Metasploit Framework Version 2.1
User Crash Course

Table of Contents

Introduction

This document is the user guide for version 2.0 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.

What is it?

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.

Installation

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.

To perform a system-wide installation, we recommend that you copy the entire Framework directory into a globally accessible location (/usr/local/share/msf) and then create symbolic links from the msf* applications to a directory in the system path (/usr/local/bin).

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 bundles a stripped-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 the file 'docs/QUICKSTART.cygwin' in the installation directory; there are a number of issues with installing the Term::ReadLine::Gnu and Net::SSLeay modules that require jumping through hoops to solve.

Platform Caveats

While we have tried to support as many platforms as possible, there are some compatibility bugs that have cropped up. One of the most annoying ones is a bug in the version of Internet Explorer shipped with Mac OS X. For some reason the browsers refuses to display web pages until the connection has closed or all data has finished loading. This prevents the *msfweb* user interface from working correctly with Internet Explorer. This issue can be worked around by using a different browser, such as a Safari or Mozilla.

Getting Started

The Console Interface

After you have installed the Framework, you should verify that everything is working correctly. The quickest way to do this is to execute the *msfconsole* user interface. This interface should display an ASCII Metasploit logo, print the current version, number of payloads, number of exploits, 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 exploit module_name**.

Console Efficiency

The console has been designed with efficiency in mind and be used as a standard shell in many situations. If you enter an unknown command, the console will scan your PATH environment to determine if you typed 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. Tab completion defaults to file-name matching when the command entered is not an internal console command. This allows you to navigate the file system normally, similar to using a bash shell.

Selecting an Exploit

From the msf prompt, you can choose an exploit with the **use** command. This command takes the name of the exploit module as the first argument, enters exploit mode, and loads the Temporary environment for that exploit. You can switch between active exploits with the **use** command and drop back to the main shell with the **back** command.

Exploit Basics

After selecting an exploit, your available command selection changes. Enter the **help** command again to get an idea of what is available. The **show** command now has a completely different set of arguments, these allow you to view the standard options, advanced options, exploit targets, and compatible payloads. The **check** command invokes the vulnerability check mode of the selected exploit. The **exploit** command actually launches the selected exploit.

Environments

The environment system is a core component of the Framework; the interfaces use it to configure various options, the payloads use it generate the resulting opcodes, the exploits use it to define parameters, and it is used internally to pass options between disparate 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.

Global Environment

The Global environment is accessed through the console via the "g" family of functions (**setg**, **unsetg**). The following example is 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.

```
+ -- --=[ msfconsole v2.0 [22 exploits - 30 payloads]

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

Temporary Environment

The Temporary environment in only available in exploit mode (via **use exploitname**). Every exploit module has its own separate Temporary environment and this environment will be saved when moving between different exploit modules. The **save** command can be used to synchronize the Global and all Temporary environments to disk, where they will be loaded as defaults next time the interface starts.

The following example shows how the **use** command selects an active exploit and how the **back** command reverts to the main mode.

```
msf > use apache_chunked_win32
msf apache_chunked_win32 > set
msf apache_chunked_win32 > set FOO BAR
FOO -> BAR
msf apache_chunked_win32 > set
FOO: BAR
msf apache_chunked_win32 > back
msf > use poptop_negative_read
msf poptop_negative_read > set
msf poptop_negative_read > back
msf > use apache_chunked_win32
msf apache_chunked_win32 > set
FOO: BAR
msf apache_chunked_win32 >
```

Using the Environments Effectively

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 12344
LPORT -> 12344
msf > setg LHOST 192.168.0.10
LHOST -> 192.168.0.10
msf > setg PAYLOAD winreverse
PAYLOAD -> winreverse
msf > use apache_chunked_win32
msf apache_chunked_win32 > show options

Exploit and Payload Options
===========================

 Exploit:   Name     Default   Description
 --------   ------   -------   --------------------------
 optional   SSL               Use SSL
 required   RHOST             The target address
 optional   PAD               Specify the exact pad value
 required   RPORT     80       The target port

 Payload:   Name      Default       Description
 --------   -------   -----------   ----------------------------------------
 optional   EXITFUNC  seh            Exit technique: "process", "thread", "seh"
 required   LHOST     192.168.0.10   Local address to receive connection
 required   LPORT     12344          Local port to receive connection
```

Advanced Environment Settings

The environment allow you to control aspects of the user interface, logging options, and the behavior of certain routines. Many components of the Framework can be controlled through environment settings, some examples of these are below. Please note the environment name can change, depending on whether the option is being set in the Global or Temporary environment. Many of the module-specific options require the full module name to specified when set in the Global environment.

Global Name: DebugLevel

Temporary Name: DebugLevel

Description: This option tells the Framework and Modules how verbose their output should be, increasing the DebugLevel gives you more detailed information about whats going on under the hood. Supported values of DebugLevel range from 0 to 5.

