各种消息。根据到达的消息，我们可以添加自己的代码来覆盖 Windows 的普通处理流程。例如，当我们点击窗口上的“X”时，Windows 就会通过消息处理程序发送一个消息说“嘿，用户想要关闭这个窗口”。我们可以让消息通过，这样的话 Windows 就会处理它并关闭窗口，或者我们可以“捕获”这个消息，做任何我们想做的（可能弹出一个对话框说“你还未保存，确定退出？”）。

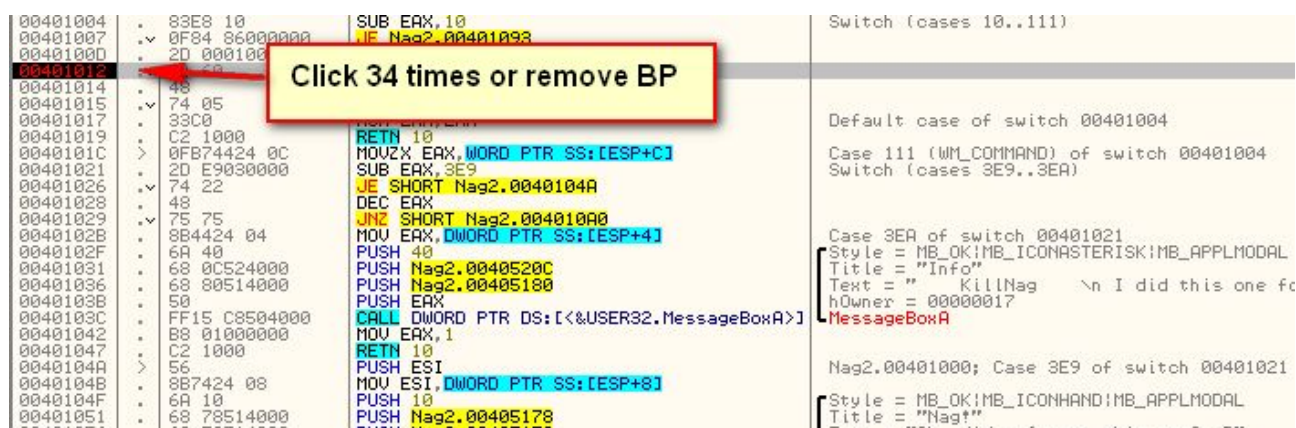
我们的断点刚好在这个的中间，所以已经来到的第一个消息与应用程序想要覆盖的以便于显示 nag 窗口的消息不匹配：

00401007	0F84 86000000	JE Nag2.00401093	
0040100D	2D 00010000	SUB EAX,100	
00401012	74 60	JE SHORT Nag2.00401074	
00401014	48	DEC EAX	Default case of switch 00401004
00401015	74 05	JE SHORT Nag2.0040101C	
00401017	33C0	XOR EAX,EAX	
00401019	C2 1000	RETN 10	Case 111 (WM_COMMAND) of switch 00401004
0040101C	0FB74424 0C	MOVZX EAX,WORD PTR SS:[ESP+C]	Switch (cases 3E9..3EA)
00401021	2D E9030000	SUB EAX,3E9	
00401026	74 22	JE SHORT Nag2.0040104A	
00401028	48	DEC EAX	
00401029	75 75	JNZ SHORT Nag2.004010A0	
0040102B	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]	Case 3EA of switch 00401021
0040102F	6A 40	PUSH 40	Style = MB_OK;MB_ICONASTERISK;MB_APPLMODAL
00401031	68 0C524000	PUSH Nag2.0040520C	Title = "Info"
00401036	68 80514000	PUSH Nag2.00405180	Text = "KillNag \n I did this one for"
0040103B	50	PUSH EAX	hOwner = FFFFFFF0
0040103C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
00401042	B8 01000000	MOV EAX,1	
00401047	C2 1000	RETN 10	
0040104A	56	PUSH ESI	

咱们继续，按 **F9** 运行程序，然后断在同一个 **BP**，不过这一次跳转会实现，将显示 **nag** 窗口。咱们让 **Ollly** 别显示它：

```
C 0 ES 002E
P 1 CS 002E
A 0 SS 002E
Z 0 DS 002E
S 0 FS 005E
T 0 GS 002E
O 0
O 0 LastErr
```

现在，如果我们留着这个断点，通过这个消息处理程序将会发送超过 **34** 个消息。你可以留着这个 **BP** 然后运行 **34** 次（这种情况下，在某一时刻你会看到有窗口出现，按钮被绘制等），你也可以删除断点然后就运行一次。这样的话，对 **nag** 窗口的调用就不会再次执行，所以删除断点再运行程序比较好：



