IDA - Interactive Disassembler

By Wr4ith

챕터 1 – 사전 준비

파트 0 – 소개

안녕하세요 여러분, 이 문서는 저의 첫 번째 튜토리얼이자 레슨입니다. 따라서 다소 미숙할 수 있으니 너그럽게 봐주시길 바랍니다. 영어가 저의 모국어가 아니기 때문에 약간의 오타가 있을 수 있습니다. 오타를 발견한다면 수정을 위해 저에게 연락을 해주시기 바랍니다.

몇몇 사람들은 크래킹에 대하여 지식이 있는 사람들은 이미 IDA를 사용하는 법을 알고, 뉴비분들은 대부분 W32DASM를 쓰다가 나중에 고급적인 툴들을 사용하고 있음에도 불구하고 저에게 왜 이 튜토리얼을 작성했는지 묻습니다. 저는 좀 색다른 접근 방법을 시도하려고 하는 것인데 문서를 작성했을 시기는 2003년인데 W32DASM 툴은 상당히 많은 오류가 발견되었고 IDA에 비하여 강력하지 않습니다. 따라서 저는 뉴비분들이 처음부터 IDA를 사용하게 만들고 싶었습니다. IDA는 자동 주석을 제공하기 때문에 어셈블러 언어가 뉴비분들에게 그리 어렵지 않습니다.

물론 어셈블러 책을 참고용으로 가지고 있는 것은 매우 좋으나 몇몇의 어셈블러 들은 그냥 주석을 보는 것이 더 명확하게 이해할 수 있습니다. 이 튜토리얼을 보기 위해서는 어셈블러에 대한 지식이 없어도 되고, 두 번째 튜토리얼부터 어셈블러에 대해 다룰 예정입니다. 이번 튜토리얼에서는 IDA에서 제공되는 대부분의 기능들을 다룰 것입니다. 물론 완벽하지 않을 테고 IDA의 도움말(help file)을 대체할 수 없을 겁니다. IDA 사용 도중 문제가 발생한다면 반드시 도움말을 읽어보시길 바랍니다.

저는 최대한 많은 것들을 스크린 샷과 함께 설명하지만 모든 부분에 대하여 친절하게 단계별로 사진과 함께 설명이 있다는 기대는 하지 말기 바랍니다.

파트 1 – IDA 설치

IDA를 설치하는 것은 굉장히 간단합니다. (최근 버전은 제공하는 비밀번호를 기반으로 그냥 설치를 진행하면 됩니다.) 그냥 압축 파일을 원하는 곳에 풀면 됩니다. 이번 튜토리얼에서는 [1]"IDA PRO Advanced 4.30"을 사용합니다.
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced 4.30₩

파트2 – 플러그인과 추가기능 다운로드

지금까지 잘 따라오셨길 바랍니다. 인터넷에서는 IDA에 대한 플러그인과 추가 기능들이 있는데 지금 알려주는 것들은 꽤 쓸 만한 것들이고 대부분의 경우에 필요하다가 말할 수 있습니다.

LoadINT 4.21	- IDA에서 자동으로 표시되는 주석 변경
Flair Tools 4.16	- 사용자 자신만의 시그니처 생성
SIE Plugin	- 문자열, 임포트, 익스포트에 대한 윈도우 창 생성 * 최신 버전의 IDA에서는 기본적으로 제공되는 기능임
Ida2Softice	- 어플리케이션 디버깅을 쉽게 하기 위해 NMS 파일 생성
Ida 4.3 SDK	- 사용자 자신만의 플러그인을 작성할 수 있음

위에서 언급한 것들은 다음의 URL에서 받을 수 있습니다.
http://mostek.subcultural.com
http://wasm.ru/toollist.php?list=13

* URL이 유효하지 않은 경우 구글 등의 검색 엔진을 통해 다운받으면 됩니다.

1) 2015-02-14 기준으로 최신 버전은 6.9 이기 때문에 약간의 기능/UI 차이가 있을 수 있으나 기본 기능은 동일하기 때문에 뉴비 분들은 이 문서를 통해 기분 기능을 배우고 나중에 좀 더 고급적인 기능을 배우면 될 듯 하빈다.

파트3 – 플러그인과 추가기능 설치

LoadInt 4.21

IDA 메인 폴더에 모든 파일을 압축 풀면 됩니다. 나중에 읽어야 하는 경우가 있을 수 있으니 반드시 LoadInt의 README 파일의 이름을 변경하기 바랍니다.

e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced 4.30₩

Flair Tools 4.16

IDA 폴더 내에서 별도의 서브 폴더를 생성하고 모든 파일/폴더를 압축 풀면 됩니다.

e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced 4.30₩Flair Tools

Ida Pro SDK 4.30

IDA 폴더 내에서 별도의 서브 폴더를 생성하고 모든 파일/폴더를 압축 풀면 됩니다.

e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced 4.30₩Ida SDK 4.3

SIE Plugin (설치)

임시 폴더에 모든 파일을 압축 해제하고, "plugs.plw" 파일을 IDA 플러그인 폴더에 복사하면 됩니다.

e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced 4.30₩Plugins

반드시 사용하고자 하는 IDA의 버전에 맞는 파일을 복사하기 바랍니다. 예를 들어 만약 IDA PRO Advanced 4.30을 사용한다면 "plugs.plw" 파일은 다음과 같은 폴더에 있는 것을 복사해야 합니다.

e.g. C:₩tempdirectory₩4.30₩plugs.plw

SIE Plugin (변경)

SIE Plugin 사용을 위해서 IDA 플러그인 폴더에 있는 "plugins.cfg" 파일을 변경해야 합니다.

e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced 4.30₩Plugins₩Plugins.cfg

해당 파일의 마지막에 다음과 같은 라인들을 추가해야 합니다.

```
Strings_BugFix    plugs    0          3
Exports           plugs    SHIFT-E    2
Imports           plugs    SHIFT-I    1
Strings           plugs    SHIFT-S    0
```

추가한 사항에 대하여 저장하고 임시 폴더를 삭제하면 됩니다. 좀 더 자세한 사항은 "Plugins.cfg" 파일 내의 예제 부분을 참고하길 바랍니다.

Ida2Softice Plugin

임시 폴더에 모든 파일을 압축 해제하고, "i2s.plw" 파일을 IDA 플러그인 폴더에 복사합니다.
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced₩4.30₩Plugins

SIE 플러인과 마찬가지로 IDA PRO Advanced 4.30을 사용하고자 한다면 다음 폴더에 있는 "i2s.plw" 파일을 사용해야 합니다.
e.g. C:₩tempdirectory₩4.30₩i2s.plw
이제 IDA 플러그인 폴더 내의 "plugins.cfg" 파일을 변경해야 합니다. 해당 파일의 끝부분에 다음과 같은 라인을 추가해야 합니다.

I2S_Setup	i2s	0	3
I2S_Source_Info	i2s	Ctrl-F12	2
I2S_Save_NMS	i2s	Shift-F12	1
I2S_Conversion	i2s	F12	0

추가한 사항에 대하여 저장하고 임시 폴더를 삭제하면 됩니다. 좀 더 자세한 사항은 "Plugins.cfg" 파일 내의 예제 부분을 참고하길 바랍니다.

파트4 – 자동 주석 변경

만약 LoadINT 4.21를 설치했다면 IDA 메인 폴더에 "PC.CMT" 파일이 있을 겁니다. 사용하는 에디터를 이용하여 파일 내용을 변경하고 주석이 필요한 추가적은 어셈블러 명령어 대하 자신만의 주석을 추가하면 됩니다. 또한 좋은 어셈블러 책을 참고로 주석들을 자신만의 방식으로 변경하면 됩니다. 다만 이것은 어셈블러를 막 사용하기 시작한 사람을 위한 일종의 힌트 정도입니다.

IDA에서 사용자가 변경한 자동 주석을 보이기 하기 위해서는 다음과 같은 작업이 필요합니다.

1. "PC.CMT" 파일을 변경하고 저장
2. IDA 메인 폴더의 "COMPILE.BAT" 파일 호출

좀 더 자세한 설명은 "LoadINT 4.21" 압축 해제 파일 중 Readme 파일을 살펴보길 바랍니다. 참고로 "COMPILE.BAT" 파일을 실행하는 동안에는 IDA가 실행되지 않아야 합니다. 그렇지 않으면 "CMT" 파일의 문법에 오류가 없음에도 에러가 발생합니다. 왜냐하면 컴파일 프로그램이 "IDA.INT" 파일 내용을 변경하는 도중에, IDA가 실행중이면 권한 문제가 발생해 제대로 열리지 않기 때문입니다.
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced 4.30₩PC.CMT
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced 4.30₩Compile.bat
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced 4.30₩README

파트5 – 설정 파일

설정 파일을 아는 것은 꽤 유용한데, 왜냐하면 IDA는 사용자가 세팅한 옵션을 자동으로 저장하지 않기 때문입니다. 지금까지의 챕터들을 모두 따라왔다면 아마 추가적인 변경을 원할 수도 있을 텐데, 설정 파일에 대한 자세한 설명은 하지 않을 것입니다. 사실 해당 파일들은 꽤 상세한 주석이 제공되고 사용자의 요구사항(예를 들어 매크로를 추가하거나, 단축키 수정 등)에 따라 쉽게 수정할 수 있습니다.

설정 파일들의 경로와 파일 이름들은 다음과 같습니다.
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced₩IDA.CFG
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced₩IDAGUI.CFG
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced₩IDATUI.CFG
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced₩SIG₩Autoload.cfg
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced₩Plugins₩Plugins.cfg
e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced₩IDS₩IDSNAMES

물론 더 많은 파일들이 있지만, 이는 다른 프로세서 모듈을 위한 것이기 때문에 여기서는 필요하지 않습니다.

챕터 2 – 초기 접근

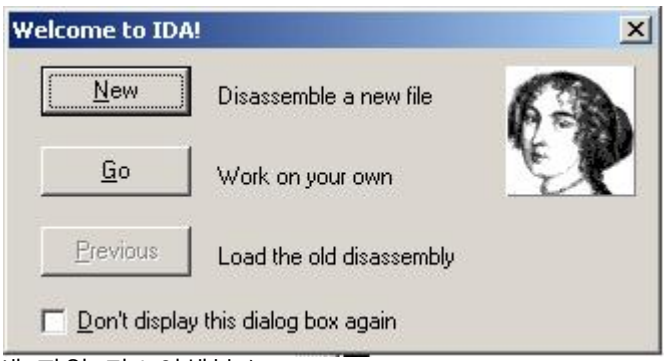
파트 6 – 초기 접근 방법

다양한 것들을 보여주고 사전 준비를 보여주었으니 이제 IDA를 시작하여 프로그램을 살펴볼 차례인데, IDA 메인 폴더에는 꽤 다양한 바이너리 파일이 존재합니다.

IDA2.EXE - OS/2 환경에서 실행 시 사용되는 파일
IDAX.EXE - DOS 환경이나, 프로텍터 모드에서 DOS/4GW 확장자 실행 시 사용되는 파일
IDAW.EXE - 일반적인 DOS 모드에서 실행 시 사용
IDAG.EXE - Win95 이상에서 GUI 환경으로 실행 시 사용
* 최근에는 GUI 환경만 지원하고 바이너리가 "idaq.exe"로 변경 됨

이번 튜토리얼에서는 가장 많이 사용되는 GUI 환경에서의 IDA만 다룰 예정입니다.

IDAG.EXE (최신 버전은 IDAQ.EXE)를 실행하고 라이선스 다이얼로그 창이 나타나면 "OK" 버튼을 누르면, 다음과 같이 3가지 선택이 주어지는 창을 보게 됩니다.



