第十四章：NAG 窗口（我不是在说你妈）[注①]

一、简介

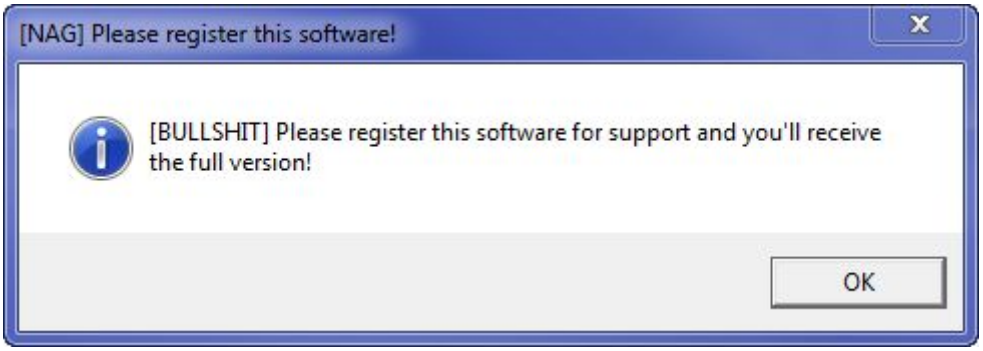
Nags，或者叫 Nag 窗口，是普通的消息对话框。它弹出来是提醒你你的试用结束了、你需要注册、关于访问网站的提醒....。基本上任何事它都要唠叨，而且还是不必要的（像大多数的 boss 一样）。许多免费软件之所以免费，是因为它们充满了 nag（广告、限时试用、重定向）。商业软件通常也有这些玩意儿，提醒你“你只剩下 18 天来使用此产品”等等。在逆向工程领域，除掉 nag 窗口是一个中心主题，有时候也提出了很多挑战。本章我们将会研究两个有 nag 的程序。我们将会绕过它们，之后它们就再也不会显示了，然后再打上补丁，这样它们就永远不会回来了。

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。

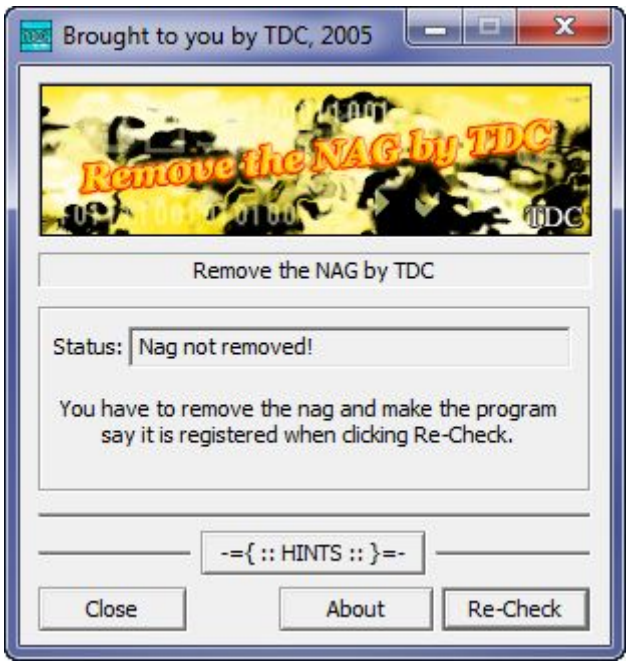
二、第一个应用程序

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:



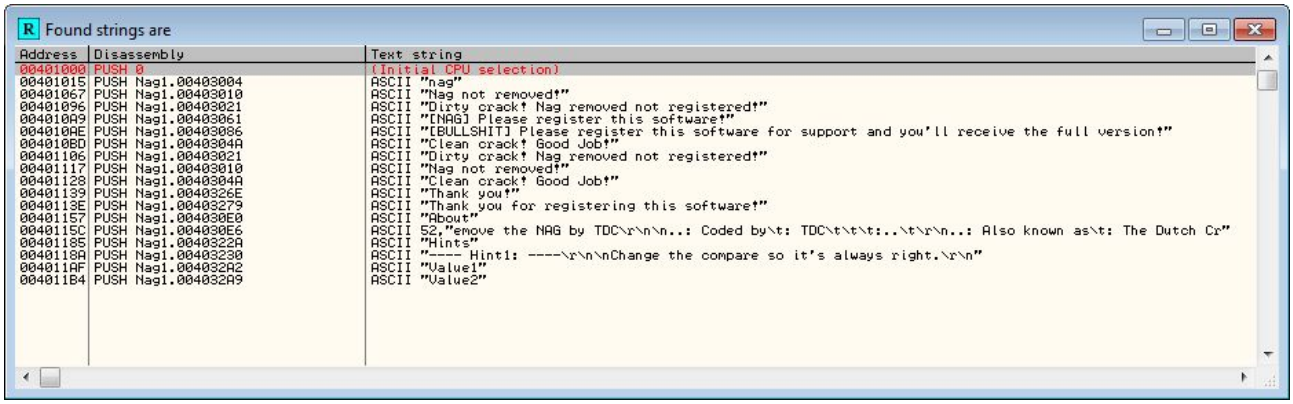
它明摆着告诉你这就是一个 cracker 写的。不管怎么样，点了 OK 就可以看到主窗口：

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）:

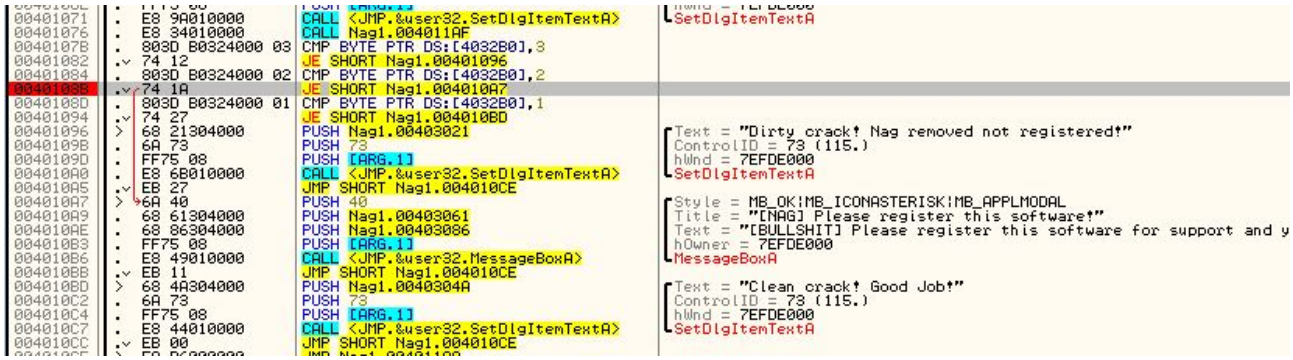
Gee，谢谢。Olly 载入应用，咱们试试老方法——搜索字符串:



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：

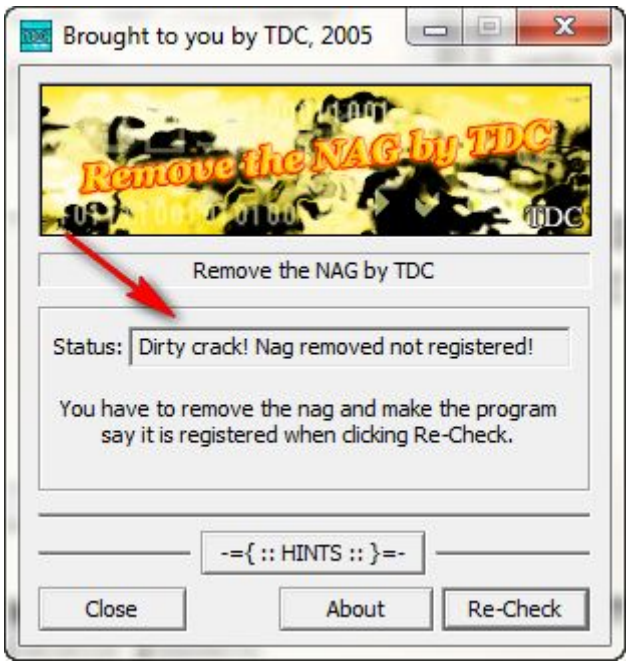
嗯，它上面有一个有趣的字符串，不过咱们现在先不管。咱们看看 4010A7 处，也就是调用 MessageBoxA 函数的第一行，看看哪里调用了它：

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：



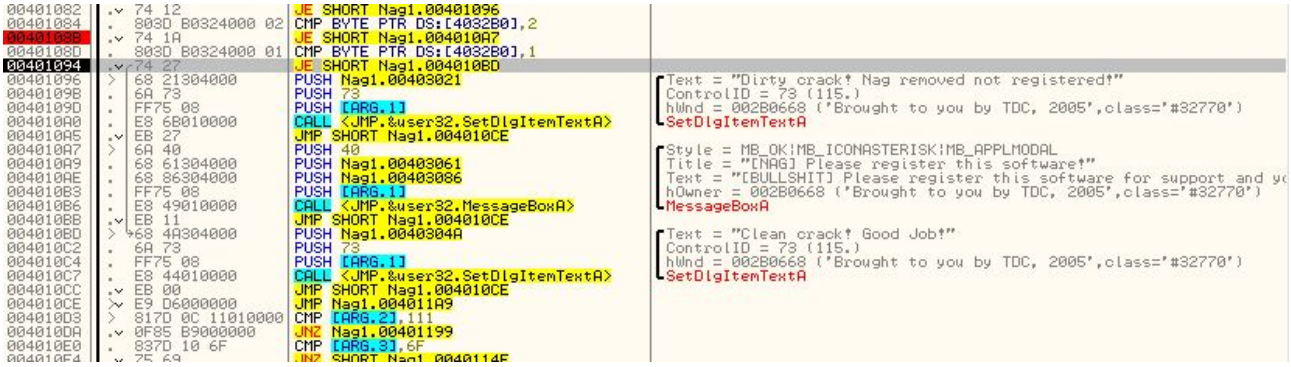
运行程序。然后我们断在了那个 BP，能够看到我们将跳转到 nag 窗口那，所以不能让它跳：

