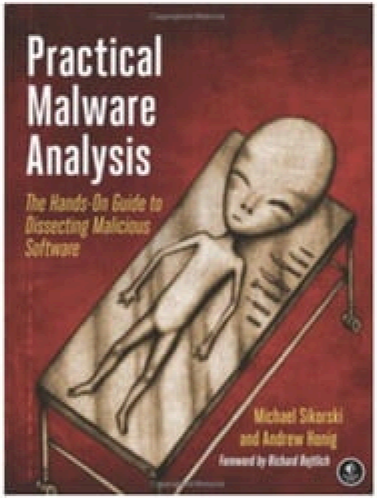
Practical Malware Analysis

Ch 12: Covert Malware Launching


Practical Malware Analysis
The Hands-On Guide to Dissecting Malicious Software
Michael Sikorski and Andrew Honig
Foreword by Richard Bejtlich

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 lanching 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
  - **`FindResource`**
  - **`LoadResource`**
  - **`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

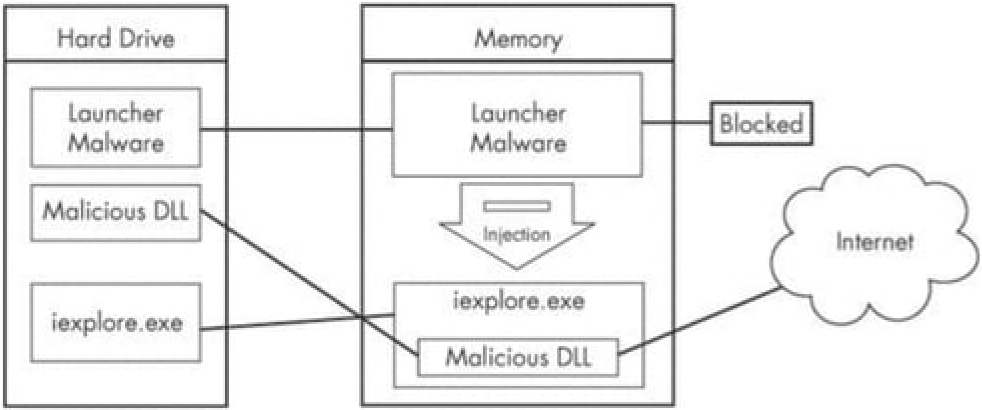
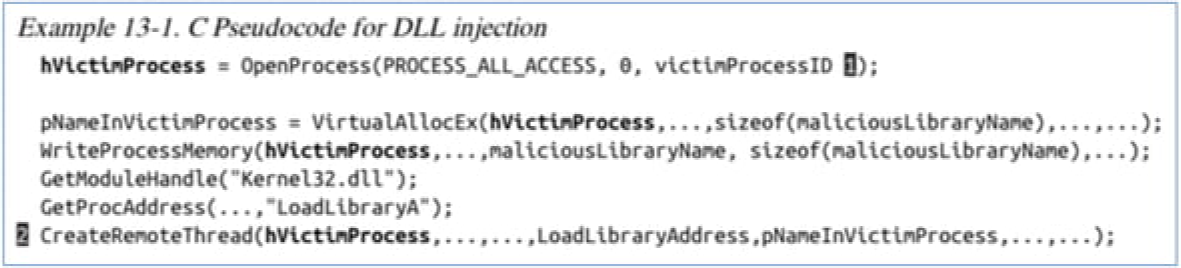


Figure 13-1. DLL injection—the launcher malware cannot access the Internet until it injects into iexplore.exe.

