The Legend Of Random

Programming and Reverse Engineering

Tutorial #7: More Crackmes

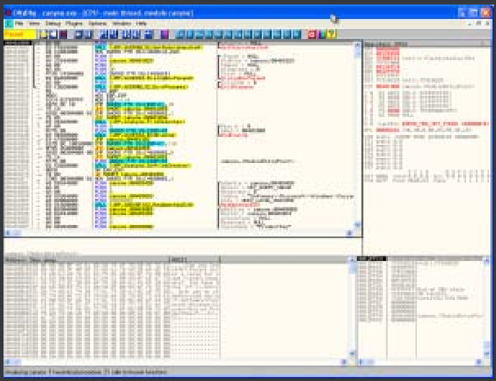
by R4ndom on Jun.15, 2012, under Reverse Engineering, Tools, Tutorials

Introduction

Welcome to Part 7 of of R4ndom's tutorials on Reverse Engineering. This time, we will be cracking two crackmes; one to re-iterate last tutorial's concepts, and one that we are going to have a little fun with 😃 In the download of this tutorial, you will find these two crackmes as well as the program "Resource Hacker" that we will be using on the second crackme. You can also download this tool on the tools page.

Investigating the binary

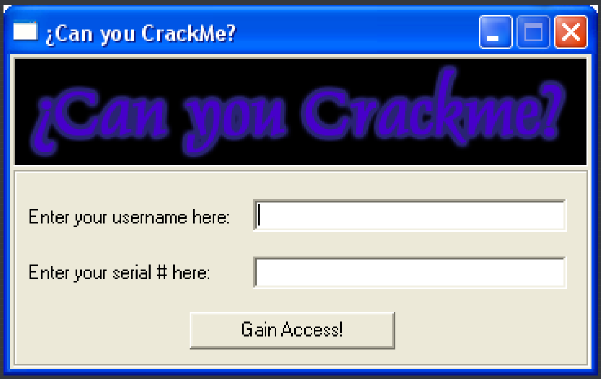
Let's jump right in. Load up canyou.exe (make sure the canyou.dll file is in the same place) into Olly:



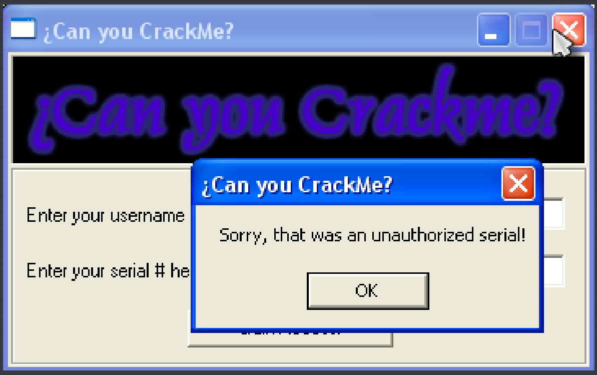
As I have said before, one of the most important things you can do before getting started is running the app and studying it. It gives you a plethora of information; is there a time trial? Are certain features disabled? Are there a certain amount of times it can be run? Is there a registry screen that you can enter a registration code?

These are all really important things to know, and as you get better in reverse engineering, you will gain more and more experience as to what you should be looking for (how long did it take to validate the code? Is it forcing you to a web site?…)

When we run the app we get:



and after entering some data we get:



¿Can you CrackMe?

Enter your username

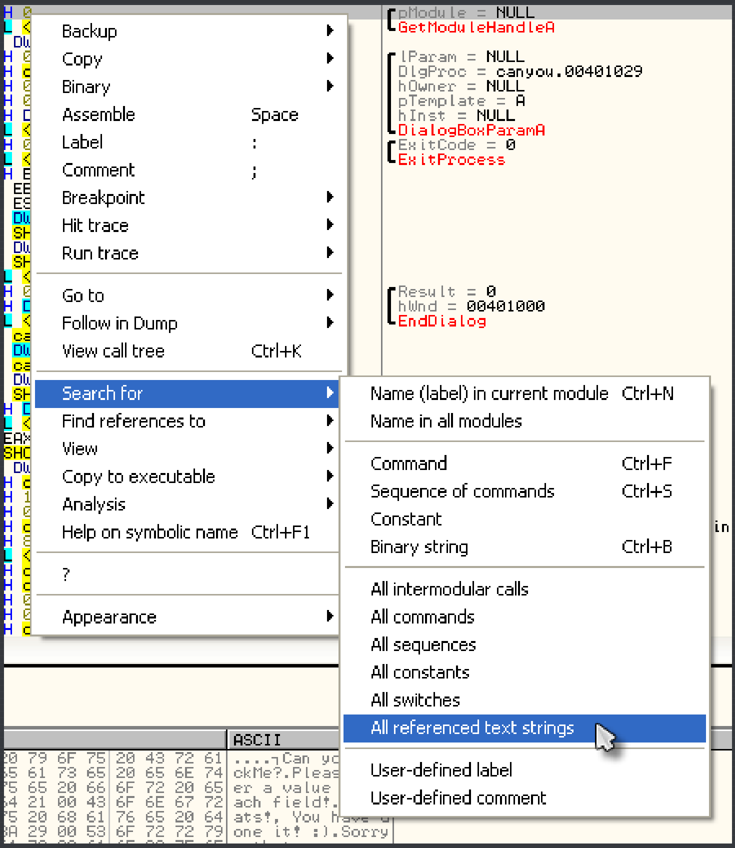
Enter your serial # he

¡Can you CrackMe?

Sorry, that was an unauthorized serial!

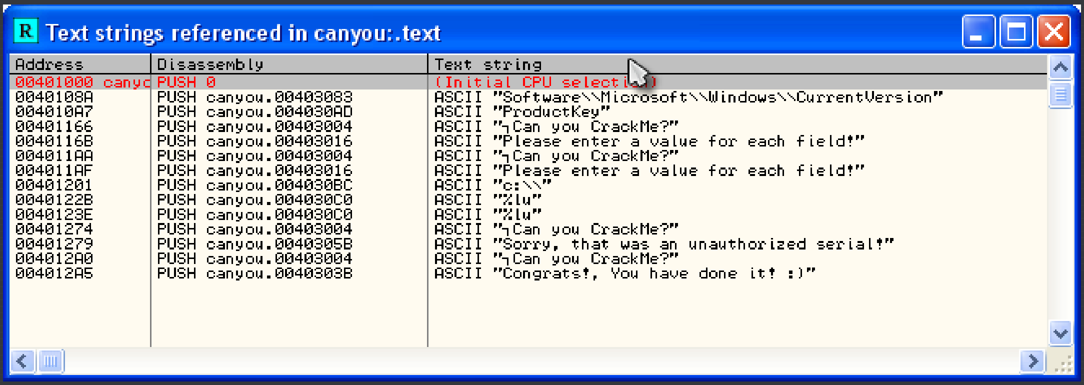
OK

You should know the drill by now. Click back in Olly and let's seewhat strings we can find:



```
pModule = NULL
GetModuleHandleA
lParam = NULL
DlgProc = canyou.00401029
hOwner = NULL
pTemplate = A
hInst = NULL
DialogBoxParamA
ExitCode = 0
ExitProcess
Result = 0
hWnd = 00401000
EndDialog
```

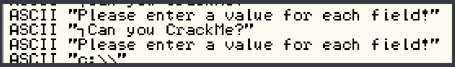
and we see what ASCII this crackme has to offer:



Text strings referenced in canyou:.text

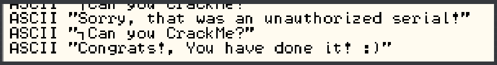
```
Address         Disassembly              Text string
00401000 canyo  PUSH 0                   (Initial CPU selection)
0040108A        PUSH canyou.00403083     ASCII "Software\\Microsoft\\Windows\\CurrentVersion"
004010A7        PUSH canyou.004030AD     ASCII "ProductKey"
00401166        PUSH canyou.00403004     ASCII "┐Can you CrackMe?"
0040116B        PUSH canyou.00403016     ASCII "Please enter a value for each field!"
004011AA        PUSH canyou.00403004     ASCII "┐Can you CrackMe?"
004011AF        PUSH canyou.00403016     ASCII "Please enter a value for each field!"
00401201        PUSH canyou.004030BC     ASCII "c:\\"
0040122B        PUSH canyou.004030C0     ASCII "%lu"
0040123E        PUSH canyou.004030C0     ASCII "%lu"
00401274        PUSH canyou.00403004     ASCII "┐Can you CrackMe?"
00401279        PUSH canyou.0040305B     ASCII "Sorry, that was an unauthorized serial!"
004012A0        PUSH canyou.00403004     ASCII "┐Can you CrackMe?"
004012A5        PUSH canyou.0040303B     ASCII "Congrats!, You have done it! :)"
```

Well, we obviously see a couple of important ones here. The first thing we notice is that something MUST be entered into each field:



