

R4ndom's Tutorial #14: NAGS (And I don't Mean Your Mother)

by R4ndom on Jul.16, 2012, under [Beginner](#), [Reverse Engineering](#), [Tutorials](#)

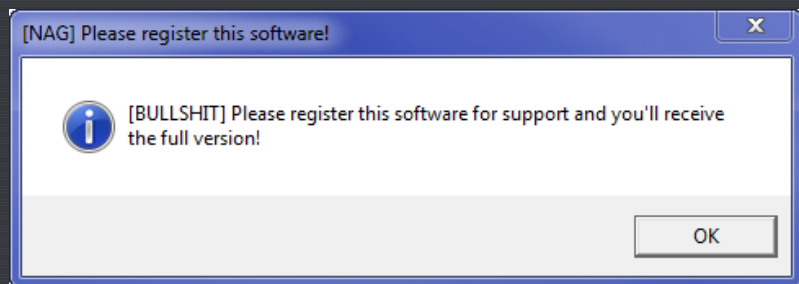
Introduction

Nags, or nag screens, are generally message boxes that pop up to remind you that your trial is ending, you need to register, a reminder about visiting the website... basically anything that's nagging and not necessary (like most bosses 😏). Many Freeware programs come free because they're full of nags (ads, time-trials, re-directs). Commercial software also includes them often, reminding you "you have 18 days left to try this product." etc. Getting rid of nags is a central theme in reverse engineering, and sometimes provides it's own set of challenges. In this tutorial we will be going over two apps that have nags. We will then bypass them so they no longer show, and then patch them so they won't ever come back.

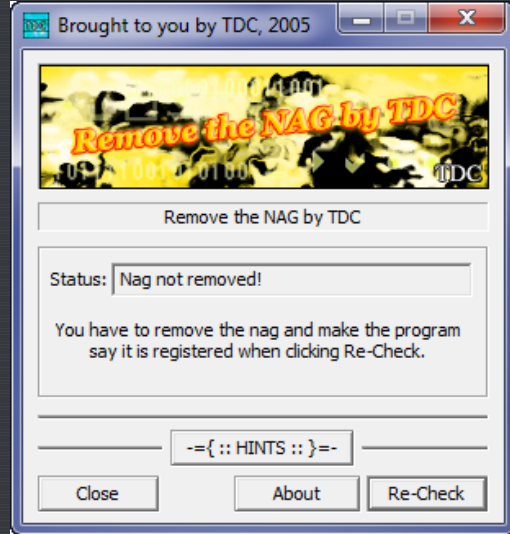
I will also be introducing a new plugin for Olly called IDAFicator. It has many features and settings. you can download the plugin from the [tools](#) page. Because there are so many features, I am also including a tutorial by the author of IDAFicator in the download for this tutorial. I highly recommend watching it as there are a lot of very cool features to this plugin.

The First App

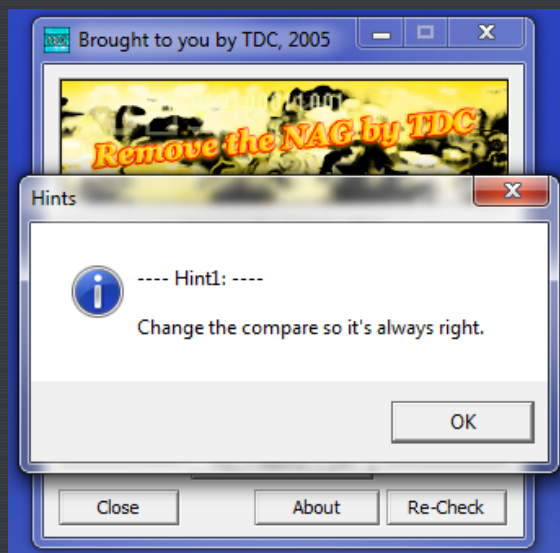
The first binary we will look at is Nag1.exe. Running the program immediately pops up the nag:



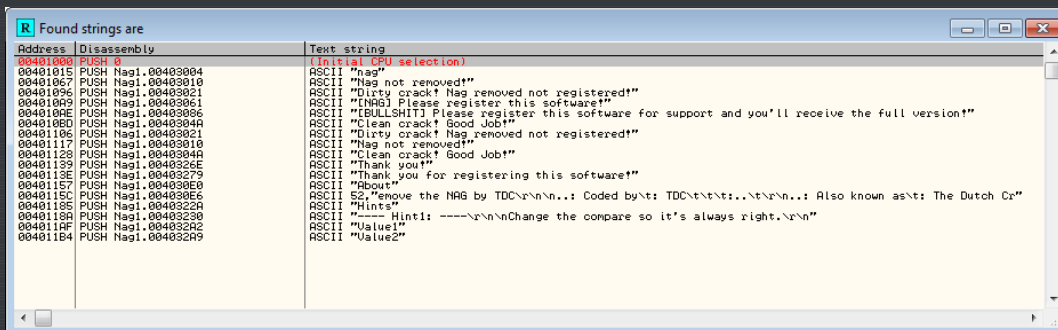
You can obviously tell this was made by a cracker 😏. Anyway, after clicking OK you get the main screen:



Notice it says "Nag not removed!". I, of course, could not help clicking the "Hints" button and was rewarded with some very detailed information:



Gee, thanks. Load the app in Olly and let's try old reliable: search for strings:



and we're in luck. You can see the text for the nag screen at address 4010AE. Let's dbl-click that and jump to where the nag is created:

00401078	803D B0324000 03	CMP BYTE PTR DS:[4032B01],3	CALL Nag1.00401096	Text = "Dirty crack! Nag removed not registered!" ControlID = 73 (115.) hWnd = 7EFDE000 SetDlgItemTextA
00401082	74 12	JE SHORT Nag1.00401096		
00401084	803D B0324000 02	CMP BYTE PTR DS:[4032B01],2		
00401088	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	817D 0C 11010000	CMPL [ARG.2],111		
004010DA	0F85 B9000000	JNZ Nag1.00401199		
004010E0	837D 10 6F	CMPL [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		

Hmm, an interesting string above it, but let's ignore that for now. Let's click on the first line of the MessageBoxA code at address 4010A7 to see where it's called from:

00401078	803D B0324000 03	CMP BYTE PTR DS:[4032B01],3	CALL Nag1.00401096	Text = "Dirty crack! Nag removed not registered!" ControlID = 73 (115.) hWnd = 7EFDE000 SetDlgItemTextA
00401082	74 12	JE SHORT Nag1.00401096		
00401084	803D B0324000 02	CMP BYTE PTR DS:[4032B01],2		
00401088	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		

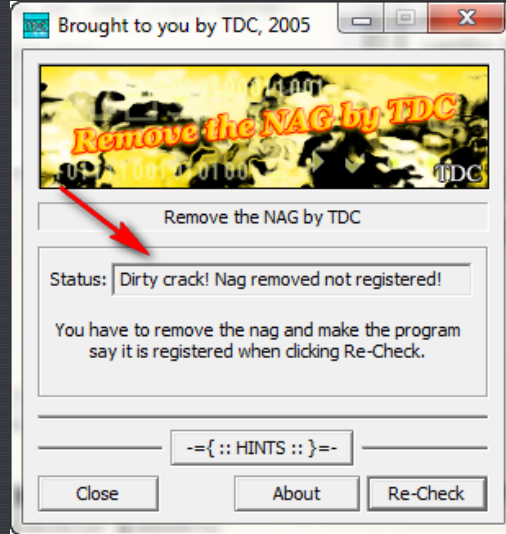
And we can see that it is called from a JE instruction at address 40108B, right after a compare. Well, we certainly recognize this scenario 😊. Let's place a BP on that JE instruction:

00401078	803D B0324000 03	CMP BYTE PTR DS:[4032B01],3	CALL Nag1.00401096	Text = "Dirty crack! Nag removed not registered!" ControlID = 73 (115.) hWnd = 7EFDE000 SetDlgItemTextA
00401082	74 12	JE SHORT Nag1.00401096		
00401084	803D B0324000 02	CMP BYTE PTR DS:[4032B01],2		
00401088	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		

And run the app. We break at our BP and see that we are going to jump to the nag screen instructions, so let's make it not jump:

C	0	ES	002
D	0	CS	002
E	0	SS	002
F	0	DS	002
S	0	FS	002
T	0	GS	002
D	0		
0	0		

and then run the app:



So, that's what the "Dirty Crack" thing was all about- apparently we didn't patch enough. Let's restart the app and Olly will pause at our BP. Zero out the zero flag again:

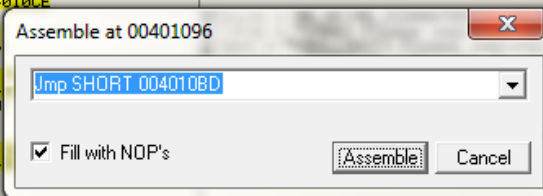
```
C 0 ES 002
P 1 CS 002
A 0 SS 002
Z 0 DS 002
S 0 FS 002
T 0 GS 002
D 0
```

and let's step twice to the next jump. As you could probably guess by now, this jump SHOULD jump to our good boy, but instead falls through to our bad boy:

