

Analyzing Windows Malware Practical Labs

Software Security

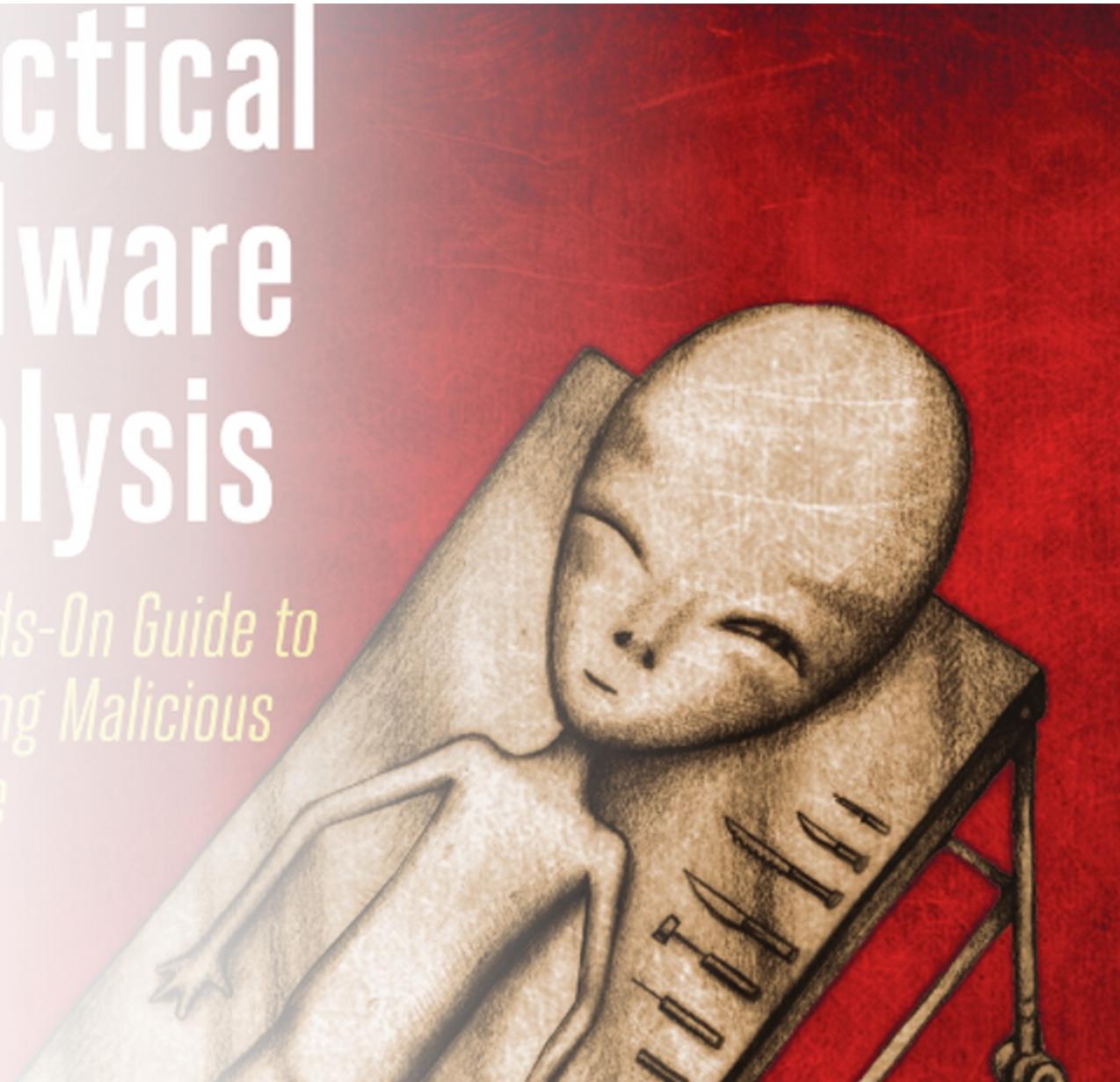
a.a. 2022/2023

Laurea Magistrale in Ing. Informatica

Roberto Natella

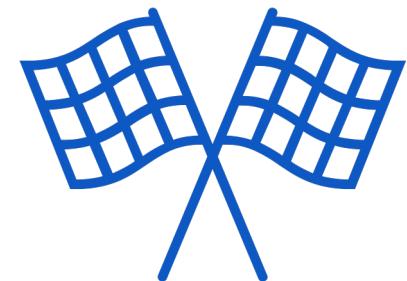
practical Malware analysis

*A Hands-On Guide to
Dissecting Malicious
Software*



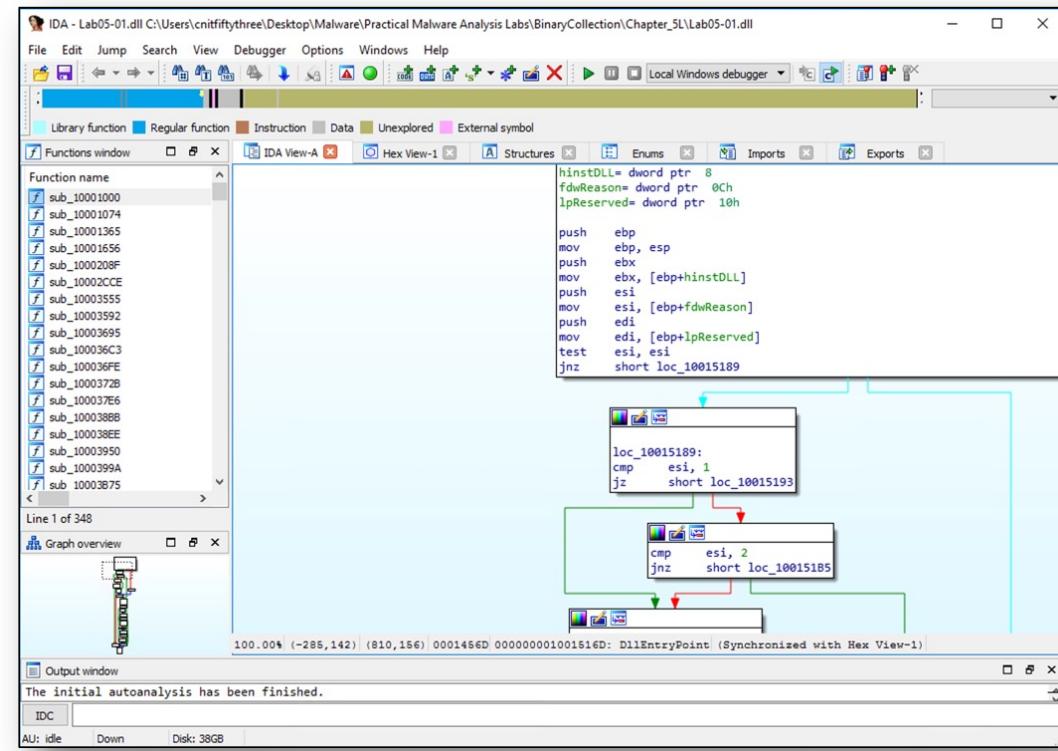
Labs overview

- Follow the steps in the slides
- The first target asks for **secret "flags"** to check your progress!



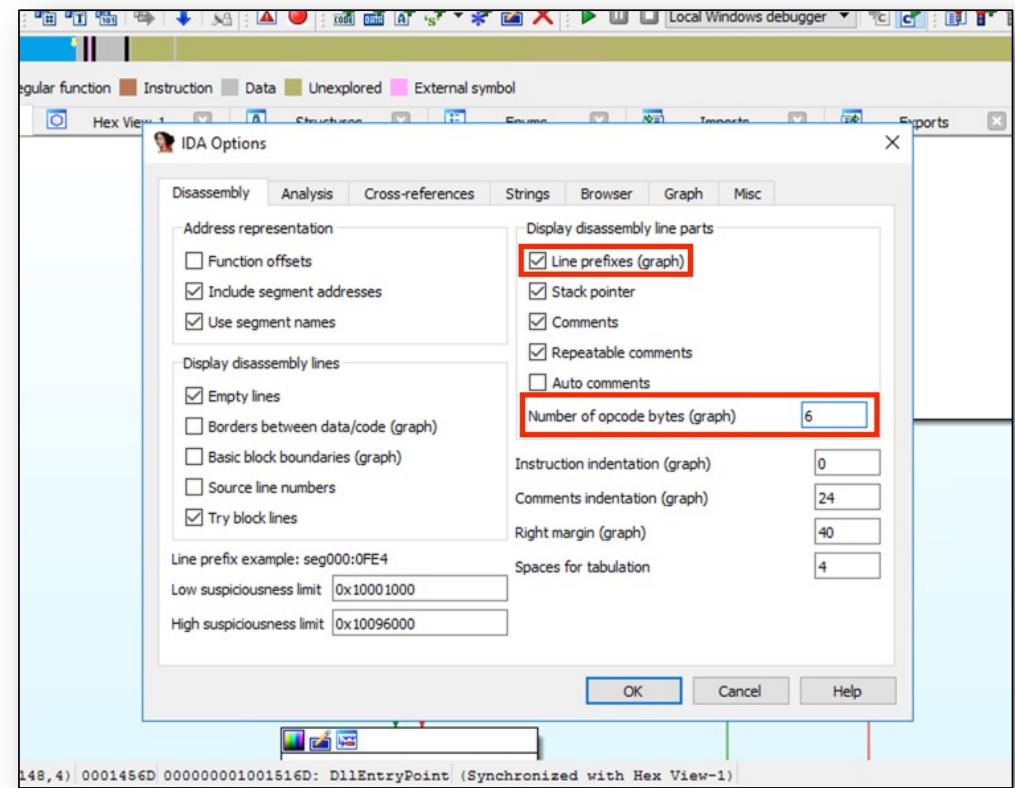
Malware Static Analysis

- Run IDA Pro
- Open the malware sample **Lab05-01.dll**



Adjusting Graph Mode

- The code doesn't show line numbers or hexadecimal instructions. To fix that, click **Options, General**
- In the IDA Options box, on the **Disassembly** tab, in the top right, in the "**Display disassembly line parts**" section, make these changes, as shown below:
 - Check "**Line prefixes (graph)**"
 - Change "**Number of opcode bytes**" to **6**