1. New (새 파일 디스어셈블)
2. Go (아무것도 없는 환경에서 작업 시작)
3. Previous (이전에 디스어셈블 된 파일 로딩)

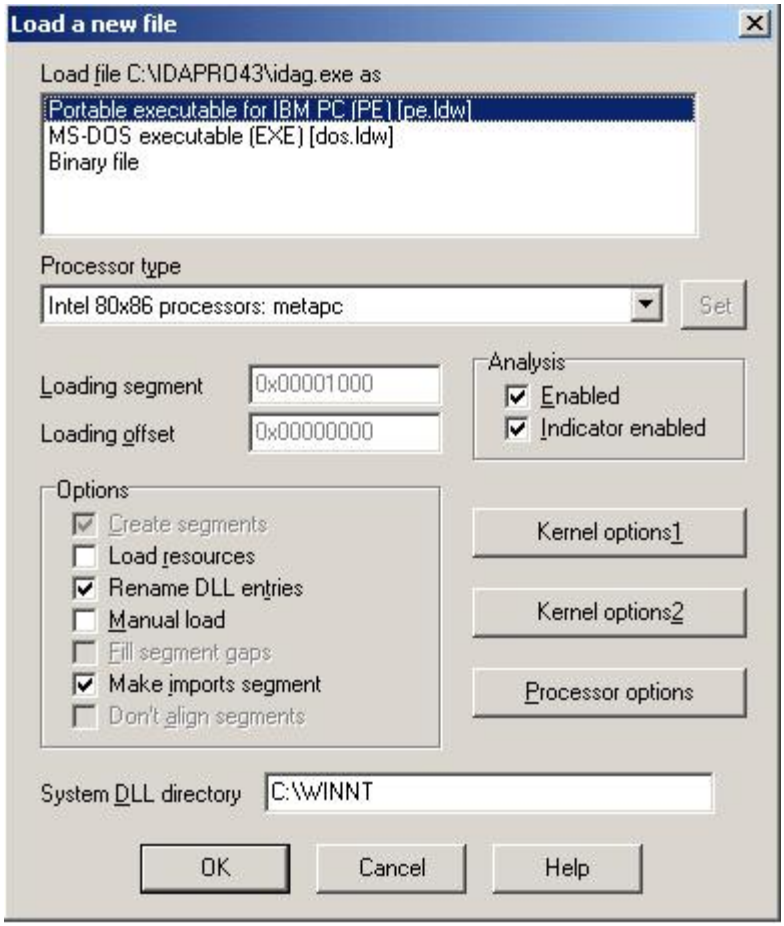
하단의 "Don't display this dialogs box again"을 선택하면 다음부터는 자동으로 2번째 선택(Go)로 실행하는데, 그럼에도 불구하고 계속 위 사진의 창이 보이면 설정 파일에서 변경하면 됩니다. 지금은 처음으로 IDA를 다루는 것이니, "New"를 선택하면 그 다음 "File Dialog" 창이 나타나는데 해당 창에서 디스어셈블하고자 하는 파일을 선택합니다. 튜토리얼과 동일한 버전의 IDA를 사용하면 IDA 폴더의 "IDAG.EXE" 파일을 선택[2]한 다음 "OK" 버튼을 클릭하면 됩니다.

e.g. C:₩Program Files₩Datarescue₩IDA PRO Advanced 4.30₩IDAG.EXE

---

2) IDA의 기능을 소개하는 튜토리얼이기 때문에 굳이 동일한 바이너리가 아니어도 됩니다.

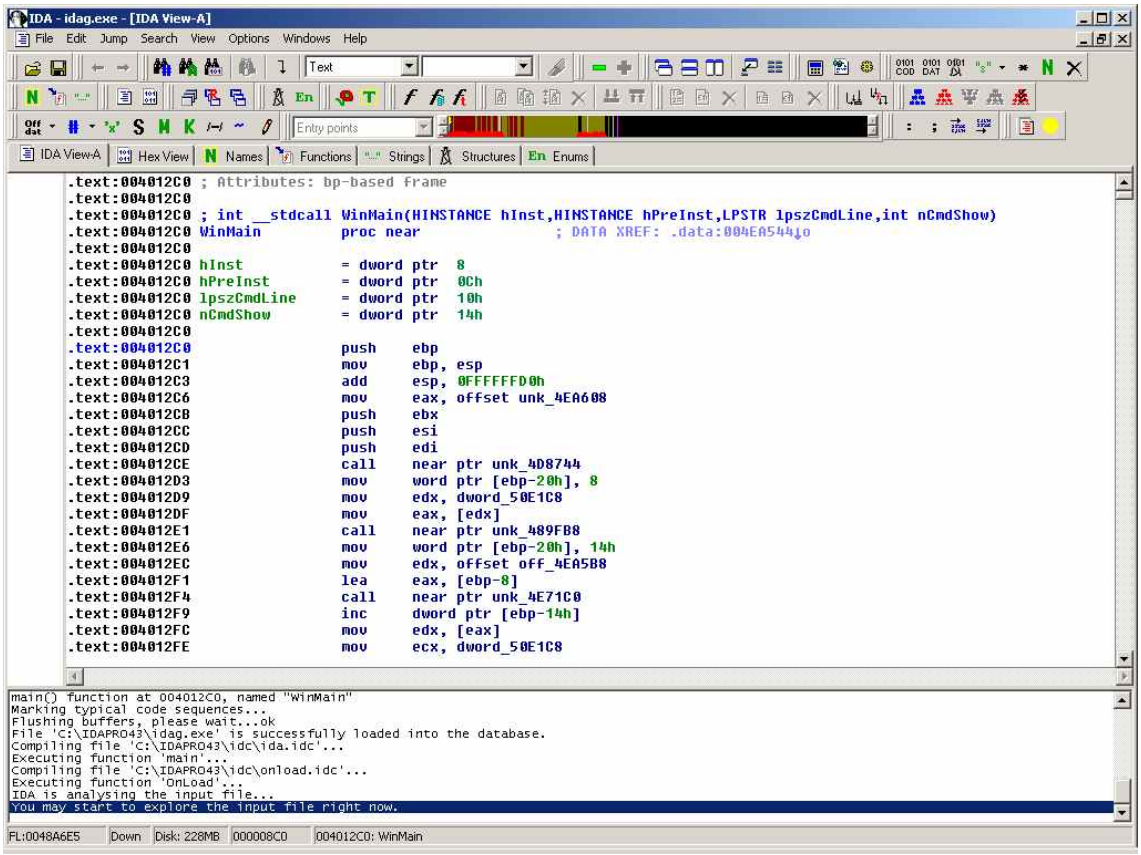
그러면 IDA는 다음 사진과 같은 또 다른 다이얼로그 창을 띄웁니다.



위 다이얼로그에서는 사용자가 IDA에게 분석하고자 하는 파일에 대한 정보를 알려 줄 수 있습니다. 대부분의 윈도우 파일들은 PE 파일로 불리는데, 이는 파일이 어떻게 보이는지 정의된 일종의 포맷입니다. 따라서 자동으로 선택되는 것이 99% 이상 맞는 경우입니다. 참고로 프로세서 타입은 "Intel 80x86 processors: metapc" 인데, 이는 IDA가 인텔이라고 명시했음에도 불구하고 모든 어셈블러 명령어 해석을 시도한다는 뜻이며, 해당 세팅에는 언급하지 않은 다른 것들이 많지만 이 세팅이 가장 쓸 만합니다. 만약 프로그램에서 어떤 아키텍처를 사용했는지 정확히 알고 있다면 프로세서 타입을 변경해도 무방합니다. 다만 이번 튜토리얼에서 프로세서 세팅은 대부분 "metapc"를 이용합니다.

프로세서 타입을 어떻게 처리할지 알았다면 "Kernel option", "Processor Options"들을 살펴보아 약간의 튜닝 작업을 시도할 수 있습니다. 그러나 지금은 우선 기본적인 세팅을 이용할 것이기 때문에 바로 "OK" 버튼을 누르면 됩니다.

그러면 IDA가 분석 작업을 시작합니다. 몇몇의 메시지를 화면에 출력하면 드디어 처음으로 디스어셈블 된 코드를 볼 수 있게 됩니다. 처음 할 일은 툴바를 정리하는 것으로서 "Overview Navigation" 윈도우를 툴바 쪽으로 이동시키는 것이고, 다음에는 "IDA View A" 윈도우의 사이즈를 최대한 크게 늘리는 것입니다. 이제 IDA는 다음과 같이 보이게 됩니다.



참고로 위 사진에서 3번째 툴바의 우측에 보이는 작은 원형 아이콘의 색은 IDA의 진행 상태를 의미합니다. 해당 아이콘은 총 3가지 색을 가질 수 있습니다.

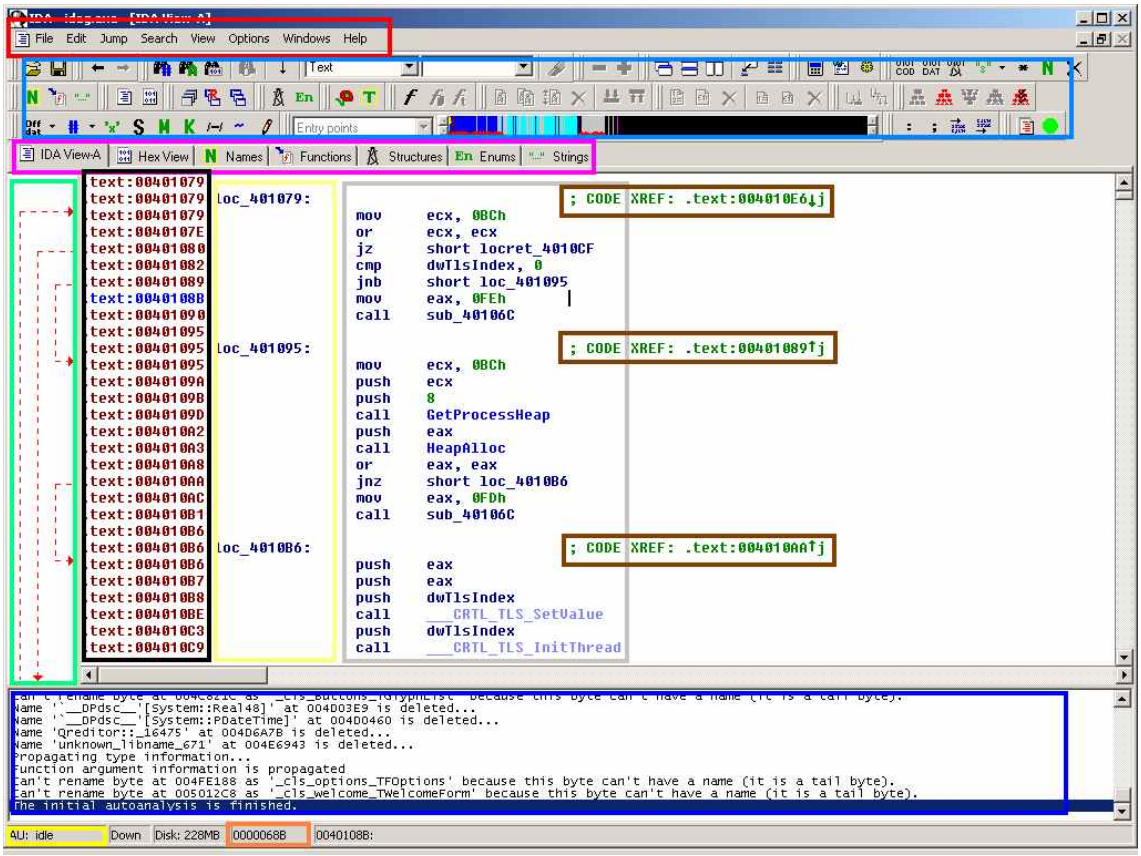
초록색 : 준비 완료

노란색 : 분석 중

빨간색 : 오류 발생

분석 환경의 CPU에 따라서 디스어셈블 과정이 약간의 시간이 걸릴 수 있습니다. 디스어셈블링이 완료되면 해당 원 아이콘은 초록색으로 변경되고 "The initial autoanalysis is finished" 라는 메시지가 "status window(상태 창)"에 나타납니다.

파트7 – 메인 윈도우



위 사진에서 여러 가지 색으로 이루어진 사각형 박스에 대한 설명은 다음과 같습니다.

