Simple registration checks in unpacked applications

potassium of ARTeam

Version 1.0 – 13th December 2005

1. Abstract 2
2. Where to begin, what and how to find, what to do 3
3. References 18
4. Conclusions 18
5. Greetings 18

Keywords

Reverse code engineering (RCE), cracking, registration checks.

1. Abstract

When constructing commercial software with the intent to make money from it, it becomes crucial to incorporate some kind of protection scheme into the product. Thus, controlling who is able to run the application and who is not. Since programmers are just as human as you and me, they often make it as simple as possible for themselves. It is very common that they use one specially designated pointer that holds the information regarding registration, eg registered = 1 (you have purchased the application) or register = 0 (you have not..). Debugging the code of a certain application and look how it flows (interactive) depending on the value of the designated pointer and conditional jumps often gives you information of what value to put into the designated pointer. In some cases, disassembly of the target exe into a static text file (deadlist) will do the trick.

The purpose with this document is simply to give some very important information to the crowd of newbies on the scene. Since most of the software available on the internet today is packed or/and encrypted, unpacking becomes a more and more important skill. However, if you lack the knowledge of the basic reversing you are guaranteed to fail in your quest.

Note:

This document is ONLY for educational purposes, if you intend to use your newly acquired knowledge for criminal acts that is **your** decision. ViceVersa Pro Build 2.0.0.9 herein called “the target” is simply an example and the theory of this document works on many other targets. Do not use this tutorial to crack this application if you intend to use it. As always, if you like an application and find it useful, you should **buy** it!

2. Where to begin, what and how to find, what to do

First of all, to get started, you need some basic knowledge of the assembler language and some terminology. This you can (if not acquired earlier) get by reading previous materials written by various authors. *The Lazy Beginning Reverser's Guide to Windows Assembly* by Vortex168 (2002)[1] is highly recommended since it is short, easily understood and very informative. Due to lack of time I will not provide a terminology list, the search engines on the net can provide you with that information.

Then, as any craftsman of any kind, you will need TOOLS. The more you got, the better. Recommended must-haves are: Ollydbg & Imprec with a bunch of plug-ins and of course, Lord PE, PEiD and Soft-ICE. Classics like these also comes in handy from time to time: W32Dasm, APIS32 and Hiew.

Some download links :

http://www.ollydbg.de/odbg110.zip
http://www.exetools.com/ - Here you can find some candy
http://www.tgrmn.com/web/downloads/vvpro.exe - ViceVersa Pro

Now you have to acquire a target to practice on, that preferable isn't packed. You can verify this (at least in most cases) using the tool PEiD. Take note of the contents in figure 2.1, PEid reports that this application is written in M$ Visual C++ 7.0 and in this particular case, not packed.

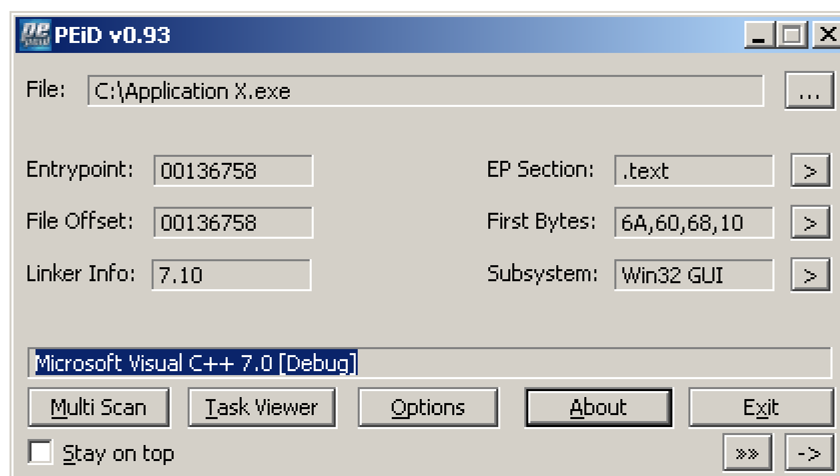

Figure 2.1 Checking with PEiD

Next, we start the target application and look for certain clues that might be useful to us.

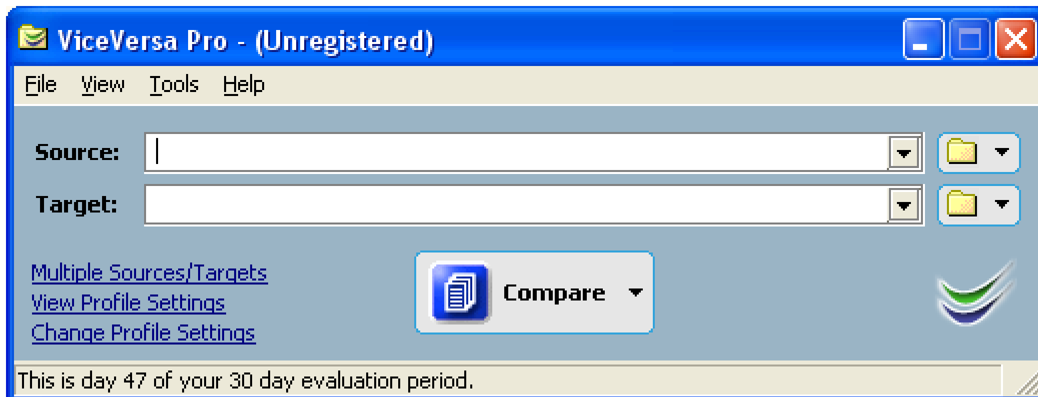

Figure 2.2 Mainwindow of the target application

Okay, here we have a nice lead, the string "- (Unregistered)". It sound reasonable that the string could alternatively be "- (Registered)", which is what we want to accomplish, right?

Close the target. From here we can take two alternative routes:

1. Disassemble the exe into a deadlist
2. Use an interactive debugger like Ollydbg (or others like SoftIce, IDA etc.)

Since this tutorial is at the newbie level I will present you both methods. There are some tutorials about the usage of Olly available on http://tutorials.accessroot.com. I recommend you to read them first and/or consult the help file included with Olly.

Method 1 – Deadlist disassembly using W32Dasm 8.93

First, make a copy of the target exe and open the copy in W32Dasm.

