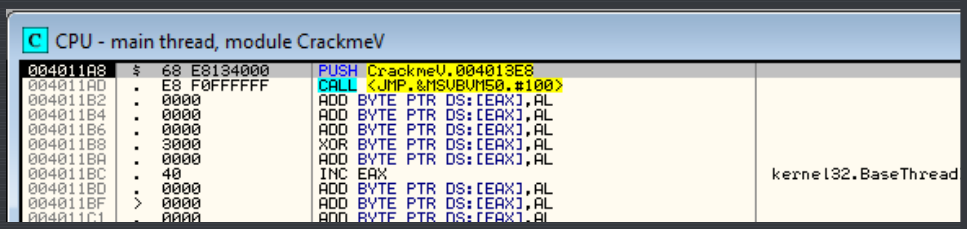
R4ndom's Tutorial #20B: Working With Visual Basic Binaries, Pt 2

by R4ndom on Sep.07, 2012, under Beginner, Reverse Engineering, Tutorials

This is part 2 of Working with Visual Basic Binaries. In this tutorial we will be using VB Decompiler which is available in the download from part 1 of this tutorial. We will also be using MapConvert and OllyVBHelper-plugins for Olly, P32Dasm and some additional crackmes, all of which are available in the download of this tutorial on the tutorials page.

Investigating CrackmeVB3

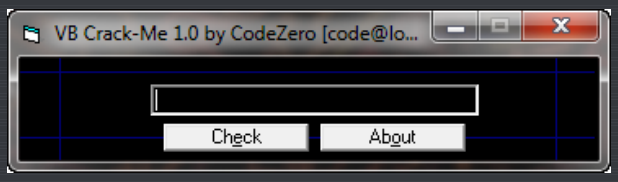
Let's begin by loading the crackme into Olly:



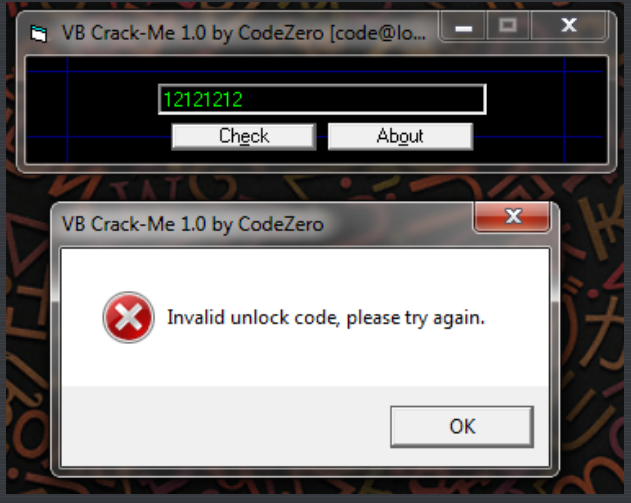
Here, we see our typical call into the VB runtime. You have to admit, it's pretty amazing (bonehead?) that we can write an entire executable in two lines of code!

Now, because we don't know much about this binary, let's investigate it. Running the target, we see that it has a nag set to a timer of 5 seconds:

We'll definitely want to get rid of that! After 5 seconds we see the main serial screen:



and entering a wrong serial, the badboy is displayed:



Let's take a look at the target's guts, shall we? This time, instead of VB Decompiler, let's use P32Dasm.

Using P32Dasm

P32Dasm is a native and P-code decompiler. It is very similar to VB Decompiler, though it does have some added benefits (like exporting MAP files that work).

Loading CrackmeVB3.exe into P32Dasm, we see the main screen, with some data about the target:

P32Dasm v2.80 - CrackmeVB3.exe

File Edit References Tools About

```
File: C:\Users\Random\Desktop\Random\Tutorials\Intro to Reverse Engineeri
P32Dasm v2.80

VB5 Application detected ... NCode


004055F4: Sub Form1.Command1_Click
004056D0: mov d,[ebp][XX], "VB Crack-Me 1.0 by CodeZero"
004056DF: mov d,[ebp][XX], "Please enter the registation code:
00405721: push     "55555"
0040575E: mov d,[ebp][XX], "VB Crack-Me 1.0 by CodeZero"
0040577B: mov d,[ebp][XX], "Congratulations! you've really made it :-)"
0040579F: mov d,[ebp][XX], "Invalid unlock code, please try again."


0040583D: Sub Form1.Command2_Click
0040588A: push     " "


00405905: Sub Form1.Form_Load
00405957: push     "??????????????????????????"
00405971: push     "  ?@"


004059C2: Sub Form2.Command1_Click
00405A87: push     "??????????????????????????"


00405AC5: Sub Form2.Timer1_Timer


00405BCE: Sub Form2.Timer2_Timer


00405CD7: Sub Form2.Timer3_Timer


00405DE0: Sub Form2.Timer4_Timer


00405EE9: Sub Form2.Timer5_Timer
```

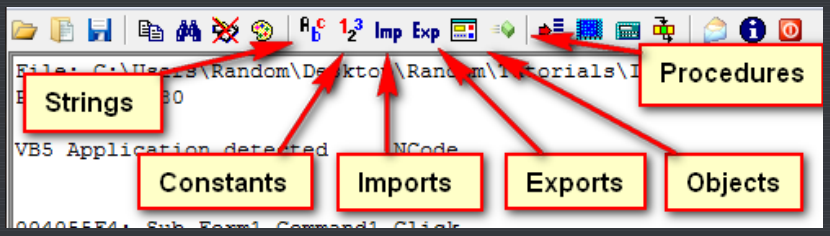
Strings

Buttons

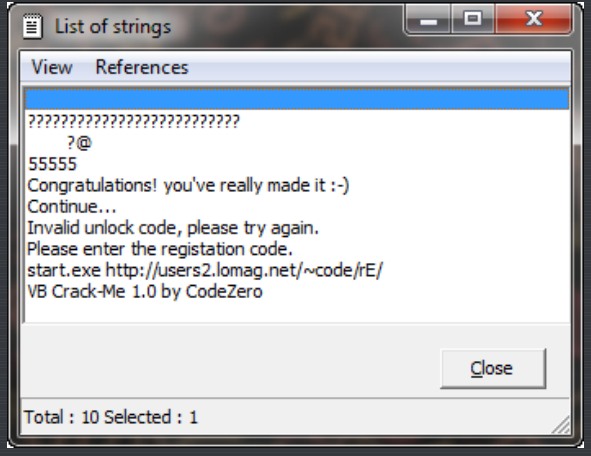
Timers

Idle | Errors: 0 Unknown: 0 Procs: 11/11 (0.1 sec)

You should notice some similarities to VB Decompiler, especially the Form1.Command2.click. This is the callback to a clicking of a button. At the top are ASCII strings used in the target (obviously not obfuscated), and at the bottom we see several timer functions. At the top of the P32Dasm window are some toolbar buttons that you should be familiar with:

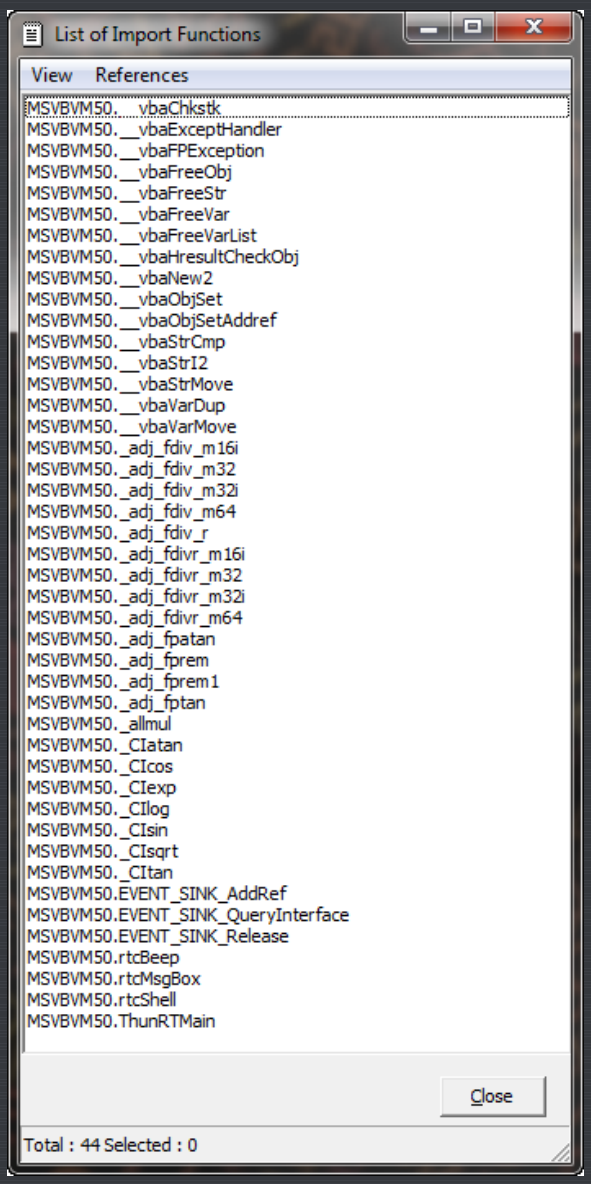


Clicking on “Strings”, we see something similar to the “Search For -> Strings” in Olly:



Though, unlike Olly, double clicking on one does not take you to the disassembly of that section of code.

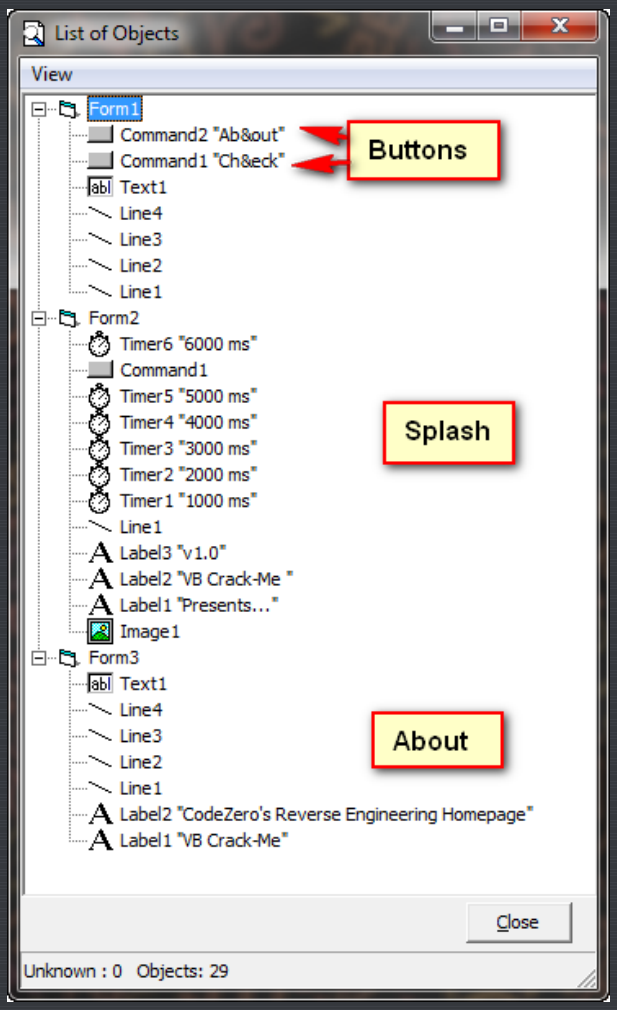
Next is the Constants toolbar button, but clicking on this reveals that there aren't any. Then we come to 'Imports', which is similar to "All intermodular calls" in Olly:



The _vbaStrCmp should stick out like a sore thumb…

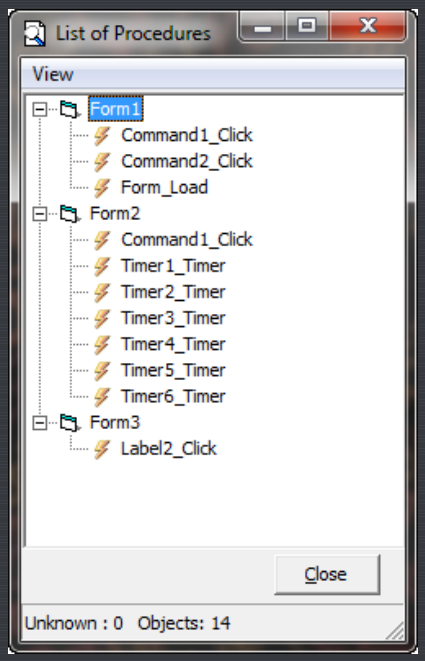
Next is the Exports, but this target doesn't have any, so it's blank.

