Practical Malware Analysis

Ch 12: Covert Malware Launching

Last revised: 4-10-17
Hiding Malware

- Malware used to be visible in Windows Task Manager
  - But users often know how to look there
- So malware authors now try to blend their malware into the normal Windows landscape
- Covert launching techniques
Launchers
Purpose of a Launcher

• Sets itself or another piece of malware
  • For immediate or future covert execution
• Conceals malicious behavior from the user
• Usually contain the malware they're loading
  – An executable or DLL in its own resource section
• Normal items in the resource section
  – Icons, images, menus, strings
  – Not considered part of the executable
Encryption or Compression

- The resource section may be encrypted or compressed
- Resource extraction will use APIs like
  - \texttt{FindResource}
  - \texttt{LoadResource}
  - \texttt{SizeofResource}
- Malware also often contains privilege escalation code
Process Injection
Process Injection

• The most popular covert launching technique
  • Two types: DLL Injection and Direct Injection
• Injects code into a running process
• Conceals malicious behavior
• May bypass firewalls and other process-specific security mechanisms
• Common API calls:
  – `VirtualAllocEx` to allocate space in another process's memory
  – `WriteProcessMemory` to write to it
DLL Injection

• The most commonly used covert launching technique
• Inject code into a remote process that calls `LoadLibrary`
• Forces the DLL to load in the context of that process
• On load, the OS automatically calls `DLLMain` which contains the malicious code
Example

- Launcher wants Internet access
  - To download more code
- But a process-specific firewall won't let the launcher's process access the Internet
- Solution: inject malicious code into Internet Explorer process
  - Which already has Internet access
Gaining Privileges

• Malware code has the same privileges as the code it is injected into

Figure 13-1. DLL injection—the launcher malware cannot access the Internet until it injects into iexplore.exe.
Example 13-1. C Pseudocode for DLL injection

```c
hVictimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, victimProcessID);
pNameInVictimProcess = VirtualAllocEx(hVictimProcess,...,sizeof(maliciousLibraryName),...,...);
WriteProcessMemory(hVictimProcess,...,maliciousLibraryName, sizeof(maliciousLibraryName),...);
GetModuleHandle("Kernel32.dll");
GetProcAddress(...,"LoadLibraryA");
CreateRemoteThread(hVictimProcess,...,...,LoadLibraryAddress,pNameInVictimProcess,...,...);
```

- **CreateRemoteThread** uses 3 parameters
  - Process handle `hProcess`
  - Starting point `lpStartAddress` (LoadLibrary)
  - Argument `lpParameter` Malicious DLL name
CALL DWORD PTR DS:[<&KERNEL32::OpenProcess>]
MOV DWORD PTR SS:[EBP-1008], EAX
CMP DWORD PTR SS:[EBP-1008], -1
JNZ SHORT DLLinjcc.004076D8
OR EAX, FFFFFFFF
JMP DLLinjcc.0040779D
MOV DWORD PTR SS:[EBP-100C], 7D0
JMP DLLinjcc.00407646
PUSH 4
PUSH 3000
PUSH 104
MOV EAX, DWORD PTR SS:[EBP-1008]
PUSH EAX
CALL DWORD PTR DS:[<&KERNEL32::VirtualAllocEx>]
MOV DWORD PTR SS:[EBP-1010], EAX
CMP DWORD PTR SS:[EBP-1010], 0
JNZ SHORT DLLinjcc.00407719
OR EAX, FFFFFFFF
JMP DLLinjcc.0040779D
PUSH 0
PUSH 719
PUSH 104
PUSH 104
LEA ECX, DWORD PTR SS:[EBP-1180]
PUSH ECX
MOV EDX, DWORD PTR SS:[EBP-1010]
PUSH EDX
MOV EAX, DWORD PTR SS:[EBP-1008]
PUSH EAX
CALL DWORD PTR DS:[<&KERNEL32::WriteProcessMemory>]
PUSH DLLinjcc.0040AACC
CALL DWORD PTR DS:[<&KERNEL32::GetModuleHandleW>]
MOV DWORD PTR SS:[EBP-1188], EAX
PUSH DLLinjcc.0040ACK8
MOV ECX, DWORD PTR SS:[EBP-1188]
PUSH ECX
CALL DWORD PTR DS:[<&KERNEL32::GetProcAddress>]
MOV DWORD PTR SS:[EBP-1190], EAX
PUSH 0
PUSH 0
MOV EDX, DWORD PTR SS:[EBP-1010]
PUSH EDX
MOV EAX, DWORD PTR SS:[EBP-1190]
PUSH EAX
PUSH 0
MOV ECX, DWORD PTR SS:[EBP-1008]
PUSH ECX
CALL DWORD PTR DS:[<&KERNEL32::CreateRemoteThread>]

