Beyond EIP

spoonm & skape

BlackHat Federal, 2006
Part I

Introduction
Who are we?

- spoonm
  - Dropout bum
  - Metasploit developer since late 2003

- skape
  - Lead software developer by day
  - Independent security researcher by night
  - Joined the Metasploit project in 2004
  - Responsible for all cool features
What’s this presentation about?

- What it’s not about
  - New exploit / attack vectors
  - New exploitation techniques
  - 0day, bugs, etc
What’s this presentation about?

▶ What it’s not about
  ▶ New exploit / attack vectors
  ▶ New exploitation techniques
  ▶ 0day, bugs, etc

▶ What it is about
  ▶ What you can do after owning EIP
  ▶ The techniques to do it
  ▶ Our tools to support it
Plan of attack

- Introduction
  - Payload background
  - Technologies used as a basis
- Post-exploitation tools
  - Background & review of existing tools
  - The technology behind our tools
  - How they can be used
  - Crazy cool features for the end-user
Our definitions: the exploitation cycle

- **Pre-exploitation** - Before the attack
  - Find a bug, isolate, write exploit
  - Write any other tools, payloads, etc
Our definitions: the exploitation cycle

- **Pre-exploitation** - Before the attack
  - Find a bug, isolate, write exploit
  - Write any other tools, payloads, etc

- **Exploitation** - Leveraging the vulnerability
  - Recon, information gathering, find target
  - Initialize tools and infrastructure
  - Launch the exploit
Our definitions: the exploitation cycle

- **Pre-exploitation** - Before the attack
  - Find a bug, isolate, write exploit
  - Write any other tools, payloads, etc

- **Exploitation** - Leveraging the vulnerability
  - Recon, information gathering, find target
  - Initialize tools and infrastructure
  - Launch the exploit

- **Post-exploitation** - Manipulating the target
  - Arbitrary command execution
  - Command execute via shell
  - File access, VNC, pivoting, etc
  - Advanced payload interaction
What’s a payload?

**Definition**

- Arbitrary code that is to be executed upon successful exploitation
What’s a payload?

Definition

- Arbitrary code that is to be executed upon successful exploitation

How a payload works

- Client prepares the payload for execution
What’s a payload?

Definition
► Arbitrary code that is to be executed upon successful exploitation

How a payload works
► Client prepares the payload for execution
► Data may be embedded (cmd to execute, hostname, port, etc)
What’s a payload?

**Definition**
- Arbitrary code that is to be executed upon successful exploitation

**How a payload works**
- Client prepares the payload for execution
- Data may be embedded (cmd to execute, hostname, port, etc)
- Client transmits the payload via an exploit
What’s a payload?

Definition

▶ Arbitrary code that is to be executed upon successful exploitation

How a payload works

▶ Client prepares the payload for execution
▶ Data may be embedded (cmd to execute, hostname, port, etc)
▶ Client transmits the payload via an exploit
▶ Target executes the payload
Payload stagers

- Stagers are typically network based and follow three basic steps
  - Establish connection to attacker (reverse, portbind, findsock)
  - Read in a payload from the connection
  - Setup connection information and branch to stage
Payload stagers

- Stagers are typically network based and follow three basic steps
  - Establish connection to attacker (reverse, portbind, findsock)
  - Read in a payload from the connection
  - Setup connection information and branch to stage

- The three steps make it so stages are independent of the connection method
  - 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
- 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
- Provides an abstraction level for loading code onto a remote machine through any medium
Existing payload stager technology

- Standard reverse, portbind, and findsock stagers included in Metasploit 2.2+
- LSD Win32 Assembly Components
- Found in public exploits (Solar Eclipse OpenSSL)
Payload stages

- Payload stages are executed by payload stagers and perform arbitrary tasks
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 (ex adduser)
  - Download an executable from a URL and execute it
Why are payload stages useful?

► Highly reusable (connection independent, etc)
► Can conform to some sort of ABI
Why are payload stages useful?

- Highly reusable (connection independent, etc)
- Can conform to some sort of ABI
- Not subject to size limitations of individual vulnerabilities
- This means they can be arbitrarily complex
Part II

Post Exploitation
What is post-exploitation?

