::IDA pro v6.1 사용법::

부제: 지뢰찾기 리버싱을 통한 IDA 사용법 익히기 + α

2012/12 초
Anch0vy
blog.naver.com/aaaa875 & anch0vy.tistory.com
Email: aaaa875@naver.com

이 문서는 리버싱과 프로그램 동작 원리에 대해 기초지식을 가지고 있는 리버서를 대상으로 합니다. 리버싱을 처음 배우기 시작하면서 주 툴을 IDA로 쓰시려는 분에게는 이 문서가 이해가 안 될 수도 있습니다.

IDA는 매우 좋은 '디스어셈블러' 입니다. 하지만 IDA 사용법에 대해 구글링을 해보면 한글은 물론 영어로 된 자세한 설명 자료는 별로 없다는 것을 알 수 있습니다. (어쩌면 필자의 구글링 실력이 부족한 것일 수도 있지만;;) 그래서 이 주제를 잡고 직접 문서를 써보게 되었습니다.
앞으로 IDA로 지뢰찾기를 PDB 파일 없이 리버싱하면서 IDA의 여러 기능과 + (이러저러한 리버싱 팁들)를 설명할 것입니다.
이 문서는 총 3개의 장으로 나누어져 있습니다.
첫 번째 장에서는 IDA의 자주 쓰이는 기능을 중점으로 지뢰찾기를 분석합니다. IDA 사용설명이 주목적이기 때문에 이 장에서는 지뢰찾기를 모두 분석하지는 않습니다.
두 번째 장에서는 첫 번째 장에서 다하지 못한 지뢰찾기 분석을 마저 합니다.
세 번째 장에서는 다른 예제 프로그램을 가져와서 특수한 경우에 대해 설명을 합니다. 또 첫째 장에서 설명하지 못한 기능들도 설명합니다.
그리고 지뢰찾기를 분석하면서 Resource Hacker 나 CheatEngn 같은 외부 프로그램을 쓸 것이지만 자세한 설명은 하지는 안했습니다.

분석에 사용된 프로그램들은 http://anch0vy.tistory.com/12 에서 받을 수 있습니다. (지뢰찾기 포함

사소한것
OK버튼 대신 컨트롤 엔터
드래그 마우스 휠로 하기

-IDA란?
IDA는 Interactive Disassembler 의 약자로 디스어셈블러입니다. 따라서 주 기능역시 디스어셈블입니다. 반대로 말하면 디버깅 기능은 불편하고 약합니다.(최신 버전은 많이 나아졌지만...) 이걸 보시고 디스어셈블러랑 디버거이랑 비슷한 게 아니냐, 생각하시는 분은 정적분석과 동적 분석의 차이점을 떠올려 보시면 됩니다.
이러한 특성은 지뢰찾기를 리버싱 하면서 하나씩 설명하겠습니다.

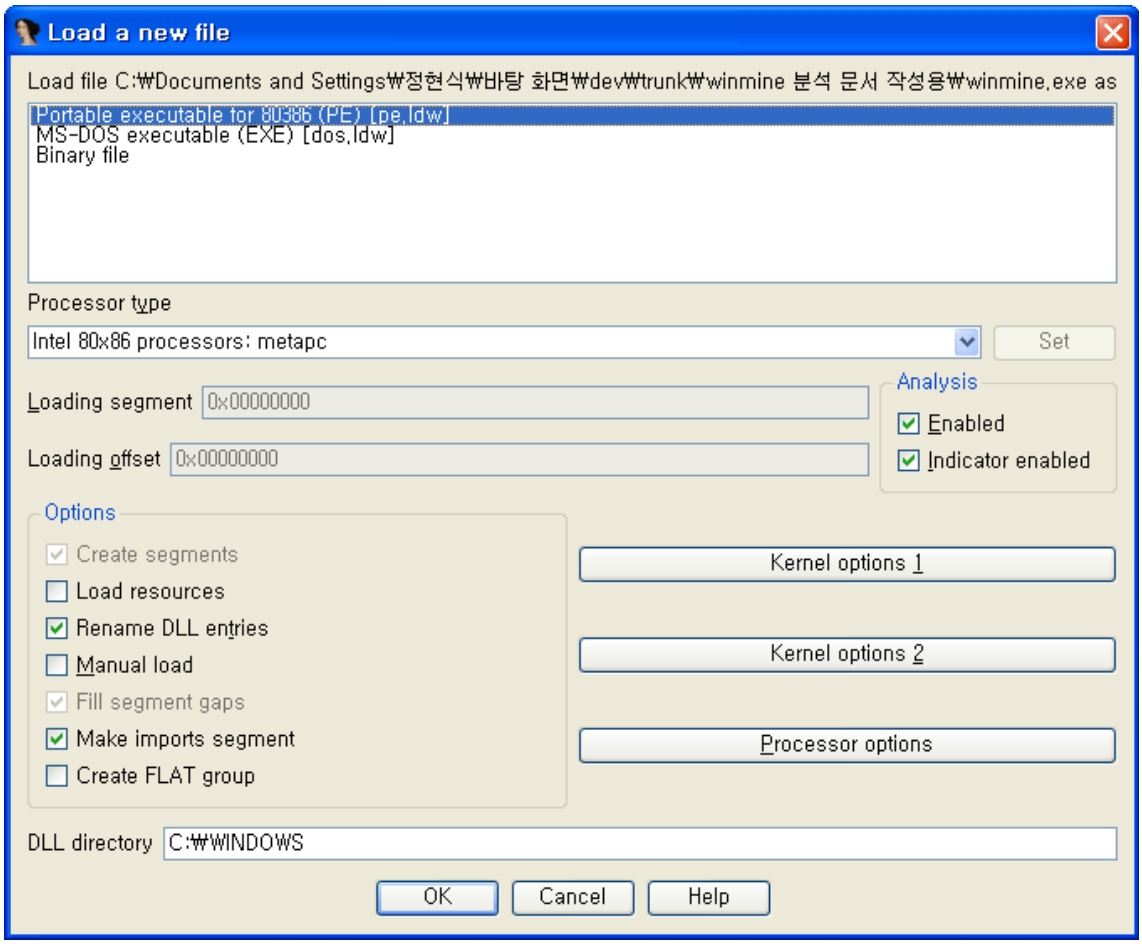
미리 알아둬야 할 IDA의 중요한 특징
-IDA에서 작업할 때는 원본 실행파일이 아닌 idb라는 데이터베이스에 작업을 하게 됩니다. 다시 말해 IDA에서 무슨짓을 하던 간에 원본 실행파일에는 변화가 없습니다. 또 원본파일이 없더라도 idb파일만 있으면 분석을 할 수 있습니다.
-IDA에는 되돌리기 기능이 없습니다. CTRL + Z는 전혀 다른 단축키이며 어떤 메뉴에서도 되돌리기 기능을 찾아볼 수 없습니다. 만약 무언가 큰 실수를 해서 작업한 것을 되돌리려면 IDA를 끌 때 DON'T SAVE the database 에 체크를 해야 합니다.(아래에 자세한 설명이 나옴)
-IDA는 많은 키에 단축키를 할당합니다. 그러므로 아무 생각 없이 키보드 키를 눌렀다간 매우 난처한 상황이 발생할 수 있습니다.

문서를 읽을때 유의할 점
-필자의 생각으로 단축키를 쓰기보단 직접 메뉴를 열어서 하는 게 낫다고 생각되는 경우는 단축키가 뭔지 쓰지 않았습니다. 반대로 말하면 이 문서에 나와 있는 단축키들은 매우 유용하고 자주 쓰이는 것이므로 외워두면 좋습니다.

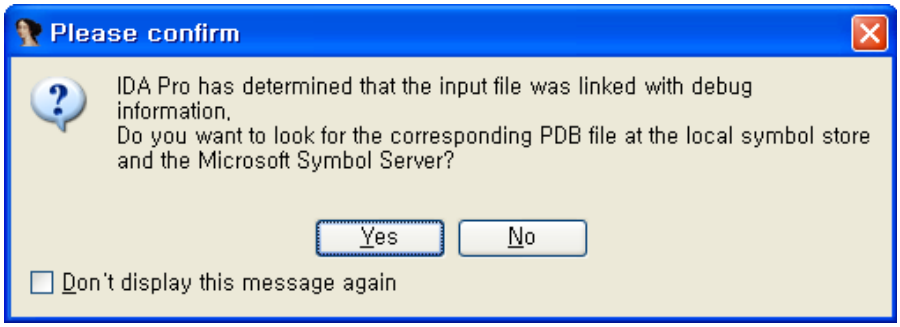
Chapter 1

지뢰찾기 분석을 통한 IDA 기본 사용법 익히기

IDA로 지뢰찾기 프로그램을 열면 다음과 같은 창이 뜹니다.



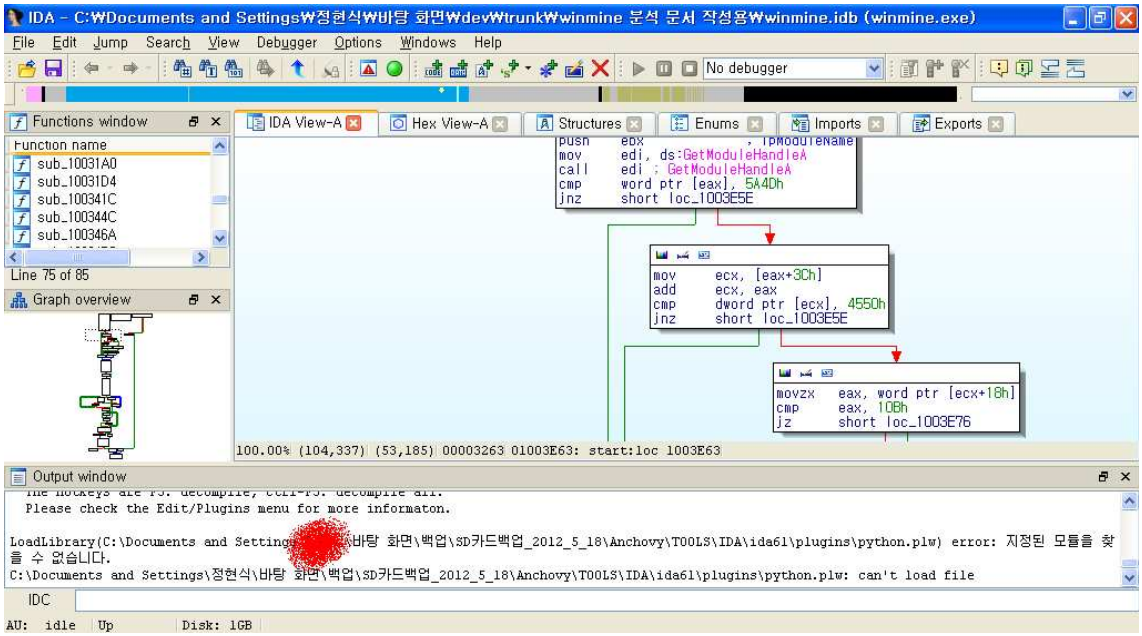
처음에 지뢰찾기를 열면 나오는 창입니다.
파일을 열 때 각종 설정들을 하는 창인데 일반적인 PE파일이나 ELF 파일을 분석하는 경우 대부분 이 창을 건드릴 일이 없습니다. 다만 맨 위에 리스트들은 파일의 종류를 설정하는 창인데 실제 파일의 종류와 IDA가 찾은 파일의 종류가 다를 경우 리버서가 직접 지정해줄 수 있습니다.
OK버튼을 누르면 IDA가 자동 분석을 시작하고 여러 가지 일을 하게 됩니다. 지뢰찾기의 경우 다음과 같은 창이 먼저 뜨게 됩니다.



PDB 파일을 로드할지 묻는 창인데 여기서는 PDB파일의 도움을 받지 않을 것이므로 로드하지 않겠습니다.
로드를 하면서 여러 가지 정보들이 하단의 output window 창에 뜨게 됩니다. 에러가 나거나 플러그인을 쓰는 경우를 제외하고는 이 창을 볼일은 별로 없습니다.

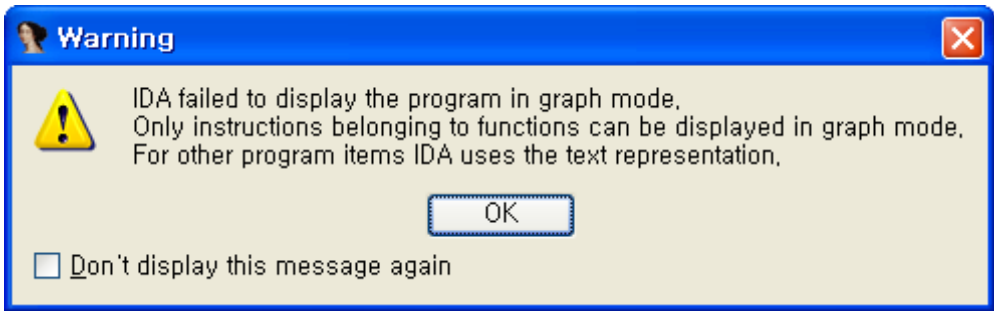
IDA가 분석하는 동안 여러 가지 IDA만의 분석기법을 적용하면서 동그란 아이콘()이 노란색이 되어있습니다. 이때도 코드를 브라우징할 수 있지만 아직 다 분석을 하지 않았기 때문에 코드에 수정(변수명 변경이라든지)을 하는 것은 좋지 않습니다.

다음과 같이() 아이콘이 초록색으로 바뀌면 그때부터 분석을 하는 것이 좋습니다.
참고로 프로그램이 복잡하거나 크기가 클 경우 IDA가 프로그램을 분석하는데 걸리는 시간은 생각보다 길 수 있습니다.(1분 이상까지)
분석이 다 끝나면 다음과 같은 창이 뜨고 실행파일이 있는 폴더에 id0 id1 nam til 파일이 생깁니다.



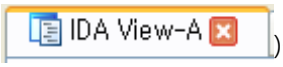
가장 큰 영역을 차지하는 창은 IDA view 창으로 IDA의 제일 핵심적인 부분입니다. IDA의 큰 장점중 하나인 그래프 뷰나 텍스트 뷰 디스어셈블 화면을 볼 수 있는 화면입니다.
텍스트 뷰 화면을 보게 되는 경우는 그래프 뷰를 보는 것이 불가능하거나 데이터 부분을 볼 때입니다.
그래프 뷰와 텍스트 뷰를 전환하는 방법은 오른쪽 버튼을 눌러서 graph view 나 text view

를 선택하거나 간단하게 스페이스바를 누르면 됩니다.
만약 그래프 뷰가 불가능한 부분이라면 다음과 같은 창이 뜨게 됩니다.

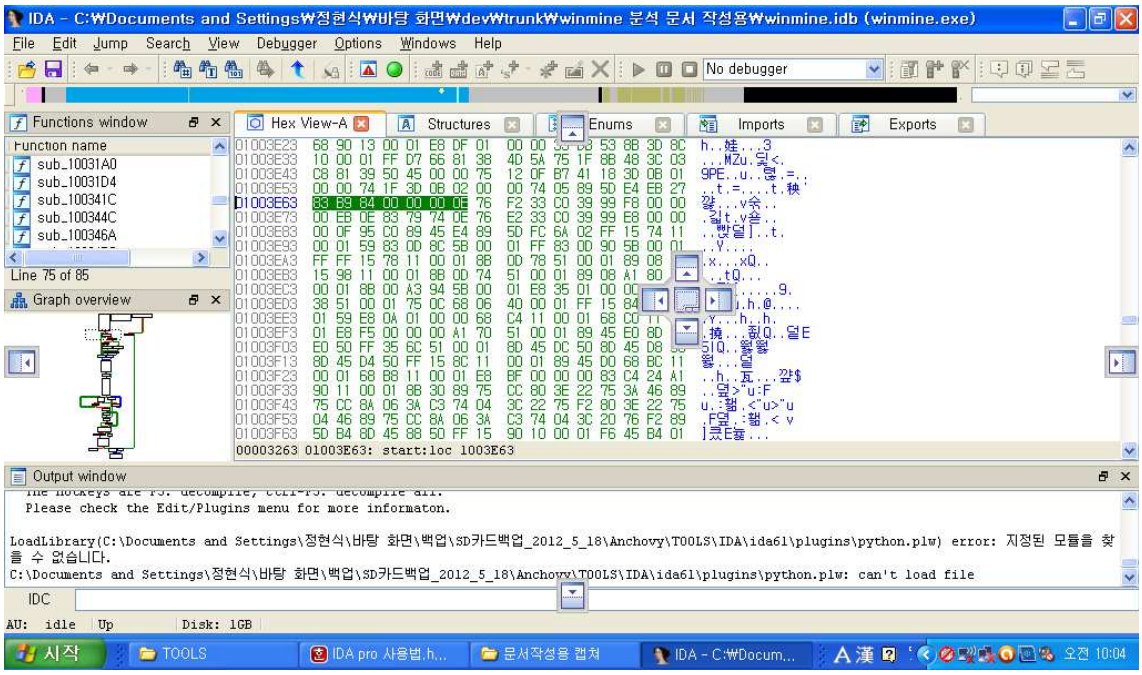


모니터 매우 큰 사람은 IDA의 기본 창 모양을 그대로 써도 좋지만 대부분의 경우는 3개의 창과 디스어셈블 창을 띄우기에는 모니터 공간이 부족합니다. 이럴 땐 화면 배치를 바꿔주면 됩니다. 화면 배치를 바꾸는 법은 간단합니다.