接着运行程序：

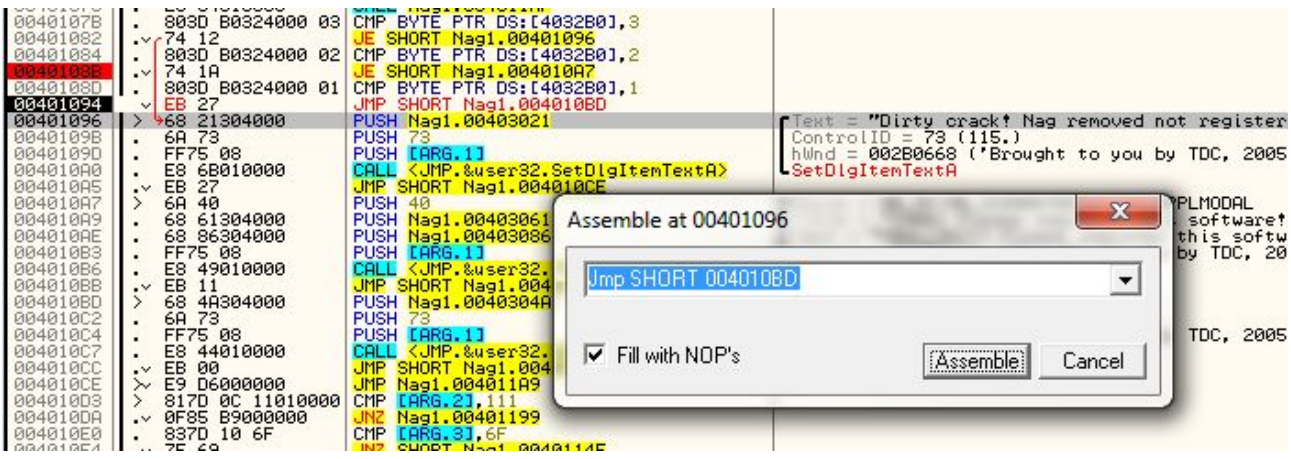


这就是“Dirty crack!”的出处，显然咱们的补丁打的还不够。重启应用，Olly 断在了 BP 那。再次将 0 标志位清零：

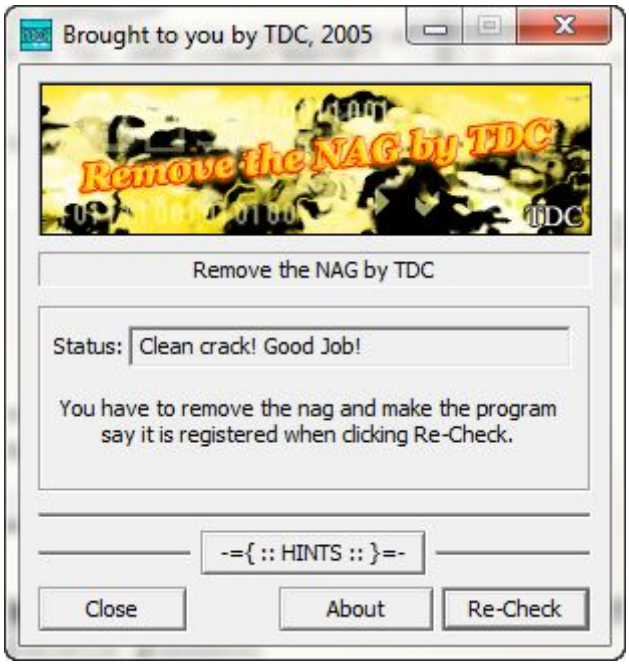
咱们单步执行两次到下一个跳转那。你可能已经猜出来了，这个跳转应该是跳到好消息那的，而不是到坏消息那：



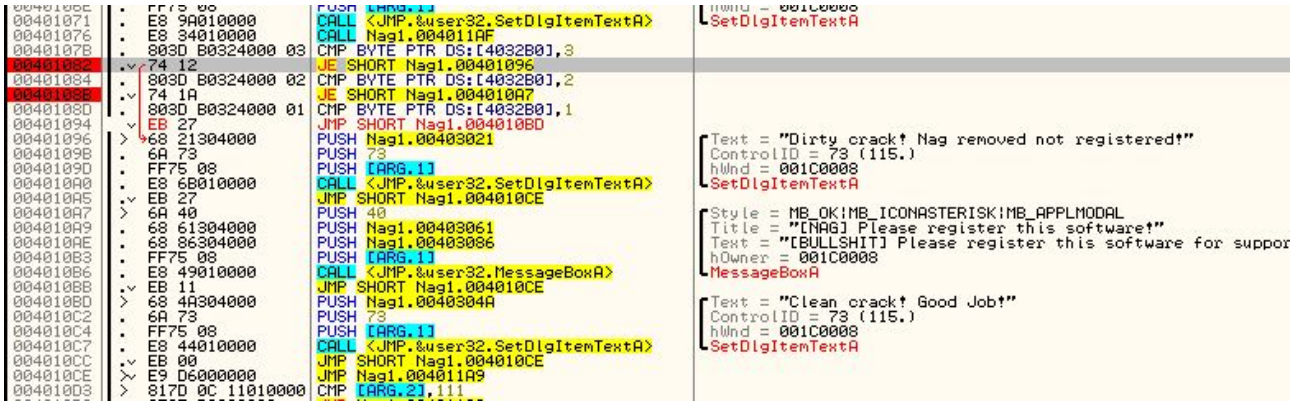
咱们打个补丁，让它直接跳：



运行程序，可以看到咱们弄对了：



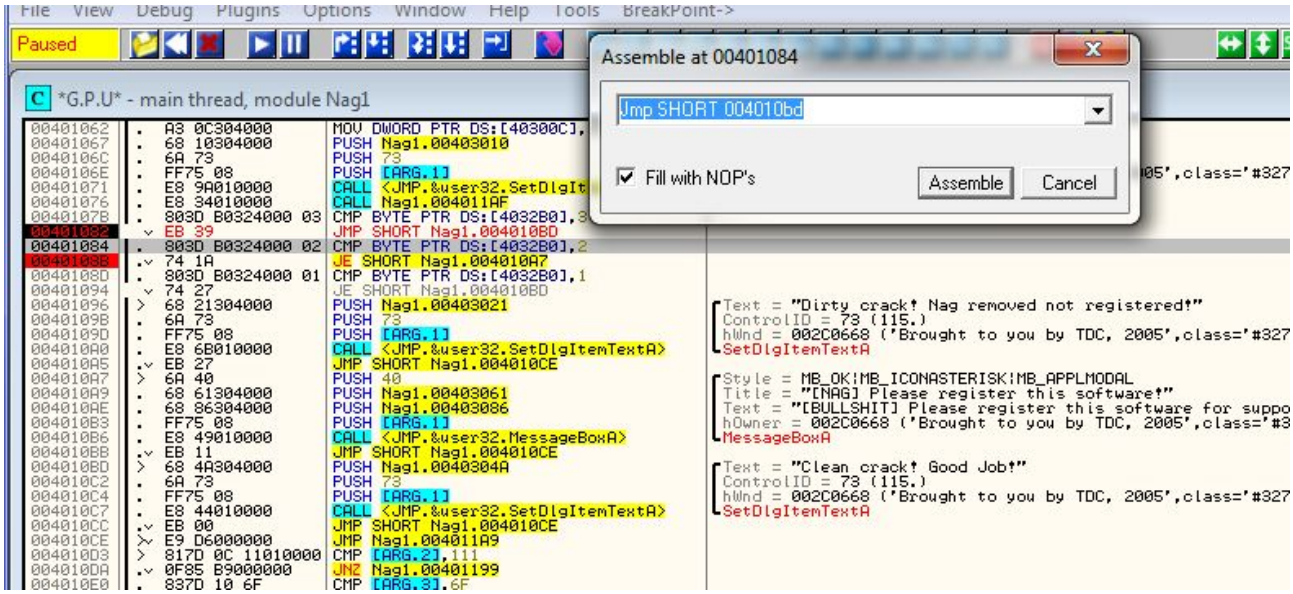
很明显，现在这个补丁就可以解决这个程序，咱们回到 40108B 那（咱们原来将 0 标志位清零的地方）给它打上补丁让它永远不会跳转。保存这两个补丁程序就会很好的运行。不过我也想向你展示（我以前提到过，如果我没有提到的话，那我应该提到的），通常总是有别的方法来给程序打补丁。重启应用，在我们的 BP 处断下来：



