恶意代码分析与防治技术

第12章 隐蔽执行技术

（Covert Malware Lauching）

王志
zwang@nankai.edu.cn
南开大学 网络空间安全学院

2023-2024学年

主观题 10分

设置

黑客有哪些方法可以隐蔽的执行恶意代码，使被感染的用户难以发现有恶意代码被执行?

作答

知识点

- 启动器（Launchers）
- 进程注入（Process Injection）
  - 重点：DLL注入、直接注入
- 进程替换（Process Replacement）
  - 难点：Suspended State
- Hook注入（Hook Injection）
  - 难点：Local Hook、Remote Hook
- Detours技术
  - 难点：R77
- APC注入（APC Injection）
  - 难点：Alertable State

启动器（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

多选题 10分

设置

启动器（Launcher）的功能包括哪些？

A 解密

B 隐蔽启动

C 权限提升

D 解压缩

提交

南開大學 Nankai University

进程注入（Process Injection）

主观题 10分

设置

讨论题：什么是进程注入（Process Injection）？

正常使用主观题需2.0以上版本雨课堂

作答

南開大學 Nankai University

进程注入（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
- 常见的API函数:
  - **VirtualAllocEx** to allocate space
  - **WriteProcessMemory** to write to it

进程注入

Process Injection

Sub-techniques (12)

Adversaries may inject code into processes in order to evade process-based defenses as well as possibly elevate privileges. Process injection is a method of executing arbitrary code in the address space of a separate live process. Running code in the context of another process may allow access to the process's memory, system/network resources, and possibly elevated privileges. Execution via process injection may also evade detection from security products since the execution is masked under a legitimate process.

There are many different ways to inject code into a process, many of which abuse legitimate functionalities. These implementations exist for every major OS but are typically platform specific.

More sophisticated samples may perform multiple process injections to segment modules and further evade detection, utilizing named pipes or other inter-process communication (IPC) mechanisms as a communication channel.

ID: T1055

Sub-techniques: T1055.001, T1055.002, T1055.003, T1055.004, T1055.005, T1055.008, T1055.009, T1055.011, T1055.012, T1055.013, T1055.014, T1055.015

ⓘ Tactics: Defense Evasion, Privilege Escalation

ⓘ Platforms: Linux, Windows, macOS

ⓘ Defense Bypassed: Anti-virus, Application control

Contributors: Anastasios Pingios; Christiaan Beek, @ChristiaanBeek; Ryan Becwar

Version: 1.3

Created: 31 May 2017

Last Modified: 30 March 2023

Version Permalink

进程注入

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

DLL 注入

- 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

获取权限

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

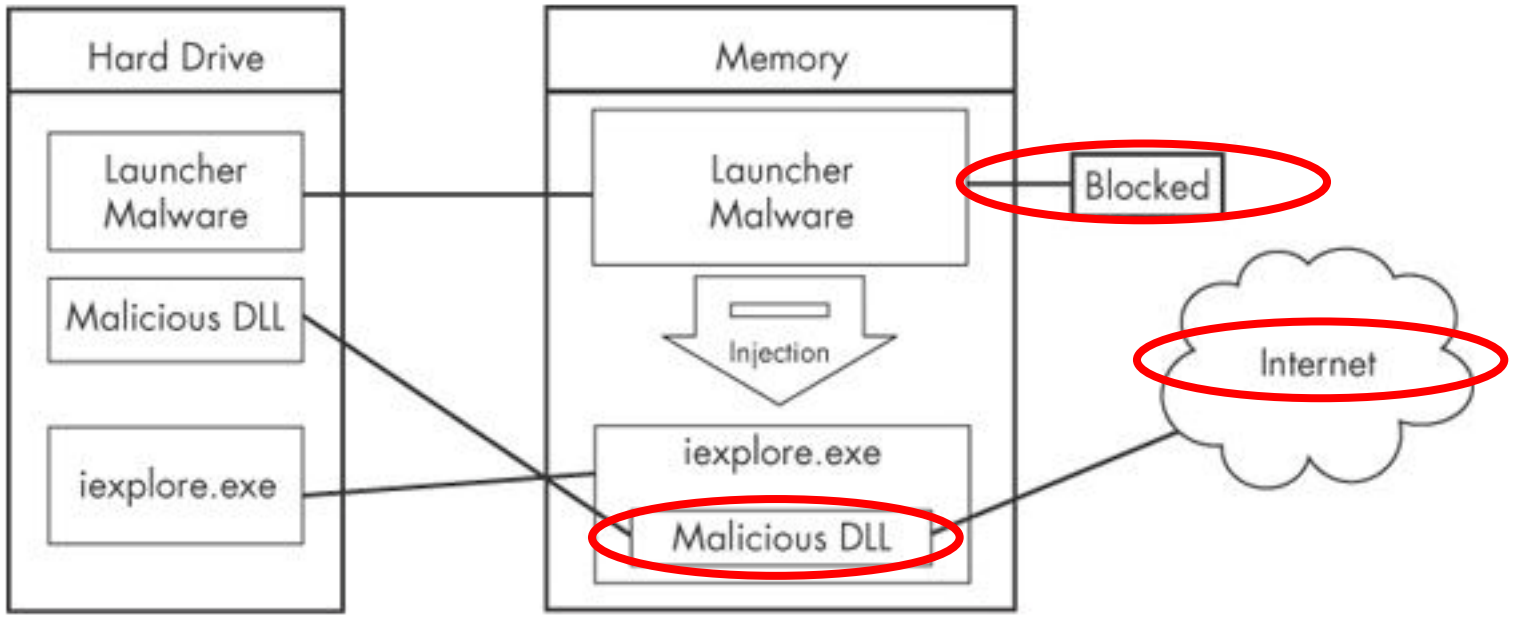


Figure 13-1. DLL injection—the launcher malware cannot access the Internet until it injects into iexplore.exe.