각 창의 상단을(

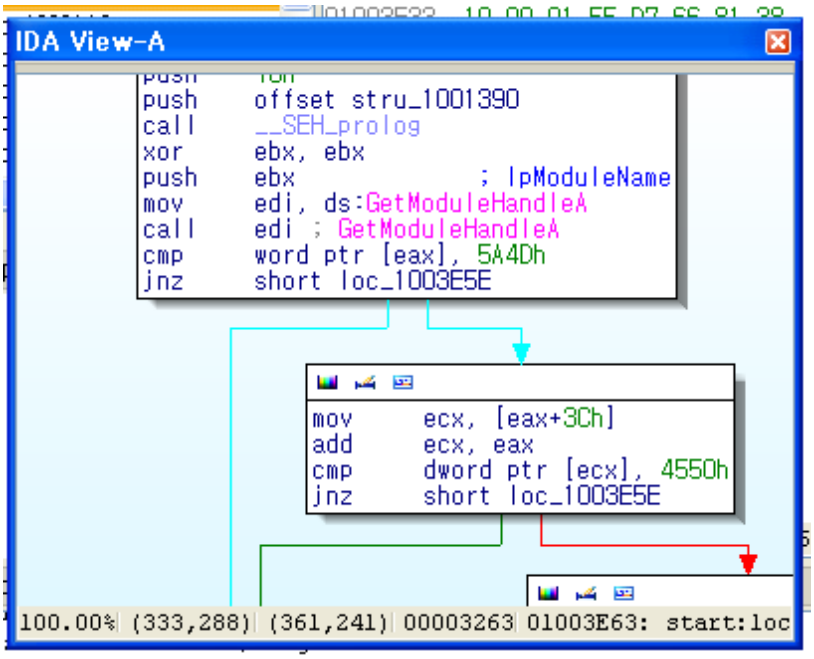
, 

)을 잡고 드래그 해서 옮기면 됩니다.



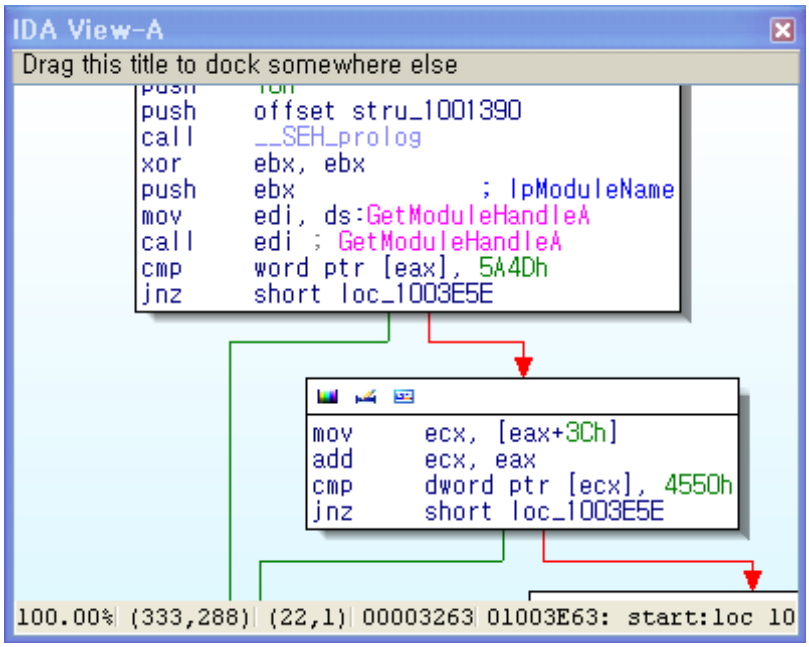
드래그를 하면 이렇게 창에 몇 가지 그림이 뜨는데 직접 옮겨보면서 어떻게 화면이 바뀌는지 직접 해보는 게 가장 좋습니다.

만약 저기 뜬 그림(네모난 칸)이 아닌 IDA 위 아무 화면으로 창을 옮기면 프로그램과 독립된 새 창이 뜹니다.

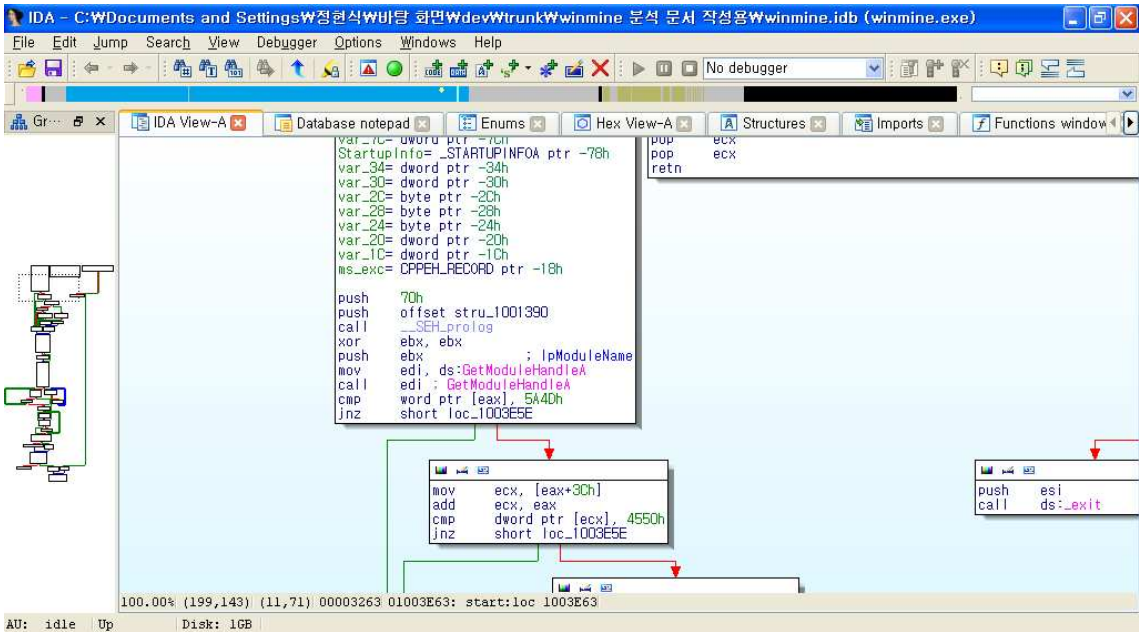


이 창은 크기조절이 가능하지만 최대화는 될 때가 있고 안 될 때가 있습니다.(버그) 여러 대의 모니터를 쓰는데 이창만 따로 띄우고 최대화를 하고 싶은데 안 된다면 ResizeEnable 란 프로그램을 사용하면 어느 정도 해결이 됩니다.

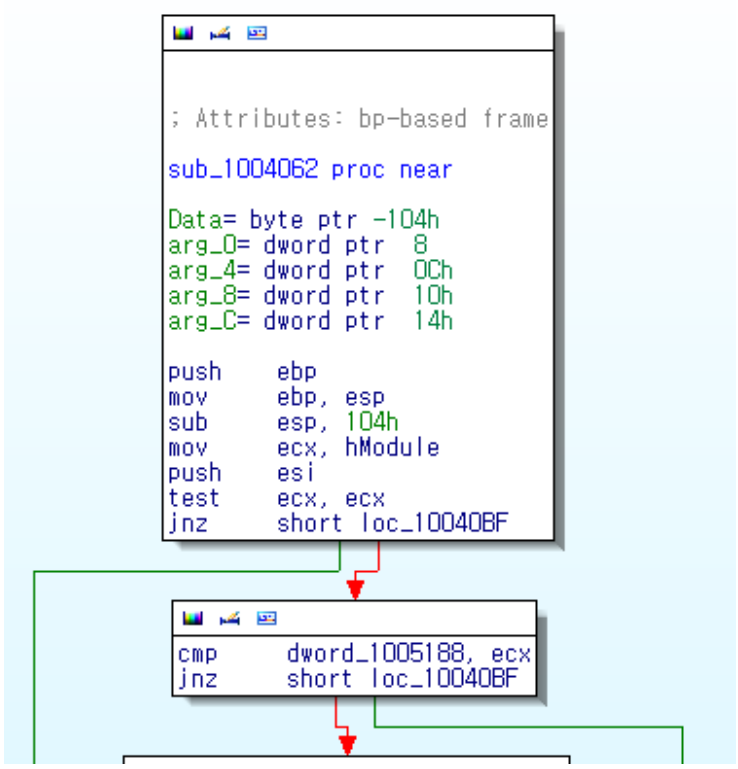
그리고 이 창을 다시 프로그램 내부 독(dock)으로 옮기고 싶으면 상단의 회색 선으로 마우스를 가져가면 드래그 할 수 있는 부분이 생깁니다.



창들을 이리저리 배치해서 마음에 드는 창으로 세팅하면 메뉴의 Windows - Save desktop 으로 저장하면 되는데 이때 Default에 체크하면 IDA를 껐다 켜도 계속 이 창을 유지하게 됩니다.
참고로 제가 쓰는 IDA 화면 배치는 다음과 같습니다.



그리고 좌측의 그레프 오버 뷰 창을 실수로 껏을때는 메뉴-view-graph over view 로 다시 키면 됩니다.

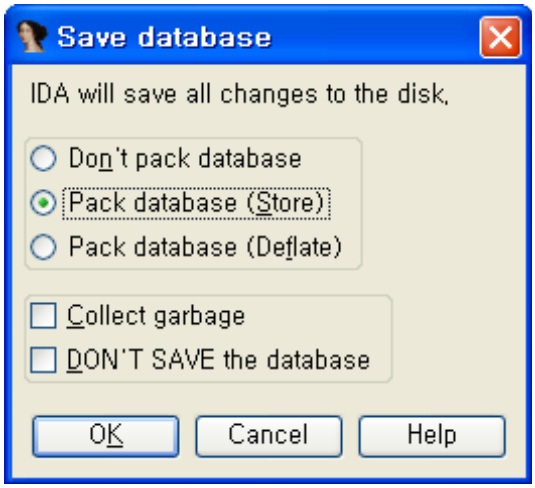


call 명령어를 제외한 프로그램의 흐름이 바뀌는 명령어(jxx)를 기준으로 코드가 나누어져 있

습니다. 분기에서 참인 경우 초록색 화살표, 거짓일 경우 빨간색 화살표가 가리키는 방향으로 프로그램 흐름이 이동합니다. 그리고 파란색 선은 무조건 점프하는 경우를 뜻합니다.
이 선들을 더블클릭하면 현재 화면의 위치에 따라 화살표가 시작하는 곳과 끝나는 곳으로 화면이 바뀝니다.
그래프 뷰 내에서 이동할땐 화면의 빈 공간을 누르고 움직이거나 SHIFT 키를 누르고 아무대나 누른 상태(코드가 화면을 가득 채울때 유용)로 이동하면 됩니다. 확대와 축소는 CTRL + 마우스 휠, 또는 CTRL +,- 로 할 수 있습니다.
그래프 뷰는 텍스트 뷰와 달리 주소값이 기본으로 나와있지 않습니다. 필요에 따라 주소값을 봐야 할 때가 있는데 이때는 메뉴 - Options - General - Disassembly 탭에서 Line prefixes 를 체크하면 보입니다.

상단의 IDA View-A , Hex View-A , Enums .... 이 보입니다. 이들은 마우스로 직접 클릭해서 이동해줄수 있지만 Ctrl + tab(윈도우의 alt 탭 과 비슷한 기능)으로 이동하거나 Ctrl + 1 로 이동할 수 있습니다. 둘의 차이점이 있는데 전자는 현재 떠있는 창만 이동이 가능하고 후자는 꺼져있는 창도 리스트에 나옵니다. 또 IDA View 나 Hex view 를 선택했을때 열려있는 창이 아닌 새로운 창을 띄웁니다.

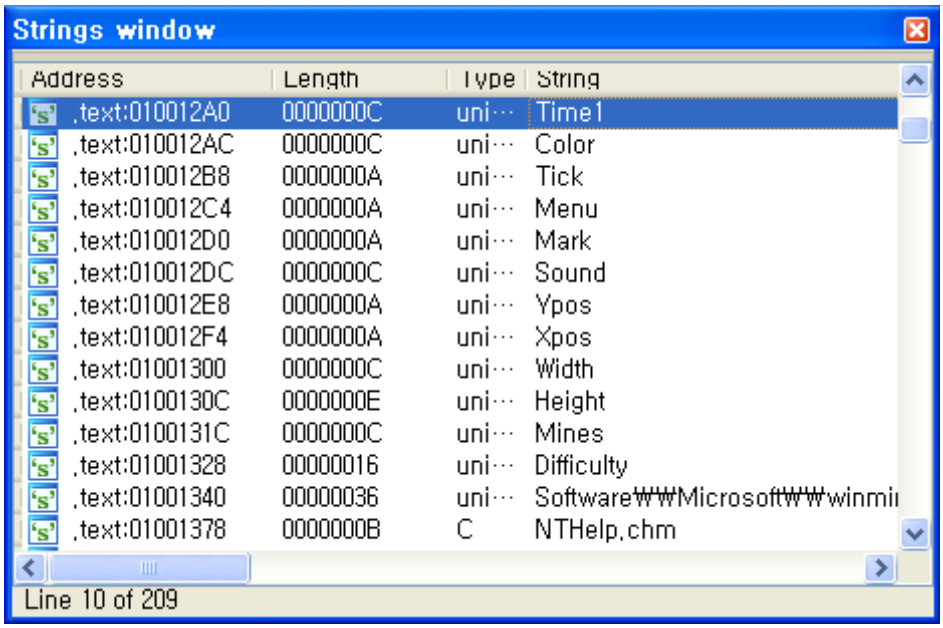
IDA를 끌 때에는 이런 화면이 나옵니다.



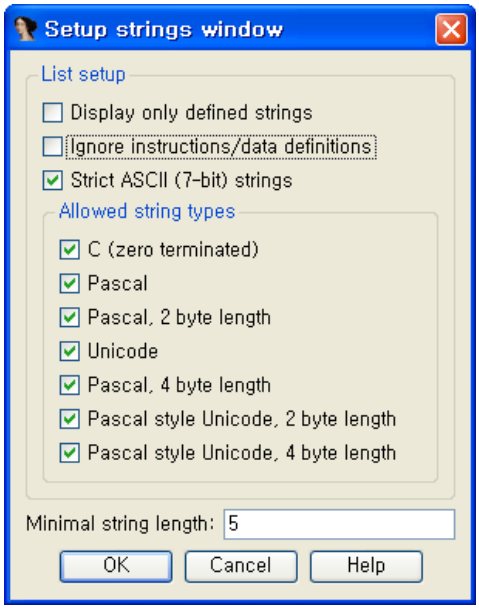
IDA를 끄면서 id0 id1 nam til 파일들이 idb 파일로 묶이게 되는데 이에 대해 설정하는 란입니다. Don't pack database 는 파일을 묶지 않고 Pack database(Store, Deflate) 는 파일을 묶게 됩니다. Deflate설정은 묶을 때 압축을 하게 됩니다. 특별한 이유가 있지 않으면 Pack database(Store)로 설정하면 됩니다.
Collect garbage 는 자바같은 프로그램의 가비지 컬렉터를 생각하면 됩니다.
DON'T SAVE the database는 IDA를 연 후에 작업했던 것들을 저장하지 '않는' 옵션입니다. 만약 프로그램을 처음 열었고 닫을 때 이 옵션을 선택하면 idb 파일은 생성되지 않습니다.

이번엔 그래프 뷰 화면에 대해 기본적인 설명을 해보겠습니다.

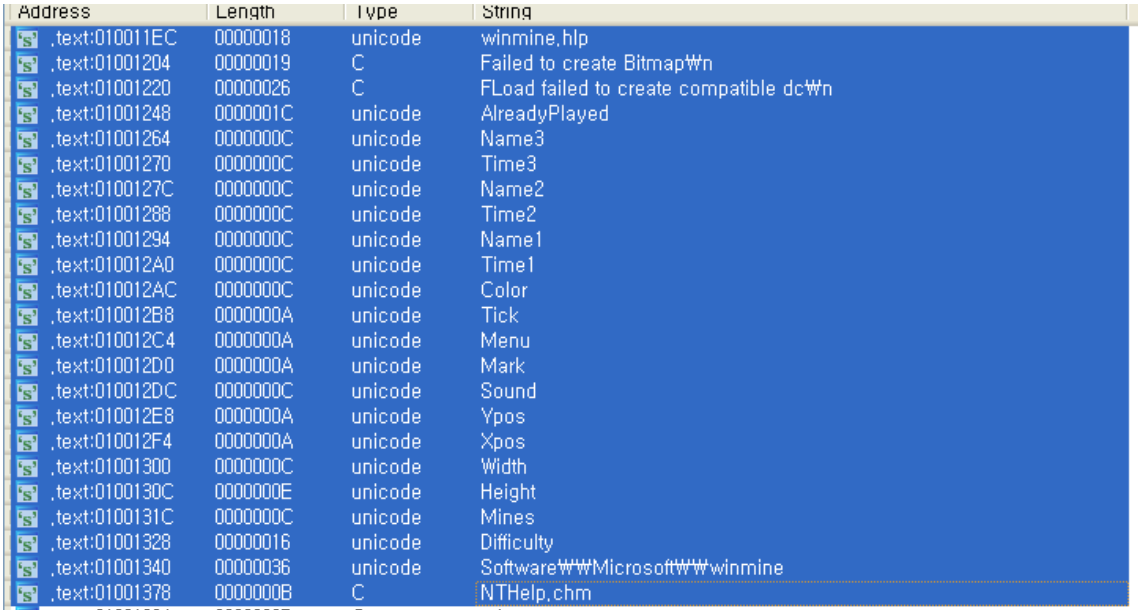
이제 지뢰찾기 분석을 시작하겠습니다.
먼저 프로그램에 있는 각종 문자열들을 검색해 보는걸로 시작하겠습니다.
IDA의 Strings 창이 이 기능을 하는데 기본적으로 화면에 나와있지 않습니다. 메뉴의 View - open subviews - Strings 로 열거나 Quick view 창(CTRL + 1) 을 띄어서 Strings를 선택하면 됩니다.
Quick view 창은 현재 열려있지 않은 창도 보여주고 언제든지 사용할 수 있으므로 창에서 창으로 이동할 때 사용하면 유용합니다.



이게 Strings 창인데 왼쪽서부터 문자열의 주소, 길이, 타입, 내용을 나타냅니다.
그런데 IDA의 기본설정으로는 파일 내에 있는 모든 문자열을 찾아낼 수 없습니다. 따라서 Strings 창에서 오른쪽 버튼을 눌러서 Setup으로 들어가서 다음과 같이 설정을 바꿔줍니다.



