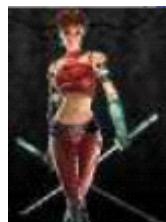


# 01.Olly + assembler + patching a basic reverseme

2011년 12월 1일 목요일

오후 11:00

## Lena Reversing



# 01.Olly + assembler + patching a basic reverseme

번역자 : re4lf10w / re4lf10w@gmail.com

Last updated: 2012.02.22

The creator of this movie or the ISP(s) hosting this movie or any content in this movie take no responsibility for the way you use the information provided in this movie.

Movie 만든 사람이나 ISP에 이 movie를 hosting하거나 이 movie 속의 어떤 content에서도 제공된 정보를 사용하는 방식에 대해 책임지지 않습니다.

These file and anything else on this site and in this movie are here for private purposes only and should not be downloaded or viewed whatsoever!

이 site에서 file들과 모든 것들은 오직 개인적인 목적으로 download 하거나 볼 수 없습니다.

If you are affiliated with any government, or Anti-Piracy group or any other related group or were formally a worker of one you cannot enter this web site nor see this movie,

당신이 어떤 정부 또는 불법 복제 방지 그룹 또는 기타 관련 단체와 제휴하거나 공식적으로 하나의 노동자로 일할 경우 이 web site에 접속할 수 없습니다. 또한 이 movie를 볼 수 없습니다.

cannot access any of its files and you cannot view any of the HTML files.

어떤 file에도 접속할 수 없습니다. 어떤 HTML file도 볼 수 없습니다.

All the objects on this site and in this movie are private property and are not meant for viewing or any other purposes other than bandwidth space.

이 movie 안의 Site에 있는 모든 object와 개인적인 재산이다. 다른 목적이나 대역폭을 의미하지 않는다.

Do not see this movie whatsoever!

이 movie를 참조하지 마십시오.

If you see this movie you are not agreeing to these terms and you are violating code 431.322.12 of the Internet Privacy Act signed by Bill Clinton in 1995 and that means that you can't threaten our ISP(s) or any person(s) or company storing these file or these movies, cannot prosecute any person(s) affiliated with this page and movie which includes family, friends or individuals who run or enter this site or see this movie.

이 movie를 볼 경우 이 약관에 동의하지 않습니다. 그리고 당신은 1995년 빌 클린턴의 서명이 인터넷 개인 정보 보호법의 코드 431.322.12를 위반하고 당신이 우리의 ISP들 또는 사람들 위협할 수 없다는 것을 의미합니다. 또는 이러한 파일이나 이러한 movie를 실행하고 저장하는 회사, 가족, 친구거나 사이트에 들어오거나 영화를 볼 개인을 포함하거나 페이지 및 영화와 제휴한 다른 사람들을 기소할 수 없습니다.

번역 주) 이 부분 번역이 제일 어려웠습니다. 이해도 잘 안되고.

Hello everybody.

안녕 모두들.

Welcome to this Part1 in my series about reversing for newbies/beginners.

나의 reversing 초보자를 위한 series Part 1에 온 것을 환영해.

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

이 saga의 대상은 reversing에서 Programming 경험조차 없는 완벽한 초보자다.

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 skip할 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 click해.

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

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

Click here as soon as you finished reading(on each screen !)

네가 읽기가 끝났을 때 이 곳을 클릭해. (이번 screen에서)

## 1. Abstract

Perhaps you wonder what reversing is. Well, reversing (reverse engineering) is the changing of the code of a program.

아마 네가 reversing 하기를 원한다면. 좋아, reversing은 program code를 바꾸는 것이다.

This is done to change the looks of a program, to clean from virii, to enhance the program with other functions, to make it compatible with other systems ...

이것은 program을 Virii로부터 깨끗하기 위해, program 성능을 다른 function들과 같이 증가하기 위해 다른 system과 호환이 되는 좋게 바꾸기 위해 끝냈다.

But in general, to get a better insight in the working of a program. My goal with this series of tutorials is to show you some techniques in this matter.

그러나 일반적으로, program에서 좋은 통찰력을 얻기 위해. 이 series의 목표는 이 문제들에 대해서 약간의 기술을 보여주는 것이다.

Starting on an easy target but making this more difficult step by step. No computer knowledge is required except working in windows.

시작할 때는 매우 쉬운 목표물이다. 그러나 단계를 밟아 조금씩 어려워진다. Windows에서 작업하는 지식 빼고 Computer 지식은 필요하지 않다.

The end will be that you will be able to do some basic reversing (I hope).

마지막에 너는 약간의 기본적인 reversing을 할 수 있을 것이다.(내 희망)

Hopefully, you will exploit your new gained knowledge in a positive way. I am not responsible for what you do with your newly acquired "wisdom".

희망하건대, 너는 새로 얻은 지식을 긍정적으로 이용해. 나는 네가 새로 얻은 "지식으로" 무엇을 하든 책임지지 않는다.

Therefore, I will only target ReverseMe's, (very) old software or applications that are no longer updated.

그러므로, 나는 오직 ReverseMe's 목표물은 오래된 software나 update가 오래된 application이다.

The tools we will be using will be freely available (freeware) or no longer supported.

Tools은 자유롭게 사용할 수 있거나 더 이상 지원되지 않습니다.

Please, purchase the software you use after trial period. Authors need money to live and deserve it for their hard work !!!

제발, trial 기간이 끝나면 software를 구매 해. 제작자들은 살아가기 위해서 돈이 필요해. 그리고 그들의 강도 높은 노동의 대가를 받을 만하다.

All this being said for this and also for the next parts in this series, let's get this started.

이것을 위해 처음에 말한다. 이 series의 다음 part를 위해. 이제 시작하자.

In this first part, I will have to give you some theoretical insights.

이 첫번째 part에서 나는 너에게 이론적인 통찰력을 줄 것이다.

I believe that practice is better than dry theory. So, even in this first tutorial, we are going to reverse an easy, basic ReverseMe.

나는 믿는다. 이 연습이 재미없는 이론보다 낫다. 그래서 이 첫번째 tutorial에서 조차 우리는 reverse를 쉽게 하기 위해 기본적인 ReverseMe를 할 것이다.

Part 01 till 03 may be the most difficult for starters because some basic general knowledge about reversing and assembler language is handled.

Part 01에서 03까지는 초보자에게는 꽤 어려울 것이다. 왜냐하면 reversing에 대한 약간의 기본적인 지식들과 assembler language를 조종한다.

## 2. Tools and Targets

Throughout most of these tutorials, you will use Ollydebug and ... your brain.

The first can be obtained for free at

<http://www.ollydbg.de>

대부분 tutorial들에서, Ollydebug를 사용하게 될 것이다. 그리고 너의 두뇌도.

첫번째 무료로 <http://www.ollydbg.de> 를 얻을 수 있다.

...the second is your responsibility ;)

두번째는 너의 책임감이다.

The target(ReverseMe) is also included in this package. I "cut" a rather big amount of code from the reverseme to make the code very clear about left the open space not to make it too obvious.

목표물은(ReverseMe)은 이 package에 포함 됐다. 나는 reverseme로부터 꽤 많은 양의 코드를 명확하게 만들기 위해 잘랐다.

INFO:

Whilst at the site from Ollydebug, take a look around, especially at the "shortcut key's page". Knowing these is a big help in faster working. I also included that page.

Ollydebug를 얻은 site에서 주변을 둘러봐. 특별한 "단축키" 페이지가 보일 것이다. 우리가 일을 빨리 하기 위해 그것은 아주 큰 도움이 된다. 나 또한 그 페이지를 첨부했다.

During most of the Parts in this series, I will use ARTeam-Ollydbg.ini file for my OllyDbg. This file holds the options for Ollydbg. It will make that mine will look a little different from the original. However, I have included the ARTeam-Ollydbg-ini file in this package and have already renamed it into its original filename (ollydbg.ini). Overwrite the original one in ollydbg's directory with the one that I included if you want to see the same as me. 이 series의 대부분의 Parts에서 나는 ARTeam-Ollydbg.ini를 사용할 것이다. File은 옵션을 저

장하고 있다. 그것은 original과 약간 다르다. 그러나, 나는 ARTeam-Ollydbg.ini 파일을 첨부 했다. 그리고 이미 이름도 original과 같이 바꿔놨다.(ollydbg.ini) Ollydbg의 디렉토리에 있는 original.ini에 덮어 씌우면 된다.

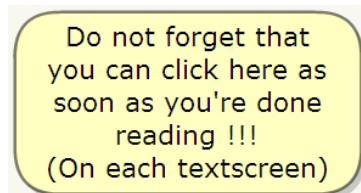
### 3. Ollydbg and some general info

Ollydebug, hereafter often called "Olly", is a Ring3 debugger. It means that Olly works at windows application level, but still can control other applications. In human language: with this magnificent tool(thanks Oleh Yushuk), we can find the "bugs" in a program. Not only find but also correct the program.

Ollydebug는 자주 "Olly"로 불리고 Ring3 debugger 다. 이것은 Olly가 windows application level로 일할 수 있다는 것을 뜻한다. 다른 프로그램들도 컨트롤 할 수 있다. 사람이 이해하기 쉽게 말하자면: 유용한 툴과 함께 할 수 있다.(Oleh Yushuk님 고마워요) 우리는 프로그램에서 "bugs"를 찾을 수 있다. 찾는 것뿐만 아니라 프로그램이 정확한지도 볼 수 있다.

You probably already noticed the "weird" background? That is because I have already opened the ReverseMe in Olly. Let's have a look at it.

내 화면 뒤에 이상한 게 있지? 맞아. 내가 이미 ReverseMe를 Olly에 열어놨기 때문이야. 어떻게 생긴 것인지 보자.



Do not forget that you can click here as soon as you're done reading!!!

(On each textscreen)

모든 것을 다 읽었을 때 이 곳을 클릭하는 것을 잊지 말아줘.

So, have you downloaded Olly yet?

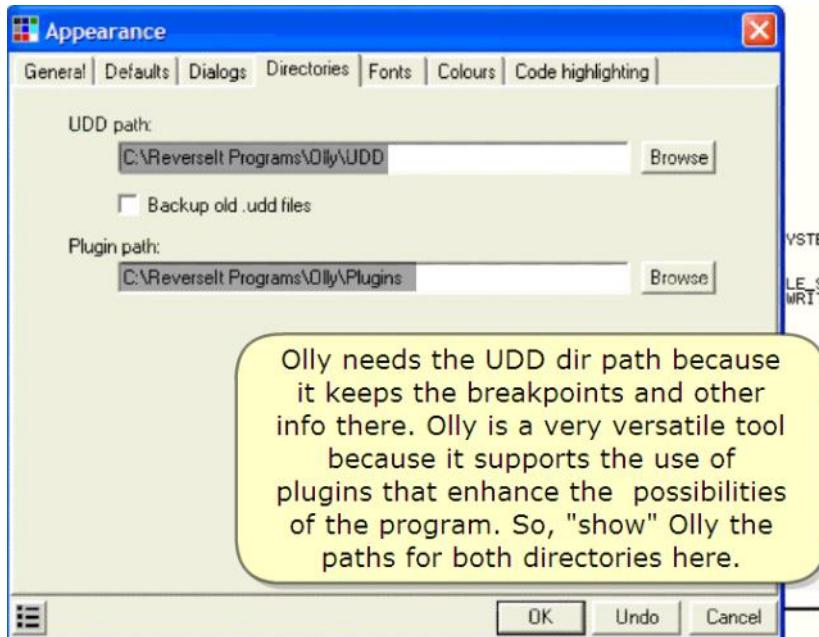
Ok, Olly doesn't need installing. Just unzip to a directory of your choice....et voila...start Olly.

Olly를 다운로드 했어?

Ok, Olly는 install이 필요하지 않아. Directory에 압축 풀고 Olly를 시작해.

At first startup, Olly will ask for the UDD and Plugins directory. Solve this like this....

먼저, Olly는 UDD와 Plugins 디렉토리를 물어. 이렇게 해결하면 돼.



Olly needs the UDD dir path because it keeps the breakpoints and other info there. Olly is a very versatile tool because it supports the use of plugins that enhance the possibilities of the program. So, "show" Olly the paths for both directories here.

Olly에게 UDD 경로가 필요해. 왜냐하면 그것은 breakpoints와 다른 정보들을 저장하고 있으니까. Olly는 다재 다능한 tool이다. 왜냐하면 그것은 다양한 plugin을 지원하니까. 그것은 프로그램의 가능성을 증가시킨다. 그래서 Olly의 경로가 2개의 디렉토리에 있다는 것을 보여주는 거야.

After changing the path, you need to restart Olly and  
Open the ReverseMe

경로를 바꾼 후에는 Olly를 재시작하고 ReverseMe를 열어.

to get the same view as me. It may be that you need to rearrange the windows. BTW, all the buttons you see above will be explained during the tutorials, but you may want to play around a bit with them to explore a little.

나와 같은 화면을 볼 거야. Windows 재정렬 할거야. 걱정하지 마. 이 tutorial에서 모든 버튼을 설명할 거야. Olly 주변이 어떻게 생겼는지 살펴 봐.

**INFO :** always leave your mouse pointer here and click here if you want to continue to the next screen !!!

클릭하면 넘어가니까 마우스 여기에 갖다 놓지마. 다음 screen으로 넘기기 위해서 이곳을 클릭해.

Let's look a little closer at the windows that are visible.

좋아. Windows의 보여지는 부분을 조금 자세히 살펴보자.

**INFO:**

If you are completely new to all this and in case you have had no previous assembler education at all, then all this may be a bit overwhelming at first sight. All will eventually become clear though. Just let it soak in ...

네가 완벽히 모든 것에 처음이고 assembler 교육이 이전에 없더라도. 걱정하지 마. 처음에는

적용하기 힘들 거야. 결국 모든 것이 완벽해 질 거야. 리버싱의 마법에 빠져보자!!

The main window is the CPU window. We see ...

```
00401000 E8 00          PUSH 0
00401002 E8 64020000    CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 H3 77214000    MOU DWORD PTR DS:[402177],EAX
0040100C C705 97214000 03 MOU DWORD PTR DS:[402197],4003
00401016 C705 A3214000 00 MOU DWORD PTR DS:[402198],reverseM.004011
00401020 C705 9E214000 00 MOU DWORD PTR DS:[40219F],0
0040102A C705 A3214000 00 MOU DWORD PTR DS:[4021A3],0
00401034 A1 77214000    MOU EA, DWORD PTR DS:[402177]
00401039 A3 97214000    MOU DWORD PTR DS:[4021A7],EAX
0040103E 6A 04          PUSH 4
00401040 50             PUSH EAX
00401041 E8 3F030000    CALL <JMP.&USER32.LoadIconA>
00401046 A3 AB214000    MOU DWORD PTR DS:[4021AB],EAX
00401048 68 007F0000    PUSH 0
00401050 6A 00          PUSH 0
00401052 E8 C8020000    CALL <JMP.&USER32.LoadCursorA>
00401057 A3 AF214000    MOU DWORD PTR DS:[4021AF],EAX
0040105C 6A 00          PUSH 0
0040105E 68 6F214000    PUSH 0
00401063 6A 03          PUSH 3
00401065 6A 00          PUSH 0
00401067 68 00000000    PUSH 0
00401069 68 79204000    PUSH 0
00401073 E8 0B020000    CALL <JMP.&KERNEL32.CreateFileA>
00401078 83F8 FF        CMP EAX,-1
0040107D 75 1D          JNE SHORT reverseM.0040109A
0040107F 6A 00          PUSH 0
00401080 68 00204000    PUSH reverseM.00402000
00401084 68 17204000    PUSH reverseM.00402017
00401089 6A 00          PUSH 0
0040108B 68 D7020000    CALL <JMP.&USER32.MessageBoxA>
00401090 E8 24020000    CALL <JMP.&KERNEL32.ExitProcess>
00401094 >   E8 83010000  JMP reverseM.0040121D
00401099 6A 00          PUSH 0
0040109C 68 73214000    PUSH reverseM.00402173
004010A1 6A 46          PUSH 46
004010A3 68 18214000    PUSH reverseM.0040211A
004010A8 50             PUSH EAX
004010A9 E8 2F020000    CALL <JMP.&KERNEL32.ReadFile>
004010AE 85C0            TEST EAX,EAX
004010B0 7F 2A          INT 3
```

00401000 E8 00 PUSH 0
00401002 E8 64020000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 H3 77214000 MOU DWORD PTR DS:[402177],EAX
0040100C C705 97214000 03 MOU DWORD PTR DS:[402197],4003
00401016 C705 A3214000 00 MOU DWORD PTR DS:[402198],reverseM.004011
00401020 C705 9E214000 00 MOU DWORD PTR DS:[40219F],0
0040102A C705 A3214000 00 MOU DWORD PTR DS:[4021A3],0
00401034 A1 77214000 MOU EA, DWORD PTR DS:[402177]
00401039 A3 97214000 MOU DWORD PTR DS:[4021A7],EAX
0040103E 6A 04 PUSH 4
00401040 50 PUSH EAX
00401041 E8 3F030000 CALL <JMP.&USER32.LoadIconA>
00401046 A3 AB214000 MOU DWORD PTR DS:[4021AB],EAX
00401048 68 007F0000 PUSH 0
00401050 6A 00 PUSH 0
00401052 E8 C8020000 CALL <JMP.&USER32.LoadCursorA>
00401057 A3 AF214000 MOU DWORD PTR DS:[4021AF],EAX
0040105C 6A 00 PUSH 0
0040105E 68 6F214000 PUSH 0
00401063 6A 03 PUSH 3
00401065 6A 00 PUSH 0
00401067 68 00000000 PUSH 0
00401069 68 79204000 PUSH 0
00401073 E8 0B020000 CALL <JMP.&KERNEL32.CreateFileA>
00401078 83F8 FF CMP EAX,-1
0040107D 75 1D JNE SHORT reverseM.0040109A
0040107F 6A 00 PUSH 0
00401080 68 00204000 PUSH reverseM.00402000
00401084 68 17204000 PUSH reverseM.00402017
00401089 6A 00 PUSH 0
0040108B 68 D7020000 CALL <JMP.&USER32.MessageBoxA>
00401090 E8 24020000 CALL <JMP.&KERNEL32.ExitProcess>
00401094 > E8 83010000 JMP reverseM.0040121D
00401099 6A 00 PUSH 0
0040109C 68 73214000 PUSH reverseM.00402173
004010A1 6A 46 PUSH 46
004010A3 68 18214000 PUSH reverseM.0040211A
004010A8 50 PUSH EAX
004010A9 E8 2F020000 CALL <JMP.&KERNEL32.ReadFile>
004010AE 85C0 TEST EAX,EAX
004010B0 7F 2A INT 3

The main windows is the CPU windows. We see...

우리가 보는 Main windows는 CPU Window야.

