

## **Covert Malware Launching**

CYBR473 - Malware and Reverse Engineering (2024/T1)

Lecturers: Arman Khouzani, Alvin Valera

Victoria University of Wellington - School of Engineering and Computer Science

- 1. Launcher
- 2. Process Injection
- 3. Process Hollowing
- 4. Hook Injection
- 5. Detours
- 6. APC Injection

## Part IV: Malware Functionality

Ch.12: Covert Malware Launching

"Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software", Michael Sikorski and Andrew Honig, 2012

# Practical Malware Analysis

The Hands-On Guide to Dissecting Malicious Software

> Michael Sikorski and Andrew Honig Foreword by Richard Bejtlich

A malware analyst must be able to recognise launching techniques in order to know how to find malware on a live system

- Describe common techniques for adding malicious functionality to programs
- > Recognise how this allows malware to hide itself

#### Launchei

**Process Injection** 

Launcher

**Process Hollowing** 

**Hook Injection** 

Detours

**APC** Injection

Launcher (a.k.a Loader) is a type of malware that sets itself or another piece of malware for immediate or future covert execution

Launchers often contain the malware that they're designed to load.

- The most common example is an executable or DLL (the real payload) embedded in its own resource section (.rsrc).
  - Examples of the normal contents of the resource section include icons, images, menus, and strings
- If the resource section is compressed or encrypted, the malware must perform resource section extraction before loading.
- This often means that the launcher uses resource-manipulation API functions such as FindResource, LoadResource, and SizeofResource.

```
HRSRC FindResourceA(
  [in, optional] HMODULE hModule,
  [in] LPCSTR lpName,
  [in] LPCSTR lpType
);
```

Determines the location of a resource with the specified type & name in the specified "module" (Windows term for an executable or DLL).

Parameters:

- [in, optional] hModule
   A handle to the module whose executable file contains the resource. If NULL, the module used to create the current process.
- [in] lpName The name of the resource (as a LPCSTR, or an integer identifier).
- [in] lpType The resource type (as a LPCSTR, or an integer identifier).

```
HGLOBAL LoadResource(
  [in, optional] HMODULE hModule,
  [in] HRSRC hResInfo
);
```

Retrieves a handle that can be used to obtain a pointer to the first byte of the specified resource in memory.

Parameters:

- [in, optional] hModule handle to the module whose executable file contains the resource. Default is the module that created the current process.
- [in] hResInfo handle to the resource to be loaded, returned by FindResource.

```
DWORD SizeofResource(
  [in, optional] HMODULE hModule,
  [in] HRSRC hResInfo
);
```

Retrieves the size, in bytes, of the specified resource.

Parameters:

- [in, optional] hModule handle to the module whose executable file contains the resource. Default is the module that created the current process.
- [in] hResInfo handle to the resource to be loaded, returned by **FindResource**.

*Note:* Malware launchers often must be run with administrator privileges or escalate themselves to have those privileges.

Average user processes can't perform all of the techniques that we will introduce in this lecture.

- We discussed privilege escalation previously (the "Malware behaviour" topic).

The fact that launchers may contain privilege-escalation code provides another way to identify them.

## **Process Injection**

#### Launcher

**Process Injection** 

**Process Hollowing** 

**Hook Injection** 

Detours

**APC** Injection

#### **Process Injection**

**Process Injection:** inject code into another running process, and that process unwittingly executes the malicious code.

- to conceal the malicious behaviour of their code
- to bypass host-based firewalls or process-specific security

Commonly used Windows API calls used in process injection:

- VirtualAllocEx: allocate space in an external process's memory
- ▷ WriteProcessMemory: write data to that allocated space.

Essential to these loading techniques that we will discuss:

- ► DLL Injection
- Process Replacement
- Hook Replacement

```
LPVOID VirtualAllocEx(

[in] HANDLE hProcess,

[in, optional] LPVOID lpAddress,

[in] SIZE_T dwSize,

[in] DWORD flAllocationType,

[in] DWORD flProtect

);
```

Reserves, commits, or changes (determined by flAllocationType) the state of a region of memory within the virtual address space of a process (specified by hProcess, which must have the **PROCESS\_VM\_OPERATION** access right.).

If the function succeeds, the return value is the base address of the allocated region of pages. If it fails, returns **NULL**.

#### BOOL WriteProcessMemory(

```
[in] HANDLE hProcess,
[in] LPVOID lpBaseAddress,
[in] LPCVOID lpBuffer,
[in] SIZE_T nSize,
[out] SIZE_T *lpNumberOfBytesWritten
);
```

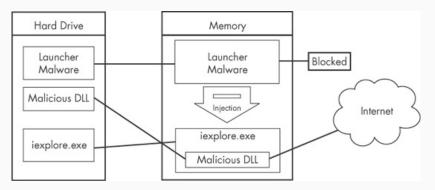
Writes data to an area of memory in a process (specified by hProcess, which must have **PROCESS\_VM\_WRITE** and **PROCESS\_VM\_OPERATION** access rights to the process).

The return value is nonzero if it succeeds and 0 if it fails. It fails if the requested write crosses into an inaccessible area of the process.

#### **DLL Injection:**

- a remote process is injected with a code containing calls to LoadLibrary with a path to a malicious DLL on the disk.
- ► the OS then loads the DLL and automatically calls its DllMain function, which has as much access to the system as the process in which it is running (it is in the same security context).
  - also everything the malicious DLL does will appear to originate from the compromised process.

While DLL injection can have legitimate purposes, e.g. debugging, monitoring or performance analysis, it is a common technique used by malware to evade detection, escalate privileges, or steal data.



The launcher gets to access the Internet via injecting into iexplore.exe.

```
HMODULE LoadLibraryA(
   [in] LPCSTR lpLibFileName
);
```

Loads the specified module into the address space of the calling process. The module may cause other modules to be loaded.

Parameters:

- [in] lpLibFileName The (full-path of the) file name of the (DLL) module

If the function succeeds, the return value is a handle to the module.

For DLL injection, the launcher malware must first obtain a **handle** to the victim process:

- Use the Windows API calls CreateToolhelp32Snapshot,
   Process32First, and Process32Next to search the process list for the injection target.
- Once the target process PID is found, use it to obtain the handle via a call to OpenProcess

Then **CreateRemoteThread** is commonly used to create and execute a thread in the target process, passed 3 main parameters:

- ▷ the process handle (hProcess) obtained with OpenProcess
- b the starting point of the injected thread (lpStartAddress), e.g. set to LoadLibrary
- ▷ an argument for that thread (lpParameter), e.g. name of the malicious DLL.

#### HANDLE CreateToolhelp32Snapshot(

```
[in] DWORD dwFlags,
[in] DWORD th32ProcessID
);
```

Takes a snapshot of the specified processes, as well as the heaps, modules, and threads used by these processes.

If the function succeeds, it returns an open handle to the specified snapshot. If it fails, returns INVALID\_HANDLE\_VALUE.

```
BOOL Process32First(
```

```
[in] HANDLE hSnapshot,
[in, out] LPPROCESSENTRY32 lppe
);
```

Parameters:

-

[in] hSnapshot
 A handle to the snapshot returned from a previous call to the
 CreateToolhelp32Snapshot function.

[in, out] lppe

A pointer to a **PROCESSENTRY32** structure. It contains process information such as the name of the executable file, the process identifier, and the process identifier of the parent process.

typedef	stru	ıct	tagPROCESSENTRY32	{
DWORD		dwSize;		
DWORD		cnt	:Usage;	
DWORD		th3	<b>32ProcessID;</b>	
ULONG_	PTR	th3	B2DefaultHeapID;	
DWORD		th3	32ModuleID;	
DWORD		cnt	Threads;	
DWORD		th3	32ParentProcessID;	
LONG		pcI	PriClassBase;	
DWORD		dwI	lags;	
CHAR		szł	<pre>ExeFile[MAX_PATH];</pre>	

} PROCESSENTRY32;

```
BOOL Process32Next(
```

```
[in] HANDLE hSnapshot,
[out] LPPROCESSENTRY32 lppe
);
```

Retrieves information about the next process recorded in a system snapshot.