옆에 나온 설명이 직관적이므로 상황에 따라서 적절히 설정을 살짝살짝 바꿔주면 됩니다.
하나만 설명하자면 Ignore instruction/data definition은 IDA가 코드나 데이터라고 생각되는 것을 무시하는게 '아니라' 그 반대의 의미입니다. 이 옵션이 체크 되어있으면 명령어나 데이터라 생각되는 문자열부분도 리스트에서 보여줍니다. 만약 찾는 문자열이 보이지 않을 때 이 옵션을 키고 다시 살펴보면 좋습니다.



Address	Length	Type	String
.text:010011EC	00000018	unicode	winmine.hlp
.text:01001204	00000019	C	Failed to create Bitmap₩n
.text:01001220	00000026	C	FLoad failed to create compatible dc₩n
.text:01001248	0000001C	unicode	AlreadyPlayed
.text:01001264	0000000C	unicode	Name3
.text:01001270	0000000C	unicode	Time3
.text:0100127C	0000000C	unicode	Name2
.text:01001288	0000000C	unicode	Time2
.text:01001294	0000000C	unicode	Name1
.text:010012A0	0000000C	unicode	Time1
.text:010012AC	0000000C	unicode	Color
.text:010012B8	0000000A	unicode	Tick
.text:010012C4	0000000A	unicode	Menu
.text:010012D0	0000000A	unicode	Mark
.text:010012DC	0000000C	unicode	Sound
.text:010012E8	0000000A	unicode	Ypos
.text:010012F4	0000000A	unicode	Xpos
.text:01001300	0000000C	unicode	Width
.text:0100130C	0000000E	unicode	Height
.text:0100131C	0000000C	unicode	Mines
.text:01001328	00000016	unicode	Difficulty
.text:01001340	00000036	unicode	Software₩₩Microsoft₩₩winmine
.text:01001378	0000000B	C	NTHelp.chm

눈에 뜨이는 부분이 많습니다. 일단 AlreadyPlayed부터 Difficulty 까지는 각종 설정&저장된 값인것 같고 winmine.hlp 와 NTHelp.chm 은 도움말 파일, Software₩₩Microsoft₩₩winmine 은 레지스트리 설정할 때 쓰인 문자열로 보입니다.
이중 하나를 더블클릭하면 IDA view 창의 해당 문자열 부분으로 이동하게 됩니다.
저는 Software₩₩Microsoft₩₩winmine 를 선택하였습니다.

```
.text:01001340 ; const WCHAR SubKey
.text:01001340 SubKey:                                 ; DATA XREF: sub_1002BC2+19↓o
.text:01001340                                         ; sub_1002DAB+17↓o ...
.text:01001340                 unicode 0, <Software₩Microsoft₩winmine>,0
```

먼저 제일 좌측에 보이는 text:01001340은 섹션명과 '가상주소'입니다.
그런데 잘 보면 똑같은 주소가 4번이나 나오는데 맨 아래 있는것이 실제 값을 나타내고 나머지는 그 주소에 달린 주석들을 나타냅니다.
첫째줄에 보이는 const WCHAR Subkey는 IDA가 자동분석에서 판별한 변수 타입과 이름입니다. IDA의 강력한 기능중 하나인데 IDA는 문자열을 찾으면 그 문자열이 참조되는 코드를 찾아내고 어떤 라이브러리 함수의 인자로 들어갔는지 확인한 후 그에 맞는 이름을 지어줍니다. 이 경우는 RegCreateKeyExW 함수의 Subkey 인자로 들어갔으므로 변수명이 SubKey로 붙었습니다. 만약 라이브러리 함수의 인자가 아닐경우는 'a + 문자열의 일부분' 으로 문자열에 이름을 붙여줍니다.
그리고 4번째 줄의 unicode는 해당 문자열의 타입입니다.

오른쪽에 보이는 DATA XREF 는 이 데이터(문자열)이 참조된 코드를 보여주는 기능입니다. 마우스를 올리면 다음과 같은 화면이 뜨면서 참조된 코드의 일부를 볼 수 있습니다.

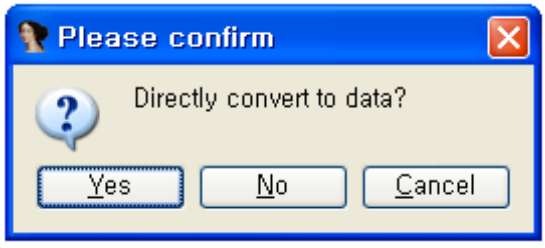
```
push    eax             ; lpdwDisposition
push    offset hKey     ; phkResult
xor     edi, edi
push    edi             ; lpSecurityAttributes
push    20019h          ; samDesired
push    edi             ; dwOptions
push    edi             ; lpClass
push    edi             ; Reserved
push    offset SubKey   ; "Software##Microsoft##winmine"
push    80000001h       ; hKey
```

여기서 마우스 휠을 돌리면 더 볼 수 있습니다.
다시 Strings 창으로 돌아가 맨 아래로 가보면 XYZZY 가 보이는데 더블클릭해서 이동해줍니다.

```
.data:01005034 word_1005034    dw 58h
.data:01005036                 db  59h ; Y
.data:01005037                 db    0
.data:01005038                 db  5Ah ; Z
.data:01005039                 db    0
.data:0100503A                 db  5Ah ; Z
.data:0100503B                 db    0
.data:0100503C                 db  59h ; Y
.data:0100503D                 db    0
.data:0100503E                 db    0
.data:0100503F                 db    0
```

문자열 창에선 유니코드로 인식이 잘 되었는데 디스어셈블 창에선 제대로 인식하지 못했습니다. 이 경우엔 사용자가 직접 데이터 타입을 지정해주어야 하는데 먼저 각 문자에 대한 설명을 하겠습니다.
word_1005035 는 변수의 이름입니다. '데이터의 타입_데이터의 시작주소' 의 형태로 이름이 지어집니다.
그리고 db, dw, dd는 데이터의 크기를 나타냅니다. 각각 byte,word,double 의 약자이고 1,2,4 바이트를 뜻합니다. 이를 바꾸려면 단축키 d 를 눌르면 db -> dw -> dd -> db 순으로 바뀝니다. 데이터 타입을 바꾸는데 필요한 바이트수가 부족하다면 확인창이 뜨게 됩니다.

```
db    0
db    0
dw 58h
dd 5A0059h
db  5Ah ; Z
db    0
db  59h ; Y
db    0
db    0
db    0
```



이 경우는 dw에서 dd로 바꿀려 할때 발생한 확인창입니다. dw는 2바이트고 dd는 4바이트니까 dd로 바꿀려면 2바이트(db 2개)가 필요한데 바로 아래에는 dd 하나가 있습니다. 만약 강제로 dw를 dd로 바꾸려면 dd(5A0059) 를 db로 바꾸고 상위 두 바이트를 dw -> dd 로 만드는데 쓰이는데 이때 5A0059 의 데이터 타입이 깨지게 됩니다. 그래서 이를 확인하려고 묻는 창을 띄우는 것입니다. 여기서 Yes를 누르면 아래사진과 같이 변합니다.

```
dd 590058h
db  5Ah ; Z
db    0
```

직접 데이터 타입을 바꿔보면서 해보면 이해가 바로 될겁니다.

```
.data:01005034 word_1005034     dw 58h
.data:01005036                  db  59h ; Y
.data:01005037                  db    0
.data:01005038                  db  5Ah ; Z
.data:01005039                  db    0
.data:0100503A                  db  5Ah ; Z
.data:0100503B                  db    0
.data:0100503C                  db  59h ; Y
.data:0100503D                  db    0
.data:0100503E                  db    0
.data:0100503F                  db    0
```

다시 돌아와서 지금 저 XYZZY는 실제로 유니코드이지만 IDA는 dw 하나와 db들로 인식을 했습니다.

Hex View 창으로 보면 이게 유니코드인지 확인할 수 있습니다.

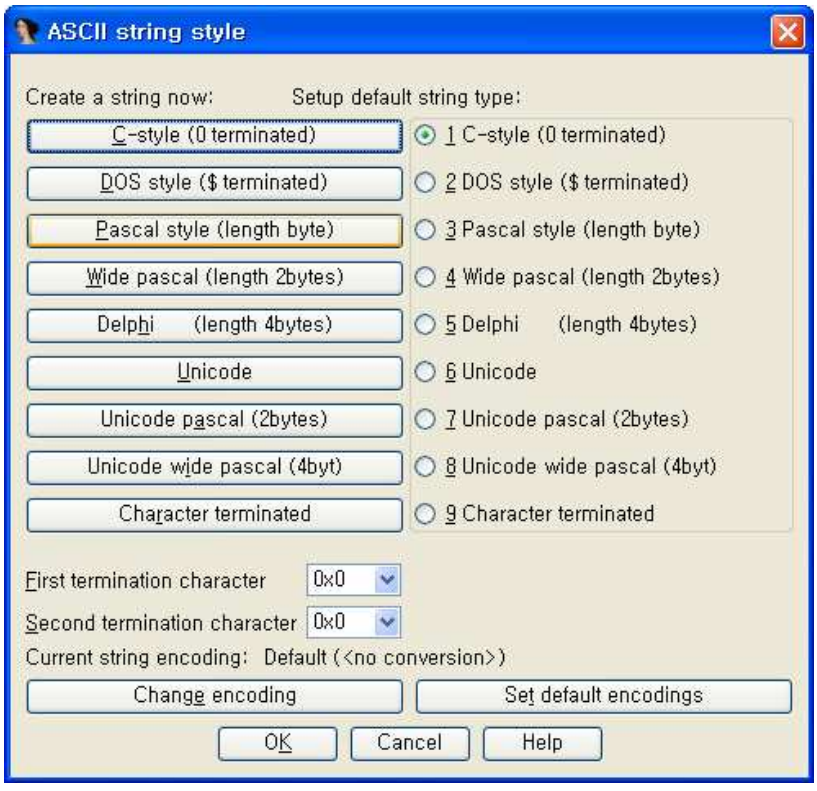
```
0100502D  00 00 00 1E 00 00 00 58  00 59 00 5A 00 5A 00 59  .......X.Y.Z.Z.Y
0100503D  00 00 00 8D 00 00 00 E8  03 00 00 8E 00 00 00 E9  ........
```

Hex view 창으로 가면 해당 주소가 바로 나올텐데 그 이유는 기본적으로 IDA view-A 창과 Hex view-A 창은 싱크로되어있기 때문입니다.. 싱크로 되어있다는 것은 지금 커서의 위치를 동기화 한다는 것입니다. 만약 IDA view 창에서 401000 부분을 보고 있다면 싱크로된 Hex view 창에서도 401000 부분이 보이고 이 반대의 경우도 마찬가지입니다.

이에 대한 설정은 오른쪽 마우스 - synchronize with에서 선택을 하면 됩니다.

참고로 IDA view 창과 Hex view 창은 쿽뷰(CTRL + 1)를 통해 여러개를 띄울 수 있습니다.

해당 문자열이 유니코드인걸 확인했으니 데이터 타입이 유니코드인것을 IDA에 알려줘야 됩니다. 문자열의 시작인 01005034부터 XYZZY 그리고 마지막 NULL까지인 0100503F 까지 드래그를 하거나 시작부분인 01005034에 커서(포인트)를 두고 ALT + A 를 눌르면 다음과 같은 창이 뜹니다.



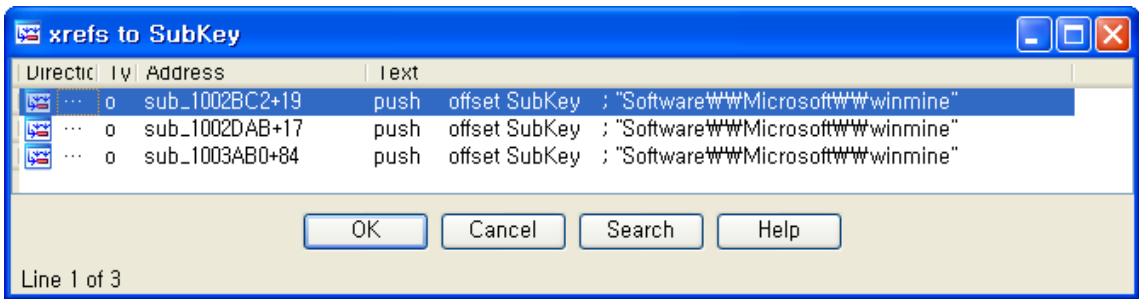
좌측의 버튼형식은 현재 선택된 데이터를 해당 문자열로 바꿔줍니다.
우측의 라디오 버튼은 디폴트 문자열 타입을 지정할 수 있습니다.(현재 데이터 형식이 바뀌지는 않음) 후에 단축키 A를 사용하여 여기서 셋팅한 설정의 문자열 형식으로 바로 바꿀 수 있습니다.
이제 왼쪽의 Unicode를 눌러서 IDA에게 이 데이터가 유니코드임을 알려주면 다음과 같이 변합니다.

```
aXyzzy:
                unicode 0, <XYZZY>,0
```

데이터에서 문자열로 변한것 말고도 좌측에 문자열 변수 명(aXyzzy)가 붙여지고 코드 전체에 영향을 끼치게 됩니다. 이게 무엇을 뜻하는지는 다음 그림을 보면 쉽게 알 수 있습니다.

```
jge     loc_10021A9      ; jumptabl        jge     loc_10021A9      ; jumptable 01001F75
mov     cx, word_1005034[eax+2]    ->     mov     cx, word ptr aXyzzy[eax+2] ; "XYZZY"
sub     cx, word ptr [ebp+wParam]          sub     cx, word ptr [ebp+wParam]
```

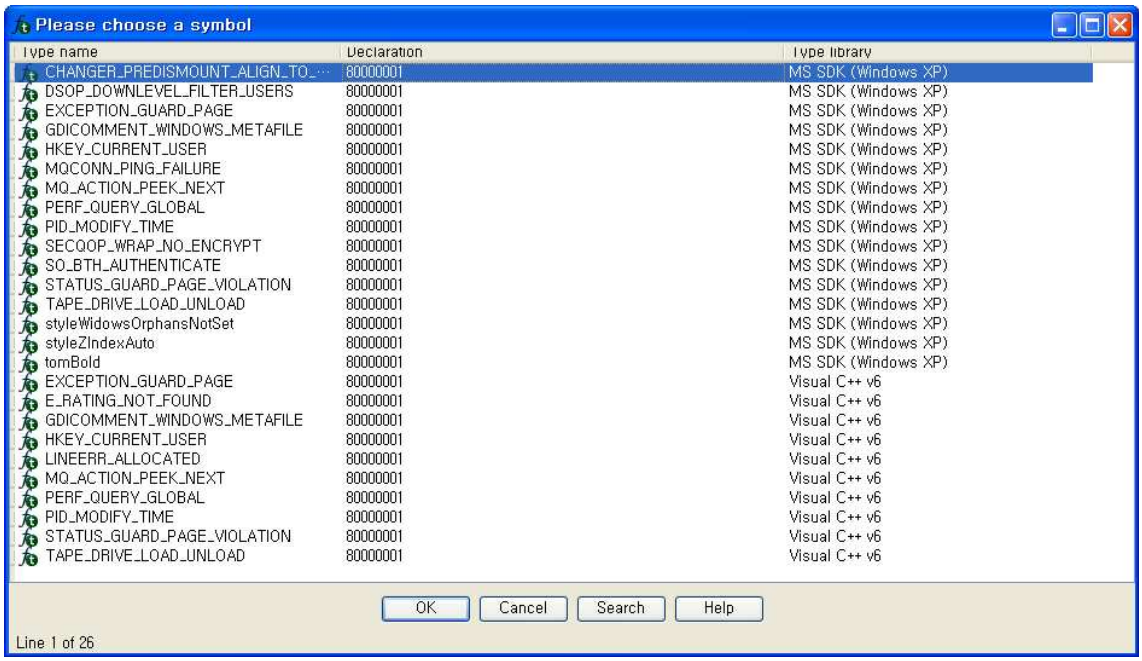
이제 다시 SubKey 문자열로 돌아갑니다. 지금까지 그대로 이 문서를 따라하신 분이라면 esc 키를 누르면 SubKey 문자열이 있는 곳으로 갈것입니다. esc는 이전 위치로 가는 단축키로 올리디버거의 '-' 단축키와 똑같은 기능을 합니다. 툴바에 똑같은 기능을 하는 버튼이 있습니다.()
돌아가서 Subkey를 누르고 단축키 X를 누르면 IDA의 강력한 기능중 하나인 xref to 창이 뜹니다.



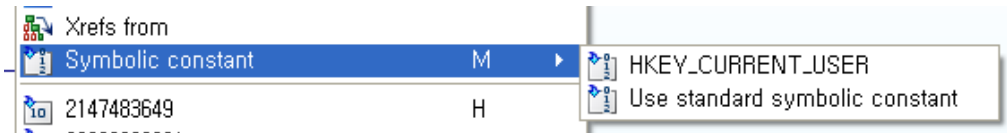
xref to는 함수나 데이터(문자열, 상수 등등) 가 사용된 곳을 모두 찾아주는 기능입니다. 이 함수나 데이터가 어떤 용도로 쓰이는지 알아보거나 프로그램의 흐름을 파악하는데 큰 도움을 줍니다.
창에 나온 정보는 Direction(코드의 상대적 위치,위,아래), Type(o,r,w), Address(주소) Text(코드)입니다. 여기서 Type은 Open, Read, Write의 약자로 코드의 쓰임새를 간단하게 파악할 수 있습니다.(예로 mov eax , subkey 는 R)
OK 버튼을 누르거나 더블클릭하면 이동하는데 여기선 두 번째 sub_1002DAB + 17 부분으로 이동합니다.

```
lea     eax, [esp+8+dwDisposition]
push    eax             ; lpdwDisposition
push    offset hKey     ; phkResult
xor     esi, esi
push    esi             ; lpSecurityAttributes
push    20006h          ; samDesired
push    esi             ; dwOptions
push    esi             ; lpClass
push    esi             ; Reserved
push    offset SubKey   ; "Software₩₩Microsoft₩₩winmine"
push    80000001h       ; hKey
call    ds:RegCreateKeyExW
```

RegCreateKeyExW 함수가 쓰이는 부분인데 IDA는 자동으로 각각 함수의 인자 명들을 주석으로 넣어주지만 8000001h 와 같이 상수가 뭘 의미하는지는 자동으로 알려주지는 않습니다. 하지만 사용자가 설정을 할 수 있습니다.
push 80000001h 의 80000001h 을 오른쪽 클릭하고 Use standart symbolic constant 선택하면 창이 하나 뜹니다.



