FIT5124 Advanced Topics in Security

Lecture 9: Malware – Functionality and Analysis Techniques

Ron Steinfeld
Clayton School of IT
Monash University

April 2015

Malware – Functionality and Analysis Techniques

Malware:

Today: A look at malware functionality and techniques for analysing malware.

Plan for this lecture:

- Malware Functionality:
  - Common Malware Function Overview: Backdoors, Credential Stealers, Persistence mechanisms, Covert methods
  - Look at common Covert techniques:
    - Covert Code Execution (Launchers): Process injection, Process hiding
    - Covert Data Interception: Hook injection
- Malware Analysis Techniques and Tools:
  - Malware Behaviour Analysis
  - Malware Code Analysis
  - Anti-analysis techniques

Malware Functionality

Malware comes in various flavours, depending on attacker's goal. We mention a few common types.

Backdoor: Allows attacker to remotely access target machine

- Usually communicate to attacker over HTTP (port 80).
- Often support many OS functions (e.g. enumerate displayed windows, create/open files, ...).
- Other variants:
  - Reverse shell connections: Provide attacker with full shell access to target machine. (e.g. use netcat to remotely run cmd.exe)
  - Remote Administration Tools (RATs), e.g. poisonivy
  - Botnets

Malware Functionality

Credential Stealers:

- Hash dumping (e.g. pwdump)
- keystroke logging:
  - kernel-based keylogging: Modify keyboard driver of OS
  - User-space keylogging: Use Windows API services

Malware Functionality

Common types of Malware Functionality (cont.)

Persistence Mechanisms:

- Modify the Windows Registry (e.g. `HKEY_LOCAL_MACHINE` - global settings section (key) of registry).
- Modify Dynamic Link Libraries (DLLs): add malicious code to empty part of DLL, jump back to original code.

Malware Functionality

Common types of Malware Functionality (cont.)

Covert Techniques:

- ‘Rootkit’ techniques: Hiding existence and actions of attacker code:
  - Process hiding
  - Process injection

Malware Functionality – Covert Techniques

Covert Code Execution: Process Hiding Windows OS background:

- Dynamic Link Libraries (DLLs) contain executable code (like .exe files), but can be shared among processes
- Typical memory map of a Windows process:

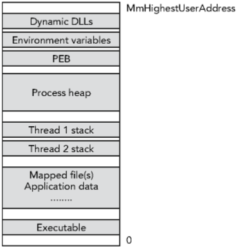


Figure 7-1: A high-level diagram of the typical contents of process memory

- The Process Environment Block (PEB) stores information on the location of items like DLLs, heaps, ...

Malware Functionality – Covert Techniques

Covert Code Execution: Process Hiding

Hiding DLLS via unlinking DLL list:

- The PEB contains 3 linked lists of loaded DLLs
- Standard Windows system calls/utilities (e.g. listdlls) use those lists
- Idea: Attacker unlinks the list to skip entry for attacker's DLL

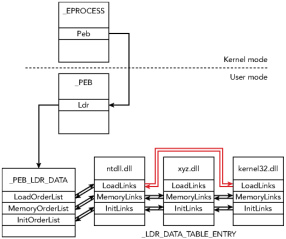


Figure 8-3: A diagram showing how the PEB points to three doubly linked lists of DLLs

Countermeasure: Volatility tool can find trace of unlinked DLL from kernel table. (harder to modify).

Malware Functionality – Covert Techniques

Covert Code Execution: Process Injection

Often, security software (such as Firewalls) blocks access to resources (e.g. Internet access) except from authorized processes.

Q: How can malicious process gain access to blocked resource?

Possible A: Process injection – Malicious process injects code into authorized process.

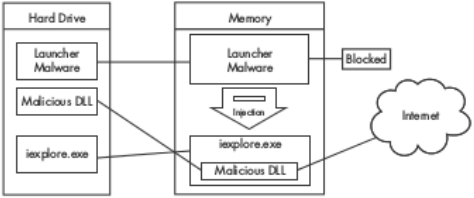


Figure 12-1: DLL injection—the launcher malware cannot access the Internet until it injects into iexplore.exe.