Adjusting Graph Mode

The screenshot shows the IDA Pro interface for analyzing the DLL entry point of Lab05-01.dll. The assembly view displays the initial autoanalysis results, including variable declarations and the main entry point code.

```

000000001001516D hinstDLL= dword ptr  8
000000001001516D fdwReason= dword ptr  0Ch
000000001001516D lpReserved= dword ptr  10h
000000001001516D
000000001001516D 000 55 push    ebp
000000001001516E 004 8B EC mov     ebp, esp
0000000010015170 004 53 push    ebx
0000000010015171 008 8B 5D 08 mov     ebx, [ebp+hinstDLL]
0000000010015174 008 56 push    esi
0000000010015175 00C 88 75 0C mov     esi, [ebp+fdwReason]
0000000010015178 00C 57 push    edi
0000000010015179 010 8B 7D 10 mov     edi, [ebp+lpReserved]
000000001001517C 010 85 F6 test   esi, esi
000000001001517E 010 75 09 jnz    short loc_10015189

```

The graph mode visualization highlights specific nodes and edges:

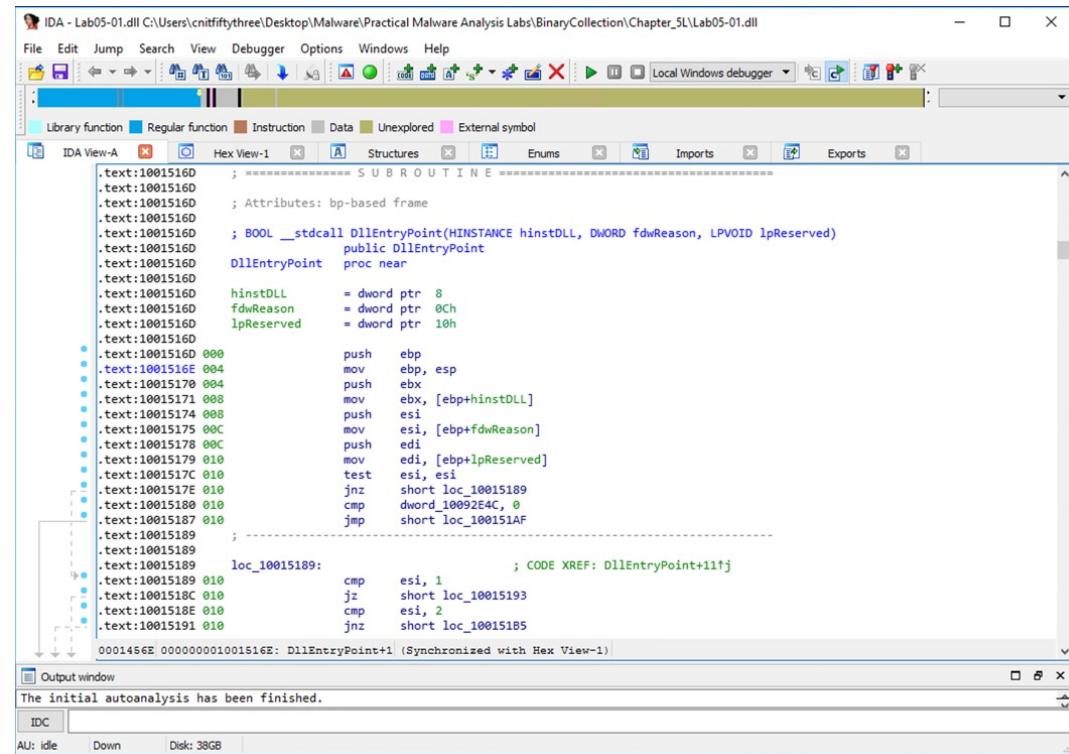
- loc_10015189:** A node containing the assembly code for the entry point. It has three outgoing edges to other nodes.
- loc_1001518E:** A node containing assembly code. It receives one edge from the entry point node and has two outgoing edges to another node.
- loc_10015191:** A node containing assembly code. It receives one edge from the previous node and has one outgoing edge to a final node.
- loc_10015185:** A node containing assembly code. It receives one edge from the third node and has no outgoing edges.

The bottom status bar indicates: 100.00% (41,129) (1049,430) 0001456D 000000001001516D: DllEntryPoint (Synchronized with Hex View-1).

The Output window shows: The initial autoanalysis has been finished.

Adjusting Graph Mode

- Press **SPACEBAR** to switch to text-only view
- Press **SPACEBAR** again to go back to graph mode



The screenshot shows the IDA Pro interface with the assembly view selected. The window title is "IDA - Lab05-01.dll C:\Users\cniftfiftythree\Desktop\Malware\Practical Malware Analysis Labs\BinaryCollection\Chapter_5L\Lab05-01.dll". The assembly code for the DllEntryPoint function is displayed:

```
.text:10015160 ; ===== SUBROUTINE =====
.text:10015160 ; Attributes: bp-based frame
.text:10015160
.text:10015160 ; BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
.text:10015160     public DllEntryPoint
.text:10015160 DllEntryPoint proc near
.text:10015160
.text:10015160     hinstDLL    = dword ptr  8
.text:10015160     fdwReason   = dword ptr  0Ch
.text:10015160     lpReserved  = dword ptr  10h
.text:10015160
.text:10015160     .text:10015160 000          push   ebp
.text:10015160     .text:10015160 004          mov    ebp, esp
.text:10015160     .text:10015170 008          push   ebx, [ebp+hinstDLL]
.text:10015160     .text:10015174 008          push   esi, [ebp+fdwReason]
.text:10015160     .text:10015175 00C          push   edi, [ebp+lpReserved]
.text:10015160     .text:10015179 010          mov    edi, [ebp+lpReserved]
.text:10015160     .text:1001517C 010          test   esi, esi
.text:10015160     .text:1001517E 010          jnz    short loc_10015189
.text:10015160     .text:10015180 010          cmp    dword_10092E4C, 0
.text:10015160     .text:10015187 010          jmp    short loc_100151AF
.text:10015160
.text:10015160     .text:10015189 loc_10015189:           ; CODE XREF: DllEntryPoint+11+j
.text:10015160     .text:10015189 010          cmp    esi, 1
.text:10015160     .text:1001518C 010          jz    short loc_10015193
.text:10015160     .text:1001518E 010          cmp    esi, 2
.text:10015160     .text:10015191 010          jnz    short loc_10015185
0001456E 000000001001516E: DllEntryPoint+1 (Synchronized with Hex View-1)
```

The bottom status bar indicates "The initial autoanalysis has been finished." and "AU: idle".

IDA on non-OS X/Retina Hi-DPI displays

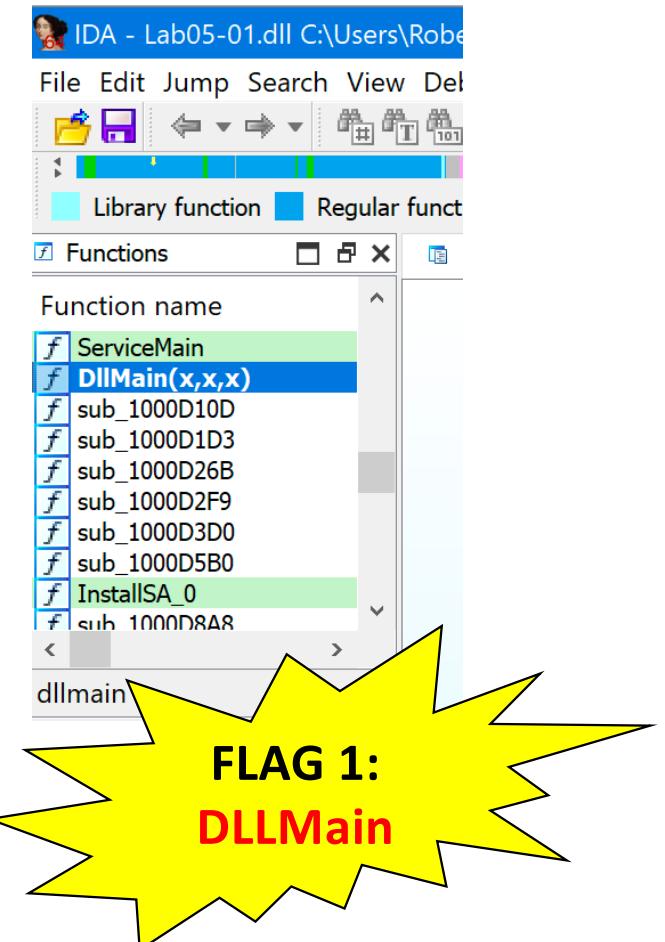
- Some users running IDA on Windows & Linux X11 platforms with Hi-DPI displays, have reported that IDA looks rather odd: the navigator bar is too narrow, the text under it gets truncated, and there is overall feeling of packing & clumsiness:
- On Linux X11 & Windows, try setting the environment variable **QT_AUTO_SCREEN_SCALE_FACTOR** to 1:
- E.g., on Linux/X11:

```
~# export QT_AUTO_SCREEN_SCALE_FACTOR=1
~# path/to/ida my.idb
```