进程遍历

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

创建远程线程

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

```
004076DB   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-1100]
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
```

DLL 注入

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

DLL注入

Process Injection: Dynamic-link Library Injection

Other sub-techniques of Process Injection (12)

Adversaries may inject dynamic-link libraries (DLLs) into processes in order to evade process-based defenses as well as possibly elevate privileges. DLL injection is a method of executing arbitrary code in the address space of a separate live process.

DLL injection is commonly performed by writing the path to a DLL in the virtual address space of the target process before loading the DLL by invoking a new thread. The write can be performed with native Windows API calls such as `VirtualAllocEx` and `WriteProcessMemory`, then invoked with `CreateRemoteThread` (which calls the `LoadLibrary` API responsible for loading the DLL). [1]

Variations of this method such as reflective DLL injection (writing a self-mapping DLL into a process) and memory module (map DLL when writing into process) overcome the address relocation issue as well as the additional APIs to invoke execution (since these methods load and execute the files in memory by manually preforming the function of `LoadLibrary`).[2][1]

Another variation of this method, often referred to as Module Stomping/Overloading or DLL Hollowing, may be leveraged to conceal injected code within a process. This method involves loading a legitimate DLL into a remote process then manually overwriting the module's `AddressOfEntryPoint` before starting a new thread in the target process.[3] This variation allows attackers to hide malicious injected code by potentially backing its execution with a legitimate DLL file on disk.[4]

Running code in the context of another process may allow access to the process's memory, system/network resources, and possibly elevated privileges. Execution via DLL injection may also evade detection from security products since the execution is masked under a legitimate process.

ID: T1055.001

Sub-technique of: T1055

Tactics: Defense Evasion, Privilege Escalation

Platforms: Windows

Permissions Required: User

Defense Bypassed: Anti-virus, Application control

Contributors: Boominathan Sundaram

Version: 1.3

Created: 14 January 2020

Last Modified: 11 August 2023

Version Permalink

直接注入

- 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

直接注入

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

进程注入

Procedure Examples

ID	Name	Description
C0028	2015 Ukraine Electric Power Attack	During the 2015 Ukraine Electric Power Attack, Sandworm Team loaded BlackEnergy into svchost.exe, which then launched iexplore.exe for their C2. [1]
S0469	ABK	ABK has the ability to inject shellcode into svchost.exe.[2]
S0331	Agent Tesla	Agent Tesla can inject into known, vulnerable binaries on targeted hosts.[3]
S1074	ANDROMEDA	ANDROMEDA can inject into the `wuauclt.exe` process to perform C2 actions.[4]
G0050	APT32	APT32 malware has injected a Cobalt Strike beacon into Rundll32.exe.[5]
G0067	APT37	APT37 injects its malware variant, ROKRAT, into the cmd.exe process.[6]
G0096	APT41	APT41 malware TIDYELF loaded the main WINTERLOVE component by injecting it into the iexplore.exe process.[7]
S0438	Attor	Attor's dispatcher can inject itself into running processes to gain higher privileges and to evade detection.[8]
S0347	AuditCred	AuditCred can inject code from files to other running processes.[9]
S0473	Avenger	Avenger has the ability to inject shellcode into svchost.exe.[2]
S0093	Backdoor.Oldrea	Backdoor.Oldrea injects itself into explorer.exe.[10][11]
S1081	BADHATCH	BADHATCH can inject itself into an existing explorer.exe process by using `RtlCreateUserThread`.[12][13]
S0534	Bazar	Bazar can inject code through calling `VirtualAllocExNuma`.[14]
S0470	BBK	BBK has the ability to inject shellcode into svchost.exe.[2]

多选题 10分

设置

进程注入的优点有哪些？

A 获得被注入进程的权限

B 伪装恶意行为

C 可以有效穿透防火墙、躲避检测工具

D 影响被注入进程的正常执行

提交

多选题 10分

设置

进程注入需要用到的API函数有哪些？

A VirtualAllocEx

B WriteProcessMemory

C CreateRemoteThread

D OpenProcess

提交

南開大學 Nankai University

投票　最多可选1项

设置

恶意代码通常选择DLL注入还是直接注入？

A　DLL注入

B　直接注入

提交

南開大學 Nankai University

主观题 10分

设置

如何检测进程的注入?

作答

检测进程注入

Detection

ID	Data Source	Data Component	Detects
DS0022	File 1	File Metadata	Monitor for contextual data about a file, which may include information such as name, the content (ex: signature, headers, or data/media), user/owner, permissions, etc.
		File Modification	Monitor for changes made to files that may inject code into processes in order to evade process-based defenses as well as possibly elevate privileges.
DS0011	Module 2	Module Load	Monitor DLL/PE file events, specifically creation of these binary files as well as the loading of DLLs into processes. Look for DLLs that are not recognized or not normally loaded into a process.
DS0009	Process 3	OS API Execution	Monitoring Windows API calls indicative of the various types of code injection may generate a significant amount of data and may not be directly useful for defense unless collected under specific circumstances for known bad sequences of calls, since benign use of API functions may be common and difficult to distinguish from malicious behavior. Windows API calls such as `CreateRemoteThread`, `SuspendThread`/`SetThreadContext`/`ResumeThread`, `QueueUserAPC`/`NtQueueApcThread`, and those that can be used to modify memory within another process, such as `VirtualAllocEx`/`WriteProcessMemory`, may be used for this technique.[79] Monitoring for Linux specific calls such as the ptrace system call should not generate large amounts of data due to their specialized nature, and can be a very effective method to detect some of the common process injection methods.[80] [81] [82] [83]
		Process Access	Monitor for processes being viewed that may inject code into processes in order to evade process-based defenses as well as possibly elevate privileges.
		Process Metadata	Monitor for process memory inconsistencies, such as checking memory ranges against a known copy of the legitimate module.[84]
		Process Modification	Monitor for changes made to processes that may inject code into processes in order to evade process-based defenses as well as possibly elevate privileges.