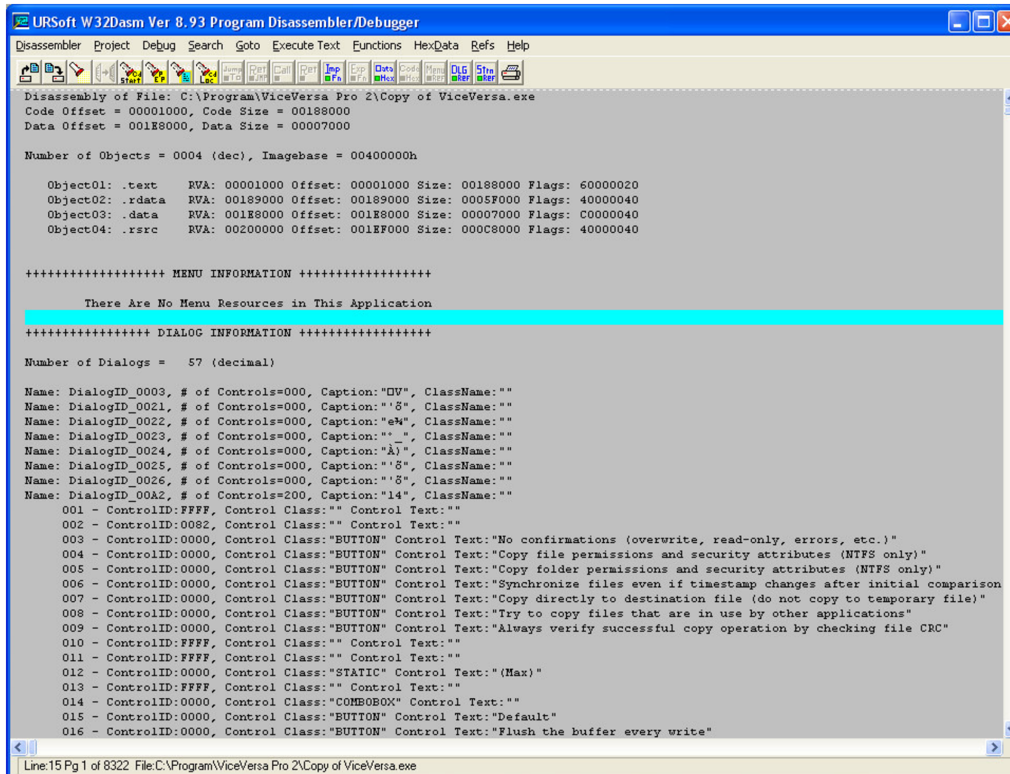

Figure 2.3 W32Dasm

Press the search button and enter the text "Unregistered" and search for it.

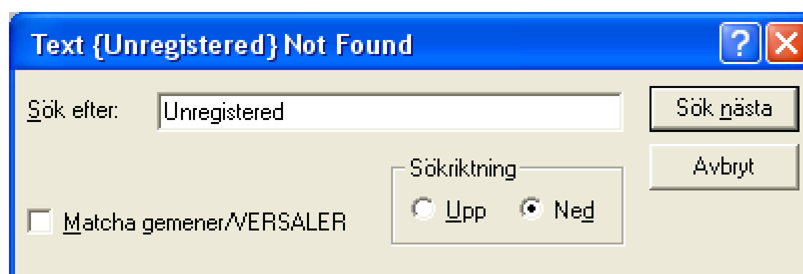

Figure 2.4 Searching for strings with W32Dasm.

Hmmm.. Nothing found?! That's odd. Or is it?

No, not really. Strings are usually stored as ASCII or Unicode. And W32Dasm do not always reference to strings. The string "Hey Man!" would look like this in ASCII: **48 65 79 20 4D 61 6E 21**. But if it was Unicode things it would look like this: **00 48 00 65 00 79 00 20 00 4D 00 61 00 6E 00 21**.

This difference is crucial when searching for strings, in this case the string is stored as ASCII.

Any hexeditor will do the job here. I will use Hiew.

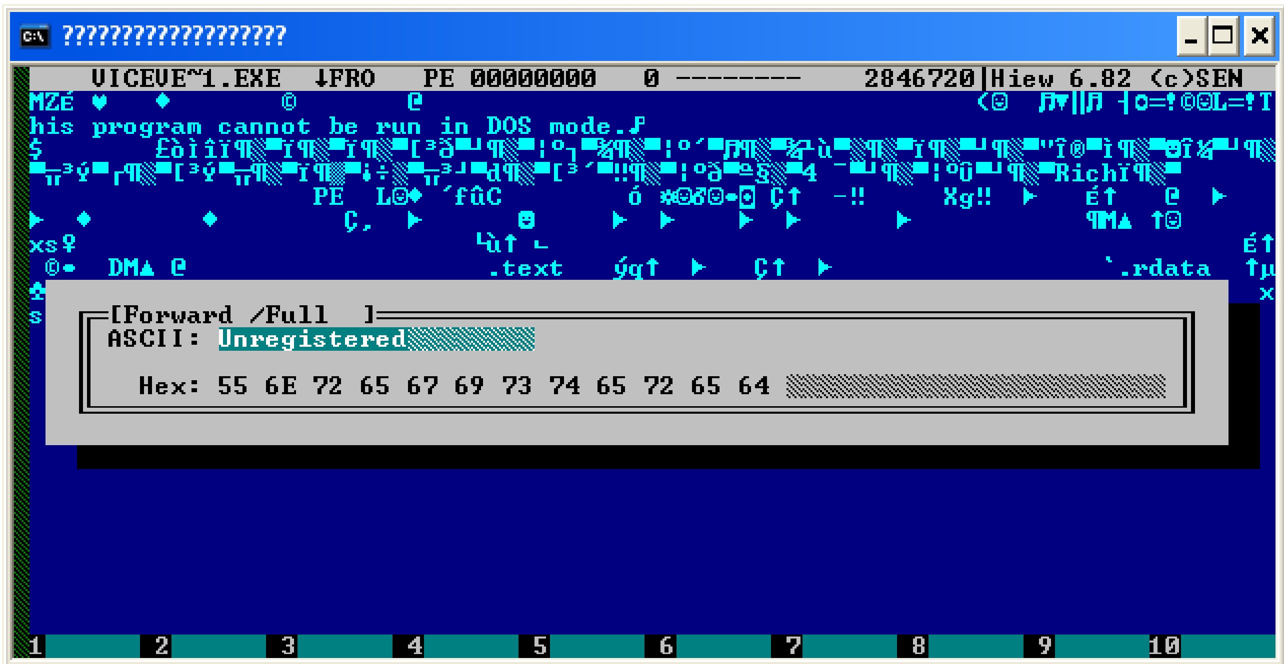

Figure 2.5 Searching for strings with Hiew.

Open the target exe in Hiew and search for the text "Unregistered" and you will end up here.