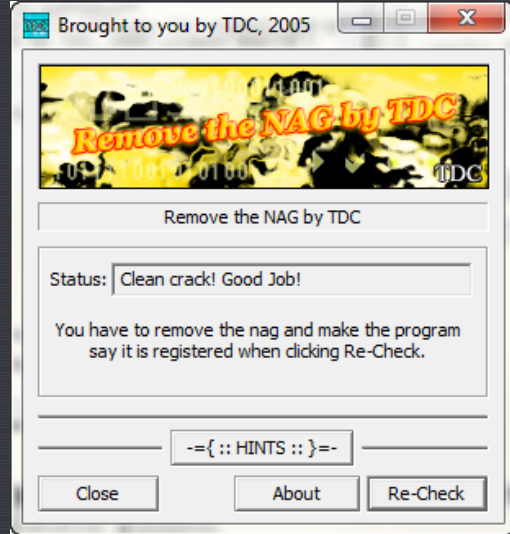
00401092	> 74 12	JMP SHORT Nag1.00401096	
00401094	> 803D B0324000 02	CMP BYTE PTR DS:[4032B0],2	
00401096	> 74 1A	JMP SHORT Nag1.004010A7	
00401098	> 803D B0324000 01	CMP BYTE PTR DS:[4032B0],1	
0040109A	> EB 27	JMP SHORT Nag1.004010BD	
0040109C	> 6A 40	PUSH Nag1.00403021	
0040109E	> FF75 08	PUSH [ARG.1]	
004010A0	> E8 6B010000	CALL <JMP.&user32.SetDlgItemTextA>	
004010A2	> EB 11	JMP SHORT Nag1.004010CE	
004010A4	> 6A 40	PUSH Nag1.00403021	
004010A6	> FF75 08	PUSH [ARG.1]	
004010A8	> E8 49010000	CALL <JMP.&user32.MessageBoxA>	
004010AA	> EB 11	JMP SHORT Nag1.004010CE	
004010AC	> 6A 73	PUSH Nag1.0040304A	
004010AE	> FF75 08	PUSH [ARG.1]	
004010B0	> E8 44010000	CALL <JMP.&user32.SetDlgItemTextA>	
004010B2	> EB 00	JMP SHORT Nag1.004010CE	
004010B4	> E9 D6000000	JMP Nag1.004011A9	
004010B6	> 817D 0C 11010000	CMP [ARG.2],111	
004010B8	> 0F85 B9000000	JNZ Nag1.00401199	
004010BA	> 837D 10 6F	CMP [ARG.3],6F	
004010BC	> 75 69	JNZ SHORT Nag1.0040114F	

Let's just patch that to always jump:

00401092	> 74 12	JMP SHORT Nag1.00401096	
00401094	> 803D B0324000 02	CMP BYTE PTR DS:[4032B0],2	
00401096	> 74 1A	JMP SHORT Nag1.004010A7	
00401098	> 803D B0324000 01	CMP BYTE PTR DS:[4032B0],1	
0040109A	> EB 27	JMP SHORT Nag1.004010BD	
0040109C	> 6A 40	PUSH Nag1.00403021	
0040109E	> FF75 08	PUSH [ARG.1]	
004010A0	> E8 6B010000	CALL <JMP.&user32.SetDlgItemTextA>	
004010A2	> EB 11	JMP SHORT Nag1.004010CE	
004010A4	> 6A 40	PUSH Nag1.00403021	
004010A6	> FF75 08	PUSH [ARG.1]	
004010A8	> E8 49010000	CALL <JMP.&user32.MessageBoxA>	
004010AA	> EB 11	JMP SHORT Nag1.004010CE	
004010AC	> 6A 73	PUSH Nag1.0040304A	
004010AE	> FF75 08	PUSH [ARG.1]	
004010B0	> E8 44010000	CALL <JMP.&user32.SetDlgItemTextA>	
004010B2	> EB 00	JMP SHORT Nag1.004010CE	
004010B4	> E9 D6000000	JMP Nag1.004011A9	
004010B6	> 817D 0C 11010000	CMP [ARG.2],111	
004010B8	> 0F85 B9000000	JNZ Nag1.00401199	
004010BA	> 837D 10 6F	CMP [ARG.3],6F	
004010BC	> 75 69	JNZ SHORT Nag1.0040114F	



and when we run the app we see we were right:



We can obviously patch this program by keeping our current patch and going back to address 40108B (where we originally zeroed out the zero flag) and patch it to never jump. Saving these two patches will work fine. But I also want to show you (as I have mentioned before, and if I haven't I should have) that there are ALWAYS other ways to patch an app, usually many. Restart the app and scroll to our BP:

0040108E	FF75 08	PUSH [ARG.1]	hWnd = 001C0008
00401071	E8 90010000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
00401076	E8 34010000	CALL Nag1.004011AF	
0040107B	803D B0324000 03	CMP BYTE PTR DS:[4032B0],3	
00401082	74 12	JE SHORT Nag1.00401096	
00401084	803D B0324000 02	CMP BYTE PTR DS:[4032B0],2	
0040108B	74 1A	JE SHORT Nag1.004010A7	
0040108D	803D B0324000 01	CMP BYTE PTR DS:[4032B0],1	
00401094	EB 27	JMP SHORT Nag1.004010BD	
00401096	68 21304000	PUSH Nag1.00403021	Text = "Dirty crack! Nag removed not registered!"
0040109B	6A 73	PUSH 73	ControlID = 73 (115.)
0040109D	FF75 08	PUSH [ARG.1]	hWnd = 001C0008
004010A0	E8 6B010000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
004010A5	EB 27	JMP SHORT Nag1.004010CE	
004010A7	6A 40	PUSH 40	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL
004010A9	68 61304000	PUSH Nag1.00403061	Title = "[NAG] Please register this software!"
004010AB	68 86304000	PUSH Nag1.00403086	Text = "[BULLSHIT] Please register this software for support
004010B3	FF75 08	PUSH [ARG.1]	hOwner = 001C0008
004010B6	E8 49010000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
004010BB	EB 11	JMP SHORT Nag1.004010CE	
004010BD	68 4A304000	PUSH Nag1.0040304A	Text = "Clean crack! Good Job!"
004010C2	6A 73	PUSH 73	ControlID = 73 (115.)
004010C4	FF75 08	PUSH [ARG.1]	hWnd = 001C0008
004010C7	E8 44010000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
004010CC	EB 00	JMP SHORT Nag1.004010CE	
004010CE	E9 D6000000	JMP Nag1.004011A9	
004010D3	817D 0C 11010000	CMP [ARG.2],111	

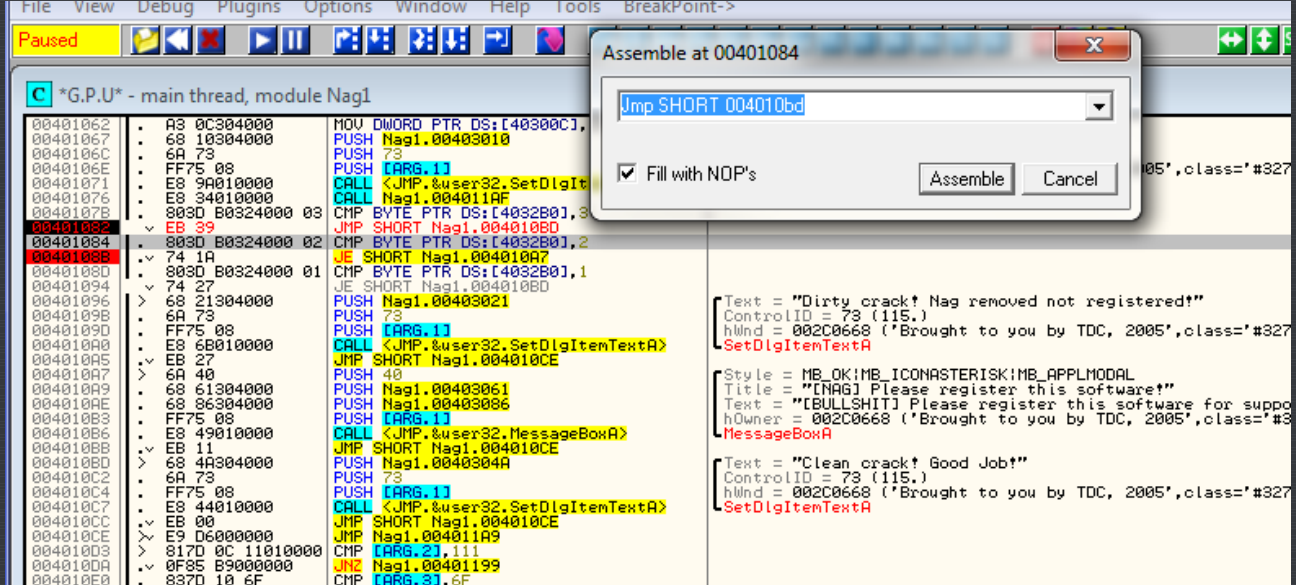
Notice that this collection of instructions is something like this (in a high-level language):

```

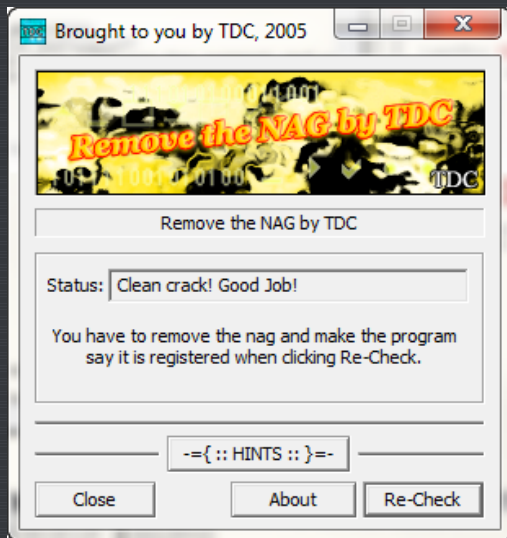
if (contents of 4032B0 == 3)
    jump "Dirty Crack"
else if (contents of 4032B0 == 2)
    jump to "Show Nag Screen"
else if (contents of 4032B0 == 1)
    jump to Good Boy Msg
else
    Display "Dirty Crack"

