Blackout: What Really Happened

Jamie Butler and Kris Kendall

MANDIANT
INTELLIGENT INFORMATION SECURITY

Black Hat

Outline

- Code Injection Basics
- User Mode Injection Techniques
- Example Malware Implementations
- Kernel Mode Injection Techniques
- Advanced Code Injection Detection via Raw Memory Analysis

Code Injection Basics

- "Code Injection" refers to techniques used to run code in the context of an existing process
- Motivation:
  - Evasion: Hiding from automated or human detection of malicious code
    - IR personnel hunt for malicious processes
  - Impersonation: Bypassing restrictions enforced on a process level
    - Windows Firewall, etc
    - Pwdump, Sam Juicer

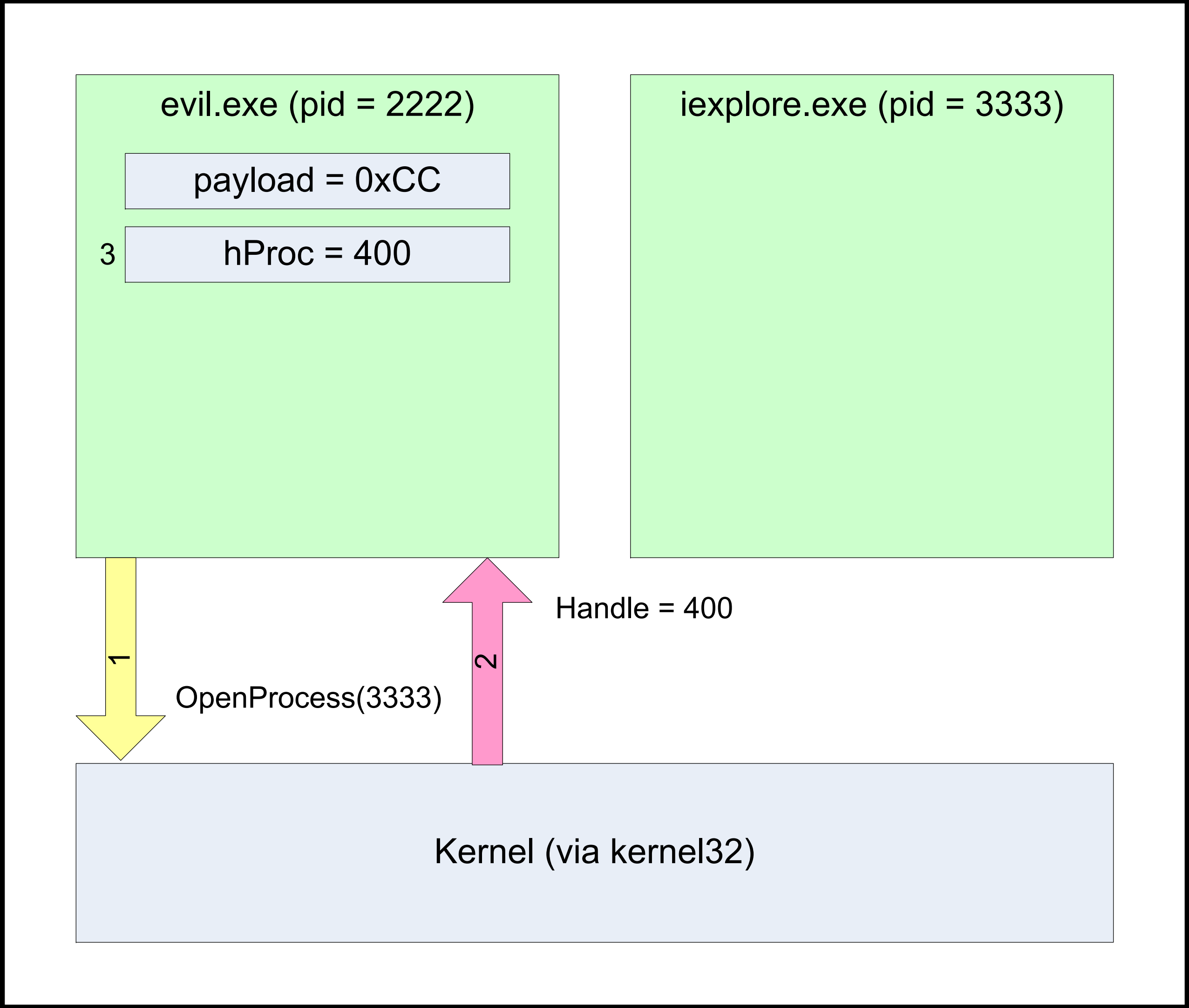
User Mode Injection Techniques

- Techniques
  - Windows API
  - AppInit_Dll
  - Detours

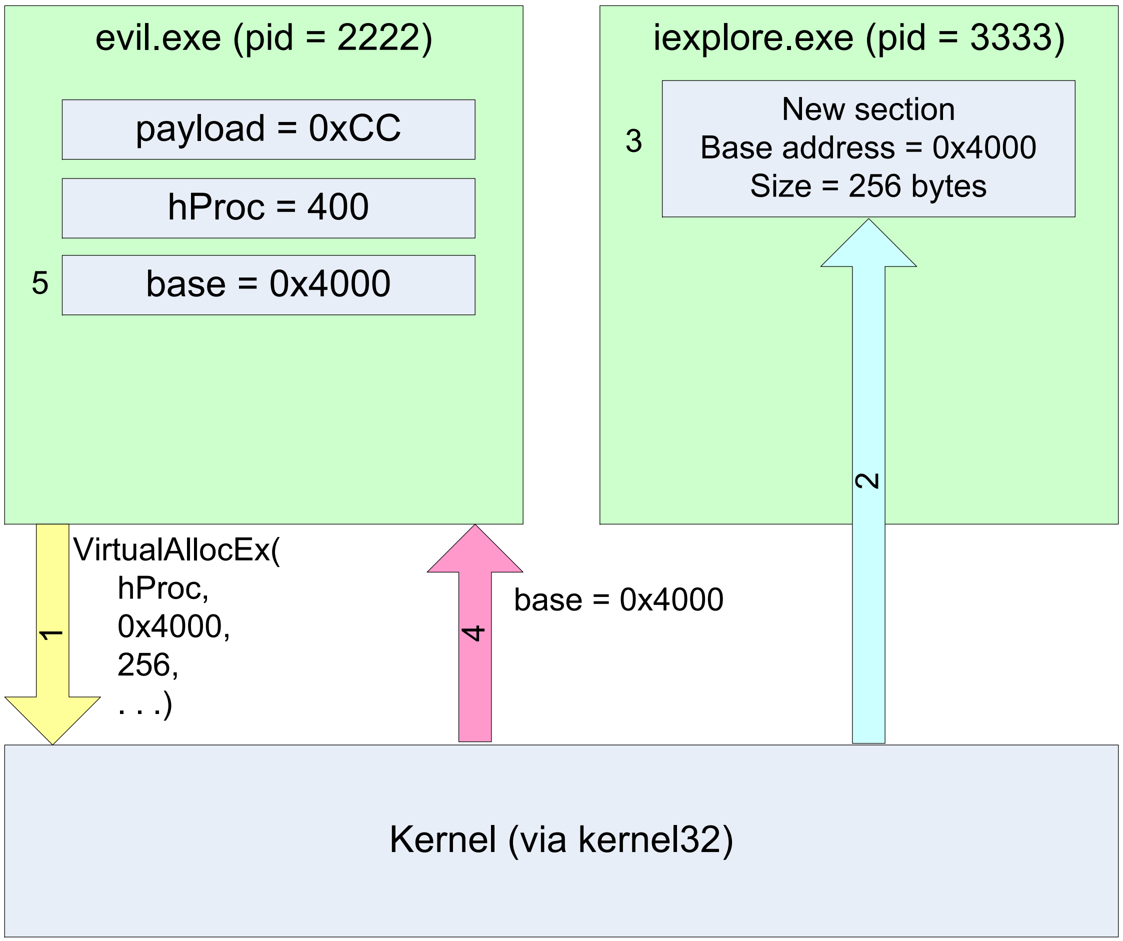
Injecting code via the Windows API

- Somewhat surprisingly, the Windows API provides everything you need for process injection
- Functions:
  - `VirtualAllocEx()`
  - `WriteProcessMemory()`
  - `CreateRemoteThread()`
  - `GetThreadContext()` / `SetThreadContext()`
  - `SetWindowsHookEx()`