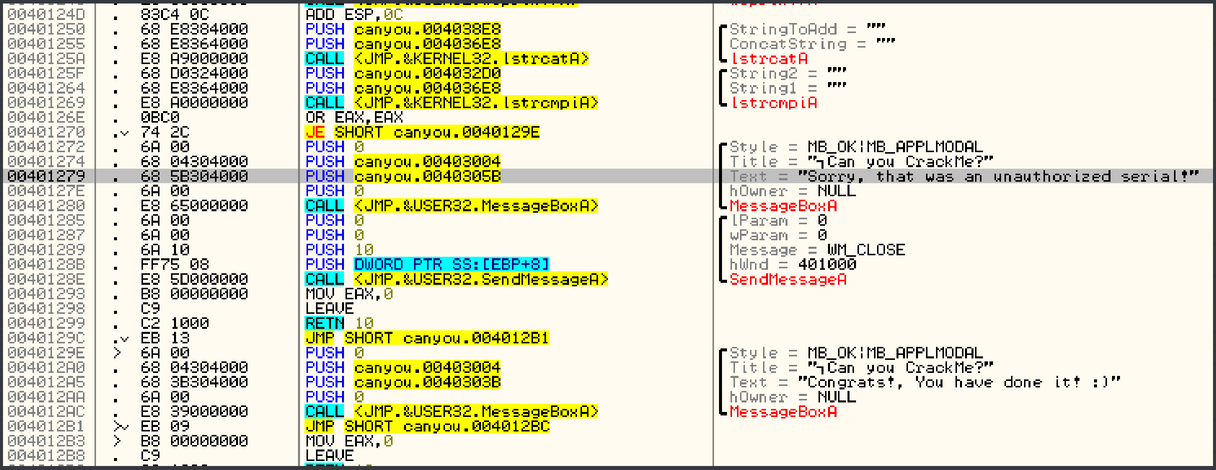
```
ASCII "Please enter a value for each field!"
ASCII "┐Can you CrackMe?"
ASCII "Please enter a value for each field!"
ASCII "c:\\"
```

and then we see the really important stuff:



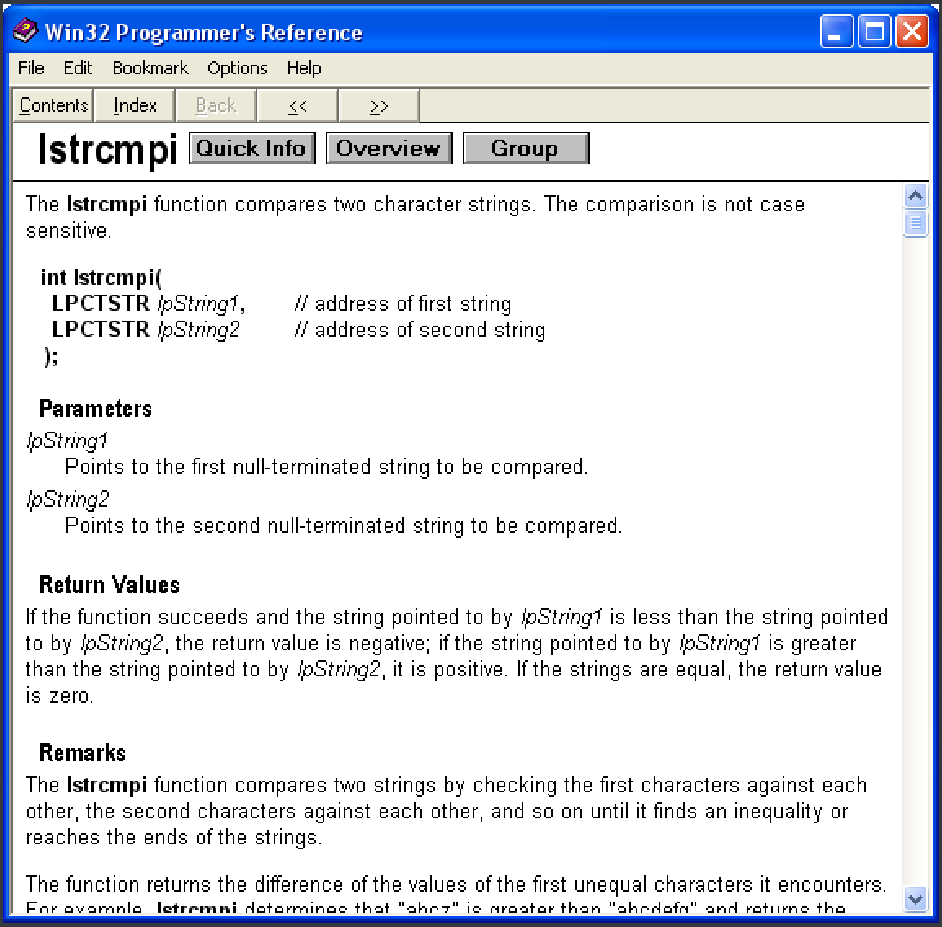
```
ASCII "Sorry, that was an unauthorized serial!"
ASCII "┐Can you CrackMe?"
ASCII "Congrats!, You have done it! :)"
```

Lets double-click on one and see where we go:



```
0040124D  .   83C4 0C        ADD ESP,0C
00401250  .   68 E8384000    PUSH canyou.004038E8                     [StringToAdd = ""
00401255  .   68 E8364000    PUSH canyou.004036E8                     |ConcatString = ""
0040125A  .   E8 A9000000    CALL <JMP.&KERNEL32.lstrcatA>            lstrcatA
0040125F  .   68 D0324000    PUSH canyou.004032D0                     [String2 = ""
00401264  .   68 E8364000    PUSH canyou.004036E8                     |String1 = ""
00401269  .   E8 A0000000    CALL <JMP.&KERNEL32.lstrcmpiA>           lstrcmpiA
0040126E  .   0BC0           OR EAX,EAX
00401270  .v  74 2C          JE SHORT canyou.0040129E
00401272  .   6A 00          PUSH 0                                   [Style = MB_OK|MB_APPLMODAL
00401274  .   68 04304000    PUSH canyou.00403004                     |Title = "┐Can you CrackMe?"
00401279  .   68 5B304000    PUSH canyou.0040305B                     |Text = "Sorry, that was an unauthorized serial!"
0040127E  .   6A 00          PUSH 0                                   |hOwner = NULL
00401280  .   E8 65000000    CALL <JMP.&USER32.MessageBoxA>           MessageBoxA
00401285  .   6A 00          PUSH 0                                   [lParam = 0
00401287  .   6A 00          PUSH 0                                   |wParam = 0
00401289  .   6A 10          PUSH 10                                  |Message = WM_CLOSE
0040128B  .   FF75 08        PUSH DWORD PTR SS:[EBP+8]                |hWnd = 401000
0040128E  .   E8 5D000000    CALL <JMP.&USER32.SendMessageA>          SendMessageA
00401293  .   B8 00000000    MOV EAX,0
00401298  .   C9             LEAVE
00401299  .   C2 1000        RETN 10
0040129C  .v  EB 13          JMP SHORT canyou.004012B1
0040129E  >   6A 00          PUSH 0                                   [Style = MB_OK|MB_APPLMODAL
004012A0  .   68 04304000    PUSH canyou.00403004                     |Title = "┐Can you CrackMe?"
004012A5  .   68 3B304000    PUSH canyou.0040303B                     |Text = "Congrats!, You have done it! :)"
004012AA  .   6A 00          PUSH 0                                   |hOwner = NULL
004012AC  .   E8 39000000    CALL <JMP.&USER32.MessageBoxA>           MessageBoxA
004012B1  >v  EB 09          JMP SHORT canyou.004012BC
004012B3  >   B8 00000000    MOV EAX,0
004012B8  .   C9             LEAVE
```

This should start looking familiar: we have a bad boy, followed by a good boy, with a very obvious jump right before the bad boy, presumably jumping to the good boy. Also, I want you to notice that before the jump is a call to the Windows API *lstrcmpi*. If we right-click on that and choose "Help on symbolic name" we see:



lstrcmpi

The **lstrcmpi** function compares two character strings. The comparison is not case sensitive.

```
int lstrcmpi(
  LPCTSTR lpString1,    // address of first string
  LPCTSTR lpString2     // address of second string
);
```

Parameters

lpString1
Points to the first null-terminated string to be compared.

lpString2
Points to the second null-terminated string to be compared.

Return Values

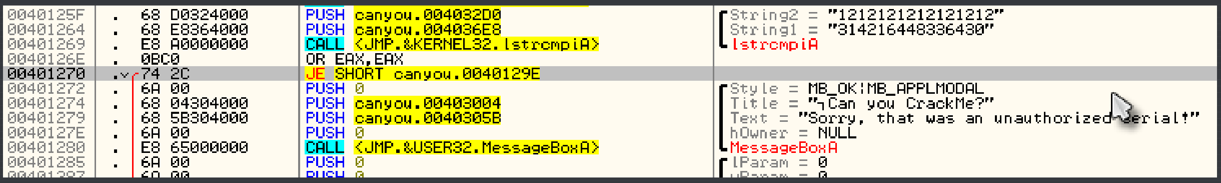
If the function succeeds and the string pointed to by *lpString1* is less than the string pointed to by *lpString2*, the return value is negative; if the string pointed to by *lpString1* is greater than the string pointed to by *lpString2*, it is positive. If the strings are equal, the return value is zero.

Remarks

The **lstrcmpi** function compares two strings by checking the first characters against each other, the second characters against each other, and so on until it finds an inequality or reaches the ends of the strings.