主观题 10分

设置

如何阻止或者缓解进程注入?

作答

缓解进程注入

Mitigations

ID	Mitigation	Description
M1040	Behavior Prevention on Endpoint 1	Some endpoint security solutions can be configured to block some types of process injection based on common sequences of behavior that occur during the injection process. For example, on Windows 10, Attack Surface Reduction (ASR) rules may prevent Office applications from code injection. [78]
M1026	Privileged Account Management 2	Utilize Yama (ex: /proc/sys/kernel/yama/ptrace_scope) to mitigate ptrace based process injection by restricting the use of ptrace to privileged users only. Other mitigation controls involve the deployment of security kernel modules that provide advanced access control and process restrictions such as SELinux, grsecurity, and AppArmor.

进程替换（Process Replacement）

主观题 10分

设置

讨论题：什么是进程替换（Process Replacement）？与进程注入有什么区别。

正常使用主观题需2.0以上版本雨课堂

作答

南開大學 Nankai University

进程替换

- **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

进程替换

- 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.

进程替换

Process Injection: Process Hollowing

Other sub-techniques of Process Injection (12)

Adversaries may inject malicious code into suspended and hollowed processes in order to evade process-based defenses. Process hollowing is a method of executing arbitrary code in the address space of a separate live process.

Process hollowing is commonly performed by creating a process in a suspended state then unmapping/hollowing its memory, which can then be replaced with malicious code. A victim process can be created with native Windows API calls such as `CreateProcess`, which includes a flag to suspend the processes primary thread. At this point the process can be unmapped using APIs calls such as `ZwUnmapViewOfSection` or `NtUnmapViewOfSection` before being written to, realigned to the injected code, and resumed via `VirtualAllocEx`, `WriteProcessMemory`, `SetThreadContext`, then `ResumeThread` respectively.[1][2]

This is very similar to Thread Local Storage but creates a new process rather than targeting an existing process. This behavior will likely not result in elevated privileges since the injected process was spawned from (and thus inherits the security context) of the injecting process. However, execution via process hollowing may also evade detection from security products since the execution is masked under a legitimate process.

ID: T1055.012

Sub-technique of: T1055

Tactics: Defense Evasion, Privilege Escalation

Platforms: Windows

Permissions Required: User

Defense Bypassed: Anti-virus, Application control

Version: 1.3

Created: 14 January 2020

Last Modified: 11 August 2023

Version Permalink

进程替换

Procedure Examples

ID	Name	Description
S0331	Agent Tesla	Agent Tesla has used process hollowing to create and manipulate processes through sections of unmapped memory by reallocating that space with its malicious code.[3]
S0373	Astaroth	Astaroth can create a new process in a suspended state from a targeted legitimate process in order to unmap its memory and replace it with malicious code.[4][5]
S0344	Azorult	Azorult can decrypt the payload into memory, create a new suspended process of itself, then inject a decrypted payload to the new process and resume new process execution.[6]
S0128	BADNEWS	BADNEWS has a command to download an .exe and use process hollowing to inject it into a new process.[7][8]
S0234	Bandook	Bandook has been launched by starting iexplore.exe and replacing it with Bandook's payload.[9][10][11]
S0534	Bazar	Bazar can inject into a target process including Svchost, Explorer, and cmd using process hollowing.[12][13]
S0127	BBSRAT	BBSRAT has been seen loaded into msiexec.exe through process hollowing to hide its execution.[14]
S0660	Clambling	Clambling can execute binaries through process hollowing.[15]
S0154	Cobalt Strike	Cobalt Strike can use process hollowing for execution.[16][17]
S0354	Denis	Denis performed process hollowing through the API calls CreateRemoteThread, ResumeThread, and Wow64SetThreadContext.[18]

单选题 10分

设置

以下哪项需要用到进程的Suspended状态？

A Privilege escalation

B Process replacement

C DLL injection

D Direct injection

提交

南開大學 Nankai University

单选题 10分

设置

以下哪项技术需要大量的定制化代码（customized code）？

A Privilege escalation

B Process replacement

C DLL injection

D Direct injection

提交

南開大學 Nankai University

以下哪项技术需要调用FindResource函数?

A Privilege escalation

B Process replacement

C DLL injection

D Direct injection

E Resource extraction

提交

多选题 10分

设置

以下哪项技术需要调用CreateRemoteThread 函数?

A Privilege escalation

B Process replacement

C DLL injection

D Direct injection

E Resource extraction

提交

南開大學 Nankai University

主观题 10分

设置

如何检测和缓解进程替换攻击?

作答

南開大學 Nankai University

进程替换攻击

Mitigations

ID	Mitigation	Description
M1040	Behavior Prevention on Endpoint	Some endpoint security solutions can be configured to block some types of process injection based on common sequences of behavior that occur during the injection process.

