School of

Engineering and Computer Science

Te Kura Mātai Pūkaha, Pūrorohiko

CYBR 473 T1 2023

Malware and Reverse Engineering

OllyDbg

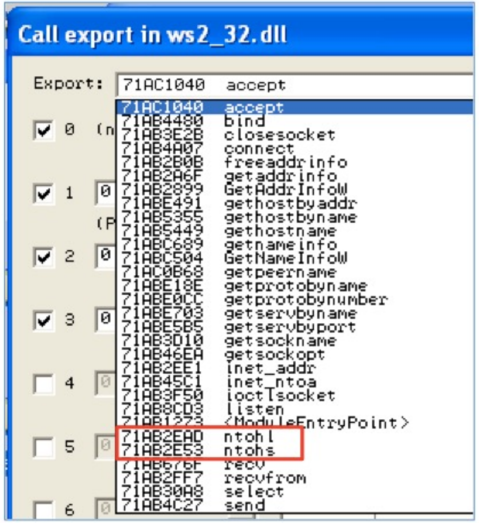
Chapter 9: “*Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software*”, Michael Sikorski and Andrew Honig, 2012

History

- OllyDbg was developed more than a decade ago
- First used to crack software and to develop exploits
- The OllyDbg 1.1 source code was purchased by Immunity and rebranded as Immunity Debugger
- The two products are very similar

Don't Use OllyDbg 2!

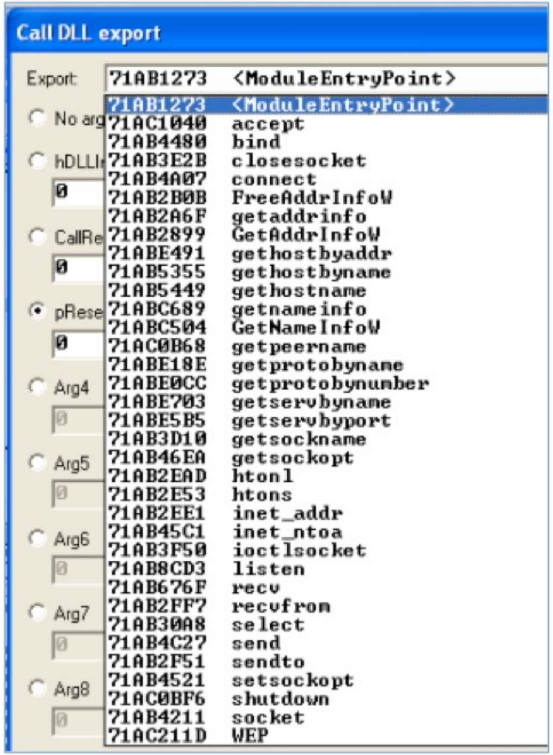
OllyDbg 1.10



Call export in ws2_32.dll

```
Export:  71AC1040   accept

71AC1040   accept
71AB4480   bind
71AB3E2B   closesocket
71AB4A07   connect
71AB2B0B   freeaddrinfo
71AB2A6F   getaddrinfo
71AB2899   GetAddrInfoW
71ABE491   gethostbyaddr
71AB5355   gethostbyname
71AB5449   gethostname
71ABC689   getnameinfo
71ABC504   GetNameInfoW
71AC0B68   getpeername
71ABE18E   getprotobyname
71ABE0CC   getprotobynumber
71ABE703   getservbyname
71ABE5B5   getservbyport
71AB3D10   getsockname
71AB46EA   getsockopt
71AB2EE1   inet_addr
71AB45C1   inet_ntoa
71AB3F50   ioctlsocket
71AB8CD3   listen
71AB1273   <ModuleEntryPoint>
71AB2EAD   ntohl
71AB2E53   ntohs
71AB676F   recv
71AB2FF7   recvfrom
71AB30A8   select
71AB4C27   send
```

OllyDbg 2.01



Call DLL export

```
Export  71AB1273   <ModuleEntryPoint>

71AB1273   <ModuleEntryPoint>
71AC1040   accept
71AB4480   bind
71AB3E2B   closesocket
71AB4A07   connect
71AB2B0B   FreeAddrInfoW
71AB2A6F   getaddrinfo
71AB2899   GetAddrInfoW
71ABE491   gethostbyaddr
71AB5355   gethostbyname
71AB5449   gethostname
71ABC689   getnameinfo
71ABC504   GetNameInfoW
71AC0B68   getpeername
71ABE18E   getprotobyname
71ABE0CC   getprotobynumber
71ABE703   getservbyname
71ABE5B5   getservbyport
71AB3D10   getsockname
71AB46EA   getsockopt
71AB2EAD   htonl
71AB2E53   htons
71AB2EE1   inet_addr
71AB45C1   inet_ntoa
71AB3F50   ioctlsocket
71AB8CD3   listen
71AB676F   recv
71AB2FF7   recvfrom
71AB30A8   select
71AB4C27   send
71AB2F51   sendto
71AB4521   setsockopt
71AC0BF6   shutdown
71AB4211   socket
71AC211D   WEP
```

LOADING MALWARE

Ways to Debug Malware – *Load*

- You can load EXEs or DLLs directly into OllyDbg

- Opening and EXE
    - File, Open
    - Add command-line arguments if needed
    - 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
        - Configurable in Options, Debugging Options

Ways to Debug Malware (cont.) – *Attach*

- If the malware is already running, you can attach OllyDbg to the running process

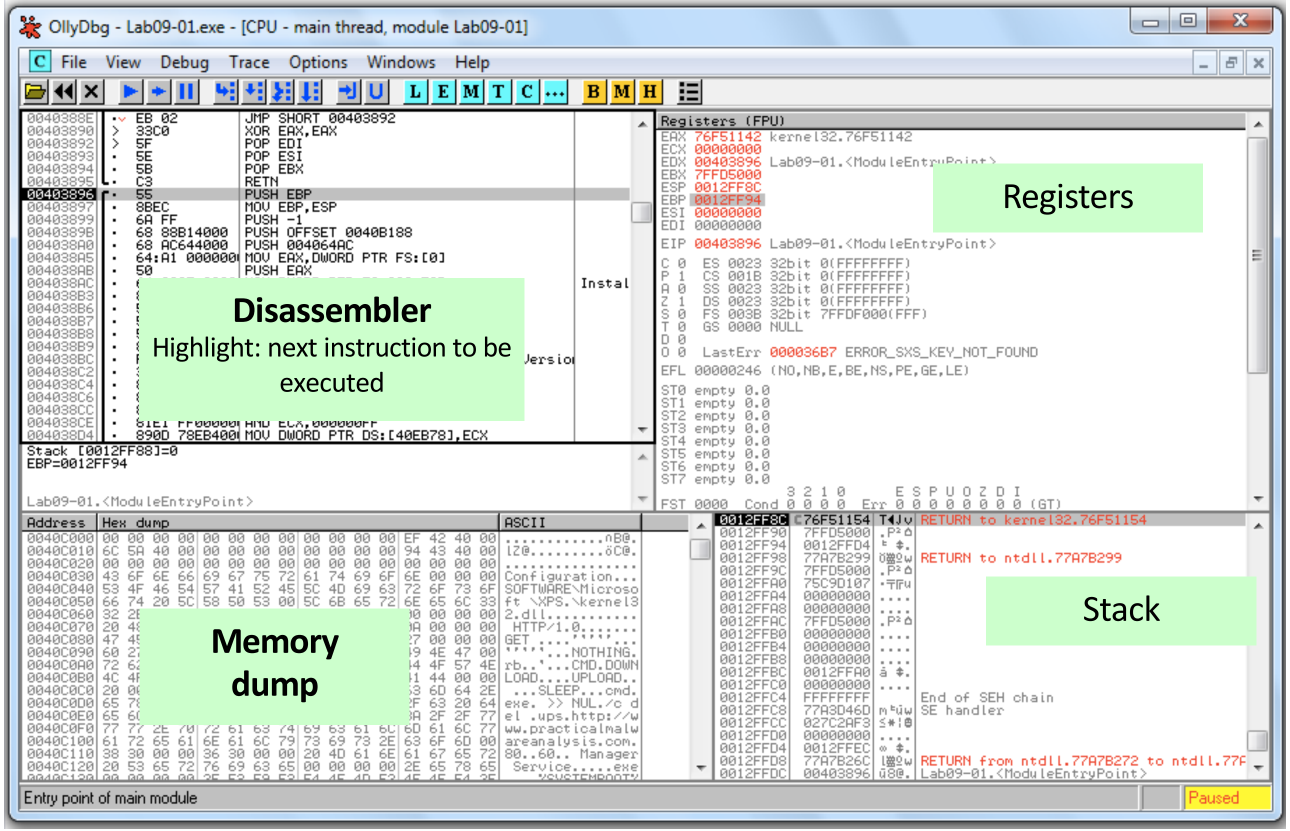
- Attaching to a running process
  - File, Attach
  - OllyDbg breaks in and pauses the program and all threads
    - If you catch it in DLL, set a breakpoint on access to the entire code section to get to the interesting code

Reloading a File

- Ctrl+F2 reloads the current executable
- F2 sets a breakpoint

THE OLLYDBG INTERFACE

The OllyDbg Interface


OllyDbg - Lab09-01.exe - [CPU - main thread, module Lab09-01]
C File View Debug Trace Options Windows Help
L E M T C ... B M H

```
00403888E  .v  EB 02          JMP SHORT 00403892
00403890   >   33C0           XOR EAX,EAX
00403892   >   5F             POP EDI
00403893   .   5E             POP ESI
00403894   .   5B             POP EBX
00403895  L.   C3             RETN
00403896  r.   55             PUSH EBP
00403897   .   8BEC           MOV EBP,ESP
00403899   .   6A FF          PUSH -1
0040389B   .   68 88B14000    PUSH OFFSET 0040B188
004038A0   .   68 AC644000    PUSH 004064AC
004038A5   .   64:A1 000000   MOV EAX,DWORD PTR FS:[0]
004038AB   .   50             PUSH EAX
004038AC   .                                                  Instal
004038B3   .
004038B6   .
004038B7   .
004038B8   .
004038B9   .
004038BC   .                                                 Versio
004038C2   .
004038C4   .
004038C6   .
004038CC   .
004038CE   .   81E1 FF000000  AND ECX,000000FF
004038D4   .   890D 78EB400   MOV DWORD PTR DS:[40EB78],ECX
```

```
Stack [0012FF88]=0
EBP=0012FF94

Lab09-01.<ModuleEntryPoint>
```

```
Registers (FPU)
EAX 76F51142 kernel32.76F51142
ECX 00000000
EDX 00403896 Lab09-01.<ModuleEntryPoint>
EBX 7FFD5000
ESP 0012FF8C
EBP 0012FF94
ESI 00000000
EDI 00000000

EIP 00403896 Lab09-01.<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 000036B7 ERROR_SXS_KEY_NOT_FOUND

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

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0 (GT)
```

```
Address   Hex dump                                          ASCII
0040C000  00 00 00 00 00 00 00 00 00 00 00 00 EF 42 40 00  ............nB@.
0040C010  6C 5A 40 00 00 00 00 00 00 00 00 00 94 43 40 00  lZ@.........öC@.
0040C020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C030  43 6F 6E 66 69 67 75 72 61 74 69 6F 6E 00 00 00  Configuration...
0040C040  53 4F 46 54 57 41 52 45 5C 4D 69 63 72 6F 73 6F  SOFTWARE\Microso
0040C050  66 74 20 5C 58 50 53 00 5C 6B 65 72 6E 65 6C 33  ft \XPS.\kernel3
0040C060  32 2E                               00 00 00 00  2.dll...........
0040C070  20 48                               0A 00 00 00   HTTP/1.0.......
0040C080  47 45                               27 00 00 00  GET ....''''...
0040C090  60 27                               49 4E 47 00  ''''...NOTHING.
0040C0A0  72 62                               44 4F 57 4E  rb..'...CMD.DOWN
0040C0B0  4C 4F                               41 44 00 00  LOAD....UPLOAD..
0040C0C0  20 00                               63 6D 64 2E   ...SLEEP...cmd.
0040C0D0  65 78                               2F 63 20 64  exe. >> NUL./c d
0040C0E0  65 6C                               3A 2F 2F 77  el .ups.http://w
0040C0F0  77 77 2E 70 72 61 63 74 69 63 61 6C 6D 61 6C 77  ww.practicalmalw
0040C100  61 72 65 61 6E 61 6C 79 73 69 73 2E 63 6F 6D 00  areanalysis.com.
0040C110  38 30 00 00 36 30 00 00 20 4D 61 6E 61 67 65 72  80..60.. Manager
0040C120  20 53 65 72 76 69 63 65 00 00 00 00 2E 65 78 65   Service.....exe
```

```
0012FF8C  76F51154  T◄Jv  RETURN to kernel32.76F51154
0012FF90  7FFD5000  .P²⌂
0012FF94  0012FFD4  ╘ ‡.
0012FF98  77A7B299  ŏ▒♀w  RETURN to ntdll.77A7B299
0012FF9C  7FFD5000  .P²⌂
0012FFA0  75C9D107  •╤╔u
0012FFA4  00000000  ....
0012FFA8  00000000  ....
0012FFAC  7FFD5000  .P²⌂
0012FFB0  00000000  ....
0012FFB4  00000000  ....
0012FFB8  00000000  ....
0012FFBC  0012FFA0  á ‡.
0012FFC0  00000000  ....
0012FFC4  FFFFFFFF        End of SEH chain
0012FFC8  77A3D46D  m╘ûw  SE handler
0012FFCC  027C2AF3  ≤*¦☻
0012FFD0  00000000  ....
0012FFD4  0012FFEC  ∞ ‡.
0012FFD8  77A7B26C  l▒♀w  RETURN from ntdll.77A7B272 to ntdll.77F
0012FFDC  00403896  û8@.  Lab09-01.<ModuleEntryPoint>
```

Entry point of main module

Paused

Disassembler

Highlight: next instruction to be executed

Registers

Memory dump

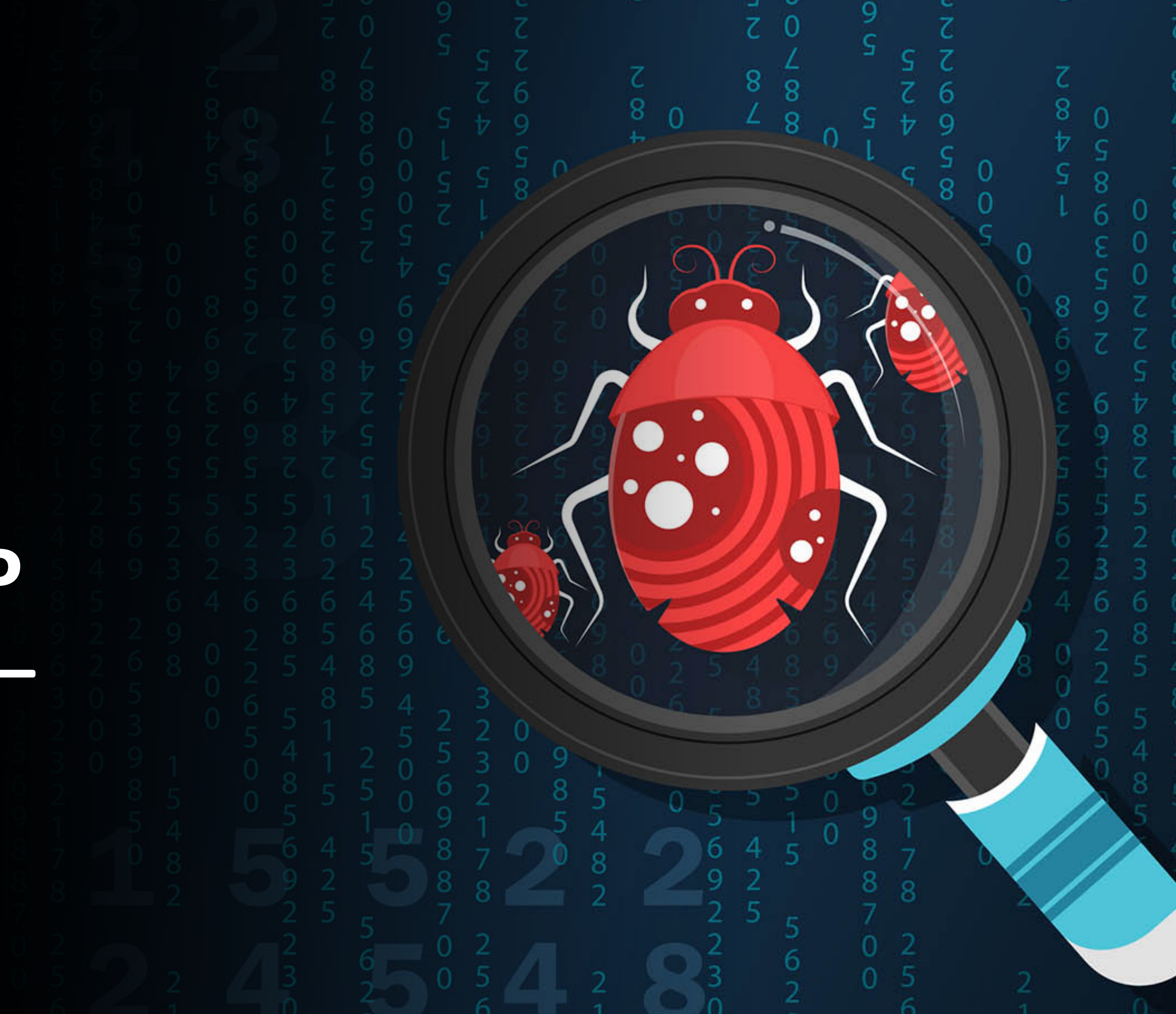
Stack

Modifying Data

- Disassembler window
  - Press **spacebar**
- Registers or Stack
  - **Right-click**, modify
- Memory dump
  - **Right-click**, Binary, Edit
  - Ctrl+G to go to a memory location
  - **Right-click** a memory address in another pane and click “Follow in dump”