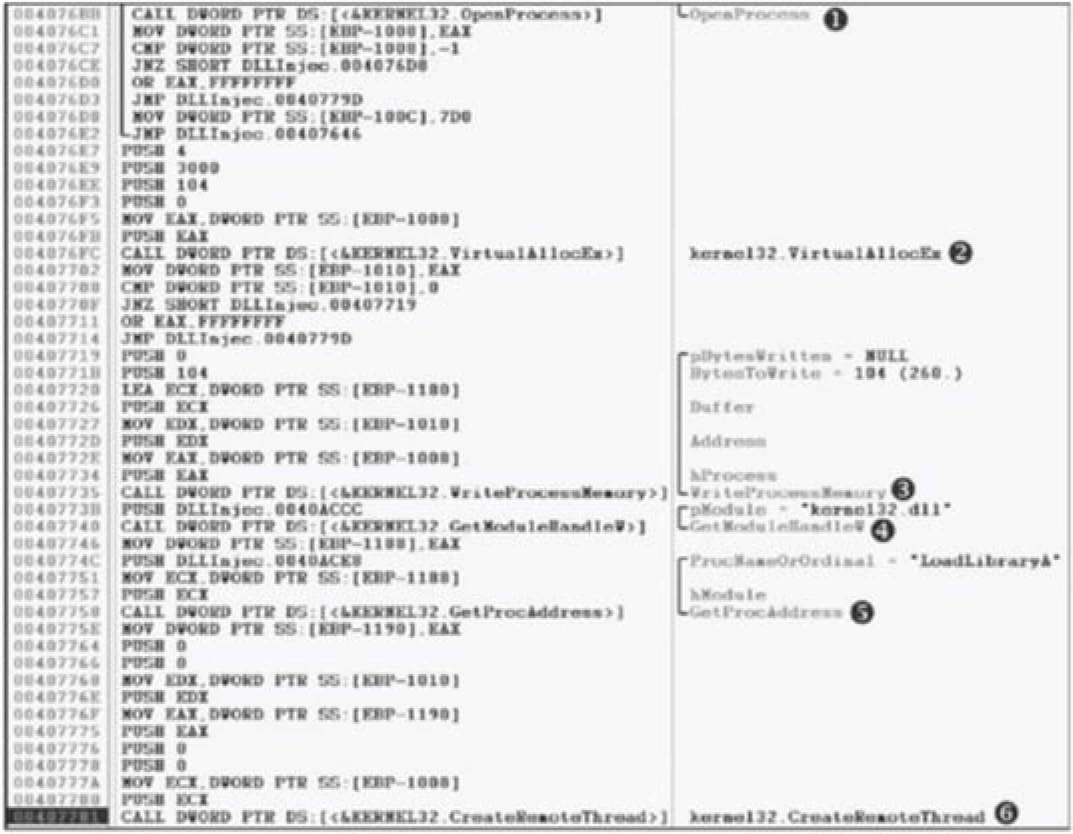


Example 13-1. C Pseudocode for DLL injection

```
  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,...,...);
```

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



```
004076BB   CALL DWORD PTR DS:[<&KERNEL32.OpenProcess>]        └OpenProcess ❶
004076C1   MOV DWORD PTR SS:[EBP-1008],EAX
004076C7   CMP DWORD PTR SS:[EBP-1008],-1
004076CE   JNZ SHORT DLLInjec.004076D8
004076D0   OR EAX,FFFFFFFF
004076D3   JMP DLLInjec.0040779D
004076D8   MOV DWORD PTR SS:[EBP-100C],7D0
004076E2   JMP DLLInjec.00407646
004076E7   PUSH 4
004076E9   PUSH 3000
004076EE   PUSH 104
004076F3   PUSH 0
004076F5   MOV EAX,DWORD PTR SS:[EBP-1008]
004076FB   PUSH EAX
004076FC   CALL DWORD PTR DS:[<&KERNEL32.VirtualAllocEx>]      kernel32.VirtualAllocEx ❷
00407702   MOV DWORD PTR SS:[EBP-1010],EAX
00407708   CMP DWORD PTR SS:[EBP-1010],0
0040770F   JNZ SHORT DLLInjec.00407719
00407711   OR EAX,FFFFFFFF
00407714   JMP DLLInjec.0040779D
00407719   PUSH 0                                              ┌pBytesWritten = NULL
0040771B   PUSH 104                                            │BytesToWrite = 104 (260.)
00407720   LEA ECX,DWORD PTR SS:[EBP-1108]                     │
00407726   PUSH ECX                                            │Buffer
00407727   MOV EDX,DWORD PTR SS:[EBP-1010]                     │
0040772D   PUSH EDX                                            │Address
0040772E   MOV EAX,DWORD PTR SS:[EBP-1008]                     │
00407734   PUSH EAX                                            │hProcess
00407735   CALL DWORD PTR DS:[<&KERNEL32.WriteProcessMemory>]  └WriteProcessMemory ❸
0040773B   PUSH DLLInjec.0040ACCC                              ┌pModule = "kernel32.dll"
00407740   CALL DWORD PTR DS:[<&KERNEL32.GetModuleHandleW>]    └GetModuleHandleW ❹
00407746   MOV DWORD PTR SS:[EBP-1188],EAX
0040774C   PUSH DLLInjec.0040ACE8                              ┌ProcNameOrOrdinal = "LoadLibraryA"
00407751   MOV ECX,DWORD PTR SS:[EBP-1188]                     │
00407757   PUSH ECX                                            │hModule
00407758   CALL DWORD PTR DS:[<&KERNEL32.GetProcAddress>]      └GetProcAddress ❺
0040775E   MOV DWORD PTR SS:[EBP-1190],EAX
00407764   PUSH 0
00407766   PUSH 0
00407768   MOV EDX,DWORD PTR SS:[EBP-1010]
0040776E   PUSH EDX
0040776F   MOV EAX,DWORD PTR SS:[EBP-1190]
00407775   PUSH EAX
00407776   PUSH 0
00407778   PUSH 0
0040777A   MOV ECX,DWORD PTR SS:[EBP-1008]
00407780   PUSH ECX
00407781   CALL DWORD PTR DS:[<&KERNEL32.CreateRemoteThread>]  kernel32.CreateRemoteThread ❻
```

