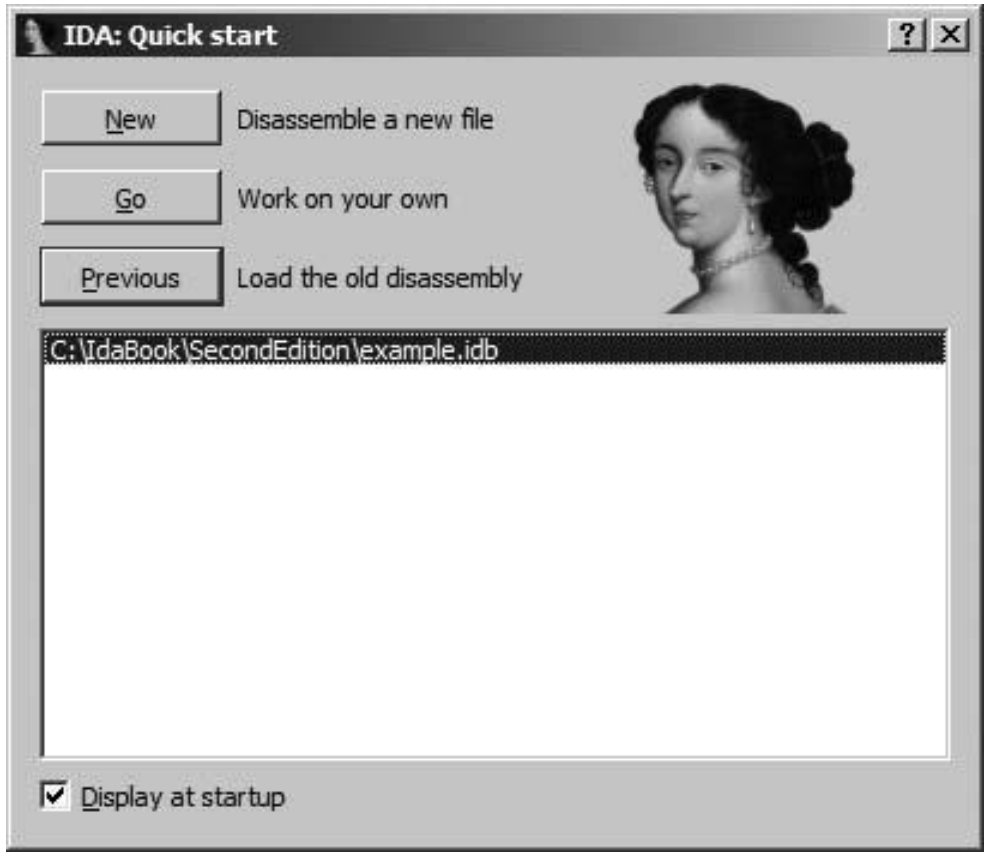
# Getting started with IDA

Best tool for Reversing!

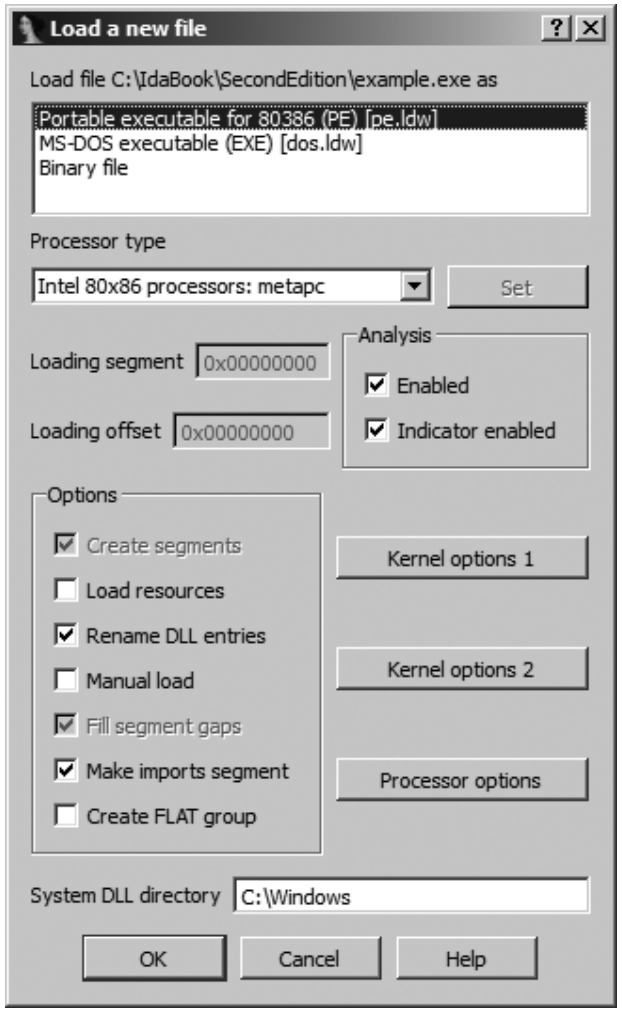
# Launching IDA



# Launching IDA

- New
  - Choosing New opens a standard File Open dialog to select the file to be analyzed.
- Go
  - Open IDA with an empty workspace.
  - Drag and drop
  - Or use the options open a file in menu
- Previous
  - when you wish to open one of the files in the list of recent files

