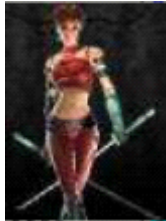


02.Keyfiling the reverseme + assembler

2012년 1월 16일 월요일

오전 6:15

Lena Reversing



02.Keyfiling the reverseme + assembler

번역자 : re4lfl0w / re4lfl0w@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 431.322.12을 위반하고 당신이 우리의 ISP들 또는 사람들을 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 다른사람들을 기소할 수 없습니다.

번역 주)이 부분 번역이 제일 어려웠습니다. 이해도 잘 안되고.

Hello everybody.

안녕 모두들.

Welcome to this Part2 in my series about reversing for newbies/beginners.

나의 reversing 초보자를 위한 series Part 2에 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)

2. Tools and Target

Throughout most of these tutorials, you will use Ollydebug and... your brain. The first can be obtained for free at

대부분의 tutorial을 통해, 너는 Olly와 너의 두뇌를 사용할 것이다.

첫번째로 여기에서 무료로 얻을 수 있다.

<http://www.ollydbg.de>

...again, the second is your responsibility ;)

2번째는 너의 책임감이 필요하다.

For your convenience, I included the target (ReverseMe) in this package. In this package is also included a file "Keyfile.dat".

너의 편리를 위해, 이 package에 ReverseMe를 포함했다. 또한 "Keyfile.dat"도 포함되어 있다.

Do not extract it to the same directory as the ReverseMe if you want to try this tutorial yourself. (Practice is the best way to learn).

이 tutorial을 혼자서 시도할 때 같은 디렉토리에 풀 수 없다. (연습이 배울 때 가장 좋다)

3. The use of plugins

Study of the target is always extremely important. However, we did that for this target already in Part 1, so, I'll refer you to Part 1 for it. In fact, this tutorial will build some things on Part 1 and if you have not already done so, I strongly suggest to see that tutorial first ;))

공부 목표가 정말로 중요하다. 그러나, 우리는 이미 Part1에서 목표물을 정했다. 그래서 나는 Part 1을 참조하라고 얘기한다. 사실, 이 tutorial은 Part1에서 무엇을 만든다. 그리고 네가 먼저 보지 않았다면, 강력히 먼저 볼 것을 추천한다.

Keep your mouse pointer here and click whenever you are ready reading on each textscreen

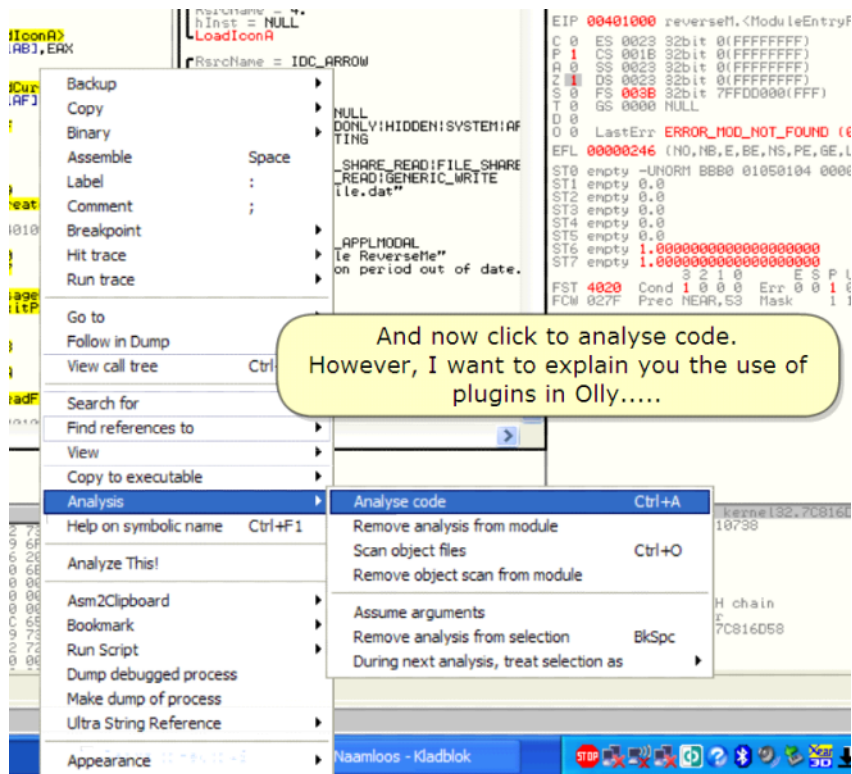
네가 다음 textscreen을 읽고자 할 때 너의 마우스를 여기에 위치하고 클릭 해.

When loading a target, Olly will automatically analyze the code to comment the code for us. (Not in certain conditions, see later Parts)

If you doubt that Olly has effectively done it, you can always make Olly do that like this....

Target이 loading 될 때, Olly는 자동으로 code에 comment를 달아 해석한다.(이것은 명확한 조건은 아니다, 나중 Parts를 보라)

Olly가 효과적으로 완료했는지 의심이 든다면, 이것처럼 항상 Olly가 할 수 있게 만든다.

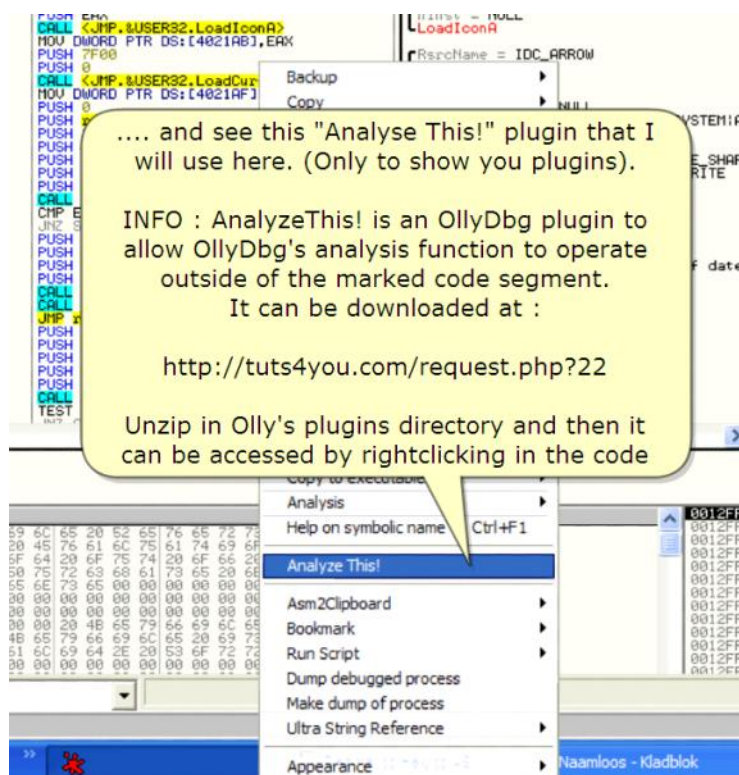


And now click to analyze code.

However, I want to explain you the use of plugins in Olly....

해석 code를 click 하자.

그러나, Olly에서 사용하기 좋은 plugins을 소개하고자 한다.



...and see this "analyze This!" plugin that I will use here. (Only to show you plugins).

그리고 "analyze This!" plugin을 여기에서 사용할 것이다.(너에게만 보여주는 plugin)

INFO : Analyze This! Is an OllyDbg plugin to allow OllyDbg's analysis function to
operate outside of the marked code segment.

It can be downloaded at :

<http://tuts4you.com/request.php?22>

Unzip in Olly's plugins directory and then it can be accessed by rightclicking in the code

정보 : Analyze This는 Ollydbg의 해석 함수가 표시된 code segment를 외부에서 조작할 수 있는 Ollydbg plugin 입니다.

여기에서 다운로드 가능하다.

<http://tuts4you.com/request.php?22>

Olly plugin directory에 풀고 code에서 rightclicking으로 접근할 수 있다.

Ok. In the blink of an eye, Olly has analyzed the code.

Ok. 눈을 깜빡이자, Olly는 code를 해석했다.

4. Solving the ReverseMe

See also Part 1 in this tutorial series but to understand everything fine, we will need to broaden your overall knowledge a little concerning bits - bytes - numbers - hex.

I'll try to keep it to the minimum that is required to understand this series though.

이 tutorial의 Part1을 보라. 그러나 네가 이해가 됐다면 우리는 지식이 넓히는데 bits - bytes - numbers - hex를 집중할 수 있다.

나는 작게 유지하기 위해 이 series를 이해하는 것이 요구된다.

If you have an outstanding knowledge in this matter, it is clear that you can skip this piece. Let's go!

만약에 뛰어난 지식을 가진다면, 이것은 명확하다. 너는 이 조각을 넘길 수 있다.

INFO:

- The decimal number system.

The easiest way to understand bits is to compare them to something known: digits. A digit is a single place that can hold numerical values between 0 and 9 in our numeric decimal system.

가장 쉽게 bit를 이해할 수 있는 방법은 숫자들을 비교하는 것이다. Digit는 한 자리다. 그것은 0과 9 사이의 10진수 system이다.

Digits are normally combined together in groups to create larger numbers.

For example, 6357 is a decimal number and has four digits.

숫자들은 보통 그룹 안에서 큰 숫자들을 만들 때 조합된다.

예를 들어, 6357은 10진수 system이고 4개의 숫자로 되어 있다.

That should all feel pretty comfortable -- we work with decimal digits every day. The neat thing about number systems is that there is nothing that forces you to have 10 different values in a digit.

매우 편안하다. -- 우리가 매일 10진수를 사용한다. 단정된 숫자 system은 아무것도 아니다. 숫자에서 10가지의 다른 값들을 가진다.

Our base-10 number system likely grew up because we have 10 fingers, but if we happened to evolve to have eight fingers instead, we would probably have a base-8 number system.

Base-10 진수는 성장했다. 왜냐하면 우리는 10개의 손가락을 가지기 때문이다. 그러나 만약에 우리가 8개의 손가락에서 진화했다면, 우리는 아마 8진수 system을 사용했을 것이다.

You can have base-anything number systems. In fact, there are lots of good reasons to use different bases in different situations.

너는 다른 number system을 가질 수 있다. 사실, 그것은 많은 다른 상황 속에서 다르게 사용하는 이유가 있다.

So, all this seems pretty simple, but how do we calculate in the decimal number system, I mean how do we assign values for numbers?

그래서, 매우 간단하다, 그러나 우리가 어떻게 10진수 system에서 계산을 해야되나? 어떻게 숫자 value 값을 할당하나?

INFO:

It is understood that in the number 6,357, the 7 is filling the 1s place, while the 5 is filling the 10s place, the 3 is filling the 100s place and the 6 is filling the 1,000s place. 6357은 7은 1의 자리, 5는 10의 자리, 3은 100의 자리, 6은 1000의 자리.

So you could express things this way if you wanted to be explicit:

네가 원하는 명쾌한 것이 있다면 표현할 수 있다.

$$(6 * 1000) + (3 * 100) + (5 * 10) + (7 * 1) \\ = 6000 + 300 + 50 + 7 = 6357$$

Another way to express it would be to use powers of 10. Assuming that we are going to represent the concept of "raised to the power of" with the "^" symbol (so "10 squared" is written as "10^2"), then another way to express it is like this:

다른 방법으로 표현할 수 있다. 그것은 10진법으로 사용할 수 있다. 생각하건데 우리는 "raised to the power of" 개념을 설명하기 위한 상징으로 "^"(그래서 "10 squared"는 "10^2"로 쓰여졌다), 다른 방법으로 이것처럼 표현할 수 있다.

$$(6 * 10^3) + (3 * 10^2) + (5 * 10^1) + (7 * 10^0) = 6,357 \\ 6000 + 300 + 50 + 7 = 6357$$

What you can see from this expression is that each digit is a placeholder for the next higher power of 10, starting in the first digit with 10 raised to the power of zero.

이 표현식에서 너는 볼 수 있다. 각 숫자들은 다음 10진수를 위해 자리를 지킨다. 처음 시작하는 10의 진수는 0이다.

Computers happen to operate using the base-2 number system, also known as the binary number system (just like the base-10 number system is known as the decimal number system). Let's find out why and how that works...

Computer는 오직 2진수에서만 작동한다, binary number system(10진수 시스템은 decimal number system)을 알고 있다. 자, 왜 사용하는지 어떻게 사용하는지 찾아보자

INFO:

- The base-2 System and the 8-bit Byte.
- 2진수 and 8byt Byte

The reason computers use the base-2 system is because it makes it a lot easier to implement them with current electronic technology. So computers use binary numbers, and therefore use binary digits in place of decimal digits. The word bit is a shortening of the words "Binary digIT."

