

# SEC 530 IDA and Ollydbg

Before we start, please apply the following instructions.

**Change your background image!!**

How to change the background image:

**Win-7**

<https://www.dummies.com/computers/operating-systems/windows-7/how-to-change-the-desktop-background-in-windows-7/>

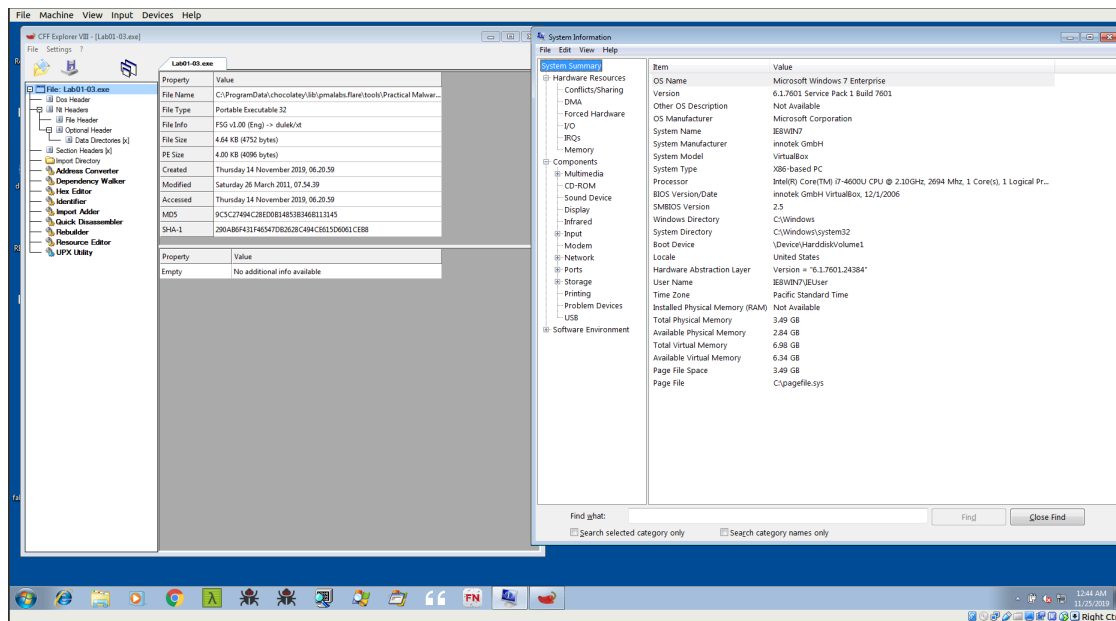
**Win-10**

<https://www.dummies.com/computers/operating-systems/windows-10/how-to-change-the-desktop-background-in-windows-10/>

**Don't forget to include your machine system information for every screenshot.**

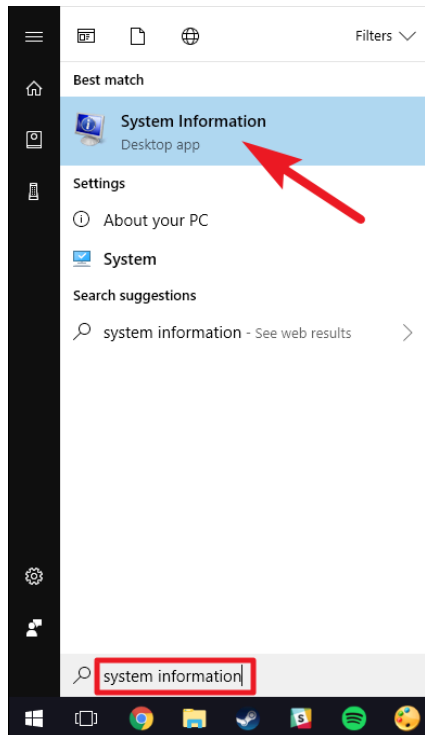
How to check system information on Windows 7: **msinfo32** command

<https://www.youtube.com/watch?v=-hfFTV0axl>



## In Window 10

Type "system information" into the initial search box, and then select the result.



<https://www.youtube.com/watch?v=wNTWNh4WyQk>

Take a look at the second way in the link above.

*Don't forget to take a snapshot before we start!!!*

*No SCREENSHOT no points!*

*Read the questions carefully!*

*If you don't have the tools, then install them!*

***SUBMISSION MUST BE A PDF FILE!!!***

*The objective of this lab is to familiarize yourself with IDA Pro.*

Files:

[https://drive.google.com/file/d/1afN4IFw\\_kbVCxzBOF-zd-lvfBsva29ez/view?usp=sharing](https://drive.google.com/file/d/1afN4IFw_kbVCxzBOF-zd-lvfBsva29ez/view?usp=sharing)

Binary : a1.exe

1.1 Use Basic Static Analysis to this malware's (interesting) internet related imports . What conclusions can you draw from them?

Explanation is expected!

Screenshot is expected!

