

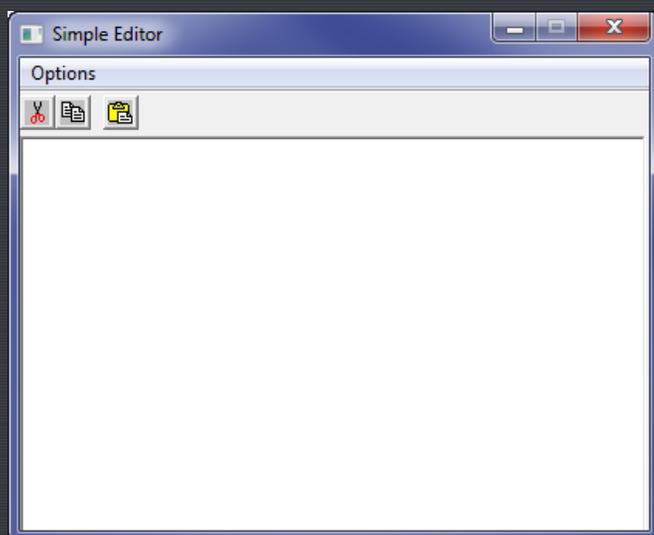
## Modifying Binaries: Adding a Menu Item

by R4ndom on Jun.21, 2012, under Intermediate, Tutorials

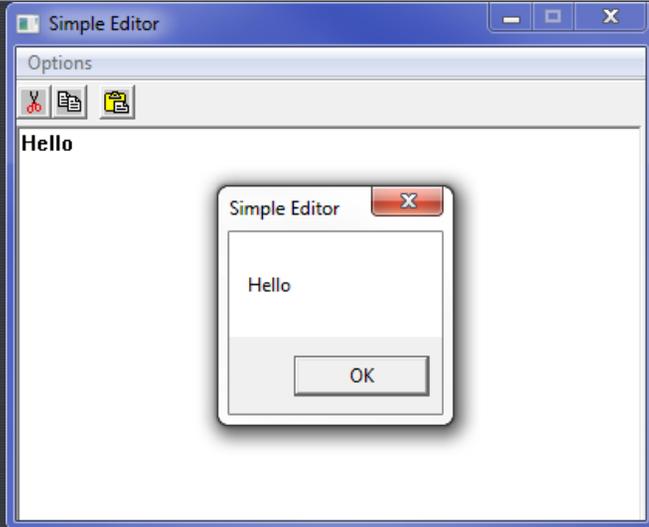
### Introduction

In this tutorial we are going to be adding a menu item to a binary. There are many uses for adding a menu item to an existing binary. Perhaps you find yourself typing the same phrase over and over – it may be a lot easier just to make it a menu item that when selected, pasted that phrase into your app. Maybe you would like to add a little sunshine to a co-workers day by adding a “UNDO” menu item, that instead deletes the file, reminding him or her of the importance of not taking the undo button for granted. There are many reasons for adding a menu item, but most of all, it's just interesting how it works 😊

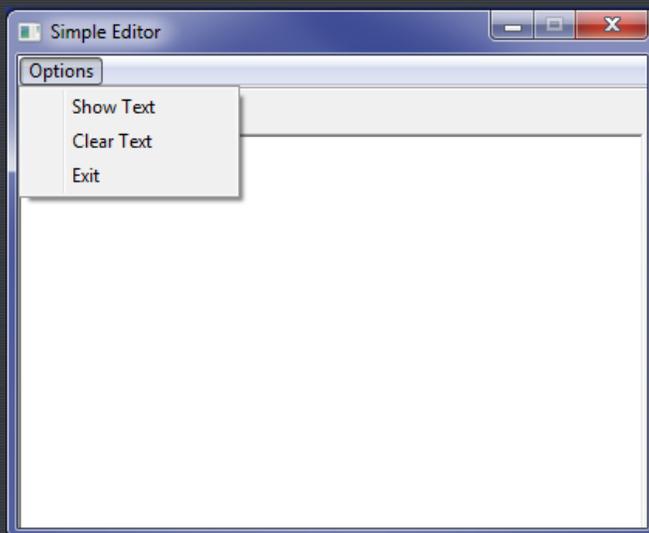
Our target will be a simple app I made for my regular reverse engineering tutorials called Simple Editor. It doesn't do much, but it helps showing how this is done on a simple app so you don't get bogged down in superfluous code. Go ahead and run the app:



You can see that there's not much to it (and I'm pretty sure the toolbar buttons are wrong, but it doesn't matter for this tutorial. Trust me, I'll fix it before the 'Adding Toolbar Buttons tutorial' 😊)The Simple Editor basically allows you to do three things. One is to repeat what you typed in a message box:



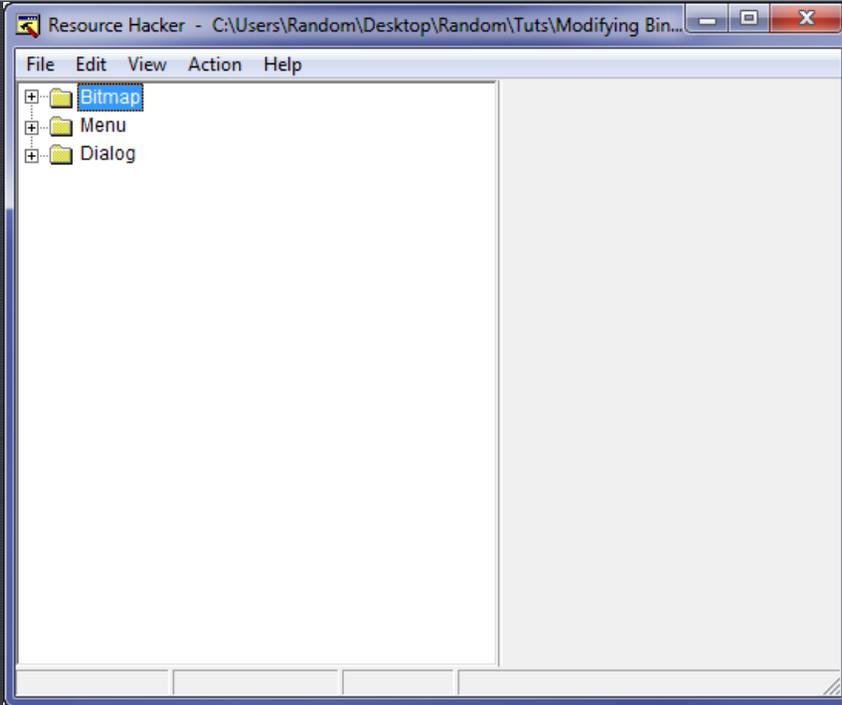
The other two things are clear the text window and exit:



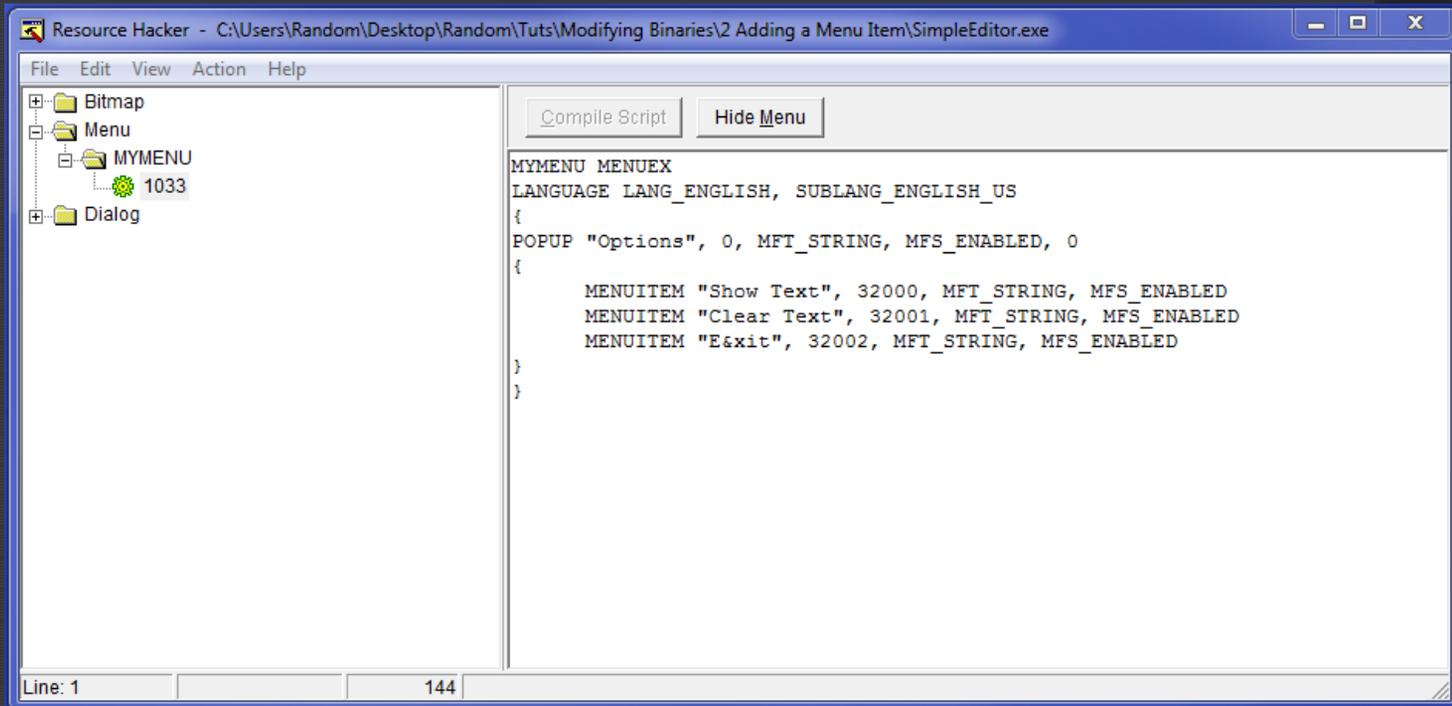
So what we are going to do is add a menu item to this app called "Shout Out!!!" that will insert a helpful message into the text area.