# IDA File Loading



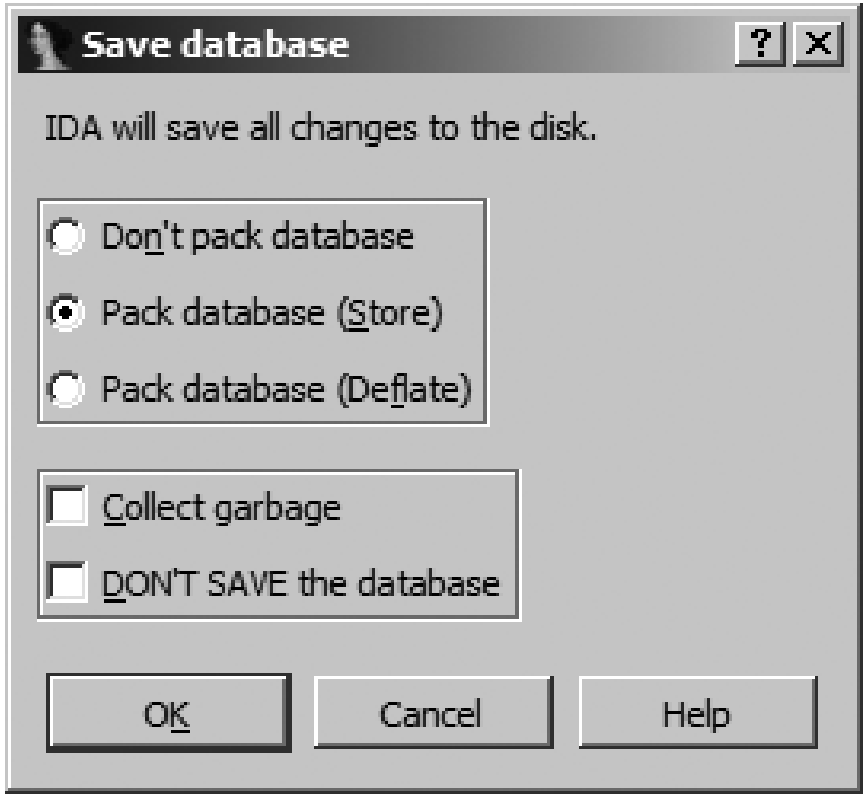
# IDA File Loading

- IDA generates a list of potential file types and displays that list at the top of the dialog.
- Usually we will choose the default one. PE.
- The Processor Type drop-down menu allows you to specify which processor module.
- Just don’t touch anything and let it be the default setting.

# IDA Database Files

- After loading the file in IDA
- IDA create IDA database in the folder of the binary reside.
- Each with a base name matching the selected executable and whose extensions are .id0, .id1, .nam, and .til.
- Don’t worry about this.

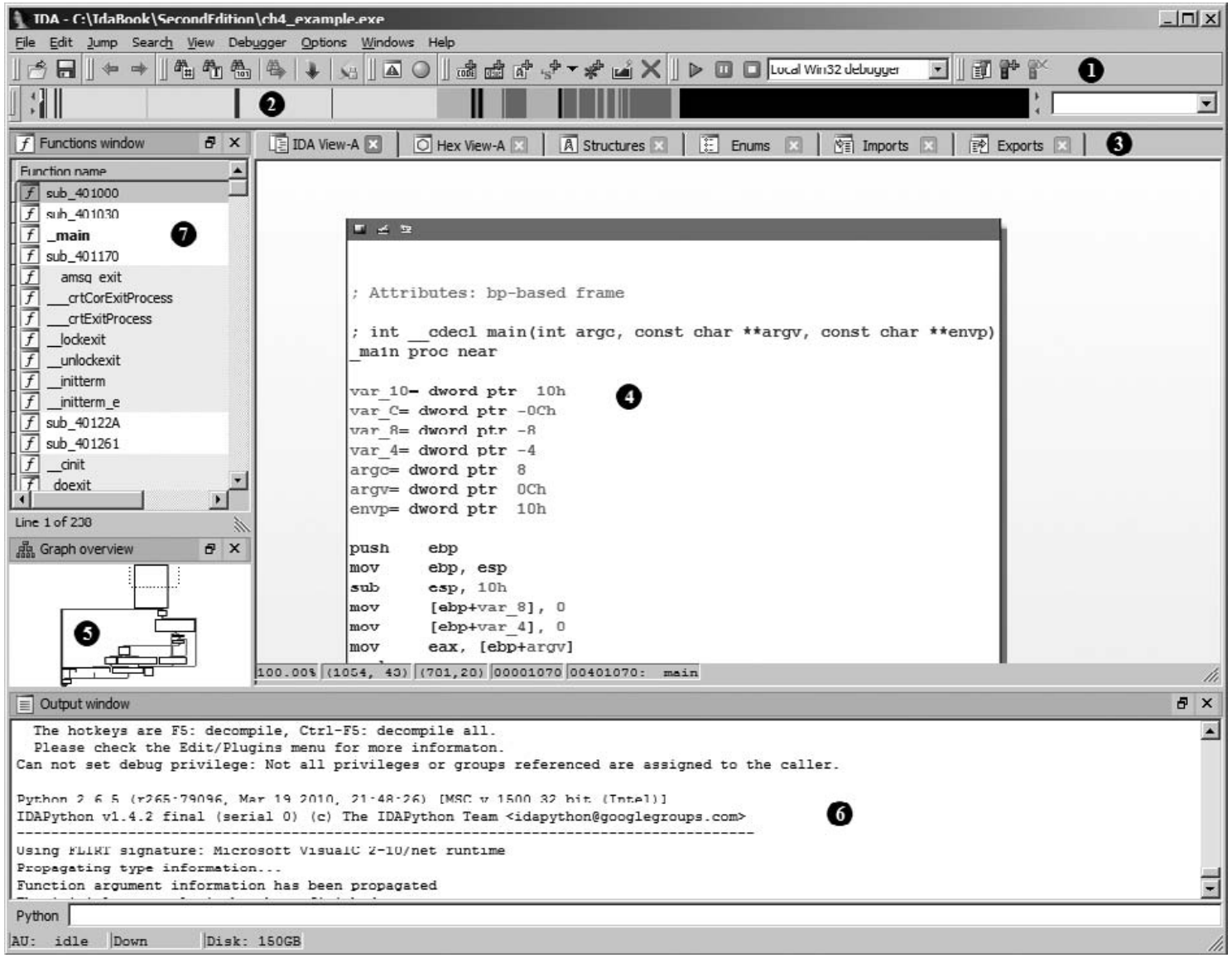
# Closing IDA Databases



# Closing IDA Databases

- you will be presented with the Save Database dialog when we close IDA
- The available save options and their associated implications are summarized in the following list:
  - Pack database (Store)
    - Selecting the Store option results in the four database component files being archived into a single IDB file.
  - DON'T SAVE the database
    - Not to save our work