1.2 In IDA, analyse the following functions.

1)->sub\_403611

2)->sub\_40122B

3)->sub\_402C7A

### **1.2.1: What is the purpose of each function?**

Explanation is expected!

Screenshot is expected!

### **1.2.2: Write the C code for the last two functions (2-3).**

C code and explanation is expected!

Screenshot is expected!

Binary : a2.exe

2.1 In IDA, analyze all the functions called by main.

### **2.1.1: What is the purpose of each function?**

C code and Explanation is expected!

Screenshot is expected

### **2.1.2: Write the C code for all.**

C code and Explanation is expected!

Screenshot is expected

3) Please download exe from the zip file and analyze the exe to answer the following questions.

Link :

[https://drive.google.com/file/d/1seA9fFI\\_Bw9N4a3sTg0F3YuzB-poeCka/view?usp=sharing](https://drive.google.com/file/d/1seA9fFI_Bw9N4a3sTg0F3YuzB-poeCka/view?usp=sharing)

Please be careful!! Real malware inside !!!!! This is not a worm as far as I know.

Password: malware

**3.1 Find the directory where malware resides after running it**

Take a screenshot and explain the way you found it

**3.2 Find out what has been added into the registry?**

Take a screenshot and explain the way you found it

**3.3 Find out how this malware achieved persistence?**

Take a screenshot and explain the way you found it

Also explained the persistence mechanism.

**Screenshots needed! For all the answers !**

Download the following : No Google account is needed ! Download this to WIN10

<https://drive.google.com/file/d/1fPLZ2ePdSRr5qBjzo1razQiMoBF1GG1d/view?usp=sharing>

**4.1 Describe the C constructs that you see in the 1.exe** (Convert 1.bin to 1.exe).

**Screenshot is needed! Explain your answer.**

**4.2 Write C code and explain the purpose of the code. What is exe trying to do?**

C code and assembly explanations are expected.

Explain your answer. **Screenshots needed!**

**5.1 In this question, describe the C constructs that you see in the 2.exe (Convert 2.bin to 2.exe).**

**Screenshot is needed! Explain your answer.**

## 5.2 Write C code and explain the purpose of the code. What is exe trying to do?

C code and assembly explanations are expected.

Explain your answer. **Screenshots needed!**

## 6.1 In this question, describe the C constructs that you see in the 3.exe (Convert 3.bin to 3.exe).

Snapshot is needed! Explain your answer.

## 6.2 Explain the purpose of the code. What is exe trying to do? Explain your answer. Screenshots needed!

7.1. Discover how you would run the executable 4.exe (Convert 4.bin to 4.exe)so that it prints “Yes!!!”.

What kind of input do you need ?

First find a way to run exe on the command line(cmd) then find a way to print “Yes!!!”.

## Ollydbg

To download OllyDbg, you must go to OllyDbg website (<https://www.ollydbg.de/>) and the [Download](#) part. In the bottom you will see [Download OllyDbg 1.10](#) (final version). Please download the OllyDbg and extract it in a dedicated folder. After downloading and extracting into a dedicated file, you can run it by double clicking on ollydbg.exe.

## So What is OllyDbg ?

- An x86 debugger developed by Oleh Yuschuk.
- Provides the ability to analyze malware while it is running.
- Commonly used by malware analysts and reverse engineers because
  - it's free,
  - it's easy to use,
  - and it has many plugins that extend its capabilities.

### Small History

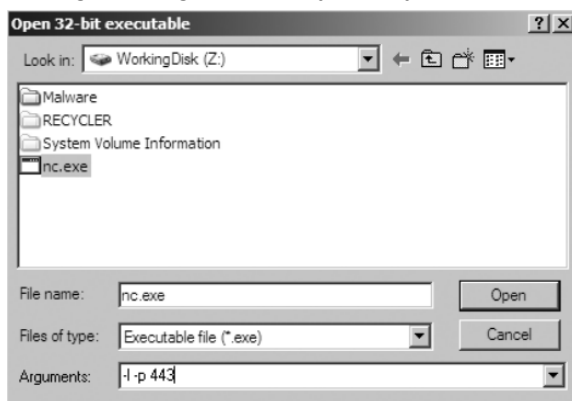
- OllyDbg was developed more than a decade ago
- First used to crack software and to develop exploits
  - Before malware analysis
- The OllyDbg 1.1 source code was purchased by Immunity and rebranded as Immunity

## Debugger

- The two products are very similar
  - Everything you'll learn in this chapter applies to both.
- The only item of note is that many plugins for OllyDbg won't automatically run in ImmDbg.

## Loading Malware

- There are several ways to begin debugging malware with OllyDbg.
  - You can load executables and even DLLs directly.
- If the malware is already running, you can attach OllyDbg to the running process.
- OllyDbg provides a flexible system to run malware with command-line options or to execute specific functionality within a DLL.
- File, Open (Easiest way to debug malware)
  - then browse to the executable you wish to load
- Add command-line arguments if needed
  - Specify them in the Arguments field of the Open dialog.
- During loading is the only time you can pass command-line arguments to OllyDbg.