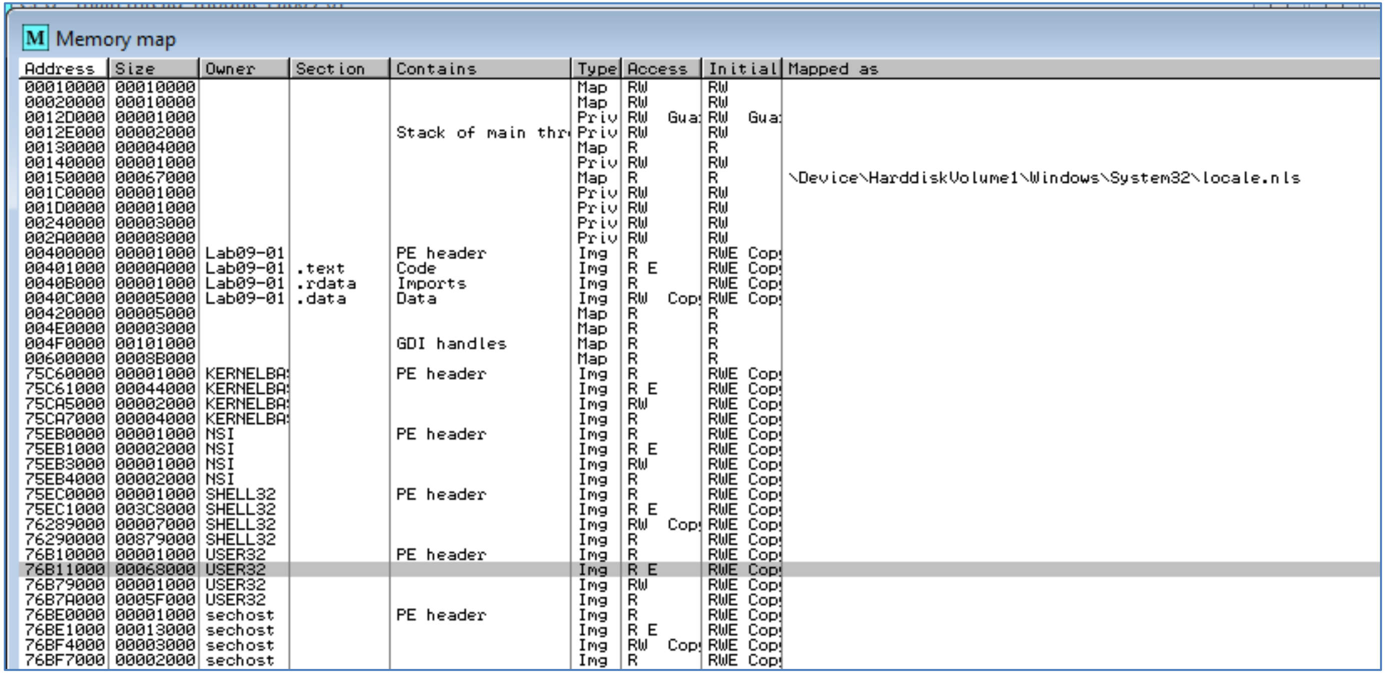
MEMORY MAP

View, Memory Map



Memory Map

- EXE and DLLs are identified
- **Double-click** any row to show a memory dump
- **Right-click**, View in Disassembler



Memory map

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00010000				Map	RW	RW	
00020000	00010000				Map	RW	RW	
0012D000	00001000				Priv	RW Gua	RW Gua	
0012E000	00002000			Stack of main thr	Priv	RW	RW	
00130000	00004000				Map	R	R	
00140000	00001000				Priv	RW	RW	
00150000	00067000				Map	R	R	\Device\HarddiskVolume1\Windows\System32\locale.nls
001C0000	00001000				Priv	RW	RW	
001D0000	00001000				Priv	RW	RW	
00240000	00003000				Priv	RW	RW	
002A0000	00008000				Priv	RW	RW	
00400000	00001000	Lab09-01		PE header	Img	R	RWE Cop	
00401000	0000A000	Lab09-01	.text	Code	Img	R E	RWE Cop	
0040B000	00001000	Lab09-01	.rdata	Imports	Img	R	RWE Cop	
0040C000	00005000	Lab09-01	.data	Data	Img	RW Cop	RWE Cop	
00420000	00005000				Map	R	R	
004E0000	00003000				Map	R	R	
004F0000	00101000			GDI handles	Map	R	R	
00600000	0008B000				Map	R	R	
75C60000	00001000	KERNELBA		PE header	Img	R	RWE Cop	
75C61000	00044000	KERNELBA			Img	R E	RWE Cop	
75CA5000	00002000	KERNELBA			Img	RW	RWE Cop	
75CA7000	00004000	KERNELBA			Img	R	RWE Cop	
75EB0000	00001000	NSI		PE header	Img	R	RWE Cop	
75EB1000	00002000	NSI			Img	R E	RWE Cop	
75EB3000	00001000	NSI			Img	RW	RWE Cop	
75EB4000	00002000	NSI			Img	R	RWE Cop	
75EC0000	00001000	SHELL32		PE header	Img	R	RWE Cop	
75EC1000	003C8000	SHELL32			Img	R E	RWE Cop	
76289000	00007000	SHELL32			Img	RW Cop	RWE Cop	
76290000	00879000	SHELL32			Img	R	RWE Cop	
76B10000	00001000	USER32		PE header	Img	R	RWE Cop	
76B11000	00068000	USER32			Img	R E	RWE Cop	
76B79000	00001000	USER32			Img	RW	RWE Cop	
76B7A000	0005F000	USER32			Img	R	RWE Cop	
76BE0000	00001000	sechost		PE header	Img	R	RWE Cop	
76BE1000	00013000	sechost			Img	R E	RWE Cop	
76BF4000	00003000	sechost			Img	RW Cop	RWE Cop	
76BF7000	00002000	sechost			Img	R	RWE Cop	

Rebasing

- Rebasing occurs when a module *is not loaded at its preferred base address*
- PE files have a preferred base address
  - The image base in the PE header
  - Usually, the file is loaded at that address
  - Most EXEs are designed to be loaded at 0x00400000
- EXEs that support *Address Space Layout Randomization* (ASLR) will often be relocated

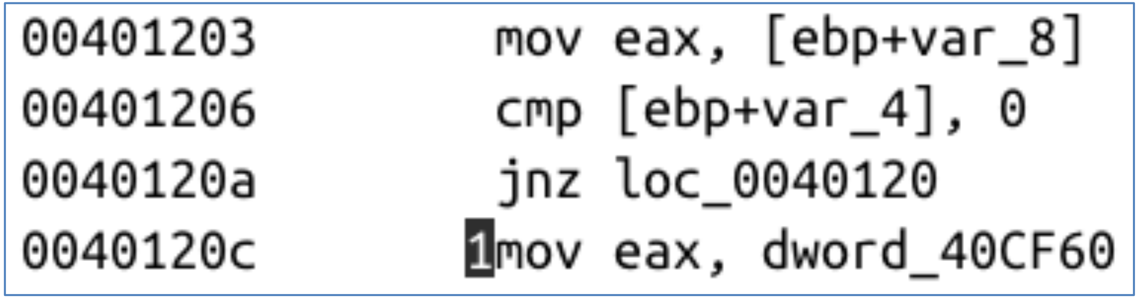
DLL Rebasing

- DLLs are more commonly relocated
  - Because a single application may import many DLLs
  - **Windows DLLs** have <u>different base addresses</u> to avoid this
  - **Third-party DLLs** often have the <u>same preferred base address</u>

Absolute vs. Relative Addresses

- The first 3 instructions will work fine if relocated because they use *<u>relative addresses</u>*

- The last one has an *<u>absolute address</u>* that will be wrong if the code is relocated



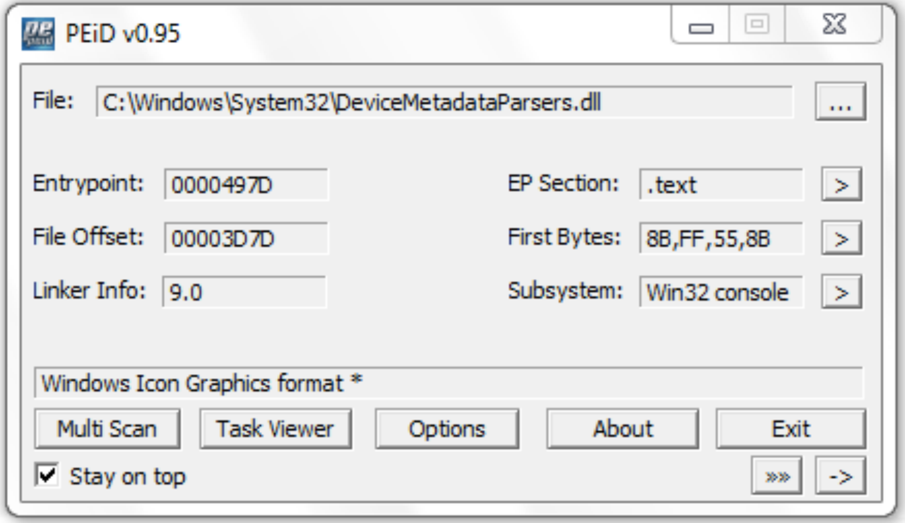
```
00401203        mov eax, [ebp+var_8]
00401206        cmp [ebp+var_4], 0
0040120a        jnz loc_0040120
0040120c       1mov eax, dword_40CF60
```

Fix-up Locations

- Most DLLS have a list of fix-up locations in the `.reloc` section of the PE header
  - These are instructions that must be changed when code is relocated

- DLLs are loaded after the EXE and in any order
- You cannot predict where DLLs will be located in memory if they are rebased
- Example `.reloc` section on next slide

Fix-up Locations (cont.)



PEiD v0.95

File: C:\Windows\System32\DeviceMetadataParsers.dll

Entrypoint: 0000497D

EP Section: .text

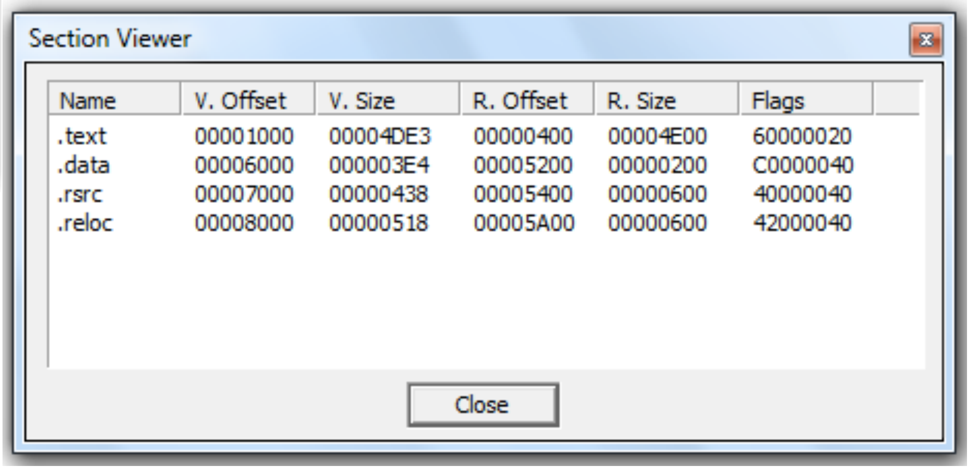
File Offset: 00003D7D

First Bytes: 8B,FF,55,8B

Linker Info: 9.0

Subsystem: Win32 console

Windows Icon Graphics format *



Section Viewer

Name	V. Offset	V. Size	R. Offset	R. Size	Flags
.text	00001000	00004DE3	00000400	00004E00	60000020
.data	00006000	000003E4	00005200	00000200	C0000040
.rsrc	00007000	00000438	00005400	00000600	40000040
.reloc	00008000	00000518	00005A00	00000600	42000040

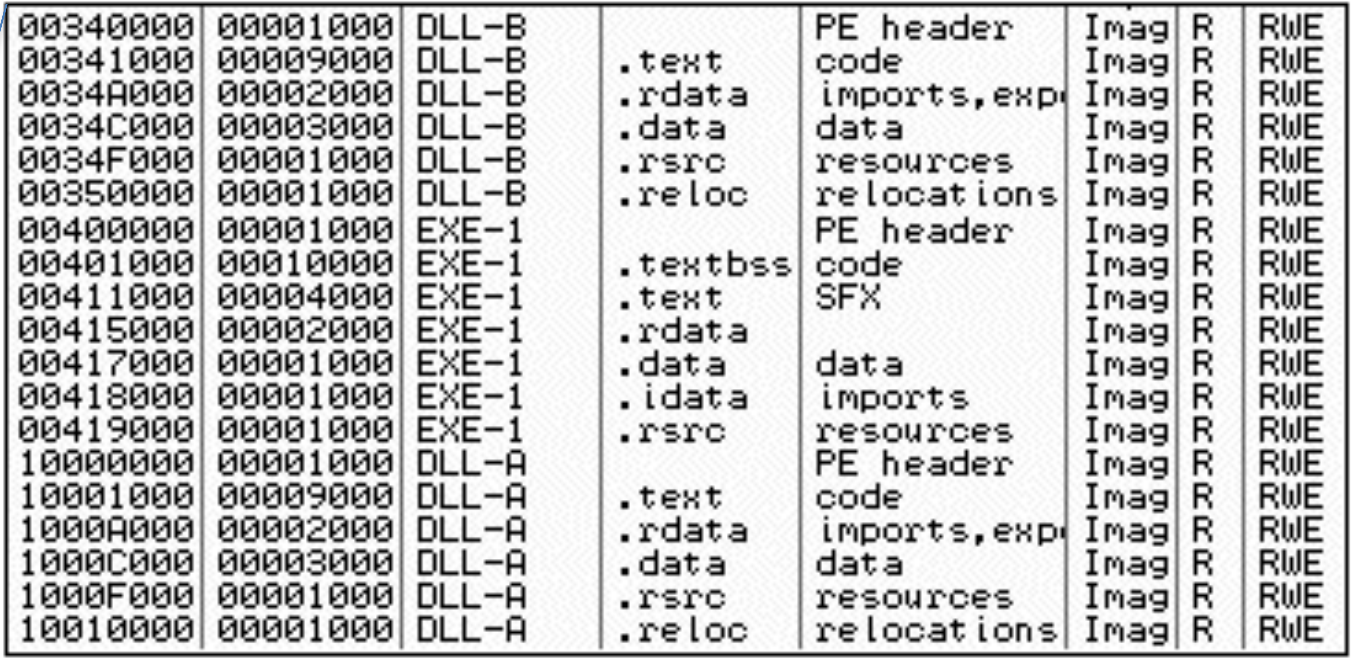
DLL Rebasing

- DLLS can have their `.reloc` removed
  - Such a DLL cannot be relocated
  - Must load at its preferred base address

- Relocating DLLs **is bad for performance**
  - Adds to load time
  - So good programmers specify non-default base addresses when compiling DLLs

Example of DLL Rebasing Olly Memory Map

- DLL-A and DLL-B prefer location **`0x10000000`**



```
00340000 00001000 DLL-B                PE header    Imag R   RWE
00341000 00009000 DLL-B    .text       code         Imag R   RWE
0034A000 00002000 DLL-B    .rdata      imports,exp  Imag R   RWE
0034C000 00003000 DLL-B    .data       data         Imag R   RWE
0034F000 00001000 DLL-B    .rsrc       resources    Imag R   RWE
00350000 00001000 DLL-B    .reloc      relocations  Imag R   RWE
00400000 00001000 EXE-1                PE header    Imag R   RWE
00401000 00010000 EXE-1    .textbss    code         Imag R   RWE
00411000 00004000 EXE-1    .text       SFX          Imag R   RWE
00415000 00002000 EXE-1    .rdata                   Imag R   RWE
00417000 00001000 EXE-1    .data       data         Imag R   RWE
00418000 00001000 EXE-1    .idata      imports      Imag R   RWE
00419000 00001000 EXE-1    .rsrc       resources    Imag R   RWE
10000000 00001000 DLL-A                PE header    Imag R   RWE
10001000 00009000 DLL-A    .text       code         Imag R   RWE
1000A000 00002000 DLL-A    .rdata      imports,exp  Imag R   RWE
1000C000 00003000 DLL-A    .data       data         Imag R   RWE
1000F000 00001000 DLL-A    .rsrc       resources    Imag R   RWE
10010000 00001000 DLL-A    .reloc      relocations  Imag R   RWE
```