Figure 13-2. DLL injection debugger view
Analyzing DLL Injection

- Once you find DLL injection activity in disassembly
  - Look for strings containing the name of the malicious DLL and the victim process
  - Or put a breakpoint in the injection code and examine the stack to find them
Direct Injection

• Injects code directly into the remote process
• Without using a DLL
• More flexible than DLL injection
• Requires a lot of customized code
  • To run without negatively impacting the host process
• Difficult to write
Process Replacement
Process Replacement

- Overwrites the memory space of a running object with malicious code
- Disguises malware as a legitimate process
- Avoids risk of crashing a process with process injection
- Malware gains the privileges of the process it replaces
- Commonly replaces `svchost.exe`
Suspended State

- In a *suspended state*, the process is loaded into memory but the primary thread is suspended
  - So malware can overwrite its code before it runs
- This uses the `CREATE_SUSPENDED` value
- in the `dwCreationFlags` parameter
- In a call to the `CreateProcess` function
Example 13-2. Assembly code showing process replacement

00401535  push  edi       ; lpProcessInformation
00401536  push  ecx       ; lpStartupInfo
00401537  push  ebx       ; lpCurrentDirectory
00401538  push  ebx       ; lpEnvironment
00401539  push  CREATE_SUSPENDED ; dwCreationFlags
0040153B  push  ebx       ; bInheritHandles
0040153C  push  ebx       ; lpThreadAttributes
0040153D  lea  edx, [esp+94h+CommandLine]
00401541  push  ebx       ; lpProcessAttributes
00401542  push  edx       ; lpCommandLine
00401543  push  ebx       ; lpApplicationName
00401544  mov  [esp+0A0h+StartupInfo.dwFlags], 101h
0040154F  mov  [esp+0A0h+StartupInfo.wShowWindow], bx
00401557  call  ds:CreateProcessA
Example 13-3. C pseudocode for process replacement

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

- **ZwUnmapViewOfSection** releases all memory pointed to by a section
- **VirtualAllocEx** allocates new memory
- **WriteProcessMemory** puts malware in it
Example 13-3. C pseudocode for process replacement

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

• **SetThreadContext** restores the victim process's environment and sets the entry point

• **ResumeThread** runs the malicious code
Hook Injection
Hooks

• Windows hooks intercept messages destined for applications

• Malicious hooks
  – Ensure that malicious code will run whenever a particular message is intercepted
  – Ensure that a DLL will be loaded in a victim process's memory space
Figure 13-3. Event and message flow in Windows with and without hook injection
Local and Remote Hooks

- *Local hooks* observe or manipulate messages destined for an internal process.
- *Remote hooks* observe or manipulate messages destined for a remote process (another process on the computer).
High-Level and Low-Level Remote Hooks

• *High-level remote hooks*
  – Require that the hook procedure is an exported function contained in a DLL
  – Mapped by the OS into the process space of a hooked thread or all threads

• *Low-level remote hooks*
  – Require that the hook procedure be contained in the process that installed the hook
Keyloggers Using Hooks

- Keystrokes can be captured by high-level or low-level hooks using these procedure types
  - `WH_KEYBOARD`
  - or
  - `WH_KEYBOARD_LL`
Using `SetWindowsHookEx` for Remote Windows Hooking

- **Parameters**
  - `idHook` - type of hook to install
  - `lpfn` - points to hook procedure
  - `hMod` - handle to DLL, or local module, in which the `lpfn` procedure is defined
  - `dwThreadId` - thread to associate the hook with. Zero = all threads

- **The hook procedure must call** `CallNextHookEx` **to pass execution to the next hook procedure so the system continues to run properly**
Thread Targeting

• Loading into all threads can degrade system performance
• May also trigger an IPS
• Keyloggers load into all threads, to get all the keystrokes
• Other malware targets a single thread
• Often targets a Windows message that is rarely used, such as `WH_CBT` (a computer-based training message)
Explanation of Next Slide

- Malicious DLL *hook.dll* is loaded
- Malicious hook procedure address
  *MalwareProc* obtained
- The hook procedure calls only
  *CallNextHookEx*
- A *WH_CBT* message is sent to a Notepad thread
- Forces *hook.dll* to be loaded by Notepad
- It runs in the Notepad process space
Example 13-4. Hook injection, assembly code

```
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
```
Detours
A Microsoft Product