## Adding The Resource

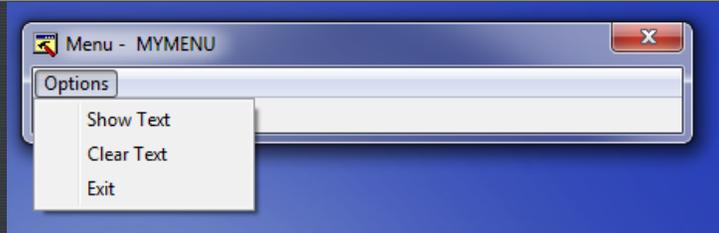
First, load the Simple Editor into Resource Hacker (Resource Hacker can be downloaded from my [tools](#) page) and you will see the three resources it has found in our program:



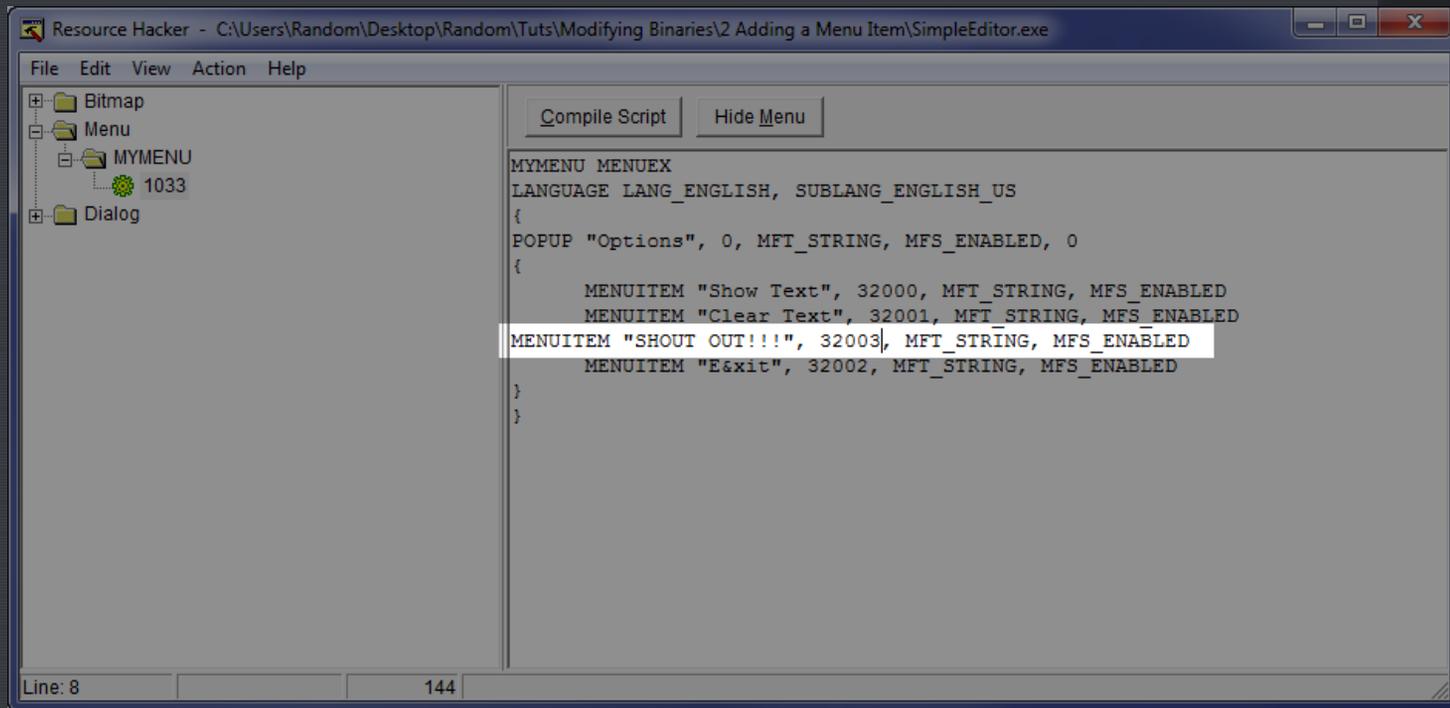
Resource Hacker has found a Bitmap (the toolbar), the menu, and a dialog box, which is our main app's window. Go ahead and open the menu folder, open the "Mymenu" folder and click on the 1033 with the pretty flower next to it:



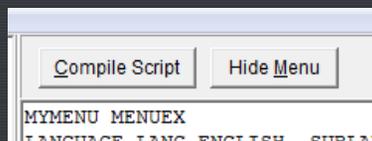
As you can see, this opens the various characteristics of the menu in our app as well as a sample window showing the menu below. If you click on the options menu in the example window, it will show you what it will look like:



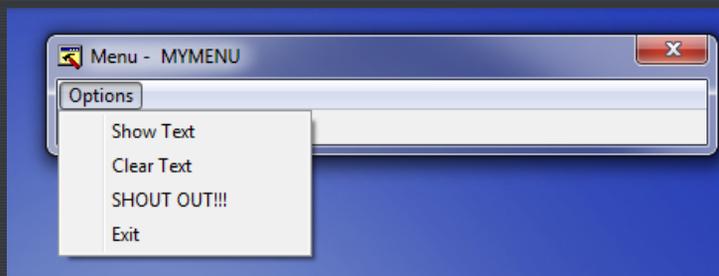
Now, move up to the main window and look at the menu data. Here you can see that the name of the menu is Mymenu, the menu is in English (duh), and the information about this specific menu items. First is the POPUP "Options" telling us that this is the title of the menu and will pop-up additional items. Next are three menu items, each corresponding to the three item that are currently in our menu. What we are going to do is very simple: we are going to add (copy and paste is fine) a line with our own menu item in it. This line must contain all of the same data as the other item except it should have it's own name and a unique code:



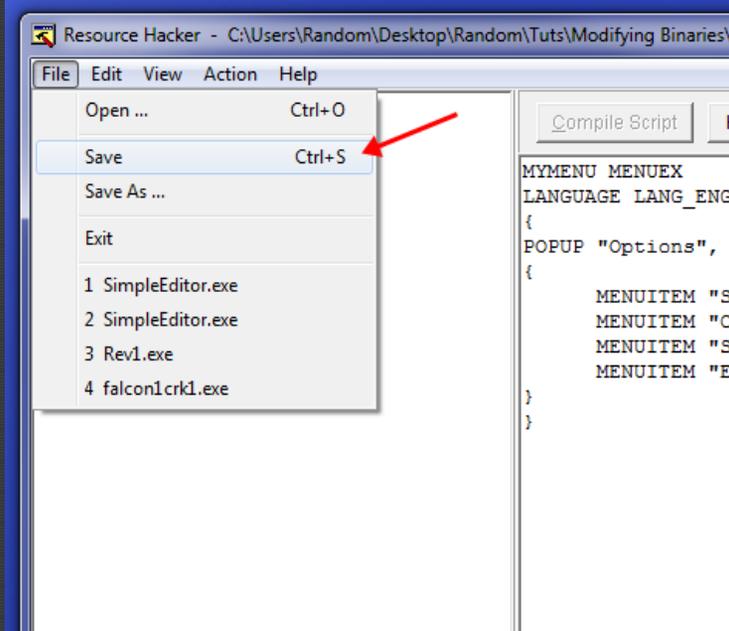
In this case, I have added an item called "SHOUT OUT!!!". I can see that the codes 32000, 32001, and 32002 have been taken so I will pick 32003. These will be VERY important later, so make sure you at least notice them. At the end of the line are some attributes telling the app we want to see it as text blah blah blah. Now, click on the "Compile Script" button at the top:



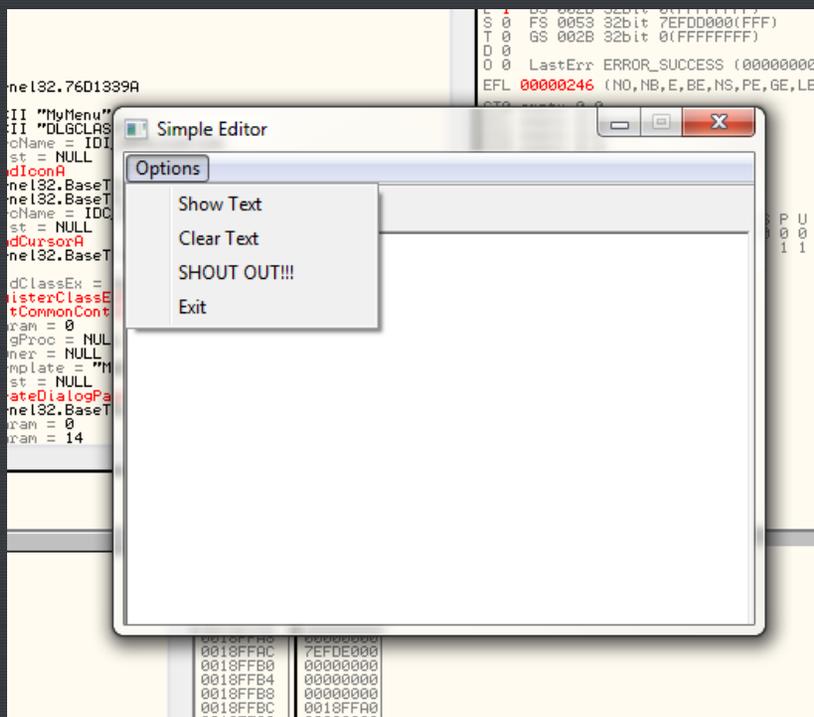
And now if you go to the example window and click on the Options menu, you will see our new menu item!