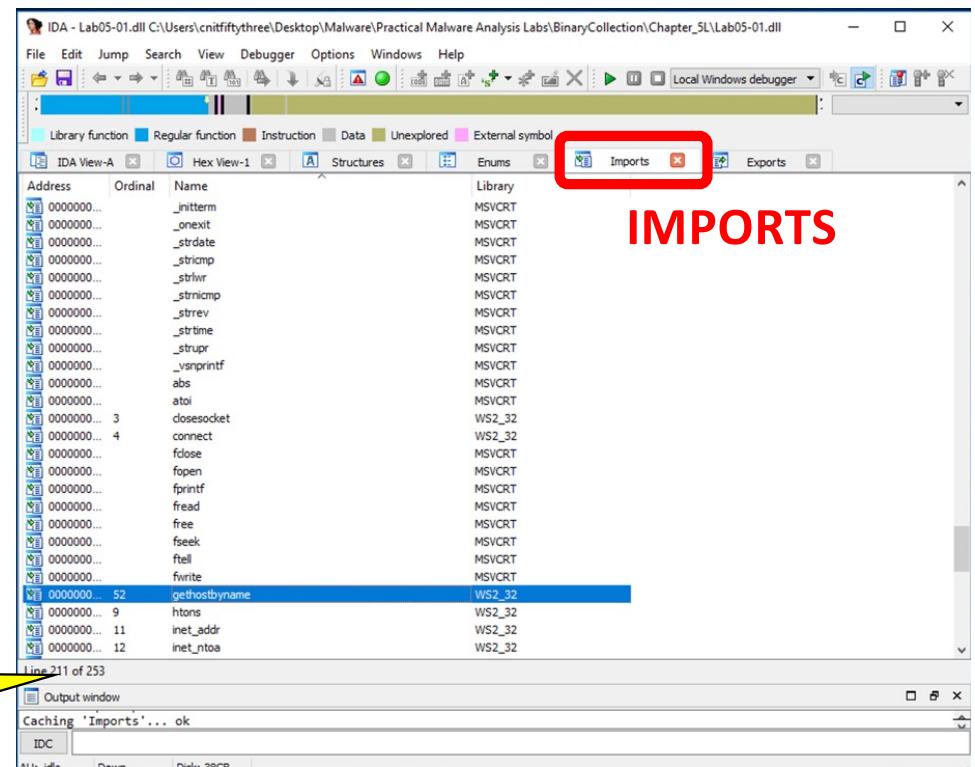
Functions

- What is the **address of DllMain?**
- You can find it in the "Functions" view
- For a quick look-up:
 - Click within the view
 - Type the first letters of the function name
- Scroll the view to find more info
- You can double-click to find it in the graph view
- If the screen is too cluttered, close the Functions view, or make it floating



Imports

- Use the Imports window to browse to **gethostbyname**
- At which address is the import located?
 - Click **View**, "Open subview", **Imports**
 - Click the **Name** header to sort by name
 - Find "gethostbyname" by scrolling
 - Or, type the first few letters of the name
 - Double-click to view it in the graph



Xrefs

- How many functions call gethostbyname?
- Press **Ctrl+x** to open the "xrefs to gethostbyname" box
- Note: only count xrefs with type "p" (calls)

CLICK, CONTROL+X



CLICK, CONTROL+X

; struct hostent *(_stdcall *gethostbyname)(const char *name)
extrn gethostbyname:dword : CODE XREF: sub_10001074:loc_10

Direct	Type	Address	Text
Up	p	sub_10001656+101	call ds:gethostbyname
Up	p	sub_1000208F+3A1	call ds:gethostbyname
Up	p	sub_10002CCE+4F7	call ds:gethostbyname

Line

OK Cancel Search Help

; int (_stdcall *send)(SOCKET s, const char *buf, int len, int flags)

DNS

- Focusing on the **call to gethostname located at 0x10001757**, can you figure out which **DNS request** will be made?
- Double-click on the label of the symbol (`off_10019040`) to find its definition

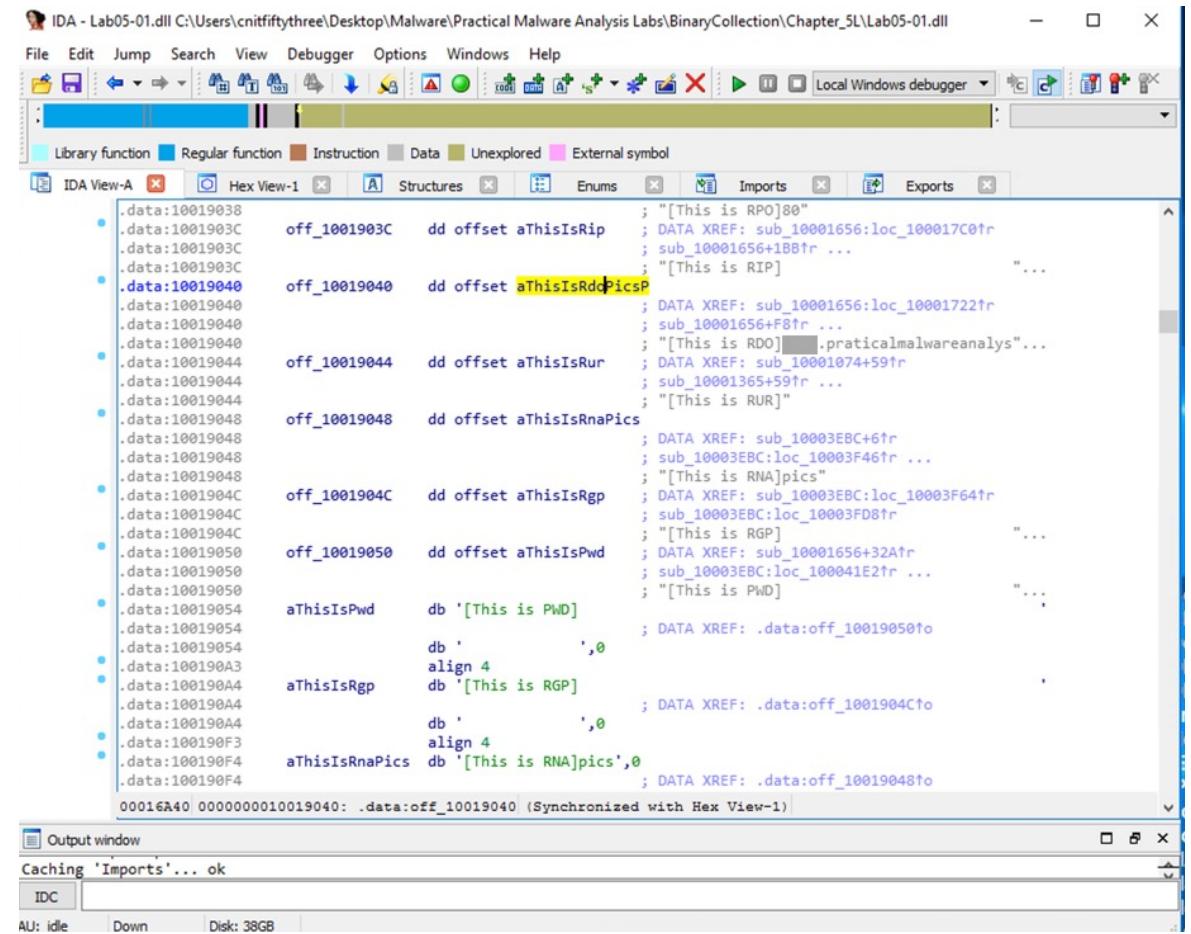
```

IDA - Lab05-01.dll C:\Users\cnitfiftythree\Desktop\Malware\Practical Malware Analysis Labs\BinaryCollection\Chapter_5L\Lab05-01.dll
File Edit Jump Search View Debugger Options Windows Help
Library function Regular function Instruction Data Unexplored External symbol
IDA View-A Hex View-1 Structures Enums Imports Exports
0000000010001737 694 B3 C4 0C add esp, 0Ch
000000001000173A 688 B5 C0 test eax, eax
000000001000173C 688 0F 84 AB 00 00 jz loc_100017ED
0000000010001742 688 39 1D CC E5 08 10 cmp dword_1008E5CC, ebx
0000000010001748 688 0F 85 9F 00 00 jnz loc_100017ED
000000001000174E 688 A1 40 90 01 10 mov eax, off_10019040
0000000010001753 688 B3 C0 0D add eax, 0Dh
0000000010001756 688 50 push eax ; name
0000000010001757 68C FF 15 CC 63 01 10 call ds:gethostname
000000001000175D 688 8B F0 mov esi, eax
000000001000175F 688 3B F3 cmp esi, ebx
0000000010001761 688 74 5D jz short loc_100017C0
00000000100017C0 688 A1 3C 90 01 10 mov eax, off_1001903C
00000000100017C5 688 B8 35 B8 62 01 10 mov esi, ds:strncpy
00000000100017CB 688 B3 C0 0D add eax, 0Dh
00000000100017CE 688 6A 10 push 10h ; Co
00000000100017D0 68C 50 push eax ; So
00000000100017D1 690 68 D8 E5 08 10 push offset Dest : De
BF 46 0A movsx eax, word ptr [esi+0Ah]
46 0C push eax ; Size
30 mov eax, [esi+0Ch]
44 24 40 push dword ptr [eax] ; Src
44 24 40 lea eax, [esp+690h+Dst]
44 24 40 push eax ; Dst
C9 37 01 00 call memcp
8B 46 08 mov ax, [esi+81]
100.00% (6492,1965) (899,252) 00000B57 0000000010001757: sub_10001656+101 (Synchronized with Hex View-1)
Output window
Caching 'Imports'... ok
IDC
AU: idle Down Disk: 38GB

```

DNS

- The Text view shows that this location contains a **pointer to a string** containing "**practicalmalwareanalys**"



The screenshot shows the IDA Pro debugger interface. The assembly window displays the following code snippet:

```

.data:10019038 off_1001903C dd offset aThisIsRip ; "[This is RPO]80"
.data:1001903C .data:1001903C .data:1001903C .data:1001903C
.data:10019040 off_10019040 dd offset aThisIsRdcPicsP ; DATA XREF: sub_10001656:loc_10001722r ...
.data:10019040 .data:10019040 .data:10019040 .data:10019040
.data:10019044 off_10019044 dd offset aThisIsRur ; "[This is RDO]practicalmalwareanalys...
.data:10019044 .data:10019044 .data:10019044 .data:10019044
.data:10019048 off_10019048 dd offset aThisIsRnaPics ; "[This is RUR]"
.data:10019048 .data:10019048 .data:10019048 .data:10019048
.data:1001904C off_1001904C dd offset aThisIsRgp ; DATA XREF: sub_10003EBC+6tr ...
.data:1001904C .data:1001904C .data:1001904C .data:1001904C
.data:10019050 off_10019050 dd offset aThisIsPwd ; "[This is RNA]pics"
.data:10019050 .data:10019050 .data:10019050 .data:10019050
.data:10019054 aThisIsPwd db '[This is PWD]' ; DATA XREF: sub_10003EBC:loc_10003F64tr ...
.data:10019054 .data:10019054 .data:10019054 .data:10019054
.data:10019054 db ' ',0 align 4 ; DATA XREF: .data:off_10019050to
.data:100190A3 aThisIsRgp db '[This is RGP]' ; "[This is RGP]"
.data:100190A3 .data:100190A3 .data:100190A3 .data:100190A3
.data:100190A4 db ' ',0 align 4 ; DATA XREF: .data:off_1001904Cto
.data:100190F3 .data:100190F3 .data:100190F3 .data:100190F3
.data:100190F4 aThisIsRnaPics db '[This is RNA]pics',0 ; DATA XREF: .data:off_10019048to
.data:100190F4 .data:100190F4 .data:100190F4 .data:100190F4

```

The comments in the assembly code provide context for the strings found at various memory locations. The strings themselves are visible in the assembly code above, such as "practicalmalwareanalys" and "[This is RGP]".