RegCreateKeyExW 함수를 아는 사람이나 MSDN에서 찾아보면 이 값을 가진 심볼릭 상수중 적당한것은 HKEY_CURRENT_USER 라는걸 알 수 있습니다. 더블클릭해서 적용하면 다음과 같이(push HKEY_CURRENT_USER ; hKey)변합니다.
만약 잘못 적용했다 싶으면 오른쪽 버튼을 누르면 나오는 여러 표현방식중 적절한걸 골라주면 됩니다.
이제 Enums 창으로 가봅니다.
가보면 [COLLAPSED ENUM MACRO_HKEY. PRESS KEYPAD "+" TO EXPAND] 라는 글이 나와있습니다. 이는 방금 심볼릭 상수를 지정하면서 관련된 심볼릭 상수가 자동으로 로드된 것입니다.
후에 비슷한 심볼릭 상수를 설정할때는 다음과 같이 선택창에 따로 뜹니다.



이와 똑같은 방식으로 위에 있는 push 20006h ; samDesired 도 KEY_WRITE 로 바꿔줍니다.

그리고 IDA view 창이나 Hex View 창에서 클릭하면 그와 똑같은 문자열들은 노란색으로 하이라이팅이 됩니다.

esi 를 클릭했을때

```
push    ecx
push    esi
lea     eax, [esp+8+dwDisposition]
push    eax             ; lpdwDisposition
push    offset hKey     ; phkResult
xor     esi, esi
push    esi             ; lpSecurityAttributes
push    KEY_WRITE       ; samDesired
push    esi             ; dwOptions
push    esi             ; lpClass
push    esi             ; Reserved
```

e를 그래그 했을 때

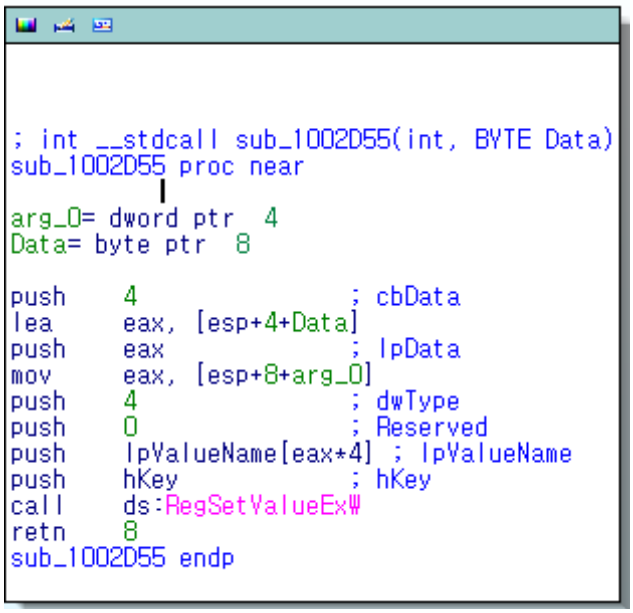
```
push    ecx
push    esi
lea     eax, [esp+8+dwDisposition]
push    eax             ; lpdwDisposition
push    offset hKey     ; phkResult
xor     esi, esi
push    esi             ; lpSecurityAttributes
push    KEY_WRITE       ; samDesired
push    esi             ; dwOptions
push    esi             ; lpClass
push    esi             ; Reserved
```

이 기능은 한 레지스터가 함수내에서 어떻게 변하는지 보거나 call같은 명령어가 어딨는지 볼때 주로 씁니다. 이 상태(하이라이팅)을 유지하려면 이 버튼( , 툴바에 있음)을 누르면 다른 문자를 클릭해도 하이라이팅이 풀리지 않습니다.

아래로 좀 내려가면 똑같은 함수가 계속 반복되는 부분을 볼 수 있습니다.

```
push    dword_10056AC   ; Data
push    3               ; int
call    sub_1002D55
```

sub_1002D55를 더블클릭하면 그 함수 내부로 들어갑니다.

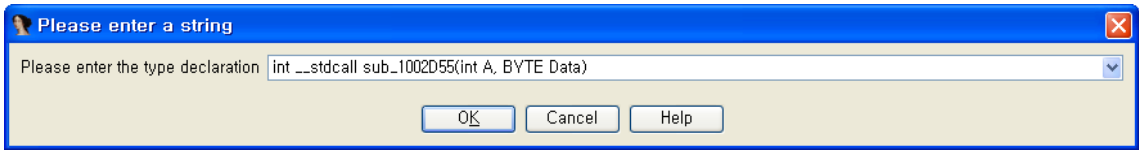


```
; int __stdcall sub_1002D55(int, BYTE Data)
sub_1002D55 proc near

arg_0= dword ptr  4
Data= byte ptr  8

push    4               ; cbData
lea     eax, [esp+4+Data]
push    eax             ; lpData
mov     eax, [esp+8+arg_0]
push    4               ; dwType
push    0               ; Reserved
push    lpValueName[eax*4] ; lpValueName
push    hKey            ; hKey
call    ds:RegSetValueExW
retn    8
sub_1002D55 endp
```

상단에 이 함수에 대한 IDA가 달아둔 주석이 있습니다. C언어 형태의 함수의 리턴 타입, 인자들과 그 타입과 함수의 이름이 나와 있습니다. 이 함수의 이름은 리버서가 임의로 바꿀 수 있습니다. 그리고 그 바로 아래 나와 있는 부분은 이 함수 내에서 쓰이는 변수 목록입니다. 다만, 모두 나와있지는 않고 IDA가 찾은것만 나와 있습니다.
이 함수의 인자 타입을 보면 이상한 점이 있습니다. int, BYTE Data에서 BYTE 뒤에는 Data라는 인자명이 있는 반면 int 뒤에는 인자 명이 나와있질 않습니다. 그 이유는 IDA가 함수를 분석할 때 적절한 인자명을 결정하지 못했기 때문입니다. 이때 리버서가 인자명을 지어줄 수 있습니다.
단축키 Y를 누르면 나오는 창에서 C언어 함수 형태에 맞게 적절하게 이름을 넣어주면 됩니다. (아래 사진의 A는 제가 넣었습니다.)



이 창은 함수의 타입을 설정하는 창입니다. 단순히 변수명만 바꾸는 것 외에 함수의 리턴값, 호출방식, 인자 타입이나 개수 등을 바꾸거나 추가할 수 있습니다.

위 사진처럼 int 뒤 A라고 변수명을 지정해주면 디스어셈블 화면도 그에 맞게 변합니다. (argv_0 -> A)

아래는 변수명을 지정해준 후 모습입니다.

```
push    4               ; cbData
lea     eax, [esp+4+Data]
push    eax             ; lpData
mov     eax, [esp+8+A]
```

보면 2번째 줄과 4번째 줄에서 [esp+4+Data]나 [esp+8+A]란 다소 이상해 보이는 표현을 씁니다. 하지만 그 위치가 실제 어디인지 살펴보면 편리한 표현인걸 알 수 있습니다. (1번째 줄과 3번째 줄에 스택이 PUSH된 걸 반영한것임)
esc를 눌러서 함수가 호출된 부분으로 가보면 아까 넣었던 이름이 주석으로 추가로 붙은걸 확인할 수 있습니다.

```
push    uValue          ; Data
push    2               ; A
call    sub_1002D55
```

이런식으로 API가 아닌 경우도 함수 설정에 맞게 주석을 붙여줍니다.

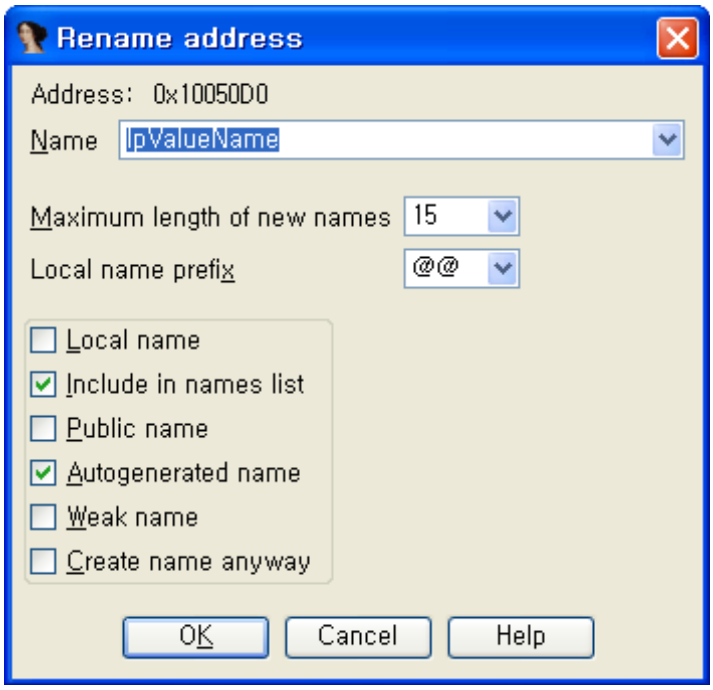
아까 본 함수를 분석하자면 *lpValueName[A * 4] 의 이름에 해당하는 레지스터에 Data 값을 설정하는 동작을 합니다.
그럼 A는 주소값 배열의 인덱스 번호를 의미하므로 적당한 변수명으로 바꿔줍니다. 저는 c_index로 하였습니다.

다음으로 lpValueName 을 더블클릭해서 들어가봅니다.

```
.data:010050D0 ; LPCWSTR lpValueName
.data:010050D0 lpValueName     dd offset aDifficulty   ; DATA XREF: sub_1002B27+13↑r
.data:010050D0                                         ; sub_1002B80+12↑r ...
.data:010050D0                                         ; "Difficulty"
.data:010050D4                 dd offset aMines        ; "Mines"
.data:010050D8                 dd offset aHeight       ; "Height"
.data:010050DC                 dd offset aWidth        ; "Width"
.data:010050E0                 dd offset aXpos         ; "Xpos"
.data:010050E4                 dd offset aYpos         ; "Ypos"
.data:010050E8                 dd offset aSound        ; "Sound"
.data:010050EC                 dd offset aMark         ; "Mark"
.data:010050F0                 dd offset aMenu         ; "Menu"
.data:010050F4                 dd offset aTick         ; "Tick"
.data:010050F8                 dd offset aColor        ; "Color"
.data:010050FC                 dd offset aTime1        ; "Time1"
.data:01005100                 dd offset aName1        ; "Name1"
.data:01005104                 dd offset aTime2        ; "Time2"
.data:01005108                 dd offset aName2        ; "Name2"
.data:0100510C                 dd offset aTime3        ; "Time3"
.data:01005110                 dd offset aName3        ; "Name3"
.data:01005114                 dd offset aAlreadyplayed ; "AlreadyPlayed"
```

IDA가 생각하기에 어떤 값의 오프셋이라 판단되면 위와같이 나타내 집니다. 'dd offset' 다음에 나오는것이 해당 주소 값의 이름이며 더블클릭하면 이동합니다. 오른쪽에 회색 주석은 각각 오프셋에 해당하는 주소에 있는 데이터를 뜻합니다.

모양새를 보니 게임에 관련된 설정이나 데이터를 저장하는듯 합니다. 그럼 그에 맞게 변수명을 바꿀 수 있습니다. lpValueName에 커서를 두고 단축키 N을 눌르면 다음과 같은 창이 뜹니다.



여기서 적당한 이름으로 바꿔주면 됩니다.

각각 설정이 의미하는바는 다음과 같습니다.

Local name: 특정 함수 내부에서만 쓰이는 이름으로 지정

Include in names list: Name 창 리스트에 추가함

Public name: 외부 라이브러리에서 익스포트된 이름으로 설정함

Autogenerrated name: IDA가 설정한 이름일 경우 체크됨
Weak name: 이 변수명을 weak 변수로 설정함
Create name anyway: 설정한 이름이 다른 변수명과 동일하더라도 그대로 설정함(비추천)

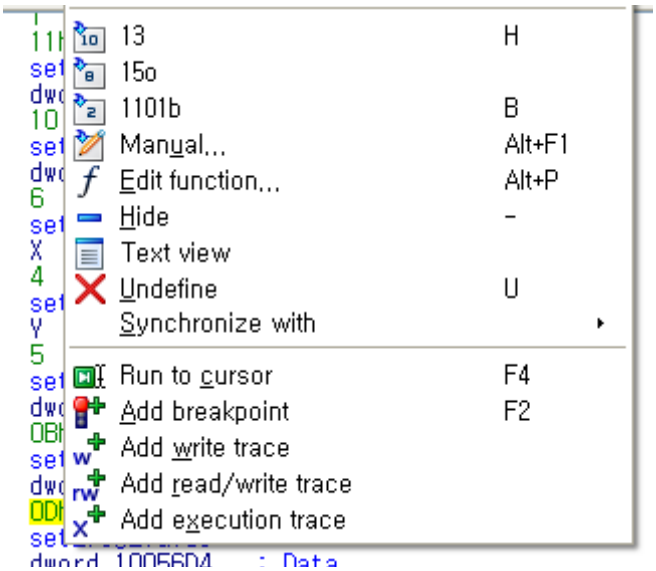
Weak name의 특수한 경우에 사용되는 옵션인데 일반적인 리버싱을 하고 있는 경우라면 쓸 필요가 없다고 생각됩니다. 아래는 헥스레이 IDA 헬프 페이지에서의 설명입니다.

Weak name
You can declare a name as a weak name. If the current
assembler supports the "weak" directive, IDA will use it.
Otherwise, the weakness of the name will be displayed as a comment.

Autogenerrated name 의 경우는 리버서가 체크하는게 아니라 IDA가 체크하는 것입니다. 만약 리버서가 이름을 새로 바꾸면 따로 체크해제 하지 않더라도 자동으로 체크가 해제됩니다. 어쨌든 여기 옵션들은 모두 건드릴 필요가 없습니다.
문자열의 최대 길이를 설정하는 옵션은 기본으로 15글자로 되어있는데 만약 변수명이 15글자를 넘어가면 최대길이를 변경하겠냐는 창이 뜹니다. 여기서 OK를 누르면 알아서 적당하게 바꿔줍니다.
참고로 이름을 정할 때 제약사항이 있는데 위반하면 "You've entered a bad identifier" 라는 경고창이 뜹니다. 정확하게 어떤 제약사항이 있는지는 나와있질 않아서 모르겠지만 보통 문제를 일으키는건 시작문자가 영어가 아닌 경우, 이름에 특정 특문이 있는 경우, 이름에 한글이 들어간 경우입니다. 다만 맨 앞 글자에 ? 나 _ 도 들어갈수 있습니다.
그리고 이름 내부에 제약사항을 지키는 특문이 있더라도 화면에는 _ 로 표시됩니다. 그렇기에 그 변수에는 xref to 기능이나 rename 기능이 잘 먹히지 않습니다.
이런 복잡한 문제 때문에 제가 추천하는 이름짓기 규칙은 다음과 같습니다.
-이름은 항상 _ 와 ? 와 영어 만 쓰자
-숫자는 맨 끝에만 사용하자
그리고 변수명을 바꾸고 나서 다시 원래의 변수명으로 돌아가고 싶으면 rename 창에서 name 란에 공백을 넣으면 됩니다. 다만 지금 보는 변수인 lpValueName 처럼 IDA가 판단해서 이름을 지어준 변수의 경우 원래의 변수명으로 돌아가지 않습니다.
예) off_10050D0 -> something_value -> off_10050D0
예) lpValueName -> something_value -> off_10050D0

다시 분석으로 돌아와서 lpValueName 을 적당한 이름으로 바꿉니다. 저는 arr_setting_name 으로 하겠습니다.
이제 이 함수는 c_index 에 해당하는 arr_setting_name 이름을 가지는 레지스터에 data값을 쓰는 역할을 합니다.
그럼 함수 이름을 적당하게 바꿔줍니다. 단축키 Y로 함수 타입 설정창을 띄어서 이름을 바꿔줘도 되지만 단축키 N으로 lpValueName 이름을 바꿀때와 똑같이 해도 됩니다.
저는 set_reg_value 로 하였습니다.

함수 밖으로 다시 나와서 아래로 가다보면 c_index 값이 쭉 나와있는데 10이 넘는 값들은 hex값으로 쓰여져 있어서 한눈에 알아보기 어렵습니다. IDA는 보통 상수들을(단일 문자 포함) hex값으로 나타내는데 이를 10진수나 문자로 바꿀려면 마우스 오른쪽 버튼을 눌러서 나오는 것중 적절한걸 선택하면 됩니다. 만약 10진수로 바꾸는거라면 단축키 H, 문자로 바꾸는 거라면 단축키 R를 써도 됩니다.