```

We know that since the nag screen is displayed by default, the contents of memory address 4032B0 will always equal 2, as that's the jump that is taken. Well, what if we just bypassed this whole if/then clause and immediately jumped to the good boy? So if we replace the very first jump to just jump to the good boy, we would only need one patch. Try it:

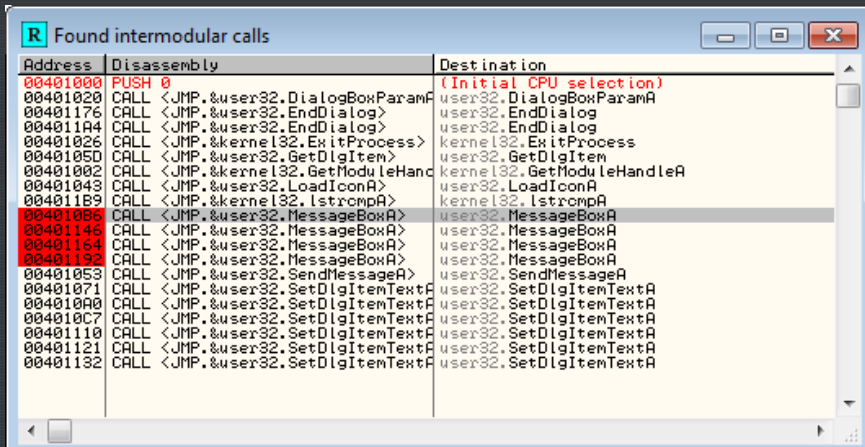


Now run the app:



And you can see that it accomplished the same thing. Another, even more elegant solution may be to think, "If the contents of 4032B0 are always equal to 2, and to hit the good boy message it needs to be 1, why not just place a 1 in this memory location and we'll always hit the good boy?" You should try this. Restart the app, click on the dump window, go to address 4032B0 and binary edit it to be a one. Did it work?

Another thing to keep in mind is there are always other ways to find the code section we are looking for. For example, if we couldn't use strings in this example, we could do a search for Intermodular calls:



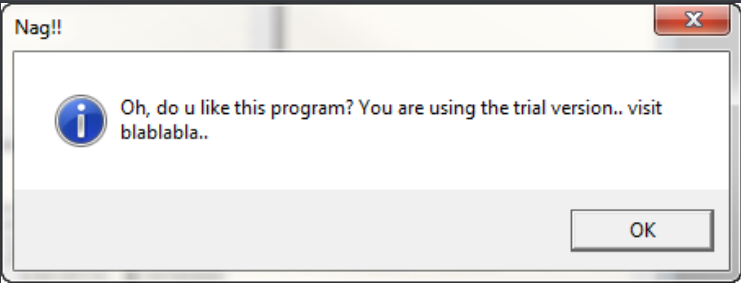
Notice that there are 4 calls to MessageBoxA. Right click one of them and choose "Place a breakpoint on every call to MessageBoxA". When you run the app, before anything is displayed we stop at the following line of code:

00401094	> 74 27	JE SHORT Nag1.004010BD	
00401096	> 68 21304000	PUSH Nag1.00403021	Text = "Dirty crack! Nag removed not registered!"
00401098	> 6A 73	PUSH 73	ControlID = 73 (115.)
0040109D	> FF75 08	PUSH [ARG.1]	hWnd = 0030063E ('Brought to you by TDC, 2005',class='#32770')
004010A0	> E8 65010000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
004010A5	> EB 27	JMP SHORT Nag1.004010CE	
004010A7	> 6A 40	PUSH 40	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL
004010A9	> 68 61304000	PUSH Nag1.00403061	Title = "[NAG] Please register this software!"
004010AE	> 68 86304000	PUSH Nag1.00403086	Text = "[BULLSHIT] Please register this software for support and y
004010B3	> FF75 08	PUSH [ARG.1]	hOwner = 0030063E ('Brought to you by TDC, 2005',class='#32770')
004010B6	> E8 49010000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
004010BB	> EB 11	JMP SHORT Nag1.004010CE	
004010BD	> 68 4A304000	PUSH Nag1.0040304A	Text = "Clean crack! Good Job!"
004010C2	> 6A 73	PUSH 73	ControlID = 73 (115.)
004010C4	> FF75 08	PUSH [ARG.1]	hWnd = 0030063E ('Brought to you by TDC, 2005',class='#32770')
004010C7	> E8 44010000	CALL <JMP.&user32.SetDlgItemTextA>	SetDlgItemTextA
004010CC	> EB 00	JMP SHORT Nag1.004010CE	
004010CE	> E9 06000000	JMP Nag1.004011A9	
004010D3	> 817D 0C 11010000	CMPL [ARG.2],111	
004010D9	> 75 82000000	JNZ Nag1.00401198	

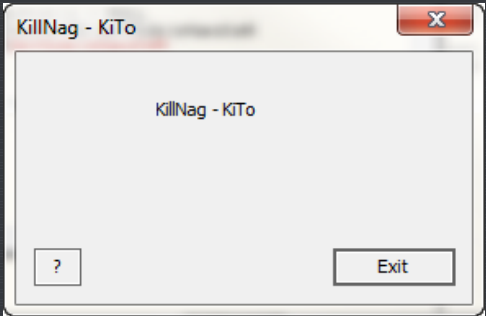
Look familiar? It is the nag messagebox !! So always keep in mind that there are more than one way to accomplish something. Soon, we will also be learning some other techniques that can be used (like windows message handlers) that will give you an even bigger bag of tricks.

The Second App

Now let's take a look at Nag2.exe. It is similar but we will solve it in different ways. When we start the app, we get the expected nag:



and after clicking OK we get the main screen:



At this point I closed the app and loaded it into Olly:

```

C *G.P.U* - main thread, module Nag2
004010F3  6A 60      PUSH 60
004010F5  68 18524000  PUSH Nag2.00405218
004010FA  E8 810D0000  CALL Nag2.00401E80
004010FF  BF 94000000  MOV EDI,94
00401104  8BC7      MOV EAX,EDI
00401106  E8 D50E0000  CALL Nag2.00401FE0
0040110B  8965 E8     MOV DWORD PTR SS:[EBP-18],ESP
0040110E  8BF4      MOV ESI,ESP
00401110  893E      MOV DWORD PTR DS:[ESI],EDI
00401112  56        PUSH ESI
00401113  FF15 1C504000  CALL DWORD PTR DS:[<&KERNEL32.GetVersionExA]
00401117  8B4E 10     MOV ECX,DWORD PTR DS:[ESI+10]
0040111C  8900 B8724000  MOV DWORD PTR DS:[4072B8],ECX
00401122  8B46 04     MOV EAX,DWORD PTR DS:[ESI+4]
00401125  A3 C4724000  MOV DWORD PTR DS:[4072C4],EAX
0040112A  8B56 08     MOV EDX,DWORD PTR DS:[ESI+8]
0040112D  8915 C8724000  MOV DWORD PTR DS:[4072C8],EDX
00401133  8B76 0C     MOV ESI,DWORD PTR DS:[ESI+C]
00401136  81E6 FF7F0000  AND ESI,7FFF
0040113C  8935 BC724000  MOV DWORD PTR DS:[4072BC],ESI
00401142  83F9 02     CMP ECX,2
00401145  74 0C      JE SHORT Nag2.00401153

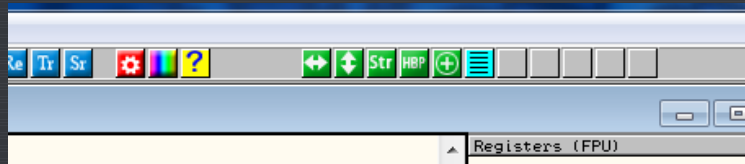
```

```

pVersionInformation = NULL
GetVersionExA
kernel32.BaseThreadInitThunk
Nag2.<ModuleEntryPoint>

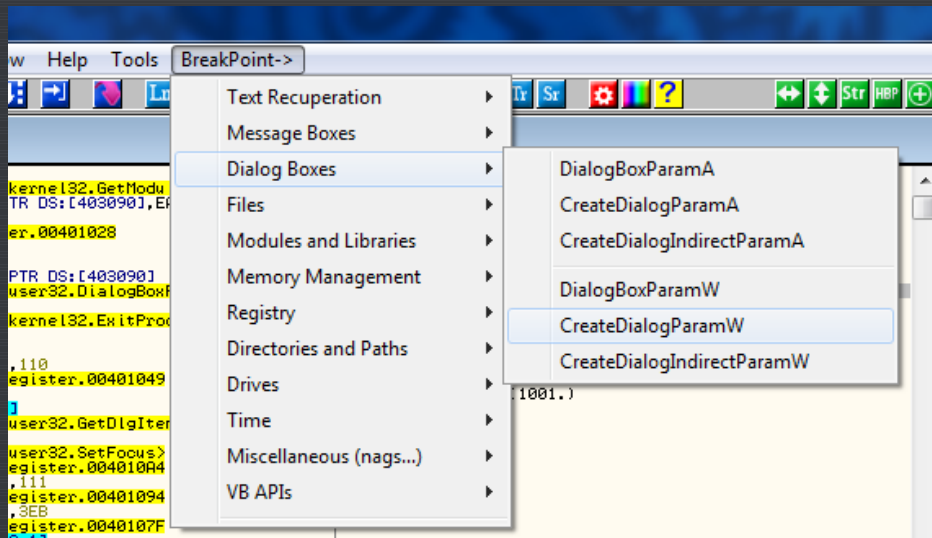
```