# Introduction to the IDA Desktop


IDA - C:\IdaBook\SecondEdition\ch4_example.exe
File Edit Jump Search View Debugger Options Windows Help
Local Win32 debugger
Functions window
Function name
sub_401000
sub_401030
_main
sub_401170
amsg_exit
__crtCorExitProcess
__crtExitProcess
__lockexit
__unlockexit
__initterm
__initterm_e
sub_40122A
sub_401261
__cinit
doexit
Line 1 of 238
Graph overview
IDA View-A
Hex View-A
Structures
Enums
Imports
Exports
; Attributes: bp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near
var_10= dword ptr 10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
push ebp
mov ebp, esp
sub esp, 10h
mov [ebp+var_8], 0
mov [ebp+var_4], 0
mov eax, [ebp+argv]
100.00% (1054, 43) (701,20) 00001070 00401070: main
Output window
The hotkeys are F5: decompile, Ctrl-F5: decompile all.
Please check the Edit/Plugins menu for more informaton.
Can not set debug privilege: Not all privileges or groups referenced are assigned to the caller.
IDAPython v1.4.2 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
Using FLIRT signature: Microsoft VisualC 2-10/net runtime
Propagating type information...
Function argument information has been propagated
Python
AU: idle Down Disk: 150GB
1
2
3
4
5
6
7

# 1. Toolbar area

- Contains tools corresponding to the most commonly used IDA operations.
- Toolbars are added to and removed from the desktop using the View > Toolbars command.
- Using drag-and-drop, you can reposition each of the toolbars to suit your needs.

# 2. Overview navigator

- The navigation band presents a linear view of the address space of the loaded file.
- You can zoom in and out of the address range by right-clicking anywhere within the navigation band and selecting one of the available zoom options.
- Different colors represent different types of file content, such as data or code.
- A small current position indicator (yellow by default)
- Clicking the navigation band jumps the disassembly view to the selected location within the binary.

# 3. Tabs

- Tabs are provided for each of the currently open data displays.
- Data displays contain information extracted from the binary.
- The majority of your analysis work is likely to take place through interaction with the available data displays.
- Additional data displays are available via the View?Open Subviews menu

# 4. Disassembly view

- Primary data display
- Two primary view
  - Graph view (default)
  - Listing view
- In graph view, IDA displays a flowchart-style graph of a single function at any given time.
- Easy to gain an understanding of the flow of the function

# 5. Graph overview

- Fit entire graph of a function into the display area at one time.

# 6. Output window

- Where you can expect to find any informational messages generated by IDA
- Here you will find status messages concerning the progress of the file-analysis phase, along with any error messages resulting from user-requested operations.