빨간색

대부분의 프로그램에 있는 것처럼 IDA의 메뉴 바를 의미합니다.

하늘색(연파란색)

IDA의 대부분 기능들을 클릭하여 사용할 수 있는 툴바를 의미합니다.

핑크색

"IDA View A (메인 윈도우)", Hewview, Strings, Names, Functions 등의 각각 다른 윈도우 창들의 목록을 의미합니다.

초록색

코드 블록에서 존재하는 분기문이 어디로 이동되는지 보여주는 화살표를 의미합니다. 이는 호출 함수나 반복문 등을 분석할 때 유용하게 사용됩니다.

검은색
가상 주소를 기반으로 나타나는 섹션 이름을 의미합니다. 이는 Softice / OllyDbg 등에서 보이는 주소와 동일합니다.(ASLR 미적용인 경우)

검은색 사각형 내의 파란색 테스트
현재 위치를 보여주는 코드 라인를 의미합니다.

연노란색
점프 표시에 비교될 수 있는 코드 위치. 함수 자신을 제외하고는 모든 분기문(점프)은 해당 위치가 마킹됩니다.

회색
디스어셈블 된 프로그램의 코드를 의미합니다.

갈색
코드 레퍼런스 : 프로그램의 어느 부분에서 해당 코드 주소를 액세스하는지 의미합니다.
해당 부분을 더블클릭하면 해당 주소/함수가 호출/점프되는 곳으로 이동 됩니다.

남색
사용자의 마지막 작업에 대한 상태창(로그 정보)을 의미하며, 현재 IDA가 어떤 작업을 하고 있는지 보여줍니다.

노란색
IDA가 동작중인지, 만약 동작중이라면 어떤 주소에서 동작중인지 보여주는 상태창를 의미합니다.

주황색
현재의 코드 주소에 대한 파일 오프셋 : 프로그램을 패치할 때 정확한 주소를 찾기 위해 (가상 주소 <-> 파일 오프셋)를 변환할 때 유용하게 사용됩니다.

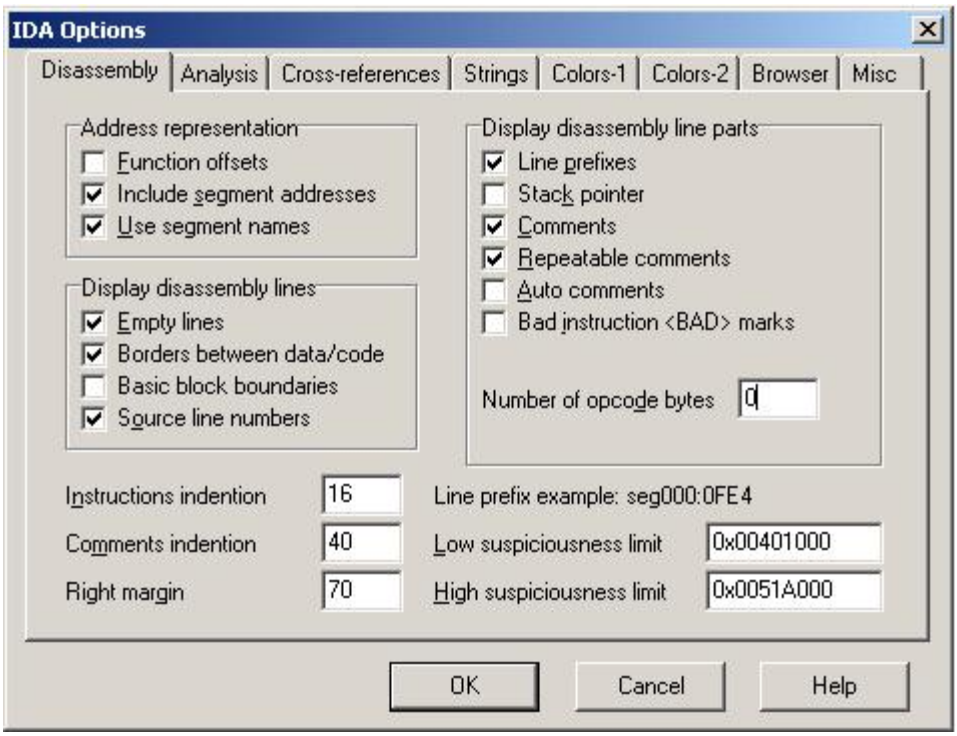
파트 8 – 플러그인 사용

본 튜토리얼 문서의 파트 3에서 IDA 플러그인의 설치법에 대하여 설명한 부분을 잘 따라왔다면 드디어 IDA에서 해당 플러그인들을 사용할 차례입니다.

크게 2가지 방법이 있는데, "plugins.cfg" 파일을 변경할 때 추가했던 단축키를 이용하거나 다음과 같은 메뉴를 이용하여 수동적으로 이용하는 방법이 있습니다.
e.g. Edit/Plugins/String References
e.g. Edit/Plugins/Imports
e.g. Edit/Plugins/Exports
e.g. Edit/Plugins/Ida2Softice
플러그인들을 호출하면 새로운 윈도우 창이 팝업되거나, 이전 사진의 핑크색 박스에 새로운 윈도우가 추가됩니다.

파트 9 – 옵션 알아보기 (및 정리)

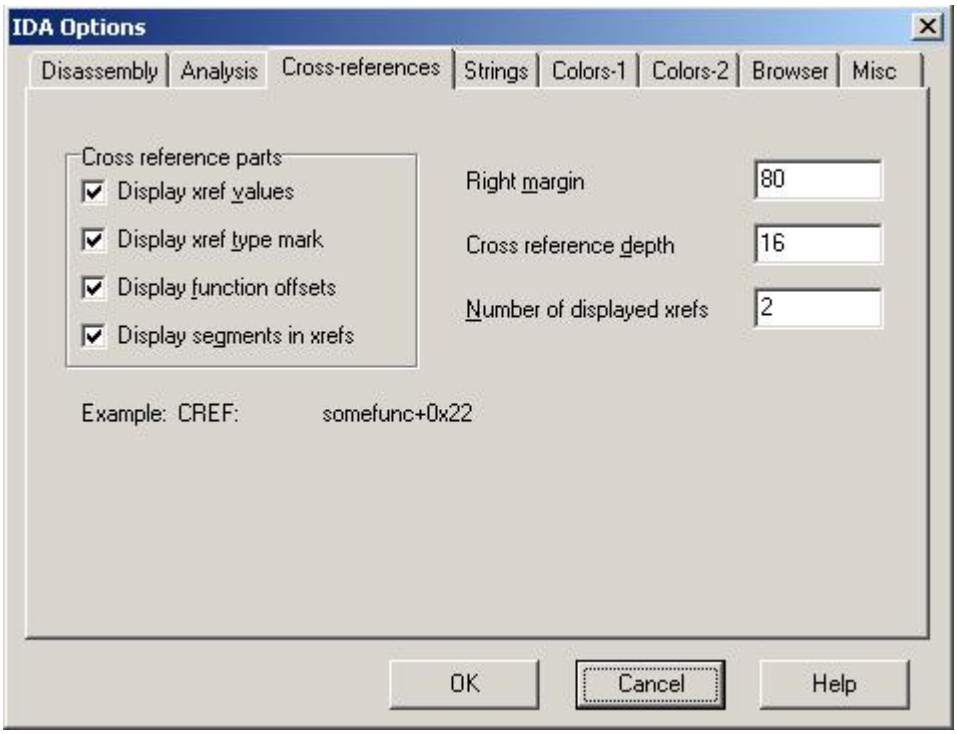
옵션 메뉴를 사용하기 위해서는 메뉴바의 "Options/General"을 누르면 됩니다. 그러면 다음과 같은 창이 나타나게 됩니다.



위 옵션 창에서 "Stack Pointer, Auto comments, Bad instruction marks, Basic Block boundaries" 체크박스에 체크를 하고, "Number of opcode bytes" 항목은 '8'로 세팅합니다.

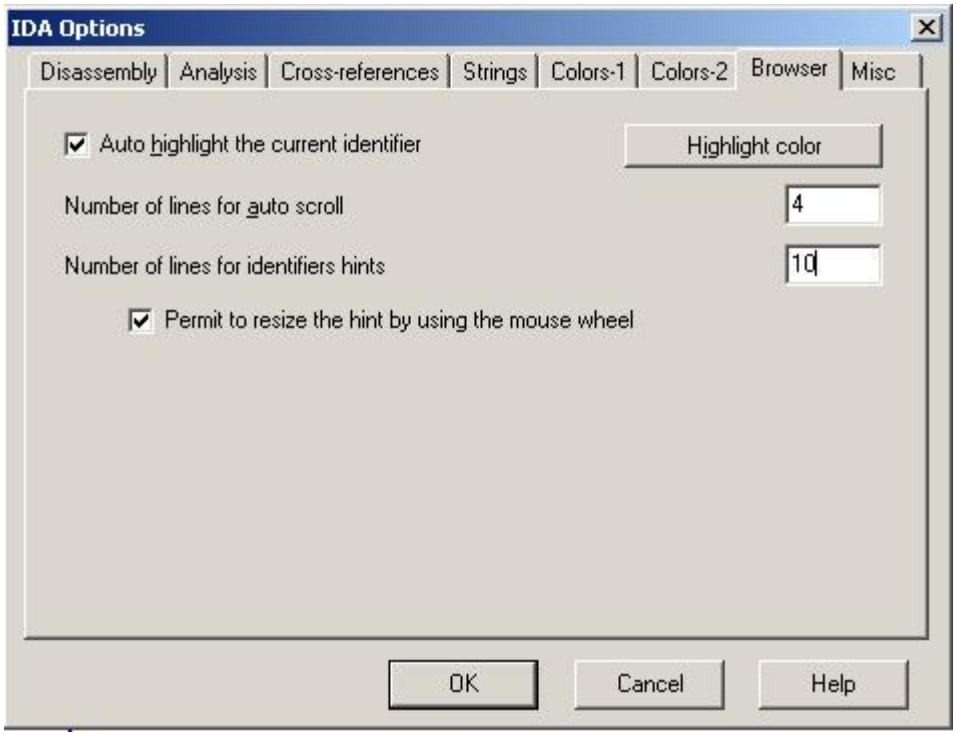
그리고 좌측 하단의 "Instruction indention, Comments indention" 값들을 증가시키면 메인 윈도우에서 해당 값들이 우측으로 이동하고, 감소시키면 좌측으로 이동합니다. 이런 값들을 조정시켜 자신의 해상도에 맞게 위치를 조정하면 됩니다.

이제 옵션 창 상단에서 "Disassembly -> Cross-references"로 이동하면 다음과 같은 창이 보여집니다.



가장 관심을 가져야 하는 항목은 "Number of displayed xrefs" 인데, 파트 7에서 상호 참조에 대하여 짧게나마 언급했었습니다. IDA는 기본적으로 2개의 레퍼런스를 보여주는데 만약 특정 함수가 3번 이상 호출된다면 이를 전부 보기 위해서는 해당 값을 변경해야 합니다. 물론 메인 윈도우 창에서 단축키를 통해 모든 레퍼런스를 볼 수도 있지만 해당 값을 적절히 변경하여 메인 윈도우에서 보는 것도 나쁘지 않은 선택입니다. 참고로 해당 값을 변경하면 반드시 설정 파일을 확인하여 제대로 저장되었는지 체크해야 합니다.

그리고 "Color-1 / Color-2" 메뉴에서는 사용자가 원하는 색으로 기존 값들을 변경할 수 있습니다. 꽤 간단한 작업이기 때문에 별도의 설명은 하지 않습니다. 색 변경 대신에 "Browser" 메뉴를 살펴 볼 것인데, 이 호출 명령어로 이동하는 것만으로도 호출 되는 코드를 볼 수 있게 해주는 꽤 유용한 기능입니다.



위 사진에서 유용한 세팅 값은 다음과 같습니다.