First things first, let's see if there are any strings. One thing I wanted to point out here is the plugin IDAFicator. Among many additions, it provides a group of buttons along the top that making searching for strings a lot easier. When clicking the strings button (Str), it shows both ASCII and Unicode and also brings the cursor up to the top automatically so you don't have to scroll to the top your self. Here's what the buttons look like:



The first button (the left and right arrow) take you forward and back. For example, if you click on a call, then press enter to go to that call, clicking the first icon will take you back to the call instruction. Right-clicking takes you forward. The second button will attempt to find the beginning of the current function, while right-clicking will attempt to find the end. The next is the strings button. Next is the Hardware BreakPoints button. It brings up a nice dialog that shows you all of your hardware breakpoints. Very handy. The target icon opens the folder where your app resides and the list icon brings up a dialog to enter multiple lines of assembly code, used for if you are changing a substantial part of the exe.

You will also notice a new menu item called "Breakpoint->" that opens a drop down of many used API calls so you can set breakpoints on them automatically:



Lastly, there is a context menu item added that allows you to restore hidden bytes, which we will get to in a future tutorial.

So go ahead and click in the strings button ("Str") on the new button bar:

Address	Disassembly	Text string
00401000	MOV EAX, DWORD PTR SS:[ESP+8]	(Initial CPU selection)
00401001	PUSH Nag2.0040520C	ASCII "Info"
00401006	PUSH Nag2.00405190	ASCII "KillNag \n I did this one for your newbies who wants to practice more on nags\n kito_leet@hotmail.com \n htt
00401001	PUSH Nag2.00405178	ASCII "Nag"
00401006	PUSH Nag2.00405158	ASCII "Oh, did u forget this one? :P"
0040107A	PUSH Nag2.00405150	ASCII "Nag!"
0040107F	PUSH Nag2.00405100	ASCII "Oh, do u like this program? You are using the trial version.. visit blablalbla.."
004012CF	PUSH Nag2.00405634	ASCII "mscore.dll"
004012DE	PUSH Nag2.00405224	ASCII "CoRExitProcess"
004014F8	PUSH Nag2.00405634	ASCII "Program name unknown"
0040152B	PUSH Nag2.00405630	ASCII "..."
0040155F	PUSH Nag2.00405614	ASCII "Runtime Error!\n\nProgram: "
00401571	PUSH Nag2.00405610	ASCII "\n\n"
00401580	PUSH Nag2.004056E3	ASCII "Microsoft Visual C++ Runtime Library"
004020EC	PUSH Nag2.004056D0	ASCII "user32.dll"
00402107	PUSH Nag2.004056C4	ASCII "MessageBox"
00402118	PUSH Nag2.00405684	ASCII "GetActiveWindow"
00402120	PUSH Nag2.004056A0	ASCII "GetLastErrorPopup"
00402181	PUSH Nag2.00405684	ASCII "GetUserObjectInformation"
0040214C	PUSH Nag2.00405684	ASCII "GetProcessWindowStation"
0040305E	MOV EDI, Nag2.0040586C	ASCII "Unknown security failure detected"
00403093	MOV DWORD PTR SS:[EBP-128], Nag2.00405798	ASCII "A security error of unknown cause has been detected which has\ncorrupted the program's internal state. The program
004030A9	MOV DWORD PTR SS:[EBP-128], Nag2.00405798	ASCII "Buffer overrun detected"
004030D2	PUSH Nag2.00405694	ASCII "A buffer overrun has been detected which has corrupted the program's\ninternal state. The program cannot safely co
00403E13	PUSH Nag2.00405630	ASCII "..."
00403E43	MOV EDI, Nag2.00405610	ASCII "\n\n"
00403E4F	PUSH Nag2.004056C0	ASCII "Program: "
00403E79	PUSH Nag2.004056E8	ASCII "Microsoft Visual C++ Runtime Library"

and on the seventh line down we see our nag's text, so let's double-click:

0040106B	SE	POP ESI	kernel32.7697339A
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
0040108B	B8 01000000	MOV EAX, 1	
00401090	C2 1000	RETN 10	
00401093	> 8B5424 04	MOV EDX, DWORD PTR SS:[ESP+4]	Case 10 (WM_CLOSE) of switch 00401004
00401097	6A 00	PUSH 0	Result = 0
00401099	52	PUSH EDX	hWnd = 004010F3
0040109A	FF15 CC504000	CALL DWORD PTR DS:[<&USER32.EndDialog]	EndDialog
004010A0	> B8 01000000	MOV EAX, 1	Default case of switch 00401021
004010A5	C2 1000	RETN 10	
004010A8	CC	INT3	
004010A9	CC	INT3	

To see the nag's method. It is a self contained method (there is a RETN above and below it) so we know it is called from somewhere. Click on the first line of it at address 401074 to see where it's called from:

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 Nag2.00401000	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 Nag2.0040520C	Text = "KillNag \n I did this one for your newb
00401036	68 80514000	PUSH Nag2.00405100	hOwner = 76973388
0040103B	50	PUSH EAX	MessageBoxA
0040103C	FF15 C8504000	CALL DWORD PTR DS:[<&USER32.MessageBoxA]	
00401042	B8 01000000	MOV EAX, 1	
00401047	C2 1000	RETN 10	Case 3E9 of switch 00401021
00401049	>	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	MessageBoxA
0040105B	56	PUSH ESI	Result = 1
0040105C	FF15 C8504000	CALL DWORD PTR DS:[<&USER32.MessageBoxA]	hWnd = NULL
00401062	6A 01	PUSH 1	EndDialog
00401064	56	PUSH ESI	kernel32.7697339A
00401065	FF15 CC504000	CALL DWORD PTR DS:[<&USER32.EndDialog]	
0040106B	5E	POP ESI	
0040106C	B8 01000000	MOV EAX, 1	Case 110 (WM_INITDIALOG) of switch 00401004
00401071	C2 1000	RETN 10	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL
00401074	> 8B4C24 04	MOV ECX, DWORD PTR SS:[ESP+4]	Title = "Nag!"
00401078	6A 40	PUSH 40	Text = "Oh, do u like this program? You are using the
0040107A	68 50514000	PUSH Nag2.00405150	hOwner = NULL
0040107F	68 00514000	PUSH Nag2.00405100	MessageBoxA
00401084	51	PUSH ECX	
00401085	FF15 C8504000	CALL DWORD PTR DS:[<&USER32.MessageBoxA]	
0040108B	B8 01000000	MOV EAX, 1	
00401090	C2 1000	RETN 10	

and we can see it's called from 401012, a JE instruction. Let's put a breakpoint on this and run the app:

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	

and we break on that JE instruction. Notice that it is not calling our nag screen. The reason for this is we happen to be in the middle of Window's message handler. I will be going into depth on message handlers in another tutorial, but for now just know that all GUI windows programs have a message handler and Windows sends various message through it. Depending on which message comes thru (and whether we

wish to do anything out of the ordinary (such as we can add our own code to override Window's normal routines. For example, when we click the 'X' to close a window, Windows will send a message through the message handler that says "hey, the user wants to close the window." We can either let the message go through, in which case Windows will handle it and close the window, or we can 'trap' this message and do what we want (maybe pop up a dialog that says "You have not saved, are you sure you want to quit?").