Detection

ID	Data Source	Data Component	Detects
DS0009	Process	OS API Execution	Monitoring Windows API calls indicative of the various types of code injection may generate a significant amount of data and may not be directly useful for defense unless collected under specific circumstances for known bad sequences of calls, since benign use of API functions may be common and difficult to distinguish from malicious behavior. Windows API calls such as `CreateRemoteThread`, `SuspendThread`/`SetThreadContext`/`ResumeThread`, and those that can be used to modify memory within another process, such as `VirtualAllocEx`/`WriteProcessMemory`, may be used for this technique.[2]
		Process Access	Monitor for processes being viewed that may inject malicious code into suspended and hollowed processes in order to evade process-based defenses.
		Process Creation	Monitor for newly executed processes that may inject malicious code into suspended and hollowed processes in order to evade process-based defenses. Adversaries may start legitimate processes and then use their memory space to run malicious code. This analytic looks for common Windows processes that have been abused this way in the past; when the processes are started for this purpose they may not have the standard parent that we would expect. This list is not exhaustive, and it is possible for cyber actors to avoid this discepency. These signatures only work if Sysmon reports the parent process, which may not always be the case if the parent dies before sysmon processes the event. **Analytic 1 - Processes Started From Irregular Parents** `mismatch_processes = filter processes where (parent_exe exists AND (exe="smss.exe" AND (parent_exe!="smss.exe" AND parent_exe!="System") OR (exe="csrss.exe" AND (parent_exe!="smss.exe" AND parent_exe!="svchost.exe")) OR (exe="wininit.exe" AND parent_exe!="smss.exe") OR (exe="winlogon.exe" AND parent_exe!="smss.exe") OR (exe="lsass.exe" AND (parent_exe!="wininit.exe" AND parent_exe!="winlogon.exe")) OR (exe="LogonUI.exe" AND (parent_exe!="winlogon.exe" AND parent_exe!="wininit.exe")) OR (exe="services.exe" AND parent_exe!="wininit.exe") OR (exe="spoolsv.exe" AND parent_exe!="services.exe") OR (exe="taskhost.exe" AND (parent_exe!="services.exe" AND parent_exe!="svchost.exe")) OR (exe="taskhostw.exe" AND (parent_exe!="services.exe" AND parent_exe!="svchost.exe")) OR (exe="userinit.exe" AND (parent_exe!="dwm.exe" AND parent_exe!="winlogon.exe"))`
		Process Modification	Monitor for changes made to processes that may inject malicious code into suspended and hollowed processes in order to evade process-based defenses.

References

投票 最多可选1项

设置

进程替换是否只能对新创建的进程进行替换？ 已经运行的进程是否可以进行替换?

A 可以

B 不可以

C 不知道

D 应该可以

提交

南開大學 Nankai University

线程执行劫持（Thread Execution Hijacking）

Process Injection: Thread Execution Hijacking

Other sub-techniques of Process Injection (12)

Adversaries may inject malicious code into hijacked processes in order to evade process-based defenses as well as possibly elevate privileges. Thread Execution Hijacking is a method of executing arbitrary code in the address space of a separate live process.

Thread Execution Hijacking is commonly performed by suspending an existing process then unmapping/hollowing its memory, which can then be replaced with malicious code or the path to a DLL. A handle to an existing victim process is first created with native Windows API calls such as `OpenThread`. At this point the process can be suspended then written to, realigned to the injected code, and resumed via `SuspendThread` , `VirtualAllocEx`, `WriteProcessMemory`, `SetThreadContext`, then `ResumeThread` respectively.[1]

This is very similar to Process Hollowing but targets an existing process rather than creating a process in a suspended state.

Running code in the context of another process may allow access to the process's memory, system/network resources, and possibly elevated privileges. Execution via Thread Execution Hijacking may also evade detection from security products since the execution is masked under a legitimate process.

ID: T1055.003

Sub-technique of: T1055

ⓘ Tactics: Defense Evasion, Privilege Escalation

ⓘ Platforms: Windows

ⓘ Permissions Required: User

ⓘ Defense Bypassed: Anti-virus, Application control

Version: 1.1

Created: 14 January 2020

Last Modified: 18 October 2021

Version Permalink

Hook 注入（Hook Injection）

主观题 10分

设置

讨论题：什么是Hook？是否只有恶意代码在使用Hook？

作答

Hook 注入

- Windows是消息驱动的
  - 例如游戏中的鼠标输入、键盘输入，通过消息机制与游戏程序交互
- Windows hooks are used to intercept messages destined for applications（回调函数，消息响应函数）
- 恶意的hook注入
  - 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

SetWindowsEx

- idHook 挂钩类型
  - WH_CALLWNDPROC 在系统将消息发送到目标窗口过程之前监视消息
  - WH_CALLWNDPROCRET 在目标窗口过程处理消息后监视消息
  - WH_CBT接收对 CBT 应用程序有用的通知
  - WH_DEBUG 用于调试其他挂钩过程的挂钩过程
  - WH_FOREGROUNDIDLE 在应用程序的前台线程变为空闲状态时调用的挂钩过程

Syntax

C++

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

WH_KEYBOARD 2	安装用于监视击键消息的挂钩过程。 有关详细信息，请参阅 KeyboardProc 挂钩过程。
WH_KEYBOARD_LL 13	安装用于监视低级别键盘输入事件的挂钩过程。 有关详细信息，请参阅 [LowLevelKeyboardProc] (/windows/win32/winmsg/lowlevelkeyboardproc) 挂钩过程。
WH_MOUSE 7	安装监视鼠标消息的挂钩过程。 有关详细信息，请参阅 MouseProc 挂钩过程。
WH_MOUSE_LL 14	安装用于监视低级别鼠标输入事件的挂钩过程。 有关详细信息，请参阅 LowLevelMouseProc 挂钩过程。
WH_MSGFILTER -1	安装挂钩过程，用于监视由于对话框、消息框、菜单或滚动条中的输入事件而生成的消息。 有关详细信息，请参阅 MessageProc 挂钩过程。
WH_SHELL 10	安装一个挂钩过程，用于接收对 shell 应用程序有用的通知。 有关详细信息，请参阅 ShellProc 挂钩过程。
WH_SYSMSGFILTER 6	安装挂钩过程，用于监视由于对话框、消息框、菜单或滚动条中的输入事件而生成的消息。 挂钩过程监视与调用线程位于同一桌面中的所有应用程序的消息。 有关详细信息，请参阅 SysMsgProc 挂钩过程。

https://learn.microsoft.com/zh-cn/windows/win32/api/winuser/nf-winuser-setwindowshookexa

