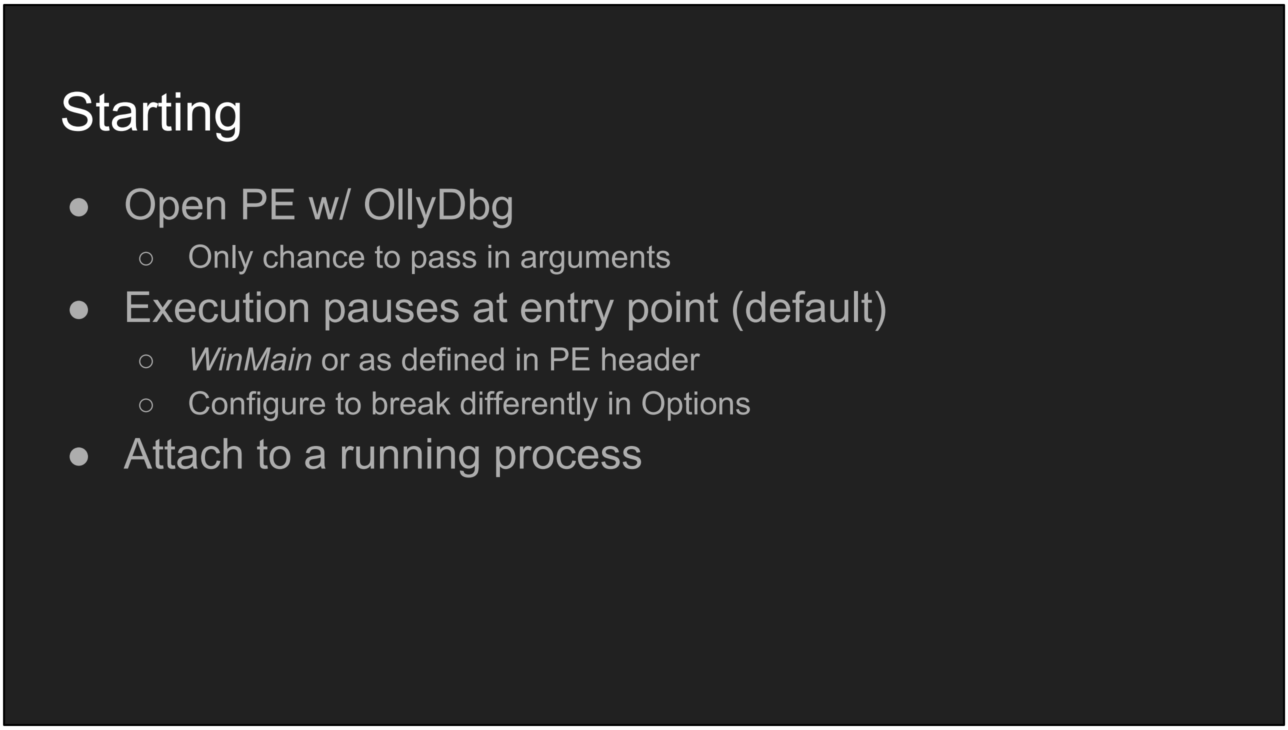
Practical Malware Analysis

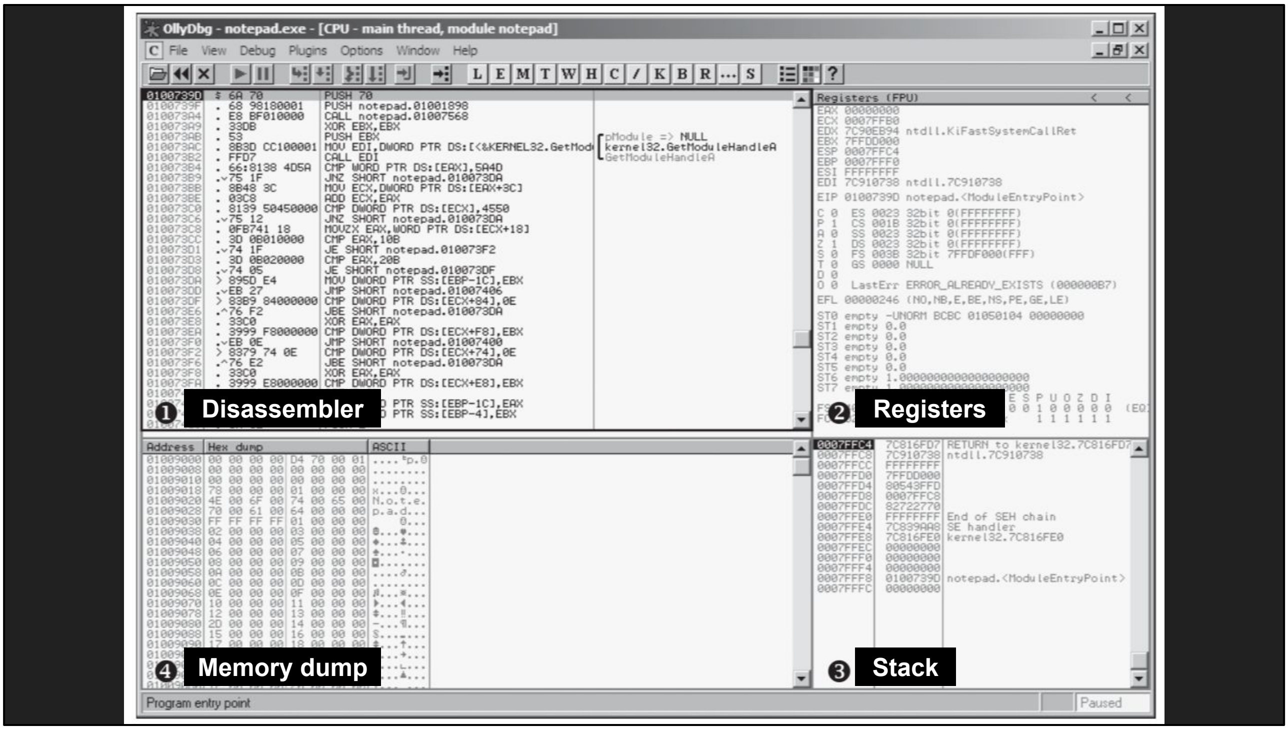
Lecture 9 | OllyDbg



Starting

- Open PE w/ OllyDbg
  - Only chance to pass in arguments
- Execution pauses at entry point (default)
  - *WinMain* or as defined in PE header
  - Configure to break differently in Options
- Attach to a running process

