19.Debugger detected and anti-anti-techniques

2012년 2월 5일 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.

Virii와 worm 같이 악의적인 소프트웨어와 또한 protectors와 standard program은 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-anti를 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 19에서, 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 movie를 보기 전에 네가 처음부터 이 series를 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 tutorial에서의 목표는 프로그램들이 어떻게 행동하는지 알려주는 것이다. 내가 생각하는 바로 이것은 누구에게도 해가 되지 않는다. 나는 AntiSniff 버전을 찾았다. 우리는 중요한 exe에서 anti-debugging을 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 Media에 시선을 가져라. 나는 또한 이미 "cracked" 된 프로그램을 찾았다. 그래서, 이 소프트웨어를 선택됐다. 그 이유는 이것은 리버싱 tutorial에 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 tutorial에서 4가지 reverseme's들은 Parts 1&2에서 쓰인 것과 거의 같다. 나는 이번 tutorial을 시작하며 이 series에 있는 Parts 1&2를 이해할 것을 바란다.

나는 너에게 새로 얻은 지식을 긍정적인 방향으로 이용할 것을 바란다.

이번 강좌에서, 나는 Part 1을 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

1. **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 D를 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 19에서 의논 할 때 너의 연구에 의미 있는 것이다. 네가 테스팅하는 것을 편하게 하기 위해, 나는 ollydbg plugin HideDebugger.dll을 포함했다.

1. **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 tutorial을 위해 필요하다. 보호되지 않은 것. 가장 좋은 것은 Olly를 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 series의 Part 1&2를 봤다면 아직까지 우리가 필요했던 ReverseMe의 keyfile을 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.dat가 우리에게 필요합니다. Reverseme's A, B, C and D를 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.dat을 같은 디렉토리에 넣었다. ReverseMe.A를 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&2를 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 ReverseMe를 열어놓았고 우리가 있는 곳은 EP다. 코드 넘기기를 시작하고 행동을 배우자.

Is keyfile.dat present?

Keyfile.dat가 있는가?

Yes!

Keyfile.dat is found

예,

Keyfile.dat 찾았다.

What is in the keyfile?

어떤 것이 Keyfile이냐?

This is in the keyfile!

이것이 keyfile에 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 16바이트가 되나? " 테스트를 통과했다.

Are there G's in the keyfile?

Keyfile에 G들이 있나?

Also remember that now comes a loop "How many G's are in the keyfile ?"

또한 기억해라. 이제 곧 오는 반복문에 "얼마나 많은 G가 keyfile에 있느냐?"

And this is the jump out of the loop, after all bytes have been verified for "G" char.

그리고 이것은 G가 맞는지 확인한 후에 반복문을 벗어나는 jump다.

Which jumps here to this call..

jump를 하면 이 call문에 도착한다.

Followed by "Are at least 8 "G"s present in the keyfile, what ultimately decides about Goodboy or Badboy.

최소한 G가 8개 이상 keyfile에 존재해야 따라간다. 궁긍적으로 이것이 Goodboy나 Badboy를 결정한다.

So, let's breakpoint after the loop and run till BP

Breakpoint를 설치하자. 루프와 실행을 BP까지 한다.

We break in the BP. Continue stepping F8

우리는 BP에서 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 Debug를 하자.

I have done that but removed it from this movie for size till we…

나는 이미 완성했다. 그러나 그것은 이 movie에서 용량 때문에 삭제됐다.

1. **Finding the patches**

...land again in the breakpoint. Now step IN the call F7

…to land here IN the call.

다시 BP에 도착했다. 이제 F7로 call 안으로 들어가자.

그래서 call 안에 도착했다.

Studying the code here immediately reveals something very dubious …

이곳의 공부할 수 있는 code를 즉시 매우 의심스러운 것을 드러낸다.

… but let's first see the rest of the code

그러나 첫번째로 나머지 code들을 봐라.

Indeed, see that after this API IsDebuggerPresent, there is a compare for EAX

정말, 이것은 API IsDebuggerPresent 뒤에 EAX에서 비교한다.

And such that if EAX is 1 ---> badboy !!!

그리고 EAX가 1이면 ---> badboy!!!

Badboy !!!

Now think with me …

나랑 같이 생각해 보자.

...it can only mean that this API is guilty, right?

API가 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 IsDebuggerPresent에 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.hlp는 제공해 줬다. 하지만 이번 시간에는 무엇을 할지 찾았다. 만약에 네가 원하는 것을 Win32.hlp에서 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 value는 EAX에 set된다.

Conclusion: Olly gets detected ---> EAX is set to 1 --> the compare makes us jump to the badboy!

결론: Olly를 찾았다. ---> EAX를 1로 set한다. ---> 비교는 badboy로 jump하게 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

Return까지 모두 NOP

Or only NOP the JE etc etc

아니면 JE를 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 IsDebuggerPresent를 ReverseMe에서 검증한다.

1. **Patching and testing**

!

Press enter to follow in kernel32.dll

kernel32.dll 따라 들어가자.