- Detours makes it easy for application developers to modify applications and the OS
- Used in malware to add new DLLs to existing binaries on disk
- Modifies the PE structure to create a `.detour` section
- Containing original PE header with a new import address table
• **setdll** is the Microsoft tool used to point the PE to the new import table

• There are other ways to add a `.detour` section

---

**Figure 13-4. A PEview of Detours and the evil.dll**

<table>
<thead>
<tr>
<th>pFile</th>
<th>Data</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010FA4</td>
<td>0001499E</td>
<td>Hint/Name RVA</td>
<td>01E4 _snwprintf</td>
</tr>
<tr>
<td>00010FA8</td>
<td>000149AC</td>
<td>Hint/Name RVA</td>
<td>0260 exit</td>
</tr>
<tr>
<td>00010FAC</td>
<td>000149B4</td>
<td>Hint/Name RVA</td>
<td>00A8 __acmdln</td>
</tr>
<tr>
<td>00010FB0</td>
<td>000149BE</td>
<td>Hint/Name RVA</td>
<td>006D __getmainargs</td>
</tr>
<tr>
<td>00010FB4</td>
<td>000149CE</td>
<td>Hint/Name RVA</td>
<td>013B __initterm</td>
</tr>
<tr>
<td>00010FB8</td>
<td>000149DA</td>
<td>Hint/Name RVA</td>
<td>009A __setusermatherr</td>
</tr>
<tr>
<td>00010FBC</td>
<td>000149EE</td>
<td>Hint/Name RVA</td>
<td>0036 _adjust_fdiv</td>
</tr>
<tr>
<td>00010FC0</td>
<td>000149FE</td>
<td>Hint/Name RVA</td>
<td>0080 __p_commode</td>
</tr>
<tr>
<td>00010FC4</td>
<td>00014A0E</td>
<td>Hint/Name RVA</td>
<td>0085 __p_fmode</td>
</tr>
<tr>
<td>00010FC8</td>
<td>00014A1C</td>
<td>Hint/Name RVA</td>
<td>0098 __set_spp_type</td>
</tr>
<tr>
<td>00010FCC</td>
<td>00014A2E</td>
<td>Hint/Name RVA</td>
<td>00D6 __controlfp</td>
</tr>
<tr>
<td>00010F0D</td>
<td>00000000</td>
<td>End of Imports</td>
<td>msvcr7.dll</td>
</tr>
<tr>
<td>00010F0F</td>
<td>80000000</td>
<td>Ordinal</td>
<td>0001 evil.dll</td>
</tr>
<tr>
<td>00010F24</td>
<td>00000000</td>
<td>End of Imports</td>
<td>0001 evil.dll</td>
</tr>
</tbody>
</table>
APC Injection
Asynchronous Procedure Call (APC)

• Directs a thread to execute other code prior to executing its regular path
• Every thread has a queue of APCs attached to it
• These are processed when the thread is in an alterable state, such as when these functions are called
  – WaitForSingleObjectEx
  – WaitForMultipleObjectsEx
  – Sleep
Two Forms of APCs

• Kernel-Mode APC
  – Generated for the system or a driver
• User-Mode APC
  – Generated for an application
• APC Injection is used in both cases
APC Injection from User Space

- Uses API function `QueueUserAPC`
- Thread must be in an alterable state
- `WaitForSingleObjectEx` is the most common call in the Windows API
- Many threads are usually in the alterable state
QueueUserAPC Parameters

- **hThread**  handle to thread
- **PFNAPC**  defines the function to run
- **dwData**  parameter for function
• 1: Opens a handle to the thread
• 2: `QueueUserAPC` is called with `pfnAPC` set to `LoadLibraryA` (loads a DLL)
• `dwData` contains the DLL name (`dbnet.dll`)
• `Svchost.exe` is often targeted for APC injection
APC Injection from Kernel Space

• Malware drivers and rootkits often want to execute code in user space
• This is difficult to do
• One method is APC injection to get to user space
• Most often to `svchost.exe`
• Functions used:
  – `KeInitializeApc`
  – `KeInsertQueueApc`
Example 13-6. User-mode APC injection from kernel space

```
000119BD    push    ebx
000119BE    push    1
000119C0    push    [ebp+arg_4]
000119C3    push    ebx
000119C4    push    offset sub_11964
000119C9    push    2
000119CB    push    [ebp+arg_0]
000119CE    push    esi
000119CF    call    ds:KeInitializeApc
000119D5    cmp    edi, ebx
000119D7    jz     short loc_119EA
000119D9    push    ebx
000119DA    push    [ebp+arg_C]
000119DD    push    [ebp+arg_8]
000119E0    push    esi
000119E1    call    edi ;KeInsertQueueApc
```