

08.Debugging with W32Dasm, RVA, VA and offset, using LordPE as a hexeditor

2012년 1월 28일 토요일

오후 8:27

Hello everybody.

모두들 안녕.

Welcome to this Part 8 in my series about reversing for newbies/beginners.

나의 초보자 reversing series Part 8에 온 것을 환영해.

This "saga" is intended for complete starters in reversing, also for those without any programming experience at all.

이 "saga"는 완벽히 reversing 초보자를 맞춰서 만들어졌다. 또한 어떠한 programming 경험이 없어도 된다.

Lena151 (2006)

Set your screen resolution to 1152*864 and press F11 to see the movie full screen !!!

Again, I have made this movie interactive.

You screen 해상도를 1152*864로 설정해 그리고 full screen으로 movie를 보기 위해 F11를 눌러

So, if you are a fast reader and you want to continue to the next screen, just click here on this invisible hotspot. You don't see it, but it IS there on text screens.

그래서, 네가 이것을 빨리 읽고 다음 screen을 보고 싶다면, 보이는 hotspot 여기를 눌러. 보고 싶지 않을 때는 여기에 두지마.

Then the movie will skip the text and continue with the next screen.

Movie는 text와 다음 screen을 skip 할 수 있다.

If something is not clear or goes too fast, you can always use the control buttons and the slider below on this screen.

무언가 명확하지 않거나 빨리 넘기고자 할 때, 항상 control button과 이 screen 밑에 있는 slider 바를 사용해.

He, try it out and click on the hotspot to skip this text and to go to the next screen now!!!

도전해봐. 그리고 이 text와 다음 screen을 보기 위해 hotspot을 click 해.

During the whole movie you can click this spot to leave immediately

이 movie 어디에서나 즉시 떠나기 위해 이 spot을 click 할 수 있다.

1. Abstract

In this part 8, we will reverse a "real" application to learn something about working with the disassembler/debugger W32Dasm and LordPE as hexeditor, while studying the reversing of a registration scheme.

이번 part 8 에서, Reversing 의 등록 계획을 공부하는 동안 우리는 뭔가 배우기 위해서 disassembler/debugger W32Dasm 과 LoadPE 로 "real" application 을 reverse 할 것이다.

That is because indeed, the best practice is found in real applications.

Real application 에서 최고의 연습을 찾았다.

For better comprehension and if you are a newbie, I advise you to first see the previous parts in this series before seeing this movie.

더 좋은 이해력을 위해 네가 만약에 초보라면, 나는 이 movie 를 보기 전에 이 series 의 첫번째 part 부터 보라고 조언한다.

The goal of this tutorial is to teach you something about a program's behaviour.

이 tutorial 의 목표는 program's behaviour 를 가르치기 위함이다.

In my search not to harm authors, I found the program ArtGem v1.1 which was discontinued in 2003 and not downloadable any longer.

나의 연구는 제작자에게 해가 되지 않는다, 나는 ArtGem v1.1 이 2003 년 이후로 더 이상 download 되지 않는 것을 찾았다.

Taking a look in the specialized media, I also found this application to be "cracked" already.

특화된 media 를 가져봐, 나는 또한 이 application 의 "cracked" 된 버전을 이미 찾았다.

Here, this application is only chosen because it is ideal for this tutorial in reversing and it is targeted for educational purposes only.

여기, 이 application 은 오직 선택됐다. 왜냐하면 이것은 이 tutorial 은 reversing 과 교육적인 목적의 target 으로 이상적이다.

I hope you will exploit your newly acquired knowledge in a positive way.

너에게 네가 얻은 새로운 지식을 긍정적인 방향으로 사용하기를 바란다.

In this matter, I also want to refer to Part 1.

이 문제는, Part 1 을 참고하기를 바란다.

Set your screen resolution to 1152*864 and press F11 to see the movie full screen !!!

Again, I have made this movie interactive.

You screen 해상도를 1152*864 로 설정해 그리고 full screen 으로 movie 를 보기 위해 F11 를 눌러

So, if you are a fast reader and you want to continue to the next screen, just click here on this invisible hotspot. You don't see it, but it IS there on text screens.

그래서, 네가 이것을 빨리 읽고 다음 screen 을 보고 싶다면, 보이는 hotspot 여기를 눌러. 보고 싶지 않을 때는 여기에 두지마.

Then the movie will skip the text and continue with the next screen.

Movie 는 text 와 다음 screen 을 skip 할 수 있다.

If something is not clear or goes too fast, you can always use the control buttons and the slider below on this screen.

무언가 명확하지 않거나 빨리 넘기고자 할 때, 항상 control button 과 이 screen 밑에 있는 slider 바를 사용해.

He, try it out and click on the hotspot to skip this text and to go to the next screen now!!!

도전해봐. 그리고 이 text 와 다음 screen 을 보기 위해 hotspot 을 click 해.

During the whole movie you can click this spot to leave immediately

이 movie 어디에서나 즉시 떠나기 위해 이 spot 을 click 할 수 있다.

2. Tools and Target

The tools for today are : W32Dasm, LordPE and ... your brain.

오늘의 tools 은 : W32Dasm, LordPE and 너의 두뇌

W32Dasm was a program developed by Ursoft and sold as shareware.

Ursoft 에 의해 W32Dasm 은 발전됐다. 그리고 shareware 가 판매된다.

It was once a beloved tool for many reverser but was passed by other tools years ago.

많은 reverser 가 사랑하는 도구였지만 몇 년 전부터 다른 tools 에 의해 사용되지 않았다.

It would be a shame not to show some work with this program as it can still come in handy occasionally.

그것은 아직도 가끔 유용하게 사용할 수 있는데 이 program 과 함께 작품을 보여주는 게 부끄럽다.

The company no longer exists but the program can still be found at

Company 는 더 이상 존재하지 않는다. 그러나 program 은 여전히 찾을 수 있다.

<http://www.cracklab.ru/download.php?action=get&n=MzA=>

REMARK: this is the patched version, or else you won't be able to debug with the program!

주목: 이 patch 된 version 은, 네가 원하던 program 과 함께 하는 debug 가 아닐 수 있다.

LordPE is a free tool and can be down'ed at

LordPE 는 무료 툴이고 여기에서 다운 가능하다.

<http://www.cracklab.ru/download/php?action=get&n=MjA=>

Again, the brain is your responsibility ;)

다시, 너의 두뇌는 너의 책임감이다.

Todays target is a program called ArtGem1.1 Because it can no longer be downloaded, I have included it in this package for research.

오늘 target program 은 ArtGem1.1 로 불린다. 왜냐하면 그것은 더 이상 download 되지 않는다, 나는 이 package 에 연구를 위해 포함했다.

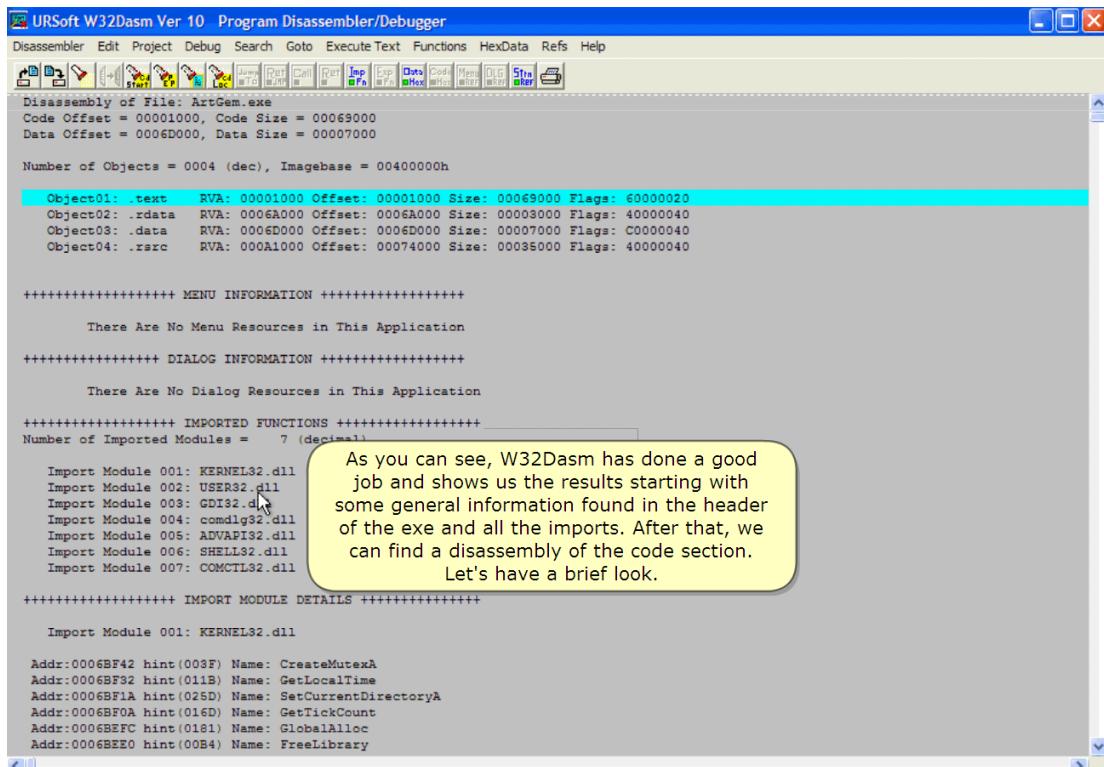
3. Behaviour of the program

What you see here this time is a lot different from what you're used to see. Indeed, it is W32Dasm that is opened.

이 시간에는 무엇이 보이나? 네가 지금까지 봐왔던 것과 꽤 다를 것이다. 정말, 이것은 W32Dasm 을 열어놓아서 그렇다.

W32Dasm is basically a disassembler (also called a deadlister because it makes a "deadlist") but it has some nice debugging features too. Let's load and disassemble ArtGem first

W32Dasm 은 기본적으로 disassembler 다.(deadlister 라 불리기도 한다. 왜냐하면 그것은 "deadlist"가 만들었기 때문이다.) 그러나 이것은 괜찮은 debugging 특징을 가진다. ArtGem 을 먼저 Load 하고 disassembler 하자.



As you can see, W32Dasm has done a good job and shows us the results starting with some general information found in the header of the exe and all the imports.

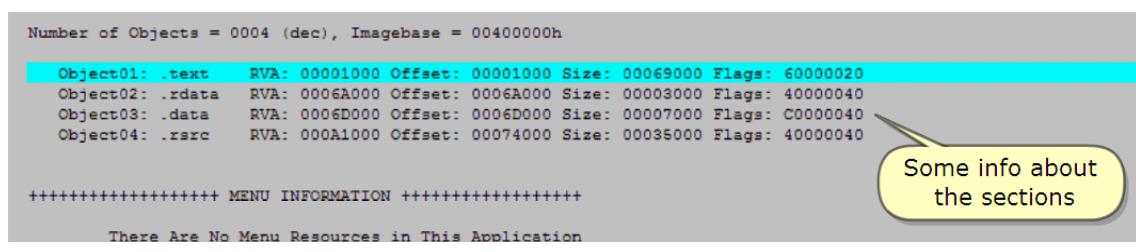
네가 볼 때, W32Dasm 은 좋은 능력을 가졌다. 그리고 exe 의 header 와 모든 imports 에서 약간의 시작하는 일반적인 정보를 찾은 results 보여준다.

After that, we can find a disassembly of the code section.

그 후에, 우리는 disassembly 의 code section 을 찾았다.

Let's have a brief look.

간단히 보자.



Some info about the sections

```
++++++ IMPORT MODULE DETAILS ++++++
Import Module 001: KERNEL32.dll

Addr:0006BF42 hint(003F) Name: CreateMutexA
Addr:0006BF32 hint(011B) Name: GetLocalTime
Addr:0006BF1A hint(025D) Name: SetCurrentDirectoryA
Addr:0006BF0A hint(016D) Name: GetTickCount
Addr:0006BEFC hint(0181) Name: GlobalAlloc
Addr:0006BEE0 hint(00B4) Name: FreeLibrary
```

and the imports

And the imports

Let's take a look and scroll down

밑을 보기 위해서 Scroll 내려봐.

```
:00401000 E80B000000          call 00401010
:00401005 E916000000          jmp   00401020
:0040100A 90                 nop
:0040100B 90                 nop
:0040100C 90                 nop
:0040100D 90                 nop
:0040100E 90                 nop
:0040100F 90                 nop

* Referenced by a CALL at Address:
| :00401000
|
:00401010 6A01              push 00000001
```

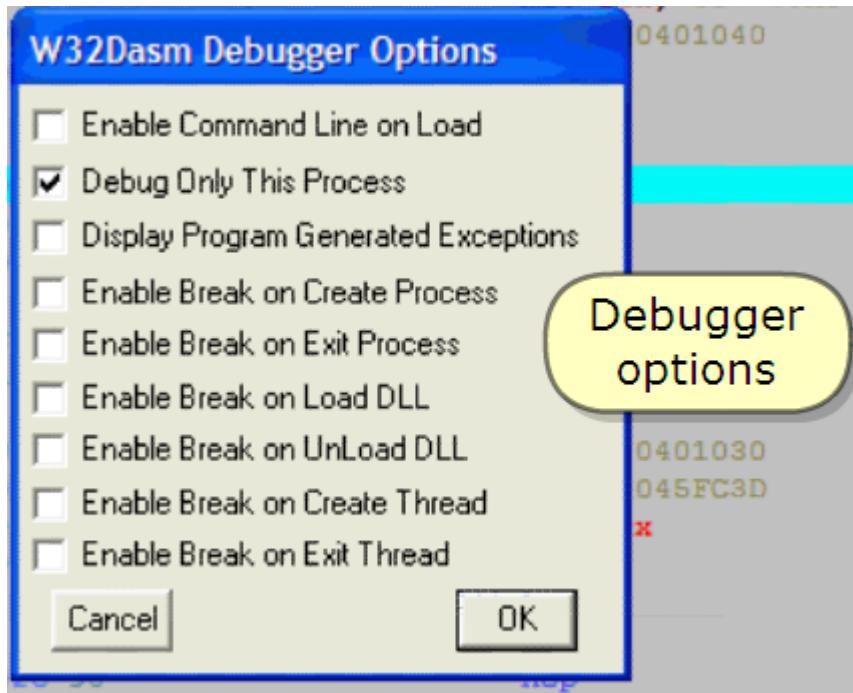
The start of the
disassembled code section

The start of the disassembled code section

Disassemble 된 code section 의 시작 부분이다.