컴퓨터가 2진수를 사용하는 이유는 쉽게 시행할 수 있고 전기적인 기술이기 때문이다. 그래서 컴퓨터는 2진수를 사용하고 2진수를 10진수 자리에 사용한다.

세계적으로 bit는 "Binary digIT"의 약어다.

Whereas decimal digits have 10 possible values ranging from 0 to 9, bits have only two possible values: 0 and 1. Therefore, a binary number is composed of only 0s and 1s, like this: 1011. How do you figure out what the value of the binary number 1011 is?

대조적으로 10진수는 0~9까지 10개의 값을 가진다, bit는 2가지 값을 가진다: 0과 1. 그런 까닭으로 2진수는 오직 0과 1로 되어 있다. 이것처럼 : 1011. 어떻게 2진수 1011은 계산할 것인가?

You do it in the same way as for 6357, but you use a base of 2 instead of a base of 10.

너는 6357를 할 때와 같은 방법으로 할 수 있다. 그러나 10진수 대신에 2진수를 사용해야 된다.

$$(1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (1 * 2^0) = 11$$
$$= 8 + 0 + 2 + 1 = 11$$

You can see that in binary numbers, each bit holds the value of increasing powers of 2.

너는 2진수에서 볼 수 있다. 각 bit는 증가하는 2진수의 값을 가진다.

That makes counting in binary pretty easy.

Starting at zero and going through 20, counting in decimal and binary looks like this:

2진수에서 쉽게 셀 수 있다.

0에서 시작하고 20까지, 계산한 10진수와 2진수를 보라. 이렇게

INFO

0 = 0

1 = 1

2 = 10

3 = 11

4 = 100

5 = 101

6 = 110

7 = 111

8=1000
9=1001
10=1010
11=1011
12=1100
13=1101
14=1110
15=1111
16=10000
17=10001
18=10010
19=10011
20=10100

INFO

Bits are rarely seen alone in computers. They are almost always bundled together into 8-bit collections, and these collections are called bytes.

컴퓨터 분야에서 드물게 쓰인다. 그들은 거의 항상 8bit 꾸러미로 다닌다. 그리고 그것들의 꾸러미는 1byte로 불린다.

Why are there 8 bits in a byte? Well, a similar question is, "Why are there 12 eggs in a dozen?" The 8-bit byte is something that people settled on through trial and error over the past 50 years.

왜 8bit를 1byte라고 부르냐? 이것은 이 질문과 같다. "왜 12개의 계란을 한 다스라고 부르냐?" 8bit는 사람들이 지난 50년간 시도와 에러를 통하여 합의를 봤다.

With 8 bits in a byte, you can represent 256 values ranging from 0 to 255, as shown here:

8bit는 1byte다. 너는 256가지의 값의 범위로 0~255까지 표현할 수 있다. 보이는 것과 같이

0 = 0000 0000
1 = 0000 0001
2 = 0000 0010
...
...
254 = 1111 1110
255 = 1111 1111

INFO

Bytes values are also used to hold individual characters in the ASCII character set, each binary value between 0 and 127 is given a specific character.

Bytes 값은 ASCII에서 각각 문자로 사용된다. 각 binary value는 0~127까지 특별한 문자로 구성되어져 있다.

Most computers extend the ASCII character set to use the full range of 256 characters available in a byte. The upper 128 characters handle special things like

accented characters from common foreign languages. See included file called Dec2Hex2Ascii.htm

대부분의 컴퓨터는 ASCII를 확장했다. Byte에서 256 문자를 사용할 수 있게 됐다. 상위 128 문자는 공통적인 외국어에서 accent 문자같이 특별한 것들을 조종할 수 있다. 포함되어 있는 파일인 Dec2Hex2Ascii.htm을 보라.

But I will discuss this further on too.

그러나 나는 나중에 이것을 다시 의논한다.

INFO

- The Hexadecimal Number Base System
- 16진수 System

Now, the next step -assuming you understood the previous- is rather easy to make: from the binary to the hexadecimal system.

이제, 이전 강의를 이해한 것으로 가정할 때 다음 step으로 binary에서 hexadecimal 으로 바꾸는 게 오히려 더 쉽다.

A big problem with the binary system is verbosity. To represent the value 202, it requires eight binary digits.

binary system의 가장 큰 문제는 장황하다. 202 값을 표현하려면, 8자리의 binary 숫자가 필요하다.

번역 주) Binary: 11001010(8자리), Dec: 202(3자리), Hex: CA(2자리)

The decimal version requires only three decimal digits and, thus, represents numbers much more compactly than does the binary numbering system.

10진법은 오직 10진수 3자리만 필요하고 2진수 system보다 표현하는데 간편하다.

This fact was not lost on the engineers who designed binary computer systems.

When dealing with large values, binary numbers quickly become too unwieldy.

사실 2진수 computer system을 개발한 engineer들은 잃은 게 없다. Large 값을 처리할 때, 2진수는 빠르게 늘어난다.

The hexadecimal (base 16) numbering system solves these problems.

Hexadecimal numbers offer the two features:

16진수 system은 이 문제들을 해결한다.

16진수는 2가지 사실을 제공한다.

- Hex numbers are very compact
- It is easy to convert from hex to binary and binary to hex.
- 16진수는 매우 간편하다.
- 그것은 16 -> 2, 2 -> 16으로 변경하는데 쉽다.

Now, all the previously explained concerning the calculation for the value represented by the binary/decimal notation equally works for this hexadecimal system.

이제, 모든 것들을 설명하기 전에 값들을 표현하기 위해 계산하는데 집중해라. 2진수/16진수 표기법은 16진수 system을 위해 똑같은 일을 한다.

If you like, you can try it out for yourself(the square value calc)
네가 좋아한다면, 너는 네 자신을 위해 도전할 수 있다.(실효값 계산)

INFO

Since we'll need to enter hexadecimal numbers into the computer system, we'll need a different mechanism for representing hexadecimal numbers to display their associated value.

우리가 컴퓨터의 16진수를 사용하기 위해서는 필요할 것이다, 관련된 16진수를 표현하기 위해 다른 mechanism이 필요하다.

In Assembly Language programming, most assemblers require the first digit of a hexadecimal number to be 0, and we place an H(h) at the end of the number to denote the number base.

Assembly programming에서, 대부분 assembler들은 16진수의 첫번째 자리가 0이 되는 것이 필요하다. 그리고 우리는 h를 숫자의 마지막에 number base를 표현하기 위해 붙인다.

To denote the difference with binary, we place a B(b) behind the number and a D (d) for the decimal number.

2진수를 다른 것과 구분하기 위해, 숫자 뒤에 b를 붙이고 Decimal 숫자 뒤에는 d를 붙인다.

The Hexadecimal Number System:

16진수 system

- Uses base 16
- Includes only the digits 0 through 9 and the letters A, B, C, D, E, and F
- 16진수를 사용한다.
- 오직 0~9와 A,B,C,D,E,F 문자를 사용한다.

A hex number example is 1C9EB7F3h

16진수 예로 1C9EB7F3h가 있다.

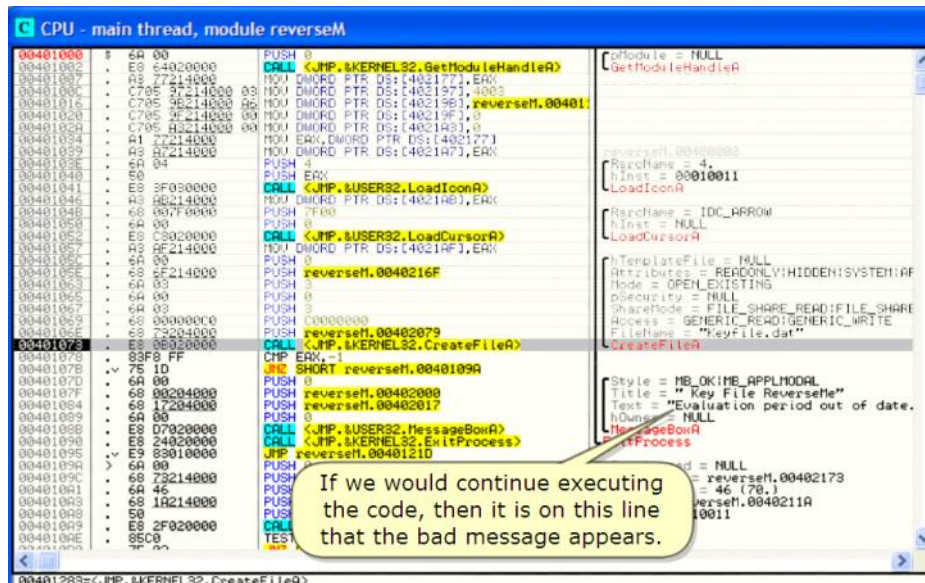
It suffices to know these basics and to be able to calculate them to the decimal system and vice versa using windows calculator(scientific in hex) (Example is 480163827d)

이것은 기본을 알고 windows 계산기를(16진수에서) 사용하여 10진수로 계산이 가능하고 반대로 할 수 있다.

번역 주) 즉 쉽게 이야기해서 16진수 -> 10진수, 10진수 -> 16진수 가능하다는 이야기 (예를 들어, 480163827d)

So far our little detour in bits, bytes, binary, decimal and hex. But we were going to solve the ReverseMe, remember? Let's continue and step over(F8) to see what goes on.

지금까지 bit, bytes, binary, decimal and hex를 약간 우회한다. 그러나 우리는 ReverseMe를 해결하기 위해 왔다 기억나? 시작하자. 그리고 F8로 무슨 일이 일어나는지 보자.



So far all good. Remember from Part 1 that here, the action takes place? Let me refresh your memory and let's go see the API again.

지금까지는 좋다. Part 1에서 이곳을 얻기 위한 액션이 기억나? 다시 새롭게 너의 기억에 넣어주마. 그리고 API를 다시 보라.

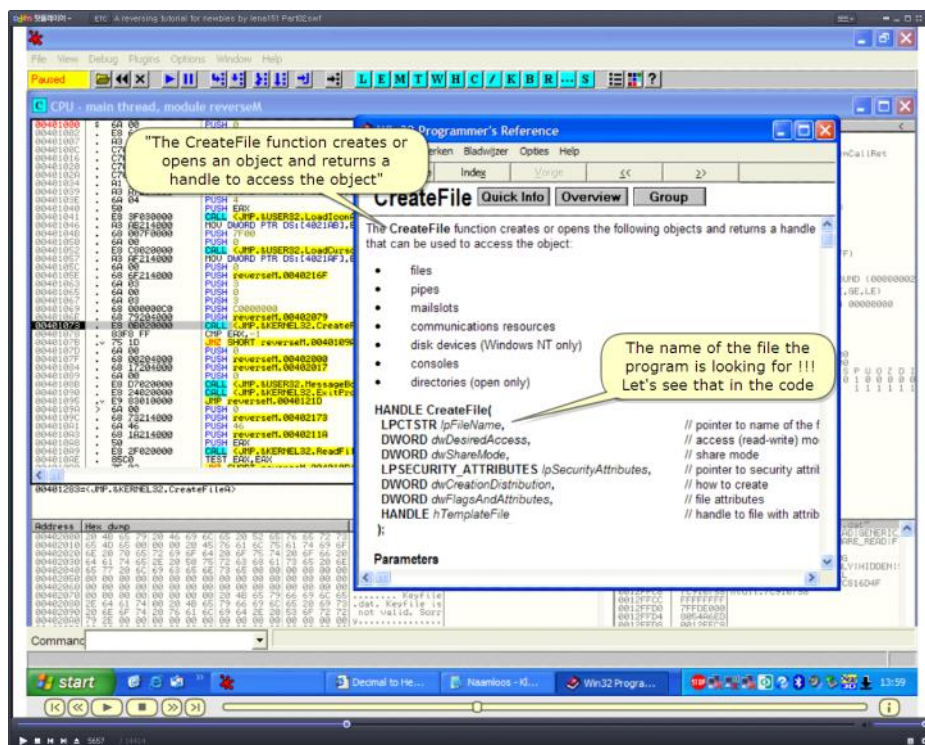
If we would continue executing the code, then it is on this line that the bad message appears.

만약 우리가 code를 실행한다면 이 line이 bad message가 나타난다.

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

우리가 badboy를 피하지 않는 한

Rightclick



"The CreateFile function creates or opens an object and returns a handle to access

"CreateFile function은 obejct와 object를 접근한 handle을 return을 만들거나 연다."

The name of the file the program is looking for!!!