"Objects" should remind you of the VB Decompiler screen:



This shows all of the VB 'objects' in the target, such as buttons, labels and timers. We can plainly see that the "Check" button is called "Command1", and we can assume that this is our main check button callback.

Last up is the "Procedures" window:



This shows us all of the callbacks in the target. We can see that our check button callback is called

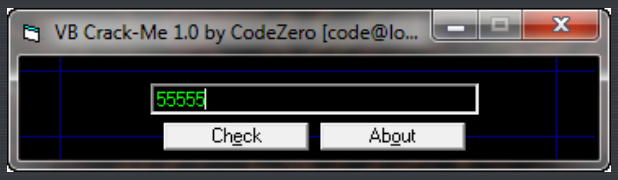
"Command1_Click", as Command1 is the callback of our "Check" button.

One thing I wanted to point out is that very suspicious series of five's in the strings section:

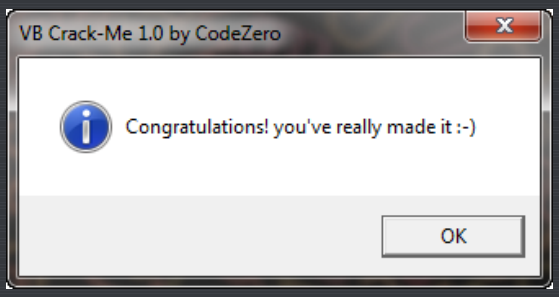
```
VB5 Application detected ... NCode


004055F4: Sub Form1.Command1_Cli
004056D0: mov d,[ebp][XX], "VB Crack-Me 1.0 by CodeZero"
004056DF: mov d,[ebp][XX], "Please enter the registation code."
00405721: push     "55555"
0040575E: mov d,[ebp][XX], "VB Crack-Me 1.0 by CodeZero"
0040577B: mov d,[ebp][XX], "Congratulations! you've really made it :-)"
0040579F: mov d,[ebp][XX], "Invalid unlock code, please try again."


0040583D: Sub Form1.Command2_Click
```

?????

It couldn't really be that simple, could it?



No, please tell me it isn't:



Oh brother. Oh well, let's continue to examine this crackme so we can see how using P32Dasm will help us in other ways (like removing that nag). One tool we have is map files…

Making a MAP File

A MAP file is a collection of names for procedure calls that have been compiled into the VB code. Remember, VB uses actual string names to reference callbacks, so we can extrapolate these and import them into Olly. We can do this in VB Decompiler Pro (File -> Save Procedure List) but since the pro version is not free, we can also use P32Dasm. Open the target in P32Dasm again and select "File" -> "Export to MAP file". Save the MAP file and then load CrackmeVB3.exe in Olly. Let's take a look at our main suspicious callback before loading the map file. I chose the address of the Command1_Click callback at 4055F4:

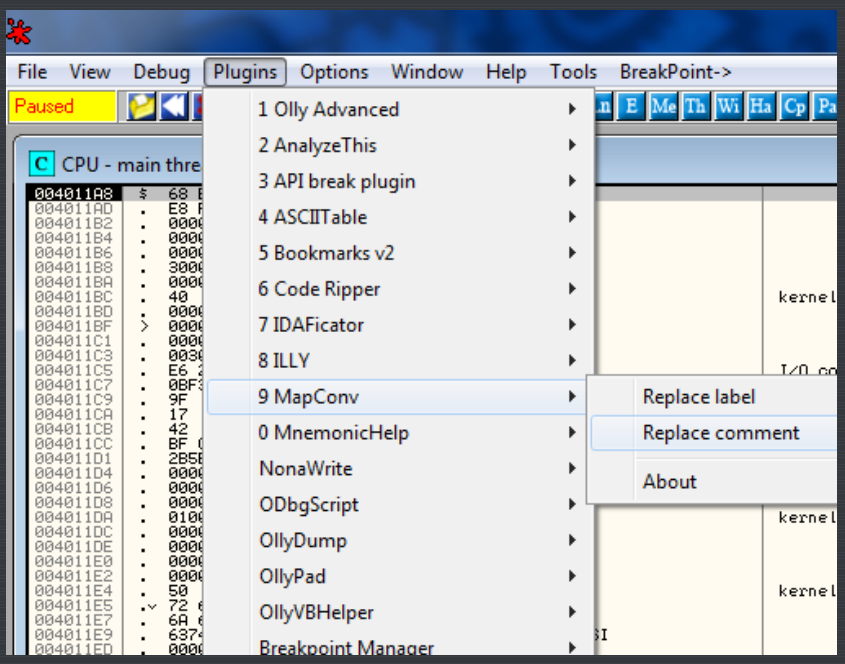
```
004055F4: Sub Form1.Command1_Click
004056D0: mov d,[ebp][XX], "VB Crack-Me 1.0 by CodeZero"
004056DF: mov d,[ebp][XX], "Please enter the registation code."
00405721: push     "55555"
0040575E: mov d,[ebp][XX], "VB Crack-Me 1.0 by CodeZero"
0040577B: mov d,[ebp][XX], "Congratulations! you've really made it :-)"
0040579F: mov d,[ebp][XX], "Invalid unlock code, please try again."
```

Loading the target in Olly and jumping to that address, we see where the Check button callback begins:

```
004055F4  > 55                PUSH EBP
004055F5  . 8BEC              MOV EBP,ESP
004055F7  . 83EC 0C           SUB ESP,0C
004055FA  . 68 A6104000       PUSH <JMP.&MSVBVM50.__vbaExceptHandle   SE handler installation
004055FF  . 64:A1 00000000    MOV EAX,DWORD PTR FS:[0]
00405605  . 50                PUSH EAX                                kernel32.BaseThreadInitThunk
00405606  . 64:8925 00000000  MOV DWORD PTR FS:[0],ESP
0040560D  . 81EC 9C000000     SUB ESP,9C
00405613  . 53                PUSH EBX
00405614  . 56                PUSH ESI
00405615  . 57                PUSH EDI
00405616  . 8B7D 08           MOV EDI,DWORD PTR SS:[EBP+8]
00405619  . 8BC7              MOV EAX,EDI
0040561B  . 83E7 FE           AND EDI,FFFFFFFE
0040561E  . 8965 F4           MOV DWORD PTR SS:[EBP-C],ESP
00405621  . 83E0 01           AND EAX,1
00405624  . 8B37              MOV ESI,DWORD PTR DS:[EDI]
00405626  . C745 F8 00104000  MOV DWORD PTR SS:[EBP-8],CrackmeV.004
0040562D  . 57                PUSH EDI
0040562E  . 8945 FC           MOV DWORD PTR SS:[EBP-4],EAX            kernel32.BaseThreadInitThunk
00405631  . 897D 08           MOV DWORD PTR SS:[EBP+8],EDI
00405634  . FF56 04           CALL DWORD PTR DS:[ESI+4]
00405637  . 8BB6 04030000     MOV ESI,DWORD PTR DS:[ESI+304]
0040563D  . 33DB              XOR EBX,EBX
0040563F  . 57                PUSH EDI
00405640  . 895D E8           MOV DWORD PTR SS:[EBP-18],EBX
00405643  . 895D E4           MOV DWORD PTR SS:[EBP-1C],EBX
00405646  . 895D D4           MOV DWORD PTR SS:[EBP-2C],EBX
00405649  . 895D C4           MOV DWORD PTR SS:[EBP-3C],EBX
```

Beginning of Command1 callback

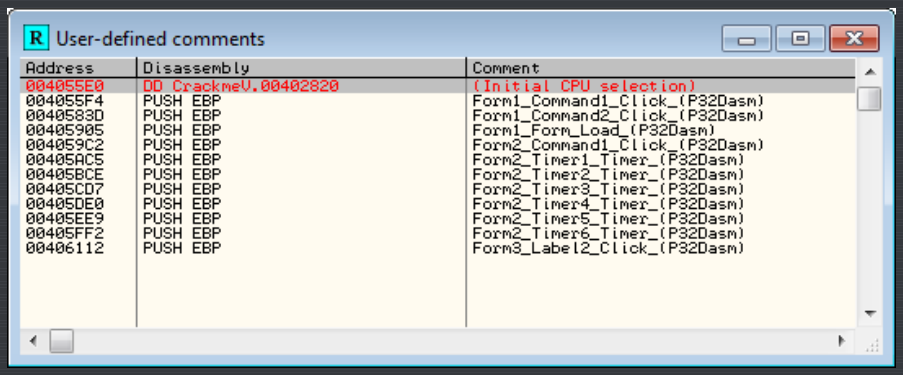
We will use the MAP converter plugin for Olly (included in the download) in order to bring in the MAP file we created in P32Dasm. Save the DLL for the MapConv plugin in the plugins directory for Olly and restart Olly (if you haven't already). Load the target and select "Plugins" -> "MapConv" -> "Replace Comments":



Select the MAP file we created in P32Dasm. This allows us to put the MAP file info in the comments column. You can also load them into the label column, though I personally find this harder to read. Now, when we look at the callback, we see that the callback information has been added to help us out:

```
004055F0      E9              DB E9
004055F1      E9              DB E9
004055F2      E9              DB E9
004055F3      E9              DB E9
004055F4  > 55                PUSH EBP                                Form1_Command1_Click_(P32Dasm)
004055F5  . 8BEC              MOV EBP,ESP
004055F7  . 83EC 0C           SUB ESP,0C
004055FA  . 68 A6104000       PUSH <JMP.&MSVBVM50.__vbaExceptHandle   SE handler installation
004055FF  . 64:A1 00000000    MOV EAX,DWORD PTR FS:[0]
00405605  . 50                PUSH EAX                                kernel32.BaseThreadInitThunk
00405606  . 64:8925 00000000  MOV DWORD PTR FS:[0],ESP
0040560D  . 81EC 9C000000     SUB ESP,9C
00405613  . 53                PUSH EBX
00405614  . 56                PUSH ESI
00405615  . 57                PUSH EDI
00405616  . 8B7D 08           MOV EDI,DWORD PTR SS:[EBP+8]
00405619  . 8BC7              MOV EAX,EDI
0040561B  . 83E7 FE           AND EDI,FFFFFFFE
0040561E  . 8965 F4           MOV DWORD PTR SS:[EBP-C],ESP
00405621  . 83E0 01           AND EAX,1
00405624  . 8B37              MOV ESI,DWORD PTR DS:[EDI]
00405626  . C745 F8 00104000  MOV DWORD PTR SS:[EBP-8],CrackmeV.004
0040562D  . 57                PUSH EDI
0040562E  . 8945 FC           MOV DWORD PTR SS:[EBP-4],EAX            kernel32.BaseThreadInitThunk
00405631  . 897D 08           MOV DWORD PTR SS:[EBP+8],EDI
00405634  . FF56 04           CALL DWORD PTR DS:[ESI+4]
00405637  . 8BB6 04030000     MOV ESI,DWORD PTR DS:[ESI+304]
0040563D  . 33DB              XOR EBX,EBX
0040563F  . 57                PUSH EDI
00405640  . 895D E8           MOV DWORD PTR SS:[EBP-18],EBX
00405643  . 895D E4           MOV DWORD PTR SS:[EBP-1C],EBX
00405646  . 895D D4           MOV DWORD PTR SS:[EBP-2C],EBX
00405649  . 895D C4           MOV DWORD PTR SS:[EBP-3C],EBX
0040564C  . 895D B4           MOV DWORD PTR SS:[EBP-4C],EBX
0040564F  . 895D A4           MOV DWORD PTR SS:[EBP-5C],EBX
```