## Opening an Executable (EXE)

- Once you've opened an executable,
  - OllyDbg will load the binary using its own loader.
  - This works similarly to the way that the Windows OS loads a file.
- OllyDbg will stop at the entry point, WinMain, if it can be determined.
- Otherwise it will break at the entry point defined in the PE Header

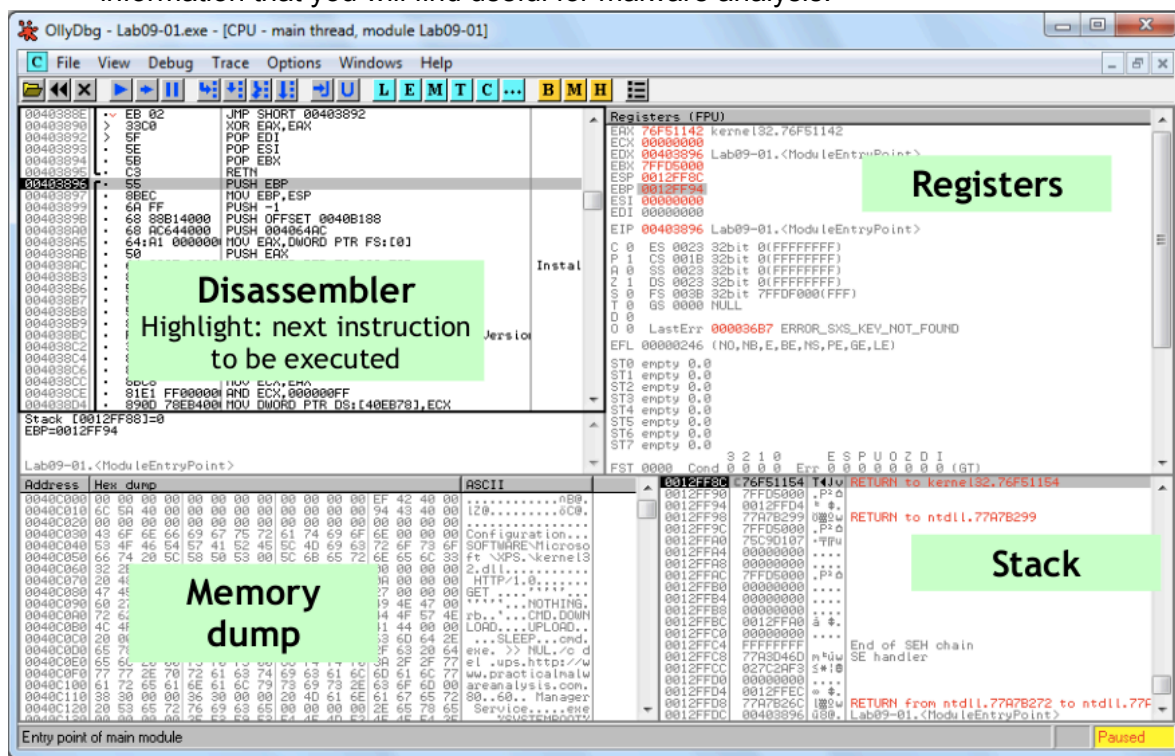
## Attaching to a Running Process

- In addition to opening an executable directly, you can attach OllyDbg to a running process.
  - Useful when you want to debug running malware.
- To attach OllyDbg to a process, select "File" then "Attach"
- This will bring up a menu in which you can select the process to which you want to attach.

- OllyDbg breaks in and pauses the program and all threads
  - If you catch it in DLL, set a breakpoint on access to the entire code section to get to the interesting code.

## The OllyDbg Interface

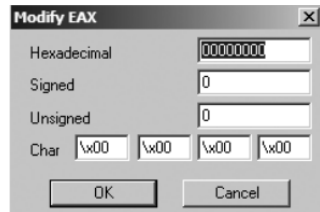
- As soon as you load a program into OllyDbg, you will see four windows filled with information that you will find useful for malware analysis.



- Disassembler window
  - This window shows the debugged program's code
  - Current instruction pointer with several instructions before and after it.
  - Next instruction to be executed will be highlighted in this window.
  - To modify or add instructions or data

- press the spacebar within this window.

- Registers window
  - This window shows the current state of the registers for the debugged program.
  - As in the disassembler window, you can modify data in the registers window as the program is debugged by right-clicking any register value and selecting Modify.