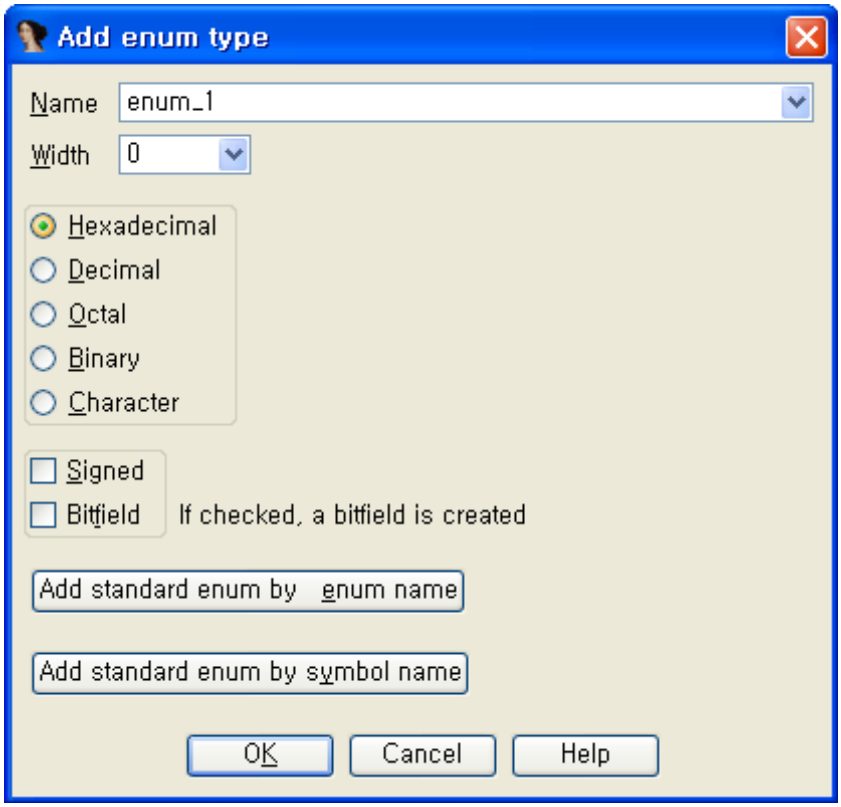


hex값으로 된 문자들을 10진수로 바꿔서 몇 번째 offset인지는 바로 알 수 있지만 그게 실제 어느것인지 알려면 arr_setting_name 에 가서 확인해봐야 해서 아직 불편합니다. IDA는 이런 불편함을 해결할 기능을 가지고 있습니다. 위에서 비슷한 기능이 쓰였는데 바로 symbolic constant 설정 기능입니다.
위에서는 라이브러리에 있는 standard 심볼을 가져가 썻지만 이것 말고도 IDA에서 사용자가 직접 심볼을 정의해서 쓸 수 있습니다. Enums 창에 가서 이러한 것들을 할 수 있습니다.

```
Ins/Del/Ctrl-E: create/delete/edit enumeration types
N/Ctrl-N      : create/edit a symbolic constant
U             : delete a symbolic constant
; or :        : set a comment for the current item
```

Enums 창에 가면 위 사진의 간략한 사용설명이 나와있습니다. 사실 이것만 읽으면 심볼을 만드는데 문제없지만 한번 설명하겠습니다.
IDA에서는 심볼을 근냥 저장하는게 아니라 비슷비슷한 심볼을 모아놓은 enumeration을 만들어서 그 내부에 심볼을 저장합니다.(사실 enumeration란 표현이 맞는지는 모르겠습니다... 일단 이 문서에서는 enumeration란 표현을 쓰겠습니다)

먼저 ins 키로 enumeration를 만듭니다.

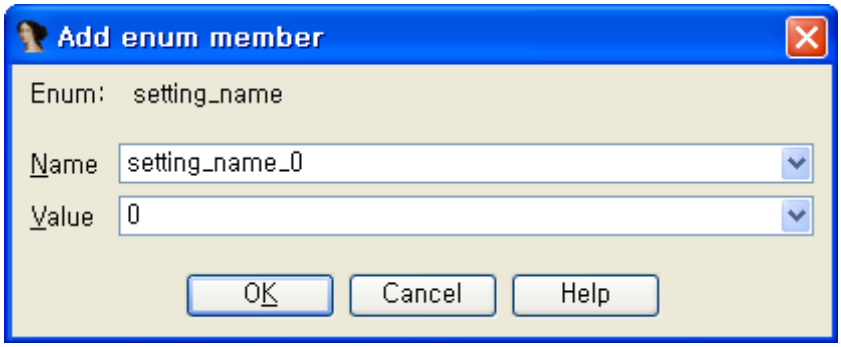


이때 Hexadecimal, Decimal 등을 고를수 있는데 따로 설명이 나와있지 않습니다. 이건 설정값에 영향을 끼치지는 않지만 기본으로 보이는 심볼 값의 표현 방식입니다. 간단하게 말해 enums 창에서 보이는 심볼 값을 10진수로 볼꺼냐 16진수로 볼꺼냐 선택하는 것입니다. 이것 말고는 다른곳에 이 설정이 영향을 끼치지는 않습니다. 개인적으로는 Decimal를 추천합니다.
width 설정은 심볼값의 크기를 뜻합니다. 아무 숫자나 넣을순 없고 2의 승수로만 (1,2,4,8,...64) 넣을 수 있습니다. 기본으로 0이 들어가 있는데 이 값을 그냥 써도 됩니다.
signed 는 양수,음수 구분이 있을때 체크하고 Bitfield 는 constant_1 | constant_2 이런식으로 OR 연산자를 통해 비트를 설정하는 식의 심볼을 설정할 때 쓰입니다.
아래 두 버튼은 일반적인 라이브러리에서 쓰이는 심볼을 추가할 때 쓰이는 버튼입니다.

여기서 저는 setting_name 이란 이름과 Decimal 로 하겠습니다.

```
; enum setting_name
```

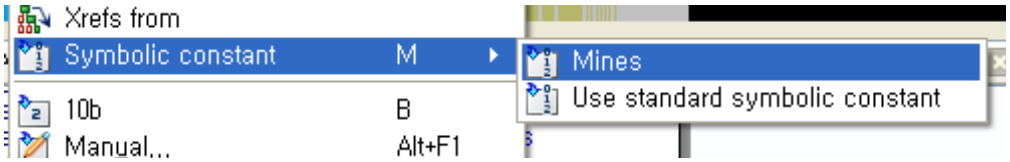
이렇게 enum 이 추가되는데 여기에 심볼을 추가하려면 반드시 커서가 해당 enum 에 있어야 합니다.



그리고 그상태에서 N을 누르면 새로운 심볼을 추가하는 창이 뜹니다. 여기에 적절한 이름과 상수를 넣으면 되고 상수를 넣을때 16진수는 0x 형태로, 문자는 '문자' 의 형태로 넣으면 됩니다.
이제 arr_setting_name에 있는 값들을 바탕으로 알맞게 심볼을 만들어 줍니다. 참고로 IDA view 창을 독립시키고 enums 창을 보고 하면 편합니다.

```
; enum setting_name
Difficulty      = 0
Mines           = 1
Height          = 2
Width           = 3
Xpos            = 4
Ypos            = 5
Sound           = 6
Mark            = 7
Menu            = 8
Tick            = 9
Color           = 10
Time1           = 11
Name1           = 12
Time2           = 13
Name2           = 14
Time3           = 15
Name3           = 16
AlreadyPlayed   = 17
```

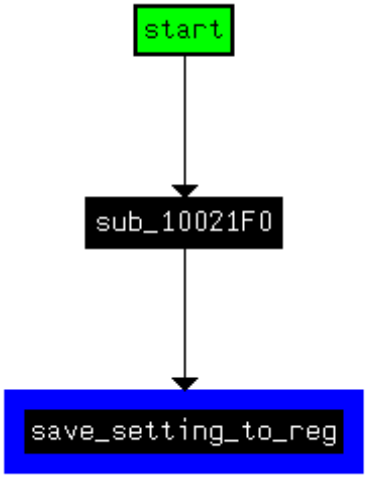
그후 set_reg_value 를 호출하는 함수로 돌아가서 상수를 심볼로 모두 바꿔줍니다.(중간에 Data 가 1인 부분이 있는데 이건 바꾸면 안되요) 바꾸는 방법은 위에서 했던 심볼을 지정하는 방법과 같습니다.



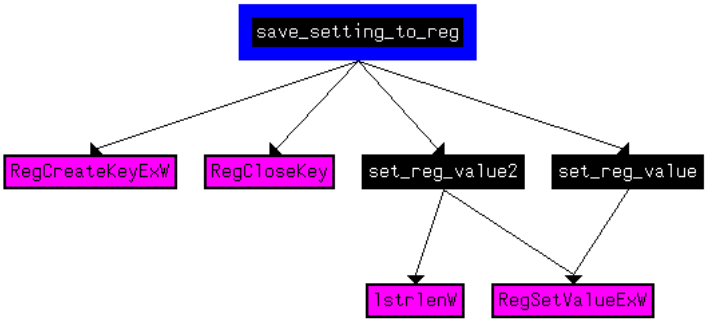
일일이 마우스로 하는게 불편하다면 단축키 M을 써도 됩니다.

아래 가다보면 호출 형태는 비슷한데 다른 함수(sub_1002D7A)가 보입니다. 분석해보면 set_reg_value 함수와 레지스트리 타입과 그 길이만 다릅니다.
비슷한 이름(전 set_reg_value2)을 붙이고 그 함수를 호출하는 인자들도 심볼로 바꿔줍니다.
이제 sub_1002DAB(위 2개 함수를 호출하는 함수)를 모두 분석했습니다. 이 함수는 지뢰찾기의 설정들을 모두 레지스터에 저장하는 역할을 합니다. 함수의 역할을 알았으니 먼저 sub_1002DAB의 이름을 알맞게 바꿔줍니다. 저는 save_setting_to_reg 하겠습니다.
그럼 이제 궁금해지는것은 이 함수는 언제 호출되느냐는 것입니다.

위에서 썻던 xref to 기능을 쓰면 바로 이 함수를 누가 호출하는지 알 수 있지만 다른 기능을 써보겠습니다. (하지만 단축키 X의 xref to 기능을 추천합니다)
IDA에는 그래프와 관련된 여러 가지 기능이 있습니다. 그중 하나가 Function calls입니다. 메뉴 - View - Graphs - Function calls 를 선택하면 그래프 뷰어가 나오면서 프로그램 전체의 흐름을 한눈에 보여줍니다. (혹시 여기서 에러가 나오는 분은 3부를 살펴봐 주세요) 이 기능의 단점은 너무 많은 정보를 보여줘서 리버서가 원하는 것을 뽑아내기 힘들다는 것입니다. (직접 해보시면 무슨말인지 알겁니다) 그렇기에 여기서 누가 save_setting_to_reg 를 호출하는지 알아보기에는 조금 힘듭니다.
다음 기능은 xrefs to 기능입니다. 위에서 말한 단축키 X의 xrefs to와 이름이 똑같습니다. 메뉴 - View - Graphs - xrefs to 로 실행할 수 있습니다. 하는일은 이 함수에 도달하기까지 모든 경우의 함수 흐름을 그래프로 보여줍니다. 결과는 다음과 같습니다.

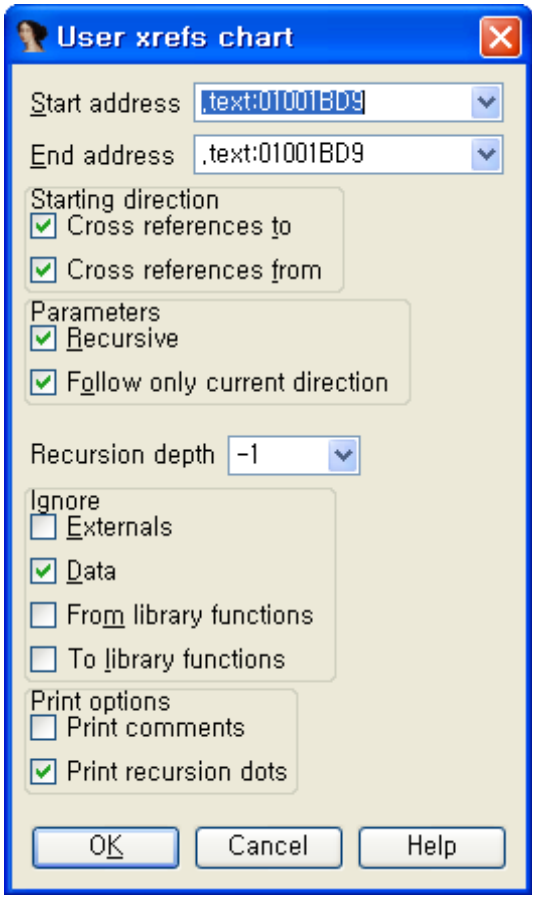


이 기능은 이 함수를 직접 호출하는 함수뿐만 아니라 그 상위 함수까지 모두 보여줍니다. 위의 그림의 경우 start 가 sub_10021F0 을 호출하고 sub_10021F0 이 save_setting_to_reg를 호출하는 걸 알 수 있습니다. 다만 이건 그래프일 뿐이라 더블클릭한다고 이동하지도 않고 드래그조차 되지 않습니다.
xrefs to 기능을 통해 이 함수를 호출하는 흐름을 알 수 있었지만 나머지 그래프 기능을 봅시다.
메뉴 - View - Graphs - xref from 은 이 함수에서 호출하는 함수들을 나열해 줍니다. 결과는 다음과 같습니다.

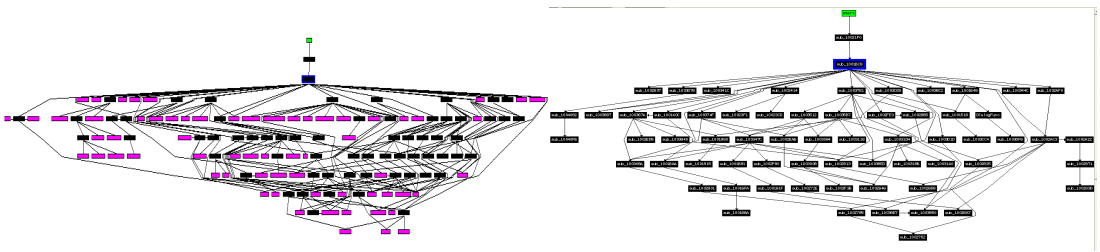


xrefs to 처럼 직접 호출되는 함수뿐만 아니라 거기에서 호출되는 함수까지 모두 보여줍니

다. 참고로 보라색의 상자는 라이브러리 함수를 뜻합니다.
이번엔 xrefs to 와 xref from 을 동시에 볼수 있는 기능을 소개하겠습니다.
user xrefs chart 인데 상당히 편리합니다.
메뉴 - View - Graphs - user xrefs chart 로 가면 다음과 같은 창이 뜹니다.

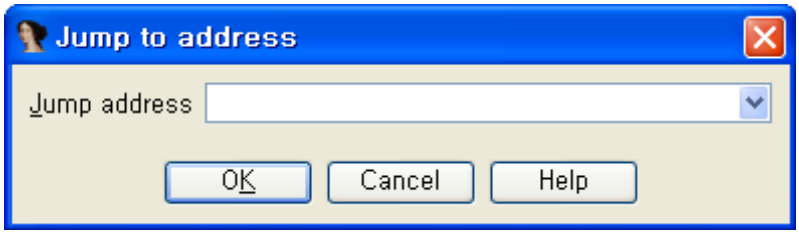


위 start와 end address 는 IDA가 알아서 지정해 줍니다.
여러 옵션중 중요한건 depth 설정과 ignore 설정입니다. depth는 말 그대로 깊이를 설정하는건데 3으로 설정하면 3번에 걸쳐서 호출하거나 호출된 함수를 그래프에 나타냅니다. 직접 해보면 바로 알 수 있습니다. -1이 기본값인데 이렇게 하면 깊이에 제한이 없습니다.
ignore은 보지 않아도 되는 항목을 없애줍니다. 아까 위 그래프에선 보라색으로 라이브러리 함수 호출이 그래프에 나와있습니다. 이게 한두개이면 괜찮지만 함수의 개수가 많아지면 생각보다 그 양이 많아져서 그래프가 매우 엉망이 되버립니다. 그럴때 Externals 에 체크하면 라이브러리 호출 함수는 그래프에 표시를 하지 않습니다. 아래는 그 예입니다.

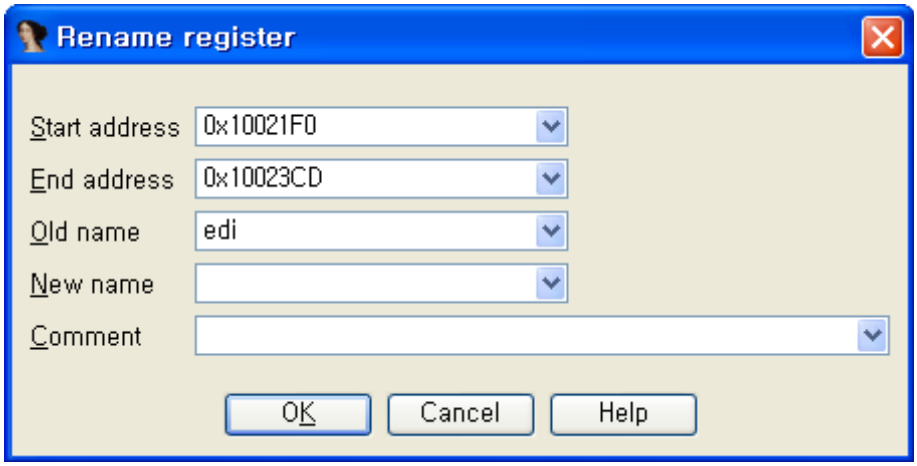


다시 분석으로 돌아와서 이 함수는 sub_10021F0에 의해 호출된다는걸 알았습니다. 그럼 그

곳으로 이동해봐야 하는데 여기서는 단축키 X 의 xref to 가 아닌 그래프의 xref to 방식으로 주소를 알게 되었습니다. 고로 주소나 함수 이름을 가지고 이동을 해야 합니다. 이때 쓰이는것이 단축키 G(Go)입니다.
단축키 G를 누르면 다음과 같은 창이 뜹니다.