The function returns the difference of the values of the first unequal characters it encounters. For example, **lstrcmpi** determines that "abcz" is greater than "abcdefg" and returns the

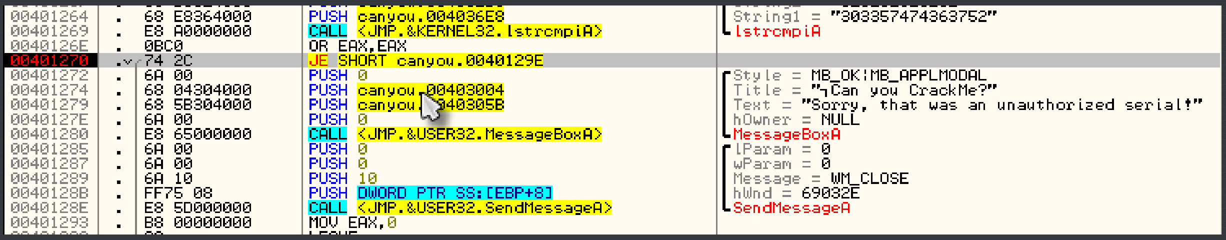
As you can see, lstrcmp compares two strings. This function is Very Important in reverse engineering and you will see it over and over. It is used in many registration/password schemes to compare the string that the user entered against a string that the app either has hard-coded in or has created. If the string compare comes back zero, the user has entered the correct info, meaning the strings are the same. If it comes back non-zero, the strings don't match. In the case of this crackme, our entered serial is probably checked against an internal or dynamically created string, and if EAX returns a zero, they are the same, otherwise they are not. Right know, Olly doesn't know what these strings are going to be as we haven't started the app and entered anything yet, but once we get going, Olly will replace the String1 = "" and String2 = "" lines with real strings. If we set a BP on the jump and then run the app, entering a serial (in this case "1212121212121212121212″), Olly will show us the strings that will be compared:



```
0040125F  .  68 D0324000    PUSH canyou.004032D0          String2 = "1212121212121212"
00401264  .  68 E8364000    PUSH canyou.004036E8          String1 = "314216448336430"
00401269  .  E8 A0000000    CALL <JMP.&KERNEL32.lstrcmpiA>  lstrcmpiA
0040126E  .  0BC0           OR EAX,EAX
00401270  .v 74 2C          JE SHORT canyou.0040129E
00401272  .  6A 00          PUSH 0                        Style = MB_OK|MB_APPLMODAL
00401274  .  68 04304000    PUSH canyou.00403004          Title = "¬Can you CrackMe?"
00401279  .  68 5B304000    PUSH canyou.0040305B          Text = "Sorry, that was an unauthorized serial!"
0040127E  .  6A 00          PUSH 0                        hOwner = NULL
00401280  .  E8 65000000    CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
00401285  .  6A 00          PUSH 0                        lParam = 0
```

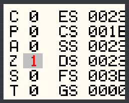
If you look at the lines above our jump instruction you can see that our password was compared with the value "314216448336430″, whatever that is. On return, EAX will contain a zero if they were identical and anything else if they weren't. Obviously, in this case, they are not going to match. The OR EAX, EAX is just a fancy way of finding out if EAX was zero or not, and if it is zero, the "JE SHORT canyou.0040129E" jumps to the good boy. I wanted to point this string compare routine out to you because in future tutorials, we will need to find out how this 15 digit number was created, and searching for *lstrcmp* can lead us to it's creation.

But in the mean time, let's do what we know. Set a BP on the JE instruction at line 401270 and re-start the app. Enter a username and serial and Olly will break on our BP:



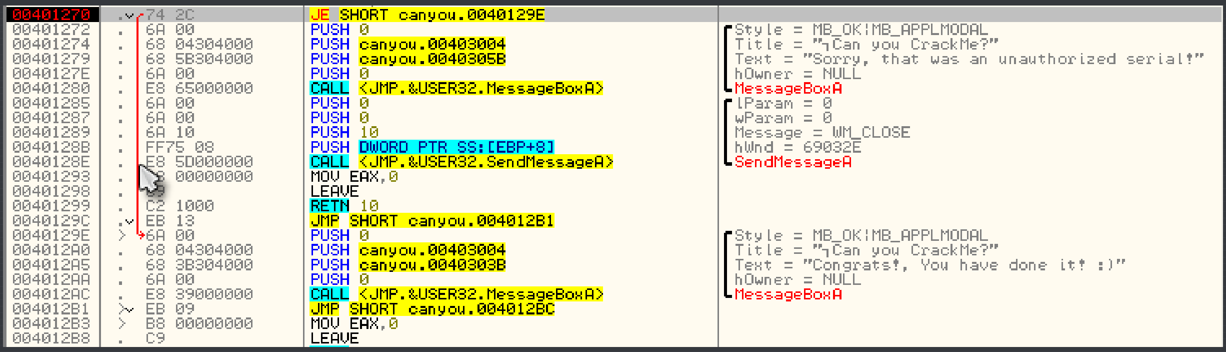
```
00401264  .  68 E8364000    PUSH canyou.004036E8          String1 = "303357474363752"
00401269  .  E8 A0000000    CALL <JMP.&KERNEL32.lstrcmpiA>  lstrcmpiA
0040126E  .  0BC0           OR EAX,EAX
00401270  .v 74 2C          JE SHORT canyou.0040129E
00401272  .  6A 00          PUSH 0                        Style = MB_OK|MB_APPLMODAL
00401274  .  68 04304000    PUSH canyou.00403004          Title = "¬Can you CrackMe?"
00401279  .  68 5B304000    PUSH canyou.0040305B          Text = "Sorry, that was an unauthorized serial!"
0040127E  .  6A 00          PUSH 0                        hOwner = NULL
00401280  .  E8 65000000    CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
00401285  .  6A 00          PUSH 0                        lParam = 0
00401287  .  6A 00          PUSH 0                        wParam = 0
00401289  .  6A 10          PUSH 10                       Message = WM_CLOSE
0040128B  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]     hWnd = 69032E
0040128E  .  E8 5D000000    CALL <JMP.&USER32.SendMessageA>  SendMessageA
00401293  .  B8 00000000    MOV EAX,0
```

We notice by the grey arrow that Olly is not going to jump to the good boy and instead plans on falling through to the bad boy so let's help him with that:



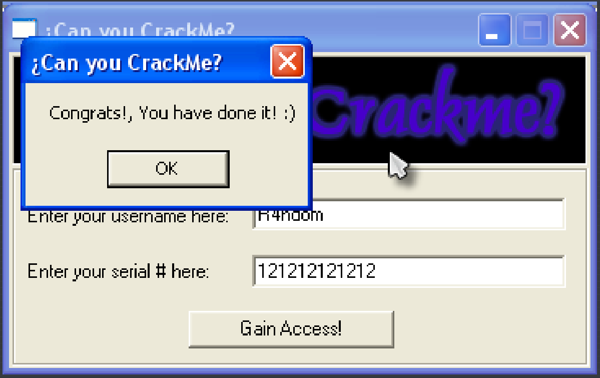
```
C 0  ES 0023
P 0  CS 001B
A 0  SS 0023
Z 1  DS 0023
S 0  FS 003B
T 0  GS 0000
```

and now Olly has it right:



```
00401270  .v 74 2C          JE SHORT canyou.0040129E
00401272  .  6A 00          PUSH 0                        Style = MB_OK|MB_APPLMODAL
00401274  .  68 04304000    PUSH canyou.00403004          Title = "¬Can you CrackMe?"
00401279  .  68 5B304000    PUSH canyou.0040305B          Text = "Sorry, that was an unauthorized serial!"
0040127E  .  6A 00          PUSH 0                        hOwner = NULL
00401280  .  E8 65000000    CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
00401285  .  6A 00          PUSH 0                        lParam = 0
00401287  .  6A 00          PUSH 0                        wParam = 0
00401289  .  6A 10          PUSH 10                       Message = WM_CLOSE
0040128B  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]     hWnd = 69032E
0040128E  .  E8 5D000000    CALL <JMP.&USER32.SendMessageA>  SendMessageA
00401293  .  B8 00000000    MOV EAX,0
00401298  .  C9             LEAVE
00401299  .  C2 1000        RETN 10
0040129C  .v EB 13          JMP SHORT canyou.004012B1
0040129E  >  6A 00          PUSH 0                        Style = MB_OK|MB_APPLMODAL
004012A0  .  68 04304000    PUSH canyou.00403004          Title = "¬Can you CrackMe?"
004012A5  .  68 3B304000    PUSH canyou.0040303B          Text = "Congrats!, You have done it! :)"
004012AA  .  6A 00          PUSH 0                        hOwner = NULL
004012AC  .  E8 39000000    CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004012B1  >v EB 09          JMP SHORT canyou.004012BC
004012B3  >  B8 00000000    MOV EAX,0
004012B8  .  C9             LEAVE
```

Let's run the app to make sure:



¡Can you CrackMe?

