Chapter 12

Covert Malware Launching
Launchers

- Malware that sets itself up for immediate or future covert execution
  - Often contain malware that is to be executed in a resource section
  - Extracts an embedded executable or DLL from resource section and launch it
  - Uses `FindResource`, `LoadResource`, and `SizeofResource` API calls to extract
  - Must be admin privileges/or escalate (identify by privilege escalation code)
Process Injection

- Inject code into another running process
- Most popular covert launching technique
- Bypasses host-based firewalls and process-specific security mechanisms
- Force process to call `VirtualAllocEx` (allocate space in an external process’s memory), then `WriteProcessMemory` to inject code (write data to that allocated space)
- Two injection types: DLL injection, direct injection
DLL Injection

- Force remote process to load a malicious DLL
  - Most common covert loading technique
  - Remotely inject code into process that calls \texttt{LoadLibrary}
  - OS automatically executes \texttt{DllMain} of newly loaded libraries
  - All actions appear to originate from compromised process
DLL Injection

Before

Attacker’s Process
with DLL Injection features and Debug right

Victim Process
DLL for displaying information on screen
Other DLLs

After

Attacker’s Process
with DLL Injection features and Debug right

Victim Process
Attacker’s Code overwrites DLL Code
Other DLLs
**DLL Injection**

- **Method #1**
  1. Obtain handle to victim process (*CreateToolhelp32Snapshot*, *Process32First*, *Process32Next* API calls to search process list for victim)
  2. Get PID of victim and use *OpenProcess* to obtain handle
  3. Allocate space for name of malicious DLL in victim process
     *VirtualAllocEx* allocates space in remote process given handle
  4. Call *WriteProcessMemory* to write string into victim process where *VirtualAllocEx* obtained space
  5. Call *CreateRemoteThread* to start a new thread in victim
     - *lpStartAddress*: starting address of thread (set to address of *LoadLibrary*)
     - *lpParameter*: argument for thread (point to above memory that stores name of malicious DLL)
  - Identify DLL injection based on the above procedures
DLL Injection

- Malware launcher never calls a malicious function – malicious code located in DLLMain – automatically called by the OS
- Goal is to call `CreateRemoteThread`
  - Create remote thread `LoadLibrary`
- See example Fig. 12-2 in Book
  - Look for the victim process name – no string name seen (because they were accessed before code executes)
  - Set a breakpoint to check `WriteProcessMemory` – it dumps the content to the stack
Process Replacement (Hollowing)

- Overwrite memory space of running process with malicious executable
- Disguise malware as legit process without risking crashes from partial injection
- Common attack: `svchost.exe`
  1. Start `svchost` in suspended state
  2. Pass CREATE_SUSPENDED as the `dwCreationFlags` parameter when calling `CreateProcess`
  3. Release all memory using `ZwUnmapViewOfSection`
  4. Allocate memory for malicious code via `VirtualAllocEx`
  5. `WriteProcessMemory` to write malware sections to the victim’s space (in loop)
  6. `SetThreadContext` to fix entry point to point to malicious code
  7. `ResumeThread` to initiate malware
  8. Bypasses firewalls and intrusion prevention systems since `svchost` runs many network daemons
- Users had no idea that the known process is unmapped.
call @System\w\FillChar$Sqq$pvic ; System::_linkproc__FillChar(Void *,Int,char)
mov [ebp\+StartupInfo.cb], 4Ah
lea eax, [ebp\+ProcessInformation]
push eax ; IpProcessInformation
lea eax, [ebp\+StartupInfo]
push eax ; IpStartUpInfo
push 0 ; IpCurrentDirectory
push 0 ; IpEnvironment
push 4 ; dwCreationFlags Process created in suspended state
push 0 ; bInheritHandles
push 0 ; IpThreadAttributes
push 0 ; IpProcessAttributes
mov eax, [ebp\+var_8]
call @System\w\StrToPChar$qrx$17System\w\AnsiString ; System::_linkproc__LStrToPChar(System::AnsiString)
push eax ; IpCommandLine
push 0 ; IpApplicationName
Call CreateProcessA

lea eax, [ebp\+lpAddress]
call sub_45AD34
mov [ebp\+lpContext], eax
cmp [ebp\+lpContext], 0
jz loc_45A5F2

mov eax, [ebp\+lpContext]
mov dword ptr [eax], 10007h
mov eax, [ebp\+lpContext]
push eax ; lpContext
mov eax, [ebp\+ProcessInformation.hThread]
push eax ; hThread
Call GetThreadContext

test eax, eax
jz loc_45AFE2
Hook Injection

- Hooks – handle messages and events going to/from applications and operating system (intercept function calls)

- Use malicious hooks to run certain code whenever a particular message is intercepted (i.e. keystrokes)

- Use malicious hooks to ensure a particular DLL is loaded in a victim's memory space (i.e. process loaded event)

- Calling SetWindowsHookEx to install a hook routine into the hook chain.

- Types of hooks
  - Local hooks: observe and manipulate messages internally within process
  - Remote hooks: observe and manipulate messages destined for a remote process (another process on OS)
Hook Injection Examples

- **Keyboard hooks**
  - Registering hook code using `WH_KEYBOARD` or `WH_KEYBOARD_LL` hook procedure types to implement keyloggers
    - High-level: running in the a remote process or the process install the hook.
    - Low-level: installs a hook that requires the callback to be implemented in your own program.

- **Windows hooks**
  - Register hook with `SetWindowsHookEx` to capture window events

- **Targeting threads**
  - Hooks must determine which thread to attach to
  - Malware implements code to get `dwThreadId` of victim

- **Hook targets often obscure to evade Intrusion Prevention Systems**
  - `WH_CBT` hook for computer-based training messages *(not frequently used)*
  - Call `SetWindowsHookEx` to install hook on remote thread
  - Then, initiate `WH_CBT` message to force load hook.dll by notepad.exe (load into notepad process space and malicious code in DLMMain)
APC Injection

- APC = Asynchronous Procedure Call
- An asynchronous procedure call (APC) is a function that executes asynchronously in the context of a particular thread. When an APC is queued to a thread, the system issues a software interrupt. The next time the thread is scheduled, it will run the APC function.
  - Malware using CreateRemoteThread easily detected (not efficient either)
  - Invoke a function on existing thread (APC)
  - APC allows for a stealthier way to execute code
  - Each thread has an APC function queue attached to it
  - Threads execute all functions in APC queue when in an alertable state after calls to WaitForSingleObjectEx, WaitForMultipleObjectsEx, and SleepEx
  - Malware performs APC injection to preempt threads in an alertable state to get immediate execution of their code

- Two forms
  - Kernel-mode: APC generated for the system or a driver
  - User-mode: APC generated for an application
push [ebp+eax*4+dwThreadId] ; dwThreadId
push 0 ; bInheritHandle
push 1F03FFh ; dwDesiredAccess
call ds:OpenThread
mov esi, eax
mov [ebp+var_2C], esi
test esi, esi
jz short loc_100039E5

push ebx ; dwData
push esi ; hThread
push ds:LoadLibraryA ; pfnAPC
call QueueUserAPC
push esi ; hObject
call ds:CloseHandle

loc_100039E5:
inc [ebp+nSize]
jmp short loc_100039AC