SetWindowsEx

- lpfn
  - 指向挂钩过程的指针。
- hmod
  - DLL 的句柄，其中包含 lpfn 参数指向的挂钩过程。
- dwThreadId
  - 要与挂钩过程关联的线程的标识符
- 返回值 HHOOK
  - 如果函数成功，则返回值是挂钩过程的句柄。
  - 如果函数失败，则返回值为 NULL。
  - UnhookWindowsHookEx()

C++

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

C++

```
BOOL UnhookWindowsHookEx(
  [in] HHOOK hhk
);
```

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()`

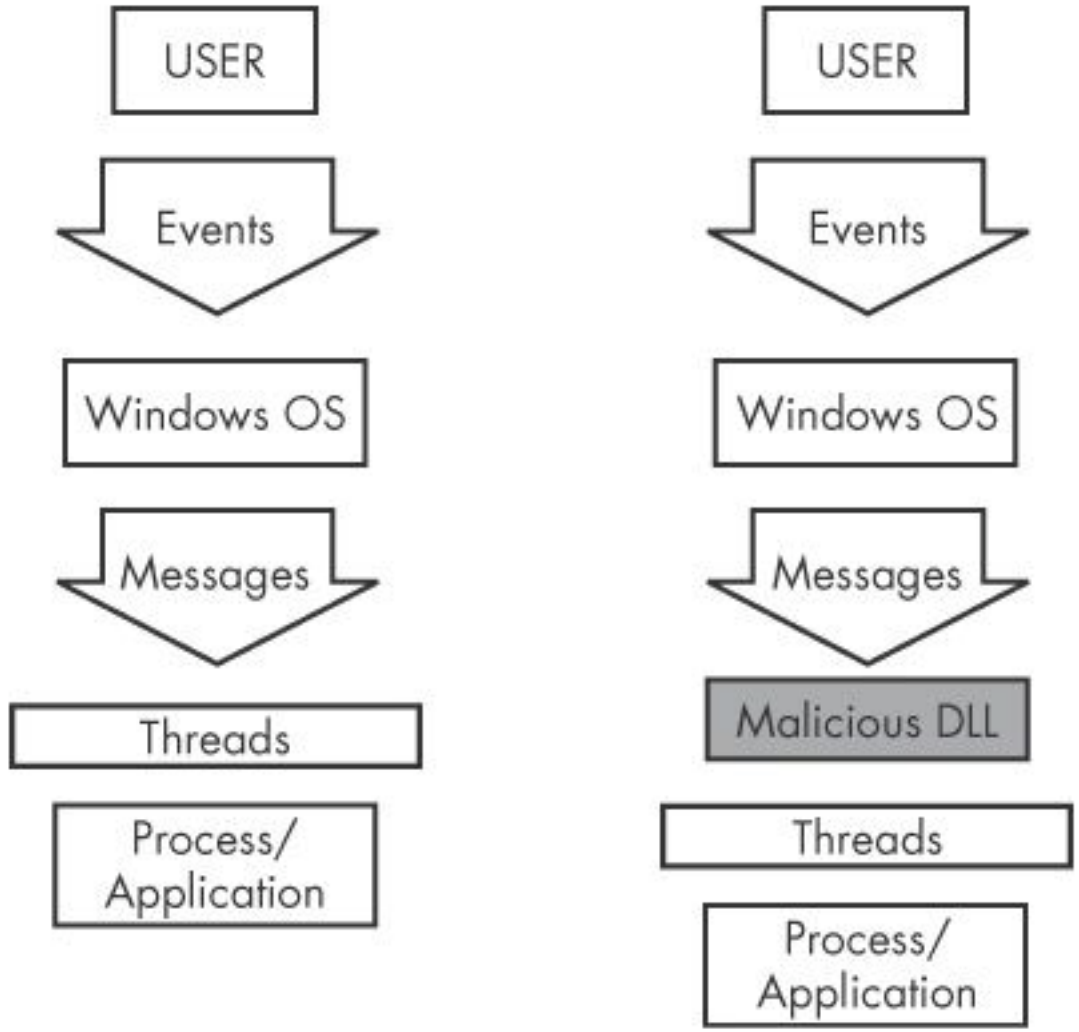


Figure 13-3. Event and message flow in Windows with and without hook injection

本地钩子和远程钩子

- *Local hooks* observe or manipulate messages destined for an internal process
  - 处理钩子所在进程的消息
- *Remote hooks* observe or manipulate messages destined for a remote process (another process on the computer)
  - 处理其它进程的消息

High-Level and Low-Level Remote Hooks

- *High-level remote hooks*系统钩子
  - 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
  - 系统开销大
- *Low-level remote hooks* 线程钩子
  - 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`**

WH_KEYBOARD 2	安装用于监视击键消息的挂钩过程。 有关详细信息，请参阅 KeyboardProc 挂钩过程。
WH_KEYBOARD_LL 13	安装用于监视低级别键盘输入事件的挂钩过程。 有关详细信息，请参阅 [LowLevelKeyboardProc] (/windows/win32/winmsg/lowlevelkeyboardproc) 挂钩过程。
WH_MOUSE 7	安装监视鼠标消息的挂钩过程。 有关详细信息，请参阅 MouseProc 挂钩过程。
WH_MOUSE_LL 14	安装用于监视低级别鼠标输入事件的挂钩过程。 有关详细信息，请参阅 LowLevelMouseProc 挂钩过程。
WH_MSGFILTER -1	安装挂钩过程，用于监视由于对话框、消息框、菜单或滚动条中的输入事件而生成的消息。 有关详细信息，请参阅 MessageProc 挂钩过程。
WH_SHELL 10	安装一个挂钩过程，用于接收对 shell 应用程序有用的通知。 有关详细信息，请参阅 ShellProc 挂钩过程。
WH_SYSMSGFILTER 6	安装挂钩过程，用于监视由于对话框、消息框、菜单或滚动条中的输入事件而生成的消息。 挂钩过程监视与调用线程位于同一桌面中的所有应用程序的消息。 有关详细信息，请参阅 SysMsgProc 挂钩过程。

