恶意代码分析与防治技术

第12章 隐蔽执行技术

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2022-2023学年

知识点

- 启动器（Launchers）
- 进程注入技术（Process Injection）
- 进程替换技术（Process Replacement）
- Hook注入技术（Hook Injection）
- Detours技术
- APC注入技术（APC Injection）

Launchers

Purpose of a Launcher

- Sets itself or another piece of malware
- For immediate or future covert execution
- Conceals malicious behavior from the user

Purpose of a Launcher

- Usually contain the malware they're loading
- An executable or DLL in its own resource section or PE overlay.
  - Normal items in the resource section
  - Icons, images, menus, strings

Encryption or Compression

- The resource section may be encrypted or compressed (without the second PE header)
- Resource extraction will use APIs like
  - **FindResource**
  - **LoadResource**
  - **SizeofResource**
- Often contains privilege escalation code

Process Injection

Process Injection

- The most popular covert launching process
- Inject code into a running process
  - Conceals malicious behavior
  - May bypass firewalls and other process-specific security mechanisms
- Common API calls:
  - **VirtualAllocEx** to allocate space
  - **WriteProcessMemory** to write to it

Process Injection

- DLL注入（DLL Injection）
- 直接注入（Direct Injection）

DLL Injection

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

Gaining Privileges

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

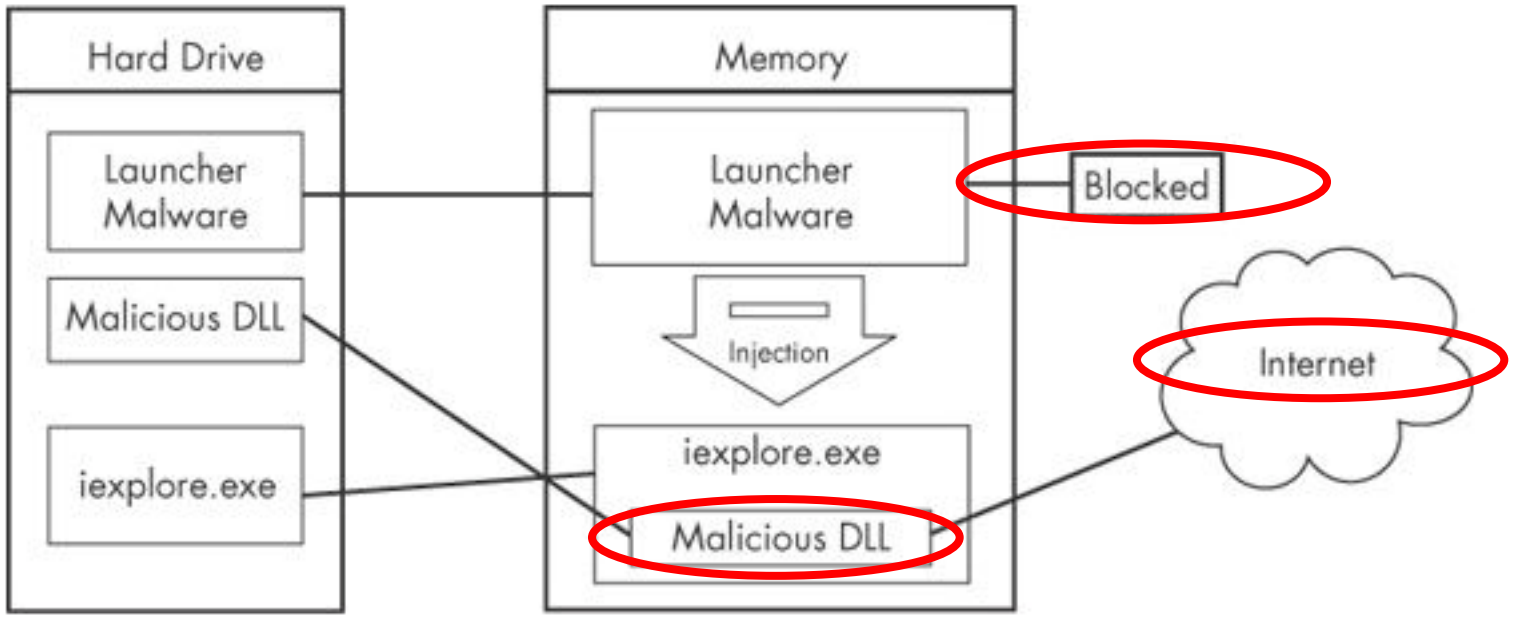


Figure 13-1. DLL injection—the launcher malware cannot access the Internet until it injects into iexplore.exe.