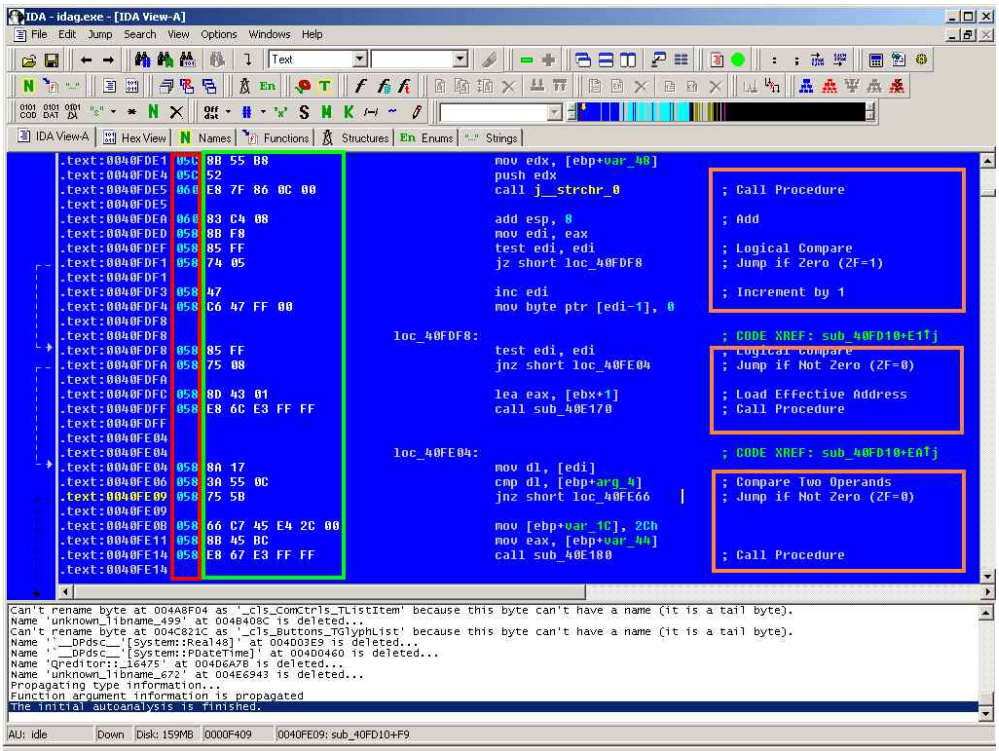
Auto highlight the current identifier : 체크

Number of liens for auto scroll : 4

Number of lines for identifiers hints : 30-40

Permit to resize the hiny by using the mouse wheel : 체크

꽤 많은 값들을 세팅했기 때문에 이제 뭐가 달라졌는지 볼 차례입니다. 이제 IDA의 메인 윈도우 창은 다음 사진과 비슷하게 나타나게 됩니다.



빨간색 사각형

해당 부분에서 스택 포인터를 볼 수 있습니다. "push" 명령어를 이용하여 스택에 어떤 값을 넣을 때 마다 해당 값은 4씩 증가하게 되며, "pop" 명령어를 이용하여 스택에서 값을 빼올 때마다 해당 값은 4씩 감소하게 됩니다. 이는 어떤 push 명령어들이 어떤 함수에서 사용되는지 알 수 있게 해줍니다.

주황색 사각형

파트 4에서 해당 영역 내에 존재하는 자동 주석을 변경하는 것에 대하여 다루었는데, 위 사진에서 해당 주석 각각을 볼 수 있습니다. 각각의 주석 라인들은 미리 정의된 주석들을 기반으로 자동으로 추가됩니다. 코드 블록 내에서 해당 값들이 실제로 어떻게 이용되는지 쉽게 알 수 있게 해주고, 뉴비분들은 이를 통해 더 이상 어셈블러가 마치 처음 봤을 때 암호처럼 느껴지지 않게 됩니다.

초록색 사각형

해당 영역은 명령어의 오피 코드를 의미합니다. 사실 좌측에 있는 명령어에 대한 16진수 값에 불과하기 때문에 중요하다고 볼 순 없습니다. 그럼에도 불구하고 굳이 해당 값을 보여주게 바꾼 이유는 가끔 대상 프로그램을 패치하고 싶을 때 IDA는 패치를 지원하지 않기 때문에 헥사 에디터를 이용해야 하는데 이 때 해당되는 16진수 값을 기준으로 검색하여 패치 해야 하기 때문입니다. 대부분의 기초적인 크랙 예제에서는 16진수 값을 74h->75h로 변경하는 것을 상세한 설명 없이 보여줄 텐데, 다음 2개의 라인은 이런 크랙에 대한 예제입니다.

.text:0040FDF1 058 74 05 jz short loc_40FDF8 ; Jump if Zero (ZF=1)

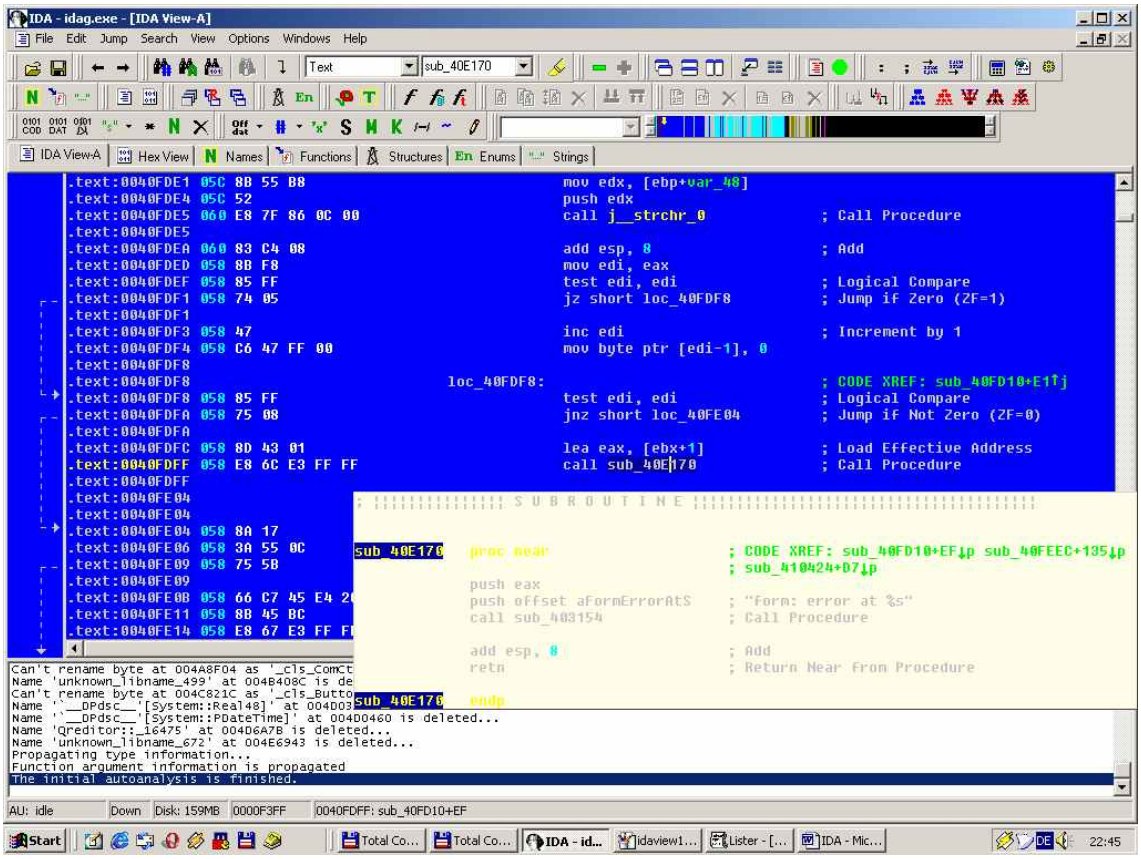
.text:0040FDFA 058 75 08 jnz short loc_40FE04 ; Jump if Not Zero (ZF=0)

058 로 시작하는 부분은 스택 포인터이기 때문에 무시하시길 바랍니다. 각각의 명령어들은 2바이트로 구성되어 있고, 74는 JZ(Jump if Zero) / 75는 JNZ(Jump if Not Zero)를 의미합니다. 따라서 가상 주소 "0040FDF1"에 대하여 74->75로 변경하는 것은 다음과 같은 코드로 변경됩니다.

.text:0040FDF1 058 75 05 jnz short loc_40FDF8 ; Jump if Not Zero(ZF=0)

탐색기 기능 (Browser Function)

call / jump를 클릭하는 것만으로 해당 부분에 위치하는 코드(이동하기 위해 더블 클릭)를 볼 수 있습니다. 해당되는 코드를 그냥 보는 것만으로도 좀 더 깊은 분석을 해야 하는지 / 아니면 그냥 보는 것만으로도 충분한지 결정할 수 있는 경우가 종종 존재하기 때문에 꽤 유용한 기능이라 말할 수 있습니다. (다음 페이지 사진 참고 바람)

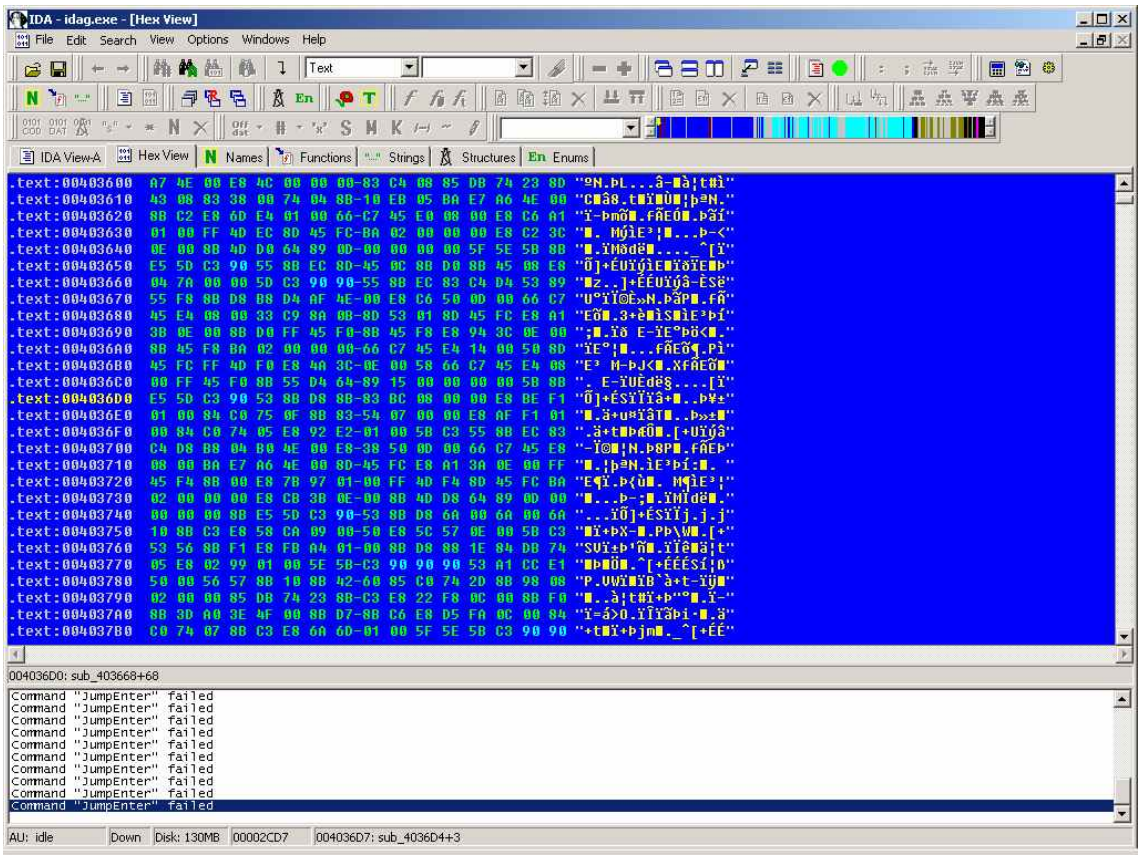


기본 블록 경계 (Basic Block Boundaries)

한눈에 이 세팅을 알아보는 것은 쉽지 않은 일인데 가독성을 높이기 위해 각각의 코드 라인 사이사이에 추가적인 공백이 존재합니다. 일반적인 경우에는 모든 코드 라인이 공백 없이 바로 하단에 위치하게 됩니다.

