19.Debugger detected and anti-anti-techniques

2012 년 2 월 5 일 일요일
오후 9:43

1. Abstract

For obvious reasons, malicious software like virii and worms, but also protectors and standard programs in corporate anti-debugging techniques in their software.
Virii 와 worm 같이 악의적인 소프트웨어와 또한 protectors 와 standard program 은 그들의 소프트웨어에서 기업의 anti-debugging 기술이 있는 필요한 이유다.

Hence, if you ever want to be able to attain the summum of reversing (fighting virii), you will definitely need to master some anti-anti.
이런 이유로, 만약에 네가 리버싱에서 최고가 되고자 한다면(virii 대적), 너에게 분명히 anti-anti 를 마스터 하는 것이 필요하다.

In this Part 19, we will reverse a reverseme called "ReverseMe.A.exe", we will reverse a second reverseme called "Debugger Detected.exe" and we will also partially reverse a program.
이번 Part 19 에서, 우리는 "ReverseMe.A.exe"라 불리는 리버싱 할 것이다. 두번째로 "Debugger Detected.exe"라 불리는 것과 또한 우리는 부분적으로 프로그램을 리버싱 할 것이다.

All this to learn something about anti-debugging techniques and the defenses against them. For better comprehension and if you are a newbie, I advise you to first see all previous parts in this series before watching this movie.
이번에는 anti-debugging 기술과 그들을 방어하는 것을 배울 것이다. 좀 더 빠른 이해를 위해 네가 초보자라면 이 movie 를 보기 전에 네가 처음부터 이 series 를 볼 것을 추천한다.

The goal of this tutorial is to teach you something about a program's behaviour. In my search not to harm somebody, I found AntiSniff(version 1.01, last updated Sep 9 1999) from which we will only discuss the anti-debugging in the main exe.
이번 tutorial 에서의 목표는 프로그램들이 어떻게 행동하는지 알려주는 것이다. 내가 생각하는 바로 이것은 누구에게도 해가 되지 않는다. 나는 AntiSniff 버전을 찾았다. 우리는 중요한 exe 에서 anti-debugging 을 의논할 것이다.

There are also newer versions out. Taking a look in the specialized media, I also found this application to be "cracked" already. Here, this software is only chosen because it is ideal for this tutorial in reversing and it is targeted for educational purposes only.

그곳에는 또한 새로운 버전이 없었다. 특화된 Media 에 시선을 가져라. 나는 또한 이미 "cracked" 된 프로그램을 찾았다. 그래서, 이 소프트웨어를 선택됐다. 그 이유는 이것은 리버싱 tutorial 에 이상적이다. 그리고 이것은 교육적인 목적으로만 쓰일 수 있게 목표를 잡았다.

The four reverseme's in this tutorial are almost the same as the one I used for Parts 1&2. I will start this tutorial assuming you have seen and understood Parts1&2 in this series.
I hope you will exploit your newly acquired knowledge in a positive way.
In this matter, I also want to refer to Part 1.
이번 tutorial 에서 4 가지 reverseme's 들은 Parts 1&2 에서 쓰인 것과 거의 같다. 나는 이번 tutorial 을 시작하며 이 series 에 있는 Parts 1&2 를 이해할 것을 바란다.
나는 너에게 새로 얻은 지식을 긍정적인 방향으로 이용할 것을 바란다.
이번 강좌에서, 나는 Part 1 을 참조할 것을 원한다.

이것도 똑같음
Set your screen resolution to 1152*864 and press F11 to see the movie full screen !!!
Again, I have made this movie interactive. 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. Then the movie will skip the text and continue with the next screen. If something is not clear or goes too fast, you can always use the control buttons and the slider below on this screen.
He, try it out and click on the hotspot to skip this text and to go to the next screen now !!!
Click here as soon as you finished reading (on each screen!)
During the whole movie you can click this spot to leave immediately

2. Tools and Target

이것도 똑같음
The tools for today are : Ollydebug and... your brain.
The first can be obtained for free at
http://www.ollydbg.de

Unfortunately, no download for the brain ;)

Todays targets are ReverseMe.A.exe, DebuggerDetected.exe and AntiSniff 1.01.
오늘의 목표는 ReverseMe.A.exe, DebuggerDetected.exe 와 AntiSniff 1.01 이다.
I included them in this package for research. Also included are the slightly different versions B, C and D of the reverseme.A.
나는 그것들을 연구를 위해 이 패키지에 포함했다. 또한 조금 다른 버전 B, C, 와 D 를 포함했다.
These are meant for your own research as we will discuss during this Part 19. For your convenience and for testing, I also included the ollydbg plugin HideDebugger.dll

그것을 우리는 이번 Part 19 에서 의논 할 때 너의 연구에 의미 있는 것이다. 네가 테스팅하는 것을 편하게 하기 위해, 나는 ollydbg plugin HideDebugger.dll 을 포함했다.

3. Behaviour of the program

INFO :
First of all, I need to say that for this tutorial, you will need an "unprotected" (unpatched) Olly. Best is to download a fresh Olly without extra plugins.
제일 먼저, 이번 tutorial 을 위해 필요하다. 보호되지 않은 것. 가장 좋은 것은 Olly 를 추가적인 plugin 없이 새로 다운 받기.
Second : if you have seen Parts 1&2 in this series, do you still remember that we needed to keyfile a ReverseMe?
만약에 네가 이 series 의 Part 1&2 를 봤다면 아직까지 우리가 필요했던 ReverseMe 의 keyfile 을 기억합니까?
Well, the keyfile.dat we needed to make for that reverseme is the same as for the reverseme's A,B,C and D.
예, keyfile.dat 가 우리에게 필요합니다. Reverseme's A, B, C and D 를 위해 똑같이 만들어야 합니다.
I have put this keyfile.dat in the same directory as the ReverseMe.A that I have opened here. So, the ReverseMe should be registered, right?
나는 keyfile.dat 을 같은 디렉토리에 넣었다. ReverseMe.A 를 열어놨다. 그래서 ReverseMe 등록해라.

As always, it is extremely important to study your target well before attacking it. This may give you extra hints in how to solve the problem.
항상, 이것은 그것을 공격하기 전에 절대적으로 너의 목표를 공부하는 게 중요하다. 이것은 너에게 추가적으로 문제를 어떻게 푸는지에 대한 많은 힌트를 준다.
Let's do that together whilst stepping the code. Also whilst stepping, I will refresh your memory briefly : see part 1&2 if you have not yet.
우리 코드를 넘기며 함께하자. 또한 넘기는 동안, 나는 너의 기억을 새롭게 해주겠다. : 네가 아직 part 1&2 를 보지 않았다면 봐라.
I have already opened the ReverseMe and we are here at its EP. Let's start stepping and study the behaviour.
나는 이미 ReverseMe 를 열어놓았고 우리가 있는 곳은 EP 다. 코드 넘기기를 시작하고 행동을 배우자.