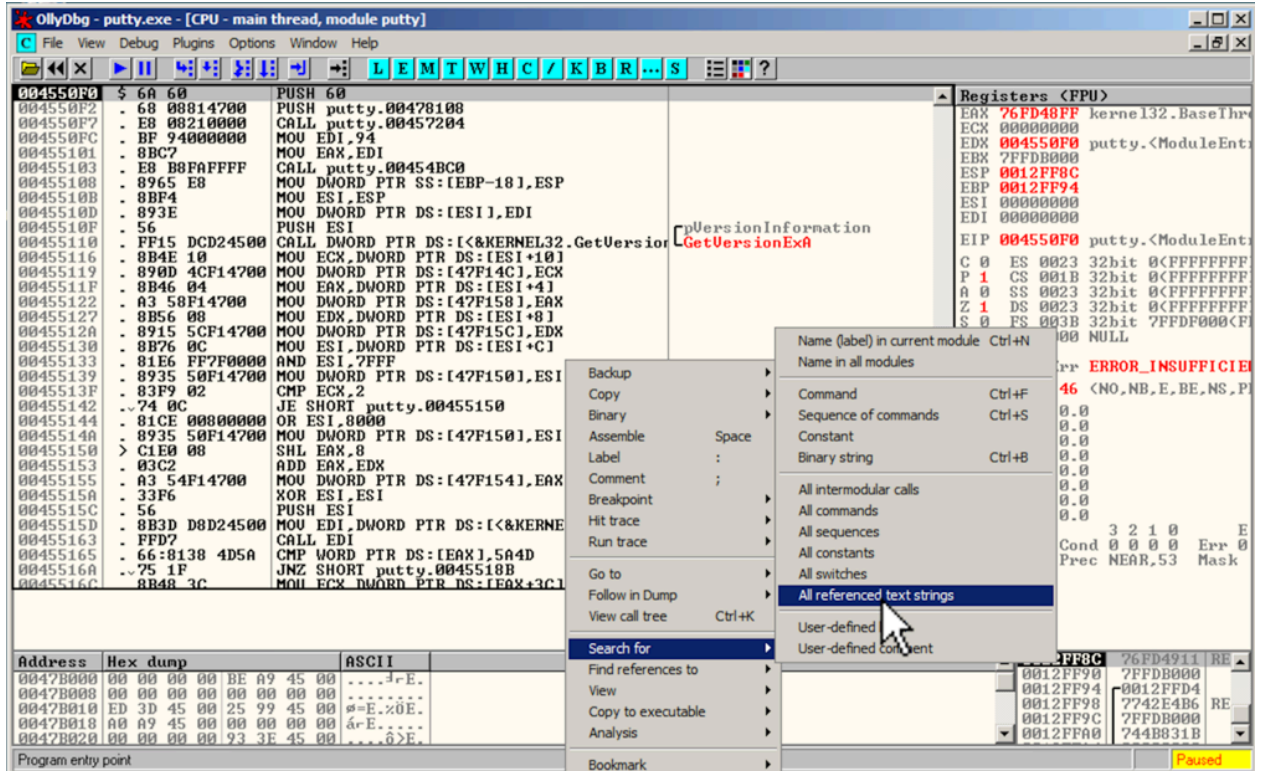


- Stack window
  - This window shows the current state of the stack in memory for the thread being debugged.
  - You can manipulate stacks in this window by right-clicking a stack location and selecting Modify.
  - OllyDbg places useful comments on some stack locations that describe the arguments placed on the stack
- Memory dump window
  - This window shows a dump of live memory for the debugged process.
  - Ctrl+G to go to a memory location
  - To edit memory in this window, right-click it and choose "Binary" then "Edit".

## Searching in Ollydbg (Strings)

- In Ollydbg, in the "Assembly Code" pane, right-click. Point to "**Search for**".
- Click "**All referenced text strings**", as shown below.





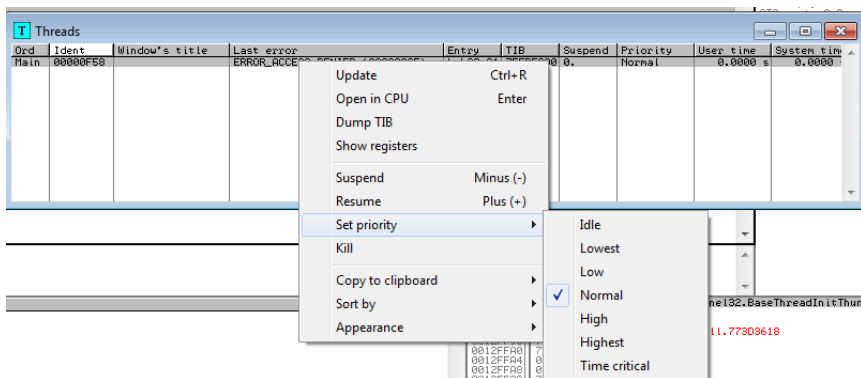
## Searching in Ollydbg (Imports)

- In Ollydbg, in the "Assembly Code" pane, right-click. Point to "**Search for**".
- Click "**Names**", as shown below.
  - Ctrl + N -> Names