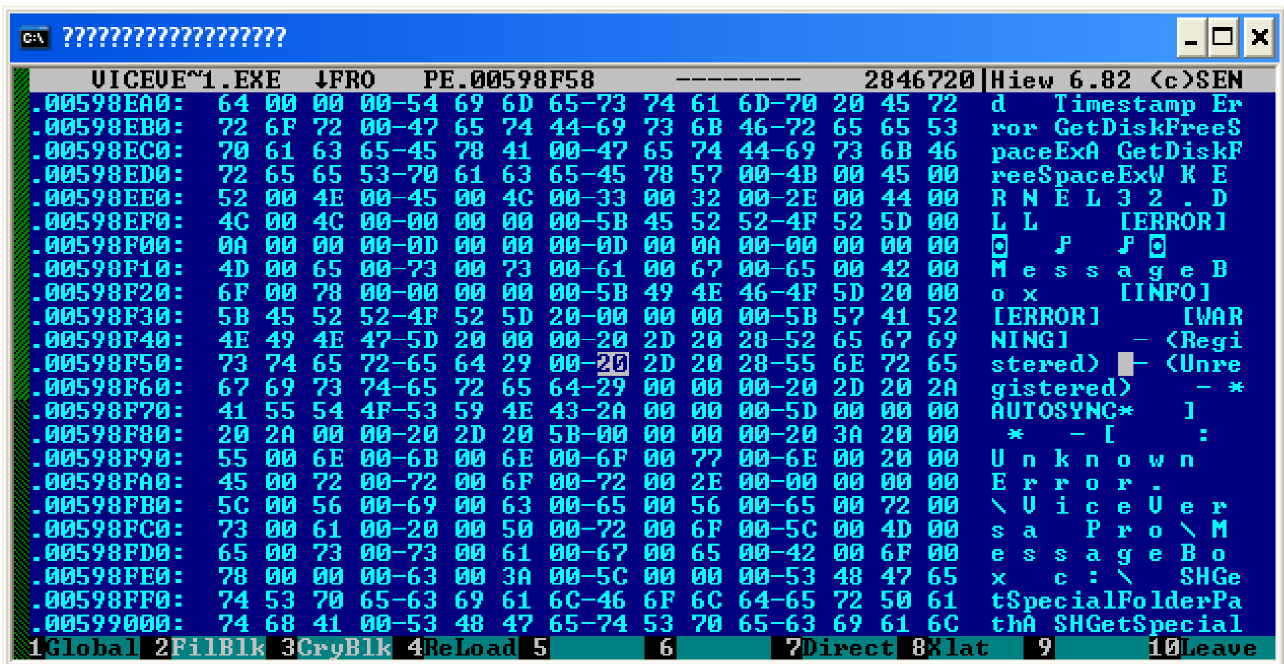

Figure 2.6 Found it!

Do you recognize the string from the application mainwindow? (fig 2.2) Well, there it is, starting at offset 00598f58. This is what we will search for in W32Dasm.

So, switch back to W32Dasm. Search for the text “00598f58” and will land here :

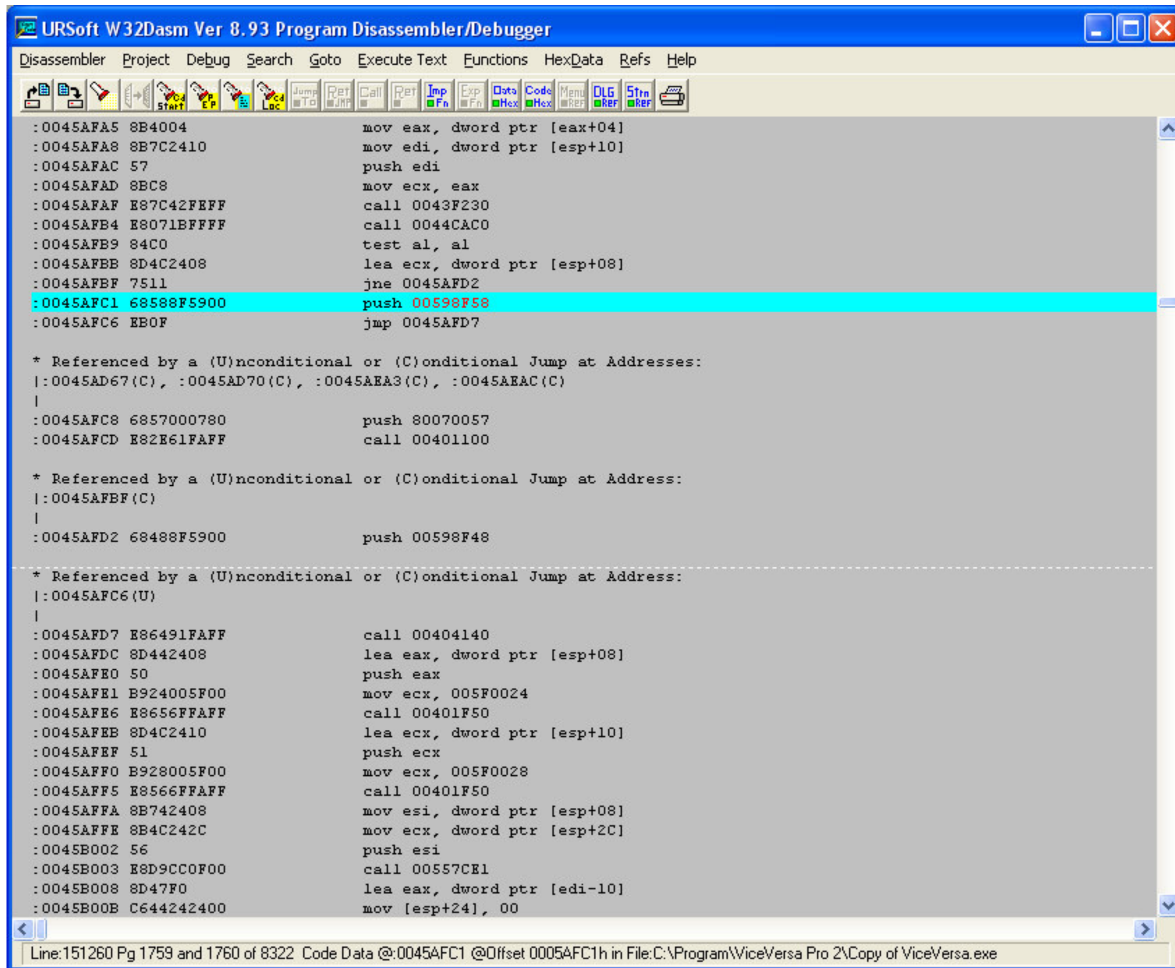

Figure 2.7 Searching for text pointer”00598f58”

This is also the one and ONLY occurrence of the instruction *push 00598f58* and therefore this place is the place of interest. See the rows a little bit up ? A call to 0044CAC0, test al, al and then jne 0045AFD2.

This code tells me that (probably) the value maintained in al (the low byte of eax) controls whether the text “- (Unregistered)” or “- (Registered)” should be displayed in the window caption. Check your Hiew 16 ($10) bytes before the string “- (Unregistered)” and you will see the string “- (Registered)”. The OP code 75 does not mean ‘jump if not equal’ as W32Dasm says, in fact it means ‘jump if not zero’[2]. So, if al is non zero the cpu jumps to 0045AFD2! Lets assume that the preceding call to 0044CAC0 stores or returns something interesting in the al register.

Mark the row at 0045AFB4 and press the “Call” button located in the toolbar.