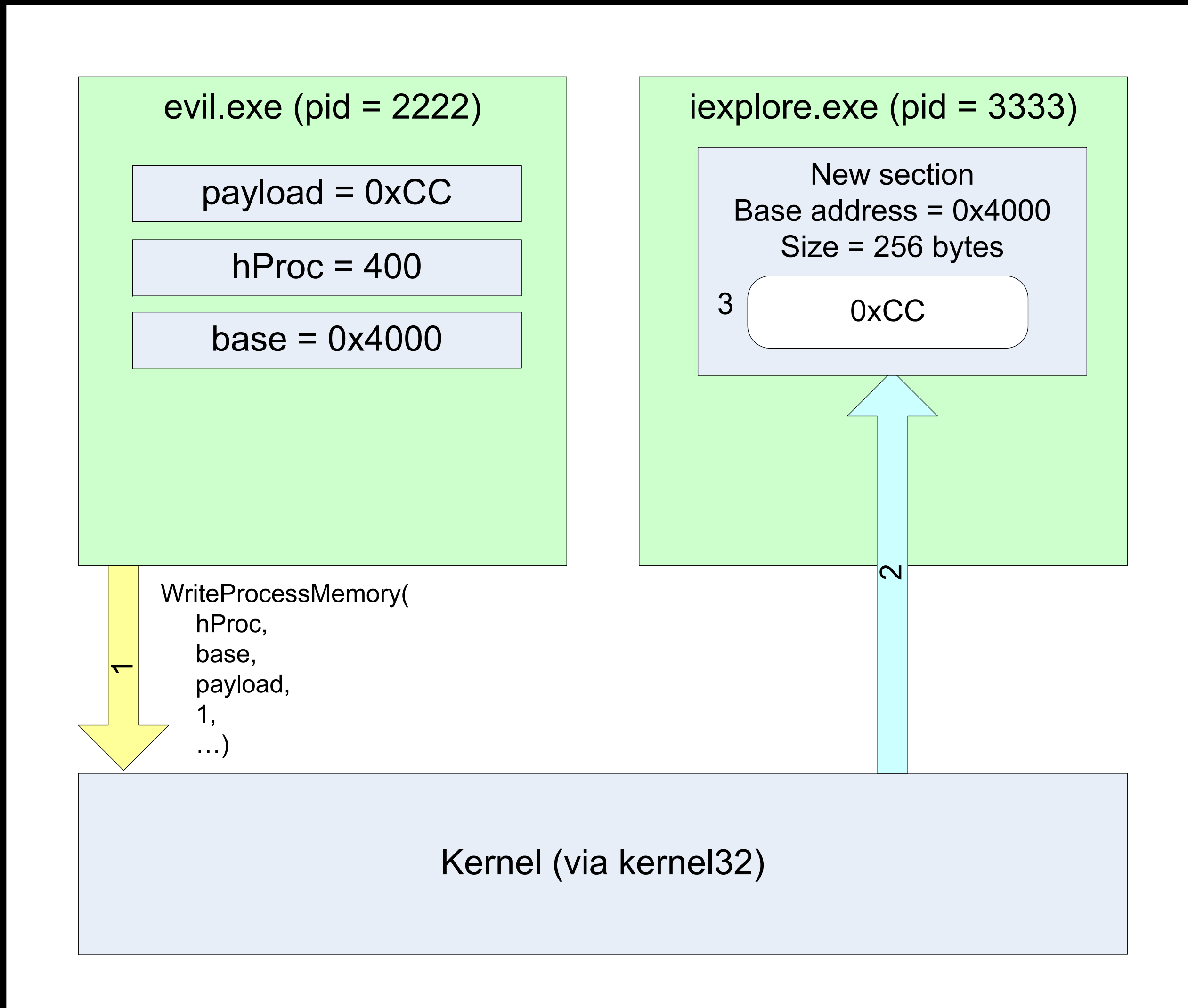
1. OpenProcess



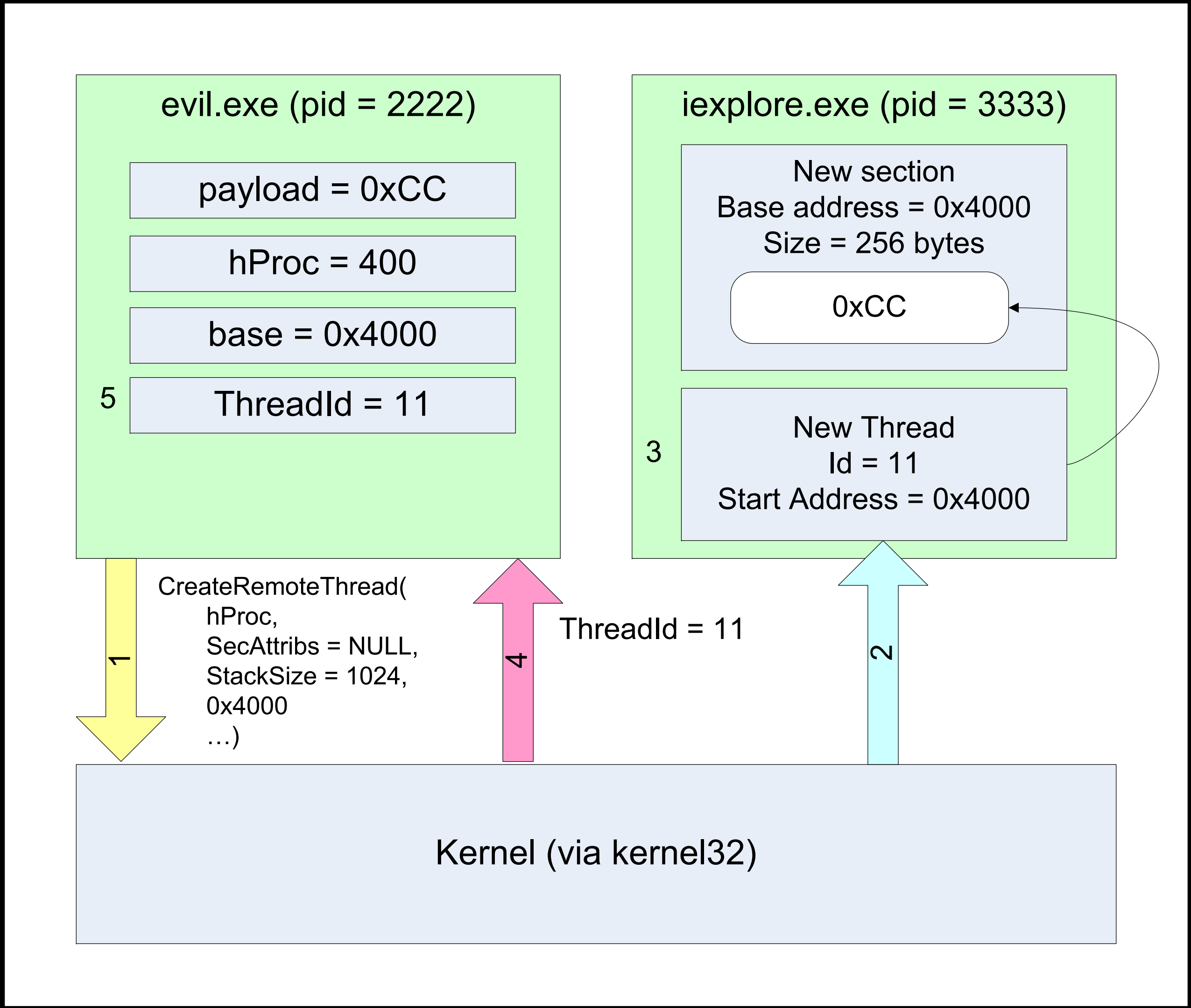
2. VirtualAllocEx



3. WriteProcessMemory



4. CreateRemoteThread



```
#Inject an infinite loop into a running process

import pydbg
k32 = pydbg.kernel32
payload = '\xEB\xFE'
pid =  int(args[0])
...

h = k32.OpenProcess(PROCESS_ALL_ACCESS,\
                    False, pid)
m = k32.VirtualAllocEx(h, None, 1024,\
                       MEM_COMMIT,\
                       PAGE_EXECUTE_READWRITE)
k32.WriteProcessMemory(h, m, payload,\
                       len(payload), None)
k32.CreateRemoteThread(h, None, 1024000,
                       m, None, 0, None)
```