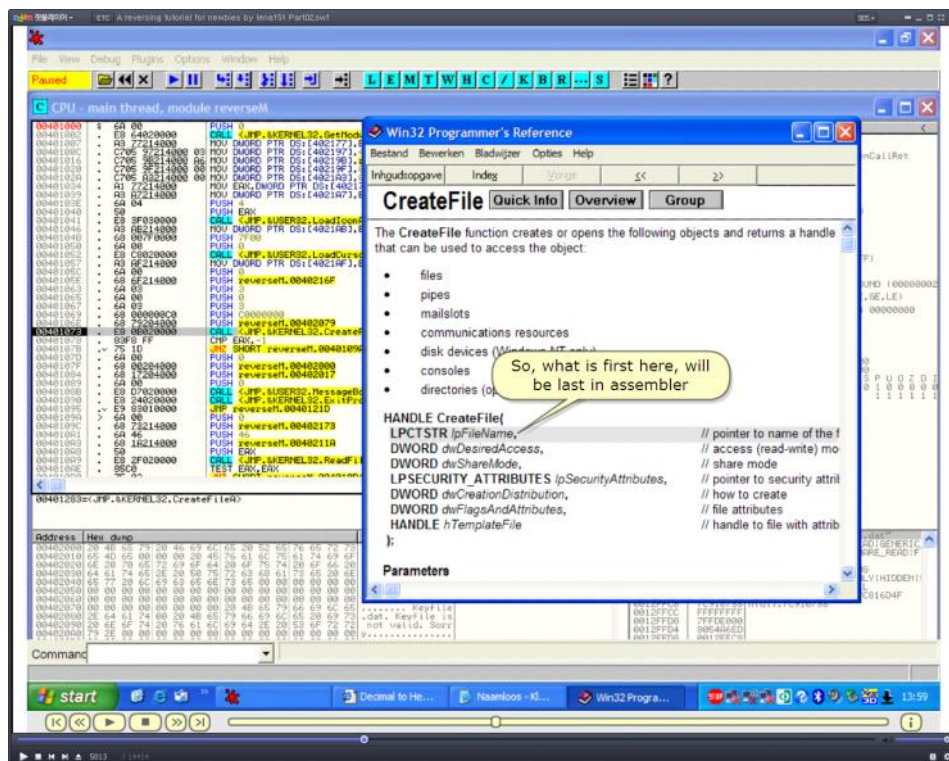
Let's see that in the code

Program file name을 보라.

코드에서 찾아보라.

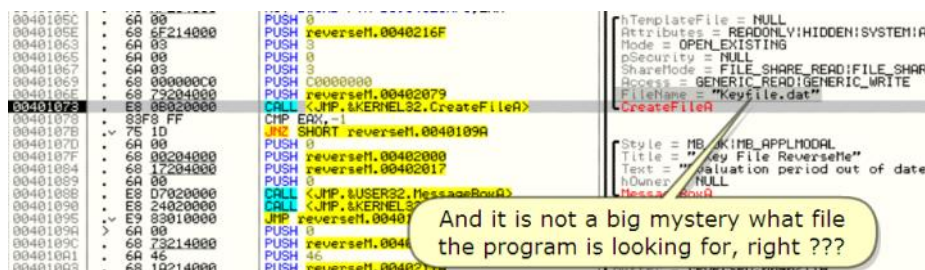
Don't forget: in assembler, you will find the parameters of a function pushed to the stack in reversed order (when POP'ed they come in regular order)

잊지 마라: assembler에서 너는 stack에 반대 순서로 넣어진 function의 paramter들을 찾을 수 있다. (규칙적인 순서로 꺼내질 때, POP이라 불린다.)



So, what is first here, will be last in assembler

그래서, 처음이 무엇인지 보라, assembler에 마지막으로 위치할 것이다.



And it is not a big mystery what file the program is looking for, right???

OK. The ReverseMe wants a file called "Keyfile.dat", well then ... let's give the ReverseMe what it wants and make one!

이것은 큰 문제가 아니다. File program이 무엇을 찾고 있는지 보이나???

Ok. ReverseMe는 "Keyfile.dat"라고 불리는 파일을 원한다. ReverseMe에게 원하는 것을 만들어주자.

Open windows notepad

And save Keyfile.dat....

Yes, leave notepad open, we may need it again :)

Notepad를 열고 Keyfile.dat를 저장해라.

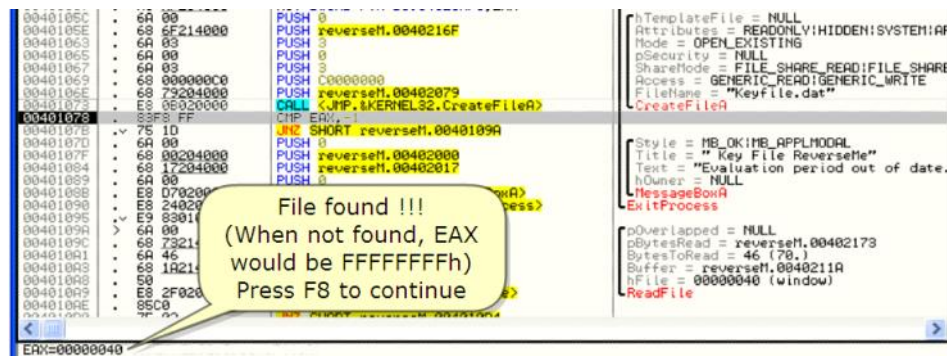
예, notepad는 열린 채로 놔둬, 우리는 다시 필요할 때가 올 거야.

So, let's execute the API.

Press F8

API를 실행하자.

F8 눌러.



File found!!!

(When not found, EAX would be FFFFFFFFh)

Press F8 to continue

...and we will jump passed the badboy!

Let's continue.

파일을 찾았다!!!

(파일을 찾지 못했을 때, EAX는 FFFF FFFFh이 될 것이다.)

F8을 눌러 그리고 우리는 badboy를 통과할 것이다.

해보자.

Scroll up to see what the future will bring us :)

And continue stepping over the code

이 사실이 우리에게 무엇을 가져올지 보기 위해 scoll 올려.

그리고 code를 실행해보자.

All right. Now the file will be read by ReadFile. Let's see the API

맞아. 이제 file은 ReadFile에 의해 읽힐 것이다. API를 보자.

See the explanation?

That clarifies a lot, doesn't it?

설명서 보이나?

꽤 명확하다. 그렇지 않냐?

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.

요약하면, ReadFile은 Keyfile.dat의 명확한 개수의 bytes를 읽기 위해 도전한다. 그것은 정확한 address의 buffer에 들어간다.

Take a look at the parameters

To understand the following in assembler.

Paramter 들을 보라.

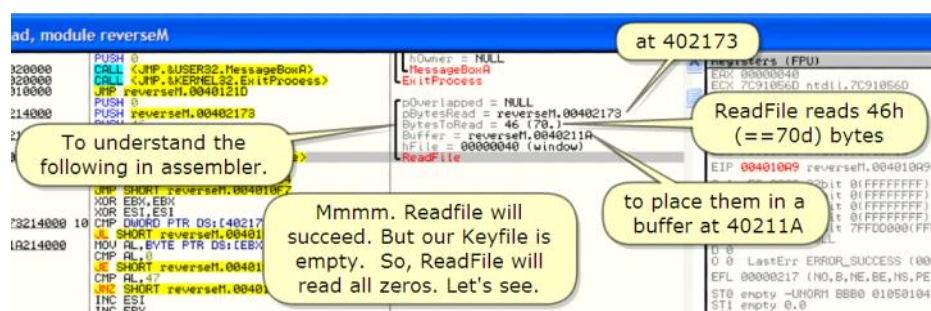
Assembler를 이해하고 따라와라.

ReadFile reads 46h(==70d) bytes

At 402173

ReadFile은 46h(==70d) bytes를 읽는다.

번역 주)h는 16진수(hexa decimal), a는 10진수(decimal)라는 것을 이해할 것이다.

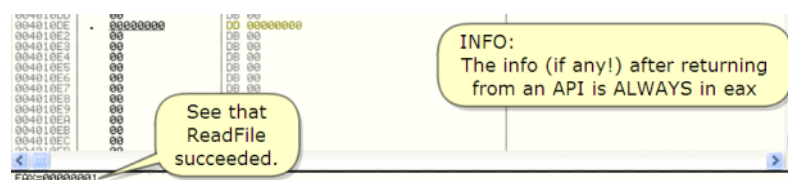


To place them in a buffer at 40211A

Mmmm. Readfile will succeed. But our Keyfile is empty. So, ReadFile will read all zeros. Let's see.

Buffer의 40211A에 저장한다.

Mmmmm. Readfile은 성공할 것이다. 그러나 우리의 Keyfile은 비어있다. 그래서 ReadFile은 zero들을 읽을 것이다. 보라.



See that ReadFile succeeded.

ReadFile이 성공한 것을 보라.

INFO:

The info (if any!) after returning from an API is ALWAYS in EAX

API에서 Return 후에 정보는 항상 EAX 안에 들어있다.

번역 주)중요한 정보. 항상 return값은 EAX 안에 저장 됩니다. TEST EAX, EAX 같은 것들이 이해될 겁니다.

And so, the jump is taken.

Continue.

그래서, jump를 한다. 계속 하자.

Do you still remember that

XOR REG, REG

Makes the register zero.

(XOR'ing the same register)

This is done to prepare EBX and ESI for the next operations.

아직 기억하냐?

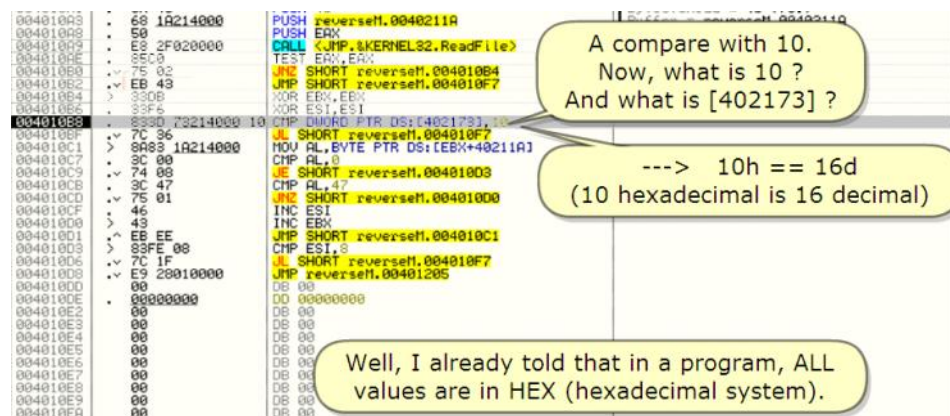
XOR REG, REG

Register를 zero로 만든다.

(같은 register를 XOR 한다.)

다음 명령어들을 위해 EBX와 ESI를 준비했다.

번역 주) C으로 초기화 시켜서 다음 명령어가 들어와도 문제 없게 해놨다는 뜻



A compare with 10.

Now, what is 10?

And what is [402173]?

10과 비교한다.

번역 주) 여기에서 10은 10h다. 10a와 헷갈리지 말자. 10h == 16d

왜 10인가?

[402173]은 무엇인가?

Well, I already told that in a program, ALL values are in HEX (hexadecimal system).

나는 이미 프로그램에서 모든 값은 hex라고 말했다.

---> 10h == 16d

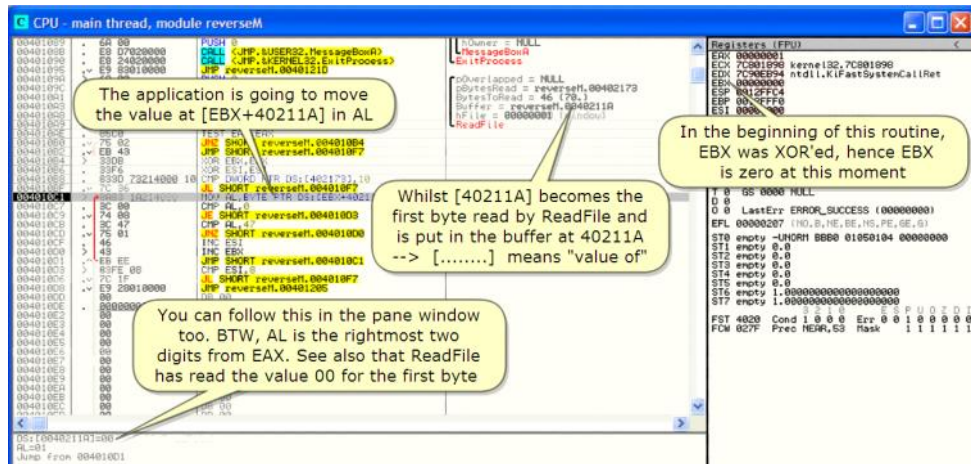
(10 hexadecimal is 16 decimal)

번역 주) 여기에서 왜 byte가 읽히냐면 답는 그릇이 AL이기 때문이다. AL은 byte, AX는 2byte, EAX는 4byte다.