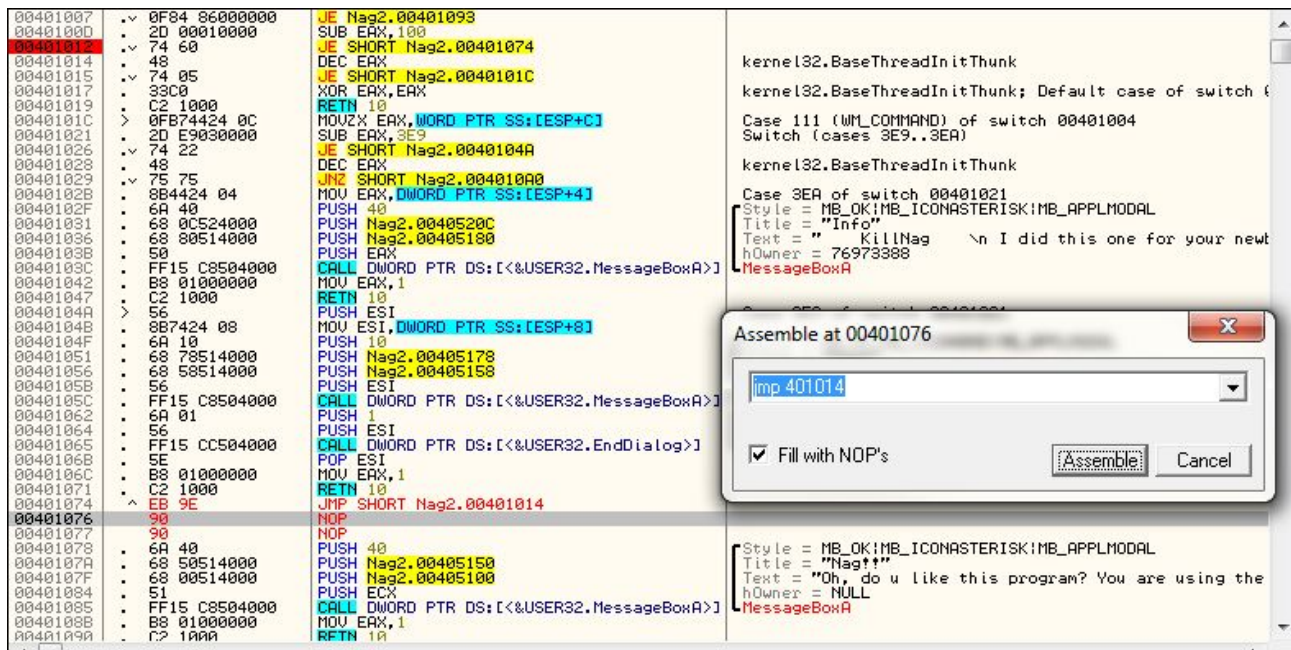
然后就看到了主窗口：



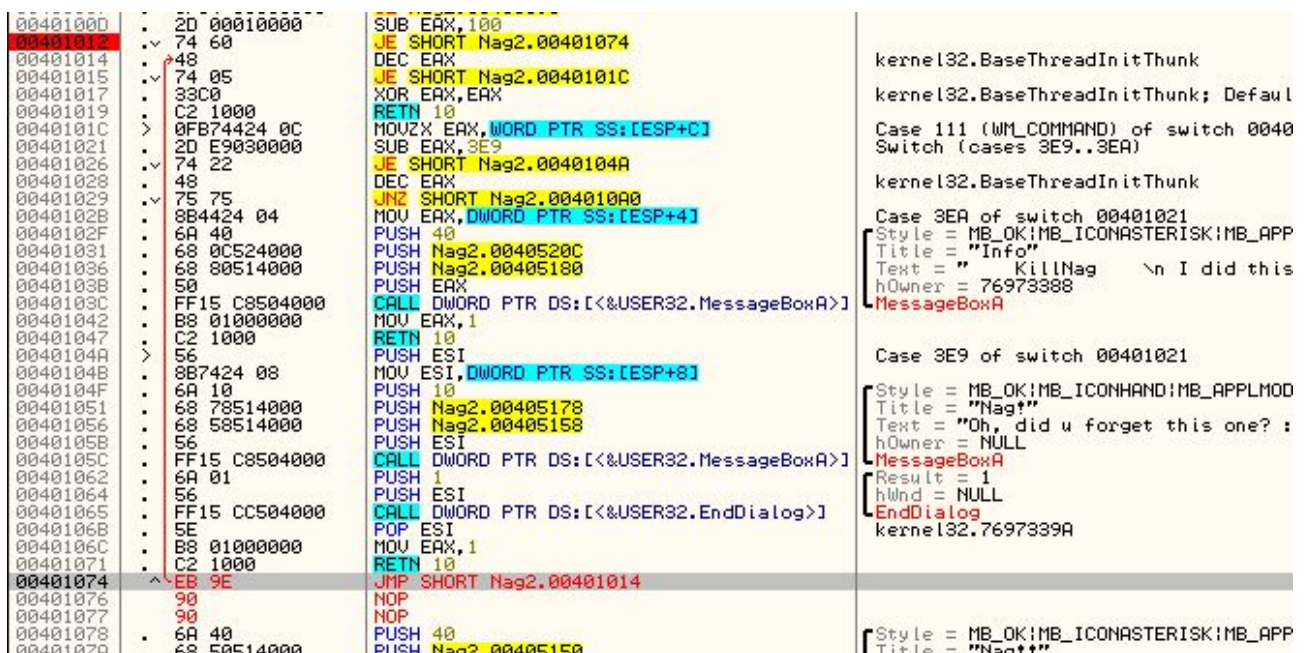
注意初始的 **nag** 窗口已经消失了。

四、给程序打补丁

通常我们所要做的是给跳转到 **nag** 的 **JE** 指令打补丁，将其 **NOP** 掉。这样的话，它就是再也不会跳了，不过我想告诉你另一个种方法也可以实现。我们知道当正确的消息来到消息处理程序时（这里指的是第二个消息），我们的 **nag** 代码将会被调用。好吧，如果我们允许跳到 **nag** 那，但是将 **nag** 改成再跳回去会怎么样？



这里，将会跳转到 401074 的 `nag` 指令，但是我们让它又立即跳回了初始跳转的后面那行 (401014)。基本上，我们的程序会跳转，然后又跳回到下一行：