Our breakpoint happens to be right in the middle of this, so the first message that has come through does not match the message that this app expects to override in order to show the nag:

00401007	0F34 86000000	JE Nag2.00401093	
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	Default case of switch 00401004
0040101C	0FB74424 0C	MOVZX EAX,WORD PTR SS:[ESP+C]	Case 111 (WM_COMMAND) of switch 00401004
00401021	2D E9030000	SUB EAX,3E9	Switch (cases 3E9..3EA)
00401026	74 22	JE SHORT Nag2.0040104A	
00401028	48	DEC EAX	
00401029	75 75	JNZ SHORT Nag2.004010A0	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 Nag2.0040520C	Text = " KillNag \n I did this one fo
00401036	68 80514000	PUSH Nag2.00405180	hOwner = FFFFFFF20
0040103B	50	PUSH EAX	MessageBoxA
0040103C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	
00401042	B8 01000000	MOV EAX,1	
00401047	C2 1000	RETN 10	
0040104D	56	PUSH ESI	

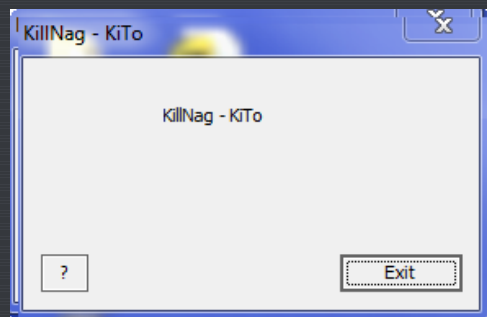
Go ahead and hit F9 to run the app and we will stop at the same BP, but this time, the jump will be taken, showing our nag. Let's tell Olly to not show the nag:

```
C 0 ES 002E
P 1 CS 002E
A 0 SS 002E
Z 0 DS 002E
S 0 FS 0053
T 0 GS 002E
D 0
O 0 LastErr
```

Now, if we leave this breakpoint, 34 more messages will be sent through this message handler. You can either keep the BP in place and click run 34 times (in which case, at some point you will see the window appear, the buttons being drawn etc) or you can remove the BP and just hit run once. In this case, the call is not made to the nag again so removing the BP and running it is fine:

00401004	83E8 10	SUB EAX,10	Switch (cases 10..111)
00401007	0F34 86000000	JE Nag2.00401093	
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	Default case of switch 00401004
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	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 Nag2.0040520C	Text = " Info KillNag \n I did this one fo
00401036	68 80514000	PUSH Nag2.00405180	hOwner = 00000017
0040103B	50	PUSH EAX	MessageBoxA
0040103C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	
00401042	B8 01000000	MOV EAX,1	
00401047	C2 1000	RETN 10	
0040104D	56	PUSH ESI	Nag2.00401000; 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 Nag2.00405178	Text = "Nag!"
00401052	50	PUSH EAX	

We then have our main screen:

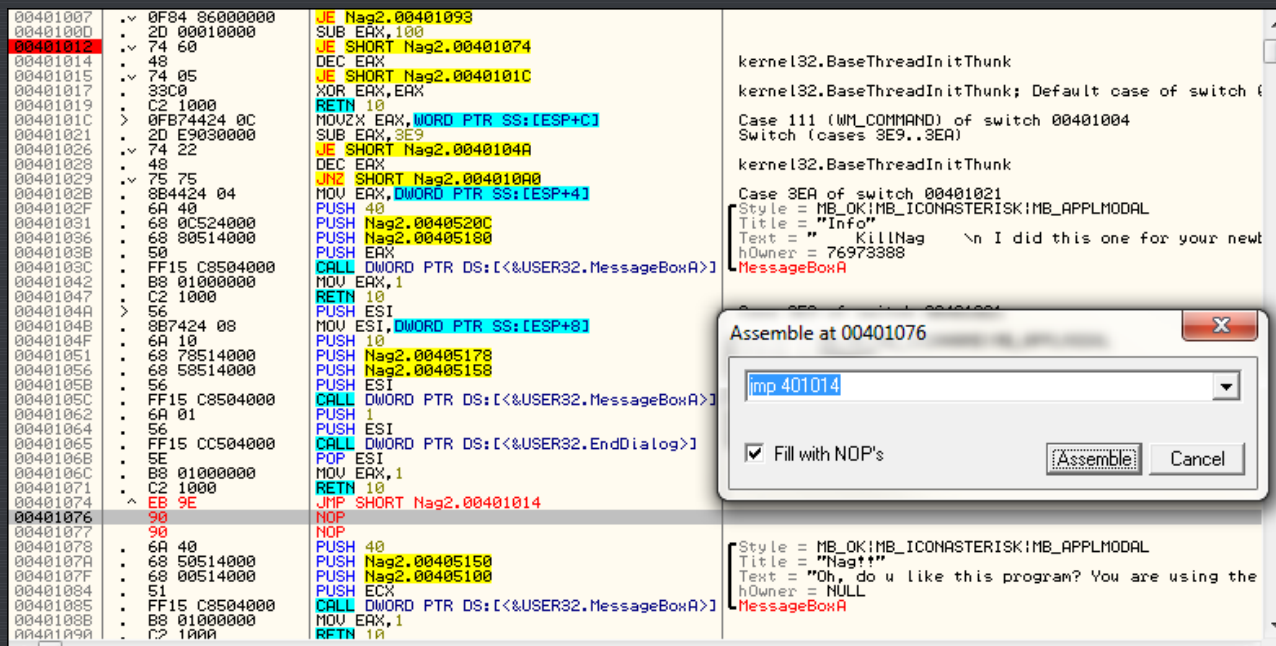


Notice that the initial nag is now gone.

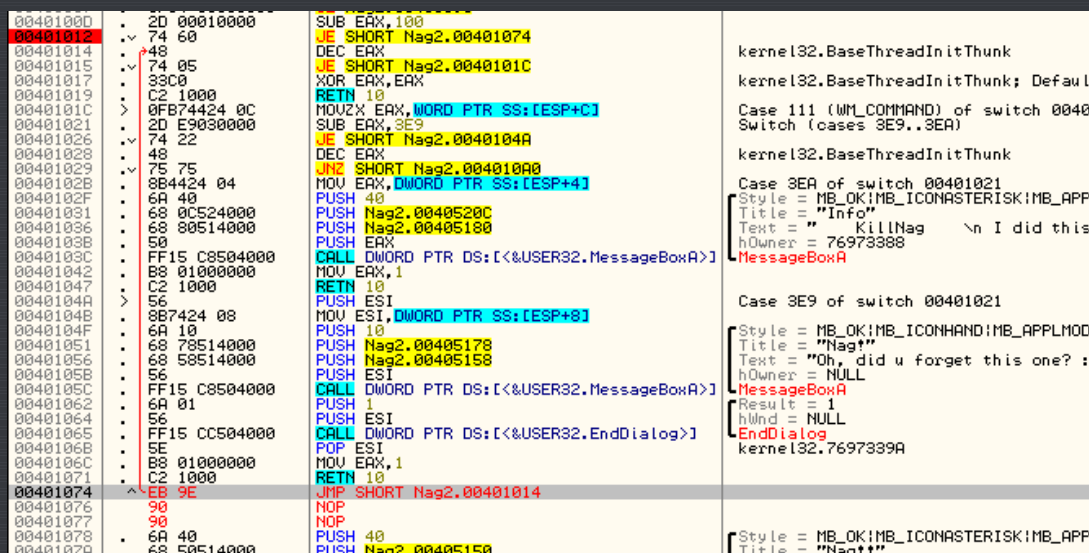
Patching the App

Normally what we would do is patch the JE instruction that jumps to the nag with a NOP so that it never

jumps, but I wanted to show you another way to accomplish this. We know that when the correct message comes through the message handler (in this case the second message) our nag code will be called. Well, what if we allowed the jump to the nag, but changed the nag to just jump right back again?