Return Value:

- TRUE if the next entry of the process list has been copied to the buffer or FALSE otherwise. The ERROR\_NO\_MORE\_FILES error value is returned by the GetLastError function if no processes exist or the snapshot does not contain process information.

```
HANDLE OpenProcess(
   [in] DWORD dwDesiredAccess,
   [in] BOOL bInheritHandle,
   [in] DWORD dwProcessId
);
```

Opens an existing local process object.

Important Parameters:

- [in] dwDesiredAccess

The access to the process object. This access right is checked against the security descriptor for the process.

- [in] dwProcessId

The identifier of the local process to be opened. If

**SeDebugPrivilege** is enabled, all rights will be granted.

If succeeds, returns an open handle to the process, otherwise, NULL.

#### HANDLE CreateRemoteThread(

[in]	HANDLE	hProcess,
[in]	LPSECURITY_ATTRIBUTES	lpThreadAttributes,
[in]	SIZE_T	dwStackSize,
[in]	LPTHREAD_START_ROUTINE	lpStartAddress,
[in]	LPVOID	lpParameter,
[in]	DWORD	dwCreationFlags,
[out	] LPDWORD	lpThreadId
);		

Creates a thread that runs in the virtual address space of another process. The thread has access to all objects that the process opens.

If succeeds, returns a handle to the new thread, **NULL** if fails.

Main Parameters:

#### [in] hProcess

A handle to the process in which the thread is to be created. The handle must have the **PROCESS\_CREATE\_THREAD**, **PROCESS\_QUERY\_INFORMATION**, **PROCESS\_VM\_OPERATION**, **PROCESS\_VM\_WRITE**, and **PROCESS\_VM\_READ** access rights.

[in] lpStartAddress

A pointer to the application-defined function to be executed by the thread and represents the starting address of the thread in the remote process. The function must exist in the remote process.

[in] lpParameter

A pointer to a variable to be passed to the thread function.

```
FARPROC GetProcAddress(
   [in] HMODULE hModule,
   [in] LPCSTR lpProcName
);
```

Retrieves the address of an exported function (also known as a procedure) or variable from the specified dynamic-link library (DLL).

Parameters:

[in] hModule

A handle to the DLL module that contains the function or variable, e.g. returned by **GetModuleHandle** function.

- [in] lpProcName

The function or variable name, or the function's ordinal value.

Returns address of the exported function or variable, or **NULL** if fails.

Recall that for DLL injection, **CreateRemoteThread** is called with 3 main parameters:

- ▷ the process handle (hProcess) obtained with OpenProcess
- b the starting point of the injected thread (lpStartAddress) set to the LoadLibrary function, obtained via GetProcAddress.
- ▷ an argument for that thread (lpParameter), i.e., the name of the malicious DLL.

This assumes that **LoadLibrary** function and the malicious library name string exist within the victim process's memory space. **LoadLibrary** is in **kernel32**, which is always available. For the latter, malware uses **VirtualAllocEx** and **WriteProcessMemory**:

- VirtualAllocEx to allocate space in the target process if a handle to that process is provided.
- WriteProcessMemory to write the malicious library name string into that allocated memory space.

#### C Pseudocode for performing DLL injection:

The easiest way to identify DLL injection is by identifying this trademark pattern of Windows API calls in disassembly.

In DLL injection, the launcher never calls a malicious function: the malicious code is located in **DllMain**, which is automatically called by the OS when the DLL is loaded by the **LoadLibrary** thread.