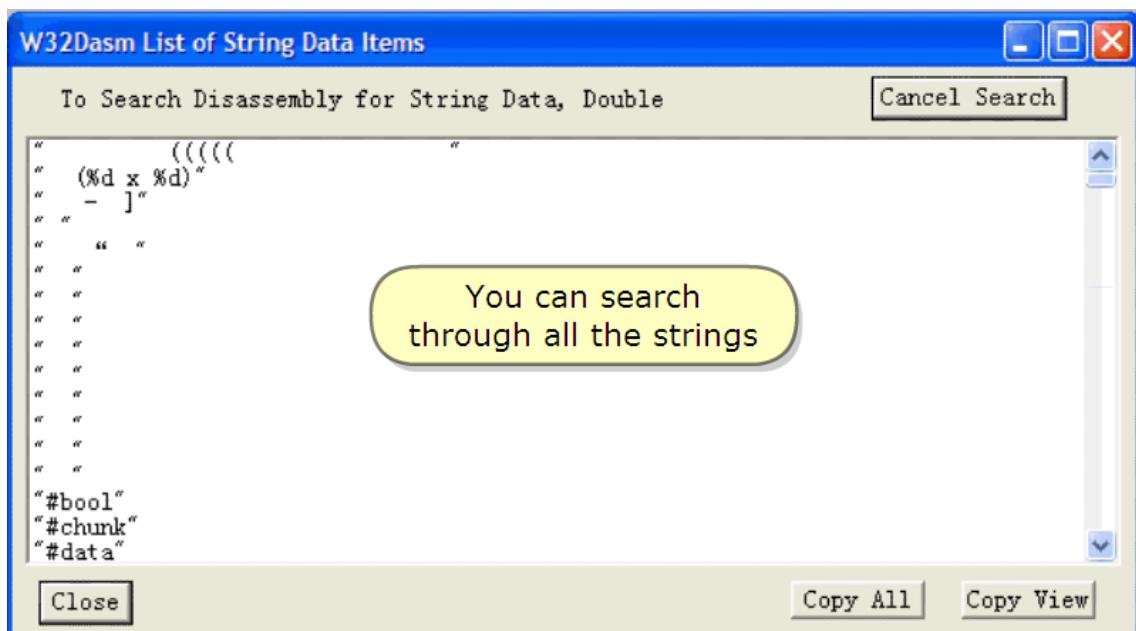
All right, let's now study the application and some features of this debugger together

좋아, 이제 application 과 debugger 의 특징을 함께 공부하자



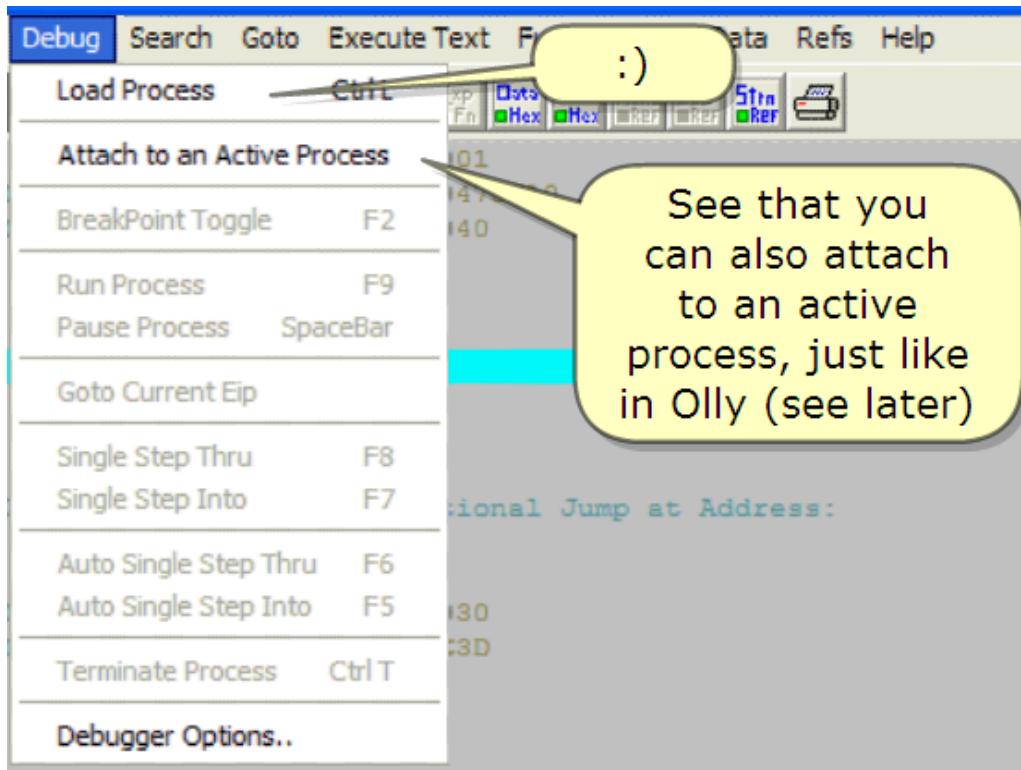
Debugger options

Debugger options



You can search through all the strings

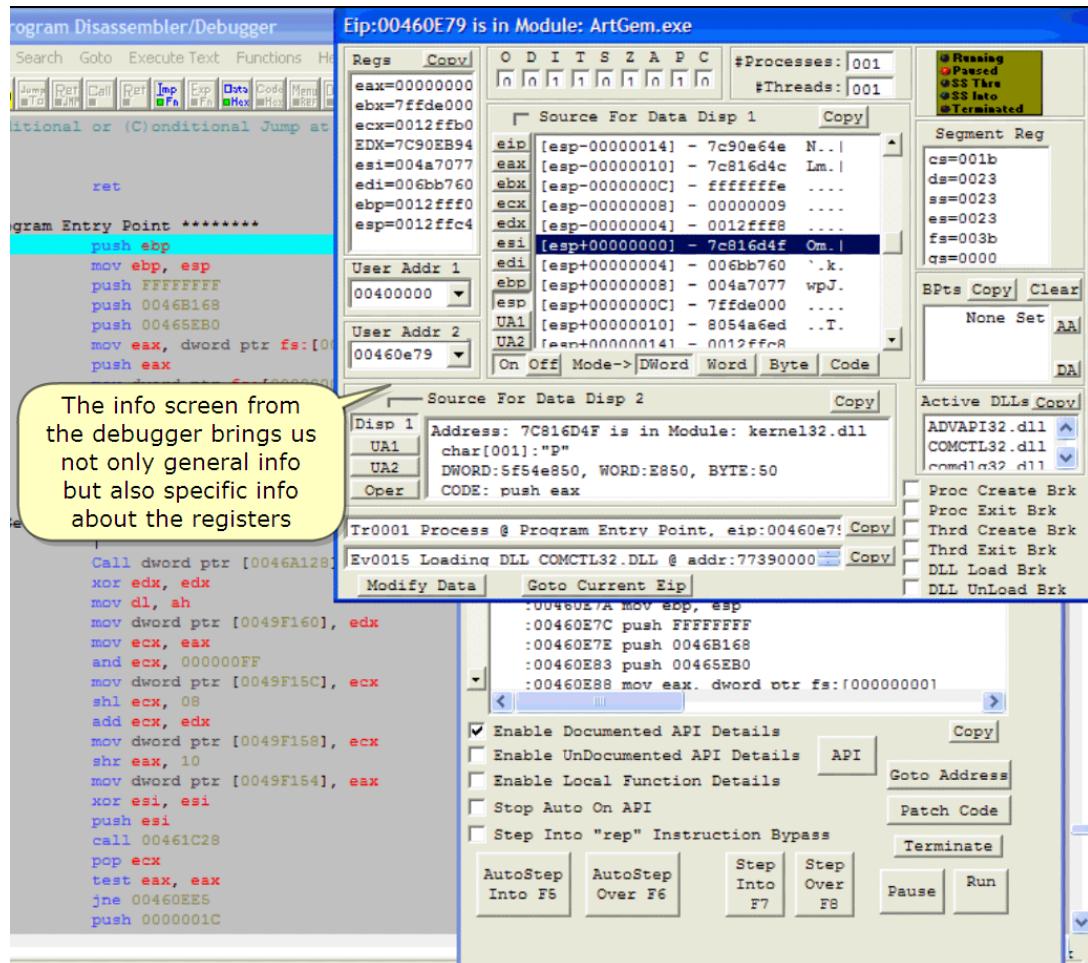
너는 모든 string 을 통해 찾을 수 있다.



See that you can also attach to an active process, just like in Olly (see later)

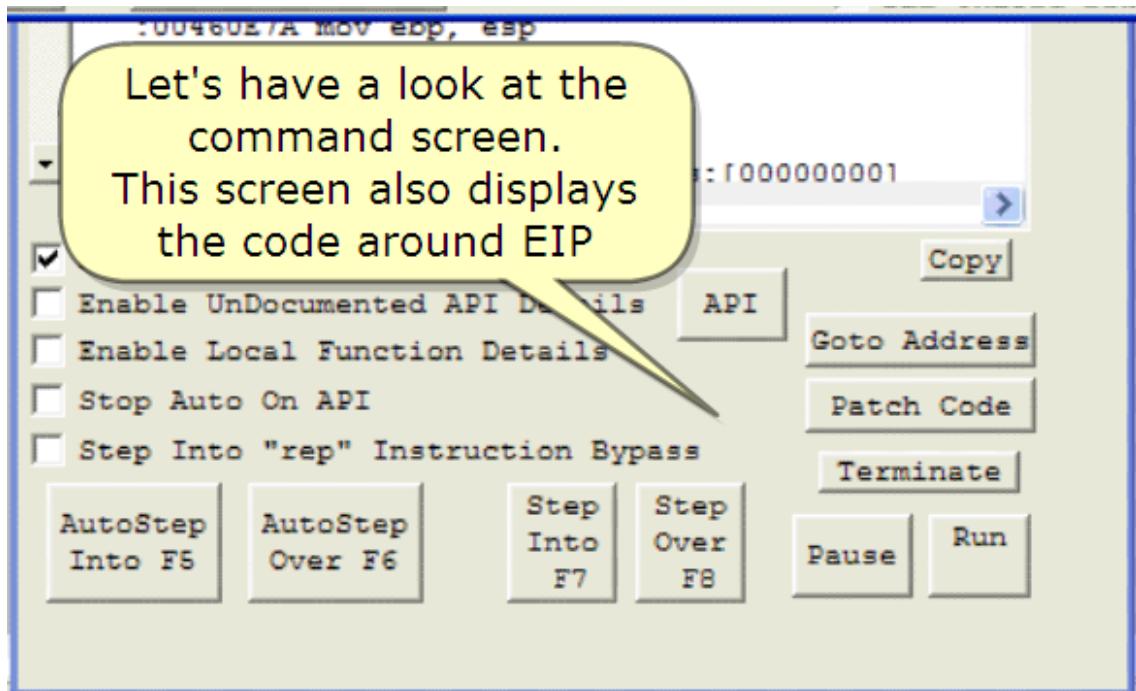
너는 또한 첨부한 active process 를 볼 수 있다. Olly 와 비슷하다.(나중에 보자)

:)



The info screen from the debugger brings us not only general info but also specific info about the registers

Info screen 은 Debugger에서 오직 일반적인 정보뿐만 아니라 특별한 register에 대한 정보들도 가져온다.

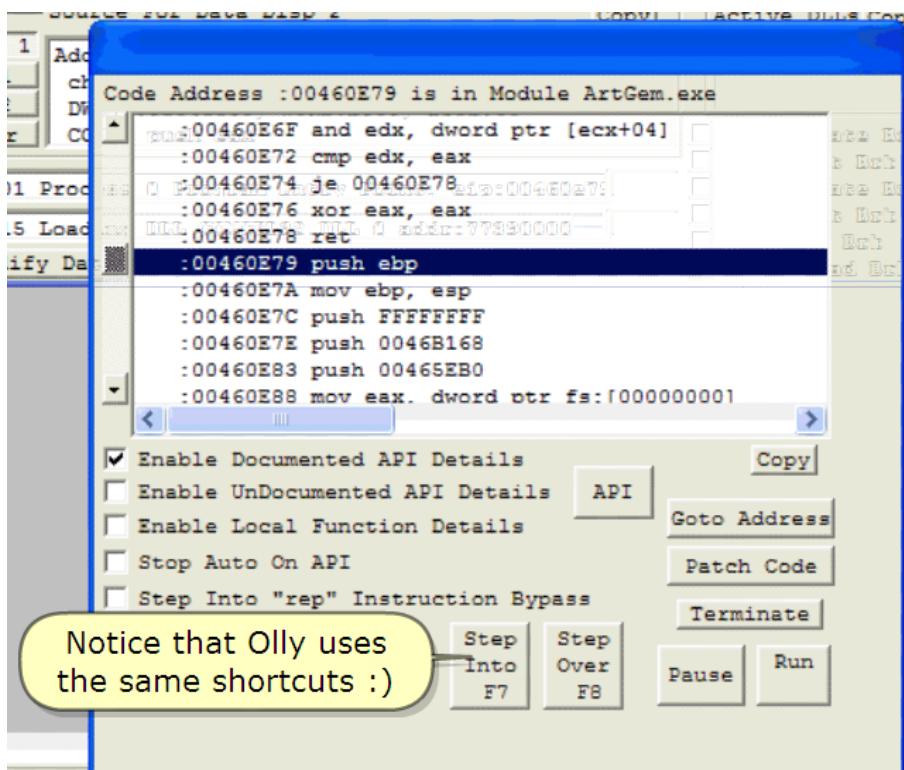


Let's have a look at the command screen.

Command screen 을 봐.

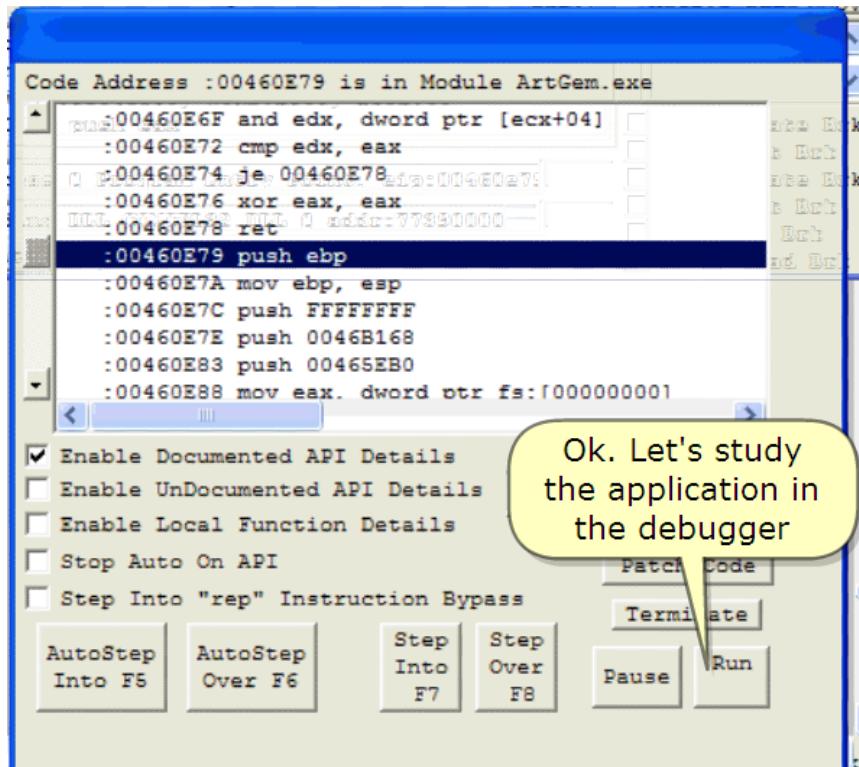
This screen also displays the code around EIP

Olly screen 은 또한 EIP code 에 대하여 보여준다.



