Malware Behavior and Anti-Reverse Engineering

Instructor
Davide Maiorca

Web Security and Malware Analysis

M.Sc. in Computer Engineering, Cybersecurity and Artificial Intelligence

University of Cagliari, Italy

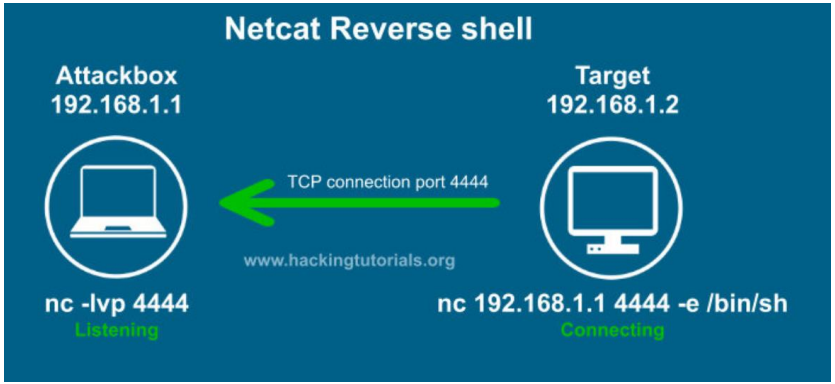
Malware Behavior – Downloaders and Launchers

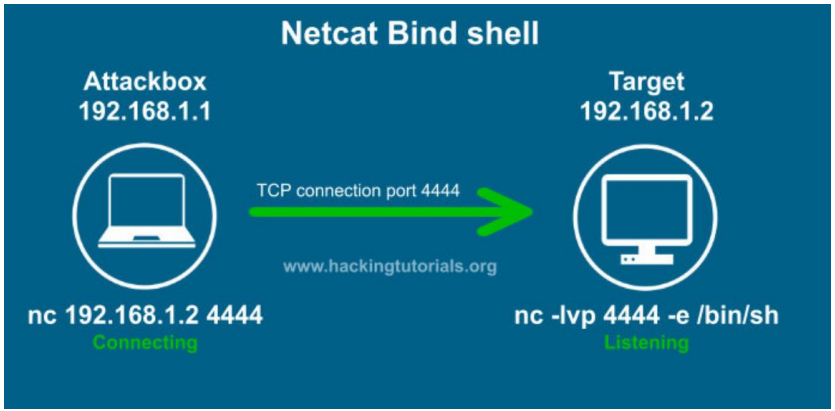
- Malware can be organized in categories, depending on how the actions are performed
- Two popular categories are **downloaders** and **launchers**
- **Downloaders**
  - Drops a file from the internet, which is subsequently installed
  - Resorts to the **DownloadToFileA** API to download the file, followed by a call to **WinExec**
- **Launcher**
  - A malicious sample that is devised to load another sample in memory

Backdoors

- **Malware** that opens a direct connection between the attacker and the victim
  - By typically using port 80 or other custom ports
- Multiple functionalities
  - Manipulation of Registry Keys
  - Directory Creation
  - File Search
- Backdoors often contain **reverse shells**
  - Shells that spawn inside the victim machine and connect to the **attacker who is listening**
  - Concept that is different from a standard **bind shell**
  - **Bind shells** spawn in the machine of the victim and listen for the attacker
- Reverse shells are often generated through the **CreateProcess API**
  - The API is called on the process cmd.exe
  - A socket is created, and a remote connection is established
  - Stdin and stdout are bound to the socket

Reverse Shells vs Bind Shells (an Example with Netcat)





The netcat command is often used
to establish remote client-server connections

It is often used in environments different
from Windows

RATs, Botnets, Credential Stealers

- RAT is the acronym for **Remote Administration Tools**
  - Type of malware that allows for remote management of the victims
  - It typically resorts to ports 80 and 443
  - Typically focuses on one or a few victims

- **Botnets** are a number of infected machines (**zombies**) controlled by a **master (or controller)**
  - Botnets can infect even millions of hosts (differently to RATs, which are focused on few machines)
  - The actions of the master influence all the zombies
  - Used in mass Attacks

- **Credential Stealers** are malicious software that attempts to steal the credentials of the user
  - Some of them wait for the user to login to steal his credentials
  - Others dump the **password hashes** stored
  - Finally, other tools capture the user's keystrokes (**keyloggers**)