#### DLL injection viewed in a debugger. Function calls are labelled **1**-**6**:

D040766BB [CALL DUORD PTR DS:[CAKERENT.32.OpenProcess)] WO DVORD PTR SS:[ERP-1000].EAX (WO DVORD PTR SS:[ERP-1000]1 0040766D ] WO DVORD PTR SS:[ERP-1000].700 (WO DVORD PTR SS:[ERP-1000].700 (WO DVORD PTR SS:[ERP-1000].700 (WO DVORD PTR SS:[ERP-1000]. WO DVORD PTR SS:[ERP-1000] WO DVORD PTR SS:[ERP-1000] WO DVORD PTR SS:[ERP-1000] WO DVORD PTR SS:[ERP-1000] WO DVORD PTR SS:[ERP-1010]. WE EAX FPFFFF WO EAX STORED PTR SS:[ERP-1180] WE EAX FPF SS (ERP-1180] WE EAX FPF SS (ERP FPF SS			
<pre>004076C1   NOV DUCKD FTR SS: [ERP-1000], EAX 004076C1   NOV DUCKD FTR SS: [ERP-1000], EAX 004076C2   DEC DUFT SS: [ERP-1000], I = 1 004076C3   DEC DUFT SS: [ERP-1000] 00407663   DES H = 0 00407664   DES H = 0 00407664   DES H = 0 00407667   DES H = 0 00407768   DES H = 0 00407769   DES H = 0 00407760   DES H = 0 0 00</pre>	004076BB	CALL DWORD PTR DS: [ <&KERNEL32.0penProcess>]	-OpenProcess
004076CE JJEZ_SEQUET_DILLIDGE.004076D8 004076CB JEXEFFFFFFF 004076CE JFE LiDGE.005770 004076E7 PUSH 4 004076E7 PUSH 4 00407711 ON FAX_FFFFFFFF 0 00407712 PUSH 55:[EP-100] 00407712 PUSH 104 00407712 PUSH 104 00407712 PUSH 104 00407712 PUSH 104 00407720 PUSH EXX 00407721 PUSH 104 00407721 PUSH 104 00407750 CALL DUGOD PTR DS:[EBP-1100] 00407750 CALL DUGOD PTR DS:[EBP-1100] 00407750 PUSH 104 00407750 PUSH 1051 00407750 PUSH 1051 00407750 PUSH 1051 00407750 PUSH PUSH 104 00407750 PUSH PUSH 104 00407750 PUSH 1051 00407750 PUSH 1051 0050 PUSH 10	004076C1		
004076CE JJEZ_SEQUET_DILLIDGE.004076D8 004076CB JEXEFFFFFFF 004076CE JFE LiDGE.005770 004076E7 PUSH 4 004076E7 PUSH 4 00407711 ON FAX_FFFFFFFF 0 00407712 PUSH 55:[EP-100] 00407712 PUSH 104 00407712 PUSH 104 00407712 PUSH 104 00407712 PUSH 104 00407720 PUSH EXX 00407721 PUSH 104 00407721 PUSH 104 00407750 CALL DUGOD PTR DS:[EBP-1100] 00407750 CALL DUGOD PTR DS:[EBP-1100] 00407750 PUSH 104 00407750 PUSH 1051 00407750 PUSH 1051 00407750 PUSH 1051 00407750 PUSH PUSH 104 00407750 PUSH PUSH 104 00407750 PUSH 1051 00407750 PUSH 1051 0050 PUSH 10	004076C7	CMP DWORD PTR SS: [EBP-1008], -1	
00407603 JWF DILINJEC.00407730 00407603 VW POCD PTR 55: [EBP-1000] 00407682 JWF DUSLINJEC.0040745 00407682 PUSH 104 00407685 NOVE EXI, DVORD PTR 55: [EBP-100] 00407685 NOVE EXI, DVORD PTR 55: [EBP-100] 00407762 NOV DVORD PTR 55: [EBP-101], EXX 00407762 NOV DVORD PTR 55: [EBP-101], EXX 00407771 PUSH 104 00407773 PUSH 104 00407737 PUSH 104 00407737 PUSH 105 00407737 PUSH 105 00407737 PUSH 105 00407737 PUSH 104 00407737 PUSH 104 0040774 CML DWORD PTR 55: [EBP-1100] 0040774 CML DWORD PTR 55: [EBP-1100] 0040774 PUSH 104 0040775 PUSH 104 0040775 PUSH 04 0040775 PUSH 05 0040775 PUSH 05			
00407660 [ NOV DUCKD FTR SS: [EBP-1000].7D0 00407662 [ PUSH 4 00407667 [ PUSH 4 00407678 PUSH 104 00407678 PUSH 104 00407678 PUSH 104 00407678 PUSH 6 00407767 PUSH 6 00407767 PUSH 6 00407767 PUSH 6 00407776 PUSH 6 0040776 PUSH 6	004076D0	OR EAX, FFFFFFFF	
00407682 [JWP DILInjec.00407646 00407685 PUSH 400 00407685 PUSH 500 00407685 PUSH 500 00407685 NOV EXX, VVORD PTR 55: [EBP-1000] 00407685 NOV EXX, VVORD PTR 55: [EBP-1010]. 00407767 PUSH EXX 00407787 JNZ SHORT DILInjec.00407719 00407778 PUSH EXX 00407720 [LEA ECX, VVORD PTR 55: [EBP-1100] 00407721 [LEA ECX, VVORD PTR 55: [EBP-1100] 00407724 PUSH EXX 00407724 PUSH EXX 00407725 PUSH EXX 00407726 PUSH EXX 00407727 PUSH EXX 00407726 PUSH EXX 00407727 PUSH EXX 00407727 PUSH EXX 00407727 PUSH EXX 00407727 PUSH EXX 00407726 PUSH PUSH PUSH PUSH EXS: [EBP-1100] 00407727 PUSH EXX 00407727 PUSH EXX 00407727 PUSH EXX 00407726 PUSH PUSH PUSH EXS: [EBP-1100] 00407757 PUSH EXX 00407756 PUSH DUTR 55: [EBP-1100] 00407757 PUSH EXX 00407756 PUSH EXX 00407757 PUSH EXX 00407756 PUSH EXX 00	004076D3		
00407687 PUSH 104 00407687 PUSH 104 00407687 PUSH 104 00407687 PUSH 104 00407687 PUSH 104 00407767 PUSH EXX 00407710 PUSH EXX 00407710 PUSH EXX 00407711 OVE AXX FFFFFFFF 00407711 OVE AXX FFFFFFFF 00407713 PUSH 104 00407712 PUSH 104 00407712 PUSH 104 00407712 PUSH 104 00407720 PUSH EXX 00407720 PUSH EXX 00407720 PUSH EXX 00407721 PUSH 104 00407720 PUSH EXX 00407720 PUSH EXX 00407720 PUSH PUSH 104 00407720 PUSH PUSH 104 00407750 CALL DUGOD PTR DS: [EBP-1100] 00407750 CALL PUGOD PTR DS: [EBP-1100] 00407750 CALL PUGOD PTR DS: [EBP-1100] 00407750 PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH	004076D8	MOW DWORD PTR SS: [EBP-100C], 7D0	
004076E59 PUSH 104 004076E7 PUSH 104 004076E7 PUSH 104 0040776E7 CALL DWORD PTR DS: [EDP-1000] 00407707 UK DWORD PTR DS: [EDP-1010]. EAX 00407708 CALL DWORD PTR SS: [EDP-1010]. EAX 00407709 UK DWORD PTR SS: [EDP-1100] 00407726 PUSH ECX 00407727 PUSH ECX 00407727 PUSH ECX 00407727 PUSH ECX 00407727 PUSH ECX 00407727 CALL DWORD PTR DS: [EDP-1000] 00407725 CALL DWORD PTR DS: [EDP-1000] 00407725 CALL DWORD PTR DS: [EDP-1100] 00407725 CALL DWORD PTR DS: [EDP-1100] 00407725 CALL DWORD PTR DS: [CAKEREL32. GetWooduleHandleV] 00407751 WOY DWORD PTR DS: [EDP-1100] 00407751 WOY DWORD PTR DS: [EDP-100] 00407751 WOY DWORD PTR DS: [EDP-100] 004077	004076E2	LJMP DILInjec.00407646	
0040768E PTSH 04 0040768E PTSH 0 0040768E PTSH 0 0040768E PTSH 0 0040768E PTSH 0 0040778E VALL DWORD PTR 55: [EBP-100].0 0040778E VALL DWORD PTR 55: [EBP-1010].5 00407778E VALL DWORD PTR 55: [EBP-1010].0 00407778E PTSH 104 00407773E PTSH 104 00407774E PTSH 104 00407774E PTSH 104 00407774E PTSH 104 00407775E PTSH 104 00407775E PTSH 104 00407775E PTSH 104 00407775E PTSH 104 00407775E PTSH 104 00407775E PTSH 0 00407775E PTSH 0 0040775E PTSH 0 0040775E PTSH 0 00407775E PTSH 0 0040775E PTSH 0 0040775	004076E7	PUSH 4	
004076873 [VTSH 0 004076873 [VTSH 0 00407687] [VTSH 253; [EEP-1010], EAX 00407798 [CM PUORD PTR 55; [EEP-1100] 00407771 [VTSH 04 00407771 [VTSH 04 00407725 [VTSH 253] 00407725 [VTSH 253] 00407725 [VTSH 253] 00407725 [VTSH 253] 00407725 [VTSH 253] 00407726 [VTSH 253			
00407665 NOV EXX, VOCRD FTR 55: [EBP-1000] 00407665 NOV EXX, VOCRD FTR 55: [EBP-1100]. 00407767 (VEST 64 (VES	004076EE	PUSH 104	
00407667 2025 EAX 00407672 (ALL DWORD PTR 55: [EBP-1101]. KAX 00407767 (ALL DWORD PTR 55: [EBP-1101]. KAX 00407721 (DR EAX, FFFFFFFF 00407711 (DR EAX, FFFFFFFF 00407711 (DR EAX, FFFFFFFF 00407721 PUSH 10 00407721 PUSH 10 00407721 PUSH 10 00407722 PUSH 10 00407722 PUSH 10 00407723 PUSH 10 00407724 PUSH 10 00407724 PUSH 10 00407724 PUSH 10 00407725 PUSH ECX 00407734 PUSH 10 00407724 PUSH 10 00407724 PUSH 10 00407724 PUSH 10 00407725 PUSH ECX 00407734 PUSH 10 00407726 PUSH ECX 00407734 PUSH 10 00407734 PUSH 10 00407734 PUSH 10 00407734 PUSH 10 00407740 CALL DWORD PTR 55: [EBP-1108] 00407740 CALL DWORD PTR 55: [EBP-1108] 00407750 CALL DWORD PTR 55: [EBP-1108] 00407756 PUSH 00 00407756 PUSH 00 PUSH 00 PUSH 00 PUSH 00 PUSH 00 P	004076F3		