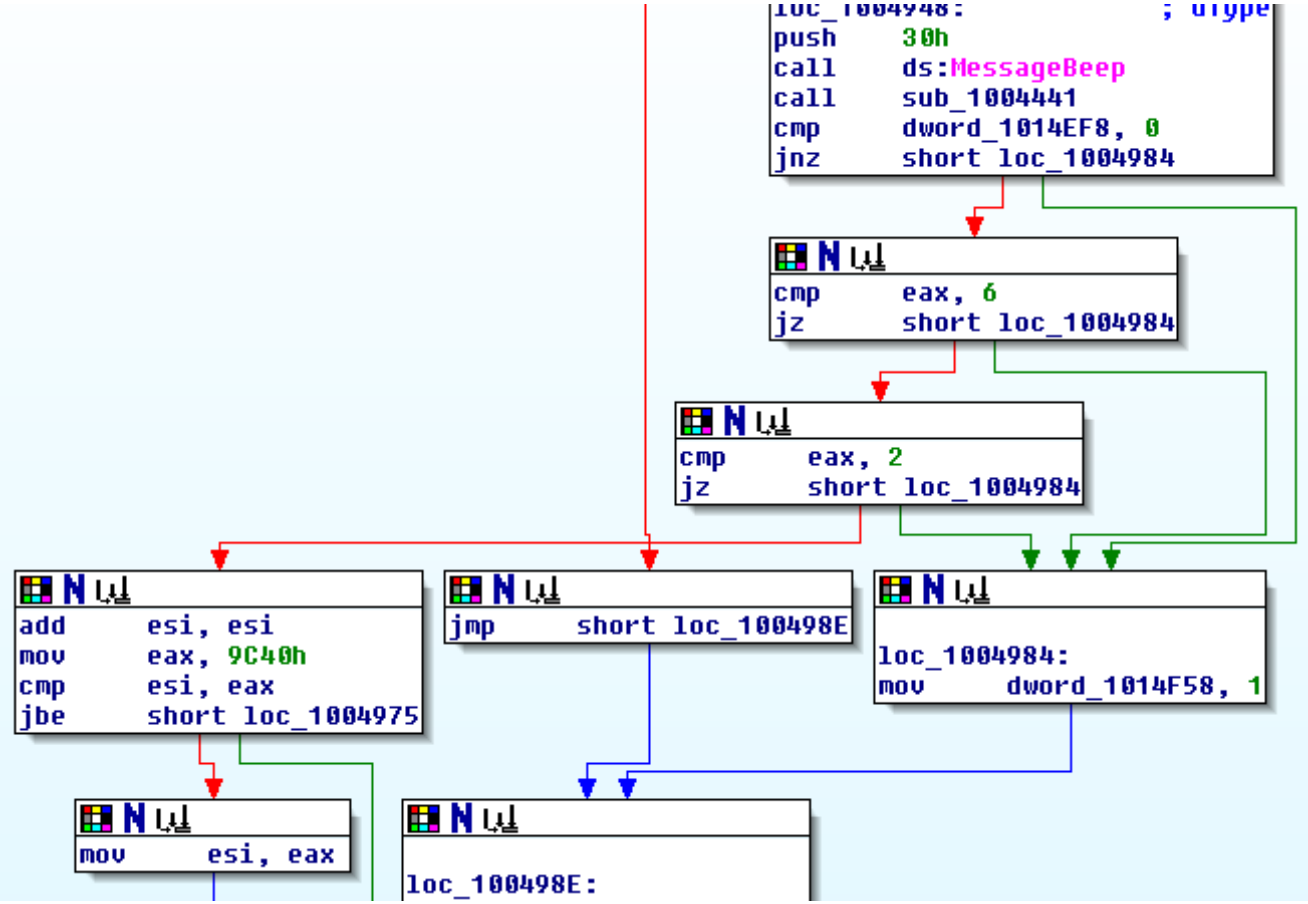
# 7. Function window

- List all functions of the binary

# IDA Graph View

# IDA Graph view

- Function is broken up into basic blocks so you can visualize the function's control flow from one block to another.



# IDA Graph view

- You'll notice IDA uses different colored arrows to distinguish various types of flows
- A conditional jump generate two possible flows
  - Yes arrow - branch is taken - green
  - No arrow – branch not taken - red
- Basic blocks that terminate with only one potential successor block utilize a Normal edge (blue by default) to point to the next block to be executed.
- We can zoom in and zoom out

# IDA Text View

# IDA Text view

- The text display presents the entire disassembly listing of a program

```
.text:004011B5
.text:004011B5 ; =============== S U B R O U T I N E =======================================
.text:004011B5
.text:004011B5 ; Attributes: bp-based frame
.text:004011B5
.text:004011B5 sub_4011B5      proc near               ; CODE XREF: _main+41↓p ❸
.text:004011B5
.text:004011B5 arg_0           = dword ptr  8
.text:004011B5 arg_4 ❷         = dword ptr  0Ch
.text:004011B5 arg_8           = dword ptr  10h
.text:004011B5
.text:004011B5                 push    ebp
.text:004011B6                 mov     ebp, esp
.text:004011B8                 mov     ecx, [ebp+arg_8]
.text:004011BB                 mov     edx, [ebp+arg_4]
.text:004011BE                 mov     eax, [ebp+arg_0]
.text:004011C1                 test    ecx, ecx
.text:004011C3                 jz      short loc_4011D1
.text:004011C5
.text:004011C5 loc_4011C5:                             ; CODE XREF: sub_4011B5+1A↓j ❸
.text:004011C5                 test    edx, edx
.text:004011C7                 jz      short loc_4011CC
.text:004011C9                 dec     eax
.text:004011CA                 jmp     short loc_4011CD
.text:004011CC ; ---------------------------------------------------------------------------
.text:004011CC
.text:004011CC loc_4011CC:                             ; CODE XREF: sub_4011B5+12↑j ❸
.text:004011CC                 inc     eax
.text:004011CD
.text:004011CD loc_4011CD:                             ; CODE XREF: sub_4011B5+15↑j ❸
.text:004011CD                 test    ecx, ecx
.text:004011CF                 jnz     short loc_4011C5
.text:004011D1
.text:004011D1 loc_4011D1:                             ; CODE XREF: sub_4011B5+E↑j ❸
.text:004011D1                 pop     ebp
.text:004011D2                 retn
.text:004011D2 sub_4011B5      endp
```

# 1. Arrows window

- Used to depict nonlinear flow within a function.
- Solid arrows represent unconditional jumps, while dashed arrows represent conditional jumps.
- When a jump (conditional or unconditional) transfers control to an earlier address in the program, a heavy weighted line (solid or dashed) is used.

# 2. Declarations

- IDA's best estimate concerning the layout of the function's stack frame.