To begin debugging with OllyDbg, you can simply open a PE file with Olly and pass in any arguments, Olly will pause execution at the entry point defined by the developer or, if that can’t be found, the entry point defined in the PE header. Olly can also be told to break elsewhere, such as before any code executes at all. You may also attach Olly to a running process using the File > Attach option and selecting the PID of the desired process.m


OllyDbg - notepad.exe - [CPU - main thread, module notepad]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
1 Disassembler
2 Registers
3 Stack
4 Memory dump
Program entry point
Paused

```
01007390D $ 6A 70          PUSH 70
0100739F  . 68 98180001    PUSH notepad.01001898
010073A4  . E8 BF010000    CALL notepad.01007568
010073A9  . 33DB           XOR EBX,EBX
010073AB  . 53             PUSH EBX                                    [pModule => NULL
010073AC  . 8B3D CC100001  MOV EDI,DWORD PTR DS:[<&KERNEL32.GetMod     kernel32.GetModuleHandleA
010073B2  . FFD7           CALL EDI                                    GetModuleHandleA
010073B4  . 66:8138 4D5A   CMP WORD PTR DS:[EAX],5A4D
010073B9  .v75 1F          JNZ SHORT notepad.010073DA
010073BB  . 8B48 3C        MOV ECX,DWORD PTR DS:[EAX+3C]
010073BE  . 03C8           ADD ECX,EAX
010073C0  . 8139 50450000  CMP DWORD PTR DS:[ECX],4550
010073C6  .v75 12          JNZ SHORT notepad.010073DA
010073C8  . 0FB741 18      MOVZX EAX,WORD PTR DS:[ECX+18]
010073CC  . 3D 0B010000    CMP EAX,10B
010073D1  .v74 1F          JE SHORT notepad.010073F2
010073D3  . 3D 0B020000    CMP EAX,20B
010073D8  .v74 05          JE SHORT notepad.010073DF
010073DA  > 895D E4        MOV DWORD PTR SS:[EBP-1C],EBX
010073DD  .vEB 27          JMP SHORT notepad.01007406
010073DF  > 83B9 84000000  CMP DWORD PTR DS:[ECX+84],0E
010073E6  .^76 F2          JBE SHORT notepad.010073DA
010073E8  . 33C0           XOR EAX,EAX
010073EA  . 3999 F8000000  CMP DWORD PTR DS:[ECX+F8],EBX
010073F0  .vEB 0E          JMP SHORT notepad.01007400
010073F2  > 8379 74 0E     CMP DWORD PTR DS:[ECX+74],0E
010073F6  .^76 E2          JBE SHORT notepad.010073DA
010073F8  . 33C0           XOR EAX,EAX
010073FA  . 3999 E8000000  CMP DWORD PTR DS:[ECX+E8],EBX
010074
                           D PTR SS:[EBP-1C],EAX
                           D PTR SS:[EBP-4],EBX
```

```
Registers (FPU)
EAX 00000000
ECX 0007FFB0
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFDD000
ESP 0007FFC4
EBP 0007FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738

EIP 0100739D notepad.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_ALREADY_EXISTS (000000B7)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM BCBC 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
                 E S P U O Z D I
FS               0 0 1 0 0 0 0 0  (EQ
FC                   1 1 1 1 1 1
```

```
Address   Hex dump                 ASCII
01009000  00 00 00 00|D4 70 00 01  ....ᵗp.☺
01009008  00 00 00 00|00 00 00 00  ........
01009010  00 00 00 00|00 00 00 00  ........
01009018  78 00 00 00|01 00 00 00  x...☺...
01009020  4E 00 6F 00|74 00 65 00  N.o.t.e.
01009028  70 00 61 00|64 00 00 00  p.a.d...
01009030  FF FF FF FF|01 00 00 00      ☺...
01009038  02 00 00 00|03 00 00 00  ☻...♥...
01009040  04 00 00 00|05 00 00 00  ♦...♣...
01009048  06 00 00 00|07 00 00 00  ♠...•...
01009050  08 00 00 00|09 00 00 00  ◘.......
01009058  0A 00 00 00|0B 00 00 00  ....♂...
01009060  0C 00 00 00|0D 00 00 00  ........
01009068  0E 00 00 00|0F 00 00 00  ♫...☼...
01009070  10 00 00 00|11 00 00 00  ►...◄...
01009078  12 00 00 00|13 00 00 00  ↕...‼...
01009080  2D 00 00 00|14 00 00 00  -...¶...
01009088  15 00 00 00|16 00 00 00  §...▬...
01009090  17 00 00 00|18 00 00 00  ↨...↑...
```

```
0007FFC4   7C816FD7  RETURN to kernel32.7C816FD7
0007FFC8   7C910738  ntdll.7C910738
0007FFCC   FFFFFFFF
0007FFD0   7FFDD000
0007FFD4   80543FFD
0007FFD8   0007FFC8
0007FFDC   82722770
0007FFE0   FFFFFFFF  End of SEH chain
0007FFE4   7C839AA8  SE handler
0007FFE8   7C816FE0  kernel32.7C816FE0
0007FFEC   00000000
0007FFF0   00000000
0007FFF4   00000000
0007FFF8   0100739D  notepad.<ModuleEntryPoint>
0007FFFC   00000000
```

The Interface

At (1) the Disassembler window displays the debugged process's code, with a few lines before and after the instruction pointer showing. The next instruction to be executed is typically highlighted. Instructions and data may be added or modified with [space].

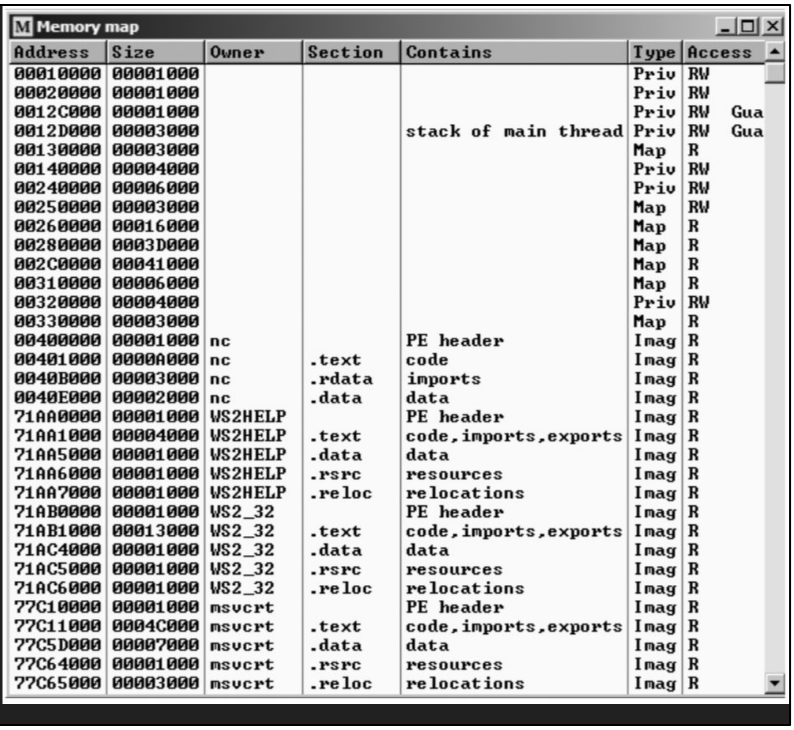
(2), the Registers window, shows the registers in their current state, updating with each instruction that executes. Registers can be modified with a right-click.
(3) shows the current state of the Stack for the thread being debugged, starting at the top of the stack, which can be modified with a right-click.