---> [.....] means "value of"

[.....] 의 뜻은 "값을"

번역 주) 즉, [40211A]의 주소에 들어있는 값



You can follow this in the pane window too.

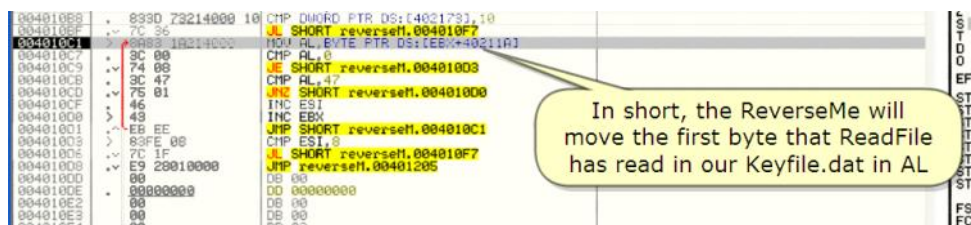
BTW, AL is the rightmost two digits from EAX.

See also that ReadFile has read the value 00 for the first byte.

너는 pane window에서 결과를 볼 수 있다.

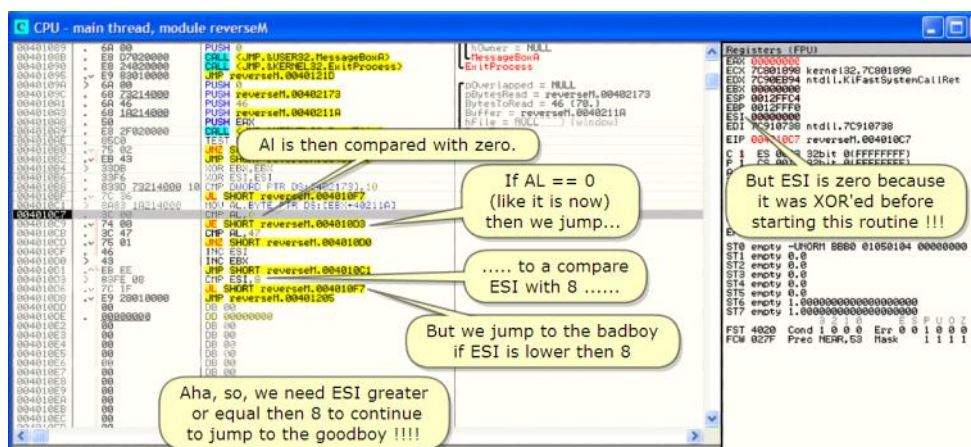
By the way, AL은 EAX에서 2가지 값이 있다.

ReadFile은 first byte의 값인 00을 읽은 것을 봐.



In short, the ReverseMe will move the first byte that ReadFile has read in our Keyfile.dat in AL

요약하면, ReverseMe는 Keyfile.dat에서 ReadFile이 읽었던 첫번째 byte를 AL로 이동할 것이다.



AL is then compared with zero.

AL은 zero와 비교된다.

If AL == 0 (like it is now) then we jump...

AL이 0과 같다면 우리는 jump 한다.

번역 주) 여기에서는 당연히 jump한다. 왜냐하면 ReadFile에서 읽은 게 0이니까. EAX의 값 0과 0을 비교하면 Z-Flag를 1로 set한다.

...to a compare ESI with 8

ESI가 8과 비교된다.

But we jump to the badboy if ESI is lower then 8

그러나 만약에 ESI가 8보다 작다면 우리는 badboy로 jump 한다.

Aha, so, we need ESI greater or equal then 8 to continue to jump to the goodboy !!!!

아하, 그래서, ESI 가 8보다 크거나 같아야 goodboy로 jump할 수 있다.

But ESI is zero because it was XOR'ed before starting this routine !!!

Let's continue stepping to see it in the code

그러나 ESI는 zero다. 왜냐하면 이 routine을 시작할 때 XOR 연산을 했기 때문이다.

Code를 넘기면서 보자.