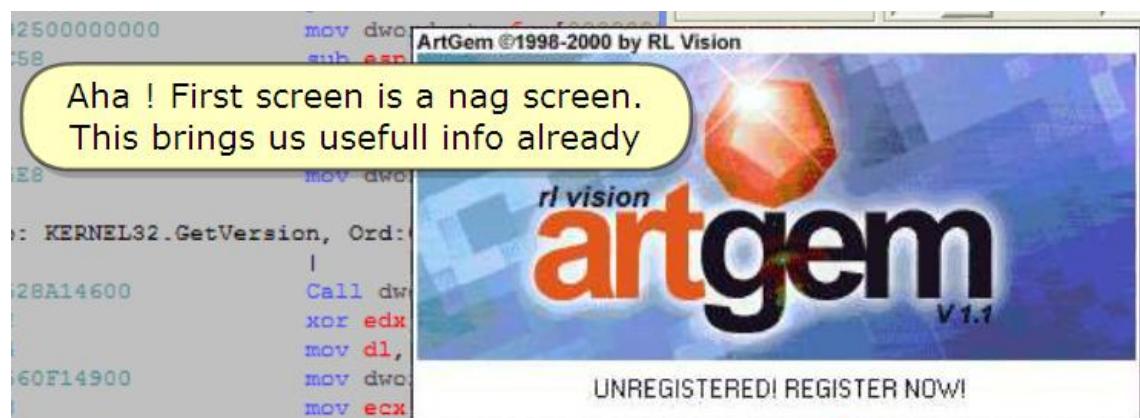
Notice that Olly uses the same shortcuts :)

Olly 는 같은 shortcut 사용하는 것을 보여준다.



Ok. Let's study the application in the debugger

Ok. Debugger에서 Application 을 공부하자.



Aha! First screen is a nag screen.

먼저 nag screen 이다.

This brings us useful info already

이미 유용한 정보를 가져온다.

And after some seconds, the main screen appears

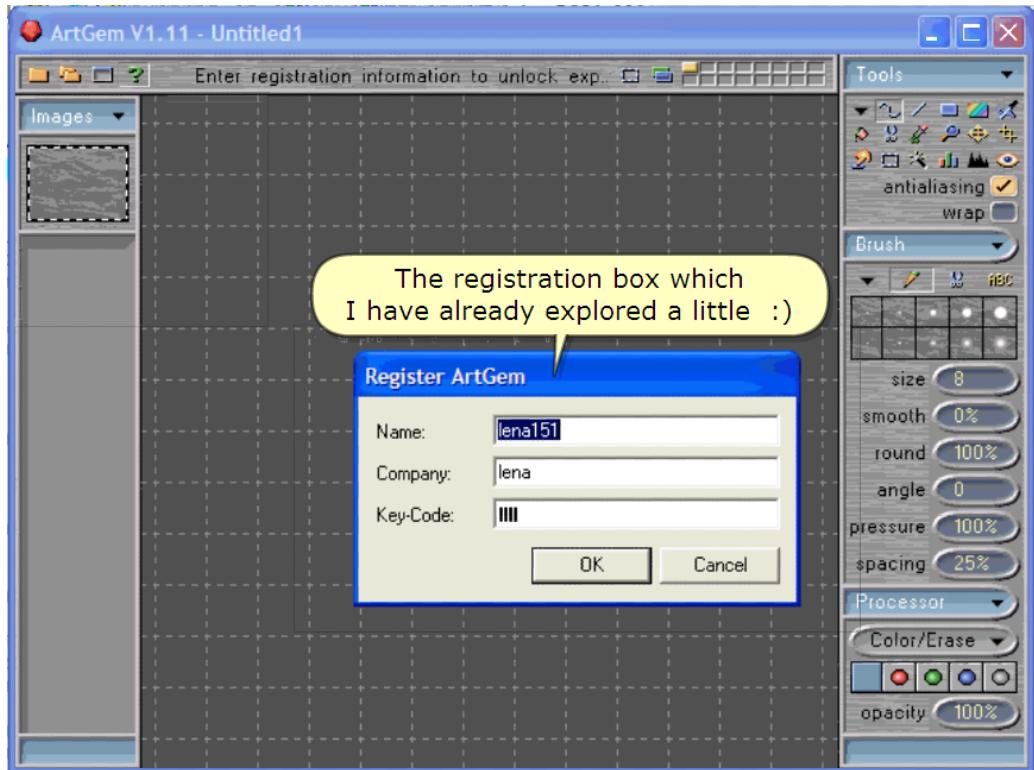
그리고 몇 초 후에, main screen 이 보인다.

:)

Mmmm, so the nag screen is also the about box screen !

음, about box screen 또한 nag screen 이다.

:)



The registration box which I have already explored a little :)

Registration box 는 내가 이미 약간 탐험했다.

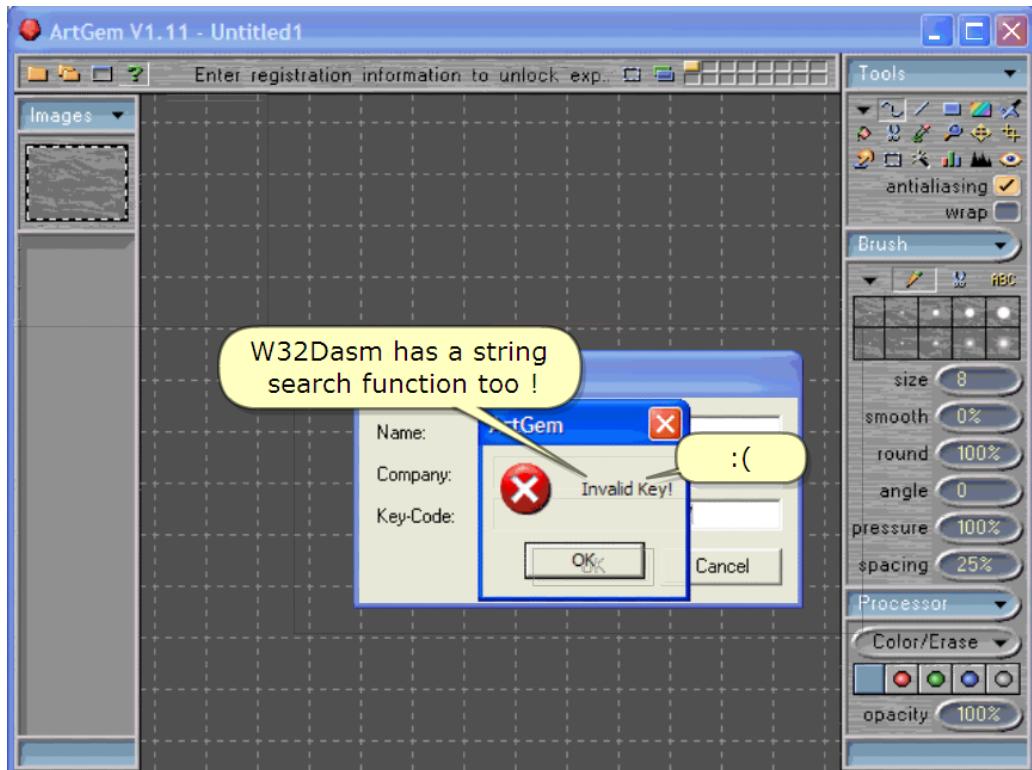
All right. This should give us enough information. Let's go!

좋아. 이것은 우리에게 충분한 정보를 주었다. 시작하자!

4. Finding the patches

Well, indeed, why not see what the registration looks like. Let's see it ...

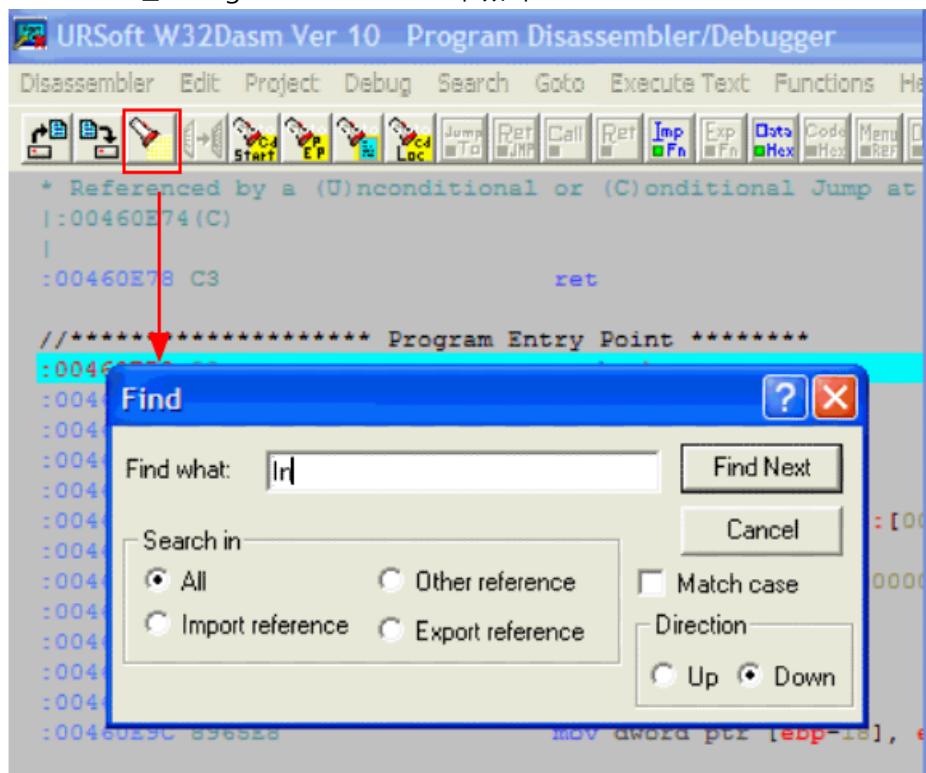
좋아, 우리는 왜 이 화면처럼 등록된 화면을 볼 수 없나? 보자.

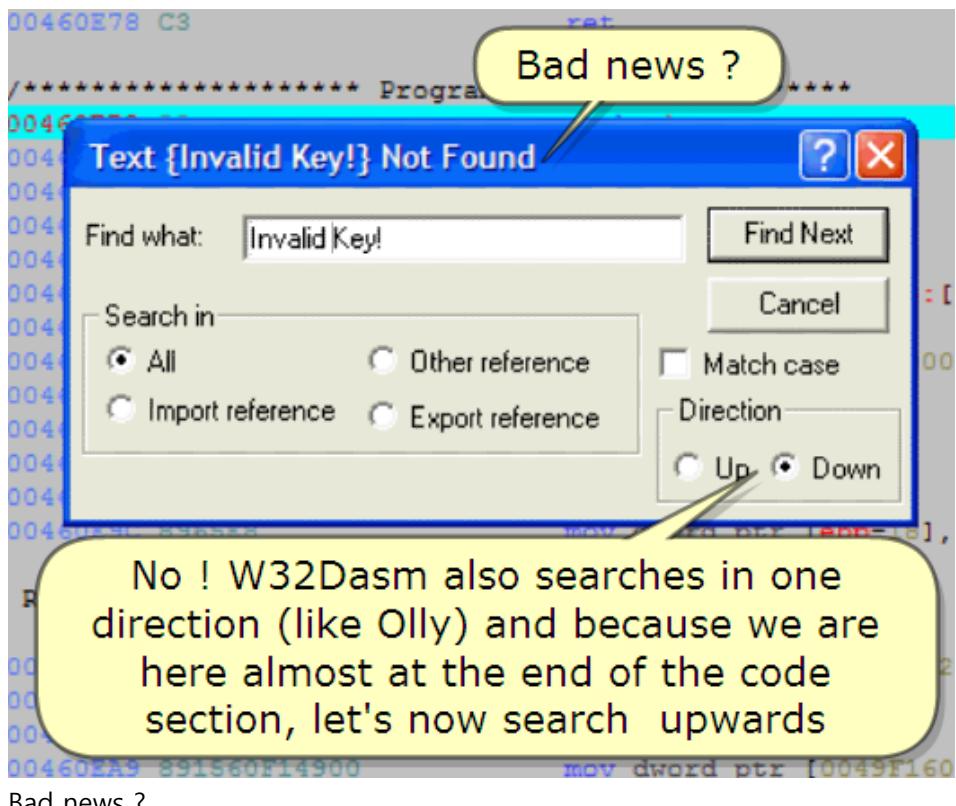


:)

W32Dasm has a string search function too !

W32Dasm 은 string search function 이 있다.

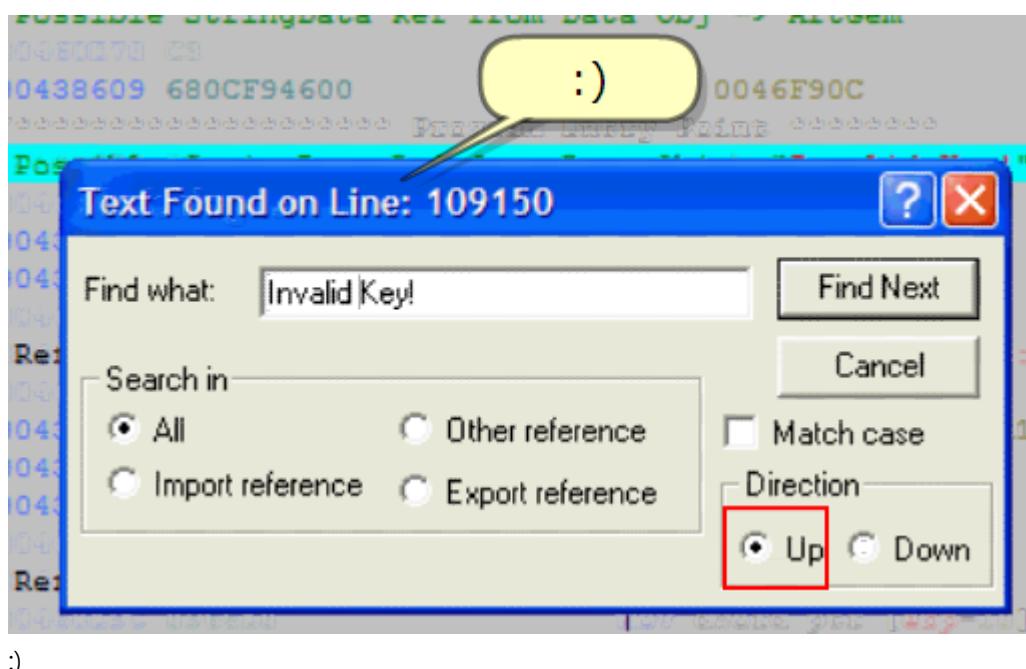




Bad news ?