The memory dump window (4) shows a live view of memory used by the process. CTRL+G will allow you to dump memory for any given address. Memory can also be modified with a right-click, binary -> edit.

Memory Map

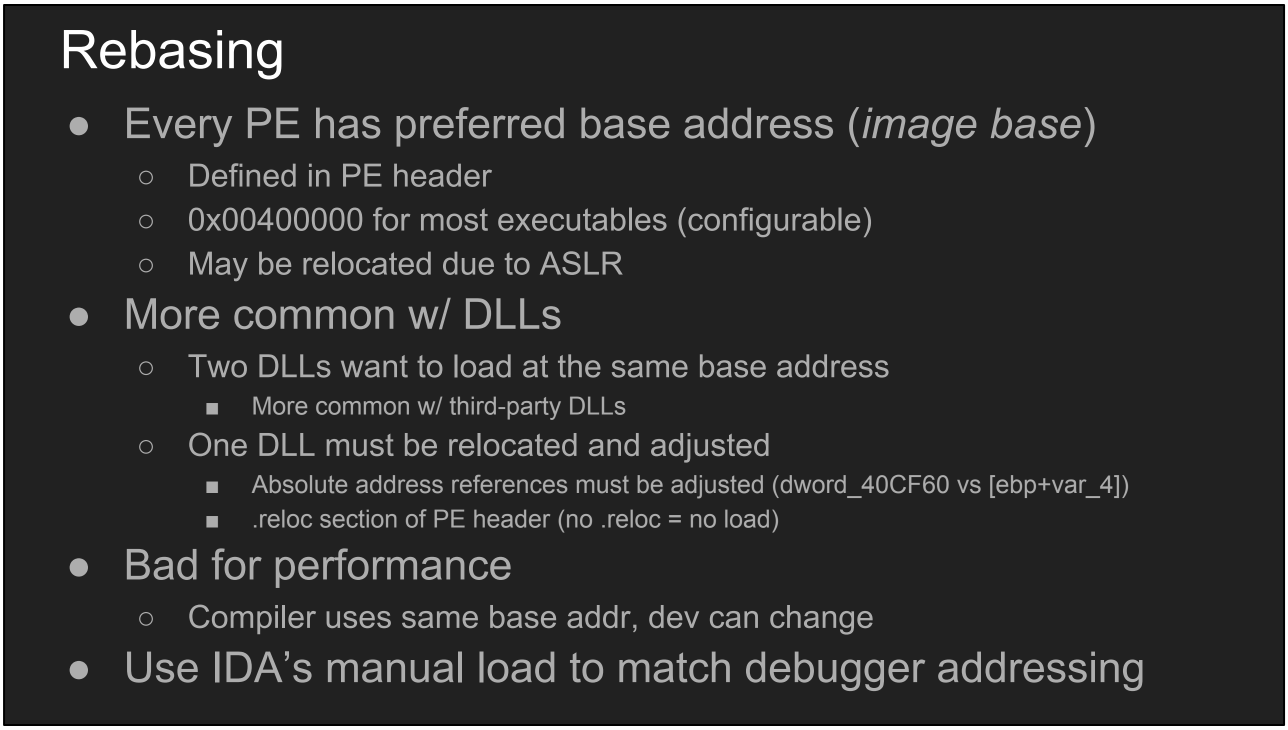
View -> Memory, shows all block allocated by program

- See how a program is in memory (includes DLLs)
- Double-click for memory dump
- Right-click to view in disassembler



Address	Size	Owner	Section	Contains	Type	Access
00010000	00001000				Priv	RW
00020000	00001000				Priv	RW
0012C000	00001000				Priv	RW Gua
0012D000	00003000			stack of main thread	Priv	RW Gua
00130000	00003000				Map	R
00140000	00004000				Priv	RW
00240000	00006000				Priv	RW
00250000	00003000				Map	RW
00260000	00016000				Map	R
00280000	0003D000				Map	R
002C0000	00041000				Map	R
00310000	00006000				Map	R
00320000	00004000				Priv	RW
00330000	00003000				Map	R
00400000	00001000	nc		PE header	Imag	R
00401000	0000A000	nc	.text	code	Imag	R
0040B000	00003000	nc	.rdata	imports	Imag	R
0040E000	00002000	nc	.data	data	Imag	R
71AA0000	00001000	WS2HELP		PE header	Imag	R
71AA1000	00004000	WS2HELP	.text	code,imports,exports	Imag	R
71AA5000	00001000	WS2HELP	.data	data	Imag	R
71AA6000	00001000	WS2HELP	.rsrc	resources	Imag	R
71AA7000	00001000	WS2HELP	.reloc	relocations	Imag	R
71AB0000	00001000	WS2_32		PE header	Imag	R
71AB1000	00013000	WS2_32	.text	code,imports,exports	Imag	R
71AC4000	00001000	WS2_32	.data	data	Imag	R
71AC5000	00001000	WS2_32	.rsrc	resources	Imag	R
71AC6000	00001000	WS2_32	.reloc	relocations	Imag	R
77C10000	00001000	msvcrt		PE header	Imag	R
77C11000	0004C000	msvcrt	.text	code,imports,exports	Imag	R
77C5D000	00007000	msvcrt	.data	data	Imag	R
77C64000	00001000	msvcrt	.rsrc	resources	Imag	R
77C65000	00003000	msvcrt	.reloc	relocations	Imag	R

View -> Memory, shows all block allocated by program
See how a program is in memory (includes DLLs)
Double-click for memory dump
Right-click to view in disassembler