Names in putty					
Address	Section	Type	Ordinal	Name	Comments
00400000		Analysér		<STRUCT IMAGE_DOS_HEADER>	
00400100		Analysér		<STRUCT IMAGE_NT_SIGNATURE>	
00400104		Analysér		<STRUCT IMAGE_FILE_HEADER>	
00400118		Analysér		<STRUCT IMAGE_OPTIONAL_HEADER>	
00400178		Analysér		<STRUCT IMAGE_DATA_DIRECTORY>	
00400178		Analysér		<STRUCT IMAGE_SECTION_HEADER>	
00400220		Analysér		<STRUCT IMAGE_SECTION_HEADER>	
00400248		Analysér		<STRUCT IMAGE_SECTION_HEADER>	
00400270		Analysér		<STRUCT IMAGE_SECTION_HEADER>	
00450000	.text	Export		<ModuleEntryPoint>	
00450000	.rdata	Import		&ADVAPI32.RegCloseKey	
00450004	.rdata	Import		&ADVAPI32.RegQueryValueExA	
00450008	.rdata	Import		&ADVAPI32.RegOpenKeyA	
0045000C	.rdata	Import		&ADVAPI32.GetUserNAmeA	
00450010	.rdata	Import		&ADVAPI32.EqualSid	
00450014	.rdata	Import		&ADVAPI32.CopySid	
00450018	.rdata	Import		&ADVAPI32.GetLengthSid	
0045001C	.rdata	Import		&ADVAPI32.AllocateAndInitializeSid	
00450020	.rdata	Import		&ADVAPI32.SetSecurityDescriptorDacl	
00450024	.rdata	Import		&ADVAPI32.SetSecurityDescriptorOwner	
00450028	.rdata	Import		&ADVAPI32.InitializeSecurityDescriptor	
0045002C	.rdata	Import		&ADVAPI32.RegCreateKeyA	
00450030	.rdata	Import		&ADVAPI32.RegSetValueExA	
00450034	.rdata	Import		&ADVAPI32.RegDeleteKeyA	
00450038	.rdata	Import		&ADVAPI32.RegEnumKeyA	
0045003C	.rdata	Import		&ADVAPI32.RegDeleteValueA	
00450040	.rdata	Import		&ADVAPI32.RegCreateKeyExA	
00450048	.rdata	Import		&COMCTL32.LBItemFromPt	
0045004C	.rdata	Import		&COMCTL32.DrawInsert	
00450050	.rdata	Import		&COMCTL32.InitCommonControls	
00450054	.rdata	Import		&COMCTL32.MakeDragList	
00450058	.rdata	Import		&GDI32.CreateBitmap	
0045005C	.rdata	Import		&GDI32.IntersectClipRect	
00450060	.rdata	Import		&GDI32.ExcludeClipRect	
00450064	.rdata	Import		&GDI32.UpdateColors	
00450068	.rdata	Import		&GDI32.DeleteDC	
0045006C	.rdata	Import		&GDI32.GetTextExtentPoint32A	
00450070	.rdata	Import		&GDI32.CreateCompatibleDC	
00450074	.rdata	Import		&GDI32.DeleteObject	
00450078	.rdata	Import		&GDI32.TextOutA	
0045007C	.rdata	Import		&GDI32.SetBkColor	
00450080	.rdata	Import		&GDI32.SetTextColor	
00450084	.rdata	Import		&GDI32.Rectangle	
00450088	.rdata	Import		&GDI32.CreateSolidBrush	
0045008C	.rdata	Import		&GDI32.GetStockObject	
00450090	.rdata	Import		&GDI32.SelectObject	
00450094	.rdata	Import		&GDI32.CreateFontIndirectA	
00450098	.rdata	Import		&GDI32.GetTextExtentExPointA	
0045009C	.rdata	Import		&GDI32.SetMapMode	
004500A0	.rdata	Import		&GDI32.GetDeviceCaps	
004500A4	.rdata	Import		&GDI32.GetTextMetricsA	
004500A8	.rdata	Import		&GDI32.CreateFontA	
004500AC	.rdata	Import		&GDI32.RealizePalette	
004500B0	.rdata	Import		&GDI32.SelectPalette	
004500B4	.rdata	Import		&GDI32.CreatePalette	
004500B8	.rdata	Import		&GDI32.ExtTextOutA	
004500BC	.rdata	Import		&GDI32.GetCharacterPlacementW	
004500C0	.rdata	Import		&GDI32.SetBkMode	
004500C4	.rdata	Import		&GDI32.GetBkMode	
004500C8	.rdata	Import		&GDI32.ExtTextOutW	
004500CC	.rdata	Import		&GDI32.GetCharABCDWidthsFloatA	
004500D0	.rdata	Import		&GDI32.GetPixel	
004500D4	.rdata	Import		&GDI32.GetPixel	

## Viewing Threads and Stacks

- Malware often uses multiple threads.
- View the current threads within a program by selecting **View** then **Threads**
  - Threads window will come up.
- This window shows the memory locations of the threads and their **current status**:
  - **Active**
  - **Paused**
  - **Suspended**








- Since OllyDbg is single-threaded,
  - you might need to pause all of the threads

- set a breakpoint,
- and then continue to run the program in order to begin debugging within a particular thread.

