MALWARE ANALYSIS

IDA Pro (Interactive Disassembler Professional)

Dr. Zeyad Safaa Younus Saffawi

Dr. Zeyad Safaa Younus

IDA Pro

- **IDA Pro** is one of the most powerful and widely used **disassemblers** in cybersecurity. It is an essential tool for:
  - **malware analysts**
  - **reverse engineers**
  - **vulnerability researchers**.
- **How IDA Pro work**
  1. **Disassembles Programs**
     - Converts executable files (EXE, DLL) into **human-readable assembly code**.
  2. **Analyzes Program Structure**
     - Shows functions, loops, variables, strings, API calls, and logic flow.
     - Helps you understand the internal behavior of programs.
  3. **Helps When Source Code is Not Available**
     - Most malware does **not** include source code.
     - IDA Pro "reconstructs" how the program works using assembly.
     - Bridges the gap between **machine code and source code**.

Dr. Zeyad Safaa Younus

IDA versions

- There are **two commercial versions** of IDA Pro:
  - **Standard Version**
    - Supports **x86 (32-bit)** executables.
    - Limited architecture support.
  - **Advanced (Professional) Version**
    - Paid version with **full feature set**.
    - Supports many processors including **x64 (64-bit)**
    - Essential for modern malware analysis.
- **IDA Pro** uses a built-in signature recognition system called **FLIRT.**
    - **FLIRT** stands for **Fast Library Identification and Recognition Technology**.

Dr. Zeyad Safaa Younus

Fast Library Identification and Recognition Technology (FLIRT)

- **What does FLIRT do?**
  - Automatically identifies **common library functions** inside a program.
  - Detects functions added by compilers.
  - Helps analysts skip unimportant code (like C runtime functions).
  - Saves time by labeling known functions instead of reversing them manually.

- **Why is FLIRT important?**
  - Without FLIRT, analysts would spend hours analyzing compiler-generated code that is not related to malware behavior.

Dr. Zeyad Safaa Younus

Disassembly Window Modes

- The disassembly window can be viewed in **two modes**:
  - **Graph Mode** (Default View)
  - **Text Mode**

- Switching Between Modes at any time by pressing the **Spacebar**.
  - This instantly toggles between **Graph Mode** and **Text Mode**.

Dr. Zeyad Safaa Younus

Graph Mode

- IDA Pro shows the program as a **Control Flow Graph**.
- Each **basic block** (a sequence of instructions with no branches) is shown as a **node (box)**.
- **Arrows** connect these blocks to show how the program flows.
- **Displaying More Information in Graph Mode**
  - By default, IDA hides some helpful details like opcode bytes.
  - To enable them:
    - Go to Options.
    - Enable **Line prefixes**.
    - Set **Number of Opcode Bytes** to 6.
  - This shows memory addresses and raw opcode bytes for each instruction.
- IDA Pro uses colored arrows to show program flow:
  - **Red** – Conditional jump **not taken**
  - **Green** – Conditional jump **taken**
  - **Blue** – Unconditional jump
  - ⬆️ **Upward arrows** → Usually indicate **loops** (jumping back)
- **Highlighting in Graph Mode**
  - If you highlight any text (variable name, offset, string), **all matching references** are highlighted across the graph.
  - Helps track variables, function names, or constants inside across the code

Dr. Zeyad Safaa Younus

Text Mode

- Provides a **traditional linear view** of assembly code.
- Similar to what you see in **debuggers** or classic disassemblers.
- Required when analyzing **data regions**, not just code.
  - Example display:

    0040105B .text 83EC18

    - 0040105B → Memory address
    - .text → Section name containing this instruction
    - 83EC18 → Opcode bytes

    This mode shows the exact layout of the binary as it appears in memory.

- The left section, called the **arrows window**, represents **nonlinear control flow**:
  - **Solid lines** – Unconditional jumps
  - **Dashed lines** – Conditional jumps
  - **Upward arrows** – Loops

  This helps you trace how the program branches and repeats.

- IDA also displays:
  - The **stack layout** of the function (local variables and arguments)
  - **Auto-generated comments** (starting with ;) to help understand the code

Text Mode

Arrows
Solid = Unconditional
Dashed = Conditional
Up = Loop

Section

Address

Comment Generated by IDA Pro

```
.text:00401015                 jz      short loc_40102B
.text:00401017                 push    offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C                 call    sub_40105F
.text:00401021                 add     esp, 4
.text:00401024                 mov     eax, 1
.text:00401029                 jmp     short loc_40103A
.text:0040102B ; ---------------------------------------------------------------------------
.text:0040102B
.text:0040102B loc_40102B:                             ; CODE XREF: sub_401000+15↑j
.text:0040102B                 push    offset aError1_1NoInte ; "Error 1.1: No Internet\n"
.text:00401030                 call    sub_40105F
.text:00401035                 add     esp, 4
.text:00401038                 xor     eax, eax
.text:0040103A
.text:0040103A loc_40103A:                             ; CODE XREF: sub_401000+29↑j
.text:0040103A                 mov     esp, ebp
.text:0040103C                 pop     ebp
```

Dr. Zeyad Safaa Younus

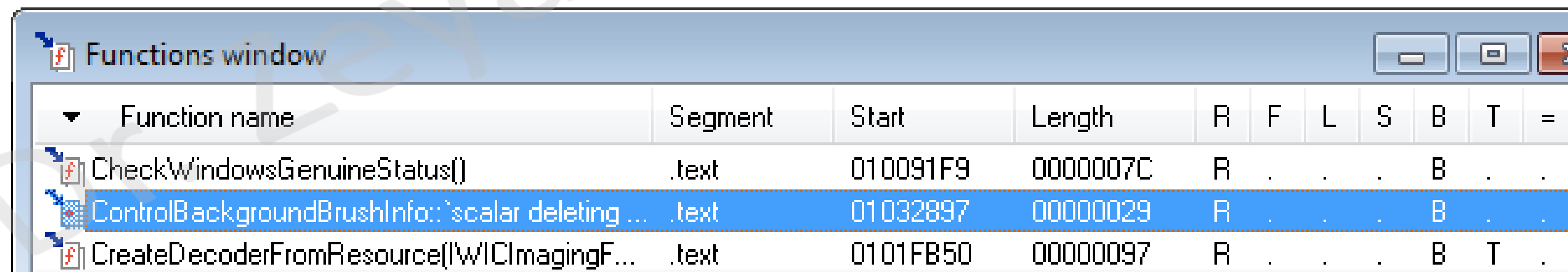
Useful Windows for Analysis in IDA Pro

- In addition to the main disassembly window, IDA Pro provides several **specialized windows** that help analysts focus on important parts of the executable code. These windows make it easier to:
  - **Navigate large executables**
  - **Identify important elements** (functions, strings, imports, or data)
  - **find suspicious areas quickly during malware analysis**