Is keyfile.dat present?
Keyfile.dat 가 있는가?

Yes!
Keyfile.dat is found
예,

Keyfile.dat 찾았다.

What is in the keyfile?
어떤 것이 Keyfile 이냐?

This is in the keyfile!
이것이 keyfile 에 있다.

Scroll up to see what's next
스크롤을 올리고 다음에 무엇이 있을지 보자.

We passed the test "Is the size of the keyfile at least 16d bytes ?"
우리는 "최소한 크기가 16 바이트가 되나? " 테스트를 통과했다.
Are there G's in the keyfile?
Keyfile 에 G 들이 있나?
Also remember that now comes a loop "How many G's are in the keyfile ?"
또한 기억해라. 이제 곧 오는 반복문에 "얼마나 많은 G 가 keyfile 에 있느냐?"
And this is the jump out of the loop, after all bytes have been verified for "G" char.
그리고 이것은 G 가 맞는지 확인한 후에 반복문을 벗어나는 jump 다.
Which jumps here to this call..
jump 를 하면 이 call 문에 도착한다.
Followed by "Are at least 8 "G"s present in the keyfile, what ultimately decides about Goodboy or Badboy.
최소한 G 가 8 개 이상 keyfile 에 존재해야 따라간다. 궁긍적으로 이것이 Goodboy 나 Badboy 를 결정한다.
So, let's breakpoint after the loop and run till BP
Breakpoint 를 설치하자. 루프와 실행을 BP 까지 한다.
We break in the BP. Continue stepping F8
우리는 BP 에서 멈춘다. 계속 하기 위해서 F8

Oops! What has happened?
How is that possible ???
웁스! 무슨 일이야?
어떻게 이것이 가능한가?

This can only mean there is an extra verification in this call, right?
이것은 하나로 귀결된다. 이 call 안에서 추가적인 검증을 한다는 뜻이다.
All right, let's debug this by restarting and stepping IN the call(F7)
괜찮다. Restart 후에 call 안으로 들어가며 Debug 를 하자.

I have done that but removed it from this movie for size till we...
나는 이미 완성했다. 그러나 그것은 이 movie 에서 용량 때문에 삭제됐다.

4. Finding the patches

...land again in the breakpoint. Now step IN the call F7
...to land here IN the call.
다시 BP 에 도착했다. 이제 F7 로 call 안으로 들어가자.
그래서 call 안에 도착했다.

Studying the code here immediately reveals something very dubious ...
이곳의 공부할 수 있는 code 를 즉시 매우 의심스러운 것을 드러낸다.
... but let's first see the rest of the code
그러나 첫번째로 나머지 code 들을 봐라.
Indeed, see that after this API IsDebuggerPresent, there is a compare for EAX
정말, 이것은 API IsDebuggerPresent 뒤에 EAX 에서 비교한다.
And such that if EAX is 1 ---> badboy !!!
그리고 EAX 가 1 이면 ---> badboy!!!
Badboy !!!

Now think with me ...
나랑 같이 생각해 보자.
...it can only mean that this API is guilty, right?
API 가 의심스럽다. 그렇지?

INFO :
I suspect you know by now what I'm going to do? Indeed, look for some more info on the API IsDebuggerPresent.
나는 네가 알고 있는 것을 의심한다. 이제 내가 무엇을 할 것 같은가? 정말, API IsDebuggerPresent 에 대해서 조금 많은 정보를 보자.
However, this time, let's also find out what to do if you can not find what you want in Win32.hlp, or if you need more info that offered by Win32.hlp
네가 많은 정보를 필요로 할 때 Win32.hlp 는 제공해 줬다. 하지만 이번 시간에는 무엇을 할지 찾았다. 만약에 네가 원하는 것을 Win32.hlp 에서 찾지 못했다면

I have already done the necessary meanwhile : I have searched MSDN (search on the site)
나는 이미 "IsDebuggerPresent"에 대한 필요한 정보를 MSDN 검색을 통해 해결했다.
http://msdn.microsoft.com/library

For "IsDebuggerPresent"

Well, I suppose this doesn't need further explaining. We know enough already but ...
... for completeness.
좋아, 나는 이것보다 더 자세한 설명은 필요하지 않다고 생각한다. 우리가 이미 충분하다는 것을 안다.
완전하다.

INFO :
The return value for IsDebuggerPresent is set in EAX
IsDebuggerPresent return value 는 EAX 에 set 된다.

Conclusion: Olly gets detected ---> EAX is set to 1 --> the compare makes us jump to the badboy!
결론: Olly 를 찾았다. ---> EAX 를 1 로 set 한다. ---> 비교는 badboy 로 jump 하게 만든다.

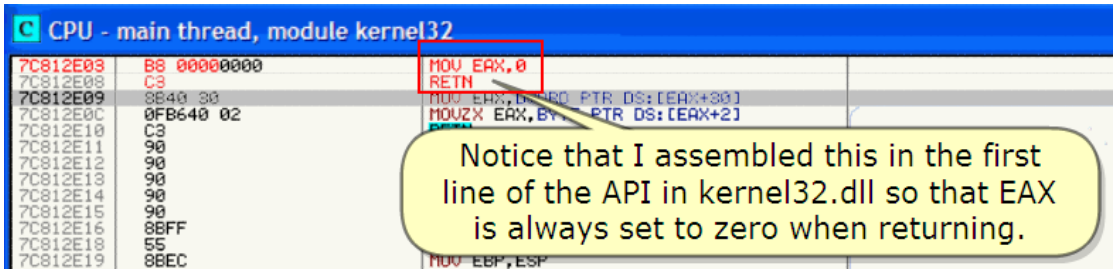
The detection can be very easily avoided however and as always, there are different possibilities.
검출을 매우 쉽게 회피할 수 있다. 그러나 항상 가능한 것은 아니다.

For example: NOP everything till the return
Return 까지 모두 NOP
Or only NOP the JE etc etc
아니면 JE 를 NOP

But let me show you to avoid the checking in the API in kernel32.dll as a general method (in case there are other IsDebuggerPresent checks in the ReverseMe).
그러나 나는 너에게 kernel32.dll API 속에서 회피하는 범용적인 방법을 보여줄 것이다.(이번에는 IsDebuggerPresent 를 ReverseMe 에서 검증한다.