Rebasing

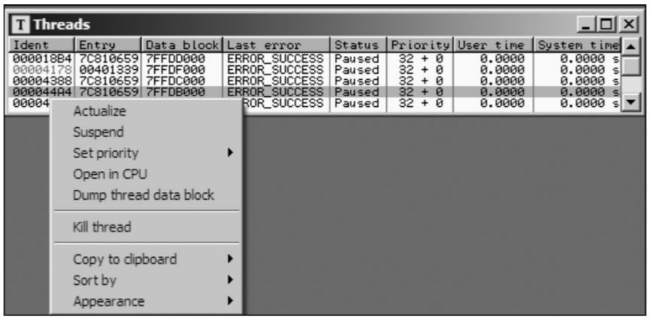
- Every PE has preferred base address (*image base*)
  - Defined in PE header
  - 0x00400000 for most executables (configurable)
  - May be relocated due to ASLR
- More common w/ DLLs
  - Two DLLs want to load at the same base address
    - More common w/ third-party DLLs
  - One DLL must be relocated and adjusted
    - Absolute address references must be adjusted (dword_40CF60 vs [ebp+var_4])
    - .reloc section of PE header (no .reloc = no load)
- Bad for performance
  - Compiler uses same base addr, dev can change
- Use IDA’s manual load to match debugger addressing

Rebasing

Every PE has preferred base address (image base)
Defined in PE header
0x00400000 for most executables (configurable)
May be relocated due to ASLR
More common w/ DLLs
Two DLLs want to load at the same base address
More common w/ third-party DLLs
One DLL must be relocated and adjusted
Absolute address references must be adjusted (dword_40CF60 vs [ebp+var_4])
.reloc section of PE header (no .reloc = no load)
Bad for performance
Compiler uses same base addr, dev can change
Use IDA’s manual load to match debugger addressing

Threads and Stacks

- View -> threads (and current status)
    - Olly uses one thread
        - Pause all threads, set break, resume to debug w/i another thread
        - Kill threads w/ right-click
    - Each thread has its own stack
        - Use memory map to view



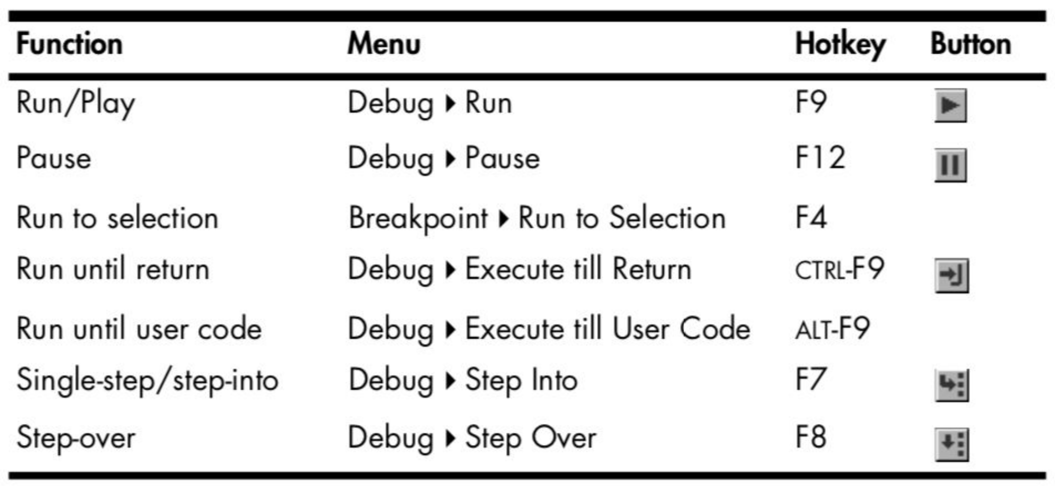
Threads

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
000018B4	7C810659	7FFDD000	ERROR_SUCCESS	Paused	32 + 0	0.0000	0.0000 s
00004178	00401339	7FFDF000	ERROR_SUCCESS	Paused	32 + 0	0.0000	0.0000 s
00004388	7C810659	7FFDC000	ERROR_SUCCESS	Paused	32 + 0	0.0000	0.0000 s
000044A4	7C810659	7FFDB000	ERROR_SUCCESS	Paused	32 + 0	0.0000	0.0000 s
00004			ROR_SUCCESS	Paused	32 + 0	0.0000	0.0000 s

Figure 9-6: Threads window showing five paused threads and the context menu for an individual thread

View -> threads (and current status)
Olly uses one thread
Pause all threads, set break, resume to debug w/i another thread
Kill threads w/ right-click
Each thread has its own stack
Use memory map to view

Running Code



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	↓

To run code in Ollly:

Run/Play simply starts execution from wherever it is stopped at
Pause pauses execution, but in a unspecific place that is generally not as helpful as breakpoints
Run to selection executes code until just before the selected instruction executes (unless it doesn’t, then indefinitely)
Run until return is pretty self-explanatory
Run until user code is useful when you get caught debugging library code (typically returns to .text section code)

Single-step will move execution forward by one instruction, step into will do the same, entering a function call if that’s where the cursor is
Step-over can be used to move to the other side of a function call using a hidden breakpoint, if the function being stepped-over never returns, the program may run indefinitely.

Breakpoints

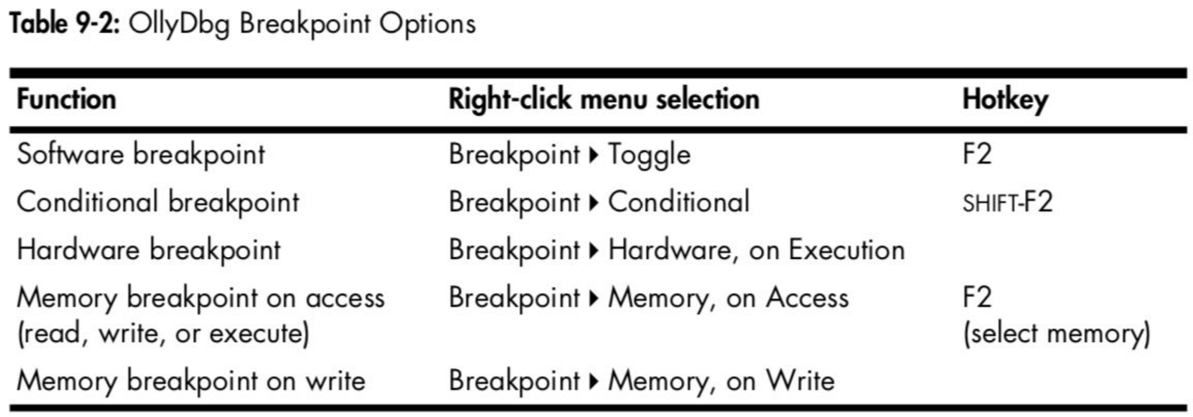


Table 9-2: OllyDbg Breakpoint Options

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	

Olly uses software breakpoints by default, but can also set hardware breakpoints and up to one memory breakpoint (at a time). Memory breakpoints can be configured to break on read, write, execute, or any access, though any memory break requires a lot of overhead.