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

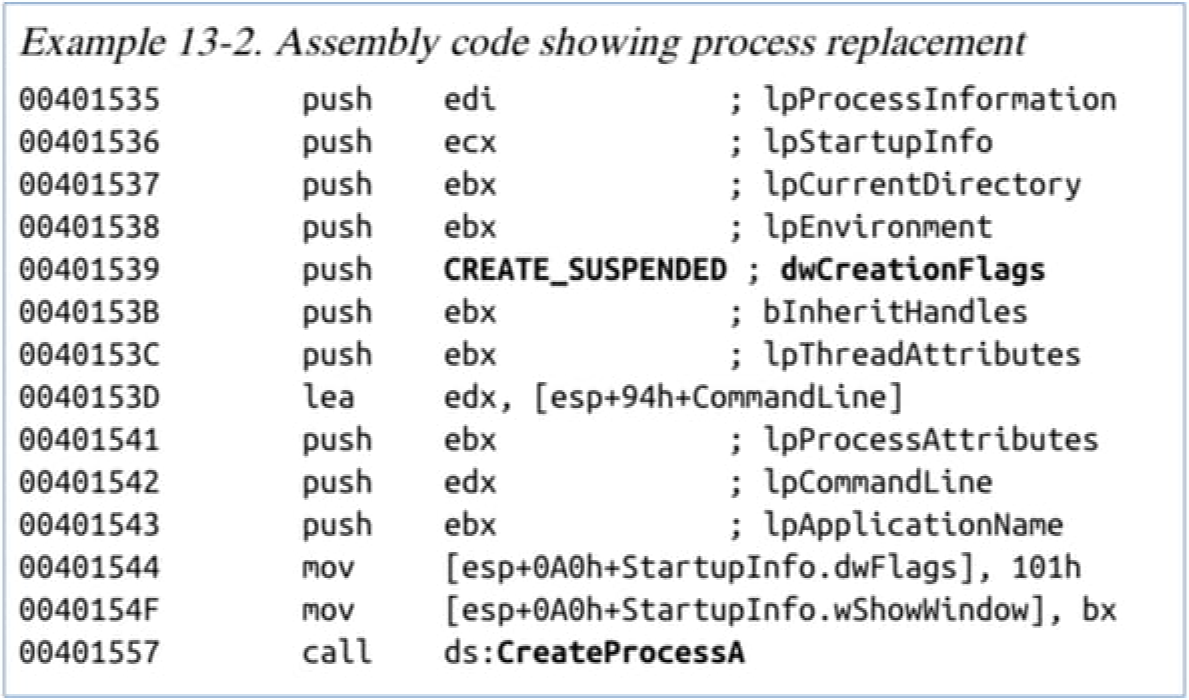
Process Replacement

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*

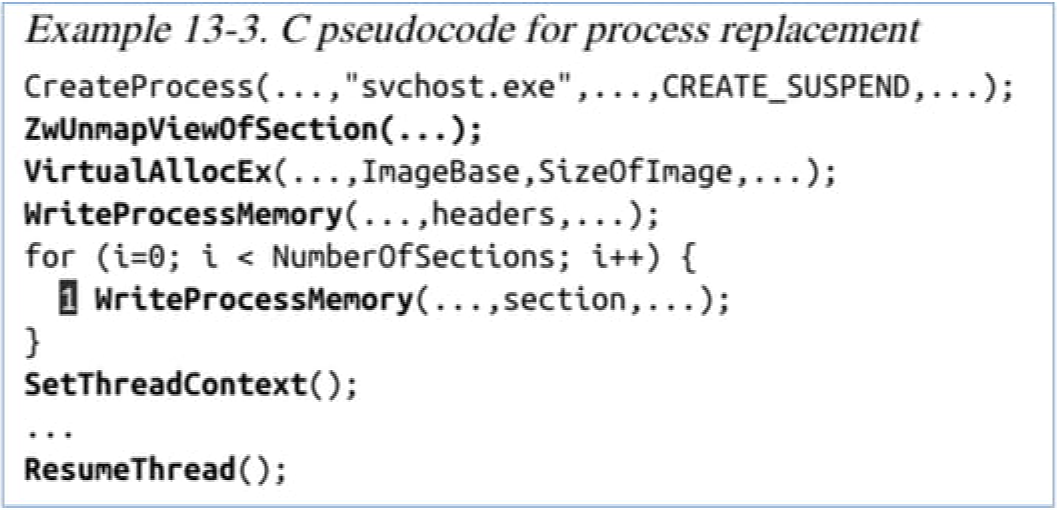
Suspended State

- 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

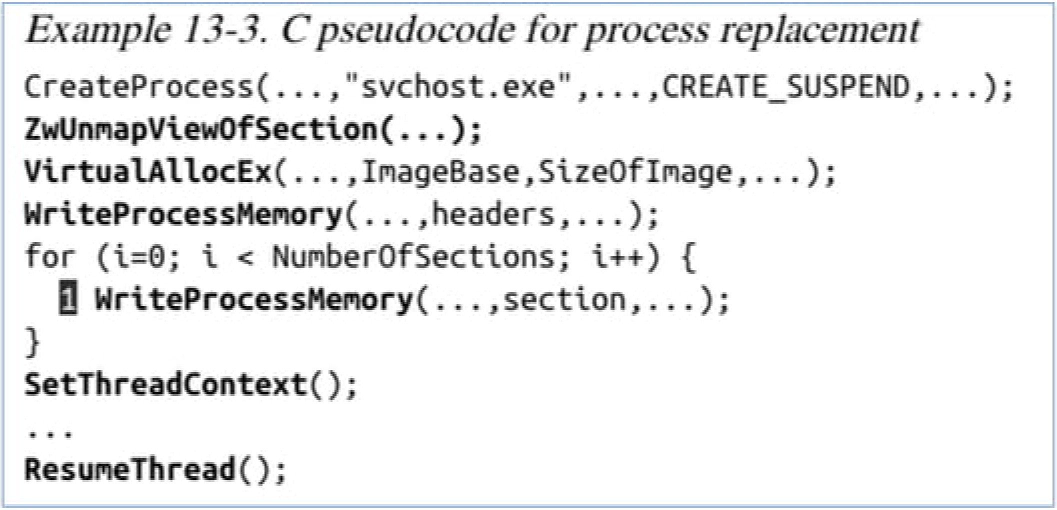
```
00401535        push    edi                 ; lpProcessInformation
00401536        push    ecx                 ; lpStartupInfo
00401537        push    ebx                 ; lpCurrentDirectory
00401538        push    ebx                 ; lpEnvironment
00401539        push    CREATE_SUSPENDED ; dwCreationFlags
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

```
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 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