The VA's. (Virtual addresses) When starting a program, the windows loader loads the program at a certain memory location. (Not always the same, see later)

```
00401000 E8 00          PUSH 0
00401002 E8 64020000    CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 H3 77214000    MOU DWORD PTR DS:[402177],EAX
0040100C C705 97214000 03 MOU DWORD PTR DS:[402197],4003
00401016 C705 A3214000 00 MOU DWORD PTR DS:[402198],reverseM.004011
00401020 C705 9E214000 00 MOU DWORD PTR DS:[40219F],0
0040102A C705 A3214000 00 MOU DWORD PTR DS:[4021A3],0
00401034 A1 77214000    MOU EA, DWORD PTR DS:[402177]
00401039 A3 97214000    MOU DWORD PTR DS:[4021A7],EAX
0040103E 6A 04          PUSH 4
00401040 50             PUSH EAX
00401041 E8 3F030000    CALL <JMP.&USER32.LoadIconA>
00401046 A3 AB214000    MOU DWORD PTR DS:[4021AB],EAX
00401048 68 007F0000    PUSH 0
00401050 6A 00          PUSH 0
00401052 E8 C8020000    CALL <JMP.&USER32.LoadCursorA>
00401057 A3 AF214000    MOU DWORD PTR DS:[4021AF],EAX
0040105C 6A 00          PUSH 0
0040105E 68 6F214000    PUSH 0
00401063 6A 03          PUSH 3
00401065 6A 00          PUSH 0
00401067 68 00000000    PUSH 0
00401069 68 79204000    PUSH 0
00401073 E8 0B020000    CALL <JMP.&KERNEL32.CreateFileA>
00401078 83F8 FF        CMP EAX,-1
0040107D 75 1D          JNE SHORT reverseM.0040109A
0040107F 6A 00          PUSH 0
00401080 68 00204000    PUSH reverseM.00402000
00401084 68 17204000    PUSH reverseM.00402017
00401089 6A 00          PUSH 0
0040108B 68 D7020000    CALL <JMP.&USER32.MessageBoxA>
00401090 E8 24020000    CALL <JMP.&KERNEL32.ExitProcess>
00401094 >   E8 83010000  JMP reverseM.0040121D
00401099 6A 00          PUSH 0
0040109C 68 73214000    PUSH reverseM.00402173
004010A1 6A 46          PUSH 46
004010A3 68 18214000    PUSH reverseM.0040211A
004010A8 50             PUSH EAX
004010A9 E8 2F020000    CALL <JMP.&KERNEL32.ReadFile>
004010AE 85C0            TEST EAX,EAX
004010B0 7F 2A          INT 3
```

00401000 E8 00 PUSH 0
00401002 E8 64020000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 H3 77214000 MOU DWORD PTR DS:[402177],EAX
0040100C C705 97214000 03 MOU DWORD PTR DS:[402197],4003
00401016 C705 A3214000 00 MOU DWORD PTR DS:[402198],reverseM.004011
00401020 C705 9E214000 00 MOU DWORD PTR DS:[40219F],0
0040102A C705 A3214000 00 MOU DWORD PTR DS:[4021A3],0
00401034 A1 77214000 MOU EA, DWORD PTR DS:[402177]
00401039 A3 97214000 MOU DWORD PTR DS:[4021A7],EAX
0040103E 6A 04 PUSH 4
00401040 50 PUSH EAX
00401041 E8 3F030000 CALL <JMP.&USER32.LoadIconA>
00401046 A3 AB214000 MOU DWORD PTR DS:[4021AB],EAX
00401048 68 007F0000 PUSH 0
00401050 6A 00 PUSH 0
00401052 E8 C8020000 CALL <JMP.&USER32.LoadCursorA>
00401057 A3 AF214000 MOU DWORD PTR DS:[4021AF],EAX
0040105C 6A 00 PUSH 0
0040105E 68 6F214000 PUSH 0
00401063 6A 03 PUSH 3
00401065 6A 00 PUSH 0
00401067 68 00000000 PUSH 0
00401069 68 79204000 PUSH 0
00401073 E8 0B020000 CALL <JMP.&KERNEL32.CreateFileA>
00401078 83F8 FF CMP EAX,-1
0040107D 75 1D JNE SHORT reverseM.0040109A
0040107F 6A 00 PUSH 0
00401080 68 00204000 PUSH reverseM.00402000
00401084 68 17204000 PUSH reverseM.00402017
00401089 6A 00 PUSH 0
0040108B 68 D7020000 CALL <JMP.&USER32.MessageBoxA>
00401090 E8 24020000 CALL <JMP.&KERNEL32.ExitProcess>
00401094 > E8 83010000 JMP reverseM.0040121D
00401099 6A 00 PUSH 0
0040109C 68 73214000 PUSH reverseM.00402173
004010A1 6A 46 PUSH 46
004010A3 68 18214000 PUSH reverseM.0040211A
004010A8 50 PUSH EAX
004010A9 E8 2F020000 CALL <JMP.&KERNEL32.ReadFile>
004010AE 85C0 TEST EAX,EAX
004010B0 7F 2A INT 3

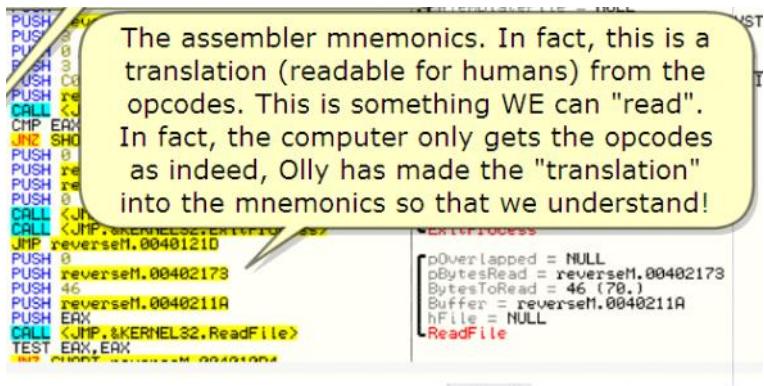
The VA's. (Virtual addresses) When starting a program, the windows loader loads the program at a certain memory location. (Not always the same, see later)

가상 메모리이다. 프로그램 시작할 때, 윈도우 로더는 프로그램을 정확한 메모리 장소에 로드 한다. (항상 같은 장소는 아니다.)

The opcodes. In fact, these are the codes that the computer reads and can understand.

명령코드. 사실, 이것들은 코드다. 이것은 컴퓨터가 읽고 이해할 수 있다.

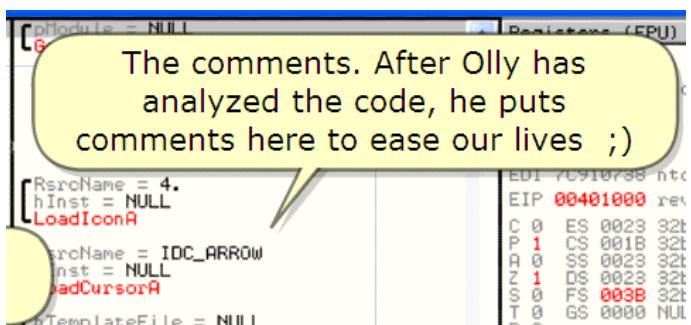
번역 주)opcode: Operation code. 즉, 명령 코드.



The assembler mnemonics. In fact, this is a translation (readable for humans) from the opcodes. This is something WE can "read". In fact, the computer only gets the opcodes as indeed, Olly has made the "translation" into the mnemonics so that we understand!

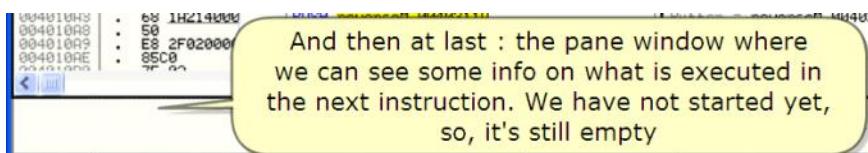
어셈블러 상징이다. 사실, 이것은 명령코드가 번역됐다.(인간이 읽기 쉽게) 이것은 우리가 읽을 수 있다. 사실, 컴퓨터는 바보라서 오직 명령코드만 이해해. Olly는 "번역"을 상징어로 만들었어. 그래서 우리는 그것(명령코드)을 이해할 수 있다.

번역 주)Olly는 opcode(컴퓨터가 이해하는)을 인간이 읽기 쉽게 바꿔주는 것. 예를 들자면 6A 00(opcode)을 우리가 이해하기 쉽게 꼼 PUSH C으로 바꿔주는 것.



The comments. After Olly has analyzed the code, he puts comments here to ease our lives ;)

이 comment는 Olly가 해석한 code다. Olly는 우리가 쉽게 사용할 수 있게 comment를 넣는다.



And then at last : the pane window where we can see some info on what is executed in the next instruction. We have not started yet, so, it's still empty

다음 명령어가 실행되는 동안 Pane windows에서 약간의 정보를 볼 수 있다. 우리는 아직 시작하지 않았기 때문에 아직 비어있다.

#### INFO:

Some essential info regarding sizes first. Make sure you do understand these before continuing

Code를 이해하기 위한 size의 필수적인 정보다. 진행하기 전에 이해해야 한다.

2 digits == 1 byte

ddress	Hex dump
04020000	20 4B 65 79 20 46 69 6C 6E
04020040	CC

2digits == 1byte

1 byte == 8 bits

20 46 69 60 65 20 52 65 76 65

1byte == 8bits

LOC	OP	WT	INSTR
040	50		PUSH EAX
041	E8 3F030000		CALL JKMP.
046	A3 BB214000		MOV DIWORD
04B	68 00 F00000		PUSH 7F00
050	C0 00		DUCH C

1 opcode == 1 byte

1 opcode == 1 byte

2 bytes == 1 word

2bytes == 1word

4 bytes == 1 dword

4bytes == 1dword

번역 줄) *dword = Double word*

2 words == 1

2words == 1dword

번역 주) *dword == double word*

Let's continue looking around in Olly. First, I advise you to download the win32.hlp file.

This will be a good help understanding API's and their working in a program.

계속 Olly를 살펴보자. 첫번째로 Win32.hlp file을 다운 받아.

이것은 프로그램에서 작동하는 API를 이해하는데 좋아.

BTW, API's(Application Programming Interface) are the way in which a program interacts with the kernel.

By the way, API는 kernel과 program이 대화하는 방법이다.

Install this helpfile in a dir on your HD and show Olly the path towards it. Olly is then able (rightclick) to easily show you help on API's.

Helpfile을 directory에 install 해. Olly path를 알려줘. 그 후에 Olly에서 rightclick 하면 쉽게 API에 대한 정보를 얻을 수 있어.

See further. Download here:

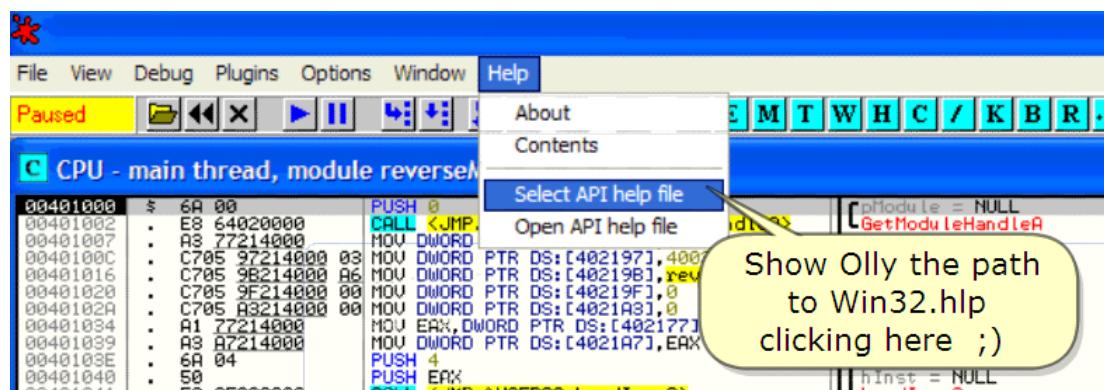
Download는 여기에서 해.

<http://www.tuts4you.com/request.php?258>

The extended winapi helpfile is here:

확장된 winapi help은 여기.

<http://www.tuts4you.com/request.php?1499>



Show Olly the path to Win32.hlp clicking here ;)

Olly는 Win32.hlp 경로를 보여준다.

If you want some indepth reading, then I strongly advise to read Goppit's tutorial

"Portable Executable File Format Compendium"

for a better understanding and winter evenings read and also gabri3l's

"weakness of the Windows API"

Search for those title here:

<http://arteam.accessroot.com/tutorials.html>

만약에 네가 깊이 있는 것을 읽고자 한다면, Goppit's tutorial "Portable Executable File Format Compendium" 을 강력히 추천한다.

좀 더 이해하기 좋다. 그리고 이것도 좋다.

"weakness of the Windows API"

```

FileHandleA> EAX 4003 reverseM.00401: 0 0 77J EAX
FileHandleA> [pfModule = NULL GetModuleHandleA
RsrcName = 4. hInst = NULL LoadIconA
FileA> FileA> pSecurity = NULL ShareMode = FILE_SHARE_READ|FILE_SHARE_ACCESS = GENERIC_READ|GENERIC_WRITE FileName = "Keyfile.dat" CreateFileA
B B Style = MB_OK|MB_APPLMODAL Title = "Key File ReverseMe" Text = "Evaluation period out of date. hOwner = NULL MessageBoxA
DkA> DkA> pOverlapped = NULL pBytesRead = reverseM.00402173 ExitProcess

```

Registers:

- EAX: 0
- ECX: 0
- EDX: 0
- EBX: 0
- ESP: 0
- EBP: 0
- ESI: 0
- EDI: 0
- EIP: 1
- C: 0
- P: 1
- A: 0
- Z: 1
- S: 0
- T: 0
- D: 0
- O: 0
- EFL: 0
- ST0: 0
- ST1: 0
- ST2: 0
- ST3: 0
- ST4: 0
- ST5: 0
- ST6: 0
- ST7: 0
- FST: 0
- FCW: 0

But for now, let's focus on the next window : the register's window.

이제는 다음 windows에 포커스를 맞추자. : register's window

First of all : notice the "weird" values for all data. This is due to the hexadecimal system.

I'm not going to explain this in this Part because we won't need it yet.

처음으로 : 모두 이상한 값이 들어있다. 이것은 16진수 시스템으로 되어 있다.

For more info : see Part 2 where I'll explain briefly. Just remember from this point on that computers ALWAYS work in the hexadecimal system. Or find more info googling for hexadecimal system.

좀 더 많은 정보 : Part 2를 봐라. 그곳에서 간단히 설명한다. 이곳에서 컴퓨터는 항상! 16진수 시스템으로 일한다는 것을 기억해. 16진수에 대해 좀 더 많은 정보는 구글링을 해.

Registers are "special places" in your computer's memory where we can store data.

You can see a register as a little box, wherein we can store something: a name, a number, a sentence. You can see a register as a placeholder.

레지스터는 "특별한 장소" 너의 컴퓨터 메모리에 우리는 데이터를 저장할 수 있다.

이 레지스터는 작은 박스 같아. 이곳에 무언가를 저장할 수 있어: 이름, 전화번호, 문장.

너는 지갑 같은 레지스터를 볼 수 있다.

#### ASEMBLY INFO:

On todays average WinTel CPU you have 9 32bit registers (w/o flag registers).

Their names are :

오늘날 평균적으로 WinTel(Windows/Intel) CPU는 9개의 32bit 레지스터를 사용한다. (w/o flag registers)

eax : Extended Accumulator Register

ebx : Extended Base Register

ecx : Extended Counter Register

edx : Extended Data Register

esi : Extended Source Index  
edi: Extended Destination Index  
ebp : Extended Base Pointer  
esp : Extended Stack Pointer  
eip : Extended Instruction Pointer

번역 주)

EAX : 함수 리턴값 저장. Win32 API 함수들은 모두 return value를 EAX에 저장한 후 return 합니다.

EBX : DS segment에서 data를 가리킬 때 사용합니다.

ECX : 반복문 명령어(LOOP)에서 참조 카운트로 사용됩니다. (루프 돌 때마다 ECX를 1씩 감소시킵니다.)

ESP : stack memory address를 가리킵니다. 어떤 명령어들(PUSH, POP, CALL, RET)은 ESP를 직접 조작하기도 합니다.(stack memory 관리는 프로그램에서 매우 중요하기 때문에 ESP를 다른 용도로 사용하지 않는 것이 좋습니다)

EBP : 함수가 호출되었을 때 그 순간의 ESP를 저장하고 있다가, 함수가 return하기 직전에 다시 ESP에 값을 되돌려줘서 stack이 깨지지 않도록 합니다.

ESI, EDI : 특정 명령어(LODS, STOS, REP MOVS, etc)와 함께 주로 메모리 복사에 사용됩니다.  
출처:reverscore.com

#### ASSEMBLY INFO:

Generally the size of those registers is 32bit(= 4bytes). They can hold data from 0-FFFFFF (unsigned). In the beginning most registers had certain main functions which the names imply, like ECX = Counter, but in these days you can -nearly- use whichever register you like for a counter.

대부분의 레지스터들은 32bit다. 그들은 data를 0000 0000 - FFFF FFFF까지 가지고 있다.(unsigned). 대부분의 레지스터들은 시작하는데 있어 중요한 function이 있다. 이름은 ECX처럼 = Counter을 암시한다. 요즘은 counter처럼 사용할 수 있다.

번역 주)unsigned: -가 없이 +만 있다. 즉 32bit sign(부호 있음) 일 때는  $-2^{31} \sim +2^{31}$  까지 표현할 수 있고 unsigned일 때는  $+2^{32}$  까지 표현할 수 있다.

There's one more thing you have to know about registers: although they are all 32bit large, some parts of them (16bit or even 8bit) can not be addressed directly.

좀 더 생각 해볼 게 있다. 네가 알고 있는 레지스터들은 큰 32bit 이다, 하지만 약간의 작은 부분도(16bit or 8bit) 있다. 작은 부분(16bit or 8bit)에는 주소에 직접적으로 접근할 수 없다.

Accessible are :

32bit register	16bit register	8bit
EAX	AX	AH/AL
EBX	BX	BH/BL
ECX	CX	CH/CL
EDX	CX	CH/CL

ESI	SI	-
EDI	DI	-
EBP	BP	-
ESP	SP	-
EIP	IP	-

This is all very theoretical, an example will make it clear :

이것은 매우 이론적이다. 이 예를 보면 명확해진다.

For example :

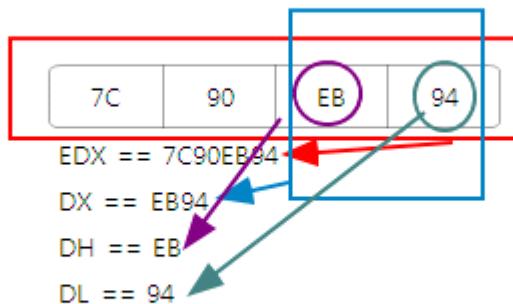
7C	90	EB	94
----	----	----	----

EDX == 7C90EB94

DX == EB94

DH == EB

DL == 94



You can find this and more about it and most other subjects in the Basics of assembler.doc that is included in this package.