DLL-B is relocated into a different memory address from its requested location

IDA Pro

- IDA Pro is not attached to a real running process

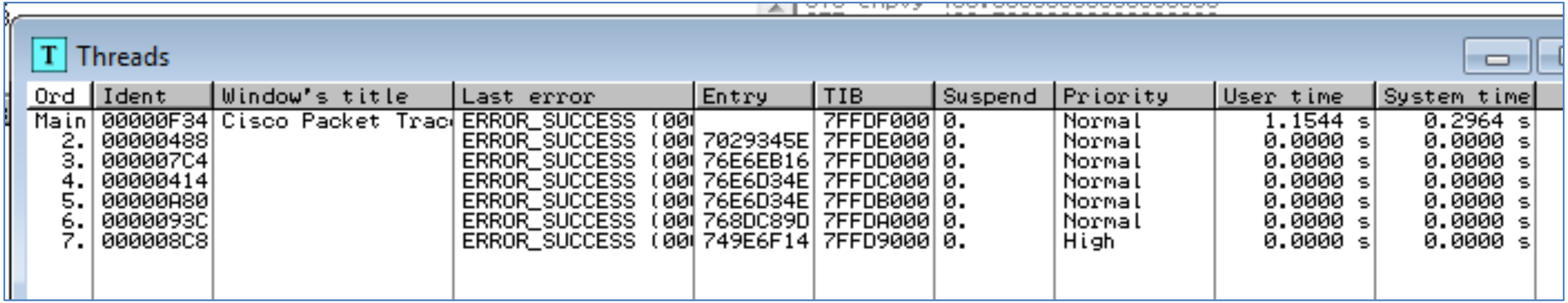
- It doesn't know about rebasing

- If you use OllyDbg and IDA Pro at the same time, you may get different results
  - To avoid this, use the "**Manual Load**" option in IDA Pro
  - Specify the virtual base address manually

Viewing Threads and Stacks

- View, Threads

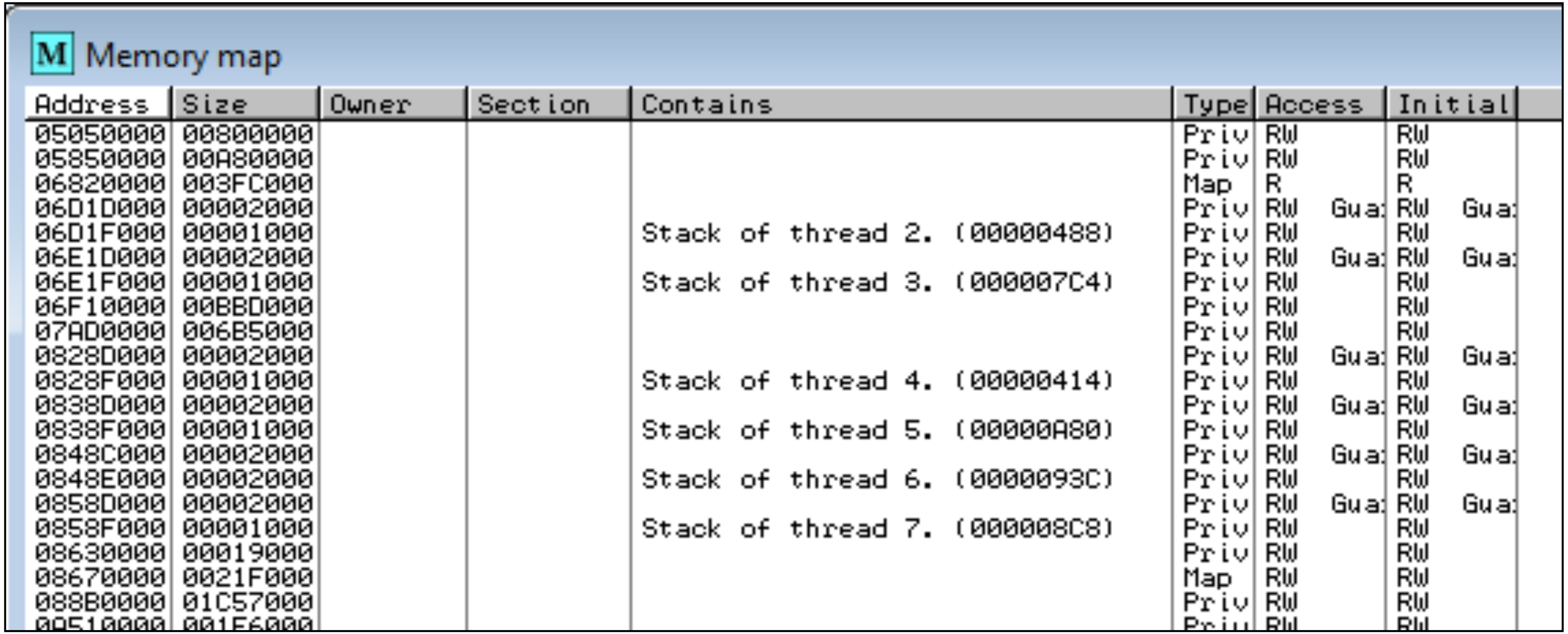
- Right-click a thread to “Open in CPU”, kill it, etc.



Ord	Ident	Window's title	Last error	Entry	TIB	Suspend	Priority	User time	System time
Main	00000F34	Cisco Packet Trac	ERROR_SUCCESS (00		7FFDF000	0.	Normal	1.1544 s	0.2964 s
2.	00000488		ERROR_SUCCESS (00	7029345E	7FFDE000	0.	Normal	0.0000 s	0.0000 s
3.	000007C4		ERROR_SUCCESS (00	76E6EB16	7FFDD000	0.	Normal	0.0000 s	0.0000 s
4.	00000414		ERROR_SUCCESS (00	76E6D34E	7FFDC000	0.	Normal	0.0000 s	0.0000 s
5.	00000A80		ERROR_SUCCESS (00	76E6D34E	7FFDB000	0.	Normal	0.0000 s	0.0000 s
6.	0000093C		ERROR_SUCCESS (00	768DC89D	7FFDA000	0.	Normal	0.0000 s	0.0000 s
7.	000008C8		ERROR_SUCCESS (00	749E6F14	7FFD9000	0.	High	0.0000 s	0.0000 s

Each Thread Has its Own Stack

- Visible in Memory Map



Address	Size	Owner	Section	Contains	Type	Access	Initial
05050000	00800000				Priv	RW	RW
05850000	00A80000				Priv	RW	RW
06820000	003FC000				Map	R	R
06D1D000	00002000				Priv	RW Gua:	RW Gua:
06D1F000	00001000			Stack of thread 2. (00000488)	Priv	RW	RW
06E1D000	00002000				Priv	RW Gua:	RW Gua:
06E1F000	00001000			Stack of thread 3. (000007C4)	Priv	RW	RW
06F10000	00BBD000				Priv	RW	RW
07AD0000	006B5000				Priv	RW	RW
0828D000	00002000				Priv	RW Gua:	RW Gua:
0828F000	00001000			Stack of thread 4. (00000414)	Priv	RW	RW
0838D000	00002000				Priv	RW Gua:	RW Gua:
0838F000	00001000			Stack of thread 5. (00000A80)	Priv	RW	RW
0848C000	00002000				Priv	RW Gua:	RW Gua:
0848E000	00002000			Stack of thread 6. (0000093C)	Priv	RW	RW
0858D000	00002000				Priv	RW Gua:	RW Gua:
0858F000	00001000			Stack of thread 7. (000008C8)	Priv	RW	RW
08630000	00019000				Priv	RW	RW
08670000	0021F000				Map	RW	RW
088B0000	01C57000				Priv	RW	RW
0A510000	001F6000				Priv	RW	RW

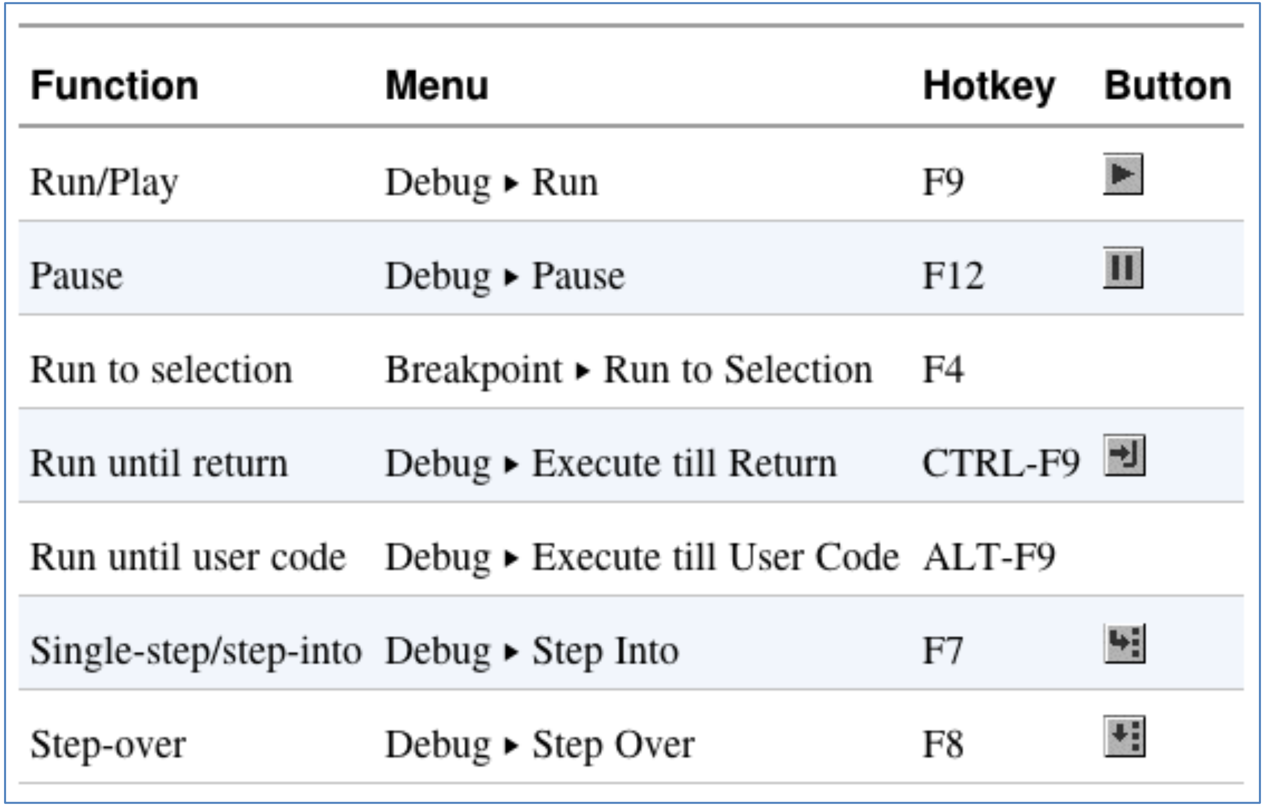
ASLR is Fading

- Address Space Layout Randomization
  - *“ASLR is fundamentally flawed in sandboxed environments such as JavaScript and future defenses **should not rely on** randomized virtual addresses as a building block.”*

- https://www.theregister.com/2021/02/26/chrome_aslr_bypass/

EXECUTING CODE

OllyDbg Code-Executing 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	↓

Run and Pause

- 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

Run and Run to Selection

- **Run** is useful to resume execution after hitting a breakpoint

- **Run to Selection** will execute until just before the selected instruction is executed
  - If the selection is never executed, it will run indefinitely

Execute till Return

- Pauses execution until just before the current function is set to return

- Can be useful if you want to finish the current function and stop

- But if the function never ends, the program will continue to run indefinitely

Execute till User Code

- 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

Stepping Through Code

- **F7**—Single-step (also called step-into)
- **F8**—Step-over
  - Stepping-over means all the code is executed, but you don't see it happen
- Some malware is designed to fool you, by calling routines and never returning, so stepping over will miss the most important part

BREAKPOINTS

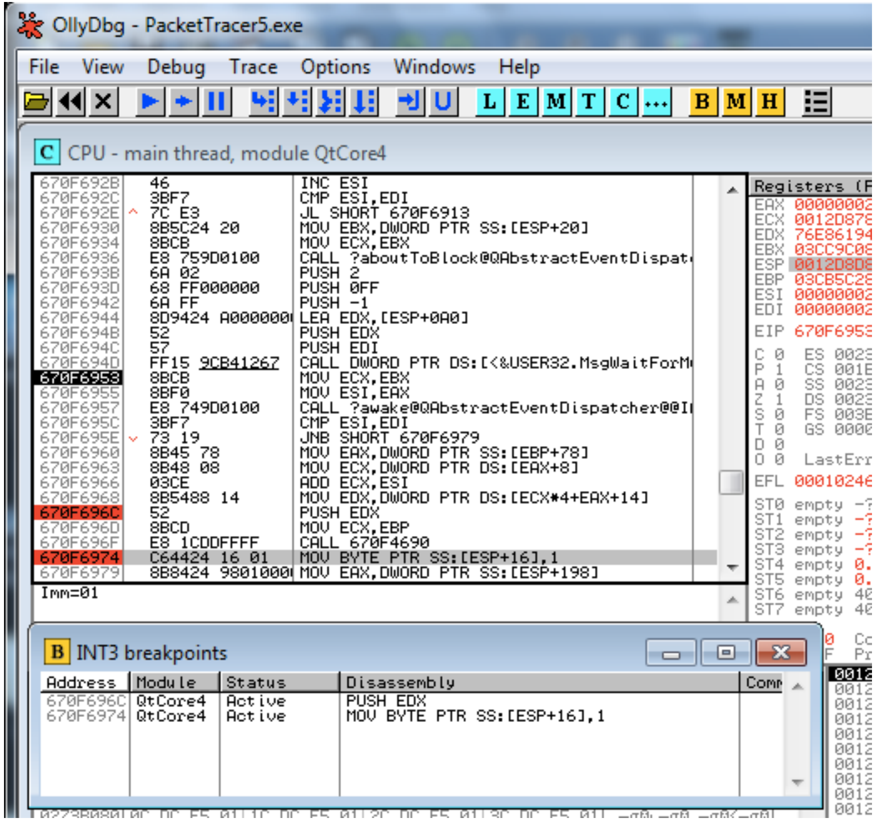
Types of Breakpoints

- **Software** breakpoints
- **Hardware** breakpoints
- **Conditional** breakpoints
- Breakpoints on **memory**