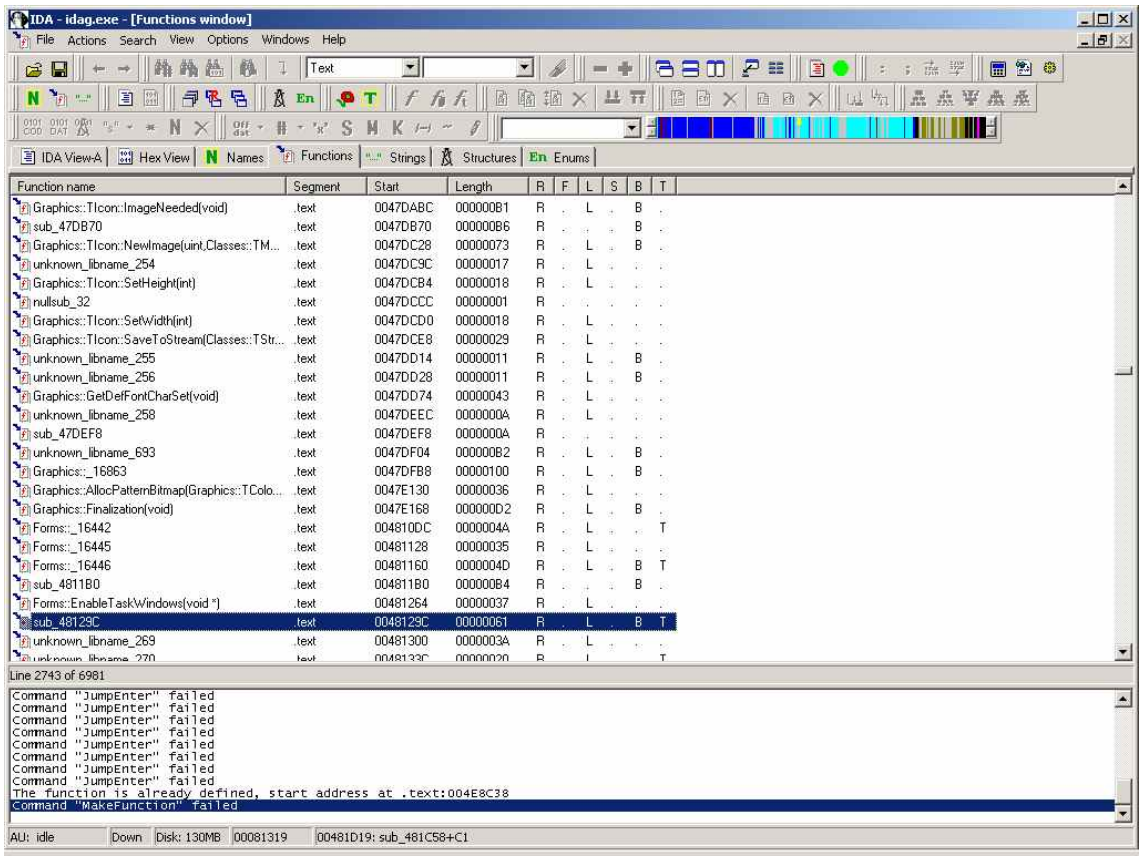
챕터 3

파트 10 – 헥스뷰



헥스뷰 윈도우는 그리 많은 설명이 필요 없다고 생각됩니다. 헥사값(16진수)을 클릭하면 메인 윈도우에서 해당되는 코드의 주소로 헥스뷰에서 이동됩니다. 참고로 상태 창에서 "Command 'JumpEnter' failed"가 나타나도 정상적으로 이동됩니다.

파트 11 - 함수

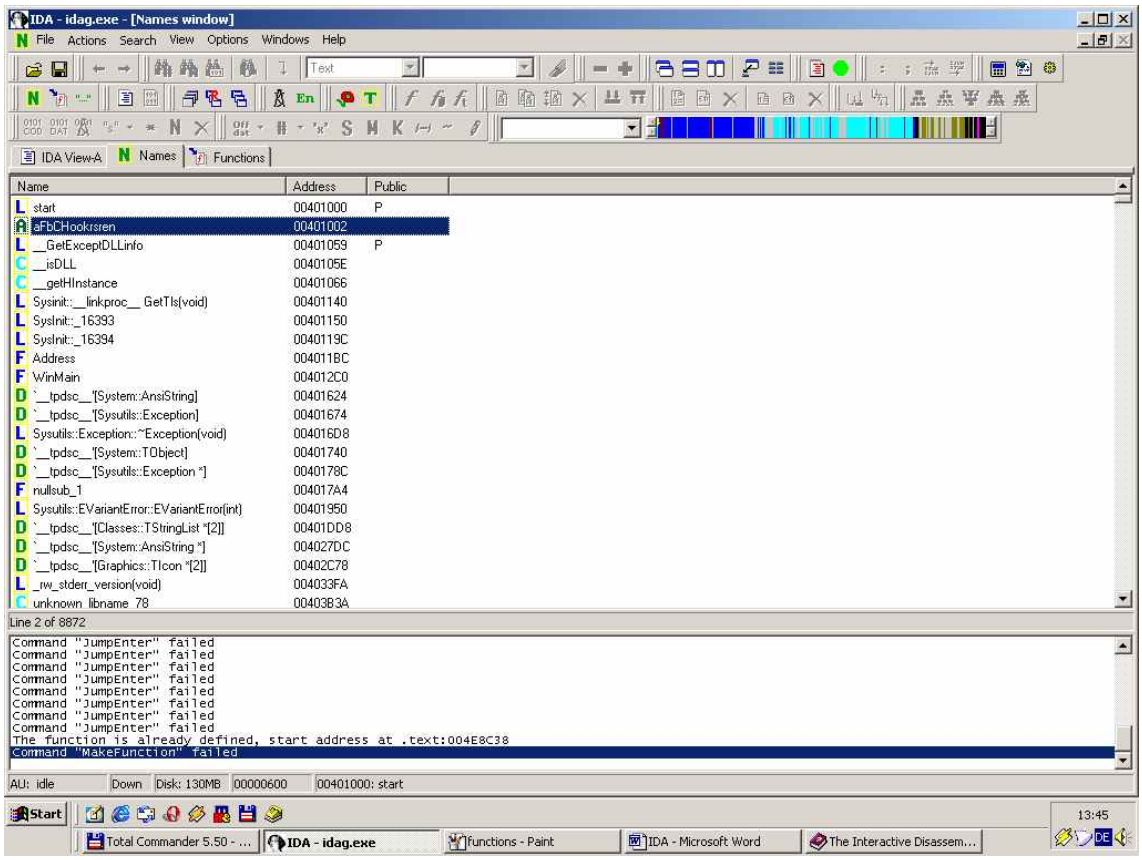


함수 창에서는 IDA에서 인식한 모든 함수 목록을 볼 수 있습니다. 함수 중 하나를 더블클릭 하거나 클릭 후 "Enter" 키를 누르면 메인 윈도우에서 해당 되는 코드로 즉시 이동됩니다. 사용할만한 또 다른 기능으로는 검색 기능인데, 검색하고자 하는 문자열을 입력하고 결과가 있으면 해당 되는 문자열 주소로 이동됩니다. 참고로 분석하는 동안 "Action" 메뉴를 주의 깊게 살펴보길 바랍니다.

함수 이름 앞쪽에 보면 어떤 글자들이 보이는데 다음과 같은 의미를 지닙니다.

R - 함수가 호출자(Caller) 주소로 리턴 됨
F - 주소 위치가 멀리 있는 함수
L - 라이브러리 함수
S - 정적(Static) 함수
B - BP 레지스터 기반 프레임. IDA는 자동적으로 모든 프레임 포인터를 [BP+xxx] 형태로 스택 변수로 변환함
T - 해당 함수에 타입 관련 정보가 있음

파트 12 – 이름 (Names)

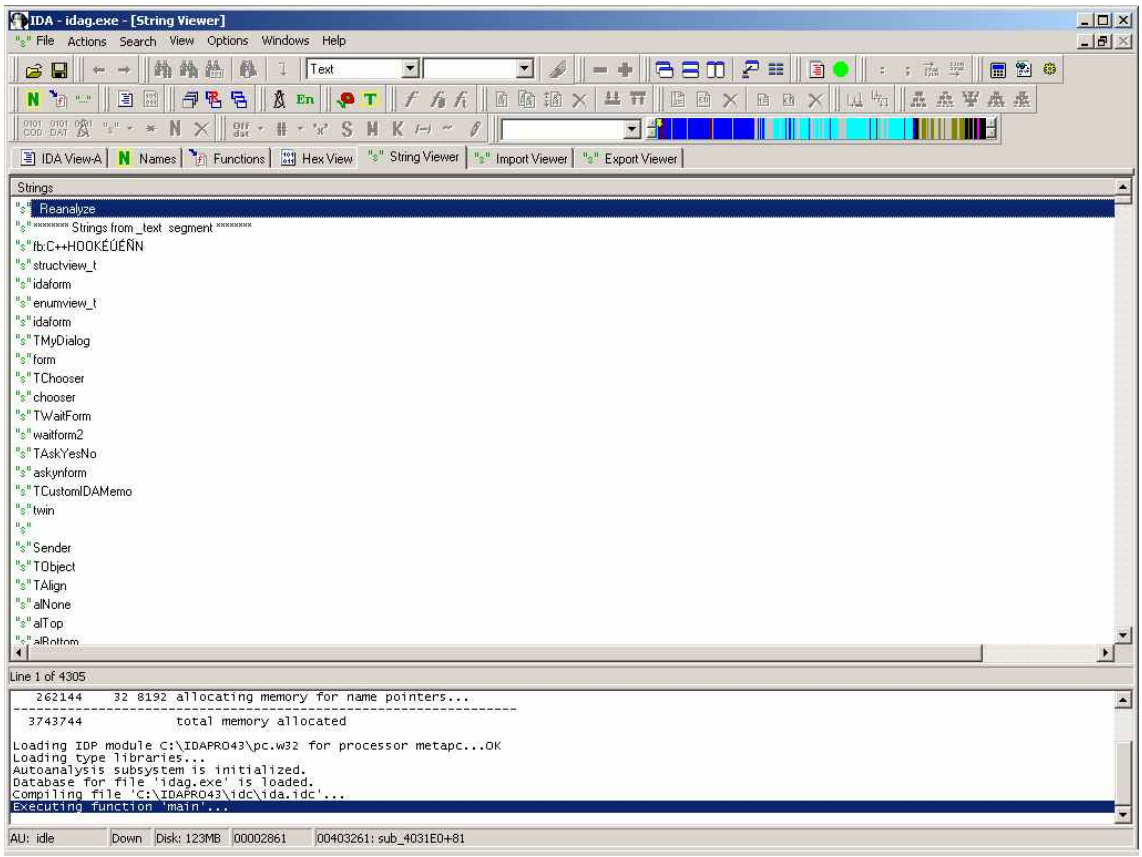


이름 창에서는 찾을 수 있는 모든 이름들을 보여줍니다. 이름 창 역시 검색 기능을 제공하여 더블 클릭하거나, "Enter" 키를 누르면 해당 되는 이름의 주소로 이동 됩니다. 이름 앞에 위치하고 있는 아이콘들은 다음과 같은 의미를 가집니다.