As you can see, we now have a comment for our callback. Now right click and select “Search for” -> “All user-defined comments”:

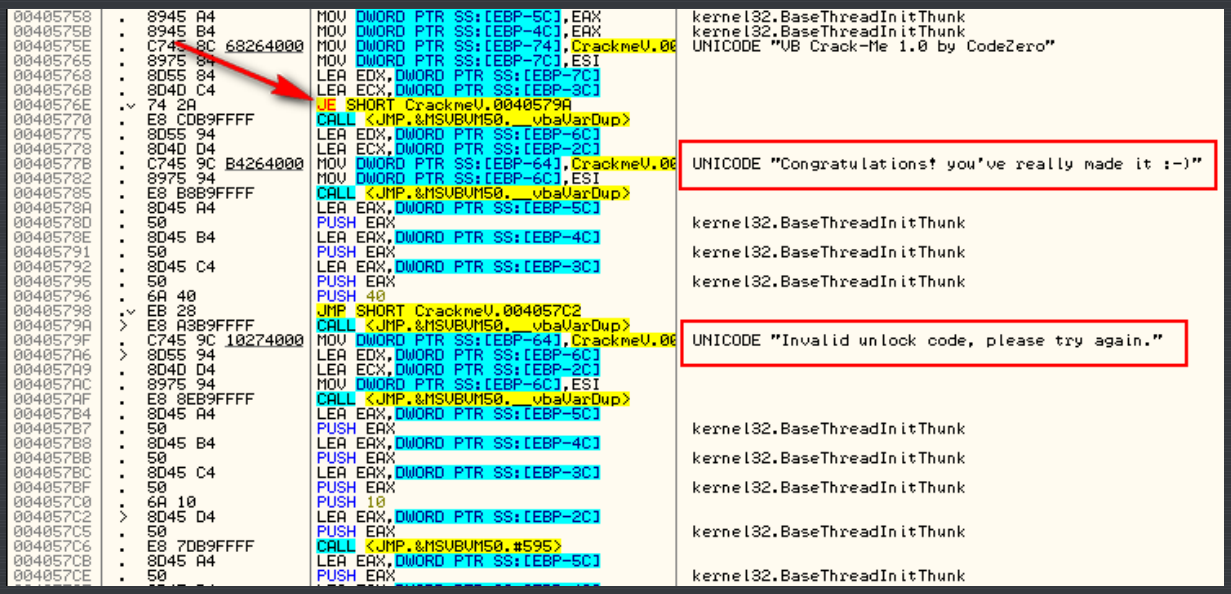


User-defined comments

Address	Disassembly	Comment
004055E0	DD CrackmeV.00402820	(Initial CPU selection)
004055F4	PUSH EBP	Form1_Command1_Click_(P32Dasm)
0040583D	PUSH EBP	Form1_Command2_Click_(P32Dasm)
00405905	PUSH EBP	Form1_Form_Load_(P32Dasm)
004059C2	PUSH EBP	Form2_Command1_Click_(P32Dasm)
00405AC5	PUSH EBP	Form2_Timer1_Timer_(P32Dasm)
00405BCE	PUSH EBP	Form2_Timer2_Timer_(P32Dasm)
00405CD7	PUSH EBP	Form2_Timer3_Timer_(P32Dasm)
00405DE0	PUSH EBP	Form2_Timer4_Timer_(P32Dasm)
00405EE9	PUSH EBP	Form2_Timer5_Timer_(P32Dasm)
00405FF2	PUSH EBP	Form2_Timer6_Timer_(P32Dasm)
00406112	PUSH EBP	Form3_Label2_Click_(P32Dasm)

Now we can see all the names of our callbacks!

Scrolling down in the Command1_Click callback, we can see the goodboy, badboy, and the obvious patch that needs to be made:



```
00405758  .   8945 A4          MOV DWORD PTR SS:[EBP-5C],EAX          kernel32.BaseThreadInitThunk
0040575B  .   8945 B4          MOV DWORD PTR SS:[EBP-4C],EAX          kernel32.BaseThreadInitThunk
0040575E  .   C745 8C 68264000 MOV DWORD PTR SS:[EBP-74],CrackmeV.00  UNICODE "VB Crack-Me 1.0 by CodeZero"
00405765  .   8975 84          MOV DWORD PTR SS:[EBP-7C],ESI
00405768  .   8D55 84          LEA EDX,DWORD PTR SS:[EBP-7C]
0040576B  .   8D4D C4          LEA ECX,DWORD PTR SS:[EBP-3C]
0040576E  .v  74 2A            JE SHORT CrackmeV.0040579A
00405770  .   E8 CDB9FFFF      CALL <JMP.&MSVBVM50.__vbaVarDup>
00405775  .   8D55 94          LEA EDX,DWORD PTR SS:[EBP-6C]
00405778  .   8D4D D4          LEA ECX,DWORD PTR SS:[EBP-2C]
0040577B  .   C745 9C B4264000 MOV DWORD PTR SS:[EBP-64],CrackmeV.00  UNICODE "Congratulations! you've really made it :-)"
00405782  .   8975 94          MOV DWORD PTR SS:[EBP-6C],ESI
00405785  .   E8 B8B9FFFF      CALL <JMP.&MSVBVM50.__vbaVarDup>
0040578A  .   8D45 A4          LEA EAX,DWORD PTR SS:[EBP-5C]
0040578D  .   50               PUSH EAX                               kernel32.BaseThreadInitThunk
0040578E  .   8D45 B4          LEA EAX,DWORD PTR SS:[EBP-4C]
00405791  .   50               PUSH EAX                               kernel32.BaseThreadInitThunk
00405792  .   8D45 C4          LEA EAX,DWORD PTR SS:[EBP-3C]
00405795  .   50               PUSH EAX                               kernel32.BaseThreadInitThunk
00405796  .   6A 40            PUSH 40
00405798  .v  EB 28            JMP SHORT CrackmeV.004057C2
0040579A  >   E8 A3B9FFFF      CALL <JMP.&MSVBVM50.__vbaVarDup>
0040579F  .   C745 9C 10274000 MOV DWORD PTR SS:[EBP-64],CrackmeV.00  UNICODE "Invalid unlock code, please try again."
004057A6  >   8D55 94          LEA EDX,DWORD PTR SS:[EBP-6C]
004057A9  .   8D4D D4          LEA ECX,DWORD PTR SS:[EBP-2C]
004057AC  .   8975 94          MOV DWORD PTR SS:[EBP-6C],ESI
004057AF  .   E8 8EB9FFFF      CALL <JMP.&MSVBVM50.__vbaVarDup>
004057B4  .   8D45 A4          LEA EAX,DWORD PTR SS:[EBP-5C]
004057B7  .   50               PUSH EAX                               kernel32.BaseThreadInitThunk
004057B8  .   8D45 B4          LEA EAX,DWORD PTR SS:[EBP-4C]
004057BB  .   50               PUSH EAX                               kernel32.BaseThreadInitThunk
004057BC  .   8D45 C4          LEA EAX,DWORD PTR SS:[EBP-3C]
004057BF  .   50               PUSH EAX                               kernel32.BaseThreadInitThunk
004057C0  .   6A 10            PUSH 10
004057C2  >   8D45 D4          LEA EAX,DWORD PTR SS:[EBP-2C]
004057C5  .   50               PUSH EAX                               kernel32.BaseThreadInitThunk
004057C6  .   E8 7DB9FFFF      CALL <JMP.&MSVBVM50.#595>
004057CB  .   8D45 A4          LEA EAX,DWORD PTR SS:[EBP-5C]
004057CE  .   50               PUSH EAX                               kernel32.BaseThreadInitThunk
```

Now all that’s left is…

Removing the Nag

Looking back in P32Dasm, let’s take a look at the timer calls:

P32Dasm v2.80 - CrackmeVB3.exe

File Edit References Tools About

```
0040579F: mov d,[ebp][XX], "Invalid unlock code, please try again."


0040583D: Sub Form1.Command2_Click
0040588A: push     " "


00405905: Sub Form1.Form_Load
00405957: push     "??????????????????????????"
00405971: push     "  ?@"

004059C2: Sub Form2.Command1_Click
00405A87: push     "??????????????????????????"


00405AC5: Sub Form2.Timer1_Timer


00405BCE: Sub Form2.Timer2_Timer


00405CD7: Sub Form2.Timer3_Timer


00405DE0: Sub Form2.Timer4_Timer


00405EE9: Sub Form2.Timer5_Timer


00405FF2: Sub Form2.Timer6_Timer
0040604B: push     "Continue..."


00406112: Sub Form3.Label2_Click
00406165: mov d,[ebp][XX], "start.exe http://users2.lomag.net/~code/rE/"



Other details:

File processed OK.
```

Nag button

Form 2

Idle | Errors: 0 Unknown: 0 Procs: 11/11 (0.09 sec)

From this screen, we can see that Form2 is the nag screen (because it is the one calling the timers).

****You may wonder why we call a timer 6 times; this is because obviously the person who created this crackme did not know how to properly use timers, so they call a one second timer 6 times to mimic a six-second countdown. ****

We can also see from this screen that Form2.Command1_Click is the callback for clicking the OK button after the timer has expired. A very simple solution (though probably not the most elegant) is to simply override the first timer call and make it jump to the callback for the OK button being clicked. This means that when the first callback for the timer is called (meaning we just started the target and we are starting the first one-second timer), we will instead call the code that handles the clicking of the OK button after the timer has expired. This will in effect simply fool the program into calling the 'close nag screen' code instead of the 'start first timer' code.

Looking at the first timer callback code at address 405AC5:

```
00405AB8  .  5E              POP ESI                                  kernel32.7491339A
00405AB9  .  64:890D 00000000 MOV DWORD PTR FS:[0],ECX
00405AC0  .  5B              POP EBX                                  kernel32.7491339A
00405AC1  .  C9              LEAVE
00405AC2  .  C2 0400         RETN 4
00405AC5     55              PUSH EBP
00405AC6     8BEC            MOV EBP,ESP
00405AC8     83EC 0C         SUB ESP,0C
00405ACB  .  68 A6104000     PUSH <JMP.&MSVBVM50.__vbaExceptHandler>  SE handler installation
00405AD0  .  64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
00405AD6  .  50              PUSH EAX                                 kernel32.BaseThreadInitThunk
00405AD7  .  64:8925 00000000 MOV DWORD PTR FS:[0],ESP
00405ADE  .  83EC 18         SUB ESP,18
00405AE1  .  53              PUSH EBX
00405AE2  .  56              PUSH ESI
00405AE3  .  57              PUSH EDI
00405AE4  .  8B7D 08         MOV EDI,DWORD PTR SS:[EBP+8]
00405AE7  .  8BC7            MOV EAX,EDI
00405AE9  .  83E7 FE         AND EDI,FFFFFFFE
00405AEC  .  8965 F4         MOV DWORD PTR SS:[EBP-C],ESP
00405AEF  .  83E0 01         AND EAX,1
00405AF2  .  8B37            MOV ESI,DWORD PTR DS:[EDI]
00405AF4  .  C745 F8 30104000 MOV DWORD PTR SS:[EBP-8],CrackmeV.00401030
00405AFB  .  57              PUSH EDI
00405AFC  .  8945 FC         MOV DWORD PTR SS:[EBP-4],EAX             kernel32.BaseThreadInitThunk
00405AFF  .  897D 08         MOV DWORD PTR SS:[EBP+8],EDI
00405B02  .  FF56 04         CALL DWORD PTR DS:[ESI+4]
00405B05  .  8365 E8 00      AND DWORD PTR SS:[EBP-18],0
00405B09  .  8365 E4 00      AND DWORD PTR SS:[EBP-1C],0
00405B0D  .  57              PUSH EDI
```