Better Payloads

- Breakpoints and Loops are fun, but what about real payloads?
- If we directly inject code it must be “position independent”
- Any addresses that were pre-calculated at compile time would be wrong in the context of a new process

Better Payloads

- Building large position independent payloads is possible, but not trivial
- However, DLL injection is much simpler
- DLLs are designed to be loaded in a variety of processes, addresses are automatically fixed up when the DLL is loaded

DLL Injection

- Use the basic process we just described
- DLLs are loaded using kernel32!LoadLibrary
- kernel32 is at the same address in every process → we know its address in the remote process (ignoring ASLR)
- Allocate space for the name of the DLL to be loaded, then create a thread with a start address that points to LoadLibrary

```
#DLL Injection Excerpt

import pydbg
k32 = pydbg.kernel32
pid =  int(args[0])
dllname = args[1]
...
h = k32.OpenProcess(PROCESS_ALL_ACCESS,\
                    False, pid)
m = k32.VirtualAllocEx(h, None, 1024,\
                       MEM_COMMIT,\
                       PAGE_EXECUTE_READWRITE)
k32.WriteProcessMemory(h, m, dllname,\
                       len(dllname), None)
k32.CreateRemoteThread(h, None, 1024,
                       k32.LoadLibrary, m, 0,
None)
```

User Mode API Variants

- Rather than create a new remote thread, we can hijack an existing thread using `GetThreadContext`, `SetThreadContext`
- `SetWindowsHookEx` can also be used to inject a DLL into a single remote process, or every process running on the current Desktop

SetWindowsHookEx

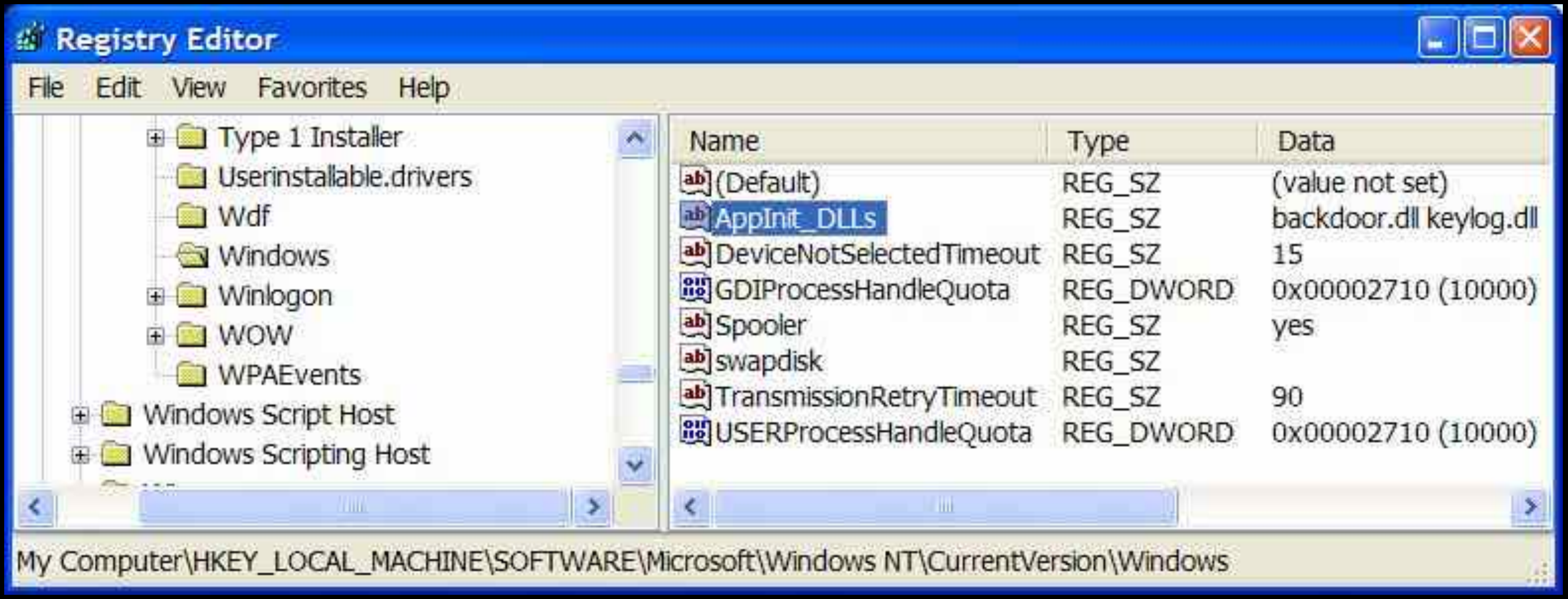
- SetWindowsHookEx defines a hook procedure within a DLL that will be called in response to specific events
- Example events: WH_KEYBOARD, WH_MOUSE, WH_CALLWNDPROC, WH_CBT
- Whenever the hooked event is first fired in a hooked thread, the specified DLL is be loaded

Permissions and Security

- To open a process opened by another user (including SYSTEM), you must hold the SE_DEBUG privilege
- Normally SE_DEBUG is only granted to member of the Administrator group
- However, even if you are running as a normal user, malware can still inject into another process that you own

Injecting code via AppInit_DLLs

- The AppInit_DLLs registry value provides another convenient method of DLL injection



Injecting code via Detours

- Detours is a library developed by Microsoft Research in 1999
- The library uses the same techniques already described, wrapped up in slick package