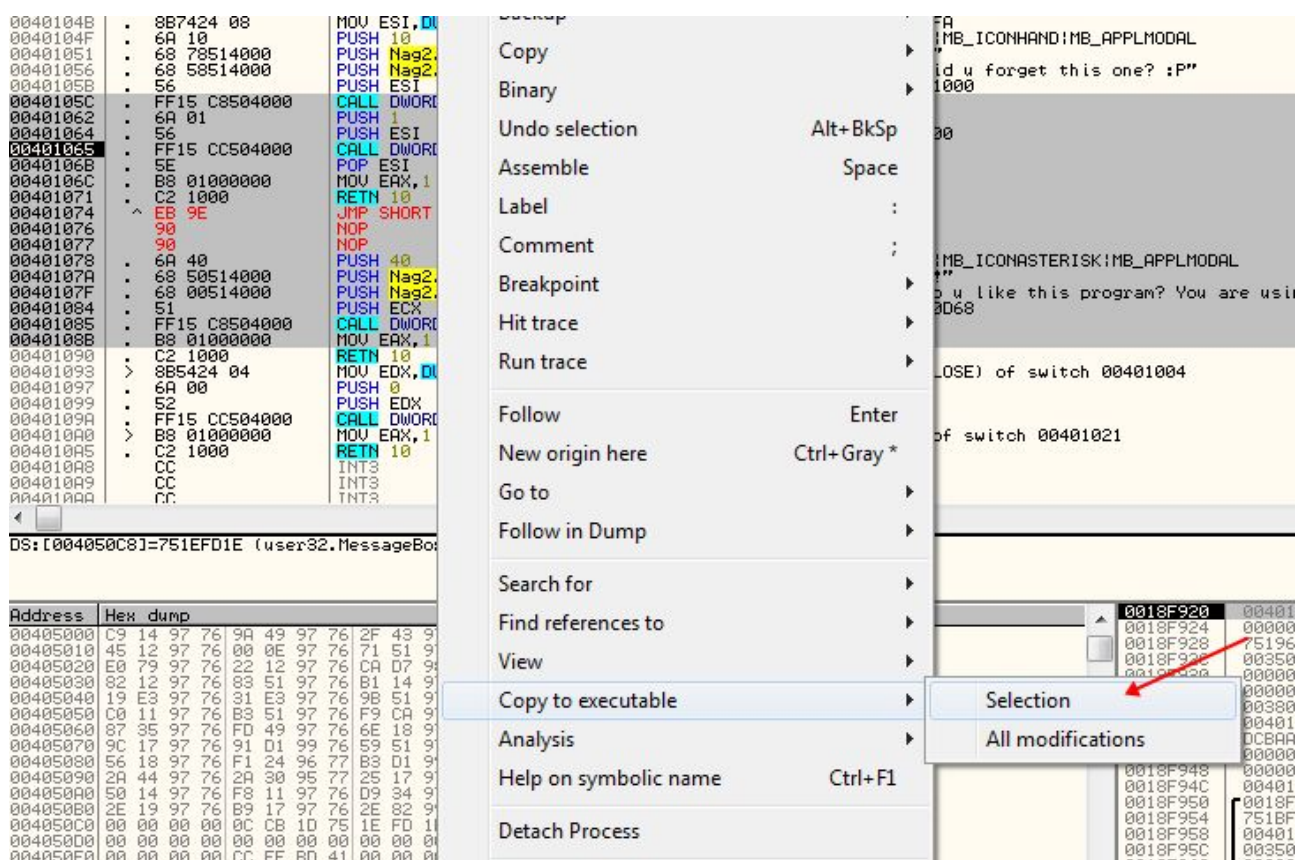


将 401012 处的 `JE` 指令 `NOP` 掉与添加一个跳回到 401074 的跳转真的没啥不同，不过我想让你开始注意到总是有多个打补丁的方法，有时候 `NOP` 掉一个 `CALL` 并不是最好的方法。记住，这个二进制文件是你的，你可以添加任何你想添加的代码，所以别害怕修改它，尤其是在学习的时候。