We have altered this binary so we must save it with this new resource in it, so select File -> Save (or Save As if you're not the daring type). I saved it as SimpleEditor2, but then again, I'm not the daring type:



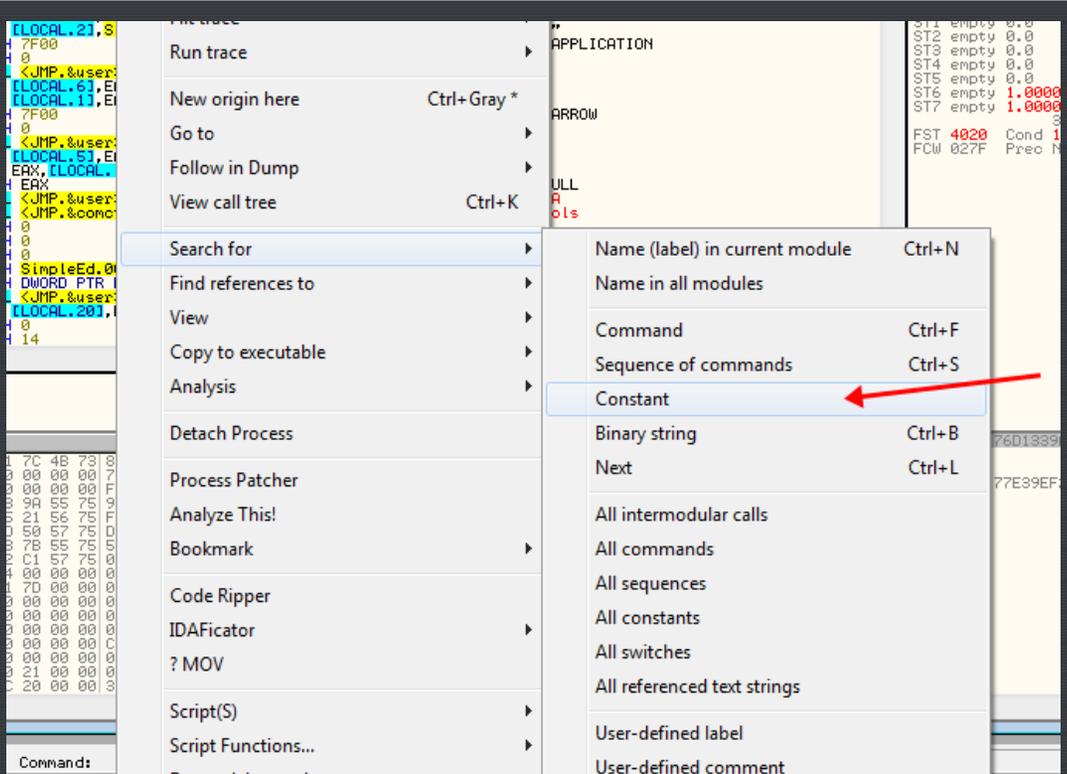
Now don't go and think "I've done it! I a l33t h4k0r now!!!". All you've done is added a menu resource to the binary. The binary has no idea what to actually do with it, and will show you this as soon as you run it and choose our menu item:



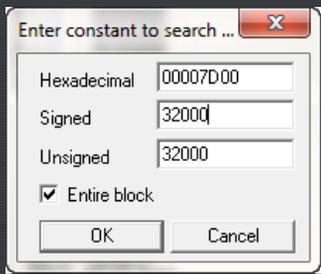
I'll break the suspense; it does nothing. That is because when this program was written, the author never took into consideration a SHOUT OUT!!! button so never told the program what to do if a user selected it. Now begins the hard part: we must tell the program what we want to do.

## Finding The Menu Compare Instructions

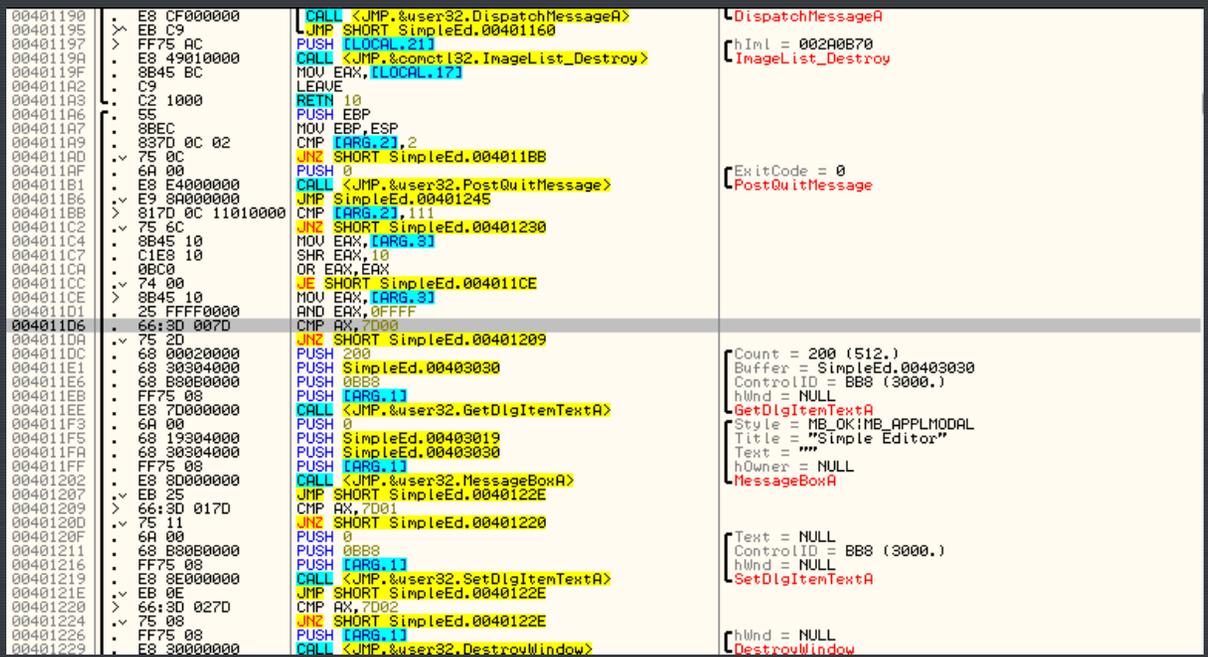
Fortunately, we are dealing with a fairly small app, and fortunately the ID's that were used for the menu resources were quite unique (thank me later). Most of the time, the app will be far larger and the IDs will be 0, 1, 2 etc. The reason we are glad that these IDs are unique is that we can search directly for them and not hit too many false positives (actually, in our case, none). So remember back to those IDs that were in the menu section and you will remember that the first one was 32000. Let's search the binary for this constant as we know somewhere the app will need to compare something against this ID to see if that was the menu option selected (and not say, the exit menu item). Open the app in OllyDBG (you can get this on my [tools](#) page as well), Right click in the assembly window and select "Search for" -> Constant:



Now, this 32000 is in decimal, so I put it into the Signed (or Unsigned) decimal edit box, make sure "Entire block" is selected and choose OK:



Fortunately, Olly has taken us directly to the section of code we are looking for:



Notice the compare AX, 7D00? 7D00 is hex for 32000. Every Windows program will have something that

resembles this grouping of code. Well, there's a menu (and that's not 't). Sometimes it will not be easy to find, especially if the program was coded in something other than C/C++, but the more you do this the more you will grow to recognize this section.

A second option is to simply run the app, select a menu item that you can figure out an API call for, put a BP on every call to that API, and eventually you will find your way to this area.

This area I'm referring to is the section of code that checks which menu item was selected. If you look above where Olly has stopped, you will see that if in fact AX is equal to 32000, it will fall through to a GetDlgItem and a MessageBox. This should signal to you that this is the "Show text" menu option, as it grabs the text that has been typed and shows it in a message box. You could also just remember that the code 32000 corresponded to the "Show text" menu item resource in Resource Hacker. If you look a little past that you will see that AX is then compared with 7D01 (32001), and if it's equal we call the SetDlgItemTextA (clearing the screen), and if it's not we compare AX with 32002 (7D02) which will then call the DestroyWindow API:

|          |                   |   |  |
|----------|-------------------|---|--|
| 004011C4 | . 8B45 10         | MOV EAX, [ARG_3]                                |  |
| 004011C7 | . C1E8 10         | SHR EAX, 10                                     |  |
| 004011CA | . 0BC0            | OR EAX, EAX                                     |  |
| 004011CC | > 74 00           | JE SHORT SimpleEd.004011CE                      |  |
| 004011CE | > 8B45 10         | MOV EAX, [ARG_3]                                |  |
| 004011D1 | . 25 FFF000       | AND EAX, 0FFFF                                  |  |
| 004011D6 | > 66:3D 007D      | CMP AX, 7D00                                    |  |
| 004011DA | > 75 2D           | JNZ SHORT SimpleEd.00401209                     |  |
| 004011DC | . 68 00020000     | PUSH 200  | Count = 200 (512.)<br>Buffer = SimpleEd.00403030<br>ControlID = BB8 (3000.)<br>hWnd = NULL<br><b>GetDlgItemTextA</b> |
| 004011E1 | . 68 30304000     | PUSH SimpleEd.00403030                          |  |
| 004011E6 | . 68 B80B0000     | PUSH BB8  |  |
| 004011EB | . FF75 08         | PUSH [ARG_1]                                    |  |
| 004011EE | . E8 7D000000     | CALL <JMP.&user32.GetDlgItemTextA>              |  |
| 004011F3 | . 6A 00           | PUSH 0  |  |
| 004011F5 | . 68 19304000     | PUSH SimpleEd.00403019                          | Style = MB_OK   MB_APPLMODAL<br>Title = "Simple Editor"<br>Text = ""<br>hWnd = NULL<br><b>MessageBoxA</b>            |
| 004011FA | . 68 30304000     | PUSH SimpleEd.00403030                          |  |
| 004011FF | . FF75 08         | PUSH [ARG_1]                                    |  |
| 00401202 | . E8 8D000000     | CALL <JMP.&user32.MessageBoxA>                  |  |
| 00401207 | > EB 25           | JMP SHORT SimpleEd.0040122E                     |  |
| 00401209 | > 66:3D 017D      | CMP AX, 7D01                                    |  |
| 0040120D | > 75 11           | JNZ SHORT SimpleEd.00401220                     |  |
| 0040120F | . 6A 00           | PUSH 0  | Text = NULL<br>ControlID = BB8 (3000.)<br>hWnd = NULL<br><b>SetDlgItemTextA</b>                                      |
| 00401211 | . 68 B80B0000     | PUSH BB8  |  |
| 00401216 | . FF75 08         | PUSH [ARG_1]                                    |  |
| 00401219 | . E8 8E000000     | CALL <JMP.&user32.SetDlgItemTextA>              |  |
| 0040121E | > EB 0E           | JMP SHORT SimpleEd.0040122E                     |  |
| 00401220 | > 66:3D 027D      | CMP AX, 7D02                                    |  |
| 00401224 | > 75 08           | JNZ SHORT SimpleEd.0040122E                     |  |
| 00401226 | . FF75 08         | PUSH [ARG_1]                                    | hWnd = NULL<br><b>DestroyWindow</b>  |
| 00401229 | . E8 30000000     | CALL <JMP.&user32.DestroyWindow>                |  |
| 0040122E | > EB 15           | JMP SHORT SimpleEd.00401245                     |  |
| 00401230 | > FF75 14         | PUSH [ARG_1]                                    | lParam = 1901E3H<br>wParam = 0<br>Message = MSG_FFFFFFFFF<br>hWnd = NULL<br><b>DefWindowProcA</b>                    |
| 00401233 | . FF75 0C         | PUSH [ARG_2]                                    |  |
| 00401236 | . FF75 08         | PUSH [ARG_1]                                    |  |
| 00401239 | . E8 17000000     | CALL <JMP.&user32.DefWindowProcA>               |  |
| 00401241 | . C9              | LEAVE   |  |
| 00401242 | . C2 1000         | RETN 10   |  |
| 00401245 | > 33C0            | XOR EAX, EAX                                    |  |
| 00401247 | . C9              | LEAVE   |  |
| 00401248 | . C2 1000         | RETN 10   |  |
| 0040124B | . CC              | JMPS  |  |
| 0040124C | \$. FF25 14204000 | JMP DWORD PTR DS:[<&gdi32.DeleteObject>]        | gdi32.DeleteObject   |
| 00401252 | \$. FF25 74204000 | JMP DWORD PTR DS:[<&user32.CreateDialogParamA>] | user32.CreateDialogParamA  |
| 00401258 | \$. FF25 70204000 | JMP DWORD PTR DS:[<&user32.DefWindowProcA>]     | ntdll.12.NtdllDefWindowProc_A  |