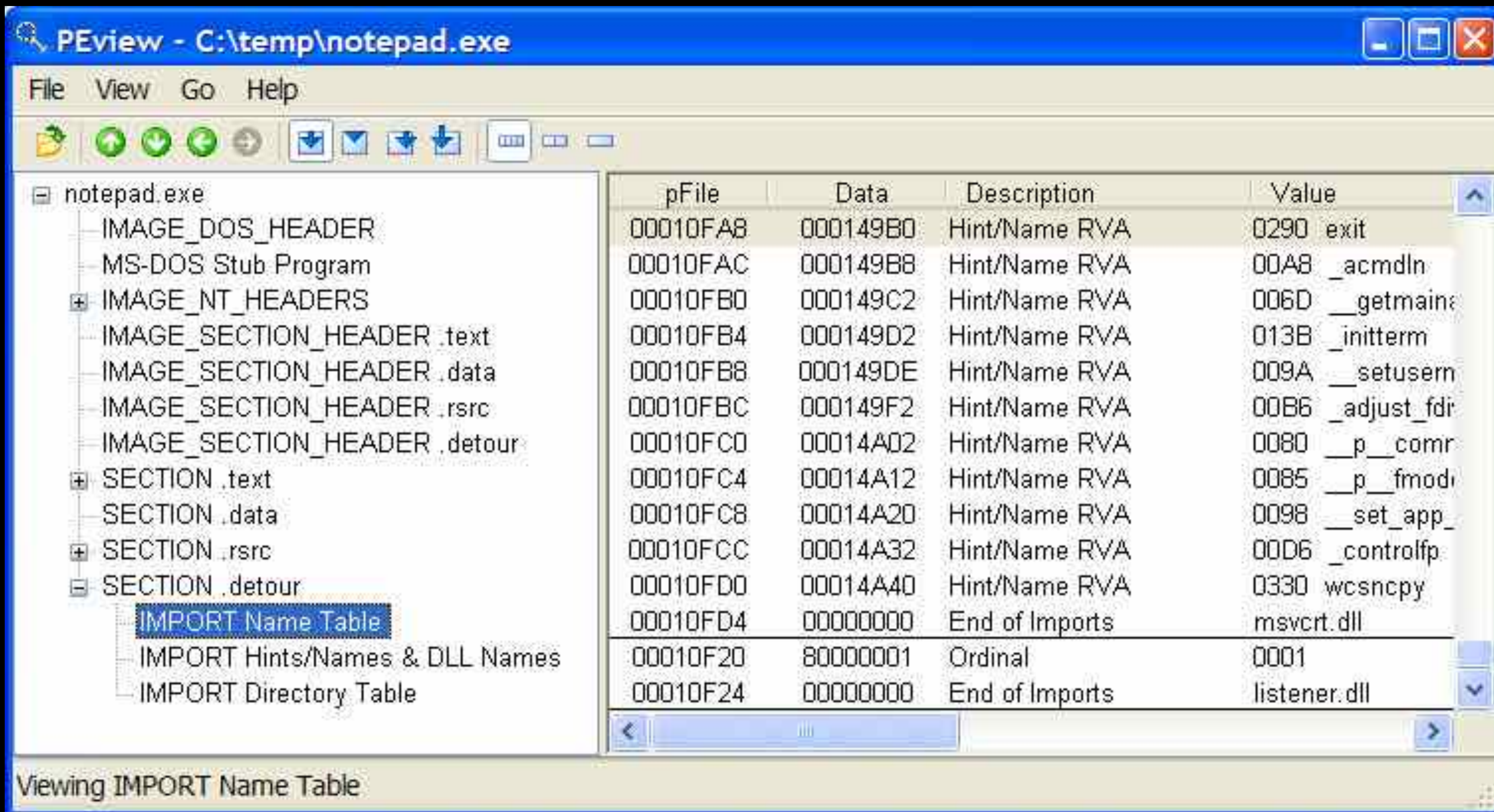
Detours Features

- Function hooking in running processes
- Import table modification
- Attaching a DLL to an existing program file
- Detours comes with great sample programs:
  - Withdll
  - Injdll
  - Setdll
  - Traceapi

Setdll

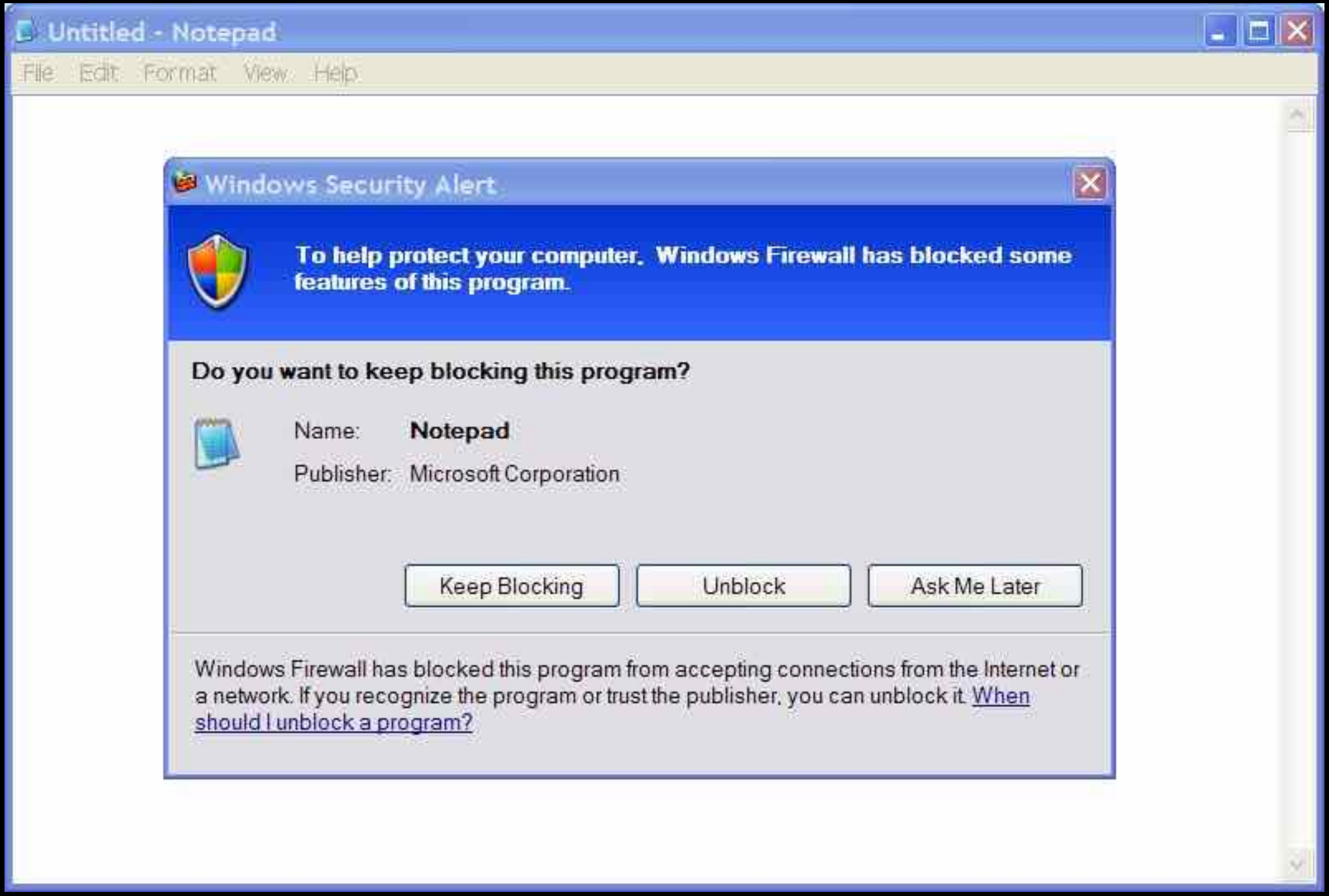
- Detours can add a new DLL to an existing binary on disk. How?
- Detours creates a section named ".detours" between the export table and debug symbols
- The .detours section contains the original PE header, and a new IAT
- Detours modifies the PE header to point at the new IAT (reversible)

Setdll Demo



PEview - C:\temp\notepad.exe
File View Go Help
notepad.exe
IMAGE_DOS_HEADER
MS-DOS Stub Program
IMAGE_NT_HEADERS
IMAGE_SECTION_HEADER .text
IMAGE_SECTION_HEADER .data
IMAGE_SECTION_HEADER .rsrc
IMAGE_SECTION_HEADER .detour
SECTION .text
SECTION .data
SECTION .rsrc
SECTION .detour
IMPORT Name Table
IMPORT Hints/Names & DLL Names
IMPORT Directory Table
pFile Data Description Value
00010FA8 000149B0 Hint/Name RVA 0290 exit
00010FAC 000149B8 Hint/Name RVA 00A8 _acmdln
00010FB0 000149C2 Hint/Name RVA 006D __getmain
00010FB4 000149D2 Hint/Name RVA 013B _initterm
00010FB8 000149DE Hint/Name RVA 009A __setusern
00010FBC 000149F2 Hint/Name RVA 00B6 _adjust_fdi
00010FC0 00014A02 Hint/Name RVA 0080 __p__comr
00010FC4 00014A12 Hint/Name RVA 0085 __p__fmod
00010FC8 00014A20 Hint/Name RVA 0098 __set_app_
00010FCC 00014A32 Hint/Name RVA 00D6 _controlfp
00010FD0 00014A40 Hint/Name RVA 0330 wcsncpy
00010FD4 00000000 End of Imports msvcrt.dll
00010F20 80000001 Ordinal 0001
00010F24 00000000 End of Imports listener.dll
Viewing IMPORT Name Table

Setdll Demo


Untitled - Notepad
File Edit Format View Help
Windows Security Alert
To help protect your computer, Windows Firewall has blocked some features of this program.
Do you want to keep blocking this program?
Name: Notepad
Publisher: Microsoft Corporation
Keep Blocking
Unblock
Ask Me Later
Windows Firewall has blocked this program from accepting connections from the Internet or a network. If you recognize the program or trust the publisher, you can unblock it. When should I unblock a program?