CPU - main thread, module reverseM

00401009 * 6A 00 PUSH 0
0040100B * E8 D7020000 CALL JMP, &USER32.MessageBoxA
0040100D * E8 24020000 CALL JMP, &KERNEL32.ExitProcess
0040100F * E9 83010000 JMP reverseM.00401210
00401011 * 6A 00 PUSH 0
00401013 * 68 73214000 PUSH reverseM.00402173
00401015 * 6A 46 PUSH 46
00401017 * 68 1A214000 PUSH reverseM.0040211A
00401019 * 50 PUSH EAX
0040101B * E8 2F020000 CALL JMP, &KERNEL32.ReadFile
0040101D * 85C0 TEST EAX, EAX
0040101F * 75 02 JNC SHORT reverseM.004010B4
00401021 * EB 43 JMP SHORT reverseM.004010F7
00401023 * 330B XOR EBX, EBX
00401025 * 33F6 XOR ESI, ESI
00401027 * 833D 73214000 CMP DUOED PTR DS:14001731, 10
00401029 * 7C 36 JLE SHORT reverseM.004010F7
0040102B * 8A83 1A214000 MOV AL, BYTE PTR DS:1EA4+40211A1
0040102D * 3C 00 CMP AL, 0
0040102F * 74 05 JE SHORT reverseM.00401003
00401031 * 3C 47 CMP AL, 47
00401033 * 75 01 JNC SHORT reverseM.00401000
00401035 * 46 INC ESI
00401037 * 43 INC EBX
00401039 * EB EE JMP SHORT reverseM.004010C1
0040103B * 83FE 08 CMP ESI, 8
0040103D * 7C 1F JLE SHORT reverseM.004010F7
0040103F * E9 20010000 JMP reverseM.00401205
00401041 * 00 DB 00
00401043 * 00 DB 00
00401045 * 00 DB 00
00401047 * 00 DB 00
00401049 * 00 DB 00
0040104B * 00 DB 00
0040104D * 00 DB 00
0040104F * 00 DB 00
00401051 * 00 DB 00
00401053 * 00 DB 00
00401055 * 00 DB 00
00401057 * 00 DB 00
00401059 * 00 DB 00
0040105B * 00 DB 00
0040105D * 00 DB 00
0040105F * 00 DB 00
00401061 * 00 DB 00
00401063 * 00 DB 00
00401065 * 00 DB 00
00401067 * 00 DB 00
00401069 * 00 DB 00
0040106B * 00 DB 00
0040106D * 00 DB 00
0040106F * 00 DB 00
00401071 * 00 DB 00
00401073 * 00 DB 00
00401075 * 00 DB 00
00401077 * 00 DB 00
00401079 * 00 DB 00
0040107B * 00 DB 00
0040107D * 00 DB 00
0040107F * 00 DB 00
00401081 * 00 DB 00
00401083 * 00 DB 00
00401085 * 00 DB 00
00401087 * 00 DB 00
00401089 * 00 DB 00
0040108B * 00 DB 00
0040108D * 00 DB 00
0040108F * 00 DB 00
00401091 * 00 DB 00
00401093 * 00 DB 00
00401095 * 00 DB 00
00401097 * 00 DB 00
00401099 * 00 DB 00
0040109B * 00 DB 00
0040109D * 00 DB 00
0040109F * 00 DB 00
004010A1 * 00 DB 00
004010A3 * 00 DB 00
004010A5 * 00 DB 00
004010A7 * 00 DB 00
004010A9 * 00 DB 00
004010AB * 00 DB 00
004010AD * 00 DB 00
004010AF * 00 DB 00
004010B1 * 00 DB 00
004010B3 * 00 DB 00
004010B5 * 00 DB 00
004010B7 * 00 DB 00
004010B9 * 00 DB 00
004010BB * 00 DB 00
004010BD * 00 DB 00
004010BF * 00 DB 00
004010C1 * 00 DB 00
004010C3 * 00 DB 00
004010C5 * 00 DB 00
004010C7 * 00 DB 00
004010C9 * 00 DB 00
004010CB * 00 DB 00
004010CD * 00 DB 00
004010CF * 00 DB 00
004010D1 * 00 DB 00
004010D3 * 00 DB 00
004010D5 * 00 DB 00
004010D7 * 00 DB 00
004010D9 * 00 DB 00
004010DB * 00 DB 00
004010DD * 00 DB 00
004010DF * 00 DB 00
004010E1 * 00 DB 00
004010E3 * 00 DB 00
004010E5 * 00 DB 00
004010E7 * 00 DB 00
004010E9 * 00 DB 00
004010EB * 00 DB 00
004010ED * 00 DB 00
004010EF * 00 DB 00
004010F1 * 00 DB 00
004010F3 * 00 DB 00
004010F5 * 00 DB 00
004010F7 * 00 DB 00
004010F9 * 00 DB 00
004010FB * 00 DB 00
004010FD * 00 DB 00
004010FF * 00 DB 00
00401101 * 00 DB 00
00401103 * 00 DB 00
00401105 * 00 DB 00
00401107 * 00 DB 00
00401109 * 00 DB 00
0040110B * 00 DB 00
0040110D * 00 DB 00
0040110F * 00 DB 00
00401111 * 00 DB 00
00401113 * 00 DB 00
00401115 * 00 DB 00
00401117 * 00 DB 00
00401119 * 00 DB 00
0040111B * 00 DB 00
0040111D * 00 DB 00
0040111F * 00 DB 00
00401121 * 00 DB 00
00401123 * 00 DB 00
00401125 * 00 DB 00
00401127 * 00 DB 00
00401129 * 00 DB 00
0040112B * 00 DB 00
0040112D * 00 DB 00
0040112F * 00 DB 00
00401131 * 00 DB 00
00401133 * 00 DB 00
00401135 * 00 DB 00
00401137 * 00 DB 00
00401139 * 00 DB 00
0040113B * 00 DB 00
0040113D * 00 DB 00
0040113F * 00 DB 00
00401141 * 00 DB 00
00401143 * 00 DB 00
00401145 * 00 DB 00
00401147 * 00 DB 00
00401149 * 00 DB 00
0040114B * 00 DB 00
0040114D * 00 DB 00
0040114F * 00 DB 00
00401151 * 00 DB 00
00401153 * 00 DB 00
00401155 * 00 DB 00
00401157 * 00 DB 00
00401159 * 00 DB 00
0040115B * 00 DB 00
0040115D * 00 DB 00
0040115F * 00 DB 00
00401161 * 00 DB 00
00401163 * 00 DB 00
00401165 * 00 DB 00
00401167 * 00 DB 00
00401169 * 00 DB 00
0040116B * 00 DB 00
0040116D * 00 DB 00
0040116F * 00 DB 00
00401171 * 00 DB 00
00401173 * 00 DB 00
00401175 * 00 DB 00
00401177 * 00 DB 00
00401179 * 00 DB 00
0040117B * 00 DB 00
0040117D * 00 DB 00
0040117F * 00 DB 00
00401181 * 00 DB 00
00401183 * 00 DB 00
00401185 * 00 DB 00
00401187 * 00 DB 00
00401189 * 00 DB 00
0040118B * 00 DB 00
0040118D * 00 DB 00
0040118F * 00 DB 00
00401191 * 00 DB 00
00401193 * 00 DB 00
00401195 * 00 DB 00
00401197 * 00 DB 00
00401199 * 00 DB 00
0040119B * 00 DB 00
0040119D * 00 DB 00
0040119F * 00 DB 00
004011A1 * 00 DB 00
004011A3 * 00 DB 00
004011A5 * 00 DB 00
004011A7 * 00 DB 00
004011A9 * 00 DB 00
004011AB * 00 DB 00
004011AD * 00 DB 00
004011AF * 00 DB 00
004011B1 * 00 DB 00
004011B3 * 00 DB 00
004011B5 * 00 DB 00
004011B7 * 00 DB 00
004011B9 * 00 DB 00
004011BB * 00 DB 00
004011BD * 00 DB 00
004011BF * 00 DB 00
004011C1 * 00 DB 00
004011C3 * 00 DB 00
004011C5 * 00 DB 00
004011C7 * 00 DB 00
004011C9 * 00 DB 00
004011CB * 00 DB 00
004011CD * 00 DB 00
004011CF * 00 DB 00
004011D1 * 00 DB 00
004011D3 * 00 DB 00
004011D5 * 00 DB 00
004011D7 * 00 DB 00
004011D9 * 00 DB 00
004011DB * 00 DB 00
004011DD * 00 DB 00
004011DF * 00 DB 00
004011E1 * 00 DB 00
004011E3 * 00 DB 00
004011E5 * 00 DB 00
004011E7 * 00 DB 00
004011E9 * 00 DB 00
004011EB * 00 DB 00
004011ED * 00 DB 00
004011EF * 00 DB 00
004011F1 * 00 DB 00
004011F3 * 00 DB 00
004011F5 * 00 DB 00
004011F7 * 00 DB 00
004011F9 * 00 DB 00
004011FB * 00 DB 00
004011FD * 00 DB 00
004011FF * 00 DB 00
00401201 * 00 DB 00
00401203 * 00 DB 00
00401205 * 00 DB 00
00401207 * 00 DB 00
00401209 * 00 DB 00
0040120B * 00 DB 00
0040120D * 00 DB 00
0040120F * 00 DB 00
00401211 * 00 DB 00
00401213 * 00 DB 00
00401215 * 00 DB 00
00401217 * 00 DB 00
00401219 * 00 DB 00
0040121B * 00 DB 00
0040121D * 00 DB 00
0040121F * 00 DB 00
00401221 * 00 DB 00
00401223 * 00 DB 00
00401225 * 00 DB 00
00401227 * 00 DB 00
00401229 * 00 DB 00
0040122B * 00 DB 00
0040122D * 00 DB 00
0040122F * 00 DB 00
00401231 * 00 DB 00
00401233 * 00 DB 00
00401235 * 00 DB 00
00401237 * 00 DB 00
00401239 * 00 DB 00
0040123B * 00 DB 00
0040123D * 00 DB 00
0040123F * 00 DB 00
00401241 * 00 DB 00
00401243 * 00 DB 00
00401245 * 00 DB 00
00401247 * 00 DB 00
00401249 * 00 DB 00
0040124B * 00 DB 00
0040124D * 00 DB 00
0040124F * 00 DB 00
00401251 * 00 DB 00
00401253 * 00 DB 00
00401255 * 00 DB 00
00401257 * 00 DB 00
00401259 * 00 DB 00
0040125B * 00 DB 00
0040125D * 00 DB 00
0040125F * 00 DB 00
00401261 * 00 DB 00
00401263 * 00 DB 00
00401265 * 00 DB 00
00401267 * 00 DB 00
00401269 * 00 DB 00
0040126B * 00 DB 00
0040126D * 00 DB 00
0040126F * 00 DB 00
00401271 * 00 DB 00
00401273 * 00 DB 00
00401275 * 00 DB 00
00401277 * 00 DB 00
00401279 * 00 DB 00
0040127B * 00 DB 00
0040127D * 00 DB 00
0040127F * 00 DB 00
00401281 * 00 DB 00
00401283 * 00 DB 00
00401285 * 00 DB 00
00401287 * 00 DB 00
00401289 * 00 DB 00
0040128B * 00 DB 00
0040128D * 00 DB 00
0040128F * 00 DB 00
00401291 * 00 DB 00
00401293 * 00 DB 00
00401295 * 00 DB 00
00401297 * 00 DB 00
00401299 * 00 DB 00
0040129B * 00 DB 00
0040129D * 00 DB 00
0040129F * 00 DB 00
004012A1 * 00 DB 00
004012A3 * 00 DB 00
004012A5 * 00 DB 00
004012A7 * 00 DB 00
004012A9 * 00 DB 00
004012AB * 00 DB 00
004012AD * 00 DB 00
004012AF * 00 DB 00
004012B1 * 00 DB 00
004012B3 * 00 DB 00
004012B5 * 00 DB 00
004012B7 * 00 DB 00
004012B9 * 00 DB 00
004012BB * 00 DB 00
004012BD * 00 DB 00
004012BF * 00 DB 00
004012C1 * 00 DB 00
004012C3 * 00 DB 00
004012C5 * 00 DB 00
004012C7 * 00 DB 00
004012C9 * 00 DB 00
004012CB * 00 DB 00
004012CD * 00 DB 00
004012CF * 00 DB 00
004012D1 * 00 DB 00
004012D3 * 00 DB 00
004012D5 * 00 DB 00
004012D7 * 00 DB 00
004012D9 * 00 DB 00
004012DB * 00 DB 00
004012DD * 00 DB 00
004012DF * 00 DB 00
004012E1 * 00 DB 00
004012E3 * 00 DB 00
004012E5 * 00 DB 00
004012E7 * 00 DB 00
004012E9 * 00 DB 00
004012EB * 00 DB 00
004012ED * 00 DB 00
004012EF * 00 DB 00
004012F1 * 00 DB 00
004012F3 * 00 DB 00
004012F5 * 00 DB 00
004012F7 * 00 DB 00
004012F9 * 00 DB 00
004012FB * 00 DB 00
004012FD * 00 DB 00
004012FF * 00 DB 00
00401301 * 00 DB 00
00401303 * 00 DB 00
00401305 * 00 DB 00
00401307 * 00 DB 00
00401309 * 00 DB 00
0040130B * 00 DB 00
0040130D * 00 DB 00
0040130F * 00 DB 00
00401311 * 00 DB 00
00401313 * 00 DB 00
00401315 * 00 DB 00
00401317 * 00 DB 00
00401319 * 00 DB 00
0040131B * 00 DB 00
0040131D * 00 DB 00
0040131F * 00 DB 00
00401321 * 00 DB 00
00401323 * 00 DB 00
00401325 * 00 DB 00
00401327 * 00 DB 00
00401329 * 00 DB 00
0040132B * 00 DB 00
0040132D * 00 DB 00
0040132F * 00 DB 00
00401331 * 00 DB 00
00401333 * 00 DB 00
00401335 * 00 DB 00
00401337 * 00 DB 00
00401339 * 00 DB 00
0040133B * 00 DB 00
0040133D * 00 DB 00
0040133F * 00 DB 00
00401341 * 00 DB 00
00401343 * 00 DB 00
00401345 * 00 DB 00
00401347 * 00 DB 00
00401349 * 00 DB 00
0040134B * 00 DB 00
0040134D * 00 DB 00
0040134F * 00 DB 00
00401351 * 00 DB 00
00401353 * 00 DB 00
00401355 * 00 DB 00
00401357 * 00 DB 00
00401359 * 00 DB 00
0040135B * 00 DB 00
0040135D * 00 DB 00
0040135F * 00 DB 00
00401361 * 00 DB 00
00401363 * 00 DB 00
00401365 * 00 DB 00
00401367 * 00 DB 00
00401369 * 00 DB 00
0040136B * 00 DB 00
0040136D * 00 DB 00
0040136F * 00 DB 00
00401371 * 00 DB 00
00401373 * 00 DB 00
00401375 * 00 DB 00
00401377 * 00 DB 00
00401379 * 00 DB 00
0040137B * 00 DB 00
0040137D * 00 DB 00
0040137F * 00 DB 00
00401381 * 00 DB 00
00401383 * 00 DB 00
00401385 * 00 DB 00
00401387 * 00 DB 00
00401389 * 00 DB 00
0040138B * 00 DB 00
0040138D * 00 DB 00
0040138F * 00 DB 00
00401391 * 00 DB 00
00401393 * 00 DB 00
00401395 * 00 DB 00
00401397 * 00 DB 00
00401399 * 00 DB 00
0040139B * 00 DB 00
0040139D * 00 DB 00
0040139F * 00 DB 00
004013A1 * 00 DB 00
004013A3 * 00 DB 00
004013A5 * 00 DB 00
004013A7 * 00 DB 00
004013A9 * 00 DB 00
004013AB * 00 DB 00
004013AD * 00 DB 00
004013AF * 00 DB 00
004013B1 * 00 DB 00
004013B3 * 00 DB 00
004013B5 * 00 DB 00
004013B7 * 00 DB 00
004013B9 * 00 DB 00
004013BB * 00 DB 00
004013BD * 00 DB 00
004013BF * 00 DB 00
004013C1 * 00 DB 00
004013C3 * 00 DB 00
004013C5 * 00 DB 00
004013C7 * 00 DB 00
004013C9 * 00 DB 00
004013CB * 00 DB 00
004013CD * 00 DB 00
004013CF * 00 DB 00
004013D1 * 00 DB 00
004013D3 * 00 DB 00
004013D5 * 00 DB 00
004013D7 * 00 DB 00
004013D9 * 00 DB 00
004013DB * 00 DB 00
004013DD * 00 DB 00
004013DF * 00 DB 00
004013E1 * 00 DB 00
004013E3 * 00 DB 00
004013E5 * 00 DB 00
004013E7 * 00 DB 00
004013E9 * 00 DB 00
004013EB * 00 DB 00
004013ED * 00 DB 00
004013EF * 00 DB 00
004013F1 * 00 DB 00
004013F3 * 00 DB 00
004013F5 * 00 DB 00
004013F7 * 00 DB 00
004013F9 * 00 DB 00
004013FB * 00 DB 00
004013FD * 00 DB 00
004013FF * 00 DB 00
00401401 * 00 DB 00
00401403 * 00 DB 00
00401405 * 00 DB 00
00401407 * 00 DB 00
00401409 * 00 DB 00
0040140B * 00 DB 00
0040140D * 00 DB 00
0040140F * 00 DB 00
00401411 * 00 DB 00
00401413 * 00 DB 00
00401415 * 00 DB 00
00401417 * 00 DB 00
00401419 * 00 DB 00
0040141B * 00 DB 00
0040141D * 00 DB 00
0040141F * 00 DB 00
00401421 * 00 DB 00
00401423 * 00 DB 00
00401425 * 00 DB 00
00401427 * 00 DB 00
00401429 * 00 DB 00
0040142B * 00 DB 00
0040142D * 00 DB 00
0040142F * 00 DB 00
00401431 * 00 DB 00
00401433 * 00 DB 00
00401435 * 00 DB 00
00401437 * 00 DB 00
00401439 * 00 DB 00
0040143B * 00 DB 00
0040143D * 00 DB 00
0040143F * 00 DB 00
00401441 * 00 DB 00
00401443 * 00 DB 00
00401445 * 00 DB 00
00401447 * 00 DB 00
00401449 * 00 DB 00
0040144B * 00 DB 00
0040144D * 00 DB 00
0040144F * 00 DB 00
00401451 * 00 DB 00
00401453 * 00 DB 00
00401455 * 00 DB 00
00401457 * 00 DB 00
00401459 * 00 DB 00
0040145B * 00 DB 00
0040145D * 00 DB 00
0040145F * 00 DB 00
00401461 * 00 DB 00
00401463 * 00 DB 00
00401465 * 00 DB 00
00401467 * 00 DB 00
00401469 * 00 DB 00
0040146B * 00 DB 00
0040146D * 00 DB 00
0040146F * 00 DB 00
00401471 * 00 DB 00
00401473 * 00 DB 00
00401475 * 00 DB 00
00401477 * 00 DB 00
00401479 * 00 DB 00
0040147B * 00 DB 00
0040147D * 00 DB 00
0040147F * 00 DB 00
00401481 * 00 DB 00
00401483 * 00 DB 00
00401485 * 00 DB 00
00401487 * 00 DB 00
00401489 * 00 DB 00
0040148B * 00 DB 00
0040148D * 00 DB 00
0040148F * 00 DB 00
00401491 * 00 DB 00
00401493 * 00 DB 00
00401495 * 00 DB 00
00401497 * 00 DB 00
00401499 * 00 DB 00
0040149B * 00 DB 00
0040149D * 00 DB 00
0040149F * 00 DB 00
004014A1 * 00 DB 00
004014A3 * 00 DB 00
004014A5 * 00 DB 00
004014A7 * 00 DB 00
004014A9 * 00 DB 00
004014AB * 00 DB 00
004014AD * 00 DB 00
004014AF * 00 DB 00
004014B1 * 00 DB 00
004014B3 * 00 DB 00
004014B5 * 00 DB 00
004014B7 * 00 DB 00
004014B9 * 00 DB 00
004014BB * 00 DB 00
004014BD * 00 DB 00
004014BF * 00 DB 00
004014C1 * 00 DB 00
004014C3 * 00 DB 00
004014C5 * 00 DB 00
004014C7 * 00 DB 00
004014C9 * 00 DB 00
004014CB * 00 DB 00
004014CD * 00 DB 00
004014CF * 00 DB 00
004014D1 * 00 DB 00
004014D3 * 00 DB 00
004014D5 * 00 DB 00
004014D7 * 00 DB 00
004014D9 * 00 DB 00
004014DB * 00 DB 00
004014DD * 00 DB 00
004014DF * 00 DB 00
004014E1 * 00 DB 00
004014E3 * 00 DB 00
004014E5 * 00 DB 00
004014E7 * 00 DB 00
004014E9 * 00 DB 00
004014EB * 00 DB 00
004014ED * 00 DB 00
004014EF * 00 DB 00
004014F1 * 00 DB 00
004014F3 * 00 DB 00
004014F5 * 00 DB 00
004014F7 * 00 DB 00
004014F9 * 00 DB 00

What's up next?