Dr. Zeyad Safaa Younus

Functions Window

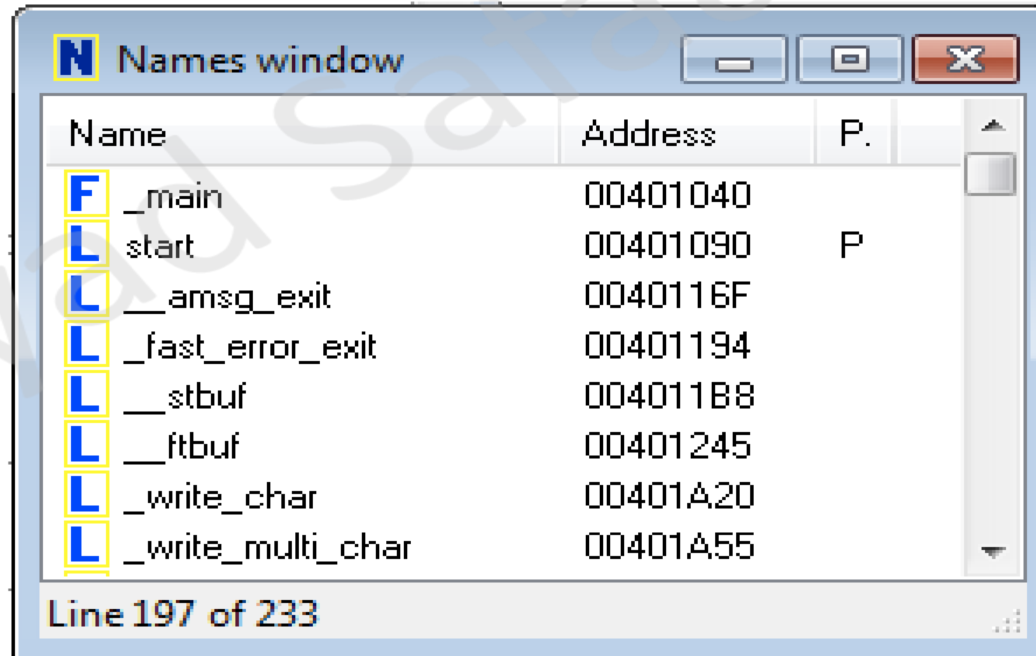
- Displays **all functions** found in the executable, along with the **length of each function**.
- You can **sort by function length**, making it easy to:
  - Identify **large and complex functions** that may contain malicious logic.
  - Ignore or skip **small, trivial functions** that are usually unimportant.
- The window also uses **flags** to classify functions:
  - L → Library function (compiler-generated, can usually be skipped)
  - F, S, etc. → Other classifications
  - The L flag is particularly helpful because skipping library functions saves analysis time and lets you focus on custom or suspicious code.



Dr. Zeyad Safaa Younus

Names Window

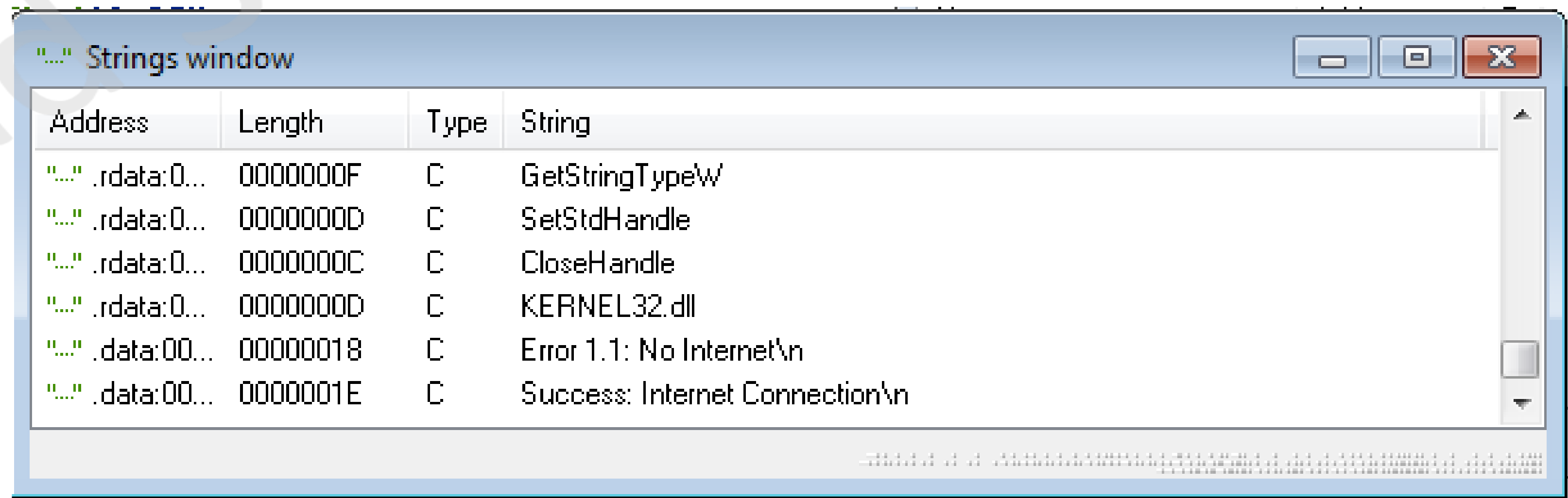
- Lists **every named address** in the binary:
  - Functions
  - Named code blocks
  - Named data
  - Strings
- Useful for quickly locating specific functions or variables by name.