시스템 생성 대체 텍스트: 더Q끼-耐inth隅d,moduleker麗132
노－
7C81ZE03
'… q' 「.뻑
켰〕812En9
7C8!ZEOC
7C812ㅌle
7C812Ell
7C812EiZ
7c8!2E13
7C8!ZE14
7c812ㅌ15
7C812Ei6
7c812Ei8
기〕812Ei9
BSO000eO00
b닌
SB4O80
OFB64O02
C3
9e
9e
9e
9e
90
SBFF
SS
SBEC
"--1l티珖．e
문楡눙
"OVZXE댜＜,
PTRD숭：[ㅌ自기＋80】
'(rRDS:[ERX+2]
NㅇtiCethat1aSSembledthisinthefirst
"neoftheAPIinkernel32.dllSOthatEA×
isalwayssettㅇzerㅇwhenreturning.

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.dll의 첫번째 줄에서 조립했다. 그래서 returning할 때 EAX는 항상 0으로 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 Olly를 재실행 할 때마다 재조립 할 수 있다 : Olly 는 dll의 patche를 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 API에 적용되는 범용적인 방법이다. 물론 네가 API 를 확인하기 위해 API가 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 API들이 debugging에 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 API들은 매우 간단하고 자주 사용되지 않는다. 다행스럽게도 처음 시작할 때 리버서에게 anti-anti-debugging plugins이 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.dll은 IsDebuggerPresent를 찾는다. plugin을 포함했다. 제작자에게 감사한다. 노력했다. Reverseme's에 포함하기로 또한 plugin은 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/protector들은 API안에서 변화하는 것들이나 BP을 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 minus를 누르면 코드에서 어떤 일이 일어나는지 봐라.

그러나 code를 넘긴 후에 무슨 일이 생기는지 봐라, 나는 너에게 다른 방법을 보여주겠다. 따라와라.

시스템 생성 대체 텍스트: ．『 - -
一一--一--一一一- 一 一 一7프FE-－뜨J==
더Cp【j-閥inthrㄼ비m여luleReverseㅆ
hUnPPt'''
l盼ESESEBESED
OC41-110그:
O〔4010:굽
oe4I-jle」
P apt'0뱁len딩n킹aOL a킬쌤‘2
．■‘ rr 7를TT
E욕〃눌l 갸놀· 11LLS츤구S
EE
23000000
FE08
0S
ZCOOaonO
0e
00204000
862e4Oe0
ee
72020000
eFO100e〔l
댜IPR【－..
JESHDRTR슬ver룔eh.oe4Oie03
댜IPRL,'-
JZSHORTRev'rs층h.OO4eleDe
!NCESI
汪"CE테l
JnP하읫】RTRev순t驪슨h.004010CI
댜졈LLR·닌∥＄e".eO4e!OFB
C"PESI,：ㅌ：
JL하緘取TRev솥r＇솥什．eO4OieEZ
驪孔LRever＄해．O64OlleE
PUSHO
PUSHRever‘얘，eO4OZeOO
댔jSHR。v.r'.".00402086
PUSH自
公聊긴糖坪간닸Uㄴ｀깁긔
口7一＝7Fr7T
87'
04凸p
V ^
II
II
"Ke
하（이겯
「t
[큔
Rever'ehe"
.n。tva!id.s。rr보·"
『 .1' '
기7솥〔一b
oe000e17
0.12FFCO
0012FFFO
OO00000R
7C910738
0야홑el!eo
ESOOa3
CS0e!B
550023
DSee23
FS0e38
GS0eee
CnLL<J麟’.잃c.r^.192.1$Oebugg.tPr'$.nt>
LastErr·
Oe00024乙
-Ue．이
畛叩叩
eee
虜쐴―
…면卜卜（샌트헷sers흰少塋무엔Bo"m
LㅐL'iJ『】r·乙kerheI4<.L써Lr「r。Ce룔52
RETH
n"
r'
rrPrr
nC던-U
C"11H
nZ이00요O
FS01
R「뀨졉
PUSHO
＇놈f탭．·씸 ,
舟
紬
'一
쵸
鄒Oe一
743c祗4643EBEe貂7cES胡66臼얘E8臼臼姃驢沁沁c8배f-pe
旺1eL呱m陛班leEleFleFleF旺四臣臣口닌
勿柏柏衲40枋柏柏物4(l盼낍汕勿汕衲一

And now, step the API

이제 API를 실행

:)

And notice that EAX is zero, hence what I assembled in kernel32.dll did its job!

그래서 EAX는 zero로, 그리하여 이것이 kernel32.dll 안에서 했던 일이다.

So, even without NOP'ing the JE, we would not have jumped to the badboy!

그래서, JE를 NOP'ing하는 것이 필요 없다. 우리는 badboy로 jump하지 않겠다.

...and notice that we return from the call to run the Goodboy!

그리고 우리는 call을 한 곳에서 Goodboy로 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 4가지의 약간 다른 ReverseMe's를 포함했다. 도전하자 Olly 안에서 배우자: 우리는 Olly 밖에서 등록할 수 있다.(keyfile.dat를 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

Olly는 불평 : Debugger는 process exception에 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 debugger가 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 setting과 option을 기억해라. 예를 들어, 만약에 네가 auto analysis를 삭제했었다면 ---> 다시 바꿔라. 다음 목표를 향해 아니면 너는 정확한 것을 놓치게 된다.

1. **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 debugger를 찾지 못한다고 말해 줄 것이다.

그러나, 만약에 네가 보호되지 않은 debugger에서 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 D를 포함했다. 그것들에 도전해봐.

Ok. Let's debug this reverseme. Restart and step F8 to take an overview first.