- **F2** – Add or remove a breakpoint

Viewing Active Breakpoints

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

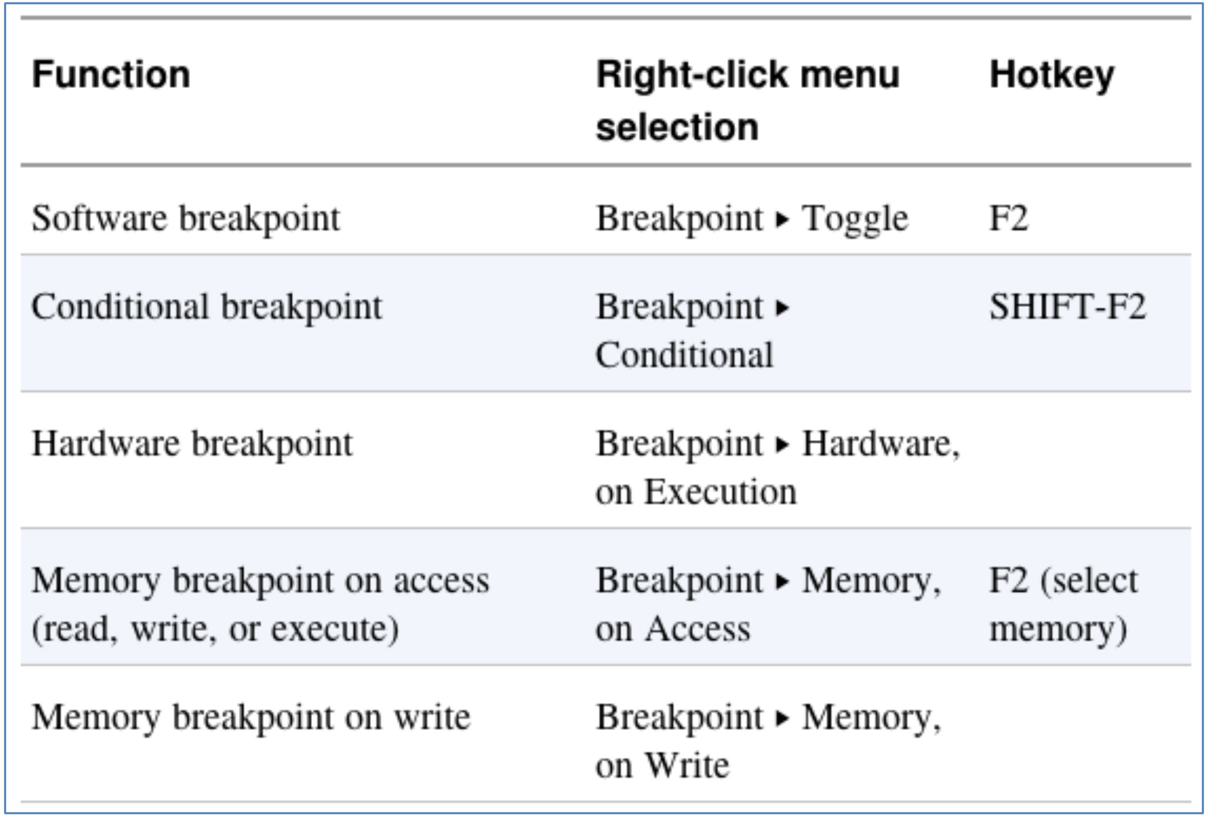

OllyDbg - PacketTracer5.exe
File
View
Debug
Trace
Options
Windows
Help
L
E
M
T
C
...
B
M
H
C
CPU - main thread, module QtCore4

```
670F692B    46              INC ESI
670F692C    3BF7            CMP ESI,EDI
670F692E  ^ 7C E3           JL SHORT 670F6913
670F6930    8B5C24 20       MOV EBX,DWORD PTR SS:[ESP+20]
670F6934    8BCB            MOV ECX,EBX
670F6936    E8 759D0100     CALL ?aboutToBlock@QAbstractEventDispat
670F693B    6A 02           PUSH 2
670F693D    68 FF000000     PUSH 0FF
670F6942    6A FF           PUSH -1
670F6944    8D9424 A000000  LEA EDX,[ESP+0A0]
670F694B    52              PUSH EDX
670F694C    57              PUSH EDI
670F694D    FF15 9CB41267   CALL DWORD PTR DS:[<&USER32.MsgWaitForM
670F6953    8BCB            MOV ECX,EBX
670F6955    8BF0            MOV ESI,EAX
670F6957    E8 749D0100     CALL ?awake@QAbstractEventDispatcher@@I
670F695C    3BF7            CMP ESI,EDI
670F695E  v 73 19           JNB SHORT 670F6979
670F6960    8B45 78         MOV EAX,DWORD PTR SS:[EBP+78]
670F6963    8B48 08         MOV ECX,DWORD PTR DS:[EAX+8]
670F6966    03CE            ADD ECX,ESI
670F6968    8B5488 14       MOV EDX,DWORD PTR DS:[ECX*4+EAX+14]
670F696C    52              PUSH EDX
670F696D    8BCD            MOV ECX,EBP
670F696F    E8 1CDDFFFF     CALL 670F4690
670F6974    C64424 16 01    MOV BYTE PTR SS:[ESP+16],1
670F6979    8B8424 9801000  MOV EAX,DWORD PTR SS:[ESP+198]
Imm=01
```

```
Registers (F
EAX 00000002
ECX 0012D878
EDX 76E86194
EBX 03CC9C08
ESP 0012D8D8
EBP 03CB5C28
ESI 00000002
EDI 00000002

EIP 670F6953

C 0  ES 0023
P 1  CS 001E
A 0  SS 0023
Z 1  DS 0023
S 0  FS 003E
T 0  GS 0000
D 0
O 0  LastErr

EFL 00010246

ST0 empty -?
ST1 empty -?
ST2 empty -?
ST3 empty -?
ST4 empty 0.
ST5 empty 0.
ST6 empty 40
ST7 empty 40
```

Address	Module	Status	Disassembly	Comm
670F696C	QtCore4	Active	PUSH EDX	
670F6974	QtCore4	Active	MOV BYTE PTR SS:[ESP+16],1	

OllyDbg Breakingpoint 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	

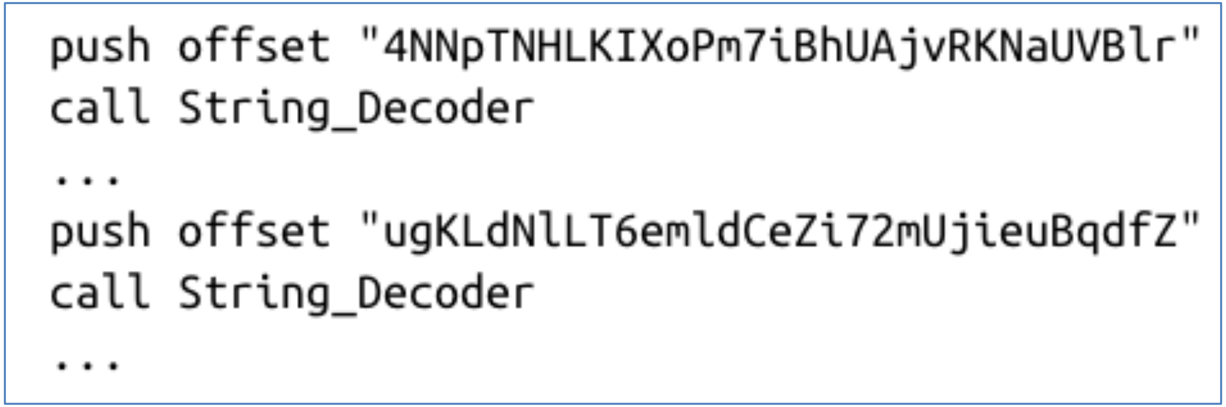
Saving Breakpoints

- When you close OllyDbg, it saves your breakpoints
- If you open the same file again, the breakpoints are still available

Software Breakpoints

- Useful for string decoders

- Malware authors often obfuscate strings
  - With a **string decoder** that is called before each string is used



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

String Decoders

- 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

Conditional Breakpoints

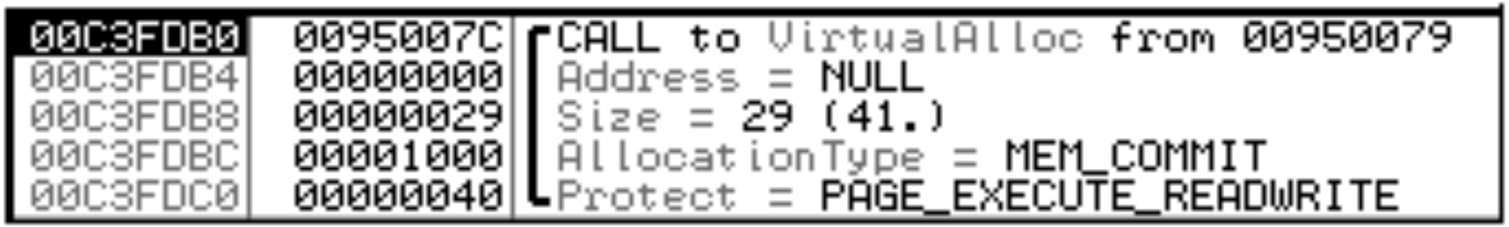
- Breaks only when a condition is true

- 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`**

Normal Breakpoint

- Put a standard breakpoint at the start of the **`VirtualAlloc`** function

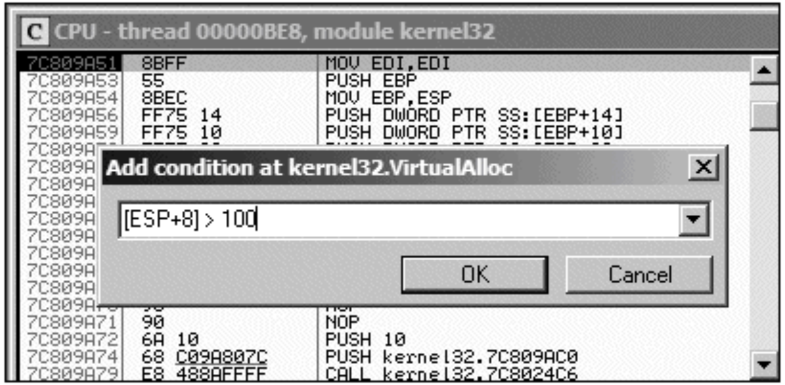
- Here's the stack when it hits, showing five items:
  - Return address
  - 4 parameters (Address, Size, AllocationType, Protect)



```
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
```

Conditional Breakpoint

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.



CPU - thread 00000BE8, module kernel32

```
7C809A51   8BFF          MOV EDI,EDI
7C809A53   55            PUSH EBP
7C809A54   8BEC          MOV EBP,ESP
7C809A56   FF75 14       PUSH DWORD PTR SS:[EBP+14]
7C809A59   FF75 10       PUSH DWORD PTR SS:[EBP+10]
7C809A
7C809A
7C809A
7C809A
7C809A
7C809A
7C809A
7C809A
7C809A
7C809A
7C809A71   90            NOP
7C809A72   6A 10         PUSH 10
7C809A74   68 C09A807C   PUSH kernel32.7C809AC0
7C809A79   E8 488AFFFF   CALL kernel32.7C8024C6
```

Add condition at kernel32.VirtualAlloc

[ESP+8] > 100

Hardware Breakpints

- Don't alter code, stack, or any target resource
- Don't slow down execution
- But you can only set 4 at a time
- Click **Breakpoint**, “**Hardware, on Execution**”
- You can set OllyDbg to use hardware breakpoints by default in Debugging Options
    - Useful if malware uses anti-debugging techniques

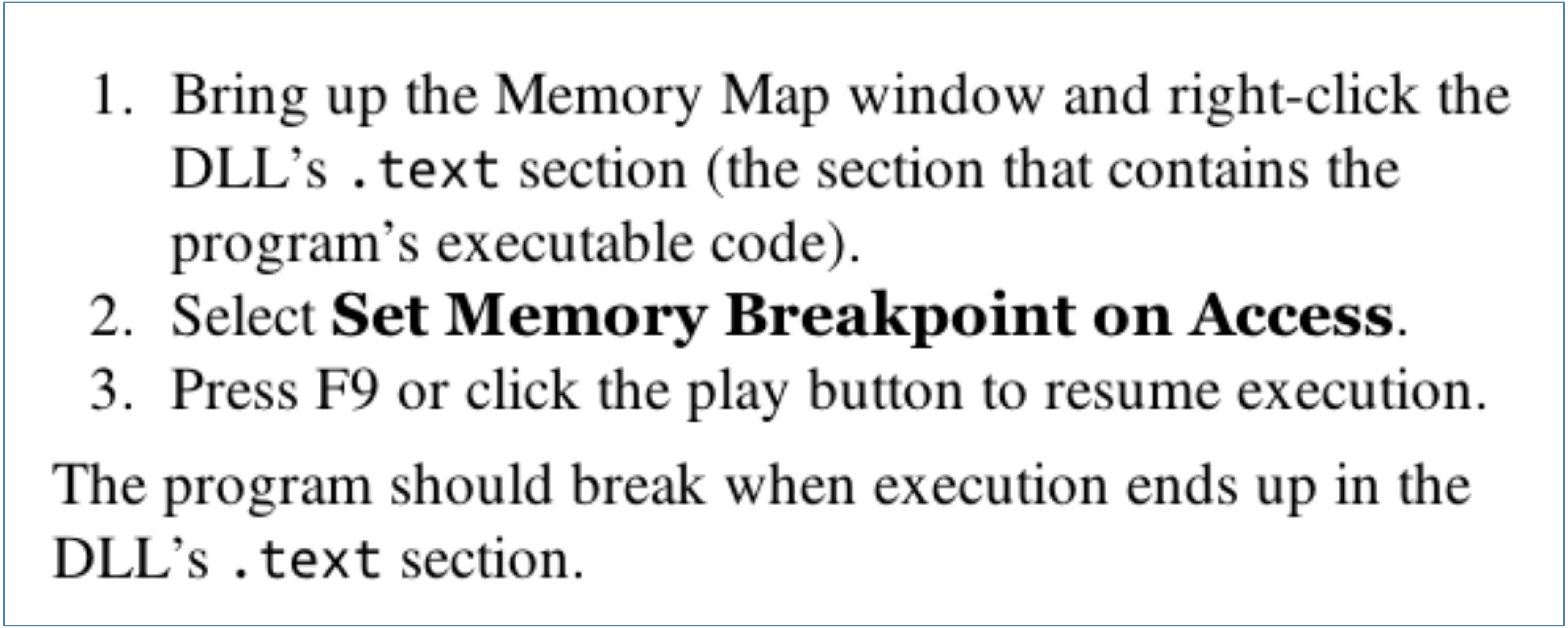
Memory Breakpoints

- Code breaks on access to specified memory location
- OllyDbg supports software and hardware memory breakpoints
- Can break on read, write, execute, or any access
- Right-click memory location, click **Breakpoint**, “**Memory, on Access**”

Memory Breakpoints (cont.)

- You can only set ***one*** memory breakpoint at a time
- OllyDbg implements memory breakpoints by changing the attributes of memory blocks
- This technique is not reliable and has considerable overhead
- Use memory breakpoints sparingly

When is a DLL Used?



1. Bring up the Memory Map window and right-click the DLL's `.text` section (the section that contains the program's executable code).
2. Select **Set Memory Breakpoint on Access**.
3. Press F9 or click the play button to resume execution.

The program should break when execution ends up in the DLL's `.text` section.

LOADING DLLS

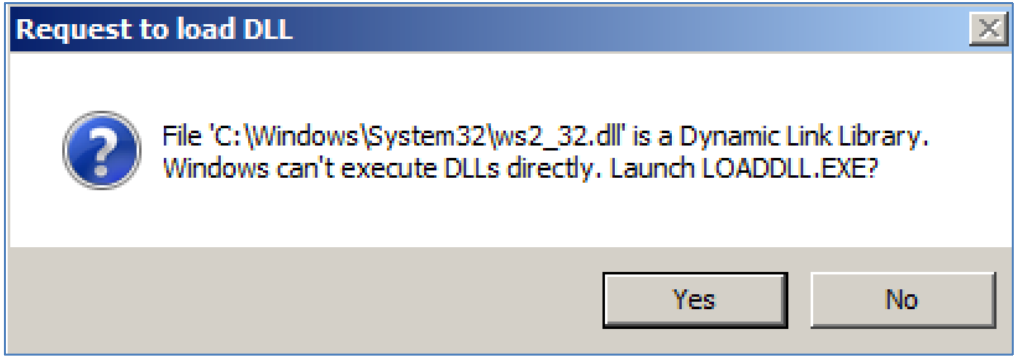
`loaddll.exe`

- DLLs cannot be executed directly
- OllyDbg uses a dummy **`loaddll.exe`** program to load them
- Breaks at the DLL entry point **DLLMain** once the DLL is loaded
- Press Play to run **DLLMain** and initialize the DLL for use

Demo

- Get OllyDbg 1.10, NOT 2.00 or 2.01
- Use Win 2016 Server, 64 bit
- In OllyDbg, open

  `C:\Windows\SysWOW64\ws2_32.dll`
- Click **Yes** at this box

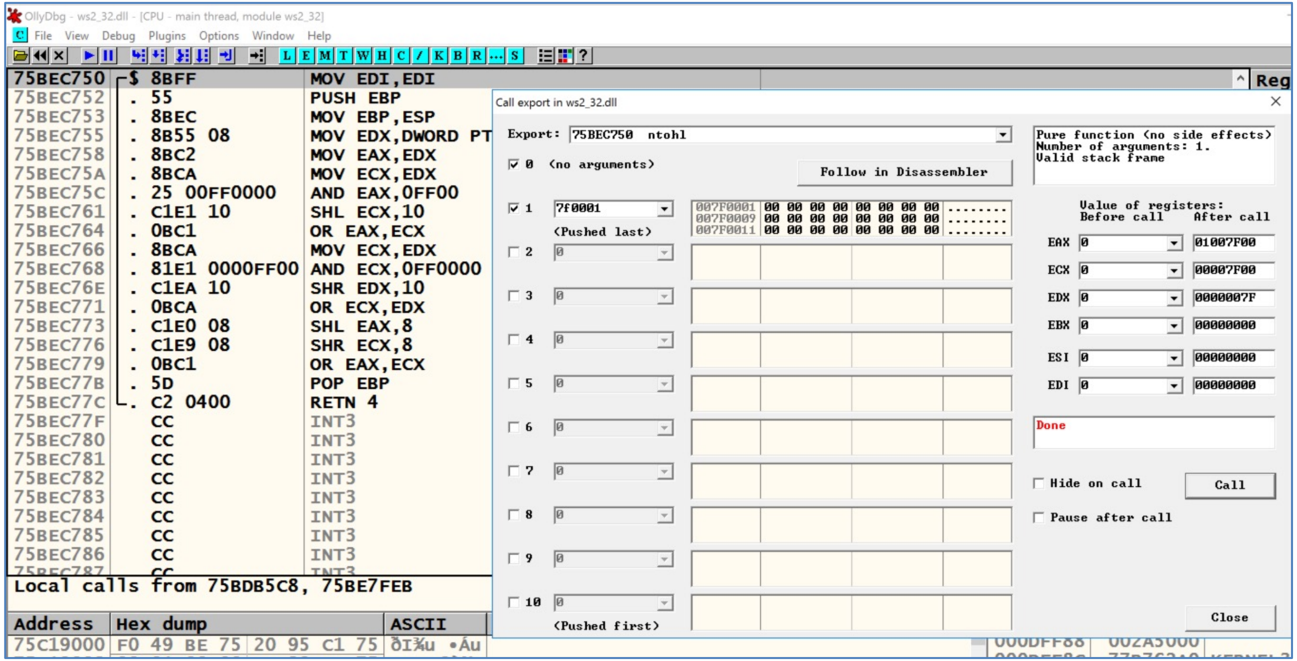


Request to load DLL

File 'C:\Windows\System32\ws2_32.dll' is a Dynamic Link Library.
Windows can't execute DLLs directly. Launch LOADDLL.EXE?

