

05.Comparing on changes in cond jumps, animate over/in, breakpoints

2012년 1월 28일 토요일

오후 1:16

Hello everybody.

모두들 안녕

Welcome to this Part 5 in my series about reversing for newbies/beginners.

나의 초보자를 위한 reversing series Part 5에 와서 반가워.

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

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

Lena151 (2006)

1. Abstract

In this Part 5, we will reverse a "real" application to learn something about finding the places to patch.

이번 Part 5에서, 우리는 "real" application 을 무언가를 배우고 patch 를 찾기 위해 reverse 한다.

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

이것은 real application 에서 매우 좋은 방법을 찾았다.

For better comprehension, I advise you to first see the first parts in this series before looking at this if you are a newbie.

이해력을 위해 좋다. 네가 newbie라면 이 movie 를 보기 전에 이 series 의 첫번째 part 부터 봐.

The goal of this tutorial is to teach you something about a program's behaviour In my search not to harm anybody, I came across Coffeecup Visualsite Designer.

이 tutorial 의 목표는 누구에게든지 해를 주지 않는 범위 내에서 너에게 program 의 행동을 가르치는 것이다. 나는 Coffeecup Visualsite Designer 를 봐왔다.

This is a program not updated since 11/2003 and thus probably no longer updated.

이 program 은 2003/11 이후로 update 되지 않았고 아마 더 이상 update 되지 않는다.

However, because it could be misused, I only included the main executable and some more necessary dll's to run (not the install exe !) for your research in the shown techniques.

그러나, 남용하지 말자. 보여준 technique 에서 너의 연구를 위해 오직 main executable 과 실행하기 위해 필요한 약간의 dll 을 포함했다.(install exe 는 없다)

This makes the program useless for other purposes than study material. Taking a look in the specialized media, I also found this application to be "cracked" numerous times before.

이것은 다른 목적으로 쓸모 없는 program 을 만듭니다. 특별한 media 를 봐, 나는 application 이 많이 "crack"된 것을 찾았다.

Although the registration is not important because the app has no limitations in unregistered state, the patches will also "register" the app by securing the app to run always.

등록이 안된 상태임에도 불구하고 Registration은 중요하지 않다. 왜냐하면 app은 제한이 없다.
Patches는 항상 실행하기 위해 보호된 app에 의해 app을 등록한다.

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

여기, 이 application은 선발됐다. 왜냐하면 이것은 reversing tutorial에 이상적이고 오직 교육적인 목적인 target이다.

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

네가 새로 얻은 지식을 긍정적인 방향으로 이용하기를 바란다.

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

이 문제는, part 1을 참고하기를 바란다.

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.

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 두자마.

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

Movie는 text와 다음 screen을 skip 할 수 있다.

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

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

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

도전해봐. 그리고 이 text와 다음 screen을 보기 위해 hotspot을 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.

2. Tools and target

이것도 똑같음

The tools for today are : Ollydebug and... your brain.

오늘 도구들은 Ollydebug와 너의 두뇌야.

The first can be obtained for free at

첫번째 무료로 얻을 수 있다.

<http://www.ollydbg.de>

Again, the second is your responsibility

다시 한번, 두 번째는 너의 책임감이다.

Todays target is a program called Coffecup Visualsite Designer 3.0. It is a program to design websites in the WYSIWYG system.

오늘 target program 은 Coffecup Visualsite Designer 3.0 으로 불린다. 그것은 WYSIWYG system 에서 website 를 design 하기 위한 program 이다.

I included the main exe and the needed dll's to run the program for your research

나는 main exe 를 첨부했다. 그리고 너의 연구를 위해 program 을 실행하기 위해 dll 이 필요하다.

3. Behaviour of the program

Let's study that together.

Run the application in Olly.

함께 공부를 시작하자.

Olly 에서 application 을 실행 해.



First shows a nag screen saying that this is a trial version.

먼저, nag screen 은 trial version 라는 것을 보여준다.

Which asks us to buy now

현재 사겠냐고 묻는다.

And telling us that we can try this program 10 times (this is the first startup)

그리고 우리가 program 을 10 회 실행할 수 있다고 말한다.(제일 처음에 실행 됐을 때)

I want to emphasize once again that all this also proves that at this point, the program must already have made a check

나는 이것을 다시 한 번 강조하기 위해 했다. 또한 point 를 증명한다. Program 은 이미 check 를 했다.
"Am I registered or not ?"

등록됐냐? 안됐냐?

If registered, this nag can't be shown !!!

만약에 등록 되었다면, nag 은 보이지 않는다.

INFO :

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

Mouse pointer 를 여기에 유지해. 네가 textscreen 을 읽었을 때 click 해.

For now, let's continue studying the program

이제, program 을 배우자.

Then comes some welcome message with wizard facilities. It is not important for us, so let's skip this screen for next times.

환영 message 가 마법적인 설비와 함께 온다. 그것은 우리에게 중요하지 않다, 그래서 다음에는 이 screen 을 skip 한다.

So now, we can see the program.

Let's also dig a little deeper.

이제, 우리는 program 을 볼 수 있다.

조금 깊게 들어가자.

Although the website states the application was updated 2003, here it says 2000

Website state 에서 2003 년도에 update 라고 하는데 이곳은 2000 이라고 말한다.

Ok, I suppose we know enough?

Ok, 우리는 충분히 알았다고 생각하지?

Oops. Aha, there is also an ugly advertising screen as closing nag. It will be a nice feature to test our reversing capabilities.

웁스. 아하, 그곳은 또한 closing nag 할 때 못 된 광고 screen 이 있다. 이것은 우리의 reversing 능력을 test 하기 위한 좋은 현상이다.

Fine, let's study all this better in the code.

좋아, code 를 study 하자.

So, restart the application until you ...

... land here in Olly, at the EP of the program

그래서, program 의 EP 인 여기에 도착할 때까지 application 을 restart 해라.

4. Finding the patches

In this Part5, I will show you how to manually find the patches to make in an application.

이번 Part5 는, 너에게 application 에서 어떻게 수동으로 patch 를 찾아야 하는지 보여주겠다.

Let's think that over first. If a program shows a behaviour that changes when registered or not, then ... there also must be changes in the code followed, right?

먼저 생각해봐. 만약에 등록했거나 등록하지 않았을 때 program 은 바뀐 행동을 보여준다. 또한 code 에서 바뀐 점이 있다.

Other behaviour ==> other code followed !!

우리의 행동 ==> 우리의 code 를 따라 다닌다.

An example will make this more clear : the nags that are shown in this application, are only shown when NOT registered.

이 example 은 명확하게 만든다 : 오직 등록되지 않았을 때 nags 는 application 에서 나타난다.

So, there must be some conditional jumps somewhere as follows : (or similar, this is example)

Unregistered --> don't jump

Registered --> jump past nag

그래서, 약간의 조건 jump 는 있다. 따라 다닐 때 (비슷하거나, 이 예제)

미등록 --> No jump

등록 --> nag 을 jump

Now, if we can find these conditional jumps ...

이제, 우리는 다른 조건 jump 를 찾을 수 있다.

We also immediately find an indication for the registration scheme : just before the conditional jump, there will be a compare to see if the program is already registered.

우리는 또한 등록된 scheme 을 위해 즉시 지시자를 찾을 수 있다. 그리고나서 조건 jump 한다.

Program 이 이미 등록되어 있다면 그것은 보기 위해 비교됐다.

We need to find that !!

우리는 그것을 찾기 위해 필요하다.

Remark : when I say "conditional jump", I also include the variable jumps. A variable jump is for example JMP eax+1C

주목 : "conditional jump"를 말할 때, 나는 다양한 jump 를 포함했다. 변수 jump 는 이 example 에서 JMP EAX+1C 다.

In this case, depending on a changing value for EAX if registered or not, it could jump the nag or not.

등록하거나 등록하지 않은 경우, EAX 의 변화되는 값에 의존적이다. 그것은 nag 을 jump 할 수 있거나 없다.

Example : when not registered, EAX holds the value 00401354

예 : 등록되지 않았을 때, EAX 는 00401354 를 잡고 있다.

when registered, EAX holds the value 004013A0 which could jump past the nag.

Don't forget them in the future !!!

BTW, there are none here in this program.

등록되었을 때, EAX 는 004013A0 를 잡고 있다. 그것은 past nag 을 jump 할 수 있다.

그것을 미래에도 잊지마 !!!

By the way, 그것은 이 program 에 없다.

What have we done so far? We studied the behaviour from the program, but I have also already restarted the program in Olly and done some work

우리는 지금까지 무엇을 했나? 우리는 program 으로부터 behaviour 를 배웠다. 그러나 나는 이미 program 을 restart 했다. 그리고 약간의 일을 끝냈다.

: I also stepped the app completely, starting at the EP and stepping F8 and F7 "Through" the first nag, "through" the main program and through the closing nag, Exactly like I will do once more now.

나는 app 을 완벽히 step 했다. EP 에서 시작하는 것과 F8 을 눌러 stepping 하고 F7 을 눌러 첫번째 nag 를 통과하여, main program 을 통과하고 closing nag 를 통과한다. 정확히 나는 한 번 더 하는 것이다.

But I have removed it from this movie to reduce its size and to skip repetitive tasks.

나는 이 movie 에서 size 를 줄이고 반복적인 임무를 삭제했다.

Important : I have studied all conditional jumps (+ variable) and noted down if they jump or not !!!!

중요 : 나는 모든 조건 jump 를 공부했다. 그리고 그것이 jump 하는지 안 하는지 적었다.

That way, I can detect where the code chooses another path because meanwhile ... I have also expired the program by using all 10 available trials !!!

그 방법은, 다른 경로를 선택하는 code 를 발견할 수 있다. program 이 10 번 사용할 수 있는 제한에 의해 종료됐다.

Conclusion : if I find the jumps where the code path changes, then I will easily have found a way to register the program !!!

결론 : 내가 경로를 바꿀 수 있는 jump 를 찾는다면, 나는 매우 쉽게 등록한 program 을 찾는다.

Well, let's do that, but first, this is also an ideal opportunity to explain another nice feature of Olly : animate over/in (Ctrl + F8 / F7)

좋아, 시작하자, 먼저 이것은 Olly 에서 다른 좋은 특징들을 설명하기 위한 이상적인 기회다. Animate over/in (Ctrl + F8/F7)

To reduce the size of this movie, I won't show it here, but you : go ahead, and press CTRL+F8 (animate over) and look at what happens.

Movie 의 size 를 줄이기 위해, 나는 여기에서 보여주지 않는다, 그러나 너는 해봐. Ctrl + F8(animate over)을 누르고 어떤 일이 일어나는지 봐

Some of my stuff from today is based upon it and I will come back later on this. It is also important that you can see once how code is executed.

오늘날 약간의 내 재료들은 기본이다. 그리고 나는 다음에 돌아온다. 이것은 또한 Code 가 어떻게 실행되는지 보는 게 중요하다.

Remark : I will already tell you that a more advanced feature for this is TC (trace call == run trace), where you can log and set conditions. But we will discuss that feature in another part of this series.

주목 : 나는 이미 너에게 TC 를 위해 많은 진화된 사실들을 말했다.(trace call == 추적 실행) 그곳은 log 를 남기고 condition 을 set 할 수 있다. 그러나 우리는 series 의 다른 part 에서 feature 를 의논한다.

Ok. Have you seen it?

Now, to stop animating, you can always press F12 or click the pause button.

지금까지 본 적 있어?

이제, animating 을 멈출께. 너는 항상 F12 를 누르거나 pause button 을 click 하면 돼.

Ok, restart and come back here to EP. I'm still at the EP of the soft. See me step F8, but I will stop my stepping when necessary to show you some important stuff.

Ok, 재시작하고 EP 로 돌아오자. 나는 계속 EP 에 있을거야. F8 눌러, 중요한 재료를 보여주고 필요하다고 느껴질 때 나의 stepping 을 stop 할거야.

Let's finally start exploring F8

마지막으로 F8 로 탐험을 시작하자.

```

00480368: 55          PUSH EBP
00480369: 8BEC        MOV EBX, ESP
0048036A: 8D 4C 00     PUSH VisualSt.004BD028
0048036B: C7 44 00 00  K JMP MSVCRT._except_handler3
0048036C: 44 AD 00 00  HLLP_Clipper_EPE_PFN
0048036D: 00           ; 004BD028
0048036E: 00           ; 004BD028
0048036F: 00           ; 004BD028
00480370: 00           ; 004BD028
00480371: 00           ; 004BD028
00480372: 00           ; 004BD028
00480373: 00           ; 004BD028
00480374: 00           ; 004BD028
00480375: 00           ; 004BD028
00480376: 00           ; 004BD028
00480377: 00           ; 004BD028
00480378: 00           ; 004BD028
00480379: 00           ; 004BD028
0048037A: 00           ; 004BD028
0048037B: 00           ; 004BD028
0048037C: 00           ; 004BD028
0048037D: 00           ; 004BD028
0048037E: 00           ; 004BD028
0048037F: 00           ; 004BD028
00480380: 00           ; 004BD028
00480381: 00           ; 004BD028
00480382: 00           ; 004BD028
00480383: 00           ; 004BD028
00480384: 00           ; 004BD028
00480385: 00           ; 004BD028
00480386: 00           ; 004BD028
00480387: 00           ; 004BD028
00480388: 00           ; 004BD028
00480389: 00           ; 004BD028
0048038A: 00           ; 004BD028
0048038B: 00           ; 004BD028
0048038C: 00           ; 004BD028
0048038D: 00           ; 004BD028
0048038E: 00           ; 004BD028
0048038F: 00           ; 004BD028
00480390: 00           ; 004BD028
00480391: 00           ; 004BD028
00480392: 00           ; 004BD028
00480393: 00           ; 004BD028
00480394: 00           ; 004BD028
00480395: 00           ; 004BD028
00480396: 00           ; 004BD028
00480397: 00           ; 004BD028
00480398: 00           ; 004BD028
00480399: 00           ; 004BD028
0048039A: 00           ; 004BD028
0048039B: 00           ; 004BD028
0048039C: 00           ; 004BD028
0048039D: 00           ; 004BD028
0048039E: 00           ; 004BD028
0048039F: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
004803A9: 00           ; 004BD028
004803A0: 00           ; 004BD028
004803A1: 00           ; 004BD028
004803A2: 00           ; 004BD028
004803A3: 00           ; 004BD028
004803A4: 00           ; 004BD028
004803A5: 00           ; 004BD028
004803A6: 00           ; 004BD028
004803A7: 00           ; 004BD028
004803A8: 00           ; 004BD028
0
```

나는 example 을 보여줄 것이다.

...see the JNZ here

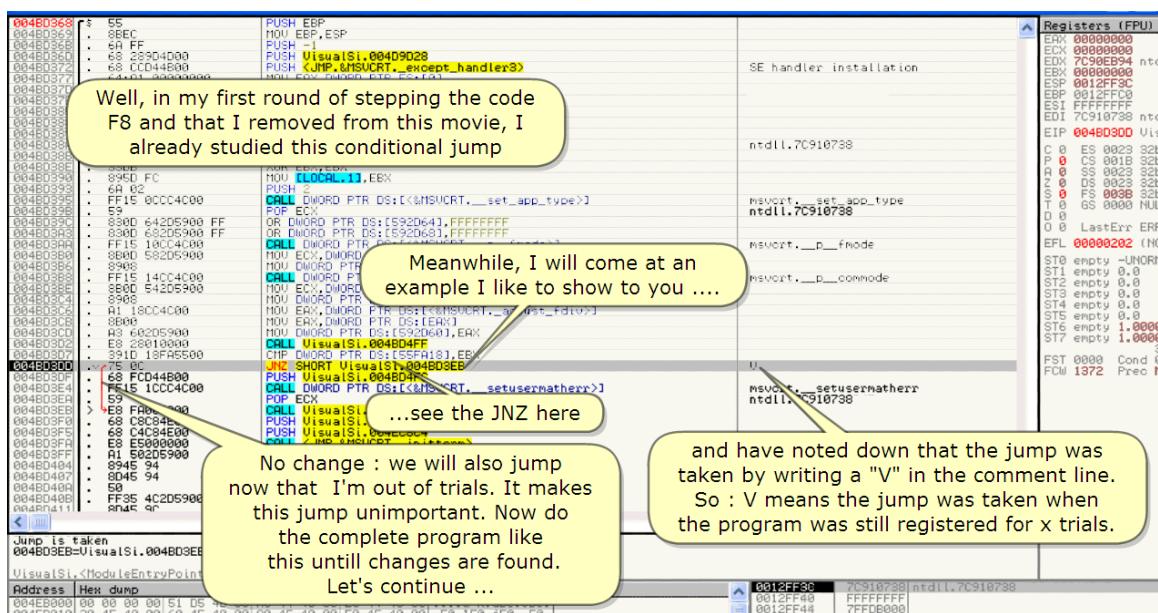
JNZ 가 이곳인 걸 봐.

Well, in my first round of stepping the code F8 and this movie, I already studied this conditional jump and have noted down that the jump was taken by writing a "V" in the comment line.

좋아, 첫번째로 code F8 을 눌러 stepping 할 수 있다. 나는 이미 조건 jump 를 공부했다. Comment line 에 Jump 되는 곳은 V 로 적었다.

So : V means the jump was taken when the program was still registered for x trials.

그래서 : V 는 Program 0이 x trial 을 위해 등록 됐을 때 jump 를 했다.



No change : we will also jump now that I'm out of trials. It makes this jump unimportant. Now do the complete program like this until changes are found.

안 바뀐 점 : 우리는 이제 jump 할 수 있다. Trial에서 나간다. 그것은 jump 가 중요하지 않다. 이제 바뀐 점을 찾을 때까지 완벽한 program 이다.

Let's continue...

Now, scroll up

계속하자.

이제, 스크롤 올려

C CPU - main thread, module VisualSI

```

004BD404  . 8945 94    MOU [LOCAL.27],EAX
004BD407  . 8045 94    LEA EAX,[LOCAL.27]
004BD40A  . 50          PUSH EAX
004BD40B  . FF35 4C2D5900 PUSH DWORD PTR DS:[592D40]
004BD411  . 8045 9C    LEA EAX,[LOCAL.25]
004BD414  . 50          PUSH EAX
004BD415  . 8045 90    LEA EAX,[LOCAL.28]
004BD418  . 50          PUSH EAX
004BD419  . 8045 A0    LEA EAX,[LOCAL.24]
004BD41C  . 50          PUSH EAX
004BD41D  . FF15 24CC4000 CALL DWORD PTR DS:[&MSVCRT._getmainCRT,_acmdln]
004BD423  . doesn't jump either now,
004BD428  . so, no change and we continue
004BD42D  . 803E 22    MOU [LOCAL.29],ESI
004BD432  . 8975 8C    CMP BYTE PTR DS:[ESI],22
004BD435  . 004BD47E  JNZ SHORT VisualSI.004BD47E
004BD43C  . 803E 22    INC ESI
004BD43F  . 8975 8C    MOU [LOCAL.29],ESI
004BD443  . 8006        MOU AL,BYTE PTR DS:[ESI]
004BD44A  . 3AC3        CMP AL,BL
004BD44C  . 74 04        JE SHORT VisualSI.004BD452
004BD44E  . 3C 22        CMP AL,22
004BD450  . 75 F2        JNZ SHORT VisualSI.004BD444
004BD452  . 883E 22    CMP BYTE PTR DS:[ESI],22
004BD455  . 75 44        JNZ SHORT VisualSI.004BD458
004BD457  . 46          INC ESI
004BD459  . 8975 8C    MOU [LOCAL.29],ESI
004BD45B  . 8006        MOU AL,BYTE PTR DS:[ESI]
004BD45D  . 3AC3        CMP AL,BL
004BD45F  . 74 04        JE SHORT VisualSI.004BD465
004BD461  . 3C 28        CMP AL,28
004BD463  . 75 F2        JBE SHORT VisualSI.004BD457
004BD465  . 895D D0    MOU [LOCAL.12],EBX
004BD467  . 8045 A4    LEA EAX,[LOCAL.29]
004BD46B  . 50          PUSH EAX
004BD46C  . FF15 C8C14C00 CALL DWORD PTR DS:[&KERNEL32.GetStartupInfoA]
004BD472  . F645 D0 B1    TEST BYTE PTR SS:[EBP-30],1
004BD476  . 74 11        JE SHORT VisualSI.004BD489
004BD478  . 0FB745 D4    MOUVX EAX,WORD PTR SS:[EBP-2C]
004BD47C  . EB 0E        JMP SHORT VisualSI.004BD48C
004BD47F  . 48A9F 20    CMP RTTF PTR DS:[ESTI1] PA

```

didn't jump when
"registered" for x trials

Didn't jump when "registered" for x trials

Doesn't jump either now,

so, no change and we continue

등록 되었을 때 jump 하지 않는다.

어느 것 하나 jump 하지 않는다.

그래서 바꾸지 않는다. 그리고 우리는 계속한다.

C CPU - main thread, module VisualSi

```

004BD0404  . 8945 94 MOU [LOCAL_27],EAX
004BD0407  . 8045 94 LEA EAX,[LOCAL_27]
004BD0409  . 50 PUSH EAX
004BD040B  . FF35 4C2D5900 PUSH DWORD PTR DS:[592D4C]
004BD0411  . 8D45 9C LEA EAX,[LOCAL_25]
004BD0414  . 50 PUSH EAX
004BD0415  . 8D45 90 LEA EAX,[LOCAL_28]
004BD0418  . 50 PUSH EAX
004BD0419  . 8D45 A0 LEA EAX,[LOCAL_24]
004BD041C  . 50 PUSH EAX
004BD041D  . 4C31 C0 CALL WORD PTR DS:[<&MSVCRT._acmdln>]
004BD0423  . 8045 94 MOU [LOCAL_27],EAX
004BD0428  . 8045 94 LEA EAX,[LOCAL_27]
004BD042D  . 50 PUSH EAX
004BD0432  . 4C31 C0 CALL WORD PTR DS:[<&MSVCRT._acmdln>]
004BD0435  . 8045 94 MOU [LOCAL_27],EAX
004BD0439  . 8B30 MOU ESI,DWORD PTR DS:[EAX]
004BD043D  . 8975 8C MOU [LOCAL_29],ESI
004BD043F  . 803E 22 CMP BYTE PTR DS:[ESI],22
004BD0442  . 75 3A JNZ SHORT VisualSi.004BD047E
004BD0444  . 46 INC ESI
004BD0445  . 8975 8C MOU [LOCAL_29],ESI
004BD0443  . 8A06 MOU AL,BYTE PTR DS:[ESI]
004BD0440  . 3AC3 CMP AL,BL
004BD0442  . 74 04 JE SHORT VisualSi.004BD0452
004BD0445  . 3C 22 CMP AL,22
004BD0452  . 75 F2 JNZ SHORT VisualSi.004BD0444
004BD0453  . 883E 22 CMP BYTE PTR DS:[ESI],22
004BD0455  . 75 04 JNE SHORT VisualSi.004BD0458
004BD0458  . 46 INC ESI
004BD0459  . 8975 8C MOU [LOCAL_29],ESI
004BD045B  . 8A06 MOU AL,BYTE PTR DS:[ESI]
004BD045D  . 3AC3 CMP AL,BL
004BD045F  . 74 04 JE SHORT VisualSi.004BD0465
004BD0461  . 3C 20 CMP AL,20
004BD0463  . 76 F2 JBE SHORT VisualSi.004BD0457
004BD0465  . 895D 00 MOU [LOCAL_12],EBX
004BD0468  . 8D45 A4 LEA EAX,[LOCAL_23]
004BD046B  . 50 PUSH EAX
004BD046D  . FF15 C8C14C00 CALL DWORD PTR DS:[<&KERNEL32.GetStartupInfoA>]
004BD0472  . F645 00 01 TEST BYTE PTR SS:[EBP-30],1
004BD0476  . 74 11 JE SHORT VisualSi.004BD0489
004BD0478  . 0FB745 D4 MOUZX EAX,WORD PTR SS:[EBP-20]
004BD047D  . EB 0E JMP SHORT VisualSi.004BD049C
004BD047F  . 893F 9A CMP RPTE PTR DS:[ESI1],20

```

...and get in a loop with cond jumps.