- The purpose of an exploit is to manipulate a target

Manipulation of a target begins in post-exploitation
- Command shells are executed
- Files are downloaded

Represents the culmination of the exploitation cycle
What is post-exploitation?

- The purpose of an exploit is to manipulate a target
- Manipulation of a target begins in post-exploitation
  - Command shells are executed
  - Files are downloaded
What is post-exploitation?

- The purpose of an exploit is to manipulate a target
- Manipulation of a target begins in post-exploitation
  - Command shells are executed
  - Files are downloaded
- Represents the culmination of the exploitation cycle
What do most people do in post-exploitation?

- Most people spawn a command shell
  - Poor automation support
  - Reliant on the shell’s intrinsic commands
  - Limited to installed applications
  - Can’t provide advanced features
What do most people do in post-exploitation?

- Most people spawn a command shell
  - Poor automation support
  - Reliant on the shell’s intrinsic commands
  - Limited to installed applications
  - Can’t provide advanced features

- Some people use syscall proxies
  - Good automation support
  - Partial or full access to target native API
  - Can be clumsy when implementing complex features
  - Typically require specialized build steps
The idea is to have interactive shellcode
And be able to keep a very low footprint
- The idea is to have interactive shellcode
- And be able to keep a very low footprint
- But also have lots of optional power
The idea is to have interactive shellcode
And be able to keep a very low footprint
But also have lots of optional power

Basically a shellcode read-eval-print loop
First stage loops, reading/executing code
DispatchNinja "modules" are sent and executed
This is what we call "dispatching"
The idea is to have interactive shellcode
And be able to keep a very low footprint
But also have lots of optional power

Basically a shellcode read-eval-print loop
First stage loops, reading/executing code
DispatchNinja "modules" are sent and executed
This is what we call "dispatching"

Modules are responsible for their own mini-protocols
Each module has a corresponding handler on client side
The idea is to have interactive shellcode
And be able to keep a very low footprint
But also have lots of optional power

Basically a shellcode read-eval-print loop
First stage loops, reading/executing code
DispatchNinja "modules" are sent and executed
This is what we call "dispatching"

Modules are responsible for their own mini-protocols
Each module has a corresponding handler on client side

Modules have a simple C ABI, and have a main function
Most of our dN modules were written in C (shellforge)
DispatchNinja - Client side APIs

- Client side APIs wrap handler and module code
- Msf3 has ruby dN client side APIs
DispatchNinja - Client side APIs

- Client side APIs wrap handler and module code
- Msf3 has ruby dN client side APIs

- APIs modeled after the ruby APIs (Dir, File, etc)
- Our APIs should support the majority of Ruby functionality
irb#1(main):001:0> c = @c
=> #<Rex::Post::DispatchNinja::Client:0xb7bf542c
  @sock=#<TCPSocket:0xb7bf5440>>
irb#1(main):002:0> c.dir.entries(’/tmp’)
=> [“.”, “..”, “.X11-unix”, “.ICE-unix”, “.font-unix”]

irb#1(main):004:0> puts c.file.stat(’/etc/passwd’).pretty
  Size: 1036  Blocks: 8  IO Block: 4096  Type: 0
  Device: 774  Inode: 81499  Links: 1
  Mode: 100644/rw-r--r--
  Uid: 0  Gid: 0
  Access: Tue Jul 26 20:08:09 EDT 2005
  Modify: Wed Jul 06 20:45:04 EDT 2005
  Change: Wed Jul 06 20:45:04 EDT 2005
=> nil

irb#1(main):005:0> Process.pid
=> 1496
irb#1(main):006:0> c.process.pid
=> 1498
What is Meterpreter?

- Short for *Meta-Interpreter*
- An advanced post-exploitation system
- Based on library injection technology
- First released with Metasploit 2.3
- Detailed write-up can be found in reference materials
What is Meterpreter?

- Short for *Meta-Interpreter*
- An advanced post-exploitation system
- Based on library injection technology
- First released with Metasploit 2.3
- Detailed write-up can be found in reference materials

- After exploitation, a Meterpreter server DLL is loaded on the target
What is Meterpreter?

- Short for *Meta-Interpreter*
- An advanced post-exploitation system
- Based on library injection technology
- First released with Metasploit 2.3
- Detailed write-up can be found in reference materials

- After exploitation, a Meterpreter server DLL is loaded on the target
- Attackers use a Meterpreter client to interact with the server to...
  - Load run-time extensions in the form of DLLs
  - Interact with communication channels
What is Meterpreter?