Ex. set memory break on a DLLs .text section to see when its code is executed

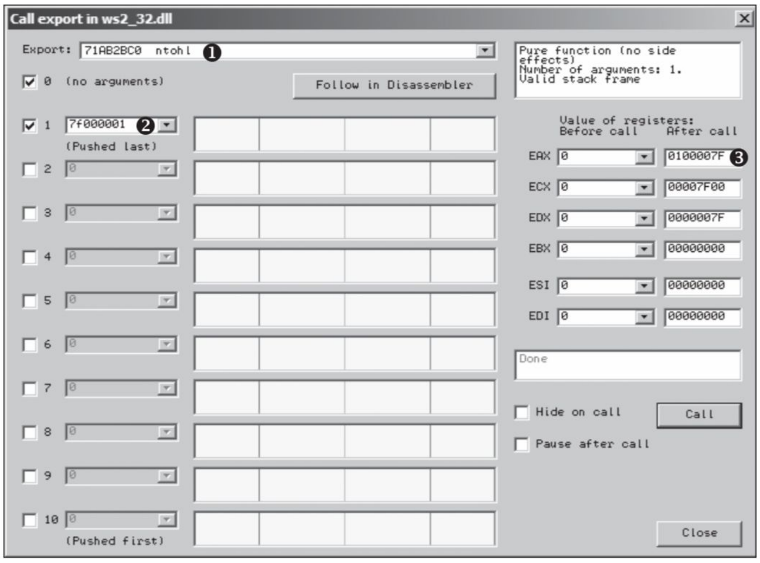
View active breakpoints with View -> breakpoints or the [B] icon
Set and remove breakpoints by selecting an instruction (or memory) and pressing [F2]

Olly usually saves breakpoints, allowing you to resume debugging a program without resetting breakpoints

Conditional breakpoints require an expression such as [ESP+8] > 100 (if we're waiting for an argument greater than 100)

Loading DLLs

- *loaddll.exe* dummy program
  - Breaks at *DllMain*
  - Run DllMain
  - Call specific exports w/ Debug -> Call DLL Export



Call export in ws2_32.dll

Export: 71AB2BC0 ntohl ❶

☑ 0 (no arguments)

☑ 1 7f000001 ❷ (Pushed last)

□ 2 0

□ 3 0

□ 4 0

□ 5 0

□ 6 0

□ 7 0

□ 8 0

□ 9 0

□ 10 0 (Pushed first)

Pure function (no side effects)
Number of arguments: 1.
Valid stack frame

Value of registers:

	Before call	After call
EAX	0	0100007F ❸
ECX	0	00007F00
EDX	0	0000007F
EBX	0	00000000
ESI	0	00000000
EDI	0	00000000