It's pretty easy to tell from this picture that there are three menu items and three sections of code corresponding to each of them. That seems pretty perfect, seeing as our app originally contained three menu items, but what if we want four? Well, that's where the code cave comes in.

## Finding The Code Cave

The problem is we can't just go adding another compare and API call for our new menu item in the middle of all this code. I mean, this code does shit- you can't just write over it! So what we are going to have to do is call our own code from somewhere in this section of code, and our code needs to check our own menu item ID (32003) to see if it was selected. The problem is compounded by the fact that when we put a jump to our custom code, we will have to overwrite some instructions to make the jump, so those instructions that were overwritten will need to be moved to our cave as well. If you are a little hazy on code caves, please see my tutorial "Adding a Dialog Box" which goes over the basics of them.

Well, the first thing we need to do is find some space to put our code. Normally I would use the Find Code Cave script used in the previously mentioned tutorial, but since this app is so small, you only need to scroll down about a page (right past the jump table) and you'll hit a bunch of empty memory:

```

0040120C $- FF25 00204000 JMP DWORD PTR DS:[&conct132.ImageList_Add] conct132.ImageList_Add
004012E2 $- FF25 0C204000 JMP DWORD PTR DS:[&conct132.ImageList_Create] conct132.ImageList_Create
004012E8 $- FF25 08204000 JMP DWORD PTR DS:[&conct132.ImageList_Destroy] conct132.ImageList_Destroy
004012EE $- FF25 04204000 JMP DWORD PTR DS:[&conct132.InitCommonControls] conct132.InitCommonControls
004012F4 00 DB 00
004012F5 00 DB 00
004012F6 00 DB 00
004012F7 00 DB 00
004012F8 00 DB 00
004012F9 00 DB 00
004012FA 00 DB 00
004012FB 00 DB 00
004012FC 00 DB 00
004012FD 00 DB 00
004012FE 00 DB 00
004012FF 00 DB 00
00401300 00 DB 00
00401301 00 DB 00
00401302 00 DB 00
00401303 00 DB 00
00401304 00 DB 00
00401305 00 DB 00
00401306 00 DB 00
00401307 00 DB 00
00401308 00 DB 00
00401309 00 DB 00
0040130A 00 DB 00
0040130B 00 DB 00
0040130C 00 DB 00
0040130D 00 DB 00
0040130E 00 DB 00
0040130F 00 DB 00
00401310 00 DB 00
00401311 00 DB 00
00401312 00 DB 00
00401313 00 DB 00
00401314 00 DB 00
00401315 00 DB 00
00401316 00 DB 00
00401317 00 DB 00

```

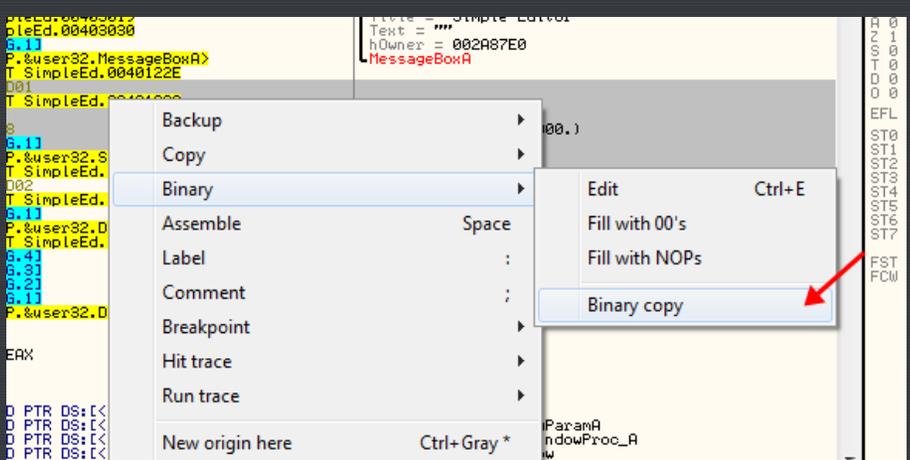
So our code cave will start at address 4012F5 (This skips one byte of zero, but I like to have a little buffer anyway 😊). Now we need to scroll up and find out where we can make the jump to our code cave, a place where it will be easy to fix what we overwrite. Since the first check (for 'Show Text') is longer than the others, I'm gonna leave that one. That leaves either the second (for 'Clear Text') or the third (for 'Exit'). It really doesn't matter which of the two you pick, but I picked the second. As we saw earlier, this second section of code starts at address 401209 with the initial compare to AX, and ends at address 40121E, with a jump out of this section after the call to SetDlgItemTextA:

```

004011FA . 83 30304000 PUSH SIMPLEEd.00403030
004011FF . FF75 08 CALL [ARG_1]
00401202 . E9 3D000000 CALL <JMP.&user32.MessageBoxA>
00401207 . EB 25 JMP SHORT SimpleEd.0040122E
00401209 > 66:3D 017D CMP AX,7D01
0040120D . 75 11 JNZ SHORT SimpleEd.00401220
0040120F . 6A 00 PUSH 0
00401211 . 68 B80B0000 PUSH 0BB8
00401216 . FF75 08 CALL [ARG_1]
00401219 . E9 3D000000 CALL <JMP.&user32.SetDlgItemTextA>
0040121E . EB 0E JMP SHORT SimpleEd.0040122E
00401220 > 66:3D 027D CMP AX,7D02
00401224 . 75 08 JNZ SHORT SimpleEd.0040122E
00401226 . FF75 08 CALL [ARG_1]
00401229 . E9 3D000000 CALL <JMP.&user32.DestroyWindow>

```

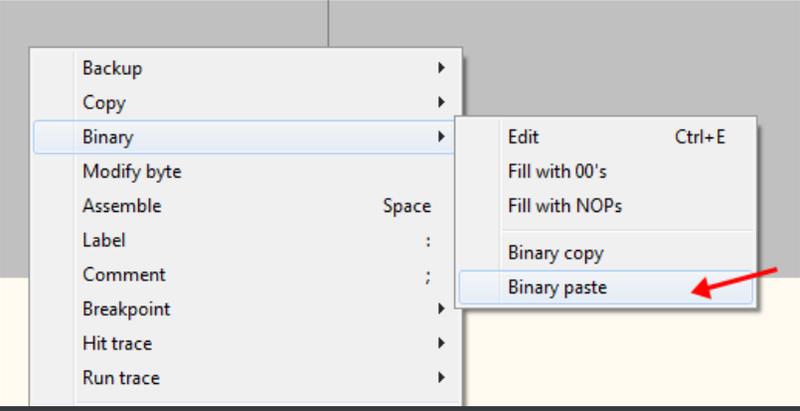
Now what I'm going to do is copy this entire section of code and put it at the beginning of our code cave. I am taking the whole thing because it is not very big and that way I don't need to worry about saving registers and stack state. This section of code will be completely contained in our code cave, along with our additional code. So highlight EXACTLY the lines I have highlighted in the previous picture, right click on it and choose "Binary" -> "Binary copy":



Now scroll down to our code cave, start highlighting the line at address 4012F5 and highlight a large section (enough to fit the copied code):

|          |               |   |                             |
|----------|---------------|---|-----------------------------|
| 004012BE | FF25 50204000 | JMP DWORD PTR DS:[&user32.TranslateMessage]     | user32.TranslateMessage     |
| 004012C4 | FF25 54204000 | JMP DWORD PTR DS:[&user32.UpdateWindow]         | user32.UpdateWindow         |
| 004012CA | FF25 20204000 | JMP DWORD PTR DS:[&kernel32.ExitProcess]        | kernel32.ExitProcess        |
| 004012D0 | FF25 1C204000 | JMP DWORD PTR DS:[&kernel32.GetCommandLineA]    | kernel32.GetCommandLineA    |
| 004012D6 | FF25 24204000 | JMP DWORD PTR DS:[&kernel32.GetModuleHandleA]   | kernel32.GetModuleHandleA   |
| 004012DC | FF25 00204000 | JMP DWORD PTR DS:[&comctl32.ImageList_Add]      | comctl32.ImageList_Add      |
| 004012E2 | FF25 0C204000 | JMP DWORD PTR DS:[&comctl32.ImageList_Create]   | comctl32.ImageList_Create   |
| 004012E8 | FF25 08204000 | JMP DWORD PTR DS:[&comctl32.ImageList_Destroy]  | comctl32.ImageList_Destroy  |
| 004012EE | FF25 04204000 | JMP DWORD PTR DS:[&comctl32.InitCommonControls] | comctl32.InitCommonControls |
| 004012F4 | 00            | DB 00   |                             |
| 004012F5 | 00            | DB 00   |                             |
| 004012F6 | 00            | DB 00   |                             |
| 004012F7 | 00            | DB 00   |                             |
| 004012F8 | 00            | DB 00   |                             |
| 004012F9 | 00            | DB 00   |                             |
| 004012FA | 00            | DB 00   |                             |
| 004012FB | 00            | DB 00   |                             |
| 004012FC | 00            | DB 00   |                             |
| 004012FD | 00            | DB 00   |                             |
| 004012FE | 00            | DB 00   |                             |
| 004012FF | 00            | DB 00   |                             |
| 00401300 | 00            | DB 00   |                             |
| 00401301 | 00            | DB 00   |                             |
| 00401302 | 00            | DB 00   |                             |
| 00401303 | 00            | DB 00   |                             |
| 00401304 | 00            | DB 00   |                             |
| 00401305 | 00            | DB 00   |                             |
| 00401306 | 00            | DB 00   |                             |
| 00401307 | 00            | DB 00   |                             |
| 00401308 | 00            | DB 00   |                             |
| 00401309 | 00            | DB 00   |                             |
| 0040130A | 00            | DB 00   |                             |
| 0040130B | 00            | DB 00   |                             |
| 0040130C | 00            | DB 00   |                             |
| 0040130D | 00            | DB 00   |                             |
| 0040130E | 00            | DB 00   |                             |
| 0040130F | 00            | DB 00   |                             |
| 00401310 | 00            | DB 00   |                             |
| 00401311 | 00            | DB 00   |                             |
| 00401312 | 00            | DB 00   |                             |
| 00401313 | 00            | DB 00   |                             |
| 00401314 | 00            | DB 00   |                             |