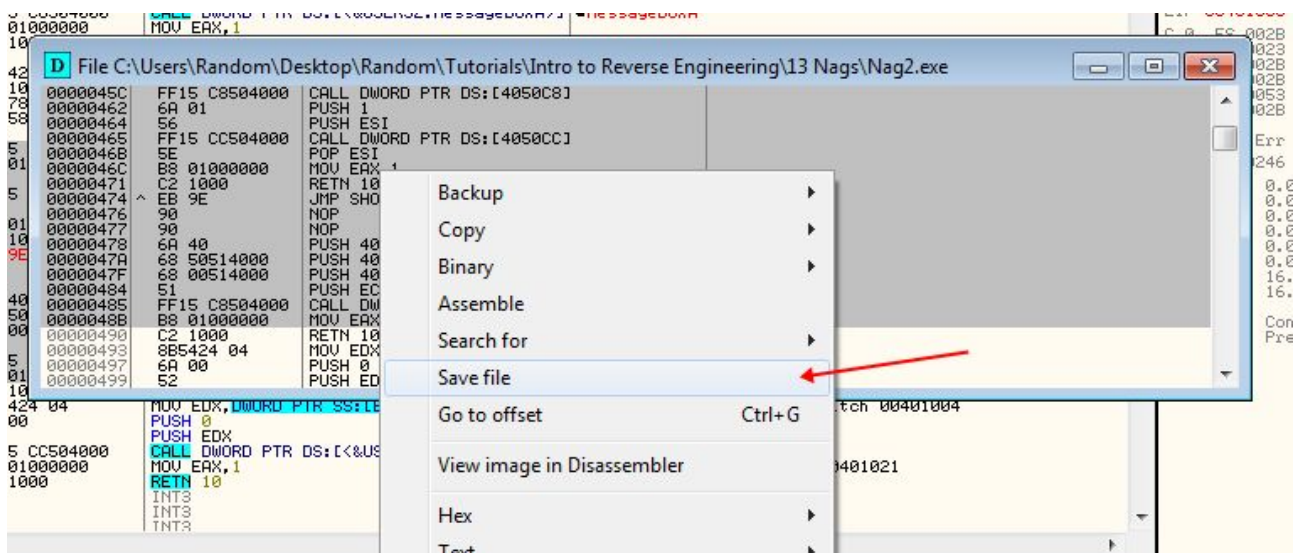
运行程序，可以看到 `nag` 窗口同样被绕过了：



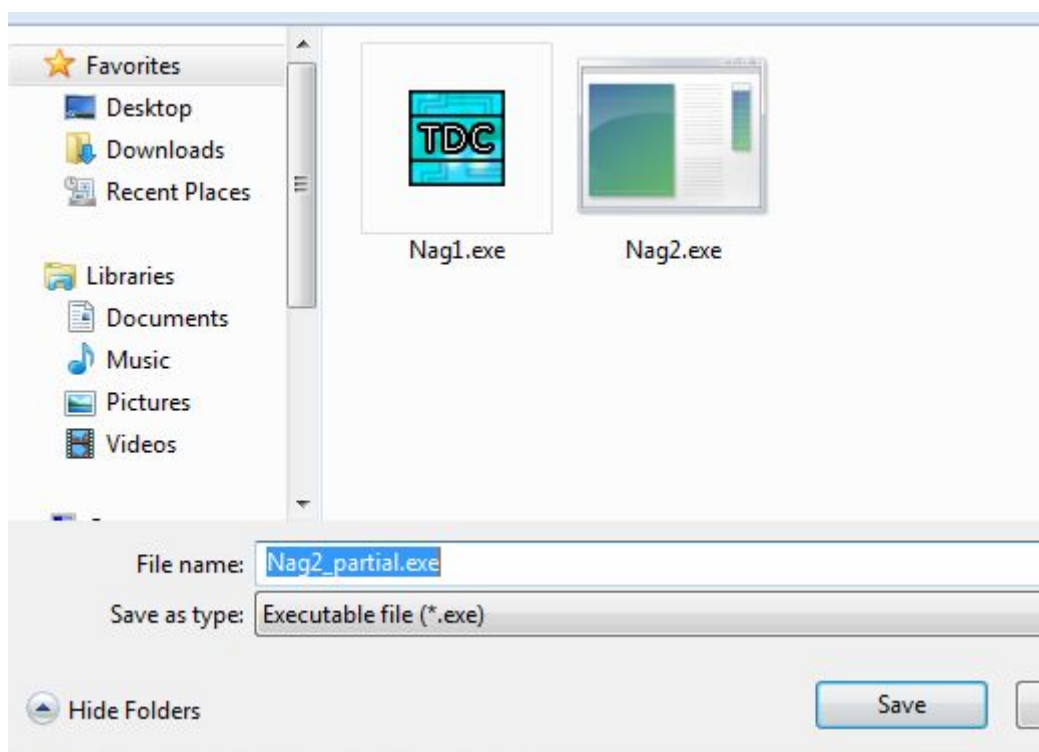
现在保存补丁。选中修改的代码（如果你选多了也没事。译者注：就是说你选中了那些没有修改的也没关系。），选择“Copy to executable” -> “Selection”：



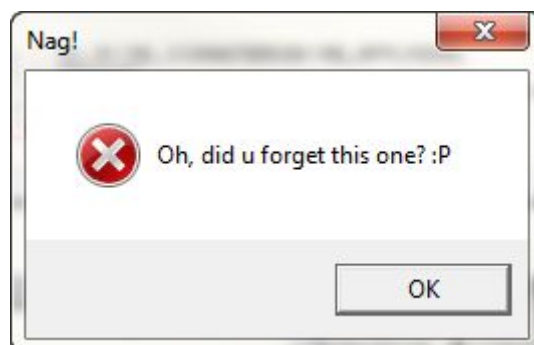
在新窗口上右键，选择“Save file”：



我将它保存为不同名字的文件，这里是 Nag2-partial.exe。等会你会明白我为什么把它取名为 partial:



OK。咱们继续，Olly 载入这个打过补丁的程序试试看。我们直接跳到了主窗口，所以我们知道那个补丁起作用了。现在点击 **exit**，但是弹出了这个：




```
00401017 . 33C0 XOR EAX,EAX
00401019 . C2 1000 RETN 10
0040101C > 0FB74424 0C MOVZX EAX,WORD PTR SS:[ESP+C]
00401021 . 2D E9030000 SUB EAX,3E9
00401026 . 74 22 JE SHORT Nag2.0040104A
00401028 . 48 DEC EAX
00401029 . 75 75 JNZ SHORT Nag2.004010A0
0040102B . 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4]
0040102F . 6A 40 PUSH 40
00401031 . 68 0C524000 PUSH Nag2.0040520C
00401036 . 68 80514000 PUSH Nag2.00405180
0040103B . 50 PUSH EAX
0040103C . FF15 C8504000 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00401042 . B8 01000000 MOV EAX,1
00401047 . C2 1000 RETN 10
0040104A > 56 PUSH ESI
0040104B . 8B7424 08 MOV ESI,DWORD PTR SS:[ESP+8]
0040104F . 6A 10 PUSH 10
00401051 . 68 78514000 PUSH Nag2.00405178
00401056 . 68 58514000 PUSH Nag2.00405158
0040105B . 56 PUSH ESI
0040105C . FF15 C8504000 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00401062 . 6A 01 PUSH 1
00401064 . 56 PUSH ESI
00401065 . FF15 CC504000 CALL DWORD PTR DS:[<&USER32.EndDialog>]
0040106B . 5E POP ESI
0040106C . B8 01000000 MOV EAX,1
00401071 . C2 1000 RETN 10
00401074 . EB 9E JMP SHORT Nag2.00401014
00401076 . 90 NOP
00401077 . 90 NOP
00401078 . 6A 40 PUSH 40
0040107A . 68 50514000 PUSH Nag2.00405150
```

Can't put a jump back...