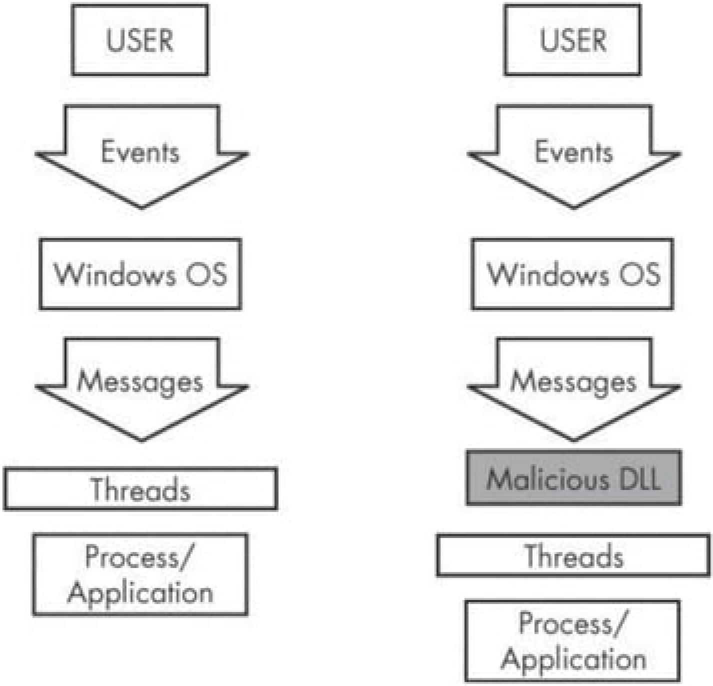


Figure 13-3. Event and message flow in Windows with and without hook injection

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

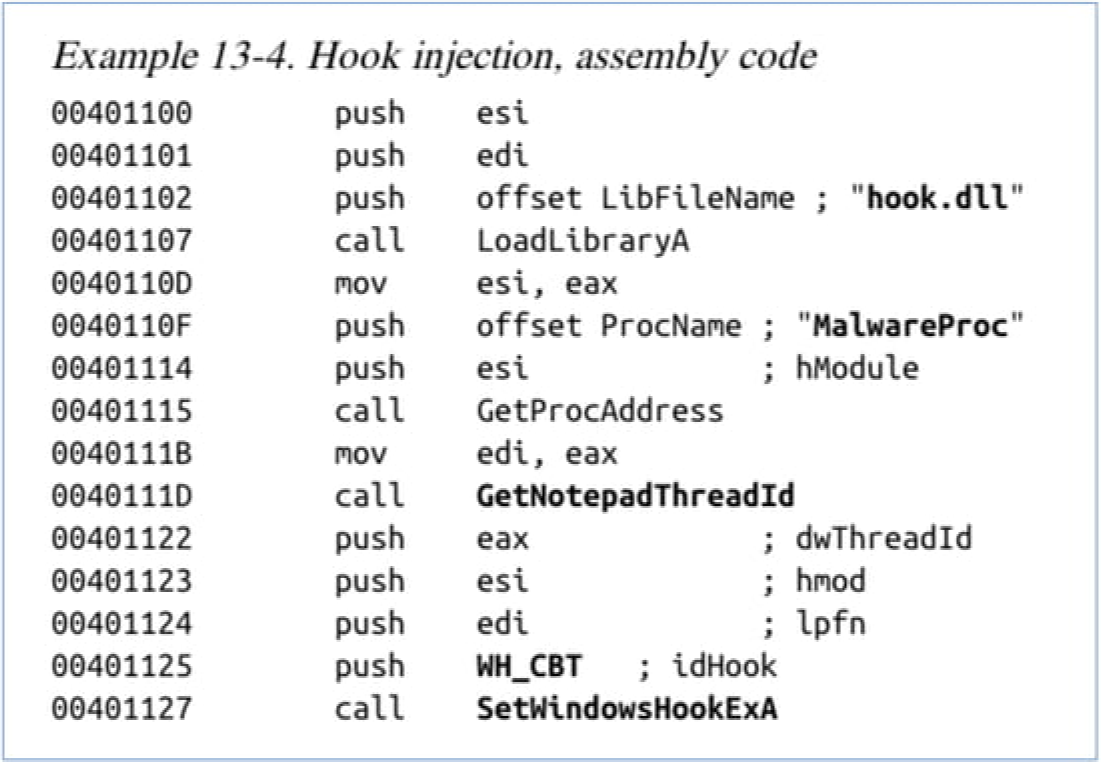
- Parameters
  - **idHook** - type of hook to install
  - **lpfn** - points to hook procedure
  - **hMod** - handle to DLL, or local module, in which the **lpfn** procedure is defined
  - **dwThreadId**- thread to associate the hook with. Zero = all threads
- The hook procedure must call **CallNextHookEx**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



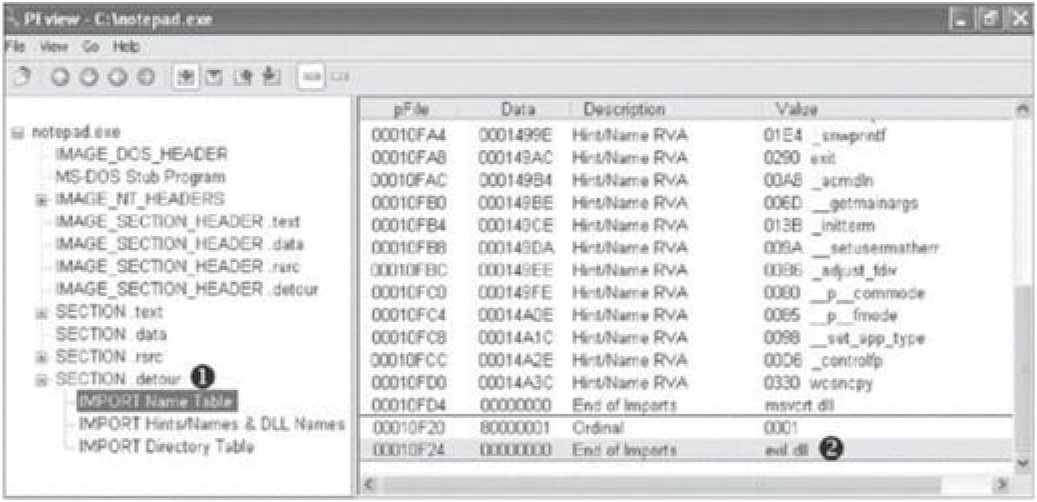
Example 13-4. Hook injection, assembly code

```
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