Dr. Zeyad Safaa Younus

Strings Window

- Displays all **strings** found in the binary.
- By default:
  - Shows **only ASCII strings** longer than **5 characters**.
- You can **customize** the display:
  - Right-click in the window and select **“Setup”** to adjust string length or encoding (e.g., Unicode).
- **Why it matters:**
  - Strings often reveal valuable information such as:
    - IP addresses
    - Domain names
    - File paths
    - Commands
    - Debug messages



Dr. Zeyad Safaa Younus

Imports and Exports Window

- **Imports Window**
  - Lists all **imported functions** used by the executable.
  - Useful for identifying external APIs and system functions the program relies on.
  - Malware often uses specific API calls (e.g., **CreateProcess, InternetOpen**) to perform malicious actions.

- **Exports Window**
  - Lists all **exported functions** in the file.
  - This window is especially valuable when analyzing **DLL files**:
    - Exported functions can show the interface provided by or connected with malware.
    - It can detect the **entry points** that other processes might call.

Imports

Address	Ordinal	Name	Library
004060A0		GetStringTypeA	KERNEL32
004060A4		GetStringTypeW	KERNEL32
004060A8		SetStdHandle	KERNEL32
004060B0		InternetGetConnectedState	WININET

Line 1 of 44

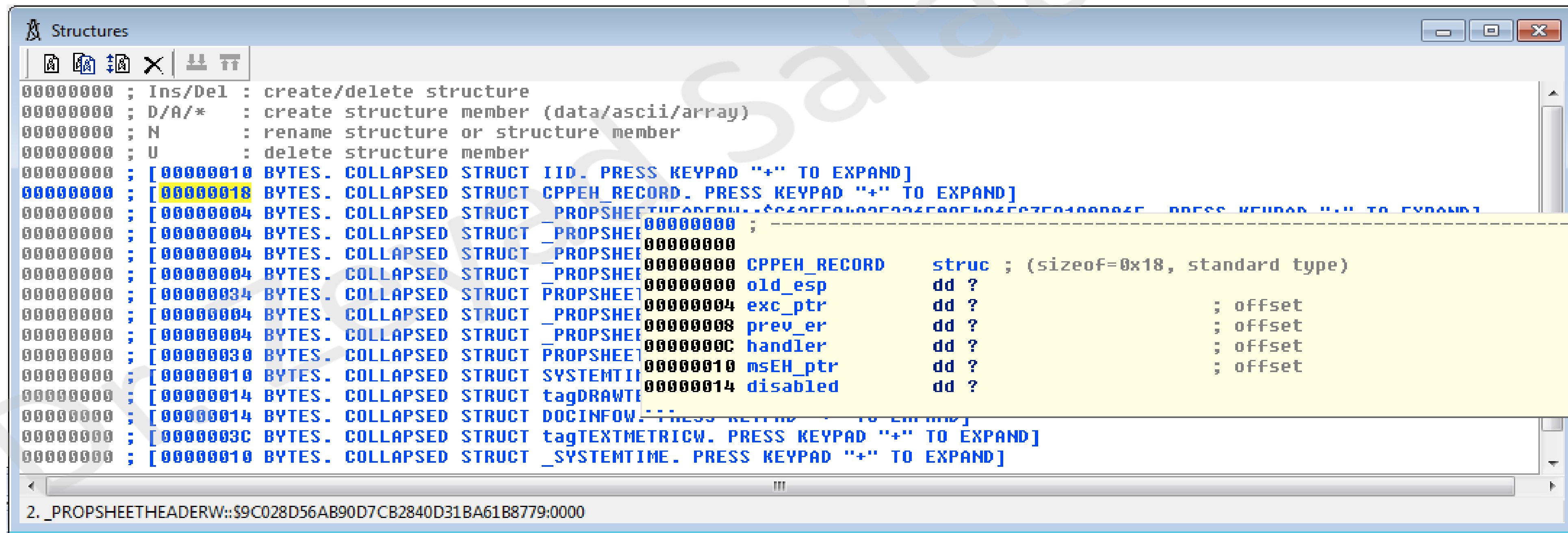
Exports

Name	Address	Ordinal
start	00401090	

Dr. Zeyad Safaa Younus

Structures Window