Ok. Reverseme를 debug 해보자. 재시작 하고 F8을 눌러 처음을 봐.

I have cut from this movie stepping 8 times F8 and BAM !!!

나는 이 movie에서 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 dialogbox를 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 procedure는 dialog box를 위해 존재한다. 모든 것이 코스에 존재한다. 코드를 보자!

That's it! Let's BP there

맞아. BP를 걸자.

Ok. We break in the BP. Now, scroll down for better view and step F8 to take an overview.

우리는 BP에서 멈췄다. 스크롤을 내리는 게 보는데 편하다. 그리고 F8로 관점을 갖자.

Aha, we land in User32.dll Press Alt-F9 to return to user code

아하! 우리는 User32.dll 안에 도착했다. Alt-F9를 눌러 return to user code로 가자.

Bam! We land again in the BP.

우리는 다시 BP에 도착했다.

No problem continue F8

문제없다. F8을 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. Ring0에서 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 section에서 Memory BP를 찾기 위해서 다른 case에서

Bam! When stepping this call, you can hear the beep ----> debugger detected.

Call을 할 때 너는 beep를 들을 수 있다. ----> debugger를 찾았다는 것을

So, it's in this call! BP the call and restart to run till BP after removing the old BP.

그래서 이 call에 old BP는 삭제하고 BP를 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

F7을 눌러

And step F8 again to continue our search.

그리고 다시 F8로 검색하자.

Study the code better here. You see lstrcmpiA and other suspicious things. Let's find some help.

Code를 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.hlp에서 도움을 받을 수 없다면 어떻게 하는지 말한다.

그래서 나는 약간의 준비를 했다. 이 MSDN link를 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? API는 모든 실행되고 있는 프로세스에 스냅샷을 만든다.

Code에서 좀 더 보자.

Ok. Step F8 to make the snapshot.

Ok. F8을 누르면 스냅샷을 만든다.

Copy the string to search for in EDI

And the reverseme wants to search for this string

문자열을 검색을 위해 EDI에 복사한다.

그리고 reverseme는 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. F8을 누르고 보기 위해 스크롤을 밑으로 내려라.

시스템 생성 대체 텍스트: r'
l너댜끼-耐inth醜비耐
uleDebugger
7드FC
380200e0
C8
2B
4'24
PUSHㅌ51
PUSH[LOCnㄴ．1]
(J"P.
ERX.E
各kern순【32.PrㅇO슬S륭32F孔rst》
山塔
rP炚0開뜨헤t珊＿드－뻣프rp6C
l므눼．p$h으L7U..UUUb븝'닐내。。니.
‘「r。0．룝5珊Z「Lr룔r
S닌ORT며b니99
RE沁＜.OUORDP
하춈EnX
.r.Oe401227
TR05:tESI+24】
낡sH鯖SH爵sH擺L
대TE无LEPUPU邙TEf門盼
r038pCr
LFFFFFF
뱁흩뱁횹．솝’욥'．쵸·．솥ㅓ■‘
ㅓ‘뱁‘뱀‘뱀홑.．춈ㅓ요ㅓ■‘·1''''
eoapenUn!
444444
OOnpoq
00습UO츤난
a40
계40
J40
[
'나、92· OO8eoeel仝7?
r'「．91="OLLVt）텨」．D<E"
3Ce20000
C0
2R
L'trc"ptR
r·．L
7드FC
23OZeoee
C0
le
4524
EDI
《JHP.하〔·rn'132.t'trcnptR>
EnX,E아《
ORlrD.b니99er.OO401234
ESI
tUr루L.l]
《J엥P.各k·rn't32.PrOC.'!32N상ㅒt》
뉴__논11'
12FB6C
卜하、ap:hot=
e0
oeoeee
68t니LndO니）
Pr。c'''32N·ㅒt
TESTERX,En닝
JESHORlrD·bU99er.084elZ騙，
LERERX,DUORDPTRDS:〔ESI+24〕
PUSH
PUSH
rLn92
rinql
e0OeO0el수爭？
"OLLVOBG.EXE"
2le2000o
CO
eF
E3
7드「C
Egol0000
CnLL
TEST
E아엡
EDI
<J伊．各kernet32.【strc히plR>
rst
嶸
ronp
TERk,En닝
S볐》Rlroebu99er.eO4el
SㅐOF첩TDebugqer.eO4O
234
12On
tLI勵nL.효】
<JHP.흽kerne【32.CIO료eHandl.>
robJ.=t므OO0eee68《낚tndo닐】
C【OseHand【e
Debugger.004O3a4C
飇sHLLPP
bU99.r.0盼홍03011
沚髓r·eO4盼e貂
뱌》．毓‘$er32.衿·＄'ag·Boㅐ습＞
X
HP.各kern.【32.E써itPrㅇC상룔t>
:t,'e="B－아이∥〔IC（劇HRH미H〔얀PL"coRL
ㅜit.e="ODebu99erOet.ct·dtut。rLa별
T슬ㅒt="VOurdebuqgerL륭deteoted"빵”
h〔니n'r므UUU7U36R(-戮U.r.f,'.CIa''='sut
"e'｀슬g·8。Hn
;L卜〔中」e프1
EㅐirProce$'
r■■■■■■Lr■L
배 柏D.De[P<J태＜J
EE -Un、oen·n
P翩꺼SHSHSHsHLLSHU。0。0。自
PUSPOPOPOLE麗PUPUPU円8円m嵋D6DSDSDerlP
1e
l!3040e0
583e4e00
7드08
·bF8rD4De78rp46F854De멩rsr구4BFSFEB93nHS
5FE878드5E87灌딛澾匿湲녕攫灌匯陽낟555乙褐驢麗
·…갼··…큔＞…>.··…엇·＞·!·…＞·
…BE3S7CHBC·‘3rO7nHFdp,'2604ZOBe
紛泌21212121212122222222222223盼盼23盼乙2324
-犯紛牝4e牝4e4e4e犯牝4e柏40柏4e40柏4e柏4e牝4e헬
[朋朋騁開腑朋的애閔觸騁에騁開開騁에00卵開朋開開一
NㅇtiCethatWeWㅇ미d
havejumpedtothe
badboyiftheprocess
wo미dhavebeen이―y
J""pL흥nOI'‘낟en
00401234=Deb니99er.00401234