Beginning of first timer

Let's just change the beginning to point directly to the code that handles the closing of the nag screen. From P32Dasm, we can see that this callback is at address 4059C2 (the Command1_Click callback code):

```
0040583D: Sub Form1.Command2_Click
0040588A: push      " "


00405905: Sub Form1.Form_Load
00405957: push      "??????????????????????????"
00405971: push      "   ?@"


004059C2: Sub Form2.Command1_Click
00405A87: push      "??????????????????????????"


00405AC5: Sub Form2.Timer1_Timer
```

Now, let's patch it:

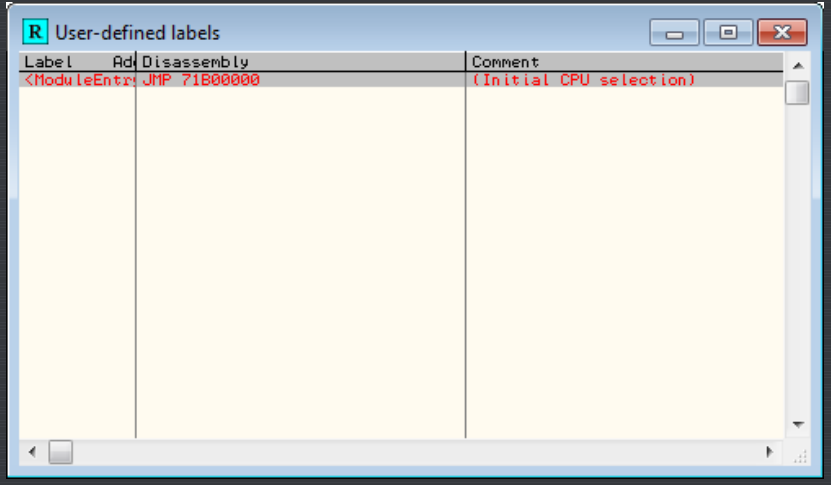
```
00405AB9  .  64:890D 00000000 MOV DWORD PTR FS:[0],ECX
00405AC0  .  5B              POP EBX
00405AC1  .  C9              LEAVE
00405AC2  .  C2 0400         RETN 4
00405AC5  ^  E9 F8FEFFFF     JMP CrackmeV.004059C2
00405ACA     90              NOP
00405ACB  .  68 A6104000     PUSH <JMP.&MSVBVM50.__vbaExceptHandler>
00405AD0  .  64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
00405AD6  .  50              PUSH EAX                                 kernel32.BaseThreadInitThunk
00405AD7  .  64:8925 00000000 MOV DWORD PTR FS:[0],ESP
00405ADE  .  83EC 18         SUB ESP,18
00405AE1  .  53              PUSH EBX
00405AE2  .  56              PUSH ESI
00405AE3  .  57              PUSH EDI
00405AE4  .  8B7D 08         MOV EDI,DWORD PTR SS:[EBP+8]
00405AE7  .  8BC7            MOV EAX,EDI
00405AE9  .  83E7 FE         AND EDI,FFFFFFFE
00405AEC  .  8965 F4         MOV DWORD PTR SS:[EBP-C],ESP
00405AEF  .  83E0 01         AND EAX,1
```

Now we will jump to closing the nag

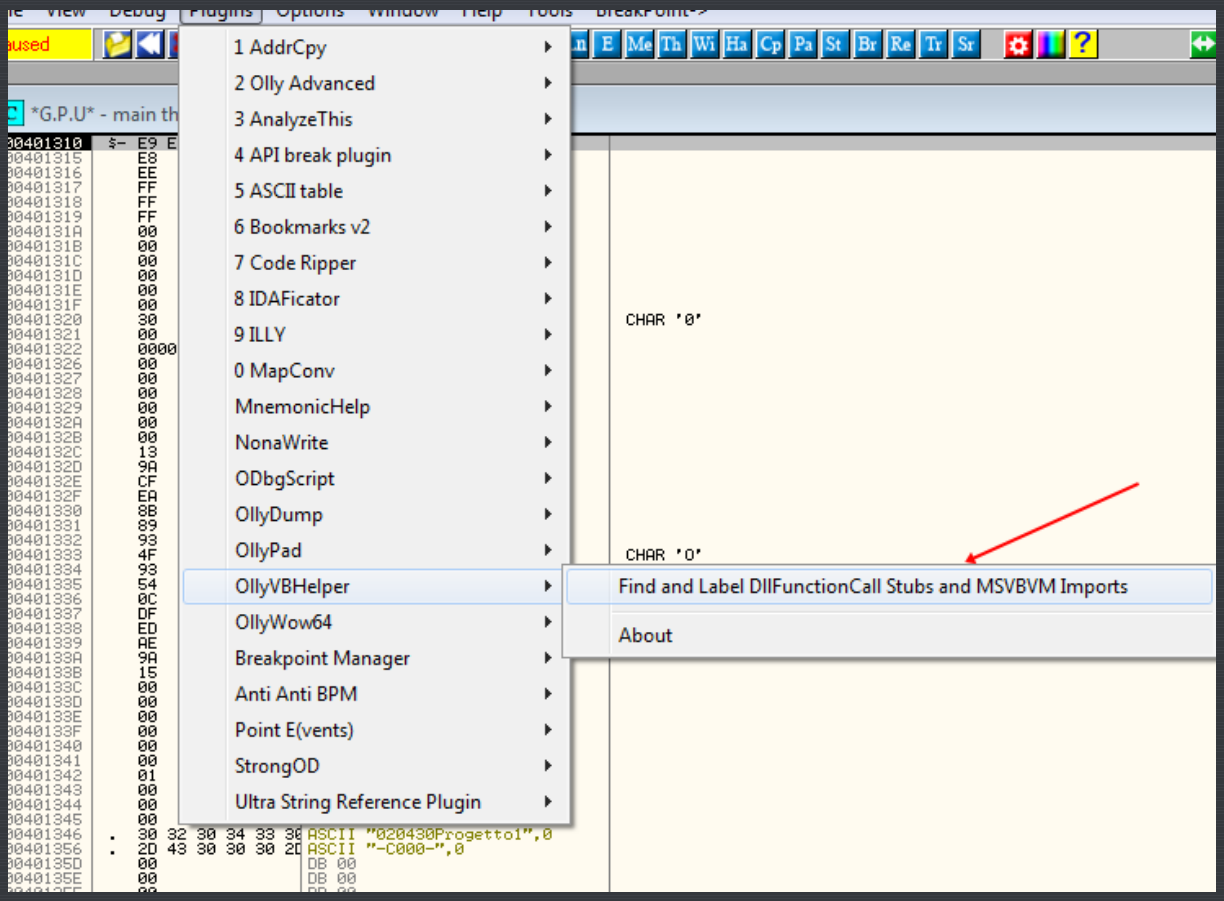
Now, if you re-start the target, you will see the nag show up for just a quick second, then disappear. Like I said, not the most elegant, but this tutorial is not about "elegantly removing nags" 😃

OllyVBHelper Plugin

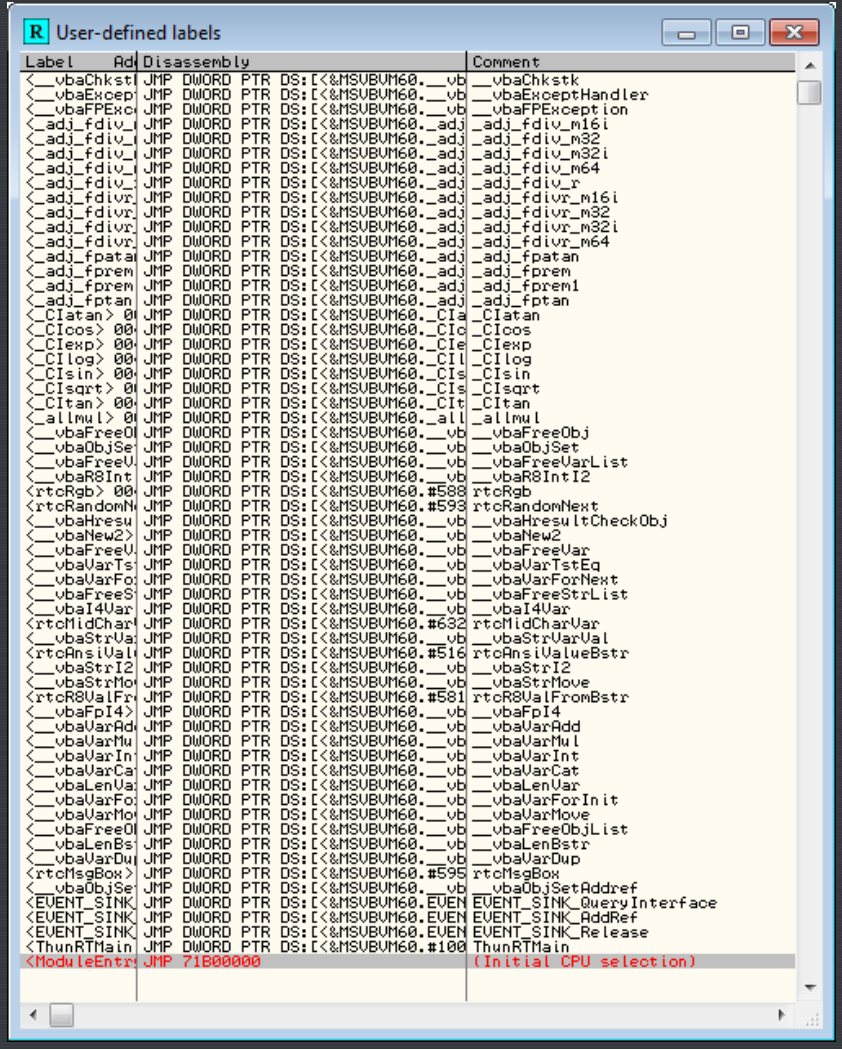
Another helpful tool in the battle with Visual Basic is the OllyVBHelper plugin. The purpose of this plugin is to find and re-label natively compiled VB imports (DLLs). It also finds and renames DLL function call stubs. As an example, I loaded one of the crackmes in Olly (doesn't matter which one) and chose "Search for" -> "All user labels". Before running the plugin, his window is empty:



Now, running the plugin:



and we can see that we now have all of our method calls displayed, similar to if we imported a MAP file.



A nice little time-saver.

Using the ‘Point-H’ Technique

Point-H is a technique introduced by Ricardo Narvaja in his cracking tutorials (in Spanish). The ‘H’ stands for Hmemcopy, a very old Windows 95 API call. This call was run directly by the OS for copying ASCII strings, and could be used to find points of interest in cracking. The Point-H is a more modern way of achieving this.

In ntdll32.dll, there is an API that Windows calls when it wants to copy a string. This function is used often, from copying names of imported DLLs, to comparing Windows messages, to API functions like GetDlgItemTextA and SetDlgItemTextA. In the later case, we can use this API to trap when a username or password is initially copied from the window and returned to our program. For example, in our crackme there may be a section that, after clicking the “Sign in” button, our program get’s the entered password from the password field by calling GetDlgItemTextA. When we call this function, kernel32 calls it’s own internal API that copies this value from the window into a temporary variable. Kernel32 then returns this value to our program as a return value of GetDlgItemTextA.

If we know the location of this internal string copying API, we can pause at it, check what string is being copied, and if it is one we’re interested in (like a password) we can then follow execution until it returns back to our target program’s code and we will see where the target receives the password.