Password Hashes

- Windows passwords are typically hidden and encrypted in Windows as **hashes**
- There are essentially three ways to hash windows passwords
- **LM Hashes**
  - Very old technique to hash passwords (based on DES, but weak)
  - Supports at maximum 14 length characters passwords (non case-sensitive)
  - The password is split into two sets of seven characters (which are hashed and joined)
  - Disabled by default
- **NT and NTLM Hashes**
  - **NT** stands for **New Technology**
  - Passwords are first encoded using UTF-16 then hashed with the MD4 algorithm
  - They can be cracked
- The hashes are saved in the Security Account Manager (SAM) file

Retrieving Password Hashes – DLL Injection

- Malware often employs code from free tools whose goal is recovering the password hashes from the SAM
  - Example: **pwdump, pass_the_hash**

- The dump is carried out by performing **DLL Injection** inside the Local Security Authority Subsystem Service (LSASS) process (better known as *lsass.exe*)

- While a DLL is injected, the library can get access to the process space and invoke APIs with the privileges of *lsass.exe*

```
10001119        push    offset LibFileName ; "secur32.dll"
1000111E        call    ds:LoadLibraryA
10001130        push    offset ProcName ; "LsaEnumerateLogonSessions"
10001135        push    esi             ; hModule
10001136        call    ds:GetProcAddress ❶
...
10001670        call    ds:GetSystemDirectoryA
10001676        mov     edi, offset aMsv1_0_dll ; \\msv1_0.dll
...
100016A6        push    eax             ; path to msv1_0.dll
100016A9        call    ds:GetModuleHandleA ❷
```

Listing 11-3: Unique API calls used by a whosthere-alt variant's export function `TestDump`

This is a piece of DLL that is injected into LSASS

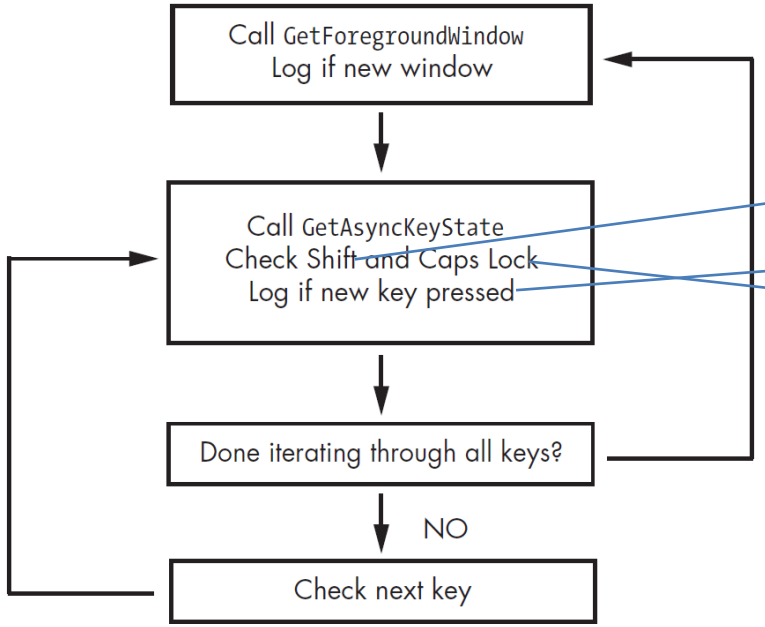
The injected dll employs other DLLs (e.g., secur32.dll, msv1_0.dll, etc.)

The GetProcAddress function is used to retrieve the address of **LsaEnumerate LogonSessions (part of secur32.dll)**

Keyloggers

- Malware samples that record the keys types by the victim
  - They are often installed as keyboard drivers
- Two implementations for keyloggers
  - Hooking
  - Polling
- **Hooking** notifies the malware each time that a key is pressed (by using the SetWindowsHookEx)
- **Polling** works differently by checking the *state* of the key that was pressed
  - **GetAsyncKeyState** is used to check whether a key has been pressed or not after the last call to the function itself
  - It also states if the key is currently pressed or not
  - **GetForegroundWindow** is employed to check which window is being logged