Now select "Binary" -> "Binary paste":



Now you will see the copied code in our code cave:

|          |               |   |                             |
|----------|---------------|---|-----------------------------|
| 004012B6 | FF25 78204000 | JMP DWORD PTR DS:[&user32.SendDlgItemMessageA]  | user32.SendDlgItemMessageA  |
| 004012BC | FF25 44204000 | JMP DWORD PTR DS:[&user32.SetDlgItemTextA]      | user32.SetDlgItemTextA      |
| 004012B8 | FF25 48204000 | JMP DWORD PTR DS:[&user32.SetFocus]             | user32.SetFocus             |
| 004012BD | FF25 4C204000 | JMP DWORD PTR DS:[&user32.ShowWindow]           | user32.ShowWindow           |
| 004012BE | FF25 50204000 | JMP DWORD PTR DS:[&user32.TranslateMessage]     | user32.TranslateMessage     |
| 004012C4 | FF25 54204000 | JMP DWORD PTR DS:[&user32.UpdateWindow]         | user32.UpdateWindow         |
| 004012CA | FF25 20204000 | JMP DWORD PTR DS:[&kernel32.ExitProcess]        | kernel32.ExitProcess        |
| 004012D0 | FF25 1C204000 | JMP DWORD PTR DS:[&kernel32.GetCommandLineA]    | kernel32.GetCommandLineA    |
| 004012D6 | FF25 24204000 | JMP DWORD PTR DS:[&kernel32.GetModuleHandleA]   | kernel32.GetModuleHandleA   |
| 004012DC | FF25 00204000 | JMP DWORD PTR DS:[&comctl32.ImageList_Add]      | comctl32.ImageList_Add      |
| 004012E2 | FF25 0C204000 | JMP DWORD PTR DS:[&comctl32.ImageList_Create]   | comctl32.ImageList_Create   |
| 004012E8 | FF25 08204000 | JMP DWORD PTR DS:[&comctl32.ImageList_Destroy]  | comctl32.ImageList_Destroy  |
| 004012EE | FF25 04204000 | JMP DWORD PTR DS:[&comctl32.InitCommonControls] | comctl32.InitCommonControls |
| 004012F4 | 00            | DB 00   |                             |
| 004012F5 | 66 3D 017D    | CMPL AX,7D01                                    |                             |
| 004012F9 | 75 11         | JNZ SHORT SimpleEd.0040130C                     |                             |
| 004012FB | 6A 00         | PUSH 0  |                             |
| 004012FD | 68 0B0B0000   | PUSH 0B0B                                       |                             |
| 00401302 | FF75 08       | PUSH DWORD PTR SS:[EBP+8]                       |                             |
| 00401305 | E8 8E000000   | CALL SimpleEd.00401398                          |                             |
| 0040130A | EB 0E         | JMP SHORT SimpleEd.0040131A                     |                             |
| 0040130C | 00            | DB 00   |                             |
| 0040130D | 00            | DB 00   |                             |
| 0040130E | 00            | DB 00   |                             |
| 0040130F | 00            | DB 00   |                             |
| 00401310 | 00            | DB 00   |                             |
| 00401311 | 00            | DB 00   |                             |
| 00401312 | 00            | DB 00   |                             |
| 00401313 | 00            | DB 00   |                             |
| 00401314 | 00            | DB 00   |                             |
| 00401315 | 00            | DB 00   |                             |
| 00401316 | 00            | DB 00   |                             |
| 00401317 | 00            | DB 00   |                             |
| 00401318 | 00            | DB 00   |                             |
| 00401319 | 00            | DB 00   |                             |
| 0040131A | 00            | DB 00   |                             |
| 0040131B | 00            | DB 00   |                             |
| 0040131C | 00            | DB 00   |                             |
| 0040131D | 00            | DB 00   |                             |
| 0040131E | 00            | DB 00   |                             |
| 0040131F | 00            | DB 00   |                             |
| 00401320 | 00            | DB 00   |                             |

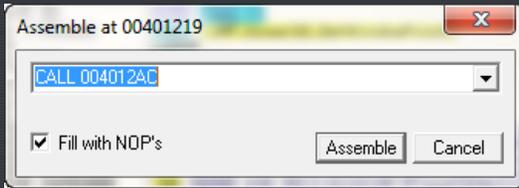
There are a couple points we need to make about this code. The first is that it conveniently calls the first line of empty code where our custom code will go if AX does not equal 7D01. This is nice 😊 Only, it will have to be changed 😞

```

004012F4 66:3D 017D CMP AX,7D01 DB 00
004012F5 75 11 JNZ SHORT SimpleEd.0040130C
004012F9 6A 00 PUSH 0
004012FB 68 B80B0000 PUSH 0BB8
004012FD FF75 08 PUSH DWORD PTR SS:[EBP+8]
00401302 E8 8E000000 CALL SimpleEd.00401398
00401305 EB 0E JMP SHORT SimpleEd.0040131A
0040130C 00 DB 00
0040130D 00 DB 00
0040130E 00 DB 00
0040130F 00 DB 00
00401310 00 DB 00

```

Secondly, you need to know that the call to SetDlgItemTextA that is now at address 401305 is wrong. Notice it is a SHORT call, meaning it doesn't call a specific address, but an address at a specific OFFSET from this current line. (SHORT calls must call code within FF bytes). So what we need to do is go back up to the original call to SetDlgItemTextA and find the actual address it calls, and then paste that address into this code. Scroll back up to the original call at address 401219 and hit the space bar on it to bring up the assembly window. This shows us the actual address instead of the API name. Copy this address:

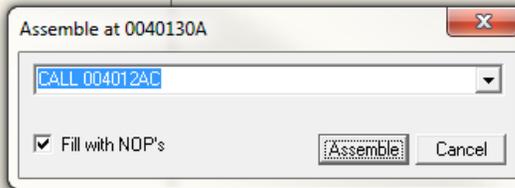


and now go back to our code cave and paste this correct address into the call:

```

0040120C 44 00 ADD DWORD PTR DS:[&comctl32.ImageList_Add],0
0040120E 75 11 JNZ SHORT SimpleEd.0040130C
0040120F 6A 00 PUSH 0
00401211 68 B80B0000 PUSH 0BB8
00401213 FF75 08 PUSH DWORD PTR SS:[EBP+8]
00401215 E8 A2FFFFFF CALL <JMP.&user32.SetDlgItemTextA>
00401218 EB 0E JMP SHORT SimpleEd.0040131A
00401219 0000 ADD BYTE PTR DS:[EAX],AL
0040121A 0000 ADD BYTE PTR DS:[EAX],AL
0040121B 0000 ADD BYTE PTR DS:[EAX],AL
0040121C 0000 ADD BYTE PTR DS:[EAX],AL
0040121D 0000 ADD BYTE PTR DS:[EAX],AL
0040121E 0000 ADD BYTE PTR DS:[EAX],AL
0040121F 0000 ADD BYTE PTR DS:[EAX],AL
00401220 0000 ADD BYTE PTR DS:[EAX],AL
00401221 0000 ADD BYTE PTR DS:[EAX],AL
00401222 0000 ADD BYTE PTR DS:[EAX],AL
00401223 0000 ADD BYTE PTR DS:[EAX],AL
00401224 0000 ADD BYTE PTR DS:[EAX],AL
00401225 0000 ADD BYTE PTR DS:[EAX],AL
00401226 0000 ADD BYTE PTR DS:[EAX],AL
00401227 00 DB 00
00401228 00 DB 00
00401229 00 DB 00
0040122A 00 DB 00
0040122B 00 DB 00
0040122C 00 DB 00
0040122D 00 DB 00
0040122E 00 DB 00