No ! W32Dasm also searches in one direction (like Olly) and because we are here almost at the end of the code section, let's now search upwards

아니야! W32Dasm 또한 한 방향으로(Olly 와 같이) 찾는다. 그리고 왜냐하면 여기는 거의 code section 의 끝이다. Search 를 위쪽으로 해보자.



:)

URSoft W32Dasm Ver 10 Program Disassembler/Debugger

Disassembler Edit Project Debug Search Goto Execute Text Functions Help

:00438607 6A10 j s (0) unconditional push 00000010 ; al, Jump to
 :00460E74 (C)
 * Possible StringData Ref from Data Obj ->"ArtGem"
 :00460E70 C9 ; int
 :00438609 680CF94600 push 0046F90C
 //***** Program Entry Point *****
 * Possible StringData Ref from Data Obj ->"Invalid Key!"
 :004
 :0043860E 68CCF94600 |
 :00438613 51 :) CC
 :004
 * Reference To: USER32.MessageBoxA, Ord:01BEh : [0]
 :004
 :00438614 FF15ACA14600 Call dword ptr [0046A1AC]
 :0043861A 6A01 push 00000001
 :0043861C 53 push ebx
 :004
 :)

It is rather crowded here. I won't need the register info screen right away because I always take an overview first. So, I'll put it away for better visibility

이곳은 꽤 복잡해 보인다. 나는 옳은 방법에서 register info screen 이 필요하지 않다. 왜냐하면 나는 항상 먼저 관점을 가지기 때문이다. 나는 좀 더 좋은 방향으로 놓을 것이다.

:00438607 6A10 j s (0) unconditional push 00000010 ; al, Jump to
 :00460E74 (C)
 * Possible StringData Ref from Data Obj ->"ArtGem"
 :00460E70 C9 ; int
 :00438609 680CF94600 push 0046F90C
 //***** Program Entry Point *****
 * Possible StringData Ref from Data Obj ->"Invalid Key!"
 :004
 :0043860E 68CCF94600 |
 :00438613 51 push 0046F9CC
 :004
 * Reference To: USER32.MessageBoxA, Ord:01BEh : [0]
 :004
 :00438614 FF15ACA14600 Right. Press down button on the
 :0043861A 6A01 keypad to scroll down and take a look.
 :0043861C 53
 :004
 * Reference To: USER32.EndDialog, Ord:00B9h

Right. Press down button on the keypad to scroll down and take a look.

좋아. Scroll down 을 위해 Keypad 에 있는 Down button 을 눌러. 그리고 봐.

Study the code.

Code 를 공부하자.

The screenshot shows a debugger interface with assembly code. Several annotations with yellow boxes and arrows point to specific instructions:

- An annotation points to the instruction `push 00000001` with the text "This means that this piece of code is reached from a conditional jump at VA 438578".
- An annotation points to the instruction `pop esi` with the text "Here is a return...".
- An annotation points to the instruction `push ebx` with the text "Study the code.". A larger callout box contains the text "... so we get here from this conditional jump (always keep an eye on the jumps, especially the conditional jumps !)".
- An annotation points to the instruction `push ebx` with the text "But how do we get here ?".
- An annotation points to the instruction `push ebx` with the text "The program executes this piece of code to push the 'Invalid Key!' string on the stack".

A right-click context menu is open on the assembly code, showing options like "Enable Documented API Detail", "Step Into 'rep' Instruction", and buttons for "AutoStep Into F5", "AutoStep Over F6", and "Step Into F7".

The program executes this piece of code to push the "Invalid Key!" string on the stack

But how do we get here?

Program 은 이 조각 code 들은 "Invalid Key!" string 을 stack 에 넣는 것을 실행한다. 우리는 여기에 어떻게 올까?

Here is a return...

여기 return 이다.

... so we get here from this conditional jump (always keep an eye on the jumps, especially the conditional jumps !)

... 그래서 우리는 이 조건 jump 에서 여기를 얻을 수 있다.(항상 눈을 jump 에 유지해라, 특별한 조건 jump !)

This means that this piece of code is reached from a conditional jump at VA 438578

Hence, we need to go to VA 00438758 to see why we jumped here. So, scroll up

이이 조각 code 들은 조건 jump VA 438578 에서 여기로 이르렀다.

그리하여, 우리는 VA 00438758 를 보기 위해 jump 하는 것이 필요하다. 그래서, scroll 올려봐.

Huh? So, we jumped over the goodboy !!!

음? 그래서, 우리는 goodboy로 jump 했다.

Still better news. Scroll up to the cond jump ...

좋은 뉴스다. Scroll을 조건 jump 까지 열려 봐.

There we arrived.

우리가 도착했다.

This is the cond jump to the messagebox

이 조건 jump는 messagebox로 jump 한다.

Proceeded by a

TEST EAX, EAX

Which decides on jumping or not !!!

진행하다.

TEST EAX, EAX

이것이 jump 할지 말지 결정한다.

... and I suppose you already understand it's all decided in the proceeding call ???

이 진행중인 call에서 이것이 결정한다는 것을 너는 이미 이해했다고 생각한다.

Press F2 to set a BP exactly like in Olly(Shown in yellow)

BP를 set하기 위해 Olly에서처럼 F2를 눌러(yellow로 보인다)

Because we want to go look IN the call what sets EAX and makes us jump to the badboy !!!

왜냐하면 우리는 이 call에서 EAX를 무엇이 set하고 badboy로 jump하는지 보는 것을 원한다.

The screenshot shows the assembly view of OllyDbg with several assembly instructions highlighted in yellow. A yellow callout box points to one of the highlighted lines with the text: "Right. This line is breakpointed. We can press run to see the badboy and then register again to break in this BP". To the right of the assembly view is a stack dump window titled "Code Address :7C90EB94 is in Module ntdll.dll". The stack dump shows several memory locations starting with :7C90EB90, with the instruction at :7C90EB94 highlighted. Below the stack dump is a toolbar with various debugger controls. At the bottom of the assembly view, there is a status bar with the text "Line:109059 Pg 1226 of 2377 Code Data @:0043856E @Offset 0003856Eh in File A" and "File:7C90EB94 is in Module: ntdll.dll".

Right. This line is breakpointed. We can press run to see the badboy and then register again to break in this BP

좋아. 이 line에 breakpoint를 설치했다. 우리는 badboy를 보기 위해 run을 누른다. 그리고 다시 등록하기 위해 BP에서 멈춘다.

:)

We break in the BP !

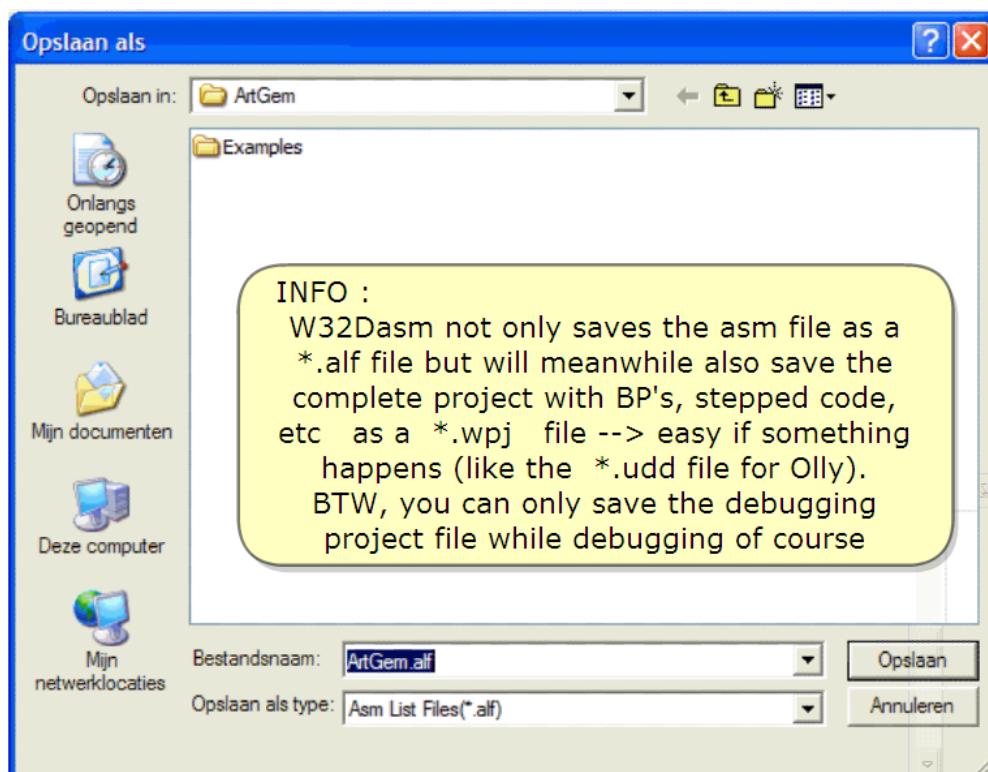
우리는 BP에 멈췄다.

So far, so good. W32Dasm also offers a possibility to save BP's, stepped code, etc

Let me show you how first

지금까지 좋아. W32Dasm은 또한 BP를 저장할 가능성을 제공한다. Code를 진행하자.

먼저 어떻게 보는지 보여주겠다.



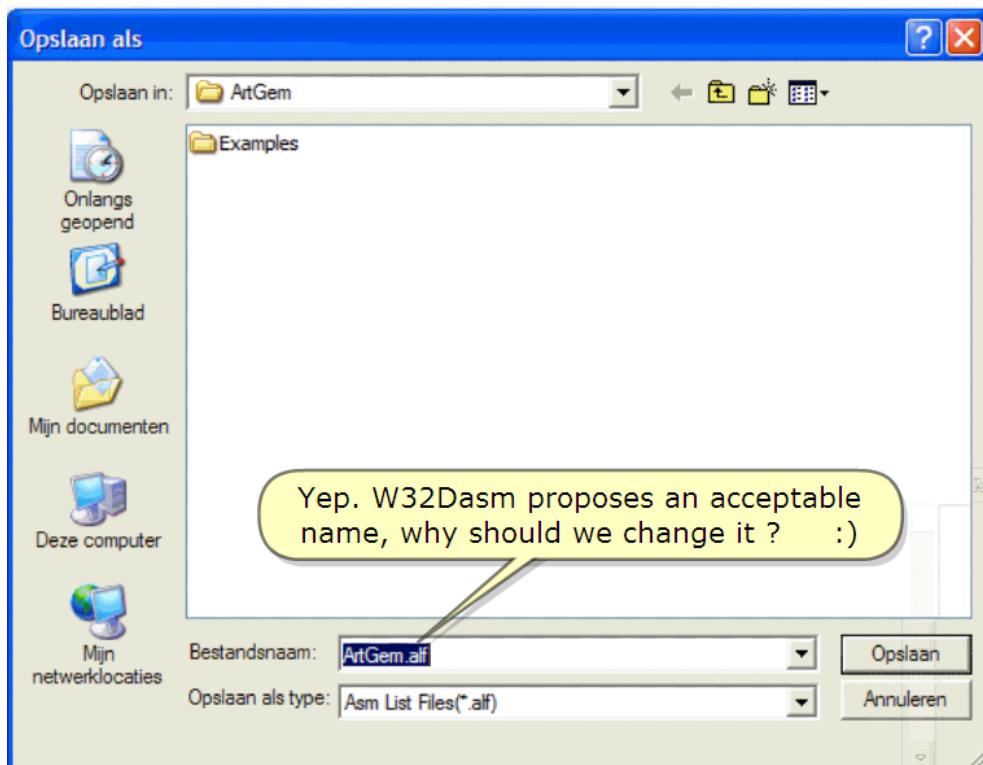
INFO :

W32Dasm not only saves the asm file as a *.alf file but will meanwhile also save the complete project with BP's, stepped code, etc as a *.wpj file --> easy if something happens (like the *.udd file for Olly).

W32Dasm은 alf file일 때 오직 asm file로만 저장하는 것이 아니다. 그러나 BP와 함께 완벽한 project를 저장할 수 있다. Code를 계속 진행하자. *.wpj file --> 무슨 일이 일어나기 쉽다.(Olly의 *.udd file과 같이)

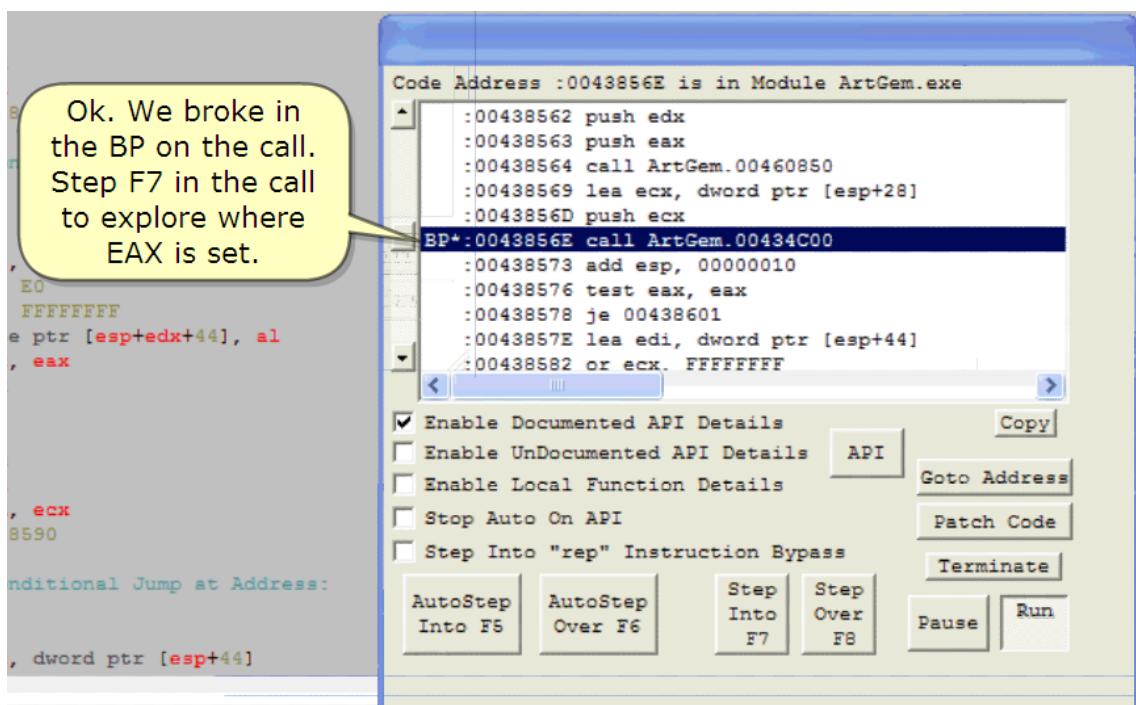
BTW, you can only save the debugging project file while debugging of course

By the way, 물론 debugging하는 동안 오직 debugging project file을 저장할 수 있다.