Done

□ Hide on call

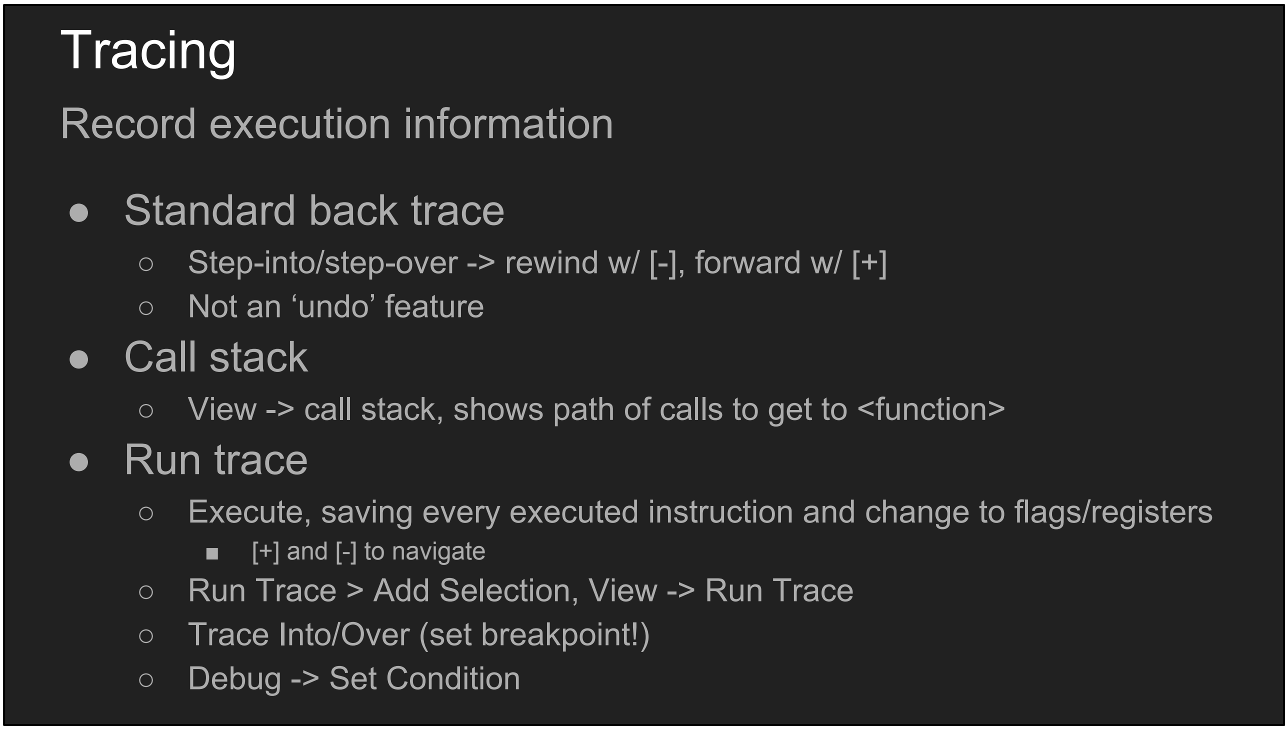
□ Pause after call

loaddll.exe dummy program

Breaks at DllMain

Run DllMain

Call specific exports w/ Debug -> Call DLL Export



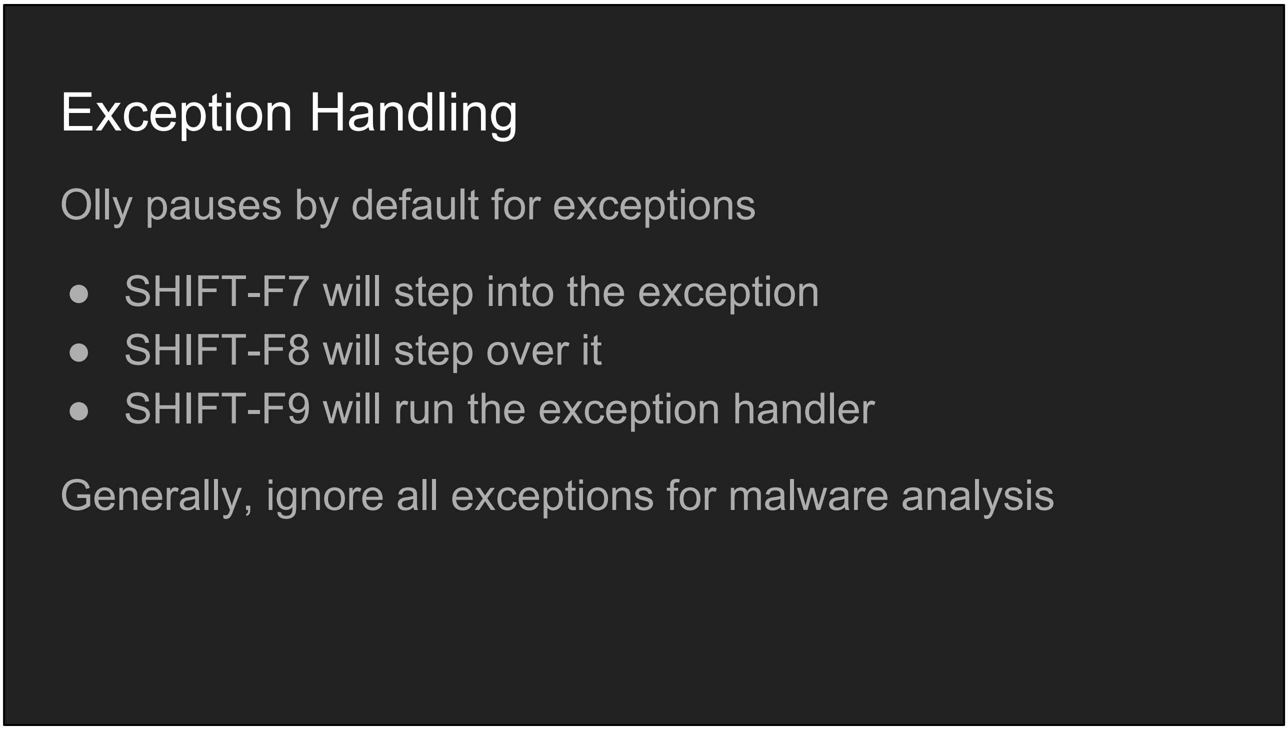
Tracing

Record execution information

- Standard back trace
  - Step-into/step-over -> rewind w/ [-], forward w/ [+]
  - Not an ‘undo’ feature
- Call stack
  - View -> call stack, shows path of calls to get to <function>
- Run trace
  - Execute, saving every executed instruction and change to flags/registers
    - [+] and [-] to navigate
  - Run Trace > Add Selection, View -> Run Trace
  - Trace Into/Over (set breakpoint!)
  - Debug -> Set Condition

Tracing
Record execution information

Standard back trace
    Step-into/step-over -> rewind w/ [-], forward w/ [+]
    Not an ‘undo’ feature
Call stack
    View -> call stack, shows path of calls to get to <function>
Run trace
    Execute, saving every executed instruction and change to flags/registers
        [+] and [-] to navigate
    Run Trace > Add Selection, View -> Run Trace
    Trace Into/Over (set breakpoint!)
    Debug -> Set Condition



Exception Handling

Olly pauses by default for exceptions

- SHIFT-F7 will step into the exception
- SHIFT-F8 will step over it
- SHIFT-F9 will run the exception handler