Basics of assembler.doc에서 그것에 대한 많은 정보를 그리고 다른 주제들을 찾을 수 있다.  
그것은 이 package에 포함되어 있다.

The Stack is a part in memory where you can store different things for later use. See it as a pile of books in a chest where the last put in, is the first to grab out. Or imagine the stack as a paperbasket where you put in sheets. The basket is the stack and a sheet is a memory address (indicated by the stack pointer) in that stack segment. Remember following rule: the last sheet of paper you put in the stack, is the first one you'll take out! "Top is first off". The command 'push' saves the contents of a register onto the stack. The command 'pop' grabs the last saved contents of a register from the stack and puts it in the (addressed) register.

That brings us to the Stack window

Stack window를 가져온다.

The Stack is a part in memory where you can store different things for later use. See it as a pile of books in a chest where the last put in, is the first to grab out. Or imagine the stack as a paperbastet where you put in sheets.

스택은 메모리의 부분이다. 그곳에 다른 용도로 저장할 수 있다. 비스킷 통처럼 보인다. 비스킷 통에 마지막으로 넣는 것이 먼저 나온다. 스택은 비스킷 통 같으니까 상상을 해봐.

The basket is the stack and a sheet is a memory address(indicated by the stack pointer) in that stack segment.

통은 스택이고 sheet는 stack segment의 메모리 주소다.(스택포인터에 의해 가리켜진다)

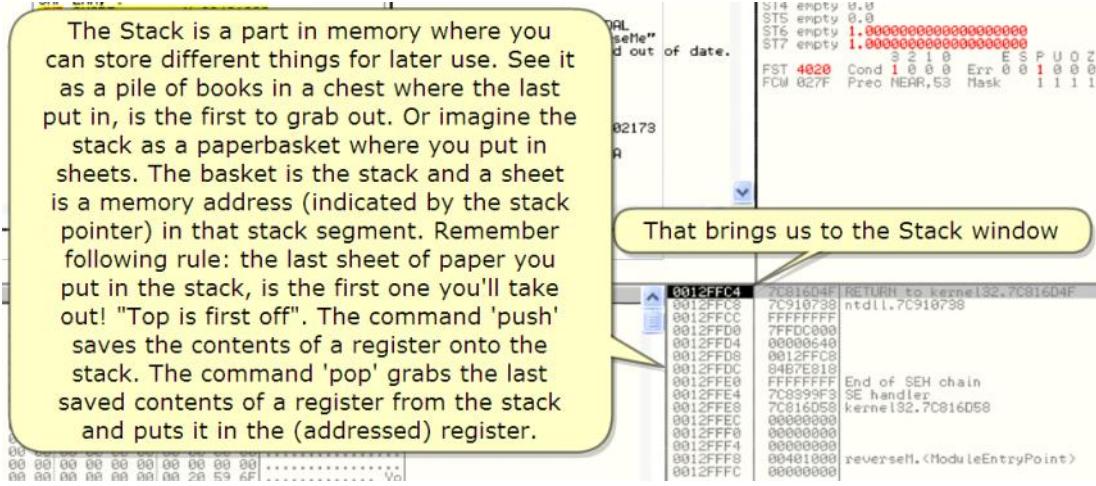
Remember following rule: the last sheet of paper you put in the stack, is the first one you'll take out! "Top is first off".

이 룰을 기억해라: '마지막 통 sheet는 stack에 넣은 것'은 첫번째로 나온다.

"제일 위에 있는 놈이 먼저 나온다."

The command 'push' saves the contents of a register onto the stack. The command 'pop' grabs the last saved contents of a register from the stack and puts it in the (addressed) register.

PUSH 명령어는 레지스터의 content를 stack에 넣는다. POP 명령어는 레지스터의 마지막에 저장된 content를 stack으로부터 꺼낸다. 그리고 그것을 주소 레지스터(EIP)에 넣는다.



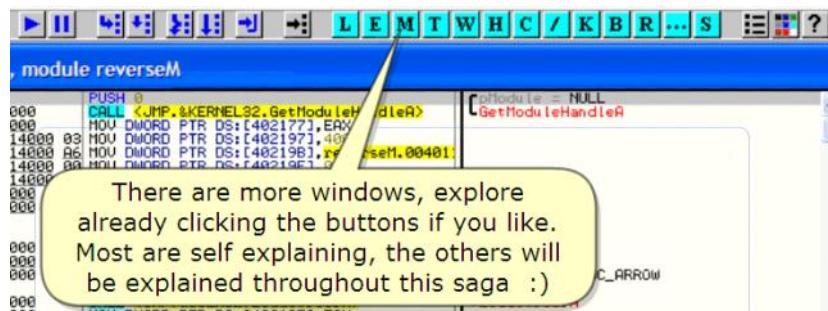
That brings us to the Stack window

And then finally, the dump window. Here you see the opcodes nicely organized in rows and columns. This can be arranged otherwise, just rightclick in the dump window and play a little with the settings.

Address	Hex dump	ASCII
00402000	20 4B 65 79   20 46 69 60   65 20 52 65   76 65 72 73	Key File Revers
00402010	65 4D 65 00   00 00 20 47   76 61 6C 75   61 74 69 6F	eMe... Evaluatio
00402020	6E 20 70 65   72 69 6F 64   20 6F 75 74   20 6F 66 20	n period out of
00402030	64 61 74 65   2E 20 50 75   72 63 68 61   73 65 20 6E	date. Purchase n
00402040	65 77 20 6C   69 63 65 6E   73 65 00 00   00 00 00 00	ew license.....
00402050	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00	.....
00402060	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00	.....
00402070	00 00 00 00   00 00 00 00   00 20 4B 65   79 66 69 6C 65	..... Keyfile
00402080	2E 64 61 74   00 20 4B 65   79 66 69 6C 65   20 69 73	.dat. Keyfile is
00402090	20 6E 6F 74   20 76 61 6C   69 64 2E 20   53 6F 72 72	not valid. Sorr
004020A0	79 2E 00 00   00 00 00 00   00 00 00 00   00 00 00 00	y.....
004020B0	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00	.....
004020C0	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00	.....
004020D0	00 00 00 00   00 00 00 00   00 00 00 00   00 20 59 6F	..... Yo

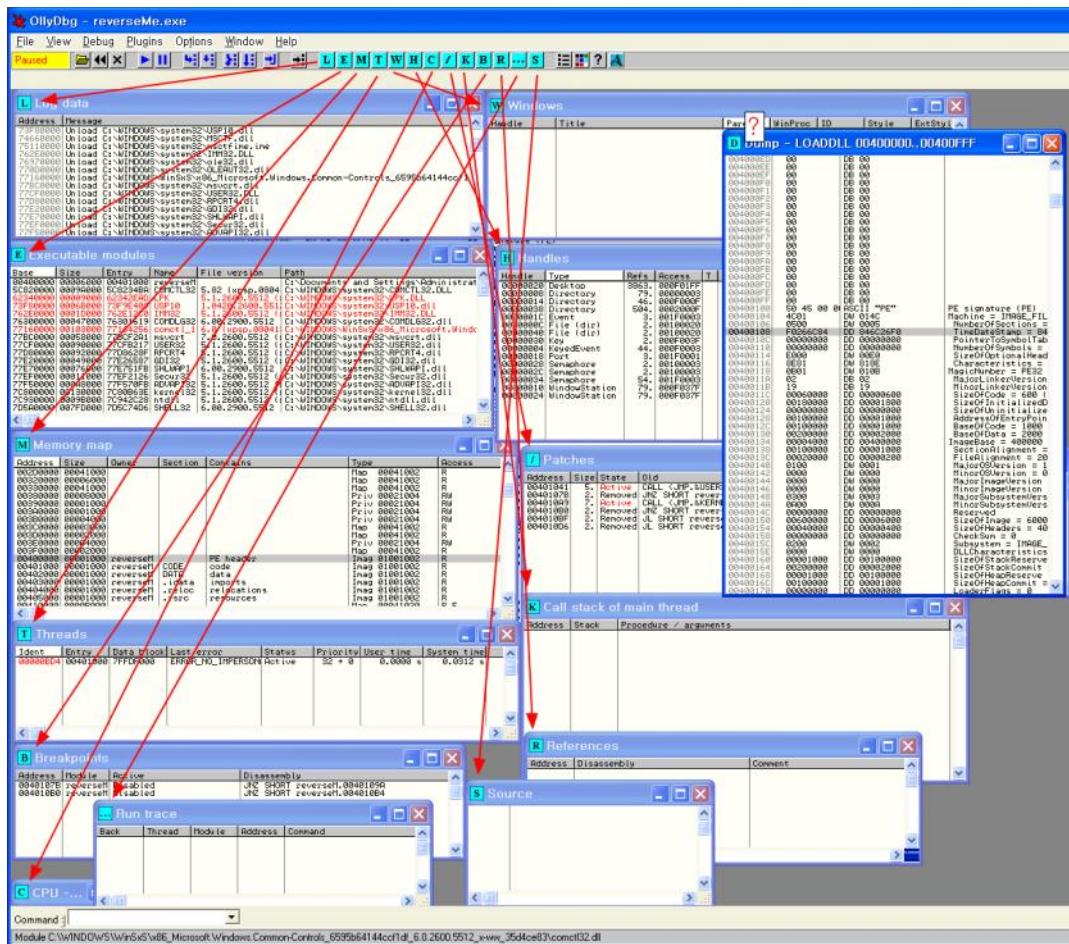
And then finally, the dump window. Here you see the opcodes nicely organized in rows and columns. This can be arranged otherwise, just rightclick in the dump window and play a little with the settings.

마지막으로 dump window다. 이곳은 opcode가 행과 열로 조직화 되어 있다. 정렬되어 있다. dump window에서 right 클릭하고 약간의 세팅을 하면 된다.



There are more windows, explore already clicking the buttons if you like. Most are self explaining, the others will be explained throughout this saga :)

많은 windows가 있다. 이미 탐험해 봤다. 이 버튼들은 대부분 자기 자신을 스스로 설명한다. 예를 들자면 L은 Log data, E는 Executable modules 등



And then finally, time has come to take a look in the first application we will reverse.

이제 마지막으로, 첫번째 application을 reverse 하는 시간이 왔다.

#### 4. Study of the target

We have loaded the ReverseMe in Olly and Olly has stopped execution of the program in the EP(entry point) by placing a INT3 (CC) instruction before the line.

우리는 ReverseMe를 Olly에서 로드했다. 그리고 Olly는 프로그램 시작하는 EP에 멈춰있다.

Line 뒤에 INT3 명령어가 위치해 있다.

Let's say for now (more exact, see later when we talk about the TLS Callback function) that this is the first line of code from the program that is executed. This way, Olly can control the program whilst we.... can control Olly.

내가 말하고자 하는 것은 (좀 더 정확한 것은 TLS CallBack function을 말할 때 보자.) 그것은 실행된 프로그램의 첫번째 줄이다. Olly는 프로그램을 제어할 수 있다. 그때 우리가 Olly를 제어할 수 있다.

**번역 주) 먹고 먹히는 관계...**

Studying the behaviour of the program is very important because it can give us clues how to attack the target. So, first, let's take a look at the behaviour of the ReverseMe.

프로그램의 행동을 아는 것은 매우 중요하다. 왜냐하면 그것은 우리에게 어떻게 목표물을 공격해야 할지 단서를 준다. 먼저, ReverseMe의 행동을 살펴보자.

If we click this "Run" button, Olly will let the program run freely. BTW, shortcut is F9.

If we click this "Run" button, Olly will let the program run freely. BTW, shortcut is F9

만약에 우리가 "Run"을 누르게 되면 Olly는 프로그램이 자유롭게 실행되게 놔둔다. 단축키는 F9

Bam. And our ReverseMe is running.

Bam. And our ReverseMe is running.

우리의 ReverseMe가 실행 중이다.

But.... we get some bad message(hereafter referred to as "Badboy"), right?  
and what happens if we click the "OK" button ???

그러나 우리는 약간의 나쁜 message를 얻었다. (이제부터 badboy로 부른다) 알았지?  
그리고 OK 버튼을 누르면 어떤 일이 발생할까?

Oops ... the program just exits !!!  
That is not what we want, right ???

Oops... the program just exits !!!

That is not what we want, right???

웁스. 프로그램이 종료됐다.

이것은 우리가 원한 게 아니잖아. 그렇지???

Mmmm, so let's restart and see  
better what happens.... to see if  
we can do something about it

Mmmm, so let's restart and see better what happens... to see if we can do something about it.

그래서 재시작하고 어떤 일이 일어나는지 보자.

## 5. Searching the patches

```
hTemplateFile = NULL
Attributes = READONLY|HIDDEN|SYSTEM|AF
Mode = OPEN_EXISTING
pSecurity = NULL
ShareMode = FILE_SHARE_READ|FILE_SHARE_WRITE
Access = GENERIC_READ|GENERIC_WRITE
FileName = "Keyfile.dat"
CreateFileA
```

BTW, note also that Olly has meanwhile analyzed the code. You can see the results here.  
Olly는 해석해서 적었고, 그 결과를 여기에서 볼 수 있다.

```
$ 6A 00 PUSH @
E8 64020000 CALL <JMP.> [402177].EBX
R3 77214000 MOU DWORD PTR [402177].EBX
```

What is the goal now? Well, we will step the code a line at a time to see what happens.  
We can do this by stepping over the code (shortcut key F8 or clicking here). This is also called "tracing". It is of course very important that you understand the code. Fortunately, there are only a few commands (mnemonics) to know and it is very easy to understand.  
This time, I will comment this assembler code a little but in following parts, you will already know and understand this yourself. More about these mnemonics (assembler code) can also be found in the document file included in this package. I said it already : assembler is relatively easy and there are only a few mnemonics to understand (for starting). Btw, code is always executed top to bottom unless the code is "redirected" elsewhere by jumps or calls (see more later).

What is the goal now? Well, we will step the code a line at a time to see what happens.  
We can do this by stepping over the code(shortcut key F8 or clicking here).

이제 목표가 무엇이냐? 우리는 code를 한 번에 한 줄씩 넘길 수 있다. 무슨 일이 일어나는지 봐라.

우리는 한 줄씩 code를 넘기는 것을 할 수 있다. (단축키는 F8이나 이곳을 누른다.)

This is also called "tracing". It is of course very important that you understand the code. Fortunately, there are only a few commands(mnemonics) to know and it is very easy to understand.

이것을 우리는 "추적"이라고 부른다. 이것은 네가 code를 이해하는데 매우 중요하다.

This time, I will comment this assembler code a little but in following parts, you will

already know and understand this yourself. More about these mnemonics (assembler code) can also be found in the document file included in this package.

이제, assembler code에 약간의 comment를 달겠으니 이번 parts는 나를 따라와. 너는 이미 알고 있고 이해했을 것이다. 좀더 많은 상징들은(assembler code) 내가 첨부한 문서에서 찾을 수 있다.

I said it already : assembler is relatively easy and there are only a few mnemonics to understand (for starting). Btw, code is always executed top to bottom unless the code is "redirected" elsewhere by jumps or calls (see more later).

내가 이미 말했잖아 : assembler는 비교적 쉬워. 오직 몇 개의 상징만 이해하면 된다.(시작하기 위해서). Code는 call문이나 jump에 의해 "redirected"(방향 재지정) 하지 않는 한 항상 code의 위에서 아래로 실행된다.

### ASSEMBLY INFO: PUSH

Syntax:

PUSH	operand
------	---------

PUSH is the opposite of POP. It stores a value on the stack and decreases it by the size of the operand that was pushed, so that ESP points to the value that was PUSHed.

PUSH의 반대는 POP이다. 그것은 값을 stack에 저장하고 ESP가 명령어의 크기만큼 줄어든다. 그래서 PUSH를 한 후에 ESP 포인트는 PUSH된 값을 가리킨다.

Hehe, so much for the theory, but don't worry, it will all become clear in a while.

헤헤, 너무 이론적이지?, 걱정하지 마. 그것은 곧 명확해 질 거야^^

First line of code is executed and nothing has happened yet but...

... notice the function of the stack and see how this "PUSH" instruction has pushed the value on the stack(00000000 == 0), let's continue ...

First line of code is executed and nothing has happened yet but...

첫번째 줄의 code는 실행됐다. 그리고 아직까지 아무런 일이 없다. 그러나

... notice the function of the stack and see how this "PUSH" instruction has pushed the value on the stack(00000000 == 0), let's continue...

Stack의 function이 알려주잖아. "PUSH" 명령이 들어온 후에 stack(0000 0000)의 값이 0이 된

것을 봐. 계속 진행하자.

#### ASSEMBLY INFO: CALL(Call)

Syntax:

CALL	something
------	-----------

The instruction CALL pushes the RVA(Relative Virtual Address) of the instruction that follows the CALL to the stack to know where to return to after the call and then executes a sub program/procedure.

CALL 명령은 RVA(상대적인 가상 메모리 주소)의 명령을 넣는다. call 명령은 call을 실행한 후에 return할 주소를 stack에 넣는다. 그리고 sub program/procedure를 실행한다.

"Call" can be used in the following ways:

"Call"은 이 방법대로 사용된다.

CALL 404000 (CALL address)

CALL EAX (CALL register - executes the procedure with address == value of EAX)

EAX에 있는 주소값을 실행한다.

CALL DWORD PTR [EAX] executes procedure at address with value of EAX

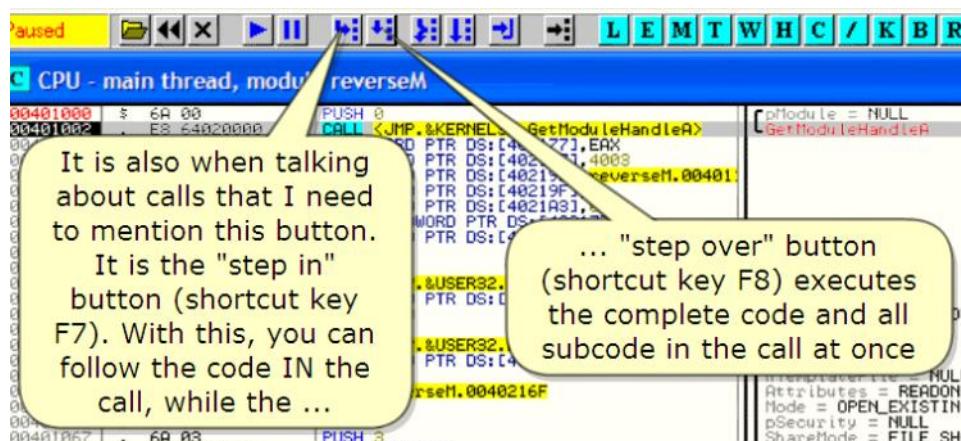
EAX에 있는 주소값을 실행한다.

CALL DWORD PTR [EAX+5] executes procedure at address with value of EAX+5

EAX+5에 있는 주소값을 실행한다.