Then you will end up here:

```
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger
Disassembler  Project  Debug  Search  Goto  Execute Text  Functions  HexData  Refs  Help

:0044CABC CC                      int 03
:0044CABD CC                      int 03
:0044CABE CC                      int 03
:0044CABF CC                      int 03

* Referenced by a CALL at Addresses:
|:0044051F   , :004405B3   , :004407D9   , :0044552F   , :004476A1
|:0044F686   , :0045094C   , :00450A8E   , :00450AAE   , :00450E03
|:00450EA0   , :0045AFB4
|
:0044CAC0 A0D4FB5E00              mov al, byte ptr [005EFBD4]
:0044CAC5 C3                      ret


:0044CAC6 CC                      int 03
:0044CAC7 CC                      int 03
:0044CAC8 CC                      int 03
:0044CAC9 CC                      int 03
:0044CACA CC                      int 03
:0044CACB CC                      int 03
:0044CACC CC                      int 03
:0044CACD CC                      int 03
:0044CACE CC                      int 03
:0044CACF CC                      int 03

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044CC54(U)
|
:0044CAD0 56                      push esi
:0044CAD1 57                      push edi
:0044CAD2 8B7C240C                mov edi, dword ptr [esp+0C]
:0044CAD6 8BF1                    mov esi, ecx
:0044CAD8 8B06                    mov eax, dword ptr [esi]
:0044CADA 8B48F0                  mov ecx, dword ptr [eax-10]
:0044CADD 8B11                    mov edx, dword ptr [ecx]
:0044CADF 83E810                  sub eax, 00000010
:0044CAE2 6A01                    push 00000001
:0044CAE4 57                      push edi
:0044CAE5 50                      push eax

Line:125635 Pg 1461 and 1462 of 8322  Code Data @:0044CAC0 @Offset 0004CAC0h in File:C:\Program\ViceVersa Pro 2\Copy of ViceVersa.exe
```

Figure 2.8 Finding the pointer [005EFBD4]

Ahaa! The byte value stored in the pointer [005EFBD4] is moved into al. This is it!!

Now we have to find the location in the code where the registration info is verified and the non zero value is put into [005EFBD4]. Since this is probably accomplished by moving a value into the pointer we will now search for “[005EFBD4],” Also this one occurs only once in this particular target. But one should always keep an open mind; programmers are sometime sneaky and puts in several registration checks. ;-)

This is what you will find:

```
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger
Disassembler  Project  Debug  Search  Goto  Execute Text  Functions  HexData  Refs  Help

:0044F2DF 50                      push eax
:0044F2E0 E81BE9FFFF              call 0044DC00
:0044F2E5 8BCC                    mov ecx, esp
:0044F2E7 8964240C                mov dword ptr [esp+0C], esp
:0044F2EB 51                      push ecx
:0044F2EC C744241C00000000        mov [esp+1C], 00000000
:0044F2F4 E8B7EDFFFF              call 0044E0B0
:0044F2F9 83C404                  add esp, 00000004
:0044F2FC C7442418FFFFFFFF        mov [esp+18], FFFFFFFF
:0044F304 E887FDFFFF              call 0044F090
:0044F309 8B4C2410                mov ecx, dword ptr [esp+10]
:0044F30D A2D4FB5E00              mov byte ptr [005EFBD4], al
:0044F312 64890D00000000          mov dword ptr fs:[00000000], ecx
:0044F319 83C41C                  add esp, 0000001C
:0044F31C C3                      ret


:0044F31D CC                      int 03
:0044F31E CC                      int 03
:0044F31F CC                      int 03

* Referenced by a CALL at Address:
|:0044F440
|
:0044F320 B80C100000              mov eax, 0000100C
:0044F325 E8768B0E00              call 00537EA0
:0044F32A A188D25E00              mov eax, dword ptr [005ED288]
:0044F32F 89842408100000          mov dword ptr [esp+00001008], eax
:0044F336 6800080000              push 00000800
:0044F33B 8D442408                lea eax, dword ptr [esp+08]
:0044F33F 50                      push eax
:0044F340 C744240800000000        mov [esp+08], 00000000
:0044F348 FF1558E65E00            call dword ptr [005EE658]
:0044F34E 85C0                    test eax, eax
:0044F350 7515                    jne 0044F367

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044F377(C)
|
:0044F352 33C0                    xor eax, eax
:0044F354 8B8C2408100000          mov ecx, dword ptr [esp+00001008]

Line:130786 Pg 1521 and 1522 of 8322  Code Data @:0044F304 @Offset 0004F304h in File:C:\Program\ViceVersa Pro 2\Copy of ViceVersa.exe
```

Figure 2.9 Locating the registered check call

There you are! Follow the call at 0044F304 (call 0044F090). Now scroll down a bit. A call can return to the caller or jump on to another subroutine, in this case the call is ended by a RET without a preceding push and thus returning to the caller.

See that line at 0044F132 (xor al, al) ? An eXclusive OR (XOR) is made with al on al. If a value is XOR:ed with it self it is **always** zero. Hence, this is leaving al = 0! Changing this to '*mov al, 01*' is a suitable solution since it also is 2 OP codes long (B0 01).

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
:0044F0FC 83CAFF                  or edx, FFFFFFFF
:0044F0FF F0                      lock
:0044F100 0FC111                  xadd dword ptr [ecx], edx
:0044F103 4A                      dec edx
:0044F104 85D2                    test edx, edx
:0044F106 7F08                    jg 0044F110
:0044F108 8B08                    mov ecx, dword ptr [eax]
:0044F10A 8B11                    mov edx, dword ptr [ecx]
:0044F10C 50                      push eax
:0044F10D FF5204                  call [edx+04]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044F106(C)
|
:0044F110 8D47F0                  lea eax, dword ptr [edi-10]
:0044F113 C744243CFFFFFFFF        mov [esp+3C], FFFFFFFF
:0044F11B 8D480C                  lea ecx, dword ptr [eax+0C]
:0044F11E 83CAFF                  or edx, FFFFFFFF
:0044F121 F0                      lock
:0044F122 0FC111                  xadd dword ptr [ecx], edx
:0044F125 4A                      dec edx
:0044F126 85D2                    test edx, edx
:0044F128 7F08                    jg 0044F132
:0044F12A 8B08                    mov ecx, dword ptr [eax]
:0044F12C 8B11                    mov edx, dword ptr [ecx]
:0044F12E 50                      push eax
:0044F12F FF5204                  call [edx+04]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044F128(C)
|
:0044F132 32C0                    xor al, al
:0044F134 8B4C2434                mov ecx, dword ptr [esp+34]
:0044F138 64890D00000000          mov dword ptr fs:[00000000], ecx
:0044F13F 5F                      pop edi
:0044F140 5E                      pop esi
:0044F141 83C438                  add esp, 00000038
:0044F144 C3                      ret
```

Line:130560 Pg 1519 of 8322 Code Data @:0044F10D @Offset 0004F10Dh in File:C:\Program\ViceVersa Pro 2\Copy of ViceVersa.exe

Figure 2.10 Browsing the registered check call