Generally, ignore all exceptions for malware analysis

Olly pauses by default for exceptions

SHIFT-F7 will step into the exception

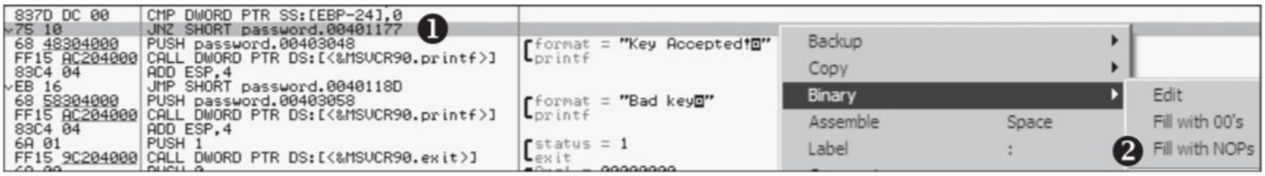
SHIFT-F8 will step over it

SHIFT-F9 will run the exception handler

Generally, ignore all exceptions for malware analysis

Patching

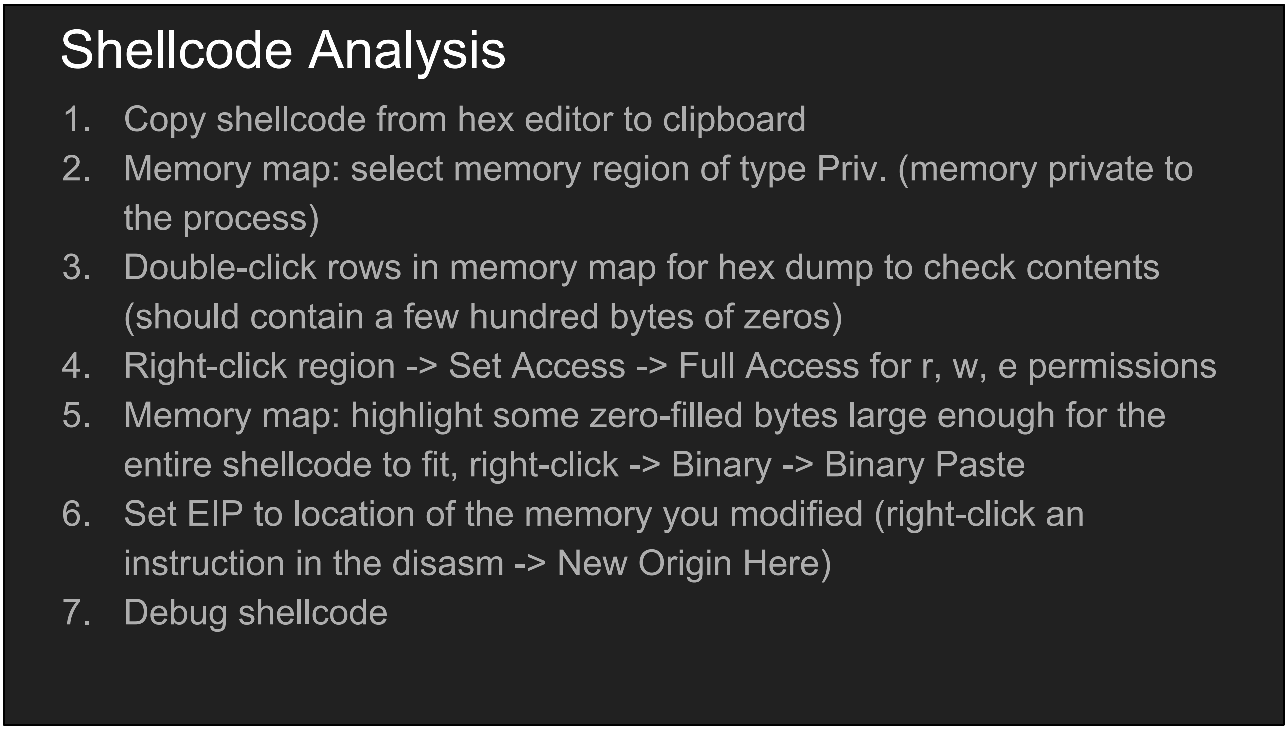
- Modify instr/data w/ right-click -> binary -> edit
  - Patch exists only in memory unless saved to a file



```
 837D DC 00      CMP DWORD PTR SS:[EBP-24],0
v75 10           JNZ SHORT password.00401177
 68 48304000     PUSH password.00403048             [format = "Key Accepted!◘"
 FF15 AC204000   CALL DWORD PTR DS:[<&MSVCR90.printf>]  printf
 83C4 04         ADD ESP,4
vEB 16           JMP SHORT password.0040118D
 68 58304000     PUSH password.00403058             [format = "Bad key◘"
 FF15 AC204000   CALL DWORD PTR DS:[<&MSVCR90.printf>]  printf
 83C4 04         ADD ESP,4
 6A 01           PUSH 1                             [status = 1
 FF15 9C204000   CALL DWORD PTR DS:[<&MSVCR90.exit>]    exit
```

Figure 9-13: Patching options in OllyDbg

Patching:
Modify instructions or data by selecting it, right-click -> binary -> edit
Patch exists only in memory unless saved to a file

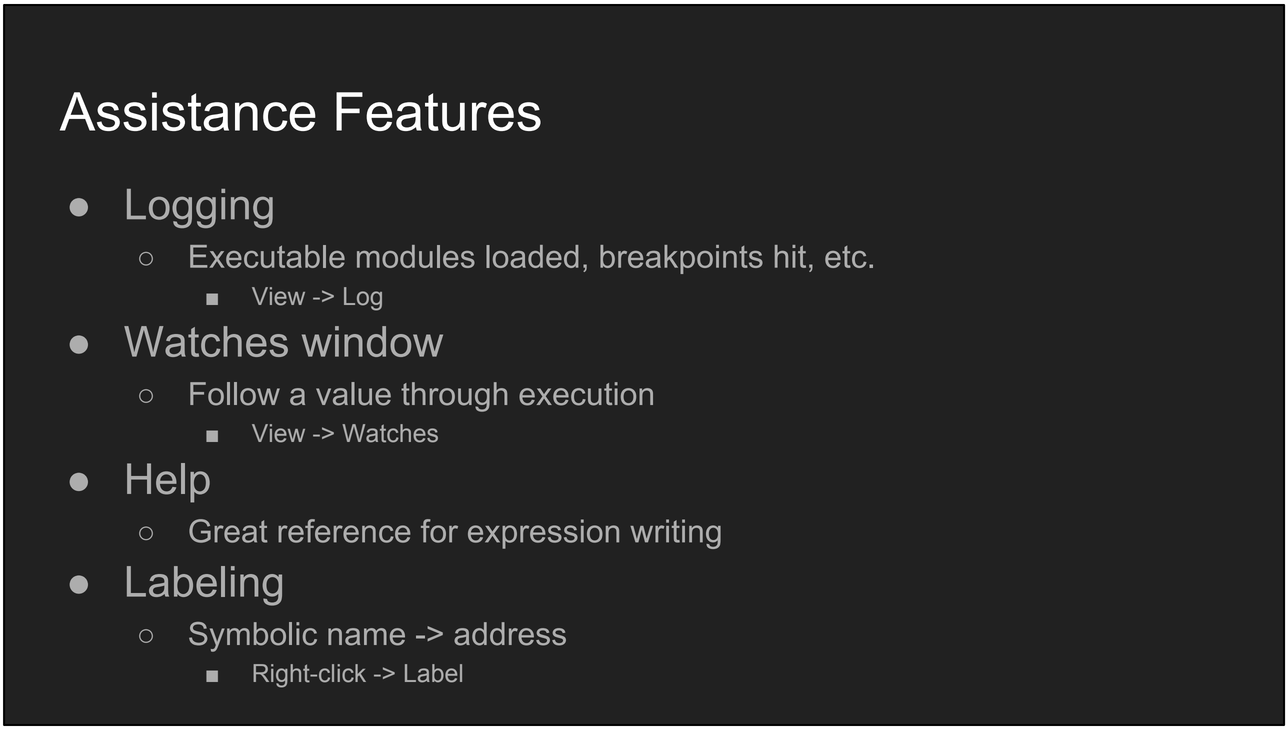


Shellcode Analysis

1. Copy shellcode from hex editor to clipboard
2. Memory map: select memory region of type Priv. (memory private to the process)
3. Double-click rows in memory map for hex dump to check contents (should contain a few hundred bytes of zeros)
4. Right-click region -> Set Access -> Full Access for r, w, e permissions
5. Memory map: highlight some zero-filled bytes large enough for the entire shellcode to fit, right-click -> Binary -> Binary Paste
6. Set EIP to location of the memory you modified (right-click an instruction in the disasm -> New Origin Here)
7. Debug shellcode

Olly’s easy (undocumented) shellcode analysis:

1. Copy shellcode from hex editor to clipboard
2. Memory map: select memory region of type Priv. (memory private to the process)
3. Double-click rows in memory map for hex dump to check contents (should contain a few hundred bytes of zeros)
4. Right-click region -> Set Access -> Full Access for r, w, e permissions
5. Memory map: highlight some zero-filled bytes large enough for the entire shellcode to fit, right-click -> Binary -> Binary Paste
6. Set EIP to location of the memory you modified (right-click an instruction in the disasm -> New Origin Here)
7. Debug shellcode