L **(남색)** - 라이브러리 함수
F **(남색)** - 정규 함수
C **(파란색)** - 명령어
A **(진녹색)** - ASCII 문자열
D **(연두색)** - 데이터
I **(보라색)** - 임포트 이름

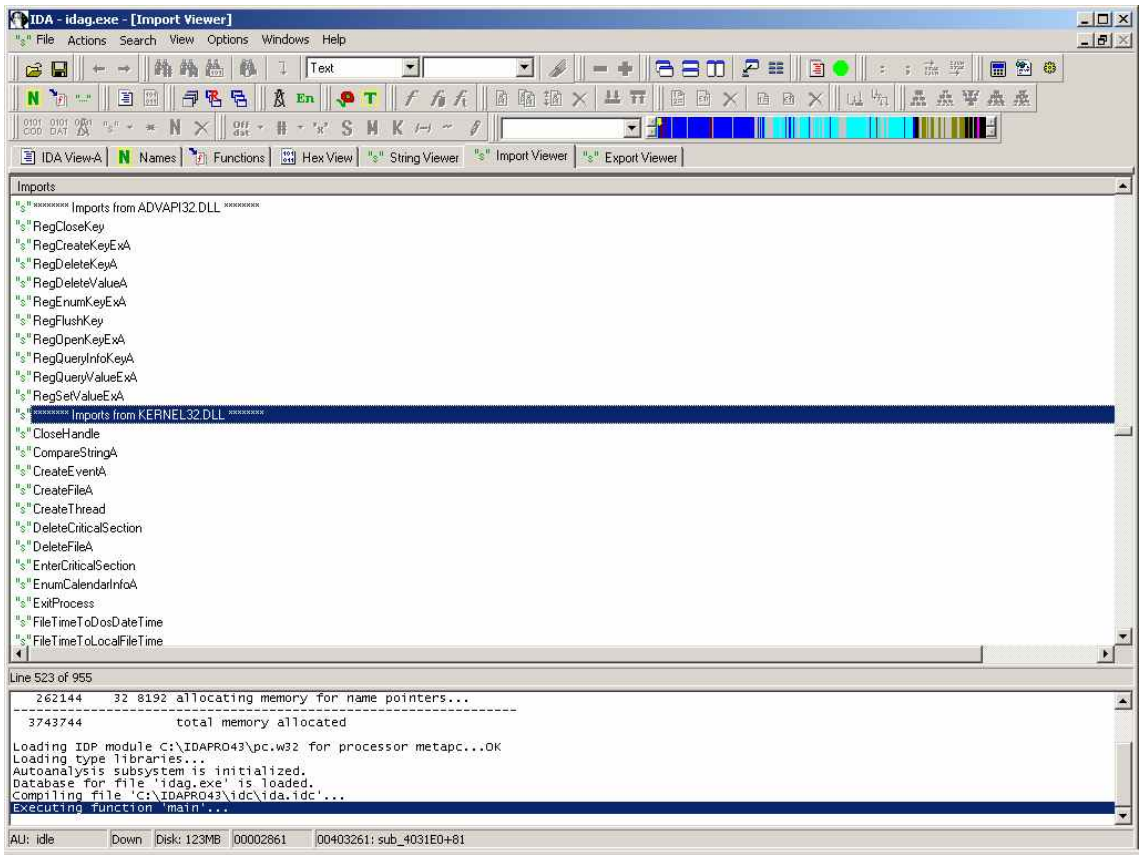
마찬가지로 "Action" 메뉴를 주의 깊게 살펴보길 바랍니다.

파트 13 – 문자열 뷰어



이전 문자열 창에서 검색 된 결과를 보여주는 일종의 검색 결과 화면입니다. “Demo”, “Shareware”, “Trial”, “Invalid registration key” 등과 같은 문자열을 찾고자 한다면 문자열 뷰어 창부터 분석을 시작하는 것이 좋습니다. 이전과 마찬가지로 더블클릭이나 “Enter” 키를 눌러 원하는 문자열의 주소로 이동할 수 있습니다.

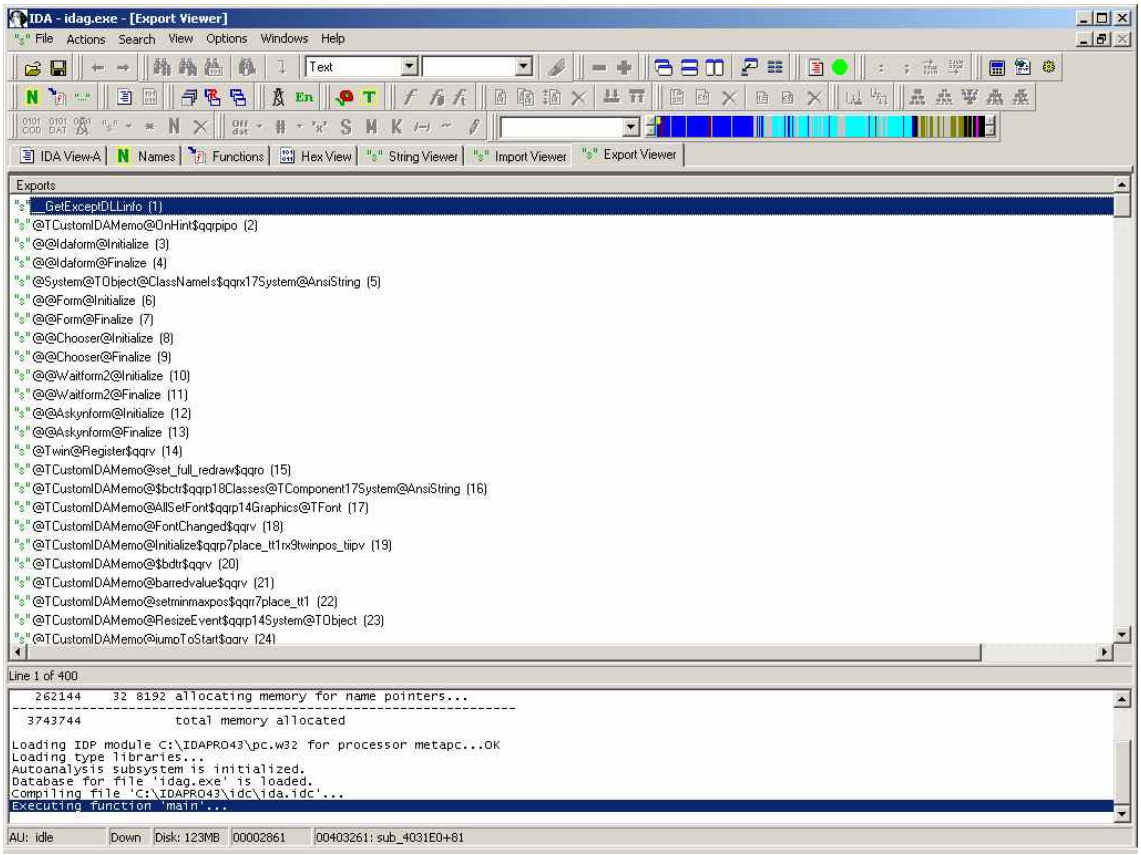
파트 14 – 임포트 뷰어



임포트 윈도우는 매우 중요합니다. 왜냐하면 해당 창에서 분석하고자 하는 프로그램이 다른 DLL 라이브러리로부터 사용하는 모든 함수를 볼 수 있기 때문이다. 알고 있는 것처럼 표시되는 함수를 다시 작성할 필요는 없습니다. 이는 마치 정렬 알고리즘을 사용하고자 할 때마다 다시 작성하지 않는 것과 동일한 방식입니다.(함수의 재사용에 대해 얘기하는 듯)

많은 함수들이 이미 DLL 파일에 존재하기 때문에 우리는 그냥 해당 함수를 사용하기만 하면 됩니다. 임포트 창은 우리에게 어떤 DLL 파일의 어떤 함수가 호출되는지 보여주는데 예를 들면 레지스트리를 읽고 쓰는 경우이다. 이런 방법은 정품 인증이 필요한 프로그램에서 시리얼 번호나 등록 키를 저장할 때 가끔씩 사용되고 파일을 읽고 쓰는 것이 키파일 루틴을 위해 자주 사용되는 방식입니다. 프로그램을 크랙할 임포트 윈도우를 이용하여 여러 가지 아이디어를 이용하여 프로그램 방어 기법을 공격하거나 기타 디버거에서 유용하게 사용할만한 브레이크 포인트를 찾을 수 있습니다. 다른 윈도우와 마찬가지로 원하는 문자열을 찾는 기능 역시 제공됩니다.

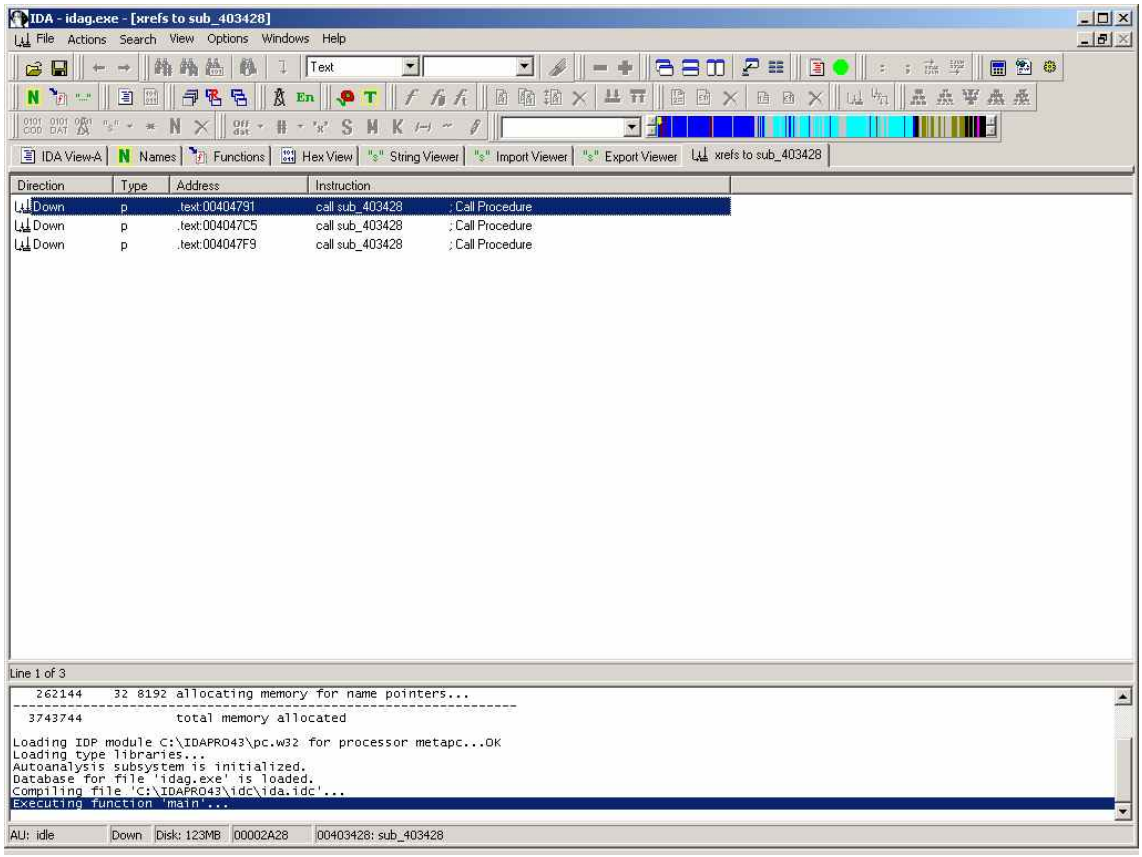
파트 15 – 익스포트 뷰어



이 창은 DLL 파일을 리버싱할 때 유용하게 사용되는데, 해당 창은 다른 프로그램에서 호출하고 사용하는 함수들을 보여주기 때문입니다.