DNS

- The four bytes starting at 10019040 contain a 32-bit address in little-endian order (**highlighted in blue**)
- That address is 10019194. There's a series of ASCII values at that address (**highlighted in green**)
- Skipping the first 13 bytes leaves a string ending in ".practicalmalwareanalysis.com"
- The flag is covered by a green box

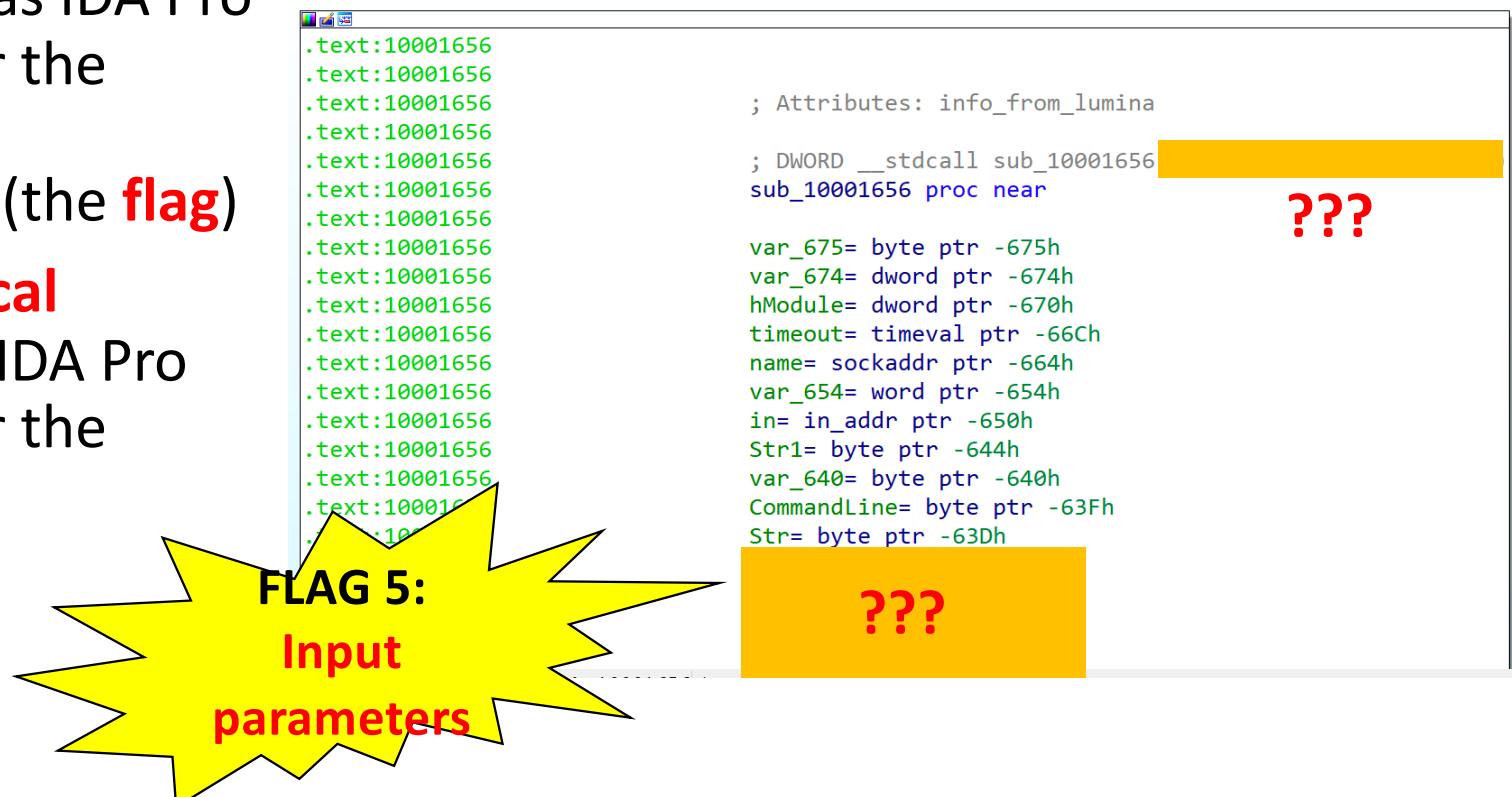


IDA View-A Hex View-1 Structures Enums

Address	Value	Content
10019000	00 00 00 00ep..CH..è...
10019010	65 70 00 10
10019020	43 A4 00 10	~'..~'..~'..p..
10019030	E8 11 01 10	\'..H'..4'..ä..
10019040	84 92 01 10	''..D'..ô..H..
10019050	34 92 01 10	T...[This is PWD
10019060	F4 90 01 10].....
10019070	A4 90 01 10
10019080	54 90 01 10
10019090	58 54 68 69
100190A0	73 20 69 73[This is RGP
100190B0	20 20 20 20]
100190C0	20 20 20 20
100190D0	20 20 20 20
100190E0	20 20 20 20
100190F0	58 54 68 69[This is RNA
10019100	73 20 69 73]pics.....
10019110	20 20 20 20
10019120	00 00 00 00
10019130	00 00 00 00
10019140	58 54 68 69[This is RUR
10019150	73 20 69 73]
10019160	20 20 20 20
10019170	00 00 00 00
10019180	00 00 00 00
10019190	52 44 4F[This is RDO
100191A0	61 74 69 63].practicalma
100191B0	61 6C 6D 61	lwareanalysis.co
100191C0	6C 79 73 69	m.....
100191D0	73 20 69 73[This is RIP
100191E0	52 49 50
00016A40	0000000010019040:	.data:off_10019040 (Synchronized with IDA View-A)

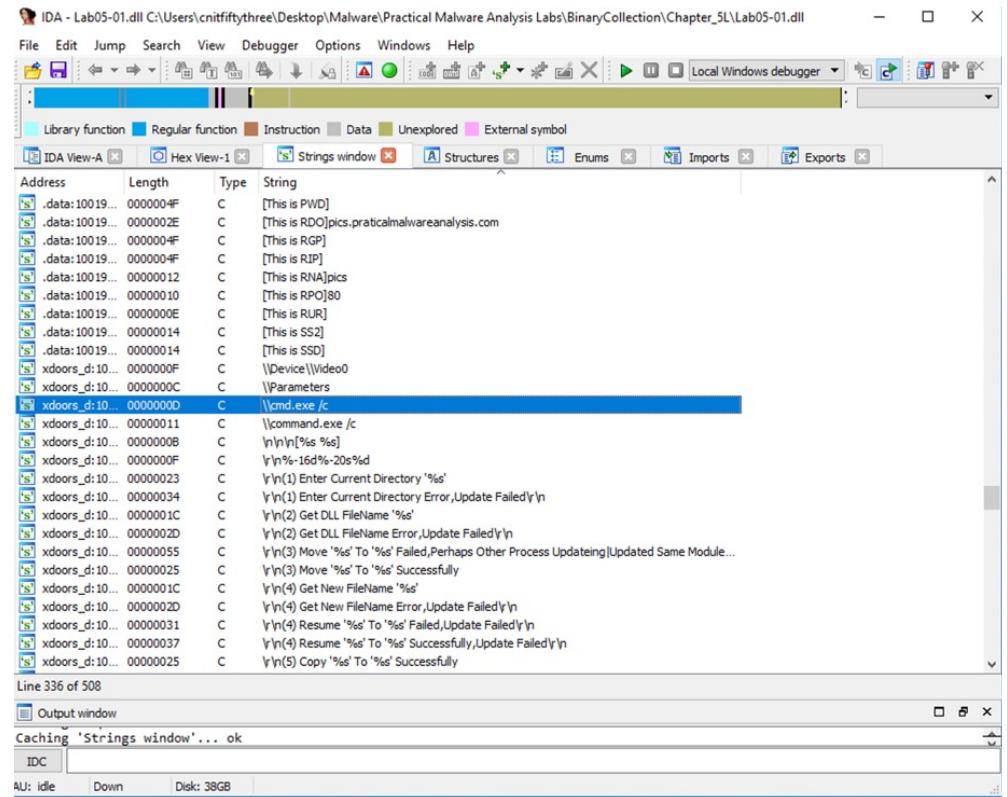
Local vars, parameters

- How many input parameters has IDA Pro recognized for the subroutine at 0x10001656? (the flag)
 - How many local variables has IDA Pro recognized for the subroutine at 0x10001656?



Strings

- Use the **Strings** window to locate the string "**\cmd.exe /c**" in the disassembly. Where is it located?
 - Click View, "Open subview", **Strings**
 - Type the first letters to find the string



Strings

- What is happening in the **area of code that references \cmd.exe /c?**
 - Click in the word **cmd** to highlight it
 - Press **Ctrl+x** to find references from code
 - Double-click **sub_1000FF58+278**

Strings

- There are two boxes of code, one starting with "**cmd.exe -c**" (used in 32-bit systems) and the other starting with "**command.exe /c**" (used in 16-bit systems)
 - Any idea about its purpose?