무슨 일이냐?

Aha, notice that AL

(which is our first byte read, remember) is compared to 47h

아하, AL이 47h와 비교됐다.(AL에 있는 내용은 우리가 처음에 읽은 byte다.)

"But what is 47h ?" you ask.

Well, every char, number, sign, has its hexadecimal counterpart.

(See explaining in info's before).

"47h이 뭐야?"

모든 문자, 숫자, 부호는 16진수로 이루어져 있다.

(이전에 설명했다)

I have prepared a html page with these for you and also included it in this package.

나는 이미 이것과 함께 너를 위해 html page를 이 package 안에 포함해서 준비해뒀다.

BTW, there exist also tools to do this. A very good one is "Crackerstool" which I have also included in this package.

Remark: place the included comdlg32.ocx in the same dir if it doesn't run on your machine.

By the way, tools가 일을 하기 위해 존재한다. 매우 좋은 "Crackerstool" 이다. 이미 이 package에 포함해뒀다.

주목: comdlg32.ocx를 같은 directory에 포함됐다. 만약에 이것이 없으면 너의 컴퓨터에서 실행되지 않는다.

This is the chart with the dec/hex/ascii representations. (See also before). Each byte can have a value ranging from 00 to FF

Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII
0	00	NUL	32	20	SP	64	40	@	96	60	'
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	\$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(72	48	H	104	68	h
9	09	HT	41	29)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s

Indeed : 47h == G in ascii

In our example, 47h (equals 71d) stands for the letter G (in ascii)

This is the chart with the dec/hex/ascii representations. (See also before). Each byte can have a value ranging from 00 to FF

이 chart는 dec/hex/ascii 표기법이다. 각 byte는 00~FF 까지 범위를 가진다.

번역 주) 2진수, 16진수 이해가 안되면 힘들어요. 쉬운 책부터 시작하세요. 꼭 2진수, 16진수를 이해해야 됩니다.

In our example, 47h (equals 71d) stands for the letter G(in ascii)

Indeed : 47h == G in ascii

이번 예제에서는, 47h는 (71d와 같다) G 문자를 뜻한다(ascii)

47h == G (ascii에서)

```

004010A9  . 5B          PUSH     EBX
004010AA  . E8 2F020000 CALL     <JMP.<kernel32.ReadFile>
004010AD  . 85C0         TEST     EAX, EAX
004010AE  . 75 02        JNZ     SHORT reverseMe.004010B4
004010AF  . EB 43        JMP     SHORT reverseMe.004010B4
004010B0  . 33DB        XOR     EBX, EBX
004010B1  . 33F6        XOR     ESI, ESI
004010B2  . 833D 73214000 CMP     DWORD PTR DS:[402173], 10
004010B3  . 7C 36        JL      SHORT reverseMe.004010B7
004010B4  . 8A83 1A214000 MOV     AL, BYTE PTR DS:[EBX+40211A]
004010B5  . 3C 00        CMP     AL, 0
004010B6  . 74 08        JE      SHORT reverseMe.004010B3
004010B7  . 47          CMP     AL, 47
004010B8  . 75 01        JNZ     SHORT reverseMe.004010BD
004010B9  . 46          INC     ESI
004010BA  . 43          INC     EBX
004010BB  . EB EE        JMP     SHORT reverseMe.004010C1
004010BC  . 33FE 08      CMP     ESI, 8
004010BD  . 7C 1F        JL      SHORT reverseMe.004010F7
004010BE  . E9 28010000 JMP     reverseMe.00401205
004010BF  . 00          DB      00
004010C0  . 00          DB      00
004010C1  . 00          DB      00
004010C2  . 00          DB      00
004010C3  . 00          DB      00
004010C4  . 00          DB      00
004010C5  . 00          DB      00
004010C6  . 00          DB      00
004010C7  . 00          DB      00
004010C8  . 00          DB      00
004010C9  . 00          DB      00
004010CA  . 00          DB      00
004010CB  . 00          DB      00
004010CC  . 00          DB      00
004010CD  . 00          DB      00
004010CE  . 00          DB      00
004010CF  . 00          DB      00
004010D0  . 00          DB      00
004010D1  . 00          DB      00
004010D2  . 00          DB      00
004010D3  . 00          DB      00
004010D4  . 00          DB      00
004010D5  . 00          DB      00
004010D6  . 00          DB      00
004010D7  . 00          DB      00
004010D8  . 00          DB      00
004010D9  . 00          DB      00
004010DA  . 00          DB      00
004010DB  . 00          DB      00
004010DC  . 00          DB      00
004010DD  . 00          DB      00
004010DE  . 00          DB      00
004010DF  . 00          DB      00
004010E0  . 00          DB      00
004010E1  . 00          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
004010F2  . 00          DB      00
004010F3  . 00          DB      00
004010F4  . 00          DB      00
004010F5  . 00          DB      00
004010F6  . 00          DB      00
004010F7  . 00          DB      00
004010F8  . 00          DB      00
004010F9  . 00          DB      00
004010FA  . 00          DB      00
004010FB  . 00          DB      00
004010FC  . 00          DB      00
004010FD  . 00          DB      00
004010FE  . 00          DB      00
004010FF  . 00          DB      00

```

Mmmm, so the ReverseMe verifies if the first found byte is G

음, ReverseMe는 첫번째로 찾은 byte가 G인지 검증한다.

```

004010B0  . 75 02        JNZ     SHORT reverseMe.004010B4
004010B1  . EB 43        JMP     SHORT reverseMe.004010B7
004010B2  . 33DB        XOR     EBX, EBX
004010B3  . 33F6        XOR     ESI, ESI
004010B4  . 833D 73214000 CMP     DWORD PTR DS:[402173], 10
004010B5  . 7C 36        JL      SHORT reverseMe.004010B7
004010B6  . 8A83 1A214000 MOV     AL, BYTE PTR DS:[EBX+40211A]
004010B7  . 3C 00        CMP     AL, 0
004010B8  . 74 08        JE      SHORT reverseMe.004010B3
004010B9  . 47          CMP     AL, 47
004010BA  . 75 01        JNZ     SHORT reverseMe.004010BD
004010BB  . 46          INC     ESI
004010BC  . 43          INC     EBX
004010BD  . EB EE        JMP     SHORT reverseMe.004010C1
004010BE  . 33FE 08      CMP     ESI, 8
004010BF  . 7C 1F        JL      SHORT reverseMe.004010F7
004010C0  . E9 28010000 JMP     reverseMe.00401205
004010C1  . 00          DB      00
004010C2  . 00          DB      00
004010C3  . 00          DB      00
004010C4  . 00          DB      00
004010C5  . 00          DB      00
004010C6  . 00          DB      00
004010C7  . 00          DB      00
004010C8  . 00          DB      00
004010C9  . 00          DB      00
004010CA  . 00          DB      00
004010CB  . 00          DB      00
004010CC  . 00          DB      00
004010CD  . 00          DB      00
004010CE  . 00          DB      00
004010CF  . 00          DB      00
004010D0  . 00          DB      00
004010D1  . 00          DB      00
004010D2  . 00          DB      00
004010D3  . 00          DB      00
004010D4  . 00          DB      00
004010D5  . 00          DB      00
004010D6  . 00          DB      00
004010D7  . 00          DB      00
004010D8  . 00          DB      00
004010D9  . 00          DB      00
004010DA  . 00          DB      00
004010DB  . 00          DB      00
004010DC  . 00          DB      00
004010DD  . 00          DB      00
004010DE  . 00          DB      00
004010DF  . 00          DB      00
004010E0  . 00          DB      00
004010E1  . 00          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
004010F2  . 00          DB      00
004010F3  . 00          DB      00
004010F4  . 00          DB      00
004010F5  . 00          DB      00
004010F6  . 00          DB      00
004010F7  . 00          DB      00
004010F8  . 00          DB      00
004010F9  . 00          DB      00
004010FA  . 00          DB      00
004010FB  . 00          DB      00
004010FC  . 00          DB      00
004010FD  . 00          DB      00
004010FE  . 00          DB      00
004010FF  . 00          DB      00

```

Mmmm, if the byte is not "G", then we jump over the command

만약에 G가 아니면 명령을 jump한다.

INC ESI

Let's see what INC means

INC 의미가 무엇인지 보자.

ASSEMBLER INFO:

INC (Increment)

Syntax:

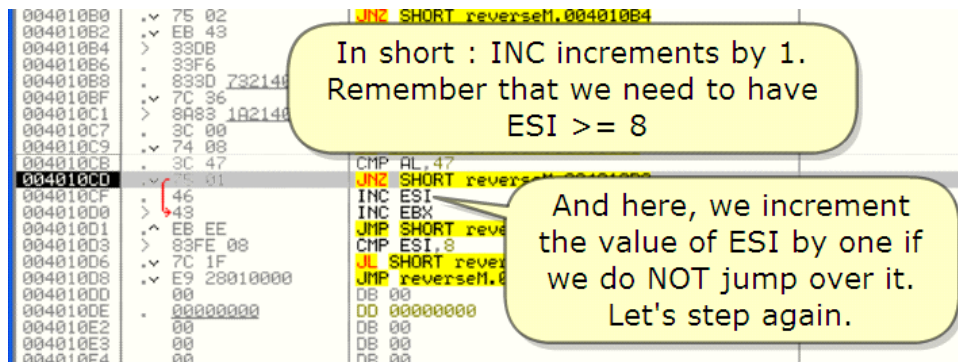
INC	Register
-----	----------

INC is the opposite of the DEC instruction it increases values by 1.

INC can set the Z/O flags.

INC 반대는 DEC 명령어다. INC 뜻은 값을 1만큼 증가시킨다.

INC는 Z/O flags를 set한다.



In short : INC increments by 1.

Remember that we need to have

ESI >= 8

요약하면, INC는 1을 증가시킨다.

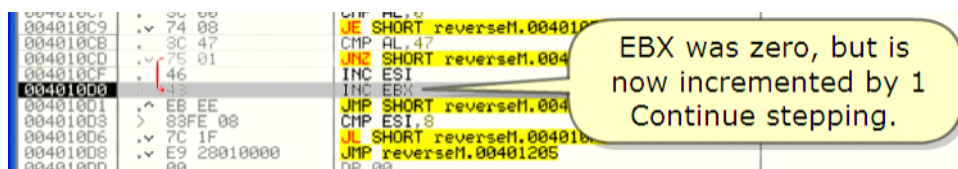
기억해. 우리는 ESI가 8이상 되는 것이 필요하다.

And here, we increment the value of ESI by one if we do NOT jump over it.

Let's step again.

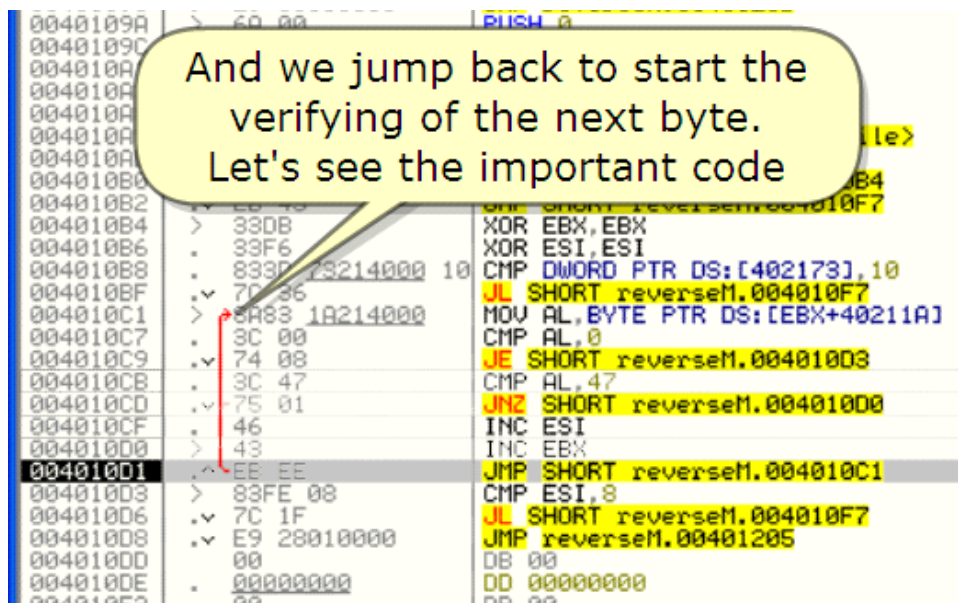
그리고 이곳은, ESI value를 하나씩 증가한다. 우리는 이것을 jump 할 수 없다.