Malware Functionality – Covert Techniques

Covert Code Execution: Process Injection (cont.)

Several known variants of Process Injection:

- DLL injection: malware DLL exists on disk, get target process to load it (e.g. using Windows LoadLibrary API call).
- Direct Injection: Malware code written directly into target process memory and executed within target.
- Reflective DLL injection: Malware DLL written directly into target process memory (no Windows loader API call).
- Process Replacement/Hollowing: Malicious process starts new instance of legit. target process and replaces target code with malware code.

Malware Functionality – Covert Techniques

DLL injection: Malware DLL exists on disk, malware process A gets target process B to run it

Outline of example implementation of process A in Windows:

- Enable debug privilege (`SE_DEBUG_PRIVILEGE`): Gives A right to read and write Process B's memory.
- Opens a handle to process B (`OpenProcess`): Get handle for subsequent process B read/write operations.
- Allocate memory inside Process B for malicious DLL (`VirtualAllocEx`).
- Write path `Malpath` to malicious DLL on disk into Process B (`WriteProcessMemory`).
- Start a new thread in Process B that loads malicious DLL into memory (`CreateRemoteThread`):
  - Pass to CreateRemoteThread ptr to `LoadLibrary` function with argument ptr to `Malpath`.
  - After malicious DLL is loaded, Windows automatically runs its `DllMain` function – malicious code!

Malware Functionality – Covert Techniques

DLL injection: Malware DLL exists on disk, malware process A gets target process B to load it using Windows API call (e.g. LoadLibrary).

Example Windows implementation code for process A:

```
hVictimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, victimProcessID ❶);

pNameInVictimProcess = VirtualAllocEx(hVictimProcess,...,sizeof(maliciousLibraryName),...,...);
WriteProcessMemory(hVictimProcess,...,maliciousLibraryName, sizeof(maliciousLibraryName),...);
GetModuleHandle("Kernel32.dll");
GetProcAddress(...,"LoadLibraryA");
❷ CreateRemoteThread(hVictimProcess,...,...,LoadLibraryAddress,pNameInVictimProcess,...,...);
```

Listing 12-1: C Pseudocode for DLL injection

Malware Functionality – Covert Techniques

Direct Injection: Malware code written directly into target process memory and executed within target.

- Similar implementation to DLL injection, except process A copies malicious code into process B and runs it with `CretateRemoteThread`.

Reflective DLL Injection: Hybrid of DLL and direct injection.

Malware Functionality – Covert Techniques

DLL/Direct Injection is tricky to implement without crashing target process.

Alternative - **Process Replacement/Hollowing:** Malicious process A starts new instance of legit. target process B and replaces target code with malware code.

Outline of example implementation of process A in Windows:

- Create instance of process B in suspended execution mode. (`CreateProcess` with `CREATE_SUSPENDED` argument).
- Release memory used by process B headers/code (`ZwUnmapViewofSection`).
- Allocate above memory in Process B for malicious headers/code (`VirtualAllocEx`).
- Write malicious headers/code into Process B (`WriteProcessMemory`).
- Set start address of suspended process B thread to start of malicious code (`SetThreadContext`).
- Resume suspended thread of process B - run malicious code!

Malware Functionality – Covert Techniques

Process Replacement/Hollowing: Malicious process A starts new instance of legit. target process B and replaces target code with malware code.

Example Windows implementation code for process A:

```
CreateProcess(...,"svchost.exe",...,CREATE_SUSPEND,...);
ZwUnmapViewOfSection(...);
VirtualAllocEx(...,ImageBase,SizeOfImage,...);
WriteProcessMemory(...,headers,...);
for (i=0; i < NumberOfSections; i++) {
 ❶ WriteProcessMemory(...,section,...);
}
SetThreadContext();
...
ResumeThread();
```

Listing 12-3: C pseudocode for process replacement

Malware Functionality – Covert Techniques

Covert Data Interception: Hook injection

Uses Windows hooks to intercept messages from Windows triggered by certain events (e.g. keystrokes).