```
notepad.exe
    IMAGE_DOS_HEADER
    MS-DOS Stub Program
  + IMAGE_NT_HEADERS
    IMAGE_SECTION_HEADER .text
    IMAGE_SECTION_HEADER .data
    IMAGE_SECTION_HEADER .rsrc
    IMAGE_SECTION_HEADER .detour
  + SECTION .text
    SECTION .data
  + SECTION .rsrc
  - SECTION .detour ❶
      IMPORT Name Table
      IMPORT Hints/Names & DLL Names
      IMPORT Directory Table
```

pFile	Data	Description	Value
00010FA4	0001499E	Hint/Name RVA	01E4 _snwprintf
00010FA8	000149AC	Hint/Name RVA	0290 exit
00010FAC	000149B4	Hint/Name RVA	00A8 _acmdln
00010FB0	000149BE	Hint/Name RVA	006D __getmainargs
00010FB4	000149CE	Hint/Name RVA	013B _initterm
00010FB8	000149DA	Hint/Name RVA	009A __setusermatherr
00010FBC	000149EE	Hint/Name RVA	0086 _adjust_fdiv
00010FC0	000149FE	Hint/Name RVA	0080 __p__commode
00010FC4	00014A0E	Hint/Name RVA	0085 __p__fmode
00010FC8	00014A1C	Hint/Name RVA	0098 __set_app_type
00010FCC	00014A2E	Hint/Name RVA	00D6 _controlfp
00010FD0	00014A3C	Hint/Name RVA	0330 wcsncpy
00010FD4	00000000	End of Imports	msvcrt.dll
00010F20	80000001	Ordinal	0001
00010F24	00000000	End of Imports	evil.dll ❷

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

- **`setdll`** is the Microsoft tool used to point the PE to the new import table
- There are other ways to add a **`.detour`** section

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

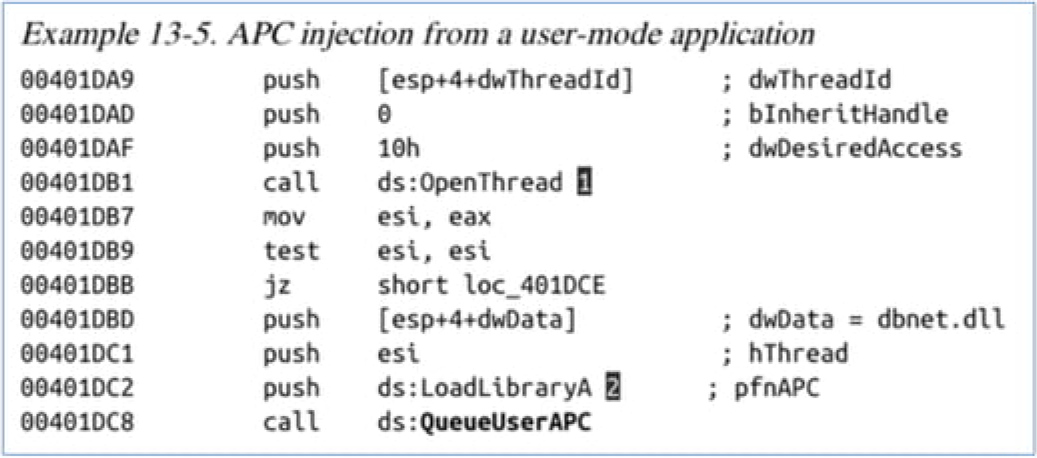
- Kernel-Mode APC
  - Generated for the system or a driver
- User-Mode APC
  - Generated for an application
- APC Injection is used in both cases

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



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 ❶
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 ❷       ; pfnAPC
00401DC8          call    ds:QueueUserAPC
```

- 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 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
```