Polling Keyloggers - Example




```
00401162        call    ds:GetForegroundWindow
...
00401272        push    10h ❶                   ; nVirtKey Shift
00401274        call    ds:GetKeyState
0040127A        mov     esi, dword_403308[ebx] ❷
00401280        push    esi                     ; vKey
00401281        movsx   edi, ax
00401284        call    ds:GetAsyncKeyState
0040128A        test    ah, 80h
0040128D        jz      short loc_40130A
0040128F        push    14h                     ; nVirtKey Caps Lock
00401291        call    ds:GetKeyState
...
004013EF        add     ebx, 4 ❸
004013F2        cmp     ebx, 368
004013F8        jl      loc_401272
```

dword_403308 is, in this case, the array that contains the logged keys

Persistence Mechanisms

- Mechanisms with which malicious samples ensure to be reloaded (or to keep their effects) when the system is off, or after a long time

- Multiple ways to achieve persistence
  - Registry modifications
  - Trojanizing binaries
  - Dll *load-order hijacking*
- Registry locations for persistence
  - Some tools (e.g., autoruns program by Sysinternals) automatically look for automatic execution places
- **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run**
  - Already seen in previous lectures
- **AppInit DLLs**
  - Mechanism that loads automatically specific DLLs (in processes that use User32.dll)
  - It should be enabled in the Registry:
    HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows
  - All processes that employ User32.dll may load a specific malicious routine automatically!

Persistence Mechanisms - Registry

- **WinlogonNotify**
  - Automatically loads the malicious sample through login/logout events
  - Events are logged with the **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon**

- **SvcHost DLL**
  - Malware can be installed and used as a service loaded from a DLL (as svchost.exe)
  - Each instance of svchost.exe contains multiple services
  - **Groups are stored in: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost**
  - **Service names are stored in:** HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services*ServiceName*

- Typically, malware adds itself to a group of services, instead of creating new ones

Persistence Mechanisms - Trojanized Binaries

- Malicious sample can patch system binaries or dll to achieve persistence
- Typically, malicious code is added to an empty section of the binary or library
- The entry point of the system binary is patched so that malicious code is executed first
  - And finally, the legitimate code
- Check the hashes of system libraries

Original code	Trojanized code

`DllEntryPoint(HINSTANCE hinstDLL,`
`  DWORD fdwReason, LPVOID lpReserved)`
 	
`mov  edi, edi`
`push  ebp`
`mov  ebp, esp`
`push  ebx`
`mov  ebx, [ebp+8]`
`push  esi`
`mov  esi, [ebp+0Ch]`
	`DllEntryPoint(HINSTANCE hinstDLL,`
	`  DWORD fdwReason, LPVOID lpReserved)`
	
	`jmp  DllEntryPoint_0`

Persistence Mechanisms – DLL Load-Order Hijacking

- Known DLLs are typically stored in a Registry Key
  - HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs

- However, when a DLL is *not known in advance,* the Operating System will look for the DLL file in these paths:
  - The directory where the application is loaded
  - The current directory
  - The system directory (e.g., Windows/System32)
  - The 16-bit system directory (e.g., Windows/System)
  - The Windows directory
  - The directory listed in the Path Environment

- The DLL with a target name that is found *first* is loaded

- This means that the attacker can place a DLL with the same name of target one in an area with higher-priority

Privilege Escalation

- Set of techniques with which malware can obtain administrator privileges without the user's authorization
- Normally, escalation can be done through DLL Load-Order Hijacking if the target process (that loads the DLL) has higher privileges than the ones of the user
- Goal: setting up a special *access token* that contains the security descriptors of a process
- The functions **OpenProcessToken and AdjustTokenPrivileges** can be used to set up the token
- The access token should be set so that the special function **SeDebugPrivilege** can be called
  - This function grants the access to all the system processes (also the administrator ones)
- The escalation procedure is typically complex, but the functions described above are typically good indicators of privilege escalation