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

钩子链（hook chain）

- 如果对同一个消息安装了多个钩子，多个钩子构成一个钩子链
- The hook procedure must call ***CallNextHookEx*** to pass execution to the next hook procedure so the system continues to run properly

目标线程

- 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)
- 同一个消息，既有high-level钩子又有low-level钩子，先执行low-level钩子

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
```

目标线程

- 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

单选题 10分

设置

Which technique manipulates messages from a process to itself?

A Local hook

B High-level remote hook

C Low-level remote hook

D Keylogger

提交

南開大學 Nankai University

单选题 10分

设置

以下哪种方法会挂钩所有的线程?

A Local hook

B High-level remote hook

C Low-level remote hook

提交

南開大學 Nankai University

多选题 1分

设置

以下哪个描述是正确的?

A 对于同一事件（如鼠标消息）既安装了线程钩子又安装了系统钩子，那么系统会自动先调用线程钩子，然后调用系统钩子。

B 对同一事件消息可安装多个钩子处理过程，这些勾子处理过程形成了钩子链。

C 当前钩子处理结束后应把钩子信息传递给下一个钩子函数。

D 系统钩子会消耗消息处理时间，降低系统性能。只有在必要的时候才安装钩子，在使用完毕后要及时卸载。

提交

多选题 1分

设置

以下哪个描述是正确的?

A 钩子函数在完成对消息的处理后，要继续传递消息，必须调用CallNextHookEx来传递它，以执行钩子链表下一个钩子函数

B 钩子使用UnhookWindowsHookEx()卸载

C SetWindowsHookEx()把钩子安装到钩子链表中

D 钩子的卸载顺序一定要和安装顺序相反

提交

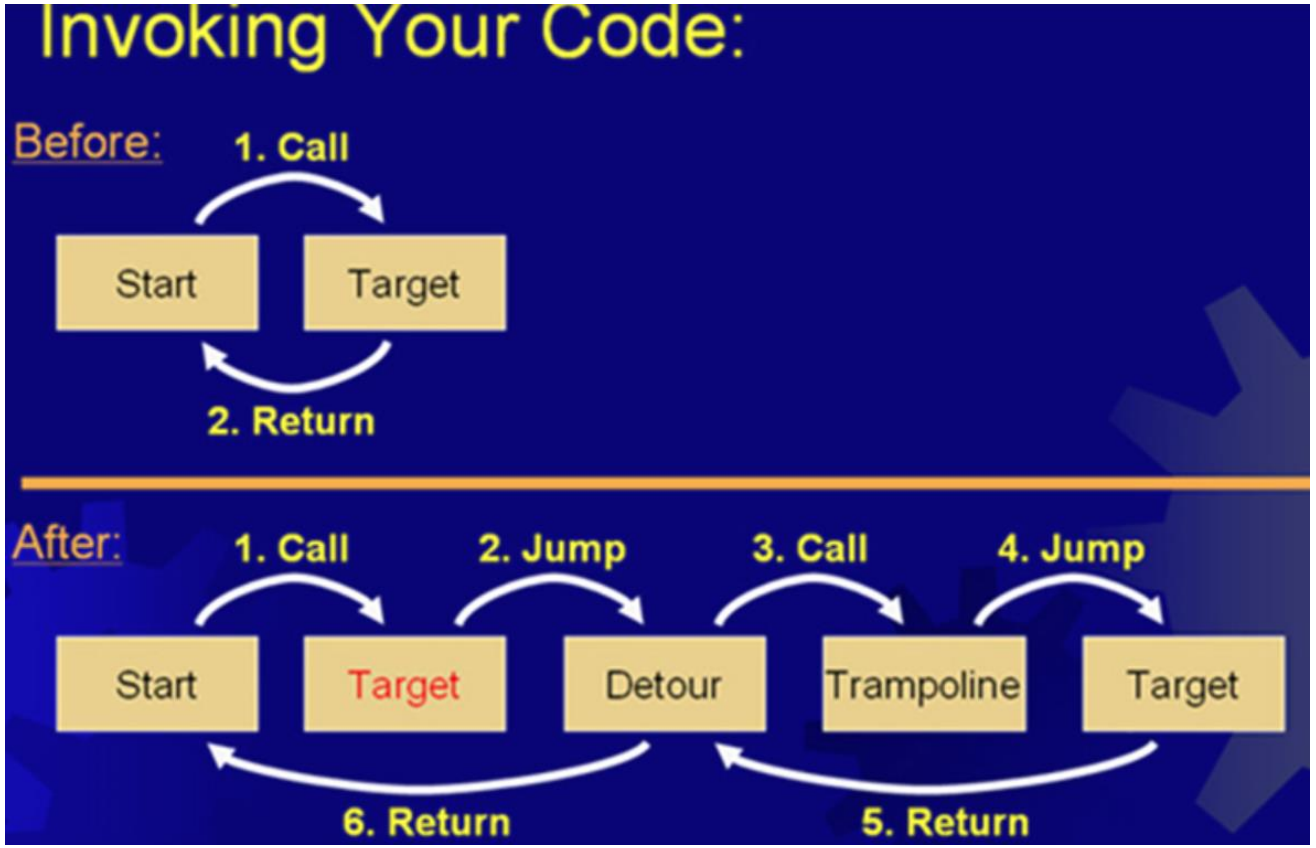
Detours技术

Detours技术

- Detours is a library developed by Microsoft.
  - easily instrument and extend existing OS and application functionality.
  - 截获任意Win32函数调用的工具库
  - 使用无条件转移指令来替换目标函数的最初几条指令，将控制流转移到截获函数。
- Detours library is used by malware authors
  - modify important tables
  - attach DLLs
  - and function hooks

Detours

- Target function, Trampoline function, Detour function



Detouring a Function:

Before:

```
;; Target Function
Target Fun:
  push   ebp            [1 byte]
  mov    ebp,esp        [2 bytes]
  push   ebx            [1 bytes]
  push   esi            [1 byte]
  push   edi
  ....
;; Trampoline Function
Trampoline Fun :
  jmp  Target Fun

;; Detour Function
Detour Fun:
   Do you want to do,in this
   function you can get the
   parameters pass to the Target
   Fun.
  ....
```

After:

```
;; Target Function
Target Fun :
  jmp Detour Fun        [5 bytes]
  push   edi
  ....
;; Trampoline Function
Trampoline Fun :
  push   ebp
  mov    ebp,esp
  push   ebx
  push   esi
  jmp    Target Fun +5
;; Detour Function
Detour Fun :
   Do you want to do,in this
   function you can get the
   parameters pass to the Target
   Fun, and call the Trampoline
   Fun to get the Target Fun
   return Values.
```

Detours技术

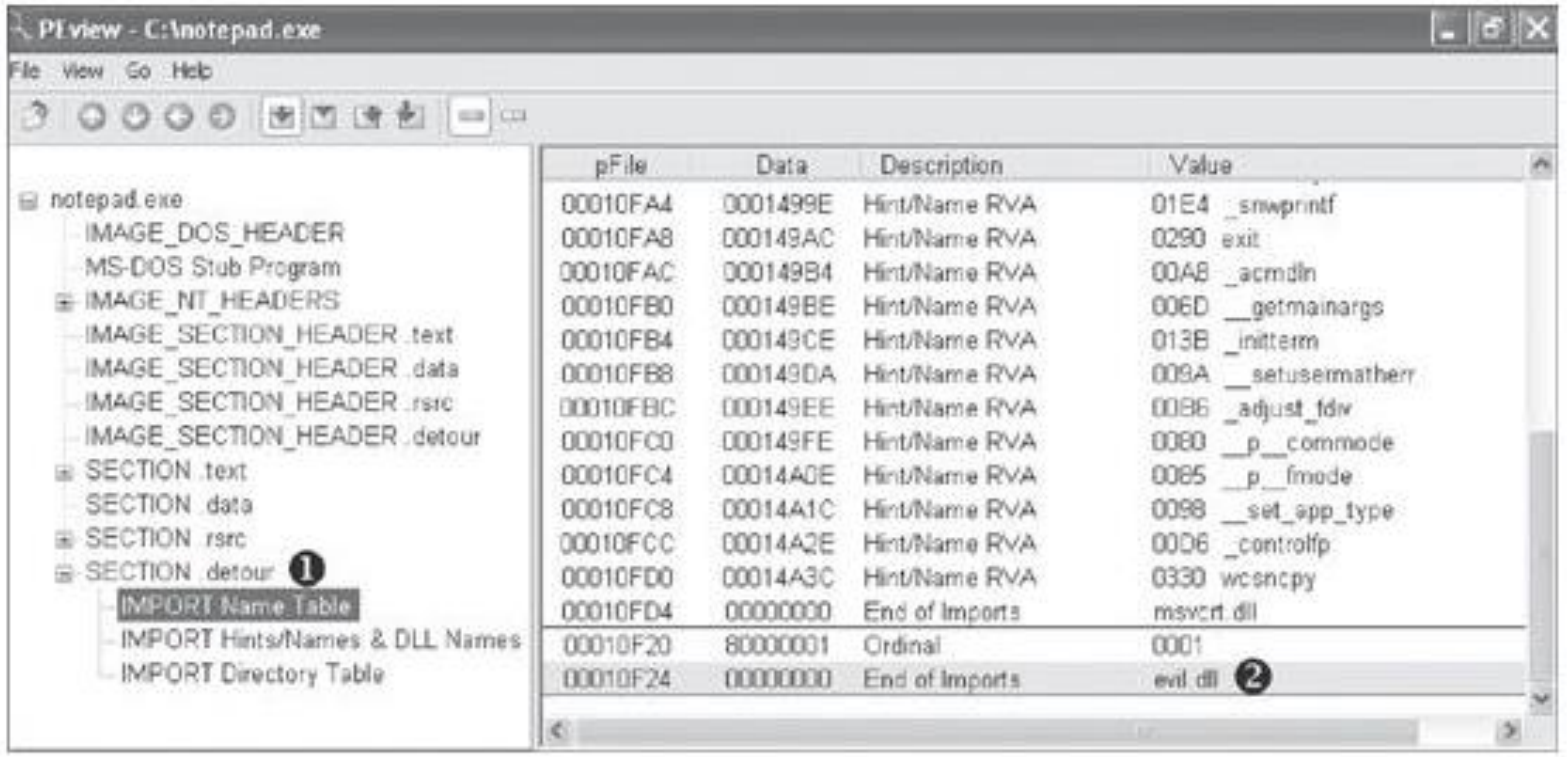


Figure 13-4. A PEview of Detours and the evil.dll

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技术

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

多选题 10分

设置

下面哪种隐蔽方式是通过修改文件实现的？

A Detour

B DLL Injection

C Direct Injection

D Hooks

提交

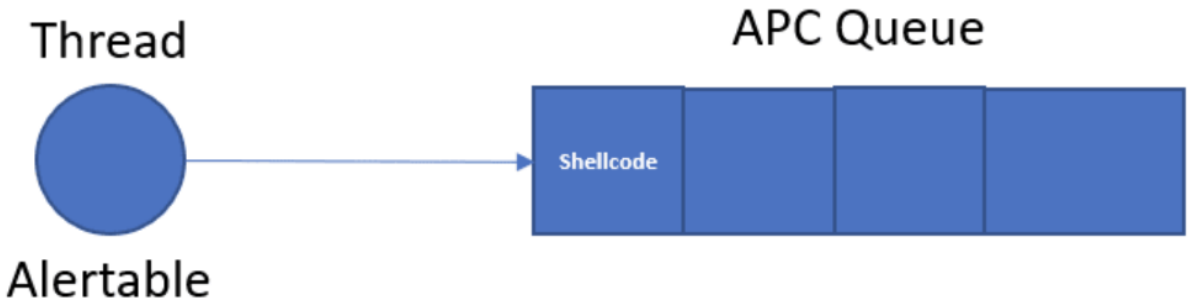
APC注入（APC Injection）

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 注入

- 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 注入

- 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.

APC 注入

- 内核模式APC（Kernel-Mode APC）
  - Generated for the system or a driver
- 用户模式APC（User-Mode APC）
  - Generated for an application
- APC Injection is used in both cases

用户模式APC 注入

- 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

- **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

内核模式APC注入

- 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
```

