IDA Pro:

IDA stands for Interactive Disassembly. This chapter deals with how to utilize the product to understand a sample’s functionality and how it effects a computer or a network to develop host/network signatures or even to decide if a sample is indeed malicious or not.

FLIRT- Fast Library Identification and Recognition Technology- a pun.

What is pe.ldw for for IDA free?

We are trying to learn what we can from Lab 5.dll. we will go through each section and try and understand what is up with the sample. Ok?

sample 5 is helping me to understand IDA pro- there are lots of things to learn and this could be a career in a way.

Disassembly and using IDA pro is a long topic which needs a lot of Practice. We are going to do it every day until ends come.

Finally got the files unzipped- the idea is to copy the files some where else. Also- the folder was read only as it was marked in the VM box settings. You cannot create directories in read only scenarios.

Best resource for learning IDA pro is The IDA Pro Book: The Unofficial Guide to the World’s Most Popular Disassembler, 2nd Edition.

Using IDA pro:

As discussed in the previous chapter, most samples that are available to malware analysts are executables or code, which are already compiled. Though preliminary static and dynamic analysis tools help us to derive useful conclusions as to how the suspicious sample works, we need advanced tools to dig a little deeper and make accurate assumptions to come to a definite conclusion. Advanced analysis tools help us to do the following;

1. Look at the source code of the sample and make accurate conclusions as to how the malware works in a computer system and how the functions written are implemented. IDA pro can be labeled as an advanced static analysis tool.
2. Advanced dynamic tools such as olly debug help us to see a step by step progression of the execution of the sample in an environment, which helps us create a signature of the sample.

IDA Pro Basics:

1. When we first load a file, ida pro will try to recognize a file type and processer architecture.
2. When we try to disassemble a sample using IDA pro, ida pro tries to simulate how an OS handles this program by loading it similarly into the memory. We can also see a file as a binary file by selecting the option in the initial dialog box. This is helpful with certain samples which try to append additional shell code/parameters/data to certain PE files. Since the extra appendages won’t be loaded into memory while the file is being executed by either OS or while being disassembled by ida pro, opening some samples as binary files will help us to look at this code. Also, to disassemble a raw binary file which might contain shellcode, this option is preferable.
3. Manual Load option: All PE files are compiled to be loaded into memory at a specific base address (Virtual). If the base address preferred by a PE file is being used by another process, the OS rebases [It’s a shame that you don’t know about rebasing because you should have read about it last year in operating systems class, idiot]. To enter an address at which you want the PE file to be loaded at, click on the manual load option.
4. By default, IDA pro does not disassemble PE file header or resource sections and you can disassemble and analyze those sections by selecting the manual load option, which prompts ida pro to ask the user which sections need to be included in the disassembly.
5. Graph Mode: When you are in the function section, you can switch from text mode to graph mode by pressing space bar. Graph mode is especially useful to understand the flow of the sample code and in the sample code, how the program moves from one block to another block. The arrows suggest a decision-making process before the jump, and color of the arrows suggest the following;
6. Red if a conditional jump is not taken.
7. Green if a jump is taken.
8. Blue is for an unconditional jump.
9. Upward arrows suggest a looping situation.

Text Mode:

1. The left most section contains the following information;
2. The extreme left side is called the arrow window and indicates the non-linear flow of the program. The arrows indicate jumps taken by the sample with the following caveats;
3. Solid lines indicate unconditional jumps.
4. Arrows facing up indicate loops.
5. Non-solid lines indicate conditional jumps.
6. section of the file being analyzed (.text/rsrc/header etc). Unless selected while being loaded (see point 4 above), only the text portion the file will be displayed.
7. Memory location and opcodes and opcodes are also translated to machine code instructions.
8. By switching on the auto commends, ida pro helps clueless guys like ourselves to help speed this up.

Useful windows for analysis:

By default, IDA pro launches itself in the hex view window, but there are other windows available to us to help is analysis of IDA pro. Available windows are; Functions, strings, names, imports, exports and structures. Below are some key points which will help in the analysis;

1. Functions can be sorted by length or type. Ignore library functions and small functions.
2. Imports and Exports can be examined to check for further analysis.
3. Structures help us to see any active data structures in the sample.

Navigation:

1. Tool bar has forward or backward arrows.
2. Click on any location in the Navigation band (Color Coded).
3. Press G to go to any location.
4. Jump-File Offset to go to a location in the file.

**Cross-References:**

Cross references are instances of items from other windows being used in the current window and are denoted in the Hex-A window by using the word Xref.

Code Xref: Any opcode used in a subroutine.

Data Xref: Any variable used in another subroutine and how it is accessed through the sample.

Function Xref: Press x on the cross-reference window to see how many times a function was called. This is not working.

*Protip:*  One example of using preliminary static analysis to gather data to use for further samples is as below; use the string program to gather interesting strings and search for them in IDA pro. It will result in some interesting conditions.

Having been studying IDA pro and analyzing samples in a simpler way, we tried to understand small bits and pieces of how the malware/sample is being broken down into smaller chunks. Now we move to the next step in learning how to use IDA pro, Analyzing functions;

**Analyzing Functions:**

One of the key advantages to using IDA pro is that it can break down code into functions and related local variables and parameters. Examining the sample in page 130;

1. IDA pro notifies that this subroutine is using an EBP based stack frame.
2. Ida pro has recognized the arguments(prefix \_arg) and variables (prefix \_var) and also their offsets. These are calculated on the location of EBP and offsets of these variables are calculated from EBP. Variables are stored at negative offset relative to EBP and arguments are at positive off sets. Bp based frame? EBP vs BP based frame, I think it just 16 bit base pointer.

Sometime even IDA pro might not be able to recognize some of the functions. You have to be able to read the instructions and decide if it’s a subroutine or not.

**Using Graphing Options:**

Play with the graph options to see what is happening.

Page 133- This is just an introduction into IDA pro- there is a lot of information about using IDA pro and that can be gained from it. Since this is a topic of deductive reasoning- a lot of times you go through blind alleys. Keep at it and you shall be successful.

Need to execute labs for Ida pro.