```
???????????????????
  VICEVE~1.EXE   ↓FRO    PE.0044F132 a32 ---------     2846720|Hiew 6.82 (c)SEN
.0044F132: 32C0                        xor        al,al
.0044F134: 8B4C2434                    mov        ecx,[esp][34]
.0044F138: 64890D00000000              mov        fs:[00000000],ecx
.0044F13F: 5F                          pop        edi
.0044F140: 5E                          pop        esi
.0044F141: 83C438                      add        esp,038 ;"8"
.0044F144: C3                          retn
.0044F145: 6A01                        push       001
.0044F147: 8D4C244C                    lea        ecx,[esp][4C]
.0044F14B: E8F02AFBFF                  call      .000401C40   -----↑ (1)
.0044F150: 50                          push       eax
.0044F151: 8D44241C                    lea        eax,[esp][1C]
.0044F155: 50                          push       eax
.0044F156: E8A5E9FFFF                  call      .00044DB00   -----↑ (2)
.0044F15B: 83C408                      add        esp,008 ;"◘"
.0044F15E: 83781810                    cmp        d,[eax][18],010 ;"►"
.0044F162: C644243C02                  mov        b,[esp][3C],002 ;"☻"
.0044F167: 7205                        jb        .00044F16E   -----↓ (3)
.0044F169: 8B4004                      mov        eax,[eax][04]
.0044F16C: EB03                        jmps      .00044F171   -----↓ (4)
.0044F16E: 83C004                      add        eax,004 ;"♦"
.0044F171: 50                          push       eax
.0044F172: 8D4C2414                    lea        ecx,[esp][14]
1Global 2FilBlk 3       4ReLoad 5OrdOff 6lbyte  7Direct 8Xlat   9Auto   10Leave
```

Figure 2.11 Locating the registered check call in Hiew

Lets find this place in Hiew. Open the original target exe press F5 and enter the location 0044F132.