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 Olly를 찾았다면 badboy로 jump했을 것이다.

그러나 이것이 끝이 아니다 : reverseme 다음 프로세스(Process32Next) 를 고른다. 그리고 loop를 돌아 Olly를 찾기 위해 비교한다. 따라가자.

It is clear that the loop searches all running processes for OllyDbg.exe and if found …

...badboy.

명확하다. 이 loop는 모든 실행되고 있는 프로세스 중에서 OllyDbg.exe를 찾는다. 그리고 만약에 찾는다면 badboy...

...and we loop to pick the next process to verify

그리고 우리는 next process가 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 loop를 벗어나는 모든 프로세스가 검증이 됐다면 이 곳에서 goodboy로 jump한다.

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

몇몇의 프로세스들을 검증한다. 그리고 Olly를 찾는 것을 봐라.

Look here …

Olly detected !!!

Let's see it.

Olly를 찾았다. 봐라.

Jump to badboy!

:(

Badboy로 jump한다.

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. 2가지를 보여주겠다. 그러나 restart가 먼저다. 그리고 우리는 EP에 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 call을 기억하냐?

우리는 간단하게 NOP를 하면 되나

REMARK :

Notice that I breakpointed the beginning of the verification routine before restarting the reverseme in Olly

주목 :

나는 탐지 루틴의 처음에 BP가 걸렸다.

시스템 생성 대체 텍스트: 더cP卜閥inth隅비mγ紬leDebu盼er
·．·비口【Cl■. 8「FC
HE[rㅓ
『10U
EBP,ESP-
ESP,一1욘C
Andthis15ar1Other
S이Utiㅇrl.
DebIJgger.O04Ole8C
Let'stestthereverseme
.各毓ern'132.Creat.T。。【hetp32SnaO$
r학얀卜‘I틔둣瑗＾一－．∼.
I〔L.’二느·히尸디＇>＇므마fHL'
'Lreate'00LheIp驪귑bnap드hot
pFrㅇ。essentr닛=
ㅑ:hap'hㅇt=00a9
PtOCe륭흥32「Lr륭t
o.bugg.r
836R〔닐i
.004ele8C
ndo니)
긔trin92="，솥샅．
드tringi=HULL
lstrcf,p'R
r■．LL r■■I
00401
00401
00401
00401
004el
00401
00401
004ei
e0401
0e401
004ei
00401
00401
e0401
ee4ei
00401
00401
eO401
e0401
e040i
00401
00401
Oe4el
e0401
OO4ei
OO4ei
00401
00401
00401
e0401
00401
00401
00401
댜L.i],라눼Y
.LLOGfIL.75】
9D4C9e4000
7드FC
38OZO0ee
C0
2B
4'24
3CeZe60O
【＿EnED!.DUOPDpTPD송＝【409041:그
PUSHESI
PUSH[L以거L.!1
댜겸LL<JhP.읾‘.rne132.PtOO슬＄륭32Fit륭t》
TESTEnX.E댜《
JESㅐ이랙TO밗》니go.r.00401227
LERER넝，山＇아귑DPTRDS:【ES!+24】
PUSHEnX
PUSHEDI
驛졈LL＜야뿌．"(ern.'32.tstrc畛礖）
TESTEnX,티甥
JESHORTDeb니99零r.OO481234
29020000
PUSㅐ
PUSH
CnLL
L(rnL·l]
J千．各k.rneI32.PrOC．룔S32Neㅒt》
[
「．Fr。야＇$ent'』=O。bugg'r.ee4eleeC
빅드n'p탉졉。t:0009036RI니indo닐ㅎ
PrOCe5SaaHe냠t
r'7첵E
lH
eZ rD
츤닙 ''
D6F8540878한큐4'pLS
SO6FE878rDrOEa7r큐FE
··…갼··…>.>··
SCCO
r··‘ '
TESTERX
JES냈）RT
L텨눕EnX,
PUSHEnX
PUSHEDI
.EnX
0ebu99er.6a4e1227
DUORDPTRDS:〔ESI+24】
21020060
C0
eF
E3
7SFC
E90】0000
CnLL
TEST
JES
JP.各ketnel32.l룝ttC혀pin>
沈rln92=",1솬’
Ctrtn91:"ULL
l$ttc혀Oin
'bu99·r.e94e1234
De以‘gger.O04O12OR
,En
O걍
r．■■■．■■■■■■·．■·
慨IEglEFIFelF3綴IFclFF槪2e6硼獗輟戮鞭ZL122S뜨
4「- 견R
nL.i]
.U《．rn.132.CIO륭．Hahdl.>
沈놀
lO
Ject2000903히눼(닐Ind。낚＇
흗eHandLe
翩RTOROC飇
<EHOSH[L십－-
4DaC7솝or구4BF^Dl
7^QrDr구E87EFEI

And this is another solution.