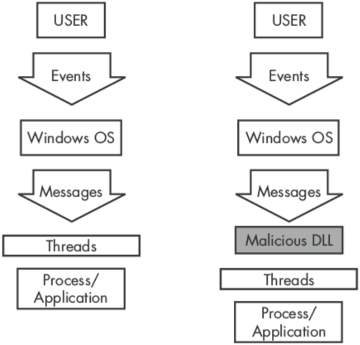


Figure 12-3: Event and message flow in Windows with and without hook injection

Malware Functionality – Covert Techniques

Covert Data Interception: Hook injection Hooks usually implemented in Windows with `SetWindowsHookEx` function Has 4 parameters:

- `idHook`: type of hook procedure, e.g. `WH_CBT` for keyboard/mouse events.
- `lpfn`: ptr to hook procedure.
- `hMod`: handle for DLL containing hook procedure.
- `dwThreadId`: identifier of thread associated with hook (if set to 0, all threads running in same desktop!)

Covert Data Interception: Hook injection

Example SetWindowsHookEx call in Assembly:

```
00401100        push    esi
00401101        push    edi
00401102        push    offset LibFileName ; "hook.dll"
00401107        call    LoadLibraryA
0040110D        mov     esi, eax
0040110F        push    offset ProcName ; "MalwareProc"
00401114        push    esi             ; hModule
00401115        call    GetProcAddress
0040111B        mov     edi, eax
0040111D        call    GetNotepadThreadId
00401122        push    eax             ; dwThreadId
00401123        push    esi             ; hmod
00401124        push    edi             ; lpfn
00401125        push    WH_CBT    ; idHook
00401127        call    SetWindowsHookExA
```

Listing 12-4: Hook injection, assembly code

Malware Analysis – Techniques and Tools

- Behavioural (aka dynamic) analysis: What does the malware do when it runs?
    - Input-output behaviour: system calls by malicious process, files written/read, ...
- Code-based (aka static) analysis: Understand the disassembled/decompiled code

Combination of the two – reverse engineering.
Variety of tools to exist to help in those tasks (brief look).

Malware Analysis – Techniques and Tools

‘Basic’ Static (code) analysis: Scan malware code for system calls / imported DLLs

- Header of executable file (Windows ‘PE’ Header) contains useful information
- Lists DLLs used by executable and functions imported for each DLL
  - Often gives hints on usage: e.g. imported function `SetWindowsHookEx`!
- E.g. useful tool for extracting this info: Dependency Walker (`www.dependencywalker.com`).

Malware Analysis – Techniques and Tools

'Basic' Static (code) analysis (cont.): Scan malware executable file for other clues

Windows executable (PE) file contains several sections:

Table 1-4: Sections of a PE File for a Windows Executable

Executable	Description
`.text`	Contains the executable code
`.rdata`	Holds read-only data that is globally accessible within the program
`.data`	Stores global data accessed throughout the program
`.idata`	Sometimes present and stores the import function information; if this section is not present, the import function information is stored in the `.rdata` section
`.edata`	Sometimes present and stores the export function information; if this section is not present, the export function information is stored in the `.rdata` section
`.pdata`	Present only in 64-bit executables and stores exception-handling information
`.rsrc`	Stores resources needed by the executable
`.reloc`	Contains information for relocation of library files

Tools such as `PEview` and `Resource Hacker` may extract more useful clues

- e.g. strings stored in PE 'resource' section.

Malware Analysis – Techniques and Tools

'Basic' Dynamic (behaviour) analysis: Run malware in a Virtual Machine (VM) and observe its behaviour

Some useful Windows tools:

- `rundll32.exe` (comes with Windows): allows to easily run a (suspected malicious) DLL to observe its behaviour
  - e.g. `rundll32.exe mal.dll Install` runs `Install` function of `mal.dll`.
  - Can get a list of functions exported by DLL using `PEview` tool.