Search Process

- Search for the injection target
  - **CreateToolhelp32Snapshot**
  - **Process32First**
  - **Process32Next**
- Retrieve the process identification(PID)
- Obtain the handle
  - OpenProcess

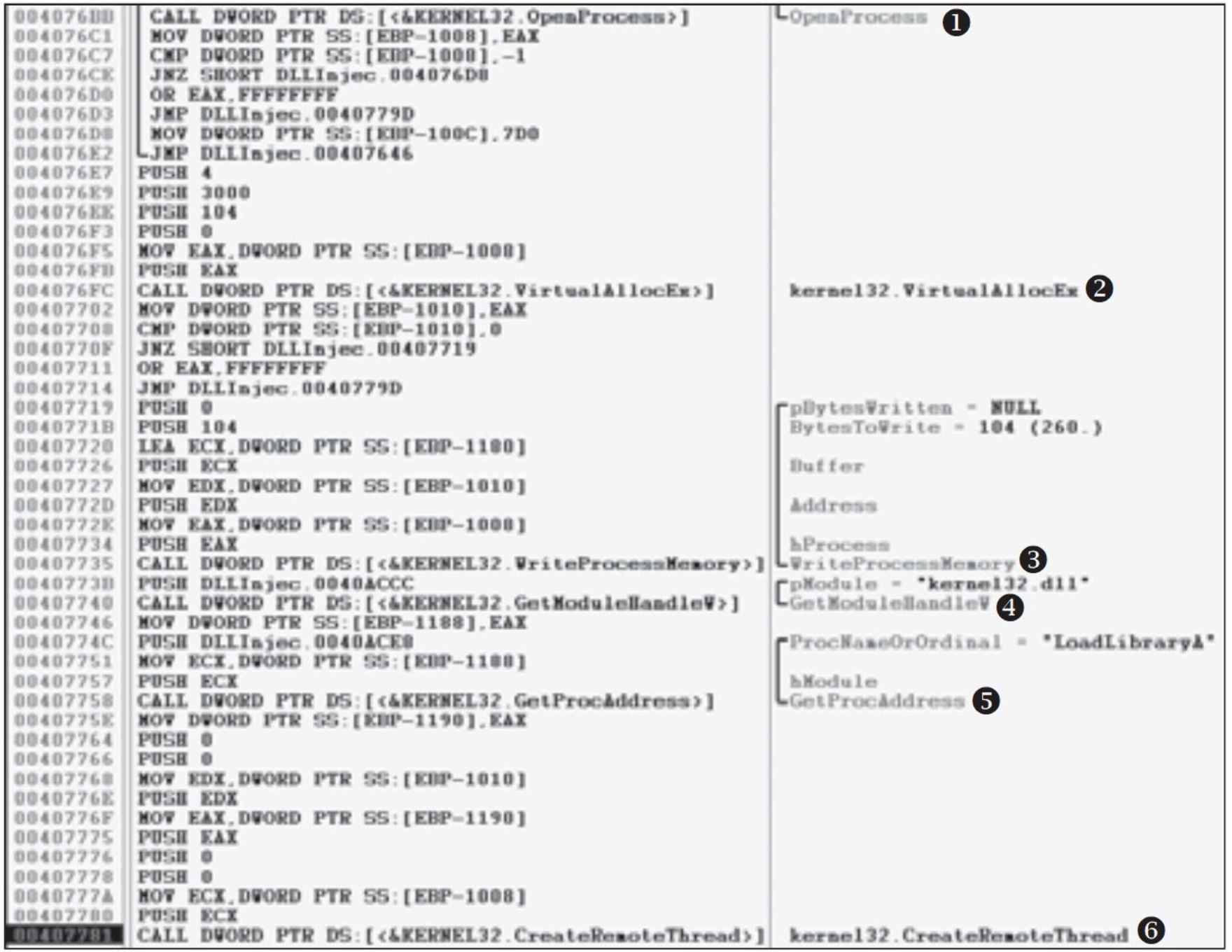
Create Remote Thread

Example 13-1. C Pseudocode for DLL injection

```
  hVictimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, victimProcessID 1);

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

- **CreateRemoteThread** uses 3 parameters
  - Process handle **hProcess**
  - Starting point **lpStartAddress** (LoadLibrary)
  - Argument **lpParameter** Malicious DLL name
    - VirtualAllocEx, WriteProcessMemory



DLL Injection

- For malware analysts
  - Find the victim process name
  - Find the malicious DLL name
  - Recognize injection code pattern

Direct Injection

- Injects code directly into the remote process
  - Without using a DLL
  - Requires a lot of customized code
- Difficult to write without negatively impact to host process
  - shellcode

Direct Injection

- VirtualAllocEx
- WriteProcessMemory
- CreateRemoteThread
- Compiled code
  - LoadLibrary
  - GetProcessAddress

Process Replacement

Process Replacement

- **Replace** the victim process's memory space with malicious executable
- 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

```
CreateProcess(...,"svchost.exe",...,CREATE_SUSPEND,...);
