UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

# Analyzing Windows Malware Practical Labs

Software Security

a.a. 2023/2024

Laurea Magistrale in Ing. Informatica

Roberto Natella

# 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

# Adjusting Graph Mode

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

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

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

- 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

**FLAG 2:**
**Imports**

# 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**

**FLAG 3:**
**Xrefs**

# DNS

- Focusing on the **call to gethostbyname 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

# DNS

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

# 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 "**.praticalmalwareanalysis.com**"
- The flag is covered by a green box

**FLAG 4:**
**Domain name**

# 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?

```
.text:10001656
.text:10001656
.text:10001656                  ; Attributes: info_from_lumina
.text:10001656
.text:10001656                  ; DWORD __stdcall sub_10001656
.text:10001656                  sub_10001656 proc near
.text:10001656
.text:10001656                  var_675= byte ptr -675h
.text:10001656                  var_674= dword ptr -674h
.text:10001656                  hModule= dword ptr -670h
.text:10001656                  timeout= timeval ptr -66Ch
.text:10001656                  name= sockaddr ptr -664h
.text:10001656                  var_654= word ptr -654h
.text:10001656                  in= in_addr ptr -650h
.text:10001656                  Str1= byte ptr -644h
.text:10001656                  var_640= byte ptr -640h
.text:10001                     CommandLine= byte ptr -63Fh
                                Str= byte ptr -63Dh
```

???

???

**FLAG 5:**
**Input parameters**

# 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?

```
000000001001019C 16D4 FF 15 F4 61 01 10 call    ds:GetStartupInfoA
00000000100101A2 16D0 8B 45 F0          mov     eax, [ebp+hWritePipe]
00000000100101A5 16D0 68 00 04 00 00    push    400h            ; uSize
00000000100101AA 16D4 89 45 CC          mov     [ebp+StartupInfo.hStdError], eax
00000000100101AD 16D4 89 45 C8          mov     [ebp+StartupInfo.hStdOutput], eax
00000000100101B0 16D4 8D 85 40 F5 FF FF lea     eax, [ebp+CommandLine]
00000000100101B6 16D4 66 89 5D BC       mov     [ebp+StartupInfo.wShowWindow], bx
00000000100101BA 16D4 50                push    eax             ; lpBuffer
00000000100101BB 16D8 C7 45 B8 01 01 00+mov     [ebp+StartupInfo.dwFlags], 101h
00000000100101C2 16D8 FF 15 D0 61 01 10 call    ds:GetSystemDirectoryA
00000000100101C8 16D0 39 1D C4 E5 08 10 cmp     dword_1008E5C4, ebx
00000000100101CE 16D0 74 07             jz      short loc_100101D7

00000000100101D0 16D0 68 34 5B 09 10    push    offset aCmdExeC ; "\\cmd.exe /c "
00000000100101D5 16D4 EB 05             jmp     short loc_100101DC

00000000100101D7
00000000100101D7                        loc_100101D7:
00000000100101D7 16D0 68 20 5B 09 10    push    offset aCommandExeC ; "\\command.exe /c "

00000000100101DC
00000000100101DC                        loc_100101DC:
00000000100101DC 16D4 8D 85 40 F5 FF FF lea     eax, [ebp+CommandLine]
00000000100101E2 16D4 50                push    eax             ; Dest
00000000100101E3 16D8 E8 B8 4D 00 00    call    strcat
00000000100101E8 16D8 59                pop     ecx
00000000100101E9 16D4 8D 85 40 FA FF FF lea     eax, [ebp+Dst]
00000000100101EF 16D4 59                pop     ecx

.00% (340,3806) (1087,164) 0000F5D0 00000000100101D0: sub_1000FF58+278 (Synchronized with Hex View-1)
```

# 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

# 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

# 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
.text:10001682    push    3A98h
.text:10001687    mov     dword_1008E5
.text:1000168C    call    ds:Sleep
.text:10001692    call    sub_100110FF
.text:10001697    lea     eax, [esp+68
.text:1000169E    push    eax
.text:1000169F    push    202h
.text:100016A4    call    ds:WSAStartu
.text:100016AA    cmp     eax, ebx
```

```
; Attributes: bp-based frame

sub_10003695    proc near               ; CODE XREF: sub_10001656+1D↑p
                                        ; sub_10003B75+7↓p ...