<pre>0040776FC (ALL DWORD PTR DS: (c4KERNEI22. VirtualAllocEx)] 00407708 00407708 00407708 00407708 00407708 00407728 00407728 00407728 00407728 00407728 00407728 00407728 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 00407725 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 0040775 004077 0040775 004077 0040775 0040775 004077 0040775 004077 0040775 0040775 004077 0040775 00407 0040775 00407 0040775 00407 0040775 00407 0040775 00407 0040775 00407 0040775 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 00407 004</pre>	004076F5	MOV EAX, DWORD PTR SS: [EBP-1008]	
00407702 [NOT DPORD PTR 55: [EBP-1100]. 0 00407702 [NOT DPORD PTR 55: [EBP-1100]. 0 00407704 [NP DXLTsjec. 00407719 00407719 [USB 104 00407720 [USB 104 00407720 [USB 104] 00407720 [USB 104] 00407721 [USB 104] 00407724 [USB 104] 00407740 [CALL DWORD PTR 55: [EBP-1180]. 00407751 [CALL DWORD PTR 55: [EBP-1180]. 00407752 [CALL DWORD PTR 55: [EBP-1180]. 00407754 [USB 104] 00407755 [USB 104] 004	004076FB		
004077090 CMP DV0RD PTR SS: [EBP-1100].0 00407771 JQR EAX FFFFFFF 00407711 QR EAX FFFFFFF 00407711 QR EAX FFFFFFF 00407711 QTR DLLipic.0040779 00407721 PTS 104 00407725 QTS EAX 00407725 QTS EAX 00407725 QTS EAX 00407735 CALL DV0RD PTR SS: [EBP-1100] 00407735 CALL DV0RD PTR DS: [CAKERENIZ 32. WriteFrocessHearry] 00407736 CALL DV0RD PTR DS: [CAKERENIZ 32. WriteFrocessHearry] 00407737 QTS EAX 00407737 QTS EAX 00407737 QTS EAX 00407751 QTS EAX 00407751 QTS EAX 00407751 QTS EAX 00407751 QTS EAX 00407751 QTS EAX 00407751 QTS EAX 00407755 CALL DV0RD PTR DS: [CAKERENIZ 32. GetFrocAddress)] 00407755 QTS EAX 00407757 QTS EAX 00407750 QTS EAX 00407757 QTS EAX 00407750 QTS			kernel32.VirtualAllocEx (2)
00407710 [XZ_SIGNT DLILipic: 00407719 00407714 ] JNP DLILipic: 00407790 00407720 [VSB 104 00407720 [VSB 104 00407727 [VSB 104 00407727 [VSB 104 00407727 [VSB 104 00407727 [VSB 104 00407727 [VSB 104 00407727 [VSB 104 00407724 [VSB 104 00407724 [VSB 104 00407724 [VSB 104 00407724 [VSB 104 00407725 [VSB 104 00407725 [VSB 104 00407725 [VSB 104 00407725 [VSB 104 00407725 [VSB 104 00407725 [VSB 104 00407755 [VSB 104] 00407755 [			
00407711 OR EAX.FFFFFFF 00407711 OR EAX.FFFFFFF 00407713 PUBLID: 00407725 PUBLID: 00407726 PUBLIX: 00407726 PUBLIX: 00407727 PUBLIX: 00407727 PUBLIX: 00407727 PUBLIX: 00407727 PUBLIX: 00407727 PUBLIX: 00407726 PUBLIX: 00407740 CALL DWORD PTR DS:[EBP-1100] 00407740 CALL DWORD PTR DS:[EBP-1100] 00407740 CALL DWORD PTR DS:[EBP-1100] 00407750 CALL DWORD PTR DS:[EBP-1100] 00407750 CALL DWORD PTR DS:[EBP-1100] 00407751 CALL DWORD PTR DS:[EBP-1100] 00407756 PUBLIX: 00407756 PUBLIX: 00407757 PUBLIX: 00407756 PUBLIX:	00407708		
00407731 WP DSL 04 00407731 WSL 0 00407731 WSL 0 00407732 WSL 04 00407732 WSL 04 00407732 WSL 04 00407732 WOY BAX, DVORD PTR SS: [EBP-1100] 00407732 WOY EAX, DVORD PTR SS: [EBP-1008] 00407735 CALL DVORD PTR SS: [EBP-108] 00407735 CALL DVORD PTR SS: [EBP-1188] 00407735 WOY EAX, DVORD PTR SS: [EBP-1188] 00407751 WOY EXX, DVORD PTR SS: [EBP-1188] 00407751 WOY EXX, DVORD PTR SS: [EBP-1188] 00407751 WOY EXX, DVORD PTR SS: [EBP-1190] 00407751 WOY EXX, DVORD PTR SS: [EBP-1190] 00407757 WSL BAX 00407757 WSL BAX 00407751	0040770F	JNZ SHORT DLLInjec.00407719	
004077215 PUSH 10 004077216 PUSH 104 00407726 PUSH 104 00407727 PUSH 104 00407727 PUSH 104 00407727 PUSH 104 00407727 PUSH EXX 00407727 PUSH EXX 00407726 CALL DUOLD PTR 55: [EBP-1008] 00407740 CALL DUOLD PTR 55: [EBP-1180]. FAX 00407740 CALL DUOLD PTR 55: [EBP-1180]. FAX 00407740 CALL DUOLD PTR 55: [EBP-1180]. FAX 00407750 CALL DUOLD PTR 55: [EBP-1180]. FAX 00407751 CALL DUOLD PTR 55: [EBP-1180]. FAX 00407756 CALL DUOLD PTR 55: [EBP-1180]. FAX 004077576 CALL DUOLD PTR 55: [EBP-1190]. FAX 004077576 CALL DUOLD PTR 55: [EBP-1190]. FAX 004077576 VOY DUOLD PTR 55: [EBP-1190]. FAX 004077576 VOY FAX, DUORD PTR 55: [EBP-1190]. FAX 004077576 VOY FAX DUORD PTR 55: [EBP-1190]. FAX 004077576 VOY FAX, DUORD PTR 55: [EBP-1190]. FA	00407711	OR EAX.FFFFFFFF	
00407721 [B] PUSE 104 00407725 [B] REA RCX, DVORD PTR SS: [EBP-1180] 00407726 [PUSE ECX 00407727 [VOY EDX, DVORD PTR SS: [EBP-1008] 00407737 [VOY EDX, DVORD PTR SS: [EBP-1008] 00407737 [VOY EDX, DVORD PTR SS: [EBP-1180] 00407737 [VOY EDX, DVORD PTR SS: [EBP-1180] 00407751 [VOY EXX, DVORD PTR SS: [EBP-1180] 00407751 [VOY EDX EXX 00407751 [VOY EDX EXX 00407755 [VOSE EXX 00407755 [VOSE EXX 00407755 [VOSE EXX 00407755 [VOSE EXX 00407755 [VOSE EXX 00407755 [VOSE DTR SS: [EBP-1190] 00407755 [VOSE 0X 00407755 [VOSE 0X 005 [VOSE 0X 005 [VOSE 0X 005	00407714	JMP DLLInjec.0040779D	
00407726   EEA ECT, DVORD PTR SS: [EBP-1100] 00407727   VISH ECX 00407727   VISH ECX 00407727   VISH ECX 00407727   VISH ECX 00407724   VISH ECX 00407724   VISH ECX 00407725   VISH ECX 00407726   VISH ECX 00407	00407719	PUSH 0	pBytes∀ritten = NULL
00407727 DVSE ECX 00407725 VOVED PTR SS: [EBP-1010] 00407725 VOVED VOVED PTR SS: [EBP-1008] 00407735 CALL DVGRD PTR DS: [CAKERNEL32.VriteProcessMemory) 00407735 CALL DVGRD PTR DS: [CAKERNEL32.VriteProcessMemory) 00407740 CALL DVGRD PTR DS: [CAKERNEL32.CetKoduleHandleV) 00407740 CALL DVGRD PTR DS: [CBP-1108] 00407747 PUSH ECX 00407751 CALL DVGRD PTR SS: [EBP-1190] 00407755 VOVED PTR SS: [EBP-1190] 00407757 PUSH ECX 00407751	0040771B	PUSH 104	BytesToVrite = 104 (260.)
00407727 NOV EDX, DVORD PTR 55: [EBP-1010] 00407727 WOV EXX, DVORD PTR 55: [EBP-1008] 00407725 CALL DVOLT PTS 5: [CBP-1008] 00407735 CALL DVOLT PTR 55: [CBP-108] 00407746 NOV DVORD PTR 55: [EBP-1188]. 00407746 NOV DVORD PTR 55: [EBP-1188] 00407751 NOV ECX, DVORD PTR 55: [EBP-1188] 00407751 NOV ECX, DVORD PTR 55: [EBP-1188] 00407751 NOV EXX, DVORD PTR 55: [EBP-119] 00407756 PUSH ECX 00407756 PUSH ECX 004077575 PUSH ECX 004077575 PUSH ECX 004077575 PUSH ECX 004077576 PUSH ECX 00407756 PUSH ECX 004077576 PUSH ECX 00407756 PUSH ECX 004077576 PUSH ECX 00407756 PUSH ECX 00407756 PUSH ECX 004077576 PUSH ECX 004077576 PUSH ECX 004077576 PUSH ECX 004077576 PUSH ECX 004077576 PUSH ECX 00407756 PUSH ECX 004077576 PUSH ECX 00407756 PUSH ECX 004077576 PUSH ECX 00407757 PUSH ECX 00407757 PUSH ECX		LEA ECX, DWORD PTR SS: [EBP-1180]	
00407722D PUSH EXX 00407722D VDVSR DY 00407724 (DVSR AX, DVSRD PTR SS: [EBP-1008] 00407724 (DVSR AX, DVSRD PTR SS: [CAKEENEL12. VriteProcessMesory)] 00407730 (CALL DVSRD PTR DS: [CAKEENEL12. GetModuleRandleV)] 00407740 (CALL DVSRD PTR DS: [CAKEENEL12. GetModuleRandleV)] 00407740 (CALL DVSRD PTR DS: [CAKEENEL12. GetModuleRandleV)] 00407740 (CALL DVSRD PTR DS: [EBP-1188], EAX 00407750 (CALL DVSRD PTR DS: [EBP-1198], EAX 00407751 (CALL DVSRD PTR DS: [EBP-1198], EAX 00407756 (VSRD DVSRD PTR DS: [EBP-1198], EAX 004077576 (VSRD DVSRD PTR DS: [EBP-1190] 004077576 VSRD DVSRD PTR SS: [EBP-1190] 00407758 VSRD VSRD VSRD PTR SS: [EBP-1190] 00407758 VSRD VSRD VSRD PTR SS: [EBP-1190] 00407758 VSRD VSRD VSRD VSRD VSRD VSRD VSRD VSRD			Buffer