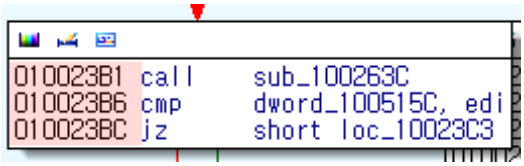


여기에는 주소를 적을 수도 있지만 IDA에서 쓰이는 대부분의 이름(함수명, 변수명 등)은 모두 사용할 수 있습니다.
sub_10021F0을 적어서 그 함수로 이동합니다. 아래(010023BE, 끝자락에 있음)에 보면 save_setting_to_reg 를 호출하는 모습을 볼 수 있습니다.
바로 위에 보면 sub_100263C를 호출하고 dword_100515C, edi 를 비교하는 구문이 나옵니다. dword_100515C 로 들어가봐도 초기값인 0으로 되어있어서 어떤 역할을 하는지 알 수 없습니다. 그럼 이번엔 함수 내의 edi 레지스터의 변화를 찾아서 올라가봅니다. edi를 클릭해 하이라이팅 된 상태에서 하면 편합니다.
계속 edi 의 변화를 위로 따라가다 보면 한번도 edi값은 변화되지 않다가 맨 위에서 edi가 xor edi,edi 명령어 때문에 0으로 되는걸 볼 수 있습니다. 그리고 그 바로 위에는 함수 호출 시의 edi 값을 스텍에 PUSH하고 맨 마지막에는 다시 POP하는걸 볼 수 있습니다. 즉 이 함수 전체에서 edi의 값은 0입니다.
이럴때 사용하는것이 rename register 기능입니다. 레지스터 역시 이름을 리버서가 임의로 지정할 수 있습니다. 다만 다소 그 방법이 귀찮습니다.
먼저 이름을 바꾸고자 하는 레지스터(여기선 edi)를 누르고 단축키 N을 누르면 창이 뜹니다.



new name 에 원하는 이름을 넣어주면 됩니다. 까다로운건 start,end address입니다. 기본으로 start와 end 주소는 함수의 처음과 끝이 들어가 있습니다. 여기선 edi가 0이 되는 부분부터 pop 하는 부분까지만 설정해 줘야 하는데 그 주소를 그래프 뷰에선 기본으로 보이지 않습니다.(1번째 문제) 그리고 어떤 흐름 내의 특정 레지스터만 이름을 바꾸고자 할때 분기가 많으면 일일이 프로그램 구간들을 나누어서 start와 end 주소를 입력해 주어야 합니다. 너무

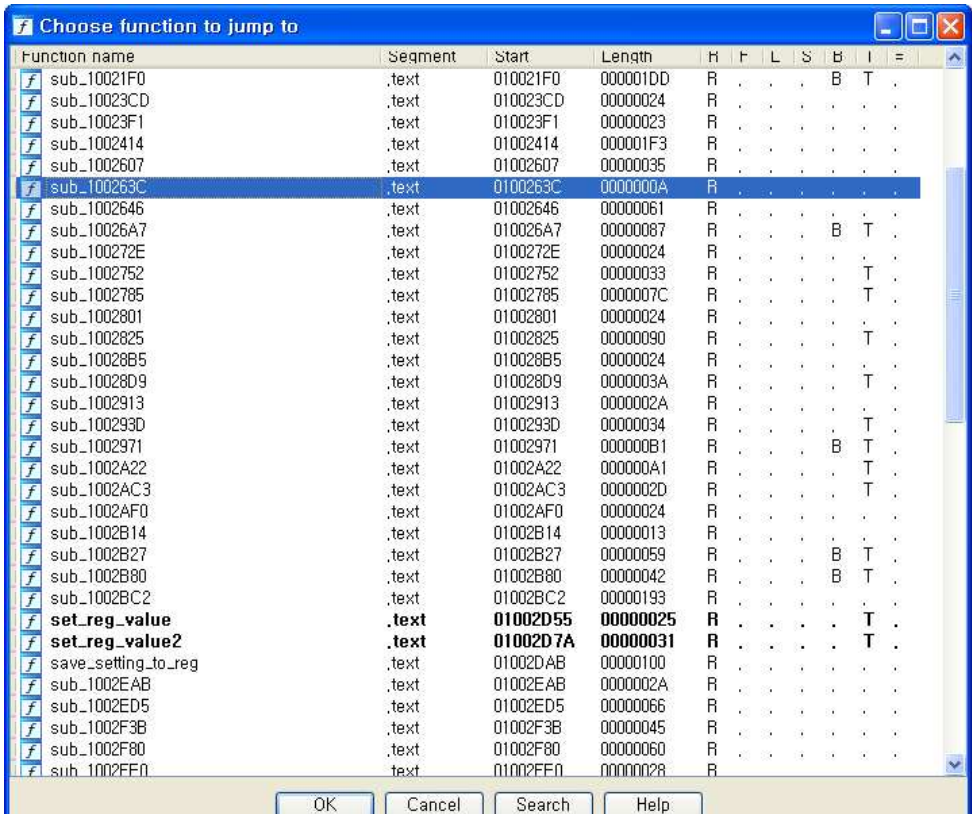
귀찮죠.(2번째 문제)
1번째 문제는 그래프 뷰에서 주소를 볼 수 있게 해서 해결할 수 있습니다.
메뉴 - Options - General - Disassembly 탭의 Line prefixes(우측에 있어요)를 체크하면 됩니다.



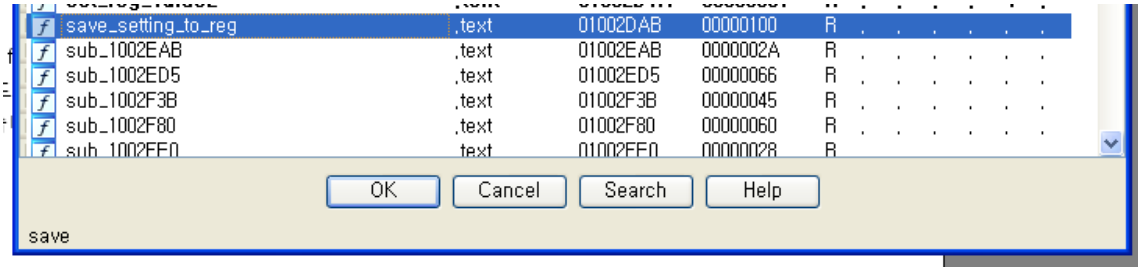
이걸 보고 메모장같은데다 시작주소와 끝 주소를 적은 후 창을 띄어서 옮겨 적으면 됩니다.
2번째 방법은 hexray 사에서 기능개선을 하지 않는이상 딱히 해결방법이 없습니다. 저는 암호,복호화 루틴같은 같은 중요한 함수에서만 레지스터 재명명을 해서 2번째 문제를 피합니다.
그리고 레지스터 재명명이 불가능한 때가 있습니다. 함수 영역이 아닌 부분에서입니다. 함수와 함수가 아닌 부분을 구별하는법은 간단합니다. 스페이스바를 눌러서 그래프 뷰가 되면 함수영역이고 그래프 뷰가 되질 않는다 하면 함수가 아닌 부분입니다.

이 함수는 함수 전체의 edi 가 0이므로 그냥 edi_0 으로 모두 바꾸겠습니다. 참고로 IDA의 이름 제약사항 때문에 그냥 0을 넣을순 없습니다. 저처럼 레지스터_다른이름 이런식으로 하는걸 추천합니다.

이제 아까 분석했던 save_setting_to_reg로 돌아갑니다. 설명을 위해 G,esc키 대신 다른 방법으로 가보겠습니다. Functions 창을 통해 가도 좋지만 Ctrl + P 를 눌러봅니다.

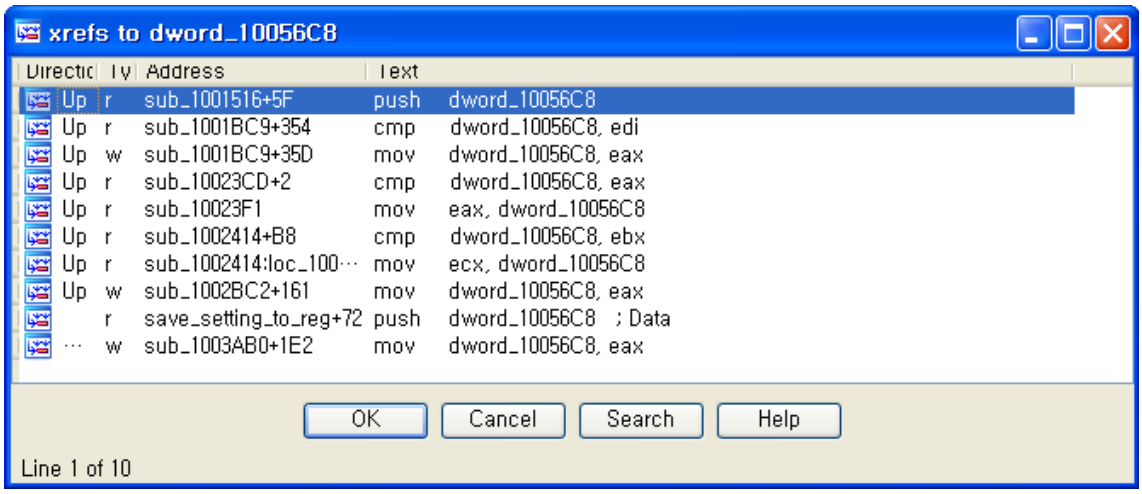


사실 이창은 functions 창과 완전히 동일합니다. 하지만 단축키로 열 수 있다는 점이 편합니다. 그리고 드래그를 해가면서 함수를 찾을 수 있지만 함수 이름을 알 경우 타이핑해서 찾을 수도 있습니다. 그냥 이 창에서(search 누르지 않은 상태로) save만 입력해 봅니다.



이런식으로도 함수를 바로 찾을 수 있습니다. 그리고 이 기능은 여기뿐만 아니라 strings 창, import, export 창 외 리스트가 나오는 창에는 모두 적용이 가능합니다.

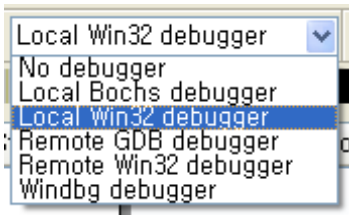
이동해보면 사실 아직 분석이 안된 부분이 있습니다. 바로 set_reg_value 의 Data 인자에 무엇이 있는지입니다. 이들을 xref to 로 보면 알겠지만 생각보다 여러군데에서 쓰입니다.



안에 무슨 데이터가 있는지 알아보려고 들어가보면 정체불명의 ? 만 나와있습니다. 이는 헥스 뷰로 봐도 마찬가지입니다.

?는 IDA에서 모르는 데이터를 나타낼때 쓰입니다. 여기서 말하는 모르는 데이터는 지금 리버싱 하는 파일에 아예 존재하지 않는 데이터를 말합니다. 이런 데이터는 IDA가 프로그램을 분석하면서 파일에 존재하지 않는다고 생각하는 부분에 접근하는 명령어를 바탕으로 이런 공간을 만들어서 이름을 붙여주고 ?? 로 채워놓습니다.
이런 곳에 어떤 데이터가 들어가는지는 프로그램을 실행시켜봐야지만 알 수 있습니다.

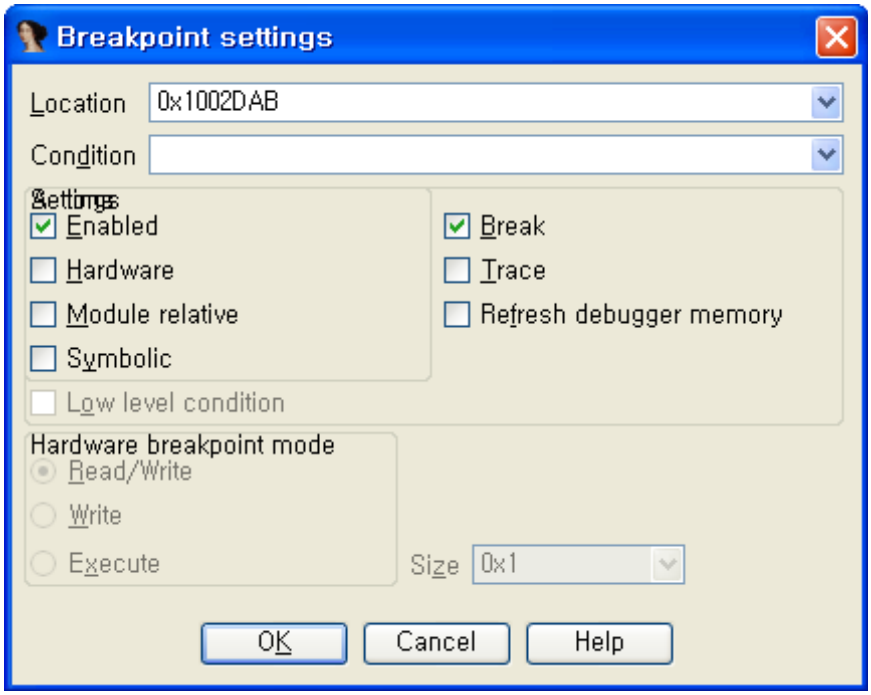
이 함수에서는 c_index 값 때문에 대충 Data의 용도를 알 수 있지만 여기서는 한번 프로그램을 실행시켜서 직접 볼것입니다.
우선 제일 먼저 설정해야 할것은 디버거 설정입니다.



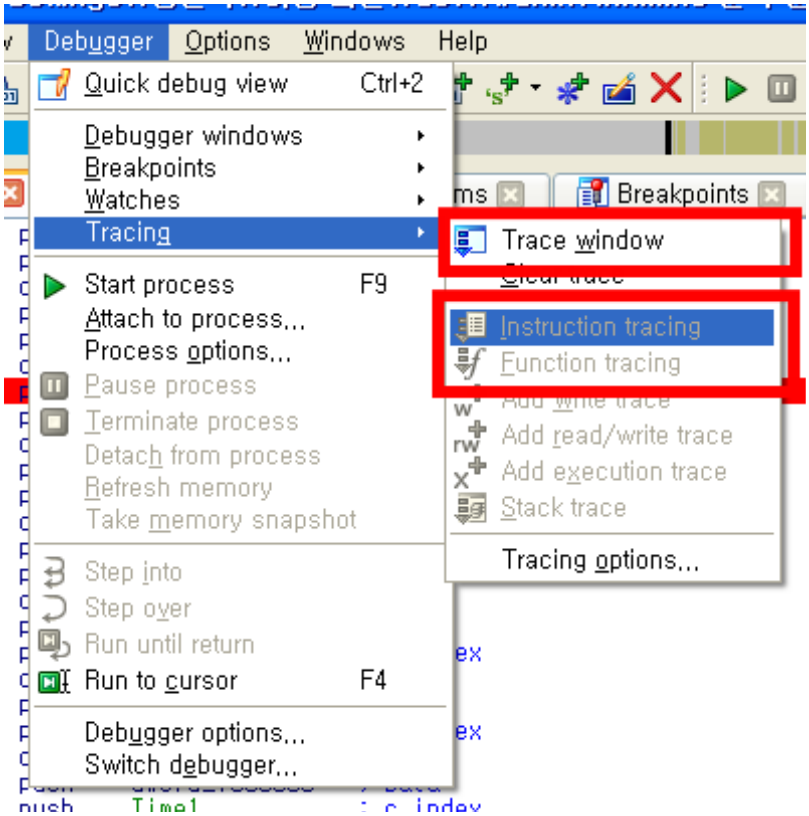
툴바나 메뉴 - Debugger - switch debugger 또는 select debugger 에 들어가서 선택할 수 있습니다.
local 은 현재 컴퓨터에서, remote는 외부(가상 컴퓨터 포함)에서 디버깅할 때 쓰입니다.
Bochs는 Vmware 같은 가상화 프로그램입니다. IDA에서는 모드에 따라 이 Bochs를 cpu 에뮬레이터로 쓰는데 코드만 에뮬레이팅 할 수 있습니다.
Bochs에대해 자세한건 http://www.hex-rays.com/products/ida/support/idadoc/1329.shtml 에서 볼 수 있습니다.
Remote에 관한건 3부에서 설명하겠습니다.

여기서는 현재 컴퓨터에서 디버깅하는 옵션인 Local Win32 debugger을 선택합니다. 그럼 Debugger 메뉴가 활성화 되면서 디버깅에 관련된 기능을 쓸 수 있습니다. 반대로 말하면 No debugger 상태이면 디버깅과 관련된 모든 기능은 쓸 수 없으니 참고하세요.
여기서 디버깅에 관련된 간단한 사항들을 설명하고 가겠습니다.
디버깅시 자주 쓰이는 단축키는 올리 디버거와 비슷합니다. F2는 브레이크 포인트(소프트) F7 은 step into F8은 step over 을 뜻합니다. 다만 F5는 IDA View 창에서는 디컴파일(hexray) 기능이고 IDA View EIP 창에서만 Run 기능입니다.

브레이크 포인트는 F2로 걸때 소프트 브레이크 포인트고 그 외 다른 브레이크 포인트를 걸려면 먼저 F2로 브포를 걸고 오른쪽 클릭 - Edit breakpoint 로 가야 합니다.