CALL <JMP to API> is in fact also a CALL address, but it's kind of special because it executes an API (see further)

CALL <JMP to API> 또한 CALL address다. 그러나 이것은 특별해. 왜냐하면 API를 실행할 수 있거든.(나중에)

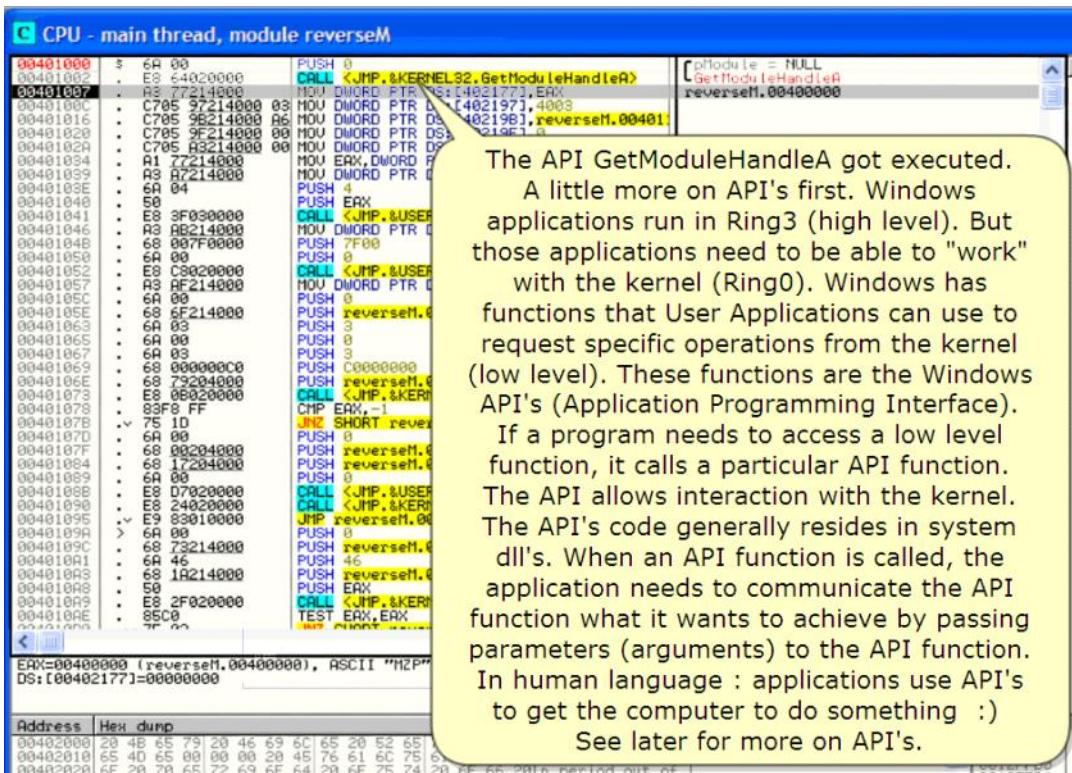


It is also when talking about calls that I need to mention this button. It is the "step in"

button(shortcut key F7). With this, you can follow the code IN the call, while the...

..."step over" button (shortcut key F8) executes the complete code and all subcode in the call at once

이것은 우리가 calls에 대해 말한 때 언급하기 위해 이 버튼이 필요하다. 이것은 "step in" 버튼이다.(단축키는 F7). 이 button을 누르면 call문 안으로 따라갈 수 있어. 반면에 "step over" button(단축키 F8)은 call문으로 자동으로 들어가서 code를 실행하고 우리는 다음 줄에 위치하게 돼지. 즉, call문 안으로 따라 들어가지 않고 간편하게 code를 실행할 수 있어.



The API GetModuleHandleA got executed. A little more on API's first. Windows applications run in Ring3 (high level). But those applications need to be able to "work" with the kernel(Ring0). Windows has functions that User applications can use to request specific operations from the kernel(low level). These functions are the Windows API's (Application Programming Interface).

API GetModuleHandleA가 실행됐다. 먼저 약간의 API의 정보를 알려 줄께. Windows applications은 Ring3(high level) 실행된다. Windows의 User applications은 kernel로부터 (low level) 특별한 명령을 요구하는데 사용되는데 사용한다.

If a program needs to access a low level function, it calls a particular API function. The API allows interaction with the kernel. The API's code generally resides in system dll's.

만약에 프로그램이 low level 제어가 필요할 때, API function을 불러. API는 kernel과 대화하는 것이 허락되거든. API는 보통 system dll에 존재한다.

When an API function is called, the application needs to communicate the API function what it wants to achieve by passing parameters(arguments) to the API function. In human language : applications use API's to get the computer to do something :)

API를 부를 때, application은 API function과 대화하는 게 필요하다. Application은 API function으로 넘겨진 parameter로 API와 대화할 수 있어. 인간은 : application은 computer에게 무언가를 시키는데 API를 사용한다.

See later for more on API's.

나중에 API에 대해 좀 더 살펴보자.

**ASSEMBLY INFO : MOV (Move)**  
**Syntax: MOV dest,src**  
 $(\text{dest}=\text{destination}, \text{src}=\text{source})$

This is an easy to understand instruction.  
MOV copies the value from src to dest and  
src stays what it was before.

There are some variants of MOV:  
**MOVS/MOVSB/MOVSW/MOVSD EDI,ESI:**  
Those variants copy the byte/word/dword  
ESI points to, to the space EDI points to.  
**MOVsx: MOVsx expands Byte or Word**  
**operands to Word or Dword size and keeps**  
**the sign of the value.**  
**MOVzx: MOVzx expands Byte or Word**  
**operands to Word or Dword size and fills the**  
**rest of the space with 0.**

ASSEMBLY INFO: MOV (Move)

Syntax:

Mov	dest	src
-----	------	-----

$(\text{dest}=\text{destination}, \text{src}=\text{source})$

This is an easy to understand instruction. MOV copies the value from src to dest and src stays what it was before.

이것은 이해하기 쉬운 명령어다. MOV는 src에서 dest로 값을 복사한다. 그리고 src 값은 명령을 실행하기 전의 값을 유지한다.

There are some variants of MOV:

다양한 MOV가 있다.

**MOVS/MOVSB/MOVSW/MOVSD EDI,ESI:**

Those variants copy the byte/word/dword ESI points to, to the space EDI points to.

다양하게 Byte/word/dword ESI points에서 EDI 공간에 값을 복사한다.

**MOVsx: MOVsx expands Byte or Word operands to Word or Dword size and keeps the sign of the value.**

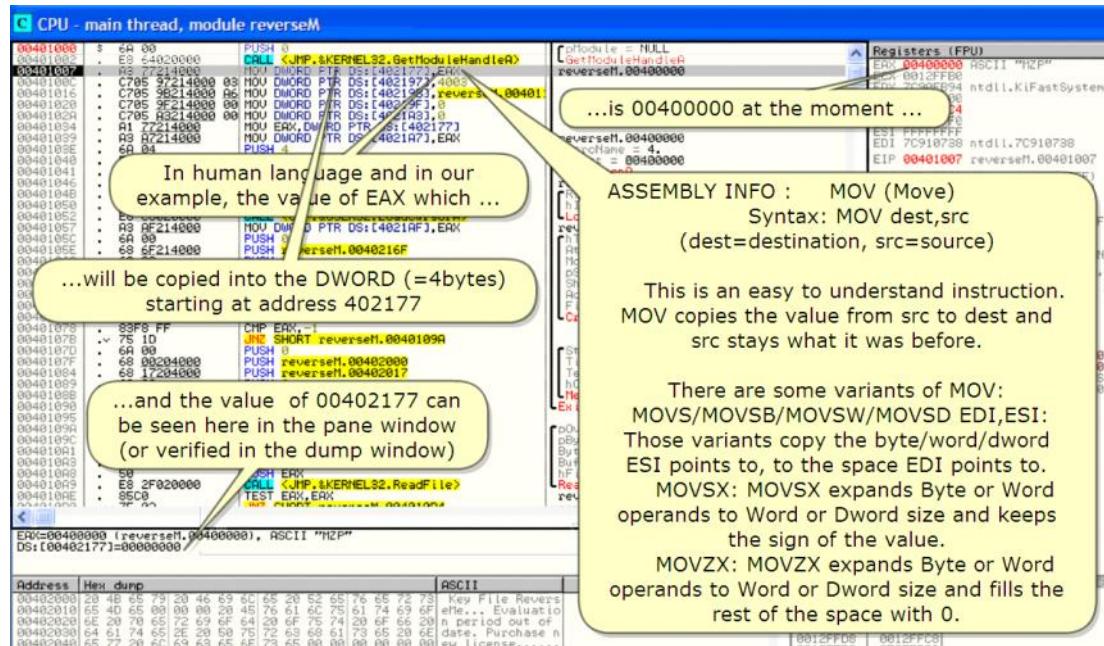
Movsx는 Bytes나 Word에서 Word나 Dword 크기로 확장한다. 그리고 sign 값은 유지한다.

번역 주)sign값은 +,- 값이 있는 것. 즉 Byte 크기에서 Word 크기로 확장할 때 sign값(부호)를 유지한다. 예를 들자면 byte값인 0000 0001을 word값인 0000 0000 0000 0001로 확장할 때 sign값(부호)를 유지합니다.

**MOVzx: MOVzx expands Byte or Word operands to Word or Dword size and fills the rest of the space with 0.**

Movzx는 Bytes나 Word에서 Word나 Dword 크기로 확장한다. 그리고 나머지 영역은 0으로

채운다.



In human language and in our example, the value of EAX which...

...is 00400000 at the moment...

...will be copied into the DWORD (=4bytes) starting at address 402177

인간이 이해하기에 예제에서 현재 EAX 값이 0040 0000 이다.

402177의 시작 주소로 Dword(4bytes) 값이 복사되어진다.

...and the value of 00402177 can be seen here in the pane window(or verified in the dump window)

00402177의 값을 pane window 볼 수 있다. (또는 dump windows를 검증됐다)

## INFO:

If this sounds like rubbish, once again, don't worry, it is not that important here. If you are new to this, just study it a moment and try to understand. It will all become clear after a little while.

내 말이 잘 이해가 되지 않니? 다시 말하지만 걱정 하지마. 이것은 여기에서 중요하지 않아.

네가 초보라면 이것만 배워. 그리고 이해하려고 해. 지금 이것들은 약간이 시간이 지난 후에 명확해 질 것이다.

We just continue our journey in outer space. Follow along. Some instructions that you understand by now are carried out until...

우리의 여행을 시작하자. 따라와. 약간의 명령어들은 네가 이해했을 거야.

Ok. So far, all good. Now, perhaps you think that nothing is happening but...take a look at this text...

지금까지, 좋아. 이제 어쩌면 네가 생각하고 있는 것이 아무 일도 안 일어날 것이라 생각할거야. 그러나 텍스트를 봐.

CPU - main thread, module reverseM

```

00401000 5: 6A 00          PUSH 0
00401002  .: E8 54020000  CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007  .: E9 52214000  MOU DIWORD PTR DS:[402177],EAX
0040100C  .: C705 55214000 03  MOU DIWORD PTR DS:[402157],0003
00401015  .: C705 55214000 A5  MOU DIWORD PTR DS:[402158],reverseM.004010
00401028  .: C705 55214000 00  MOU DIWORD PTR DS:[402149],0
00401034  .: C705 55214000 00  MOU DIWORD PTR DS:[402149],0
00401032  .: A1 77214000  MOU DIWORD PTR DS:[402177],EAX
00401035  .: A9 77214000  MOU DIWORD PTR DS:[402149],EAX
00401040  .: 50  PUSH ERX
00401041  .: E8 3F030000  CALL <JMP.&USER32.LoadIconA>
00401045  .: A3 RB214000  MOU DIWORD PTR DS:[4021HB],ERX
00401048  .: E8 00710000  PUSH 7F00
00401050  .: E8 00  PUSH 0
00401052  .: E8 C3020000  CALL <JMP.&USER32.LoadIconA>
00401057  .: A3 BE214000  MOU DIWORD PTR DS:[402149],ERX
0040105C  .: 50  PUSH reverseM.004010
0040105E  .: 50  PUSH 3
00401063  .: 6A 00  PUSH 0
00401065  .: 6A 00  PUSH 0
00401067  .: 6A 03  PUSH 3
00401069  .: 68 00000000  PUSH C0000000
0040106E  .: 68 73204000  PUSH reverseM.00402079
00401073  .: E8 00020000  CALL <JMP.&KERNEL32.CreateFileA>
00401078  .: 83F8 FF  CMP EAX,-1
0040107B  .: 75 1D  JNC SHORT reverseM.0040109A
0040107D  .: 6A 00  PUSH 0

```

Ok. So far, all good. Now, perhaps you think that nothing is happening but .... take a look at this text ...

... but this is the badboy message !!!

So, if we would continue executing the code, then it is here that the bad message appears. Understood?

그러나 이것은 badboy message!!!

우리가 계속 code를 실행하면 우리는 이곳에 위치하고 bad message를 보게 될 거야. 이해 돼?

And see why the software exits after the badboy message!

...unless of course we can avoid somehow this badboy + exit ???

..., but this is badboy message !!!

So, if we would continue executing the code, then it is here that the bad message appears. Understood?

그러나 이것은 badboy message!!!

우리가 계속 code를 실행하면 우리는 이곳에 위치하고 bad message를 보게 될 거야. 이해 돼?

And see why the software exits after the badboy message!

...unless of course we can avoid somehow this badboy + exit ???

badboy message 후에 software를 종료하려 하잖아. 봐봐.

우리가 badboy + exit를 회피하지 않는 한

```

00401057  .: H9 HE214000  MOU DIWORD PTR DS:[4021HF],ERX
0040105C  .: 6A 00  PUSH 0
0040105E  .: 68 55214000  PUSH reverseM.0040216F
00401063  .: 6A 03  PUSH 3
00401065  .: 6A 00  PUSH 0
00401067  .: 6A 03  PUSH 3
00401069  .: 68 00000000  PUSH C0000000
0040106E  .: 68 73204000  PUSH reverseM.00402079
00401073  .: E8 00020000  CALL <JMP.&KERNEL32.CreateFileA>
00401078  .: 83F8 FF  CMP EAX,-1
0040107B  .: 75 1D  JNC SHORT reverseM.0040109A
0040107D  .: 6A 00  PUSH 0
00401084  .: 68 00204000  PUSH reverseM.00402173

```

Let's take a better look in the API here. Aha, if you downloaded the win32.hlp, it will already come in handy! Let's see this more in detail using this win32.hlp because here it must be decided.

Let's take a better look in the API here. Aha, if you downloaded the win32.hlp, it will already come in handy! Let's see this more in detail using this win32.hlp because here it must be decided.

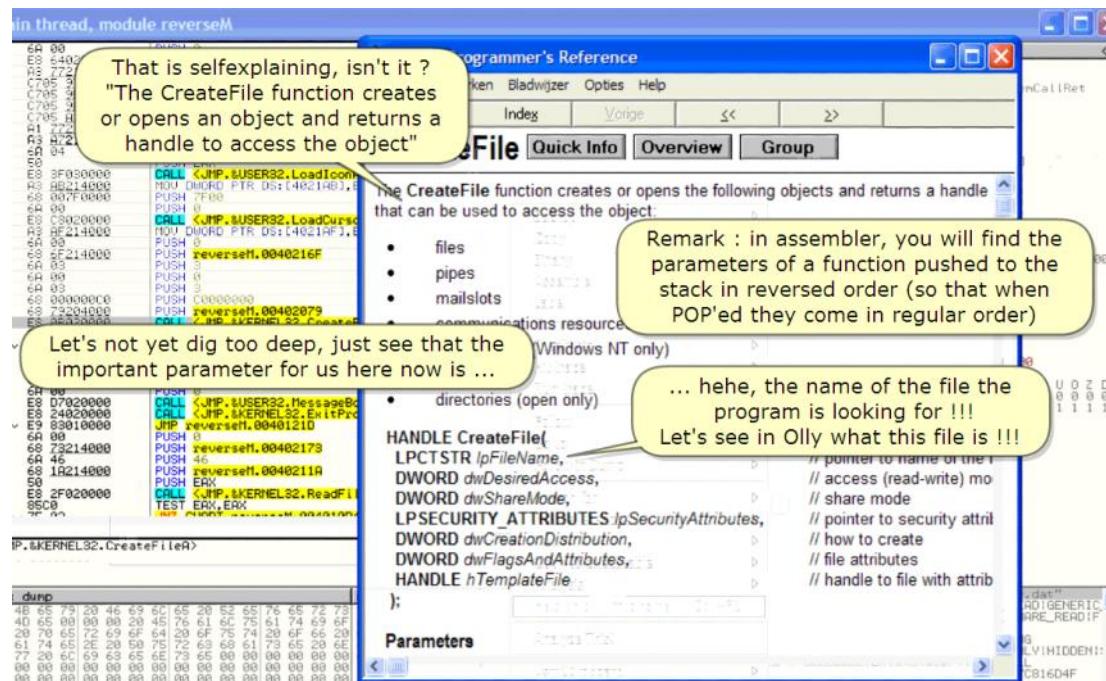
API를 봐라. 아하! 만약에 네가 Win32.hlp를 다운 받았다면, 이것은 이미 내 손안에 있지. 자세한 정보는 Win32.hlp를 사용해서 보자. 왜냐하면 이것에 의해 badboy나 goodboy가 결정된다.

INFO: Very often, it's in the call just before a conditional jump that is decided if the jump will eventually be taken or not.

Call문 안에서 많이 결정 돼. 그 후에 conditional jump를 한다. 결국 이 call문이 jump를 할지 말지 결정해.

That's it ...

바로 그것이다...



That is selfexplaining, isn't it? "The CreateFile function creates or opens an object and returns a handle to access the object"

여기 설명이 끝내주게 잘 되어있네. 그렇지 않나? CreateFile function은 object를 만들거나 연다. 그리고 object에 접근 handle을 돌려준다.

Remark : in assembler, you will find the parameters of a function pushed to the stack in reversed order (so that when POP'ed they come in regular order)

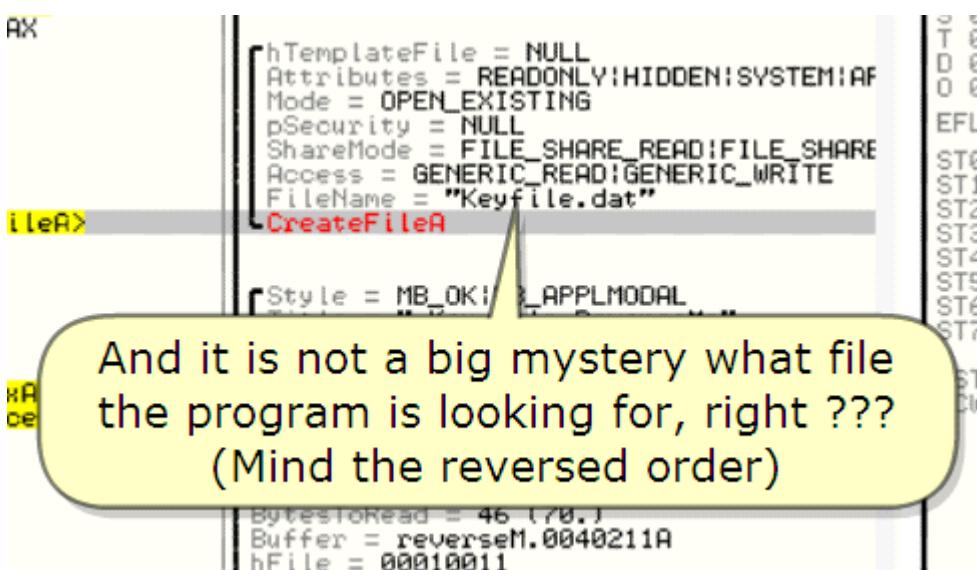
주목 : 어셈블러에서 너는 stack에 function의 parameter가 반대의 순서로 넣어지는 것을 찾을 수 있다.(그래야 POP 할 때 순서대로 나온다. Last In First Out)

Let's not yet dig too deep, just see that the important parameter for us here now is...