ESI=00141EE0, (ASCII "C:\Program Files\VisualSite Designer\VisualSite Designer.exe")
Jump from <ModuleEntryPoint>+0E8

...and get in a loop with cond jumps.

Loop 를 조건 jump 와 함께 들어간다.

C CPU - main thread, module VisualSi

```

004BD0404  . 8945 94 MOU [LOCAL_27],EAX
004BD0407  . 8045 94 LEA EAX,[LOCAL_27]
004BD0409  . 50 PUSH EAX
004BD040B  . FF35 4C2D5900 PUSH DWORD PTR DS:[592D4C]
004BD0411  . 8D45 9C LEA EAX,[LOCAL_25]
004BD0414  . 50 PUSH EAX
004BD0415  . 8D45 90 LEA EAX,[LOCAL_28]
004BD0418  . 50 PUSH EAX
004BD0419  . 8D45 A0 LEA EAX,[LOCAL_24]
004BD041C  . 50 PUSH EAX
004BD041D  . 4C31 C0 CALL WORD PTR DS:[<&MSVCRT._acmdln>]
004BD0423  . 8045 94 MOU [LOCAL_27],EAX
004BD0428  . 8045 94 LEA EAX,[LOCAL_27]
004BD042D  . 50 PUSH EAX
004BD0432  . 4C31 C0 CALL WORD PTR DS:[<&MSVCRT._acmdln>]
004BD0435  . 8045 94 MOU [LOCAL_27],EAX
004BD0439  . 8B30 MOU ESI,DWORD PTR DS:[EAX]
004BD043D  . 8975 8C MOU [LOCAL_29],ESI
004BD043F  . 803E 22 CMP BYTE PTR DS:[ESI],22
004BD0442  . 75 3A JNZ SHORT VisualSi.004BD047E
004BD0444  . 46 INC ESI
004BD0445  . 8975 8C MOU [LOCAL_29],ESI
004BD0443  . 8A06 MOU AL,BYTE PTR DS:[ESI]
004BD0440  . 3AC3 CMP AL,BL
004BD0442  . 74 04 JE SHORT VisualSi.004BD0452
004BD0445  . 3C 22 CMP AL,22
004BD0452  . 75 F2 JNZ SHORT VisualSi.004BD0444
004BD0453  . 883E 22 CMP BYTE PTR DS:[ESI],22
004BD0455  . 75 04 JNE SHORT VisualSi.004BD0458
004BD0458  . 46 INC ESI
004BD0459  . 8975 8C MOU [LOCAL_29],ESI
004BD045B  . 8A06 MOU AL,BYTE PTR DS:[ESI]
004BD045D  . 3AC3 CMP AL,BL
004BD045F  . 74 04 JE SHORT VisualSi.004BD0465
004BD0461  . 3C 20 CMP AL,20
004BD0463  . 76 F2 JBE SHORT VisualSi.004BD0457
004BD0465  . 895D 00 MOU [LOCAL_12],EBX
004BD0468  . 8D45 A4 LEA EAX,[LOCAL_23]
004BD046B  . 50 PUSH EAX
004BD046D  . FF15 C8C14C00 CALL DWORD PTR DS:[<&KERNEL32.GetStartupInfoA>]
004BD0472  . F645 00 01 TEST BYTE PTR SS:[EBP-30],1
004BD0476  . 74 11 JE SHORT VisualSi.004BD0489
004BD0478  . 0FB745 D4 MOUZX EAX,WORD PTR SS:[EBP-20]
004BD047D  . EB 0E JMP SHORT VisualSi.004BD049C
004BD047F  . 893F 9A CMP RPTE PTR DS:[ESI1],20

```

...and get in a loop with cond jumps.

Well, a loop is first studied, we see that the loop is only going to read an unimportant "location" string

ESI=00141EE1, (ASCII "C:\Program Files\VisualSite Designer\VisualSite Designer.exe")
Stack SS:[0012FF4C]=00141EE0, (ASCII "C:\Program Files\VisualSite Designer\VisualSite Designer.exe")

Well, a loop is first studied, we see that the loop is only going to read an unimportant "location" string

Loop 제일 먼저 배운다.

loop 가 오직 중요하지 않은 "location" string 을 읽기 위해 돈다.

So, it's unimportant. That's also why I haven't made any V or X comments here. Else, we must count the number of loops taken.

그래서, 이것은 중요하지 않다. 그것은 또한 여기에 어떠한 V 나 X 를 만들지 못했다. 우리는 loop 얼마나 도는지 횟수를 센다.

...and get in a loop with cond jumps.

그리고 우리는 cond jump 와 함께 loop 를 돈다.

The screenshot shows the CPU pane of Immunity Debugger with assembly code for the VisualSI module. The code is annotated with several callouts:

- A yellow callout points to a conditional jump instruction: "...and get in a loop with cond jumps."
- A yellow callout points to another conditional jump instruction: "Also notice that this is the place where we will jump to leave the loop"
- A yellow callout points to a loop exit point: "Well, a loop is first that the loop is on an unimportant 'l'"
- A yellow callout points to the end of the assembly code: "Jump is NOT taken 004BD452=VisualSI.004BD452"
- A yellow callout points to the start of the assembly code: "So, it ma m

```
004BD404 8945 94 MOV [LOCAL.27],EBX
004BD407 8945 94 LEA EBX,[LOCAL.27]
004BD40A 50 PUSH EBX
004BD40B FF35 4C2D5900 PUSH DWORD PTR DS:[592D4C]
004BD411 8945 9C LEA EBX,[LOCAL.25]
004BD414 50 PUSH EBX
004BD415 8945 90 LEA EBX,[LOCAL.28]
004BD418 50 PUSH EBX
004BD419 8945 A0 LEA EBX,[LOCAL.24]
004BD41C 50 PUSH EBX
004BD41D 8945 94 MOV [LOCAL.27],EBX
004BD423 8945 94 LEA EBX,[LOCAL.27]
004BD428 8945 94 LEA EBX,[LOCAL.27]
004BD432 8945 94 LEA EBX,[LOCAL.27]
004BD435 8945 94 LEA EBX,[LOCAL.27]
004BD438 8B30 MOV ESI,DWORD PTR DS:[&MSUCRT._getmainargs]
004BD43C 8975 8C MOV [LOCAL.29],ESI
004BD43F 893E 22 MOV AL,BYTE PTR DS:[ESI]
004BD442 75 3A CMP BYTE PTR DS:[ESI],22
004BD444 > 46 JE SHORT VisualSI.004BD452
004BD445 8975 8C MOV [LOCAL.29],ESI
004BD448 8A06 MOV AL,BYTE PTR DS:[ESI]
004BD453 8AC3 CMP AL,BL
004BD44C 74 04 JE SHORT VisualSI.004BD452
004BD44E 3C 22 CMP AL,22
004BD458 75 F2 JE SHORT VisualSI.004BD444
004BD452 802E 22 CMP BYTE PTR DS:[ESI],22
004BD455 75 04 JE SHORT VisualSI.004BD45B
004BD457 > 46 INC ESI
004BD458 > 46 INC ESI
004BD45B > 46 INC ESI
004BD45F > 46 INC ESI
004BD461 8945 94 LEA EBX,[LOCAL.27]
004BD463 8945 94 LEA EBX,[LOCAL.27]
004BD465 8945 94 LEA EBX,[LOCAL.27]
004BD468 8945 94 LEA EBX,[LOCAL.27]
004BD470 50 PUSH EBX
004BD472 FF15 C8C14C00 CALL DWORD PTR DS:[&KERNEL32._GetModuleHandleA]
004BD476 F645 D0 01 TEST BYTE PTR SS:[EBP-30],1
004BD477 > 74 11 JE SHORT VisualSI.004BD489
004BD478 0FB745 D4 MOUZX EBX,WORD PTR SS:[EBP-2C]
004BD47C EB 0E JMP SHORT VisualSI.004BD49C
004BD47F 803F 20 CMP BYTE PTR DS:[EST1],20
```

Also notice that this is the place where we will jump to leave the loop

또한 장소라는 것을 알린다. 그 장소에서 loop 를 떠나기 위해 jump 한다.

So, we can safely put a BP there

그래서 우리는 안전하게 이곳에 BP 를 넣는다.

And now, press F9 to run the loop

This way, we don't have to trace rounds like a squirrel in a wheel.

Yep, we remove the BP

그리고 이제, loop 를 실행하기 위해 F9 를 눌러.

이 방법은, 추적하기 위해 다람쥐 마냥 셋바퀴를 돌지 않는다.

예, BP 를 삭제해.

Notice that this step doesn't bring any changes either

BTW, making changes in the comment line is done by doubleclicking the comment line. Normally, I write these down on a piece of paper (old school huh). But for your convenience, I've done it in the comment line this time, so that you can follow it all easily. See how to do it ...

```
004BD428: 68 00004E0000 PUSH VisualSI.004EB000
004BD42D: E8 B2000000 CALL <MF.&MSVCRT._(InitTerm>
004BD432: 83C4 24 MOV ECX,4
004BD435: H1 28CC4C00 MOV ECX,28CC4C00
004BD439: 89D9 MOV ECX,DWORD PTR DS:[ECX]
004BD43D: 33D2 MOV ECX,ECX
004BD441: 89E8 MOV ECX,DWORD PTR DS:[ECX]
004BD445: 89E9 MOV ECX,DWORD PTR DS:[ECX]
004BD449: 89E9 3A MOV ECX,3A
004BD44A: > 46 INC ESI
004BD44B: 8975 8C MOV AL,BYTE PTR DS:[ESI]
004BD44C: 89E6 MOV ECX,DWORD PTR DS:[ESI]
004BD44D: 3AC3 CMP AL,BL
004BD44E: > 74 04 JE SHORT_VisualsSI.004BD452
004BD44F: 3C 22 CMP AL,22
004BD450: 75 F2 JNZ SHORT_VisualsSI.004BD444
004BD452: 89E9 22 MOV ECX,DWORD PTR DS:[ESI]
004BD455: > 75 04 JNZ SHORT_VisualsSI.004BD458
004BD457: 46 INC ESI
004BD458: 8975 8C MOV AL,BYTE PTR DS:[ESI]
004BD45B: > 89E6 MOV ECX,DWORD PTR DS:[ESI]
004BD45D: 3AC3 CMP AL,BL
004BD45F: > 74 04 JE SHORT_VisualsSI.004BD461
004BD461: 3C 20 CMP AL,20
004BD463: > 76 F2 JBE SHORT_VisualsSI.004BD465
004BD465: 8950 00 MOV ECX,DWORD PTR DS:[ESI]
004BD468: 8D45 A4 LEA EBX,DWORD PTR DS:[ESI]
004BD46B: 50 PUSH EBX
004BD46C: FF15 C8C14C00 CALL DWORD PTR DS:[&KE]
004BD472: F645 00 B1 TEST BYTE PTR SS:EBP-3
004BD476: > 74 11 JE SHORT_VisualsSI.004BD478
004BD478: 0FB745 D4 MOUVZ EBX,MWORD PTR SS:[EBP]
004BD47C: EB 0E JMP SHORT_VisualsSI.004BD448
004BD47F: 89E9 20 MOV ECX,DWORD PTR DS:[ESI]
Jump NOT taken
004BD45B=VisualsSI.004BD458
VisualSI <ModuleEntryPoint>+0ED
```

Notice that this step doesn't bring any changes either

알린다. 이 step 에서는 바뀐 어떤 것도 가져오지 않는다.

BTW, making changes in the comment line is done by doubleclicking the comment line.

By the way, 주석 라인의 변화를 만드는 것은 comment line 을 doubleclick 을 하면 된다.

Normally, I write these down on a piece of paper (old school huh). But for your convenience, I've done it in the comment line this time, so that you can follow it all easily. See how to do it ...

보통, 그것들을 종이 위에 적는다. (예전 방식 후..) 그러나 너의 편리함을 위해, 내가 이미 comment line 에 해놨다. 그래서 너는 쉽게 따라갈 수 있다. 어떻게 하는지 봐.

:)

```
CPU - main thread, module VisualSI
004BD0454: 8945 94        MOU EAX,[LOCAL_27]
004BD0457: 8045 94        LEA EAX,[LOCAL_27]
004BD0458: 59              PUSH EAX
004BD0459: FF35 4C2D5900  PUSH DWORD PTR DS:[592D40]
004BD045B: 8045 9C        LEA EAX,[LOCAL_25]
004BD045C: 58              PUSH EAX
004BD045D: 8045 90        LEA EAX,[LOCAL_28]
004BD045E: 59              PUSH EAX
004BD045F: 8045 A0        LEA EAX,[LOCAL_24]
004BD0460: 58              PUSH EAX
004BD0461: FF15 24CC4C00  CALL DWORD PTR DS:[<&MSVCR7.0!_getmainargs>]
004BD0462: 68 C0C84E00  PUSH VisualSI!004EC8C0
004BD0463: 68 00B04E00  PUSH VisualSI!004EB000
004BD0464: E8 B2000000
004BD0465: 89C4 24        MOU EAX,[LOCAL_22]
004BD0466: A1 28004C00  LEA EAX,[LOCAL_20]
004BD0467: 5930
004BD0468: 5975 8C        CMP AL,BL
004BD0469: 893E 22        JNE SHORT VisualSI!004BD0465
004BD0470: 75 3A        CMP AL,CL
004BD0471: > 8945 46        MOU EAX,[LOCAL_26]
004BD0472: 8975 8C        LEA EAX,[LOCAL_28]
004BD0473: 8006
004BD0474: 3AC3
004BD0475: ^ 74 04        JNE SHORT VisualSI!004BD0457
004BD0476: ^ 3C 20        CMP AL,DL
004BD0477: ^ 76 F2        JNE SHORT VisualSI!004BD0457
004BD0478: ^ 895D 00        MOU EAX,[LOCAL_12]
004BD0479: 8045 H4        LEA EAX,[LOCAL_28]
004BD047A: 59              PUSH EAX
004BD047B: FF15 C8C14C00  CALL DWORD PTR DS:[<&KERNEL32.GetStartupInfoA>]
004BD047C: F645 D0 01        TEST BYTE PTR SS:[EBP-30],1
004BD047D: 74 11        JE SHORT VisualSI!004BD0489
004BD047E: 0FBF745 D4        MOUZX EAX,MORD PTR SS:[EBP-20]
004BD047F: EB 0E        JMP SHORT VisualSI!004BD048C
004BD047F: 893F 2A        FTRM BYTE PTR DS:[FEST1+20]

Jump is NOT taken
004BD045B=VisualSI!004BD045B
```

BTW, when stepping a call F8 and the nag appears, you have to BP the call and restart the app.

By the way, 우리가 call 을 F8 로 넘기고 nag 이 나타날 때, 너는 그 call 에 BP 를 걸고 app 을 재시작 해.

Then run till BP to step F7 in the call and continue F8 to dig deeper and deeper till you finally find where exactly the window (for nag or program) is created.

BP 까지 실행하고 F7 로 call 안으로 들어가 그리고 F8 로 깊게 파고들어 그리고 마지막으로 정확히 윈도우가(nag 이나 program) 만들어진 곳을 찾아라.

Remark : if a change would take place deep in a call for example, we would also notice changes in program behaviour here. For example by an apparition of another nag.

주목 : 만약에 변화가 이 예제에서처럼 깊다면, 우리는 또한 program behaviour 가 바뀐 것을 느낀다. 이 예제에서처럼 다른 nag 의 유형같이.

We would always easily find the change in the code.

우리는 항상 code에서 쉽게 변화를 찾을 수 있다.

So again, continue stepping F8 and keep looking for changes in the jumps.

그래서 계속 F8 을 눌러. 그리고 jump에서 변화된 점을 주의 깊게 봐.

Scroll up

스크롤 올려

The screenshot shows the CPU window of the Visual Studio debugger for the main thread of the VisualSI module. The assembly code is displayed in the left pane, and registers and memory dump tabs are visible on the right.

Annotations in the assembly pane:

- A call instruction at address 004BD0497 is highlighted with a yellow box. A callout points to it with the text: "So, now I know that I have to step IN the call F7".
- An instruction at address 004BD0497 is highlighted with a yellow box. A callout points to it with the text: "to find where the nag is created."
- A call instruction at address 004BD0497 is highlighted with a yellow box. A callout points to it with the text: "I noticed that the nag showed up when stepping this call F8".
- A call instruction at address 004BD0497 is highlighted with a yellow box. A callout points to it with the text: "On this line, press F7 and remember that this is the very first call where the nag is shown".
- A RETN instruction at address 004BD0498 is highlighted with a yellow box. A callout points to it with the text: "OK".
- A call instruction at address 004BD0497 is highlighted with a yellow box. A callout points to it with the text: "The first time I stepped this code F8 ...".

OK

The first time I stepped this code F8 ...

첫번째로 code 를 넘기기 위해 F8 을 눌러.

I noticed that the nag showed up when stepping this call F8

Call 을 F8 로 넘길 때, nag 이 나타나는 것을 느낀다.

So, now I know that I have to step IN the call F7

그래서, 나는 call 에서 step 을 넘기기 위해 F7 을 눌러야 한다.

To find where the nag is created.

Nag 이 만들어지는 곳을 찾아라.

On this line, press F7 and remember that this is the very first call where the nag is shown

이 line에서, F7 을 누르고 nag 이 나타나는 첫번째 call 이라는 것을 기억해라.

C CPU - main thread, module VisualSi

```

004B0510    . 83C4 14    MOU ECX,DWORD PTR SS:[ESP+10]
004B0514    . E8 43000000  PUSH DWORD PTR SS:[ESP+10]
004B0518    . C2 1000    RETN 10
004B051C    . FF7424 10    PUSH DWORD PTR SS:[ESP+10]
004B0520    . E8 43000000  CALL JMP.&MFC42.#1576
004B0524    . C2 1000    RETN 10
004B0528    . FF7424 10    PUSH DWORD PTR SS:[ESP+10]
004B052C    . E8 43000000  CALL JMP.&MFC42.#1168
004B0530    . C2 1000    RETN 10
004B0534    . FF7424 10    PUSH DWORD PTR SS:[ESP+10]
004B0538    . E8 43000000  CALL JMP.&MFC42.#1576
004B0542    . C2 1000    RETN 10
004B0544    . FF15 A8CB4C00  CALL DWORD PTR DS:[<&MSUCRT._setmbcp>]
004B0548    . 59          POP ECX
004B054B    . 6A 01        PUSH 1
004B054D    . 58          POP EAX
004B054E    . C2 0800    RETN 8
004B0551    . E8 00000000  JMP VisualSi.004B0556
004B0556    . 68 00060000  PUSH 600
004B0558    . 6A 00        PUSH 0
004B055D    . E8 C6FFFFFF  CALL VisualSi.004B0528
004B0562    . A2 5C2D5900  MOU BYTE PTR DS:[592D5C],AL
004B0567    . C3          RETN
004B0568    . FF25 8CCB4C00  CALL DWORD PTR DS:[<&MFC42.#1576>]
004B056E    . CC          INT3
004B056F    . CC          INT3
004B0570    . 8B4D F0        MOV ECX,DWORD PTR SS:[EBP-10]
004B0573    . ^ E9 BAEFFFFF  JMP <JMP.&MFC42.#768>
004B0578    . 8B4D F0        MOV ECX,DWORD PTR SS:[EBP-10]
004B057B    . 81C1 90000000  ADD ECX,90
004B0581    . - FF25 7CC04C00  CALL DWORD PTR DS:[<&Controls.CWheelControl::CWheelControl>]
004B0587    . 8B4D F0        MOV ECX,DWORD PTR SS:[EBP-10]
004B058A    . 81C1 74020000  ADD ECX,274
004B0590    . - FF25 80C04C00  JMP DWORD PTR DS:\[<&Controls.CSliderControl::CSliderControl>\]
004B0596    . 8B4D F0        MOV ECX,DWORD PTR SS:[EBP-10]
004B0599    . 81C1 E8030000  ADD ECX,3E8
004B059F    . - FF25 80C04C00  JMP DWORD PTR DS:\[<&Controls.CSliderControl::CSliderControl>\]
004B05A5    . 8B4D F0        MOV ECX,DWORD PTR SS:[EBP-10]
004B05A8    . 81C1 4C050000  ADD ECX,54C
004B05AE    . ^ E9 79EFFFFF  JMP <JMP.&MFC42.#609>
004B05B3    . 8B4D F0        MOV ECX,DWORD PTR SS:[EBP-10]
004B05B6    . 81C1 8C050000  ADD ECX,58C
004B05C1    . ^ F9 AFFFFFFF  JMP <JMP.&MFC42.#795>

```

and so now, press

F7 to step in the call

because again, the nag showed here, ...

Same here ...

To land here in the call. Same here ...

Because again, the nag showed here, ...

And so now, press

F7 to step in the call

Call 안의 이곳에 도착했다. 똑같은 위치다.

왜냐하면 다시, nag 이 나타난다.

그리고 F7 을 눌러 call 안으로 들어가라.

C CPU - main thread, module VisualSi

```

004BD510 004BD514 004BD518 004BD51C 004BD520 004BD525 004BD528 004BD531 004BD534 004BD537 004BD53A 004BD540 004BD542 004BD544 004BD549 004BD54E 004BD551 004BD556 004BD55B 004BD55D 004BD562 004BD567 004BD568 004BD56E 004BD56F 004BD570 004BD572
    FF7424 10 E8 43000000 C2 1000
    PUSH DWORD PTR SS:[ESP+10]
    CALL <JMP.&MFC42.#1576>
    RETN 10
    JMP <JMP.&MFC42.#1168>
    PUSH DWORD PTR SS:[ESP+4]
    CALL <JMP.&MFC42.#1168>
    POP ECX
    MOV ESI,ESI.004BD556
    PUSH 600
    PUSH 0
    JNZ SHORT VisualSi.004BD54B
    PUSH -3
    CALL DWORD PTR DS:[<&MSVCRT._setmbcp>]
    POP ECX
    MOV ECX,DWORD PTR SS:[EBP-10]
    JMP <JMP.&MFC42.#1576>
    INT3
    INT3
    MOV ECX,DWORD PTR SS:[EBP-10]
    JMP <JMP.&MFC42.#1576>

```

and so now, press F7 to step in the call because again, the nag showed here, ... to land here. Notice that we will jump to a

Same here ...