Congrats!, You have done it! :)

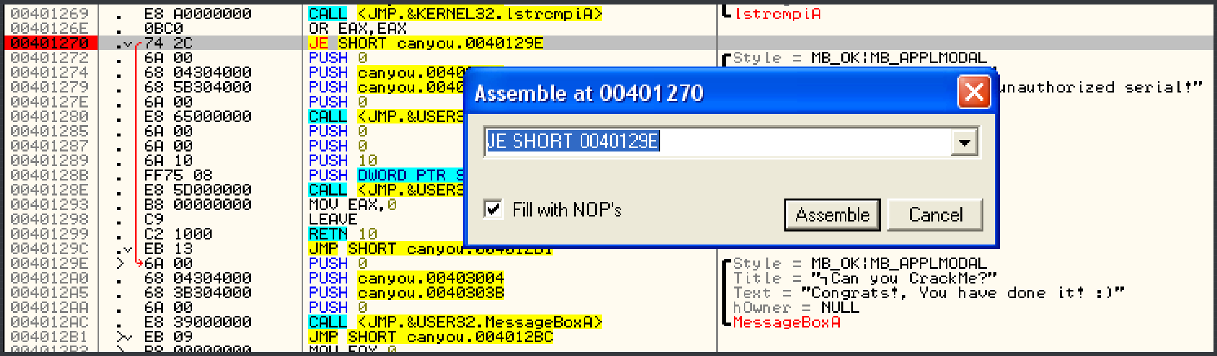
Enter your username here: R4ndom

Enter your serial # here: 121212121212

Now lets…

Patch The App

This time we do not want to NOP out the jump as that would make the program show the bad boy every time. Instead, we want to GUARANTEE that it jumps every time, that way it jumps to our good boy message. So go to the the line our BP is on (you can open the “Breakpoint Window” and double-click on the BP if you’re lost) and let’s change the instruction. Make sure you click on the JE instruction and then hit the space bar:



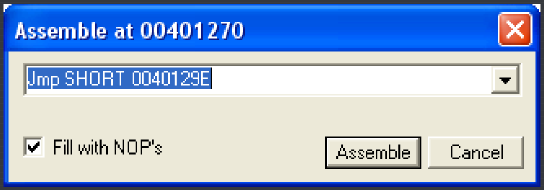
```
00401269  .   E8 A0000000    CALL <JMP.&KERNEL32.lstrcmpiA>           └lstrcmpiA
0040126E  .   0BC0           OR EAX,EAX
00401270  .v  74 2C          JE SHORT canyou.0040129E
00401272  .   6A 00          PUSH 0                                   ┌Style = MB_OK|MB_APPLMODAL
00401274  .   68 04304000    PUSH canyou.0040
00401279  .   68 5B304000    PUSH canyou.0040                          unauthorized serial!"
0040127E  .   6A 00          PUSH 0
00401280  .   E8 65000000    CALL <JMP.&USER3
00401285  .   6A 00          PUSH 0
00401287  .   6A 00          PUSH 0
00401289  .   6A 10          PUSH 10
0040128B  .   FF75 08        PUSH DWORD PTR S
0040128E  .   E8 5D000000    CALL <JMP.&USER3
00401293  .   B8 00000000    MOV EAX,0
00401298  .   C9             LEAVE
00401299  .   C2 1000        RETN 10
0040129C  .v  EB 13          JMP SHORT canyou.
0040129E  >   6A 00          PUSH 0                                   ┌Style = MB_OK|MB_APPLMODAL
004012A0  .   68 04304000    PUSH canyou.00403004                     │Title = "¡Can you CrackMe?"
004012A5  .   68 3B304000    PUSH canyou.0040303B                     │Text = "Congrats!, You have done it! :)"
004012AA  .   6A 00          PUSH 0                                   │hOwner = NULL
004012AC  .   E8 39000000    CALL <JMP.&USER32.MessageBoxA>           └MessageBoxA
004012B1  >v  EB 09          JMP SHORT canyou.004012BC
004012B3  >   B8 00000000    MOV EAX,0
```

Assemble at 00401270

JE SHORT 0040129E

☑ Fill with NOP's

Again, notice that the highlighted instruction has been entered into our edit box. Now, let’s change this JE (Jump on Equal) to a JMP (always jump, no matter what):

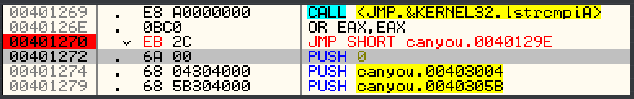


Assemble at 00401270

Jmp SHORT 0040129E

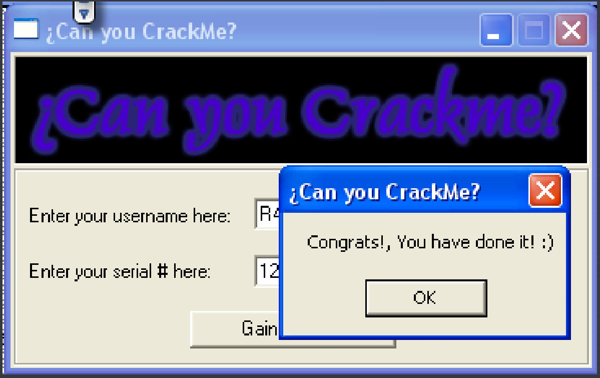
☑ Fill with NOP's

Click the Assemble button, and then the cancel button and you will see our change has been put in the code:



```
00401269  .   E8 A0000000    CALL <JMP.&KERNEL32.lstrcmpiA>
0040126E  .   0BC0           OR EAX,EAX
00401270   v  EB 2C          JMP SHORT canyou.0040129E
00401272  .   6A 00          PUSH 0
00401274  .   68 04304000    PUSH canyou.00403004
00401279  .   68 5B304000    PUSH canyou.0040305B
```

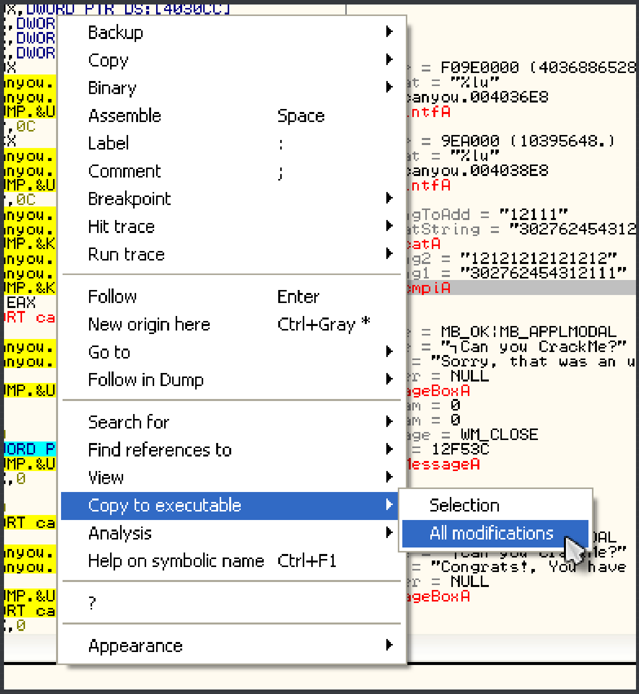
No, let’s run the app and make sure it works:



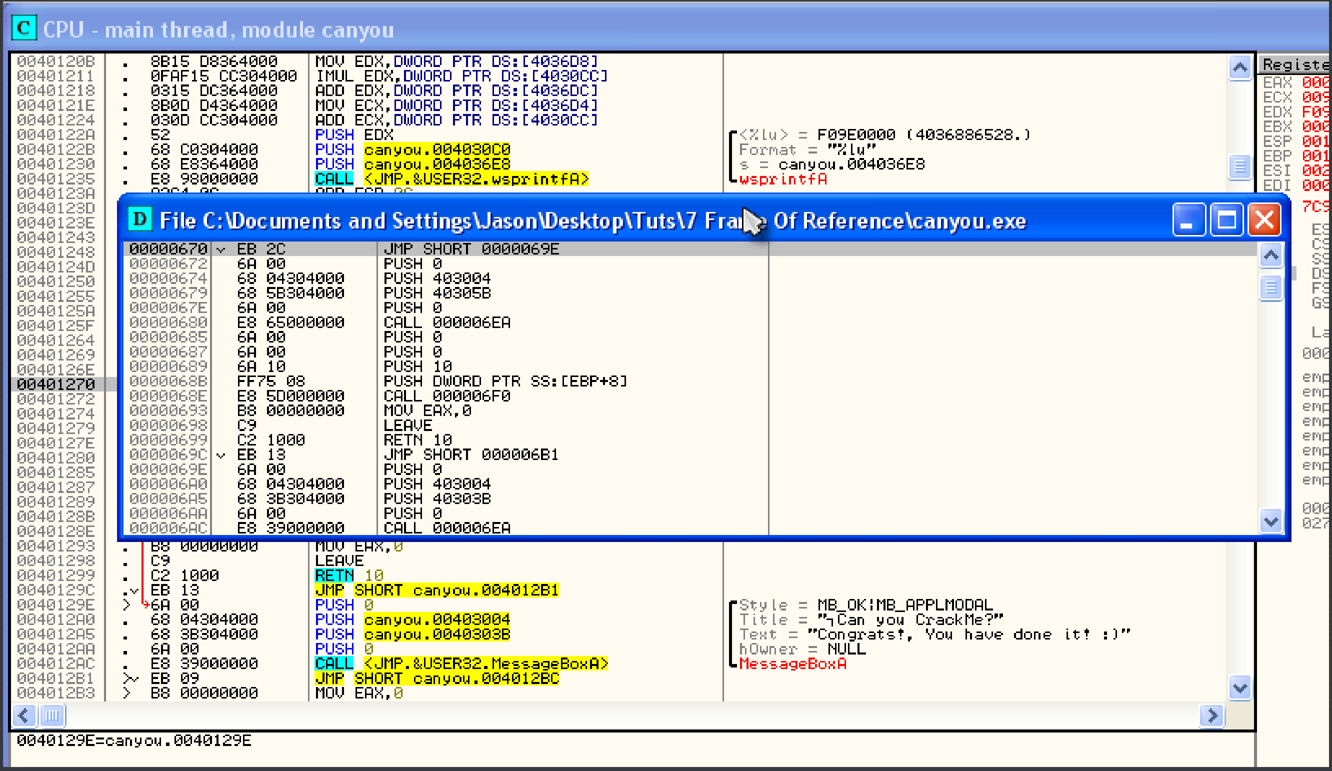
¿Can you CrackMe?