```

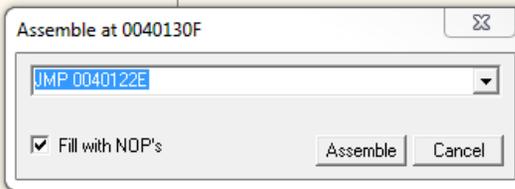


Notice the call changes to show a correct call to SetDlgItemTextA. This is why the original JNZ code that we thought was so nice of it to point to the right address is now wrong- we have added bytes. This will need to be fixed (later). We also need to fix the jump at the end as it uses a SHORT jump, and since we've moved it, the offset will be different. So go up to the original jump at 40121E, hit the space bar, copy the contents, and then paste them into this last JMP instruction (don't forget to delete the SHORT word or you'll get an error:

```

004012EE 44 00 ADD DWORD PTR DS:[&comctl32.InItCommonControls],comctl32.InItCommonControls
004012F0 66:3D 017D CMP AX,7D01 DB 00
004012F1 75 11 JNZ SHORT SimpleEd.0040130C
004012F3 6A 00 PUSH 0
004012F5 68 B80B0000 PUSH 0BB8
004012F7 FF75 08 PUSH DWORD PTR SS:[EBP+8]
004012F9 E8 A2FFFFFF CALL <JMP.&user32.SetDlgItemTextA>
004012FB E9 1FFFFFFF JMP SimpleEd.0040122E
004012FC 90 NOP
004012FD 0000 ADD BYTE PTR DS:[EAX],AL
004012FE 0000 ADD BYTE PTR DS:[EAX],AL
004012FF 0000 ADD BYTE PTR DS:[EAX],AL
00401300 0000 ADD BYTE PTR DS:[EAX],AL
00401301 0000 ADD BYTE PTR DS:[EAX],AL
00401302 0000 ADD BYTE PTR DS:[EAX],AL
00401303 0000 ADD BYTE PTR DS:[EAX],AL
00401304 0000 ADD BYTE PTR DS:[EAX],AL
00401305 0000 ADD BYTE PTR DS:[EAX],AL
00401306 0000 ADD BYTE PTR DS:[EAX],AL
00401307 0000 ADD BYTE PTR DS:[EAX],AL
00401308 0000 ADD BYTE PTR DS:[EAX],AL
00401309 0000 ADD BYTE PTR DS:[EAX],AL
0040130A 0000 ADD BYTE PTR DS:[EAX],AL
0040130B 0000 ADD BYTE PTR DS:[EAX],AL
0040130C 0000 ADD BYTE PTR DS:[EAX],AL
0040130D 0000 ADD BYTE PTR DS:[EAX],AL
0040130E 0000 ADD BYTE PTR DS:[EAX],AL
0040130F 0000 ADD BYTE PTR DS:[EAX],AL
00401310 0000 ADD BYTE PTR DS:[EAX],AL
00401311 0000 ADD BYTE PTR DS:[EAX],AL
00401312 0000 ADD BYTE PTR DS:[EAX],AL
00401313 0000 ADD BYTE PTR DS:[EAX],AL
00401314 0000 ADD BYTE PTR DS:[EAX],AL
00401315 0000 ADD BYTE PTR DS:[EAX],AL
00401316 0000 ADD BYTE PTR DS:[EAX],AL
00401317 0000 ADD BYTE PTR DS:[EAX],AL
00401318 0000 ADD BYTE PTR DS:[EAX],AL
00401319 0000 ADD BYTE PTR DS:[EAX],AL
0040131A 0000 ADD BYTE PTR DS:[EAX],AL
0040131B 0000 ADD BYTE PTR DS:[EAX],AL
0040131C 0000 ADD BYTE PTR DS:[EAX],AL
0040131D 0000 ADD BYTE PTR DS:[EAX],AL
0040131E 0000 ADD BYTE PTR DS:[EAX],AL
0040131F 0000 ADD BYTE PTR DS:[EAX],AL
00401320 0000 ADD BYTE PTR DS:[EAX],AL
00401321 0000 ADD BYTE PTR DS:[EAX],AL
00401322 0000 ADD BYTE PTR DS:[EAX],AL
00401323 0000 ADD BYTE PTR DS:[EAX],AL
00401324 0000 ADD BYTE PTR DS:[EAX],AL
00401325 0000 ADD BYTE PTR DS:[EAX],AL
00401326 0000 ADD BYTE PTR DS:[EAX],AL
00401327 00 DB 00
00401328 00 DB 00
00401329 00 DB 00
0040132A 00 DB 00
0040132B 00 DB 00
0040132C 00 DB 00
0040132D 00 DB 00
0040132E 00 DB 00

```

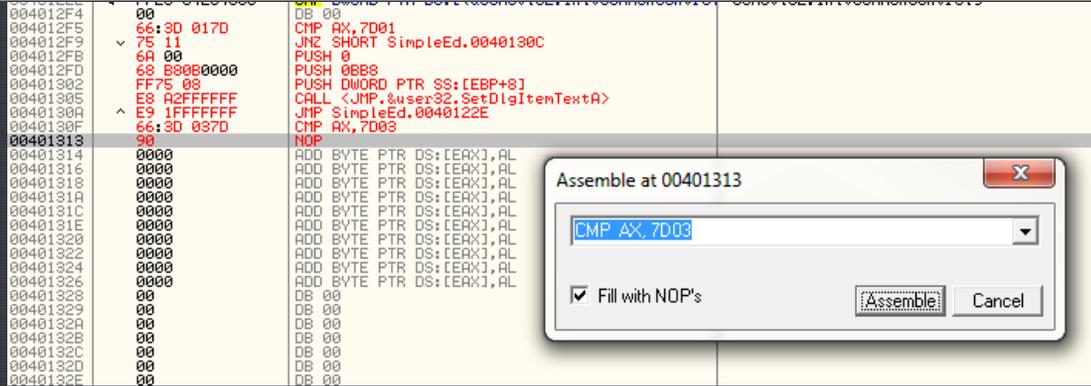


Now we need to add our own custom handler of our custom menu item. We know that the ID is 7D03 (32003) so we start with a compare AX to this value:

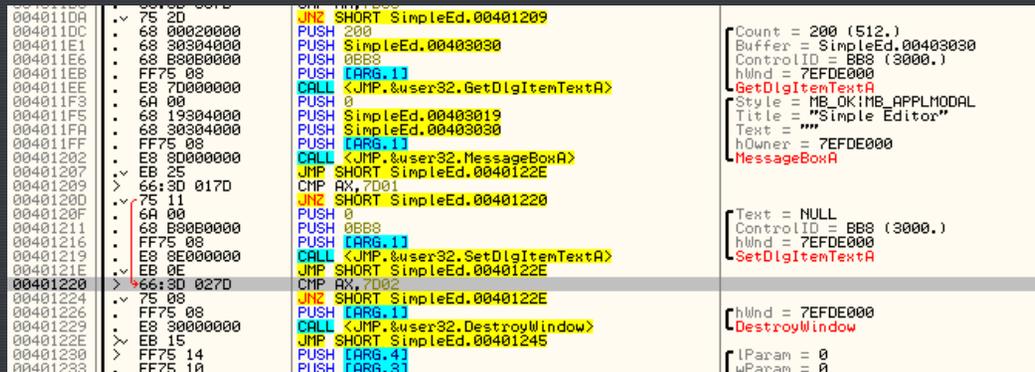
```

CMP AX, 7D03

```

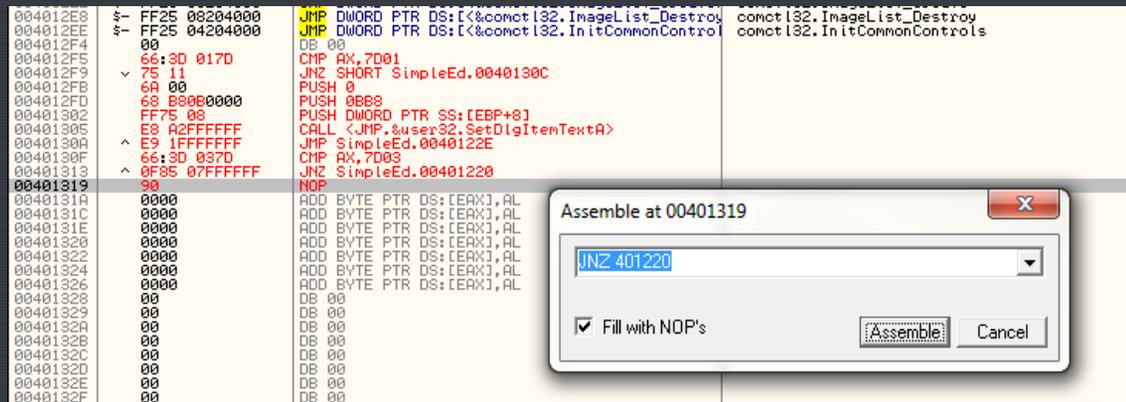


We then need to jump to the next check if our ID does not match. To find this address, go back up to the beginning of the third section (the section for "Exit" menu item) and note what the address is of the beginning of it:

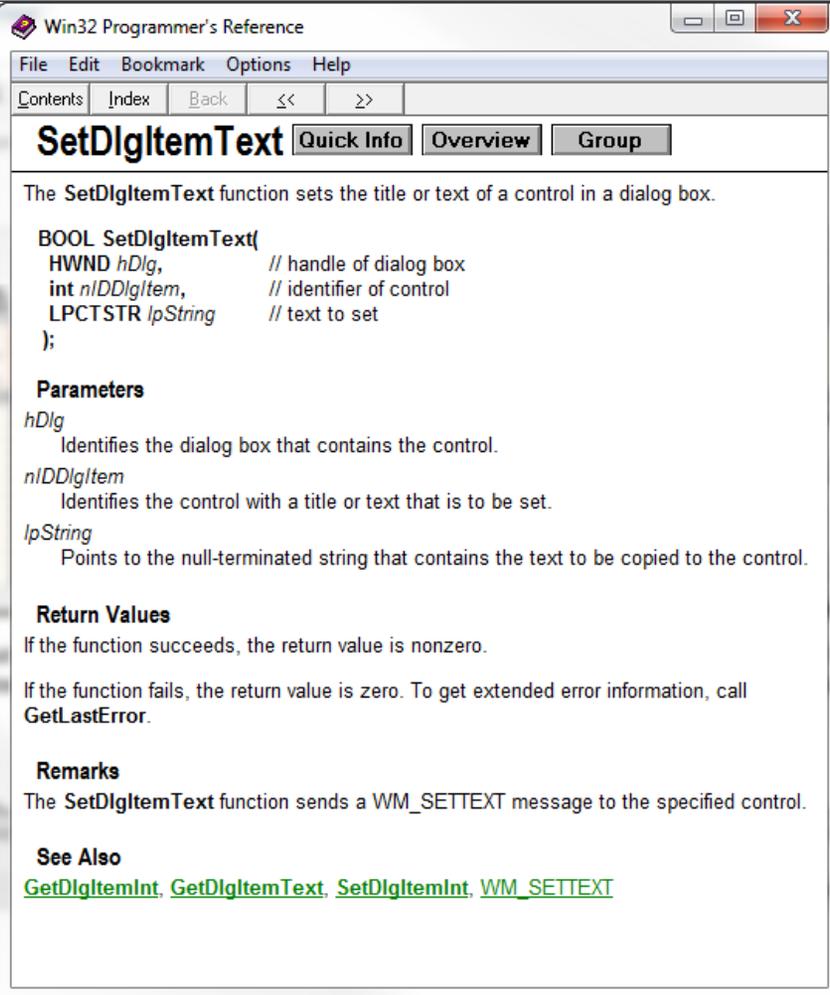


We can see that it is at address 401220 (and you can see that the prior section jumps to here from the red arrow 😊). So we need to add the next line of our custom code that performs this jump if our menu item was not selected:

JNZ 401220



OK, now we need to decide what it is we want to do with our menu item. I have decided that every time you choose "SHOUT OUT!!!" it will put in a friendly message into the edit text box. Fortunately, this is very similar to the 'Clear text' code used at the beginning of our code cave. Let's do a quick look at SetDlgItemTextA:



Notice the parameters; handle to window, ID of control, and a pointer to the text to display. If you look at the section we copied, you can see that they sent zero as the ptr to text (meaning none), the ID is 0BB8, and the handle to the window is `DWORD PTR SS:[EBP+8]`:

```

0040120C  FF25 00204000  JMP DWORD PTR DS:[&comctl32.ImageList_Add>
00401212  FF25 0C204000  JMP DWORD PTR DS:[&comctl32.ImageList_Creat
00401218  FF25 08204000  JMP DWORD PTR DS:[&comctl32.ImageList_Dest
0040121E  FF25 04204000  JMP DWORD PTR DS:[&comctl32.InItCommonCont
004012F4  00          DB 00
004012F5  66:3D 017D    CMP AX,7D01
004012F9  75 11        JNZ SHORT SimpleEd.0040130C
004012FB  6A 00        PUSH 0 ← Ptr to text
004012FD  68 B80B0000  PUSH 0BB8 ← ID
00401302  FF75 08        PUSH DWORD PTR SS:[EBP+8] ← handle
00401305  E8 A2FFFFFF  CALL <JMP.&user32.SetDlgItemTextA>
00401309  E9 1FFFFFFF  JMP SimpleEd.0040122E

```

So, we are going to do the same thing, but instead of pushing a zero like the first code section does (to signify no text), we are going to push an address of the text we want to inject 😊 Now, as we have not entered our text to send yet, when I first assemble the push with the pointer to the text, I am just going to throw a guess as to what address it's going to be, then later, when I know the exact address, I'll come back and replace it. This is just so that Olly reserves enough bytes for the jump. So lets add our next line of code as `PUSH 401328`:

|          |             |                                    |
|----------|-------------|------------------------------------|
| 004012FB | 6A 00       | PUSH 0                             |
| 004012FD | 68 B80B0000 | PUSH 0BB8                          |
| 00401302 | FF75 08     | PUSH DWORD PTR SS:[EBP+8]          |
| 00401305 | E8 A2FFFFFF | CALL <JMP.&user32.SetDlgItemTextA> |
| 00401309 | E9 1FFFFFFF | JMP SimpleEd.0040122E              |
| 0040130F | 6E:3D 037D  | CMP AX,7D03                        |
| 00401313 | 75 08       | JNZ SimpleEd.00401320              |
| 00401319 | 68 28134000 | PUSH SimpleEd.00401328             |
| 0040131E | 0000        | ADD BYTE PTR DS:[EAX],AL           |
| 00401320 | 0000        | ADD BYTE PTR DS:[EAX],AL           |
| 00401322 | 0000        | ADD BYTE PTR DS:[EAX],AL           |
| 00401324 | 0000        | ADD BYTE PTR DS:[EAX],AL           |
| 00401326 | 0000        | ADD BYTE PTR DS:[EAX],AL           |
| 00401328 | 00          | DB 00                              |
| 00401329 | 00          | DB 00                              |
| 0040132A | 00          | DB 00                              |
| 0040132B | 00          | DB 00                              |
| 0040132C | 00          | DB 00                              |
| 0040132D | 00          | DB 00                              |
| 0040132E | 00          | DB 00                              |
| 0040132F | 00          | DB 00                              |
| 00401330 | 00          | DB 00                              |
| 00401331 | 00          | DB 00                              |

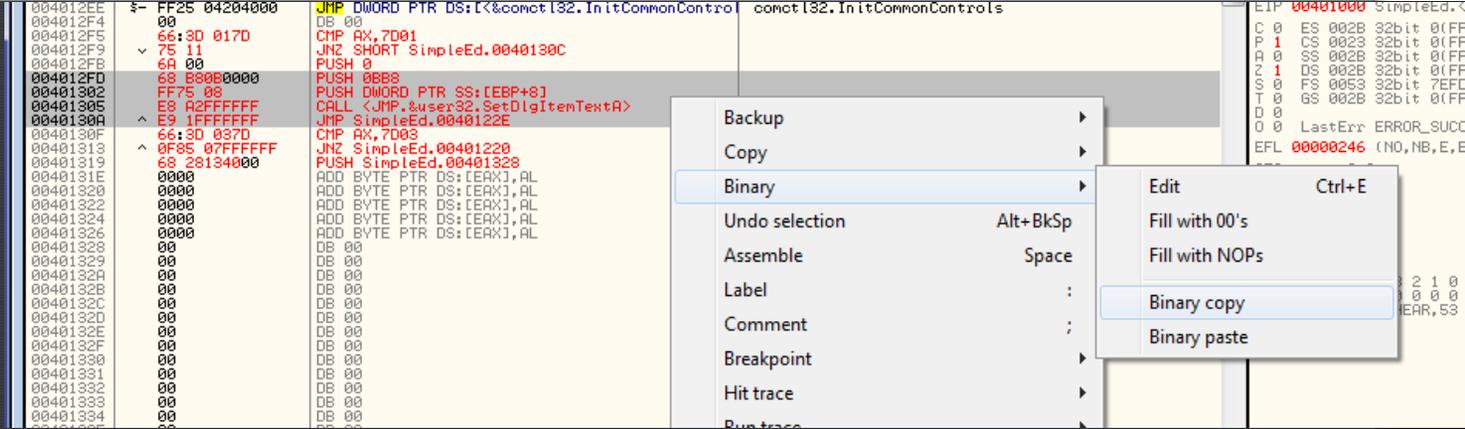
Assemble at 0040131E

PUSH 401328

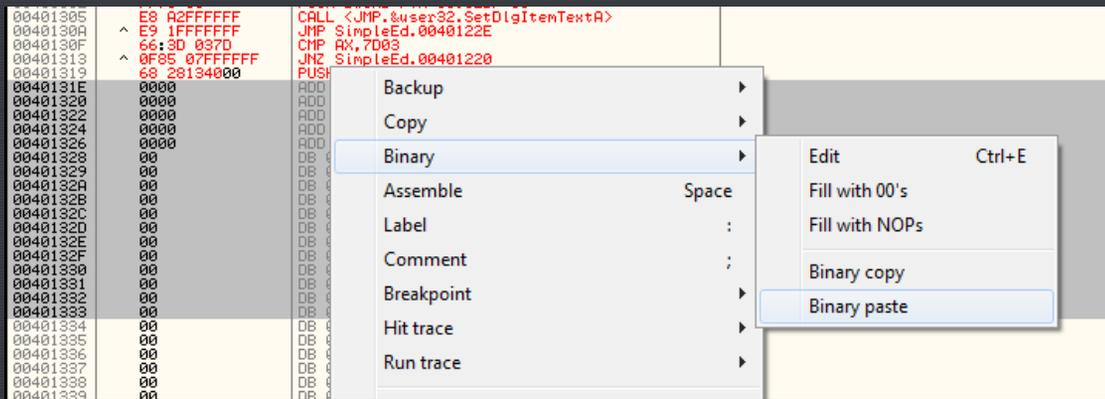
Fill with NOP's

Assemble Cancel

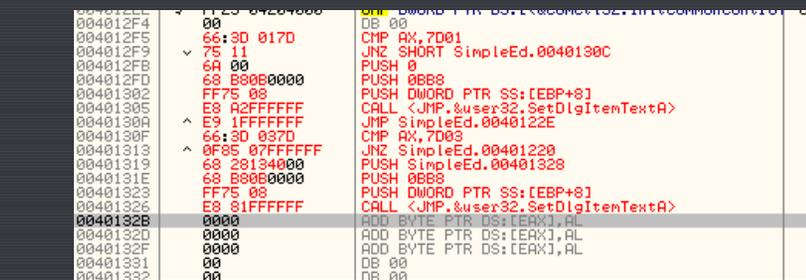
Now, since the second and third argument, as well as the call to `SetDlgItemTextA` are the same, I will just binary copy them from address range 4012FD-40130A:



Then binary paste by selecting a bunch of lines (including the NOP at the end) and selecting "Binary" -> "Binary paste" at the end of our custom code:

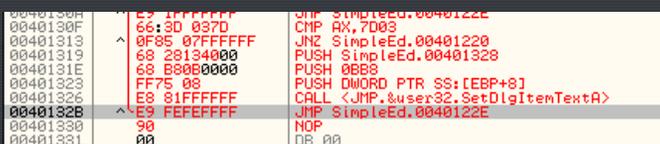


In this new section, let's first fix the call to SetDlgItemTextA; highlight the call to SetDlgItemTextA above, hit space bar, copy the address, and paste it into the last call of our code ( you will see the call change to SetDlgItemTextA):



\*ps. Ignore the ADD BYTE PTR DS:[EAX],AL instructions you see. This was a mistake I made when making this tutorial and I had to set the memory back to zeroes- and when you do this, Olly displays them in this way, instead of the normal DB 00. They both mean the same thing though 😊

Now, the final instruction is to jump back into the original code after inserting our text. This jump will be the same as the jump at address 40130A, the same place the first section of our code jumps to when it's done performing it's functionality. In fact, it's the same place that all code sections that check for a menu item clicked jump to when they're done doing whatever they do. So replace the last jump with the address of the jump at address 40130A:



\* You can ignore the NOP at the end, or you can modify it to be a zero if you're a perfectionist 😊