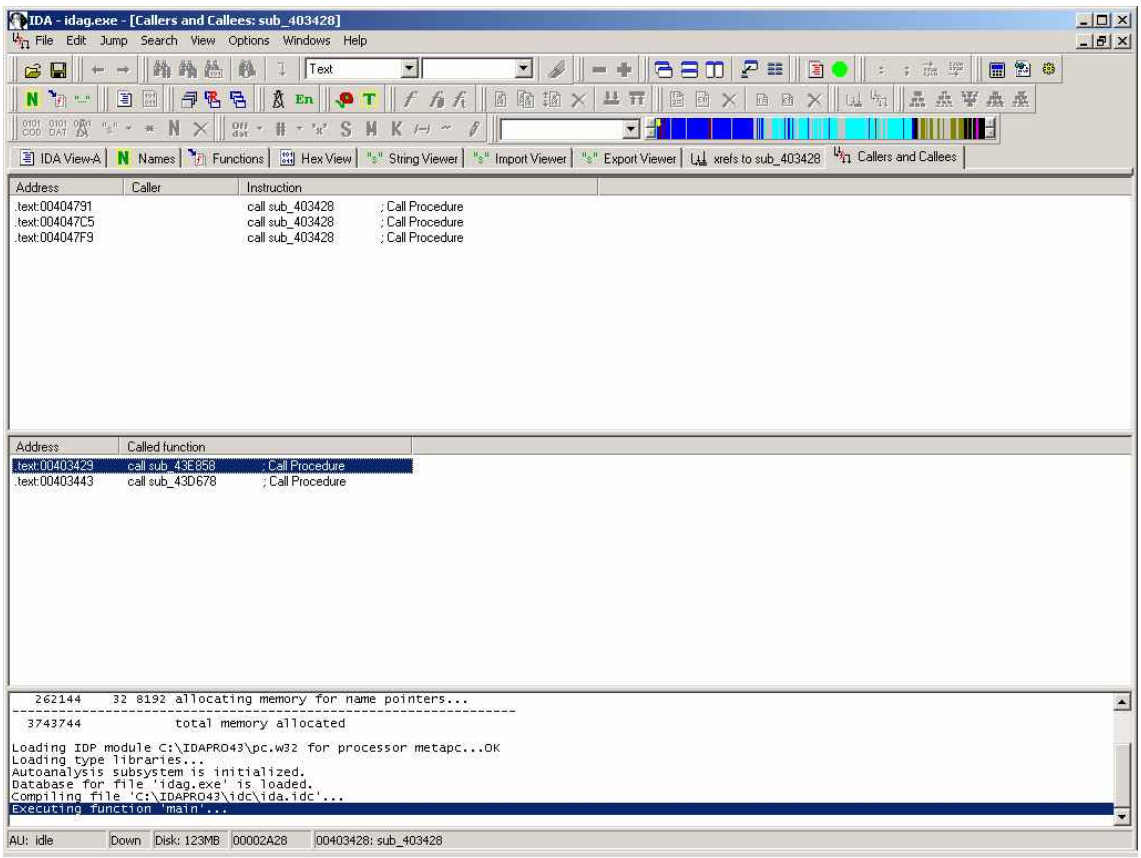
예를 들어, 어떤 소프트웨어 제작 회사에서 자체 방어 스키마를 가지고 있고 등록 관련 함수를 전부 DLL 파일로 매핑한 경우라면, 어떤 함수에서 해당 DLL이 제공되는지 보면 됩니다. 이 창 역시 검색 기능을 제공합니다.

파트 16 – 크로스 레퍼런스



이 윈도우는 함수의 모든 상호 참고 목록을 사용자가 생성한 윈도우 창에 보여줍니다. 다시 말하면 분석하고자 하는 함수가 호출되는 모든 주소가 나타난다는 뜻입니다. 본 창을 생성하려면 함수의 헤더에 커서를 놓고, "View / Open Subview / Cross-reference"를 클릭하면 해당 함수에 해당되는 "Cross-References" 창이 새롭게 생성됩니다. 생성된 창의 이름은 "xrefs to '함수 이름'" 와 같은 형태입니다. 일반적으론 메인 창에서 모든 상호 참고 목록을 볼 수 있으나 가끔 참조 목록이 3개 이상일 경우 메인 윈도우에 너무 많이 표시되는 것을 방지하기 위해 상호 참고 윈도우를 생성하는 것이 좋습니다. 목록에서 원하는 주소를 클릭하면 해당 코드 위치로 이동합니다.

파트 17 – 함수 호출



함수 창에서는 윗부분에 상호 참조를 보여주고, 추가적으로 특정 함수에 의해 호출되는 모든 함수를 보여줍니다. 함수의 전체적인 부분을 살펴보고 앞으로 얼마나 많은 함수를 분석해야하는지 알아보기 위해 사용하는 꽤 좋은 기능입니다.

예를 들어 분석하고자 하는 함수가 시리얼 체크 루틴이고, 해당 함수에서 처음 호출되는 함수가 사용자의 입력을 16진수로 바꾸는 것이고, 두 번째 함수가 시리얼이 올바른지 체크하는 것이라고 가정할 경우, 이 때 상호 참조 창은 시리얼 체크나 어디서 몇 번이나 수행되는지 알려줍니다. 상호 참조 창을 열기 위해서는 커서를 분석 함수의 첫 번째 코드 라인에 위치시키고 "View / Open Subview / Function Calls"를 클릭하면 됩니다.

챕터 4 – 코드 분석

파트 18 – 코드 앞쪽의 화살표

코드 앞쪽의 화살표들은 실행 흐름, 즉 다시 말하면 분기 명령어들을 의미합니다. 또한 화살표는 다양한 색을 가질 수 있는데 다음과 같은 의미를 가집니다.

빨간색
화살표의 출발지와 목적지가 같은 함수가 아니라는 것을 의미하는데 대부분의 경우 분기문들은 같은 함수 내에서 위치하기 때문에 빨간색은 누가 봐도 분기문의 출발지가 다른 함수라는 것으로 생각할 수 있습니다.

검정색
현재 선택된 화살표를 의미합니다. 선택을 하려면 "Up", "Down" 키나 화살표의 시작이나 끝부분을 클릭하면 됩니다. 참고로 "PageUp", "PageDown","Home","End" 키나 스크롤바를 이용하는 것은 선택 사항에 영향을 주지 않습니다. 따라서 이는 선택된 화살표를 변경하지 않고 해당 화살표를 좀 더 분석할 수 있게 해줍니다.

회색
선택 된 화살표 이외의 모든 화살표를 의미합니다. 이 때 화살표들의 두께는 다음과 같습니다.

두꺼움
이전으로 돌아가는 화살표로써, 대부분의 경우 반복문을 의미합니다.

얇음
현재 주소 이후로 분기되는 것을 의미합니다.

그리고 화살표들은 실선이거나 점선으로 이루어 질 수 있는데 점선 화살표는 조건 분기를 의미하고, 실선 화살표는 일반 분기를 의미합니다.

파트 19 – 분기문(JMP)과 호출(CALL)문 분석

이번 파트에서는 짧은 예제를 보여주는 것이 최고의 방법이라 생각합니다. 다음과 같은 주소를 분석하고 있다고 가정합니다.

.text:0040326F 414 0F 84 9D 00 00 00 jz loc_403312

"loc_403312"를 더블클릭하면 "00403312" 라인에 위치하고 있는 코드위치에 도달할 수 있습니다.

```
text:00403312                   loc_403312:
.text:00403312
.text:00403312  424 8B C3               mov eax, ebx
.text:00403314  424 81 C4 08 04 00 00   add esp, 408h
.text:0040331A  01C 5F                  pop edi
.text:0040331B  018 5E                  pop esi
.text:0040331C  014 5B                  pop ebx
.text:0040331D  010 C3                  retn
```

분기 명령어의 주소를 더블 클릭하면 분기가 되는 주소로 이동됩니다. 이와 유사한 방법을 호출 명령어("CALL")에도 사용할 수 있습니다. 예제는 다음과 같습니다.

```
.text:00403244  414 8B D7               mov edx, edi
.text:00403246  414 8B C6               mov eax, esi
.text:00403248  414 E8 2B B6 03 00      call sub_43E878
.text:0040324D  414 33 C9               xor ecx, ecx
```

현재 커서가 "00403248" 주소에 있다고 가정하고 "sub_43E878"을 클릭합니다. 그러면 다음과 같이 "43E878" 주소에 도달하게 됩니다.

```
.text:0043E878                  sub_43E878 proc.text:0043E878
.text:0043E878
.text:0043E878  000 53                  push ebx
.text:0043E879  004 8B D8               mov ebx, eax
.text:0043E87B  004 56                  push esi
```

* 참고로 코드의 가독성을 위해 주석과 상호 참조 목록을 지우지 않았습니다.

파트 20 – 전진/후진 (Forward/Backward) 화살표 사용

툴바에서 3~4번째 아이콘은 대부분의 2개의 화살표를 의미합니다. 하나는 좌측 방향(후진 화살표)을 가리키고, 나머지 하나는 우측 방향(전진 화살표)을 가리킵니다. 이 2개의 화살표를 이용하여 코드 창에서 앞/뒤로 이동할 수 있습니다. 파트 19에 존재하는 2개의 예제를 생각하면 될 듯합니다. 분기(JMP) 명령어를 실행한 뒤 다시 뒤로 가고 싶으면 후진 화살표(좌측 방향)를 클릭하면 됩니다. 그리고 다시 분기 명령어에 의해 분기되는 주소로 가고 싶으면 전진 화살표(우측 방향)를 클릭하면 됩니다. 마지막에 이동된 사항에 대해서는 상태 창에 저장되기 때문에 코드를 분석하는 것에 있어서 큰 도움이 됩니다.

파트 21 – 상호 참조 사용

이번에도 예제와 함께 설명하도록 하겠습니다. 다음과 같은 코드를 현재 분석하고 있다고 가정합니다.

```
.text:004559A9     loc_4559A9:                ; CODE XREF: sub_4557A8+A8   j
.text:004559A9                                ; sub_4557A8+11C   j
.text:004559A9                                ; sub_4557A8+1B7   j
.text:00455949
.text:004559A9     0CC 5F    pop edi
.text:004559AA     0C8 5E    pop esi
.text:004559AB     0C4 5B    pop ebx
```

위에서 보이는 주소 영역이 실행되지 않게 해야 하는 나쁜 영역(badguy-location)이라고 판단되었기 때문에 이 주소로 분기하는 모든 주소를 체크해야 하는 상황이라고 가정합니다. 이 때 "sub_455748+A8 j"를 더블클릭 하면 이 주소로 분기하는 1번째 명령어로 이동됩니다. 2번째 명령어는 "sub_4557A8+11C j"를 더블클릭하면 됩니다. 이런 식으로 하나하나 클릭하면서 확인하면 되고, 위 코드 영역의 상호 참조 주소들은 다음과 같다고 가정할 수 있습니다.

```
.text:00455850    0CC E9 54 01 00 00    jmp loc_4559A9          ; Jump
.text:004558C4    0CC E9 E0 00 00 00    jmp loc_4559A9          ; Jump
.text:0045595F    0CC EB 48             jmp short loc_4559A9    ; Jump
```

* 1. 455850 (4557A8 + A8) 2. 4558C4 (4557A8 + 11C)
* 3. 45595F (4557A8 + 1B7)

이번에는 주소 말고 함수로 분기되는 코드 참조 목록을 보도록 하겠습니다. 다음과 같은 코드가 있다고 가정합니다.

```
.text:0045565C    sub_45565C   proc near   ; CODE XREF: sub_455464+B1       p
.text:0045565C                                ; sub_455CAC+44          p
.text:0045565C                                ; sub_457784+110         p
.text:0045565C
=== some code ===
.text:004557A5    sub_45565C   endp
```

위 함수가 시리얼을 체크하는 함수라고 가정하고 어떤 주소에서 해당 함수를 호출해야 하는지 분석해야 한다면, "sub_45564+B1 p"를 하면 함수를 호출하는 1번째 주소로 이동하고, 이전과 마찬가지로 계속 클릭하면 함수를 호출하는 2번째, 3번째, n번째 주소로 이동합니다. 호출하는 주소는 다음과 같은 형태의 명령어들이라고 할 수 있습니다.