VersionInformation= _OSVERSIONINFOA ptr -94h

                push    ebp
                mov     ebp, esp
                sub     esp, 94h
```

# 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                 push    ebp
.text:10003696                 mov     ebp, esp
.text:10003698                 sub     esp, 94h
.text:1000369E                 lea     eax, [ebp+VersionInformation]
.text:100036A4                 mov     [ebp+VersionInformation.dwOSVersionInfoSize], 94h
.text:100036AA                 push    eax             ; lpVersionInformation
.text:100036AB                 call    ds:GetVersionExA ; Get extended information about the
.text:100036AB                                         ; version of the operating system
.text:100036B1                 xor     eax, eax
.text:100036B3                 cmp     [ebp+VersionInformation.dwPlatformId], 2
.text:100036BE                 setz    al
.text:100036C1                 leave
.text:100036C2                 retn
.text:100036C2 sub_10003695    endp
.text:100036C2
```

Sets AL to 1 if the OS has a dwPlatformId value of 2. This is based on whether the previous compare statement has a 'Zero Flag' (false output) or not, so essentially just sets AL to true or false

# 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

**FLAG 7:**
**Global var**

# 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
                retn
_main           endp
```

# Persistence

- References to Service Control Manager

```
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
push    0                       ; hModule
call    ds:GetModuleFileNameA
push    0                       ; lpPassword
push    0                       ; lpServiceStartName
push    0                       ; lpDependencies
push    0                       ; lpdwTagId
lea     ecx, [esp+414h+BinaryPathName]
push    0                       ; lpLoadOrderGroup
push    ecx                     ; lpBinaryPathName
push    0                       ; dwErrorControl
push    2                       ; dwStartType
push    10h                     ; dwServiceType
push    2                       ; dwDesiredAccess
push    offset DisplayName ; "Malservice"
push    offset DisplayName ; "Malservice"
push    esi                     ; hSCManager
call    ds:CreateServiceA
xor     edx, edx
lea     eax, [esp+404h+DueTime]
mov     dword ptr [esp+404h+SystemTime.wYear], edx
lea     ecx, [esp+404h+SystemTime]
mov     dword ptr [esp+404h+SystemTime.wDayOfWeek], edx
push    eax                     ; lpFileTime
mov     dword ptr [esp+408h+SystemTime.wHour], edx
push    ecx                     ; lpSystemTime
mov     dword ptr [esp+40Ch+SystemTime.wSecond], edx
mov     [esp+40Ch+SystemTime.wYear], 834h
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
```

```
push    0                   ; uExitCode
call    ds:ExitProcess
```

```
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(LPVOID)
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
```

```
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     [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    0FFFFFFFFh      ; 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?

# 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
```

```
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
```

```
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
```

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

# Injected process

```
IDA View-A

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

mov     eax, [ebp+var_112C]
mov     ecx, [ebp+eax*4+dwProcessId]
push    ecx             ; dwProcessId
push    0               ; bInheritHandle
push    430h            ; dwDesiredAccess
call    ds:OpenProcess
mov     [ebp+hProcess], eax
cmp     [ebp+hProcess], 0FFFFFFFFh
jnz     short loc_40127D

loc_40128C:             ; flProtect
push    4
push    3000h           ; flAllocationType
push    104h            ; dwSize
push    0               ; lpAddress
mov     edx, [ebp+hProcess]
push    edx             ; 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             ; lpBuffer
mov     ecx, [ebp+lpParameter]
push    ecx             ; lpBaseAddress
mov     edx, [ebp+hProcess]
push    edx             ; 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             ; hModule
call    ds:GetProcAddress
mov     [ebp+lpStartAddress], eax
push    0               ; lpThreadId
push    0               ; dwCreationFlags
mov     ecx, [ebp+lpParameter]
push    ecx             ; lpParameter
mov     edx, [ebp+lpStartAddress]
push    edx             ; lpStartAddress
push    0               ; dwStackSize
push    0               ; lpThreadAttributes
mov     eax, [ebp+hProcess]
push    eax             ; 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

```
; Attributes: bp-based frame

; DWORD __stdcall sub_10001030(LPVOID)
sub_10001030 proc near

var_18= dword ptr -18h
Parameter= byte ptr -14h

push    ebp
mov     ebp, esp
sub     esp, 18h
mov     [ebp+var_18], 0
```

```
loc_1000103D:
mov     eax, 1
test    eax, eax
jz      short loc_10001088
```

```
mov     ecx, [ebp+var_18]
push    ecx
push    offset aPracticalMalwa ; "Practical Malware Analysis %d"
lea     edx, [ebp+Parameter]
push    edx             ; char *
call    sprintf
add     esp, 0Ch
push    0               ; lpThreadId
push    0               ; dwCreationFlags
lea     eax, [ebp+Parameter]
push    eax             ; lpParameter
push    offset StartAddress ; lpStartAddress
push    0               ; dwStackSize
push    0               ; lpThreadAttributes
call    ds:CreateThread
push    60000           ; dwMilliseconds
call    ds:Sleep
mov     ecx, [ebp+var_18]
add     ecx, 1
mov     [ebp+var_18], ecx
jmp     short loc_1000103D
```

```
loc_10001088:
mov     eax, 1
mov     esp, ebp
pop     ebp
retn    4
sub_10001030 endp
```