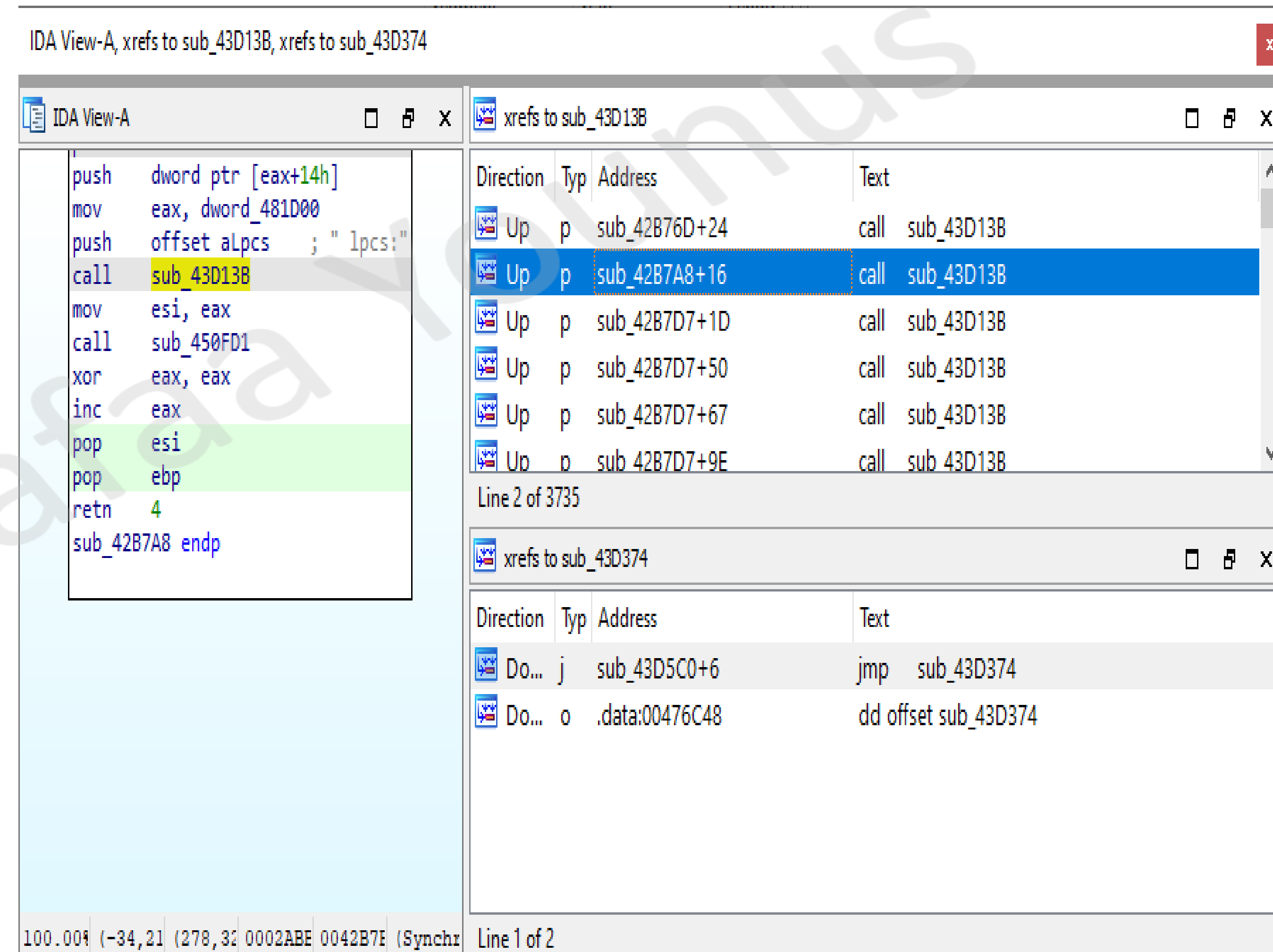
- Displays the **layout of data structures** recognized by IDA Pro.
- You can also **create your own custom structures** to:
  - Represent memory layouts
  - Interpret raw data regions more meaningfully
  - Simplify the tracking of pointer and variable during analysis



Dr. Zeyad Safaa Younus

Cross-Reference Feature (XREF)

- One of the **most powerful features** of these windows is their **cross-reference capability**, which allows you to quickly locate **where a function, variable, or string is used in the code**.
- If you want to find **where an imported function is called**,
  - Open the **Imports Window**,
  - Double-click the function name,
  - Then use the **cross-reference view** to navigate to the **exact location in the disassembly**.
- This makes it much faster to trace program behavior and identify suspicious code paths.



Dr. Zeyad Safaa Younus

Returning to the Default View in IDA Pro

- **Reset the Interface**
  - Go to the top menu bar and select:
    `Windows → Reset Desktop`
  - This returns the interface to the original default view.

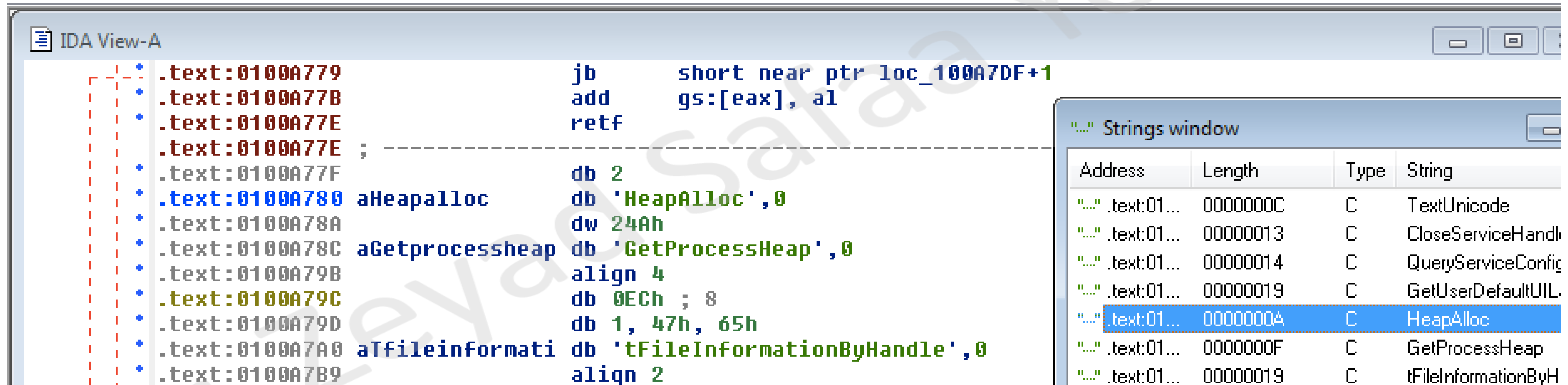
Saving Your Own Layout

- If you've **customized the interface** (for example, rearranged the windows in a way that works best for your workflow), you don't have to lose your setup.
- To save your customized view:
  - Go to **`Windows → Save Desktop`**.
  - IDA will remember your current window arrangement.