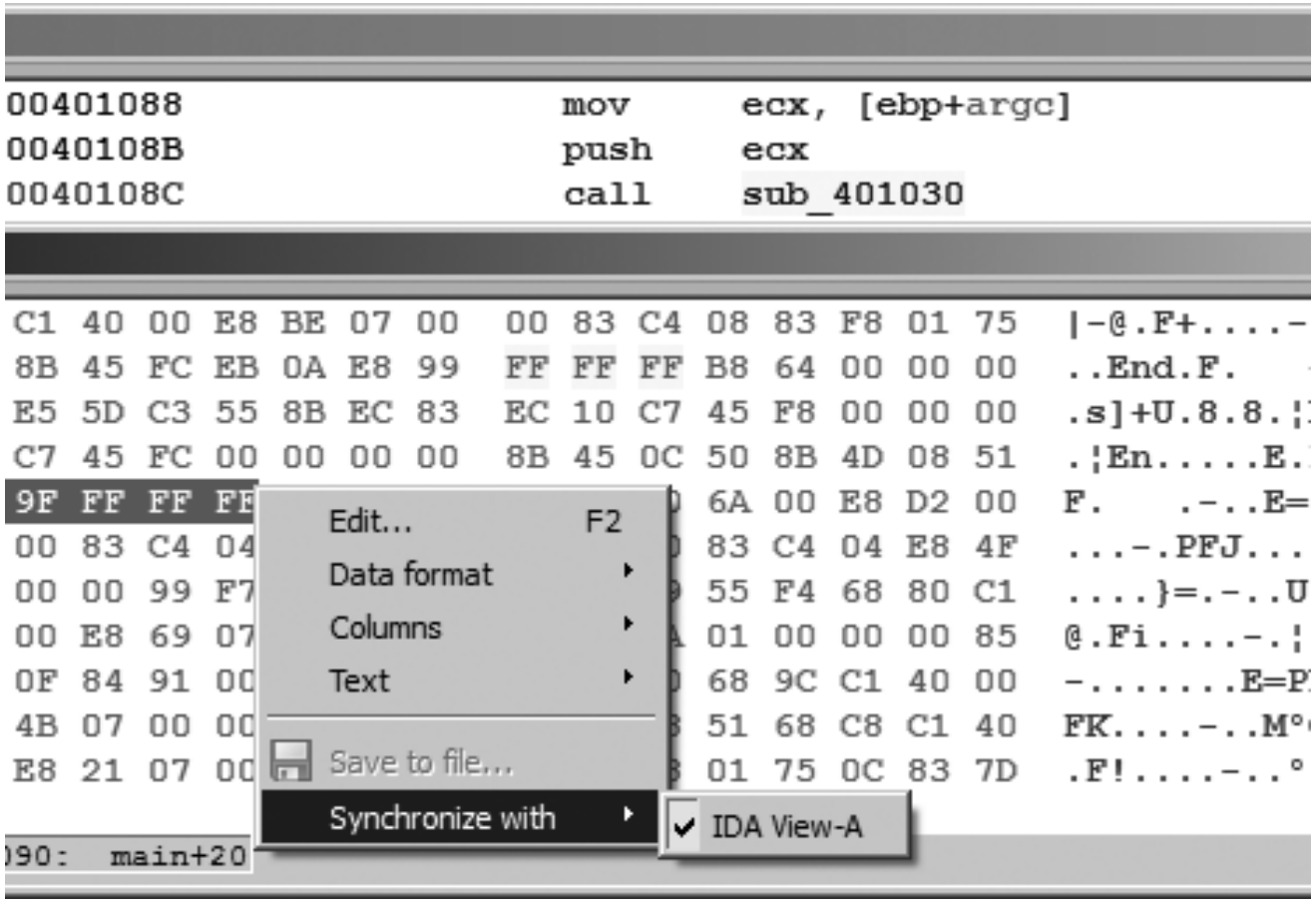
# 3. Comments

- In this case we see code cross-references
- Indicate that another program instruction transfers control to the location containing the cross-reference comment.

# Secondary IDA Displays

# The Hex View Window

- IDA Hex View window can be configured to display a variety of formats and doubles as a hex editor.
- The first Hex window is titled Hex View-A, the second Hex View-B, the next Hex View-C, and so on.
- If any, disassembly view you would like to synchronize a particular hex display.



# The Exports Window

- The Exports window lists the entry points into a file.
- Exported functions are commonly found in shared libraries such as Windows DLL files.

| | | |
|---|---|---|
| LoadLibraryA | 7C801D77 | 578 |

# The Imports Window

- It lists all functions that are imported by the binary being analyzed.
- The Imports window is relevant only when a binary makes use of shared libraries.

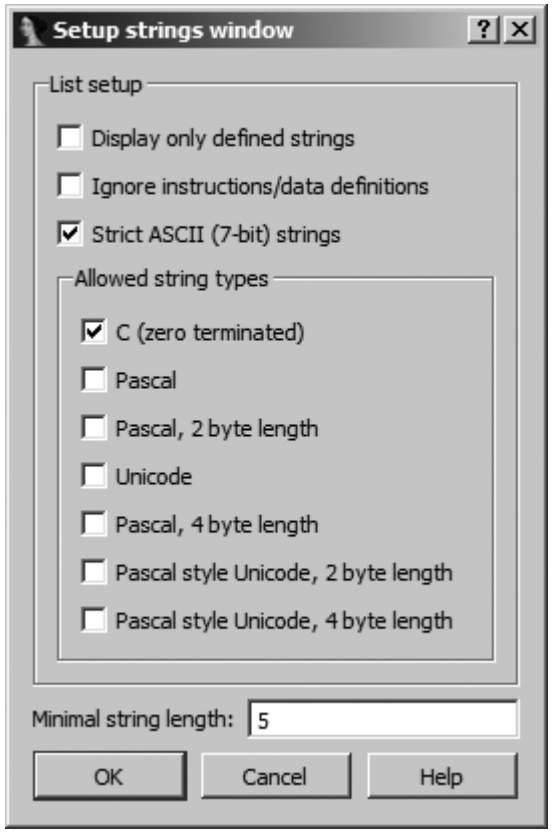
| | | |
|---|---|---|
| 0040E108 | GetModuleHandleA | KERNEL32 |

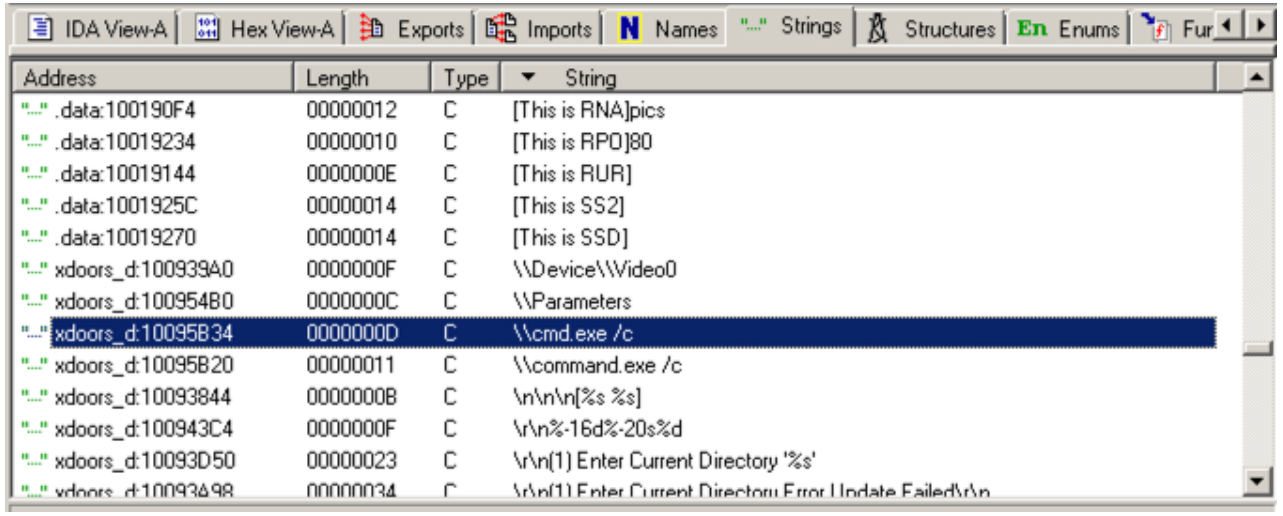
# Tertiary IDA Displays

# The Strings Window

- The Strings window is the built-in IDA equivalent of the strings utility
- Available via View > OpenSubviews > Strings.
- To display a list of strings extracted from a binary along with the address at which each string resides.
- Can use for cross-reference too!
- The default string type that IDA scans for is a C-style, null-terminated, 7-bit, ASCII string of at least five characters in length.