Yep. W32Dasm proposes an acceptable name, why should we change it? :)

예, W32Dasm 은 괜찮은 이름을 제안했다. 우리는 왜 바꾸려고 하는가?



Ok. We broke in the BP on the call. Step F7 in the call to explore where EAX is set.

Ok. 우리는 이 call 의 BP에서 멈쳤다. EAX 가 set 되는 곳을 F7 을 눌러 call 에서 찾자.

```
:00434BFE 90          nop
:00434BFF 90          nop

* Referenced by a CALL at Addresses:
!:0043856E , :00439E9B
!

:00434C00 63E024      sub esp, 00000024
:00434C03 8B442428    mov eax, dword ptr [esp+28]
:00434C07 53          push ebx
:00434C08 55          push ebp
:00434C09 56          push esi
:00434C0A 8B08        mov ecx, dword ptr [eax]
:00434C0C 57          push edi
:00434C0D 894C2424    mov dword ptr [esp+24], ecx
:00434C11 8B5004      mov edx, dword ptr [eax+04]
:00434C14 89542428    mov dword ptr [esp+28], edx
:00434C18 8B4808      mov ecx, dword ptr [eax+08]
:00434C1B 89C4242C    mov dword ptr [esp+2C], ecx
:00434CLF 8B500C      mov edx, dword ptr [eax+0C]
:00434C22 8A44242E    mov al, byte ptr [esp+2E]
:00434C26 3C2D        cmp al, 2D
:00434C28 89542430    mov dword ptr [esp+30], edx
:00434C2C 0F855D010000 jne 00434D8F
:00434C32 8A442432    mov al, byte ptr [esp+32]
:00434C36 84C0          test al, al
:00434C38 0F8551010000 jne 00434D8F
:00434C3E 33D2        xor edx, edx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
!:00434C5E(C)
!

:00434C40 014C1142        mov cl, byte ptr [esp+edi+124]

```

Mmmmm, this looks like a verification of the serial. Ok. Now step F8 to explore and see what all goes on here.

Code Address :00434C00 is in Module ArtGem.exe

- :00434BF9 add esp, 00000020
- :00434BFC ret
- :00434BFD nop
- :00434BFE nop
- :00434BFF nop
- :00434C00 sub esp, 00000024**
- :00434C03 mov eax, dword ptr [esp+28]
- :00434C07 push ebx
- :00434C08 push ebp
- :00434C09 push esi
- :00434C0A mov ecx, dword ptr [eax]

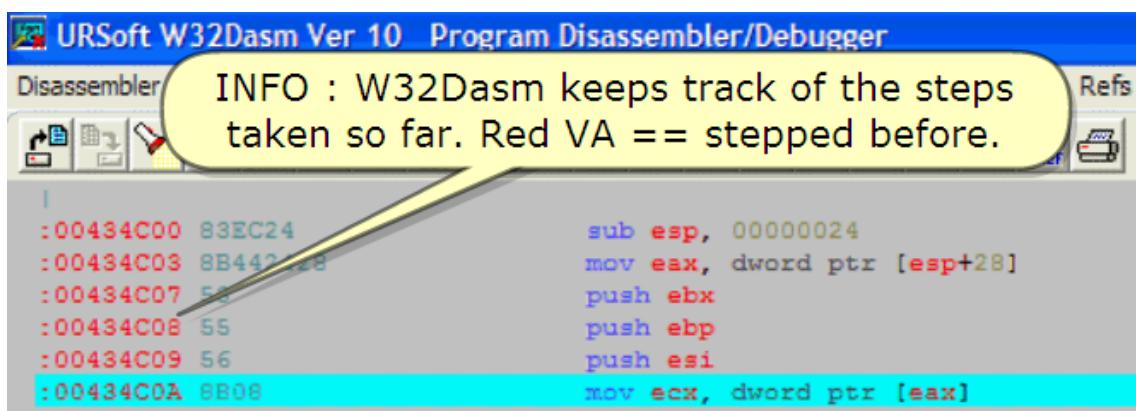
Enable Documented API Details

Mmmmm, this looks like a verification of the serial. Ok.

Now step F8 to explore and see what all goes on here.

음, serial 을 검증하는 것 같다. Ok.

이제 탐험하기 위해 F8 을 누르자. 그리고 어디로 가는지 봐라.



INFO : W32Dasm keeps track of the steps taken so far. Red VA == stepped before.

W32Dasm 은 step 의 track 을 유지한다. Red VA == 전에 실행됐다.

:00434C11	BB5004	mov edx, dword ptr [eax+04]	
:00434C14	B9542428	mov dword ptr [esp+28], edx	
:00434C18	BB4808	mov ecx, dword ptr [eax+08]	
:00434C1B	B94C242C	mov dword ptr [esp+2C], ecx	
:00434C1E	BB500C	mov edx, dword ptr [eax+0C]	
:00434C21	BA44242E	mov al, byte ptr [esp+2E]	
:00434C26	3C2D	cmp al, 2D	Seems like preparing a certain spot from the serial in AL and then comparing with 2Dh (ascii "-")
:00434C28	B9542430	mov dword ptr [esp+30], edx	
:00434C2C	0F855D010000	ine 00434DBF	

Seems like preparing a certain spot from the serial in AL and then comparing

With 2Dh (ascii "-")

AI의 serial 부터 중요한 spot 을 준비하는 것 같다. 그리고 비교한다.

2D ≡ (ascii "-")

And we jump if this spot is not "-"

Ok. Let's jump ...

이 spot 이 "-"이 아니라면 jump 한다.

Ok. 계속 jump 하자.

```
:00434D8D 7414          je 00434DA3

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00434C2C(C), :00434C38(C), :00434D25(C), :00434D2C(C), :00434D3E(C)
|:00434DSF(C), :00434D66(C), :00434DA1(C), :00434DB0(C)
|
:00434D8F SF            pop edi
:00434D90 5E            pop esi
:00434D91 5D            pop ebp
:00434D92 33C0          xor eax, eax
:00434D94 5B            pop ebx
:00434D95 83C424        add esp, 00000024
:00434D98 C3            ret

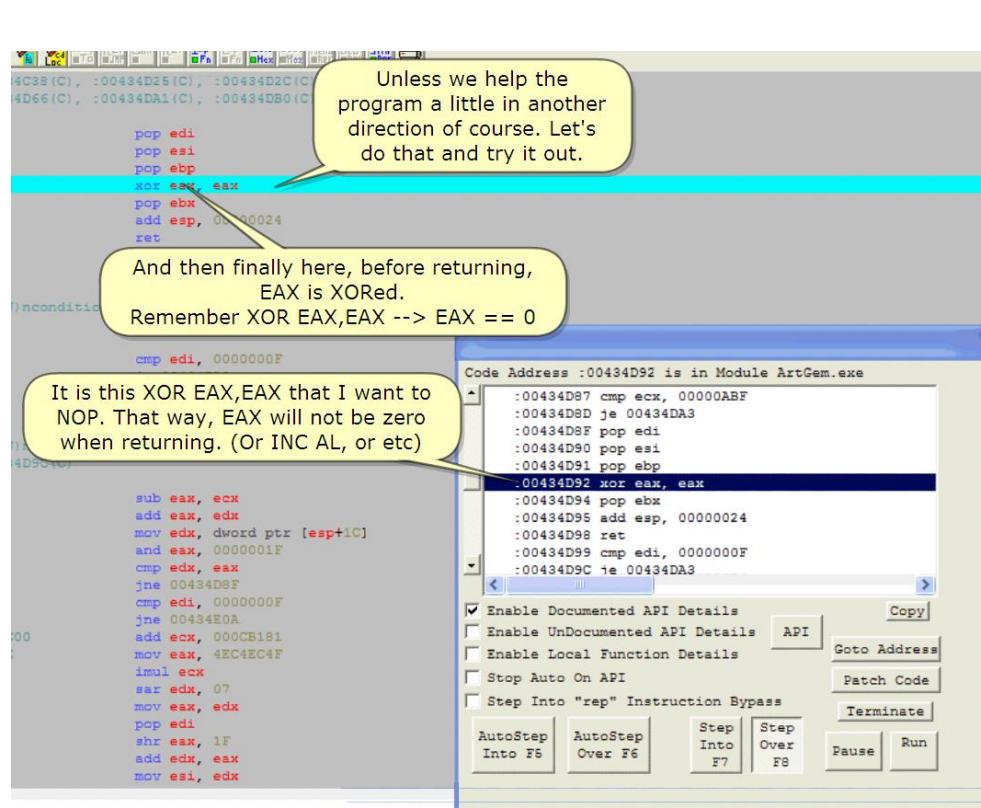
* Referenced by a (U)nconditional or (C)onditional
|:00434D85(C)
|
:00434D99 83FF0F        cmp edi, 0000000F
:00434D9C 7405          je 00434DA3
```

... oops, at once to the end of the routine instead of comparing the rest of the serial

웁스, Serial 의 나머지를 검사하는 대신에 routine 의 끝으로 왔다.

See all the other jumps here if a bad serial is detected along the routine

Bad serial 은 이 routine 에 의해서 밝혀졌다면 여기의 모든 다른 jump 를 봐라.



And then finally here, before returning, EAX is XORed.

그리고 마지막으로 여기, EAX 를 XOR 후에 returning 한다.

Remember XOR EAX, EAX --> EAX == 0

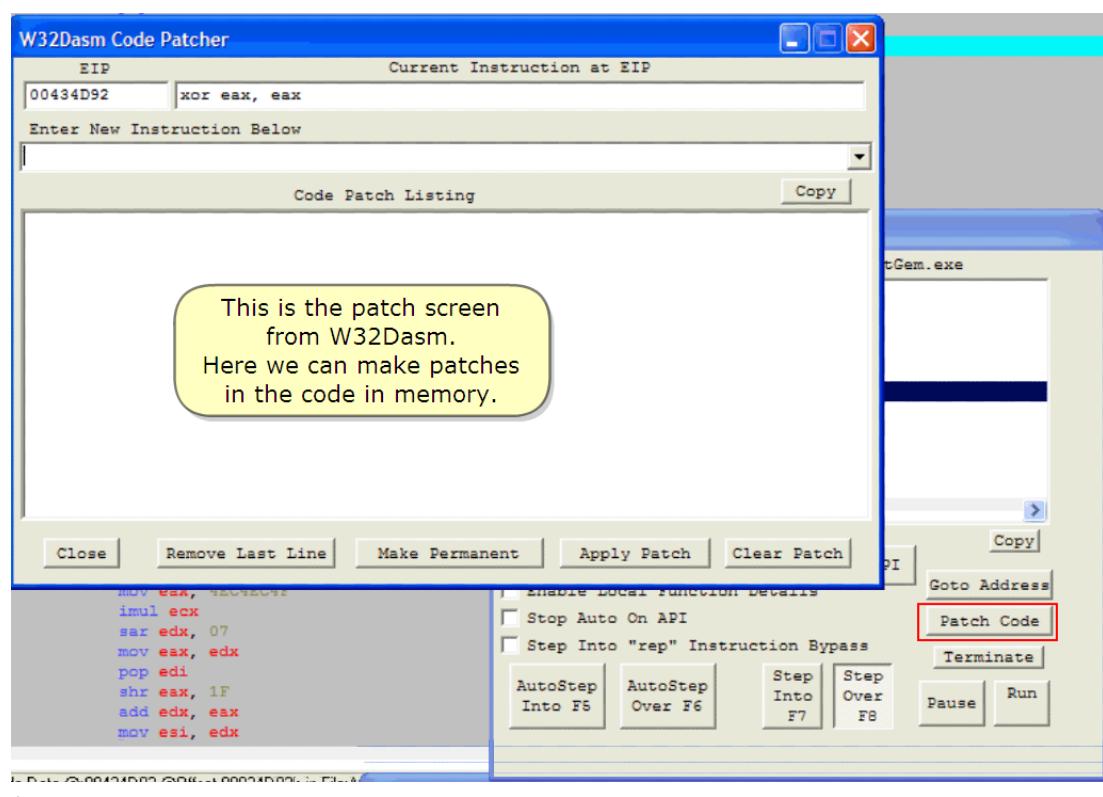
기억해 XOR EAX, EAX --> EAX == 0 이다.

Unless we help the program a little in another direction of course. Let's do that and try it out.

우리가 program 을 다른 방향으로 가게 돋지 않는 한. 그것을 실행해 봐.