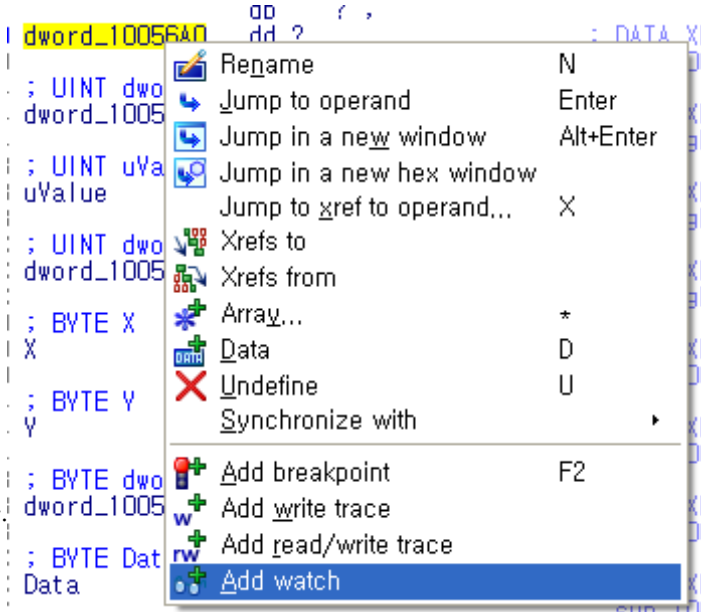


대부분의 것들은 그냥 옆에 나온 이름에 맞게 하면 됩니다. 그런데 제가 잘못 알고 쓰고 있는건지는 몰라도 소프트 브레이크 포인트인 상태에서 Trace 는 체크를 해도 다시 확인해보면 풀려 있습니다. 그리고 하드웨어 브레이크 포인트를 걸면 Trace 에 체크를 하지 않아도 다시 확인해 보면 자동으로 걸려 있습니다.
그리고 오른쪽 클릭 - Add xxxx trace 로 트레이싱을 걸면 빨간색으로 브포가 걸립니다.
edit breakpoint 로 설정을 살펴보면 하드웨어 브레이크 포인트가 걸려있는데(Excute 가 아닌 Read/Write로 걸려있음) Enabled 가 체크해제 되있고 Break 는 체크, 그리고 Trace가 체크 해제 되어 있습니다.(이게 버그인지는 모르겠습니다.) 하지만 트레이싱은 제대로 됩니다.



트레이스 창은 기본으로 뜨는 창이 아니기 때문에 트레이싱한 것을 보려면 메뉴 - Debugger - Trace window 에 들어가서 보면 됩니다.
아래 Instriction 과 Function tracing 은 디버깅 중에만 활성화되는데 말 그대로 모든 명령어 또는 모든 함수를 트레이싱 합니다.
아래 Tracing options 에 들어가면 옵션창이 뜨는데 여기서 상황에 맞게 설정하면 됩니다. (대부분 건드릴 필요가 없음)
그리고 그 바로 위에있는 stacck trace 는 다름이 아닌 콜스텍 창입니다. 이 창은 프로세서가 브포에 걸려서 멈춘 상태인데도 아무것도 나오지 않는 버그가 가끔 발생하는데 이때는 창을 꺼주고 다시 키면 잘 보입니다.

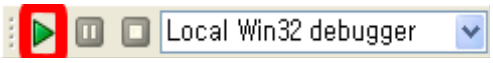
위에 메뉴중 watches (메뉴 - watches) 는 말 그대로 보는 기능입니다. 데이터 영역에서 watch 하기를 원하는 데이터를 오른쪽 마우스로 클릭해서 watch 목록에 추가할 수 있습니다.



watch 리스트는 기본 창이 아니기 때문에 리스트를 보려면 메뉴 - Debugger - Watches - Watch list 로 켜줘야 합니다.

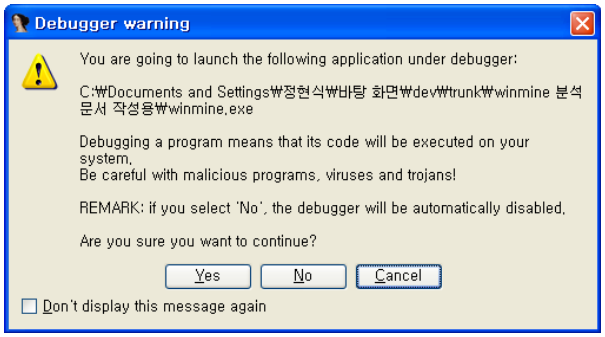
참고로 watch는 리버서가 보고 싶은 값들을 단순히 한곳으로 모으는 기능입니다. 그래서 리스트에 있는 값들은 디버깅시 항상 실시간으로 반영되는게 아니라 디버거에게 프로그램 제어권이 있을때에만 실시간으로 반영됩니다. 그리고 보통 쓰는 단축키(X, N 등등)가 먹히지 않고 더블클릭해도 해당 영역으로 이동하지 않습니다.

이제 어느정도 기본적인 디버깅에 관한 설명은 끝났습니다. 다시 리버싱으로 돌아가서 원래 하려고 했던 set_reg_value 에 들어가는 Data 의 값들을 직접 볼 차례입니다. 모두 다 보기에는 시간이 많이 걸리므로 각각에 해당하는 이름에 맞는지만 확인하기 위해 하나만 선택해서 보겠습니다.

저는 c_index 로 Height가 들어갈때의 Data인 uValue에 Read/Write trace를 걸겠습니다. 그리고 F9를 누르거나 

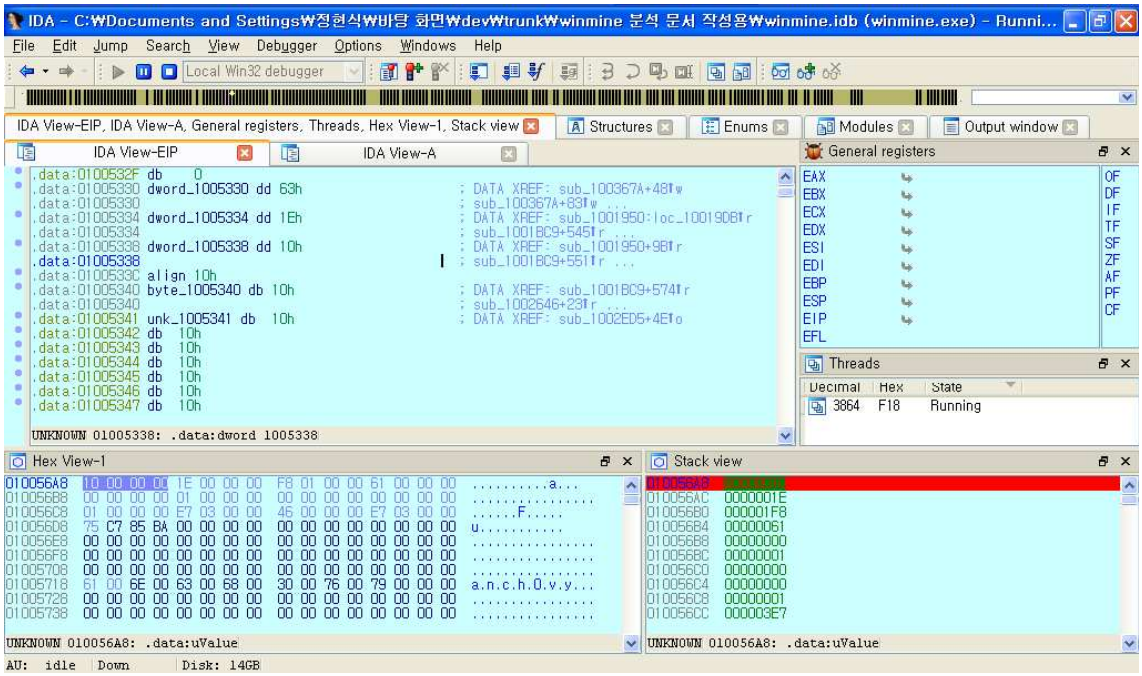
를 눌러서 프로그램을 실행시킨 후 트레이스 창으로 가봅니다.

이때(프로그램을 실행시키기 바로 전) 이런 창이 뜰것입니다.



이 창은 프로그램을 실행시키겠냐는 마지막 확인창입니다. 이런 창을 띄우는 이유는 이 프로그램이 악성코드일 경우를 위해서입니다. 이 창을 보기 싫으면 아래 체크를 해서 띄우지 않을 수 있습니다. 하지만 저같은 경우 만약의 경우(악성코드 분석중 F9라든지...)를 위해서 체크를 하지 않고 있습니다.

실행을 해보면 IDA의 전체적인 창 모양이 바뀔 것입니다. IDA가 디버깅 모드일때만 나타나는 창입니다. 그런데 이 창 역시 매우 불편합니다. 그래서 디버깅을 하기 전에 자신의 입맛에 맞게 창을 변경한 후 메뉴 - Windows - Save desktop에서 defalt 설정을 하고 저장하는 걸 추천합니다.
아래는 제가 쓰는 창입니다. (넷북이라 화면이 무지 작습니다 ㅠㅠ)



참고로 제가 강력 추천하는 것이 2개 있는데 하나는 툴바의 Jump 버튼입니다. 툴바에서 오른쪽 클릭해서 Jump 를 누르면 좌측 상단에 나옵니다. 기능은 esc 기능과 똑같은 버튼인데 앞으로도 갈 수 있습니다.
그리고 나머지 하나는 IDA View 창입니다. IDA View EIP 창이 나와있는 상태에서 ctrl + 1 로 IDA view 창을 띄우는 위 화면처럼 켜지는데 생각보다 무지 편합니다.
그리고 하나 알아야 할 점이 있는데 디버깅 상태에서는 Hex view 창과 IDA view창은 기본적으로 synchronize 되어 있지 않습니다.

이제 트레이싱창을 보면 사람에 따라 약간 다른 값이 들어갈 것입니다. 저같은 경우는 0x10(16) 이 들어가 있습니다. c_index값(Height)를 바탕으로 16이란 값의 의미를 유추해보면 지뢰찾기 판의 높이라는걸 알 수 있습니다.

사실 설명을 위해서 이런식으로 데이터의 의미를 유추하였지만 실제로는 프로그램을 많이 이해하지 않고서는 거의 불가능한 방법입니다. (게다가 억지성이...) 현실적인 방법은 이따 3부에서 설명하겠습니다.

이제 IDA의 기본적인 설명을 한 챕터 1은 끝났습니다. 사실 '기본적인' 기능을 설명했기보다는 지뢰찾기의 함수 딱 2개를 리버싱 하면서 쓰이는 기능들만 설명하긴 했습니다. 하지만 이것만 알아도 올리디버거를 쓸때보다는 훨씬 '분석' 하기 편할것입니다.
이제 챕터 2에서는 챕터 1에서 설명하지 못했던 기능들을 설명할 것입니다.
챕터 1처럼 예제위주가 아닌 챕터 2부터는 기능 설명 주 목적으로 하겠습니다. 그리고 때에 따라서 제가 만든 예제들을 가져오겠습니다.

Chapter 2

그 외 IDA의 기능들

-Hex-Rays decompiler

IDA의 강력한 기능중 하나입니다. 하지만 헥스레이는 플러그인이기 때문에 기본적으로 IDA에 포함되어 있는것이 아니라 따로 구입해야 합니다.(같이 구매시 50% 할인)

Part#	Product	Price	Quantity	Total
MS Windows				
IDASTACW	IDA Starter Base Computer License (MS Windows)	819 USD		
IDAPROCW	IDA Pro Base Computer License (MS Windows)	1589 USD		
HEXX86W	x86 Decompiler Base License (MS Windows)	2239 USD		
HEXARMW	ARM Decompiler Base License (MS Windows)	2240 USD		
HEXALLW	x86+ARM Decompiler Base License (MS Windows)	3359 USD		
Linux				

사용법은 F5입니다. 챕터 1에서 분석했던 save_setting_to_reg 함수로 가서 디컴파일 한 결과입니다.

```
LSTATUS __thiscall save_setting_to_reg(DWORD this)
{
  DWORD dwDisposition; // [sp+0h] [bp-4h]@1

  dwDisposition = this;
  RegCreateKeyExW(HKEY_CURRENT_USER, L"Software\\Microsoft\\winmine", 0, 0, 0, KEY_WRITE, 0, &hKey, &dwDisposition);
  set_reg_value(0, dword_10056A0);
  set_reg_value(Height, uValue);
  set_reg_value(Width, dword_10056AC);
  set_reg_value(Mines, dword_10056A4);
  set_reg_value(Mark, Data);
  set_reg_value(AlreadyPlayed, 1u);
  set_reg_value(Color, dword_10056C8);
  set_reg_value(Sound, dword_10056B8);
  set_reg_value(Xpos, X);
  set_reg_value(Ypos, Y);
  set_reg_value(Time1, dword_10056CC);
  set_reg_value(Time2, dword_10056D0);
  set_reg_value(Time3, dword_10056D4);
  set_reg_value2(Name1, &byte_10056D8);
  set_reg_value2(Name2, &word_1005718);
  set_reg_value2(Name3, &word_1005758);
  return RegCloseKey(hKey);
}
```

또 디컴파일된 코드 또는 어셈블리어 코드에서 Tab 키를 누르면 서로 해당하는 코드로 이동을 하게 됩니다.

예) set_reg_value(Sound, dword_10056B8); <--> push dword_10056B8 ; Data
push Sound ; c_index
call set_reg_value

그리고 코드 전체를 디컴파일 하려면 CTRL + F5 로 하면 됩니다. 그리고 디컴파일은 함수 부분에서만 가능합니다.

그리고 디컴파일 창에서 오른쪽 버튼을 눌러서 나오는 것중 copy to assembly가 있습니다. 별로 추천하고 싶지는 않은 메뉴입니다. 하게되면 다음과 같이 디스어셈블 창이 변합니다.

```
01002DD2 ; 6:   set_reg_value(0, dword_10056A0);
01002DD2 movzx  eax, word ptr dword_10056A0
01002DD9 push   eax             ; Data
01002DDA push   esi             ; c_index
01002DDB call   set_reg_value
01002DE0 ; 7:   set_reg_value(Height, uValue);
```

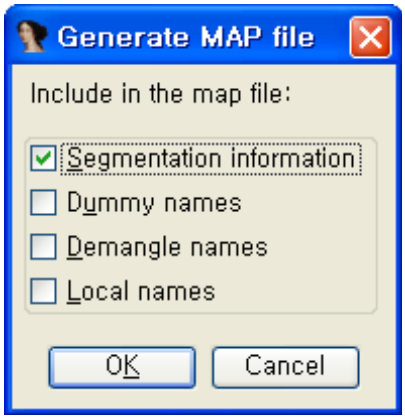
그리고 이걸 다시 없애려면 메뉴 - Edit - comment - delete pseudocode comments 를 선택하면 됩니다.

-New instance

메뉴 - File - New instance 로 IDA를 하나 더 열 수 있습니다.

-Produce File

메뉴 - File produce file 에 가보면 IDA에서 여러 가지 파일을 생성할 수 있습니다.
이중 MAP 파일은 프로그램에 있는 여러 정보가 들어있는 파일입니다. create MAP file 을 선택하고 저장위치를 정하면 다음과 같은 창이 뜹니다.



여기서 Local names 를 선택해야 IDA에서 설정한 이름들이 파일로 출력됩니다. 저같은 경우 IDA에서 분석을 하고 올리디버거에서 디버깅을 할 때 모든 정보를 담아(4개 모두 체크) MAP 파일로 생성후 올리디버거 플러그인 GODUP.dll 로 로드해서 사용합니다.
ASM은 말 그대로 어셈블리어로 된 파일을 만들어 줍니다. 아무것도 선택하지 않았으면 파일 전체를 어셈블리어로 출력하고 드래그가 되어있다면 그 부분만 출력해 줍니다.
INC는 구조체와 enums 정보가 정의된 파일을 출력합니다.
LST는 IDA 디스어셈블 창의 내용을 그대로 출력합니다. ASM과 같이 드래그를 하지 않으면 모든 파일의 내용을 출력합니다.
EXE는 말 그대로 실행파일로 출력해 줍니다. 하지만 이 기능은 지금 분석하는 파일과 CPU 타입에 따라 비활성화되며 대부분의 경우가 이렇습니다.

지뢰찾기조차 EXE로 다시 못만들겠다고 하는 IDA
DIF는 수정된 바이트 정보를 출력해 줍니다. 형태는 주소: 원본바이트 바뀐바이트 로 매우 간단합니다.
HTML 은 다름이 아니라 LST 파일을 HTML을 사용해서 색을 입혀서 실제 IDA창과 비슷하게 만든 파일입니다.
그 아래 두 개의 GDL 파일을 생성하는것은 그래프를 생성하는 것입니다. IDA에서는 내장된 그래프 뷰를 가지고 있는데 이 그래프 뷰 대신 다른 그래프 뷰 프로그램으로 보려 하거나 다른사람에게 그래프를 보내려 할때 사용합니다. 참고로 GDL은 Graph Description

Language입니다.
C header은 enums 나 구조체를 C언어 형태로 출력해 줍니다.