5. Patching and testing

!
Press enter to follow in kernel32.dll
kernel32.dll 따라 들어가자.



Notice that I assembled this in the first line of the API in kernel32.dll so that EAX is always set to zero when returning.

나는 API kernel32.dll 의 첫번째 줄에서 조립했다. 그래서 returning 할 때 EAX 는 항상 0 으로 setting 된다.

INFO :

Of course you would need to reassemble this at each restart: Olly doesn't keep patches nor BP's (unless changing .ini file) in dll's.

물론 너는 Olly 를 재실행 할 때마다 재조립 할 수 있다 : Olly 는 dll 의 patche 를 기억하지 못한다.(.ini 파일을 바꾸기 전까지)

The method can be used for every API in a general way. Sure, you need to know which API "does" the checking.

이 방법은 모든 API 에 적용되는 범용적인 방법이다. 물론 네가 API 를 확인하기 위해 API 가 하는 것을 알아야 한다.

Also, I need to inform you that there are many other API's that can be used to check for debugging..... Too many to list here.

또한, 나는 많은 API 들이 debugging 에 사용되는지 정보를 얻기 위해 필요하다. 많은 목록들이 여기에 있다.

The API I'm showing you here is very basic and isn't often used any longer. Fortunately, many skilled reversers help the starter by making anti-anti-debugging plugins for our debugger.

여기에서 보여주는 API 들은 매우 간단하고 자주 사용되지 않는다. 다행스럽게도 처음 시작할 때 리버서에게 anti-anti-debugging plugins 이 많은 도움이 된다.

I have included the Hidedebugger.dll plugin which "repairs" the IsDebuggerPresent detection. Thanks to the author. Try it out with the included Reverseme's but also without the plugin !!!

나는 Hidedebugger.dll 은 IsDebuggerPresent 를 찾는다. plugin 을 포함했다. 제작자에게 감사한다. 노력했다. Reverseme's 에 포함하기로 또한 plugin 은 없이.

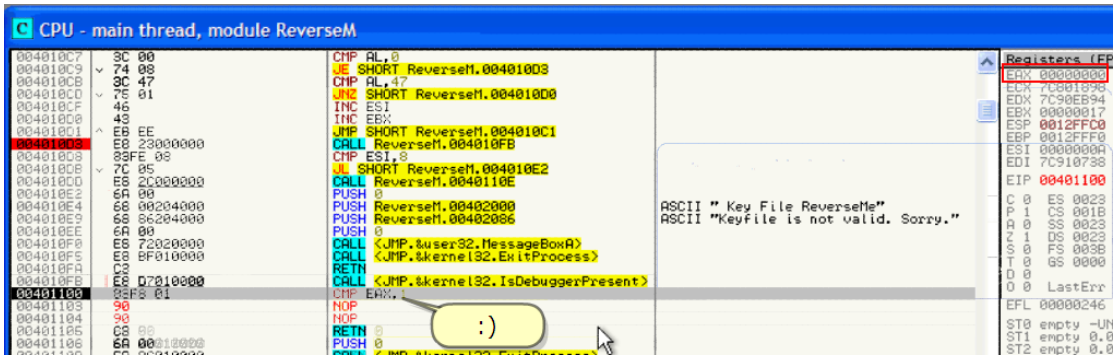
REMARK :

Some software or packers/protectors (see later Parts in this series) are able to detect changes and / or breakpooints in API's !!!

주목 :

약간의 소프트웨어나 packer/protector 들은 API 안에서 변화하는 것들이나 BP 을 탐지할 수 있다.

So, let's return pressing the minus sign on our keyboard to see all this happen in the code.
But before stepping the code to see what happens, let me also show another method (nopping the JE). Follow along.
그래서, 돌아갈 때 minus 를 누르면 코드에서 어떤 일이 일어나는지 봐라.
그러나 code 를 넘긴 후에 무슨 일이 생기는지 봐라, 나는 너에게 다른 방법을 보여주겠다. 따라와라.



And now, step the API
이제 API 를 실행
:)
And notice that EAX is zero, hence what I assembled in kernel32.dll did its job!
그래서 EAX 는 zero 로, 그리하여 이것이 kernel32.dll 안에서 했던 일이다.
So, even without NOP'ing the JE, we would not have jumped to the badboy!
그래서, JE 를 NOP'ing 하는 것이 필요 없다. 우리는 badboy 로 jump 하지 않겠다.
...and notice that we return from the call to run the Goodboy!
그리고 우리는 call 을 한 곳에서 Goodboy 로 돌아간다.

OK! Fine!
오케이! 좋아!

INFO :
I have included four slightly different ReverseMe's in this package. Try out and study them in Olly to learn: all win run registered outside Olly (put keyfile.dat in the same dir of course !), but they will all give another result in Olly !!!
나는 이 package 안에 4 가지의 약간 다른 ReverseMe's 를 포함했다. 도전하자 Olly 안에서 배우자: 우리는 Olly 밖에서 등록할 수 있다.(keyfile.dat 를 같은 폴더에 넣자) 그러나 우리는 다른 방법을 쓰겠다.
A: is the one we studied together (invalid file)
멍청한 방법이다.(검증되지 않은 파일)
B: Olly complains : Debugger was unable to process exception
Olly 는 불평 : Debugger 는 process exception 에 연결될 수 없다.
C: The program simply exits!
프로그램은 간단히 존재한다.

D: Olly complains : Don't know how to step...
Olly 불평 : 다음에 어떻게 하는지 모르겠다.

All this only to show you some possibilities and showing you that you can expect anything if your debugger gets detected!
너에게 약간의 가능한 방법을 보여주겠다. 만약에 너의 debugger 가 검출 능력을 가졌다는 것을 예상할 수 있다.

INFO :
Always remember to reset to your initial settings and options in Olly. For example, if you have removed the auto analysis for the ReverseMe's ---> change again for the next target or you will miss certain things!!!
항상 reset 할 때 너의 Olly 안에서 초기 setting 과 option 을 기억해라. 예를 들어, 만약에 네가 auto analysis 를 삭제했었다면 ---> 다시 바꿔라. 다음 목표를 향해 아니면 너는 정확한 것을 놓치게 된다.