To land here. Notice that we will jump to a ...

C CPU - main thread, module MFC42

```

7304CF2B 8BFF MOU EDI,ED1
7304CF2D 53 PUSH EBX
7304CF2E 56 PUSH ESI
7304CF2F 57 PUSH EDI
7304CF30 83CB FF OR EBX,FFFFFFFF
7304CF33 E8 CD40FFFF CALL MFC42.#1175
7304CF38 8BF0 MOU ESI,EAX
7304CF3A E8 97B30000 CALL MFC42.#1168
7304CF3F FF7424 1C PUSH DWORD PTR SS:[ESP+1C]
7304CF43 8B78 04 MOU EDI,DWORD PTR DS:[EAX+4]
7304CF46 FF7424 1C PUSH DWORD PTR SS:[ESP+1C]
7304CF4A FF7424 1C PUSH DWORD PTR SS:[ESP+1C]
7304CF4E FF7424 1C PUSH DWORD PTR SS:[ESP+1C]
7304CF52 E8 C1CC0000 CALL MFC42.#1575
7304CF57 85C0 TEST EAX,EAX
7304CF59 74 3C JE SHORT MFC42.7304CF97
7304CF5B 85FF TEST EDI,EDI
7304CF5D 74 0E JE SHORT MFC42.7304CF6D
7304CF5F 8B07 MOU EAX,DWORD PTR DS:[EDI]
7304CF61 88CF MOU ECX,EDI
7304CF63 FF90 8C000000 CALL DWORD PTR DS:[EAX+8C]
7304CF69 85C0 TEST EAX,EAX
7304CF6B 74 2A JE SHORT MFC42.7304CF97
7304CF6D 8B06 MOU ECX,DWORD PTR DS:[ESI]
7304CF6F 88CE MOU ECX,ESI
7304CF71 FF50 58 CALL DWORD PTR DS:[EAX+58]
7304CF74 85C0 TEST EAX,EAX
7304CF76 75 16 JNZ SHORT MFC42.7304CF8E
7304CF78 3946 20 CMP DWORD PTR DS:[ESI+20],EAX
7304CF7B 74 08 JE SHORT MFC42.7304CF85
7304CF7D 884E 20 MOU ECX,DWORD PTR DS:[ESI+20]
7304CF80 8B01 MOU EAX,DWORD PTR DS:[ECX]
7304CF82 FF50 60 CALL DWORD PTR DS:[EAX+60]
7304CF85 8B06 MOU EAX,DWORD PTR DS:[ESI]
7304CF87 88CE MOU ECX,ESI
7304CF89 FF50 70 CALL DWORD PTR DS:[EAX+70]
7304CF8C 74 07 JMP SHORT MFC42.7304CF95
7304CF8E 8B06 MOU EAX,DWORD PTR DS:[ESI]
7304CF90 88CE MOU ECX,ESI
7304CF92 FF50 5C CALL DWORD PTR DS:[EAX+5C]
7304CF95 8B08 MOU EBX,EAX
7304CF97 E8 37B6FFFF CALL MFC42.#1577
7304CF9C 5F POP EDI
    POP ECX

```

windows dll(system dll)

여기에 도착한다. 우리가 windows dll 로(system dll) jump 하는 걸 인식한다.

"What now?" you probably ask. Well, we need to know "Is the nag created here ?" or is this only "passing through" ?

"뭐야?" 너는 아마 물을 거야. 좋아, 우리는 "어디에서 nag 이 생성되는지" 알기 위하거나 "통과"가 필요하다.

We can find the answer in different ways because there is no need to verify the jumps in a windows dll : they always stay the same.

우리는 다른 방법으로 answer 를 찾았다. 왜냐하면 그곳은 windows dll 에서 jump 를 검증하기 위해 필요없다. : 그들은 항상 같은 자리에 머물러 있으니까.

Remark : but you do need to verify them in application specific dll's of course !!!!!

주목 : 네가 그것들을 이 application 에서 검증하기 위해 특정 dll 이 필요하다

Different ways to verify are for example placing a memory BP on code section.

검증하기 위한 다른 방법은 이 example 처럼 Code section 의 memory BP 에 놓여있다.

I will discuss that later in another Part.

나는 나중에 다른 part 에서 의논할 것이다.

This time, I want to use hardware BP's (HW BP's). But I will need to give you two words of explanation first

이번 시간에는, hardware BP's 를 사용하기 원한다.(HW BP's). 먼저 2 가지 설명을 하겠다.

Basically, there are 2 kinds of breakpoints we can set : software breakpoints and hardware breakpoints.

기본적으로, 2 가지 breakpoint 를 set 할 수 있다. : software breakpoints and hardware breakpoints
You could suspect a third one, the memory breakpoints, but see these as special software BP's that can be set on the memory Olly has of a process(program).

너는 세번째 Memory breakpoints 의심스럽다. 이것들을 봐라. Special software BP's 가 memory 에서 set 됐을 때 Olly 는 process 가 있습니다.

You can only set one mem BP at a time.

너는 오직 mem 에 BP 를 set 할 수 있다.

If you set another one, the first is removed! To set this breakpoint, OllyDbg needs to alter the attributes of the memory block containing the selection (in chunks of 4096 bytes).

만약에 네가 다른 것을 set 한다면, 첫번째 것은 제거된다. Breakpoints 를 set 하기 위해서, OllyDbg 는 memory block 이 포함하고 있는 범위의 속성이 변하는 게 필요하다.(4096 bytes 단위)
Memory breakpointing just one single byte will force Olly to "guard" the whole block.
Memory breakpointing 을 single byte 만 하면, Olly 가 전체 block 을 "보호" 할 것입니다.

It is clear that this will result in a lot of false breaks.

이것은 명확하다. 꽤 많은 false break 이 발생할 것이다.

Memory BP's substantially lower Olly's speed of processing because Olly constantly also needs to verify the memory he has about the process.

Memory BP's Olly 의 processing speed 는 상당히 낮다. 왜냐하면 Olly 는 끊임없이 memory 검증이 필요하다. Olly 는 그런 process 를 가지고 있다.

I already briefly mentioned the standard software breakpoint in a previous part. In Olly, you can set these by pressing F2 on the line or by doubleclicking the opcodes.

나는 이미 전 Part 에서 Standard software breakpoint 를 간단히 언급했다. Olly 에서 F2 를 누르거나 doubleclick 으로 set 할 수 있다.

Remember that a software BP is set by the debugger (Olly) who needs to "rewrite" the code to place a INT3 instruction (==Trap to Debugger) (== CC opcode) in front of the command.

기억해라. Software BP 는 Debugger 에 의해 set 됐다. Code 에 "rewrite" 하기 위해 INT3 instruction 을 놓기 위해 필요하다.

Without wanting to jump too deep into it : when such a INT3 instruction is encountered, the handler gets control about what's up next. Hehe, and there comes Olly in the game :)

깊게 jump 하기를 원치 않는다면 : IN3 instruction 과 마주칠 때, handler 는 다음 것을 control 하기를 원한다. 헤헤, Olly 가 온다.

Hardware BP's however are completely different. These are originated in their respective debug registers who can each hold one breakpoint address, directly in the CPU.

Hardware BP 는 완벽히 다르다. 각자의 debug register 에서 CPU 에 직접적으로 breakpoint address 를 잡고 있는 것에서 비롯됐다.

You understand already there must be 4 debug registers because there are 4 HW BP's possible.

너는 이미 4 debug register 를 이해하고 있을 것이다. 왜냐하면 4 HW BP 는 가능성이 있다.

Again, without wanting to dig deeper into this : when such a BP is encountered, the CPU generates a INT1 exception which is then given to the exception handler... and that is Olly in our case :)

다시 한번 말하자면, 깊게 jump 를 하고 싶지 않다면 ; BP 는 마주칠 때, CPU 는 INT1 예외를 생산해낸다. INT1 exception 은 그리고 이 case 에서처럼 Olly 에게 exception handler 를 준다.

BTW, HW BP's are not available under win95/98

By the way, HW BP's 는 win95/98 아래 버전은 가능하지 않다.

Your next questions is undoubtedly :

너의 다음 질문은 확실하다.

"And how to know which kind of BP to use?" In fact, it is not difficult.

그래서 어떻게 사용할 때 다른가?" 사실, 이것은 어려운 것이 아니다.

They each have their own restrictions in use. The INT3 BP's (software BP's) for example can basically only be set on commands in the CPU window.

각자 사용할 수 있는 범위의 제약이 있다. 이 예제에서처럼 INT3 BP 는 기본적으로 오직 CPU window 에서 set 될 수 있다.

Also, they can't be set on data or in the middle of a command. In our specific case here, we will use the HW BP's because INT3 BP's in windows dll's are not saved by Olly although there is a way around this with the BP manager plugin or specific settings.

또한, 그들은 명령의 중간에서 data 를 set 할 수 없다. 이런 명확한 경우에, 우리는 HW BP 를 사용할 수 있을 것이다. 왜냐하면 INT3 BP's 는 Olly 에 의해 window dll's 에 저장이 되지 않는다. 다른 방법으로 BP manager plugin 을 쓰거나 명확한 setting 을 한다.

It's clear that when restarting a program, the memory about it was deleted and so, obviously, mem BP's can't be used here either. Some practice will quickly learn you what kind of breakpoint to use in which circumstances.

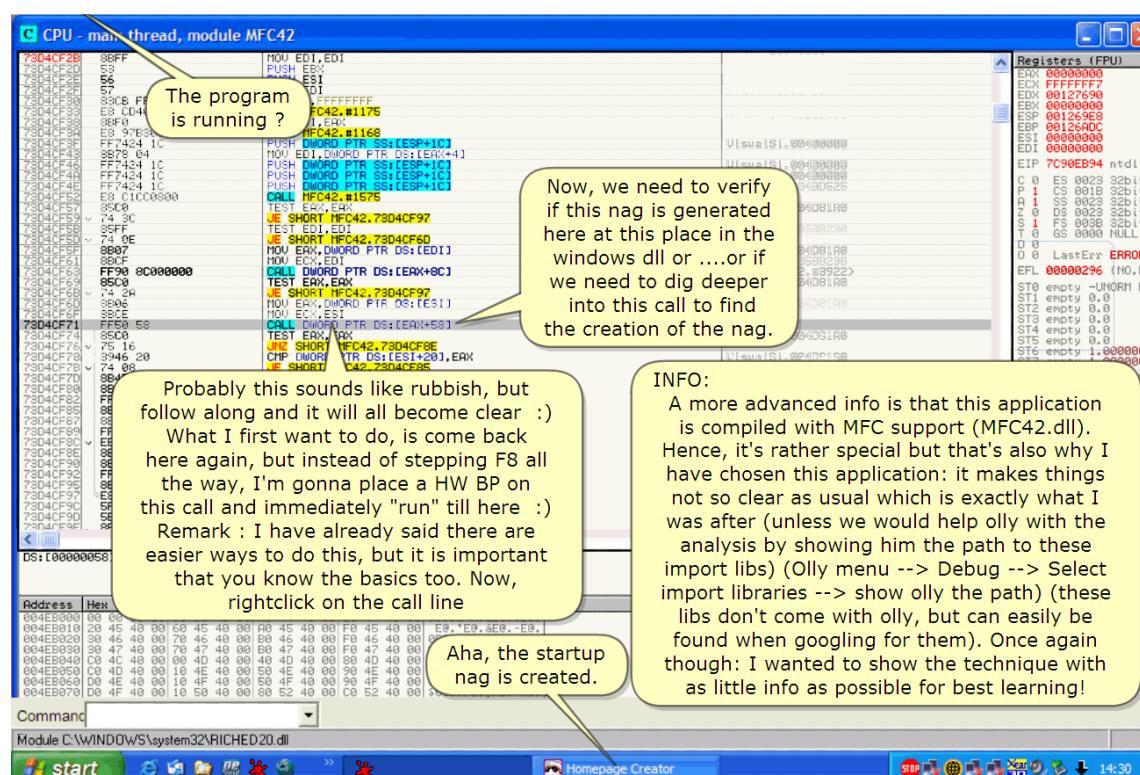
이것은 명확합니다. 우리가 program 을 restart 할 때, memory 는 삭제됐다. 그리고 정확히, memory BP 는 사용될 수 없다. 약간의 연습을 하면 어떤 종류의 breakpoint 를 어떤 상황에서 쓰는지 빨리 배울 수 있습니다.

Now that you understand this, let me show you how to use the HW BP's in this MFC42.dll because you need to understand indeed : the usual software BP's in windows dll's are not saved by Olly.

이제 이것을 이해했을 것이다, 나는 너에게 MFC42.dll 에서 HW BP 를 어떻게 사용하는지 보여주겠다. 왜냐하면 너는 정말로 이해하는 것이 필요하다. : 보통 software BP 는 Olly 에 의해 windows dll's 에서 저장되지 않는다.

I said already that you can circumvent this with the BP manager plugin or specific settings + some other plugins). But first, let's continue stepping F8.

(나는 이미 말했다. 너는 BP manger plug 이나 특정한 setting + 다른 plugin) 피해갈 수 있다. 그러나 먼저, F8 을 눌러 계속 하겠다.



The program is running?

Program 이 실행되고 있니?

Aha, the startup nag is created.

아하, startup nag 는 만들어졌다.

Now, we need to verify if this nag is generated here at this place in the windows dll or Or if we need to dig deeper into this call to find the creation of the nag.

이제, 우리는 검증하는 것이 필요하다. 만약에 이 nag 가 windows dll 이 곳에서 생성됐다거나 우리는 이 call 속으로 깊게 파고들어 nag 을 만드는 곳을 찾는 것이 필요하다.

INFO:

A more advanced info is that this application is compiled with MFC support (MFC42.dll).

많은 발전된 정보는 application 이 MFC 와 함께 compile 됐다.(MFC42.dll)

Hence, it's rather special but that's also why I have chosen this application: it makes things not so clear as usual which is exactly what I was after

그리하여, 이것은 꽤 special 하다. 그러나 이것은 또한 나는 이 application 을 선택했다 : things 가 일반적일 때 명확하지 않다. 그것은 정확히 였다.

(unless we would help Olly with the analysis by showing him the path to these import libs)

(우리가 Import libs 경로를 보여줌으로써 분석하는 Olly 를 도와줄 수 없다면)

(Olly menu --> Debug --> Select import libraries --> show Olly the path)

(Olly menu --> Debug --> Select import libraries --> show Olly the path)

(these libs don't come with Olly, but can easily be found when googling for them).

이 lib 들은 Olly 와 함께 오지 않는다. 그러나 구글링으로 그것들을 쉽게 찾을 수 있다.

Once again though: I wanted to show the technique with as little info as possible for best learning!

다시 한 번 말한다 : 나는 기술적인 부분을 가능한 최고의 공부가 될 수 있도록 약간의 정보와 함께 보여주겠다.

Probably this sounds like rubbish, but follow along and it will all become clear :)

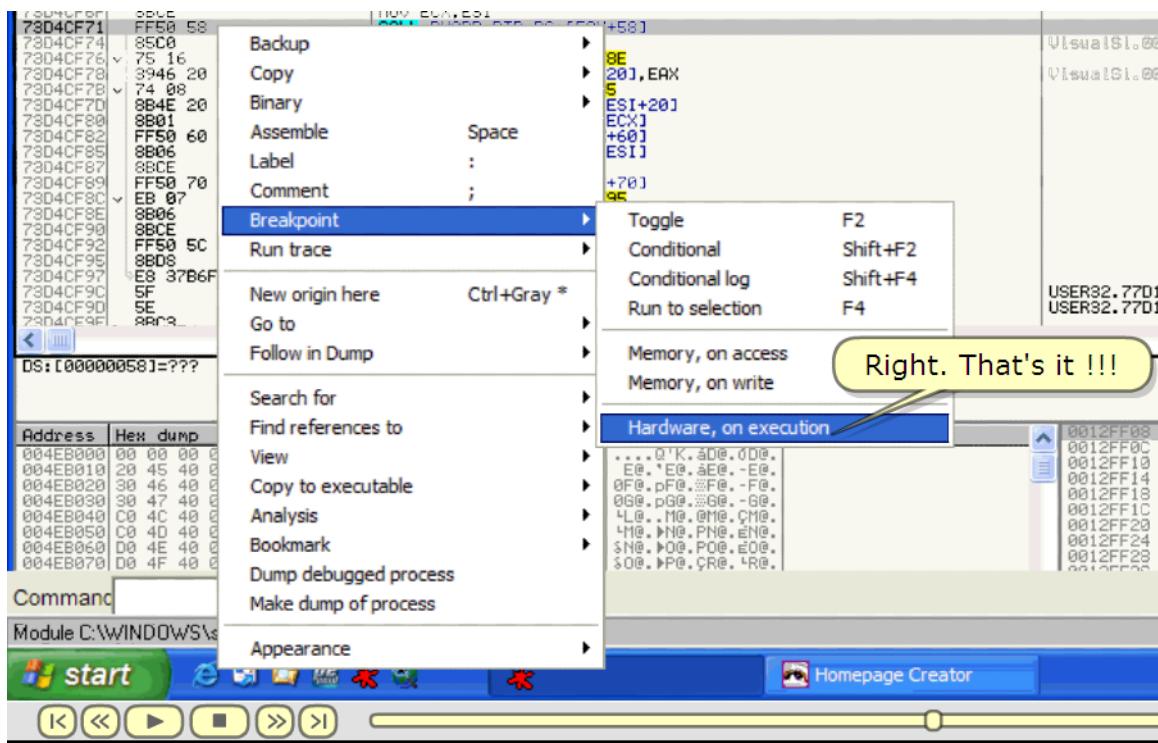
아마 미친놈 같은 소리일 것이다, 그러나 따라와. 그러면 명확해 질 것이다.

What I first want to do, is come back here again, but instead of stepping F8 all the way, I'm gonna place a HW BP on this call and immediately "run" till here :)

내가 먼저 무엇을 해야 될까, 이곳에 돌아왔다. 그러나 F8 누르는 방법 대신에, 나는 이 call 에 HW 를 설치하겠다. 그리고 즉시 이곳까지 실행해.

Remark : I have already said there are easier ways to do this, but it is important that you know the basics too. Now, rightclick on the call line

주목 : 나는 이미 말했다. 그곳은 매우 쉬운 방법이다. 그러나 이것은 매우 너도 알다시피 기본적인 것이 중요하다. 이제, call line 에서 rightclick 해.



Right. That's it !!!

C CPU - main thread, module MFC42

```

73D4CF2B 8BFF MOU EDI,EDI
73D4CF2D 53 PUSH EBX
73D4CF2E 56 PUSH ESI
73D4CF2F 57 PUSH EDI
73D4CF30 83CB FF OR EBX,FFFFFFF
73D4CF33 E3 CD48FFFF CALL MFC42.#1175
73D4CF38 8BF0 MOV ESI,ESI
73D4CF39 E3 97B30800 CALL MFC42.#1168
73D4CF3F FF7424 1C PUSH DWORD PTR SS:[ESP+1C]
73D4CF43 3B78 04 MOV EDI,DWORD PTR DS:[ECX+4]
73D4CF46 FF7424 1C PUSH DWORD PTR SS:[ESP+1C]
73D4CF4D E3 2A24 1C PUSH DWORD PTR SS:[ESP+1C]
73D4CF50 FF90 8C00 00 CALL DWORD PTR DS:[EAX+8C]
73D4CF53 85C0 TEST EAX,EAX
73D4CF56 74 3C JE SHORT MFC42.73D4CF97
73D4CF58 3B3E 00 MOV ECX,EDX
73D4CF5A 83CE 00 MOU ECX,ESI
73D4CF5C FF50 58 CALL DWORD PTR DS:[EAX+58]
73D4CF5F 85C0 TEST EAX,EAX
73D4CF62 74 3C JE SHORT MFC42.73D4CF8E
73D4CF64 3B3E 00 MOV ECX,EDX
73D4CF66 FF50 70 CALL DWORD PTR DS:[EAX+70]
73D4CF68 85C0 TEST EAX,EAX
73D4CF6B 74 3C JE SHORT MFC42.73D4CF81
73D4CF6D FF50 58 CALL DWORD PTR DS:[EAX+58]
73D4CF70 85C0 TEST EAX,EAX
73D4CF73 74 3C JE SHORT MFC42.73D4CF81
73D4CF76 85C0 TEST EAX,EAX
73D4CF78 74 3C JE SHORT MFC42.73D4CF81
73D4CF7A 85C0 TEST EAX,EAX
73D4CF7C 74 3C JE SHORT MFC42.73D4CF81
73D4CF7E 85C0 TEST EAX,EAX
73D4CF80 74 3C JE SHORT MFC42.73D4CF81
73D4CF82 FF50 5C CALL MFC42.#1577
73D4CF84 8BD8 MOV EBX,ERX
73D4CF86 FF50 70 CALL MFC42.#1577
73D4CF88 85C0 TEST EAX,EAX
73D4CF8A 74 3C JE SHORT MFC42.73D4CF81
73D4CF8C 85C0 TEST EAX,EAX
73D4CF8E 74 3C JE SHORT MFC42.73D4CF81
73D4CF90 85C0 TEST EAX,EAX
73D4CF92 85C0 TEST EAX,EAX
73D4CF94 FF50 70 CALL MFC42.#1577
73D4CF96 85C0 TEST EAX,EAX
73D4CF98 74 3C JE SHORT MFC42.73D4CF81
73D4CF9A 85C0 TEST EAX,EAX
73D4CF9C FF50 70 CALL MFC42.#1577
73D4CF9E 85C0 TEST EAX,EAX
73D4CF9F FF50 70 CALL MFC42.#1577

```

Remember: stepping F8 and the nag appears means : put BP and restart + run + F7 in the call + F8 again till nag appears...

...to dig deeper in the call in the same way, always deeper and deeper till the nag appears.

Remember: stepping F8 and the nag appears means : put BP and restart + run + F7 in the call + F8 again till nag appears...

기억해 : F8로 넘기고 nag은 나타난다. : BP를 설치하고 restart + run + F7을 눌러 call 안으로 + nag이 나타날 때까지 F8을 계속 눌러

...to dig deeper in the call in the same way, always deeper and deeper till the nag appears.

Call에서 같은 방법으로 계속 파고들면, 항상 깊게 그리고 또 깊게 nag이 나타날 때까지 파고들어.

To reduce movie size, I have done this but I have removed it from the movie

Movie size를 줄이기 위해서 삭제했다. 나는 이미 했어. 그러나 movie에서 삭제했다.

and so, we land here again in the same call in

... and so, we step F7 in the call.