Dr. Zeyad Safaa Younus

Navigating IDA Pro — Imports or Strings

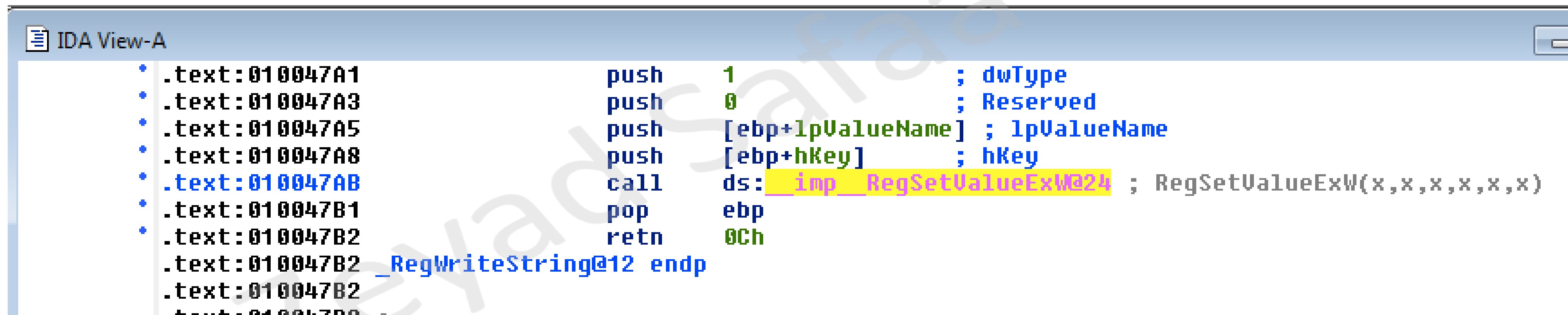
- Many windows and text in IDA are **clickable links**.
  - Double-clicking an item in **Imports**, **Strings**, or **Names** jumps you to that address in the disassembly.



Dr. Zeyad Safaa Younus

Navigating IDA Pro — Using Links

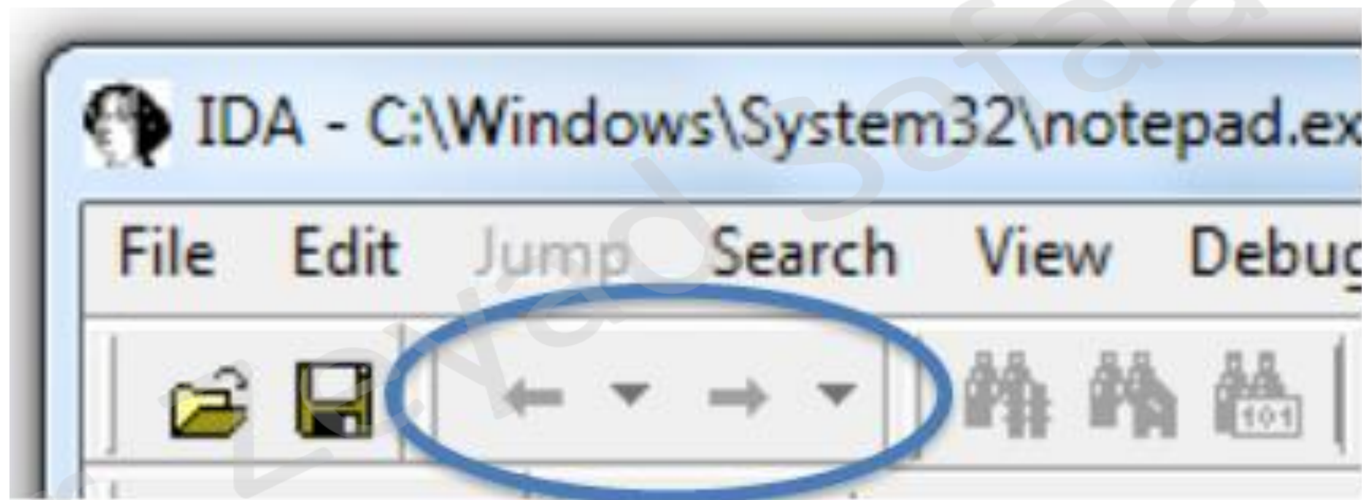
- Double-click any address in the disassembly window to display that location



Dr. Zeyad Safaa Younus

Navigating IDA Pro — History

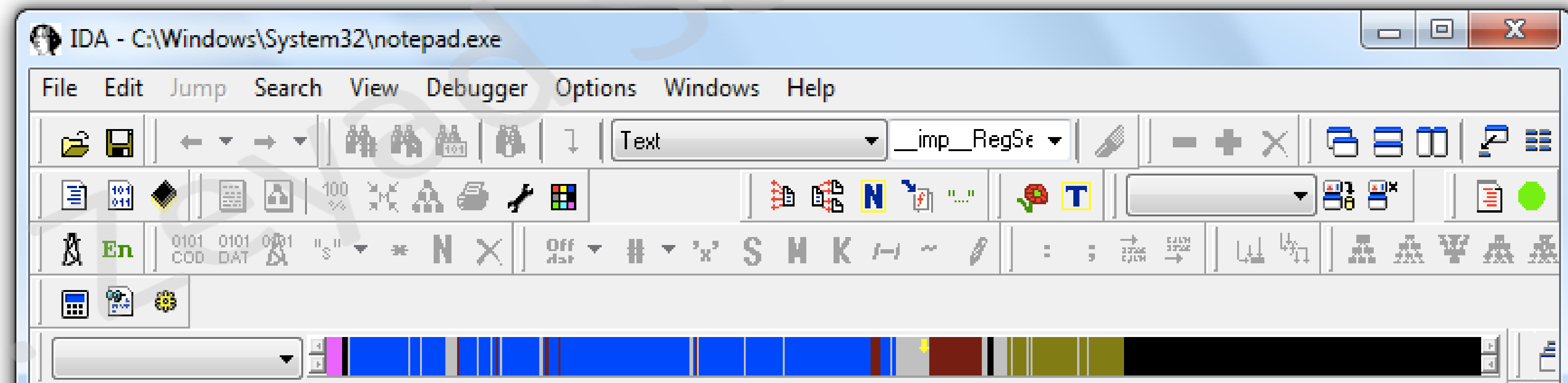
- Forward and Backward buttons work like a Web browser