5. Debugger Detected.exe

Run this second target outside Olly and it will say that it didn't detect a debugger. However, if you run this target in an unprotected debugger, you see ....
.... This.
실행해라. 다음 목표를 그리고 그것은 너에게 그것은 debugger 를 찾지 못한다고 말해 줄 것이다.
그러나, 만약에 네가 보호되지 않은 debugger 에서 목표를 실행했다면 어떻게 되는지 봐.
.... (이렇게 돼. Your debugger is detected!!!)

INFO :
I hope it is clear that in a real program, the author is not going to help you, but he will play all kind of tricks with you.
That's why i have included ReverseMe B,C and D for more info and some possibilities. Try them!
나는 그것이 실제 프로그램에서 정확하기를 희망한다. 제작자는 너에게 도움을 주지 못한다. 그러나 그는 여러 종류의 트릭을 너와 함께 사용할 수 있다.
그래서 많은 정보와 약간의 가능성을 위해 ReverseMe B, C and D 를 포함했다. 그것들에 도전해봐.

Ok. Let's debug this reverseme. Restart and step F8 to take an overview first.
Ok. Reverseme 를 debug 해보자. 재시작 하고 F8 을 눌러 처음을 봐.

I have cut from this movie stepping 8 times F8 and BAM !!!
나는 이 movie 에서 건너 뛰는 것을 잘랐다.

Right. Perhaps you wonder now how to find where the problem can be? After a few steps, we are already detected and we have only stepped a dialogbox !!!
Let me show you, but I assume you will begin to know what I will do first. Right ???
맞다. 만약에 네가 문제를 어디에서 찾는지 원한다면? 몇 번의 코드를 실행 시켜보고, 우리는 이미 발견했다. 그리고 우리는 무조건 dialogbox 를 지나가야 된다!!!
너에게 보여주겠다. 그러나 나는 네가 처음 시작할 때 내가 무엇을 했는지 알 것이라고 생각한다.

Did you guess it ???
추측할 수 있겠니?

Now, study this. Can you find what I could want to show?
이제 공부하자. 내가 어떤 것을 보여줄지 찾았니?
This is what interests me. The procedure for the dialog box. Everything is in there of course! Let's see it in the code
날 흥분하게 만든다. 이 procedure 는 dialog box 를 위해 존재한다. 모든 것이 코스에 존재한다.
코드를 보자!
That's it! Let's BP there
맞아. BP 를 걸자.

Ok. We break in the BP. Now, scroll down for better view and step F8 to take an overview.
우리는 BP 에서 멈췄다. 스크롤을 내리는 게 보는데 편하다. 그리고 F8 로 관점을 갖자.
Aha, we land in User32.dll Press Alt-F9 to return to user code
아하! 우리는 User32.dll 안에 도착했다. Alt-F9 를 눌러 return to user code 로 가자.
Bam! We land again in the BP.
우리는 다시 BP 에 도착했다.
No problem continue F8
문제없다. F8 을 눌러라.

INFO :
You will find that Olly sometimes has difficulties returning from ring0 (finding the first step back in the program's code).
너는 찾는다. Olly 가끔씩 어렵다. Ring0 에서 returning 할 때.(찾아봐 첫번째 줄로 돌아오는 코드를)
You may need to set mem BP on code section to find it in such case.
너에게 필요할 것이다. code section 에서 Memory BP 를 찾기 위해서 다른 case 에서
Bam! When stepping this call, you can hear the beep ----> debugger detected.
Call 을 할 때 너는 beep 를 들을 수 있다. ----> debugger 를 찾았다는 것을
So, it's in this call! BP the call and restart to run till BP after removing the old BP.
그래서 이 call 에 old BP 는 삭제하고 BP 를 설치하고 재시작 하자.
Let's restart. I want to break here indeed!

다시 시작하자. 나는 여기에서 break 걸릴 것을 원한다.

Right! I suppose you understand it must all be in the call ???
맞아! 나는 네가 이 call 안에서 있는 것을 이해할 것을 생각한다.

And press F7
F7 을 눌러
And step F8 again to continue our search.
그리고 다시 F8 로 검색하자.

Study the code better here. You see lstrcmpiA and other suspicious things. Let's find some help.
Code 를 공부하기에는 이곳이 낫다. 봐라.lstrcmpiA 그리고 다른 의심스러운 것들을. 약간의 도움이 될만한 것들을 찾자
BTW, I told you already where to find help if Win32.hlp doesn't offer any. So I...
... have prepared some. Remember the link for MSDN?
내가 너에게 이미 win32.hlp 에서 도움을 받을 수 없다면 어떻게 하는지 말한다.
그래서 나는 약간의 준비를 했다. 이 MSDN link 를 기억하지?

I can not say this better ;)
이것보다 더 좋게 말 할 수 없다.
:(
I suppose this is clear?
This API "makes a snapshot" of all running processes ...
Let's see more in the code
명확하지? API 는 모든 실행되고 있는 프로세스에 스냅샷을 만든다.
Code 에서 좀 더 보자.

Ok. Step F8 to make the snapshot.
Ok. F8 을 누르면 스냅샷을 만든다.

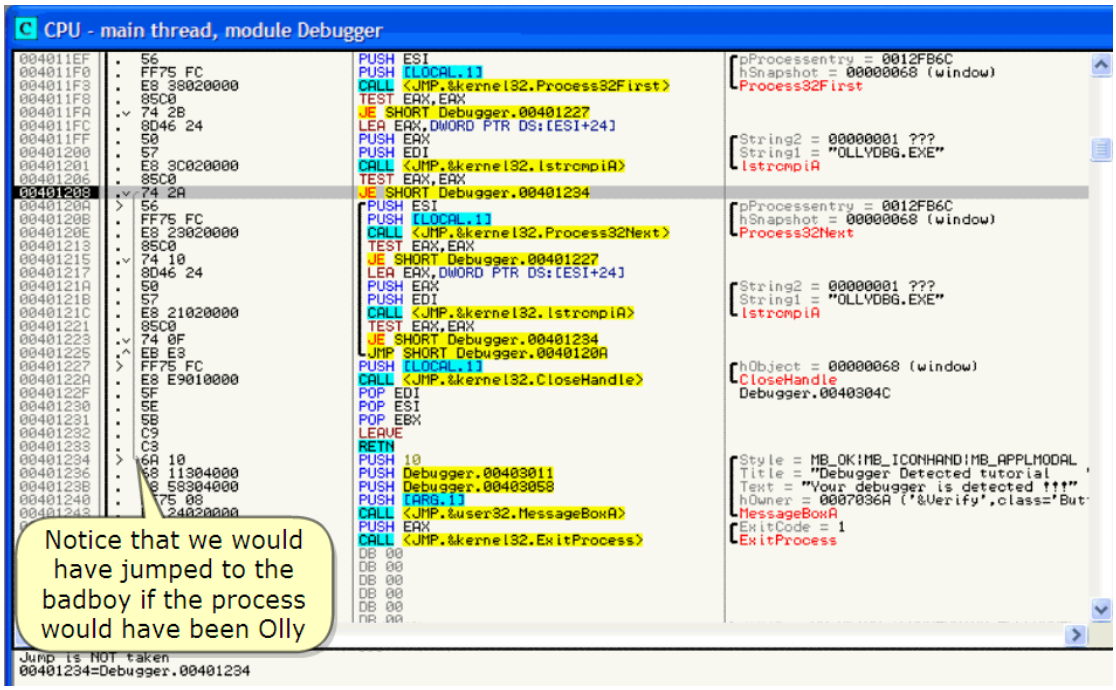
Copy the string to search for in EDI
And the reverseme wants to search for this string
문자열을 검색을 위해 EDI 에 복사한다.
그리고 reverseme 는 이 문자열을 검색하기 원한다.