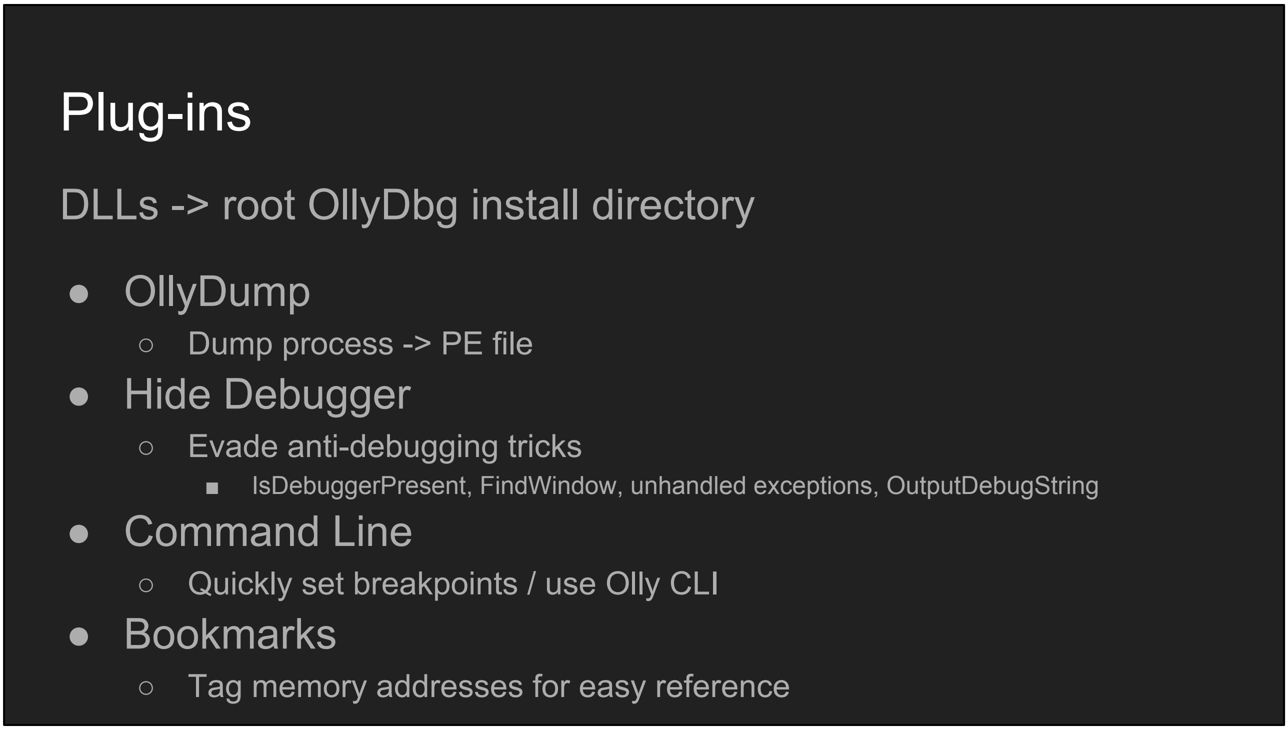


Assistance Features

- Logging
  - Executable modules loaded, breakpoints hit, etc.
    - View -> Log
- Watches window
  - Follow a value through execution
    - View -> Watches
- Help
  - Great reference for expression writing
- Labeling
  - Symbolic name -> address
    - Right-click -> Label

Assistance Features

Logging
Executable modules loaded, breakpoints hit, etc.
View -> Log
Watches window
Follow a value through execution
View -> Watches
Help
Great reference for expression writing
Labeling
Symbolic name -> address
Right-click -> Label



Plug-ins

DLLs -> root OllyDbg install directory

- OllyDump
  - Dump process -> PE file
- Hide Debugger
  - Evade anti-debugging tricks
    - IsDebuggerPresent, FindWindow, unhandled exceptions, OutputDebugString
- Command Line
  - Quickly set breakpoints / use Olly CLI
- Bookmarks
  - Tag memory addresses for easy reference

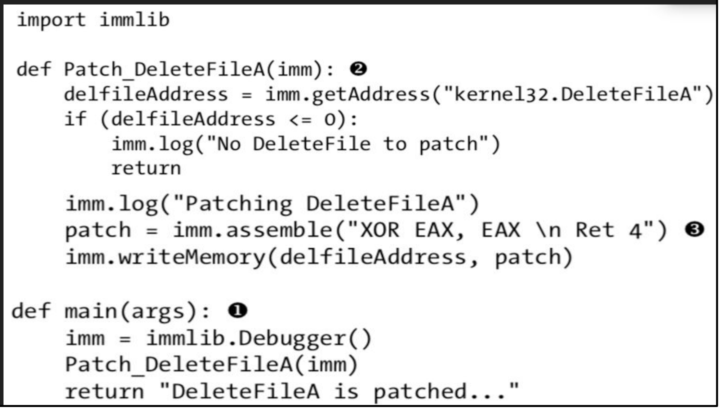
Plug-ins

DLLs -> root OllyDbg install directory
OllyDump
Dump process -> PE file
Hide Debugger
Evade anti-debugging tricks
IsDebuggerPresent, FindWindow, unhandled exceptions,
OutputDebugString
Command Line
Quickly set breakpoints / use Olly CLI
Bookmarks
Tag memory addresses for easy reference

Scriptable Debugging

Recommend ImmDbg for Python support + API

- Anti-debugger patching, inline function hooking, etc
- *PyCommand*
  - Execute from command bar w/ *!*



```
import immlib

def Patch_DeleteFileA(imm): ❷
    delfileAddress = imm.getAddress("kernel32.DeleteFileA")
    if (delfileAddress <= 0):
        imm.log("No DeleteFile to patch")
        return

    imm.log("Patching DeleteFileA")
    patch = imm.assemble("XOR EAX, EAX \n Ret 4") ❸
    imm.writeMemory(delfileAddress, patch)

def main(args): ❶
    imm = immlib.Debugger()
    Patch_DeleteFileA(imm)
    return "DeleteFileA is patched..."
```

Recommend ImmDbg for Python support + API
Anti-debugger patching, inline function hooking, etc
PyCommand
Execute from command bar w/ !

Resources

.NET Framework 4 Standalone Installer

WinDbg Standalone Installer

OllyDbg Installer

Olly Plug-ins

Gray Hat Python by Justin Seitz (No Starch Press, 2009)