INFO : remember :

F8 == stepping over, executing the complete call with all underlying code at once

F7 == step in the call, no code is executed yet

Address	Hex dump	ASCII	Comment
004EB000	00 00 00 00	40 00 E0 44 40 00	0012FF08
004EB010	20 45 40 00	40 00 F0 45 40 00	0012FF0C
004EB020	30 46 40 00	70 46 40 00	0012FF10
004EB030	30 47 40 00	70 47 40 00	0012FF14
004EB040	C0 40 40 00	00 40 40 00	0012FF18
004EB050	C0 40 40 00	10 40 40 00	0012FF1C
004EB060	D0 40 40 00	10 4F 40 00	0012FF20
004EB070	D0 4F 40 00	10 50 40 00	0012FF24

DS:[004D81F8]=00489310 (VisualStudio.00489310)

the HW BP

Hardware breakpoint 1 at MFC42.73D4CF71

And so, we land here again in the same call in

그래서, 우리는 같은 call 안의 이곳에 다시 도착했다.

The HW BP

Hardware breakpoint 1 at MFC42.73D4CF71

... and so, we step F7 in the call.

그리고, 우리는 call 안으로 들어가기 위해 F7을 눌러.

INFO : remember :

F8 == stepping over, executing the complete call with all underlying code at once

F7 == step in the call, no code is executed yet

기억해 :

F8 == 넘기는 것, call 을 한번에 근본적으로 code 를 완벽히 실행 하는 것.

F7 == call 안으로 들어가기, code 는 아직 실행되지 않았다.

Bam !! To land again in the main exe.

Notice that it was indeed NOT where the nag is created

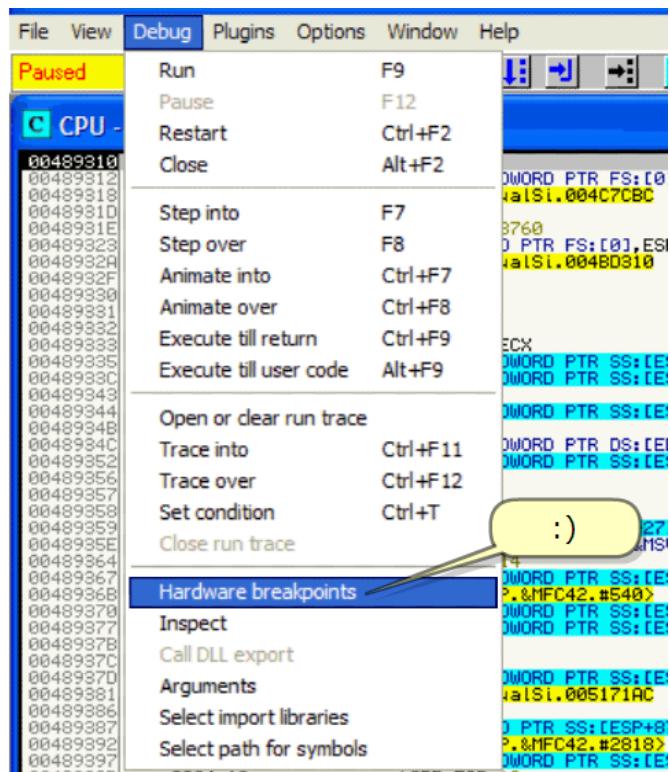
!! 우리는 다시 main.exe에 도착했다.

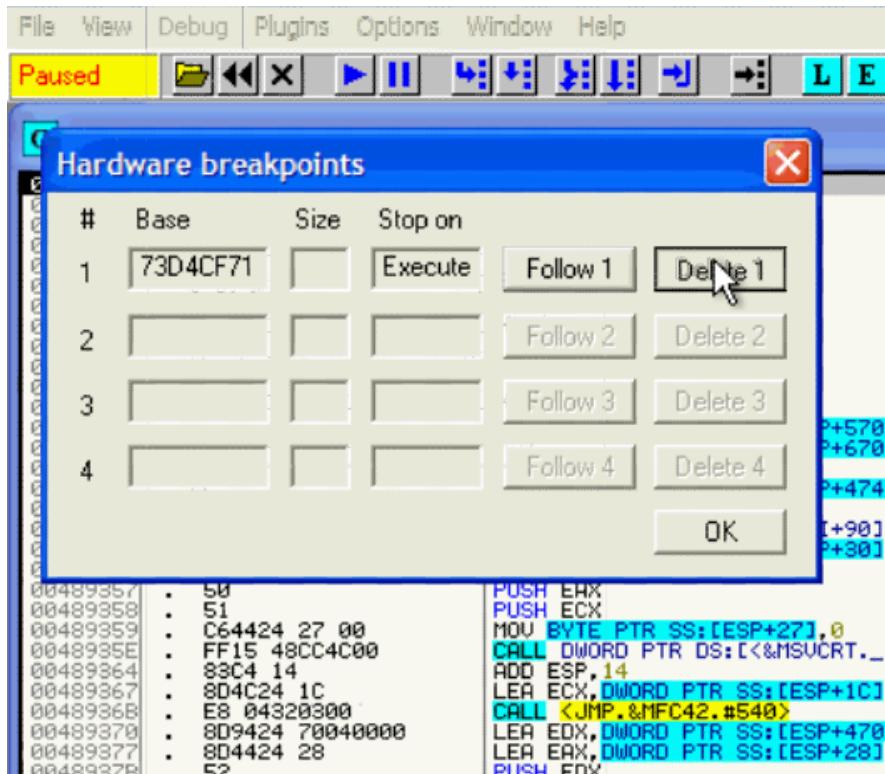
그곳은 nag 을 생성하지 않는다.

INFO:

If you are a coder, then this is clear and known beforehand: of course this wasn't where the nag is created. For the non-coder, a little experience and API's will learn this

만약에 네가 code 라면, 이것은 명확하다 그리고 사전에 알았다 : 물론 이것은 nag 이 만들어지는 곳이 아니다. Noncoder 라면, 약간의 경험과 API 를 배워야 한다.





Because we won't need the HW BP anymore, and because we have only 4 HW BPs at our disposal, just remove it like this.

왜냐하면 우리는 HW BP 가 더 이상 필요하지 않다. 그리고 우리는 오직 4 개의 HW BP 를 처리할 수 있다. 이제 이것을 삭제하자.

And now continue stepping F8 on the search for the creation of the NAG.

그리고 nag 이 만들어지는 곳을 찾으려고 계속 F8 을 눌러.

Notice also that we are getting deeper and deeper into the call at VA 004BD497 where the nag was first shown !!! Still remember that ?

우리는 깊게 파고들고 call 안의 004BD497 에 도착했다. 이곳은 nag 이 처음으로 보여지는 곳이다. 기억하고 있어?

And do you still remember the animate over/in function in Olly from the very beginning in this Part?

매우 기초적인 Part 에서 했던 Olly 의 animate over/in function 을 기억하고 있어?

Now, this is the time to come back to that Ctrl-F8 you pressed then.

이제, 돌아올 시간이 됐다. Ctrl-F8 을 눌러.

If you have tried it, you will have seen that Olly animated straight towards the nag following the exact path we too have followed so far and further !!!

도전해 봤다면, Olly 가 직접 nag 이 실행되는 정확한 경로를 animation 으로 봤을 것이다. 우리는 지금까지 그리고 더 멀리 따라간다.

What we could have done is the following : from EP, press Ctrl-F8 to animate over the code till the NAG where Olly pauses.

우리는 따라 다녔다. : EP에서, Ctrl+F8을 눌러. Nag이 있는 곳까지 animation 시켜. 그곳에서 Olly는 정지한다.

Then press "-" to retrace Olly's steps till back at EP(this may take a while) and then retrace the steps Olly has made, noting down all the cond jumps as well as the variable jumps.

-를 눌러. Olly는 원래 왔던 곳인 EP를 추적한다.(잠시 동안) 그리고 Olly가 만든 step을 추적한다. This way, you never need to BP, restart the app and run till the BP to step F7 in the call to search for the nag.

이 방법은, 네게 BP가 더 이상 필요하지 않다. App을 재시작하고 BP까지 실행해. F7을 눌러 call 안으로 들어가서 nag을 찾자.

REMARK : this is exactly what TC (trace call == run trace) can do and more. In fact, it would be stupid to do it with animate over/in.

주목 : 이것은 정확히 TC(trace call == run trace)은 좀 더 할 수 있다. 사실, 이것은 명청한 짓이다. Animate over/in으로 할 수 있다.

My only goal here is to explain a little more in-depth what all possibilities you have in Olly.

오직 이곳에서 나의 목표는 약간의 깊이 있는 것을 설명하기 위한 모든 가능성을 Olly는 가지고 있다.

Now that you know all this, you will also understand that when animating over with Olly, I immediately found where the nag is created, and I have put a BP there already.

이제 너도 모든 것을 알 것이다. Animating이 될 때 이해할 것이다. 나는 즉시 Nag이 생성된 곳을 찾는다. 그리고 이미 BP를 설치해 놨다.

One important remark however : always check the last steps from Olly till the created window to verify if they are really the last ones.

중요한 것은 이것이다 주목 : 만약에 그들이 정말 마지막 것이라면 항상 Olly에서 만들어진 window를 검증하기 위해 마지막 step을 check 한다.

Olly sometimes animates OVER the last call -showing you the window too soon-instead of animating IN it....

Olly는 가끔 마지막 call을 보여주기 위해서.(너에게 빨리 보여주기 위해) animate를 한다.

Now comes a long way of stepping F8 without jump changes. To reduce movie size, I have removed it from this movie, and so, we arrive

Jump 없이 F8을 눌러 진행하는 다른 방법이 온다. Movie size를 줄이기 위해, 나는 이 movie에서 삭제했다. 그리고 우리는 도착했다.

...here, in a JE that again brings no changes to before.

여기, JE 는 전과 마찬가지로 바뀌지 않는다.

This jump could have been important though because it could jump to an XOR (setting register to zero)

이 jump 는 중요했다. 왜냐하면 그것은 XOR 로 jump 할 수 있다.(register 를 zero 로 setting)

Remark where the nag was created when I first stepped this program. Let's continue stepping. Pay attention because this is undoubtedly where it will all happen

주목. 처음으로 program 을 진행할 때 이 곳에서 nag 은 생성됐다. 계속 해보자.

집중해. 왜냐하면 이것은 무슨 일이 발생할지 확실하다.

CPU - main thread, module VisualSI	
004898900	.v 74 0B
004898901	> 8BC8
004898902	E8 E732FFFF
004898903	33ED
004898904	8BCF
004898905	MOU ECX,ECX
004898906	CALL VisualSI.0047CB90
004898907	MOU EBP,EO1
004898908	JMP SHORT
004898909	XOR EBP,EBP
00489890A	MOU ECX,ECX
00489890B	MOU BYTE PTR SS:[ESP+20]
00489890C	MOU DWORD PTR SS:[ESP+20]
00489890D	CALL VisualSI.00489890
00489890E	FUSH 0
00489890F	MOU ECX,EO1
004898910	CALL K JMP &MFC42.#5503
004898911	MOU ECX,EO1
004898912	CALL VisualSI.00489890
004898913	TEST AL,AL
004898914	JE VisualSI.004898908
004898915	MOV AL,BYTE PTR DS:[EDI+E0]
004898916	TEST AL,AL
004898917	JNZ VisualSI.004898A29
004898918	MOU EDI,DWORD PTR DS:[EDI+E4]
004898919	PUSH EDI
00489891A	TEST ERX,ERX
00489891B	JE VisualSI.00489998
00489891C	LEA ECX,DWORD PTR SS:[ESP+210]
00489891D	CALL VisualSI.004ABD070
00489891E	LEA ECX,DWORD PTR SS:[ESP+20C]
00489891F	MOV BYTE PTR SS:[ESP+8778],0B
004898920	CALL K JMP &MFC42.#2514
004898921	CMP EAX,1
004898922	JE SHORT VisualSI.00489950
004898923	MOV ECX,EO1
004898924	CALL VisualSI.00488F30
004898925	LEA ECX,DWORD PTR SS:[ESP+2B0]
004898926	MOU BYTE PTR SS:[ESP+8778],0D
004898927	CALL K JMP &MFC42.#7652
004898928	LEA ECX,DWORD PTR SS:[ESP+270]
004898929	MOU BYTE PTR SS:[ESP+8778],0C
004898930	CALL K JMP &MFC42.#7952
004898931	MOU BYTE PTR SS:[ESP+8778],5
004898932	LEA ECX,DWORD PTR SS:[ESP+20C]
004898933	JMP SHORT VisualSI.00489903
004898934	LEA ECX,DWORD PTR SS:[ESP+2B0]
004898935	MOU BYTE PTR SS:[ESP+8778],0F
004898936	CALL K JMP &MFC42.#74E5
004898937	
004898938	
004898939	
004898940	
004898941	
004898942	
004898943	
004898944	
004898945	
004898946	
004898947	
004898948	
004898949	
004898950	
004898951	
004898952	
004898953	
004898954	
004898955	
004898956	
004898957	
004898958	
004898959	
004898960	
004898961	
004898962	
004898963	
004898964	
004898965	
004898966	
004898967	
004898968	
004898969	
004898970	
004898971	
004898972	
004898973	
004898974	
004898975	
004898976	
004898977	
004898978	
004898979	
004898980	
004898981	
004898982	
004898983	
004898984	
004898985	
004898986	
004898987	
004898988	
004898989	
004898990	
004898991	
004898992	
004898993	
004898994	
004898995	
004898996	
004898997	
004898998	
004898999	
004898900	
004898901	
004898902	
004898903	
004898904	
004898905	
004898906	
004898907	
004898908	
004898909	
004898910	
004898911	
004898912	
004898913	
004898914	
004898915	
004898916	
004898917	
004898918	
004898919	
004898920	
004898921	
004898922	
004898923	
004898924	
004898925	
004898926	
004898927	
004898928	
004898929	
004898930	
004898931	
004898932	
004898933	
004898934	
004898935	
004898936	
004898937	
004898938	
004898939	
004898940	
004898941	
004898942	
004898943	
004898944	
004898945	
004898946	
004898947	
004898948	
004898949	
004898950	
004898951	
004898952	
004898953	
004898954	
004898955	
004898956	
004898957	
004898958	
004898959	
004898960	
004898961	
004898962	
004898963	
004898964	
004898965	
004898966	
004898967	
004898968	
004898969	
004898970	
004898971	
004898972	
004898973	
004898974	
004898975	
004898976	
004898977	
004898978	
004898979	
004898980	
004898981	
004898982	
004898983	
004898984	
004898985	
004898986	
004898987	
004898988	
004898989	
004898990	
004898991	
004898992	
004898993	
004898994	
004898995	
004898996	
004898997	
004898998	
004898999	
004898900	
004898901	
004898902	
004898903	
004898904	
004898905	
004898906	
004898907	
004898908	
004898909	
004898910	
004898911	
004898912	
004898913	
004898914	
004898915	
004898916	
004898917	
004898918	
004898919	
004898920	
004898921	
004898922	
004898923	
004898924	
004898925	
004898926	
004898927	
004898928	
004898929	
004898930	
004898931	
004898932	
004898933	
004898934	
004898935	
004898936	
004898937	
004898938	
004898939	
004898940	
004898941	
004898942	
004898943	
004898944	
004898945	
004898946	
004898947	
004898948	
004898949	
004898950	
004898951	
004898952	
004898953	
004898954	
004898955	
004898956	
004898957	
004898958	
004898959	
004898960	
004898961	
004898962	
004898963	
004898964	
004898965	
004898966	
004898967	
004898968	
004898969	
004898970	
004898971	
004898972	
004898973	
004898974	
004898975	
004898976	
004898977	
004898978	
004898979	
004898980	
004898981	
004898982	
004898983	
004898984	
004898985	
004898986	
004898987	
004898988	
004898989	
004898990	
004898991	
004898992	
004898993	
004898994	
004898995	
004898996	
004898997	
004898998	
004898999	
004898900	
004898901	
004898902	
004898903	
004898904	
004898905	
004898906	
004898907	
004898908	
004898909	
004898910	
004898911	
004898912	
004898913	
004898914	
004898915	
004898916	
004898917	
004898918	
004898919	
004898920	
004898921	
004898922	
004898923	
004898924	
004898925	
004898926	
004898927	
004898928	
004898929	
004898930	
004898931	
004898932	
004898933	
004898934	
004898935	
004898936	
004898937	
004898938	
004898939	
004898940	
004898941	
004898942	
004898943	
004898944	
004898945	
004898946	
004898947	
004898948	
004898949	
004898950	
004898951	
004898952	
004898953	
004898954	
004898955	
004898956	
004898957	
004898958	
004898959	
004898960	
004898961	
004898962	
004898963	
004898964	
004898965	
004898966	
004898967	
004898968	
004898969	
004898970	
004898971	
004898972	
004898973	
004898974	
004898975	
004898976	
004898977	
004898978	
004898979	
004898980	
004898981	
004898982	
004898983	
004898984	
004898985	
004898986	
004898987	
004898988	
004898989	
004898990	
004898991	
004898992	
004898993	
004898994	
004898995	
004898996	
004898997	
004898998	
004898999	
004898900	
004898901	
004898902	
004898903	
004898904	
004898905	
004898906	
004898907	
004898908	
004898909	
004898910	
004898911	
004898912	
004898913	
004898914	
004898915	
004898916	
004898917	
004898918	
004898919	
004898920	
004898921	
004898922	
004898923	
004898924	
004898925	
004898926	
004898927	
004898928	
004898929	
004898930	
004898931	
004898932	
004898933	
004898934	
004898935	
004898936	
004898937	
004898938	
004898939	
004898940	
004898941	
004898942	
004898943	
004898944	
004898945	
004898946	
004898947	
004898948	
004898949	
004898950	
004898951	
004898952	
004898953	
004898954	
004898955	
004898956	
004898957	
004898958	
004898959	
004898960	
004898961	
004898962	
004898963	
004898964	
004898965	
004898966	
004898967	
004898968	
004898969	
004898970	
004898971	
004898972	
004898973	
004898974	
004898975	
004898976	
004898977	
004898978	
004898979	
004898980	
004898981	
004898982	
004898983	
004898984	
004898985	
004898986	
004898987	
004898988	
004898989	
004898990	
004898991	
004898992	
004898993	
004898994	
004898995	
004898996	
004898997	
004898998	
004898999	
004898900	
004898901	
004898902	
004898903	
004898904	
004898905	
004898906	
004898907	
004898908	
004898909	
004898910	
004898911	
004898912	
0048989	

Remark that this JE if executed would jump over the pag

Now scroll down to see the jump

주목. JE 가 실행된다면 naq 을 jump 할 것이다.

스크롤 밑으로 내려서 jump 하는 곳을 봄.

C CPU - main thread, module VisualSi

```
004899F0 . 80D8C24 10020000 LEA ECX,DWORD PTR SS:[ESP+210] 004899F5 . E8 D0240200 CALL VisualSi.004AB070 004899F9 . 80D8C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+20C] 004899FD . C68424 78870000 0B MOU BYTE PTR SS:[ESP+8778],0B 004899F12 . E8 132B0300 CALL K JMP.&MFC42.#2514> 83F8 81 CMP EAX,1 004899F13 . JE SHORT VisualSi.00489950 MOU ECX,EDI 004899F14 . CALL VisualSi.004898F0 004899F18 . 80D8C24 80020000 LEA ECX,DWORD PTR SS:[ESP+280] 004899F22 . C68424 78870000 0D MOU BYTE PTR SS:[ESP+8778],0D 004899F26 . E8 81300600 CALL K JMP.&MFC42.#765> 004899F2A . 80D8C24 78870000 0E 004899F30 . C68424 78870000 0C 004899F34 . E8 DB2B0300 CALL K JMP.&MFC42.#795> 004899F38 . C68424 78870000 05 004899F42 . 80D8C24 00020000 MOU BYTE PTR SS:[ESP+8778],5 004899F46 . E8 77 004899F50 . 80D8C24 80020000 004899F54 . C68424 78870000 0F 004899F58 . E8 48300300 004899F62 . 80D8C24 78870000 0C 004899F66 . C68424 78870000 0E 004899F70 . E8 A22B0300 004899F74 . C68424 78870000 05 004899F78 . 80D8C24 0C020000 004899F82 . E8 8C000000 004899F86 . 80D8C24 B0010000 004899F90 . E8 CCEB0100 004899F94 . 80D8C24 AC010000 004899F98 . C68424 78870000 10 004899FB3 . E8 722A0300 004899FB7 . 83F8 81 004899FBB . 74 58 004899FBF . 80D8C24 2C 004899FC . E8 6C5FFFFF 004899CC . C68424 7887 004899D3 . 80D8C24 20 004899D7 . E8 192E0300 004899E4 . 80D8C24 14 004899E9 . C68424 78870000 00 004899F0 . E8 622B0300
```

Jump is NOT taken

Mmmm, jumping could perhaps be a solution but let's first see what happens without trying this.

Scroll back up.

Later, we will find that this jump leads to program's exit.

Mmmm, jumping could perhaps be a solution but let's first see what happens without trying this.

음, jumping 은 아마 해결책이다. 도전 없이 먼저 무엇이 발생했는지 봐.

Scroll back up.

스크롤을 원래대로 돌려.

Later, we will find that this jump leads to program's exit.

우리는 jump 가 program 을 exit 하는 곳으로 lead 할 것이다.

C CPU - main thread, module VisualSi

```
004898CC . E8 AFF0FFFF
004898D1 . 84C0
004898D3 . v 0F84 FF000000
004898D9 . 8487 E0000000
004898DF . 84C0
004898E1 > 0F85 42010000
004898E7 . 8887 E0000000
004898ED . 6A
004898EF . 85C0
004898F1 . v 0F8E A1000000
004898F7 . 808C24 10020000
004898FE . E8 6D240200
00489903 . 808C24 0C020000
0048990A . C68424 78870000 0B
00489917 . E8 13200300
0048991A . 88F8 01
0048991C . v 74 40
0048991E . 88CF
0048991F . E8 0DF6FFFF
00489923 . 808C24 B0020000
0048992H . C68424 78870000 0D
00489932 . E8 81300500
00489937 . 808C24 70020000
0048993E . C68424 78870000 0C
00489946 . E8 DE2B0300
0048994B . C68424 78870000
00489953 . 808C24 0C020000
0048995A . v EB 77
0048995C > 808C24 B0020000
00489963 . C68424 78870000
0048996B . E8 48300300
00489970 . 808C24 70020000
00489977 . C68424 78870000 0E
0048997F . E8 A22B0300
00489984 . 808C24 80010000
0048998C . 808C24 0C020000
00489993 . v E8 8C000000
0048999F . 808C24 B0010000
004899A4 . E8 CC840100
004899A8 . 808C24 AC010000
004899B3 . C68424 78870000 10
004899B8 . E8 722A0300
004899B9 . 88F8 01
004899B9 > v 74 58
004899B9 RARE
004899B9
```

JMP is NOT taken
00489A29=VisualSi.00489A29

Mmmm, and this JNZ, if executed, jumps past the Nag too
This is another possible solution

Mmmm, and this JNZ, if executed, jumps past the Nag too

This is another possible solution

음, 이 JNZ 가 실행된다면, nag 을 jump 한다.

이것은 다른 해결책이다.