Demo: Calling DLL Exports

- Click **Debug, Call DLL Export** – it fails because DLLMain has not yet been run
- Reload the DLL (Ctrl+F2), click **Run** button once
- Click **Debug, Call DLL Export** – now it works


OllyDbg - ws2_32.dll - [CPU - main thread, module ws2_32]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
Reg

```
75BEC750 ┌$ 8BFF          MOV EDI,EDI
75BEC752 │. 55            PUSH EBP
75BEC753 │. 8BEC          MOV EBP,ESP
75BEC755 │. 8B55 08       MOV EDX,DWORD PT
75BEC758 │. 8BC2          MOV EAX,EDX
75BEC75A │. 8BCA          MOV ECX,EDX
75BEC75C │. 25 00FF0000   AND EAX,0FF00
75BEC761 │. C1E1 10       SHL ECX,10
75BEC764 │. 0BC1          OR EAX,ECX
75BEC766 │. 8BCA          MOV ECX,EDX
75BEC768 │. 81E1 0000FF00 AND ECX,0FF0000
75BEC76E │. C1EA 10       SHR EDX,10
75BEC771 │. 0BCA          OR ECX,EDX
75BEC773 │. C1E0 08       SHL EAX,8
75BEC776 │. C1E9 08       SHR ECX,8
75BEC779 │. 0BC1          OR EAX,ECX
75BEC77B │. 5D            POP EBP
75BEC77C └. C2 0400       RETN 4
75BEC77F    CC            INT3
75BEC780    CC            INT3
75BEC781    CC            INT3
75BEC782    CC            INT3
75BEC783    CC            INT3
75BEC784    CC            INT3
75BEC785    CC            INT3
75BEC786    CC            INT3
Local calls from 75BDB5C8, 75BE7FEB

Address   Hex dump                 ASCII
75C19000  F0 49 BE 75 20 95 C1 75  ðI¾u •Áu
```

Call export in ws2_32.dll

Export: 75BEC750 ntohl

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

☑ 0 (no arguments)

Follow in Disassembler

☑ 1 7f0001 (Pushed last)

```
007F0001 00 00 00 00 00 00 00 00 ........
007F0009 00 00 00 00 00 00 00 00 ........
007F0011 00 00 00 00 00 00 00 00 ........
```

☐ 2 0
☐ 3 0
☐ 4 0
☐ 5 0
☐ 6 0
☐ 7 0
☐ 8 0
☐ 9 0
☐ 10 0 (Pushed first)

Value of registers:

	Before call	After call
EAX	0	01007F00
ECX	0	00007F00
EDX	0	0000007F
EBX	0	00000000
ESI	0	00000000
EDI	0	00000000

Done

☐ Hide on call

Call

☐ Pause after call

Close

Demo: Running `ntohl`

- Converts a 32-bit number from network to host byte order
- Click argument 1, type in **`7f000001`**
  - 127.0.0.1 in “network” byte order
- Click “Follow in Disassembler” to see the code
- Click “Call” to run the function
- Answer in EAX

TRACING

Tracing

- Powerful debugging technique
- Records detailed execution information
- Types of Tracing
  - Standard Back Trace
  - Call Stack Trace
  - Run Trace

Tracing: *Standard Back Trace*

- You move through the disassembler with the **Step Into** and **Step Over** buttons

- OllyDbg is recording your movement

- Use minus (-) key on keyboard to see previous instructions
  - But you won't see previous register values

- Plus (+) key takes you forward
  - If you used **Step Over**, you cannot go back and decide to **Step Into**

Tracing: *Call Stack Trace*

- Views the execution path to a given function

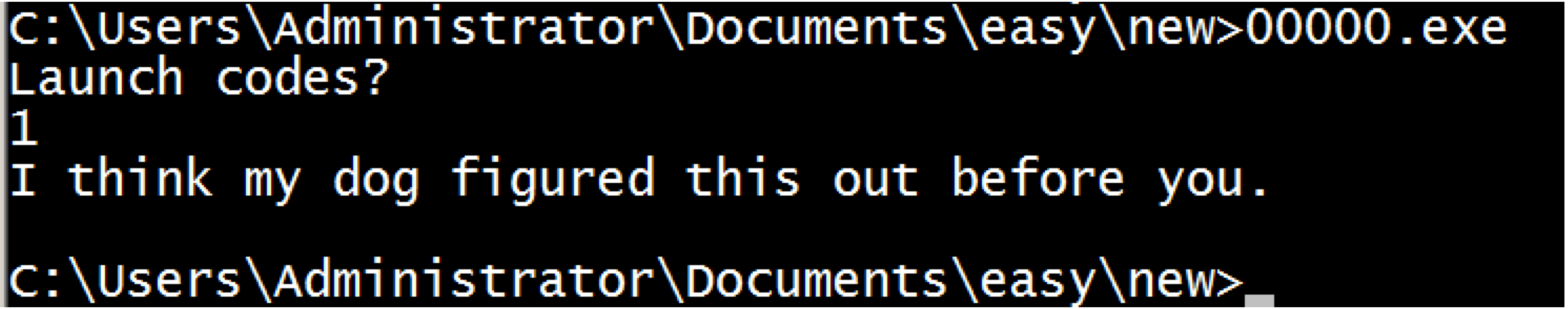
- Click **View**, **Call Stack**

- Displays the sequence of calls to reach your current location

Demo from EasyCTF 2017

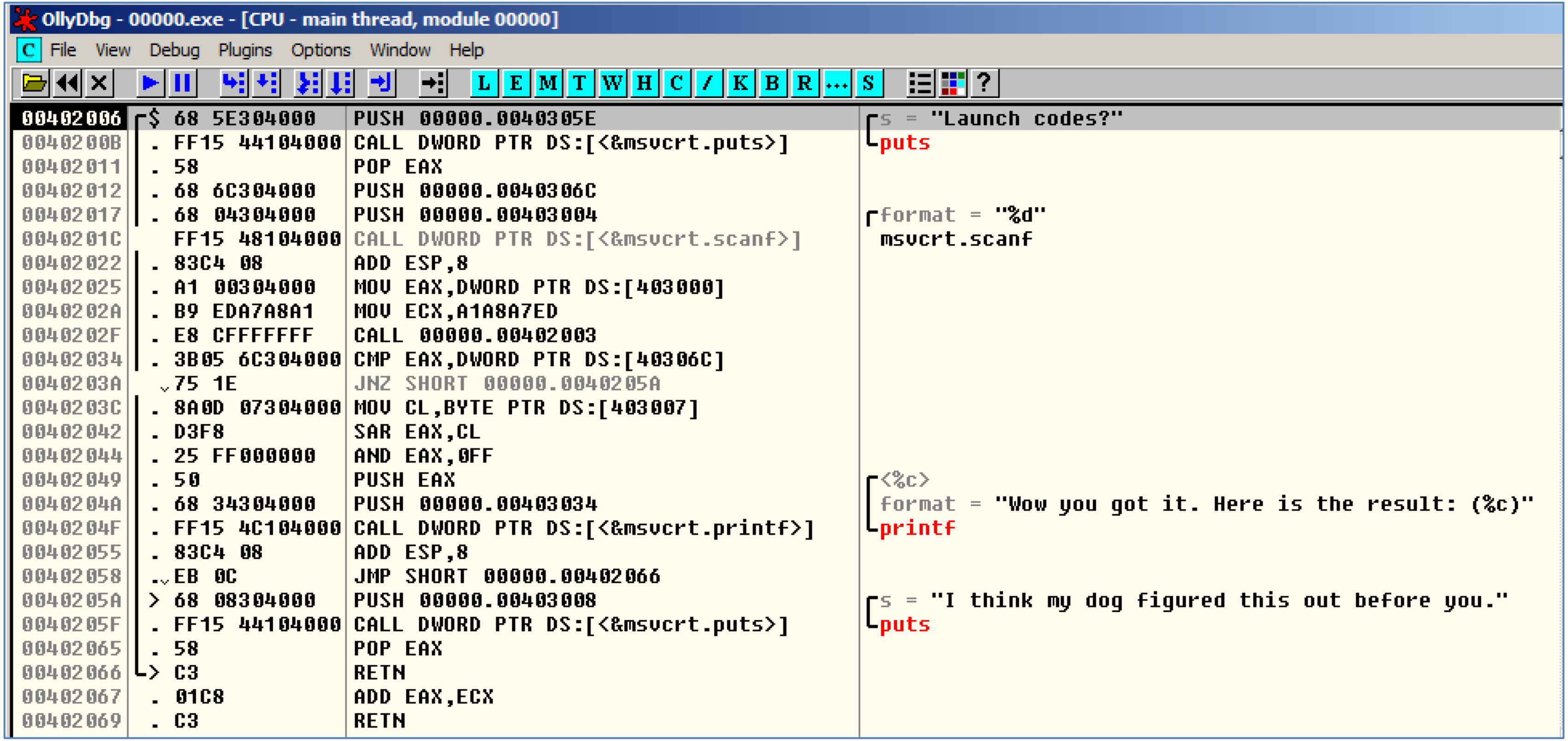
- Simple guessing game

- Wrong answer produces an insult



```
C:\Users\Administrator\Documents\easy\new>00000.exe
Launch codes?
1
I think my dog figured this out before you.

C:\Users\Administrator\Documents\easy\new>
```

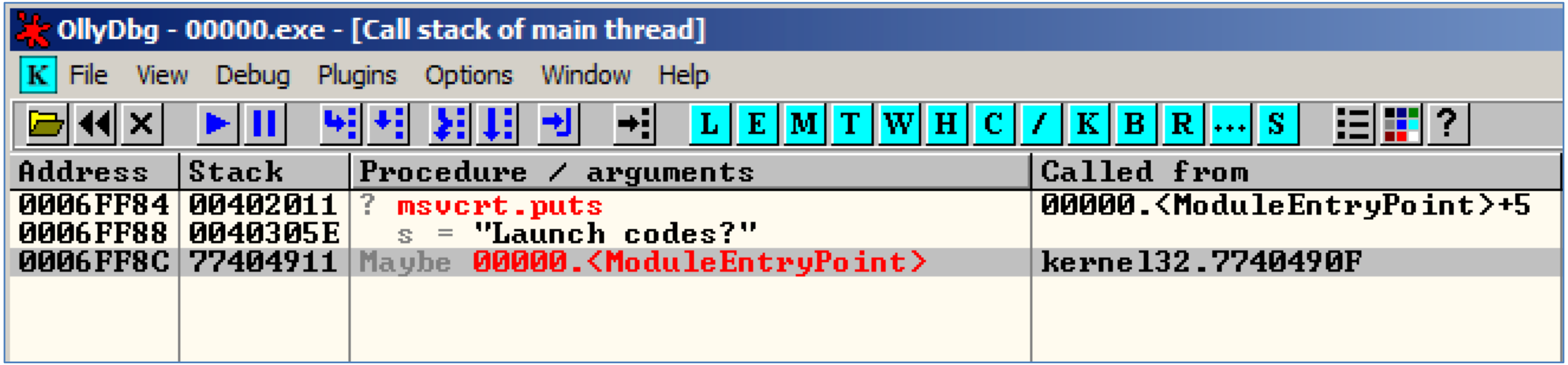
Entire main() in OllyDbg


OllyDbg - 00000.exe - [CPU - main thread, module 00000]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?

```
00402006  ┌$ 68 5E304000    PUSH 00000.0040305E                      ┌s = "Launch codes?"
0040200B  │. FF15 44104000  CALL DWORD PTR DS:[<&msvcrt.puts>]       └puts
00402011  │. 58             POP EAX
00402012  │. 68 6C304000    PUSH 00000.0040306C
00402017  │. 68 04304000    PUSH 00000.00403004                      ┌format = "%d"
0040201C     FF15 48104000  CALL DWORD PTR DS:[<&msvcrt.scanf>]       msvcrt.scanf
00402022  │. 83C4 08        ADD ESP,8
00402025  │. A1 00304000    MOV EAX,DWORD PTR DS:[403000]
0040202A  │. B9 EDA7A8A1    MOV ECX,A1A8A7ED
0040202F  │. E8 CFFFFFFF    CALL 00000.00402003
00402034  │. 3B05 6C304000  CMP EAX,DWORD PTR DS:[40306C]
0040203A   ⌄75 1E           JNZ SHORT 00000.0040205A
0040203C  │. 8A0D 07304000  MOV CL,BYTE PTR DS:[403007]
00402042  │. D3F8           SAR EAX,CL
00402044  │. 25 FF000000    AND EAX,0FF
00402049  │. 50             PUSH EAX                                 ┌<%c>
0040204A  │. 68 34304000    PUSH 00000.00403034                      │format = "Wow you got it. Here is the result: (%c)"
0040204F  │. FF15 4C104000  CALL DWORD PTR DS:[<&msvcrt.printf>]     └printf
00402055  │. 83C4 08        ADD ESP,8
00402058  │.⌄EB 0C          JMP SHORT 00000.00402066
0040205A  │> 68 08304000    PUSH 00000.00403008                      ┌s = "I think my dog figured this out before you."
0040205F  │. FF15 44104000  CALL DWORD PTR DS:[<&msvcrt.puts>]       └puts
00402065  │. 58             POP EAX
00402066  └> C3             RETN
00402067   . 01C8           ADD EAX,ECX
00402069   . C3             RETN
```

Step into puts

- Press **F7** twice
- Click **View, Call Stack**

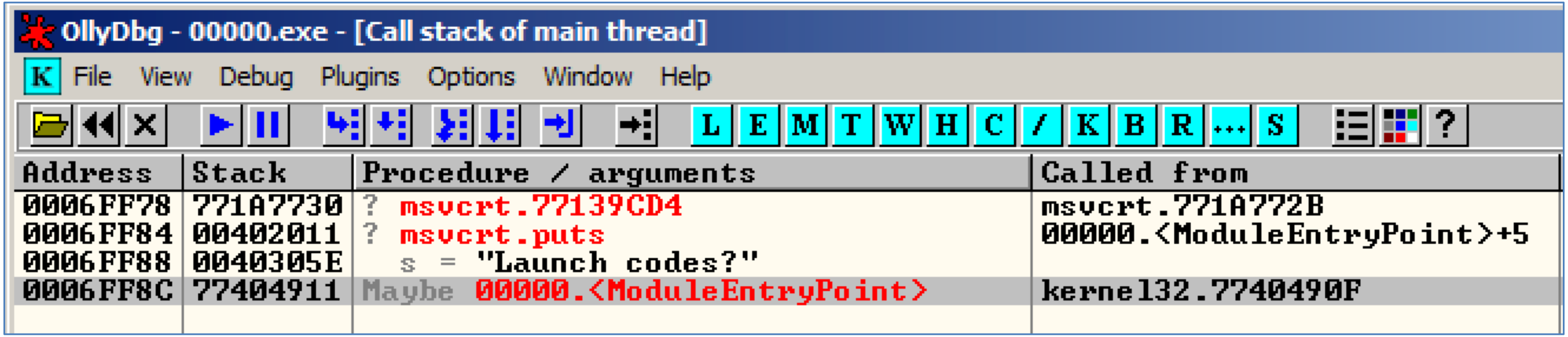


OllyDbg - 00000.exe - [Call stack of main thread]
K File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?