T1055.004

Process Injection: Asynchronous Procedure Call

Other sub-techniques of Process Injection (12)

Adversaries may inject malicious code into processes via the asynchronous procedure call (APC) queue in order to evade process-based defenses as well as possibly elevate privileges. APC injection is a method of executing arbitrary code in the address space of a separate live process.

APC injection is commonly performed by attaching malicious code to the APC Queue [1] of a process's thread. Queued APC functions are executed when the thread enters an alterable state.[1] A handle to an existing victim process is first created with native Windows API calls such as `OpenThread`. At this point `QueueUserAPC` can be used to invoke a function (such as `LoadLibrayA` pointing to a malicious DLL).

A variation of APC injection, dubbed "Early Bird injection", involves creating a suspended process in which malicious code can be written and executed before the process' entry point (and potentially subsequent anti-malware hooks) via an APC. [2] AtomBombing [3] is another variation that utilizes APCs to invoke malicious code previously written to the global atom table.[4]

Running code in the context of another process may allow access to the process's memory, system/network resources, and possibly elevated privileges. Execution via APC injection may also evade detection from security products since the execution is masked under a legitimate process.

ID: T1055.004

Sub-technique of: T1055

ⓘ Tactics: Defense Evasion, Privilege Escalation

ⓘ Platforms: Windows

ⓘ Defense Bypassed: Anti-virus, Application control

Version: 1.1

Created: 14 January 2020

Last Modified: 18 October 2021

Version Permalink

ACP注入的案例

Procedure Examples

ID	Name	Description
S0438	Attor	Attor performs the injection by attaching its code into the APC queue using NtQueueApcThread API.[5]
S1081	BADHATCH	BADHATCH can inject itself into a new `svchost.exe -k netsvcs` process using the asynchronous procedure call (APC) queue.[6][7]
S1039	Bumblebee	Bumblebee can use asynchronous procedure call (APC) injection to execute commands received from C2.[8]
S0484	Carberp	Carberp has queued an APC routine to explorer.exe by calling ZwQueueApcThread.[9]
G0061	FIN8	FIN8 has injected malicious code into a new svchost.exe process.[10]
S0483	IcedID	IcedID has used `ZwQueueApcThread` to inject itself into remote processes.[11]
S0260	InvisiMole	InvisiMole can inject its code into a trusted process via the APC queue.[12]
S0517	Pillowmint	Pillowmint has used the NtQueueApcThread syscall to inject code into svchost.exe.[13]
S1018	Saint Bot	Saint Bot has written its payload into a newly-created `EhStorAuthn.exe` process using `ZwWriteVirtualMemory` and executed it using
S1085	Sardonic	Sardonic can use the `QueueUserAPC` API to execute shellcode on a compromised machine.[15]
S0199	TURNEDUP	TURNEDUP is capable of injecting code into the APC queue of a created Rundll32 process as part of an "Early Bird injection."[2]

主观题 10分

设置

讨论：如何防治或者缓解APC注入攻击？

作答

APC注入攻击的防治

Mitigations

ID	Mitigation	Description
M1040	Behavior Prevention on Endpoint	Some endpoint security solutions can be configured to block some types of process injection based on common sequences of behavior that occur during the injection process.

Detection

ID	Data Source	Data Component	Detects
DS0009	Process	OS API Execution	Monitoring Windows API calls indicative of the various types of code injection may generate a significant amount of data and may not be directly useful for defense unless collected under specific circumstances for known bad sequences of calls, since benign use of API functions may be common and difficult to distinguish from malicious behavior. Windows API calls such as `SuspendThread`/`SetThreadContext`/`ResumeThread`, `QueueUserAPC`/`NtQueueApcThread`, and those that can be used to modify memory within another process, such as `VirtualAllocEx`/`WriteProcessMemory`, may be used for this technique.[16]
		Process Access	Monitor for processes being viewed that may inject malicious code into processes via the asynchronous procedure call (APC) queue in order to evade process-based defenses as well as possibly elevate privileges.
		Process Modification	Monitor for changes made to processes that may inject malicious code into processes via the asynchronous procedure call (APC) queue in order to evade process-based defenses as well as possibly elevate privileges.

单选题 10分

设置

Which technique only works for threads in an alterable state?

A Detours

B APC

C Hook

D Injection

提交

南開大學 Nankai University

单选题 1分

设置

APC注入只能用于用户空间的程序，不能用在内核空间。

A 正确

B 错误

提交

知识点

- 启动器（Launchers）
- 进程注入（Process Injection）
  - 重点：DLL注入、直接注入
- 进程替换（Process Replacement）
  - 难点：Suspended State
- Hook注入（Hook Injection）
  - 难点：Local Hook、Remote Hook
- Detours技术
  - 难点：R77
- APC注入（APC Injection）
  - 难点：Alertable State

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

2023-2024学年