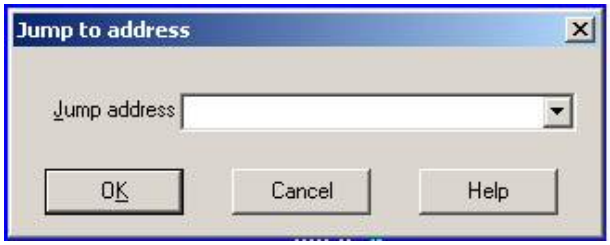
```
.text:00455515    0B0 E8 42 01 00 00      call sub_45565C          ; Call Procedure
.text:00455CF0    048 E8 67 F9 FF FF      call sub_45565C          ; Call Procedure
.text:00457894    064 E8 C3 DD FF FF      call sub_45565C          ; Call Procedure
```

눈치가 빠른 사람이라면 알겠지만 사실 파트 19와 크게 다를 게 없습니다. 그저 다른 방법으로 이동했을 뿐입니다.

파트 22 – 분기(점프) 메뉴

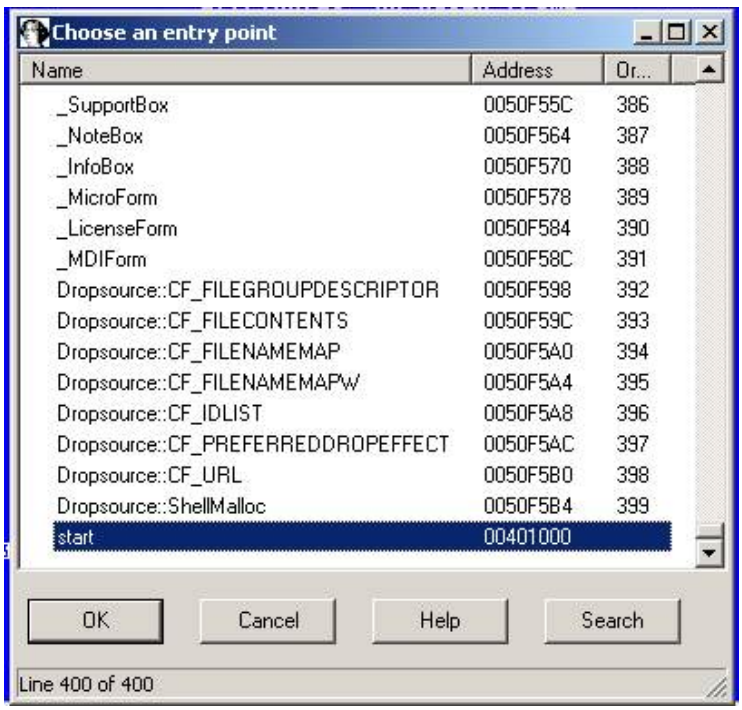
메뉴 이름 자체에서 해당 기능을 설명하고 있습니다만 꽤 쓸 만한 2개의 예제를 보여주고자 합니다. 참고로 다른 메뉴에 있는 단축키 또한 살펴보길 바랍니다. 꽤 유용하게 사용됩니다.

특정 주소로 점프 (단축키 : 'g')



이해하기 쉬운 꽤 직관적인 창인 것을 알 수 있습니다. 그냥 이동하고 싶은 주소를 입력하고 "OK" 버튼을 누르면 됩니다. 해당 창에서는 마지막으로 분기 했던 주소 기록 또한 제공합니다.

엔트리 포인트로 점프 (단축키 : 'Ctrl+E')



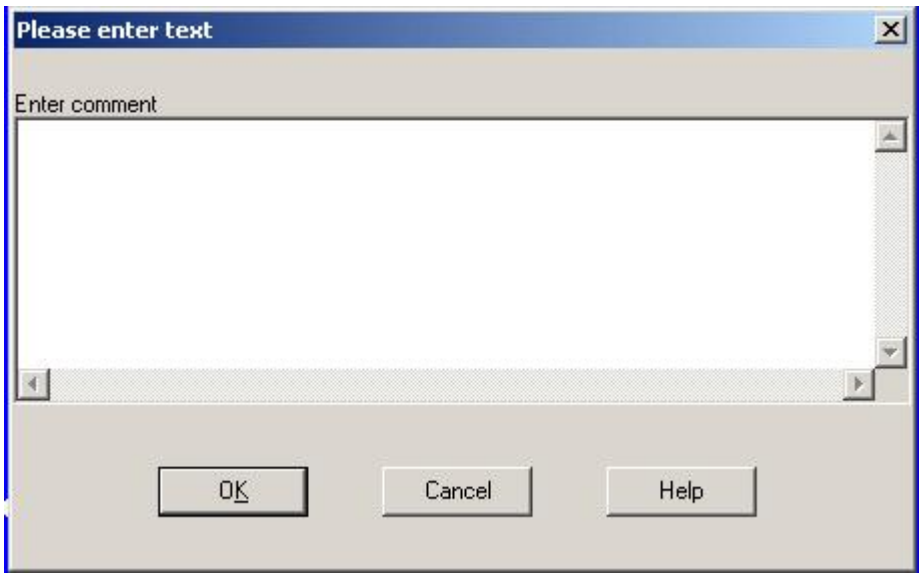
리스트에서 원하는 이름을 클릭하고 "OK" 버튼을 누르면 됩니다. "start" 엔트리 포인트는 대부분의 프로그램에서 시작점을 의미합니다. 분석 프로그램이 시작할 때 나타나는 Nag 창을 제거하기 위하여 주로 사용되는 부분입니다.

챕터 5 – 코드 가독성 높이기

파트 23 – 주석 추가

이미 분석을 했고 어느 정도 이해를 했던 부분에 대해서는 주석을 다는 것이 좋습니다. 안 그러면 했던 작업들 또 다시 하는 시간 낭비가 일어날 수 있기 때문입니다. 주석은 모든 코드 라인 각각에 대해서 추가할 수 있는 총 3가지 방법을 이용할 수 있습니다.

1. 툴바에 있는 ":" 아이콘을 이용합니다.
2. 주석을 원하는 코드 라인 끝 부분에 커서를 두고, 마우스 우측 키를 누른 다음 "Enter Comment"를 누릅니다.
3. 단축키 ":"를 이용합니다.



위와 같은 창에서 단일, 다중 라인의 주석을 입력할 수 있습니다. "OK" 버튼을 누르면 추가한 주석이 메인 윈도우 창에 나타납니다. 또한 "반복 주석 (repeatable comments)"를 입력할 수도 있습니다. 이 역시 3가지 방법이 존재합니다.

1. 툴바에 있는 ";" 아이콘을 이용합니다.
2. 주석을 원하는 코드 라인 끝 부분에 커서를 두고, 마우스 우측 키를 누른 다음 "Enter repeatable Comment"를 누릅니다.
3. 단축키 ";"를 이용합니다.

참고로 반복 주석은 몇몇 경우에 대해서만 유용합니다.

파트 24 – 라인 추가

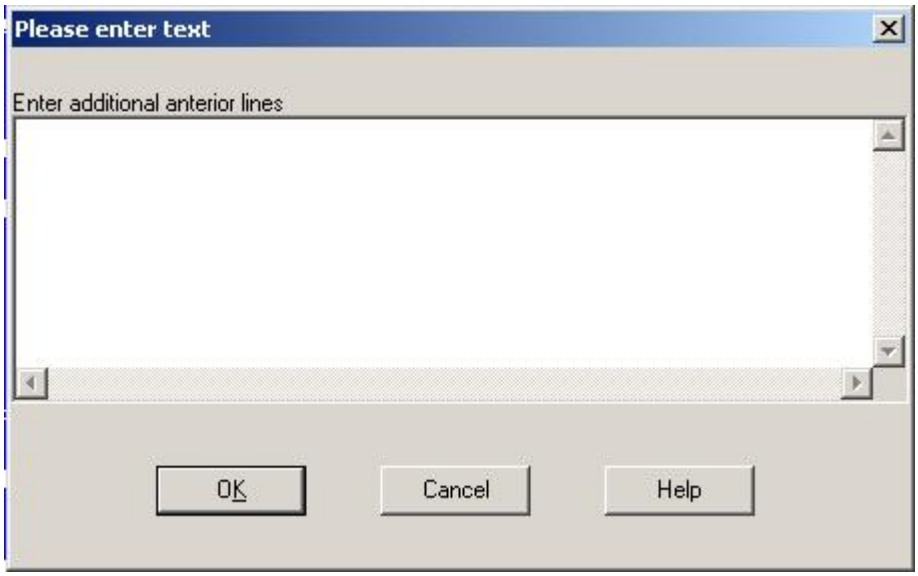
코드 사이사이에 주석을 포함하는 라인을 추가할 수 있습니다. 추가를 위해서 2가지 방법을 이용할 수 있습니다.

1. 현재 코드 주소 이전에 라인 추가 (추가적인 앞(anterior) 라인 추가)
2. 현재 코드 주소 이후에 라인 추가 (추가적인 뒤(posterior) 라인 추가)

위 옵션들은 다음과 같은 방법을 이용할 수 있습니다.

1. 단축키 "INS", "Shift+INS"를 이용합니다.
2. 툴바의 "", ";" 다음에 위치하는 아이콘을 이용합니다.

주석을 입력할 수 있는 창은 다음과 같은 형태를 가집니다. 선택한 옵션에 따라 "anterior" 이나 "posterior" 텍스트가 나타납니다.

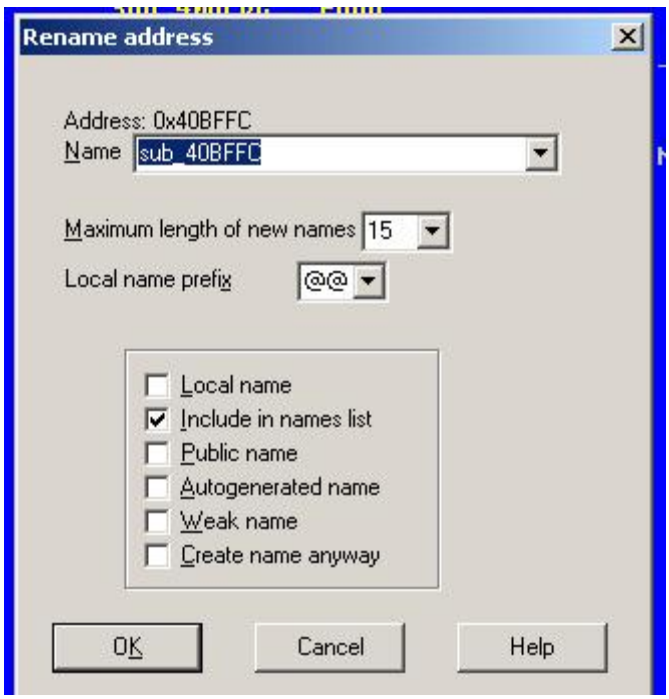


파트 25 – 함수와 주소 이름 변경

처음엔 아마도 이름을 변경하는 것이 쓸모없다고 생각할 수 있습니다. 그러나 특정 함수의 기능을 알고 있거나, 특정 주소가 루프인지 변수인지 알고 있다면 이름을 바꾸는 것만으로도 분석에 상당한 도움을 줄 수 있습니다.

함수 이름 변경

함수의 헤더에 커서를 놓고, 마우스 우측 클릭을 한 다음 "Rename"을 클릭합니다. 그러면 다음과 같은 창이 팝업 됩니다.



새로운 이름을 입력하고 "OK" 버튼을 누르면 해당되는 함수 이름 변경 작업이 완료됩니다. 이제 메인 윈도우에서 해당 함수는 새로운 이름으로 나타납니다.
e.g. call sub_40BFFC to call SerialCheck.

*참고로 Ida2Softice 플러그인을 이용하여 NMS 파일을 생성한 뒤 함수 이름을 변경하면, 이전 함수 이름으로 나타납니다.

주소 이름 변경

원하는 주소에 커서를 놓은 뒤 마우스 우측을 클릭하고 "Rename" 버튼을 클릭합니다. 그럼 다음과 같은 창이 나타납니다.

해당 주소에 대한 새로운 이름을 입력하고 "OK" 버튼을 누르면 변경 작업이 완료됩니다.
e.g. jnz loc_40C0B3 to jnz myname

IDA에서는 함수, 주소 이외에도 많은 것들에 대하여 이름을 변경할 수 있습니다. 위 사진에서 보이는 나머지 옵션 또한 코드의 가독성을 높이는데 큰 도움이 됩니다.