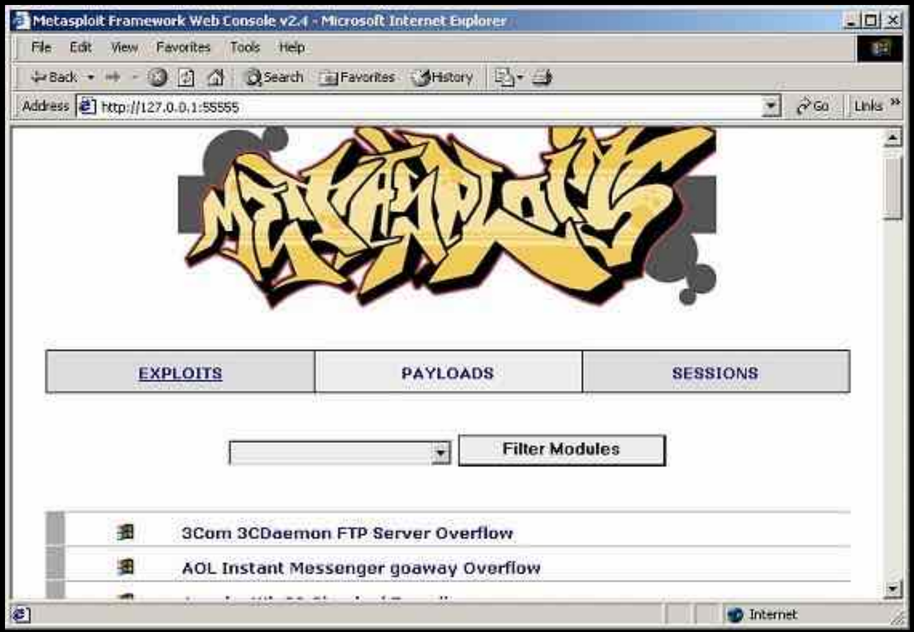
Avoiding the Disk

- When we perform DLL injection, `LoadLibrary` expects the DLL to be on the disk (or at least an SMB share)
- The Metasploit project eliminates this requirement using a clever hooking strategy
- By hooking functions that are involved in reading the file from disk, they fool Windows into thinking the DLL is on disk

Meterpreter

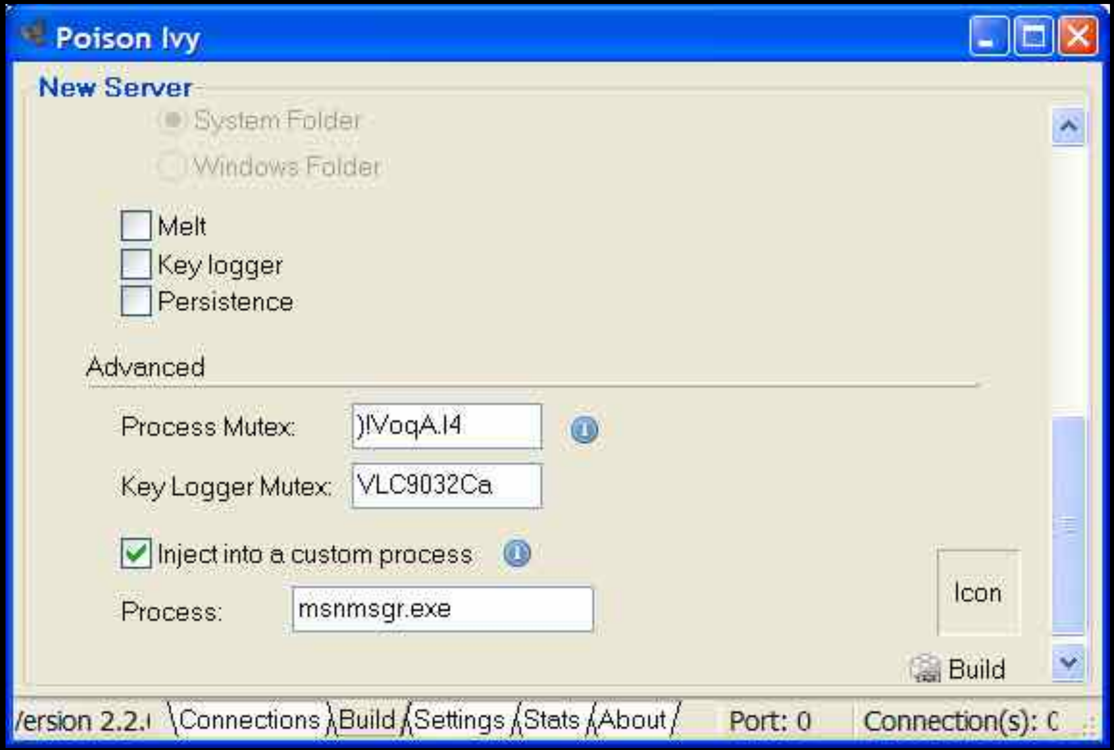
- Hook → Call LoadLibrary → Unhook
- Hooked functions:
  - NtMapViewOfSection
  - NtQueryAttributesFile
  - NtOpenFile
  - NtCreateSection
  - NtOpenSection
- See remote_dispatch.c and libloader.c in MSF 3.0