아직 깊게 들어가지 마라. 우리에게 중요한 parameter를 봐라.



번역 주) 위 사진에서는 `lpFileName`이 먼저 선언되어 있는데 Olly에서 보면 제일 마지막에 `FileName0` 있다. 이상하게 생각하지 말자. Program에서 sub program으로 parameter를 넘길 때는 항상 반대로 넘긴다. Parameter는 sub program으로 넘기는 변수다.



...hehe, the name of the file the program is looking for !!! Let's see in Olly what this file is !!!

And it is not a big mystery what file the program is looking for, right ??? (Mind the reversed order)

헤헤, Olly에서 찾는 file 이름이 무엇인지 봐.

그리고 이것은 프로그램의 큰 문제가 아니야. (반대 순서를 기억하자.)

In short: the ReverseMe is looking for a file called "Keyfile.dat", but will not find it BECAUSE THE PROGRAM IS UNREGISTERED! Ok. Let's continue and see what goes on next.

요약하자면: ReverseMe가 찾는 file은 "Keyfile.dat"로 불린다. 그러나 그것을 찾을 수 없다. 왜냐하면 프로그램은 등록되지 않았기 때문에. 이제 계속해서 다음에 어떻게 진행되는지 보자.

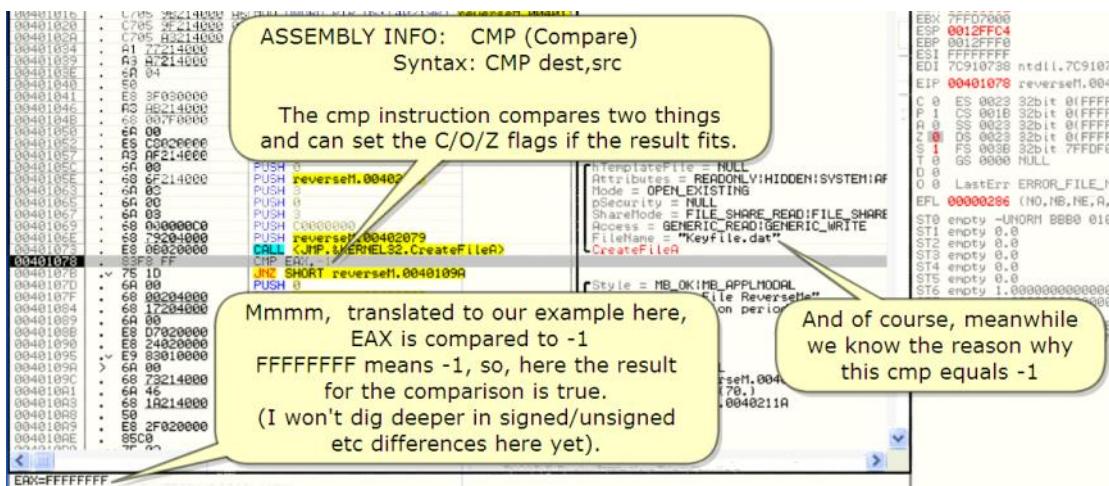
ASSEMBLY INFO: CMP (Compare)

Syntax:

CMP

dest

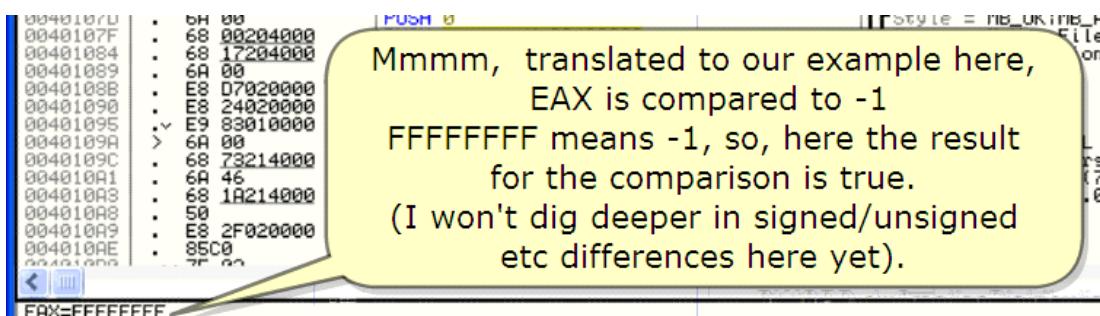
src



The cmp instruction compares two things and can set the C/O/Z flags if the result fits.

2가지 값을 비교하고 그에 맞는 C/O/Z flag를 set한다.

번역 주)Carry-flag, Overflow-flag, Zero-flag.



Mmmmm, translated to our example here, EAX is compared to -1

FFFFFFFFFF means -1, so, here the result for the comparison is true.

(I won't dig deeper in signed/unsigned etc differences here yet).

And of course, meanwhile we know the reason why this cmp equals -1

여기서 우리의 예제를 번역하자면, EAX는 -1과 비교된다.

FFFF FFFF는 -1과 같다. 여기에서 비교 값은 true다.

(나는 signed/unsigned가 다른 점을 아직 깊게 들어가지 않는다.)

물론, 우리는 그 이유를 안다. Cmp가 왜 -1과 같은지

번역 주)FFFF FFFF + 1 = 0 0이 됨. 그러니까 FFFF FFFF = -1 임. 2진수 변환하기 찾아보시면 도움이 될니다.

#### IMPORTANT INFO:

Throughout this series, I will deliberately tell some inconsistencies not to have to dig too deep in yet if I feel that you may not have sufficient/profound knowledge to understand already. I hope you will pardon me for this...

이 시리즈를 하는 동안, 나는 고의로 모순적으로 말할 거야. 아직 깊게 들어가지 않을 거야.

네가 이것을 충분히 따라올 수 있는 지식이 없다고 생각해. 그 점에 대해서 미안하게 생각해.

CPU - main thread, module reverseM

REMARK : see more info on jumps in the document included. This instruction is really important. See all (conditional) jumps there.

Our example here :

JNZ == Jump if not zero ZF=0 (z-flag)

The result from the compare however IS zero (they are equal) and so the jump will not be executed.. but the badboy will be executed.

So, we need to find a way to jump passed this badboy, right ?

This is also where the settings from ARTeam's ollydbg.ini file comes in handy ... You can easily see if the JNZ is going to be executed. The arrow would have turned red if jumping !!! (You can also change these options manually. See later)

REMARK : see more info on jumps in the document included. This instruction is really important. See all(conditional) jumps there.

주목 : jump에 대해 좀 더 많은 정보를 볼 수 있는 문서를 첨부했다. 이 명령은 정말로 중요하다. 조건 jump를 그곳에서 봐.

Our example here :

JNZ == Jump if not zero ZF= 0 (z-flag)

The result from the compare however IS zero (they are equal) and so the jump will not be executed.. but the badboy will be executed.

JNZ == zero가 아닐 때 jump ZF = 0(Z-Flag)

이 결과값은 비교에서 나온다. 그러나 결과값이 zero 이고(EAX와 -1은 같다) jump가 되지 않아서 badboy가 실행될 것이다.

This is also where the settings from ARTeam's ollydbg.ini file comes in handy... You can easily see if the JNZ is going to be executed. The arrow would have turned red if jumping !!!(You can also change these options manually. See later)

So, we need to find a way to jump passed this badboy, right?

이것은 ARTeam's ollydbg.ini 파일에서 온 setting이다. 너는 JNZ가 실행 후에 도착할 곳을 쉽게 볼 수 있다. Jump하게 된다면 붉은색 화살표로 바뀐다. (또한 수동으로 그것들의 option을 바꿀 수 있다. 나중에 봐)

그래서, 우리는 jump를 해서 badboy를 피하는 방법을 찾는 것이 필요하다. 이해했어?

ASSEMBLY INFO:

And this is where the flags come in ...

Flags are single bits which indicate the status of something. The flag register on modern 32bit CPU's is 32bits large. There are 32 different flags, but don't worry. You will mostly only need 3 of them in reversing. The Z-Flag the O-Flag and the C-Flag. For reversing you need to know these flags to understand if a jump is executed or not. (Or use trial and error LOL). This register is in fact a collection of different 1-bit flags. A flag is a sign, just like a green light means: 'ok' and a red one 'not ok'. A flag can only be '0' or '1', meaning 'not set' or 'set'.

#### ASSEMBLY INFO:

And this is where the flags come in...

Flags는 여기에 있다.

Flags are single bits which indicate the status of something. The flag register on modern 32bit CPU's is 32bits large. There are 32 different flags, but don't worry.

Flag들은 어떤 상태를 가리키는 단일 bit다. 현대 32bit CPU의 Flag register는 32bits 보다 크다. 그들은 32개의 다른 flags 들이 있다. 그러나 걱정하지 마라.

You will mostly only need 3 of them in reversing. The Z-Flag the O-Flag and the C-Flag. For reversing you need to know these flags to understand if a jump is executed or not. (Or use trial and error LOL).

거의 대부분 리버싱에서는 3개만 필요하다. Z-Flag, O-Flag, C-Flag.

리버싱을 위해 Jump가 실행되거나 실행되지 않을 때 flag들을 이해해야 한다.(실험과 오류로) This register is in fact a collection of different 1-bit flags. A flag is a sign, just like a green light means: 'ok' and a red one 'not ok'. A flag can only be '0' or '1', meaning 'not set' or 'set'.

Register는 다른 1bit의 모임이다. Flag는 sign, green light와 같다: 'ok' 그리고 red는 'not ok'.

Flag는 '0' 이거나 '1', 뜻은 'not set', 'set'

Conclusion for our example : just change the Z-flag (by doubleclicking the value) and we will jump passed the badboy !!!

이 예제의 결론은 : Z-Flag를 바꾼다.(doubleclick 한 값에 의해) 그리고 badboy를 피해서 jump할 것이다.

C CPU - main thread, module reverseM

```

00401000 $ 6A 00          PUSH 0
00401002 . E8 64020000  CALL <JMP.&KERNEL32.GetModuleHandleA>
00401004 . A3 77214000  MOU DWORD PTR DS:[402177],EAX
00401006 . C705 97214000  MOU DWORD PTR DS:[402197],4003
00401016 . C705 9B214000  MOU DWORD PTR DS:[402198],reverseM.004011
00401020 . C705 9C214000  MOU DWORD PTR DS:[4021A0],3
00401024 . A1 77214000  MOU EAX,DWORD PTR DS:[402177]
00401028 . A3 97214000  MOU DWORD PTR DS:[4021A7],EAX
00401030 . E8 04          PUSH 4
00401032 . S8 00          PUSH EAX
00401034 . E8 3F030000  CALL <JMP.&USER32.LoadIconA>
00401046 . A3 9B214000  MOU DWORD PTR DS:[4021A8],EAX
00401048 . E8 007F0000  PUSH 7F00
00401050 . E8 00          CALL <JMP.&USER32.LoadCursorA>
00401052 . E8 9C020000  MOU DWORD PTR DS:[4021AF],EAX
00401057 . A3 9F214000  MOU DWORD PTR DS:[4021A9],reverseM.0040216F
00401059 . E8 00          PUSH 0
00401062 . E8 6F214000  PUSH reverseM.0040216F
00401063 . E8 03          PUSH 3
00401065 . E8 00          PUSH 0
00401067 . E8 03          PUSH 3
00401069 . E8 000000C8  PUSH C0000000
0040106E . E8 79204000  PUSH reverseM.00402079
00401072 . E8 00          PUSH 0
00401077 . E8 00          PUSH 0
0040107D . E8 00204000  PUSH reverseM.00402000
0040107F . E8 17204000  PUSH reverseM.00402017
00401084 . E8 00          CALL <JMP.&USER32.MessageBoxA>
00401090 . E8 24020000  MOU DWORD PTR DS:[4021A0],reverseM.00401210
00401095 . E8 83010000  MOU DWORD PTR DS:[4021A7],EAX
0040109A . E8 00          PUSH 0
0040109C . E8 73214000  PUSH reverseM.00402173
004010A1 . E8 46          PUSH 46
004010A3 . E8 10214000  PUSH reverseM.0040211A
004010A8 . E8 00          PUSH EAX
004010A9 . E8 2F020000  CALL <JMP.&KERNEL32.ReadFile>
004010AE . E8 95C0 FF      TEST EAX,EAX

```

Hehe, see the magic !!!  
So, if we continue stepping now ...

Hehe, see the magic !!!

So, if we continue stepping now...

헤헤, 예술이다 !!!

그래서, 계속 하겠다.

번역 주)위의 사진과 비교해보면 알겠지만 Z-Flag가 0 -> 1로 바뀌었다.

But first something else. Here, we can set a breakpoint (=BP) ourselves. Olly will set this as a visible red VA. It's easy for later to remember where we will need to make changes. Set a BP by pressing F2 or by doubleclicking the opcodes.

그러나 먼저 다른 방법을 사용하겠다. 여기에 BP를 설정할 수 있다. Olly는 VA가 red로 보일 때 설정된 것이다. 이것은 나중에 우리가 변화하는 지점을 기억하는데 매우 좋다. BP를 세팅 할 때 F2를 눌러라. 또는 opcodes를 doubleclick 해라.

번역 주)BP: breakpoint(0이하 BP), 데이터 중단점. BP를 설정한 곳 바로 전까지 실행이 된다.

BP가 걸려있는 곳의 첫번째 byte를 int3로 바꿔서 멈추는 것이다. 자세한 내용은 0/ series 후에 나온다.

C CPU - main thread, module reverseM

```

00401000 $ 6A 00          PUSH 0
00401002 . E8 64020000  CALL <JMP.&KERNEL32.GetModuleHandleA>
00401004 . A3 77214000  MOU DWORD PTR DS:[402177],EAX
00401006 . C705 97214000  MOU DWORD PTR DS:[402197],4003
00401016 . C705 9B214000  MOU DWORD PTR DS:[402198],reverseM.004011
00401020 . C705 9C214000  MOU DWORD PTR DS:[4021A0],3
00401024 . A1 77214000  MOU EAX,DWORD PTR DS:[402177]
00401028 . A3 97214000  MOU DWORD PTR DS:[4021A7],EAX
00401030 . E8 04          PUSH 4
00401032 . S8 00          PUSH EAX
00401034 . E8 3F030000  CALL <JMP.&USER32.LoadIconA>
00401046 . A3 9B214000  MOU DWORD PTR DS:[4021A8],EAX
00401048 . E8 007F0000  PUSH 7F00
00401050 . E8 00          CALL <JMP.&USER32.LoadCursorA>
00401052 . E8 9C020000  MOU DWORD PTR DS:[4021AF],EAX
00401057 . A3 9F214000  MOU DWORD PTR DS:[4021A9],reverseM.0040216F
00401059 . E8 00          PUSH 0
00401062 . E8 6F214000  PUSH reverseM.0040216F
00401063 . E8 03          PUSH 3
00401065 . E8 00          PUSH 0
00401067 . E8 03          PUSH 3
00401069 . E8 000000C8  PUSH C0000000
0040106E . E8 79204000  PUSH reverseM.00402079
00401072 . E8 00204000  CALL <JMP.&KERNEL32.CreateFileA>
00401077 . E8 00          PUSH 0
0040107D . E8 00204000  PUSH reverseM.00402000
0040107F . E8 17204000  PUSH reverseM.00402017
00401084 . E8 00          CALL <JMP.&USER32.MessageBoxA>
00401090 . E8 24020000  MOU DWORD PTR DS:[4021A0],reverseM.00401210
00401095 . E8 83010000  MOU DWORD PTR DS:[4021A7],EAX
0040109A . E8 00          PUSH 0
0040109C . E8 73214000  PUSH reverseM.00402173

```

Let's execute the JNZ and continue the journey in outer space LOL

C CPU - main thread, module reverseM

```

00401000 $ 6A 00 CALL QWORD PTR DS:[4021773],BX
00401002 . E8 64020000 MOU DWORD PTR DS:[4021373],4008
00401007 . E8 77214000 MOU DWORD PTR DS:[40219F3],4008
0040100C . C702 97214000 03 MOU DWORD PTR DS:[40219F3],reverseM.00401
00401015 . C702 98214000 A6 MOU DWORD PTR DS:[40219F3],reverseM.00401
00401020 . C705 R5214000 00 MOU DWORD PTR DS:[40219F3],0
00401025 . C705 R5214000 00 MOU DWORD PTR DS:[40219F3],0
0040102E . E8 95214000 00 MOU EQX,DWORD PTR DS:[40219F3],0
00401033 . E8 H7214000 MOU EQX,DWORD PTR DS:[40219F3],ERX
00401038 . E8 04 PUSH EDX
00401041 . E8 3F030000 CALL QWORD PTR DS:[4021H83],ERX
00401045 . E8 A8214000 MOU DWORD PTR DS:[4021AF3],ERX
00401048 . E8 007F0000 PUSH 7F00
00401052 . E8 C9020000 MOU 0
00401055 . E8 RF214000 CALL QWORD PTR DS:[4021AF3],ERX
0040105C . E8 00 PUSH 0
0040105E . E8 6F214000 PUSH reverseM.0040216F
00401063 . E8 03 PUSH 3
00401067 . E8 00 PUSH 0
00401069 . E8 00000000 PUSH 0
0040106E . E8 79204000 PUSH 79204000
00401073 . E8 00020000 PUSH 0
00401078 . 83F8 FF JNZ SHORT reverseM.0040109A
0040107D . E8 00 PUSH 0
0040107F . E8 00204000 PUSH reverseM.00402000
00401084 . E8 17204000 PUSH reverseM.00402017
00401089 . E8 00 PUSH 0
0040108B . E8 D7020000 CALL QWORD PTR DS:[4021H83],ERX
00401090 . E8 24020000 CALL QWORD PTR DS:[4021H83],ERX
00401095 . E8 83010000 CALL QWORD PTR DS:[4021H83],ERX
0040109A > E8 00 PUSH 0
0040109C . E8 73214000 PUSH reverseM.00402173
004010A1 . E8 46 PUSH 46

```

Hehe, ain't that great?  
We have jumped the badboy

Let's execute the JNZ and continue the journey in outer space LOL

Hehe, ain't that great ?