C CPU - main thread, module VisualSi

```

00489980F . 84C0 TEST AL,AL
0048998E1 . ✓ 0F85 42010000 JNC VisualSi.00489A29
0048998E7 . 8B87 E4000000 MOU ECX,DWORD PTR DS:[EDI+E4]
0048998ED . 6A 00 PUSH 0
0048998EF . 85C0 TEST EAX,EAX
0048998F1 . ✓ 0F8E R10000000 JLE VisualSi.00489998
0048998F7 . 808C24 10020000 LEA ECX,DWORD PTR SS:[ESP+210]
0048998FE . E8 6D240200 CALL VisualSi.004AB070
004899903 . 808C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+20C]
00489990A . C68424 78870000 0B MOU BYTE PTR SS:[ESP+87781],0B
00489912 . E8 132B0300 CALL KJMP.&HFC42.#2514
00489917 . 83F8 01 CMP EAX,1
0048991A . ✓ 74 40 JE SHORT VisualSi.0048999C
0048991C . 8BCF MOU ECX,EDI
0048991E . E8 0DF6FFFF NAG
00489923 . 808C24 B0020000 V VisualSi.0058B230
0048992A . C68424 78870000 0E
00489932 . E8 8130
00489937 . C68424 78870000 0F
0048993E . C68424 78870000 0B
00489946 . C68424 78870000 0C
0048994B . C68424 78870000 0D
00489953 . 808C24 0C020000
0048995A . ✓ EB 77 LEA ECX,DWORD PTR SS:[ESP+20C]
0048995C . > 808C24 B0020000 JMP SHORT VisualSi.00489903
00489963 . C68424 78870000 0F LEA ECX,DWORD PTR SS:[ESP+2B0]
0048996B . E8 48300300 MOU BYTE PTR SS:[ESP+87781],0F
00489970 . 808C24 70020000 CALL KJMP.&HFC42.#765
00489977 . C68424 78870000 0E LEA ECX,DWORD PTR SS:[ESP+270]
0048997F . E8 A22B0300 MOU BYTE PTR SS:[ESP+87781],0E
00489984 . C68424 78870000 05 CALL KJMP.&HFC42.#795
00489985 . 808C24 0C020000 MOU BYTE PTR SS:[ESP+87781],5
00489993 . ✓ E9 8C000000 LEA ECX,DWORD PTR SS:[ESP+20C]
00489994 . 808C24 B0010000 JMP VisualSi.00489A24
0048999F . E8 CCB40100 LEA ECX,DWORD PTR SS:[ESP+1B0]
0048999B . 808C24 AC010000 MOU BYTE PTR SS:[ESP+97781],10
0048999B . C68424 78870000 10 CALL KJMP.&HFC42.#2514
0048999B . E8 722A0300 CMP EAX,1
0048999B . 83F8 01 JE SHORT VisualSi.00489A15
0048999B . ✓ 74 58 MOU ECX,EDI
004899BD . 8BCF CALL VisualSi.00488F30
004899C4 . E8 6CF5FFFF VisualSi.0058B230
004899CC . C68424 78870000 05 MOU BYTE PTR SS:[ESP+87781],5
004899C4 . ; 808C24 AC010000 LEA ECX,DWORD PTR SS:[ESP+1AC]
004899C4 . ; FR 40200000 CALL KJMP.&HFC42.#A415

```

Oho, and this is the one that has changed!
See how easy it is in this way to find what needs to be patched !!!!

Oho, and this is the one that has changed!

See how easy it is in this way to find what needs to be patched !!!!

오호, 이것은 바뀌었다.

패치가 필요한데 그 방법이 얼마나 쉬운지 이 방법을 통해 찾아보자.

C CPU - main thread, module VisualSi

```

00489980F . 84C0 TEST AL,AL
0048998E1 . ✓ 0F85 42010000 JNC VisualSi.00489A29
0048998E7 . 8B87 E4000000 MOU ECX,DWORD PTR DS:[EDI+E4]
0048998ED . 6A 00 PUSH 0
0048998EF . 85C0 TEST EAX,EAX
0048998F1 . ✓ 0F8E R10000000 JLE VisualSi.00489998
0048998F7 . 808C24 10020000 LEA ECX,DWORD PTR SS:[ESP+210]
0048998FE . E8 6D240200 CALL VisualSi.004AB070
004899903 . 808C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+20C]
00489990A . C68424 78870000 0B MOU BYTE PTR SS:[ESP+87781],0B
00489912 . E8 132B0300 CALL KJMP.&HFC42.#2514
00489917 . 83F8 01 CMP EAX,1
0048991A . ✓ 74 40 JE SHORT VisualSi.0048999C
0048991C . 8BCF MOU ECX,EDI
0048991E . E8 0DF6FFFF NAG
00489923 . 808C24 B0020000 V VisualSi.0058B230
0048992A . C68424 78870000 0E
00489932 . E8 8130
00489937 . C68424 78870000 0F
0048993E . C68424 78870000 0B
00489946 . C68424 78870000 0C
0048994B . C68424 78870000 0D
00489953 . 808C24 0C020000
0048995A . ✓ EB 77 LEA ECX,DWORD PTR SS:[ESP+20C]
0048995C . > 808C24 B0020000 JMP SHORT VisualSi.00489903
00489963 . C68424 78870000 0F LEA ECX,DWORD PTR SS:[ESP+2B0]
0048996B . E8 48300300 MOU BYTE PTR SS:[ESP+87781],0F
00489970 . 808C24 70020000 CALL KJMP.&HFC42.#765
00489977 . C68424 78870000 0E LEA ECX,DWORD PTR SS:[ESP+270]
0048997F . E8 A22B0300 MOU BYTE PTR SS:[ESP+87781],0E
00489984 . C68424 78870000 05 CALL KJMP.&HFC42.#795
00489985 . 808C24 0C020000 MOU BYTE PTR SS:[ESP+87781],5
00489993 . ✓ E9 8C000000 LEA ECX,DWORD PTR SS:[ESP+20C]
00489994 . 808C24 B0010000 JMP VisualSi.00489A24
0048999F . E8 CCB40100 LEA ECX,DWORD PTR SS:[ESP+1B0]
0048999B . 808C24 AC010000 MOU BYTE PTR SS:[ESP+97781],10
0048999B . C68424 78870000 10 CALL KJMP.&HFC42.#2514
0048999B . E8 722A0300 CMP EAX,1
0048999B . 83F8 01 JE SHORT VisualSi.00489A15
004899BD . 8BCF CALL VisualSi.00488F30
004899C4 . E8 6CF5FFFF VisualSi.0058B230
004899CC . C68424 78870000 05 MOU BYTE PTR SS:[ESP+87781],5
004899C4 . ; 808C24 AC010000 LEA ECX,DWORD PTR SS:[ESP+1AC]
004899C4 . ; FR 40200000 CALL KJMP.&HFC42.#A415

```

Notice that this jump, now executed because of trial expiration, will bring us over the nag. Let's first see what happens now before deciding what and where to patch.
So press F8 to take the jump.

Notice that this jump, now executed because of trial expiration, will bring us over the nag.

이 jump 는, 왜냐하면 trial 만기를 실행했다. Nag 을 실행하지 않는다.

Let's first see what happens now before deciding what and where to patch.

So press F8 to take the jump

첫번째로 무엇이 발생하는지 보고 무엇을 그리고 어디에 patch 할지 결정해.

그래서 F8 을 눌러 jump 를 하자.

And we land here.

Continue stepping F8

우리는 여기에 도착한다.

F8 을 눌러 계속하자.

C CPU - main thread, module VisualSi

004898DF . 84C0 TEST AL,AL	004898E1 . v 0F8E 42010000 JNZ VisualSi.00489A29	X
004898E7 . 3B87 E4000000 MOV EAX,DWORD PTR DS:[EDI+E4]	004898E9 6A 00 PUSH 0	
004898ED . 85C0 TEST EAX,EAX	004898EF JLE VisualSi.00489998	
004899F1 . v 7FSE A1000000 LEA ECX,DWORD PTR SS:[ESP+210]	004899F7 . 80EC24 10020000 CALL VisualSi.00489B01	X
004899FE . E8 60240200 MOU BYTE PTR SS:[ESP+778],0B	004899F3 . 80EC24 0C020000 MOU BYTE PTR SS:[ESP+200]	
004899F0 . C66424 78870000 0B CALL K JMP,&MFC42,#2514	004899F4 . E8 182B0300 CMP EAX,1	NAG
004899F1 . 83F8 01 JE SHORT VisualSi.0048995C	004899F7 . 74 40 MOU ECX,EDI	V
004899F2 . 0048991C CALL VisualSi.00489520	004899F8 . E9 00F6FFFF LEA ECX,DWORD PTR SS:[ESP+210]	
004899F3 . 80EC24 B0020000 MOU BYTE PTR SS:[ESP+200]	004899F9 . C66424 78870000 0D CALL K JMP,&MFC42,#2514	
004899F4 . C66424 78870000 0D CALL K JMP,&MFC42,#2514	004899FA . E8 81300300 LEA ECX,DWORD PTR SS:[ESP+210]	
004899F5 . 80EC24 78870000 0D CALL K JMP,&MFC42,#2514	004899FB . C66424 78870000 0C LEA ECX,DWORD PTR SS:[ESP+200]	
004899F6 . C66424 78870000 0C CALL K JMP,&MFC42,#2514	004899FC . E8 DB2B0300 LEA ECX,DWORD PTR SS:[ESP+210]	
004899F7 . C66424 78870000 05 CALL K JMP,&MFC42,#2514	004899FD . 80EC24 B0020000 MOU BYTE PTR SS:[ESP+200]	
004899F8 . 80EC24 B0020000 MOU BYTE PTR SS:[ESP+200]	004899FE . C66424 78870000 05 CALL K JMP,&MFC42,#2514	
004899F9 . C66424 78870000 05 CALL K JMP,&MFC42,#2514	004899FF . E8 48300300 LEA ECX,DWORD PTR SS:[ESP+210]	
004899FA . E8 48300300 CALL K JMP,&MFC42,#2514	004899FB . 80EC24 B0020000 MOU BYTE PTR SS:[ESP+200]	
004899FB . 80EC24 B0020000 MOU BYTE PTR SS:[ESP+200]	004899FC . C66424 78870000 0E CALL K JMP,&MFC42,#2514	
004899FD . C66424 78870000 0E CALL K JMP,&MFC42,#2514	004899FD . E8 A22B0300 LEA ECX,DWORD PTR SS:[ESP+210]	
004899FE . E8 A22B0300 CALL K JMP,&MFC42,#2514	004899FB . C66424 78870000 05 CALL K JMP,&MFC42,#2514	
004899FF . C66424 78870000 05 CALL K JMP,&MFC42,#2514	004899FB . E8 CD000000 MOU ECX,EDI	
004899FB . E8 CD000000 CALL VisualSi.00488F30	004899FB . 803F 00000000 MOU ECX,EDI	
004899FB . 803F 00000000 CALL VisualSi.00488F30	004899FB . 83F8 00000000 MOU ECX,EDI	
004899FB . 83F8 00000000 CALL VisualSi.00488F30	004899FB . 74 55 JNZ VisualSi.00488F30	
004899FB . 74 55 JNZ VisualSi.00488F30	004899FB . 80EC24 6CF5FFFF MOU BYTE PTR SS:[ESP+8778],5	
004899FB . 80EC24 6CF5FFFF MOU BYTE PTR SS:[ESP+8778],5	004899FB . C66424 AC010000 LEA ECX,DWORD PTR SS:[ESP+1AC]	
004899FB . C66424 AC010000 LEA ECX,DWORD PTR SS:[ESP+1AC]	004899FB . F8 40000000 CALL K JMP,&MFC42,#641	

Trial period expired



CoffeeCup Software
VisualSite Designer

Your trial-period has expired. You must purchase the full version of the program in order to continue using it. Please click the "Purchase" button to visit our secure online shop.

Purchase Quit

See what happens if a trial limit has been reached.

See what happens if a trial limit has been reached.

Trial 이 제한이 됐다면 무엇이 발생하는지 봐.

C CPU - main thread, module VisualSi

```

0048980F  .: 84C0          TEST AL,AL
00489811  .: C0F8 42010000 JNC VisualSi.00489A29
00489812  .: EB87 E4000000 MOV ECX,DWORD PTR DS:[EDI+E4]
00489813  .: 6A 00          PUSH 0
00489814  .: 85C0          TEST ECX,ECX
00489815  .: 74 40          JE SHORT VisualSi.0048995C
00489912  .: 004BC42A        CALL K JMP.&MFC42.#2514>
00489917  .: 88F8 01          CMP ECX,1
0048991C  .: 8BCF          MOV ECX,EDI
0048991E  .: EB 00F6FFFF    CALL VisualSi.00488F30
00489923  .: 8D8C24 B0020000 LEA ECX,DWORD PTR SS:[ESP+280]
00489924  .: C68424 78870000 0D MOU BVTE PTR SS:[ESP+8778],0D
00489925  .: EB 6D240200    LEA ECX,DWORD PTR SS:[ESP+200]
00489926  .: 8D8C24 0C020000 MOU BVTE PTR SS:[ESP+8778],0B
00489927  .: C68424 78870000 0B MOU BVTE PTR SS:[ESP+8778],0B
00489928  .: EB 132B0300    CALL K JMP.&MFC42.#2514>
00489929  .: 88F8 01          CMP ECX,1
0048992A  .: 8BCF          MOV ECX,EDI
0048992B  .: EB 00F6FFFF    CALL VisualSi.00488F30
0048992C  .: 8D8C24 B0020000 LEA ECX,DWORD PTR SS:[ESP+280]
0048992D  .: C68424 78870000 0D MOU BVTE PTR SS:[ESP+8778],0D
0048992E  .: EB 81300300    LEA ECX,DWORD PTR SS:[ESP+765]
0048992F  .: 8D8C24 70020000 MOU BVTE PTR SS:[ESP+2780]
00489930  .: C68424 78870000 0C MOU BVTE PTR SS:[ESP+8778],0C
00489931  .: EB DB2B0300    CALL K JMP.&MFC42.#795>
00489932  .: 8D8C24 B0020000 MOU BVTE PTR SS:[ESP+8778],05
00489933  .: C68424 78870000 05 LEA ECX,DWORD PTR SS:[ESP+280]
00489934  .: EB 77            JMP SHORT VisualSi.004899D3
00489935  .: 8D8C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+280]
00489936  .: C68424 78870000 0F MOU BVTE PTR SS:[ESP+8778],0F
00489937  .: EB 48300300    CALL K JMP.&MFC42.#765>
00489938  .: 8D8C24 70020000 LEA ECX,DWORD PTR SS:[ESP+2780]
00489939  .: C68424 78870000 0E MOU BVTE PTR SS:[ESP+8778],0E
0048993A  .: EB A22B0300    CALL K JMP.&MFC42.#795>
0048993B  .: 8D8C24 78870000 05 MOU BVTE PTR SS:[ESP+8778],05
0048993C  .: C68424 0C020000 LEA ECX,DWORD PTR SS:[ESP+280]
0048993D  .: EB SC000000    JMP VisualSi.00489924
0048993E  .: 8D8C24 B0010000 LEN ECX,DWORD PTR SS:[ESP+180]
0048993F  .: C68424 AC010000 CALL VisualSi.004AA4E70
00489940  .: EB CC642100    LEA ECX,DWORD PTR SS:[ESP+180]
00489941  .: 8D8C24 78870000 10 MOU BVTE PTR SS:[ESP+8778],10
00489942  .: C68424 78870000 10 CALL K JMP.&MFC42.#2514>
00489943  .: EB 22403000    CMP ECX,1
00489944  .: 88F8 01          CALL SHORT VisualSi.00489A15
00489945  .: 8BCF          MOV ECX,EDI
00489946  .: EB 6CF6FFFF    
```

This nag was generated
in this call huh. Breakpointing it
is also a good visual aid.

This nag was generated in this call huh. Breakpointing it is also a good visual aid.

이 nag 은 이 call 에서 생성됐다. 또한 Breakpoint 은 좋다.

Now, we only need to study the code further as we partially did already. You can do it for yourself and detect that there are different possibilities (like stated already).

이제, 우리가 이미 부분적으로 했을 때 우리는 code 를 멀리 공부하는 게 필요하다. 자신을 위해 할 수 있다. 그리고 다른 가능성은 발견했다.(이미 정해진 것처럼)

Running

```

C CPU - main thread, module VisualSI
0048980F . 84C0 TEST AL,AL
004898E1 . 0F85 42010000 JNZ VisualSI.00489A29
004898E7 . 88E7 E4000000 MOU EAX, DWORD PTR DS:[EDI+E4]
004898ED . 6A 00 PUSH 0
004898EF . 85C0 TEST EAX,EAX
004898F1 . 8F8E A1000000 JLE VisualSI.00489998
004898F7 . 80EC24 10020000 LEA ECX, DWORD PTR SS:[ESP+210]
004898FE . | E8 60240200 CALL VisualSI.0048B070
00489903 . | 80EC24 0C020000 LEA ECX, DWORD PTR SS:[ESP+200]
00489909 . | C68424 78870000 0B MOU BYTE PTR SS:[ESP+8778],0B
00489912 . | E8 132B0300 CALL KJMP.&MFC42.#2514
00489917 . | 85F8 01 CMP EAX,1
00489919 . | 74 40 JE SHORT VisualSI.0048995C
0048991C . | 88CF MOU ECX,EDI
00489923 . | E8 00F6FFFF CALL VisualSI.00488F30
00489927 . | 80EC24 80020000 LEA ECX, DWORD PTR SS:[ESP+280]
00489930 . | C68424 78870000 0D MOU BYTE PTR SS:[ESP+8778],0D
00489932 . | E8 E1300300 CALL KJMP.&MFC42.#765
00489937 . | 80EC24 70020000 LEA ECX, DWORD PTR SS:[ESP+270]
0048993E . | C68424 78870000 0C MOU BYTE PTR SS:[ESP+8778],0C
00489940 . | E8 A22B0300 CALL KJMP.&MFC42.#795
00489944 . | C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],5
00489948 . | 80EC24 0C020000 LEA ECX, DWORD PTR SS:[ESP+200]
00489950 . | E8 8C000000 JNP SHORT VisualSI.00489908
0048995C . | 80EC24 80020000 LEA ECX, DWORD PTR SS:[ESP+280]
00489960 . | C68424 78870000 0F MOU BYTE PTR SS:[ESP+8778],0F
00489964 . | E8 43000000 CALL KJMP.&MFC42.#6419
00489968 . | 80EC24 70020000 LEA ECX, DWORD PTR SS:[ESP+270]
00489970 . | C68424 78870000 0E MOU BYTE PTR SS:[ESP+8778],0E
00489974 . | E8 A22B0300 CALL KJMP.&MFC42.#795
00489977 . | C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],5
00489981 . | E8 8C000000 LEA ECX, DWORD PTR SS:[ESP+200]
00489984 . | 80EC24 B0010000 JNP VisualSI.00489A24
00489988 . | E8 CCE4B100 LEA ECX, DWORD PTR SS:[ESP+1AC0]
0048998B . | 80EC24 AC010000 MOU BYTE PTR SS:[ESP+8778],18
0048998C . | C68424 78870000 10 CALL KJMP.&MFC42.#2514
0048998D . | E8 722A0300 CMP EAX,1
0048998E . | 85F8 01 JE SHORT VisualSI.00489A15
0048998F . | 74 58 MOU ECX,EDI
00489990 . | 88CF CALL VisualSI.00488F30
00489994 . | E8 60C5FFFF MOU BYTE PTR SS:[ESP+8778],5
00489997 . | 80EC24 80020000 LEA ECX, DWORD PTR SS:[ESP+280]
00489999 . | C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],0F
0048999C . | E8 43000000 CALL KJMP.&MFC42.#6419
0048999E . | 80EC24 70020000 LEA ECX, DWORD PTR SS:[ESP+270]
004899A0 . | C68424 78870000 0E MOU BYTE PTR SS:[ESP+8778],0E
004899A2 . | E8 A22B0300 CALL KJMP.&MFC42.#795
004899A5 . | C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],5
004899A8 . | E8 8C000000 LEA ECX, DWORD PTR SS:[ESP+200]
004899AB . | 80EC24 B0010000 JNP VisualSI.00489A24
004899B1 . | E8 CCE4B100 LEA ECX, DWORD PTR SS:[ESP+1AC0]
004899B4 . | 80EC24 AC010000 MOU BYTE PTR SS:[ESP+8778],18
004899B7 . | C68424 78870000 10 CALL KJMP.&MFC42.#2514
004899B8 . | E8 722A0300 CMP EAX,1
004899B9 . | 85F8 01 JE SHORT VisualSI.00489A15
004899BD . | 74 58 MOU ECX,EDI
004899C0 . | 88CF CALL VisualSI.00488F30
004899C4 . | E8 60C5FFFF MOU BYTE PTR SS:[ESP+8778],5
004899C7 . | 80EC24 80020000 LEA ECX, DWORD PTR SS:[ESP+280]
004899C9 . | C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],0F
004899CC . | E8 43000000 CALL KJMP.&MFC42.#6419
004899D1 . | 80EC24 70020000 LEA ECX, DWORD PTR SS:[ESP+270]
004899D3 . | C68424 78870000 0E MOU BYTE PTR SS:[ESP+8778],0E
004899D5 . | E8 A22B0300 CALL KJMP.&MFC42.#795
004899D8 . | C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],5
004899D9 . | E8 8C000000 LEA ECX, DWORD PTR SS:[ESP+200]
004899D2 . | 80EC24 B0010000 JNP VisualSI.00489A24
004899D5 . | E8 CCE4B100 LEA ECX, DWORD PTR SS:[ESP+1AC0]
004899D8 . | 80EC24 AC010000 MOU BYTE PTR SS:[ESP+8778],18
004899D9 . | C68424 78870000 10 CALL KJMP.&MFC42.#2514

```

The cond jump, just above this screen at 004898D8 (not visible here, see before) leads to exiting the app : no need to touch that. Which leaves us with two other possibilities.