## Executing Code

- There are many different ways to execute code in OllyDbg.
- Most popular methods

*Table 10-1. OllyDbg Code-Execution Options*

Function	Menu	Hotkey	Button
Run/Play	Debug ► Run	F9	
Pause	Debug ► Pause	F12	
Run to selection	Breakpoint ► Run to Selection	F4	
Run until return	Debug ► Execute till Return	CTRL-F9	
Run until user code	Debug ► Execute till User Code	ALT-F9	
Single-step/step-into	Debug ► Step Into	F7	
Step-over	Debug ► Step Over	F8	

## Execute till Return Option

- Pauses execution until just before the current function is set to return
  - This can be useful when you want a program to pause immediately after the current function is finished executing.
- But if the function never ends, the program will continue to run indefinitely

## The Execute till User Code Option

- **Useful if you get lost in library code during debugging**
- Program will continue to run until it hit compiled malware code
  - Typically the **.text** section
- Press **Debug** then **Execute till User Code**

## Stepping Through Code

- OllyDbg provides several ways to step through code.

- OllyDbg offers the two types of stepping described in the previous chapter:
  - Single-stepping (also known as stepping-into) (F7)
  - Stepping-over (F8)
- **Some malware is designed to fool you, by calling routines and never returning, so stepping over will miss the most important part.**
- For example, if you single-step the instruction call 01007568 , OllyDbg will pause at the address 01007568
  - Because the call instruction transferred EIP to that address.

## Breakpoint & Run & Pause

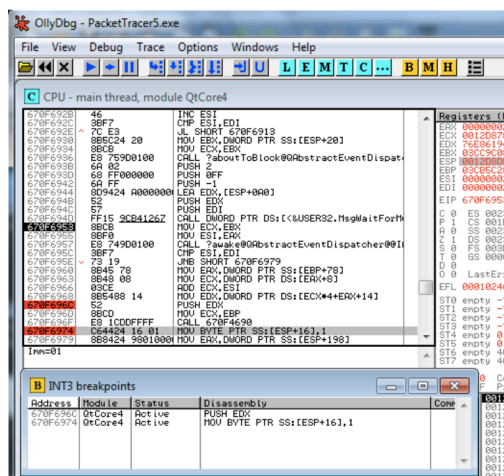
- The simplest options
- You could **Run** a program and click **Pause** when it's where you want it to be
- But that's sloppy and might leave you somewhere uninteresting,
  - such as inside library code.
- Setting breakpoints is much better (F2)
- **Run** is useful to resume execution after hitting a breakpoint
- **Run to Selection** option will execute until just before the selected instruction is executed
  - If the selection is never executed, it will run indefinitely

## Types of Breakpoints

- OllyDbg supports all of those types.
  - Software breakpoints
  - Hardware breakpoints
  - Conditional breakpoints
  - Breakpoints on memory
- You can **add** or **remove** a breakpoint by selecting the instruction in the disassembler window and pressing **F2**.

## Viewing Active Breakpoints

- Click **View**, then **Breakpoints**,
- or click **B** icon on toolbar



## OllyDbg Breakpoint Options

Function	Right-click menu selection	Hotkey
Software breakpoint	Breakpoint ► Toggle	F2
Conditional breakpoint	Breakpoint ► Conditional	SHIFT-F2
Hardware breakpoint	Breakpoint ► Hardware, on Execution	
Memory breakpoint on access (read, write, or execute)	Breakpoint ► Memory, on Access	F2 (select memory)
Memory breakpoint on write	Breakpoint ► Memory, on Write	

- After you close or terminate a debugged program, OllyDbg will typically save the breakpoint locations you set
  - If you open the same file again, the breakpoints are still available

## Software Breakpoints

- Software breakpoints are particularly useful when debugging a string decoder function.
- Malware authors often obfuscate strings
  - With a string decoder that is called before each string is used
  - Put a breakpoint at the end of the decoder routine
  - The string becomes readable on the stack each time you press Play in OllyDbg, the program will execute and will break when a string is decoded for use
  - This method will only reveal strings as they are used

```
push offset "4NNpTNHLKIXoPm7iBhUAjvRKNaUVBlr"  
call String_Decoder  
...  
push offset "ugKLdNlLT6emldCeZi72mUjieuBqdfZ"  
call String_Decoder  
...
```