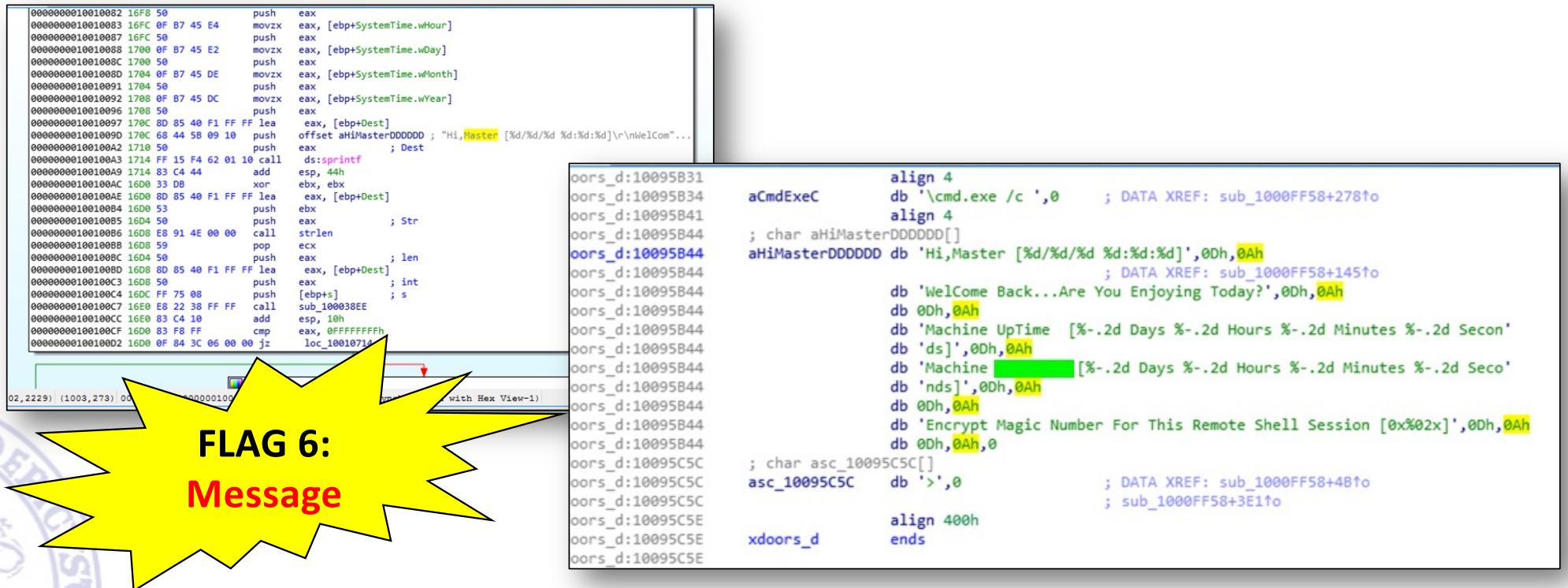
The screenshot displays three windows from the OllyDbg debugger:

- Assembly Window:** Shows assembly code starting at address 000000001001010C. It includes instructions like push, mov, and call, with labels such as loc_100101D7 and ds:GetStartupInfoA.
- Memory Dump (Stack View):** Shows the stack state at address 000000001001010D. It contains pushes for command-line arguments and environment variables, followed by a jump to loc_100101DC.
- Memory Dump (Stack View):** Shows the stack state at address 00000000100101D7. It shows the continuation of the stack with pushes for command-line arguments and environment variables, followed by a jump to loc_100101D7.
- Memory Dump (Stack View):** Shows the stack state at address 00000000100101DC. It shows the continuation of the stack with pushes for command-line arguments and environment variables, followed by a jump to loc_100101DC.

Arrows indicate the flow of control between the assembly code and the stack dump windows, showing how each instruction's destination is reflected in the stack's current state.

Strings

- Navigate upwards in the graph, to find the message "Hi, Master"
- Double-click "Hi, Master" to see more of the string, and find the flag



The screenshot shows a debugger interface with assembly code on the left and a corresponding hex dump on the right. A large yellow starburst points to the string "Hi, Master" in the assembly code.

```

0000000010010082 16F8 50      push    eax
0000000010010083 16FC 0F B7 45 E4  movzx  eax, [ebp+SystemTime.wHour]
0000000010010087 16FC 50      push    eax
0000000010010088 1700 0F B7 45 E2  movzx  eax, [ebp+SystemTime.wDay]
0000000010010088 1700 50      push    eax
000000001001008D 1704 0F B7 45 DE  movzx  eax, [ebp+SystemTime.wMonth]
0000000010010091 1704 50      push    eax
0000000010010092 1708 0F B7 45 DC  movzx  eax, [ebp+SystemTime.wYear]
0000000010010096 1708 50      push    eax
0000000010010097 170C 8D 85 40 F1 FF FF lea    eax, [ebp+Dest]
000000001001009D 170C 68 44 58 09 10  push   offset aHiMasterDDDDDD ; "Hi,Master [%d/%d/%d %d:%d:%d]\r\nWelCom...
00000000100100A2 1710 50      push    eax
00000000100100A3 1714 FF 15 F4 62 01 10 call ds:sprintf
00000000100100A9 1714 83 C4 44  add    esp, 44h
00000000100100AC 1608 33 DB  xor    ebx, ebx
00000000100100AE 1608 8D 85 40 F1 FF FF lea    eax, [ebp+Dest]
00000000100100B4 1608 53      push    ebx
00000000100100B5 1604 50      push    eax
00000000100100B6 1608 E8 91 4E 00 00  call  strlen
00000000100100B8 1608 59      pop     ecx
00000000100100B8 1604 50      push    eax
00000000100100BD 1608 8D 85 40 F1 FF FF lea    eax, [ebp+Dest]
00000000100100C3 1608 50      push    eax
00000000100100C4 160C FF 75 08  push   [ebp+s]
00000000100100C7 1608 E8 22 38 FF FF call  sub_100038EE
00000000100100CC 1608 83 C4 10  add    esp, 10h
00000000100100CF 1608 83 F8 FF  cmp    eax, 0FFFFFFFh
00000000100100D2 1608 0F 84 3C 06 00 00  jz   loc_10010714

oors_d:10095B31      align 4
aCmdExeC           db '\cmd.exe /c ',0      ; DATA XREF: sub_1000FF58+278t0
oors_d:10095B34      align 4
oors_d:10095B41      ; char aHiMasterDDDDDD[]
oors_d:10095B44      aHiMasterDDDDDD db 'Hi,Master [%d/%d/%d %d:%d:%d]',0Dh,0Ah
oors_d:10095B44      ; DATA XREF: sub_1000FF58+145t0
oors_d:10095B44      db 'WelCome Back...Are You Enjoying Today?',0Dh,0Ah
oors_d:10095B44      db 0Dh,0Ah
oors_d:10095B44      db 'Machine UpTime [%-.2d Days %.2d Hours %.2d Minutes %.2d Secon'
oors_d:10095B44      db 'ds',0Dh,0Ah
oors_d:10095B44      db 'Machine [%.2d Days %.2d Hours %.2d Minutes %.2d Seco'
oors_d:10095B44      db 'nds]',0Dh,0Ah
oors_d:10095B44      db 0Dh,0Ah
oors_d:10095B44      db 'Encrypt Magic Number For This Remote Shell Session [0x%02x]',0Dh,0Ah
oors_d:10095B44      db 0Dh,0Ah,0
oors_d:10095C5C      ; char asc_10095C5C[]
oors_d:10095C5C      asc_10095C5C db '>',0
oors_d:10095C5C      ; DATA XREF: sub_1000FF58+4Bt0
oors_d:10095C5E      oors_d:10095C5E align 400h
oors_d:10095C5E      ends

```

**FLAG 6:
Message**

Global variable

- In the same area, at 0x100101C8, it looks like **dword_1008E5C4** is a **global variable** that helps decide which path to take
- How does the malware **set dword_1008E5C4**?
- Hint: Use dword_1008E5C4's cross-references

The screenshot shows a debugger interface with assembly code on the left and a 'xrefs to dword_1008E5C4' dialog on the right.

Assembly Code:

```

    .text:100101B0      lea     eax, [ebp+CommandLine]
    .text:100101B6      mov     [ebp+StartupInfo.wShowWindow], bx
    .text:100101BA      push    eax ; lpBuffer
    .text:100101B8      mov     [ebp+StartupInfo.dwFlags], 101h
    .text:100101C2      call    ds:GetSystemDirectoryA
    • .text:100101C8      cmp     dword 1008E5C4, ebx
    .text:100101CE      jz     short loc_100101D7
    .text:100101D0      push    offset aCmd_execC ; "\cmd.exe /c "
    .text:100101D5      jmp    short loc_100101DC
    .text:100101D7
    .text:100101D7
    • .text:100101D7
    .text:100101DC      Up     w sub_10001656+22      mov     dword_1008E5C4, eax
    .text:100101DC      Up     r sub_10007312+E      cmp     dword_1008E5C4, edi
    • .text:100101DC      r sub_1000FF58+270      cmp     dword_1008E5C4, ebx
    .text:100101E2
    .text:100101E3
    .text:100101E8
    .text:100101E9
    .text:100101EF
    .text:100101F0
    • .text:100101F5
    .text:100101F6
  
```

xrefs to dword_1008E5C4 Dialog:

Dire	T	Address	Text
Up	w	sub_10001656+22	mov dword_1008E5C4, eax
Up	r	sub_10007312+E	cmp dword_1008E5C4, edi
	r	sub_1000FF58+270	cmp dword_1008E5C4, ebx

Buttons: OK, Cancel, Help, Search

Line 1 of 3

Global variable

- Following xrefs, we can see that the **output of sub_10003695** will be moved directly into dword_1008E5C4

```
.text:1000166F          mov    [esp+688h+hModule], ebx
.text:10001673          call   sub_10003695
.text:10001678          mov    dword_1008E5C4, eax
.text:1000167D          call   sub_100036C3 ; Attributes: bp-based frame
.text:10001682          push   3A98h
.text:10001687          mov    dword_1008E5C4, sub_10003695 proc near
.text:1000168C          call   ds:Sleep
.text:10001692          call   sub_100110FF
.text:10001697          lea    eax, [esp+68]VersionInformation= _OSVERSIONINFOA ptr -94h
.text:1000169E          push   eax
.text:1000169F          push   202h
.text:100016A4          call   ds:WSAStartup
.text:100016AA          cmp    eax, ebx
.text:100016AC          short loc_100044CD
```