# The Strings Window



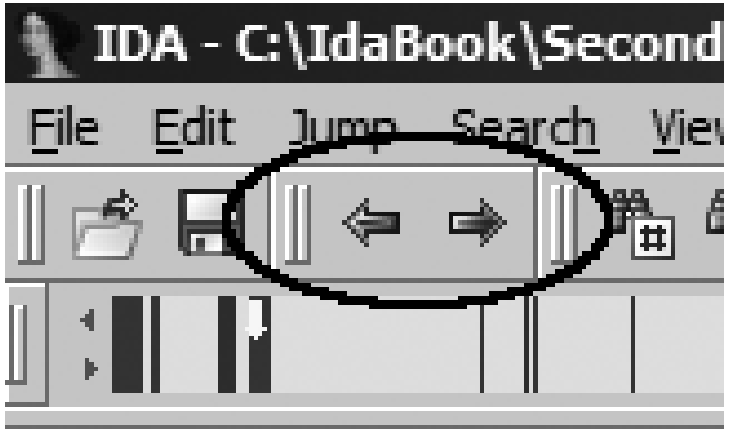



# Jump to Address

- Easiest way to get to a known disassembly location is to make use of the Jump to Address
- The Jump to Address dialog is accessed via Jump?Jump to Address
- Or by using the G hotkey while the disassembly window is active.

# Navigation History

- Another feature IDA shares with traditional web browsers is the concept of forward and backward navigation.
- Each time you navigate to a new location within a disassembly, your current location is appended to a history list.



# Cross Reference and Graphing

# Cross reference

- Common questions asked
  - “Where is this function called from?’
  - “What functions access this data?”
- IDA helps to answer these types of questions through its extensive cross-referencing features.
- Two category
  - code cross-references
  - data cross-references

# Code cross-reference

- A code cross-reference is used to indicate that an instruction transfers or maytransfer control to another instruction.

```
.text:00401000 callflow        proc near               ; CODE XREF: _main+20↓p
.text:00401000                                          ; _main:loc_401054↓p
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 pop     ebp
.text:00401004                 retn
.text:00401004 callflow        endp
```

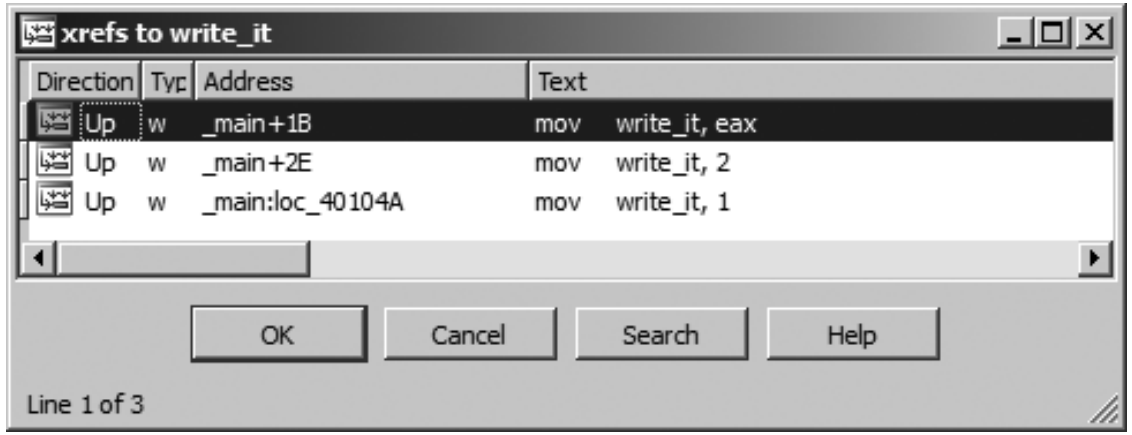
# Data cross-reference

- Data cross-references are used to track the manner in which data is accessed within a binary.

```
.data:0040B720 read_it       dd ?                    ; DATA XREF: _main+E↑r
.data:0040B720                                       ; _main+25↑r
.data:0040B724 write_it      dd ?                    ; DATA XREF: _main+1B↑w
.data:0040B724                                      ⑩; _main+2E↑w ...
.data:0040B728 ref_it        db    ? ;               ; DATA XREF: _main+4↑o
.data:0040B729               db    ? ;
.data:0040B72A               db    ? ;
.data:0040B72B               db    ? ;
```

# Cross-Reference Lists

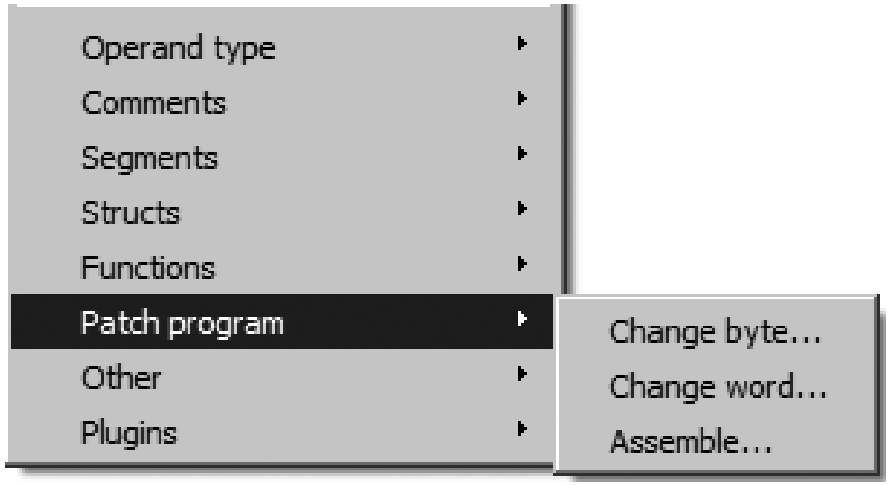
- Pressing hotkey ctrl+x can display us the complete list of cross-references to a given location.
- As with other windows that display lists of addresses, double-clicking any entry repositions the disassembly display to the corresponding source address.