We have jumped the badboy

JNZ를 실행하고 우리들의 공간으로 여행을 떠나자.

헤헤, 대단하지 않냐?

우리는 badboy를 jump했다.

Let me resume so far what happened.

We have executed some code from the ReverseMe till the point where the program verifies the existence of a file.

어떤 일이 일어났는지 설명할게.

ReverseMe에서 파일이 존재하는지 검증하기 위해 이 지점까지 실행됐다.

I have changed a flag so that the program thinks this file was found. Thus, the ReverseMe does not jump to the badboy but continues its normal execution.

나는 flag를 바꿨다. 그래서 program은 file을 '찾았다'로 생각한다. 이렇게 하여, ReverseMe는 badboy로 jump하지 않는다. 그러나 일반적인 실행을 계속해보자.

- > file not found == BadBoy
- > file found == continue execution
- > 파일을 못 찾으면 == badboy
- > 파일을 찾으면 == 실행을 하자.

Scroll up for better view

스크롤을 올려보는 게 좋다.

```

00401095 > EB 83010000 JMP reverseM.00401210
00401096 > 6A 01 PUSH 01
0040109C . 69 73214000 PUSH reverseM.00402173
004010A1 . 6A 46 PUSH 46
004010A3 . 68 18214000 PUSH reverseM.0040211A
004010A8 . 50 PUSH ERX
004010A9 . EB 2F020000 CALL <JMP.&KERNEL32.ReadFile>
004010B0 > 85C8 TEST ERX,ERX
004010B1 > 48 02 JNP SHORT reverseM.00401084
004010B2 > EB 43 JNP SHORT reverseM.004010F7
004010B4 > 33D8 XOR EBX,EBX
004010B6 . 33F6 XOR ESI,ESI
004010B8 . 83D0 73214000 10 CMP DWORD PTR DS:[402173],10
004010B9 > 7C 36 JL SHORT reverseM.004010F7
004010C1 > 8403 18214000 MOV AL,BYTE PTR DS:[EBBX+40211A]
004010C3 . 3C 00 CMP AL,00
004010C5 > 75 09 JNP SHORT reverseM.00401003
004010C7 > 3C 47 INC EBX
004010CD > 75 01 JNP SHORT reverseM.00401000
004010D0 . 46 CMP AL,46
004010D2 > 43 INC ESI
004010D3 > EB EE JMP SHORT reverseM.004010C1
004010D5 > 83FE 00 CMP ESI,8
004010D6 > 75 1F JNP SHORT reverseM.004010F7
004010D8 > E9 29001000 JMP reverseM.00401285
004010D9 . 00 DB 00
004010DE . 00 DD 00000000

```

Mmmm, a whole bunch of jumps and conditional jumps ahead ... and then nothing...

모든 가지의 jump와 조건 jump들... 어디로 jump해야 할지 안 보여. 살펴보자

Perhaps you have already understood that indeed again, we will need to dig in the API.

Let's go.

That clarifies a lot, doesn't it?

Gogo!

어쩌면 너는 이미 이해 했을 것이다. 우리는 API를 파헤치는 게 필요하다.

많은 것이 명확하다. 맞지 않느냐?

고고!

In short: ReadFile tries to read our Keyfile.dat for a certain number of bytes which it puts in a buffer at a certain address if successfull. Understood?

요약하면: ReadFile은 성공하면 Keyfile.dat를 정확한 byte를 읽어 Buffer의 정확한 주소에 넣는다.

Keyfile.dat was not found by CreateFile of course, so the info here is missing (FFFFFFF instead of the value)

CreateFile에 의해서 Keyfile.dat는 찾지 못했다. 그래서 정보는 못 찾았다.(FFFF FFFF 값을 대신해서 넣었다)

```

00401095 > EB 83010000 JMP reverseM.00401210
00401096 > 6A 01 PUSH 01
0040109C . 69 73214000 PUSH reverseM.00402173
004010A1 . 6A 46 PUSH 46
004010A3 . 68 18214000 PUSH reverseM.0040211A
004010A8 . 50 PUSH ERX
004010A9 . EB 2F020000 CALL <JMP.&KERNEL32.ReadFile>
004010B0 > 85C8 TEST ERX,ERX
004010B1 > 48 02 JNP SHORT reverseM.00401084
004010B2 > EB 43 JNP SHORT reverseM.004010F7
004010B4 > 33D8 XOR EBX,EBX
004010B6 . 33F6 XOR ESI,ESI
004010B8 . 83D0 73214000 10 CMP DWORD PTR DS:[402173],10
004010B9 > 7C 36 JL SHORT reverseM.004010F7
004010C1 > 8403 18214000 MOV AL,BYTE PTR DS:[EBBX+40211A]
004010C3 . 3C 00 CMP AL,00
004010C5 > 75 09 JNP SHORT reverseM.00401003
004010C7 > 3C 47 INC EBX
004010CD > 75 01 JNP SHORT reverseM.00401000
004010D0 . 46 CMP AL,46
004010D2 > 43 INC ESI
004010D3 > EB EE JMP SHORT reverseM.004010C1
004010D5 > 83FE 00 CMP ESI,8
004010D6 > 75 1F JNP SHORT reverseM.004010F7
004010D8 > E9 29001000 JMP reverseM.00401285
004010D9 . 00 DB 00
004010DE . 00 DD 00000000

```

ReadFile was going to read 46h (==70d) bytes

at 402173 (would normally have been filled by CreateFile)

to place them in a buffer at 40211A

ReadFile은 46h를 읽기 위해 402173에서 실행된다. (CreateFile에 의해서 채워진다)

그것들은 40211A Buffer에 넣어진다.

I hope it's clear that this is again bad news. The following conditional jumps will lead to badboys again, don't you think?

Let's take a look.

그것은 나쁜 소식이라는 게 다시 한 번 명확해진다. 조건 jump를 따라가면 다시 badboy를 본다. 그렇게 생각하지 않나?

#### ASSEMBLY INFO: TEST

Syntax:

TEST	operand1	operand2
------	----------	----------

This instruction is in 99% of all cases used for "TEST EAX, EAX". It performs a Logical AND (see AND instruction) but does not save the values. It only sets the Z-Flag, when EAX is 0 or clears it, when EAX is not 0. The O/C flags are always cleared.

이 명령어는 99% "TEST EAX, EAX" 다. 그것은 논리 AND를 실행한다. 그러나 값을 저장하지 않는다. Z-Flag는 EAX가 0일 때 set되거나, EAX가 0이 아닐 때 clear 된다. The O/C flag는 항상 지워진다.

#### ASSEMBLY INFO: AND (Logical And)

Syntax:

AND	dest	src
-----	------	-----

The AND instruction uses a logical AND on two values. This instruction \*will\* clear the O-Flag and the C-Flag and can set the Z-Flag. To understand AND better, consider those two binary values:

AND 명령어는 논리 AND에 두 가지 값으로 사용된다. 이 명령어는 O-Flag와 C-Flag를 clear 하고 Z-Flag를 set한다. AND 이해했으리라 생각한다. 두 가지 binary 값만 고려한다.

1001010110  
AND 0101001101  
0001000100



If you AND them, the result is 0001000100 When two 1 stand below each other, the result is of this bit is 1, if not: The result is 0.

You can use calc.exe to calculate AND

네가 AND 할 때, 결과값은 0001000100 이다. 두 값이 각각 1이면 결과값은 1로 set되고 아니면 결과값은 0 이다.

계산기를 사용할 때 AND를 계산할 수 있다.

After this TEST EAX, EAX ...

그리고 TEST EAX, EAX

```

004010A3 . 68 1A214000 PUSH reverseM.0040211A
004010A8 . 50 PUSH EAX
004010A9 . E8 2F020000 CALL <JMP.&KERNEL32.ReadFile>
004010AE . 85C0 TEST EAX,EAX
004010B0 . ^ 75 02 JNZ SHORT reverseM.004010B4
004010B2 . ^ EB 43 JMP SHORT reverseM.004010B7
004010B4 > 33DB XOR EBX,EBX
004010B6 . 33F6 XOR ESI,ESI
004010B8 . 833D 73214000 10 CMP DWORD PTR DS:[EAX+40211A]
004010BF . ^ 7C 36 JL SHORT reverseM.004010B8
004010C1 > 8A83 1A214000 MOV AL,BYTE PTR DS:[EAX+40211A]
004010C7 . 3C 00 CMP AL,0
004010C9 . ^ 74 08 JE SHORT reverseM.004010C3
004010CB . 3C 47 CMP AL,47
004010CD . ^ 75 01 JNZ SHORT reverseM.004010C1
004010CF . 46 INC ESI
004010D0 . ^ 43 INC EBX
004010D1 > EB EE JMP SHORT reverseM.004010D3
004010D3 . 83FE 08 CMP ESI,ESI
004010D6 . ^ 7C 1F JL SHORT reverseM.004010D7
004010D8 . ^ E9 28010000 JMP reverseM.004010D7
004010DD . 00 DB 00
004010DE . 00000000 DD 00000000
004010E2 . 00 DB 00
004010E3 . 00 DB 00
004010E4 . 00 DB 00
004010E5 . 00 DB 00
004010E6 . 00 DB 00
004010E7 . 00 DB 00
004010E8 . 00 DB 00
004010E9 . 00 DB 00
004010EA . 00 DB 00
004010EB . 00 DB 00
004010EC . 00 DB 00
004010ED . 00 DB 00
EAX=00000000

```

```

004010B9 . 6A 00 PUSH 0
004010BB . E8 D7020000 CALL <JMP.&USER32.MessageBoxA>
004010BD . E8 24020000 CALL <JMP.&KERNEL32.ExitProcess>
004010B5 . ^ E9 83010000 JNP reverseM.00401210
004010B9 > 6A 00 PUSH 0
004010B9C . 68 73214000 PUSH reverseM.00402173
004010B9D . 6A 46 PUSH 46
004010B9E . 68 1A214000 PUSH reverseM.0040211A
004010B9F . 50 PUSH EAX
004010B9G . E8 2F020000 CALL <JMP.&KERNEL32.ReadFile>
004010AE . 85C0 TEST EAX,EAX
004010B0 . ^ 75 02 JNZ SHORT reverseM.004010B4
004010B2 . ^ EB 43 JMP SHORT reverseM.004010F7
004010B4 > 33DB XOR EBX,EBX
004010B6 . 33F6 XOR ESI,ESI
004010B8 . 833D 73214000 10 CMP DWORD PTR DS:[EAX+40211A]
004010BFB . ^ 70 01 JU SHORT reverseM.004010B9
004010C1 > 33F7 HOU AL,BYT
004010C7 . ^ 74 08 CMP AL,0
004010C9 . 3C 47 CMP AL,47
004010CD . ^ 75 01 JNZ SHORT reverseM.004010C1
004010CF . 46 INC ESI
004010D0 . ^ 43 INC EBX
004010D1 > EB EE JMP SHORT reverseM.004010C1
004010D3 . 83FE 08 CMP ESI,ESI
004010D6 . ^ 7C 1F JL SHORT reverseM.004010F7
004010D8 . ^ E9 28010000 JMP reverseM.004010F7
004010DD . 00 DB 00
004010DE . 00000000 DD 00000000
004010E2 . 00 DB 00
004010E3 . 00 DB 00
004010E4 . 00 DB 00
004010E5 . 00 DB 00
004010E6 . 00 DB 00
004010E7 . 00 DB 00
004010E8 . 00 DB 00
004010E9 . 00 DB 00
004010EA . 00 DB 00
004010EB . 00 DB 00
004010EC . 00 DB 00
004010ED . 00 DB 00
Jump is NOT taken.
004010B4=reverseM.004010B4

```

...comes JUMP if NOT zero, so, we are not going to jump

because EAX "IS" zero (the Z-flag is set)

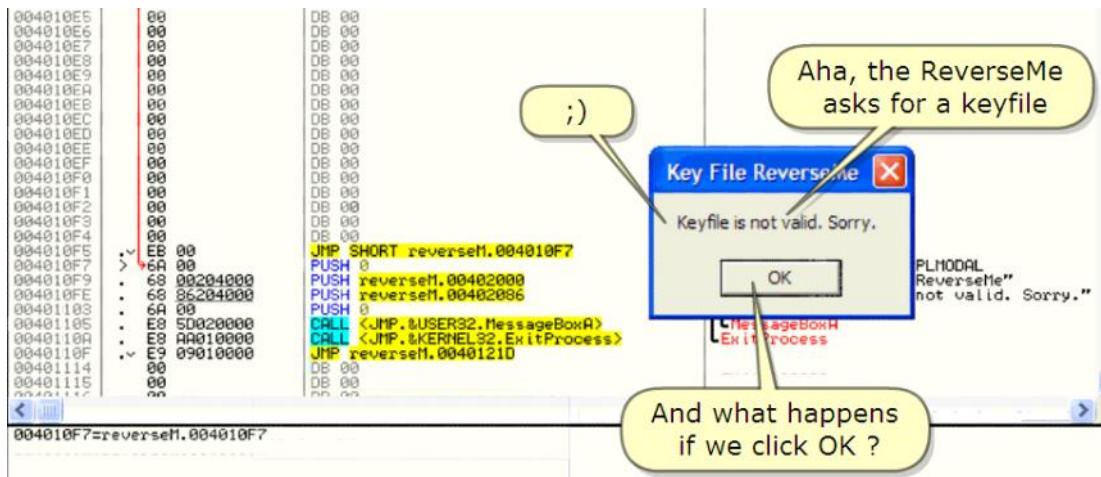
Indeed.

If we don't jump now, then next is an unconditional jump. Let's see where that leads to ...

Zero가 아닐 때 jump 한다. 그래서 우리는 jump하지 않는다.

왜냐하면 EAX가 0이기 때문이다.(Z-Flag가 set 됐다)

우리는 현재 jump하지 않는다. 그리고 다음은 무조건 jump다. 따라가면 무엇이 있는지 봐라.



Woops! Follow the jump. Scroll down.

Right, we jump to...

...another badboy !!!

Let's run and have a look at it

웁스! Jump를 따라가 보자. Scroll 내려라.

맞아, 여기로 jump한다.

다른 badboy로

실행 해보고 봐라.

;)

Aha, the ReverseMe asks for a keyfile

And what happens if we click OK?

Right, the program exits :)

아하, ReverseMe는 keyfile을 묻는다.

그리고 우리가 click하면 무슨 일이 발생할까?

맞다, program은 종료된다.

Suddenly, we are back here again.

Indeed, to reduce the size of this movie, I have cut from this movie...

...the restarting...

...and coming back right here where we jumped to the 2nd badboy (in the same way as before) changing the Z-flag in the first JNZ.

갑자기, 다시 여기로 왔다.

정말, movie의 size를 줄이기 위해, 이 movie에서 restart과 이 곳으로 되 돌아오는 것을 삭제했다. 이 곳은 badboy를 2개 jump했다.(전에 했던 같은 방법으로) 첫 번째 JNZ에서 Z-Flag를 바꾼다.

This time however, let's avoid jumping to the badboy again. I suppose you have understood how to do it ???

badboy로 다시 jump하는 것을 회피하자. 너는 어떻게 하는지 이해 했을 거다.

```

00401090 . E8 24020000 CALL <JMP.&KERNEL32.ExitProcess>
00401095 .> E9 83010000 JMP reverseM.0040121D
0040109A .> 6A 00
0040109C .> 68 73214000
004010A1 .> 6A 46
004010A3 .> 68 1B214000
004010A8 .> 50
004010A9 .> E8 2F020000 CALL <JMP.&KERNEL32.ReadFile>
004010AE .> 85C0 TEST ERX,ERX
004010B0 .> 75 02 JNZ SHORT reverseM.00401084
004010B2 .> EB 43 JMP SHORT reverseM.004010F7
004010B4 .> 33DB XOR EBK,EBK
004010B6 .> 33F6 XOR ESI,ESI
004010B8 .> 833D 73214000 10 CMP DWORD PTR DS:[402173],10
004010B9 .> 7C 36 JE SHORT reverseM.004010F7
004010C1 .> 8903 1B214000 MOU AL,BYTE PTR DS:[EBX+40211A]
004010C2 .> 9C 00 CMP AL,?
004010C3 .> 3B C7 JNE SHORT reverseM.00401003
004010C5 .> 3C 47 CMP CL,?
004010C6 .> 75 01 JNZ SHORT reverseM.00401000
004010C7 .> 46 INC ESI

```

ExitProcess

pOverlapped = NULL  
 pBytesRead = reverseM.00402173  
 BytesToRead = 46 (70.)  
 Buffer = reverseM.0040211A  
 hFile = NULL  
 ReadFile

Exactly

Exactly

정확하다.

```

00401088 . E9 50 2F020000 CALL <JMP.&KERNEL32.ReadFile>
00401089 .> E8 2F020000 TEST ERX,ERX
0040108E .> 85C0
0040108F .> 75 02 JNZ SHORT reverseM.00401084
00401090 .> EB 43 JMP SHORT reverseM.004010F7
00401092 .> 33DB XOR EBX,EBX
00401094 .> 33F6 XOR ESI,ESI
00401096 .> 833D 73214000 10 CMP DWORD PTR DS:[402173],10
00401098 .> 7C 36 JE SHORT reverseM.004010F7
0040109A .> 8903 1B214000 MOU AL,BYTE PTR DS:[EBX+40211A]
0040109C .> 9C 00 CMP AL,?
0040109D .> 3B C7 JNE SHORT reverseM.00401003
0040109E .> 3C 47 CMP CL,?
0040109F .> 75 01 JNZ SHORT reverseM.00401000
004010A0 .> 46 INC EBX
004010A1 .> EB EE JMP SHORT reverseM.004010C1
004010A2 .> 83FE 08 CMP ESI,?
004010A4 .> 7C 1F JE SHORT reverseM.004010F7
004010A5 .> E9 28010000 JMP reverseM.00401205
004010A6 .> 00 DB 00
004010A7 .> 00 DD 00000000
004010A8 .> 00 DB 00
004010A9 .> 00 DB 00
004010AA .> 00 DB 00

```

And the magic has happened again.

And the magic has happened again.

Magic이 다시 일어났었다.

Let's resume : I have changed the Z-flag once more to make the ReverseMe think it has successfully read the file.

--> the ReverseMe will continue its normal execution as if all were ok !!!

재개하자 : 나는 ReverseMe가 성공적으로 파일을 읽기 위해 Z-Flag를 한 번 이상 바꿨다.

--> 우리가 ok 됐을 때 ReverseMe는 일반적인 방법으로 실행 된다.

Remark : basically, what I'm doing, is guiding the execution of the ReverseMe as if everything it verifies is simply.... ok...

근본적으로, 내가 무슨 일을 했냐면? 모든 것을 간단히 검사할 때 까지 ReverseMe를 인도했다.