'Basic' Dynamic (behaviour) analysis: Run malware in a Virtual Machine (VM) and observe its behaviour

Some useful Windows tools (cont.): `procmon`: Windows Process Monitor – records process activity

- Registry, File system activity
- Network activity
- Process, thread activity
- Can filter to see only only relevant activity (e.g. interesting process).
- Limitation: Doesn't capture everything, e.g. misses `SetWindowsHookEx` calls.

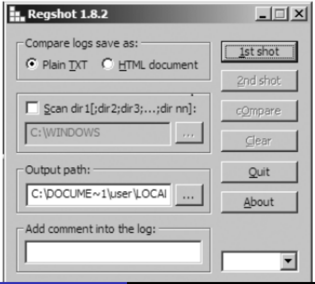
Seq.	Time ...	Process Name	Operation	Path	Result	Detail
200	1:55:31.	mm32.exe	CloseFile	Z:\Malware\mw2mmgr32.dll	SUCCESS	
201	1:55:31.	mm32.exe	ReadFile	Z:\Malware\mw2mmgr32.dll	SUCCESS	Offset: 11,776, Length: 1,024, I/O Flag
202	1:55:31.	mm32.exe	ReadFile	Z:\Malware\mw2mmgr32.dll	SUCCESS	Offset: 12,800, Length: 32,768, I/O Fla
203	1:55:31.	mm32.exe	ReadFile	Z:\Malware\mw2mmgr32.dll	SUCCESS	Offset: 1,024, Length: 9,216, I/O Flags
204	1:55:31.	mm32.exe	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Exec	NAME NOT ...	Desired Access: Read
205	1:55:31.	mm32.exe	ReadFile	Z:\Malware\mw2mmgr32.dll	SUCCESS	Offset: 45,568, Length: 25,088, I/O Fla
206	1:55:31.	mm32.exe	QueryOpen	Z:\Malware\imagehlp.dll	NAME NOT ...	
207	1:55:31.	mm32.exe	QueryOpen	C:\WINDOWS\system32\imagehlp.dll	SUCCESS	CreationTime: 2/28/2006 8:00:00 AM,
208	1:55:31.	mm32.exe	CreateFile	C:\WINDOWS\system32\imagehlp.dll	SUCCESS	Desired Access: Execute/Traverse, S
209	1:55:31.	mm32.exe	CloseFile	C:\WINDOWS\system32\imagehlp.dll	SUCCESS	
210	1:55:31.	mm32.exe	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Exec	NAME NOT ...	Desired Access: Read
211	1:55:31.	mm32.exe	ReadFile	Z:\Malware\mw2mmgr32.dll	SUCCESS	Offset: 10,240, Length: 1,536, I/O Flag
212	1:55:31.	mm32.exe	CreateFile	C:\Documents and Settings\All Users\Application Data\mw2mmgr.txt	SUCCESS	Desired Access: Generic Write, Read
213	1:55:31.	mm32.exe	ReadFile	C:\$Directory	SUCCESS	Offset: 12,288, Length: 4,096, I/O Flag
214	1:55:31.	mm32.exe	CreateFile	Z:\Malware\mm32.exe	SUCCESS	Desired Access: Generic Read, Dispo
215	1:55:31.	mm32.exe	ReadFile	Z:\Malware\mm32.exe	SUCCESS	Offset: 0, Length: 64

Figure 3-2: Procmon mm32.exe *example*

'Basic' Dynamic (behaviour) analysis: Run malware in a Virtual Machine (VM) and observe its behaviour

Some useful Windows tools (cont.):

- `Process Explorer` (Microsoft): Shows processes in a tree structure, DLLs loaded in memory, ...
- `Regshot`: Compare registry and file system state before and after malware running
  - Shows changes to registry made between two snapshots