...because of this

所以接下来你可能会想“咱们就把 401026 的 JE 指令修改成跳到 EndDialog，跳过显示 nag 的 MessageBoxA 的指令”。这个想法不错，咱们来试试：

```
00401017 . 33C0 XOR EAX,EAX
00401019 . C2 1000 RETN 10
0040101C > 0FB74424 0C MOVZX EAX,WORD PTR SS:[ESP+C]
00401021 . 2D E9030000 SUB EAX,3E9
00401026 . 74 22 JE SHORT Nag2.0040104A
00401028 . 48 DEC EAX
00401029 . 75 75 JNZ SHORT Nag2.004010A0
0040102B . 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4]
0040102F . 6A 40 PUSH 40
00401031 . 68 0C524000 PUSH Nag2.0040520C
00401036 . 68 80514000 PUSH Nag2.00405180
0040103B . 50 PUSH EAX
0040103C . FF15 C8504000 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00401042 . B8 01000000 MOV EAX,1
00401047 . C2 1000 RETN 10
0040104A > 56 PUSH ESI
0040104B . 8B7424 08 MOV ESI,DWORD PTR SS:[ESP+8]
0040104F . 6A 10 PUSH 10
00401051 . 68 78514000 PUSH Nag2.00405178
00401056 . 68 58514000 PUSH Nag2.00405158
0040105B . 56 PUSH ESI
0040105C . FF15 C8504000 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00401062 . 6A 01 PUSH 1
00401064 . 56 PUSH ESI
00401065 . FF15 CC504000 CALL DWORD PTR DS:[<&USER32.EndDialog>]
0040106B . 5E POP ESI
0040106C . B8 01000000 MOV EAX,1
00401071 . C2 1000 RETN 10
00401074 . EB 9E JMP SHORT Nag2.00401014
00401076 . 90 NOP
00401077 . 90 NOP
00401078 . 6A 40 PUSH 40
0040107A . 68 50514000 PUSH Nag2.00405150
```

So make this...

...jump to here instead

将 401026 的 JE 指令修改成跳转到 401062，也就是跳到 EndDialog 的第一行：

```
00401007 . 0F84 8B000000 JE Nag2.00401095
0040100D . 2D 00010000 SUB EAX,100
00401012 . 74 60 JE SHORT Nag2.00401074
00401014 . 48 DEC EAX
00401015 . 74 05 JE SHORT Nag2.0040101C
00401017 . 33C0 XOR EAX,EAX
00401019 . C2 1000 RETN 10
0040101C > 0FB74424 0C MOVZX EAX,WORD PTR SS:[ESP+C]
00401021 . 2D E9030000 SUB EAX,3E9
00401026 . 74 3A JE SHORT Nag2.00401062
00401028 . 48 DEC EAX
00401029 . 75 75 JNZ SHORT Nag2.004010A0
0040102B . 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4]
0040102F . 6A 40 PUSH 40
00401031 . 68 0C524000 PUSH Nag2.0040520C
00401036 . 68 80514000 PUSH Nag2.00405180
0040103B . 50 PUSH EAX
0040103C . FF15 C8504000 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00401042 . B8 01000000 MOV EAX,1
00401047 . C2 1000 RETN 10
0040104A > 56 PUSH ESI
0040104B . 8B7424 08 MOV ESI,DWORD PTR SS:[ESP+8]
0040104F . 6A 10 PUSH 10
00401051 . 68 78514000 PUSH Nag2.00405178
00401056 . 68 58514000 PUSH Nag2.00405158
0040105B . 56 PUSH ESI
0040105C . FF15 C8504000 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00401062 . 6A 01 PUSH 1
00401064 . 56 PUSH ESI
00401065 . FF15 CC504000 CALL DWORD PTR DS:[<&USER32.EndDialog>]
0040106B . 5E POP ESI
0040106C . B8 01000000 MOV EAX,1
00401071 . C2 1000 RETN 10
00401074 . EB 9E JMP SHORT Nag2.00401014
00401076 . 90 NOP
```