Dr. Zeyad Safaa Younus

Navigating IDA Pro — Navigation Band

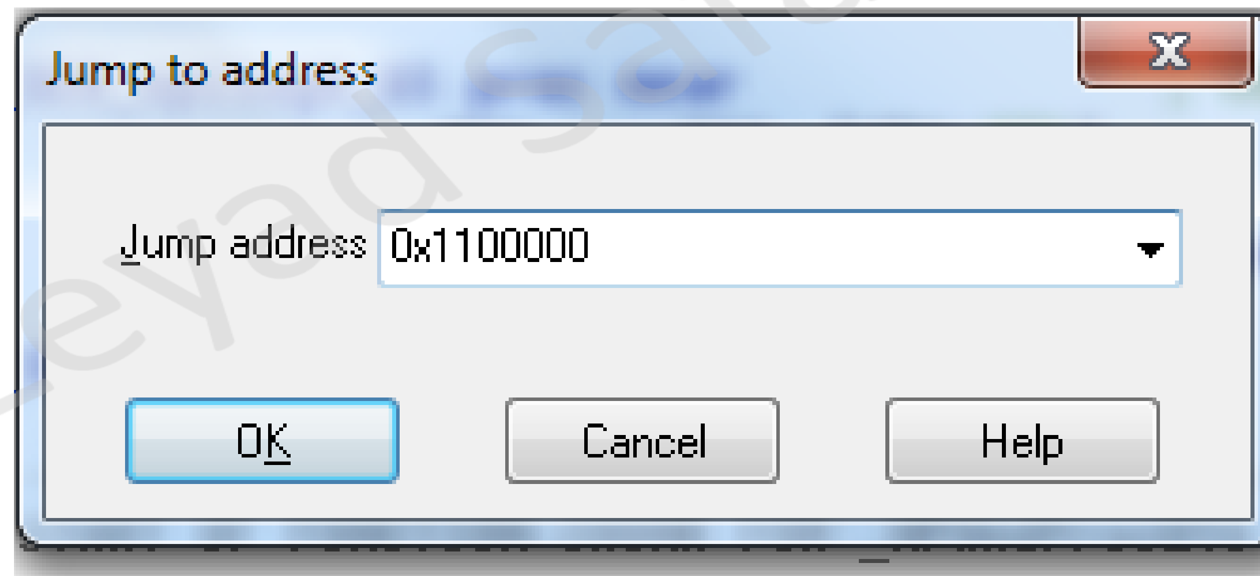
- A horizontal color bar at the top/bottom shows a **linear map** of the binary's address space.
  - **Light blue** = FLIRT-identified library code.
  - **Red** = compiler-generated code.
  - **Dark blue** = user-written code (your best area to analyze).



Dr. Zeyad Safaa Younus

Navigating IDA Pro — Jumping to addresses or symbols

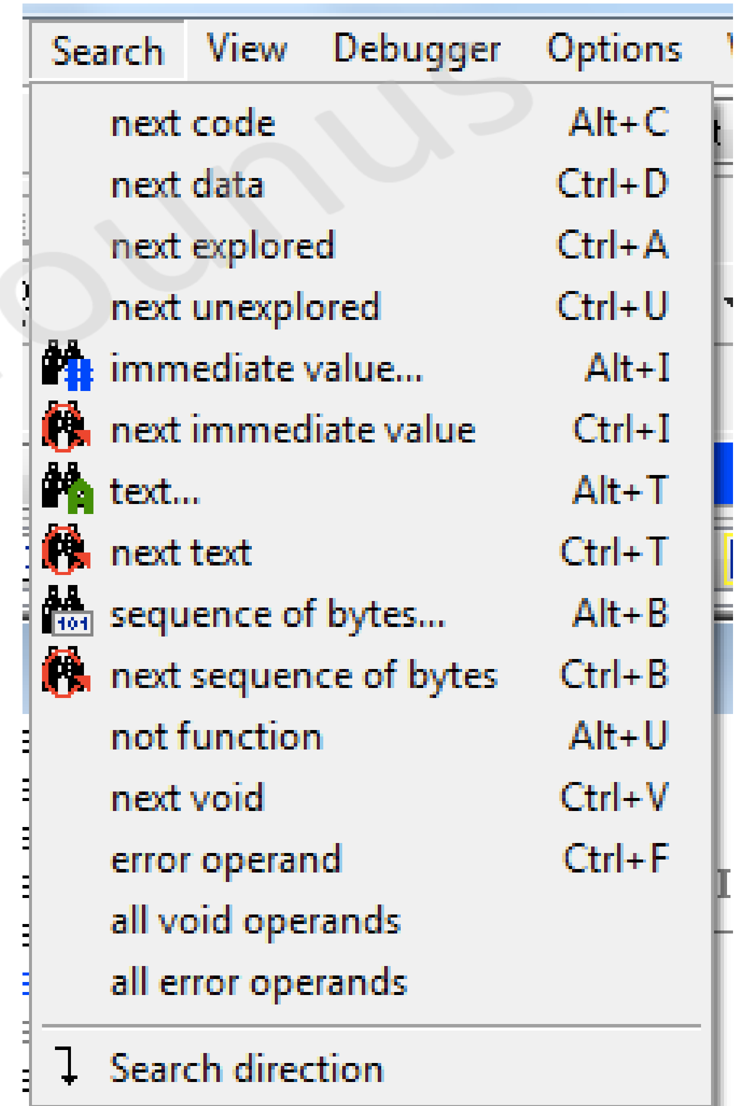
- Press **G** in the disassembly window to **go to**:
  - a virtual address (e.g., 0x401000) or
  - a named symbol (sub_401730 or printf).
- To jump to a **file offset** (hex editor offset), use Jump → Jump to File Offset.



Dr. Zeyad Safaa Younus

Navigating IDA Pro — Searching

- Search menu offers:
  - **Next Code** — find next occurrence of an instruction pattern.
  - **Text** — search for strings or identifiers in the disassembly.
  - **Sequence of Bytes** — search for raw byte patterns in the hex view (useful for shellcode/opcodes).
- Use search to locate strings (like “Bad key”) and then jump to their code references.