Address	Stack	Procedure / arguments	Called from
0006FF84	00402011	? msvcrt.puts	00000.<ModuleEntryPoint>+5
0006FF88	0040305E	s = "Launch codes?"	
0006FF8C	77404911	Maybe 00000.<ModuleEntryPoint>	kernel32.7740490F

Step Into Again

- Click **View**, **CPU**
- Press **F7** three times
- Click **View**, **Call Stack**
- New function appears at top

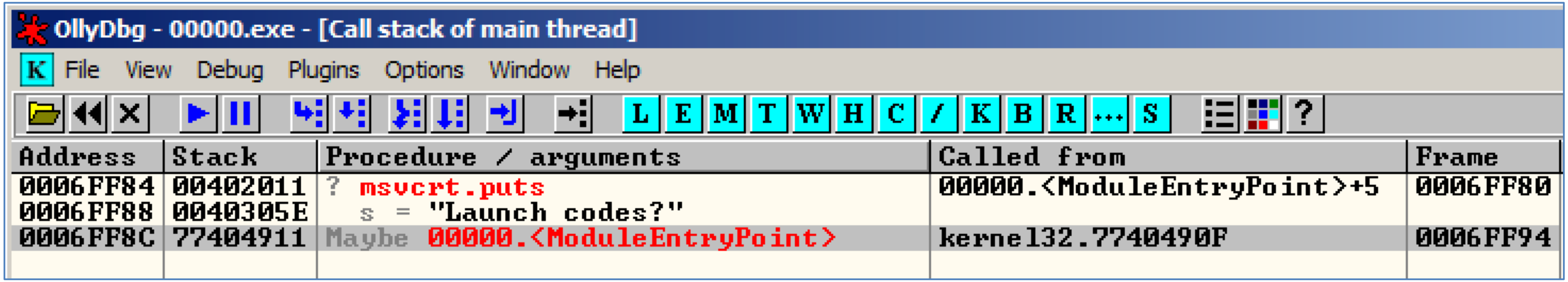


OllyDbg - 00000.exe - [Call stack of main thread]
K File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?

Address	Stack	Procedure / arguments	Called from
0006FF78	771A7730	? msvcrt.77139CD4	msvcrt.771A772B
0006FF84	00402011	? msvcrt.puts	00000.<ModuleEntryPoint>+5
0006FF88	0040305E	s = "Launch codes?"	
0006FF8C	77404911	Maybe 00000.<ModuleEntryPoint>	kernel32.7740490F

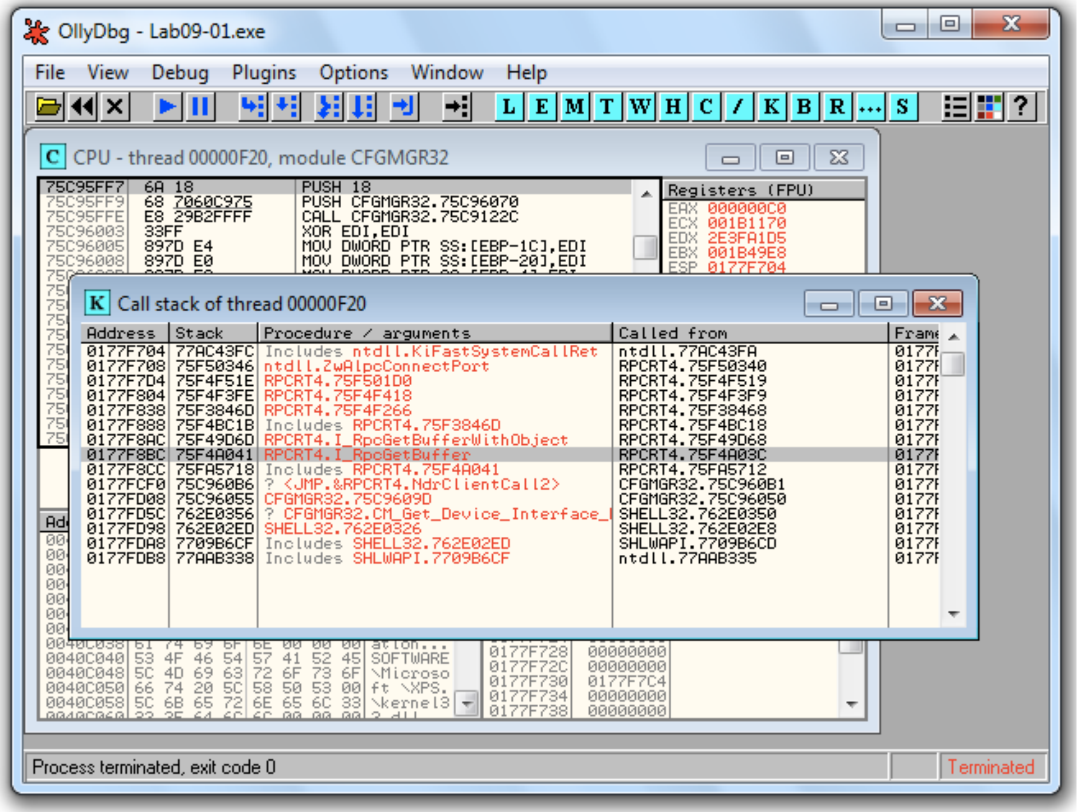
Return

- Click **View**, **CPU**
- Press **F7** 22 times, until the RETN and execute it
- Click **View**, **Call Stack**


OllyDbg - 00000.exe - [Call stack of main thread]
K File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?

Address	Stack	Procedure / arguments	Called from	Frame
0006FF84	00402011	? msvcrt.puts	00000.<ModuleEntryPoint>+5	0006FF80
0006FF88	0040305E	s = "Launch codes?"		
0006FF8C	77404911	Maybe 00000.<ModuleEntryPoint>	kernel32.7740490F	0006FF94

A Deeper Call Stack



```
75C95FF7   6A 18          PUSH 18
75C95FF9   68 7060C975    PUSH CFGMGR32.75C96070
75C95FFE   E8 29B2FFFF    CALL CFGMGR32.75C9122C
75C96003   33FF           XOR EDI,EDI
75C96005   897D E4        MOV DWORD PTR SS:[EBP-1C],EDI
75C96008   897D E0        MOV DWORD PTR SS:[EBP-20],EDI
```

```
Registers (FPU)
EAX 000000C0
ECX 001B1170
EDX 2E3FA1D5
EBX 001B49E8
ESP 0177F704
```

```
Address  Stack    Procedure / arguments                    Called from               Frame
0177F704 77AC43FC Includes ntdll.KiFastSystemCallRet        ntdll.77AC43FA            0177F
0177F708 75F50346 ntdll.ZwAlpcConnectPort                   RPCRT4.75F50340           0177F
0177F7D4 75F4F51E RPCRT4.75F501D0                           RPCRT4.75F4F519           0177F
0177F804 75F4F3FE RPCRT4.75F4F418                           RPCRT4.75F4F3F9           0177F
0177F838 75F3846D RPCRT4.75F4F266                           RPCRT4.75F38468           0177F
0177F888 75F4BC1B Includes RPCRT4.75F3846D                  RPCRT4.75F4BC18           0177F
0177F8AC 75F49D6D RPCRT4.I_RpcGetBufferWithObject           RPCRT4.75F49D68           0177F
0177F8BC 75F4A041 RPCRT4.I_RpcGetBuffer                     RPCRT4.75F4A03C           0177F
0177F8CC 75FA5718 Includes RPCRT4.75F4A041                  RPCRT4.75FA5712           0177F
0177FCF0 75C960B6 ? <JMP.&RPCRT4.NdrClientCall2>            CFGMGR32.75C960B1         0177F
0177FD08 75C96055 CFGMGR32.75C9609D                         CFGMGR32.75C96050         0177F
0177FD5C 762E0356 ? CFGMGR32.CM_Get_Device_Interface_I      SHELL32.762E0350          0177F
0177FD98 762E02ED SHELL32.762E0326                          SHELL32.762E02E8          0177F
0177FDA8 7709B6CF Includes SHELL32.762E02ED                 SHLWAPI.7709B6CD          0177F
0177FDB8 77AAB338 Includes SHLWAPI.7709B6CF                 ntdll.77AAB335            0177F
```

```
0040C038 61 74 69 6F 6E 00 00 00 ation...
0040C040 53 4F 46 54 57 41 52 45 SOFTWARE
0040C048 5C 4D 69 63 72 6F 73 6F \Microso
0040C050 66 74 20 5C 58 50 53 00 ft \XPS.
0040C058 5C 6B 65 72 6E 65 6C 33 \kernel3
```

```
0177F728 00000000
0177F72C 00000000
0177F730 0177F7C4
0177F734 00000000
0177F738 00000000
```

Process terminated, exit code 0

Tracing: *Run Trace*

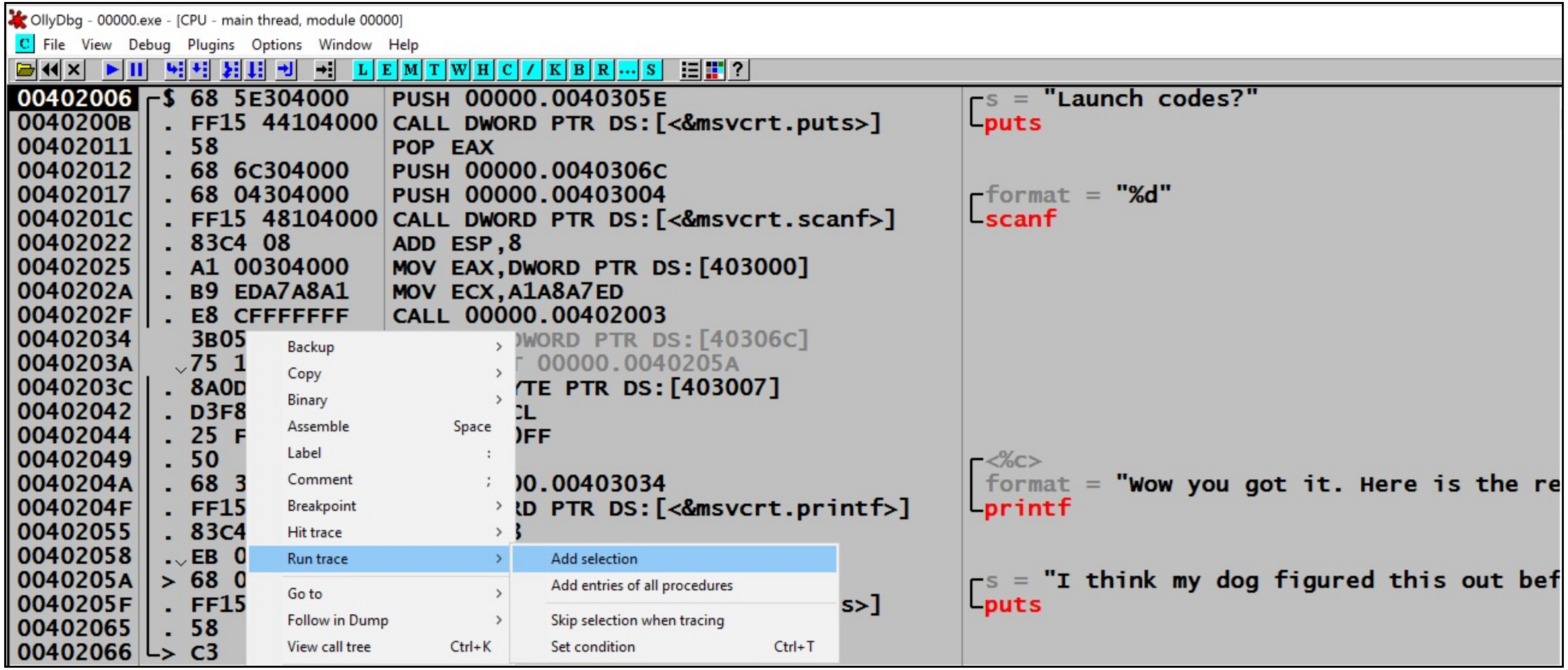
- Code runs, and OllyDbg saves every executed instruction and all changes to registers and flags

- Highlight code, right-click, **Run Trace**, **Add Selection**

- After code executes, **View**, **Run Trace**
  - To see instructions that were executed
  - + and - keys to step forward and backwards

Demo: Run Trace of 00000.exe

- Highlight code, right-click, **Run Trace, Add Selection**

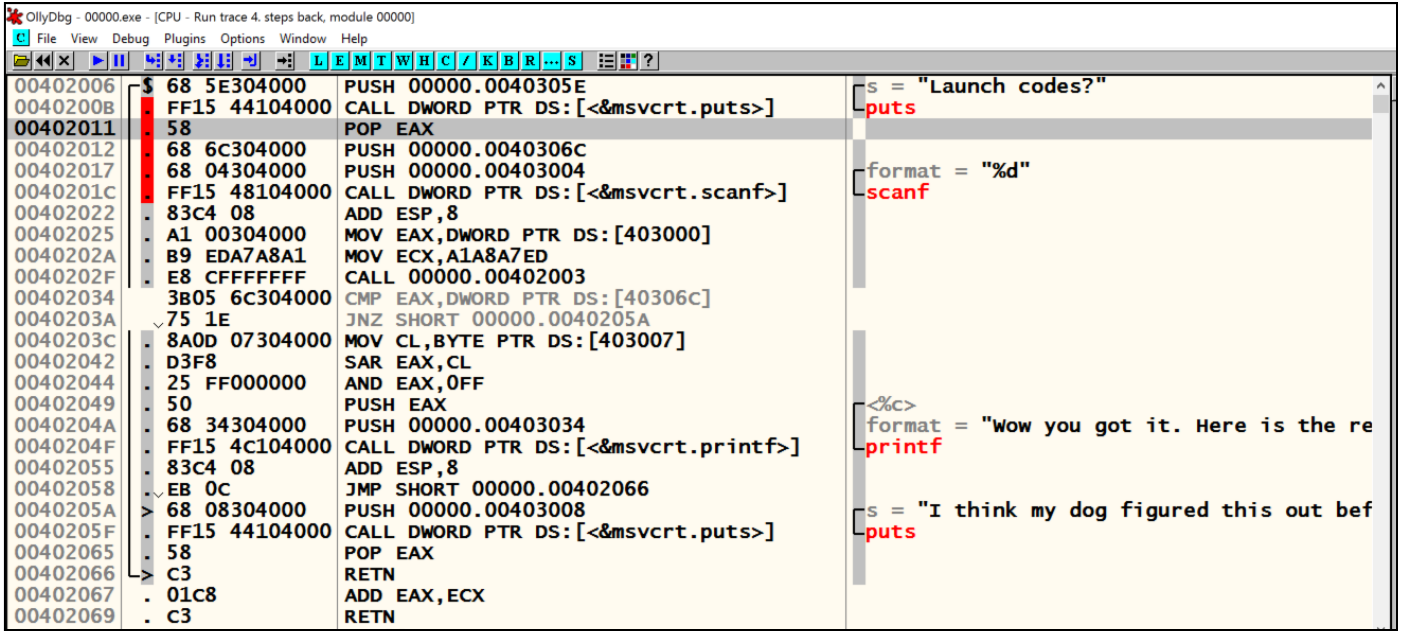


OllyDbg - 00000.exe - [CPU - main thread, module 00000]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
Backup
Copy
Binary
Assemble Space
Label :
Comment ;
Breakpoint
Hit trace
Run trace
Go to
Follow in Dump
View call tree Ctrl+K
Add selection
Add entries of all procedures
Skip selection when tracing
Set condition Ctrl+T

```
00402006 ┌$ 68 5E304000   PUSH 00000.0040305E                     ┌s = "Launch codes?"
0040200B │. FF15 44104000 CALL DWORD PTR DS:[<&msvcrt.puts>]      └puts
00402011 │. 58            POP EAX
00402012 │. 68 6C304000   PUSH 00000.0040306C
00402017 │. 68 04304000   PUSH 00000.00403004                     ┌format = "%d"
0040201C │. FF15 48104000 CALL DWORD PTR DS:[<&msvcrt.scanf>]     └scanf
00402022 │. 83C4 08       ADD ESP,8
00402025 │. A1 00304000   MOV EAX,DWORD PTR DS:[403000]
0040202A │. B9 EDA7A8A1   MOV ECX,A1A8A7ED
0040202F │. E8 CFFFFFFF   CALL 00000.00402003
00402034    3B05          DWORD PTR DS:[40306C]
0040203A   ⌄75 1          T 00000.0040205A
0040203C │. 8A0D          TE PTR DS:[403007]
00402042 │. D3F8          CL
00402044 │. 25 F          FF
00402049 │. 50                                                    ┌<%c>
0040204A │. 68 3          0.00403034                              │format = "Wow you got it. Here is the re
0040204F │. FF15          RD PTR DS:[<&msvcrt.printf>]            └printf
00402055 │. 83C4          3
00402058 .⌄EB 0
0040205A  > 68 0                                                  ┌s = "I think my dog figured this out bef
0040205F │. FF15          s>]                                     └puts
00402065 │. 58
00402066 └> C3
```

Demo: Run Trace of `00000.exe` (cont.)