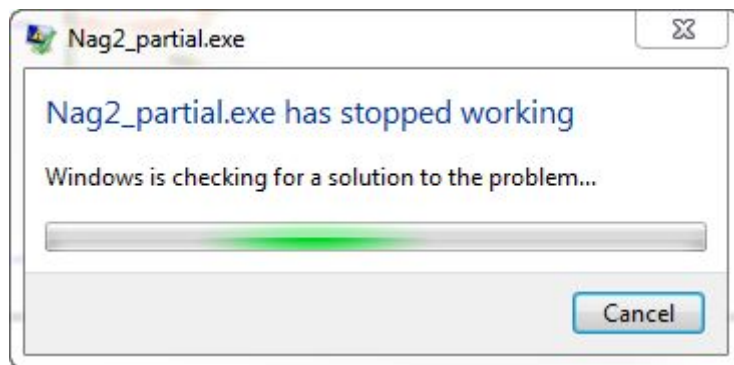
Assemble at 00401028

JE SHORT 00401062

☒ Fill with NOP's

Assemble Cancel

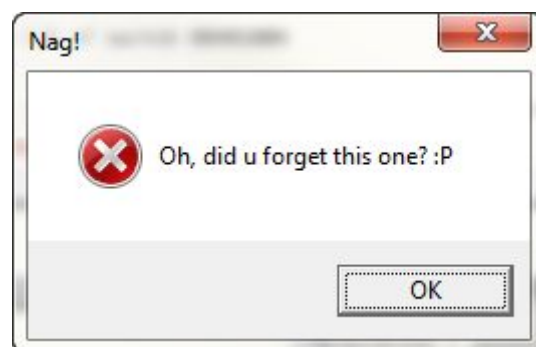
运行程序：



看起来前途灰暗呐。我们明显做错了什么。下面是我们准备做的：运行没有补丁的程序，单步执行它，看看是什么情况。然后再运行带补丁的程序，再看看两者之间有什么不同。重启应用并点击“Exit”，我们会断在打补丁的地方（因为我们重启了应用，所以补丁消失了）：

00401017	. 33C0	XOR EAX,EAX	
00401019	. C2 1000	RETN 10	
0040101C	> 0FB74424 0C	MOVZX EAX,WORD PTR SS:[ESP+C]	
00401021	. 2D E9030000	SUB EAX,3E9	
00401026	74 22	JE SHORT Nag2_par.0040104A	Switch (cases 3E9..3EA)
00401028	. 48	DEC EAX	
00401029	. 75 75	JNZ SHORT Nag2_par.004010A0	Case 3EA of switch 00401021
0040102B	. 8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL
0040102F	. 6A 40	PUSH 40	Title = "Info"
00401031	. 68 0C524000	PUSH Nag2_par.0040520C	Text = "KillNag \n I did this one for your ne
00401036	. 68 80514000	PUSH Nag2_par.00405180	hOwner = NULL
0040103B	. 50	PUSH EAX	MessageBoxA
0040103C	. FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	
00401042	. B8 01000000	MOV EAX,1	Nag2_par.00401000; Case 3E9 of switch 00401021
00401047	. C2 1000	RETN 10	
0040104A	> 56	PUSH ESI	
0040104B	. 8B7424 08	MOV ESI,DWORD PTR SS:[ESP+8]	
0040104F	. 6A 10	PUSH 10	Style = MB_OK MB_ICONHAND MB_APPLMODAL
00401051	. 68 78514000	PUSH Nag2_par.00405178	Title = "Nag!"
00401056	. 68 58514000	PUSH Nag2_par.00405158	Text = "Oh, did u forget this one? :P"
0040105B	. 56	PUSH ESI	hOwner = 00401000
0040105C	. FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
00401062	. 6A 01	PUSH 1	Result = 1
00401064	. 56	PUSH ESI	hWnd = 00401000
00401065	. FF15 CC504000	CALL DWORD PTR DS:[&USER32.EndDialog]	EndDialog
0040106B	. 5E	POP ESI	user32.751962FA
0040106C	. B8 01000000	MOV EAX,1	