계속 해.



EBX was zero, but is now incremented by 1 Continue stepping.

EBX는 zero, 그러나 이 step을 계속하면 1이 증가된다.



And we jump back to start the verifying of the next byte.

Let's see the important code

그리고 우리는 다음 byte를 검증하기 위해 jump 한다.

중요한 code를 잘 보라.

Assembly code snippet showing instructions like `INC ESI` and `JE SHORT reverseH.00401003`. A yellow callout box contains the text: "We need to increment ESI to 8 --> this can only if the byte is 'G'"

We need to increment ESI to 8

--> this can only if the byte is "G"

우리는 ESI를 8로 증가시키는 게 필요하다.

--> 오직 byte는 G만 된다.

Assembly code snippet showing instructions like `INC ESI` and `JE SHORT reverseH.00401003`. A yellow callout box contains the text: "And if the byte is zero (end of the file), it is verified if ESI is at least 8 --> Were there at least 8 times 'G' ? Also remember that in the beginning was verified if there are at least 16d bytes read in the file"

And if the byte is zero (end of the file), it is verified if ESI is at least 8

--> Were there at least 8 times "G" ?

그리고 만약에 byte가 zero라면(file의 마지막), ESI가 8보다 작은지 검증됐다.

--> G의 갯수가 8보다 작다면?

Also remember that in the beginning was verified if there are at least 16d bytes read in the file

또한 기억해라. 처음 시작할 때 File에서 최소한 16d bytes 이상이어야 읽는다는 것을 검증했다.

Assembly code snippet showing instructions like `INC ESI` and `JE SHORT reverseH.00401003`. A yellow callout box contains the text: "Do you understand that EBX has the role of counter ? EBX was incremented to 1. This value is added to the pointer's value from the first read byte. So, we will verify the second byte from Keyfile.dat now. If this is not clear, take your time and study it all over again. Let's continue ..."

Do you understand that EBX has the role of counter? EBX was incremented to 1. This value is added to the pointer's value from the first read byte.

So, we will verify the second byte from Keyfile.dat now.

EBX가 이 role의 counter라는 것을 이해했어? EBX는 1씩 증가된다. 이 값은 첫번째 byte를 읽어 pointer 값에 추가됐다.

그래서 우리는 Keyfile.dat에서 second byte를 검증한다.

If this is not clear, take your time and study it all over again. Let's continue...

만약에 이게 명확하지 않다면, 너의 시간을 들여 모든 것을 다시 공부해라. 계속하자.

Assembly code snippet showing instructions like `INC ESI` and `JE SHORT reverseH.00401003`. A yellow callout box contains the text: "Once again : if the verified byte is 00 (empty) ..."

Once again : if the verified bytes is 00 (empty) ...

다시 한 번 : 검증된 bytes는 00 이다. (비어있다)

```

004010C1 > 8A 03 1A214000 MOV AL,BYTE PTR DS:[EBX+40211A]
004010C3 > 3C 00 CMP AL,0
004010C5 > 74 03 JE SHORT reverseM.004010D3
004010C7 > 47 00 CMP AL,0
004010C9 > 75 01 JNZ SHORT reverseM.004010D0
004010CB > 46 00 INC ESI
004010CD > 43 00 INC EBX
004010CF > EB EE JMP SHORT reverseM.004010C1
004010D1 > 83 FE 08 CMP ESI,8
004010D3 > 7C 1F JL SHORT reverseM.004010F7
004010D5 > E9 20010000 JMP reverseM.00401205
004010D7 > 00 00 DB 00
004010D9 > 00 00 DB 00
004010DB > 00 00 DB 00
004010DD > 00 00 DB 00
004010DE > 00 00 DB 00
004010DF > 00 00 DB 00
004010E0 > 00 00 DB 00
004010E1 > 00 00 DB 00
004010E2 > 00 00 DB 00
004010E3 > 00 00 DB 00
004010E4 > 00 00 DB 00
004010E5 > 00 00 DB 00
004010E6 > 00 00 DB 00
004010E7 > 00 00 DB 00
004010E8 > 00 00 DB 00
004010E9 > 00 00 DB 00
004010EA > 00 00 DB 00
004010EB > 00 00 DB 00
004010EC > 00 00 DB 00
004010ED > 00 00 DB 00
004010EE > 00 00 DB 00
004010EF > 00 00 DB 00
004010F0 > 00 00 DB 00
004010F1 > 00 00 DB 00
004010F2 > 00 00 DB 00
004010F3 > 00 00 DB 00
004010F4 > 00 00 DB 00
004010F5 > EB 00 JMP SHORT reverseM.004010F7
004010F7 > 6A 00 PUSH 0
004010F9 > 68 00204000 PUSH reverseM.00402000
004010FB > 68 36204000 PUSH reverseM.00402086
004010FD > 6A 00 PUSH 0
004010FF > E8 5D020000 CALL <JMP.<USER32.MessageBoxA>
00401101 > E8 AA010000 CALL <JMP.<KERNEL32.ExitProcess>
00401103 > E9 09010000 JMP reverseM.0040121D

```

....then we jump to verify if ESI is equal or bigger then 8.

.... Then we jump to verify if ESI is equal or bigger then 8.

Scroll up.

그래서 우리는 ESI가 8보다 같거나 크다는 것을 검증하기 위해 jump 한다.

Scroll 올려.

```

004010D1 > EB EE JMP SHORT reverseM.004010C1
004010D3 > 83 FE 08 CMP ESI,8
004010D5 > 7C 1F JL SHORT reverseM.004010F7
004010D7 > E9 20010000 JMP reverseM.00401205
004010D9 > 00 00 DB 00
004010DB > 00 00 DB 00
004010DD > 00 00 DB 00
004010DE > 00 00 DB 00
004010DF > 00 00 DB 00
004010E0 > 00 00 DB 00
004010E1 > 00 00 DB 00
004010E2 > 00 00 DB 00
004010E3 > 00 00 DB 00
004010E4 > 00 00 DB 00
004010E5 > 00 00 DB 00
004010E6 > 00 00 DB 00
004010E7 > 00 00 DB 00
004010E8 > 00 00 DB 00
004010E9 > 00 00 DB 00
004010EA > 00 00 DB 00
004010EB > 00 00 DB 00
004010EC > 00 00 DB 00
004010ED > 00 00 DB 00
004010EE > 00 00 DB 00
004010EF > 00 00 DB 00
004010F0 > 00 00 DB 00
004010F1 > 00 00 DB 00
004010F2 > 00 00 DB 00
004010F3 > 00 00 DB 00
004010F4 > 00 00 DB 00
004010F5 > EB 00 JMP SHORT reverseM.004010F7
004010F7 > 6A 00 PUSH 0
004010F9 > 68 00204000 PUSH reverseM.00402000
004010FB > 68 36204000 PUSH reverseM.00402086
004010FD > 6A 00 PUSH 0
004010FF > E8 5D020000 CALL <JMP.<USER32.MessageBoxA>
00401101 > E8 AA010000 CALL <JMP.<KERNEL32.ExitProcess>
00401103 > E9 09010000 JMP reverseM.0040121D

```

And if there were not yet 8 "G" then we jump to the badboy.

And if there were not yet 8 "G"

Then we jump to the badboy.

그리고 아직 "G"가 8이 되지 않았다면 우리는 badboy로 jump 한다.

```

004010D1 > EB EE JMP SHORT reverseM.004010C1
004010D3 > 83 FE 08 CMP ESI,8
004010D5 > 7C 1F JL SHORT reverseM.004010F7
004010D7 > E9 20010000 JMP reverseM.00401205
004010D9 > 00 00 DB 00
004010DB > 00 00 DB 00
004010DD > 00 00 DB 00
004010DE > 00 00 DB 00
004010DF > 00 00 DB 00
004010E0 > 00 00 DB 00
004010E1 > 00 00 DB 00
004010E2 > 00 00 DB 00
004010E3 > 00 00 DB 00
004010E4 > 00 00 DB 00
004010E5 > 00 00 DB 00
004010E6 > 00 00 DB 00
004010E7 > 00 00 DB 00
004010E8 > 00 00 DB 00
004010E9 > 00 00 DB 00
004010EA > 00 00 DB 00
004010EB > 00 00 DB 00
004010EC > 00 00 DB 00
004010ED > 00 00 DB 00
004010EE > 00 00 DB 00
004010EF > 00 00 DB 00
004010F0 > 00 00 DB 00
004010F1 > 00 00 DB 00
004010F2 > 00 00 DB 00
004010F3 > 00 00 DB 00
004010F4 > 00 00 DB 00
004010F5 > EB 00 JMP SHORT reverseM.004010F7
004010F7 > 6A 00 PUSH 0
004010F9 > 68 00204000 PUSH reverseM.00402000
004010FB > 68 36204000 PUSH reverseM.00402086
004010FD > 6A 00 PUSH 0
004010FF > E8 5D020000 CALL <JMP.<USER32.MessageBoxA>
00401101 > E8 AA010000 CALL <JMP.<KERNEL32.ExitProcess>
00401103 > E9 09010000 JMP reverseM.0040121D

```

Else, we jump to the goodboy !!!

Else, we jump to the goodboy !!!

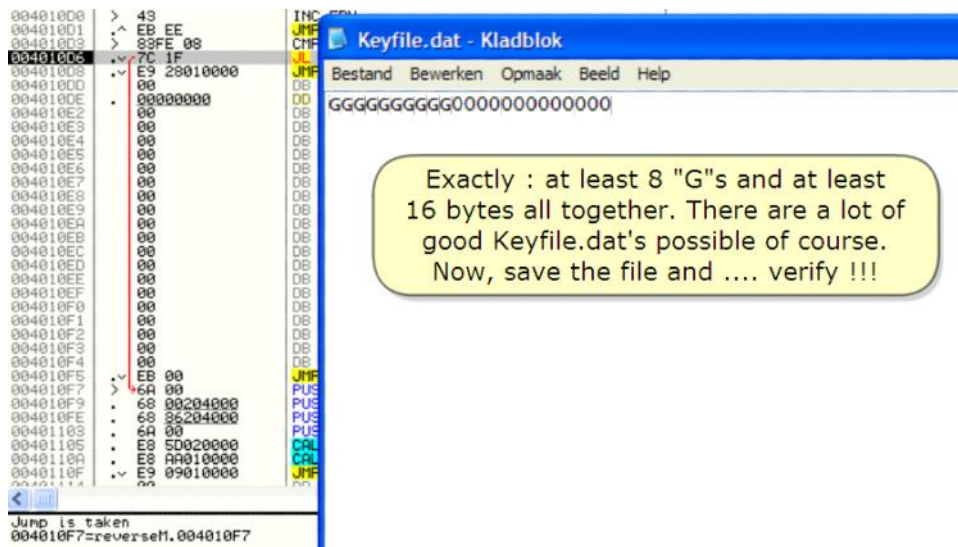
아니면, 우리는 goodboy로 jump 한다!!!

So, let's avoid this situation ;)

We have enough information now to make ourselves a right Keyfile.dat

우리는 이 상황을 피하겠다.

우리는 올바른 Keyfile.dat를 만들 충분한 정보를 가지고 있다.



Exactly : at least 8 "G"s and at least 16 bytes all together. There are a lot of good Keyfile.dat's possible of course.

Now, save the file and ... verify !!!

정확 : 최소한 "G"가 8개이고 최소한 16bytes 가 되어야 한다. 좋은 Keyfile.dat를 만들 가능성이 꽤 많다.

이제, file을 저장하고 검증하자 !!!

번역 주) 저는 이곳에서 꽤 헤맸습니다. 첫번째 조건은 G가 최소한 8개이고 두번째 조건은 최소한 16bytes 입니다. 두 가지 조건을 모두 만족해야 됩니다.

Save the changes

바뀐 것을 저장해라.

Restart the ReverseMe

ReverseMe를 재시작 해라.

5. Testing the keyfile



With Keyfile.dat in the same directory as the ReverseMe, let's follow and verify

Yep, the jump over the first badboy is taken

Keyfile.dat는 ReverseMe와 같은 directory에 있어야 한다. 따라가보며 검증하자. 첫번째 badboy를 Jump했다.

004010BE	85C0	TEST EAX,EAX
004010B0	75 02	JNZ SHORT reverseM.004010B4
004010B2	EB 43	JMP SHORT reverseM.004010F7
004010B4	3308	XOR EBX,EBX
004010B6	33F6	XOR ESI,ESI
004010B8	8330 73214000 10	CMP DWORD PTR DS:[4021731],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
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

Jump is NOT taken
004010F7=reverseM.004010F7

No jumping to the badboy now !!!

No jumping to the badboy now !!!

이제 Badboy로 jump하지 않는다.