ZwUnmapViewOfSection(...);
VirtualAllocEx(...,ImageBase,SizeOfImage,...);
WriteProcessMemory(...,headers,...);
for (i=0; i < NumberOfSections; i++) {
  1 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++) {
  1 WriteProcessMemory(...,section,...);
}
SetThreadContext();
...
ResumeThread();
```

- **`SetThreadContext`** restores the victim process's environment and sets the entry
- **`ResumeThread`** runs the malicious code

Process Replacement

- Bypass firewalls
- Bypass intrusion prevention systems(IPSs)
- From the process list, see only the original binary's path and known binary executable
  - with no idea that it was replaced.

Hook Injection

SetWindowsHookExA function (winuser.h)

Article • 07/28/2022 • 7 minutes to read

Feedback

Installs an application-defined hook procedure into a hook chain. You would install a hook procedure to monitor the system for certain types of events. These events are associated either with a specific thread or with all threads in the same desktop as the calling thread.

Syntax

C++ Copy

```
HHOOK SetWindowsHookExA(
  [in] int       idHook,
  [in] HOOKPROC  lpfn,
  [in] HINSTANCE hmod,
  [in] DWORD     dwThreadId
);
```

https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexa

SetWindowsHookEx()

Using SetWindowsHookEx() to perform Remote Process Injection

https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexa

```
HHOOK SetWindowsHookExA(
  int       idHook,
  HOOKPROC  lpfn,
  HINSTANCE hmod,
  DWORD     dwThreadId
);
```

- Using a process ID get a thread ID which we want to hook into
  - `GetThreadID()`
- Load the DLL library, and get the address of the exported function you are going to call
  - `LoadLibrary()`
  - `LoadLibraryEx()`
  - `GetProcAddress()`
- Find a Window associated with the process name
  - `FindWindow()`
- Get the Window Thread ID
  - `GetWindowThreadProcessId()`
- Set a Hook into this thread ID so that when the event triggers, our DLL exported function gets called
  - `SetWindowsHookEx()`
- Optionally Unhook
  - `UnhookWindowsHookEx()`

Hook Injection

- Windows是消息驱动的
- Windows hooks are used to intercept messages destined for applications
- Malicious hooks injection
  - 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

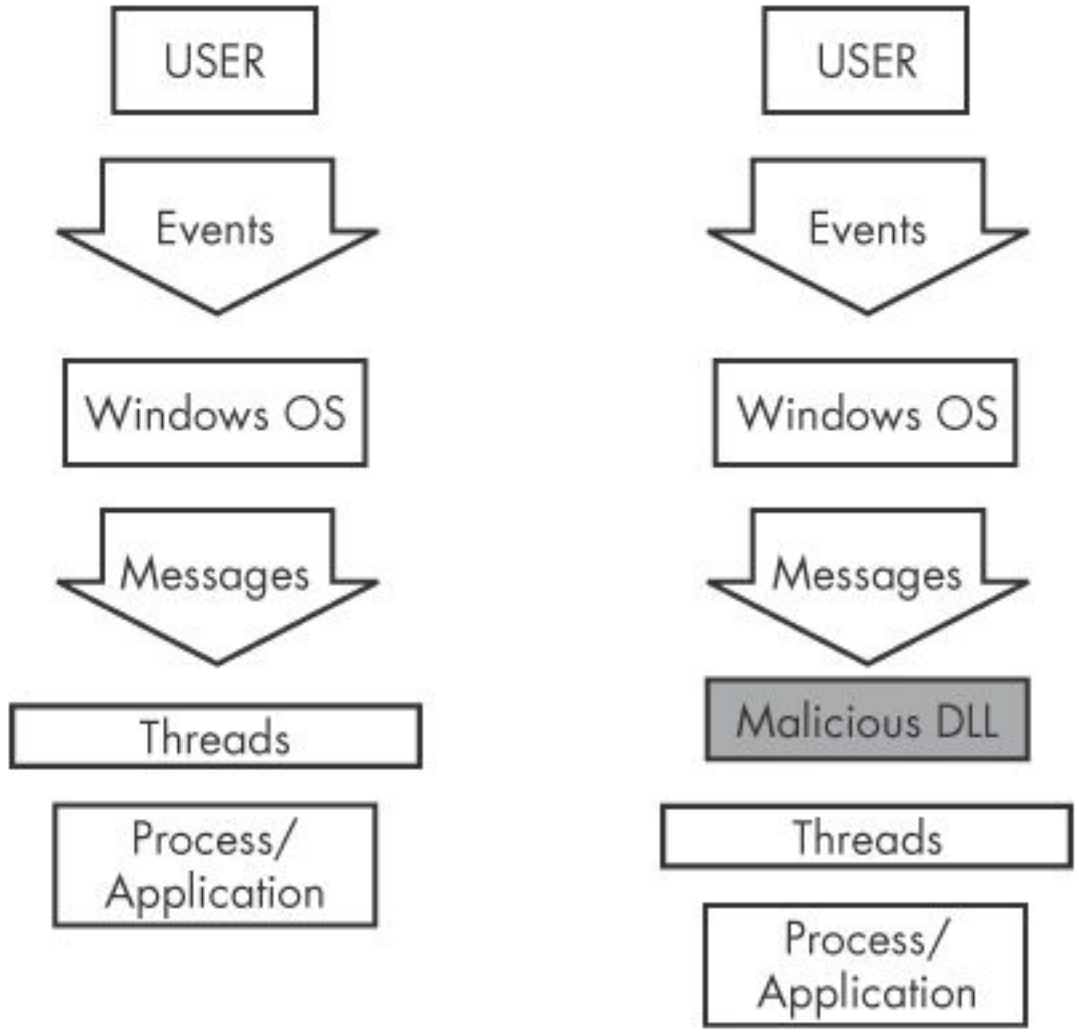


Figure 13-3. Event and message flow in Windows with and without hook injection

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 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**

- 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

WH_KEYBOARD 2	Installs a hook procedure that monitors keystroke messages. For more information, see the KeyboardProc hook procedure.
WH_KEYBOARD_LL 13	Installs a hook procedure that monitors low-level keyboard input events. For more information, see the LowLevelKeyboardProc hook procedure.
WH_MOUSE 7	Installs a hook procedure that monitors mouse messages. For more information, see the MouseProc hook procedure.
WH_MOUSE_LL 14	Installs a hook procedure that monitors low-level mouse input events. For more information, see the LowLevelMouseProc hook procedure.
WH_MSGFILTER -1	Installs a hook procedure that monitors messages generated as a result of an input event in a dialog box, message box, menu, or scroll bar. For more

Using `SetWindowsHookEx`

- Hook chain
- 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)

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
```

Thread Targeting

- Load malicious DLL *hook.dll*
- Obtain hook procedure address
- 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