Let's test the reverseme

다른 해결 방법이다.

Reverseme를 테스트 하자.

;)

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

CreateToolhelp32Snapshot detection 이겼다. 그리고 우리는 다음 프로그램에 도착한다.

1. **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 API가 왜 anti-debugging에 쓰였는지 찾는다. 그래서 내가 하나 이상의 목표물을 Part 19에 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-technique를 API 검색없이 debug 할 수 있다. 이번 3번째 목표물에서, 나는 debugger가 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 debugger를 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-debugging에 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.

API문제의 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 EP에 도착했다. 이미 Olly로 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 stack에서 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, Olly의 Call Stack에서 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 stack은 중요하다: 이것은 Olly 진행할 수 없다: Olly는 존재하지 않는 주소를 읽는다고 생각한다.

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

이제 검증하자.

The call stack is useless !!

Call stack은 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 function에서 TC와 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 TC는 정말로 오래 걸린다. "실행 행동을 추적하기 위해서 충분한 메모리가 필요하다. 보통 16~35 명령어는 mode에 의존적이다. 그리고 매우 느리다. Windows NT 이하 500Mhz processor에서 추적 하는데 1초에 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 5에서 광범위하게 설명한다.

간단하게 : 재시작 하고 F8을 눌러 error 창이 뜰 때까지 call 한다. --> call에 BP --> 재시작 해서 BP까지 실행 --> F7로 들어가고 --> 다시 F8로 popup을 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 stack을 자주 가져오는 것을 보여줬다. 따라와라.

1. **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 FOODBAAD의 위에 뭔가가 아마 있다.