Find the first process (this is self explaining, no Win32.hlp needed)
첫번째 프로세스를 찾는다(스스로 설명한다, Win32.hlp 필요없다)

We don't jump because the first process is found

우리는 jump 하지 않는다. 왜냐하면 첫번째 프로세스를 찾았기 때문이다.

See what processes will be compared but are not equal(yet). Step F8 and scroll down to see.
봐라. 프로세스가 무엇을 비교했는지 그러나 아직은 서로 같지 않다. F8 을 누르고 보기 위해 스크롤을 밑으로 내려라.



Notice that we would have jumped to the badboy if the process would have been Olly
But it is not finished : the reverseme picks the next process (Process32Next) and loops to compare them all to search for Olly. Just follow
만약에 프로세스가 Olly 를 찾았다면 badboy 로 jump 했을 것이다.
그러나 이것이 끝이 아니다 : reverseme 다음 프로세스(Process32Next) 를 고른다. 그리고 loop 를 돌아 Olly 를 찾기 위해 비교한다. 따라가자.

It is clear that the loop searches all running processes for OllyDbg.exe and if found ...
...badboy.
명확하다. 이 loop 는 모든 실행되고 있는 프로세스 중에서 OllyDbg.exe 를 찾는다. 그리고 만약에 찾는다면 badboy...
...and we loop to pick the next process to verify
그리고 우리는 next process 가 정확한지 loop 한다.
And we only can jump out of the loop here to continue to the goodboy if all processes were verified
그리고 우리는 loop 를 벗어나는 모든 프로세스가 검증이 됐다면 이 곳에서 goodboy 로 jump 한다.

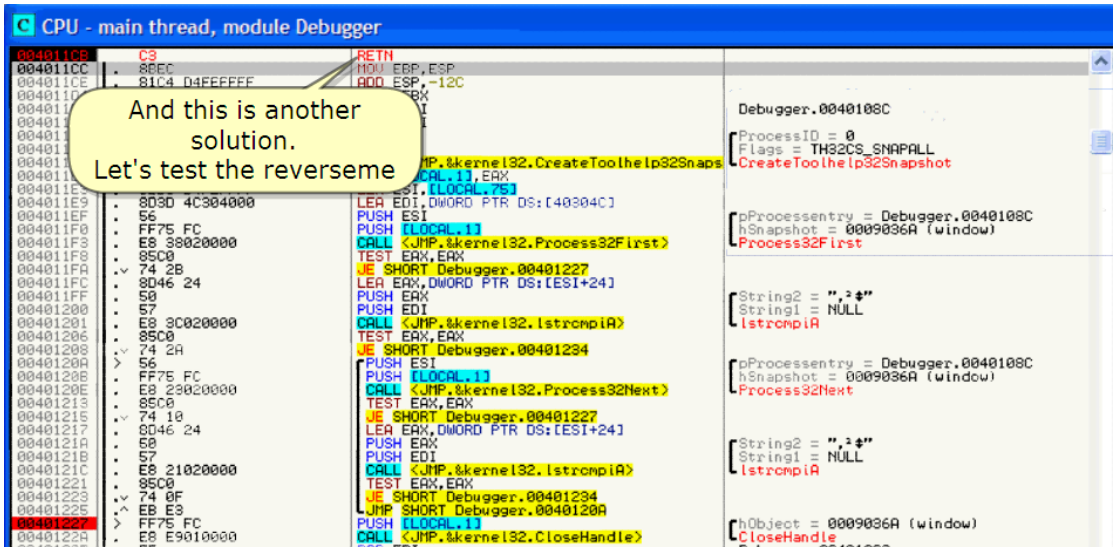
Let's verify some of the processes and see if Olly gets detected.

몇몇의 프로세스들을 검증한다. 그리고 Olly 를 찾는 것을 봐라.
Look here ...
Olly detected !!!
Let's see it.
Olly 를 찾았다. 봐라.
Jump to badboy!
:(
Badboy 로 jump 한다.

Now, how to avoid this detection?
이제, 어떻게 탐지을 회피하겠나?
Well, there are always more possibilities. Let me show you two of them, but restart first.
...and so we land here at EP.
(Cut from movie to reduce size)
항상 여러가지 가능성이 있다. 2 가지를 보여주겠다. 그러나 restart 가 먼저다. 그리고 우리는 EP 에 도착한다.
(movie 사이즈를 줄이느라 삭제했다)

Remember the call it all happens in?
Well, we could simply NOP the whole call of course or ...
이 이 모든 일이 일어났던 call 을 기억하냐?
우리는 간단하게 NOP 를 하면 되나

REMARK :
Notice that I breakpointed the beginning of the verification routine before restarting the reverseme in Olly
주목 :
나는 탐지 루틴의 처음에 BP 가 걸렸다.



And this is another solution.

Let's test the reverseme

다른 해결 방법이다.

Reverseme 를 테스트 하자.

;)

CreateToolhelp32Snapshot detection beaten, and so land in the next program.

CreateToolhelp32Snapshot detection 이겼다. 그리고 우리는 다음 프로그램에 도착한다.

6. Behavior of the third program

INFO :

I imagine it is not always that easy for a newbie to find exactly what API causes the anti-debugging. That's why I have included one more target in this Part 19.

나는 이것이 항상 초보자에게 쉽지 않다는 것을 상상한다. 정확히 API 가 왜 anti-debugging 에 쓰였는지 찾는다. 그래서 내가 하나 이상의 목표물을 Part 19 에 포함시켰다.