```
???????????????????
  VICEVE~1.EXE   ↓FWO    PE 0004F134 a32 <Editor>      2846720|Hiew 6.82 (c)SEN
 0004F132: B001                        mov        al,001 ;"☺"
 0004F134: 8B4C2434                    mov        ecx,[esp][34]
 0004F138: 64890D00000000              mov        fs:[00000000],ecx
 0004F13F: 5F                          pop        edi
 0004F140: 5E                          pop        esi
 0004F141: 83C438                      add        esp,038 ;"8"
 0004F144: C3                          retn
 0004F145: 6A01                        push       001
 0004F147: 8D4C244C                    lea        ecx,[esp][4C]
 0004F14B: E8F02AFBFF                  call       000001C40
 0004F150: 50                          push       eax
 0004F151: 8D44241C                    lea        eax,[esp][1C]
 0004F155: 50                          push       eax
 0004F156: E8A5E9FFFF                  call       00004DB00
 0004F15B: 83C408                      add        esp,008 ;"◘"
 0004F15E: 83781810                    cmp        d,[eax][18],010 ;"►"
 0004F162: C644243C02                  mov        b,[esp][3C],002 ;"☻"
 0004F167: 7205                        jb         00004F16E
 0004F169: 8B4004                      mov        eax,[eax][04]
 0004F16C: EB03                        jmps       00004F171
 0004F16E: 83C004                      add        eax,004 ;"♦"
 0004F171: 50                          push       eax
 0004F172: 8D4C2414                    lea        ecx,[esp][14]
1       2       3       4       5       6       7       8       9       10
```

Figure 2.12 Patching the application in Hiew

Then press F3 to edit, B0 01. Press F9 to save the changes. F10 to quit.