Meterpreter Demo

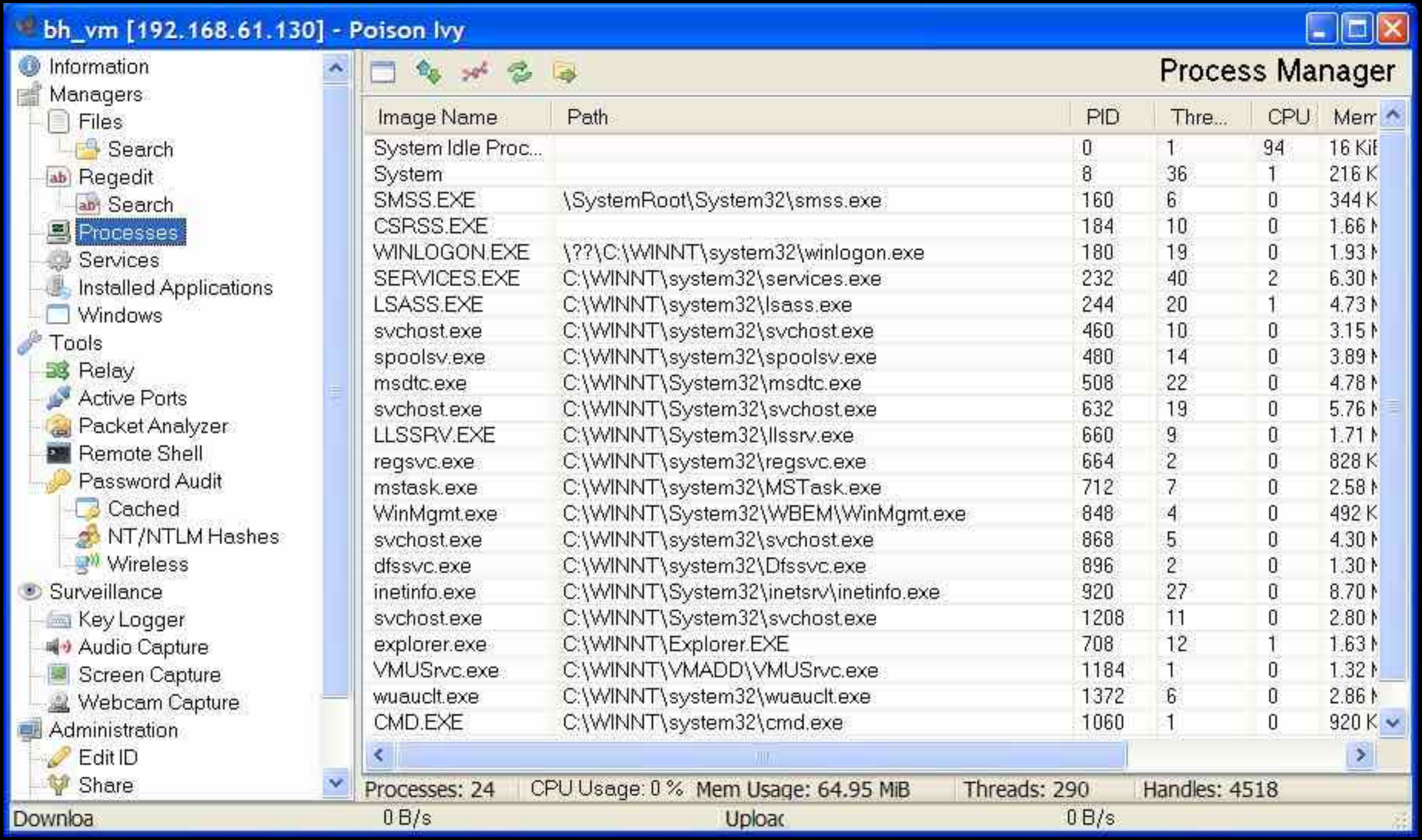


Poison Ivy RAT

- Tons of malware uses Code Injection
- We'll quickly dig into the details of one example



Poison Ivy Capabilities



Step 1: Inject to Explorer

- Poison Ivy client immediately injects to Explorer and then exits
- Output from WinApiOverride32 for pi.exe

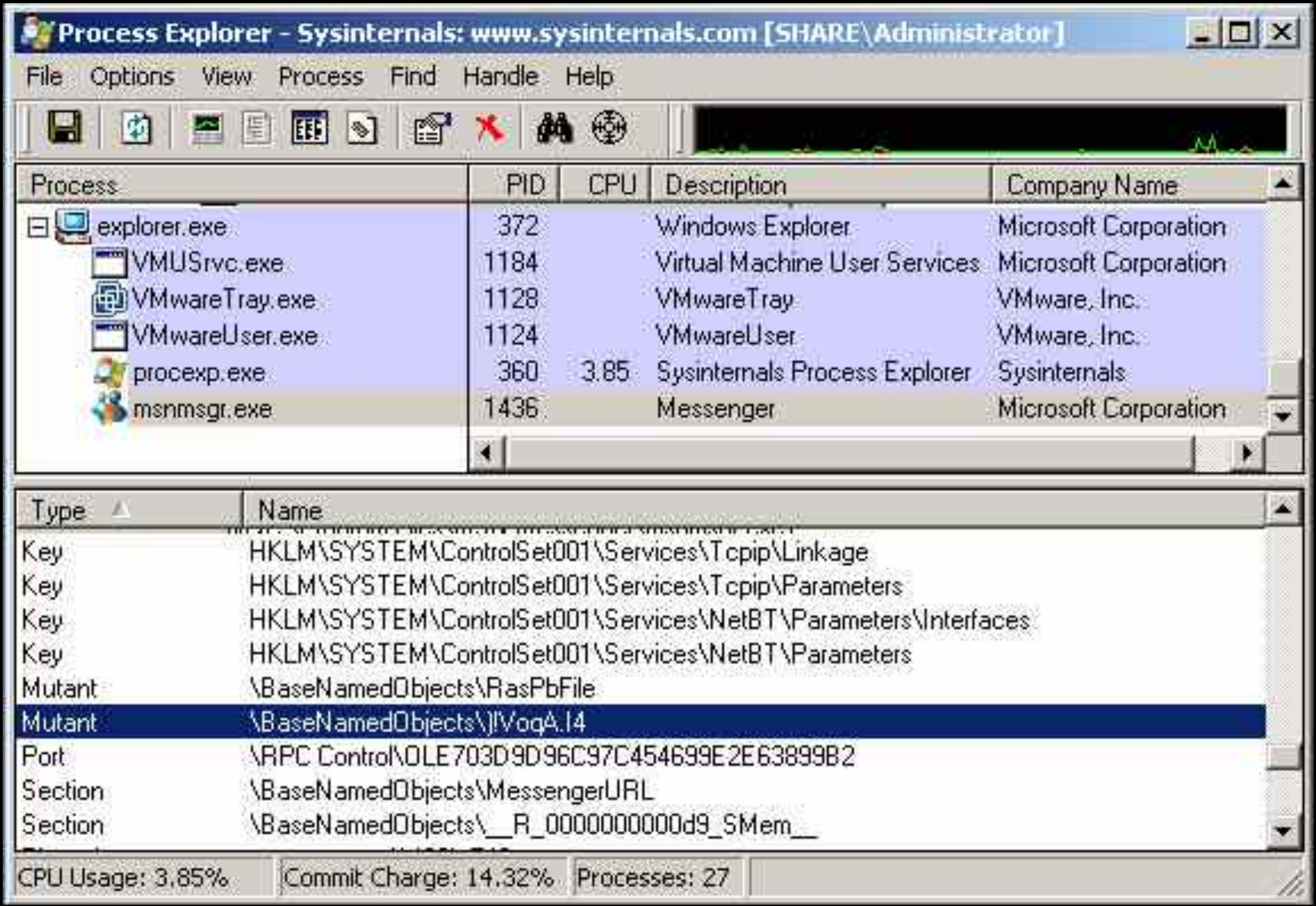
Id	Dir	Call
52	Out	Process32Next(hSnapshot:0x464C,lppe: 0x12FE88: {dwSize=296,cntUsage=0,th32ProcessID=0x38C,t...)
53	In	lstrcmpi(lpString1:0x12FEAC:"dfssvc.exe",lpString2:0x401363:"explorer.exe")
54	Out	Process32Next(hSnapshot:0x464C,lppe: 0x12FE88: {dwSize=296,cntUsage=0,th32ProcessID=0x4B8,t...)
55	In	lstrcmpi(lpString1:0x12FEAC:"svchost.exe",lpString2:0x401363:"explorer.exe")
56	Out	Process32Next(hSnapshot:0x464C,lppe: 0x12FE88: {dwSize=296,cntUsage=0,th32ProcessID=0x174,t...)
57	In	lstrcmpi(lpString1:0x12FEAC:"explorer.exe",lpString2:0x401363:"explorer.exe")
58	In	CloseHandle(hObject:0x464C)
59	In	OpenProcess(dwDesiredAccess:0x1F0FFF,bInheritHandle:0x0,dwProcessId:0x174)
60	Out	VirtualAllocEx(hProcess:0x464C,lpAddress:0x00000000: Bad Pointer,dwSize:0x1B93,flAllocationType:0x3...
61	In	WriteProcessMemory(hProcess:0x464C,lpBaseAddress:0x055F0000: Bad Pointer,lpBuffer: 0x40138E: {55 ...
62	Out	VirtualAllocEx(hProcess:0x464C,lpAddress:0x00000000: Bad Pointer,dwSize:0xB53,flAllocationType:0x30...
63	In	WriteProcessMemory(hProcess:0x464C,lpBaseAddress:0x05600000: Bad Pointer,lpBuffer: 0x403500: {00 ...
64	Out	CreateRemoteThread(hProcess:0x464C,lpThreadAttributes:0x00000000: Bad Pointer,dwStackSize:0x0,lp...
65	In	CloseHandle(hObject:0x464C)
66	In	TlsFree(dwTlsIndex:0x1)