So, I will debug its anti-techniques without even searching what API causes them. In this third target, I will ONLY look into the side-effect of our debugger being detected: the program simply won't run.

그래서 나는 anti-technique 를 API 검색없이 debug 할 수 있다. 이번 3 번째 목표물에서, 나는 debugger 가 탐지하는 부수효과를 보기만 할 것이다. : 이 프로그램은 간단히 실행되지 않는다.

Indeed, you certainly understand that a program will react differently outside the debugger <==> inside the debugger when the debugger gets detected!!

정말, 정확히 이해해야 한다. 프로그램이 debugger 밖에서 다르게 반응할 것이다. <==> debugger 안에서는 debugger 를 탐지한다.

When detected, the program will do whatever the author wants it to do: it may crash, exit, display weird messages, etc etc

검출할 때, 프로그램은 제작자가 누구인지 상관없이 : 충돌, 종료, 이상한 메시지 보여줄 것이다.
BTW, if you have understood all about the anti-debugging of all ReverseMe's hereby included, you may first want to try this third target on your own.
네가 anti-debugging 에 대해 이해했다면, 너는 첫번째로 세번째 목표물을 도전해 봐.
Don't search for what API causes the problems, just try to find where it goes all wrong and patch it.
API 문제의 원인을 검색하지마, 어디에서 잘못 됐는지 찾고 그리고 패치 해라.
It really isn't all that difficult. Good luck!
이것은 정말로 어려운 것이 아니다. 행운을 빌어!

And so, we land here at the EP of the program that I have previously opened in Olly --> removed from movie for size.
Let's run the program and study its behavior together
그래서 우리는 프로그램의 EP 에 도착했다. 이미 Olly 로 열어놓았다. Movie 용량을 줄이기 위해서

Oops! What is that? But the program ran fine outside the debugger!
And what address is that ???
웁스! 무슨 일이야? 프로그램은 debugger 밖에서 정상적이다.
그리고 주소가 뭐야?

So, let's debug this inconvenience ;)
Now, think with me, what can be done about this?
We could look what happened before! Perhaps the Call stack can learn us something?
불편한 상황에서 debug 하자.
이제 나와 같이 생각하자, 무슨 일이 발생한 거야?
우리는 무슨 일이 일어났는지 볼 것이다. 만약에 Call stack 에서 무엇을 배웠냐?

INFO :
In cases like this, Olly will seldom (never in fact) be able to give us info by the Call Stack. But let's verify
이번 상황처럼, Olly 의 Call Stack 에서 좀처럼 우리에게 정보를 주지 않는다.
그러나 검증해보자.

Let me meanwhile already point to the system stack: this is why Olly can't continue: Olly is supposed to read an inexistent address!
이미 System stack 은 중요하다: 이것은 Olly 진행할 수 없다: Olly 는 존재하지 않는 주소를 읽는다고 생각한다.

Like I said, let's verify the call stack now

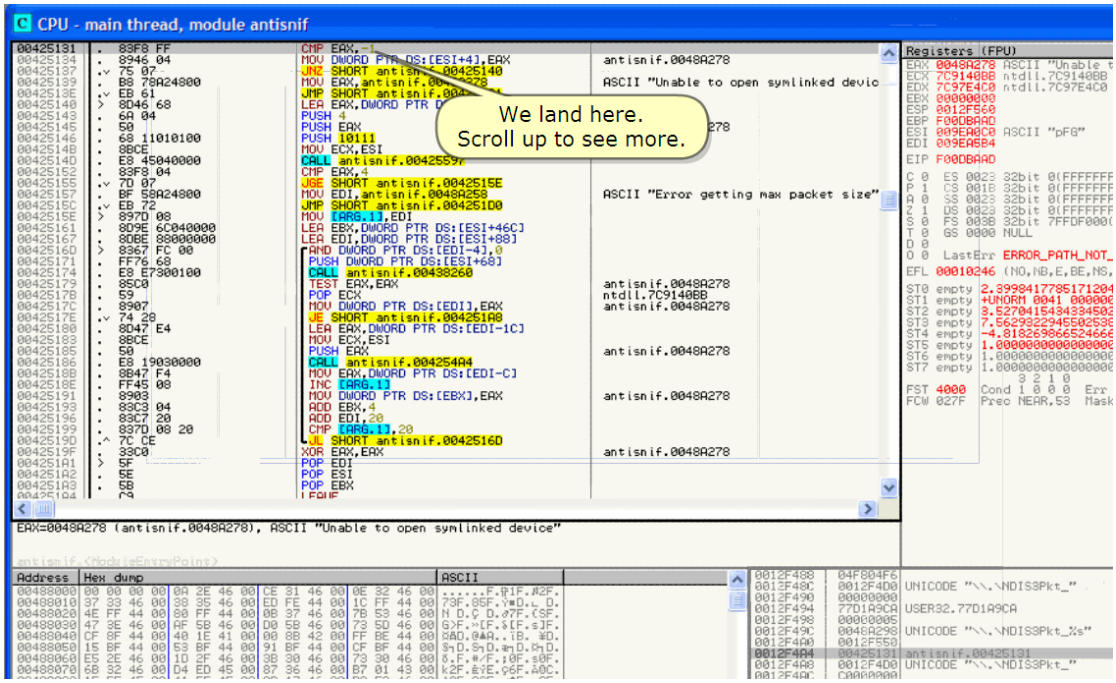
이제 검증하자.

The call stack is useless !!
Call stack 은 쓸모없다.