-Code 와 Undefine
각각 단축키는 C와 U입니다. 이 기능은 언패킹이나 난독화에서 유용하게 쓰입니다. 간단한 프로그램 하나를 대상으로 해보겠습니다.
프로그램은 다음과 같습니다. (대략적인것만, 위에서 받은 첨부파일중 code_undefine.exe를 열면 됩니다)

```
call test
db e8 // 데이터!!
mov eax,1
mov ebx,2
mov ecx,3
mov edx,4

test
inc dword ptr [esp]
```

test 함수가 리턴 어드레스를 1 올리는 역할을 합니다. 그럼 dd e8은 넘어가버리게 되고 mov eax,1 이 실행이 됩니다. 하지만 보통의 경우는 리턴 어드레스를 조작하지 않으므로 call 다음부터 디스어셈블을 하게 되면서 dd e8 을 코드로 인식하게 됩니다.(참고로 e8 은 call명령어의 1번째 바이트입니다, 항상은 아님) 다음은 IDA가 디스어셈블한 것입니다.

```
public start
start proc near
call    sub_40106F
call    near ptr word_4011C2
add     [ebx+2], bh
mov     ecx, 3
mov     edx, 4
retn
start endp
```

사실 이걸 보고 저게 잘못된 디스어셈블이란걸 알기는 쉽지 않지만 디버깅을 하면서 step 해보면 바로 알 수 있겠죠.
그럼 이제 IDA에게 저게 데이터라고 알려야 합니다. 이럴때 쓰는게 Code 와 Undefine입니다.
먼저 Call near ptr word_4011c2 를 클릭하고 U를 누릅니다. (리스트 뷰 모드에서 하세요)

```
        call    sub_40106F
-----------------------------------------
        db 0E8h ;
        db 0B8h ;
        db    1
        db    0 ;  OFF32 SEGDEF [0,BB000000]
        db    0
        db    0
        db 0BBh ;
        db    2
        db    0
        db    0
        db    0
        db 0B9h ;
        db    3
        db    0
        db    0
        db    0 ;  OFF32 SEGDEF [0,4BA00]
        db 0BAh ;
        db    4
        db    0
```

그다음 데이터인 db 0e8h 다음 db 0b8h를 클릭하고 C(code)를 눌러줍니다.

```
                public start
start:
                call    sub_40106F
; ---------------------------------------
                db 0E8h
; ---------------------------------------
                mov     eax, 1
                mov     ebx, 2
                mov     ecx, 3
                mov     edx, 4
                retn
; ---------------------------------------
```

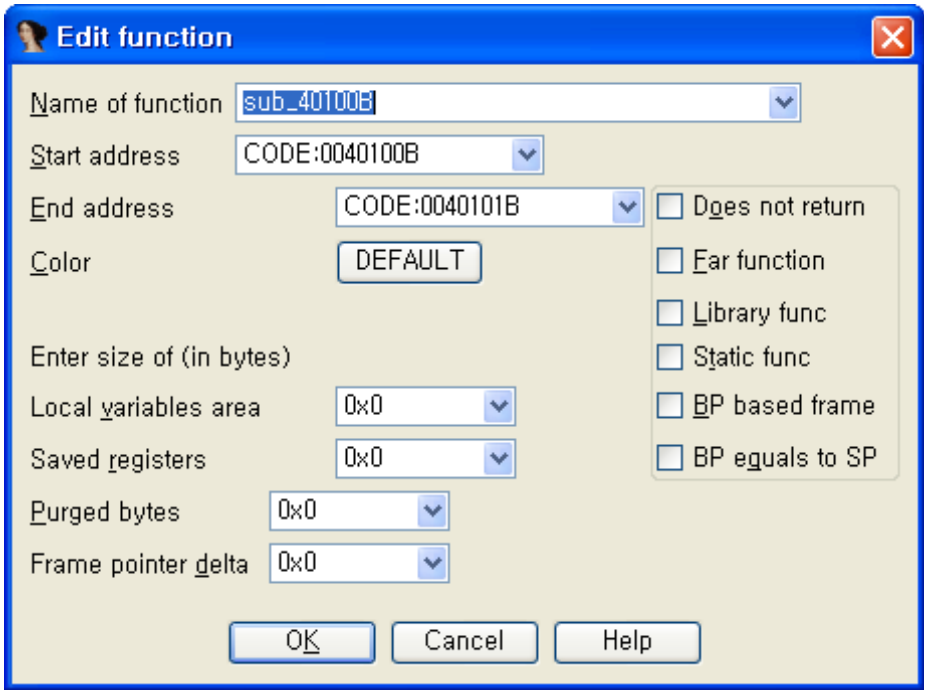
이제 제대로 된 디스어셈블 결과가 나와습니다.
이런식으로 쓸수도 있고 언패킹시 패킹이 풀린 위치의 시작점에서 C(code)를 눌러서 IDA에게 코드로 분석하고 할 수도 있습니다.

-Create function 와 edit function

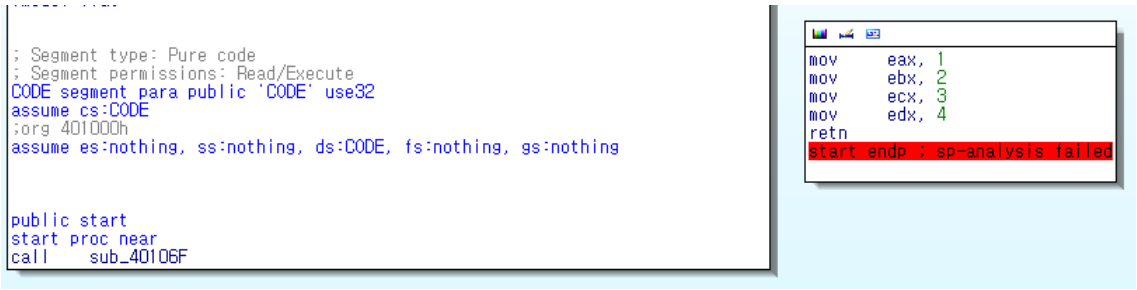
여기서는 바로 위에서 썻던 프로그램을 그대로 씁니다. 제대로 했다면 바로 위 사진과 같은 상태가 될 것입니다. 하지만 그래프 뷰도 되질 않고 그래프 뷰로 보면 저 둘이 연결됬다는 걸 알 수가 없습니다. 이때 Create function 기능을 사용하면 됩니다.
이 기능은 함수가 아닌 부분을 함수로 만드는 역할을 합니다. 위 프로그램의 경우 리버서가 undefine 을 하고 다시 코드로 만들었기 때문에 함수가 아닌 상태입니다. 물론 그래프 뷰 또한 불가능한 상태입니다.
이 상태에서 mov eax,1 을 클릭하고 단축키 P를 누르면 IDA가 00401006부터 0040101A까지 함수로 만들어 줍니다.

```
sub_40100B      proc near
                mov     ebx, 2
                mov     ecx, 3
                mov     edx, 4
                retn
sub_40100B      endp
```

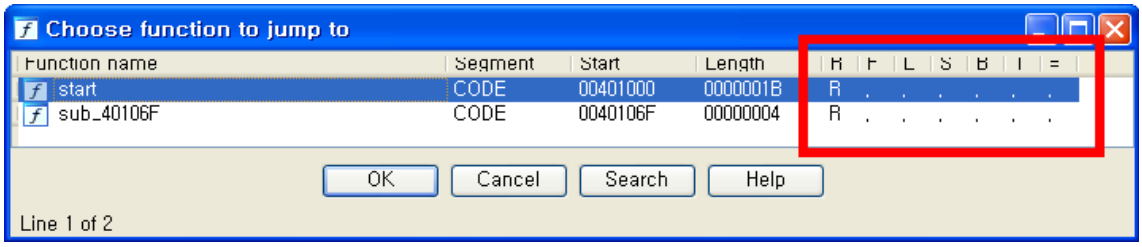
이 상태에서 오른쪽 클릭 - edit funcion 으로 들어갑니다.



여기서 시작 주소에 call sub_40106F 가 있는 주소인 00401000을 넣어주고 OK버튼을 누르면 둘이 한 함수로 묶이게 됩니다.



위에 edit function을 좀더 설명하겠습니다.
color 는 함수 전체 블록의 바탕색을 지정할 수 있습니다. 그리고 어떤 값을 쓸 수 있는 4개의 칸은 함수의 스텍에 관한 설정입니다. 하지만 대부분의 프로그램이 fastcall,stdcall,cdecl 등 평범한 콜링 컨벤션(http://en.wikipedia.org/wiki/Calling_convention 참조, 영문)을 쓰기 때문에 따로 만져줄 필요가 별로 없어서 설명을 하지 않겠습니다.
오른쪽에 체크할수 있는 6개의 옵션은 함수의 특성을 나타냅니다. 이 6개의 옵션이 function 창 오른쪽에 있던 6개의 칸의 정체입니다.



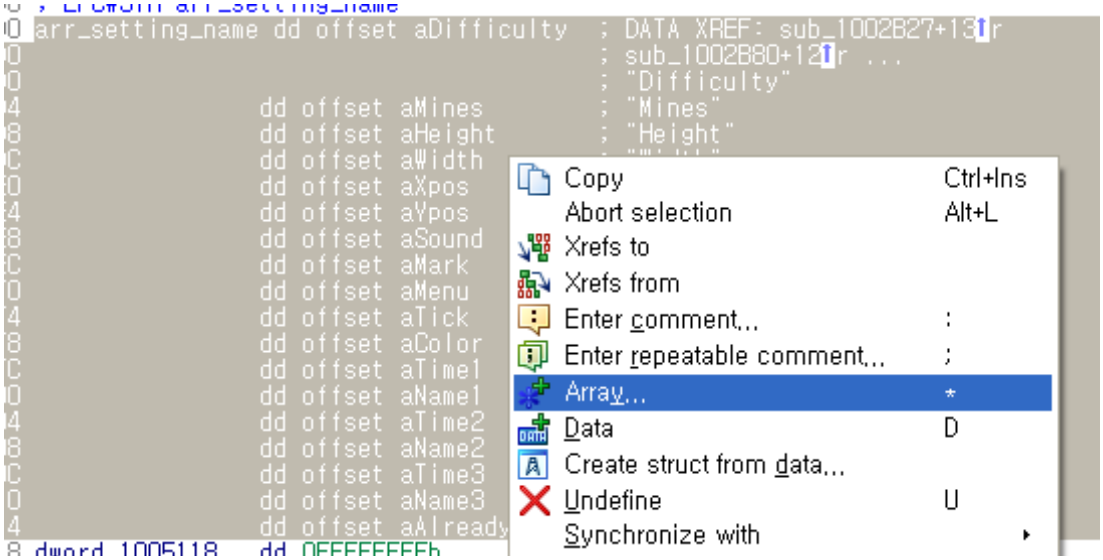
그런데 이 6개 옵션은 대부분 IDA가 알맞게 알아서 체크해주기 때문에 이역시 설명하지 않겠습니다.

-output 창의 용도

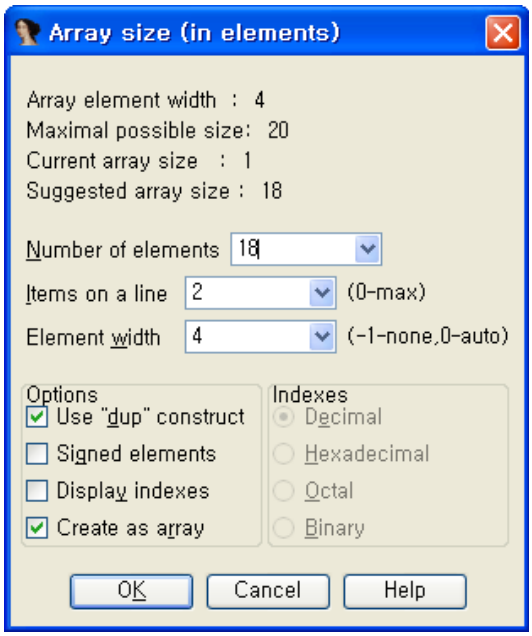
지금까지 저같은 경우는 output 창은 모두 아래에서 탭으로 옮겨버렸습니다. 하지만 기본 위치가 화면의 하단인 이유가 있습니다. output 창은 각종 알림과 에러와 플러그인의 결과가 나오는 창입니다. 그리고 하단의 IDC 칸에는 IDA만의 명령어를 입력할 수 있습니다. 그렇기에 하단에다 놓고서 에러를 보는게 좋습니다. IDA의 에러는 알림창이 나오는 경우가 있지만 아무것도 나오지 않고 output 창에 에러한줄 뿜고서 끝나는 경우가 있기 때문입니다. 하지만 화면이 좁다면 탭으로 옮겨놓고 쓰면 좋습니다. 참고로 아예 끌수는 없습니다. (버그인지 모르겠지만 꺼도 ctrl + tab키를 누르면 계속 나타남)

-배열와 dup의 의미

챕터 1에서 분석했던 지뢰찾기를 열어서 arr_setting_name(010050D0)으로 갑니다. 비슷한 용도의 값이 연속으로 나와있으니 이걸로 배열로 만들어 보겠습니다.



위 사진처럼 드래그를 하고 오른쪽 클릭 - Array를 선택합니다.

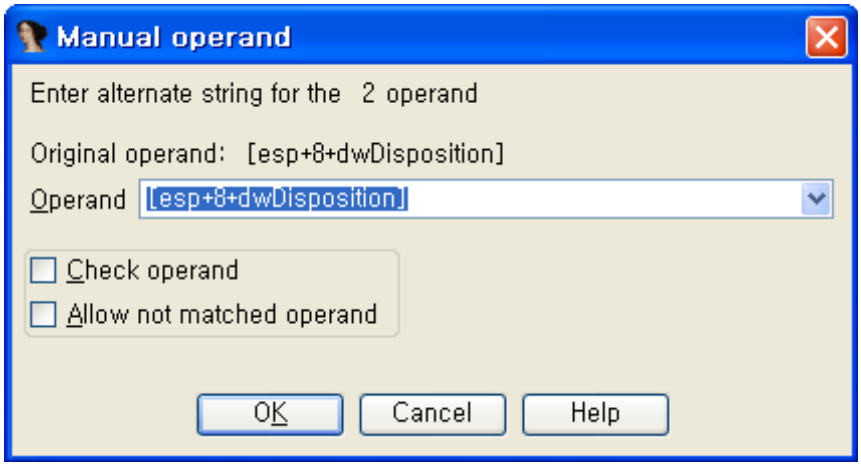


대부분 IDA가 알아서 값을 넣어주며 사진에는 items on a line 이 2로 되어있는데 기본값은 0으로 되어 있습니다. 여기서 max값은 ida.cfg 파일에 설정되어 있는 값을 가져다가 쓰게 됩니다.
아래 옵션중 dup 을 사용하는 옵션이 있습니다. dup은 duplication 의 약자로 보입니다. 사용 형태는 다음과 같습니다.
db(또는 dw,dd) 중복된 수(hex로) dup (중복된 값)
예로 0 이 10번 반복되면 다음과 같이 나타내 집니다. ---> db 0Ah dup(0)

위 사진에서 OK버튼을 누르면 다음과 같이 깔끔하게 나옵니다.

```
arr_setting_name dd offset aDifficulty,offset aMines
                                        ; DATA XREF: sub_1002B27+13↑r
                                        ; sub_1002B80+12↑r ...
                 dd offset aHeight,offset aWidth ; "AlreadyPlayed"
                 dd offset aXpos,offset aYpos
                 dd offset aSound,offset aMark
                 dd offset aMenu,offset aTick
                 dd offset aColor,offset aTime1
                 dd offset aName1,offset aTime2
                 dd offset aName2,offset aTime3
                 dd offset aName3,offset aAlreadyplayed
```

-Manual operand
IDA에서는 다양한 방식으로 오퍼렌드를 나타냅니다. 그런데 그중에서도 자신이 원하는 오퍼렌드가 없으면 리버서가 직접 오퍼렌드를 수정할 수 있습니다.
메뉴 - operand type - manual 에 들어가면 됩니다.



기본값은 Check operand 에 체크되어 있는데 이걸 해제하면 리버서가 원하는 오퍼랜드를 아무거나 넣을 수 있습니다. 심지어 이런것도 가능합니다.

```
lea     eax, [esp+8+dwDisposition] ---> lea     eax, 멸치 바보
```

다시 되돌리려면 공백을 넣고 설정하면 됩니다.

-Manual instruction

위의 매뉴얼 오퍼랜드의 확장판이라 봐도 됩니다. 명령어를 통째로 리버서가 원하는 걸로 대신 표현할 수 있습니다.

```
안녕                    ; cbData
멸치
push    eax             ; lpData
```

어디 블로그를 가보니 이 기능을 마치 명령어 수정 기능처럼 설명하고 있던데 사실은 표현법만 바꾸는 것입니다. (사실 제가 전에 어디가서 낚여서 이게 명령어 수정이라 블로그에 게제를....)

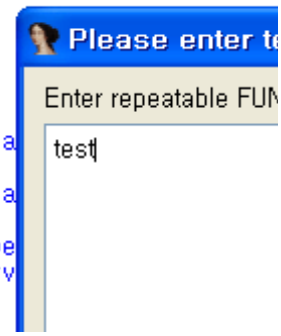
-여러가지 주석들

IDA에는 6개의 주석이 있습니다. 그중 주소 쓰이는 것은 2개입니다. 그것만 설명하겠습니다.
먼저 일반 주석입니다. 단축키는 : 이며 말 그대로 일반적인, 코드 오른쪽에 나오는 주석입니다. 주의할점은 이 주석을 넣을때 꼭! SHIFT + ; 로 해야 일반 주석이 들어갑니다.
다음은 반복 주석입니다. 단축키는 ;입니다. 일반주석 넣을때 실수로 SHIFT를 안눌렀다면 이 반복주석이 들어가게 됩니다.
함수의 시작점에 있는 반복주석이 그 함수를 나타내는 주석이 됩니다.
무슨말인지는 아래를 보면 압니다.

```
set_reg_value proc near

c_index= dword ptr  4
Data= byte ptr  8

push    4               ; cbData
lea     eax, [esp+4+Data]
push    eax             ; lpData
mov     eax, [esp+8+c_index]
push    4               ; dwType
push    0               ; Reserv
push    arr_setting_name[eax*4]
push    hKey            ; hKey
```



set_reg_value 의 첫 번째 명령어인 push 4에 반복주석 test를 넣었습니다. 그리고 호출부로 가봅니다.

```
call    set_reg_value   ; test
push    uValue          ; Data
push    Height          ; c_index
call    set_reg_value   ; test
push    dword_10056AC   ; Data
push    Width           ; c_index
call    set_reg_value   ; test
push    dword_10056A4   ; Data
push    Mines           ; c_index
call    set_reg_value   ; test
```

이렇게 회색으로 반복주석이 모두 나타납니다. 주석을 다시 없애려면 주석을 넣은 위치에 가서 빈칸으로 만들고 OK를 누르면 됩니다.
참고로 종류가 다른 주석은 같은 코드 위치에 동시에 올 수 있습니다. 이때 다 나타나는건 아니고 우선순위가 젤 높은것이 나타나는데 사용자가 넣은 일반주석이 우선순위가 제일 높습니다.

-플러그인
플러그인을 쓰는법은 간단합니다. 다운받은 플러그인을 IDA 폴더에 가서 plugin 폴더에 넣은 후 IDA를 키고 Edit - Plugin에 가서 선택하면 됩니다.

-Mark position
리버서가 자주 이동하는 부분을 마킹하고 마킹한 부분으로 이동하는 기능입니다.
마크를 찍을때는 alt + M 마크로 이동할때는 ctrl + M입니다. 둘다 메뉴 - Jump 에 있습니다. 그리고 해당 위치에 가서 마크 설명을 공백으로 해도 지워지지 않습니다.(no description 이라 나옴) 마크를 지우려면 메뉴 - jump - clear mark 에 들어가서 지워야 합니다.