Detours

Detours

- Detours is a library developed by Microsoft.
  - easily instrument and extend existing OS and application functionality.
- Detours library is used by malware authors
  - modify important tables
  - attach DLLs
  - and function hooks

Detours

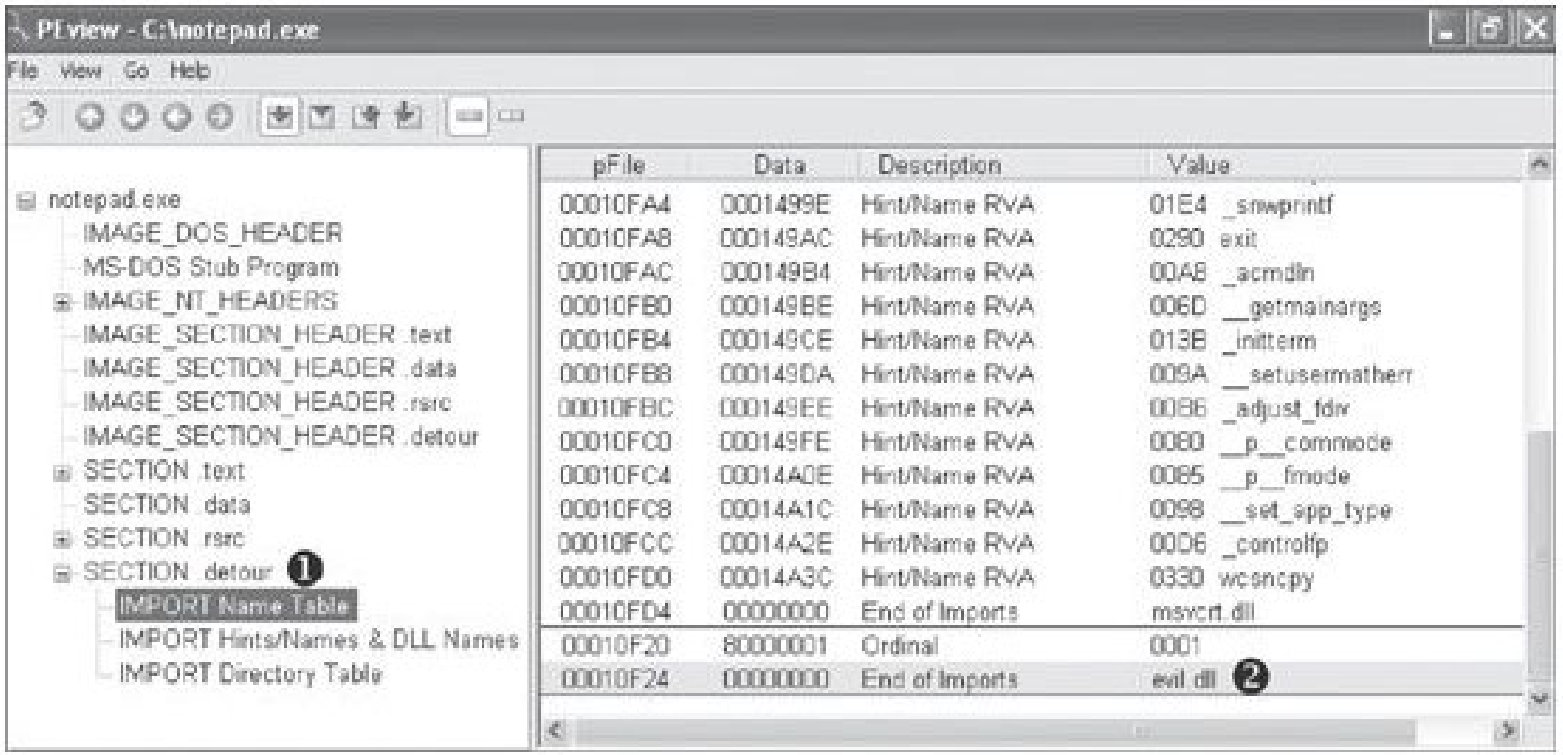


Figure 13-4. A PEview of Detours and the evil.dll

Trojanize notepad.exe

- Create .detours section
- Add a new import table
- Contains evil.dll
- Load evil.dll whenever notepad is launched.