Dr. Zeyad Safaa Younus

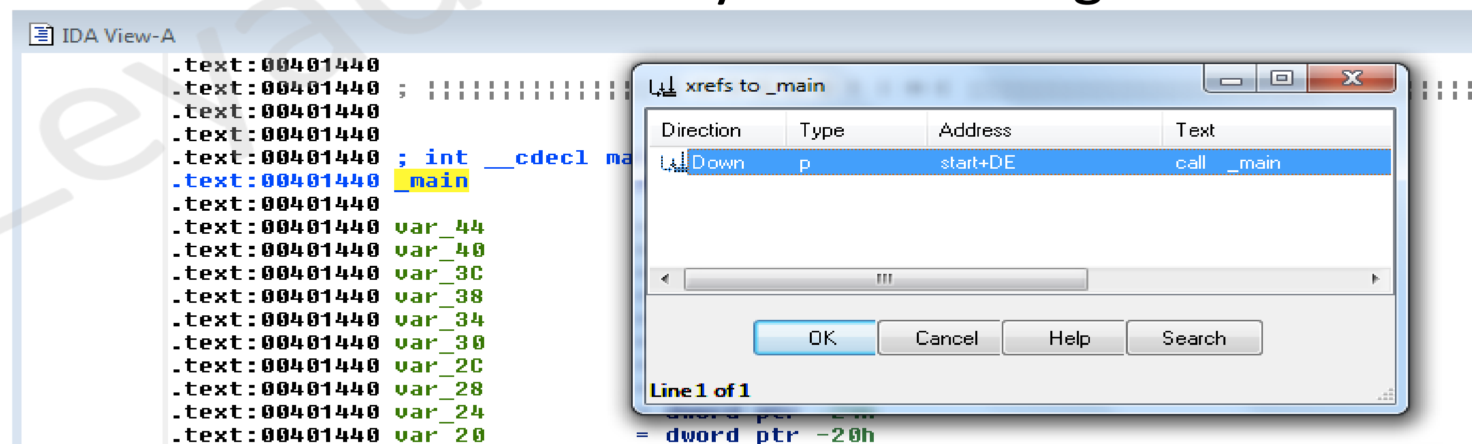
Cross-References

- **Cross-References (Xrefs)** are one of the most powerful features for analyzing how functions and data are used inside a binary.
- A **cross-reference** tells you:
  - From **Where the function is called**.
  - **Where a string or variable is used** in the code.

Dr. Zeyad Safaa Younus

Code Cross-References

- Code XREFS show where a **function is called** or where a **jump instruction** leads.
- This means you can use XREFS to **trace exactly where a function is invoked**.
- To view all XREFS for a function:
  - Click on the **function name**.
  - Press **X** on your keyboard.
  - A window will appear listing all the locations where the function is referenced.
- This is very useful if a function is called many times throughout the binary.



Dr. Zeyad Safaa Younus

Data Cross-References

- Data XREFS track how **data is accessed** in the binary (such as IP addresses or strings).
- Identify where sensitive or suspicious data is used.
- Track IP addresses, credentials, or text strings throughout the binary.

Dr. Zeyad Safaa Younus

Function Recognition in IDA Pro

- IDA Pro can automatically identify **functions** in a binary and provide a clear structure for analyzing them, including:
  - **Name the Function**
  - **Name the local variables**
- This makes it easier to understand what a function does without reading raw assembly instructions line by line.

Dr. Zeyad Safaa Younus

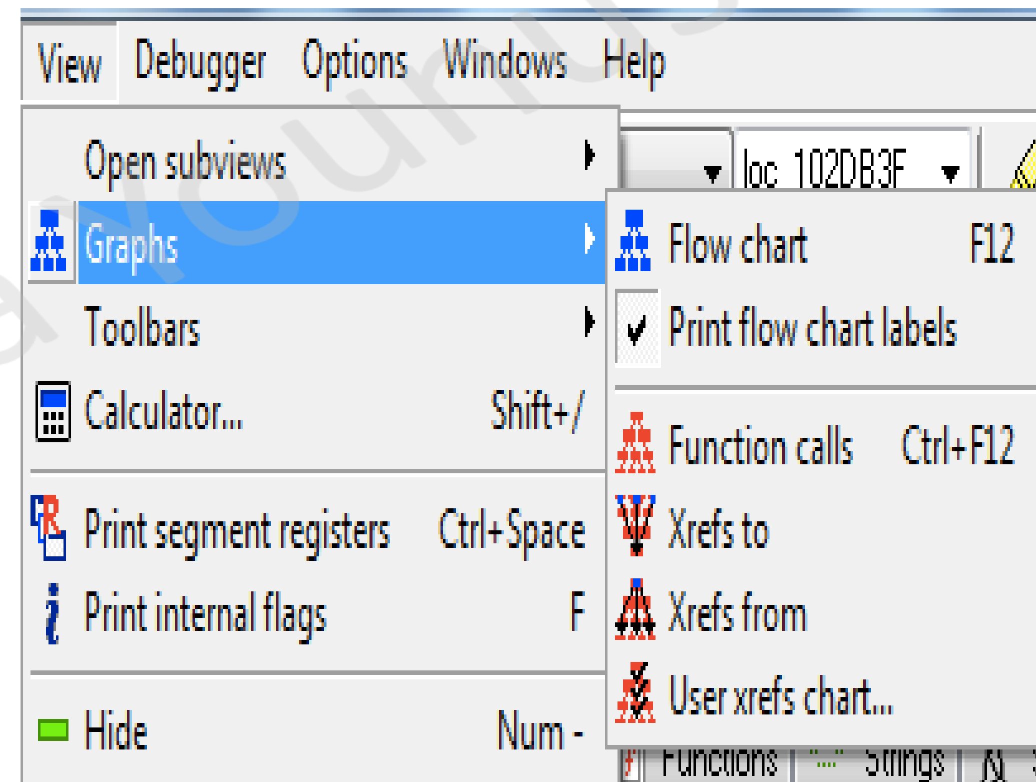
Using Graphing Options

- IDA Pro provides several **graphing options** that help visualize program flow and function relationships.
- These graphs are displayed using **WinGraph32**, which is a legacy application.
  - Unlike the interactive graph in the disassembly window, these graphs **cannot be modified directly** in IDA Pro.
- **Toolbar buttons** provide access to five different graphing options.