单步运行几行，当你单步步过对 MessageBoxA 的调用时，你就会看到 nag 窗口：



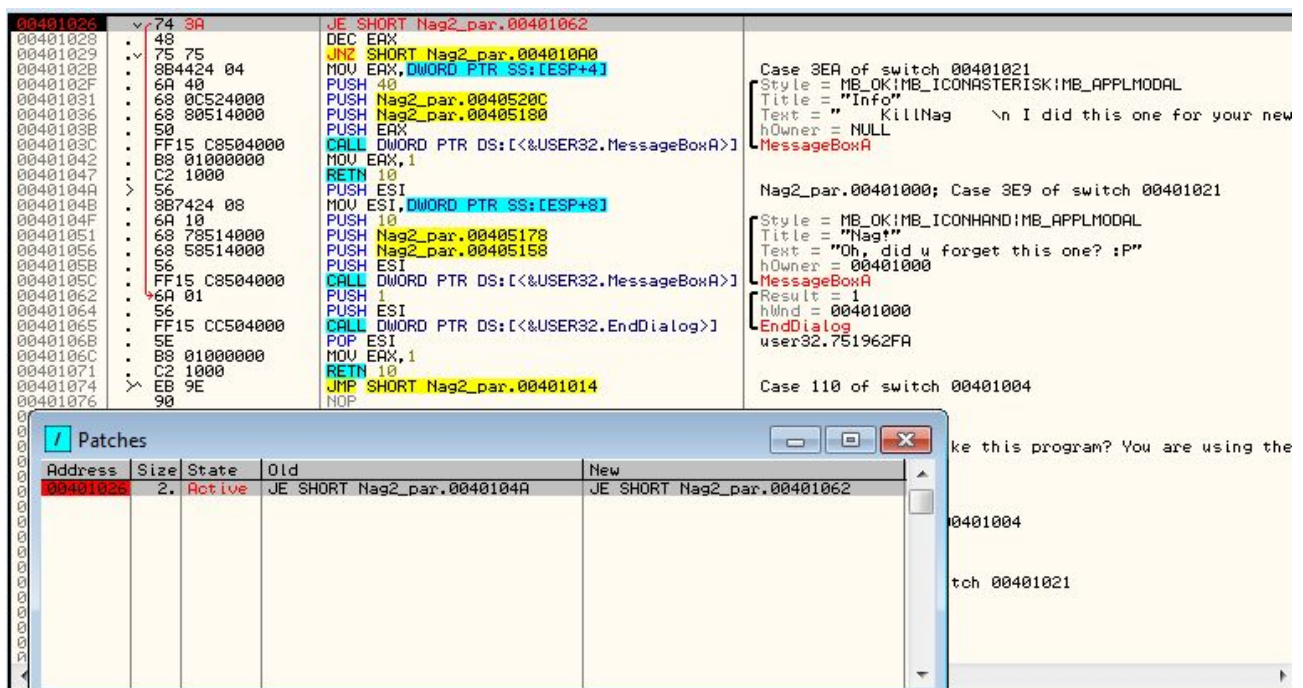
再单步两次，直到对 EndDialog 调用的那个 CALL：

0040104A	> 56	PUSH ESI	Case 3E9 of switch 00401021
0040104B	. 8B7424 08	MOV ESI,DWORD PTR SS:[ESP+8]	Nag2_par.00401000
0040104F	. 6A 10	PUSH 10	Style = MB_OK MB_ICONHAND MB_APPLMODAL
00401051	. 68 78514000	PUSH Nag2_par.00405178	Title = "Nag!"
00401056	. 68 58514000	PUSH Nag2_par.00405158	Text = "Oh, did u forget this one? :P"
0040105B	. 56	PUSH ESI	hOwner = 00111166 ('KillNag - KiTo',class='#32770')
0040105C	. FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
00401062	. 6A 01	PUSH 1	Result = 1
00401064	. 56	PUSH ESI	hWnd = 00111166 ('KillNag - KiTo',class='#32770')
00401065	. FF15 CC504000	CALL DWORD PTR DS:[&USER32.EndDialog]	EndDialog
0040106B	. 5E	POP ESI	
0040106C	. B8 01000000	MOV EAX,1	
00401071	. C2 1000	RETN 10	Case 110 of switch 00401004
00401074	> EB 9E	JMP SHORT Nag2_par.00401014	
00401076	. 90	NOP	

咱们来看看堆栈。在堆栈中看到四个项目：我们窗口的句柄、对话框的返回值、指向第一行代码（401000）的指针、返回到 user32 的地址。

```
0018F920 00111166 hWnd = 00111166 ('KillNag - KiTo',class='#32770')
0018F924 00000001 Result = 1
0018F928 00401000 Nag2_par.00401000
0018F92C 751962FA RETURN to user32.751962FA
0018F930 00111166
0018F934 00000111
0018F938 000003E9
0018F93C 000B1124
0018F940 00401000 Nag2_par.00401000
0018F944 DCBAABCD
0018F948 00000001
0018F94C 00000000
0018F950 00401000 Nag2_par.00401000
0018F954 0018F9CC
0018F958 751BF943 RETURN to user32.751BF943 from user32.751962D7
0018F95C 00401000 Nag2_par.00401000
0018F960 00111166
0018F964 00000111
0018F968 000003E9
0018F96C 000B1124
0018F970 0E132BDD
0018F974 00000000
0018F978 00000111
```

重启应用，当我们来到打补丁的地方后将其激活（打开补丁窗口，选中后按一下空格键）：



现在我们将跳过 nag 消息框。单步执行直到来到对 EndDialog 的调用处：

```
0040105C . FF15 C8504000 CALL DWORD PTR DS:[&USER32.MessageBoxA]
00401062 . 6A 01 PUSH 1
00401064 . 56 PUSH ESI
00401065 . FF15 CC504000 CALL DWORD PTR DS:[&USER32.EndDialog]
0040106B . 5E POP ESI
0040106C . B8 01000000 MOV EAX,1
00401071 . C2 1000 RETN 10
00401074 . EB 9E JMP SHORT Nag2_par.00401014
```

看看堆栈，有窗口的句柄、返回值、返回到 user32，但是没有了指向代码的第一行也就是 401000 的指针!!!

```

0018F92E 00401000 hWnd = 00401000
0018F92F 00000001 Result = 1
0018F930 751962FA RETURN to user32.751962FA
0018F931 000E11AA
0018F932 00000111
0018F933 000003E9
0018F934 000B1148
0018F935 00401000 Nag2_par.00401000
0018F936 DCBAABCD
0018F937 00000001
0018F938 00000000
0018F939 00401000 Nag2_par.00401000
0018F93A 0018F9CC
0018F93B 751BF943 RETURN to user32.751BF943 from user32.751962D7
0018F93C 00401000 Nag2_par.00401000
0018F93D 000E11AA
0018F93E 00000111
0018F93F 000003E9
0018F940 000B1148
0018F941 58F6A79B
0018F942 00000000
0018F943 00000111
0018F944 00A377C0

```

如果你向上滚动看看第二个 nag 的 CALL，你会发现消息框被创建之前，ESI 被压栈了。这是一个指向我们代码的指针。在调用消息框前，它刚好被压入堆栈，虽然它也可以在这之后被压栈。所以我们丢失了一个重要的 push 操作，为了正确的运行 EndDialog 程序需要它。问题是，我们有一些想要的初始化代码，然后是一个我们不想要的对 nag 窗口的 CALL，之后又是一个我们想要的对 EndDialog 调用的 CALL：