00407727 [VOT EAX, DVORD PTR SS:[EEP-1008] 00407735 CALL DVORD PTR DS:[( <kkeknel32.v:itefrocessmemory) 00407735 [VOT EAX, DVORD PTR DS:[(<kkeknel32.getmodulehandlev)] 00407736 [VOT DVORD PTR DS:[(<kkeknel32.getmodulehandlev)] 004077571 NOT EXX, DVORD PTR SS:[EEP-1108] 004077578 [VOT DVORD PTR SS:[EEP-1109] 004077578 [VOT DVORD PTR SS:[EEP-1109] 004077578 [VOT DVORD PTR SS:[EEP-1109] 004077578 [VOT DVORD PTR SS:[EEP-1109] 004077578 [VOT EXX, DVORD PTR SS:[EEP-1109] 00407775 [VOT EXX, DVORD PTR SS:[EEP-1109] 00407775 [VOT EXX, DVORD PTR SS:[EEP-1008] 00407775 [VOT EXX, DVOR</kkeknel32.getmodulehandlev)] </kkeknel32.getmodulehandlev)] </kkeknel32.v:itefrocessmemory) 	00407727	MOW EDX, DWORD PTR SS: [EBP-1010]	
004077374 [VCSH EAX 00407735 (CALL DWOED PTR DS:[<(KKEKNEL32.VriteProcessMesory)] 00407735 [VCSH EAX 0040774 [VCSH EAX 0040774 [VCSH EAX 0040774 [VCSH EAX 0040775 [VCSH EAX 0040776 [VCSH EAX 0040	0040772D		Address
00407725       CALL DWORD PTR DS: (ckKENEL32. WriteProcessHearry)       VriteProcessHearry)         00407735       VRIteProcessHearry)       VriteProcessHearry)         00407736       VRIteProcessHearry)       VriteProcessHearry)         00407736       VRIteProcessHearry)       VriteProcessHearry)         00407751       VRIteProcessHearry)       VriteProcessHearry)         00407751       VRV DWORD PTR DS: (EBP-1100)       CathoduleHandley)         00407751       VRV DWORD PTR DS: (EBP-1100)       CathoduleHandley)         00407751       VRV DWORD PTR DS: (CAKENEL32. GetProcAddress)       ProcHaseOrOrdinal - *LoadLibrary*         00407751       VRV DWORD PTR DS: (CAKENEL32. GetProcAddress)       Module         00407752       VRV DWORD PTR DS: (CAKENEL32. GetProcAddress)       Module         00407754       VRV DWORD PTR DS: (EBP-1010)       Module         00407754       VRV EXL DWORD PTR SS: (EBP-1190]       Module         00407757       VRV EXL DWORD PTR SS: (EBP-1008)       Module         004077757       VRV EXL DWORD PTR SS: (EBP-1008)       Module         004077757       VRV EXL DWORD PTR SS: (EBP-1008)       Module         004077757       VRV EXL DWORD PTR SS: (EBP-1008)       Module			
00407723 [PUSH DLLInjcc.0040ACCC 00407746 [ALL DWORD PTR DS:[4KEWEL32.GetKoduleBandleV>] 00407746 [NOV DVGND PTR DS:[4KEWEL32.GetKoduleBandleV>] 00407747 [PUSH DLLInjcc.0040ACEW 00407751 [NOV ECK,DWORD PTR SS:[EBP-1100] 00407751 [NOV ECK,DWORD PTR SS:[EBP-1100] 00407755 [NOV DVGND PTR SS:[EBP-1100] 00407756 [PUSH 0 00407756 [PUSH 0 00407756 [PUSH 0 00407757 [PUSH EAX 000 VV EDX,DWORD PTR SS:[EBP-1100] 00407757 [PUSH EAX 00407756 [PUSH 0 00407757 [PUSH EAX 00407756 [PUSH 0 00407775 [PUSH EAX 00407756 [PUSH 0 00407775 [PUSH EAX 000 PTR SS:[EBP-1000] 00407775 [PUSH EAX			hProcess
00407740 (ALL DUORD PTR DS: [(cAKERNEL2.2.GetModuleHandleV)] 00407746 (NOV DVORD PTR DS: [[EN-1108], EAX 00407746 (PUSH DLILINGE0040ACR8 00407757 (PUSH DCX 00407757 (CALL DUORD PTR DS: [ENP-1198]) 00407757 (CALL DUORD PTR DS: [ENP-1190] 00407756 (PUSH DCX 00407756 (PUSH DCX 00407756 (PUSH DCX 00407757 (PUSH ECX 00407775 (PUSH ECX 00407775 (PUSH ECX 00407775 (PUSH CCX 00407775 (PUSH CCX 00407775 (PUSH CCX 00407775 (PUSH CCX 0040775 (PUSH CCX 00407			-WriteProcessMemory 6
00407746 NOV DVORD FTR SS: [EBP-1100] 00407745 [VOTB DLLing-0.0040ACE 00407751 NOV ECX.D00RD FTR SS: [EBP-1100] 00407757 [VOTB ECX. 00407757 [VOTB DVCD FTR SS: [EBP-1100] 00407756 VOT DV0RD FTR SS: [EBP-1100] 00407756 VOT DV0RD FTR SS: [EBP-1100] 00407756 PUSH 0 00407757 [VOTB ECX. 00407757 [VOTB ECX. 00407758 [V			pHodule = "kernel32.dll"
0040774C [205H DILLiges.00404C88 00407751 [WOY ECX,UPOED PTR 55: [EBP-1188] 00407757 [PUEN ECX 00407756 [WOY DOKOD PTR 55: [EBP-1190].EAX 00407756 [PUEN 6 00407756 [PUEN 6 00407756 [WOY EXX,UPOED PTR 55: [EBP-1100] 00407775 [PUEN EAX 00407775 [PUEN 6 00407776 [PUEN 6 0040776 [PUEN 6 0 0 0 0 0 0 0 0 0 0 0 0 0			-GetModuleHandleW
004077551 NOV ECX.DVORD PTR 55:[EBP-1108] 00407757 DVG HCX 00407758 (CALL DVGND PTR D5:[ckkEKNEL32.GetProcAddress)] 00407758 (VO DVGND PTR D5:[ckkEKNEL32.GetProcAddress)] 00407754 (VIGI 0 00407754 (VIGI 0 00407754 (VIGI 0 00407755 (VIGI 0 00407755 (VIGI 0 00407757 (VIGI 2 00407757 (VIGI 2 00407757 (VIGI 2 00407757 (VIGI 0 00407775 (VIGI 0 00407775 (VIGI 0 00407775 (VIGI 0 00407775 (VIGI 0 00407775 (VIGI 0 00407775 (VIGI 0 00407776 (VIGI 0 00407776 (VIGI 0 00407776 (VIGI 0 0 0 0 0 0 0 0 0 0 0 0 0 0			•
00407757 [ DDSH ECX [ AKEKNEL32.GetProcAddress)] 00407755 [ ACIL DWORD PTR DS:[(4KEKNEL32.GetProcAddress)] 00407756 [ DDSN 00 PORD PTR SS:[EBP-1190].EAX 00407766 [ PDSN 0 00407766 [ PDSN 0 004077676 [ PDSN 0 00407776 [ PDSN E0X ] 004077776 [ PDSN E0X ] 004077776 [ PDSN E0X ] 00407777 [ PDSN E0X ] 00407777 [ PDSN E0X ] 00407777 [ PDSN E0X ] 00407777 [ PDSN E0X ] 00407778 [ PDSN 007 ECX, DWORD PTR SS:[EBP-1008] 00407780 [ PDSN E0X ] 00407778 [ PDSN E0X ] 0040778 [ PDSN E0X ] 0040777			ProcNameOrOrdinal = "LoadLibraryA"
00407755 (CALL DUGUD PTR D5: (c&REKRE12.2.GetProcAddress>) 00407755 (VoV DVORD PTR D5: (c&REP-1190) EAX 00407764 [PUSH D0 00407765 [PUSH EDX 004077765 [PUSH EDX 004077775 [PUSH EAX 004077775 [PUSH EAX 004077775 [PUSH EAX 00407775 [PUSH EAX 0040775 [PUSH EAX			
0040775E MOT DVORD PTR SS:[EBP-1190].EAX 00407766 PUSH 0 00407766 PUSH 0 00407768 PUSH E0X 00407768 PUSH E0X 00407776 PUSH E0X 00407775 PUSH EAX 00407775 PUSH EAX 00407775 PUSH EAX 00407778 PUSH E0X 00407778 PUSH E0X			
00407764 PUSH 0 00407766 PUSH 0 00407768 NOV EDS 0 004077678 NOV EDS 0 00407776 PUSH 0 00407775 PUSH EAX 00407775 PUSH 6 00407778 PUSH 0 00407778 PUSH 0 00407778 PUSH 0 0040778 PUSH 0 0 0040778 PUSH 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			-GetProcåddress
00407766 PUSH 00 KDX, DWORD PTR SS: [EEP-1010] 00407768 PUSH E0X 00407776 PUSH E0X 00407775 PUSH EAX 00407775 PUSH EAX 00407776 PUSH EAX 00407776 PUSH EAX 00407770 NOV ECX, DWORD PTR SS: [EEP-1008]			
00407756   NOV EDX, DUORD PTR SS:[EBP-1010] 00407756   NOV EDX, DUORD PTR SS:[EBP-1190] 004077575   NOV EAX, DUORD PTR SS:[EBP-1190] 00407775   PUSH 0 00407775   PUSH 0 00407774   NOV ECX, DUORD PTR SS:[EBP-1008] 00407770   PUSH ECX			
0040776E PUSH EOX 0040776F PUSH EOX 00407775 PUSH & AX 00407775 PUSH & AX 00407776 PUSH & AX 00407776 PUSH & AX 00407770 NOV ECX, DWORD PTR 55:[EDP-1008] 00407700 PUSH ECX			
0040775F NOV EAX, DUORD PTR SS:[EBP-1190] 00407775 PUSH EAX 00407775 PUSH 0 00407778 PUSH 0 00407778 NOV ECX, DUORD PTR SS:[EBP-1008] 00407778 PUSH ECX			
00407775 PUSH EAX 00407776 PUSH 0 00407778 PUSH 0 0040778 PUSH EAX			
00407775 [ PUSH 0 00407778   PUSH 0 00407778   NOV ECX, NOVED PTR SS:[EBP-1008] 00407700   PUSH ECX			
00407778 PUSE 0 0040777A NOV ECX, DWORD PTR SS:[EBP-1008] 00407780 PUSE ECX			
0040777A NOV ECX, DWORD PTR SS: [EBP-1008] 00407780 PUSH ECX			
00407780 PUSH ECX			
00407780 POSH ECX 00407781 CALL DWORD PTR DS:[<&KERNEL32.CreateRemoteThread>] kernel32.CreateRemoteThread 0			
CALL DWORD FTR DS:[<&KERNEL32.CreateRemoteThread>] kernel32.CreateRemoteThread U	00407780	PUSH ECX	
	00407781	CALL DWORD FTR DS:[<&KERNEL32.CreateRemoteThread>]	kernel32.CreateRenoteThread 😈