The Point-H location will be the same on my computer no matter which target I am running, but will be a different location from another computer’s address, therefore, when we find the address of Point-H on my system, you will have to substitute your own address on your system.

This method is especially useful on targets that are heavily obfuscated, encrypted, or just too hard to find the right code section to start with (like Visual Basic executables). It gives you at least a starting point to begin at…

I have included a crackme with the files of this tutorial called “Point-H Crackme.exe”. It is not a VB target,

as we will just use this to find the actual address of this point-h API call.

Finding ‘Point-H’

First, I would suggest doing these steps with a clean install of Olly (just move the ‘plugins’ folder in the Olly install directory temporarily to your desktop) as I have had problems with certain plugins and false breaks in the past. Load the Point-H Crackme into Olly:

```
C *G.P.U* - main thread, module Point-H_
00401000  $- E9 FBEF6F71     JMP 71B00000
00401005    00               DB 00
0040100C    00A3 CA204000    ADD BYTE PTR DS:[EBX+4020CA],AH
0040100E    6A 00            PUSH 0                          Title = NULL
00401013    68 F4204000      PUSH Point-H_.004020F4          Class = "No need to disasm the cod
00401018    E8 A6040000      CALL <JMP.&USER32.FindWindowA>  FindWindowA
0040101A    0BC0             OR EAX,EAX                      kernel32.BaseThreadInitThunk
0040101C  .v 74 01           JE SHORT Point-H_.0040101D
0040101D    C3               RETN
0040101E    C705 64204000 034 MOV DWORD PTR DS:[402064],4003
00401027    C705 68204000 281 MOV DWORD PTR DS:[402068],Point
00401031    C705 6C204000 000 MOV DWORD PTR DS:[40206C],0
0040103B    C705 70204000 000 MOV DWORD PTR DS:[402070],0
00401045    A1 CA204000      MOV EAX,DWORD PTR DS:[4020CA]
0040104A    A3 74204000      MOV DWORD PTR DS:[402074],EAX   kernel32.BaseThreadInitThunk
0040104F    6A 64            PUSH 64                         RsrcName = 100.
00401051    50               PUSH EAX                        hInst = 759B3388
00401052    E8 D1030000      CALL <JMP.&USER32.LoadIconA>    LoadIconA
00401057    A3 78204000      MOV DWORD PTR DS:[402078],EAX   kernel32.BaseThreadInitThunk
```

**** Make sure you are paused at 401000 and not at the raw entry to the file. If you are not paused at the real OEP (401000) try pressing F9 once- Olly should then pause at the real entry point. If that doesn’t work, just open the ‘Memory’ window (“Me” icon in the toolbar), highlight the Point-H Crackme ‘CODE’ section and hit enter. This will take you to the entry point of the actual binary. ****

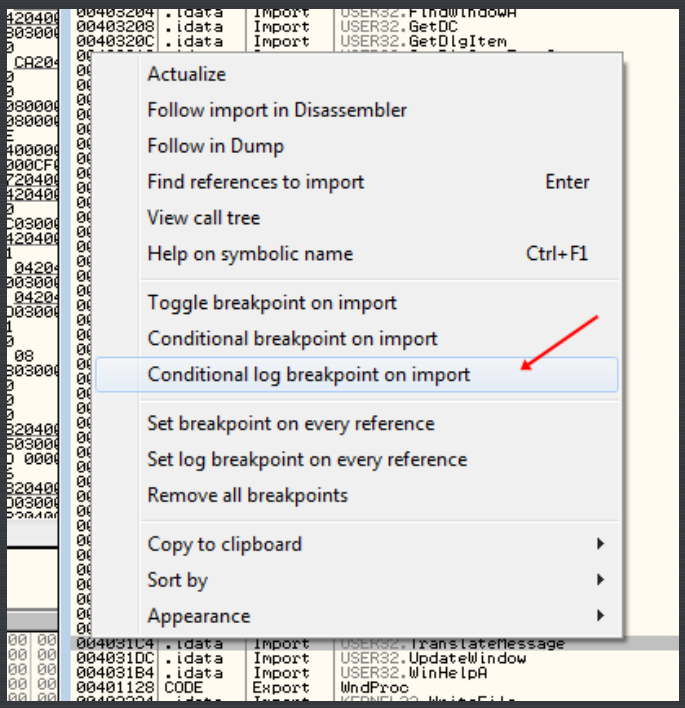
Now right click in the diassembly window and select “Search for” -> “Name (label) in current modile”, or hit Ctrl-N. This will bring up the names window:

Names in Point-H_

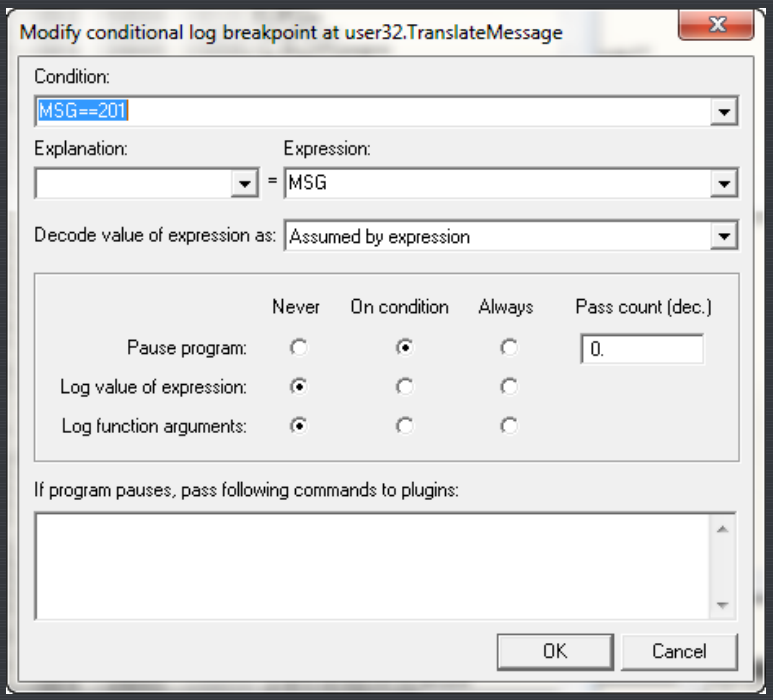
Address	Section	Type	Name
004031E4	.idata	Import	USER32.BeginPaint
00403230	.idata	Import	KERNEL32.CloseHandle
00403250	.idata	Import	COMCTL32.CreateToolbar
0040324C	.idata	Import	COMCTL32.CreateToolbarEx
004031E8	.idata	Import	USER32.CreateWindowExA
004031EC	.idata	Import	USER32.DefWindowProcA
00403278	.idata	Import	GDI32.DeleteDC
00403274	.idata	Import	GDI32.DeleteObject
004031F0	.idata	Import	USER32.DialogBoxParamA
004031F4	.idata	Import	USER32.DispatchMessageA
004031F8	.idata	Import	USER32.DrawMenuBar
004031FC	.idata	Import	USER32.EndDialog
00403270	.idata	Import	GDI32.EndDoc
0040326C	.idata	Import	GDI32.EndPage
00403200	.idata	Import	USER32.EndPaint
00403240	.idata	Import	KERNEL32.ExitProcess
00403204	.idata	Import	USER32.FindWindowA
00403208	.idata	Import	USER32.GetDC
0040320C	.idata	Import	USER32.GetDlgItem
00403210	.idata	Import	USER32.GetDlgItemTextA
0040321C	.idata	Import	KERNEL32.GetLocalTime
00403214	.idata	Import	USER32.GetMessageA
00403238	.idata	Import	KERNEL32.GetModuleHandleA
00403284	.idata	Import	COMDLG32.GetOpenFileNameA
00403280	.idata	Import	COMDLG32.GetSaveFileNameA
00403268	.idata	Import	GDI32.GetStockObject
00403188	.idata	Import	USER32.GetSystemMetrics
00403264	.idata	Import	GDI32.GetTextMetricsA
00403198	.idata	Import	USER32.GetWindowRect
00403228	.idata	Import	KERNEL32.GlobalAlloc
00403224	.idata	Import	KERNEL32.GlobalFree
00403248	.idata	Import	COMCTL32.InitCommonControls
004031B8	.idata	Import	USER32.InvalidateRect
00403184	.idata	Import	USER32.KillTimer
00403190	.idata	Import	USER32.LoadAcceleratorsA
004031A4	.idata	Import	USER32.LoadBitmapA
0040318C	.idata	Import	USER32.LoadCursorA
004031A0	.idata	Import	USER32.LoadIconA
004031C8	.idata	Import	USER32.LoadMenuA
0040319C	.idata	Import	USER32.LoadStringA
0040322C	.idata	Import	KERNEL32.lstrlen
00403194	.idata	Import	USER32.MessageBeep
004031AC	.idata	Import	USER32.MessageBoxA
00401000	CODE	Export	<ModuleEntryPoint>
004031C0	.idata	Import	USER32.MoveWindow
00403220	.idata	Import	KERNEL32.OpenFile
004031B0	.idata	Import	USER32.PostQuitMessage
00403288	.idata	Import	COMDLG32.PrintDlgA
0040323C	.idata	Import	KERNEL32.ReadFile
004031E0	.idata	Import	USER32.RegisterClassA
004031D0	.idata	Import	USER32.SendMessageA
004031A8	.idata	Import	USER32.SetFocus
004031D4	.idata	Import	USER32.SetTimer
004031D8	.idata	Import	USER32.SetWindowPos
004031CC	.idata	Import	USER32.ShowWindow
00403260	.idata	Import	GDI32.StartDocA
0040325C	.idata	Import	GDI32.StartPage
00403258	.idata	Import	GDI32.TextOutA
004031BC	.idata	Import	USER32.TranslateAcceleratorA
004031C4	.idata	Import	USER32.TranslateMessage
004031DC	.idata	Import	USER32.UpdateWindow
004031B4	.idata	Import	USER32.WinHelpA
00401128	CODE	Export	WndProc
00403234	.idata	Import	KERNEL32.WriteFile

Toward the bottom will be the TranslateMessage API. Right-click on this and select “Conditional log

breakpoint on import”:

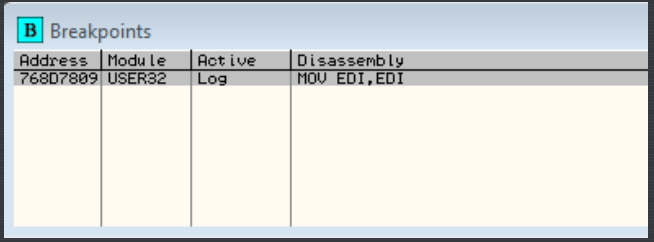


This will bring up the conditional breakpoint screen:



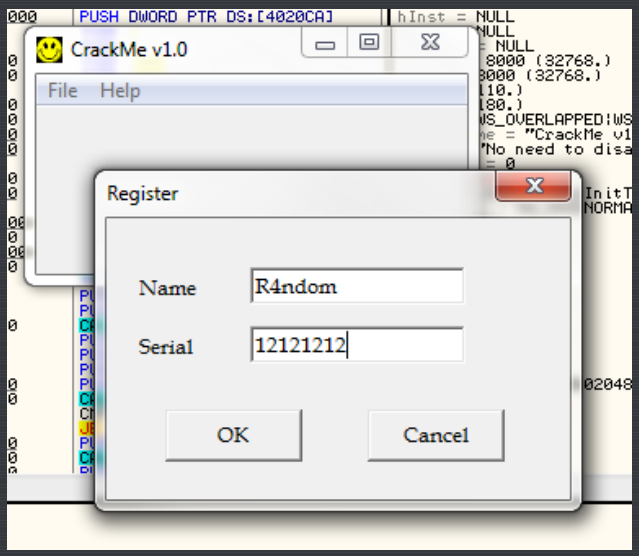
Set up the screen as shown. We will use this conditional breakpoint to weed out all of the calls to translate message until we hit the message with the ID of 201. Looking at our old cheat sheet for Windows message IDs (provided in tutorial 16A), we see that this ID corresponds to the left mouse button down message. This is to trap the TranslateMessage function when processing the click of the “OK” button in the crackme.