조건 jump 는, 이 screen 의 004898D8 에서(이곳에서는 보여지지 않는다, 나중에 보자) app 종료로 이끈다. 더 이상 제어가 필요하지 않다. 우리는 2 가지 가능성과 함께 떠난다.

```

C CPU - main thread, module VisualSI
0048980F . 84C0 TEST AL,AL
004898E1 . 0F85 42010000 JNZ VisualSI.00489A29
004898E7 . 88E7 E4000000 MOU EAX, DWORD PTR DS:[EDI+E4]
004898ED . 6A 00 PUSH 0
004898EF . 85C0 TEST EAX,EAX
004898F1 . 8F8E A1000000 JLE VisualSI.00489998
004898F7 . 80EC24 10020000 LEA ECX, DWORD PTR SS:[ESP+210]
004898FE . | E8 60240200 CALL VisualSI.0048B070
00489903 . | 80EC24 0C020000 LEA ECX, DWORD PTR SS:[ESP+200]
00489909 . | C68424 78870000 0B MOU BYTE PTR SS:[ESP+8778],0B
00489912 . | E8 132B0300 CALL KJMP.&MFC42.#2514
00489917 . | 85F8 01 CMP EAX,1
00489919 . | 74 40 JE SHORT VisualSI.0048995C
0048991C . | 88CF MOU ECX,EDI
00489923 . | E8 00F6FFFF CALL VisualSI.00488F30
00489927 . | 80EC24 80020000 LEA ECX, DWORD PTR SS:[ESP+280]
00489930 . | C68424 78870000 0E MOU BYTE PTR SS:[ESP+8778],0E
00489932 . | E8 A22B0300 CALL KJMP.&MFC42.#795
00489937 . | C68424 78870000 0C MOU BYTE PTR SS:[ESP+8778],0C
00489940 . | E8 77 000000 LEA ECX, DWORD PTR SS:[ESP+270]
00489944 . | 80EC24 80020000 MOU BYTE PTR SS:[ESP+8778],0C
00489948 . | C68424 78870000 05 CALL KJMP.&MFC42.#765
00489950 . | 80EC24 80020000 MOU BYTE PTR SS:[ESP+280]
00489953 . | C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],5
00489955 . | E8 77 000000 LEA ECX, DWORD PTR SS:[ESP+270]
0048995C . | 80EC24 B0010000 JNP SHORT VisualSI.00489908
00489963 . | E8 CCE4B100 LEA ECX, DWORD PTR SS:[ESP+1AC0]
00489966 . | 80EC24 AC010000 MOU BYTE PTR SS:[ESP+8778],18
00489970 . | C68424 78870000 0E CALL KJMP.&MFC42.#6419
00489974 . | 80EC24 70020000 LEA ECX, DWORD PTR SS:[ESP+270]
00489977 . | C68424 78870000 0F MOU BYTE PTR SS:[ESP+8778],0E
00489978 . | E8 A22B0300 CALL KJMP.&MFC42.#795
00489981 . | C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],5
00489984 . | E8 8C000000 LEA ECX, DWORD PTR SS:[ESP+200]
00489988 . | 80EC24 B0010000 JNP VisualSI.00489A24
0048998B . | E8 CCE4B100 LEA ECX, DWORD PTR SS:[ESP+1AC0]
0048998C . | 80EC24 AC010000 MOU BYTE PTR SS:[ESP+8778],18
0048998D . | C68424 78870000 10 CALL KJMP.&MFC42.#2514
0048998E . | E8 722A0300 CMP EAX,1
0048998F . | 85F8 01 JE SHORT VisualSI.00489A15
00489990 . | 74 58 MOU ECX,EDI
00489991 . | 88CF CALL VisualSI.00488F30
00489992 . | E8 60C5FFFF MOU BYTE PTR SS:[ESP+8778],5
00489993 . | 80EC24 80020000 LEA ECX, DWORD PTR SS:[ESP+280]
00489994 . | C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],0F
00489995 . | E8 43000000 CALL KJMP.&MFC42.#6419
00489996 . | 80EC24 70020000 LEA ECX, DWORD PTR SS:[ESP+270]
00489997 . | C68424 78870000 0E MOU BYTE PTR SS:[ESP+8778],0E
00489998 . | E8 A22B0300 CALL KJMP.&MFC42.#795
00489999 . | C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],5
0048999A . | E8 8C000000 LEA ECX, DWORD PTR SS:[ESP+200]
0048999B . | 80EC24 B0010000 JNP VisualSI.00489A24
0048999C . | E8 CCE4B100 LEA ECX, DWORD PTR SS:[ESP+1AC0]
0048999D . | 80EC24 AC010000 MOU BYTE PTR SS:[ESP+8778],18
0048999E . | C68424 78870000 10 CALL KJMP.&MFC42.#2514

```

Right. If we would take this jump, it jumps past both nags.

맞아. 우리가 jump 를 한다면, 2 가지 nag 을 jump 할 수 있다.

Also, if executed, doesn't it jump to exit the program ???

또한, 실행된다면, program 이 종료되는 jump 가 필요하지 않다.

This will prove to be the easiest patch.

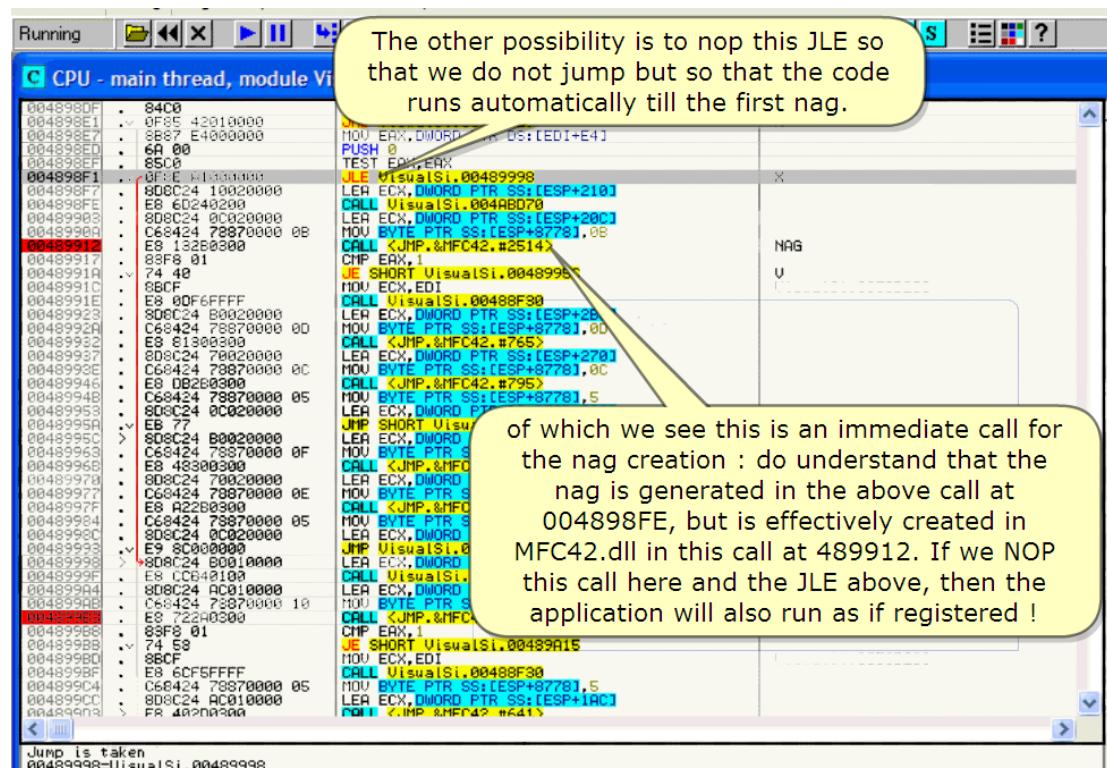
쉬운 patch 인지 증명할 수 있다.

I'll come back to this solution later.

나는 나중에 다른 해결책을 가지고 돌아올 것이다.

But because we are here to learn, let's also study the other possibilities

왜냐하면 우리는 여기에서 배워야 한다. 또한 다른 가능성을 배우자.



The other possibility is to NOP this JLE so that we do not jump but so that the code runs automatically till the first nag.

이 JLE 에서 jump 하지 않도록 다른 가능성이 있는 것은 NOP 다. code 가 자동적으로 첫번째 nag 까지 실행된다.

Of which we see this an immediate call for the nag creation : do understand that the nag is generated in the above call at 004898FE, but is effectively created in MFC42.dll in this call at 489912.

즉시 nag 만드는 call 을 본다. : 이 call 의 004898FE 위에서 nag 이 생성되는 곳을 이해할 수 있다.
효과적으로 MFC42.dll 안의 489912 에서 생성되었다.

If we NOP this call here and the JLE above, then the application will also run as if registered !

우리는 이 call 에서 아무 동작하지 않고 JLE 가 위에 있으면, application 은 등록된 것처럼 실행된다.

Of course there are always other possibilities.

물론 그것은 항상 다른 가능성들이 있다.

For example : assembling this line from the call as MOV EAX, 1 instead of NOP'ing it ...

예 : call 로부터 MOV EAX, 1 일 때 대신해서 이 line 을 assembling 해서 NOP 해야 된다.

Registers the app and jumps straight to the main program

App 을 등록하고 main program 으로 직접 jump 하자.

However, in the case, each other start up unregisters the app because of a doublecheck

With another annoying nag as result.

Then you also need to make the following changes to also jump over that nag :

그러나, 이번 경우는 각각 다른 start up app 이 미등록 되었다. 왜냐하면 다른 nag 을 회피할 때 2 번 check 한다.

너는 또 nag 을 jump 하기 위해 다음 변경이 필요하다.

RVA	488C83	MOV AL,1
-----	--------	----------

Or

RVA	488C85	JMP 488D10
-----	--------	------------

(or other possibilities to jump past that nag)

(과거의 nag 을 jump 하기 위해 다른 가능성)

Try it out if you want to learn !!!

네가 배우기를 원한다면 도전 해봐.

Another possibility is not noping this JLE but jump always

but then change this Call to the expired nag
(remark that it is the same call as above)

As you can see, plenty of possibilities for you to try out (they all are working solutions)

and assemble this call as

MOV EAX,1
etc etc etc

Another possibility is not noping this JLE but jump always

다른 가능성은 NOP 을 하지 않는다. 이 JLE 에서는 항상 jump 가

But then change this Call to the expired nag (remark that is the same call as above)

그러나 종료되는 nag에서 이 call을 바꿔야 한다.(주목. 이것은 위와 같은 call이다.)

And assemble this call as

MOV EAX, 1

etc etc etc

As you can see, plenty of possibilities for you to try out (they all are working solutions)

그리고 이 call이 MOV EAX,1 일 때 assemble.

네가 볼 때, 도전하기 위한 많은 가능성들이 너에게 있다.(그들은 모두 훌륭한 해결책이다.)

Indeed, let's come back to our first possibility to jump past both nags at once

정말, 과거의 2 개의 nag 을 한 번에 jump 하기 위해 첫번째 가능성으로 돌아와.

If you tried it out, you will have found that it turns out as the easiest solution because it's only one patch.

만약에 네가 도전한다면, 너는 돌아오는 것을 찾을 것이다. 매우 쉬운 방법이다. 왜냐하면 오직 1 번 patch 하면 된다.

I tried it too, but it is something you understand by now and so I removed it from this movie to reduce the movie's size.

나는 도전했다, 그러나 그것은 무엇이다. 지금 네가 이해했고 나는 그것을 Movie size 때문에
사제했다

BTW, it causes no other nags either
By the way, 그걸로 다른 nag 이 있음을 소개하지 않아도

By the way, 그짓은 다른 허가를 필요로 한다.

C CPU - main thread, module VisualSi

```

004898DF . 84C0 TEST AL,AL
004898E1 . 0F85 42010000 JN2 VisualSi.00489A29
004898E7 . 8887 E4000000 MOU EAX,DWORD PTR DS:[EDI+E4]
004898ED . 6A 00 PUSH 0
004898EF . 85C0 TEST EAX,EAX
004898F1 . 0F8E A1000000 JLE VisualSi.00489998
004898F7 . 8D8C24 10020000 LEA ECX,DWORD PTR SS:[ESP+210]
004898FE . E8 6D240200 CALL VisualSi.00489D70
00489903 . 8D8C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+20C]
0048990A . C68424 78870000 0B MOU BYTE PTR SS:[ESP+87781,0B]
00489912 . E8 13280300 CALL KJMP.&MFC42.#25
00489917 . 83F8 01 CMP EAX,1
0048991A . 74 40 JE SHORT VisualSi.00489950
0048991C . 8BCF MOU ECX,EDI
0048991E . E8 0DF6FFFF CALL VisualSi.00488980
00489923 . 8D8C24 B0020000 LEA ECX,DWORD PTR SS:[ESP+200]
0048992A . C68424 78870000 0D MOU ECX,EDI
00489932 . E8 81300300 CALL VisualSi.00489980
00489937 . 8D8C24 70020000 LEA ECX,DWORD PTR SS:[ESP+200]
0048993E . C68424 78870000 0C MOU ECX,EDI
00489946 . E8 DB280300 CALL VisualSi.00489980
00489948 . C68424 78870000 05 MOU ECX,EDI
00489953 . 8D8C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+20C]

```

Conclusion :

TEST AL, AL means :

"Is the soft registered ?"

Scroll down.

Conclusion :

TEST AL, AL means :

"Is the soft registered ?"

Scroll down.

결론 :

TEST AL, AL 뜻 :

"어떻게 soft 등록을 해?"

스크롤 내려봐.

C CPU - main thread, module VisualSi

```

004898B9 . 896F 20 MOU DWORD PTR DS:[EDI+20],EBP
004898BC . E8 4FEEFFFF CALL VisualSi.00488710
004898C1 . 6A 00 PUSH 0
004898C3 . 8BCF MOU ECX,EDI
004898C5 . E8 59360300 CALL KJMP.&MFC42.#5583
004898C9 . 8BCF MOU ECX,EDI
004898CC . E8 AFFF0FFF CALL VisualSi.00488980
004898D1 . 84C0 TEST AL,AL
004898D3 . 0F84 FF000000 JE VisualSi.004899D8
004898D9 . 8A87 E0000000 MOU AL,BYTE PTR DS:[EDI+E0]
004898D0 . 84C0 TEST AL,AL
004898E1 . 0F85 42010000 JN2 VisualSi.00489A29
004898ED . 8887 E4000000 MOU EAX,DWORD PTR DS:[EDI+E4]
004898F1 . 6A 00 PUSH 0
004898F7 . 85C0 TEST EAX,EAX
004898FE . 0F8E A1000000 JLE VisualSi.00489980
00489903 . 8D8C24 10020000 LEA ECX,DWORD PTR SS:[ESP+210]
00489923 . C68424 78870000 0B MOU BYTE PTR SS:[ESP+87781,0B]
0048992A . 8D8C24 78870000 0D CALL KJMP.&MFC42.#765
00489932 . E8 81300300 LEA ECX,DWORD PTR SS:[ESP+200]
00489937 . 8D8C24 70020000 MOU BYTE PTR SS:[ESP+87781,0C]
0048993E . C68424 78870000 0C CALL KJMP.&MFC42.#795
00489946 . E8 DB280300 MOU BYTE PTR SS:[ESP+87781,15]
00489948 . C68424 78870000 05 LEA ECX,DWORD PTR SS:[ESP+20C]
00489953 . 8D8C24 0C020000 CALL KJMP.&MFC42.#795
00489956 . EB 77 JMP SHORT VisualSi.004899D8
0048995C . > 8D8C24 B0020000 LEA ECX,DWORD PTR SS:[ESP+200]
00489963 . C68424 78870000 0F MOU BYTE PTR SS:[ESP+87781,0F]
00489968 . E8 48300300 CALL KJMP.&MFC42.#765
00489970 . 8D8C24 70020000 LEA ECX,DWORD PTR SS:[ESP+270]
00489977 . C68424 78870000 0E MOU BYTE PTR SS:[ESP+87781,0E]
0048997F . E8 A2280300 CALL KJMP.&MFC42.#795
00489984 . C68424 78870000 05 MOU BYTE PTR SS:[ESP+87781,5]
0048998C . 8D8C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+20C]
00489993 . E8 8C000000 CALL KJMP.&MFC42.#795
00489998 . > 8D8C24 B0010000 MOU VisualSi.00489A24
00489999 . FR C0R4100 LEA ECX,DWORD PTR SS:[ESP+1B0]
0048999A . RETN VisualSi.00404F20

```

This is the cond jump that leads to exiting the app.

This is the cond jump that leads to exiting the app.

이 조건 jump 는 app 을 종료 시킨다.

C CPU - main thread, module VisualSi

```

004898B9 . 896F 20 MOU DWORD PTR DS:[EDI+20],EBP
004898BC . E8 4FEEFFFF CALL VisualSi.00488710
004898C1 . 6A 00 PUSH 0
004898C3 . 8BCF MOV ECX,EDI
004898C5 . E8 58360300 CALL <JMP.&MFC42.#5503>
004898C9 8BCF MOV ECX,EDI
004898D1 . E8 AFFF0FFF CALL VisualSi.00488980
004898D1 . 84C0 TEST AL,AL
004898D8 . 0F84 FF000000 JE VisualSi.00489908 X
004898D9 . 8A87 E0000000 MOV AL,BYTE PTR DS:[EDI+E0]
004898D9 . 84C0 TEST AL,AL
004898E1 . 0F85 42010000 JNZ VisualSi.00489A29 X
004898E1 . 8A87 E4000000 MOV EAX,DWORD PTR DS:[EDI+E4]
004898ED . 84C0 TEST AL,AL
004898F1 . 0F85 A1000000 JNE VisualSi.00489998 X
004898F7 . 8D8C24 10020000 LEA ECX,DWORD PTR SS:[ESP+210]
004898FE . E8 6D240200 CALL VisualSi.004AB070
00489903 . 8D8C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+20C]
00489903 . C68424 78870000 0B MOU BYTE PTR SS:[ESP+87781],0B
00489912 . E8 132B0300 CALL VisualSi.00489950 NAG
00489912 . 83F8 01 CMP EAX,1
00489912 . 74 40 JE SHORT VisualSi.00489950 U
00489912 . 8BCF MOU ECX,ESI
0048991E . E8 0DF6FFFF
00489923 . 8D8C24 B0000000
00489929 . E8 813003
00489937 . 8D8C24 78870000
00489937 . E8 D62B0300
00489946 . 8D8C24 78870000
00489946 . E8 0B280300
00489948 . 8D8C24 78870000 05
00489953 . 8D8C24 0C020000
00489953 . EB 77
00489953 . 8D8C24 B0020000
00489953 . E8 48300300
00489963 . 8D8C24 78870000 0F
00489970 . 8D8C24 70020000
00489977 . C68424 78870000 0E
00489977 . E8 A22B0300
00489984 . C68424 78870000 05
00489984 . 8D8C24 0C020000
00489993 . E8 80000000
00489998 . 8D8C24 B0010000
00489998 . FR CCR401000
00489998 . F0I VisualSi.00404F20

```

Jump is taken
004899D8=VisualSi.004899D8

We decided to patch 004898E1 to jump always. This will be plain stupid patching.

I will come back to this in next tutorials.

우리는 004898E1 을 항상 jump 하기로 결정했다. 이것은 분명히 멍청한 patch 다.

나는 다음 tutorial 에 이것을 가지고 돌아오겠다.

C CPU - main thread, module VisualSi

```

004898B9 . 896F 20 MOU DWORD PTR DS:[EDI+20],EBP
004898BC . E8 4FEEFFFF CALL VisualSi.00488710
004898C1 . 6A 00 PUSH 0
004898C3 . 8BCF MOV ECX,EDI
004898C5 . E8 58360300 CALL <JMP.&MFC42.#5503>
004898C9 8BCF MOV ECX,EDI
004898D1 . E8 AFFF0FFF CALL VisualSi.00488980
004898D1 . 84C0 TEST AL,AL
004898D8 . 0F84 FF000000 JE VisualSi.00489908 X
004898D9 . 8A87 E0000000 MOU AL,BYTE PTR DS:[EDI+E0]
004898D9 . 84C0 TEST AL,AL
004898E1 . 0F85 42010000 JNZ VisualSi.00489A29 X
004898E1 . 8A87 E4000000 MOU EAX,DWORD PTR DS:[EDI+E4]
004898ED . 84C0 TEST AL,AL
004898F1 . 0F85 A1000000 JNE VisualSi.00489998 X
004898F7 . 8D8C24 10020000 LEA ECX,DWORD PTR SS:[ESP+210]
004898FE . E8 6D240200 CALL VisualSi.004AB070
00489903 . 8D8C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+20C]
00489903 . C68424 78870000 0B MOU BYTE PTR SS:[ESP+87781],0B
00489912 . E8 132B0300 CALL VisualSi.00489950 NAG
00489912 . 83F8 01 CMP EAX,1
00489912 . 74 40 JE SHORT VisualSi.00489950 U
00489912 . 8BCF MOU ECX,ESI
0048991E . E8 0DF6FFFF
00489923 . 8D8C24 B0000000
00489929 . E8 813003
00489937 . 8D8C24 78870000
00489937 . E8 D62B0300
00489946 . 8D8C24 78870000 05
00489953 . 8D8C24 0C020000
00489953 . EB 77
00489953 . 8D8C24 B0020000
00489953 . E8 48300300
00489963 . 8D8C24 78870000 0F
00489970 . 8D8C24 70020000
00489977 . C68424 78870000 0E
00489977 . E8 A22B0300
00489984 . C68424 78870000 05
00489984 . 8D8C24 0C020000
00489993 . E8 80000000
00489998 . 8D8C24 B0010000
00489998 . FR CCR401000
00489998 . F0I VisualSi.00404F20

```

Jump is taken
004899D8=VisualSi.004899D8

For now, I'll just state that indeed, we could dig deeper and find out where EDI+E0 is set

Do you understand that the value for EDI+E0 is where the decision on being registered or not is stored ??? Try to understand this already but it's no problem if you don't understand here, as said, I'll come back to such situations in later parts

For now, I'll just state that indeed, we could dig deeper and find out where EDI+E0 is set

지금은, 명시했다. 우리가 깊게 들어가고 EDI+E0 이 set 된 것을 찾아.

Do you understand that the value for EDI+E0 is where the decision on being registered or not is stored ???

이해했어. EDI+E0 값이 등록 됐는지 아닌지 결정한다.

Try to understand this already but it's no problem if you don't understand here, as said, I'll come back to such situations in later parts

이해하기 위해 노력해봐. 네가 여기에서 이해하지 못해도, 문제 없어. 내가 "나는 나중 part에서 다른 해결 방법으로 돌아올 거야." 라고 말했지.

C CPU - main thread, module VisualSi

```
004898B9 . 896F 20 MOU DWORD PTR DS:[EDI+20],EBP
004898BC . E8 4FFFFFFF CALL VisualSi.00488710
004898C1 . 6A 00 PUSH 0
004898C3 . 8BCF MOU ECX,EDI
004898C5 . E8 58360300 CALL <JMP.&FC42.#5503>
004898C9 . 8BCF MOU ECX,EDI
004898CC . E8 AFFF0FFF CALL VisualSi.00488980
004898D1 . 84C0 TEST AL,AL
004898D2 . 0F84 FF000000 JE VisualSi.004899D8
004898D9 . 84C0 E0000000 MOV AL,BYTE PTR DS:[EDI+E0]
004898DF . 0F84 E4000000 TEST AL,AL
004898E1 . 004898E1 UNL VisualSi.00489A29
004898E7 . 8BC0 EDX,DWORD PTR DS:[EDI+F4]
004898ED . 6A 00
004898EF . 85C0
004898F1 . 0F84 A1000000
004898F7 . 8D8C24 10020000
004898FE . E8 6D248200
00489903 . 8D8C24 0C020000
00489909 . C68424 78870000 0B
00489912 . E8 132B0300
00489917 . 83F8 01
0048991A . 74 40
0048991C . 8BCF
0048991E . E8 0DF6FFFF
00489923 . 8D8C24 B0020000
00489924 . C68424 78870000 0D
00489932 . E8 81300300
00489937 . 8D8C24 70020000
0048993E . C68424 78870000 0C
00489946 . E8 DB2B0300
00489948 . C68424 78870000 05
00489953 . 8D8C24 B0020000
00489955 . EB 77
0048995C . 8D8C24 B0020000
00489963 . C68424 78870000 0F
00489968 . E8 4B300300
00489970 . 8D8C24 70020000
00489977 . C68424 78870000 0E
0048997F . E8 A22B0300
00489984 . C68424 78870000 05
0048998C . 8D8C24 B0020000
00489993 . E8 8C000000
00489998 . 8D8C24 B0010000
0048999E . F8 C7C40100
<...>
```

Jump is taken
004899D8=VisualSi.004899D8

To quickly show you the results,
I first removed this BP (where we
exit app) and re-ran the soft till ...