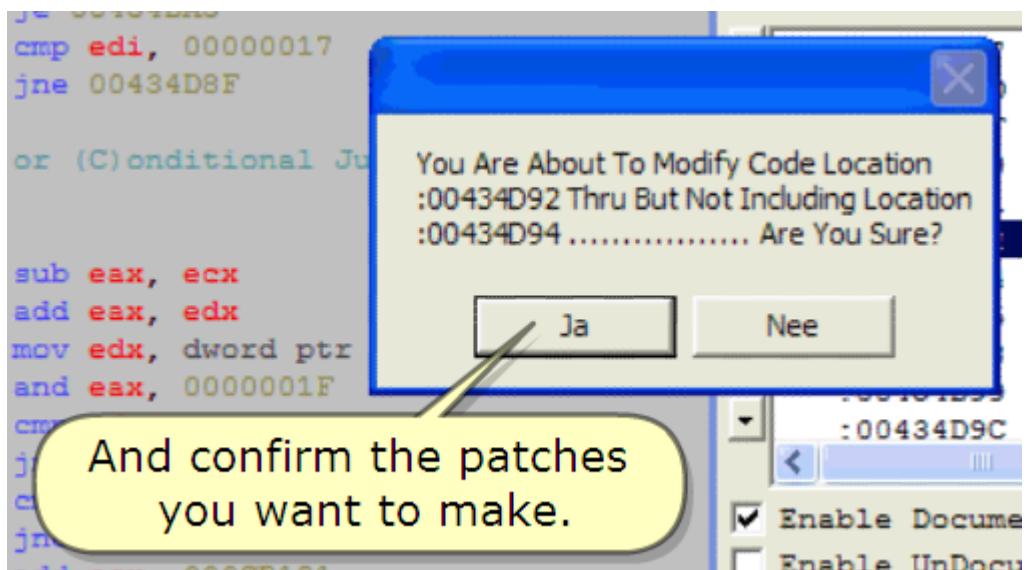
It is this XOR EAX, EAX that I want to NOP. That way, EAX will not be zero when returning. (Or INC AL, or etc)

이 XOR EAX, EAX 를 NOP 하기를 원한다. 이 방법은, returning 할 때 EAX 가 zero 가 되지 않는다. (아니면 INC al, etc)



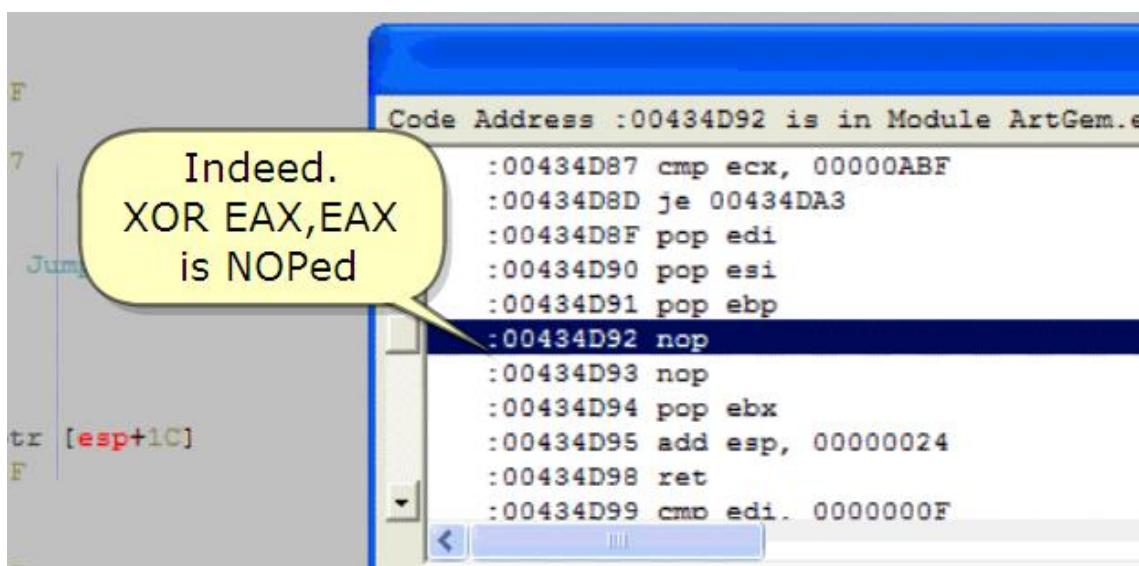
This is the patch screen from W32Dasm. Here we can make patches in the code in memory.

W32Dasm 의 patch screen 이다. 우리는 memory 의 code 에서 patch 할 수 있다.



And confirm the patches you want to make

그리고 네가 만든 patch 를 확인할 수 있다.



Indeed.

XOR EAX, EAX is NOPed

정말

XOR EAX, EAX 는 NOP 됐다.

Also place a BP on this line.

또한 이 line 에 BP 를 설치해.

Press F2

Code Address :00434D92 is in Module ArtGem.exe

```
:00434D87 cmp ecx, 00000ABF
:00434D8D je 00434DA3
:00434D8F pop edi
:00434D90 pop esi
:00434D91 pop ebp
BP*:00434D92 nop
:00434D93 nop
:00434D94 pop ebx
:00434D95 add esp, 00000024
:00434D98 ret
:00434D99 cmp edi, 0000000F
```

And try the patch.
Press run.

Enable Documented API Details Copy
Enable UnDocu
Enable Local
Stop Auto On A
Step Into "rep" Instruction Bypass
AutoStep Into F5 AutoStep Over F6 Step Into F7 Step Over F8 Pause Run
Goto Address Patch Code Terminate

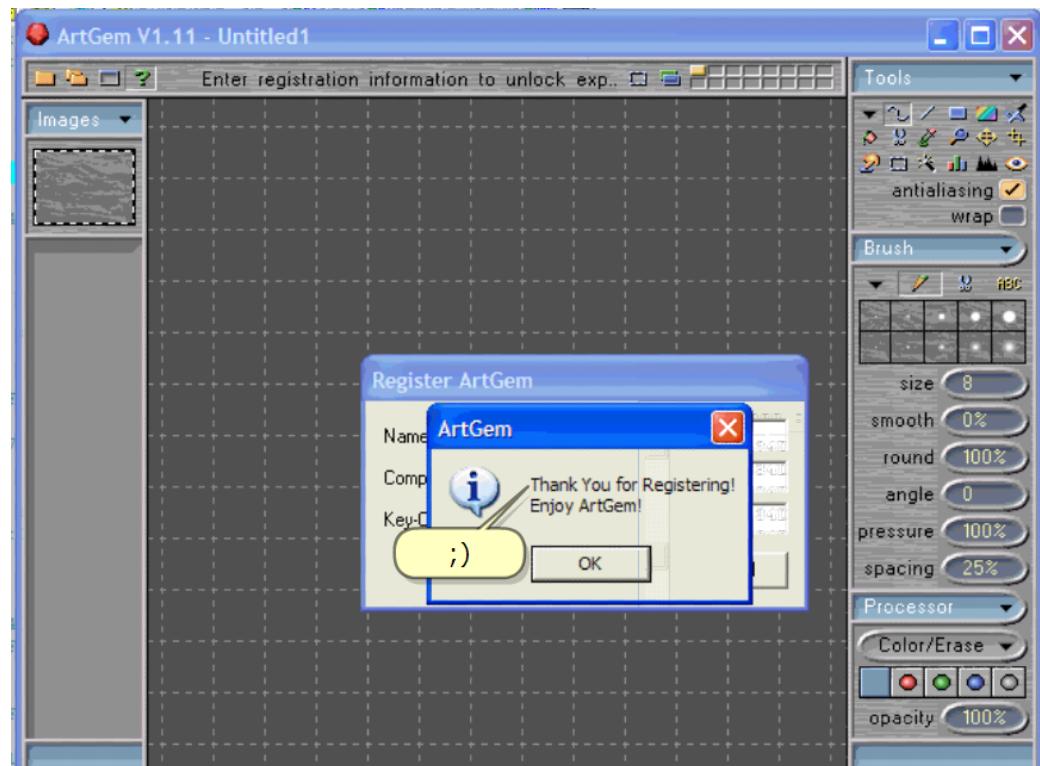
And try the patch.

Press F2

F2 를 눌러.

Patch 를 해.

F2 를 눌러.



;)

This seems okay.

Ok 가 나타났다.

5. Using LordPE as hexeditor and patching the exe

Compared to Olly where it is easy to make permanent changes in a file, it is impossible to do it with W32Dasm.

어느 file에서 영원히 변화할 곳을 만드는 게 쉬운지 Olly 와 비교해. 이것은 W32Dasm 으로 가능하다.

With W32Dasm, you can only patch memory, not on disk.

W32Dasm 와 함께, 너는 오직 memory patch 를 할 수 있다. Disk 는 아니다.

So, we will need another tool to do that.

그래서, 우리는 이 일을 하기 위해 다른 tool 이 필요할 것이다.

But let's first look in our debugger what and where changes are needed.

그러나 먼저 우리의 debugger 를 무엇이 그리고 어디에서 변화가 필요한지 봐라.

So, click in the debugger window

Debugger windows 를 click 해.

The screenshot shows a debugger window with assembly code. A yellow callout bubble points to the assembly line at address 00434D92, which contains the instruction 'nop'. Another callout bubble points to the same line with the text 'at VA 00434D92'. A third callout bubble points to the assembly line at address 00434D92 with the text 'which is at offset 00034D92'. A context menu is open over the assembly line at 00434D92, listing options like 'Enable Documented API Details', 'Copy', 'Goto Address', 'Patch Code', and 'Terminate'. The assembly code includes instructions like cmp edi, je 00434DA3, and jne 00434D8F.

And notice that we need to edit opcodes 33 C0

그리고 opcode 33 C0 의 edit 가 필요하다.

at VA 00434D92

VA 00434D92

Or see it here at VA 00434D92

또는 VA 00434D92 를 봐라.

Which is at offset 0034D92

Offset 0| 0034D92 다.

And NOP them !!!

그리고 그들을 NOP 하자.

번역 주) VA, offset 은 꼭 필요하다.

INFO :

In Part 03, I explained you that an RVA is an offset from the base address at which an executable was loaded in memory.

Part 03에서, 나는 너에게 설명했다. RVA 는 base address 부터 offset 이다. Memory 에서 실행 가능하게 load 됐다.

This is not necessarily the same as the offset within the file on disk because of the section alignment requirements.

이것은 File 안의 Offset 과 필연적으로 같지 않다. 왜냐하면 section alignment 가 있기 때문이다.

The PE header specifies the section alignment requirements for an executable image.

PE header 는 executable image 를 위해 section alignment 가 필요하다는 것을 명시한다.

A section has to be loaded at a memory address that is a multiple of the section alignment.

Section 은 다양한 section alignment 의 memory address 에 load 됐다.

The section alignment has to be a multiple of the page size.

Section alignment 는 다양한 page size 를 가졌다.

This is because different sections have different page attribute requirements (read, write, execute, ...).

이것은 왜냐하면 다른 section 들은 다른 page attribute 가 필요하다.(읽고, 쓰고, 실행, ...)

Hence, a page cannot span section boundaries.

그리하여, page 는 section 경계가 걸쳐있지 않다.

INFO :

Because the PE format always talks in terms of RVAs, it's difficult to find the location of the required information within a file.

왜냐하면 PE format 은 항상 RVA term 을 이야기 한다. 이것은 file 에서 요구된 위치 정보를 찾기 어렵다.

It's a bit complicated to calculate the address for the given RVA in a memory-mapped file.

이것은 memory-mapped files 에서 주어진 RVA address 를 계산하기 꽤 복잡하다.

You first need to find out the section in which the given RVA lies.

먼저 RVA 가 자리하고 있는 section 을 찾는 게 필요하다.

You can accomplish this by iterating through the section table.

Section table 을 반복하는 것으로 이것을 완성할 수 있다.

Each section header stores the starting RVA for the section and the size of the section.

각 section header 는 section 과 section 의 size 가 시작되는 RVA 에 저장된다.

A section is guaranteed to be contiguously loaded in memory.

section 은 memory 에 인접하게 load 되게끔 보장됐다.

Hence, the offset from the start of the section for a particular piece of data is bound to be the same whether the file is memory mapped or loaded by the operating system loader for execution.

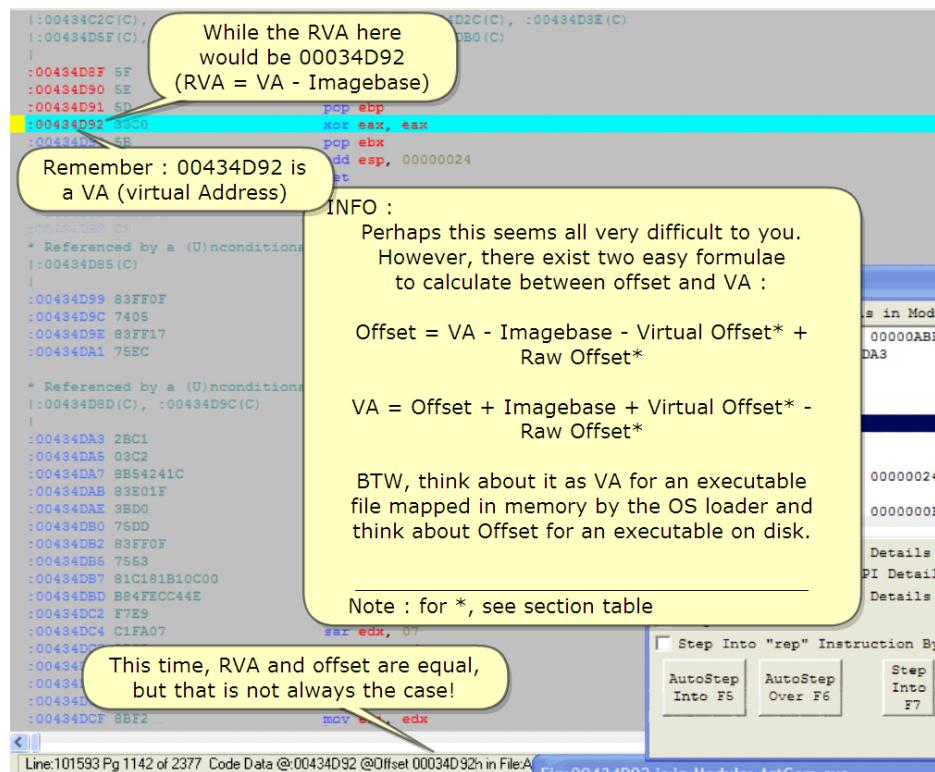
그리하여, data 의 특정 조각 section 의 시작부분부터 OS 에 의하여 실행되기 위하여 load 됐을 때 File 이 memory mapped 되거나 loaded 될 때 offset 은 같아질 가능성이 있다.

So, to find out the address in a memory-mapped file, you simply need to add this offset to the base address of the section in the memory-mapped file.

그리하여, memory-mapped file 에서 address 를 찾을 수 있다. 너는 간단히 memory-mapped file 에서 base address 의 section 에 offset 을 추가하기 위해 필요하다.

Now, this base address can be calculated from within the file offset of the section, which is also stored in the respective section header.

이제, 이 base address 는 file offset 의 section 으로부터 계산되어진다. 그곳은 또한 각각의 section header 에 저장된다.



INFO :

Perhaps this seems all very difficult to you.

아마 이것은 너에게 매우 어렵다.

However, there exist two easy formulae to calculate between offset and VA :

그러나, 이것은 2 가지 쉬운 공식이 offset 과 VA 를 계산하여 존재한다.