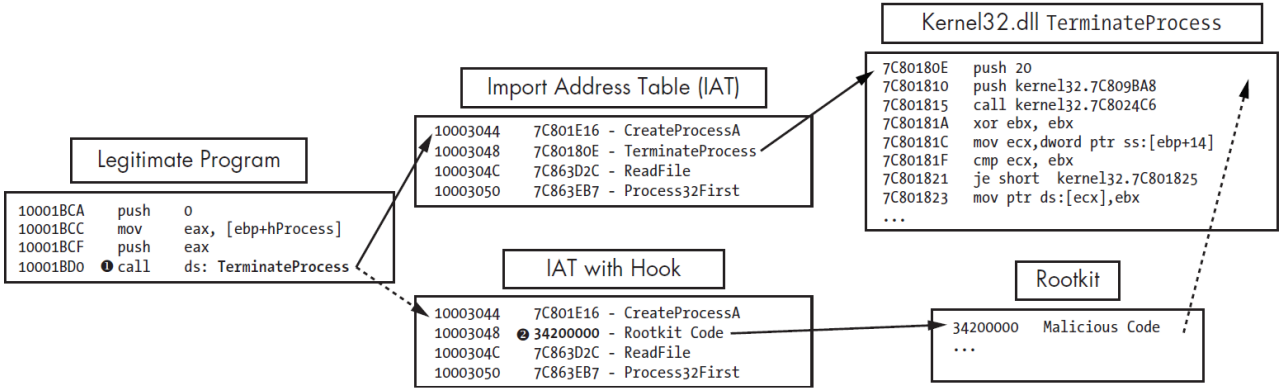
Rootkits

- Malware samples that perform modifications on the Operating System level (often to the kernel)
  - The goal is to hide files, malicious functionalities, etc.
  - They can also work in User Mode

- **IAT Hooking**
  - The Import Address Table of a DLL is patched so that the malicious code is loaded instead of an imported function

- **Inline Hooking**
  - The function code is directly patched with a call to the malicious code (typically through a jump)

Rootkits – IAT Hooking Example



Ransomware

- Family of malware that performs large-scale encryption of the file contained in the victim machine

- A screen with a ransom request is shown, where the malicious sample asks the victim to pay a sum of money (typically in bitcoins) to get the key to unlock the files
  - It is not guaranteed that the key is sent when the ransom is paid

- Ransomware typically employ complex encryption and obfuscation techniques
  - Very long encryption keys (128 bit)
  - Anti-debugging
  - Persistence Mechanisms
  - Privilege Escalation

- Ransomware can also employ Windows API or Windows Powershell to kill certain processes
  - This is done to allow encryption of files that may be kept open by other processes

Anti-Reverse Engineering

- With this term, we point out a set of techniques used by attackers to defy both static and dynamic analysis
- Typically, when we defy static analysis, we refer to **anti-decompilation**
- When we defy dynamic analysis, we refer to **anti-debugging**

Anti-Disassembling – Linear Sweep vs Recursive Traversal

- With the term *linear sweep*, we define disassembling that occurs by following every byte of the original assembly code
  - The bytes are followed as a buffer (no semantic check is performed)
  - Does not make any assumption about the semantic structure of the code
  - **Cannot distinguish between data and code**

- We define as *recursive traversal (or flow oriented)* a disassembly that occurs by semantically following each instruction
  - For example, if a jump is found, the location of the jump is immediately disassembled
  - Much better resilience when data is put between the code lines
  - Has to make choices about the semantics of the application

Anti-Disassembling – Defeating Linear Sweep

```
                test    eax, eax
              ❶jz       short loc_1A
              ❷push     Failed_string
              ❸call     printf
              ❹jmp      short loc_1D
; ---------------------------------------------------------------------------
Failed_string:  db 'Failed',0
; ---------------------------------------------------------------------------
loc_1A: ❺
                xor     eax, eax
loc_1D:
                retn
```

The string “Failed String” is inserted between the code lines (correct disassembly)

```
                test    eax, eax
                jz      short near ptr loc_15+5
                push    Failed_string
                call    printf
                jmp     short loc_15+9
Failed_string:
                inc     esi
                popa
loc_15:
                imul    ebp, [ebp+64h], 0C3C03100h
```

However, the string is interpreted as a bunch of instructions by a linear diassembler!