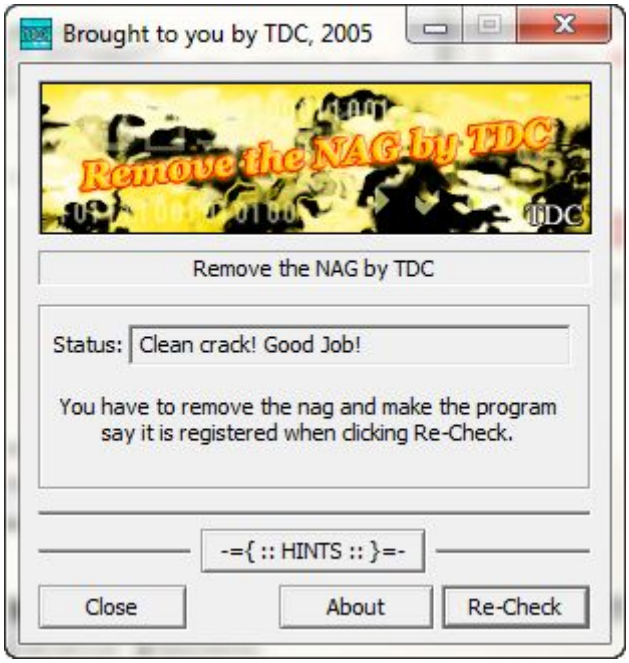
这些指令与下面这些（用高级语言表示）类似：

```
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：

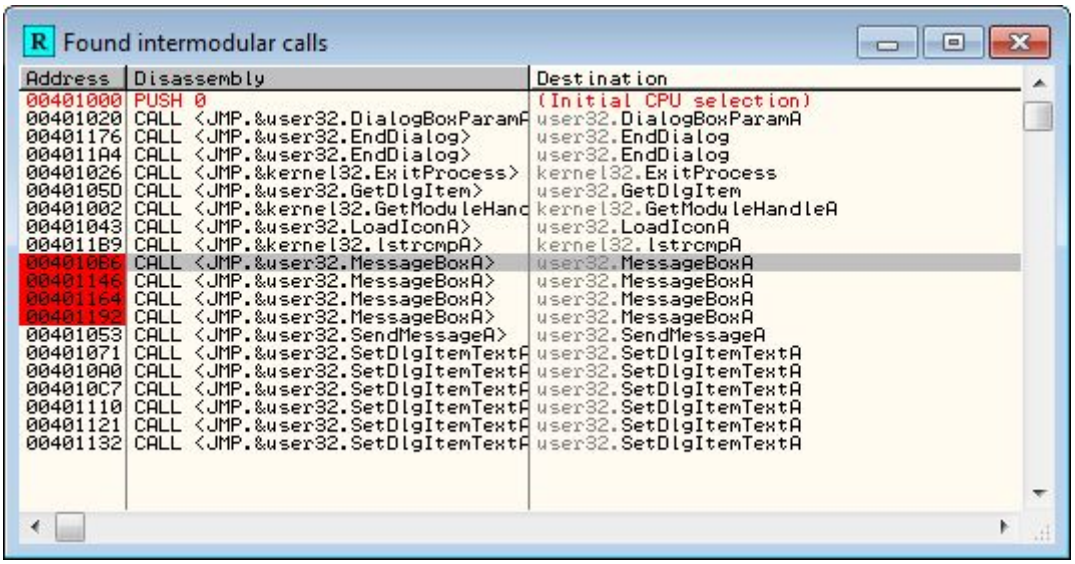


运行下程序：



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）:

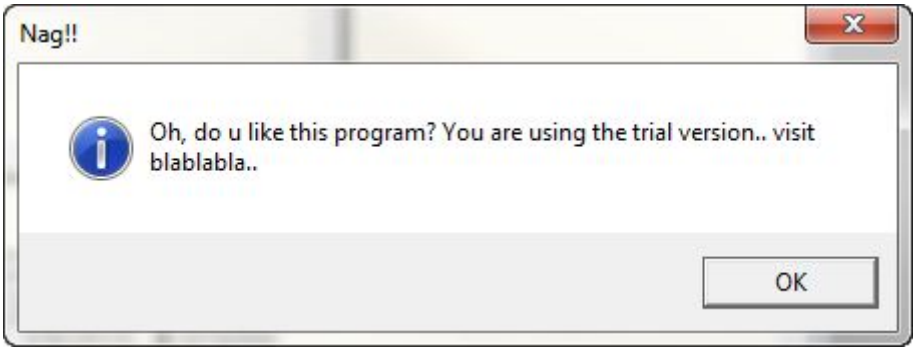


注意有四个对 MessageBoxA 的调用。右键其中一个，选择 “Place a breakpoint on every call to MessageBoxA”。当你运行程序时，在显示任何东西之前，我们会停在下面这行代码:

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。

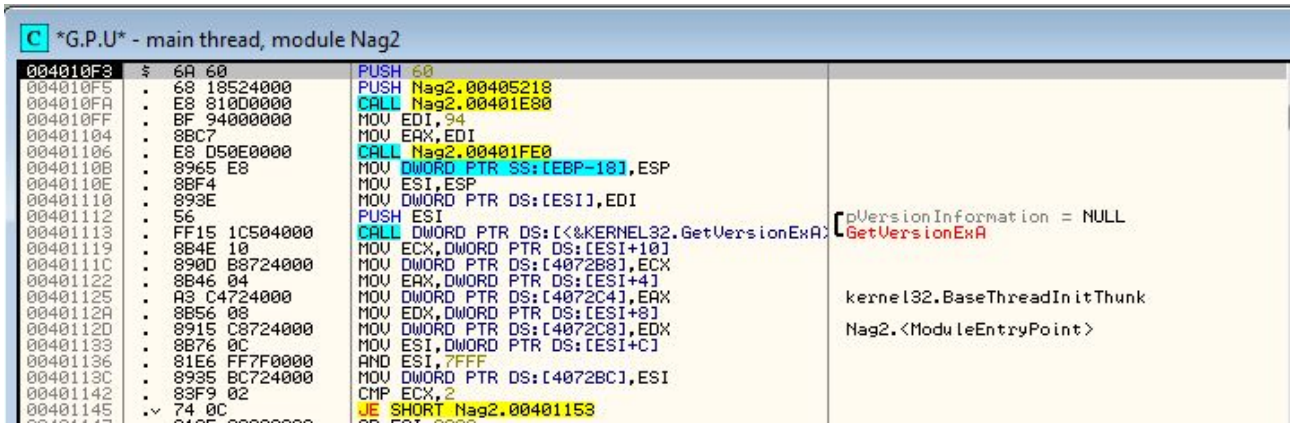
三、第二个应用程序

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:



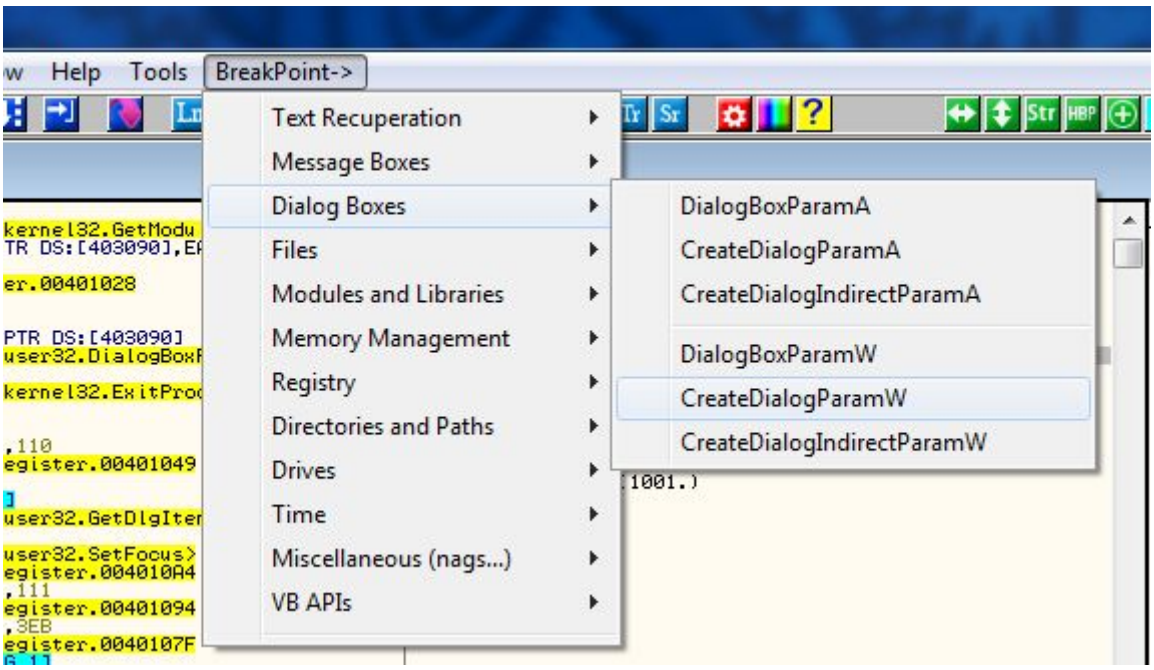
在点了 OK 后，我们看到了主窗口:

此时我关了程序，将其载入到 Olly 中：



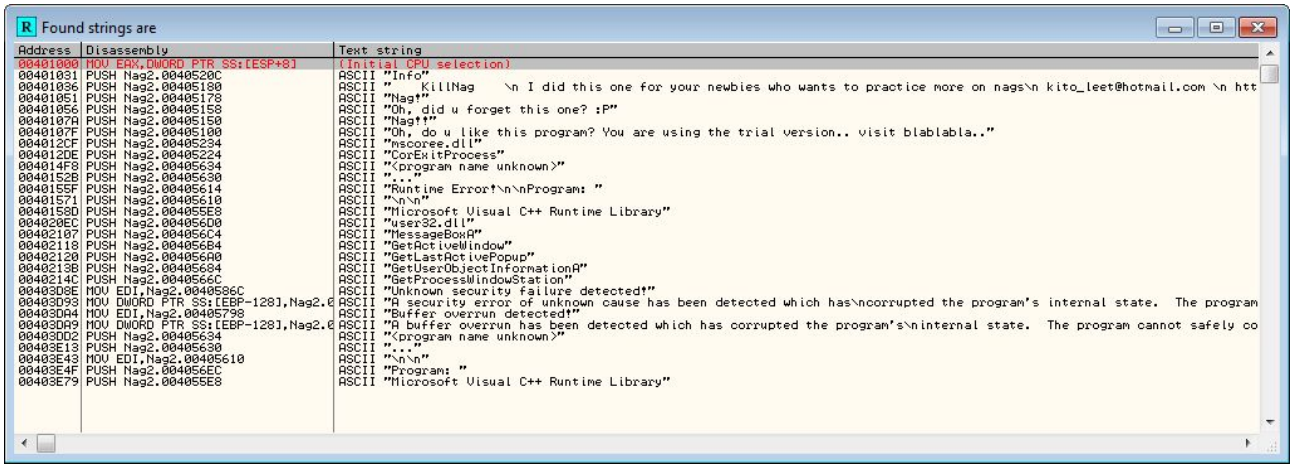
首先，咱们来看看有没有字符串。这里我想提的一件事是 IDAFicator 插件。在众多的添加功能中，它在程序顶部提供了一组按钮，让搜索字符串变得更加的简单。当点击字符串按钮（Str）时，它会显示 ASCII 和 Unicode 两种字符串，并自动的将光标移到顶部，这样你就不用自己滚动到顶部了。下面是那些按钮的样子：

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：

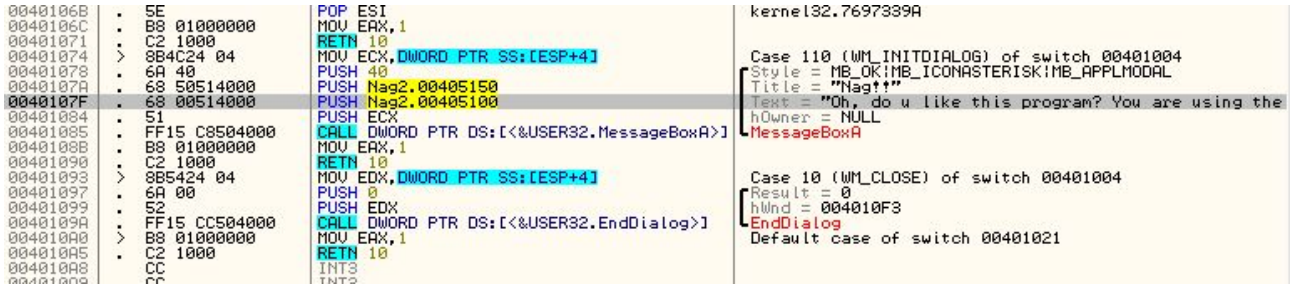


最后，有一个上下文菜单可以让你恢复隐藏的菜单，我们会在后面的章节中讨论。

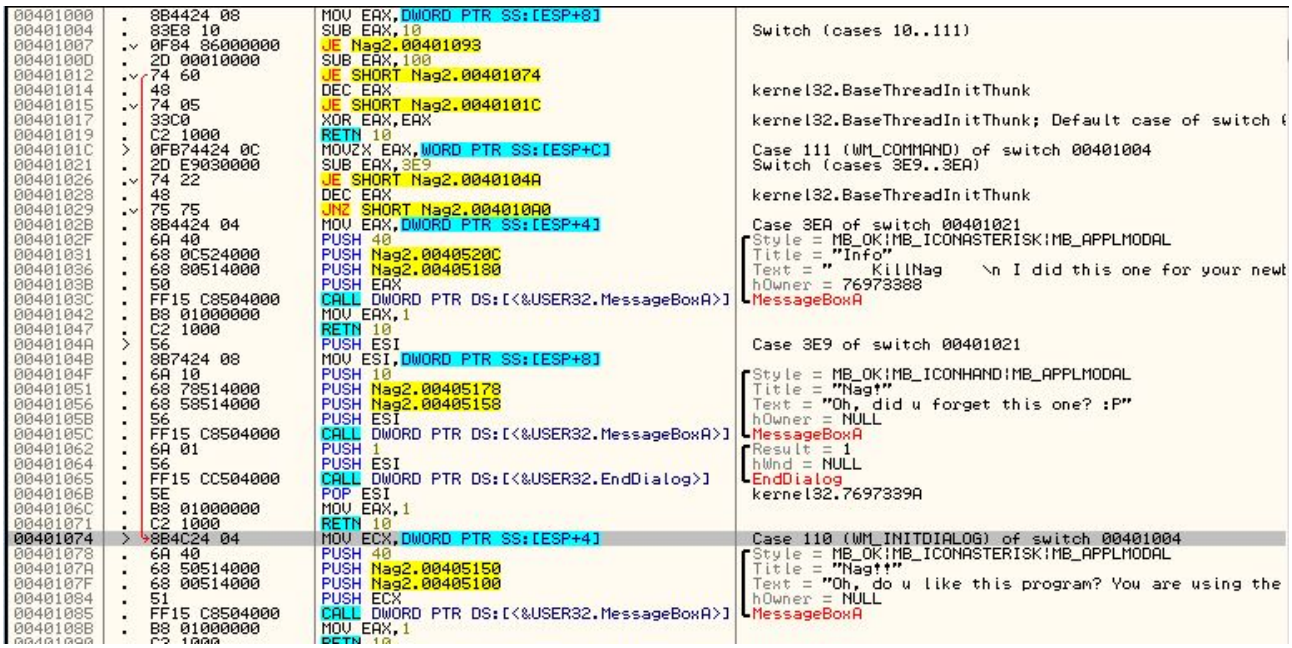
咱们继续，点击新的按钮栏中的字符串（“Str”）按钮：



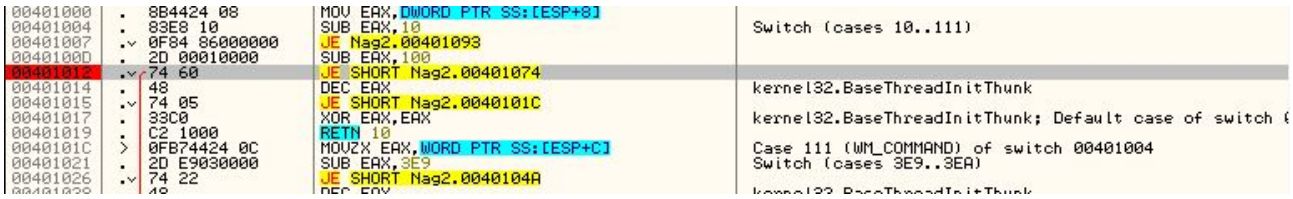
在第七行，我们看到了 nag 上面的文本，双击该行：



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：

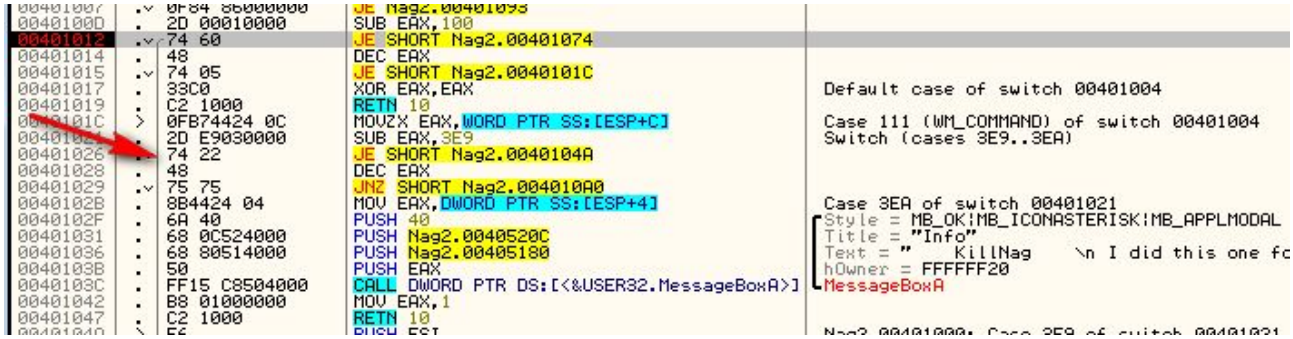


可以看到是被 401012 的 JE 指令调用。咱们在这里设置一个断点，运行程序:



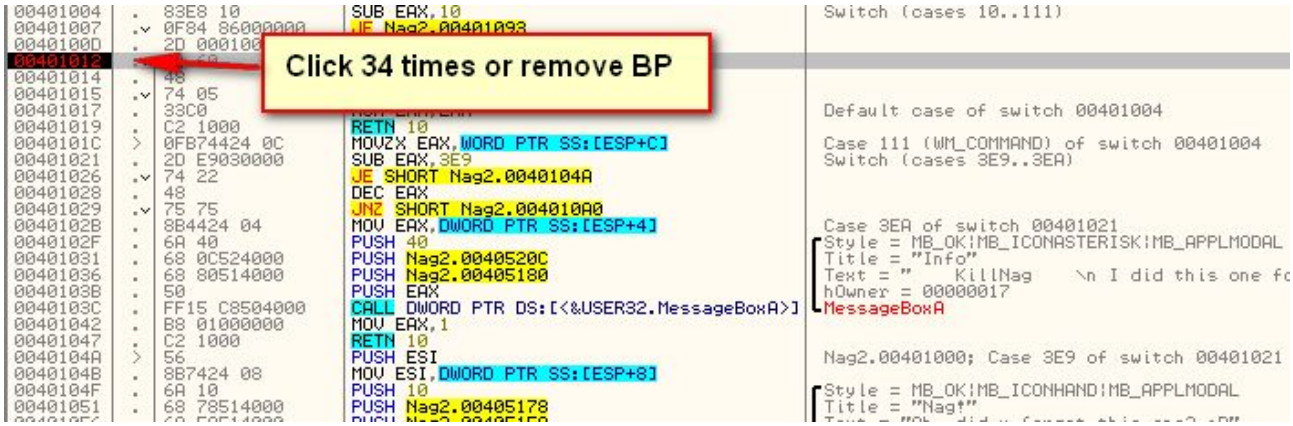
咱们断在了 JE 指令处。注意它没有调用我们的 nag 窗口代码。原来是我们刚好在 windows 的处理的中间。我们会在另一章深入讨论消息处理，不过目前我们只需要知道所有的 GUI Windows 程序都有一个消息处理程序，并且 Windows 通过它来发送各种消息。根据到达的消息，我们可以添加自己的代码来覆盖 Windows 的普通处理流程。例如，当我们点击窗口上的“X”时，Windows 就会通过消息处理程序发送一个消息说“嘿，用户想要关闭这个窗口”。我们可以让消息通过，这样的话Windows 就会处理它并关闭窗口，或者我们可以“捕获”这个消息，做任何我们想做的（可能弹出一个对话框说“你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



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 Olly 别显示它：

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：

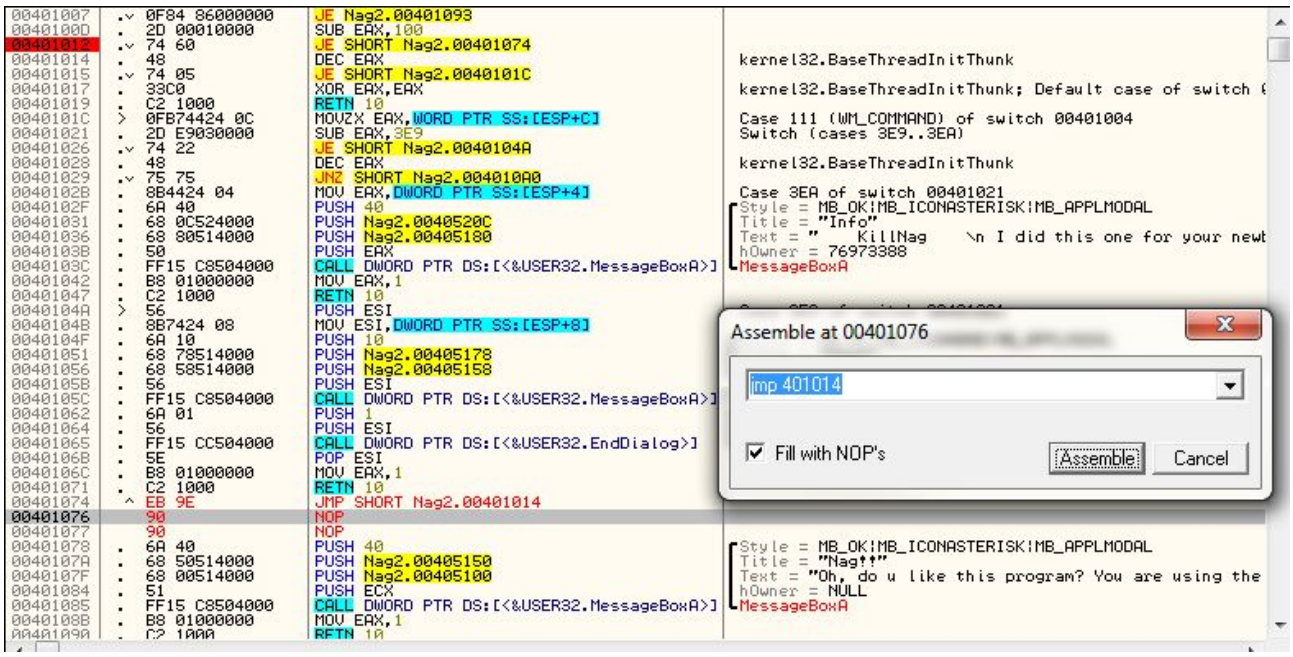


然后就看到了主窗口：

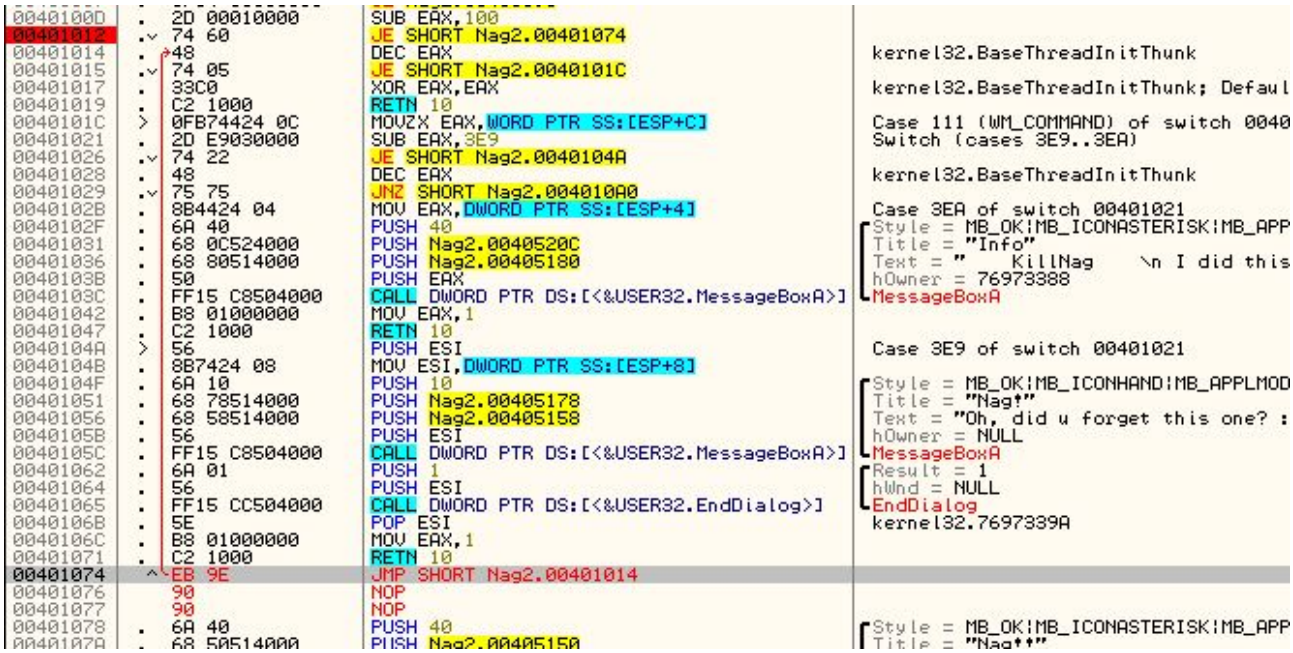
注意初始的 nag 窗口已经消失了。

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？



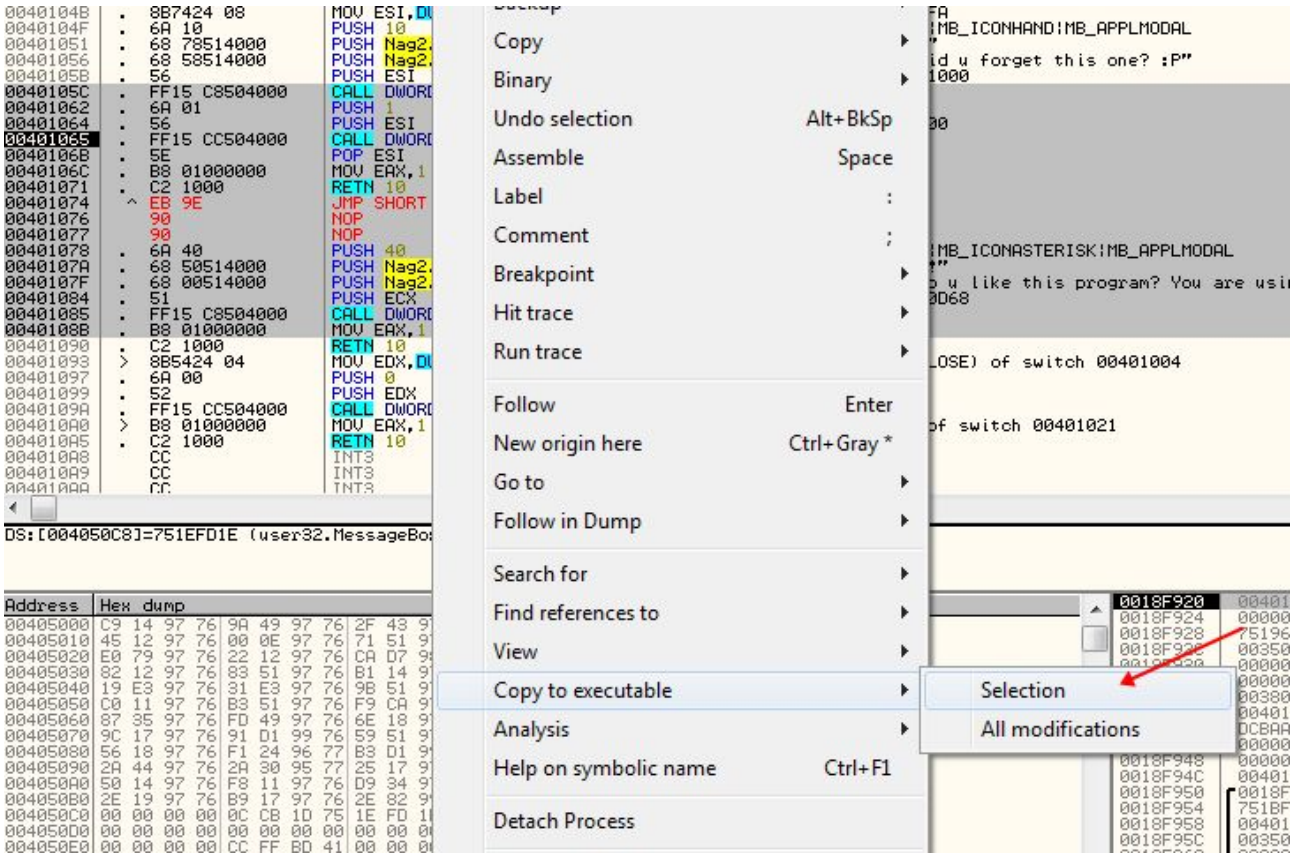
这里，将会跳转到 401074 的 nag 指令，但是我们让它又立即跳回了初始跳转的后面那行（401014）。基本上，我们的程序会跳转，然后又跳回到下一行：