## Conditional Breakpoints

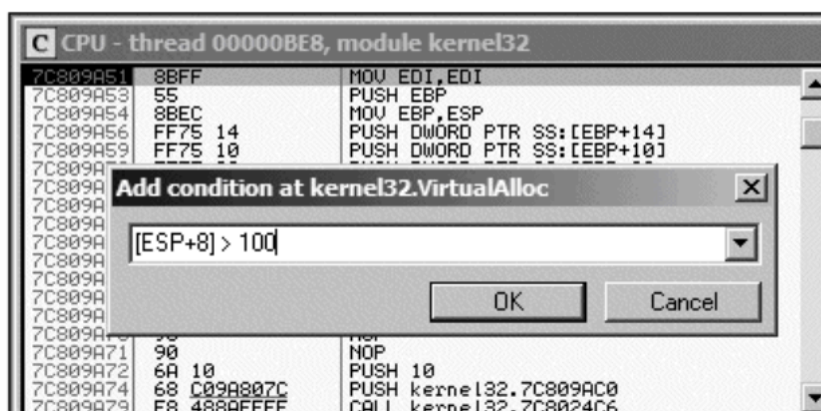
- Breaks only when a condition is true
- Conditional software breakpoints can be particularly useful when you want to save time when trying to pause execution once a certain parameter is passed.
- You can use conditional breakpoints to detect memory allocations above a certain size.
- Breaks only when memory allocations above a certain size.

## Conditional Breakpoints -*Poison Ivy* backdoor

- Ex: *Poison Ivy* backdoor
  - *Poison Ivy* allocates memory to house the shellcode it receives from Command and Control (C&C) servers
  - Most memory allocations are for other purposes and uninteresting
  - Set a conditional breakpoint at the VirtualAlloc function in Kernel32.dll
  - ***if you set a conditional breakpoint when the allocation size is greater than 100 bytes, the program will not pause when the smaller (and more frequent) memory allocations occur.***
  - Put a standard breakpoint at the start of the VirtualAlloc function
    - 4 parameters (Address, Size, AllocationType, Protect)

00C3FDB0	0095007C	CALL to VirtualAlloc from 00950079
00C3FDB4	00000000	Address = NULL
00C3FDB8	00000029	Size = 29 (41.)
00C3FDBC	00001000	AllocationType = MEM_COMMIT
00C3FDC0	00000040	Protect = PAGE_EXECUTE_READWRITE

1. Right-click in the disassembler window on the first instruction of the function, and select **Breakpoint ► Conditional**. This brings up a dialog asking for the conditional expression.
2. Set the expression and click **OK**. In this example, use **[ESP+8]>100**.
3. Click **Play** and wait for the code to break.



Top of the stack is pointed to by the ESP register in order to access the Size field, we must reference it in memory as [ESP+8].

## Hardware Breakpoints



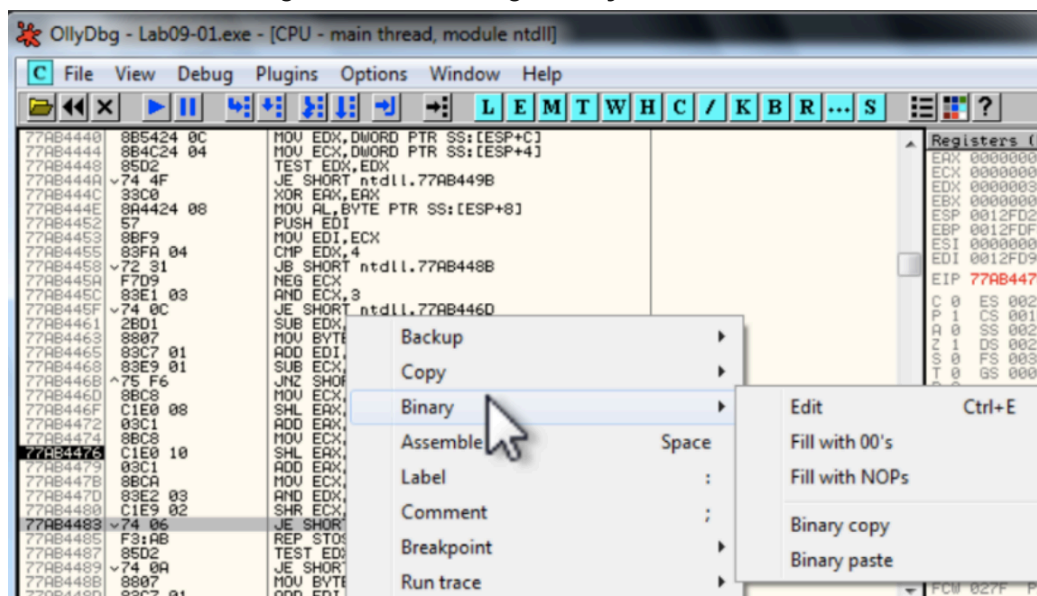
- Hardware breakpoints are powerful because they don't alter your code, stack, or any target resource.
  - Don't slow down execution speed.
  - **Problem with hardware breakpoints is that you can set only *four* at a time.**
  - Click **Breakpoint**, then **Hardware, on Execution**
- You can set OllyDbg to use hardware breakpoints by default in Debugging Options
  - Useful if malware uses anti-debugging techniques

## Memory Breakpoints

- Code breaks on access to specified memory location
- OllyDbg supports software and hardware memory breakpoints
- Can break on read, write, execute, or any access
- To set a basic memory breakpoint, select a portion of memory in the **memory dump window** or a section in the memory map
  - Right-click memory location, click **Breakpoint**, then **"Memory, on Access"**
  - You can only set one memory breakpoint at a time
  - OllyDbg implements software memory breakpoints by changing the attributes of memory blocks
  - This technique is not reliable and has considerable overhead
  - **Use memory breakpoints sparingly (Carefully).**

## Patching

- OllyDbg makes it easy to modify just about any live data
  - such as registers and flags.
- **Binary > Edit**
  - Modify instructions or memory by highlighting a region, right-clicking that region, and selecting **Binary** then **Edit**.