Here, the jump will be made to the nag instructions at 401074, but then we will immediately jump back to the line after the initial jump (401014). Basically, our program will jump, then jump right back to the next line:

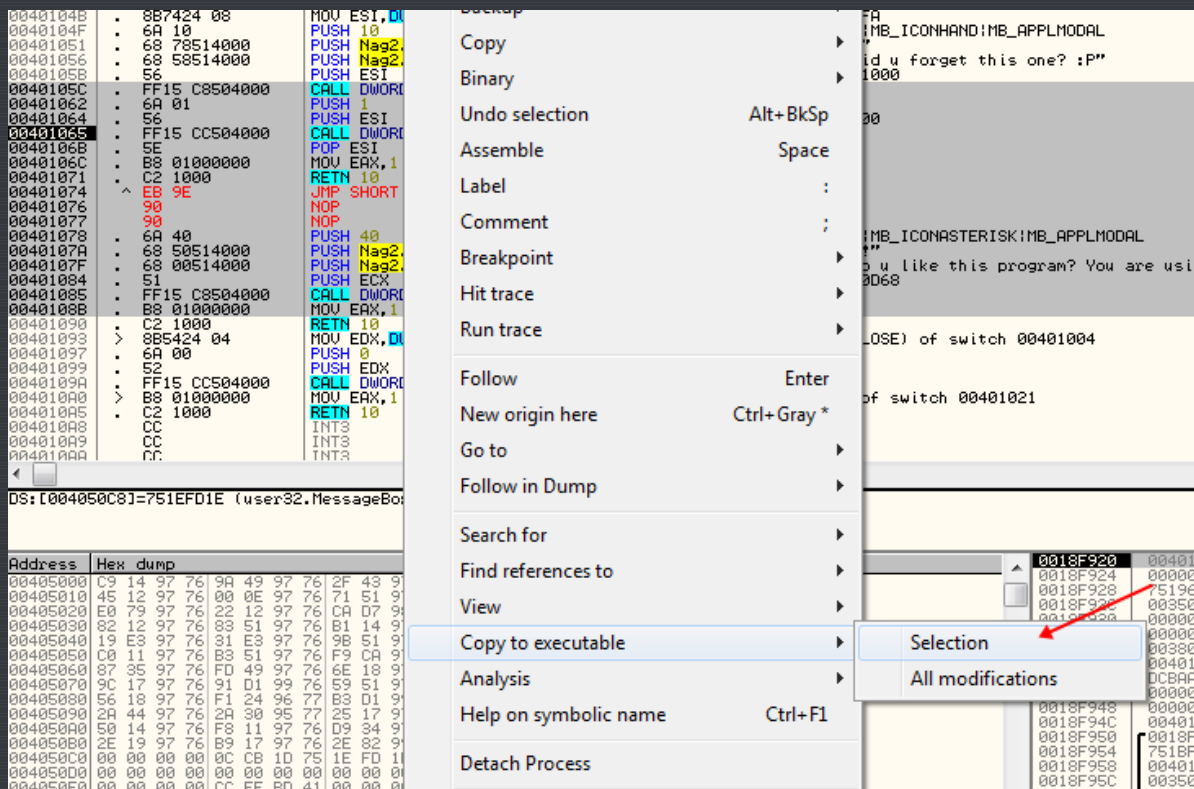


There is really no difference between NOPing the JE instruction at 401012 or adding a jump back at 401074, but I wanted you to start noticing that there are always multiple ways to patch- sometimes NOPing a call is not the best way. Remember, you OWN this binary- you can add whatever code you want, so don't be afraid to modify it, especially when learning.

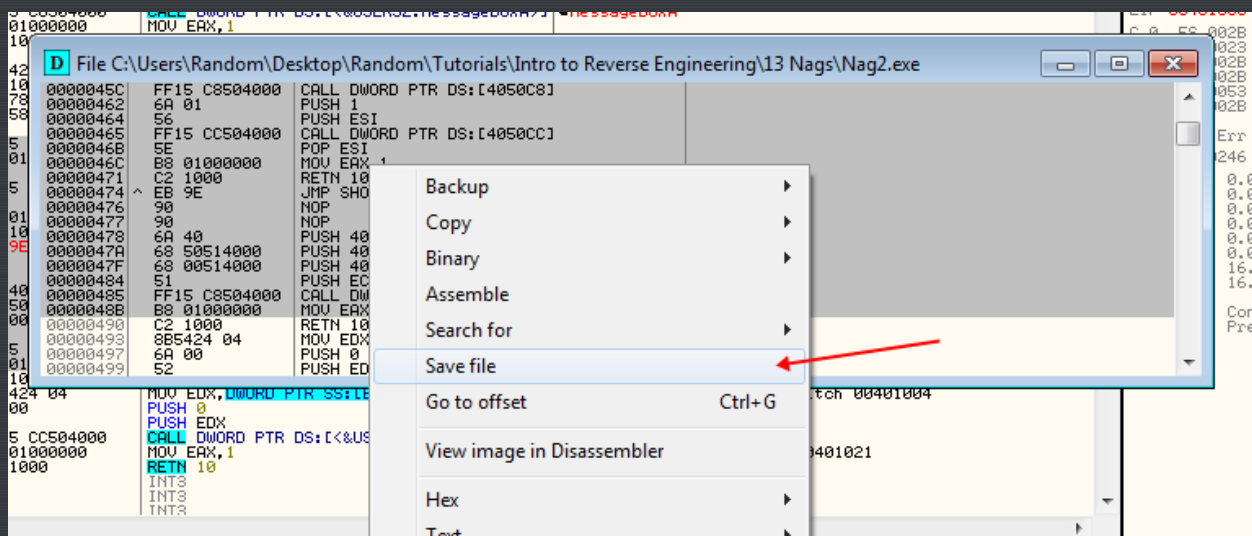
Running the app shows that the nag has been bypassed just the same:



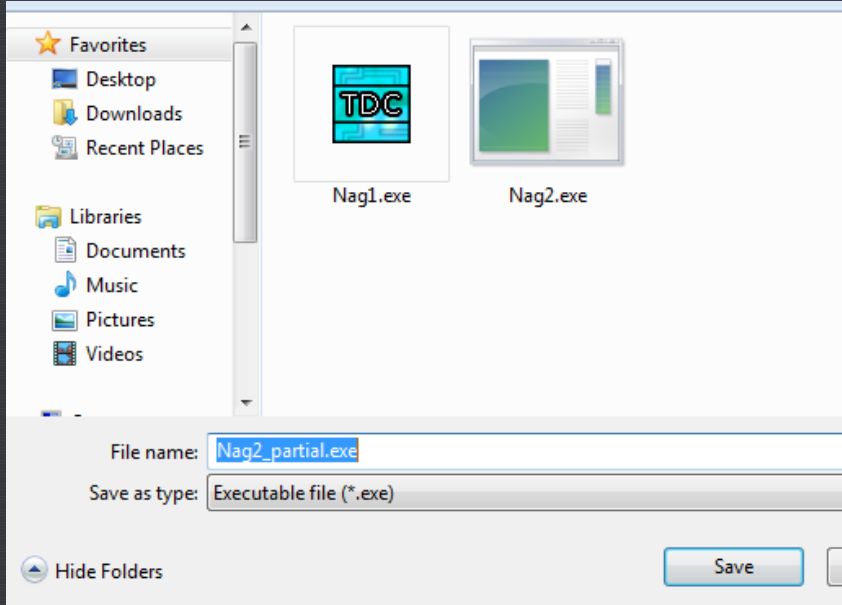
Now let's save the patch. Highlight the changed code (it's OK if you highlight more), and select "Copy to executable" -> "Selection":



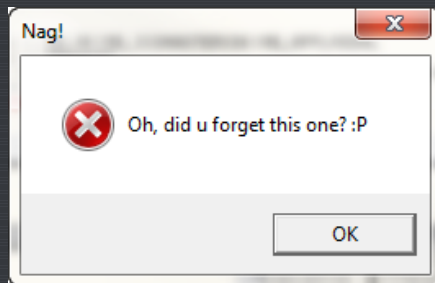
Then right click in this new window and select "Save file":



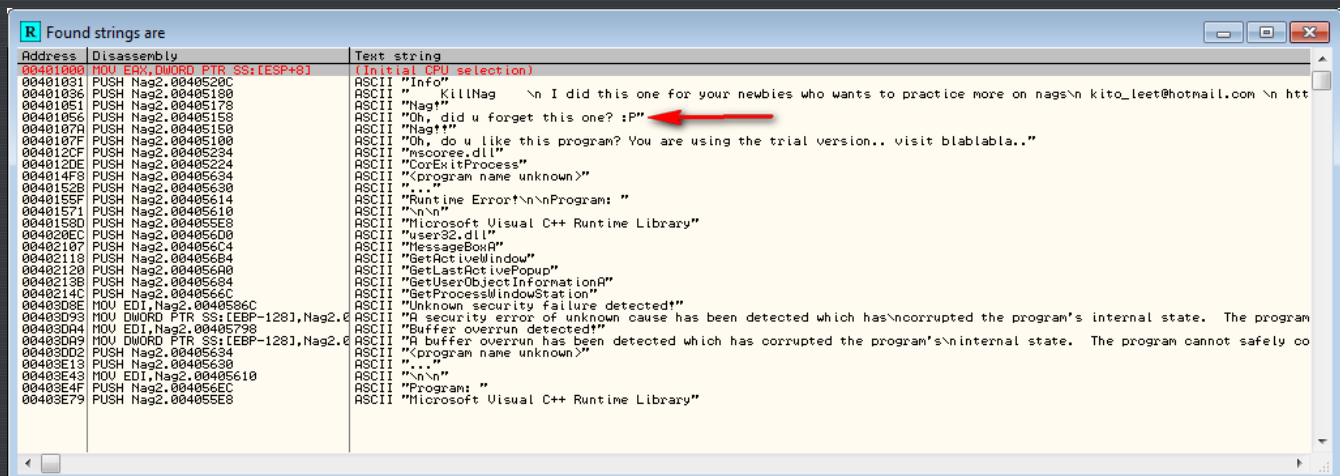
I saved it as a different name, in this case Nag2_partial.exe. You'll see why I called it partial in a minute:



OK. Go ahead and load this new patched program in Olly and let's try it out. We jump right to the main screen, so we know the patch worked. Now click exit and we get:



Uh oh. I guess the author was really determined here. Let's go find this second nag. Go back to strings and we can see that this nag's text is in there as well:



Many, many, many apps do this; they start with a nag, and after closing the app, they add another one. Most of the time, when searching for the first nag's text string, you will just automatically look for any others. Dbl-click on this text:

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]	
0040104F	6A 10	PUSH 10	
00401051	68 78514000	PUSH Nag2.00405178	
00401055	68 58514000	PUSH Nag2.00405158	
0040105B	56	PUSH ESI	
0040105C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
00401062	6A 01	PUSH 1	
00401064	56	PUSH ESI	
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 Nag2.00401014	
00401076	90	NOP	
00401077	90	NOP	
00401078	6A 40	PUSH 40	
0040107A	68 58514000	PUSH Nag2.00405150	
0040107F	68 00514000	PUSH Nag2.00405100	
00401084	51	PUSH ECX	
00401085	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
0040108B	B8 01000000	MOV EAX,1	
00401090	C2 1000	RETN 10	

and here we see the method for this nag. Clicking on the first line of it we can see that the second nag is being called right after the first nag was called, but it uses a different message to trigger it (probably a window destroy message). So when this message comes through, signaling that the user has selected "Exit", the second nag will be called.

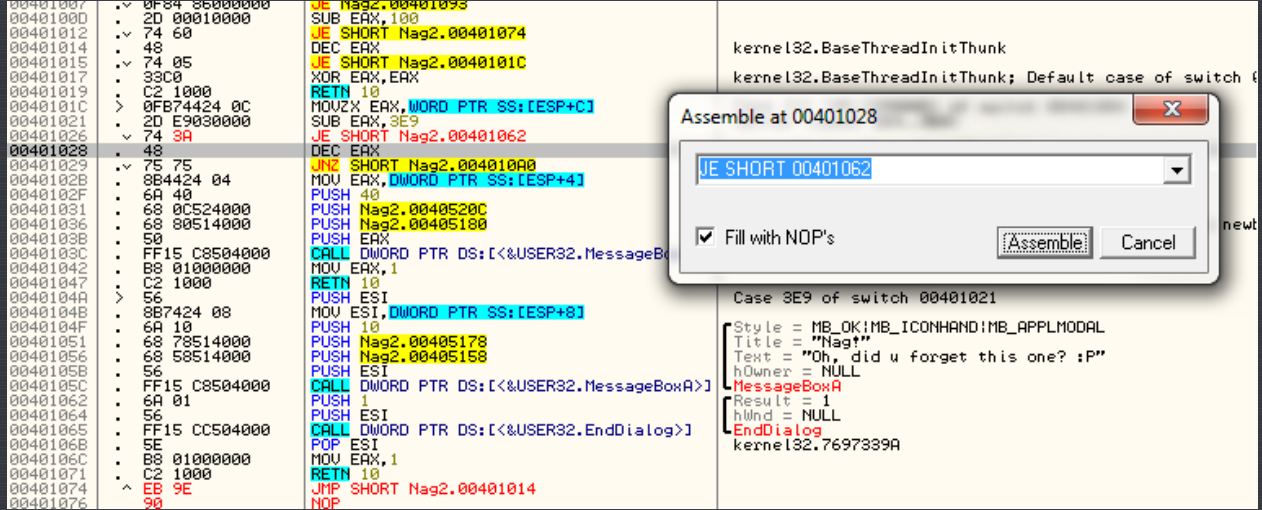
Your first thought may be "why don't we just put another jump in this one to jump right back like we did in the last one. Well, looking closely at this method, we can see that it calls the second nag, but then it immediately calls EndDialog. So jumping right back will not work as our dialog will never close:

00401017	33C0	XOR EAX,EAX	kernel32.BaseThreadInitThunk; Default case of switch
00401019	C2 1000	RETN 10	
0040101C	> 0FB74424 0C	MOVZX EAX,WORD PTR SS:[ESP+C]	Case 111 (WM_COMMAND) of switch 00401004
00401021	2D E9030000	SUB EAX,3E9	Switch (cases 3E9..3EA)
00401026	74 22	JE SHORT Nag2.0040104A	
00401028	48	DEC EAX	kernel32.BaseThreadInitThunk
00401029	75	JNZ SHORT Nag2.004010A0	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 = "Nag!"
00401031	68 0C524000	PUSH Nag2.0040520C	Text = "Oh, did u forget this one? :P"
00401036	68 80514000	PUSH Nag2.00405180	hOwner = NULL
0040103B	50	PUSH EAX	Result = 1
0040103C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
00401042	B8 01000000	MOV EAX,1	hWnd = NULL
00401047	C2 1000	RETN 10	EndDialog
0040104A	> 56	PUSH ESI	kernel32.7697339A
0040104B	8B7424 08	MOV ESI,DWORD PTR SS:[ESP+8]	
0040104F	6A 10	PUSH 10	
00401051	68 78514000	PUSH Nag2.00405178	
00401055	68 58514000	PUSH Nag2.00405158	
0040105B	56	PUSH ESI	
0040105C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	MessageBoxA
00401062	6A 01	PUSH 1	
00401064	56	PUSH ESI	
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 Nag2.00401014	
00401076	90	NOP	
00401077	90	NOP	
00401078	6A 40	PUSH 40	
0040107A	68 58514000	PUSH Nag2.00405150	

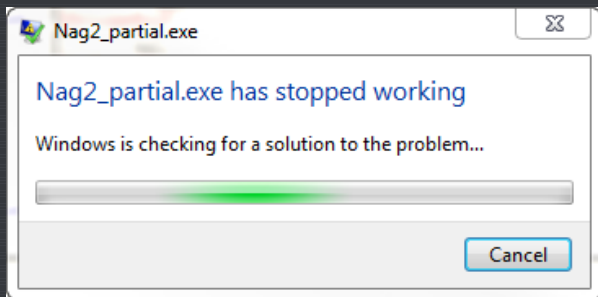
So your next thought might be, "let's just change the JE instruction at 401026 to jump to the EndDialog, jumping right over the nag MessageBoxA instruction." This is a good thought, so let's try it:

00401017	33C0	XOR EAX,EAX	kernel32.BaseThreadInitThunk; Default case of switch
00401019	C2 1000	RETN 10	
0040101C	> 0FB74424 0C	MOVZX EAX,WORD PTR SS:[ESP+C]	Case 111 (WM_COMMAND) of switch 00401004
00401021	2D E9030000	SUB EAX,3E9	Switch (cases 3E9..3EA)
00401026	74 22	JE SHORT Nag2.0040104A	kernel32.BaseThreadInitThunk
00401028	48	DEC EAX	Case 3EA of switch 00401021
00401029	75	JNZ SHORT Nag2.004010A0	Style = MB_OK;MB_ICONASTERISK;MB_APPLMODAL
0040102B	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]	Title = "Info"
0040102F	6A 40	PUSH 40	Text = "KillNag \n I did this one for your new"
00401031	68 0C524000	PUSH Nag2.0040520C	hOwner = 76973388
00401036	68 80514000	PUSH Nag2.00405180	MessageBoxA
0040103B	50	PUSH EAX	
0040103C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	Case 3E9 of switch 00401021
00401042	B8 01000000	MOV EAX,1	Style = MB_OK;MB_ICONHAND;MB_APPLMODAL
00401047	C2 1000	RETN 10	Title = "Nag!"
0040104A	> 56	PUSH ESI	Text = "Oh, did u forget this one? :P"
0040104B	8B7424 08	MOV ESI,DWORD PTR SS:[ESP+8]	hOwner = NULL
0040104F	6A 10	PUSH 10	Result = 1
00401051	68 78514000	PUSH Nag2.00405178	hWnd = NULL
00401055	68 58514000	PUSH Nag2.00405158	EndDialog
0040105B	56	PUSH ESI	kernel32.7697339A
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 58514000	PUSH Nag2.00405150	

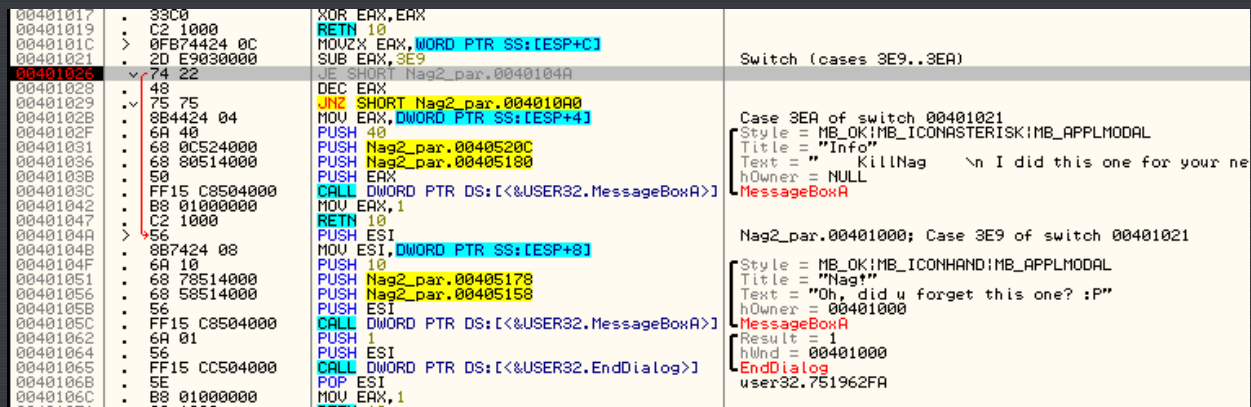
Change the JE instruction at 401026 to jump to 401062 instead, jumping to the first line of EndDialog:



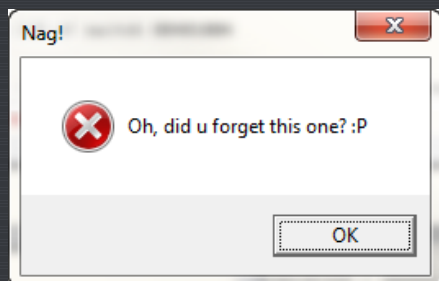
and run the app:



Well, that doesn't look too promising. So we obviously did something wrong. Here's what we're going to do; let's run the app without our patch, stepping through it, and see what it does, then run it with the patch and see how they are different. Re-start the app and click "Exit" and we will break at our patch (which is gone now that we re-started the app):



Step a couple lines and when you step over the call to `MessageBoxA` you will see the nag:



Now step two more times until we are on our call to `EndDialog`:

00401047	> 52 1000	RETN 10	Case 3E9 of switch 00401021
00401048	• 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	

and let's look at the stack. We can see that there are four items on the stack; a handle to our window, the result of the end dialog, a pointer to the first line of our code (401000), and a return address to user32.

0018F9C0	00111166	hWnd = 00111166 ('KillNag - KiTo',class='#32770')
0018F9C4	00000001	Result = 1
0018F9C8	00401000	Nag2_par.00401000
0018F9CC	751962FA	RETURN to user32.751962FA
0018F9D0	00111166	
0018F9D4	00000111	
0018F9D8	000003E9	
0018F9DC	000B1124	
0018F9E0	00401000	Nag2_par.00401000
0018F9E4	DCBAABCD	
0018F9E8	00000001	
0018F9EC	00000000	
0018F9F0	00401000	Nag2_par.00401000
0018F9F4	0018F9CC	
0018F9F8	751BF943	RETURN to user32.751BF943 from user32.751962D7
0018F9FC	00401000	Nag2_par.00401000
0018FA00	00111166	
0018FA04	00000111	
0018FA08	000003E9	
0018FA0C	000B1124	
0018FA10	0E1328D0	
0018FA14	00000000	
0018FA18	00000111	

Now restart the app, and when we get to our patch, activate it (patches window, hit space bar with it selected):

00401026	74 3A	JE SHORT Nag2_par.00401062	Case 3E9 of switch 00401021
00401028	48	DEC EAX	Style = MB_OK;MB_ICONASTERISK;MB_APPLMODAL
00401029	75 75	JNZ SHORT Nag2_par.004010A0	Title = "Info"
0040102B	8B4424 04	MOV EAX, DWORD PTR SS:[ESP+4]	Text = "KillNag \n I did this one for your new"
0040102F	6A 40	PUSH 40	hOwner = NULL
00401031	68 0C524000	PUSH Nag2_par.0040520C	MessageBoxA
00401036	68 80514000	PUSH Nag2_par.00405180	
0040103B	50	PUSH EAX	
0040103C	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	Nag2_par.00401000; Case 3E9 of switch 00401021
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	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	
00401071	C2 1000	RETN 10	Case 110 of switch 00401004
00401074	EB 9E	JMP SHORT Nag2_par.00401014	
00401076	90	NOP	

Address	Size	State	Old	New
00401026	2	Active	JE SHORT Nag2_par.0040104A	JE SHORT Nag2_par.00401062

Now we will jump over the message box nag call. Step until you get to the EndDialog call:

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	Nag2_par.00401000
0040106C	• B8 01000000	MOV EAX, 1	
00401071	• C2 1000	RETN 10	
00401074	> EB 9E	JMP SHORT Nag2_par.00401014	Case 110 of switch 00401004

and let's look at our stack. We have the handle to the window, the result code, and a return to user32. We are missing the pointer to the first line of our code at 401000!!!

```

00401000 hWin = 00401000
001BF924 RETURN to user32.751962FA
001BF924 751962FA RETURN to user32.751962FA
001BF924 000011AA
001BF924 00000111
001BF924 000003E9
001BF938 000B1148
001BF93C 00401000 Nag2_par.00401000
001BF940 DCBABCDC
001BF944 00000001
001BF948 00000000
001BF94C 00401000 Nag2_par.00401000
001BF950 001BF9CC
001BF954 751BF943 RETURN to user32.751BF943 from user32.751962D7
001BF958 00401000 Nag2_par.00401000
001BF95C 000E11AA
001BF960 00000111
001BF964 000003E9
001BF968 000B1148
001BF96C 58F6A79B
001BF970 00000000
001BF974 00000111
001BF978 00A377C0

```

If you scroll up and take a look at the call to the second nag, you will notice that before the message box is created, ESI is pushed onto the stack. This is a pointer to our code. It just so happens that this program pushes it before calling message box, though it could have been done after. So we are missing an important push that the app needs in order to run EndDialog properly. The problem is we have some initialization code we want, then a call to a nag we don't want, then a call to EndDialog that we do want:

0040101C	> 0FB74424 0C	MOVZX EAX,WORD PTR SS:[ESP+0]	
00401021	2D E9030000	SUB EAX,3E9	
00401026	74 3A	JE SHORT Nag2_par.00401062	Switch (cases 3E9..3EA)
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 = "KillNag \n I did this one for your new"
0040103B	50	PUSH EAX	hOwner = NULL
0040103E	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]	user32.751962FA
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	hWin = 00401000
00401065	FF15 CC504000	CALL DWORD PTR DS:[&USER32.EndDialog]	EndDialog
0040106B	5E	POP ESI	Nag2_par.00401000
0040106C	B8 01000000	MOV EAX,1	
00401071	C2 1000	RETN 10	
00401074	EB 9E	JMP SHORT Nag2_par.00401014	Case 110 of switch 00401004

Well, let's get rid of the code we don't want. Highlight the MessageBoxA instructions (from 40104F to 40105C) and right click. Select "Binary" -> "fill with NOPS":

0040102B	6A 40	PUSH 40	Case 3EA of switch 00401021
00401031	68 0C524000	PUSH Nag2_par.0040520C	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL
00401036	68 80514000	PUSH Nag2_par.00405180	Title = "Info"
0040103B	50	PUSH EAX	Text = "KillNag \n I did this one for your new"
0040103E	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	hOwner = NULL
00401042	B8 01000000	MOV EAX,1	MessageBoxA
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	Backup
00401051	68 78514000	PUSH Nag2_par.00405178	Copy
00401056	68 58514000	PUSH Nag2_par.00405158	Binary
0040105B	56	PUSH ESI	Space
0040105C	FF15 CC504000	CALL DWORD PTR DS:[&USER32.EndDialog]	Label
00401062	6A 01	PUSH 1	Comment
0040106C	B8 01000000	MOV EAX,1	Breakpoint
00401071	C2 1000	RETN 10	Hit trace
00401074	EB 9E	JMP SHORT Nag2_par.00401014	Run trace
00401076	90	NOP	
00401077	90	NOP	
00401078	6A 40 68 50 51 40	ASCIZ "j0hP00",0	
0040107F	51	PUSH EAX	
00401084	51	PUSH EAX	
00401085	FF15 C8504000	CALL DWORD PTR DS:[&USER32.MessageBoxA]	
0040108B	B8 01000000	MOV EAX,1	
00401090	C2 1000	RETN 10	
00401093	> 8B5424 04	MOV EDI,DWORD PTR SS:[ESP+4]	
00401097	6A 00	PUSH 0	
00401099	52	PUSH EDI	

And bam! no more call to our nag:

00401021	. 20 E9030000	SUB EAX,3E9	Switch (cases 3E9..3EA)
00401026	74 22	JE SHORT Nag2_par.0040104A	
00401028	48	DEC EAX	
00401029	75 75	JNE 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 newb
00401038	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	
00401056	90	NOP	Text
00401057	90	NOP	
00401058	90	NOP	
00401059	90	NOP	
0040105A	90	NOP	
0040105B	90	NOP	hOwner
0040105C	90	NOP	MessageBoxA
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
00401068	5E	POP ESI	user32.751962FA
0040106C	B8 01000000	MOV EAX,1	

Now when you run the app, you will notice that the app closes normally. You can now save this patch and there will be no nags left 😊 .

-Till next time

R4ndom