Global variable

- Looking at **sub_10003695**, we can find that it compares the **dw platform ID** to the value '2'

```
.text:10003695 VersionInformation= _OSVERSIONINFOA ptr -94h
.text:10003695
.text:10003695
.text:10003695 Sets AL to 1 if the OS
.text:10003695 has a dwPlatformId
.text:10003695 value of 2. This is based
.text:10003695 on whether the previous
.text:10003695 compare statement has
.text:10003695 a 'Zero Flag' (false
.text:10003695 output) or not, so
.text:10003695 essentially just sets AL
.text:10003695 to true or false
.text:100036BE
.text:100036C1
.text:100036C2
.text:100036C2 sub_10003695
.text:100036C2

push    ebp
mov     ebp, esp
sub     esp, 94h
lea     eax, [ebp+VersionInformation]
mov     [ebp+VersionInformation.dwOSVersionInfoSize], 94h
push    eax           ; lpVersionInformation
call    ds:GetVersionExA ; Get extended information about the
                        ; version of the operating system
xor    eax, eax
cmp    [ebp+VersionInformation.dwPlatformId], 2
setz    al
leave
ret
endp
```



Global variable

- The malware takes a different path, depending on the OS
- **Which OS** triggers the malware?
- The flag is one of the strings listed in the first column

Fields

MacOSX	6	The operating system is Macintosh. This value was returned by Silverlight. On .NET Core, its replacement is Unix.
Unix	4	The operating system is Unix.
Win32NT	2	The operating system is Windows NT or later.
Win32S	0	The operating system is Win32s. This value is no longer in use.
Win32Windows	1	The operating system is Windows 95 or Windows 98. This value is no longer in use.
WinCE	3	The operating system is Windows CE. This value is no longer in use.
Xbox	5	The development platform is Xbox 360. This value is no longer in use.

<https://docs.microsoft.com/en-us/dotnet/api/system.platformid>



Extra tasks (for later study)

- A few hundred lines into the subroutine at 0x1000FF58, a series of comparisons use memcmp to compare strings. What happens if the string comparison to **robotwork** is successful (when memcmp returns 0)?
- What does the export **PSLIST** do?
- Use the graph mode to graph the cross-references from sub_10004E79. Which API functions could be called by entering this function? Based on the API functions alone, what could you rename this function?
- How many Windows API functions does DllMain call directly? How many at a depth of 2?
- At 0x10001358, there is a call to Sleep (an API function that takes one parameter containing the number of milliseconds to sleep). Looking backward through the code, how long will the program sleep if this code executes?
- At 0x10001701 is a call to **socket**. What are the three parameters?

Extra tasks (for later study)

- Using the MSDN page for socket and the named symbolic constants functionality in IDA Pro, can you make the parameters more meaningful? What are the parameters after you apply changes?
- Search for usage of the in instruction (opcode 0xED). This instruction is used with a **magic string VMXh to perform VMware detection**. Is that in use in this malware? Using the cross-references to the function that executes the in instruction, is there further evidence of VMware detection?
- Jump your cursor to 0x1001D988. What do you find?
- If you have the IDA Python plug-in installed, run **Lab05-01.py**, an IDA Pro Python script provided with the malware. (Make sure the cursor is at 0x1001D988.) What happens after you run the script?
- With the cursor in the same location, how do you turn this data into a single ASCII string?
- Open the script with a text editor. How does it work?



Windows malware

- Analyze the malware found in the file **Lab07-01.exe**
 - Questions:
 1. How does this program ensure that it continues running (achieves persistence) when the computer is restarted?
 2. Why does this program use a mutex?
 3. What is a good host-based signature to use for detecting this program?
 4. What is a good network-based signature for detecting this malware?
 5. What is the purpose of this program?
 6. When will this program finish executing?



Persistence

- Main function of the program

```
; int __cdecl main(int argc,const char **argv,const char *envp)
_main          proc near             ; CODE XREF: start+AF↓p

ServiceStartTable= SERVICE_TABLE_ENTRYA ptr -10h
var_8           = dword ptr -8
var_4           = dword ptr -4
argc            = dword ptr 4
argv            = dword ptr 8
envp            = dword ptr 0Ch

sub    esp, 10h
lea    eax, [esp+10h+ServiceStartTable]
mov    [esp+10h+ServiceStartTable.lpServiceName], offset aMalservice ; "MalService"
push   eax           ; lpServiceStartTable
mov    [esp+14h+ServiceStartTable.lpServiceProc], offset sub_401040
mov    [esp+14h+var_8], 0
mov    [esp+14h+var_4], 0
call   ds:StartServiceCtrlDispatcherA
push   0
push   0
call   sub_401040
add   esp, 18h
ret
endp
```



Persistence

- References to Service Control Manager

```

00401000  sub    ds:OpenSCManagerA ; Establish a connection to the service
00401000  sub    ; control manager on the specified computer
00401000  sub    ; and opens the specified database
00401000  mov    esi, eax
00401000  call   ds:GetCurrentProcess
00401000  lea    eax, [esp+404h+BinaryPathName]
00401000  push   3E8h           ; nSize
00401000  push   eax            ; lpFilename
00401000  push   0              ; hModule
00401000  call   ds:GetModuleFileNameA
00401000  push   0              ; lpPassword
00401000  push   0              ; lpServiceStartName
00401000  push   0              ; lpDependencies
00401000  push   0              ; lpdwTagId
00401000  lea    ecx, [esp+414h+BinaryPathName]
00401000  push   0              ; lpLoadOrderGroup
00401000  push   ecx            ; lpBinaryPathName
00401000  push   0              ; dwErrorControl
00401000  push   2              ; dwStartType
00401000  push   10h             ; dwServiceType
00401000  push   2              ; dwDesiredAccess
00401000  push   offset DisplayName ; "Malservice"
00401000  push   offset DisplayName ; "Malservice"
00401000  push   esi            ; hSCManager
00401000  call   ds>CreateServiceA
00401000  xor    edx, edx
00401000  lea    eax, [esp+404h+DueTime]
00401000  mov    dword ptr [esp+404h+SystemTime.wYear], edx
00401000  lea    ecx, [esp+404h+SystemTime]
00401000  mov    dword ptr [esp+404h+SystemTime.wDayOfWeek], edx
00401000  push   eax            ; lpFileTime
00401000  mov    dword ptr [esp+408h+SystemTime.wHour], edx
00401000  push   ecx            ; lpSystemTime
00401000  mov    dword ptr [esp+40Ch+SystemTime.wSecond], edx
00401000  mov    [esp+40Ch+SystemTime.wYear], 834h
00401000  call   ds:SystemTimeToFileTime

```



Mutex

```

sub_401040 proc near
    SystemTime= SYSTEMTIME ptr -400h
    DueTime= LARGE_INTEGER ptr -3F0h
    BinaryPathName= byte ptr -3E8h

    sub esp, 400h
    push offset Name ; "HGL345"
    push 0 ; bInheritHandle
    push 1F0001h ; dwDesiredAccess
    call ds:OpenMutexA
    test eax, eax
    jz short loc_401064

loc_401064:
    push esi
    push offset Name ; "HGL345"
    push 0 ; bInitialOwner
    push 0 ; lpMutexAttributes
    call ds>CreateMutexA
    push 3 ; dwDesiredAccess
    push 0 ; lpDatabaseName
    push 0 ; lpMachineName
    call ds:OpenSCManagerA ; Establish a connection to the service
                           ; control manager on the specified computer
                           ; and opens the specified database
    mov esi, eax
    call ds:GetCurrentProcess
    lea eax, [esp+404h+BinaryPathName]
    push 3E8h ; nSize
    push eax ; lpFilename

```



Network-based signature

```
; DWORD __stdcall StartAddress(LPUVOID)
StartAddress proc near
push    esi
push    edi
push    0          ; dwFlags
push    0          ; lpszProxyBypass
push    0          ; lpszProxy
push    1          ; dwAccessType
push    offset szAgent ; "Internet Explorer 8.0"
call    ds:InternetOpenA
mov     edi, ds:InternetOpenUrlA
mov     esi, eax
ret
```



```
loc_40116D:           ; dwContext
push    0
push    80000000h      ; dwFlags
push    0              ; dwHeadersLength
push    0              ; lpszHeaders
push    offset szUrl  ; "http://www.malwareanalysisbook.com"
push    esi            ; hInternet
call    edi            ; InternetOpenUrlA
jmp    short loc_40116D
StartAddress endp
```

Purpose

```

push    eax          ; lpFileTime
mov     dword ptr [esp+408h+SystemTime.wHour], edx
push    ecx          ; lpSystemTime
mov     dword ptr [esp+40Ch+SystemTime.wSecond], edx
mov     dword ptr [esp+40Ch+SystemTime.wYear], 2100
call    ds:SystemTimeToFileTime
push    0             ; lpTimerName
push    0             ; bManualReset
push    0             ; lpTimerAttributes
call    ds>CreateWaitableTimerA
push    0             ; fResume
push    0             ; lpArgToCompletionRoutine
push    0             ; pfnCompletionRoutine
lea     edx, [esp+410h+DueTime]
mov     esi, eax
push    0             ; lPeriod
push    edx          ; lpDueTime
push    esi          ; hTimer
call    ds:SetWaitableTimer
push    0FFFFFFFh      ; dwMilliseconds
push    esi          ; hHandle
call    ds:WaitForSingleObject
test   eax, eax
jnz    short loc_40113B
push    edi
mov     edi, ds>CreateThread
mov     esi, 20

loc_401126:           ; CODE XREF: sub_401040+F8↓j
push    0             ; lpThreadId
push    0             ; dwCreationFlags
push    0             ; lpParameter
push    offset StartAddress ; lpStartAddress
push    0             ; dwStackSize
push    0             ; lpThreadAttributes
call    edi ; CreateThread
dec    esi
jnz    short loc_401126
pop    edi