Once you find DLL injection activity in disassembly, you should start looking for the strings containing the names of the malicious DLL and the victim process.

They must be accessed before this code executes. The victim process name can often be found in a **strncmp** function (or equivalent) when the launcher determines the victim process's PID.

To find the malicious DLL name, we could set a breakpoint at **0x407735** and dump the contents of the stack to reveal the value of Buffer as it is being passed to **WriteProcessMemory**.

#### Direct Injection:

Like DLL injection, but instead of writing a separate DLL and forcing the remote process to load it, direct-injection malware injects the malicious code directly into the remote process.

Direct injection is more flexible, but requires a lot of customized code in to run successfully without negatively impacting the host process.

This can be used to inject compiled code, but more often, shellcode.

There will typically be two calls to **VirtualAllocEx** and **WriteProcessMemory**:

- ▷ The first allocates and writes the **data** used by the remote thread.
- ▷ The second allocates and writes the remote thread **code**.

The call to **CreateRemoteThread** contains location of the remote thread code (lpStartAddress) and the data (lpParameter).

Since the data and functions used by the remote thread must exist in the victim process, normal compilation procedures will not work.

For example, strings are not in the normal .data section, and LoadLibrary/GetProcAddress will need to be called to access functions that are not already loaded.

To analyze the remote thread's code, you may need to debug the malware and dump all memory buffers that occur before calls to **WriteProcessMemory** to be analyzed in a disassembler.

Since these buffers most often contain shellcode, you will need **shellcode** analysis skills.

## **Process Hollowing**

Launcher

**Process Injection** 

Process Hollowing

**Hook Injection** 

Detours

**APC** Injection

Rather than inject code into a host program, a method known as **process replacement** overwrites the memory space of a running process with a malicious executable, providing it with the same privileges as the process it is replacing.