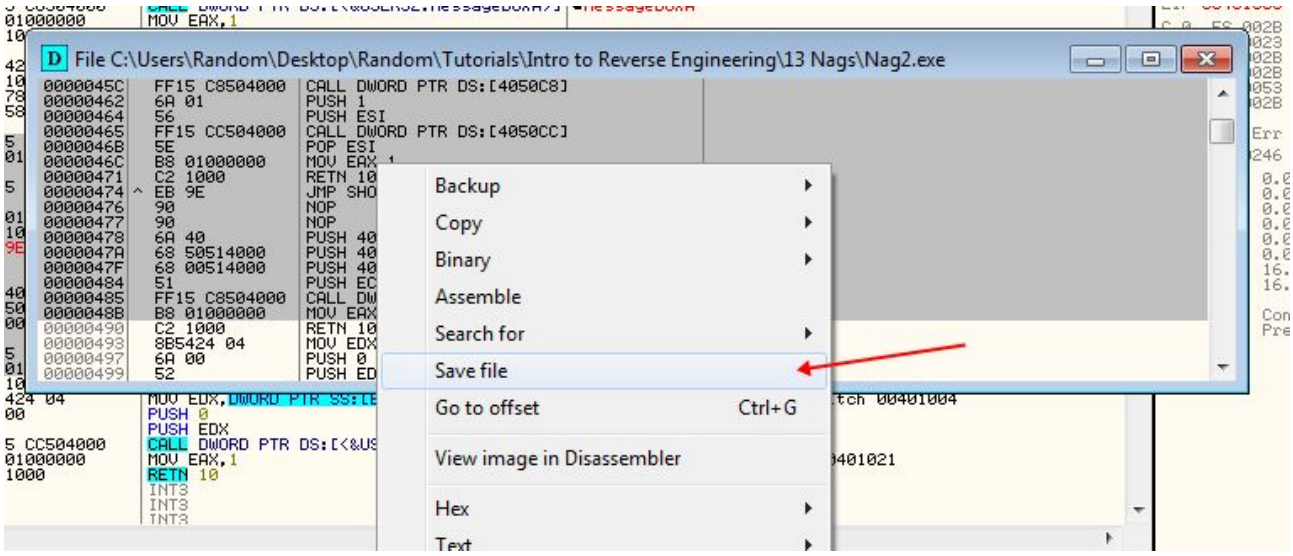
将 401012 处的 JE 指令 NOP 掉与添加一个跳回到 401074 的跳转真的没啥不同，不过我想让你开始注意到总是有多个打补丁的方法，有时候 NOP 掉一个 CALL 并不是最好的方法。记住，这个二进制文件是你的，你可以添加任何你想添加的代码，所以别害怕修改它，尤其是在学习的时候。

运行程序，可以看到 nag 窗口同样被绕过了：

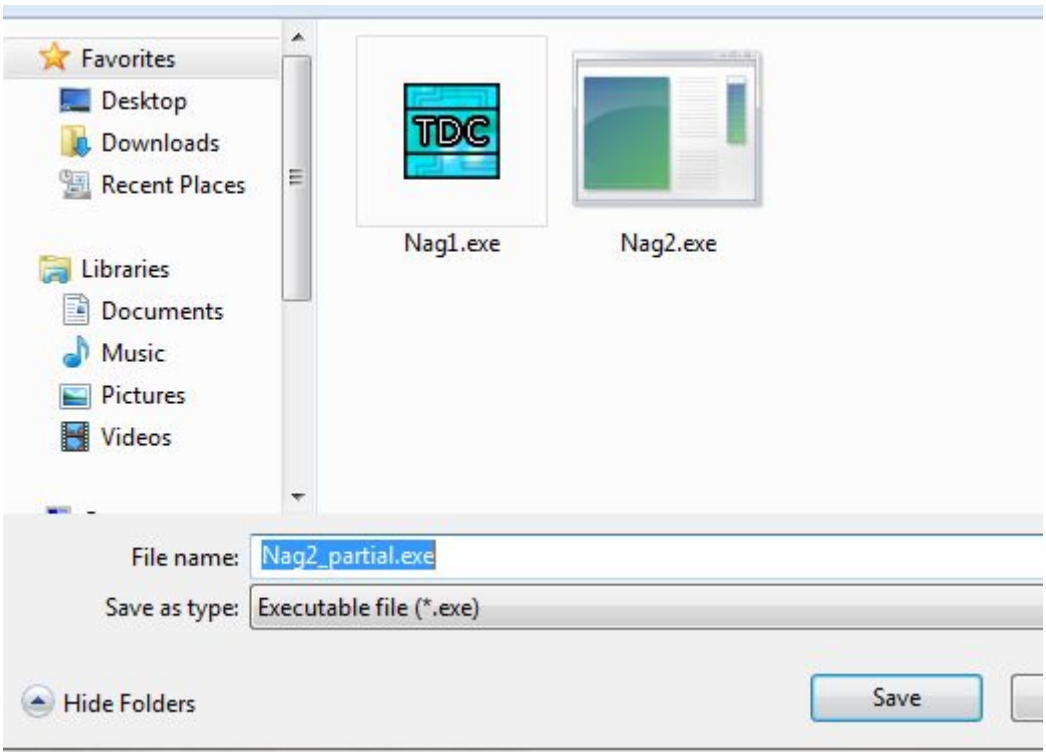
现在保存补丁。选中修改的代码（如果你选多了也没事。译者注：就是说你选中了那些没有修改的也没关系。），选择“Copy to executable” -> “Selection”：



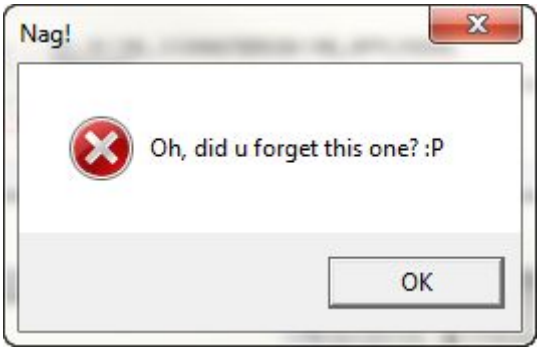
在新窗口上右键，选择“Save file”：



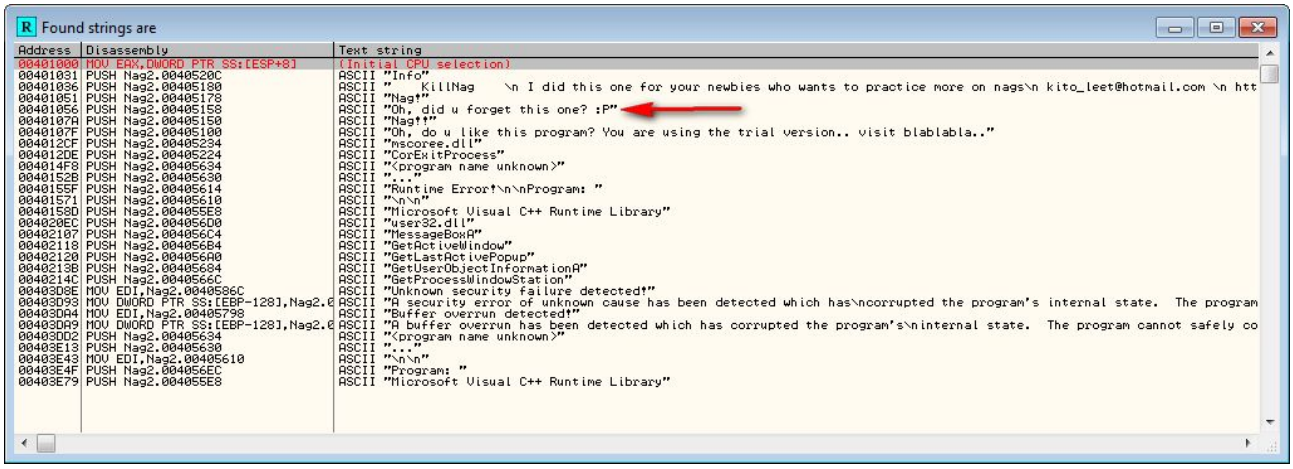
我将它保存为不同名字的文件，这里是 Nag2_partial. exe。等会你会明白我为什么把它取名为 partial:



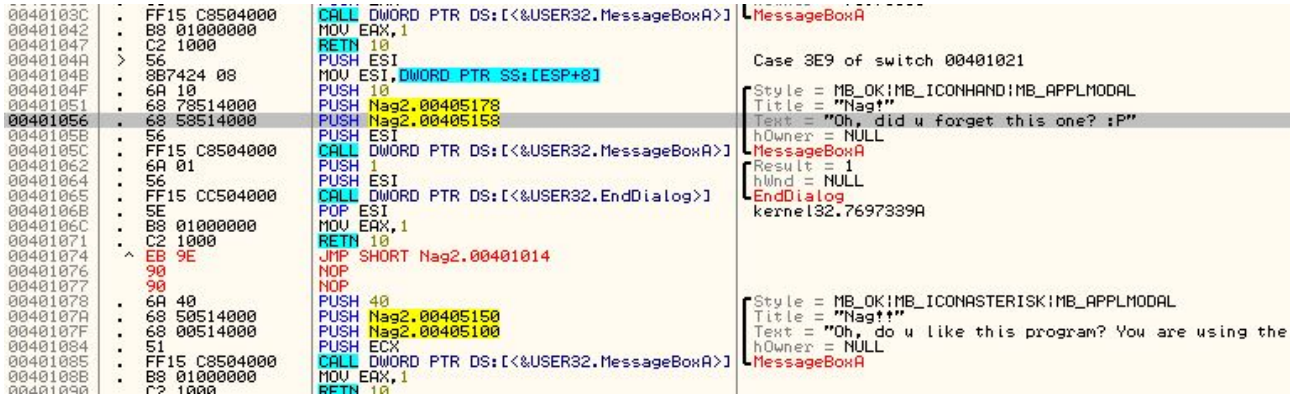
OK。咱们继续，Olly 载入这个打过补丁的程序试试看。我们直接跳到了主窗口，所以我们知道那个补丁起作用了。现在点击 exit，但是弹出了这个:



噢。我才这里作者是真的很确定（译者注：确定你没有干掉这个 nag）。咱们来找找这第二个 nag。回到字符串窗口那，可以发现这个 nag 的文本也在那：

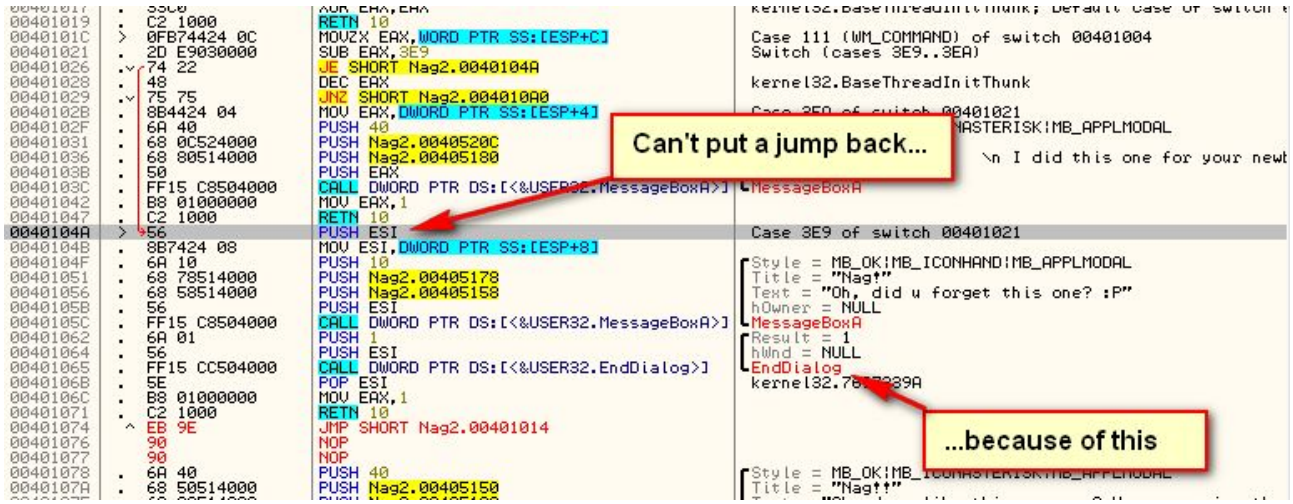


许多许多许多应用程序会这么干。它们启动的时候有个 nag，关闭了之后又来一个。大部分时候，你在搜索第一个 nag 的文本字符串时，你也应该自动找找其他的。在该文本上双击：

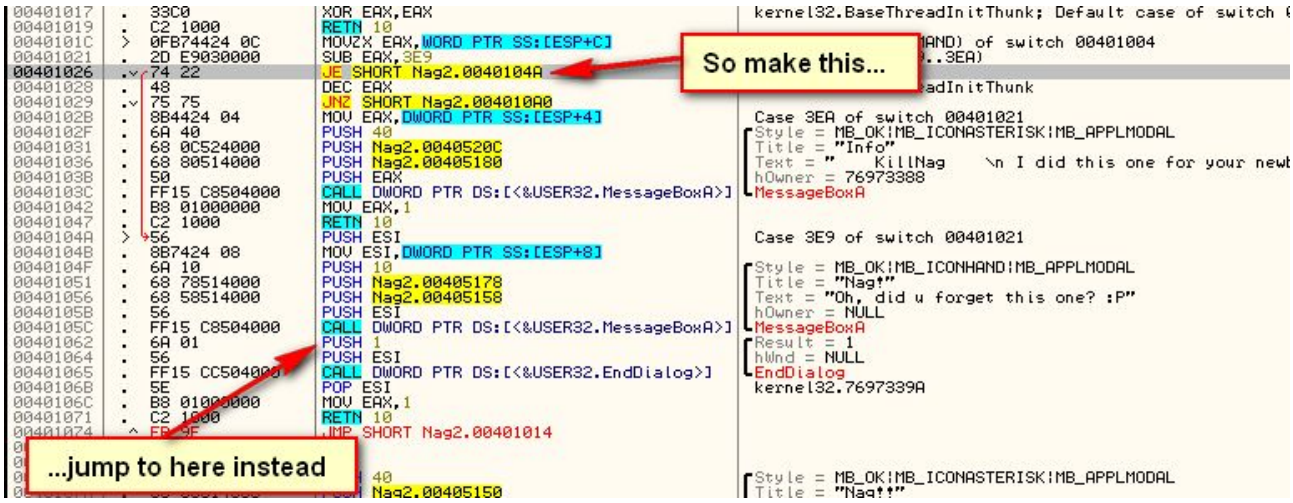


我们在这里看到了该 nag 的方法。点击该方法的第一行，我们能够看到第二个 nag 正是在第一个 nag 被调用之后调用的。但是它用了一个不同的消息来触发它（很可能是窗口销毁消息）。所以当这个消息到达时，发出用户选择 “Exit” 的信号，第二个 nag 就会被调用。

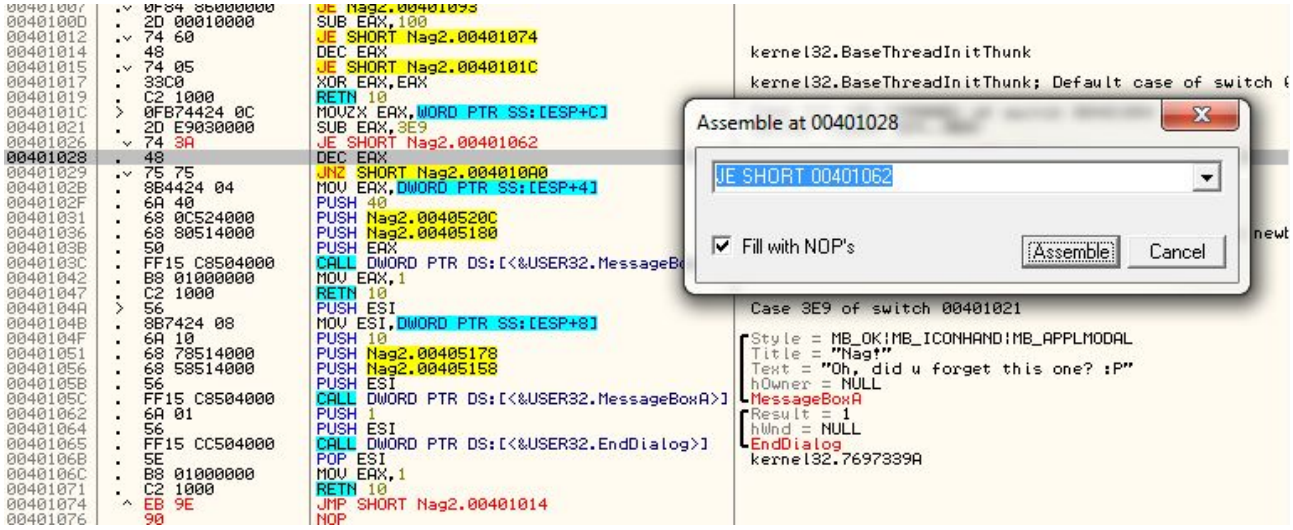
你可能首先会想到 “我们为什么不能像上一个那样，放置一个跳转让它再跳回去呢。” 好吧，你再仔细看看这个方法，我们能看到它调用了第二个 nag，但是然后它立即调用了 EndDialog。所以跳回去不会起作用，因为我们的对话框永远不会被关闭：



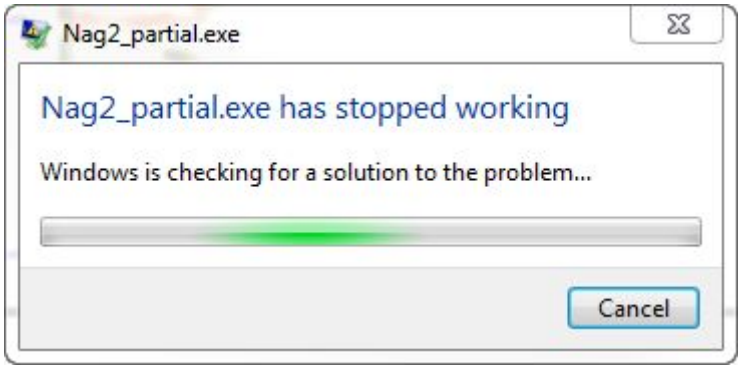
所以接下来你可能会想“咱们就把 401026 的 JE 指令修改成跳到 EndDialog，跳过显示 nag 的 MessageBoxA 的指令”。这个想法不错，咱们来试试：



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



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）：

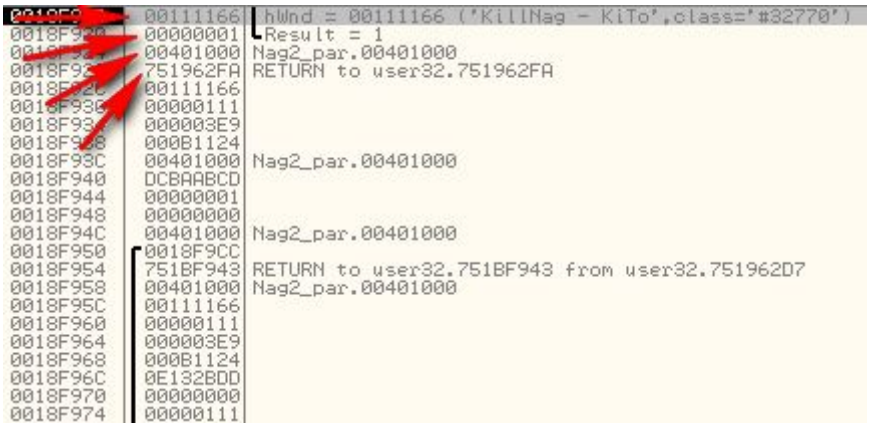
```
00401017  .  33C0            XOR EAX,EAX
00401019  .  C2 1000         RETN 10
0040101C  >  0FB74424 0C     MOVZX EAX,WORD PTR SS:[ESP+C]
00401021  .  2D E9030000     SUB EAX,3E9                          Switch (cases 3E9..3EA)
00401026  v  74 22           JE SHORT Nag2_par.0040104A
00401028  .  48              DEC EAX
00401029  .v 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 = "    KillNag    \n I did this one for your ne
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                             Nag2_par.00401000; Case 3E9 of switch 00401021
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：