Anti-Disassembling – Problems in Recursive Traversal

- Recursive traversal is also characterized by problems
- For example, what happens if some semantic assumptions are wrong?
- Typically, recursive-traversal disassemblers resolve **conditions that are false first**
  - In this way, it disassembles the instructions right after the **if** condition
  - And finally, it structures the rest of the disassembling according to its choices
- Imagine a **call** instruction, followed by a string with a specific name (e.g., "hello")
  - The call instruction is equivalent to a **jump**
  - Recursive traversal typically disassembles the bytes **right after** the call and treats them as instructions
  - However, if a byte of the beginning of the string is the same as a legitimate instruction, disassembling may fail!

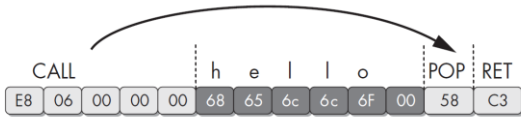


Figure 15-1: call instruction followed by a string

```
E8 06 00 00 00       call    near ptr loc_4011CA+1
68 65 6C 6C 6F     ❶push     6F6C6C65h

                    loc_4011CA:
00 58 C3            add     [eax-3Dh], bl
```

Anti-Disassembly Techniques

- Set of techniques to defeat even recursive traversal analysis or to confuse the analyst

- **Double Jump to the same target**
  - A jz instruction to X location is followed by a jnz instruction to the same location
  - This is equivalent to an unconditional jump!
  - However, the disassembler will disassemble a never-reached instruction that is right after the jumps

- **Jump through a constant condition**
  - A jump is performed by setting a condition that would never change, as it is hardcoded in the assembly code
  - Hence, the conditional jump is again an unconditional jump

- **Rogue Byte (impossible disassembly)**
  - Placing a data byte after a jump so that it destroys the disassembling of the other instructions
  - In the most complex cases, a rogue byte can be part of two instructions

Anti-Disassembly - Jump Examples

```
74 03                jz      short near ptr loc_4011C4+1
75 01                jnz     short near ptr loc_4011C4+1
                     loc_4011C4:                     ; CODE XREF: sub_4011C0
                                                     ; ❷sub_4011C0+2j
E8 58 C3 90 90       ❶call    near ptr 90D0D521h
```

Double jump anti-disassembly

```
33 C0                xor     eax, eax
74 01                jz      short near ptr loc_4011C4+1
       loc_4011C4:                               ; CODE XREF: 004011C2j
                                                 ; DATA XREF: .rdata:004020ACo
E9 58 C3 68 94       jmp     near ptr 94A8D521h
```

Jump with constant condition

Anti-Debugging Techniques

- Set of techniques that the malware uses to defeat dynamic analysis and to discover if there is a debugger attached

- **Using Windows API**
  - It is possible to use some Windows API to scan for the presence of debuggers
  - **IsDebuggerPresent:** checks a structure called **Process Environment Block (PEB)** to find the flag IsDebugged
  - **CheckRemoteDebuggerPresent:** similar to IsDebuggerPresent, but it can be employed to check the status of any executed process (the previous can be used just for the current process)
  - **NtQueryInformationProcess**: employs a function in the ntdll.dll API to query the status of the process (including debugging)
  - **OutputDebugSetting:** sends a string to the debugger to check for its presence

- Windows API can be easily bypassed by the analyst during the analysis

Memory Structures and Behavior Detection

- Instead of using Windows API, **the PEB can be checked directly**
  - A data structure containing multiple flags and arrays related to process information
  - **BeingDebugged:** a special flag that is contained in the PEB structure related to the execution of a debugger in a process
  - **ProcessHeap:** contained inside one of the PEB arrays, it contains a special flag that tells if the heap was created through a debugger
  - **NTGlobalFlag:** special flag that is set up by combining other flags related to the structure of the heap (the debugger creates the heap in a slightly different way)

- Identifying Behavior
  - **Check for INT 3**: software debuggers overwrite the first byte of the debugged instruction. Hence, the malware sample can check for that opcode
  - **Calculating MD5 (or checksum) of a specific portion of code in memory** (often more effective than INT 3 check)
  - **Timing Checks:** calculating the amount of time required to execute specific instructions. The execution under debugger control is significantly slower
- **Some malware can execute TLS callbacks (secret functions called even before the entry point) to check for debugger presence**

Thank you!

THANK YOU FOR ATTENDING THE WEB SECURITY AND MALWARE ANALYSIS COURSE

References and Tools

- M.Sikorski, A.Honig. Practical Malware Analysis, Chapter 11, 15, 16