0040101C	> 0FB74424 0C	MOVZX EAX,WORD PTR SS:[ESP+C]	Switch (cases 3E9..3EA)
00401021	. 2D E9030000	SUB EAX,3E9	
00401022	> 74 3A	JE SHORT Nag2_par.00401062	
00401023	. 48	DEC EAX	
00401024	> 75 75	JNZ SHORT Nag2_par.004010A0	
00401025	. 8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]	Case 3EA of switch 00401021
00401026	. 6A 40	PUSH 40	Style = MB_OK;MB_ICONASTERISK;MB_APPLMODAL
00401027	. 68 0C524000	PUSH Nag2_par.0040520C	Title = "Info"
00401028	. 68 80514000	PUSH Nag2_par.00405180	Text = "KillNag \n I did this one for your new"
00401029	. 50	PUSH EAX	hOwner = NULL
0040102A	. FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
0040102B	. B8 01000000	MOV EAX,1	
0040102C	> C2 1000	RETN 10	
0040102D	. 8B7424 08	MOV ESI,DWORD PTR SS:[ESP+8]	Nag2_par.00401000; Case 3E9 of switch 00401021
0040102E	. 6A 10	PUSH 10	user32.751962FA
0040102F	. 68 78514000	PUSH Nag2_par.00405178	Style = MB_OK;MB_ICONHAND;MB_APPLMODAL
00401030	. 68 58514000	PUSH Nag2_par.00405158	Title = "Nag!"
00401031	. 56	PUSH ESI	Text = "Oh, did u forget this one? :P"
00401032	. FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	hOwner = 00401000
00401033	. 6A 01	PUSH 1	MessageBoxA
00401034	. 56	PUSH ESI	Result = 1
00401035	. FF15 CC504000	CALL DWORD PTR DS:[&USER32.EndDialog]	hWnd = 00401000
00401036	. 5E	POP ESI	EndDialog
00401037	. B8 01000000	MOV EAX,1	Nag2_par.00401000
00401038	> C2 1000	RETN 10	
00401039	. EB 9E	JMP SHORT Nag2_par.00401014	Case 110 of switch 00401004

好吧，那咱们就去除掉不想要的代码。选中 MessageBoxA 指令(从 40104F 到 40105C) 然后右键，选择 “Binary” -> “fill with NOPS”：

0040102B	. 8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]	Case 3EA of switch 00401021
0040102C	. 6A 40	PUSH 40	Style = MB_OK;MB_ICONASTERISK;MB_APPLMODAL
0040102D	. 68 0C524000	PUSH Nag2_par.0040520C	Title = "Info"
0040102E	. 68 80514000	PUSH Nag2_par.00405180	Text = "KillNag \n I did this one for your new"
0040102F	. 50	PUSH EAX	hOwner = NULL
00401030	. FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
00401031	. B8 01000000	MOV EAX,1	
00401032	> C2 1000	RETN 10	
00401033	. 8B7424 08	MOV ESI,DWORD PTR SS:[ESP+8]	Nag2_par.00401000; Case 3E9 of switch 00401021
00401034	. 6A 10	PUSH 10	
00401035	. 68 78514000	PUSH Nag2_par.00405178	
00401036	. 68 58514000	PUSH Nag2_par.00405158	
00401037	. 56	PUSH ESI	
00401038	. FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	
00401039	. 6A 01	PUSH 1	
0040103A	. 56	PUSH ESI	
0040103B	. FF15 CC504000	CALL DWORD PTR DS:[&USER32.EndDialog]	
0040103C	. 5E	POP ESI	
0040103D	. B8 01000000	MOV EAX,1	
0040103E	> C2 1000	RETN 10	
0040103F	. EB 9E	JMP SHORT Nag2_par.00401014	
00401040	. 90	NOP	
00401041	. 90	NOP	
00401042	. 6A 40 68 50 51 40	ASCII "jehP00",0	
00401043	. 68 80514000	PUSH Nag2_par.00405180	
00401044	. 51	PUSH EAX	
00401045	. FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	
00401046	. B8 01000000	MOV EAX,1	
00401047	> C2 1000	RETN 10	
00401048	. 8B5424 04	MOV EDI,DWORD PTR SS:[ESP+4]	
00401049	. 6A 00	PUSH 0	
0040104A	. 52	PUSH EDI	

然后，砰！再也没有对 nag 的 CALL 了：

00401021	20 E9030000	SUB EAX,3E9	Switch (cases 3E9..3EH)
00401026	74 22	JE SHORT Nag2_par.0040104A	
00401028	48	DEC EAX	
00401029	75 75	JNZ SHORT Nag2_par.004010A0	
0040102B	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]	Case 3EA of switch 00401021
0040102F	6A 40	PUSH 40	Style = MB_OK!MB_ICONASTERISK!MB_APPLMODAL
00401031	68 0C524000	PUSH Nag2_par.0040520C	Title = "Info"
00401036	68 80514000	PUSH Nag2_par.00405180	Text = "\n I did this one for your new
0040103B	50	PUSH EAX	hOwner = NULL
0040103C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
00401042	B8 01000000	MOV EAX,1	
00401047	C2 1000	RETN 10	
0040104A	56	PUSH ESI	
0040104B	8B7424 08	MOV ESI,DWORD PTR SS:[ESP+8]	Nag2_par.00401000; Case 3E9 of switch 00401021
0040104F	90	NOP	Style
00401050	90	NOP	Title
00401051	90	NOP	
00401052	90	NOP	
00401053	90	NOP	
00401054	90	NOP	
00401055	90	NOP	Text
00401056	90	NOP	
00401057	90	NOP	
00401058	90	NOP	
00401059	90	NOP	
0040105A	90	NOP	hOwner
0040105B	90	NOP	MessageBoxA
0040105C	90	NOP	
0040105D	90	NOP	
0040105E	90	NOP	
0040105F	90	NOP	
00401060	90	NOP	
00401061	90	NOP	
00401062	6A 01	PUSH 1	Result = 1
00401064	56	PUSH ESI	hwnd = 00401000
00401065	FF15 CC504000	CALL DWORD PTR DS:[&USER32.EndDialog]	EndDialog
0040106B	5E	POP ESI	user32.751962FA
0040106C	B8 01000000	MOV EAX,1	

现在当你在运行程序的时候，会发现程序可以正常关闭了。你可以保存补丁，并且没有 nag 窗口了😁。