Offset = VA - Imagebase - Virtual Offset* + Raw Offset*

VA = Offset + Imagebase + Virtual Offset* + Raw Offset*

BTW, think about it as VA for an executable file mapped in memory by the OS loader and think about Offset for an executable on disk.

By the way, 운영체제 loader 에 의해 memory 에 맵핑된 실행 파일을 위한 VA 로 그것에 대해 생각하고 disk 에 있는 실행 파일에 대한 offset 을 생각해 봐.

Note : for *, see section table

번역 주) 여기 내용은 뭔가 다르다. <http://reversecore.com> 참조하라.

Remember : 00434D92 is a VA(Virtual Address)

기억 : 00434D92 는 VA 일 때(가상 주소)

While the RVA here would be 00034D92

RVA 가 0034D92 다.

(RVA = VA - Imagebase)

This time, RVA and offset are equal, but that is not always the case!

이 시간, RVA 와 offset 은 같다. 그러나 항상 같지는 않다.

And then at last comes the good news :

마지막으로 좋은 뉴스가 있다.

W32Dasm gives us both data :)

W32Dasm 은 2 개의 data 를 준다.

So, no worries : use the offset in a hexeditor ... but I'll show you a much easier solution :)

그리하여, 걱정하지 마 : hex editor 에서 offset 을 사용할 수 있다. 그러나 매우 쉬운 해결책을 보여줄 것이다.

First however, let's save this project (just in case) and terminate the debugging.

그러나 먼저, project 를 저장하고(이번 경우에는) debugging 을 종료하자.

The easy solution for all this is not to use a hexeditor: hexeditors use offsets.

쉬운 해결책은 Hex editor 를 사용하지 않는 것이다. : hex editor 의 offset 을 사용하자.

Instead, use LordPE. LordPE has come really interesting features for this. Let's start.

정말, LordPE 를 사용하자. LordPE 는 정말 재미있는 기능이 있다. 시작하자.

REMARK:

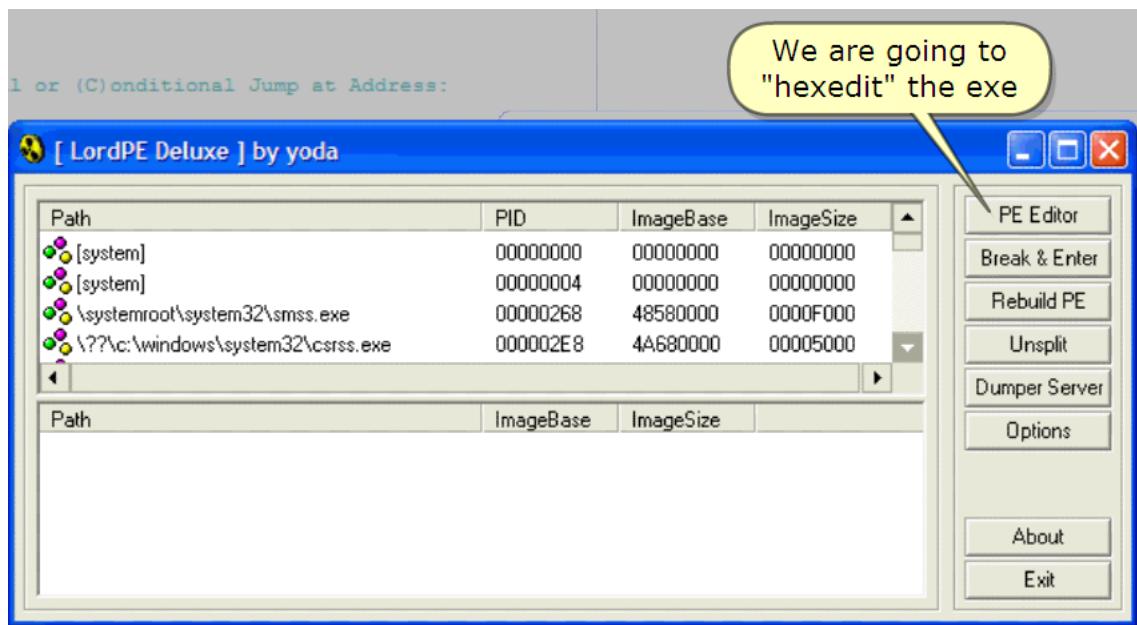
More complicated work may require a real hexeditor though.

So, I have NOT said that hexeditors are useless ... we will use them for other stuff.

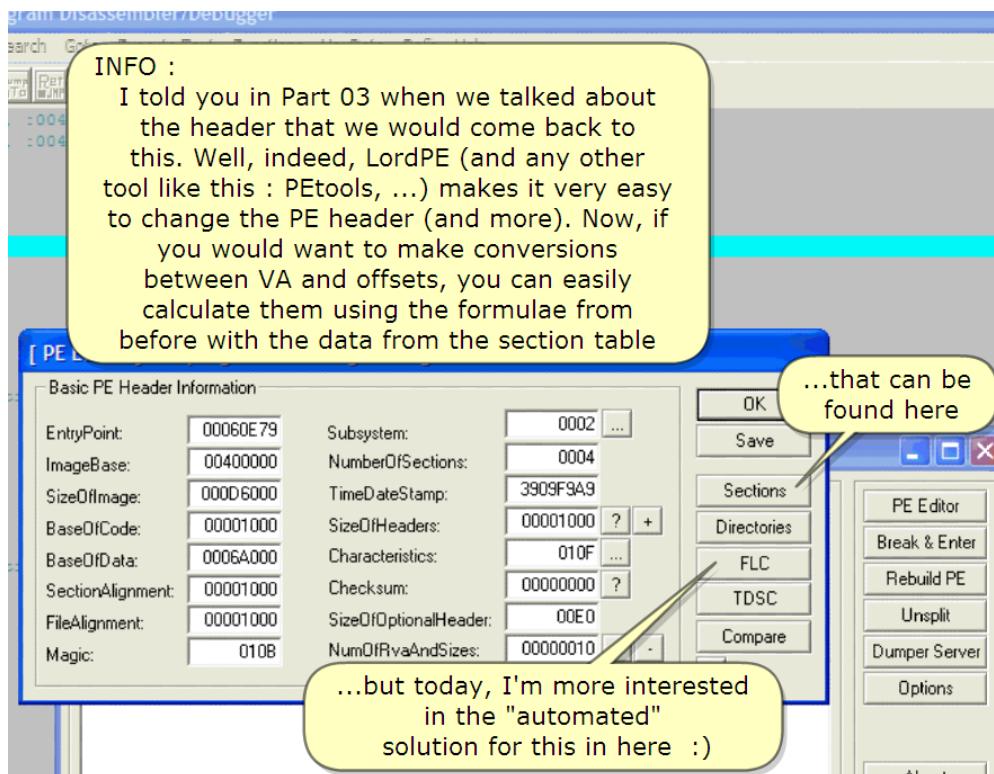
주목:

좀 복잡한 일은 진정한 hex editor 가 필요할 것이다.

그래서, 나는 hex editor 가 필용벗다고 말하지 않았다. 우리는 다른 구성에서 그것들을 사용할 것이다.



We are going to "hexedit" the exe



INFO :

I told you in Part 03 when we talked about the header that we would come back to this.
우리가 header 에 대해서 이야기 할 때 다시 돌아온다고 Part 03 에서 말했다.

Well, indeed, LordPE (and any other tool like this : Petools, ...) makes it very easy to change the PE header (and more).

좋아. LordPE(그리고 다른 tool 도 있다. :petools, ...)는 PE header 를 매우 쉽게 바꿀 수 있다.(그리고 더)

Now, if you would want to make conversions between VA and offsets, you can easily calculate them using the formulae from before with the data from the section table

이제, VA 와 offset 을 변환하기를 원한다면, 쉽게 수식을 사용하는 사용하여 section table 의 data 로부터 계산할 수 있습니다.

번역 주)FLC(File Location Calculator):File 위치 계산기

....that can be found here

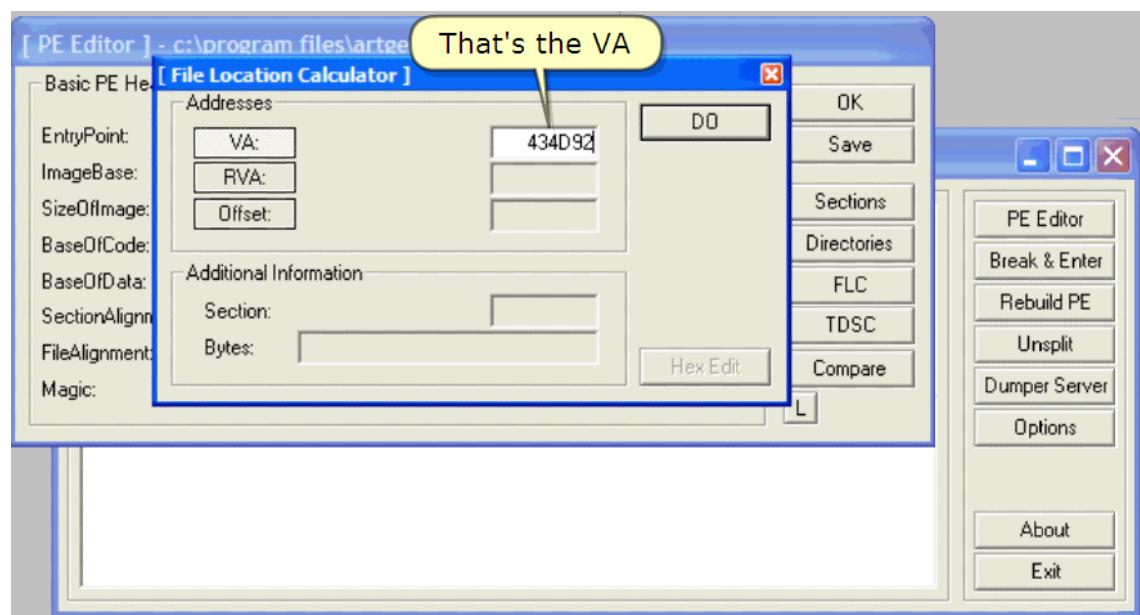
여기에서 찾을 수 있다.

...but today, I'm more interested in the "automated" solution for this in here :)

그러나 오늘, 나는 좀 더 흥미로운 "automated" 해결책을 보겠다.

And with the push of a button, LordPE will calculate all three : VA, RVA and offset !!!

PUSH button 을 누르고, LordPE 는 그곳을 계산할 수 있다 : VA, RVA and offset !!!

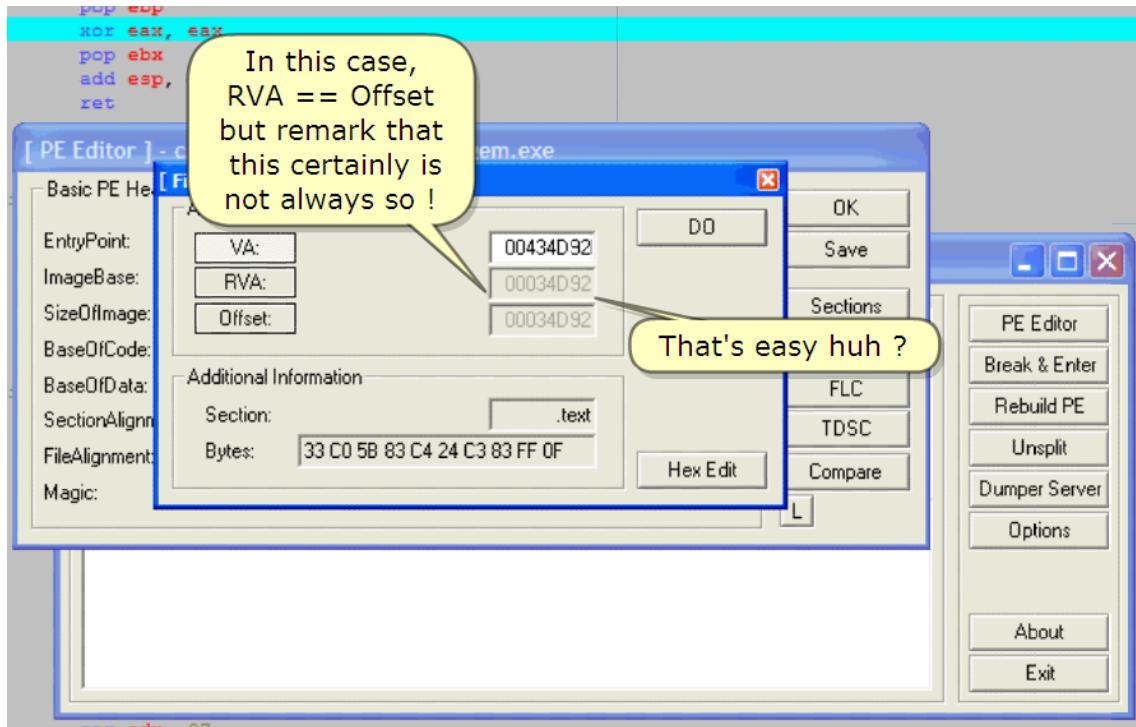


That's the VA

VA 다.

...that we always know from debugging of course

우리는 debugging 과정에서 항상 알고 있다.



That's easy huh ?

쉽지 않아?

In this case, RVA == Offset but remark that this certainly is not always so !

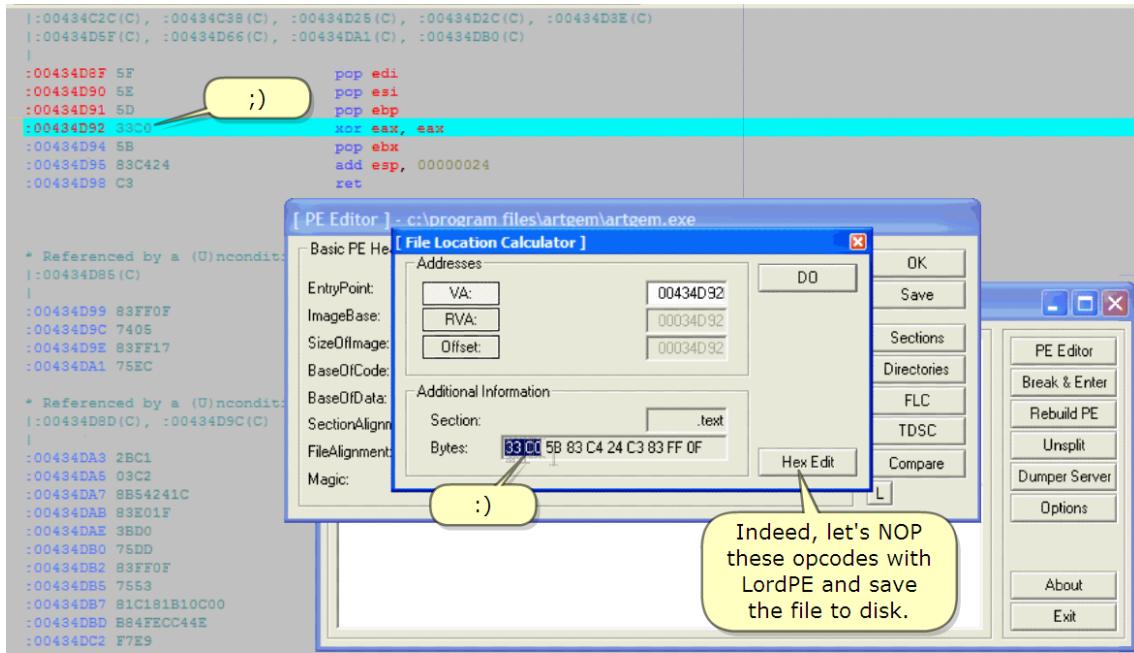
이번 경우, RVA == offset입니다. 그러나 항상 같지는 않다.