Global Name: Encoder

Temporary Name: Encoder

Description: This option allows you to specify the exact order in which the Encoder modules should be used. The default configuration is to cycle through these until one is capable of encoding the payload correctly, this option can be used to change the precedence. The value should be a comma-separated list of encoder names.

Global Name: Nop

Temporary Name: Nop

Description: This has the same behavior as the Encoder entry above, except it is used to specify the preferred nop generator module.

Global Name: **Logging**

Temporary Name: **Logging**

Description: If set to a non-zero value, this will enable session logging. Session logs are stored in ~/.msflogs 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.

Global Name: **LogDir**

Temporary Name: **LogDir**

Description: This option specifies what directory the log files should be stored in. It defaults to ~/.msflogs.

Global Name: **Msf::Socket::ConnectTimeout**

Temporary Name: **ConnectTimeout**

Description: 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.

Global Name: **Msf::Socket::RecvTimeout**

Temporary Name: **RecvTimeout**

Description: 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.

Global Name: **Msf::Socket::RecvTimeoutLoop**

Temporary Name: **RecvTimeoutLoop**

Description: 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.

Global Name: **Msf::Socket::Proxies**

Temporary Name: **Proxies**

Description: 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. The 2.0 release includes support for socks4 and http proxy types.

Global Name: **Msf::Nop::Pex::RandomNops**

Temporary Name: **RandomNops**

Description: This option is specific to the Pex nop generator, it allows randomized nop sleds to be used instead of the standard nop character. This may cause issues with exploits that depend on registers to be set with specific values. It should only be used after being tested with the relative exploit.

Using the Framework

Configuring an Exploit

Once you have selected an exploit, the first 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. Once this is complete, you can try the **check** command to determine if the remote host is vulnerable. Not all exploits have a vulnerability check routine, others return a response and expect you to analyze it before proceeding.

Selecting the Payload

Now that the exploit options have been set, you need to choose a payload. The payload is the actual code which 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 **info payload payload_name** to get more information about a given payload.

Once you have decided on a payload, use the **set** command to specify the PAYLOAD environment variable. Once the payload has been set, use the **show options** command to display all available payload options. Most payloads require at least one option to be set for them to function.

Launching the Exploit

After the exploit and payload options have been set, you may need to configure the exploit target. If a default target is not set, the exploit command will return an error. You can use the **show targets** command to list all available targets and then the **set** command to specify a value for the TARGET environment variable. Once an appropriate target has been selected, you can use the **exploit** command to attack the remote host. If everything went well, your payload will execute and potentially drop you to a command shell in the console.

The Command Line Interface

If the console is overkill for your needs, you may want to try the msfcli interface. 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 compared against the available exploits, in the case that more than one is found, it provides a list.

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 checks, and E to exploit. The saved environment will be loaded and used at startup, this allows you to configure various default options in the Global environment of *msfconsole*, save them, and take advantage of the in the *msfcli* interface.

The command line interface is well suited for automated exploitation and batch testing, combined with a custom payload and an intelligent scanner, it could be ruthless :)

The Web Interface

The *msfweb* interface is a functional web server that allows you launch attacks from your web browser. This interface is still very primitive, but might be useful for users working in a team environment (pen-testing, etc). The connection to the exploited host is proxied to a random listening port on the web server and the user is given a telnet protocol link to this dynamically created listener.

Be aware that this interface has no security measures in place, anyone on the network may connect to this web server or the dynamically selected proxy port. The default configuration is to listen on the loopback address only, this can be changed by passing the -a option with a value consisting of address:port. Just like the command line interface, the saved environment is loaded on startup and can affect module settings.

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.

InlineEgg Python Payloads

InlineEgg is a library for assembly programs using the Python scripting language, the more common use of this library is create advanced exploit payloads. The Framework supports InlineEgg payload through the ExternalPayload module interface; this allows transparent support if the Python scripting language is installed.

This release includes InlineEgg examples for Linux, BSD, and Windows. The Linux examples are linx86reverse_ie, linux86reverse_bind, and linux86reverse_xor. These payloads can be used in the exact same way as any other; however the contents are dynamically generated by the Python scripts found the payloads/external subdirectory. The BSD examples work in the exact same way, as long as Python is installed, they should be available.

The Windows InlineEgg example is named win32reverse_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 payload, 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

Impurity ELF Injection

Impurity was a concept developed by Alexander Cuttergo that outlined a method of executing an ELF image 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 linx86reverse_imp 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 impurity subdirectory for more information about this process. The included "shelldemo" application allows you to list, access, read, write and open file handles in the exploited process. The original mailing list post is archived online at:

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

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 Msf::Socket::Proxies Global 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.

UploadExec Payloads

Although Unix systems normally include all of the tools you need post-exploitation, Windows systems are notoriously lacking in a decent command line tool kit. The UploadExec payloads included in this release allow you to simultaneously exploit a 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.

More Information

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 Windows shellcode archive.

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.

Developers

If you would like to get involved in the development of the next version of the Framework, please contact the developers. They can be reached at: msfdev [at] metasploit.com.