- Run code
- Step back with - and forward with +



OllyDbg - 00000.exe - [CPU - Run trace 4. steps back, module 00000]

C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?

```
00402006  ┌$ 68 5E304000   PUSH 00000.0040305E                    ┌s = "Launch codes?"
0040200B  │. FF15 44104000 CALL DWORD PTR DS:[<&msvcrt.puts>]     └puts
00402011  │. 58            POP EAX
00402012  │. 68 6C304000   PUSH 00000.0040306C
00402017  │. 68 04304000   PUSH 00000.00403004                    ┌format = "%d"
0040201C  │. FF15 48104000 CALL DWORD PTR DS:[<&msvcrt.scanf>]    └scanf
00402022  │. 83C4 08       ADD ESP,8
00402025  │. A1 00304000   MOV EAX,DWORD PTR DS:[403000]
0040202A  │. B9 EDA7A8A1   MOV ECX,A1A8A7ED
0040202F  │. E8 CFFFFFFF   CALL 00000.00402003
00402034     3B05 6C304000 CMP EAX,DWORD PTR DS:[40306C]
0040203A    ⌄75 1E         JNZ SHORT 00000.0040205A
0040203C  │. 8A0D 07304000 MOV CL,BYTE PTR DS:[403007]
00402042  │. D3F8          SAR EAX,CL
00402044  │. 25 FF000000   AND EAX,0FF
00402049  │. 50            PUSH EAX                               ┌<%c>
0040204A  │. 68 34304000   PUSH 00000.00403034                    │format = "Wow you got it. Here is the re
0040204F  │. FF15 4C104000 CALL DWORD PTR DS:[<&msvcrt.printf>]   └printf
00402055  │. 83C4 08       ADD ESP,8
00402058  │.⌄EB 0C         JMP SHORT 00000.00402066
0040205A  │> 68 08304000   PUSH 00000.00403008                    ┌s = "I think my dog figured this out bef
0040205F  │. FF15 44104000 CALL DWORD PTR DS:[<&msvcrt.puts>]     └puts
00402065  │. 58            POP EAX
00402066  └> C3            RETN
00402067   . 01C8          ADD EAX,ECX
00402069   . C3            RETN
```

Trace Into and Trace Over

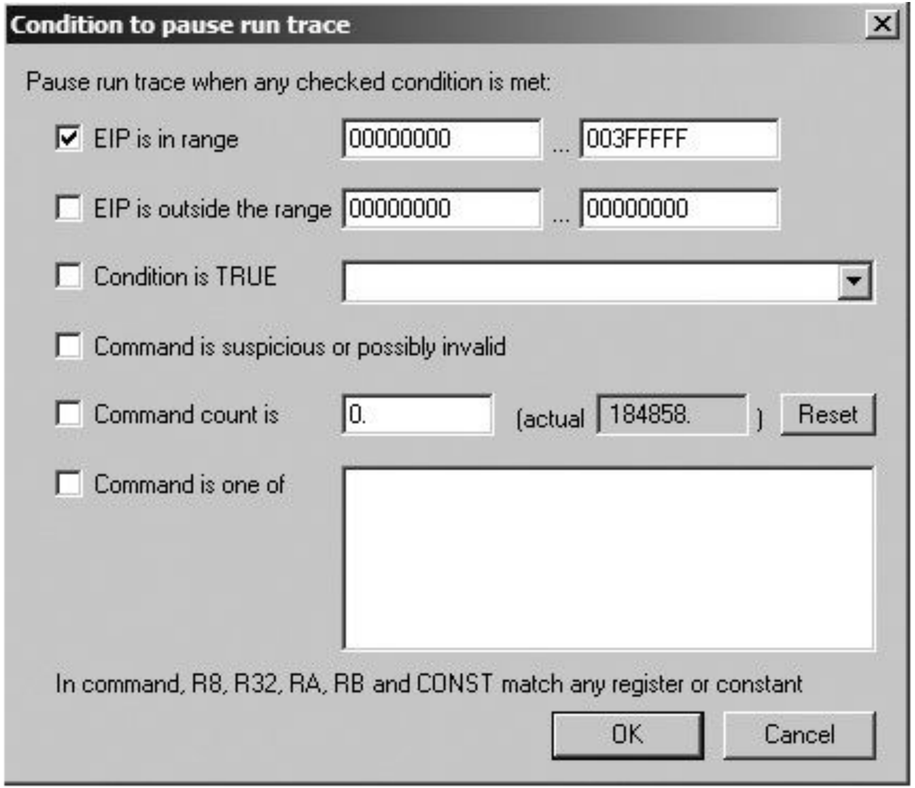
- Buttons below “Options”

- Easier to use than Add Selection

- If you don't set breakpoints, OllyDbg will attempt to *trace the entire program*, which could take a long time and a lot of memory

Debug, Set Condition

- Traces until a condition hits
- This condition catches **Poison Ivy** shellcode, which places code in dynamically allocated memory below `0x400000`



Condition to pause run trace

Pause run trace when any checked condition is met:

☑ EIP is in range 00000000 ... 003FFFFF

□ EIP is outside the range 00000000 ... 00000000

□ Condition is TRUE

□ Command is suspicious or possibly invalid

□ Command count is 0. (actual 184858.)

□ Command is one of

In command, R8, R32, RA, RB and CONST match any register or constant

EXCEPTION HANDLING

When an Exception Occurs

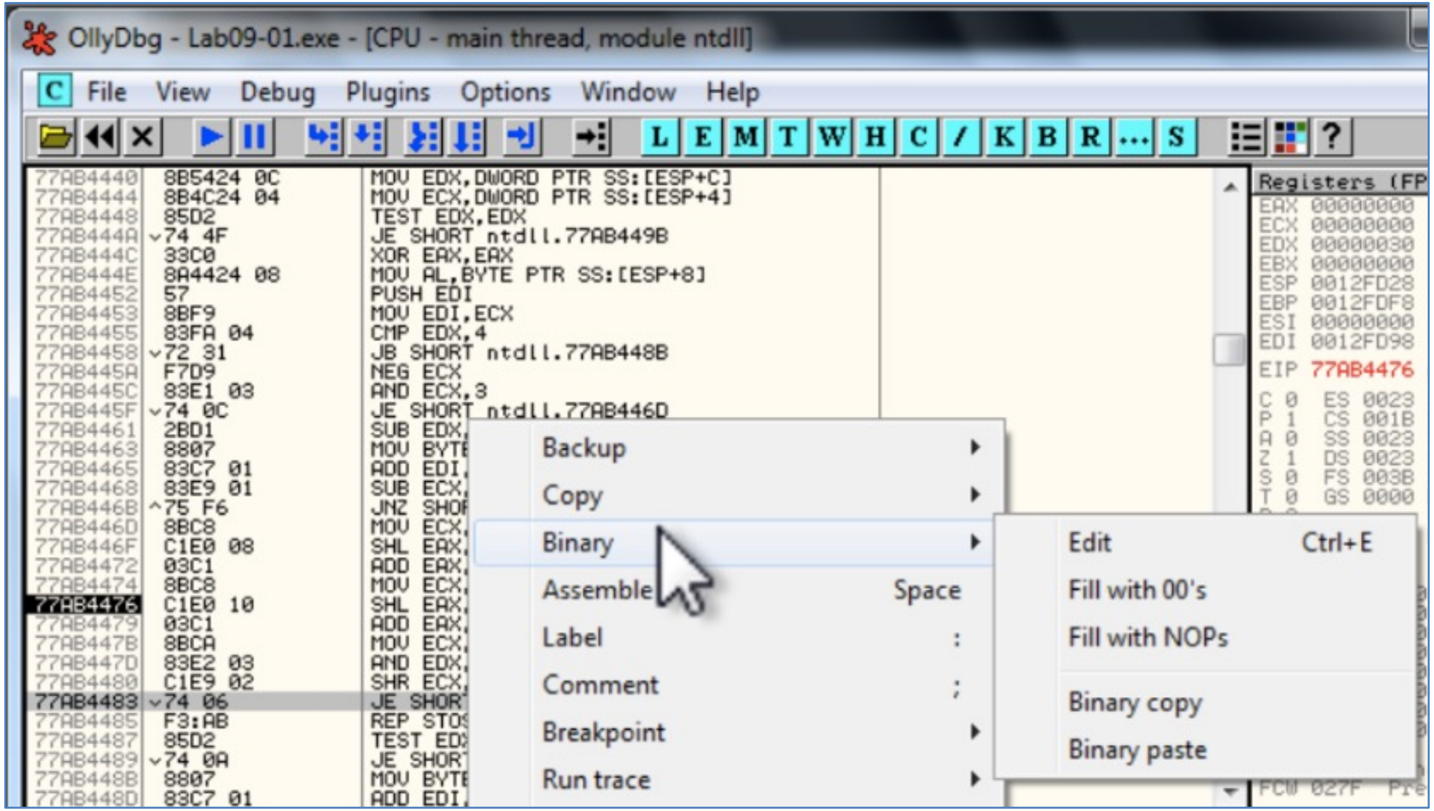
- OllyDbg will stop the program

- You have these options to pass the exception into the program:
  - **Shift+F7**: Step into exception
  - **Shift+F8**: Step over exception
  - **Shift+F9**: Run exception handler

- Often you just ignore all exceptions in malware analysis
  - We are not trying to fix problems in code

PATCHING

Binary Edit



```
77AB4440   8B5424 0C     MOV EDX,DWORD PTR SS:[ESP+C]
77AB4444   8B4C24 04     MOV ECX,DWORD PTR SS:[ESP+4]
77AB4448   85D2          TEST EDX,EDX
77AB444A  v74 4F         JE SHORT ntdll.77AB449B
77AB444C   33C0          XOR EAX,EAX
77AB444E   8A4424 08     MOV AL,BYTE PTR SS:[ESP+8]
77AB4452   57            PUSH EDI
77AB4453   8BF9          MOV EDI,ECX
77AB4455   83FA 04       CMP EDX,4
77AB4458  v72 31         JB SHORT ntdll.77AB448B
77AB445A   F7D9          NEG ECX
77AB445C   83E1 03       AND ECX,3
77AB445F  v74 0C         JE SHORT ntdll.77AB446D
77AB4461   2BD1          SUB EDX,
77AB4463   8807          MOV BYTE
77AB4465   83C7 01       ADD EDI,
77AB4468   83E9 01       SUB ECX,
77AB446B  ^75 F6         JNZ SHOR
77AB446D   8BC8          MOV ECX,
77AB446F   C1E0 08       SHL EAX,
77AB4472   03C1          ADD EAX,
77AB4474   8BC8          MOV ECX,
77AB4476   C1E0 10       SHL EAX,
77AB4479   03C1          ADD EAX,
77AB447B   8BCA          MOV ECX,
77AB447D   83E2 03       AND EDX,
77AB4480   C1E9 02       SHR ECX,
77AB4483  v74 06         JE SHORT
77AB4485   F3:AB         REP STOS
77AB4487   85D2          TEST EDX
77AB4489  v74 0A         JE SHORT
77AB448B   8807          MOV BYTE
77AB448D   83C7 01       ADD EDI,
```

```
Registers (FP
EAX 00000000
ECX 00000000
EDX 00000030
EBX 00000000
ESP 0012FD28
EBP 0012FDF8
ESI 00000000
EDI 0012FD98
EIP 77AB4476
C 0  ES 0023
P 1  CS 001B
A 0  SS 0023
Z 1  DS 0023
S 0  FS 003B
T 0  GS 0000
FCW 027F  Pre
```

Fill

- Fill with 00

- Fill with NOP (0x90)
  - Used to skip instructions
  - e.g. to force a branch

Saving Patched Code

- Right-click disassembler window after patching
  - Copy To Executable, All Modifications, Save File
  - Copy All

- Right-click in new window
  - Save File

ANALYSING SHELLCODE

Undocumented technique

Easy Way to Analyse Shellcode

- Copy shellcode from a hex editor to clipboard
- Within memory map, select a region of type “Priv” (Private memory)
- Double-click rows in memory map to show a hex dump
  - Find a region of hundreds of consecutive zeroes
- Right-click chosen region in Memory Map, Set Access, Full Access (to clear NX bit)

Analysing Shellcode

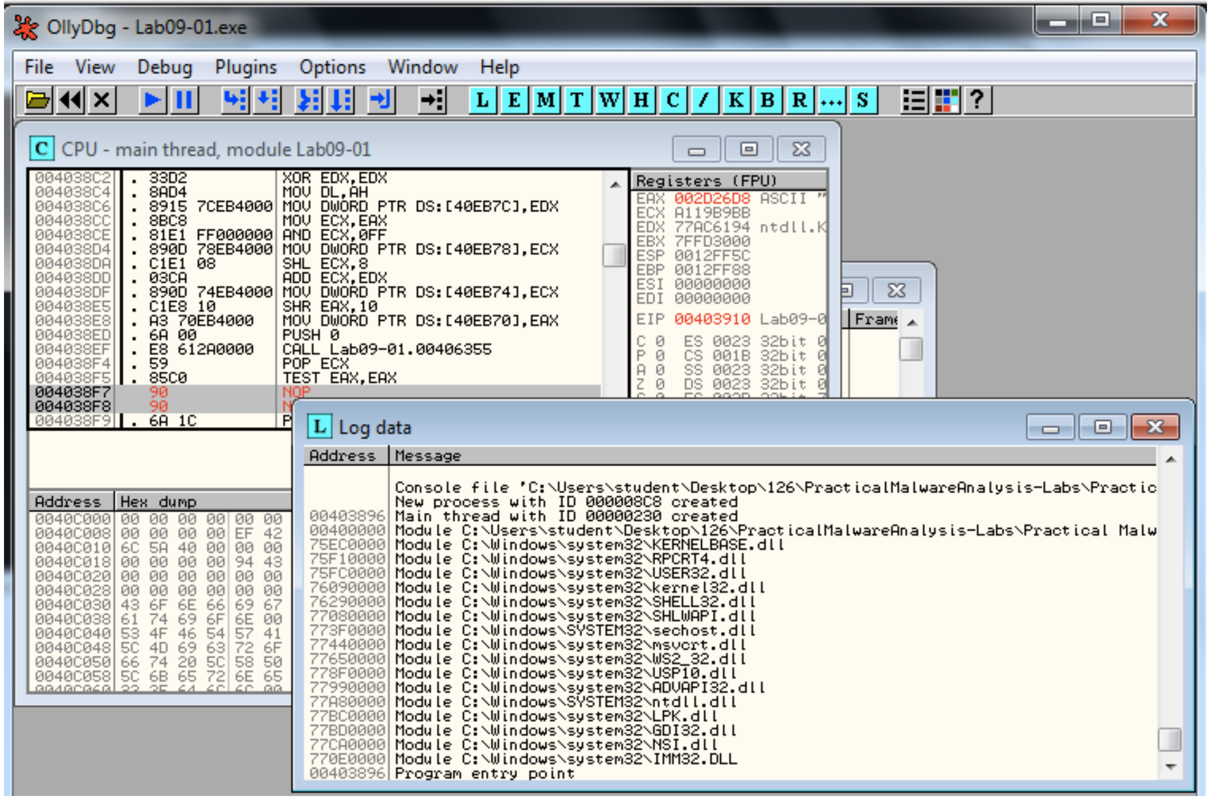
- Highlight a region of zeroes, Binary, Binary Paste

- Set **`EIP`** to location of shellcode
  - Right-click first instruction, **New Origin Here**