- Short for *Meta-Interpreter*
- An advanced post-exploitation system
- Based on library injection technology
- First released with Metasploit 2.3
- Detailed write-up can be found in reference materials

- After exploitation, a Meterpreter server DLL is loaded on the target
- Attackers use a Meterpreter client to interact with the server to...
  - Load run-time extensions in the form of DLLs
  - Interact with communication channels

- But before understanding Meterpreter, one should understand library injection...
Library injection

- Provides a method of loading a library (DLL) into an exploited process
Library injection

- Provides a method of loading a library (DLL) into an 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
Library injection

- Provides a method of loading a library (DLL) into an 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
Library injection

- Provides a method of loading a library (DLL) into an 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

- Two 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
- Both 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 is 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
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
- No disk access means no forensic trace if the machine loses power
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 file 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 file 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
In-Memory library injection on Windows

But why not just write a stub loader instead of using NTDLL.DLL?
In-Memory library injection on Windows

- But why not just write a stub loader instead of using NTDLL.DLL?

- Lots of reasons...
  - Requires manual import processing
  - Requires manual relocation fix-ups
  - Requires loading dependent DLLs
  - May require manual insertion into the loaded module lists
  - Other uncommon PE features that wouldn’t be supported
In-Memory library injection on Windows

- But why not just write a stub loader instead of using NTDLL.DLL?

- Lots of reasons...
  - Requires manual import processing
  - Requires manual relocation fix-ups
  - Requires loading dependent DLLs
  - May require manual insertion into the loaded module lists
  - Other uncommon PE features that wouldn’t be supported

- No compelling reason to re-implement what is already supplied in NTDLL.DLL
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

By using the generic library loading stager, VNC was simply plugged in. Extremely useful when illustrating security weaknesses. Suits understand mouse movement much better than command lines.
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 stager, 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 stager, VNC was simply plugged in
- Extremely useful when illustrating security weaknesses
- Suits understand mouse movement much better than command lines
Primary design goals are to be...
Meterpreter: Design goals

- Primary design goals are to be...
  - **Stealthy**: no disk access and no new process by default
Meterpreter: Design goals

- Primary design goals are to be...
  - **Stealthy**: no disk access and no new process by default
  - **Powerful**: channelized communication and robust protocol

The current server implementation is only for Windows.
Meterpreter: Design goals

- Primary design 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

Portability also a design consideration

The current server implementation is only for Windows
Primary design 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

Portability also a design consideration

- The current server implementation is only for Windows
Architecture - design goals

▶ Very flexible protocol; should adapt to extension requirements without modification

▶ Should expose a channelized communication system for extensions (like openssh)

▶ Should be as stealthy as possible

▶ Should be portable to various platforms

▶ Clients on one platform should work with servers on another

▶ All non-critical features should be implemented by extensions
Architecture - design goals

- Very flexible protocol; should adapt to extension requirements without modification
- Should expose a channelized communication system for extensions (like openssh)
Architecture - design goals

- Very flexible protocol; should adapt to extension requirements without modification
- Should expose a channelized communication system for extensions (like openssh)
- Should be as stealthy as possible
Architecture - design goals

- Very flexible protocol; should adapt to extension requirements without modification
- Should expose a channelized communication system for extensions (like openssh)
- Should be as stealthy as possible
- Should be portable to various platforms
Architecture - design goals

- Very flexible protocol; should adapt to extension requirements without modification
- Should expose a channelized communication system for extensions (like openssh)
- Should be as stealthy as possible
- Should be portable to various platforms
- Clients on one platform should work with servers on another
Architecture - design goals

- Very flexible protocol; should adapt to extension requirements without modification
- Should expose a channelized communication system for extensions (like openssh)
- Should be as stealthy as possible
- Should be portable to various platforms
- Clients on one platform should work with servers on another
- All non-critical features should be implemented by extensions
Architecture - protocol

- Uses TLV (Type-Length-Value) to support opaque data
Architecture - protocol

- Uses TLV (Type-Length-Value) to support opaque data.
- Every packet is composed of zero or more TLVs.
Architecture - protocol