Congrats!, You have done it! :)

Now let's save this patch to disk. Remember, if you re-start the app you will have to re-enable the patch (Patch Window, highlight the patch and hit space bar), but since our app is still running, we can just click over to Olly, right click the disassembly window and choose "Copy to executable" -> "All modifications":



Choose "Save all" and our new memory process window will open (with our patch at the top of it):



CPU - main thread, module canyou

```
0040120B  .  8B15 D8364000  MOV EDX,DWORD PTR DS:[4036D8]
00401211  .  0FAF15 CC304000 IMUL EDX,DWORD PTR DS:[4030CC]
00401218  .  0315 DC364000  ADD EDX,DWORD PTR DS:[4036DC]
0040121E  .  8B0D D4364000  MOV ECX,DWORD PTR DS:[4036D4]
00401224  .  030D CC304000  ADD ECX,DWORD PTR DS:[4030CC]
0040122A  .  52             PUSH EDX                         [<%lu> = F09E0000 (4036886528.)
0040122B  .  68 C0304000    PUSH canyou.004030C0             |Format = "%lu"
00401230  .  68 E8364000    PUSH canyou.004036E8             |s = canyou.004036E8
00401235  .  E8 98000000    CALL <JMP.&USER32.wsprintfA>     [wsprintfA
0040123A
0040123D
0040123E
00401243
00401248
0040124D
00401250
00401255
0040125A
0040125F
00401264
00401269
0040126E
00401270
00401272
00401274
00401279
0040127E
00401280
00401285
00401287
00401289
0040128B
0040128E
00401293  .  B8 00000000    MOV EAX,0
00401298  .  C9             LEAVE
00401299  .  C2 1000        RETN 10
0040129C  .v EB 13          JMP SHORT canyou.004012B1
0040129E  >  6A 00          PUSH 0                           [Style = MB_OK|MB_APPLMODAL
004012A0  .  68 04304000    PUSH canyou.00403004             |Title = "¬Can you CrackMe?"
004012A5  .  68 3B304000    PUSH canyou.0040303B             |Text = "Congrats!, You have done it! :)"
004012AA  .  6A 00          PUSH 0                           |hOwner = NULL
004012AC  .  E8 39000000    CALL <JMP.&USER32.MessageBoxA>   [MessageBoxA
004012B1  >v EB 09          JMP SHORT canyou.004012BC
004012B3  >  B8 00000000    MOV EAX,0
```

```
0040129E=canyou.0040129E
```

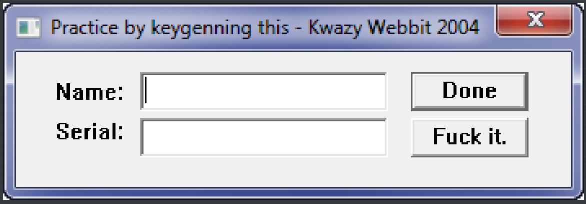
File C:\Documents and Settings\Jason\Desktop\Tuts\7 Fra Of Reference\canyou.exe

```
00000670 v EB 2C          JMP SHORT 0000069E
00000672   6A 00          PUSH 0
00000674   68 04304000    PUSH 403004
00000679   68 5B304000    PUSH 40305B
0000067E   6A 00          PUSH 0
00000680   E8 65000000    CALL 000006EA
00000685   6A 00          PUSH 0
00000687   6A 00          PUSH 0
00000689   6A 10          PUSH 10
0000068B   FF75 08        PUSH DWORD PTR SS:[EBP+8]
0000068E   E8 5D000000    CALL 000006F0
00000693   B8 00000000    MOV EAX,0
00000698   C9             LEAVE
00000699   C2 1000        RETN 10
0000069C v EB 13          JMP SHORT 000006B1
0000069E   6A 00          PUSH 0
000006A0   68 04304000    PUSH 403004
000006A5   68 3B304000    PUSH 40303B
000006AA   6A 00          PUSH 0
000006AC   E8 39000000    CALL 000006EA
```

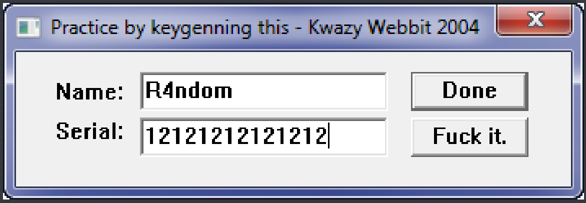
Now, let's save it to disk…right click in this new window and click "Save File". Save it as canyou_patched (or anything you want), load this new patched file back in to Olly and run it. You actually don't need to load it into Olly anymore if you don't want as the patch has been saved to disk- you can just run it from wherever the crackme is. Just make sure you run the patched version 😀 . Now, no matter what name and serial you put in, you will get the good boy screen 😀

Another Crackme

Let's now load up the second program, Crackme8.exe and run it in Olly:



Well, that certainly makes a point :0 Ummmm. After I enter my name and serial, which button do I press? Well, let me try one:



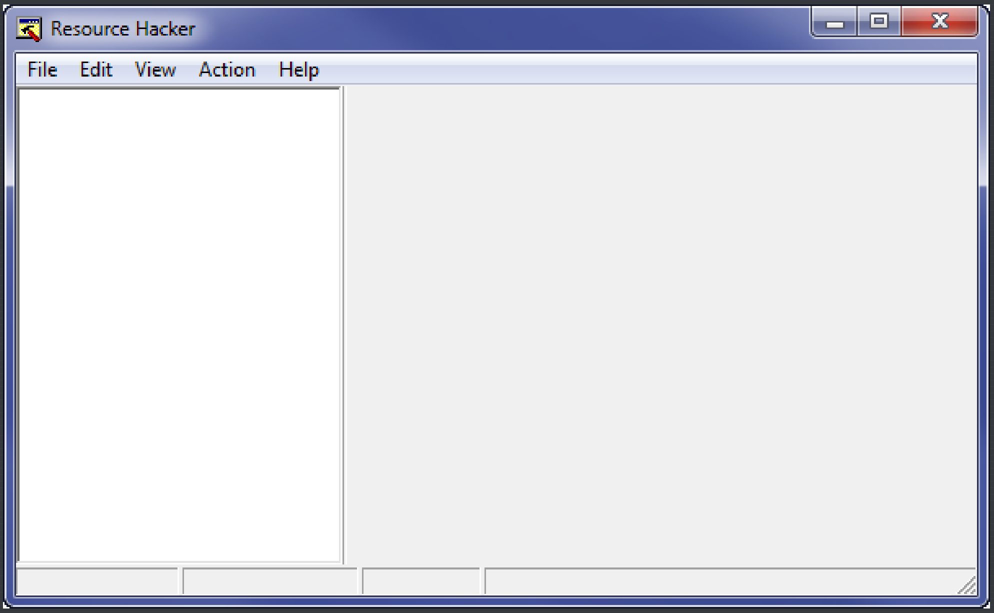
Name: R4ndom

Serial: 12121212121212

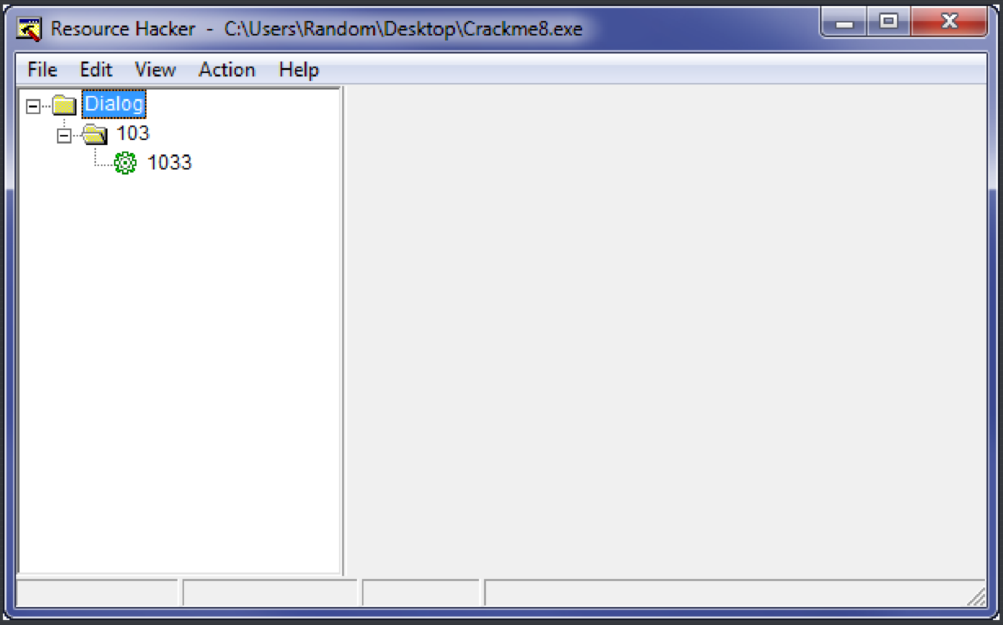
Now, Done usually means exit, so I'll try the other one. And…………, it quits the app. Obviously I should have clicked Done (?). Regardless, let's take this opportunity to modify this program and have some fun with it. Let's change the buttons to a more meaningful "Check" and "Done", or anything else you prefer 😉
.