Okay, I hope this was clear enough and is understood now.

Ok, 이것은 충분히 명확하다. 그리고 이제 이해했다.

But look why I came to LordPE too: no need for another hexeditor, with LordPE, you can do this easily too !!!

LordPE로 온 이유를 봐라 : 다른 hex editor 가 필요하지 않다. LordPE만 있으면 너는 쉽게 바꿀 수 있어 !!!

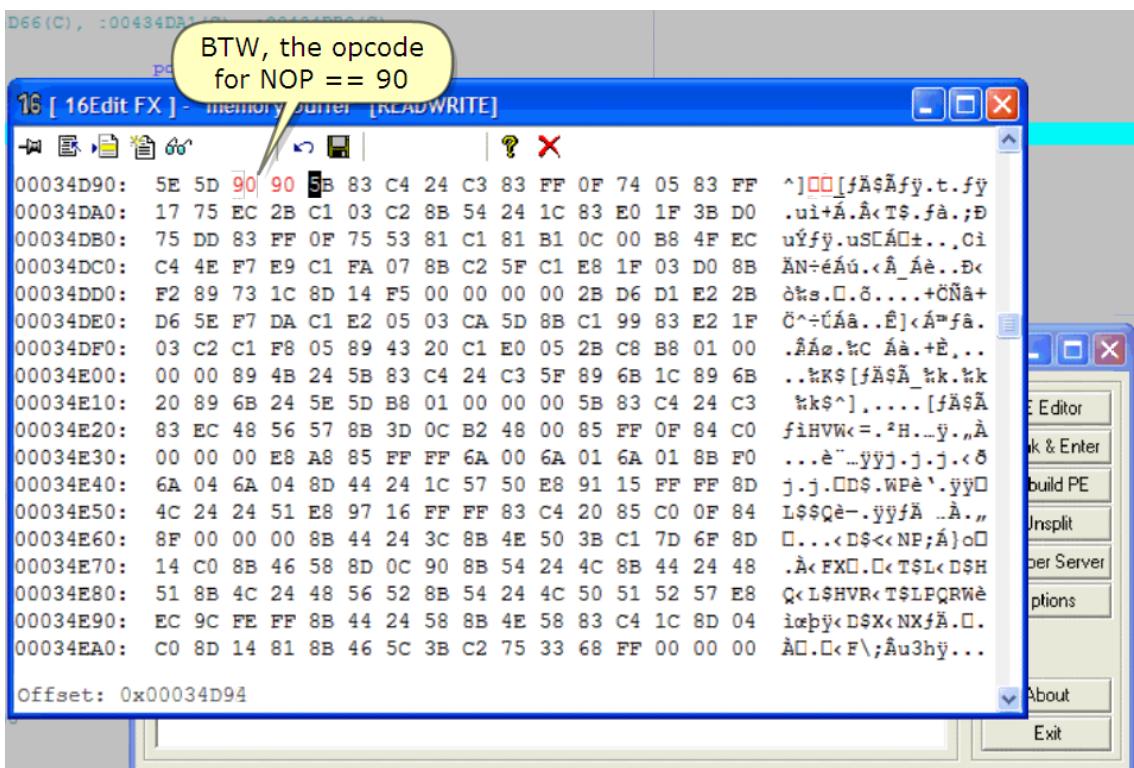


:)

;)

Indeed, let's NOP these opcodes with LordPE and save the file to disk.

정말, LordPE 와 함께 0| opcode 들을 NOP 하자. 그리고 disk 에 file 로 저장하자.



BTW, the opcode for NOP == 90

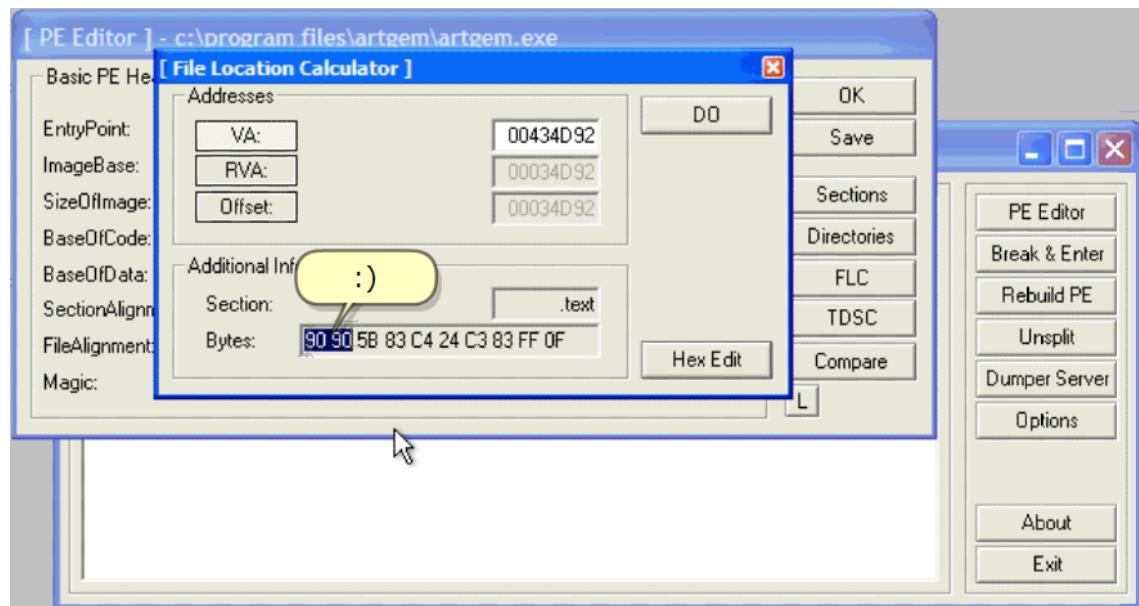
Opcde 는 NOP == 90

Save changes!

바뀐 것을 저장해!

Verify the changes

바뀐 것을 검증하자.



:)

BTW, always first make a backup file before reversing (also with Olly).

By the way, 항상 먼저 backup file 을 만들고 reversing 하자(Olly 또한)

LordPE always overwrites the file without making a backup

LordPE 는 항상 file 에 덮어쓰고 backup 을 만들지 않는다.

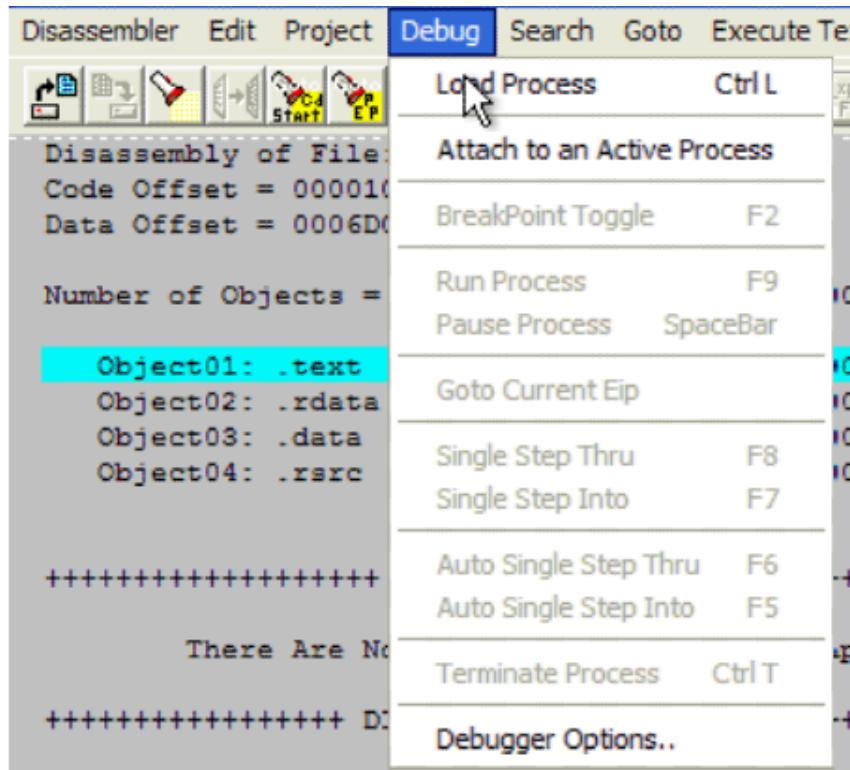
What rests now is ...

어떻게 달려있느냐?

6. Testing the patches

Notice that W32Dasm has indeed also saved the project!

W32Dasm 은 project 로 저장됐다.

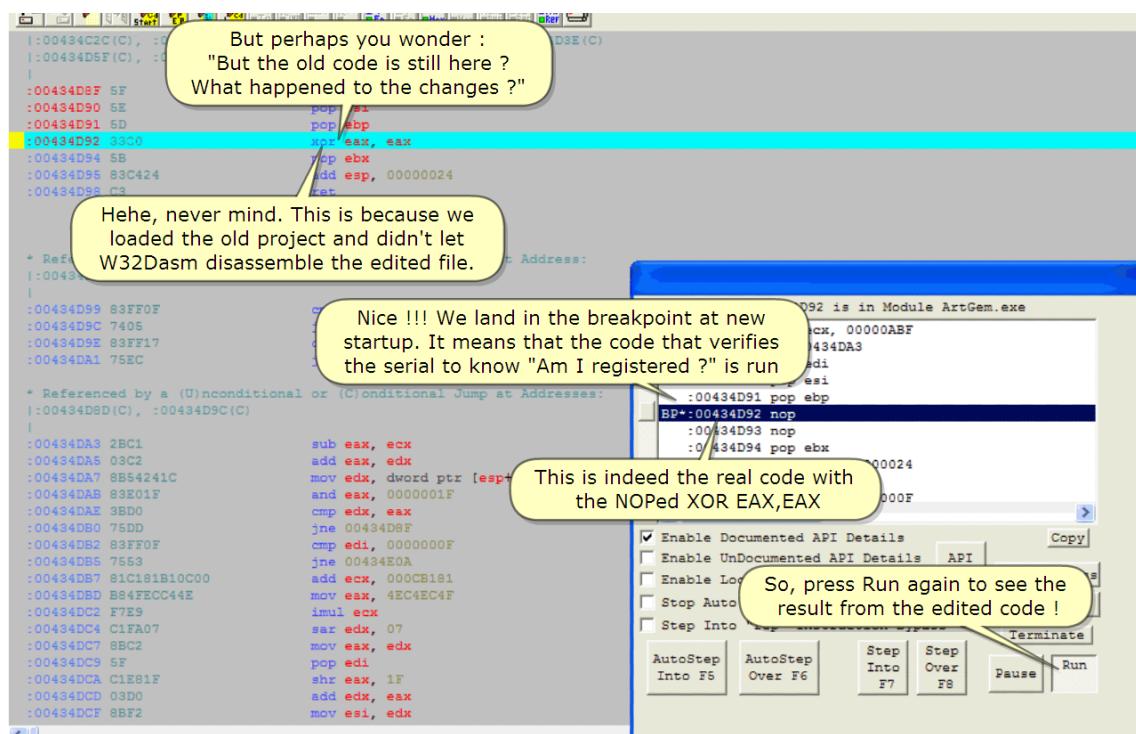


And we land again on the EP

우리는 EP에 다시 도착했다.

Let's run the program in the debugger to see if everything is fine ...

모든 것이 괜찮은지 보기 위해 Debugger에서 Program을 실행하자.



Nice!!! We land in the breakpoint at new startup.

좋아!!! 우리는 startup 의 breakpoint 에 도착했다.

It means that the code that verifies the serial to know "Am I registered ?" is run

이 뜻은 code 가 "등록됐는지?" serial 을 검증했는지 실행하자.

But perhaps you wonder :

아마 네가 원하는 것은?

"But the old code is still here ? What happened to the changes ?"

"Old code 는 아직 그대로인데? 무슨 변화가 있다는 거야?"

Hehe, never mind.

헤헤, 걱정 마.

This is because we loaded the old project and didn't let W32Dasm disassemble the edited file.

이것은 왜냐하면 우리가 old project 를 load 했기 때문이야. 그리고 W32Dasm 에게 수정된 file 을 disassemble 하라고 시키지 않았다.

This is indeed the real code with the NOPed XOR EAX, EAX

이것이 XOR EAX, EAX 가 NOP 된 real code 다.

So, press Run again to see the result from the edited code!

그래서, 수정된 code 에서 결과를 보기 위해 Run 을 다시 한 번 눌러.



It seems everything is fine !!!

모든 것이 괜찮은 것을 봐 !!!

The register button has gone !

등록 button 이 사라졌다!

In this part 8, the primary goal was to study the use of another debugger and patching a program's registration scheme using LordPE as hexeditor.

이번 part 8 에서는, 중요한 목표는 다른 debugger 를 공부하는 것이었다. 그리고 program's registration scheme 을 hex editor 인 LordPE 를 사용하여 patch 하는 것이었다.

RVA, VA and offset were handled too.

RVA, VA and offset 은 다뤄졌다.

I hope you understood everything fine and I also hope someone somewhere learned something from this. See me back in part 09 ;)

네가 모든 것을 좋게 이해했기를 희망한다. 또한 누구든지 어디서든지 이것에서 무언가를 배웠으면 좋겠다. Part 09 에서 보자.

The other parts are available at

다른 parts 는 사용 가능하다.

<http://tinyurl.com/27dzdn> (tuts4you)

<http://tinyurl.com/r89zq> (SnD Filez)

<http://tinyurl.com/l6srv> (fixdown)

Regards to all and especially to you for taking the time to look at this tutorial.

Lena151 (2006, updated 2007)

모두에게 안부를 전하고 특별히 이 tutorial 에 시간을 투자해준 너에게 감사한다.