Step 2: Inject again to msnmsgr.exe

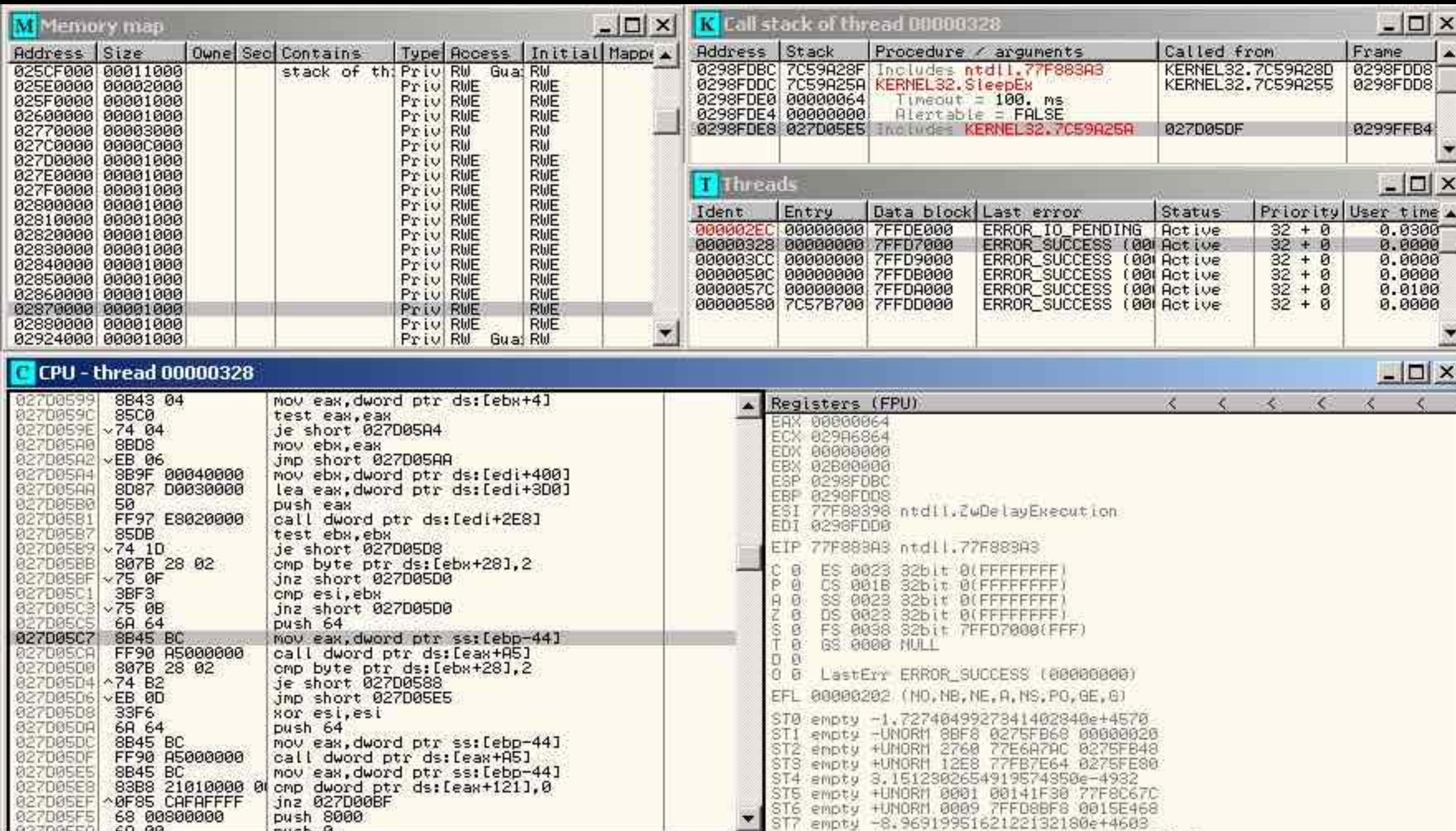
- Explorer.exe injected code then injects again…
- Interestingly, PI does not grab the SE_DEBUG privilege, so we can't inject in many existing processes
- Output from WinApiOverride32 for explorer.exe

94	In	lstrcmpi(lpString1:0x477F080:"msiexec.exe",lpString2:0x4670442:"msnmsgr.exe")	0xffffffff
95	Out	Process32Next(hSnapshot:0x414,lppe: 0x477F05C: {dwSize=296,cntUsage=0,th3…	0x00000001
96	In	lstrcmpi(lpString1:0x477F080:"msnmsgr.exe",lpString2:0x4670442:"msnmsgr.exe")	0x00000000
97	In	CloseHandle(hObject:0x414)	0x00000001
98	In	OpenProcess(dwDesiredAccess:0x1F0FFF,bInheritHandle:0x0,dwProcessId:0x5B8)	0x00004b6c
99	Out	VirtualAllocEx(hProcess:0x4B6C,lpAddress:0x00000000: Bad Pointer,dwSize:0xF9C,…	0x02870000
100	In	WriteProcessMemory(hProcess:0x4B6C,lpBaseAddress:0x02870000: Bad Pointer,lp…	0x00000001
101	Out	VirtualAllocEx(hProcess:0x4B6C,lpAddress:0x00000000: Bad Pointer,dwSize:0xB53,…	0x02880000
102	In	WriteProcessMemory(hProcess:0x4B6C,lpBaseAddress:0x02880000: Bad Pointer,lp…	0x00000001
103	Out	CreateRemoteThread(hProcess:0x4B6C,lpThreadAttributes:0x00000000: Bad Point…	0x00004b70
104	In	CloseHandle(hObject:0x4B6C)	0x00000001

Did it Work?


Process Explorer - Sysinternals: www.sysinternals.com [SHARE\Administrator]
File Options View Process Find Handle Help
Process PID CPU Description Company Name
explorer.exe 372 Windows Explorer Microsoft Corporation
VMUSrvc.exe 1184 Virtual Machine User Services Microsoft Corporation
VMwareTray.exe 1128 VMwareTray VMware, Inc.
VMwareUser.exe 1124 VMwareUser VMware, Inc.
procexp.exe 360 3.85 Sysinternals Process Explorer Sysinternals
msnmsgr.exe 1436 Messenger Microsoft Corporation
Type Name
Key HKLM\SYSTEM\ControlSet001\Services\Tcpip\Linkage
Key HKLM\SYSTEM\ControlSet001\Services\Tcpip\Parameters
Key HKLM\SYSTEM\ControlSet001\Services\NetBT\Parameters\Interfaces
Key HKLM\SYSTEM\ControlSet001\Services\NetBT\Parameters
Mutant \BaseNamedObjects\RasPbFile
Mutant \BaseNamedObjects\)!VoqA.I4
Port \RPC Control\OLE703D9D96C97C454699E2E63899B2
Section \BaseNamedObjects\MessengerURL
Section \BaseNamedObjects__R_0000000000d9_SMem__
CPU Usage: 3.85% Commit Charge: 14.32% Processes: 27

Where is the evil?



MANDIANT

Kernel Process Injection

Two Halves of the Process

- User land processes are comprised of two parts
  - Kernel Portion
    - EPROCESS and KPROCESS
    - ETHREAD and KTHREAD
    - Token
    - Handle Table
    - Page Tables
    - Etc.

Two Halves of the Process

- User land Portion
  - Process Environment Block (PEB)
  - Thread Environment Block (TEB)
  - Windows subsystem (CSRSS.EXE)
  - Etc.