Dr. Zeyad Safaa Younus

Using Graphing Options

- These are "Legacy Graphs" and cannot be manipulated with IDA
- **Flow chart of current function**
  - build flow chart of the current function for quick visualization.
- **Function calls for entire program**
  - Displays the hierarchy of all function calls in the program.
- **Cross-references Xrefs to get to the selected Xref**
  - Shows all the paths that get to a function or variable.
- **Cross-references Xrefs from the selected Xref**
  - Shows all the paths that exit from a function or variable.
- **User Xrefs Chart**
  - Allows creating a custom graph with depth, symbols, and relationships.



Dr. Zeyad Safaa Younus

Enhancing Disassembly

- **Warning: IDA** does NOT have an **undo feature**.
- **Rename locations (**Give Meaningful Names**)**
  - Replace auto names like sub_401000 with descriptive names (e.g., ReverseBackdoorThread, DNSrequest).
  - Rename once; IDA **propagates** the new name everywhere it's referenced, resulting in **huge time saver**.
- **Add comments**
  - **Local comment:** place cursor on a line and press colon (**:**) to add a single comment .
  - **Repeatable comment:** press semicolon (**;**) shown wherever the address is referenced (useful for explaining a function or data item).
- **Format operands & change Number display**
  - Hexadecimal by default
  - Right-click an operand (e.g., 0x62) to change display: decimal, octal, binary, or ASCII.

```
mov     edi, edi
push    ebp
mov     ebp, esp
mov     eax, 1320h
call    __chkstk
mov     eax, ___se
xor     eax, ebp
mov     [ebp+var_
push    offset aSe
```

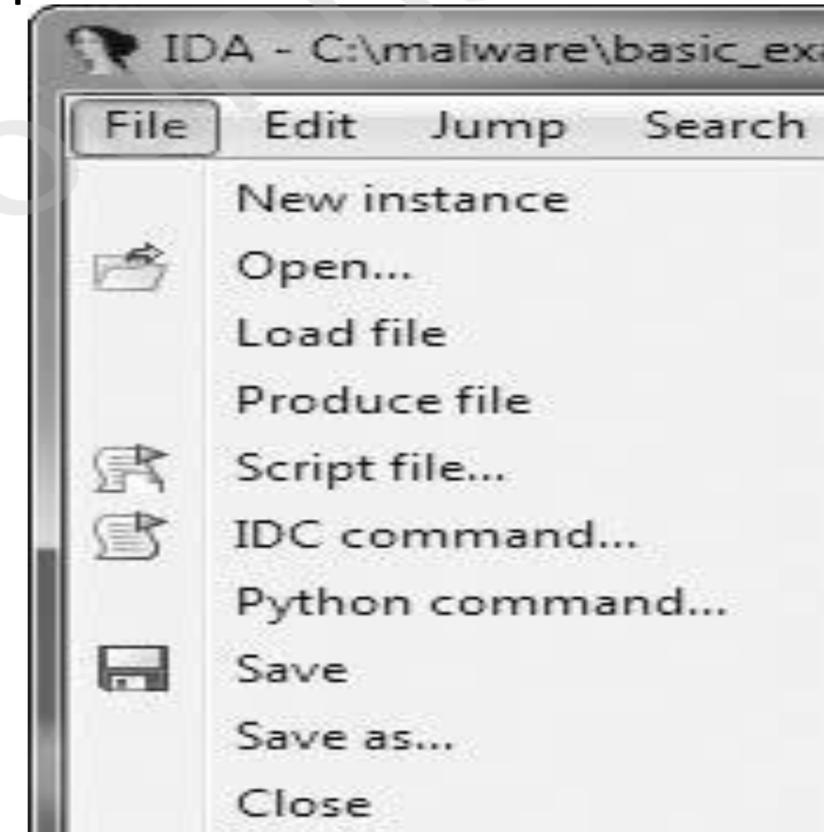
Use standard symbolic constant
4896 H
11440o
1001100100000b B

- **Using Standard Named Constants**
  - Convert raw numeric flags to Windows constants to Makes Windows API arguments clearer
- **Redefine code vs data**
- IDA sometimes mislabels bytes. Fix it:
  - U = undefine (turn code/data into raw bytes)
  - C = define bytes as **code** (disassemble)
  - D = define as **data**
  - A = define as **ASCII string**

Dr. Zeyad Safaa Younus

Extending IDA Pro with Plug-ins and Scripts

- **Using Scripts**
  - IDA Pro can be extended using **IDC** and **IDA Python** scripts to speed up analysis and add new features.
- **IDC Scripts**
  - Built-in scripting language, older than Python.
  - Uses functions with static declarations; local variable.
  - Example use: **set bookmarks, add comments, or markup disassembly**.
  - Run via **File → Script File** or **File → IDC Command**.
- **IDA Python Scripts**
  - Fully integrated Python scripting for powerful automation.
  - Operates primarily using **Effective Address (EA)** or symbol names.
  - Example: **color-code** all call instructions for easier analysis.



Dr. Zeyad Safaa Younus

Extending IDA Pro with Plug-ins and Scripts

- **Using Commercial Plug-ins**
  - **Hex-Rays Decompiler**:
    - Converts disassembly into C-like pseudocode
    - faster understanding of malware.
  - **zynamics BinDiff**:
    - Compares two IDA Pro databases to find new functions or code changes
    - useful for analyzing malware variants

Dr. Zeyad Safaa Younus

References

- Sikorski, M., & Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press.
- Bowne, S. (n.d.). *CNIT 126 5: IDA Pro* [PowerPoint slides]. SlideShare. https://www.slideshare.net/slideshow/cnit-126-5-ida-pro/71821631
- Bowne, S. (n.d.). *CNIT 126: Ch 5 IDA Pro* [Video]. YouTube. https://www.youtube.com/watch?v=h1xOSKrUXt8

Dr. Zeyad Safaa Younus