To quickly show you the results, I first removed this BP (where we exit app) and re-ran the soft till

...

빠르게 결과를 보여줄께, 먼저 첫번째 BP를 삭제해(이것은 app을 종료한다.) 그리고 다시 여기까지 시작해.

...breaking here again.(Removed it for size)

여기에 멈춰(size 때문에 movie에서 삭제했다)

C CPU - main thread, module VisualSi

```

004898E1 . 0F85 42010000 JNZ VisualSi.00489A29
004898E0 . 6A 00 MOU EAX,DWORD PTR DS:[EDI+E4]
004898E5 . 85C0 PUSH 0
004898E9 . 0F8E A10000 TEST EAX,EAX
004898F1 . 8D8C24 100200 JLE VisualSi.00489998
004898F7 . 6A 00
00489900 . 85C0
00489903 . 0F8E A10000
00489906 . 8D8C24 100200
00489909 . 6A 00
00489912 . 85C0
00489917 . 0F8E A10000
0048991A . 8D8C24 100200
0048991C . 6A 00
0048991E . 85C0
00489923 . 0F8E A10000
00489926 . 8D8C24 B00200
00489929 . C68424 788700
00489932 . E8 81300300
00489935 . 8D8C24 700200
00489938 . C68424 788700
00489941 . E8 0B280300
00489945 . 8D8C24 788700
00489948 . 6A 00
00489953 . 8D8C24 B00200
00489956 . 6A 00
00489959 . > EB 77
00489962 . 8D8C24 B00200
00489965 . C68424 788700
00489968 . E8 48300300
00489970 . 8D8C24 700200
00489973 . C68424 78870000 0E MOU BYTE PTR SS:[ESP+8778],0E
0048997F . EB R22B0300 CALL JMP,&MFC42,#795
00489984 . C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],5
0048998C . 8D8C24 0C020000 LEA ECX, DWORD PTR SS:[ESP+28C]
00489993 . > E9 80000000 JMP VisualSi.00489A24
00489998 . 8D8C24 B0010000 LEA ECX, DWORD PTR SS:[ESP+1B0]
004899A1 . E8 CCB48100 CALL VisualSi.00444E70
004899A4 . 8D8C24 AC010000 LEA ECX, DWORD PTR SS:[ESP+1AC]
004899A8 . C68424 78870000 10 MOU BYTE PTR SS:[ESP+8778],10
004899B2 . EB 722A0300 CALL JMP,&MFC42,#2514
004899B3 . 83F8 01 CMP EAX,1
004899B5 . 74 58 JE SHORT VisualSi.00489A15
004899BD . 8BCF
004899BF . E8 6CF5FFFF
004899C4 . C68424 78870000 05 MOU BYTE PTR SS:[ESP+8778],5
004899CC . 8D8C24 AC010000 LEA ECX, DWORD PTR SS:[ESP+1AC]
004899D3 . > E8 482D0300 CALL JMP,&MFC42,#641
004899D9 . 8D4C24 20 IED FCX NIMON PTR SS:[ESP+20]

```

Remember that we decided that the program will be in registered status if we always jump here.

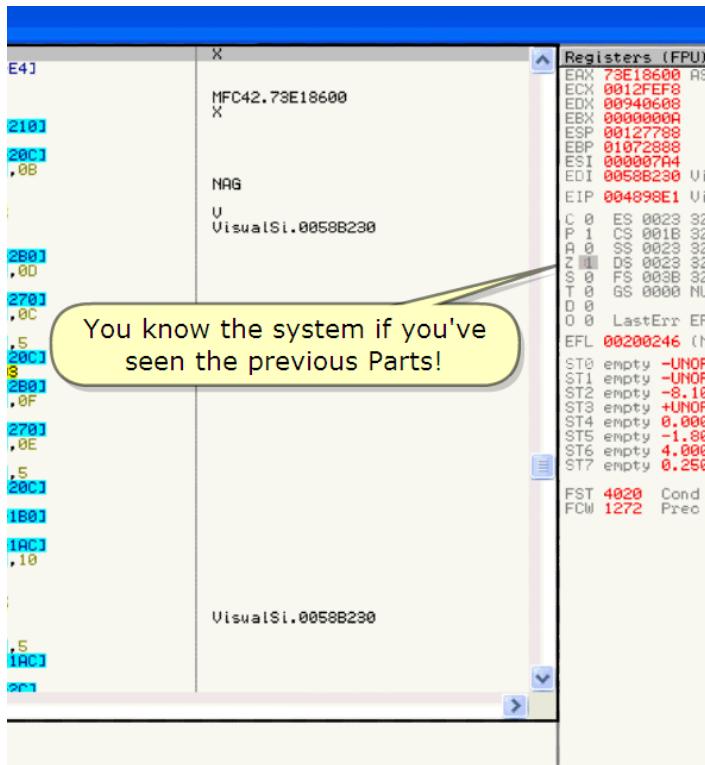
Let me first show you once more how this can be tested, but remember that if we want to make these changes permanent, that we will need to assemble this as JMP 00489A29

Remember that we decided that the program will be in registered status if we always jump here.

기억해. 우리가 항상 이곳을 jump 하면 Program 이 등록된 상태로 된다.

Let me first show you once more how this can be tested, but remember that if we want to make these changes permanent, that we will need to assemble this as JMP 00489A29

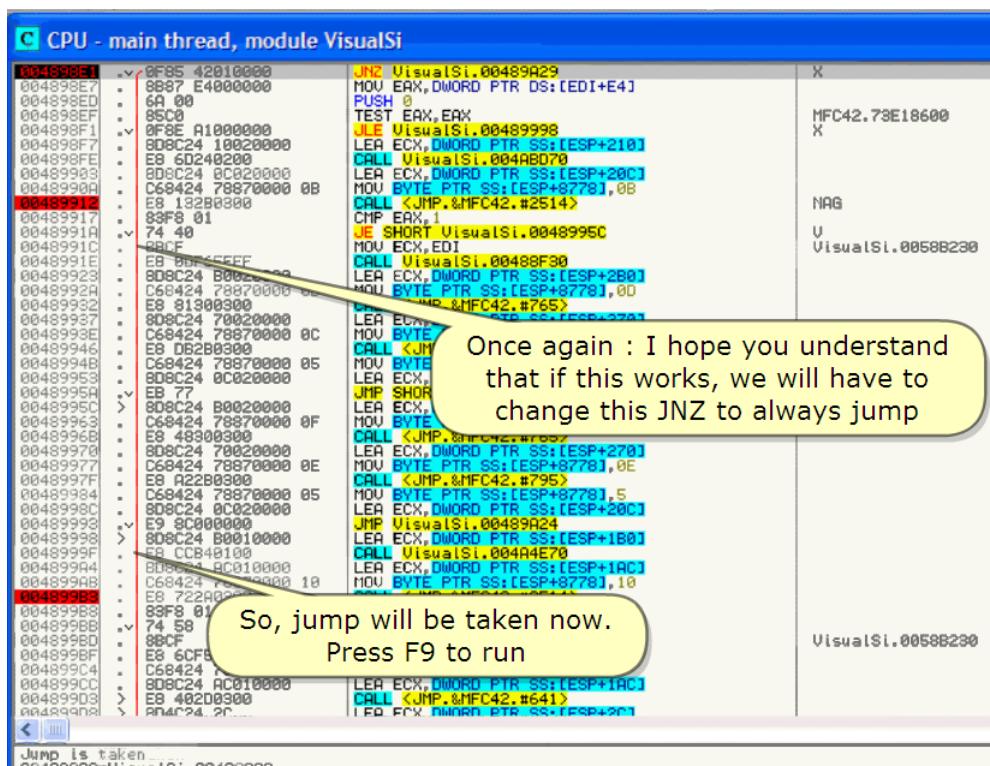
먼저 어떻게 test 하는지 보여줄께. 그러나 기억해. 영원히 바꿔야 돼. 우리는 JMP 00489A29 를 assemble 하는 게 필요하다.



You know the system if you've seen the previous Parts!

You know the system if you've seen the previous Parts!

네가 이전 강의를 봤다면 알고 있을 거야.



Once again : I hope you understand that if this works, we will have to change this JNZ to always jump

So, jump will be taken now.
Press F9 to run

Once again : I hope you understand that if this works, we will have to change this JNZ to always jump

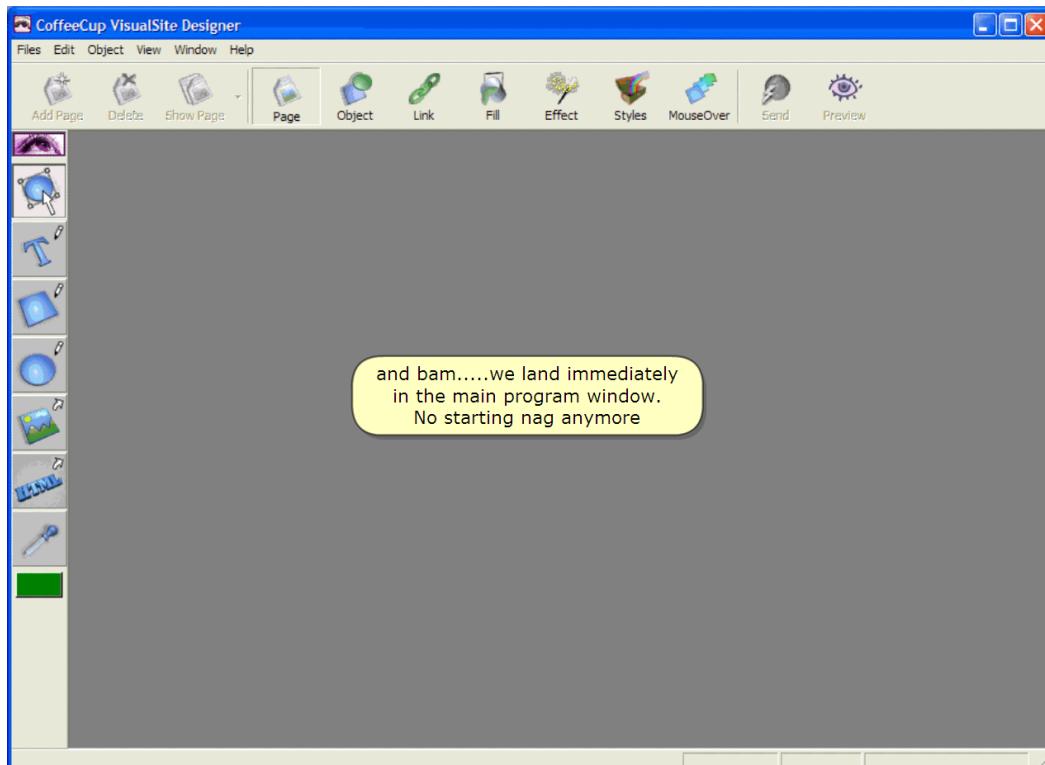
다시 한 번 : 나는 이것이 무슨 일을 하는지 네가 이해했을 거라 생각해. 우리는 JNZ를 항상 JMP하게 바꿀 것이다.

So, jump will be taken now.

그래서, 이제 jump 를 할 거야.

Press F9 to run

F9 를 눌러 실행 해.



And bam.....we land immediately in the main program window. No starting nag anymore

Of course, all this has no influence on the closing nag.

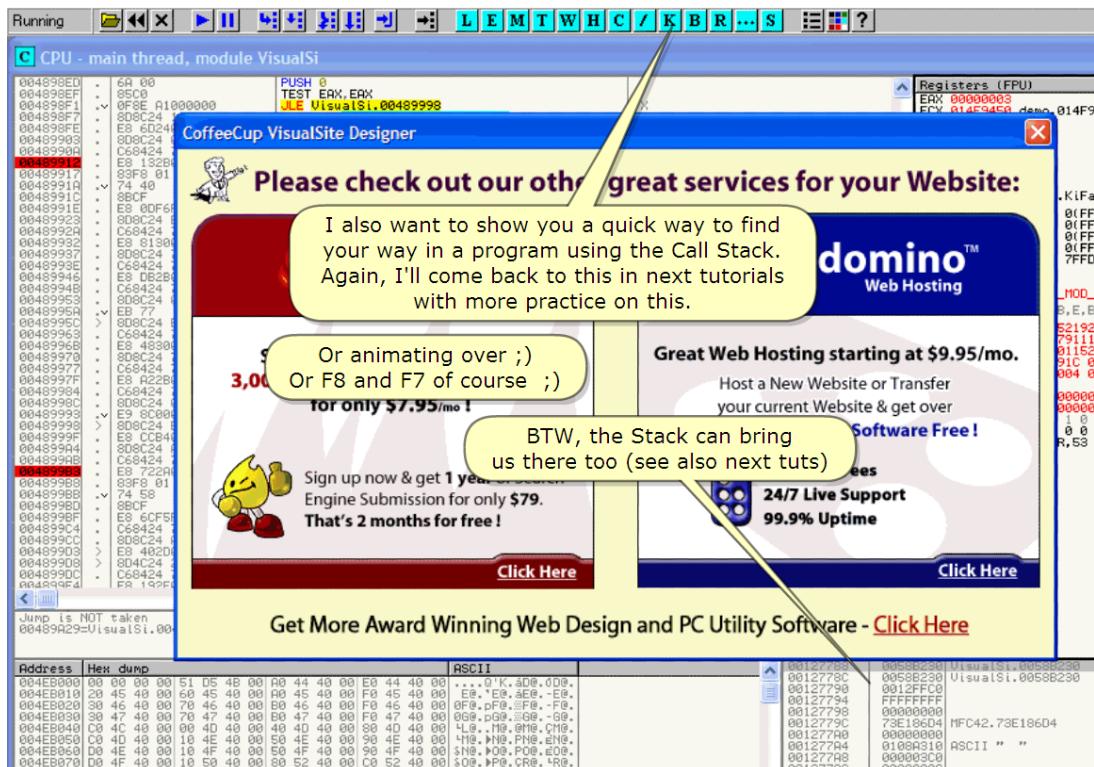
오..우리는 즉시 main program window 에 도착했다. 물론 더 이상 starting nag 이 없다. Closing nag 의 영향도 없다.

So, let's see what happens if we close the app

그래서, 우리가 app 을 닫았을 때 무슨 일이 발생하는지 봐라.

Indeed. Now, this is a completely useless advertisement, right? Let's investigate if our reversing capabilities are that good already that we find how to remove this nag :)

정말, 이제 완벽히 불필요한 광고다. 그렇지? 조사하자. 우리는 reversing 능력이 이미 있다. 우리는 이 nag 을 어떻게 삭제할지 찾을 것이다.



I also want to show you a quick way to find your way in a program using the Call Stack.

나는 너에게 보여주기 위해 program 의 사용하고 있는 Call Stack 에서 빠른 방법을 찾기를 원한다.

Again, I'll come back to this in next tutorials with more practice on this.

다시, 나는 다음 tutorial 에서 좀 더 많은 연습으로 돌아올 것이다.

BTW, the Stack can bring us there too (see also next tuts)

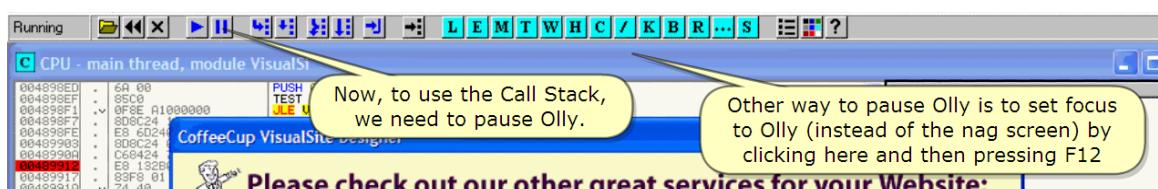
By the way, stack 은 우리에게 가져올 것이다(다음 tuts)

Or animating over ;)

아니면 animating 을 해봐.

Or F8 and F7 of course ;)

아니면 F8 과 F7 을 눌러.



Now, to use the Call Stack, we need to pause Olly.

이제, Call Stack 을 사용하기 위해서 Olly 를 멈춰야 한다.

Other way to pause Olly is to set focus to Olly (instead of the nag screen) by clicking here and then pressing F12

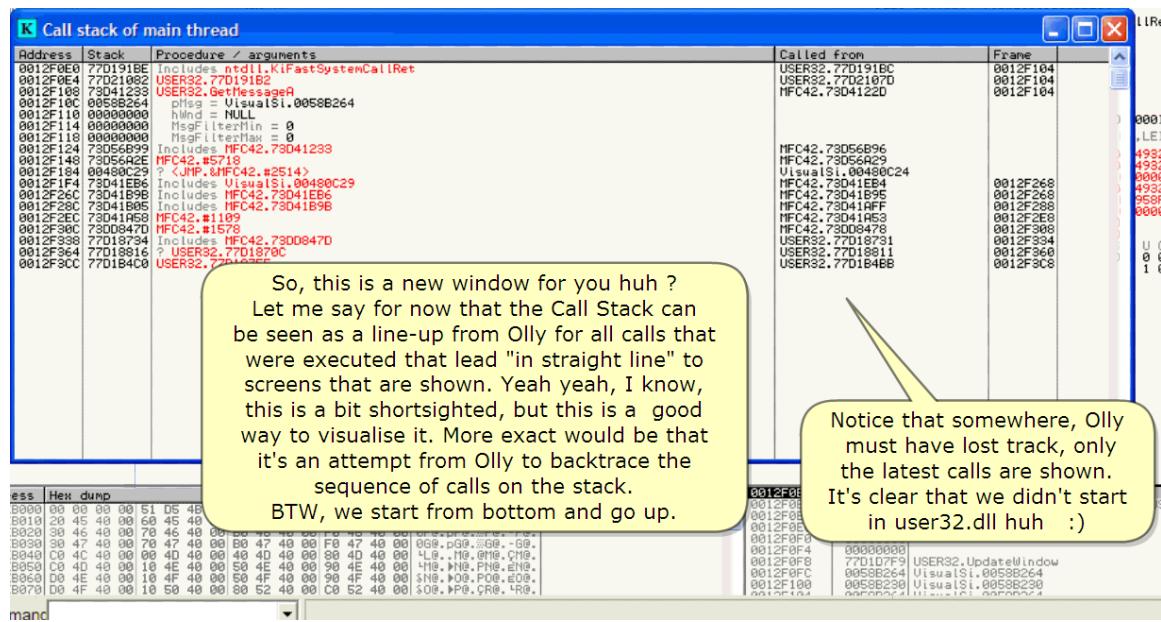
다른 방법으로 clicking 을 하거나 F12 를 누름으로 인해 Olly 를 멈추기 위해 집중하자. (nag screen 대신에)

BTW, if you want to see where in the code the app waits for your input, you can always press Ctrl-F8 (animate over) when Olly is paused and see it all happen with your own eyes

By the way, 만약에 code에서 너의 입력을 기다리는 것을 보기 원한다면. Olly가 멈췄을 때 너는 항상 Ctrl-F8(animate over)를 누를 수 있다. 그리고 무슨 일이 일어나는지 두 눈으로 봐.

And this can always be done if Olly is not running. I trust you understand this "animate over" feature can frequent be used to understand how Olly and how programs work

Olly가 실행 중이 아닐 때 항상 실행됐다. 나는 네가 "animate over" 특징을 빈번히 사용할 것이다. Olly에서 어떻게 사용하고 program은 어떻게 작동하는지 이해 했을 것이라 믿는다.



Call Stack

So, this is a new window for you huh?

이것은 너를 위한 새로운 window다.

Let me say for now that the Call Stack can be seen as a line-up from Olly for all calls that were executed that lead "in straight line" to screens that are shown.

이제 Call Stack을 본 것에 대해서 이야기 하겠다. Olly에서 모든 call이 실행 됐을 때 line-up이 된다. 보여주기 위해 "직접적인 line"으로 이끈다.

Yeah yeah, I know, this is a bit shortsighted, but this is a good way to visualize it. More exact would be that it's an attempt from Olly to backtrace the sequence of calls on the stack.

예, 나도 알아, 꽤 근시안적이라는 것을. 그러나 이것은 시각화하기 좋은 방법이다. 좀 더 정확하게 Olly에서 stack에서 call의 순서를 추적하기 위해 시도해 볼만하다.

BTW, we start from bottom and go up.

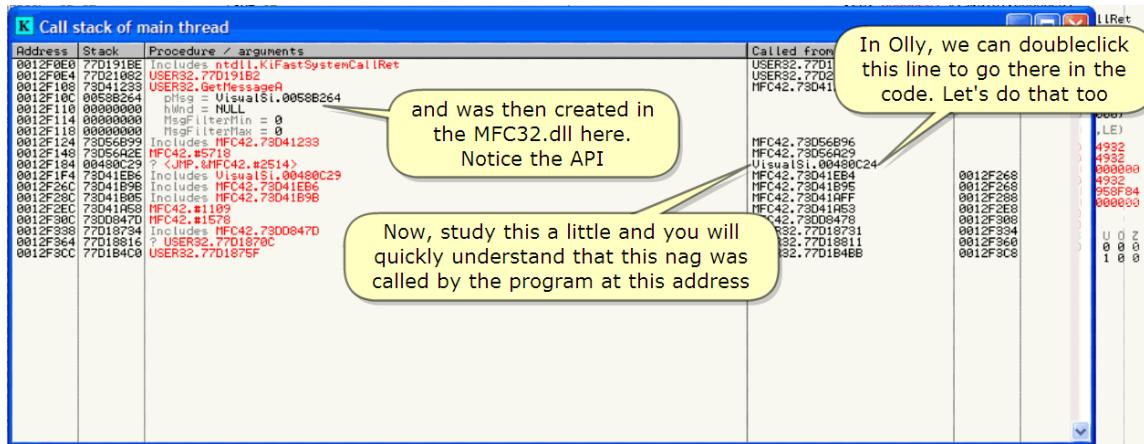
우리는 바닥에서 위로 올라간다.

Notice that somewhere, Olly must have lost track, only the latest calls are shown.

It's clear that we didn't start in user32.dll huh :)

Olly 가 길을 잃은 곳을 알려준다. 오직 최근 call 을 보여준다.

이것은 명확하다. 우리는 user32.dll 에서 시작하지 않았다.



Now, study this a little and you will quickly understand that this nag was called by the program at this address

이제, 이곳을 공부해보자. 그리고 너는 Program 의 이 주소에서 Nag 이 불러지는 것을 빠르게 이해할 것이다.

And was then created in the MFC32.dll here.

Notice the API

그리고 MFC32.dll 이곳에서 생성됐다.

이것은 API 다.