시스템 생성 대체 텍스트: 더CpU*mdinthr繃d,漱넌ule해tisnif
L륭ters
沔‘媤沔
(F댔D
7『祗르
江914eB8
7C97ㅌ4C0
「．t.:
n十.
11，·Unab【et
〔L.7lDg!40B8
【l.7C97F4I:[O
OOeOO
ZF드E.0
P
e
gEn띵
qSCI!..pF6''
hLnU4
nHCbD
aD뱁‘npg
nO00nUnl
n캉nlFd컁八U
愕燻嘯
^
00425134
0042드137
0042드!39
0042드13ㅌ
0e425140
00425143
ee425145
0042드!46
OO42514B
00425!4D
004251드2
aO425lSS
0042드!57
00425！드C
Oe425!SE
00425161
a0425!'7
Oe42드!6D
0042드171
0e425!74
a0425179
ee425!7B
Oe42드!7C
0042드17E
Oe425!80
00425183
a042드185
0042드!86
0042518B
0042518E
00425191
a0425!93
e042드!96
e0425i99
0042519D
0042드!gF
004251pi
0042FIC「
46a홍
07
7아긔24800
6l
4668
O4
nOV
JHZ
nOV
J枳
LEn
댔jSH
PUSH
댔，SH
戚）V
더겸LL
CrIP
JGE
nOV
J干
h()U
D배〕RD
SHORT
:〔ESI+4】,ERX
눕노요042E140
antL륭hif.0048R278
antl
Ep<,.nti$nif.E
SHORT솔nrL꿸nl「
E아뱁．DIJ·（)RDPTR
'4
테X
RSC!1"Unab【．
W이andhere.
t。叩＊n$,hl내h논dd.vic
닙花
110!e10o
CE
4504Oe08
수쟈솥힁무튕수학수튕
·驪*슬‘＇■‘ ·뜨'·■탭
bbbbb
솔‘2222U
3各0333N
83F804
.v7D07
BFS聊겹24800
.vES72
> 8971)88
SDgE6CO40O0e
SOBE88eO0eoe
> 8367「C00
FF7668
ESE780e100
11 '
ESI 、
i룝nLf.OO425
4
Scr이―uptoseemore·
EIPFI〕OD8pRO
89花BSE88D해騶6888E8
·>.·α＞·!…
T혜ti'n‘「.e042드15E
술ntL륭nL「.0048R258
T솔ntL맬nl「.0042드100
.1],EDI
RSC11"Err。rg*ttLng＂래p.Ck.t$죙2."
002:
00IE
송
EpE타뱁．DUORDPTRDS졀【ESI+46C】
EREDI.DUORDPTRDS:[ES!+88〕
RHD미．ORDPTRDS.【EDI,4].0
PUSHD난ORDPTRDS:[Egl+68]
聊겸LL.nt1tn쵱「.00438260
TESTE댜＜,티경X
POPECX
∥이룔며．OF궉DPTR【）S:[EDI】,E댜＜
JESHORT솔ntL$nLf.OO42C！습e
LERERX.D뻐DRDPTRDS:tEDI-IC〕
＂以，ECX,ESI
PUSㅐE대＜
d겸LLantL료hLr.O042드率눕4
0(FFFFFFF
0(FFFFFFF
0(FFFFFFF
e(FFFFFFF
7FFDFe00(
oeao
－니CS拓DSFSGs
l·수’'R'Re
∥ECX綴SHOEDI祇
LL
Last탸rr
aot'e뱁‘en킬eno
CPq뱃LSTDO
EFLOe01024'
ERROR-PnTH--llOT-
(NO,re,E,BE.HS,
anti흥h쵸「.0048n27ㅎ
ntdl【.7C914OBB
anti료n1f.004며괘278
솝nti$n길r.Oa4驪놈278
티궈X.DUDRD
ERRB·i]
며．ORDPTR
ESX.4
e
11.긴3
PTRDS'[ED!-C]
05:【E마뱁】.E댜《
ant15h쵸f.004SR278
STeenpt,
ST!”、pt,
STZehpty
ST3ehpt,
ST4e벼）탭，
STSehpt보
ST6enpt,
ST7ehpt,
FST4eoe
FCU027F
2.399읜홑177851712e넉
+U"ORH004!oeee0E
3.드270415434334드02
7.562932294560258E
-4.8!e269e6652466E
l.e0000O0eO0e00OOE
i.O000eoeeoe000O0e
i.e000000000O00e0E
32!0
COnd!e0e Err
Pre。NERR,53 ＂술튤k
OU嘴OUDDDD飇
HUt'"nHnHC
ㅐ:薛－v
ll· 89。7-
1■.v7428
1. 8D47E4
"· 畔CE
11. 50
i■. E819030000
". 8B47F4
11. FF45es
". 89B3
", 83C304
"· 8那220
"· 83709820
"·＾7cCE
■. 33C8
004요「
00.1련
<
》 SF
5E
드8
「q
'JㄴSHI〕RT솔n탭1tni「.00425160
넜）RE댜《．ERX
더PEOI
더）PESI
POPEBX
I「081「
antLshi「.004驪겸278
=8048R2
Ianti톨ni「.0e48R278),
Unablet。。pen'，衿【LnkeddevLce"
Rddre요51HeHdU
nSC11
。＞F凹lD
H6다tJ
00488OZel4E
D·J7F-(SF·
‘른'「멩.r밉7『멩■닙큔｀』돌■·훨'7솝．· · 뀀‘른'『1.＇륵＇『맬
00488030
00488040
00488050
00488060
Oe488070
."[F.5[F.룔
R..iB.
:
D.＊·D.머D.
-
0012F498
oe12F4엇D
001ZF4Rn
·띠튑F쀼뷰
0el크F4驪〕
n더17F4q「
00000005
004ER298
001ZF드드O
6.F.뽈／F.:O「．
k辦「．蠟9E.96F.
oe!2F4D0
「nn盼n댜더O
UH1CODE"\\,\hD!S鱗公t_Z$"
■■■■b솥驢．의“2됴1요뿟」■■F날｀
UHICODE..＼、．\"DIS珊坐t_"
00dodpaO仕p^
0000仕Odp^
664463'
444444·
馹6D麗BF秘이一
7873FFcF柏數
000000^
IU-"Iod긴01냐'-'
662466'
444444'
1 ㅡ7BBFe'r
-=3r구8B33'
Fㄴ·∼Boe1B7'
'-rJODaU93S^
叱·刪·朋硼螂觸騁鞭
卄h4446시4446拓一
FBEFFDL
Fr구IBZEr
OFaO3D4·
8R45뱁‘D·
00000d룁^
onpdUnUOO凸
444644444646一
FF麗SFBFZE麗l
7Fr댜r규BL
4CIE6·

;)

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 BP는 시각적인 지원이다. 우리가 도착한 곳을 그리고 보기 위한 첫번째 루틴을 기억하기 위해 만약에 우리가 이곳에 BP를 걸 수 있고 error message를 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-debugging을 삭제할 계획이다.

Step F8 to see what happens…

F8을 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 .

시스템 생성 대체 텍스트: 卜Z〕－-－一－---－一－-－一―--
之wi"32SDKo"lineㅐeIP
BesbndBewerkenBlad助ㅉ여bes峀b
-．버여d沁Pga시 .nd랜 l 蜘醜
曰回因
시dl
Z>
wsprirltfp미야助f-Q흐旦넌쁘f흐쁘p
ICannot
exPlainthis
better:=)
The새恪l)rintf釉nctionformatsandstores枓나iesofcharactersan小뎨uesinabuffe,Any^
argum야胎arecom心rtedandcoPiedtotheoutputbufferaccordingtothecorrespending
formats畔clfiCationintheformatstnngThef曲ctionaPpendsaterminatingnulㅣcharacte『
tothecflaractersitwrites,buttheretumvaIuedoesnotIncIudetheterminatingnuIㅣ
CharactefinitsCharacterCOUnt
intwsPrintf《
LPTSTR눋Out, 〃addressofbufferforoutPut
LPCTSTRIP戶鮮 〃addressofformat￡ontr이strlng
〃叩bonalarguments
);
Parameters
IPOut
Potntstoab마fertorecelvethe沁rmatted이仲ut
IPR따
Pointstoanull.termillatedatringthatcontainsthefo仰at乙ontro-sPecifiCation도ㅣn
additiontoordinaryAScl-characters.aformatspe助衍ationforeachargumentaPpears
inthi$stringFormoreinformationab이∥thefo떼atsPec耐atio『1·seetheRem鹹‘
$eCt·On.
SpeafiesoneormoreOptlonalarguments-Thenumberandtypeofargument
paramatersdependonthec。rrespondingformat《ontrolspec甬cationsInthe눋Fmt
parameter
RetUrnValUe
ㅣfthefUnctlonsucceeds,thereturn
bLlffer.notcountlngtheteαnlnatlng
IfthefUnctlonfails.theratu『『Ivalue
gete)dendederrorinformatlo八cal-
value15thenumberofcharacterssioredintheoutPut
n미ICharaCter
islessthanthelengthofthefbrmat《ontr이stringTo
GetLa던〔『『Qr

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's의 중간으로 jump 한다.