INFO :
This is one of the opportunities to find what happens using the more advanced functions of Olly like TC(Trace Call == run trace).
다른 가능성을 어떤 일이 일어나는지 많은 발전된 Olly function 에서 TC 와 같은 것을 찾아보자.
However, I always try some other possibilities first because TC can take really long : "Run trace requires plenty of memory, in average 16 to 35 bytes per command depending on the mode, and is very slow. On a 500-MHz processor under Windows NT it can trace up to 5000 commands per second".
그런, 나는 항상 여러 가지의 가능성을 시험해 본다. 왜냐하면 TC 는 정말로 오래 걸린다. "실행 행동을 추적하기 위해서 충분한 메모리가 필요하다. 보통 16~35 명령어는 mode 에 의존적이다. 그리고 매우 느리다. Windows NT 이하 500Mhz processor 에서 추적 하는데 1 초에 5000 명령어 걸린다."
(Ollyhlp) Another fine possibility is to use the basic stepping method extensively described in part 05 in this series. In short: restart the program and then step F8 till error pops up at call --> BP this call --> restart and run till BP --> step F7 in call --> step F8 again till following popup and continue like this till you find the lines where it happens.
다른 가능성들은 사용하기 위해 기초적인 방법들을 Part 5 에서 광범위하게 설명한다.
간단하게 : 재시작 하고 F8 을 눌러 error 창이 뜰 때까지 call 한다. --> call 에 BP --> 재시작 해서 BP 까지 실행 --> F7 로 들어가고 --> 다시 F8 로 popup 을 따라 간다 그리고 그 문제가 일어나는 곳을 찾는다.
Try it out on this target and you will see that it works very fine indeed! In this particular case however, I will show you that the system stack can often bring us at the right places in no time :) Just follow along...
이번 목표물에 도전해라. 그리고 이번 부분에서 너는 매우 좋은 것을 볼 것이다. 그러나 나는 너에게 정확한 장소에서(시간이 없을 때) system stack 을 자주 가져오는 것을 보여줬다. 따라와라.

7. Finding the patches

Mmmm, there is perhaps something here just above the FOODBAAD's ? Let's try it...
그곳이 만약에 무엇이라면 FOODBAAD 위에 있다면 이다. 한 번 해보자.
단지 FOODBAAD 의 위에 뭔가가 아마 있다.



;)

We land here.

Scroll up to see more

우리는 여기에 도착했다.

좀 더 많이 보기 위해 스크롤을 올리자.

First set a BP as visual aid to remember where we landed here, and also in the beginning of this routine to see if we can break here BEFORE the error message

첫번째 BP 는 시각적인 지원이다. 우리가 도착한 곳을 그리고 보기 위한 첫번째 루틴을 기억하기 위해 만약에 우리가 이곳에 BP 를 걸 수 있고 error message 를 본다.

;)

Yep! We break before the error. We are probably not far from the error message?

예, 우리는 에러가 보이기 전에 멈췄다. 우리는 아마 에러 메시지에서 멀리 있지 않을 것이다.

INFO :

At this point, it is very well possible that the program has already detected quite long it is being debugged!

Like said, I have not verified that: it doesn't even interest me!

I only plan to remove the anti-debugging actions taken by the program ...

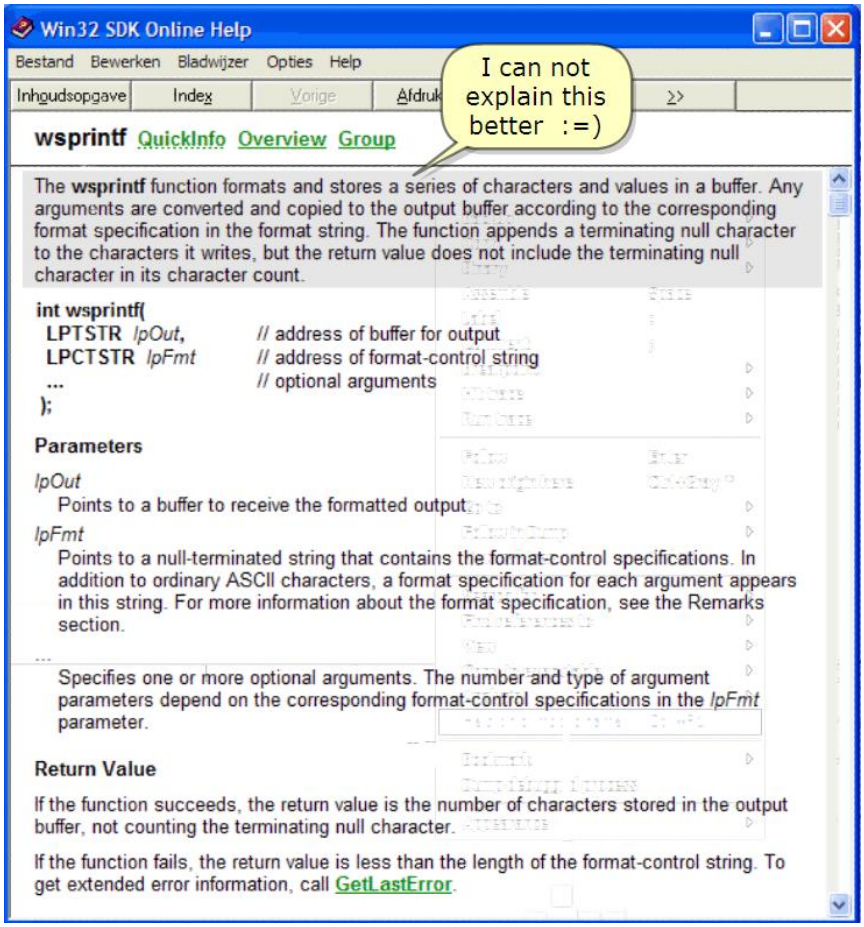
여기에서 중요한 것은, 매우 좋은 가능성이다. 프로그램은 이미 긴 시간 동안 디버깅을 위해 발견됐다.

난 검증하지 않았다. : 그것이 나조차 흥미를 끌만한 것인지.

나는 오직 프로그램에서 anti-debugging 을 삭제할 계획이다.

Step F8 to see what happens...
F8 을 눌러서 무슨 일이 일어나는지 봐라.
Aha!!! Have you seen it too ?
아하!!! 너는 봤냐?

Right !!!
So, who is guilty ???
Thanks Olly for being so helpful with the comments ;)
You can probably imagine what I will look up now ? ;)
의심스럽지?
올리야 고마워. 코멘트를 달아줘서
너는 아마 내가 지금 무엇을 보고 있는지 상상 할 거야 .