ASSISTANCE FEATURES

Log

- View, Log
  - ○ Shows steps to reach here



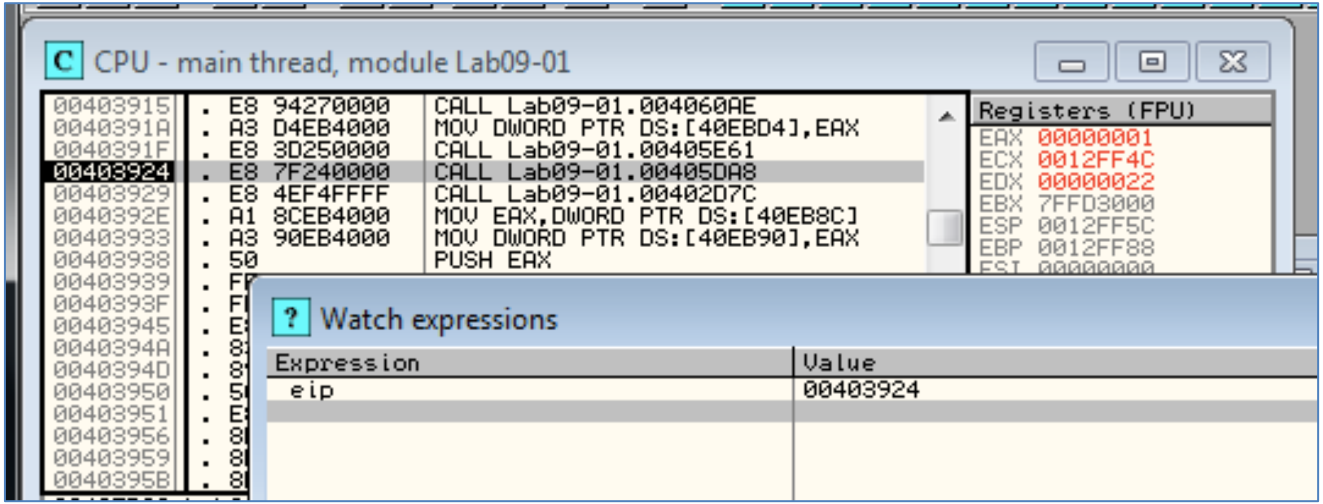
```
004038C2  .  33D2           XOR EDX,EDX
004038C4  .  8AD4           MOV DL,AH
004038C6  .  8915 7CEB4000  MOV DWORD PTR DS:[40EB7C],EDX
004038CC  .  8BC8           MOV ECX,EAX
004038CE  .  81E1 FF000000  AND ECX,0FF
004038D4  .  890D 78EB4000  MOV DWORD PTR DS:[40EB78],ECX
004038DA  .  C1E1 08        SHL ECX,8
004038DD  .  03CA           ADD ECX,EDX
004038DF  .  890D 74EB4000  MOV DWORD PTR DS:[40EB74],ECX
004038E5  .  C1E8 10        SHR EAX,10
004038E8  .  A3 70EB4000    MOV DWORD PTR DS:[40EB70],EAX
004038ED  .  6A 00          PUSH 0
004038EF  .  E8 612A0000    CALL Lab09-01.00406355
004038F4  .  59             POP ECX
004038F5  .  85C0           TEST EAX,EAX
004038F7     90             NOP
004038F8     90             N
004038F9  .  6A 1C          P
```

```
Registers (FPU)
EAX 002D26D8 ASCII "
ECX A119B9BB
EDX 77AC6194 ntdll.K
EBX 7FFD3000
ESP 0012FF5C
EBP 0012FF88
ESI 00000000
EDI 00000000

EIP 00403910 Lab09-0

C 0  ES 0023 32bit 0
P 0  CS 001B 32bit 0
A 0  SS 0023 32bit 0
Z 0  DS 0023 32bit 0
```

```
Address  Hex dump
0040C000 00 00 00 00 00 00
0040C008 00 00 00 00 EF 42
0040C010 6C 5A 40 00 00 00
0040C018 00 00 00 00 94 43
0040C020 00 00 00 00 00 00
0040C028 00 00 00 00 00 00
0040C030 43 6F 6E 66 69 67
0040C038 61 74 69 6F 6E 00
0040C040 53 4F 46 54 57 41
0040C048 5C 4D 69 63 72 6F
0040C050 66 74 20 5C 58 50
0040C058 5C 6B 65 72 6E 65
```

```
Address  Message

         Console file 'C:\Users\student\Desktop\126\PracticalMalwareAnalysis-Labs\Practic
         New process with ID 000008C8 created
00403896 Main thread with ID 00000230 created
00400000 Module C:\Users\student\Desktop\126\PracticalMalwareAnalysis-Labs\Practical Malw
75EC0000 Module C:\Windows\system32\KERNELBASE.dll
75F10000 Module C:\Windows\system32\RPCRT4.dll
75FC0000 Module C:\Windows\system32\USER32.dll
76090000 Module C:\Windows\system32\kernel32.dll
76290000 Module C:\Windows\system32\SHELL32.dll
77080000 Module C:\Windows\system32\SHLWAPI.dll
773F0000 Module C:\Windows\SYSTEM32\sechost.dll
77440000 Module C:\Windows\system32\msvcrt.dll
77650000 Module C:\Windows\system32\WS2_32.dll
778F0000 Module C:\Windows\system32\USP10.dll
77990000 Module C:\Windows\system32\ADVAPI32.dll
77A80000 Module C:\Windows\SYSTEM32\ntdll.dll
77BC0000 Module C:\Windows\system32\LPK.dll
77BD0000 Module C:\Windows\system32\GDI32.dll
77CA0000 Module C:\Windows\system32\NSI.dll
770E0000 Module C:\Windows\system32\IMM32.DLL
00403896 Program entry point
```

Watches Window



```
00403915  .  E8 94270000   CALL Lab09-01.004060AE
0040391A  .  A3 D4EB4000   MOV DWORD PTR DS:[40EBD4],EAX
0040391F  .  E8 3D250000   CALL Lab09-01.00405E61
00403924  .  E8 7F240000   CALL Lab09-01.00405DA8
00403929  .  E8 4EF4FFFF   CALL Lab09-01.00402D7C
0040392E  .  A1 8CEB4000   MOV EAX,DWORD PTR DS:[40EB8C]
00403933  .  A3 90EB4000   MOV DWORD PTR DS:[40EB90],EAX
00403938  .  50            PUSH EAX
00403939  .  FF
0040393F  .  FI
00403945  .  E
0040394A  .  8
0040394D  .  8
00403950  .  5
00403951  .  E
00403956  .  8
00403959  .  8
0040395B  .  8
```

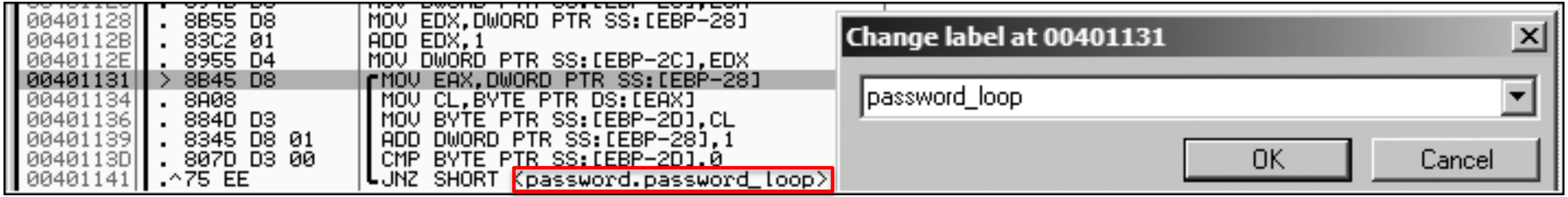
```
Registers (FPU)
EAX 00000001
ECX 0012FF4C
EDX 00000022
EBX 7FFD3000
ESP 0012FF5C
EBP 0012FF88
ESI 00000000
```

Expression	Value
eip	00403924

- View, Watches
  - Watch the value of an expression
  - Press **SPACEBAR** to set expression
  - OllyDbg Help, Contents
    - Instructions for Evaluation of Expressions

Labelling

- Label subroutines and loops
  - Right-click an address, Label



```
00401128  .  8B55 D8      MOV EDX,DWORD PTR SS:[EBP-28]
0040112B  .  83C2 01      ADD EDX,1
0040112E  .  8955 D4      MOV DWORD PTR SS:[EBP-2C],EDX
00401131  >  8B45 D8     ┌MOV EAX,DWORD PTR SS:[EBP-28]
00401134  .  8A08        │MOV CL,BYTE PTR DS:[EAX]
00401136  .  884D D3     │MOV BYTE PTR SS:[EBP-2D],CL
00401139  .  8345 D8 01  │ADD DWORD PTR SS:[EBP-28],1
0040113D  .  807D D3 00  │CMP BYTE PTR SS:[EBP-2D],0
00401141  .^75 EE        └JNZ SHORT <password.password_loop>
```

Change label at 00401131

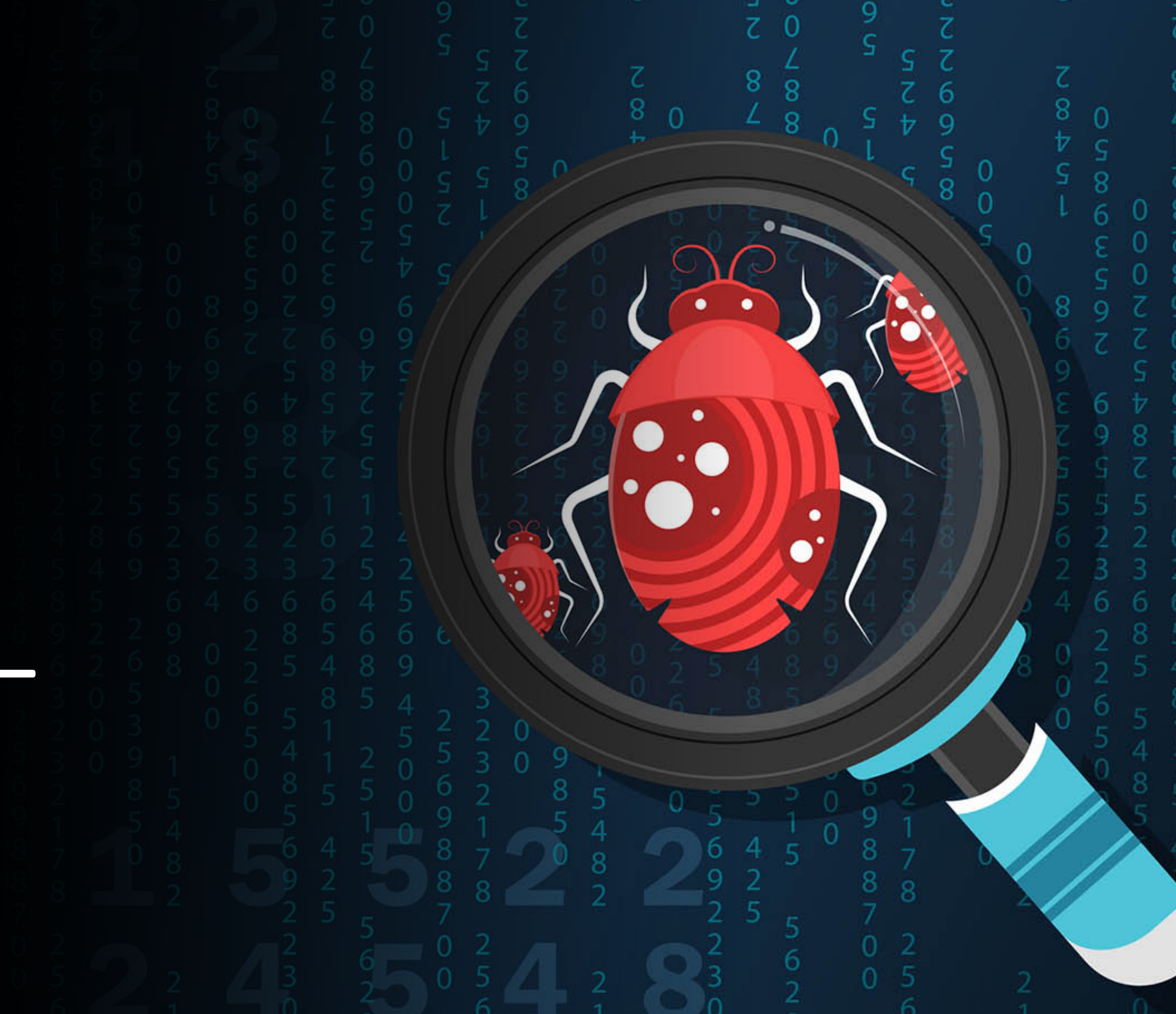
password_loop

PLUG-INS

Recommended Plugins

- **OllyDump**
  - Dumps debugged process to a PE file
  - Used for ***unpacking***
- **Hide Debugger**
  - Hides OllyDbg from ***debugger detection***
- **Command Line**
  - Control OllyDbg from the command line
  - Simpler to just use WinDbg
- **Bookmarks**
  - Included by default in OllyDbg
  - Bookmarks memory locations

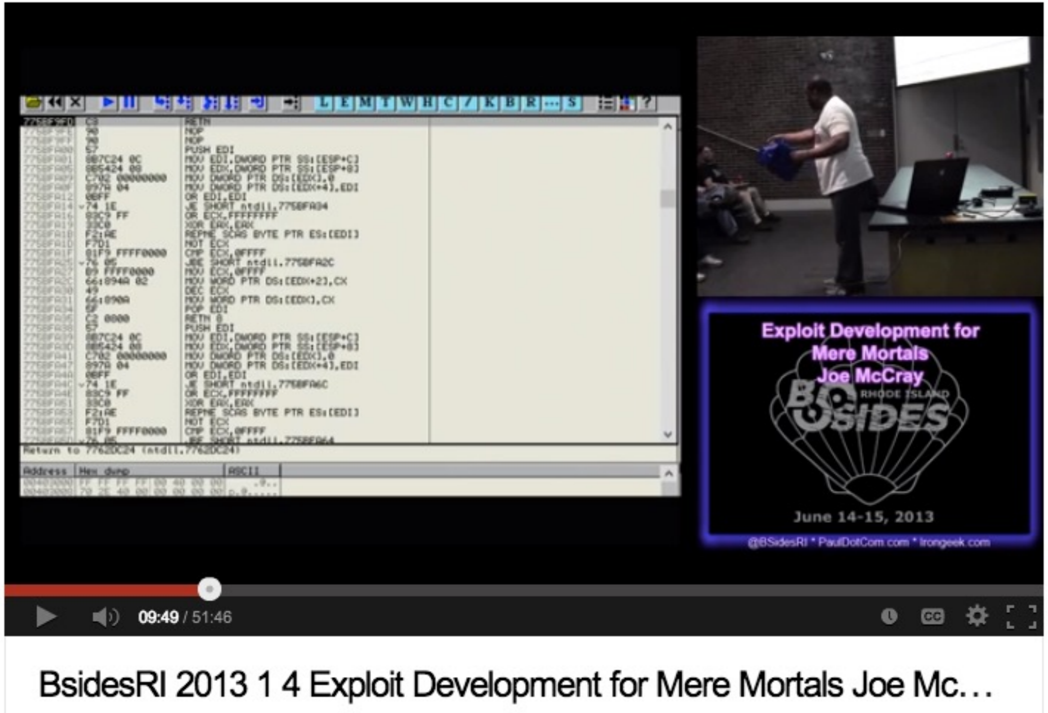
SCRIPTABLE DEBUGGING



Immunity Debugger (ImmDbg)

- Unlike OllyDbg, ImmDbg employs Python scripts and has an easy-to-use API
- Scripts are located in the **`PyCommands`** subdirectory under the install directory of ImmDbg
- Easy to create custom scripts for ImmDbg

Good Intro to OllyDbg



```
7758F9FD  C3                RETN
7758F9FE  90                NOP
7758F9FF  90                NOP
7758FA00  57                PUSH EDI
7758FA01  8B7C24 0C         MOV EDI,DWORD PTR SS:[ESP+C]
7758FA05  8B5424 08         MOV EDX,DWORD PTR SS:[ESP+8]
7758FA09  C702 00000000     MOV DWORD PTR DS:[EDX],0
7758FA0F  897A 04           MOV DWORD PTR DS:[EDX+4],EDI
7758FA12  0BFF              OR EDI,EDI
7758FA14 v74 1E             JE SHORT ntdll.7758FA34
7758FA16  83C9 FF           OR ECX,FFFFFFFF
7758FA19  33C0              XOR EAX,EAX
7758FA1B  F2:AE             REPNE SCAS BYTE PTR ES:[EDI]
7758FA1D  F7D1              NOT ECX
7758FA1F  81F9 FFFF0000     CMP ECX,0FFFF
7758FA25 v76 05             JBE SHORT ntdll.7758FA2C
7758FA27  B9 FFFF0000       MOV ECX,0FFFF
7758FA2C  66:894A 02        MOV WORD PTR DS:[EDX+2],CX
7758FA30  49                DEC ECX
7758FA31  66:890A           MOV WORD PTR DS:[EDX],CX
7758FA34  5F                POP EDI
7758FA35  C2 0000           RETN 0
7758FA38  57                PUSH EDI
7758FA39  8B7C24 0C         MOV EDI,DWORD PTR SS:[ESP+C]
7758FA3D  8B5424 08         MOV EDX,DWORD PTR SS:[ESP+8]
7758FA41  C702 00000000     MOV DWORD PTR DS:[EDX],0
7758FA47  897A 04           MOV DWORD PTR DS:[EDX+4],EDI
7758FA4A  0BFF              OR EDI,EDI
7758FA4C v74 1E             JE SHORT ntdll.7758FA6C
7758FA4E  83C9 FF           OR ECX,FFFFFFFF
7758FA51  33C0              XOR EAX,EAX
7758FA53  F2:AE             REPNE SCAS BYTE PTR ES:[EDI]
7758FA55  F7D1              NOT ECX
7758FA57  81F9 FFFF0000     CMP ECX,0FFFF
7758FA5D v76 05             JBE SHORT ntdll.7758FA64
Return to 7762DC24 (ntdll.7762DC24)

Address   Hex dump                  ASCII
00403000  FF FF FF FF 00 40 00 00      .@..
00403008  70 2E 40 00 00 00 00 00   p.@.....
```

BsidesRI 2013 1 4 Exploit Development for Mere Mortals Joe Mc…

https://www.youtube.com/watch?v=eNSWUAVxbzk

END OF LECTURE. THANK YOU.