ASSEMBLY INFO : XOR

Syntax:

XOR	dest	src
-----	------	-----

The XOR instruction connects two values using logical exclusive OR

XOR 명령어는 두 값을 배타적 논리 OR로 연결한다.

This instruction clears the O-Flag and the C-Flag and can set the Z-Flag. To understand XOR better, consider those two binary values:

이 명령어는 O-Flag와 C-Flag를 clear 하고, Z-Flag를 set한다. 두 값을 고려할 때 XOR을 좀 더 많이 이해했을 거라고 생각한다.

XOR 0101001101

1100011011



If you XOR them, the result is 1100011011

XOR 결과는 1100011011 이다.

When two bits on top of each other are equal, the resulting bit is 0. Else the resulting bit is 1. You can use calc.exe to calculate XOR. The most often seen use of XOR is "XOR EAX, EAX".

두 값이 서로 같을 때, 결과 bit는 0이고, 다른 것일 때 bit는 1이다. 계산기로 XOR를 사용할 수 있다. 자주 "XOR EAX, EAX" 사용된 것을 본다.

This will set EAX to 0, because when you XOR a value with itself, the result is always 0. I hope you understand why, else write down a value on paper and try ;) However, the result here is always zero. XOR REG, REG is often used to make sure the result is zero.

EAX가 0으로 set된다. 왜냐하면 같은 값을 XOR 할 때 결과값은 항상 0이다. 나는 네가 왜 이러한지 이해 했을 거라 생각한다. 다른 방식으로 종이에 값을 적고 도전해봐 ;) 여기 결과는 항상 0이다. XOR REG, REG는 결과값을 0으로 만들기 위해 자주 사용된다.

Aha, is this the jump to the goodboy ?

Scrool down

아하, 이 jump는 goodboy로 가나?

Scroll 내려봐.

Oh no... again to the badboy!!!

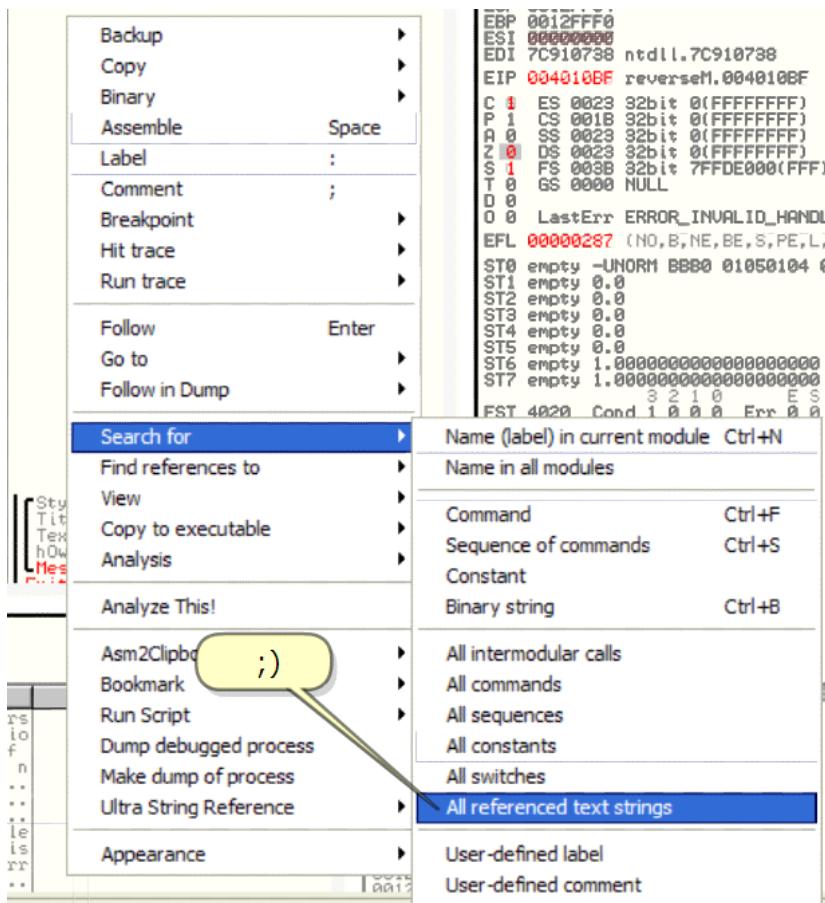
Now, I told you already I'm guiding the program to the goodboy. Deliberately, I have not yet looked at the strings in the ReverseMe not to give it all away from the beginning.

Let's do that now.

오 노!! 다시 badboy로 간다.

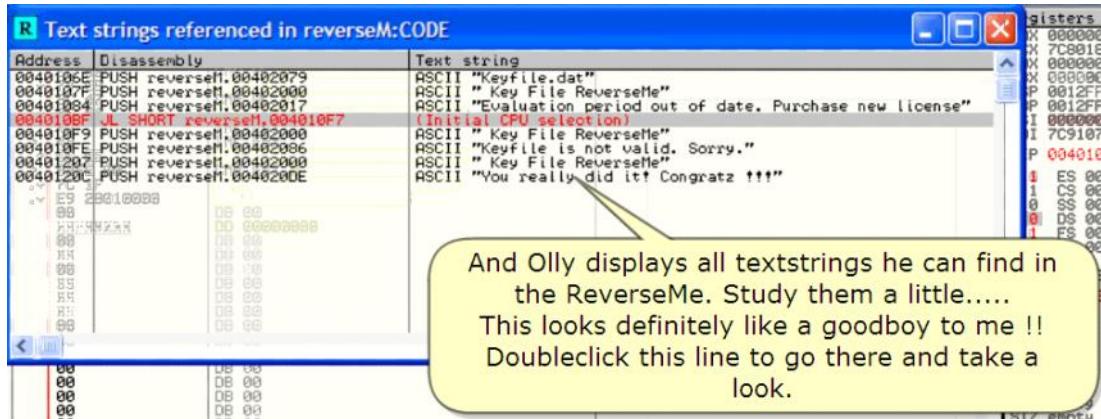
너에게 이미 프로그램을 goodboy로 인도 한다고 말했다. 일부러 goodboy를 안 봤다. 이 ReverseMe에서 처음 시작할 때부터 문자열을 주지 않는다.

이제 다시 해보자.



Rightclick

;) ;)



And Olly displays all textstrings he can find in the ReverseMe. Study them a little...

This looks definitely like a goodboy to me !!

Doubleclick this line to go there and take a look.

Doubleclicking this line brings you to this place in the program. Let's go see it there...

Olly는 찾은 모든 textstring을 보여준다. 이것들을 공부하자.

이것은 명확히 goodboy로 가는 것을 보여준다.

0| line을 Doubleclick 하고 살펴보자.

Doubleclick은 프로그램에서 이 line이 위치해 있는 곳을 너에게 가져온다. 이제 보자.

This is what it looks like in the code.  
Mmmm, see the goodboy here.  
It means that, if we can guide the program to execute this piece of code  
.... that we solved the ReverseMe.

It means that, if we can guide the program to execute this piece of code

... that we solved the ReverseMe.

이것이 무슨 code인지 봐라.

Goodboy가 여기에 있다.

이것은 약간의 조각난 코드를 실행하기 위해 프로그램을 인도 한다.

우리는 ReverseMe 문제를 해결했다.

Of course, you can also see here where the jump comes from

Jump from 00401008

But how to get here?

Easy : just follow the jumps.

This sign indicates a jump arriving here.

Let's see from where it comes. Scroll up.

그러나, 우리는 어떻게 이곳으로 오냐?

쉽다. Jump를 따라가자.

이 sign은 jump가 도착한 곳을 나타낸다.

어디서부터 왔는지 보자. Scroll 올려.

And simply follow the jump :)

Of course, you can also see here where the jump comes from

간단히 jump를 따라가자.

물론, 우리는 jump가 어디에서 시작됐는지 볼 수 있다.

C CPU - main thread, module reverseM

0040109C	. 68 73214000	PUSH reverseM.00402173
004010A1	. 6A 46	PUSH 46
004010A3	. 68 1A214000	PUSH reverseM.0040211A
004010A8	. 50	PUSH EAX
004010A9	. E8 2F020000	CALL <JMP.&KERNEL32.ReadFile>
004010AE	. 85C0	TEST EAX, EAX
004010B0	.> 75 02	JNZ SHORT reverseM.004010B4
004010B2	.> EB 43	JMP SHORT reverseM.004010F7
004010B4	> 33DB	XOR EBX, EBX
004010B6	. 33F6	XOR ESI, ESI
004010B8	. 833D 73214000 10	CMP DWORD PTR DS:[402173], 10
004010BF	.> 7C 36	JL SHORT reverseM.004010F7
004010C1	> 8A83 1A214000	MOV AL,BYTE PTR DS:[EBX+40211A]
004010C7	. 3C 00	CMP AL, 0
004010C9	.> 74 08	JE SHORT reverseM.004010D3
004010CB	. 3C 47	CMP AL, 47
004010CD	.> 75 01	JNZ SHORT reverseM.004010D0
004010CF	. 46	INC ESI
004010D0	.> 43	INC EBX
004010D1	.^ EB EE	JMP SHORT reverseM.004010C1
004010D3	> 83FE 08	CMP ESI, 8
004010D6	.> 7C 1F	JL SHORT reverseM.004010F7
004010D8	.> E9 28010000	JMP reverseM.00401205
004010D9	. 00	DB 00
004010DE	. 00000000	DB 00
004010E2	. 00	DB 00
004010E3	. 00	DB 00
004010E4	. 00	DB 00
004010E5	. 00	DB 00
004010E6	. 00	DB 00
004010E7	. 00	DB 00
004010E8	. 00	DB 00
004010E9	. 00	DB 00
004010EA	. 00	DB 00
004010EB	. 00	DB 00

So, we need to  
arrive here huh !!!

So, we need to arrive here huh !!!

우리는 여기에 도착할 필요가 있다.

and here is where we are already !!!

그리고 이곳은 이미 우리가 와 본 적이 있다.

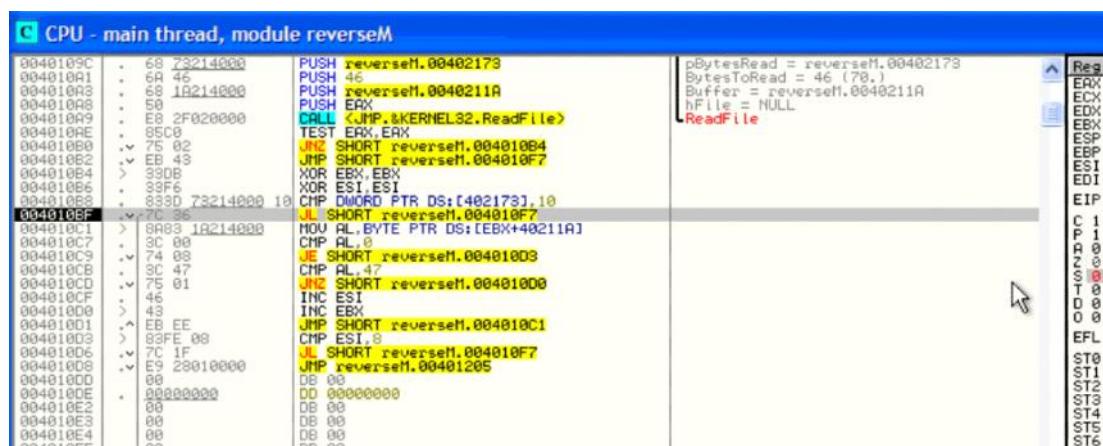
C CPU - main thread, module reverseM

0040109C	. 68 73214000	PUSH reverseM.00402173
004010A1	. 6A 46	PUSH 46
004010A3	. 68 1A214000	PUSH reverseM.0040211A
004010A8	. 50	PUSH EAX
004010A9	. E8 2F020000	CALL <JMP.&KERNEL32.ReadFile>
004010AE	. 85C0	TEST EAX, EAX
004010B0	.> 75 02	JNZ SHORT reverseM.004010B4
004010B2	.> EB 43	JMP SHORT reverseM.004010F7
004010B4	> 33DB	XOR EBX, EBX
004010B6	. 33F6	XOR ESI, ESI
004010B8	. 833D 73214000 10	CMP DWORD PTR DS:[402173], 10
004010BF	.> 7C 36	JL SHORT reverseM.004010F7
004010C1	> 8A83 1A214000	MOV AL,BYTE PTR DS:[EBX+40211A]
004010C7	. 3C 00	CMP AL, 0
004010C9	.> 74 08	JE SHORT reverseM.004010D3
004010CB	. 3C 47	CMP AL, 47
004010CD	.> 75 01	JNZ SHORT reverseM.004010D0
004010CF	. 46	INC ESI
004010D0	.> 43	INC EBX
004010D1	.^ EB EE	JMP SHORT reverseM.004010C1
004010D3	> 83FE 08	CMP ESI, 8
004010D6	.> 7C 1F	JL SHORT reverseM.004010F7
004010D8	.> E9 28010000	JMP reverseM.00401205
004010D9	. 00	DB 00
004010DE	. 00000000	DB 00
004010E2	. 00	DB 00
004010E3	. 00	DB 00
004010E4	. 00	DB 00
004010E5	. 00	DB 00
004010E6	. 00	DB 00
004010E7	. 00	DB 00
004010E8	. 00	DB 00
004010E9	. 00	DB 00
004010EA	. 00	DB 00
004010EB	. 00	DB 00
004010EC	. 00	DB 00
004010ED	. 00	DB 00
004010EE	. 00	DB 00
004010EF	. 00	DB 00
004010F0	. 00	DB 00
004010F1	. 00	DB 00

INFO : the EIP register always holds  
the VA (virtual address) of what is the  
next line of code to execute. If you  
get lost in the code, doubleclick it to  
find your way back to the origin ...

INFO : the EIP register always holds the VA(virtual address) of what is the next line of code to execute. If you get lost in the code, doubleclick it find your way back to the origin ...

EIP register는 항상 다음 실행할 code의 VA를 잡고 있다. 만약에 code 안에서 길을 잃어버렸을 때 doubleclick 하면 너의 original 길로 돌아갈 수 있다.



I suppose you know by now what need to be done ???

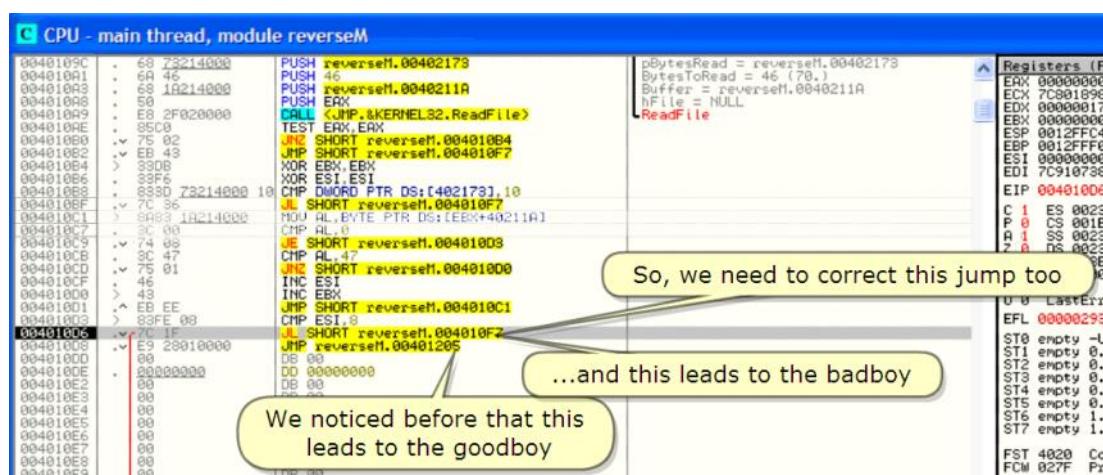
But this time, it's a JL, so do not change the Z-flag but ...

네가 이제 무엇이 우리에게 필요할지 알고 있을 거라 생각한다.

그러나 지금은, JL의 Z-Flag를 바꾸지 말자 그러나

Mmm, ok this JE does not jump to the badboy

이번 JE는 badboy로 jump하지 않는다.



We noticed before that this leads to the goodboy

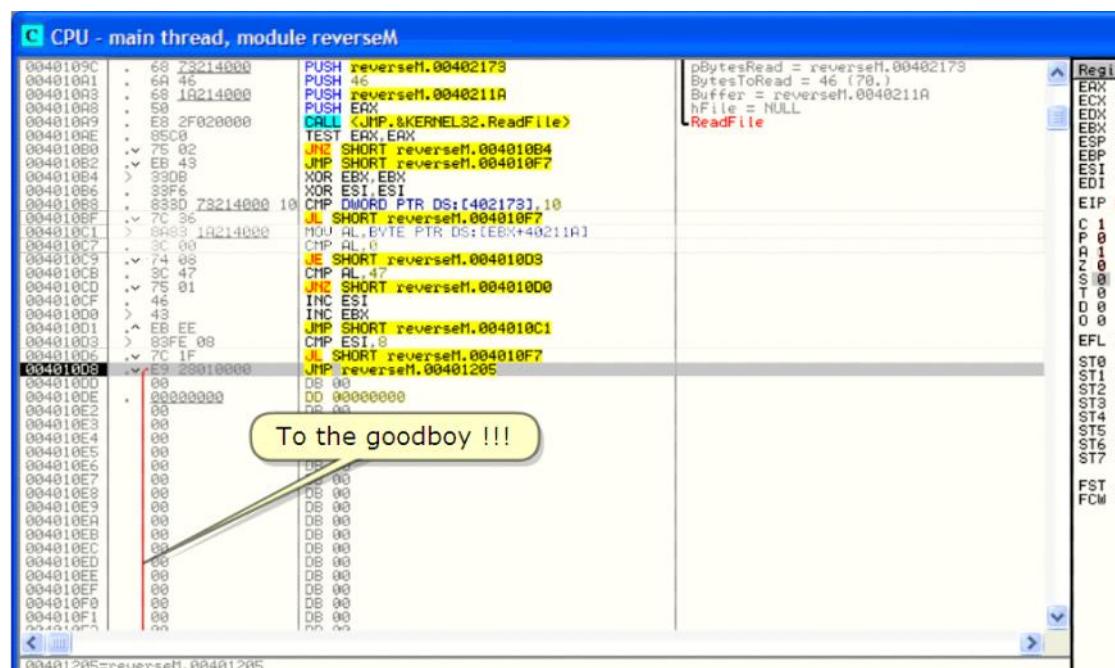
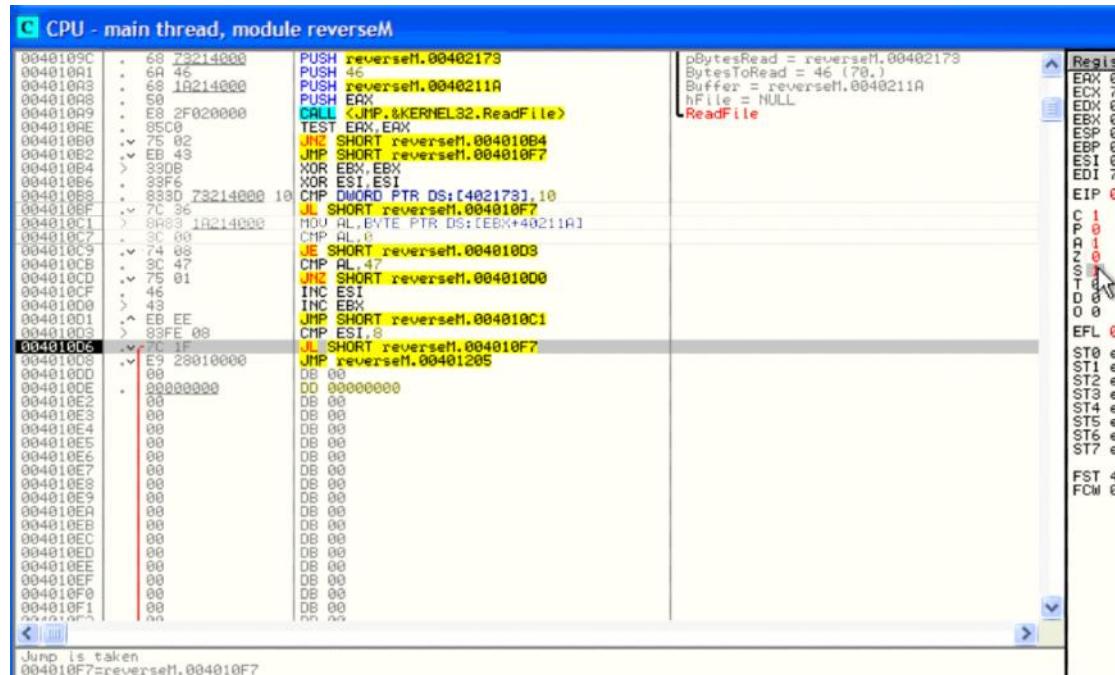
...and this leads to the badboy

So, we need to correct this jump too

이것은 goodboy로 이끄는 것을 안다.