Now we need to put in our message. Just select an address a couple addresses down from your code cave ( I like to leave some space in case I made a mistake in the cave- it gives me some room to make changes in). Let's add our message at 40133A. Highlight that line and choose "Binary" -> "Edit". Click in the top box (for ASCII), hit delete once to get rid of the zero that's already in this spot, and type your

message. You may type a message that you want, but I usually try to make mine helpful, so I put "Don't feel bad- half the people in the world are below average." When you click OK, your ASCII data will be saved, but Olly will think it's code, so he will display it wrong. Don't worry, it will show up properly soon:)

|          |                  |   |                             |
|----------|------------------|---|-----------------------------|
| 004012E2 | FF25 0C204000    | JMP DWORD PTR DS:[&conct132.ImageList_Create]   | comct132.ImageList_Create   |
| 004012E3 | FF25 08204000    | JMP DWORD PTR DS:[&conct132.ImageList_Destroy]  | comct132.ImageList_Destroy  |
| 004012EE | FF25 04204000    | JMP DWORD PTR DS:[&conct132.InItCommonControls] | comct132.InItCommonControls |
| 004012F4 | 00               | DB 00   |                             |
| 004012F5 | 66:3D 017D       | CMP AX,7D01                                     |                             |
| 004012F9 | 75 11            | JNZ SHORT SimpleEd.0040130C                     |                             |
| 004012FB | 6A 00            | PUSH 0  |                             |
| 004012FD | 68 B80B0000      | PUSH 0BB8                                       |                             |
| 00401302 | FF75 08          | PUSH DWORD PTR SS:[EBP+8]                       |                             |
| 00401305 | E8 A2FFFFFF      | CALL <JMP.&user32.SetDlgItemTextA>              |                             |
| 0040130A | E9 1FFFFFFF      | JMP SimpleEd.0040122E                           |                             |
| 0040130F | 66:3D 037D       | CMP AX,7D03                                     |                             |
| 00401313 | ^ 0F85 07FFFFFFF | JNZ SimpleEd.00401220                           |                             |
| 00401319 | 68 28134000      | PUSH SimpleEd.00401328                          |                             |
| 0040131E | 68 B80B0000      | PUSH 0BB8                                       |                             |
| 00401323 | FF75 08          | PUSH DWORD PTR SS:[EBP+8]                       |                             |
| 00401326 | E8 81FFFFFF      | CALL <JMP.&user32.SetDlgItemTextA>              |                             |
| 0040132B | ^ E9 FEFFFFFF    | JMP SimpleEd.0040122E                           |                             |
| 00401330 | 90               | NOP   |                             |
| 00401331 | 00               | DB 00   |                             |
| 00401332 | 00               | DB 00   |                             |
| 00401333 | 00               | DB 00   |                             |
| 00401334 | 00               | DB 00   |                             |
| 00401335 | 00               | DB 00   |                             |
| 00401336 | 00               | DB 00   |                             |
| 00401337 | 00               | DB 00   |                             |
| 00401338 | 00               | DB 00   |                             |
| 00401339 | 00               | DB 00   |                             |
| 0040133A | 44               | INC ESP   |                             |
| 0040133B | 6F               | OUTS DX,DWORD PTR ES:[EDI]                      | I/O command                 |
| 0040133C | 6E               | OUTS DX,BYTE PTR ES:[EDI]                       | I/O command                 |
| 0040133D |                  | DAA   |                             |
| 0040133E | 74 20            | JE SHORT SimpleEd.00401360                      |                             |
| 00401340 | 66:              | PREFIX DATASIZE:                                | Superfluous prefix          |
| 00401341 | 65:              | PREFIX DS:                                      | Superfluous prefix          |
| 00401342 | 65:6C            | INS BYTE PTR ES:[EDI],DX                        | I/O command                 |
| 00401344 | 2062 61          | AND BYTE PTR DS:[ECX+61],AH                     |                             |
| 00401347 | 64:2D 2068616C   | SUB EAX,6C616820                                | Superfluous prefix          |
| 0040134D | 66:207468 65     | AND BYTE PTR DS:[EAX+EBP*2+65],DH               |                             |
| 00401352 | 2070 65          | AND BYTE PTR DS:[EAX+65],DH                     |                             |
| 00401355 | 6F               | OUTS DX,DWORD PTR ES:[EDI]                      | I/O command                 |
| 00401356 | 70 6C            | JO SHORT SimpleEd.004013C4                      |                             |
| 00401358 | 65:2069 6E       | AND BYTE PTR GS:[ECX+6E],CH                     |                             |
| 0040135C | 207468 65        | AND BYTE PTR DS:[EAX+EBP*2+65],DH               |                             |
| 00401360 | 2077 6F          | AND BYTE PTR DS:[EDI+6F],DH                     |                             |
| 00401363 | 72 6C            | JB SHORT SimpleEd.004013D1                      |                             |
| 00401365 | 64:2061 72       | AND BYTE PTR FS:[ECX+72],AH                     |                             |
| 00401369 | 65:2062 65       | AND BYTE PTR GS:[EDX+65],AH                     |                             |
| 0040136D | 6C               | INS BYTE PTR ES:[EDI],DX                        |                             |
| 0040136E | 6F               | OUTS DX,DWORD PTR ES:[EDI]                      | I/O command                 |
| 0040136F | 77 20            | JR SHORT SimpleEd.00401391                      | I/O command                 |
| 00401371 | 61               | POPAD   |                             |
| 00401372 | 76 65            | JBE SHORT SimpleEd.004013D9                     |                             |
| 00401374 | 72 61            | JB SHORT SimpleEd.004013D7                      |                             |
| 00401376 | 67:65:0000       | ADD BYTE PTR GS:[BX+SI],AL                      |                             |
| 0040137A | 00               | DB 00   |                             |
| 0040137B | 00               | DB 00   |                             |
| 0040137C | 00               | DB 00   |                             |
| 0040137D | 00               | DB 00   |                             |

Our code cave

Our ASCII data

Now two last loose ends... Don't forget to change the address of the PUSH for the pointer to our text at address 401319. Remember, this was a temporary address we pushed, so we need to replace our original guess of 401328 with the actual address of the beginning of our ASCII text, 40133A. You'll know it worked because the text will show up right after the push:

|          |                  |                                    |  |
|----------|------------------|------------------------------------|--|
| 0040130A | ^ E9 1FFFFFFF    | JMP SimpleEd.0040122E              |  |
| 0040130F | 66:3D 037D       | CMP AX,7D03                        |  |
| 00401313 | ^ 0F85 07FFFFFFF | JNZ SimpleEd.00401220              |  |
| 00401319 | 68 3A134000      | PUSH SimpleEd.0040133A             | ASCII "Don't feel bad- half the people in the world are be |
| 0040131E | 68 B80B0000      | PUSH 0BB8                          |  |
| 00401323 | FF75 08          | PUSH DWORD PTR SS:[EBP+8]          |  |
| 00401326 | E8 81FFFFFF      | CALL <JMP.&user32.SetDlgItemTextA> |  |
| 0040132B | ^ E9 FEFFFFFF    | JMP SimpleEd.0040122E              |  |
| 00401330 | 90               | NOP                                |  |

And secondly, the first JNZ instruction at address 4012F9 is pointing to the wrong address (since we added bytes between this instruction and its destination). So go to that line and change the JNZ 40130C to JNZ 40130F. We know it's 40130F because that is the first line of the next section of code, comparing the clicked menu item with the next option:

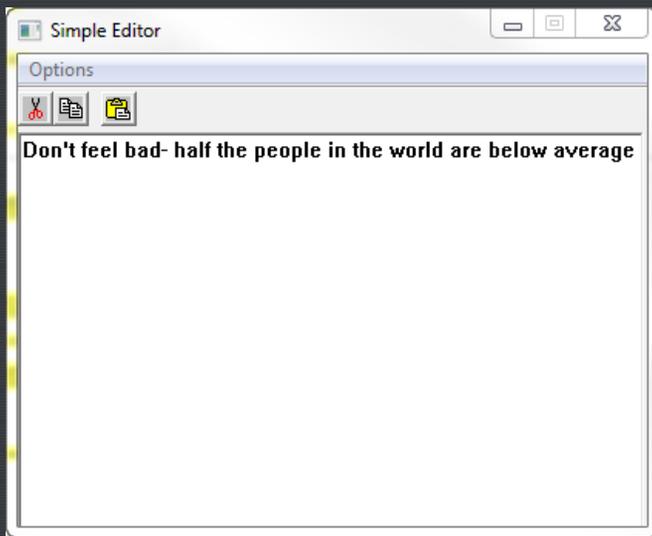
|          |                  |                                    |    |
|----------|------------------|------------------------------------|----|
| 004012F4 | 00               | DB 00                              |    |
| 004012F5 | 66:3D 017D       | CMP AX,7D01                        |    |
| 004012F9 | 75 14            | JNZ SHORT SimpleEd.0040130F        |    |
| 004012FB | 6A 00            | PUSH 0                             |    |
| 004012FD | 68 B80B0000      | PUSH 0BB8                          |    |
| 00401302 | FF75 08          | PUSH DWORD PTR SS:[EBP+8]          |    |
| 00401305 | E8 A2FFFFFF      | CALL <JMP.&user32.SetDlgItemTextA> |    |
| 0040130A | ^ E9 1FFFFFFF    | JMP SimpleEd.0040122E              |    |
| 0040130F | 66:3D 037D       | CMP AX,7D03                        |    |
| 00401313 | ^ 0F85 07FFFFFFF | JNZ SimpleEd.00401220              |    |
| 00401319 | 68 3A134000      | PUSH SimpleEd.0040133A             | AS |
| 0040131E | 68 B80B0000      | PUSH 0BB8                          |    |
| 00401323 | FF75 08          | PUSH DWORD PTR SS:[EBP+8]          |    |
| 00401326 | E8 81FFFFFF      | CALL <JMP.&user32.SetDlgItemTextA> |    |

You will also see that the grey arrow now points to the correct address- the compare at the beginning of our custom code.

OK, at this point I would save the changes you've made, seeing as things get a little dicey if you make a mistake and try to get back to this point (it's not impossible, it's just not fun). So highlight the entire section of code, from before your code cave to after your ASCII data, right-click and choose "Copy to executable" -> "Selection". Then right click on this new window and select "Save file" and save it as "SimpleEditor3.exe". After you do this, load our new file into Olly and let's see what he thinks of it. Scroll down to where our code cave is and it should look like this:



then hold your head up high, trooper, because today, you are a TOP NOTCH CRACKER!!!!!!



*\*ps. Don't forget to save your file as you have added the patch to jump to our code...and if you've run Olly, don't forget the patch will be disabled and you will need to open the Patches window and re-enable it 😊*

-Till next time

R4ndom