Restart and let's find out what we can do about it …

재시작 하고 내가 무엇을 해야 하는지 찾아보자.

:)

1. **Patching the software**

Ok. We land again in the BP.

Step F8

Ok. 우리는 다시 BP에 도착했다. F8을 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 window에 있는 9EA0C8를 보겠다.

시스템 생성 대체 텍스트: 驢
＂녑inthrㄼd.moduleanti'nif
P- '-
F- 굽뻑IJ
rL-
닌b
,D二
너「F
55
8으E〔
」■L
-7瞞
너卜ters
--―ㅕ서―----------
UI-1ㅓ'bUU「
004a드OEO
004250EI
L:IO42COE쮸
OO42FOEq
004슨COEㅣ그
OO4a드OEF
004a드0F4
Ll'4 :너'
슝IEC요더00盼000
E3
螺6
57
se兀I쵸4료（·47EI()
「「7C(l승
98FI
SDSE08
68B:5R24800
52
F尸07
804618
50
PUSH
'!l:l'
SU6
댔觸H
PUSH
PUSH
"0쟈
PUSH
"l:lt
LEn
PUSH
PUSH
댜뵙L
LEp
PLI드H
EBP
뜨캬F',ESP
eE「‘,；조〕
EBX
ㅌ51
Frll
PTRD돈:〔、戮난5ER32.니크pfL「t
F:IJ
-＂··옵
·ll ·
Da
b鹹
꺼E.:::.
r,D괴꼬〔PTf.:LI::18」+8]
rUr'·＝닙LL
rj PLrJ
첩i'nLf.OO4爵楗BS
'’스
EDI
하盼［.댜lll:IF:[[F기궉D무：【e31+18]
E나－
9D4「80
6e99R2빕겸뱁■■L
5e
FFD7 낙奴腕眈댜勿
83C41072뺘딜eo
8[)4583
l뜨eF뺘：,[LocnL,82]
―다僧누잃보뾔f.0·4硼98
l琿며됴r뜨
IROOE＇ㅏ7'gF
ASSembleat《）《》4250「［
致솟
.
첵「·
I
-As絡mb“ㅣcancel-
며a425·Fㅌ
Ll.4
Te
USER
34rp67
·TTTTT
55555
너．:길 낑〕E
0042610,
O0426lOC
0042味teo
eO42CieE
0042尸110
Oe425ll6
0e426118
0042CIle
004251iC
e0425121
00429123
I〕04으F124
U042曰125
0042(1욘q
0042드1Ze
LEgEp<,[L驪浿L.여）】
R恪HESX
R,SH탸楙
H炤HI'
驪펭L며JDRDPTRDS:t＜驢KERIㅓEL32.De「ineo。
넜’EDI.EDI
LEF겸Em볍，tL驥瀋L.32]
PUSH-EDI
PtJSH40딩00e00
댔』SH[
PUSHEOI
댔j하읠EDI
PUSㅐCg아〕00아〕
PUSHFnX
d겹LL며蛔RO"PTRDS'【＜읾（ERNEL32.Cr'at.Ft
CllPEpX,一1
臘찌O냈mDPTRDS:[ESI+4】.ERX
J"ZSH〕RTanti맬n'f.8042514o
戚꽈E댜＜.슬ntL룔nL「.004여긔278
J冊ㅏSH가괍Tant【'ntf..04Z51Fll
.「O「OVn．급npn尸T댜nq.『「qT＋蝦효1
Press<enter>ti∥al1the
wsprintfwCode15NOP'ed
GS000e
stErr
0024'
kerne【32.DefLnrb일O황”iC삳비
USER32.닐｀pr1ntfU
hTenp'.t·Fi1.'7죠）IR9B6
ptt,1hut．료:。UFRLpPP「n
塊露ur．엎면T街脯輓軫r。。，，。
Shar순h。de=FILE닙SH아국E－며RIT티77DIF졈984
Rcce'S=GEHERIC-RERD:G탸ERIC－빠緘TE
F'【이ㅓa柏e="?????????？。rㅇ：룔）????????
Cr.at'F(l“룔
FST
FC비
傘pt닛*UN
盼pt보6.0
ehptyo.0
蓴忍驢驕
e醱ty0.0
아、pt보쵸＊e
순hpty1.0
4.n. COn
027F Pre
OD4Z[
11.4'
·겨：4[=:궁
RSCll＂나、삶니．t。어）翩SV"1inkedd·vLO
1뺙「·－닉
〕．-1-4[C13E
口尸4》「14슭
첵걋
r■■■■■．■■·．L
■．■■■■■■뻐체■■■■■I
D킬 e e
貂柏却o 。花 。
지뺙 4 8 4 C 4 凸O
2 2 2댜 0 ap Z 50
7 뱀t dU e .' '?4
계찡r쵸O · O e CF4 식乙 슥
결눌19 8 0 큔0 SFO nH 鷺
l a긴츱‘U3凸댜 ’·心Oㅓㅗ
。拓FF拓。。＂、。嵋臼拓B7‘驪
盼S6胡FF貂8D戚68胡57戚애騶麗田89花BSEB勵
··'…!··…／·…>.'·－J、
걀찡
=antt'nif.0048R
(UNICDLIE"\'.노허D
kt_Zr’〕

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 FOODBAAD를 쓸 수 없다.