# Patching Binaries

# The Infamous Patch Program Menu

- The Edit > Patch Program menu is a hidden feature in the GUI version of IDA that must be enabled by editing the idagui.cfg configuration file.
- The options is available on the Edit > Patch Program submenu.

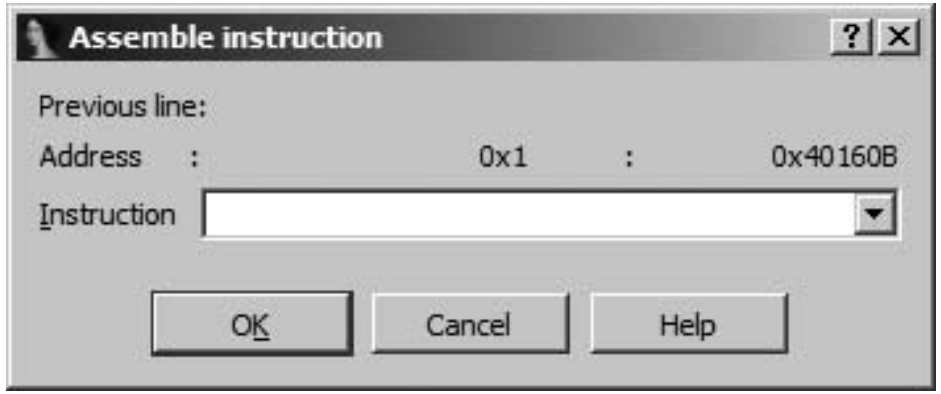


# Patching binaries

- Able to modify the binary in potentially interesting ways.
- Crack software.
- Modify malware behaviour
- Many more…

# Using the Assemble Dialog

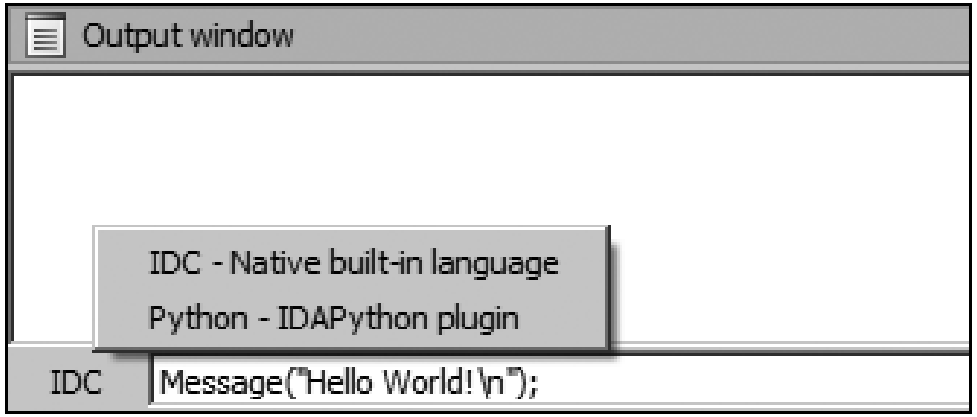
- Perhaps the most interesting capability accessible from the Patch Program menu is the Assemble option
- Edit > Patch Program > Assemble
- You can enter one instruction at a time into the Instruction field.



# Extending IDA capabilities

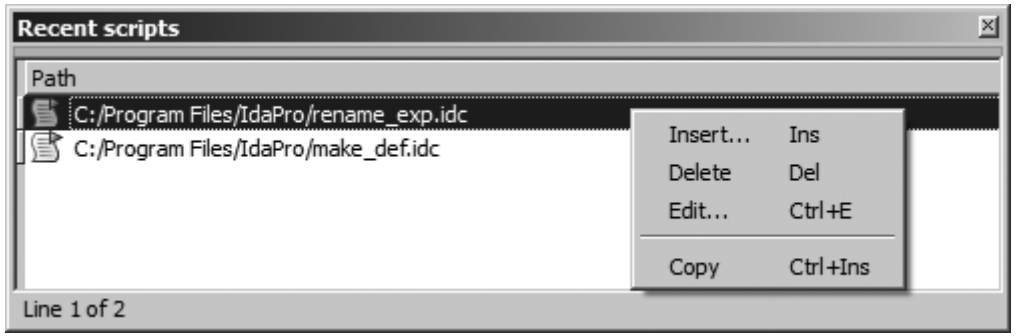
# IDA Scripting

- IDA takes the latter approach by integrating scripting features that allow users to exercise a tremendous amount of programmatic control over IDA's actions.
- IDA supports scripting using two different languages
  - IDC
  - IDAPython