When you click OK in the conditional breakpoint window, you should see the breakpoint in the ‘Breakpoints’ window:

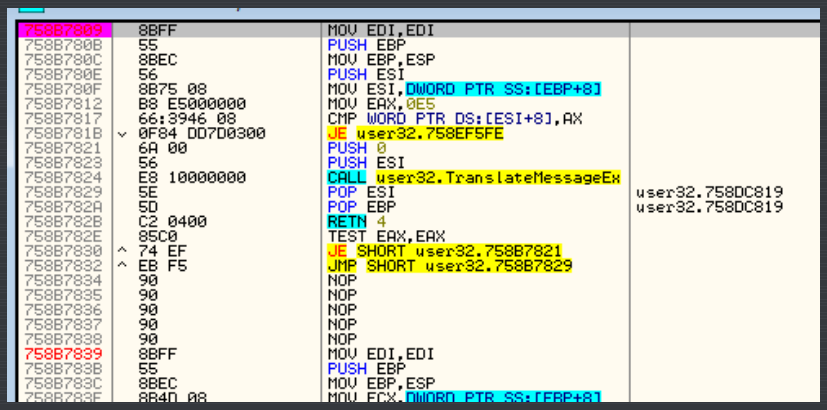


You may notice that it says “Log” under the Active column, letting us know that this is in fact a log BP.

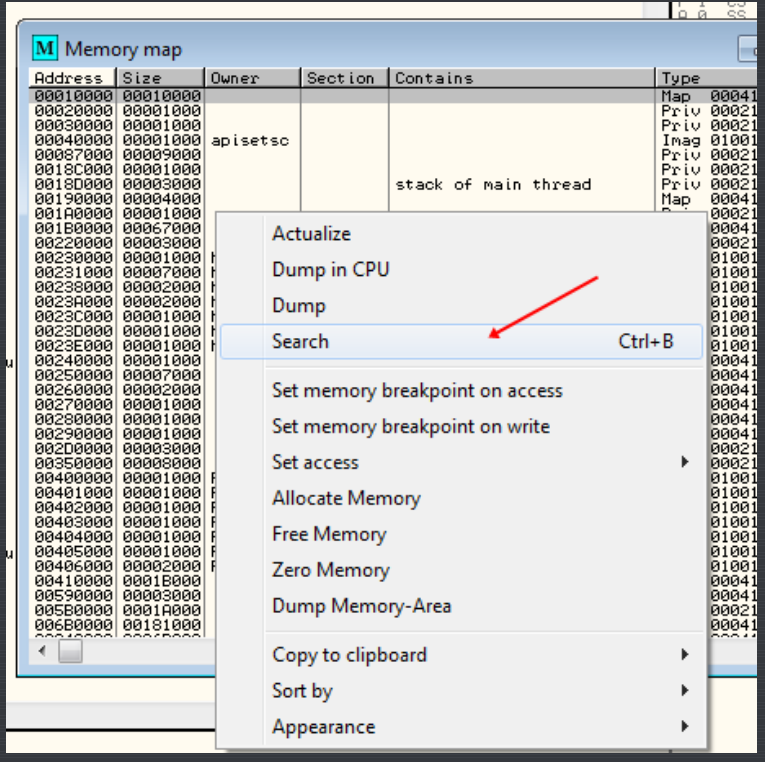
Go ahead and run the crackme. select “Edit” -> “Register”, and put in a name and serial. Make sure you use TAB to move among the fields or our left mouse click message will fire our breakpoint prematurely. I entered “R4ndom” for the name and “12121212″ for the serial:



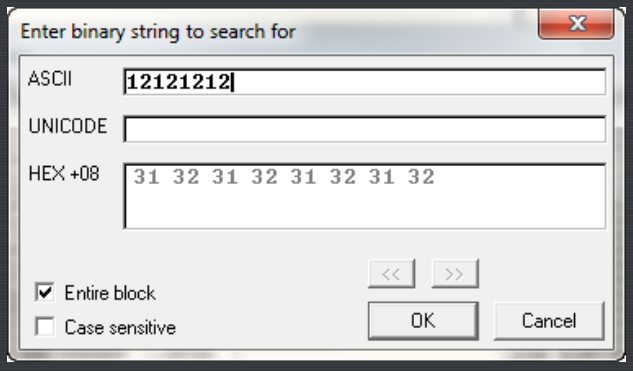
Now click OK and Olly should break at our conditional breakpoint:



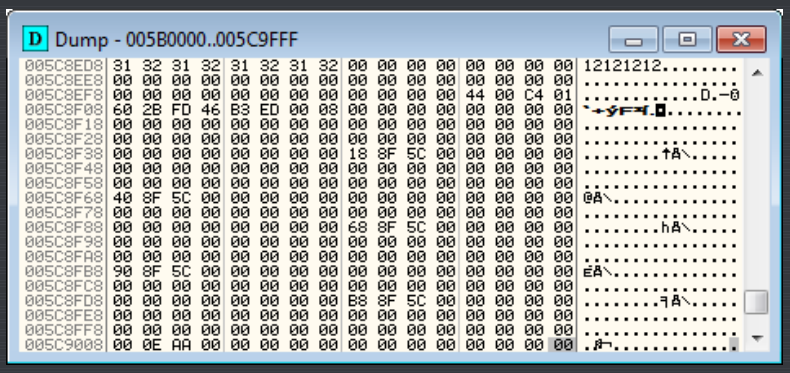
Now we want to search for our serial in memory. Open the Memory window by clicking the “Me” icon or typing Alt-M. Right-click in this window and choose “Search”:



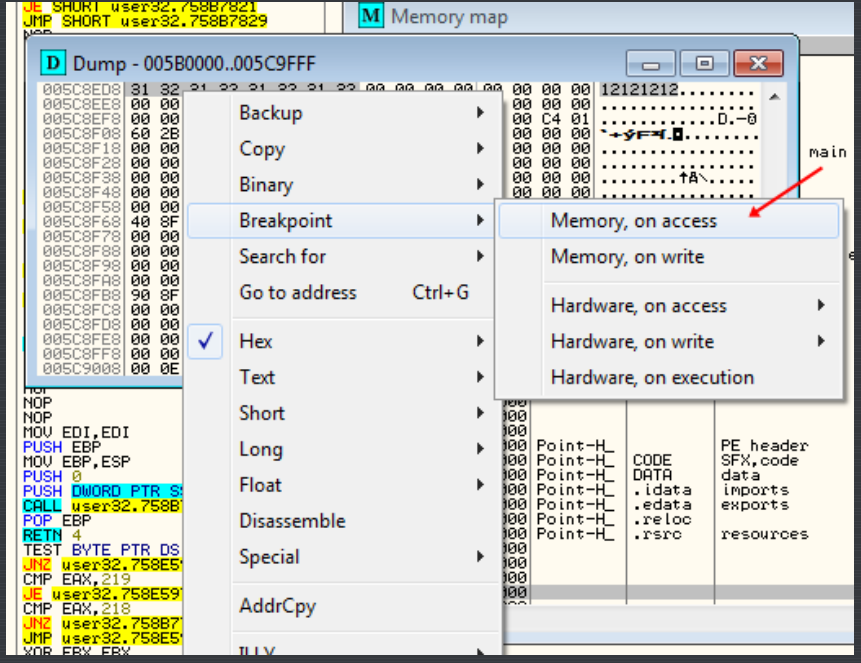
In the ASCII field, enter 12121212:



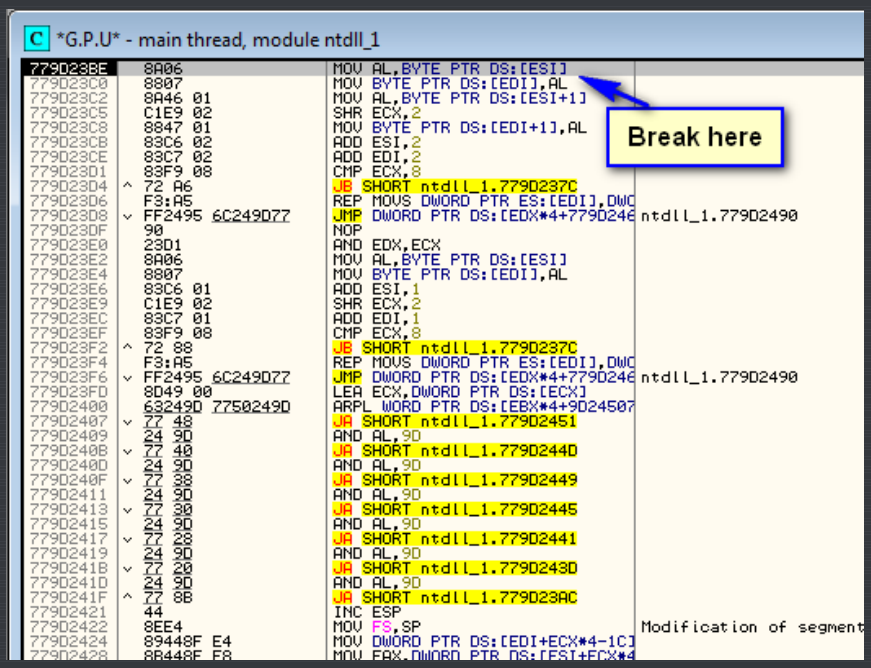
Olly should show us where in memory our serial resides:



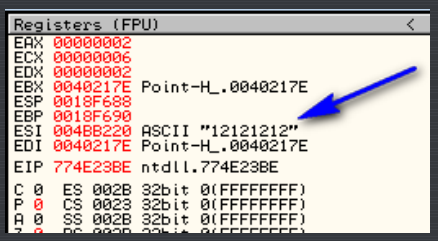
We want to tell Olly to pause when this memory address is accessed. Highlight the first byte of the serial and right-click on it. select “Breakpoint” -> “Memory, On access”:



Now hit F9 to run the target. Olly will then break on our memory breakpoint (you may break at our previous conditional BP first, in which case just hit F9 again):



This is the Point-H on your system. On mine, as we can see, it's 774E23BE. Write this down. If you look in the registers window, you will see our serial in ESI:



Let's now experiment a little with this magical breakpoint. Delete the log breakpoint as well as the memory BPs set earlier. Add a hardware BP on execute on the Point-H address (so it won't be lost on a re-start) and restart the target. We should then break before we see the main crackme window:

```
774E239C   8A46 02          MOV AL,BYTE PTR DS:[ESI+2]
774E239F   C1E9 02          SHR ECX,2
774E23A2   8847 02          MOV BYTE PTR DS:[EDI+2],AL
774E23A5   83C6 03          ADD ESI,3
774E23A8   83C7 03          ADD EDI,3
774E23AB   83F9 08          CMP ECX,8
774E23AE ^ 72 CC            JB SHORT ntdll.774E237C
774E23B0   F3:A5            REP MOVS DWORD PTR ES:[EDI],DWO
774E23B2 v FF2495 6C244E77  JMP DWORD PTR DS:[EDX*4+774E24E ntdll.774E2490
774E23B9   8D49 00          LEA ECX,DWORD PTR DS:[ECX]
774E23BC   23D1             AND EDX,ECX
774E23BE   8A06             MOV AL,BYTE PTR DS:[ESI]
774E23C0   8807             MOV BYTE PTR DS:[EDI],AL
774E23C2   8A46 01          MOV AL,BYTE PTR DS:[ESI+1]
774E23C5   C1E9 02          SHR ECX,2
774E23C8   8847 01          MOV BYTE PTR DS:[EDI+1],AL
774E23CB   83C6 02          ADD ESI,2
774E23CE   83C7 02          ADD EDI,2
774E23D1   83F9 08          CMP ECX,8
774E23D4 ^ 72 A6            JB SHORT ntdll.774E237C
774E23D6   F3:A5            REP MOVS DWORD PTR ES:[EDI],DWO
774E23D8 v FF2495 6C244E77  JMP DWORD PTR DS:[EDX*4+774E24E ntdll.774E2490
```

Point-H

And looking in the registers window, we see that the string we are copying is “OpenProcessToken”:

```
Registers (FPU)
EAX 00000002
ECX 0000000E
EDX 00000002
EBX 0018F3DC
ESP 0018F38C
EBP 0018F394
ESI 76679E7E ASCII "OpenProcessToken"
EDI 0018F3DE

EIP 774E23BE ntdll.774E23BE

C 0  ES 002B 32bit 0(FFFFFFFF)
P 0  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
```

Hitting F9 repeatedly, you will see various strings in the ESI register, each time ntdll32 calls this internal API. Besides API names, you will see various number sequences flash by as well as other things- basically anything the ntdll32 copies as a string.