그리고 완벽히 NOP 더러운 짓을 하자.

번역 주) PUSH antisnif.0048A298

PUSH EAX

CALL EDI

모두 NOP으로

시스템 생성 대체 텍스트: 띠
CpU'"'ainthread,moduleafltisnif
(F一一一一一一一一－ F「
EEF..E'
E승F',8·
EBX
브…
U04．듸」〔면
N4쥬叩S
Oe42드姃iE
心042드ODF
OO4a듀OEO
004250EI
醜4a騁「7
ee42g0Eq
OO42gOEC
죠O松輛驪－
004250F4
너04요：너1〔
凸04요「凸F→
l--IU4．컨U〔,,
T禑．．딤r괘目탭
ESI
EDI
딨）1.Llllll:IF：〔
t요Ra·IJ
EGI,E이:
pTF:D뜨
.戮녠SER3a.닐琅tLrt
USER32..'pr'ntfU
USER32.나＄Dr1nt「U
'.h魯》=
r··．·．．·．밈
---Jn!----------
아카즙，D괴〕Rn:나《U::t뜨J+8〕
翩t'$nLf.ee4爵楗B8
EB：빅
EDI
하盼l,D.I'l:IF:CFTRD〕：〔ES!+1：크〕
E여－
「뺘二，tLOCnL.82]
FOrnat=..Zh궁’
$=OOgEROCS
니＄DrintfU
密긍 · · · · · · · · · … …
r＇·．．■■．■■·■■■■·
ㄴ-F-Uㅗ234r긍69C0E
ㅑr－급UOO^pedoapoeo
04
率빽4
〔결
「r
SBEC
61［뜨C:?：더001:.1:100
53
66
57
88rD14lB4700
FF75es
88F소
蜜》5ㄴ0〕
68BaR.='4鋌10
52
FcO7
9p46:8
Sa
여눙41크ei:l
9O
90
90
9e
9e
9O
。〕더
qn
83C4
「USH
"O댜
：겠」므
PUSH
PUSH
PUSH
"O':
PU와I
＂아i
LE습
PUSH
PU드H
댜뵙L
LEI긔
PU드H
1．「F
HOP
H0P
N（】P
蠟）P
H0P
HOP
r枉峰－
HOF
nOD
ㄸR
PUSH
PUSH
PUSㅐ
驪룁L
XOR
LEn
PUSH
PU하광
PUSH
PUSH
더」SH
PUSH
PUSH
댜룁L
「Hp
ESP.le
E와렵．tL竊驪L.se]
ESX
EnX
1
며JOF궉DPTRDS'[＜盼〔ERNEL32.Oe「出．00
ED1.EOI
Ep《．[UrnL.32】
EDI
4001:1〔1000
2
k솽rn상【92.o.f蠟n戚b별O촹v쵸C샅닒
USER32.니＄Or1ntfU
ㅏ．TenplateFLle=77DIR9B6
RttribUt．드=OUERLnPPED
11Ode=
EDI
EOI
〔0e00e00
ERX
DU()RDPTRDS:[(毓KERNEL32.CreateFt
「OV -1
PS
Sh
．幣邸듀쮸輓沁r'n。
ar순ho巾죤
Rccㅎ’'=
7FI匡－SH약귑LURI
GENER!C一RERD:GE『IE
fU
TE:77D1R984
RIC-lllRITE
F나eN숲he=＂수슭？수증츱수수수수츙수수수수츌수2수수’伊수수수
Cr.at．「LI戚j
첵걋
r·■··■■■■■■L
--―닐꿱―-------'.1
뺑r 仕O nO 쨍r
4 4 C 4
l2 脯 eelZ
s e o e O C
밸‘ 02 0
45 e觸汗貂。。 a觸
8D盼騶胡FF盼8D戚68胡戚5768騶FF
nO68BC뱀i345nHㅌ
1111122222)
·놉rprDrpr구rprprDrDUVrprprorpU낟q-rDrDrprD‘·-
、‘22乙‘22222勺드22乙C25乙룔J勺요222·J
1따4444444444444444444’깍
勵硼觸觸ee觸的에ee的觸的觸觸에에騁凶ee에ee밴
꺼出」孕〔［꽈t
< …

Press <enter> till all the wsprintfw code is NOP'ed

And see what the program thinks of these changes

Wsprintfw는 NOP이 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.

1. **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 19에서 중요한 anti-debugging과 anti-anti techniques을 간단히 배웠다.

나는 네가 모든 것을 좋게 이해했기를 희망한다. 그리고 누구든지 이곳에서 무엇이든지 배웠으면 좋겠다.

Part 20에서 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 tutorial에서 특별히 시간을 아낄 수 있다고 여겨진다.

2012-02-07 오전 12:00