- Uses TLV (Type-Length-Value) to support opaque data
- Every packet is composed of zero or more TLVs
- Packets themselves are TLVs
  - Type is the packet type (request, response)
  - Length is the length of the packet
  - Value is zero or more embedded TLVs

TLVs make packet parsing simplistic and flexible. No formatting knowledge is required to parse the packet outside of the TLV structure. This allows a core TLV parsing engine without any knowledge of the extensions or their protocols.
Architecture - protocol

- Uses TLV \((\text{Type-Length-Value})\) to support opaque data
- Every packet is composed of zero or more TLVs
- Packets themselves are TLVs
  - Type is the packet type (request, response)
  - Length is the length of the packet
  - Value is zero or more embedded TLVs
- TLVs make packet parsing simplistic and flexible
  - No formatting knowledge is required to parse the packet outside of the TLV structure
  - This allows a core TLV parsing engine without any knowledge of the extensions or their protocols.
Core client/server interface

- Server written in C, client written in any language
Core client/server interface

- Server written in C, client written in any language
- Provides a minimal interface to support the loading of extensions
Core client/server interface

- Server written in C, client written in any language
- Provides a minimal interface to support the loading of extensions
- Implements basic packet transmission and dispatching
- Exposes channel allocation and management to extensions

Also includes support for migrating the server to another running process.

Metasploit 2.x has a perl Meterpreter client

Metasploit 3.x has a ruby Meterpreter client
Core client/server interface

- Server written in C, client written in any language
- Provides a minimal interface to support the loading of extensions
- Implements basic packet transmission and dispatching
- Exposes channel allocation and management to extensions
- Also includes support for migrating the server to another running process

Metasploit 2.x has a perl Meterpreter client
Metasploit 3.x has a ruby Meterpreter client
Core client/server interface

- Server written in C, client written in any language
- Provides a minimal interface to support the loading of extensions
- Implements basic packet transmission and dispatching
- Exposes channel allocation and management to extensions
- Also includes support for migrating the server to another running process

- Metasploit 2.x has a perl Meterpreter client
- Metasploit 3.x has a ruby Meterpreter client
Augmenting features at run-time

- Adding new features is as simple as loading a DLL on the server
  - Client uploads the extension DLL
  - Server loads the DLL from memory and initializes it
Augmenting features at run-time

- Adding new features is as simple as loading a DLL on the server
  - Client uploads the extension DLL
  - Server loads the DLL from memory and initializes it
- Client can begin sending commands for the new extension
Meterpreter extensions in action: Stdapi

- Included in Metasploit 3.0
- Combination of previous extensions into standard interface
Meterpreter extensions in action: Stdapi

- Included in Metasploit 3.0
- Combination of previous extensions into standard interface
- Provides access to standard OS features
Meterpreter extensions in action: Stdapi

- Included in Metasploit 3.0
- Combination of previous extensions into standard interface
- Provides access to standard OS features
- Feature set provides for robust client-side automation
Meterpreter extensions in action: Stdapi

- Included in Metasploit 3.0
- Combination of previous extensions into standard interface
- Provides access to standard OS features
- Feature set provides for robust client-side automation
- Designed to mirror the Ruby API to make it easy to use existing scripts against targets
Why is Meterpreter useful?

- 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
Why is Meterpreter useful?

- 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?

- 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?

- 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
Why is Meterpreter useful?

- 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
Some of the features Meterpreter can offer

- Command execution & manipulation
- Registry interaction
- File system interaction
- Network pivoting & port forwarding
- Complete native API proxying
- Anything you can do as a native DLL, Meterpreter can do!
- Sky’s the limit!
Part III

Demos
Part IV

Conclusion
What does the future hold?

- Exploitation vectors and techniques are mature
What does the future hold?

- Exploitation vectors and techniques are mature
- Public post-exploitation suites still very weak
What does the future hold?

- Exploitation vectors and techniques are mature
- Public post-exploitation suites still very weak
- However, post-exploitation is maturing
What does the future hold?

- Exploitation vectors and techniques are mature
- Public post-exploitation suites still very weak
- However, post-exploitation is maturing
- Metasploit 3.0 should be cool
Reference Material

Payload Stages

- Library Injection

- Meterpreter
  http://www.nologin.org/Downloads/Papers/meterpreter.pdf