그리고 이것은 badboy로 이끈다.

그래서 우리는 정확하게 jump하기 위해 이 jump가 필요하다.

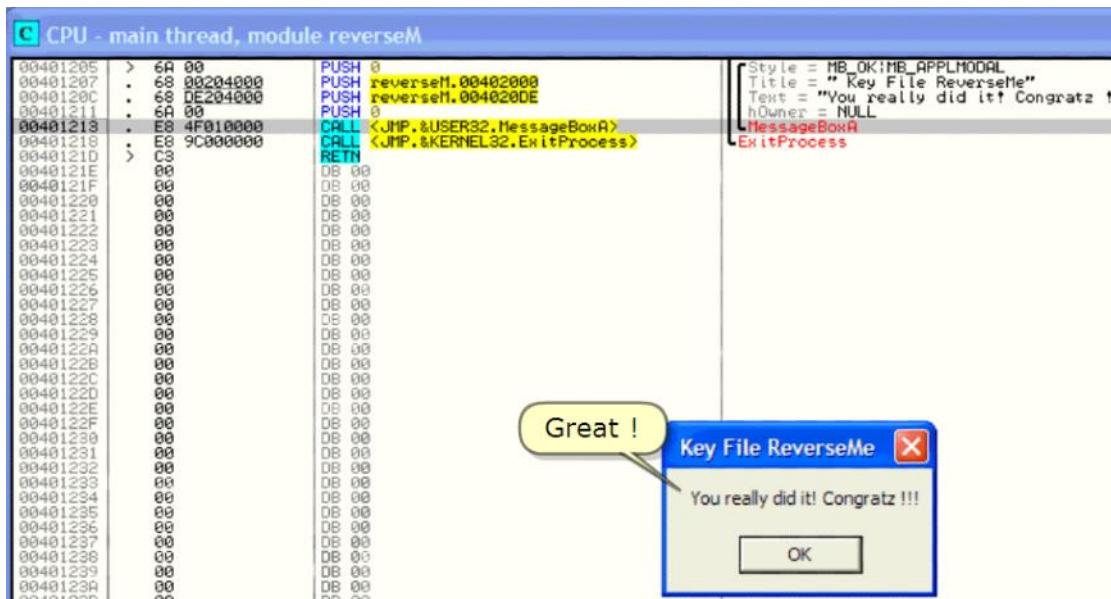


To the goodboy !!!

Goodboy로!

Ok. Continue stepping to see the result...

Ok. 결과를 보기 위해 계속 하자.



Great.

좋아!

BTW, all API have their own specific function in a program. This MessageBoxA for example displays a messagebox. You can clearly see the arguments (parameters) that were pushed on the stack. You can find info on most API's in Win32.hlp. Do that on your own if you want more info.

모든 API는 프로그램에서 자신만의 특별한 function을 가진다. 이 MessageBoxA 는 예를 들면 messagebox를 보여준다. 너는 명확히 우리가 stack에 넣었던 arguments를 볼 수 있다. 너는 대부분의 정보를 Win32.hlp에서 찾을 수 있다. 좀 더 정보를 보고 싶다면 실행하자.

I'm sure you already understand what this API will accomplish :))

물론 너는 이미 이 API가 실행하는 것을 이해했다.

So far so good. The question now is :

how can we finalize this. We are not always there to guide Olly to the goodboy ! Well, I'll show you. So, restart the ReverseMe in Olly.

여태까지는 그런대로 잘됐다. 질문은 :

우리가 어떻게 이 것을 끝내나? 우리는 항상 Olly를 goodboy로 guide 하지 않는다. 나는 너에게 보여준다. 그래서 ReverseMe를 Olly에서 restart 한다.

## 6. Finalizing the patches

I've restarted but removed it from this movie to reduce its size and so we land once again at the EP. Remember the breakpoint that we've set and look what Olly is capable of.

나는 restart 했다. 그러나 movie size를 줄이기 위해 삭제했다. 그리고 우리는 다시 한 번 EP에 도착했다. BP를 기억해라. 그것은 우리가 set할 수 있고 Olly에서 활용하는 것을 봐라.

We press "run" (F9 key)

우리는 F9를 누른다.

C CPU - main thread, module reverseM

```

00401000 5 E8 00          PUSH 0
00401002  . E8 64020000  CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007  . A9 77214000  MOV DWORD PTR DS:[402177],EAX
0040100C  . C705 57214000 03 MOV DWORD PTR DS:[402197],4003
00401015  . C705 5B214000 06 MOV DWORD PTR DS:[40219B],reverseM.004010
00401020  . C705 5F214000 00 MOV DWORD PTR DS:[40219F],0
00401029  . A1 77214000 00 MOU EAX,DWORD PTR DS:[4021A3],0
00401034  . A3 A7214000 04 MOU DWORD PTR DS:[4021A7],EAX
0040103E  . 6A 04          PUSH 4
00401040  . 50             PUSH EAX
00401041  . E8 3F030000  CALL <JMP.&USER32.LoadIconA>
00401046  . A9 AB214000  MOU DWORD PTR DS:[4021AB],EAX
0040104B  . 68 007F0000  PUSH 7F00
00401050  . A9 80          PUSH 0
00401052  . E8 C9020000  CALL <JMP.&USER32.LoadCursorA>
00401054  . 50             PUSH EAX
00401055  ...but here comes
00401056  Olly in action and ...
00401057  . 6A 00          PUSH 0
00401059  . E8 000000C0  PUSH C0000000
00401065  . E8 75204000  PUSH reverseM.00402079
00401073  . E8 00020000  CALL <JMP.&KERNEL32.CreateFileA>
00401075  . 83F8 FF        CMP EBX,-1
00401076  JNZ SHORT reverseM.0040109A
0040107D  . 6A 00          PUSH 0
0040107F  . E8 00204000  PUSH reverseM.00402000
00401084  . 68 17204000  PUSH reverseM.00402017
00401089  . 6A 00          PUSH 0
0040108B  . E8 D705 0000  CALL <JMP.&USER32.MessageBoxA>
00401090  . E8 24020000  CALL <JMP.&KERNEL32.ExitProcess>
00401095  . E9 83010000  JMP reverseM.00401210
00401097  . 85C0          TEST EBX,EAX
004010A0  . 7F 00          TEST EAX,EAX
004010A1  . 50             PUSH EAX

```

...the execution of the code is paused in the BP we set earlier.

...but here comes Olly in action and...

...the execution of the code is paused in the BP we set earlier.

그러나 Olly는 이곳으로 온다. 그리고 실행 코드가 우리가 설정한 BP에서 멈춘다.

So, now we need to change this code permanently. This can easily be done in Olly by assembling another piece of code at this offset and then saving it to file !!! In fact, we change the opcodes. Look carefully. We won't need the BP any further, so I'll first remove it.

그래서 우리는 code를 영원히 바꾸는 것이 필요하다. Olly에서 assembling에 의해 offset에서 다른 code 조각을 매우 쉽게 변경했다. 그리고 파일로 저장해라. 사실은, 우리는 opcodes를 바꿨다. 괜찮다. 우리는 더 이상 BP가 필요하지 않다. 그래서 우리는 첫번째 것을 지운다.

Doubleclick in the code or press <Spacebar> to assemble.

Code 안에서 doubleclick 하거나 Spacebar를 누르면 assemble 한다.

```

00401040  . E8 3F030000  PUSH EBX
00401041  . E8 64020000  CALL <JMP.&USER32.LoadIconA>
00401046  . A9 77214000  MOV DWORD PTR DS:[4021AB],EAX
00401050  . 68 007F0000  PUSH 7F00
00401054  . E8 C9020000  CALL <JMP.&USER32.LoadCursorA>
00401057  . A9 AF214000  MOV DWORD PTR DS:[4021AF],EAX
00401059  . 6A 00          PUSH 0
00401065  . E8 000000C0  PUSH C0000000
00401073  . E8 75204000  PUSH reverseM.00402079
00401075  . E8 00020000  CALL <JMP.&KERNEL32.CreateFileA>
00401077  . 83F8 FF        CMP EBX,-1
00401078  JNZ SHORT reverseM.0040109A
00401080  . 75 1D          PUSH 0
00401084  . 68 00204000  PUSH reverseM.00402000
00401089  . 6A 00          PUSH 0
0040108B  . E8 D705 0000  CALL <JMP.&USER32.MessageBoxA>
00401090  . E8 24020000  CALL <JMP.&KERNEL32.ExitProcess>
00401095  . E9 83010000  JMP reverseM.00401210
00401097  . 85C0          TEST EBX,EAX
004010A0  . 7F 00          TEST EAX,EAX
004010A1  . 50             PUSH EAX

```

Click <Assemble> or press <Enter>

Notice that I have the "Fill with NOPs" box checked. (See further)

Right. Do you understand that the code execution will now always jump passed this first badboy? Well, we need to assemble all the wrong jumps into good ones. Follow carefully.

<Assemble>이나 Enter를 눌러라

나는 "Fill with NOPs"(NOP으로 채워라) box를 chcek 했다. (더 봐라)

맞다. 이제 실행 code가 항상 첫번째 badboy를 jump 해서 피할 것을 이해했어? 우리는 wrong jump를 good으로 assemble 하기 위해 필요하다. 조심히 따라와.

```

0040107D: 6A 00          ; PUSH 0
0040107E: 6A 00          ; PUSH reverseM.00402000
0040107F: 6A 00          ; PUSH reverseM.00402017
00401080: 6A 00          ; PUSH 0
00401081: E8 70200000    ; CALL <JMP.&USER32.MessageBoxW>
00401082: E8 24920000    ; JMP reverseM.00401210
00401083: E8 83010000    ; PUSH 0
00401084: E8 00          ; PUSH reverseM.00402173
00401085: E8 46          ; PUSH 46
00401086: E8 18210000    ; PUSH reverseM.0040211A
00401087: E8 5C 00        ; PUSH EAX
00401088: E8 2F020000    ; CALL <JMP.&KERNEL32.ReadFile>
00401089: EB 00          ; TEST EAX, EAX
0040108A: EB 42          ; JNE reverseM.004010B4
0040108B: EB 43          ; JMP SHORT reverseM.004010F7
0040108C: 33DB            ; XOR EBX, EBX
0040108D: 33F6            ; XOR ESI, ESI
0040108E: 89D0 73214000 18 ; CMP DDWORD PTR DS:[402173], 18
0040108F: 7C 36          ; JL SHORT reverseM.004010F7
00401090: 3C 00          ; MOV AL, BYTE PTR DS:[EBX+40211A]
00401091: 74 08          ; JE SHORT reverseM.00401003
00401092: 3C 47          ; CMP AL, 47
00401093: 74 01          ; JE SHORT reverseM.00401008
00401094: 46              ; INC ESI
00401095: EB 43          ; INC EBX
00401096: EB EE 00        ; JMP SHORT reverseM.004010C1
00401097: 7C 1F          ; JL SHORT reverseM.004010F7
00401098: E9 28010000    ; JMP reverseM.00401205

```

Note: I'm showing it here in the long way. Of course we could change the destination jump here and jump right away to the goodboy message.

Remark: of course it isn't always this simple ;)

Note: 나는 여기에서 다른 방법으로 보여준다. 물론 우리는 도착 jump를 바꿀 수 있고 그리고 jump를 옮은 방법으로 goodboy message로 보낸다.

Remark: 물론 이렇게 항상 간단하지는 않다.

Remember that we may NOT jump here.

So, this code needs another treatment.

NOP == do nothing :)

여기에서 jump하는 것이 아니라는 것을 기억해.

그래서, code는 다른 치료가 필요하다.

NOP == 아무것도 안한다 :)

INFO:

There are always many different ways to assemble patches that achieve the same. But we'll see more of this later. As an example, I could also assemble the same JL but jumping to the next line instead of NOP'ing the complete line. So like:

JL SHORT reverseM. 004010C1

All this will become clearer as we continue.

항상 같은 결과를 만드는 assemble patch하는 다른 방법이 많다. 그러나 우리는 이것에 대해서 나중에 좀 더 보겠다. 이 예제에서는, 나는 JL과 같이 assemble 할 수 있다. 그러나 다음 line으로 넘어가기 위해 NOP'ing을 대신 쓰겠다. 이렇게

JL SHORT reverseMe. 004010C1

이것은 우리가 진행하면 명확해 진다.

```

00401095: E9 83010000    ; JMP reverseM.00401210
00401096: 6A 00          ; PUSH 0
00401097: 6A 00          ; PUSH reverseM.00402173
00401098: 6A 00          ; PUSH reverseM.0040211A
00401099: 50              ; PUSH EAX
0040109A: E8 00          ; PUSH 0
0040109B: E8 46          ; PUSH 46
0040109C: 89D0 73214000 18 ; CMP DDWORD PTR DS:[402173], 18
0040109D: 7C 36          ; JL SHORT reverseM.004010F7
0040109E: 3C 00          ; MOV AL, BYTE PTR DS:[EBX+40211A]
0040109F: 74 08          ; JE SHORT reverseM.00401003
004010A0: 3C 47          ; CMP AL, 47
004010A1: 74 01          ; JE SHORT reverseM.00401008
004010A2: 46              ; INC ESI
004010A3: 46              ; INC EBX
004010A4: E8 43          ; JMP SHORT reverseM.004010C1
004010A5: EB EE 00        ; JMP SHORT reverseM.004010C1

```

The checking of the box will assure that ...

... both opcodes are NOPed !!!

Checking box는 opcode들을 NOPed로 된다는 것을 확인한다.

Yep. If not, an "unwanted" opcode would remain.

예, opcode는 원치 않는 상태의 opcode로 남는다.

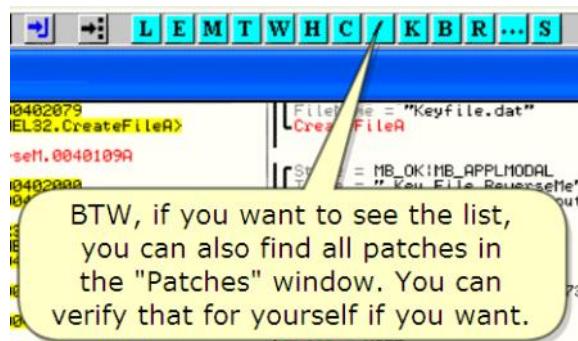
And remember, this code jumps to the goodboy. Now, we still need to save the changes.

Scroll up.

그리고 기억해라, 이 code는 goodboy로 jump한다. 이제, 우리는 바뀐 것을 저장해야 한다.  
스크롤 올려봐.

Here, we can see all the patches we have made.

우리는 우리가 만든 모든 patch를 볼 수 있다.



BTW, if you want to see the list, you can also find all patches in the "Patches" window.

You can verify that for yourself if you want.

만약에 이 list를 보기 원한다면, 너는 모든 patch를 "Patches" window에서 찾을 수 있다.

네가 원한다면 네가 검증할 수 있다.

Registers (FPU)

ERX 00000000	ECX 7C801898 kernel32.7C801898
EDX 00000000	EBX 00000000
ESP 0012FFC4	EBP 0012FFB0
ESI 00000000	EDI 7C910738 ntdll.7C910738
EIP 00401205 reverseM.00401205	

Backup 173

Copy

Binary

Undo selection Alt+BkSp

Assemble Space

Label :

Comment ;

Breakpoint

Hit trace

Run trace

Follow Enter

New origin here Ctrl+Gray =

Go to

Follow in Dump

Search for

Find references to

View

Copy to executable

Selection

All modifications

Either way will do

Either way will do

어느 쪽이든지

;

Saving under a different name !!!

다른 이름으로 저장해라.

Let's resume what we've done so far.

First, we have found what needs to be done to guide the code to run to the goodboy.

Then we have permanently changed and saved ReverseMe to do that. (=patching)

So, all that rests now is ...

지금까지 우리가 했던 것을 요약하자.

먼저, 우리는 code를 guide해서 goodboy로 실행하는 것을 찾았다.

우리는 영원히 바꿨다. 그리고 ReverseMe를 patch하고 저장했다.

그래서, 우리는 재시작 하자.

...testing the saved ReverseMe

저장된 ReverseMe를 test 하자.

## 7. Testing the patched ReverseMe



Indeed!!! You really did it.

Congratulations patching your first ReverseMe!

오!! 정말로 해냈어요.

너의 첫번째 ReverseMe를 patch 한 것을 출하해요.

In reversing, there are always different solutions to a problem.

See me back in part 2 in this series to solves this ReverseMe the proper way :)

리버싱에서 문제를 해결하는데 항상 다른 방법이 있다.

나는 part 2에서 적절한 방법으로 ReverseMe를 해결하기 위해 돌아오겠다.

I hope you understood everything fine and I also hope someone somewhere learned something from this.

See me back in part 2 in this series ;)

나는 네가 모두 이해했기를 그리고 누구든지 어디에서 이것에서 무엇을 배웠으면 하고 희망 한다.

나는 이 series의 Part 2에서 돌아오겠다.

The other parts are available at

다른 parts는 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 tutorial에 시간을 투자해준 너에게 감사한다.

lena151 (2006, updated 2007)