I can not explain this better :=)
나는 이것보다 더 좋게 설명하지 못해.
:)
;)

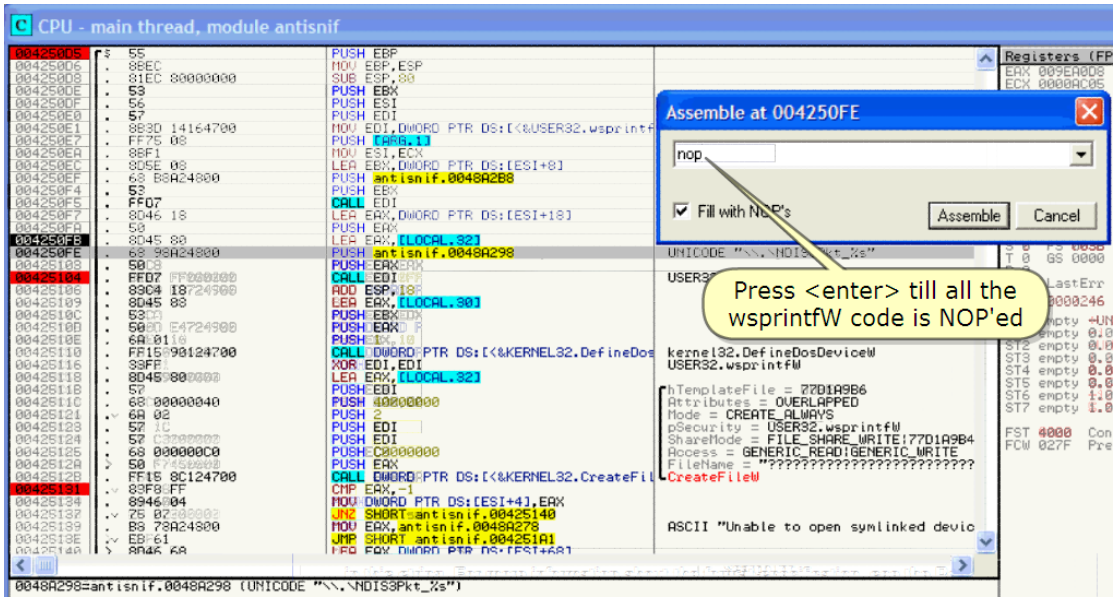
:(
Have you understood that in short : the program writes the buffer at 009EA0C8
... and will then jump a little later in the middle of the FOODBAAD's :=(
너는 이해했냐? 이 짧은 것을: 프로그램은 009EA0C8 버퍼에 쓴다.
그리고 FOODBAAD's 의 중간으로 jump 한다.

Restart and let's find out what we can do about it ...
재시작 하고 내가 무엇을 해야 하는지 찾아보자.
:)

8. Patching the software

Ok. We land again in the BP.
Step F8
Ok. 우리는 다시 BP 에 도착했다. F8 을 눌러.

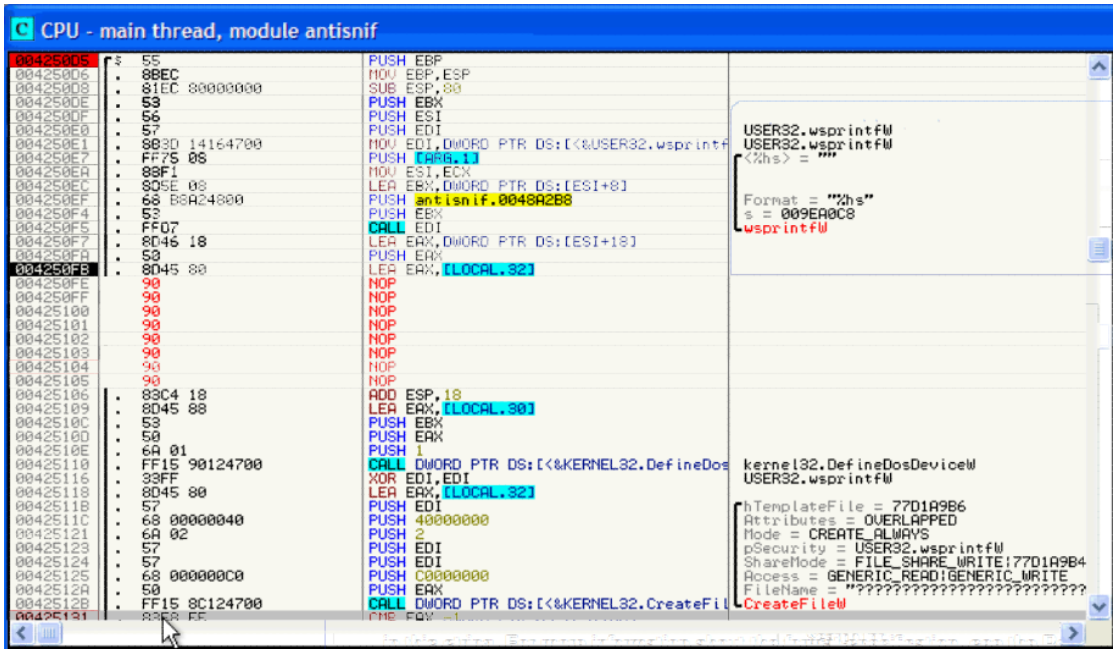
The disaster is prepared!
참사는 이미 준비되었다.
If you want, just go see in the dump window what's in the buffer at 9EA0C8
네가 원한다면 dump window 에 있는 9EA0C8 를 보겠다.



Now think with me.
What if we don't let the program effectively write the FOODBAAD ??
...and just completely NOP the foul play?
나와 같이 생각해 보자.
우리가 하지 않으면 효과적인 FOODBAAD 를 쓸 수 없다.

그리고 완벽히 NOP 더러운 짓을 하자.

번역 주) PUSH antisnif.0048A298
PUSH EAX
CALL EDI
모두 NOP 으로



Press <enter> till all the wsprintfw code is NOP'ed
And see what the program thinks of these changes
Wsprintfw 는 NOP 이 된다.
그리고 봐라. 프로그램이 어떻게 바뀌는 지를.

The program runs also in Olly!
프로그램 또한 Olly 안에서 실행됐다.
That's what we wanted all right!
이것이 우리가 원한 것이다.

Return to Olly
올리로 돌아가자.

And save the changes to file to test better in Olly.
변화된 것들을 테스트 하기 위해 파일에 저장하자.

9. Saving and testing

Press <enter> to save under different name
다른 이름을 넣고 Enter
:)
And test the program !
프로그램 테스트

The program runs fine in the debugger.
프로그램이 debugger 안에서 잘 실행
Mission accomplished.
미션 완수

In this part 19, the primary goal was to learn about some simple anti-debugging and anti-anti techniques. I hope you understood everything fine and I also hope someone somewhere learned something from this.
See me back in part 20 ;)
이번 Part 19 에서 중요한 anti-debugging 과 anti-anti techniques 을 간단히 배웠다.
나는 네가 모든 것을 좋게 이해했기를 희망한다. 그리고 누구든지 이곳에서 무엇이든지 배웠으면 좋겠다.
Part 20 에서 보자.

The other parts in this series are available at
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)
이 tutorial 에서 특별히 시간을 아낄 수 있다고 여겨진다.

2012-02-07 오전 12:00