Run the target application, and…. Look! It´s registered!

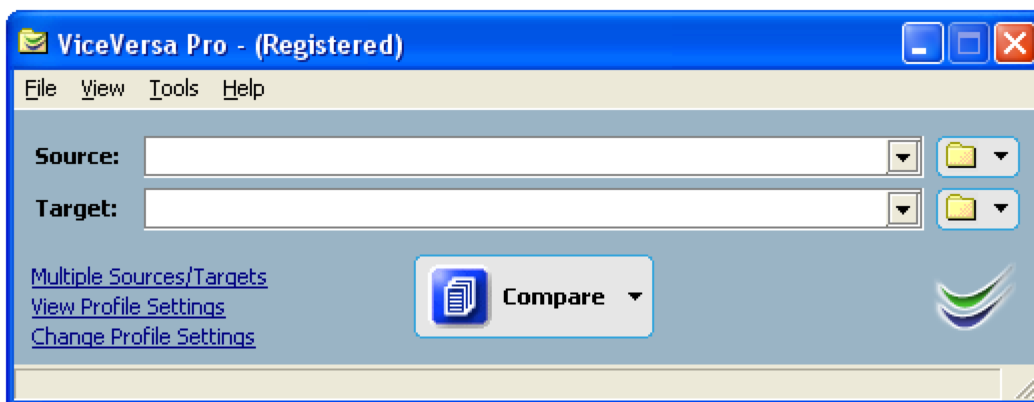

Figure 2.13 Task complete!

Method 2 – Interactive debugger (Ollydbg)

Launch Olly and open the target application.

Figure 2.14 Locating strings in Olly

Start with a ‘Search for .. All referenced text strings’

Move the marking to the top of the line and right click and choose ‘Search for text’.
Enter the text “- (Unregistered)”, if not sure of upper/lower case untick the Case sensitive box.

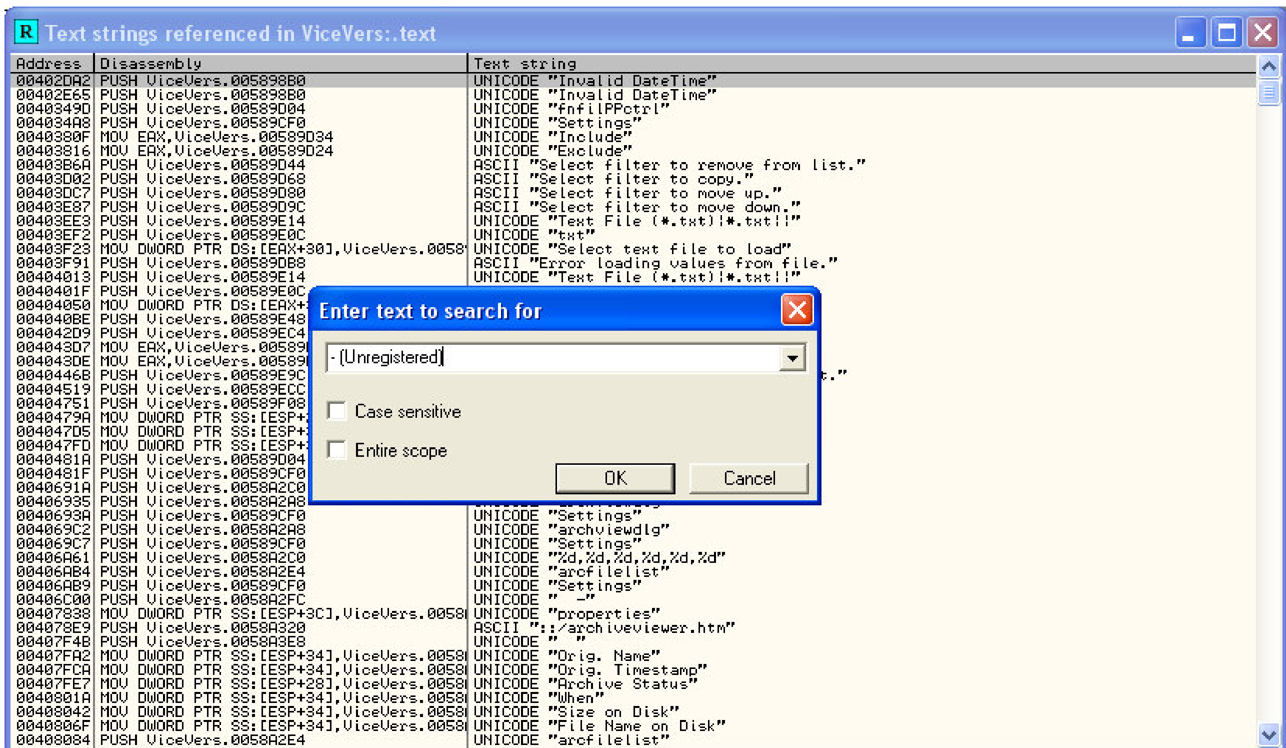

Figure 2.15 Locating strings in Olly

Press OK and you land here:

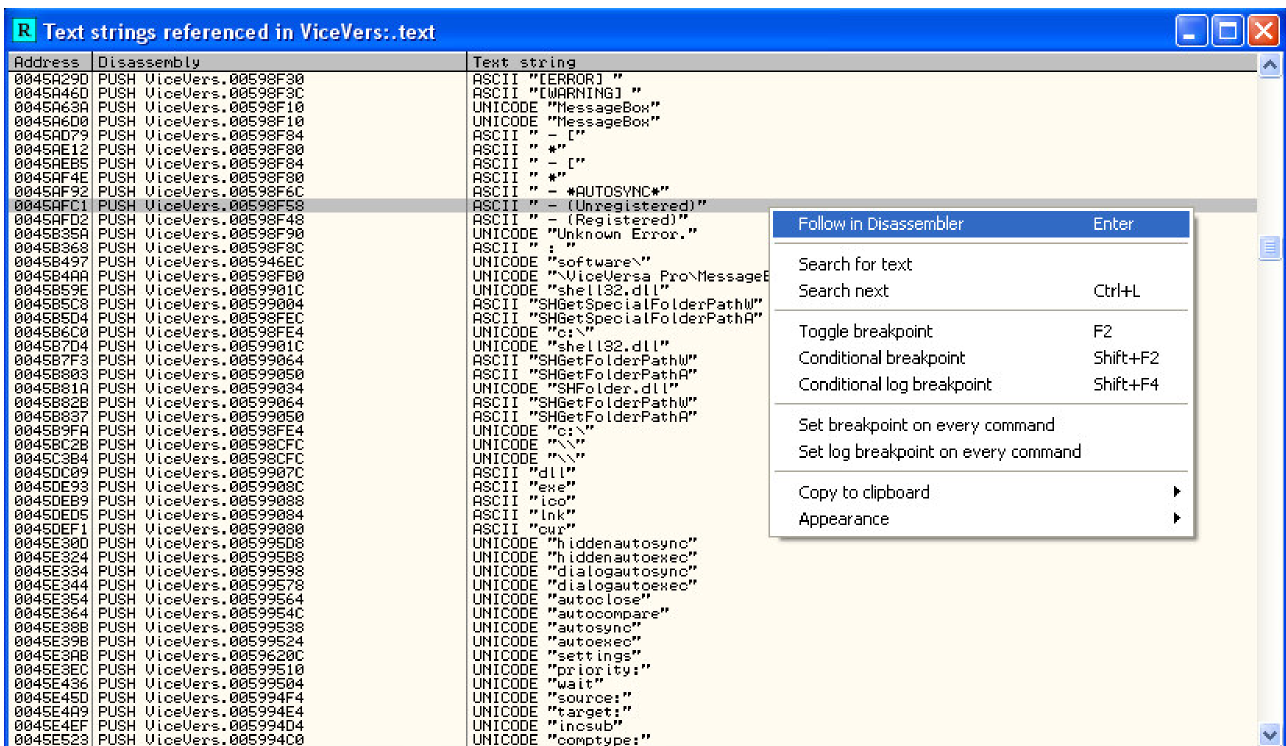

Figure 2.16 Following strings into code in Olly

When you have found the string right-click and chose 'Follow in Disassembler'.

Recognize this from earlier?

Figure 2.17 Finding the call to the pointer in Olly

Put a breakpoint (red) on the call to 0044CAC0 by marking the line and pressing F2, run the application by pressing F9. Press 'Ok, Evaluate' and Olly will break on the breakpoint, here:

Figure 2.18 Pointer Found

Figure 2.19 Locating the pointer in memory

Right click on the line 0044CAC0 and chose ‘Follow in dump.. Memory address’.

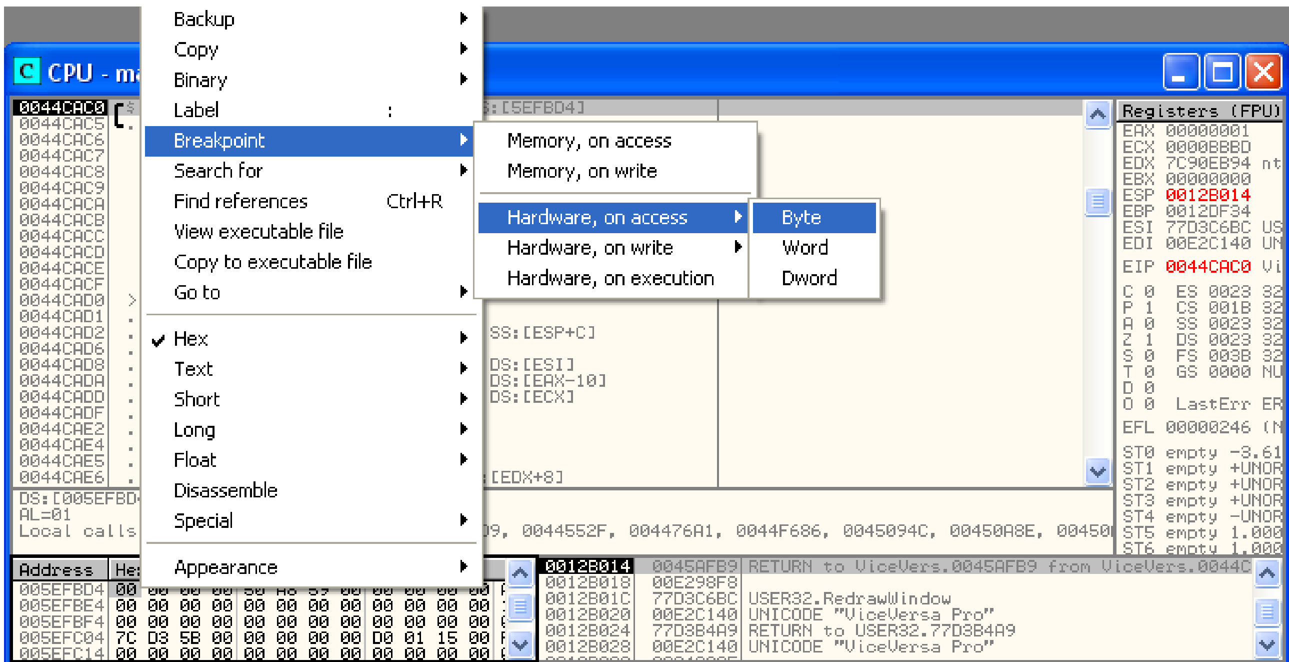

Figure 2.20 Setting a hardware breakpoint on pointer

And set a hardware breakpoint on access, byte. press ctrl+F2 to restart the application. Then F9 to run again.

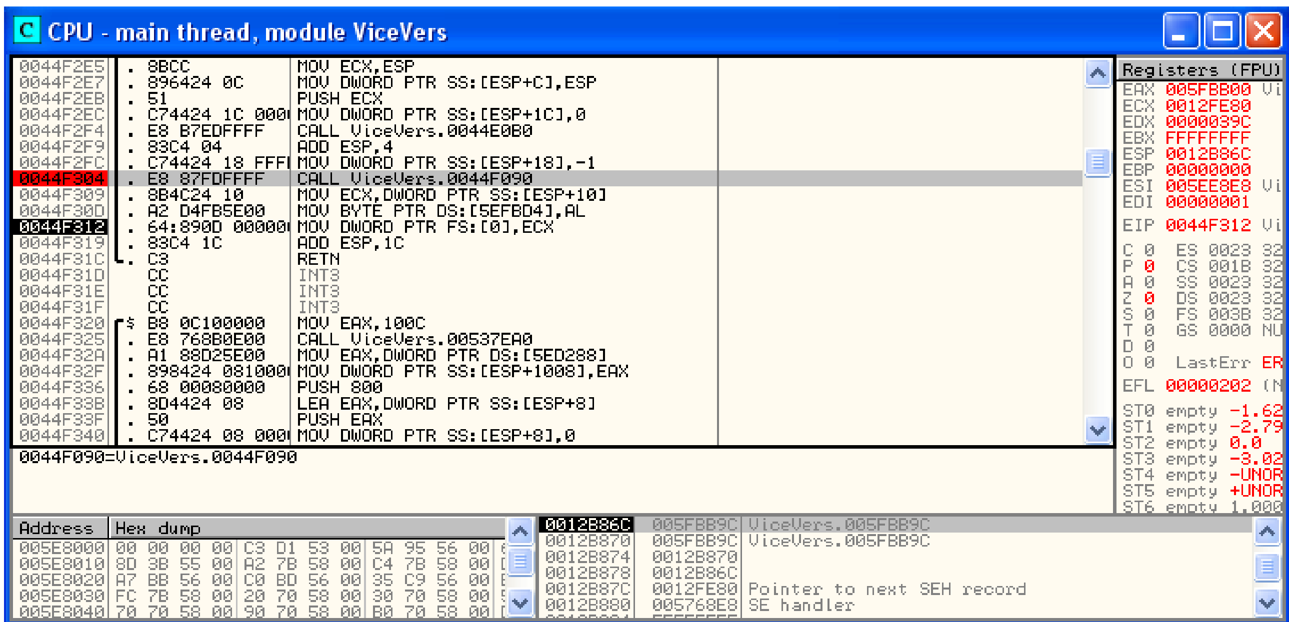

Figure 2.21 Finding the registered check call

Olly then breaks here (0044F312) scroll up a little and you will recognize the call to 0044F090! Set a breakpoint on that line (0044F304) and restart (ctrl-F2) and run again (F9).