```

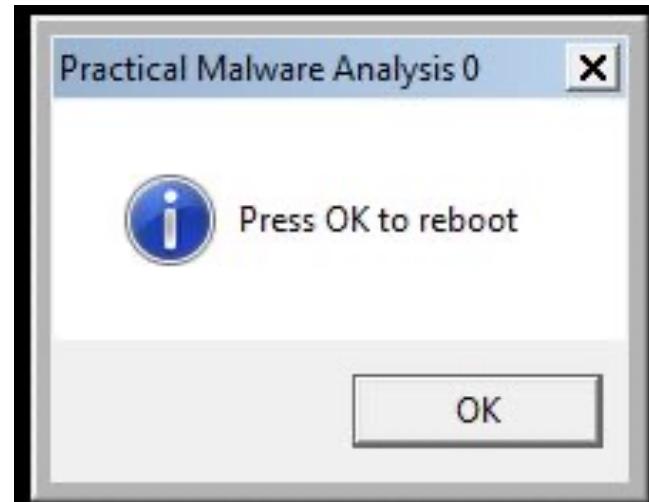
Process Injection

- Analyze the malware found in the file ***Lab12-01.exe*** and ***Lab12-01.dll***
- Make sure that these files are **in the same directory** when performing the analysis
- Questions:
 1. What happens when you **run the malware executable?**
 2. What **process** is being **injected?**
 3. How can you make the malware **stop the pop-ups?**
 4. How does this malware **operate?**



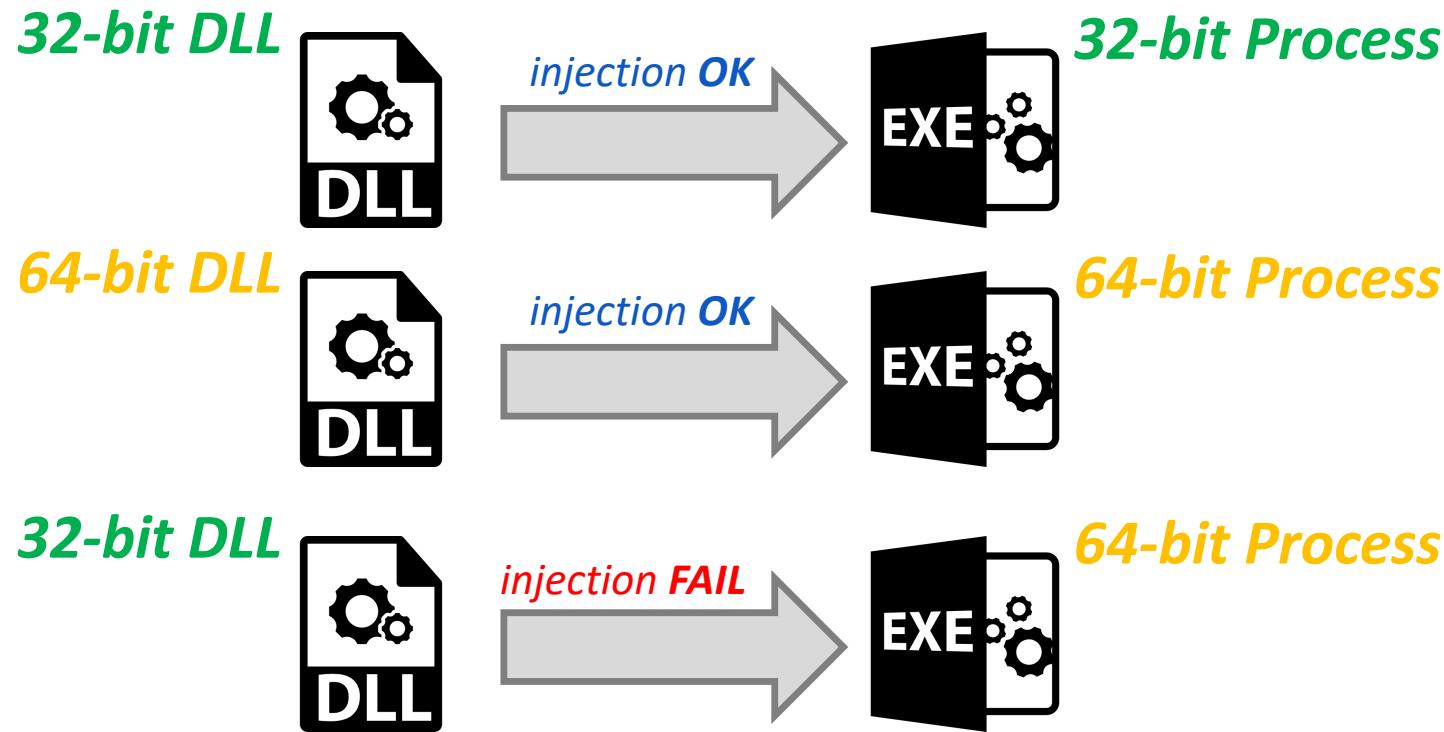
Process Injection

- If injection succeeds, the injected DLL periodically opens a **popup**



Process Injection

- **Lab12-01.exe** is unable to inject the DLL in **64-bit systems**

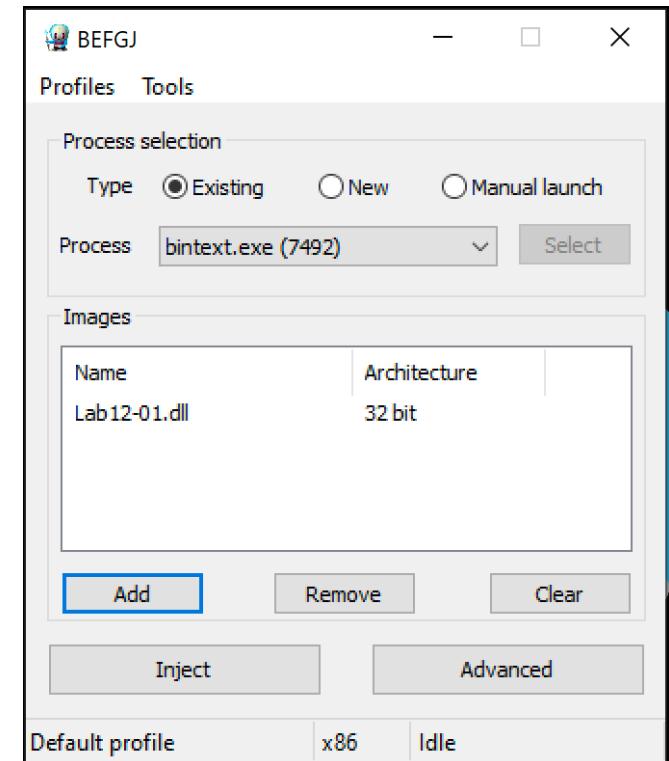


Process Injection

- You can "run" the DLL with:

rundll32.exe Lab12-01.dll,DLLMain

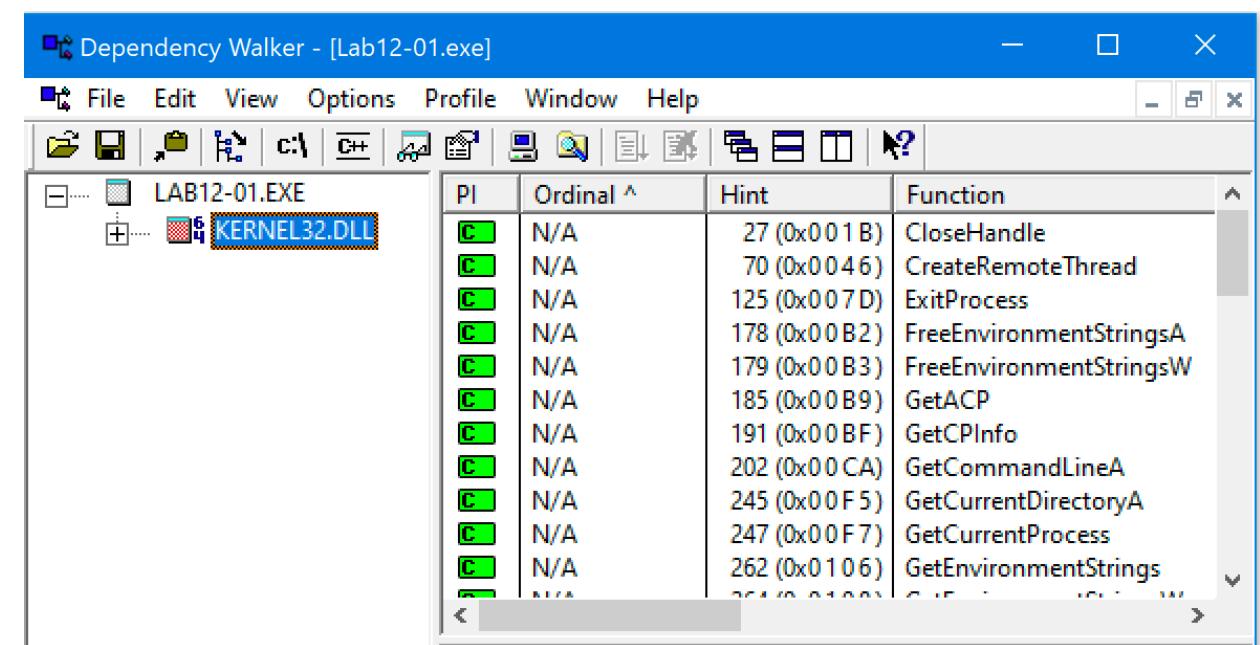
- You can use a tool (e.g., Xenos) to inject the DLL into **another 32-bit process** (e.g., BinText)



<https://github.com/DarthTon/Xenos/releases/>

Imports

- Have a look at imports by the EXE and DLL
- Can you find any evidence of **process injection**?
- Can you find any evidence of the popup opened by the DLL?