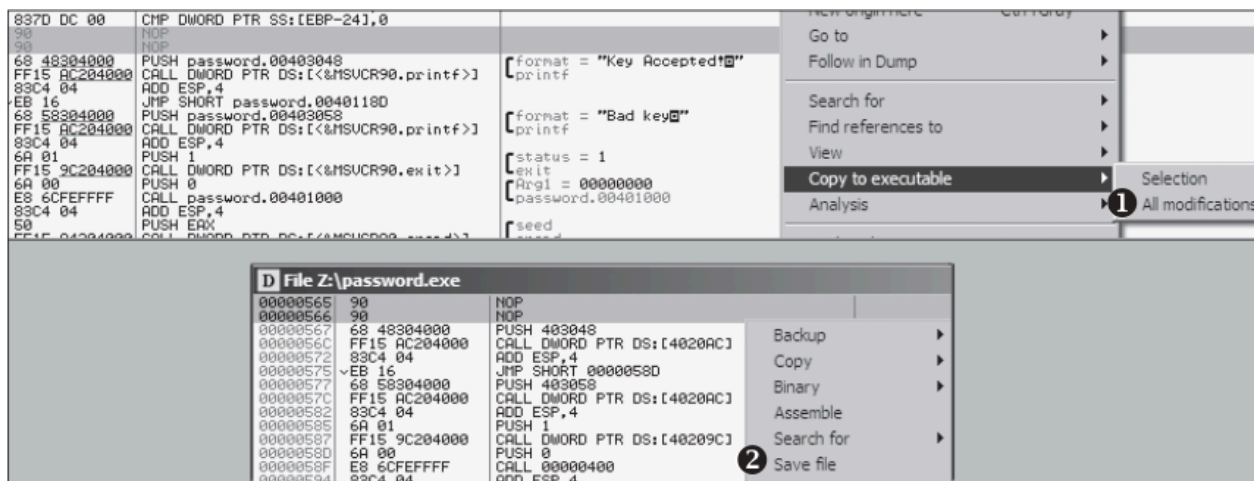


## Binary Edit

- Selecting **Binary** then **Edit**
- This will pop up a window for you to add any opcodes or data.
- OllyDbg also has special functions to fill with
  - 00 entries
  - NOP instructions
    - Used to skip instructions (To meet a condition in if else)
    - e.g. to force a branch

## Saving Patched Code

- Right-click disassembler window after patching
  - Select **Copy to Executable** then **All Modifications** as shown at (1)
- This will copy all changes you have made in live memory and pop up a new window
- Then Select **Save File**, as shown at (2), to save it to disk.



Following lab will use the same malware/binaries.

If not downloaded:

<https://drive.google.com/file/d/1fPLZ2ePdSRr5qBizo1razQiMoBF1GG1d/view?usp=sharing>

8.1 You will run the 1.exe on CMD and you will get 2 as output. Please patch the binary so that you will get 8 as the answer.

Explain your answer. Explain reason behind your answer

Screenshot is needed !!

Also saved, patched binary and included in your submission ! You can change the .exe with .bin. If you are zipping this, make sure that password is malware.

8.2 You will run the 2.exe on CMD and you will get 6 as output. Please patch the binary so that you will get 3 as the answer.



Explain your answer. Explain reason behind your answer  
Screenshot is needed !!

Also saved, patched binary and included in your submission ! You can change the .exe with .bin. If you are zipping this, make sure that password is malware.

8.3 You will run the 3.exe on CMD and you will get 9 as output. Please patch the binary so that you will get the result (number) associated with your name.

Explain your answer. Explain reason behind your answer  
Screenshot is needed !!

Also saved, patched binary and included in your submission ! You can change the .exe with .bin. If you are zipping this, make sure that password is malware.

Student ID	Output -1	Output - 2
28227	3	72
33295	2	62
29456	1	52
34937	6	88
34612	16	8721
28246	14	562
28370	13	243
29363	12	71
27366	18	113
28057	19	111
35136	12	112
29501	14	2
29181	16	222
34700	17	21
35022	18	212
3053	19	20
29377	2	99
34739	21	213
34791	15	131111
26490	87	8787
35125	14	243

8.4 You will run the 4.exe on CMD and you will get NO!! as output. Please patch the binary so that you will get YES!! Without any output!

Explain your answer. Explain reason behind your answer

Screenshot is needed !!

Also saved, patched binary and included in your submission ! You can change the .exe with .bin. If you are zipping this, make sure that password is malware.