'Basic' Dynamic (behaviour) analysis: Run malware in a Virtual Machine (VM) and observe its behaviour

Some useful Windows tools (cont.):

- `ApateDNS` (Mandiant): Simulates a DNS server and spoofs a specified response IP address
    - Useful for seeing how malware tries to communicate with external servers (e.g. command and control).
    - Captures malware's DNS requests

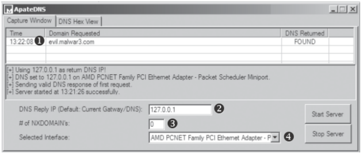


Figure 3-9: ApateDNS responding to a request for evil.malwar3.com

- `netcat`: Simulate a server/client to malware and capture
- `Inetsim`: Simulate many services, e.g. http, https, ftp, dns,...
- `wireshark`: capture network packets from malware to server.

Malware Analysis – Techniques and Tools

'Advanced' Dynamic (behaviour) analysis: Run malware in a debugger within a Virtual Machine (VM) and step through its running code

Some common Windows debugger tools:

- `OllyDbg` (aka `ImmDbg`): Useful debugger for malware analysis
  - Usual debugger facilities: breakpoints, step, etc.
  - Can search for all referenced strings in code (e.g. file name).
  - Can search process memory for a given string
  - Can set memory access breakpoints
- `Windbg`: Can also debug kernel code – device drivers.

Malware Analysis – Techniques and Tools

Anti-Analysis Techniques: Anti-Disassembly

Malware goal: Fool disassembler to output incorrect disassembly

Common anti-disassembly techniques:

- Jump instructions with same target address:
  - Two sequential conditional jumps equivalent to an unconditional jump: `jz addr_x` followed by `jnz addr_x`.
  - Address after `jnz` will never be executed, but disassembler does not realize this
  - Causes incorrect byte alignment for disassembly of following code, e.g:

```
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
```

Fix with `IDA Pro` disassembler: tell disassembler that byte following `jnz` is data byte:

```
74 03               jz      short near ptr loc_4011C5
75 01               jnz     short near ptr loc_4011C5
        ; -------------------------------------------------------------------
E8                  db 0E8h
        ; -------------------------------------------------------------------
                    loc_4011C5:                 ; CODE XREF: sub_4011C0
                                                ; sub_4011C0+2j
58                  pop     eax
C3                  retn
```

Anti-Analysis Techniques: Anti-Disassembly

Malware goal: Confuse the disassembler – incorrect disassembly

Common anti-disassembly techniques (cont.):

- Inward-pointing jump instruction:
    - A 2-byte `jmp` instruction that jumps into its own second byte
    - Second byte of `jmp` is first byte of an INC instruction
    - Causes incorrect byte alignment for disassembly of following code, e.g:

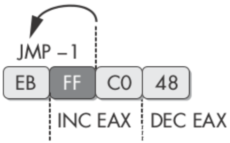


Figure 15-4: Inward-pointing jmp instruction

Fix with `IDA Pro` disassembler: replace 4 bytes with 4 NOP (1 byte) instructions.

Anti-Analysis Techniques: Anti-Debugging
Malware goal: Detect a debugger and alter behaviour
Common anti-debugger techniques:

- Using Windows API functions, e.g.:
  - `IsDebuggerPresent`: direct flag (stored in Process Environment Block – PEB).
  - `OutputDebugString`: indirect – output a string to debugger for display (returns error if no debugger present).
- Manually checking for a debugger, e.g.:
  - `BeingDebugged` flag in PEB: flag stored in Process Environment Block.
  - `ProcessHeap` flag: an undocumented flag within PEB 'reserved' area (tells kernel if heap created by debugger).
  - Searching registry/filesystem for debugger id string (e.g. 'OLLYDBG').
  - Searching own code for software interrupt (debugger breakpoint mechanism) instruction opcode (0xCC).
  - Timing check of computation to detect slowdown due to debugging.