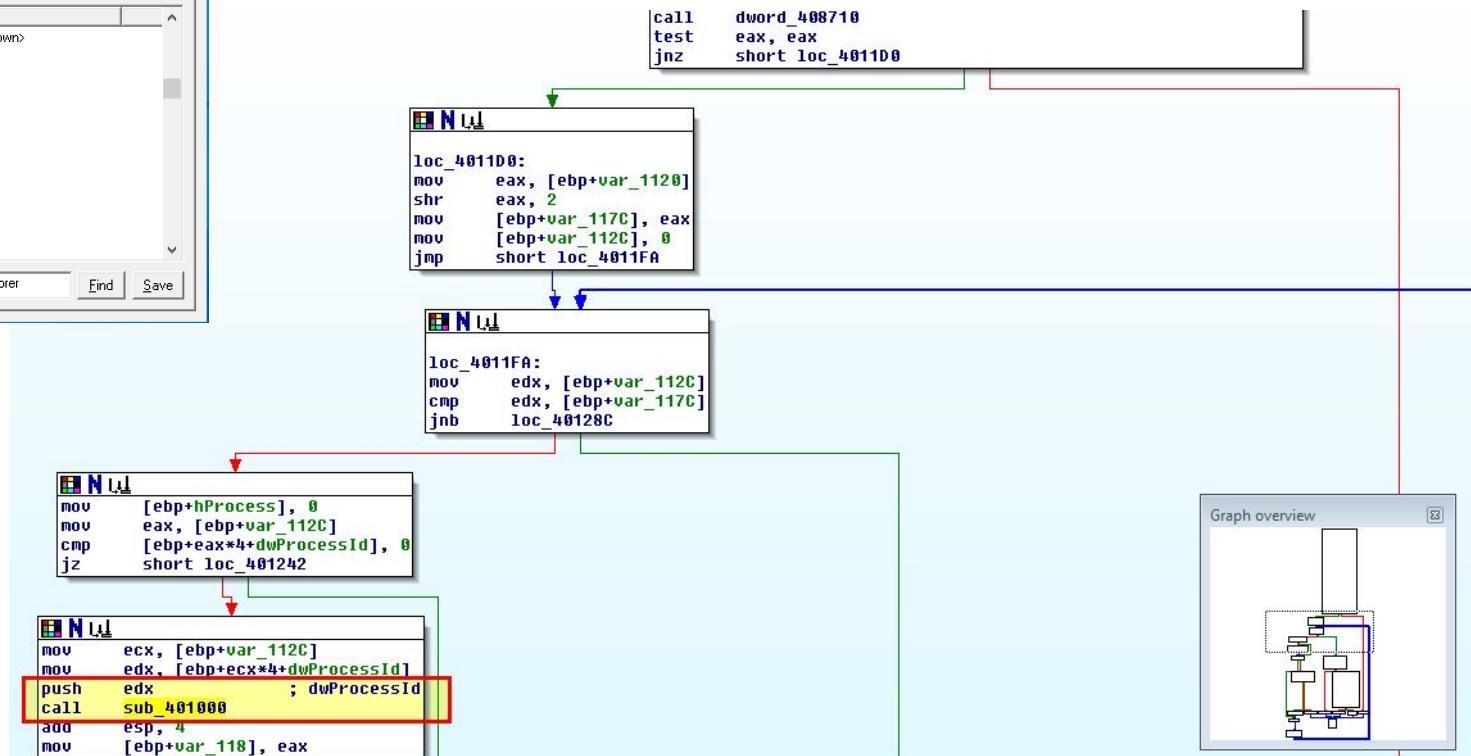
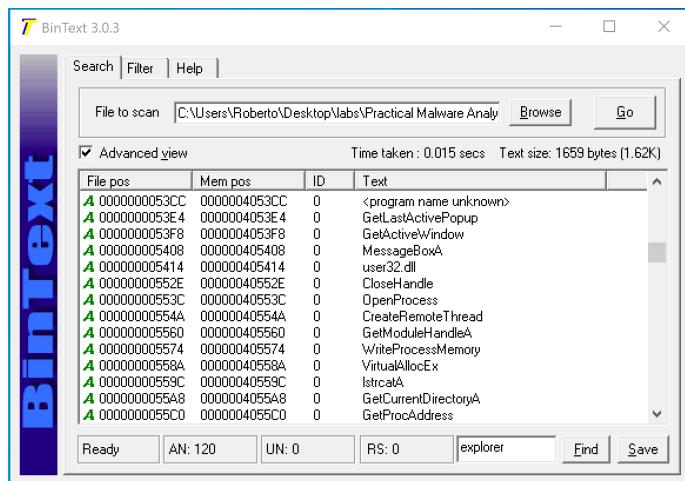


The screenshot shows the Dependency Walker application interface. The title bar reads "Dependency Walker - [Lab12-01.exe]". The menu bar includes File, Edit, View, Options, Profile, Window, and Help. The toolbar contains various icons for file operations. The left pane displays a tree view of imports: "LAB12-01.EXE" has one child node, "KERNEL32.DLL". The right pane is a table showing the imports from KERNEL32.DLL:

PI	Ordinal ^	Hint	Function
C	N/A	27 (0x001B)	CloseHandle
C	N/A	70 (0x0046)	CreateRemoteThread
C	N/A	125 (0x007D)	ExitProcess
C	N/A	178 (0x00B2)	FreeEnvironmentStringsA
C	N/A	179 (0x00B3)	FreeEnvironmentStringsW
C	N/A	185 (0x00B9)	GetACP
C	N/A	191 (0x00BF)	GetCPLinfo
C	N/A	202 (0x00CA)	GetCommandLineA
C	N/A	245 (0x00F5)	GetCurrentDirectoryA
C	N/A	247 (0x00F7)	GetCurrentProcess
C	N/A	262 (0x0106)	GetEnvironmentStrings
C	N/A	264 (0x0108)	GetFileInformationByHandle

Injected process

- Have a look at **strings** and **disassembler** to find the injected process



Injected process

```

mov    eax, [ebp+dwProcessId]
push   eax          ; dwProcessId
push   0             ; bInheritHandle
push   410h          ; dwDesiredAccess
call   ds:OpenProcess
mov    [ebp+hObject], eax
cmp    [ebp+hObject], 0
jz    short loc_401095

; Function 1
lea    ecx, [ebp+var_110]
push   ecx
push   4
lea    edx, [ebp+var_10C]
push   edx
mov    eax, [ebp+hObject]
push   eax
call  dword_408714
test  eax, eax
jz    short loc_401095

; Function 2
push  104h
lea   ecx, [ebp+var_108]
push  ecx
mov  edx, [ebp+var_10C]
push  edx
mov  eax, [ebp+hObject]
push  eax
call dword_40870C

; Function 3
loc_401095:      ; size_t
push  8Ch
push  0Ch
lea   ecx, [ebp+var_108]
push  ecx          ; char *
call  __strnicmp
add   esp, 0Ch
test  eax, eax
jnz  short loc_4010B6

```



Injected process

```

loc_401242:
    cmp    [ebp+var_118], 1
    jnz    short loc_401287

loc_40128C:
; f1Protect
    push   4
    push   3000h ; f1AllocationType
    push   104h ; dwSize
    push   0      ; lpAddress
    mov    edx, [ebp+hProcess]
    push   edx, [ebp+hProcess]
    call   ds:VirtualAllocEx
    mov    [ebp+lpParameter], eax
    cmp    [ebp+lpParameter], 0
    jnz    short loc_4012BE

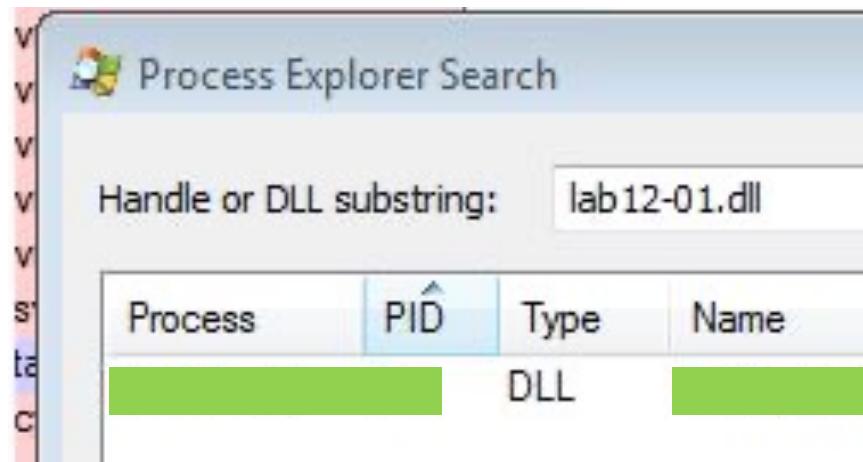
loc_4012BE:
; lpNumberOfBytesWritten
    push   0
    push   104h ; nSize
    lea    eax, [ebp+Buffer]
    push   eax, [ebp+lpParameter]
    push   ecx, [ebp+lpParameter]
    push   ecx, [ebp+lpBaseAddress]
    mov    edx, [ebp+hProcess]
    push   edx, [ebp+hProcess]
    call   ds:WriteProcessMemory
    push   offset ModuleName ; "kernel32.dll"
    call   ds:GetModuleHandleA
    mov    [ebp+hModule], eax
    push   offset aLoadlibraryA ; "LoadLibraryA"
    mov    eax, [ebp+hModule]
    push   eax, [ebp+hModule]
    call   ds:GetProcAddress
    mov    [ebp+lpStartAddress], eax
    push   0      ; lpThreadId
    push   0      ; dwCreationFlags
    mov    ecx, [ebp+lpParameter]
    push   ecx, [ebp+lpParameter]
    mov    edx, [ebp+lpStartAddress]
    push   edx, [ebp+lpStartAddress]
    push   0      ; dwStackSize
    push   0      ; lpThreadAttributes
    mov    eax, [ebp+hProcess]
    push   eax, [ebp+hProcess]
    call   ds:CreateRemoteThread
    mov    [ebp+var_1130], eax

```



Injected process

- We can check the injected process using **process explorer**, to look for any process that has loaded **Lab12-01.dll**
- You can use process explorer to also **kill the malicious process/thread**



How the malware operates