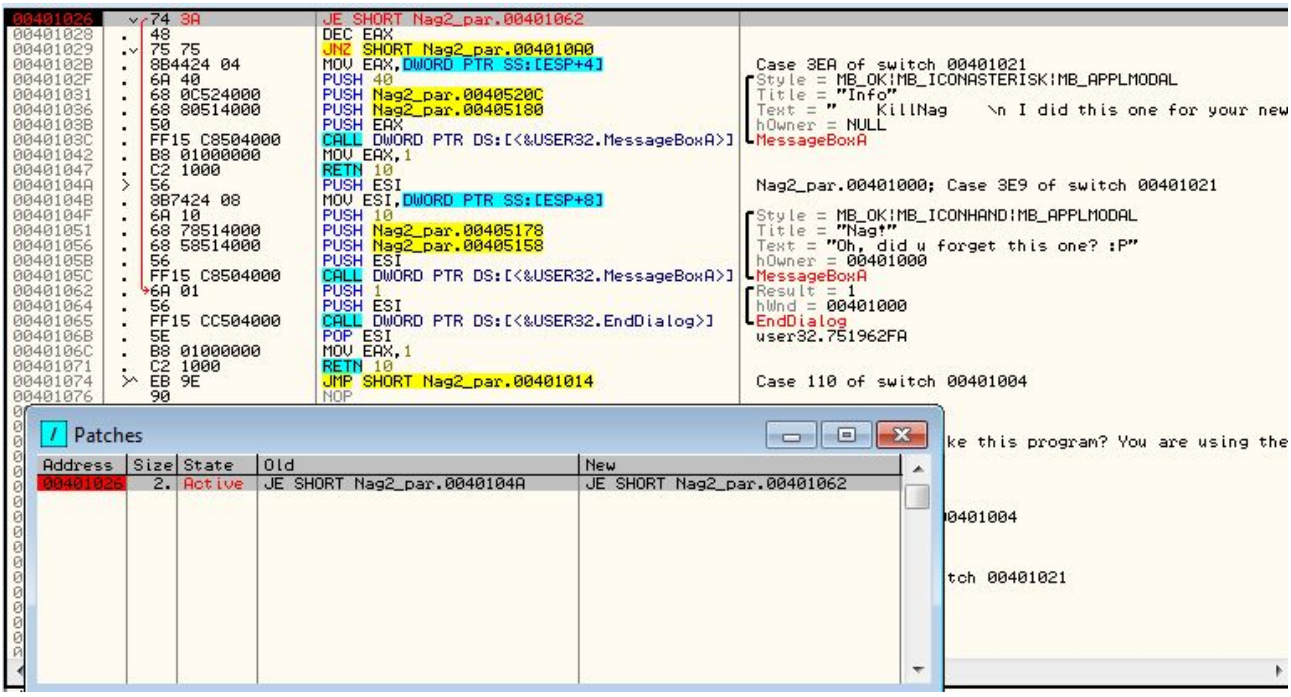
再单步两次，直到对 EndDialog 调用的那个 CALL：

```
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
00401074  >  EB 9E           JMP SHORT Nag2_par.00401014          Case 110 of switch 00401004
00401076     90              NOP
```