In Olly, we can doubleclick this line to go there in the code. Let's do that too

Olly 에서 우리는 이 line 을 doubleclick 하여 갈 수 수 있다. 해보자.

Aha, so this is the call to the nag

아하, 이 call 은 nag 으로 향한다.

And indeed, see the same call MFC42.dll.#2514 again. Let's place a BP here and scroll up to get a better view on the code above

정말, 우리는 같은 MFC42.dll.#2514 call 을 다시 본다. 이곳에 BP 를 설치하고 좀 더 좋은 시야를 위해 scroll 올려.

C CPU - main thread, module VisualSi

```

00480BED . C2 0400      RETN 4
00480BED . 90          NOP
00480BEE . 90          NOP
00480BEF . 90          NOP
00480BF0 . 6A FF      PUSH -1
00480BF2 . 68 F8764C00 PUSH VisualSi.004C76F8
00480BF7 . 50          MOU EAX,DWORD PTR FS:[0]
00480BFD . 50          PUSH EAX
00480BFE . 64:8925 00000000 MOU DWORD PTR FS:[0],ESP
00480C05 . 83EC 60     SUB ESP,60
00480C08 . E8 37C20300 CALL <JMP.&MFC42.#4501>
00480C0D . 6A 00        PUSH 0
00480C0F . 8D4C24 04    LEA ECX,DWORD PTR SS:[ESP+4]
00480C13 . E8 2833FFFF CALL VisualSi.00463F40
00480C18 . 8D4C24 00    LEA ECX,DWORD PTR SS:[ESP]
00480C1C . C74424 68 00000000 MOU DWORD PTR SS:[ESP+68],0
00480C24 . E8 01B80000 CALL <JMP.&MFC42.#2514>
00480C29 . 8D4C24 00    LEA ECX,DWORD PTR SS:[ESP]
00480C2D . C74424 68 FFFFFFFF MOU DWORD PTR SS:[ESP+68],-1
00480C35 . E8 DEBA0300 CALL <JMP.&MFC42.#641>
00480C3A . 8B4C24 60    MOU ECX,DWORD PTR FS:[0]
00480C3E . 64:8900 00000000 ADD ESP,6C
00480C45 . 89C4 6C      RETN
00480C48 . C3          NOP
00480C49 . 90          NOP
00480C4A . 90          NOP
00480C4B . 90          NOP
00480C4C . 90          NOP
00480C4D . 90          NOP
00480C4E . 90          NOP
00480C4F . 90          NOP
00480C50 . E9 0B000000 JMP VisualSi.004
00480C55 . 90          NOP
00480C56 . 90          NOP
00480C57 . 90          NOP
00480C58 . 90          NOP
00480C59 . 90          NOP
00480C5A . 90          NOP
00480C5B . 90          NOP
00480C5C . 90          NOP
00480C5D . 90          NOP
00480C5E . 90          NOP
00480C5F . 90          NOP
00480C60 . > 68 54605100 PUSH VisualSi.00516054
00480C61 . FF1E A0000000 CALL RtlMoveMemory

```

Mmmm, all right.
 This is an easy one.
 This is a direct call to the nag creation.
 If we NOP this call, the nag screen will be gone forever.
 Let's first try it out.
 So, restart.

Mmmm, all right.

This is an easy one.

This is a direct call to the nag creation.

If we NOP this call,

The nag screen will be gone forever.

Let's first try it out.

So, restart

Remember this address for patching later !!!

음, 좋아.

이것은 쉽다.

이것은 직접적으로 nag 을 생성하는 곳으로 call 한다.

우리는 이 call 을 NOP 한다면,

Nag 은 영원히 없어질 것이다.

먼저 시작해 보자.

그래서, 재시작

나중에 patch 를 위해 이 주소를 기억해.

5. Patching and testing

Yep, we land in the BPs from the cond jumps to the nags

예, 우리는 nag 으로 향하는 조건 jump 가 있는 BP 에 도착했다.

C CPU - main thread, module VisualSi

```

004898E1  . 0F85 42010000 JNZ VisualSi.00489A29
004898E7  . 8B87 E4000000 MOV EAX,DWORD PTR DS:[EDI+E4]
004898ED  . 6A 00 PUSH 0
004898EF  . 85C0 TEST EAX,EAX
004898F1  . 0F8E A1000000 JLE VisualSi.00489998
004898F7  . 8D8C24 10020000 LEA ECX,DWORD PTR SS:[ESP+210]
004898FE  . E8 6D240200 CALL VisualSi.004ABD70
00489903  . 8D8C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+20C]
0048990A  . C68424 78870000 0B MOV BYTE PTR SS:[ESP+8778],0B
00489912  . E8 13280300 CALL <JMP.&MFC42.#2514>
00489917  . 83F8 01
0048991A  . 74 40
0048991C  . 8BCF
0048991E  . E8 0DF6FFFF
00489923  . 8D8C24 B002
0048992A  . C68424 7881
00489932  . E8 81300300
00489937  . 8D8C24 7002
0048993E  . C68424 7881
00489946  . E8 DB280300
0048994B  . C68424 7881
00489953  . 8D8C24 0C01
0048995A  . EB 77
0048995C  . > 8D8C24 B002
00489963  . C68424 78870000
0048996B  . E8 48300300
00489970  . 8D8C24 70020000
00489977  . C68424 78870000 0E
0048997F  . E8 A22B0300
00489984  . C68424 78870000 05
0048998C  . 8D8C24 0C020000
00489993  . E9 8C000000
00489998  . > 8D8C24 B0010000
0048999F  . E8 CCB40100
004899A4  . 8D8C24 AC010000
004899AB  . C68424 78870000 10
004899B3  . E8 722A0300
004899B8  . 83F8 01
004899C0  . 74 F0

```

Now, we have found before that changing this JNZ to always jump was the best solution (remember it?)

Let's make this permanent but first study how this can be done.

Scroll up a little.

Now, we have found before that changing this JNZ to always jump was the best solution (remember it?)

이제, 우리는 해결책을 찾았다. 그리고나서 JNZ를 항상 jump로 바꾸는 것이 가장 좋은 해결책이다.(기억해?)

Let's make this permanent but first study how this can be done.

영원히 바꾼다. 그러나 먼저 어떻게 하는지 공부하자.

Scroll up a little.

스크롤 올려봐.

C CPU - main thread, module VisualSi

```

004898D9  . 8B87 E0000000 MOV AL,BYTE PTR DS:[EDI+E8]
004898D0  . 84C0 TEST AL,AL
004898E1  . 0F85 42010000 JNZ VisualSi.00489A29
004898E7  . 8B87 E4000000 MOV EAX,DWORD PTR DS:[EDI+E4]
004898ED  . 6A 00 PUSH 0
004898EF  . 85C0 TEST EAX,EAX
004898F1  . 0F8E A1000000 JLE VisualSi.00489998
004898F7  . 8D8C24 10020000 LEA ECX,DWORD PTR SS:[ESP+210]
004898FE  . E8 6D240200 CALL VisualSi.004ABD70
00489903  . 8D8C24 0C020000 LEA ECX,DWORD PTR SS:[ESP+20C]
0048990A  . C68424 78870000 0B MOV BYTE PTR SS:[ESP+8778],0B
00489912  . E8 13280300 CALL <JMP.&MFC42.#2514>
00489917  . 83F8 01
0048991A  . 74 40
0048991C  . 8BCF
0048991E  . E8 0DF6FFFF
00489923  . 8D8C24 B002
0048992A  . C68424 7881
00489932  . E8 81300300
00489937  . 8D8C24 7002
0048993E  . C68424 7881
00489946  . E8 DB280300
0048994B  . C68424 7881
00489953  . 8D8C24 0C01
0048995A  . EB 77
0048995C  . > 8D8C24 B002
00489963  . C68424 78870000
0048996B  . E8 48300300
00489970  . 8D8C24 70020000
00489977  . C68424 78870000 0E
0048997F  . E8 A22B0300
00489984  . C68424 78870000 05
0048998C  . 8D8C24 0C020000
00489993  . E9 8C000000
00489998  . > 8D8C24 B0010000
0048999F  . E8 CCB40100
004899A4  . 8D8C24 AC010000
004899AB  . C68424 78870000 10
004899B3  . E8 722A0300
004899B8  . 83F8 01
004899C0  . 74 F0

```

Mmmm, there are different possibilities of course

Do you see one?

Mmmm, there are different possibilities of course

음, 이곳은 다른 가능성 있다.

Do you see one?

보여?

```

004898D9 . 8A87 E0000000 MOU AL,BYTE PTR DS:[EDI+E0]
004898D9 . B0 01 MOU AL,1
004898E1 . D885 43010000 JNZ VisualSi.00489A15
004898E1 . 6887 E4000000 MOU EAX,DWORD PTR DS:[EDI+F0]
004898E1 . 6A 00 SH 0
004898E1 . 85C0 T EAX,EAX
004898E1 . 004898F1 VisualSi.00489A15
004898F1 . 004898F7 ECX,0MORD VisualSi.00489A15
004898F7 . 004898FE VisualSi.00489A15
004898FE . 00489903 ECX,0MORD PTR SS:[ESP+20C]
00489903 . 00489909 C
00489912 . 00489917 E8 132B0300 MOU ECX,DWORD PTR SS:[ESP+2B0]
00489917 . 83F8 01 C
00489917 . 74 40 C
0048991C . 9BCF M
0048991E . E8 0DF6FFFF L
00489923 . 008C24 00020000 L
00489923 . C68424 78870000 0D L
0048992D . E8 S1300300 L
00489932 . 008C24 70020000 L
00489932 . C68424 78870000 0C L
0048993E . 008C24 00020000 L
00489946 . E8 DB2B0300 L
00489946 . C68424 78870000 05 L
00489953 . 008C24 00020000 L
00489953 . C68424 78870000 0C L
00489955 . EB 77 L
00489955 . > 008C24 00020000 L
00489955 . C68424 78870000 0F L
00489955 . E8 45300300 L
00489955 . 008C24 70020000 L
00489955 . C68424 78870000 0E L
0048995F . E8 A22B0300 L
0048995F . C68424 78870000 05 L
00489984 . 008C24 0C020000 L
00489984 . C68424 78870000 05 L
0048998C . E9 0C000000 L
0048998C . 008C24 B0010000 L
0048998C . C68424 CCB40100 L
004899A0 . E8 0D010000 L
004899A0 . 008C24 AC010000 L
004899A0 . C68424 78870000 10 L
004899A0 . E8 722A0300 L
004899A0 . 83F8 01 L
004899B8 . 74 58 L
004899B8 . 9BCF L
004899B8 . E8 6CF5FFFF L
004899B8 . C68424 78870000 05 L
004899C4 . 008C24 0D010000 L

```

Right, this for example

좋은 예다.

Or the same change the line above

거나 위의 line 도 같이 바꾼다.

Anyway, this change also makes AL equal 1 and that's always more secure than just plain stupid patching the JNZ to JMP

어쨌든, 이 변화는 AL 을 1 과 같이 만든다. 그리고 분명히 명청한 JNZ to JMP patch 보다 항상 안전하다.

Now, save these changes to file in the usual way. Because you know how to do it, I skipped this in the movie to reduce its size.

이제, 바뀐 변화들을 일반적인 방법으로 파일로 저장하자. 왜냐하면 우리가 어떻게 했는지 알겠지.

Movie size 를 줄이기 위해서 skip 했다.

And so we land here to start the saved app again. Notice that the app has been saved under another name. Let's see if it works.

그리고 우리는 이곳에 저장된 app 을 다시 시작하기 위해 도착했다. App 은 밑에서 다른 이름으로 저장됐다. 어떻게 작동하는지 보자.

Run

Aha ! The starting nag is killed.

실행

아하! Starting nag 이 죽었다.

So now, let's attack the annoying closing nag.

Go to Olly and

이제 회피하는 closing nag 을 공격하자.

Olly 로 돌아가.

... we land here in the nag code because I meanwhile used the goto button to come here
(removed from video for size)

우리는 nag code 가 있는 이곳에 도착했다. 왜냐하면 나는 goto button 을 가기 위해 사용했다.
(movie size 를 줄이기 위해 삭제했다.)

Told you to remember the address huh?

내가 이야기했던 주소 기억해?

Remember this was the NAG?

Nag 이라는걸 기억해?

Of course, the BP's are not here anymore because the application was restarted under a different name.

물론, BP 는 이곳에 더 이상 없다. 왜냐하면 application 을 다른 이름으로 재시작 했기 때문이다.

I'll show you how it looked when we last left here :)

나는 너에게 우리가 마지막에 머물렀던 이곳을 어떻게 보는지 보여줄 것이다.

CPU - main thread, module VisualSi

Address	OpCode	Mnemonic	Operands
00480BC2	. 6A 00	PUSH	0
00480BC4	. 6A 00	PUSH	0
00480BC6	. 68 04605100	PUSH	VisualSi.004605100
00480CB	. 68 90B84F00	PUSH	VisualSi.004FB890
00480BD0	. 6A 00	PUSH	0
00480BD2	. FF15 A0CC4C00	CALL	DWORD PTR DS:[<&SHELL32.ShellExecuteA>]
00480BD8	. C3	RET	
00480BD9	. 90	NOP	
00480BDA	. 90	NOP	
00480BDB	. 90	NOP	
00480BDC	. 90	NOP	
00480BDD	. 90	NOP	
00480BDE	. 90	NOP	
00480BDF	. 90	NOP	
00480BE0	. 8B4C24 04	MOV	ECX, DWORD PTR SS:[ESP+4]
00480BE4	. 6A 01	PUSH	1
00480BE5	. 8B01	MOV	ERX, DWORD PTR DS:[ESP+4]
00480BE8	. FF10	CALL	DWORD PTR DS:[ESP+4]
00480BED	. C2 0400	RET	4
00480BEE	. 90	NOP	
00480BFF	. 90	NOP	
00480BF0	. 6A FF	PUSH	-1
00480BF2	. 68 F8764C00	PUSH	VisualSi.004C00
00480BF7	. 64:A1 00000000	MOV	ERX, DWORD PTR FS:00000000
00480BF0	. 50	PUSH	EAX
00480BFE	. 64:8925 00000000	MOV	DWORD PTR FS:D0
00480B05	. 83EC 60	SUB	ESP, 60
00480C03	. E8 37C20300	CALL	<JMP.&MFC42.#2514>
00480C00	. 6A 00	PUSH	0
00480C04	. 8D4C24 04	LEA	ECX, DWORD PTR SS:[ESP+4]
00480C13	. E8 2823FFFF	CALL	VisualSi.004C00
00480C18	. 8D4C24 00	LEA	ECX, DWORD PTR SS:[ESP+4]
00480C1C	. C74424 68 00000000	MOV	DWORD PTR SS:[ESP+8], 0
00480C24	. E8 B1E83000	CALL	<JMP.&MFC42.#2514>
00480C29	. 8D4C24 00	LEA	ECX, DWORD PTR SS:[ESP+4]
00480C2D	. C74424 68 FFFFFFFF	MOV	DWORD PTR SS:[ESP+8], -1
00480C35	. E8 DEBA0300	CALL	<JMP.&MFC42.#641>
00480C3A	. 884C24 60	MOV	ECX, DWORD PTR SS:[ESP+4]
00480C3E	. 64:8900 00000000	ADD	ESP, ECX
00480C45	. 83C4 6C	RET	
00480C48	. C3	NOP	
00480C49	. 90	NOP	
00480C4A	. 90	NOP	
00480C4B	. 90	NOP	

And also remember that we can NOP this right away, without needing to dig deeper in the dll
그리고 기억해. 우리는 NOP 을 할 수 있다. DLL 속으로 깊이 들어가는 게 필요하지 않다.

C CPU - main thread, module VisualSi

```

00480BC2 . 6A 00      PUSH 0
00480BC4 . 6A 00      PUSH 0
00480BC6 . 68 04605100 PUSH VisualSi.00516004
00480BCB . 68 90B84F00 PUSH VisualSi.004FB890
00480BD0 . 6A 00      PUSH 0
00480BD2 . FF15 A0CC4C00 CALL DWORD PTR DS:[&SHELL32.ShellExecuteA]
00480BD8 . C3          RETN
00480BD9 . 90          NOP
00480BDA . 90          NOP
00480BDB . 90          NOP
00480BDC . 90          NOP
00480BDD . 90          NOP
00480BDE . 90          NOP
00480BDF . 90          NOP
00480BE0 . 8B4C24 04    MOV ECX,DWORD PTR SS:[ESP+4]
00480BE4 . 6A 01      PUSH 1
00480BE6 . 8B01      MOV EAX,DWORD PTR DS:[ECX]
00480BE8 . FF10      CALL DWORD PTR DS:[EAX]
00480BEA . C2 0400    RETN 4
00480BED . 90          NOP
00480BEE . 90          NOP
00480BEF . 90          NOP
00480BF0 . 6A FF      PUSH -1
00480BF2 . 68 F8764C00 PUSH VisualSi.004C76F8
00480BF7 . 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
00480BFD . 50          PUSH EAX
00480BFE . 64:8925 00000000 MOU DWORD PTR FS:[0],ESP
00480C05 . 83EC 60     SUB ESP,60
00480C08 . E8 37C20300 CALL <JMP.&MFC42.#4501>
00480C0D . 6A 00      PUSH 0
00480C0F . 8D4C24 04    LEA ECX,DWORD PTR SS:[ESP+4]
00480C13 . E8 2833FFFF CALL VisualSi.00468F40
00480C18 . 8D4C24 00    LEA ECX,DWORD PTR SS:[ESP]
00480C1C . C74424 68 00000000 MOU DWORD PTR SS:[ESP+68],0
00480C24 . 90          NOP
00480C25 . 90          NOP
00480C26 . 90          NOP
00480C27 . 90          NOP
00480C28 . 90          NOP
00480C29 . 8D4C24 00    LEA ECX,DWORD PTR SS:[ESP]
00480C2D . C74424 68 FFFFFFFF MOU DWORD PTR SS:[ESP+68],-1
00480C35 . E8 DEBA0300 CALL <JMP.&MFC42.#641>
00480C3A . 8B4C24 60    MOU ECX,DWORD PTR SS:[ESP+60]
00480C3E . 64:8900 00000000 MOU DWORD PTR FS:[0],ECX
00480C45 . 89F4 AF    ann ESP AR

```

DefDir = NULL
Parameters = NULL
FileName = "http://
Operation = "open"
hInst = NULL
ShellExecuteA

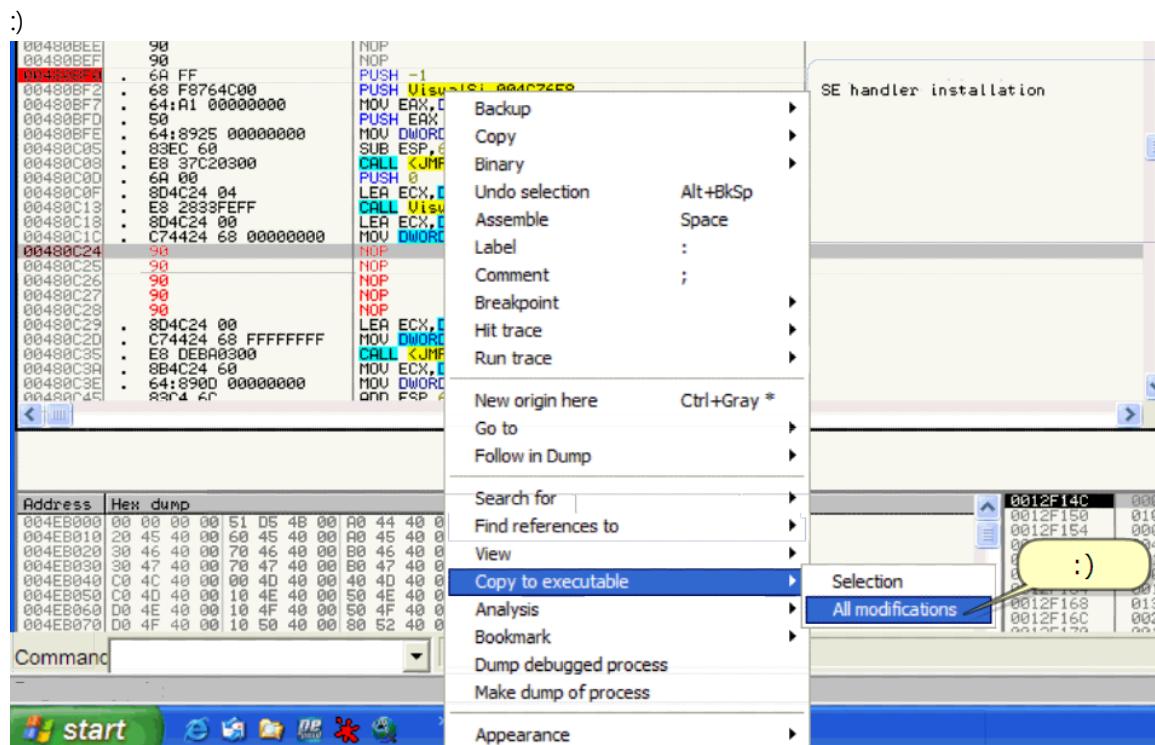
VisualSi.004D66B0

SE handler installa

Ok. Let's also make these changes permanent and save them to file

Ok. Let's also make these changes permanent and save them to file

Ok. 영원히 바꾸자. 그리고 file로 저장해.



:)

Let's immediately test the saved file

즉시 저장된 file 을 test 하자.

:)

Mmmm, this all seems all right, and now, what's with the closing nag ????

음, 좋아, 이제, closing nag 은 어떻게 됐지?

Starting and closing nags are gone !!

Starting 과 closing nag 이 없어졌다 !!

The program is working like if it were registered.

등록된 것처럼 Program 은 일한다.

6. Conclusion

질문

잘 이해가 안됨. 다시 한 번 패치하는 부분을 이해해야 됨. 점프뛰는 부분이 잘 이해가 안돼. 한글판도 보면서 이해해 보자.

In this part 5, the primary goal was to get the feeling of the code, to understand how code is executed.

이번 part 5 에서는, 가장 중요한 목표는 code 에 대한 느낌을 얻는 것이다. Code 가 어떻게 실행되는지 이해했을 것이다.

Also, we learned to find how and where to patch a program that shows visible other behaviour when or when not registered by simple stepping and comparing the conditional jumps.

또한, 우리는 어떻게 찾는지 program 을 패치하는지 배웠다. 다른 behaviour 를 보여준다. 간단한 진행과 조건 jump 의 비교에 의해 등록 됐을 때나 등록되지 않았을 때 보여준다.

Meanwhile, we also learned something about animate over/in and setting different breakpoints.

그 동안, 우리는 또한 무언가를 animate over/in 하는 것과 다른 breakpoint setting 법을 배웠다.

I hope you understood everything fine and I also hope someone somewhere learned something from this. See me back in part 06 ;)

모든 것을 잘 이해했기를 바란다. 그리고 나는 누구든지 어느 곳에서든지 이것에서 무엇이든지 배웠기를 바란다. Part 06 에서 돌아올 것이다

The other parts are available at

다른 parts 는 사용 가능하다.

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

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

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

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

Lena151 (2006, updated 2007)

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