When Olly breaks this time you should press F7 (step into the call) and you end up on 0044F304. Continue to execute the code step by step for a while by pressing F8.

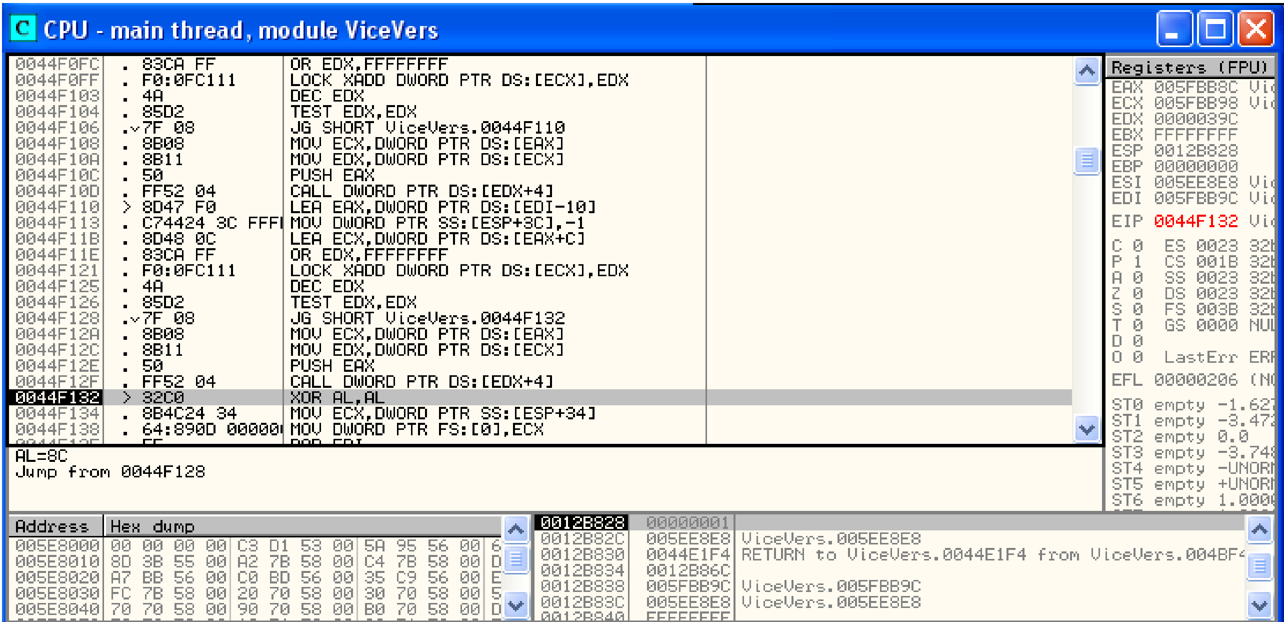

Figure 2.22 Tracing in the registered check call

Here is where al is set to 0! Press space to assemble.

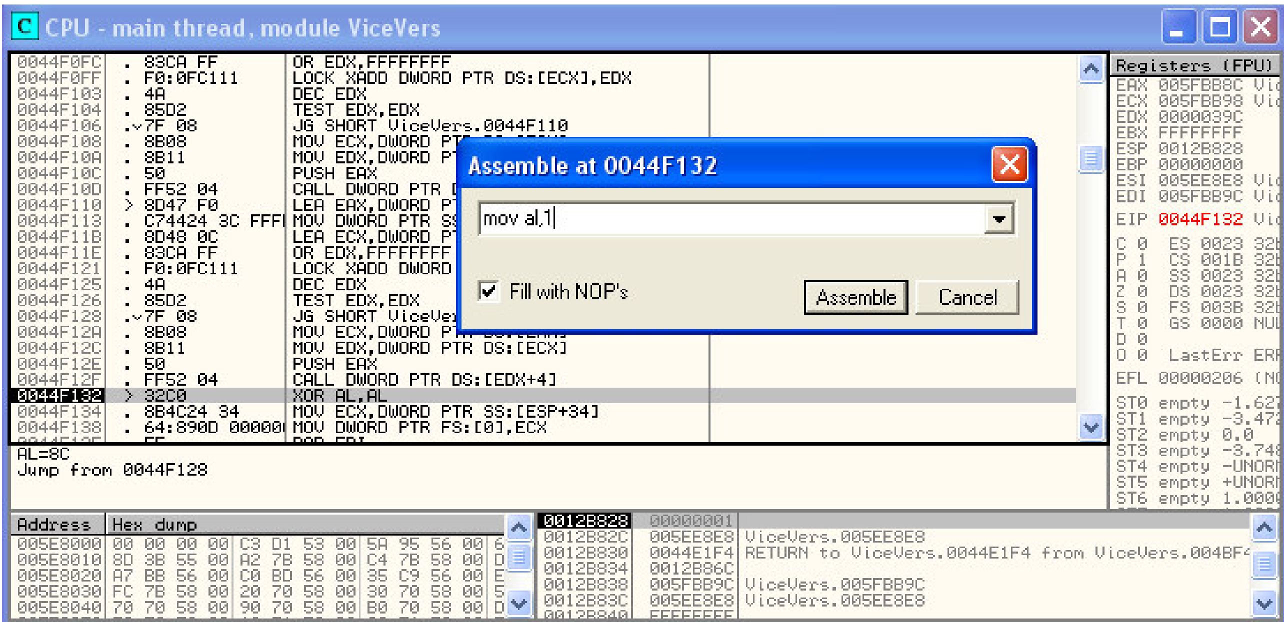

Figure 2.23 Patching with Olly

Apply the change by pressing Assemble and then cancel. Right-click and choose 'Copy to executable.. All modifications'. And then 'Copy All'. Then a new window appear, right-click on that window and then choose 'Save file'.

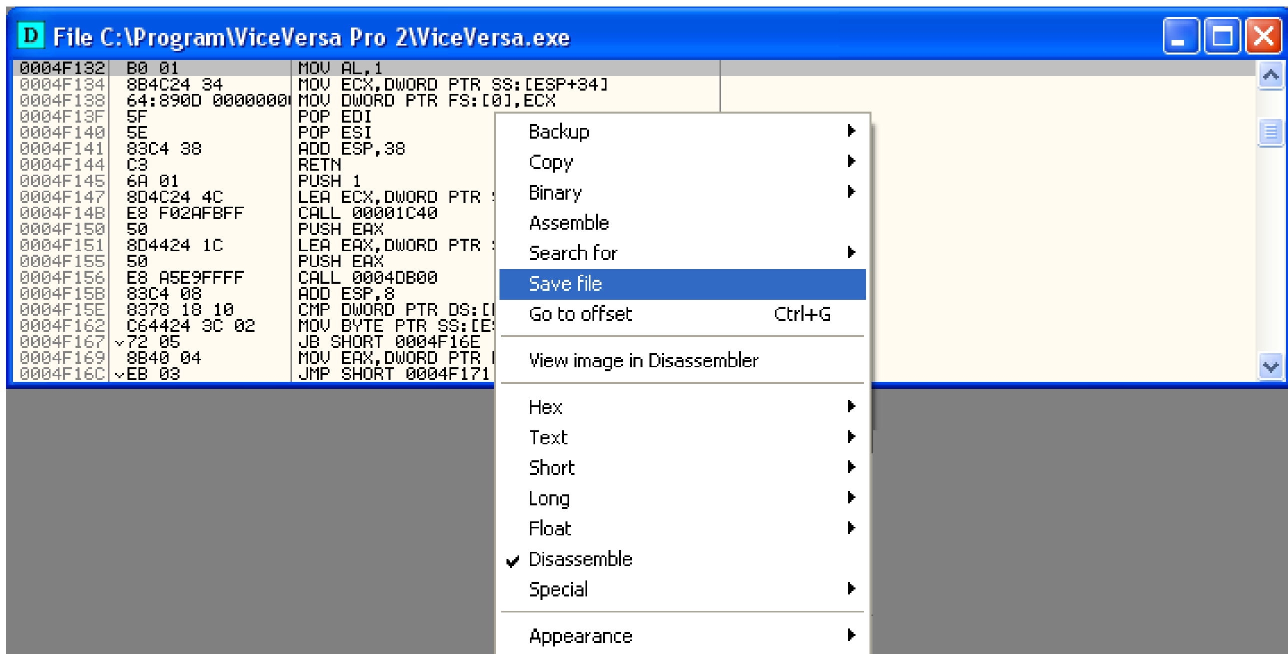

Figure 2.24 Saving the modified exe

Name the file to whatever you want, close Olly and you're done!

3. References

[1] Vortex168, *The Lazy Beginning Reverser's Guide to Windows Assembly,* 2002

[2] *Intel® Architecture Software Developer's Manual, Volume 2A, page* 3-486, *2005*

4. Conclusions

Using a single pointer dedicated for registration check is vulnerable indeed and should, if possible, be avoided. The use of an interactive debugger gives you the benefit of actually seeing the real values of pointers and registers as you go along, but is often more complex to handle and demands a great deal of knowledge to be used efficiently. The deadlist method is pretty straight forward but can sometimes be hard to follow due to the fact that you do not know the current state of registers and pointers.

5. Greetings

Fly out to all members of the victorious ARTeam and to all other whom in some way contribute to the scene.

Best regards, merry Christmas and happy reversing… potassium / ARTeam