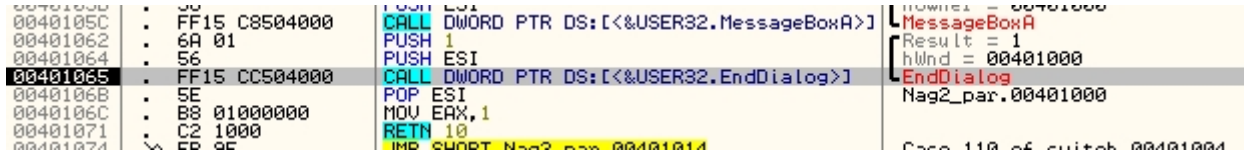
咱们来看看堆栈。在堆栈中我们看到四个项目：我们窗口的句柄、对话框的返回值、指向第一行代码（401000）的指针、返回到 user32 的地址。



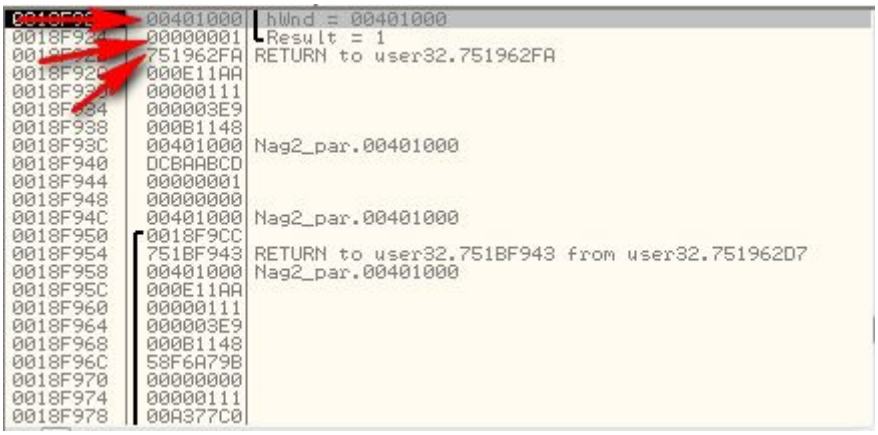
重启应用，当我们来到打补丁的地方后将其激活（打开补丁窗口，选中后按一下空格键）:



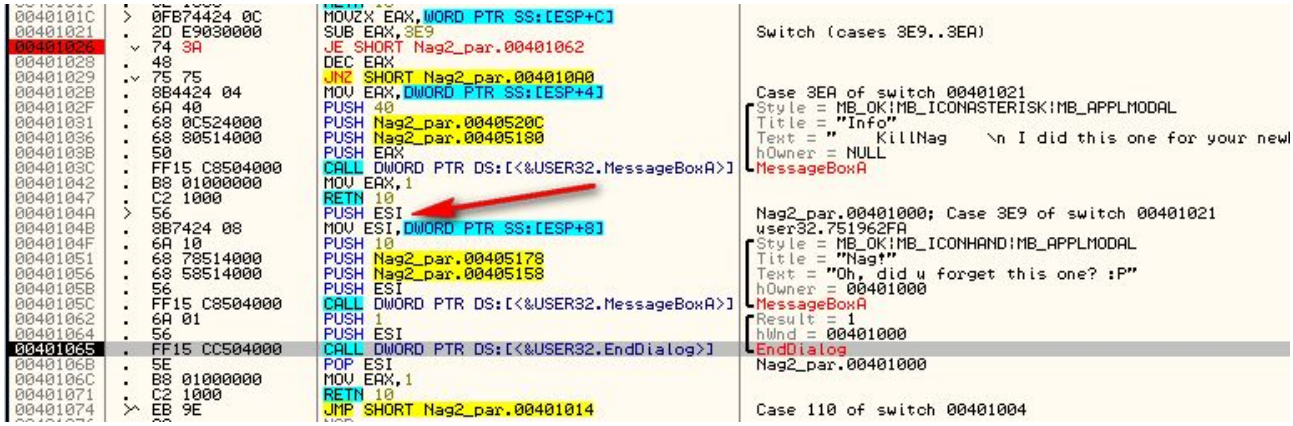
现在我们将跳过 nag 消息框。单步执行直到来到对 EndDialog 的调用处:



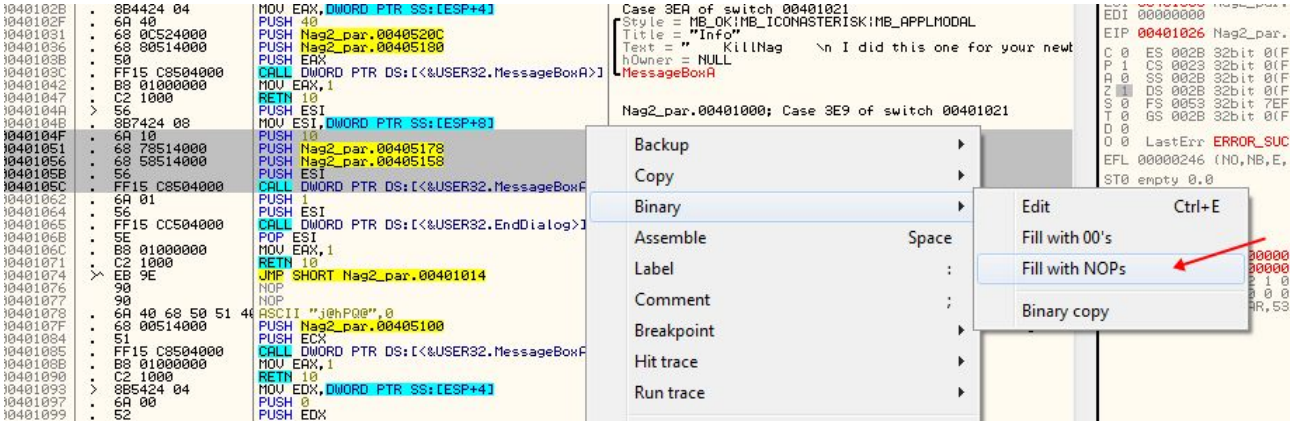
看看堆栈，有窗口的句柄、返回值、返回到 user32，但是没有了指向代码的第一行也就是 401000 的指针!!!



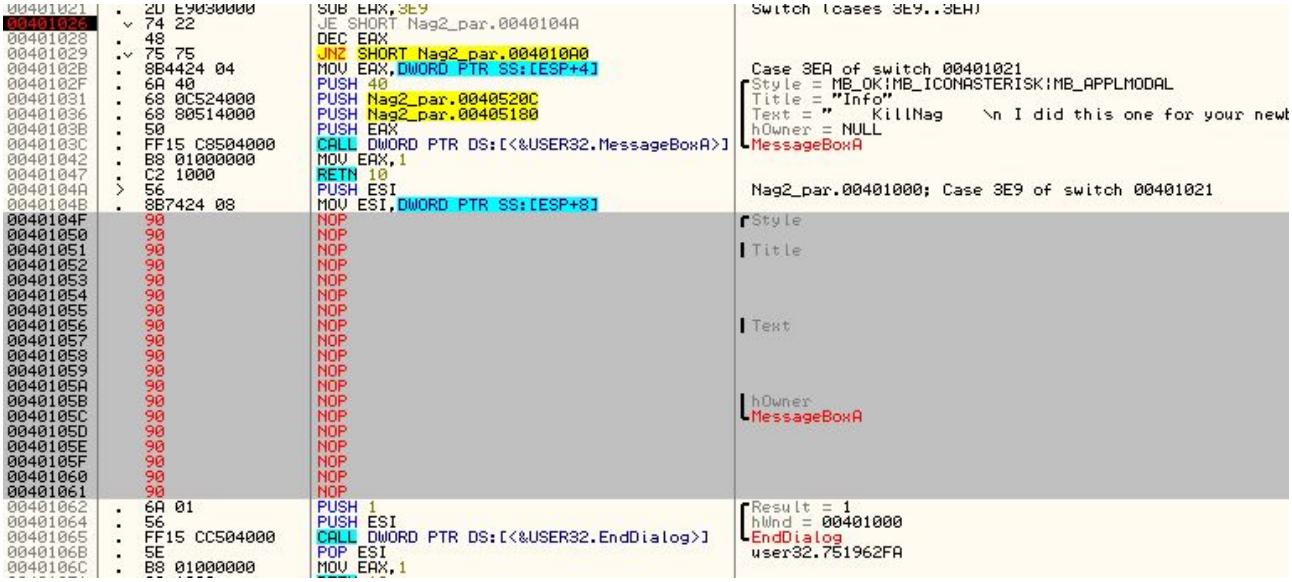
如果你向上滚动看看第二个 nag 的 CALL，你会发现在消息框被创建之前，ESI 被压栈了。这是一个指向我们代码的指针。在调用消息框前，它刚好被压入堆栈，虽然它也可以在这之后被压栈。所以我们丢失了一个重要的 push 操作，为了正确的运行 EndDialog 程序需要它。问题是，我们有一些想要的初始化代码，然后是一个我们不想要的对 nag 窗口的 CALL，之后又是一个我们想要的对 EndDialog 调用的 CALL:



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”:



然后，砰！再也没有对 nag 的 CALL 了:



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。