This is used when a malware wants to disguise as a legitimate process, without the risk of crashing it through process injection.

Key to process replacement is creating a process in a **suspended state**: the process will be loaded into memory, but the primary thread of the process is suspended.

The program will not do anything until an external program resumes the primary thread, causing the program to start running.

```
BOOL CreateProcessA(
```

```
[in, optional]
                                              lpApplicationName,
                      LPCSTR
  [in, out, optional] LPSTR
                                              lpCommandLine,
                      LPSECURITY_ATTRIBUTES lpProcessAttributes,
  [in, optional]
  [in, optional]
                      LPSECURITY ATTRIBUTES lpThreadAttributes,
                                              bInheritHandles.
  [in]
                      BOOL
  [in]
                                              dwCreationFlags,
                      DWORD
  [in, optional]
                                              lpEnvironment,
                      LPVOTD
  [in, optional]
                      LPCSTR
                                              lpCurrentDirectory,
  [in]
                      LPSTARTUPINFOA
                                              lpStartupInfo,
                      LPPROCESS_INFORMATION lpProcessInformation
  [out]
);
```

One of the possible values for dwCreationFlags:

**CREATE\_SUSPENDED** (**0x4**): The primary thread of the new process is created in a suspended state, and does not run until the **ResumeThread** function is called.

The following example shows how this suspended state is achieved by passing **CREATE\_SUSPENDED** (**0x4**) for the **dwCreationFlags** parameter when performing the call to **CreateProcess**:

00401535	push	edi ; lpPr	ocessInformation
00401536	push	ecx ; lpSt	artupInfo
00401537	push	ebx ; lpCu	rrentDirectory
00401538	push	ebx ; lpEn	vironment
00401539	push	CREATE_SUSPENDED ; dwC	reationFlags
0040153B	push	ebx ; bInh	eritHandles
0040153C	push	ebx ; lpTh	readAttributes
0040153D	lea	edx, [esp+94h+CommandL	ine]
00401541	push	ebx ; lpPr	ocessAttributes
00401542	push	edx ; lpCo	mmandLine
00401543	push	ebx ; lpAp	plicationName
00401544	mov	[esp+0A0h+StartupInfo.	dwFlags], 101h
0040154F	mov	[esp+0A0h+StartupInfo.	wShowWindow], bx
00401557	call	ds:CreateProcessA	

#### The next example shows a C pseudocode for process replacement.

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

Once the process is created, the next step is to replace the victim process's memory with the malicious executable, typically using **ZwUnmapViewOfSection** to release all memory pointed to by a section passed as a parameter.

After memory is unmapped, the loader performs **VirtualAllocEx** to allocate new memory for the malware, and uses **WriteProcessMemory** to write each of the malware sections to the victim process space, typically in a loop, as shown in the example.

Next, the malware restores the victim process environment so that the malicious code can run by calling **SetThreadContext** to set the entry point to point to the malicious code.

Finally, **ResumeThread** is called to initiate the malware, which has now replaced the victim process.

Process replacement is an effective way for malware to appear non-malicious.

By masquerading as the victim process, the malware is able to bypass firewalls or intrusion prevention systems (IPSs) and avoid detection by appearing to be a normal Windows process.

Also, by using the original binary's path, the malware deceives the savvy user who, when viewing a process listing, sees only the known and valid binary executing, with no idea that it was unmapped.

# **Hook Injection**

Launcher

**Process Injection** 

**Process Hollowing** 

Hook Injection

Detours

**APC** Injection

 $\begin{array}{c} 5 & 5 & 0 & 9 \\ 4 & 5 & 8 \\ 2 & 8 \\ 5 & 5 & 7 \\ 6 & 0 \\ 2 & 0 \end{array}$ 

•

### Hook injection:

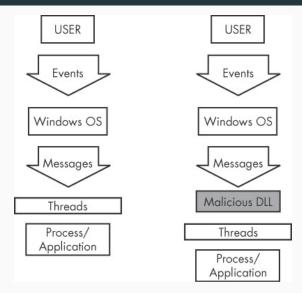
A method to load malware that takes advantage of Windows hooks, which are used to intercept messages destined for applications.

Malware authors use hook injection to accomplish two things:

- Ensure that malicious code will run whenever a particular message is intercepted
- Ensure that a particular DLL is loaded in a victim process's memory space

Users generate events that are sent to the OS, which then sends messages created by those events to threads registered to receive them. An attacker can insert a malicious DLL to intercept them.

## **Hook Injection**



Event and message flow in Windows with and without hook injection

There are two types of Windows hooks:

- Local hooks are used to observe or manipulate messages destined for an internal process.
- Remote hooks are used to observe or manipulate messages destined for a remote process (another process on the system).

Remote hooks are available in two forms:

- High-level remote hooks: require that the hook procedure be an exported function contained in a DLL, which will be 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. This procedure is notified before the OS gets a chance to process the event.

Hook injection is frequently used in malicious applications known as **keyloggers**, which record keystrokes.

Keystrokes can be captured by registering high- or low-level hooks using the **wH\_KEYBOARD** or **WH\_KEYBOARD\_LL** hook procedure types, respectively.

- For WH\_KEYBOARD procedures, the hook will often be running in the context of a remote process, but it can also run in the process that installed the hook.
- For WH\_KEYBOARD\_LL procedures, the events are sent directly to the process that installed the hook, so the hook will be running in the context of the process that created it.

Using either hook type, a keylogger can intercept keystrokes and log them to a file or alter them before passing them through.

## Hook Injection: Using SetWindowsHookEx

The principal function call used to perform remote Windows hooking is **SetWindowsHookEx**, which has the following parameters:

- ▷ **idHook**. Specifies the type of hook procedure to install.
- ▷ **lpfn.** Points to the hook procedure.
- hMod. For high-level hooks, the handle to the DLL containing the hook procedure defined by lpfn. For low-level hooks, the local module in which the lpfn procedure is defined.
- dwThreadId. the identifier of the thread with which the hook procedure is to be associated. If zero, the hook procedure is associated with all existing threads running in the same desktop as the calling thread. This must be zero for low-level hooks.

The hook procedure must call CallNextHookEx, which ensures that the next hook procedure in the call chain gets the message and that the system continues to run properly.

When targeting a specific **dwThreadId**, malware generally includes instructions for determining which system thread identifier to use, or it is designed to load into all threads.

Malware will load into all threads only if it's a keylogger or the equivalent (when the goal is message interception). However, loading into all threads can degrade the running system and may trigger an IPS. Therefore, if the goal is to simply load a DLL in a remote process, only a single thread will be injected, to remain stealthy.

If a malicious application hooks a Windows message that is used frequently, it's more likely to trigger an IPS, so malware will often set a hook with a message that is not often used, such as **WH\_CBT** (a computer-based training message).

The following example shows the assembly code for performing hook injection in order to load a DLL in a different process's memory space.

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

In the previous example, the malicious DLL (**hook.dll**) is loaded by the malware, and the malicious hook procedure address is obtained.

The hook procedure, MalwareProc, calls only CallNextHookEx.

**SetWindowsHookEx** is then called for a thread in **notepad.exe** (assuming that it is running). **GetNotepadThreadId** is a locally defined function that obtains a **dwThreadId** for **notepad.exe**.

Finally, a **WH\_CBT** message is sent to the injected **notepad.exe** to force **hook.dll** to be loaded by **notepad.exe**.

Once **hook.dll** is injected, it can execute the full malicious code stored in **DllMain**, while disguised as the **notepad.exe** process.

Since MalwareProc calls only CallNextHookEx, it should not interfere with incoming messages, but malware often immediately calls LoadLibrary and UnhookWindowsHookEx in DllMain to ensure that incoming messages are not impacted.