004010B8	8330 73214000 10	CMP DWORD PTR DS:[4021731],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
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

DS:[0040211A]=47 ('G')
AL=01
Jump from 004010D1

The first byte ...

...is a "G" ...

The first byte ...

...is a "G" ...

첫번째 byte가 "G" 다.

004010AE	85C0	TEST EAX,EAX
004010B0	75 02	JNZ SHORT reverseM.004010B4
004010B2	EB 43	JMP SHORT reverseM.004010F7
004010B4	330B	XOR EBX,EBX
004010B6	33F6	XOR ESI,ESI
004010B8	83D0 73214000 10	CMP DWORD PTR DS:[4021731],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	4E	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.00401000
004010DD	00	DB 00
004010DE	00	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

Jump is NOT taken
004010D3=reverseM.004010D3

It is not empty, so we do not jump

It is not empty, so we do not jump

이것은 비어있지 않다, 그래서 우리는 jump 하지 않는다.

004010C9	74 08	JE SHORT reverseM.004010D3
004010CB	3C 47	CMP AL,47
004010CD	75 01	JNZ SHORT reverseM.004010D0
004010CF	4E	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.00401000
004010DD	00	DB 00
004010DE	00	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

Jump is NOT taken
004010D0=reverseM.004010D0

It is equal to 47h, --> no jump either

It is equal to 47h, --> no jump either

이것은 47h와 같다. --> jump하지 않는다.

번역 주) "G"는 16진수로 47h이다. CMP에 의해서 47 - 47 이므로 0이 된다. 0이 되면 Z-Flag는 1로 set 된다. 그러므로 JNZ는 Z-Flag가 1이 아닐 때 jump하는데 Z-Flag가 1이므로 jump를 하지 않는다.

004010AE	85C0	TEST EAX,EAX
004010B0	75 02	JNZ SHORT reverseM.004010B4
004010B2	EB 43	JMP SHORT reverseM.004010F7
004010B4	330B	XOR EBX,EBX
004010B6	33F6	XOR ESI,ESI
004010B8	83D0 73214000 10	CMP DWORD PTR DS:[4021731],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	4E	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.00401000
004010DD	00	DB 00
004010DE	00	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

It was a "G", so ESI is incremented from 0 to 1

It was a "G", so ESI is incremented from 0 to 1

이것은 "G", 그래서 ESI는 0에서 1로 증가됐다.

004010E3	7C 36	JL SHORT reverseH.004010F7
004010E4	3C 00	CMP AL, BYTE PTR DS:[40214000]
004010E5	74 08	JE SHORT reverseH.004010D3
004010E6	3C 47	CMP AL, 47
004010E7	75 01	JNZ SHORT reverseH.004010D0
004010E8	46	INC ESI
004010E9	43	INC EBX
004010EA	EB EE	JMP SHORT reverseH.004010C1
004010EB	83 FE 08	CMP ESI, 8
004010EC	7C 1F	JL SHORT reverseH.004010F7
004010ED	E9 28010000	JMP reverseH.00401205

And EBX is incremented to verify the next byte in the buffer

And EBX is incremented to verify the next byte in the buffer
그리고 EBX는 buffer에서 다음 byte를 검증하기 위해 증가됐다.

004010E3	7C 36	JL SHORT reverseH.004010F7
004010E4	3C 00	CMP AL, BYTE PTR DS:[40214000]
004010E5	74 08	JE SHORT reverseH.004010D3
004010E6	3C 47	CMP AL, 47
004010E7	75 01	JNZ SHORT reverseH.004010D0
004010E8	46	INC ESI
004010E9	43	INC EBX
004010EA	EB EE	JMP SHORT reverseH.004010C1
004010EB	83 FE 08	CMP ESI, 8
004010EC	7C 1F	JL SHORT reverseH.004010F7
004010ED	E9 28010000	JMP reverseH.00401205

You see that this jump serves as a loop.

You see that this jump serves as a loop.
너는 이 loop에서 jump를 지원하는 것을 봤다.

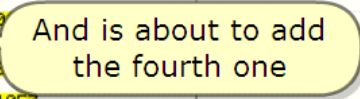
See that we verify the byte (at 40211B) now.
This is the second byte which is again "G"

See that we verify the byte (at 40211B) now.
This is the second byte which is again "G"
우리는 byte를 검증한다.(40211B에서)
이것은 2번째 byte도 "G"를 뜻한다.

004010E3	7C 36	JL SHORT reverseH.004010F7
004010E4	3C 00	CMP AL, BYTE PTR DS:[40214000]
004010E5	74 08	JE SHORT reverseH.004010D3
004010E6	3C 47	CMP AL, 47
004010E7	75 01	JNZ SHORT reverseH.004010D0
004010E8	46	INC ESI
004010E9	43	INC EBX
004010EA	EB EE	JMP SHORT reverseH.004010C1
004010EB	83 FE 08	CMP ESI, 8
004010EC	7C 1F	JL SHORT reverseH.004010F7
004010ED	E9 28010000	JMP reverseH.00401205

Look a second at the loop, to see how it all works

Look a second at the loop, to see how it all works
두번째 loop를 보라, 어떻게 일이 이루어질지 보.



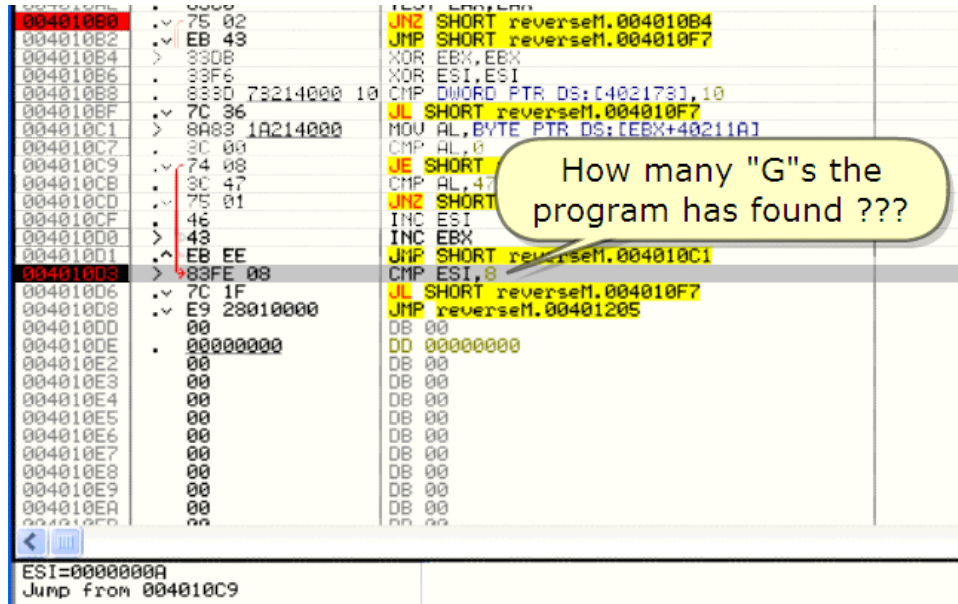
48d == 30h == 0 (ascii)

10진수 16진수 ascii

번역 주) 즉, 한마디로 ascii에서 파일의 끝은 0으로 쓰임. 꼭 이해하고 넘어가야 됨!

Be sure to understand this and look at the DEC2HEX2ASCII page for this. If you have no programming background, I'm sure this is a bit confusing at first.

물론, 이것을 이해했을 것이고 DEC2HEX2ASCII page를 봐. 만약에 programing 지식이 없다면, 처음에는 꽤 헷갈릴 것이다.



```
004010B0 75 02 JNZ SHORT reverseM.004010B4
004010B2 EB 43 JMP SHORT reverseM.004010F7
004010B4 33 08 XOR EBX,EBX
004010B6 33 F6 XOR ESI,ESI
004010B8 8B 30 73214000 10 CMP DWORD PTR DS:[402173],10
004010BF 7C 36 JLE SHORT reverseM.004010F7
004010C1 8A 83 1A214000 MOV AL,BYTE PTR DS:[EBX+40211A]
004010C7 3C 00 CMP AL,0
004010C9 74 08 JE SHORT reverseM.004010C1
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.004010C1
004010D3 83 FE 08 CMP ESI,8
004010D6 7C 1F JLE SHORT reverseM.004010F7
004010D8 E9 28010000 JMP reverseM.00401205
004010DD 00 DB 00
004010DE 00 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
ESI=0000000A
Jump from 004010C9
```

I hope you have understood all so far.

지금까지 네가 이해 했기를 희망한다.

So, let's run the ReverseMe till ...

... here.

여기까지 ReverseMe를 실행하라.

... 여기.

004010B0	> 75 02	JNZ SHORT reverseM.004010B4
004010B2	> EB 43	JMP SHORT reverseM.004010F7
004010B4	> 33 0B	XOR EBX,EBX
004010B6	> 33 F6	XOR ESI,ESI
004010B8	> 83 30 73214000 10	CMP DWORD PTR DS:[402173],10
004010BF	> 7C 36	JL SHORT reverseM.004010F7
004010C1	> 8A 83 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	> 83 FE 08	CMP ESI,8
004010D6	> 7C 1F	JL SHORT reverseM.004010F7
004010D8	> E9 28010000	JMP reverseM.00401205
004010DD	> 00	DB 00
004010DE	> 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

ESI=0000000A
Jump from 004010C9

0Ah == 10d indeed.

How many "G"s the program has found ???

0Ah == 10d indeed.

얼마나 많은 "G"를 찾았어?

0Ah == 10d 다.

Those BP's won't be needed any longer :)

이제 더이상 BP가 필요하지 않다.

No jumping to the badboy:

We jump to the goodboy :))

Badboy로 jump 하지 않는다.

우리는 goodboy로 jump 한다.

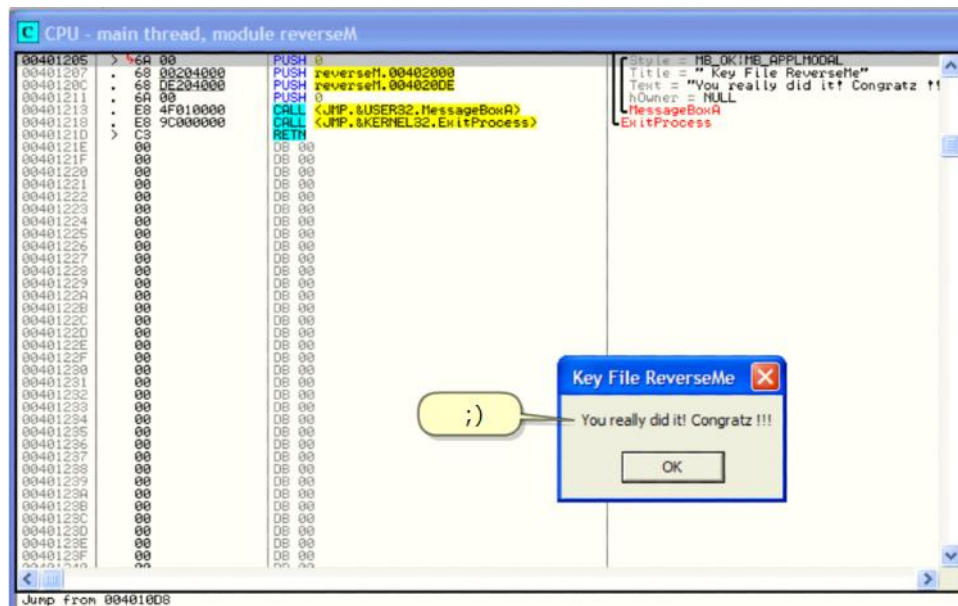
CPU - main thread, module reverseM		
00401205	> 6A 00	PUSH 0
00401207	> 68 00204000	PUSH reverseM.00402000
0040120C	> 68 DE204000	PUSH reverseM.0040200E
00401211	> 6A 00	PUSH 0
00401213	> E8 4F010000	CALL <JMP.<USER32.MessageBox>
00401218	> E8 9C000000	CALL <JMP.<KERNEL32.ExitProcess>
0040121D	> C3	RETN
0040121E	> 00	DB 00
0040121F	> 00	DB 00
00401220	> 00	DB 00
00401221	> 00	DB 00
00401222	> 00	DB 00
00401223	> 00	DB 00

Here we come !!!

Here we come !!!

Let's finally run the ReverseMe to see how things should be ...

마지막으로 어떤 일이 발생하는지 보기 위해 실행하자.



;)

6. Conclusion

Congratulations !!!

You just keyfiled your first ReverseMe.

축하해.

너는 ReverseMe를 막 끝냈다.

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

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

See me back in part 3 in this series ;)

나는 이 series의 Part 3 에서 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.