r77 Rootkit

- Ring 3 Rootkit that hides following entities from all processes:
  - Files, directories, junctions, named pipes, scheduled tasks
  - Processes
  - CPU usage
  - Registry keys & values
  - Services
  - TCP & UDP connections
- It is compatible with Windows 7 and Windows 10 in both x64 and x86 editions.
- https://github.com/bytecode77/r77-rootkit

r77 Rootkit

Technical Documentation

r77 Version	1.2.2
Release date	31.08.2021
Author	bytecode77
Website	bytecode77.com/r77-rootkit
GitHub	github.com/bytecode77/r77-rootkit

Detours

4.4 Hooked API's

Detours is the hooking library used to hook functions from `ntdll.dll`. This DLL is loaded into every process on the operating system. It is a wrapper around all syscalls, which makes it the lowest layer available in ring 3. Any WinAPI function from `kernel32.dll` or other libraries and frameworks will ultimately call `ntdll.dll` functions. It is not possible to hook syscalls directly. This is a common limitation to ring 3 rootkits.

Hiding of services exceptionally requires hooking of `advapi32.dll` and `sechost.dll` instead. Please read section 4.4.7 about why this is a requirement.

The following chapters describe each function that is hooked.

- 4.4 Hooked API's
  - 4.4.1 NtQuerySystemInformation
  - 4.4.2 NtResumeThread
  - 4.4.3 NtQueryDirectoryFile
  - 4.4.4 NtQueryDirectoryFileEx
  - 4.4.5 NtEnumerateKey
  - 4.4.6 EnumServiceGroupW
  - 4.4.7 EnumServicesStatusExW
  - 4.4.8 NtEnumerateValueKey
  - 4.4.9 NtDeviceIoControlFile

APC Injection

Asynchronous Procedure Call (APC)

- APCs (*asynchronous procedure call*) can direct a thread to execute some other code prior to executing its regular execution path.

APC Injection

- Every thread has a queue of APCs, and these are processed when the thread is in an alertable state
  - SleepEx, SignalObjectAndWait, MsgWaitForMultipleObjectsEx, WaitForMultipleObjectsEx, WaitForSingleObjectEx

APC Injection

- When in altertable state, the thread calls the APC functions one by one for all APCs in the queue.
- When the APC queue is complete, the thread continues running along its regular execution path.

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 to queue a function to a remote thread
- 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 the victim thread
- **pfnAPC** defines the function to run
- **dwData** parameter for the function

Example 13-5. APC injection from a user-mode application

```
00401DA9        push    [esp+4+dwThreadId]      ; dwThreadId
00401DAD        push    0                       ; bInheritHandle
00401DAF        push    10h                     ; dwDesiredAccess
00401DB1        call    ds:OpenThread 1
00401DB7        mov     esi, eax
00401DB9        test    esi, esi
00401DBB        jz      short loc_401DCE
00401DBD        push    [esp+4+dwData]          ; dwData = dbnet.dll
00401DC1        push    esi                     ; hThread
00401DC2        push    ds:LoadLibraryA 2       ; pfnAPC
00401DC8        call    ds:QueueUserAPC
```

- Obtain the handle to the victim thread
- **QueueUserAPC** is called with **pfnAPC** set to **LoadLibraryA** (loads a DLL)
- **dwData** contains the DLL name (*dbnet.dll*)
- *Svchost.exe* is often targeted

APC Injection from Kernel Space

- Malware drivers and rootkits often want to execute code in user space
  - 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 1
000119C0        push    [ebp+arg_4] 2
000119C3        push    ebx
000119C4        push    offset sub_11964
000119C9        push    2
000119CB        push    [ebp+arg_0] 3
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
```

知识点

- Launchers
- 进程注入技术（Process Injection）
- 进程替换技术（Process Replacement）
- Hook注入技术（Hook Injection）
  - 难点：local hook、remote hook
- Detours技术
- APC注入技术（APC Injection）

恶意代码分析与防治技术

第12章 隐蔽执行技术

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2022-2023学年