Using ‘Point-H’ to Crack the Target

Now, you may be asking yourself, “Great. I did all this work. What’s the point?” (pun intended 😀) Let’s try it out. Temporarily disable the BP on the Point-H address and re-start the target. Select “Register” again from the menu and enter a name and serial. Before hitting OK, set the BP on Point-H again, then click OK in the target.

**** If you need to find the address of Point-H to set the BP, open the memory window, click once on ntdll’s .text section (toward the bottom) and click Enter. This will bring up ntdll in the disassembly window. Now you can goto (Ctrl-G) the address of Point-H. ****

Olly breaks at our breakpoint, and looking in the registers window, we can see that this time through is with our username:

```
Registers (FPU)
EAX 00000002
ECX 00000004
EDX 00000000
EBX 0040218E Point-H_.0040218E
ESP 0018F688
EBP 0018F690
ESI 0054B1E8 ASCII "R4ndom"
EDI 0040218E Point-H_.0040218E

EIP 774E23BE ntdll.774E23BE

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
```

Hitting F9 again, we see that we are now at the copying of the serial:

```
Registers (FPU)
EAX 00000002
ECX 00000006
EDX 00000002
EBX 0040217E Point-H_.0040217E
ESP 0018F688
EBP 0018F690
ESI 0054B220 ASCII "12121212"
EDI 0040217E Point-H_.0040217E

EIP 774E23BE ntdll.774E23BE

C 0  ES 002B 32bit 0(FFFFFFFF)
P 0  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
```

Now for the magic. Open the memory window, right-click on our target’s CODE section and choose “Set break on access”. This way we will break as soon as ntdll32 returns to our target’s code. Hitting F9, Olly pauses in our code section:

```
004012C1  .   FF75 08          PUSH [ARG.1]                    hWnd = 002406AC ('Register',class='#32770')
004012C4  .   E8 07020000      CALL <JMP.&USER32.GetDlgItemTe  GetDlgItemTextA
004012C9  .   83F8 01          CMP EAX,1
004012CC  .   C745 10 EB030000 MOV [ARG.3],3EB
004012D3  .^  72 CC            JB SHORT Point-H_.004012A1
004012D5  .   6A 0B            PUSH 0B                         Count = B (11.)
004012D7  .   68 7E214000      PUSH Point-H_.0040217E          Buffer = Point-H_.0040217E
004012DC  .   68 E9030000      PUSH 3E9                        ControlID = 3E9 (1001.)
004012E1  .   FF75 08          PUSH [ARG.1]                    hWnd = 002406AC ('Register',class='#32770')
004012E4  .   E8 E7010000      CALL <JMP.&USER32.GetDlgItemTex GetDlgItemTextA
004012E9  .   B8 01000000      MOV EAX,1
004012EE  .v  EB 07            JMP SHORT Point-H_.004012F7
004012F0  >   B8 00000000      MOV EAX,0
004012F5  .^  EB 8D            JMP SHORT Point-H_.00401284
004012F7  >   50               PUSH EAX
004012F8  .   FF75 08          PUSH [ARG.1]                    ster',class='#32770')
004012FB  .   E8 B2010000      CALL <JMP.&USER32.EndDialog>
00401300  .   B8 01000000      MOV EAX,1
00401305  .^  E9 7AFFFFFF      JMP Point-H_.00401284
0040130A  .   C8 000000        ENTER 0,0
0040130E  .   53               PUSH EBX
0040130F  .   56               PUSH ESI                        Point-H_.00401253
00401310  .   57               PUSH EDI
00401311  .   817D 0C 11010000 CMP [ARG.2],111
00401318  .v  74 12            JE SHORT Point-H_.0040132C
0040131A  .   837D 0C 10       CMP [ARG.2],10
0040131E  .v  74 15            JE SHORT Point-H_.00401335
```

The call to kernel32

We break here

As you can see, we paused right after a call to GetDlgItemTextA. Deep inside ntdll32, this function eventually called our code containing the Point-H address. It then returned to our target's code at address 4012E9. We now have a starting point to try and crack this crackme. The crackme has just gotten the entered serial number, and will soon do something with it. Of course, because this crackme is not very hard, it's not really that impressive, but in a commercial app, with encryption and protection routines, being able to zero in on this code is a life-saver.

In the case of this specific crackme, continuing to step through the code, and placing access breakpoints on the code section of the crackme, it doesn't take long before we get to the relevant code for patching (I leave this up to you…).

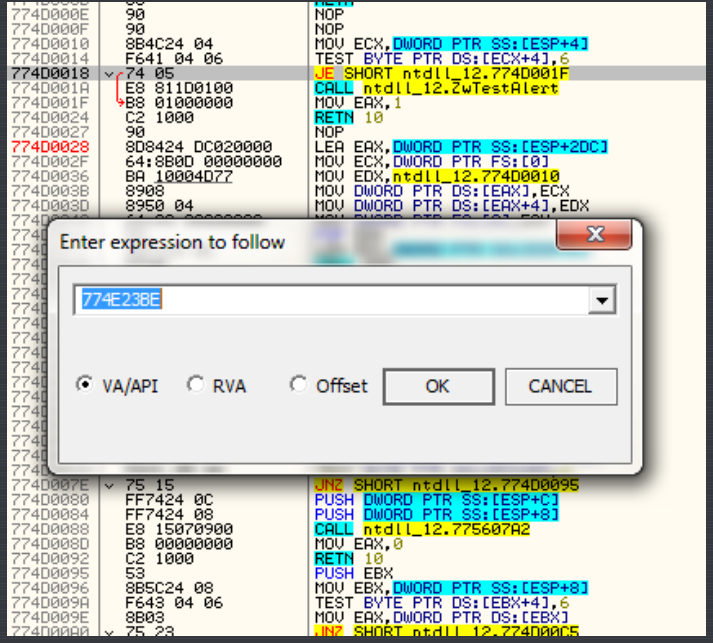
Another Target Using Point-H

Let's try another crackme using this technique. Load "CrackmeVB4.exe" into Olly. First, let's find our Point-H address; open the Memory window, scroll to the bottom, and highlight whatever section your point-h address is in. For me, that would be address 774E23BE in ntdll.dll's .text section:

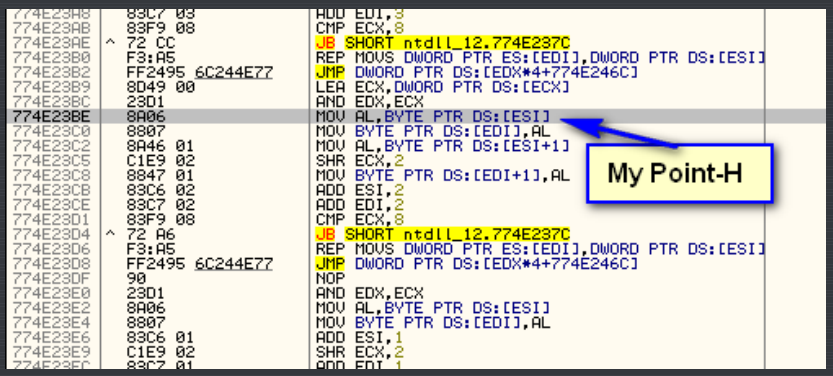
Memory map

Address	Size	Owner	Section	Contains	Type	Acce
76950000	0005B000	user32	.rsrc	resources	Imag 01001002	R
769B0000	00004000	user32	.reloc		Imag 01001002	R
76AE0000	00001000	oleaut32		PE header	Imag 01001002	R
76AE1000	00084000	oleaut32	.text	SFX,code,imports,exports	Imag 01001002	R
76B65000	00001000	oleaut32	.orpc		Imag 01001002	R
76B66000	00002000	oleaut32	.data		Imag 01001002	R
76B68000	00001000	oleaut32	.rsrc	resources	Imag 01001002	R
76B69000	00006000	oleaut32	.reloc		Imag 01001002	R
772E0000	00010000	ntdll		PE header	Imag 01001002	R
772F0000	000D6000	ntdll	.text	code,exports	Imag 01001002	R
773C6000	0000A000	ntdll			Imag 01001002	R
773D0000	00001000	ntdll	RT	data	Imag 01001002	R
773D1000	0000F000	ntdll			Imag 01001002	R
773E0000	00009000	ntdll	.data		Imag 01001002	R
773E9000	00007000	ntdll			Imag 01001002	R
773F0000	00057000	ntdll	.rsrc	resources	Imag 01001002	R
77447000	00009000	ntdll			Imag 01001002	R
77450000	00005000	ntdll	.reloc		Imag 01001002	R
77455000	00034000	ntdll			Imag 01001002	R
77490000	00001000	lpk		PE header	Imag 01001002	R
77491000	00006000	lpk	.text	SFX,code,imports,exports	Imag 01001002	R
77497000	00001000	lpk	.data	data	Imag 01001002	R
77498000	00001000	lpk	.rsrc	resources	Imag 01001002	R
77499000	00001000	lpk	.reloc		Imag 01001002	R
774C0000	00001000	ntdll_12		PE header	Imag 01001040	RWE
774D0000	000D6000	ntdll_12	.text	code,exports	Imag 01001020	R E
775B0000	00001000	ntdll_12	RT	data	Imag 01001020	R E
775C0000	00009000	ntdll_12	.data		Imag 01001004	RW
775D0000	00057000	ntdll_12	.rsrc	resources	Imag 01001002	R
77630000	00005000	ntdll_12	.reloc		Imag 01001002	R
7EFB0000	00023000				Map 00041002	R
7EFDB000	00002000				Priv 00021004	RW
7EFDD000	00001000			data block of main thread	Priv 00021004	RW
7EFDE000	00001000				Priv 00021004	RW
7EFDF000	00001000				Priv 00021004	RW
7EFE0000	00005000				Map 00041002	R
7FFE0000	00001000				Priv 00021002	R

Click Enter while this line is highlighted, which opens this module in the disassembler. Now, hit Ctrl-G and enter the address of the point-h on your system to make Olly jump to that location:



and we land at point-h:



Now, keeping this location available in the disassembly window (but without placing a breakpoint yet), and run the target. First, a nag comes up (for what seems like 3 hours), and then we see the main entry screen:

Enter a name and serial, as I've done here, and then set your breakpoint (F2) on the point-h in Olly, before clicking OK in the target. After setting the breakpoint, click OK in the target and Olly will break at our point-h:

```
*G.P.U* - main thread, module ntdll_12
774E2396   8A46 01          MOV AL,BYTE PTR DS:[ESI+1]
774E2399   8847 01          MOV BYTE PTR DS:[EDI+1],AL
774E239C   8A46 02          MOV AL,BYTE PTR DS:[ESI+2]
774E239F   C1E9 02          SHR ECX,2
774E23A2   8847 02          MOV BYTE PTR DS:[EDI+2],AL
774E23A5   83C6 03          ADD ESI,3
774E23A8   83C7 03          ADD EDI,3
774E23AB   83F9 08          CMP ECX,8
774E23AE ^ 72 CC            JB SHORT ntdll_12.774E237C
774E23B0   F3:A5            REP MOVS DWORD PTR ES:[EDI],DWORD
774E23B2 v FF2495 6C244E77  JMP DWORD PTR DS:[EDX*4+774E246C]
774E23B9   8D49 00          LEA ECX,DWORD PTR DS:[ECX]
774E23BC   23D1             AND EDX,ECX
774E23BE   8A06             MOV AL,BYTE PTR DS:[ESI]
774E23C0   8807             MOV BYTE PTR DS:[EDI],AL
774E23C2   8A46 01          MOV AL,BYTE PTR DS:[ESI+1]
774E23C5   C1E9 02          SHR ECX,2
774E23C8   8847 01          MOV BYTE PTR DS:[EDI+1],AL
774E23CB   83C6 02          ADD ESI,2
774E23CE   83C7 02          ADD EDI,2
774E23D1   83F9 08          CMP ECX,8
774E23D4 ^ 72 A6            JB SHORT ntdll_12.774E237C
774E23D6   F3:A5            REP MOVS DWORD PTR ES:[EDI],DWORD
774E23D8 v FF2495 6C244E77  JMP DWORD PTR DS:[EDX*4+774E246C]
774E23DF   90               NOP
774E23E0   23D1             AND EDX,ECX
774E23E2   8A06             MOV AL,BYTE PTR DS:[ESI]
774E23E4   8807             MOV BYTE PTR DS:[EDI],AL
774E23E6   83C6 01          ADD ESI,1
774E23E9   C1E9 02          SHR ECX,2
774E23EC   83C7 01          ADD EDI,1
774E23EF   83F9 08          CMP ECX,8
774E23F2 ^ 72 88            JB SHORT ntdll_12.774E237C

Registers (FPU)
EAX 00000002
ECX 00000013
EDX 00000003
EBX 0018F414
ESP 0018F3C4
EBP 0018F3CC
ESI 71BB54AC  ASCII "DwmSetWindowAttribute"
EDI 0018F416
EIP 774E23BE ntdll_12.774E23BE

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 16.0000000000000000000
ST7 empty 16.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0
FCW 137F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

As we can see in the registers window, this isn't the stop we want to investigate, so click F9 again. You will need to press F9 a couple more times until a string comes up that we are interested in. In this case, the "You Get Wrong…" string looks pretty promising:

```
*G.P.U* - main thread, module ntdll_12
774E2396   8A46 01          MOV AL,BYTE PTR DS:[ESI+1]
774E2399   8847 01          MOV BYTE PTR DS:[EDI+1],AL
774E239C   8A46 02          MOV AL,BYTE PTR DS:[ESI+2]
774E239F   C1E9 02          SHR ECX,2
774E23A2   8847 02          MOV BYTE PTR DS:[EDI+2],AL
774E23A5   83C6 03          ADD ESI,3
774E23A8   83C7 03          ADD EDI,3
774E23AB   83F9 08          CMP ECX,8
774E23AE ^ 72 CC            JB SHORT ntdll_12.774E237C
774E23B0   F3:A5            REP MOVS DWORD PTR ES:[EDI],DWORD P
774E23B2 v FF2495 6C244E77  JMP DWORD PTR DS:[EDX*4+774E246C]
774E23B9   8D49 00          LEA ECX,DWORD PTR DS:[ECX]
774E23BC   23D1             AND EDX,ECX
774E23BE   8A06             MOV AL,BYTE PTR DS:[ESI]
774E23C0   8807             MOV BYTE PTR DS:[EDI],AL
774E23C2   8A46 01          MOV AL,BYTE PTR DS:[ESI+1]
774E23C5   C1E9 02          SHR ECX,2
774E23C8   8847 01          MOV BYTE PTR DS:[EDI+1],AL
774E23CB   83C6 02          ADD ESI,2
774E23CE   83C7 02          ADD EDI,2
774E23D1   83F9 08          CMP ECX,8
774E23D4 ^ 72 A6            JB SHORT ntdll_12.774E237C

Registers (FPU)
EAX 00000002
ECX 0000002E
EDX 00000002
EBX 00000000
ESP 0018EE84
EBP 0018EE8C
ESI 005FBB78  UNICODE "You Get Wrong\r\nTry Again"
EDI 005FBC5E
EIP 774E23BE ntdll_12.774E23BE

C 0  ES 002B 32bit 0(FFFFFFFF)
P 0  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)
```

Now what we want to do is trap execution as soon as we get back to our target's code. Because of the nature of Visual Basic, we can't just set an access memory breakpoint on the code section of our target (as we would in a native application). So what we want to do is single step (over) until we get back to our target's code. We will first go into user32.dll (setting the cursor etc) and back through the VB runtime. The badboy will be displayed, after which we will step quite a ways further, but eventually, we will land here:

```
We first land here
                JE SHORT CrackmeV.004086DB
                MOV ESI,DWORD PTR DS:[<&MSVBVM50.__vbaStrCat>]   Msvbvm50.__vbaStrCat
                PUSH CrackmeV.00406FC0                           UNICODE "You Get It"
                PUSH CrackmeV.00406FDC                           UNICODE "\r\n"
00408689  .  FFD6    CALL ESI                                    Msvbvm50.__vbaStrCat; <&MSVBVM50.__vbaStrCat>
0040868B  .  8BD0    MOV EDX,EAX
0040868D  .  8D4D E8 LEA ECX,DWORD PTR SS:[EBP-18]
00408690  .  FF15 94B14000 CALL DWORD PTR DS:[<&MSVBVM50.__vbaStrMove>]   Msvbvm50.__vbaStrMove
00408696  .  50      PUSH EAX
00408697  .  68 E86F4000 PUSH CrackmeV.00406FE8                  UNICODE "KeyGen It Now"
0040869C  .  FFD6    CALL ESI                                    Msvbvm50.__vbaStrCat
0040869E  .  8945 CC MOV DWORD PTR SS:[EBP-34],EAX
004086A1  .  8D45 94 LEA EAX,DWORD PTR SS:[EBP-6C]
004086A4  .  8D4D A4 LEA ECX,DWORD PTR SS:[EBP-5C]
004086A7  .  50      PUSH EAX
004086A8  .  8D55 B4 LEA EDX,DWORD PTR SS:[EBP-4C]
004086AB  .  51      PUSH ECX                                    ntdll_12.774F389A
004086AC  .  52      PUSH EDX
004086AD  .  8D45 C4 LEA EAX,DWORD PTR SS:[EBP-3C]
004086B0  .  6A 00   PUSH 0
004086B2  .  50      PUSH EAX
004086B3  .  C745 C4 08000( MOV DWORD PTR SS:[EBP-3C],8
004086BA  .  FF15 24B14000 CALL DWORD PTR DS:[<&MSVBVM50.#595>]   Msvbvm50.rtcMsgBox
004086C0  .  8D4D E8 LEA ECX,DWORD PTR SS:[EBP-18]
004086C3  .  FF15 A8B14000 CALL DWORD PTR DS:[<&MSVBVM50.__vbaFreeStr>]   Msvbvm50.__vbaFreeStr
004086C9  .  8D4D 94 LEA ECX,DWORD PTR SS:[EBP-6C]
004086CC  .  8D55 A4 LEA EDX,DWORD PTR SS:[EBP-5C]
004086CF  .  51      PUSH ECX                                    ntdll_12.774F389A
004086D0  .  8D45 B4 LEA EAX,DWORD PTR SS:[EBP-4C]
004086D3  .  52      PUSH EDX
004086D4  .  8D4D C4 LEA ECX,DWORD PTR SS:[EBP-3C]
004086D7  .  50      PUSH EAX                                                 Badboy
004086D8  .  51      PUSH ECX                                    ntdll_12.774F389A
004086D9  .v EB 60   JMP SHORT CrackmeV.0040873B
004086DB  >  8B35 14B14000 MOV ESI,DWORD PTR DS:[<&MSVBVM50.__vbaStrCat>]   Msvbvm50.__vbaStrCat
004086E1  .  68 08704000 PUSH CrackmeV.00407008                  UNICODE "You Get Wrong"
004086E6  .  68 DC6F4000 PUSH CrackmeV.00406FDC                  UNICODE "\r\n"
004086EB  .  FFD6    CALL ESI                                    Msvbvm50.__vbaStrCat; <&MSVBVM50.__vbaStrCat>
004086ED  .  8BD0    MOV EDX,EAX
004086EF  .  8D4D E8 LEA ECX,DWORD PTR SS:[EBP-18]
004086F2  .  FF15 94B14000 CALL DWORD PTR DS:[<&MSVBVM50.__vbaStrMove>]   Msvbvm50.__vbaStrMove
004086F8  .  50      PUSH EAX
004086F9  .  68 28704000 PUSH CrackmeV.00407028                  UNICODE "Try Again"
004086FE  .  FFD6    CALL ESI                                    Msvbvm50.__vbaStrCat
00408700  .  8945 CC MOV DWORD PTR SS:[EBP-34],EAX
00408703  .  8D55 94 LEA EDX,DWORD PTR SS:[EBP-6C]
00408706  .  8D45 A4 LEA EAX,DWORD PTR SS:[EBP-5C]
00408709  .  52      PUSH EDX
0040870A  .  8D4D B4 LEA ECX,DWORD PTR SS:[EBP-4C]
0040870D  .  50      PUSH EAX
0040870E  .  51      PUSH ECX                                    ntdll_12.774F389A
0040870F  .  8D55 C4 LEA EDX,DWORD PTR SS:[EBP-3C]
00408712  .  6A 00   PUSH 0                                                   Call to badboy
00408714  .  52      PUSH EDX                                                 message box
00408715  .  C745 C4 08000( MOV DWORD PTR SS:[EBP-3C],8
0040871C  .  FF15 24B14000 CALL DWORD PTR DS:[<&MSVBVM50.#595>]   Msvbvm50.rtcMsgBox
00408722  .  8D4D E8 LEA ECX,DWORD PTR SS:[EBP-18]
00408725  .  FF15 A8B14000 CALL DWORD PTR DS:[<&MSVBVM50.__vbaFreeStr>]   Msvbvm50.__vbaFreeStr
0040872B  .  8D45 94 LEA EAX,DWORD PTR SS:[EBP-6C]
0040872E  .  8D4D A4 LEA ECX,DWORD PTR SS:[EBP-5C]
00408731  .  50      PUSH EAX
00408732  .  8D55 B4 LEA EDX,DWORD PTR SS:[EBP-4C]
00408735  .  51      PUSH ECX                                    ntdll_12.774F389A
```

Goodboy

Here we can clearly see that we are returning after a call to rtcMsgBox which displayed our badboy message window. Above this we see our badboy being created, and more importantly, above that we see our goodboy. Now, finding the patch (at address 408677) is extremely easy. Of course, in this specific crackme, using P32Dasm or VB Decompiler would have been far simpler, but when we run into a devilish VB target where these simple tricks won't work, the Point-H method can be invaluable.

What About Smartcheck?

This may come as quite a shock to you, but I am not going over Smartcheck. This set of tutorials is meant as a *current* guide to reverse engineering. Because Numega's Smartcheck is not compatible with operating systems beyond Windows Vista (though it is possible with a lot of work in compatibility modes, running in a virtual machine, installing old VB runtimes etc) I believe it shouldn't be a part of a current reverser's toolkit. I have shown how, using tools that work with all versions of Windows like VB Decompiler and P32Dasm, any target can be examined and cracked using just these tools. By all means, if you are using Windows XP and can get Smartcheck to work, it is a valuable tool and should be used. But until someone takes up the charge and makes Smartcheck compatible with newer operating systems, I feel it's just too old to be relevant.

-Till next time

R4ndom