Using Resource Hacker

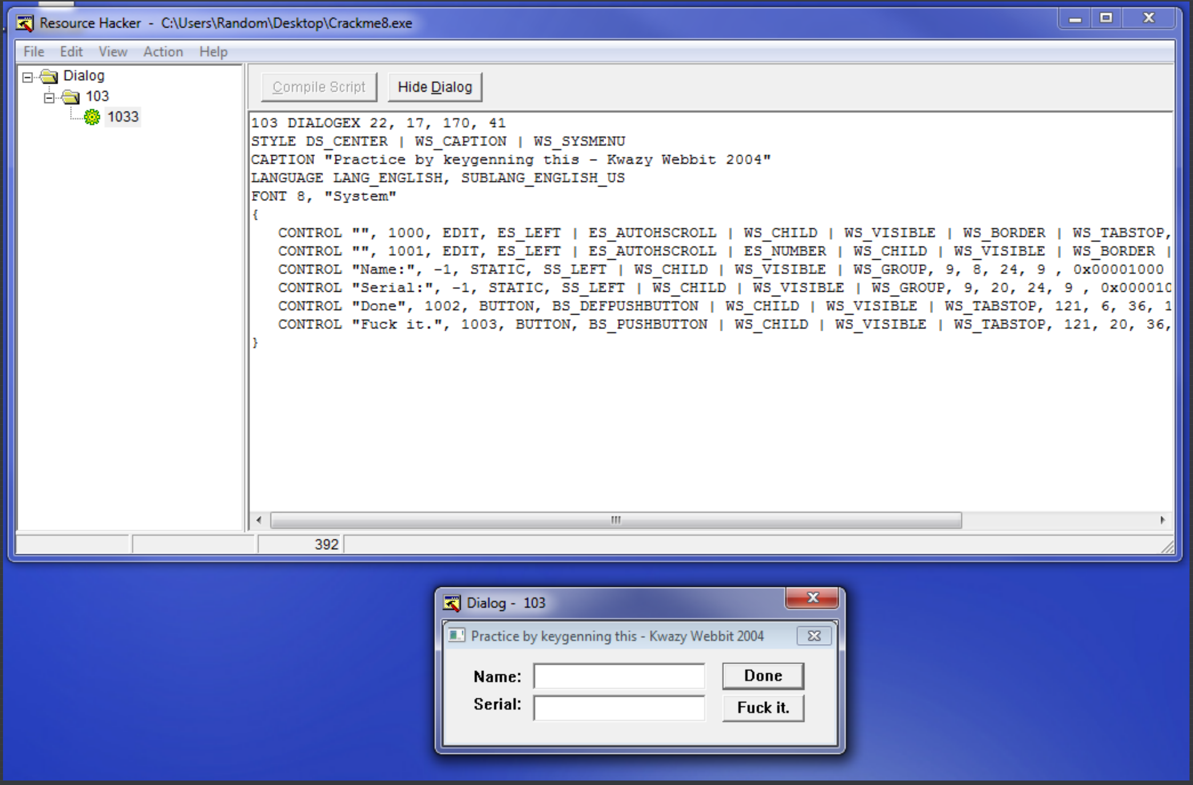
If you haven't already, go ahead and install Resource Hacker. When you first run it, it looks like this:



Go ahead and load the Crackme8 file into Resource Hacker and you should notice a folder called Dialog with a plus sign next to it. Open up the plus and click the plus next to the next folder (103) and you should see something like this:

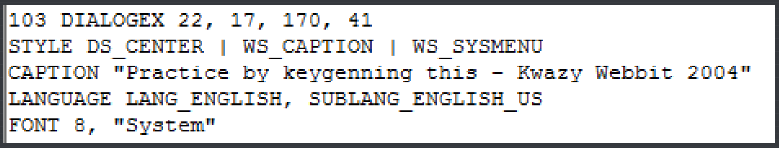

Resource Hacker - C:\Users\Random\Desktop\Crackme8.exe
File Edit View Action Help
Dialog
103
1033

Now click on the 1033 and it will populate the right pane with data about this dialog, as well as open a reference dialog showing what it looks like:


Resource Hacker - C:\Users\Random\Desktop\Crackme8.exe
File
Edit
View
Action
Help
Dialog
103
1033
Compile Script
Hide Dialog
392
Dialog - 103
Practice by keygenning this - Kwazy Webbit 2004
Name:
Serial:
Done
Fuck it.

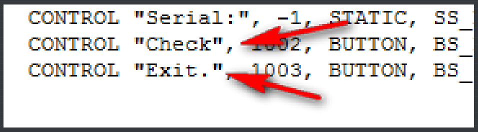
```
103 DIALOGEX 22, 17, 170, 41
STYLE DS_CENTER | WS_CAPTION | WS_SYSMENU
CAPTION "Practice by keygenning this - Kwazy Webbit 2004"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "System"
{
   CONTROL "", 1000, EDIT, ES_LEFT | ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,
   CONTROL "", 1001, EDIT, ES_LEFT | ES_AUTOHSCROLL | ES_NUMBER | WS_CHILD | WS_VISIBLE | WS_BORDER |
   CONTROL "Name:", -1, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 9, 8, 24, 9 , 0x00001000
   CONTROL "Serial:", -1, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 9, 20, 24, 9 , 0x000010
   CONTROL "Done", 1002, BUTTON, BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 121, 6, 36, 1
   CONTROL "Fuck it.", 1003, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 121, 20, 36,
}
```

At the top of the right pane you can see some generic data about the window such as the font, the caption, the styles etc:



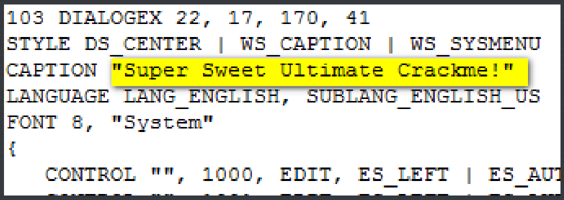
```
103 DIALOGEX 22, 17, 170, 41
STYLE DS_CENTER | WS_CAPTION | WS_SYSMENU
CAPTION "Practice by keygenning this - Kwazy Webbit 2004"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "System"
```

and underneath this you can see details about all of the elements in this dialog, including the “Name” and “Serial” labels and the two buttons. Let’s change this dialog to our own liking, shall we? First change the two button names to “Check” and “Exit”:



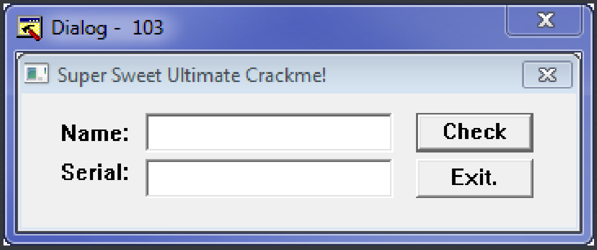
```
  CONTROL "Serial:", -1, STATIC, SS_
  CONTROL "Check", 1002, BUTTON, BS_
  CONTROL "Exit.", 1003, BUTTON, BS_
```

Now, let’s change the caption at the top:

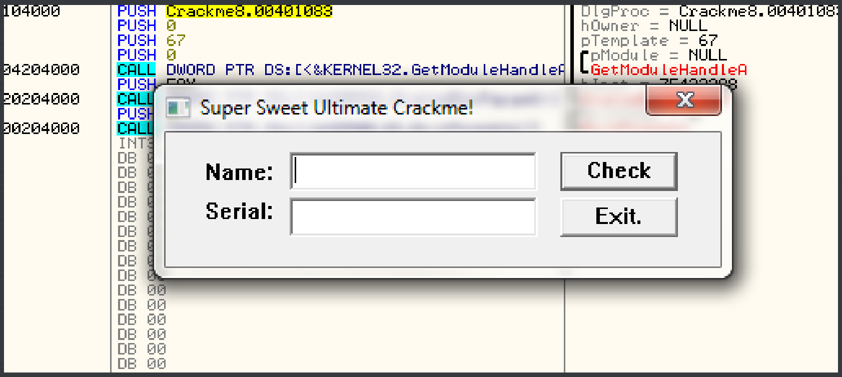


```
103 DIALOGEX 22, 17, 170, 41
STYLE DS_CENTER | WS_CAPTION | WS_SYSMENU
CAPTION "Super Sweet Ultimate Crackme!"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "System"
{
   CONTROL "", 1000, EDIT, ES_LEFT | ES_AUT
```

Now click the “Compile” button at the top and you will see our dialog update:



Well, that's better. Let's just save it ("File" -> "Save") and load this new crackme into Olly (the original crackme was saved by Resource Hacker as Crackme8_original), and run it:

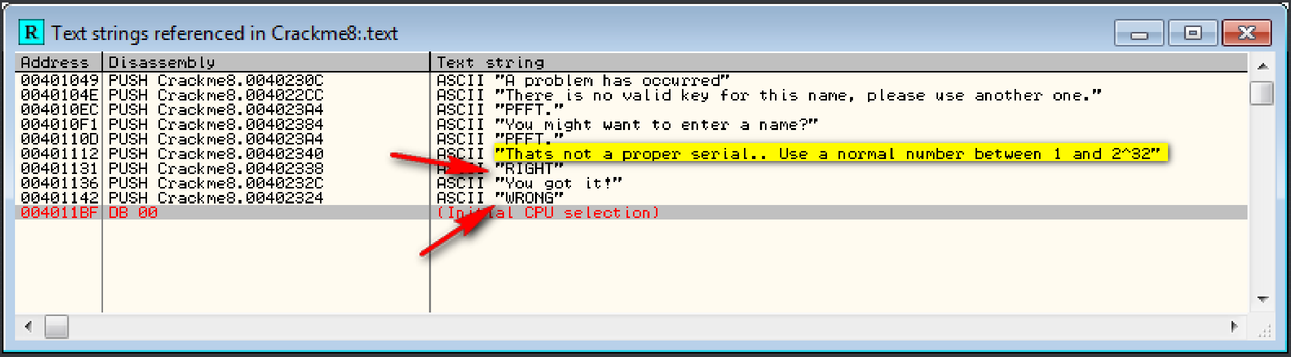


```
104000        PUSH Crackme8.00401083                          DlgProc = Crackme8.0040108
              PUSH 0                                          hOwner = NULL
              PUSH 67                                         pTemplate = 67
              PUSH 0                                          pModule = NULL
04204000      CALL DWORD PTR DS:[<&KERNEL32.GetModuleHandleA  GetModuleHandleA
              PUSH
20204000      CALL
              PUSH
00204000      CALL
              INT3
              DB 0
              DB 0
              DB 0
              DB 0
              DB 0
              DB 0
              DB 0
              DB 0
              DB 0
              DB 00
              DB 00
              DB 00
              DB 00
              DB 00
              DB 00
```

Ah, that's better. Now we can officially start…

Cracking The Program

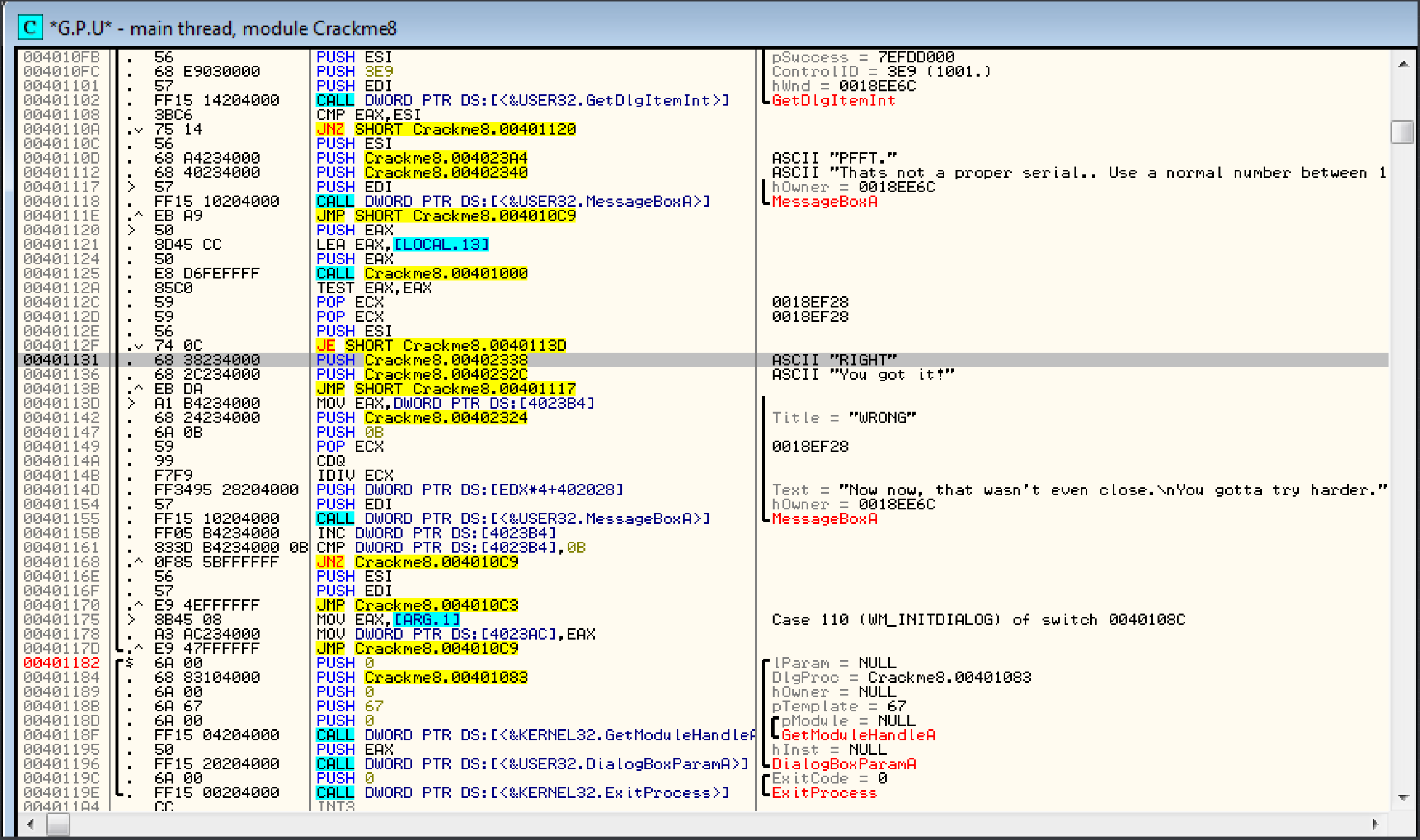
You should know the drill by now. Let's search for text strings:



Text strings referenced in Crackme8:.text

```
Address   Disassembly              Text string
00401049  PUSH Crackme8.0040230C   ASCII "A problem has occurred"
0040104E  PUSH Crackme8.004022CC   ASCII "There is no valid key for this name, please use another one."
004010EC  PUSH Crackme8.004023A4   ASCII "PFFT."
004010F1  PUSH Crackme8.00402384   ASCII "You might want to enter a name?"
0040110D  PUSH Crackme8.004023A4   ASCII "PFFT."
00401112  PUSH Crackme8.00402340   ASCII "Thats not a proper serial.. Use a normal number between 1 and 2^32"
00401131  PUSH Crackme8.00402338   ASCII "RIGHT"
00401136  PUSH Crackme8.0040232C   ASCII "You got it!"
00401142  PUSH Crackme8.00402324   ASCII "WRONG"
004011BF  DB 00                    (Initial CPU selection)
```

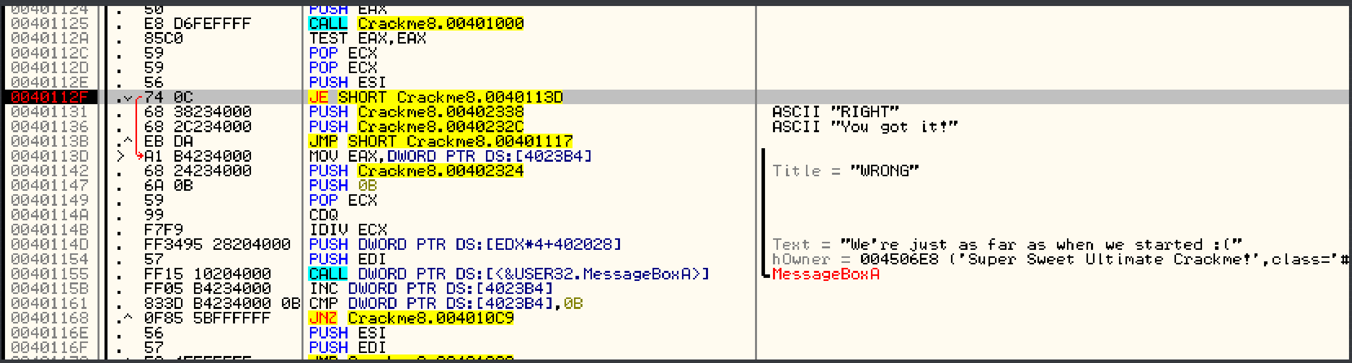
There are two things we learn; 1) the serial must be a number between 1 and a gazillion and 2) we know where the good boy and bad boy messages are being generated. Lets go to the good boy message:



G.P.U - main thread, module Crackme8

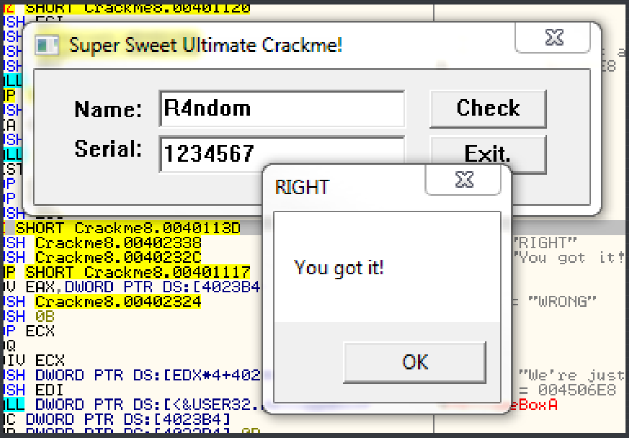
```
004010FB  .  56              PUSH ESI                                      pSuccess = 7EFDD000
004010FC  .  68 E9030000     PUSH 3E9                                      ControlID = 3E9 (1001.)
00401101  .  57              PUSH EDI                                      hWnd = 0018EE6C
00401102  .  FF15 14204000   CALL DWORD PTR DS:[<&USER32.GetDlgItemInt>]   GetDlgItemInt
00401108  .  3BC6            CMP EAX,ESI
0040110A  .v 75 14           JNZ SHORT Crackme8.00401120
0040110C  .  56              PUSH ESI
0040110D  .  68 A4234000     PUSH Crackme8.004023A4                        ASCII "PFFT."
00401112  .  68 40234000     PUSH Crackme8.00402340                        ASCII "Thats not a proper serial.. Use a normal number between 1
00401117  >  57              PUSH EDI                                      hOwner = 0018EE6C
00401118  .  FF15 10204000   CALL DWORD PTR DS:[<&USER32.MessageBoxA>]     MessageBoxA
0040111E  .^ EB A9           JMP SHORT Crackme8.004010C9
00401120  >  50              PUSH EAX
00401121  .  8D45 CC         LEA EAX,[LOCAL.13]
00401124  .  50              PUSH EAX
00401125  .  E8 D6FEFFFF     CALL Crackme8.00401000
0040112A  .  85C0            TEST EAX,EAX
0040112C  .  59              POP ECX                                       0018EF28
0040112D  .  59              POP ECX                                       0018EF28
0040112E  .  56              PUSH ESI
0040112F  .v 74 0C           JE SHORT Crackme8.0040113D
00401131  .  68 38234000     PUSH Crackme8.00402338                        ASCII "RIGHT"
00401136  .  68 2C234000     PUSH Crackme8.0040232C                        ASCII "You got it!"
0040113B  .^ EB DA           JMP SHORT Crackme8.00401117
0040113D  >  A1 B4234000     MOV EAX,DWORD PTR DS:[4023B4]
00401142  .  68 24234000     PUSH Crackme8.00402324                        Title = "WRONG"
00401147  .  6A 0B           PUSH 0B
00401149  .  59              POP ECX                                       0018EF28
0040114A  .  99              CDQ
0040114B  .  F7F9            IDIV ECX
0040114D  .  FF3495 28204000 PUSH DWORD PTR DS:[EDX*4+402028]              Text = "Now now, that wasn't even close.\nYou gotta try harder."
00401154  .  57              PUSH EDI                                      hOwner = 0018EE6C
00401155  .  FF15 10204000   CALL DWORD PTR DS:[<&USER32.MessageBoxA>]     MessageBoxA
0040115B  .  FF05 B4234000   INC DWORD PTR DS:[4023B4]
00401161  .  833D B4234000 0B CMP DWORD PTR DS:[4023B4],0B
00401168  .^ 0F85 5BFFFFFF   JNZ Crackme8.004010C9
0040116E  .  56              PUSH ESI
0040116F  .  57              PUSH EDI
00401170  .^ E9 4EFFFFFF     JMP Crackme8.004010C3
00401175  >  8B45 08         MOV EAX,[ARG.1]                               Case 110 (WM_INITDIALOG) of switch 0040108C
00401178  .  A3 AC234000     MOV DWORD PTR DS:[4023AC],EAX
0040117D  .^ E9 47FFFFFF     JMP Crackme8.004010C9
00401182  $  6A 00           PUSH 0                                        lParam = NULL
00401184  .  68 83104000     PUSH Crackme8.00401083                        DlgProc = Crackme8.00401083
00401189  .  6A 00           PUSH 0                                        hOwner = NULL
0040118B  .  6A 67           PUSH 67                                       pTemplate = 67
0040118D  .  6A 00           PUSH 0                                        pModule = NULL
0040118F  .  FF15 04204000   CALL DWORD PTR DS:[<&KERNEL32.GetModuleHandleA GetModuleHandleA
00401195  .  50              PUSH EAX                                      hInst = NULL
00401196  .  FF15 20204000   CALL DWORD PTR DS:[<&USER32.DialogBoxParamA>] DialogBoxParamA
0040119C  .  6A 00           PUSH 0                                        ExitCode = 0
0040119E  .  FF15 00204000   CALL DWORD PTR DS:[<&KERNEL32.ExitProcess>]   ExitProcess
004011A4     CC              INT3
```

and we double-click into familiar territory. We see the good boy routine beginning at 401131, and the bad boy starting at 40113D. We also see the jump (JE SHORT Crackme8.0040113D) at address 401131 and the compare (TEST EAX, EAX) at address 40112A. Let's set our BP at address 40112F and run the app. Enter a name and serial and click "Check". Olly will then break at our breakpoint:



```
00401124  .  50              PUSH EAX
00401125  .  E8 D6FEFFFF     CALL Crackme8.00401000
0040112A  .  85C0            TEST EAX,EAX
0040112C  .  59              POP ECX
0040112D  .  59              POP ECX
0040112E  .  56              PUSH ESI
0040112F  .v 74 0C           JE SHORT Crackme8.0040113D
00401131  .  68 38234000     PUSH Crackme8.00402338                        ASCII "RIGHT"
00401136  .  68 2C234000     PUSH Crackme8.0040232C                        ASCII "You got it!"
0040113B  .^ EB DA           JMP SHORT Crackme8.00401117
0040113D  >  A1 B4234000     MOV EAX,DWORD PTR DS:[4023B4]
00401142  .  68 24234000     PUSH Crackme8.00402324                        Title = "WRONG"
00401147  .  6A 0B           PUSH 0B
00401149  .  59              POP ECX
0040114A  .  99              CDQ
0040114B  .  F7F9            IDIV ECX
0040114D  .  FF3495 28204000 PUSH DWORD PTR DS:[EDX*4+402028]              Text = "We're just as far as when we started :("
00401154  .  57              PUSH EDI                                      hOwner = 004506E8 ('Super Sweet Ultimate Crackme!',class='#
00401155  .  FF15 10204000   CALL DWORD PTR DS:[<&USER32.MessageBoxA>]     MessageBoxA
0040115B  .  FF05 B4234000   INC DWORD PTR DS:[4023B4]
00401161  .  833D B4234000 0B CMP DWORD PTR DS:[4023B4],0B
00401168  .^ 0F85 5BFFFFFF   JNZ Crackme8.004010C9
0040116E  .  56              PUSH ESI
0040116F  .  57              PUSH EDI
```

We can see that, unaltered, Olly is going to jump past our good boy and straight to the bad boy. You know the routine…clear the zero flag and run the app:



Super Sweet Ultimate Crackme!

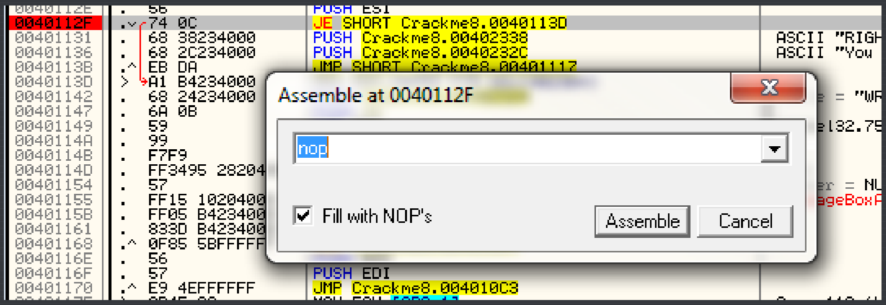
Name: R4ndom

Serial: 1234567

RIGHT

You got it!

And success. Now let's quickly create a patch: Re-start the app, go to the breakpoint (through the breakpoint window), click once on the JE instruction and hit the space bar and NOP out the jump so that we always fall through to the good boy:



```
0040112E  .   56              PUSH ESI
0040112F  .v  74 0C           JE SHORT Crackme8.0040113D
00401131  .   68 38234000     PUSH Crackme8.00402338                ASCII "RIGH
00401136  .   68 2C234000     PUSH Crackme8.0040232C                ASCII "You
0040113B  .^  EB DA           JMP SHORT Crackme8.00401117
0040113D  >   A1 B4234000
00401142  .   68 24234000                                           = "WR
00401147  .   6A 0B
00401149  .   59                                                    132.75
0040114A  .   99
0040114B  .   F7F9
0040114D  .   FF3495 28204
00401154  .   57                                                    er = NU
00401155  .   FF15 1020400                                          ageBoxA
0040115B  .   FF05 B423400
00401161  .   833D B423400
00401168  .^  0F85 5BFFFFF
0040116E  .   56
0040116F  .   57              PUSH EDI
00401170  .^  E9 4EFFFFFF     JMP Crackme8.004010C3
```

Assemble at 0040112F

nop

☑ Fill with NOP's

Hit "Assemble" then "Cancel. Right click and choose "Save to executable" -> "All modifications" and choose "copy all". Right click in the new window and select "Save file" and save it. You now have a patched and resource-altered crackme that will take any serial number you put in and display the good boy message 😃

Food For Thought

I wanted to just mention that Resource Hacker is a fun and very useful little program. It allows you to not only change many things in a file (strings, icons, labels, buttons, captions) but it also let's you change many things in Windows itself (the START button, context menus, the computer's 'About' dialog etc.) In fact, Resource Hacker is what I used to change all of the icons in my version of Olly!

-Till next time

R4ndom