# **Detours**

Launcher

**Process Injection** 

**Process Hollowing** 

**Hook Injection** 

Detours

**APC** Injection

-

#### Detours

**Detours** is a library developed by Microsoft Research in 1999. It was originally intended as a way to easily instrument and extend existing OS and application functionality. The Detours library makes it possible for a developer to make application modifications simply.

Malware authors use the Detours library to perform import table modification, attach DLLs to existing program files, and add function hooks to running processes.

Malware authors most commonly use Detours to add new DLLs to existing binaries on disk. The malware modifies the PE structure and creates a section named **.detour**, which is typically placed between the export table and any debug symbols. The **.detour** section contains the original PE header with a new import address table. The malware author then uses Detours to modify the PE header to point to the new import table, by using the **setdll** tool provided with the Detours library.

### Detours

#### A **PEview** of Detours used to trojanize **notepad.exe** is shown:

e View Go Help					
30000 2 2 2 2					
	pFile	Data	Description	Value	
a notepad.exe	00010FA4	0001499E	Hint/Name RVA	01E4 snwprintf	
-IMAGE_DOS_HEADER	00010FA8	000149AC	Hint/Name RVA	0290 exit	
-MS-DOS Stub Program	00010FAC	000149B4	Hint/Name RVA	00A8 acmdin	
■ IMAGE_NT_HEADERS	00010FB0	000149BE	Hint/Name RVA	006D getmainargs	
-IMAGE_SECTION_HEADER .text	00010FB4	000149CE	Hint/Name RVA	013B initterm	
-IMAGE_SECTION_HEADER .data	00010FB8	000149DA	Hint/Name RVA	009A setusermatherr	
- IMAGE_SECTION_HEADER .rsrc	00010FBC	000149EE	Hint/Name RVA	0086 _adjust_fdiv	
-IMAGE_SECTION_HEADER .detour	00010FC0	000149FE	Hint/Name RVA	0080 _p_commode	
SECTION .text	00010FC4	00014A0E	Hint/Name RVA	0085 p fmode	
- SECTION .data	00010FC8	00014A1C	Hint/Name RVA	0098set_app_type	
SECTION .rsrc	00010FCC	00014A2E	Hint/Name RVA	00D6 _controlfp	
SECTION detour	00010FD0	00014A3C	Hint/Name RVA	0330 wcsncpy	
IMPORT Name Table	00010FD4	00000000	End of Imports	msvcrt.dll	
<ul> <li>IMPORT Hints/Names &amp; DLL Names</li> </ul>	00010F20	80000001	Ordinal	0001	
- IMPORT Directory Table	00010F24	00000000	End of Imports	evil.dll 🙆	

Notice in the .detour section at 1 that the new import table contains evil.dll, seen at 2, which will now be loaded whenever **Notepad** is launched, as it continues to operate as usual.

# **APC Injection**

Launcher

**Process Injection** 

**Process Hollowing** 

**Hook Injection** 

Detours

1

We saw that by creating a thread using **CreateRemoteThread**, we can invoke functionality in a remote process.

However, thread creation requires overhead, so it would be more efficient to invoke a function on an existing thread. This capability exists in Windows as the **asynchronous procedure call (APC)**.

Every thread has a queue of **APC**s attached to it, which are processed when the thread is in an *alertable state*, e.g. when they call **SleepEx**, **WaitForSingleObjectEx**, and **WaitForMultipleObjectsEx**. These functions give the thread a chance to process its waiting APCs.

When the APC queue is complete, the thread continues along its regular execution path.

Malware uses APCs to preempt threads in an alertable state and get executed.

APCs come in two forms:

- ▷ kernel-mode APC: generated for the system or a driver.
- ▷ **user-mode APC**: generated for an application.

Malware generates user-mode APCs from both kernel and user space using APC injection.

# APC Injection: APC Injection from User Space

From user space, another thread can queue a function to be invoked in a remote thread, using the API function **QueueUserAPC**, with these parameters:

- ▷ **pfnAPC**,
- hThread, and
- ▷ dwData.

A call to **QueueUserAPC** is a request for the thread whose handle is **hThread** to run the function **pfnAPC** with the parameter **dwData**.

Because a thread must be in an alertable state in order to run a user-mode APC, malware will look to target threads in processes that are likely to go into that state.

Luckily for the malware analyst, **WaitForSingleObjectEx** is the most common call in the Windows API, and there are usually many threads in the alertable state.

The following example shows APC injection from a user-mode application, i.e., how malware can use **QueueUserAPC** to force a DLL to be loaded in the context of another process:

00401DA9	push	[esp+4+dwThreadId]	; dwThreadId
00401DAD	push	0	; bInheritHandle
00401DAF	push	10h	; dwDesiredAccess
00401DB1	call	ds:OpenThread	
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	; pfnAPC
00401DC8	call	ds:QueueUserAPC	

Before we arrive at this code, the malware has already picked a target thread.

During analysis, you can find thread-targeting code by looking for API calls such as **CreateToolhelp32Snapshot**, **Process32First**, and **Process32Next** for the malware to find the target process.

These API calls will often be followed by calls to **Thread32First** and **Thread32Next**, which will be in a loop looking to target a thread contained in the target process.

Alternatively, malware can also use **Nt/ZwQuerySystemInformation** with the **SYSTEM\_PROCESS\_INFORMATION** information class to find the target process. Once a target-thread identifier is obtained, the malware uses it to open a handle to the thread.

In this example, the malware is looking to force the thread to load a DLL in the remote process, so you see a call to **QueueUserAPC** with the **pfnAPC** set to **LoadLibraryA**.

The parameter to be sent to **LoadLibraryA** will be contained in **dwData** (in this example, that was set to the DLL **dbnet.dll** earlier in the code). Once this APC is queued and the thread goes into an alertable state, **LoadLibraryA** will be called by the remote thread, causing the target process to load **dbnet.dll**.

In this example, the malware targeted **svchost.exe**, which is a popular target for APC injection because its threads are often in an alertable state. Malware may APC-inject into every thread of **svchost.exe** just to ensure that execution occurs quickly.

Malware drivers and rootkits often wish to execute code in user space, but there is no easy way for them to do it.

One method they use is to perform APC injection from kernel space to get their code execution in user space.

A malicious driver can build an APC and dispatch a thread to execute it in a user-mode process (most often **svchost.exe**). APCs of this type often consist of shellcode.

Device drivers leverage two major functions in order to utilize APCs: **KeInitializeApc** and **KeInsertQueueApc**.

## **APC Injection: APC Injection from Kernel Space**

The following example shows an example of these functions in use in a rootkit:

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:KeInitia	alizeApc
000119D5	cmp	edi, ebx	
000119D7	jz	short loc_2	19EA
000119D9	push	ebx	
000119DA	push	[ebp+arg_C]	]
000119DD	push	[ebp+arg_8]	]
000119E0	push	esi	
000119E1	call	edi	KeInsertQueueApc

# APC Injection: APC Injection from Kernel Space

The APC must be initialized with a call to **KeInitializeApc**. If the 6th parameter (**NormalRoutine**) is non-zero, with the 7th parameter (**ApcMode**) set to 1, then we are looking at a user-mode type.

**KeInitializeAPC** initializes a **KAPC** structure, which must be passed to **KeInsertQueueApc** to place the APC object in the target thread's corresponding APC queue. Once **KeInsertQueueApc** is successful, the APC will be queued to run.

In this example, the malware targeted **svchost.exe**, but to make that determination, we would need to trace back the second-to-last parameter pushed on the stack to **KeInitializeApc**. This parameter contains the thread that will be injected.

In this case, it is contained in **arg\_0**. Therefore, we would need to look back in the code to check how **arg\_0** was set to see that **svchost.exe**'s threads were targeted.

# Next: Data Encoding in Malware