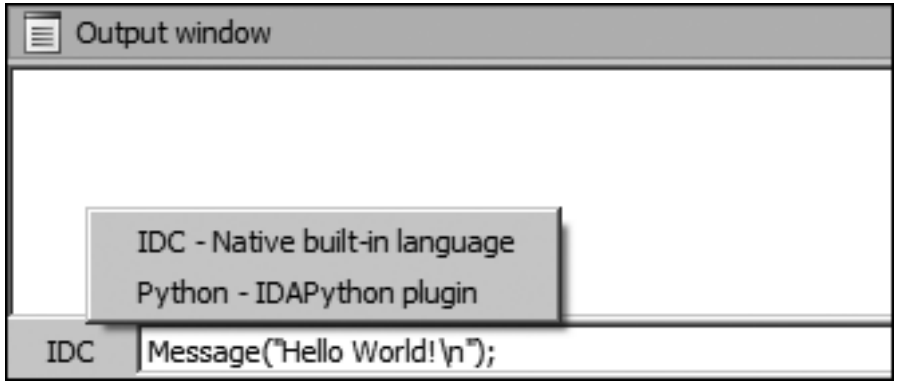
# IDA Scripting

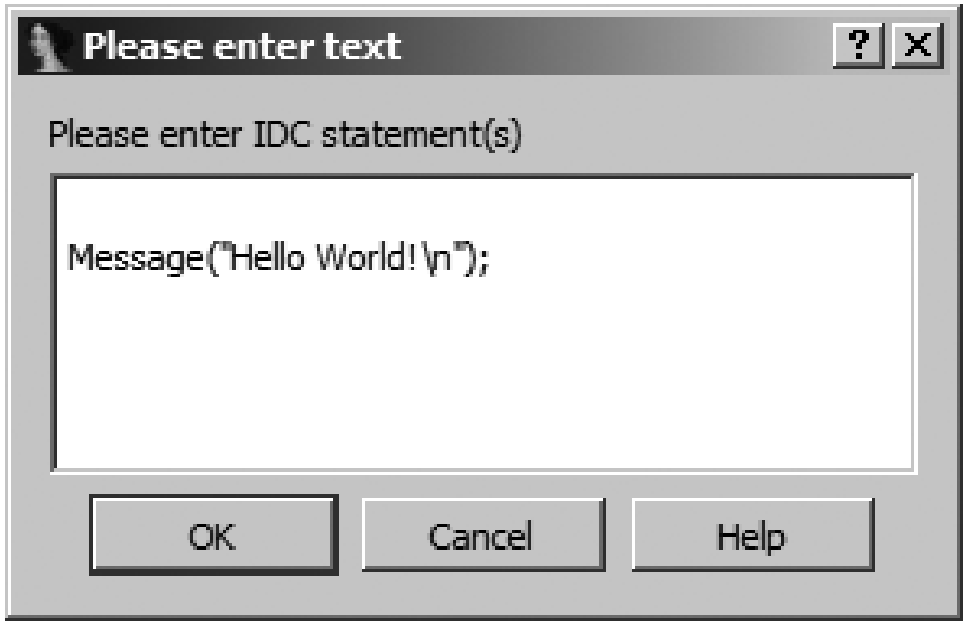
- Access via
  - File > Script File,
    - you wish to run a standalone script
  - File > IDC Command,
    - wish to execute only a few statements but don't want to go to the trouble of creating a standalone script file.
  - and File > Python Command.



# IDA Scripting

- The last way to easily execute script commands is to use IDA's commandline.
- The command line has been enabled by default since IDA 5.4.



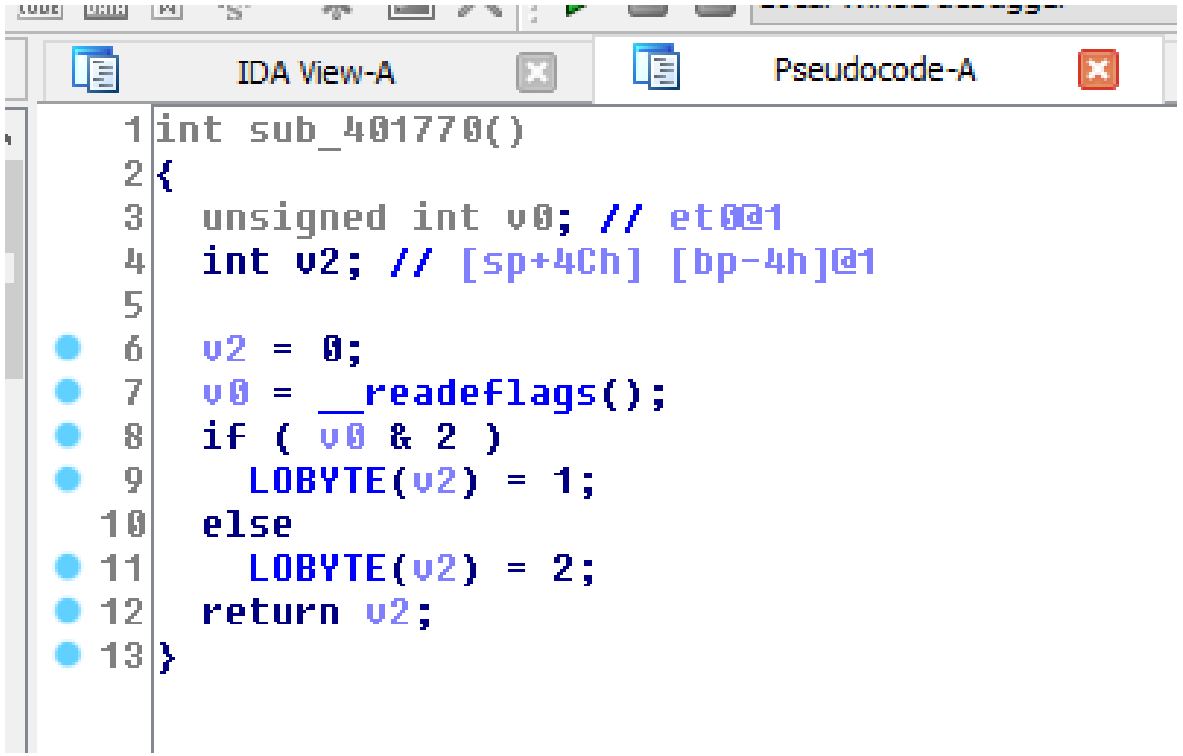
# IDA Decompiler

# Decompiler

- Decompile our assembly into a pseudocode that are readable!
- Available on IDA Pro
- Expensive but really good!
- The alternative is using Ghidra or Snowman

# Decompiler

- Click on a function, and press F5



# To Learn more about IDA

- Read this book
  - https://www.amazon.com/IDA-Pro-Book-Unofficial-Disassembler/dp/1593272898
- Watch Youtube on IDA tutorial
- Google is our friend