Kernel Process Injection Steps

- Must find suitable target
  - Has a user land portion
  - Has kernel32.dll and/or ntdll.dll loaded in its address space
  - Has an alterable thread (unless hijacking an existing thread)
- Allocate memory in target process
- Write the equivalent of “shellcode” that calls LoadLibrary
- Cause a thread in the parent to execute newly allocated code
  - Hijack an existing thread
  - Create an APC

Allocate memory in parent process

- Change virtual memory context to that of the target
  - KeAttachProcess/KeStackAttachProcess
  - ZwAllocateVirtualMemory
    - (HANDLE) -1 means current process
    - MEM_COMMIT
    - PAGE_EXECUTE_READWRITE

Creating the Shellcode

- "shellcode" that calls LoadLibrary
  - Copy function parameters into address space
  - Pass the address of function parameters to calls
  - Can use the FS register
    - FS contains the address of the TEB
    - TEB has a pointer to the PEB
    - PEB has a pointer to the PEB_LDR_DATA
    - PEB_LDR_DATA contains all the loaded DLLs

Creating the Shellcode

- As an alternative to using the FS register
  - Find the address of ntdll.dll from the driver
  - Parse its exports section
  - Does not work with all DLLs
    - Only address of ntdll.dll returned by ZwQuerySystemInformation

Thread Hijacking

- Cause a thread in the parent to execute newly allocated code - Hijack an existing thread
  - Locate a thread within the parent process
  - Change its Context record
  - Change Context record back when done
- Problems:
  - Low priority threads
  - Blocked threads
  - Changing Context back

Thread Context Hijacking

- Hijack and Context records
- lkd> dt nt!_CONTEXT
- +0x000 ContextFlags : Uint4B
- +0x004 Dr0 : Uint4B
- +0x008 Dr1 : Uint4B
- +0x00c Dr2 : Uint4B
- +0x010 Dr3 : Uint4B
- +0x014 Dr6 : Uint4B
- +0x018 Dr7 : Uint4B
- +0x01c FloatSave : _FLOATING_SAVE_AREA
- +0x08c SegGs : Uint4B
- +0x090 SegFs : Uint4B
- +0x094 SegEs : Uint4B
- +0x098 SegDs : Uint4B
- +0x09c Edi : Uint4B
- +0x0a0 Esi : Uint4B
- +0x0a4 Ebx : Uint4B
- +0x0a8 Edx : Uint4B
- +0x0ac Ecx : Uint4B
- +0x0b0 Eax : Uint4B
- +0x0b4 Ebp : Uint4B
- +0x0b8 Eip : Uint4B
- +0x0bc SegCs : Uint4B
- +0x0c0 EFlags : Uint4B
- +0x0c4 Esp : Uint4B
- +0x0c8 SegSs : Uint4B
- +0x0cc ExtendedRegisters : [512] UChar

Alternative Method: APC

- Cause a thread in the parent to execute newly allocated code - Create an APC
  - Threads can be notified to run an Asynchronous Procedure Call (APC)
  - APC has a pointer to code to execute
  - To be notified, thread should be Alertable

Alertable Threads and APCs – MSDN

Parameter Settings of KeWaitForXxx Routines	Special Kernel-Mode APC		Normal Kernel-Mode APC		User-Mode APC		Alerts
	Wait Aborted?	APC Delivered and Executed?	Wait Aborted?	APC Delivered and Executed?	Wait Aborted?	APC Delivered and Executed?	Wait Aborted?
Alertable = TRUE WaitMode = User	No	If **(A)** then **Yes**	No	If **(B)** then **Yes**	**Yes**	**Yes**, after thread returns to user mode	**Yes**
Alertable = TRUE *WaitMode* = Kernel	No	If **(A)** then **Yes**	No	If **(B)** then **Yes**	**No** (since *WaitMode* = Kernel)	**No**	**Yes**
Alertable = FALSE *WaitMode* = User	No	If **(A)** then **Yes**	No	If **(B)** then **Yes**	**No** (since *Alertable* = FALSE)	**No** (with exceptions, EX. ^C to terminate)	**No**
Alertable = FALSE *WaitMode* = Kernel	No	If **(A)** then **Yes**	No	If **(B)** then **Yes**	**No** (since *Alertable* = FALSE **and** since *WaitMode* = Kernel)	**No**	**No**

A. **IRQL < APC_LEVEL**

B. **IRQL < APC_LEVEL, thread not already in an APC, thread not in a critical section**

Finding an Alertable Thread

```
PETHREAD FindAlertableThread(PEPROCESS eproc)
{
	PETHREAD start, walk;

	if (eproc == NULL)
		return NULL;
	start = *(PETHREAD *)((DWORD)eproc + THREADOFFSET);
	start = (PETHREAD)((DWORD)start - THREADFLINK);
	walk = start;

	do
	{
		DbgPrint("Looking at thread 0x%x\n",walk);

		if (*(PUCHAR)((DWORD)walk + ALERTOFFSET) == 0x01)
			return walk;
		walk = *(PETHREAD *)((DWORD)walk + THREADFLINK);
		walk = (PETHREAD)((DWORD)walk - THREADFLINK);
	}while (walk != start);

	return NULL;
}
```

Kernel Process Injection Demo

Memory Analysis

- Motivation
  - APIs lie. The operating system can be subverted.
    - Example: Unlink injected DLLs from the PEB_LDR_DATA in the PEB.
    - Example: Hooking the Virtual Memory Manager and diverting address translation.
  - APIs are not available to “classic” forensic investigations – offline analysis

Memory Analysis

- Requirements
  - No use of APIs to gather data.
  - Ability to use any analysis solution on both live memory and offline memory image dumps.
    (Implies the ability to do all memory translation independently.)
  - Do not require PDB symbols or any other operating specific information.

Steps to Memory Analysis

- Ability to access physical memory
- Derive the version of the OS – important to know how to interpret raw memory
- Find all Processes and/or Threads
- Enumerate File Handles, DLLs, Ports, etc.

Steps to Memory Analysis

- Virtual to Physical Address Translation
  - Determine if the host uses PAE or non-PAE
  - Find the Page Directory Table – process specific
  - Translate prototype PTEs
  - Use the paging file

Derive the version of the OS

- Find the System Process
  - Allows the derivation of:
    - The major operating system version in question
    - The System Page Directory Table Base
    - HandleTableListHead
    - Virtual address of PsInitialSystemProcess
    - PsActiveProcessHead
    - PsProcessType

Operating System Version

- Find the System image name
- Walk backwards to identify the Process Block
- The spatial difference between major versions of the OS is enough to begin to tell us about the operating system version

Operating System Version

- Drawback: Ghosts
  - There can be more than one System Process
    - Open a memory crash dump in Windbg
    - Run a Windows operating system in VMWare
  - Solution:
    - Non-paged kernel addresses are global
    - We know the virtual address of PsActiveProcessHead
    - PsActiveProcessHead and other kernel addresses should be valid and present (translatable) in both live or dead memory

Memory Translation

- PAE vs non-PAE
  - Different ways to interpret the address tables
  - The sixth bit in the CR4 CPU register determines if PAE is enabled
  - Problem: We do not have access to CPU registers in memory analysis
  - Solution?
    - Kernel Processor Control Region -> KPCRB -> KPROCESSOR_STATE -> KSPECIAL_REGISTERS -> CR4

Memory Translation

- CR4 Heuristic
  - Page Directory Table Base and the Page Directory Table Pointer Base look very different.
- CR3 is updated in the KPCR
  - This can be used to identify a valid Page Directory Table
  - The Page Directory can be used to validate the PsActiveProcessHead

Enumerating Injected DLLs

- Problem:
  - APIs lie.
  - Malware can unlink from the PEB_LDR_DATA lists of DLLs
- Solution:
  - Virtual Address Descriptors (VADs)

VADs

- Self balancing binary tree [1]
- Contains:
  - Virtual address range
  - Parent
  - Left Child and Right Child
  - Flags – is the memory executable
  - Control Area

1. Russinovich, Mark and Solomon, Dave, *Microsoft Windows Internals*, Microsoft Press 2005

A Memory Map to a Name

- VAD contains a CONTROL_AREA
- CONTROL_AREA contains a FILE_OBJECT
- A FILE_OBJECT contains a UNICODE_STRING with the filename
- We now have the DLL name

Demo

Conclusion

Questions?

- Email: jamie.butler AT mandiant.com